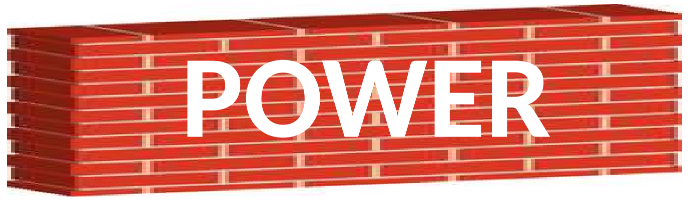
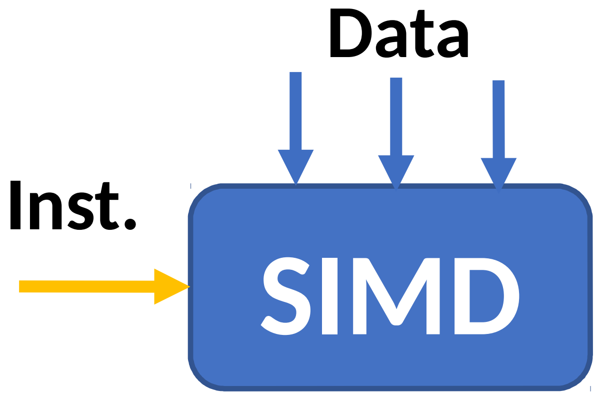
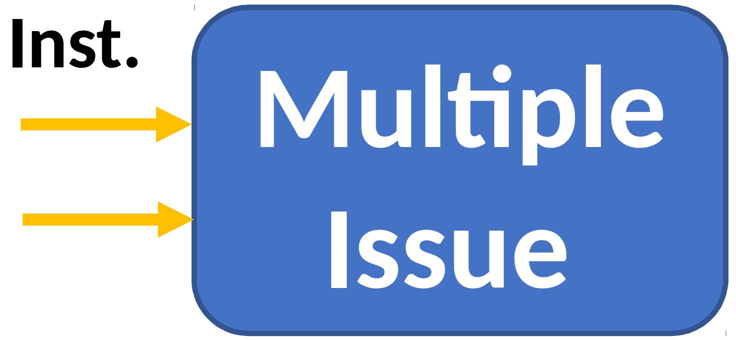
Where are we?



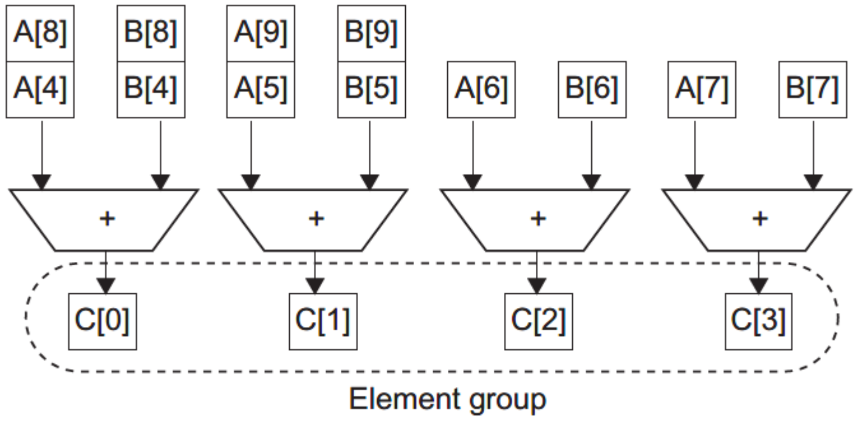


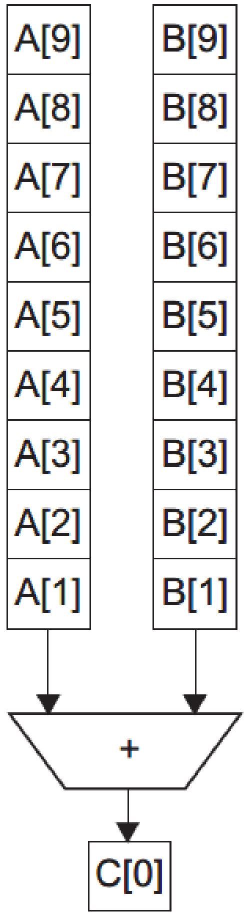
Why parallel programming is challenging?



