GPU Computing

The Ascend of the Coprocessor

Ashraf Suyyagh

In this Presentation

- What are GPUs, GPU computing?
- Why GPU Computing?
- Hardware Organization
- Programming Models – CUDA
- Performance Gains
- Limitations on Performance Gains
- Energy Efficiency
- Open Challenges and Future Research

What are GPUs, GPU Computing?

- Specialized image synthesis processors designed to handle graphic computations and scene generation, which are of parallel nature (same operation on many pixels → high throughput!)
- GPU computing is the use of GPUs for general purpose computing rather than standard graphical computing.

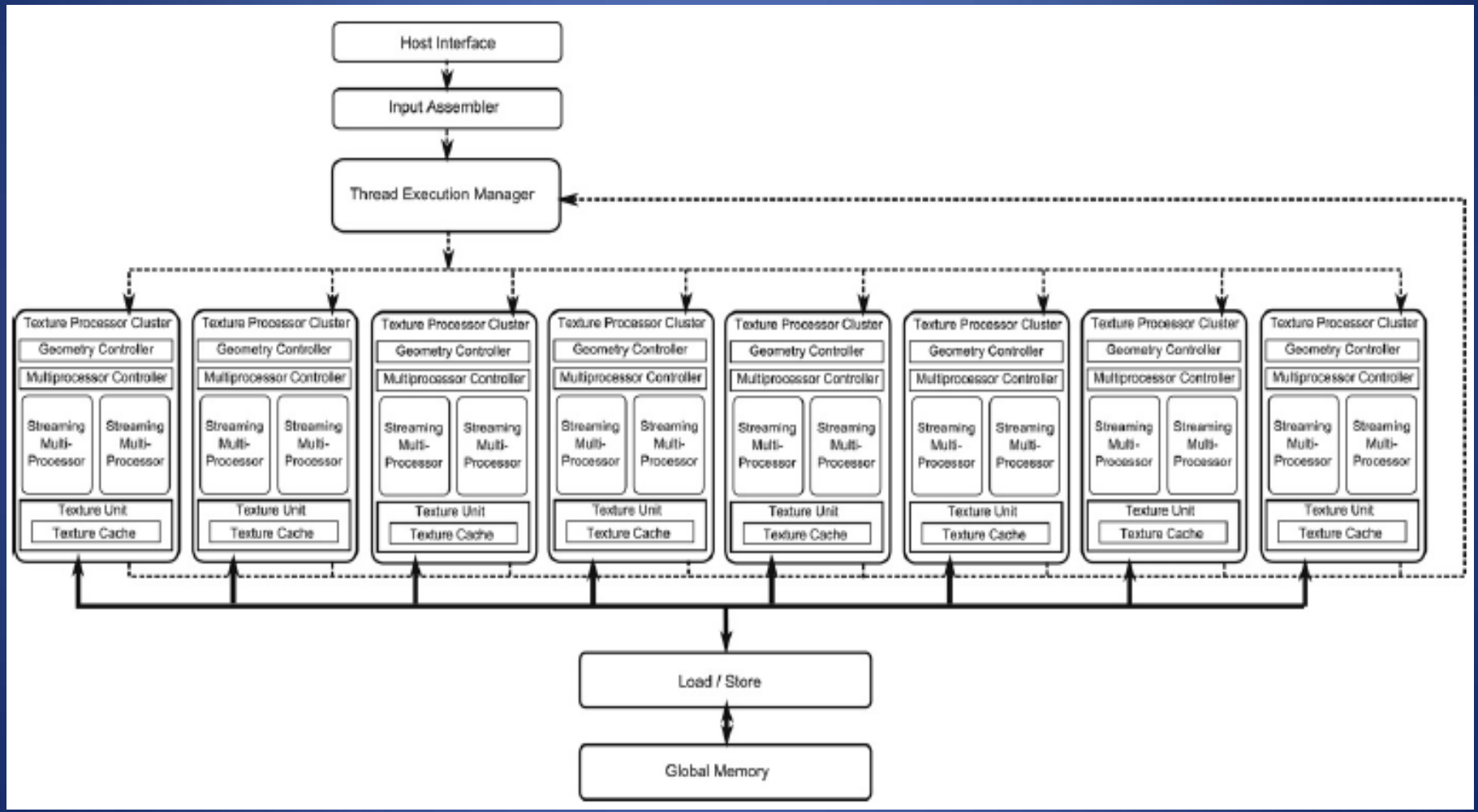
Why use GPUs for General Purpose Computations?

