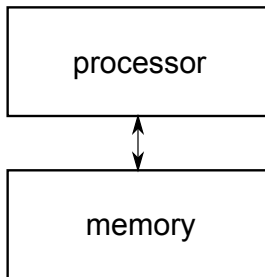


# Parallel Computer Architecture

Pangfeng Liu  
National Taiwan University

May 4, 2020

- Multiprocessor
- Multicomputer
- Flynn's Taxonomy



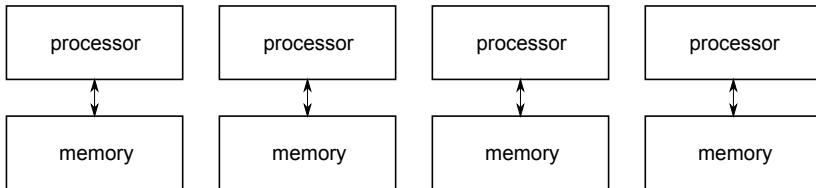
- This is how we think of a computer.

- A single processor for computation.
- A single memory for storing instructions and data.
- The CPU fetches instructions from the memory, executes the instructions, updates the contents of registers, and possibly data in the memory, and then repeats.
- Intel 486, Pentium, etc.

# Multiple Uniprocessors

- It is intuitive to have multiple uniprocessors working together to have high performance.
- Why? It is *expected* that having more processors to work together, we can solve the problem faster.
- How do they work *together*?

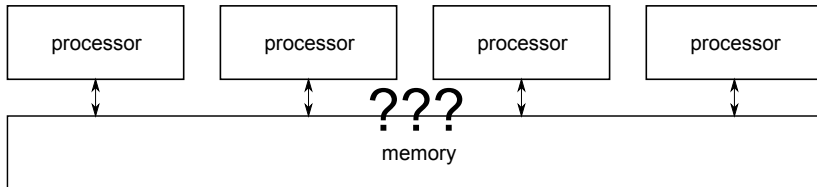
# Work Together?



???

Processors must communicate to coordinate their actions, and exchange data if necessary.

# Work Together?



- We cannot simply connect all processors to a memory.
- The memory will be in an *inconsistent* state.



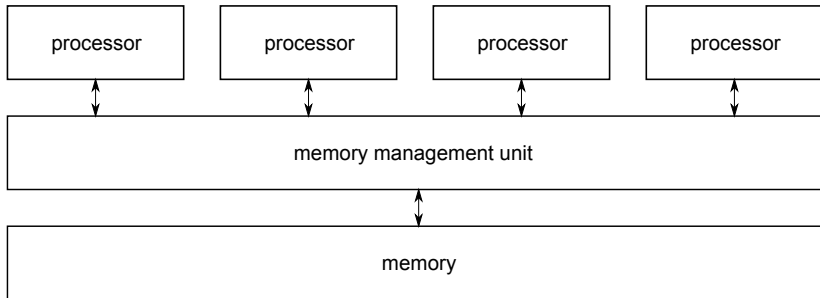
# Race Condition

- Suppose two processors want to add 1 to the same counter.
- The first processor fetches the old contents of the counter, adds 1 to it.
- Before the first processor can store the new content, the second processor fetches the old content.
- The first processor now stores its new content to memory.
- The second processor adds 1 to the old content, and stores the new content back to memory.
- The counter only increases by 1, which is *incorrect*.
- More details later.

# Multiple Uniprocessors

- We put a *memory management/arbitration unit* between the processors and the memory so that every processor can access memory.
- This memory management unit must be very efficient in providing *point-to-point* data transfer so as to provide fast memory access for every processor.

# Multiprocessor



- What could happen if three processors want to add 1, 2, and 4 into a memory with an initialized value 0?