Vector Processors

- Vector architectures represents an extension for SIMD
  - Operates on vector data (not scalar)
  - Use vector registers (e.g., 64 64-bit elements)
    1. Transfer data from memory to registers
    2. Process the data in pipelined execution SIDM units
    3. Store results to memory





Vector lanes: operations are always on ***corresponding*** elements in different register vectors.

Vector Instructions

- Vector instructions
  - lv, sv: load/store vector
  - addv.d: add vectors of double
  - addvs.d: add scalar to each element of vector of double
- Significantly reduces instruction-fetch bandwidth
  - One instruction works on up to 64 registers (MIPS vector)
  - Special register for number of processed elements
- Fewer pipelining hazards
  - Per vector vs per element
- Consumes less energy but Memory could be a bottleneck

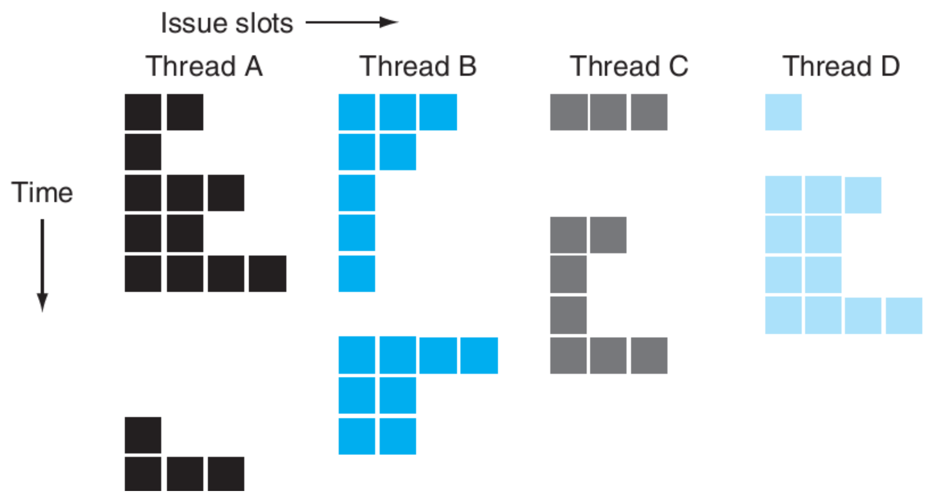
Hardware Multithreading

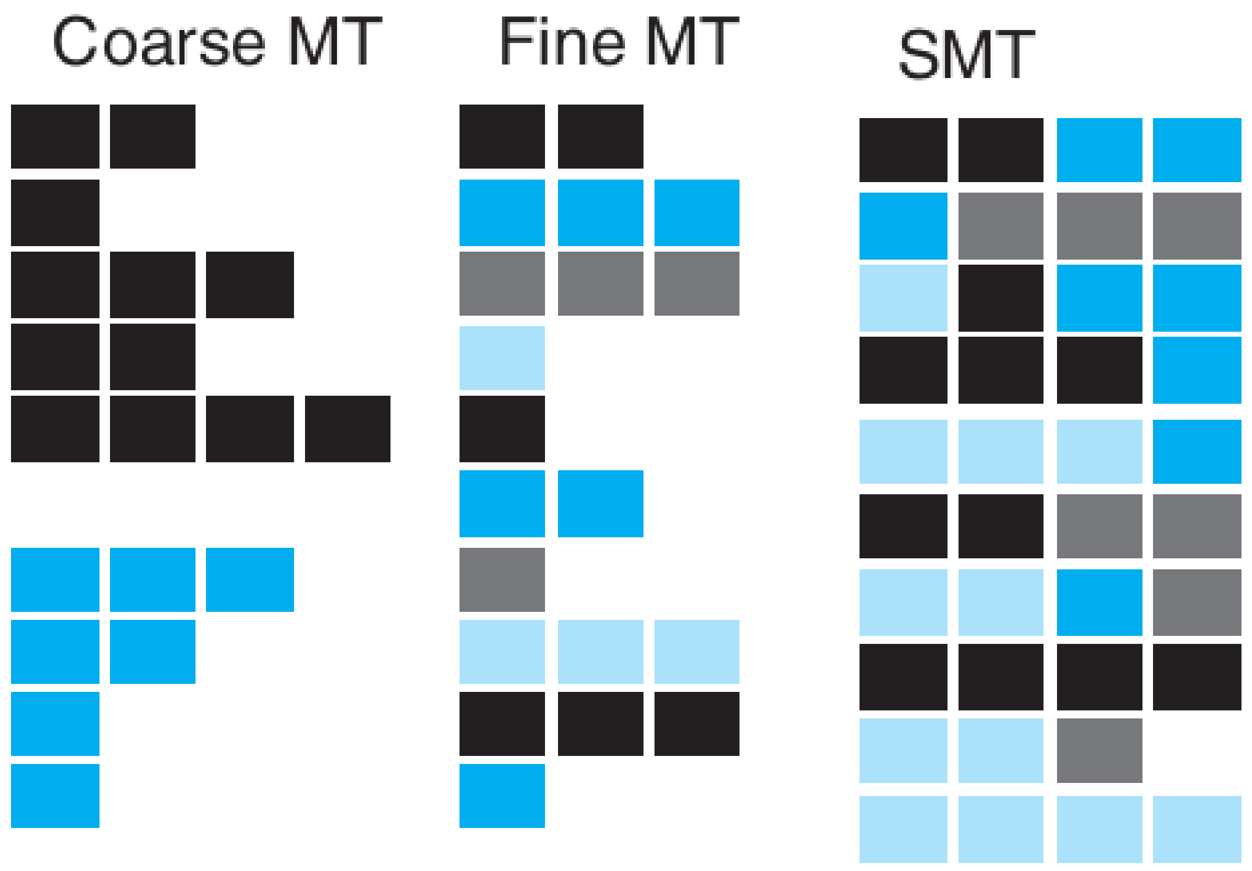
A process includes one or more threads, the address space, and the operating system state. Hence, a process switch usually invokes the operating system, but not a thread switch