- CPUs are suffering from performance growth slow down, limits to exploiting ILP, power and thermal limitations. Multi core power not used to its full!
- GPUs are a commodity, found in all PCs!
- Modern GPUs provide extensive resources: massive parallelism and processing cores, high arithmetic intensity, high memory bandwidth, high floating point precision, and most importantly flexible and increased programmability
- GPUs are energy efficient!
- Inherent parallelism in hardware design is suitable for high end computing

GPU Hardware Organization and Development

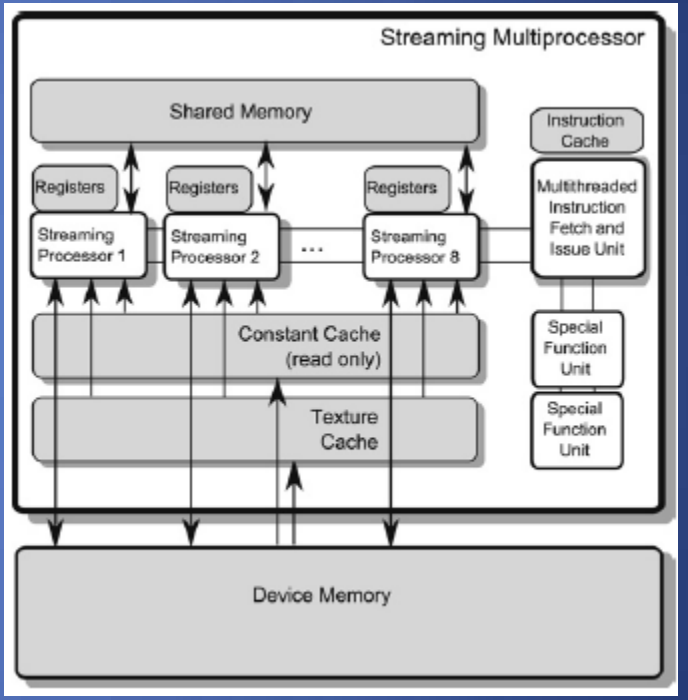
- Early Designs (1970s – 1990s): fixed hardwire pipelines, inflexible
  - → Graphic programmers restricted by amount of detail and realism, couldn't apply their own set of programs
- 1st Generation of programmable GPUs: replaced stages of graphic pipeline with ***specialized*** programmable cores, namely pixel and vertex processors (e.g. NVIDIA GeForce 6 series, 2004)
  - →Problem: depending on load, some processing cores were congested, others idle
- 2nd Generation of programmable GPUs (1st Unified shader generation): introduced unified processors (shaders) (e.g. NVIDIA GeForce 8800 GTX)
  - → Offer dynamic load balancing, each processor could be programmed to do the job of any shader!
  - → True start of general purpose computing
- 2nd Generation of programmable GPUs (2nd Unified shader generation): introduced higher memory bandwidths, 64-bit high precision for the first time, dynamic power management (e.g. NVIDIA GeForce G200 series)

GeForce 8800 GTX Organization



Closer Look at the Heart of 8800 GTX

- The scalable processor array consists of Texture Processing Clusters (TPC)
- Each TPC encompasses a couple of Streaming Multiprocessors (SM)
- Each SM has eight Streaming Processors (SP), which are single precision floating point ALUs
- A 16kB-16 banks dedicated shared memory is built into each SM, also referred to as Per-Block Shared Memory (PBSM)



