NGSSC
FORSKARSKOLA I TEKNISK-VETENSKAPLIGA BERÄKNINGAR
NATIONAL GRADUATE SCHOOL IN SCIENTIFIC COMPUTING

NGSSC, Computational Methods for Statistics with Applications

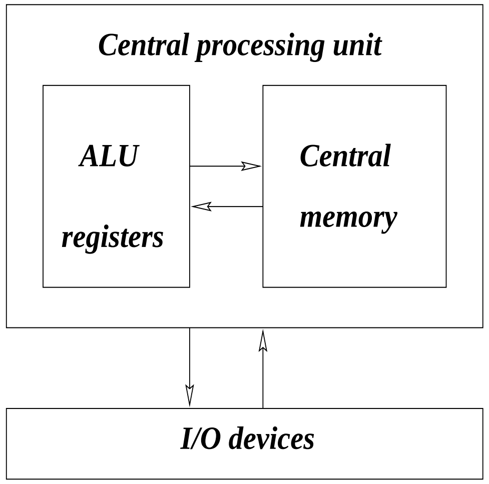
Basic parallel computing issues

Uppsala, September, 2011

- Parallel architectures
- Parallel programming paradigms
- Parallel performance measures and models - time, speedup, scalability
- Computational complexity of algorithms.
  Examples of optimal and nonoptimal algorithms
- Parallel computations in $\mathbb{R}$

- Parallel architectures
- Parallel programming paradigms
- Parallel performance measures and models - time, speedup, scalability
- Computational complexity of algorithms.
  Examples of optimal and nonoptimal algorithms
- Parallel computations in $\mathbb{R}$

The von Neumann architecture (since 1945)



- programs and data are treated in the same way; they both reside in the computer memory;
- the instructions are executed in a consecutive manner, following the order of how they have been programmed;
- conditional branches are allowed;
- during the execution, an information exchange between the CPU and memory takes place, involving data or program instruction;
- input of programs and data, as well as output of results is done by communicating to the outer world.

`fetch` $\rightarrow$ `decode` $\rightarrow$ `execute` $\rightarrow$ `store`

The von Neumann architectural principles are subject to a unique interpretation. It has been implemented in a great variety of computers which, although very different in performance, internal organization, technological base, etc, have much in common.

- Their behaviour is well determined.
- Their performance is relatively easy to analyze and to predict.
- The software is easily ported from one computer to another.

Some terminology: Parallelism and levels of parallelism

Parallelism is the process of performing tasks concurrently.

① *arithmetic and bit level* (parallel arithmetic), for example, when different elements of a vector operation are executed in parallel or within arithmetic logic circuits;

② *instruction level*, when different phases of an instruction (like "addition") are executed in parallel;

③ *task or program level (multitasking)*, when several processors are involved in the execution of parts of a program, including parallelism between subroutines (program parts) as well as within a group of operands;

④ *job level* (sometimes referred to as *multiprogramming*) when independent processors execute different jobs in parallel or parallelism between phases of a job.

Granularity

The term *granularity* is usually used to describe the complexity and type of parallelism, inherent to a parallel system.

granularity of a parallel computer and *granularity of computations*

- fine grain parallelism; fine-grained machine;
- medium grain parallelism; medium-grained machine;
- coarse grain parallelism; coarse-grained computer system.

Taxonomy of the ”non-von” computer architectures

Flynn’s taxonomy

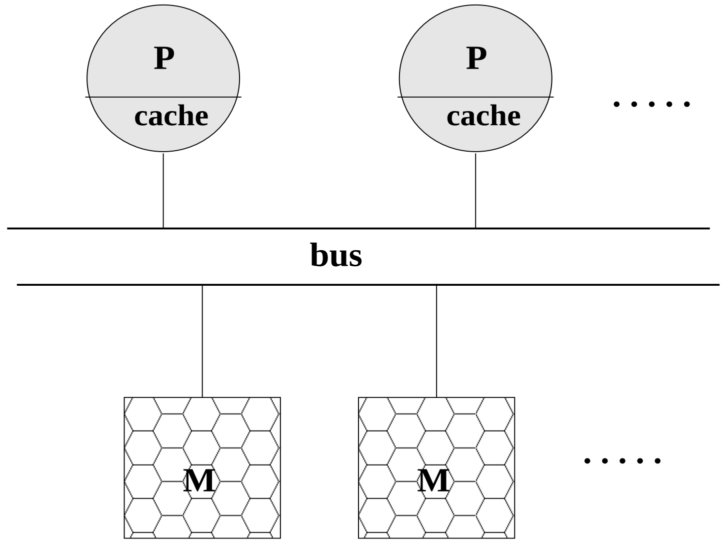
A classification from a programming point of view:

Instruction stream	Data Stream	
	SD	MD
SI	SISD	SIMD
MI	MISD	MIMD

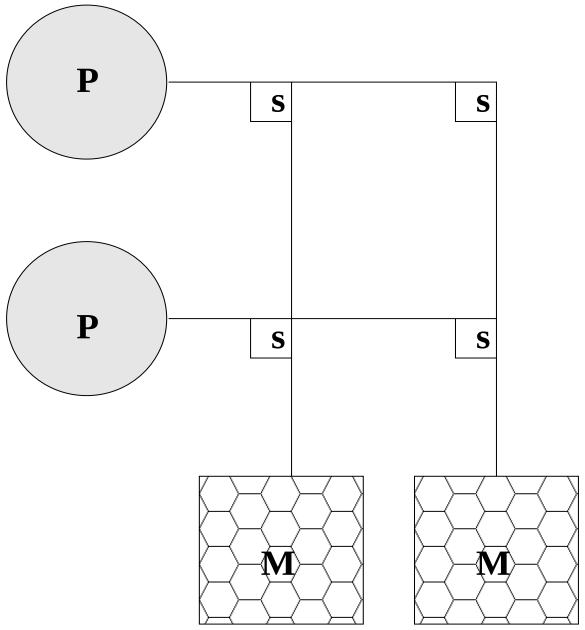
SPMD !!

Shared/distributed memory machines

Shared memory machines



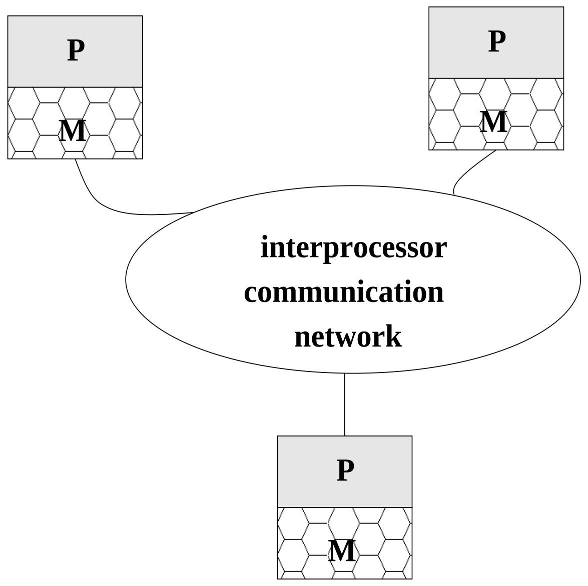
(a) bus-connected



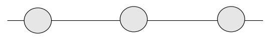
(b) crossbar-connected

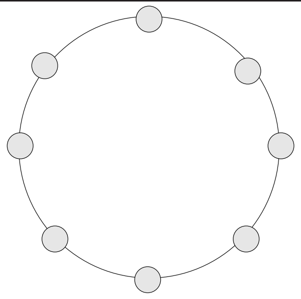
Taxonomy of the "non-von" computer architectures