A thread includes the program counter, the register state, and the stack. It is a lightweight process; whereas threads commonly share a single address space, processes don't

- Hardware multithreading allows multiple threads to share the functional units of a single processor
- Hardware multithreading improves processor utilization

Hardware Multithreading Approaches



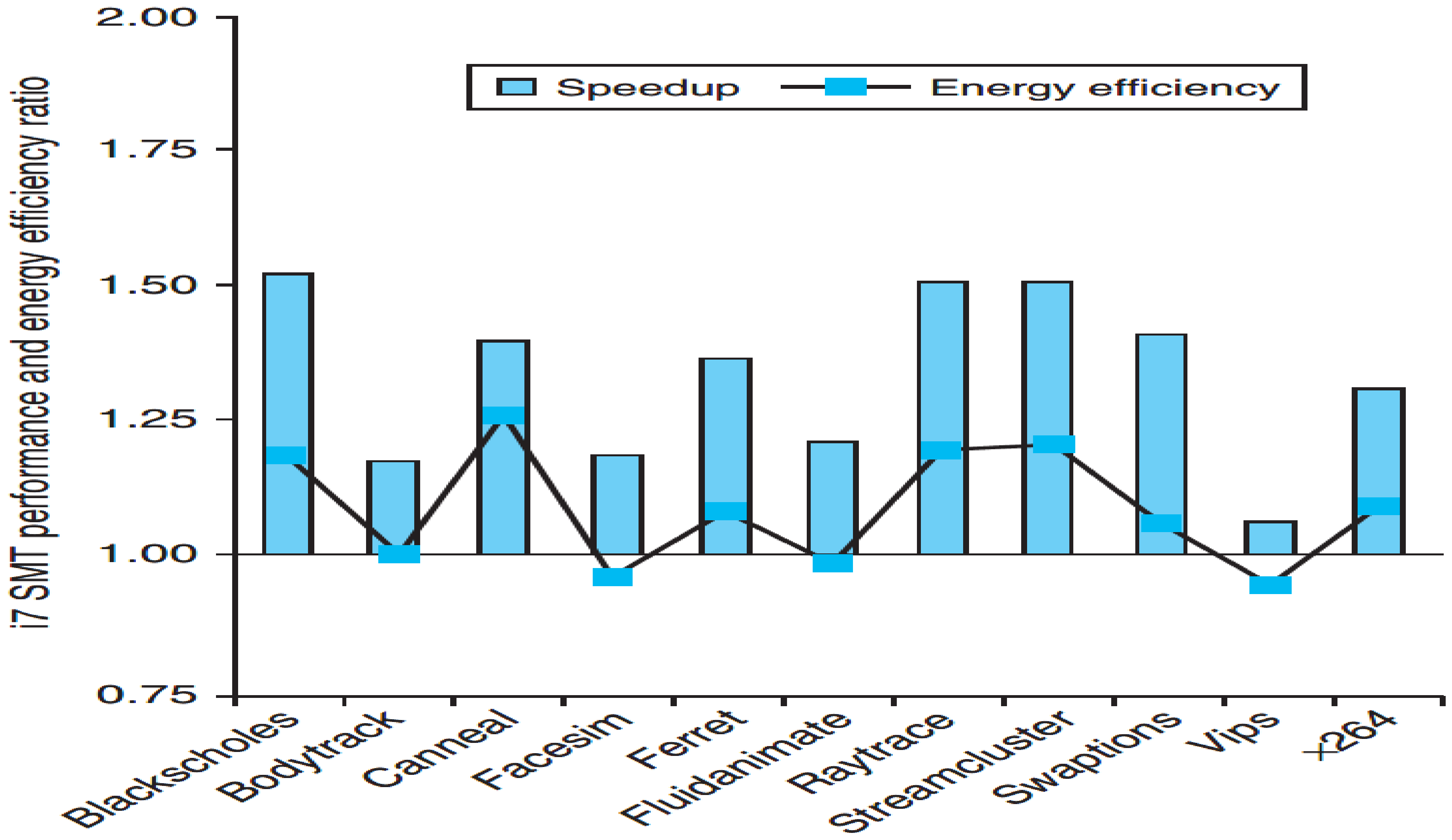


- Coarse-grain multithreading
- Fine-grain multithreading
- Simultaneous Multithreading
  - Exploits thread + instruction parallelism

Hardware Multithreading Approaches

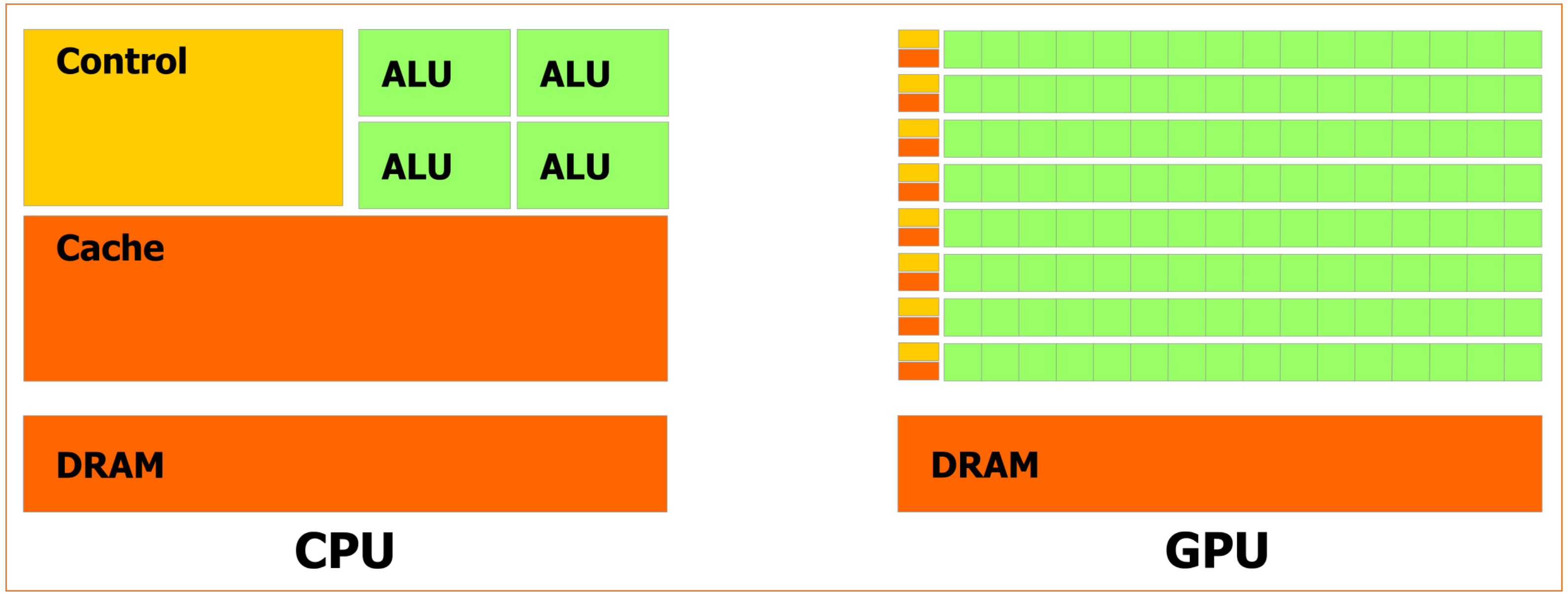
- **Coarse-grain multithreading**
  - Only switch on long stall (e.g., L2-cache miss)
  - Simplifies hardware, but doesn't hide short stalls (eg, data hazards)
- **Fine-grain multithreading**
  - Switch threads after each instruction
  - Interleave instruction execution
  - If one thread stalls, others are executed
- **Simultaneous Multithreading (SMT)**
  - Schedule instructions from multiple threads in the same clock cycle
  - Instructions from independent threads execute when function units are available
  - Within threads, dependencies handled by scheduling and register renaming

