Maximizing performance in HPC computations

Carl Nettelblad, Ali Dorostkar and Maya Neytcheva

November 23-27, 2015

Parallel performance - models and metrics

23.11.2015

About me:

About me:

- Maya Neytcheva
  Sofia University, University of Nijmegen, Uppsala University
  - *Background:* Mathematical modelling
  - *Work experience:* Design and implementation of information systems
  - *Interests:* Iterative solution methods, preconditioning, optimal solution methods, PDEs, parallelization, parallel performance
    PhD thesis: *'Arithmetic and communication complexity of preconditioning methods'*, 1995, University of Nijmegen, The Netherlands

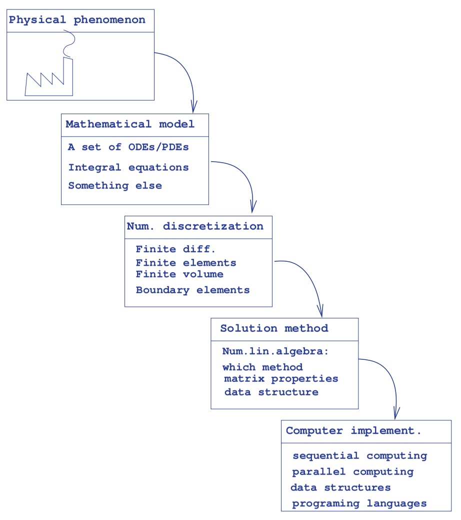
About you:

Research areas, represented in the group

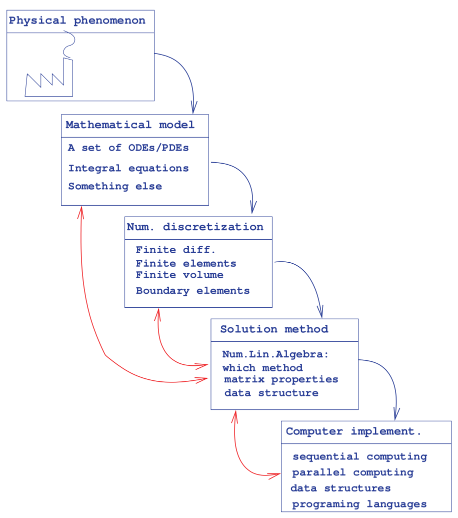
- Groundstate of anti-ferromagnetic spin chains (Hylke Donker)
- Cosmology (Adri Duivenvoorden)
- Beam-forming prediction for millimeter wave based on 3D simulation (Joo Gante
- Using HPC solutions to attain real time advanced medical imaging ultrasound techniques (Joao Amaro)
- Computer assisted image analysis (Sajith Sadanandan)
- Energy efficiency data communication with participatory sensing (Peramanathan Sathyamoorthy)
- Cut Finite Element methods for multiphase flow simulations (Thomas Frachon)
- Computer architecture, cache modeling (Moncef Mechri)
- Algorithms for the nonlinear eigenvalue problem (Giampaolo Mele)
- Data mining and recognition of historical handwritten documents (Tomas Wilkinson)
- Computer architecture / Distributed shared memory / Coherence mechanisms (Magnus Noren)
- Large scale electronic structure calculations (Anastasia Kruchinina)

About performing computer simulations/analysis:

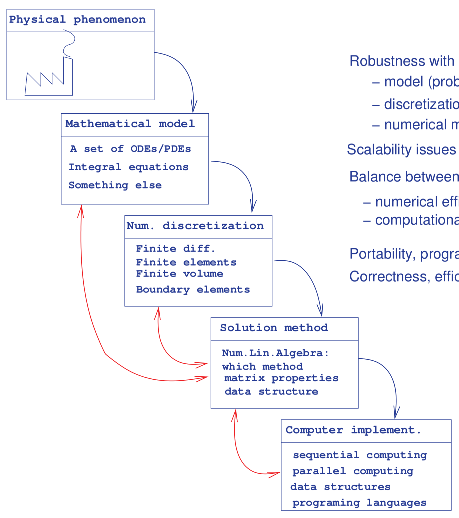
Performing computer simulations



Performing computer simulations



Performing computer simulations



Robustness with respect to
- model (problem) parameters
- discretization parameters
- numerical method parameters

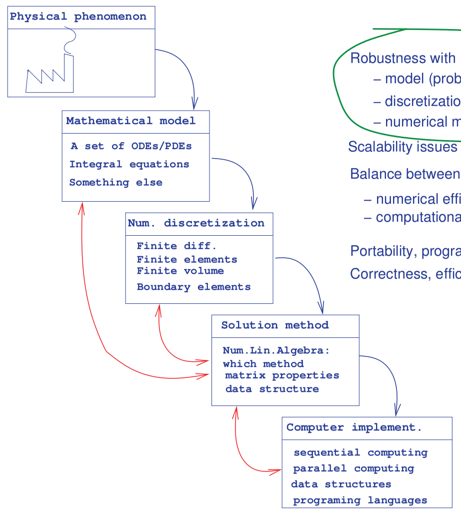
Scalability issues on various computers

Balance between
- numerical efficiency
- computational efficiency

Portability, programmability

Correctness, efficiency, optimization

Performing computer simulations: NLA issues



Robustness with respect to
- model (problem) parameters
- discretization parameters
- numerical method parameters

