Discussion on

NVIDIA's Compute Unified Device Architecture (CUDA)

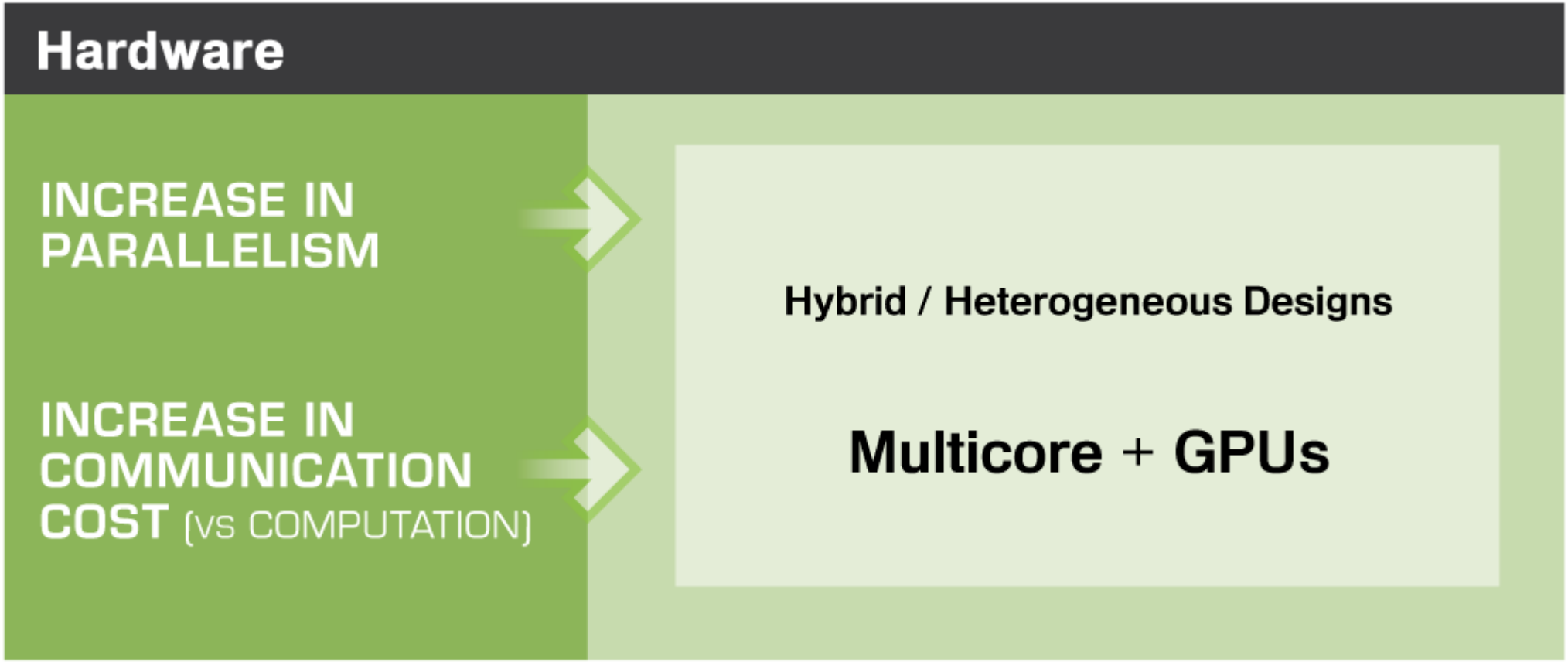
Stan Tomov

04/11/2012

(see also : http://www.nvidia.com/object/cuda_home_new.html)

Why talking about CUDA ?

- Hardware Trends



Evolution of GPUs

- GPUs continue to improve due to ever increasing computational requirements, as better games are linked to
  - **Faster** and more **realistic graphics**
    (more accurate and complex physics simulations)
- To acquire ever
  - **More power** (1 TFlop/s in single, 140 GB/s memory bandwidth)
  - **More functionality** (support fully IEEE double precision, multithreading, pointers, asynchronicity, levels of memory hierarchy, etc.)
  - **More programmability** (with CUDA no need to know graphics to program for GPUs; can use CUDA libraries to benefit from GPUs without knowing CUDA)
- Towards **hybrid architectures**, integrating (in varying proportions) two major components
  - **Multicore** CPU technology
  - Special purpose hardware and accelerators, especially **GPUs**

