

Introduction to Numerical General Purpose GPU Computing with NVIDIA CUDA

Part 1: Hardware design and programming model

Dirk Ribbrock

Faculty of Mathematics, TU dortmund 2016





Why parallel processing?

Parallel Proc. Implementations

CPU vs. GPU

GPU hardware model

GPU's programming model

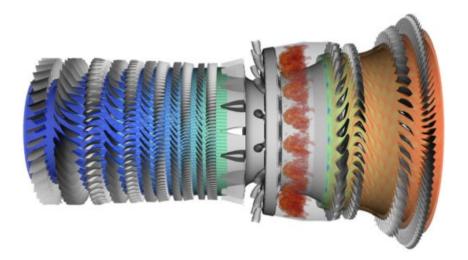
Scientific App. On GPU

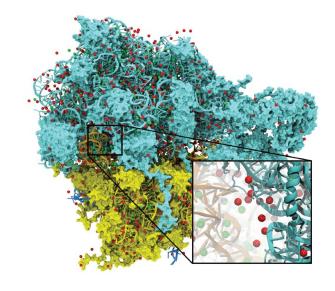
Performance Optimization



Our main challenge in scientific computing

- Long simulation times on single Processors for large problems
- High computational cost to run on super computers
- Low or moderate grid resolutions to keep the cost low





technische universität



Moore's Law

"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to Continue"

Gordon Moore (Intel), 1965

"OK, maybe a factor of two every **two years."**

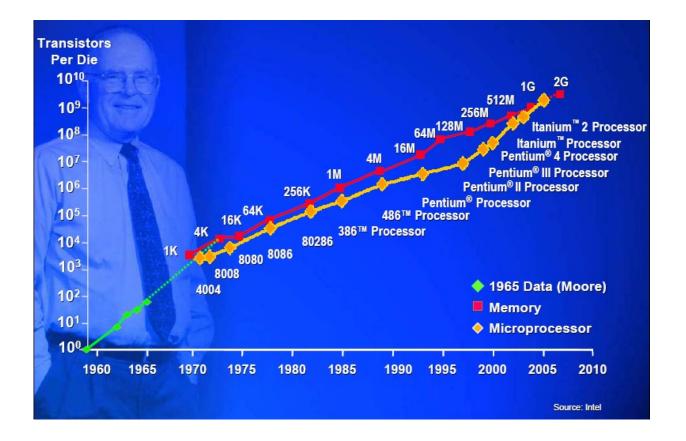
Gordon Moore (Intel), 1975 [paraphrased]



Why Parallel Processing

J technische universität dortmund

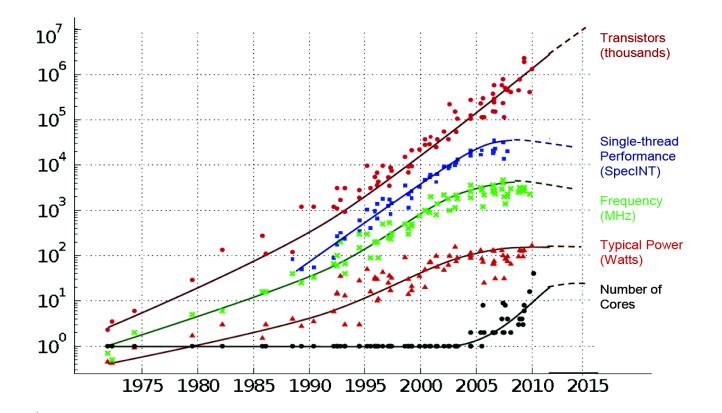
The Trend (1960-2005)





U technische universität dortmund

The Trend, (1970-2010)



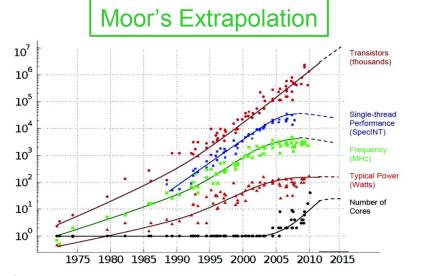
Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten Dotted line extrapolations by C. Moore

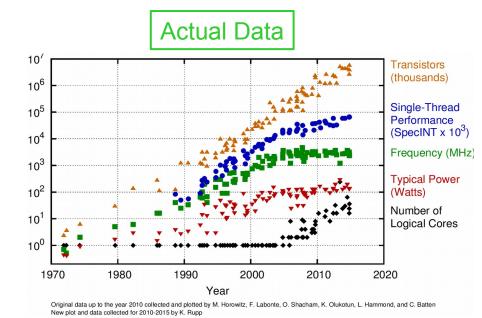


Why Parallel Processing

U technische universität dortmund

The Trend, (1960-2015)





Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten Dotted line extrapolations by C. Moore

Lesson's learnt

- Number of transistors and cores have keep increasing!
- Performance/core is only slightly increased.
- Frequency has remained constant to control heat/power.
- One must go for parallel implementations.



Parallel Processing Implementations

- Major approaches
- Distributed Memory
- Shared Memory
- GPGPU

Message Passing Interface (MPI)

OpenMP, Pthreads, Intel's TBB,...

CUDA, OpenACC, OpenCL,...





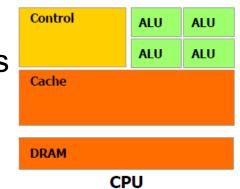


technische universität



Single Instructions, Multiple Data (SIMD)

- Large data caching and flow control units
- Few number of ALUs (cores)
- Example: Intel Xeon E5-2670 CPU
- 8 cores (16 threads)
- 2.6 GHz
- 2.3 billion transistors
- 20 MB on chip cache
- Flexible DRAM size



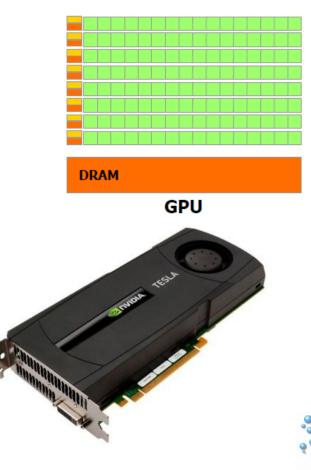




U technische universität dortmund

Single Instructions, Multiple Threads (SIMT)

- Small cache and control flow units
- Large number of ALUs (cores)
- Example: Kepler K20x GPU
 - 2688 (14 x 192) processor cores
 - 0.73 GHz
 - 28nm features
 - 7.1 billion transistors
 - 1.5 MB on-chip L2 cache
 - Only 6GB on chip memory





GPU Processing Model

GPUs are designed to apply the *same shading function* to many *pixels* simultaneously

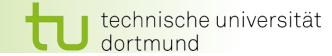


GPUs could be used to apply the same function to many data simultaneously

This is what most scientific computing need!



