
بسم الله الرحمن الرحيم

COE 599

Seminar Presentation

Massive Threading: Using GPUs to Increase the Performance of Digital Forensics Tools

Presenter:

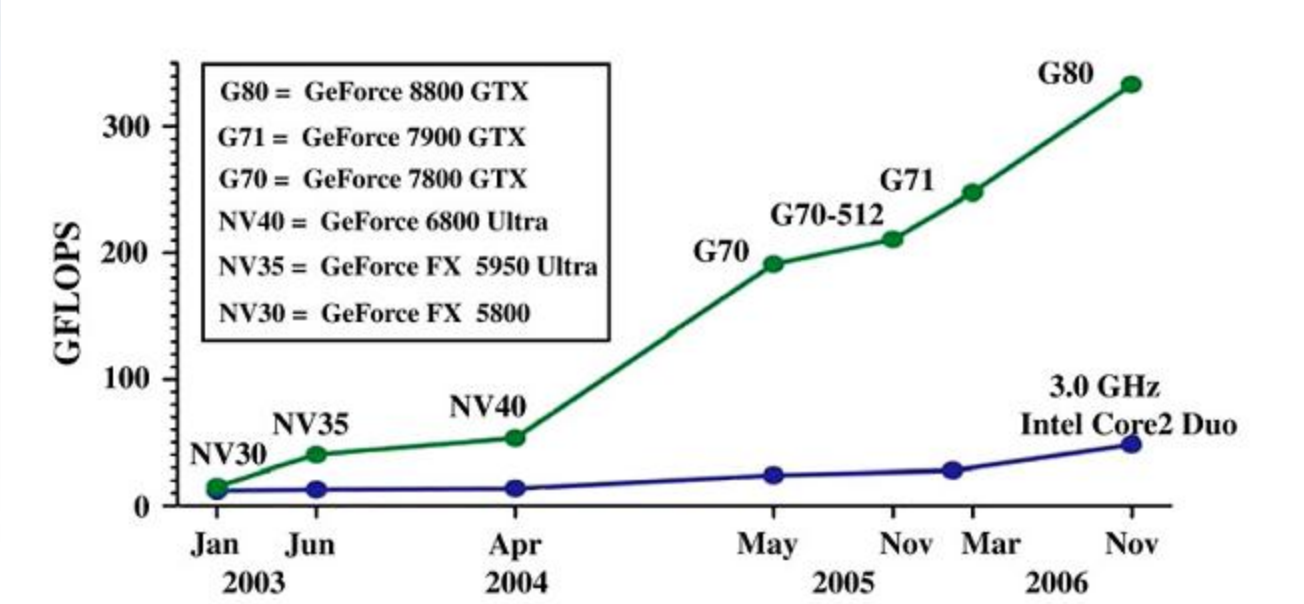
Muhammad Mohsin Butt (g201103010)

Outline

- Introduction
- Background
- Scalpel
- Implementation
- Results

Introduction

- Use of Graphics Processing Units for general purpose computing.
  - High Computation Power.



Introduction

- Why we need more computation power in Digital Forensics.
  - Increase in Digital Forensics investigations.
  - Manual Investigation takes time.
  - Data sets increasing

Introduction

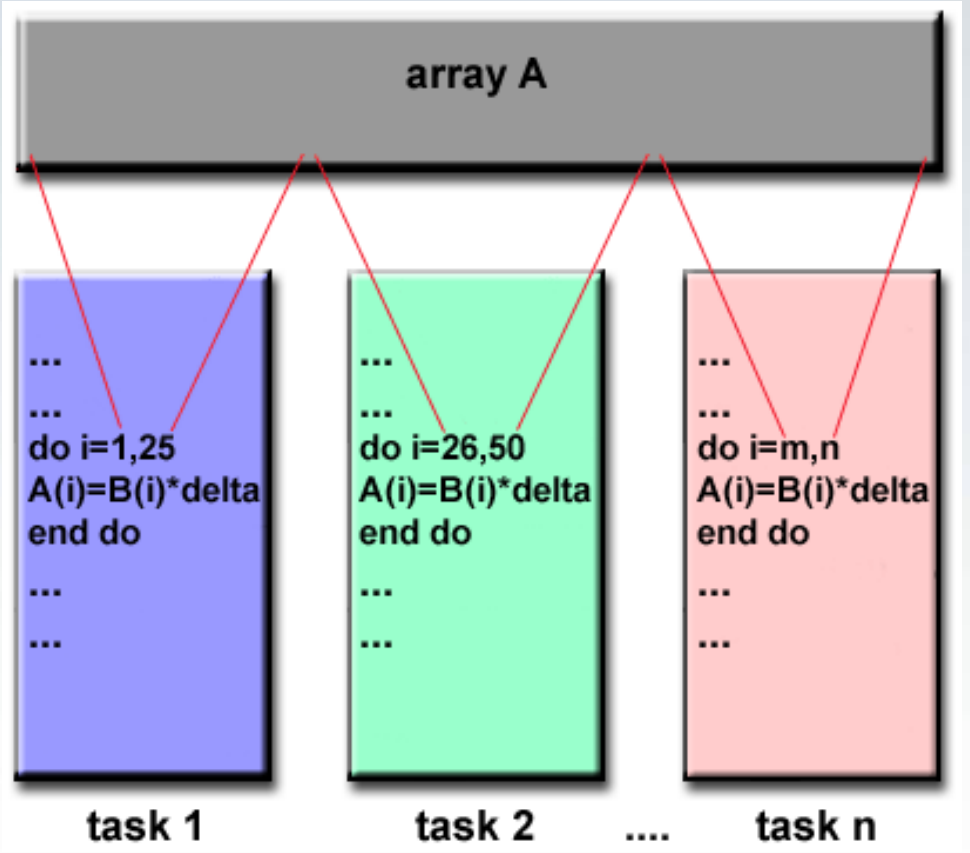
- Explore the use of GPU's in enhancing Digital Forensics tools.
- File carving application Scalpel.
- Performance enhancement is reported at the end.

Background (Parallelism)

- Executing instructions simultaneously to increase the throughput.
- Work is distributed into Threads.
- Many factors caused the shift to parallel computing.
- Hide the memory latency.
- Instruction Level Parallelism
  - Execute many instructions in parallel.
  - Pipelining, Loop Unrolling, Out of order Execution, Speculative execution.

