Outline Lecture 1

1. Computationally intensive problems
2. Developments in technology
3. Principles of parallel computing

1. Computationally intensive problems

Units of HPC

Mflop	Megaflop	10^6 flop/sec	1961
Gflop	Gigaflop	10^9 flop/sec	1983
Tflop	Teraflop	10^{12} flop/sec	1997
Pflop	Petaflop	10^{15} flop/sec	2008
MB	Megabyte	10^6 bytes	
GB	Gigabyte	10^9 bytes	
TB	Terabyte	10^{12} bytes	
PB	Petabyte	10^{15} bytes	
EB	Exabyte	10^{18} bytes	

Scientific Paradigms (1/2)

- Traditional scientific and engineering paradigms:
  - *Do theory or paper design and analysis*
  - *Perform experiments or build systems*

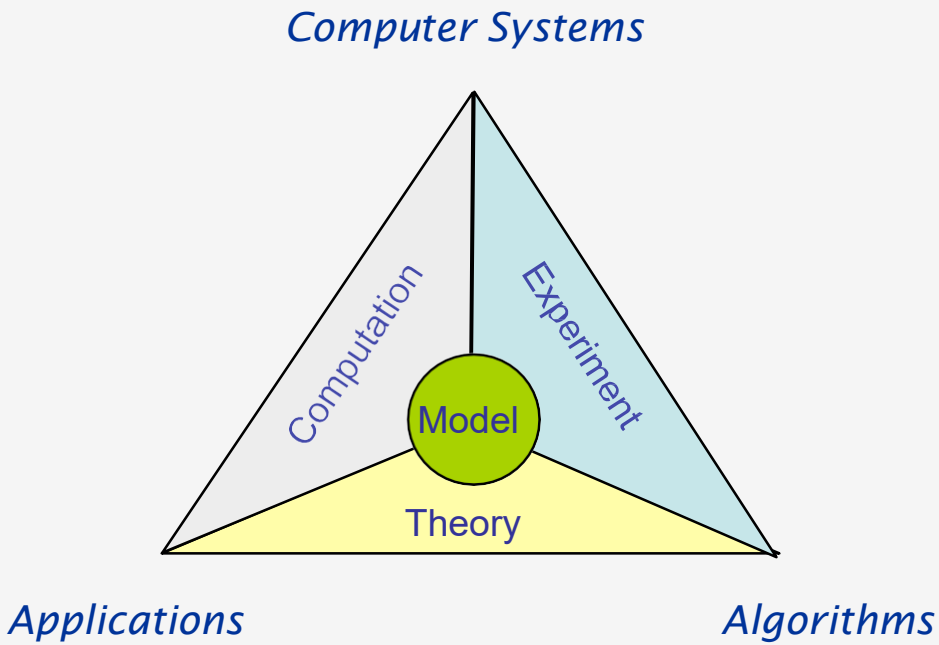
- Limitations of experiments:
  - *Too difficult:* *build large wind tunnels*
  - *Too expensive:* *build a throw-away airplane*
  - *Too slow:* *wait for climate or galactic evolution*
  - *Too dangerous:* *weapons, drug design*

Scientific Paradigms (2/2)

- Computational Science paradigm:
  - *use high-performance computer systems to simulate the phenomenon*
  - *based on known physical laws and efficient numerical methods*

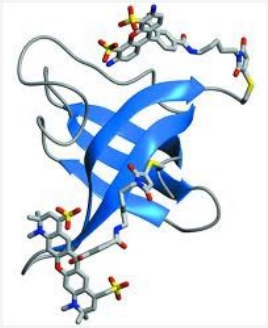
- **Question**: only physical laws?

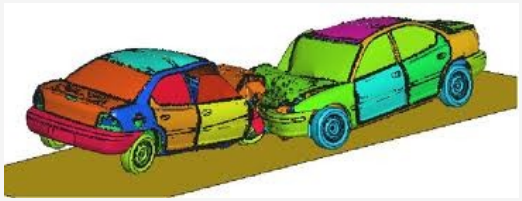
Computational Science Triangle



Some challenging computations (1/2)

- Science
  - *Global climate*
  - *Astrophysics*
  - *Biology: genome analysis, protein folding*
  - *Earthquake and tsunami prediction*
- Engineering
  - *Car-crash simulation*
  - *Semiconductor design*
  - *Structural analysis*
  - *Oil and gas exploration*
  - *Aerodynamics/fluid dynamics*





Some challenging computations (2/2)

- Business
  - *financial and economic modeling*
  - *transaction processing*
  - *web search and indexing engines*
- Defense and security
  - *nuclear weapon simulation*
  - *cryptography*
- Entertainment
  - *movie animations*
  - *games*

Global Climate Modeling Problem (1/2)

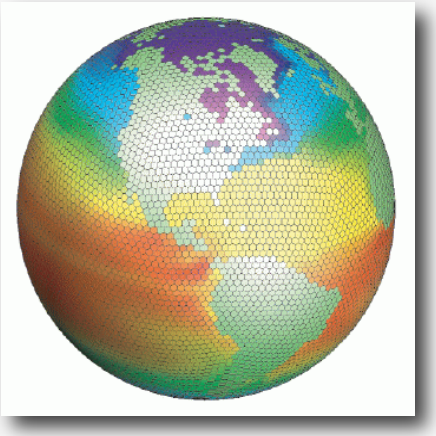
- Problem: compute

  F(latitude, longitude, elevation, time) = weather
  (temperature, pressure, humidity, wind velocity)

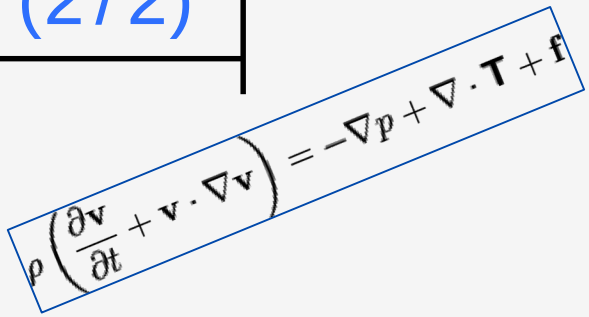
- Approach:
  - *discretize the domain, e.g., a grid with measurement points every N km*
  - *devise a model to predict climate at time t+Δt given climate at time t*

Example of a global climate model of the observed sea-surface temperature distribution.
Grid has 10,242 cells, with 240 km spacing.

Source: Randall, D. A. et. al., Climate modeling with spherical geodesic grids,
Computing in Science and Engineering, 2002.



Global Climate Modeling Problem (2/2)

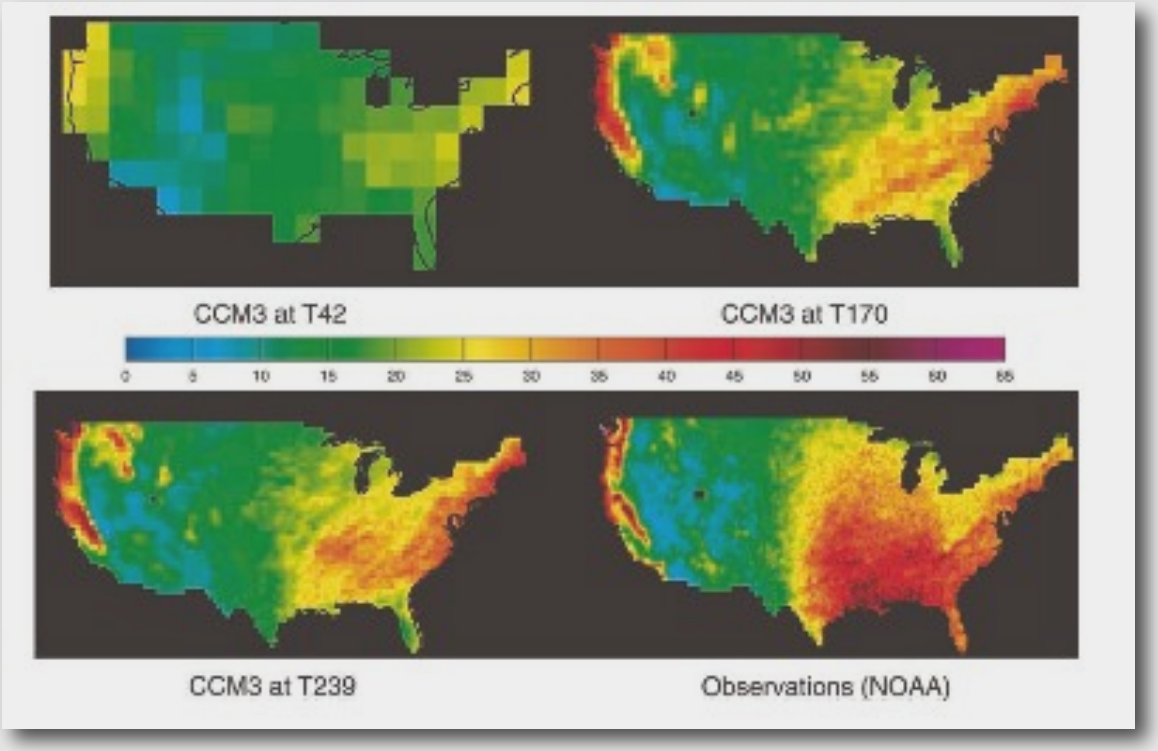


- Model the fluid flow in the atmosphere
  - *Solve the resulting Navier-Stokes equations*
  - *Roughly 100 Flops per grid point with 1 minute timestep*
  - *Earth surface is approximately 5.1×10^8 km^2*
  - *For grid patches of 1 km^2 we need to calculate 5.1×10^{10} Flops per time step*
- Computational requirements:
  - *To match real-time, 5×10^{10} flops in 60 seconds ≈ 0.8 Gflop/s*
  - *Weather prediction (7 days in 24 hours): 5.6 Gflop/s*
  - *Climate prediction (100 years in 30 days): 1 Tflop/s*
  - *Scenario analysis (100 scenarios): 100 Tflop/s*

