HETEROGENEOUS **CPU+GPU** COMPUTING

Ana Lucia Varbanescu – University of Amsterdam
a.l.varbanescu@uva.nl
Stijn Heldens – Twente University
mail@stijnh.nl

Significant contributions by: **Pieter Hijma** (UvA, NL),
Jie Shen (TUDelft, NL), **Basilio Fraguela** (A Coruna University, ESP)

PART I

Heterogeneous processing: pro’s and con’s

Hardware Performance metrics

- Clock frequency [GHz] = absolute hardware speed
  - Memories, CPUs, interconnects
- Operational speed [GFLOPs]
  - Instructions per cycle + frequency
- Memory bandwidth [GB/s]
  - differs a lot between different memories on chip
- Power [Watt]
- Derived metrics
  - FLOP/Byte, FLOP/Watt

Theoretical peak performance

Peak = chips * cores * threads/core * vector_lanes * FLOPs/cycle * clockFrequency

- Some examples:
  - Intel Core i7 CPU

    2 chips * 4 cores * 4-way vectors * 2 FLOPs/cycle * 2.4 GHz = **154 GFLOPs**
  - NVIDIA GTX 580 GPU

    1 chip * 16 SMs * 32 cores * 2 FLOPs/cycle * 1.544 GhZ = **1581 GFLOPs**

Performance ratio (CPU:GPU): 1:10 !!!

DRAM Memory bandwidth

Bandwidth = memory bus frequency * bits per cycle
* bus width

- Memory clock != CPU clock!
- In bits, divide by 8 for GB/s

- Some Examples:
  - Intel Core i7 DDR3: 1.333 * 2 * 64 = **21 GB/s**
  - NVIDIA GTX 580 GDDR5: 1.002 * 4 * 384 = **192 GB/s**

Performance ratio (CPU:GPU): 1:8 !!!

Power

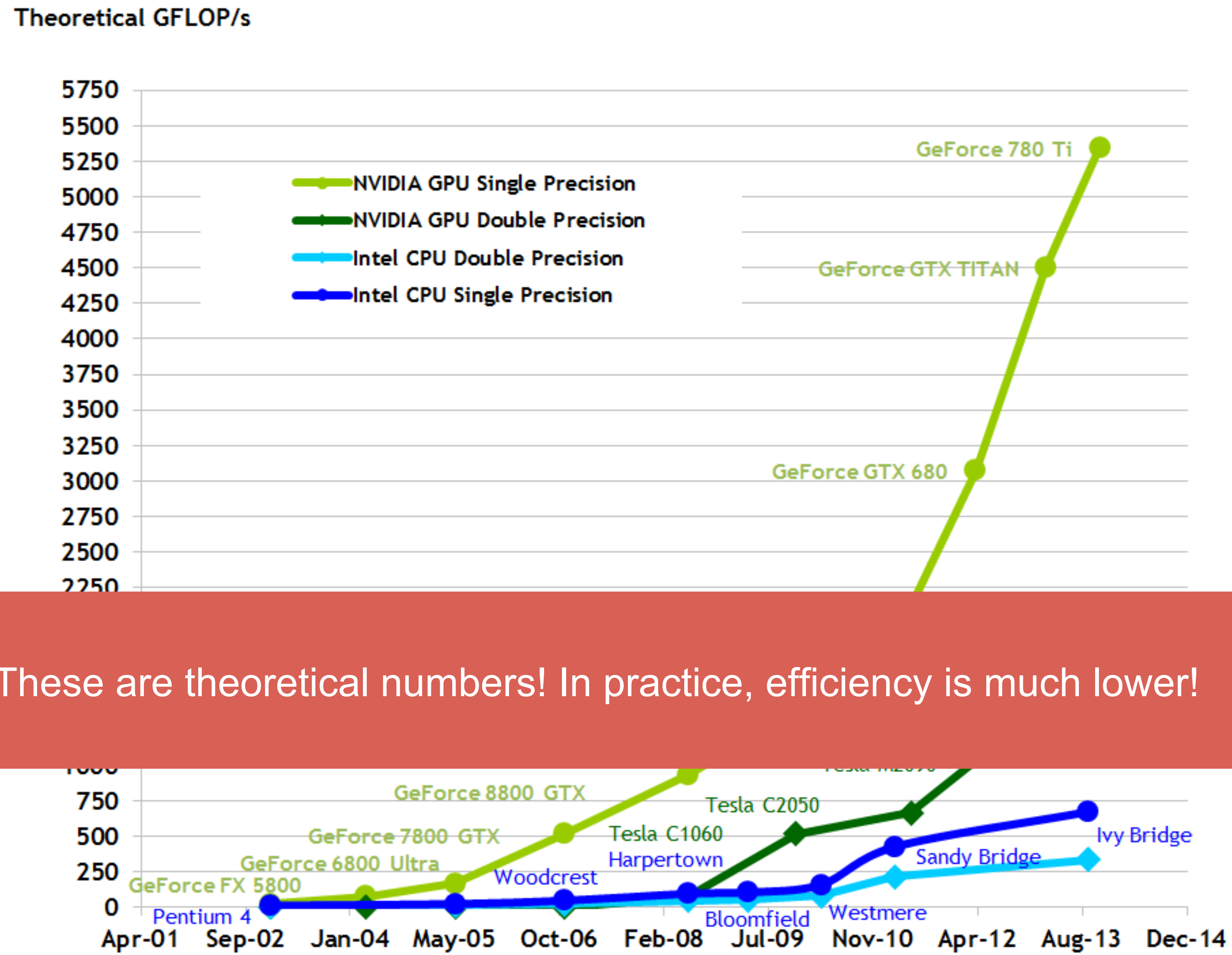
- Chip manufactures specify Thermal Design Power (TDP)
- We can measure dissipated power
  - Whole system
  - Typically (much) lower than TDP
- Power efficiency
  - FLOPS / Watt
- Examples (with theoretical peak and TDP)
  - Intel Core i7: 154 / 160 = **1.0 GFLOPs/W**
  - NVIDIA GTX 580: 1581 / 244 = **6.3 GFLOPs/W**
  - ATI HD 6970: 2703 / 250 = **10.8 GFLOPs/W**