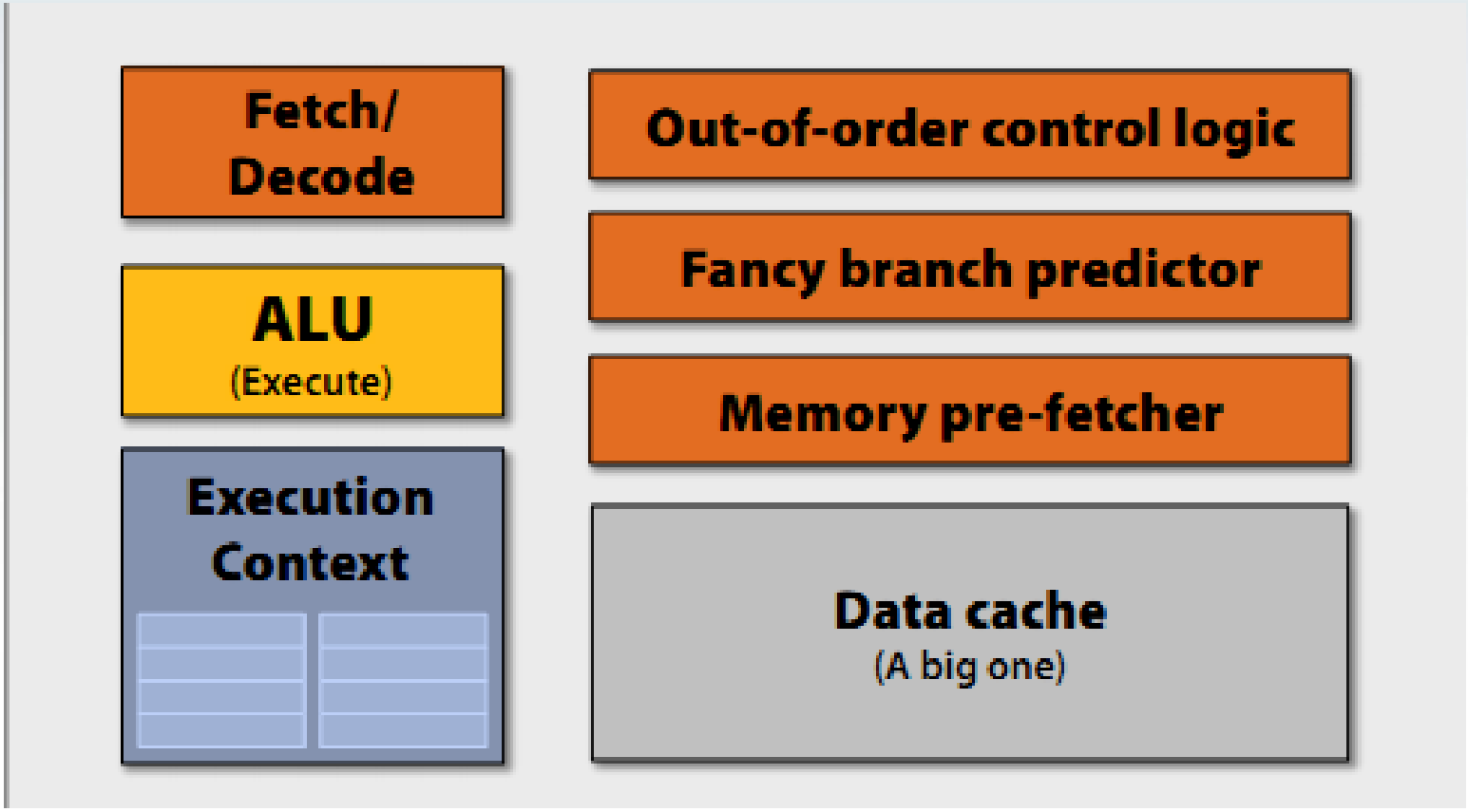
Background (Data Parallelism)

- Distribute data among different processing nodes.
- Each Node executes same instructions but operate on different data set.



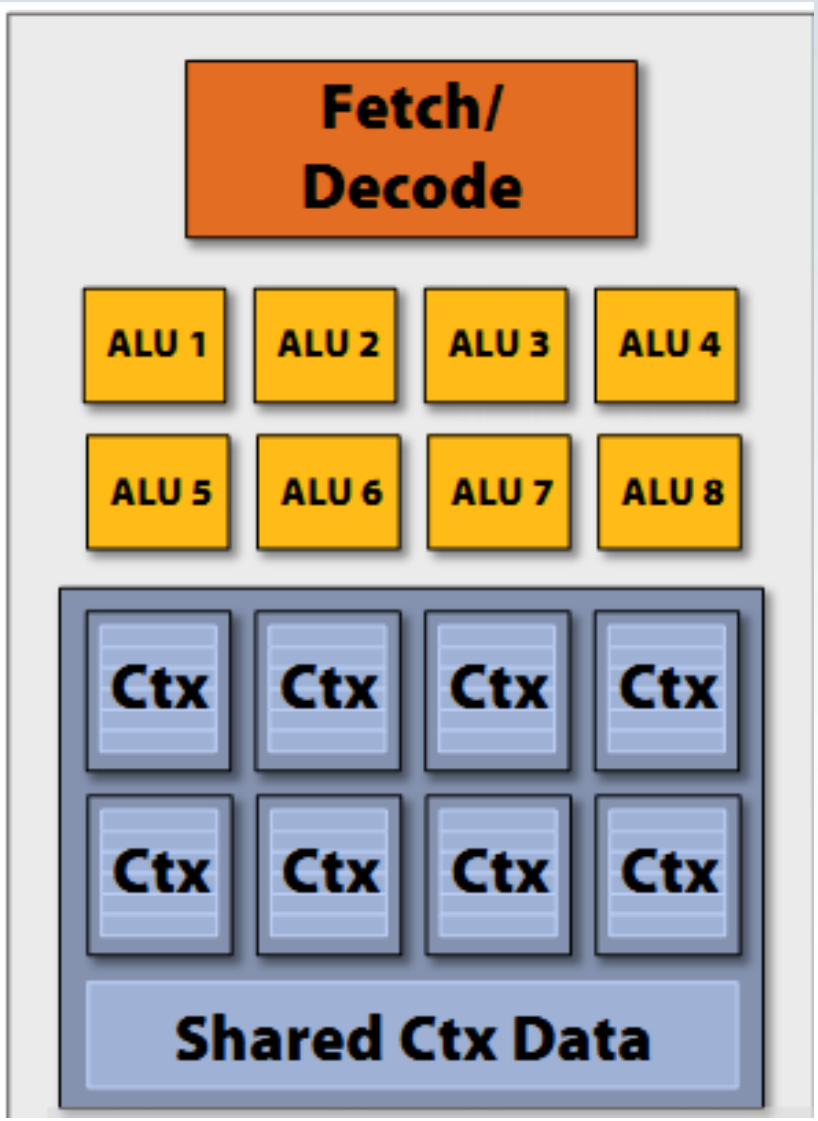
Background (Microprocessor)

- Extra Hardware to hide memory latency and provide instruction level parallelism.



