How to perform HPL on CPU&GPU clusters

Dr.sc. Draško Tomić
email: drasko.tomic@hp.com

Forecasting is not so easy, HPL benchmarking could be even more difficult

Agenda

TOP500 GPU trends

Some basics about HPL

Performance modeling of HPL on CPUGPU clusters

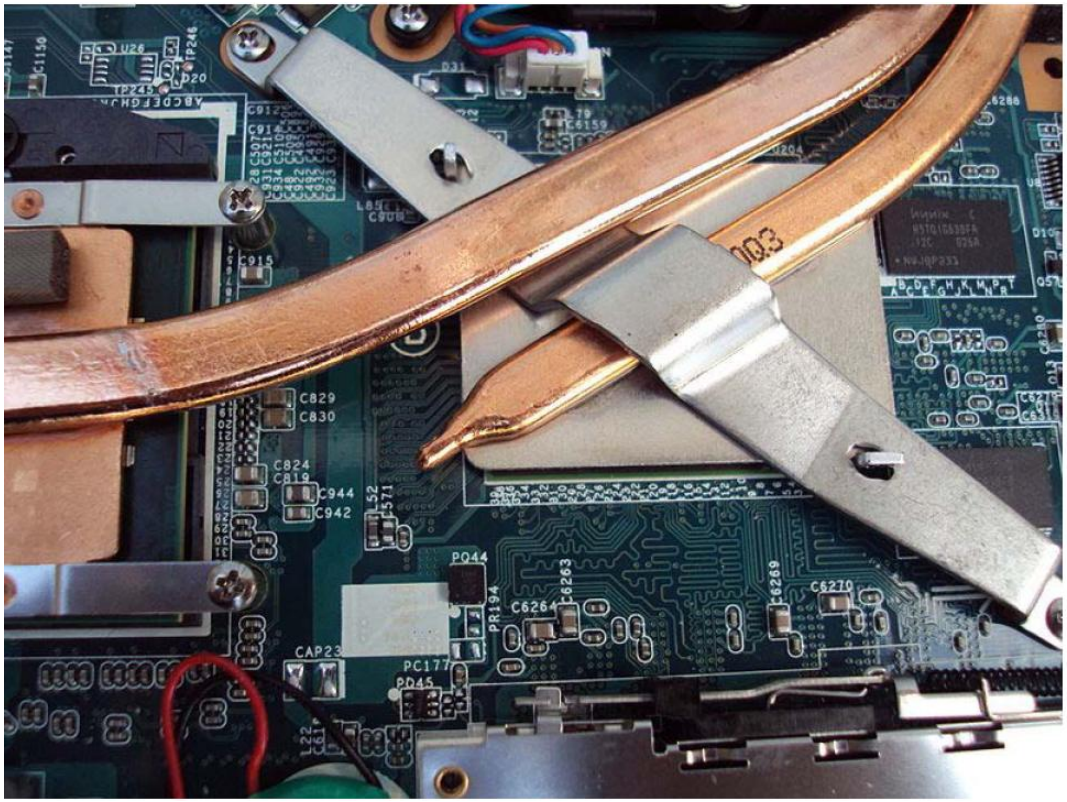
How to perform efficient HPL on CPUGPU clusters

Examples of current HPL-CPUGPU implementations

Cloud is challenging us

With the advent of GPU cards, HPC landscape has changed dramatically:

- More FLOPS/watt available
- less space for HPC needed
- Supercomputing on every workstation possible



TOP500 List – November 2009

Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron Six Core 2.6 GHz / 2009 Cray Inc.	224162	1759.00	2331.00	6950.60
2	DOE/NNSA/LANL United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband / 2009 IBM	122400	1042.00	1375.78	2345.50
3	National Institute for Computational Sciences/University of Tennessee United States	Kraken XT5 - Cray XT5-HE Opteron Six Core 2.6 GHz / 2009 Cray Inc.	98928	831.70	1028.85	
4	Forschungszentrum Juelich (FZJ) Germany	JUGENE - Blue Gene/P Solution / 2009 IBM	294912	825.50	1002.70	2268.00
5	National SuperComputer Center in Tianjin/NUDT China	Tianhe-1 - NUDT TH-1 Cluster, Xeon E5540/E5450, ATI Radeon HD 4870 2, Infiniband / 2009 NUDT	71680	563.10	1206.19	
6	NASA/Ames Research Center/NAS United States	Pleiades - SGI Altix ICE 8200EX, Xeon QC 3.0 GHz/Nehalem EP 2.93 Ghz / 2009 SGI	56320	544.30	673.26	2348.00
7	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution / 2007 IBM	212992	478.20	596.38	2329.60
8	Argonne National Laboratory United States	Blue Gene/P Solution / 2007 IBM	163840	458.61	557.06	1260.00
9	Texas Advanced Computing Center/Univ. of Texas United States	Ranger - SunBlade x6420, Opteron QC 2.3 Ghz, Infiniband / 2008 Sun Microsystems	62976	433.20	579.38	2000.00
10	Sandia National Laboratories / National Renewable Energy Laboratory United States	Red Sky - Sun Blade x6275, Xeon X55xx 2.93 Ghz, Infiniband / 2009 Sun Microsystems	41616	423.90	487.74	