The Effect of Grid Resolution

- Daily precipitation during winter time: models with different resolutions versus observations

300 km resolution *75 km resolution*



50 km resolution

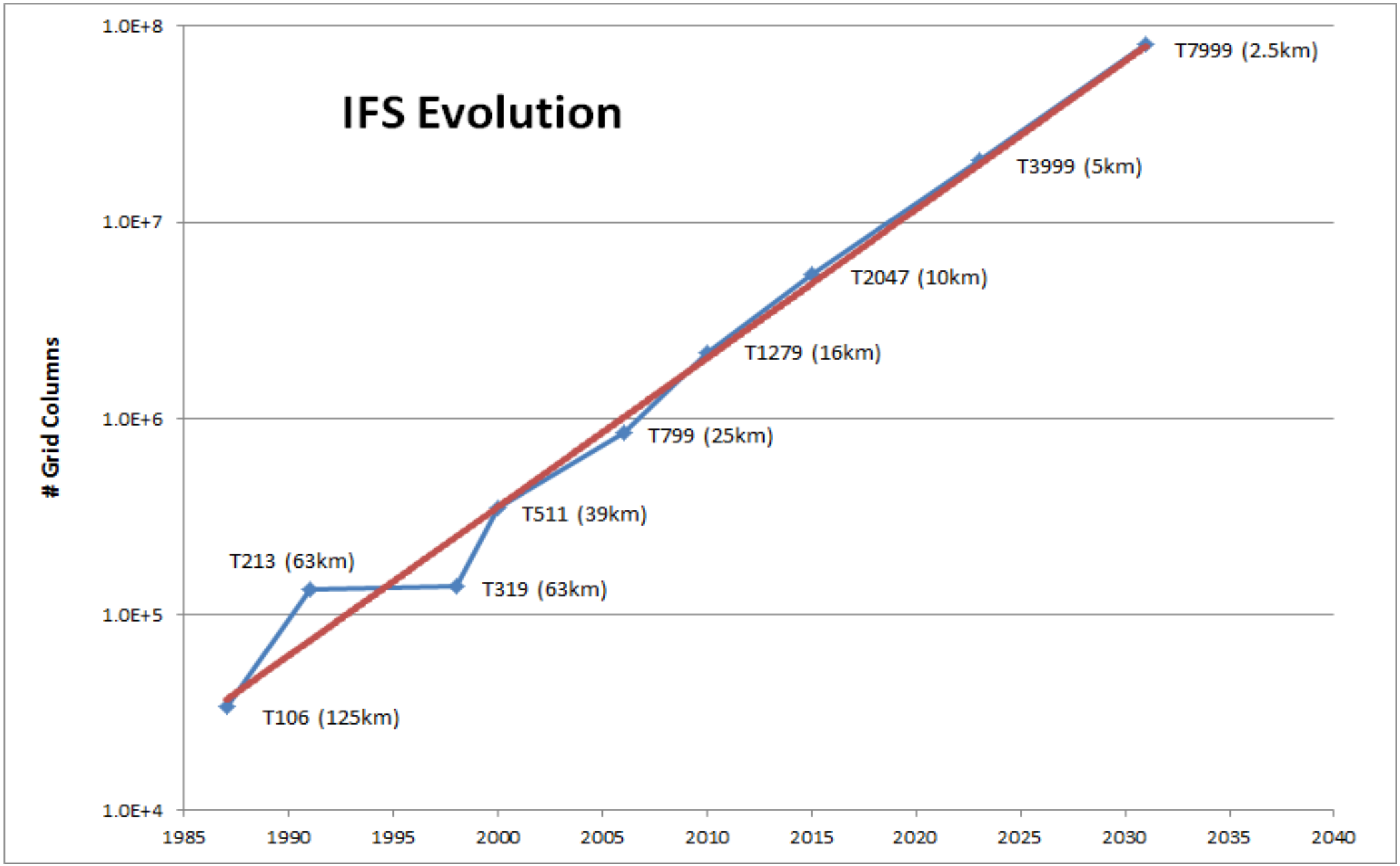
[source: Duffy et al, High resolution simulations of global climate, Climate Dynamics (2003), 21]

Global Climate Modeling Problem

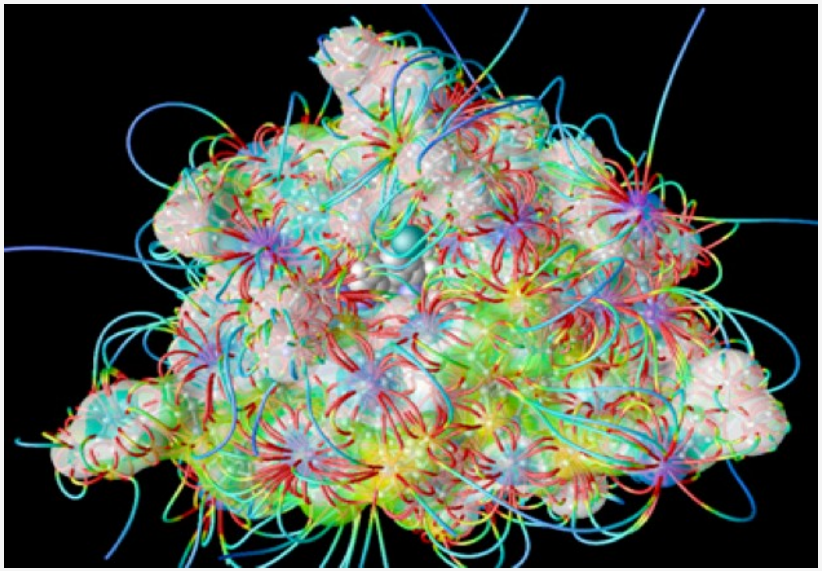
- To double the grid resolution: 4x computation (for 2D)
- Add *B* elevation levels is at least *B*x
- Need to include adequate couplings for submodels (ocean, land, atmosphere, sea-ice)
- Leads to thousands of Tflops (Pflops)
- Current (2021) top-1 supercomputer: ~442 Pflops on LINPACK
  - *BUT, LINPACK performance is NOT application performance (e.g. 'only' 16 Pflops for HPCG benchmark)*
- What can we do?
  - *use coarser and less accurate models*
  - *build bigger/better/faster systems*
  - *invent new parallel algorithms*

ECMWF Weather Modelling and Prediction

Integrated Forecast System Spatial Resolution



Protein Folding (1/2)



The case for the IBM BlueGene project: > 1,000,000 processors

Protein Folding (2/2)

