MSc Master Course – High Performance Computing

Introduction into HPC – General overview Applications, Technology, Memory Bandwidth and Locality

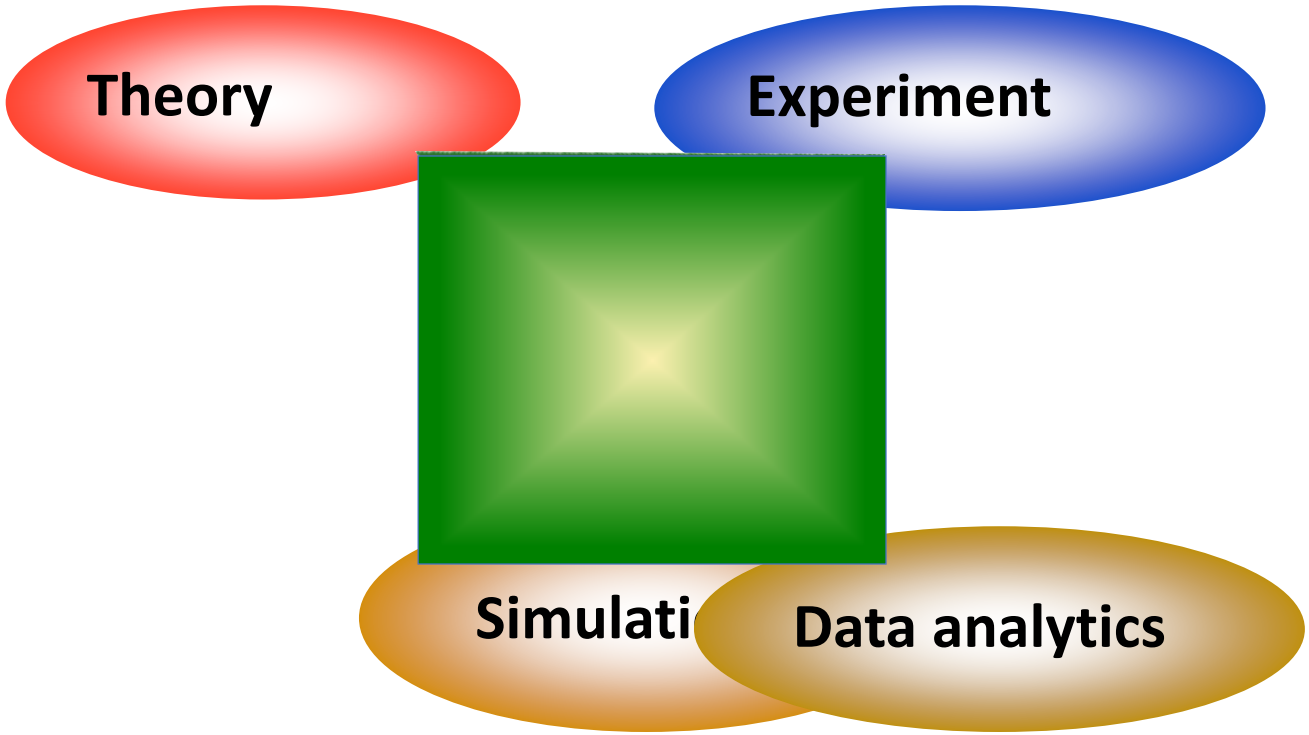
Olaf Schenk

Institute of Computational Science
USI Lugano
September 19, 2018

Outline today

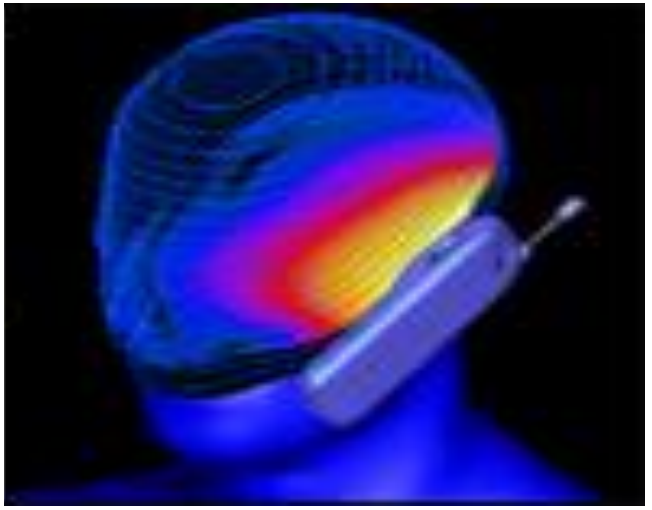
- Application of parallel architectures
- Hardware trends
- TOP 500 list
- A short introduction into parallel programming and high-performance computing
- Memory locality and bandwidth

The Fourth Paradigm of Science



HPC Applications – Simulations and why they are so important

- Simulations are replacements for experiments that are
  - **too difficult** (build large wind tunnels)
  - **too expensive** (build a throw-away passenger jet).
  - **too slow** (wait for climate evolution)
- **Optimization of systems** e.g. to simulate or predict the perfect shape of a sailing boot.
- Help to **understand** complex systems in science and engineering.



Simulation of a 3D electromagnetic field (Cooperation with Computer Simulation Technology, Germany)

The progress in HPC architectures, HPC algorithmic and HPC software technologies allows to simulate systems that might have been considered impractically large until recently.

HPC Applications – Simulations and architectures

- Many scientific or business problems require large peak performance and/or are still too challenging for current HPC architectures e.g.
  - **Seismic earth sciences** Simulate or predict the geophysical structure of the earth e.g. to exploit oil reservoirs.
  - **Climate modeling** Solve the coupled ocean-atmosphere system over the next 50 years to understand the green house effect.
  - **Web search engines** Goggle´s giant server facilities with over >millions of compute nodes

Typical Performance	Time	Architecture	Memory
1 **Gflops**	3,3 month	Intel Xeon	Rule of Thumb
10 **Gflops**	10 days	16 core AMD server	1 GByte
100 **Gflops**	2,4 h	HPC architecture	for
1 **Tflops**	14 minutes	Cluster	1 Gflop
1 **Pflops**	1 s	Supercomputer	

Units of Measure in HPC

- High Performance Computing (HPC) units are:
  - Flop: floating point operation
  - Flop/s: floating point operations per second
  - Bytes: size of data (double precision floating point number is 8)
- Typical sizes are millions, billions, trillions…

Mega	**Mflop/s** = 10^6 flop/sec	**Mbyte** = 10^6 byte (also 2^{20} = 1048576)
Giga	**Gflop/s** = 10^9 flop/sec	**Gbyte** = 10^9 byte (also 2^{30} = 1073741824)
Tera	**Tflop/s** = 10^{12} flop/sec	**Tbyte** = 10^{12} byte (also 2^{40} = 10995211627776)
Peta	**Pflop/s** = 10^{15} flop/sec	**Pbyte** = 10^{15} byte (also 2^{50} = 1125899906842624)
Exa	**Eflop/s** = 10^{18} flop/sec	**Ebyte** = 10^{18} byte

Units of Measure in HPC

- Flops: floating point operations

  PI = 3.14159265358979622463433 = 3.14159265358979622463

- Flop/s: floating point operations per second

  How many of these flops can we compute per second on my laptop?

- What kind of information do we need?

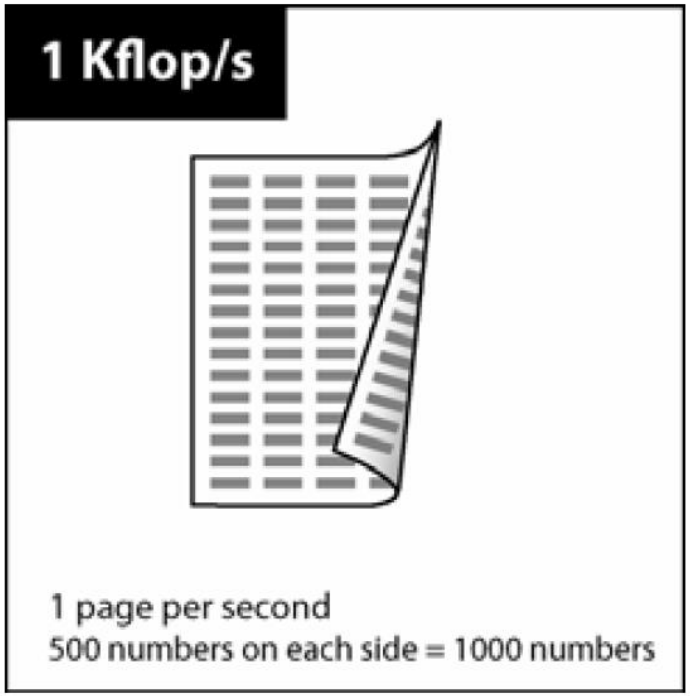
Intel Core i7/1.8 GHz, 8 cores, each core can perform one multiplication and one addition per cycle (FMA)

Solution: 1.8 GHz * 8 cores * 2FMA = 28.8 Gflops/s

In the printed version, the solutions can be found in the appendix

Units of Measure in HPC

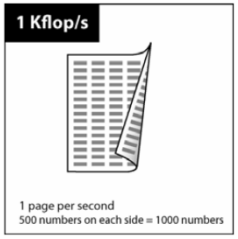
- **Let us say you can print:**
  - 5 columns of 100 number each; on both sides of the page = 1000 numbers (Kflop) in one second **(1 Kflop/s)**



Source: Jack Dongarra, CS Department, University of Tennessee

Units of Measure in HPC

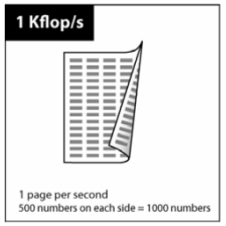
- **Let us say you can print:**
  - 5 columns of 100 number each; on both sides of the page = 1000 numbers (Kflop) in one second **(1 Kflop/s)**



  - 1000 pages about 10 cm = 10^6 numbers (Mflop
    2 reams of paper per seconds **(1 Mflop/s)**

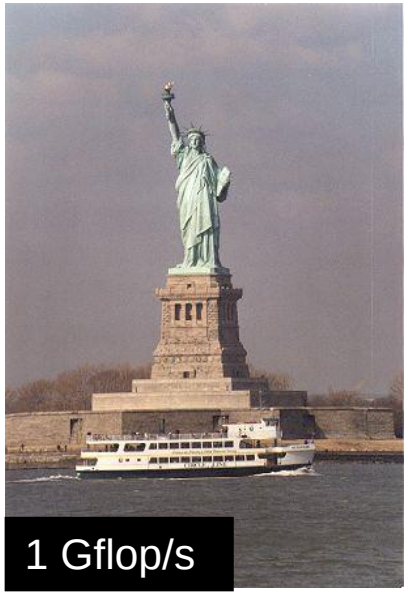
Units of Measure in HPC

- **Let us say you can print:**
  - 5 columns of 100 number each; on both sides of the page = 1000 numbers (Kflop) in one second **(1 Kflop/s)**



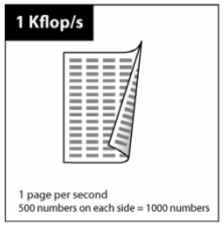
  - 1000 pages about 10 cm = 106 numbers (Mflop
    2 reams of paper per seconds **(1 Mflop/s)**

  - 10^9 numbers (Gflop) = 10000 cm = 100 m stack
    Height of Statue of Liberty printed per second **(1 Gflop/s)**



Units of Measure in HPC

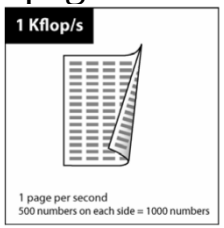
- **Let us say you can print:**
  - 5 columns of 100 number each; on both sides of the page = 1000 numbers (Kflop) in one second **(1 Kflop/s)**



  - 1000 pages about 10 cm = 106 numbers (Mflop
    2 reams of paper per seconds **(1 Mflop/s)**
  - 10^9 numbers (Gflop) = 10000 cm = 100 m stack
    Height of Statue of Liberty printed per second **(1 Gflop/s)**
  - 10^{12} numbers (Tflop) = 100 km stack: Altitude achieved by SpaceShipOne (printed per second) **(1Tflop/s)**

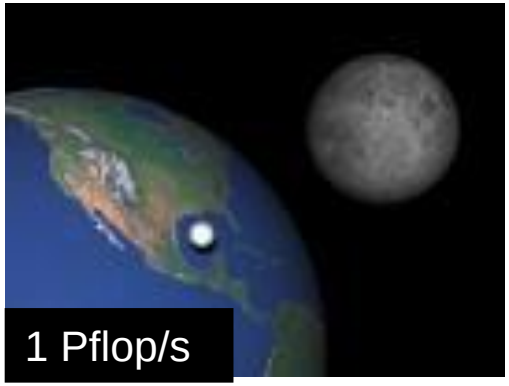
Units of Measure in HPC

- **Let us say you can print:**
  - 5 columns of 100 number each; on both sides of the page = 1000 numbers (Kflop) in one second **(1 Kflop/s)**



  - 1000 pages about 10 cm = 106 numbers (Mflop
    2 reams of paper per seconds **(1 Mflop/s)**
  - 10^9 numbers (Gflop) = 10000 cm = 100 m stack
    Height of Statue of Liberty printed per second **(1 Gflop/s)**

  - 10^{12} numbers (Tflop) = 100 km stack: Altitude achieved by SpaceShipOne (printed per second) **(1Tflop/s)**
  - 10^{15} numbers (Pflop) = 100,000 km (1/4 distance to the moon) stack printed per second **(1Pflop/s)**



HPC Applications - Global climate modeling

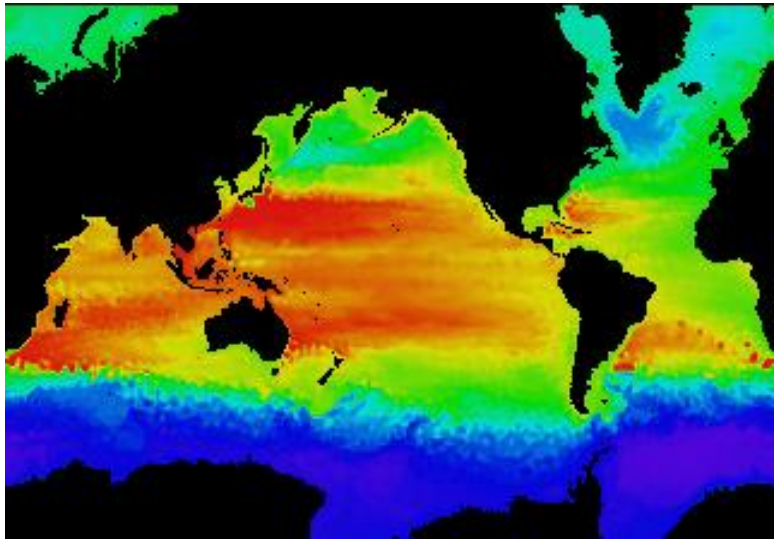
- **Climate funktion** has 4/6 Input-/Outputvariables:

  f(latitude, longitude, elevation, time) →

  temperature, pressure, humidity, wind velocity

- Method:
  - **Discretize** the domain, e.g., a measurement point every **1 km**
  - Devise an algorithm to predict weather at time **t+1** given **t**