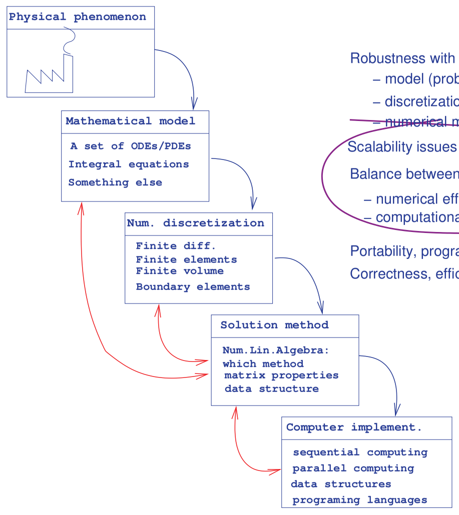
Scalability issues on various computers

Balance between
- numerical efficiency
- computational efficiency

Portability, programmability

Correctness, efficiency, optimization

Performing computer simulations: strategy



Robustness with respect to
- model (problem) parameters
- discretization parameters
- ~~numerical method parameters~~

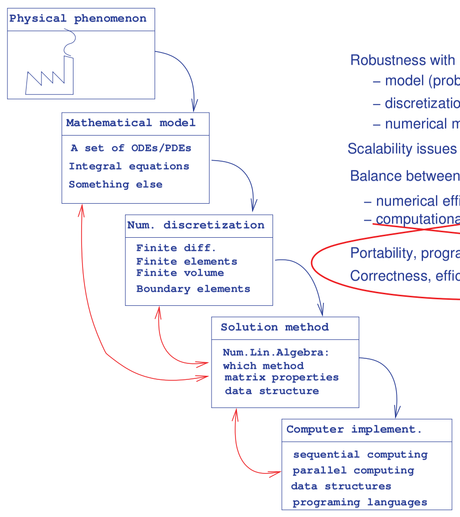
Scalability issues on various computers

Balance between
- numerical efficiency
- computational efficiency

Portability, programmability

Correctness, efficiency, optimization

Performing computer simulations: implementation issues



Robustness with respect to
- model (problem) parameters
- discretization parameters
- numerical method parameters

Scalability issues on various computers

Balance between
- numerical efficiency
- computational efficiency

Portability, programmability
Correctness, efficiency, optimization

Parallel performance issues

Plan of the presentation:

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

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

The setting:

We need to solve a problem 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?

• Did we do a good job? 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?

• The computers change all the time, how this affect the software counterpart?

Parallel performance

- We follow the historical development in the HPC area.
- However, a significant shift has taken place from single core to multi/many core and from homogeneous to heterogeneous computer architectures.

  Are the classical approaches to view performance relevant on the new computer architectures?

Computational and communication complexity:

the classical approach

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 <u>serial:</u> $T(A,1) = t_c W(A)$
  <u>parallel:</u> $T(A,p) = T_s(A) + \frac{T_p(A)}{p} + T_c(A,p)$
- 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 core):

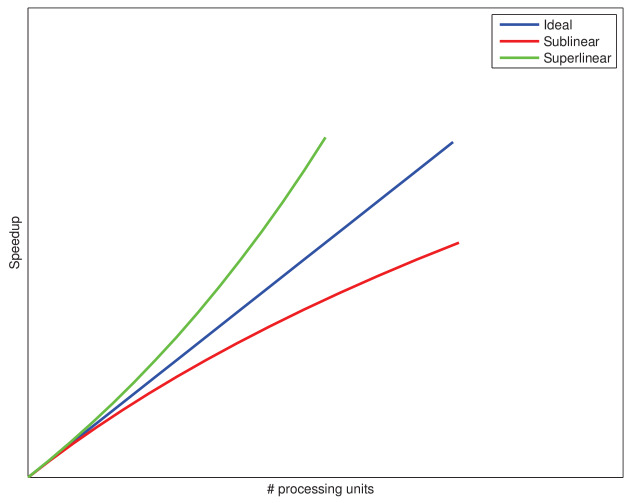
$$f = \frac{\#\textit{instructions per cycle}}{\tau}$$

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

More terminology: Granularity

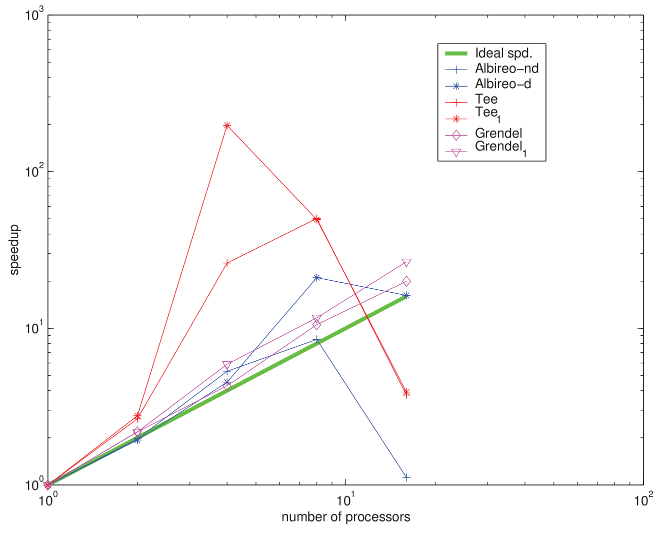
The term *granularity* is usually used to describe the complexity and type of parallelism, inherent to a parallel system.
granularity of a parallel computer and *granularity of computations*

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



Before defining the notion: speedup curves

In practice it is much more messy...



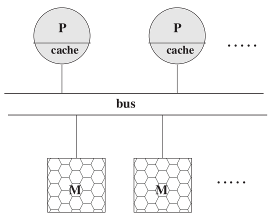
One algorithm on three parallel computers: speedup curves

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

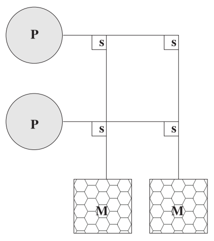
- How to measure the parallel performance?
  How to understand what we see on the performance plots?

- Recall some well-known stuff...

