Introduction

- **Pipelining**
  - Executing multiple instructions in parallel
- **Instruction Level Parallelism (ILP)**
  - Executing multiple instructions in each stage

- Parallel software is the problem
  - Otherwise, just use a faster uniprocessor
- Difficulties
  - Partitioning
  - Coordination
  - Communications overhead
  - Balancing

Issues

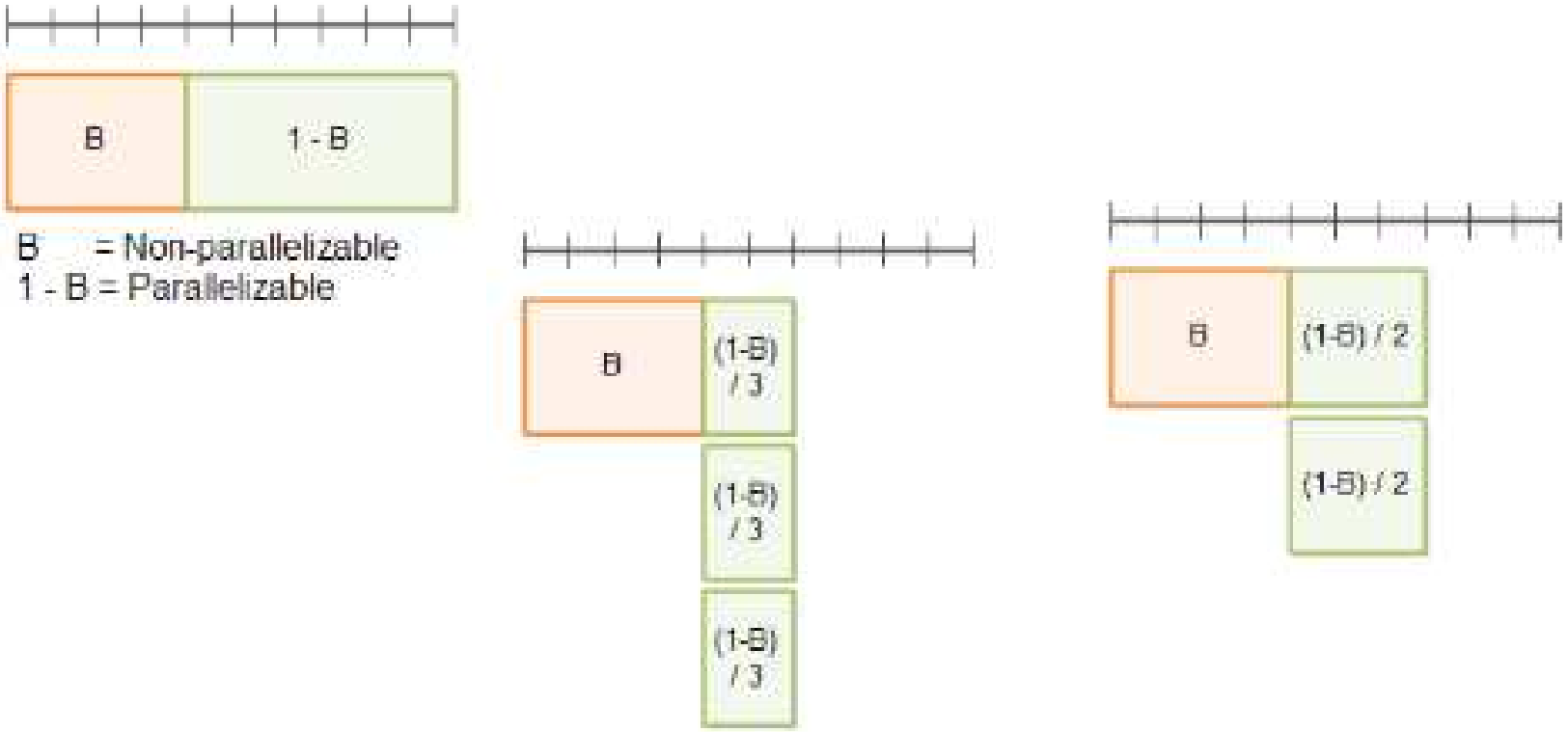
- Multiple Issue : set of instructions that issues together in one clock cycle
  - **Static multiple issue**
    - **Compiler groups**
  - **Dynamic multiple issue**
    - **CPU examines**
- Speculation
  - “Guess” what to do with an instruction
  - Start operation as soon as possible
  - Check whether guess was right
    - If so, complete the operation
    - If not, roll-back and do the right thing
  - Examples
    - Speculate on branch outcome
      - Roll back if path taken is different
    - Speculate on load
      - Roll back if location is updated

Parallelism

- Goal: connecting multiple computers to get higher performance
  - Task-level (process-level) parallelism
    - High throughput for independent jobs
  - Parallel processing program
    - Single program run on multiple processors
  - Multicore microprocessors
    - Chips with multiple processors (cores)
- Sequential/concurrent software can run on serial/parallel hardware

Amdahl's law

- The performance enhancement of an improvement is limited by how much the improved feature is used. In other words: Don’t expect an enhancement proportional to how much you enhanced something.



Amdahl's law

- Predict the theoretical maximum speedup for program processing using multiple processors.
  - T = Total time of serial execution
  - p = part parallizable
  - 1 - p = non parallizable
- Serial execution
  - Total time= T(1-p) + Tp
- s be the number of threads or CPUs

$$T(s) = (1-p)T + \frac{p}{s}T. \qquad S_{\text{latency}}(s) = \frac{T}{T(s)} = \frac{1}{1-p+\frac{p}{s}}$$

Scaling Example

- Workload: sum of 10 scalars, and 10 × 10 matrix sum
  - Speed up from 10 to 100 processors
- Single processor:
  - Time = (10 + 100) × t_{add}
- 10 processors
  - Time = 10 × t_{add} + 100/10 × t_{add} = 20 × t_{add}
  - Speedup = 110 t_{add} /20 t_{add} = 5.5
- 100 processors
  - Time = 10 × t_{add} + 100/100 × t_{add} = 11 × t_{add}
  - Speedup = 110 t_{add} /11 t_{add} = 10

$$S_{\text{latency}}(s) = \frac{T}{T(s)} = \frac{1}{1 - p + \frac{p}{s}}$$

- Assumes load can be balanced across processors

Speed-up Challenges

- Strong Scaling:
  - increasing parallelism in **hardware** achieves increased speedup
  - problem size fixed
- Weak Scaling:
  - increased parallelism is achieved only by increasing the **problem size with the hardware** parallelism size in
  - problem size proportional to number of processors
    - 10 processors, 10 × 10 matrix
      - Time = $20 \times t_{add}$
    - 100 processors, 32 × 32 matrix
      - Time = $10 \times t_{add} + 1000/100 \times t_{add} = 20 \times t_{add}$