Recent Releases

- November 2011
- June 2011
- November 2010
- June 2010
- November 2009

Inside HPC

- **Video: Live Keynote from GTC Asia, Dec. 13, 9:30am PT**
- **Sponsored Post: Grid Engine Quiz — Win an iPad2**

TOP500 List – November 2010

Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	National Supercomputing Center in Tianjin China	Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C / 2010 NUDT	186368	2566.00	4701.00	4040.00
2	DOE/SC/Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	224162	1759.00	2331.00	6950.60
3	National Supercomputing Centre in Shenzhen (NSCS) China	Nebulae - Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU / 2010 Dawning	120640	1271.00	2984.30	2580.00
4	GSIC Center, Tokyo Institute of Technology Japan	TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP	73278	1192.00	2287.63	1398.61
5	DOE/SC/LBNL/NERSC United States	Hopper - Cray XE6 12-core 2.1 GHz / 2010 Cray Inc.	153408	1054.00	1288.63	2910.00
6	Commissariat a l'Energie Atomique (CEA) France	Tera-100 - Bull bullx super-node S6010/S6030 / 2010 Bull SA	138368	1050.00	1254.55	4590.00
7	DOE/NNSA/LANL United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband / 2009 IBM	122400	1042.00	1375.78	2345.50
8	National Institute for Computational Sciences/University of Tennessee United States	Kraken XT5 - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	98928	831.70	1028.85	3090.00
9	Forschungszentrum Juelich (FZJ) Germany	JUGENE - Blue Gene/P Solution / 2009 IBM	294912	825.50	1002.70	2268.00
10	DOE/NNSA/LANL/SNL United States	Cielo - Cray XE6 8-core 2.4 GHz / 2010 Cray Inc.	107152	816.60	1028.66	2950.00

Recent Releases

- **November 2011**
- **June 2011**
- **November 2010**
- **June 2010**
- **November 2009**

Inside HPC

Video: Live Keynote from GTC Asia, Dec. 13, 9:30am PT

Why GPU on HPC clusters?

Better performance/cost ratio

Faster communication (within CPU/GPU nodes)

Smaller footprint (dense computing)

Why NOT GPU on HPC clusters?

Not a single programming models (MPI & MP & Streams)

Deployment of advanced cooling systems (e.g. water cooling)

Many applications still not supporting GPU accelerators

Not so efficient Linpack benchmarking (like on CPU only clusters)

Trends

- A trend is developing in high-performance computing in which general-purpose processors are coupled to GPUs used as accelerators.
- Such systems are known as heterogeneous (or hybrid) CPUGPU systems.

Some definitions from linear algebra

- A general band matrix has its nonzero elements arranged uniformly near the diagonal, such that: a_{ij} equal to 0 if (i-j) greater than ml or (j-i) greater than mu; ml and mu are the lower and upper band widths, respectively, and ml+mu+1 is the total band width.

- The matrix $\boldsymbol{A}$ is symmetric if it has the property $\boldsymbol{A}$ equal to $\boldsymbol{A}^{\mathsf{T}}$

- A real symmetric matrix $\boldsymbol{A}$ is indefinite if and only if $(\boldsymbol{x}^{\mathsf{T}}\boldsymbol{A}\boldsymbol{x})\ (\boldsymbol{A}\ \boldsymbol{y}^{\mathsf{T}}\boldsymbol{A}\boldsymbol{y}) < \boldsymbol{0}$ for some non-zero vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.

- A real symmetric matrix $\boldsymbol{A}$ is positive definite if and only if $\boldsymbol{x}^{\mathsf{T}}\boldsymbol{A}\boldsymbol{x}$ is positive for all nonzero vectors $\boldsymbol{x}$.

- A matrix $\boldsymbol{U}$ is an upper triangular matrix if its nonzero elements are found only in the upper triangle of the matrix, including the main diagonal; A matrix $\boldsymbol{L}$ is an lower triangular matrix if its nonzero elements are found only in the lower triangle of the matrix, including the main diagonal.

- A general tridiagonal matrix is a matrix whose nonzero elements are found only on the diagonal, subdiagonal, and superdiagonal of the matrix

Some definitions from linear algebra (cont.)

