HPC with Multicore and GPUs

Stan Tomov
Electrical Engineering and Computer Science Department
University of Tennessee, Knoxville

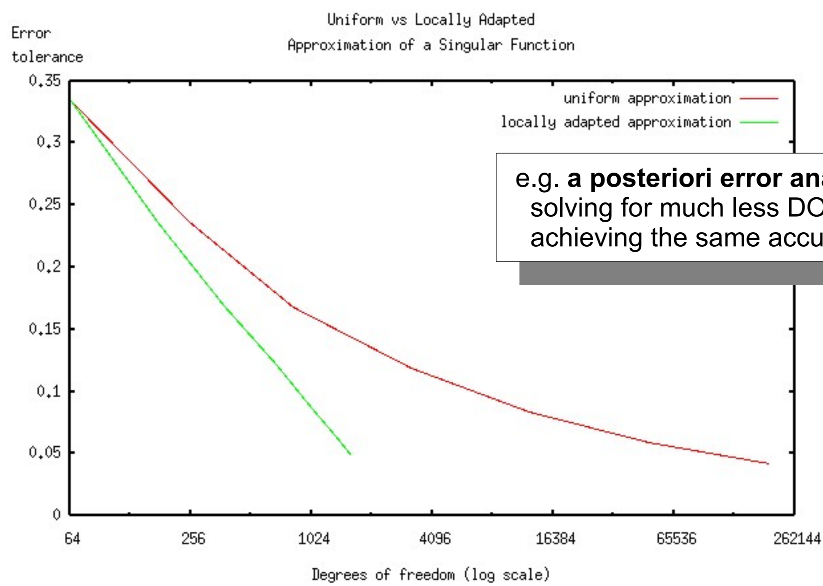
CS 594 Lecture Notes
March 28, 2012

Outline

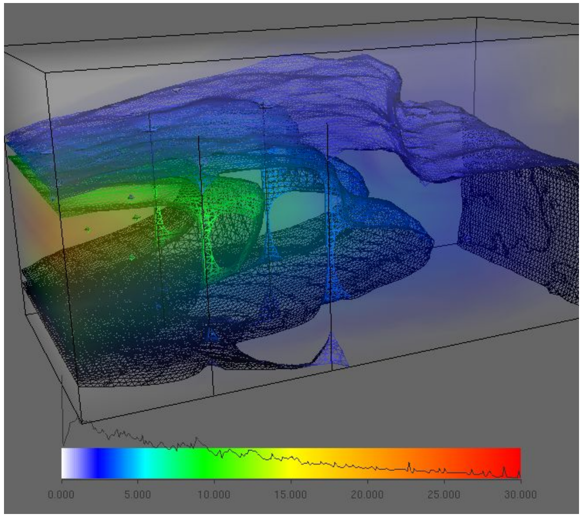
- **Introduction**
  - Hardware trends
- **Challenges of using multicore+GPUs**
- **How to code for GPUs and multicore**
  - An approach that we will study
- **Introduction to CUDA**
- **Conclusions**