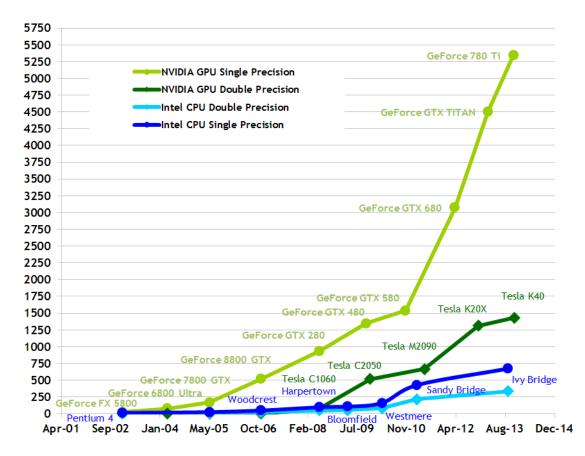
technische universität



GPU Computational Capabilities

High floating point power (5.3 TFlops in SP, 1.5 TFlops DP)

Theoretical GFLOP/s

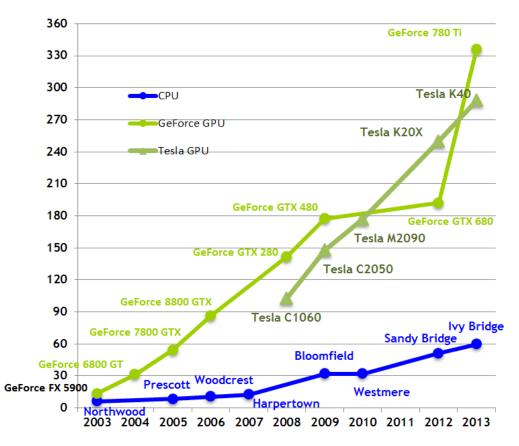






GPU Computational Capabilities

High Memory Bandwidth (more than 300 GB/s)



Theoretical GB/s



GPU Architecture

U technische universität dortmund

nVIDIA GPU Generations

GPUs come in different generations, e. g., Tesla, Fermi, Kepler,...

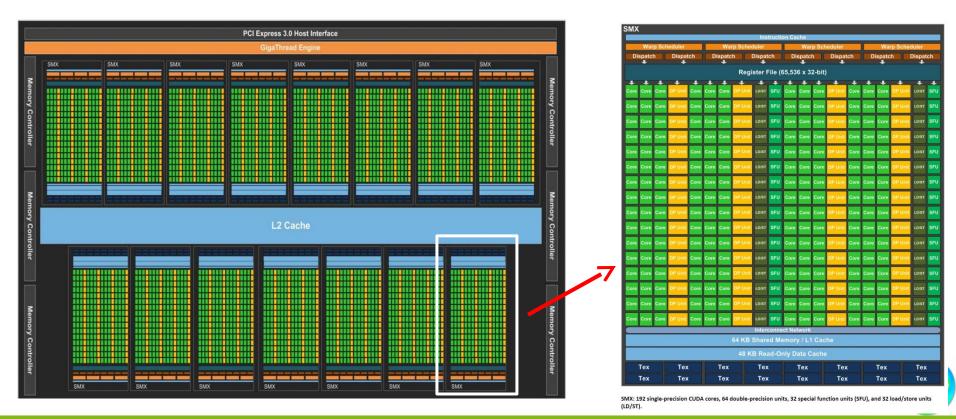
Each is labeled with a specific Compute Capability, e.g., 1.x, 2.x, 3.x, ...

GPU	G80	GT200	Fermi	Kepler			
Transistors	681 million	1.4 billion	3.0 billion	7.0 billion			
CUDA Cores	128	240	512 @ 1.15 GHz	2688 @ 0.73 GHz			
Double Precision Floating	None	30 FMA ops / clock	256 FMA ops /clock	1344 FMA ops/clock			
Point Capability							
Single Precision Floating	128 MAD	240 MAD ops /	512 FMA ops /clock	2688 FMA ops/clock			
Point Capability	ops/clock	clock					
Special Function Units	2	2	4	32			
(SFUs) / SM							
Warp schedulers (per SM)	1	1	2	2			
Shared Memory (per SM)	16 KB	16 KB	Configurable 48 KB or	Configurable 48 KB, 16			
			16 KB	KB or 32 KB			
L1 Cache (per SM)	None	None	Configurable 16 KB or	Configurable 48 KB, 16			
			48 KB	KB or 32 KB			
L2 Cache	None	None	768 KB 1.5 MB				
ECC Memory Support	No	No	Yes	Yes			
Concurrent Kernels	No	No	Up to 16	Up to 32 + Dyn. Parallel			
Load/Store Address Width	32-bit	32-bit	64-bit	64-bit			

GPU Architecture

GPU Hardware Architecture

- Set of SIMD Streaming Multiprocessors (SMX)
- Each Multiprocessor has its own set of computational resources.



Amin Safi | TU Dortmund

technische universität

GPU Architecture

Kepler Architecture (Compute Capability 3.x)

- 2688 cores are divided among14 SMXs, each having 192 processor cores.
- Each 3 cores serve as 1 double precision unit.
- Each SMX multiprocessor has a set of:
 - · 65 KB L1 / Shared memory
 - · 48 KB read-only caches
 - · Constant and texture caches
 - · Registers
- 32 special function units.

SMX																				
Instruction Cache																				
Warp Scheduler					Warp Scheduler					Warp Scheduler					Warp Scheduler					
Dispatch Dispatch			Dispatch Dispatch				Dispatch Dispatch					Dispatch Dispatch								
Register File (65,536 x 32-bit)																				
	Core	Core	DP Unit		Core		DP Unit				Core		DP Unit							
													_							
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	
													_							
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	
													_							
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	
													_							
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	
															Conc					
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SEU	
		Conc										Conc		Conc	Conc					
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	
Core	Core	Core	DP Unit	Core	Core	Coro	DP Unit	LD/ST	SFU	Coro	Core	Coro	DP Unit	Coro	Coro	Core	DP Unit	LD/ST	SELL	
Gore	Core	core	or only	Gore	cona	core			01-0	Gore	cone	Sone	or onit	Cona	Core	Gore		20101	are	
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SELL	
Core	Core	Core	- on one	Gole	Core	Core		20/01	GILO	COTE	Core	COLE	on onit	Core	Core	Gare	- Conne	Lorot		
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	
								Inter	conne	ct Ne	twork									
64 KB Shared Memory / L1 Cache																				
							48 K	B Re	ad-O	nly D	ata C	ache)							
	Tex	T	Tex		Г	Tex		Te>	(Tex	T	Te>	(Tex		Tex		
	Tex		Tex			Tex		Te>	<		Tex		Te>			Tex		Tex		
			a										L							

SMX: 192 single-precision CUDA cores, 64 double-precision units, 32 special function units (SFU), and 32 load/store units (LD/ST).



Page



Page

Programming On GPUs technische universität dortmund

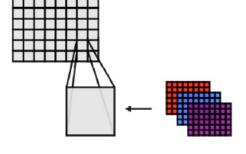
Graphical Languages, e.g., OpenGL, DirectX,...

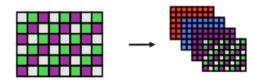
Using graphics instructions for scientific calculations

Very hard to develop codes for non-expert programmers

- Unable to fully exploit the computational power of GPUs
- Low overall efficiency





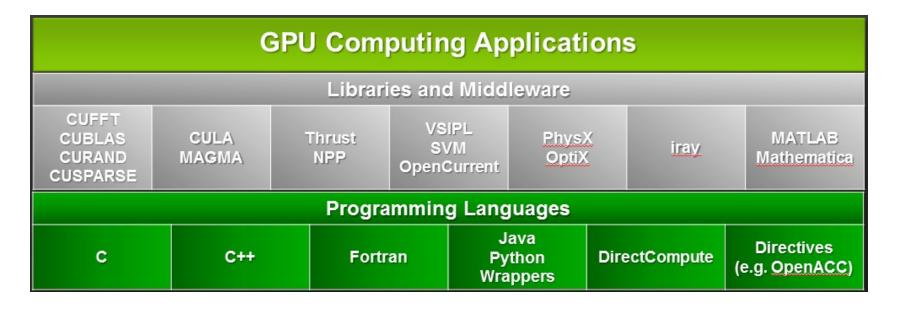


Courtesy, John Owens, UC Davis

Programming On GPUs

GPGPU Languages e.g. CUDA, OpenCL, OpenACC

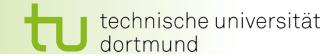
- Designed specifically for scientific programming.
- Relatively easy implementations.
- Can extract almost all the power of hardware.
- High numerical performances are then achievable.