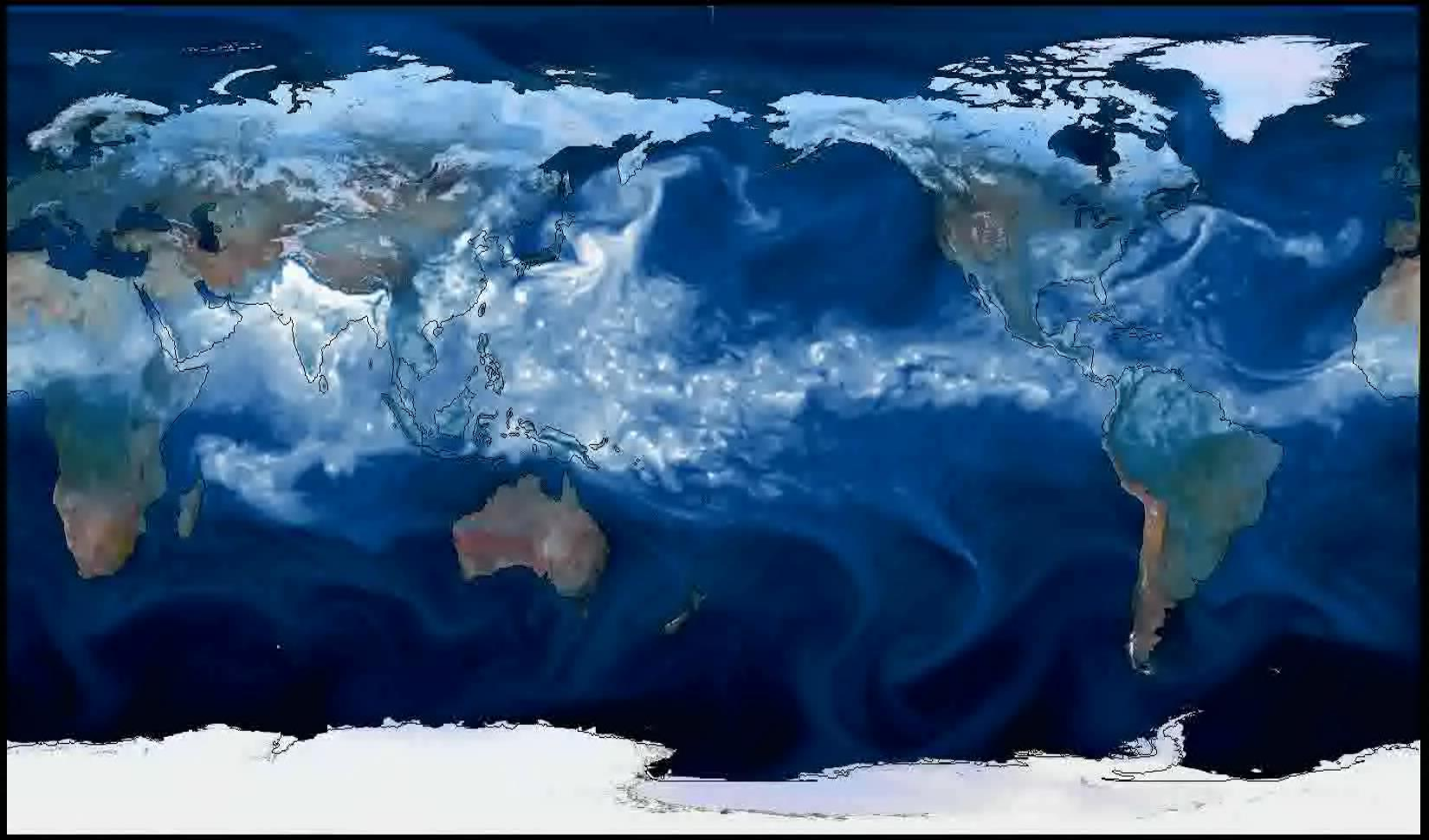
- Application:
  - Predict major events, e.g., El Nino
  - Use in setting air emissions standards



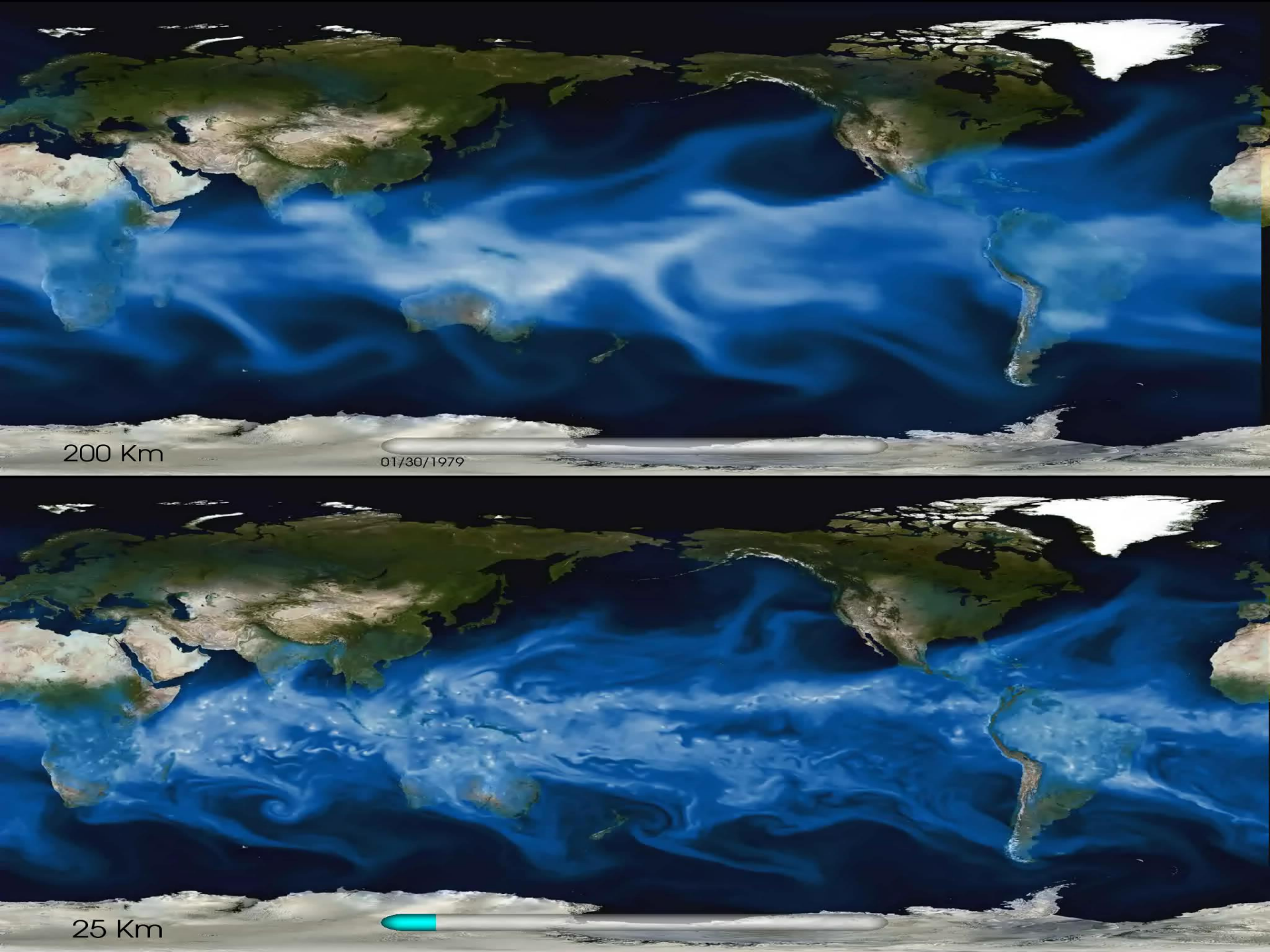
HPC Applications - Global climate modeling

- One piece is modeling the fluid flow in the atmosphere by the Navier-Stokes partial differential problem
  - Roughly **100 Flops** per grid point within 60 seconds simulation
  - Surface of the earth S = 4*PI*R*R (earth radius R = 6000 km).
  - 452 million grid points x 10 km atmosphaere -> 4.52 x 10^9 points
  - 100 flops per grid point -> 5x10^{11} flops
- Computational requirements:
  - To match real-time, need **500 Gflop** in 60 seconds (Intel processor gives 400 Mflop/s) → **8 Gflop/s** **20 PCs**
  - Weather prediction (7 days in 24 hours) → **56 Gflop/s** **140 PCs**
  - Climate prediction (50 years in 30 days) → **4.8 Tflop/s** **12.000 PCs**
  - To use in policy negotiations (50 years in 12 hours) → **288 Tflop/s** **720.000 PCs**
- **HPC architectures and efficient algorithms**
- To double the grid resolution, computation is at least 8x
- Current models are coarser than this (7km * 7km * 2km)

Simulations Show the Effects of Climate Change in Hurricanes



Michael Wehner and Prabhat, Berkeley Lab


200 Km
01/30/1979
25 Km

Some Challenging Parallel Computations

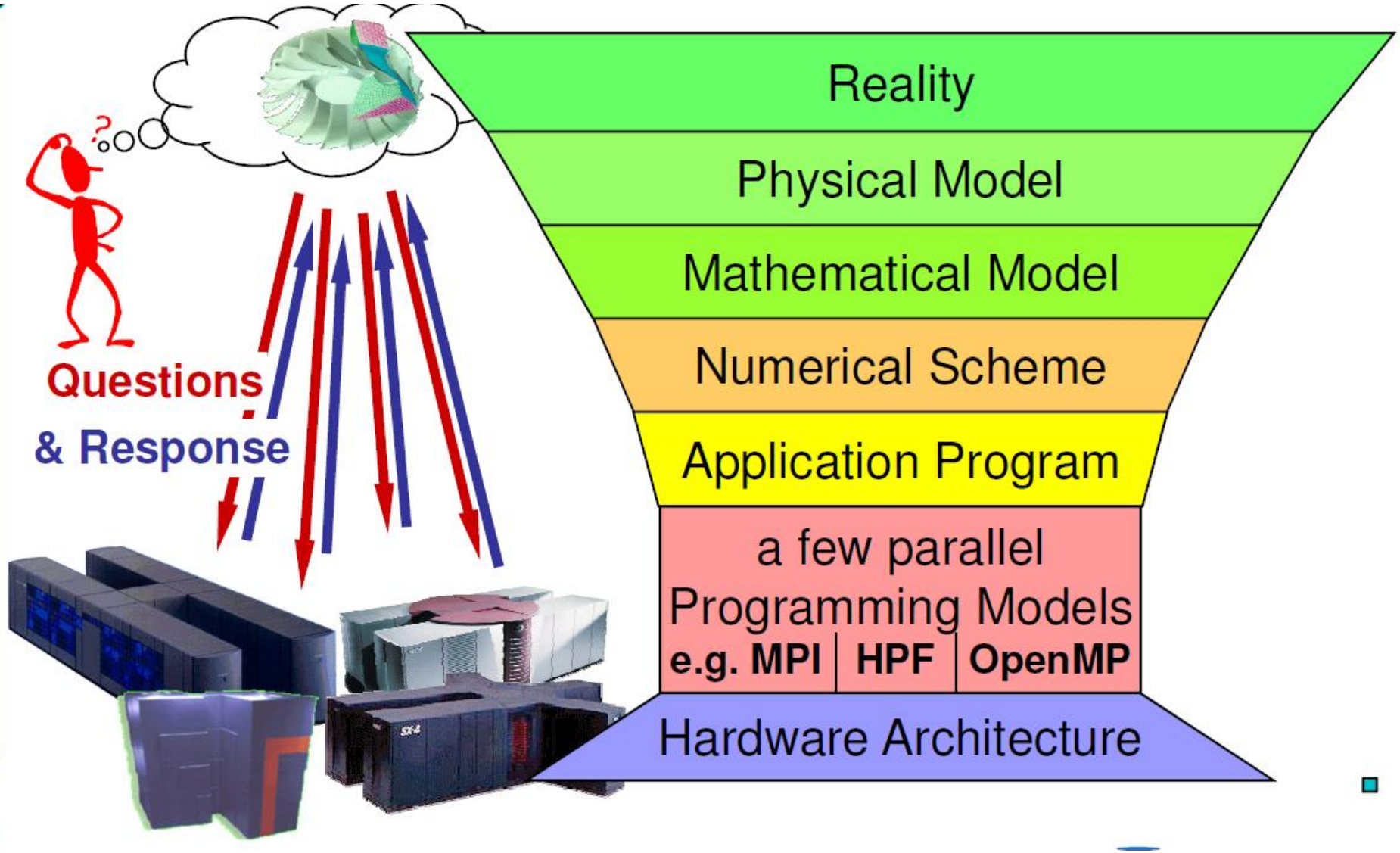
- **Science**
  - Global climate modeling
  - Astrophysical modeling
  - Biology: genomics; protein folding; drug design
  - Computational Chemistry
  - Computational Material Sciences and Nanosciences
- **Engineering**
  - Crash simulation
  - Semiconductor design
  - Earthquake and structural modeling
  - Computation fluid dynamics (airplane design)
  - Combustion (engine design)
- **Business**
  - Financial and economic modeling
  - Transaction processing, web services and search engines

Modeling and Simulation

- **Modeling and Simulation**
  - Build a mathematical model e.g based on partial differential equations
  - Primarily the problem of the user
- **Discrete approximation**
  - Replacement of the continuous model with a discrete model
  - Primarily the problem of numerical methods

- **Fast computation on HPC architectures (Computer Science)**
  - HPC architectures: Processors, Bandwidth, Locality
  - HPC information technology
  - Tools and Libraries.

Abstract Model



All (2007)

Why ~~are~~ powerful computers parallel?