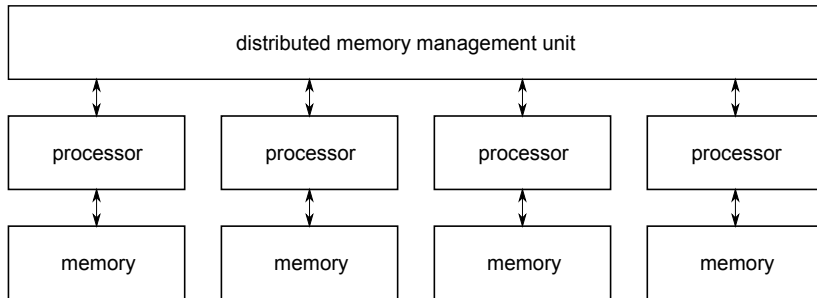
# Shared Memory

- Logically we do not care about the memory management unit – we simply believe that every processor can access this *shared memory*.
- This shared memory provides a *shared addressing space* for all processors.
- This *shared* memory is also a *global memory*.
- The cost for every processor to access every part of the memory is the same, then we have Uniform Memory Access (UMA).

# Distributed Shared Memory

- A distributed version of shared memory.
- Every processor has a local memory, and the collection of the memory form a global memory.
- We connect the processors to a memory management unit, which determines the address is local or remote.

# Multiprocessor



# Distributed Multiprocessor

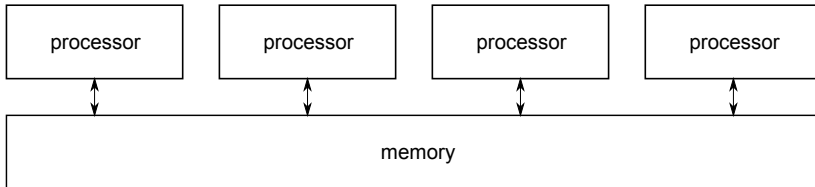
- If the address is local, then it is retrieved from the local memory.
- If the address is remote, then it is retrieved from someone else's local memory.
- The cost for every processor to access every part of the memory is *not* the same, then we have Non-Uniform Memory Access (NUMA).



- Describe the difference between NUMA and UMA.

- Logically we do not care about the memory management unit – we simply believe that every processor can access this *shared memory*.
- This shared memory provides a *shared addressing space* for all processors.
- This *shared* memory is also a *global memory*.

- In summary, we can think of our multiprocessor is like this.



- Conceptually multiple processors are connected by a shared memory – that is all we need to know in terms of programming.
- However, in some programming models we still need to consider the possibility of race condition, and use the construct provided by the programming model to avoid it.

- Every processor has its own computing resource, like ALU, registers, etc, so you can run multiple processes on them simultaneously.
- Every processor works on the tasks assigned to it, using its own computing resource.
- Every processor can read and write the shared memory, so as to communicate or synchronize with other processors.

# Synchronization

- Processor can synchronize with each other by a shared memory.
- For example, in a barrier synchronization, a processor cannot proceed until all others have reached the same conclusion.
- We can set a shared variable count. Every finished processor add 1 to count. When the value of count reaches the number of processors then every processor knows that it has synchronized with everyone.

- Describe the dependency graph using *functions* and *synchronization*.

# Memory Conflict

- Multiple processors can access the memory at the same time, causing conflicts.
- When a computation has different outcome due to different execution order, we have a *race condition*.



# An Example

- Suppose two processors want to add their variable  $n$  together into a global variable  $sum$ .
- Processor 1 has  $n = 3$  and processor 2 has  $n = 4$ .
- The computation consists of the following.
  - Load  $sum$  into a register  $r1$ .
  - Add your  $n$  into  $r1$ .
  - Store register  $r1$  back to  $sum$ .

# Race Condition

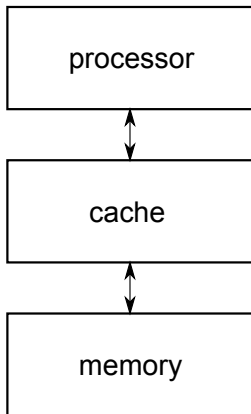
- Now imagine what will happen when P1 and P2 are doing this simultaneously.
- How many different outcomes could there be? Please try to enumerate them.

- Programming environment must provide mechanism that prevents race condition.
- Critical section, lock, synchronization, etc.
- More specific details will be provided when we discuss parallel programming.

- Describe the concept “critical section”.

- To make the case even worst, we need to deal with cache.
- Cache are fast memory – usually they are small and expensive.
- To improve performance we have cache for those often used data/instructions. If we need the data/instructions again we can access them in fast cache, instead of slow main memory.

