

Parallel Computer Architecture (brief overview)

Jarmo Rantakokko

Aim: Give an overview of different types of systems and to give understanding on what performance we can expect from these



What is a parallel computer?

Answer 1: A set of connected processors/cores

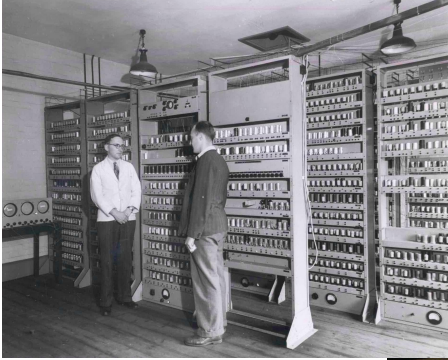
Answer 2: "All" modern computers are parallel computers (even single core computers)

Compare:

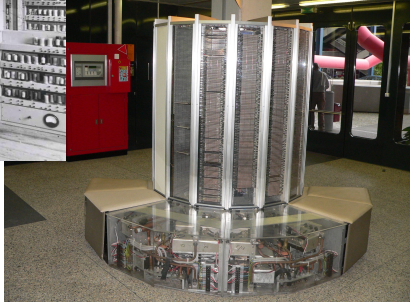
• EDSAC	(1949)	500kHz	100 Flop/s
• Cray 1	(1979)	80 MHz	4 Mflop/s
• Cray 1	(1983)	80 MHz	12 Mflop/s
• Intel P4	(2004)	3.8 GHz	7.6 Gflop/s
• Intel i7	(2010)	2.3 GHz	37 Gflop/s
• Nvidia	(2012)	732MHz	3.95 Tflop/s

Explanation: Interior parallelism!


UPPSALA
UNIVERSITET



EDSAC
(Priceless, one of a kind)



Cray 1 (\$5,000,000)



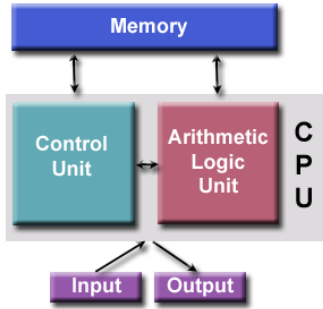
Nvidia (\$120)

UPPSALA
UNIVERSITET

The von Neumann model

For each instruction:

1. Fetch instruction
2. Decode instruction
3. Fetch operands
4. Execute instruction
5. Write result back



Observations (problems)

1. Each stage is performed sequentially
2. A lot of traffic to and from memory

- > Several cycles for an instruction to complete.
- > Slow devices/operations stalls execution.
- > Contention on the data bus to memory.



Solutions to run faster:

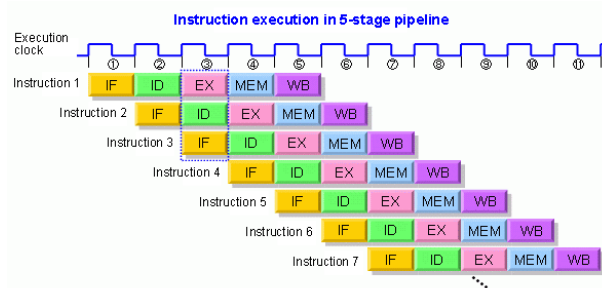
1. Increase clock rate
 - ⇒ Slow devices (memory) still stalls execution
 - ⇒ More contention on the memory bus
 - ⇒ High power consumption
 - Increase freq, increase voltage, $p \sim f \cdot v^2$
 - (Energy cost, cooling problem, battery time)
2. Introduce parallelism
 - a. Within a CPU
 - b. Multiple CPUs




Parallelism within a processor:

1. **Parallel busses**
 - Wider data bus
 - Separate data and instruction bus

2. Instruction pipeline



⇒ One instruction per time unit



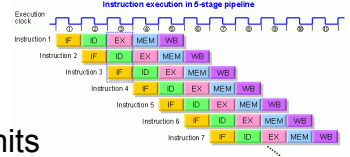
How does the performance depend on the pipeline?


- Largest machine instruction limits the time unit (the speed of pipeline)
- Uniform stages (no dead time, higher efficiency)
- Number of stages (more parallelism/speedup)
- Number of consecutive instruction (startup time minor)

Short, uniform instruction length => RISC processor (Reduced Instruction Set Computer)

Super-pipelined processor: Extremely short and simple instructions (long pipelines). Can run with higher frequency.

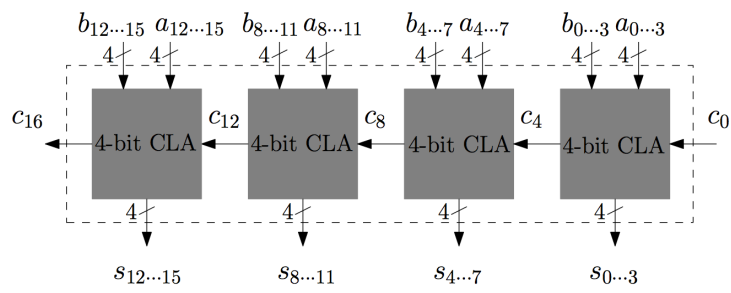
Problem: Branches in code, disrupts the pipeline
Branch prediction in hardware





3. Arithmetical pipelines

Floating point operations are heavy operations, can be decomposed into smaller operations, for example, decomposing an adder to adding a few bits at a time.



Useful for loops with arithmetic operations

```
for (i=0; i<n; i++)
    s[i]=a[i]+b[i];
```

Without pipeline: 4n t.u.
With pipeline: 4+n-1 t.u.



4. Multiple operating units

Multiple units for floating point operations (add,mult), integer arithmetic unit, logic unit, branch unit, etc,

⇒ Parallel instruction stream with 2-4 flops/cycle

Problem:

Serial sections (ordered instructions) limit performance.

- Use *out-of-order* and *speculative* execution in hardware, i.e., precompute results without knowing if the instruction will execute or not.
- Use multiple software threads and fill the units with instructions from the parallel threads, *multithreading*.



