Hybrid Systems for Solving High Performance Computing Problems

HPC Workshop

June 24, 2010, Cetraro, Italy

Janusz Kowalik and Piotr Arłukowicz
j.kowalik@comcast.net
Institute of Informatics
Faculty of Mathematics, Physics and Informatics
University of Gdańsk

Paper and Tutorial

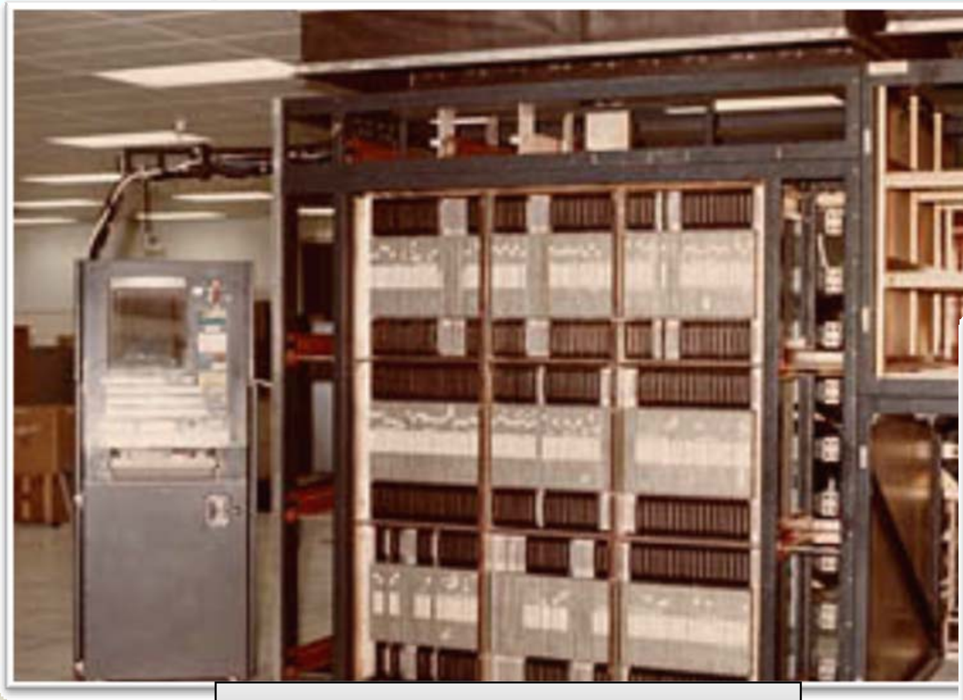
- Paper Focus: **Performance considerations**
- Tutorial focus: **CUDA programming**
- Paper and tutorial complement each other

- Definition
  - Hybrid computing is binomial. It combines sequential processing and highly parallel processing, the SPMT kind.

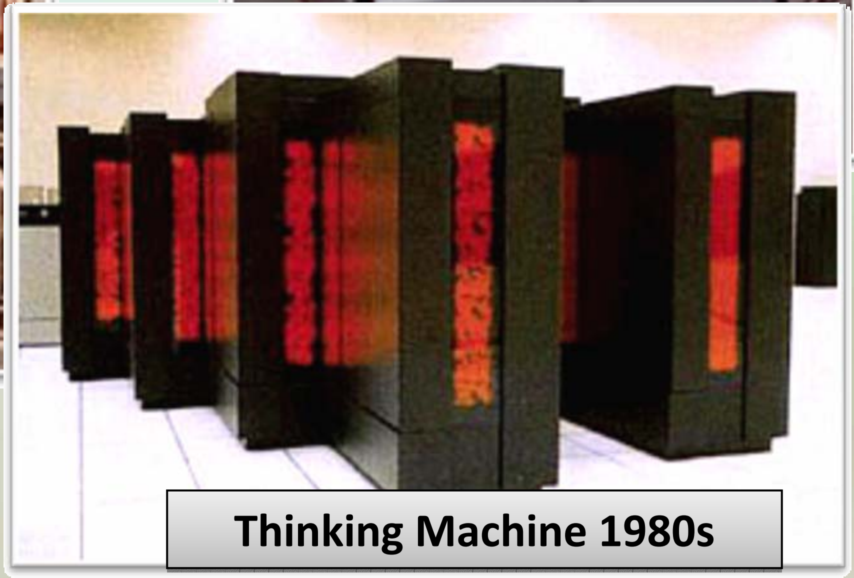
SPMT ≠ SIMD

Data parallel computing

MasPar 1987-92



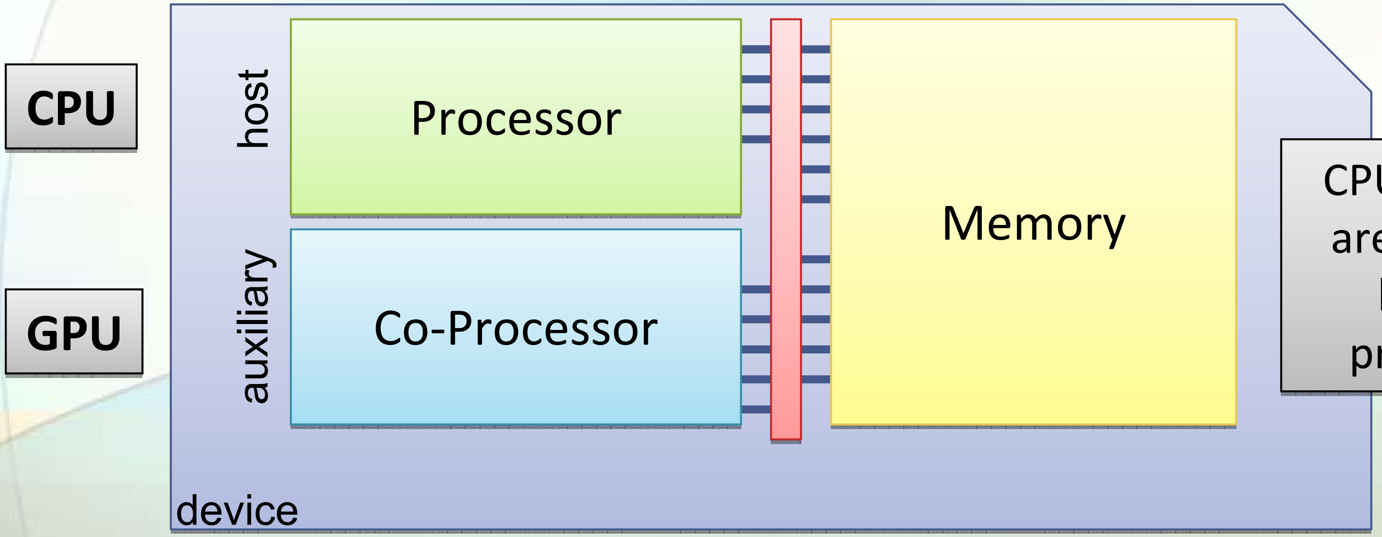
Illiac IV 1966-76



Thinking Machine 1980s

- All not successful. Limited to the SIMD parallelism.
- We needed the hybrid mode of computation.

