Bounds on the Energy Consumption of Computational Kernels

Andrew Gearhart

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-175
http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-175.html

October 23, 2014

Acknowledgement

Research partially funded by DARPA Award Number HR0011-12-2-0016, the Center for Future Architecture Research, a member of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, and ASPIRE Lab industrial sponsors and affiliates Intel, Google, Nokia, NVIDIA, Oracle, and Samsung. Any opinions, findings, conclusions, or recommendations in this paper are solely those of the authors and does not necessarily reflect the position or the policy of the sponsors.

Bounds on the Energy Consumption of Computational Kernels

by

Andrew Scott Gearhart

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

and the Designated Emphasis

in

Computational Science and Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor James W. Demmel, Chair
Professor Katherine A. Yelick
Professor Tarek I. Zohdi

Fall 2014

Bounds on the Energy Consumption of Computational Kernels

Abstract

Bounds on the Energy Consumption of Computational Kernels

by

Andrew Scott Gearhart

Doctor of Philosophy in Computer Science
and the Designated Emphasis
in
Computational Science and Engineering

University of California, Berkeley

Professor James W. Demmel, Chair

As computing devices evolve with successive technology generations, many machines target either the mobile or high-performance computing/datacenter environments. In both of these form factors, energy consumption often represents the limiting factor on hardware and software efficiency. On mobile devices, limitations in battery technology may reduce possible hardware capability due to a tight energy budget. On the other hand, large machines such as datacenters and supercomputers have budgets directly related to energy consumption and small improvements in energy efficiency can significantly reduce operating costs. Such challenges have influenced research upon the impact of applications, operating and runtime systems upon energy consumption. Until recently, little consideration was given to the potential energy efficiency of algorithms themselves.

A dominant idea within the high-performance computing (HPC) community is that applications can be decomposed into a set of key computational problems, called *kernels*. Via automatic performance tuning and new algorithms for many kernels, researchers have successfully demonstrated performance improvements on a wide variety of machines. Motivated by the large and increasingly growing dominant cost (in time and energy) of moving data, algorithmic improvements have been attained by proving lower bounds on the data movement required to solve a computational problem, and then developing *communication-optimal* algorithms that attain these bounds.

This thesis extends previous research on communication bounds and computational kernels by presenting *bounds on the energy consumption* of a large class of algorithms. These bounds apply to sequential, distributed parallel and heterogeneous machine models and we detail methods to further extend these models to larger classes of machines. We argue that the energy consumption of computational kernels is usually predictable and can be modeled via linear models with a handful of terms. Thus, these energy models (and the accompanying bounds) may apply to many HPC applications when used in composition.

Given energy bounds, we analyze the implications of such results under additional constraints, such as an upper bound on runtime, and also suggest directions for future research that may aid future development of a hardware/software co-tuning process. Further, we present a new model of energy efficiency, Cityscape, that allows hardware designers to quickly target areas for improvement in hardware attributes. We believe that combining our bounds with other models of energy consumption may provide a useful method for such co-tuning; i.e. to enable algorithm and hardware architects to develop provably energy-optimal algorithms on customized hardware platforms.

Now this is not the end.
It is not even the beginning of the end.
But it is, perhaps, the end of the beginning.

- Sir Winston Churchill, 1942

Contents

Contents ii

List of Figures v

List of Tables vii

1 Introduction 1
- 1.1 Communication Now Dominates Performance Costs . . . 1
- 1.2 Energy Efficiency at the Algorithm Level . . . 2
- 1.3 Thesis Goals and Contributions . . . 2
- 1.4 Thesis Organization . . . 3

2 Energy Consumption and Computing 5
- 2.1 Power vs. Energy . . . 6
- 2.2 Phase-based Execution of Applications . . . 7
- 2.3 Key Consumers of Energy on Desktops and Server Nodes . . . 9
  - Energy Consumption in CMOS Logic . . . 10
  - Other Hardware Components . . . 14
- 2.4 Network Energy Consumption on Distributed Parallel Machines . . . 15

3 Machine Models for Runtime and Energy 17
- 3.1 Problems, Algorithms, and Implementations . . . 17
- 3.2 Machine Models . . . 18
  - Sequential Machine Model (S) . . . 19
  - Distributed Parallel Machine Model 1 (DP1) . . . 21
  - Model Compositions and Distributed Parallel Model 2 (DP2) . . . 21
  - Heterogeneous Machine Model (H) . . . 23
- 3.3 Problems of Particular Focus . . . 25
  - Matrix-vector multiplication . . . 26
  - Matrix-matrix Multiplication . . . 29
  - $O(n^2)$ n-body problem . . . 35
- 3.4 Model Validation . . . 37

Performance Counter Measurement . . . 37
Measuring Power and Energy . . . 40
Sequential Model (S) . . . 40
Fitting the Model via Least Squares . . . 42
Distributed Parallel Models . . . 53
Heterogeneous Model . . . 53
3.5 Parameter Estimation for Machines and Implementations and Related Work . . . 58

4 Bounds on Communication, Runtime and Energy for Specific Algorithms 61
4.1 Communication Lower Bounds for Sequential and Distributed Parallel Machines . 61
Lower Bounds on the DP Models that Include Link Contention . . . 64
4.2 Energy Lower Bounds for Specific Algorithms . . . 69
$O(n^3)$ Classical Matrix Multiplication . . . 69
Strassen and Strassen-like Matrix Multiplication . . . 72
Matrix-vector multiplication . . . 74
$O(n^2)$ n-body problem . . . 75
4.3 Bounds on Heterogeneous Machines . . . 78
Input/Output Dominated Lower Bounds . . . 80
Loomis-Whitney Dominated Lower Bound . . . 82
4.4 Optimal Heterogeneous Algorithms . . . 84
Heterogeneous Matrix-Vector Multiplication . . . 84
Heterogeneous $O(n^3)$ Matrix-Matrix Multiplication . . . 86

5 Bounds on Communication, Runtime and Energy for Programs that Access Arrays 90
5.1 Bounds on Programs that Reference Arrays . . . 90
Sequential Model . . . 95
Distributed Parallel Model 1 . . . 95
Distributed Parallel Model 2 . . . 96
Heterogeneous Model . . . 97
Example: Energy Lower Bound for Matrix-matrix Multiplication . . . 98
5.2 Perfect Strong Scaling in the Distributed Machine Model . . . 99

6 Applications of Bounds on Specific Machine Models 105
6.1 Overview . . . 105
6.2 Example Machines for Analysis . . . 108
6.3 Classical $O(n^3)$ Matrix-matrix Multiplication . . . 111
6.4 $O(n^2)$ n-body problem . . . 117
6.5 Programs that access arrays with subsets of the iteration variables . . . 122

7 Implications for Hardware Designs 130
7.1 Introduction . . . 130
7.2 Cityscape Model of Energy Efficiency . . . 132

7.3 Financial cost/Job (C_{job}) 136
7.4 Further Directions 141

8 Conclusions **147**

Bibliography **150**

List of Figures

2.1 Tradeoff between minimizing energy or power 6
2.2 Typical Scientific Code Power Trace 8
2.3 Power phases of matmul benchmark on Sandy Bridge-EP 8
2.4 Power phases of heterogeneous matmul on Sandy Bridge-EP and Nvidia K20 9
2.5 CMOS inverter with n-type (nMOS) and p-type (pMOS) transistors indicated 11
2.6 Input/Output wattage curves for a Dell DH350E-S0 power supply for 2U servers [1] . . 15

3.1 Relationship between hardware, algorithm and implementation 18
3.2 Serial (S) and Distributed Parallel (DP) machine models 19
3.3 Composition of Sequential (S) and Distributed Parallel (DP) Machine Models 22
3.4 Heterogeneous machine model 24
3.5 Compressed Sparse Row (CSR) storage format 28
3.6 Two-dimensional block cyclic distribution of a matrix on a 2-by-2 processor grid . . . 31
3.7 Processor grids for 3D and 2.5D matrix-matrix multiplication [126] 32
3.8 Breadth-First or Depth-First traversals of recursion tree [98] 35
3.9 Data layouts for 1D, 1.5D and 2D n-body algorithms 36
3.10 $O(n^2)$ n-body algorithm with and without a cutoff distance 37
3.11 Counting cache misses during array copy on Sandy Bridge-EP 39
3.12 Inaccurate floating point operation counts on Sandy Bridge-EP 39
3.13 Typical wall power sample windows for several sparse matrix-vector multiplication problems 41
3.14 Sandy Bridge-EP: Flop/Word ratios for double-precision sparse matrix-vector multiplication (DSPMV) 45
3.15 Sandy Bridge-EP: Modeled (no row scaling) double-precision matrix-matrix multiplication (DGEMM) 49
3.16 Sandy Bridge-EP: Modeled (no row scaling) double-precision dense matrix-vector multiplication (DGEMV) 50
3.17 Sandy Bridge-EP: Modeled (no row scaling) double-precision sparse matrix-vector multiplication (DSPMV) 51
3.18 Heterogeneous machine for validation 53
3.19 Runtime impact of scaling either Host or GPU SGEMM size 54
3.20 Runtime impact of scaling either Host or GPU SGEMV size 56

4.1 Communication bounds for Strassen's algorithm on d-dimensional tori. The lower plot is log-log, while the upper is linear on the y-axis. Horizontal lines in the lower plot correspond to perfect strong scaling. 67
4.2 Example of heterogeneous matrix-vector data partitioning with 4 processors 85
4.3 Heterogeneous matrix-matrix computation example execution on 4 processors 89

5.1 Relationship between the per-processor and contention communication lower bounds, with labels on each region indicating lower bound dominance. F and M are constants. 93

6.1 Energy costs as node count and memory are scaled 106
6.2 Effect of constraints on energy efficiency 107
6.3 2.5D $O(n^3)$ Matrix-matrix Multiplication: Effect of replicating memory on energy efficiency 117
6.4 CA $O(n^2)$ n-body: Effect of replicating memory on energy efficiency 123
6.5 3-Body Problem: Effect of replicating memory on energy efficiency 129

7.1 Example Cityscape Model for $O(n^3)$ Matrix-matrix multiplication 135
7.2 C_{job} with various parameter sets for Algorithm 11 142
7.3 Sequential Machine with 3 levels of fast memory 143

List of Tables

3.1 Computational motifs as described in [9] . . . 26
3.2 Measuring Data Movement on Sandy Bridge-EP . . . 38
3.3 Sandy Bridge-EP: Arithmetic intensity (flop/word) for naive matrix-matrix multiplication 44
3.4 Sandy Bridge-EP: Average runtime and energy % error (no row scaling) . . . 46
3.5 Xeon 7560: Average runtime and energy % error (no row scaling) . . . 46
3.6 Sandy Bridge-EP: Average runtime and energy % error (1/F row scaling) . . . 46
3.7 Sandy Bridge-EP: Fitted sequential machine parameters without row scaling . . . 47
3.8 Sandy Bridge-EP: Modeled vs. Measured Runtime Throughputs . . . 47
3.9 Xeon 7560: Fitted sequential machine parameters without row scaling . . . 48
3.10 Xeon 7560: Modeled vs. Measured Runtime Throughputs . . . 48
3.11 Sandy Bridge-EP: Average runtime and energy % error when non-dominant terms are dropped from model (with row scaling) . . . 52
3.12 Sandy Bridge-EP: Fitted sequential machine parameters when non-dominant terms are dropped from model (with row scaling) . . . 52
3.13 Sandy Bridge-EP: Modeled vs. measured runtime throughputs when non-dominant terms are dropped from model (with row scaling) . . . 52
3.14 Fitted heterogeneous machine parameters without row scaling . . . 57

4.1 Per-processor bounds (W_{DP}) ([84, 15, 21, 20, 61]) vs. the new contention bounds (W_{DP}^{link}) on a d-dimensional torus for classical linear algebra, fast matrix multiplication, and the $O(n^2)$ n-body problem. . . . 66
4.2 Torus dimensions so that communication cost is either always contention bound ($d \leq D_1$) or never contention bound ($d \geq D_2$) for a selection of matrix multiplication algorithms. The assertions regarding the last three algorithms are under some technical assumptions / conjecture, see [20]. . . . 68

5.1 Per-processor bounds (W_{DP}^{HBL}) ([45]) vs. the new contention bounds ($W_{DP}^{\text{link}HBL}$) on a d-dimensional torus for programs that reference arrays. . . . 92

6.1 Description of Xeon 2650-based and future distributed parallel machines . . . 109
6.2 Parameters derived from machine descriptions . . . 110

7.1 Intel Xeon-based distributed parallel baseline machine . . . 133

7.2 Energy efficiency terms for several problems (all are measured in the same units, namely joules/flop) . . . 134
7.3 Additional model values . . . 138
7.4 Processor parameters . . . 138
7.5 Network adaptor and cable parameters . . . 139
7.6 DRAM parameters . . . 140
7.7 Parameters for minimal C_{job} . . . 141

Acknowledgments

Thanks to my advisor, Jim Demmel, for his support and mentorship over the past six years. Jim's experience and creativity have been extremely helpful in the development of my own research process and I look forward to his continued insight into mathematical problems in the years to come. Also thanks to my co-adviser, Tarek Zohdi. Tarek has been an excellent technical reference, as well as a point of stability and perspective regarding the graduate school experience. Example: "Thesis? Once you have a topic, just start writing something up...the process of producing tends to fill in the rest". Without Tarek's support, I would have probably gone a bit squirrely several years ago. Thanks Jim and Tarek!

Many thanks to Grey Ballard and Oded Schwartz for their insight and support on several projects. Both Grey and Oded are brilliant, patient researchers and always willing to carefully walk through a technical challenge. I wish them the best of luck in their burgeoning academic careers, and I'm excited to hear of great things from their continuing research. Thanks also to other Parlab/ASPIRE students that have been extremely helpful: Michael Anderson, Vasily Volkov, Benjamin Lipshitz, among many others.

I'd also like to extend my thanks to Kathy Yelick and Sam Williams of UC Berkeley and Lawrence Berkeley National Laboratory for their technical insight and support. Kathy is an amazing resource regarding the state of high-performance computing technology and I've yet to find a computer architecture that Sam is unable to tune. Also thanks to Harsha Simhadri for his insight into different theoretical approaches to modeling machine communication.

As this thesis has required a bit of hardware knowledge, I'd like to thank David Sheffield, Yunsup Lee, Andrew Waterman, Scott Beamer, Brian Zimmer and Ben Keller for putting up with my novice questions regarding various architectural and device-level concepts. David, Brian and Ben have also been willing to spend a significant amount of time making my life a bit easier, from tweaking build infrastructures to soldering leads.

I've been continually amazed by the quality of the support staff in the UC Berkeley EECS department and Parlab/ASPIRE. Kostadin Ilov has spent many hours moving, rebooting, reinstalling and hacking various machines on my behalf, all with a joke and a smile. Thanks also to Roxana Infante and Tamille Johnson for tirelessly handling my various "crises": from ordering parts, to missing registration deadlines, to supplying Advil. The CS graduate student advisors, La Shana Porlaris and Xuan Quach, also deserve thanks for their support in wading through the morass of paperwork required to update schedules, add a designated emphasis, or change a grade.

My friends at Intel Corporation have been exceptional resources and mentors as I've developed as a student and researcher. Special thanks to Mark Rowland and Gans Srinivasa for their mentorship, Ian Steiner for his prowess of all things IA and Hugh Caffey for his insight into Intel's performance counter infrastructure. Also, thanks to the other Intel employees that helped make my two summers interning in Hillsboro productive and insightful.

Thanks to my parents, sister Rachel, and other family members that have supported me throughout classes, exams, prelims, quals, etc. Special thanks to my boyfriend, Jade Donigan, for also putting up with my regular neurotic moments over the last several years. Finally, thanks to all the

friends and colleagues that have provided support and wisdom over the years. I'm flattered and humbled to have had the chance to interact with such amazing people during my time at Berkeley.

Research partially funded by DARPA Award Number HR0011-12-2-0016, the Center for Future Architecture Research, a member of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, and ASPIRE Lab industrial sponsors and affiliates Intel, Google, Nokia, NVIDIA, Oracle, and Samsung. Any opinions, findings, conclusions, or recommendations in this paper are solely those of the authors and does not necessarily reflect the position or the policy of the sponsors.

Chapter 1

Introduction

1.1 Communication Now Dominates Performance Costs

Historically, algorithm developers have focused on asymptotically minimizing the number of floating point operations (flops) required to execute a computational problem. If floating point costs dominate the computation, this is a viable approach to increase performance. However, many current computational kernels only attain a small fraction of the peak floating point capability of the machine and are constrained by the bandwidth and latency characteristics of the memory subsystem [9]. We refer to the traffic between levels of the memory hierarchy as communication, and note that reducing communication traffic allows for an algorithm to potentially attain a higher fraction of the machine's peak floating point rate (see the Roofline model [148] for more details). Unfortunately, the performance gap between off-chip communication bandwidth and latency currently exceeds the cost of a floating point operation by an order of magnitude or more, and the disparity is increasing with time [121]. This communication bottleneck exists across the entire spectrum of computing, from datacenters to embedded devices, and suggests a need for new classes of algorithms that focus on minimizing communication costs (which may also include synchronizations in parallel codes), as opposed to floating point operations.

Indeed, a rapidly-evolving field of algorithm research addresses the problem of communication costs. Prior to the development of optimal algorithms, attainable lower bounds on the amount of communication required to solve a computational problem must be considered. Starting with foundational work by Hong, Kung and others, there are known lower bounds on the amount of communication required for specific algorithms [86, 84]. More recently, Ballard et al.[15] presented generalized communication lower bounds for many problems in dense and sparse linear algebra (as well as a few graph problems). These communication lower bounds were then expanded to Strassen and Strassen-like algorithms [20, 98] as well as an even larger class of problems [45] via a generalization of the argument presented within [15]. From these lower bounds, it was discovered that many existing algorithms were not communication-optimal and many new optimal algorithms were created. For example, new communication-optimal algorithms have been developed for matrix-matrix multiplication [126], LU [126], QR [15] and Cholesky [14] factorizations,

matrix-powers computations [76, 111], tensor contractions [127] and the $O(n^2)$ n-body problem [61], among others. This communication optimality may come at the cost of additional flops, as in the case of certain Krylov subspace computations [76], or via the use of additional memory as in 2.5D matrix-matrix multiplication [126]. Implementations of many of these algorithms have been able to attain speedups on various machines. An introduction and overview of these results for linear algebra can be found in [13].

1.2 Energy Efficiency at the Algorithm Level

While the bandwidth and latency constraints may result in decreased floating point throughput (measured in Gflop/s), off-socket communication also costs a significant amount of energy relative to the energy cost of a floating point operation. Akin to the runtime gap between flops and communication, the energy gap between performing a floating point operation and moving a word of memory off-socket is at least an order of magnitude [43]. This issue has been addressed by device engineers and hardware architects extensively (see [89] for a review of architectural techniques to reduce energy consumption), but until recently has not received significant interest from the algorithm and application development community. Previous research into energy-efficient algorithms was mostly focused on maximizing battery life within distributed sensor networks [153, 42, 154]. More recently, researchers have begun efforts to extend the throughput roofline model of [148] to energy [44] and develop benchmarks to characterize the energy/operations characteristics of machines running various workloads [43, 28, 91]. Additional research has attempted to develop approaches to co-tuning new hardware and software implementations for increased efficiency [110, 49, 136, 119], and such research is gaining traction in both industry and government.

1.3 Thesis Goals and Contributions

This thesis adds to the growing body of work on energy efficiency by considering the impact of algorithmic changes to runtime and energy consumption. This is accomplished by combining communication bounds with models of machine behavior to construct lower bounds on the runtime and energy required to compute a problem of a given size. Such lower bounds are then used to analyze the impact of improvements in processor hardware parameters and the amount of extra memory utilized by certain algorithms. We also present new communication lower bounds for a heterogeneous machine model, and also describe communication-optimal algorithms for dense matrix-matrix and matrix-vector multiplication within heterogeneous processors.

In this thesis, we explicitly focus on desktop and server platforms, as these are easier to measure with low-cost equipment and often have relatively homogeneous hardware designs when compared to handheld and embedded devices. Also, these classes of machines tend to have a relatively small and consistent set of key energy-consuming components and lend themselves to theoretical analysis. We will argue that on the timescale of seconds, the runtime and energy characteristics of these classes of machines can be reasonably described via simple linear models that consider

processor, memory and static costs. Furthermore, we argue that most scientific applications can be decomposed into regions of constant arithmetic intensity and power that correspond to the execution of computational kernels. While not within the scope this this document, we believe that this approach is applicable to other classes of machines, such as mobile devices, with different model constructions.

The key contributions of this thesis are as follows:

- We present empirical evidence that at the application level and within a given level of the memory hierarchy, the energy consumption of computational kernels is predicable with linear models that comprise a handful of terms (Chapter 3).
- Via such models, we present non-trivial lower bounds on the runtime and energy required to execute algorithms on sequential and distributed parallel machine models. These bounds build upon recent developments in communication lower bounds and communication-optimal algorithms. In particular, we focus upon the problems of dense classical $O(n^3)$ and Strassen matrix-matrix multiplication, dense and sparse matrix-vector multiplication, and the $O(n^2)$ n-body problem (Chapter 4).
- We present runtime and energy bounds on a model of heterogeneous processing, as well as new algorithms for heterogeneous dense matrix-matrix and matrix-vector multiplication that have energy and runtime costs that are provably optimal (Chapter 4).
- We extend energy bounds for individual algorithms to a large class of programs that access arrays via affine expressions of the iteration variables (Chapter 5).
- We describe an algorithm that is provably optimal for a subset of such problems on distributed parallel machines, and show that a region of perfect strong scaling with constant energy exists as the algorithm utilized additional memory to reduce communication volume (Chapter 5).
- We use these bounds to explore the potential runtime and energy efficiency tradeoffs that occur by using additional memory to reduce communication in the presence of various constraints, such as runtime or energy limits (Chapter 6).
- Via our new Cityscape model, we propose a method to generate constraints on hardware parameters so as to attain a target level of energy efficiency for a large class of algorithms (Chapter 7).
- We show that energy and runtime bounds can be used to approximate the financial cost per job, and also optimize machine hardware according to that metric (Chapter 7).

1.4 Thesis Organization

The content of this thesis is divided into a number of chapters. In Chapter 2, we provide an overview of power and energy consumption on modern desktop and server platforms, and argue

that the energy consumption of such machines is due to the activity of a small number of key hardware components. We also argue that certain characteristics of many scientific applications aid an attempt to model energy consumption and runtime via linear models to a level of granularity sufficient for algorithm developers.

Chapter 3 describes the computational problems specifically targeted for analysis within this work, and also presents models for the runtime and energy of sequential, distributed parallel, and heterogeneous machines. Chapter 3 then describes the use of performance counters to measure memory traffic and includes evidence that linear models for runtime and energy are adequate for algorithm and hardware developers to gain a high-level idea of the runtime and energy efficiency of an algorithm. Chapter 4 presents energy bounds for specific algorithms on sequential, distributed parallel, and heterogeneous classes of machines, and Chapter 5 extends these algorithm-specific results to generalized energy bounds on a much larger class of computational problems. Chapter 5 also shows that a communication-optimal algorithm for a subset of this larger class of problems is able to halve runtime by doubling the number of processors on a fixed problem size. This perfect strong scaling in runtime is attained with constant energy by utilizing additional memory to offset communication volume.

In Chapter 6, we apply the energy and runtime bounds of Chapters 4 and 5 to a set of questions that are potentially useful to algorithm developers. In particular, we demonstrate how the use of additional memory to offset communication affects energy efficiency in the presence of various constraints on total runtime and energy, among others. Chapter 7 extends this concept by describing ranges of hardware parameters that attain a target level of energy efficiency or the optimal financial cost per job. Finally, in Chapter 8 we conclude and discuss directions for potential future research. Code, data and analysis for the results presented within this dissertation can be found at `https://github.com/agearh/dissertation.git`.

Chapter 2

Energy Consumption and Computing

As mentioned previously, this document utilizes machine models to bound the amount of energy consumed during the execution of an algorithm. These bounds are then used to provide insights into algorithm performance and future hardware design. To assist in doing this, we make four assumptions about the behavior of scientific applications, which will be justified via a combination of original and existing research:

- **Assumption 1**: Scientific applications can be divided into regions of execution that correspond to key computational kernels of the application. We refer to these regions of execution as *phases* in this document.
- **Assumption 2**: At the timescale of seconds, power consumption is relatively constant during execution of a phase.
- **Assumption 3**: The primary energy consumers on desktop and server nodes are processor execution units, static random-access memory (SRAM) caches and dynamic random-access memory (DRAM) when considering problem sizes that fit within main memory.
- **Assumption 4**: When discussing distributed machines, the energy of current internode networks may not scale with utilization and is thus another static term. We will derive a second distributed parallel machine model to reflect this situation in Chapter 3.

In this chapter, we argue for the applicability of these assumptions to many scientific applications by both citing previous research and presenting our own evidence. We begin by discussing tradeoffs between power and energy (Section 2.1) and then argue that many scientific applications demonstrate phase-based power behavior that corresponds to the execution of computational kernels (Section 2.2). Finally, we discuss key consumers of energy on desktops and server nodes (Section 2.3) as well as distributed parallel machines (Section 2.4).

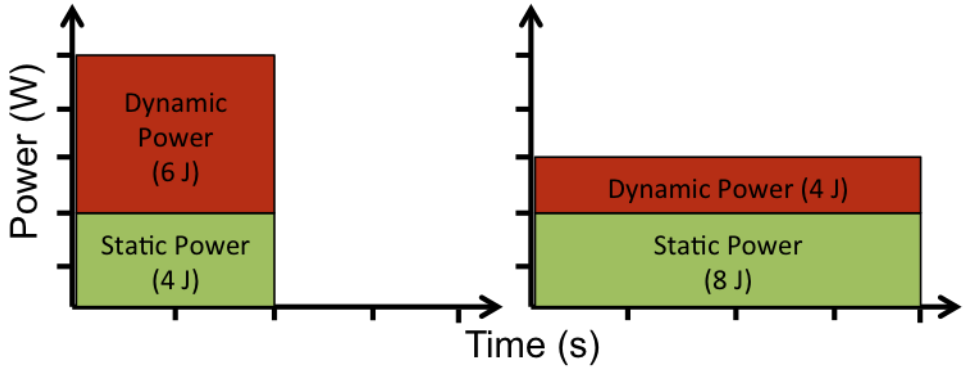


Figure 2.1: Tradeoff between minimizing energy or power

2.1 Power vs. Energy

Before discussing the assumptions presented earlier, a brief discussion of the relationship between power and energy is warranted. First and perhaps foremost in this discussion, digital circuits can be considered as devices that convert electrical energy into an application-specific output with some loss which is represented as dissipated heat. The issues of reducing energy consumption and power within computing devices are clearly linked as energy is the integral of power with respect to time, often represented simply as $E = HT$ where H is the average power over the time period T. In this document, we primarily limit our discussion (unless otherwise noted) to the problem of minimizing overall energy consumption during the execution of an algorithm. This is a useful problem for both handheld devices as well as datacenters, where the goals are to extend battery life and reduce operating costs, respectively. As a common example, the most energy-efficient means of computation may be to use a large amount of power to finish quickly and go to idle. Such an idea is called "race-to-halt", and will be discussed in greater detail later in this document. On the other hand, minimizing power consumption may involve solving the same problem at the slowest rate possible. In this situation, the power is minimized while the overall energy required to complete the computation may be suboptimal. Figure 2.1 illustrates this situation as two runs of the same problem at different processor performance settings. Note that in Figure 2.1, the first run of the algorithm consumes 10J of energy and 5W of power for two seconds. The other run (perhaps at a lower clock frequency and processor voltage) consumes 12J of energy and 3W of power over four seconds. From the standpoint of minimizing energy, the first run is superior as the workload-dependent energy consumption (the *dynamic energy*) dominates the workload-independent energy consumption (*static energy*). We will expand upon the concepts of static and dynamic energy later in this chapter, and will use this distinction to define models of energy in Chapter 3.

2.2 Phase-based Execution of Applications

In this thesis, we assume that most scientific applications are composed of a set of key computational kernels that are required for the problem to be solved in a specific order along a critical path. Each kernel may then be optimized independently of the other kernels in the application. This application structure roughly corresponds to the Bulk-Synchronous Parallel (BSP) model of parallel execution first proposed by Valiant [139], but is more general as no assumptions are made about hardware or synchronization between phases. We believe that by improving the efficiency (both in throughput and energy) of a small set of application kernels, such benefits can be applied to a large number of scientific codes. This view of scientific computing echoes the conclusions of the Berkeley View on Parallelism [9] which surveyed a large number of scientific application domains.

This work focuses on the energy and runtime costs of application kernels that execute at time scales that range from tenths of a second to minutes. At this coarse level of granularity (many thousands of clock cycles), we are not concerned with the ability to estimate or measure power to an extremely high level of fidelity (unlike chip designers or architects). At this time scale, we observe that many applications demonstrate *phase-based* power consumption behavior. By this, we mean that the power for an algorithm tends to stay relatively constant during different portions of its execution timeline and note that power phases typically occur during the execution of a kernel within the application. In Figure 2.2, we depict the wall power[1] trace of a machine running a generic application with three phases of execution. Note that Figure 2.2 shows a common, constant level of static power that does not depend on application behavior. The red portions of the figure depict the dynamic energy of the algorithm in three phases, each of which represents a region of constant wall power during its runtime. Later in this chapter, we will also show evidence that the key consumers of energy on desktops and server nodes are processor execution units, caches, and main memory. In Chapter 3 we will present empirical evidence suggesting that multiplying this phase-constant power by the runtime of a phase is an accurate way of approximating the energy consumption of a kernel for many applications.

To support the diagram of a generic wall power trace shown in Figure 2.2, we recorded the power of a dual-socket Sandy Bridge-EP server running a benchmark that allocates and initializes three square matrices prior to running the parallel double-precision dense matrix-matrix multiplication (DGEMM) implementation found in Intel's Math Kernel Library (MKL) version 11.1. This wall power trace is shown in Figure 2.3 and has a time axis that ranges to approximately 330 seconds. In the power trace, we see the benchmark executed twice for different matrix sizes selected to fit within the machine's main memory (thus, little disk access occurs and the problems are unable to fit within last level cache). To correlate phase and kernel behavior, we annotated the initialization and computation phases of the benchmarks with timestamps that were then aligned with the sample times from a meter that measures wall power. For clarity, the Linux *sleep* command was used to impose 10 second intervals between the execution of each of the different-sizes problems.

In Figure 2.3, we note that the current static power of the machine is approximately 140W and

[1] Wall power is the total power consumed by the machine, including power supply inefficiency.

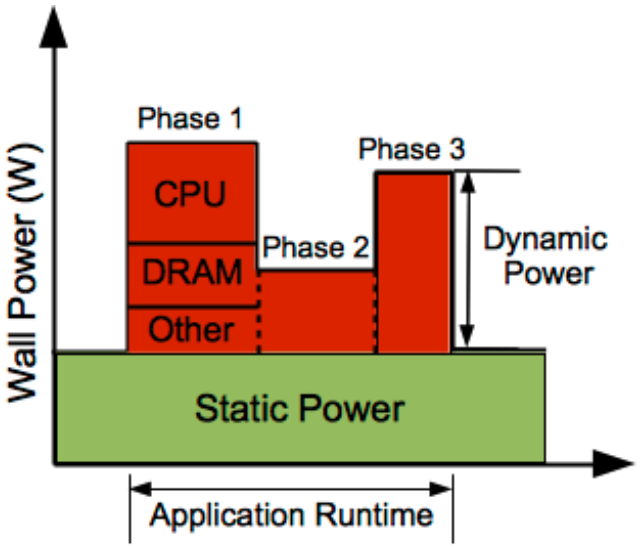
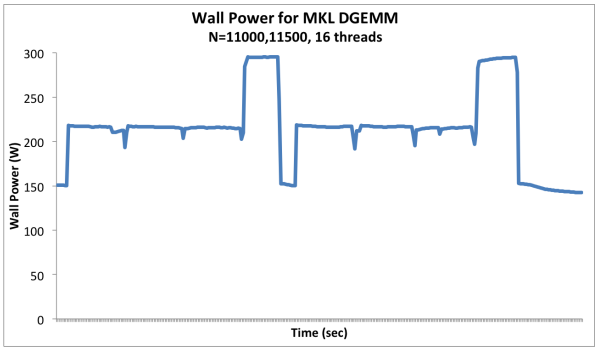
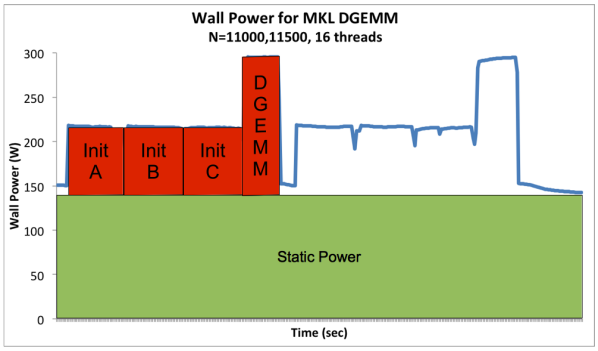


Figure 2.2: Typical Scientific Code Power Trace

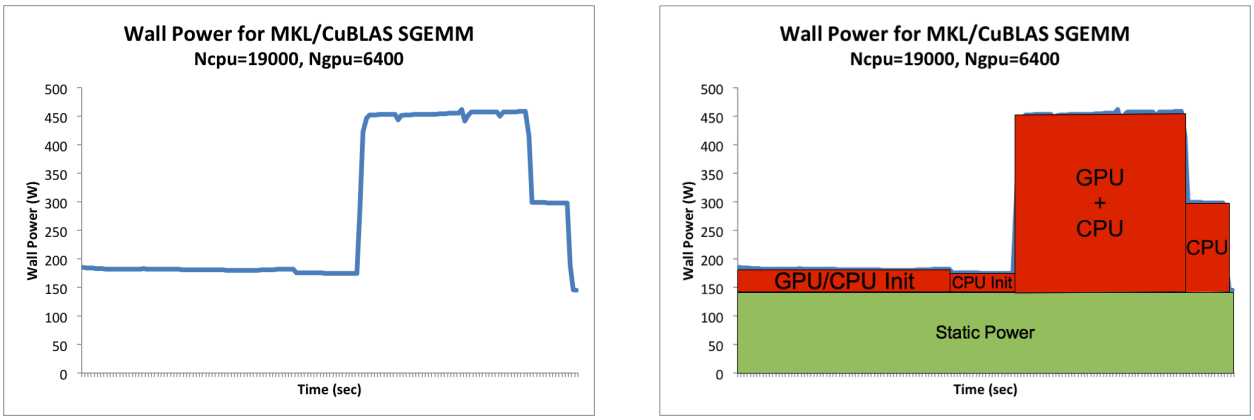


(a) Wall power for two runs of DGEMM benchmark



(b) Wall power with phase annotations

Figure 2.3: Power phases of matmul benchmark on Sandy Bridge-EP



(a) Wall power for a run of heterogenous SGEMM benchmark

(b) Wall power with phase annotations

Figure 2.4: Power phases of heterogeneous matmul on Sandy Bridge-EP and Nvidia K20

that the machine while running *sleep* consumes approximately 150W. Between these idle periods, four power phases can be clearly seen: 3 of an identical length between approximately 210-220W, and one phase between approximately 290-300W. Via the timestamps, we observe that these four power phases correspond exactly to the three parallel initialization phases of the benchmark followed by the actual matrix-matrix multiplication operation itself. This result supports the structure of the generic wall power trace in Figure 2.2. Further, we also note that power appears to be relatively constant across problem sizes within a given level of memory.

This concept of power phases can be extended further when multiple heterogeneous devices are executing during an application run. In Figure 2.4, we see an example of an annotated wall trace of a benchmark that performs single-precision dense matrix-matrix multiplications (SGEMMs) on both a host machine (CPU) and a graphics processing unit (GPU). This wall power trace was generated by running Algorithm 7 of Section 3.4 with $gpuInner = 50$, $gpuOuter = 5$, $hostInner = 2$ and square problem sizes of $n_{cpu} = 19000$ on the host and $n_{gnu} = 6400$ on the GPU. Note that with two devices executing, the constant power phase of the GPU overlaps with the power phase of the host/CPU. In Chapter 3, we discuss the implications for modeling energy consumption with multiple devices.

2.3 Key Consumers of Energy on Desktops and Server Nodes

As mentioned earlier in the chapter, we argue that the key hardware components that dominate dynamic energy consumption on desktops and servers are:

- Processor execution units (eg. floating point units, branch predictors, or reorder buffers)
- Static random-access memory (SRAM)

- Dynamic random-access memory (DRAM)

The dominance of these few components is supported by research that directly instruments hardware components for power, and then considers energy consumption while running benchmarks. Other researchers have devoted significant effort toward direct measurement of desktop and server components via specialized instrumentation devices, in particular PowerPack [66] and PowerMon [26]. Both projects involve inserting current sense resistors (CSRs) into wires between the power supply and machine devices to measure current (and thus power, as wires are of a known voltage and power $=$ current $*$ voltage). Thus, PowerPack and PowerMon avoid measurement of power supply losses while creating the new problem of differentiating component power from various supply wires (e.g. the ATX power supply specification defines a 20-wire input from the power supply to motherboard on desktop machines [10]). Both PowerPack and PowerMon address this issue via a set of benchmarks that attempt to isolate component power. PowerPack, originating from Virginia Tech, uses a commodity National Instruments data acquisition (NI-DAQ) device to measure voltage drops across sense resistors. This device has the advantage of being widely available with a stable software stack, but has a high per-unit cost. To address this limitation, the PowerMon project utilizes a custom monitoring breadboard to collect data samples. The design for this board is freely available, and it was designed to allow for cheap instrumentation of distributed systems. PowerPack [66] and PowerMon [26] both argue for the dominance of processor and DRAM energies on desktop and server nodes, and this is further supported by the data reported in studies that use PowerPack [101, 145, 103] for related research. The dominance of processors and main memory (DRAM) over components such as disks is also argued by Brown and Reams [37]. The authors of [27] take this argument further, and argue for an increasing fraction of total distributed machine energy consumed by node DRAM with successive reductions in device feature sizes.

Processor execution units, SRAMs and DRAMs each share a common underlying technology, that of complementary metal-oxide semiconductor (CMOS) logic. We have argued that research supports the dominance of these components with regard to dynamic energy, but also acknowledge that the static power of other machine components (such as power supply inefficiency, motherboards, and hard drives) may contribute a large portion of overall static energy consumption. We consider these devices later in this chapter. In the next section, we discuss dominant sources of energy consumption in CMOS and provide references for readers interested in further details.

Energy Consumption in CMOS Logic

Manufacturers utilize CMOS technology to implement digital circuits due to additional noise resistance and reduced energy consumption over other logic classes such as N-type metal-oxide-semiconductor (NMOS) or Transistor-transistor logic (TTL) (see classic text by Glasser and Dubberpuhl [68] for more details). CMOS utilizes paired p-type and n-type MOS transistors (pMOS and nMOS transistors, respectively) in a manner such that current does not flow from supply (V_{dd}) to ground (V_{ss}) once the output signal has stabilized. pMOS and nMOS transistors are both examples of field effect transistors (FETs), and as such can be considered to have 4 terminals: source,

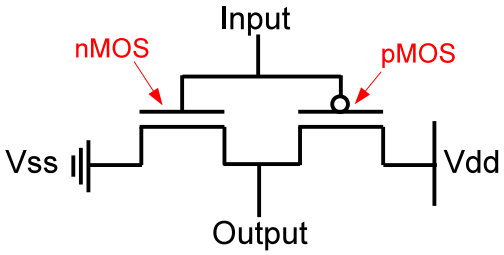


Figure 2.5: CMOS inverter with n-type (nMOS) and p-type (pMOS) transistors indicated

drain, gate and body. In this discussion, we ignore the body terminal as it is typically connected to the source.

In pMOS transistors, current flows from source to drain when there is a negative potential difference between gate and source. Inversely, current flows within nMOS transistors when there exists a positive potential difference between source and drain. Charge begins to flow from source to drain once the potential difference reaches a specific threshold voltage (V_{th}). CMOS logic builds logical gates, which implement simple binary functions from paired nMOS and pMOS transistors. As a simple example of a common CMOS gate, an inverter circuit is shown in Figure 2.5.

Figure 2.5 shows an nMOS and a pMOS transistor. When the input signal is HIGH, the pMOS is OFF and the nMOS is ON. This allows the nMOS transistor to pull down the output signal toward V_{ss}. On the other hand, if the input signal is LOW, the nMOS is OFF and the pMOS pulls the output up toward V_{dd}.

The energy consumption of CMOS circuits, E_{logic}, can be defined as the summation of three classes of terms: dynamic gate energy, static gate leakage, and interconnect energy. Dynamically, E_{logic} is composed of two terms: the switching energy E_{switch} and the short energy E_{short}. Static energy consumption can be approximated via a single term: leakage, or E_{leak}, and we express interconnect energies as a single term (E_{wire}). We express E_{logic} as the sum of these terms in Equation (2.1).

$$E_{logic} = E_{switch} + E_{short} + E_{leak} + E_{wire} \tag{2.1}$$

Dynamic Transistor Energy ($E_{switch} + E_{short}$) Because transistors can be regarded as parallel-plate capacitors, a state transition charges or discharges electrical energy. This process is one of the key components of energy consumption within CMOS circuits, and is represented by E_{switch} in Equation 2.1. Due to its relationship with capacitor charge/discharge cycles, E_{switch} can be represented via a capacitance expression

$$E_{switch} = \sum_{i=1}^{N_g} n_i c_i V_{dd}^2 \tag{2.2}$$

where N_g is the number of gates, n_i is the number of times gate i toggles during computation, c_i and V_{dd} describe the capacitance of gate i and the circuit supply voltage, respectively [69]. Note that this discussion treats gates in the same manner as transistors. We do this as gates are collections of transistors, and at a coarse level are larger capacitors themselves. From Equation (2.2), we see that reducing V_{dd} has a quadratic impact upon energy consumption. Furthermore, capacitance c_i is proportional to the size of the gate and can be represented by

$$c_i = \hat{c}_i W_i L_i \tag{2.3}$$

where $\hat{c}_i$ is the capacitance/area of the gate oxide, W_i is the gate width and L_i is the gate length.

In addition to E_{switch}, dynamic energy in CMOS logic is also consumed by the momentary connection between V_{ss} and V_{dd} as the paired nMOS and pMOS transistors shift between states. This is called the short-circuit current, and is represented by E_{short} in Equation 2.1. According to Veendrick [141], E_{short} can be minimized (to $\leq 20\%$ of E_{logic}) by matching input and output rise and fall times. Further, as noted in [114], decreasing the ratio of V_{th} to V_{dd} can also result in a 20% contribution of E_{short} to E_{logic}. As E_{short} does not directly impact the following analysis and is primarily of concern to logic designers, we do not discuss its contribution further.

Leakage Energy (E_{leak}) The static power dissipation, or leakage, of CMOS gates is represented by the E_{leak} term of Equation 2.1. While E_{leak} can be partitioned into at least eight different components [90], we will be primarily concerned with sub-threshold leakage (or weak inversion leakage). Sub-threshold currents often dominate amongst leakage components [90, 113]. Therefore, in this work the term "leakage" refers to the sub-threshold component of the overall leakage current of a device. For a more detailed analyses of leakage, we refer readers to [113] and [117].

In an ideal world, the dynamic switching component (E_{switch}) of a CMOS circuit would be the only consumer of energy. For many years, reductions in V_{dd} were combined with feature size reductions to maintain a relatively constant power density and increased clock frequency (which translates to greater potential instruction throughout). Called "Dennard Scaling" after Robert Dennard's observation in 1974 [57], this practice is no longer utilized as reductions in V_{dd} require corresponding reductions in V_{th} to maintain clock frequency increases via reductions in circuit delay[69]. To see how this has become problematic, we must first define the sub-threshold current, I_{leak}. According to Helms, Schmidt and Nebel [72], this current is

$$I_{leak} = KV_{th}^2(W/L)e^{(V_{gs}-V_{th}/(nV_T)}(1 - e^{-V_{ds}/V_T})$$

where n and K are technology parameters, W and L and the width and length of the gate, V_{ds} is the drain-source voltage and V_{gs} is the gate-source voltage. V_T is the thermal voltage, which is proportional to temperature. Note that due to the exponential relationship between threshold voltage and leakage current, a small decrease in V_{th} results in large increase in I_{leak}. It is this relationship that has effectively eliminated Dennard scaling, and contributed to the development of multicore processors that substitute increases in clock frequency with parallelism, essentially pushing the onus of performance onto algorithm, software and compiler engineers.

At the level of an entire CMOS circuit, the total leakage energy E_{leak} is the sum of each gate's leakage energy

$$E_{leak} = \sum_{i=1}^{N_g} I_{leak_i} V_{dd} T_c$$

where I_{leak_i} is the leakage current of gate i and T_c is the cycle time. In the energy models we define in Chapter 3, E_{leak} is merged into a single term with other static energy sources, such as disks. More-detailed models could be used to explicitly expose this term for analysis (see Section 7.4 for ideas on how to construct such models).

Interconnect Energy (E_{wire}) Similar to gate energy, interconnects consume energy via two components: switching energies and leakage. That is,

$$E_{wire} = E_{wireSwitch} + E_{wireLeak}.$$

The expression for switching energy of wires, $E_{wireSwitch}$, can be represented as a capacitance expression in an analogous manner to E_{switch}. Like E_{switch}, $E_{wireSwitch}$ is proportional to wire size and and quadratically dependent on voltage. On the other hand, wire static losses are inversely proportional to wire size as $E_{wireLeak}$ is dominated by resistance losses.

In CMOS circuits, pMOS and nMOS transistors are connected to form gates, gates are connected to form circuit blocks, and blocks are connected to build larger components, such as floating point units, or caches. These larger components are then linked to build processors and systems. As blocks are combined into larger components and structures, interconnect lengths increase, but the overall number of such wires decreases. The relationship between wire length and frequency can be approximated via the power-law relation described by Rent's rule [95] and the work of Donath [59, 60]. Furthermore, Sylvestor and Keutzer [135] observed that interconnects occur in two classes: local wires that scale with technology generations, and global wires that do not. This relationship between wire and technology scaling was further described in the seminal work by Ho, Mai and Horowitz [75]. Local wires can be modeled effectively via Rent's rule, but global interconnects may use different approaches that are dependent on implementation [100]. Global wires (typically clock trees or buses) present a timing bottleneck (as delay increases quadratically with wire length), which can be mitigated via such techniques as larger wires or the use of inverter chains as repeaters, but such techniques trade delay for increased dynamic and static wire energy consumption [8].

As they do not scale with process technology, global interconnects represent a significant and probably increasing fraction of system energy. Nir et al.[105] note that processor interconnect power is approximately 50% of total dynamic power, with this half split evenly between local and global wires on a testbed processor. In 2002, Basu et al.[25] observed that the off-chip bus of an embedded processor consumed 9.8-23.2% of system power, and argue that energy could be saved via fewer wire activations and then reducing the number of bit transitions within these active wires. This idea of energy-efficient global wire behavior has been further explored for both on-chip [99] and off-chip [134, 133] interconnects. Much of the existing work to reduce the time and energy

costs of long wires has been focused at the level of hardware design and architecture. While this approach is certainly insightful, it begs the question of any communication reduction to be gained at the algorithm and application level. As we will discuss in Chapter 3, the development of algorithms that perform a provably minimal amount of communication has resulted in runtime improvements on both shared and distributed-memory machines (see [13] for a survey). We hypothesize that such algorithms also reduce energy consumption, but the verification of this conjecture remains the subject of ongoing and future research. In Chapter 3, we will define energy models that include the energy and runtime cost of data movement as a combination of wire and memory energies.

Other Hardware Components

In the previous sections, we argued that the dynamic energy on modern server and desktop platforms is predominately consumed by processor packages, SRAMs (primarily caches) and DRAM and discussed the primary sources of energy consumption in CMOS logic. We here cite previous work arguing that other hardware components (such as storage drives, motherboards, fans and power supplies) represent either static or negligible dynamic contributions to total system energy consumption.

Inefficiencies in power supplies for servers and desktops can represent a reasonable portion of the wall energy consumption, typically from 5-20% percent depending on the efficiency rating of the supply and the machine load. Higher efficiency is typically correlated with an intermediate level of load, with the largest amount of energy loss occurring at high or low machine loads. For example, the input/output wattage curves for a Dell server power supply are shown in Figure 2.6. This power supply attains an average efficiency of 87.38% across all loads, with a visible inflection point at around 50% load [1]. As our power measurements are taken at the wall, power supply losses are included in results. We do not regard this as a significant problem, as we focus on modeling specific implementations of algorithms for a given computational problem; i.e. the arithmetic intensity (and thus the power supply load) is nearly constant between runs in a given level of memory (also fixed for our models). Thus, power supply losses can be regarded as a portion of static energy consumption.

Both the PowerMon and PowerPack research teams report motherboard and network interface card (NIC) power to be constant across a series of benchmarks. It is not known if these analyses only involved NICs integrated into the motherboard, but this conclusion is supported by the results of Sohan et al. [124]. Furthermore, both the PowerMon and PowerPack teams reported hard drive power to be negligible (< 10W). With PowerMon, these results held even for benchmarks designed to target NICs and disks. These results for disks contradict older results by Carrera, Pinheiro and Bianchini [39], who note that arrays of disks on certain types of servers (web and proxy, in particular) can represent a significant fraction of overall node energy consumption if using high-performance SCSI disks. NAND-based flash memory disks (SSDs) consume even smaller amounts of power than hard disks [118]. To mitigate any potential problems with storage drive and NIC energies, our experiments for sequential and heterogeneous machines perform no network accesses and are sized to be DRAM-resident.

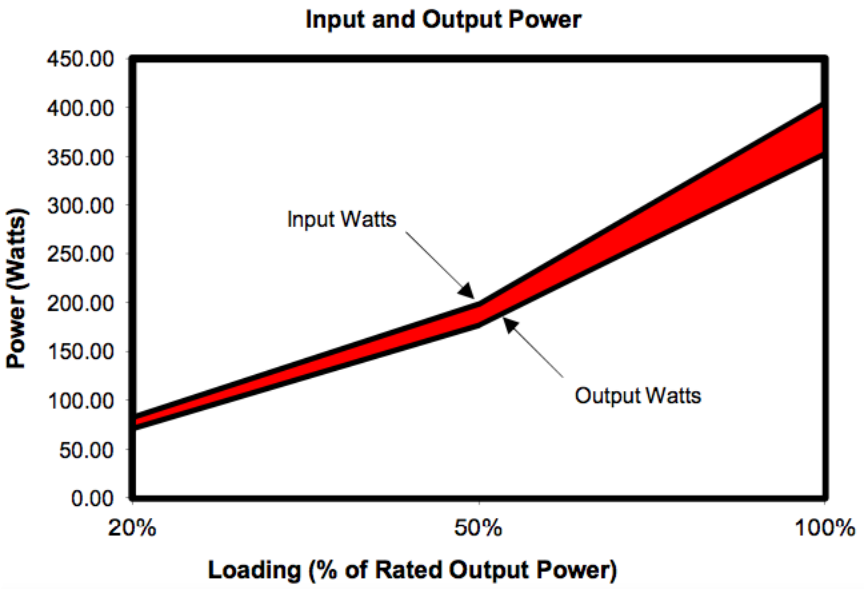


Figure 2.6: Input/Output wattage curves for a Dell DH350E-S0 power supply for 2U servers [1]

The PowerPack results also observe fan power to be constant across applications. This supports our informal observations on a 1U server of low fan idle power and relatively small increases in wall power over long, intensive benchmark runs (i.e. fans do not appear to consume a large amount of power even on a high-temperature machine). We assume that the machines being modeled do not have any other significant sources of energy consumption, such as a graphics processing unit (GPU) attached to the PCIe bus, unless otherwise noted.

2.4 Network Energy Consumption on Distributed Parallel Machines

Despite the problem of energy and power constraints on distributed supercomputers and datacenters performance, measurement equipment for such machines has yet to catch up with demand, making direct node or component-based measurement of distributed machines extremely difficult. Kamil, Shalf and Strohmaier [88] observed this difficulty in 2008, and were forced to use data from power distribution units (PDUs) to measure entire system power without further specificity. They note, however, that Cray systems have management features that allow access to power consumption information at the rack level. Unfortunately, unless the user has access to dedicated racks (difficult on large, job-scheduled systems), this information is of limited utility.

More recent attempts toward instrumentation of distributed machines appear to be focused on node-level instrumentation, such as the custom monitoring equipment of the PowerMon project [26] or use of integrated firmware energy models such as Intel's power meter for monitoring socket and DRAM energies [47]. The PowerPack [66] monitoring device and software stack has also

been applied to the largest known supercomputer instrumented for per-component power at time of writing, System G [41]. In general, integrated power monitors are most likely more scalable, as they may not require additional equipment or significant software development. The limitation of these models (which are performance-counter based, and linear) is that certain workloads may result in inaccuracy if they do not conform to the workloads used to generate model parameters. Due to the challenges in collecting accurate data from entire distributed machines, we argue that validating the sequential model for single machines is sufficient to capture node energy behavior in the distributed environment, assuming homogeneous nodes and asymptotically equivalent data and workload distribution across nodes. Further details about this validation process can be found in Chapter 3.

While some progress has been made with regard to node energy and power consumption, little attention has been paid to measuring internode network costs. Due to this problem, we were unable to directly calculate the network parameters used for distributed parallel machines in Chapters 6 and 7. Instead, we use parameter values from manufacturer specifications, industry reports (such as the Exascale computing study [27]) and modeled parameters from other researchers [44, 43, 91]. Future work may overcome this limitation, as new measurement technologies become available.

As a final point regarding distributed parallel machines, many routers and switches clearly do not demonstrate energy consumption that scales with load (energy-proportional behavior, e.g. when a 50% bandwidth load across a switch consumes 50% of the device's peak power) [106]. This observation of constant network energy is supported by measured evidence of constant NIC energy as discussed in the previous section. While currently a small fraction of overall system power, network energy may eventually become the efficiency bottleneck for future systems as the energy-proportionality of nodes increases with successive hardware generations [2]. To address this observation, we will derive two different models for distributed parallel machines in Chapter 3: one in which communication energy is proportional to bandwidth and message load, and another in which node DRAM accesses are assumed to be the dominant communication component of energy. In this second model, we will model network energy as a constant function of the number of network links.

An underlying idea throughout this chapter is that energy consumption of desktops and distributed servers is dominated by the behavior of a small set of devices; processors, SRAM caches, DRAM memory and (on distributed machines) the internode network infrastructure. These devices themselves are based on two common hardware objects: CMOS logic and communication interconnects. Both CMOS logic and interconnects can be modeled via dynamic and static energy terms, which supports the argument that the larger devices can be modeled in a similar manner. This is the approach that we will use explicitly in Chapter 3 to define energy models for sequential, distributed parallel and heterogeneous machines.

Further, we cited related work that argued for the increasing impact of global interconnects and DRAM on the energy consumption of machines. This argues for the development of communication and energy-optimal algorithms that minimize data movement. In later chapters, we will derive bounds on the amount of energy required to compute a computational kernel, and in certain present optimal algorithms that attain these bounds.

Chapter 3

Machine Models for Runtime and Energy

In this chapter, we describe energy and runtime models for three types of abstract machines: sequential, distributed parallel, and heterogeneous (Section 3.2). Further, we provide introductions to several computational problems, such as dense matrix-matrix multiplication and the $O(n^2)$ n-body problem, that will be used throughout this work to exemplify our approach to bounding energy and considering the application of those bounds (Section 3.3).

Finally, this chapter presents empirical evidence to support our runtime and energy models on sequential and heterogeneous machines. We demonstrate that simple linear models can be fitted accurately ($< 21\%$ relative error vs. direct measurement for dense matrix-matrix and matrix-vector multiplication) for runtime and energy, and argue that such results are applicable to distributed parallel machines (Section 3.4). Finally, we discuss recent related work that attempts to estimate parameters for runtime and energy models via targeted microbenchmarks (Section 3.5).

3.1 Problems, Algorithms, and Implementations

We believe that flexible, abstract models of machine runtime and energy consumption are useful to both software and hardware developers, especially combined with lower bounds on the amount of memory traffic required to solve a computational problem with a given algorithm. Before defining models, we must make some preliminary definitions. First, we define a computational *problem* as a set of questions that may be solved computationally. An example of a computational problem may be dense matrix-matrix multiplication. Further, we describe a method to solve a given computational problem as an *algorithm*. There may be any number of known algorithms for a problem. In the case of matrix-matrix multiplication, examples could be (but not limited to!) the classical $O(n^3)$ approach, Strassen's method [129] or Coppersmith-Winograd [46]. Each of these different algorithms require different asymptotic numbers of floating point operations, and differ greatly in structure.

Once an algorithm is chosen, an *implementation* must be generated for a given piece of hardware. This implementation may be produced manually by a programmer, or automatically via an autotuning framework such as ATLAS [146] for linear algebra. We then measure this implementa-

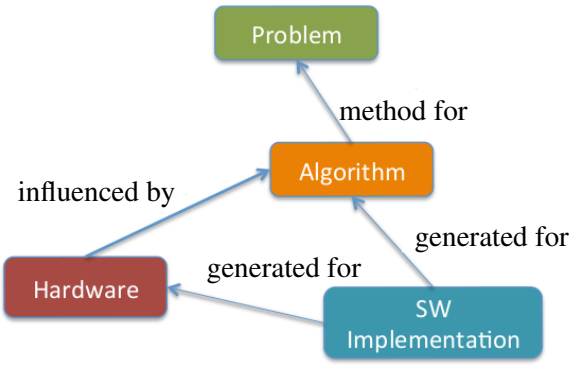


Figure 3.1: Relationship between hardware, algorithm and implementation

tion to determine the throughput or energy efficiency of the code, and improve performance via a tuning process if needed. The general relation between these concepts is illustrated in Figure 3.1. In it, we see that a problem determines a set of possible algorithms that attempt to solve it. Similarly, a set of implementations is determined both by the algorithm and the hardware upon which it runs. Finally, a target set of algorithms influences hardware design. In many cases, this set of algorithms may be very large (i.e. any computable sequence of instructions within the physical limits of storage). In the future, increasingly heterogeneous processors may significantly constrain the space of efficiently-computable algorithms for the sake of efficiency.