MEMORY PROBLEM:


Enhancements (1) - (4) => Memory can't keep up delivering data at processor speed!

Types of memory

DRAM: Dynamic Random Access Memory
 Charged based devices, needs to be refreshed at read/write – takes time $\sim 10^{-8}$ s
 Cheap but slow, use for main memory.

(Improvements: SDRAM, DDR SDRAM, RDRAM using multiple memory banks, can overlap accesses)

SRAM: Static Random Access Memory
 Gate based devices (transistors)
 No need for refreshment, fast but expensive, use for registers and cache


 UPPSALA UNIVERSITET

Memory Hierarchies

To keep costs down, create memory hierarchy:

Speed	↑	Registers	Head
		Caches	Address book
		Main memory	Phone book
Size		↓	Virtual mem (disk)

Analogy: Memory hierarchy – Phone numbers

 UPPSALA UNIVERSITET

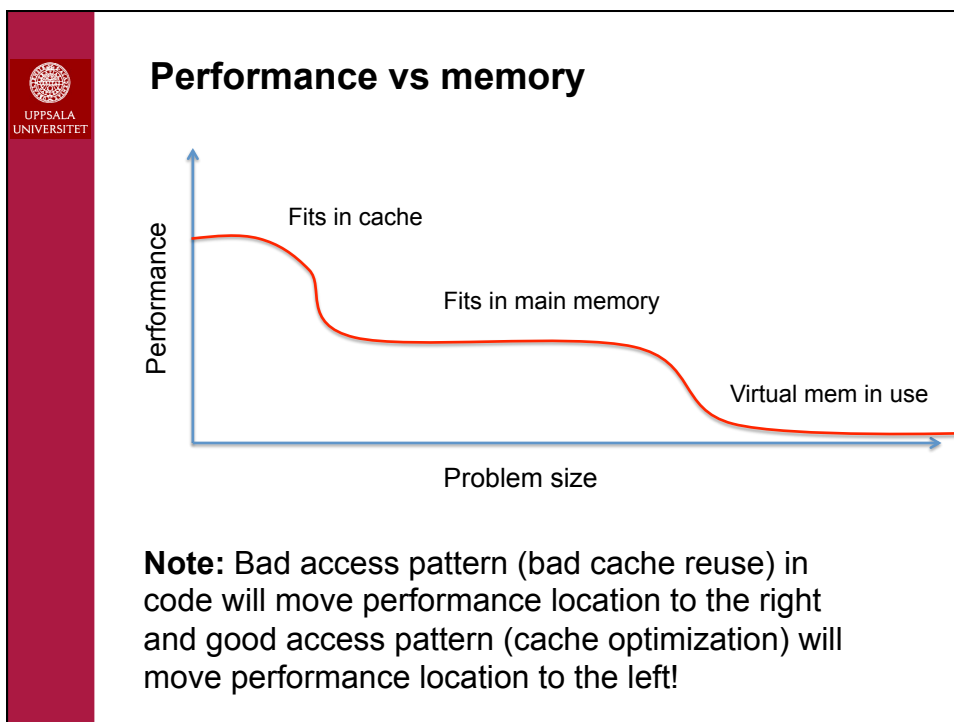
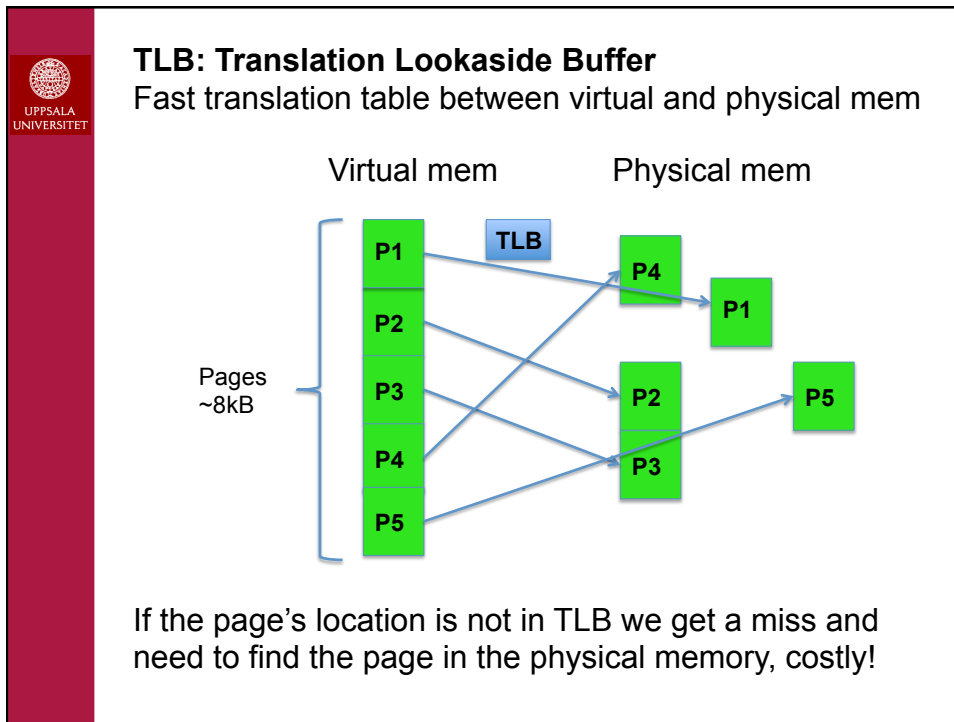
Caches: L1 Cache (~128kB, on chip)
L2 Cache (~4MB, on/off chip)
L3 Cache (+8MB, off chip)

Types of caches (techniques to store/replace data):

- direct mapped – pages alphabetical order
- fully associative – blank pages
- set associative – free sections, each section ordered

Memory hierarchies makes it very important with data layout and data accesses in your codes!

