Chapter 17 - Parallel Processing

Luis Tarrataca
luis.tarrataca@gmail.com

CEFET-RJ

Table of Contents I

Table of Contents II

Table of Contents I

Motivation

Today we will look at ways to increase processor performance:

- Always a fun topic =)

What are some of the techniques you know of to increase processor performance?

- **Frequency**
  - ``Easy way'' out =)
  - Inherent vantages and disadvantages. Can you name them?
- **Cache**
  - High speed memory;
  - Idea: diminish the number of reads from low latency memory (RAM, HD, etc);
  - Very expensive!
- **Pipeline**;
  - Idea: assembly line for instructions;
  - Leads to performance increases;
  - But also introduces a host of new considerations (hazards);

Today we will discuss another technique: parallelization

- Original μP design: sequential processing of instructions.
- Parallelism: Execute multiple instructions at the same time. How?
  - As always in engineering there are multiple strategies.
  - Choice made based on computational requirements.

Parallelism choice is made on computational requirements:

> Can you think of some of the dimensions that influence parallel computation?

Parallelism choice is made on computational requirements:

> Can you think of some of the dimensions that influence parallel computation?

What about these?

- Are we processing a single source of data? Or multiple?
- Do we need to apply the same instruction to the data? Or different ones?

Categories of Computer Systems (1/4)

- **Single instruction, single data (SISD):**
  - Single processor executes a single instruction stream to operate on data stored in a single memory.

Figure: SISD (Source: (Stallings, 2015))

CU = Control unit	SISD = Single instruction, single data stream
IS = Instruction stream	SIMD = Single instruction, multiple data stream
PU = Processing unit	MIMD = Multiple instruction, multiple data stream
DS = Data stream	
MU = Memory unit	
LM = Local memory	

Figure: Nomenclature (Source: (Stallings, 2015))

Categories of Computer Systems (2/4)

- **Single instruction, multiple data (SIMD):**
  - Single machine instruction controls the simultaneous execution of a number of processing elements on a lockstep basis;
  - Each processing element has an associated data memory:
    - Instructions are executed on $\neq$ data sets by $\neq$ processors.

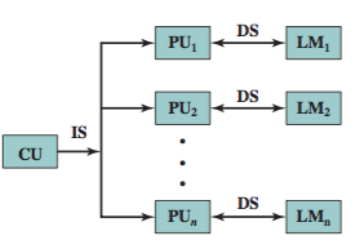


Figure: SIMD (Source: (Stallings, 2015))

CU	= Control unit	SISD	= Single instruction, single data stream
IS	= Instruction stream		
PU	= Processing unit	SIMD	= Single instruction, multiple data stream
DS	= Data stream		
MU	= Memory unit	MIMD	= Multiple instruction, multiple data stream
LM	= Local memory		

Figure: Nomenclature (Source: (Stallings, 2015))

Categories of Computer Systems (3/4)

- **Multiple instruction, single data (MISD):**
  - Sequence of data is transmitted to a set of processors, each of which executes a different instruction sequence.

Categories of Computer Systems (4/4)

- **Multiple instruction, multiple data (MIMD):**
  - Set of processors simultaneously execute $\neq$ instruction sequences on $\neq$ data sets.

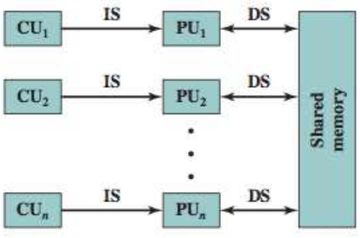


Figure: MIMD (Source: (Stallings, 2015))

CU	= Control unit	SISD	= Single instruction, single data stream
IS	= Instruction stream	SIMD	= Single instruction, multiple data stream
PU	= Processing unit	MIMD	= Multiple instruction, multiple data stream
DS	= Data stream		
MU	= Memory unit		
LM	= Local memory		

Figure: Nomenclature (Source: (Stallings, 2015))

Taxonomy of parallel computing systems

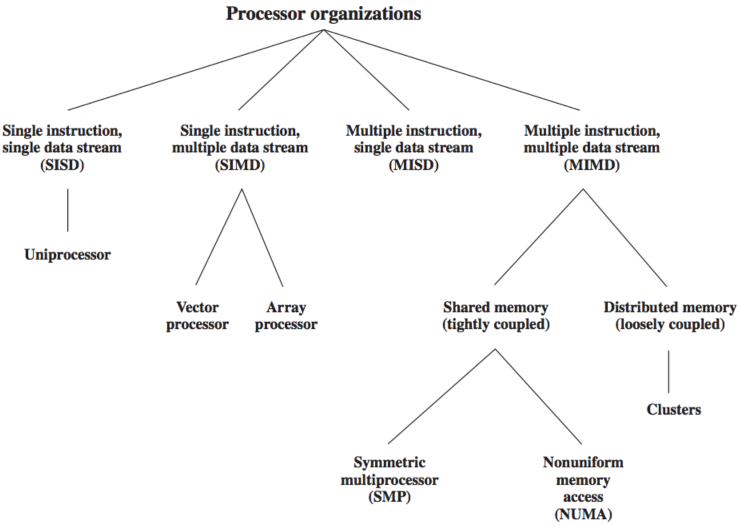


Figure: Taxonomy of Parallel Processor Architectures (Source: (Stallings, 2015))