High-level machine models of runtime and energy that relate software and hardware have the benefit of providing useful information to both software developers and hardware designers. This approach is not new, and is used commonly in both hardware and software-oriented tasks. However, this work differs in that the development of energy bounds on algorithms allows for a non-trivial "optimal" level of energy consumption to be stated. We believe that this has two main usage models:

- Using measured or fitted hardware parameters to give software experts an indication of how well their code runs on a given machine or machine design.
- Using known optimal algorithm computation and communication relationships to describe sets of hardware parameters that attain specific throughput and energy efficiency goals

3.2 Machine Models

Before one can begin discussing bounds on runtime and energy, a set of abstract machine models must be defined. Such models are simple representations of common computer architectures and allow algorithm designers to describe machine structures and functions via analytic representations. In this work, we will consider three types of machines: serial (S), distributed parallel (DP) and shared-memory heterogeneous (H). In the serial (S) model (left side of Figure 3.2), we

consider a single processing core connected to a fast cache. This cache then has access to an unbounded slow memory and we define communication as the data traffic between the fast and slow memories. For example, this models a single core on a multiprocessor with private L1 ("fast") and L2 ("slow") caches. Furthermore, the distributed parallel (DP) machine on the right side of Figure 3.2 represents a set of homogeneous processing nodes connected via a homogeneous network. Each processing node may access local memory ("fast") or obtain data from the memory of a remote node ("slow"). To avoid having to consider network effects, one could assume that the machine possesses a fully-connected network topology where each link is identical. This is overconservative, and in this work we show that a torus or a mesh network topology of an algorithm-specific degree is sufficient to avoid communication being bounded by link traffic volume. We discuss these results in Section 4.1 and 5.1. For simplicity, we assume that words are packed into contiguous messages before being communicated and represent the time cost of a single message between memories as:

$$T_{\text{msg}} = \alpha_t + \beta_t w$$

where w is the number of words transferred, β_t is the time cost per word (or inverse bandwidth) and α_t is the time cost per message. A more detailed description of the serial and distributed parallel machine models may be found in [15], and it is worth noting that these three models may be composed (e.g. hierarchically) to represent machines of increased complexity. Once abstract machine models are specified, one can define models of runtime and energy upon which to derive bounds on energy.

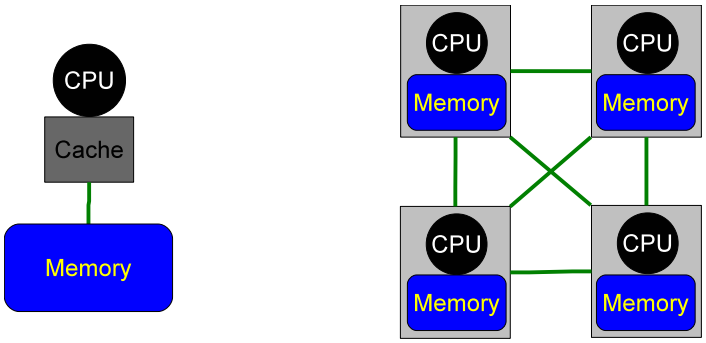


Figure 3.2: Serial (S) and Distributed Parallel (DP) machine models

Sequential Machine Model (S)

The above expression for T_{msg} can be generalized to represent the runtime of an algorithm if we consider all messages that must be transmitted, as well as including a term to describe the *work operations* of the algorithm. In our models, such operations are used as a metric of algorithmic process. A common example is a floating point operation, or flop. Throughout this document, we refer to work operations as flops, but other metrics of progress exist, such as operations over a semi-ring. Thus, we define the runtime of the sequential machine model as

$$T_S = \gamma_t F + \sum_{i=1}^{S} (\beta_t w_i + \alpha_t) = \gamma_t F + \beta_t W + \alpha_t S. \tag{3.1}$$

The three terms in Equation (3.1) are the time to compute "work" operations ($\gamma_t F$), the total time to transmit words of data ($\beta_t W$) and the total time for sending those words in packed messages ($\alpha_t S$). Thus, γ_t is the seconds/work operation, F is the number of "work" operations on data that is stored in fast memory, with the answer being stored in fast memory, β_t is the seconds per read/write of a memory word, W is the total number of words read or written, α_t is the overhead when a number of words are read/written in one operation[1], and S is the total number of read/write operations performed. One can consider γ_t,β_t and α_t to be machine-specific parameters that are derived via microbenchmarking of existing machines or firmware performance counter-based models on new architectures. Note that this definition of runtime assumes that no overlapping of communication and computation occurs. If we were to consider the case of overlapping, runtime would be represented with a $\max$ operation as opposed to a summation in Equation (3.1). Practically, this is a concern. However, communication/computation overlapping would only result in a maximum runtime improvement of 2 or 3 which does not affect any asymptotic conclusions.

Considering the model of runtime defined in Equation (3.1), we can similarly represent the amount of energy E_S consumed by a sequential machine during the course of program execution:

$$E_S = \gamma_e F + \beta_e W + \alpha_e S + \delta_e \hat{M} T_S + \epsilon_e T_S. \tag{3.2}$$

where γ_e is the joules per operation, β_e is the joules per word transferred, α_e is the joules per message, δ_e is the joules per second per word of memory and ϵ_e is the joules per second of idle components on the machine. The first three terms of the equation ($\gamma_e F + \beta_e W + \alpha_e S$) are analogous to the three terms of the runtime equation, and represent the dynamic energy component of the execution. This dynamic energy has two components: the energy to actually perform compute operations ($\gamma_e F$) and the energy to move data ($\beta_e W + \alpha_e S$). Optimally, the machine would consume no energy aside from the joules required for the dynamic component, but this is not the case. The additional idle energy of the machine is represented by the $\delta_e \hat{M} T_S + \epsilon_e T_S$ terms of Equation (3.2) with $\delta_e \hat{M} T_S$ being the idle energy of utilized slow memory and $\epsilon_e T_S$ being the combined cost of other machine components (such as fans, disks, etc) and logic leakage power. In Equation (3.2), $\hat{M}$ is the amount of slow memory utilized during the calculation. We include the energy of idle slow memory (as opposed to cache idle energy) as this conforms with intuition that a larger slow memory consumes more idle energy than a smaller cache. On current hardware, this idle memory cost may not scale with the amount of DRAM actually utilized by the algorithm, but may be the case on certain machines or specialized hardware. As we will see in Chapter 6, this idle memory cost may become large on distributed machines where the amount of utilized memory increases rapidly with the size of the problem (e.g. matrix-matrix multiplication, where the memory footprint of the algorithm grows quadratically with problem size). If desired, the idle energy of cache can be exposed by recursively defining a model with multiple levels, as will be discussed in Section

[1]For example, this could be a cache line transfer.

3.2. We will see later that the bounds on W and S may include another parameter, M, the size of fast memory .

Distributed Parallel Machine Model 1 (DP1)

In the distributed parallel machine model (right side of Figure 3.2) we use an analogous expression for runtime as in the sequential case. As the P processing nodes and communication links are assumed to be homogeneous in capability, a uniform partition of work is theoretically optimal and allows all processors to finish simultaneously in parallel. The assumption of a uniform work partition is appropriate for a lower bound, and is attained by many algorithms, though not all. In other words, we assume that each processor on the machine is asymptotically allocated the same number of work operations F and communication traffic (W and S). This is a significant assumption, but is appropriate for the algorithms discussed within this work. There may be many other problems in which F, W, and S cannot be evenly divided. Note that in the distributed parallel machine, W and S are described on a per-processor basis. Unlike the sequential model, the communication terms now describe interprocessor communication generated or received by a node as opposed to communication between levels of the cache hierarchy.

One other aspect of a distributed machine model that should be addressed is the topological nature of the interconnect. If the network is fully-connected, machine performance may only be constrained by the volume of data moved by each of the processing nodes (the *per-processor* word and message counts). On the other hand, if the network topology is such that the amount of data transferred over the worst-case network link (the *link contention* word and message counts) dominates the per-processor volume, the algorithm's performance may be adversely affected. In this work, we limit discussion of link contention exclusively to toroidal and mesh networks. This excludes indirect networks (i.e. networks with switches that do not perform computation) such as butterflies, fat trees and others.

We present an intuitive formulation of the runtime model, which is analogous to the sequential model of Equation (3.2):

$$T_{DP1} = \gamma_t F + \beta_t W + \alpha_t S. \tag{3.3}$$

If all homogeneous nodes compute in parallel and have identical runtime, the total energy required in the parallel distributed case is the energy consumed by one processor multiplied by the number of processors P:

$$E_{DP1} = P(\gamma_e F + \beta_e W + \alpha_e S + \delta_e M T_{DP1} + \epsilon_e T_{DP1}). \tag{3.4}$$

Model Compositions and Distributed Parallel Model 2 (DP2)

The machine models discussed in the previous sections represent runtime and energy at a very high level of abstraction and are suitable for analysis by hand. This level of abstraction considers communication solely between two levels of a memory hierarchy. As a result of this, the behavior of other levels of the memory hierarchy and machine components (such as branch predictors) are lumped into two parameters (γ_t and γ_e). These parameters may then vary significantly between

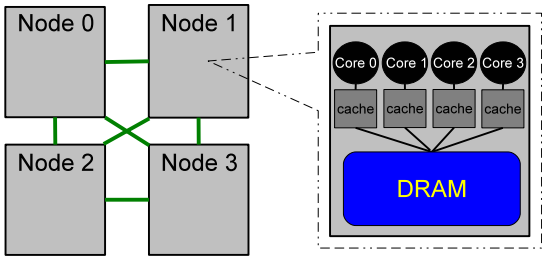


Figure 3.3: Composition of Sequential (S) and Distributed Parallel (DP) Machine Models

implementations of the same algorithm as a new implementation may be more efficient in levels of memory that have been abstracted out of the model. To some extent, this problem may be mitigated by composing models recursively and exposing more parameters. In Figure 3.3, we depict a 2-level composition of the distributed parallel (DP1) and modified sequential (S) machine models wherein a distributed machine contains nodes composed of multiple processing cores linked to DRAM via private caches. We now explicitly derive a model for this abstract machine. First, we begin with an expression for runtime (identical to the DP1 model):

$$T_{DP2} = \gamma_{t_0} F_0 + \beta_{t_0} W_0 + \alpha_{t_0} S_0$$

with the modification that we now represent parameters that pertain to the entire distributed machine as being on "level 0" and have annotated the equation accordingly (e.g. W_0 is the number of words sent from one node to another over the green links indicated in Figure 3.3). Similarly, parameters that relate to node performance will be referred to as at "level 1". To define level 1 behavior, we first replace $\gamma_{t_0} F_0$ by T_{core} where

$$T_{core} = \gamma_{t_1} F_1 + \beta_{t_1} W_1 + \alpha_{t_1} S_1$$

where $F_1 = F_0/P_1$ where P_1 is the number of homogeneous processing cores with identical fast memories. Note that we assume that all on-node processors will finish simultaneously due to perfect load balancing. This assumption can be removed by taking the maximum runtime over the nodes and cores. For the moment, we assume identical runtimes to reduce notational complexity. We can now substitute the expression for T_{core} into that of T_{DP2} to complete the runtime model of this abstract machine:

$$T_{DP2} = (\gamma_{t_1} F_1 + \beta_{t_1} W_1 + \alpha_{t_1} S_1) + \beta_{t_0} W_0 + \alpha_{t_0} S_0. \tag{3.5}$$

Regarding energy, we begin with an expression for energy identical to the DP1 model with P_0 nodes:

$$E_{DP2} = P_0(\gamma_{e_0} F_0 + \beta_{e_0} W_0 + \alpha_{e_0} S_0 + \delta_{e_0} M T_{DP2} + \epsilon_{e_0} T_{DP2})$$

and replace $\gamma_{e_0} F_0$ by $P_1 E_{core}$ where the per-core energy, E_{core}, is

$$E_{core} = \gamma_{e_1} F_1 + \beta_{e_1} W_1 + \alpha_{e_1} S_1 + \epsilon_{e_1} T_{DP2}.$$

For simplicity, we omit a term to explicitly account for cache idle energy as caches typically have high utilization and assume it to be a component of ϵ_{e_1}. Thus, the total energy for a generic 2-level model is

$$\begin{aligned} E_{DP2} = P_0(P_1(&\gamma_{e_1} F_1 + \beta_{e_1} W_1 + \alpha_{e_1} S_1 + \epsilon_{e_1} T_{DP2}) \\ &+ \beta_{e_0} W_0 + \alpha_{e_0} S_0 + \delta_{e_0} M T_{DP2} + \epsilon_{e_0} T_{DP2}). \end{aligned}$$

According to Google researchers [2], modern datacenter interconnects do not scale energy consumption with load and thus typically represent a constant fraction of the machine's energy budget. To address this situation, we may modify energy expression for the above two-level distributed model to represent the assumption that network energy scales only with the number of links, and not the amount of actual communication:

$$E_{DP2} = P_0(\gamma_{e_1} F_1 + \beta_{e_1} W_1 + \alpha_{e_1} S_1 + \delta_{e_0} M T_{DP2} + \epsilon_{e_0} T_{DP2}) + \zeta |\hat{E}(G_{Net})| \tag{3.6}$$

where we also model the nodes as sequential machines ($P_1 = 1$) for simplicity, ζ is the energy cost per link and $|\hat{E}(G_{Net})|$ is the number of links in the internode network graph G_{Net}. To simplify the model, we have also assumed that $\epsilon_{e_1} T_{DP2}$ is encompassed within $\epsilon_{e_0} T_{DP2}$. We define the runtime and energy models within Equations (3.5) and (3.6) as the DP2 model of a distributed parallel machine.

Heterogeneous Machine Model (H)

A model for heterogeneous machines is shown in Figure 3.4. This model is first described in [17] and extended here to allow for a discussion of energy. To capture the non-uniform nature of a heterogenous computing environment, we model the machine to be a set of compute elements proc_i ($1 \leq i \leq P$) with varying characteristics connected via independent links to a shared global memory. Each proc_i can be defined very abstractly; for example, one compute element may be a single processor, GPU, or a shared-memory multiprocessor itself. Each proc_i is associated with a set of element-specific parameters:

- β_{t_i}: seconds per word of proc_i's communication link to global shared memory
- α_{t_i}: seconds per message of proc_i's communication link to global shared memory
- γ_{t_i}: seconds per operation of proc_i on data that resides in local memory
- β_{e_i}: joules per word of proc_i's communication link to global shared memory
- α_{e_i}: joules per message of proc_i's communication link to global shared memory

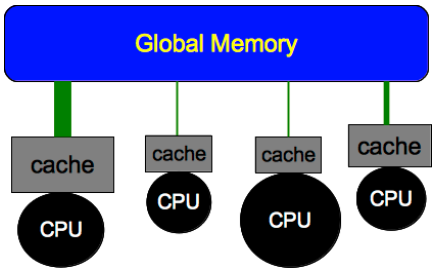


Figure 3.4: Heterogeneous machine model

- γ_{e_i}: joules per operation of proc$_i$ operating on data that resides in local memory
- δ_{e_i}: joules per word per second to store data in the local memory of proc$_i$
- M_i: Size of the local memory of proc$_i$

In addition, we define several other parameters:

- $\hat{\delta}$: joules per word per second to store data in the global memory
- ϵ_H: joules per second leakage term for the entire heterogeneous machine
- $\hat{M}$: the amount of global memory utilized by the execution

We assume that the initial data for the problem is stored within the machine's global memory. For the purposes of this work, global memory is assumed to be always able to fit the data required by the given problem and algorithm. Thus, the layout of data in global memory becomes a factor for performance: in order to read/write a set of words in one message, those words must be contiguously stored in global memory. We also assume that the output of the algorithm will be transferred to global memory. By assuming that the problem begins and ends in global memory, the size of the input and output data represent one lower bound on communication. Note that when $i = 1$, this model reduces to the sequential two-level memory model. However, in the case that all the element-specific parameters are equivalent across compute elements (e.g., $\beta_{t_i} = \beta_t$ for $1 \leq i \leq P$), this heterogeneous model does not reduce to the parallel distributed-memory model shown in Figure 1(b) as it is based on a global shared memory.

We use a similar notation for the runtime and energy expressions in the previous models, but must annotate the hardware parameters for each element as described above. Hence the runtime for $proc_i$ is

$$T_{H_i} = \gamma_{t_i} F_i + \beta_{t_i} W_i + \alpha_{t_i} S_i.$$

As in the models described previously, we assume that no communication/computation overlapping occurs. Assuming that the total number of operations F is partitioned amongst the processors, we represent the total runtime T_H of the heterogeneous machine as

$$T_H = \max_{1 \leq i \leq P} T_{H_i} = \max_{1 \leq i \leq P} (\gamma_{t_i} F_i + \beta_{t_i} W_i + \alpha_{t_i} S_i) \tag{3.7}$$

because the time to complete the execution is limited by the runtime of the slowest processing element. Note that the model implicitly assumes that no program dependencies create idle time on some processors. To model the energy of a heterogeneous machine, we first represent the energy of a single processing element as

$$E_{H_i} = \gamma_{e_i} F_i + \beta_{e_i} W_i + \alpha_{e_i} S_i.$$

In bounds on W_i and S_i, M_i will equal the physical size of processor i's fast memory. We also omit a leakage term for the individual processors. For simplicity, we represent the leakage energy for the entire machine with a single term ($\epsilon_H T_H$) and the idle global memory power with another term ($\hat{\delta}\hat{M}T_H$). Thus, we model the total execution energy of the heterogeneous machine as

$$E_H = \sum_{i=1}^{p} E_{H_i} + \hat{\delta}\hat{M}T_H + \epsilon_H T_H. \tag{3.8}$$

Depending on the future application of the model in Equation (3.8), $\hat{\delta}\hat{M}T_H$ and $\epsilon_H T_H$ may be decomposed into the individual contributions of constituent processing elements. One may also note that each word of global memory contributes an energy penalty for the entire duration of T_H, and not for its specific duration of usage. By this, we implicitly assume that the amount of global memory is constant for the duration of the algorithm. This applies to each of the M_i-sized fast memories as well. Thus, we assume the energy to use a word in global memory is $\hat{\delta}T_H$ even if the actual time of usage, is much smaller than T_H. The heterogeneous model can be extended in a hierarchical manner to more-closely relate to actual hardware. In the case that $proc_i$ represents multiple computational units (heterogeneous or not) with a shared memory, one could apply this model to the element individually to obtain a more accurate representation of γ_{t_i} and γ_{e_i}.

3.3 Problems of Particular Focus

In this work, we focus our analysis on a small set of computational kernels that are similar to a wide class of problems within the scientific computing community. In addition to wide applicability, this set of problems can be solved via several well-researched algorithms as well as representing a range in potential *arithmetic intensity*, or the ratio of the number of floating point operations to words of memory loaded or stored (i.e. the *flop/word ratio* of an implementation). In the language of computational motifs (where motifs are general classes of computational problems, see Table 3.1 or "The Berkeley View on Parallel Computing" [9] for more details), these problems represent dense linear algebra (matrix-matrix and matrix-vector multiplication), sparse linear algebra (matrix-vector multiplication) and n-body methods (the $O(n^2)$ n-body problem). In this section, we provide an overview of this set of problems as well as any associated algorithms pertinent to this work. Later in this document, we will use models of runtime and energy to derive

Dense Linear Algebra	Sparse Linear Algebra
Spectral Methods	N-Body Methods
Structured Grids	Unstructured Grids
MapReduce	Combinatorial Logic
Graph Traversal	Dynamic Programming
Back-track & Branch and Bound	Graphical Models
Finite State Machines	

Table 3.1: Computational motifs as described in [9]

lower bounds on these algorithms. With lower bounds, we will then explore the impact of runtime, energy and power constraints on energy efficiency. Finally, we will describe the implications for hardware parameters if attempting to attain a target level of energy efficiency. For simplicity, we will assume that the problems and associated algorithms are operating on double-precision floating point values with a size of 8 bytes.

Matrix-vector multiplication

The first computational problem we consider is that of matrix-vector multiplication. This problem is a common component of many dense and sparse linear algebra problems as well as several other computational motifs such as spectral and grid problems. In particular, operations that require $O(n^3)$ flops when computed directly can be accomplished in $O(n)$ or $O(nlog(n))$ flops with iterative methods that involve multiple matrix-vector multiplications (e.g. multilevel and conjugate gradient methods for solutions of differential equations, see [152] and [122] for more details). These iterative methods often involve repeated matrix-vector multiplications to converge to a solution, and as such this operation is a critical computational kernel for scientific computing. Mathematically, the matrix-vector problem is defined as

$$y_i = \sum_{j=0}^{n-1} A_{ij} x_j$$

for all $i = 0..m-1$ where A is an m-by-n matrix and x and y are vectors of length n and m, respectively. We use 0-based indexing of matrices and vectors for the remainder of this work. If A has a ratio of non-zero to total entries that approaches 1, it is often considered to be a *dense* matrix, i.e. an implementation would make no distinction between zero and nonzero values. If dense, the matrix is commonly stored in memory as a contiguous array and a naive implementation of dense matrix-vector multiplication (GEMV) is described in Algorithm 1. Note that Algorithm 1 performs 2 floating point operations on each word of A_{ij}. Thus, in many cases the theoretical upper bound on arithmetic intensity of dense matrix-vector multiplication is 2. If the vectors are unable to fit in cache, this value is reduced somewhat due to the additional memory traffic. Dense matrix-vector multiplication is easily parallelized by dividing the rows of A across the processors

and replicating the input vector x. In Chapter 4, we use this approach to describe a communication and energy-optimal algorithm on a heterogeneous machine model.

Algorithm 1 naiveMatrixVector(A,x,y,n)

Input: n-by-n matrix A and vector x of length n
Output: vector y of length m
Initialize $y_i = 0$
for $i = 0..m-1, j = 0..n-1$ **do**
  $y_i \mathrel{+}= A_{ij} \cdot x_j$
end for

If the input matrix A is mostly composed of zeros, we may wish to treat it as a *sparse* matrix and store the fewest zero values as possible. Such goals require a data structure to store A that is a bit more complex than the dense data structure. In this work, we assume that the input matrix is stored in Compressed Sparse Row (CSR) format [23], which stores only non-zeros. This data format assures contiguous access of non-zero values along rows, and requires three arrays to represent the matrix: an integer row offset array r of size $m+1$, a floating point value array v that stores the non-zero entries of A, and an integer column index array $indx$ that indicates the column location of a non-zero entry.[2] Thus, the CSR format requires a memory footprint of $3 * nnz/2 + (m+1)/2$ 8-byte words where nnz is the number of non-zero entries in sparse m-by-n matrix A. We assume 8-byte data words (elements of v) and 4-byte row offsets and column indices. An example of a 5-by-5 sparse matrix stored in CSR format is illustrated in Figure 3.5. The dense representation of the matrix is shown on the left, and the CSR representation on the right of the figure. Note that the row offset array indicates the beginning and end of each row, and is usually more memory-efficient than a format that stores both the row and column indices for each non-zero (a *coordinate* storage format). Many other types of storage formats for sparse matrices have been proposed such as block CSR, Skyline Storage (SKS), and compressed diagonal storage (CDS)[23]. In practice, the optimal choice of storage format depends on the sparsity structure of the matrix and the computational problem to be solved. An overview of such tradeoffs can be found in [23].

A naive implementation of sparse matrix-vector multiplication (SpMV) with the CSR storage format is shown in Figure 2. Note that like the dense case of Algorithm 1, the i loop iterates over all m rows of the matrix. A key difference, however, is that the column index (j) only operates on non-zero entries stored in the array v. Along each row, this access to the values of A is continuous in memory and is efficient. The problem lies with vector x as it is indirectly accessed via the column array $indx$. This irregular memory access pattern represents a significant challenge when attempting to develop an efficient implementation of SpMV, and results in an arithmetic intensity that varies throughout the execution of the algorithm. In this work, we assume that a single average intensity per matrix is sufficient to describe an SpMV execution. This assumes a level of regularity in the input sparse matrix so that relatively constant cache miss behavior during algorithm execution is attained. One could imagine a counterexample to this assumption in a sparse

[2] By convention, the final element equals nnz with zero-based indexing.

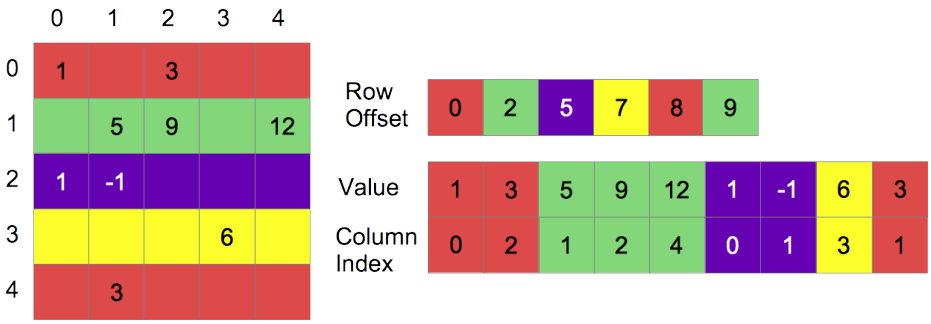


Figure 3.5: Compressed Sparse Row (CSR) storage format

matrix that has one structure for the first $m/2$ rows, and a completely different structure for the last $m/2$ rows.

Due to their low arithmetic intensity, both dense and sparse matrix-vector multiplication are often performance-limited by the bandwidth and latency parameters of the machine, and not the peak floating point rate. Thus, matrix-vector implementations typically achieve a small fraction of machine's potential floating point throughput even when expertly (or automatically) tuned. The difficulty of optimizing SpMV is such that several software tools have been developed to automatically generate optimized serial [142] and shared-memory parallel SpMV [85] operations.

Algorithm 2 sparseMatrixVector(r,v,$indx$,n)

/* m-by-n matrix A in CSR format
Input: row array r of length $m+1$, value array v of size nnz and column index array $indx$ of length nnz
Input: array x of length m
Output: vector y of length n
Initialize $y[i] = 0$
for $i = 0..m-1$ **do**
  for $j = r[i]..r[i+1]-1$ **do**
    $y[i] \mathrel{+}= v[j] \cdot x[indx[j]]$
  end for
end for

As mentioned earlier, many iterative methods repeatedly apply a matrix to a vector via matrix-vector multiplication (Krylov subspace methods, such as conjugate gradients, Lanczos, etc.). While both Algorithm 1 and Algorithm 2 only consider a single matrix-vector multiplication, some algorithms are able to implement Krylov subspace methods with provably minimal data traffic (loading A into memory only a constant number of times, as opposed to a function of the number of SpMV operations) by applying multiple matrix-vector operations simultaneously. Such algorithms have shown significant speedup over previous approaches, and may also trade extra flops for reduced communication volume by replicating intermediate values of the computation. While we only con-

sider runtime and energy lower bounds on single matrix-vector multiplications (see Chapter 4), the reader is referred to Mohiyuddin et al.[111] for a floating point and communication analysis of these algorithms. We believe our bounds could be extended to these algorithms in future work.

Computation of a matrix-vector multiplication problem on distributed parallel machines is accomplished by blocking the input data across the processor local memories, and communicating slices of the x vector between nodes. Blocks of the input matrix are not communicated. Each processor has enough data to multiply by its own on-diagonal blocks, but must obtain values of x from other processors to multiply by its off-diagonal blocks. Traditionally, optimizing the communication pattern for this algorithm was approached via minimization of the edge cut between matrix vertices on different processors. However, as noted by Hendrickson [73], communication is actually proportional to the number of vertices that must send or receive data from another processor, and not the edge cut (which overestimates the required communication). This issue can be resolved via the use of hypergraph partitioning [74, 40] to attain a better decomposition of the input matrix onto processors. We suspect that communication lower bounds for parallel distributed matrix-vector multiplication could be represented as a function of the minimal hypergraph cut, but place such a problem beyond the scope of this thesis. Thus, we only consider matrix-vector multiplication on sequential and heterogeneous machines.

Matrix-matrix Multiplication

The next computational problem is dense matrix-matrix multiplication (GEMM) on square matrices. In this work, we limit our discussion to dense matrices, but communication bounds on sparse matrix-matrix multiplication can be found within [18]. Mathematically, given three dense matrices A,B and C of size $n \times n$, we wish to compute

$$C_{ij} = \sum_{k=0}^{n-1} A_{ik} B_{kj}$$

for all $i, j = 0...n-1$ and where C_{ij} is the element at the ith row and jth column of matrix C. As we wish to iterate over all values of i, j, k, the operation as defined requires $2n^3$ floating point operations, or *flops*, to compute. We note that this operation theoretically performs $O(n^3)$ flops on $O(n^2)$ data words (the three matrices with n^2 elements), and thus has a flop/word ratio of $O(n)$. While not indicative of performance when run on actual machines, this computational intensity seems to imply that the problem has the potential to have throughput (Gflop/s) limited by the peak floating point capability of the machine, and not the much more constrained main memory bandwidth. Indeed, tuned implementations of algorithms that solve dense matrix-matrix multiplication are able to attain a significant fraction of the peak floating point rate of the machine.

In other dense matrix operations, we note that matrix-matrix multiplication often represents the most computationally-intensive component. For example, various types of matrix factorizations, such as LU, QR and the SVD, update trailing submatrices via matrix-matrix multiplications (see [56] for a detailed overview of these factorizations). For example, the LU factorization is typically used to solve linear systems of equations. It factorizes $A = LU$ where L is a matrix with zero

values above the diagonal (a *lower triangular* matrix) and U is a matrix with zero values below the diagonal (an *upper triangular* matrix). During the execution of the common algorithm to compute L and U, a subset of columns is factorized and the remainder (or trailing portion) of the matrix is updated via a matrix-matrix multiplication prior to factorization of the next set of columns. As this operation requires a significant amount of computation, optimizing matrix-matrix multiplication can significantly improve the performance of an algorithm that performs LU factorization. Thus, the study of dense matrix-matrix multiplication has significant import to the larger linear algebra community [7].

Classical $O(n^3)$ Matrix-matrix Multiplication Algorithm A direct implementation of the mathematical definition of matrix-matrix multiplication as three nested loops (see Algorithm 3) is inefficient due to the limited arithmetic intensity that it is able to achieve. Such an implementation is commonly called *naive*. In other words, the machine spends a much larger amount of time moving data as opposed to computing the required floating point arithmetic. The naive implementation of matrix-matrix multiplication can be trivially parallelized on shared memory machines via a parallel loop construct on the i or j loops (the k loop creates a write dependency on values of C).

Algorithm 3 naiveMatrixMultiplication(A,B,C,n)

Input: n-by-n matrices A and B
Output: n-by-n matrix C
for $i = 0..n-1, j = 0..n-1, k = 0..n-1$ **do**
  C_{ij} += $A_{ik} \cdot B_{kj}$
end for

Algorithm 4 blockedMatrixMultiplication(A,B,C,n)

Input: n-by-n matrices A and B
Output: n-by-n matrix C
for $i = 0..n/b-1, j = 0..n/b-1, k = 0..n/b-1$ **do**
  /* Perform block multiplication */
  naiveMatrixMultiplication($A(i*b : i*(b+1)-1, k*b : k*(b+1)-1, B(k*b : k*(b+1)-1, j*b : j*(b+1)-1), C(i*b : i*(b+1)-1, j*b : j*(b+1)-1), n/b$)
end for

Efficient (or *tuned*) implementations of classical $O(n^3)$ dense matrix-matrix multiplication utilize a number of techniques to increase the intensity and throughput, such as loop unrolling or various levels of cache and register blocking (see Algorithm 4 for an example of blocking and [70] for an overview of such techniques). In particular, it can be shown that a sequential blocked implementation is able to achieve the asymptotically maximal arithmetic intensity across levels of the memory hierarchy with $b \approx M^{1/2}$ to attain the communication lower bound [86]. Such tuned implementations can be found in linear algebra libraries such as Intel's Math Kernel Library (MKL)

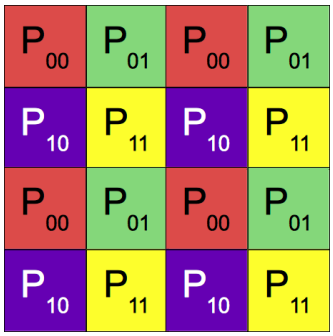


Figure 3.6: Two-dimensional block cyclic distribution of a matrix on a 2-by-2 processor grid

[138] or the AMD Core Math Library (ACML)[5], and can also be generated automatically via autotuning frameworks such as ATLAS [146].

On distributed parallel machines, matrix classical matrix-matrix multiplication can be implemented with the optimal level of arithmetic intensity via Cannon's algorithm [38]. In practice, however, classical matrix multiplication in a distributed environment is often implemented via Scalable Universal Matrix Multiplication Algorithm (SUMMA) [67]. This algorithm relies on a 2-dimensional block-cyclic data decomposition (see Figure 3.6) to map blocks of input data onto a $P^{1/2} \times P^{1/2}$ processor grid. With such a data layout, interprocessor communication becomes a series of row and column data broadcasts as each processor obtains required data from neighbors. SUMMA is easier to implement for general matrices than Cannon's algorithm, and the most-common implementation can be found in the ScaLAPACK library [32].

2.5D $O(n^3)$ Algorithm Matrix-matrix Multiplication Algorithm While SUMMA is a common algorithm to compute classical matrix-matrix multiplication, in this work we choose to model the energy and runtime characteristics of matrix-matrix multiplication on distributed parallel machines via the 2.5D algorithm [126]. This algorithm is able to replicate the input data to reduce communication volume, and thus lends itself to consideration of interesting optimization problems such as those explored in Chapter 6. Replicating data to reduce the communication volume of matrix-matrix multiplication is not a new idea, as so-called 3D matrix-matrix multiplication algorithms have been presented by a number of researchers and proven to attain theoretical lower bounds on the amount of required communication [52, 3, 4, 87]. These algorithms map data blocks onto a $P^{1/3} \times P^{1/3} \times P^{1/3}$ processor grid and replicate input data along one processor axes. A 3D matrix-matrix multiplication algorithm is described in Algorithm 5. Note that once the replicated blocks of A and B have been broadcast, the entire 3D processor grid can compute block C_{ijk} in parallel without further communication aside from a final reduction to form C along the ik face of the processor grid.

The 2.5D algorithm differs from 3D matrix-matrix multiplication algorithms in that the amount of data replication is parameterized. Thus, the input matrices are now mapped onto a $(P/c)^{1/2} \times (P/c)^{1/2} \times c$ processor grid, where c is the number of replications of the input data. If $c = 1$, the 2.5D algorithm runs identically to SUMMA (a *2D* algorithm, as it maps data blocks on a 2D

Algorithm 5 3D dense matrix-matrix multiplication (A,B,C,n,P)

Input: n-by-n matrix A distributed so that P_{ij0} owns $\frac{n}{P^{1/3}}$-by-$\frac{n}{P^{1/3}}$ block A_{ij} for each i,j
Input: n-by-n matrix B distributed so that P_{0jk} owns $\frac{n}{P^{1/3}}$-by-$\frac{n}{P^{1/3}}$ block B_{jk} for each j,k
Output: n-by-n matrix C distributed so that P_{i0k} owns $\frac{n}{P^{1/3}}$-by-$\frac{n}{P^{1/3}}$ block C_{ik} for each i,k
for all $i, j, k \in \{0, ..., P^{1/3} - 1\}$ **do**
  P_{ij0} broadcasts A_{ij} to all P_{ijk} */* Replicate A on each ij processor layer */*
  P_{0jk} broadcasts B_{jk} to all P_{ijk} */* Replicate B on each jk processor layer */*
  $C_{ijk} := A_{ij} \cdot B_{jk}$
  P_{ijk} contributes C_{ijk} to a sum-reduction to P_{i0k}
end for

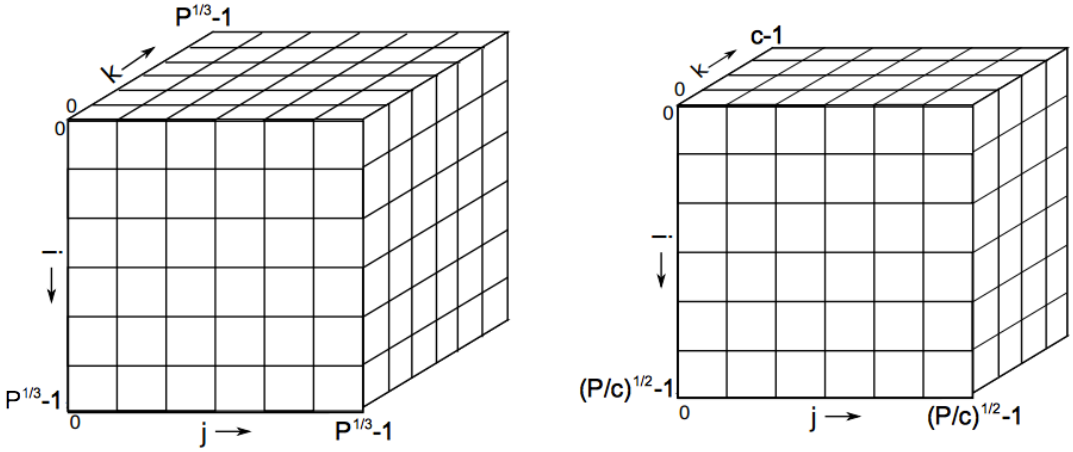


Figure 3.7: Processor grids for 3D and 2.5D matrix-matrix multiplication [126]

processor grid). At the maximal replication value of $c = P^{1/3}$, the algorithm runs identically to 3D matrix-matrix multiplication algorithms [52, 3, 4, 87].

At a certain amount of replication, the overhead of replicating data exceeds that of the communication benefits. For 2.5D matrix multiplication, this point occurs at $c = P^{1/3}$ [126]. We will later refer to this limit on data replication as a *memory-independent per-processor* communication lower bound[21]. By replicating data to reduce communication, the 2.5D algorithm is able to perfectly strong scale in runtime [126] with constant energy [55]. That is, for a fixed problem size the runtime halves with every doubling of processors while using the same amount of energy. We will further explore these scaling properties in Chapters 4, 5 and 6.

Recursive $O(n^3)$ Matrix-matrix Multiplication Algorithm In Chapter 4, we present a communication-optimal algorithm for $O(n^3)$ matrix-matrix multiplication on a heterogeneous machine model. In particular, this algorithm differs from the naive implementation in that the problem is decomposed recursively, as opposed to iteratively. Suppose we are given three matrices, A,B and C of size $2^n \times 2^n$. We can then consider the problem as the matrices each being composed of submatrices

A_{ik},B_{kj} and C_{ij}. I.e.

$$AB = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix} = \begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = C \tag{3.9}$$

where

$$\begin{aligned} C_{00} &= A_{00}B_{00} + A_{01}B_{10} \\ C_{01} &= A_{00}B_{01} + A_{01}B_{11} \\ C_{10} &= A_{10}B_{00} + A_{11}B_{10} \\ C_{11} &= A_{10}B_{01} + A_{11}B_{11} \end{aligned} \tag{3.10}$$

and the matrix-matrix multiplication problem of size $2^n \times 2^n$ has been decomposed into eight subproblems of size $2^{n-1} \times 2^{n-1}$ [34]. In practice, the recursion continues until the subproblems are small enough to fit in cache, and then a tuned classical algorithm is used to compute the base case problems. If the original input matrices are not of size $2^n \times 2^n$, they can be zero-padded to the proper size for the recursion. In Chapter 4, we will see how such a recursive decomposition provides the means to allocate an asymptotically-optimal amount of work across a set of heterogeneous processors.

$O(n^\omega)$ Fast Matrix Multiplication Algorithms In the previous discussion, we consider several algorithms for dense matrix-matrix multiplication that require $O(n^3)$ floating point operations to compute $C = A \cdot B$ with three n-by-n matrices. This amount of work is by no means required to compute the problem, and a number of algorithms are able to compute the problem with $O(n^\omega)$ flops where $\omega < 3$ [129, 120, 128, 149, 46]. Trivially, $\omega \geq 2$ as each element of the $O(n^2)$ input data must be touched at least once and the current best algorithm is able to attain $\omega = 2.3728639$ [97]. Matrix-matrix multiplication algorithms that require $O(n^\omega)$ flops for $\omega < 3$ are collectively called *fast matrix multiplication* algorithms.

While fast matrix multiplication algorithms asymptotically improve upon the performance of the classical algorithms, very few are practical to implement due to large constant factors. One such practical fast matrix multiplication algorithm is that of Volker Strassen[129], which computes matrix multiplication with $\omega = log_2 7 \approx 2.8074$. This recursive algorithm realizes its asymptotic improvement by reducing the number of required recursive calls to seven, as opposed to the eight required by the recursive classical algorithm discussed earlier. In particular, Strassen realized that the seven multiplications of submatrices shown via Equations (3.11) can be used to compute to four output submatrices of C without additional multiplications (Equations (3.12)). In practice, once the problem has been reduced to a sufficient size, a tuned classical algorithm is used to compute the base case of the recursion. Note that the reduced number of multiplications comes at the cost of additional addition operations and memory space for intermediate values. Due to this overhead, an optimized implementation of Strassen's algorithm is less-efficient than tuned classical $O(n^3)$

implementations for small problem sizes.

$$
\begin{aligned}
T_0 &= (A_{00} + A_{11})(B_{00} + B_{11}) \\
T_1 &= (A_{10} + A_{11})B_{00} \\
T_2 &= A_{00}(B_{01} - B_{11}) \\
T_3 &= A_{11}(B_{10} - B_{00}) \\
T_4 &= (A_{00} + A_{01})B_{11} \\
T_5 &= (A_{10} - A_{00})(B_{00} + B_{01}) \\
T_6 &= (A_{01} - A_{11})(B_{10} + B_{11})
\end{aligned} \tag{3.11}
$$

$$
\begin{aligned}
C_{00} &= T_0 + T_3 - T_4 + T_6 \\
C_{01} &= T_2 + T_4 \\
C_{01} &= T_1 + T_3 \\
C_{11} &= T_0 - T_1 + T_2 + T_5
\end{aligned} \tag{3.12}
$$

Implementing Strassen's algorithm on a distributed parallel machine is challenging due to the problem of efficiently distributing the subproblems of the recursion tree across the processors. However, a handful of implementations have demonstrated performance improvements over classical implementations for large enough problem sizes (see [98] for a list). In particular, Communication-Avoiding Parallel Strassen (CAPS) [98] implements Strassen's algorithm on distributed parallel machines and was able to attain speedups over several classical (and other Strassen-based) implementations [98].

CAPS is unique among parallel implementations of Strassen's algorithm in that it is able to trade additional memory usage for reduced interprocessor communication. While it also parameterizes its use of additional memory to reduce communication, CAPS does not achieve this tradeoff via input data replication like the classical $O(n^3)$ 2.5D algorithm. Instead, CAPS computes the required recursive subproblems via two different type of recursion steps: Breadth-First and Depth-First searches (BFS and DFS, respectively). Figure 3.8 illustrates the distinction between BFS and DFS steps. In a BFS recursion, the initial matrix-matrix problem $A \cdot B$ is divided into seven smaller recursive problems (see Equations (3.11)) and the work is divided between all processors. In the DFS case, all processors work in parallel to solve single subproblems in sequence. BFS steps use 1/7 of the available processors on each subproblem, but require additional memory, in order to reduce the overall communication volume. Ballard et al.[20] demonstrate that a sufficient number of DFS steps followed by all BFS steps is sufficient to prove that CAPS is a communication-optimal fast matrix multiplication algorithm. In addition, Lipshitz et al.[98], considers the performance implications of more-complicated interweavings of BFS and DFS steps. We derive runtime and energy lower bounds on fast matrix multiplication algorithms in Chapter 4.

Figure 3.8: Breadth-First or Depth-First traversals of recursion tree [98]

$O(n^2)$ n-body problem

The final class of problems that we consider is that of n-body problems. These problems can be qualitatively described as a set of objects moving in space and interacting with each other (and perhaps the environment) according to some force definition. N-body problems commonly occur in physics and material simulations, and in other scientific domains. A naive algorithm for this problem is shown in Algorithm 6. Note that this version of the problem requires $O(n^2)$ executions of the ApplyForce() function for each time step as each body interacts with every other. More complicated algorithms to solve the n-body problem, such as Barnes-Hut [22] and the Fast Multipole Method (FMM)[71], decompose the environment into tree-based data structures (quad or octtrees [94]) and calculate the desired forces with $O(nlog(n))$ or $O(n)$ force function call, respectively. For ease of analysis, we only consider the situation where $O(n^2)$ executions of applyForce() must occur.

Algorithm 6 naiveNbody(p,n,$nsteps$)

Input: Array of bodies p of length n
Input: Number of time steps to simulate: nsteps
for $step = 0..nsteps - 1$ **do**
  for $i = 0..n - 1, j = i + 1..n - 1$ **do**
    ApplyForce(p_i,p_j,f_i,f_j) // calculate force on particles p_i and p_j, and store in f_i and f_j
  end for
  for $i = 0..n - 1$ **do**
    move(p_i,f_i) // update particle position and velocity according to the forces calculated in ApplyForce()
  end for
end for

Specifically, we will lower bound and analyze the runtime and energy performance of the 1.5D $O(n^2)$ n-body algorithm proposed by [61]. This algorithm is similar to the 2.5D matrix-matrix multiplication algorithm in that it replicates data c times across the processor grid to reduce the required amount of communication. Also like the 2.5D case, the maximal amount of data replication

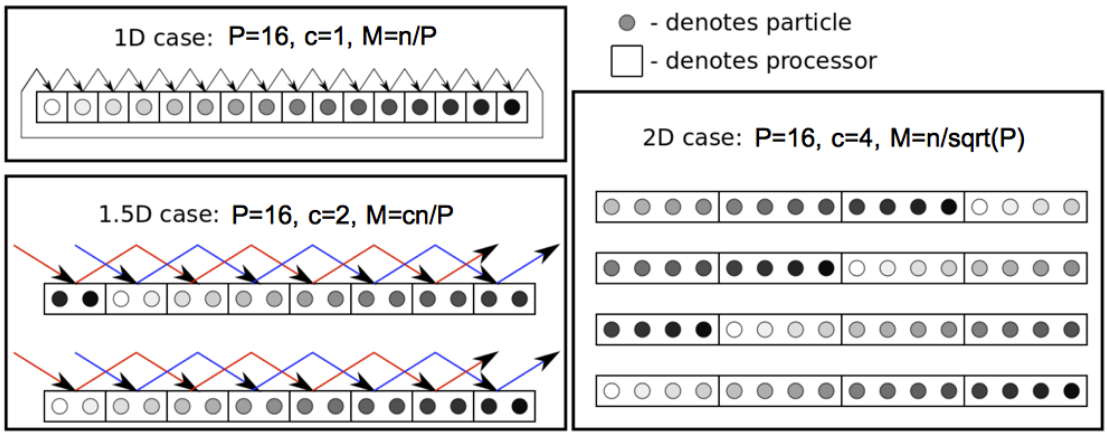


Figure 3.9: Data layouts for 1D, 1.5D and 2D n-body algorithms

is bounded by the replication overhead via a memory-independent per-processor communication lower bound (see [61] and Chapter 4 for details). The 1.5D algorithm is able to obtain speedups with a reduction in communication time, and parameterizes a spectrum of data replication between two existing approaches to solving the problem.

If there is no data replication (i.e. $c = 1$), each processor is allocated n/P particles and communication is between neighbors on a processor ring (see "1D case" in Figure 3.9). At the upper bound on replication ($c = P^{1/2}$), each group of $P^{1/2}$ can compute independently on a complete set of the input data. This requires no communication between processor groups. When $1 < c < P^{1/2}$, the processors are considered to be a (P/c)-by-c mesh wherein each group of P/c processors has an complete copy of the input data. The 1.5D n-body algorithm has been proven to asymptotically attain communication lower bounds and we will reiterate the results of Driscoll et al.[61] to argue that it is energy-optimal as well. As with the 2.5D matrix-matrix multiplication algorithm, this algorithm's ability to parameterize the tradeoff between additional memory and communication will allow us to analyze the impact of various constraints on the optimal amount of replication. Similarly to 2.5D matrix-matrix multiplication, the 1.5D n-body algorithm is able to strong scale perfectly in runtime with constant energy for a range of processors by replicating data.

In closing, we note that focusing on an $O(n^2)$ n-body is still insightful despite the existence of faster approaches. In practice, the all-to-all interaction of the bodies can often be approximated by only considering the interactions of bodies within some cutoff distance. Figure 3.10 illustrates the potential benefit of a cutoff distance. In the left portion of the figure, the grey body (p_i) must interact with each of the other four bodies in the system. On the right of the figure, p_i now only interacts with the bodies within its cutoff distance resulting in only two calls to applyForce(). This approximation is a simple modification to the $O(n^2)$ algorithm and can result in significant performance improvement without the overhead of implementing a tuned version of a more-efficient algorithm, such as FMM. Note that this calculation with cutoff is in fact the base of the FMM

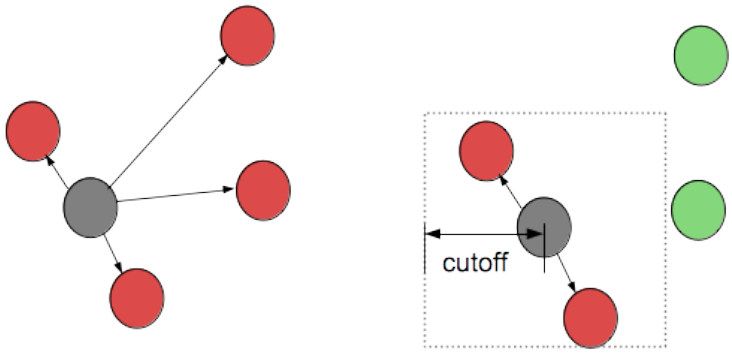


Figure 3.10: $O(n^2)$ n-body algorithm with and without a cutoff distance

divide-and-conquer algorithm: nearly particles are calculated via the direct n-body method. As noted within [61], the communication properties of the 1.5D algorithm are extensible to the use of a cutoff value and we believe that our lower bounds and analysis can be easily adapted to this situation.

3.4 Model Validation

In this section, we present evidence that the time and energy behavior of several computational kernels is accurately captured via the machine models described earlier in this chapter. In particular, we focus on the sequential and heterogeneous models and argue that modeling individual node behavior is a reasonable approach to extrapolating the efficiency of distributed parallel machines. As noted in earlier chapters, data and code for this section can be found at: `https://github.com/agearh/dissertation.git`.

Performance Counter Measurement

To validate our machine models, we wish to show that the models can reasonably fit energy and runtime. To do this on certain machines, we use a custom library named CounterHomeBrew (CHB) to measure the number of cache lines transferred between levels of cache on Intel machines. This library is designed to allow access to hardware performance counters that have not been implemented in production performance counter libraries, such as the Performance Application Programming Interface (PAPI)[112], perfmon2 [64] or Intel's Performance Counter Monitor (PCM) [147]. We note that this functionality is also exposed via the likwid suite [137], a set of tools that allow for access to performance counters on many Intel and AMD machines with very little modification to the benchmark source. We discovered problems with likwid when measuring cache misses on Sandy Bridge-EP, and thus used CHB in this work. As currently implemented, CHB is limited in functionality to Intel 7500 series and Sandy Bridge-EP servers, but may be extended in the future. To capture total machine data traffic at the various levels of cache, we enable

To Be Measured	Event	Mask	Comments
L1 to L2 Misses	0x51	0x01	Core Event, L1D replacement
L2 to L3 Misses	0xF1	0x07	Core Event, L2 lines in all
L3 to DRAM Misses	0x37	0xF	Cbo Event, filter=0x1F, LLC_VICTIMS

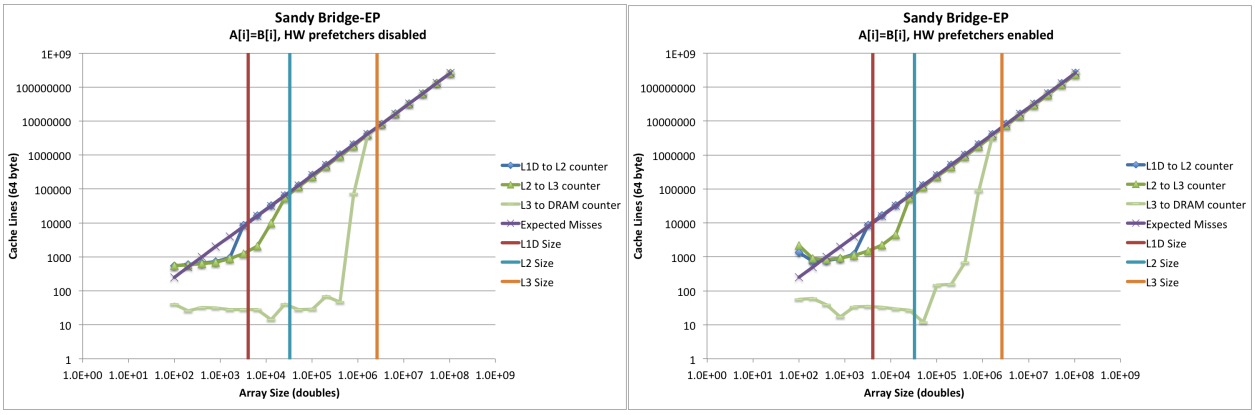
Table 3.2: Measuring Data Movement on Sandy Bridge-EP

counters on each core and last level Cache Box (CBo) and then sum the individual counters at the end of execution. We obtained counter information from the Intel 64 and IA-32 Architecture Software Developer Manuals [78] and the uncore programming guides for the Xeon 7500, E5-2600 and E7 families of processors [80, 81, 83].

Last-level cache misses on Intel 7500 series processors were successfully measured by Simhadri, Blelloch, Fineman, Gibbons and Kyrola [123] in their analysis of space-bounded schedulers. For data collected with our Xeon 7560 machine, we used CHB in a similar method to that used by Simhadri et al. As counting data traffic on Sandy Bridge-EP machines has not been validated via previous work, Table 3.2 shows the specific counters and events used to measure data traffic on these machines.

To evaluate the validity of the counters on Sandy Bridge-EP, we measured cache misses between each level of the device under test (DUT) and checked for correlation with a predicable benchmark, such as an array copy (i.e. $A[i] = B[i]$ for all $i = 0..m-1$). For example, Figure 3.11 shows a Sandy Bridge-EP machine running a benchmark that performs 10 copies of one array of doubles (8-bytes each) into another array using unit stride. The code was compiled using Intel's C compiler (ICC) with compiler flags set to forbid generation of streaming store instructions that bypass cache. In the figure, vertical lines delimit the sizes of the L1 Data, L2 and L3 caches on a single socket of this machine and the purple "Expected Misses" line indicates the expected number of misses for problems that do not obtain reuse in cache. For the 64-byte cache line sizes used on current Intel machines, this would entail a miss every eighth iteration per array (i.e. Expected Misses $= m/4$ for array copy, assuming no streaming stores). As expected, each of the three counters converges to the number of expected misses once the problem size has exceeded the given cache level being measured. Figure 3.11b also demonstrates that the counters preserve expected behavior in the presence of HW prefetcher activity. While the results of Figure 3.11 are for serial code, the same trend holds for the parallel situation wherein each core runs a sequential version of the benchmark.

We also noted that the floating point counters on Intel Sandy Bridge machines do not provide accurate measurements, even when running serial code. Figure 3.12 illustrates this problem for the various possible loop orderings of naive matrix-matrix multiplication without compiler optimizations (i.e. the '-O0' compiler flag). In the figure, we can see that four of the possible 6 loop orderings result in a large deviation from the expected (i.e. $2n^3$ for a dense, square classical matrix-matrix multiplication) number of measured flops. Due to the issues highlighted by Figure 3.12, results that require flop counts use theoretical values. This problem of counting floating point



(a) Array copy, no HW prefetchers (b) Array copy, with HW prefetchers

Figure 3.11: Counting cache misses during array copy on Sandy Bridge-EP

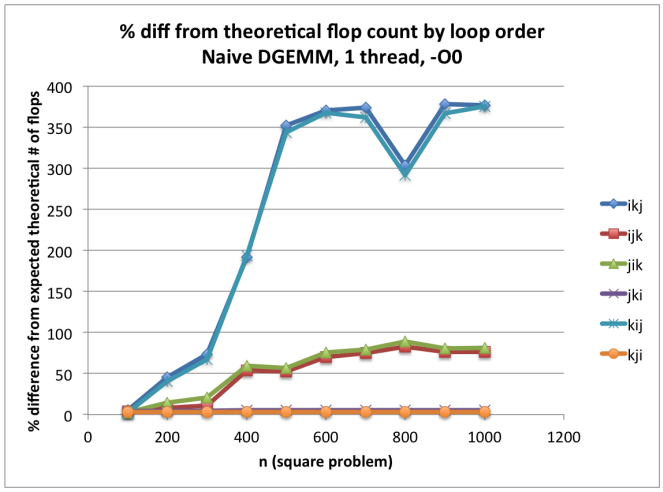


Figure 3.12: Inaccurate floating point operation counts on Sandy Bridge-EP

operations on Intel Sandy Bridge and Ivy Bridge processors has also been documented using PAPI [48].

Our Xeon 7560 machine is quad-socket (for 32 total physical cores) and has 128GB of installed DDR3-1660 DRAM. We used the Intel compiler v.14.0.2 and MKL v.11.1.2 for high-performance linear algebra functions (e.g. DGEMM or DGEMV), and the machine runs Ubuntu 12.04 with the 3.2.0-26 version of the linux kernel. Our dual-socket Sandy Bridge-EP testbed machine has two Xeon 2650 processors and 128GB of installed DDR3-1600 DRAM. On this machine, we used Intel compiler v.14.0.1.106 and MKL v.11.1.1 for tuned linear algebra functions. The Sandy Bridge-EP machine runs Ubuntu 12.04 with the 3.2.0-30 version of the linux kernel and GPU routines were accessed via version 5.5.0 of the CUDA Toolkit.

Measuring Power and Energy

In this work, we use direct measurement to obtain energy consumption data. In Section 2.2, we observed that wall power during execution of an application kernel remains relatively constant. With this in mind, may approximate kernel energy consumption by multiplying runtime and average power. We use one instrument for direct measurement of wall power data: a Wattsup Pro meter [144]. This meter is interposed between the test machine and the wall power outlet. The Wattsup Pro meter samples at a rate of 1Hz, and feeds these samples into a text file on the test machine for later analysis. Collecting wall power samples on the test machine itself does not appear to affect floating point throughput, and we assume that energy consumption is likewise unaffected. The advantage of this meter is that it is easy to move between machines, is low-cost and provides a simple interface for data collection. Unfortunately, measuring power at the wall includes power supply inefficiency (see Section 2.3) and the low sample rate of the Wattsup Pro meter requires experiments to be repeated many times for a reasonable power measurement to be obtained. In this work, we assume power supply inefficiency to be a small constant component of the machine's static power as described in Section 2.2. To obtain an average power measurement for the instruction mix representing a kernel, we use a script to align kernel timestamps and timestamps within the wall power output data[3]. Several examples of these alignments are shown in Figure 3.13. In the figure, we show (in blue) wall power traces for the repeated double precision sparse matrix vector multiplication (DSPMV) of 4 different sparse matrices from the University of Florida collection [50]. The names of the matrices are shown on each of the runs, and the data used for collection of average wall power is between the red intervals which delineate the start of a set of DSPMVs on the same input data. Note that as argued in Section 2.2, sparse matrix-vector multiplication shows relatively constant power during a run but power depends on the input data (i.e. for sparse problems, the flop/word ratio of the kernel depends on the structure of the input matrix).