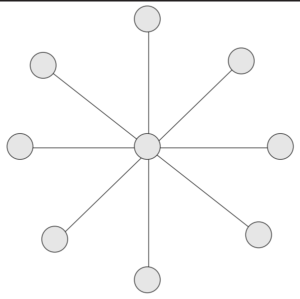
Distributed memory machines

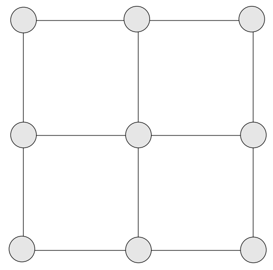


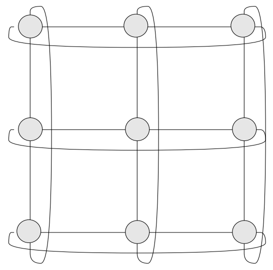
Interconnection network topologies

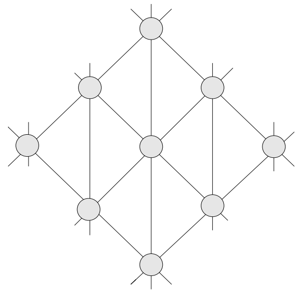

(a) linear array

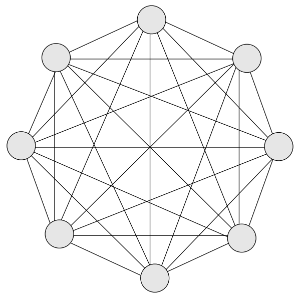

(b) ring

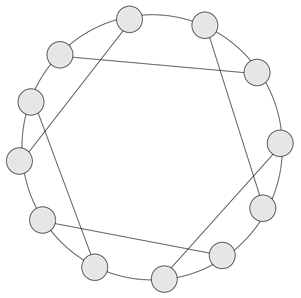

(c) star

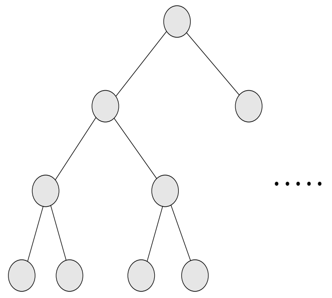

(d) 2D mesh

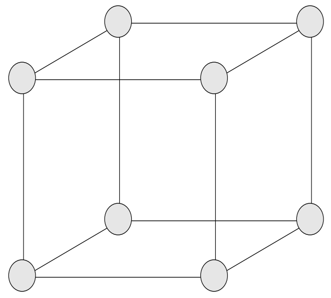

(e) 2D toroidal mesh

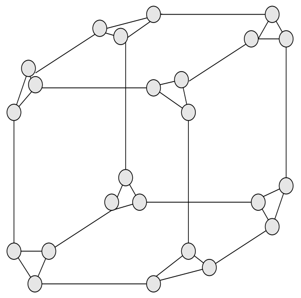

(f) systolic array


(g) completely connected


(h) chordal ring

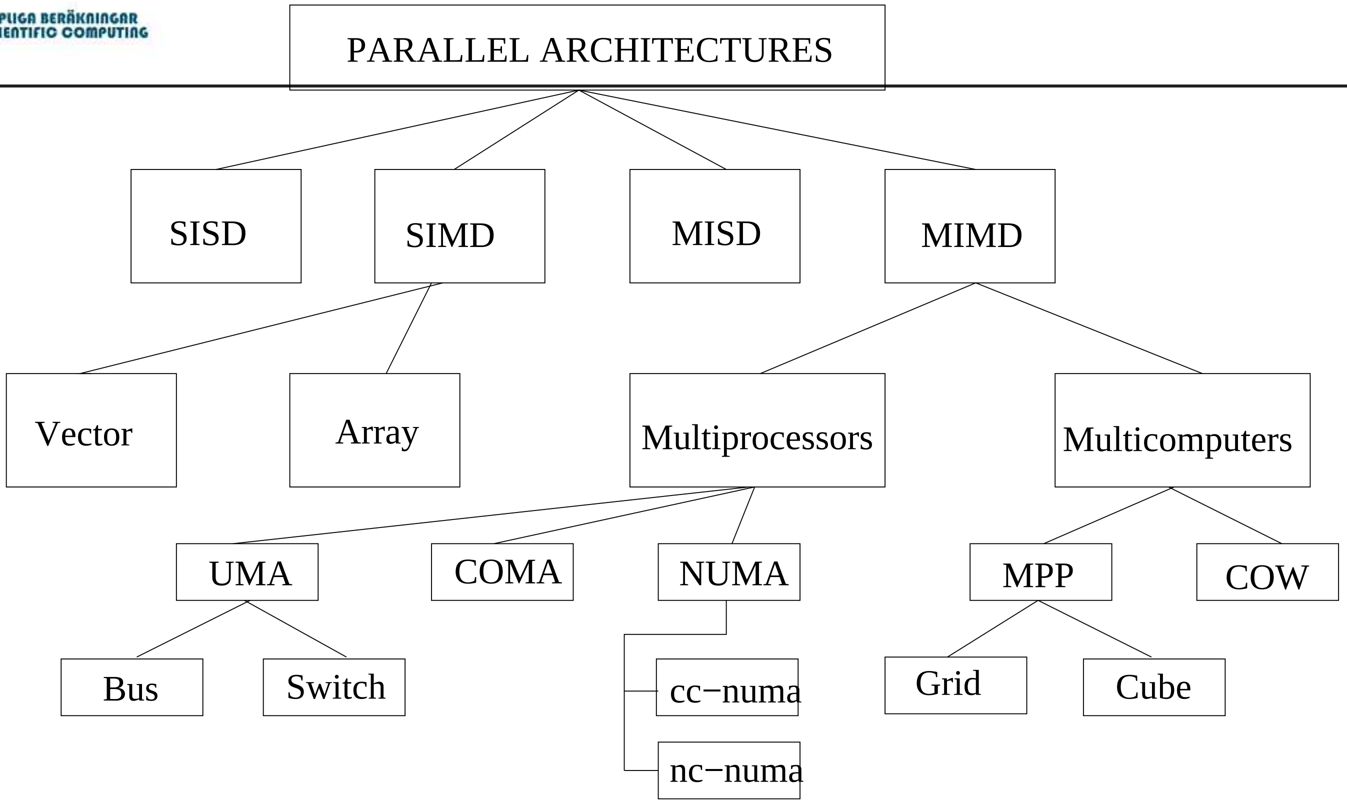

(i) binary tree


(j) 3D cube



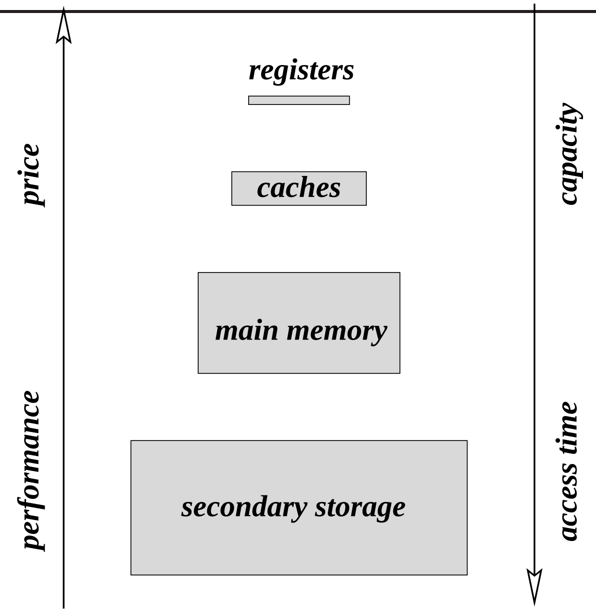
3D clustered cube

- Now each node is a multiprocessing (multicore) unit, enabling additional parallelism
- The memory is hierarchical and inhomogeneous
- CPU and GPU



UMA	Uniform Memory Access
NUMA	Nonuniform Memory Access
COMA	Cache−only Memory Access

MPP	Massively Parallel Processor
COW	Cluster of Workstations
CC−NUMA	Cache Coherent NUMA
NC−NUMA	No Cache NUMA



The memory, closest to the processor registers, is known as *cache*. It was introduced in 1968 by IBM for the IBM System 360, Model 85. Caches are intended to contain the most recently used blocks of the main memory following the principle "The more frequently data are addressed, the faster the access should be".