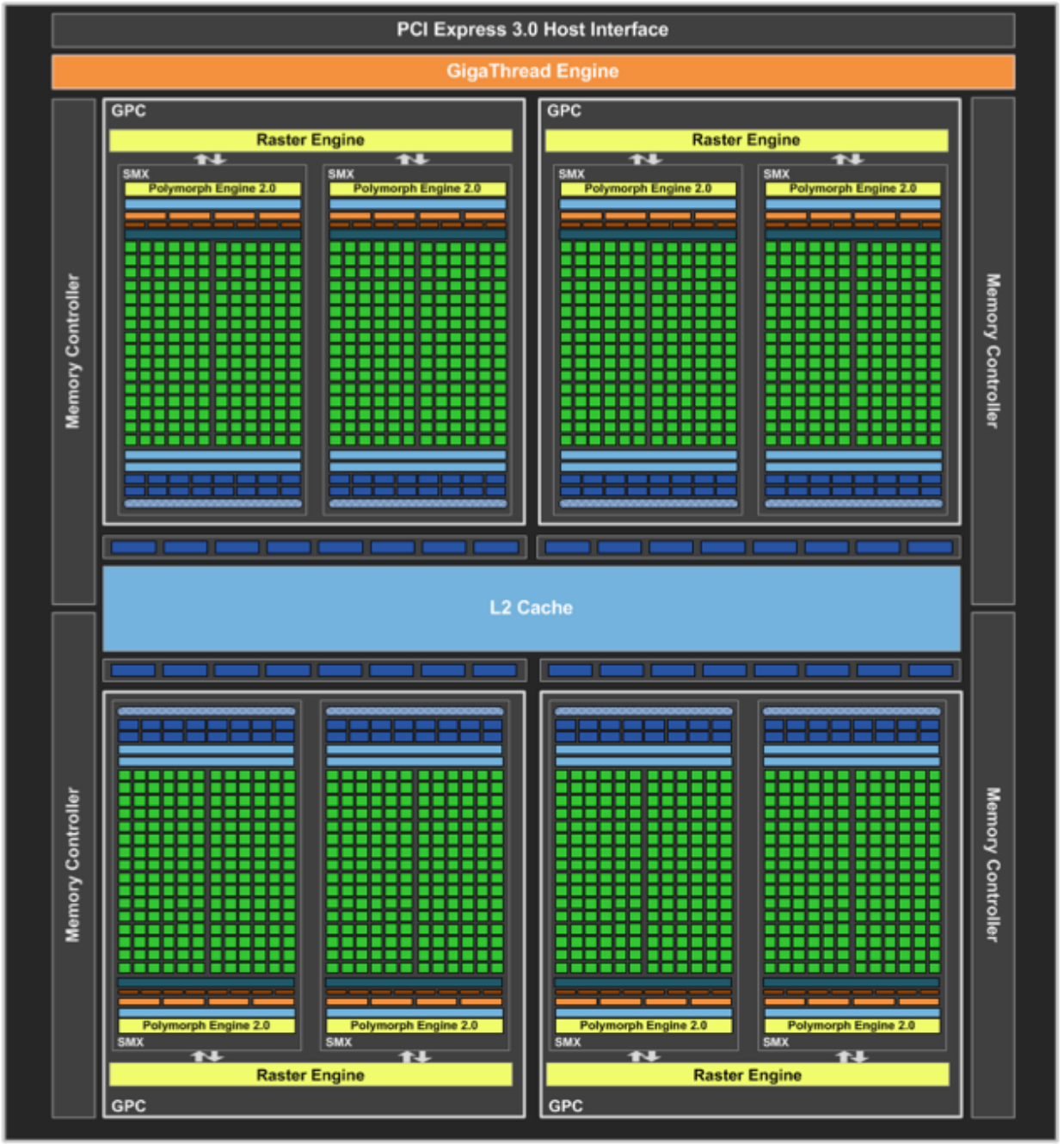
    as evident from major chip manufacturers, such as Intel, AMD, IBM, and NVIDIA