Summary

	Cores	Threads/ALUs	GFLOPS	Bandwidth
Sun Niagara 2	8	64	**11.2**	**76**
IBM BG/P	4	8	**13.6**	**13.6**
IBM Power 7	8	32	**265**	**68**
Intel Core i7	4	16	**85**	**25.6**
AMD Barcelona	4	8	**37**	**21.4**
AMD Istanbul	6	6	**62.4**	**25.6**
AMD Magny-Cours	12	12	**125**	**25.6**
Cell/B.E.	8	8	**205**	**25.6**
NVIDIA GTX 580	16	512	**1581**	**192**
NVIDIA GTX 680	8	1536	**3090**	**192**
AMD HD 6970	384	1536	**2703**	**176**
AMD HD 7970	32	2048	**3789**	**264**
Intel Xeon Phi 7120	61	240	**2417**	**352**

GPU vs. CPU performance

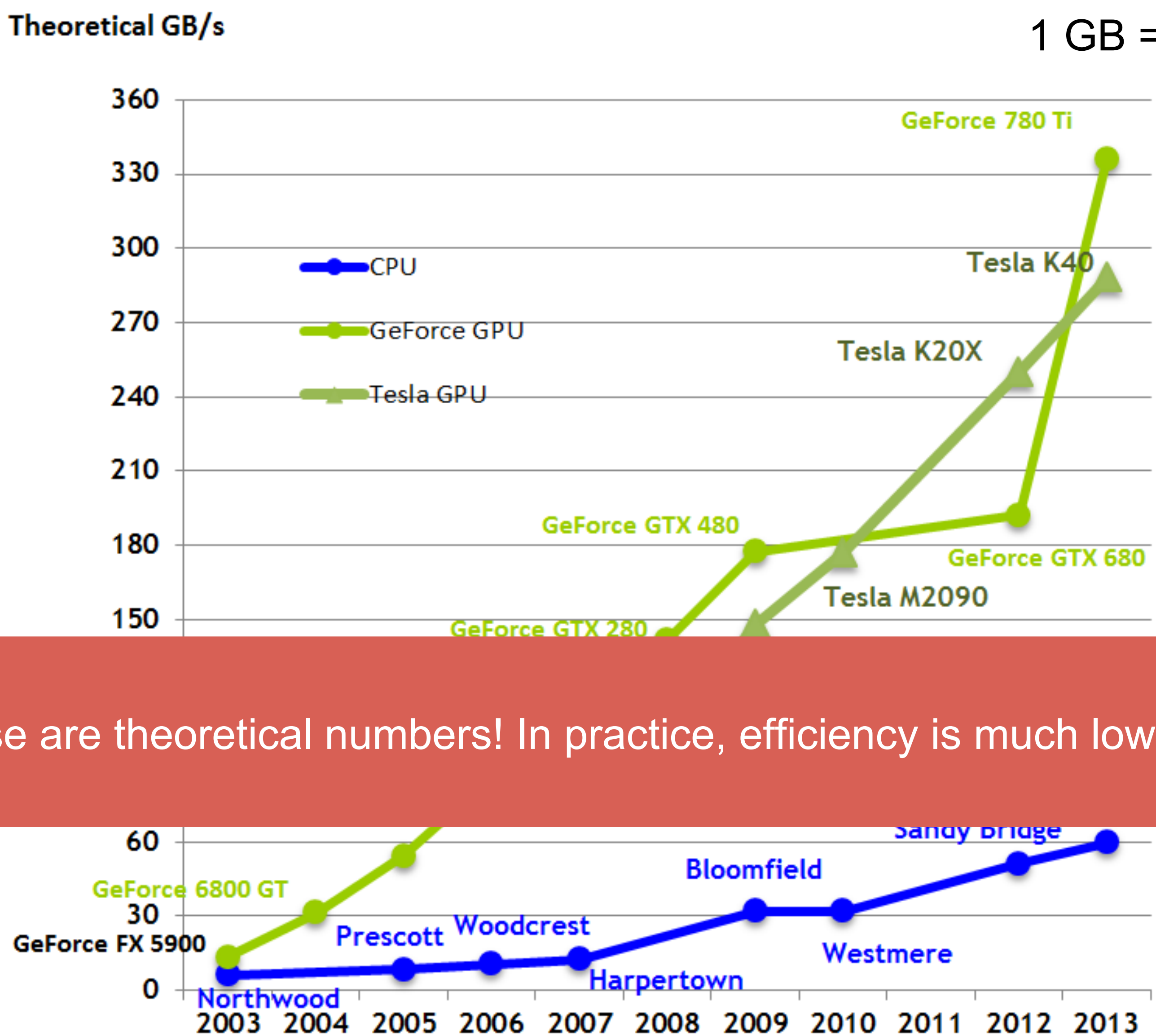
1 GFLOP = 10^9 ops



These are theoretical numbers! In practice, efficiency is much lower!

GPU vs. CPU performance

1 GB = 8 x10^9 bits



These are theoretical numbers! In practice, efficiency is much lower!