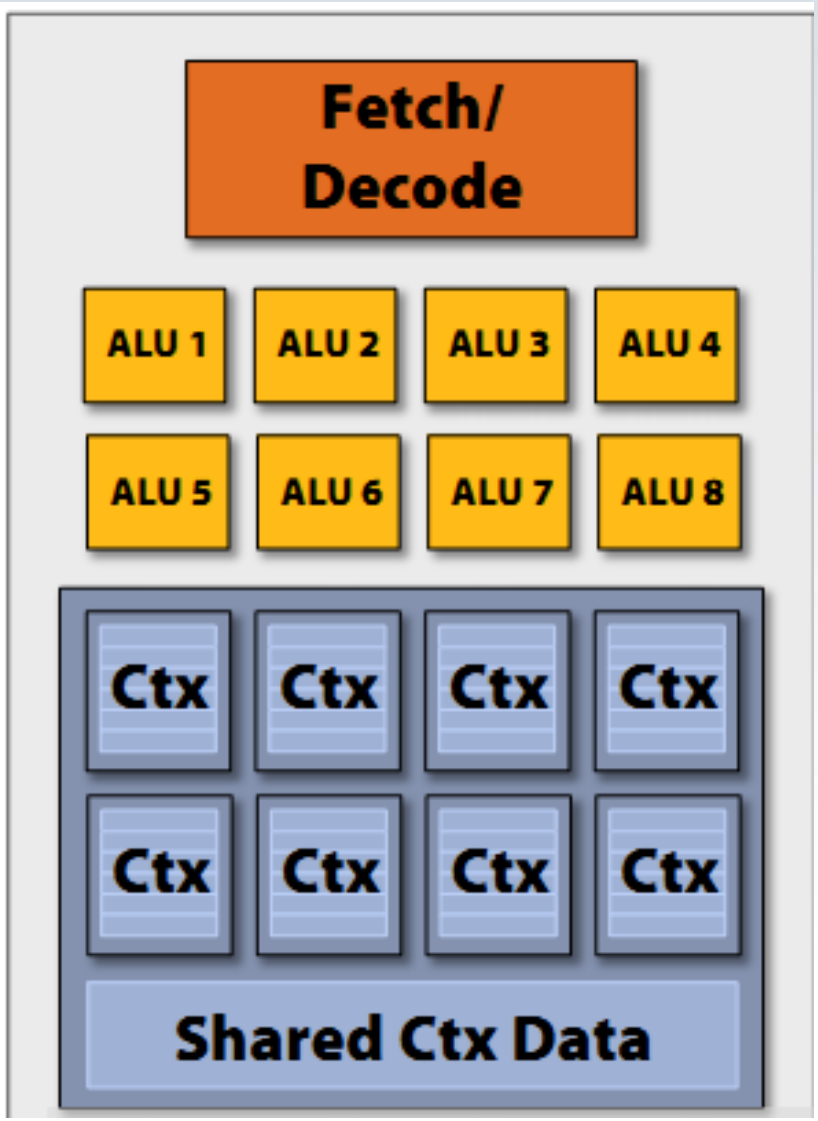
Background (GPU)

- Remove the extra hardware.
- More ALU's and few control Blocks.
- A single Instruction dispatched to all the ALU's



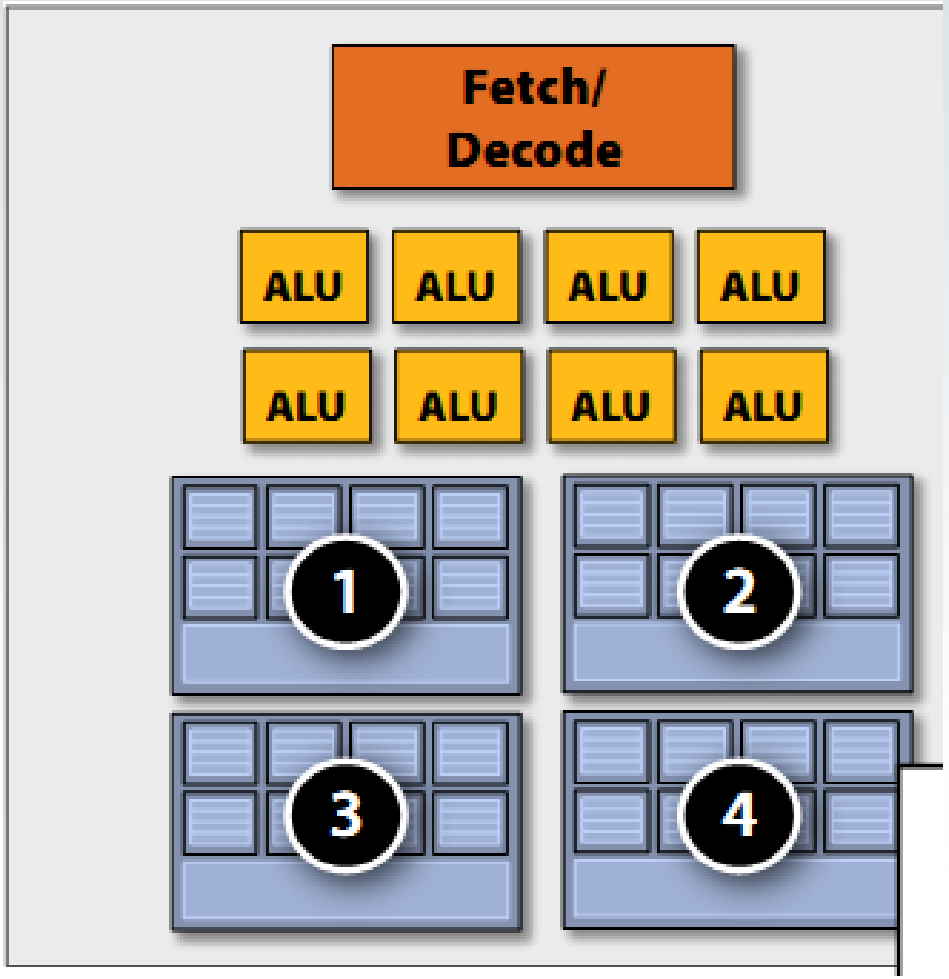
Background (GPU)

- Data Parallel Execution.
- Each ALU executes same instruction but operate on different data.
- Instruction set reduced.
- Memory Latency??