Speeding up Computer Simulations

Better numerical methods

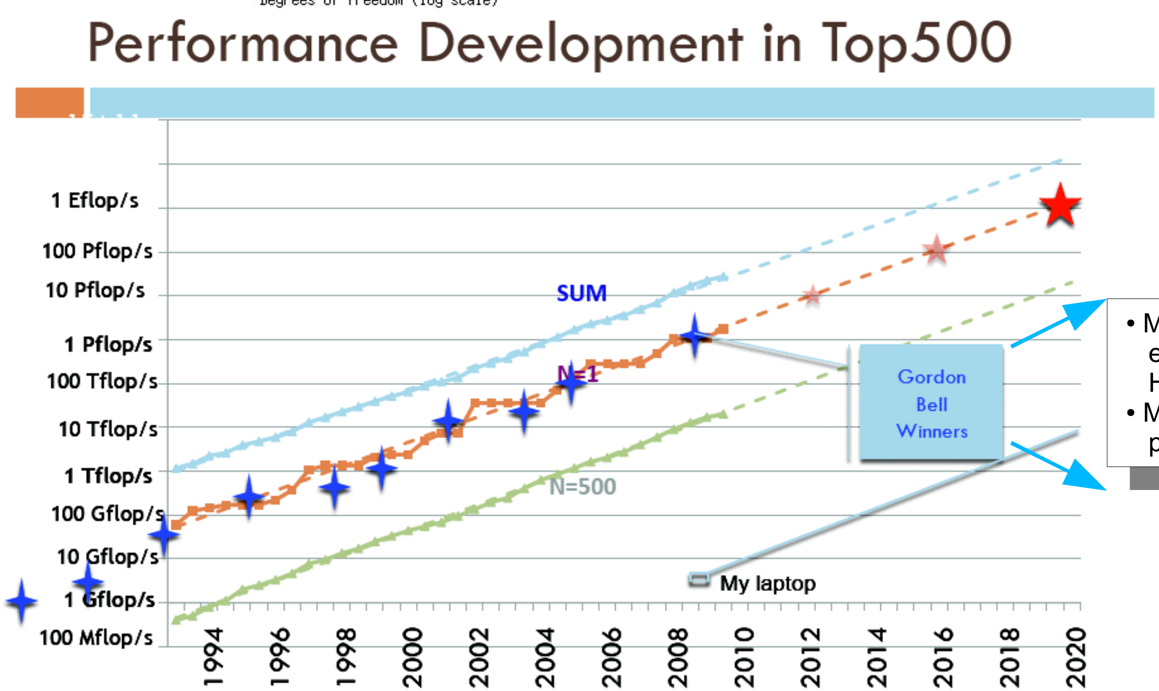


e.g. **a posteriori error analysis**: solving for much less DOF but achieving the same accuracy



http://www.cs.utk.edu/~tomov/cflow/

Exploit advances in hardware

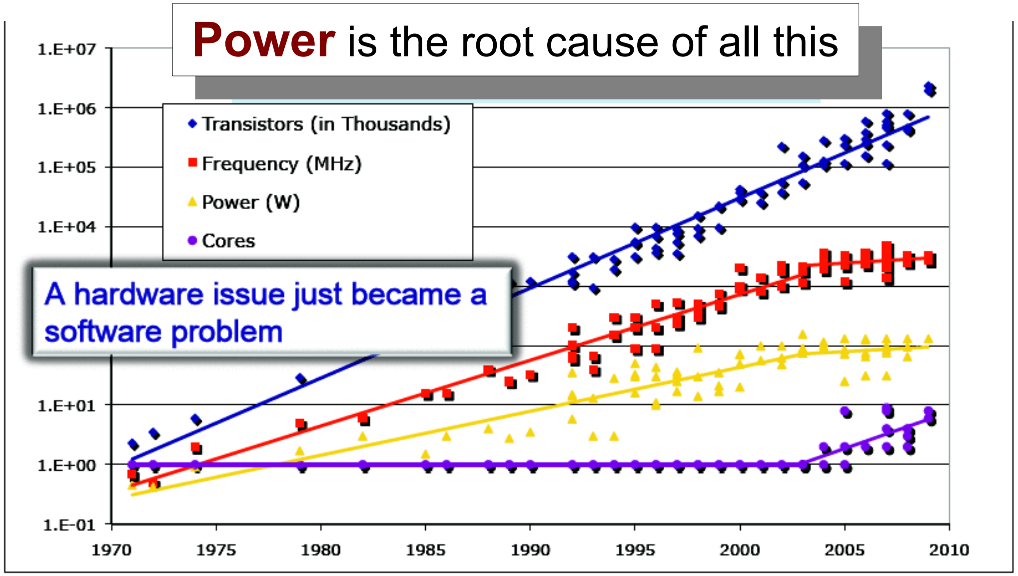


- Manage to use hardware efficiently for real-world HPC applications
- Match LU benchmark in performance !

Why multicore and GPUs?

Hardware trends

- **Multicore**



(Source: slide from Kathy Yelick)

- **GPU Accelerators**

	GeForce GTX 280	GeForce GTX 260	Tesla C1060	Tesla S1070
Form Factor	Dual slot card	Dual slot card	Dual slot card	Rackmount
TPCs	10	8	10	4x10
SMs	30	24	30	4x30
SPs	240	192	240	4x240
Graphics Freq.	602MHz	576MHz		
Processor Freq.	1296MHz	1242MHz	1300MHz	1500MHz
Memory Freq.	1107MHz	999MHz	800MHz	800MHz
Memory Bandwidth	141.7GB/s	127.9GB/s	102.4GB/s	4x102.4GB/s
Memory Capacity	1GB	896MB	4GB	4x4GB
Power	236W TDP	183W TDP	160W "Typical"	700W "Typical"
SP GFLOP/s (wo/MUL)	622.1	476.9	624.0	4x720.0
SP GFLOP/s (w/MUL)	933.1	715.4	936.0	4x1080.0
DP GFLOP/s	77.8	59.6	78.0	4x72.0

(Source: "*NVIDIA's GT200: Inside a Parallel Processor*")

Main Issues

Despite issues, **high speedups** on HPC applications are reported using GPUs
(from NVIDIA CUDA Zone homepage)

- **Increase in parallelism** *1
  How to code (programming model,
  language,
  productivity, etc.)?