Register	
Cache	➤
Main memory	➤ Cache miss
Virtual memory	➤ Page fault, TLB miss



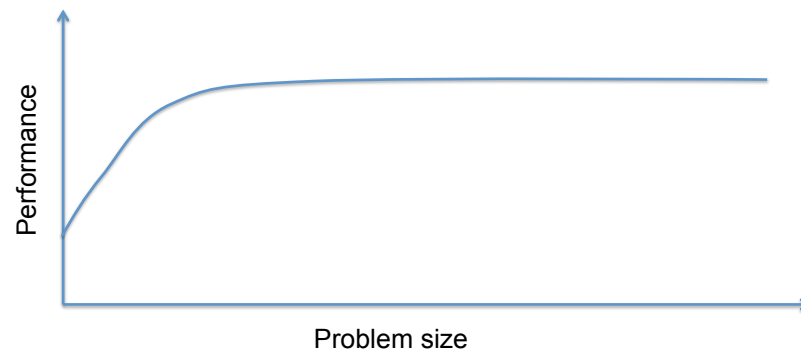


Vector processors

1. Vector instructions
Vadd, vmult, etc (complex instructions)
2. Vector registers
Compare scalar registers in RISC
3. Memory pipelines (1024 memory banks)
Pipeline memory accesses, no need for caches, deliver data at clock speed
4. Vector units
Arithmetical pipelines
5. Compiler directives for vectorization of code
Similar as for OpenMP



Performance vs memory



Very expensive, manufactured in small volumes, limited to numerical applications with long vectors.

Ex: Cray 1 (1976), Cray X1 (2003), Nec SX, Fujitsu VPP, Earth Simulator (#1 at top500 in 2002-2004)



In summary:

A “traditional” processors is very parallel!
 But, the parallelism is on a low level and “hidden” for programmer. Compiler exploits it with loop unrolling, reordering, merging, splitting etc., if using aggressive optimization flags (-fast, -O3, -O5, -unroll, or similar).
Manual tuning of code is still necessary!

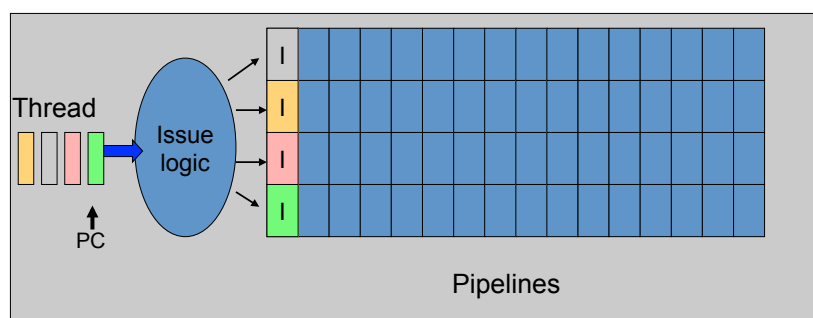
Moreover, **cache optimization** becomes increasingly important, e.g., by exploiting temporal and spatial data locality and cache blocking, we can easily improve performance with a factor of 10x.

⇒ Huge gap between practical performance and theoretical peak performance (marketing numbers).

(Course *High Performance Computing and Programming*, 5hp, period 4)



Multicore processor design



In the “traditional” processor we run one thread and issue instructions from this to the multiple pipelines.



Problems with “traditional” processor design:

#1: Running out of ILP

Not enough instructions to feed pipelines

#2: Wire delay is starting to hurt

Thinner wires, higher resistance, slower speed

#3: Memory is the bottleneck

Slow memory accesses stalls the execution

#4: Power is the limit, $P \sim F \cdot V^2$

Cooling problem, high energy cost, low battery time



Solving all the problems, exploring parallelism:

#1: Running out of ILP

> Feed one CPU with instructions from many threads

#2: Wire delay is starting to hurt


> Multiple small cores with private L1\$, shorter paths

#3: Memory is the bottleneck

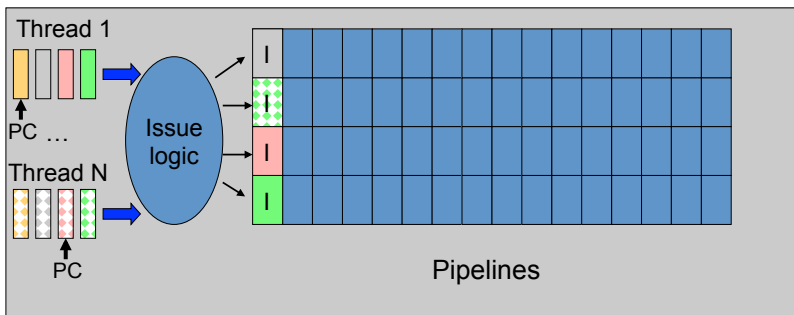
> Overlap memory accesses from many threads

#4: Power is the limit

> Multiple cores, lower F and V, better performance




SMT: Simultaneous MultiThreading

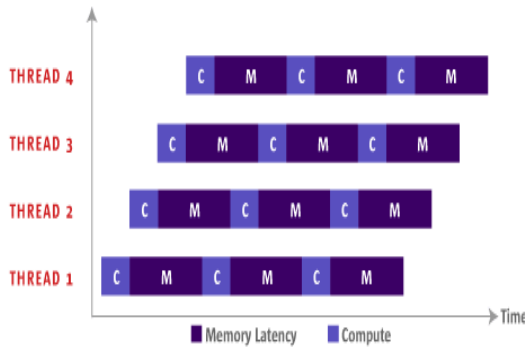


The diagram illustrates the SMT architecture. On the left, multiple threads (Thread 1 to Thread N) are shown, each with its own Program Counter (PC) and a queue of instructions. These threads feed into a central 'Issue logic' block. The issue logic then distributes instructions to a set of parallel 'Pipelines'. Each pipeline is represented by a column in a grid, and each thread's instructions are shown as colored blocks (yellow, red, green) being fed into the pipelines. The label 'Pipelines' is centered below the grid.