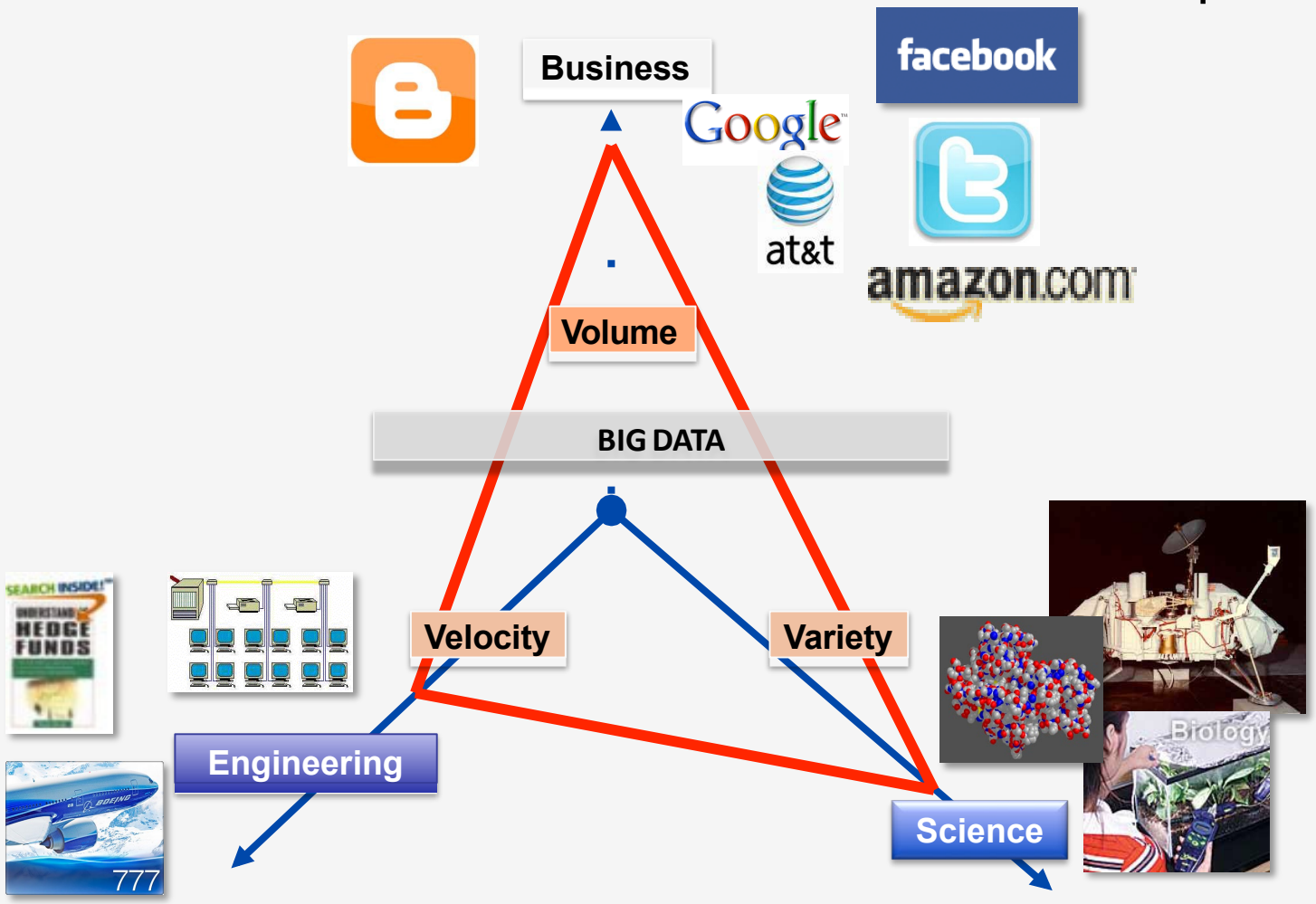
Physical time for simulation	10^{-4} s
Typical time-step size	10^{-15} s
Number of MD time steps	10^{11}
#atoms in a typical protein and water simulation	32,000
Approximate number of interactions in force calculation	10^{9}
Machine instructions per force calculation	1000
Total number of machine instructions	10^{23}

It would take 3 years to simulate 100μs on a Petaflop machine!!

New Challenges: Big Data

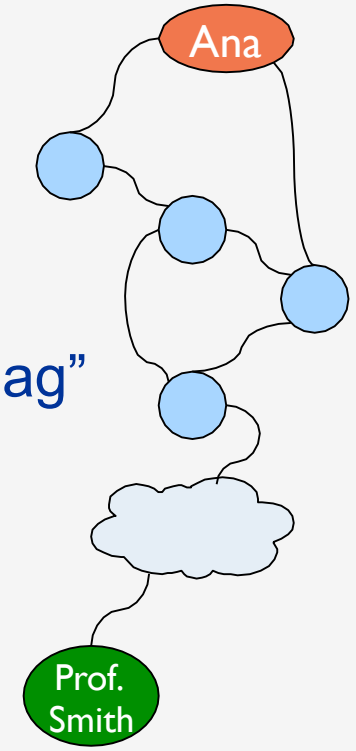
- Traditional HPC applications do processing of data in-memory
  - get data in, process data, and output results
- A new class of HPC applications has emerged that can be characterized by any of the following:
  - Huge volume of data (Volume)
  - Data is generated at a high speed (Velocity)
  - Data has large variety of data formats (Variety)
- Processing is used to do Data Analytics/Data Mining
  - Data can only be partially in core at any point in time
  - Data is often irregular (e.g., graphs)

Big Data: VVV


Business
facebook
Google
at&t
amazon.com
Volume
BIG DATA
Velocity
Variety
Engineering
Science
Biology
777

Graph Processing

- Traversing
  - "How can I reach Prof. Smith?"
- Querying
  - "Find all professionals in Physics around Den Haag"
- Mining
  - "Find the most influential parallel computing researcher in Delft?"



[courtesy Ana Varbanescu]

HPC for Web Applications (1/4)

- Web crawling, indexing, sorting
- Preprocessing of the web data to compute derived data, e.g.:
  - *inverted indexes*
  - *representations of graph structure of documents*
  - *summaries of no. of pages crawled per host*
  - *most queries per day*
- Also holds for other large unstructured data sets
- In fact, specific big data application

HPC for Web Applications (2/4): MapReduce

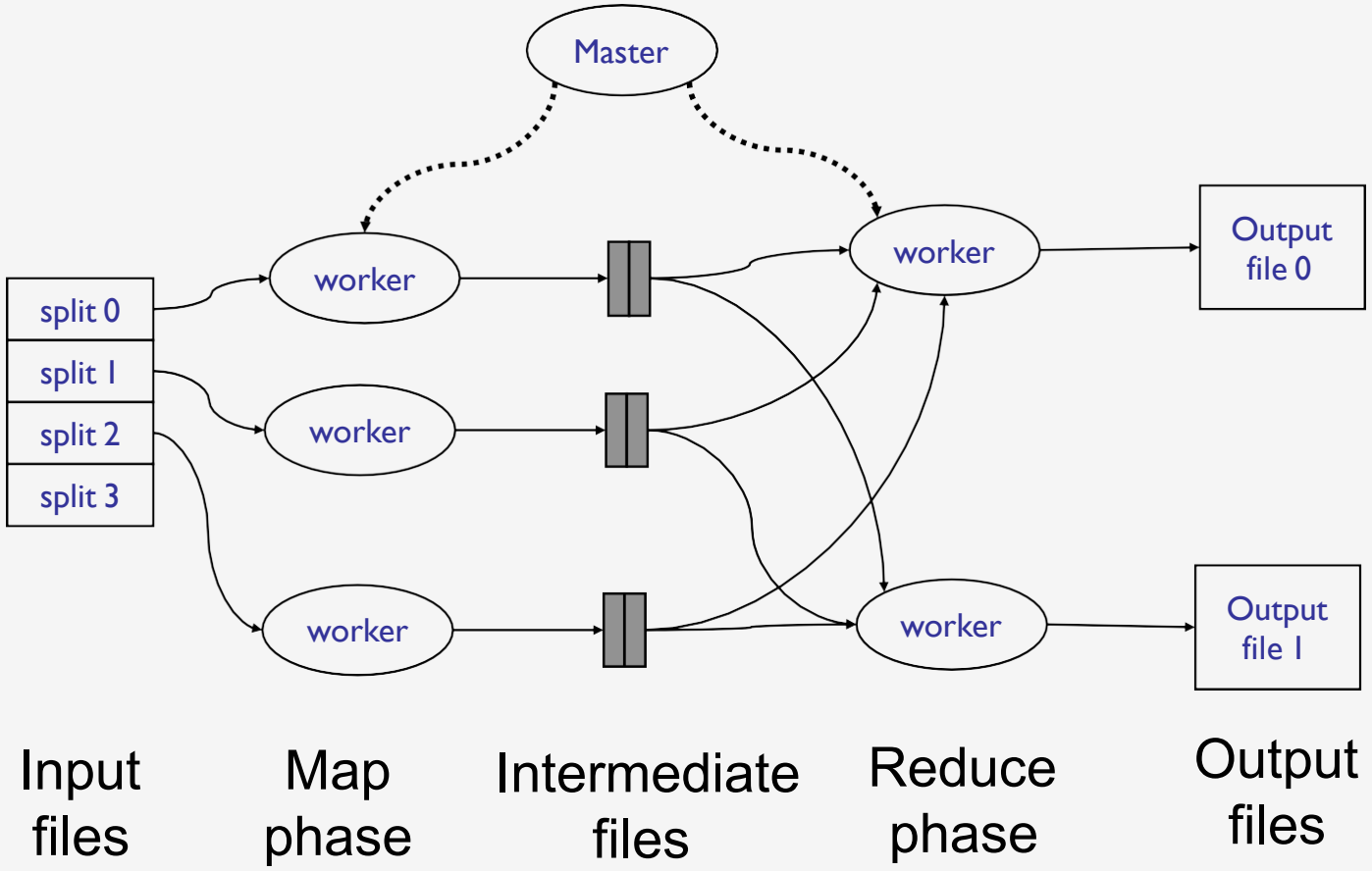
- Many of these algorithms can be done by applying a sequence of Map en Reduce function applications:

```
map(k1,v1) → list(k2,v2)

reduce(k2,list(v2)) → list(v2)
```