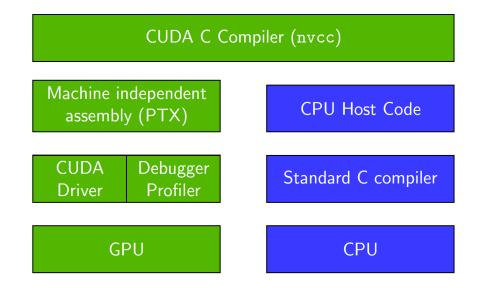
Compute Unified Device Architecture



technische universität



CUDA Toolkit



CUDA Software Development Kit (SDK)

Optimized libraries: math.h, BLAS, FFT

Integrated CPU and GPU source code

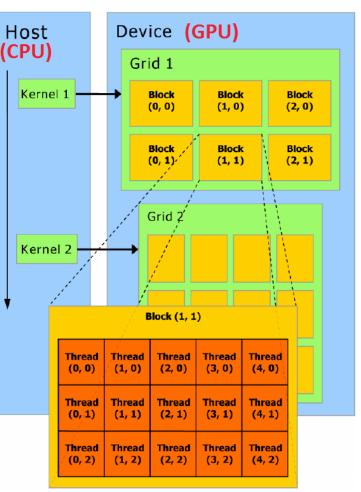


Programming Model of CUDA

- Fine-grained parallelization by launching many active threads via kernels
 Coarse grained parallelization via blocks and grid.
- Threads are grouped into blocks(1D, 2D or 3D)
- Blocks are organized into a grid(1D, 2D or 3D)
- Kepler supports max. 2048 active threads per SMX
- Threads are lightweight:
 - Small creation overhead
 - "instant "switching
 - Efficiency achieved through1000's of threads

For a complete Device query see:

<u>https://www.microway.com/hpc-tech-tips/nvidia-tesla-k20-gpu-accelerator-kepler-gk11</u> <u>0-up-close/</u>



dortmund

technische universität





Essential CUDA Extensions to C/C++

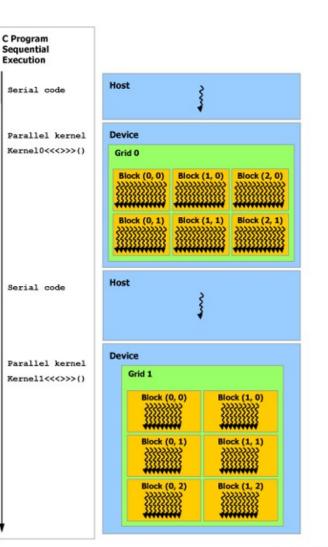
- Kernel execution directives
 myfunction<<<GridDim, BlockDim>>> (...)
- Built-in variables for grid/block size and block/thread index
 - threadIdx.x , threadIdx.y , ...
 - blockIdx.x , blockIdx.y, ... , blockDim.x, ...
- Function type qualifiers
 - · Specify where to call and execute a function
 - <u>device</u>, <u>global</u> and <u>host</u>
- Variable type qualifiers

• _____device___, ____constant__ and _____shared___



Heterogeneous workflow

- kernels execute on a GPU and the rest of the C program executes on a CPU.
- CUDA threads execute on a physically separate device.
- Allows for asynchronous pre- and post-processing on CPU.
- CUDA assumes that both the host and the device maintain their own separate memory spaces in DRAM.

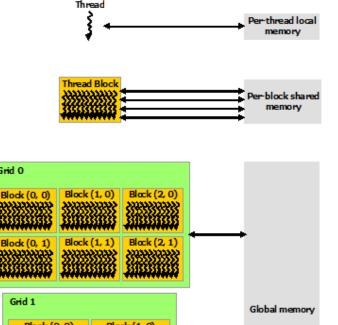


technische universität



The memory hierarchy

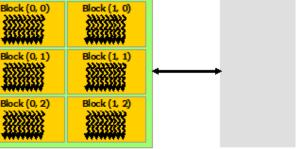
- The grid of blocks in each kernel has access to global memory.
- Data dispatched from global memory is stored in fast L2 cache lines.
- Threads within a block can read from and write to *shared memory* asynchronously.
- Each thread has access to on-chip *local memory*.
- Different memories make up the so-called device memory.



Grid O

Grid 1

technische universität







What happens to a block?

Software

-Threads from one block may cooperate:

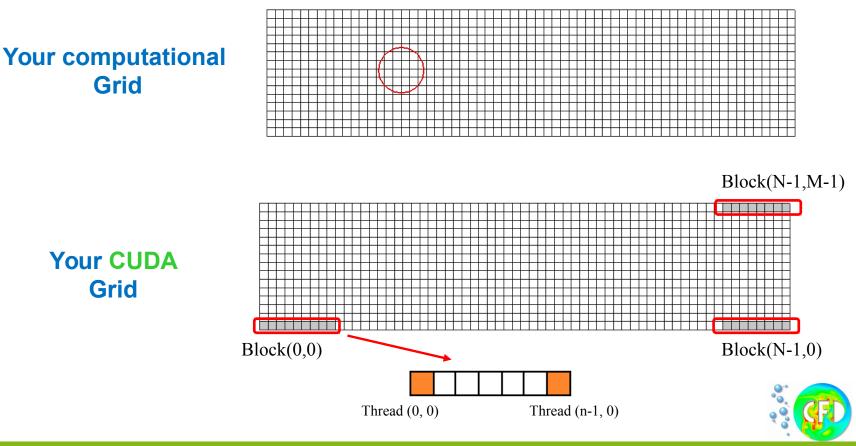
- · using data in shared memory
- can get synchronized.

Hardware

- A block runs on one multiprocessor.
- Hardware is free to schedule any block on any multiprocessor
- More than one block can reside on one multiprocessor
- A block is split into multiple warps of 32 threads (details given later).



- How do threads perform calculations in parallel?
 - In some numerical scientific applications, each thread is in charge of one data element in your computational domain.



technische universität



The nvcc compiler workflow

- GPU kernels are typically stored in files ending with .cu
- The rest of the code could be stored in the same .cu file or separately in other .cu, .c or .cpp files.
- **nvcc** separates the device code form the host code and:
 - Automatically handles #include's and linking libraries
 - Compiles the device code into an assembly form (ptx code) and/or binary form (cubin object).
 - Modifies the host code to replace <<<...>>> (for kernel calls) with associated CUDA-runtime directives in the ptx code.
 - Uses the host compiler(C/C++) to compile CPU code.
- Application can then
 - Either link to the compiled host code,
 - Or ignore the modified host code (if any) and use the CUDA driver API to load and execute the PTX code or cubin object.





A typical CUDA program includes:

- Explicitly managing host and device memory
 - Allocatoin of data on CPU & GPU
 - Transfers of data form CPU to GPU
- Setting the dimensions of blocks and grids.
- Launching kernels on GPU
- Copying the results back to CPU for post-processing.
- Freeing the memory on CPU & GPU.