Hybrid Co-processing



CPU and GPU are different kinds of processors

- CPU executes sequential components.
- Data parallel compute intensive functions are off-loaded to the GPU device.
- For example, body of C for-loops is prime candidate.
- Auxiliary Device = Accelerator

Hybrid Computing around the world.
Are all paths leading to CPU+GPU?

- Seattle based **Cray** announced a supercomputer CX1000 equipped with a **TESLA** GPU accelerator by NVIDIA.

- NVIDIA announced a new enhanced version of CUDA 3.0 for **FERMI** GPUs.

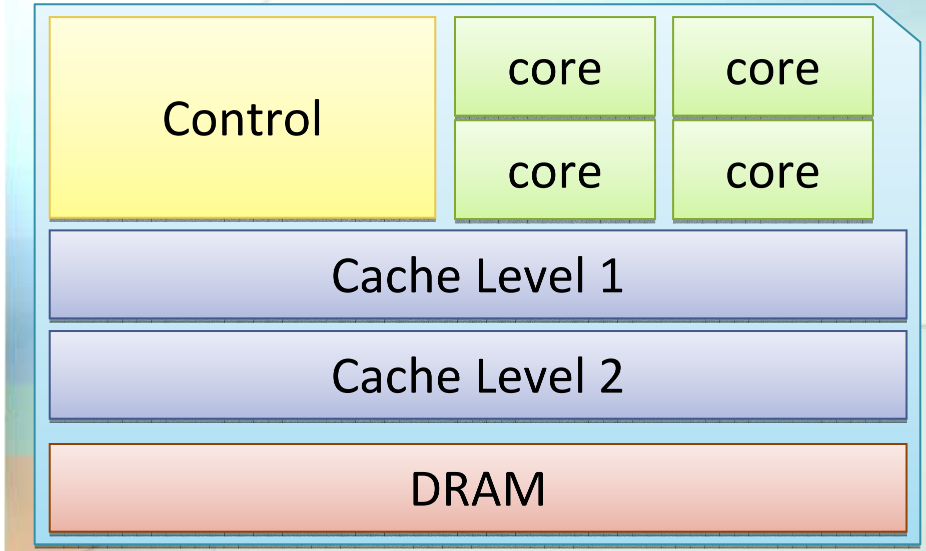
- Jack Dongarra is working on a CUDA accelerated version of **LAPAC**! Essential for HPC.

- The new **IBM server iDataPlex** will be using NVIDIA M2050 or M2070 accelerators.

Continued

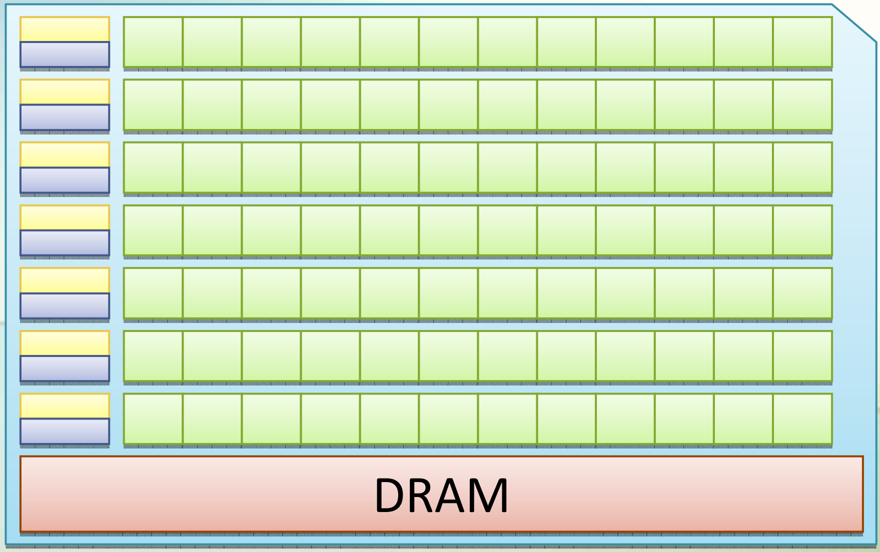
- **TOP500 (June 2010)**
  - **#2** is a Chinese super powered by NVIDIA TESLA C2050 GPUs; Linpack speed 1.27 PFLOPs.
  - **# 7** is also a Chinese hybrid CPU+GPU super with GPUs by AMD.
- **Intel** plans to design a super based on the MIC architecture (Many Integrated Cores). Unlike hybrid architectures the MIC architecture is based on the Intel standard processors.
- **SGI** plans a Petaflop hybrid supercomputer in one cabinet.
- **AMD** is developing CPU+GPU Fusion technology