Including your laptops and handhelds

Models of parallel computation

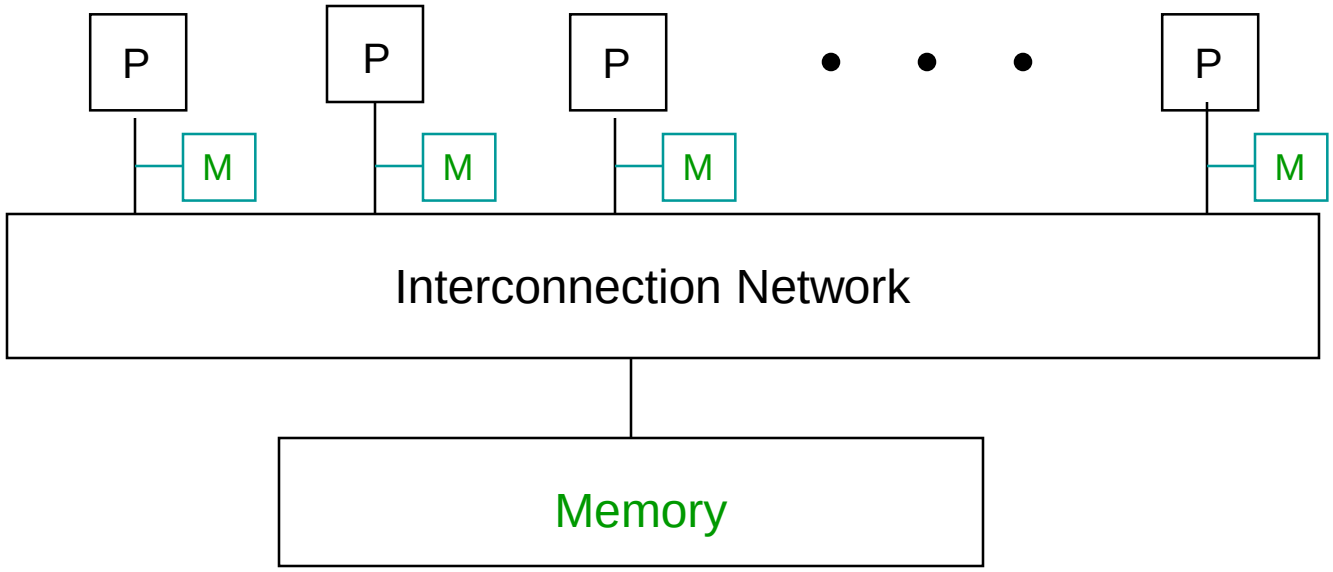
- Historically (1970s - early 1990s), each parallel machine was unique, along with its programming model and language

- Nowadays we separate the **programming model** from the underlying machine model.
  - 3 or 4 dominant programming models
  - This is still research – HPC software study is about comparing models

- Can now write portably **correct** code that runs on lots of machines.

- Writing portably **fast** code requires tuning for the architecture
  - Not always worth it – sometimes programmer time is more important
  - Challenge: design **algorithms** to make this tuning easy

A generic parallel architecture



- Where is the memory physically located?

Aspects of a parallel programming model

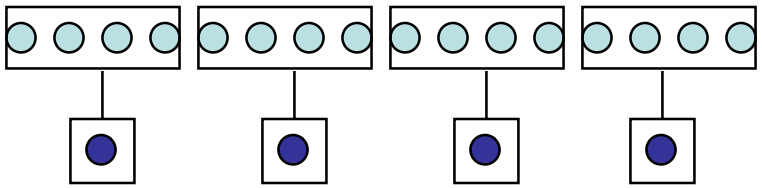
- Control
  - how is **parallelism** created
  - what **order** can operations happen in
  - how do different threads of control **synchronize**
- Naming
  - what data is **private** vs. **shared**
  - how shared data is **accessed** (or **communicated**)
- Operations
  - what are the basic operations
  - what operations are **atomic**
- Cost
  - how do we account for the cost of operations

Example: Norm of a vector

- Example: ∑ f(A[i]) from i=1 to i=n

$$\sum_{i=0}^{n-1} f(A[i])$$

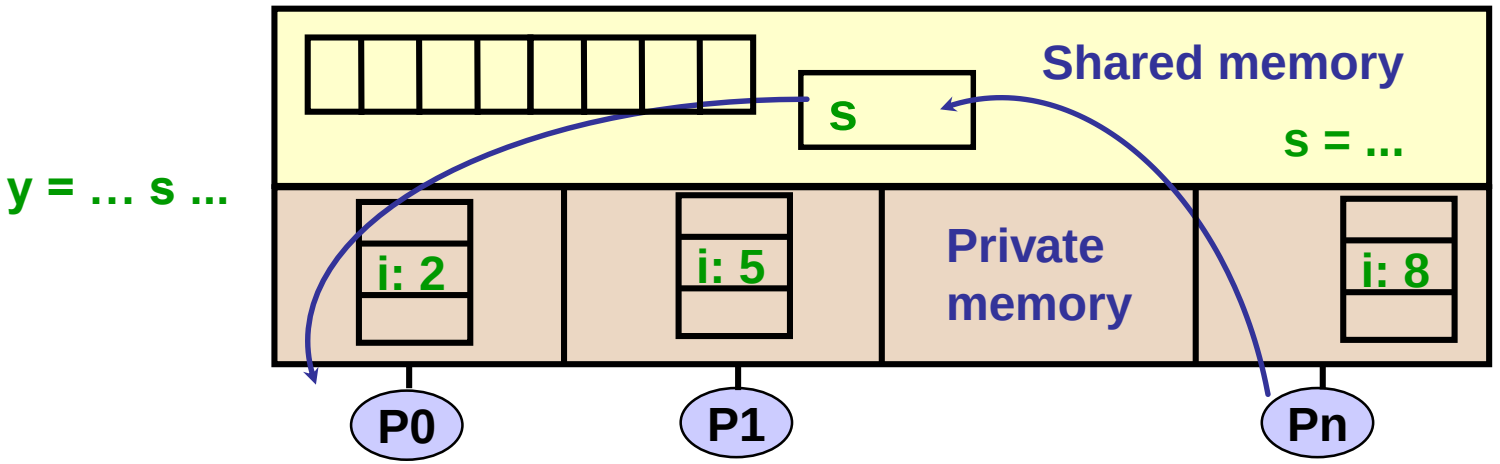
- **Parallel decomposition**:
  - Each evaluation of f and each partial sum is a task



- **Assign n/p numbers to each of p processes**
  - each computes independent "private" results and partial sums
  - one (or all) collects the p partial sums and computes the global sum
- **Two classes of data:**
  - **Shared**
    - the original n numbers, the global sum
  - **Private**
    - the individual function values
    - what about the individual partial sums?

Programming Model 1: Shared Memory

- Program is a collection of threads of control.
  - Can be created dynamically, mid-execution, in some languages
- Each thread has a set of **private variables**, e.g., local stack variables
- Also a set of **shared variables**, e.g., static variables, or global heap.
  - Threads communicate **implicitly** by writing and reading **shared** variables.
  - Threads coordinate by **synchronizing** on shared variables



Shared Memory Code for Computing a Sum

```
static int s = 27;
```

Thread 1		Thread 2	
….		…	
compute f([A[i]) and put in reg0	7	compute f([A[j]) and put in reg0	9
reg1 = s	27	reg1 = s	27
reg1 = reg1 + reg0	34	reg1 = reg1 + reg0	36
s = reg1	34	s = reg1	36
…		…	

- Assumption: s=27, f(A[i])=7 for thread1 and f(A[i])=9 for thread2
- For this program to work, s should be 43 at the end
  - but it may be 43, 34, or 36 ("**race condition**")
- Read and write operations are "atomic operations"
  - all computations are done in **private** registers of the processors.

Improved Shared-Memory Programm

```
static int s = 27;
static lock lk;
```

Thread 1

```
local_s1= 27
for i = 0, n/2-1
    local_s1 = local_s1 + f(A[i])
lock(lk);
s = s + local_s1
unlock(lk);
```

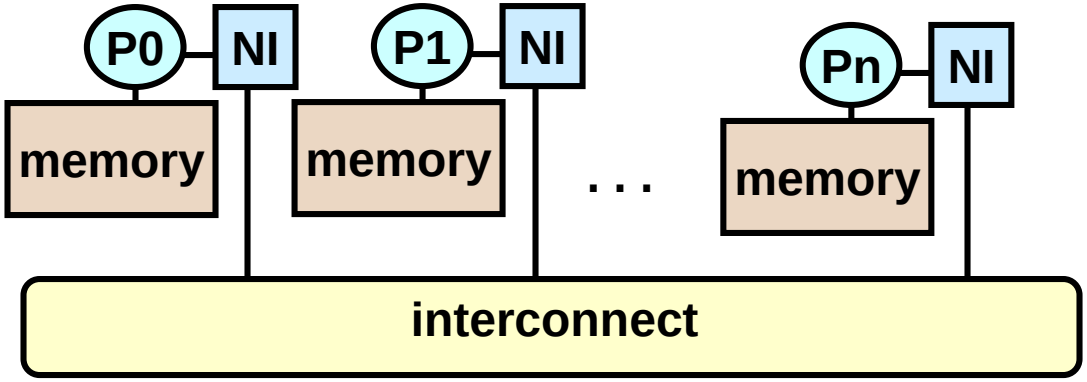
Thread 2

```
local_s2 = 27
for i = n/2, n-1
    local_s2= local_s2 + f(A[j])
lock(lk);
s = s +local_s2
unlock(lk);
```

- Most computation is on **private** variables
  - Sharing memory transfer is also reduced, which might improve speed
  - But there is still a **race condition** on the update of shared s
  - The race condition can be fixed by adding **locks**
  - Only one thread can hold a lock at a time; others wait for it

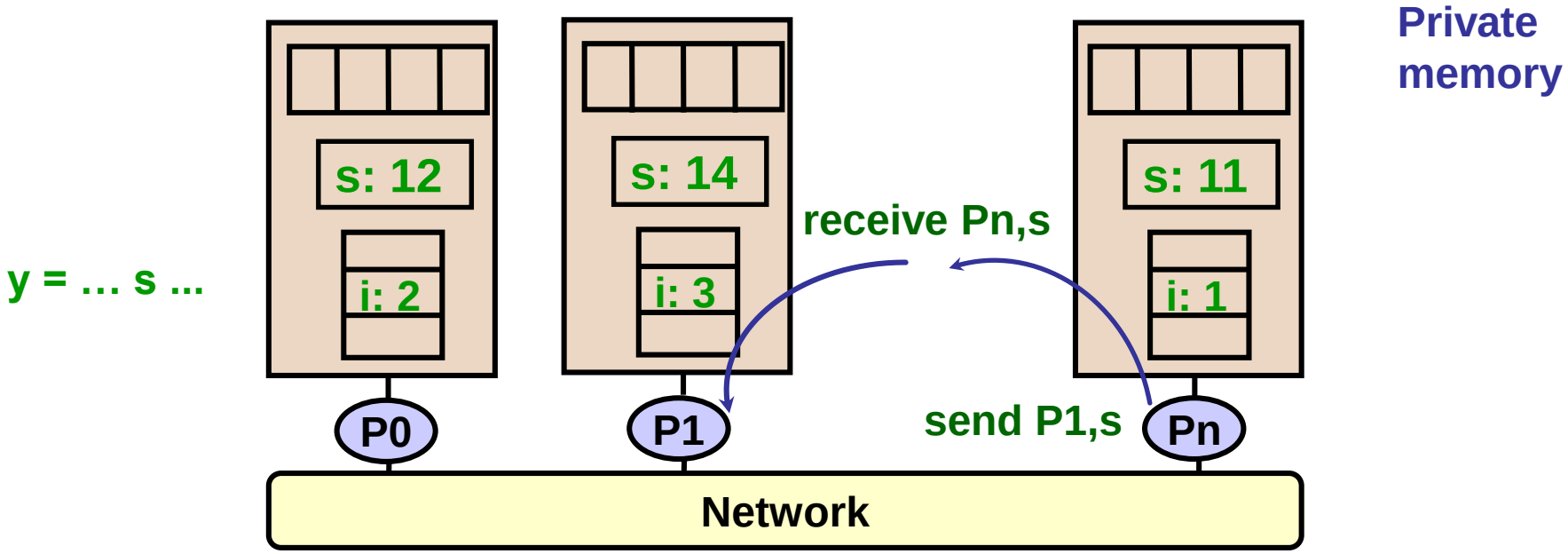
Machine Model 2: Distributed Memory

- Cray XC50, IBM BG/Q are **distributed memory** machines, but the nodes are SMPs.
- Each nodes (several cores) has its **own memory** and cache but cannot directly access another processor's memory.
- Each "node" has a network interface (NI) for all communication and synchronization.



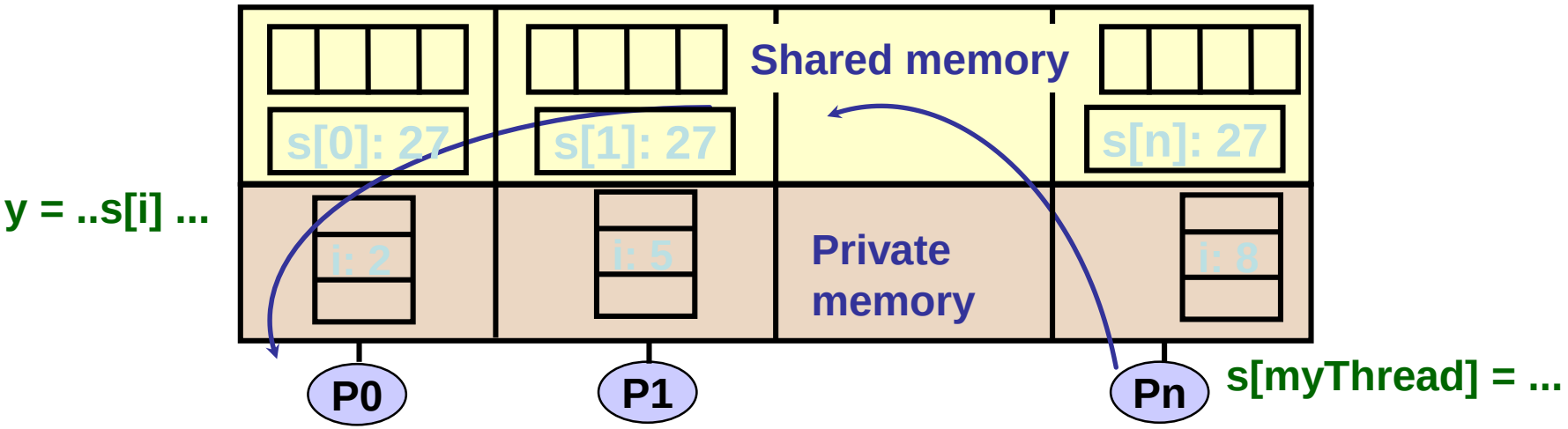
Programming Model 2a: Message Passing MPI

- Program consists of a collection of named processes.
  - Usually fixed at program startup time
  - Thread of control plus local address space – **no shared** data.
  - Logically shared data is partitioned over local processes.
- Processes communicate by explicit send/receive pairs
  - Coordination is implicit in every communication event.