How a kernel works?

An Element-wise Matrix Addition Code

CPU Program

```
void add_matrix
  ( float* a, float* b, float* c, int N ) {
    int index;
    for ( int i = 0; i < N; ++i )
        for ( int j = 0; j < N; ++j ) {
            index = i + j*N;
            c[index] = a[index] + b[index];
        }
}
int main() {
    add_matrix( a, b, c, N );
}</pre>
```

CUDA Program

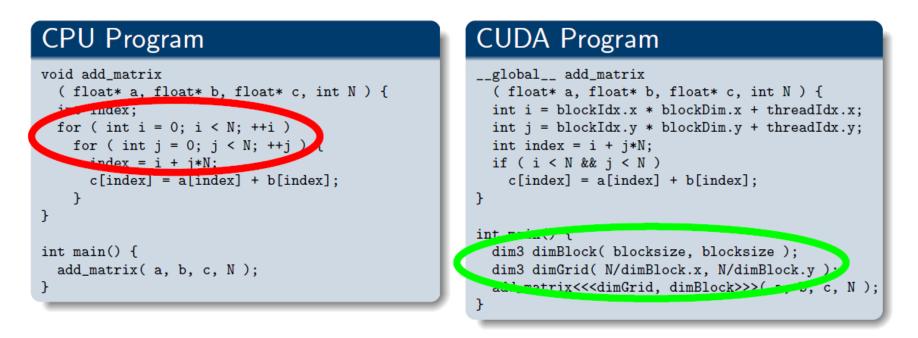
```
__global__ add_matrix
 ( float* a, float* b, float* c, int N ) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    int index = i + j*N;
    if ( i < N && j < N )
        c[index] = a[index] + b[index];
}
int main() {
    dim3 dimBlock( blocksize, blocksize );
    dim3 dimGrid( N/dimBlock.x, N/dimBlock.y );
    add_matrix<<<dimGrid, dimBlock>>>( a, b, c, N );
}
```





How a kernel works?

An Elementwise Matrix Addition Code



The nested for-loops are replaced with an implicit grid



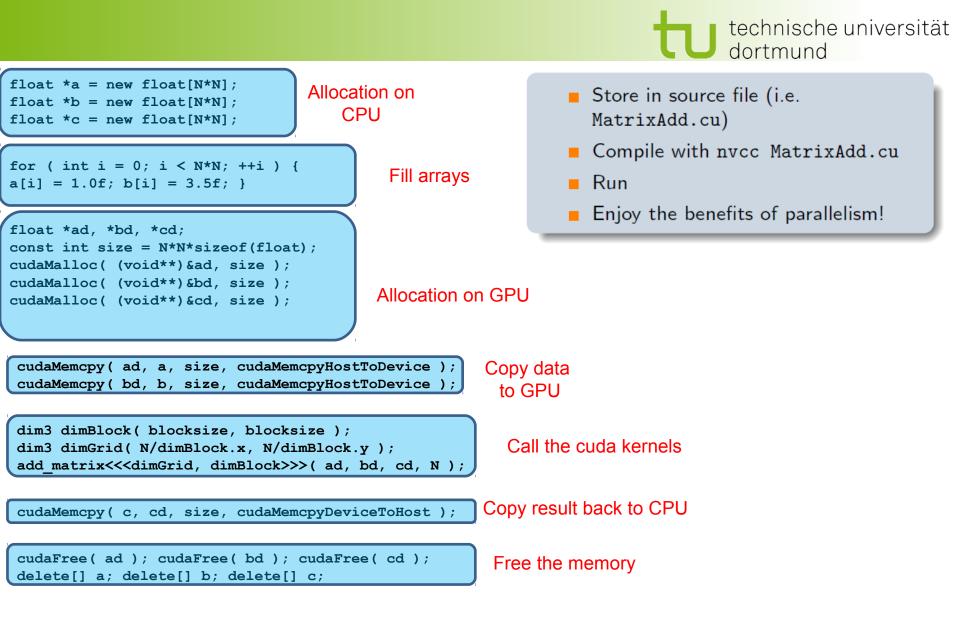


A rather complete example

Input data size and block size

Compute Kernel







technische universität dortmund

Some key notes

- The size of blocks and grids are determined in accordance to the size of problem and device memory limitations
- Kernel calls are synchronous relative to each other
- Control returns to CPU after launching a kernel (asynchronous to CPU instructions)
- Memory transfers between GPU and CPU are completely synchronous
- Memory transfers using pinned memory are asynchronous





U technische universität dortmund

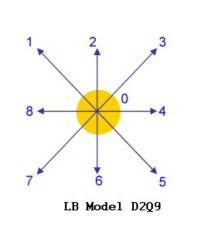
Lattice Boltzmann Simulation

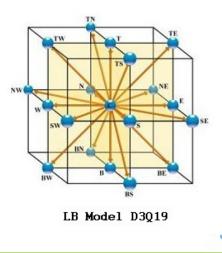
$$\frac{\partial f_i}{\partial t} + c_i \cdot \nabla f_i = Q_i = -\frac{1}{\lambda} (f_i - f_i^{eq}(\rho, u))$$

$$\rho = \sum_{i} f_{i}$$
 , $\rho u = \sum_{i} c_{i} f_{i}$

f f c_x

U is macroscopic velocity





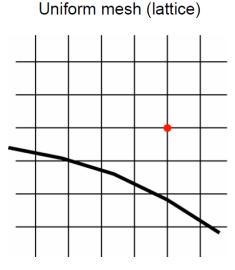


A CFD Example

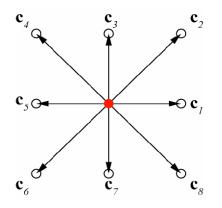


Lattice Boltzmann Simulation

- We use the D2Q9 model for 2D flow with 9 velocities
- The Navier-Stokes Eqs are recovered for incompressible, isothermal flow in hydrodynamic limit



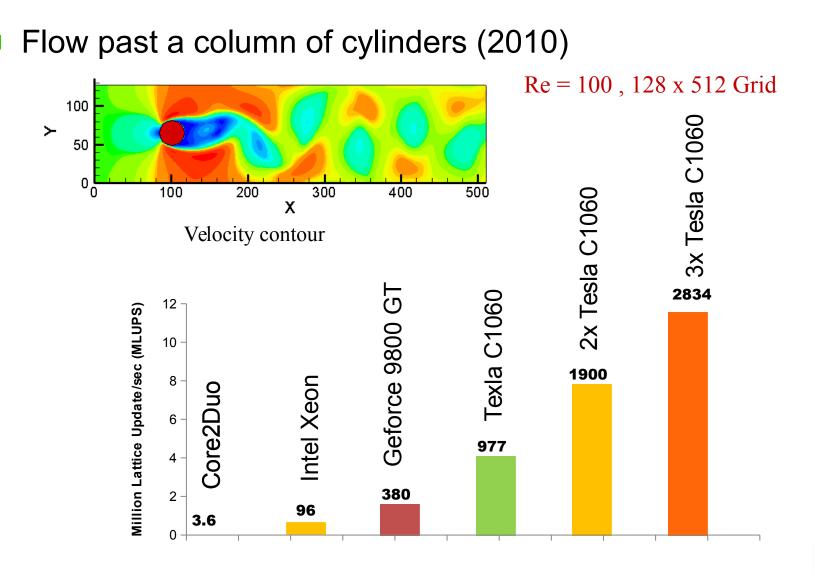
Restrict microscopic velocities to a finite set:





CFD Examples

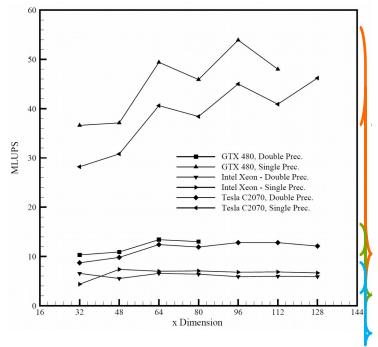
J technische universität dortmund





CFD Examples

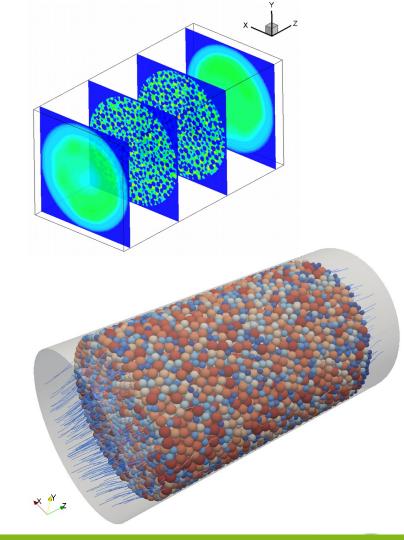
- 3D Multi-component flow of O₂ and N₂(2012)
- Air flow segregates into its ingredients
- Multicompent, Entropic LB model



GPU, Single Prec.

GPU, Double Prec.

32 core CPU, Single and Double Prec.

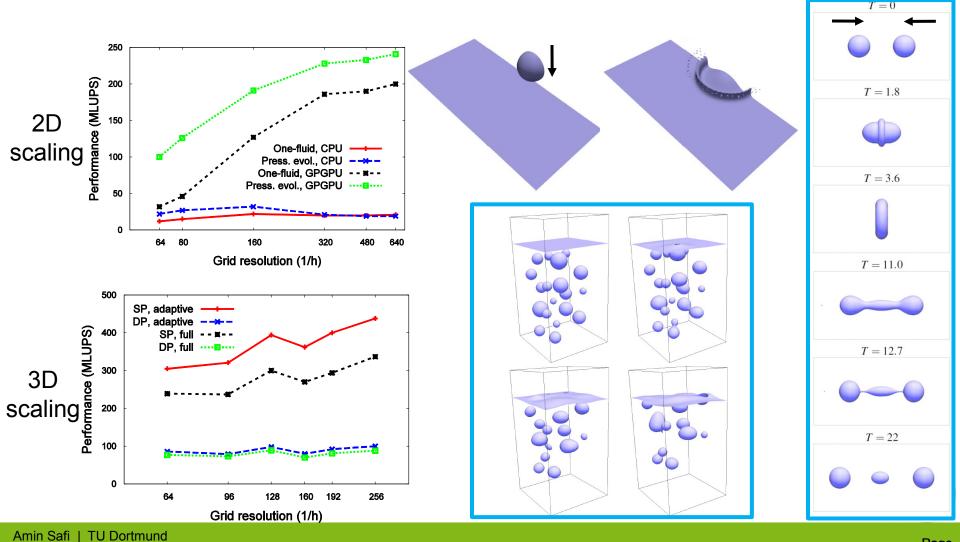


technische universität

CFD Examples

technische universität dortmund

2D and 3D two-phase flows (2015)



A CFD Example



Performance of our code

- Using different GPU generation, we achieved **10x-20x** speedup.
- Almost real-time simulations for early stage evaluations.
- SP is 3-4 times faster than DP was this free speedup if possible!!
- These speedups are for an optimized version of our code.
- Otherwise, the speedup would drop drastically even on a most modern GPUs.

Optimization is Vital !



Performance Optimization to technische universität

- 4 Major Optimization Strategies
 - Memory Access optimization
 - Increasing Hardware Occupancy
 - Control Flow Optimization
 - Instruction Optimization

The first two are the most important ones



Memory Access Optimization to technische universität dortmund

Why so important?

- Memory transfer accounts for the majority of simulation time in memory bound applications (most large data scientific applications).
- Theoretical bandwidth between GPU DRAM and SMXs is more than 250 GB/s.
- Up to 85% of this bandwidths is achievable only and only if:

The memory accesses are **coalesced** by threads in a warp

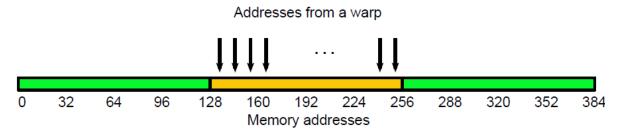
Otherwise, the effective bandwidths drops to 10% of max value.



Memory Access Optimization

Memory access anatomy

Memory accesses by a warp (32 threads) are *coalesced* into as few as one transaction when certain access requirements are met



- No. of transactions = number of cache lines necessary to service the warp.
- Cache line size: 128 byte L1 segments in Fermi, 32 byte L2 segments in Kepler.
- 100% memory performance if all required data are found in one cache line
- Poorest performance if none of the other data items in the cache line are ever used (*cache thrashing*).

Keep block sizes as multiples of 32.
 Avoid scattered, non-local data dependencies if possible!

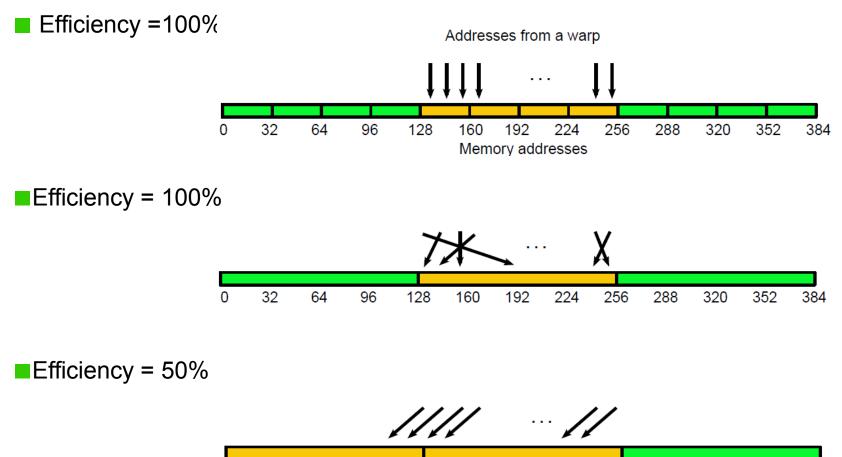


technische universität

dortmund

Memory Access Optimization

Access pattern examples:



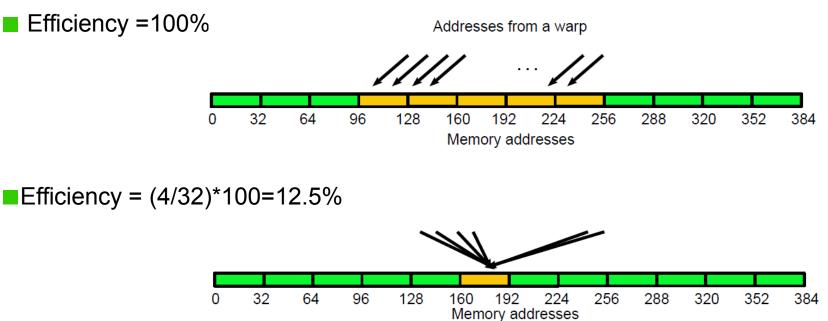


technische universität

dortmund

Memory Access Optimization

Access pattern examples:





technische universität

dortmund

Increasing Occupancy



Multiprocessor Occupancy

- Each MP has a limited register and shared memory
- Each MP manages a maximum of 2048 threads simultaneously
- Each thread takes up a certain number of registers and shared memory

 $Occupancy = \frac{number of active threads per multiprocessor}{maximum number of possible active threads}$

- Care must be taken to keep the Occupancy above 25%
- A 100% occupancy does NOT mean a high performance!!!!