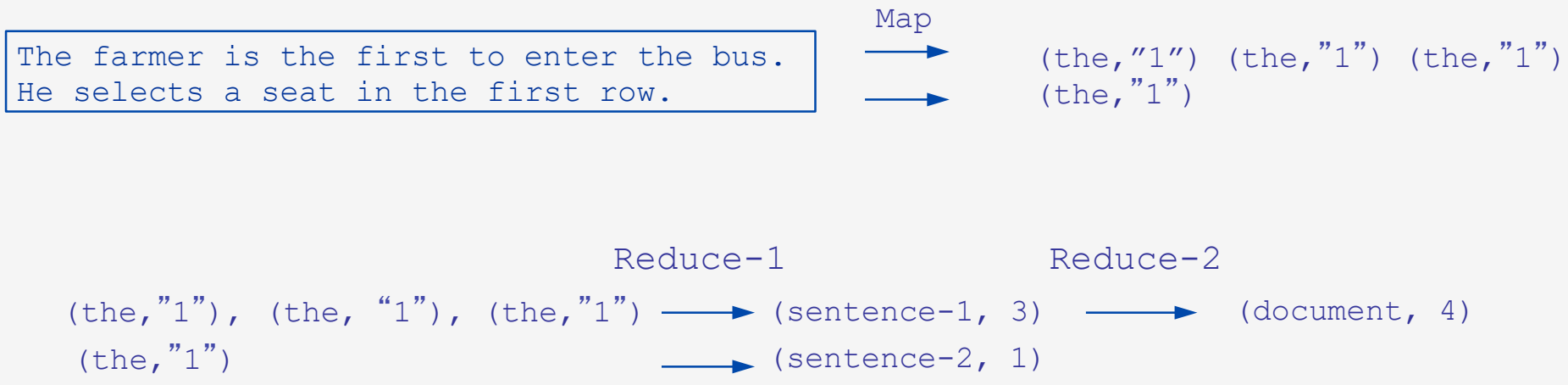
[Dean&Ghemawat, MapReduce: simplified data processing on large cluster, OSDI 2004]

HPC for Web Applications (3/4): Execution Flow



HPC for Web Applications (4/4): Example

k1="the", v1=<words in document/sentences>



Take home message

High Performance Computing is required to make computation- and/or data- intensive problems feasible in time

Modern HPC relies on:

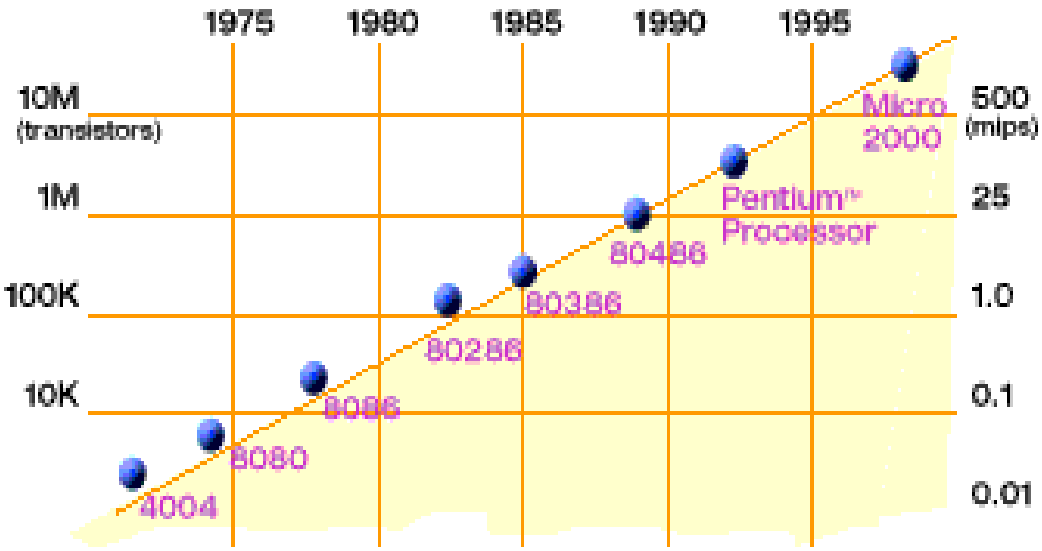
- *Supercomputers*
- *Parallelism*
- *Models and simulations*

2. Developments in Technology

Why ~~powerful~~ *all* computers are parallel (2007)

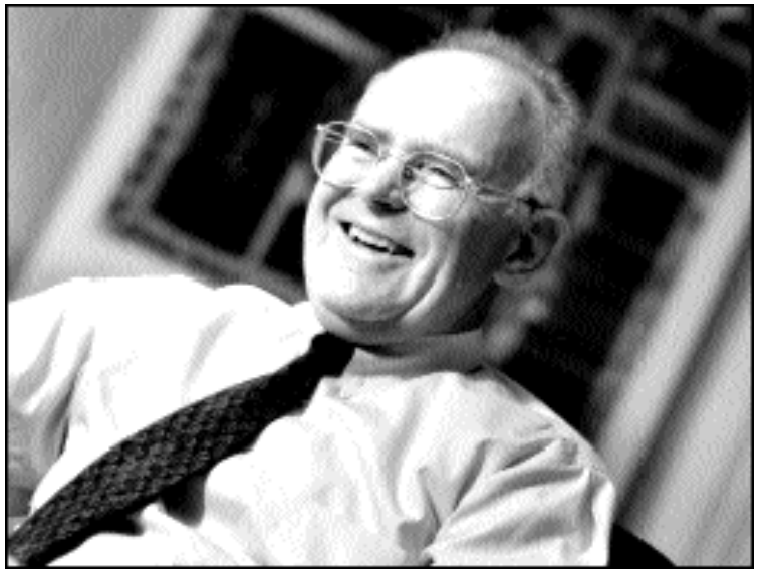
circa 1991-2006

Technology Trends: Microprocessor Capacity



2X transistors/Chip Every 1.5 years
Called "Moore's Law"

Microprocessors have become smaller, denser, and more powerful.

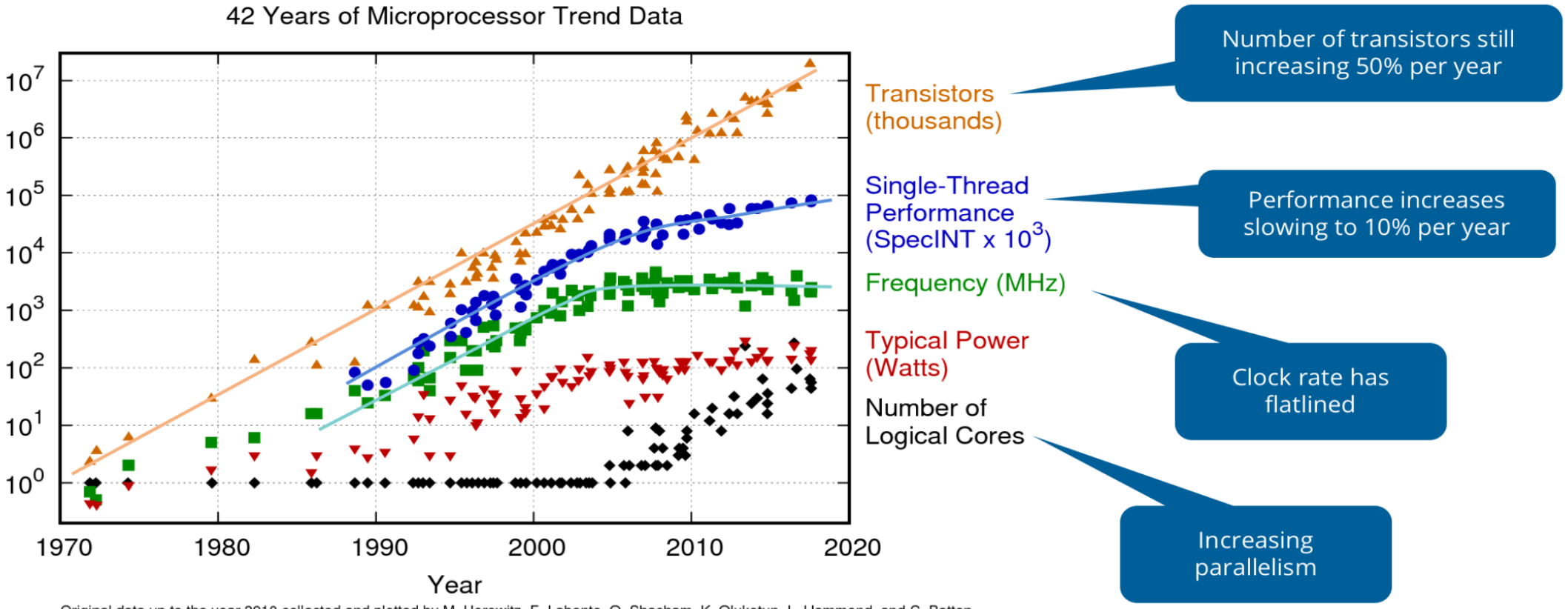


Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Slide source: Jack Dongarra

Power (heat dissipation) problems

- Higher clock speeds will have less gain
- Higher clock speeds lead to more Watts