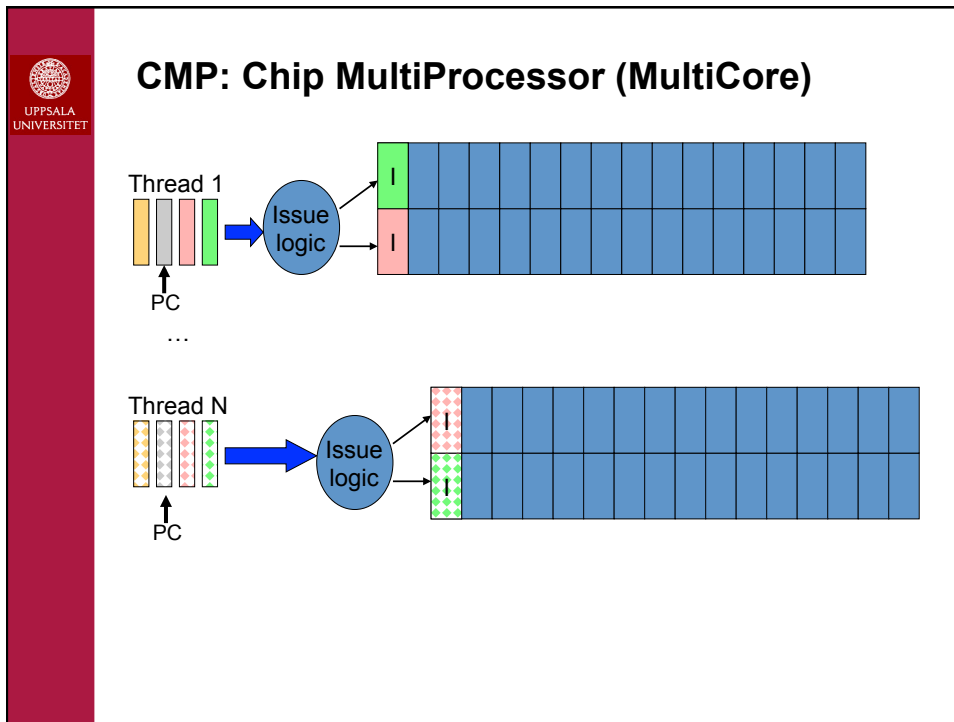
Can feed the multiple pipelines with instruction from many threads.



Overlap memory accesses with computations



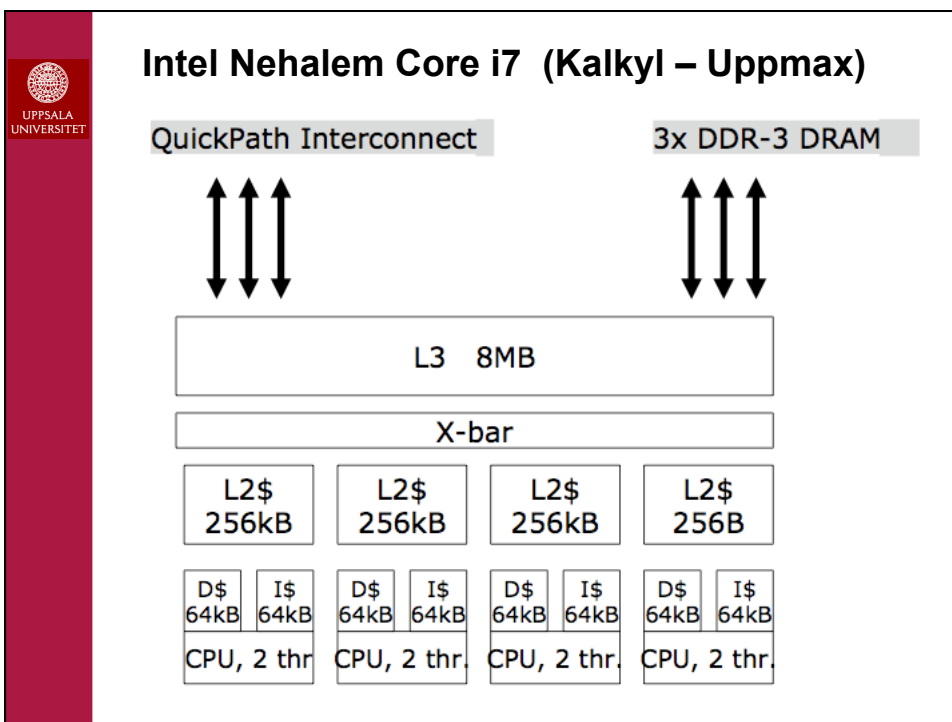
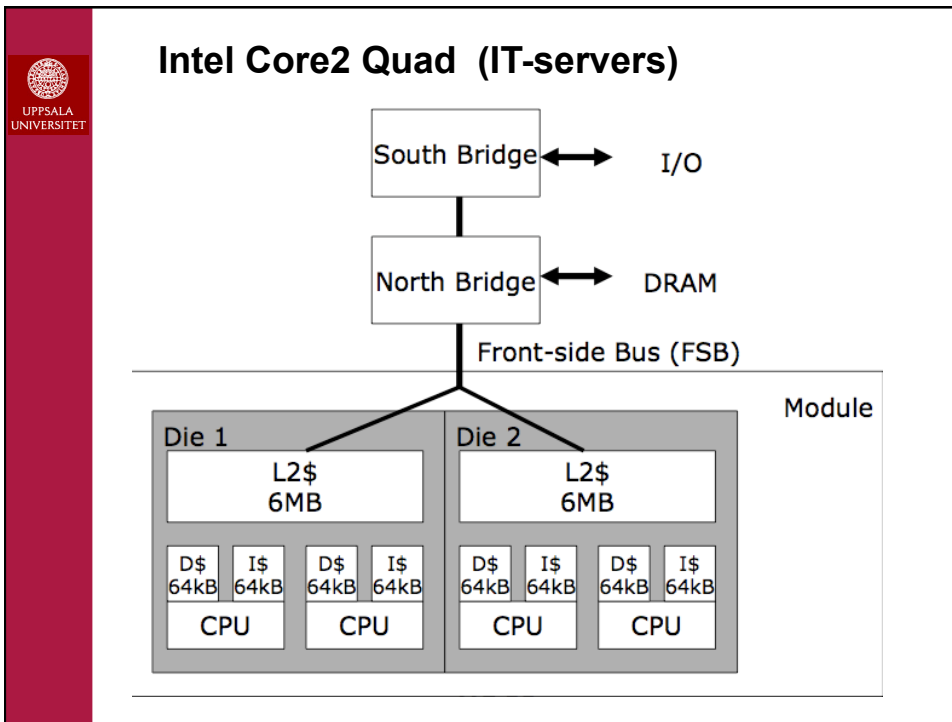
The Gantt chart shows the execution of four threads (THREAD 1 to THREAD 4) over time. The horizontal axis represents 'Time'. Each thread's execution is represented by a horizontal bar divided into segments of 'Memory Latency' (dark purple) and 'Compute' (light purple). The segments are interleaved across threads, demonstrating how memory latency for one thread can be overlapped with compute time for another. The legend at the bottom indicates that dark purple represents 'Memory Latency' and light purple represents 'Compute'.