Programming Models

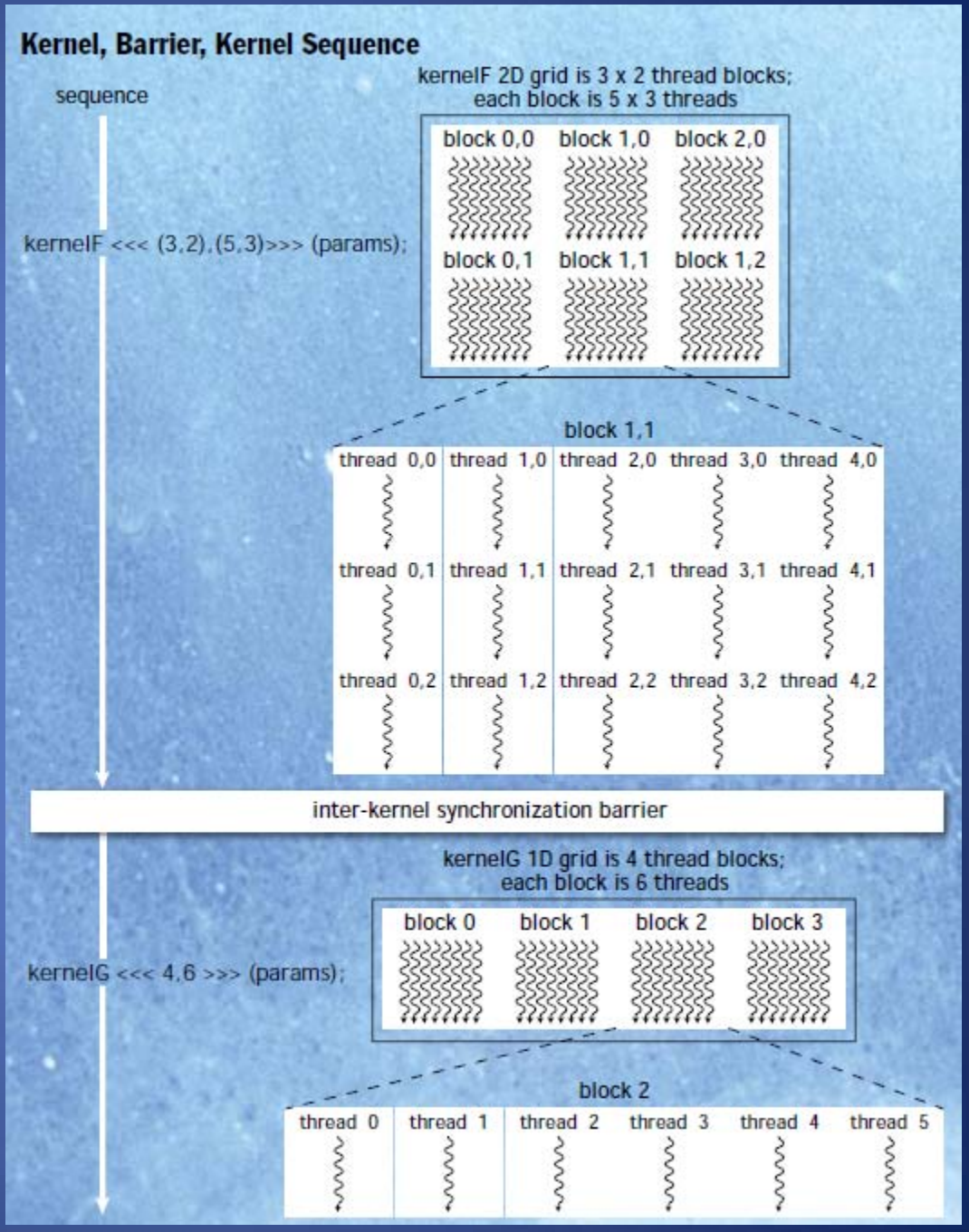
- Parallel computing needs new programming models → sequential does not scale!
- No support for general computing in the early beginnings, had to use the Graphical programming model and APIs (DirectX, OpenGL) to program applications on the CPU
  - → Needed extensive knowledge of the underlying hardware
  - → Think your problem in graphical terms!
  - → Cumbersome, inefficient and highly restrictive!
- In 2007, NVIDIA introduced the Compute Unified Device Architecture (CUDA) programming model
  - → Abstracts the complexity of the complex graphical hardware
  - → extends the C/C++ programming language, high level API