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

Moore's Law reinterpreted

- Number of cores per chip can double every two years
- Clock speed will not increase (possibly decrease)

Parallel processing instead

- ❖ Need to deal with systems with millions of concurrent threads
- ❖ Need to deal with inter-chip parallelism as well as intra-chip parallelism

Heterogeneous solutions

- Graphics and Game processors
  - *Graphics Processing Units (GPUs), e.g., NVIDIA and ATI/AMD*
  - *Game processors, e.g., Cell for PS3*
  - *Parallel processor attached to main processor (APU)*
  - *Originally special purpose, getting more general*
  - *Programming model not yet mature*
- FPGAs – Field Programmable Gate Arrays
  - *Inefficient use of chip area*
  - *More efficient than multicore for some domains*
  - *Programming challenge now includes hardware design, e.g., layout*
- Tensor processing or other NeuralNet' units for machine learning

Trends in Hardware Performance

- TOP500 Project: listing the 500 most powerful computers in the world
- R_{max} of LINPACK (in Tflops/s)
  - *solve a linear system Ax=b (dense problem, matrix A is random)*
  - *dominated by dense matrix-matrix multiply (one of the fastest computations usually possible on a processor)*
  - *HPCG benchmark meusures performance in CG iterative solution*
- Updated twice a year (June/Nov)
- All information available from the TOP500 web site at: www.top500.org

The list of top 500 fastest computers of the world (June 2021)

http://www.top500.org

#1. Fujitsu, 442 PFlop/s on Linpack (HPL), peak 537 PFlop/s, processor A64FX 48C 2.2GHz, 7630848 cores. Tofu interconnect D.

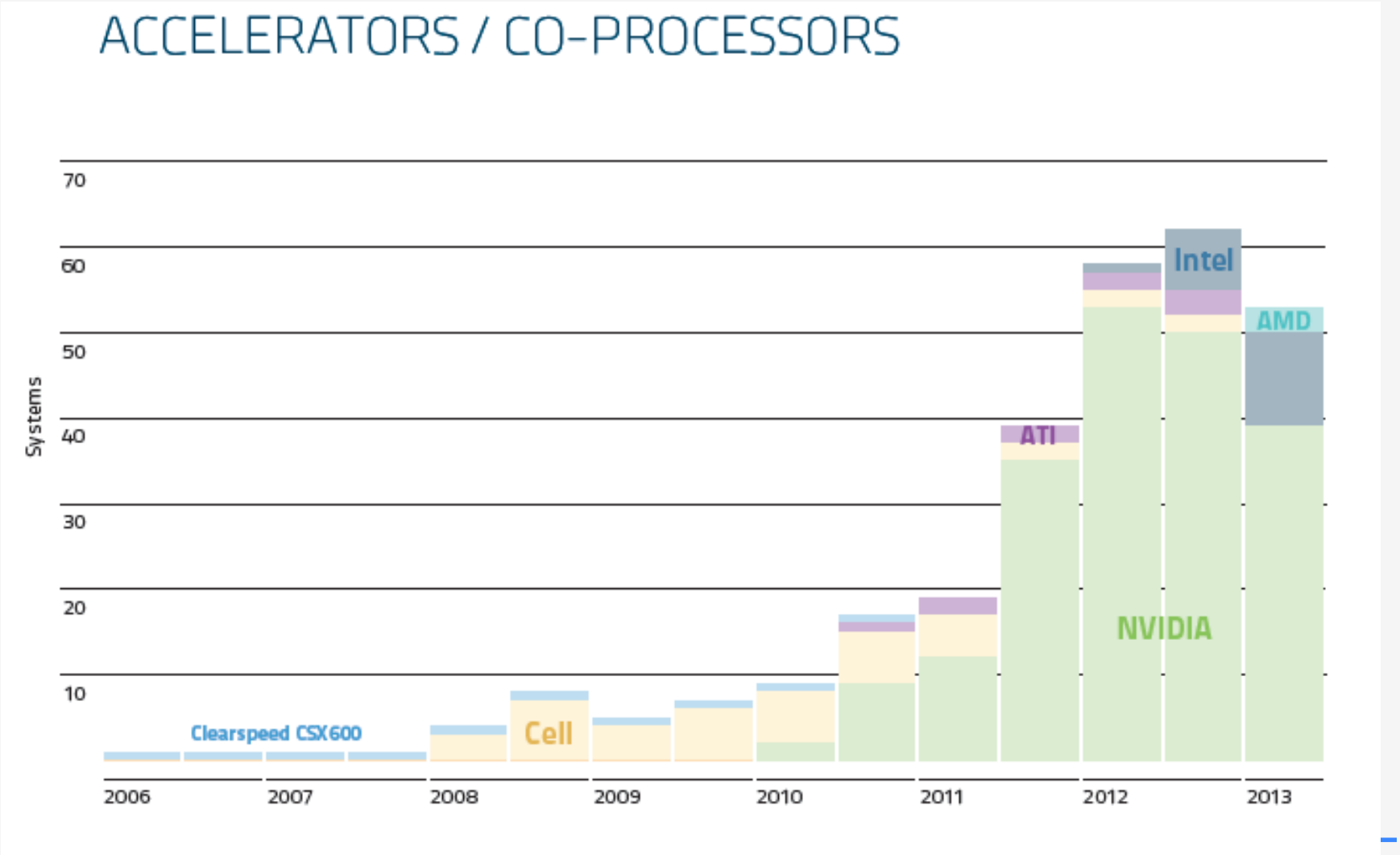
#2 **Summit**, 148.6 Pflops on Linpack (HPL), peak 200.8 Pflops, 4,356 nodes, each with two 22-core Power9 CPUs, and six NVIDIA Tesla V100 GPUs. 2,414,592 cores. Nodes are linked together with a Mellanox dual-rail EDR InfiniBand network.

#3 Sierra, 71.6 Pflops on HPL, peak 125.7 Pflops. 4,320 nodes each with two Power9 CPUs plus four NVIDIA Tesla V100 GPUs, 1,572,480 cores. Dual-rail Mellanox EDR InfiniBand as the system interconnect.

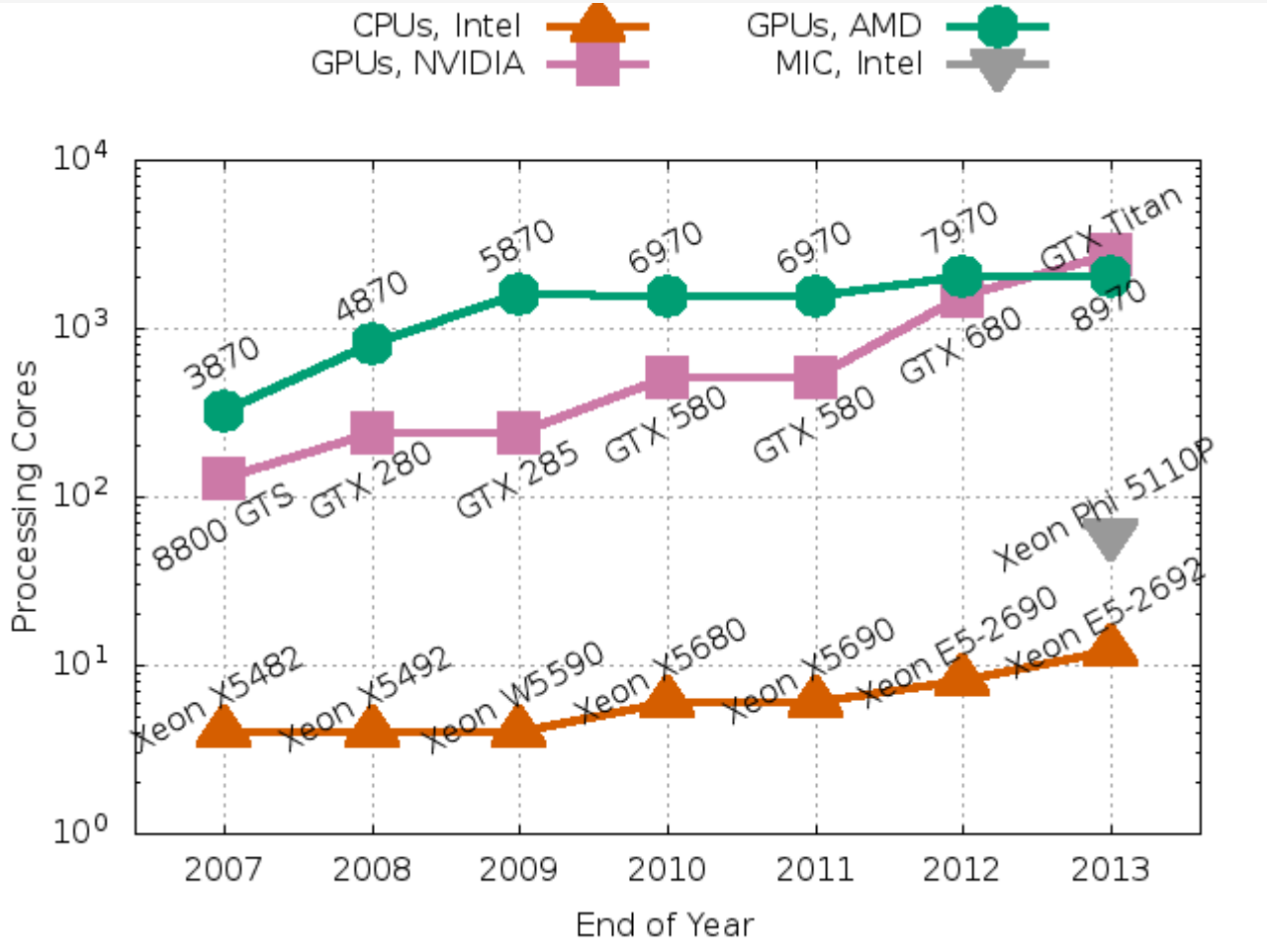
#4 Shenwei Taihu Light 93 Pflops Linpack, peak124.5 Pflops, 40960 nodes (SW26010 processors) each of 260 cores, total 10.65 million cores). Special interconnect chip set for HPC network.