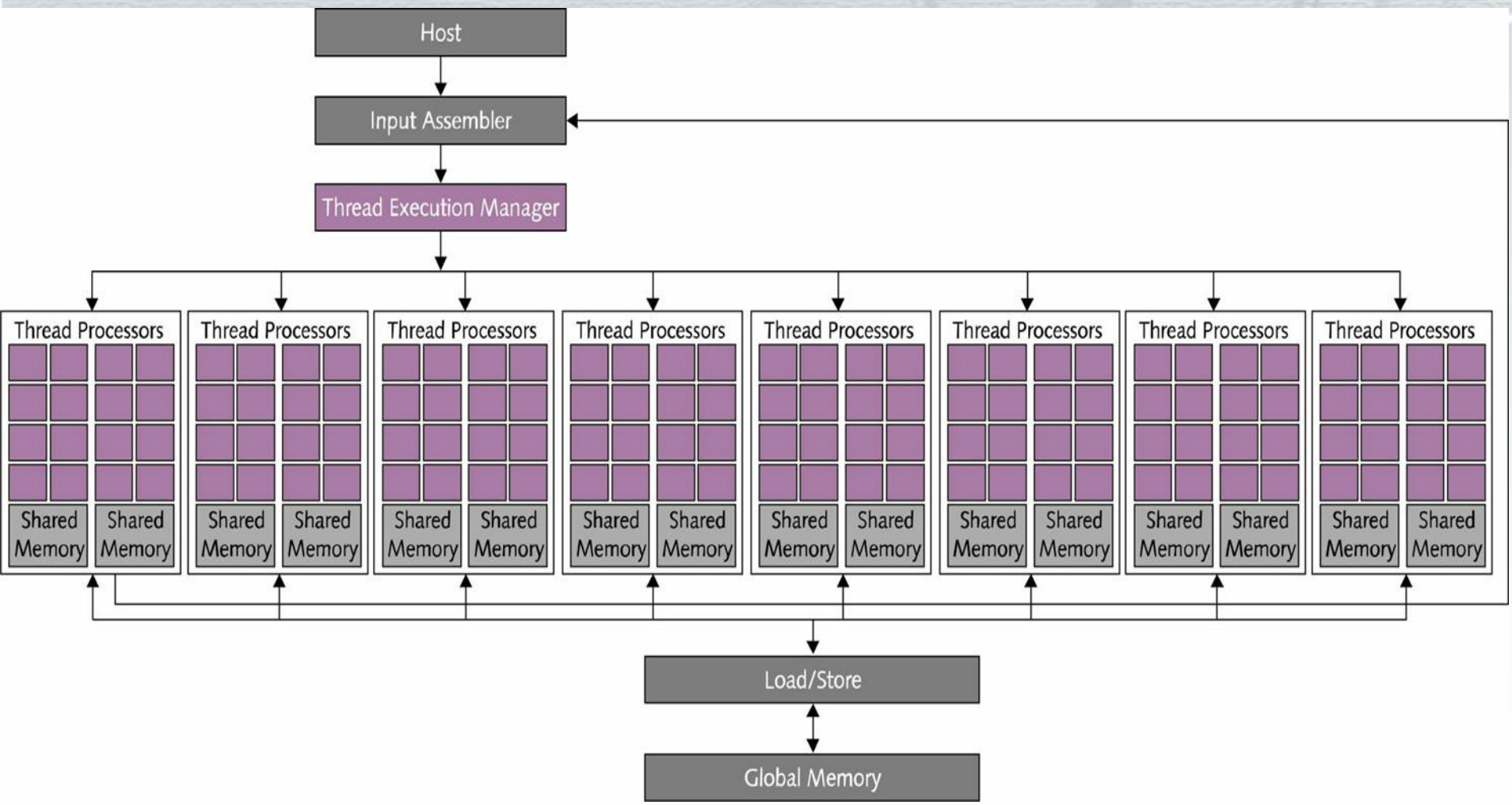


Background (GPU)

- Hide memory latency by interleaving the execution of group(warp) of threads



Background (GPU)



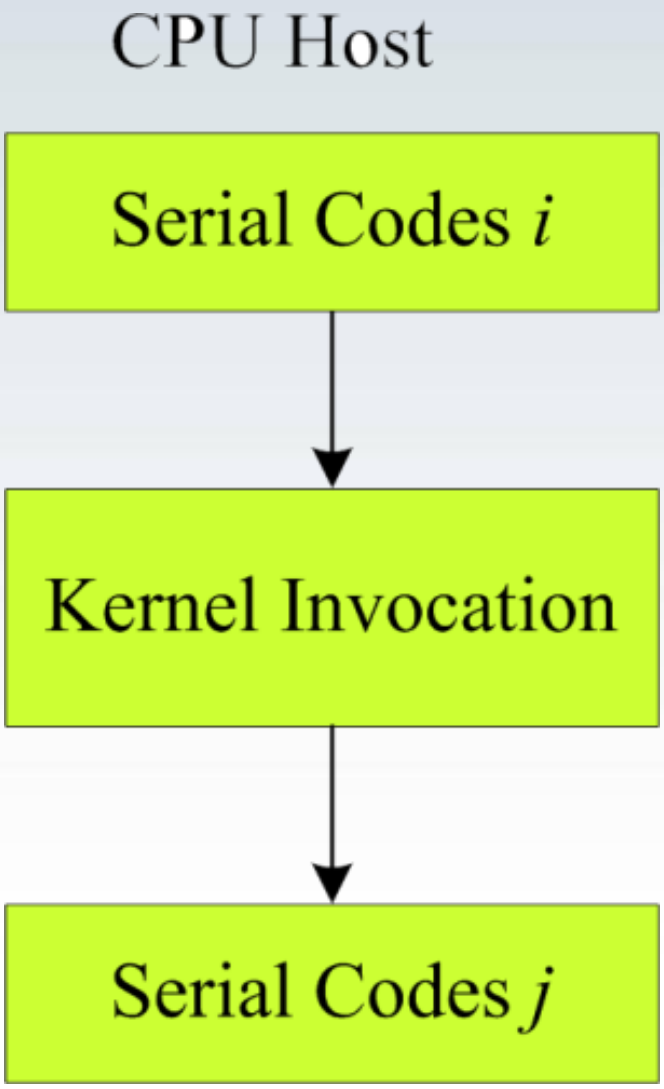
Background (GPU)

- SIMD Architecture
- High throughput.
- Cost effective.
- Used Mainly For Graphics Processing.
- Provide Massively Parallel computing.