Current NVIDIA GPUs

Feature	Fermi (GTX 580)	Kepler (GTX 680)
Fabrication	40 nm	28 nm
Power Consumption	244 Watts	195 Watts
CUDA Cores	512	1536
Compute Capability	2.1	2.1
Streaming Multiprocessors (SM)	16	8
Cores per SM	32	192
Texture Units	64	128
Warp Schedulers per SM	2	4
L1 Cache per SM	64 KB	64 KB
L2 Cache	768 KB	512 KB
Memory Interface	384 bits	256 bits
Bus Type	PCIe 2.0 x16	PCIe 3.0 x16
Memory Type	GDDR5	GDDR5
Memory Bandiwdth	192.2 GB/s	192.2 GB/s
Core Speed	772 MHz	1002 MHz

Source : http://blog.cuvilib.com/2012/03/28/nvidia-cuda-kepler-vs-fermi-architecture/
See also: "GTX 680 Kepler White paper"

Kepler Architecture



GeForce GTX 680 Block diagram

Source :
http://blog.cuvilib.com/2012/03/28/nvidia-cuda-kepler-vs-fermi-architecture/

See also: "GTX 680 Kepler White paper"

How to learn CUDA?

1) Get the hardware and install the latest
   NVIDIA drivers, CUDA Toolkit, and CUDA SDK

2) Compile, run, and study some of the projects of interest that come with the NVIDIA SDK

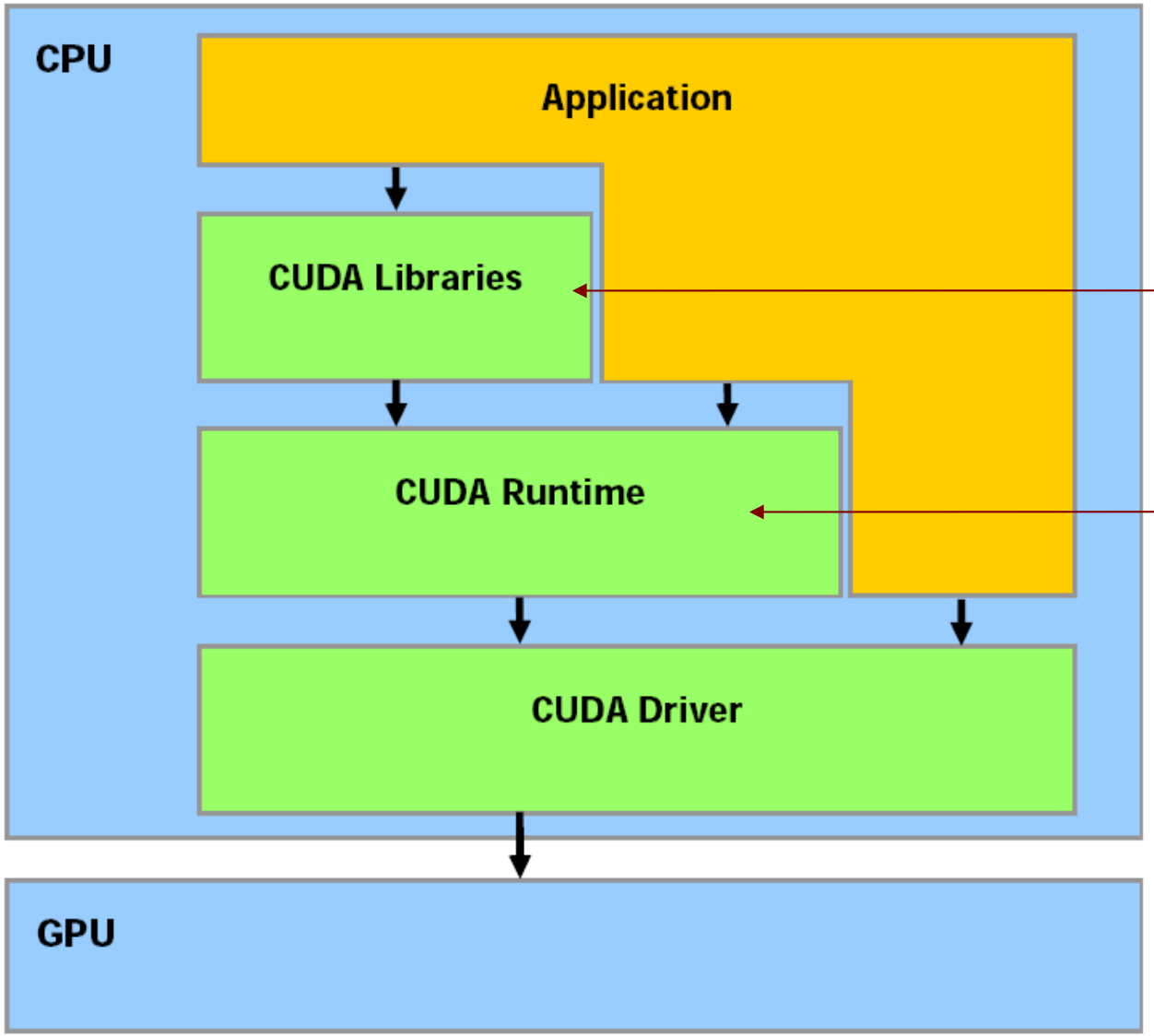
3) Do **Homework 9, Part II**
   Note: This exercise is on **Hybrid computing**, something that we encourage.
   It shows you don't need to know CUDA in order to benefit GPUs –just
   to design your algorithms on high level, splitting the computation
   between CPU and GPU, and using CUDA kernels for the GPU part.