Programming Model 2b: Global Address Space Languages

- Program consists of a collection of **named** threads.
  - Usually fixed at program startup time
  - Local and shared data, as in shared memory model
  - But, **shared data** is partitioned over **local processes**
  - Cost models says remote data is expensive
- Examples: Unified Parallel C (UPC), Chapel, X10
- In between message passing and shared memory



CSCS HPC architectures

- Selected resources at Swiss National Supercomputing Centre

	Cray XC40/XC50	**Cray XC40 KNL**
Year	2017	2017
Theoretical peak performance	25 PFlops/s	436 TFlops
Processor type	Intel® Xeon® E5-2690 v3 @ 2.60GHz (12 cores, 64GB RAM) and NVIDIA® Tesla® P100 16GB	64 cores Intel(R) Xeon Phi(TM) CPU 7230 @ 1.30GHz
Total amount of memory	46.75 Terabytes	1.5 Terabytes
Number of Multicore Compute Nodes	1,431	164
Number of Hybrid Compute Nodes	5,320	
Total System Memory	437TB	18TB
Software	MPI/OpenMP/Chapel	MPI/OpenMP
OS	Linux	Linux

A benchmark for HPC architectures — LINPACK Benchmark

- The theoretical peak performance is an upper (theoretical) bound which is defined by the frequency of the processor – typical one gets between **0.1% and 80%** of this performance
- A scientific **real (?)** benchmark: Solution of a n-n dense linear system of equation with Gaussian Elimination
  - **LINPACK** Benchmark (www.top500.org)
  - Mflop/s Millions Floating-point Operations/sec.
  - Teraflop / Petaflop (a factor of 10^3)
- **Todays first entry**: Tianhe-2 System at National University of Defense Technolgy, China
  - 10,649,600 cores, Sunway SW26010, 260C 1.45GHz
  - OS: Linux
  - Theoretical peak performance : **125 PFlop/s**
  - Linpack: **93 PFlop/s** (Matrix size: N= 12,681,215)

Some of the World's Fastest Computer

The Top500 List

Last Top 500 List

- List of the 500 fastest HPC architectures (Rmax of the Linpack Benchmark)
- List is updated twice a year :
  ISC‘xy in Germany (June), **SC‘xy** in the USA (November)

- Information:

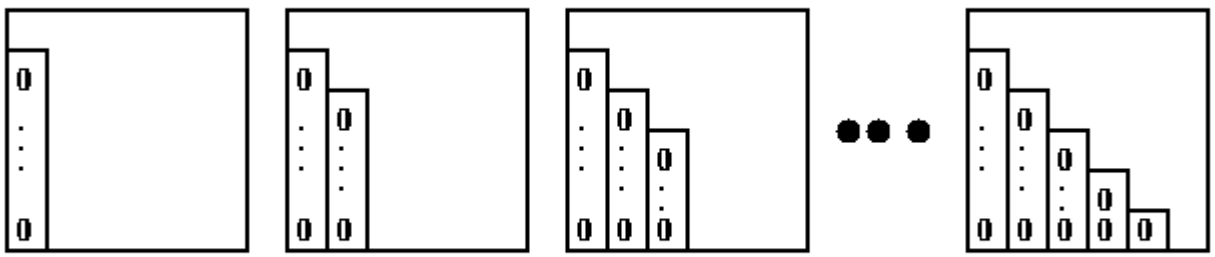
Manufacturer	Manufacturer or vendor
Computer Type	indicated by manufacturer or vendor
Installation Site	Customer
Location	Location and country
Year	Year of installation/last major update
Customer Segment	Academic,Research,Industry,Vendor,Class.
# Processors	Number of processors
Rmax	Maxmimal LINPACK performance achieved
Rpeak	Theoretical peak performance
Nmax	Problemsize for achieving Rmax
N1/2	Problemsize for achieving half of Rmax
Rank	Position within the TOP500 ranking

https://sc18.supercomputing.org/

LINPACK: Gaussian Elimination for the solution of A*x=b

- **Algorithm:** Add multiples of each row to later rows to make A upper triangular
- Solve resulting triangular system Ux = c by substitution

```
for each column i
zero it out below the diagonal by adding multiples of row i to later rows
for i = 1 to n-1
    … for each row j below row i
    for j = i+1 to n
        … add a multiple of row i to row j
        for k = i to n
            A(j,k) = A(j,k) - (A(j,i)/A(i,i)) * A(i,k)
```



Complexity

n^3 flops for a matrix of size n x n

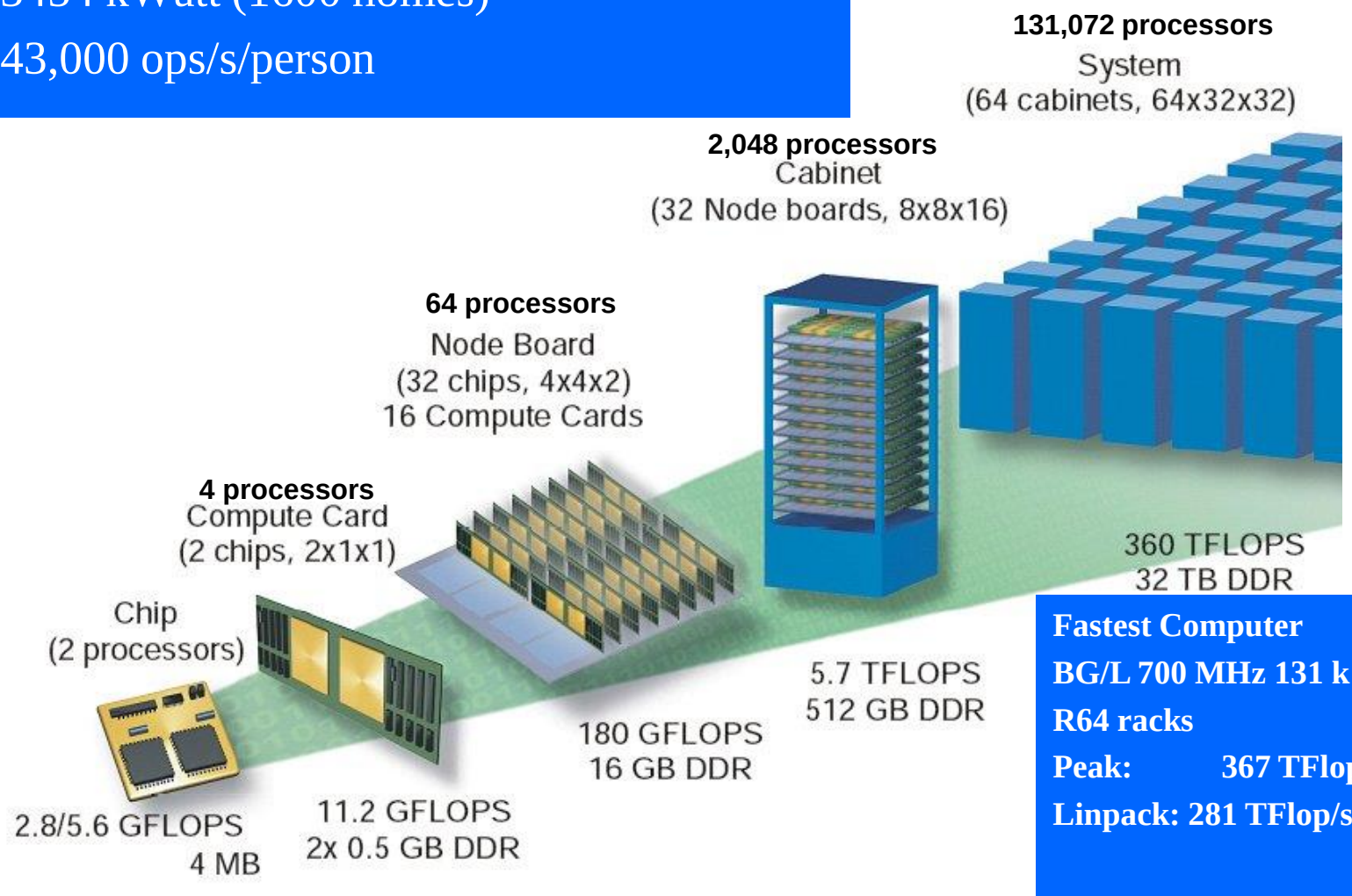
The #1 in the TOP 500 (in 2008)

IBM BlueGene/L #1 131,072 Processors

Total of 18 systems all in the Top 100

3454 kWatt (1600 homes)

43,000 ops/s/person



Fastest Computer

BG/L 700 MHz 131 k proc

R64 racks

Peak: 367 TFlop/s

Linpack: 281 TFlop/s

TOP 500 Supercomputer List (as of 2017, www.top500.org)

Rank	System	Cores	Rmax in Tflops/s	Rpeak in Tflops/s	Power (KW)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC, National Supercomputing Center in Wuxi, China	10,649,600	93,014.6	125,435.9	15,371
2	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon Phi 31S1P, National Super Computer Center in Guangzhou, China	3,120,000	33,862.7	54,902.4	17,808
3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, NVIDIA Tesla P100, Swiss National Supercomputing Centre (CSCS), Switzerland	361,760	19,590.0	25,326.3	2,272
4	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
5	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM, DOE/NNSA/LLNL United States	1,572,864	17,173.2	20,132.7	7,890
6	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc., DOE/SC/LBNL/NERSC United States	622,336	14,014.7	27,880.7	3,939
7	**Oakforest-PACS** - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Fujitsu, Joint Center for Advanced High Performance Computing, Japan	556,104	13,554.6	24,913.5	2,718.7
8	**K computer**, SPARC64 VIIIfx 2.0GHz, Tofu interconnect , Fujitsu, RIKEN Advanced Institute for Computational Science (AICS), Japan	705,024	10,510.0	11,280.4	12,659.9
9	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom , IBM, DOE/SC/Argonne National Laboratory United States	786,432	8,586.6	10,066.3	3,945
10	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, DOE/NNSA/LANL/SNL, United States	301,056	8,100.9	11,078.9	4,232.6

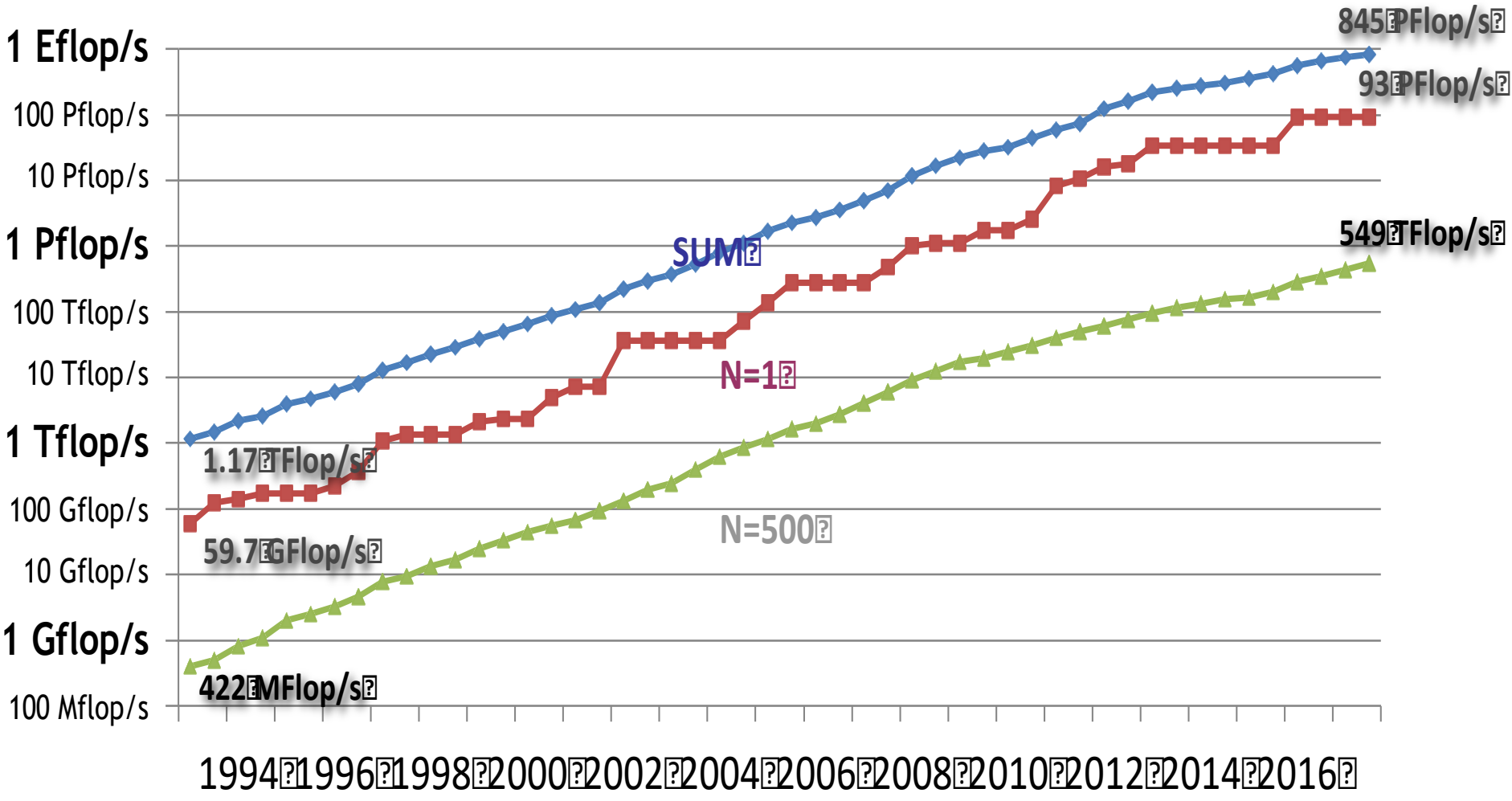
Sunway Taihulight at NRCPC in Wuxi China