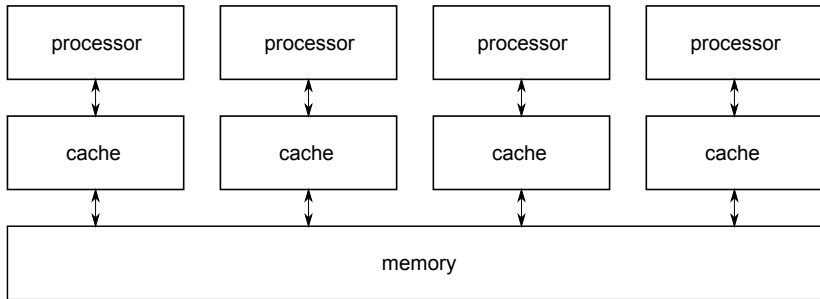
# Cache for Uniprocessor



# Race Condition

- Now every processor has a cache, and it can take data from its cache, not the main memory.
- Suppose one processor changes the content of the memory, what should happen to the cached data in some processor's cache?
- If one processor changes the data in its cache, would other processor be able to notice this change?

# Cache





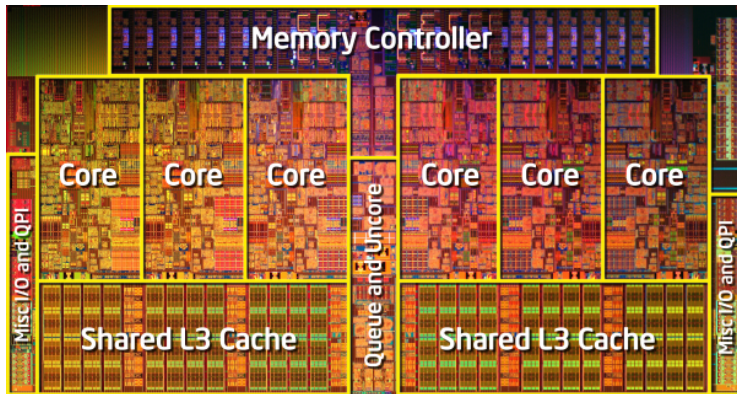
- The hardware must guarantee that the memory and cache are in a consistent state. There are various levels of guarantees.
- If one processor changes the content of the memory, the hardware should invalidate data that have been cached in other processor.
- If one processor changes the data in its cache, other processor should be able to see it.

**Write-through** Write is done synchronously both to the cache and to the backing store.

**Write-back** Initially, writing is done only to the cache. The write to the backing store is postponed until the cache blocks containing the data are about to be accessed by others.

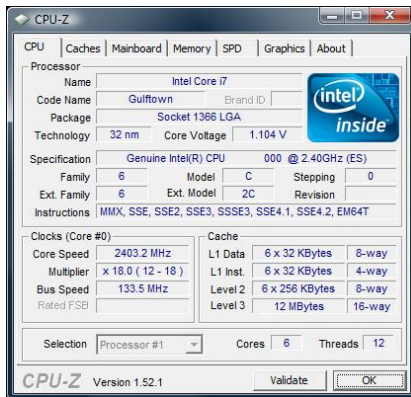
- Describe the difference between “Write-through” and “Write-back” caching.

- Core i7-9xx
- 6 cores
- 32nm
- 12MB L3 cache
- Introduced January 2011.



1

<sup>1</sup><http://gizmodo.com/5491045/intels-6+core-gulftown-gets-tested-blows-us-away>



2

<sup>2</sup><http://global.hkepc.com/database/images/2009/08/source/1015564707405292968.jpg>

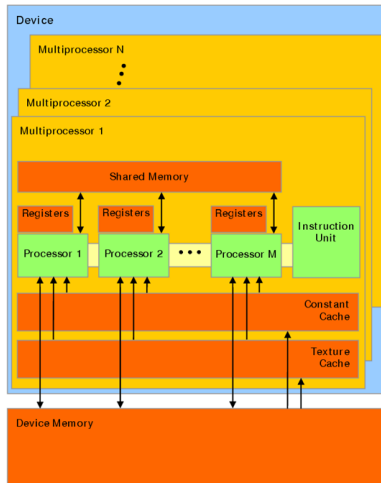
- The memory controller controls and coordinates the access to a shared memory.
- Cores can communicate with the queue.
- Two L3 caches, and each is shared by three cores.
  - Shared L3 cache indicates that if we place wrong processes/threads into the cores that share the L3 cache, the performance will suffer.