- In linear algebra, **Gaussian elimination** is an algorithm for solving systems of linear equations.
- It can also be used to find the rank of a matrix, to calculate the determinant of a matrix, and to calculate the inverse of an invertible square matrix.
- The process of Gaussian elimination has two parts. The first part (Forward Elimination) reduces a given system to either triangular or echelon form, or results in a degenerate equation with no solution, indicating the system has no solution. The second step uses back substitution to find the solution of the system above.

$$\begin{aligned} 2x + y - z &= 8 \\ -3x - y + 2z &= -11 \\ -2x + y + 2z &= -3 \end{aligned}$$

→

$$\begin{aligned} 2x + y - z &= 8 \\ \tfrac{1}{2}y + \tfrac{1}{2}z &= 1 \\ 2y + z &= 5 \end{aligned}$$

→

$$\begin{aligned} 2x + y - z &= 8 \\ \tfrac{1}{2}y + \tfrac{1}{2}z &= 1 \\ -z &= 1 \end{aligned}$$

Some definitions from linear algebra (cont.)

- The **pivot** or **pivot element** is the element of a matrix, an array, or some other kind of finite set, which is selected first by an algorithm (e.g. Gaussian elimination, Quicksort, Simplex algorithm etc.), to do certain calculations.
- In the case of Gaussian elimination, the algorithm requires that pivot elements not be zero. Interchanging rows or columns in the case of a zero pivot element is necessary.
- Example: the system below requires the interchange of rows 2 and 3 to perform elimination:

$$\left[\begin{array}{ccc|c} 1 & -1 & 2 & 8 \\ 0 & 0 & -1 & -11 \\ 0 & 2 & -1 & -3 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 1 & -1 & 2 & 8 \\ 0 & 2 & -1 & -3 \\ 0 & 0 & -1 & -11 \end{array}\right]$$

Furthermore, in Gaussian elimination it is generally desirable to choose a pivot element with

large absolute value.

This improves the numerical stability.

What is LINPACK?

- **LINPACK** is a software library for performing numerical linear algebra on digital computers. It was written in Fortran by Jack Dongarra, Jim Bunch, Cleve Moler and Gilbert Stewart, and was intended for use on supercomputers in the 1970s and early 1980s.

- The package solves linear systems whose matrices are general, banded, symmetric indefinite, symmetric positive definite, triangular, and tridiagonal square.

- In addition, the package computes the QR decomposition ($A \rightarrow QR$; $Q^T = Q^{-1}$) and singular value decompositions of rectangular matrices and applies them to least-squares problems.

- LINPACK uses column-oriented algorithms to increase efficiency by preserving locality of reference.

What is HPL?

- A measure of a system's floating point computing power.

- Introduced by Jack Dongarra, it measure how fast a computer **solves a dense *N* by *N* *system of linear equations* *Ax* = *b***, which is a common task in engineering.

- The solution is obtained by **Gaussian elimination with partial pivoting (**only row permutations permited, strategy is to switch the largest entry in the pivot column to the diagonal) with $2/3 \cdot N^3 + 2 \cdot N^2 + O(N)$ floating point operations.

- The result is reported in millions of floating point operations per second (MFLOPS).

- For large-scale distributed-memory systems, High Performance Linpack (HPL), a portable implementation of the High-Performance LINPACK Benchmark, is used as a performance measure for ranking supercomputers in the TOP500 list of the world's fastest computers.

  HPL reguires MPI and BLAS.

- The HPC benchmark is run for different matrix sizes N searching for the size N_{max} for which the maximal performance R_{max} is obtained.

- The benchmark also reports the problem size $N_{1/2}$ where half of the performance ($R_{max}/2$) is achieved.

Performance Modeling of HPL on CPUGPU clusters

Still not easy to run **efficiently** HPL on large HPC clusters with CPUGPU nodes.

In order to predict HPL performance, we need some benchmarking results from smaller clusters.

And then a right approach to scale Linpack on much larger clusters, even those that will take place on www.top500.org in forthcoming years.

Example: Performance modeling of HPL on Nvidia CPUGPU

Some facts:

Tesla NVIDIA 2090: 655 Gflops (double precision)

SL390 (2 x 5670 CPUs) Rmax = 125 Rpeak = 147

Linpack benchmarking:

Node with Intel Hex Core X5670 (dual socket) and Tesla S2050 (node sees 2 GPUs)

- Node has 48GB of RAM.
- RedHat Enterprise Linux 5.4 64-bit
- Intel compiler version 11.1.069
- Intel MKL version 10.2.5
- Openmpi version 1.4.1
- Cudatoolkit 3.1
- NVIDIA driver supporting CUDA 3.1 (NVIDIA-Linux-x86_64-256.53.run)
- Modified version of HPL from NVIDIA (hpl-2.0_FERMI_v09.tgz)

Rpeak (theoretical) = 140 GFLOPS (12 CPU cores) + 1030 GFLOPS (2 GPUs) = 1170 GFLOPS