4) Develop user specific CUDA kernels (whenever needed)
   [read the CUDA Programming guide & study projects of interest]

GPUs for HPC

- Programmability
- Performance
- Hybrid computing

Programmability



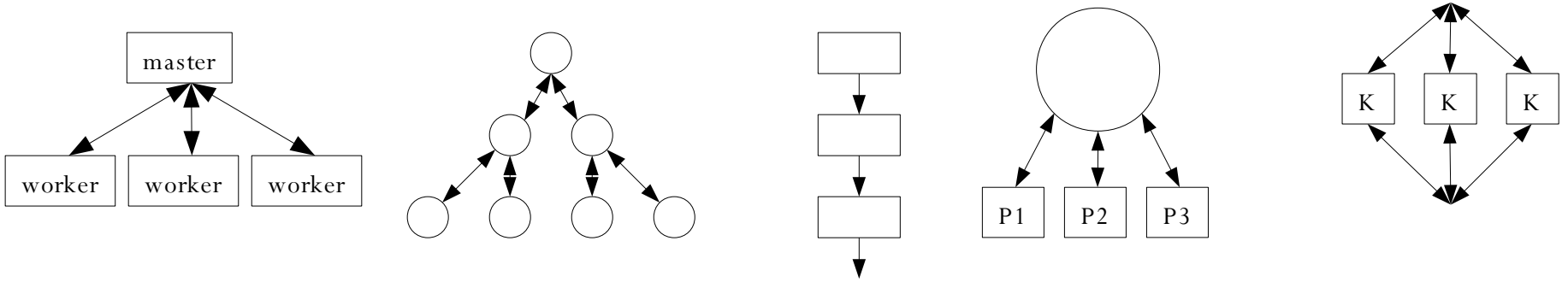
CUDA Software Stack

CUBLAS, CUFFT, MAGMA, ...

C like API

How to program in parallel?

- There are many parallel programming paradigms, e.g.,



master/worker divide and conquer pipeline work pool data parallel (SPMD)

- In reality applications usually combine different paradigms
- CUDA and OpenCL have roots in the data-parallel approach (now adding support for task parallelism)
  http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf
  http://developer.download.nvidia.com/OpenCL/NVIDIA_OpenCL_JumpStart_Guide.pdf