- Parallel architectures
- Parallel performance measures and models - time, speedup, scalability
- Computational complexity of algorithms.
- Parallel programming paradigms
- Parallel computations in $\mathbb{R}$

NGSSC
FORSKARSKOLA I TEKNISK-VETENSKAPLIGA BERÄKNINGAR
NATIONAL GRADUATE SCHOOL IN SCIENTIFIC COMPUTING

Parallel performance

We need to solve something in parallel, and as fast as possible!
Several questions arise:

• There is more than one algorithm (method) which does the job. Which one to choose?
• Can we in advance (a priori) predict the performance?
• How much does the a priori estimate depend on the computer platform? On the implementation? On the compiler? On the MPI/OpenMP implementation/Cache discipline/...?
• Can we do a posteriori analysis of the observed performance? How?
• Compare what others have done - always a good idea, but how to do this?
• We have to write a paper. How to present the parallel results? Why take up this issue? Did we do a good job?

Basic terminology

- parallel machine (homogeneous), number of PE p, size of the problem N, some algorithm A
- computational complexity $W(A,p)$, $W(A,1)$
- clock cycle
- execution time
  serial: $T(A,1) = t_c W(A)$
  parallel: $T(A,p) = T_s(A) + \frac{T_p(A)}{p} + T_c(A,p)$
- FLOPS rate (peak performance: *theoretical* vs *sustained*)

Clock cycle:
general characteristic of the speed of the processing unit.
The execution of instructions is done in quantums (unit time length) called a clock cycle:

$$\tau(s) = \frac{1}{fr} = \frac{1}{\text{frequency (Hz)}}$$

Theoretical peak performance:

$$f = \frac{\#instructions\ per\ cycle}{\tau}$$

mega-,	giga-,	tera-,	peta-,	exa-	scale performance (flops)
10^6	10^9	10^{12}	10^{15}	10^{18}	

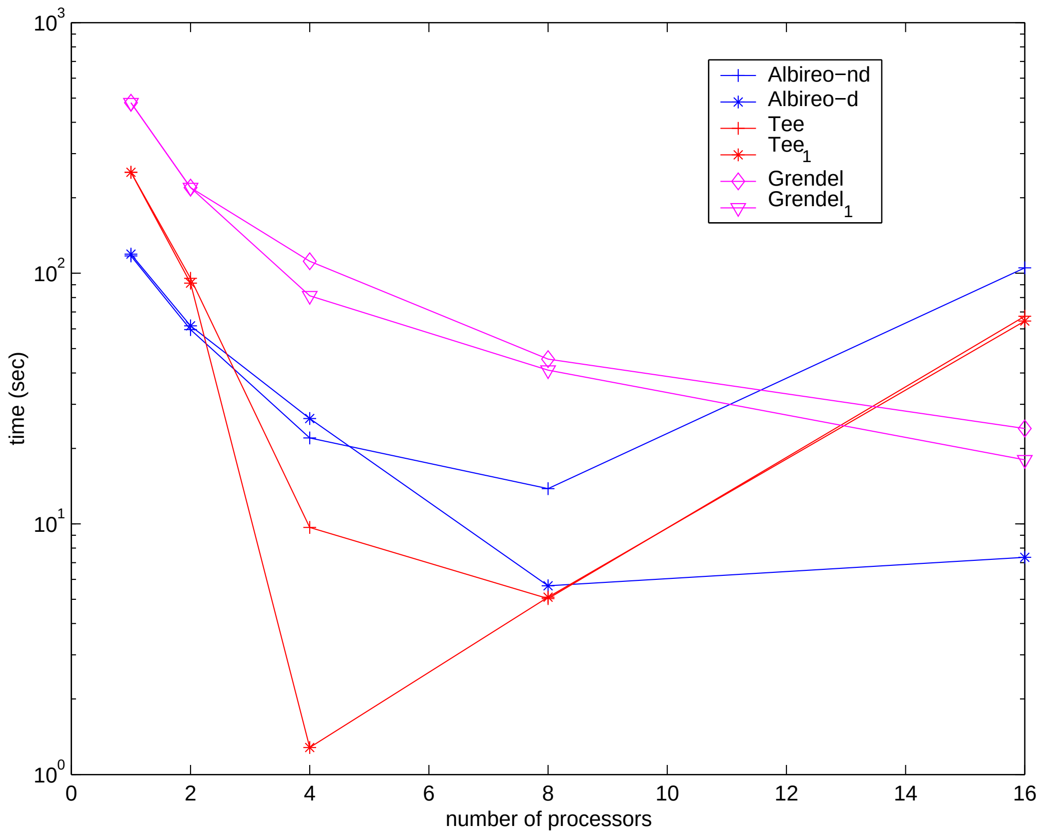
Parallel performance

Are the classical approaches relevant on the new computer architectures?