Sequential Model (S)

We validate the sequential model (S) of time and energy, in Equations (3.1) and (3.2), respectively, by running several algorithms on the Intel Xeon 7560 and Sandy Bridge-EP machines and consider goodness of the fit obtained via a non-negative least squares solver. In the experiments that follow, we choose problem sizes that fit entirely in DRAM but are too large for the last-level cache (LLC). Thus, we model communication as traffic between a fast LLC and slow DRAM. Due to the low sample rate (1Hz) and precision (1/10W) of our wall power meter, we run benchmarks in parallel on all physical cores when attempting to fit node runtime and energy performance. Thus, we model the multicore testbed machines as if they were sequential and combine the performance of multiple

[3]When calculating the average power for a kernel, we drop the first three power meter samples from the time interval reported by the benchmark as we are attempting to capture a steady-state approximation of power consumed during the kernel. Benchmark and wall power timestamps were typically correlated to within 1-2 seconds (based on the alignment of an increase in wall power with the start of a series of kernels). This alignment uncertainly was significantly reduced by dropping the first few wall power samples in the timestamped interval.

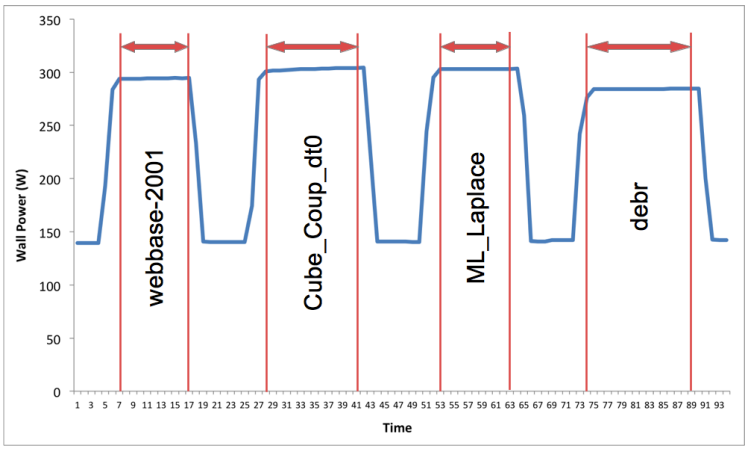


Figure 3.13: Typical wall power sample windows for several sparse matrix-vector multiplication problems

cores into a single abstract processor.[4] As we consider DRAM to be slow memory in this section, the size of fast memory (M) is the sum of LLCs on each socket of the testbed machines (e.g. the dual-socket Sandy Bridge-EP has a 20MB LLC per socket, for $M = 40MB$). Wall power data for all benchmarks was collected for at least 10 seconds, and all runs used double-precision arithmetic.

In the sequential model (S), we simplify the expressions for energy E_S in Equation (3.2) and runtime T_S in Equation (3.1) in two ways. First, on server nodes and desktops, "messages" are cache lines which are the number of words divided by some small constant, so we eliminate terms pertaining to messages (that include α_t and α_e). Second, as idle energy for the total installed amount of DRAM is included in wall power on current machines, we assume that the term representing idle memory energy ($\delta_e MT$) is part of the $\epsilon_e T$. The sequential runtime and energy models now become

$$T_S = \gamma_t F + \beta_t W \tag{3.13}$$

$$E_S = \gamma_e F + \beta_e W + \epsilon_e T_S \tag{3.14}$$

We note that E_S can also be considered in terms of the total power consumption, H_S:

$$E_S = T_S H_S = T_S(H_{S_{dyn}} + H_{S_{static}}) \equiv T_S \left(\frac{\gamma_e F + \beta_e W}{T_S} + \epsilon_e \right)$$

where $H_{S_{dyn}}$ and $H_{S_{static}}$ are the dynamic and static power components of wall power, respectively. Some additional observations about kernel power consumption make validation easier:

[4] This means that γ_t represents the time to perform $1/\#cores$ actual flops. This detail does not affect the following analysis.

1. As argued in Chapter 2, the literature suggests that energy consumption on server and desktop machines is dominated by three components: the processor package (execution units and SRAM-based caches), DRAM and static energies (e.g. disk, motherboard, etc.).

2. Arithmetic intensity also does not significantly vary with problem size within a given level of memory, unless the memory access pattern of the data structure is dependent on the input data, such as for sparse matrix computations (see Figure 3.14b).

3. The power required to compute an application phase does not vary significantly during execution (see Figures 2.3, 2.4, and Figure 3.14b). For theoretical intuition, consider the dynamic power of classical $O(n^3)$ matrix-matrix multiplication:

$$H_{dyn} = \frac{\gamma_e F + \beta_e W}{T_S} = \frac{\gamma_e F + \beta_e W}{\gamma_t F + \beta_t W} = \frac{\gamma_e F + \beta_e \frac{F}{M^{1/2}}}{\gamma_t F + \beta_t \frac{F}{M^{1/2}}} = \frac{\gamma_e + \frac{\beta_e}{M^{1/2}}}{\gamma_t + \frac{\beta_t}{M^{1/2}}}$$

   where we substitute $F = 2n^3$ (recall that F is the number of flops required to compute the matrix-matrix multiplication) and a lower bound on communication volume ($W = F/M^{1/2}$, see Chapter 4 for details). Note that F cancels out of the expression, leaving dynamic power equal to a ratio depending on γ_e,β_e,γ_t,β_e and M. As the amount of local memory M becomes larger, the dynamic power becomes increasingly close to the power γ_e/γ_e needed to compute a flop, assuming $n^2 > M$.

Fitting the Model via Least Squares

An intuitive approach to fitting the runtime and energy models of Equations (3.13) and (3.14) is to formulate it as a nonnegative linear least squares (NNLS) problem[5], and use a black box solver to fit the parameters (Matlab's *lsqnonneg*()). In detail, the NNLS problem is defined as

$$\underset{x \geq 0}{\operatorname{argmin}} \, ||Ax - b||_2^2$$

where A is an $k \times l$ matrix, b is a vector of length k and x is the solution vector of length l. We note that *lsqnonneg*() solves NNLS problems via a variant of the classic algorithm by Lawson and Hanson [96]. A so-called *active-set method*, this algorithm maintains two sets of variables: an active set of variables that violate the non-negativity constraint and a passive set of those that do not. The algorithm is iterative, with elements of the active set (the largest components of the negative gradient $A^T(b - Ax)$) selectively added to the passive set in an outermost loop. The thus-augmented passive set is then used to solve an unconstrained least squares problem. If the solution vector z to this unconstrained problem is nonnegative, the algorithm selects another member of the active set to include in the solution and repeats until the active set is empty or all components of the negative gradient are non-positive. However, if z includes non-positive values, we improve

[5]We use NNLS as negative values of runtime and energy parameters do not conform to physical intuition, e.g. a negative joule/flop (γ_e) cost is non-sensical.

solution vector x by $\eta(z - x)$ where $0 < \eta \leq 1$ is chosen to ensure the non-negativity of x. For further details, and a proof of convergence, see [96]. Several other researchers have suggested improvements to this algorithm [140, 36], including the addition of parallelism [104].

Regarding the problem at hand, we write the runtime and energy expressions of Equations (3.13) and (3.14) as NNLS problems in the following manner

$$\underset{x_t \geq 0}{\operatorname{argmin}} ||A_t x_t - b_t||_2^2 = \underset{x_t \geq 0}{\operatorname{argmin}} \left|\left| \begin{bmatrix} F_1 & W_1^* \\ F_2 & W_2^* \\ \vdots & \vdots \\ F_k & W_k^* \end{bmatrix} \begin{bmatrix} \gamma_t \\ \beta_t \end{bmatrix} - \begin{bmatrix} T_1^* \\ T_2^* \\ \vdots \\ T_k^* \end{bmatrix} \right|\right|_2^2 \tag{3.15}$$

$$\underset{x_e \geq 0}{\operatorname{argmin}} ||A_e x_e - b_e||_2^2 = \underset{x_e \geq 0}{\operatorname{argmin}} \left|\left| \begin{bmatrix} F_1 & W_1^* & T_1^* \\ F_2 & W_2^* & T_2^* \\ \vdots & \vdots & \vdots \\ F_k & W_k^* & T_k^* \end{bmatrix} \begin{bmatrix} \gamma_e \\ \beta_e \\ \epsilon_e \end{bmatrix} - \begin{bmatrix} H_1^* T_1^* \\ H_2^* T_2^* \\ \vdots \\ H_k^* T_k^* \end{bmatrix} \right|\right|_2^2 \tag{3.16}$$

where nonnegativity constraints ($x_t \geq 0$ and $x_e \geq 0$) are imposed. Each row of the matrices represents measurements from an individual trial run of the implementation to be fitted (or a set of suitable microbenchmarks that approximate the instruction mix of the targeted kernel, as will be discussed in Section 3.5). Further, variables with an asterisk in Equations (3.15) and (3.16) represent experimentally-determined values. We approximate the energy consumption of a run by multiplying the measured runtime T_i^* by the average wall power H_i^*, implicitly assuming that the deviation of wall power within a run is small. This assumption appears to be supported by the data, as discussed in Section 2.2. Using wall power to estimate energy allows for the use of cost-effective monitoring equipment, and as we argued in Section 2.2, this represents a reasonable approximation of energy consumption due to the phase-based behavior of kernels. Further, due to the limitations of counting flops on recent Intel machines (see Section 3.4 for details) we used theoretical values for the number of executed flops instead of measured values, e.g. $F = 2n^3$ for dense matrix-matrix multiplication on n-by-n matrices.

We solve the NNLS problems of Equations (3.15) and (3.16) by running the target kernel implementation for various problem sizes that fit within a targeted level of the memory hierarchy (DRAM, in this instance). For example, one may run successively larger dense matrix-matrix multiplication problems and measure the runtime, amount of transferred data and average wall power to generate A_t, A_e, b_t and b_e. Table 3.3 shows such data on a dual socket Sandy Bridge-EP server for a naive implementation of dense double-precision $O(n^3)$ matrix-matrix multiplication (see Algorithm 3) with matrix dimensions varying from $n = 1400$ to $n = 3000$. This range of problem sizes occupies between approximately 45 and 206 MB of main memory, where the total size of the last level caches on this machine is 40MB. Note that after $n = 2000$, the flop/word ratio of the computation becomes relatively constant.[6] This supports our hypothesis of program phase

[6] A minimal flop/word ratio for sequential naive matrix-matrix multiplication would be 1 assuming reads of A_{ik}

behavior, with the observation that larger problem sizes (beyond the fast memory size) are needed to eliminate residual caching effects. This latter observation is also observed in the behavior of sparse matrix-vector computations. In Figure 3.14, we see F/W^* ratios for a set of several hundred sparse matrices ordered by the memory footprint of the CSR representation of the problem. Note that the optimal F/W^* ratio of 4/3 is exceeded for many smaller problems, despite the matrix technically being larger than cache.[7] In the results that follow, we use problems with memory footprints larger than 50MB to generate reasonable NNLS fits. The data in Table 3.3 and Figure 3.14 suggest that further accuracy may be attained with footprints larger than 60-70MB on this particular machine.

n	F	W^*	F/W^*	T^*	H^*T^*
1400	5.49E+009	5.07E+006	1081.80	22.89	5587.3
1600	8.19E+009	3.11E+008	26.35	44.86	10979.1
1800	1.17E+010	9.60E+008	12.15	74.16	20944.2
2000	1.60E+010	1.80E+009	8.88	110.78	28257.1
2200	2.13E+010	2.61E+009	8.17	154.72	39304.6
2400	2.76E+010	3.35E+009	8.25	179.14	46189.1
2600	3.52E+010	4.08E+009	8.62	212.10	54717.0
2800	4.39E+010	4.80E+009	9.12	245.06	63244.9
3000	5.40E+010	5.53E+009	9.77	278.02	71772.8

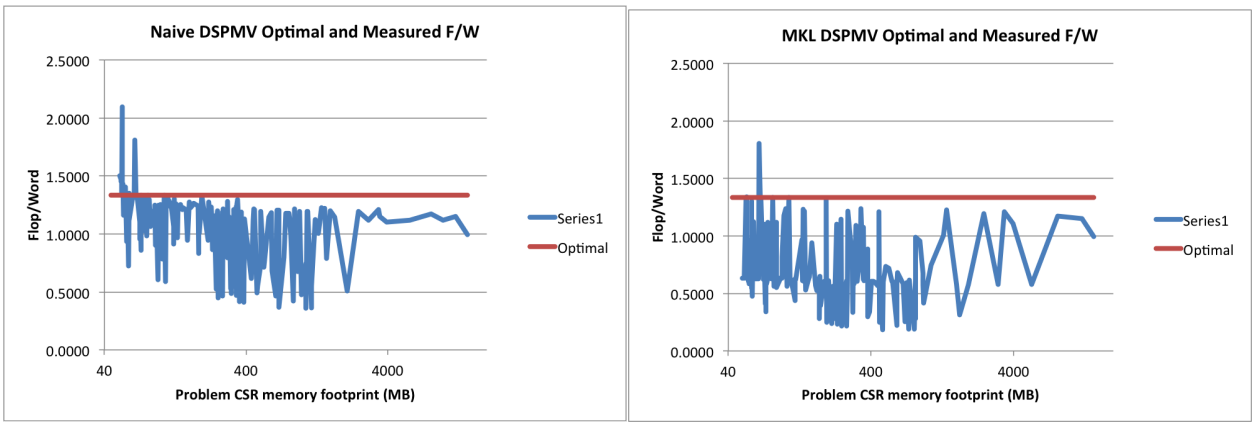
Table 3.3: Sandy Bridge-EP: Arithmetic intensity (flop/word) for naive matrix-matrix multiplication

With proper scaling of the columns of A_t and A_e to reduce the condition numbers, this method is effective at generating quality fits of runtime and energy. Average percent error values for NNLS fits across a range of problem sizes[8] with only column scaling are shown for a Sandy Bridge-EP machine in Table 3.4 and for a Xeon 7560 machine in Table 3.5. Error was calculated as the average relative difference in percent between the models and measured runtime and energy data. Models used fitted parameters, theoretical values of F, measured counts of last-level cache misses converted to words for W^*, and measured average power H^*. For the energy model, measured runtime was used as opposed to the runtime model. In the case of double-precision sparse matrix-vector multiplication (DSPMV), a number of matrices from the University of Florida Sparse Matrix Collection [50] were run for a number of iterations sufficient to obtain useful power measurements (> 10 seconds). We suspect that larger sizes than 50MB would result in a substantially better fit on this machine. The higher runtime error error value for naive DGEMM on the Xeon 7560 machine

and B_{kj} and use of a register to accumulate C_{ij}, but this code attains around 9 due to parallelism (OpenMP) and compiler optimizations (Intel C compiler, "-O3" optimization flag).

[7] Assuming the two vectors fit completely in cache, we only need to load one word of the matrix and the index array (with 4-byte integer elements, so 1/2 a word) each iteration.

[8] See spreadsheets in the git repo for this dissertation, `https://github.com/agearh/dissertation.git`, for details.



(a) Naive DSPMV (b) MKL DSPMV

Figure 3.14: Sandy Bridge-EP: Flop/Word ratios for double-precision sparse matrix-vector multiplication (DSPMV)

is due to a single run (N=3900) that required 103 seconds to run (for comparison, running N=4000 required 55 seconds). This effect persisted even when averaged over five iterations. The similarly large error with naive DGEMM on the Sandy Bridge-EP machine was likely due to performance variability at smaller problem sizes. For all benchmarks on the Sandy Bridge-EP machine, the NNLS formulation of Equations (3.15) and (3.16) resulted in average errors of less than 25%. The Xeon 7560 machine obtained similar results, with the exception of DSPMV. We believe that runtime is much more sensitive to matrix structure than energy for sparse problems, perhaps due to the large static energy component on this this machine. If we attempted to fit repeated runs of a single class of matrix structure (e.g. a set of diagonal matrices of increasing size), this issue may be mitigated.

As a side note, scaling the rows of A_t and A_e (i.e. weighted least squares) on the Sandy Bridge-EP machine by functions of F (see Table 3.6 for results when rows are scaled by $1/F$) resulted in little improvement in accuracy and often significantly worse performance (e.g. naive DSPMV).[9] We also attempted to fit parameters for the Sandy Bridge-EP machine using combined data from multiple benchmarks as well as with a least-squares solver without non negativity constraints. These alternate fits did not result in accuracy improvements, and in many cases significantly decreased accuracy (see the git repo for this dissertation for more details). Note that these results evaluate the quality of fit to the training data, and not the predictive capability of the model. One would expect the error to increase if compared with non-training data, especially if the set of training data does not capture the variability of machine behavior.

The fitted parameters for the NNLS problems without row scaling are shown in Tables 3.7 (Sandy Bridge-EP) and 3.9 (Xeon 7560). While the overall fit of runtime and energy data is

[9]Evaluated row scaling functions of F included $1/F$ for all kernels. Other functions were also evaluated: DGEMM, $1/F^{2/3}$; DGEMV, $1/F^{1/2}$; DSPMV, $2/F$ and $1/\text{memSize}$ where memSize is the size of the CSR matrix in words.

	Runtime % Error	Energy % Error
MKL DGEMM	1.82	0.54
Naive DGEMM	13.29	1.82
MKL DGEMV	1.00	2.00
Naive DGEMV	4.53	2.42
MKL DSPMV	22.59	1.39
Naive DSPMV	24.33	1.58

Table 3.4: Sandy Bridge-EP: Average runtime and energy % error (no row scaling)

	Runtime % Error	Energy % Error
MKL DGEMM	2.68	0.60
Naive DGEMM	20.41	0.79
MKL DGEMV	0.99	0.19
Naive DGEMV	12.50	0.15
MKL DSPMV	35.15	1.71
Naive DSPMV	40.42	2.52

Table 3.5: Xeon 7560: Average runtime and energy % error (no row scaling)

	Runtime % Error	Energy % Error
MKL DGEMM	1.94	0.30
Naive DGEMM	7.16	1.75
MKL DGEMV	0.82	2.38
Naive DGEMV	5.36	0.96
MKL DSPMV	31.61	1.75
Naive DSPMV	43.67	2.26

Table 3.6: Sandy Bridge-EP: Average runtime and energy % error (1/F row scaling)

reasonably accurate, these benchmarks are not an effective way to approximate model parameters for a given problem implementation. This is evidenced by several parameters fitting to zero. We believe this to be due to the fact that A_t and A_e are often low rank due to the relatively constant ratio of F to W^* when the NNLS problems are formulated by running multiple problem sizes of the same code implementation. We discuss this issue further later in the section. In Tables 3.8 and 3.10, we compare modeled floating point (Gflop/s, via γ_t) and runtime (GB/s, via β_t) throughputs with the average of measured values for the range of problem sizes used to fit the model. For each of the benchmarks, one would expect parameters for the benchmark's dominant throughput-limiting operation (floating point operations or memory traffic) to be fitted more accurately. Of the benchmarks, only MKL DGEMM is bounded by the machine's floating point capability, while the other codes are limited by DRAM bandwidth. Indeed, on the Sandy Bridge-EP test machine (Table

3.8), we observe that the modeled floating point throughput (1e-9/γ_t, Gflop/s) is significantly closer to the measured value than the modeled bandwidth (8e-9/β_t, GB/s). A similar trend can be observed with naive DGEMM and both implementations of DGEMV and DSPMV. In Figures 3.15, 3.16 and 3.17, we explore the relationship between components of the fitted models and highlight points of intersection and divergence from expected behavior with regard to benchmark characteristics on the Sandy Bridge-EP experimental machine. We define $T_{comp} = \gamma_t F$, $T_{comm} = \beta_t W^*$, $E_{comp} = \gamma_e F$, $E_{comm} = \beta_e W^*$ and $E_{idle} = \epsilon_e T^*$. Note that these are modeled values. We further define T_{actual} and E_{actual} represent measured values of runtime and energy (with energy based on multiplying runtime by a measured average power, as discussed previously). Thus, is the model were error-free, $T_{actual} = T_{comp} + T_{comm}$ and $E_{actual} = E_{comp} + E_{comm} + E_{idle}$. Later, in Section 3.5, we discuss related work that attempts to construct benchmarks to calculate these parameters accurately.

Parameter	γ_t	β_t	γ_e	β_e	ϵ_e
MKL DGEMM	3.03E-12	4.48E-10	4.57E-11	4.81E-09	277.16
Naive DGEMM	2.23E-09	3.24E-08	1.16E-08	1.89E-07	252.32
MKL DGEMV	1.47E-11	2.03E-10	0	0	250.50
Naive DGEMV	9.51E-11	1.43E-10	1.75E-08	2.68E-08	97.77
MKL DSPMV	3.32E-11	1.68E-10	0	1.33E-08	220.04
Naive DSPMV	0	2.03E-10	0	1.30E-08	223.55

Table 3.7: Sandy Bridge-EP: Fitted sequential machine parameters without row scaling

	Modeled Gflop/s	Measured Gflop/s	Modeled GB/s	Measured GB/s
MKL DGEMM	330.05	228.91	17.84	5.46
Naive DGEMM	0.45	0.17	0.25	0.12
MKL DGEMV	68.07	8.17	39.41	34.53
Naive DGEMV	10.51	5.91	56.10	25.72
MKL DSPMV	30.11	3.84	47.69	39.99
Naive DSPMV	undefined	5.81	39.43	39.37

Table 3.8: Sandy Bridge-EP: Modeled vs. Measured Runtime Throughputs

In Figure 3.15, we visualize the relationships between modeled computation and communication in runtime and energy for DGEMM on the Sandy Bridge-EP machine. The energy model also includes idle energy, and the model parameters are from Table 3.7. The vertical axes are logarithmic, and the horizontal axis is linear in problem size. The optimized implementation of DGEMM (Intel's MKL library) is represented in the upper subplots, and demonstrates a clear dominance of computation over communication. Idle energy dominates overall, which is interesting due to

Parameter	γ_t	β_t	γ_e	β_e	ϵ_e
MKL DGEMM	2.28E-12	5.14E-10	7.49E-10	3.79E-08	872.67
Naive DGEMM	0	2.82E-10	3.69E-08	5.34E-09	871.96
MKL DGEMV	2.74E-10	0	6.86E-09	0	810.00
Naive DGEMV	0	2.67E-10	5.51E-09	2.91E-08	806.41
MKL DSPMV	4.10E-11	1.72E-10	0	1.22E-08	895.98
Naive DSPMV	0	2.83E-10	3.91E-09	6.84E-09	893.33

Table 3.9: Xeon 7560: Fitted sequential machine parameters without row scaling

	Modeled Gflop/s	Measured Gflop/s	Modeled GB/s	Measured GB/s
MKL DGEMM	438.29	254.83	15.57	6.4
Naive DGEMM	undefined	2.63	28.35	31.34
MKL DGEMV	3.65	3.66	undefined	15
Naive DGEMV	undefined	7.73	29.91	33.24
MKL DSPMV	24.37	6.27	46.43	40.04
Naive DSPMV	undefined	5.16	28.23	31.73

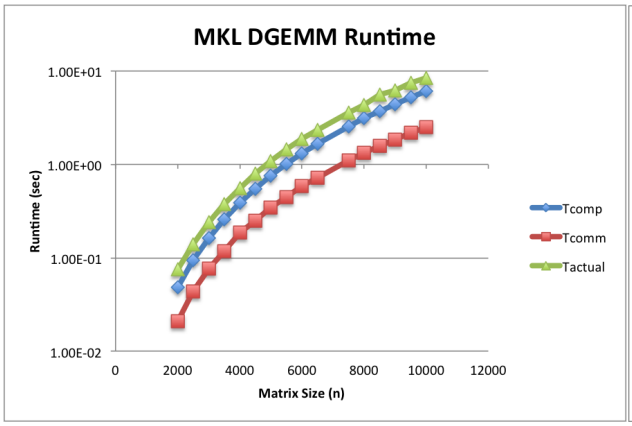
Table 3.10: Xeon 7560: Modeled vs. Measured Runtime Throughputs

the high intensity of the implementation. This is expected from a highly-optimized algorithm that attains a high fraction of peak machine performance. The lower two subplots represent the naive implementation of DGEMM. With runtime, communication dominates to the point of the γ_t being fitted to zero.[10] In the energy case, communication dominates computation (as expected from a naive algorithm), but to a lesser extent than expected. Again, idle energy ($\epsilon_e T$, or E_{idle}), dominates. Note that modeled data T_{comm} and E_{comm} drop quickly for small problem sizes. We believe this is due to cache effects causing a significantly reduced number of cache misses for smaller problems.

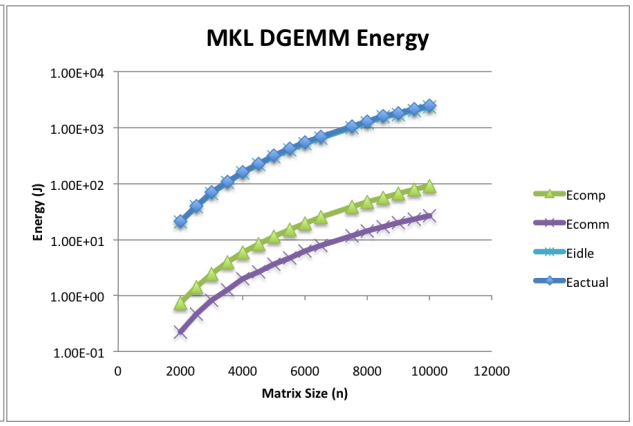
Dense double-precision matrix-vector multiplication (DGEMV, Figure 3.16) shows a clear dominance of communication runtime and idle energy. The former is to be expected from an extremely communication-bound algorithm. The lack of a dynamic energy component of the MKL implementation's energy model is surprising, especially as the model fits the data well (with 2% average error). The naive implementation shows a nearly identical relationship between the model components, with regards to both runtime and energy. One suspects that more-targeted benchmarks are needed to tease apart the relationships between model parameters in this situation.

Finally, Figure 3.17 displays runtime and energy components for both tuned and naive implementations of DSPMV. For these plots, the horizontal axis represents the memory footprint of the CSR representation of the matrix-vector multiplication problem, as opposed to the matrix size as

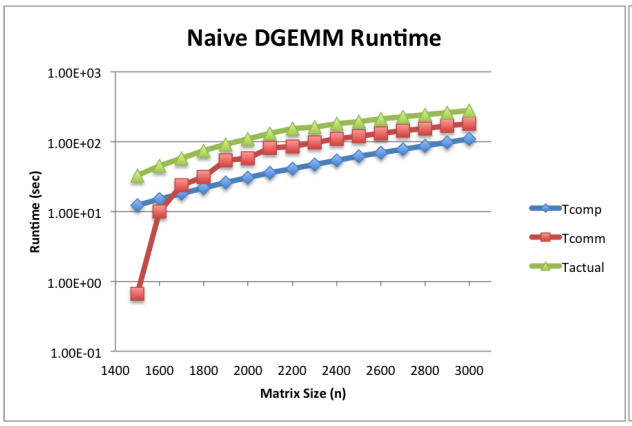
[10] We include zero-value runtime or energy components in the legend to highlight their absence.



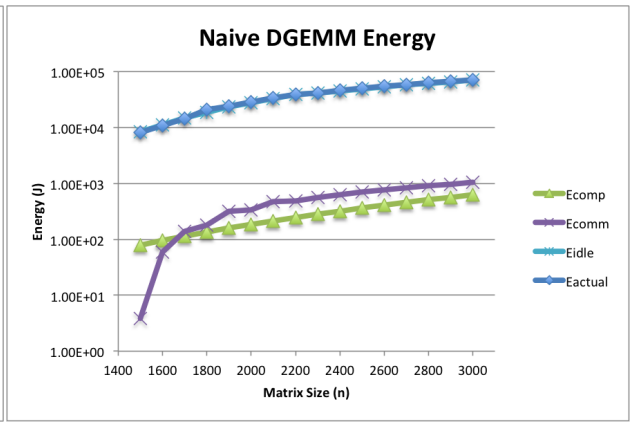
(a) MKL DGEMM Modeled Runtime



(b) MKL DGEMM Modeled Energy



(c) Naive DGEMM Modeled Runtime

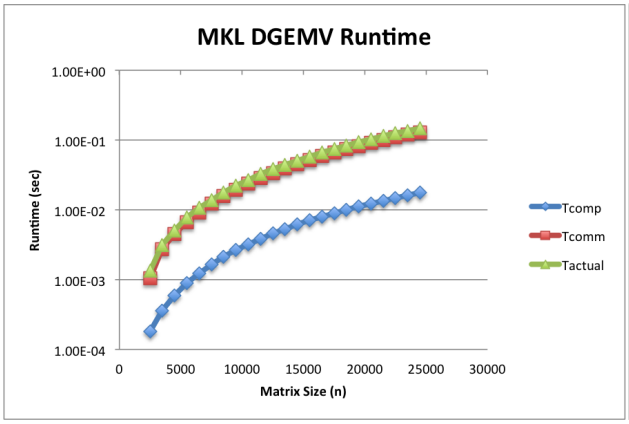


(d) Naive DGEMM Modeled Energy

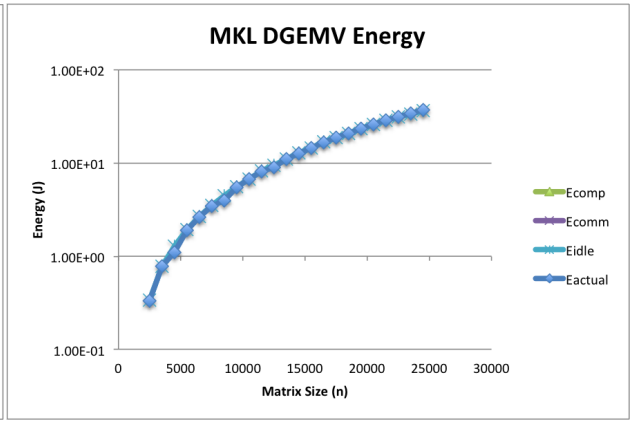
Figure 3.15: Sandy Bridge-EP: Modeled (no row scaling) double-precision matrix-matrix multiplication (DGEMM)

used in the previous two sets of figures. In both naive and tuned DSPMV, communication costs dominate the runtime models. This is to be expected for an algorithm that is performance limited by communication. Idle energy dominates for both the MKL and naive implementations, with a smaller communication component. Computation energy fits to zero, not unexpected for such a communication-dominated algorithm.

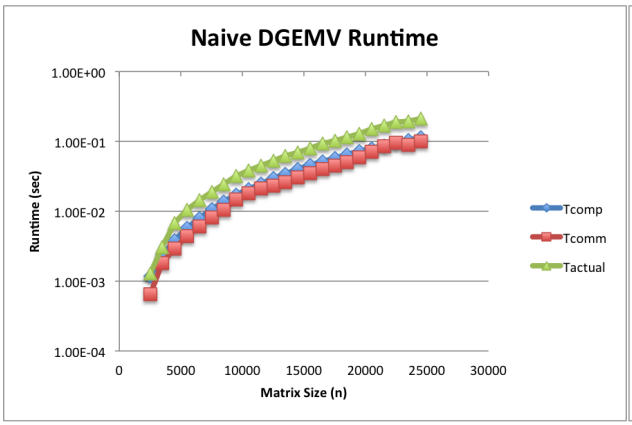
The fitted runtime parameters on the Xeon 7560 machine strongly support the hypothesis that parameters related to the underlying thoughput-limiting operation (flops or memory operations) will be fitted more accurately, with one notable aberration. In Table 3.9, we see that the modeled floating point throughput for MKL DGEMM (438.29 Gflop/s) compares reasonably well with the average measured value of 254.83 Gflop/s. Similarly, modeled bandwidth for Naive DGEMM, Naive DGEMV, and both implementations of DSPMV compare well with the measured values. Interestingly, MKL DGEMV fits the measured floating point rate nearly exactly (3.65 Gflop/s



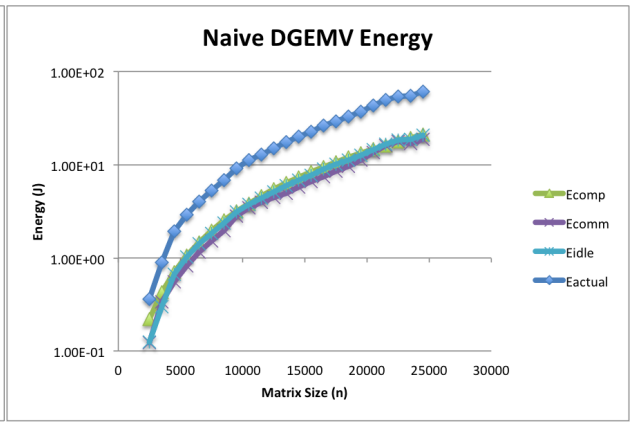
(a) MKL DGEMV Modeled Runtime



(b) MKL DGEMV Modeled Energy



(c) Naive DGEMV Modeled Runtime

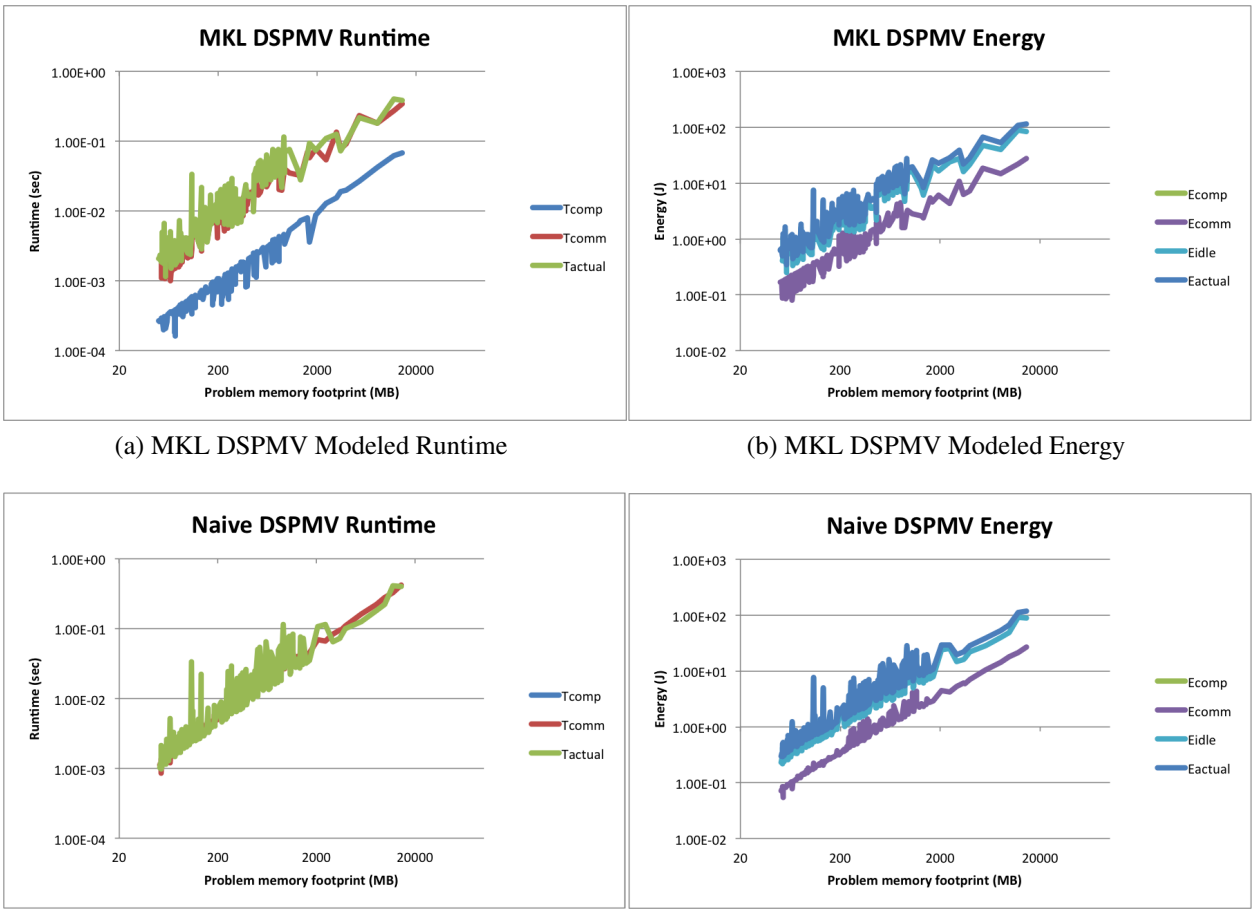


(d) Naive DGEMV Modeled Energy

Figure 3.16: Sandy Bridge-EP: Modeled (no row scaling) double-precision dense matrix-vector multiplication (DGEMV)

vs. 3.66 Gflop/s), but fits the bandwidth parameter (β_t) to zero. This code is intuitively limited by communication bandwidth, so this result is unexpected. While we don't provide a detailed discussion of the fitted parameters on this machine (unlike the Sandy Bridge machine, above), we note that the fitted values of ϵ_e are relatively consistent across benchmarks, which is a positive sign.

As noted earlier, a probable reason for our inability to accurately fit parameters is likely due to the relatively constant ratio of F/W^* in our construction of the NNLS problems. This results in A_t and A_e matrices that are low rank, and the NNLS solver is unable to properly differentiate γ and β terms. Indeed, when we only attempt to fit γ_t, γ_e and ϵ_e (dropping β_t and β_e) with MKL DGEMM and β_t, β_e, and ϵ_e with naive DGEMM, DGEMV and DSPMV, quality of fit for all implementations decreases little (with the exception of naive DGEMM, see Tables 3.11 and 3.12). In Table 3.13, we note that the runtime throughputs (Gflop/s and GB/s) agree almost exactly with



(a) MKL DSPMV Modeled Runtime

(b) MKL DSPMV Modeled Energy

(c) Naive DSPMV Modeled Runtime

(d) Naive DSPMV Modeled Energy

Figure 3.17: Sandy Bridge-EP: Modeled (no row scaling) double-precision sparse matrix-vector multiplication (DSPMV)

measured values (again with the exception of naive DGEMM).[11] The problem with naive DGEMM may be due to cache effects for small problems, as noted previously. Carefully constructing A_t and A_e as combinations of implementations (with proper row and column scaling) may be the correct approach to mitigate this issue, but our limited exploration of such fits did not yield better parameters (see analysis in the git repository associated with this work).

The NNLS fit results for DGEMM, DGEMV and DSPMV suggest that the runtime and energy behavior of these benchmarks can be accurately encapsulated via models with only a handful of parameters. However, we demonstrate that running implementations of these benchmarks for different problem sizes is not sufficient to attain accurate values of the model parameters themselves. This suggests a need for specialized benchmarks to isolate these performance parameters, as will be discussed in Section 3.5.

[11] We row scale MKL DGEMM by $1/F$ and the other implementations by $1/W^*$.

	Runtime % Error	Energy % Error
MKL DGEMM	2.11	0.30
Naive DGEMM	167.74	1.66
MKL DGEMV	1.25	2.30
Naive DGEMV	6.19	0.97
MKL DSPMV	32.35	1.58
Naive DSPMV	32.84	1.86

Table 3.11: Sandy Bridge-EP: Average runtime and energy % error when non-dominant terms are dropped from model (with row scaling)

Parameter	γ_t	β_t	γ_e	β_e	ϵ_e
MKL DGEMM	4.37E-12	N/A	3.34E-10	N/A	213.23
Naive DGEMM	N/A	1.60E-07	N/A	7.10E-07	245.45
MKL DGEMV	N/A	2.32E-10	N/A	3.32E-10	244.80
Naive DGEMV	N/A	3.14E-10	N/A	1.12E-08	246.62
MKL DSPMV	N/A	2.34E-10	N/A	1.40E-08	218.61
Naive DSPMV	N/A	2.34E-10	N/A	1.42E-08	219.03

Table 3.12: Sandy Bridge-EP: Fitted sequential machine parameters when non-dominant terms are dropped from model (with row scaling)

	Modeled Gflop/s	Measured Gflop/s	Modeled GB/s	Measured GB/s
MKL DGEMM	228.75	228.91	N/A	N/A
Naive DGEMM	N/A	N/A	0.05	0.12
MKL DGEMV	N/A	N/A	34.51	34.53
Naive DGEMV	N/A	N/A	25.45	25.72
MKL DSPMV	N/A	N/A	34.25	39.99
Naive DSPMV	N/A	N/A	34.13	39.37

Table 3.13: Sandy Bridge-EP: Modeled vs. measured runtime throughputs when non-dominant terms are dropped from model (with row scaling)

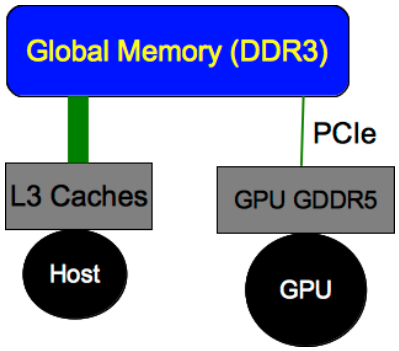


Figure 3.18: Heterogeneous machine for validation

Distributed Parallel Models

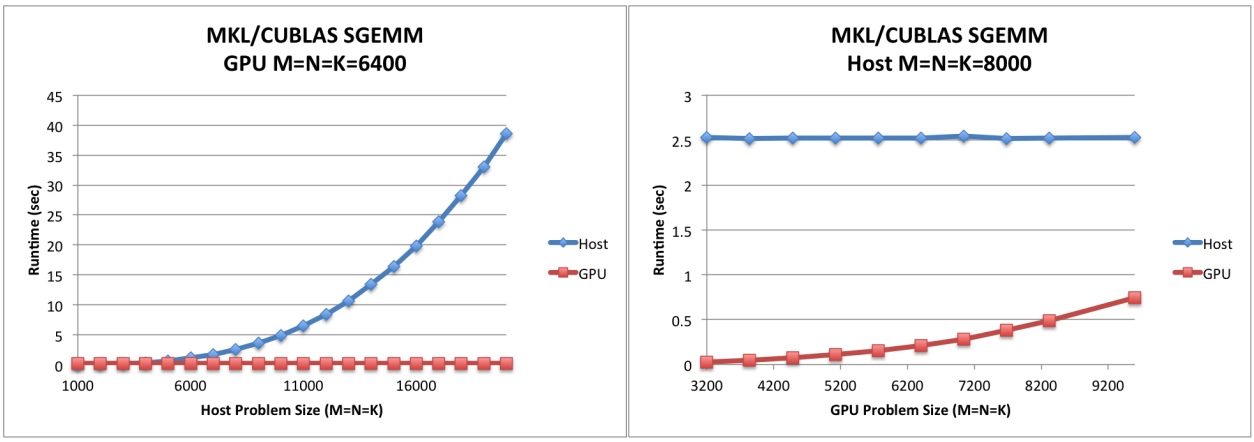
Due to limitations of our current measurement equipment, we were unable to validate the distributed parallel machine models in a manner similar to the sequential analysis discussed in the previous section. In the case of the DP2 model, a validated sequential model (with the reasonable assumption that network energy scales with the number of links) is most likely sufficient to extend our analysis to a distributed parallel machine. Due to the construction of DP1, we assume that communication energy scales proportionally with load and could calculate communication parameters from industry sources and the research of others. For the validated sequential model to be useful, we must also assume an asymptotically equal distribution of work (and input/output data) between the processing nodes. Within these assumptions, we believe that it is reasonable to make arguments about larger distributed machines. Further, as discussed in Chapter 2, the dominant communication energy on distributed machines may be due to intranode cache to DRAM transfers and not internode communication. This would make the inclusion of DP2 especially relevant to current network interconnect technologies.

Heterogeneous Model

In this section, we demonstrate that the heterogeneous model proposed earlier in this chapter is capable of representing the runtime and energy costs of a pair of heterogeneous benchmarks. These benchmarks target a specific subset of heterogeneous platforms: *host* desktops or servers with an NVIDIA GPU (Graphics Processing Unit) attached via a PCIe bus.[12] Figure 3.18 depicts our validation platform; two heterogeneous processors (a Sandy Bridge-EP multi-socket host server, and an attached GPU), with host DRAM as global memory. On the host, as with the sequential model analyzed earlier, we model communication between last-level caches and DRAM. Communication between the GPU and global memory is modeled to be over a PCIe bus.

The benchmarks themselves are of form similar to Algorithm 7, and complete a series of dense single-precision matrix-matrix (SGEMM) or matrix-vector multiplications (SGEMV) in parallel

[12]We are limited to NVIDIA devices due to the benchmark's use of NVIDIA's CUDA library, version 5.5. An OpenCL implementation would allow for wider portability, with tuned BLAS functions via AMDs clMath library [6]



(a) Scaling Host communication and computation

(b) Scaling GPU communication and computation

Figure 3.19: Runtime impact of scaling either Host or GPU SGEMM size

on the host machine and the GPU (also referred to as the *device*). Two threads are initially allocated for this task, with the first pinned to core 0 and dedicated to submitting work to the GPU. The second thread handles host work, and starts a parallel matrix multiplication that runs on all cores except core 0 (i.e. 15 cores of the 16-core host perform repeated multithreaded SGEMM or SGEMV operations). The structure for Algorithm 7 allows for ab initio knowledge of the communication volume across the PCIe link, avoiding limitations of Intel's performance counter infrastructure with regard to measuring PCIe traffic. In Figure 3.19, we can see that increasing the host's problem size while fixing the GPU workload does not increase the runtime of the other processor. In the right-hand subfigure, we can see the opposite is also true: increasing the amount of GPU work does not affect the runtime of a fixed amount of host computation. As both share a common set of memory controllers to access host DRAM, this may be due to the fact that an efficient matrix-matrix multiplication implementation does not saturate the available bandwidth. Surprisingly, in the case of matrix-vector multiplication (Figure 3.20) we see a similar performance independence between host and GPU, despite a communication-intensive algorithm. This may be due to architectural features, such as direct-memory access (DMA) copies between host and device that bypass the cache hierarchy. The results in Figures 3.19 and 3.20 support the assumption of independent communication links that is inherent to our heterogeneous machine model.

As in validation of the sequential model, we assume that communication latency is zero and eliminate terms that include α_{t_i} and α_{e_i} from the model. Similarly, we assume that DRAM idle power does not vary with the amount of memory used by the problem and thus the $\hat{\delta}\hat{M}T_H$ term in the heterogeneous energy model is merged with the static energy term, $\epsilon_H T_H$. The heterogenous runtime and energy expressions of Equations (3.7) and (3.8) now become

$$T_H = \max\left(\gamma_{t_1} F_1 + \beta_{t_1} W_1, \gamma_{t_2} F_2 + \beta_{t_2} W_2\right) \tag{3.17}$$

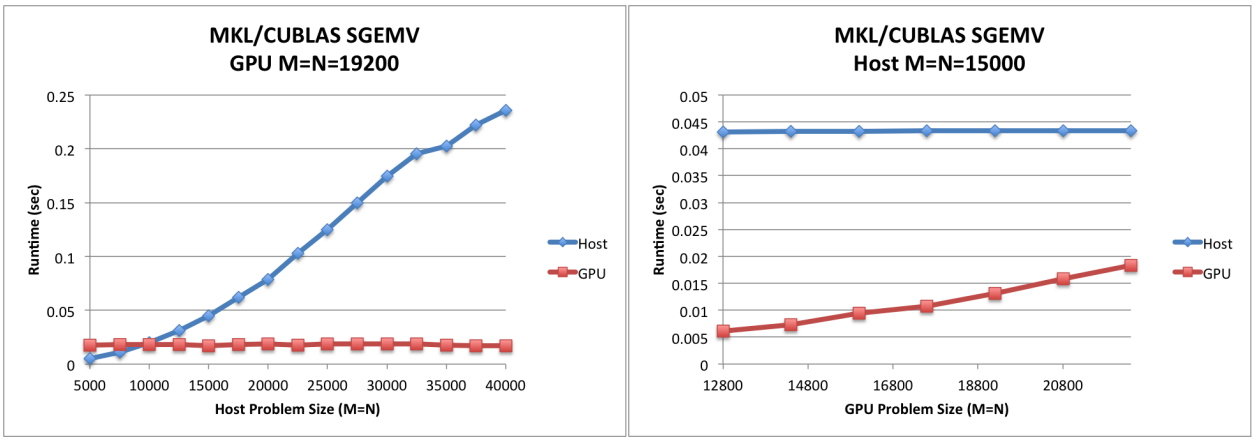
Algorithm 7 Benchmark for validation of heterogeneous model

Require: Host machine with NVIDIA GPGPU, supporting CUDA 5.5+
Require: $gpuInner$, $gpuOuter$, $hostInner$ and problem sizes for host (n_{cpu}) and GPU (n_{gpu}) devices
Begin Parallel Section with 2 threads
if THREAD == 0 **then**
  Allocate device memory (Ad,Bd,Cd) and pinned host memory (Ah,Bh,Ch)
  Initialize Ah and Bh
  for $i < gpuOuter$ **do**
    Asynchronously copy Ah, Bh into Ad and Bd
    for $j < gpuInner$ **do**
      Compute $MULTIPLY(Ad, Bd, Cd)$ on device
    end for
    Asynchronously copy Cd into Ch
  end for
  Synchronize host thread 0 with device
  Deallocate Ad,Bd,Cd,Ah,Bh,Ch
else
  Allocate host memory (A,B,C)
  Initialize A and B
  for $i < hostInner$ **do**
    Compute $MULTIPLY(A, B, C)$ on host with NCORES-1 threads
  end for
  Deallocate A,B,C
end if
End Parallel Section

$$E_H = \gamma_{e_1} F_1 + \beta_{e_1} W_1 + \gamma_{e_2} F_2 + \beta_{e_2} W_2 + \epsilon_H T_H. \tag{3.18}$$

To validate the heterogeneous model for these benchmarks, we first write the runtime and energy expressions as NNLS problems in a similar manner to the sequential model:

$$\underset{x_{t_1} \geq 0}{\operatorname{argmin}} ||A_{t_1} x_{t_1} - b_{t_1}||_2^2 = \underset{x_{t_1} \geq 0}{\operatorname{argmin}} \left|\left| \begin{bmatrix} F_{11} & W_{11}^* \\ F_{12} & W_{12}^* \\ \vdots & \vdots \\ F_{1k} & W_{1k}^* \end{bmatrix} \begin{bmatrix} \gamma_{t_1} \\ \beta_{t_1} \end{bmatrix} - \begin{bmatrix} T_{11}^* \\ T_{12}^* \\ \vdots \\ T_{1k}^* \end{bmatrix} \right|\right|_2^2 \tag{3.19}$$



(a) Scaling Host communication and computation

(b) Scaling GPU communication and computation

Figure 3.20: Runtime impact of scaling either Host or GPU SGEMV size

$$\underset{x_e \geq 0}{\operatorname{argmin}} ||A_e x_e - b_e||_2^2 = \underset{x_e \geq 0}{\operatorname{argmin}} \left|\left| \begin{bmatrix} F_{11} & W_{11}^* & F_{21} & W_{21} & T_{H_0}^* \\ F_{12} & W_{12}^* & F_{22} & W_{22} & T_{H_1}^* \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ F_{1k} & W_{1k}^* & F_{2k} & W_{2k} & T_{H_k}^* \end{bmatrix} \begin{bmatrix} \gamma_{e_1} \\ \beta_{e_1} \\ \gamma_{e_2} \\ \beta_{e_2} \\ \epsilon_H \end{bmatrix} - \begin{bmatrix} E_1^* \\ E_2^* \\ \vdots \\ E_k^* \end{bmatrix} \right|\right|_2^2 \tag{3.20}$$

where nonnegativity constraints are imposed on each element of x_{t_1} and x_e. Double subscripts represent the processor number and the experimental run (e.g. F_{12} is the number of flops performed by the first processor on during the second experimental run). We also fit runtime parameters for both processors independently (as the model assumes independent communication links), and only show the NNLS problem to calculate parameters of $T_{H_1}^* = max(T_{1i}^*, T_{2i}^*)$ in Equation (3.19), above. The matrix expression of the problem for x_{t_2} is similar. As in Section 3.4, we use theoretical values to calculate the values of F_{ij} due to limitations of the performance counting infastructure. As Algorithm 7 explicitly copies data to the GPU, we use this known value for W_{2i}.

The calculation of energy values for the experiments, E_i^*, differ from the sequential model as we may have multiple power phases with which to contend (particularly due to the 1Hz sampling rate of our wall power meter). Looking back to Figure 2.4, we note that the non-initialization portion of the benchmark has two power phases: the GPU and CPU running together and the CPU running alone while it completes its workload. As with the sequential validation, we drop the first several samples from the trace so that we may capture the steady state of a phase. This may result in smaller phases being dropped from the data, and when this happened an average of about 2% of the runtime of the benchmark was dropped. We assume this is not a significant issue as the longer phases of the computation are indeed captured. As we fit parameters for both processors together in Equation (3.20), we use a script to align benchmark timestamps with power phases and then

compute the energy for each phase. The total energy for the benchmark is then the sum of these phase energies (i.e. $E^* = \sum_{i=1}^{c} H_i^* T_i^*$ where c is the number of power phases in the wall power trace).

Table 3.14 shows parameter fits and average percent error for models generated via the NNLS problems of Equations (3.19) and (3.20). As with the sequential machine model, column scaling was used to reduce the condition number of the matrices (A_{t_1}, A_{t_2} and A_e). Unsurprisingly, we also noticed that better fits were obtained when multiple combinations of host/GPU behavior were included within the NNLS problem: scaling the host problem size and scaling the GPU problem size. Perhaps due to this greater amount of variability within the regression, average error of the models was less than 5% and fitted parameter values appeared to be closer to reasonable values than with the sequential NNLS results despite several parameters still fitted as zero.

Recall that the the heterogeneous benchmarks for SGEMM and SGEMV both use optimized implementations from either Intel's MKL library or NVIDIA's CUBLAS. Thus, we would expect SGEMM computation to dominate runtime in Table 3.14. On the other hand, SGEMV is communication-bound and we expect communication to dominate runtime. This is again supported by the fitted parameter values. Surprisingly, the energy cost of moving data (β_{e_1}) is still large in the case of SGEMM on the host. Due to the low bandwidth of the PCIe link, it is unsurprising to see GPU energy dominated by communication costs. The idle power term ϵ_H fits to a value of approximately 182W for both kernels.

	SGEMM	SGEMV
γ_{t_1} (sec/flop)	2.46E-12	0
β_{t_1} (sec/word)	0	1.76E-10
γ_{t_2} (sec/flop)	4.15E-13	1.39E-11
β_{t_2} (sec/word)	0	1.50E-09
γ_{e_1} (joule/flop)	2.44E-10	0
β_{e_1} (joule/word)	1.08E-08	1.73e-08
γ_{e_2} (joule/flop)	5.51e-11	1.14e-09
β_{e_2} (joule/word)	1.21E-06	3.56e-08
ϵ_H (joule/sec)	182.92	182.11
Runtime % error	1.28	4.52
Energy % error	1.10	1.43

Table 3.14: Fitted heterogeneous machine parameters without row scaling

As with the solutions to the sequential NNLS problems, the runtime and energy models appear to accurately capture the behavior of a simple heterogeneous machine. Our approach to validation would be difficult to scale onto more devices, as isolating the power phases of each device becomes significantly more difficult. Thus, power monitoring equipment for each device would be required in addition to wall power. Even though the fitted parameters for the heterogeneous benchmarks appear closer to reality than on the sequential machine, it is once again clear that a more-sophisticated

benchmarking framework is required if the objective of modeling is to calculate individual parameters accurately. As noted in the previous section for the sequential machine, the NNLS problems may be low rank, contributing to the difficulty in fitting the parameters themselves. We also attempted to combine the SGEMM and SGEMV into a single fit, but with no improvement in fit quality. See the git repository (`https://github.com/agearh/dissertation.git`) associated with this work for further details.

3.5 Parameter Estimation for Machines and Implementations and Related Work

As we noted in previous sections, the models of runtime and energy developed within this work are able to reproduce the runtime and energy consumption behavior of several computational kernels with a reasonable degree of accuracy. This supports our claim from Chapter 2, which argues that the behavior of such kernels is often predicable within a given execution phase. We suspect that combining models of individual phases is a good approach to modeling multi-phase applications, as hinted by the heterogeneous results, but we delegate such analysis to future work. One observation, however, is that our approach to model fitting results in parameters that are reasonably accurate with regard to overall machine behavior, but that may result in a zero-value or values that do not coincide with physical intuition (i.e. static power (ϵ_e) fitted to 0.33W on a dual-socket Sandy Bridge-EP server).

Resolving the problem of accurately calculating machine parameters via regression is a difficult problem, especially in the case of energy-based parameters. As manufacturers typically do not report energy efficiency details (beyond thermal design power, or TDP), calculating parameters that describe hardware behavior requires a set of specialized benchmarks that are able to obtain significant coverage and explore the entire power range of the targeted set of components (or at least a sufficient faction of this range). Power data from this suite of benchmarks can then be used to fit model parameters, in a similar manner to the approach described earlier in this chapter. We regard parameter calculation to be related but orthogonal to the work presented in this thesis, as we outline an approach to bounding energy that is extensible to new models and increases in applicability as methods of parameter calculation improve. This section summarizes several key developments by other researchers on this problem.

In [148], Williams et al. describe Roofline, a model that describes potential floating point throughput as a function of arithmetic intensity (we present a generalized version of Roofline, Cityscape, in Chapter 7). The Roofline model utilizes two hardware parameters: peak floating point throughput and bandwidth, and provides algorithm developers with a rapid means of determining if an implementation is limited by communication bandwidth or floating point capability (i.e. Roofline provides an upper bound on floating point throughput). As described in Chapter 1, most programs are communication-limited. One benefit of the Roofline model is that both hardware parameters can be easily calculated from manufacturer data sheets or benchmarks (such as the STREAM benchmark for communication bandwidth [108]). The Roofline model can also be

used to explore the benefit of specific code optimizations on potential throughput, as the hardware-determined upper bound on throughput is often not attainable due to limitations of the given software implementation. For example, a code that does not use vectorization on a machine capable of computing four operations simultaneously will never attain more than 25% of the machine's peak floating point rate, even if the implementation is not limited by communication bandwidth.

The Roofline model for floating point throughput has recently been extended to energy [44] via models for runtime and energy similar to those described for our sequential machine model (Equations (3.1) and (3.2)). The authors of [44] describe a microbenchmark with the flexibility to vary arithmetic intensity, and argue that it is able to quantitatively approximate the upper bound on energy efficiency described by the roofline model for energy. Results from power-aware desktops and mobile processors are used as evidence to support this claim, and the results are extended to further machine architectures in [43]. Both publications present fitted model parameters obtained via these intensity-variable benchmarks. One criticism of this work lies in the construction of these benchmarks themselves: they perform a constant amount of data movement, and then parameterize intensity by replicating a vectorized block of arithmetic operations (balanced for fused multiply-add units). At high arithmetic intensities, these benchmarks predominantly only stress floating point units and registers without much impact upon the cache hierarchy. This is problematic, as several algorithms (e.g. optimized matrix-matrix multiplication) attain high intensities while continually stressing caches. In [44], this limitation is noted during a discussion of a benchmark constructed to reproduce the instruction mix of a Fast Multipole Method (FMM) algorithm for the n-body problem.

In related work, Kestor et al. [91] use a hand-coded microbenchmark suite designed to target various caches and other functional units on a dual-socket AMD Opteron system. Performance counters and on-die power meters were used to characterize the energy required to execute specific instructions, and validation was performed via compositions of the microbenchmarks used to fit the instruction energies. Kestor et al. then describe the primary energy consumers for the NAS Parallel Benchmarks [12], and find that L1 data movement typically dominates energy consumption (as all data must be moved through L1 into registers).