- **Increase in commun. cost (vs computation)** *2
  How to redesign algorithms?

- **Hybrid Computing** *3
  How to split and schedule the computation between hybrid hardware components?

CUDA architecture & programming: *1
- A data-parallel approach that scales
- Similar amount of efforts on using CPUs *vs* GPUs by domain scientists demonstrate the GPUs' potential

Processor speed improves 59% / year but memory bandwidth by 23% latency by 5.5% *2

e.g., schedule small non-parallelizable tasks on the CPU, and large and paralelizable on the GPU *3

Evolution of GPUs

GPUs: excelling in graphics rendering

Scene model → streams of data → Graphics pipelined computation → Final image

Repeated fast over and over: e.g. TV refresh rate is 30 fps; limit is 60 fps

- Currently, can be viewed as **multithreaded multicore vector units**

This type of computation:

- Requires **enormous computational power**
- Allows for **high parallelism**
- Needs **high bandwidth *vs* low latency**

(as low latencies can be compensated with deep graphics pipeline)

Obviously, this pattern of computation is common with many other applications

Challenges of using multicore+**GPU**s

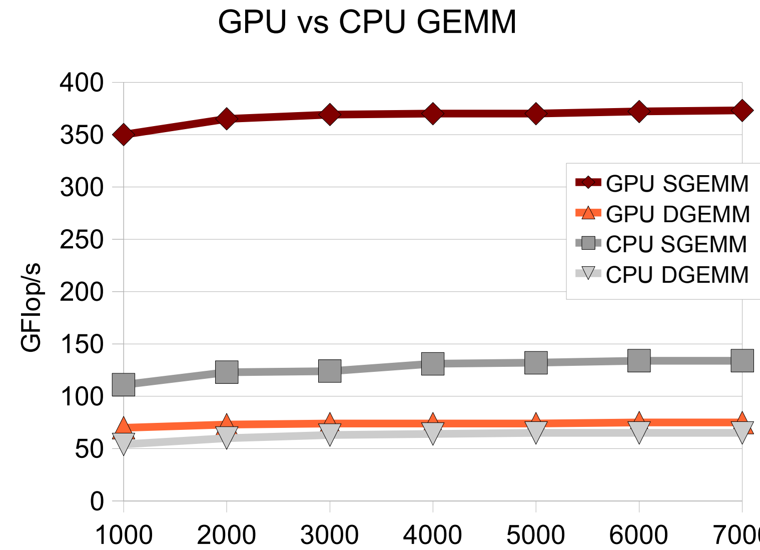
- **Massive parallelism**
  Many GPU cores, serial kernel execution
  [e.g. 240 in the GTX280; up to 512 in ***Fermi*** – to have concurrent kernel execution]

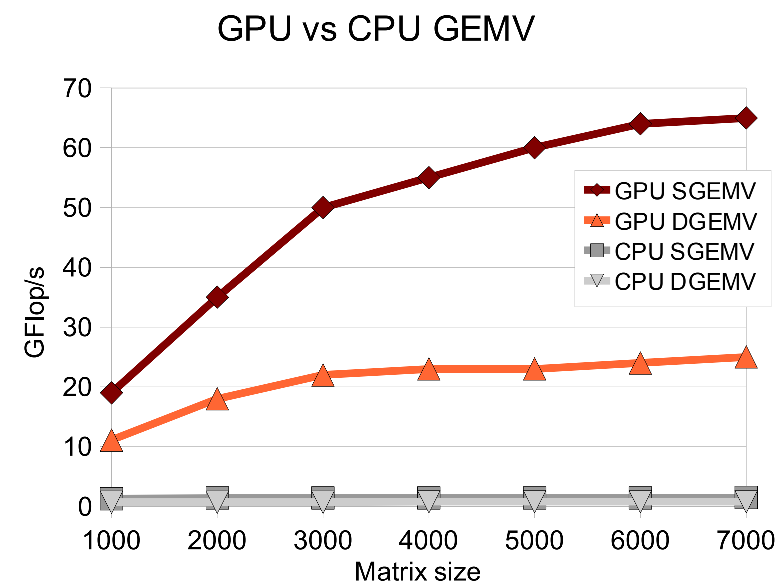
- **Hybrid/heterogeneous architectures**
  Match algorithmic requirements to architectural strengths
  [e.g. small, non-parallelizable tasks to run on CPU, large and parallelizable on GPU]