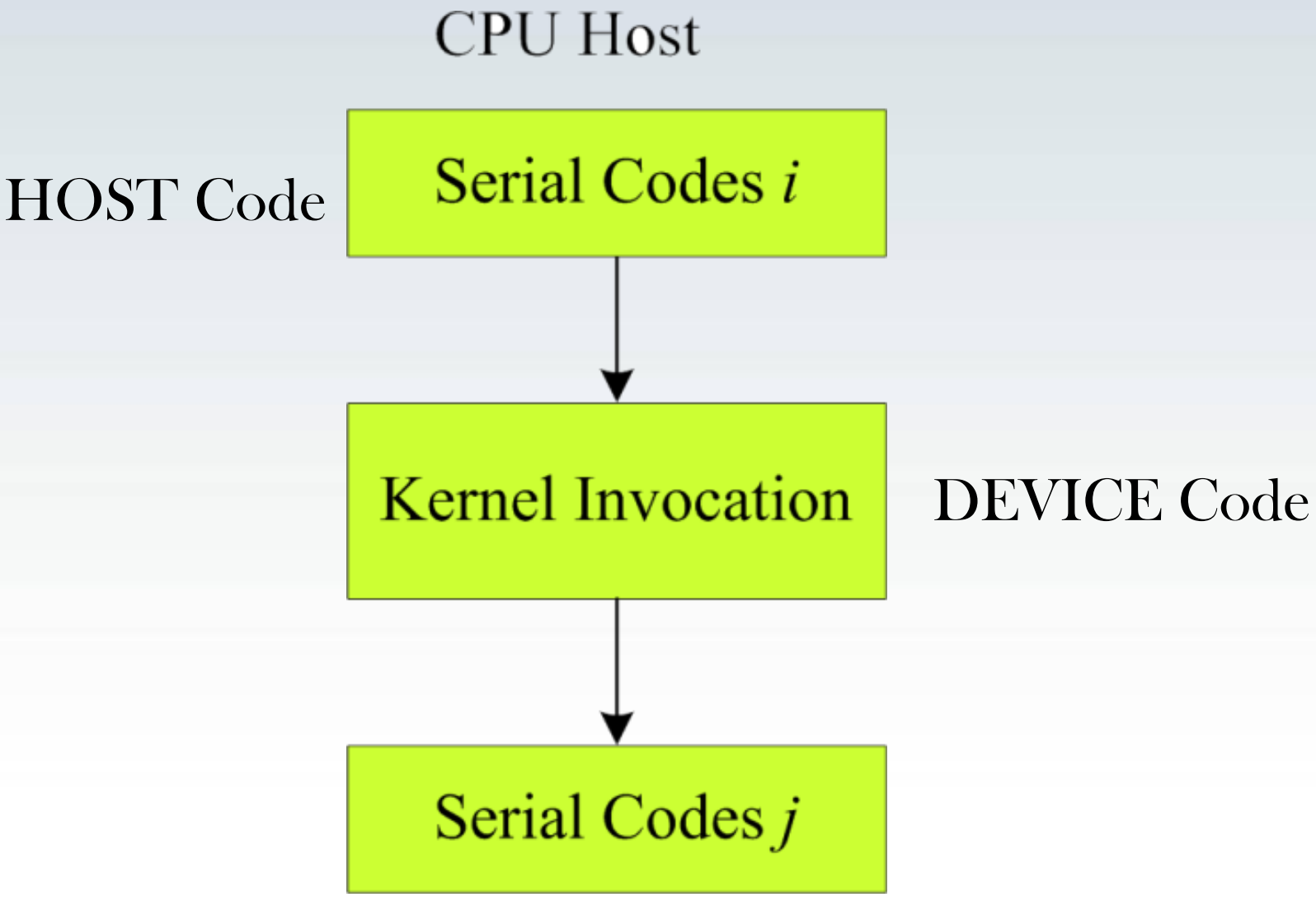
Background (CUDA)

- Compute Unified Device Architecture(CUDA).
- A new parallel programming model developed by NVIDIA.
- Use GPU for general purpose computing.
- GPGPU's
- C/C++ software development tools, function libraries and a hardware abstraction mechanism

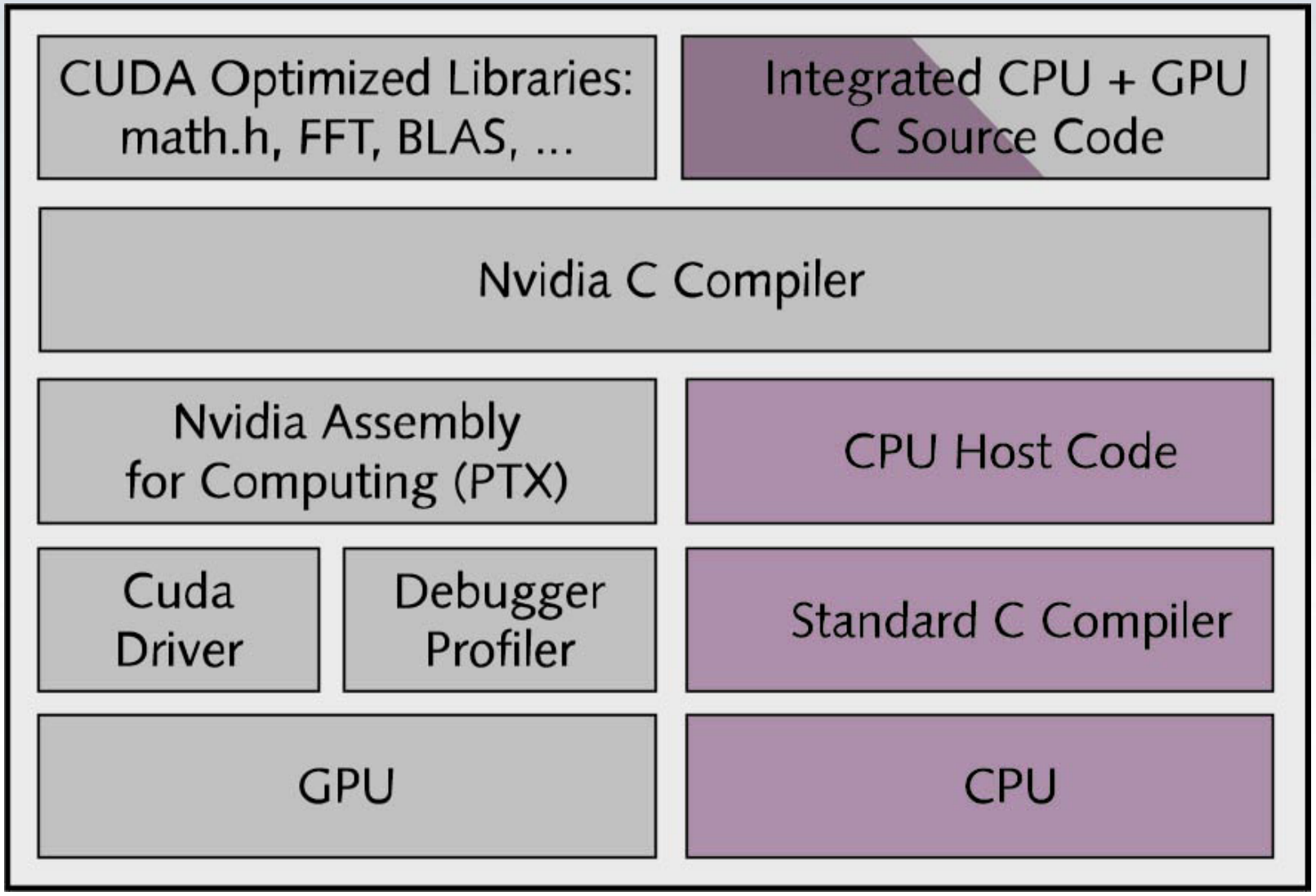
Background (CUDA)



Background (CUDA)



Background (CUDA)