The complexity of modern processor designs as well as the level of specialized knowledge required to produce an effective microbenchmark suite suggests the need for microbenchmark generators similar to autotuning frameworks for code optimization. An initial step toward such generators can be found in the work of Bertran et al. [28], and the introduction to this paper serves as an excellent reference for previous work on generating benchmarks for individual performance metrics. The work of Bertran et al., however, extends this previous corpus and describes MicroProbe, a micro benchmark generation framework. MicroProbe was then used to calculate energy/instruction parameters for IBM's Power 7 processor, and the authors highlight its ability to flexibly generate code and explore design spaces. MicroProbe currently has two major limitations: it requires detailed architectural knowledge to create hardware description files for benchmark generation, and it is not currently available to the public.

The above work suggests a growing awareness for the need to calculate energy-related metrics to describe hardware components or instructions at the algorithm level. Furthermore, researchers have begun to investigate best practices for generating benchmarks that target desired hardware

components or instructions, even to the point of contracting a framework for microbenchmark generation (MicroProbe). In the future, a MicroProbe-style approach to generating tuned benchmarks appears promising, especially if the method for generating such benchmarks can be combined with existing work dedicated to architectural probing (e.g. automatic cache way or core frequency determination) to circumvent the need for expert creation of hardware description files. New hardware designs that utilize simpler, in-order pipelines (e.g. the Rocket core running the RISC-V ISA from UC Berkeley [143]) may make this process easier due to lower hardware design complexity.

Chapter 4

Bounds on Communication, Runtime and Energy for Specific Algorithms

In this chapter, we review related work on communication lower bounds for sequential and distributed parallel machines as well as new results in collaboration with Ballard et al. [19] that bound from below the maximum number of words transferred over a single internode link (Section 4.1). We then apply these communication lower bounds to the models of runtime and energy introduced in Chapter 3 to derive lower bounds on runtime and energy for a set of computational problems (Section 4.2). These energy lower bounds build upon previous work with Demmel, Schwartz and Lipshitz [55]. We also note that specific algorithms for these problems (2.5D $O(n^3)$ matrix-matrix multiplication [126], Communication-Avoiding Parallel Strassen [98] and a communication-avoiding $O(n^2)$ n-body algorithm [61]) are able to perfectly strong scale in runtime with constant energy for a range of processors. Finally, extending work with Ballard and Demmel [17], we derive lower bounds on the runtime and energy consumption of heterogeneous machines (Section 4.3) and present algorithms for dense matrix-vector and $O(n^3)$ matrix-matrix multiplication that attain these bounds (Section 4.4).

4.1 Communication Lower Bounds for Sequential and Distributed Parallel Machines

In the sequential and distributed parallel machine models, Ballard et al. [15] proved communication lower bounds (with some technical assumptions, see [15] for details) on programs of form

$$\begin{aligned}&\text{for all } (i_1, i_2, i_3) \in \mathcal{Z} \subset \mathbb{Z}^3, \text{ in some order,}\\ &\quad C(i_1, i_2) = C(i_1, i_2) +_{i_1,i_2} A(i_1, i_3) *_{i_1,i_2,i_3} B(i_3, i_2)\end{aligned}$$

where $\mathbb{Z}^3$ is the 3-dimensional lattice of integers, M is the utilized amount of fast memory, $\mathcal{Z}$ is the iteration space and $+_{i_1,i_2}$/$*_{i_1,i_2,i_3}$ are binary operations (e.g. addition and multiplication,

in many cases). Ballard et al. noted that this model is general enough to include not just matrix multiplication, but many direct linear algebra algorithms, such as LU factorization. The cardinality of the iteration space, $F = |\mathcal{Z}|$, can be thought of as the number of "useful" work operations required to complete the algorithm and in this chapter, we will assume that the elements of $\mathcal{Z}$ represent sets of floating point operations of constant size. In the sequential machine model, the number of reads/writes between slow and fast memory is bounded below by

$$W_S^{DLA}(M, N, F) = \Omega\left(\max\left(N, \frac{F}{M^{1/2}}\right)\right) \tag{4.1}$$

where problem size $N = I + O$ is the sum of the number of input (I) and output (O) words, respectively. DLA stands for "Direct Linear Algebra". Similarly, a lower bound on the number of transferred messages can be obtained by dividing Equation (4.1) by the largest possible message size, m:[1]

$$S_S^{DLA}(M, N, F) = \Omega\left(\max\left(\frac{N}{m}, \frac{F}{mM^{1/2}}\right)\right). \tag{4.2}$$

On parallel distributed machines, Ballard et al. assume a workload distributed so that every processor performs $\Omega(1/P)$ of the computation, and that the input and output data is distributed such that every processor stores $O(1/P)$ of the data. On this machine, a single copy of the data is already distributed across the local node memories, so the $N = I + O$ bound of the sequential model does not apply. In its place, Ballard et al. [21] observed that a *memory-independent lower bound* applies for distributed machines assuming there is just one copy of the data at the start of the computation (see [21] for more details). Therefore, the number of work operations is assumed to be roughly equally distributed amongst the various processing elements and the parallel distributed communication lower bounds are:

$$W_{DP}^{DLA}(M, P, F) = \Omega\left(\max\left(\left(\frac{F}{P}\right)^{2/3}, \frac{F}{PM^{1/2}}\right)\right) \tag{4.3}$$

and

$$S_{DP}^{DLA}(M, P, F) = \Omega\left(\max\left(\frac{1}{m}\left(\frac{F}{P}\right)^{2/3}, \frac{F}{mPM^{1/2}}\right)\right). \tag{4.4}$$

Ballard et al. also picked the iteration space $\mathcal{Z}$, arrays A, B, C, and binary operations $+_{i_1,i_2}$ and $*_{i_1,i_2,i_3}$ to represent many (dense or sparse) linear algebra algorithms. More details regarding the theoretical basis of these communication bounds and their derivation can be found in [15]. At a high level, the bounds are derived by bounding the maximal number of "useful" (defined within [15]) compute operations (e.g. flops) that can be performed upon the working set that can fit within the fast memory of size M. This bound is obtained for all direct problems by a generalization of the parallel matrix-matrix multiplication bound proven by Irony, Toledo and Tiskin [84] which

[1] For the remainder of this work, we abuse the Ω-notation a bit by including constants when they are known or necessary.

mapped matrix-matrix multiplication onto a construction that allowed the result of Loomis and Whitney [102] to bound the number of useful operations on a given amount of data. In Section 4.3, we extend the sequential communication lower bounds to a heterogeneous machine model as discussed in [17] along with optimal algorithms that attain the lower bound for dense classical matrix-matrix and matrix-vector multiplication.

In [20], Ballard et al. use graph expansion[2] to derive communication bounds on fast matrix multiplication algorithms of the type discussed within Section 3.3. In the sequential machine model, the word volume and latency bounds are

$$W_S^{FMM}(M,n) = \Omega\left(\max\left(n^2, \frac{n^{\omega_0}}{\boldsymbol{M}^{\omega_0/2-1}}\right)\right) \tag{4.5}$$

and

$$S_S^{FMM}(M,n) = \Omega\left(\max\left(\frac{n^2}{m}, \frac{n^{\omega_0}}{m\boldsymbol{M}^{\omega_0/2-1}}\right)\right). \tag{4.6}$$

where ω_0 is the exponent of the asymptotic floating point complexity of the algorithm (e.g. for Strassen's 1969 algorithm [129], $\omega_0 \approx 2.81$) and FMM stands for "Fast Matrix Multiplication". The first term of the $\max()$ in the lower bounds is the $I + O = 3n^2$ bound for dense matrix matrix multiplication with 3 matrices of size n-by-n. In the parallel model and combined with memory independent bounds derived in [21] [3], we obtain lower bounds on word volume

$$W_{DP}^{FMM}(M,P,n) = \Omega\left(\max\left(\frac{n^2}{P^{2/\omega_0}}, \frac{n^{\omega_0}}{P\boldsymbol{M}^{\omega_0/2-1}}\right)\right) \tag{4.7}$$

and latency

$$S_{DP}^{FMM}(M,P,n) = \Omega\left(\max\left(\frac{n^2}{mP^{2/\omega_0}}, \frac{n^{\omega_0}}{mP\boldsymbol{M}^{\omega_0/2-1}}\right)\right). \tag{4.8}$$

As reviewed in Section 3.3, the $O(n^2)$ n-body problem (and its variants with a cutoff parameter) represent a common computational problem in scientific computing. In [61], Driscoll et al. derive memory-dependent and memory-independent communication lower bounds for this problem. In the sequential model, the communication bounds are

$$W_S^{NB}(M,n) = \Omega\left(\max\left(n, \frac{n^2}{M}\right)\right) \tag{4.9}$$

and

$$S_S^{NB}(M,n) = \Omega\left(\max\left(\frac{n}{m}, \frac{n^2}{mM}\right)\right). \tag{4.10}$$

[2] Informally, the ratio of outgoing edges (or dependencies) from a subset of vertices (or compute operations) to the total number of dependencies with an endpoint in the set. See Section 4.1 for more details.

[3] The memory-independent bounds in [21] are only derived for Strassen's 1969 algorithm, but are readily extensible to a general class of fast matrix multiplication algorithms.

where the first term of the $\max()$ in the lower bounds is the $I + O = O(n)$ bound and NB stands for "N-Body". In the distributed parallel model, the lower bounds on word volume

$$W_{DP}^{NB}(M, P, n) = \Omega\left(\max\left(\frac{n}{P^{1/2}}, \frac{n^2}{PM}\right)\right) \tag{4.11}$$

and latency are

$$S_{DP}^{NB}(M, P, n) = \Omega\left(\max\left(\frac{n}{mP^{1/2}}, \frac{n^2}{mPM}\right)\right). \tag{4.12}$$

Several algorithms are able to asymptotically attain these communication lower bounds. We refer to such algorithms as *communication-optimal*. In the case of dense $O(n^3)$ matrix-matrix multiplication, we focus on one optimal algorithm in particular: 2.5D matrix-matrix multiplication [126]. This algorithm has the property of using data replication to attain a range of perfect strong scaling with constant energy (see Section 3.3 and [126] for more details). In Chapters 6 and 7, this property will allow us to compute the amount of memory and parameter values required to attain various energy, runtime and power bounds under constraints on the distributed parallel model. Similarity, we will use the communication-optimal $O(n^2)$ n-body (see [61], and Section 3.3) and parallel Strassen (see [98], and Section 3.3) algorithms due to their ability to trade additional memory usage for communication and perfectly strong scale.

The communication lower bounds and optimal algorithms analyzed within this work represent a small portion of the existing research into minimizing communication traffic for different types of computational kernels. Interested readers are referred to the survey Ballard et al. [13], as well as Ballard's [16] and Hoemmen's [76] PhD dissertations for more details.

Lower Bounds on the DP Models that Include Link Contention

As an extension of the distributed parallel (DP) model, recent work by Ballard, Demmel, Gearhart, Lipshitz, Schwartz and Toledo[19] lower bounds contention, i.e. the maximum number of words transferred over a single network link within a given network topology. In particular, Ballard et al. prove lower bounds on contention for algorithms running on machines with d-dimensional tori and mesh topologies. Borrowing from [19], we define the contention cost for an algorithm running a problem of size N with P processors in the distributed homogeneous model to be

$$W_{DP}^{\text{link}}(M, P, n, d) = \Omega\left(\max_{r \in R} \frac{W_{DP}(M \cdot r, P/r, n)}{d \cdot r \cdot h_r(G_{Net})}\right) \tag{4.13}$$

where

$$h_r(G_{Net}) = \min_{K \subseteq V, |K| \leq r} \frac{|\hat{E}(K, V \setminus K)|}{|\hat{E}(K)|} \tag{4.14}$$

is the small set expansion[4] of the network graph G_{Net}, $\hat{E}(K)$ is the edge set of vertex set K, $\hat{E}(K, K')$ is the set of edges with one endpoint in vertex set K and another endpoint in vertex set

[4]Informally, the minimum number of edges leaving a set of vertices of size at most r. See [19] for more details.

K',

$$R = \{r : 1 \leq r \leq P/2, \exists K \subseteq V s.t. |K| = r \text{ and } h_r(G_{Net}) = |\hat{E}(K, V \setminus K)|/|\hat{E}(K)|\},$$

M is the size of each processor's local memory and W_{DP} is the per-processor communication bound. The intuition behind set R is that we only wish to maximize over all sets K that asymptotically attain the bounds on h_r, which is a much smaller set of candidates than if one considered all sets of cardinality less than $P/2$. If we assume G_{Net} to be a d-dimensional torus, Ballard et al. prove (via a simplification of an earlier proof by Bollobás and Leader [35]) that

$$h_r(G_{Net}) = \Theta\left(r^{-1/d}\right). \tag{4.15}$$

If we assume that the memory-dependent term of Equation (4.3) dominates, and substitute this and Equation (4.15) in Equation (4.13), a memory-dependent contention bound for direct linear algebra (where $F = O(n^3)$) can be derived:

$$W_{DP}^{\text{link}DLA}(M, P, n, d) = \Omega\left(\max_{r \in R} \frac{W_{DP}^{DLA}(M \cdot r, P/r, n)}{d \cdot r \cdot h_r(G_{Net})}\right) = \Omega\left(\max_{r \in R} \frac{\frac{n^3}{(P/r)(Mr)^{1/2}}}{r \cdot h_r(G_{Net})}\right)$$

$$= \Omega\left(\max_{r \in R} \frac{\frac{n^3}{(P/r)(Mr)^{1/2}}}{r^{1-(1/d)}}\right) = \Omega\left(\max_{r \in R}\left(\frac{n^3}{PM^{1/2}}\right) r^{-(1/2)+(1/d)}\right).$$

Maximal values of r in the above expression can be shown to be some fraction of P (see Ballard et al. [19] for details). So, we assume the function to be maximized when r is P divided by some constant, and for asymptotic analysis set $r = P$. Then,

$$W_{DP}^{\text{link}DLA}(M, P, n, d) = \Omega\left(\frac{n^3}{M^{1/2} P^{(3/2)-1/d}}\right).$$

With this and the result of substituting the memory-independent term of Equation (4.3) and Equation (4.15) into Equation (4.13), we produce the direct linear algebra bounds shown in Table 4.1. This table includes both the older per-processor word bounds [84, 15, 21, 20, 61] as well as the new lower bounds on link contention for toroidal and mesh networks for several classes of algorithms. The expressions in Table 4.1 beg the question: "How large does d have to be for W_{DP}^{link} to be no larger than W_{DP}?" with regard to network topology.

To address this question, we first consider the case of the Communication-Avoiding Parallel Strassen (CAPS) algorithm [98]. As noted in Section 3.3, this algorithm is able to trade memory usage for communication and is able to perfectly strong scale in runtime with constant energy for a range of processors. The size of this strong scaling range is limited by either the contention or memory-independent communication bounds, depending on the torus dimension. Consider Figure 4.1. In the upper portion of the figure, we highlight which of the four lower bounds in the middle of Table 4.1 is largest for a range of processors (x-axis) while considering several torus dimensions (y-axis). The black lines in Figure 4.1 delineate regions of bound dominance,

		Memory Dependent	Memory Independent.
Direct Linear Algebra	W_{DP}^{DLA}	$\Omega\left(\frac{n^3}{PM^{1/2}}\right)$	$\Omega\left(\frac{n^2}{P^{2/3}}\right)$
	$W_{DP}^{\text{link}DLA}$	$\Omega\left(\frac{n^3}{P^{3/2-1/d}M^{1/2}}\right)$	$\Omega\left(\frac{n^2}{P^{1-1/d}}\right)$
Strassen and Strassen -like	W_{DP}^{FMM}	$\Omega\left(\frac{n^{\omega_0}}{PM^{\omega_0/2-1}}\right)$	$\Omega\left(\frac{n^2}{P^{2/\omega_0}}\right)$
	$W_{DP}^{\text{link}FMM}$	$\Omega\left(\frac{n^{\omega_0}}{P^{\omega_0/2-1/d}M^{\omega_0/2-1}}\right)$	$\Omega\left(\frac{n^2}{P^{1-1/d}}\right)$
$O(n^2)$ n-body	W_{DP}^{NB}	$\Omega\left(\frac{n^2}{PM}\right)$	$\Omega\left(\frac{n}{P^{1/2}}\right)$
	$W_{DP}^{\text{link}NB}$	$\Omega\left(\frac{n^2}{P^{2-1/d}M}\right)$	$\Omega\left(\frac{n}{P^{1-1/d}}\right)$

Table 4.1: Per-processor bounds (W_{DP}) ([84, 15, 21, 20, 61]) vs. the new contention bounds (W_{DP}^{link}) on a d-dimensional torus for classical linear algebra, fast matrix multiplication, and the $O(n^2)$ n-body problem.

and are plotted as continuous functions of the torus dimension (d) for illustrative purposes. We find that CAPS is completely limited by the memory-independent contention bound $\Omega(n^2/P^{1-1/d})$ for tori of dimension 2 and as such does not perfectly strong scale for any number of processors (see lower portion of Figure 4.1, which plots the increase in communication volume with number of processors). The memory-dependent contention bound $\Omega(n^{\omega_0}/(P^{\omega_0/2-1/d}M^{\omega_0/2-1})$ is always dominated by the memory-independent contention bound, and is thus not visible in the figure. To see this, we set the memory-dependent and memory-independent contention bounds equal and solve for P. From this, we observe that the memory-dependent contention bound dominates the memory-independent contention bound when $P \leq n^2/M$. As we assume that the entire problem fits in the node local memories, the minimal number of processors for Strassen and Strassen-like problems is asymptotically $P = n^2/M$. Thus, the memory-independent contention bound dominates the memory-dependent contention bound for all viable numbers of processors. This argument also holds for the cases of direct linear algebra and the $O(n^2)$ n-body problem, as well as programs that access arrays with affine expressions (see Chapter 5 for details). If $d = 3$ (the blue dashed line in Figure 4.1), the memory-dependent per-processor bound dominates until $P = P_{min}^{3(\omega_0-2)/2}$ where P_{min} is the minimal number of processors necessary to hold the problem (i.e. $P_{min} = n^2/M$). While $P_{min} \leq P \leq P_{min}^{3(\omega_0-2)/2}$, CAPS is able to use additional memory to offset the additional communication volume incurred by adding additional processors to the computation. Once contention-dominated, however, this perfect scaling ends and the communication volume increases with additional processors. This is shown in the lower portion of Figure 4.1 with a blue dashed line. Finally, for tori (or meshes) of dimension 4 or greater, the range of perfect strong scaling in runtime is limited by the memory-independent per-processor bound. This allows for a

wider range of perfect scaling than if contention dominated (dashed magenta line in lower portion of Figure 4.1).

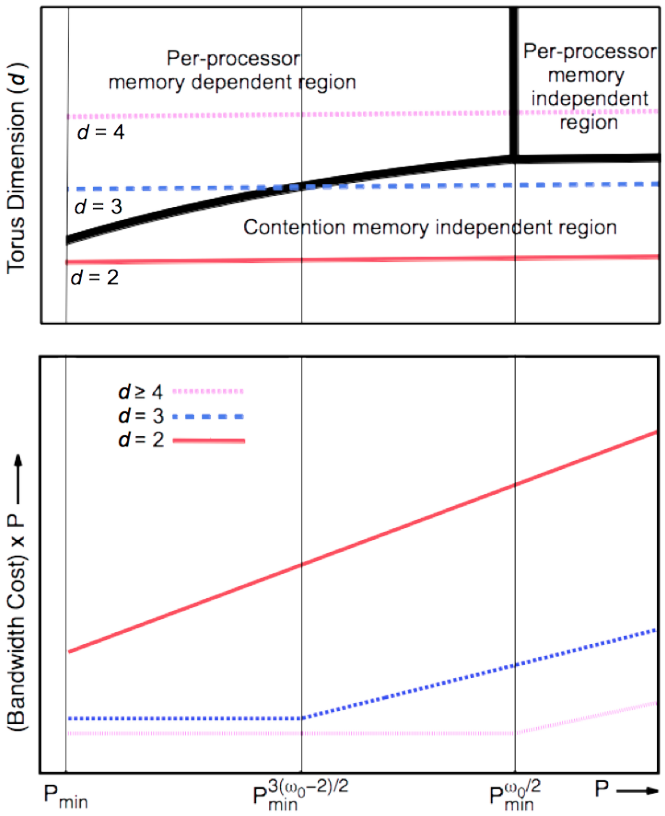


Figure 4.1: Communication bounds for Strassen's algorithm on d-dimensional tori. The lower plot is log-log, while the upper is linear on the y-axis. Horizontal lines in the lower plot correspond to perfect strong scaling.

In general, we must calculate the values of d for which the contention bounds dominate the per-processor bounds. This is accomplished by setting both memory-dependent and both memory-independent bounds equal to each other, and solving for d. In the cases of direct linear algebra, Strassen and Strassen-like algorithms and the $O(n^2)$ n-body problem, these expressions are

$$D_1 = \left\lfloor \frac{1}{s-1} \right\rfloor \text{ and } D_2 = \left\lceil \frac{s}{s-1} \right\rceil \tag{4.16}$$

where $s = \omega_0/2$ for classical $O(n^3)$ and Strassen-like matrix-matrix multiplication and $s = 2$ for the $O(n^2)$ n-body problem. We define ω_0 as the exponent of the floating point complexity of the algorithm, e.g. $\omega_0 = 3$ for classical $O(n^3)$ matrix-matrix multiplication. If $d \leq D_1$, the algorithm is always contention bound and if $d \geq D_2$ the algorithm is never contention bound. If the torus or mesh dimension lies between D_1 and D_2, a smaller range of perfect strong scaling exists before

Algorithm	ω_0	D_1	D_2
Classical	3	2	3
Strassen [129]	≈ 2.81	2	4
Schönhage [120]	≈ 2.55	3	5
Strassen [128]	≈ 2.48	4	6
Williams [149]	≈ 2.3727	5	7

Table 4.2: Torus dimensions so that communication cost is either always contention bound ($d \leq D_1$) or never contention bound ($d \geq D_2$) for a selection of matrix multiplication algorithms. The assertions regarding the last three algorithms are under some technical assumptions / conjecture, see [20].

the algorithm is dominated by contention. Table 4.2 (reproduced from [19]) presents values of D_1 and D_2 for several matrix-matrix multiplication algorithms.

While discussing runtime and energy bounds for specific algorithms in Section 4.2 we will mention the specific values of d pertinent to the algorithms under discussion, and often assume that the network is "good enough" such that it is not contention-dominated (i.e. $d \geq D_2$ for tori/meshes or the network topology is of another type that is not contention dominated for the problem at hand). Considering runtime, we realize that the program must wait for all communication traffic to finish prior to completion. Thus, bounds on maximum link contention do have an impact on runtime and can be constructed from the per-processor and memory-independent contention bounds. In the case of direct linear algebra, we define the communication lower bounds that we will use for runtime within the DP model to be

$$W_{DP_t}^{DLA}(M,P,n,d) = \Omega\left(\max\left(\frac{n^2}{P^{2/3}}, \frac{n^2}{P^{1-1/d}}, \frac{n^3}{PM^{1/2}}\right)\right) \tag{4.17}$$

and

$$S_{DP_t}^{DLA}(M,P,n,d) = \Omega\left(\max\left(\frac{n^2}{mP^{2/3}}, \frac{n^2}{mP^{1-1/d}}, \frac{n^3}{mPM^{1/2}}\right)\right). \tag{4.18}$$

Runtime bounds for the other classes of problems considered within this work can be derived in a similar manner. For Strassen and Strassen-like algorithms, we obtain

$$W_{DP_t}^{FMM}(M,P,n,d) = \Omega\left(\max\left(\frac{n^2}{P^{2/\omega_0}}, \frac{n^2}{P^{1-1/d}}, \frac{n^{\omega_0}}{PM^{\omega_0/2-1}}\right)\right) \tag{4.19}$$

and

$$S_{DP_t}^{FMM}(M,P,n,d) = \Omega\left(\max\left(\frac{n^2}{mP^{2/\omega_0}}, \frac{n^2}{mP^{1-1/d}}, \frac{n^{\omega_0}}{mPM^{\omega_0/2-1}}\right)\right). \tag{4.20}$$

For the lower bounds on the $O(n^2)$ n-body problem we obtain

$$W_{DP_t}^{NB}(M,P,n,d) = \Omega\left(\max\left(\frac{n}{P^{1/2}}, \frac{n}{P^{1-1/d}}, \frac{n^2}{PM}\right)\right) \tag{4.21}$$

and

$$S^{NB}_{DP_t}(M,P,n,d) = \Omega\left(\max\left(\frac{n}{mP^{1/2}}, \frac{n}{mP^{1-1/d}}, \frac{n^2}{mPM}\right)\right). \tag{4.22}$$

When considering energy, however, Equation (3.4) accounts for dynamic communication energy via a per-word term. Thus, the dynamic component of the energy model is agnostic to communication path and the link contention bounds do not apply (future work may address this limitation of the model). Longer runtime does require more energy, so the contention bound does indirectly affect the energy bound via the terms that are dependent on runtime. The direct linear algebra lower bounds for communication volume that we use for the dynamic portion of the energy model are

$$W^{DLA}_{DP_e}(M,P,n) = \Omega\left(\max\left(\frac{n^2}{P^{2/3}}, \frac{n^3}{PM^{1/2}}\right)\right) \tag{4.23}$$

and

$$S^{DLA}_{DP_e}(M,P,n) = \Omega\left(\max\left(\frac{n^2}{mP^{2/3}}, \frac{n^3}{mPM^{1/2}}\right)\right). \tag{4.24}$$

which are equivalent to Equations (4.3) and (4.4), respectively, if we assume $F = n^3$. For Strassen and Strassen-like algorithms, the expressions for $W^{FMM}_{DP_e}$ and $S^{FMM}_{DP_e}$ are identical to Equations (4.7) and (4.8) and similarly for the $O(n^2)$ n-body problem ($W^{NB}_{DP_e}$ and $S^{NB}_{DP_e}$) via Equations (4.11) and (4.11).

In the case of indirect networks the energy model may have to be extended to include the cost of switching nodes if the energy of routing is dependent on network load (otherwise, the energy cost of switching nodes can be added to the idle energy term of the model). Also, future work may attempt to capture the impact of path length on energy consumption and runtime by including terms that allow for per-link accounting. However, in this work we do not consider indirect networks when deriving bounds on energy and runtime.

4.2 Energy Lower Bounds for Specific Algorithms

$O(n^3)$ Classical Matrix Multiplication

Sequential (S) Model In the case of classical matrix-matrix multiplication that performs $O(n^3)$ flops, we know the following expressions for F, W_S and S_S in Equations (3.1) and (3.2) by specializing the direct linear algebra bounds in Equations (4.1) and (4.2):

$$F = \Theta(n^3), N = \Theta(n^2) \tag{4.25}$$

$$W^{CMM}_S(M,n) = \Omega\left(\max\left(n^2, \frac{n^3}{\boldsymbol{M}^{1/2}}\right)\right), \quad S^{CMM}_S(M,n) = \Omega\left(\max\left(\frac{n^2}{m}, \frac{n^3}{m\boldsymbol{M}^{1/2}}\right)\right) \tag{4.26}$$

where M is the size of fast memory, m is the size of the largest message we can send ($m \leq M$). CMM stand for "Classical Matrix-matrix Multiplication". From these expressions and Equation (3.1) the runtime lower bound for $O(n^3)$ matrix-matrix multiplication, T^{CMM}_S, becomes:

$$T_S^{CMM}(M,n) = \Omega\left(\gamma_t n^3 + \beta_t \max\left(n^2, \frac{n^3}{\boldsymbol{M}^{1/2}}\right) + \alpha_t \max\left(\frac{n^2}{m}, \frac{n^3}{m\boldsymbol{M}^{1/2}}\right)\right) \tag{4.27}$$

as we realize that the $O(n)$ flop/byte ratio for matrix-matrix multiplication implies that the second term of the bounds on W_S^{CMM} and S_S^{CMM} will dominate for all but small problems. This runtime bound is attained by blocked implementations of sequential matrix-matrix multiplication [54]. We now bound energy by utilizing Equation (3.2) to express:

$$\begin{aligned} E_S^{CMM}(M,\hat{M},n) = \Omega\Bigg(&\gamma_e n^3 + \beta_e \max\left(n^2, \frac{n^3}{\boldsymbol{M}^{1/2}}\right) + \alpha_e \max\left(\frac{n^2}{m}, \frac{n^3}{m\boldsymbol{M}^{1/2}}\right) \\ &+(\delta_e \hat{M} + \epsilon_e)\left(\gamma_t n^3 + \beta_t \max\left(n^2, \frac{n^3}{\boldsymbol{M}^{1/2}}\right) + \alpha_t \max\left(\frac{n^2}{m}, \frac{n^3}{m\boldsymbol{M}^{1/2}}\right)\right)\Bigg). \end{aligned} \tag{4.28}$$

where we recall that $\hat{M}$ is the amount of slow memory utilized during the calculation.

Distributed Parallel Model (DP1) In model DP1, M is the memory used per processor (which cannot exceed the physical memory per processor) and we assume that the machine is connected via a mesh or toroidal network of dimension d. We also assume that we use at least enough memory to store one copy of the data across all the processors, so $M = \Omega(n^2/P)$ (we again omit constant factors for simplicity). Also, this machine model assumes an asymptotically equal distribution of work between the processors (i.e. each processor has $O(n^3/P)$ of the work to compute). By substituting Equations (4.17) and (4.18) into the DP1 runtime model of Equation (3.3), we obtain a runtime lower bound for classical matrix-matrix multiplication on DP1:

$$\begin{aligned} T_{DP1}^{CMM}(M,P,n,d) = \Omega\Bigg(&\gamma_t \frac{n^3}{P} + \beta_t \max\left(\frac{n^2}{P^{2/3}}, \frac{n^2}{P^{1-1/d}}, \frac{n^3}{PM^{1/2}}\right) \\ &+\alpha_t \max\left(\frac{n^2}{mP^{2/3}}, \frac{n^2}{mP^{1-1/d}}, \frac{n^3}{mPM^{1/2}}\right)\Bigg). \end{aligned} \tag{4.29}$$

Regarding energy consumption, we similarly apply Equations (4.23) and (4.24) into the DP1 runtime model of Equation (3.4) and obtain

$$\begin{aligned} E_{DP1}^{CMM}(M,P,n,d) = \Omega\Bigg(P\Bigg(&\gamma_e \frac{n^3}{P} + \beta_e \max\left(\frac{n^2}{P^{2/3}}, \frac{n^3}{PM^{1/2}}\right) + \alpha_e \max\left(\frac{n^2}{mP^{2/3}}, \frac{n^3}{mPM^{1/2}}\right) \\ &+ (\delta_e M + \epsilon_e)\left(\gamma_t \frac{n^3}{P} + \beta_t \max\left(\frac{n^2}{P^{2/3}}, \frac{n^2}{P^{1-1/d}}, \frac{n^3}{PM^{1/2}}\right)\right. \\ &\left.+\alpha_t \max\left(\frac{n^2}{mP^{2/3}}, \frac{n^2}{mP^{1-1/d}}, \frac{n^3}{mPM^{1/2}}\right)\right)\Bigg)\Bigg) \end{aligned} \tag{4.30}$$

as a lower bound on energy consumption for a $O(n^3)$ matrix-matrix multiplication algorithm.

From prior work on 2.5D matrix-matrix multiplication [126], we know that redundant copies of matrices (by increasing M) can be used to decrease the amount of required communication (i.e. decrease W_{DP1}^{CMM} and S_{DP1}^{CMM}). In standard "2D" algorithms for matrix multiplication, each processor is given a local of problem of size $M = n^2/P$ on which to work, i.e. one copy of the data is evenly spread across the processors. In our above discussion of 2.5D matrix multiplication, we observed that for $P_{min} = n^2/M \leq P \leq n^3/M^{\frac{3}{2}}$, communication costs scale perfectly with increasing P assuming that we can use all the memory, M, that is part of each processor. Thus, each term of the runtime bound in Equation (4.29) decreases proportionately to P in this range depending on the torus dimension d. Because each term of the energy bound of Equation (4.30) is proportional to some term of the runtime bound, the energy stays constant as we increase the number of processors with a constant amount of memory per processor. At the 3D limit where $M = n^2/P^{2/3}$ and the memory-independent per-processor (MIP) bound dominates, the total energy lower is bounded by

$$\begin{aligned} E_{DP1\_MIP}^{CMM}(M,P,n) &= \Omega\left((\gamma_e+\gamma_t\epsilon_e)n^3 + \left((\beta_e+\beta_t\epsilon_e) + \frac{(\alpha_e+\alpha_t\epsilon_e)}{m}\right)n^2P^{1/3}\right. \\ &\quad \left. +\delta_e\gamma_t n^5\frac{1}{P^{2/3}} + \left(\delta_e\beta_t + \frac{\delta_e\alpha_t}{m}\right)n^4\frac{1}{P^{1/3}}\right) \end{aligned} \tag{4.31}$$

assuming that the network dimension d is sufficient such that the contention bound is also dominated by the per-processor bound. Increasing P in the 3D case decreases the energy costs due to memory usage, but increases the energy costs due to communication.

What values of d are needed on a torus or mesh to ensure a region of perfect strong scaling? As noted in Section 4.1, a 3D torus network is a perfect match to this algorithm [125], and scales in total size proportionally to P, so the $P\epsilon_e T_{DP1}^{CMM}$ term in E_{DP1}^{CMM} should capture its energy usage. To see this, note that a value of $d = 3$ in the link contention bound results in an expression identical to the memory-independent per-processor bound.

Distributed Parallel Model (DP2) Runtime and energy bounds for the DP2 model can be derived in a similar manner to DP1, albeit with the realization that the parameters at level 1 are derived from a sequential model for the nodes. So, from the DP2 runtime model in Equation (3.5) we can obtain this lower bound on runtime for $O(n^3)$ matrix-matrix multiplication:

$$\begin{aligned} T_{DP2}^{CMM}(M_0,M_1,P,n,d) = \Omega\left(\gamma_{t_1}\frac{n^3}{P} + \beta_{t_1}\max\left(\frac{n^2}{P}, \frac{n^3}{PM_1^{1/2}}\right) + \alpha_{t_1}\max\left(\frac{n^2}{mP}, \frac{n^3}{mPM_1^{1/2}}\right)\right) \\ +\beta_{t_0}\max\left(\frac{n^2}{P^{2/3}}, \frac{n^2}{P^{1-1/d}}, \frac{n^3}{PM_0^{1/2}}\right) + \alpha_{t_0}\max\left(\frac{n^2}{mP^{2/3}}, \frac{n^2}{mP^{1-1/d}}, \frac{n^3}{mPM_0^{1/2}}\right)\Bigg) \end{aligned} \tag{4.32}$$

where we must now distinguish between the size of each node's cache (M_1) and main memory (M_0), the latter of which is M in the DP1 model. Further, as all processors start with n^2/P data

in M_0, this represents the input/output bound on the node itself between M_1 and M_0. True to the definition of DP2, this bound accounts for communication on the individual nodes as well as internode communication that is potentially dominated by link contention. Along the same lines, we obtain a lower bound on DP2 energy by substituting the communication bounds into the model in Equation (3.6):

$$E_{DP2}^{CMM}(M_0, M_1, P, n, d) = \Omega\left(P\left(\gamma_{e_1}\frac{n^3}{P} + \beta_{e_1}\max\left(\frac{n^2}{P}, \frac{n^3}{PM_1^{1/2}}\right) + \alpha_{e_1}\max\left(\frac{n^2}{mP}, \frac{n^3}{mPM_1^{1/2}}\right)\right.\right.$$
$$\left.\left. +(\delta_{e_0}M_0 + \epsilon_{e_0})T_{DP2}^{CMM}\right) + \zeta|\hat{E}(G_{Net})|\right). \tag{4.33}$$

Strassen and Strassen-like Matrix Multiplication

As introduced in Section 3.3, fast matrix multiplication (FMM) algorithms multiply two $n \times n$ matrices in $O(n^{\omega_0})$ time, for some $2 < \omega_0 < 3$. For example, Strassen's algorithm [129] has exponent $\omega_0 = \log_2 7 \approx 2.81$.

Sequential Model (S) Via substitution of Equations (4.5) and (4.6) into the sequential runtime and energy expressions of Equations (3.1) and (3.2), the sequential runtime and energy lower bounds for Strassen and Strassen-like matrix-matrix multiplication algorithms are:

$$T_S^{FMM}(M, n) = \Omega\left(\gamma_t n^{\omega_0} + \beta_t \max\left(n^2, \frac{n^{\omega_0}}{\boldsymbol{M}^{\omega_0/2-1}}\right) + \alpha_t \max\left(\frac{n^2}{m}, \frac{n^{\omega_0}}{m\boldsymbol{M}^{\omega_0/2-1}}\right)\right) \tag{4.34}$$

and

$$E_S^{FMM}(M, \hat{M}, n) = \Omega\left(\gamma_e n^{\omega_0} + \beta_e \max\left(n^2, \frac{n^{\omega_0}}{\boldsymbol{M}^{\omega_0/2-1}}\right) + \alpha_e \max\left(\frac{n^2}{m}, \frac{n^{\omega_0}}{m\boldsymbol{M}^{\omega_0/2-1}}\right)\right.$$
$$\left. +(\delta_e\hat{M} + \epsilon_e)\left(\gamma_t n^{\omega_0} + \beta_t \max\left(n^2, \frac{n^{\omega_0}}{\boldsymbol{M}^{\omega_0/2-1}}\right) + \alpha_t \max\left(\frac{n^2}{m}, \frac{n^{\omega_0}}{m\boldsymbol{M}^{\omega_0/2-1}}\right)\right)\right). \tag{4.35}$$

As with the classical algorithm, we will often assume that the computation is not dominated by the input/output bound (admittedly a stronger assumption as ω_0 approaches 2).

Distributed Parallel Model (DP1) In model DP1, the bounds on runtime and energy are also derived in an identical fashion to that of $O(n^3)$ classical matrix-matrix multiplication. We substitute Equations (4.19) and (4.20) into the DP1 runtime model of Equation (3.3) to obtain

$$T_{DP1}^{FMM}(M,P,n,d) \geq \gamma_t \frac{n^{\omega_0}}{P} + \beta_t \max\left(\frac{n^2}{P^{2/\omega_0}}, \frac{n^2}{P^{1-1/d}}, \frac{n^{\omega_0}}{PM^{\omega_0/2-1}}\right)$$
$$+\alpha_t \max\left(\frac{n^2}{mP^{2/\omega_0}}, \frac{n^2}{mP^{1-1/d}}, \frac{n^{\omega_0}}{mPM^{\omega_0/2-1}}\right) \tag{4.36}$$

and

$$E_{DP1}^{FMM}(M,P,n,d) = \Omega\left(P\left(\gamma_e \frac{n^{\omega_0}}{P} + \beta_e \max\left(\frac{n^2}{P^{2/\omega_0}}, \frac{n^{\omega_0}}{PM^{\omega_0/2-1}}\right)\right.\right.$$
$$+\alpha_e \max\left(\frac{n^2}{mP^{2/\omega_0}}, \frac{n^{\omega_0}}{mPM^{\omega_0/2-1}}\right)+(\delta_e M+\epsilon_e)\left(\gamma_t \frac{n^{\omega_0}}{P} + \beta_t \max\left(\frac{n^2}{P^{2/\omega_0}}, \frac{n^2}{P^{1-1/d}}, \frac{n^{\omega_0}}{PM^{\omega_0/2-1}}\right)\right.$$
$$\left.\left.\left.+\alpha_t \max\left(\frac{n^2}{mP^{2/\omega_0}}, \frac{n^2}{mP^{1-1/d}}, \frac{n^3}{mPM^{\omega_0/2-1}}\right)\right)\right)\right) \tag{4.37}$$

are lower bounds on runtime and energy consumption for a generic $O(n^{\omega_0})$ Strassen-like matrix-matrix multiplication algorithm.

As mentioned in Section 3.3, via the Communication-Avoiding Parallel Strassen (CAPS) algorithm [98] it is possible to perform fast matrix multiplication with less communication than classical $O(n^3)$ matrix-matrix multiplication. Repeating the analysis from above where we assume that the memory-dependent per-processor (MDP) bound dominates, we find that the total energy is bounded by:

$$\begin{aligned} E_{DP1\_MDP}^{FMM}(M,P,n) &= \Omega\left((\gamma_e+\gamma_t\epsilon_e)n^{\omega_0} + \left((\beta_e+\beta_t\epsilon_e)+\frac{(\alpha_e+\alpha_t\epsilon_e)}{m}\right)\frac{n^{\omega_0}}{M^{\omega_0/2-1}}\right. \\ &\quad \left.+\delta_e\gamma_t M n^{\omega_0} + \left(\delta_e\beta_t + \frac{\delta_e\alpha_t}{m}\right)M^{2-\omega_0/2}n^{\omega_0}\right) \end{aligned} \tag{4.38}$$

in the case that $n^2/P \leq M \leq n^2/P^{2/\omega_0}$ (i.e. running fast matrix multiplication using limited memory). When $M = n^2/P^{2/\omega_0}$

$$\begin{aligned} E_{DP1\_MIP}^{FMM}(M,P,n) &= \Omega\left((\gamma_e+\gamma_t\epsilon_e)n^{\omega_0} + \left((\beta_e+\beta_t\epsilon_e)+\frac{(\alpha_e+\alpha_t\epsilon_e)}{m}\right)n^2P^{1-2/\omega_0}\right. \\ &\quad \left.+\delta_e\gamma_t n^5\frac{1}{P^{2/\omega_0}} + \left(\delta_e\beta_t+\frac{\delta_e\alpha_t}{m}\right)n^4\frac{1}{P^{4/\omega_0-1}}\right), \end{aligned} \tag{4.39}$$

where we are essentially running fast matrix multiplication using the maximal amount of usable memory and the memory-independent per-processor bound (MIP) dominates. As in the case of $O(n^3)$ classical matrix multiplication, the energy does not depend on P inside a perfect strong

scaling range (Equation (4.38)), so scaling P by some factor while holding M constant reduces the execution time by that factor without affecting the total energy. By setting the contention and per processor bounds equal to each other and solving for d, we find that link contention is a factor with Strassen-like algorithms when running on tori or meshes with dimension $d \leq 3$. A smaller region of perfect scaling exists on tori and meshes of dimension $d = 3$, but is still limited by contention.

Distributed Parallel Model (DP2) Runtime and energy bounds for the DP2 model can be derived in a similar manner to DP1, albeit with the realization that the parameters at level 1 are derived from a sequential model for the nodes. So, from the DP2 runtime model in Equation (3.5), and the communication bounds of Equations (4.19), (4.20), (4.7) and (4.8), we can obtain this lower bound on runtime for Strassen-like matrix-matrix multiplication:

$$T_{DP2}^{FMM}(M_0, M_1, P, n, d) = \Omega\left(\gamma_{t_1}\frac{n^{\omega_0}}{P} + \beta_{t_1}\max\left(\frac{n^2}{P}, \frac{n^{\omega_0}}{PM_1^{\omega_0/2-1}}\right)\right.$$
$$+\alpha_{t_1}\max\left(\frac{n^2}{mP}, \frac{n^{\omega_0}}{mPM_1^{\omega_0/2-1}}\right) + \beta_{t_0}\max\left(\frac{n^2}{P^{2/\omega_0}}, \frac{n^2}{P^{1-1/d}}, \frac{n^{\omega_0}}{PM_0^{\omega_0/2-1}}\right)$$
$$\left.+\alpha_{t_0}\max\left(\frac{n^2}{mP^{2/\omega_0}}, \frac{n^2}{mP^{1-1/d}}, \frac{n^{\omega_0}}{mPM_0^{\omega_0/2-1}}\right)\right). \tag{4.40}$$

where, as with the $O(n^3)$ matrix-matrix multiplication DP2 lower bound, we distinguish between the size of each node's cache (M_1) and main memory (M_0), the latter of which is M in the DP1 model. True to the definition of DP2, this bound accounts for communication on the individual nodes as well as internode communication that is potentially dominated by link contention. Along the same lines, we obtain a lower bound on DP2 energy by substituting the communication bounds of (4.7) and (4.8) into the model in Equation (3.6):

$$E_{DP2}^{FMM}(M_0, M_1, P, n, d) = \Omega\Bigg(P\Bigg(\gamma_{e_1}\frac{n^{\omega_0}}{P} + \beta_{e_1}\max\left(\frac{n^2}{P}, \frac{n^{\omega_0}}{PM_1^{\omega_0/2-1}}\right)$$
$$+\alpha_{e_1}\max\left(\frac{n^2}{mP}, \frac{n^{\omega_0}}{mPM_1^{\omega_0/2-1}}\right)$$
$$+ (\delta_{e_0}M_0 + \epsilon_{e_0})T_{DP2}^{FMM}\Bigg) + \zeta|\hat{E}(G_{Net})|\Bigg). \tag{4.41}$$

Matrix-vector multiplication

As mentioned in Section 3.3, we may compute a matrix-vector multiplication using either a dense or sparse input matrix. In the sparse situation, the algorithm becomes more difficult to implement

efficiently due to irregular memory access patterns. To describe energy bounds for this problem (and for simplicity), we assume that the input matrix is stored in compressed sparse row (CSR) format and the matrix is not stored in a symmetric format. We also only derive bounds for the sequential machine model, to avoid the challenge of partitioning the matrix across a distributed machine (see Section 3.3 for more details and references).

Sequential Model (S) In the sequential machine model, matrix-vector multiplication requires

$$F = 2n^2, W_S^{DMV}(n) = \Omega\left(n^2\right), S_S^{DMV}(n) = \Omega\left(\frac{n^2}{m}\right)$$

and

$$F = 2nnz, W_S^{SMV}(nnz) = \Omega\left(\frac{3}{2}nnz\right), S_S^{SMV}(nnz) = \Omega\left(\frac{3nnz}{2m}\right)$$

operations for the dense and sparse situations, respectively, where nnz is the number of non zero values in the input sparse matrix. DMV and SMV stand for "Dense Matrix-Vector multiplication" and "Sparse Matrix-Vector multiplication", respectively. We assume that both input and output vectors are of size small enough to fit in cache. The runtime and energy bounds for the dense situation then become:

$$T_S^{DMV}(n) = \Omega\left(n^2\left(\gamma_t + \beta_t + \frac{\alpha_t}{m}\right)\right)$$

$$E_S^{DMV}(\hat{M}, n) = \Omega\left(n^2\left(\gamma_e + \beta_e + \frac{\alpha_e}{m} + (\delta_e\hat{M} + \epsilon_e)\left(\gamma_t + \beta_t + \frac{\alpha_t}{m}\right)\right)\right).$$

where $\hat{M}$ is the amount of utilized slow memory. For the sparse case:

$$T_S^{SMV}(nnz) = \Omega\left(nnz\left(\gamma_t + \beta_t + \frac{\alpha_t}{m}\right)\right)$$

$$E_S^{SMV}(\hat{M}, nnz) = \Omega\left(nnz\left(\gamma_e + \beta_e + \frac{\alpha_e}{m} + (\delta_e\hat{M} + \epsilon_e)\left(\gamma_t + \beta_t + \frac{\alpha_t}{m}\right)\right)\right).$$

$O(n^2)$ n-body problem

Another example where perfect strong scaling is possible is the direct ($O(n^2)$) implementation of the n-body algorithm, where each particle (or "object") has to directly interact with every other particle (this is not limited to gravity or electrostatics, any interaction where we can basically just "sum" the results of individual interactions works). See Section 3.3 for a more detailed overview. Analogous to the case of 2.5D matrix-matrix multiplication, we can replicate data upon processors to reduce the amount of required communication. In the "1D" version of the direct n-body problem there is no replication and we move n words. In the "2D" version, we map the processors onto a $P^{1/2}$-by-$P^{1/2}$ grid and replicate the input data $\sqrt{P}$ times to reduce the number of words moved by a factor of $\sqrt{P}$. Based on these descriptions, one can imagine a "1.5D" variant of the direct n-body problem that utilizes memory in the range $n/P \leq M \leq n/P^{1/2}$. More details can be found in [61]. To address the number of flops performed in the algorithm, we add an additional parameter f that represents the number of flops necessary to compute the interaction of a pair of particles; although f is a constant, it may be quite large.

Sequential Model (S) From the communication lower bounds from Equations (4.9) and (4.10) and Equation (3.1), the runtime lower bound for the $O(n^2)$ n-body problem, T_S^{NB}, becomes:

$$T_S^{NB}(M,n) = \Omega\left(\gamma_t f n^2 + \beta_t \max\left(n, \frac{n^2}{M}\right) + \alpha_t\left(\frac{n}{m}, \frac{n^2}{mM}\right)\right) \tag{4.42}$$

as $F = \Theta(fn^2)$. We realize that the $O(n)$ flop/byte ratio for the problem implies that the second term of the bounds on W_S^{NB} and S_S^{NB} will dominate for all but small problems. We now bound energy by substituting Equations (4.9),(4.10) and (4.42) Equation (3.2) to attain:

$$E_S^{NB}(M,\hat{M},n) = \Omega\Bigg(\gamma_e f n^2 + \beta_e\left(n, \frac{n^2}{M}\right) + \alpha_e\left(\frac{n}{m}, \frac{n^2}{mM}\right) + (\delta_e \hat{M} + \epsilon_e)\left(\gamma_t f n^2 + \beta_t\left(n, \frac{n^2}{M}\right) + \alpha_t\left(\frac{n}{m}, \frac{n^2}{mM}\right)\right)\Bigg). \tag{4.43}$$

Distributed Parallel Model (DP1) In model DP1, M is the memory used per processor (which cannot exceed the physical memory per processor) and we assume that the machine is connected via a mesh or toroidal network. We also assume that we use at least enough memory to store one copy of the data across all the processors, so $M = \Omega(n/P)$ (we again omit constant factors for simplicity). Also, this machine model assumes an asymptotically equal distribution of work between the processors (i.e. $F = O(n^2/P)$). By substituting this value of F and Equations (4.21) and (4.22) into the DP1 runtime model of Equation (3.3), we obtain a runtime lower bound for the $O(n^2)$ n-body problem on DP1:

$$T_{DP1}^{NB}(M,P,n,d) = \Omega\left(\gamma_t \frac{fn^2}{P} + \beta_t \max\left(\frac{n}{P^{1/2}}, \frac{n}{P^{1-1/d}}, \frac{n^2}{PM}\right) + \alpha_t \max\left(\frac{n}{mP^{1/2}}, \frac{n}{mP^{1-1/d}}, \frac{n^2}{mPM}\right)\right). \tag{4.44}$$

Regarding energy consumption, we similarly apply Equations (4.11) and (4.12) (as we recall that $W_{DP_e}^{NB} = W_{DP}^{NB}$ and $S_{DP_e}^{NB} = S_{DP}^{NB}$) into the DP1 runtime model of Equation (3.4) and obtain

$$E_{DP1}^{NB}(M,P,n,d) = \Omega\Bigg(P\Bigg(\gamma_e \frac{fn^2}{P} + \beta_e \max\left(\frac{n}{P^{1/2}}, \frac{n^2}{PM}\right) + \alpha_e \max\left(\frac{n}{mP^{1/2}}, \frac{n^2}{mPM}\right) + (\delta_e M + \epsilon_e)\left(\gamma_t \frac{fn^2}{P} + \beta_t \max\left(\frac{n}{P^{1/2}}, \frac{n}{P^{1-1/d}}, \frac{n^2}{PM}\right) + \alpha_t \max\left(\frac{n}{mP^{1/2}}, \frac{n}{mP^{1-1/d}}, \frac{n^2}{mPM}\right)\right)\Bigg)\Bigg) \tag{4.45}$$

as a lower bound on energy consumption for a generic $O(n^2)$ n-body problem.

From prior work on the 1.5D n-body algorithm [61], we know that redundant copies of the input data (by increasing M) can be used to decrease the amount of required communication (i.e. decrease W and S). In standard "1D" algorithms for matrix multiplication, each processor is given a local of problem of size $M = n/P$) on which to work, i.e. one copy of the data is evenly spread across the processors. In our above discussion of the 1.5D algorithm, we observed that for $P_{min} = n/M \leq P \leq (n/M)^2$, communication costs scale perfectly with increasing P. Thus, each term of the runtime bound in Equation (4.44) decreases proportionately to P in this range depending on torus dimension d. Because each term of the energy bound of Equation (4.45) is proportional to some term of the runtime bound, the energy stays constant as we increase the number of processors with a constant amount of memory per processor. At the "2D" limit where $P = (n/M)^2$ and the memory-independent per-processor (MIP) bound dominates (assuming the network is not dominated by contention), the total energy lower bounded by

$$\begin{aligned} E^{NB}_{DP1\_MIP}(M,P,n) &= \Omega\left((\gamma_e+\gamma_t\epsilon_e)fn^2 + \left((\beta_e+\beta_t\epsilon_e) + \frac{(\alpha_e+\alpha_t\epsilon_e)}{m}\right)nP^{1/2}\right. \\ &\quad \left. +\delta_e\gamma_t f n^3\frac{1}{P^{1/2}} + \left(\delta_e\beta_t + \frac{\delta_e\alpha_t}{m}\right)n^2\right) \end{aligned} \tag{4.46}$$

assuming that the network dimension d is sufficient such that the contention bound is also dominated by the per-processor bound. Increasing P in the 2D case decreases the energy costs due to memory usage, but increases the energy costs due to communication.

What values of d are needed on a torus or mesh to ensure a region of perfect strong scaling? By substituting $\omega_0 = 2$ into the expressions for D_1 and D_2 (Equation (4.16)) in Section 4.1, a 2D torus network is a perfect match to this algorithm and scales in total size proportionally to P, so the $P\epsilon_e T$ term in E^{NB}_{DP1} should capture its energy usage. To see this, note that a value of $d = 2$ in the link contention bound results in an expression identical to the memory-independent per-processor bound (i.e. $n/P^{1-1/d} = n/P^{1/2}$ when $d = 2$).

Distributed Parallel Model (DP2) Runtime and energy bounds for the DP2 model can be derived in a similar manner to DP1, albeit with the realization that the parameters at level 1 are derived from a sequential model for the nodes. By substituting this value of $F = fn^2/P$ and Equations (4.21), (4.22), (4.9) and (4.10) into the DP2 runtime model of Equation (3.5), we obtain a runtime lower bound for the $O(n^2)$ n-body problem on DP2:

$$\begin{aligned} T^{NB}_{DP2}(M_0,M_1,P,n,d) = \Omega\left(\gamma_{t_1}\frac{fn^2}{P} + \beta_{t_1}\max\left(\frac{n}{P},\frac{n^2}{PM_1}\right) + \alpha_{t_1}\max\left(\frac{n}{mP},\frac{n^2}{mPM_1}\right)\right. \\ \left. +\beta_{t_0}\max\left(\frac{n}{P^{1/2}},\frac{n}{P^{1-1/d}},\frac{n^2}{PM_0}\right) + \alpha_{t_0}\max\left(\frac{n}{mP^{1/2}},\frac{n}{mP^{1-1/d}},\frac{n^2}{mPM_0}\right)\right). \end{aligned} \tag{4.47}$$

where we distinguish between the size of each node's cache (M_1) and main memory (M_0), the latter of which is M in the DP1 model. True to the definition of DP2, this bound accounts for

communication on the individual nodes as well as internode communication that is potentially dominated by link contention. Along the same lines, we obtain a lower bound on DP2 energy by substituting the communication bounds of Equations (4.11), (4.12), (4.9) and (4.10) into the model in Equation (3.6):

$$E_{DP2}^{NB}(M_0, M_1, P, n, d) = \Omega\left(P\left(\gamma_{e_1}\frac{fn^2}{P} + \beta_{e_1}\max\left(\frac{n}{P}, \frac{n^2}{PM_1}\right) + \alpha_{e_1}\max\left(\frac{n}{mP}, \frac{n^2}{mPM_1}\right)\right.\right.$$
$$\left.\left. + \delta_{e_0}M_0T_{DP2} + \epsilon_{e_0}T_{DP2}\right) + \zeta|\hat{E}(G_{Net})|\right). \quad (4.48)$$

4.3 Bounds on Heterogeneous Machines

Suppose we run an algorithm which executes F flops on the heterogeneous (H) machine model, and suppose the algorithm assigns F_i flops to proc$_i$ for $1 \leq i \leq P$, such that $\sum F_i = F$. Then we can focus our attention on one compute element proc$_i$ and model the communication between the local memory of proc$_i$ and machine global memory as two levels of a sequential machine. In this way we obtain a lower bound on the number of words W_{H_i} transferred to/from proc$_i$ by applying Equation (4.1) and, similarly, a lower bound on the number of messages S_{H_i} by applying Equation (4.2). Although we can obtain separate lower bounds for each compute element, the bounds apply only to a particular partitioning of the total flops. We would like a lower bound which applies to any assignment of the F flops to the different compute elements. Toward this end, we broaden our focus from the individual communication costs of each compute element to the total parallel runtime. As we've noted previously, this runtime model ignores any potential overlap of computation and communication, though we note that completely overlapping computation and communication will decrease the runtime by perhaps a factor of 2 or 3.

In the heterogeneous model, the parallel runtime is determined by the last compute element to finish its computation. Thus, given partition $\{F_i\}$ of the F flops (i.e., $\sum F_i = F$), partition $\{N_i\}$ of the input/output data of total size N, and the fast memory sizes $\{M_i\}$, we have

$$T_H(\{M_i\}, \{N_i\}, \{F_i\}) = \max_{1 \leq i \leq P}(\gamma_{t_i}F_i + \beta_{t_i}W_{H_i} + \alpha_{t_i}S_{H_i})$$

where F_i, W_{H_i}, and S_{H_i} are the number of flops executed, words communicated, and messages communicated, respectively, by proc$_i$ during the course of the algorithm. In order to obtain a more general lower bound, we can find the minimum over all possible partitions $\{F_i\}$ with $\sum F_i = F$, yielding

$$T_{H^*}(\{M_i\}, N, F) = \min_{\sum F_i = F}\left(\max_{1 \leq i \leq P}(\gamma_{t_i}F_i + \beta_{t_i}W_{H_i} + \alpha_{t_i}S_{H_i})\right).$$

assuming that the optimal partition of flops will imply the partition of the N input and output words into a block N_i for each processor. Assuming that Equations (4.1) and (4.2) hold, we can apply them to obtain our heterogeneous lower bound on parallel runtime:

$$T^{DLA}_{H^*}(\{M_i\}, N, F) = \Omega\left(\min_{\sum F_i = F}\left[\max_{1\le i\le P}\left(\gamma_{t_i}F_i + \beta_{t_i}\max\left(N_i, \frac{F_i}{M_i^{1/2}}\right)\right.\right.\right. \\ \left.\left.\left. + \alpha_{t_i}\max\left(\frac{N_i}{m_i}, \frac{F_i}{m_i M_i^{1/2}}\right)\right)\right]\right). \tag{4.49}$$

where again we assume the partition of flops ($\{F_i\}$) implies a partition of the input and output data ($\{N_i\}$) and DLA stands for "Direct Linear Algebra".