Increasing Occupancy



How to control Occupancy?

- Use the compiling option: --ptxas-options=-v to probe your kernels for register and shared memory consumption
- Force a maximum number of register for each thread using: -maxregcount=##
- Kernel's shared memory consumption can not be forced explicitly, its all inaside your code
- Experiment with numbers to find a proper balance, using.....

CUDA Occupancy Calculator



Increasing Occupancy

U technische universität dortmund

Occupancy calculator

<form></form>			CUDA_Occupancy_Calculatorx/s [Compatibility Mode] - Microsoft Excel	
All Control of the set		Review View Ac		
		B C		
Construction		-		
Construction	CUDA GPU Occupancy C	alculato	Click Here for detailed instructions on how to use this occupancy calculator.	
A Select Compute Capability (click): 1 B Select Compute Capability (click): 1	3			
6 Ty Sector Compute Capability (Els6): 13 (198) 7 1 Letter for an expect Compute Capability (Els6): 13 (198) 7 1 Letter for an expect Compute Capability (Els6): 13 (198) 7 1 Departs els (Els6): 13 (198) 8 2 Departs els (Els6): 13 (198) 7 2 Departs els (Els6): 13 (198) 8 2 Departs els (Els6): 13 (198) 9 2 Departs els (Els6): 13 (198) (198) 9 2 Departs els (Els6): 13 (198) (198) (198) 9 2 Departs els (Els6): 13 (198) (198) (198) (198) (198) (198) (198) <		nelp)		
7 Varying Block Size 9 2.1 For your resource using: 2.2 For your resource using: 10 Contrast pre topic for your resource using: 2.2 For your resource using: 10 Contrast pre topic for your resource using: 2.2 For your resource using: 10 Contrast pre topic for your resource using: 1.2 For your resource using: 10 Contrast pre topic for your resource using: 1.2 For your resource using: 10 Contrast pre topic for your resource using: 1.2 For your resource using: 10 Contrast pre topic for your resource using: 1.2 For your resource using: 10 Contrast pre topic for your resource using: 1.2 For your resource using: 10 Contrast pre topic for your resource using: 1.2 For your resource using: 10 Contrast pre topic for your resource using: 1.2 For your resource using: 1.2 For your resource using: 10 Previous Line for CPU Compare topic for your resource using pre topic your resource using: 1.2 For your resource using pre topic your resource your resource using pre topic your resource using pre topic your resource using pre topic your resource your resource using pre topic your resource your resource your resource your resource y		12 (11-1-)	The other data points represent the range of possible block sizes, register counts, and shared memory	ry allocation.
$\frac{1}{2} \frac{\frac{1}{2} \operatorname{Entry four resource taskge}}{\frac{1}{2} \operatorname{Port out in the graphs}}{\frac{1}{2} \operatorname{Entry four set in the graphs}}{\frac{1}{2} Entry four set in the graph in t$		1.3 (Help)		
 9) A Line type resource usage: 1) The track of the type resource usage: 1) The type resource usag	3		Varying Block Size Varying	Register Count
11 Register Par Thead 100 12 Register Par Thead 100 13 Register Par Thead 100 14 Register Par Thead 100 15 Register Par Thead 100 16 Register Par Thead 100 17 Register Par Thead 100 18 Register Par Thead 100 10 Register Par Thea				
1 Register Per Intered Shered Manny Per Block types		256 (Help)	48 48	
100 Indication and suppling below this incol 150 Active Varge per Multiprocessor 1024 170 Active Varge per Multiprocessor 1024 180 Active Varge per Multiprocessor 1024 170 Active Per Multiprocessor 1024 170 Active Per Multipr		16		
10 Outcomponent Stati singleyad here and in the graphing 13 Outcomponent Stati singleyad here and in the graphing 14 Outcomponent Stati singleyad here and in the graphing 15 Active Thread Ser Multiprocessor 16 Outcomponent Stati singleyad here and in the graphing 17 Trived Ser Multiprocessor 10 Outcomponent Ser Multiprocessor 10 Trived Ser Multiprocessor 10 Trived Ser Multiprocessor 10 Trived Ser Multiprocessor 10 Ser Multiprocessor		4096	40	
Image: Signed from and in the graphs: Image: Signed from and in the graphs: If Active Thread Blocks per Multiprocessor 1000 Physical Limits for GPU Compute Capability: 1.3 Register allocation granularity: 1000 Register allocation granularity: 1000 Maximum Thread Block Ser Multiprocessor 1000 Sime Memory Lines allocation granularity: 1000 Maximum Thread Block Ser Multiprocessor Block Ser Multiprocessor M			My Block Size My Register	
¹ ² ² ³ ² ³ ² ³ ² ³ ² ⁴ ³ ⁴ ⁴ ⁴ ⁵ ⁴ ⁴ ⁴ ⁵ ⁴ ⁴ ⁵ ⁴ ⁴ ⁵ ⁴ ⁵ ⁴ ⁵ ⁴ ⁵ ⁴ ⁴ ⁵ ⁵ ⁴ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵				
¹ ² ² ³ ² ³ ² ³ ² ³ ² ⁴ ³ ⁴ ⁴ ⁴ ⁵ ⁴ ⁴ ⁴ ⁵ ⁴ ⁴ ⁵ ⁴ ⁴ ⁵ ⁴ ⁵ ⁴ ⁵ ⁴ ⁵ ⁴ ⁴ ⁵ ⁵ ⁴ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵				
¹ ² ² ³ ² ³ ² ³ ² ³ ² ⁴ ³ ⁴ ⁴ ⁴ ⁵ ⁴ ⁴ ⁴ ⁵ ⁴ ⁴ ⁵ ⁴ ⁴ ⁵ ⁴ ⁵ ⁴ ⁵ ⁴ ⁵ ⁴ ⁴ ⁵ ⁵ ⁴ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵				
¹ ² ² ³ ² ³ ² ³ ² ³ ² ⁴ ³ ⁴ ⁴ ⁴ ⁵ ⁴ ⁴ ⁴ ⁵ ⁴ ⁴ ⁵ ⁴ ⁴ ⁵ ⁴ ⁵ ⁴ ⁵ ⁴ ⁵ ⁴ ⁴ ⁵ ⁵ ⁴ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵		32		
¹ ² ² ³ ² ³ ² ³ ² ³ ² ⁴ ³ ⁴ ⁴ ⁴ ⁵ ⁴ ⁴ ⁴ ⁵ ⁴ ⁴ ⁵ ⁴ ⁴ ⁵ ⁴ ⁵ ⁴ ⁵ ⁴ ⁵ ⁴ ⁴ ⁵ ⁵ ⁴ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁵ ⁵ ⁴ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵ ⁵		100%		
32 Physical Limits for GPU Compute Capability: 1.3 Threads per Multiprocessor 1024 To Trad # of 22 Kines Memory Per Multiprocessor 1024 Threads Per Block thread Block Size 1000 (Size 1000) (Size	1			
24 Thread Sper Warp 32 25 Warps & Multiprocessor 1024 26 Thread Sper Kultiprocessor 1024 27 Thread Sper Kultiprocessor 1634 28 Alter Sper Kultiprocessor 1634 29 Thread Sper Kultiprocessor 1634 20 Register allocation manuarity block 21 Stard Memory Per Multiprocessor (bytes) 1634 25 Stard Memory Per Multiprocessor (bytes) 1634 26 Thread Block Size 512 37 Warps 409 4 Intead by Per Multiprocessor 512 36 Allocation per Intread Block Size 512 37 Warps 409 40 Thread Block Size 512 36 Allocation Per Thread Block Size 612 37 Warps 409 40 Thread Block Size 612 31 Maximum Thread Block Size per Multiprocessor Blocks 31 Limited by Registers per Multiprocessor Blocks 32 Maximum Thread Block Size per Multiprocessor				
25 Warps per Multiprocessor 32 27 Thread Blocks per Multiprocessor 1024 27 Register allocation unit size 0 26 Register allocation unit size 0 27 Bread Block Size 1024 28 Register allocation unit size 0 29 Register allocation unit size 0 20 Marginer allock Size 512 20 Marginer allock Size 512 21 Marginer allock Size 512 24 Maximum Thread Block Size 512 26 Animum Thread Block Size 605 27 Marginer allocation per thread Block Size 605 26 Maximum Thread Block Size 605 27 Marginer allocation organiantly throcessor 605 26 Maximum Thread Block Size 605 27 Maximum Thread Block Size 605 26 Maximum Thread Block Imm Per Multi			8 8	
26 Therad Block per Multiprocessor 1024 37 Therad Block Size 1534 38 Register allocation granularity block 37 Maximum Thread Block 512 38 Register allocation granularity 512 39 Register allocation granularity 16384 30 Register allocation granularity 16384 39 Register allocation granularity 16384 30 Register allocation granularity 16384 31 Shared Memory Allocation unt size 512 30 Register allocation granularity 16384 400 Enter allocation granularity 16384 31 Allocation per thread Block 16384 31 Inter dy Max Warp / Elocka per Multiprocessor Elocka 32 Inter dy Max Warp / Elocka per Multiprocessor Elocka 33 Copyright and License 2.3 34 Inter dy Max Warp / E				