The CUDA Model

- Data arrays are partitioned into blocks which are further partitioned into elements on the condition that the blocks can be *independently* computed in parallel and the elements can *cooperatively* be computed in parallel.
- Programmers write sequential code (kernels) run on the CPU (host) which communicates with GPU (device) through drivers and instantiate threads
- *Kernels* represent parallel tasks across a set of parallel threads on the GPU
- Threads are grouped into thread blocks with a maximum set of 512 concurrent threads which can efficiently cooperate, communicate, synchronize and share data among themselves
- Thread ID (TID) number used to select work and index shared data arrays.
- Thread Block are grouped into grids, also given unique Block ID numbers
- Both grids and thread blocks can be presented in 1-, 2- or 3 dimensional fashion
- Kernel in its entirety is a sequential code, the actual specification of the dimensions of the grid and thread blocks at kernel invocation time by the programmer explicitly specifies the amount of parallelism

The CUDA Model - Continued

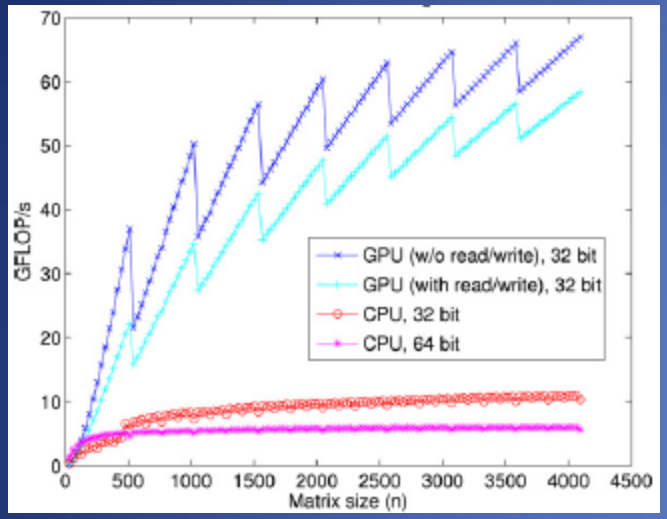
- Data communication between threads within a single thread block is through either the shared memory (low latency) or global memory (high latency)
- Data communication between threads blocks and grid through global memory only
- Independent thread blocks executed concurrently, dependent ones sequentially. Same for kernels (grids), synchronization at this level managed by user

