- *Parallel performance*
- *Parallelization of iterative solution methods*
  - *Basic iterative schemes*

- Parallel performance
- Parallel performance measures
  – time
  – speedup
  – efficiency
  – scalability
- Parallel performance models
- Computational and communication complexity of algorithms
- Examples: nonoptimal – optimal algorithms
- Energy efficiency - models and metrics

- Parallel performance
- Parallel performance measures
  – time
  – speedup
  – efficiency
  – scalability
- Parallel performance models
- Computational and communication complexity of algorithms
- Examples: nonoptimal – optimal algorithms
- Energy efficiency - models and metrics
- Summary. Tendencies

Given a computer:

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/Pthreads 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

- computational complexity $W(A,p)$, $W(A,1)$ (number of arithmetic operations to perform)
- parallel machine (homogeneous), number of PE (threads) p, size of the problem N (degrees of freedom), some algorithm A
- 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*)
- MIPS - Million Instructions per second
- Power /Watt=Joule/second)
- Energy = Power*time (Joule)

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 (of one CPU):

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

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

More terminology: 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*

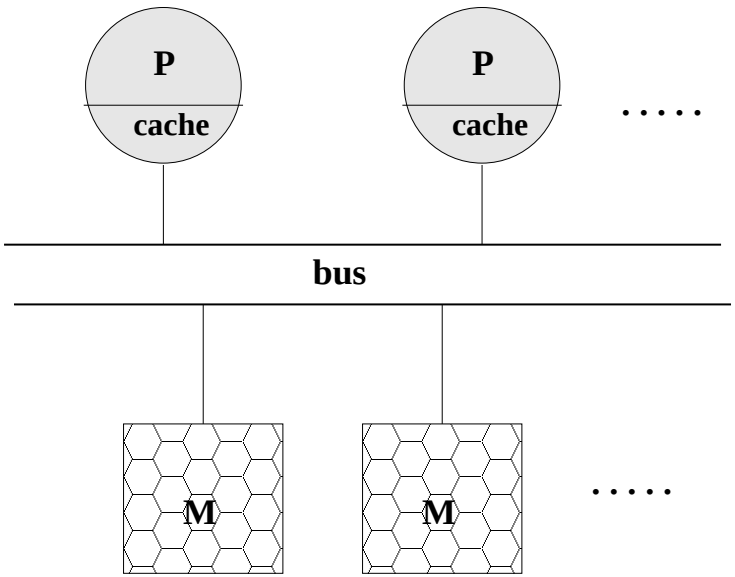
- fine grain parallelism; fine-grained machine;
- medium grain parallelism; medium-grained machine;
- coarse grain parallelism; coarse-grained computer system.

- How to measure the parallel performance?
- How to understand what we see on the performance plots?

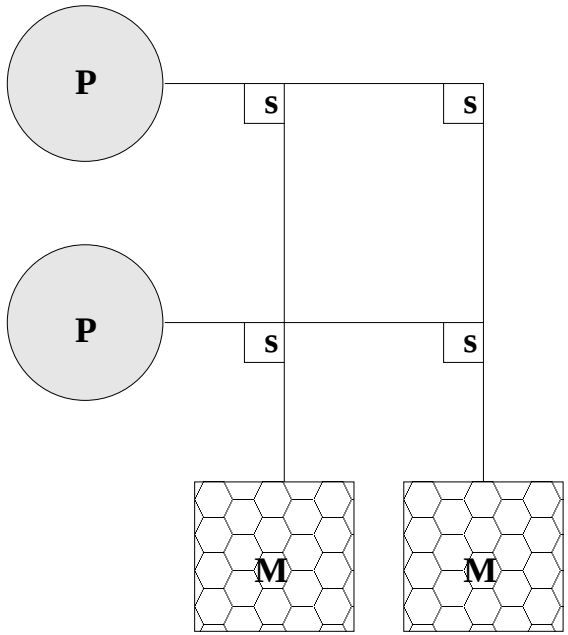
- How to measure the parallel performance?
- How to understand what we see on the performance plots?

Recall some well-known stuff
(skip type of interconnection networks»> perf.)