Programming model

A highly multithreaded coprocessor

*** thread block**

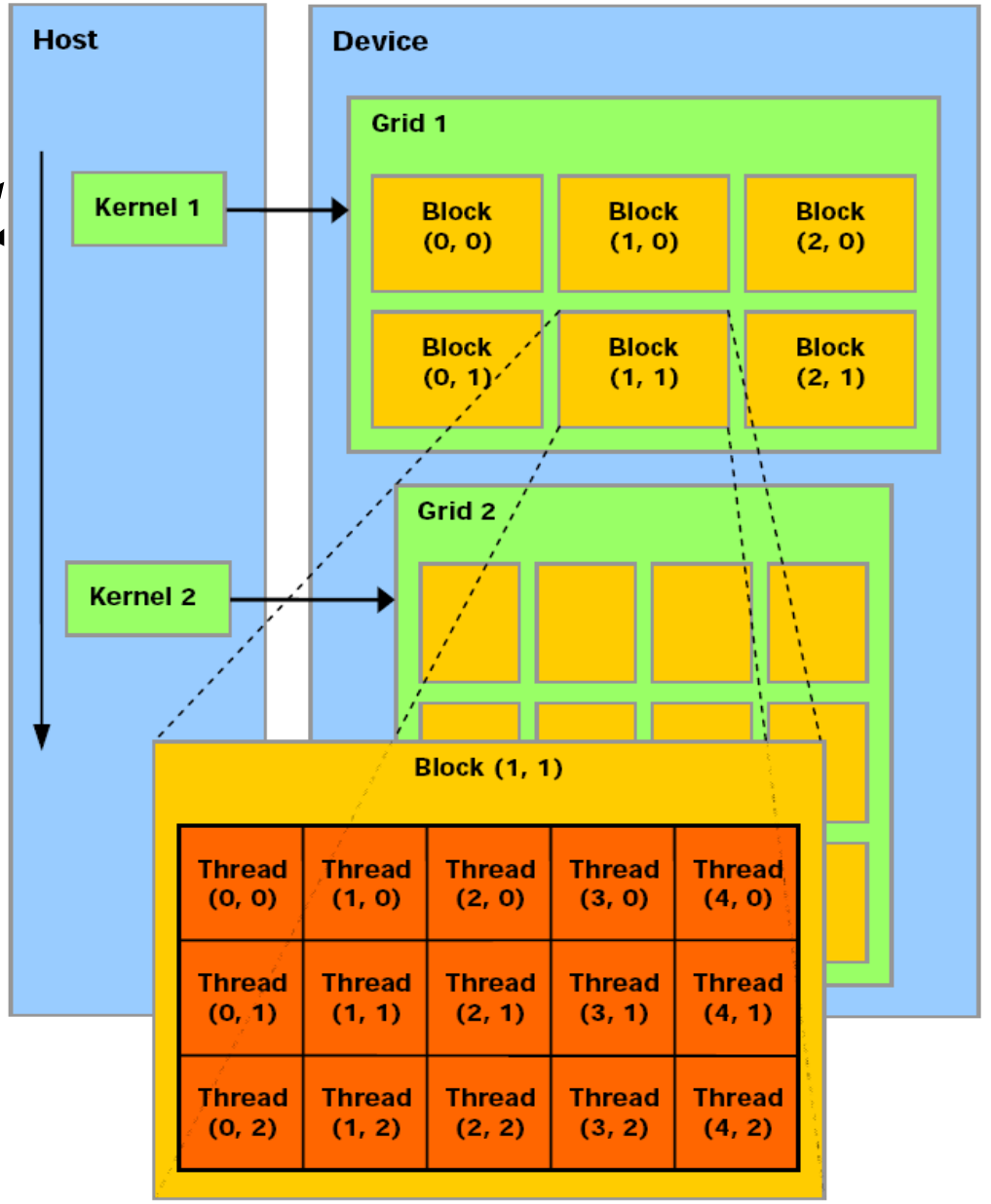
(a batch of threads with fast shared memory executes a kernel)