Recall: Shared memory machines

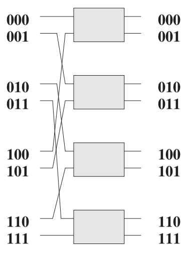


(a) bus-connected

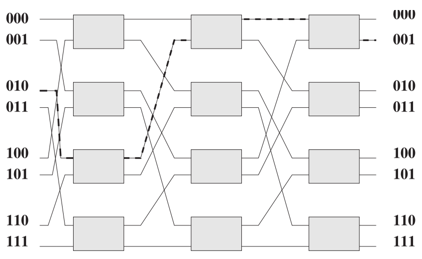


(b) crossbar-connected

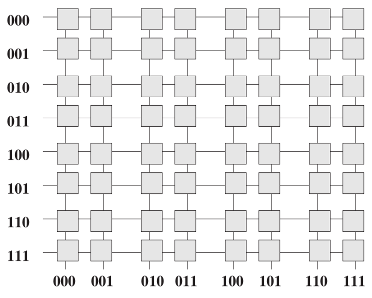
Recall: Shared memory machines



(c) Shuffle-exch.

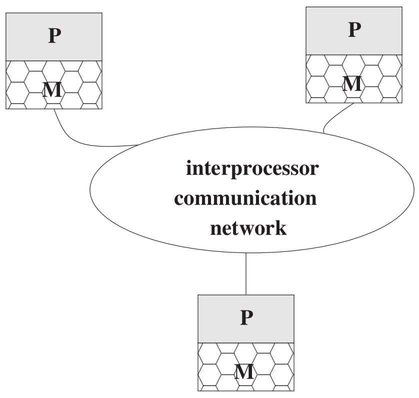


(d) Omega Network

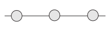


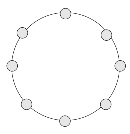
(e) Crossbar switch

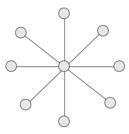
Recall: Distrib.memory: Static interconnection networks

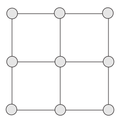


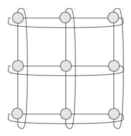
Recall: Interconnection network topologies

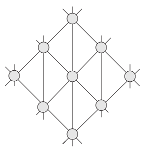

(f) linear array

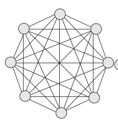

(g) ring

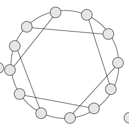

(h) star

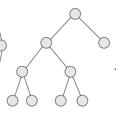

(i) 2D mesh

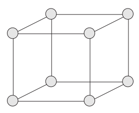

(j) 2D toroidal mesh

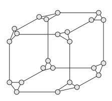

(k) systolic array


(l) completely connected


(m) chordal ring


(n) binary tree


(o) 3D cube


(p) 3D cube

Notes:

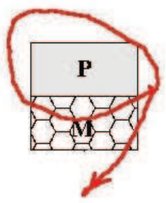
- Virtual (logical) topologies

Notes:

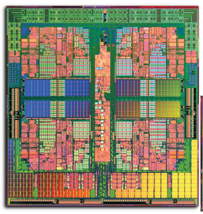
- Casualties of war: the machines are gone, the ideas live Once upon a time, in 1991, a computer was announced, KSR-1 (Kendal Square Research).
  The company went bankrupt in 1994.
  Why mention then?

Notes:

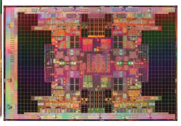
- Casualties of war: the machines are gone, the ideas live Once upon a time, in 1991, a computer was announced, KSR-1 (Kendal Square Research).
  The company went bankrupt in 1994.
  Why mention then?
- Because of the ideas!
  Novel memory architecture: the system level architecture was shared virtual memory, which was physically distributed in the machine.
  The programmer or application only saw one contiguous address space, which was spanned by a 40-bit address.
  The memory was automatically coherent - 'All-cache', a ring of rings.



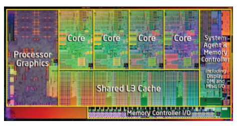
(q) What is 'P'?



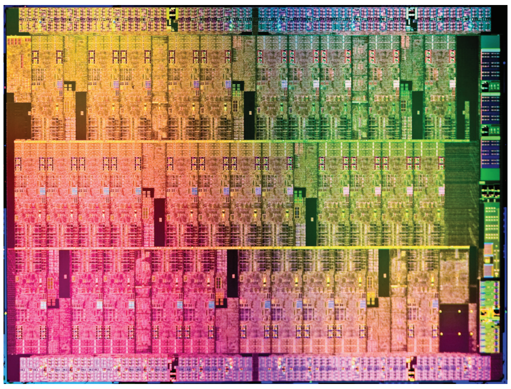
(r) AMD Opteron



(s) Intel Turkwilla



(t) Intel Sandy Bridge


Xeon Phi: The Knights Ferry MIC architecture board (32-core), later 72 cores, ...

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 the cores rather than cache-coherent protocols.

The road to even more parallelism in the hardware:

- Cluster with multi-socket blades: a limited number of procs have access to common pool of shared memory
- Same as before, but each proc. is a multicore unit
- Add accelerators - GPUs, MIC (many integrated cores)
-

Performance barriers (parallel overhead)

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

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 total 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!

Speedup:

Measuring *speedup* - pros and cons:
contra: Relative speedup "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/core 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.

- **Efficiency** : $E(A,p) = \frac{T_1}{pT_p} = \frac{1}{pS_p}$
- **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 performance of various parallel systems.

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 3000000 cores.

Scalability

How to define it?
* *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.*

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, 'fixed-size', 'strong' scalability.

How to define scalability?