We obtain a general lower bound on energy for heterogeneous machine in a similar manner. Given partition $\{F_i\}$ of the F flops (i.e., $\sum F_i = F$), partition $\{N_i\}$ of the input/output data of total size N, and the fast memory sizes $\{M_i\}$, we have

$$\begin{aligned} E_H(\{M_i\}, \hat{M}, \{N_i\}, \{F_i\}) &= \sum_{i=1}^{P} E_{H_i} + \hat{\delta}\hat{M}T_H + \epsilon_H T_H \\ &= \sum_{i=1}^{P}(\gamma_{e_i}F_i + \beta_{e_i}W_{H_i} + \alpha_{e_i}S_{H_i}) + (\hat{\delta}\hat{M} + \epsilon_H)T_H \end{aligned}$$

where we assume that energy is consumed by all global memory during the entire runtime of the algorithm (e.g. if the algorithm touches a word of global memory once, we account an idle memory cost for that word for the entire runtime. If we minimize over all the possible partitions of F:

$$E_{H^*}(\{M_i\}, \hat{M}, N, F) = \min_{\sum F_i = F}\left[\sum_{i=1}^{P}(\gamma_{e_i}F_i + \beta_{e_i}W_{H_i} + \alpha_{e_i}S_{H_i}) + (\hat{\delta}\hat{M} + \epsilon_H)T_H\right]$$

As with the bound on T_H, we can substitute Equations (4.1) and (4.2) for W_i and S_i to obtain a lower bound on energy consumption for the heterogeneous machine:

$$\begin{aligned} E^{DLA}_{H^*}(\{M_i\}, \hat{M}, N, F) = \Omega\left(\min_{\sum F_i = F}\left(\sum_{i=1}^{p}\left[\gamma_{e_i}F_i + \beta_{e_i}\max\left(N_i, \frac{F_i}{M_i^{1/2}}\right)\right.\right.\right. \\ \left.\left. + \alpha_{e_i}\max\left(\frac{N_i}{m_i}, \frac{F_i}{m_i M_i^{1/2}}\right)\right] \\ + \left(\hat{\delta}\hat{M} + \epsilon_H\right)\max_{1\le i\le P}\left(\gamma_{t_i}F_i + \beta_{t_i}\max\left(N_i, \frac{F_i}{M_i^{1/2}}\right)\right. \\ \left.\left.\left. + \alpha_{t_i}\max\left(\frac{N_i}{m_i}, \frac{F_i}{m_i M_i^{1/2}}\right)\right)\right)\right) \end{aligned} \tag{4.50}$$

where again we assume the partition of flops ($\{F_i\}$) implies a partition of the input and output data ($\{N_i\}$). In the following two subsections, we will identify two circumstances where these lower bounds may be greatly simplified. In each case, the simplifications will suggest algorithms which can attain the bounds to within constant factors. In this way, we will argue that the lower bounds given in (4.49) and (4.50) are asymptotically tight for the problems solved by these algorithms.

Input/Output Dominated Lower Bounds

In this section, we focus on the lower bound based on original inputs and final outputs for each proc_i. That is, if we ignore the lower bound guaranteed by the result based on Loomis-Whitney, we obtain other valid lower bounds which may be lower than the one in (4.49). These input/output dominated lower bounds, given by

$$T^{IO}_{H^*}(N,F) = \Omega\left(\min_{\sum F_i = F}\left[\max_{1\le i\le P}\left(\gamma_{t_i}F_i + \beta_{t_i}(N_i) + \alpha_{t_i}\left(\frac{N_i}{m_i}\right)\right)\right]\right) \tag{4.51}$$

$$E^{IO}_{H^*}(N,F) = \Omega\left(\min_{\sum F_i = F}\left(\sum_{i=1}^{P}\left[\gamma_{e_i}F_i + \beta_{e_i}N_i + \alpha_{e_i}\left(\frac{N_i}{m_i}\right)\right] + \left(\hat{\delta}\hat{M} + \epsilon_H\right)\max_{1\le i\le P}\left(\gamma_{t_i}F_i + \beta_{t_i}N_i + \alpha_{t_i}\left(\frac{N_i}{m_i}\right)\right)\right)\right) \tag{4.52}$$

are valid for any algorithm where the original inputs and final outputs must reside in global memory. IO stands for "Input/Output". We may simplify these bounds for an algorithm with a direct relationship between flops and input and output data.

For example, in the case of BLAS2 [31] operations like n-by-n dense-matrix-vector-multiplication, $F = O(n^2)$ and $N = O(n^2)$. Thus, the number of inputs and outputs in BLAS2 functions are related to the number of flops via some constant g.

Runtime-Optimal Partition of Work Suppose that we would like to determine the runtime-optimal partition of the floating point operations required to compute dense matrix-vector multiplication. To do this, we first substitute the known values of F and N as mentioned previously. Thus, we can represent inequality (4.51) as

$$T^{IO}_{H^*}(N,F) = \Omega\left(\min_{\sum F_i = n^2}\left[\max_{1\le i\le P}\left(\gamma_{t_i}F_i + \beta_{t_i}(gF_i) + \alpha_{t_i}\left(\frac{gF_i}{m_i}\right)\right)\right]\right)$$

or

$$T^{IO}_{H^*}(N,F) = \Omega\left(\min_{\sum F_i = n^2}\left(\max_{1\le i\le P}\ \mu_{t_i}F_i\right)\right),$$

where

$$\mu_{t_i} = \gamma_{t_i} + g\beta_{t_i} + \frac{g\alpha_{t_i}}{m_i} \tag{4.53}$$

and g is a constant determined by the specific number of words per flop for a given BLAS2 algorithm.

We can simplify this min-max expression for optimal runtime by solving the associated linear program. Observe that the minimum is attained when $\mu_{t_i} F_i$ is constant for $1 \leq i \leq P$ (i.e., the compute elements finish simultaneously), and we discover that a partition attaining the minimum satisfies

$$F_i = \frac{\frac{1}{\mu_{t_i}}}{\sum_j \frac{1}{\mu_{t_j}}} n^2 \tag{4.54}$$

for $1 \leq i \leq P$. Thus, for BLAS2 operations, we obtain the partition-independent, input/output dominated lower bound

$$T_{H^*}^{IO}(N, F) = \Omega\left(\max_{1 \leq i \leq P} \mu_{t_i} F_i\right) = \Omega\left(\frac{n^2}{\sum_j \frac{1}{\mu_{t_j}}}\right). \tag{4.55}$$

Again, inequality (4.55) may not be as tight a bound as (4.49) in general, but we will argue in Section 4.4 that it can be attained in the case of matrix-vector multiplication. This will imply that in that case, both bounds are equivalent and tight.

Energy-Optimal Partition of Work Similarly, we can determine the energy-optimal partition of the flops across the processors. In the case of dense matrix-vector multiplication, the input/output-dominated energy bound of Equation (4.52) becomes

$$E_{H^*}^{IO}(\hat{M}, N, F) = \Omega\left(\min_{\sum F_i = n^2} \left(\sum_{i=1}^{P} \left[\gamma_{e_i} F_i + \beta_{e_i}(gF_i) + \alpha_{e_i}\left(\frac{gF_i}{m_i}\right)\right] + \left(\hat{\delta}\hat{M} + \epsilon_H\right) \max_{1 \leq i \leq P} \left(\gamma_{t_i} F_i + \beta_{t_i}(gF_i) + \alpha_{t_i}\left(\frac{gF_i}{m_i}\right)\right)\right)\right)$$

or

$$E_{H^*}^{IO}(\hat{M}, N, F) = \Omega\left(\min_{\sum F_i = n^2} \left(\sum_{i=1}^{p} \mu_{e_i} F_i + \left(\hat{\delta}\hat{M} + \epsilon_H\right) \max_{1 \leq i \leq P} (\mu_{t_i} F_i)\right)\right) \tag{4.56}$$

where g and μ_{t_i} were defined previously and

$$\mu_{e_i} = \gamma_{e_i} + \beta_{e_i} g + \frac{\alpha_{e_i} g}{m_i}.$$

Unfortunately, the extra parameters of the energy bound result in a linear program that does not lend itself to a closed-form solution and a numerical approach to solving the linear program is required. In standard form, the linear program becomes

$$\max y^T x$$

$$\text{subject to } Ax \leq b$$

$$\text{and } x \geq 0$$

where

$$y^T = [-\mu_{e_0}, ..., -\mu_{e_P}, -(\hat{\delta}\hat{M} + \epsilon_H)]$$

$$x^T = [F_0, ..., F_P, z]$$

and the rows of A and entries of b are formed from the constraints

$$\sum_{i=1}^{P} F_i \leq n^2$$

$$-\sum_{i=1}^{P} F_i \leq -n^2$$

$$\mu_{t_i} F_i - z \leq 0 \text{ for all } 1 \leq i \leq P.$$

Note that we require two constraints to enforce the equality that $\sum_i F_i = F = n^2$ and that we replace the maximum over processor runtimes with a dummy variable, z, and constrain it with P additional expressions to produce the desired behavior.

Loomis-Whitney Dominated Lower Bound

In this section, on the other hand, we focus on the lower bounds based on Loomis-Whitney [102]. This time, ignoring the lower bound guaranteed by having to read the original inputs and write the final outputs, we obtain another lower bound which may be lower than the ones in (4.49) and (4.50). These Loomis-Whitney dominated lower bounds for runtime and energy are given by

$$T_{H^*}^{DLA}(\{M_i\}, F) = \Omega\left(\min_{\sum F_i = F}\left[\max_{1\leq i\leq P}\left(\gamma_{t_i} F_i + \beta_{t_i}\left(\frac{F_i}{M_i^{1/2}}\right) + \alpha_{t_i}\left(\frac{F_i}{m_i M_i^{1/2}}\right)\right)\right]\right). \quad (4.57)$$

$$E_{H^*}^{DLA}(\{M_i\}, \hat{M}, F) = \Omega\left(\min_{\sum F_i = F}\left(\sum_{i=1}^{P}\left[\gamma_{e_i} F_i + \beta_{e_i}\left(\frac{F_i}{M_i^{1/2}}\right) + \alpha_{e_i}\left(\frac{F_i}{m_i M_i^{1/2}}\right)\right]\right.\right.$$
$$\left.\left. + \left(\hat{\delta}\hat{M} + \epsilon_H\right)\max_{1\leq i\leq P}\left(\gamma_{t_i} F_i + \beta_{t_i}\left(\frac{F_i}{M_i^{1/2}}\right) + \alpha_{t_i}\left(\frac{F_i}{m_i M_i^{1/2}}\right)\right)\right)\right)$$

where DLA stands for "Direct Linear Algebra". For example, these lower bounds often apply to BLAS3 [31] operations such as n-by-n dense-matrix-matrix-multiplication as well as most direct linear algebra algorithms where $F = O(n^3)$.

Runtime-Optimal Partition of Work Suppose that we would like to determine the runtime-optimal partition of the floating point operations required to compute $O(n^3)$ classical matrix-matrix multiplication on dense matrices. To do this, we first substitute the known values of F, W and S as mentioned previously. Thus, we can represent inequality (4.57) as

$$T_{H^*}^{DLA}(\{M_i\}, F) = \Omega\left(\min_{\sum F_i = n^3}\left(\max_{1\leq i\leq P} \nu_{t_i} F_i\right)\right)$$

where

$$\nu_{t_i} = \gamma_{t_i} + \frac{\beta_{t_i}}{M_i^{1/2}} + \frac{\alpha_i}{m_i M_i^{3/2}} \tag{4.58}$$

is a constant.

As before, we can simplify the min-max expression above by solving the associated linear program. This implies that the partitioning $\{F_i\}$ that attains the minimum satisfies

$$F_i = \frac{\frac{1}{\nu_{t_i}}}{\sum_j \frac{1}{\nu_{t_j}}} n^3 \tag{4.59}$$

for $1 \leq i \leq P$. Thus, we obtain a partition-independent Loomis-Whitney dominated lower bound

$$T_{H^*}^{DLA}(\{M_i\}, F) = \Omega\left(\max_{1\leq i\leq P} \nu_{t_i} F_i\right) = \Omega\left(\frac{n^3}{\sum_j \frac{1}{\nu_{t_j}}}\right). \tag{4.60}$$

While inequality (4.60) may not be as tight a bound as (4.49) in general, we will argue in Section 4.4 that it can be attained in the case of $O(n^3)$ matrix-matrix multiplication. This will imply that in that case, both bounds are equivalent and tight.

Energy-Optimal Partition of Work Similarly, we can explore a desire to determine the energy-optimal partition of the flops across the processors. In the case of dense $O(n^3)$ matrix-matrix multiplication, the input/output-dominated energy bound of Equation (4.3) becomes

$$E_{H^*}^{DLA}(\{M_i\}, \hat{M}, F) = \Omega\left(\min_{\sum F_i = n^3}\left(\sum_{i=1}^{P} \nu_{e_i} F_i + \left(\hat{\delta}\hat{M} + \epsilon_H\right) \max_{1\leq i\leq P}(\nu_{t_i} F_i)\right)\right) \tag{4.61}$$

where ν_{t_i} was defined previously and

$$\nu_{e_i} = \gamma_{e_i} + \frac{\beta_{e_i}}{M_i^{1/2}} + \frac{\alpha_{e_i}}{m_i M_i^{1/2}}.$$

Like the input/output-dominated situation discussed previously, the extra parameters of the energy bound result in a linear program that does not lend itself to a closed-form solution and a numerical approach to solving the linear program is required. The standard form of this linear program is nearly identical to the input/output-dominated energy-optimal partition, with the exception of using the constants ν instead of μ.

4.4 Optimal Heterogeneous Algorithms

Heterogeneous Matrix-Vector Multiplication

In this section we present an algorithm to compute square matrix-vector multiplication (GEMV) that is able to attain the lower bounds presented in Section 4.3. In the discussion that follows, we will assume that the objective is to attain the lower bound on runtime. A similar argument can be used to show that the algorithm attains the energy lower bound of Equation (4.56). While the cost of calculating an energy-optimal work partition requires the use of a linear program solver, the number of variables is dependent on the number of processors, $P << n$, and not the problem size $N = n^2$. Thus, the overhead of a solver is a lower-order term and we can still claim that the algorithm is also optimal if one wishes to minimize communication with respect to energy.

As a BLAS2 operation, square GEMV performs $F = 2n^2$ flops upon $N = n^2 + 2n$ data. Thus, we do two flops per word of data transferred and the value of g in Equation (4.53) is $\frac{1}{2}$. With this definition of g, we can rewrite the runtime lower bound of (4.55) as

$$T_{H^*}^{IO}(N, F) = \Omega\left(\frac{F}{\sum_j \frac{1}{\mu_{t_j}}}\right) = \Omega\left(\mu_{t_i} F_i\right) = \Omega\left(\gamma_{t_i} F_i + \beta_{t_i}\left(\frac{1}{2} F_i\right) + \alpha_{t_i}\left(\frac{1}{2}\frac{F_i}{m_i}\right)\right) \tag{4.62}$$

where F_i is defined as in equation (4.54) and we note that the runtime of processor i, $\mu_{t_i} F_i$, is constant for $1 \leq i \leq P$.

An runtime-optimal algorithm for square GEMV on a heterogeneous machine is presented as Algorithm 8.[5] This algorithm divides the work among the processors by partitioning the rows of the matrix A as shown in Figure 4.2. In this way, each processor computes a subset of the entries of the output vector and there are no write contentions (though each compute element must access the entire input vector). We assume $GEMV_i$ to be a sequential version of matrix-vector multiplication that is optimized to run efficiently on $proc_i$.

Algorithm 8 Heterogeneous matrix-vector multiplication

Require: Matrix $A \in^{n \times n}$, stored in row-wise order, vector $x \in^{n}$,

1: Measure $\alpha_{t_i}, \beta_{t_i}, \gamma_{t_i}, M_i$ for each $1 \leq i \leq P$ and set μ_{t_i} according to Equation (4.53) with $c = 1/2$
2: **for** $i = 1$ to P **do**
3: Set F_i according to Equation (4.54) where $F = 2n^2$
4: Choose splitting rows r_i ($1 \leq i \leq P$, with $r_0 = 0$) in matrix A such that $r_i - r_{i-1} \approx \frac{F_i}{2n}$
5: **end for**
6: **for all** proc$_i$ ($1 \leq i \leq P$) **parallel do**
7: Compute $GEMV_i(A(r_{i-1} : r_i - 1, :), x)$
8: **end for**

[5] To modify Algorithm 8 to minimize energy, one must also measure μ_{e_i} and set F_i according to the linear programming solution of Equation (4.56), as opposed to Equation (4.54).

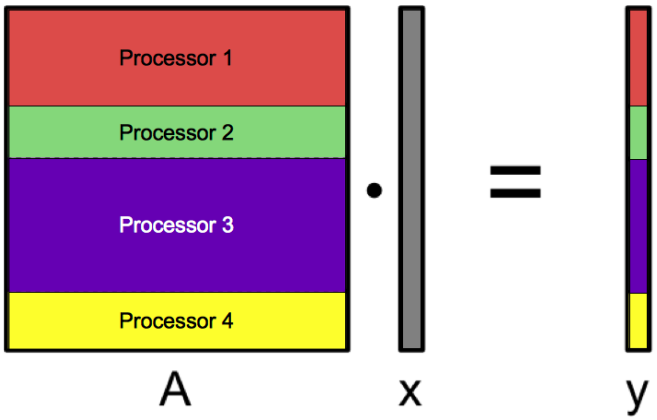


Figure 4.2: Example of heterogeneous matrix-vector data partitioning with 4 processors

For simplicity, we assume the matrix A is stored in row-wise order so that each $GEMV_i$ is performed on a contiguous block of memory. Further, we assume that each $GEMV_i$ accesses matrix entries contiguously (along rows). This implies that if the input vector does not fit in fast memory ($n > M_i$), then each input vector entry must be read from slow memory for each row of the matrix. While a blocked algorithm (with a blocked data structure) is more communication-efficient, the difference in the the number of words and messages transferred from slow memory is less than a constant factor ($2\times$).

To see that the parallel running time of this algorithm is within a constant factor of the lower bound given in equation (4.62), consider proc_i. It computes a matrix-vector product with a matrix of size $k_i \times n$ where $k_i = r_i - r_{i-1} \approx F_i/2n$. Thus, it performs $2k_in \approx F_i$ flops. Because the division of work is done by rows, even if $F_i/2n$ is not an integer, the processor is assigned no more than one row of extra work ($2n$ flops). Assuming each compute element is assigned at least one row of work, this implies that the number of flops done on the compute element is no more than $2\times$ that of the lower bound.

We now consider the communication costs. As mentioned above, if the flops are performed in row-wise order and the input vector x does not fit in fast memory, then two reads are required for each scalar multiplication. Since the cost of writing the output vector entries is a lower order term, the number of words transferred by proc_i is also $2k_in$. Since $2k_in$ is within $2\times$ of F_i, the number of words transferred is within $4\times$ of the lower bound. Since we are accessing the matrix and input vector entries in contiguous order, we can read the data in blocks of size about $M_i/2$ (in order to fit both blocks in fast memory at the same time). Thus, the number of messages is about $\frac{4k_in}{M_i}$ which is within $8\times$ of the lower bound.

Since each term (flop cost, bandwidth cost, latency cost) of the running time of proc_i is within a constant factor of the lower bound, the sum of the three terms is also within a constant factor. This argument holds for each compute element individually, so the maximum runtime over all compute elements (i.e., the parallel runtime) is within a constant factor of the lower bound given in equation (4.62). Thus, Algorithm 8 is asymptotically optimal within a factor of 8 of the lower

bound. Note that by using a blocked data structure for the matrix and corresponding algorithm, one could obtain a runtime within $4\times$ of the lower bound. Note that in practice, since our model can only approximate the true costs of complex hardware, it gives a design space for a fast algorithm, not a unique characterization of the optimal algorithm, and that autotuning, i.e. measuring the actual quantity being optimized, is needed for true optimality.

Heterogeneous $O(n^3)$ Matrix-Matrix Multiplication

In this section we present an algorithm to compute square matrix-matrix multiplication (GEMM) that attains the lower bounds presented in Section 4.3. As in the case of matrix-vector multiplication, we assume that the objective is to produce a runtime-optimal partition of work. With small modifications to the algorithm, it is able to attain the energy lower bound from Equation (4.61) as the cost of running a linear program solver scales with the number of processors, P.

For comparison to the upper bound analysis of runtime, we re-write inequality (4.60) in terms of three summands. Letting F_i be defined as in equation (4.59), and noting that $\nu_{t_i} F_i$ is constant for $1 \leq i \leq P$, we have the lower bound

$$T^{LW}_{H^*}(\{M_i\}, N, F) = \Omega\left(\frac{F}{\sum_j \frac{1}{\nu_{t_j}}}\right) = \Omega\left(\nu_{t_i} F_i\right) = \Omega\left(\gamma_{t_i} F_i + \beta_{t_i}\left(\frac{F_i}{M_i^{1/2}}\right) + \alpha_{t_i}\left(\frac{F_i}{m_i M_i^{1/2}}\right)\right). \tag{4.63}$$

We will base the algorithm on the square recursive matrix multiplication algorithm (see [33] for example). In this algorithm, each of the matrices are divided into four $\frac{n}{2} \times \frac{n}{2}$ submatrices and the blocked multiplication of these submatrices yields eight subproblems of $2(n/2)^3$ flops each, which can be solved recursively. We assume that n is a power of two in this section.

We require that the $n \times n$ input matrices A and B are stored in a block-recursive format. The block-recursive format [11, 63, 151] (also known as the bit interleaved layout, space-filling curve storage, or Morton ordering format) stores each of the four $\frac{n}{2} \times \frac{n}{2}$ submatrices contiguously, and the elements of each submatrix are ordered so that the smaller submatrices are each stored contiguously, and so on recursively. In this way, every subproblem within square recursive GEMM will be associated with contiguous data.

At a high level, the algorithm assigns subproblems of various sizes to each processor in a manner consistent with the runtime-optimal flop distribution as suggested by the lower bound.[6] It assigns as many subproblems at one level of recursion as possible before recursing to smaller subproblems. The flop assignments are represented as octal fractions in order to determine the number and size of subproblems to assign to each processor.

We will assume that for a given heterogeneous machine, the problem size is large enough such that the distribution of flops to compute elements according to equation (4.59) satisfies

$$F_i \geq (M_i/3)^{3/2} \tag{4.64}$$

[6]To modify Algorithm 9 to minimize energy, one must also measure ν_{e_i} and set F_i according to the linear programming solution of Equation (4.61), as opposed to Equation (4.59).

Algorithm 9 Heterogeneous $O(n^3)$ matrix-matrix multiplication

Require: $n \times n$ matrices A, B, stored in block-recursive order, n is a power of two
1: Measure $\alpha_{t_i}, \beta_{t_i}, \gamma_{t_i}, M_i$ and set ν_{t_i} according to Equation (4.58) for each $1 \leq i \leq P$
2: **for** $i = 1$ to P **do**
3: Set F_i according to Equation (4.59) where $F = n^3$
4: Set k_i to be the largest integer such that $3(n/2^{k_i})^2 \geq M_i$
5: Convert F_i/F into octal and round[1] to k_i^{th} digit: $0.d_1^{(i)} d_2^{(i)} \cdots d_{k_i}^{(i)}$
6: **end for**
7: Initialize $S = \{A \cdot B\}$
8: **for** $j = 1$ to $\max k_i$ **do**
9: Subdivide all problems in S into 8 subproblems according to square recursive GEMM
10: Assign $d_j^{(i)}$ subproblems to proc$_i$ and remove subproblems from S
11: **end for**
12: **for all** proc$_i$ **parallel do**
13: Compute assigned subproblems using square recursive GEMM
14: $proc_i$ contributes its computed subproblems to a sum-reduction forming C
15: **end for**
Ensure: Matrix $C = AB$, stored in block-recursive order

for each $1 \leq i \leq P$. Note that on a sequential machine, this degenerates to $3n^2 \geq M$, where the matrix multiplication problem (two input matrices and one output matrix) is too large to fit entirely in fast memory. This assumption may be violated for a small problem on a heterogeneous machine where one compute element is relatively slow (i.e., large ν_{t_i}) but has a large fast memory (i.e., large M_i).

To see that the parallel runtime is within a constant factor of the lower bound given in equation (4.63), consider proc$_i$. As with heterogeneous matrix-vector multiplication, we will argue that each of the three terms contributing to proc$_i$'s runtime are within constant factors of the corresponding terms in the lower bound.

Algorithm 9 does not assign exactly F_i flops to the proc$_i$. Instead, in line 5, F_i/F is rounded to a fraction with k_i octal digits.[7] Thus, the actual number of flops assigned is $U_i = \left(0.d_1^{(i)} d_2^{(i)} \cdots d_{k_i}^{(i)}\right)_8 \cdot F$, yielding

$$\frac{U_i}{F} - \frac{F_i}{F} \leq \frac{1}{8^{k_i}}.$$

Further, k_i is chosen in line 4 so that $3\left(\frac{n}{2^{k_i+1}}\right)^2 \leq M_i$ which implies $\frac{n^3}{8^{k_i}} \leq \left(\frac{4}{3}M_i\right)^{3/2}$. Since $F = n^3$, we have

$$U_i - F_i \leq \frac{n^3}{8^{k_i}} \leq \left(\frac{4}{3}M_i\right)^{3/2}.$$

[7] Rounding each of these fractions to a finite-digit octal representation such that the sum of octal fractions is exactly one is a nontrivial problem. For the purposes of this upper bound, we may assume that the rounding scheme always rounds up to the next k_i^{th} digit, in which case the sum will be slightly greater than one.

By our assumption in inequality (4.64), we have

$$\frac{U_i - F_i}{F_i} \leq 8 \, \frac{(M_i/3)^{3/2}}{F_i} \leq 8$$

and so the number of flops assigned to proc$_i$ in Algorithm 9 is within a constant factor of the flops given in the lower bound.

We now consider the communication costs for proc$_i$. By construction, the octal fraction representing the work assigned to proc$_i$ has no more than k_i digits. This implies that the smallest subproblem assigned to the compute element involves submatrices of size at least $n/2^{k_i} \times n/2^{k_i}$. Since $3(n/2^{k_i})^2 \geq M_i$, the smallest subproblem is too large to fit into fast memory. In this case, from [14], the number of words transferred by the square recursive GEMM on a sequential machine for each subproblem is $O(\#\text{flops}/M_i^{1/2})$, and with a block recursive data structure, the number of messages is $O(\#\text{flops}/m_i M_i^{1/2})$.[8] Thus, the total number of words transferred between proc$_i$ and slow memory is $O(U_i/M_i^{1/2})$, and the number of messages transferred is $O(U_i/m_i M_i^{1/2})$. Since U_i is within a constant factor of F_i, the number of words and messages transferred is within a constant factor of the lower bound.

We also note that for matrix multiplication, all subproblems are independent (ignoring the $O(n^2)$ work to sum the results of pairs of subproblems to form C), so there is no idle time on processors due to data dependencies. Thus, the running time of each compute element is the sum of the three terms of arithmetic and communication costs. Since each of these terms is within a constant factor of the lower bound for each compute element, the maximum runtime over all compute elements is no more than a constant factor larger than the lower bound given in inequality (4.63). Thus, Algorithm 9 is runtime-optimal.

An example execution of Algorithm 9 is demonstrated in Figure 4.3. In it, we have four heterogeneous processors {P1,P2,P3,P4} working in parallel on a problem. Before the execution of the problem, the hardware parameters of the processors were measured and the runtime was found to be minimized via the scheme in the first table. For example, the work partitioning scheme suggests that 1/4 of the work should be allocated to processor P1 (0.25). When converted to octal, these decimal representations of the work partition indicate a recursion scheme for the execution of the problem. This is shown in the smaller table. Thus, as processor P3 has an octal workload representation of $.32_8$ it executes three subproblems at the first level of recursion and two subproblems at the second level. This work allocation is shown graphically via the purple regions of the figure.

As a potential future extension, we note that a similar argument could show that any recursive matrix-matrix multiplication can be used to approximate the optimal partition of work within a constant factor. In particular, this applies to Strassen-like fast matrix multiplication algorithms that require $F = O(n^{\omega_0})$ floating point operations and each processor to move at least

$$W_{H_i}^{FMM}(M_i, N_i, F_i) = \Omega\left(\max\left(N_i, \frac{F_i}{M_i^{\omega_0/2-1}}\right)\right)$$

[8] The analysis of recursive GEMM in [14] is for a more general algorithm which handles rectangular matrices. In the case of square matrices, the algorithm reduces to square recursive GEMM.

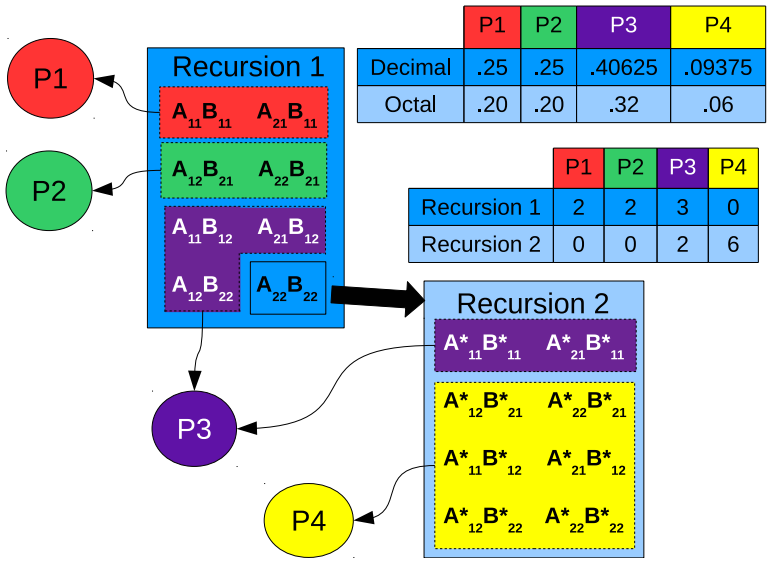


Figure 4.3: Heterogeneous matrix-matrix computation example execution on 4 processors

and

$$S_{H_i}^{FMM}(M_i, N_i, F_i) = \Omega\left(\max\left(\frac{N_i}{m_i}, \frac{F_i}{m_i M_i^{\omega_0/2-1}}\right)\right)$$

words and messages, respectively. Recall that FMM stands for "Fast Matrix Multiplication". Strassen's method [129], in particular, decomposes the problem into 7 recursive subproblems as opposed to the 8 subproblems used for the $O(n^3)$ recursive approach used in Algorithm 9. The optimal partition of flops would then be approximated as a base-7 fraction to obtain an optimal recursion scheme.

Chapter 5

Bounds on Communication, Runtime and Energy for Programs that Access Arrays

In Chapter 4, we reviewed related work on communication lower bounds for direct linear algebra [15], Strassen and Strassen-like fast matrix-matrix multiplication algorithms [20], dense matrix-vector multiplication and the $O(n^2)$ n-body problem [61]. We then applied these communication lower bounds to the models of runtime and energy from Chapter 3 to derive lower bounds on the runtime and energy consumption of sequential, distributed parallel, and heterogeneous machines. In Section 5.1, we apply a generalization of the bounds from [15] to runtime and energy bounds to the wider class of programs that reference arrays via linear expressions of the iteration variables (with some additional assumptions). Further, we highlight that a region of perfect strong scaling in runtime exists with no additional energy for a subset of these algorithms (Section 5.2).

5.1 Bounds on Programs that Reference Arrays

To address the above goals, the results of [15] and [17] have been generalized by Christ et al.[45] to problems of form

$$\begin{aligned} &\text{for all } \mathcal{I} \in \mathcal{Z} \subseteq \mathbb{Z}^u, \text{ in some order,} \\ &\quad \text{inner\_loop}(\mathcal{I}, (A_1, \ldots, A_q), (\phi_1, \ldots, \phi_q)) \end{aligned} \tag{5.1}$$

where $\mathcal{Z}$ is the iteration space, $\mathbb{Z}^u$ is the u-dimensional space of integers and the subroutine *inner_loop* represents a computation involving arrays $A_1, ..., A_q$ of dimensions $u_1, ..., u_m$ that are referenced by the corresponding subscripts $\phi_1(\mathcal{I}), ..., \phi_q(\mathcal{I})$ where ϕ_i are affine maps $\phi_j : \mathbb{Z}^u \to \mathbb{Z}^{u_j}$ for iteration $\mathcal{I} = (i_1, ..., i_u)$. The cardinality of the iteration space, $|\mathcal{Z}|$, will be represented as F in the following analysis, and is proportional to the number of work operations to be computed. For example, matrix-matrix multiplication $C = A * B$ has $(A_1, A_2, A_3) = (A, B, C)$, $\phi_1(\mathcal{I}) = \phi_1(i_1, i_2, i_3) = (i_1, i_3)$, $\phi_2(\mathcal{I}) = \phi_2(i_1, i_2, i_3) = (i_3, i_2)$, $\phi_3(\mathcal{I}) = \phi_3(i_1, i_2, i_3) = (i_1, i_2)$ and the function inner_loop() is defined as $A_3(\phi_3(\mathcal{I})) = A_3(\phi_3(\mathcal{I})) + A_1(\phi_1(\mathcal{I})) * A_2(\phi_2(\mathcal{I}))$.

Given this program, we wish to establish a lower bound on the number of read/write operations that must occur between fast and slow memory during execution. Because the work inside the loop is currently defined as a general function, the potential executions of *inner_loop* must be restricted in a reasonable manner. If executions of inner_loop$(\mathcal{I}, ...)$ for different values of $\mathcal{I}$ can not be interleaved (which, for example, forbids loop splitting) and all m array variables $A_j(\phi_j(\mathcal{I}))$ are accessed by each execution of inner_loop$(\mathcal{I}, ...)$, the program execution is defined to be a *legal sequential execution*. A similar definition holds for *legal parallel executions* where each processor's execution is a legal sequential execution (see [45] for more details regarding *legal sequential* and *legal parallel executions*). The analysis of Christ et al. any ignores data dependencies within the program. In other words, the lower bound of Christ el al. provides a lower bound for all execution orders, both correct, when they respect the dependencies, and incorrect, when they do not.

To express the lower bounds, we define the linear constraints on the vector of unknown scalars $(s_1, ..., s_q)$

$$\text{rank}(L) \leq \sum_{j=1}^{q} s_j \text{rank}(\phi_j(L)), \text{for all subgroups } L \leq \mathbb{Z}^u \tag{5.2}$$

where rank(L) is the cardinality of any maximal subset of Abelian group L that is linearly independent[1] and the "$L \leq \mathbb{Z}^u$" notation in this context indicates that L is a subgroup of $\mathcal{Z}^u$. This constraint set is finite as each rank is an integer in the range $[0, u]$. This results in a total of at most $(u+1)^{q+1}$ different inequalities. The values $s_j \in [0, 1]$ are obtained via a linear program that minimizes $s_{HBL} = \sum_{j=1}^{q} s_j$ subject to the constraints of (5.2).[2] We will refer to this linear program as sLP, and will use properties of its dual formulation in Section 5.2 to construct a communication-optimal algorithm for a subset of the programs described in (5.1).

If the execution is legal code and the constraints of Equation (5.2) are satisfied, Christ et al. lower bound the number of words W transferred between "fast" and "slow" memories on a sequential machine by

$$W_S^{HBL}(M, N, F) = \Omega\left(\max\left(N, \frac{F}{M^{s_{HBL}-1}}\right)\right) \tag{5.3}$$

where the problem size, $N = I + O$, is the sum of the number of input (I) and output words (O). We can derive a lower bound on the number of messages by dividing the bound in Equation (5.3) by the maximum message size (in terms of words read/written) m

$$S_S^{HBL}(M, N, F) = \Omega\left(\max\left(\frac{N}{m}, \frac{F}{mM^{s_{\text{HBL}}-1}}\right)\right). \tag{5.4}$$

In the parallel distributed machine model, we must divide by the total number of processors P to reflect the fraction of operations performed on each processor to obtain the *memory-dependent* lower bound for the DP model. However, if the number of processors increases beyond a certain magnitude, the *memory-independent* bound dominates. More details about the relationship

[1] The rank of an Abelian group is related to the concept of the dimension of a vector space.

[2] HBL is an acronym of 3 mathematicians' last names upon which this work is based: Hölder, Brascamp and Lieb.

		Memory Dependent	Memory Independent.
Programs Referencing Arrays	W_{DP}^{HBL}	$\Omega\left(\frac{F}{PM^{s_{\text{HBL}}-1}}\right)$	$\Omega\left(\left(\frac{F}{P}\right)^{1/s_{\text{HBL}}}\right)$
	$W_{DP}^{\text{link}HBL}$	$\Omega\left(\frac{F}{P^{s_{\text{HBL}}-1/d}M^{s_{\text{HBL}}-1}}\right)$	$\Omega\left(\frac{F^{1/s_{\text{HBL}}}}{P^{1-1/d}}\right)$

Table 5.1: Per-processor bounds (W_{DP}^{HBL}) ([45]) vs. the new contention bounds ($W_{DP}^{\text{link}HBL}$) on a d-dimensional torus for programs that reference arrays.

between the memory-dependent and independent communication lower bounds can be found in Chapter 4 and [21]. The two per-processor word bounds internode communication on DP machine models are combined to form a lower bound on word volume

$$W_{DP}^{HBL}(M,P,F) = \Omega\left(\max\left(\left(\frac{F}{P}\right)^{1/s_{HBL}}, \frac{F}{PM^{s_{\text{HBL}}-1}}\right)\right) \tag{5.5}$$

where the memory-independent lower bound is the first term in the max(), and the memory-dependent bound follows. The corresponding DP internode message bounds are derived in a similar manner to Equation (5.4).

As in Chapter 4, we also consider bounds on the link contention between nodes on a parallel distributed machine prior to stating final versions of the energy and runtime bounds for this class of problems. Now extend the definition of link contention to the case of programs that reference arrays via affine expressions:

$$W_{DP}^{\text{link}HBL}(M,P,F) = \Omega\left(\max_{r\in R}\frac{W_{DP}^{HBL}(M\cdot r, P/r, F)}{d\cdot r\cdot h_r(G_{Net})}\right)$$

where

$$R = \{r : 1 \le r \le P/2, \exists K \subseteq V s.t. |K| = r \text{ and } h_r(G_{Net}) = |\hat{E}(K, V\setminus K)|/|\hat{E}(K)|\},$$

and $h_r(G_{Net})$ is the small set expansion as defined in Equation (4.14). From this definition and the per-processor memory-dependent and memory-independent communication bounds in Equation (5.5), we can derive contention bounds for the larger class of problems bounded by Christ et al. The algebra for this derivation is nearly identical to that of Section 4.1, so we simply state the results in this instance. A summary of the link contention and per-processor communication lower bounds on the number of communication words can be seen in Table 5.1.

In Figure 5.1, we illustrate the regions within the torus dimension (d) vs. processors (P) plane where each of the four respective bounds dominate. As noted in Chapter 4, the dimension d of tori and mesh networks only takes integer values, so the black lines that delineate regions of bound dominance are for illustrative purposes. Noninteger values of d may correspond to other not-yet-analysed, or even designed, networks. This generalizes the discussion in Chapter 4, and several key observations are to be made in relation to Figure 5.1:

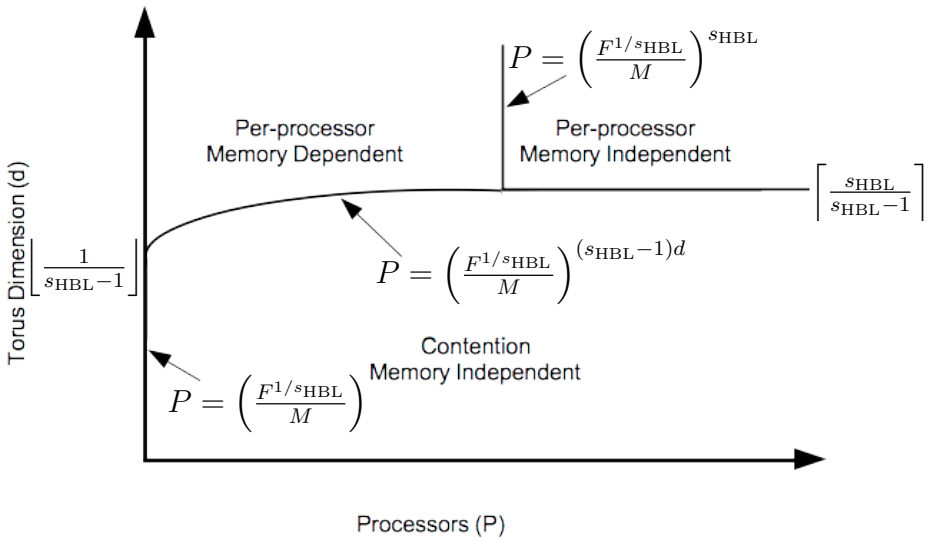


Figure 5.1: Relationship between the per-processor and contention communication lower bounds, with labels on each region indicating lower bound dominance. F and M are constants.

- If torus dimension $d > \lceil s_{\text{HBL}}/(s_{\text{HBL}} - 1)\rceil$, the network may be "good enough" and the per-processor bounds are dominant. In Section 5.2, we will prove that a region of perfect strong scaling in runtime with constant energy exists for a communication-optimal algorithm running on a good enough network when the per-processor memory-dependent bound is dominant.
- For toroidal topologies with $\left\lfloor \frac{1}{s_{\text{HBL}}-1} \right\rfloor < d < \lceil s_{\text{HBL}}/(s_{\text{HBL}}-1)\rceil$, the per-processor memory-dependent bound is dominant until $P > \left(F^{1/s_{\text{HBL}}}/M\right)^{(s_{\text{HBL}}-1)d}$. A region of perfect strong scaling with constant energy also exists on these networks when the per-processor memory-dependent bound dominates.
- As a direct consequence of the HBL bound discussed in Christ et al. [45], we find that $P = \left(F^{1/s_{\text{HBL}}}/M\right)$ will always be less than or equal to the minimum number of processors required to hold the problem. To show this, we reproduce the proof found in Ballard et al. [19]:

  Claim. Let there be an algorithm performing a computation of the form given by Equation (5.1) on P processors, each with local memory of size M, and assume a single copy of the input data is initially evenly distributed across processors. Then,

  $$\frac{F^{1/s_{\text{HBL}}}}{M} \leq \sum_{d=1}^{q} \frac{|\phi_j(\mathcal{Z})|}{M}.$$

  As the minimum number of processors required to hold the problem is the right-hand side of this inequality, we conclude that the memory-independent contention bound dominates the

memory-dependent contention bound as the two bounds are equivalent when $P = F^{1/s_{\text{HBL}}}/M$. □

Proof. The HBL bound discussed in Christ et al. [45], states (with certain assumptions) that

$$F = |\mathcal{Z}| \leq \prod_{j=1}^{q} |\phi_j(\mathcal{Z})|^{s_j}.$$

To detail an argument from Section 2 of [45], we present several greater upper bounds on F that will allow us to demonstrate the desired result:

$$F \leq \prod_{j=1}^{q} |\phi_j(\mathcal{Z})|^{s_j} \leq \prod_{j=1}^{q} \left(\max_{j=1}^{q} |\phi_j(\mathcal{Z})| \right)^{s_j}$$

$$= \left(\max_{j=1}^{q} |\phi_j(\mathcal{Z})| \right)^{\sum_{j=1}^{q} s_j} = \left(\max_{j=1}^{q} |\phi_j(\mathcal{Z})| \right)^{s_{\text{HBL}}}$$

As $\max_{j=1}^{q} x_j \leq \sum_{j=1}^{q} x_j$ if all $x_j \geq 0$,

$$F \leq \left(\max_{j=1}^{q} |\phi_j(\mathcal{Z})| \right)^{s_{\text{HBL}}} \leq \left(\sum_{j=1}^{q} |\phi_j(\mathcal{Z})| \right)^{s_{\text{HBL}}}$$

which proves the desired inequality if we take s_{HBL}th root of both sides and divide by M. □

Thus, the memory-dependent contention bound is always dominated by the memory-independent contention bound for programs of form shown in (5.1). This is a similar result to that of the contention discussion within Section 4.1. With this result, we can state communication lower bounds for the DP1 model that include contention bounds. As before, we differentiate between runtime and energy bounds because contention does not effect the dynamic energy components of the model:

$$W_{DP_t}^{HBL}(M, P, F, d) = \Omega\left(\max\left(\left(\frac{F}{P}\right)^{1/s_{HBL}}, \frac{F}{PM^{s_{\text{HBL}}-1}}, \frac{F^{1/s_{\text{HBL}}}}{P^{1-1/d}} \right) \right) \tag{5.6}$$

$$S_{DP_t}^{HBL}(M, P, F, d) = \Omega\left(\max\left(\frac{1}{m}\left(\frac{F}{P}\right)^{1/s_{HBL}}, \frac{F}{mPM^{s_{\text{HBL}}-1}}, \frac{F^{1/s_{\text{HBL}}}}{mP^{1-1/d}} \right) \right) \tag{5.7}$$

$$W_{DP_e}^{HBL}(M, P, F) = \Omega\left(\max\left(\left(\frac{F}{P}\right)^{1/s_{HBL}}, \frac{F}{PM^{s_{\text{HBL}}-1}} \right) \right) \tag{5.8}$$

$$S_{DP_e}^{HBL}(M, P, F) = \Omega\left(\max\left(\frac{1}{m}\left(\frac{F}{P}\right)^{1/s_{HBL}}, \frac{F}{mPM^{s_{\text{HBL}}-1}} \right) \right). \tag{5.9}$$

Interestingly, the solution to the dual of the linear problem to obtain s_{HBL} provides the set of optimal block sizes for the algorithm under certain assumptions, namely that the subscripts of each array are just subsets of the loop indices. This will be discussed further in Section 5.2 as we prove that a communication-optimal algorithm in the DP1 model has a region of perfect strong scaling with constant energy. As also elaborated in [45], Christ et al. note that it is currently not known if calculating the constraints of Equation (5.2) is a decidable problem. However, the researchers note that the desired extreme points (where the solution to the linear program can be found) of the linear program's feasible region may be defined via a computable subset of all the constraints in (5.2).

While the calculation of s_{HBL} is a difficult problem, a number of easy practical cases exist wherein s_{HBL} can be calculated easily. The most general case demonstrated by Christ et al. [45] is for the situation when subscripts of each array accessed within the *inner_loop* function are a subset of the loop indices. This case includes a number of common situations: linear algebra, the direct $O(n^2)$ n-body problem, database join and tensor contractions among others.

Sequential Model

For programs of the form discussed above in Algorithm 5.1, we can bound now runtime on the sequential machine model by substituting for $W_S^{HBL}(M,N)$ and $S_S^{HBL}(M,N)$ via Equations (5.3) and (5.4):

$$\begin{aligned} T_S^{HBL}(M,N,F) &= \gamma_t F + \beta_t W_S^{HBL} + \alpha_t S_S^{HBL} \\ &= \Omega\left(\gamma_t F + \beta_t \max\left(N, \frac{F}{M^{s_{HBL}-1}}\right) + \alpha_t \max\left(\frac{N}{m}, \frac{F}{mM^{s_{HBL}-1}}\right)\right). \end{aligned} \tag{5.10}$$

Similarly, we establish a lower bound for energy

$$\begin{aligned} E_S^{HBL}(M,\hat{M},N,F) &= \gamma_e F + \beta_e W_S^{HBL} + \alpha_e S_S^{HBL} \\ &\quad + (\delta_e \hat{M} + \epsilon_e)\left(\gamma_t F + \beta_t W_S^{HBL} + \alpha_t S_S^{HBL}\right) \\ &= \Omega\left(\gamma_e F + \beta_e \max\left(N, \frac{F}{M^{s_{HBL}-1}}\right) + \alpha_e \max\left(\frac{N}{m}, \frac{F}{mM^{s_{HBL}-1}}\right)\right. \\ &\quad + (\delta_e \hat{M} + \epsilon_e) \\ &\quad \left.* \left(\gamma_t F + \beta_t \max\left(N, \frac{F}{M^{s_{HBL}-1}}\right) + \alpha_t \max\left(\frac{N}{m}, \frac{F}{mM^{s_{HBL}-1}}\right)\right)\right). \end{aligned} \tag{5.11}$$

Distributed Parallel Model 1

The bound for the parallel distributed model is a bit more interesting in that there may be three different expressions that comprise the maximal lower bound on word traffic, which include the

per-processor memory-independent and dependent bounds as well as the memory-independent contention bound presented in Table 5.1. So, the runtime bound is:

$$T_{DP1}^{HBL}(M,P,F) = \frac{\gamma_t F}{P} + \beta_t W_{DP_t}^{HBL} + \alpha_t S_{DP_t}^{HBL}$$

$$= \Omega\left(\frac{\gamma_t F}{P} + \beta_t \max\left(\left(\frac{F}{P}\right)^{1/s_{HBL}}, \frac{F}{PM^{s_{\text{HBL}}-1}}, \frac{F^{1/s_{\text{HBL}}}}{P^{1-1/d}}\right)\right.$$
$$\left.+\alpha_t \max\left(\frac{1}{m}\left(\frac{F}{P}\right)^{1/s_{HBL}}, \frac{F}{mPM^{s_{\text{HBL}}-1}}, \frac{F^{1/s_{\text{HBL}}}}{mP^{1-1/d}}\right)\right) \tag{5.12}$$

The energy bound is also nearly identical to the serial model, but we account for the additional hardware by multiplying by the total number of processors, P:

$$E_{DP1}^{HBL}(M,P,F) = \Omega\left(P\left(\frac{\gamma_e F}{P} + \beta_e W_{DP_e}^{HBL} + \alpha_e S_{DP_e}^{HBL}\right.\right.$$
$$\left.\left.+(\delta_e M + \epsilon_e)T_{DP1}^{HBL}\right)\right)$$

$$= \Omega\Bigg(P\Bigg(\frac{\gamma_e F}{P} + \beta_e \max\left(\left(\frac{F}{P}\right)^{1/s_{HBL}}, \frac{F}{PM^{s_{\text{HBL}}-1}}\right)$$
$$+ \alpha_e \max\left(\frac{1}{m}\left(\frac{F}{P}\right)^{1/s_{HBL}}, \frac{F}{mPM^{s_{\text{HBL}}-1}}\right)$$
$$+ (\delta_e M + \epsilon_e)\left(\frac{\gamma_t F}{P} + \beta_t \max\left(\left(\frac{F}{P}\right)^{1/s_{HBL}}, \frac{F}{PM^{s_{\text{HBL}}-1}}, \frac{F^{1/s_{\text{HBL}}}}{P^{1-1/d}}\right)\right.$$
$$\left.+\alpha_t \max\left(\frac{1}{m}\left(\frac{F}{P}\right)^{1/s_{HBL}}, \frac{F}{mPM^{s_{\text{HBL}}-1}}, \frac{F^{1/s_{\text{HBL}}}}{mP^{1-1/d}}\right)\right)\Bigg)\Bigg). \tag{5.13}$$

Distributed Parallel Model 2

In the case of DP2, we follow a similar procedure to the derivations in Chapter 4. We begin by substituting Equations (5.3), (5.4), (5.6) and (5.7) into the DP2 runtime expression of Equation (3.5):

$$T_{DP2}^{HBL}(M_0, M_1, P, N, F) = \Omega\left(\gamma_{t_1}\frac{F}{P} + \beta_{t_1}\max\left(\frac{N}{P}, \frac{F}{PM_1^{s_{HBL}-1}}\right)\right.$$
$$+ \alpha_{t_1}\max\left(\frac{N}{Pm}, \frac{F}{mPM_1^{s_{\text{HBL}}-1}}\right)$$
$$+ \beta_{t_0}\max\left(\left(\frac{F}{P}\right)^{1/s_{HBL}}, \frac{F}{PM_0^{s_{\text{HBL}}-1}}, \frac{F^{1/s_{\text{HBL}}}}{P^{1-1/d}}\right)$$
$$\left.+\alpha_{t_0}\max\left(\frac{1}{m}\left(\frac{F}{P}\right)^{1/s_{HBL}}, \frac{F}{mPM_0^{s_{\text{HBL}}-1}}, \frac{F^{1/s_{\text{HBL}}}}{mP^{1-1/d}}\right)\right). \quad (5.14)$$

The energy lower bound then follows by substituting Equations (5.3), (5.4), (5.8), (5.9) and (5.14) into the DP2 energy expression of Equation (3.6):

$$E_{DP2}^{HBL}(M_0, M_1, P, N, F) = \Omega\Bigg(P\Bigg(\gamma_{e_1}\frac{F}{P} + \beta_{e_1}\max\left(\frac{N}{P}, \frac{F}{PM_1^{s_{HBL}-1}}\right)$$
$$+ \alpha_{e_1}\max\left(\frac{N}{Pm}, \frac{F}{mPM_1^{s_{\text{HBL}}-1}}\right)$$
$$+ (\delta_{e_0}M_0 + \epsilon_{e_0})\left(\gamma_{t_1}\frac{F}{P} + \beta_{t_1}\max\left(\frac{N}{P}, \frac{F}{PM_1^{s_{HBL}-1}}\right)\right.$$
$$\left.+\alpha_{t_1}\max\left(\frac{N}{Pm}, \frac{F}{mPM_1^{s_{\text{HBL}}-1}}\right)\right)$$
$$+ \beta_{t_0}\max\left(\left(\frac{F}{P}\right)^{1/s_{HBL}}, \frac{F}{PM_0^{s_{\text{HBL}}-1}}, \frac{F^{1/s_{\text{HBL}}}}{P^{1-1/d}}\right)$$
$$\left.+\alpha_{t_0}\max\left(\frac{1}{m}\left(\frac{F}{P}\right)^{1/s_{HBL}}, \frac{F}{mPM_0^{s_{\text{HBL}}-1}}, \frac{F^{1/s_{\text{HBL}}}}{mP^{1-1/d}}\right)\right)$$
$$\left.+\zeta|\hat{E}(G_{Net})|\right). \quad (5.15)$$

Heterogeneous Model

Due to the construction of the heterogeneous machine model, the HBL bound applies to communication traffic between each processor and global memory. Together with the models for heterogeneous runtime (Equation (3.7)) and energy (Equation (3.8)), we can derive bounds. We can obtain the heterogeneous bounds $W_{H_i}^{HBL}$ and $S_{H_i}^{HBL}$ by subscripting the HBL sequential lower bounds (Equations (5.3) and (5.4)) to differentiate the different heterogeneous processing elements:

$$W_{H_i}^{HBL}(M_i, N_i, F_i) = \Omega\left(N_i, \frac{F_i}{M_i^{s_{HBL}-1}}\right)$$

and

$$S_{H_i}^{HBL}(M_i, N_i, F_i) = \Omega\left(\frac{N_i}{m_i}, \frac{F_i}{m_i M_i^{s_{HBL}-1}}\right).$$

Then, by substituting these bounds into the heterogeneous runtime and energy models, we obtain

$$T_H^{HBL}(\{M_i\}, \{N_i\}, \{F_i\}) = \max_{1\le i\le P}\left(\gamma_{t_i}F_i + \beta_{t_i}W_{H_i}^{HBL} + \alpha_{t_i}S_{H_i}^{HBL}\right) \tag{5.16}$$

and

$$\begin{aligned} E_H^{HBL}(\{M_i\}, \hat{M}, \{N_i\}, \{F_i\}) = \Omega\Bigg(&\sum_{i=1}^{P}\Big[\gamma_{e_i}F_i + \beta_{e_i}W_{H_i}^{HBL} + \alpha_{e_i}S_{H_i}^{HBL} \\ &+\delta_{e_i}M_i\left(\gamma_{t_i}F_i + \beta_{t_i}W_{H_i}^{HBL} + \alpha_{t_i}S_{H_i}^{HBL}\right)\Big] \\ &+(\hat{\delta}\hat{M} + \epsilon_H)\max_{1\le i\le P}\left(\gamma_{t_i}F_i + \beta_{t_i}W_{H_i}^{HBL} + \alpha_{t_i}S_{H_i}^{HBL}\right)\Bigg). \end{aligned} \tag{5.17}$$

As discussed in Section 4.3, we can obtain lower bounds on heterogeneous runtime and energy by minimizing over all partitions of the work, F.

Example: Energy Lower Bound for Matrix-matrix Multiplication

To be more concrete, the lower bound of Equation (5.17) applies to BLAS3 [31] operations such as n-by-n dense-matrix-matrix-multiplication as well as most direct linear algebra algorithms. We assume the DP1 model, and here show that the energy lower bound derived in the previous chapter can be obtained via the generalized theory. For classical matrix-matrix multiplication, we note that the contention bound is subsumed in such cases when a torus network topology is of dimension 2 or 3 (the latter case applies for algorithms that trade memory for reduced communication [19]) and assume that the problem is of sufficient size that the memory-dependent bound is dominant. In the situation of $O(n^3)$ matrix-matrix multiplication, we can rewrite Equation (5.17) as

$$E_H^{HBL}(\{M_i\}, \hat{M}, \{N_i\}, \{F_i\}) = \Omega\left(\sum_{i=1}^{P}(F_i(\nu_{e_i} + \nu_{t_i}\delta_{e_i}M_i)) + (\hat{\delta}\hat{M} + \epsilon_H)\max_{1\le i\le P}F_i\nu_{t_i}\right) \tag{5.18}$$

where

$$\nu_{t_i} = \gamma_{t_i} + \frac{\beta_{t_i}}{M_i^{1/2}} + \frac{\alpha_{t_i}}{mM_i^{1/2}}$$

and

$$\nu_{e_i} = \gamma_{e_i} + \frac{\beta_{e_i}}{M_i^{1/2}} + \frac{\alpha_{e_i}}{mM_i^{1/2}}$$

are constants, $s_{\text{HBL}} = 3/2$ and $\sum_i F_i = 2n^3$. We can also write both the above energy lower bound as an integer program, and solve the associated linear program to obtain an approximation of the optimal flop partition to guide algorithms (see [17] for more details). This approach was used to construct an energy and communication-optimal algorithm for matrix-matrix multiplication in the previous chapter, but a similar strategy could also be used to construct optimal work partitions for the much larger class of problems bounded in this chapter. The major challenge in a constructing a generalized approach to communication or energy-optimal algorithms for such problems is in partitioning the input data of N words into blocks $\{N_i\}$ that attain this optimal partition of F, $\{F_i\}$.

5.2 Perfect Strong Scaling in the Distributed Machine Model

In [126], Solomonik and Demmel presented an algorithm on the DP1 model for classical $O(n^3)$ matrix-matrix multiplication that could achieve perfect strong scaling in runtime by utilizing additional memory for replicating data to asymptotically reduce the overhead of communication. By perfect strong scaling in runtime, we mean that an increase in the number of processors by a factor c results in a identical reduction in runtime for a fixed problem size. Later results also used data replication to show that new algorithms for the $O(n^2)$ n-body problem [61] and Strassen's matrix-matrix multiplication algorithm [98] also achieve this perfect strong scaling characteristic. This region of perfect strong scaling has been demonstrated empirically for these algorithms as well as a new algorithm for rectangular matrix-matrix multiplication [53]. The ability to achieve this scaling with constant energy was proven theoretically by Demmel, Gearhart, Lipshitz and Schwartz in [55] and also detailed in Chapter 4.