SMT Performance



Graphics Processor Unit (GPU)

- Very important in gaming markets (gaining grounds in other domains)
- **GPUs vs. CPUs**
  - GPUs do not need be able to perform all the tasks of a CPUs
    - accelerators that supplement a CPU
  - GPUs can accommodate ***many parallel processors (MIMD)*** as well as many threads
    - Hundreds of parallel floating-point units
  - GPUs ***rely on hardware multithreading*** to hide the latency to memory (not memory hierarchy)
    - GPU executes hundreds or thousands of threads while fetching independent data
  - GPU memory is oriented toward bandwidth rather than latency
    - Special DRAM designs



The chip area is used for caches in CPU is spent instead on computing resources and on the large number of registers to hold the state of the many threads of SIMD instructions

GPU Architectures

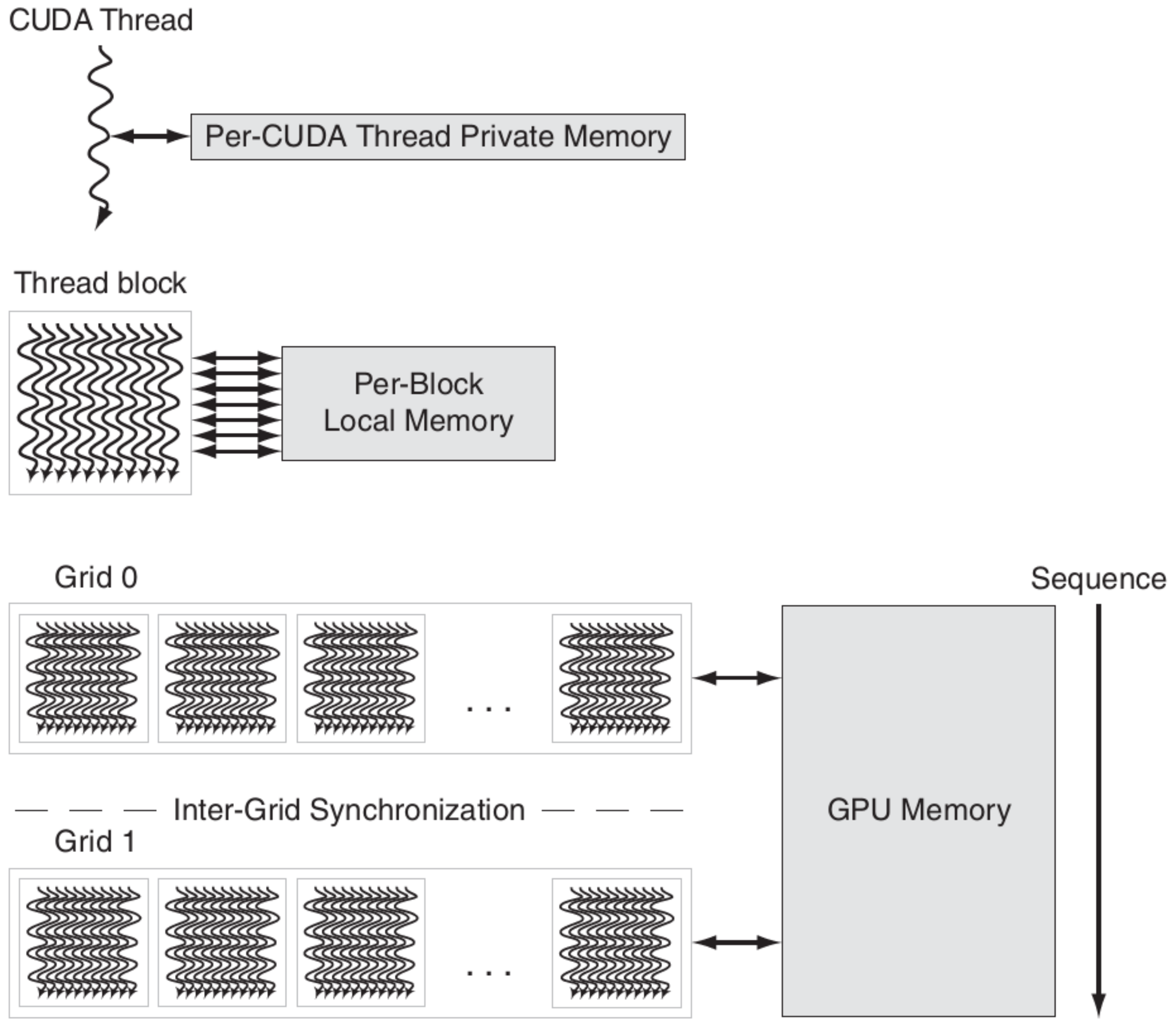
- Trend toward general purpose GPUs
  - Heterogeneous CPU/GPU systems
  - CPU for sequential code, GPU for parallel code
- Programming languages/APIs
  - DirectX, OpenGL
  - C for Graphics (Cg), High Level Shader Language (HLSL)
  - Compute Unified Device Architecture (CUDA)

```
for (int i=0;i<n;i++)
{
C[i] = A[i]+B[i];
}
```