Comparing CPU with GPU; Silicon use



CPU

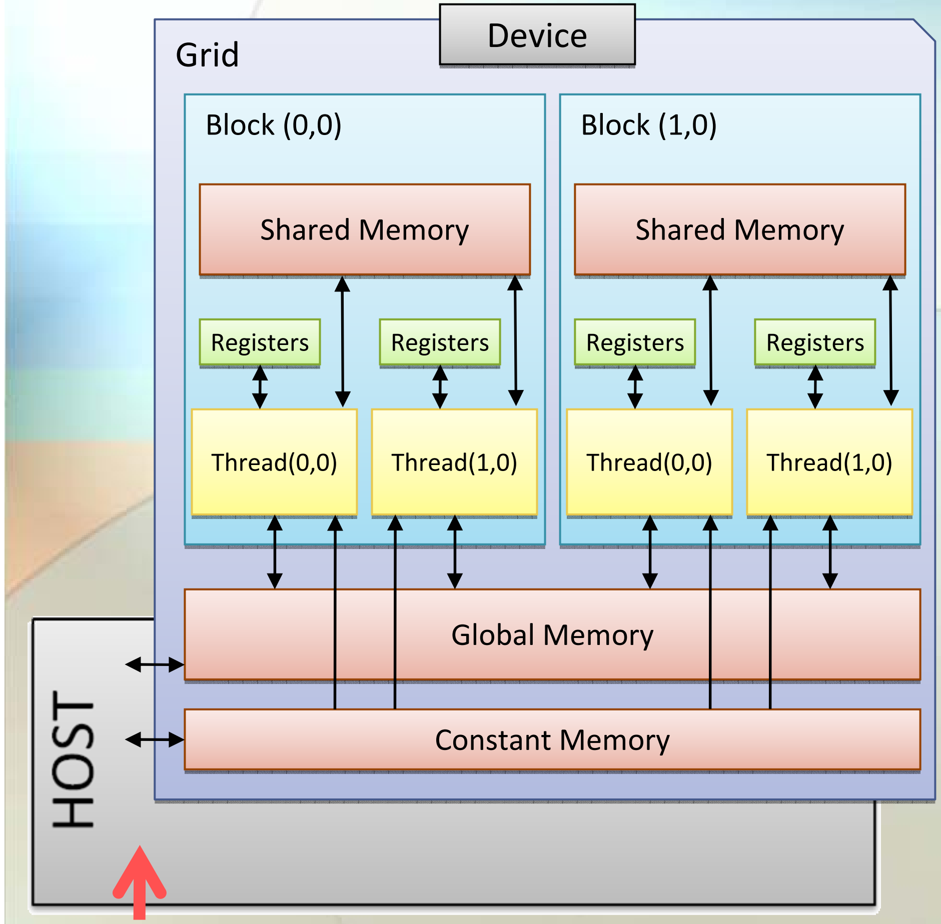
- CPU increases speed of computation by multiple cores and by using cache to reduce memory access.



GPU

- GPU speeds up computation by much higher degree of parallelism.
- More transistors are used for processors.

GPU Memory model



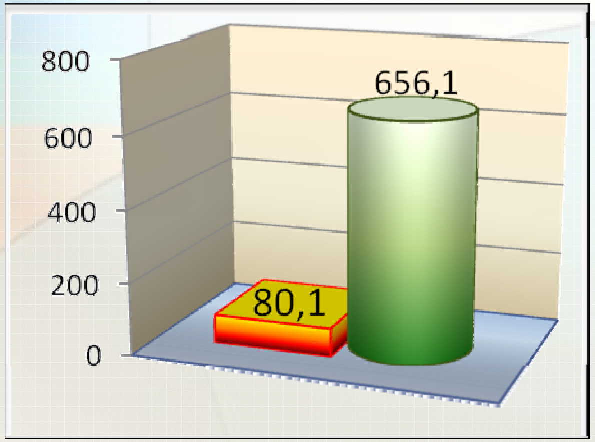
Host-device data transfer

- Each thread has its individual register memory, fast.
- Threads within a block can communicate via shared memory, fast.
- Global memory is largest and slowest.
- Constant memory is read-only. Low latency.
- All the memories have independent disjoint address spaces.