In this section, we reproduce and generalize the proof of [45] that presents an algorithm within the distributed parallel machine model that achieves a region of perfect strong scaling in runtime when the array subscript expressions ϕ_l all choose subsets K_l of the loop indices $\mathcal{I}$. In addition to slightly generalizing the argument of Christ et al., we also show that this region of perfect strong scaling in runtime occurs at constant energy.

For per-processor communication bounds to apply, we must ensure that the parallel algorithm distributes the workload so that every processor performs $\Omega(1/P)$ of the computation, and distributes the input and output data such that every processor stores $O(1/P)$ of the data. For simplicity, let us assume that the iteration space $\mathbb{Z}$ is a dense cube; i.e. that each index variable $i_k \in \mathcal{I}(k = 1 : u, u = dim(\mathcal{I}))$ ranges from $0 : n - 1$.[3] We also ignore data dependencies that may be carried across loop iterations. Thus, we consider programs of this form

$$\begin{aligned} &\text{for } i_1 = 0 : n-1, \text{for } i_2 = 0 : n-1, ..., \text{for } i_u = 0 : n-1, \\ &\qquad \text{inner\_loop}((i_1, i_2, ..., i_u), (A_1, \ldots, A_q), (\phi_1, \ldots, \phi_q)). \end{aligned} \tag{5.19}$$

[3]The bounds can also be derived if we assume the iteration space to be encompassed by this d-dimensional cube, thus allowing the size of each dimension to differ.

In the sequential model, Christ et al. [45] prove that "tiling" programs of form (5.19) into this form

$$
\begin{array}{l}
\text{// A blocked communication-optimal algorithm} \\
\text{// for programs that access arrays via subsets of the iteration space} \\
\text{for } j_1 = 0 : M^{x_1} : n-1, \text{for } j_2 = 0 : M^{x_2} : n-1, ..., \text{for } j_u = 0 : M^{x_u} : n-1, \\
\quad \text{for } k_1 = 0 : M^{x_1} - 1, \text{for } k_2 = 0 : M^{x_2} - 1, ..., \text{for } k_u = 0 : M^{x_u} - 1, \\
\quad\quad (i_1, i_2, ..., i_u) = (j_1, j_2, ..., j_u) + (k_1, k_2, ..., k_u) \\
\quad\quad \text{inner\_loop}((i_1, i_2, ..., i_u), (A_1, \ldots, A_q), (\phi_1, \ldots, \phi_q))
\end{array}
\tag{5.20}
$$

can attain the serial communication lower bounds (5.3) and (5.4) if permitted by loop carried dependencies and if the parameters $x_1, x_2, ..., x_u$ are chosen to be the solution of the dual xLP of linear program sLP (previously mentioned in Section 5):

Theorem 7.1 of [45]: *Suppose that the linear program (sLP) for s_{HBL} is feasible and that each array $A_l (l = 1...q)$ is indexed via a subset K_l of the loop indices. We define matrix $\Delta \in \mathbb{R}^{q \times u}$ as a matrix with $\Delta_{li} = 1$ if $i \in K_l$ and 0 otherwise. Then the dual linear program "xLP" for $x = (x_1, ..., x_u)^T$*

$$(xLP) \quad maximize\ 1_u^T \cdot x\ subject\ to\ 1_q \geq \Delta \cdot x,$$

is also feasible, and using the values of x in the tiled code (5.20) lets it attain the communication lower bound of $\Omega(n^u / M^{s_{\text{HBL}}-1})$ words moved. By duality, the solution x of xLP satisfies $1_u^T \cdot x = 1_q^T \cdot s = s_{\text{HBL}}$.

The pseudocode of (5.20) partitions the iteration space of n^u points into $\prod_{i=1}^{u} n/M^{x_i} = n^u/M^{s_{HBL}}$ "bricks" of size $\prod_{i=1}^{u} M^{x_i} = M^{s_{\text{HBL}}}$ that are computed with each iteration of the index variables (j). In the following, we argue that a similar tiling approach to programs of the form shown in (5.19) (with the same assumption that arrays are accessed via subsets of the index variables) yields a tight upper bound on communication traffic for the distributed parallel machine model.

If we consider the DP1 model and assume that the dimension of the torus network and F is of sufficient size such that the memory-dependent lower bound is dominant, the number of words transferred on a distributed parallel machine with P processors is lower bounded by

$$W_{DP}^{HBL}(M, P, F) = \Omega\left(\frac{F}{PM^{s_{HBL}-1}}\right). \tag{5.21}$$

As we ignore data dependencies, bricks of the algorithm in (5.20) may be executed in parallel. To construct an optimal parallel algorithm, let us first consider the minimum amount of memory required to hold the problem. As in a similar analysis by [45], we construct an algorithm that is communication-optimal for programs of type shown in (5.19). From this, we know that n^{u_l} words of memory are required to store A_l where u_l is the dimension of A_l. The total amount of required memory is $\sum_{l=1}^{q} n^{u_l} = N$ and the amount per processor is bounded below by $M \geq N/P$.

Next, we describe how the blocks of data required to compute a brick are initially laid out on the processors such that the data distribution assumption is attained. Since A_l has subscripts given by indices in S_l, we may partition A_l by blocking the subscripts i_k, for $k \in S_l$, into blocks of size M^{x_k}, so that each block contains $\prod_{k \in S_l} M^{x_k} = M^{\sum_{k \in S_l} x_k} = M^{(\Delta \cdot x)_l} \leq M$ entries and may thus be stored on a single processor. For each array $A_l, 1 \leq l \leq q$, we assign the $n^{u_l}/M^{\sum_{k \in S_l} x_k}$ blocks cyclically across the P processors. We assume that the number of blocks of each array is sufficient such that every processor stores $O(1/P)$ of the data and that block assignment is load balanced, i.e. so that all processors do not access a needed block from the same processor on a given iteration. With this initial data layout, we now present an algorithm to compute the problem in parallel:

Algorithm 10 Communication-Optimal Parallel Algorithm for Product Case Problems

1: **for all** $(j_1, ..., j_u)$ where $j_i \in [1, n/M^{x_i}]$ **do in parallel**
2: **for all** $l = 0 : q - 1$ **do**
3: get block of array A_l from correct processor
4: **end for**
5: **for all** $k_1 = 0 : M^{x_1} - 1, k_2 = 0 : M^{x_2} - 1, ..., k_u = 0 : M^{x_u} - 1$ **do**
6: $(i_1, i_2, ..., i_u) = (j_1 M^{x_1}, ..., j_u M^{x_u}) + (k_1, k_2, ..., k_u)$
7: inner_loop$((i_1, i_2, ..., i_u), (A_1, \ldots, A_q), (\phi_1, \ldots, \phi_q))$
8: **end for**
9: **end for**

The pseudocode of Algorithm 10 partitions the iteration space of n^u points into $\prod_{i=1}^{u} n/M^{x_i} = n^u/M^{s_{HBL}}$ "bricks" of size $\prod_{i=1}^{u} M^{x_i} = M^{s_{\text{HBL}}}$, P of which are computed per PARFOR iteration of the algorithm in Algorithm10. The algorithm is clearly load-balanced (assuming we have at least P bricks) and the computation of each brick requires $O(M)$ communication (see [45]). Thus, Algorithm 10 attains the desired lower bound on word traffic (Equation (5.21)).

Now we wish to show that this algorithm is able to achieve perfect strong scaling in runtime by utilizing the additional memory provided by adding additional processors to the system. To begin, we write $M = cN/P$, where c is the number of copies of the data we will use; $c = 1$ corresponds to the minimum memory necessary to hold the problem. Combining this expression for M with the number of bricks, we may write

$$\text{\# bricks } = \left[\frac{n^{u/s_{\text{HBL}}} P}{Nc} \right]^{s_{\text{HBL}}}.$$

We now divide the P processors into c groups of P/c processors each. Since we assume there is enough memory PM for c copies of the input data, each group of P/c processors will have its own copy of the input data and no inter-group communication will be required as we chose to ignore data dependencies. So, each group of P/c processors would require

$$\frac{n^u P^{s_{\text{HBL}}-1}}{N^{s_{\text{HBL}}} c^{s_{\text{HBL}}-1}}$$

parallel phases of the parallel blocked Algorithm 10 to complete all the bricks. However, we have c groups of processors. Thus, we actually need

$$\frac{n^u P^{s_{\text{HBL}}-1}}{N^{s_{\text{HBL}}} c^{s_{\text{HBL}}}} = \frac{K_{min}}{c^{s_{\text{HBL}}}} \tag{5.22}$$

parallel phases to complete the problem and $K_{min} = n^u P^{s_{\text{HBL}}-1}/N^{s_{\text{HBL}}}$ represents the number of phases with the minimal amount of memory (or data replications, i.e. $c = 1$).

If we wish to group the processors P in to groups of P/c, we must partition the iteration space into contiguous chunks of work that allow for the above discussion to hold. To do this, we define a "superbrick" to consist of

$$\left(\frac{N^{1/u}}{M^{x_k/s_{\text{HBL}}}}\right)$$

contiguous bricks in each direction for

$$\prod_{k=1}^{u}\left(\frac{N^{1/u}}{M^{x_k/s_{\text{HBL}}}}\right) = \frac{(N^{1/u})^u}{(M^{\sum_{k=1}^{u} x_k})^{1/s_{\text{HBL}}}} = \frac{N}{(M^{s_{\text{HBL}}})^{1/s_{\text{HBL}}}} = \frac{N}{M} = \frac{P}{c}$$

total bricks per superbrick as $M = cN/P$. Thus, each group of P/c processors executes one superbrick of P/c contiguous bricks per parallel iteration.

Suppose that we initially use the minimum number of processors required to hold the problem: $P_{min} = N/M$. This is the situation when there is only enough memory for $c = 1$ copies of the input data. Then according to Equation (5.22) we need K_{min} parallel phases to process all the bricks. We also know that the total number of iterations executed by the entire machine is $F = K_{min} P_{min} M^{s_{\text{HBL}}}$ as each brick contains $M^{s_{\text{HBL}}}$ operations. According to Equation (5.12) (assuming the memory-dependent per-processor bound dominates), the runtime bound with P_{min} processors is

$$\begin{aligned} T_{DP1}^{HBL}(M, P_{min}, F) &= \Omega\left(\gamma_t \frac{F}{P_{min}} + \beta_t \frac{F}{P_{min} M^{s_{\text{HBL}}-1}} + \alpha_t \frac{F}{m P_{min} M^{s_{\text{HBL}}-1}}\right) \\ &= \Omega\left(K_{min} M^{s_{\text{HBL}}}\left(\gamma_t + \frac{\beta_t}{M^{s_{\text{HBL}}-1}} + \frac{\alpha_t}{m M^{s_{\text{HBL}}-1}}\right)\right). \end{aligned}$$

Note that we must divide F by the number of processors P_{min} to obtain the number of iterations executed by a single processor. Now, suppose that we scale the number of processors by c; i.e. $P = cP_{min}$ so that we have memory space for c copies of the input data. Now, the number of parallel phases is

$$\frac{n^u (cP_{min})^{s_{HBL}-1}}{N^{s_{HBL}} c^{s_{HBL}}} = \frac{n^u P_{min}^{s_{HBL}-1}}{N^{s_{HBL}} c} = \frac{K_{min}}{c}.$$

Note that by scaling by c processors, we are able to reduce the amount of parallel phases by the same factor. With this in mind, the runtime bound with $P = cP_{min}$ processors becomes

$$T_{DP1}^{HBL}(M,P,F) = \frac{K_{min}M^{s_{\text{HBL}}}}{c}\left(\gamma_t + \frac{\beta_t}{M^{s_{\text{HBL}}-1}} + \frac{\alpha_t}{mM^{s_{\text{HBL}}-1}}\right) = \frac{T_{DP1}^{HBL}(M,P_{min},F)}{c}. \tag{5.23}$$

This result demonstrates that a region of perfect strong scaling in runtime exists for a certain range of processors (and some assumptions on network topology). Furthermore, to generalize an argument used in [55], the energy lower bound in Equation (5.13) (assuming the memory-dependent per-processor bound dominates) is such that each term is dependent on either the number of flops, number of words, or number of messages. So, the energy to run on P_{min} processors is

$$E_{DP1}^{HBL}(M,P_{min},F) = \Omega\left(P_{min}\left(\gamma_e\frac{F}{P_{min}} + \beta_e\frac{F}{P_{min}M^{s_{\text{HBL}}-1}} + \alpha_e\frac{F}{mP_{min}M^{s_{\text{HBL}}-1}} + (\delta_e M + \epsilon_e)T_{DP1}^{HBL}(M,P_{min},F)\right)\right)$$

$$= \Omega\left(P_{min}\left(K_{min}M^{s_{\text{HBL}}}\left(\gamma_e + \frac{\beta_e}{M^{s_{\text{HBL}}-1}} + \frac{\alpha_e}{mM^{s_{\text{HBL}}-1}}\right) + (\delta_e M + \epsilon_e)T_{DP1}^{HBL}(M,P_{min},F)\right)\right)$$

Now, if we run on $P = cP_{min}$ processors the energy becomes

$$E_{DP1}^{HBL}(M,cP_{min},F) = \Omega\Bigg(cP_{min}\Bigg(\frac{K_{min}M^{s_{\text{HBL}}}}{c}\left(\gamma_e + \frac{\beta_e}{M^{s_{\text{HBL}}-1}} + \frac{\alpha_e}{mM^{s_{\text{HBL}}-1}}\right) + \frac{(\delta_e M + \epsilon_e)T_{DP1}^{HBL}(M,P_{min},F)}{c}\Bigg)\Bigg) = E_{DP1}^{HBL}(M,P_{min},F) \tag{5.24}$$

if we recall that the number of parallel phases is now K_{min}/c and the strong scaling property of T_{DP1}^{HBL} shown in (5.23). This done, we see that the amount of energy required to strong scale perfectly in runtime is constant. To summarize the runtime (T_{DP1}^{HBL}) and energy (E_{DP1}^{HBL}) to run on $P = cP_{min}$ processors is:

$$T_{DP1}^{HBL}(M,P,F) = \frac{T_{DP1}^{HBL}(M,P_{min},F)}{c}$$
$$E_{DP1}^{HBL}(M,P,F) = E_{DP1}^{HBL}(M,P_{min},F).$$

Now that we've demonstrated the utility of extra copies of the input data to reduce communication, we must explore an upper bound on the value of the replication factor, c. To be load balanced, there must be at least P bricks available for computation. Thus, $F/M^{s_{\text{HBL}}} = n^u/M^{s_{HBL}} \geq P$ recalling that $F = n^u$. Thus allows us to obtain an upper bound on the size of M, and consequently a

lower bound on the number of words according to Equation (5.21). This expression has been discussed previously, and is the parallel memory-independent per-processor bound in Equation (5.13). Recalling that $M = cN/P$, we obtain

$$\frac{F^{1/s_{HBL}}}{P^{1/s_{HBL}}} \geq \frac{cN}{P}$$

$$c \leq P^{1-1/s_{HBL}} \frac{F^{1/s_{HBL}}}{N} \tag{5.25}$$

And substituting $F = n^u$, we obtain

$$c \leq P^{1-1/s_{HBL}} \frac{n^{u/s_{HBL}}}{N} \tag{5.26}$$

as an upper bound on the size of c.

In the situation where a toroidal or mesh network is of degree d such that $\left\lfloor \frac{1}{s_{\text{HBL}}-1} \right\rfloor < d < \lceil s_{\text{HBL}}/(s_{\text{HBL}}-1) \rceil$, a similar argument to the above can be made to show that a region of perfect strong scaling in runtime with no additional energy also exists with the memory-dependent per-processor bound ($W_{DP}^{HBL} = \Omega(F/PM^{s_{\text{HBL}}-1})$) eventually dominated by the memory-independent contention bound ($W_{DP}^{\text{link}HBL} = \Omega(F^{1/s_{\text{HBL}}}/P^{1-1/d})$) when c is such that inequality (5.25) no longer holds.

Chapter 6

Applications of Bounds on Specific Machine Models

The bounds on energy presented in Chapters 4 and 5 allow us to consider a set of interesting problems that may be of use to algorithm designers and autotuning software:

1. What is the minimum energy required for a computation?
2. Given a maximum allowed runtime T, what is the minimum energy E needed to achieve it?
3. Given a maximum energy budget E, what is the minimum runtime T that we can attain?
4. The ratio $H = E/T$ gives us the average total power required to run the algorithm. Can we minimize the average power consumed?
5. Given a bound on average power, can we minimize energy or runtime?

In this chapter, we first provide a high-level overview of our analysis so that readers may build an intuition of the relationships between problem size, memory utilization, and number of processors on distributed parallel machines (Section 6.1). Next, we describe two examples of distributed parallel machines that will be analyzed within this chapter (Section 6.2). Then, in an extension of collaborative work with Ballard et. al [55], we analyze classical $O(n^3)$ matrix-matrix multiplication, the $O(n^2)$ n-body problem and the class of programs that reference arrays via subsets of their iteration variables (see Chapter 5 and Christ et al. [45] for more details) and visualize energy efficiency scaling on two hypothetical machine architectures with the presence of energy and runtime constraints (Sections 6.3 and 6.4). Scripts used to generate the figures presented in this chapter can be found at `https://github.com/agearh/dissertation.git`.

6.1 Overview

If we assume that the hardware parameters (γ_t, γ_e, β_t, β_e, α_t, α_e, ϵ_e) of the machine model are fixed, the interesting parameter becomes that of the total amount of utilized local memory, M.

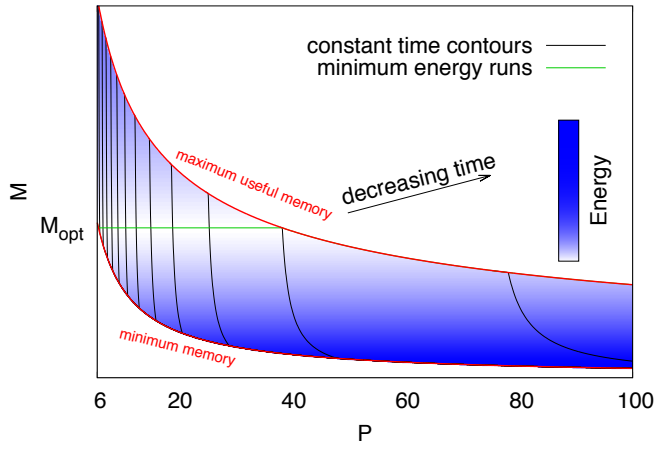


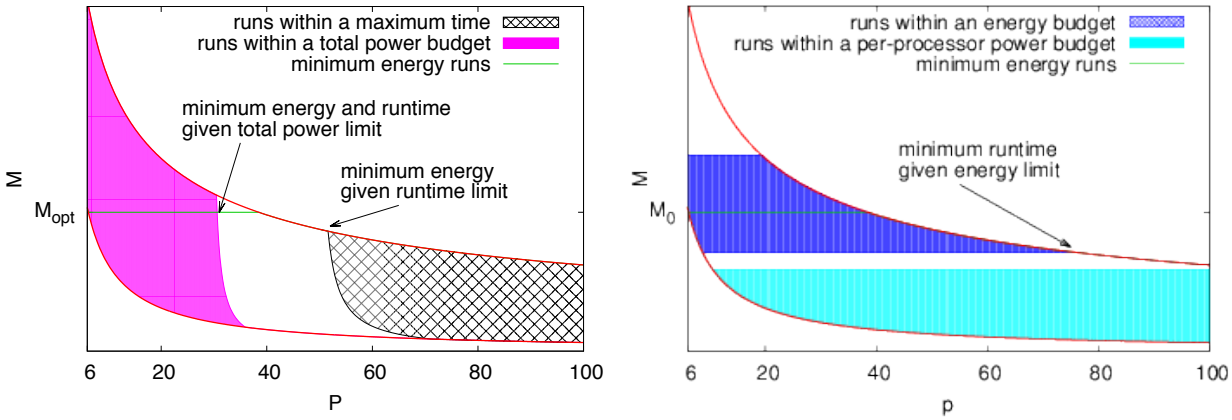
Figure 6.1: Energy costs as node count and memory are scaled

This parameter varies depending on the problem size and sometimes can be increased to reduce communication for specific algorithms. In the sequential model, considering the above problems with respect to the amount of utilized slow memory, $\hat{M}$, is not extremely interesting as the parameter only appears in one term. As none of the other runtime parameters (F,W and S) diminish with larger $\hat{M}$, the best approach to minimize energy consumption asymptotically would be to use the smallest amount of slow memory, which is the minimal size of the problem. As noted in Section 3.2, this formulation fits with the intuition of a larger slow memory consuming a larger amount of idle energy than a smaller cache. With this observation regarding sequential machines, we focus our analysis on the distributed parallel model, but the same questions may also be asked for heterogeneous machines (where the interesting parameters are the fast memory sizes, M_i).

As we are fixing hardware parameters in the models, there are three variables that may be considered: problem size n, node memory size M, and the number of processors P.[1] To illustrate the design space explored later in this chapter, we present Figures 6.1 and 6.2 to illustrate the general properties of the problems presented.

Each of the figures assumes a fixed problem size (n) and plots the number of nodes (P) vs. the amount of node memory (M) available for data replication. Node memory is constrained by two functions: the minimal amount of memory required to hold the problem ("minimum memory"), and the amount of memory at which the memory-independent communication bound begins to dominate ("maximum useful memory"). Within these two constraints and for a fixed M, we obtain perfect strong scaling in runtime with constant energy by scaling the number of nodes. This scaling can be seen in that energy is constant with P for a fixed M. Note that operating within the region above the "maximum useful memory" is possible, but without the property of perfect scaling. Note that we present M as a continuous axis, i.e., the figures assume that all additional memory can be used to reduce communication, even if there is not enough available for another full copy of the data.

[1] As the runtime parameters F,W and S are functions of these variables.



(a) Runtime and average power constraints

(b) Energy and per-processor power constraints

Figure 6.2: Effect of constraints on energy efficiency

If we focus upon Figure 6.1, we observe that both excessively small (where communication dominates energy costs) and excessively large (idle memory energy begins to dominate) amounts of node memory M increase the amount of energy required to solve the problem. Later, we will derive expressions for the optimal amount of utilized memory for the algorithms under consideration, and with the 2.5D $O(n^3)$ matrix-matrix multiplication and communication-avoiding (CA) $O(n^2)$ n-body algorithms, calculate the value analytically. In the figure, this energy-optimal amount of memory is M_{opt}. If we assume that each node utilizes M_{opt} words of memory and the number of processors is scaled, we can achieve perfect strong scaling in runtime while spending the minimal amount of energy (see green line). This optimal amount of energy is constant with processor scaling due to the usage of more memory to offset increased communication costs. Note that if the algorithm is only able to utilize cM_{min} words of memory, where M_{min} is the minimal amount of memory to hold the problem and c is a natural number, the most efficient possible run is to select the value of c closest to M_{opt}/M_{min}. We also represent lines of constant runtime in Figure 6.1 via black lines. Note that as communication time increases (smaller values of M), more processors are required to attain a desired runtime bound.

The plots within Figure 6.2 illustrate the impact of energy, runtime and power constraints when applied to the problem. Both plots, like Figure 6.1, consider P vs. M and display the constraints of minimum and maximum useful memory. The energy-optimal scaling region that occurs with M_{opt} words of utilized node memory is again displayed as a green line. In Figure 6.2a, the magenta region highlights runs that occur within a maximum total power budget (we refer to such a budget or limit as H_{DP1} in the following analysis). Note that the point of minimum energy and runtime within the power budget is the intersection of the energy-optimal scaling region (a run with M_{opt} words of memory) and the upper bound on the number of processors imposed by the budget. In addition, Figure 6.2a shows a crosshatched region of runs that could occur within a time bound (T_{max}). As energy consumption only depends on M between the minimum and maximum useful

memory constraints, the minimum energy point that attains T_{max} is found at the point of maximal possible local memory within the crosshatched region (see arrow in figure).

In Figure 6.2b, we illustrate a cyan region that represents runs under a per-processor power constraint (h_{DP1}). This region does not depend on the number of processors, just the amount of node memory, M. In addition to the cyan region, Figure 6.2b shows a blue region that indicates a region of runs below a maximum energy bound, or budget (E_{max}). As the amount of energy required for a run does not depend on the number of processors within the perfect strong scaling region, the only variation in energy is along the M axis, which has a minimum at M_{opt}. Hence, the region of runs constrained by E_{max} includes the green line of energy-optimal runs.

6.2 Example Machines for Analysis

To be more concrete, we will analyze two different distributed parallel machines: a one with modern nodes assuming Intel Xeon 2650 processors and a potential future machine that represents a significant improvement in hardware performance over current technology. We assume nodes each have 2 multicore processors. We also assume that per-message time and energy costs (α_e and α_t, respectively) are zero for simplicity when applying algebraic results to these example machines.[2] We assume a torus interconnect of dimension $d = 3$, which is sufficient such that link contention bounds do not dominate the per-processor bounds for the problems considered in this chapter (see Sections 4.1 and 5.1 for more details on contention bounds). The characteristics of the generic machines under discussion are shown in Table 6.1 and model parameters derived from these characteristics are shown in Table 6.2. We assume that the algorithms under analysis are each able to attain these parameters. We note that in actual implementations of the algorithms to be analyzed, the attained parameters may be significantly worse than the limits imposed by hardware (see Choi et al. [44]).

For the Xeon 2650 nodes, word size, cores per processor, core frequency, vectorization (SIMD), fused-multiply-add (FMA), and processor Thermal Design Power (TDP) are obtained from manufacturer sources [78, 82]. We assume use of dual-port 10Gb/s Intel X540-T2 ethernet network interface cards (NICs), with peak power of 13.4W also obtained from Intel sources [79]. We assume that the network card draws 85% of this peak power while idle, following evidence [106, 2] that NIC power consumption shows little variation with bandwidth utilization. As we assumed that the network is a 3-dimensional torus, each node has 3 NICs on-board (for six connections to neighboring machines in the torus). We also assume that node memory is composed of Micron 1600Mhz 16GB DDR3 DRAM DIMM modules and calculate memory power via the Virtium DIMM Memory Power Calculator [58]. For more details regarding values in Table 6.1, we refer the reader to Section 7.3. Parameters for the future machine are arbitrarily improved over the Xeon 2650-based machine for illustrative purposes. From the machine descriptions in Table 6.1, we use the equations in (6.1) to derive parameters for our runtime and energy models. Note that our method of calculating DRAM power assumes an upper bound (M_{hw}) on the amount of physical

[2]Note that latency terms in the model also have the benefit of being reduced by the largest message size, m. As such, communication volume will tend to dominate unless m is small or the per-message costs (α_t or α_e) are high.

Parameters	Xeon 2650	Future Machine
Word Size (bytes)	8	8
Processors	2	2
Cores	8	50
Vectorization (SIMD)	4	8
Fused-Multiply-Add (FMA)	2	2
Processor Freq. (Ghz)	2	2
Processor Total Power (W)	95	45
Processor Idle Fraction	0.15	0.15
NIC BW (Gb/s)	10	20
NIC Total Power (W)	13.4	13.4
NIC Idle Fraction	0.85	0.85
Torus Dim. (d)	3	3
Installed DRAM (M_{hw},GB)	128	128
Peak DRAM BW (GB/s)	102.4	102.4
DRAM Dynamic Power/GB	0.6729	0.3365
DRAM Idle Power/GB	0.2533	0.127
Node Base Power (W)	100	50

Table 6.1: Description of Xeon 2650-based and future distributed parallel machines

installed memory per node (128GB). DRAM dynamic energy is likely a mixture of both intranode and internode activity as caches and NICs operate[3]. According to the JEDEC standard [51], each DDR3-1600 DIMM module has a peak bandwidth of 12.8GB/s. If the system has 128GB of installed memory with 16GB DIMMs, we have 8 DIMMs per node. This results in 102.4GB/s of total peak DRAM bandwidth. On the other hand, each of the three 10Gb/s NICs has a peak bandwidth of 1.25GB/s for a total network bandwidth of 3.75GB/s. This is 3.7% of the peak DRAM bandwidth, and we assume that network traffic also consumes 3.7% of DRAM dynamic power. The derived parameters are shown in Table 6.2.

$$
\begin{aligned}
\gamma_t &= 1/(\text{Processor Freq} * \text{SIMD} * \text{Cores} * \text{Processors} * \text{FMA} * 1e9) \\
\gamma_e &= \gamma_t * \text{Processor Dynamic Power} * \text{Processors} \\
\beta_t &= 1/(\text{Peak Network BW}/\text{Words Per GB}) \\
\beta_e &= \beta_t * (\text{DRAM Dynamic Power} + \text{NIC Dynamic Power}) \\
\delta_e &= \text{DRAM Idle Power Per GB}/\text{Words Per GB} \\
\epsilon_e &= \text{Processors} * \text{Processor Idle Power} + \text{Node Base Power} + \text{NIC Idle Power}
\end{aligned} \tag{6.1}
$$

[3]Unless the NIC is integrated on the processor, in which case internode communication can bypass main memory. In this work, we assume NIC integration does not occur.

where

$$\begin{aligned}
\text{Processor Idle Power} &= \text{Processor Total Power} * \text{Processor Idle Fraction} \\
\text{Processor Dynamic Power} &= \text{Processor Total Power} * (1 - \text{Processor Idle Fraction}) \\
\text{DRAM Dynamic Power} &= (M_{hw} * \text{DRAM Dynamic Power Per GB}) \\
&\quad * (\text{Peak Network BW}/\text{Peak DRAM BW}) \\
\text{NIC Idle Power} &= \text{NIC Total Power} * \text{NIC Idle Fraction} * d \\
\text{NIC Dynamic Power} &= \text{NIC Total Power} * (1 - \text{NIC Idle Fraction}) * d \\
\text{Peak Network BW} &= (\text{NIC BW} * d)/\text{Bits Per Byte} \\
\text{Words Per GB} &= 1e9/\text{Word Size}.
\end{aligned}$$

	Xeon 2650	Future Machine
Processor Idle Power (W)	14.25	6.75
Processor Dynamic Power (W)	80.75	38.25
DRAM Dynamic Power (W)	3.15	6.31
NIC Idle Power (W)	34.17	34.17
NIC Dynamic Power (W)	6.03	6.03
Peak Network BW (GB/s)	3.75	15.00
γ_t (sec/flop)	3.91E-12	3.13E-13
γ_e (joule/flop)	6.31E-10	2.39E-11
β_t (sec/word)	6.40E-09	1.60E-09
β_e (joule/word)	5.88E-08	1.97E-08
δ_e (joules/sec)/word	2.03E-09	1.02E-09
ϵ_e (joules/sec)	162.67	97.67

Table 6.2: Parameters derived from machine descriptions

With these example machines, and the high-level overview of impact of constraints on runtime and energy from Section 6.1, we now analyze a subset of the algorithms available to perform dense $O(n^3)$ matrix-matrix factorization, the $O(n^2)$ n-body problem, and an example of a program lower bounded by the HBL results discussed within Chapter 5. This final example will generalize the analysis of the first two problems, and demonstrate the applicability of our analysis to a much larger class of programs. Each of the algorithms is able to use additional memory to reduce communication, and perfectly strong scale in runtime with constant energy for a range of processors (see Section 3.3 for a discussion of these algorithms, and Sections 4.1 and 5.2 for a discussion of perfect scaling).

In the discussion that follows, we will algebraically explore the impact of constraints upon energy consumption and runtime and present heat plots with the color scale representing energy efficiency (i.e. performance per watt, or Gflop/s/W). In the DP1 model for the class of problems

that access arrays via subsets of the iteration variables[4], the energy efficiency, $C_{perf}^{HBL} = F/E_{DP1}^{HBL}$, is

$$C_{perf}^{HBL}(M,P,F) = \frac{F}{E_{DP1}^{HBL}(M,P,F)} = \frac{F}{P\left(\gamma_e \frac{F}{P} + \beta_e W_{DP_e}^{HBL} + (\delta_e M + \epsilon_e)\left(\gamma_t \frac{F}{P} + \beta_t W_{DP_t}^{HBL}\right)\right)}.$$

In this section, we assume that network contention does not dominate, so the communication bound for the runtime model is equivalent to the energy bound (i.e. $W_{DP_e}^{HBL} = W_{DP_t}^{HBL}$, see Equations (5.6) and (5.8)), and we substitute Equation (5.6) into the equation for C_{perf}^{HBL} to obtain

$$C_{perf}^{HBL}(M,P,F) = \Omega\left(F\left[P\left(\gamma_e \frac{F}{P} + \beta_e \max\left(\frac{F}{PM^{s_{\text{HBL}}-1}}, \left(\frac{F}{P}\right)^{1/s_{\text{HBL}}}\right)\right.\right.\right.$$
$$\left.\left.\left.+(\delta_e M + \epsilon_e)\left(\gamma_t \frac{F}{P} + \beta_t \max\left(\frac{F}{PM^{s_{\text{HBL}}-1}}, \left(\frac{F}{P}\right)^{1/s_{\text{HBL}}}\right)\right)\right)\right]^{-1}\right)$$

which simplifies to

$$C_{perf}^{HBL}(M,P,F) = \Omega\left(\left[\gamma_e + \beta_e \max\left(M^{1-s_{\text{HBL}}}, \left(\frac{P}{F}\right)^{1-1/s_{\text{HBL}}}\right)\right.\right.$$
$$\left.\left.+(\delta_e M + \epsilon_e)\left(\gamma_t + \beta_t \max\left(M^{1-s_{\text{HBL}}}, \left(\frac{P}{F}\right)^{1-1/s_{\text{HBL}}}\right)\right)\right]^{-1}\right).$$

For ease of analysis, we will henceforth assume that the memory-dependent per-processor bound dominates, and the expression becomes

$$C_{perf\_MDP}^{HBL}(M) = \Omega\left(\left[\gamma_e + \beta_e M^{1-s_{\text{HBL}}} + \delta_e M \gamma_t + \delta_e \beta_t M^{2-s_{\text{HBL}}}\right.\right.$$
$$\left.\left.+\epsilon_e \gamma_t + \epsilon_e \beta_t M^{1-s_{\text{HBL}}}\right]^{-1}\right). \tag{6.2}$$

where s_{HBL} is a problem-specific constant (see Chapter 5 for details) and "MDP" stands for the "Memory Dependent per-Processor" bound.

To visualize energy efficiency, we will hold the number of processors, P, constant and scale the problem size, n, with the amount of data replication (c). In this manner, we regard node memory as a resource to be used by an algorithm within the upper bound on replication imposed by the memory-independent per-processor communication bound (and the amount of DRAM installed on the node, M_{hw}).

6.3 Classical $O(n^3)$ Matrix-matrix Multiplication

The lower bounds on runtime (Equation (4.29)) and energy (Equation (4.30)) consumption on a distributed parallel machine model (DP1) for a load-balanced $O(n^3)$ matrix-matrix multiplication algorithm that starts with one copy of the data are

[4]Some other technical assumptions apply, see Chapter 5 and [45]

$$T^{CMM}_{DP1\_DMP}(M,P,n) = \Omega\left(\frac{\gamma_t n^3}{P} + \frac{\beta_t n^3}{PM^{1/2}} + \frac{\alpha_t n^3}{mPM^{1/2}}\right) \tag{6.3}$$

and

$$\begin{aligned} E^{CMM}_{DP1\_MDP}(M,n) &= \Omega\left((\gamma_e + \gamma_t\epsilon_e)n^3 + \left((\beta_e + \beta_t\epsilon_e) + \frac{(\alpha_e + \alpha_t\epsilon_e)}{m}\right)\frac{n^3}{M^{\frac{1}{2}}}\right. \\ &\quad \left. + \delta_e\gamma_t M n^3 + \left(\delta_e\beta_t + \frac{\delta_e\alpha_t}{m}\right)M^{\frac{1}{2}}n^3\right) \end{aligned} \tag{6.4}$$

if we assume that the memory-dependent per-processor (MDP) bound is dominant. To review Section 3.3, this bound is attained via the 2.5D matrix multiplication algorithm [126], which is able to use additional memory to replicate input data and reduce total communication volume. In the following discussion, we will use this algorithm and model potential energy and runtime performance in the presence of various constraints. We choose the variable c to represent the memory replication factor. To reiterate Section 3.3, at $c = 1$, the algorithm executes the SUMMA matrix-matrix multiplication algorithm [67]. At the largest value of c for this algorithm, $c = P^{1/3}$, the 2.5D algorithm executes the 3D matrix-matrix multiplication algorithm [52, 3, 4, 87]. As we mentioned in Chapter 4, this ability to scale data replication allows for perfect strong scaling in runtime with constant energy for a range of processors (and consequently, a range of available memory to be used for data replication).

Minimizing energy for the computation. Note that the energy usage in Equation (6.4) is independent of P, and assuming the parameters of Table 6.2 are fixed, we need to choose the value of M that optimizes energy use. Setting

$$A = (\beta_e + \beta_t\epsilon_e) + \frac{(\alpha_e + \alpha_t\epsilon_e)}{m}, \qquad B = \delta_e\gamma_t, \qquad C = \delta_e\beta_t + \frac{\delta_e\alpha_t}{m},$$

we see that Equation (6.4) becomes

$$E^{CMM}_{DP1\_MDP}(M,n) = \Omega\left(n^3((\gamma_e + \epsilon_e\gamma_t) + M^{-1/2}A + MB + M^{1/2}C)\right).$$

If we set $x = M^{1/2}$ by the chain rule $dE/dx = (dE/dM)/(dx/dM)$, the derivative of $E^{CMM}_{DP1\_MDP}$ with respect to x is

$$\frac{dE^{CMM}_{DP1\_MDP}}{dx} = n^3\left(-\frac{A}{x^2} + 2Bx + C\right) \tag{6.5}$$

which, according to Maple 15.01 [107] has a unique positive real root at

$$M_{opt} = \left(\frac{1}{6}\left(\frac{D}{B} + \frac{C^2}{BD} - \frac{C}{B}\right)\right)^2 \tag{6.6}$$

where

$$D = \left(6\sqrt{3}(27A^2B^4 - AB^2C^3)^{\frac{1}{2}} + 54AB^2 - C^3\right)^{\frac{1}{3}}.$$

Equation (6.6) is a minimum if $M_{opt}^{1/2}$ is positive, as would be expected for realistic values of M.[5] As in Chapter 4, energy consumption is constant in the range $n^2/P \leq M \leq n^2/P^{2/3}$, assuming that M is held fixed. This means it is possible to attain the minimum energy use for any

$$\frac{n^2}{M_{opt}} \leq P \leq \frac{n^3}{M_{opt}^{3/2}}.$$

Minimizing energy given an upper bound on the runtime. As in the situation of minimizing energy without a constraint, we can solve this problem in a closed form despite imposing an upper bound on runtime, T_{max}. Two potential situations may arise:

1. It is possible to achieve the required runtime T_{max} and use the minimum amount of energy. This is the case if $T_{DP1\_MDP}^{CMM} \leq T_{max}$ when $M = M_{opt}$. In this case, for example, taking $P = n^3/M_{opt}^{3/2}$ and $M = M_{opt}$ is energy-optimal.
2. Otherwise, we need to use the maximal usable amount of memory to achieve T_{max} (i.e. run as the 3D multiplication algorithm, see Section 3.3). In this case, we must use at least $P_{T_{max}}$ processors, where the time to run the algorithm equals T_{max}. So, we must solve

$$T_{max} = T_{DP1\_MDP}^{CMM}(M, P_{T_{max}}, n) = \frac{n^3\left(\gamma_t m M^{1/2} + \beta_t m + \alpha_t\right)}{m P_{T_{max}} M^{1/2}}$$

for $P_{T_{max}}$ when $M = n^2/P_{T_{max}}^{2/3}$ (as we recall that $M = cn^2/P$ where $c = P_{T_{max}}^{1/3}$ to run 2.5D matrix-matrix multiplication as a 3D algorithm). Finally, one chooses the value of P that minimizes $E_{DP1\_MDP}^{CMM}$ when running the 2.5D algorithm with $c = P^{1/3}$, subject to the constraint that $P \geq P_{T_{max}}$. An example of an upper bound on runtime can be seen in Figure 6.2a.

Minimizing runtime given an upper bound on energy. Conversely, if we fix the maximum allowed energy to some bound E_{max}, this will limit M to lie in some range, as we saw illustrated by the blue region in Figure 6.2b. Minimizing $T_{DP1\_MDP}^{CMM}$ will always result in a 3D run (i.e. a run that uses the largest amount of replication, until limited by the memory-independent per-processor bound), as runtime decreases in P and energy does not increase with the number of processors in this range. Thus, we wish to calculate the maximal P that is able to attain E_{max}, i.e. solve

[5] The derivative of Equation (6.5) is $2A/x^3 + 2B$, which has sign determined by the sign of $x = M^{1/2}$ as B and A are positive constants. If this second derivative of $E_{DP1\_MDP}^{CMM}$ is positive, Equation (6.6) is a minimum energy value of M.

$$E_{max} = E^{CMM}_{DP1\_MDP}(n^2/P^{2/3}, n)$$

for P, and consider the largest real root. This equation becomes

$$\begin{aligned} 0 = {} & n^2(\beta_e m + \alpha_e + \epsilon_e\alpha_t)P + m(\gamma_e n^3 + \epsilon_e\gamma_t n^3 - E_{max})P^{2/3} \\ & + n^4\delta_e(\beta_t m + \alpha_e)P^{1/3} + n^2(\delta_e\gamma_t n^3 m + \epsilon_e\beta_t) \end{aligned}$$

which notably can not admit a positive, real root unless $E_{max} > (\gamma_e + \epsilon_e\gamma_t)n^3$. Note that this inequality is necessary but not sufficient to guarantee that a real solution exists.

Minimizing the per-processor average power. By considering that average power for a processor is $h^{CMM}_{DP1\_MDP} = E^{CMM}_{DP1\_MDP}/(PT^{CMM}_{DP1\_MDP})$ and assuming that all processors complete the task utilizing the same amount of time and energy, we can combine our previous expressions for $E^{CMM}_{DP1\_MDP}$ and $T^{CMM}_{DP1\_MDP}$ to obtain an expression for per-processor power $h^{CMM}_{DP1\_MDP}$:

$$\begin{aligned} h^{CMM}_{DP1\_MDP}(M, P, n) &= \frac{E^{CMM}_{DP1\_MDP}(M, n)}{PT^{CMM}_{DP1\_MDP}(M, P, n)} \\ &= \Omega\left(\frac{\gamma_e n^3 + \frac{\beta_e n^3}{M^{1/2}} + \frac{\alpha_e n^3}{mM^{1/2}} + (\delta_e M + \epsilon_e)\left(\gamma_t n^3 + \frac{\beta_t n^3}{M^{1/2}} + \frac{\alpha_t n^3}{mM^{1/2}}\right)}{PT^{CMM}_{DP1\_MDP}(M, P, n)}\right) \\ &= \Omega\left(\frac{\left(\gamma_e n^3 + \frac{\beta_e n^3}{M^{1/2}} + \frac{\alpha_e n^3}{mM^{1/2}}\right)}{\left(\gamma_t n^3 + \frac{\beta_t n^3}{M^{1/2}} + \frac{\alpha_t n^3}{mM^{1/2}}\right)} + \delta_e M + \epsilon_e\right) \end{aligned}$$

thus,

$$h^{CMM}_{DP1\_MDP}(M) = \Omega\left(\frac{\gamma_e M^{1/2} m + \beta_e m + \alpha_e}{\gamma_t M^{1/2} m + \beta_t m + \alpha_t} + \delta_e M + \epsilon_e\right). \tag{6.7}$$

Note that this definition of per-processor average power assumes that network power also scales with the number of processors, which may not be accurate for certain type of networks. This problem could be resolved by defining the per-processor average power in terms of the per-node energy from the DP2 model (which does not include internode energy costs). If we assume a constant problem size n and substitute $M = cn^2/P$ and $x = c/P$ into Equation (6.7), we obtain

$$\begin{aligned} h^{CMM}_{DP1\_MDP}(xn^2) &= \Omega\left(\frac{\gamma_e nmx^{1/2} + \beta_e m + \alpha_e}{\gamma_t nmx^{1/2} + \beta_t m + \alpha_t} + \delta_e n^2 x + \epsilon_e\right) \\ &= \Omega\left(\frac{Ax^{1/2} + B}{Cx^{1/2} + D} + Jx + \epsilon_e\right) \end{aligned}$$

where $A = \gamma_e nm$, $B = \beta_e m + \alpha_e$, $C = \gamma_t nm$, $D = \beta_t m + \alpha_t$ and $J = \delta_e n^2$. As the goal is to minimize average per-processor power, we solve $dh^{CMM}_{DP1\_MDP}/dx = 0$ so that we may obtain potential critical points:

$$\frac{dh^{CMM}_{DP1\_MDP}(xn^2)}{dx} = \frac{A(Cx^{1/2} + D) - C(Ax^{1/2} + B)}{2x^{1/2}(Cx^{1/2} + D)^2} + J = 0.$$

Via some rearrangement of terms, this becomes

$$2C^2Jx^{3/2} + 4CDJx + 2D^2Jx^{1/2} + (AD - BC) = 0.$$

If we substitute $y = x^{1/2}$, we obtain a cubic polynomial

$$2C^2Jy^3 + 4CDJy^2 + 2D^2Jy + (AD - BC) = 0. \tag{6.8}$$

Considering Equation (6.8), there are two cases wherein we minimize power.

1. If $BC > AD$, i.e.
$$\frac{\beta_e m + \alpha_e}{\gamma_e} > \frac{\beta_t m + \alpha_t}{\gamma_t},$$
then there is a unique positive value of $x = c/P$ that minimizes power, which is given by the square of the positive root of Equation (6.8). This root is a minima as the second derivative of $h^{CMM}_{DP1\_MDP}$ is clearly a positive monotonically-increasing function. Note that this inequality can be interpreted as: the number of flops that can be performed using the same *energy* as sending a maximum-length message is greater than the number of flops that can be performed in the same *time* as sending a maximum-length message.
2. Otherwise, power is an increasing function of c/P for all positive values, and the minimum average power is attained by setting $c = 1$, and P as large as possible, and $h^{CMM}_{DP1\_MDP}$ approaches
$$\Omega\left(\frac{\beta_e m + \alpha_e}{\beta_t m + \alpha_t} + \epsilon_e\right) \tag{6.9}$$
which is the leakage ϵ_e plus the power to send one message of the largest possible size (m).

Minimizing runtime or energy given a bound on average power By considering that the total average power consumed is $H^{CMM}_{DP1\_MDP} = E^{CMM}_{DP1\_MDP}/T^{CMM}_{DP1\_MDP} = Ph^{CMM}_{DP1\_MDP}$, we can use Equations (6.4) and (6.3) for $E^{CMM}_{DP1\_MDP}$ and $T^{CMM}_{DP1\_MDP}$ to obtain an expression for total average power $H^{CMM}_{DP1\_MDP}$:

$$H^{CMM}_{DP1\_MDP}(M, P) = \Omega\left(P\left(\frac{\gamma_e M^{\frac{1}{2}} m + \beta_e m + \alpha_e}{\gamma_t M^{\frac{1}{2}} m + \beta_t m + \alpha_t} + \delta_e M + \epsilon_e\right)\right) \tag{6.10}$$

An upper bound on total power (H_{max}) thus translates into an upper bound on the number of processors:

$$P \leq P_{H_{max}} = \Omega\left(H_{max}\left(\frac{\gamma_e M^{\frac{1}{2}} m + \beta_e m + \alpha_e}{\gamma_t M^{\frac{1}{2}} m + \beta_t m + \alpha_t} + \delta_e M + \epsilon_e\right)^{-1}\right) \tag{6.11}$$

Using the maximum number of processors, the running time simply becomes

$$T^{CMM}_{DP1\_MDP}(M, P_{H_{max}}, n) = \frac{E^{CMM}_{DP1\_MDP}(M, n)}{H_{max}},$$

and the problem of minimizing time or energy given total power is reduced to minimizing energy, which we have already solved, with the additional constraint (6.11) between P and M.

Alternately we may want to minimize the runtime given a bound on the power per processor (h_{max}). The bound is

$$h_{max} = \Omega\left(\frac{\gamma_e M^{\frac{1}{2}} m + \beta_e m + \alpha_e}{\gamma_t M^{\frac{1}{2}} m + \beta_t m + \alpha_t} + \delta_e M + \epsilon_e\right),$$

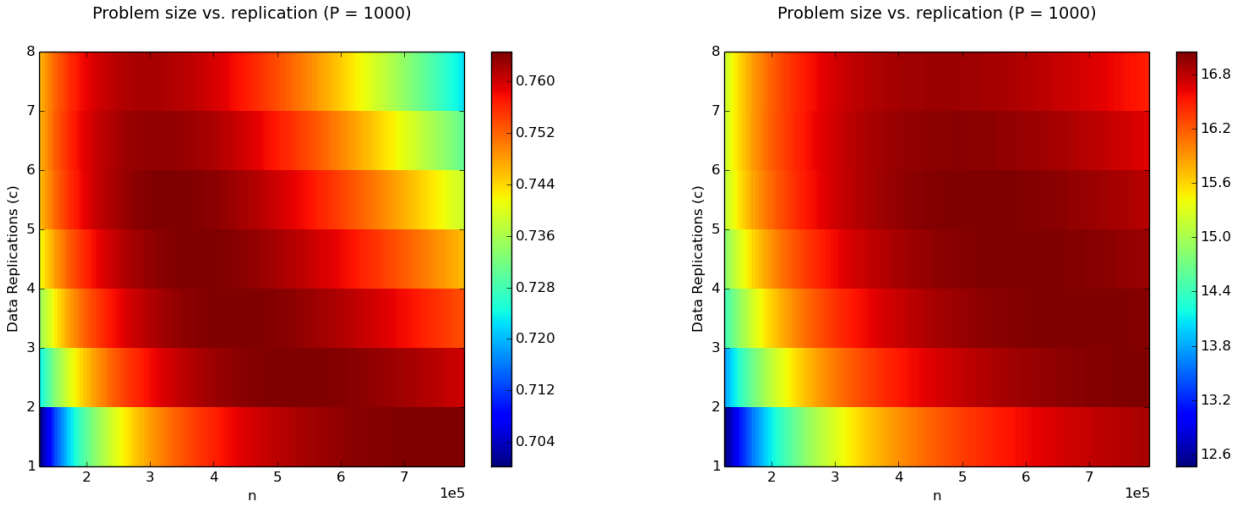
which we may solve for analytically for M since this is really a cubic equation in $M^{1/2}$, just like Equation (6.5). To minimize $T^{CMM}_{DP1\_MDP}$, we would like as many processors and as much memory as possible subject to the constraints on M formed by solving(6.11) and $M \leq n^2/P^{2/3}$. This implies running using the 3D algorithm.

If we wish to minimize energy with the constraint imposed by h_{max} and the energy-optimal amount of memory (M_{opt}, Equation (6.6)) is in the range allowed by h_{max}, then the global minimum energy can be attained within a per-processor power budget h_{max}. If not, since $E^{CMM}_{DP1\_MDP}$ is a decreasing function of M for $M < M_{opt}$, the minimum energy is when M takes its maximum value allowed by the above constraint on M and P is anywhere in the range $n^2/M < P < n^3/M^{3/2}$.

We now consider the effect of scaling problem size and number of data replications on the metric of energy efficiency (C_{perf_MDP}). For 2.5D $O(n^3)$ matrix-matrix multiplication, $s_{\text{HBL}} = 3/2$, and Equation (6.2) becomes

$$C^{CMM}_{perf\_MDP}(M) = \Omega\left(\left[\gamma_e + \beta_e M^{-1/2} + \delta_e M\gamma_t + \delta_e\beta_t M^{1/2} + \epsilon_e\gamma_t + \epsilon_e\beta_t M^{-1/2}\right]^{-1}\right) \tag{6.12}$$

where we recall that $M = cn^2/P$ where c is the number of replications of the input data. In Figure 6.3, we see two plots that represent the energy efficiency of 2.5D matrix-matrix multiplication on the Xeon 2650-based (Figure 6.3a) and future distributed (Figure 6.3b) machines (according to model parameters detailed in Table 6.2). We assume the number of processors to be fixed at



(a) Xeon 2650-based distributed machine

(b) Future distributed machine

Figure 6.3: 2.5D $O(n^3)$ Matrix-matrix Multiplication: Effect of replicating memory on energy efficiency

$P = 1000$, and scale the problem size from $n = 125000$ to $n = 800000$.[6] Up to 8 replications of the input data (the y-axis) are considered, and the machine in this range of memory utilization does not attain more than approximately 0.77 Gflops/W. Note that for low memory utilization (bottom left corner of the plots), efficiency is low due to the inability of memory utilization to offset communication costs (terms that include β_e or β_t). For a given problem size, replicating the input data increases efficiency until the idle memory costs (terms that include δ_e) begins to decrease efficiency (the upper-right corner of the figures). For matrix-matrix multiplication, idle memory energy increases rapidly due the quadratic dependence of M on the problem size n. Via improved hardware parameters, the future machine is able to achieve higher efficiency ($C^{CMM}_{perf\_MDP} \approx 17$) for a wider range of problem sizes before idle memory costs begin to dominate.

6.4 $O(n^2)$ n-body problem

In the case of the $O(n^2)$ n-body problem, many questions can be answered with relatively simple expressions. To explore the effect of using data replication to reduce communication volume, we assume use of the communication-optimal n-body algorithm discussed in Chapter 3 and first described in Driscoll et al. [61]. In Chapter 4, we derived a lower bound on energy for this algorithm (Equation (4.45)). Assuming that the network topology is sufficient for contention not to dominate, communication is bounded by the maximum of the memory-dependent and memory-independent

[6] A range of sizes chosen to highlight scaling trends while keeping total utilized memory less than the amount of installed node memory (128GB).

per-processor bounds. When the memory-dependent bound dominates, the runtime and energy lower bounds of Equation (4.44) and (4.45) become

$$T^{NB}_{DP1\_MDP}(M,P,n) = \Omega\left(\frac{\gamma_t f n^2}{P} + \frac{\beta_t n^2}{PM} + \frac{\alpha_t n^2}{mPM}\right) \tag{6.13}$$

and

$$E^{NB}_{DP1\_MDP}(M,n) = \Omega\left(\left(\gamma_e f n^2 + \beta_e \frac{n^2}{M} + \alpha_e \frac{n^2}{mM} + (\delta_e M + \epsilon_e)\left(\gamma_t f n^2 + \beta_t \frac{n^2}{M} + \alpha_t \frac{n^2}{mM}\right)\right)\right) \tag{6.14}$$

where "MDP" stands for the "Memory Dependent per-Processor" bound and for $n/M \leq P \leq (n/M)^2$ the algorithm is able to perfectly strong scale in runtime with constant energy until the memory-independent bound dominates.

Minimizing energy for the computation. The minimum energy for the $O(n^2)$ n-body problem is

$$E^{NB*}_{DP1\_MDP} = E^{NB}_{DP1\_MDP}(M_{opt}, n)$$

$$= \Omega\left(n^2\left(f(\gamma_e + \gamma_t\epsilon_e) + \delta_e\left(\beta_t + \frac{\alpha_t}{m}\right) + 2\left(\delta_e\gamma_t f\left(\beta_e + \beta_t\epsilon_e + \frac{\alpha_e + \alpha_t\epsilon_e}{m}\right)\right)^{\frac{1}{2}}\right)\right)$$

when using the energy-optimal amount of memory

$$M_{opt} = \left(\frac{\beta_e + \beta_t\epsilon_e + (\alpha_e + \alpha_t\epsilon_e)/m}{\delta_e\gamma_t f}\right)^{\frac{1}{2}}.$$

It is possible to attain the minimum energy use for P in the range

$$\frac{n}{M_{opt}} \leq P \leq \frac{n^2}{M^2_{opt}}.$$

Minimizing energy given an upper bound on the runtime. There are two cases

1. If
$$T_{max} \geq \gamma_t f M_0^2 + (\beta_t + (\alpha_t/m))M_{opt}$$
then it is possible to achieve the absolute minimum energy $E^{NB*}_{DP1\_MDP}$ within time T_{max}, for example by setting $M = M_{opt}$, $P = n^2/M^2_{opt}$.
2. Otherwise, it is necessary to use the 2D algorithm (i.e. with $M = n/P^{1/2}$) to achieve the T_{max}. To be precise, it is necessary to use at least
$$P_{T_{max}} = \left(\frac{(\beta_t + \alpha_t/m)n + ((\beta_t + \alpha_t/m)^2 n^2 + 4T_{max}\gamma_t f n^2)^{\frac{1}{2}}}{2T_{max}}\right)^2$$
processors, which we obtain by solving the quadratic equation that results from solving $T^{NB}_{DP1\_MDP}$ for P.

Minimizing runtime given an upper bound on energy. Conversely, suppose we fix the maximum allowed energy E_{max} and want to minimize the running time. Minimizing $T^{NB}_{DP1\_MDP}$ will always use the 2D algorithm, since increasing P during use of the 1.5D algorithm until it hits the 2D boundary decreases $T^{NB}_{DP1\_MDP}$. Further, the 2D runtime is a decreasing function of P, so we only need to determine the maximum P such that the 2D algorithm fits in the energy bound. This value of P, $P_{E_{max}}$, is maximum of the two solutions[7]

$$P_{E_{max}} = \left(\frac{E_{max} - An^2}{2Bn} \pm \frac{((-E_{max} + An^2) - 4Bn^4\delta_e\gamma_t f)^{1/2}}{2Bn} \right)^2$$

where

$$A = f(\gamma_e + \gamma_t\epsilon_e) + \delta_e(\beta_t + \alpha_t/m)$$
$$B = \beta_e + \alpha_e/m + \epsilon_e\beta_t + \epsilon_e\alpha_t/m$$

Note that this expression has an imaginary component if the energy bound $E_{\max}$ is not attainable.