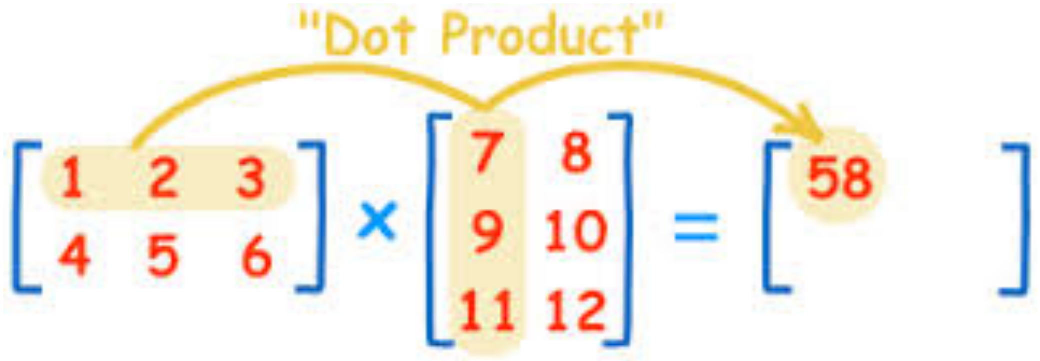
Heterogeneity vs. Homogeneity

- Increase performance
  - Both devices work in parallel
    - Gain is much more than 10%
  - Decrease data communication
    - Which is often the bottleneck of the system
  - Different devices for different roles
- Increase flexibility and reliability
  - Choose one/all *PUs for execution
  - Fall-back solution when one *PU fails
- Increase power efficiency
- Cheaper per flop

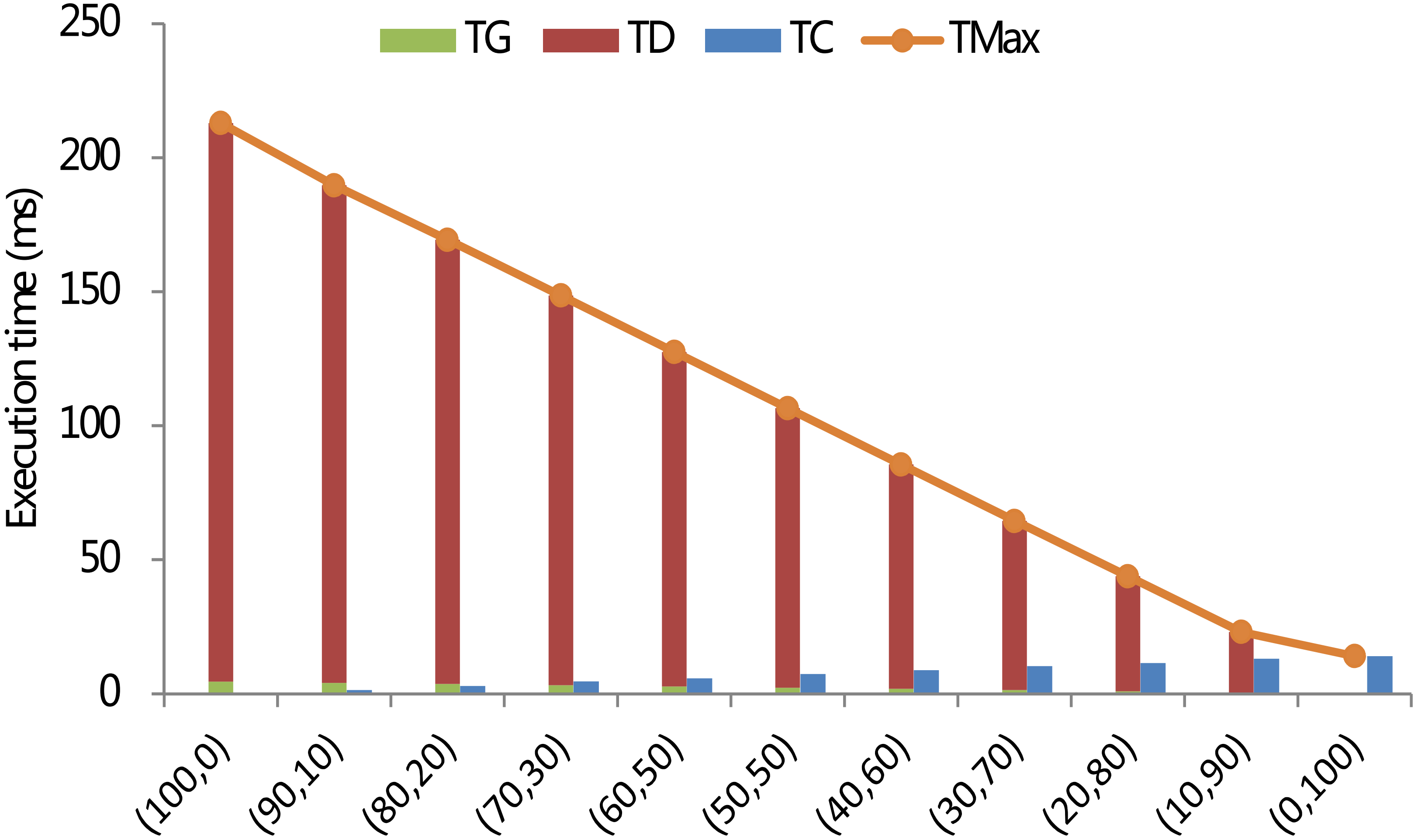
Example 1: dot product

- Dot product
  - Compute the dot product of 2 (1D) arrays
- Performance
  - T_G = execution time on GPU
  - T_C = execution time on CPU
  - T_D = data transfer time CPU-GPU

- GPU best or CPU best?