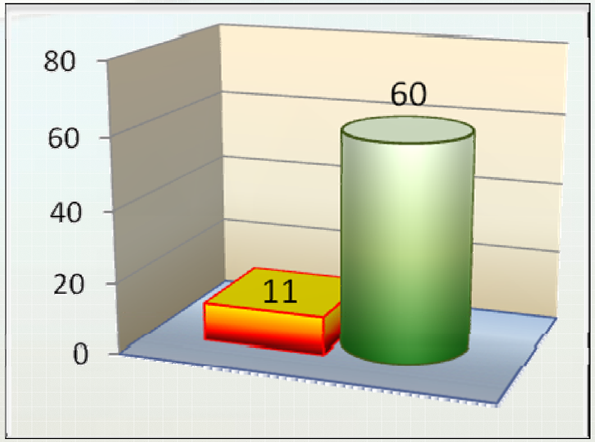
Performance comparison

- Intel processor vs. Intel+TESLA C2060

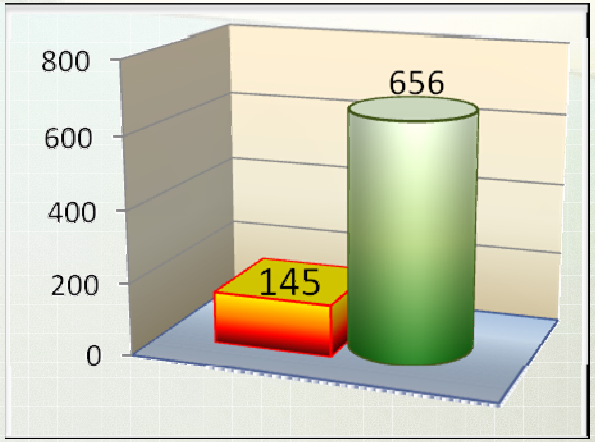
Peak performance
Gflops



Peak performance / $
Gflops / $K



Peak performance / watt
Gflops / kW

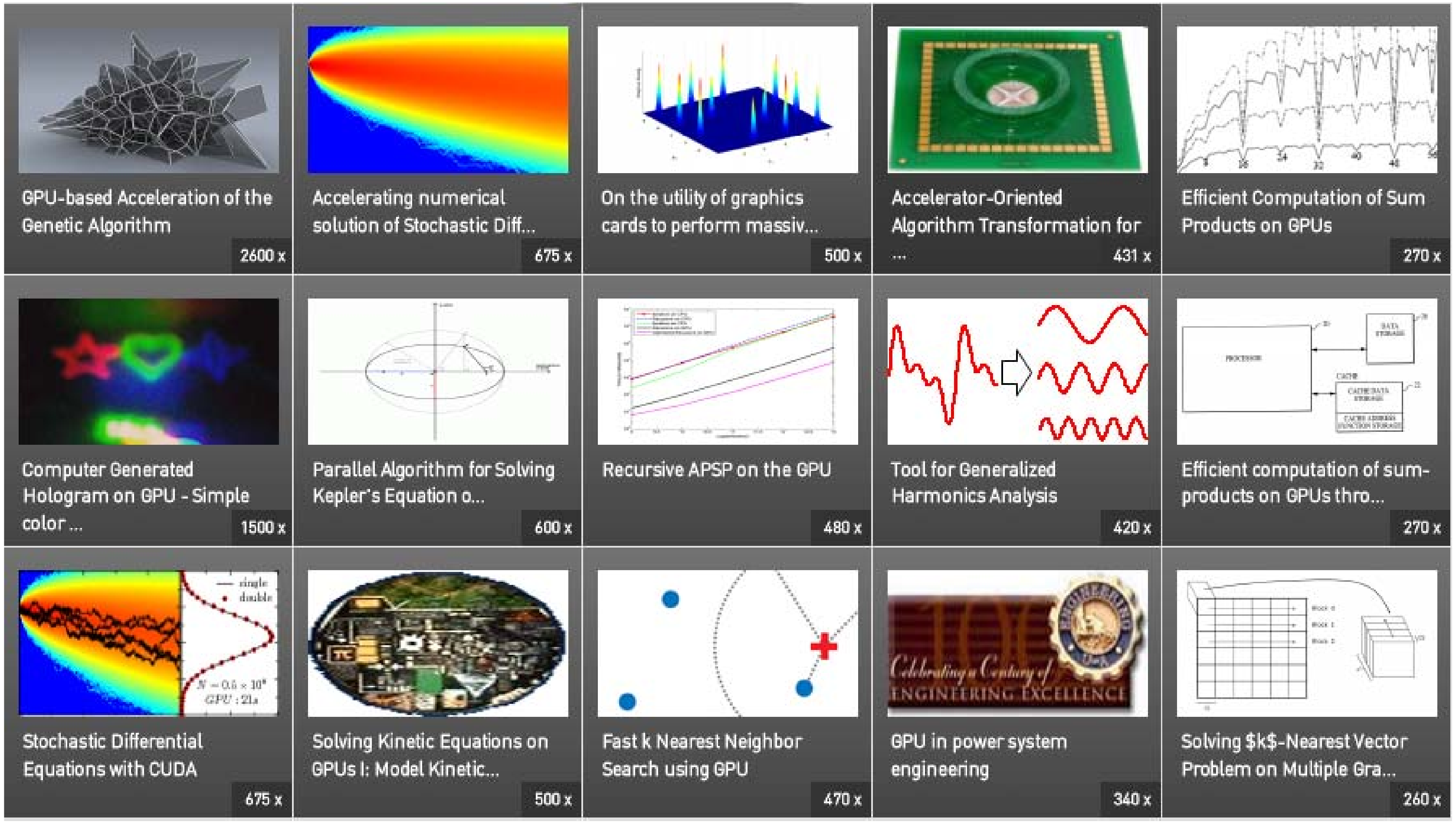


🟥 **CPU 1U server: 2x Intel Xeon X6660 (Nehalem) 2.68 GHz, 48GB memory, $7K, 0.66kW**

🟩 **GPU-CPU 1U server: 2x Tesla C2060 + 2x Intel Xeon X6660, 48GB memory, $11k, 1.0kW**

Source: HPCwire May 2010

Scientific/engineering applications speedups



GPU-based Acceleration of the Genetic Algorithm
2600 x

Accelerating numerical solution of Stochastic Diff...
675 x

On the utility of graphics cards to perform massiv...
500 x

Accelerator-Oriented Algorithm Transformation for ...
431 x

Efficient Computation of Sum Products on GPUs
270 x

Computer Generated Hologram on GPU - Simple color ...
1500 x

Parallel Algorithm for Solving Kepler's Equation o...
600 x

Recursive APSP on the GPU
480 x

Tool for Generalized Harmonics Analysis
420 x

Efficient computation of sum-products on GPUs thro...
270 x

Stochastic Differential Equations with CUDA
675 x

Solving Kinetic Equations on GPUs I: Model Kinetic...
500 x

Fast k Nearest Neighbor Search using GPU
470 x

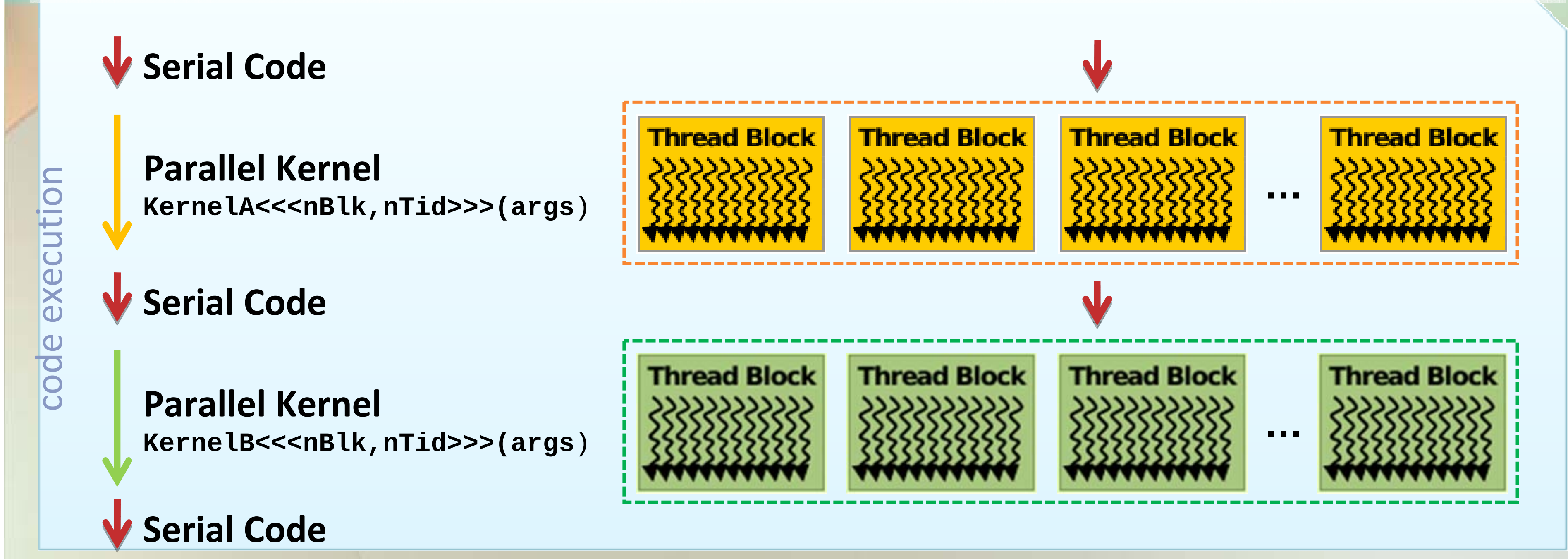
GPU in power system engineering
340 x

Solving k-Nearest Vector Problem on Multiple Gra...
260 x

Source: NVIDIA

Hybrid algorithm execution

- **CUDA** = mixed serial code and parallel kernel, all in C
  - **Serial C** code executes in a CPU thread
  - **Parallel kernel C** code executes in thread blocks across multiple processing elements