Rmax (actual) = 727.3 GFLOPS (62% efficiency)

Node with 2 x Intel Hex Core X5670 (dual socket) and Tesla S2090 (node sees 2 GPUs)

Rpeak (theoretical) = 280 GFLOPS (24 CPU cores) + 1310 GFLOPS (2 GPUs) = 1590 GFLOPS

Steps to run HPL on NVIDIA GPUs

Install NVIDIA driver

Load the driver

Test it is working

Install CUDA toolkit

Install OpenMPI

Set your environment variables to point to Intel compilers

Compile OpenMPI

Install HPL from NVIDIA

Compile HPL

Launch HPL processes

Prepare node files

Launch Tesla GPU version of HPL

Cluster Performance

CASPUR Jazz Cluster			
Node	16x HP DL360 G7	**Interconnect**	Infiniband QDR
CPU	2x X5650 @ 2.67 GHz (6 cores)	**Switch**	HP (Voltaire) 36P Managed Switch
RAM	48 GB	**HCA**	HP InfiniBand 4X QDR ConnectX-2 PCIe G2 Dual Port
Peak Perf.	1152 GFLOPS (DP)	**GPU**	NVIDIA S2050 (half x node)
xHPL Perf.	892 GFLOPS (DP)		(2x GPUs C2050 @ 1.15 GHz & 3 GB DDR5 RAM)
Notes	MKL_NUM_THREADS = 6 /CPU		OMPL_NUM_THREADS = 6 /CPU
	Infiniband QDR as OpenIB no RDMA		GFLOPS: best results per # Procs
	mpirun -mca tbl openib,self -mca btl_openib_flags 1 -bysocket -bind-to-socket -hostfile ./myhostfile -np #Procs ./run_linpack		

						GFLOPS			
#Nodes	**#Procs**	**PxQ**	**N**	**MemTot(MB)**	**MemNode(MB)**	**Nb=768**	**x Node**	**% xHPL**	**% peak**
1	1	1x1	70000	41870	41870	410	410	80.6%	64.4%
1	2	1x2	70000	41870	41870	640	640	71.7%	55.6%
2	4	2x2	105000	94208	47104	1319	660	73.9%	57.2%
4	8	2x4	150000	192261	48065	2551	638	71.5%	55.4%
6	12	3x4	180000	276855	46143	3814	636	71.3%	55.2%
8	16	4x4	210000	376831	47104	5092	637	71.4%	55.3%
10	20	4x5	235000	471893	47189	6295	630	70.6%	54.6%
12	24	4x6	256000	560000	46667	7498	625	70.0%	54.2%
14	28	4x7	280000	669922	47852	8771	627	70.2%	54.4%
15	30	5x6	290000	718628	47909	9147	610	68.4%	52.9%
16	32	4x8	300000	769043	48065	10050	628	70.4%	54.5%

How to HPL-GPU?

How to make efficient HPL on large CPUGPU clusters?

We already know how to enable Linpack even on largest heterogeneous HPC clusters

We have good Linpack result (53% efficiency) even on largest heterogeneous HPC clusters, e.g. Tsubame: http://matsu-www.is.titech.ac.jp/~endo/papers/endo-ipdps10.pdf

We expect NextGen Intel processor will be at least >25% better in doing floats over Westmere (more cores, faster bus…)

MPI fundamentals

Message Passing Interface (**MPI**) is a standardized and portable message-passing system designed by a group of researchers from academia and industry to function on a wide variety of parallel computers.

The standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message-passing programs in Fortran 77 or the C programming language.

Several well-tested and efficient implementations of MPI include some that are free and in the public domain.

Both point-to-point and collective communication are supported.

MPI is a message passing appplication programming interface.

MP fundamentals

Multiprocessing is the use of two or more central processing units (CPUs) within a single computer system.

The term also refers to the ability of a system to support more than one processor and/or the ability to allocate tasks between them.

There are many variations on this basic theme, and the definition of multiprocessing can vary with context, mostly as a function of how CPUs are defined (multiple cores on one die, multiple dies in one package, multiple packages in one system unit, etc.).

Stream processing

Stream processing is a computer programming paradigm, related to SIMD (single instruction, multiple data), that allows some applications to more easily exploit a limited form of parallel processing.

Such applications can use multiple computational units, such as the FPUs on a GPU or field programmable gate arrays (FPGAs), without explicitly managing allocation, synchronization, or communication among those units.

The stream processing paradigm simplifies parallel software and hardware by restricting the parallel computation that can be performed.

Given a set of data (a *stream*), a series of operations (*kernel functions*) are applied to each element in the stream.

Uniform streaming, where one kernel function is applied to all elements in the stream, is typical.

Kernel functions are usually pipelined, and local on-chip memory is reused to minimize external memory bandwidth. Since the kernel and stream abstractions expose data dependencies, compile tools can fully automate and optimize on-chip management tasks.