Today's class will focus on:

- MIMD parallel processing:
  - SMP (building block);
  - Clusters (arrangement of building blocks).
    - Data centers;
    - Supercomputers.
- SIMD parallel processing:
  - General-purpose computing on graphics processing units (GPGPU)
  - Intel XEON Phi.

Symmetric Multiprocessor Parallelism

SMP has the following characteristics (1/2):

- There are two or more similar processors of comparable capability.
- Processors share
  - Main memory;
  - I/O facilities and devices;
- Processors are interconnected by:
  - Bus or other connection scheme;
  - Memory access time is approximately the same for each processor.

Symmetric Multiprocessor Parallelism

SMP has the following characteristics (2/2):

- Processors perform the same functions:
  - Hence the term symmetric
- System is controlled by an operating system managing:
  - Processors and programs;
  - How processes / threads are to be executed in the processors;
  - User does not need to worry about anything =)

Processor Organization

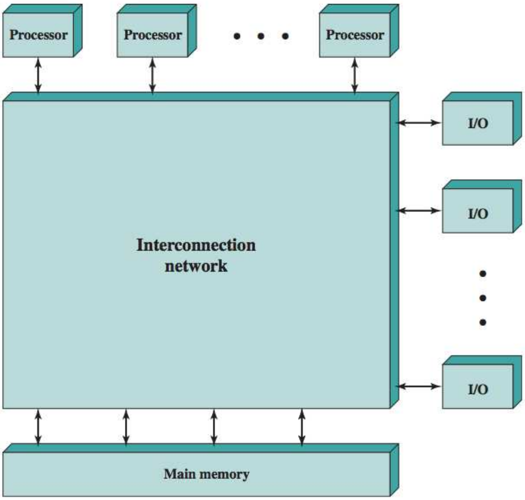


Figure: Generic Block Diagram of a Tightly Coupled Multiprocessor (Source: (Stallings, 2015))

- Two or more processors:
  - Each processor includes: a control unit, ALU, registers, cache;
- Each processor has access to:
  - Shared main memory;
  - I/O devices.
- Processors communicate with each other through:
  - Memory: messages and status information.
  - Exchanging signals directly.

With all the components involved:

Do you think it would be easy for a programmer to control everything at assembly level? Any ideas?

With all the components involved:

Do you think it would be easy for a programmer to control everything at assembly level? Any ideas?

- It would be a nightmare...

How are then the different computational resources best used? Any ideas?

How are then the different computational resources best used? Any ideas?

- Combination of operating system support tools:
  - Processes;
  - Threads.
- Alongside application programming interfaces;
- Integral part of an Operating System course.

Most common organization:

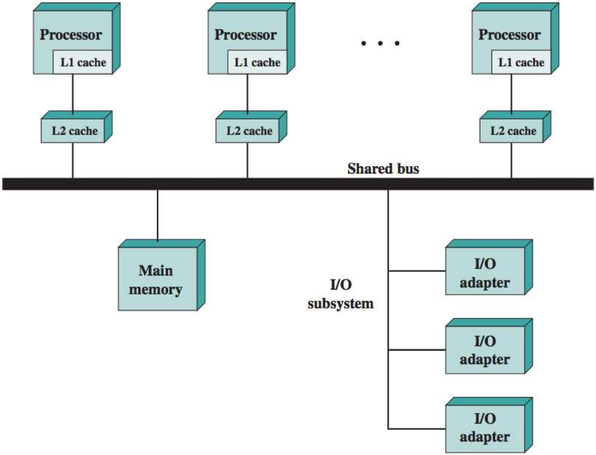


Figure: Symmetric Multiprocessor Organization (Source: (Stallings, 2015))

But how is the bus managed with multiple processors? Any ideas?

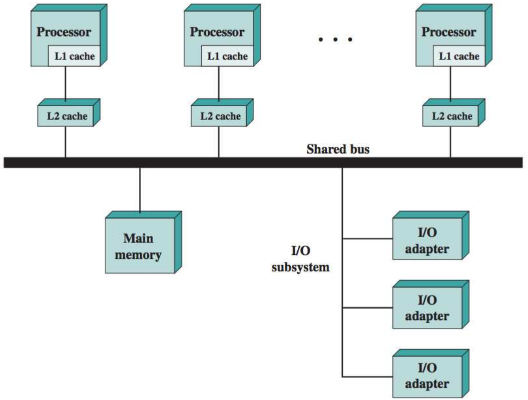


Figure: Symmetric Multiprocessor Organization (Source: (Stallings, 2015))

But how is the bus managed with multiple processors? Any ideas?