*** Grid of thread blocks**

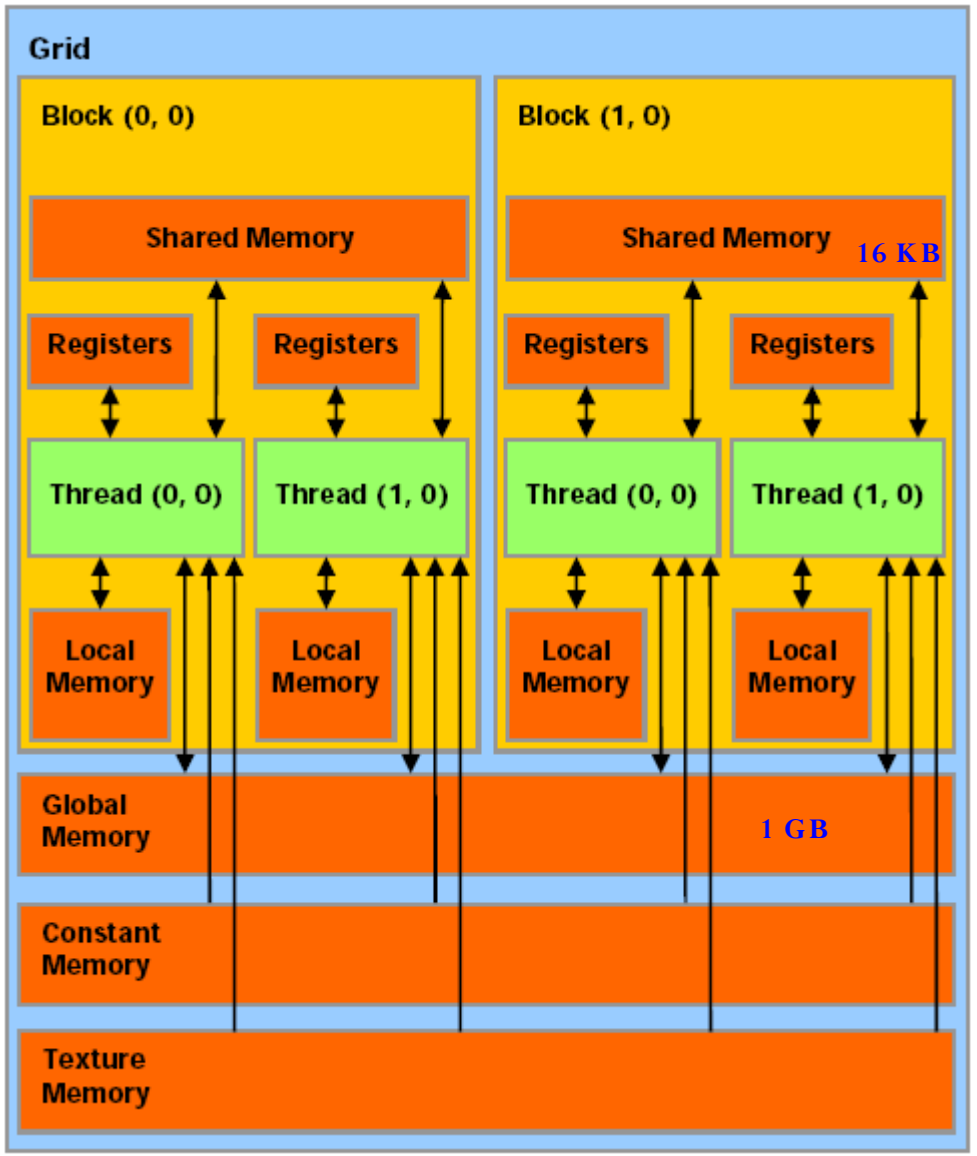
(blocks of the same dimension, grouped together to execute the same kernel; reduces thread cooperation)

```
//set the grid and thread configuration
Dim3 dimBlock(3,5);
Dim3 dimGrid(2,3);

//Launch the device computation
MatVec<<<dimGrid, dimBlock>>>( . . . );
```

```
__global__ void MatVec( . . . ) {
  //Block index
  int bx = blockIdx.x;
  int by = blockIdx.y;

  //Thread index
  int tx = threadIdx.x;
  int ty = threadIdx.y;
  . . .
}
```