- **Compute *vs* communication gap**
  Exponentially growing gap; persistent challenge
  [Processor speed improves 59%, memory bandwidth 23%, latency 5.5%]
  [on all levels, e.g. a GPU Tesla C1070 (4 x C1060) has compute power of O(1,000) Gflop/s but GPUs communicate through the CPU using O(1) GB/s connection]

How to Code for GPUs?

- Complex question
  - Language, programming model, user productivity, etc
- Recommendations
  - **Use CUDA / OpenCL**
    [**already demonstrated benefits in many areas; data-based parallelism; move to support task-based**]
  - **Use GPU BLAS**
    [high level; available after introduction of **shared memory** – can do data reuse; leverage existing developments]
  - **Use Hybrid Algorithms**
    [currently GPUs – massive parallelism but serial kernel execution; hybrid approach – small non-parallelizable tasks on the CPU, large parallelizable tasks on the GPU]



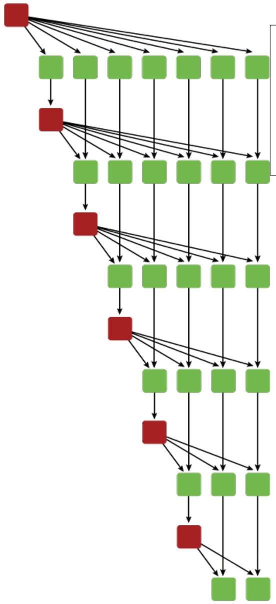


<u>**Typical order of acceleration**</u>:

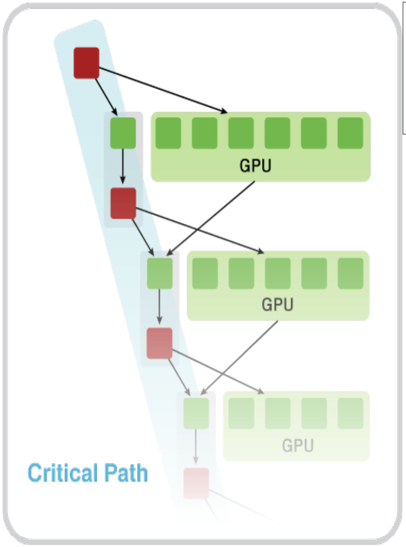
dense matrix-matrix	**O(1) X**
dense matrix-vector	**O(10) X**
sparse matrix-vector	**O(100) X**

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**)



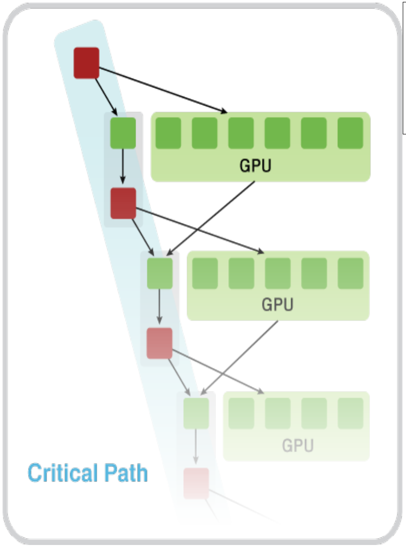
Hybrid CPU+GPU algorithms
(small tasks for multicores and
large tasks for GPUs)