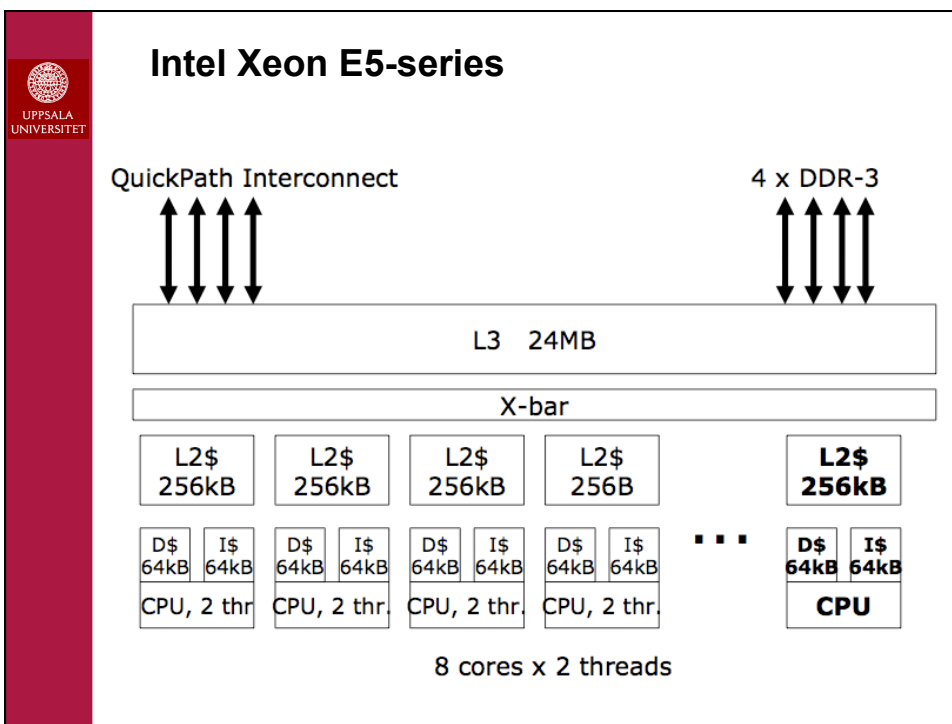
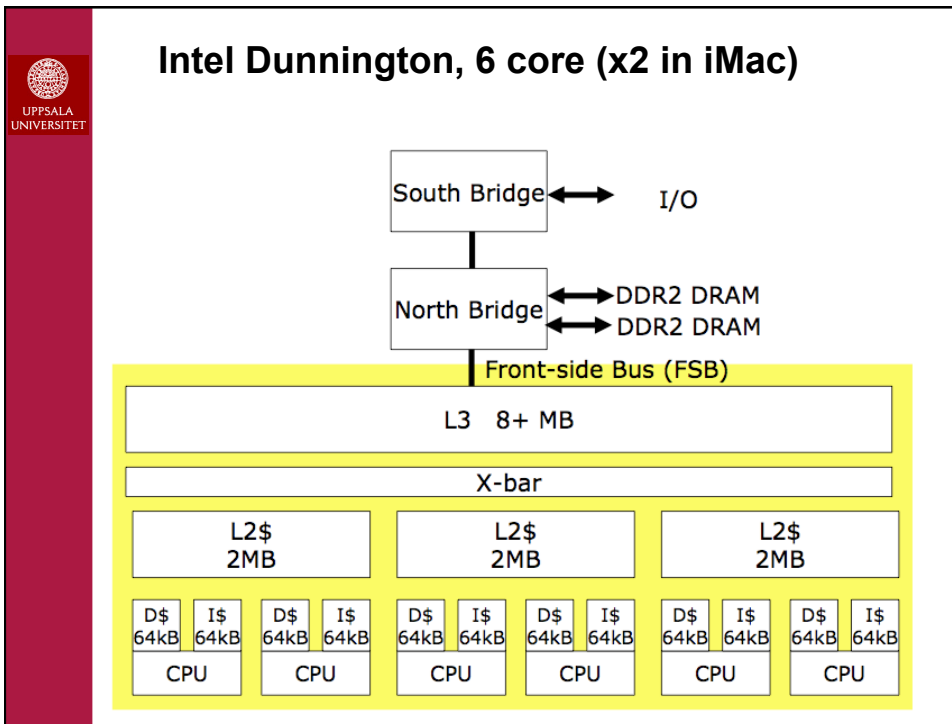



Consequences:

- # Non-parallelized programs will not utilize the full potential > Waste of resources
- # Non-parallelized programs will run slower on the new CPUs (lower frequency)

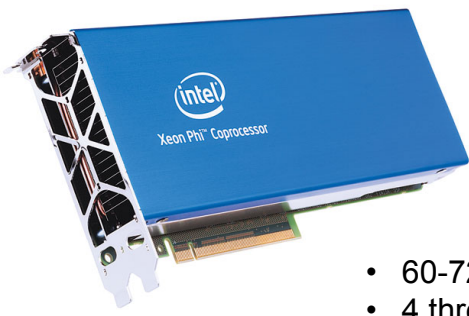
=> Need parallel programming even for one single CPU !