- **Definition 1:** *A parallel system is scalable if the performance is linearly proportional to the number of processors used.*

- **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).*

How to define scalability?

- **Definition 1:** *A parallel system is scalable if the performance is linearly proportional to the number of processors used.*

- **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).*

How to define scalability?

- **Definition 1:** *A parallel system is scalable if the performance is linearly proportional to the number of processors used.*
- **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.

How to define scalability?

- **Definition 1:** *A parallel system is scalable if the performance is linearly proportional to the number of processors used.*
- **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).*

- **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.*

Strong vs weak scalability

Fixed size speedup (strong scalability):
Compare scalability figures when the problem size is kept fixed and the number of PEs is increased.

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.

Scalability example:

The parallelization of the summation problem $A = \sum_{i=1}^{N} a_i$ has been modeled for a broadcast medium to have an execution time proportional to $N/P + (\gamma + 1)P$, where γ is a constant of proportionality that can be interpreted as the the number of floating point additions that can be done in the time it takes to send one word.
Choosing P as a function of N can yield arbitrarily large speedup in apparent contradiction to Amdahl's Law. For the summation problem,

$$E_P = \left(1 + (\gamma + 1)\frac{P^2}{N}\right)^{-1}$$

For fixed N, this implies that the efficiency goes to zero as P goes to infinity.
But if we choose P as a function of N as N increases, we can obtain an efficiency that does not go to zero, as it would for the case of a fixed N. For example, suppose P and N are related by the equation $P = \sqrt{N}$. Then the efficiency is 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 cores/processing units due to synchronization and load imbalance;
(iii) due to the parallel algorithm itself;
(iv) communication delays.

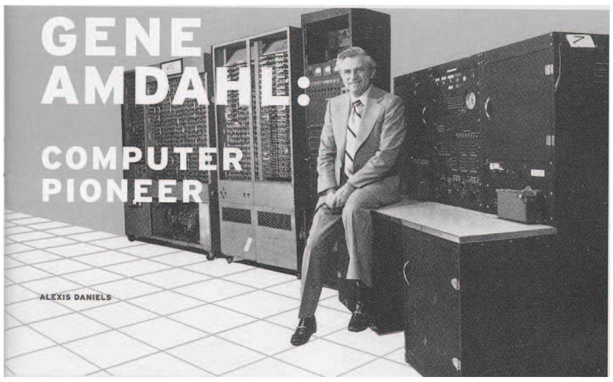
More on measuring parallel performance: cost

Definition: A parallel system is said to be *cost-optimal* if the cost of solving a problem in parallel is proportional to the execution time of the fastest-known sequential algorithm on a single processor.

The cost is understood as the product pT_p, i.e.

$$pT_p \sim T_{bestserial}.$$

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



Gene Amdahl, 1965

Born on Nov 11, 1922, 93 years old.

Gene Amdahl, March 13, 2008

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.

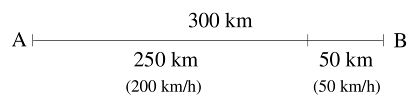
$$T = \sum_{i=1}^{k} t_i = \sum_{i=1}^{k} \frac{N_i}{R_i} = \sum_{i=1}^{k} \frac{f_i N}{R_i}; \quad \overline{R} = \frac{T}{N} N / \sum (f_i N / R_i) = \frac{1}{\sum_{i=1}^{k} f_i / R_i}$$

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

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

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

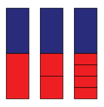
$$\begin{aligned} T(A,1) &= (1-f_p) + f_p p \\ T(A,p) &= (1-f_p) + f_p = 1 \quad \text{properly scaled problem} \\ S &= T(A,1) = p - (p-1)(1-f_p) \end{aligned}$$

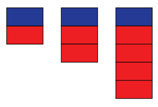
An example:

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: $S \leq 32 - 31 * 0.01 = 31.69$


(u) Amdahl's


(v) Gustafson-Barsis'

Let $f_p = 0.5$.

No.S PEs	Amdahl's	Gustafsson-Barsis'
2	$S = \frac{1}{0.5+0.5*2} = 1.33$	$S = 2 + 0.5(2-1) = 1.5$
4	$S = \frac{1}{0.5+0.5*4} = 1.6$	$S = 4 + 0.5(4-1) = 2.5$

- What has happened to the computer hardware since 1988?

- What has happened to the computer hardware since 1988?
- What has happened to the computer hardware since 1993?

1993	2010
2003	2011
2006	2012
	2015

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 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 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 40 Tflop.

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.
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 2.93 GHz 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
. . .
9 Forschungszentrum Juelich, Germany, JUGENE - Blue Gene/P IBM

Top 500, November 2010

Computer no 1: Peta-flop machine
Configuration of the system:
Computing node: 2560 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 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 μs
Programming framework for hybrid architecture - adaptive task partition and streaming data access.

Top 500, November 2011

1 RIKEN Advanced Institute for Computational Science (AICS) Japan, K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu, 705024 cores, 10.51 PetaFlops
2 National Supercomputing Center in Tianjin China, 186368 cores
3 DOE/SC/Oak Ridge National Laboratory United States, 224162 cores

Top 500, November 2011

Tokyo and Tsukuba, Japan, November 18, 2011 - A research group from RIKEN, the Univ. Tsukuba, the Univ. Tokyo, and Fujitsu Ltd announced that research results obtained using the "K computer" were awarded the ACM Gordon Bell Prize.
The award-winning results, revealed the electron states of silicon nanowires, which have attracted attention as a core material for next-generation semiconductors.
To verify the computational performance of the K computer, quantum-mechanical computations were performed on the electron states of a nanowire with approx. 10^5 atoms (20 νm in diameter and 6 νm long), close to the actual size of the materials, and achieved execution performance (*2) of 3.08 PFLOPS (representing execution efficiency of 43.6%).
The results of the detailed calculations on the electron states of silicon nanowires, comprised of 10,000 to 40,000 atoms, clarified that electron transport characteristics will change depending on the cross-sectional shape of the nanowire.