Recall: Shared memory machines

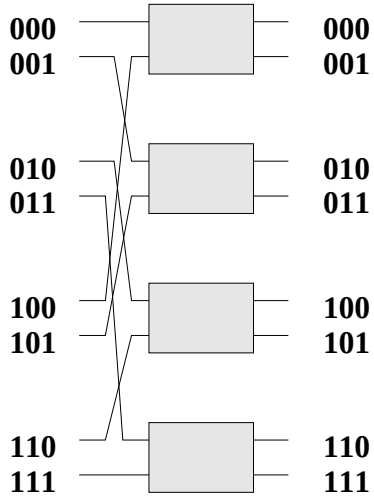


(a) bus-connected

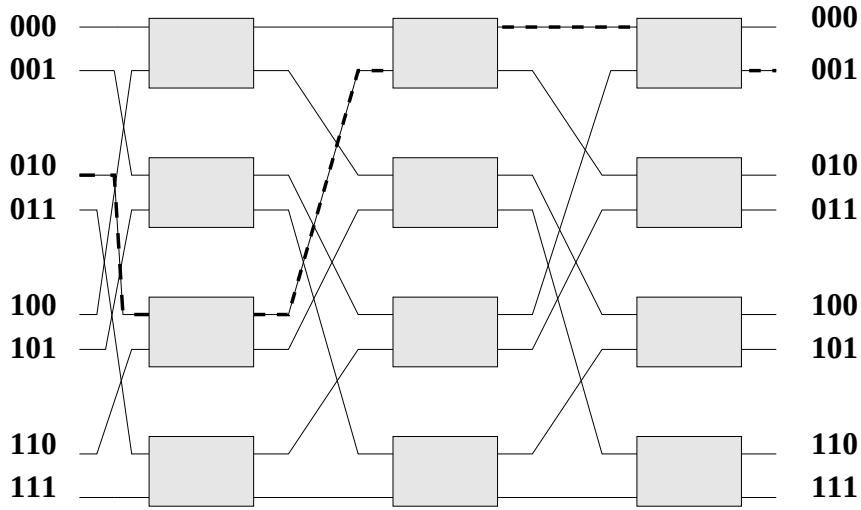


(b) crossbar-connected

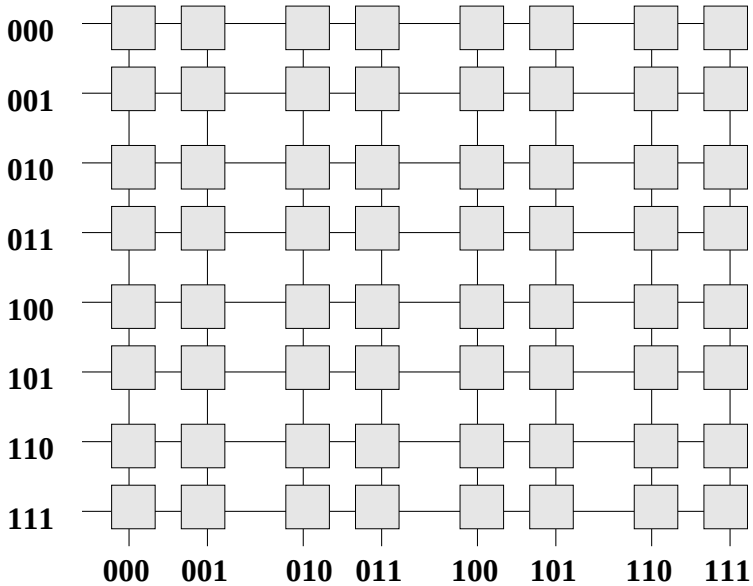
Recall: Shared memory machines



(c) Shuffle-exch.

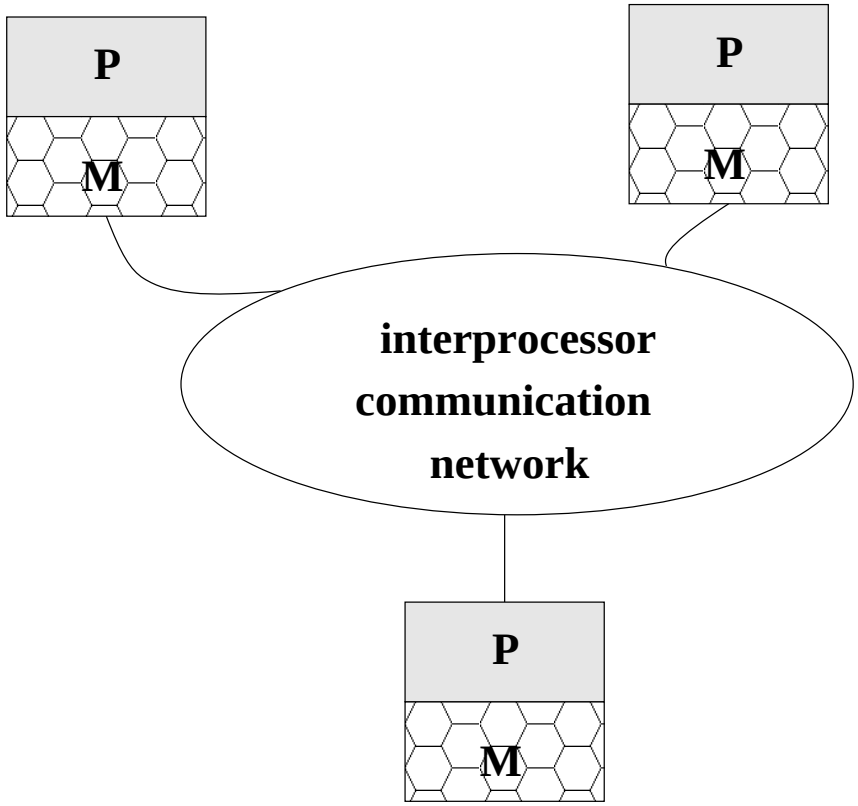


(d) Omega Network

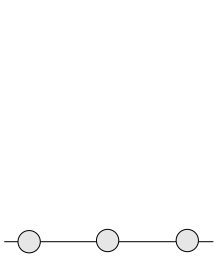


(e) Crossbar switch

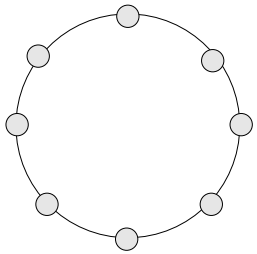
Recall: Distrib.memory: Static interconnection networks