Minimizing the per-processor average power. By considering that average power for a processor is $h^{NB}_{DP1\_MDP} = E^{NB}_{DP1\_MDP}/(PT^{NB}_{DP1\_MDP})$ and assuming that all processors complete the task utilizing the same amount of time and energy, we can combine our Equations (6.14) and (6.13) for $E^{NB}_{DP1\_MDP}$ and $T^{NB}_{DP1\_MDP}$ to obtain an expression for power $h^{NB}_{DP1\_MDP}$:

$$\begin{aligned} h^{NB}_{DP1\_MDP}(M,P,n) &= \frac{E^{NB}_{DP1\_MDP}(M,n)}{PT^{NB}_{DP1\_MDP}(M,P,n)} \\ &= \Omega\left(\frac{\gamma_e f n^2 + \frac{\beta_e n^2}{M} + \frac{\alpha_e n^2}{mM} + (\delta_e M + \epsilon_e)\left(\gamma_t f n^2 + \frac{\beta_t n^2}{M} + \frac{\alpha_t n^2}{mM}\right)}{PT^{NB}_{DP1\_MDP}(M,P,n)} \right) \\ &= \Omega\left(\frac{\left(\gamma_e f n^2 + \frac{\beta_e n^2}{M} + \frac{\alpha_e n^2}{mM}\right)}{\left(\gamma_t f n^2 + \frac{\beta_t n^2}{M} + \frac{\alpha_t n^2}{mM}\right)} + \delta_e M + \epsilon_e \right) \end{aligned}$$

thus,

$$h^{NB}_{DP1\_MDP}(M) = \Omega\left(\frac{\gamma_e f M m + \beta_e m + \alpha_e}{\gamma_t f M m + \beta_t m + \alpha_t} + \delta_e M + \epsilon_e \right) \tag{6.15}$$

As similar to the case of matrix-matrix multiplication, we substitute $M = cn/P$ into Equation (6.15) and assume that $x = c/P$ to simplify the analysis. We obtain:

$$h^{NB}_{DP1\_MDP}(xn) = \Omega\left(\frac{Ax + B}{Cx + D} + Jx + \epsilon_e \right)$$

[7] In some cases there are zero real solutions when the second term is imaginary.

where $A = \gamma_e fmn$, $B = \beta_e m + \alpha_e$, $C = \gamma_t fmn$, $D = \beta_t m + \alpha_t$ and $J = \delta_e n$. To minimize $h^{NB}_{DP1\_MDP}$, we must consider the critical points of the previous expression by solving $dh^{NB}_{DP1\_MDP}/dx = 0$. Taking the derivative of $h^{NB}_{DP1\_MDP}$ w.r.t. to x, the equation becomes

$$\frac{dh^{NB}_{DP1}(xn)}{dx} = \frac{A(Cx+D) - C(Ax+B)}{(Cx+D)^2} + J = 0.$$

And after rearranging terms,

$$JC^2x^2 + 2CDJx + (JD^2 + AD - BC) = 0. \tag{6.16}$$

Considering Equation (6.16), there are two cases wherein we minimize power.

1. If $BC > AD + JD^2$, i.e.

$$\frac{\beta_e m + \alpha_e}{\gamma_e} > \frac{\beta_t m + \alpha_t}{\gamma_t} + \frac{\delta_e(\beta_t m + \alpha_t)^2}{\gamma_e \gamma_t fm} \tag{6.17}$$

   then there is a unique positive value of $x = c/P$ that minimizes power, which is given by the positive root of Equation (6.16). We obtain such a real root as the discriminant of the quadratic in Equation (6.16) is positive if

$$\frac{\beta_e m + \alpha_e}{\gamma_e} \geq \frac{\beta_t m + \alpha_t}{\gamma_t}$$

   which is attained if the inequality in Equation (6.17) is satisfied. This root is a minima as the second derivative of $h^{NB}_{DP1\_MDP}$ is clearly a positive monotonically-increasing function. Note that this inequality can be interpreted as: the number of flops that can be performed using the same *energy* as sending a maximum-length message is greater than the number of flops that can be performed in the same *time* as sending a maximum-length message.

2. Otherwise, power is an increasing function of c/P for all positive values, and the minimum average power is attained by setting $c = 1$, and P as large as possible, and $h^{NB}_{DP1\_MDP}$ approaches

$$\Omega\left(\frac{\beta_e m + \alpha_e}{\beta_t m + \alpha_t} + \epsilon_e\right)$$

   as in the case of 2.5D matrix-matrix multiplication.

Minimizing runtime or energy given a bound on average power By considering that the total average power consumed is $H^{NB}_{DP1\_MDP} = E^{NB}_{DP1\_MDP}/T^{NB}_{DP1\_MDP} = Ph^{NB}_{DP1\_MDP}$, we can use Equations (6.14) and (6.13) for $E^{NB}_{DP1\_MDP}$ and $T^{NB}_{DP1\_MDP}$ to obtain an expression for total average power $H^{NB}_{DP1\_MDP}$:

$$H^{NB}_{DP1\_MDP}(M,P) = \Omega\left(P\left(\frac{\gamma_e f + \beta_e/M + \alpha_e/(mM)}{\gamma_t f + \beta_t/M + \alpha_t/(mM)} + \delta_e M + \epsilon_e\right)\right) \tag{6.18}$$

An upper bound on total power thus translates into an upper bound on the number of processors

$$P \leq P_{H_{max}} = \Omega\left(H_{max}\left(\frac{\gamma_e f + \beta_e/M + \alpha_e/(mM)}{\gamma_t f + \beta_t/M + \alpha_t/(mM)} + \delta_e M + \epsilon_e\right)^{-1}\right) \tag{6.19}$$

where H_{max} is the total average power bound. Using the maximum number of processors, the running time simply becomes

$$T^{NB}_{DP1\_MDP}(M, P_{H_{max}}, n) = \frac{E^{NB}_{DP1\_MDP}(M, n)}{H_{max}},$$

and the problem of minimizing time or energy given total power is reduced to minimizing energy, which we have already solved, with the additional constraint (6.19) between P and M.

Alternately we may want to minimize the runtime given a bound on the power per processor. The bound on per-processor average power (h_{max}) is

$$h_{max} = \Omega\left(\frac{\gamma_e f + \beta_e/M + \alpha_e/(mM)}{\gamma_t f + \beta_t/M + \alpha_t/(mM)} + \delta_e M + \epsilon_e\right),$$

which we may solve for M

$$M = O\left(\frac{A + (A^2 - 4\gamma_e\gamma_t f B)^{1/2}}{2\delta_e\gamma_t f}\right)$$

where

$$A = \gamma_t f h_{max} - \gamma_e f - \epsilon_e\gamma_t f - \delta_e(\beta_t + \alpha_t/m)$$

and

$$B = \beta_e + \alpha_e/m - (\beta_t + \alpha_t/m)h_{max} - \epsilon_e(\beta_t + \alpha_t/m).$$

This is the region of M for which h_{max} is attained. To minimize $T^{NB}_{DP1\_MDP}$, we would use as many processors as possible, and as much memory as possible subject to the above inequality and $M \leq n/P^{1/2}$.

If M_{opt} is in the range allowed by $h_{\max}$, then the global minimum energy can be attained within a per-processor power budget $h_{\max}$. If not, since $E^{NB}_{DP1\_MDP}$ is a decreasing function of M for $M < M_{opt}$, the minimum energy is when M takes its maximum value allowed by the above inequality and P is anywhere in the range $\frac{n}{M} < P < \frac{n^2}{M^2}$.

We now consider the effect of scaling problem size and number of data replications on the metric of energy efficiency (C_{perf_MDP}). For the CA $O(n^2)$ n-body algorithm, $s_{\mathrm{HBL}} = 2$, and Equation (6.2) becomes

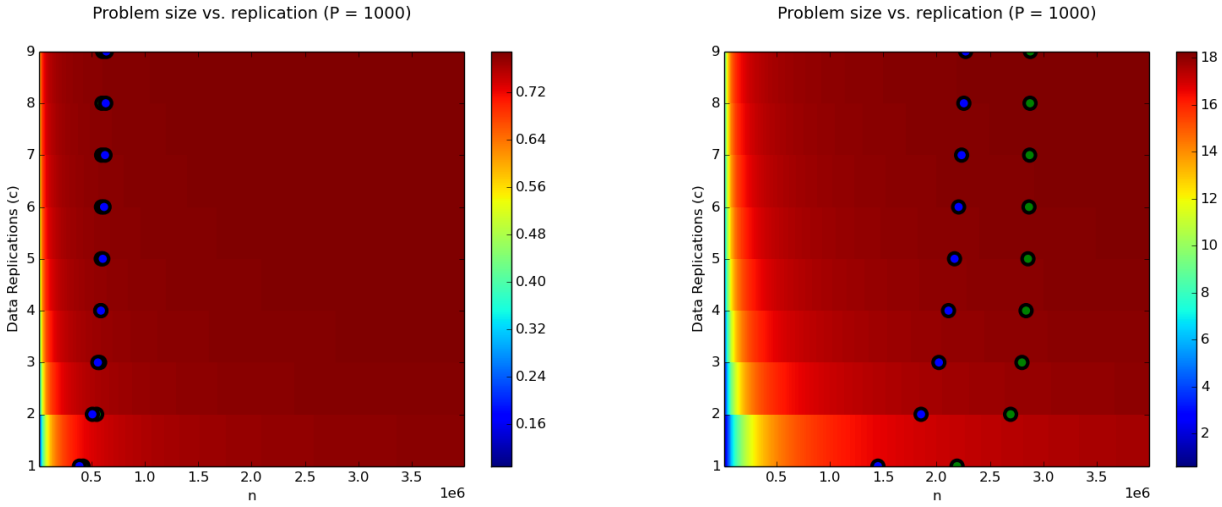
$$C^{NB}_{perf\_MDP}(M) = \Omega\left(\left[\gamma_e f + \beta_e M^{-1} + \delta_e M \gamma_t f + \delta_e \beta_t + \epsilon_e \gamma_t f + \epsilon_e \beta_t M^{-1}\right]^{-1}\right) \tag{6.20}$$

where we recall that $M = cn/P$ where c is the number of replications of the input data. In Figure 6.4, we see two plots that represent the energy efficiency of the CA $O(n^2)$ n-body algorithm on the Xeon 2650-based (Figure 6.4a) and future distributed (Figure 6.4b) machines (according to model parameters detailed in Table 6.2). We assume the number of processors to be fixed at $P = 1000$, $f = 11$, and scale the problem size from $n = 5000$ to $n = 4000000$. Up to 9 replications of the input data (the y-axis) are considered, and the machine in this range of memory utilization does not attain more than approximately 0.73 Gflops/W. Note that for low memory utilization (bottom left corner of the plots), efficiency is low due to the inability of memory utilization to offset communication costs (terms that include β_e or β_t). Unlike the case of 2.5D matrix-matrix shown in Figure 6.3, idle memory energy does not begin to decrease efficiency for this range of problem sizes due to the linear dependence of M on the problem size n. Via improved hardware parameters, the future machine is able to achieve higher efficiency ($C^{CMM}_{perf\_MDP} \approx 18$), but this peak efficiency is attained at noticeably larger problem sizes than the case of the Xeon 2650-based machine.

Figure 6.4 also includes example runtime ($T_{max} = 0.02$) and energy ($E_{max} = 5000$) bounds for both machine models as blue and green points, respectively. The bounds are represented as discrete points for each integer value of c as the current CA $O(n^2)$ n-body algorithm is only able to utilize full replications of the data, as opposed to utilizing a continuous range of additional memory. Runs that attain the bounds are in the regions to the left of the colored points, and we note that replicating data allows for larger problem sizes to be run before the bound is reached. This only provides a significant benefit for the first few replications, as communication costs are decreased by a factor of $1/c$.

6.5 Programs that access arrays with subsets of the iteration variables

Via the theoretical apparatus constructed in Chapter 5, we were able to lower bound the runtime and energy consumption for programs that access arrays via affine expressions of the iteration variables. In this section, we will analyze a subset of such problems: programs that access arrays via subsets of the iteration variables. This includes both $O(n^3)$ classical matrix-matrix multiplication and the $O(n^2)$ n-body problem, and thus generalizes earlier analysis in this chapter. Assuming that certain assumptions hold (see Section 5.2 and Christ et al. [45] for details), such programs can be executed in a communication-optimal manner via Algorithm 10. Assuming that the program is in such a form, the network topology is sufficient such that communication is not dominated by contention, and that the memory-independent per-processor bound does not dominate, the runtime and energy lower bounds for the DP1 machine model (Equations (5.12) and (5.13)) become



(a) Xeon 2650-based distributed machine

(b) Future distributed machine

Figure 6.4: CA $O(n^2)$ n-body: Effect of replicating memory on energy efficiency

$$T^{HBL}_{DP1\_MDP}(M,P,F) = \Omega\left(\frac{\gamma_t F}{P} + \frac{\beta_t F}{PM^{s_{\text{HBL}}-1}} + \frac{\alpha_t F}{mPM^{s_{\text{HBL}}-1}}\right) \tag{6.21}$$

and

$$\begin{aligned} E^{HBL}_{DP1\_MDP}(M,F) &= \Omega\left((\gamma_e+\gamma_t\epsilon_e)F + \left((\beta_e+\beta_t\epsilon_e) + \frac{(\alpha_e+\alpha_t\epsilon_e)}{m}\right)\frac{F}{M^{s_{\text{HBL}}-1}}\right. \\ &\quad \left. +\delta_e\gamma_t MF + \left(\delta_e\beta_t + \frac{\delta_e\alpha_t}{m}\right)M^{2-s_{\text{HBL}}}F\right). \end{aligned} \tag{6.22}$$

where "MDP" stands for the "Memory Dependent per-Processor" bound.

In addition to $O(n^3)$ matrix-matrix multiplication and the $O(n^2)$ n-body problem, another example of a computational problem bounded below in communication by the results of Chapter 5 is outlined via Algorithm 11 which can be described as computing interactions between particles that depend on three particles at a time, not two as in Algorithm 6. As the problem in Algorithm

Algorithm 11 3-body interactions

Require: Arrays K and L of length n

1: **for** $i = 1 : n$, **parallel do**
2: **for** $j = 1 : n, k = 1 : n$, **do**
3: $K(i)$ += $func(L(i), L(j), L(k))$
4: **end for**
5: **end for**

11 has arrays that are accessed via subsets of the iteration variables and if suppose the assumptions of Theorem 4.1 of [45], we can calculate that $s_{\text{HBL}} = 3$ via the solution of the linear program *xLP* as described in Section 5.2. If $s_j \in [0, 1]$ and $s_{\text{HBL}} = \sum_{j=1}^{q} s_j$, s_{HBL} is upper bounded by q. We actually have $q = 4$ in Algorithm 11 as array accesses may be aliased according to [45]. Thus, the bound is not violated. With a value for s_{HBL}, we can bound the communication volume required to execute the problem when

$$F = n^3, W^{HBL}_{DP1\_MDP}(M, P, n) \geq \frac{n^3}{PM^2}, S^{HBL}_{MDP\_MDP}(M, P, n) \geq \frac{n^3}{mPM^2}.$$

Also, as Algorithm 11 has arrays that are accessed via subsets of the iteration variables, we can attain the communication lower bounds with the blocked parallel Algorithm 10.[8] According to Section 5.2, the range of perfect strong scaling in runtime with constant energy for this problem is

$$\frac{N}{P} \leq M \leq \left(\frac{n^3}{P}\right)^{1/3}$$

where the problem size in memory is $N = 2n$, and the right-hand side expression is the minimal amount of memory required to fit the problem and the left-hand side represents the maximum possible amount of memory before assumptions of load balancing break down. In this portion of the chapter, we will consider the impact of constants upon the larger class of problems lower bounded by the results of Chapter 5 and will use the computational problem of Algorithm 11 as a specific example of such problems when necessary for clarity.

Minimizing energy for the computation. To minimize energy, and similar to the cases of $O(n^3)$ matrix-matrix multiplication and the $O(n^2)$ n-body problem, we must solve for the stationary points of the derivative w.r.t. M of the energy bound in Equation (6.22):

$$0 = \frac{dE^{HBL}_{DP1\_MDP}(M, F)}{dM} = B(1 - s_{\text{HBL}})M^{-s_{\text{HBL}}} + C + D(2 - s_{\text{HBL}})M^{1-s_{\text{HBL}}} \quad (6.23)$$

where

$$B = \left((\beta_e + \beta_t\epsilon_e) + \frac{(\alpha_e + \alpha_t\epsilon_e)}{m}\right)F,\ C = \delta_e\gamma_t F,\ D = \left(\delta_e\beta_t + \frac{\delta_e\alpha_t}{m}\right)F.$$

In contrast to the two specific problems discussed previously, a closed-form solution for the optimal amount of local memory (M_{opt}) is wanting due to lack of knowledge regarding the exponents of M. In the case of the problem shown in Algorithm 11 (i.e. $s_{\text{HBL}} = 3$), we must solve for the roots of the cubic equation

$$-2B + CM^3 - DM = 0$$

[8] For this specific example, we assume the cost of *func()* is negligible. If desired, one could account for the floating point cost of this function by multiplying the computation costs ($\gamma_t F$ and $\gamma_e F$) by a constant factor f (similarly to the n-body problem described in Section 4.2).

to obtain the optimal value, M_{opt}. Assuming that we are running the communication-optimal blocked parallel Algorithm 10, perfect strong scaling with constant energy is attained for $2n/P \leq M \leq (F/P)^{1/s_{\text{HBL}}}$. Thus, it is possible to attain the minimum energy use for P in the range

$$\frac{2n}{M_{opt}} \leq P \leq \frac{F}{M_{opt}^{s_{\text{HBL}}}}.$$

Minimizing energy given an upper bound on the runtime. Assuming the problem is of form amenable to use Algorithm 10, there are two cases if we have an upper bound on runtime, T_{max}:

1. It is possible to achieve the required runtime T_{max} and use the minimum amount of energy. This is the case if $T_{DP1\_MDP}^{HBL} \leq T_{max}$ when $M = M_{opt}$. In this case, for example, taking $P = F/M_{opt}^{s_{\text{HBL}}}$ and $M = M_{opt}$ is optimal.
2. Otherwise, we need to run Algorithm 10 with the largest amount of replication (i.e. the largest value of c prior to violation of the load imbalance assumptions discussed in Chapter 5 and [45]) to achieve T_{max}. In this case, we must use at least $P_{T_{max}}$ processors, where the time to run the algorithm equals T_{max}. So, we must solve

$$T_{max} = T_{DP1\_MDP}^{HBL}(M, P_{T_{max}}, F)$$

$$T_{max} = \frac{\gamma_t F}{P_{T_{max}}} + \frac{\beta_t F}{P_{T_{max}} M^{s_{\text{HBL}}-1}} + \frac{\alpha_t F}{m P_{T_{max}} M^{s_{\text{HBL}}-1}}$$

$$T_{max} = \frac{F\left(\gamma_t m M^{s_{\text{HBL}}-1} + \beta_t m + \alpha_t\right)}{m P_{T_{max}} M^{s_{\text{HBL}}-1}}$$

for $P_{T_{max}}$ when $M = F^{1/s_{\text{HBL}}}/P_{T_{max}}^{1/s_{\text{HBL}}}$ (as we recall that $M = cN/P$ where $c \leq (P^{s_{\text{HBL}}-1}F)^{1/s_{\text{HBL}}}/N$ via Equation (5.25)). Finally, one chooses the value of P that minimizes $E_{DP1\_MDP}^{HBL}$ when running Algorithm 10 with $c = (P^{s_{\text{HBL}}-1}F)^{1/s_{\text{HBL}}}/N$, subject to the constraint that $P \geq P_{T_{max}}$.

Minimizing runtime given an upper bound on energy. Conversely, suppose we fix the maximum allowed energy E_{max} and want to minimize the running time with the communication-optimal Algorithm 10. Minimizing $T_{DP1\_MDP}^{HBL}$ will always use the maximum amount of memory, since increasing P during use of the algorithm until it hits the upper bound $P \leq F/M^{s_{\text{HBL}}}$ decreases $T_{DP1\_MDP}^{HBL}$. Further, the runtime using the largest amount of memory is a decreasing function of P, so we only need to determine the maximum P such that the algorithm fits in the energy bound. This value of P, $P_{E_{max}}$, is given by solving the following equation with $M = F^{1/s_{\text{HBL}}}/P_{E_{max}}^{1/s_{\text{HBL}}}$:

$$0 = AP_{E_{max}}^{1+1/s_{\text{HBL}}} + BP_{E_{max}} + CP_{E_{max}}^{2/s_{\text{HBL}}} + CP_{E_{max}}^{1/s_{\text{HBL}}}$$

where

$$A = F^{1/s_{\text{HBL}}}\left(\beta_e + \frac{\alpha_e}{m} + \epsilon_e\beta_t + \frac{\epsilon_e\alpha_t}{m}\right).$$
$$B = F^{2/s_{\text{HBL}}}\delta_e\left(\beta_t + \frac{\alpha_t}{m}\right)$$
$$C = F(\gamma_e + \epsilon_e\gamma_t) - E_{max}$$
$$D = F^{1+1/s_{\text{HBL}}}\delta_e\gamma_t$$

and we note that the equation can not admit a real root unless $E_{max} > F(\gamma_e + \epsilon_e\gamma_t)$.

Minimizing the per-processor average power. By considering that average power for a processor is $h^{HBL}_{DP1\_MDP} = E^{HBL}_{DP1\_MDP}/(PT^{HBL}_{DP1\_MDP})$ and assuming that all processors complete the task utilizing the same amount of time and energy, we can combine Equations (6.21) and (6.22) for $T^{HBL}_{DP1\_MDP}$ and $E^{HBL}_{DP1\_MDP}$ to obtain an expression for power $h^{HBL}_{DP1\_MDP}$:

$$\begin{aligned}
h^{HBL}_{DP1_MDP}(M,P,F) &= \frac{E^{HBL}_{DP1_MDP}(M,F)}{PT^{HBL}_{DP1_MDP}(M,P,F)} \\
&= \Omega\left(\frac{\gamma_e F + \frac{F}{M^{s_{\text{HBL}}-1}}\left(\beta_e + \frac{\alpha_e}{m}\right) + (\delta_e M + \epsilon_e)PT^{HBL}_{DP1_MDP}(M,P,F)}{PT^{HBL}_{DP1_MDP}(M,P,F)}\right) \\
&= \Omega\left(\frac{\left(\gamma_e F + \frac{F}{M^{s_{\text{HBL}}-1}}\left(\beta_e + \frac{\alpha_e}{m}\right)\right)}{\left(\gamma_t F + \frac{F}{M^{s_{\text{HBL}}-1}}\left(\beta_t + \frac{\alpha_t}{m}\right)\right)} + \delta_e M + \epsilon_e\right)
\end{aligned}$$

thus,

$$h^{HBL}_{DP1\_MDP}(M) = \Omega\left(\frac{\gamma_e m M^{s_{\text{HBL}}-1} + \beta_e m + \alpha_e}{\gamma_t m M^{s_{\text{HBL}}-1} + \beta_t m + \alpha_t} + \delta_e M + \epsilon_e\right) \tag{6.24}$$

As similar to the case of 2.5D $O(n^3)$ matrix-matrix multiplication and the CA $O(n^2)$ n-body algorithm, we substitute $M = cN/P$ into Equation (6.25) and assume that $x = c/P$ to simplify the analysis. We obtain:

$$\begin{aligned}
h^{HBL}_{DP1_MDP}(xN) &= \Omega\left(\frac{\gamma_e m N^{s_{\text{HBL}}-1}x^{s_{\text{HBL}}-1} + \beta_e m + \alpha_e}{\gamma_t m N^{s_{\text{HBL}}-1}x^{s_{\text{HBL}}-1} + \beta_t m + \alpha_t} + \delta_e N x + \epsilon_e\right) \\
&= \Omega\left(\frac{Ax^{s_{\text{HBL}}-1} + B}{Cx^{s_{\text{HBL}}-1} + D} + Jx + \epsilon_e\right)
\end{aligned}$$

where $A = \gamma_e m N^{s_{\text{HBL}}-1}$, $B = \beta_e m + \alpha_e$, $C = \gamma_t m N^{s_{\text{HBL}}-1}$, $D = \beta_t m + \alpha_t$ and $J = \delta_e N$. To minimize $h^{HBL}_{DP1\_MDP}$, we must consider the critical points of the previous expression by solving $dh^{HBL}_{DP1\_MDP}/dx = 0$. The expression then becomes

$$\frac{dh^{HBL}_{DP1\_MDP}(xN)}{dx} = \frac{(s_{\text{HBL}} - 1)x^{s_{\text{HBL}}-2}(A(Cx^{s_{\text{HBL}}-1} + D) - C(Ax^{s_{\text{HBL}}-1} + B))}{(Cx^{s_{\text{HBL}}-1} + D)^2} + J = 0.$$

In a similar analysis to the cases of 2.5D $O(n^2)$ matrix-matrix multiplication and the CA $O(n^2)$ n-body algorithm, we can simplify the above derivative

$$C^2Jx^{2s_{\text{HBL}}-2} + 2CDJx^{s_{\text{HBL}}-1} + (s_{\text{HBL}} - 1)(AD - BC)x^{s_{\text{HBL}}-2} + D^2J = 0 \qquad (6.25)$$

where we note that $AD - BC$ must be negative for a positive real root to exist. This is the case elaborated upon for the specific algorithms (2.5D $O(n^3)$ matrix-matrix multiplication and CA $O(n^2)$ n-body), and occurs when $s_{\text{HBL}} > 1$ (by the definition of program xLP in Theorem 5.2, $s_{\text{HBL}} \geq 1$) and

$$\frac{\beta_e m + \alpha_e}{\gamma_e} \geq \frac{\beta_t m + \alpha_t}{\gamma_t}.$$

As noted previous, this inequality can be interpreted as: the number of flops that can be performed using the same *energy* as sending a maximum-length message is greater than the number of flops that can be performed in the same *time* as sending a maximum-length message. On the other hand, if the the $x^{s_{\text{HBL}}-2}$ term in Equation (6.25) is nonnegative, power is an increasing function of c/P for all positive values, and the minimum average per-processor power is attained by setting $c = 1$ and P as large as possible, and the approaches an identical limit to the cases of the 2.5D matrix-matrix and CA $O(n^2)$ n-body algorithms (Equation (6.9)).

Minimizing runtime or energy given a bound on average power By considering that the total average power consumed is $H^{HBL}_{DP1\_MDP} = E^{HBL}_{DP1\_MDP}/T^{HBL}_{DP1\_MDP} = Ph^{HBL}_{DP1\_MDP}$, we can use Equations (6.21) and (6.22) for $T^{HBL}_{DP1\_MDP}$ and $E^{HBL}_{DP1\_MDP}$ to obtain an expression for total average power $H^{HBL}_{DP1\_MDP}$:

$$H^{HBL}_{DP1\_MDP}(M, P) = \Omega\left(P\left(\frac{\gamma_e M^{s_{\text{HBL}}-1}m + \beta_e m + \alpha_e}{\gamma_t M^{s_{\text{HBL}}-1}m + \beta_t m + \alpha_t} + \delta_e M + \epsilon_e\right)\right) \qquad (6.26)$$

An upper bound on total power (H_{max}) thus translates into an upper bound on the number of processors:

$$P \leq P_{H_{max}} = \Omega\left(H_{max}\left(\frac{\gamma_e M^{s_{\text{HBL}}-1}m + \beta_e m + \alpha_e}{\gamma_t M^{s_{\text{HBL}}-1}m + \beta_t m + \alpha_t} + \delta_e M + \epsilon_e\right)^{-1}\right) \qquad (6.27)$$

Using the maximum number of processors, the running time simply becomes

$$T^{HBL}_{DP1\_MDP}(M, P_{H_{max}}, F) = \frac{E^{HBL}_{DP1\_MDP}(M, F)}{H_{max}},$$

and the problem of minimizing time or energy given total power is reduced to minimizing energy, which we have already solved, with the additional constraint (6.27) between P and M.

Alternately we may want to minimize the runtime given a bound on the power per processor (h_{max}). The bound is

$$h_{max} = \Omega\left(\frac{\gamma_e M^{s_{\text{HBL}}-1} m + \beta_e m + \alpha_e}{\gamma_t M^{s_{\text{HBL}}-1} m + \beta_t m + \alpha_t} + \delta_e M + \epsilon_e\right),$$

which we may solve for M. To minimize $T^{HBL}_{DP1\_MDP}$, we would as many processors as possible, and as much memory as possible subject to the constraint on M formed by solving the above inequality and $M \leq (F/P)^{1/s_{\text{HBL}}}$.

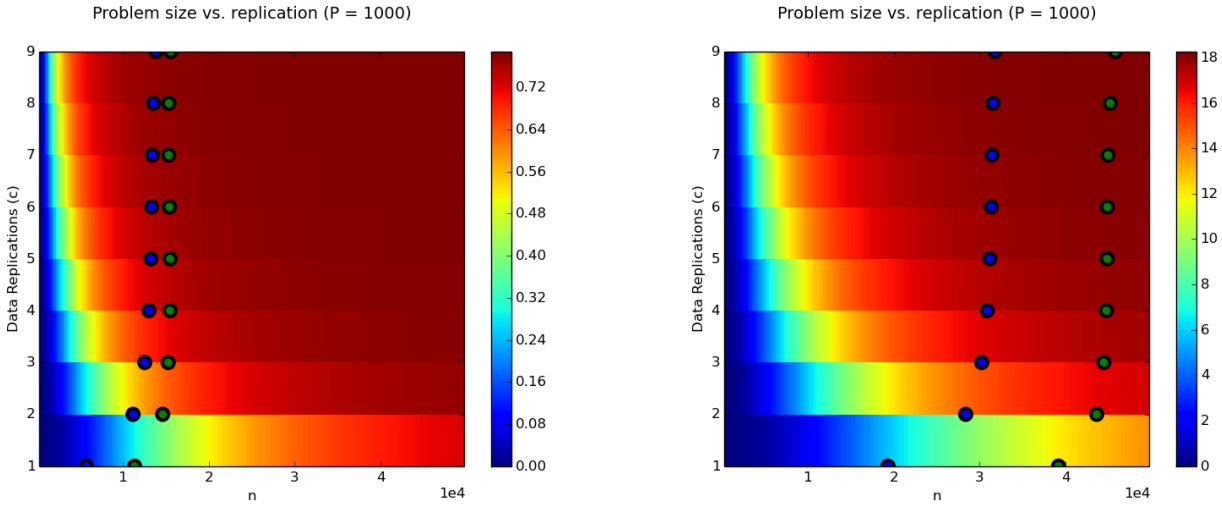
If M_{opt} is in the range allowed by h_{max}, then the global minimum energy can be attained within a per-processor power budget h_{max}. If not, since $E^{HBL}_{DP1\_MDP}$ is a decreasing function of M for $M < M_{opt}$ (where M_{opt} is the energy-optimal amount of memory obtained by solving Equation (6.23)), the minimum energy is when M takes its maximum value allowed by the above constraint on M and P is anywhere in the range $\frac{N}{M} < P < \frac{F}{M^{s_{\text{HBL}}}}$.

We now consider the effect of scaling problem size and number of data replications on the metric of energy efficiency (C_{perf_MDP}). If the 3-body interaction problem of Algorithm 11 is computed with the communication-optimal parallel blocked Algorithm 10, $s_{\text{HBL}} = 3$, and Equation (6.2) becomes

$$C^{3B}_{perf\_MDP}(M) = \Omega\left(\left[\gamma_e + \beta_e M^{-2} + \delta_e M\gamma_t + \delta_e\beta_t M^{-1} + \epsilon_e\gamma_t + \epsilon_e\beta_t M^{-2}\right]^{-1}\right) \tag{6.28}$$

where we recall that $M = cn/P$ where c is the number of replications of the input data and "3B" stands for "3-Body". In Figure 6.5, we see two plots that represent the energy efficiency of Algorithm 10 computing Algorithm (11) on the Xeon 2650-based (Figure 6.5a) and future distributed (Figure 6.5b) machines (according to model parameters detailed in Table 6.2). We assume the number of processors to be fixed at $P = 1000$, and scale the problem size from $n = 250$ to $n = 50000$. Up to 9 replications of the input data (the y-axis) are considered, and the machine in this range of memory utilization does not attain more than approximately 0.73 Gflops/W. Note that for low memory utilization (bottom left corner of the plots), efficiency is low due to the inability of memory utilization to offset communication costs (terms that include β_e or β_t). Unlike the case of 2.5D matrix-matrix shown in Figure 6.3, idle memory energy does not begin to decrease efficiency (for this range of problem sizes) due to the linear dependence of M on the problem size n. Via improved hardware parameters, the future machine is able to achieve higher efficiency ($C^{3B}_{perf\_MDP} \approx 18$), but this peak efficiency is attained at noticeably larger problem sizes than the case of the Xeon 2650-based machine.

Figure 6.4 also includes example runtime ($T_{max} = 0.01$) and energy ($E_{max} = 5000$) bounds for both machine models as blue and green points, respectively. The bounds are represented as discrete points for each integer value of c as the communication-optimal parallel blocked algorithm is only able to utilize full replications of the data, as opposed to utilizing a continuous range of additional memory. Runs that attain the bounds are in the regions to the left of the colored points, and we



(a) Xeon 2650-based distributed machine

(b) Future distributed machine

Figure 6.5: 3-Body Problem: Effect of replicating memory on energy efficiency

note that replicating data allows for larger problem sizes to be run before the bound is reached. This only provides a benefit for the first few replications, as communication decreases by a factor of $1/c^2$. In this chapter, we fixed the hardware parameters (γ_t,γ_e,β_t,β_e,δ_e,ϵ_e) of runtime and energy bounds (see Chapters 4 and 5) and considered a set of interesting problems that can be addressed via such bounds on distributed parallel machines. Many of these problems involve a bound on runtime, energy, or power. We also used two specific examples of distributed parallel machines to highlight the impact of scaling problem size and input data replication on energy efficiency (C_{perf},Equation (6.2)). In the next chapter, we will allow the hardware parameters to vary to attain targeted values of C_{perf} and also optimize for the financial cost per job over a range of machine hardware choices.

Chapter 7

Implications for Hardware Designs

In Chapter 6, we explored questions related to the energy lower bounds derived in Chapters 4 and 5 assuming fixed hardware parameters (γ_t, γ_e, β_t, β_e, α_t, α_e, ϵ_e) for sequential and distributed parallel machines. Such an approach allows one to address interesting questions about energy consumption and runtime as well as highlighting the potential benefits of algorithms that are designed to reduce communication via additional memory utilization. In this chapter, we take the analysis further and pose the question: *How do hardware parameters affect an efficiency metric?*

With this question in mind, in Section 7.2 we present a new way to visualize improvements in hardware energy efficiency: the Cityscape model.[1] Via greedy scaling of terms in the expression for energy efficiency (Equation (6.2)), Cityscape allows hardware experts to generate a set of constraints on the parameters of runtime and energy models that are needed to attain a targeted level of improvement in energy efficiency. In addition to Cityscape, we also consider the financial cost to execute a computational kernel of a specific size under different machine hardware configurations (Section 7.3). This is a useful metric for the design of large systems, as capital (CapEx) (e.g. purchasing compute hardware) and operational (OpEx) (e.g. paying for the energy of a kernel execution) expenses often represent significant financial expense. Scripts used to generate the figures presented in this chapter can be found at `https://github.com/agearh/dissertation.git`.

7.1 Introduction

Prior to any further analysis, we must first include assumptions and refine the desired question. Due to the nature of our models and the hardware, this approach has several major assumptions:

- As noted previously, a machine capable of a high level of efficiency will not attain that efficiency without an efficient implementation of the algorithm.

[1] We use the name Cityscape because a machine is modeled by a collection of Roofline models [148].

- By basing models on the performance of existing processors, the hardware space under consideration is limited to the regime of processors that reasonably resemble previous hardware designs.

We address these limitations by first assuming a perfect implementation of an algorithm: i.e. that the software is able to attain the asymptotic communication bounds of the algorithm and also utilize the machine to its highest possible level of efficiency. These two assumptions differ in that an algorithm may not be tuned for target architecture and our models ignore many details, and thus result in low efficiency despite being asymptotically communication-optimal in communication volume (W) and messages (S). This assumption may be reasonable in certain cases due to the existence of throughput-oriented autotuning frameworks (e.g. ATLAS [146]), and is especially applicable in the case of algorithms with high arithmetic intensity (e.g. $O(n^3)$ matrix-matrix multiplication or the 3-body problem from Algorithm 11). We delegate to future research the challenge of determining if tuning for runtime efficiency also achieves a significant fraction of peak energy efficiency (although some evidence does support this idea [44]).

Taking the above list of assumptions into consideration, we modify the research question appropriately:

How do the hardware parameters of general-purpose machines affect an efficiency metric when running a workload that looks asymptotically like a specific algorithm for a target problem?

In other words, we do not consider emerging technologies such as quantum computing nor the development of custom hardware designed to implement a specific algorithm (i.e. field-programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs)). The heterogeneous model may be able to capture the performance of machines that encompass a collection of specialized processors. The problems analyzed in this chapter are assumed to run on the DP1 machine model defined in Section 3.2, in which communication energy is proportional to utilized link bandwidth. We note a similar analysis could be applied to the heterogeneous and DP2 models (see Section 3.2), depending on the nature of the problem and target hardware. For simplicity, we assume that the latency cost of message passing on the distributed machine to be negligible and thus drop the latency terms from the model. Such an assumption may be problematic for machines with a large network diameter, and is worth investigation in future research.

As discussed in Chapter 5, work by Christ et al. [45] has extended previous communication lower bounds to a much larger class of problems, that of programs that access arrays with affine expressions[2]. In Chapter 5, we extended these communication lower bounds to lower bounds on runtime and energy. For a subset of the problems that access their arrays via subsets of the iteration variables (also referred to as the *product case* in [45]), Christ et al. describe blocked algorithms for sequential (Algorithm 5.20) and distributed parallel machines (DP1) (Algorithm 10) that are asymptotically communication-optimal. Problems that fall within the product case include dense matrix-matrix multiplication and the $O(n^2)$ n-body problem as well as the 3-body problem described in Algorithm 11. In Chapter 5, we described the optimal distributed parallel algorithm of Algorithm 10 in further detail and slightly generalized the analysis in [45]. We also showed

[2]Along with certain other assumptions, see Chapter 5 or [45] for more details.

that this algorithm is able to perfectly strong scale in runtime with constant energy. In this chapter, we assume use of this communication-optimal blocked parallel algorithm for such problems when another communication-optimal algorithm is not available with such scaling properties (e.g. 2.5D matrix-matrix multiplication). Via the Cityscape model and financial cost per job (C_{job}), we demonstrate how algorithm designers may evaluate the impact of varying hardware parameters.

7.2 Cityscape Model of Energy Efficiency

In Chapter 6, we visualized the effects of scaling problem size (n) and number of data replications (c) upon energy efficiency (Gflops/W), or C_{perf}, for algorithms that are able to trade memory for communication. This section extends that analysis by considering the question: "How do hardware parameters need to be improved to attain a specific improvement in overall system energy efficiency?". With this question in mind, we present a new way to visualize the impact of hardware parameters upon energy efficiency: the Cityscape model. Cityscape builds upon previous work on Roofline models for floating point throughput [148] and energy [44] and allows hardware designers to constrain the improvements in hardware parameters needed to reach a desired factor of energy-efficiency improvement over a baseline machine (C^*_{perf}). We focus on the DP1 model for distributed parallel machines, but the technique could be applied to both the sequential and heterogeneous machine models as well.

To begin, we modify the definition of energy efficiency (Gflops/W) for programs that reference arrays from Section 6.2 to be

$$C^{HBL}_{perf}(P,F,\Gamma) = \Omega\left(\left[\gamma_e + \beta_e \max\left(M_{hw}^{1-s_{\text{HBL}}}, \left(\frac{P}{F}\right)^{1-1/s_{\text{HBL}}}\right) + (\delta_e M_{hw} + \epsilon_e)\left(\gamma_t + \beta_t \max\left(M_{hw}^{1-s_{\text{HBL}}}, \left(\frac{P}{F}\right)^{1-1/s_{\text{HBL}}}\right)\right)\right]^{-1}\right)$$

where $\Gamma = [\gamma_e, \beta_e, \delta_e, \epsilon_e, \gamma_t, \beta_t, M_{hw}]$. We assume that the algorithm utilizes the entire installed memory M_{hw}, as opposed to the analysis in Chapter 6 where we assumed a fixed M_{hw} and that the application was able to utilize a desired M-sized portion of this fixed amount. We also assume that the network topology is sufficient such that contention costs do not dominate the per-processor bounds (see Section 5.1 for details on contention bounds). If we further assume that the memory-dependent per-processor (MDP) communication bound dominates the memory-independent per-processor bound, energy efficiency becomes

$$C^{HBL}_{perf\_MDP}(\Gamma) = \Omega\left([\gamma_e + \beta_e M_{hw}^{1-s_{\text{HBL}}} + \delta_e M_{hw}\gamma_t + \delta_e\beta_t M_{hw}^{2-s_{\text{HBL}}} + \epsilon_e\gamma_t + \epsilon_e\beta_t M_{hw}^{1-s_{\text{HBL}}}\right]^{-1}\Big). \tag{7.1}$$

In Section 6.2, we defined a distributed parallel machine with dual-socket Intel Xeon nodes and an Ethernet torus network (parameters reproduced in Table 7.1). It is this machine that we use as a

baseline upon which to apply the Cityscape model, with the addition of a fixed amount of installed memory (M_{hw}) of 64GB. Similar sets of parameters for an intensity-variable microbenchmark on several types of existing hardware can be found in [43], and may be readily adapted to our analysis.

	Xeon 2650
Processor Idle Power (W)	14.25
Processor Dynamic Power (W)	80.75
DRAM Dynamic Power (W)	3.15
NIC Idle Power (W)	34.17
NIC Dynamic Power (W)	6.03
Peak Network BW (GB/s)	3.75
M_{hw} (GB)	64
γ_t (sec/flop)	3.91E-12
γ_e (joules/flop)	6.31E-10
β_t (sec/word)	6.40E-09
β_e (joules/word)	5.88E-08
δ_e (joules/sec/word)	2.03E-09
ϵ_e (joules/sec)	162.67

Table 7.1: Intel Xeon-based distributed parallel baseline machine

Clearly, if any of the terms in the denominator of Equation (7.1) exceed $1/C^*_{perf}$, a desired level of efficiency C^*_{perf} cannot be attained. Considering Table 7.1, this trivial upper bound (using $\epsilon_e\gamma_t$, as it is the largest term) indicates the peak efficiency to be 1.57 Gflops/Watt. Thus, on current server nodes, idle energy (the dominant parameter in $\epsilon_e\gamma_t$) may dominate communication energy. Along this line of reasoning, one can attain an idea of the best parameters to optimize by considering their magnitudes. This is done in Table 7.2 for the baseline machine described in Table 7.1. The table includes rows for $O(n^3)$ matrix-matrix multiplication ($s_{\text{HBL}} = 1.5$), the $O(n^2)$ n-body problem ($s_{\text{HBL}} = 2$) and the 3-body problem in Algorithm 11 ($s_{\text{HBL}} = 3$). Note that larger values of s_{HBL} increase the potential reuse of local data, and accordingly require less inter-node communication traffic. This decreases the terms that include β_e and β_t. In Table 7.2, we note that the terms that depend on the per-word time (β_t) and energy (β_e) costs are never dominant as the level of available reuse is sufficient to offset communication costs. Thus, for the machine described in Table 7.1, increases in $C^{HBL}_{perf\_MDP}$ can be obtained for each of the example problems described in Table 7.2 via reductions in terms that do not involve the communication interconnect.[3]

To target improvements in hardware parameters, we must observe the effect of decreasing each of the six denominator terms of $C^{HBL}_{perf\_MDP}$ (Equation (7.1)). Thus, we define $z_1, ..., z_6$ to be scaling factors on these terms, i.e;

[3]Only up to a point. Eventually, at least for $s_{\text{HBL}} = 1.5$, the $\epsilon_e\beta_t M_{hw}^{1-s_{\text{HBL}}}$ term will dominate after enough scaling.

s_{HBL}	γ_e	$\beta_e M_{hw}^{1-s_{\text{HBL}}}$	$\delta_e M_{hw} \gamma_t$	$\delta_e \beta_t M_{hw}^{2-s_{\text{HBL}}}$	$\epsilon_e \gamma_t$	$\epsilon_e \beta_t M_{hw}^{1-s_{\text{HBL}}}$
1.5	6.31E-10	6.34E-13	6.80E-11	1.20E-12	6.35E-10	1.12E-11
2.0	6.31E-10	6.84E-18	6.80E-11	1.30E-17	6.35E-10	1.21E-16
3.0	6.31E-10	7.97E-28	6.80E-11	1.51E-27	6.35E-10	1.41E-26

Table 7.2: Energy efficiency terms for several problems (all are measured in the same units, namely joules/flop)

$$C_{perf\_MDP}^{HBL}(\Gamma) = \Omega\left(\left[\frac{\gamma_e}{z_1} + \frac{\beta_e M_{hw}^{1-s_{\text{HBL}}}}{z_2} + \frac{\delta_e M_{hw}\gamma_t}{z_3} + \frac{\delta_e \beta_t M_{hw}^{2-s_{\text{HBL}}}}{z_4} + \frac{\epsilon_e \gamma_t}{z_5} + \frac{\epsilon_e \beta_t M_{hw}^{1-s_{\text{HBL}}}}{z_6}\right]^{-1}\right). \tag{7.2}$$

Without loss of generality, let us suppose that the largest term in the denominator of $C_{perf\_MDP}^{HBL}$ is term 1 (γ_e). We then begin increasing z_1 by a fixed factor g (decreasing γ_e and thus increasing $C_{perf\_MDP}^{HBL}$ until another term (say, term 2) becomes dominant. We then begin to increase z_2 by g along with z_i. This process of greedily decreasing the largest terms in the denominator of $C_{perf\_MDP}^{HBL}$ continues until the target energy efficiency, C_{perf}^{*}, is attained.

In Figure 7.1, we apply this scaling scheme to $O(n^3)$ dense matrix-matrix multiplication ($s_{\text{HBL}} = 3/2$) and demonstrate a Cityscape model for this algorithm and machine with $g = 1.25$. The x-axis of this log-log plot represents the factor of energy efficiency improvement over the baseline (C_{perf}^{*}) and the y-axis shows the factor of increase in the terms z_i. From the values shown in Table 7.2, we see that $\epsilon_e \gamma_t$ (corresponding to z_5) dominates followed by γ_e (corresponding to z_1). Indeed, the Cityscape of Figure 7.1 begins increasing z_5 by g and then almost immediately begins increasing z_1 as γ_e starts to dominate. This process results in a figure with a handful of "rooflines" that correspond to the scaling of terms in $C_{perf\_MDP}^{HBL}$.

As the terms of $C_{perf\_MDP}^{HBL}$ are typically composed of several energy and runtime parameters, the Cityscape model defines a set of constraints on these parameters. As an example, suppose a hardware designer wishes to increase the energy efficiency of the baseline machine running $O(n^3)$ matrix-matrix multiplication by a factor of 10 (i.e. $C_{perf}^{*} = 10$). From Figure 7.1, the x-axis value of $C_{perf}^{*} = 10$ results in $\{z_1, ..., z_6\} = \{14.55, 1.00, 1.56, 1.00, 18.19, 1.00\}$. We now consider the fact that the terms scaled by the z_i are actually combinations of hardware parameters, and that each z_i can be viewed as a function of individual parameter scaling factors. For example, $z_5 = z_{\gamma_t} z^{\epsilon_e}$ or $z_2 = z_{\beta_t} z_{M_{hw}}^{-1/2}$. Thus, the parameter scaling factors are $\{z_{\gamma_e}, z_{\gamma_t}, z_{\beta_e}, z_{\beta_t}, z_{\delta_e}, z_{\epsilon_e}, z_{M_{hw}}\}$, and from the Cityscape model we can form a set of constraints on the hardware parameters, which can be

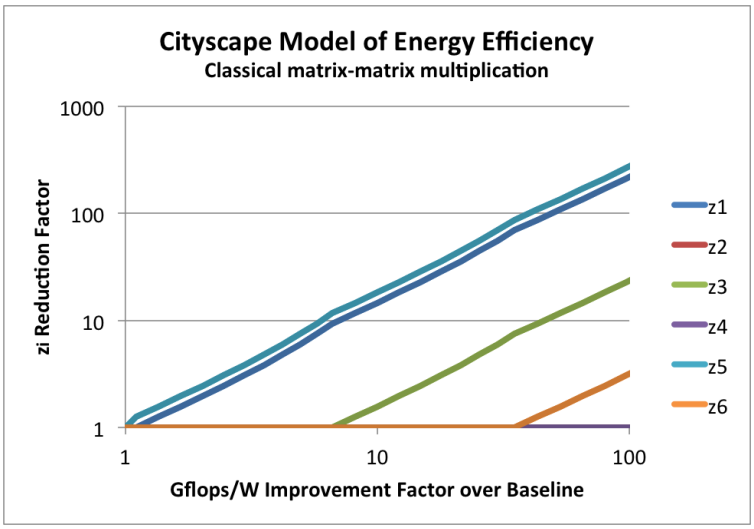


Figure 7.1: Example Cityscape Model for $O(n^3)$ Matrix-matrix multiplication

linearized by applying logarithms:

$$
\begin{aligned}
log(z_1) &= log(z_{\gamma_e}) \geq log(14.55) \\
log(z_2) &= log(z_{\beta_e}) - \frac{1}{2} log(z_{M_{hw}}) \geq 0 \\
log(z_3) &= log(z_{\delta_e}) + log(z_{M_{hw}}) + log(z_{\gamma_t}) \geq log(1.56) \\
log(z_4) &= log(z_{\delta_e}) + log(z_{\beta_t}) + \frac{1}{2} log(z_{M_{hw}}) \geq 0 \\
log(z_5) &= log(z_{\epsilon_e}) + log(z_{\gamma_t}) \geq log(18.19) \\
log(z_6) &= log(z_{\epsilon_e}) + log(z_{\beta_t}) - \frac{1}{2} log(z_{M_{hw}}) \geq 0.
\end{aligned} \tag{7.3}
$$

These constraints imply tradeoffs in the hardware design process; e.g. the designer may find it easier to improve M_{hw} as opposed to δ_e or γ_t, and so concentrate on that parameter attain an overall improvement of 1.56 in term three ($\delta_e M_{hw} \gamma_t$). Note that the above formulation of the constraints may be overly-restrictive, as stating that $log(z_2) \geq 0$ means that z_2 cannot get smaller (increasing the size of $\beta_e M^{1-s_{\text{HBL}}}$). If this term is extremely small, it could become significantly larger without dominating. To allow such flexibility, one could alternatively formulate the constraints in this manner:

$$\frac{\beta_e M_{hw}^{1-s_{\text{HBL}}}}{z_2} \leq \frac{\gamma_e}{z_1}$$

or perhaps

$$\frac{\beta_e M_{hw}^{1-s_{\text{HBL}}}}{z_2} \leq \frac{a\gamma_e}{z_1}$$

where $0 \leq a \leq 1$. Such alternate constraints can be linearized via logarithms similarly to the constraints of (7.3).

Two limitations of the current version of the Cityscape model are that it assumes parameters can be scaled independently (e.g. increasing M_{hw} does not effect β_e) and that one parameter is as easy to scale as another (e.g. increasing the size of installed memory, M_{hw}, is probably easier than reducing γ_e). However, provided with the constraints of (7.3), the hardware designer is able to target improvements at parameters with the greatest impact upon energy efficiency. This could be combined with energy and runtime bounds on machines represented as compositions of the heterogeneous, sequential and distributed parallel models discussed in Chapter 3 to form constraint sets on more specific hardware parameters, such as the size of last level cache or the time cost of moving a word between L1 and L2 caches. We believe that the Cityscape model could be a component of a future hardware autotuning system that evaluates a space of potential hardware designs for energy efficiency.

7.3 Financial cost/Job (C_{job})

In this section, we define the metric of financial cost (in USD) per job, or C_{job}. We assume the machine has a fixed lifetime (c_{life}) and a fixed energy cost per joule (c_{energy}). While there are many factors that contribute to calculating the true cost of owning a distributed parallel machine (see [24], [92] and [115] for more details), the financial cost factor of C_{job}, $C_{financial}$, only considers the capital expenditure of purchasing a target number of servers and networking equipment. Thus,

$$\begin{aligned} C_{financial}(P, G) &= C_{nodes}(P) + C_{net}(P, G) \\ &= P(C_{proc} + C_{mem} + C_{base}) + (C_{NIC}(P, G) + C_{cable}(P, G) + C_{switch}(P, G)) \end{aligned} \tag{7.4}$$

where for number of processors P and network graph G, C_{nodes} is the sum of the node costs and C_{net} is the total cost of network equipment. These two costs are then subdivided further. We can represent C_{nodes} as the sum of processor cost (C_{proc}), memory cost (C_{mem}) and a baseline cost for the other node components (C_{base}). The network cost, C_{net}, can be represented as the sum of network interface card (NIC) costs (C_{NIC}), cabling costs (C_{cable}) and routers/switches (C_{switch}).

We can now combine this expression for $C_{financial}$ with our DP1 runtime and energy models (Equations (3.3) and (3.4), respectively, and modified to vary P and Γ) to obtain C_{job}:

$$
\begin{aligned}
C_{job}(P,G,\Gamma) &= \frac{C_{financial}(P,G)T_{DP1}^{HBL}(P,\Gamma)}{c_{life}} + E_{DP1}(P,\Gamma)c_{energy} \\
&= \frac{1}{c_{life}}\Bigg(P(C_{proc}+C_{mem}+C_{base}) + (C_{NIC}(P,G) + C_{cable}(P,G) \\
&\quad + C_{switch}(P,G))\Bigg)(\gamma_t F + \beta_t W_t) \\
&\quad + c_{energy}P(\gamma_e F + \beta_e W_e + (\delta_e M_{hw} + \epsilon_e)(\gamma_t F + \beta_t W_t))
\end{aligned} \tag{7.5}
$$

where c_{life} is the machine lifetime in seconds, c_{energy} is the dollar cost per joule, $\Gamma = [\gamma_e, \beta_e, \delta_e, \epsilon_e, \gamma_t, \beta_t, M_{hw}]$, and W_t and W_e are the runtime and energy communication lower bounds expressed in Equations (5.6) and (5.8), respectively. Again, for simplicity, latency terms are eliminated from the expression. Note that unlike the previous section, we make no assumptions about dominance of any one of the communication bounds. Note that T_{DP1}^{HBL}/c_{life} is the fraction of the life of the machine to execute one job, so the first term $C_{financial}T_{DP1}^{HBL}/c_{life}$ is the same fraction of the cost of the machine. The second term of C_{job} is the energy in joules E_{DP1} to run one job time the cost in dollars per joule, c_{energy}. Thus, C_{job} is a combination of the amortized cost of server hardware and the energy to operate the machine during the task.

A more detailed cost metric would include additional operating (e.g. the cost of machine cooling) and capital expenditures (such as building construction). Details on the construction of more-detailed cost metrics can be found in [92], [115] and via the Sustained System Performance (SSP) metric [93]. Future work may attempt to include aspects of such models to increase the accuracy of our analysis.

To present an example analysis on the DP1 distributed parallel machine model, we assume that the machine is comprised of dual-socket multicore nodes with processors that resemble Intel's Sandy Bridge-EP. We assume that this node has a base power consumption of 100W and that $C_{base} = \$1943.96$. The base cost is calculated from an IBM System x3550 M4-x791462U 1U rack server ($4959.00),[4] once the cost of processors and memory have been subtracted.[5] We assume use of a communication-optimal parallel blocked Algorithm 10 to compute Algorithm 11 (see Chapter 5). Recall that this algorithm has $s_{\text{HBL}} = 3$ and $F = n^3$. As we make no assumptions about which communication bound dominates, we cannot eliminate the problem size, n, from discussion. It is thus assumed that $n = 100000$, the number of processors is from 1000-9000 in increments of 1000, and the installed node memory size (M_{hw}) is between 16GB and 128GB in increments of 16GB. We also assume c_{energy} to be 1.894e-8 dollars/joule, or 6.82 cents/KWhr, the average industrial electricity rate in the United States during 2013 [62].[6] We also assume a machine lifetime of 5 years (or 1.57785e8 seconds). These parameters are summarized in Table 7.3.

[4] Pricing obtained from `http://www-304.ibm.com/`

[5] Processors are 2 E5-2665 at $2880, and 8GB of DDR3-1600 at $125.05. See later tables for more information.

[6] 1 KWhr = 3.6e6 joules.

Word Size (bytes)	8
s_{HBL}	3
Problem Size (n)	100000
F	n^3
Nodes (P)	1000-9000
Node Memory (M_{hw}, in words)	16GB-128GB
Node Base Power	100W
AVX Vectorization	4
Fused-Multiply-Add (FMA)	2
$C_{base}(dollars)$	1943.96
$c_{life}(sec)$	1.5778e8
$c_{energy}(dollars/joule)$	1.894e-8

Table 7.3: Additional model values

The data for a range of Intel Sandy Bridge-EP processors can be found in Table 7.4.[7] In this analysis, we assume that the algorithm in question (an optimal implementation of Algorithm 11, see Chapter 5) attains the peak floating point rate while running at the Thermal Design Power (TDP) for the processor. We also assume that 15% of the peak processor power is idle, and contributes to the machine's static energy consumption. Note that Sandy Bridge-EP processors can compute four double-precision floating point operations in parallel via Advanced Vector Extension (AVX) vectorization, and also execute paired multiplications and additions in parallel (fused multiply-add, or FMA). Thus, each processor can compute 8 double-precision operations per cycle.

Processor Model	Frequency (Ghz)	Cores	Power (W)	Idle/Total Power	Cost
E5-2650L	1.8	8	70	0.2	1107.00
E5-2650	2.0	8	95	0.2	1107.00
E5-2658	2.1	8	95	0.2	1186.00
E5-2660	2.2	8	95	0.2	1329.00
E5-2665	2.4	8	115	0.2	1440.00
E5-2670	2.6	8	115	0.2	1552.00
E5-2680	2.7	8	130	0.2	1723.00
E5-2690	2.9	8	135	0.2	2057.00

Table 7.4: Processor parameters

We also assume that the internode network is a torus of dimension d so as to apply the con-

[7] Processor data obtained from ark.intel.com, with the exception of the ratio of idle to total processor power, which is assumed.

tention lower bound defined in Chapter 5. Thus, network interface cards (NICs) on each compute node are assumed to have a total of $2d$ ports and all connections are assumed to be point-to-point without switches or external routers (i.e. $C_{switch} = 0$). The network technology is assumed to be Ethernet [109, 77], NICs are each assumed to be dual-port, and all network cables are assumed to be 5 meters in length. So, the torus network includes d NICs per node and Pd total cables. NIC and cable parameters can be found in Table 7.5.[8] We also ignore network cable energy losses, and assume that the predominant internode communication energy consumers are the node NICs and DRAM power. The NICs are assumed to consume 85% of peak power at idle. This assumption is likely to be over-generous, as several results have found little to no difference between the idle and active power of NICs [124, 66, 26]. We assume that the active NIC consumes the entire rated TDP power of the device.

NIC BW (Gb/s)	NIC Model	Power (W)	Idle/Power	NIC Cost	Cable Type	Cable Cost
1	Intel I350-T2	4.4	0.85	128.00	RJ-45 Cat-5	3.40
10	Intel X540-T2	13.4	0.85	508.00	RJ-45 Cat-6a	7.32
40	Chelsio T580-LP-CR	17	0.85	1023.49	QSFP+ twinax	154.00

Table 7.5: Network adaptor and cable parameters

Data regarding node DRAM is shown in Table 7.6. We assume use of 16GB, 1.5V, dual-rank, x4 DDR3, error-correcting code (ECC), registered DIMM modules, and consider the impact of using one of three different DIMM clock frequencies.[9] As in Chapter 6 and in Section 7.2, we model power data via the Virtium DIMM Memory Power Calculator [58].[10] In the Virtium tool, we assumed On-die termination (ODT) RD/WR Term power to be 100mW and also assumed 100% usage of the DIMM. Due to the DIMM parameters stated earlier, there are 36 DRAM chips per DIMM. The DRAM active (IDD7) and active standby currents (IDD3N) were obtained from Table 20 of Micron's DDR3 SDRAM MT41J1G4 data sheet (as utilized by the Micron DDR3 SDRAM RDIMM MT36JSF2G72PZ data sheet in Table 14).[11] Cost data was obtained for the Micron DIMM modules in Table 7.6 from digikey.com. For example, for the DDR3-1333 part, the data sheet reports that $IDD7 = 190$mA and $IDD3N = 35$mA. When the above assumptions are entered into the Virtium tool, an active DIMM consumes 9.875W. This value is a combination of the active power for one rank of 18 DRAM chips ($(IDD7 * V_{DD} + ODT) * 18$), and the standby power for another rank ($IDD3N * V_{DD} * 18 + 2$, as the calculator assumes 2W of power from registers/AMB and PLLs for registered DIMMs). As one rank must be in standby while the other is active, we use 9.875W/16GB = 0.6171W/GB for our machine model. A similar calculation

[8] Intel NIC data obtained from ark.intel.com, Chelsio NIC data from chelsio.com and cable pricing obtained via Amazon.com.