Performance



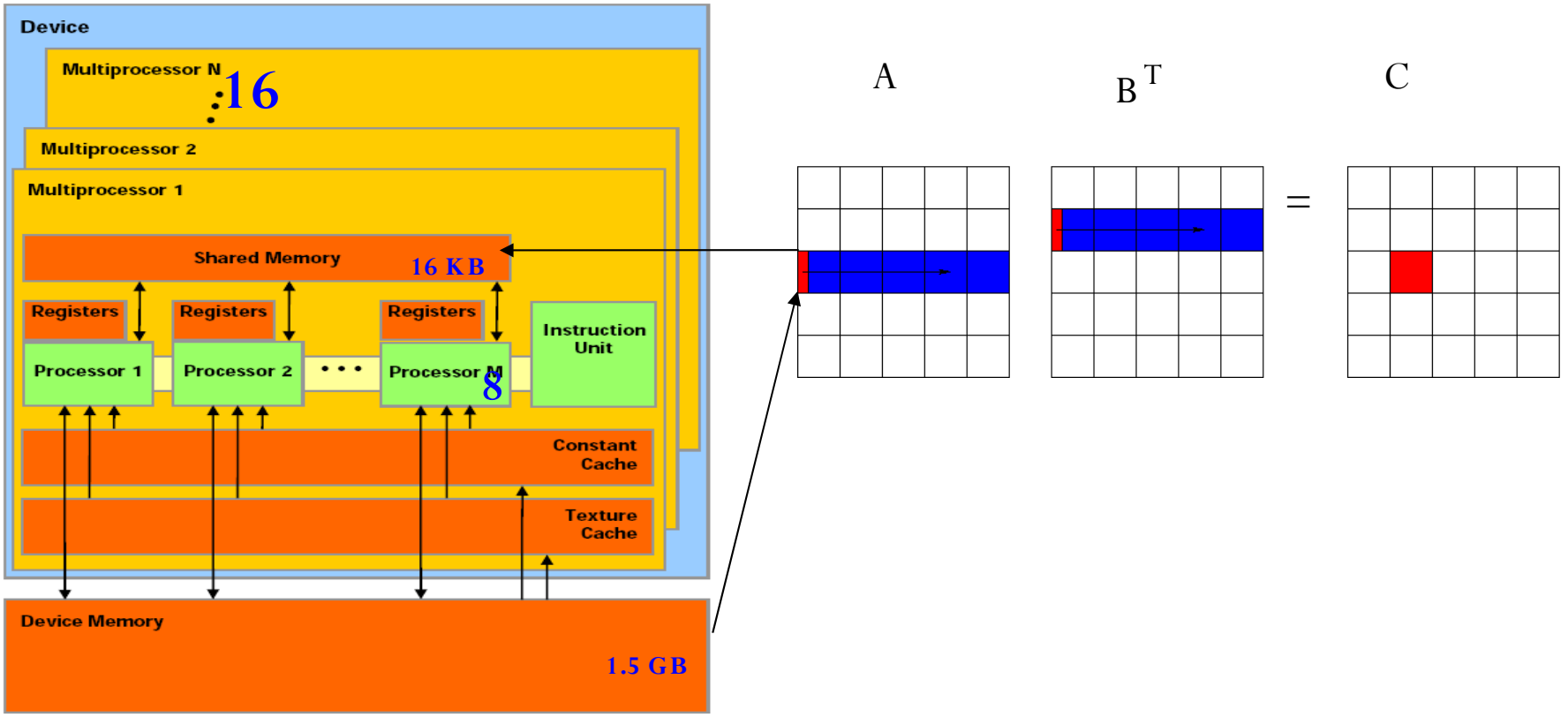
Memory model [numbers for **GTX 280**]

High performance derives from

- High parallelism
  [**1536 processing elements**]
- Memory hierarchy
  Global, L2, L1 cache & Shared memory + registers
  [allows for **memory reuse**]
- High bandwidth to memory
  [**192 GB/s**]
- CPU- GPU Interface: PCI Express x 16
  [up to **4 GB/s peak per direction**
  up to 8 GB/s concurrent bandwidth]