Serial Code

Parallel Kernel

```
KernelA<<<nBlk,nTid>>>(args)
```

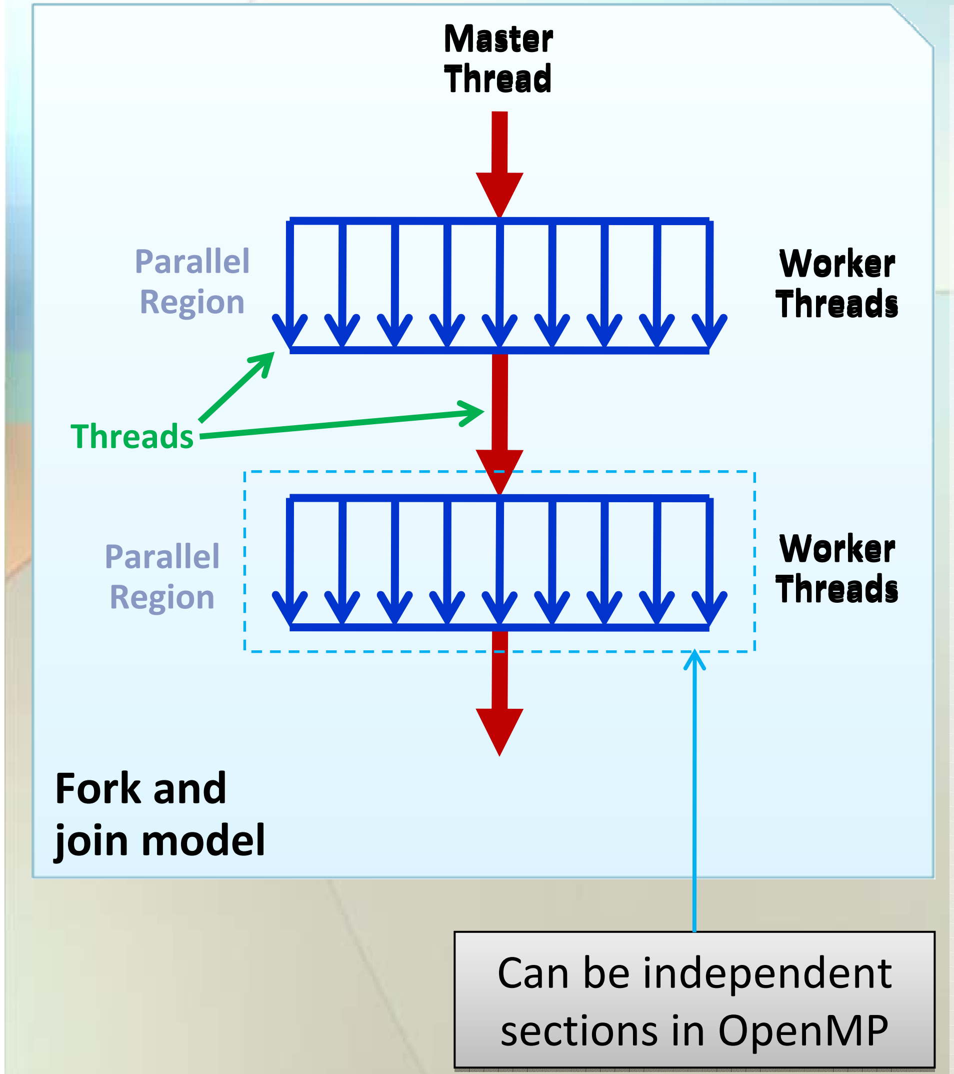
Serial Code

Parallel Kernel

```
KernelB<<<nBlk,nTid>>>(args)
```

Serial Code

CUDA similar to OpenMP



Fork and join model

Can be independent sections in OpenMP

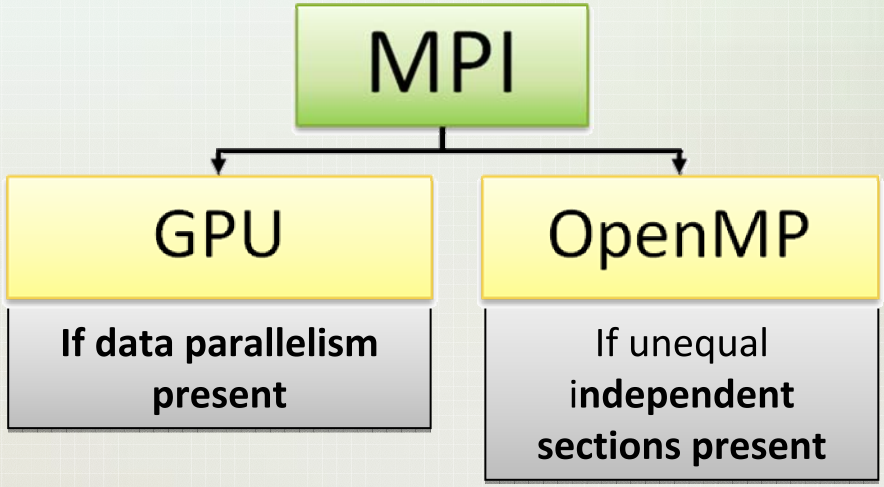
- CUDA is more **scalable to** massive parallelism
- CUDA is similar to OpenMP but **less general**
- CUDA is strictly limited to SPMT
- Single parallel function (kernel)
- Both allow incremental parallelization
- CUDA threads are very **lightweight**.
- **How easy to program?**
- Optimizing performance of CUDA programs is hard, for example Mark Harris' Optimizing Parallel Reduction in CUDA.