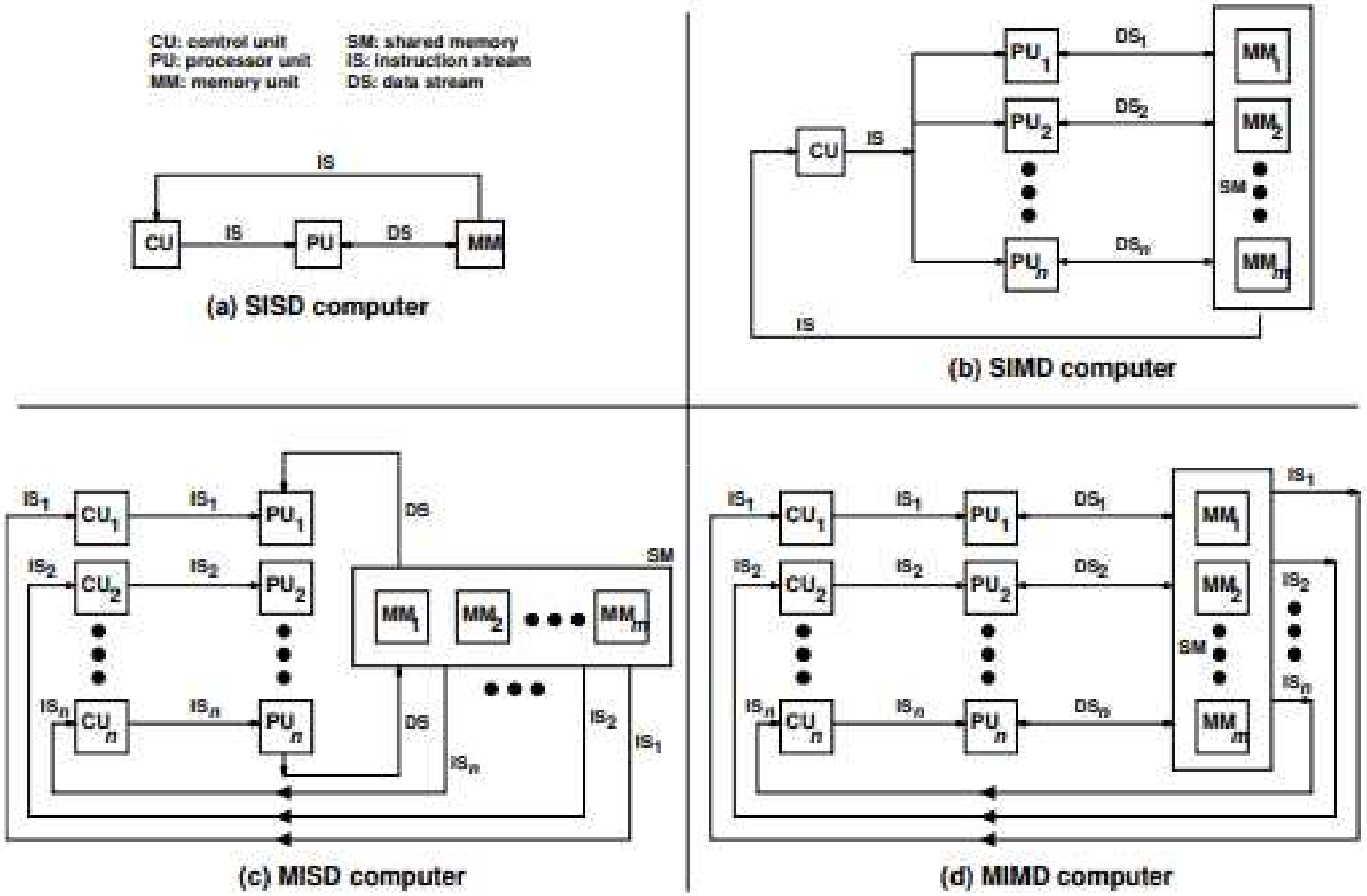
Instruction and Data Streams

- Flynn's taxonomy

		Data Streams	
		Single	Multiple
Instruction Streams	Single	**SISD**: Intel Pentium 4	**SIMD**: SSE instructions of x86
	Multiple	**MISD**: No examples today	**MIMD**: Intel Xeon e5345

SPMD – Single Program Multiple Data
MPMD- Multiple Program Multiple Data

Flynn’s Classification of Parallelism



Single program, multiple data (SPMD): Single program run on every node on a MIMD style platform.

Multimedia Extensions (MMX)

- *Streaming SIMD Extensions (SSE)*
- *Advanced Vector Extensions (AVX)*
- Subword parallelism

Vector Processors

- Highly pipelined function units
- SIMD
- data-level parallelism
- Stream data from/to vector registers to units
  - Data collected from memory into registers
  - Results stored from registers to memory
- Example: Vector extension to MIPS
  - 32 × 64-element registers (64-bit elements)
  - 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