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

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

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

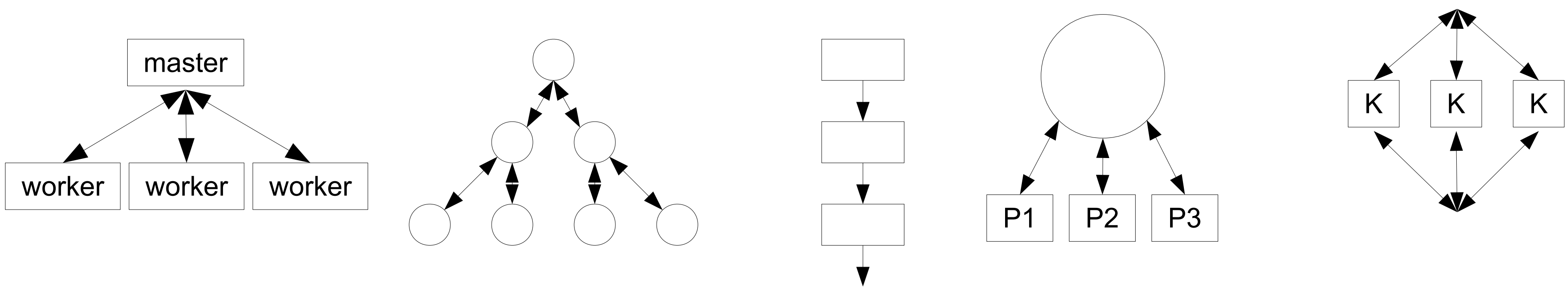


Hybrid CPU+GPU algorithms
(small tasks for multicores and
large tasks for GPUs)

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

How to program in parallel?

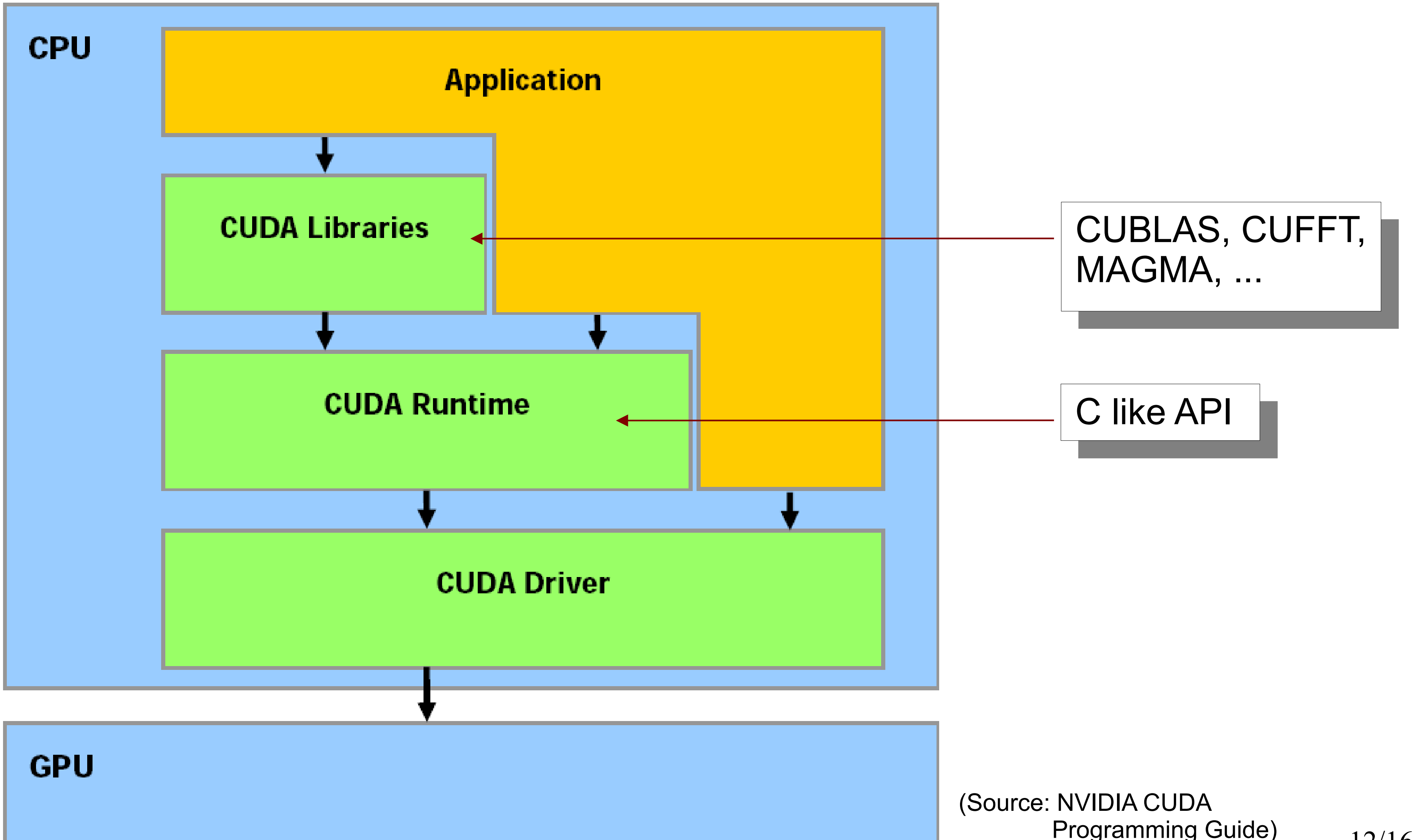
- There are many parallel programming paradigms, e.g.,