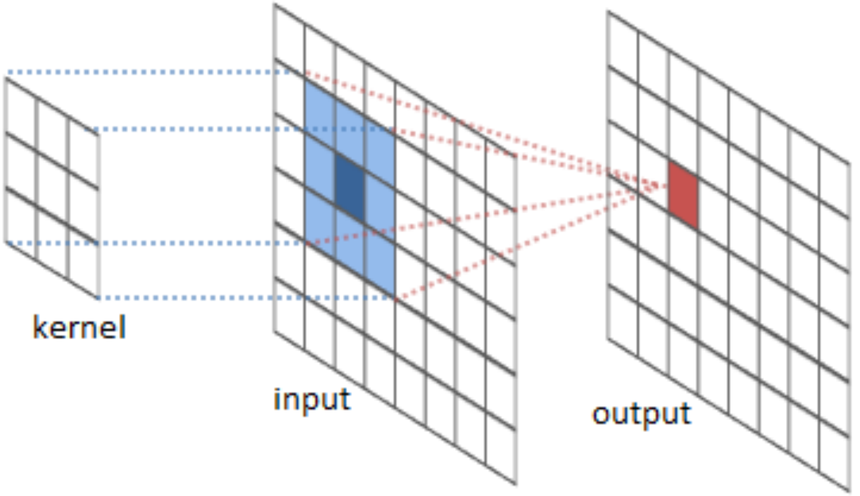


Example 1: dot product

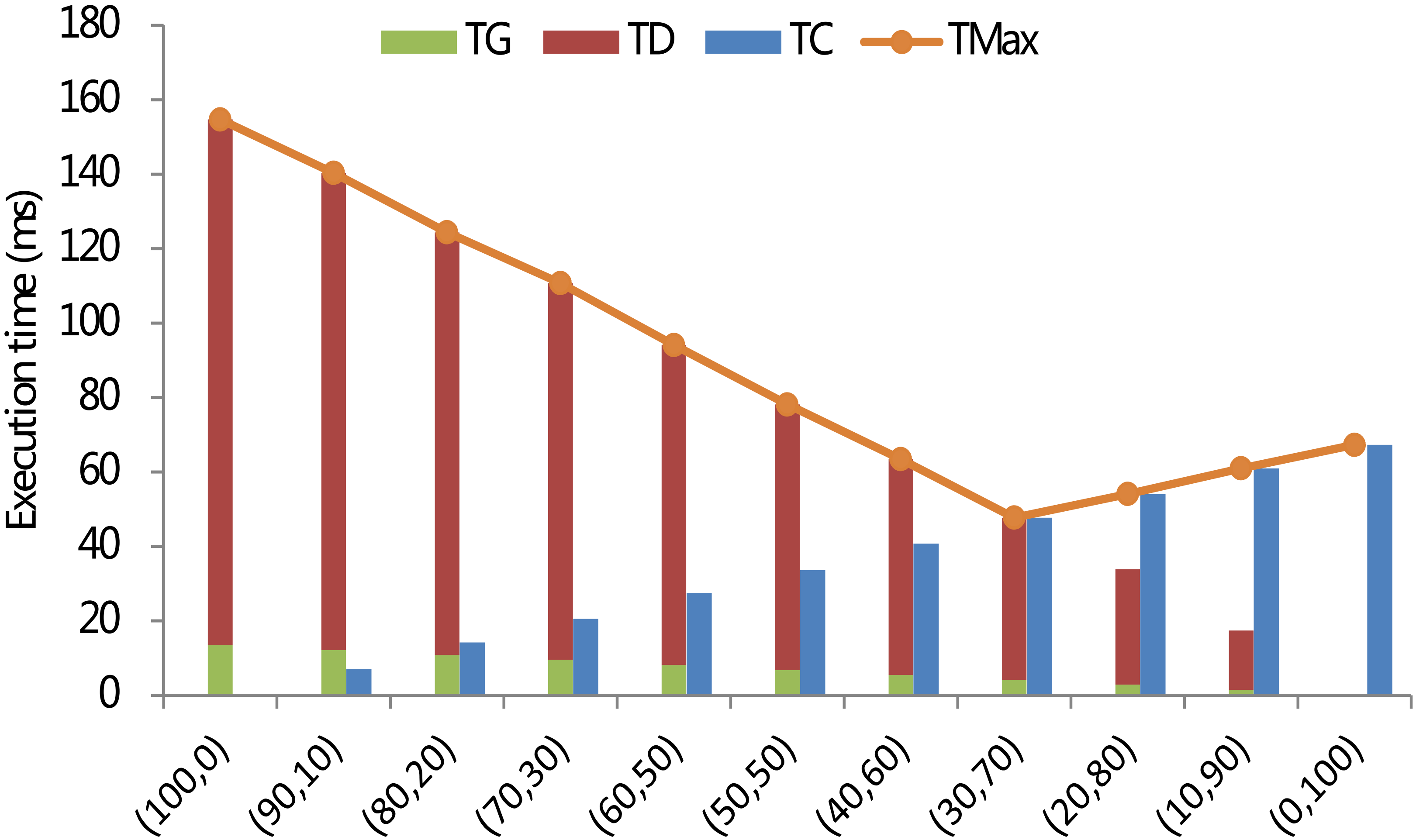


Example 2: separable convolution

- Separable convolution (CUDA SDK)
  - Apply a convolution filter (kernel) on a large image.
  - Separable kernel allows applying
    - Horizontal first
    - Vertical second
- Performance
  - T_G = execution time on GPU
  - T_C = execution time on CPU
  - T_D = data transfer time
- GPU best or CPU best?