Peak:	125 PFlop/s	**Processor:**	40,960 Sunway		
Linpack:	105 PFlop/s	**Cores/proc:**	256	**Memory:**	1.31 PB
Power:	15 MW	**Clock:**	1.45 GHz	**Memory BW:**	5.6 PB/s
Cost:	$273M USD	**Proc Peak**:	3.06 TFlop/s	**Storage:**	20 PB

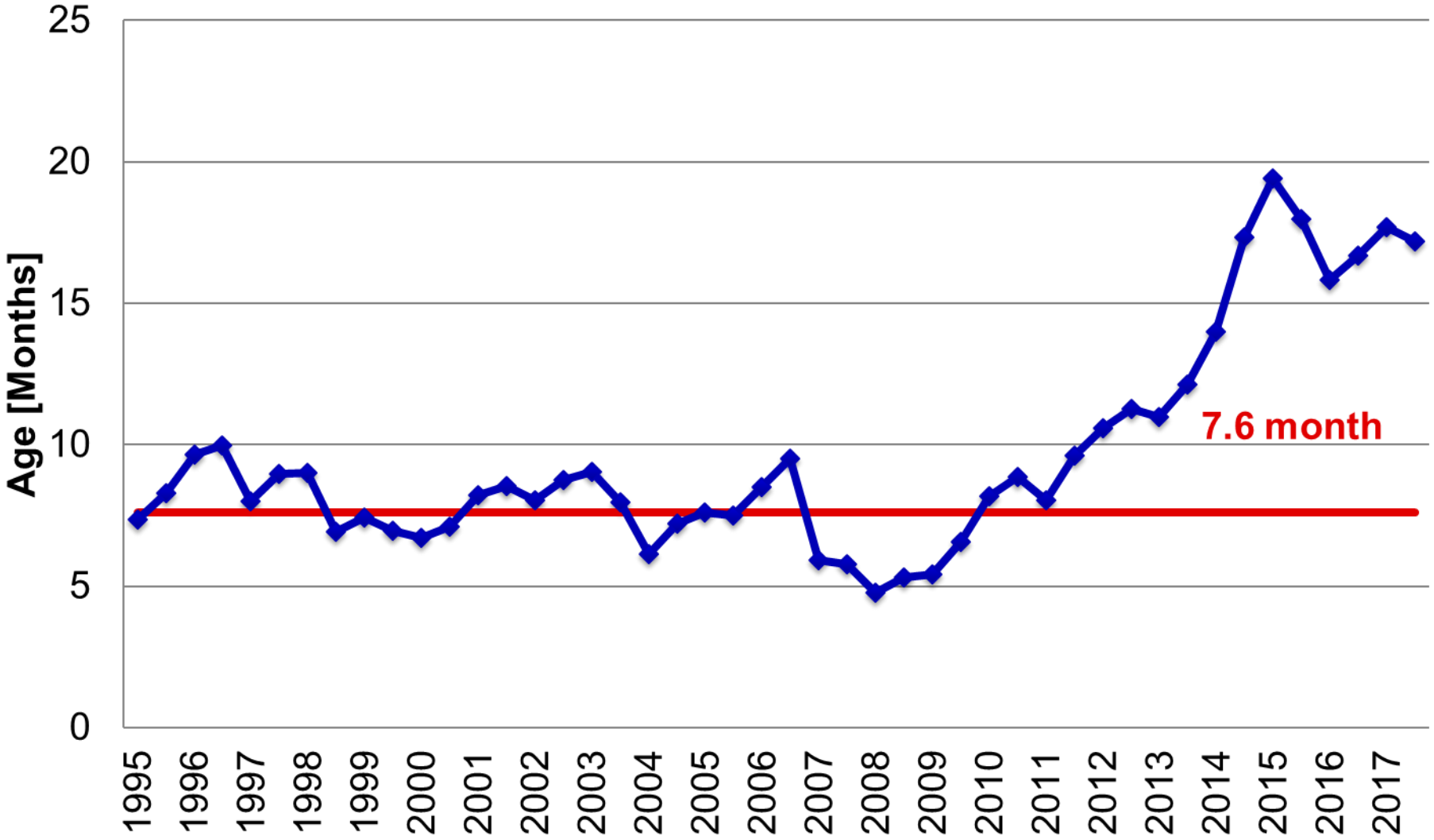
Cori at NERSC in Berkeley, CA

Phase 1	**Peak:**	2.8 PFlop/s	**Proc:** 3800 Haswell (ph 1) **Cores:** 2.3 GHz, 16/proc **Proc Peak**: 0.6 TFlop/s	**Memory:** **Storage:**	0.3 PB shared
Phase 2	**Peak:** **LinPack: 14 Pflops/s** **Power:**	28PFlop/s **4 MW**	**Processor:** 9300 KNL **Cores:** 1.4 GHz, 68/proc **Node Peak**: 3 TFlop/s	**Memory:** **MemBW:** 1 PB/s HBM,DDR **Storage:**	1.31 PB 28 PB

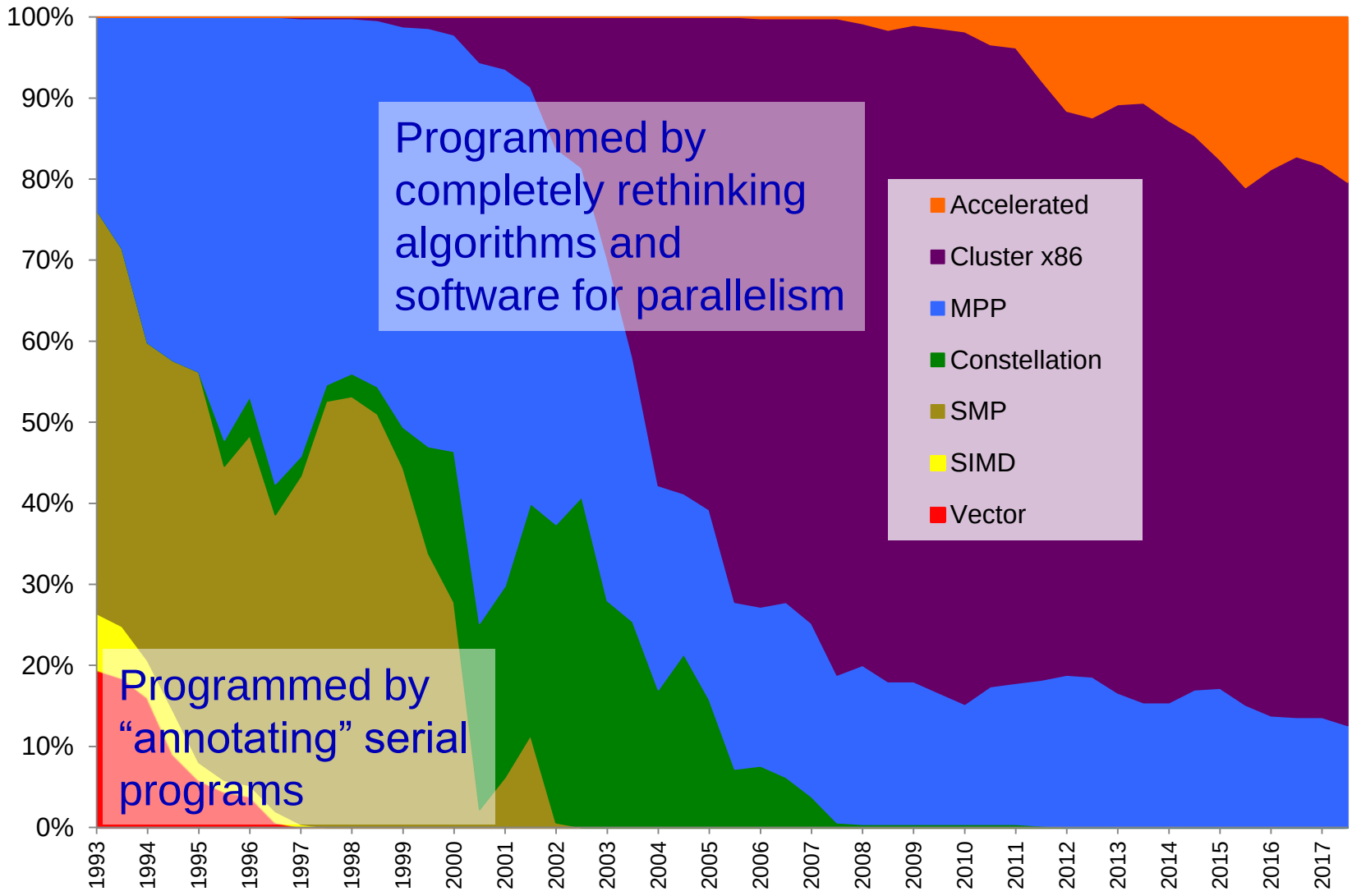
Average System Age



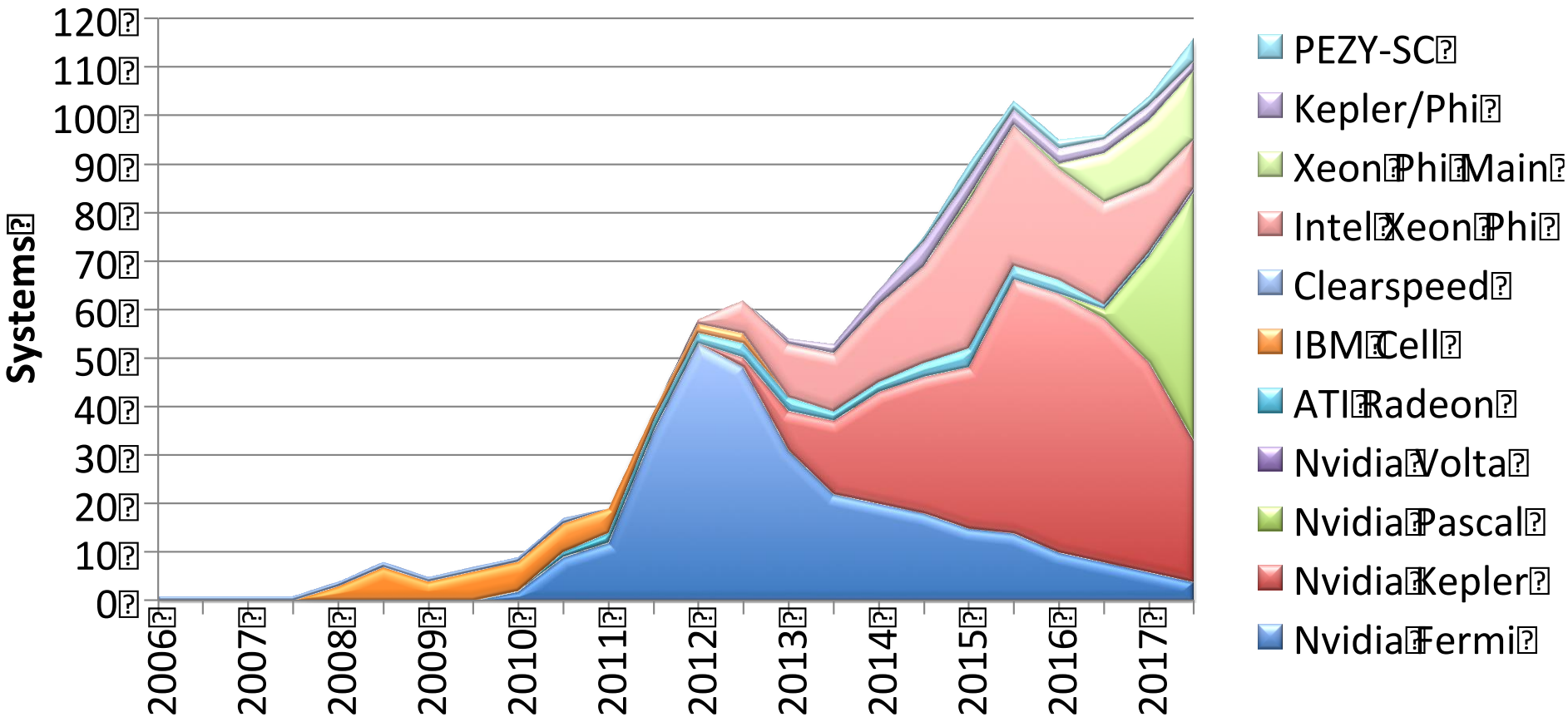
Average System Age



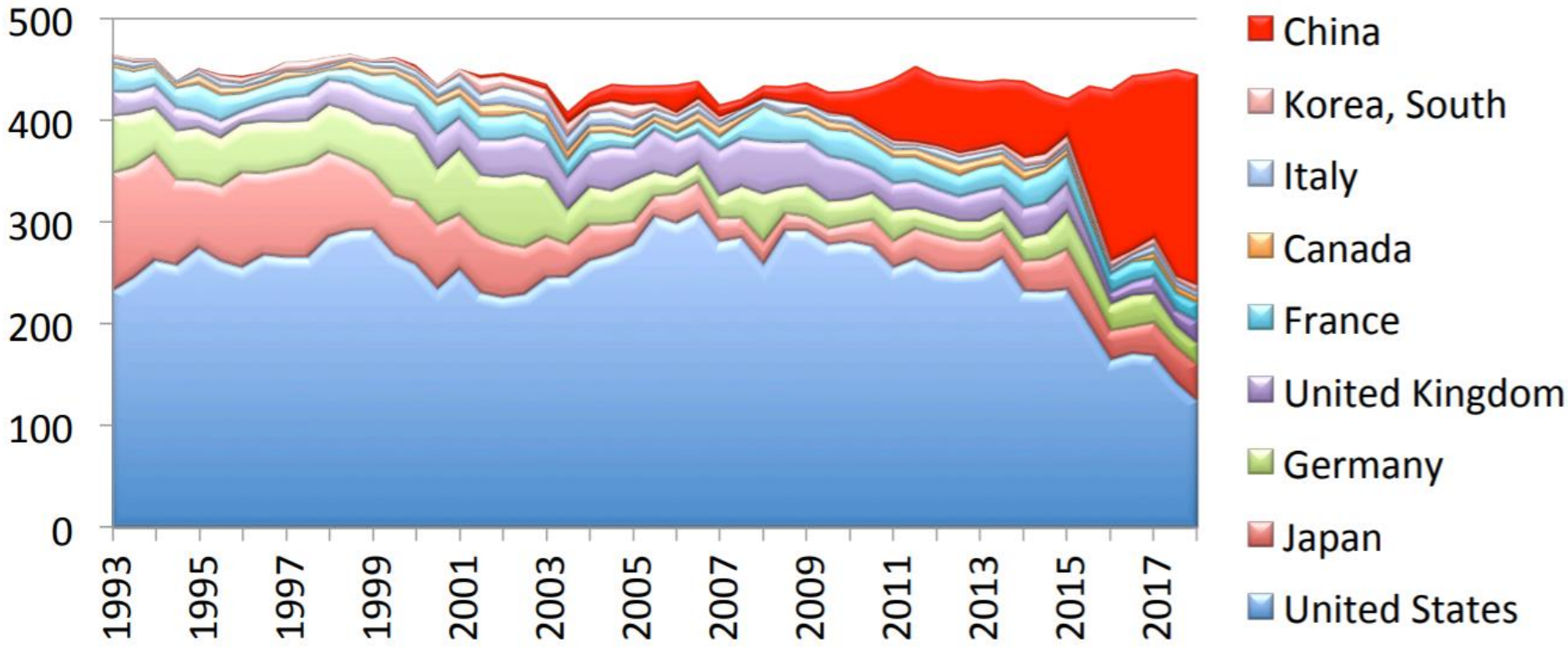
From Vector Supercomputers to Massively Parallel Accelerator Systems