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

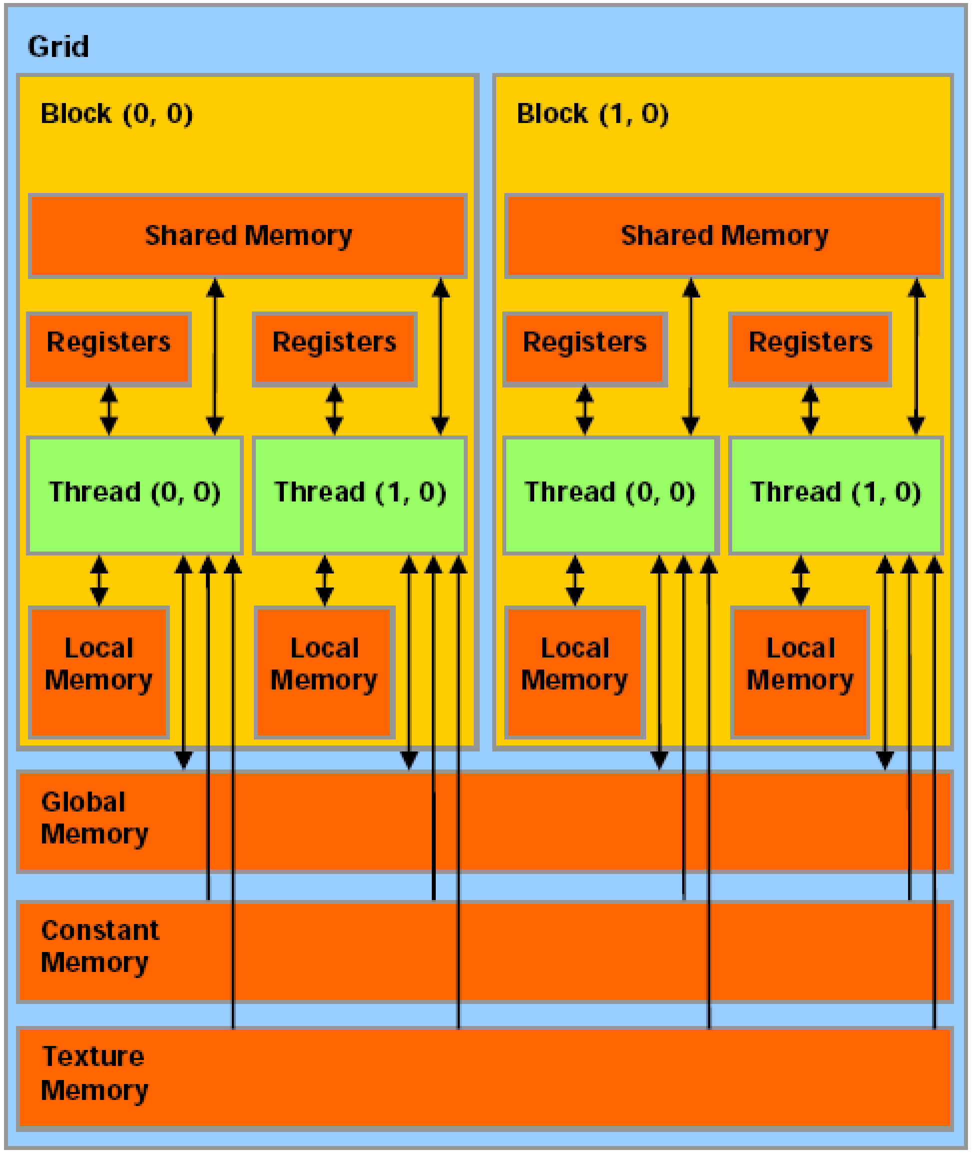
- In reality applications usually combine different paradigms
- CUDA and OpenCL have roots in the data-parallel approach (now adding support for task parallelism)
  http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf
  http://developer.download.nvidia.com/OpenCL/NVIDIA_OpenCL_JumpStart_Guide.pdf