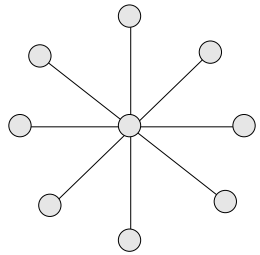
Recall: Interconnection network topologies



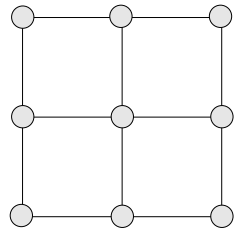
(f) linear array



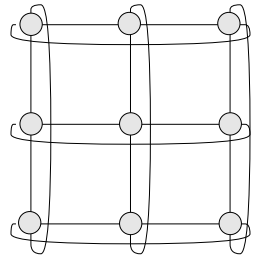
(g) ring



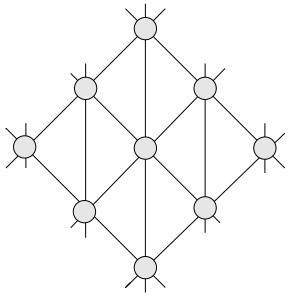
(h) star



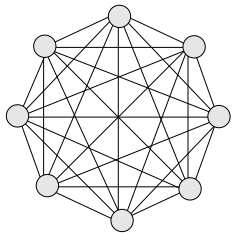
(i) 2D mesh



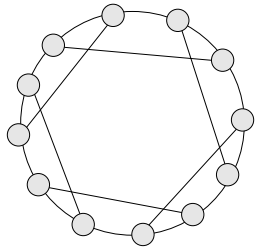
(j) 2D toroidal mesh



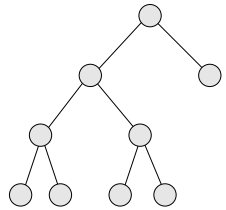
(k) systolic array



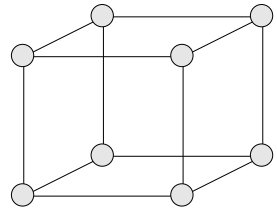
(l) completely connected



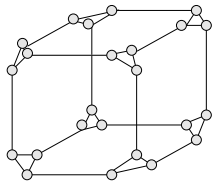
(m) chordal ring



(n) binary tree

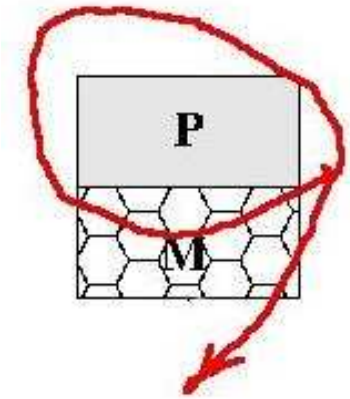


(o) 3D cube

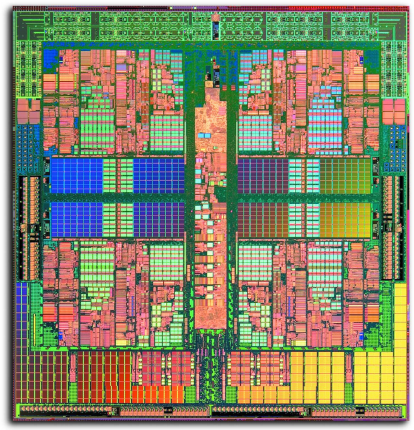


(p) 3D cube

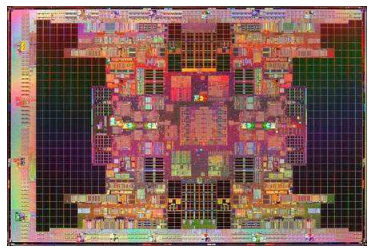
Virtual (logical) topologies



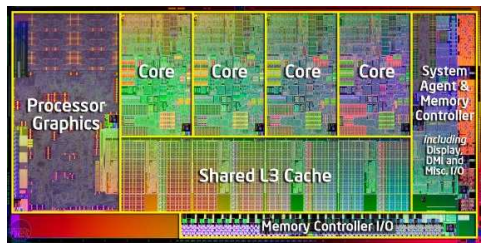
(q) What is 'P'?