Speedup

- Notation:
  - **f -** the parallel fraction
  - **t -** device speed/CPU speed
  - max Speedup = 1 / (1-f)

$$\text{Speedup} \approx \frac{1}{\frac{f}{t} + (1 - f)}$$

- Multilevel Parallel processing



MPI

GPU

If data parallelism present

OpenMP

If unequal **independent sections present**

The hybrid computer at the University of Gdansk

Part	Description
GPU Device	**Tesla C1060 (NVIDIA grant)**
The number of GPU cores	**240**
Cores internal clock	**1.3 GHz**
Global memory	4 GB (4294705152 bytes)
Memory bandwidth	102 GB/sec (peak, dev to dev)
Memory internal clock	800 MHz
CPU host	Intel Core i7 920C
CPU host clock	**2.8 GHz**
Host memory	12 GB PC-1200

Many HPC algorithms are hybrid

- Conjugate **G**radient **M**ethod $x^{i+1} = x^i + \lambda^i p^i$

- The **MRI** problem requires solving large **Ax=b** where A is positive definite.
- The formation of A involves highly parallel matrix/matrix multiplication
  - **Computational complexity: O(n^3)**
- In CGM The most compute intensive operation is the matrix/vector multiplication **Ap**
  - **Computational complexity: O(n^2)**

CGM algorithm

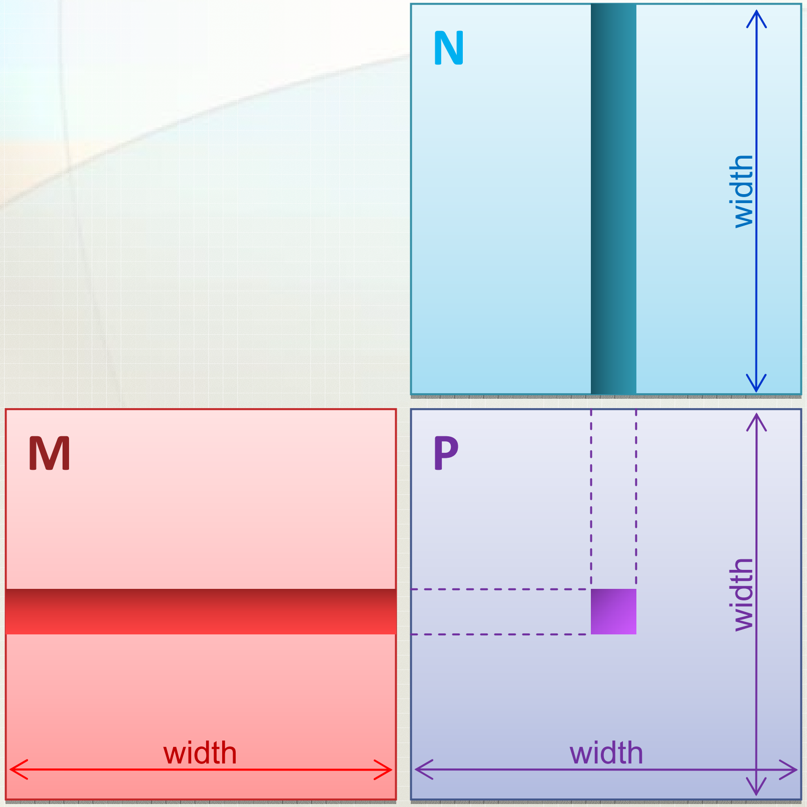
```
($x^{(0)} \in \Re^n$ given)
$x := x^{(0)}$
$r := b - Ax$
$p := r$
$\alpha := ||r||^2$
while $\alpha > tol^2$:
        $\lambda := \alpha / (p^T Ap)$
        $x := x + \lambda p$
        $r := r - \lambda Ap$
        $p := r + (||r||^2/\alpha)p$
        $\alpha := ||r||^2$
end
```

Per iteration: 1 matrix-vector
multiplication
2 dot products
3 scalar-vector
multiplications

Storage: 4 vectors (x, r, p, Ap)

The Example

- A key mathematical computation in MRI
  - Square **matrix/matrix multiplication**