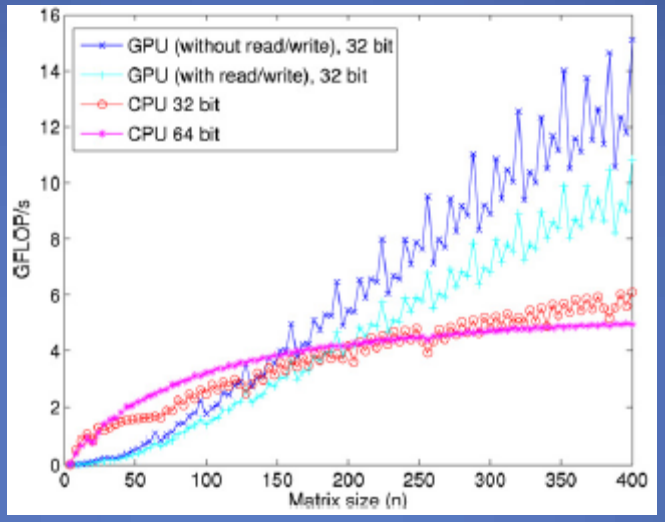


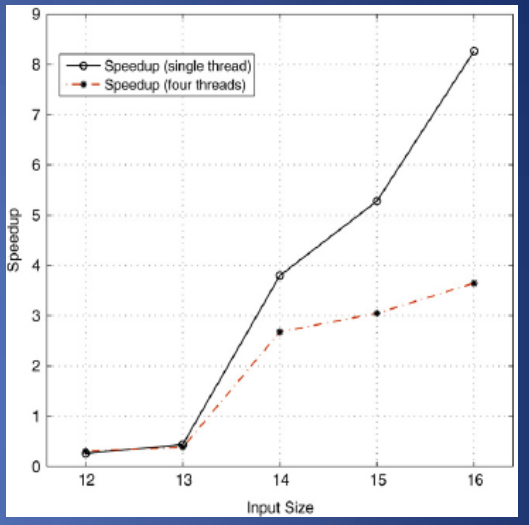
Performance Evaluation

- Many researches were conducted to compare the performance of GPUs against CPUs for high end computing and inherently parallel tasks!
- GPUs outperform CPUs in this application domain
- Speed Ups are orders of magnitude higher than over single and multithreaded implementations of the same application run on CPUs

Sample Results







Performance of a triangular matrix equations solver algorithms on CPU and GPU for large matrices n < 4000 (left) and small matrices n<400 (middle), speed up for a neural network application (left)

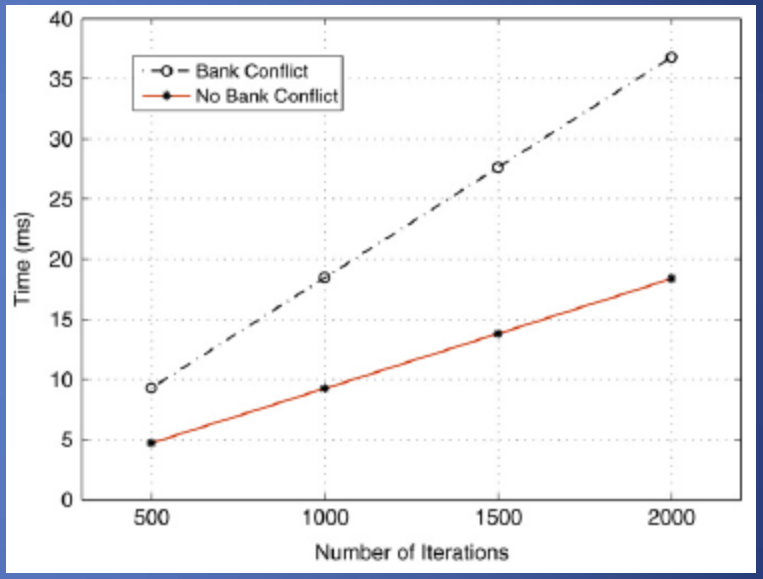
- Effect of memory overhead in GPUs
- Thread start up overhead
- Bank conflicts and per-block thread number effects

Performance Limitations (1)

Hardware Organization Implications:

- *Memory overhead*: The total bandwidth in between the on board graphics memory and the GPU is an order of magnitude higher than that of the PCI Express Bus (e.g. 86.4GB/s vs 8 GB/s)
- *Bank Conflicts*: the SM 16kB shared memory is divided into 16 banks, maximum performance is achieved when addresses in one row.