Computational and communication complexity

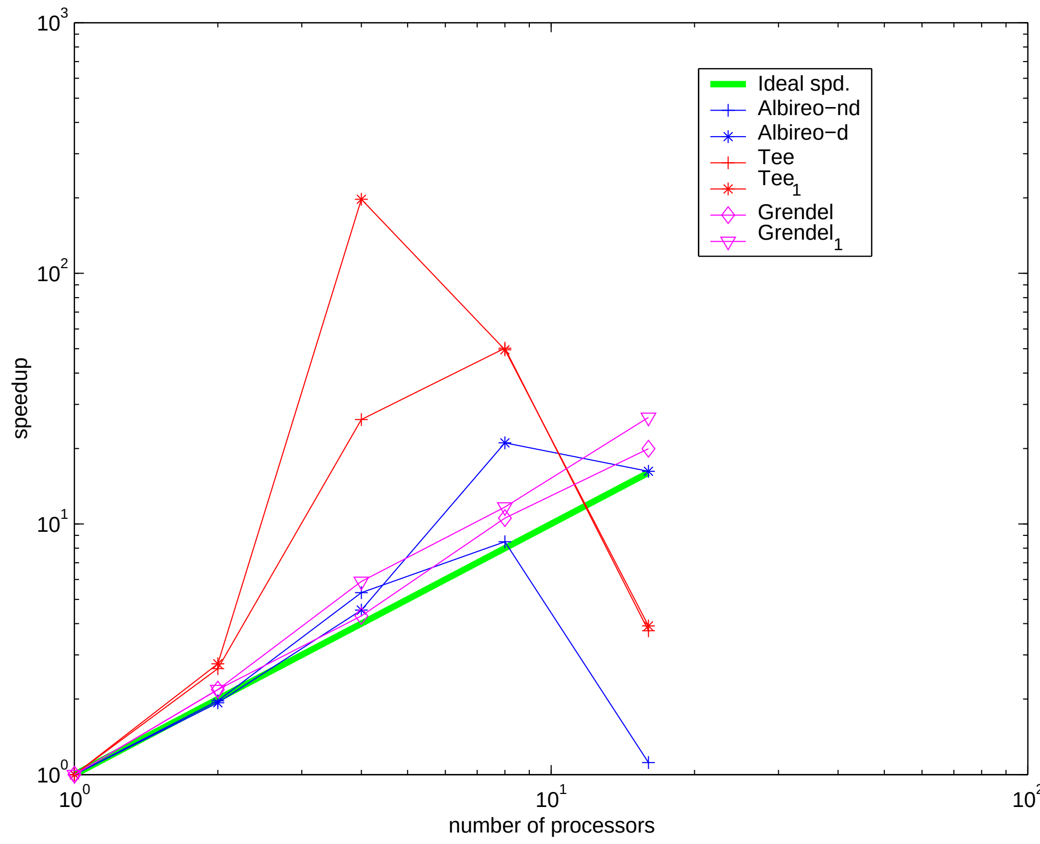
the classical approach

Parallel performance



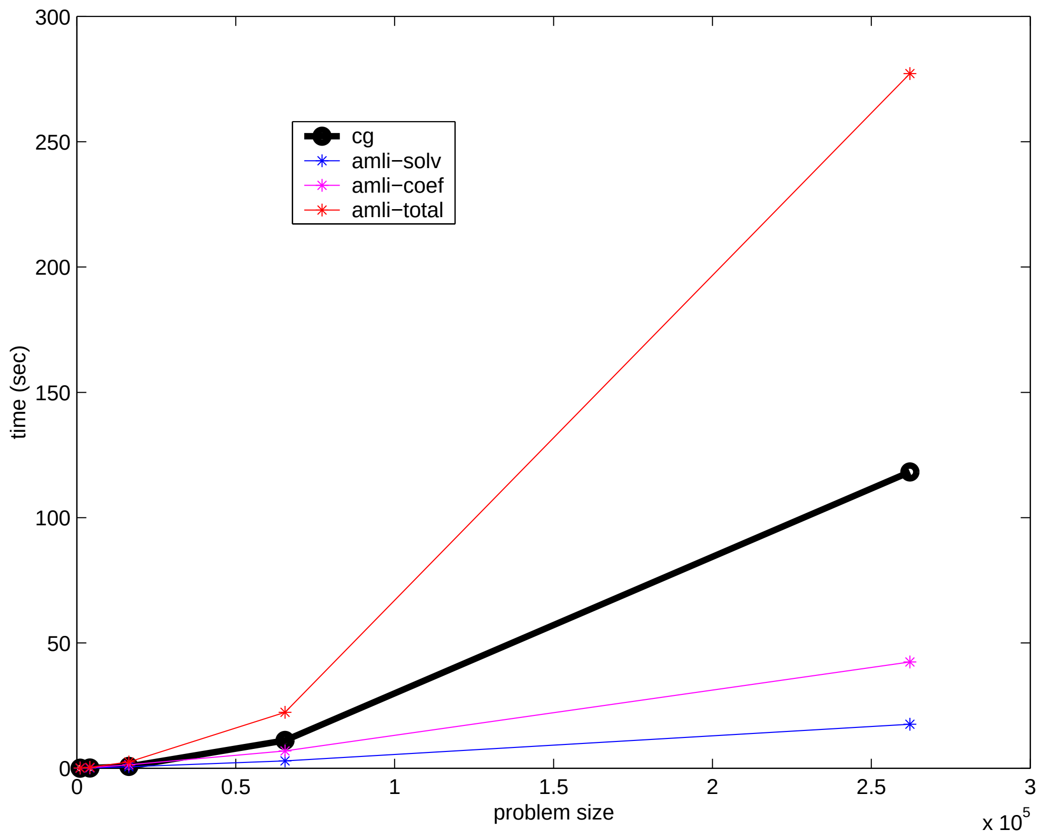
One algorithm on three parallel computers: execution times

Parallel performance



One algorithm on three parallel computers: speedup curves

Parallel performance



Two algorithms on one parallel computer: execution times

Parallel performance

How to measure the parallel performance?

Parallel performance metrics

- $T(A,p)$ is the primary metric !!!
- speedup $S(A,p) = \frac{T(A,1)}{T(A,p)} \leq p$; relative, absolute
- efficiency $E(A,p) = \frac{S(A,p)}{p} \leq 1$
- redundancy $W(A,p)/W(A,1)$
- $\cdots$
- scalability
- power consumption

Parallel performance metrics

Question: Now one PE has several cores. Does the 'serial version' utilize those?

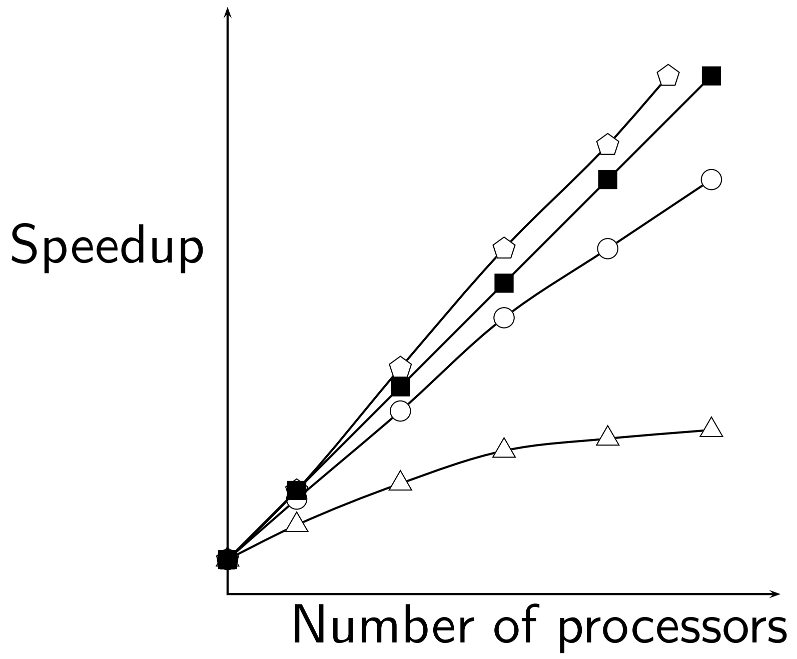
Parallel performance metrics

- $T(A,p)$
  Not much to say - we measure and observe the time.

- speedup

  – relative: $S(A,p) = \frac{T(A,1)}{T(A,p)}$
  (the same algorithm is run on one and on p PEs)
  – absolute: $\widetilde{S}(A,p) = \frac{T(A^*,1)}{T(A,p)}$
  (the performance of the parallel algorithm on p PEs is compared with the best known serial algorithm on one PE - A^*) $\cdots$ if we dare!

Parallel performance metrics



Examples of speedup curves: ideal, almost linear, sublinear, superlinear

Measuring *speedups* - pros and cons: *contra*- relative speedup is that it "hides" the possibility for $T(A, 1)$ to be very large. The relative speedup "favors slow processors and poorly-coded programs" because of the following observation.

Let the execution times on a uni- and p-processor machine, and the corresponding speedup be $T_0(A, 1)$ and $T_0(A, p)$ and $S_0 = \dfrac{T_0(A, 1)}{T_0(A, p)} > 1$.

Next, consider the same algorithm and optimize its program implementation. Then usually

$T(A, p) < T_0(A, p)$ but also $S < S_0$.

