Computing architectures Part 2

TMA4280—Introduction to Supercomputing

NTNU, IMF

January 26. 2018

Supercomputing

What is the motivation for Supercomputing?

Solve complex problems fast and accurately:

— efforts in modelling and simulation push sciences and engineering applications forward,

— computational requirements drive the development of new hardware and software.

$\rightarrow$ architectures and software models evolved with the algorithms.

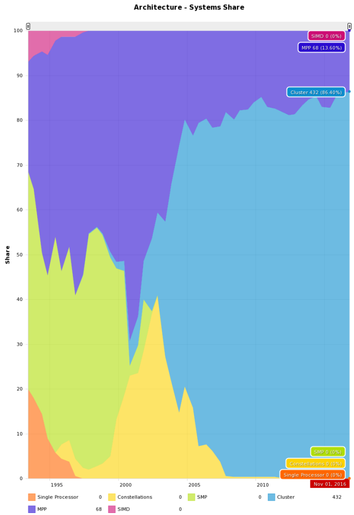
Supercomputers at NTNU: History

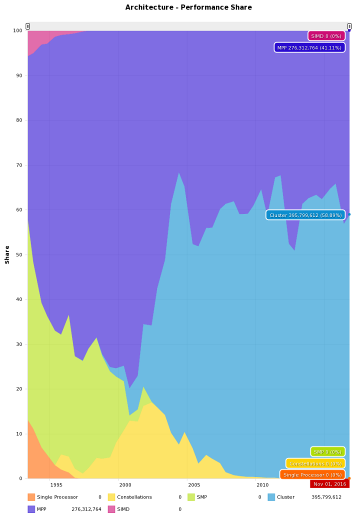
Supercomputing center established in 1983.

Year	System	Processors	Type	GFLOPS
1986–1992	Cray X-MP	2	Vector	0.5
1992–1996	Cray Y-MP	4	Vector	1.3
1995–2003	Cray J90	8	Vector	1.6
1992–1999	Intel Paragon	56	MPP	5.0
1996–2003	Cray T3E	96	MPP	58
2000–2001	SGI O2	160	ccNUMA	100
2001–2008	SGI O3	898	ccNUMA	1000
2006–2011	IBM P5+	2976	Distributed SMP	23500
2012–	SGI Altix ICE X	23040	Distributed SMP	497230

$\rightarrow$ representative of the evolution of supercomputers

Evolution: system architecture





Comparison of the evolution w.r.t system and performance share.

Supercomputers

— 70's–80's: vector processors (CRAY-1 1976);
one or a few expensive, custom-made chips.

— 80's–: MPP systems, Constellations;
many processors; standard micro-processors but possibly proprietary interconnects.

— Current trend: multicore systems;
heterogeneous computing.

$\rightarrow$ chosen solutions are a compromise between requirements and costs.

Flynn’s Taxonomy

Table: Flynn’s taxonomy:

	SD	MD
SI MD	Von Neumann (single-processor)	SIMD (vector processors) MIMD (multiprocessors)

grain-size: amount of computation performed by a parallel task.
$\rightarrow$ granularity influences the choice of a solution.

— SISD: ILP, memory hierarchy

— SIMD: DLP, fine-grained parallelism

— MIMD: TLP, coarse-grained parallelism

Multi-processor systems

Challenges:

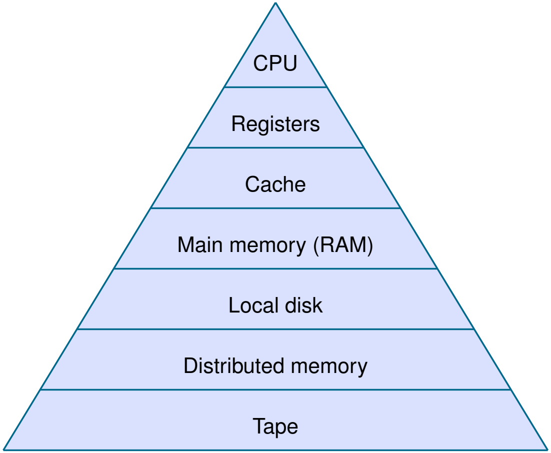
— communication between processors (memory access, programming models);

— computational methods or algorithms;

— scalability (hardware and algorithms);

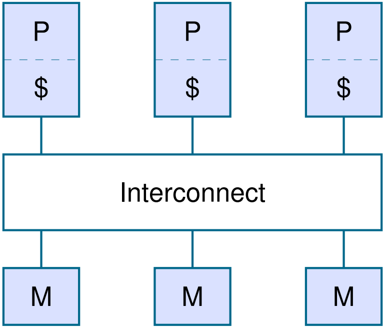
— large volumes of data (storage and visualization).