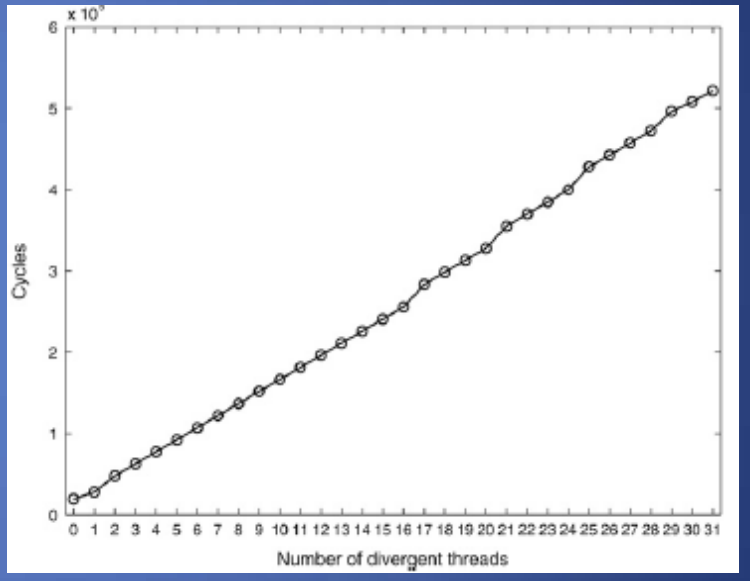
→ Therefore to maximize GPU computing efficiency and achieve higher speed ups and performance gains, programmers need specific knowledge of the underlying hardware architecture and implementation as well as memory hierarchy.



Performance overhead of bank conflicts

Performance Limitations (2)

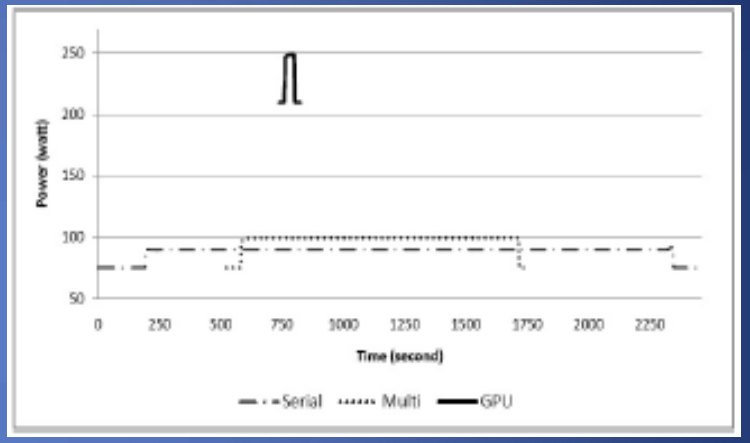
- Programming Model Limitations:
  - *Control flow overhead*: CUDA is not a purely data-parallel model due to thread divergence issue. Thread divergence is caused by control flow instructions (i.e. *if* and *switch* statements) when threads within a warp follow different branches.
  - *Producer- Consumer Overhead:* the CUDA programming model specifies that thread blocks are run to completion and leave no persistent state in the per-block shared memory. To communicate results in between kernels, in a dependent producer consumer kernel pair relationship, is only achieved through the global device memory
  - *Non-contiguous memory access*



Performance overhead of divergent threads

Energy Efficiency

- GPUs are often perceived as power hungry devices because they have high power ratings! (new models 145W – 236W!!)
- Research show that GPUs are Energy efficient though relative to CPUs



A plot of the execution times and power consumption of the serial/multrihreaded/CUDA GEM software implementations

Open Challenges

- Need for an even higher level of programming models and APIs
  → Data placement management, communication and synchronization are still managed by users (complex)
- Overcome the limitations of the heterogonous memory structure which provides for memory bandwidth bottlenecks
- Little research has been conducted on the optimization of GPU energy performance and efficiency. Power and thermal issues will soon prove challenging
- Provide mechanisms to exploit the power of CPUs and GPUs in solving generic problems based on collaborative and a heterogeneous environment with dynamic load balancing

The Future

- GPUs are here to stay; they are not to be considered rivals to CPUs but rather as cooperators for each processor has its own application domain at which it excels most.
- NVIDIA introduced its new Architecture: Fermi with 512 cores, L2 cache and Giga thread scheduler (September 2009)
- Intel entered the domain by announcing an x86 based GPU version: Larabee (2009)

Thank You