Vector Processors

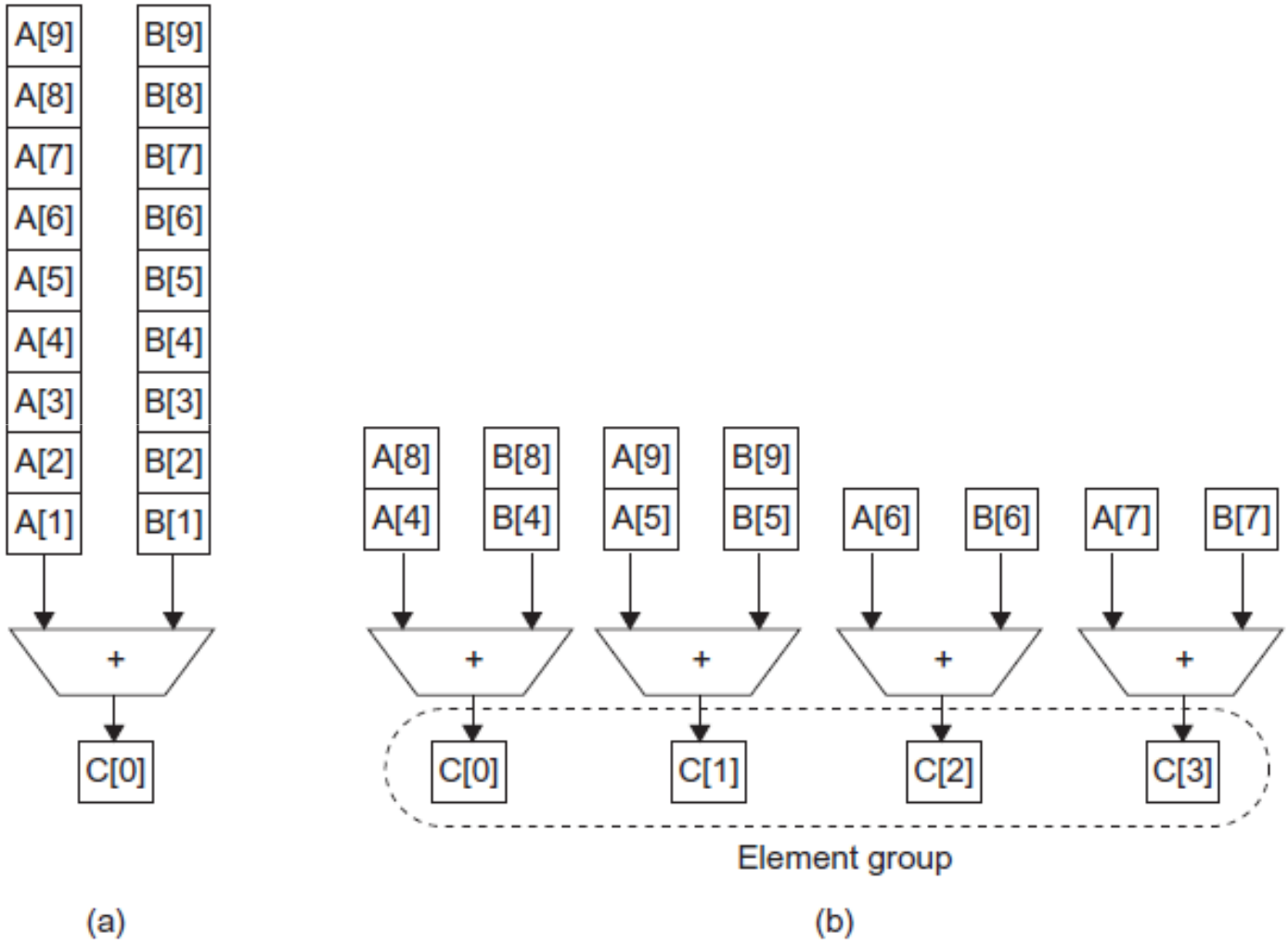
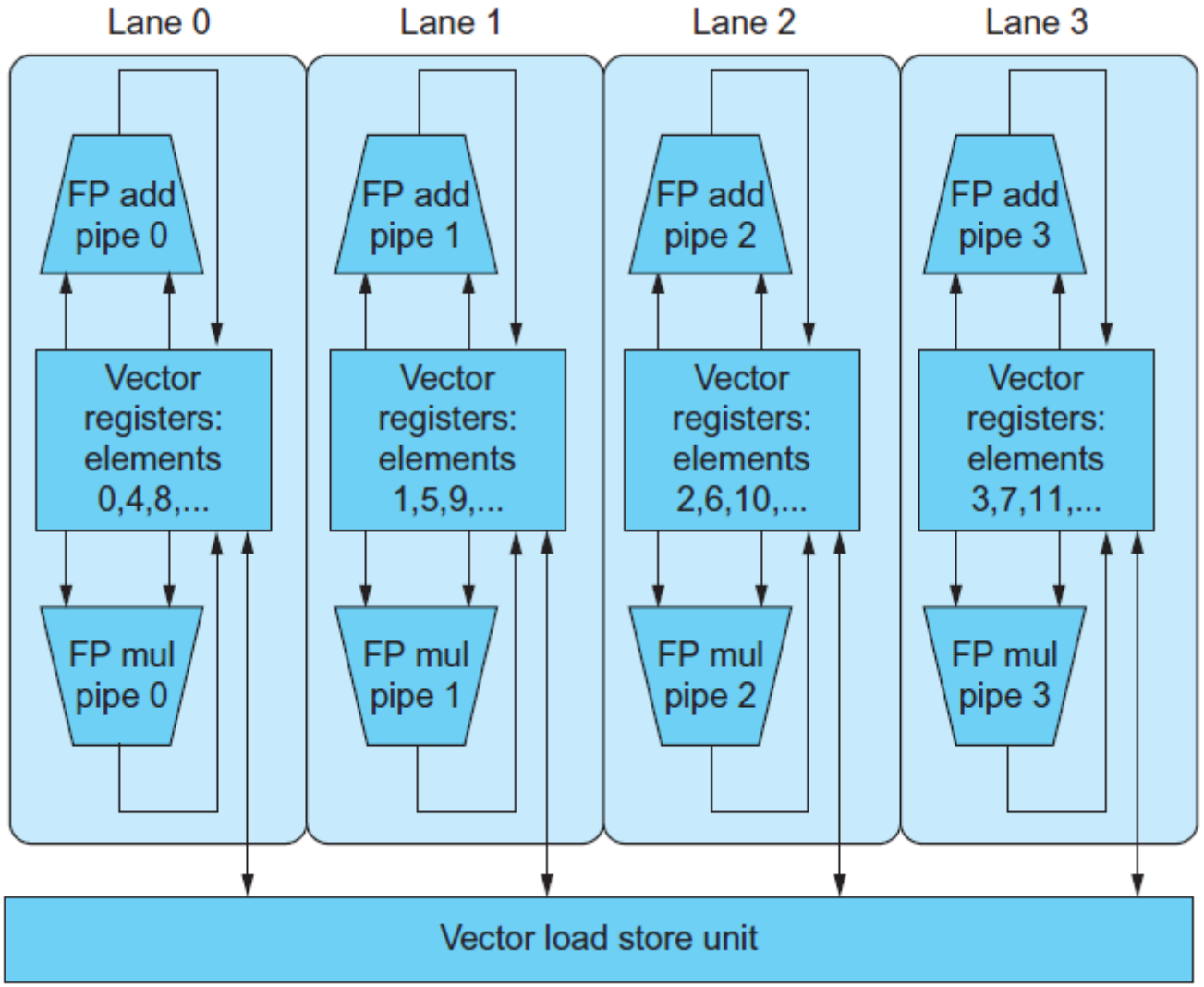