Thus, the straightforward conclusion is that WORSE PROGRAMS HAVE BETTER SPEEDUP.

A closer look:

$T(A,p) = \beta T_0(A,p)$ for some $\beta < 1$. However, $T(A,1)$ is also improved, say $T(A,1) = \alpha T_0(A,1)$ for some $\alpha < 1$.

What might very well happen is that $\alpha < \beta$. Then, of course, $\dfrac{S_0}{S} = \dfrac{\beta}{\alpha} > 1$.

When the comparison is done via the absolute speedup formula, namely

$$\frac{\widetilde{S}_0}{\widetilde{S}} = \frac{T(A^*,1)}{T_0(A,p)} \frac{T(A,p)}{T(A^*,1)} = \beta < 1.$$

In this case $T(A^*,1)$ need not even be known explicitly. Thus, the absolute speedup does provide a reliable measure of the parallel performance.

Parallel performance metrics

"As a realistic goal, when developing parallel algorithms for massively parallel computer architectures one aims at efficiency which tends to one with both increasing problem size and number of processors."

Massively parallel?

Scalability

* ***scalability of a parallel machine***: The machine is scalable if it can be incrementally expanded and the interconnecting network can incorporate more and more processors without degrading the communication speed.

* ***scalability of an algorithm***: If, generally speaking, it can use all the processors of a scalable multicomputer effectively, minimizing idleness due to load imbalance and communication overhead.

* ***scalability of a machine-algorithm pair***

How to define scalability?

Definition 1: *A parallel system is* scalable *if the performance is linearly proportional to the number of processors used.*
BUTS: impossible to achieve in practice

Definition 2: *A parallel system is scalable if the parallel execution time remains constant when the number of processors p increases linearly with the size of the problem N (time-bounded model).* **BUTS:** too much to ask for since there is communication overhead.

Definition 3: *A parallel system is scalable if the achieved average speed of the algorithm on the given machine remains constant when increasing the number of processors, provided that the problem size is increased properly with the system size.*

Scaled speedup:
Compare scalability figures when problem size **and** number of PEs are increased simultaneously in a way that the load per individual PE is kept large enough and approximately constant.

Parallel performance metrics

Presuming an algorithm is parallelizable, i.e., a significant part of it can be done concurrently, we can achieve large speed-up of the computational task using

(a) well-suited architecture;
(b) well-suited algorithms;
(c) well-suited data structures.

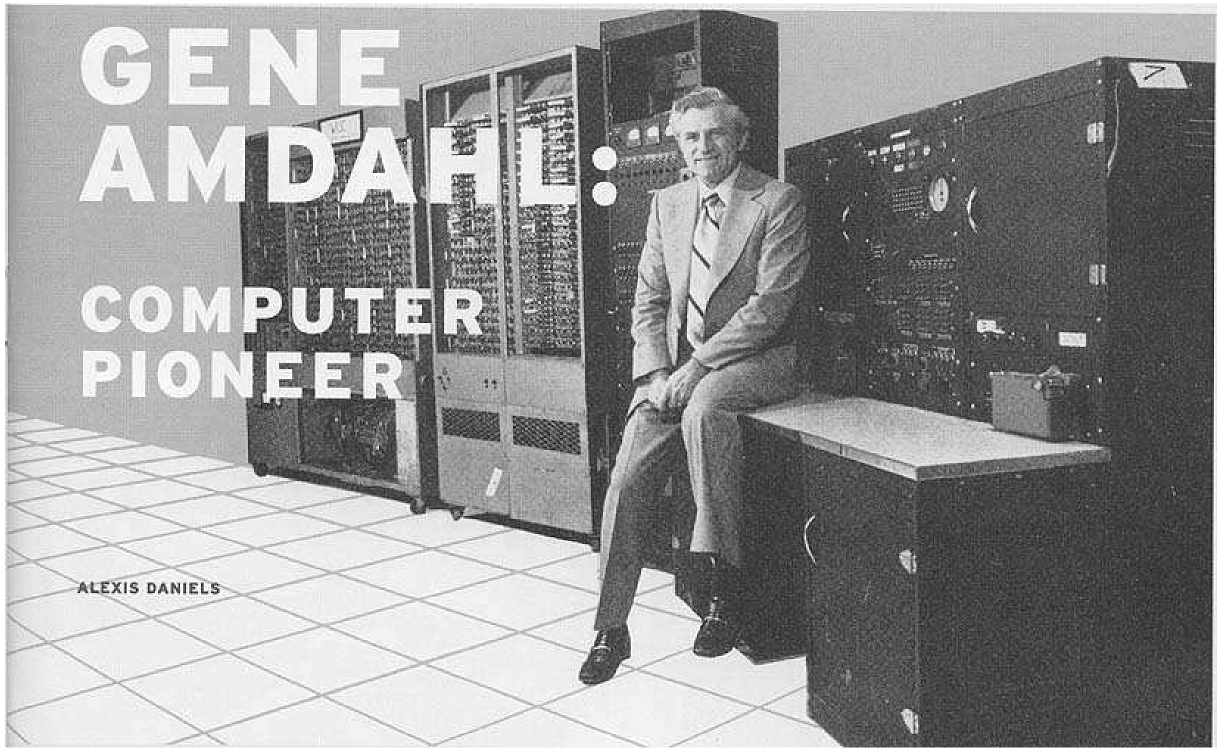
A degraded efficiency of a parallel algorithm can be due to either the computer architecture or the algorithm itself:

(i) lack of a perfect degree of parallelism in the algorithm;
(ii) idleness of computers due to synchronization and load imbalance;
(iii) of the parallel algorithm;
(iv) communication delays.

Parallel performance metrics

- Parallel performance
- Parallel performance measures
  - time
  - speedup
  - efficiency
  - scalability
- Parallel performance models
- Computational complexity of algorithms
- Optimal order algorithms
- Summary. Tendencies

Parallel performance metrics



Gene Amdahl, 1965

Parallel performance metrics

Gene Amdahl, March 13, 2008

Parallel performance metrics

Gene Amdahl:

For over a decade prophets have voiced the contention that the organization of a single computer has reached its limits and that truly significant advances can be made only by interconnection of a multiplicity of computers in such a manner as to permit cooperative solution...The nature of this overhead (in parallelism) appears to be sequential so that it is unlikely to be amendable to parallel processing techniques. Overhead alone would then place an upper limit on throughput on five to seven times the sequential processing rate, even if the housekeeping were done in a separate processor...At any point in time it is difficult to foresee how the previous bottlenecks in a sequential computer will be effectively overcome.

Parallel performance models

- The fundamental principle of computer performance; Amdahl's law (1967)
  Given: N operations, grouped into k subtasks $N_1, N_2, \cdots, N_k$, which must be done sequentially, each with rate R_i.

$$T = \sum_{i=1}^{k} t_i = \sum_{i=1}^{k} \frac{N_i}{R_i} = \sum_{i=1}^{k} \frac{f_i\, N}{R_i}; \quad \overline{R} = \frac{T}{N} N / \sum (f_i N / R_i) = \frac{1}{\sum_{i=1}^{k} f_i / R_i}$$

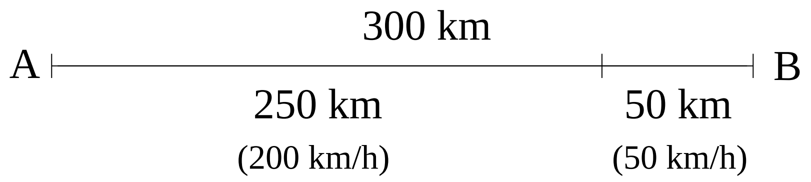
Hence, the average rate $\overline{R}(= N/R)$ for the whole task is the weighted harmonic mean of $R_1, R_2, \ldots, R_k$.
For the special case of only two subtasks - f_p (parallel) and $1 - f_p$ - serial, then

$$\overline{R}(f_p) = \frac{1}{\frac{f_p}{R_p} + \frac{1-f_p}{R_s}} \quad \text{and} \quad S = \frac{p}{f_p + (1 - f_p)p} \leq \frac{1}{1 - f_p}.$$

Parallel performance models

$$\boxed{\text{Thus, the speedup is bounded from above by the inverse of the serial fraction.}}$$

Example:



$$V = \frac{1}{\frac{5}{6}200 + \frac{1}{6}50} = 133.3\ km/h$$

If we drive $125\ km/h$ on the highway, then the total time would increase with only 15%.
So, why bother to drive fast on the highway?!