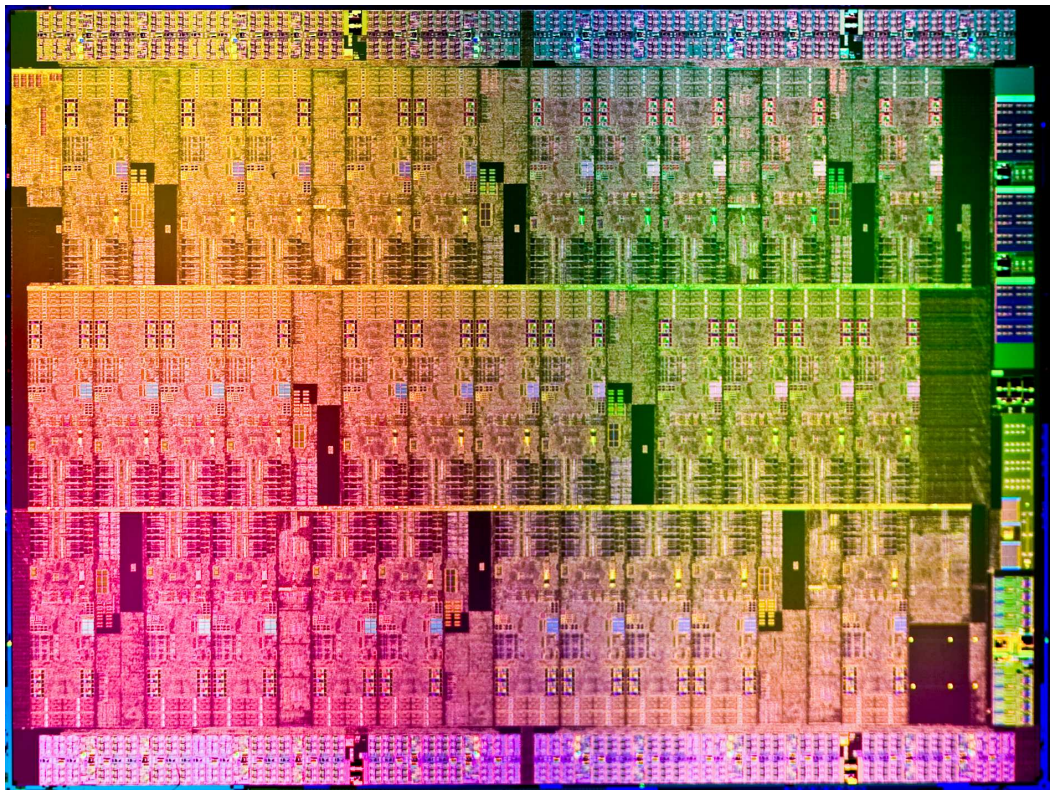
(r) AMD Opteron



(s) Intel Turkwilla



(t) Intel Sandy Bridge



The Knights Ferry MIC architecture board houses this 32-core Aubrey Isle processor.

Interconnection networks for multicore

- Initially employed busses and crossbar switches between the cores and cache banks
- Such solutions are not scalable to 1000 cores!
- On-chip technologies should scale close to linearly
- Scalable on-chip communication networks will borrow ideas from large-scale packet-switched networks.
- IBM Cell employ multiple ring networks to connect 9 processors on the chip and employ software-managed memory to communicate between between the cores rather than cache-coherent protocols.

Performance barriers (parallel overhead)

- Startup (latency) time
- Communication overhead
- Synchronization costs
- Imbalance of system resources (I/O channels and CPUs)
- Redundant computation
- load (dis-)balance

each of these can be in the range of milliseconds, i.e., millions of flops on modern computer systems

---

♦ Tradeoff: to run fast in parallel there must be a large enough amount of work per processing unit but not so large that there is not enough parallel work.

Parallel performance metrics

- $T(A,p)$ is the primary metric (was?)
- 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 $W(A,p)/W(A,1)$
- work wasted, $\cdots$
- scalability (strong, weak)

- $T(A,p)$

  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!

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.
* Numerically inefficient methods scale better.

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.

Skip 'Efficiency' » Cont2

- **Efficiency** : $E(A,p) = \dfrac{T_1}{pT_p} = \dfrac{1}{pS_p}$
- **Isoefficiency** : It has been observed (Grama, Gupta, Kumar etc.) that efficiency increases with the problem size N and decreases with increasing the number of processors p. The idea is to keep $E(A,p)$ constant, while suitably increase N and p simultaneously. Consider efficiency based on relative speedup, given as

$$E = \frac{1}{1 + p\frac{T(A,p)}{T(A,1)}}.$$

Since $T(A,1)$ is some function of N ($f(N)$), then
$f(N) = \frac{E}{1-E}\, pT(A,p)$.
Using some algebraic manipulations, it is possible to rewrite the latter as

$$N = \mathcal{E}(p)$$

and the function $\mathcal{E}(p)$, called the *isoefficiency function*, relates the growth of N and p so that E remains equal to a chosen constant.
The isoefficiency metric – useful in the analysis of the parallel

Both speedup and efficiency, as well as MFLOPSrate, are tools for analysis but not a goal of parallel computing.