27 Therad Blocks per Multiprocessor 8 28 Therad Block incidence production unit size 16344 29 Register allocation unit size 16344 29 Register allocation unit size 16344 29 Register allocation unit size 16344 20 Register allocation unit size 16344 20 Maximum Thread Block Size 16344 30 Maximum Thread Block Size 16346 36 Addition consulting the occupancy data in blue 164 10 Thread Block Limit Per Multiprocessor 164 31 Imited by Max Warps / Blocks per Multiprocessor 164 32 Maximum Thread Block Limit Per Multiprocessor 164 34 Maximum Thread Block Size 16394 34 Maximum Thread Block Size 16394 41 Imited by Max Warp / Blocks per Multiprocessor 164 35 Imited by Max Warp / Blocks per Multiprocessor 164 40 164 182 164 164 164 164 164 164 164 164 164 164 164 164 164 </td <td></td> <td></td> <td></td> <td></td>				
29 Register allocation unit size 512 Threads Per Block Register allocation unit size 512 30 Register allocation granularity block Block Register allocation granularity Register allocatio		8	0 64 128 192 256 320 384 448 512 576 640 704 768 832 896 9601024	4 5 5 5 5 5 5 7 7 8 8 8 8 9 9 1 1 1 1 1 1 1 1 1 1 1 1 1 1
3 Neglister allocation (min State 0 Ji Z 9 Register allocation granularity block 15 Shared Memory Ver Multiprocessor (bytes) 15384 25 Mared Memory Allocation granularity (for register allocation) 2 26 Maximum Thread Block 512 27 Warps 4096 28 mared Memory 4096 29 Maximum Thread Block 4096 20 Thread Block Size 4096 20 Thread Block Size 4096 20 Thread Block Size 6006 20 Thread Block Size 6006 21 Maximum Thread Blocks Per Multiprocessor Elocks 21 Minde by Max Warps / Blocks per Multiprocessor 61 22 Maximum Thread Block Limit Per Multiprocessor 61 21 Minde by Max Warps / Blocks per Multiprocessor 61 21 Minde by Shared Memory per Multiprocessor 61 22 Maximum Thread Block Limit Per Multiprocessor 61 23 Copyright and License 23 24 Maximum Thread Block Limit Per Multiprocessor 61 25 Limited by Cocupancy Calculator 7 26 Limited by Copyright and License 23 26 Limited by Cocupancy Calculator 80				
31 Shared Memory per Multiprocessor (bytes) 16384 22 Shared Memory Allocation unit size 512 33 Shared Memory Allocation unit size 512 34 Maximum Thread Block Size 512 36 Adaximum Thread Block Size 512 36 Adaximum Thread Block Size 4096 37 Warp Sace Memory Lissee 4096 38 Registers 4096 30 Thread Blocks Per Multiprocessor Blocks 31 Shared Memory Per Multiprocessor Blocks 32 Limited by Registers per Multiprocessor Blocks 34 Maximum Thread Block Per Multiprocessor 6 34 Limited by Charled Memory Per Multiprocessor 6 34 CubA Coccupancy Calculator 6 35 Copyright and License 2.3 36 Copyright and License 2.3 36 Copyright and License 2.3				egisters Per Thread
22 Shared Memory Allocation unit size 512 33 Warp allocation granularity (for register allocation) 2 54 Maximum Thread Block 512 56 Allocation Per Thread Block 6 77 Warps 6 58 4096 59 Shared Memory Usage 60 These data are used in computing the occupancy data in blue 14 Limited by Max Warps / Blocks per Multiprocessor Blocks 12 Maximum Thread Block Limit Per Multiprocessor 6 16 Thread Block Limit Per Multiprocessor 6 16 Thread Block Limit Per Multiprocessor 1 16 CupA Occupancy Calculator 2.3 16 Copyright and License 2.3 16 Copyright and License 2.3				
33 Warp allocation granularity (for register allocation) 2 44 Maximum Thread Block Size 512 56 Allocation Per Thread Block 8 57 Warps 8 58 Ausimum Thread Blocks 8 59 Allocation granularity (for register allocation) 4096 50 Ausimum Thread Blocks Per Multiprocessor Blocks 51 Limited by Max Warps / Blocks per Multiprocessor Blocks 51 Limited by Shared Memory per Multiprocessor Blocks 51 Limited by Shared Memory per Multiprocessor Blocks 61 CUDA Occupancy Calculator 2 74 Version: 2.3 51 Copyright and License 2.3				
Maximum Thread Block Size 512 Allocation Per Thread Block 8 Warps 8 Begisters 4096 Othese data are used in computing the occupancy data in blue 4096 Imited by Max Warps / Blocks per Multiprocessor Blocks Imited by Max Warps / Blocks per Multiprocessor Blocks Imited by Shared Memory Per Multiprocessor 4 Imited by Shared Memory Per Block 5 Imited by Shared Memory Per Bl			Varving Shared Memory Usage	
66 Allocation Per Thread Block 7 Warps 8 8 Registers 4096 99 Shared Memory 4096 40 These data are used in computing the occupancy data in blue 40 41 Imited by Max Warps / Blocks per Multiprocessor Blocks 13 Limited by Shared Memory per Multiprocessor Blocks 14 Limited by Shared Memory per Multiprocessor 40 16 Thread Block Limit Per Multiprocessor 10 16 CUDA Occupancy Calculator 2.3 19 Version: 2.3 10 Copyright and License 5		512		
37 Warps 8 38 Registers 4096 39 Shared Memory 4096 40 These data are used in computing the occupancy data in blue 4096 41 Maximum Thread Blocks Per Multiprocessor Blocks 13 Limited by Max Warps / Blocks per Multiprocessor 6 14 Limited by Shared Memory per Multiprocessor 4 14 Limited by Shared Memory per Multiprocessor 4 16 Thread Block Limit Per Multiprocessor 4 17 Thread Block Limit Per Multiprocessor 4 19 Version: 2.3 10 Copyright and License 2.3 11 Shared Memory Per Block			48	
38 Registers 4096 39 Shared Memory 4096 30 These data are used in computing the occupancy data in blue 41 Aximum Thread Blocks Per Multiprocessor Blocks 42 Maximum Thread Blocks per Multiprocessor Blocks 43 Limited by Registers per Multiprocessor Flocks 44 Limited by Registers per Multiprocessor 40 45 Limited by Shared Memory per Multiprocessor 40 46 Thread Block Limit Per Multiprocessor 40 47 CuDA Occupancy Calculator 23 48 CUDA Occupancy Calculator 23 49 Version: 23 51 Copyright and License 51		8		
39 Shared Memory 4096 10 These data are used in computing the occupancy data in blue 11 12 Maximum Thread Blocks Per Multiprocessor 13 Limited by Max Warps / Blocks per Multiprocessor 14 Limited by Shared Memory per Multiprocessor 15 Limited by Shared Memory per Multiprocessor 16 Thread Block Limit Per Multiprocessor 16 Thread Block Limit Per Multiprocessor 17 18 CUDA Occupancy Calculator 19 Version: 20 Copyright and License		4096	My Shared	
14 Limited by Registers per Multiprocessor 4 15 Limited by Shared Memory per Multiprocessor 4 16 Thread Block Limit Per Multiprocessor 4 17 RED 0 18 CUDA Occupancy Calculator 0 19 Version: 2.3 10 Copyright and License 11 5			2 32 - Memory 4096	
4 Limited by Registers per Multiprocessor 4 5 Limited by Shared Memory per Multiprocessor 4 6 Thread Block Limit Per Multiprocessor 4 7 Total Block Limit Per Multiprocessor 4 8 CDA Occupancy Calculator 9 9 Version: 2.3 0 Copyright and License 1 Shared Memory Per Block				
4 Limited by Registers per Multiprocessor 4 5 Limited by Shared Memory per Multiprocessor 4 6 Thread Block Limit Per Multiprocessor 4 7 Total Block Limit Per Multiprocessor 4 8 CDA Occupancy Calculator 9 9 Version: 2.3 0 Copyright and License 1 Shared Memory Per Block		Disala		
14 Limited by Registers per Multiprocessor 4 5 Limited by Shared Memory per Multiprocessor 4 6 Thread Block Limit Per Multiprocessor 4 77 RED 0 18 CUDA Occupancy Calculator 20 19 Version: 2.3 10 Copyright and License		DIOCKS		
15 Limited by Shared Memory per Multiprocessor 4 16 Thread Block Limit Per Multiprocessor highlighted RED 17 CUDA Occupancy Calculator 0 18 CUDA Occupancy Calculator 0 19 Version: 2.3 0 Copyright and License 11 Shared Memory Per Block		4		
47 CUDA Occupancy Calculator 43 CUDA Occupancy Calculator 44 <		4	8	
0 Copyright and License 2.3 01 Copyright and License Shared Memory Per Block	6 Thread Block Limit Per Multiprocessor highlighted RED			
19 Version: 2.3 Copyright and License 1			0 20 8 1 1 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 1 1 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 4 0 8 2 2 2 2 2 4 0 8 2 2 2 2 4 0 8 2 2 2 2 2 4 0 8 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	
50 Copyright and License Snareo Wentory Per block		2.3	0 0 0 4 0 0 0 0 4 0 0 0 4 0 0 0 4 0	
51		2.3	Shared Memory Per Block	
I I Calculator / Help / GPU Data / Copyright & License /♥⊋ /				
	← → ▶ Calculator / Help / GPU Data / Copyright & License /	2		