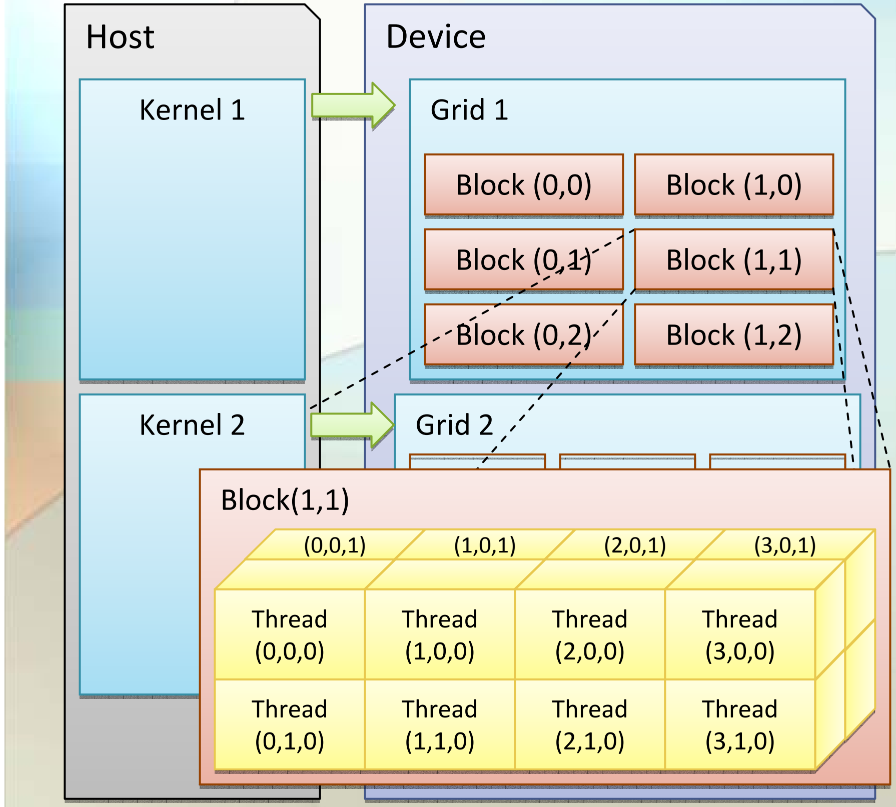


M·N=P

width*width independent dot products

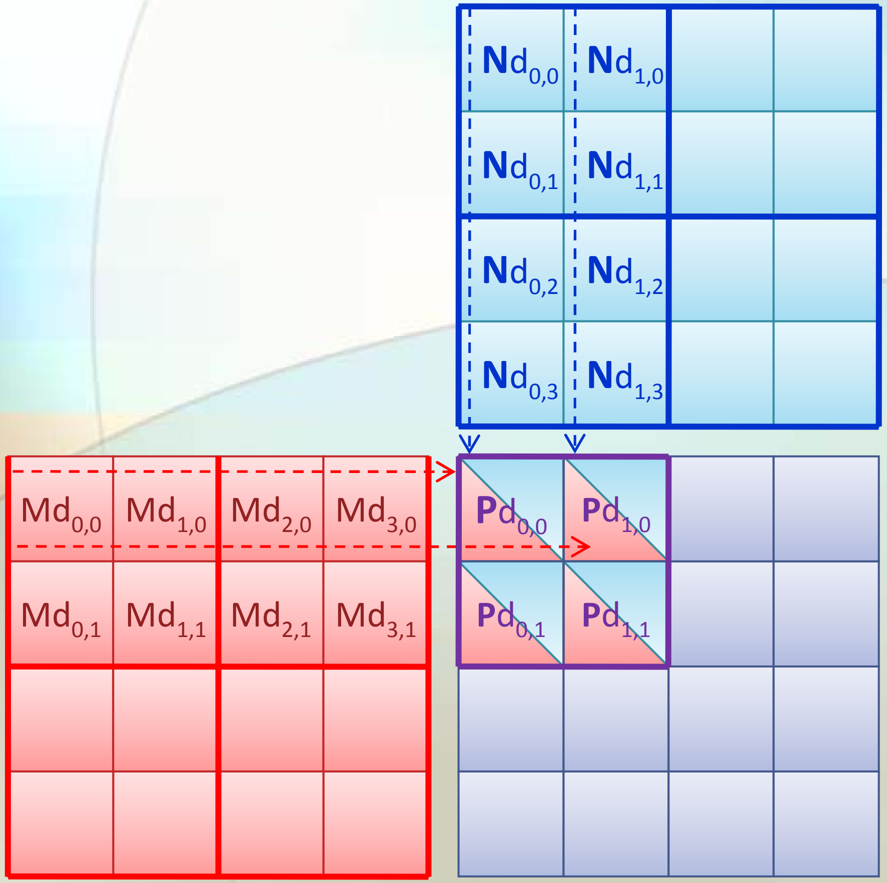
For width=1,000 we may have 1,000,000 parallel threads

Grouping of threads



- Grid contains blocks (1D or 2D).
- Blocks contain threads (1D, 2D or 3D).
- **Limitation:**
  - **Up to 512 threads per one block.**
  - If we use only one block and each element of P is calculated by one thread the largest matrix we can handle is 16x16 since 256 <512.
  - For larger matrices we have to use multiple blocks.

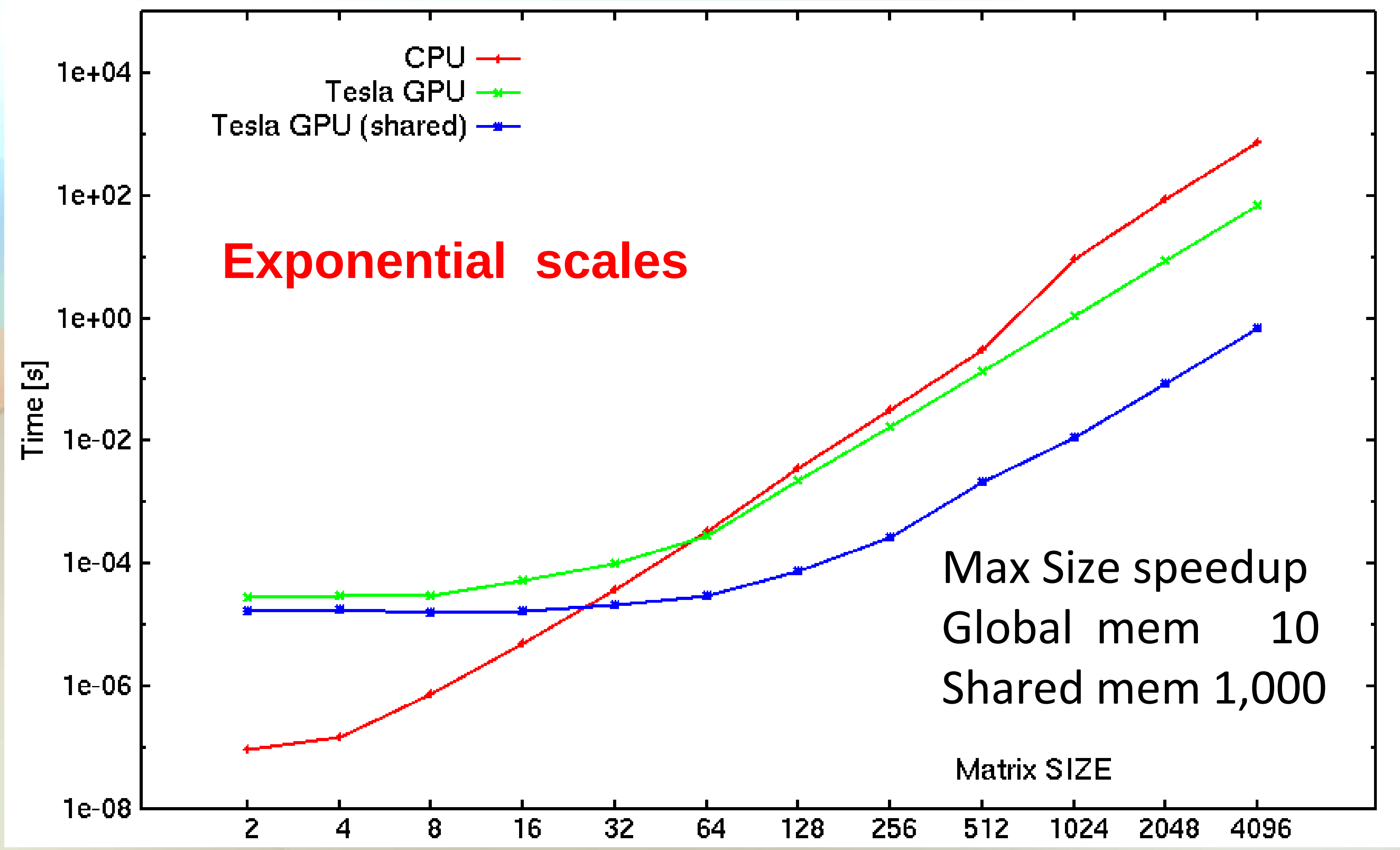
Tiling matrices to utilize shared memory



Multiplying efficiently large matrices requires Tiling

- The matrices are divided into multiple tiles and each tile of P is assigned to one block of threads.
- This way the number of threads per block does not exceed 512 and every element of P is calculated by one thread.
- The size of the tile is chosen so data can fit into the shared memory.