None of these alone is a sufficient criterion to judge whether the performance of a parallel system is satisfactory or not. Furthermore, there is a tradeoff between the parallel execution time and the efficient utilization of many processors, or between efficiency and speedup.

One way to observe this is to fix N and vary p. Then for some p_1 and p_2 we have the relation

$$\frac{E(A,p_1)}{E(A,p_2)} = \frac{p_2 T(A,p_2)}{p_1 T(A,p_1)}.$$

If we want $E(A,p_1) < E(A,p_2)$ and $T(A,p_1) > T(A,p_2)$ to hold simultaneously, then $\frac{p_2}{p_1} < \frac{T(A,p_1)}{T(A,p_2)}$, i.e., the possibility of utilizing more processors is limited by the gain in execution time.

"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/processing units/cores."

Massively parallel ...?

We deal now with computers from 2, 4, 16, 32, to over 1000000 cores.

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 efficiency* $E(A, p)$ *can become bigger than some given efficiency* $E_0 \in (0, 1)$ *by increasing the size of the problem, i.e.,* $E(A, p)$ *stays bounded away from zero when* N *increases (efficiency-conserving model).*

Definition 3: *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 4: *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 (weak scalability):

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.

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:

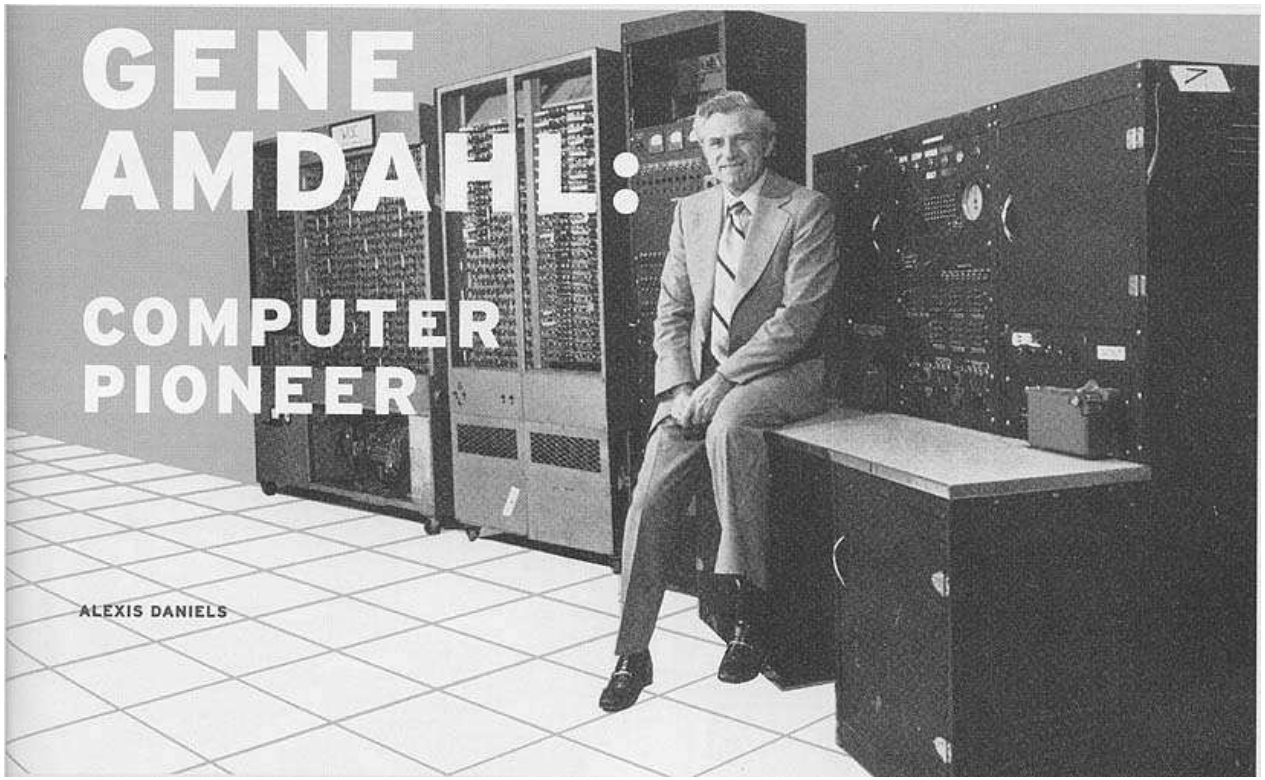
(i) lack of a perfect degree of parallelism in the algorithm;

(ii) idleness of computers due to synchronization and load imbalance;

(iii) of the parallel algorithm;

(iv) communication delays.

- Parallel performance
- Parallel performance measures
  - time
  - speedup
  - efficiency
  - scalability
- Parallel performance models
- Computational and communication complexity of algorithms
- Examples: nonoptimal – optimal algorithms
- Energy efficiency - models and metrics



Gene Amdahl, 1965

Gene Amdahl, March 13, 2008

Gene Amdahl's prophecy:

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.