- **Time-sharing:**
  - When one module is controlling the bus:
    - Other modules are locked out;
    - If necessary: modules suspend operation until bus access is achieved.

- Bus has the same structure for a single-processor:
  - Control, address, and data lines.
- To facilitate DMA transfers the following features are provided:
  - Addressing: to distinguish modules on the bus;
  - Arbitration: Any I/O module can temporarily function as ``master''.
    - Mechanism arbitrates competing requests for bus control:
    - Using some sort of priority scheme.

Now we have a single bus being used by multiple processors. Can you see any problems with this?

Now we have a single bus being used by multiple processors. Can you see any problems with this?

- All memory accesses pass through the common bus;
- Bus cycle time limits the speed of the system;
- This results in a performance bottleneck;

But this leads to the following question:

> What is the most common way for reducing the number of memory accesses?

But this leads to the following question:

What is the most common way for reducing the number of memory accesses?

- Cache ;)

In a multiprocessor architecture:

Can you see any problems with using multiple caches? Any ideas?

Cache coherence problem

Each local cache contains an image of a portion of memory:

- if a word is altered in one cache:
  - Could conceivably invalidate a word in another cache;
- To prevent this:
  - Other processors must be alerted that an update has taken place
- Commonly addressed in hardware rather than by software.

Cache Coherence Protocols

Dynamic recognition at run time of potential inconsistency conditions:

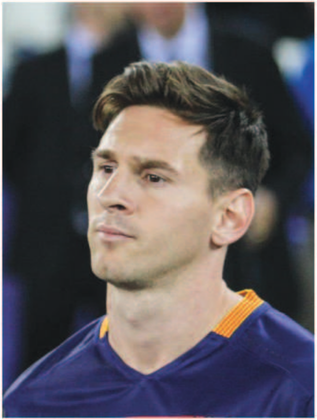
- Always cache memory;
- Problem is only dealt with when it actually arises;
- Better performance over a software approach,

Transparent to the programmer and the compiler:

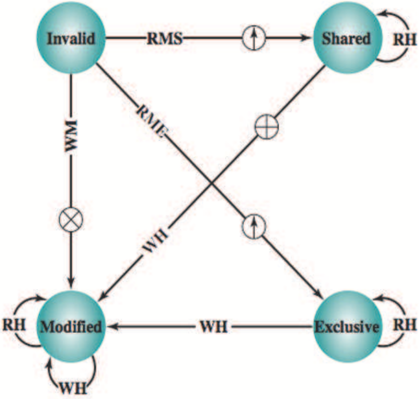
- Reducing the software development burden.

We will look at one such protocol: **MESI**

Not this Mesi though...



Vs.



MESI protocol

Each **line** of the cache can be in one of four states (1/2):

- **Modified:**
  - Line in the cache has been modified ($\neq$ from main memory)
  - Line is not present in any other cache.
- **Exclusive:**
  - Line in the cache is the same as that in main memory;
  - Line is not present in any other cache.

MESI protocol

Each **line** of the cache can be in one of four states (2/2):

- **Shared:**
  - Line in the cache is the same as that in main memory;
  - Line may be present in another cache.
- **Invalid:**
  - Line in the cache does not contain valid data.

This time in table form:

	M Modified	E Exclusive	S Shared	I Invalid
This cache line valid?	Yes	Yes	Yes	No
The memory copy is …	out of date	valid	valid	—
Copies exist in other caches?	No	No	Maybe	Maybe
A write to this line …	does not go to bus	does not go to bus	goes to bus and updates cache	goes directly to bus

Figure: MESI cache line states (Source: (Stallings, 2015))

At any time a cache line is in a single state:

- If next event is from the attached processor:
  - Transition is dictated by figure below on the left;
- If next event is from the bus:
  - Transition is dictated by figure below on the right.