FIGURE **Using multiple functional units to improve the performance**

Vector Processors



Vector vs. Scalar

- Scalar Code

Example DXPY ax+y

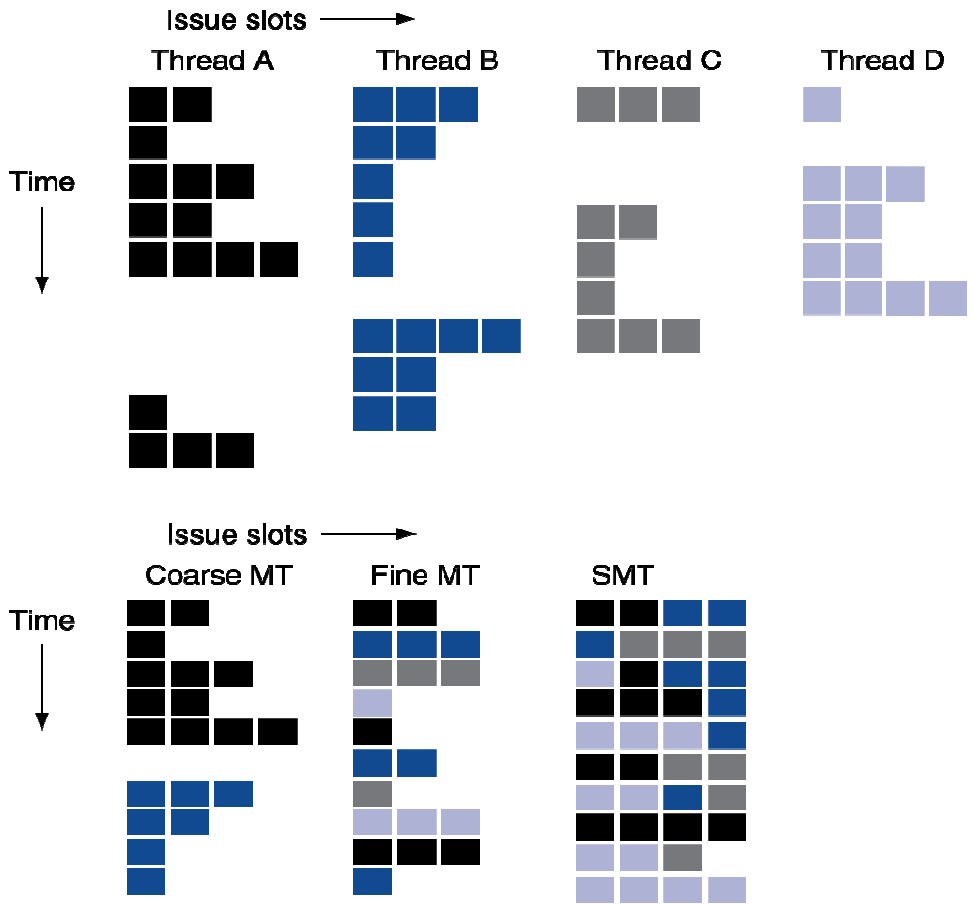
```
      l.d     $f0,a($sp)      :load scalar a
      addiu   $t0,$s0,#512    :upper bound of what to load
loop: l.d     $f2,0($s0)      :load x(i)
      mul.d   $f2,$f2,$f0     :a x x(i)
      l.d     $f4,0($s1)      :load y(i)
      add.d   $f4,$f4,$f2     :a x x(i) + y(i)
      s.d     $f4,0($s1)      :store into y(i)
      addiu   $s0,$s0,#8      :increment index to x
      addiu   $s1,$s1,#8      :increment index to y
      subu    $t1,$t0,$s0     :compute bound
      bne     $t1,$zero,loop  :check if done
```

- Vector Code

```
l.d     $f0,a($sp)      :load scalar a
lv      $v1,0($s0)      :load vector x
mulvs.d $v2,$v1,$f0     :vector-scalar multiply
lv      $v3,0($s1)      :load vector y
addv.d  $v4,$v2,$v3     :add y to product
sv      $v4,0($s1)      :store the result
```

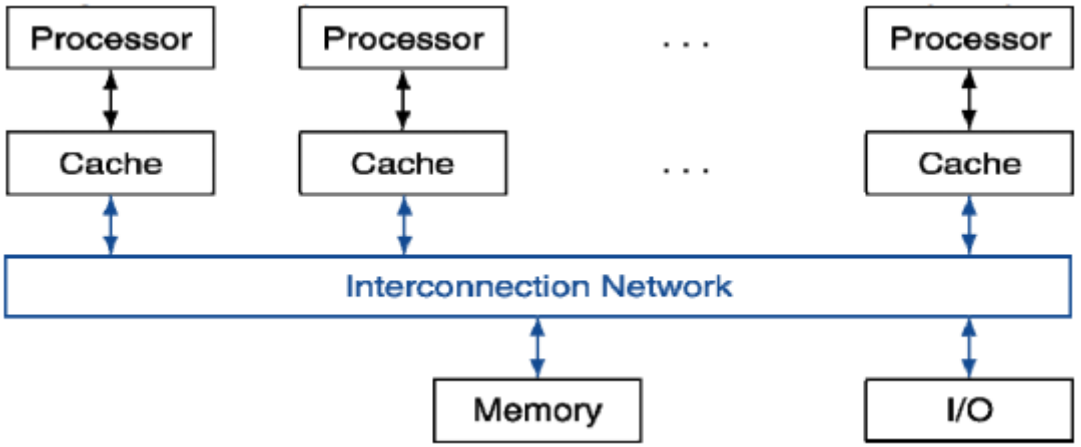
Hardware Multithreading

- Performing multiple threads of execution in parallel
  - Replicate registers, PC, etc.
  - Fast switching between threads



SMP: shared memory multiprocessor

•**SMP: shared memory multiprocessor**
• Small number of cores - multicore chips – shared address multiprocessor
•Single Physical address space
•Implementing a large memory in a single module would create a **bottleneck** when many processors make requests to access the memory simultaneously
•This problem is alleviated by distributing the **memory across multiple modules** so that simultaneous requests from different processors are more likely to access different memory modules, depending on the addresses of those requests.
• **Lock -** Only one processor at a time can acquire the lock, and other processors interested in shared data must wait until the original processor unlocks the variable.