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

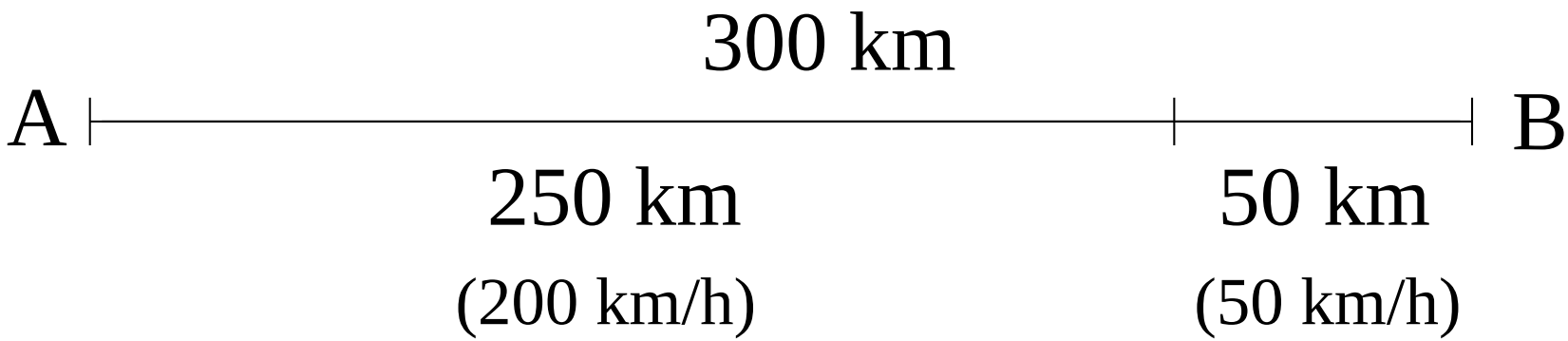
  Hence, the average rate $\overline{R}(= N/R)$ for the whole task is the weighted harmonic mean of $R_1, R_2, \ldots, R_k$.

  For the special case of only two subtasks - f_p (parallel) and $1 - f_p$ - serial, then

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

  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
  ($S = 1024/(0.996 + 0.004 * 1024) \approx 201$).

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

An example:

32 cores, 1% serial part and 0.99% parallel part

Amdahl's law: $S \leq 1/(0.01 + 0.99/32) = 24.43$

Gustafson's law: $S \leq 32 - 31 * 0.01 = 31.69$

Top 500, November 2012

1	Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x, 560 640 cores, 27.1125 PFlop/s, DOE/SC/Oak Ridge National Laboratory
2	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom, 1 572 864 cores, DOE/NNSA/LLNL
3	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect, 705 024 cores
4	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom, 786 432 cores, DOE/SC/Argonne National Laboratory
5	JUQUEEN BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect, 393 216 cores
⋮	⋮
83	Triolith - Cluster Platform SL230s Gen8, Xeon E5-2660 8C 2.200GHz, Infiniband FDR, 19 136 cores, National Supercomputer Centre

Top 500, November 2012

Titan – the first major supercomputing system to utilize a hybrid architecture: utilizes both conventional 16-core AMD Opteron CPUs and NVIDIA Tesla K20 GPU Accelerators.

Top 500, November 2012

BlueGene-Q:

- IBM PowerPC A2 1.6 GHz, 16 cores per node
- Networks
  - 5D Torus 40 GBps; 2.5 μsec latency (worst case)
  - Collective network – part of the 5D Torus; collective logic operations supported
  - Global Barrier/Interrupt – part of 5D Torus
  - 1 GB Control Network – System Boot, Debug, Monitoring
- L2 cache – multi-versioned, supporting transactional memory and speculative execution; has hardware support for atomic operations.

Models and metrics for the 'small-sized'

Skeleton-Based Approach
The 'Roofline' approach
Skipped »> Cont3

New performance models and metrics:

Examples:
Estimating Parallel Performance, A Skeleton-Based Approach
Oleg Lobachev and Rita Loogen, Proceeding HLPP'10 Proceedings of the 4th international workshop on High-level parallel programming and applications, ACM New York, NY, USA, 2010, 25–34.

Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures
Samuel Webb Williams, Andrew Waterman and David A. Patterson, Communications of the ACM, 52 (2009), 65-76.