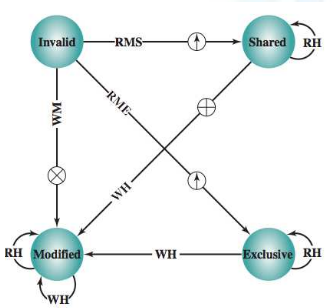


Figure: Line in cache at **initiating** processor (Source: (Stallings, 2015))

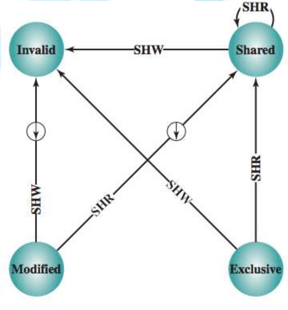


Figure: Line in **snooping** cache (Source: (Stallings, 2015))

Caption for the previous figure:

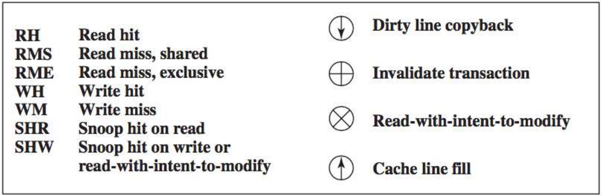


Figure: (Source: (Stallings, 2015))

Lets see what all of this means ;)

What happens when a read miss occurs? Any ideas?

What happens when a read miss occurs? Any ideas?

Well it depends...:

- Has any cache a copy of the line in the **exclusive** state?
- Do any caches have a copy of the line in the **shared** state?
- Has any cache a copy of the line in the **modified** state?
- What if no other copies exist?

Read miss: occurs in the local cache, the processor:

1. initiates a main memory read of the missing address;
2. inserts a signal on the bus that:
    - Alerts all other processor/cache units to snoop the transaction;
3. A number of possible outcomes may occur (1/4):
    - **Possibility 1:** One cache has a clean copy of the line in the **exclusive** state:

Read miss: occurs in the local cache, the processor:

1. initiates a main memory read of the missing address;
2. inserts a signal on the bus that:
   - Alerts all other processor/cache units to snoop the transaction;
3. A number of possible outcomes may occur (1/4):
   - **Possibility 1:** One cache has a clean copy of the line in the **exclusive** state:
     1. Cache returns a signal indicating that it shares this line;

Read miss: occurs in the local cache, the processor:

1. initiates a main memory read of the missing address;
2. inserts a signal on the bus that:
   - Alerts all other processor/cache units to snoop the transaction;
3. A number of possible outcomes may occur (1/4):
   - **Possibility 1:** One cache has a clean copy of the line in the **exclusive** state:
     1. Cache returns a signal indicating that it shares this line;
     2. Responding processor transitions from **exclusive** state to **shared**;

Read miss: occurs in the local cache, the processor:

1. initiates a main memory read of the missing address;
2. inserts a signal on the bus that:
    - Alerts all other processor/cache units to snoop the transaction;
3. A number of possible outcomes may occur (1/4):
    - **Possibility 1:** One cache has a clean copy of the line in the **exclusive** state:
        1. Cache returns a signal indicating that it shares this line;
        2. Responding processor transitions from **exclusive** state to **shared**;
        3. Initiating processor reads line from memory: goes from **invalid** to **shared**.

Read miss occurs in the local cache, the processor:

1. Initiates a main memory read of the missing address;
2. Inserts a signal on the bus that:
    - Alerts all other processor/cache units to snoop the transaction;
3. A number of possible outcomes may occur (2/4):
    - **Possibility 2:** ≥ 1 Caches have a clean copy of the line in the **shared** state:

Read miss occurs in the local cache, the processor:

1. Initiates a main memory read of the missing address;
2. Inserts a signal on the bus that:
    - Alerts all other processor/cache units to snoop the transaction;
3. A number of possible outcomes may occur (2/4):
    - **Possibility 2:** ≥ 1 Caches have a clean copy of the line in the **shared** state:
        1. Each cache signals that it shares the line;

Read miss occurs in the local cache, the processor:

1. Initiates a main memory read of the missing address;
2. Inserts a signal on the bus that:
   - Alerts all other processor/cache units to snoop the transaction;
3. A number of possible outcomes may occur (2/4):
   - **Possibility 2:** $\geq$ 1 Caches have a clean copy of the line in the **shared** state:
     1. Each cache signals that it shares the line;
     2. Initiating processor reads from memory and line goes from **invalid** to **shared**.

Read miss occurs in the local cache, the processor:

1. Initiates a main memory read of the missing address;
2. Inserts a signal on the bus that:
   - Alerts all other processor/cache units to snoop the transaction;
3. A number of possible outcomes may occur (3/4):
   - **Possibility 3:** If one other cache has a modified copy of the line:

Read miss occurs in the local cache, the processor:

1. Initiates a main memory read of the missing address;
2. Inserts a signal on the bus that:
    - Alerts all other processor/cache units to snoop the transaction;
3. A number of possible outcomes may occur (3/4):
    - **Possibility 3:** If one other cache has a modified copy of the line:
        1. Cache containing modified line blocks memory read from initiating processor;

Read miss occurs in the local cache, the processor:

1. Initiates a main memory read of the missing address;
2. Inserts a signal on the bus that:
   - Alerts all other processor/cache units to snoop the transaction;
3. A number of possible outcomes may occur (3/4):
   - **Possibility 3:** If one other cache has a modified copy of the line:
     1. Cache containing modified line blocks memory read from initiating processor;
     2. Cache containing modified line provides the line to the requesting cache;

Read miss occurs in the local cache, the processor:

1. Initiates a main memory read of the missing address;
2. Inserts a signal on the bus that:
   - Alerts all other processor/cache units to snoop the transaction;
3. A number of possible outcomes may occur (3/4):
   - **Possibility 3:** If one other cache has a modified copy of the line:
     1. Cache containing modified line blocks memory read from initiating processor;
     2. Cache containing modified line provides the line to the requesting cache;
     3. Responding cache then changes its line from **modified** to **shared**;

Read miss occurs in the local cache, the processor:

1. Initiates a main memory read of the missing address;
2. Inserts a signal on the bus that:
    - Alerts all other processor/cache units to snoop the transaction;
3. A number of possible outcomes may occur (3/4):
    - **Possibility 3:** If one other cache has a modified copy of the line:
        1. Cache containing modified line blocks memory read from initiating processor;
        2. Cache containing modified line provides the line to the requesting cache;
        3. Responding cache then changes its line from **modified** to **shared**;
        4. Memory controller updates the contents of the line in memory;

Read miss occurs in the local cache, processor:

1. Initiates a main memory read of the missing address;
2. Inserts a signal on the bus that:
    - Alerts all other processor/cache units to snoop the transaction;
3. A number of possible outcomes may occur (4/4):
    - **Possibility 4:** If no other cache has a copy of the line, no signals are returned:
        1. Initiating processor reads the line from memory;

Read miss occurs in the local cache, processor:

1. Initiates a main memory read of the missing address;
2. Inserts a signal on the bus that:
   - Alerts all other processor/cache units to snoop the transaction;
3. A number of possible outcomes may occur (4/4):
   - **Possibility 4:** If no other cache has a copy of the line, no signals are returned:
     1. Initiating processor reads the line from memory;
     2. Initiating processor transitions the line in its cache from **invalid** to **exclusive**.

Read hit occurs on a line currently in the local cache:

1. Processor reads the required item;
2. No state change: state remains {modified, shared, or exclusive}.

Write miss occurs in the local cache:

1. Initiating processor attempts to read the line of main memory;
2. Processor signals read-with-intent-to-modify (**RWITM**) on bus.
3. **Possibility 1:** some other cache may have a **modified** copy of this line:
    1. Alerted processor signals that it has a modified copy of the line;
4. Processor signals read-with-intent-to-modify (**RWITM**) on bus:
    - Reads the line from main memory;
    - Modifies the line in the cache;
    - Marks the line in the modified state.

Write miss occurs in the local cache:

1. Initiating processor attempts to read the line of main memory;
2. Processor signals read-with-intent-to-modify (**RWITM**) on bus.
3. **Possibility 1:** some other cache may have a **modified** copy of this line:
    1. Alerted processor signals that it has a modified copy of the line;
    2. Initiating processor surrenders the bus and waits;
4. Processor signals read-with-intent-to-modify (**RWITM**) on bus:
    - Reads the line from main memory;
    - Modifies the line in the cache;
    - Marks the line in the modified state.

Write miss occurs in the local cache:

1. Initiating processor attempts to read the line of main memory;
2. Processor signals read-with-intent-to-modify (**RWITM**) on bus.
3. **Possibility 1:** some other cache may have a **modified** copy of this line:
    1. Alerted processor signals that it has a modified copy of the line;
    2. Initiating processor surrenders the bus and waits;
    3. Other processor writes the modified line to main memory;
4. Processor signals read-with-intent-to-modify (**RWITM**) on bus:
    - Reads the line from main memory;
    - Modifies the line in the cache;
    - Marks the line in the modified state.

Write miss occurs in the local cache:

1. Initiating processor attempts to read the line of main memory;
2. Processor signals read-with-intent-to-modify (**RWITM**) on bus.
3. **Possibility 1:** some other cache may have a **modified** copy of this line:
    1. Alerted processor signals that it has a modified copy of the line;
    2. Initiating processor surrenders the bus and waits;
    3. Other processor writes the modified line to main memory;
    4. Other processor transitions the state of the cache line to **invalid**.
4. Processor signals read-with-intent-to-modify (**RWITM**) on bus:
    - Reads the line from main memory;
    - Modifies the line in the cache;
    - Marks the line in the modified state.