- Gustafson-Barsis law (1988):

Perhaps, the first breakthrough of the Amdahl's model is the result achieved by the 1988 Gordon Bell's prize winners - a group from Sandia Laboratories.

On a 1024 processor nCUBE/10 and with f_p computed to be in the range of $(0.992, 0.996)$ they encountered a speedup of 1000 while the Amdahl's law prediction was only of the order of 200 $(S = 1024/(0.996 + 0.004 * 1024) \approx 201)$.

$$\begin{aligned} T(A,1) &= (1-f_p)+f_p p \\ T(A,p) &= (1-f_p)+f_p = 1 \quad \text{properly scaled problem} \\ S &= T(A,1) = p-(p-1)(1-f_p) \end{aligned}$$

What has happened to the computer hardware since 1993?

Top 500, June 1993

1	Los Alamos Nat.Lab., CM-5/1024 Fat tree SuperSPARC I 32 MHz (0.128 GFlops) Hypercube, tree
2	Minnesota Supercomputer Center CM-5/544 Fat tree
3	NCSA United States CM-5/512 Fat tree
4	National Security Agency CM-5/512 Fat tree
5	NEC Japan SX-3/44R NEC NEC 400 MHz (6.4 GFlops) Multi-stage crossbar

Top 500, November 1998

1	Sandia National Laboratories ASCI Red, Intel IA-32 Pentium Pro 200 MHz (0.2 GFlops)
2	US Government, T3E1200 Cray Inc.Alpha EV56 598 MHz (1.2 GFlops) 3D torus, separate networks
3	US Government, T3E900 Cray Inc.
4	Los Alamos National Laboratory, ASCI Blue Mountain SGI MIPS R10000 250 MHz (0.5 GFlops) Interconnect HIPPI (HIgh Performance Parallel Interface)
5	United Kingdom Meteorological Office United Kingdom T3E900 Cray Inc.

Top 500, November 1998

Computer no 1:
The original ASCI Red was the first computer to rate above 1 teraFLOPS on the MP-Linpack benchmark (1996), with a peak of 1.34 TFLOPS as noted in Top500 Supercomputer sites. After being upgraded with Pentium II Overdrive processors, the computer demonstrated sustained MP-Linpack benchmarks above 2 teraFLOPS.

Different partitions of the machine used different operating systems. To the programmer, it appeared as a normal Unix machine, running "Teraflops OS", Intel's distributed OSF/1 AD-based system.

A portion of ASCI Red is in the permanent collection of The Computer History Museum in Mountain View, California.

Top 500, November 2003

1 The Earth Simulator Center Japan Earth-Simulator NEC NEC 1000 MHz (8 GFlops), Multi-stage crossbar
2 Los Alamos Nat.Lab., ASCI Q - AlphaServer SC45, 1.25 GHz HP
3 Virginia Tech X - 1100 Dual 2.0 GHz Apple G5/Mellanox Infiniband 4X/Cisco GigE Self-made
4 NCSA Tungsten - PowerEdge 1750, P4 Xeon 3.06 GHz, Myrinet Dell
5 Pacific Northwest National Laboratory Mpp2 - Cluster Platform 6000 rx2600 Itanium2 1.5 GHz, Quadrics HP
. . .
8 Lawrence Livermore National Laboratory ASCI White, SP Power3 375 MHz IBM, SP Switch
(clusters - faster)

Top 500, November 2003

Computer no 1:
The ES: a highly parallel vector supercomputer system of the distributed-memory type.
Consisted of 640 processor nodes (PNs) connected by 640x640 single-stage crossbar switches.
Each PN is a system with a shared memory, consisting of 8 vector-type arithmetic processors (APs), a 16-GB main memory system (MS), a remote access control unit (RCU), and an I/O processor.
The peak performance of each AP is 8Gflops.
The ES as a whole consists of 5120 APs with 10 TB of main memory and the theoretical performance of 40Tflop.

Top 500, November 2006

1	DOE/NNSA/LLNL BlueGene/L - eServer Blue Gene Solution IBM PowerPC 440 700 MHz (2.8 GFlops), 32768 GB
2	NNSA/Sandia Nat.Lab., Red Storm - Sandia/ Cray Red Storm, Opteron 2.4 GHz dual core Cray Inc.
3	IBM Thomas J. Watson Research Center BGW - eServer Blue Gene Solution IBM
4	DOE/NNSA/LLNL ASCI Purple - eServer pSeries p5 575 1.9 GHz IBM
5	Barcelona Supercomputing Center MareNostrum - BladeCenter JS21 Cluster, PPC 970, 2.3 GHz, Myrinet IBM

Top 500, November 2006

Computer no 1:
The machine was scaled up from 65,536 to 106,496 nodes in five rows of racks.

Each Blue Gene/L node is attached to three parallel communications networks:
- a 3D toroidal network for peer-to-peer communication between compute nodes,
- a collective network for collective communication,
- a global interrupt network for fast barriers.

The I/O nodes, which run the Linux operating system, provide communication with the world via an Ethernet network. The I/O nodes also handle the filesystem operations on behalf of the compute nodes. Finally, a separate and private Ethernet network provides access to any node for configuration, booting and diagnostics.

Top 500, November 2010

1 National Supercomputing Center in Tianjin, China, Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C NUDT
2 DOE/SC/Oak Ridge Nat.Lab., Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz
3 National Supercomputing Centre in Shenzhen (NSCS) China Nebulae - Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU Dawning
4 GSIC Center, Tokyo Institute of Technology, Japan TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows NEC/HP
5 DOE/SC/LBNL/NERSC Hopper - Cray XE6 12-core 2.1 GHz

Top 500, November 2010

Computer no 1: Peta-flop machine, 186368 cores

* 2560 computing nodes in total

* Each computing node is equipped with 2 Intel Xeon EP CPUs (4 cores) , 1 AMD ATI Radeon 4870x2 (2GPUs, including 1600 Stream Processing Units - SPUs), and 32GB memory

Operation node:

* 512 operation nodes in total

* Each operation node is equipped with 2 Intel Xeon CPUs (4 cores) and 32GB memory

Interconnection subsystem:

* Infiniband QDR

* The point-to-point communication bandwidth is 40Gbps and the MPI latency is 1.2 ms

Provides a programming framework for hybrid architecture, which supports adaptive task partition and streaming data access.