#**5 Perlmutter** – 64.6 Pflops on HPL, peak 89.8 Pflops, HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, 706,304 cores, 1536 NVIDIA A100 accelerated nodes.

TOP500: Accelerators



Many-core processors are main stream in HPC



Take home message

- HPC today relies on powerful clusters and supercomputers
- Technology-wise major advances in using multi-core processors and accelerators
- Consequently, hardware parallelism has increased
- All machines, supercomputers, nodes of clusters, desktops, mobiles .... are or will be parallel
- Parallel computing is the key to HPC
  - for both performance **and** energy consumption

3. Principles of parallel computing

Terminology

- A parallel application has multiple computational units, called tasks, that run concurrently (i.e., in parallel)

- An application can have multiple layers of parallelism, which are typically nested
  Example: a task can be further parallelized in multiple subtasks, which also run concurrently

- Parallel applications run on parallel hardware by mapping tasks (i.e., processes) to processors

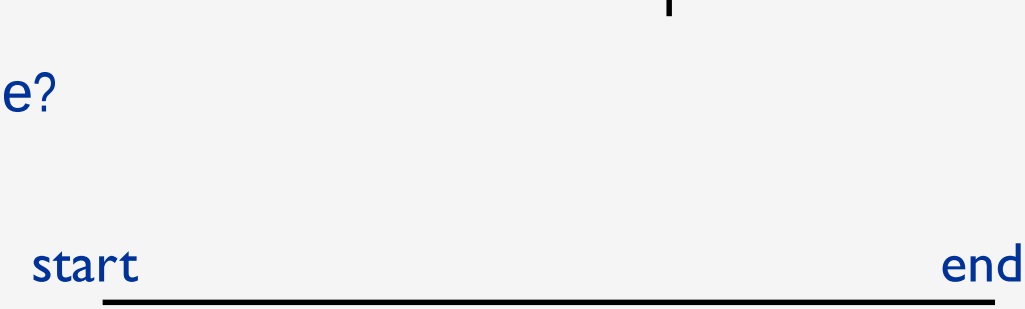
Towards parallel computing

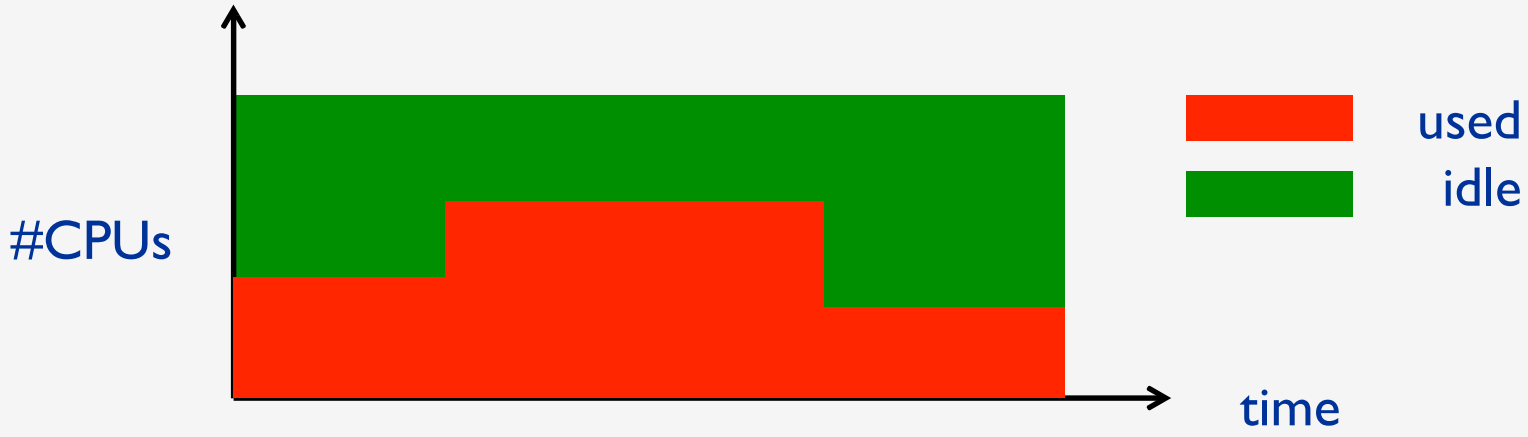
- What are suitable performance metrics?
- What is the appropriate parallelism granularity (~ task size)?
- How to benefit from data locality?
- How to map, coordinate, and synchronize tasks?
- How to scheduling and load balance applications?

Parallel programming is much more difficult than sequential programming.

Performance metrics

- What should we measure?
- Execution time
  - absolute or relative ?
- Speed-up
  - relative to what?
- Utilization (fraction of capacity (time x #CPU) used)





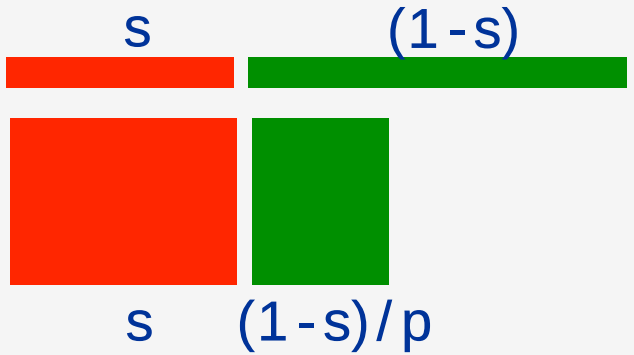
“Invisible” parallelism: exist on every computer

- Bit level parallelism – by **compiler**
  - *e.g., within floating point operations*
- Instruction level parallelism (ILP) – **by compiler/HW**
  - *multiple instructions execute per clock cycle*
- Memory system parallelism – **by compiler/HW**
  - *prefetch, overlap of memory operations with computation*
- Job parallelism – **by OS**
  - *multiple jobs are run in parallel by the OS*

At the parallel application level (parallel tasks and their management), programmers are on their own!