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, Sweden.

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.

Top 500, November 2015

1 Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.2 GHz, TH Express-2, Intel Xeon Phi 31S1P, 3,120,000 cores, 54902.4 TFLOP

2 Titan - Cray XK7 , Opteron 6274 16C 2.2 GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc., 560 640 cores, 27112.5 TFLOP

... ...

52 Beskow - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Aries interconnect Cray Inc., 53,632 cores, 1,397 TFLOP

Top 500, November 2012

Tianhe-2 (MilkyWay-2) :

- 16000 nodes, 32000 Intel Xeon CPUs, 48000 Intel Xeon Phis, 4096 FT CPUs
- Memory 1PB in total, 88 GB per node
- Interconnect: fat-tree topology, opto-interconnection, bidirectional bandwidth at 160 Gbps

Models and metrics for the 'small-sized'

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

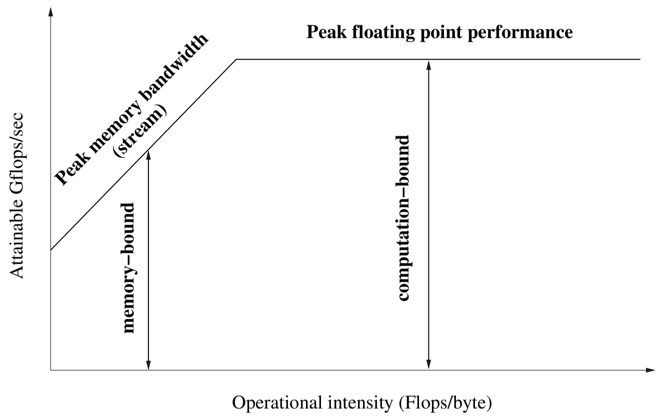
$$T(n,p) = T(n)/p + \widetilde{A}(n,p)$$

and

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

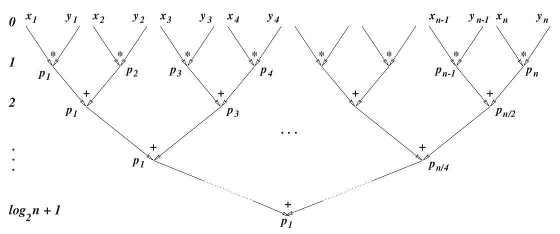
Task: try to estimate accurately $T(n)$ and $\widetilde{A}(n,p)$.
Then we can predict $T(n,p)$. Use 'skeletons' as abstract descriptions of the parallelization paradigm ('divide and conquer', 'iteration').

The 'Roofline' approach:


Peak floating point performance
Peak memory bandwidth (stream)
memory–bound
computation–bound
Attainable Gflops/sec
Operational intensity (Flops/byte)

- Parallel performance
- Parallel performance metrics
  - 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)



0
1
2
$\log_2 n+1$
x_1 y_1 x_2 y_2 x_3 y_3 x_4 y_4 x_{n-1} y_{n-1} x_n y_n
p_1 p_2 p_3 p_4 p_{n-1} p_n
p_1 p_3 $p_{n/2}$
p_1 $p_{n/4}$
p_1

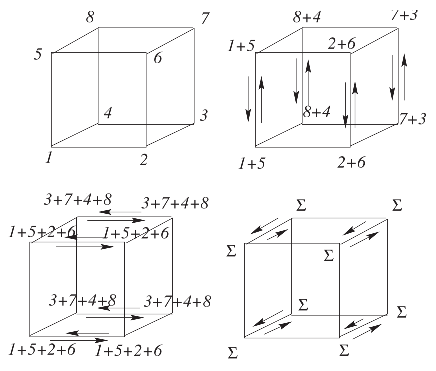
$T_1 = (2n-1)\tau$. For $p = n$ we get

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

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

and we see that the theoretical speedup is degraded by the factor $(1+\alpha)^{-1}$ due to data transport.

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



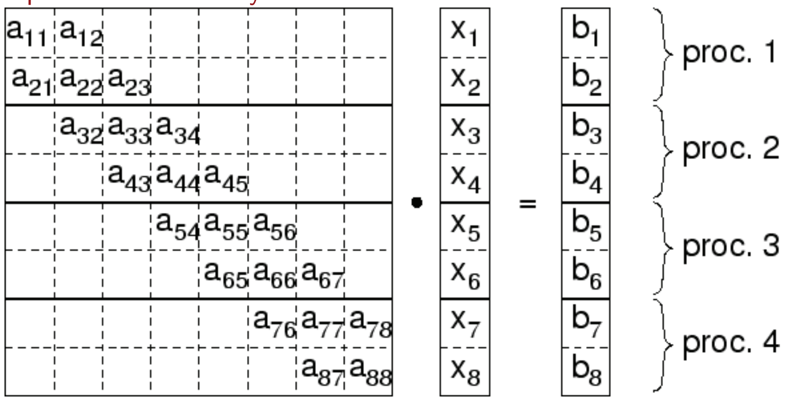
$\Sigma = 1+2+3+4+5+6+7+8$

$$T_1 = (2n-1)\tau$$
$$T_p = \frac{n}{p}\tau + \alpha\, d\, \tau$$

Matrix-vector product, distributed memory

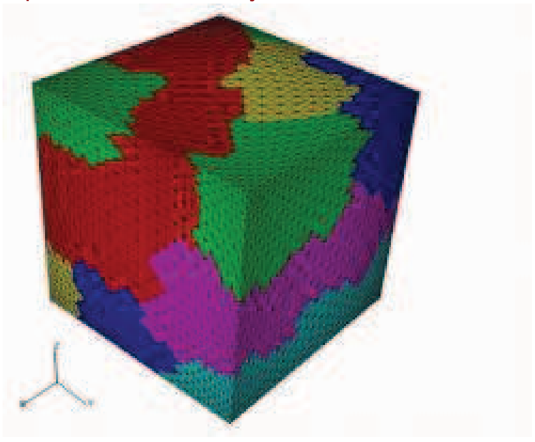
Question: How is the matrix distributed over the processing units?