Intel Xeon Phi Co-processor (Many Integrated Core Architecture)



- 60-72 Compute cores
- 4 threads per core
- Up to 1.2 Tflops
- Tianhe-2 Supercomputer



x86 64-bit Architecture Evolution

	2003	2005	2007	2008	2009	2010
	AMD Opteron™	AMD Opteron™	"Barcelona"	"Shanghai"	"Istanbul"	"Magny-Cours"
Mfg. Process	90nm SOI	90nm SOI	65nm SOI	45nm SOI	45nm SOI	45nm SOI
CPU Core	K8 	K8 	Greyhound 	Greyhound+ 	Greyhound+ 	Greyhound+ 
L2/L3	1MB/0	1MB/0	512kB/2MB	512kB/6MB	512kB/6MB	512kB/12MB
Hyper Transport™ Technology	3x 1.6GT/s	3x 1.6GT/s	3x 2GT/s	3x 4.0GT/s	3x 4.8GT/s	4x 6.4GT/s
Memory	2x DDR1 300	2x DDR1 400	2x DDR2 667	2x DDR2 800	2x DDR2 1066	4x DDR3 1333

	2010	2011	2012	2013
6000 Series CPU	Magny-Cours 8- to 12-Core	Interlagos 12- to 16-Core	Terramar Up to 20-Core	Dublin Up to 20-Core

gullviva.it.uu.se: 1 CPU with 16 cores
tintin.uppmax.uu.se: 320 CPUs with 8 cores



Graphical Processing Units (GPU)

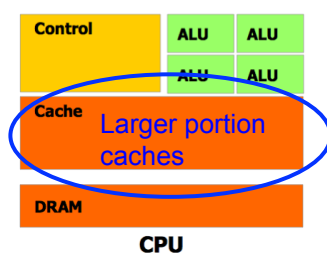
Main vendors NVIDIA, AMD

Architectural features:

- Many simple processing elements (16-2668)
- 1000s – 1 000 000s of threads
- Hardware thread scheduling (1 cycle)
- Focus on throughput (data parallel tasks)
- Limited memory (small on chip mem)
- Limited bandwidth CPU \leftrightarrow GPU (bottleneck)



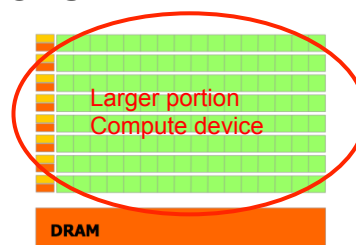
CPU vs GPU



CPU

- Big caches, hierarchy
- Branch predictors
- Out-of-order
- Multiple-issue
- Speculative execution
- Double-precision


Reduce mem latency with caches and hide with other instructions




GPU

- None or small caches
- 1000's of threads
- 1 cycle context switch
- SIMT instructions (32 threads)
- Single precision

Hide mem latency with work from other threads

 UPPSALA UNIVERSITET

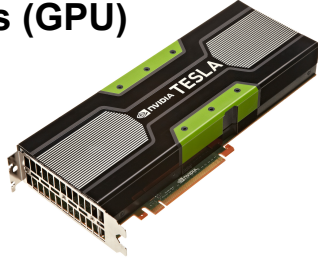
Graphical Processing Units (GPU)



NVIDIA GeForce 650M

- 384 cores
- 650 Gflops SP
- 1 Gbyte Mem
- Power 64W
- Price \$120

(C.f. CPU: 4 core, 2.3GHz,
4 flop/cycle => 37 Gflops DP)




NVIDIA Tesla K20X

- 2668 cores
- 3.95 Tflops SP (1.31 DP)
- 6 Gbyte Mem
- Power 235W
- Price > \$3200

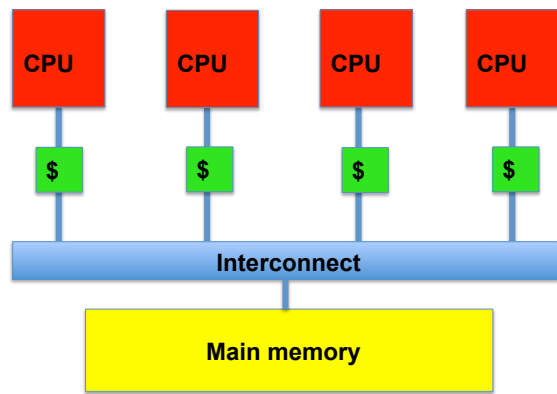
(Used in Titan supercomputer)

GPU is a significant source of computational power!

 UPPSALA UNIVERSITET

Models for Parallel Computers

1. Shared memory (multiprocessor)



```

graph TD
    subgraph CPUs
        C1[CPU]
        C2[CPU]
        C3[CPU]
        C4[CPU]
    end
    C1 --- B1[ ]
    C2 --- B2[ ]
    C3 --- B3[ ]
    C4 --- B4[ ]
    B1 --- I[Interconnect]
    B2 --- I
    B3 --- I
    B4 --- I
    I --- MM[Main memory]
  
```

Ex: IT-servers, 2 proc sharing memory



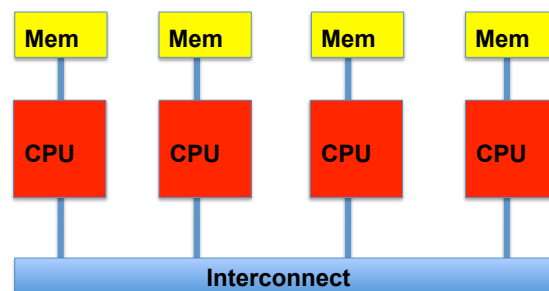
Problem:

- Not scalable (above 64 proc)
Memory bottleneck becomes worse
- Cache-coherency problem (CC)
Several processors have the same data element in its cache and one changes the data, the other caches must be invalidated.
(Handled with snoop or directory based protocols.)