Memory hierarchy

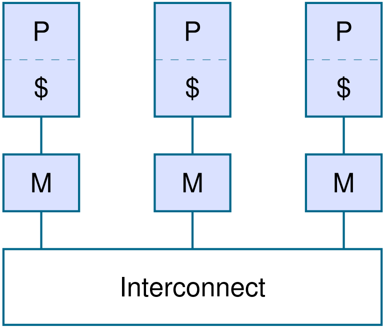


The same bottleneck applies to multiprocessors: different memory architectures and interconnect topologies are developed to overcome the limitations.

Shared memory access

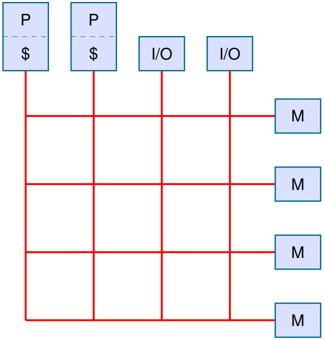


Distributed memory access



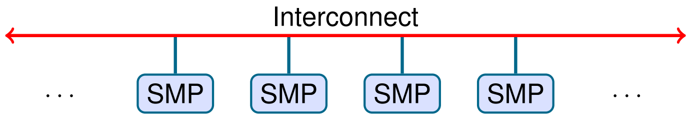
Shared memory: uniform access

This is called a *symmetric multi-processor*. Examples: bus-based, switch-based and crossbar organizations. Challenges: cache coherency and cost.



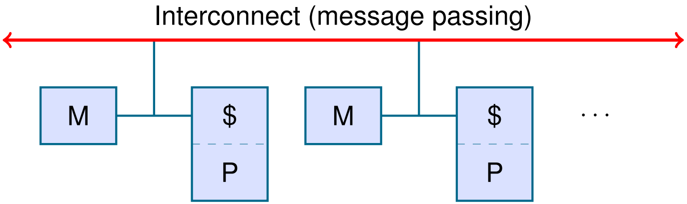
Shared memory: non-uniform access

This is called NUMA or ccNUMA (*cache-coherent non-uniform memory access*). Example: Several SMPs connected with a high-speed low-latency network. Each SMP has uniform memory access internally.



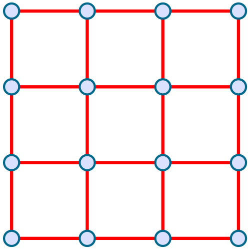
Distributed memory systems

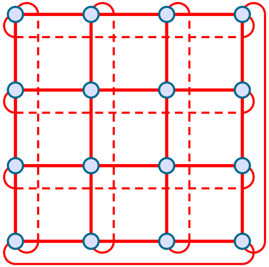
Only the local address space is available to each processor. Data from other processors are only available through explicit message-passing.



Network topology

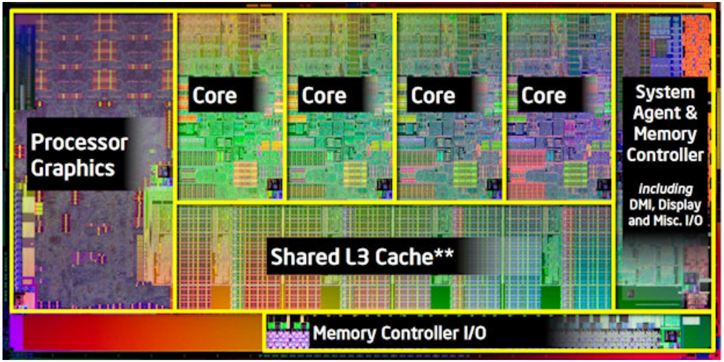
Examples: 2D mesh or toroid. Vilje is an eight-dimensional hyper-cube (!).





The current supercomputer at NTNU

Based on the Intel Sandy Bridge microprocessor, an octa-core chip (image shows the quad-core version)

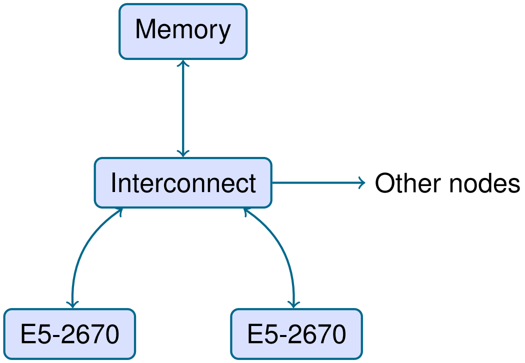


The current supercomputer at NTNU

Intel Sandy bridge E5-2670:

- An octa-core chip (8 physical processing cores)
- Caches and memory:
  - private L1 cache (32kB instruction+32kB data) 3 clocks;
  - private L2 cache (256kB) 8 clocks;
  - shared L3 cache (20MB) $\sim$ 30 clocks (could not find info);
  - main memory (32GB) $\sim$ 150 clocks (could not find info).
- FMA capable AVX unit, meaning 8 Flop per cycle, SSE 4.x.
- Simultaneous multi-threading (SMT): Intel calls this “hyperthreading”: each processor core can handle two instruction streams at the same time. Problem: Shared SIMD units.