Write miss occurs in the local cache:

1. Initiating processor attempts to read the line of main memory;
2. Processor signals read-with-intent-to-modify (**RWITM**) on bus.
3. **Possibility 2:** No other cache has a **modified** copy of the requested line:
    1. No signal is returned by the other processors;

Write miss occurs in the local cache:

1. Initiating processor attempts to read the line of main memory;
2. Processor signals read-with-intent-to-modify (**RWITM**) on bus.
3. **Possibility 2:** No other cache has a **modified** copy of the requested line:
    1. No signal is returned by the other processors;
    2. Initiating processor proceeds to read in the line and modify it;

Write miss occurs in the local cache:

1. Initiating processor attempts to read the line of main memory;
2. Processor signals read-with-intent-to-modify (**RWITM**) on bus.
3. **Possibility 2:** No other cache has a **modified** copy of the requested line:
    1. No signal is returned by the other processors;
    2. Initiating processor proceeds to read in the line and modify it;
    3. If one or more caches have a clean copy of the line in the **shared** state:

Write miss occurs in the local cache:

1. Initiating processor attempts to read the line of main memory;
2. Processor signals read-with-intent-to-modify (**RWITM**) on bus.
3. **Possibility 2:** No other cache has a **modified** copy of the requested line:
    1. No signal is returned by the other processors;
    2. Initiating processor proceeds to read in the line and modify it;
    3. If one or more caches have a clean copy of the line in the **shared** state:
        - Each cache **invalidates** its copy of the line.

Write miss occurs in the local cache:

1. Initiating processor attempts to read the line of main memory;
2. Processor signals read-with-intent-to-modify (**RWITM**) on bus.
3. **Possibility 2:** No other cache has a **modified** copy of the requested line:
    1. No signal is returned by the other processors;
    2. Initiating processor proceeds to read in the line and modify it;
    3. If one or more caches have a clean copy of the line in the **shared** state:
        - Each cache **invalidates** its copy of the line.
    4. If one cache has a clean copy of the line in the **exclusive** state

Write miss occurs in the local cache:

1. Initiating processor attempts to read the line of main memory;
2. Processor signals read-with-intent-to-modify (**RWITM**) on bus.
3. **Possibility 2:** No other cache has a **modified** copy of the requested line:
    1. No signal is returned by the other processors;
    2. Initiating processor proceeds to read in the line and modify it;
    3. If one or more caches have a clean copy of the line in the **shared** state:
        - Each cache **invalidates** its copy of the line.
    4. If one cache has a clean copy of the line in the **exclusive** state
        - That cache **invalidates** its copy of the line.

Write hit occurs on a line in the local cache, effect depends on line state:

- **Possibility 1 - Shared state:**
  - Before updating, processor must gain exclusive ownership of the line:

Write hit occurs on a line in the local cache, effect depends on line state:

- **Possibility 1 - Shared state:**
  - Before updating, processor must gain exclusive ownership of the line:
    - Processor signals its intent on bus;

Write hit occurs on a line in the local cache, effect depends on line state:

- **Possibility 1 - Shared state:**
  - Before updating, processor must gain exclusive ownership of the line:
    - Processor signals its intent on bus;
    - Each processor that has a shared copy of the line goes from **shared** to **invalid**.

Write hit occurs on a line in the local cache, effect depends on line state:

- **Possibility 1 - Shared state:**
  - Before updating, processor must gain exclusive ownership of the line:
    - Processor signals its intent on bus;
    - Each processor that has a shared copy of the line goes from **shared** to **invalid**.
    - Initiating processor updates line and then state from **shared** to **modified**.

Write hit occurs on a line in the local cache, effect depends on line state:

- **Possibility 2 - Exclusive state:**
    - Processor already has exclusive control of this line:
        - Simply performs the update;

Write hit occurs on a line in the local cache, effect depends on line state:

- **Possibility 2 - Exclusive state:**
  - Processor already has exclusive control of this line:
    - Simply performs the update;
    - Transitions its copy of the line from **exclusive** to **modified**.

Write hit occurs on a line in the local cache, effect depends on line state:

- **Possibility 3 - Modified state:**
  - Processor has exclusive control of this line and marked as modified:(<+->)
    - Simply performs the update;

Seems complex?

Seems complex?

- Imagine multiple cache levels that can be shared...

Seems complex?

- Imagine multiple cache levels that can be shared...

Clusters

Group of computers working together as a unified computing resource:

- Each computer in a cluster is typically referred to as a node.

Some benefits:

- **Scalability** - Possible to add new nodes to the cluster;
- **Availability** - Failure of one node does not mean loss of service.

How is information stored within the cluster? Any ideas?