Option 1: PETSc-style


a_{11} a_{12}
a_{21} a_{22} a_{23}
a_{32} a_{33} a_{34}
a_{43} a_{44} a_{45}
a_{54} a_{55} a_{56}
a_{65} a_{66} a_{67}
a_{76} a_{77} a_{78}
a_{87} a_{88}
•
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8
=
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8
proc. 1
proc. 2
proc. 3
proc. 4

Matrix-vector product, distributed memory

Question: How is the matrix distributed over the processing units?
Option 2: METIS-style



An algorithm which should scale very well ...

CG

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

… but does not scale …

Dusty-shelf Fortran code `stencil-based Conjugate Gradient method:`

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

Problem size	4			8		
	It	Time	Time/It	It	Time	Time/It
500^2	1474	19.69	0.0134	2447	56.96	0.0233
1000^2	2873	137.03	0.0477	4737	386.58	0.0816

Numerical efficiency – parallel efficiency – time

FEM-DD-PCG-SPAI

Assume A, B and $\mathbf{b}$ are distributed and the initial guess $\mathbf{x}^{(0)}$ is replicated.

CG-FEM-METIS style

```
        g^(0)   = $A\mathbf{x}^{(0)} - \mathbf{b}$,     $\mathbf{g}^{(0)} = replicate(\mathbf{g}^{(0)})$
        h       = $B\mathbf{g}^{(0)}$
        δ_0     = $(\mathbf{g}^{(0)}, \mathbf{h})$      $\mathbf{h} = replicate(\mathbf{h})$
        d^(0)   = $-\mathbf{h}$
For $k = 0, 1, \cdots$ until convergence
(1)     h       = $A\mathbf{d}^{(k)}$
(2)     τ       = $\delta_0/(\mathbf{h}, \mathbf{d}^{(k)})$
(3)     x^(k+1) = $\mathbf{x}^{(k)} + \tau\mathbf{d}^{(k)}$
(4)     g^(k+1) = $\mathbf{g}^{(k)} + \tau\mathbf{h}$,   $\mathbf{g}^{(k+1)} = replicate(\mathbf{g}^{(k+1)})$
(5)     h       = $B\mathbf{g}^{(k+1)}$,
(6)     δ_1     = $(\mathbf{g}^{(k+1)}, \mathbf{h})$    $\mathbf{h} = replicate(\mathbf{h})$
(7)     β       = $\delta_1/\delta_0$,                  $\delta_0 = \delta_1$
(8)     d^(k+1) = $-\mathbf{h} + \beta\mathbf{d}^{(k)}$
```

Poisson's equation

P	h_l	Probl. size	$E(n,P)$ $(n=2^{10})$	Total Time	Comm. Time	IT
1	$1/2^{10}$	1048576	1.0	3.8375	0	11
4	$1/2^{11}$	4194304	0.9841	3.8994	0.2217	11
16	$1/2^{12}$	16777216	1.0687	3.5908	0.2209	10
64	$1/2^{13}$	67108864	0.9008	4.2601	0.3424	10
256	$1/2^{14}$	268435456	0.8782	4.3697	0.3648	10
1024	$1/2^{15}$	1073741824	0.9274	4.1379	0.3962	9

Borrowed from a presentation by Ridgway Scott, Valpariaso, Jan 2011.
Parallel performance of U-cycle multigrid on IBM Blue Gene/L

FEM-SPAI: Scalability figures: Constant problem size

$\#proc$	n_{fine}	$t_{B_{11}^{-1}}/t_A$	t_{repl} [s]	$t_{solution}$ [s]	$\#$ $iter$
4	197129	0.005	0.28	7.01	5
16	49408	0.180	0.07	0.29	5
64	12416	0.098	0.02	0.03	5

Problem size: 787456
Solution method: *PCG*
Relative stopping criterium: $< 10^{-6}$

FEM-SPAI: Scalability figures: Constant load per processor

$\#proc$	$t_{B_{11}^{-1}}/t_A$	t_{repl} [s]	$t_{solution}$ [s]	# $iter$
1	0.0050	-	0.17	5
4	0.0032	0.28	7.01	5
16	0.0035	0.24	4.55	5
64	0.0040	0.23	12.43	5

Local number of degrees of freedom: 197129
Solution method: PCG
Relative stopping criterium: $< 10^{-6}$

Linear elasticity:

Solve
$$\begin{bmatrix} F & B^T \\ B & -M \end{bmatrix}, \quad \text{preconditioned by} \quad \begin{bmatrix} F & 0 \\ B & -S \end{bmatrix}.$$