Compute Unified Device Architecture (CUDA) Software Stack



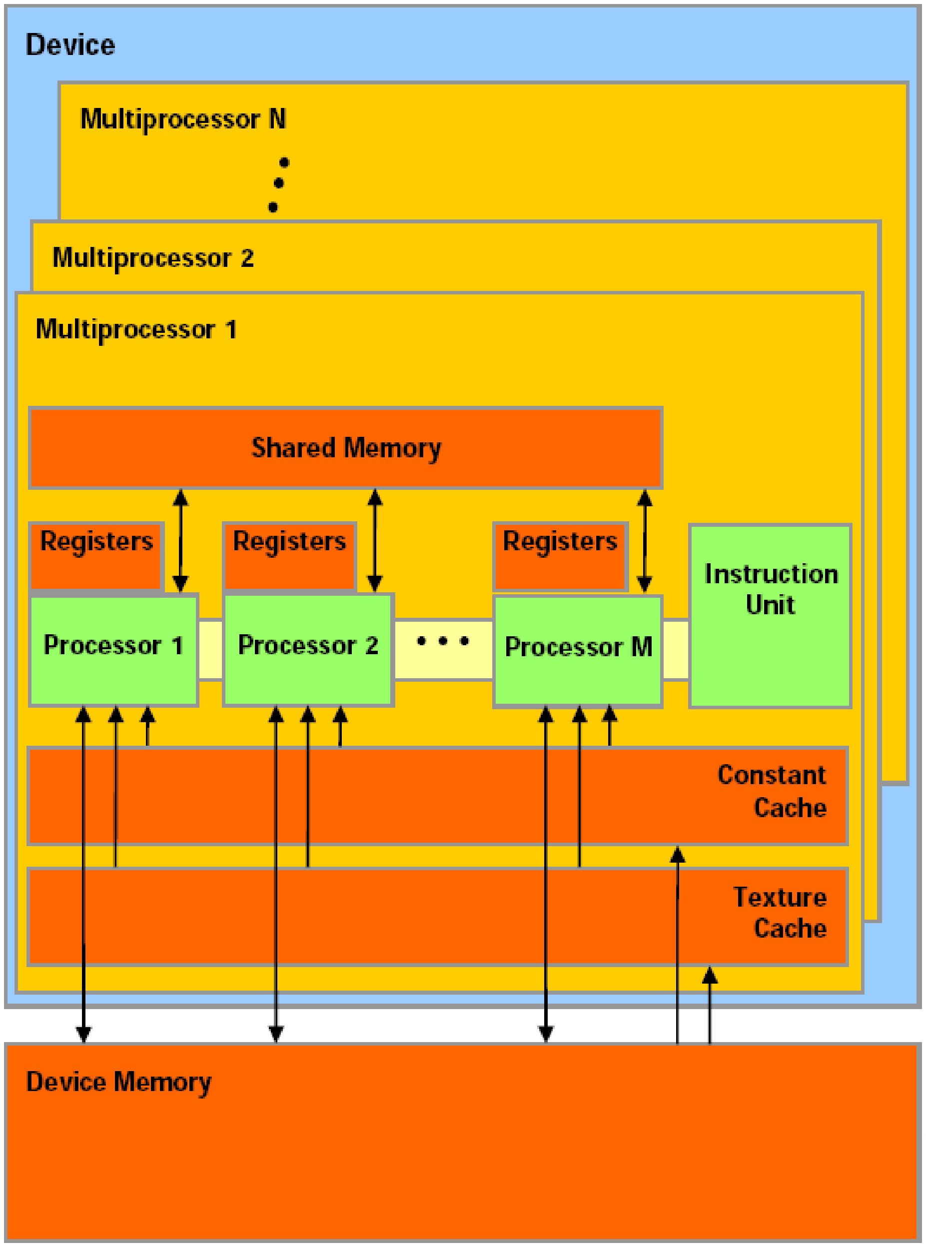
(Source: NVIDIA CUDA Programming Guide)

CUDA Memory Model



(Source: NVIDIA CUDA Programming Guide)

CUDA Hardware Model



(Source: NVIDIA CUDA Programming Guide)

CUDA Programming Model

- Grid of thread blocks
  (blocks of the same dimension, grouped together to execute the same kernel)
- Thread block
  (a batch of threads with fast shared memory executes a kernel)
- Sequential code launches asynchronously GPU kernels