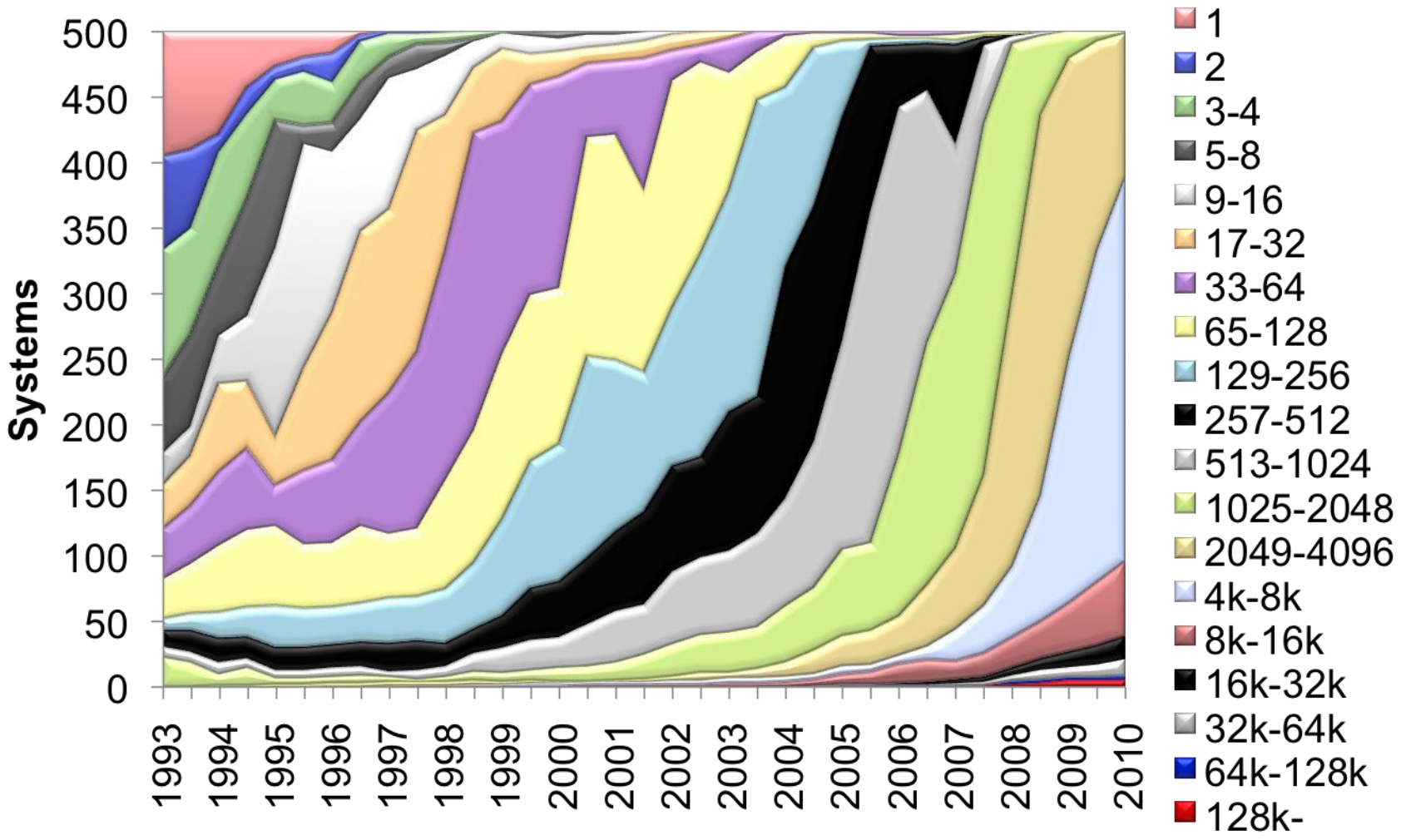
Performance of Accelerators (2017)



Countries (2017)



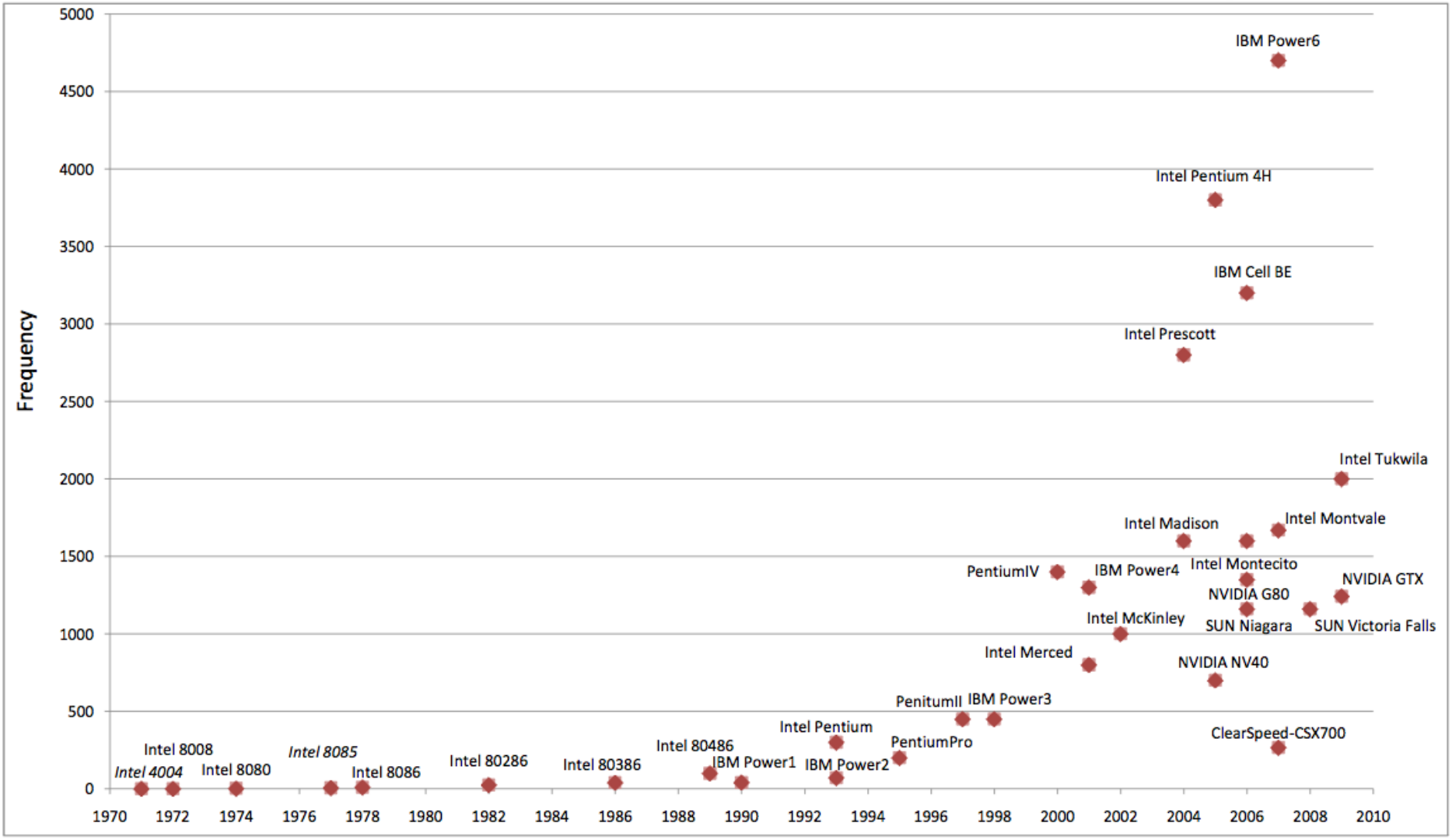
Core Count


Systems
500
450
400
350
300
250
200
150
100
50
0
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
1
2
3-4
5-8
9-16
17-32
33-64
65-128
129-256
257-512
513-1024
1025-2048
2049-4096
4k-8k
8k-16k
16k-32k
32k-64k
64k-128k
128k-

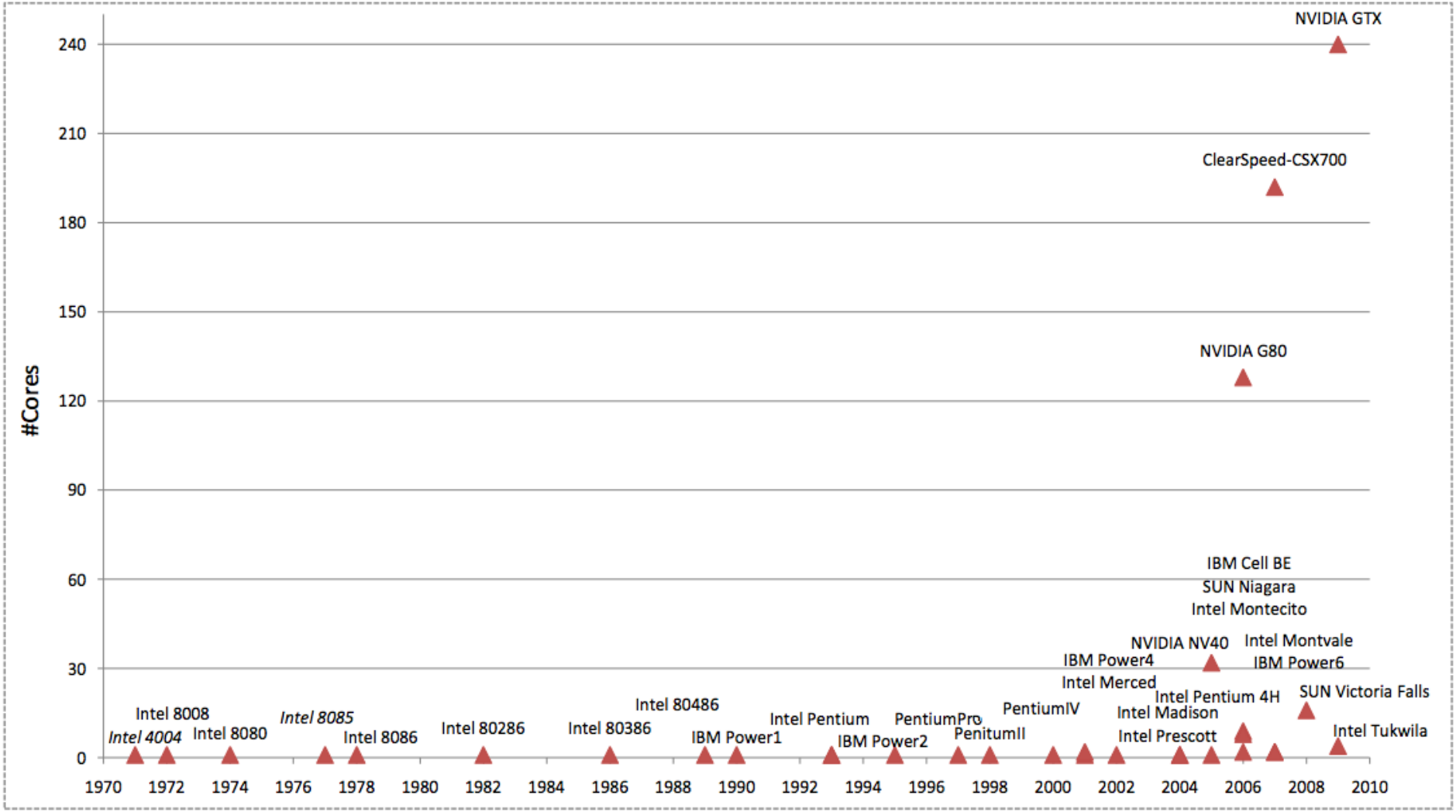
TOP 500 List from 2017 (Switzerland)

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272.0
82	**Piz Daint Multicore** - Cray XC40, Xeon E5-2695v4 18C 2.1GHz, Aries interconnect , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	44,928	1,410.7	1,509.6	519
264	**EPFL Blue Brain IV** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect , IBM Swiss National Supercomputing Centre (CSCS) Switzerland	65,536	715.6	838.9	328.8