Linear elasticity, MPI

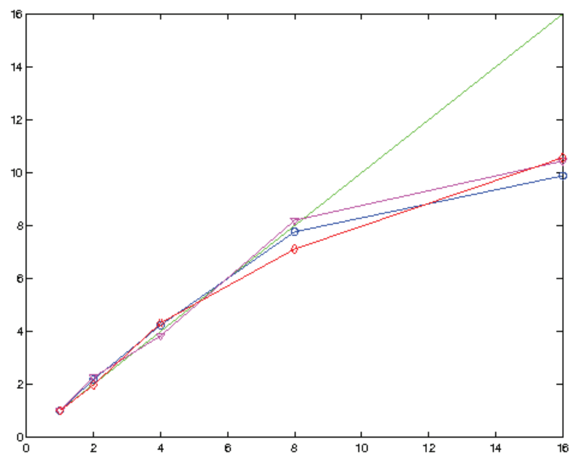
Cores	DOFs	$\nu = 0.2$		
		Iterations	Setup	Solve
1	1 479 043	10(8, 1)	25.3	86.9
4		10(8, 1)	7.11	23.2
8		10(8, 1)	3.62	12.9
16		10(8, 1)	2.19	9.51
32		10(8, 1)	1.26	6.8
1	5 907 203	10(9, 1)	101	435
4		10(9, 1)	29.1	99.2
8		10(8, 1)	14.6	49.8
16		10(9, 1)	8.68	41.1
32		10(9, 1)	4.27	30.9
64		10(9, 1)	5.22	21.1
4	23 610 883	10(11, 1)	165	563
8		10(11, 1)	65.4	267
16		10(10, 1)	36.9	298
32		10(11, 1)	18.4	120
64		10(11, 1)	9.69	75.8
128		10(11, 1)	9.57	60.8
256		10(11, 1)	6.61	33.3
32	94 407 683	10(12, 1)	82.1	536
64		10(12, 1)	40	507
128		10(12, 1)	21.5	272
256		10(11, 1)	14	98.9
512		10(11, 1)	9.79	72.9

Does this algorithm scale or not?

A version of `wave`:
Solve 2D advection equation with forcing term numerically with the Leap-frog scheme

Problem size	1	2	4	8	16
Dec, 2010					
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	-
Dec, 2011					
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

The Speedup of the wave solver



An algorithm that scales ...

Grid size	Coarsest level No (total no. of levels)	Number of PEs					Time (sec)
		4	8	16	32	64	
256^2	10(16)	403.49 5.31	189.06 5.93	93.86 5.58	49.18 4.62	28.71 3.89	outer comm.
512^2	12(18)			632.60 629.44 14.28	304.24 302.71 12.14	154.65 153.81 10.14	total outer comm.
1024^2	12(20)				1662.73 1655.73 29.89	829.71 826.22 22.26	total outer comm.

Stokes problem: Performance results on the Cray T3E-600 computer

Another algorithm that scales ...

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.

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
... Additionally, this work is significant in that no special purpose algorithms or implementations were required to achieve a highly scalable performance on a common parallel computer.

3D Multiphase flow simulations

Size	71874	549250	4293378
	(i1), PCG with AMG		
$N_1/N_2/N_3$	2 / 12 / 5	2 / 11 / 5	2 / 10 / 5
T	4.82	36.86	287.40
R		7.64	7.79
	(i2), PCG with AMG		
N_1/N_3	26/4	20/5	18/5
T	5.02	33.42	236.97
R		6.66	7.09

Average iteration counts N_1, N_2, and N_3, average wall time T in seconds and factor R of increase of T for three consecutive mesh refinements, methods (i1) and (i2) on one processor.

3D Multiphase flow simulations

Size	549250	4293378	33949186	270011394
No. cores	1	8	64	512
	(i1)			
$N_1/N_2/N_3$	3 / 12 / 24	3 / 10 / 38	3 / 11 / 57	3 / 10 / 103
T	39.1	136.4	293.9	520.7
R_{CG}		1.32	1.65	1.64
R		3.49	2.16	1.77
	(i2)			
N_1/N_3	20 / 22	18 / 35	17 / 56	14 / 95
T	70.2	127.4	217.2	355.6
R_{CG}		1.60	1.60	1.70
R		1.81	1.71	1.64

Weak scalability test, CG used for systems with $M + \varepsilon\sqrt{\beta}K$. Average iteration counts N_1, N_2, and N_3, average wall time T in seconds, factor R_{CG} of increase of the inner CG iterations and factor R of increase of T after one mesh refinement

3D Multiphase flow simulations

Size	549250	4293378	33949186	270011394
No. cores	1	8	64	512
		(i1)		
$N_1/N_2/N_3$	3 / 11 / 5	3 / 10 / 6	3 / 9 / 6	3 / 9 / 6
T	35.36	91.02	145.66	275.20
R		2.57	1.60	1.89
		(i2)		
N_1/N_3	20/5	17/5	17/6	14/6
T	31.45	67.53	136.56	140.59
R		2.15	2.02	1.03

Weak scalability test, PCG with AMG preconditioner used for systems with $M + \varepsilon\sqrt{\beta}K$. Average iteration counts N_1, N_2, and N_3, average wall time T in seconds and factor R of increase of T after one refinement of the mesh

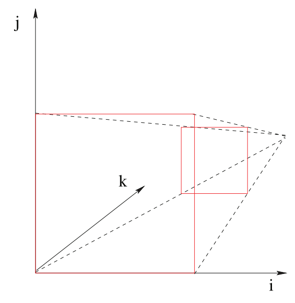
Gauss Elimination

Dependences in Gaussian elimination System of equations and standard sequential algorithm for Gaussian elimination:

```
for k=1,n
    for i=k+1,n
        l(i,k) = a(i,k)/a(k,k)
    endfor(i)
    for j=k+1,n
       for i=k+1,n
           a(i,j) = a(i,j) - l(i,k) * a(k,j)
       endfor(i)
    endfor(j)
endfor(k)
```

Sequential Gaussian elimination: multiple loops

Gauss Elimination



The iteration space for the standard sequential algorithm for Gaussian elimination forms a trapezoidal region with square cross-section in the i, j plane.
Within each square (with k fixed) there are no dependencies.