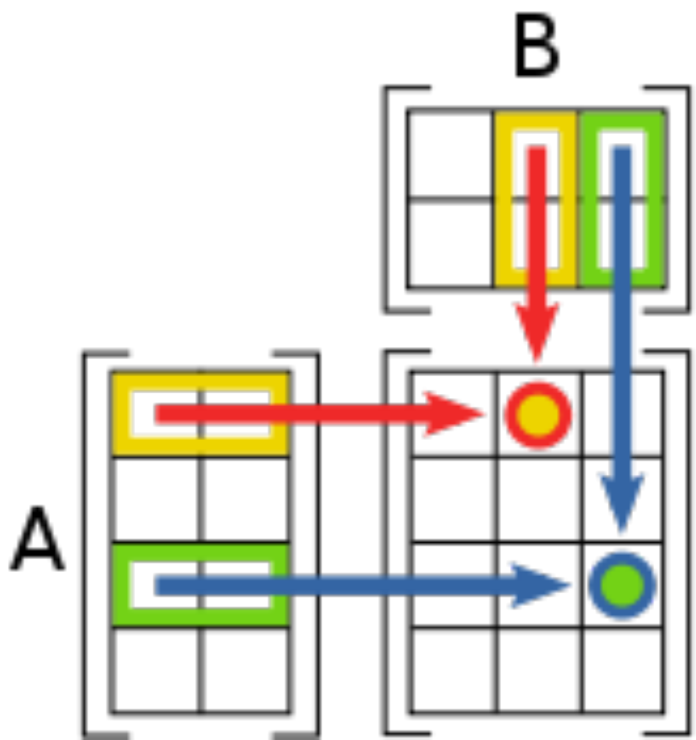
Example 2: separable convolution



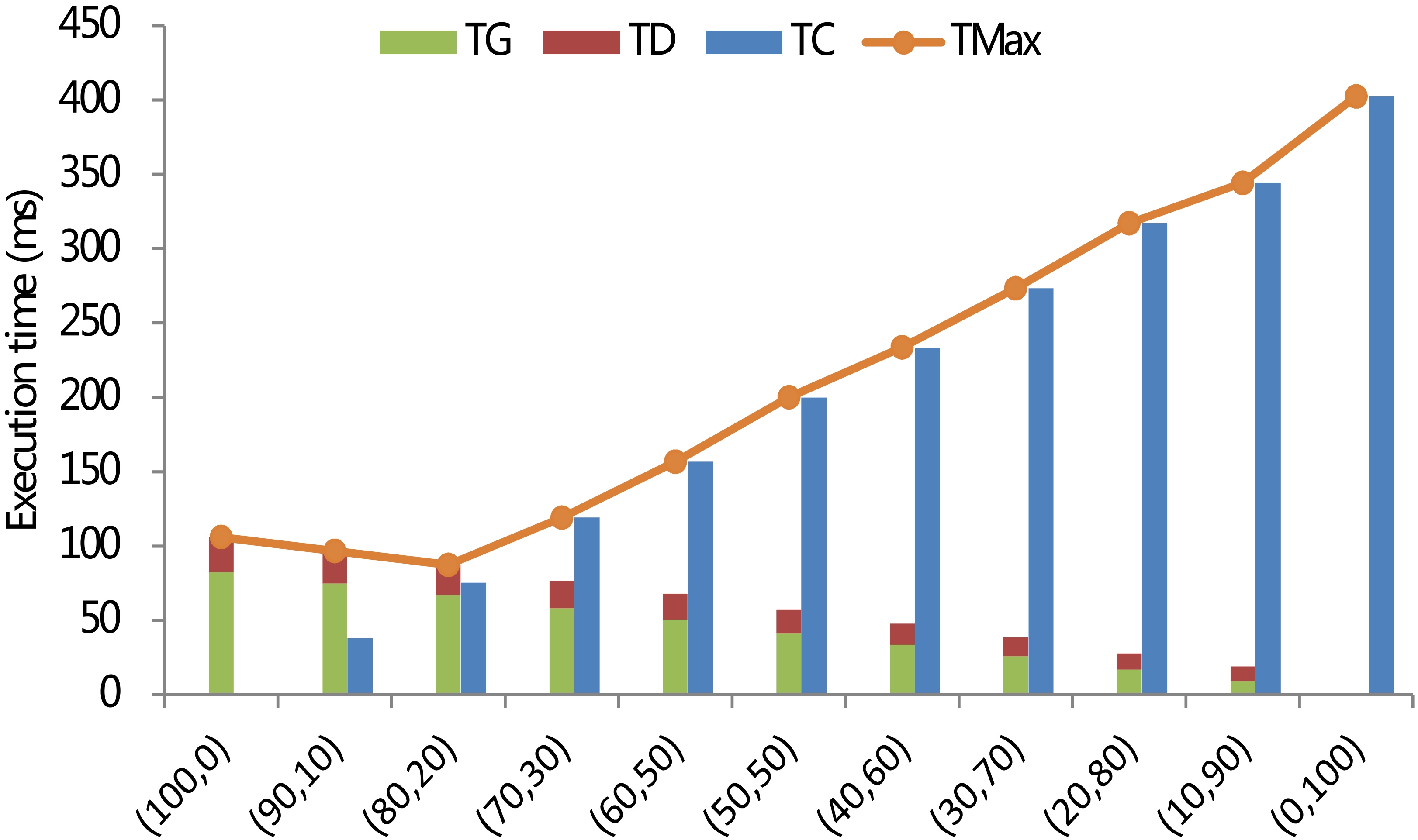
Example 3: matrix multiply

- Matrix multiply
  - Compute the product of 2 matrices
- Performance
  - T_G = execution time on GPU
  - T_C = execution time on CPU
  - T_D = data transfer time CPU-GPU

- GPU best or CPU best?