SMP: shared memory multiprocessor

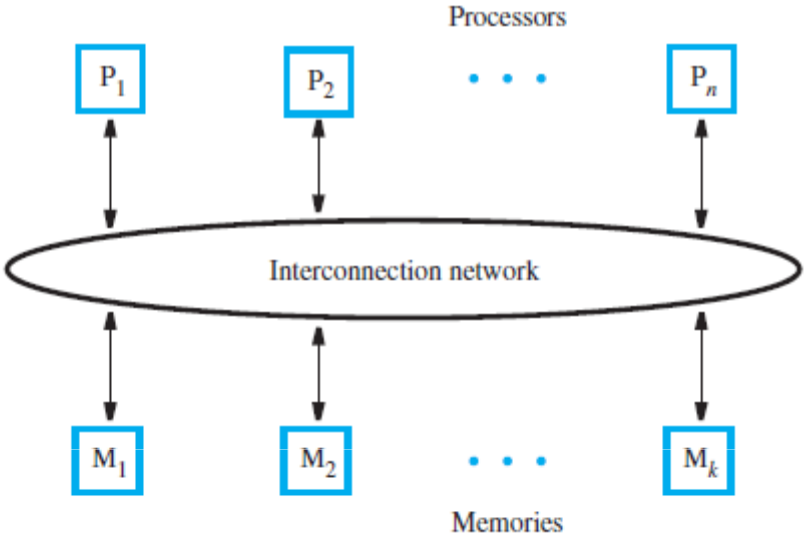


Figure . A UMA multiprocessor.

latency to a word in memory does not depend on which processor asks for it.
Such machines are called **uniform memory access (UMA)**

main memory is divided and attached to different microprocessors or to different memory controllers on the same chip: **nonuniform memory access (NUMA)**

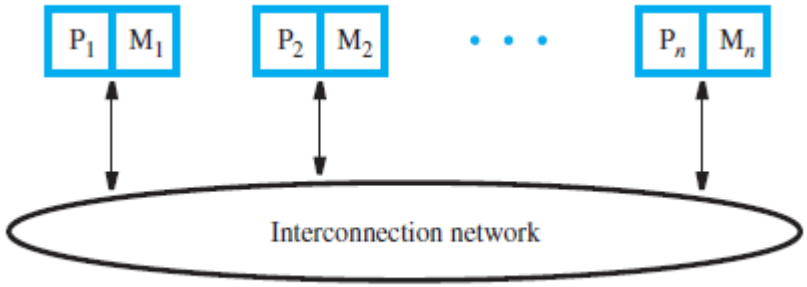


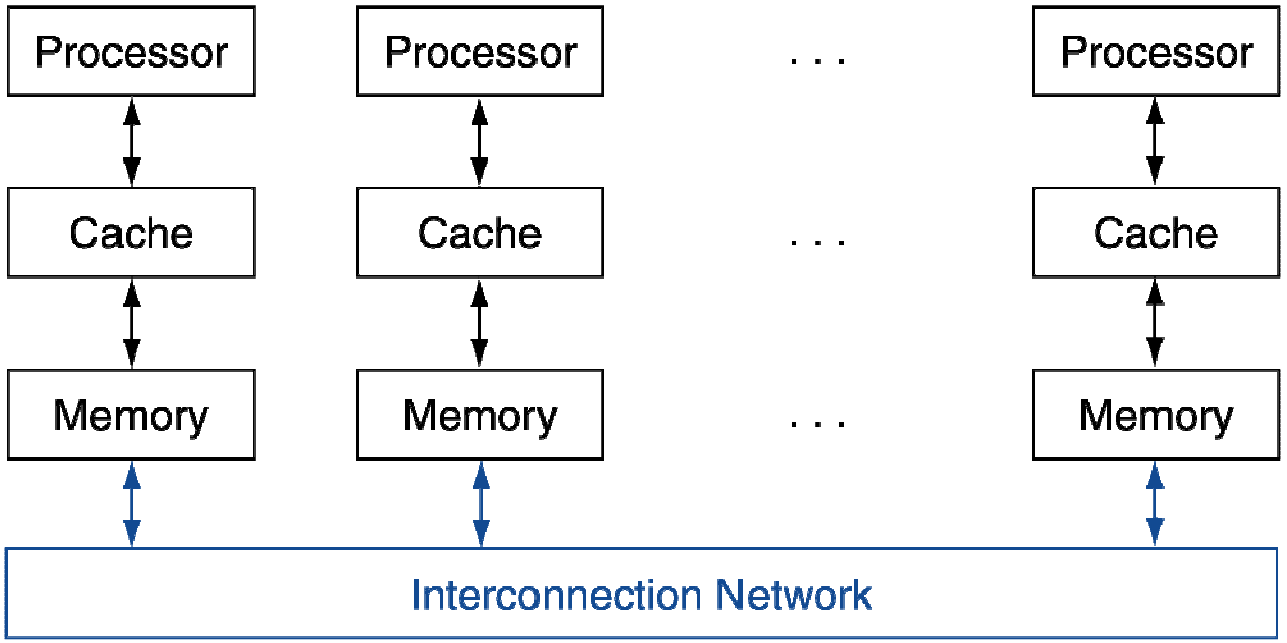
Figure A NUMA multiprocessor.

Parallel Programming

- **Message-Passing Multicomputers**
  - Distributed shared memory (DSM)
  - Clusters
  - WSC
  - Cloud

Parallel Programming

- Distributed shared memory (DSM) – Memory distributed among processors
- Parallel Programming
- Each processor has private physical address space
- Hardware sends/receives messages between processors (Message Passing)



cluster

- A cluster is a type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers cooperatively working together as a single, integrated computing resource.