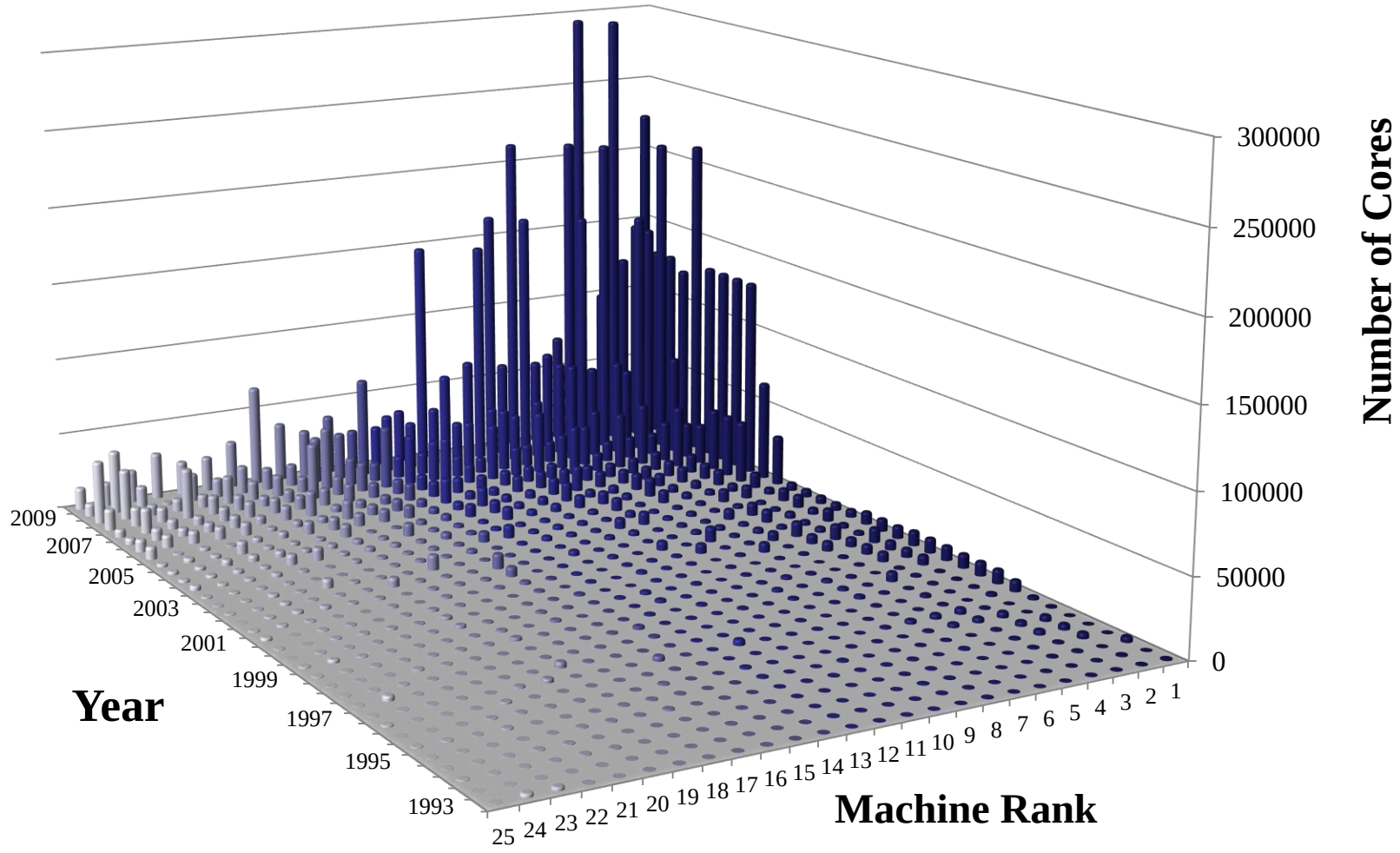
Development of frequency



Number of cores on a node



Number of Cores of the 500 fastest Computer



Kflop/s per Person, November 2004, Top500

Hint: Peter Jackson ….

WETA Digital (Lord of the Rings)

1600
1400
1200
1000
800
600
400
200
0

China
Brazil
Italy
Mexico
France
Korea, South
Saudia Arabia
Germany
Belarus
Switzerland
Canada
Spain
Japan
Israel
United Kingdom
New Zealand
United States

Source: Jack Dongarra

TOP 500 List from 2004 (WETA Digital)

259	WETA Digital New Zealand/2005	HS20 Cluster, Xeon EM64T 3.6 GHz 1000	3666.99 7200
177	WETA Digital New Zealand/2003	BladeCenter Cluster Xeon 2.8 GHz 1176	2026 6585.6
180	WETA Digital New Zealand/2003	BladeCenter Cluster Xeon 2.8 GHz 1080	2026 6048

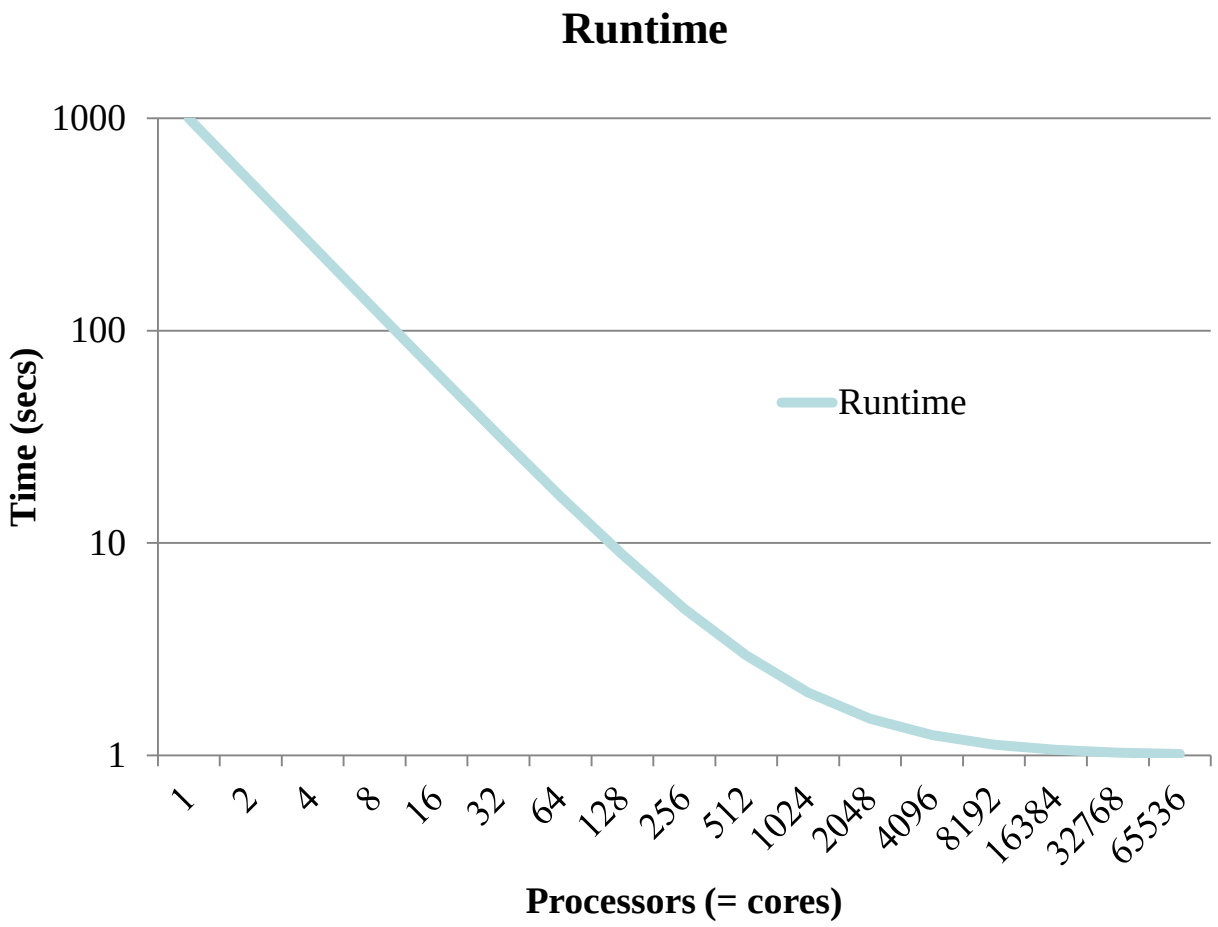
Measuring Performance

Different ways to show performance results

or

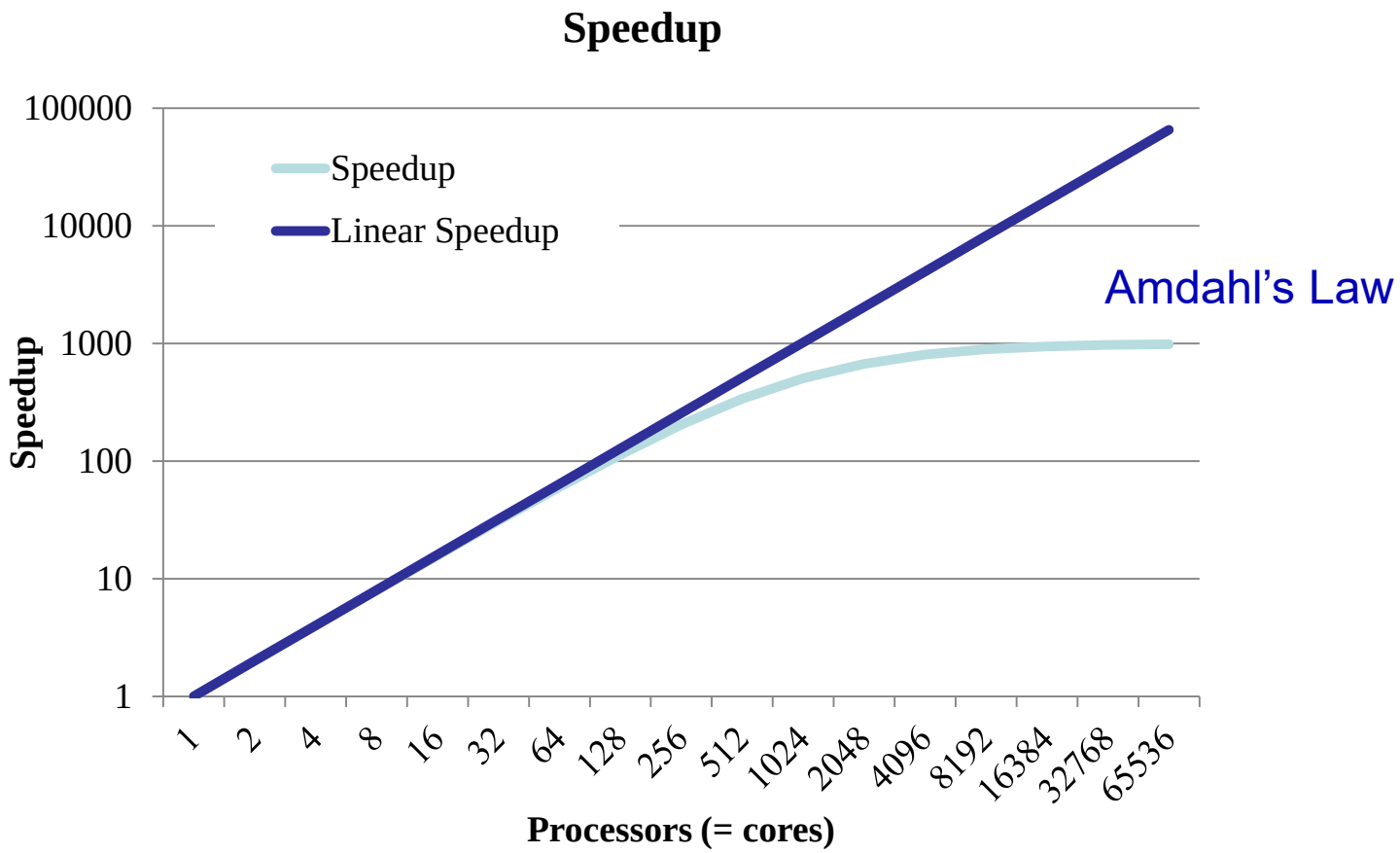
How to fool the masses when reporting parallel performance numbers

Reporting running time, but hide the results



A bit better (log scale), but the point of this is to show the value of parallelism

Reporting Speedup



Speedup(P) = Time(1)/Time(P)

This is a **strong scaling** plot: fixed problem size, vary number of processors

Amdahl's Law

- Suppose only part of an application is parallel
- Amdahl's law
  - **s** = fraction of work done sequentially (Amdahl fraction), so (1-s) is fraction parallelizable
  - **P** = number of processors

$$\text{Speedup}(P) = \text{Time}(1)/\text{Time}(P)$$

$$\leq 1/(s + (1-s)/P)$$

$$\leq 1/s$$

- Even if the parallel part speeds up perfectly performance is limited by the sequential part
- Top500 list: currently fastest machine has P ~ 10M; even Piz Daint has P ~ 350K

Amdahl‘s Law

- The acceleration due to parallelism is limited by the part of the computation that is performed sequentially

$$Sp = \frac{1}{q\frac{1}{p} + (1-q)}$$

Sp^* = Acceleration (speed up) with p Processors

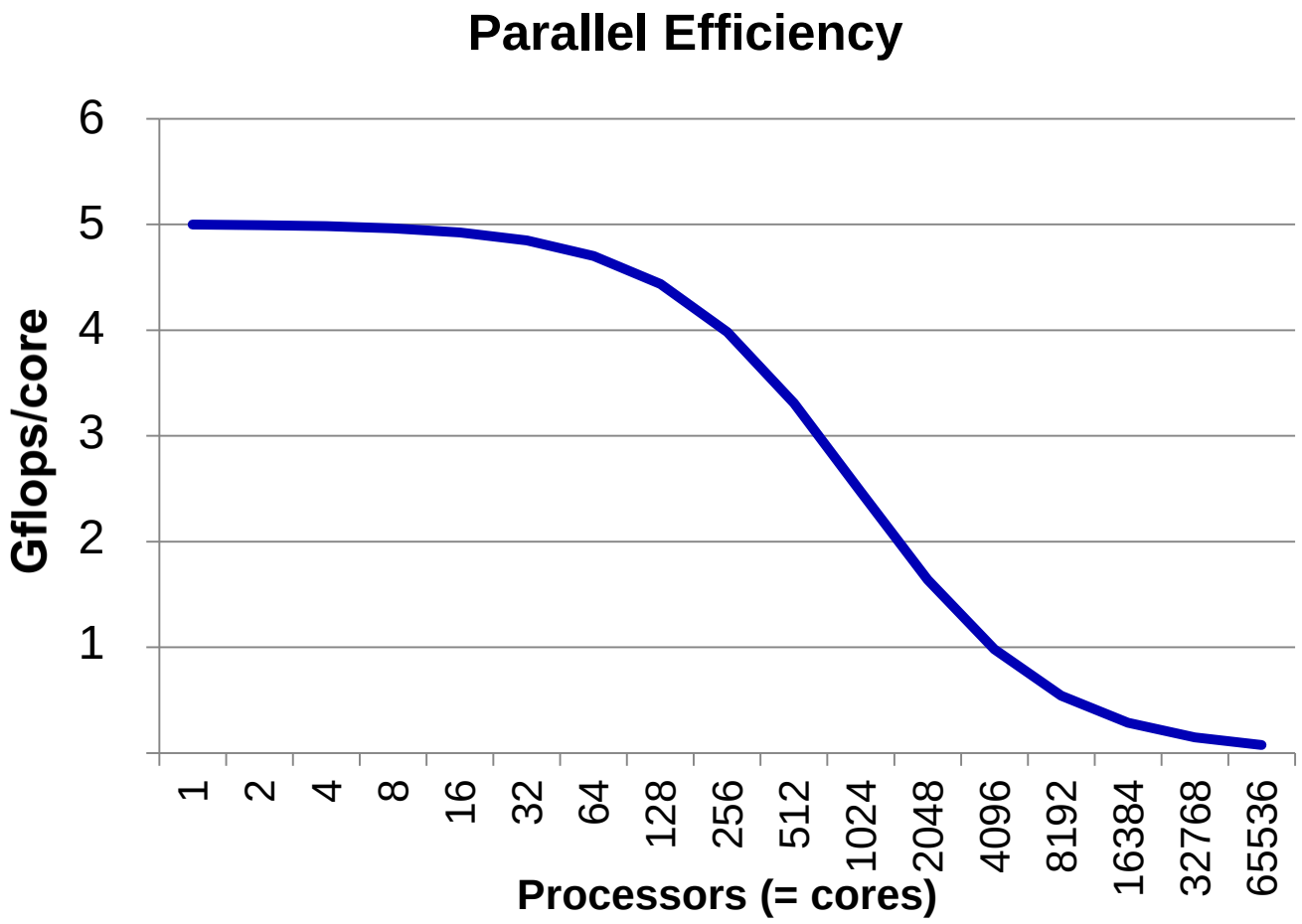
q = Fraction of operation that can be parallized.