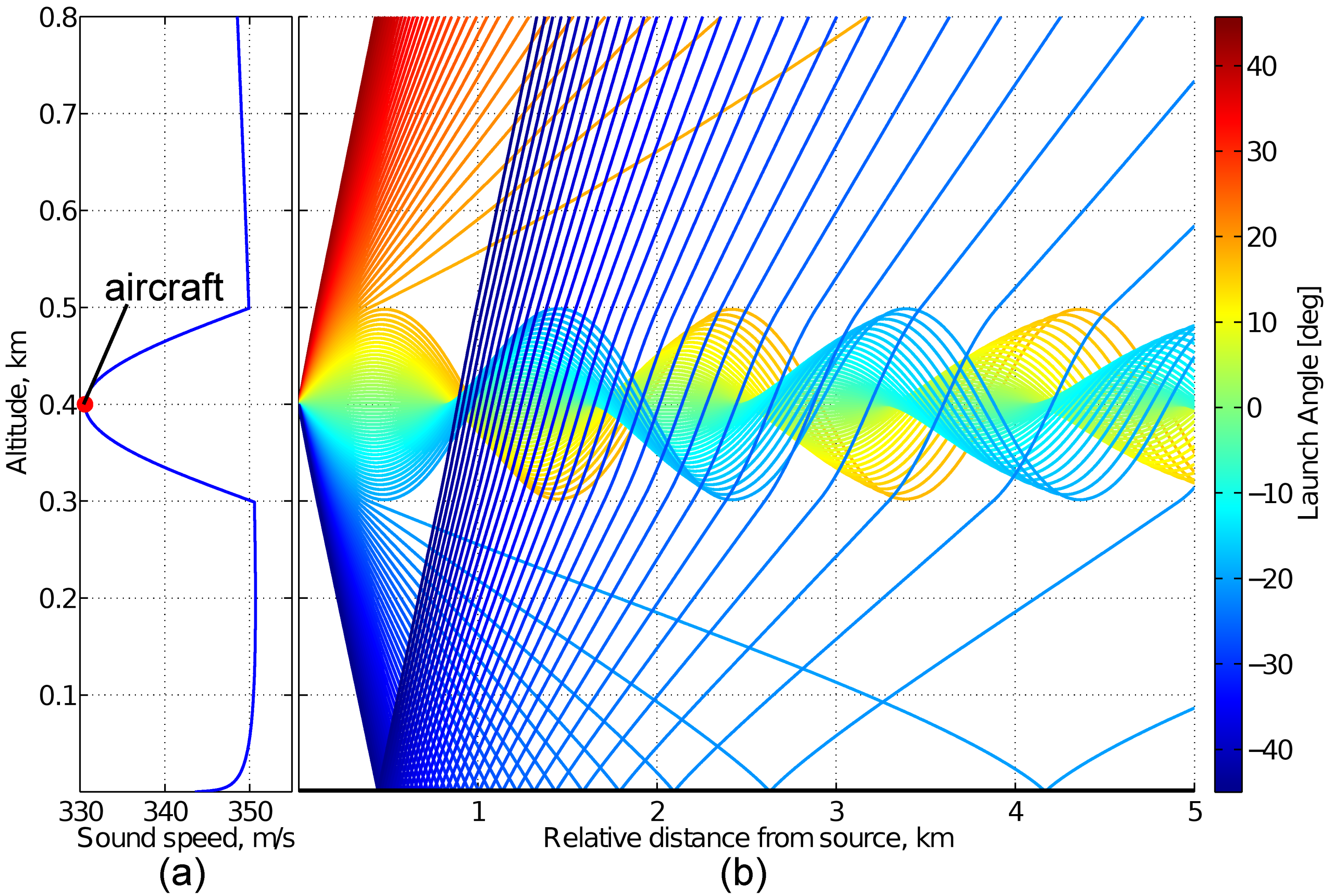


Example 3: matrix multiply



Example 4: Sound ray tracing

Example 4: Sound ray tracing

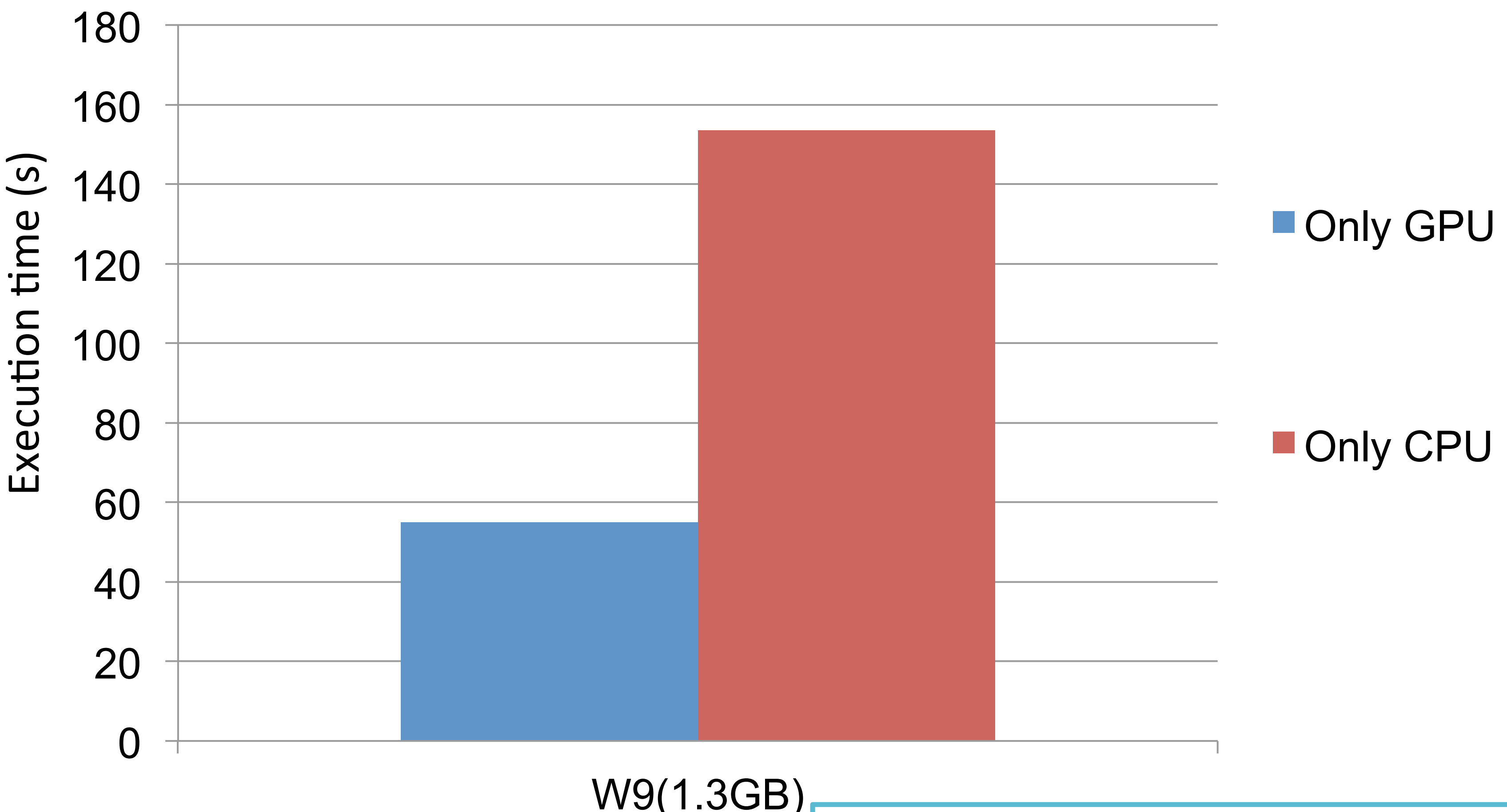


Which hardware?

- Our application has …