Enough parallelism: 1st approximation

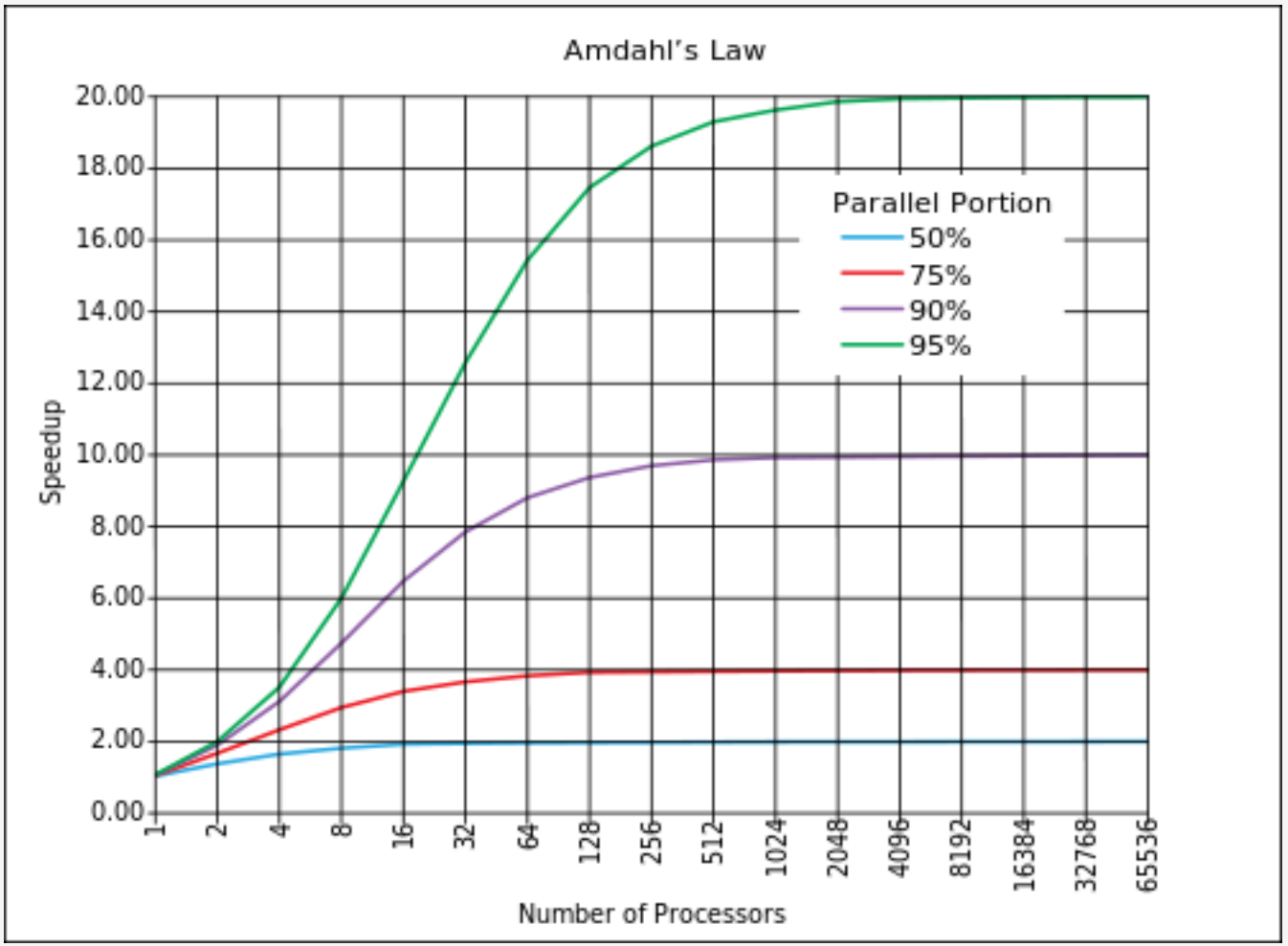
- Every application has an intrinsically sequential part
- Amdahl's law:
  - *let s be the fraction of work that is sequential, then (1-s) is the fraction that is parallelizable*
  - *p = number of processors*
  - *S = Speedup*



$$S = T_{seq}/T_{par}$$
$$= 1/(s + (1-s)/p)$$
$$\leq 1/s$$

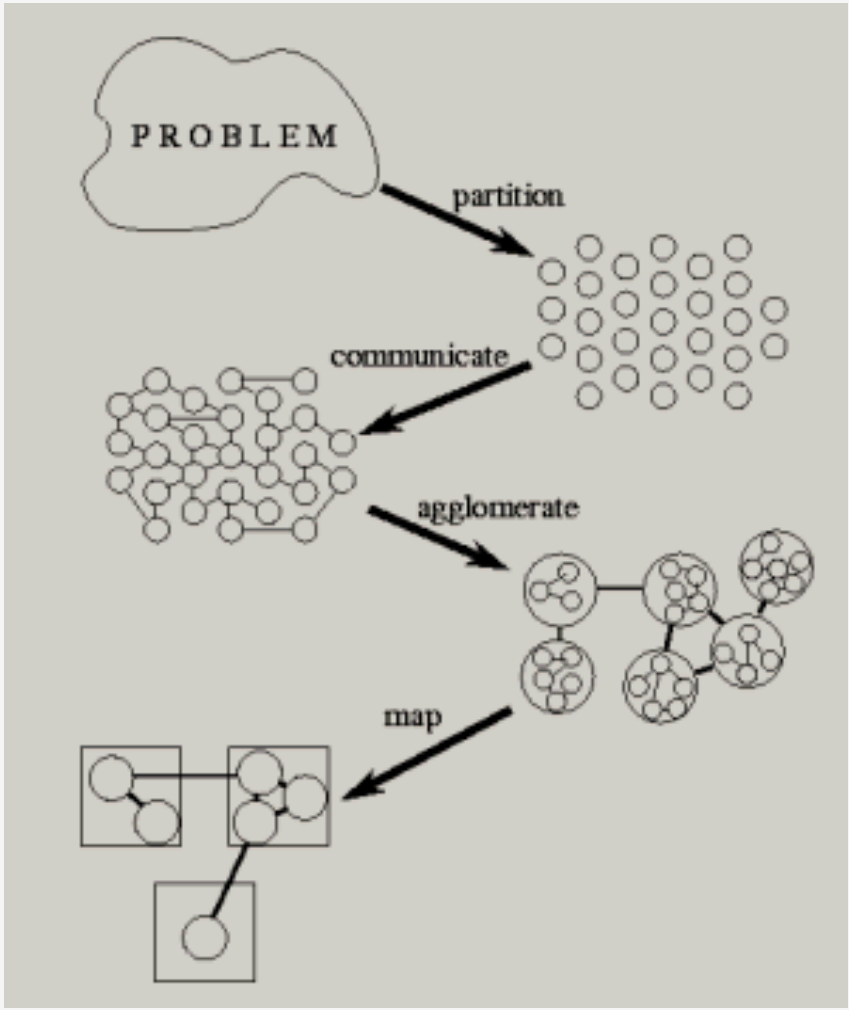
Speedup is bounded by the sequential fraction.

Amdhal's Law in a picture



Granularity, mapping

- Partitions = tasks
  - *right size for mapping*
- Communication =>
  - *overhead*
- Data locality =>
  - *agglomerate*
- Mapping =>
  minimize overheads and allow for good load balancing



Beware: agglomerate doesn't always work,
Larger granularity ≠ better data locality

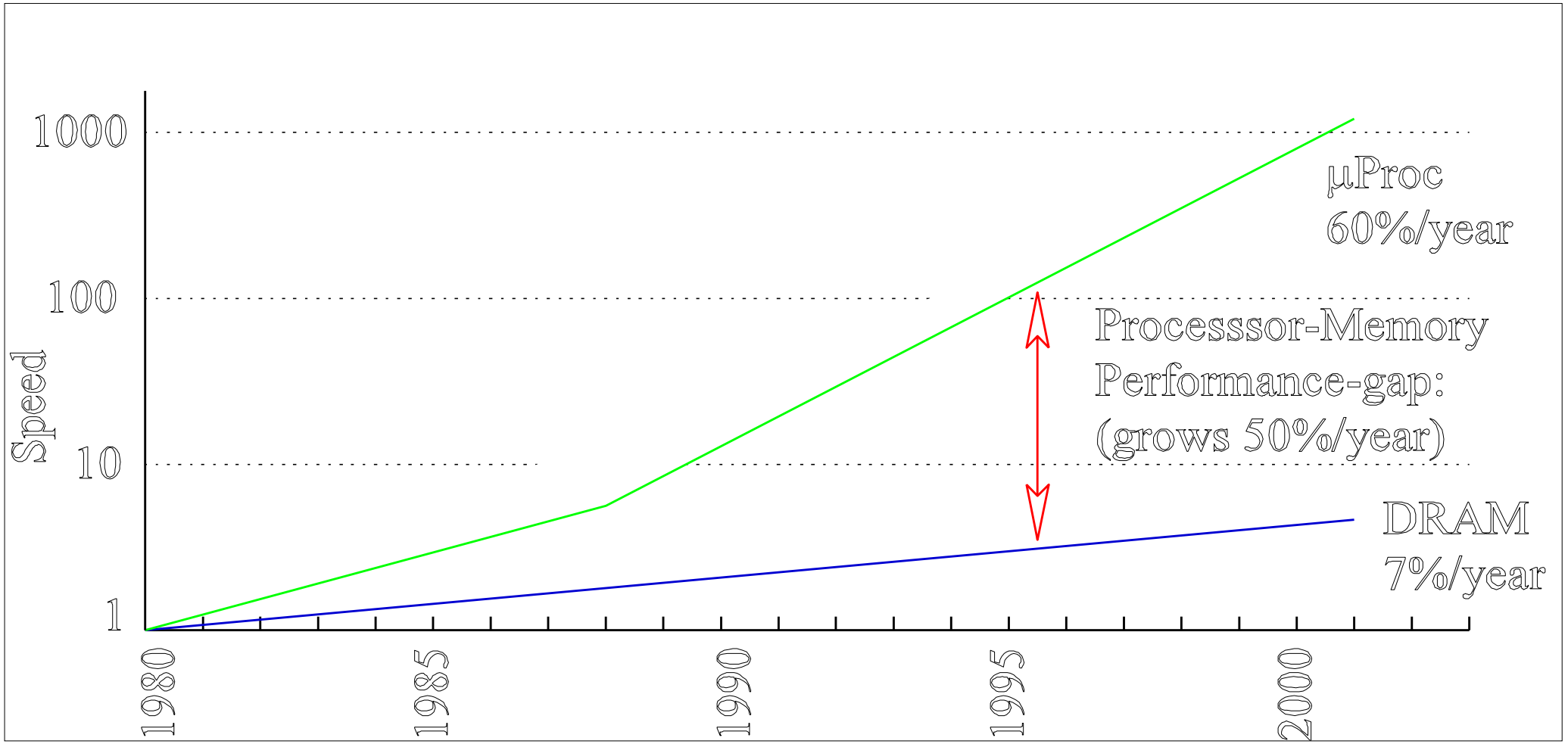
Data locality

- The further the data is from the computation, the slower the computation becomes
- Algorithm should do most work on local data
  - *requires smart data distribution*
- When locality is not possible, use latency hiding techniques
  - *caching and prefetching*
  - *overlapping computation with data transfers*

$$data\ locality\ ratio = \frac{computing\ time}{mem.access \text{ or } communication\ time}$$