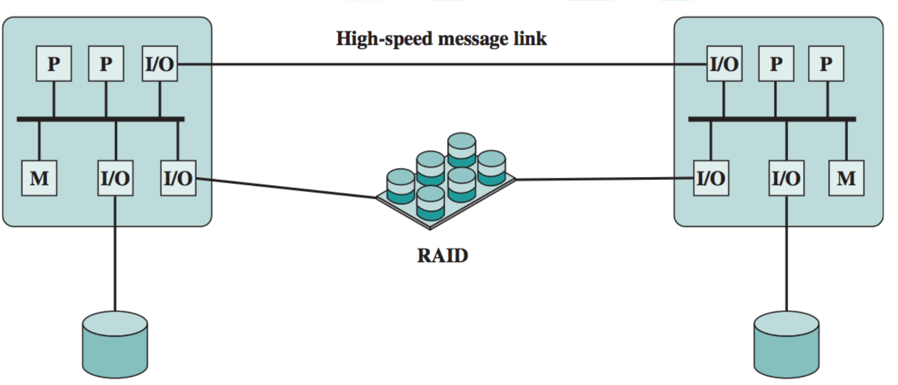
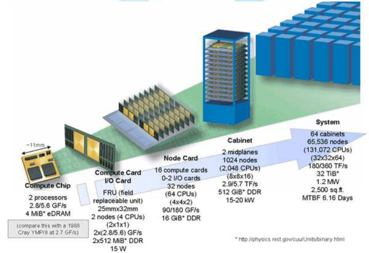


Figure: Shared Disk Cluster Configuration (Source: (Stallings, 2015))

A cluster system raises a series of important design issues:

- **Failure Management:** How are failures managed by a cluster?
  - A computation can be restarted on a different node;
  - Or a computation can be continued on a different node;
- **Load Balancing:** How is the processing load among available computers?
- **Parallelize Computation:** How is the computation parallelized?
  - Compiler? Purposely programmed? Parametric?

Supercomputers are cluster systems:



- High-speed networks and physical proximity allow for better performance.
- Type of system used by the Santos Dumont supercomputer at LNCC.

Vector Computation

Perform arithmetic operations on arrays of floating-point numbers:

- *Example:* system simulation:
  - These can typical be done by a set of equations;
  - Basic Linear Algebra Subprograms (BLAS).

Same operation is performed to different data:

- Thus Single Instruction Multiple Data (SIMD).

Lets look at an example: We want to sum the elements of an array.

$$C_i = A_i + B_i$$

$$\begin{bmatrix} 1.5 \\ 7.1 \\ 6.9 \\ 100.5 \\ 0 \\ 59.7 \end{bmatrix} + \begin{bmatrix} 2.0 \\ 39.7 \\ 1000.003 \\ 11 \\ 21.1 \\ 19.7 \end{bmatrix} = \begin{bmatrix} 3.5 \\ 46.8 \\ 1006.093 \\ 111.5 \\ 21.1 \\ 79.4 \end{bmatrix}$$

$$A \quad + \quad B \quad = \quad C$$

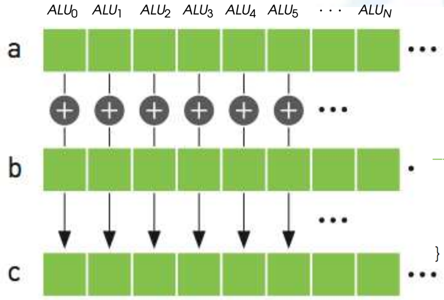
Figure: Example of Vector Addition (Source: (Stallings, 2015))

General-purpose computer

- iterate through each element of the array.
- Overkill and inefficient! Why?
  - CPU can do floating-point operations;
  - But can also do a host of other things:
    - I/O operations;
    - Memory management;
    - Cache management;
- We are mostly interested in arithmetic operations:
  - Arithmetic Logic Unit (ALU)

Idea: General-purpose parallel computation based on ALUs

- Instead of using processors for parallel computation;
- ALUs are also cheap =)
- Careful memory management:
  - Since memory is shared by the different ALUs.




```
__global__ void add( int *a, int *b, int *c ) {
    int tid = blockIdx.x;     // handle the data at thi
    if (tid < N)
        c[tid] = a[tid] + b[tid];
}
```

Figure: Summing two vectors (Source: (Sanders and Kandrot, 2011))

- variable tid is the identification of the ALU;

Two main categories:

- Pipelined ALU;
- Parallel ALUs (GPGPUs):
    - Pipelined or Not-pipelined.