- You can find all the facts of Intel CPU here.
- <http://www.intel.com/pressroom/kits/quickreffam.htm>



- The Tesla graphics processing unit (GPU) is Nvidia's third brand of GPUs.
- Tesla is based on high-end GPUs from the G80 (and on), as well as the Quadro lineup.
- Tesla is Nvidia's first dedicated *General Purpose GPU* (GPGPU).
- [http://en.wikipedia.org/wiki/Nvidia\\_Tesla](http://en.wikipedia.org/wiki/Nvidia_Tesla)

# GPU Architecture



A set of SIMT multiprocessors with on-chip shared memory.

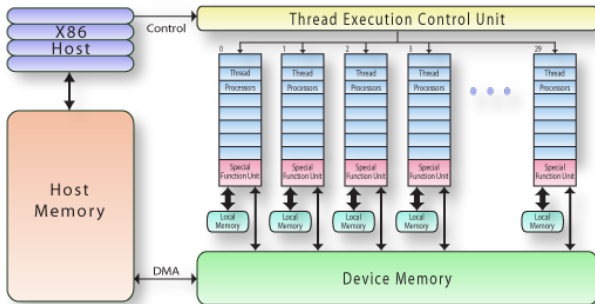
Figure 3-1. Hardware Model

# Nvidia Tesla



- 240 processors at 1.30 GHz.
- 4096 MB of GDDR3 memory.
- 102.4 GB/s memory bandwidth.
- 933.12 GFLOPs single precision, 77.76 GFLOPs double precision.

# Tesla Architecture



# Characteristics

- GPU is a device, and it needs a host to get its data/instructions.
- A large number of cores – usually much larger than a CPU.
- Instructions are streamed to processors for execution, which means they must run the same set of instructions.
- Processors have their local memory, as well as access to a shared device memory.
- More details in the “OpenCL” lectures.

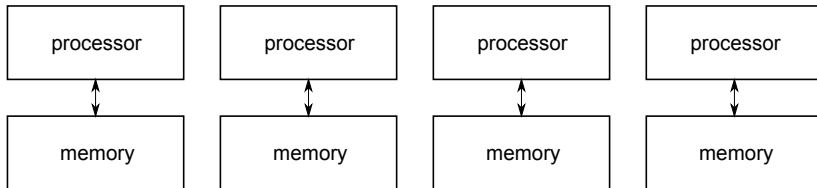
- Describe the difference between CPU and GPU.

# Multiple Uniprocessor

- It is intuitive to have multiple uniprocessors working together to have high performance.
- How do they work *together*?



# Work Together

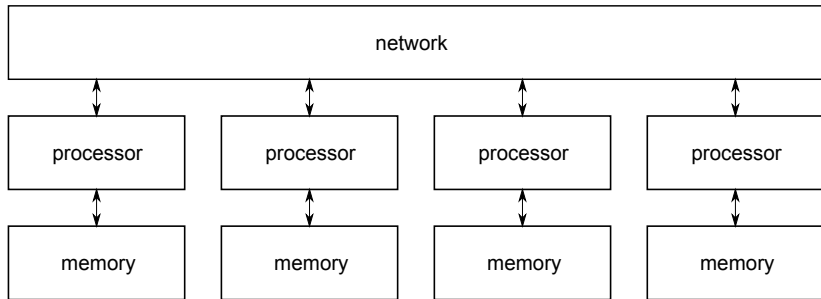


???

# Multiple Uniprocessor

- We connect all the computers with a *network* so they can send *messages* to each other.
- This network could be very slow or very fast, depending on the applications.
- The network must provide *point-to-point* data transfer to move data from one processor to another.
- Processors can only synchronize themselves via the network, which is a difficult task.

# A Network



- Every processor works on the tasks assigned to it, using its own computing resource.
- Every processor can send message to each other, so as to communicate or synchronize with other processors.