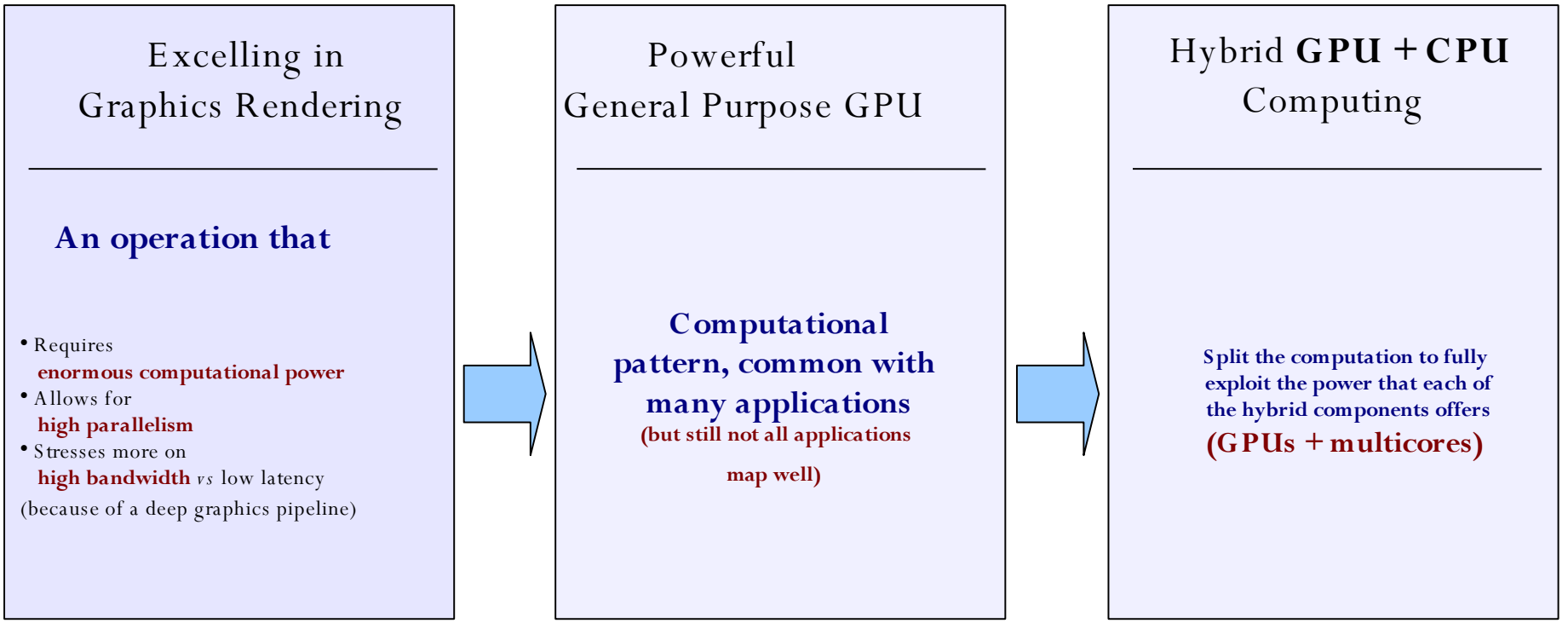
An Example of Memory Reuse

(through use of shared memory)



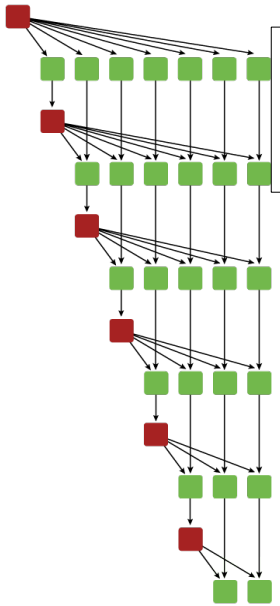
[see sgemm example file from lecture 10]

Hybrid Computing



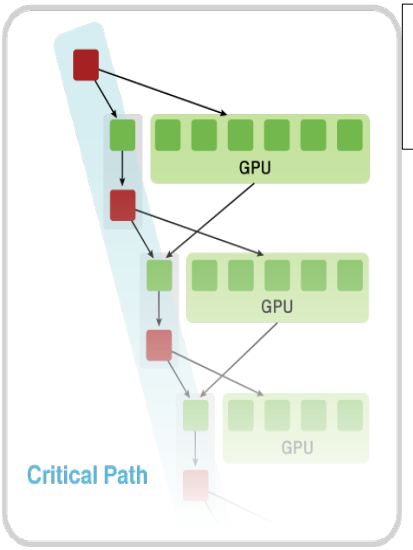
An approach for multicore+GPUs

- Split algorithms into **tasks** and **dependencies** between them, e.g., represented as DAGs
- Schedule the execution in parallel without violating data dependencies



Algorithms as DAGs
(small tasks/tiles for homogeneous **multicore**)

e.g., in the **PLASMA** library
for Dense Linear Algebra
http://icl.cs.utk.edu/plasma/



Hybrid CPU+GPU algorithms
(small tasks for multicores and large tasks for GPUs)

e.g., in the **MAGMA** library
for Dense Linear Algebra
http://icl.cs.utk.edu/magma/

Discussion

- Dense Linear Algebra
  - Matrix-matrix product
  - LAPACK with CUDA
- Sparse Linear Algebra
  - Sparse matrix-vector product
- Projects using CUDA

Conclusions

- **GPU computing**
  Significantly outperform current multicores on many real world applications (illustrated for DLA which has been traditionally of HP on x86 architectures)
  - **New algorithms needed** (increased parallelism and reduced communication)
  - **Speed *vs* accuracy** trade-offs
  - **Autotuning**

- **Hybrid GPU+CPU computing**
  There are still applications – or at least part of them – that do not map well on GPU architectures and would benefit much more a hybrid one

- **Architecture trends**
  Towards heterogeneous/hybrid designs, integrating (in varying proportions) two major components
  - **Multicore** CPU technology
  - Special purpose hardware and accelerators, especially **GPUs**