Hybrid (MPI, MP, Streaming) HPL programming model

Standard open-source HPL is designed for homogeneous clusters.

The solution is obtained by performing LU factorization of the dense matrix *A with partial pivoting, and then solves the re*sulting triangular system of equations.

The workload of the Linpack benchmark is $(2/3)N^3 + 2N^2 + O(N)$.

The LU factorization takes almost all the computation time of the benchmark. The computation time of the LU factorization is dominated by the matrix update and the upper (U) matrix factor.

The matrix update is a form of the matrix-matrix multiply (DGEMM (Double Precision General Matrix Multiply)) which is an $O(N^3)$ *operation. The latter uses a triangular* solver with multiple right-hand-sides (DTRSM) kernel which is an $O(N^2)$ *operation.*

To make use of the computing capacity of the whole system, it is neccessarry to fully exploit the parallel task, thread and data possibilities of the HPL.

Hybrid (MPI, MP, Streaming) HPL programming model (cont.)

The MPI is used by HPL originally for the homogeneous distributed-memory computers.

However, we can map one MPI process on each compute element (CPUGPU node), and then proceed with distributed computation within elements.

These compute elements connect each other with high-speed cluster interconnect, therefore CPU can perform the communication (swap and broadcast of the pivot row) with others, and the compute-intensive tasks (DGEMM and DTRSM) can be executed by the CPU and GPU cooperatively.

To parallelize the compute-intensive tasks, MP programming model on the host and parallel threads can be spawned at runtime when the progress enters the parallel region.

In this phase it is crucial to optimally balance parallel threads on CPU cores, in order to achieve maximum utilization of resources, and on the same time, to avoid processes to wait on each others.

Therefore, right load balancing scheme in MP is everything.

Hybrid (MPI, MP, streaming) HPL programming model (cont.)

For GPUs only, the good strategy is to use the streaming computing to develop the data parallel of the work assigned to them.

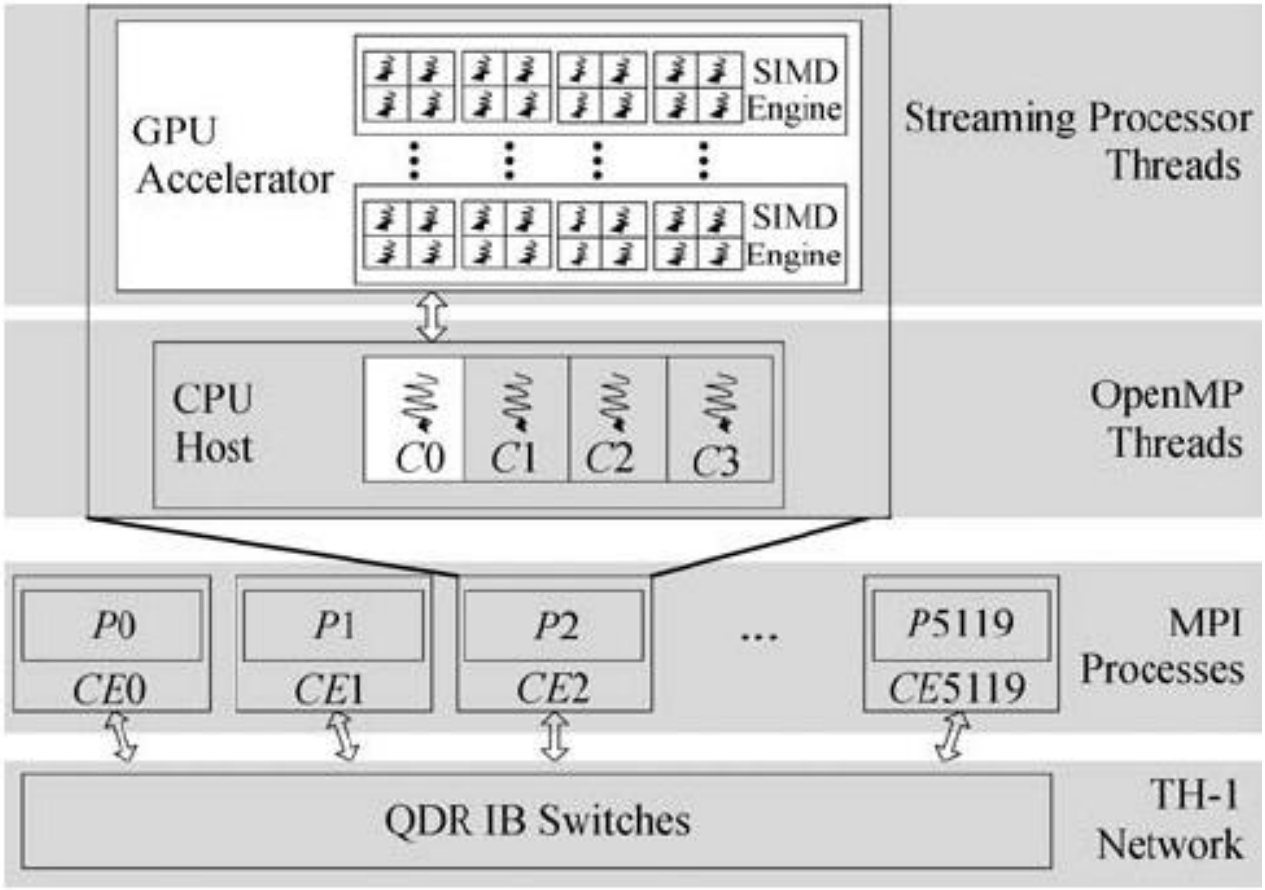
For example, one can use CAL (Compute Abstraction Layer) to program the kernels running on the GPU.