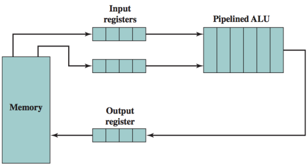


Figure: Pipelined ALU (Source: (Stallings, 2015))

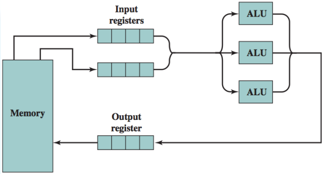


Figure: Parallel ALU (Source: (Stallings, 2015))

Floating-point operations are complex operations:

- Decompose a floating-point operation into stages;
  - Pipeline for floating-point operations =)

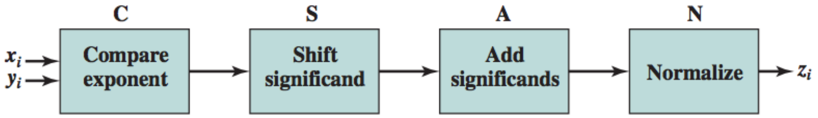


Figure: The different stages for the pipeline for floating-point operations (Source: (Stallings, 2015))

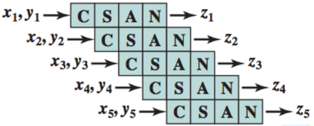


Figure: Pipelined ALU (Source: (Stallings, 2015))

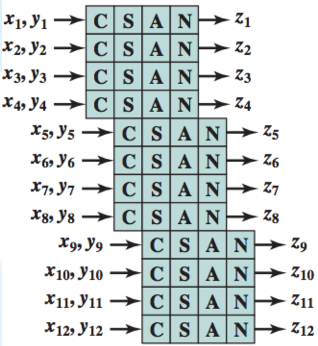


Figure: Parallel ALU (Source: (Stallings, 2015))

- C - Compare exponent, S - shift exponent, A - add significands, N - normalize
- Same problems that occur with processor pipelining also occur here.

As a curiosity lets look at the architecture of a GeForce8800:

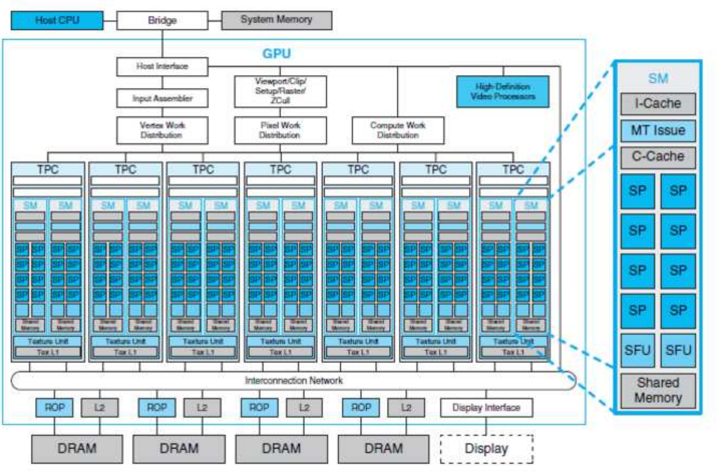


Figure: GeForce 8800 architecture (Source: NVIDIA)

- 14 Streaming Multiple Processor (SM) each containing:
  - Each SM contains 8 Streaming Processor (SP):
  - Each SP processor has a fully pipelined
    - integer arithmetic logic unit (ALU);
    - floating point unit (FPU).
  - Total: 224 ALUs.
- This an old card from 2006, recent models contain thousands of ALUs!
- Supercomputers are a combination of processors and GPGPU nodes.

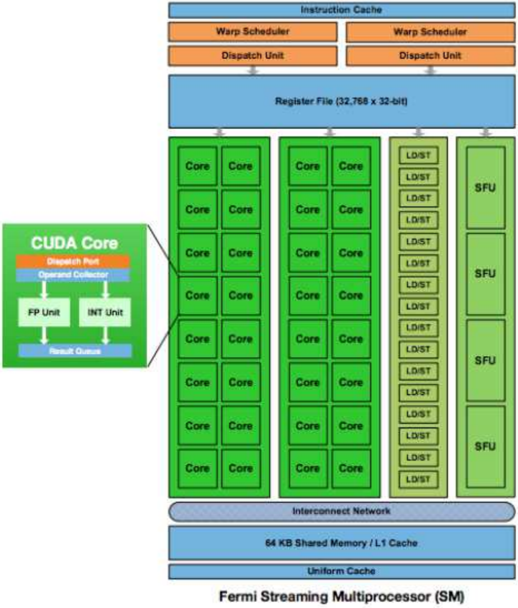


Figure: NVIDIA GPU Architecture(Source: NVIDIA)

Where to focus your study

After this class you should be able to:

- Summarize the types of parallel processor organizations;
- Present an overview of design features of symmetric multiprocessors;
- Understand the issue of cache coherence in a multiple processor system;
- Explain the key features of the MESI protocol.
- Understand the cluster systems and vector systems.

Less important to know how these solutions were implemented:

- details of specific hardware solutions for parallel computation.

Your focus should always be on the building blocks for developing a solution =)

References I

Sanders, J. and Kandrot, E. (2011).

Cuda by Example: An Introduction to General-purpose GPU Programming.

Addison Wesley Professional.

Stallings, W. (2015).

Computer Organization and Architecture: Designing for Performance.

Pearson Education, 10th edition edition.