The 'Skeleton' approach:

No of PEs - 'p', problem size - 'n', $W(n)$, $T(n) = W(n)$,
$T(n,p)$ - execution time on p PEs; assume $W(n,p) = pT(n,p)$.
In a parallel execution, the sequential work is distributed over the processors.
This causes an overhead, $A(n,p)$ (a penalty), which is also distributed over the p elements, thus, $A(n,p) = p\widetilde{A}(n,p)$. Then

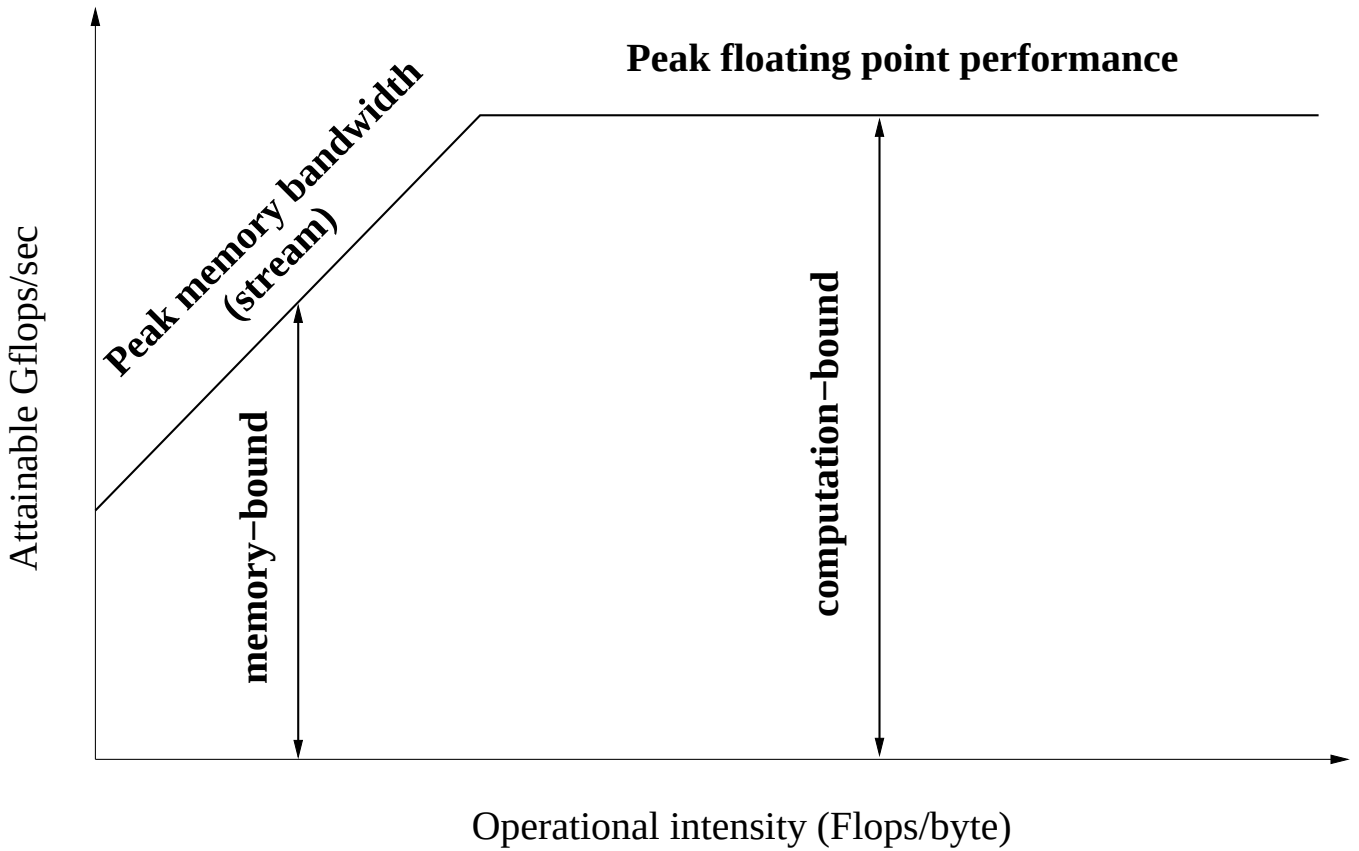
$$T(n,p) = T(n)/p + \widetilde{A}(n,p)$$