CAL can make users access the low-level native instruction set and memory of the massively parallel streaming processors in the GPU architecture.

CAL API exposes the stream processors as a single instruction, multiple data (SIMD) array of GPUs computational processors.

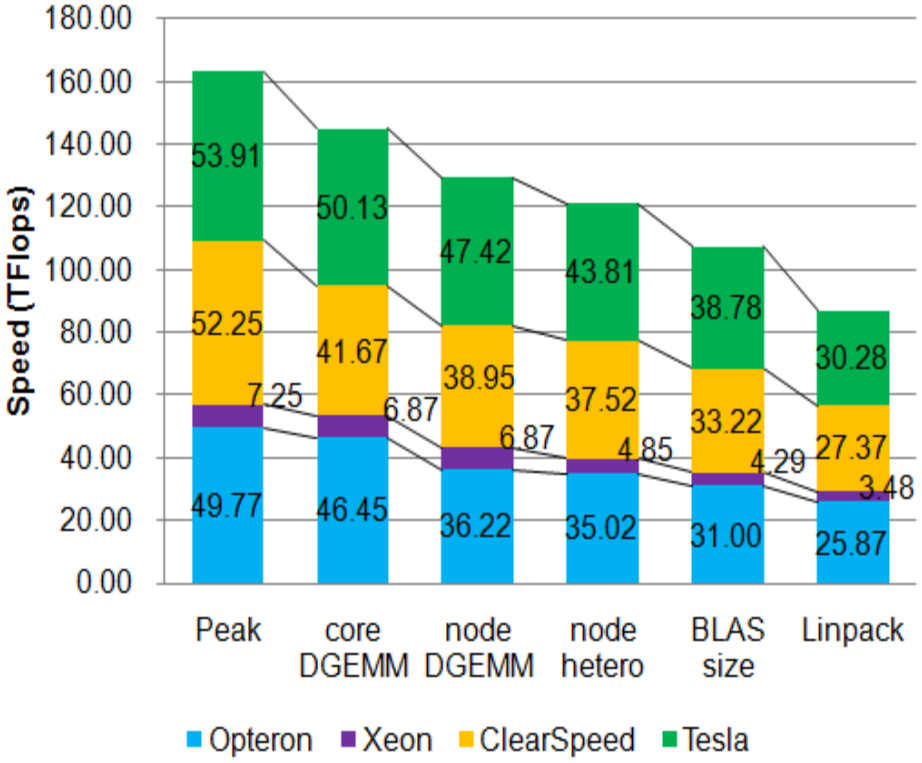
GPU device has a scheduler that distributes the workload to the SIMD engines according to the domain of the execution specified as invoked.

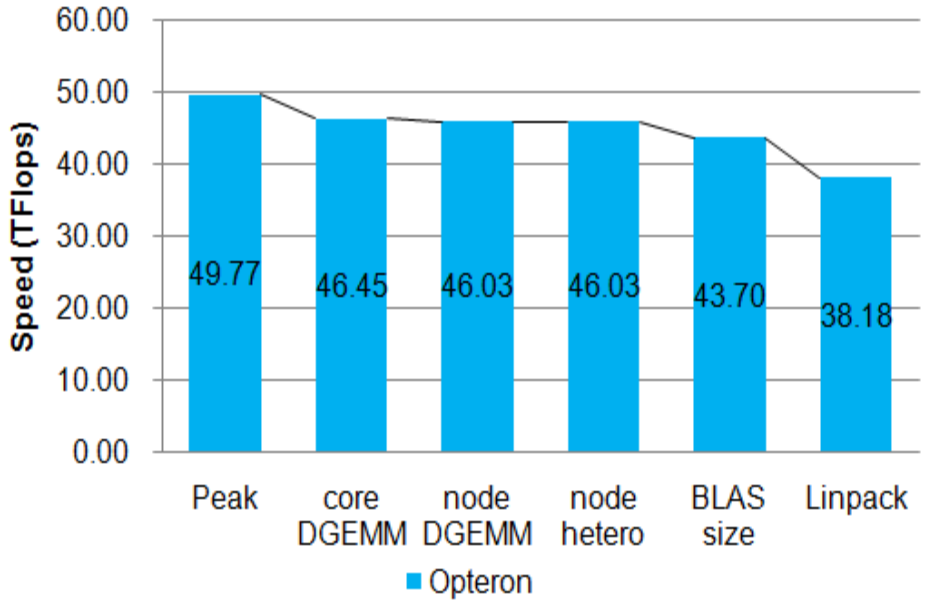
An example: Hybrid programming model of TianHe-1 system