Top 500, June 2011

1 Advanced Inst. for Computational Science, Japan, K computer, SPARC64 VIIIfx 2.0 GHz, Tofu interconnect

2 National Supercomputing Center in Tianjin, China, Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C NUDT

3 DOE/SC/Oak Ridge Nat.Lab., Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz

Top 500, June 2011

Computer no 1: Peta-flop machine
548352 cores
8 quadrillion calculations per second
SPARC64 VIIIfx 2.0GHz, Tofu interconnect

- Parallel performance
- Parallel performance measures
  - time
  - speedup
  - efficiency
  - scalability
- Computational and communication complexity of algorithms
- Examples (optimal – nonoptimal order algorithms)
- Summary. Tendencies

Does this algorithm scale or not scale?

A version of `wave`:
Solve 2D advection equation with forcing term numerically with the Leap-frog scheme
(Kalkyl, Dec 13, 2010)

Problem size	1	2	4	8	16
			One year old results		
256^2	2.71	1.37	0.72	0.73	-
512^2	21.79	11.23	5.83	5.93	-
1024^2	172.35	88.75	47.25	52.62	-
			Kalkyl		
256^2	3.26	1.49	0.77	0.42	0.33
512^2	25.99	11.45	6.78	3.17	2.49
1024^2	208.04	105.32	48.17	29.25	19.69

An algorithm which does not scale ...

Given A, $\mathbf{b}$ and an initial guess $\mathbf{x}^{(0)}$.

```
$\mathbf{g}^{(0)} = A\mathbf{x}^{(0)} - \mathbf{b}$,
$\delta_0 = (\mathbf{g}^{(0)}, \mathbf{g}^{(0)})$
$\mathbf{d}^{(0)} = -\mathbf{g}$
For $k = 0, 1, \cdots$ until convergence
(1)  $\mathbf{h} = A\mathbf{d}^{(k)}$
(2)  $\tau = \delta_0 / (\mathbf{h}, \mathbf{d}^{(k)})$
(3)  $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \tau\mathbf{d}^{(k)}$
(4)  $\mathbf{g}^{(k+1)} = \mathbf{g}^{(k)} + \tau\mathbf{h}$,
(5)  $\delta_1 = (\mathbf{g}^{(k+1)}, \mathbf{g}^{(k+1)})$
(6)  $\beta = \delta_1/\delta_0$, $\delta_0 = \delta_1$
(7)  $\mathbf{d}^{(k+1)} = -\mathbf{g}^{(k+1)} + \beta\mathbf{d}^{(k)}$
```

Convergence rate, computational cost per iteration

An algorithm which does not scale ...

Dusty-shelf Fortran-77 code `grid-oriented Conjugate Gradient method`:

Problem size	1			2		
	It	Time	Time/It	It	Time	Time/It
500	756	3.86	0.0051	1273	7.77	0.0061
1000	1474	33.66	0.0228	2447	72.72	0.0297

Problem size	4			8		
	It	Time	Time/It	It	Time	Time/It
500	1474	19.69	0.0134	2447	56.96	0.0233
1000	2873	137.03	0.0477	4737	386.58	0.0816

Numerical efficiency – parallel efficiency – time

NGSSC
FORSKARSKOLA I TEKNISK-VETENSKAPLIGA BERÄKNINGAR
NATIONAL GRADUATE SCHOOL IN SCIENTIFIC COMPUTING

An algorithm which scales ...

Grid size	Coarsest level No (total no. of levels)	Number of PEs					Time (sec)
		4	8	16	32	64	
256^2	10(16)	406.62	190.54	94.61	49.55	28.90	total
		403.49	189.06	93.86	49.18	28.71	outer
		159.75	80.09	41.63	21.97	13.20	coars.
		5.31	5.93	5.58	4.62	3.89	comm.
512^2	12(18)			632.60	304.24	154.65	total
				629.44	302.71	153.81	outer
				363.38	183.18	96.15	coars.
				14.28	12.14	10.14	comm.
1024^2	12(20)				1662.73	829.71	total
					1655.73	826.22	outer
					810.11	422.25	coars.
					29.89	22.26	comm.

Stokes problem: Performance results on the Cray T3E-600 computer
Numerical efficiency – parallel efficiency – time

Ultrascalable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom
M.. Adams, H.. Bayraktar, T.. Keaveny, P. Papadopoulos
ACM/IEEE Proceedings of SC2004: High Performance Networking and Computing, 2004

Bone mechanics, AMG, 4088 processors, the ACSI White machine (LLNL):

"We have demonstrated that a mathematically optimal algebraic multigrid method (smoothed aggregation) is computationally effective for large deformation finite element analysis of solid mechanics problems with up to 537 million degrees of freedom.
We have achieved a sustained flop rate of almost one half a Teraflop/sec on 4088 IBM Power3 processors (ASCI White).
These are the largest published analyses of unstructured elasticity problems with complex geometry that we are aware of, with an average time per linear solve of about 1 and a half minutes.

- Parallel architectures
- Parallel performance measures and models
- Computational complexity of algorithms
  Examples of optimal and nonoptimal algorithms
- Parallel programming paradigms
- Parallel computations in $\mathbb{R}$

OpenMP

OpenMP

OpenMP is an industry-wide standard for directive-based parallel programming on SMP (Symmetric MultiProcessor) systems.

OpenMP Is an *Application Program Interface (API)* enabling explicit direct multi-threaded, shared memory parallelism.

OpenMP is of three primary API components:

- compiler directives;
- runtime library routines;
- environment variables.

OpenMP

OpenMP is portable: the API is specified for C/C++ and Fortran.

OpenMP is standardized: jointly defined and endorsed by a group of major computer hardware and software vendors. Expected to become an ANSI standard later.

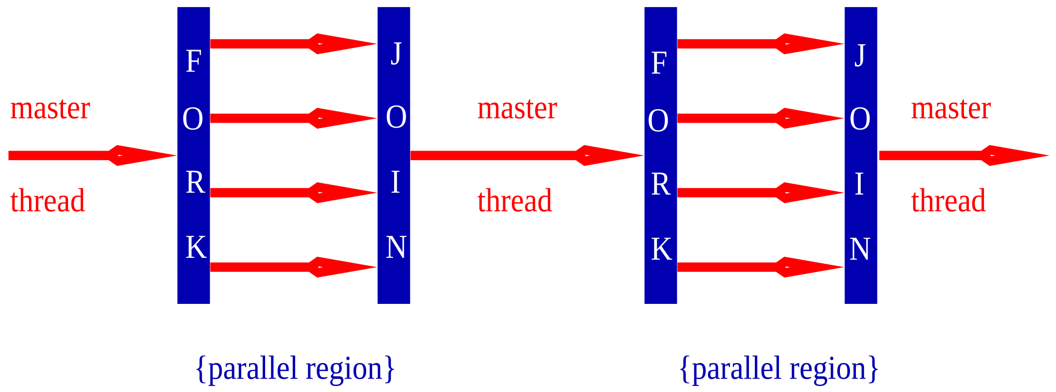
OpenMP is Not:

- meant for distributed memory parallel systems (by itself);
- necessarily implemented identically by all vendors;
- guaranteed to make the most efficient use of shared memory (currently there are no data locality constructs).