and

$$W(n,p) = T(n) + p\widetilde{A}(n,p) = T(n) + A(n,p)$$

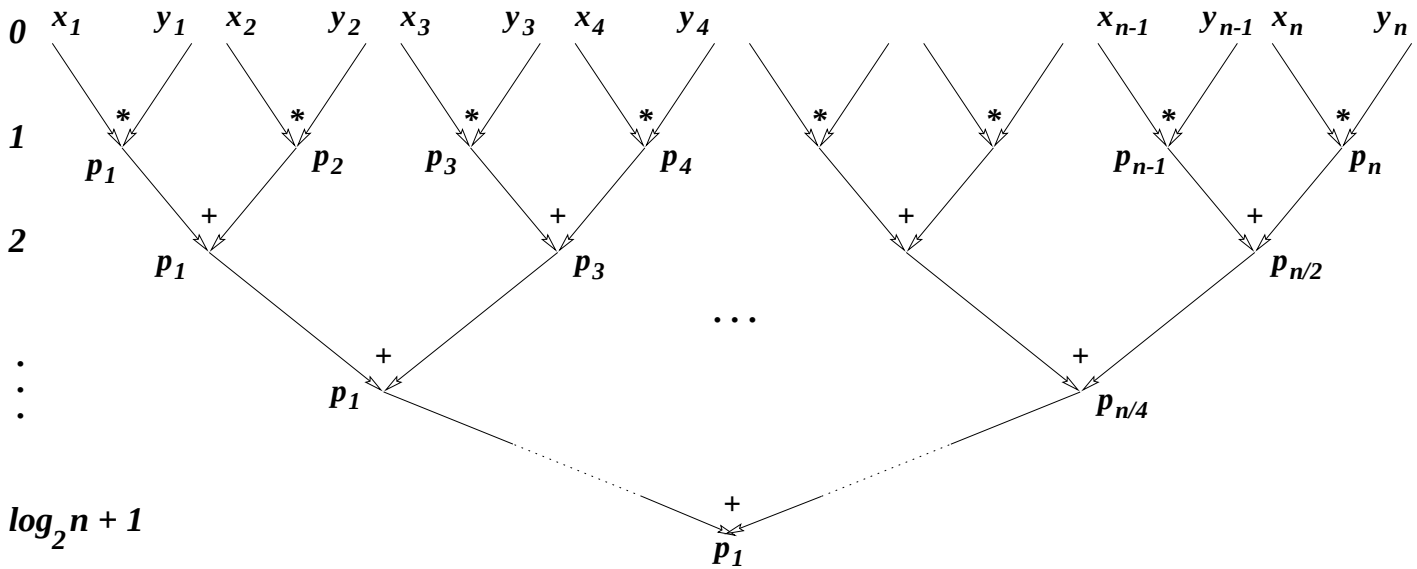
Task: try to estimate accurately $T(n)$ and $\widetilde{A}(n,p)$.
Then we can predict $T(n,p)$. Use 'skeletons' as abstract descriptions of the parallelization paradigm ('divide and conquer', 'iteration').

The 'Roofline' approach:



- Parallel performance
- Parallel performance measures
  – time
  – speedup
  – efficiency
  – scalability
- Computational and communication complexity of algorithms
- Examples: nonoptimal – optimal algorithms
- Energy efficiency - models and metrics
- Summary. Tendencies

A scalar product of two vectors of order n on p PE's (tree)



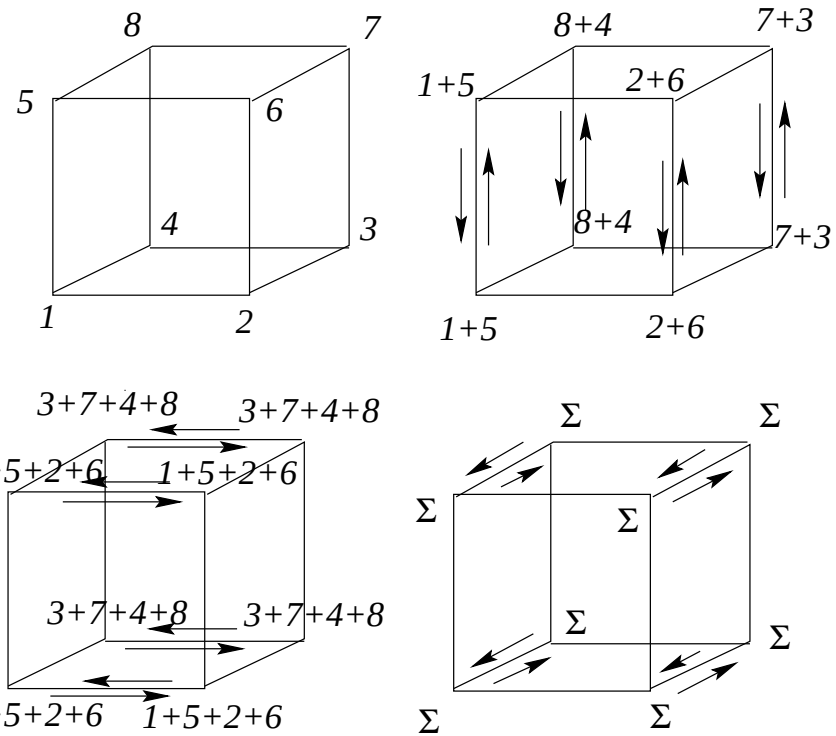
$T_1 = (2n-1)\tau$. For $p = n$ we get

$$T_p = (\log_2 n + 1)\tau + \alpha(\log_2 n + 1)\tau,$$

$$S = \frac{T_1}{T_p} = \frac{2n-1}{(\log_2 n + 1)} \cdot \frac{1}{1+\alpha}$$

and we see that the theoretical speedup is degraded by the factor $(1+\alpha)^{-1}$ due to data transport. If $p < n$, the data transport during the first $\lfloor n/p \rfloor - 1$ levels of the algorithm need correspondingly more data transports.

A scalar product of two vectors of order n on 3D hypercube



Σ = *1+2+3+4+5+6+7+8*

$$T_1 = (2n-1)\tau$$

$$T_p = \frac{n}{p}\tau + \alpha\, d\, \tau$$

GOTO the basic iterative methods...