Increase data locality by using a data element many times after it is read into cache or communicated from remote (slow) memory.

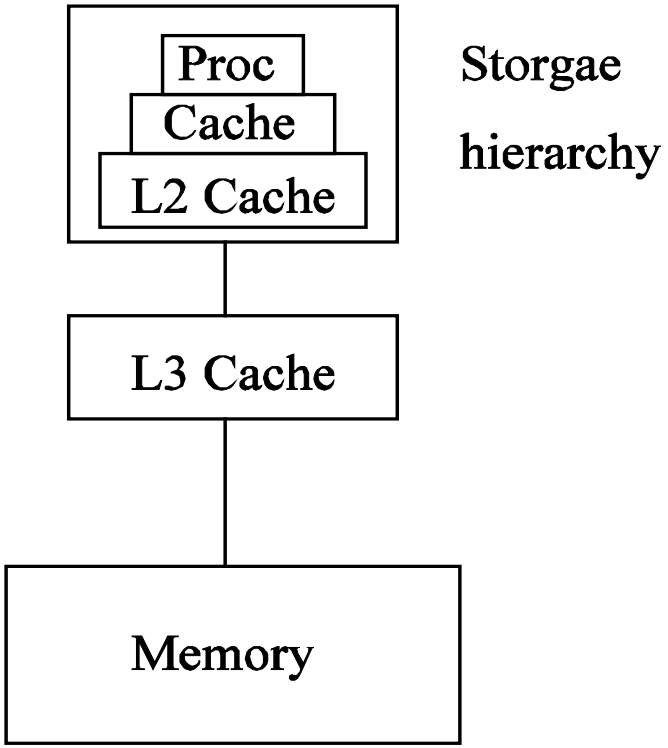
Inside one CPU:
Big gap in speed between processor and memory



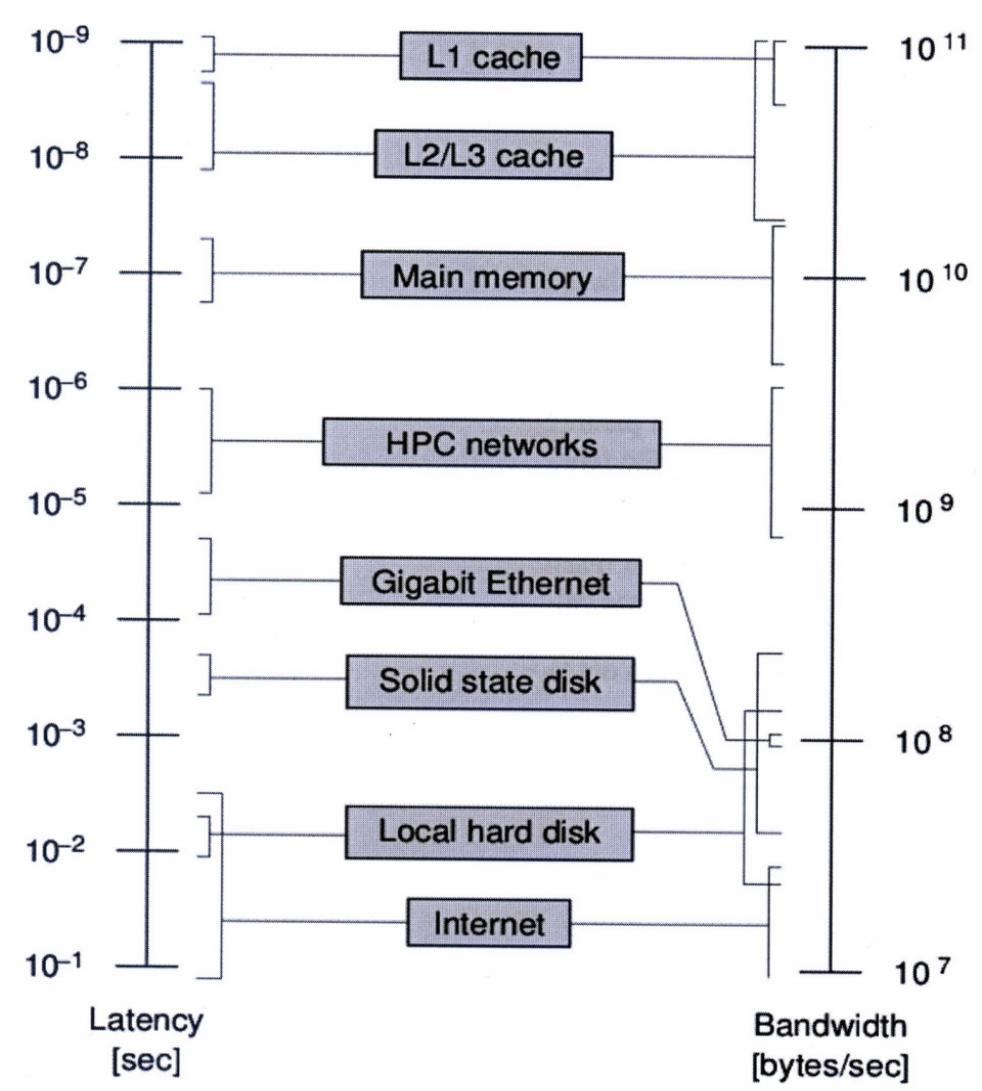
Memory hierarchy: NUMA

In order reduce the big delay of directly accessing the main or remote memory, it requires:

- Optimizing the data movement, maximize reuse of data already in fastest memory;
- Minimizing data movement between ‘remote’ memories (communication)



Across processors and CPUs:
Speed differences in memory and networks



Challenges

- **Hardware**: processor technology and interconnection network;
- **Software**: Parallel compilers; Grid computing environment (e.g., Condor, Globus, .net, etc.)
- **Methods and Algorithms**: Only computations with large degree of parallelism can efficiently use parallel and distributed computers. (Another important factor is algorithmic efficiency: High performance algorithms are of the same importance as high performance computers!)

Hunger for more computing power

Tremendous increase in speed up to now:

- Clock speed: 10^6
- Parallel processing: 10^6
- Efficient algorithms: 10^6

Computational scientists and engineers demand for ever more computing power

Software and Tools Challenges

Current situation: OpenMP works for SMP systems with a small number of processors/cores. For large systems and distributed memory systems, data distribution/communication must be done manually by the programmer, mostly with MPI.

Programming GPU type of accelerators using CUDA, OpenCL etc. has sort resemblance of programming vector processors in the old days. Very high performance for certain type of operations, but programmability and applicability are some what limited.

Programming difficulty is getting severer

In contrast to the fast development in hardware, the development of parallel compiler and programming lack behind.

Moving towards larger and larger systems enlarges this problem even further

- Heterogeneity
- Debugging
- …

Applications

Applications that require Exaflops computing power, examples ([4],[6]):

- Climate and atmospheric modelling
- Astrophysics
- Energy research (e.g., combustion and fusion)
- Biology (genetics, molecular dynamics, …)
- …

Are there applications which can use 1 million processors?

- Parallelism is inherent in nature
- Serialization is a way we deal with complexity
- Some mathematical and computational models may have to be reconsidered

Algorithms

Algorithms with a large degree of parallelism is essential

Data locality is important to efficiency

- Data movement at the cache level(s)
- Data movement (communication) between processors/nodes

Algorithms: Load imbalance

- Workload is imbalanced between processors
  - *less parallelism than processors*
  - *differently sized tasks*
- Can all applications be load-balanced?
  - *adapting to "interesting parts of a domain"*
  - *tree-structured computations*
  - *fundamentally unstructured problems*
- Consequences:
  - *idle resources*
  - *poor hardware utilization*
  - *poor performance*