- Massive data-parallelism …
- No data dependency between rays …
- Compute-intensive per ray …

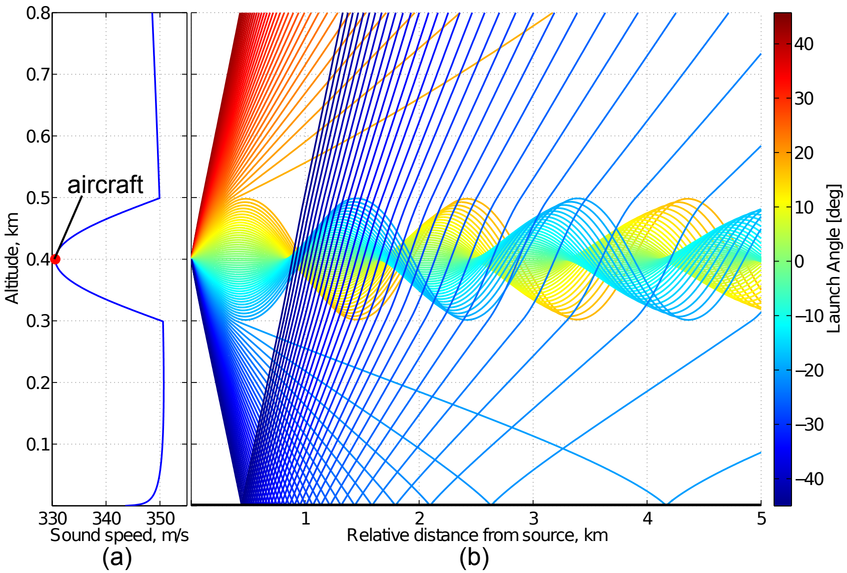
- … clearly, this is a perfect GPU workload !!!

Results [1]