Gauss Elimination

(1) The dominant part of the computation when solving a system with a direct solver is the factorization, in which L and U are determined.
The triangular system solves require less computation.
(2) There are no loop-carried dependences in the inner-most two loops (the i and j loops) because $i, j > k$. Therefore, these loops can be decomposed in any desired fashion.
(3) The algorithm for Gaussian elimination can be parallelized using a message-passing paradigm. It is based on decomposing the matrix column-wise, and it corresponds to a decomposition of the middle loop (the j loop). A typical decomposition would be cyclic, since it provides a good load balance.

Gauss Elimination

```
for k=1,n
   if( " I own column k " )
     for i=k+1,n
         l(i,k) = a(i,k)/a(k,k)
     endfor(i)
     "broadcast" l(k+1 : n)
   else "receive" l(k+1 : n)
   endif
   for j=k+1,n ("modulo owning column j")
      for i=k+1,n
          a(i,j) = a(i,j) - l(i,k) * a(k,j)
      endfor(i)
   endfor(j)
endfor(k)
```

Standard column-storage parallelization Gaussian elimination.

Gauss Elimination

We can estimate the time of execution of the standard Gaussian elimination algorithm as follows. For each value of k, $n-k$ divisions are performed in computing the multipliers $l(i,k)$, then these multipliers are broadcast to all other processors. Once these are received, $(n-k)^2$ multiply-add pairs (as well as some memory references) are executed, all of which can be done in parallel. Thus the time of execution for a particular value of k is

$$c1(n-k) + c2\frac{(n-k)^2}{P}$$

where the constants c_i model the time for the respective basic operations. Here, c_2 can be taken to be essentially the time to compute a 'multiply-add pair' $a = a - b * c$ for a single processor. The constant c_1 can measure the time both to compute a quotient and to transmit a word of data.

Gauss Elimination - speedup, efficiency and scalability

Summing over k, the total time of execution is

$$\sum_{k=1}^{n-1}\left(c_1(n-k)+c_2\frac{(n-k)^2}{P}\right)\approx\frac{1}{2}c_1n^22+\frac{1}{3}c_2\frac{n^3}{P}$$

Time to execute this algorithm sequentially is $\frac{1}{3}c_2n^3$.
Speed-up for standard column-storage parallelization of Gaussian elimination is

$$S_{P,n}=\left(\frac{2}{3}\frac{\gamma}{n}+\frac{1}{P}\right)^{-1}=P\left(\frac{2}{3}\frac{\gamma}{n}P+1\right)^{-1}$$

where $\gamma=c_1/c_2$ - ratio of communication to computation time. Efficiency is

$$E_{P,n}=\left(\frac{2}{3}\frac{\gamma P}{n}+\frac{1}{P}\right)^{-1}\approx 1-\frac{2}{3}\frac{\gamma P}{n}.$$

Thus the algorithm is scalable; we can take $P_nn=\epsilon n$ and have a fixed efficiency of

$$\left(\frac{2}{3}\gamma\epsilon+1\right)^{-1}.$$

The efficiency will be the same for values of P and n which have the same ratio P/n.

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

Energy efficiency

The topic is very broad and has various aspects:

- Hardware design
- Scheduling policies
- Algorithmic aspects

Metrics:

MIPS per Watt	MIPS^2 per Watt	MIPS^3 per Watt
Energy per operation	FLOPS per Watt	Performance per Watt
Speedup per Watt	Power per speedup	Energy per target

Dynamic Voltage Frequency Scaling (DVFS)

Energy efficiency, hardware design

- Et^2 On the level of VLSI (CMOS)
  (A. Martin, Mika Nyström, P. Pénzes)
  Observation: Et^2 is independent of the voltage
- Power-aware speedup
- Iso-energy efficiency: (Song, Su, Ge, Vishnu, Cameron)
  As the system scales, maintain constant per-node performance
  to power

Energy efficiency, scheduling policies

Metrics and task scheduling policies for energy saving in multicore computers
Majr, Leung, Huang, 2010
Energy=Power$\times$time
Power efficiency is not equal to energy efficiency.
Speedup per Watt - how to use power to achieve high performance
Power per speedup - how much power is needed for each performance unit (find out the best hardware configuration that uses min energy and still achieve the required performance.

Energy efficiency, algorithmic aspects

A new energy aware performance metric
Costas Bekas and Alessandro Curioni, Computer Science, Volume 25, Nr. 3-4, 187-195, 2010

Consider the following target:
Computing systems and algorithms that run on them should be designed in harmony and simultaneously optimized such that both power consumption and time to solution are minimized.
The proposed metric is

$$f(\text{time to solution})\cdot \text{energy (FTTSE)}$$

where $f(\cdot)$ is an application-dependent function of time.
(Contrast to Flops per Watt, where the two ingredients are optimized separately.)

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

Final words: High Performance Computing, where are we?

◇ Performance:

- Sustained performance has increased substantially during the last years.
- On many applications, the *price-performance* ratio for the parallel systems has become smaller than that of specialized supercomputers. But · · ·
- Still, some applications remain hard to parallelize well (adaptive methods).

◇ Languages and compilers:

- Standardized, portable, high-level languages exist. But · · ·
- Message passing programming is tedious and hard to debug. OpenMP has its limitations. Combination - good.
- GPUs still rather specialized
- Programming difficulties remain still a major obstacle for mainstream scientists to parallelize existing codes.
- Revisit some 'old' algorithms, come up with new languages.

Final words: High Performance Computing, where are we?

We are witnessing and we are given the chance to participate in the exiting process of parallel computing achieving its full potential power and solving the most challenging problems in Scientific Computing/Computational Mathematics/Bioinformatics/Data Mining etc.