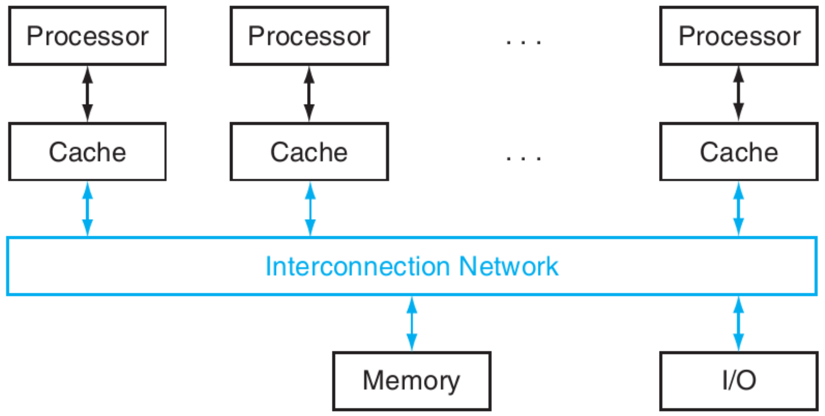
```
VecAdd(A,B,C)
{
int i =threadIdx …
C[i] = A[i]+B[i]
}
VecAdd<<<n>>>(A,B,C)
```

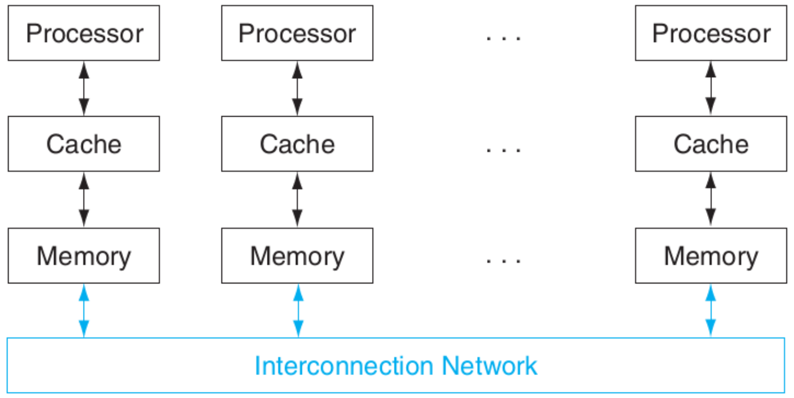
GPU Operation

- GPU hardware has two levels of hardware schedulers:
  - ***A thread Block Scheduler*** assigns blocks of threads to multithreaded SIMD processors
  - ***An SIMD Thread Scheduler*** schedules when SIMD threads should run



Memory Organization for Multiple Processors





- Shared address space
- Requires ***Synchronization*** to coordinating the different processors
- E.g., Multicore processors.

- Multiple address spaces
- Requires ***message passing*** for coordination of different processors
- E.g. Data centres

Suggested Reading

- Sections 6.4 – 6.7
  - These sections are partially covered.