- Sometimes we use the term “node”, since these processors are by themselves “computers”.
  - Remember the 16,000 nodes in the Tianhe 2 cluster.
- Each node has its own CPU's, memory, even I/O devices.
- The nodes can communicate with each other by the network, usually through standard TCP/IP protocol.

- There is no *shared addressing space* for all processors – each processor use its own memory.
- The memory of every processor is called *local memory*.
- Since a memory is accessed by only one processor, we do *not* have memory conflict.

# Synchronization


- Processor can synchronize with each other by messages.
- In a barrier synchronization, a processor cannot proceed until all others have reached the same conclusion.
- We can ask every finished processor to send a message to a *master* when it is done.
- When the master receives a message from every processor, he knows everybody has finished.
- The master sends a message to everyone that it can continue.

- Compare the way multicomputer and multiprocessor does a barrier synchronization.



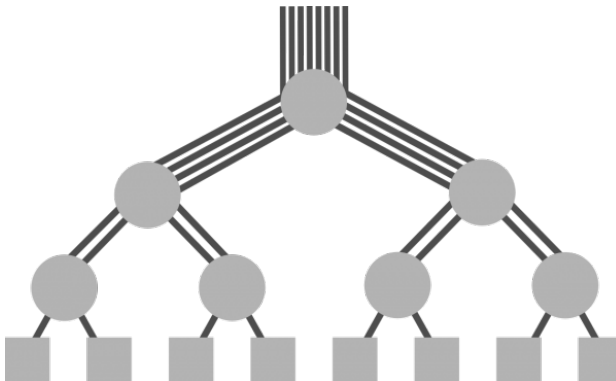
- We do care about the network – a fast network provide better connectivity.<sup>3</sup>
  - Ethernet 10Mbit/s
  - Fast Ethernet 100Mbit/s
  - Gigabit Ethernet 1 Gbit/s
  - 10 Gigabit Ethernet 10 Gbit/s
  - Myrinet 10 Gbit/s
  - 100 Gigabit Ethernet 100 Gbit/s
  - InfiniBand (12X EDR) 300 Gbit/s

---

<sup>3</sup>[http://en.wikipedia.org/wiki/Network\\_bandwidth](http://en.wikipedia.org/wiki/Network_bandwidth) 

- We cannot connect a large number of nodes to a single switch, so the topology of the network becomes important.
  - Ring
  - Tree and fat tree
  - Two or higher dimensional mesh and torus
  - Hypercube
  - FFT (butterfly)

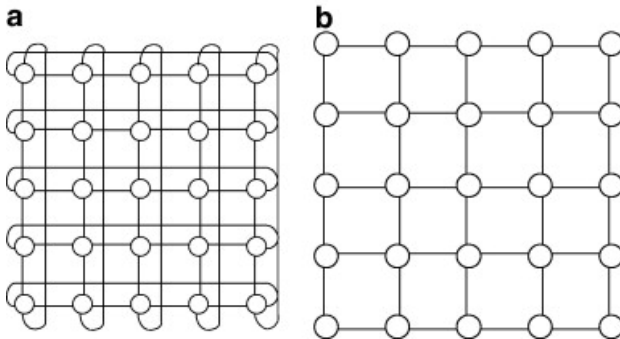
# Fat Tree



4

<sup>4</sup>[http://clusterdesign.org/wp-content/uploads/2012/02/fat\\_tree\\_varying\\_ports-600x365.png](http://clusterdesign.org/wp-content/uploads/2012/02/fat_tree_varying_ports-600x365.png)

# Mesh and Torus

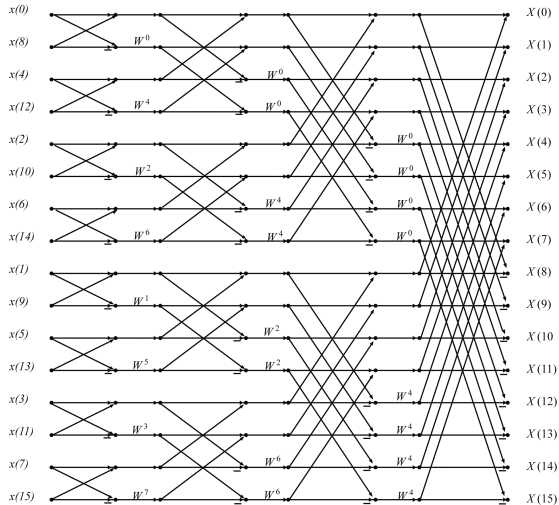


5

<sup>5</sup>[http:](http://ars.els-cdn.com/content/image/1-s2.0-S1383762107000495-gr2.jpg)