Performance Optimization to technische universität

Profiling and final check

- Always profile your kernels to evaluate:
 - Memory access quality
 - kernel Occupancy
 - Each kernel's contribution to the total time
- Use the Compute Visual Profiler to check these

eigenvalues - Compute Visual Profiler -	matrix_size_4096 - Devic	e_0 - Context_0 [CUDA]]								-			
Eile Session View Options Wine	dow <u>H</u> elp												- 8
🗋 🍙 🔚 🗐 🕨 × 🚦	1 🖬 📶 💻 🚺	E											
ssions 6	× Profiler Output	GPU Time Summary P	lot 🔀 GPI	U Time Height Plot	t 🗙								
matrix_size_512 Device_0 Context 0 [CUDA]						Summ	ary Plot						ſ
matrix_size_2048		0.00%	Time (Total) 6 5.50%	11.00%	16.51%	22.01%	27.51%	33.01%	38.52%	44.02%	49.52%	55.02%	60.52%
Context_0 [CUDA] matrix size 4096	bisectKernelLarge_M bisectKernelLarge_C	neIntervals (100)											
Device_0 Context_0 [CUDA]	m	KernelLarge (100) eemcpyDtoH (207) memcpyHtoD (15)											
		0.00%	6 5.50%	11.00%	16.51%	22.01%	27.51%	33.01%	38.52%	44.02%	49.52%	55.02%	60.52%
	<					III							Þ
put													8
trix size 2048 - Device 0 - Context 0 [CUDA	1 : Profiler table column 'blo	ck size Y' having all 1 value	s is hidden.										
titix_size_2048 - Device_0 - Context_0 [CUDA trix_size_4096 - Device_0 - Context_0 [CUDA	 Profiler table column 'dy Profiler table column 'str Profiler table column 'loc Profiler table column 'loc Profiler table column 'me 	namic shared memory per l eam id' having all zero valu al block size' having all zero m transfer size (bytes)' ha	block' having all : ies is hidden. values is hidde ving all zero valu	n.	len.								
atrix_size_4096 - Device_0 - Context_0 [CUDA atrix_size_4096 - Device_0 - Context_0 [CUDA atrix_size_4096 - Device_0 - Context_0 [CUDA atrix_size_4096 - Device_0 - Context_0 [CUDA	 Profiler table column 'loc) Profiler table column 'tex) Profiler table column 'tex) Profiler table column 'gri 	al store' having all zero val < cache hit' having all zero < cache miss' having all zero d size Y' having all 1 values	ues is hidden. values is hidden. o values is hidde : is hidden.										
atrix_size_4096 - Device_0 - Context_0 [CUDA atrix_size_4096 - Device_0 - Context_0 [CUDA													





Thank You





Amin Safi | TU Dortmund

Programming On GPUs

CUDA Technology

Introduced to market by nVIDIA in 2006

- An Integrated computational architecture to exploit all the computational resource of GPUs.
- Comes with a compiler based on C and other scientific languages.
- Enables computing on low price, small GPUs personal Super Computer







technische universität

dortmund

Page