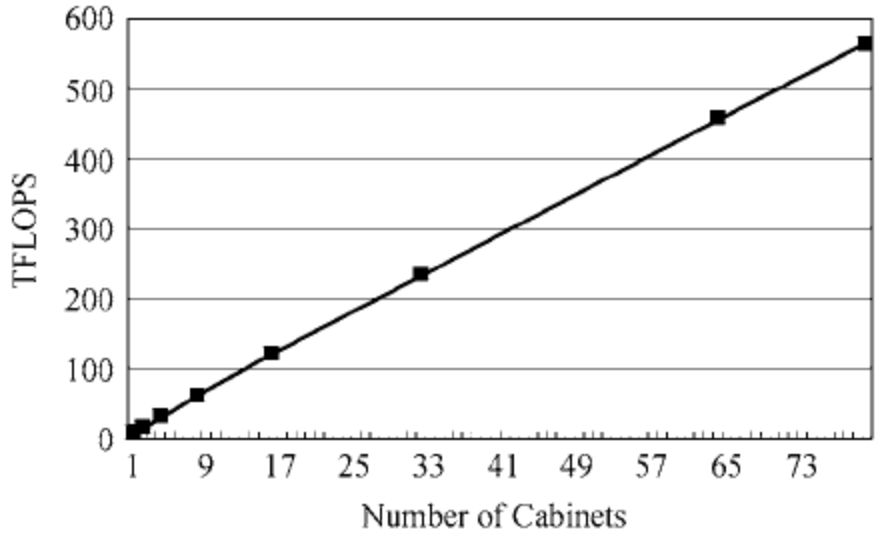
Example: HPL on TSUBAME 1.0. Each color indicates the type of processors.