[//ars.els-cdn.com/content/image/1-s2.0-S1383762107000495-gr2.jpg](http://ars.els-cdn.com/content/image/1-s2.0-S1383762107000495-gr2.jpg)





- The most popular topology appears to be fat tree.
  - For example, the Tianhe 2 cluster connects all its processors with 13 switches as fat tree.
- Two or higher dimension torus are also popular.
  - The Tofu network of K-computer is a six dimensional torus.

- What is the difference between torus and a mesh?

# Massively Parallel Computer

- A computer cluster consists of a set of *tightly* connected *high performance* computers that work together so that in many respects they can be viewed as a single system.
- Tightly connected means they are connected by *fast* network.
- The performance is paramount, and usually achieved by aggregation of a large number of processors.

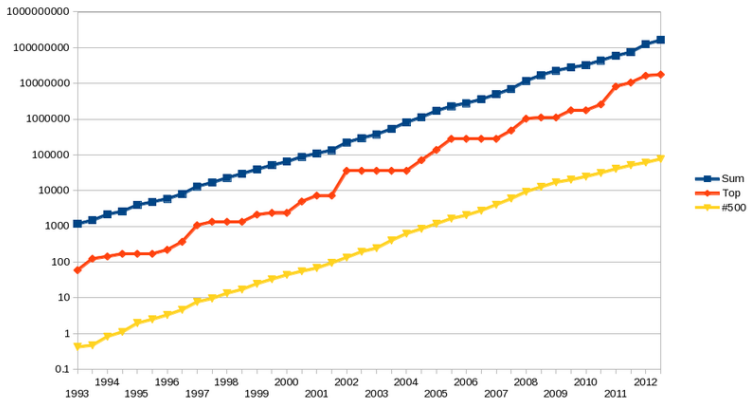


- The TOP500 project ranks and details the 500 most powerful (non-distributed) computer systems in the world.
- The project aims to provide a reliable basis for tracking and detecting trends in high-performance computing and bases rankings on HPL, a portable implementation of the High-Performance LINPACK benchmark written in Fortran for distributed-memory computers.<sup>7</sup>

---

<sup>7</sup><http://top500.org/>

# Top 500



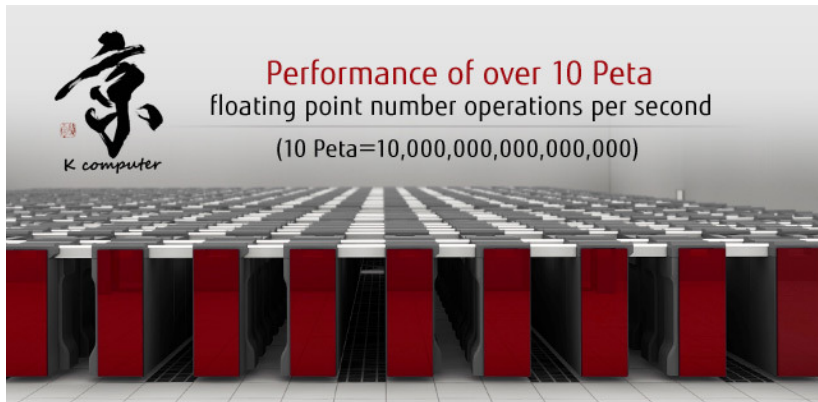
- Exponential growth of supercomputers performance, based on data from top500.org<sup>8</sup>.