- Network of independent computers – Nodes
  - separate memories
  - distinct copy of the operating system
  - Connected using I/O system
    - E.g., Ethernet/switch, Internet
- Suitable for applications with independent tasks
    - Web servers, databases, simulations
- Much easier to disconnect and replace a broken computer

Warehouse-Scale Computers

- *Warehouse-Scale Computers (WSC)*
- connects and houses 50,000 to 100,000 servers
- **Software as a Service (SaaS) -** interactive Internet services
- Scaleled to *cloud computing*

Graphics Processing Units

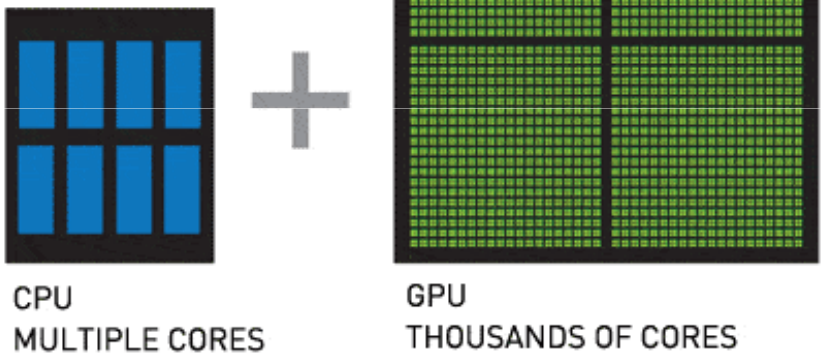
- A graphics processing unit (GPU) is a computer chip that performs rapid mathematical calculations
- occasionally called **visual processing unit** (**VPU**)
- GPU is able to render images more quickly than a CPU because of its parallel processing architecture
- SIMD
- MTSIMD processors
- **Moore's Law** made sense to improve graphics processing

Graphics Processing Units

- Appplication
  - Rapidly growing game market encouraged- developing faster graphics hardware
  - Scientific and Multimedia applications
  - High Performance Computing (HPC), data center, and machine learning applications
- GPUs are used in
  - Embedded Systems
  - Mobile phones
  - Personal computers
  - Workstations
  - Game consoles

- Processing is highly data-parallel
  - GPUs are highly multithreaded
  - Use thread switching to hide memory latency
  - Graphics memory is wide and high-bandwidth
- Programming languages/APIs
  - DirectX, OpenGL
  - C for Graphics (Cg), High Level Shader Language (HLSL)
  - Compute Unified Device Architecture (CUDA)

CPU & GPU



- CPU is the *host*, GPU is the *device*
- GPU used for repetitive and highly-parallel computing task
- Programming model is “Single Instruction Multiple Thread” (SIMT)
- GPU have thousands of core
- Serial portion of code run in CPU and parallel portion in GPU
- CPU perform complex operations
- GPUs rely on hardware multithreading to hide the latency to memory.
- Special graphics DRAM chips available for GPU
- GPU - incredible speed time-sensitive calculations

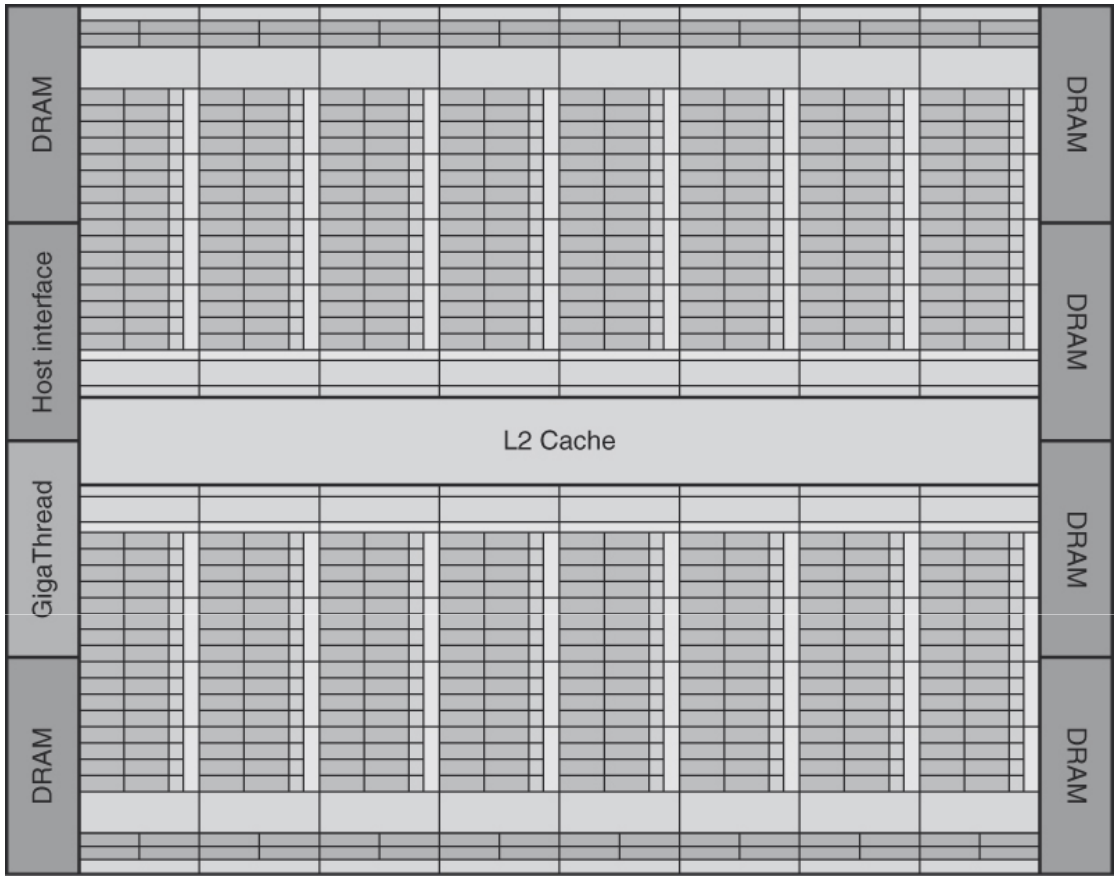
Figure 1. NVIDIA Tesla V100 SXM2 Module with Volta GV100 GPU

NVIDIA GPU Architecture

- GPUs work well only with data-level parallel problems
- CUDA parallel programming language
- Parallelism - *CUDA Thread*
- Threads are blocked together and executed in groups of 32 at a time
- Nvidia introduced the first GPU, the GeForce 256, in 1999
- Others include AMD, Intel and ARM.
- In 2012, Nvidia released a virtualized GPU, which offloads graphics processing from the server CPU in a virtual desktop infrastructure.