Opteron cores → 10480; Xeon cores → 640; NVIDIA → 624; ClearSpeed 648

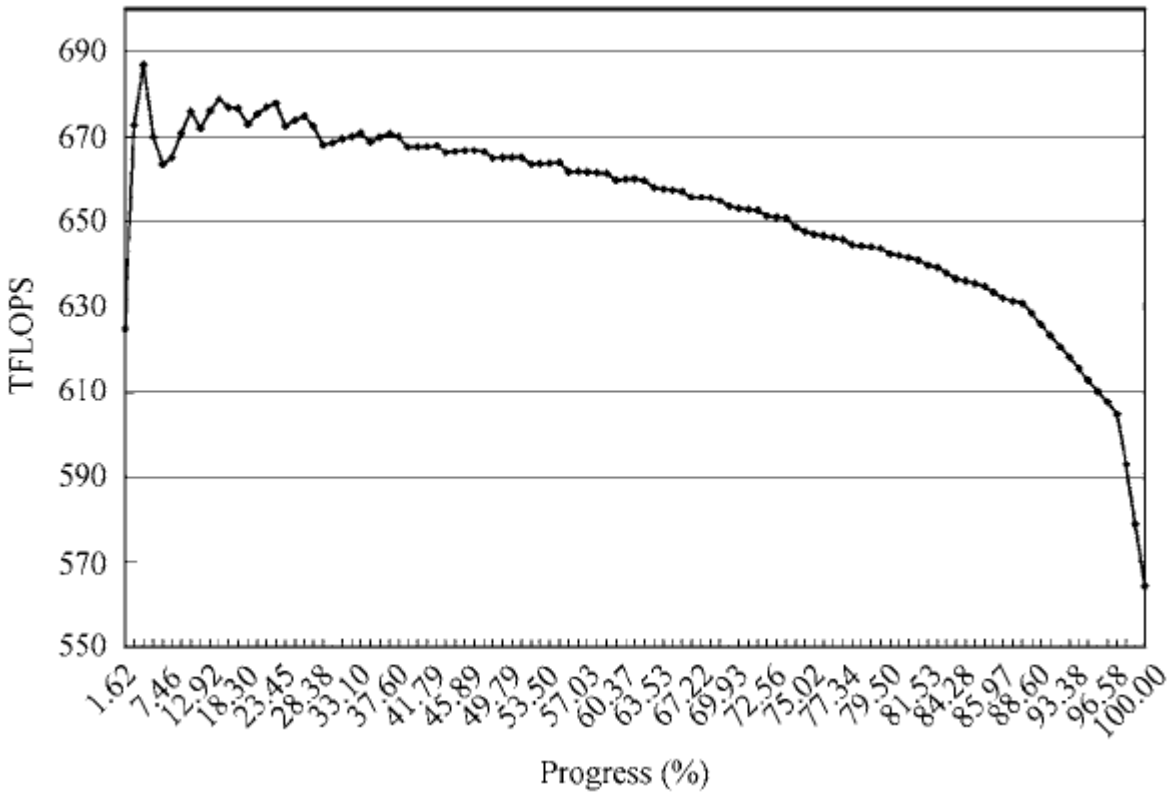




Example: HPL on Tianhe-1 system



Example: Tianhe-1 system (cont.)



Cloud is challenging us

We expect more and more HPC systems will be on Cloud.

HPC Cloud systems are even more challenging systems than traditional CPUGPU systems.

“Cloud” CPUGPU nodes can have various performance

“Cloud” communication links between nodes can have various bandwidth

Challenge: How to run HPL efficientlly on HPC Cloud?

How to…

- ☺ to distribute load on computational nodes in a way to minimize processing times…
- ☺ while minimizing communication times between nodes?

Example: Second Largest Eigenvalue Minimizer (SLEM)

- considers both processing power of computational nodes and speed of communication links
- while minimizing response times in the cloud

Some math needed here

objective:

$$\textit{Minimize}\ \ \mu(P) \qquad (1)$$

subject to:

$$P \geq 0, \quad P\mathbf{1} = \mathbf{1}, \quad P = P^T$$

P	Doubly stochastic and symmetric matrix
$\mu(P)$	Second matrix eigenvalue
P^T	Matrix transpose
1	Unary vector

Breaking the symmetry

$$eps_i = \sum_{j=1}^{m} \frac{ps_{ij}}{u_{ij}} \qquad (2)$$

$$Y = \begin{bmatrix} eps_1 & bw_{12} & . & bw_{1n} \\ bw_{21} & eps_2 & . & bw_{2n} \\ . & , & . & . \\ bw_{n1} & . & bw_{n,n-1} & eps_n \end{bmatrix} \qquad (3)$$

$$Y_s = \begin{bmatrix} eps_1 & \min(bw_{12}, bw_{21}) & . & \min(bw_{1n}, bw_{n1}) \\ \min(bw_{21}, bw_{12}) & eps_2 & . & \min(bw_{2n}, bw_{n2}) \\ . & , & . & . \\ \min(bw_{n1}, bw_{1n}) & . & \min(bw_{n,n-1}, bw_{n-1,n}) & eps_n \end{bmatrix} \qquad (4)$$

Grid adjacency (A) matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & . & a_{1n} \\ a_{21} & a_{22} & . & a_{2n} \\ . & . & . & . \\ a_{n1} & a_{n2} & . & a_{nn} \end{bmatrix} \qquad (5)$$

Hadamard (Ys * A)

$$P_n = \begin{bmatrix} eps_1 \cdot a_{11} & \min(bw_{12}, bw_{21}) \cdot a_{12} & . & \min(bw_{1n}, bw_{n1}).a_{1n} \\ \min(bw_{21}, bw_{12}) \cdot a_{21} & eps_2 \cdot a_{22} & . & \min(bw_{2n}, bw_{n2}) \cdot a_{2n} \\ . & . & . & . \\ \min(bw_{n1}, bw_{1n}) \cdot a_{n1} & \min(bw_{n2}, bw_{2n}) \cdot a_{n2} & . & eps_n \cdot a_{nn} \end{bmatrix} \qquad (6)$$

Computing optimal distribution

$$P_{n+1} = (D_n)^{-1/2} P_n (D_n)^{-1/2} \quad ; n = 1,\ldots,\infty \qquad (7)$$

$$D_n = \begin{bmatrix} d_1 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 \\ 0 & 0 & . & 0 \\ 0 & 0 & 0 & d_n \end{bmatrix} \qquad (8)$$

$$d_i = \sum_{j=1}^{n} P_{ij} \qquad ; \quad i = 1,\ldots,n \qquad (9)$$

Minimize $\mu(P)$ (10)

Subject to: $P \geq 0,\ P1 = 1,\ P = P^T$

$$Q_{n-1} = (D_n)^{1/2} Q_n (D_n)^{1/2} \quad ; n = \infty,\ldots,1 \qquad (11)$$

$$Q_{n-1} = (D_n)^{1/2} Q_n (D_n)^{1/2} \quad ; n = 100,\ldots,1 \qquad (12)$$

$$A_0 = \frac{trace(A)}{trace(Q_0)} Q_0 \qquad (13)$$