<sup>8</sup><http://en.wikipedia.org/wiki/File:Supercomputers.png>

- The K computer – named for the Japanese word “kei” (京), meaning 10 quadrillion ( $10^{16}$ ).
- A supercomputer manufactured by Fujitsu, currently installed at the RIKEN Advanced Institute for Computational Science campus in Kobe, Japan.
- In June 2011, TOP500 ranked K the world’s fastest supercomputer, with a rating of over 8 petaflops, and in November 2011, K became the first computer to top 10 petaflops.<sup>9</sup>

---

<sup>9</sup>[http://en.wikipedia.org/wiki/K\\_computer](http://en.wikipedia.org/wiki/K_computer)



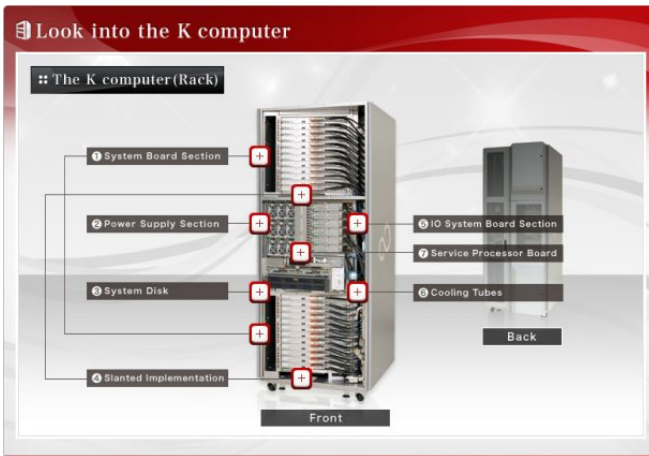
10

---

<sup>10</sup>[http://cdn0.sbnation.com/entry\\_photo\\_images/2197300/k-computer\\_large\\_verge\\_medium\\_landscape.jpg](http://cdn0.sbnation.com/entry_photo_images/2197300/k-computer_large_verge_medium_landscape.jpg)

# Configuration

- 864 cabinets, 88,128 SPARC64 VIIIfx processors, over 640,000 cores.
- A proprietary six-dimensional torus interconnect called Tofu.
- A two-level local/global file system with parallel/distributed functions, which provides users with an automatic staging function for moving files between global and local file systems.
- Linux operating system.
- 9.89 MW – the equivalent of almost 10,000 suburban homes.



11

<sup>11</sup><http://cdn-static.zdnet.com/i/story/30/40/093162/k-computer-riken-4.jpg>

- What is the difference between K-computer and Tianhe 2?

- A computer cluster consists of a set of *loosely* connected computers that work together so that in many respects they can be viewed as a single system.
- Loosely connected means they are *not* connected by fast network.
- An economical alternative to those who cannot afford expensive parallel computers.



- A cluster donated by TrendMicro for the Cloud Computing Program.
- Loosely connected means they are not connected by fast network.
- A economical alternative to those who cannot afford expensive parallel computers.

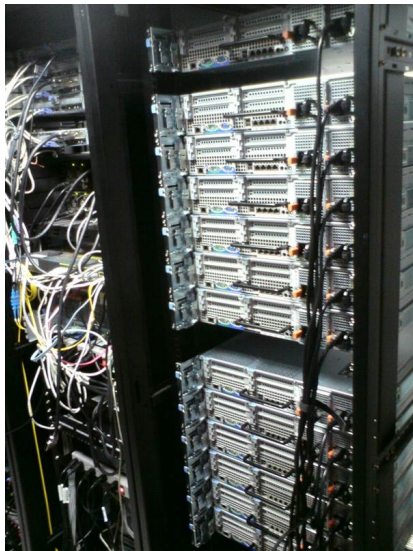
# Configuration

- 1 cabinet, 15 Intel X5570 quad-core processors, 120 cores.
- A standard Ethernet switch.
- Gluster file system.
- Roystonea operating system, developed by Parallel and Distributed Processing Laboratory, Department of Computer Science and Information Engineering, National Taiwan University.
- Never made it to top 500.

# TrendMicro Cluster



# TrendMicro Cluster



- We (the Parallel and Distributed Processing Laboratory) learn many things in building Roystonea for the Trend cluster.
- The things we learned include network virtual machine management, virtual machines deployment, network management for cluster, distributed file system for cluster, distributed database and NoSQL database for cloud system.
- We built a cloud OS called “Roystonea” to manage the Trend cluster.
- These experiences later enables us to take on more ambitious projects, like the optimization of billing system of ChungHwa Telecommunication.