Graphics Processing Units

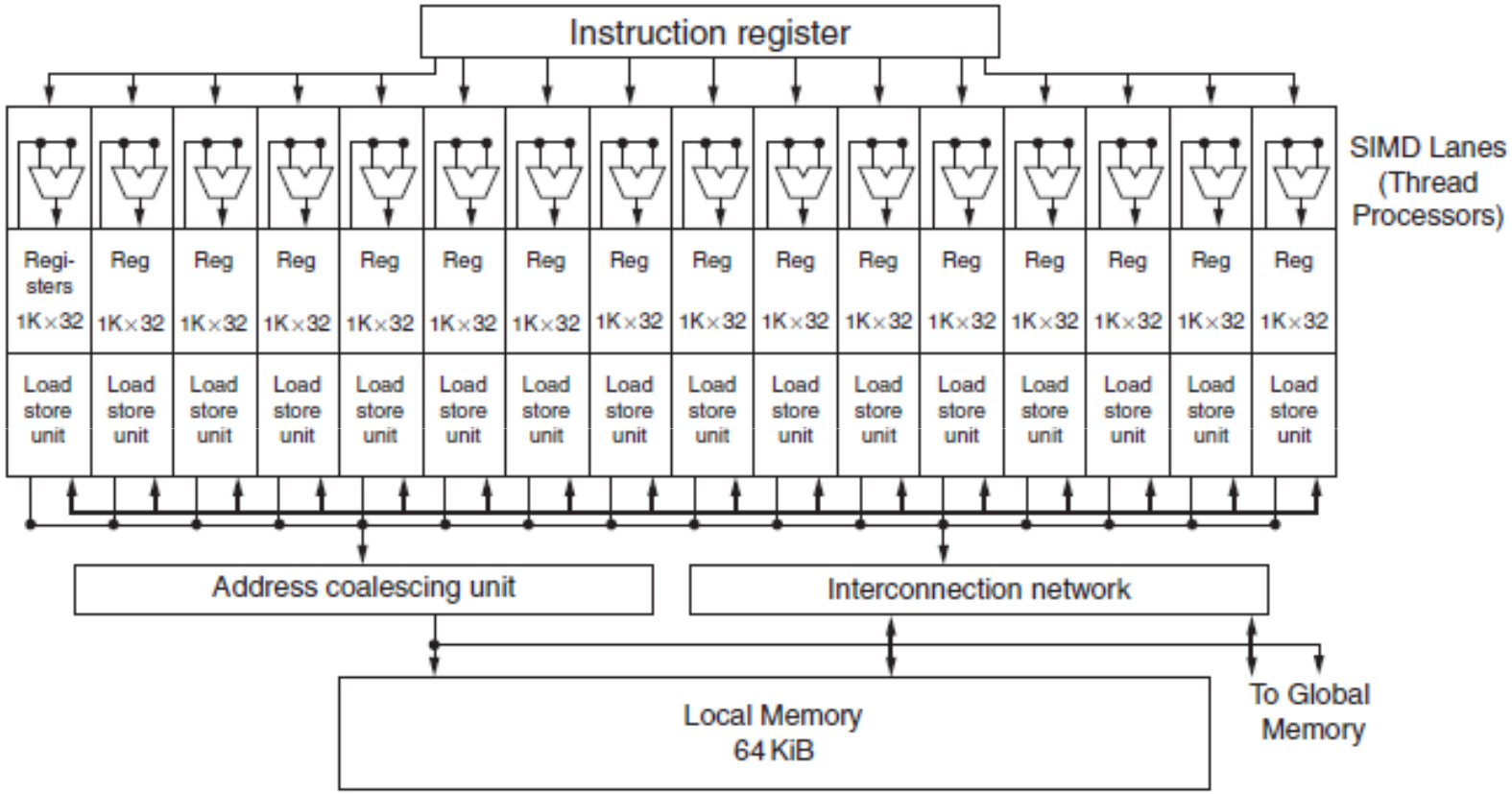
- Two levels of hardware schedulers:
  1. The ***Thread Block Scheduler*** *that assigns blocks of threads to multithreaded* SIMD processors
  2. The **SIMD Thread Scheduler** *within a SIMD processor, which schedules* when SIMD threads should run.



Floor plan of the Fermi GTX 480 GPU.

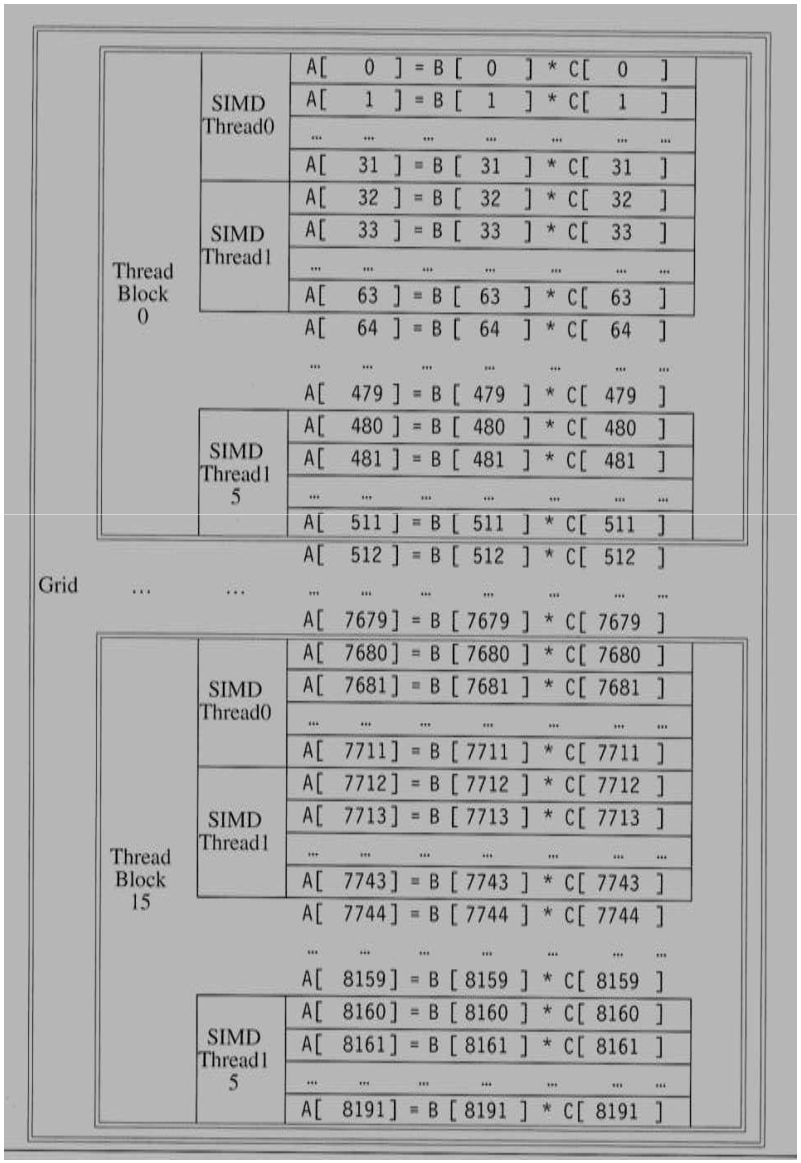
- 16 multithreaded SIMD Processors
- The Thread Block Scheduler is highlighted on the left
- **GDDR5 - graphics double data rate type five synchronous random-access memory**,
- 6 GDDR5 ports, each 64 bits wide, supporting up to 6 GB of capacity
- The Host Interface –PCI
- .