A node: two E5-2670 chips



Key data

Vilje:

— 1440 nodes or 23040 physical cores;

— 16-core shared memory within a single node;

— distributed memory across nodes;

— 394 TB storage.

— 8.6GB/s aggregated bandwidth.

Programming models:

— shared memory programming model (OpenMP) within a node;

— message passing (MPI) across nodes;

— also possible: message passing within a single node;

— also possible: both models within the same program, hybrid.

Levels of parallelism: Single processor

Core

- — pipelining
- — superscalar execution
- — vector processing (SIMD unit)
- — branch prediction
- — caching techniques
- — multithreading
- — prefetching
- — ...

Instruction-level parallelism, Concurrency, SIMD unit

Levels of parallelism: Multi-processor

Compute node

- multiple cores on a chip
- core sharing cache memory
- affinity, locality groups
- accelerators
- …

Shared memory model (OpenMP)

Levels of parallelism: Distributed system

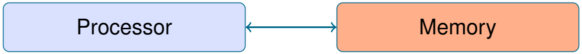
Cluster (system comprised of several compute nodes)

- network topologies
- optimized libraries
- communication patterns
- . . .

Distributed memory model (MPI)

From single processor to multiprocessors

Von Neumann model:



1. A Central Processing Unit (CPU) executes instructions corresponding to programs and data.
2. A Main Memory contains both programs and data.
3. A Bus connects the CPU and the Main Memory.

$\rightarrow$ data movement to/from the Main Memory is a crucial aspect.

The processor performance is define by an operation rate $r = 1/\tau$ (FLOPS) such that

$$T_1 = N\,\tau = \frac{N}{r}$$

is the time to execute one program with N floating-point operations on a single processor.

From single processor to multiprocessors

$$T_1(\tau) = \frac{N}{r}$$

1. T_1 the time to execute the program [s].
2. N the number of floating-poing operations [$FLOP$]
3. r the operation rate [$FLOPS$].

$r = 1/\tau$ depends on the number of instructions that the processor can execute per clock cycle.

Vector units (SIMD) like AVX2 can dramatically increase the operation rate, for instance with FMA3 (Fused-Multiply-Add).

Challenges of parallel computing

Model for parallel computations:

$$T_p = \alpha \frac{T_1}{p} + (1 - \alpha) T_1$$

with α fraction of time spent in parallel sections of the code.

The fraction α:

— $\rightarrow$ 1 for purely parallel case

— $\rightarrow$ 0 for purely serial case

Ideal speed-up when solving a **fixed** problem on *p* processors:

$$S_p = \frac{T_1}{T_p} = p$$

Linear strong scaling: $\alpha = 1$, no *serialization*.

Performance scaling

Limited parallelism in programs:

Amdahl's law: speed-up on p processor w.r.t serial

$$\mathcal{S}_p = \frac{T_1}{T_p} = \frac{p}{\alpha + (1-\alpha)p}$$

with α fraction of time spent in parallel sections of the code.

The fraction α determines the maximum speed-up possible:

— $\alpha = 0.9$, $S_p \leq 10$

— $\alpha = 0.99$, $S_p \leq 100$

The scalability of the algorithm and implementation will be limited by the tiniest serial code section. For PDEs, algorithms consist mostly of loops so scalability can be excellent.