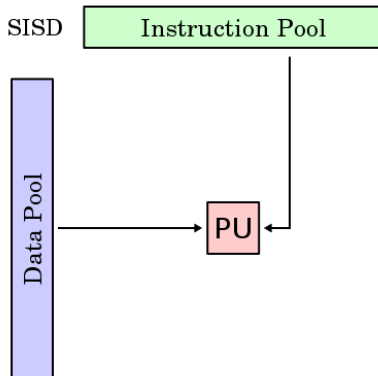
- What does the name “Roystonea” come from?

# Flynn's Taxonomy

- Single instruction stream and multiple instruction stream.
- Single data stream and multiple data stream.
- We have four combinations – SISD, SIMD, MISD, and MIMD.

- SISD (single instruction, single data)
- A computer architecture in which a single uniprocessor executes a single instruction stream to operate on a data stream.
- Standard von Neumann architecture.

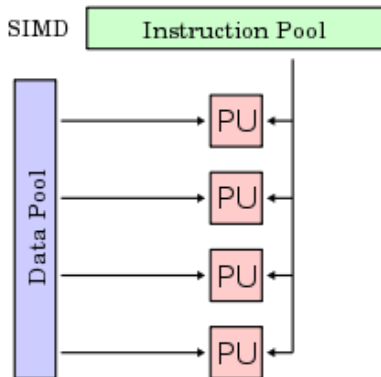




- This is how we have been doing – sequential programming.
- The compiler for sequential programming is quite mature and can transform source program into efficient binaries.
- System support for sequential programming, e.g., system call, user library, debugging are also very useful.

- Give an example of SISD machine.

- Single instruction, multiple data (SIMD)
- A computer architecture in which multiple processors execute a single instruction stream to operate on multiple data streams.



13

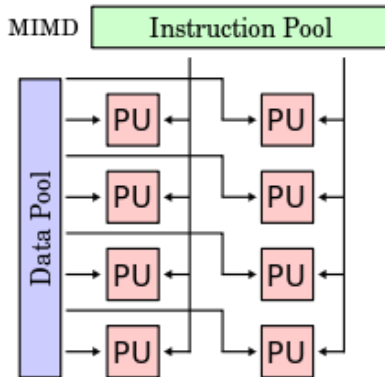
<sup>13</sup><http://en.wikipedia.org/wiki/SIMD>

- Strongly related to data parallel programming since the same instruction is applied on different data, so as to achieve performance by data parallelism.
- The architecture of GPUs follows the SIMD model – the host issues the same command to all processors so as to process a large amount of data simultaneously.

- Give an example of SIMD machine.

- Multiple instruction, multiple data (MIMD)
- A computer architecture in which multiple processors execute multiple instruction streams to operate on multiple data streams.





14

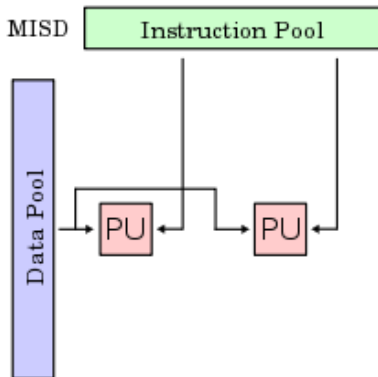
<sup>14</sup><http://en.wikipedia.org/wiki/MIMD>

- Strongly related to functional parallelism since different processors execute different instructions on different data.
- The instructions are different because they are from different tasks in a wavefront, i.e., tasks that can be done in parallel.
- The data are different because they are for different tasks.
- Most multicomputers support MIMD computation model – computers work independently, and synchronize themselves when necessary.

- Give an example of MIMD machine.

Wait! You forgot MISD!

- Multiple instruction, single data (MISD)
- A computer architecture in which multiple processors execute multiple instruction streams to operate on a single data stream.
- Does it make sense to you?



15

- In fact if we consider MISD as multiple copies of SISD, then it can tolerate faults.
- The same computation is repeated multiple times, by different processors, so that at least of them can deliver the results, in case some of them fail.
- For example, in Google MapReduce computation certain tasks are duplicated exactly for the purpose of fault tolerance and performance improvement.

- Give an example of MIMD machine.