NVIDIA GPU Architecture



Simplified block diagram of the datapath of a multithreaded SIMD Processor.

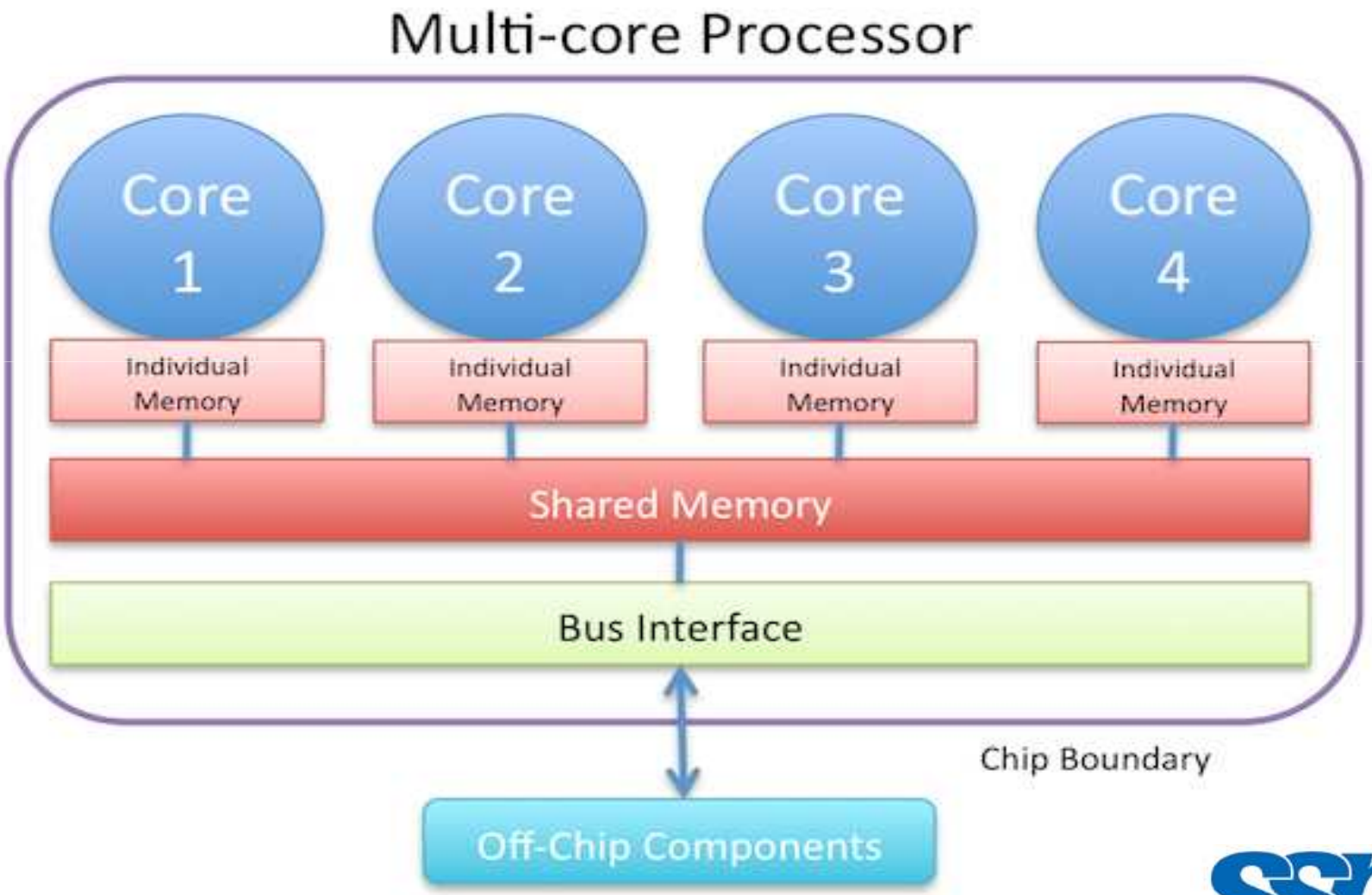
Grid, Threads, and Blocks



Why Multicores ?

- Difficult to make single core clock frequency higher
- Many new applications are multithreaded
- General trend in Computer Architecture is shift toward more parallelism

Multicore Architectures…



Multicore Processors

- Instead of designing and building faster microprocessors, put multiple processors on a single integrated circuit
- It’s a special kind of Multiprocessor
- Introducing parallelism!!!
- Single physical processor contains more than one processor (Core)
  - core” = central processing unit (CPU)
  - each core has its own execution pipeline
  - each core has the resources required to run without blocking resources needed by the other

Multicore Processors

- MIMD
  - Different cores executes different threads (Multiple Instructions),operates on different parts of memory(Multiple Data)
- Shared Memory Multiprocessors.
  - All cores share the same memory.
- Core design enables
  - two or more cores to run
    - at somewhat slower speeds
    - at much lower temperatures
    - and at a much lower level of power consumption
  - combined throughput of these cores delivers processing power greater than the maximum available today on single-core processors
  - Ex: 16 core MIT RAW processor operates at 425 MHz can perform 100 time the number of operations per second than Intel Pentium-3 with 600MHz.

•

Multicore Processors

- Advantages
  - Occupies less space on PCB
  - Higher throughput
  - Consume less power
  - Performs more operations/sec with less frequency
- Disadvantages
  - Requires changes in operating system and existing application software
  - It becomes hot
  - Resource Sharing

Multicore applications

- Data base servers
- Web servers
- Compilers
- Multimedia Applications
- Scientific Applications
- General applications with TLP as opposed to ILP
- Downloading s/w while running Anti virus s/w
- Editing photo while recording TV show.