Performance scaling

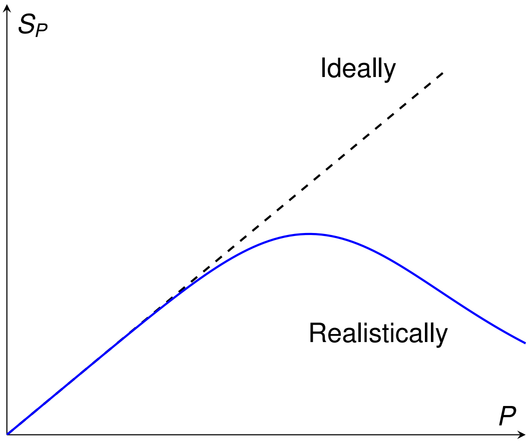
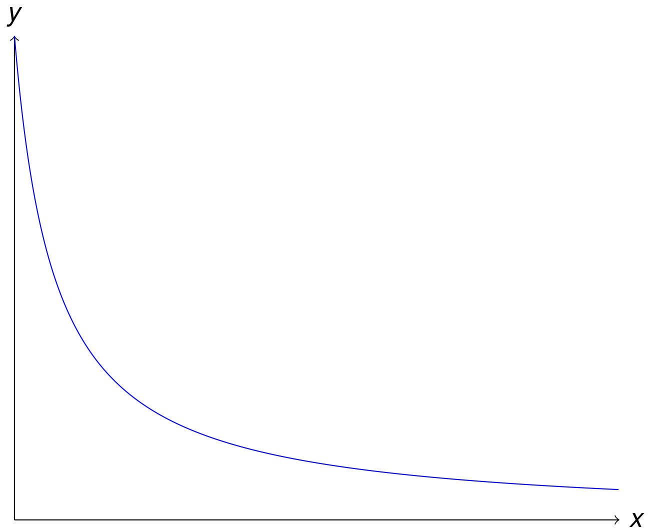


Figure: Ideal speedup ($S_P = P$) and realistic speedup: interpretation in terms of balance between local work and communication overhead.

$\rightarrow$ strong scaling should be interpreted carefully!
$\rightarrow$ weak scaling should also be considered.

Challenges of parallel computing

Example Amdhal function for a node on Vilje: 16 processors



$\rightarrow$ steep decrease in efficiency

Performance scaling

Amdahl's law: relative speed-up

$$S_p = \frac{1}{\frac{\alpha}{\bar{S}_p} + (1 - \alpha)}$$

with α fraction of time spent in **parallel** sections of the code.

Example: How to reach 80 percent of theoretical with 100 processors?

$\rightarrow$ less then 0.25 should be spent in serial.

$\rightarrow$ crucial to (re-)think algorithms for parallelism.

Communication cost

— data movement between the different level of memory hierarchy
— connections (bus, network) may exhibit hight latencies

Latencies can be addressed by:

1. architectures: caching data.
2. software: rewriting data structures to improve locality.

The network communication is of order of a microsecond while cache latency is of order of a nanosecond:

$$(1-\alpha)T_1 \approx T_c$$

with T_c a communication cost.

Linear model for communication: $Tc_(N) = T_s + N\gamma$ with T_s an initialization latency and γ a transmission speed (inverse bandwidth).

Relative performance

Relative speed-up analysis between two code versions

$$S_p = \frac{T_1}{T_p} = \frac{p}{1 + p\dfrac{T_c}{T_1}}$$

with T_c the communication time, if $T_c = 0$ ideal speed-up $S_p = p$.

Consider two versions of the same code with execution times:

$$T_1^* << T_1$$

given for an optimized implementation $*$.

Comparing S_p and S_p^*

$$\frac{S_p^*}{S_p} = \frac{1 + p\frac{T_c}{T_1}}{1 + p\frac{T_c}{T_1^*}}$$

Relative performance

Comparing S_p and S_p^*

$$\frac{S_p^*}{S_p} = \frac{1 + p\frac{T_c}{T_1}}{1 + p\frac{T_c}{T_1^*}}$$

$$\frac{1}{T_1^*} > \frac{1}{T_1} \Rightarrow S_p^* < S_p$$

The maximum speed-up for the optimized (faster) version is lower than the slow code: communication overhead is hidden.

But, the optimized code remains faster!

$$T_p^* = \frac{T_1^*}{p} + T_c$$

since $T_1^* << T_1$

Memory effect

Assume that the performance relation reads:

$$T_p = \frac{T_1}{p} + T_c$$

As before the first part consist of the cost of floating point operations and the second the cost of communication.

Consider the operation rate when all the data can be stored in the cache: no memory latency

$$T_1(\tau) = N\tau$$

for an optimal operation rate: τ is the average time spent for one floating-point operation.

Consider the operation rate when some data needs to be fetched from the main memory

$$T_1(\bar{\tau}) = N\bar{\tau}$$

with $\bar{\tau} = \tau + \mu$, μ the average penalty for fetch data from the main memory.

Memory effect

Assume that the performance relation reads:

$$T_p(\tau) = \frac{T_1(\tau)}{p} + T_c = \frac{N\tau}{p} + T_c$$

$$S_p = \frac{T_1(\bar{\tau})}{T_p(\tau)}$$

$$S_p = \frac{p}{\frac{\tau}{\bar{\tau}} + p\frac{T_c}{T_1(\bar{\tau})}}$$

If all the data fits in the cache then $\tau = \bar{\tau}$, otherwise the speed can be super-linear: the explanation is that for big problems for low number of processes any access to the main memory will incur a penalty.