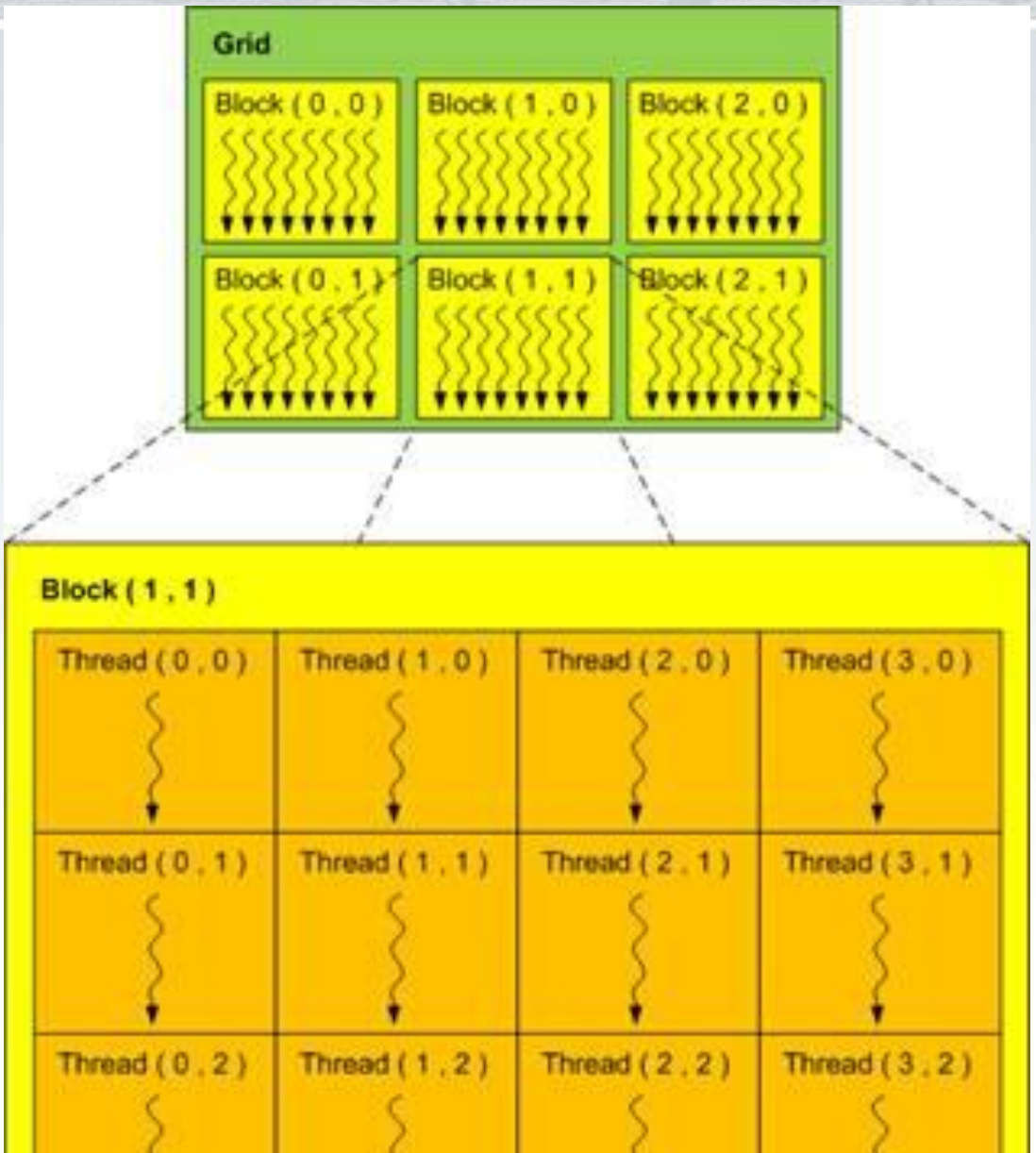
Background (CUDA Thread Model)

- Each thread is assigned to a single ALU.
- Warp is fundamental execution unit in CUDA. Consists of groups of 32 threads.
- Threads can be grouped.
- Number of threads are specified when invoking a kernel.

Background (CUDA Thread Model)