`http://www.openmp.org`

OpenMP Programming Model

Thread Based Parallelism:	A shared memory process can consist of multiple threads. OpenMP is based upon the existence of multiple threads in the shared memory programming paradigm.
Explicit Parallelism:	OpenMP is an explicit (not automatic) programming model, offering the programmer full control over parallelization.
Fork - Join Model:	OpenMP uses the fork-join model of parallel execution:



C / C++ - General Code Structure

```
#include <omp.h>
      main ()  {
      int var1, var2, var3;
      Serial code
            ...
            ...
      Beginning of parallel section. Fork a team of threads.
      Specify variable scoping

      #pragma omp parallel private(var1, var2) shared(var3)
        {
        Parallel section executed by all threads
              ...
              ...
        All threads join master thread and disband
        }
      Resume serial code
            ...
      }
```

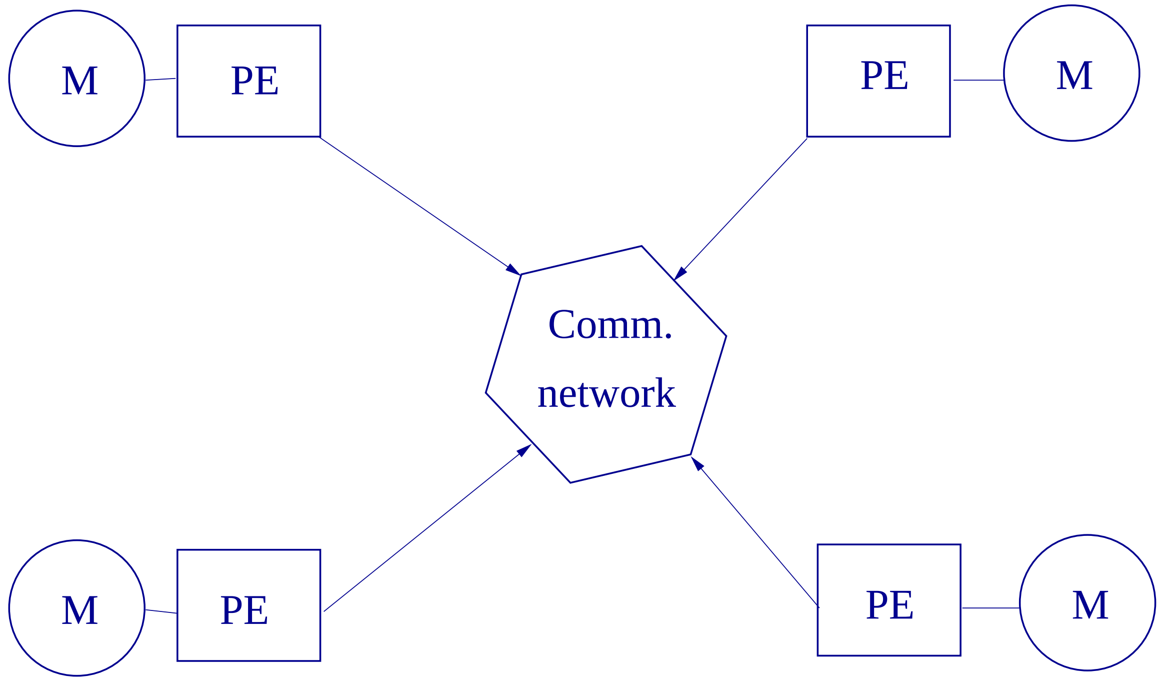
OpenMP

- Sources of parallelism - loops
- Data dependencies

```
do i=1,n
   A(i) = B(i) + C(i)
   D(i) = A(i-1) * A(i+1)
enddo
```

```
forall i = 1:n
     temp(i) = A(i+1)
end
forall i = 1:n
      A(i) = B(i)+C(i)
end
forall i = 1:n
     D(i) = A(i-1)* temp(i)
end
```

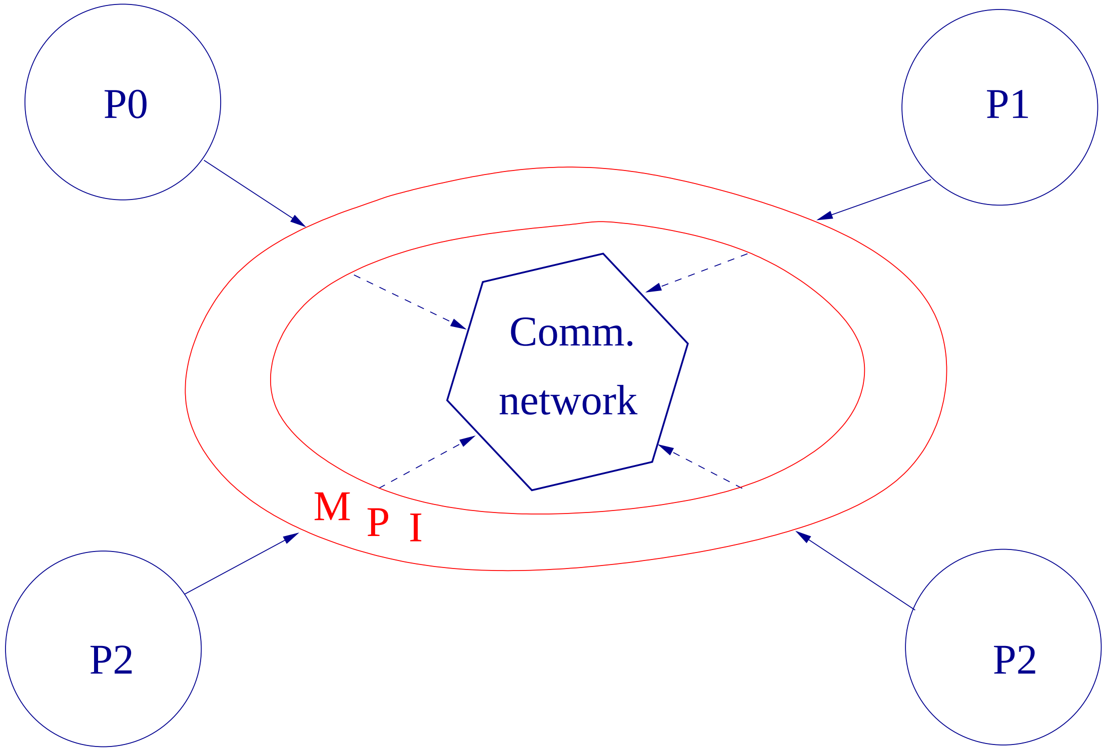
Message passing computer architecture paradigm



PE - processing elements

M - memories

Message passing programming paradigm



P - processes

Message passing programming paradigm (cont.)

- A single program is run on each processor.
- All variables are private.
- Processes communicate via special subroutine calls - MPI is just a library.
- There is no "magic" parallelism.
- The program is written in a conventional sequential language, i.e. C, C++, Fortran

Messages

- Messages are packets of data moving between processes.
- The message passing system has to be told the following information:
  - Sending process
  - Source location
  - Data type
  - Data length
  - Receiving process(es)
  - Destination location
  - Size of receive buffer(s)

Message passing system

A message passing system is similar to a mail box, phone line or a fax machine.

A process needs to be connected to a message passing interface.
Thus,

- The sender must have addresses to sent the message to
- Receiving process must:
  - participate (cf. have a mailbox it checks, a phone it answers, ...)
  - have capacity to receive (have a big enough mailbox etc)

MPI: a standard Message Passing Interface

- Defined by MPI Forum – 40 vendor and academic/user organizations
- Provides source code portability across all systems
- Allows efficient implementation.
- Provides high level functionality.
- Supports heterogeneous parallel architectures.
- An addition to MPI-1 – MPI-2.

The essence of MPI

The MPI standard contains many functions ($\geq$ 125).

The number of **basic building blocks** in MPI is small.

MPI_INIT	initialize MPI
MPI_COMM_SIZE	determine how many processes there are
MPI_COMM_RANK	find out which is my process number
MPI_SEND	send a message
MPI_RECV	receive a message
MPI_FINALIZE	terminate MPI

Now, back to $\mathbb{R}$ to see what is available for parallel computations.