Important problem to consider when programming OpenMP, we have *true sharing* and *false sharing* effects which results in communication of invalid data in caches.



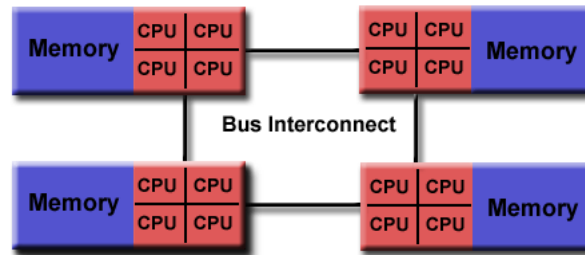
2. Distributed memory (multicomputers)



Separate memory for each CPU
Ex: PC-cluster with Ethernet interconnect



3. Cluster of SMP's (symmetric multiprocessor)



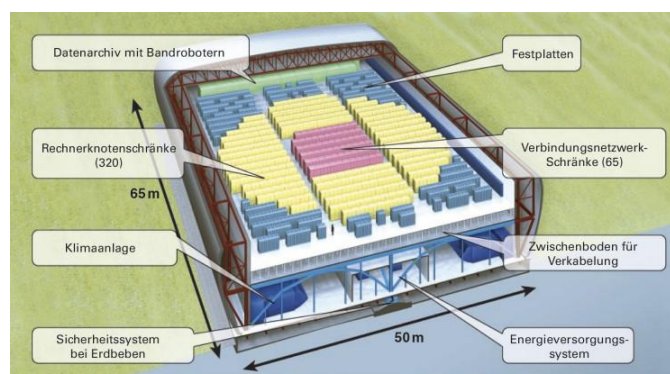
Connect several shared memory computers.
If hardware supports global address space we have a numa-machine.

Ex: Cluster of IT-servers, 2 processor nodes connected with Ethernet.



4. Parallel Vector Processors (PVP)

Connect several vector processors



Ex: Earthsimulator (2002), 5120 vector processors



Flynn's taxonomy: (characterize parallel computers)

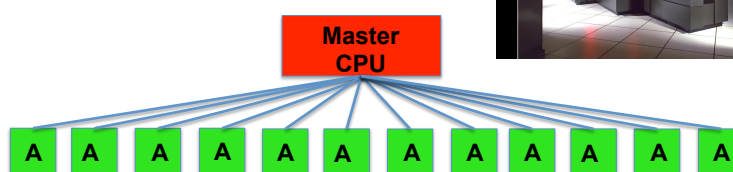
Instruction stream: single (same instr on all cpus)
multiple (diff instr on diff cpus)

Data stream: single (same data on all cpus)
multiple (diff data on diff cpus)

		Instruction stream	
		Single	Multiple
Data stream	Single	SISD "Traditional serial CPU"	MISD "Not applicable"
	Multiple	SIMD "Array processor"	MIMD "General parallel comp"



Array processor (SIMD)



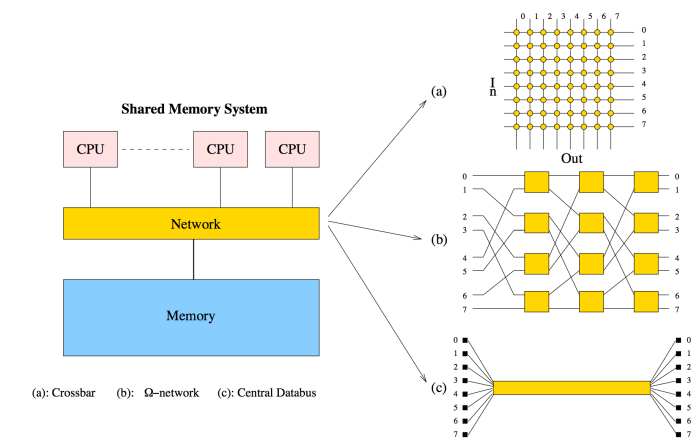
Master – Control processor, front end
A – Slave processor, ALU+Mem

The slave processors are synchronized, all doing the same operation at the same time (SI) but with different data (MD). Popular ~20y ago, Thinking Machine CM5 with up 64k slave processors. (C.f. GPU today)



Network (interconnect)

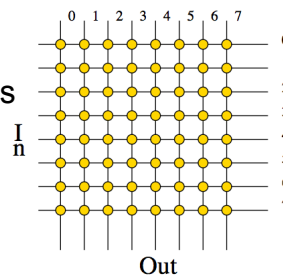
The performance of the network is often critical for the total performance of the parallel computer (depends on the application).



1. Dynamic interconnects

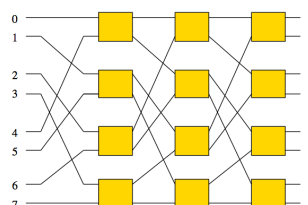
Crossbar switch

- + Multiple independent paths
- Expensive P^2 switches
- Not scalable



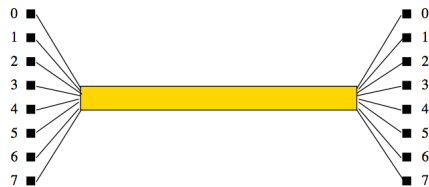
Multi stage switch

- Less paths
- + Less expensive
- + More scalable





2. Bus based network

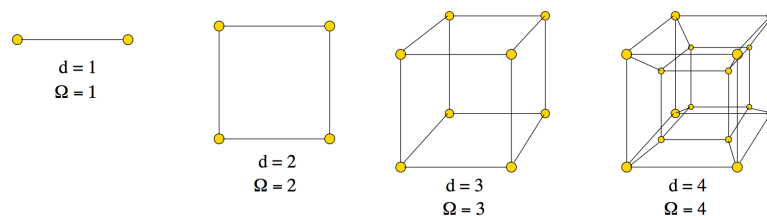


- Processors are connected with a global shared bus
- + Simple to construct
 - + Inexpensive
 - Saturates at a small number of processors (especially with snooping protocols)