[9] We assume that the Intel Xeon E5-26XX processors are capable of supporting 1866Mhz DRAM.

[10] Details on the expressions used to generate values for the Virtium calculator can be found in [65]. NB: There is a typo in the examples on this site; the DDR2 power calculation for 2 DIMMs should include standby current from 54 DRAM chips, not 36.

[11] Data sheets obtained from micron.com.

is used for standby power per GB. We assume that the fraction of peak DRAM dynamic power associated with the peak total internode bandwidth contributes to β_e. Thus, if the installed DRAM has a total BW of 100GB/s and the installed NICs have a peak bandwidth of 5GB/s, 5% of peak DRAM power is assumed to be associated with β_e.

Type	Part Num. (Micron)	Cost/Gb	Idle Power/Gb	Dyn. Power/Gb	DRAM BW (w/s/GB)
DDR3-1333	MT36JSF2G72PZ-1G4D1	15.00	0.2431	0.6171	10.60
DDR3-1600	MT36JSF2G72PZ-1G6D1	15.63	0.2533	0.6729	12.80
DDR3-1866	MT36JSF2G72PZ-1G9D1	13.44	0.2634	0.7303	14.90

Table 7.6: DRAM parameters

The parameters for C_{job} can be calculated from Tables 7.3, 7.4, 7.5 and 7.6. Some, such as C_{base}, are straightforward. Others are derived from several table entries, as shown:

$$\begin{aligned} C_{proc} &= 2 * \text{Processor Cost} \\ C_{mem} &= (M_{hw} * \text{DRAM Cost Per GB})/\text{Words Per GB} \\ C_{NIC} &= d * P * \text{NIC cost} \\ C_{cable} &= d * P * \text{Cable cost} \end{aligned} \tag{7.6}$$

Other hardware parameters are calculated identically to the Equations in (6.1), with the exception of β_e. As we assume utilization of all installed memory, we calculate β_e as a function of M_{hw}:

$$\begin{aligned} \beta_t &= 1/\text{Peak Network BW}/\text{Words Per GB} \\ \beta_e &= \beta_t * (\text{DRAM Dynamic Power} + \text{NIC Dynamic Power}) \end{aligned} \tag{7.7}$$

where

$$\begin{aligned} \text{DRAM Dynamic Power} &= M_{hw} * \text{DRAM Peak Power Per Word} * ((1/\beta_t)/\text{DRAM Peak BW}) \\ \text{DRAM Peak Power Per Word} &= \text{Dyn. Power Per GB}/\text{Words Per GB} \\ \text{DRAM Peak BW} &= (\text{DRAM BW Per GB}/\text{Words Per GB}) * M_{hw} \\ \text{Peak Network BW} &= (\text{NIC BW} * d)/\text{Bits Per Byte} \\ \text{Words Per GB} &= 1e9/\text{Word Size}. \end{aligned}$$

As the space of potential parameter values is discrete in this formulation of C_{job}[12], we can calculate the optimal set of model parameters by exhaustive search (see Table 7.7). We observe that for the problem within Algorithm 11, the minimal number of nodes, network dimension, and cheapest memory are optimal. Note that the reduced cost of DDR3-1833 offsets its increased energy consumption with this algorithm and problem size. Due to the high arithmetic intensity

[12] 9 different values of P, 8 values of M_{hw}, 8 types of processors (Table 7.4), 3 network bandwidths (Table 7.5), 3 types of DRAM (Table 7.6) and torus dimension $d = 1..4$ for 20736 total combinations of values.

of this problem, floating point operations dominate the computation. Thus, the optimal design point is to minimize internode communication resources to reduce power and cost. The number of processors is low due to the large base power of a node (100W). One would expect codes with low intensity to require a larger amount of memory and a faster network to minimize C_{job}.

Nodes (P)	1000
Memory Size (M_{hw})	16GB
Torus Dim (d)	1
Processor	Intel E5-2670
Memory	DDR3-1866
Network	1Gb/s
$T_{DP1}(sec)$	3.01
$E_{DP1}(joule)$	1.02E+06
Total CapEx	$5,395,360
CapEx/Job	$0.10
OpEx/Job	$0.02
C_{job}	$0.12

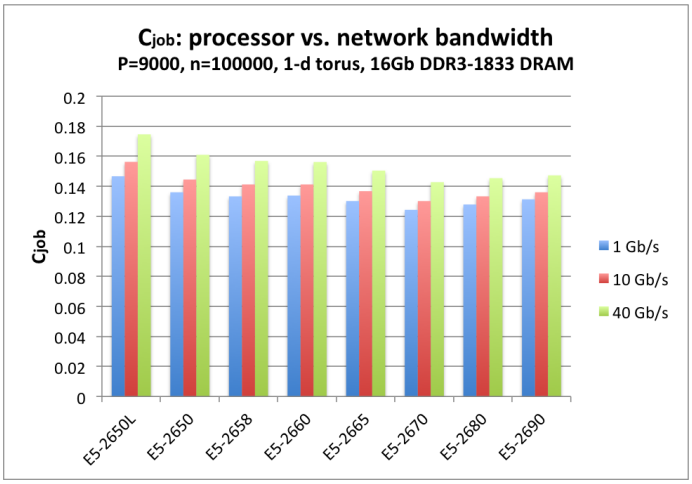
Table 7.7: Parameters for minimal C_{job}

Figure 7.2 illustrates the tradeoffs between processor type and either network bandwidth (Subfigure 7.2a) or torus dimension d (Subfigure 7.2b). In both cases, the optimal processor choice is an Intel E5-2670. As network cost is a function of torus dimension, note that each increase in d results in an increase in C_{job}. As a 40Gb/s ethernet torus is significantly more costly (in terms of $C_{financial}$) than either 1Gb/s or 10Gb/s tori, the increase in C_{job} is greater when moving from a 10Gb/s to 40Gb/s network than moving from 1Gb/s to 10Gb/s.

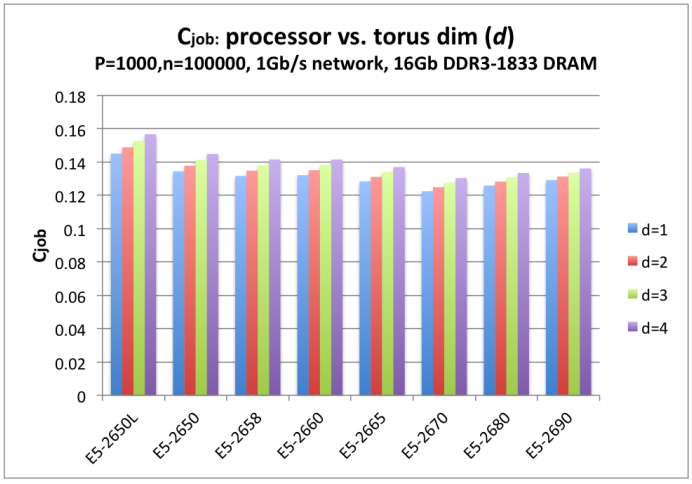
To conclude, the previous two sections described an approach to optimizing energy and runtime models for a distributed parallel machine. This optimization was considered within the Cityscape model of energy efficiency and the metric of financial cost/job (C_{job}). We believe that our approaches may aid hardware designers in targeting machines to benchmarks that approach optimal implementations of communication-optimal algorithms.

7.4 Further Directions

A potential criticism of the analyses in the previous sections is that the level of abstraction for the machine models is too high to properly obtain constraint information from designers and architects that is useful for extrapolation and integration with a hardware/software co-tuning process. In particular, the DP1 distributed parallel model encompasses much of the energy and runtime performance of an entire server node within the parameters γ_e and γ_t. This hides any performance interplay between caches and functional units (amongst other components).



(a) Processor vs. network bandwidth



(b) Processor vs. torus dimension

Figure 7.2: C_{job} with various parameter sets for Algorithm 11

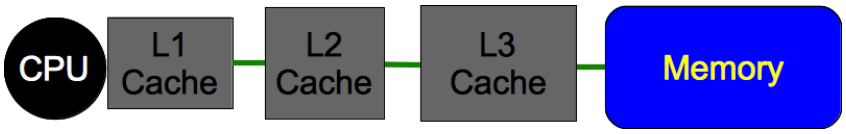


Figure 7.3: Sequential Machine with 3 levels of fast memory

While we were unable to address this observation quantitatively in this work, we suggest two approaches that may address this limitation. In the first, we expose more parameters by recursively expanding the machine model to encompass all levels of the memory hierarchy. However, this does not include the behavior of memories such as Translation Look-aside Buffers (TLBs), instruction caches and DRAM write buffers. In the second approach, we suggest a generalization of existing work that derives accurate linear power models from performance counter activity (which may include the behavior of additional memories, such as TLBs).

Expansion of Existing Models As in the derivation of the DP2 model, we can recursively define models of runtime and energy at successive levels of the memory hierarchy to expose lower-level parameters to the designer. For example, suppose we wish to analyze a sequential machine with three levels of fast memory: L1, L2 and L3 (see Figure 7.3). We assume that the L3 cache then fetches values from main memory, typically DRAM. If we attempt to use the sequential models of Equations (3.1) and (3.2) to represent this machine, the communication traffic from L1 to L2 and L2 to L3 are all abstracted into two terms: γ_t and γ_e. However, if we define these parameters in terms of sequential models of the next lower level of memory, a higher degree of fidelity can be obtained.

We derive the algebraic expressions for runtime and energy for this multilevel machine model in a similar manner to the DP2 distributed parallel model (Section 3.2). For illustrative purposes, we assume that the machine is able to perfectly overlap communication and computation at all levels of memory. We also assume that per-word and per-message costs differ by a small constant factor (e.g. messages are cache line transfers) and drop the α terms from the model for the sake of simplicity. Further, to differentiate parameters at the different levels of the machine, we refer to L3/Memory communication as "at level 3", L2/L3 communication as "at level 2", etc. Thus, the total runtime of the machine becomes:

$$T_3(M_1, M_2, M_3, F) = \max(\gamma_{t_3} F, \beta_{t_3} W_3) = \max\left(\frac{T_2}{F} F, \beta_{t_3} W_3\right)$$

as $\gamma_{t_3} = T_2/F$. If we continue this process and define T_2, replace γ_{t_2} with a definition of T_1, we obtain

$$T_3(M_1, M_2, M_3, F) = \max(\gamma_{t_1} F, \beta_{t_1} W_1, \beta_{t_2} W_2, \beta_{t_3} W_3)$$

as a runtime model for the machine in Figure 7.3. Via an identical process (without use of the $\max()$ function as energy consumption cannot be hidden), we obtain a model of energy for the 3-level sequential machine:

$$E_3(M_1, M_2, M_3, F) = ((\gamma_{e_1} F + \beta_{e_1} W_1 + (\delta_{e_1} + \epsilon_{e_1}) T_3) + \beta_{e_2} W_2 + (\delta_{e_2} + \epsilon_{e_2}) T_3) + \beta_{e_3} W_3 + (\delta_{e_3} + \epsilon_{e_3}) T_3.$$

Note that we assumed that energy is consumed by idle machine components for the entire runtime (i.e. $T_1 = T_2 = T_3$). In the case of dense $O(n^3)$ matrix-matrix multiplication, we note that

$$F = \Omega(n^3), W_3 = \Omega\left(\frac{F}{M_3^{1/2}}\right), W_2 = \Omega\left(\frac{F}{M_2^{1/2}}\right), W_1 = \Omega\left(\frac{F}{M_1^{1/2}}\right)$$

and we can see that on a typical machine of this model, a larger volume of data is transferred between faster levels of memory than slower as would be expected assuming the caches are actually able to take advantage of some memory locality in the application kernel.

As such, recursively-defined models of runtime and energy expose hardware parameters that are abstracted away in the DP1 and DP2 models. The challenge of modeling individual hardware components has been significantly addressed within the academic literature. For example, the CACTI tool [150] models cache access latency given parameters such as cache size and associativity. We also suspect that the relationship between cache size and energy consumption could also be added to the optimization constants. For example, [130] models the relationship between cache size and energy and uses such models to recommend cache design decisions that increase energy efficiency.

Integration with Low-level Performance Counter-based Models Another approach to reducing the abstraction of high-level models is to represent hardware parameters as compositions of performance counters. Performance counters have been effectively used to model hardware behavior in a large body of research (see [116] and [131] for surveys of power modeling techniques, including with performance counters) and have the benefit of directly correlating with events useful to hardware architects and designers. A generic representation of a performance counter energy model is as follows:

$$E = \sum_i w_i c_i$$

where the total power, E, is the summation of performance event counts c_i weighted by the constants w_i (which are in units of joules/event).

Parameters for such energy models (w_i) are typically regression fits of switching activity from gate-level simulation of a microbenchmark suite, RC delays, and power simulation via tools such as Synopsys PrimeTime ®[29]. Assuming the microbenchmark suite is able to accurately represent the target workloads (see Section 3.5 for a discussion of this challenge), the w_i can then be used to accurately simulate energy consumption for much faster register-transfer level (RTL) simulations (as with the PrEsto simulator [132]) or direct FPGA emulation [30].

In [30], Bhattacharjee, Contreras and Martonosi construct performance counter models to accurately represent power consumption of hardware components such as caches, memory management units and functional units. The total processor power thus becomes

$$H = \sum_i H_i = \sum_i \left(\sum_j w_{ij} c_{ij} \right)$$

where H is the sum of the component performance counter models, H_i. While this work specifically pertains to power models, a similar approach could be used for energy.

Recall the energy model for the sequential machine (Equation (3.2)):

$$E_S = \gamma_e F + \beta_e W + \alpha_e S + \delta_e \hat{M} T_S + \epsilon_e T_S.$$

This model can be linked to low-level counter events via two approaches:

- Direct representation of parameters via performance counter model weights (e.g. $\gamma_e = \sum_i w_i$)
- Representation of parameters as sets of component model weights, of type similar to [30].

If this second approach is used, the energy model now becomes:

$$E_S = \left(\sum_{ij} w_{0ij} \right) F + \left(\sum_{ij} w_{1ij} \right) W + \left(\sum_{ij} w_{2ij} \right) S + \left(\sum_{ij} w_{3ij} \right) \hat{M} T_S + \left(\sum_{ij} w_{4ij} \right) T_S$$

where each parameter γ_e,β_e,α_e,δ_e and ϵ_e is represented by summations of the fitted weights for A components, each of which is modeled with B terms. If performance counters are used for F,W, etc., and compared to modeled values, we can determine whether our implementation is close to attaining the lower bound, or perhaps whether hardware should be improved, e.g. perhaps with a better cache replacement policy.

It is currently unclear as to which approach would best be used to represent the parameters for the energy (and runtime, via a similar approach) models described in this work. Direct representation of parameters via performance counter model weights eliminates the step of partitioning the counters into component-specific sets, but may struggle to capture the behavior of instructions (e.g. memory accesses) that exhibit wide variation in energy consumption in certain cases (e.g. a cache miss or page fault). Perhaps nonlinear models could be used to model components with complex behavior.

One general limitation of using performance counter models for devices larger than the processor die is that performance events simply not be available, such as for network queues and router routing decisions. Even if the set of available counters is able to capture the behavior of the entire machine, access can be difficult due to lack of documentation from hardware manufacturers. Counters are not needed for proper execution of the device, and as such may not be validated to a high degree. These limitations suggest that performance counter energy models may be best suited to small processor or system on chip (SoC) designs.

We believe that either of the above two approaches, expanding models recursively or with performance counters, may be used to successfully lower the abstraction of the energy and runtime

models proposed in this thesis to a level that is easier for hardware experts to utilize. Both ideas must still address the challenge of effective microbenchmark creation. If these models are directly extended via recursive definition to more parameters, a research challenge still lies in interpreting parameters from a hardware perspective. For example, one may ask "How does knowing the fitted joule/word cost of L1 to L2 data allow me to improve my hardware design?" On the other hand, performance counter models are limited by the availability, accessibility and correctness of counters. Both techniques represent interesting areas of potential future research.

Chapter 8

Conclusions

This work focuses on deriving and analyzing bounds on the energy consumption of computational kernels on several types of abstract machine models. These kernels represent the building blocks of many high-performance computing (HPC) algorithms, and thus allow the results of this thesis to apply to a wide range of common algorithms when used in composition. This work builds on the existing body of research that bounds data communication, and our machine models are designed so as to allow integration with new communication bounds and hardware designs as they emerge. We also believe that such bounds may form the basis for a modeling infrastructure that would allow for hardware and software engineers to co-tune for the efficiency of energy and communication-optimal algorithms.

Specifically, we presented empirical evidence that energy consumption for building blocks of HPC applications, computational kernels, behave predictably during phases of almost constant arithmetic intensity (the flop/byte ratio of the implementation of an algorithm). That is, we observed that several kernels demonstrated nearly constant machine wall power during execution of a constant instruction mix. With this observation, we determined that the energy consumption and runtime of such application phases (which correspond to kernels) can be effectively modeled via linear expressions composed of a handful of terms. We validated these linear models for sequential and heterogeneous abstract machines, and argued (due to limitations in measurement equipment) for the applicability of such models to distributed parallel machines (Chapter 3). We also noted that the empirically extracted machine parameters often did not agree with expected values, especially for parameters in terms that do not dominate the computation (e.g. calculating per-flop parameters γ_t and γ_e from communication-dominated matrix-vector or naive matrix-matrix multiplication implementations). This suggests that specialized micro benchmarks (which may be automatically generated) are needed to properly calculate energy and runtime hardware parameters.

In Chapter 4 we combined these energy and runtime models with existing communication lower bounds (see [13] for details) to prove lower bounds on energy consumption and runtime for both the classical $O(n^3)$ and Strassen-like matrix-multiplication algorithms, dense and sparse matrix-vector multiplication and the $O(n^2)$ n-body problem. These bounds apply to sequential machines, and with the exception of matrix-vector multiplication, were also derived for two types of distributed parallel machines. We also proved communication and energy lower bounds for

dense matrix-matrix and matrix-vector multiplication on a heterogeneous model of computing, and communication and energy-optimal algorithms were described for both problems (Chapter 4). These energy lower bounds were then generalized to a larger class of programs that access arrays with affine expressions, and we generalized an existing optimal algorithm for the case where arrays are accessed via subsets of the iteration variables and showed that it attains a range of perfect strong scaling in runtime with constant energy consumption (Chapter 5).

With lower bounds on energy, we minimized energy or runtime in the presence of various constraints (e.g. upper bounds on runtime, energy or average power) while scaling memory usage and processors (Chapter 6) on a distributed parallel machine model. For example, we derived an explicit formula for the energy-optimal amount of utilized node memory for the $O(n^2)$ n-body problem and also considered the effect of scaling problem size and data replication on the energy efficiency of two example machines. In Chapter 7, we allowed hardware parameters in the model to vary and explored examples of optimization problems that utilize energy and runtime lower bounds to maximize energy efficiency. In particular, we presented a new way to visualize improvements in energy efficiency: the Cityscape model. This model generates constraint sets on how hardware parameters for a baseline machine need to scale to attain a desired energy efficiency improvement. In Chapter 7 we constructed a metric for the financial cost per job on distributed parallel machines, and described the characteristics of an optimal machine under this metric for a 3-body force calculation problem of a specific size.

This thesis suggests several major directions for future research, both in algorithms and computer hardware. Some key questions:

- What is the best method to measure energy consumption on distributed parallel machines? External monitoring devices, on-board firmware, or some combination of both? Ideally, such a method would balance measurement granularity with cost and ease of implementation.

- Do all computational motifs (i.e. classes of computational kernels) indeed exhibit predictable energy consumption? In this thesis, we demonstrated such behavior for several problems within two motifs: dense and sparse linear algebra. It is uncertain whether algorithms in other motifs exhibit this behavior, or even if all the algorithms within the dense and sparse linear algebra motifs are similarly predicable.

- How "tight" are these energy lower bounds? That is, when constants are carefully considered, how do the bounds compare to the energy consumption of optimal algorithms? Can the optimal algorithm for programs that access arrays via subsets of their iteration variables be used to attain performance and efficiency improvements on real machines?

- Can the energy optimal algorithm for heterogeneous matrix-matrix multiplication be generalized to the larger class of problems that access arrays with subsets of their iteration variables?

- What is the best way to combine our high-level models for energy and runtime with models that directly inform architects and device engineers? Can this be used in a co-tuning environment?

- How can the work of this thesis be extended to machine and algorithm features that dynamically vary energy and performance (e.g. varying node-level power limits to target higher performance states to intensive applications)? How does this fit in with work on adaptive runtime systems?

With computing technology evolving simultaneously toward miniaturization and large-scale systems, energy efficiency has become a key metric in determining the capability of devices and designs. This problem is not becoming easier as increasingly-heterogeneous devices begin to proliferate the computing landscape, and present problems even for basic benchmarking. To some extent, this problem may be addressed by automated tuning technology. However, with any metric, from floating point throughput to energy efficiency, the question "How well are we doing?" becomes vastly more interesting when considered in the light of "How well can we do?" This thesis addresses the latter question, and will hopefully serve to enlighten a portion of the former.

Bibliography

[1] *80 PLUS Verification and Testing Report: Dell Dh350E-S0*. Plug Load solutions. Sept. 2011. URL: http://www.plugloadsolutions.com/psu_reports/DELL_DH350E-SO_ECOS%202779_350W_Report.pdf.

[2] Dennis Abts, Michael R. Marty, Philip M. Wells, Peter Klausler, and Hong Liu. "Energy Proportional Datacenter Networks". In: *Proceedings of the 37th Annual International Symposium on Computer Architecture*. ISCA '10. Saint-Malo, France: ACM, 2010, pp. 338–347. ISBN: 978-1-4503-0053-7. DOI: 10.1145/1815961.1816004. URL: http://doi.acm.org/10.1145/1815961.1816004.

[3] Ramesh C Agarwal, Susanne M Balle, Fred G Gustavson, M Joshi, and P Palkar. "A three-dimensional approach to parallel matrix multiplication". In: *IBM Journal of Research and Development* 39.5 (1995), pp. 575–582.

[4] Alok Aggarwal, Ashok K Chandra, and Marc Snir. "Communication complexity of PRAMs". In: *Theoretical Computer Science* 71.1 (1990), pp. 3–28.

[5] *AMD Core Math Library (ACML)*. Advanced Micro Devices, Inc. 2013. URL: http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/05/acml.pdf.

[6] *AMD Math Libraries: OpenCL Basic Linear Algebra Subprograms Levels 1, 2, and 3*. Advanced Micro Devices, Inc. 2013. URL: http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-math-libraries.

[7] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Third Edition. Society for Industrial and Applied Mathematics, 1999. DOI: 10.1137/1.9780898719604.

[8] Alexandru Andrei, Marcus Schmitz, Petru Eles, Zebo Peng, and Bashir M Al Hashimi. "Simultaneous communication and processor voltage scaling for dynamic and leakage energy reduction in time-constrained systems". In: *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*. IEEE. 2004, pp. 362–369.

[9] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. *The Landscape of Parallel Computing Research: A View from Berkeley*. Tech. rep. UCB/EECS-2006-183. EECS Department, University of California, Berkeley, 2006. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html.

[10] *ATX Specification Version 2.01*. Intel Corporation. URL: http://www.formfactors.org/developer%5Cspecs%5Catx2_1.PDF.

[11] M. Bader, R. Franz, S. Günther, and A. Heinecke. "Hardware-oriented Implementation of Cache Oblivious Matrix Operations Based on Space-filling Curves". In: *Proceedings of the 7th International Conference on Parallel Processing and Applied Mathematics*. PPAM '07. Gdansk, Poland: Springer-Verlag, 2008, pp. 628–638. ISBN: 3-540-68105-1, 978-3-540-68105-2. URL: http://portal.acm.org/citation.cfm?id=1786194.1786267.

[12] David H Bailey, Eric Barszcz, John T Barton, David S Browning, Russell L Carter, Leonardo Dagum, Rod A Fatoohi, Paul O Frederickson, Thomas A Lasinski, Rob S Schreiber, et al. "The NAS parallel benchmarks". In: *International Journal of High Performance Computing Applications* 5.3 (1991), pp. 63–73.

[13] G Ballard, E Carson, J Demmel, M Hoemmen, N Knight, and O Schwartz. "Communication lower bounds and optimal algorithms for numerical linear algebra". In: *Acta Numerica* 23 (2014), pp. 1–155.

[14] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. "Communication-optimal Parallel and Sequential Cholesky Decomposition". In: *SIAM Journal on Scientific Computing* 32.6 (2010), pp. 3495–3523. DOI: 10.1137/090760969.

[15] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. "Minimizing Communication in Numerical Linear Algebra". In: *SIAM J. Matrix Analysis Applications* 32.3 (2011), pp. 866–901.

[16] Grey Ballard. "Avoiding Communication in Dense Linear Algebra". PhD thesis. EECS Department, University of California, Berkeley, 2013. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-151.html.

[17] Grey Ballard, James Demmel, and Andrew Gearhart. "Communication bounds for heterogeneous architectures". In: *23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2011)*. ("Brief Announcement"). 2011.

[18] Grey Ballard, Aydin Buluc, James Demmel, Laura Grigori, Benjamin Lipshitz, Oded Schwartz, and Sivan Toledo. "Communication optimal parallel multiplication of sparse random matrices". In: *Proceedings of the 25th ACM Symposium on Parallelism in Algorithms and Architectures*. ACM. 2013, pp. 222–231.

[19] Grey Ballard, James Demmel, Andrew Gearhart, Benjamin Lipshitz, Oded Schwartz, and Sivan Toledo. *Contention Bounds for Combinations of Computation Graphs and Network Topologies*. Tech. rep. UCB/EECS-2014-147. EECS Department, University of California, Berkeley, 2014. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-147.html.

[20] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. "Graph Expansion and Communication Costs of Fast Matrix Multiplication: Regular Submission". In: *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA '11. San Jose, California, USA: ACM, 2011, pp. 1–12. ISBN: 978-1-4503-0743-7. DOI: 10.1145/1989493.1989495. URL: http://doi.acm.org/10.1145/1989493.1989495.

[21] Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. "Strong Scaling of Matrix Multiplication Algorithms and Memory-Independent Communication Lower Bounds". In: *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA '12. ("Brief Announcement"). Pittsburgh, Pennsylvania, USA: ACM, 2012, pp. 77–79. ISBN: 978-1-4503-1213-4. DOI: 10.1145/2312005.2312021. URL: http://doi.acm.org/10.1145/2312005.2312021.

[22] J. Barnes and P. Hut. "A hierarchical O(N log N) force-calculation algorithm". In: *Nature* Vol.324 (Dec. 1986), pp. 446–449. DOI: 10.1038/324446a0.

[23] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. Philadelphia, PA: SIAM, 1994.

[24] Luiz André Barroso and Urs Hölzle. "The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines". In: *Synthesis Lectures on Computer Architecture* 4.1 (2009), pp. 1–108.

[25] K Basu, A Choudhary, Jayaprakash Pisharath, and M Kandemir. "Power protocol: reducing power dissipation on off-chip data buses". In: *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE Computer Society Press. 2002, pp. 345–355.

[26] D. Bedard, Min Yeol Lim, R. Fowler, and A. Porterfield. "PowerMon: Fine-Grained and Integrated Power Monitoring for Commodity Computer Systems". In: *IEEE SoutheastCon 2010 (SoutheastCon), Proceedings of the*. 2010, pp. 479–484. DOI: 10.1109/SECON.2010.5453824.

[27] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, Katherine Yelick, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Stephen Keckler, Dean Klein, Peter Kogge, R. Stanley Williams, and

Katherine Yelick. *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*. 2008.

[28] Ramon Bertran, Alper Buyuktosunoglu, Meeta S. Gupta, Marc Gonzalez, and Pradip Bose. "Systematic Energy Characterization of CMP/SMT Processor Systems via Automated Micro-Benchmarks". In: *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-45. Vancouver, B.C., CANADA: IEEE Computer Society, 2012, pp. 199–211. ISBN: 978-0-7695-4924-8. DOI: 10.1109/MICRO.2012.27. URL: http://dx.doi.org/10.1109/MICRO.2012.27.

[29] Himanshu Bhatnagar. *Advanced ASIC Chip Synthesis: Using Synopsys Design Compiler Physical Compiler and Prime Time*. 2nd. Norwell, MA, USA: Kluwer Academic Publishers, 2002. ISBN: 0792376447.

[30] A Bhattacharjee, G. Contreras, and M. Martonosi. "Full-system chip multiprocessor power evaluations using FPGA-based emulation". In: *Low Power Electronics and Design (ISLPED), 2008 ACM/IEEE International Symposium on*. 2008, pp. 335–340. DOI: 10.1145/1393921.1394010.

[31] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M.Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. "An Updated Set of Basic Linear Algebra Subroutines (BLAS)". In: *ACM Trans. Math. Soft.* 28.2 (2002).

[32] L Susan Blackford, Jaeyoung Choi, Andy Cleary, Eduardo D'Azevedo, James Demmel, Inderjit Dhillon, Jack Dongarra, Sven Hammarling, Greg Henry, Antoine Petitet, et al. *ScaLAPACK users' guide*. Vol. 4. SIAM, 1997.

[33] R. Blumofe, M. Frigo, C. Joerg, C. Leiserson, and K. Randall. "Dag-consistent Distributed Shared Memory". In: *IPPS '96: Proceedings of the 10th International Parallel Processing Symposium*. 1996, pp. 132–141.

[34] Robert D Blumofe, Matteo Frigo, Christopher F Joerg, Charles E Leiserson, and Keith H Randall. "Dag-consistent distributed shared memory". In: *Parallel Processing Symposium, 1996., Proceedings of IPPS'96, The 10th International*. IEEE. 1996, pp. 132–141.

[35] Béla Bollobás and Imre Leader. "Edge-isoperimetric inequalities in the grid". In: *Combinatorica* 11.4 (1991), pp. 299–314.

[36] Rasmus Bro and Sijmen De Jong. "A fast non-negativity-constrained least squares algorithm". In: *Journal of Chemometrics* 11.5 (1997), pp. 393–401. ISSN: 1099-128X. DOI: 10.1002/(SICI)1099-128X(199709/10)11:5<393::AID-CEM483>3.0.CO;2-L. URL: http://dx.doi.org/10.1002/(SICI)1099-128X(199709/10)11:5<393::AID-CEM483>3.0.CO;2-L.

[37] David J Brown and Charles Reams. "Toward energy-efficient computing". In: *Communications of the ACM* 53.3 (2010), pp. 50–58.

[38] L. E. Cannon. "A Cellular Computer to Implement the Kalman Filter Algorithm". PhD thesis. Bozeman, MT, USA, 1969.

[39] Enrique V. Carrera, Eduardo Pinheiro, and Ricardo Bianchini. "Conserving Disk Energy in Network Servers". In: *Proceedings of the 17th Annual International Conference on Supercomputing*. ICS '03. San Francisco, CA, USA: ACM, 2003, pp. 86–97. ISBN: 1-58113-733-8. DOI: 10.1145/782814.782829. URL: http://doi.acm.org/10.1145/782814.782829.

[40] Umit V Catalyurek and Cevdet Aykanat. "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication". In: *Parallel and Distributed Systems, IEEE Transactions on* 10.7 (1999), pp. 673–693.

[41] *CHECS Computing Resources*. http://www.checs.eng.vt.edu/resources.php. Accessed: 2014.08.03.

[42] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. "Span: An Energy-efficient Coordination Algorithm for Topology Maintenance in Ad hoc Wireless Networks". In: *Wireless Networks* 8.5 (2002), pp. 481–494.

[43] Jee Choi, M. Dukhan, Xing Liu, and R. Vuduc. "Algorithmic Time, Energy, and Power on Candidate HPC Compute Building Blocks". In: *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. 2014, pp. 447–457. DOI: 10.1109/IPDPS.2014.54.

[44] Jee Whan Choi, Daniel Bedard, Robert Fowler, and Richard Vuduc. "A Roofline Model of Energy". In: *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IPDPS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 661–672. ISBN: 978-0-7695-4971-2. DOI: 10.1109/IPDPS.2013.77. URL: http://dx.doi.org/10.1109/IPDPS.2013.77.

[45] Michael Christ, James Demmel, Nicholas Knight, Thomas Scanlon, and Katherine A. Yelick. *Communication Lower Bounds and Optimal Algorithms for Programs That Reference Arrays - Part 1*. Tech. rep. UCB/EECS-2013-61. EECS Department, University of California, Berkeley, 2013. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-61.html.

[46] D. Coppersmith and S. Winograd. "Matrix Multiplication via Arithmetic Progressions". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC '87. New York, New York, USA: ACM, 1987, pp. 1–6. ISBN: 0-89791-221-7. DOI: 10.1145/28395.28396. URL: http://doi.acm.org/10.1145/28395.28396.

[47] Intel Corporation. "Intel 64 and IA-32 Architectures Software Developer's Manual". In: (2011). URL: http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.html.

[48] *Counting Floating Point Operations on Intel Sandy Bridge and Ivy Bridge*. http://icl.cs.utk.edu/projects/papi/wiki/PAPITopics:SandyFlops. Accessed: 2014-09-02.

[49] Kent Czechowski, Casey Battaglino, McClanahan, Chris, Aparna Chandramowlishwaran, and Richard Vuduc. "Balance principles for algorithm-architecture co-design". In: *Proc. USENIX Wkshp. Hot Topics in Parallelism (HotPar), Berkeley, CA, USA* (2011).

[50] Timothy A. Davis and Yifan Hu. "The University of Florida Sparse Matrix Collection". In: *ACM Trans. Math. Softw.* 38.1 (Dec. 2011), 1:1–1:25. ISSN: 0098-3500. DOI: 10.1145/2049662.2049663. URL: http://doi.acm.org/10.1145/2049662.2049663.

[51] *DDR3 SDRAM Standard*. Joint Electron Device Engineering Council (JEDEC). URL: http://www.jedec.org/standards-documents/docs/jesd-79-3d.

[52] Eliezer Dekel, David Nassimi, and Sartaj Sahni. "Parallel matrix and graph algorithms". In: *SIAM Journal on computing* 10.4 (1981), pp. 657–675.

[53] J. Demmel, D. Eliahu, A. Fox, S. Kamil, B. Lipshitz, O. Schwartz, and O. Spillinger. "Communication-Optimal Parallel Recursive Rectangular Matrix Multiplication". In: *Parallel Distributed Processing Symposium (IPDPS), 2013 IEEE 27th International*. 2013.

[54] James Demmel. *Dense Linear Algebra, Part 1*. http://www.cs.berkeley.edu/~demmel/cs267_Spr14/Lectures/lecture12_densela_1_jwd14_v2_4pp.pdf. 2014.

[55] James Demmel, Andrew Gearhart, Benjamin Lipshitz, and Oded Schwartz. "Perfect Strong Scaling Using No Additional Energy". In: *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IPDPS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 649–660. ISBN: 978-0-7695-4971-2. DOI: 10.1109/IPDPS.2013.32. URL: http://dx.doi.org/10.1109/IPDPS.2013.32.

[56] James W. Demmel. *Applied Numerical Linear Algebra*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1997. ISBN: 0-89871-389-7.

[57] Robert H Dennard, Fritz H Gaensslen, V Rideout, E Bassous, and LeBlanc, A. "Design of Ion-implanted MOSFET's with very Small Physical Dimensions". In: *Solid-State Circuits, IEEE Journal of* 9.5 (1974), pp. 256–268.

[58] *DIMM Memory Power Calculator*. http://www.virtium.com/resources/design-support/power-calculator/. Accessed: 2014-09-14.

[59] Wilm E Donath. "Placement and average interconnection lengths of computer logic". In: *Circuits and Systems, IEEE Transactions on* 26.4 (1979), pp. 272–277.

[60] Wilm E Donath. "Wire length distribution for placements of computer logic". In: *IBM Journal of Research and Development* 25.3 (1981), pp. 152–155.

[61] Michael Driscoll, Evangelos Georganas, Penporn Koanantakool, Edgar Solomonik, and Katherine Yelick. "A communication-optimal N-body algorithm for direct interactions". In: *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE. 2013, pp. 1075–1084.

[62] *Electric Power Monthly:Table 5.3. Average Retail Price of Electricity to Ultimate Customers*. http://www.eia.gov/electricity/monthly/epm_table_grapher.cfm?t=epmt_5_3. Accessed: 2014-09-02.

[63] E. Elmroth, F. Gustavson, I. Jonsson, and B. Kågström. "Recursive Blocked Algorithms and Hybrid Data Structures for Dense Matrix Library Software". English. In: *SIAM Review* 46.1 (2004), pp. 3–45. ISSN: 00361445. URL: http://www.jstor.org/stable/20453467.

[64] Stephane Eranian. *The perfmon2 interface specification*. Tech. rep. 2005.

[65] *Estimate memory power based on DRAM IDD modes and utilization rates*. http://www.rampedia.com/index.php/AE2e. Accessed: 2014-09-14.

[66] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and K.W. Cameron. "PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications". In: *Parallel and Distributed Systems, IEEE Transactions on* 21.5 (2010), pp. 658–671. ISSN: 1045-9219. DOI: 10.1109/TPDS.2009.76.

[67] Robert A. van de Geijn and Jerrell Watts. *SUMMA: Scalable Universal Matrix Multiplication Algorithm*. Tech. rep. Austin, TX, USA, 1995.

[68] Lance Glasser and Daniel Dobberpuhl. *The Design and Analysis of VLSI Circuits*. Addison-Wesley, 1985.

[69] Ricardo Gonzalez, Benjamin M Gordon, and Mark A Horowitz. "Supply and threshold voltage scaling for low power CMOS". In: *Solid-State Circuits, IEEE Journal of* 32.8 (1997), pp. 1210–1216.

[70] Kazushige Goto and Robert A Geijn. "Anatomy of high-performance matrix multiplication". In: *ACM Transactions on Mathematical Software (TOMS)* 34.3 (2008), p. 12.

[71] Leslie Greengard and Vladimir Rokhlin. "A fast algorithm for particle simulations". In: *Journal of Computational Physics* 73.2 (1987), pp. 325–348.

[72] Domenik Helms, Eike Schmidt, and Wolfgang Nebel. "Leakage in CMOS circuits–an introduction". In: *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*. Springer, 2004, pp. 17–35.

[73] Bruce Hendrickson. "Graph partitioning and parallel solvers: Has the emperor no clothes?" In: *Solving Irregularly Structured Problems in Parallel*. Springer, 1998, pp. 218–225.

[74] Bruce Hendrickson and Tamara G Kolda. "Graph partitioning models for parallel computing". In: *Parallel Computing* 26.12 (2000), pp. 1519–1534.

[75] Ron Ho, Kenneth W Mai, and Mark A Horowitz. "The future of wires". In: *Proceedings of the IEEE* 89.4 (2001), pp. 490–504.

[76] Mark Frederick Hoemmen. "Communication-avoiding Krylov subspace methods". PhD thesis. EECS Department, University of California, Berkeley, 2010. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-37.html.

[77] *IEEE 802.3ł-2012 IEEE Standard for Ethernet*. Institute of Electrical and Electronics Engineers (IEEE).

[78] *Intel 64 and IA-32 Architecture Software Developer Manual*. Intel Corporation. 2014. URL: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf.

[79] *Intel Ethernet Converged Network Adapter X540-T2*. http://ark.intel.com/products/58954/Intel-Ethernet-Converged-Network-Adapter-X540-T2. Accessed: 2014-09-14.

[80] *Intel Xeon Processor 7500 Series Uncore Programming Guide*. Intel Corporation. 2010. URL: http://www.intel.com/assets/en_US/pdf/designguide/323535.pdf.

[81] *Intel Xeon Processor E5-2600 Product Family Uncore Performance Monitoring Guide*. Intel Corporation. 2012. URL: http://www.intel.com/content/dam/www/public/us/en/documents/design-guides/xeon-e5-2600-uncore-guide.pdf.

[82] *Intel Xeon Processor E5-2650*. http://ark.intel.com/products/64590/. Accessed: 2014-09-14.

[83] *Intel Xeon Processor E7 Family Uncore Performance Monitor Programming Guide*. Intel Corporation. 2011. URL: http://www.intel.com/content/dam/www/public/us/en/documents/performance-briefs/xeon-e7-family-uncore-performance-programming-guide.pdf.

[84] D. Irony, S. Toledo, and A. Tiskin. "Communication lower bounds for distributed-memory matrix multiplication". In: *J. Parallel Distrib. Comput.* 64.9 (Sept. 2004), pp. 1017–1026. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2004.03.021. URL: http://dx.doi.org/10.1016/j.jpdc.2004.03.021.

[85] Ankit Jain. "pOSKI: An Extensible Autotuning Framework to Perform Optimized SpMVs on Multicore Architectures". MA thesis. United States: University of California, Berkeley, 2008.

[86] H. Jia-Wei and H. T. Kung. "I/O complexity: The red-blue pebble game". In: *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*. STOC '81. Milwaukee, Wisconsin, United States: ACM, 1981, pp. 326–333. DOI: 10.1145/800076.802486. URL: http://doi.acm.org/10.1145/800076.802486.

[87] S Lennart Johnsson. "Minimizing the communication time for matrix multiplication on multiprocessors". In: *Parallel Computing* 19.11 (1993), pp. 1235–1257.

[88] Shoaib Kamil, John Shalf, and Erich Strohmaier. "Power efficiency in high performance computing". In: *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE. 2008, pp. 1–8.

[89] S. Kaxiras and M. Martonosi. *Computer Architecture Techniques for Power-Efficiency*. 1st. Morgan and Claypool Publishers, 2008. ISBN: 1598292080, 9781598292084.

[90] Ali Keshavarzi, Kaushik Roy, and Charles F. Hawkins. "Intrinsic Leakage in Low-Power Deep Submicron CMOS ICs". In: *Proceedings of the IEEE International Test Conference*. Washington, DC, USA: IEEE Computer Society, 1997, pp. 146–155. ISBN: 0-7803-4209-7. URL: http://dl.acm.org/citation.cfm?id=648019.745110.

[91] G. Kestor, R. Gioiosa, D.J. Kerbyson, and A. Hoisie. "Quantifying the energy cost of data movement in scientific applications". In: *Workload Characterization (IISWC), 2013 IEEE International Symposium on*. 2013, pp. 56–65. DOI: 10.1109/IISWC.2013.6704670.

[92] Jonathan Koomey, Kenneth Brill, Pitt Turner, John Stanley, and Bruce Taylor. *A simple model for determining true total cost of ownership for data centers*. Tech. rep. Uptime Institute, 2007.

[93] William Kramer. "How to measure useful, sustained performance". In: *State of the Practice Reports*. ACM. 2011, p. 2.

[94] Rensselaer Polytechnic Institute. Image Processing Laboratory and D.J.R. Meagher. *Octree Encoding: a New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*. 1980. URL: http://books.google.com/books?id=CgRPOAAACAAJ.

[95] B.S. Landman and Roy L. Russo. "On a Pin Versus Block Relationship For Partitions of Logic Graphs". In: *Computers, IEEE Transactions on* C-20.12 (1971), pp. 1469–1479. ISSN: 0018-9340. DOI: 10.1109/T-C.1971.223159.

[96] Charles L. Lawson and Richard J. Hanson. *Solving least squares problems*. Vol. 15. Classics in Applied Mathematics. Revised reprint of the 1974 original. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 1995, pp. xii+337. ISBN: 0-89871-356-0.

[97] François Le Gall. "Powers of Tensors and Fast Matrix Multiplication". In: *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. ISSAC '14. Kobe, Japan: ACM, 2014, pp. 296–303. ISBN: 978-1-4503-2501-1. DOI: 10.1145/2608628.2608664. URL: http://doi.acm.org/10.1145/2608628.2608664.

[98] Benjamin Lipshitz, Grey Ballard, James Demmel, and Oded Schwartz. "Communication-avoiding Parallel Strassen: Implementation and Performance". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '12. Salt Lake City, Utah: IEEE Computer Society Press, 2012, 101:1–101:11. ISBN: 978-1-4673-0804-5. URL: http://dl.acm.org/citation.cfm?id=2388996.2389133.

[99] Chun Liu, Anand Sivasubramaniam, and Mahmut Kandemir. "Optimizing bus energy consumption of on-chip multiprocessors using frequent values". In: *Journal of Systems Architecture* 52.2 (2006), pp. 129–142.

[100] Dake Liu and Christer Svensson. “Power consumption estimation in CMOS VLSI chips”. In: *Solid-State Circuits, IEEE Journal of* 29.6 (1994), pp. 663–670.

[101] Charles Lively, Xingfu Wu, Valerie Taylor, Shirley Moore, Hung-Ching Chang, and Kirk Cameron. “Energy and performance characteristics of different parallel implementations of scientific applications on multicore systems”. In: *International Journal of High Performance Computing Applications* 25.3 (2011), pp. 342–350.

[102] L. H. Loomis and H. Whitney. “An inequality related to the isoperimetric inequality”. In: *Bulletin of the American Mathematical Society* 55.10 (Oct. 1949), pp. 961–962. URL: http://projecteuclid.org/euclid.bams/1183514163.

[103] Hatem Ltaief, Piotr Luszczek, and Jack Dongarra. “Profiling high performance dense linear algebra algorithms on multicore architectures for power and energy efficiency”. In: *Computer Science-Research and Development* 27.4 (2012), pp. 277–287.

[104] Yuancheng Luo and Ramani Duraiswami. “Efficient parallel nonnegative least squares on multicore architectures”. In: *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2848–2863.

[105] Nir Magen, Avinoam Kolodny, Uri Weiser, and Nachum Shamir. “Interconnect-power dissipation in a microprocessor”. In: *Proceedings of the 2004 International Workshop on System Level Interconnect Prediction*. ACM. 2004, pp. 7–13.

[106] Priya Mahadevan, Puneet Sharma, Sujata Banerjee, and Parthasarathy Ranganathan. “A Power Benchmarking Framework for Network Devices”. English. In: *NETWORKING 2009*. Ed. by Luigi Fratta, Henning Schulzrinne, Yutaka Takahashi, and Otto Spaniol. Vol. 5550. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 795–808. ISBN: 978-3-642-01398-0. DOI: 10.1007/978-3-642-01399-7_62. URL: http://dx.doi.org/10.1007/978-3-642-01399-7_62.

[107] *Maple User Manual*. Maplesoft. 2011. URL: http://www.maplesoft.com/view.aspx?sl=5883.

[108] John D. McCalpin. “Memory Bandwidth and Machine Balance in Current High Performance Computers”. In: *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* (Dec. 1995), pp. 19–25.

[109] Robert M. Metcalfe and David R. Boggs. “Ethernet: Distributed Packet Switching for Local Computer Networks”. In: *Commun. ACM* 19.7 (July 1976), pp. 395–404. ISSN: 0001-0782. DOI: 10.1145/360248.360253. URL: http://doi.acm.org/10.1145/360248.360253.

[110] Marghoob Mohiyuddin, Mark Murphy, Leonid Oliker, John Shalf, John Wawrzynek, and Samuel Williams. “A design methodology for domain-optimized power-efficient supercomputing”. In: *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*. IEEE. 2009, pp. 1–12.

[111] Marghoob Mohiyuddin, Mark Hoemmen, James Demmel, and Katherine Yelick. "Minimizing Communication in Sparse Matrix Solvers". In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM. 2009, p. 36.

[112] Philip J. Mucci, Shirley Browne, Christine Deane, and George Ho. "PAPI: A Portable Interface to Hardware Performance Counters". In: *In Proceedings of the Department of Defense HPCMP Users Group Conference*. 1999, pp. 7–10.

[113] S.G. Narendra and A.P. Chandrakasan. *Leakage in Nanometer CMOS Technologies*. Integrated Circuits and Systems. Springer, 2010. ISBN: 9781441938268. URL: http://books.google.com/books?id=Ht93cgAACAAJ.

[114] Koichi Nose and Takayasu Sakurai. "Analysis and Future Trend of Short-circuit Power". In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 19.9 (2000), pp. 1023–1030.

[115] Chandrakant D Patel and Amip J Shah. *Cost model for planning, development and operation of a data center*. Tech. rep. HPL-2005-107R1. HP Labs, 2005.

[116] Sherief Reda and Abdullah N. Nowroz. "Power Modeling and Characterization of Computing Devices: A Survey". In: *Found. Trends Electron. Des. Autom.* 6.2 (Feb. 2012), pp. 121–216. ISSN: 1551-3939. DOI: 10.1561/1000000022. URL: http://dx.doi.org/10.1561/1000000022.

[117] Kaushik Roy, Saibal Mukhopadhyay, and Hamid Mahmoodi-Meimand. "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits". In: *Proceedings of the IEEE* 91.2 (2003), pp. 305–327.

[118] M.A.A. Sanvido, F.R. Chu, A. Kulkarni, and R. Selinger. "NAND Flash Memory and Its Role in Storage Architectures". In: *Proceedings of the IEEE* 96.11 (2008), pp. 1864–1874. ISSN: 0018-9219. DOI: 10.1109/JPROC.2008.2004319.

[119] Patrick R. Schaumont. *A Practical Introduction to Hardware/Software Codesign*. 1st. Springer Publishing Company, Incorporated, 2010. ISBN: 1441959998, 9781441959997.

[120] A. Schönhage. "Partial and Total Matrix Multiplication". In: *SIAM J. Computing* 10.3 (1981), pp. 434–455. DOI: 10.1137/0210032. URL: http://link.aip.org/link/?SMJ/10/434/1.

[121] John Shalf, Sudip Dosanjh, and John Morrison. "Exascale computing technology challenges". In: *High Performance Computing for Computational Science–VECPAR 2010*. Springer, 2011, pp. 1–25.

[122] Jonathan R Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Tech. rep. Pittsburgh, PA, USA, 1994.

[123] Harsha Vardhan Simhadri, Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, and Aapo Kyrola. "Experimental Analysis of Space-bounded Schedulers". In: *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA '14. Prague, Czech Republic: ACM, 2014, pp. 30–41. ISBN: 978-1-4503-2821-0. DOI: 10.1145/2612669.2612678. URL: http://doi.acm.org/10.1145/2612669.2612678.

[124] Ripduman Sohan Sohan, Andrew Rice Rice, W. Moore Andrew Andrew, and Kieran Mansley Mansley. "Characterizing 10 Gbps Network Interface Energy Consumption". In: *Proceedings of the 2010 IEEE 35th Conference on Local Computer Networks*. LCN '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 268–271. ISBN: 978-1-4244-8387-7. DOI: 10.1109/LCN.2010.5735719. URL: http://dx.doi.org/10.1109/LCN.2010.5735719.

[125] E. Solomonik, A. Bhatele, and J. Demmel. "Improving communication performance in dense linear algebra via topology aware collectives". In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '11. Seattle, Washington: ACM, 2011, 77:1–77:11. ISBN: 978-1-4503-0771-0. DOI: 10.1145/2063384.2063487. URL: http://doi.acm.org/10.1145/2063384.2063487.

[126] Edgar Solomonik and James Demmel. "Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms". In: *Proceedings of the 17th international conference on Parallel processing - Volume Part II*. Euro-Par'11. Bordeaux, France: Springer-Verlag, 2011, pp. 90–109. ISBN: 978-3-642-23396-8. URL: http://dl.acm.org/citation.cfm?id=2033408.2033420.

[127] Edgar Solomonik, Devin Matthews, Jeff Hammond, and James Demmel. "Cyclops Tensor Framework: Reducing Communication and Eliminating Load Imbalance in Massively Parallel Contractions". In: *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IPDPS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 813–824. ISBN: 978-0-7695-4971-2. DOI: 10.1109/IPDPS.2013.112. URL: http://dx.doi.org/10.1109/IPDPS.2013.112.

[128] V. Strassen. "Relative bilinear complexity and matrix multiplication". In: *Journal fűr die reine und angewandte Mathematik (Crelles Journal)* 1987.375–376 (1987), pp. 406–443.

[129] Volker Strassen. "Gaussian elimination is not optimal". English. In: *Numerische Mathematik* 13.4 (1969), pp. 354–356. ISSN: 0029-599X. DOI: 10.1007/BF02165411. URL: http://dx.doi.org/10.1007/BF02165411.

[130] Ching-Long Su and Alvin M. Despain. "Cache Design Trade-offs for Power and Performance Optimization: A Case Study". In: *Proceedings of the 1995 International Symposium on Low Power Design*. ISLPED '95. Dana Point, California, USA: ACM, 1995, pp. 63–68. ISBN: 0-89791-744-8. DOI: 10.1145/224081.224093. URL: http://doi.acm.org/10.1145/224081.224093.

[131] Hameedah Sultan, Gayathri Ananthanarayanan, and Smruti R. Sarangi. "Processor Power Estimation Techniques: A Survey". In: *Int. J. High Perform. Syst. Archit.* 5.2 (May 2014), pp. 93–114. ISSN: 1751-6528. DOI: 10.1504/IJHPSA.2014.061448. URL: http://dx.doi.org/10.1504/IJHPSA.2014.061448.

[132] Dam Sunwoo, Gene Y Wu, Nikhil A Patil, and Derek Chiou. "PrEsto: An FPGA-accelerated power estimation methodology for complex systems". In: *Field Programmable Logic and Applications (FPL), 2010 International Conference on*. IEEE. 2010, pp. 310–317.

[133] Dinesh C Suresh, Banit Agrawal, Jun Yang, and Walid Najjar. "Energy-efficient encoding techniques for off-chip data buses". In: *ACM Transactions on Embedded Computing Systems (TECS)* 8.2 (2009), p. 9.

[134] Dinesh C Suresh, Banit Agrawal, Jun Yang, Walid Najjar, and Laxmi Bhuyan. "Power efficient encoding techniques for off-chip data buses". In: *Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems*. ACM. 2003, pp. 267–275.

[135] Dennis Sylvester and Kurt Keutzer. "Getting to the bottom of deep submicron". In: *Proceedings of the 1998 IEEE/ACM International Conference on Computer-Aided Design*. ACM. 1998, pp. 203–211.

[136] Jürgen Teich. "Hardware/Software Codesign: The Past, the Present, and Predicting the Future". In: *Proceedings of the IEEE Special Centennial Issue* 100 (2012), pp. 1411–1430.

[137] Jan Treibig, Georg Hager, and Gerhard Wellein. "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments". In: *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*. IEEE. 2010, pp. 207–216.

[138] *User's Guide for Intel(R) Math Kernel Library 11.1 Update 3 for Linux* OS*. Intel Corporation. 2014. URL: http://software.intel.com/sites/products/documentation/doclib/mkl_sa/111/11.1.3/mklman.pdf.

[139] Leslie G. Valiant. "A Bridging Model for Parallel Computation". In: *Commun. ACM* 33.8 (Aug. 1990), pp. 103–111. ISSN: 0001-0782. DOI: 10.1145/79173.79181. URL: http://doi.acm.org/10.1145/79173.79181.

[140] Mark H. Van Benthem and Michael R. Keenan. "Fast algorithm for the solution of large-scale non-negativity-constrained least squares problems". In: *Journal of Chemometrics* 18.10 (2004), pp. 441–450. ISSN: 1099-128X. DOI: 10.1002/cem.889. URL: http://dx.doi.org/10.1002/cem.889.

[141] Harry JM Veendrick. "Short-circuit Dissipation of Static CMOS Circuitry and its Impact on the Design of Buffer Circuits". In: *Solid-State Circuits, IEEE Journal of* 19.4 (1984), pp. 468–473.

[142] Richard Vuduc, James W Demmel, and Katherine A Yelick. "OSKI: A library of automatically tuned sparse matrix kernels". In: *Journal of Physics: Conference Series*. Vol. 16. 1. IOP Publishing. 2005, p. 521.

[143] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanovic. *The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA*. Tech. rep. UCB/EECS-2011-62. EECS Department, University of California, Berkeley, 2011. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-62.html.

[144] *Watts Up? Plug Load Meters*. https://www.wattsupmeters.com/secure/products.php?pn=0. Accessed: 2014-09-02.

[145] Vincent M Weaver, Matt Johnson, Kiran Kasichayanula, James Ralph, Piotr Luszczek, Daniel Terpstra, and Shirley Moore. "Measuring energy and power with PAPI". In: *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*. IEEE. 2012, pp. 262–268.

[146] R. Clint Whaley and Jack Dongarra. "Automatically Tuned Linear Algebra Software". In: *SuperComputing 1998: High Performance Networking and Computing*. 1998.

[147] Thomas Willhalm. *Intel Performance Counter Monitor - A better way to measure CPU utilization*. 2012. URL: http://software.intel.com/en-us/articles/intel-performance-counter-monitor-a-better-way-to-measure-cpu-utilization.

[148] Samuel Williams, Andrew Waterman, and David Patterson. "Roofline: An Insightful Visual Performance Model for Multicore Architectures". In: *Commun. ACM* 52.4 (Apr. 2009), pp. 65–76. ISSN: 0001-0782. DOI: 10.1145/1498765.1498785. URL: http://doi.acm.org/10.1145/1498765.1498785.

[149] Virginia Vassilevska Williams. "Multiplying matrices faster than Coppersmith-Winograd". In: *Proceedings of the 44th Symposium on Theory of Computing*. Proc. 45th STOC. New York, New York, USA: ACM, 2012, pp. 887–898. ISBN: 978-1-4503-1245-5. DOI: 10.1145/2213977.2214056. URL: http://doi.acm.org/10.1145/2213977.2214056.

[150] S. J E Wilton and N.P. Jouppi. "CACTI: An Enhanced Cache Access and Cycle Time Model". In: *Solid-State Circuits, IEEE Journal of* 31.5 (1996), pp. 677–688. ISSN: 0018-9200. DOI: 10.1109/4.509850.

[151] D. Wise. "Ahnentafel Indexing into Morton-Ordered Arrays, or Matrix Locality for Free". In: *Euro-Par 2000 Parallel Processing*. Ed. by Arndt Bode, Thomas Ludwig, Wolfgang Karl, and Roland Wismller. Vol. 1900. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2000, pp. 774–783.

[152] Jinchao Xu. "An introduction to multilevel methods". In: *Wavelets, multilevel methods and elliptic PDEs (Leicester, 1996)*. Numer. Math. Sci. Comput. Oxford Univ. Press, New York, 1997, pp. 213–302.

[153] Ya Xu, John Heidemann, and Deborah Estrin. "Geography-informed energy conservation for ad hoc routing". In: *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*. ACM. 2001, pp. 70–84.

[154] Ossama Younis and Sonia Fahmy. “HEED: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks”. In: *Mobile Computing, IEEE Transactions on* 3.4 (2004), pp. 366–379.