Threads group together to form Blocks

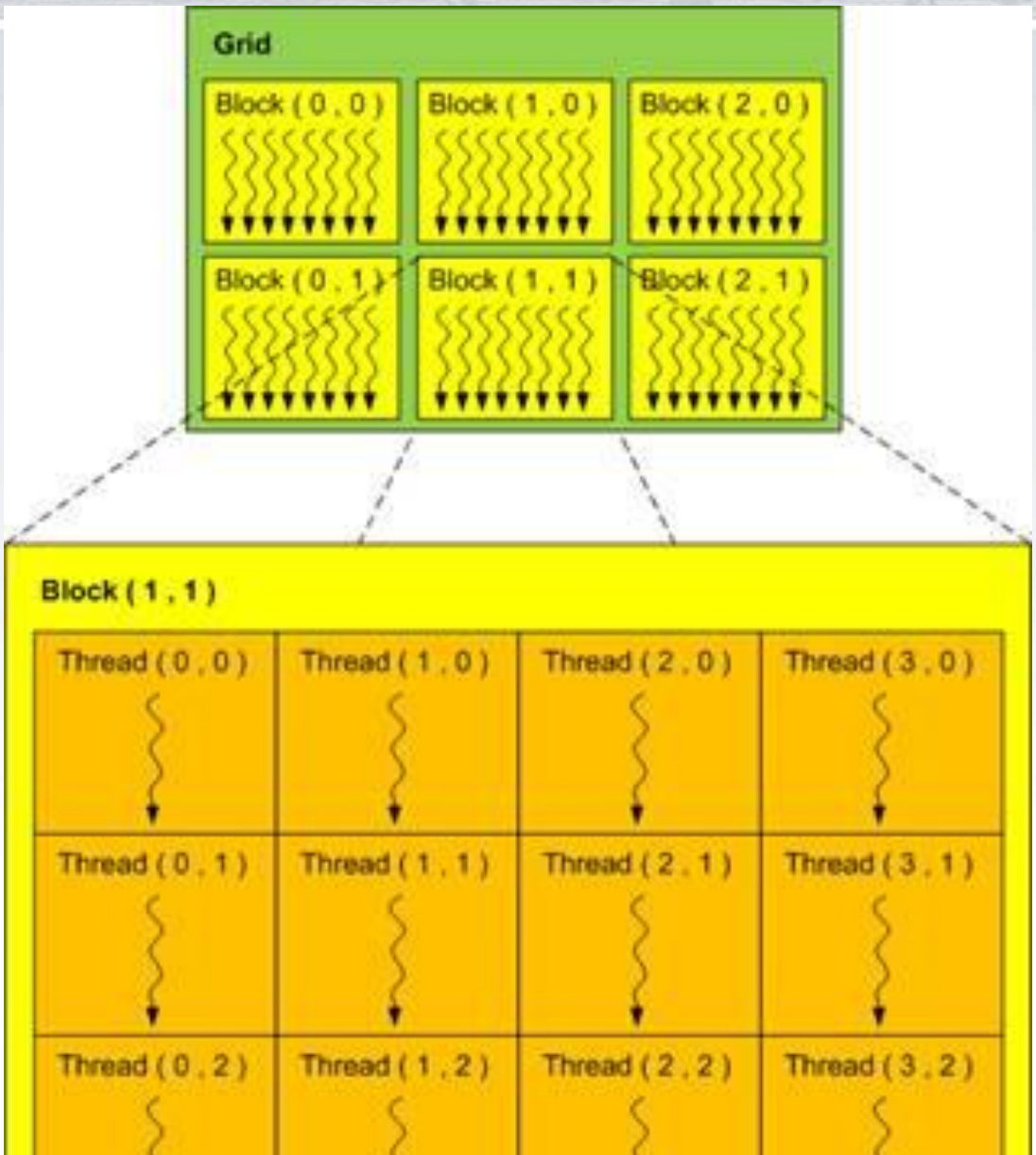
Blocks Can be 1D, 2D or 3D



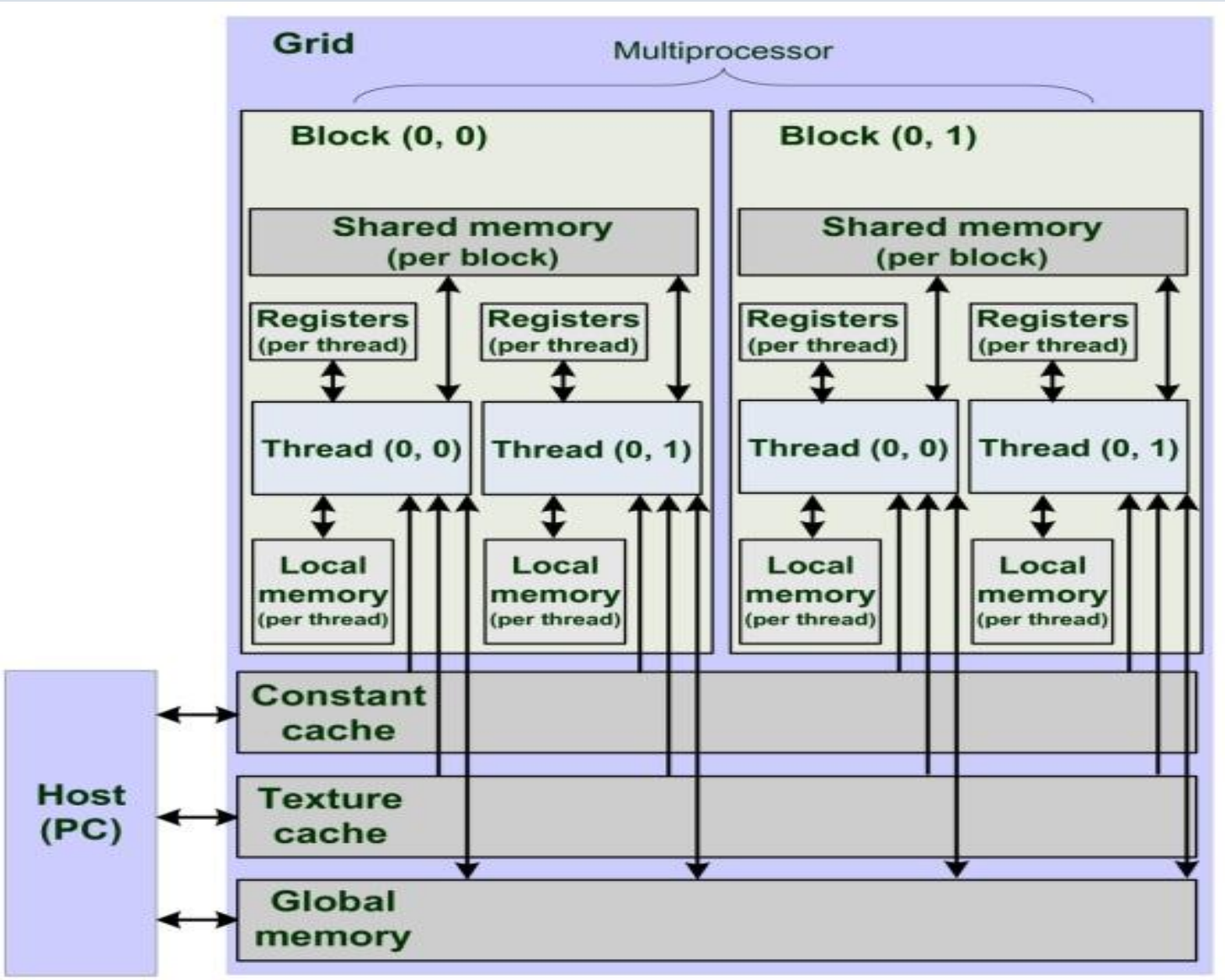
Background (CUDA Thread Model)

Blocks group together to form Grid

Grid Can be 1D, 2D or 3D



Background (CUDA Memory Model)



Background (CUDA Execution Model)

- Define a kernel.
  - Use __global__ declaration specifier
  - Executed on device and called from host

- Allocate memory on device(GPU) using `cudaMalloc()`
- Copy Data to GPU Global Memory.
- Invoke the kernel.
- Each thread of kernel is given a unique thread ID, accessible within the kernel via built-in variable threadIdx

Background (CUDA Execution Model)

```
__global__ void VecAdd(float* A, float* B, float* C){
int i = threadIdx.x;
//Each threads performs one pair-wise addition
C[i] = A[i] + B[i];
}
```

```
int main(){
// Kernel invocation
VecAdd<<<1, N>>>(A, B, C);
}
```