3. Static networks (expensive, dedicated)

- Completely connected (all to all)
- Star connected
- Linear array
- 2D/3D Mesh or torus
- Fat tree (wider connections close to root)
- Hypercube



Example: 1,2,3, and 4 dim Hypercubes



Communication models

1. Store-and-forward

Each intermediate node waits for the entire message to arrive before it forwards it.
Inefficient but avoids blocking paths

2. Cut-through

The message is decomposed into smaller pieces and pipelined through the network

$$\Rightarrow T_c = t_s + n t_w$$

t_s – start up time, time for the first piece to reach the receiver (algorithm, switch time, distance)

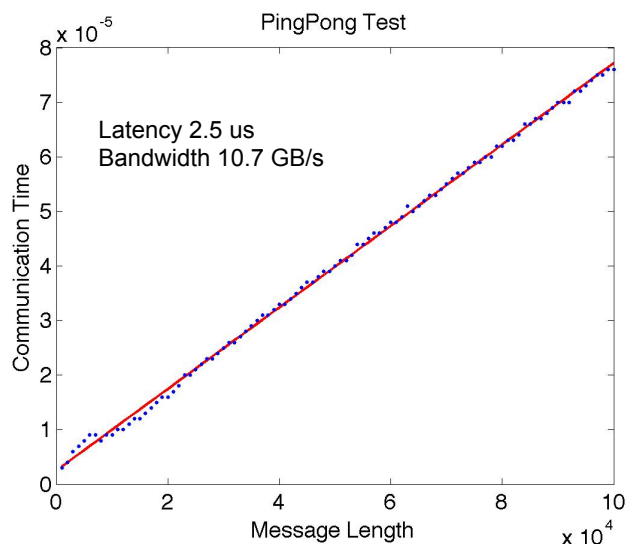
t_w – transfer rate (1/bandwidth)

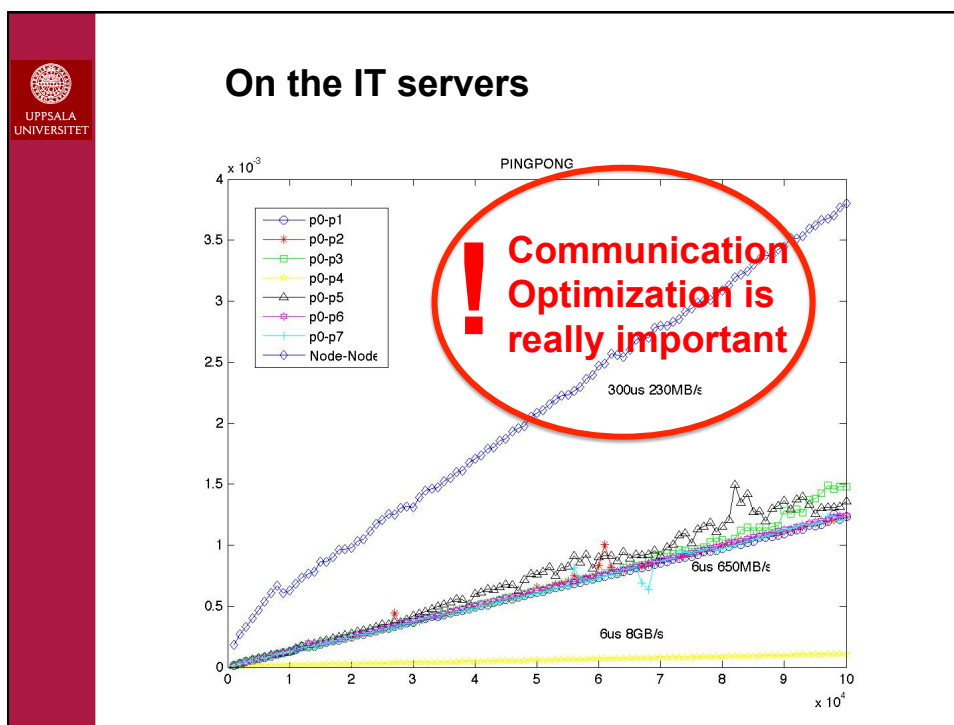
n – message length

Use “pingpong” program to measure t_s and t_w at lab 1.



On the Mac





Parallel Computers at Uppsala University

IT-servers:	<ul style="list-style-type: none"> • beurling.it.uu.se • celsius.it.uu.se • geijer.it.uu.se • fries.it.uu.se • linne.it.uu.se • polhem.it.uu.se • scheele.it.uu.se • sernander.it.uu.se • svedberg.it.uu.se • tiselius.it.uu.se
Intel Core2 Quad 10 nodes x 2 CPUs x 4 cores Gigabit Ethernet between nodes	
AMD Opteron 16 core	<ul style="list-style-type: none"> • gullviva.it.uu.se • tussilago.it.uu.se • vitsippa.it.uu.se



Kalkyl.uppmass.uu.se (2010)

- 348 nodes x 2 proc x 4 cores = 2784 cores
- Intel Xeon 5520 (Nehalem, Core i7) 2.26 GHz
- DDR Infiniband interconnect
- Peak performance 20,5 Tflop/s
- Total memory 9,5 TByte RAM



Halvan.uppmass.uu.se (2011)

- 1 node x 8 proc x 8 cores = 64 cores
- Intel Xeon 6550 2.00 GHz
- Shared memory 2048 GByte RAM




Tintin.uppmass.uu.se (2012)

- 160 nodes x 2 proc x 8 core = 2560 cores
- AMD Opteron 6220
- QDR Infiniband interconnect
- 64 GByte RAM per node (10.2 Tbyte)
- 4 GPU Nvidia M2050, each 448 cores.

Milou.uppmass.uu.se (2013)

- 208 nodes x 2 proc x 8 core = 3328 cores
- Intel Xeon E5-2660
- 4xQDR Infiniband interconnect
- 128 GByte RAM per node (26.6 Tbyte)



UPPSALA
UNIVERSITET

Guided tour at UPPMAX server room,
gather inside entrance at Ångström
Laboratory at 15.00 (and 15.20).