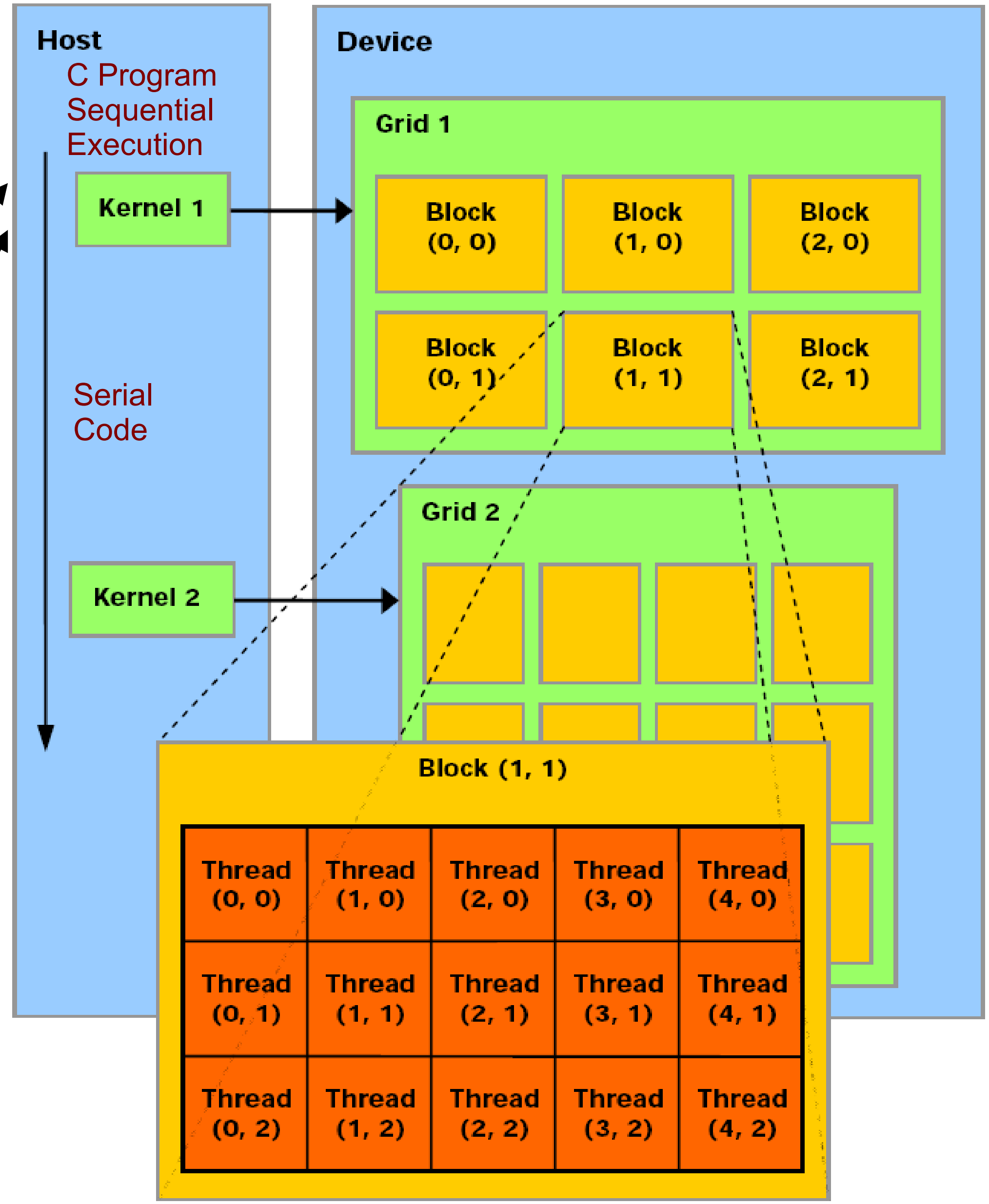
```
// set the grid and thread configuration
Dim3 dimBlock(3,5);
Dim3 dimGrid(2,3);

// 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;
   . . .
}
```

(Source: NVIDIA CUDA Programming Guide)



Conclusions

- **Hybrid Multicore+GPU computing:**
  - **Architecture trends:**
    towards heterogeneous/hybrid designs
  - **Can significantly accelerate linear algebra** [*vs* just multicores] **;**
  - **Can significantly accelerate algorithms that are slow on homogeneous architectures**