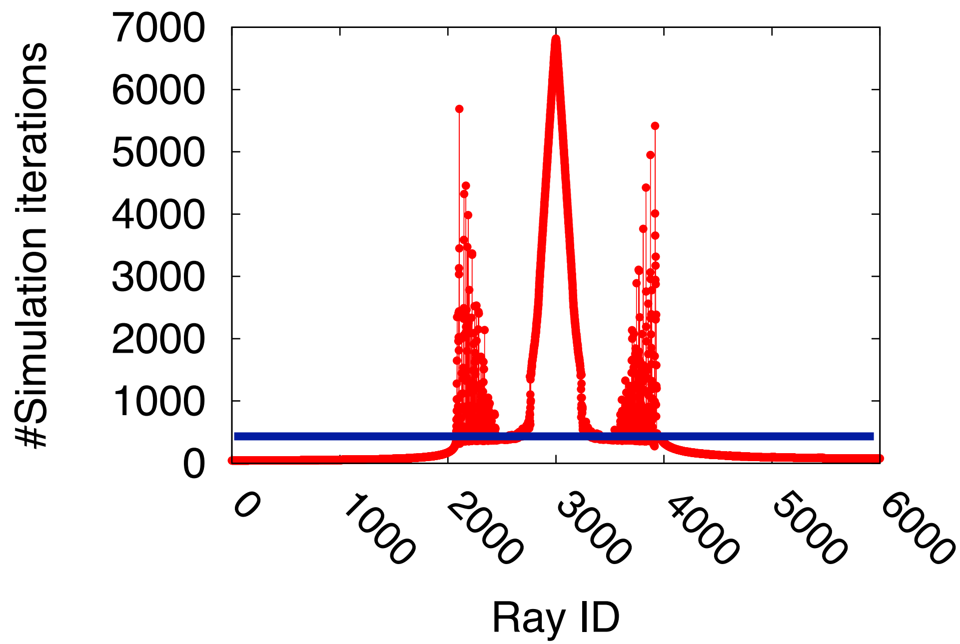


Only 2.2x performance improvement!
We expected 100x …

Workload profile

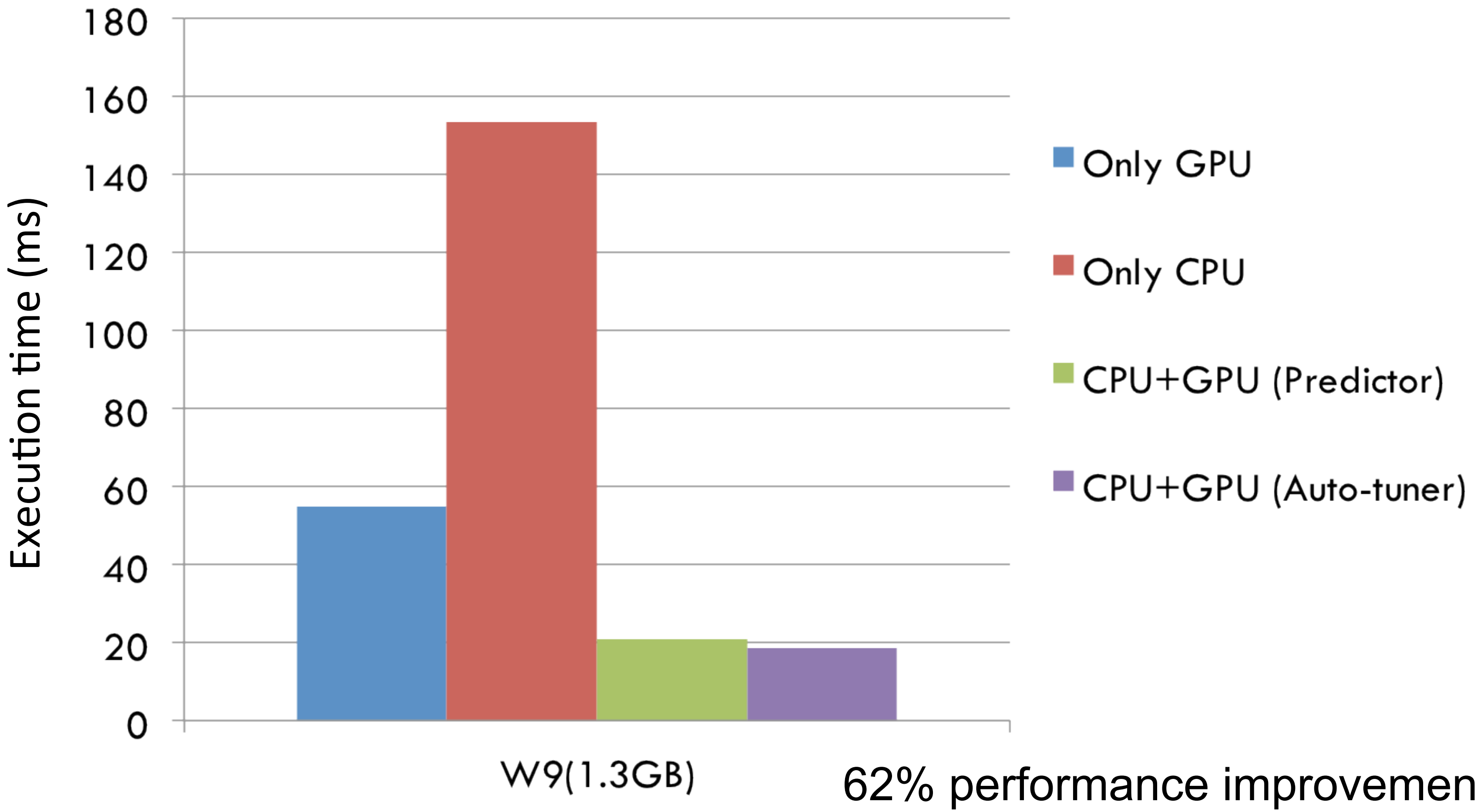


Peak
Processing iterations: ~7000



Bottom
Processing iterations: ~500

Results [2]



62% performance improvement compared to “Only-GPU”

So …

- There are **very** few GPU-only applications
  - CPU – GPU communication bottleneck.
  - Increasing performance of CPUs
- A part of the computation can be done by the CPU.
  - How to program an application to enable this?
  - Which part?

Main challenges: **programming** and **workload partitioning!**

PART III

Challenge 2: Programming

Programming models (PMs)

- Heterogeneous computing = a mix of different processors on the same platform.
- Programming
  - Single programming model (unified)
    - OpenCL is a popular choice
  - Mix of programming models
    - One(/several?) for CPUs – OpenMP
    - One(/several?) for GPUs – CUDA

Low level

NVIDIA CUDA

OpenCL

High level

OpenACC Directives for Accelerators

OpenMP 4.0

Heterogeneous Programming Library

BSC OmpSs

Heterogeneous computing PMs

- CUDA + OpenMP/TBB
  - Typical combination for NVIDIA GPUs
  - Individual development per *PU
  - Glue code can be challenging
- OpenCL
  - Functional portability => can be used as a unified model
  - Performance portability via code specialization
- HPL
  - Library on top of OpenCL, to automate code specialization

Heterogeneous computing PMs

- StarPU
  - Special API for coding
  - Runtime system for scheduling
- OmpSS
  - C + OpenCL/CUDA kernels
  - Runtime system for scheduling and communication optimization

Heterogeneous computing PMs

- Cashmere
  - Dedicated to Divide-and-conquer solutions
  - OpenCL backend.
- GlassWing
  - Dedicated to MapReduce applications
- TOTEM
  - Graph processing
  - CUDA+Multi-threading
- HyGraph
  - Graph processing
  - Based on CUDA+OpenMP

Heterogeneous Computing PMs

High productivity; not all applications are easy to implement.

Domain and/or application specific. Focus on: productivity and performance

High level

Generic

OpenACC, OpenMP 4.0
OmpSS, StarPU, …
HPL

HyGraph,
Cashmere,
GlassWing

Specific

OpenCL
OpenMP+CUDA

TOTEM

Low level

The most common atm. Useful for performance, more difficult to use in practice

Domain specific, focus on performance. More difficult to use.

End of part III

- Questions ?

PART III

Workload partitioning models