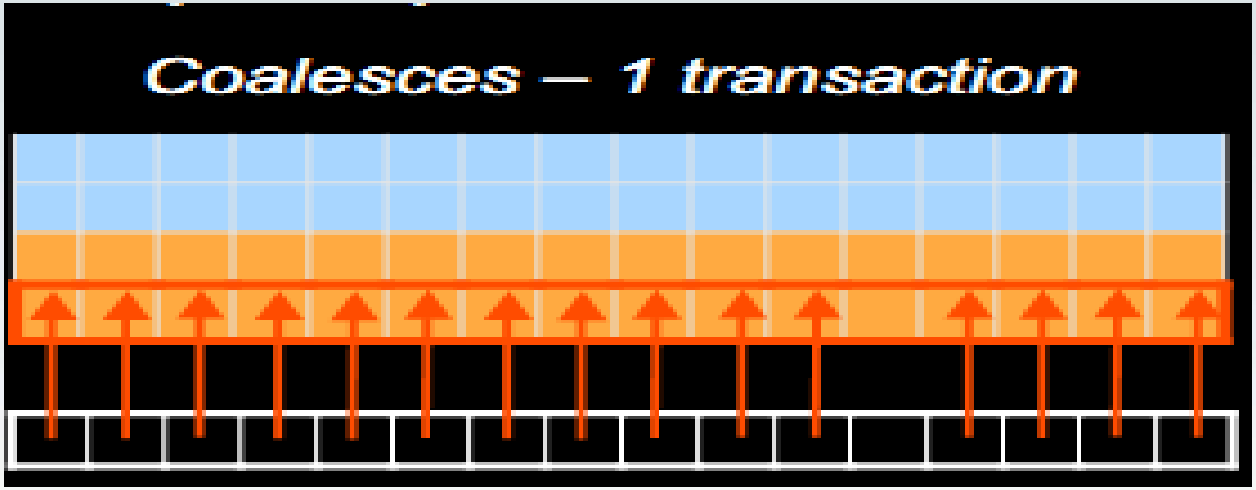
Background (CUDA Tiling)

- Global Memory access is slow. Shared memory access is fast
- Minimize Global Memory access.
- Partition the data into subsets called tiles so that each tile fits into the shared memory
- Increases the computation to memory ratio

Background (CUDA Coalesced Memory Access)

- GPU memory controller granularity is 64 or 128 bytes.
- When a thread loads a float (4 bytes),the controller loads 64 bytes, throws 60 bytes away.
- If threads read consecutive memory location from global memory?
- Access Becomes Coalesced.

Background (CUDA Coalesced Memory Access)



One read operation will read data for 16 threads from Global Memory.

Scalpel

- High speed file carving tool for digital forensics.
- Works by processing disk images in two passes.
- First pass
  - Reads the input in 10 MB chunks.
  - Conduct Header/Footer Searches
  - Schedule is constructed for each chunk.
- Second Pass
  - Performs the Carving operation based on the schedule constructed.

Implementation (Multi Core)

- Multi Core Machine
  - Spawn a thread for each file carving rule.
  - Each thread waits for data and performs the header/footer search once its available.
  - No overlapping disk I/O with computation
  - No parallelization of data.

Implementation (GPU)

- GPU
  - Kernel Operates on 10 MB data.
  - Carving Rules are copied to Constant Memory.
  - 65536 threads are created (Block Dimension or Grid Dimension not provided.)
  - Search is performed by each thread.
  - Discovered locations copied back to Host.
- Data Sets
  - 20 GB, 100GB and 500 GB Disk Images.
  - No information regarding fragmentation is provided.

Results (20 GB Disk Image)

Table 1 – Results for carving 20 GB disk image on dual processor, dual core Sun Ultra 40 (2.6 GHz AMD Opteron 2218 processors, 16 GB RAM)

Scalpel 1.60 "vanilla" (s)	2672
Scalpel 1.60 "new q" (s)	1784
Scalpel 1.70MT-multicore (s)	1054
Scalpel 1.70MT-gpu-0.20 (s)	860

Thirty file types, ~3 M files carved. Each result is the average of multiple, sequential runs.

- Inefficiency in scalpel 1.60 second pass is corrected in scalpel 1.60 "new q"
- No information about the inefficiency provided.

Results (100 GB Disk Image)

Table 2 – Results for carving 100 GB disk image on dual processor, dual core Sun Ultra 40 (2.6 GHz AMD Opteron 2218 processors, 16 GB RAM)	
Scalpel 1.60 "vanilla" (s)	13,067
Scalpel 1.60 "new q" (s)	8725
Scalpel 1.70MT-multicore (s)	4958
Scalpel 1.70MT-gpu-0.20 (s)	5185

Thirty file types, ~15 M files carved. Each result is the average of multiple, sequential runs.

- Multi core implementation gave better results.
- More threads are created for the GPU.

Results (Massive Threading)

Table 3 – Results for carving 20 GB disk image on single processor, dual core Dell XPS 710 (2.4 GHz Core2Duo processor, 4 GB RAM)

Scalpel 1.60 "new q" (s)	1260
Scalpel 1.70MT-multicore (s)	861
Scalpel 1.70MT-gpu-0.20 (s)	686
Scalpel 1.70MT-gpu-0.30 (s)	446

Thirty file types, ~3 M files carved. Each result is the average of multiple, sequential runs.

Table 4 – Results for carving 100 GB disk image on single processor, dual core Dell XPS 710 (2.4 GHz Core2Duo processor, 4 GB RAM)

Scalpel 1.60 "new q" (s)	7105
Scalpel 1.70MT-multicore (s)	5096
Scalpel 1.70MT-gpu-0.20 (s)	4192
Scalpel 1.70MT-gpu-0.30 (s)	3198

Thirty file types, ~15 M files carved. Each result is the average of multiple, sequential runs.

- No Information regarding the time to get data from Host → GPU and GPU → Host is shown.

Results(Discrepancy in Results)

Table 2 – Results for carving 100 GB disk image on dual processor, dual core Sun Ultra 40 (2.6 GHz AMD Opteron 2218 processors, 16 GB RAM)

Scalpel 1.60 "vanilla" (s)	13,067
Scalpel 1.60 "new q" (s)	8725
Scalpel 1.70MT-multicore (s)	4958
Scalpel 1.70MT-gpu-0.20 (s)	5185

Thirty file types, ~15 M files carved. Each result is the average of multiple, sequential runs.

Table 4 – Results for carving 100 GB disk image on single processor, dual core Dell XPS 710 (2.4 GHz Core2Duo processor, 4 GB RAM)

Scalpel 1.60 "new q" (s)	7105
Scalpel 1.70MT-multicore (s)	5096
Scalpel 1.70MT-gpu-0.20 (s)	4192
Scalpel 1.70MT-gpu-0.30 (s)	3198

Thirty file types, ~15 M files carved. Each result is the average of multiple, sequential runs.

- Changing machine improved performance of original GPU implementation???

Results (500 GB Disk Image)

Table 5 – Results for carving 500 GB disk image on single processor, dual core Dell XPS 710 (2.4 GHz Core2Duo processor, 4 GB RAM)	
Scalpel 1.60 "new q" (s)	9946
Scalpel 1.70MT-multicore (s)	9922
Scalpel 1.70MT-gpu-0.30 (s)	12,168
Two file types, ~73,000 files carved.	

- Increasing disk size reduces the performance.
- Memory Transfer Overhead.

Conclusion

- GPU's can provide performance boost in Digital forensics application but only for small data sets.
- Memory transfer over head are a major issue affecting performance for large data sets.
- No information regarding correctness are comparison with the serial implementation is provided.
- Implementing Compression scheme to hide memory transfer bottlenecks??? Lossless compression has very few hard from data parallel point of view.