Relative Performance MatMult



Exponential scales

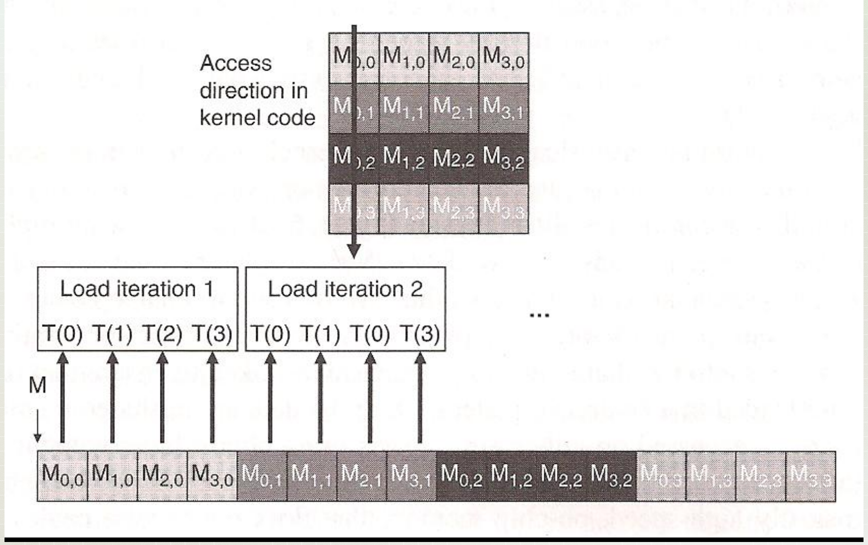
Max Size speedup
Global mem 10
Shared mem 1,000

Hybrid computer performance

- From the performance perspective the major limitation of hybrid computation is the **GPU global memory bandwidth**.
- **LOAD BALANCING**
- SPMT but **if-then-else** instruction can split the execution path. This may cause a longer execution time and loss of performance.

Coalescing global memory access

- Small example of the memory pattern for coalescing global memory access that accelerates processing
- Information residing in shared memory does not need coalescing to achieve high speed data access.
- Global memory access **coalescing** takes place when the same instruction for all threads in a warp accesses consecutive global memory locations.
- In this case hardware combines (coalesces) all accesses into one consolidated and faster access.


Access direction in kernel code
$M_{0,0}$ $M_{1,0}$ $M_{2,0}$ $M_{3,0}$
$M_{0,1}$ $M_{1,1}$ $M_{2,1}$ $M_{3,1}$
$M_{0,2}$ $M_{1,2}$ $M_{2,2}$ $M_{3,2}$
$M_{0,3}$ $M_{1,3}$ $M_{2,3}$ $M_{3,3}$
Load iteration 1
T(0) T(1) T(2) T(3)
Load iteration 2
T(0) T(1) T(0) T(3)
...
M
$M_{0,0}$ $M_{1,0}$ $M_{2,0}$ $M_{3,0}$ $M_{0,1}$ $M_{1,1}$ $M_{2,1}$ $M_{3,1}$ $M_{0,2}$ $M_{1,2}$ $M_{2,2}$ $M_{3,2}$ $M_{0,3}$ $M_{1,3}$ $M_{2,3}$ $M_{3,3}$

Summary

- Hybrid computing is a paradigm discontinuity.
- IMAGINE A WORLD WHERE EVERYBODY HAS AN ACCESS TO PERSONAL SUPERCOMPUTING
  - Every researcher, information technology worker, every academic teacher has a Teraflop Computer on her desk.
    - This calls for resetting our research agenda
    - Create new programming methods for hybrid computing
    - Revisit parallel software packages (LAPACK; Jack D.) algorithms and related mathematics

Literature and References

[1] Calvin Lin and Lawrence Snyder, *Principles of Parallel Programming*, Pearson International Edition, Addison Wesley 2009.

[2] David B.Kirk and Wen-mei W.Hwu, *Programming Massively Parallel Processors*, Morgan Kaufmann, Elsevier 2010.

[3] The CUDA Programming Guide, NVIDIA.

[4] Mark Harris, Optimizing Parallel Reduction in CUDA, NVIDIA, 2007.

[5] Liang Z.P. and Lauterbur P., *Principles of Magnetic Resonance Imaging*, A signal processing perspective, Wiley, New York, 1999.

THE END THANK YOU

Timing data 1

- SINGLE CPU Core i7 64bit 2.8GHz

```
SIZE    TIME            RUN COUNT
2       9.743286e-08    4096
4       1.494033e-07    2048
8       7.512598e-07    1024
16      5.197445e-06    512
32      3.879064e-05    256
64      3.382112e-04    126
128     3.561758e-03    64
256     3.198436e-02    32
512     3.056566e-01    16
1024    9.015563e+00    8
2048    8.564798e+01    4
4096    7.524644e+02    2
```

Times in seconds

Max time=752 sec

Timing Data 2

- TESLA GPU 240 Cores 1.3GHz GLOBAL MEMORY

```
SIZE    TIME            RUN  COUNT
2       2.896618e-05    4096
4       2.955391e-05    2048
8       3.000104e-05    1024
16      5.449054e-05    512
32      1.006258e-04    256
64      2.920939e-04    126
128     2.319826e-03    64
256     1.673443e-02    32
512     1.333204e-01    16
1024    1.062235e+00    8
2048    8.486859e+00    4
4096    6.796577e+01    2
```

For the largest size matrix tested the speedup is about 10.

Timing data 3

- TESLA GPU 240 Cores 1.3GHz SHARED MEM

```
SIZE    TIME            RUN COUNT
2       1.689454e-05    4096
4       1.807400e-05    2048
8       1.638880e-05    1024
16      1.720564e-05    512
32      2.136294e-05    256
64      3.029413e-05    126
128     7.786980e-05    64
256     2.657310e-04    32
512     2.124511e-03    16
1024    1.163225e-02    8
2048    8.727042e-02    4
4096    6.912488e-01    2
```

For the largest size matrix tested the speedup is about 1,000

Workshop vs. conference

- Conference
  - We report what we have done
  - Indicate some future research plans
  - Few short discussions and panels

- Workshop
  - We discuss ideas to be explored
  - Mainly discussions and panels
  - Reports from small working groups