Example for Speedup

q	p = 32	p = 64	p = 128
1.00	32.0	64.0	128.0
0.98	19.8	28.3	36.2
0.95	12.6	15.4	17.4
0.90	7.8	8.8	9.3
0.70	3.1		
0.50	1.9		

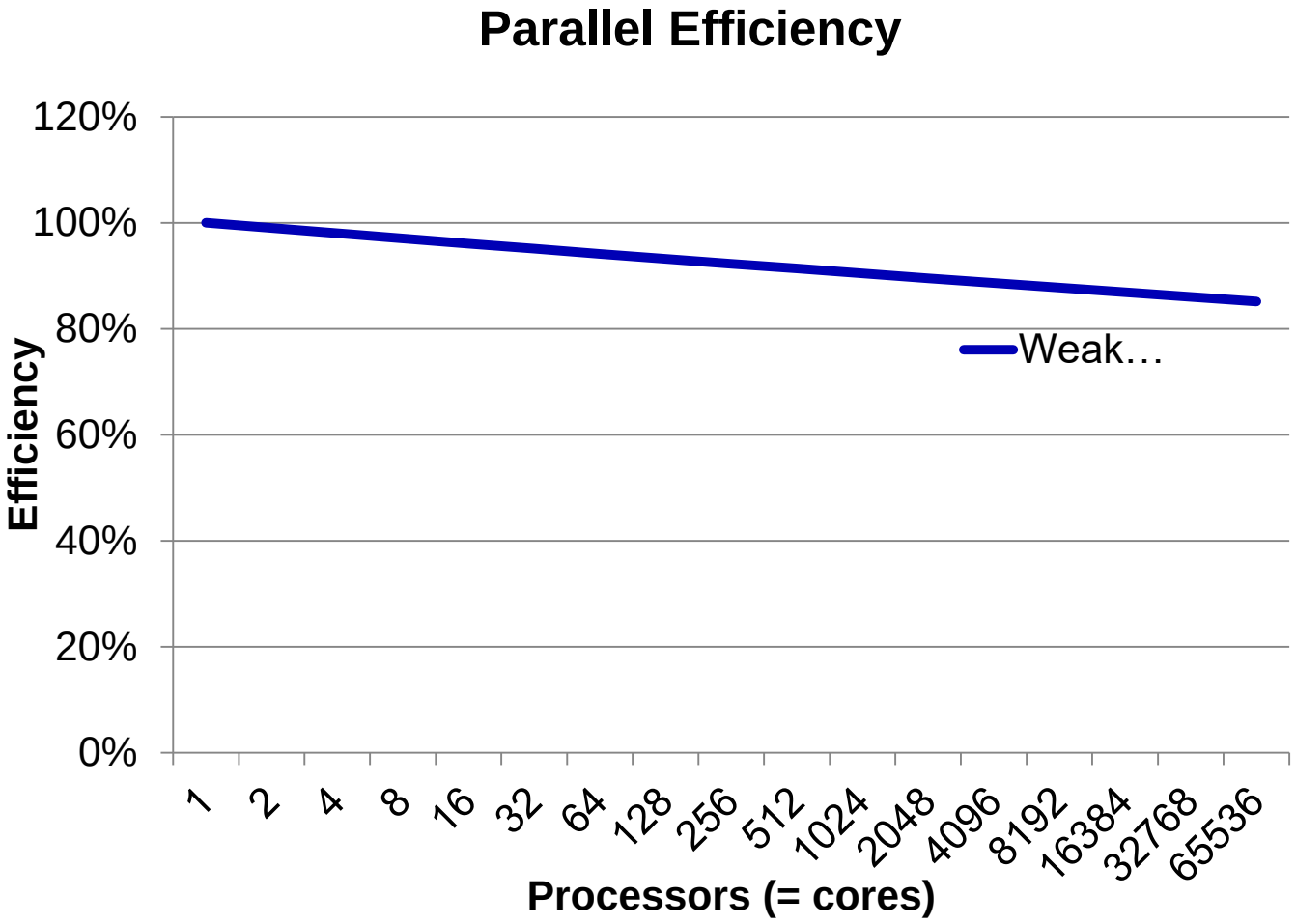
* Parallel overhead und data bandwidth is not taken into account

Parallel Efficiency



Same **strong scaling** results shown as efficiency

Parallel Efficiency (Weak Scaling)



Weak scaling uses a fixed problem size **per processor.** Can report as:

- Flop/s (or other rate) per processor; Efficiency based on rate per processor
- Time (if algorithm is linear in data size)

Carefully choose and report your baseline (P=1)

See David Bailey's Twelve Ways to Fool the Masses..

1. Use 64-bit baseline, 32-bit parallel numbers
2. Use a bad algorithm for the baseline
3. Use a bad implementation for the baseline
4. Use a parallel code (with your new fancy and expensive runtime) for the baseline
5. Use a slower processor for the baseline
6. Don't report running times at all

Report your baseline in absolute running time

Please read

http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/twelve-ways.pdf

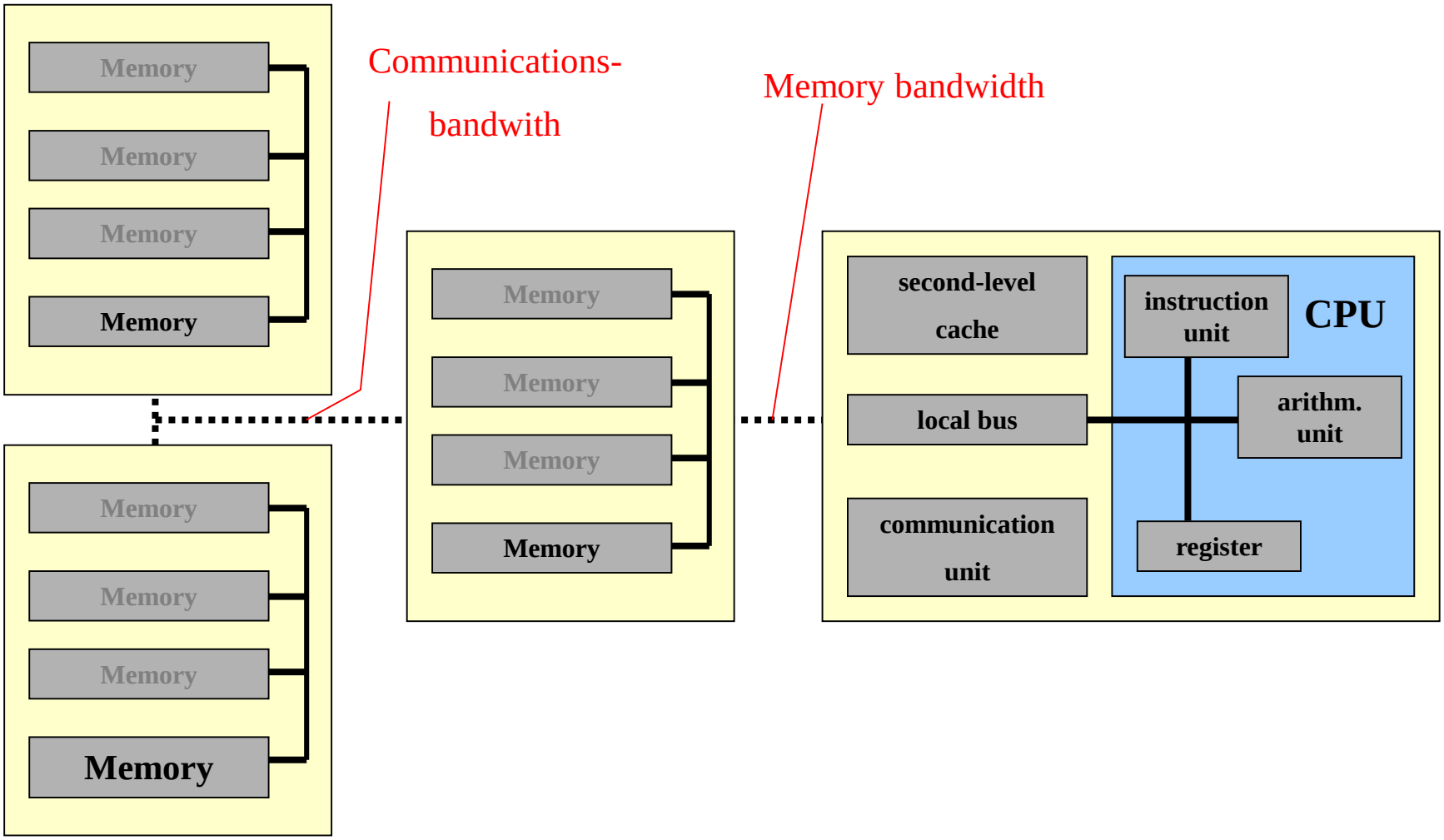
TOP500

- Please try

  https://www.top500.org/statistics/overtime/

- Operating System Family (over the last 10 years)
- Cores per Socket (over the last 10 years)

Data paths, memory hierarchies und bottlenecks



Speed (ns): 10000 100 10 1

Size (bytes): T 1 G 1 MB 10 K

Data paths, memory hierarchies and bottlenecks

	yearly increase	Typical value in 2005	Typical value in 2010	Typical value in 2020
Single-Processor Floating-Point Performance	**59%**	**4 GFLOP/s**	**32 GFLOP/s**	**3.300 GFLOP/s**
DRAM Latency	**5.5%**	**70 ns = 280 FP**	**50 ns = 1.600 FP**	**28 ns = 94.000 FP**
Communication with other CPUs	**5.5%**	**28.000 FP**	**1.160.000 FP**	**94.000.000 FP**

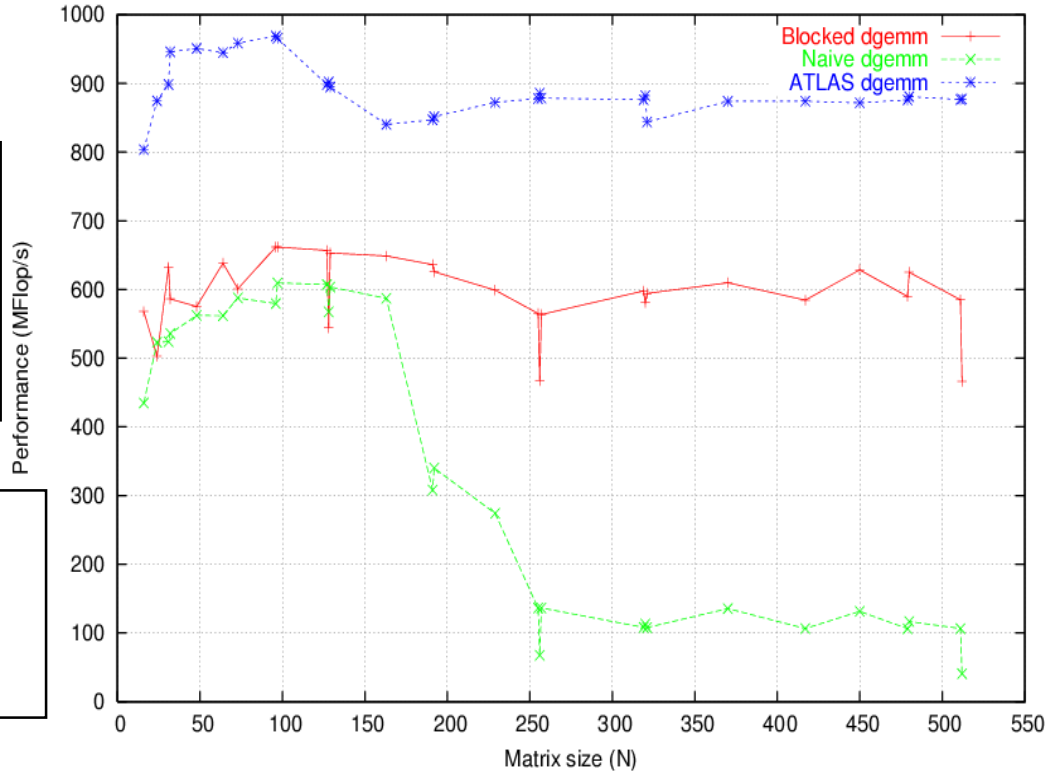
- Source: *Getting Up to Speed: The Future of Supercomputing*, National Research Council, 222 pages, 2004, National Academies Press, Washington DC, ISBN 0-309-09502-6.

Matrix-matrix multiplication (C = C + A*B)

- Naive approach

```
do i = 1 to n
 do j = 1 to n
   do k = 1 to n
     C[i][j]+=A[i][k]*B[k][j]
```

Performance (Mflops) in relation to N:
- compiled (green)
- own optimized Version (red)
- optimized library (blue)



- Only **50 Mflops** with simple approach and **1000 Mflops** with algorithms, which try to minimize **data access** pattern.

The number of floating point operation is the same, but the data access pattern is really important for the performance!

Programming of HPC architectures

It is not as trivial as …

Parallelization — Matrix Multiplication

MPI (Message Passing Interface)

```
// Initialization
MPI_INIT( … );
MPI_COMM_RANK( …, myid, …);
MPI_COMM_SIZE(…, nproc, …);
// Broadcast parameter and matrices
MPI_Bcast(n, …);
 …
MPI_Bcast (A, …);
// compute partial results
mmul( … );
// collect results
if (myid == 0) {    // master
    for ( iproc=0; i<nproc; i++)
        MPI_Recv(C, …);
}
else {   // slave
    MPI_Send(C, …)
}
// finalize
MPI_Finalize( …);
```

OpenMP (Shared Memory Threads)

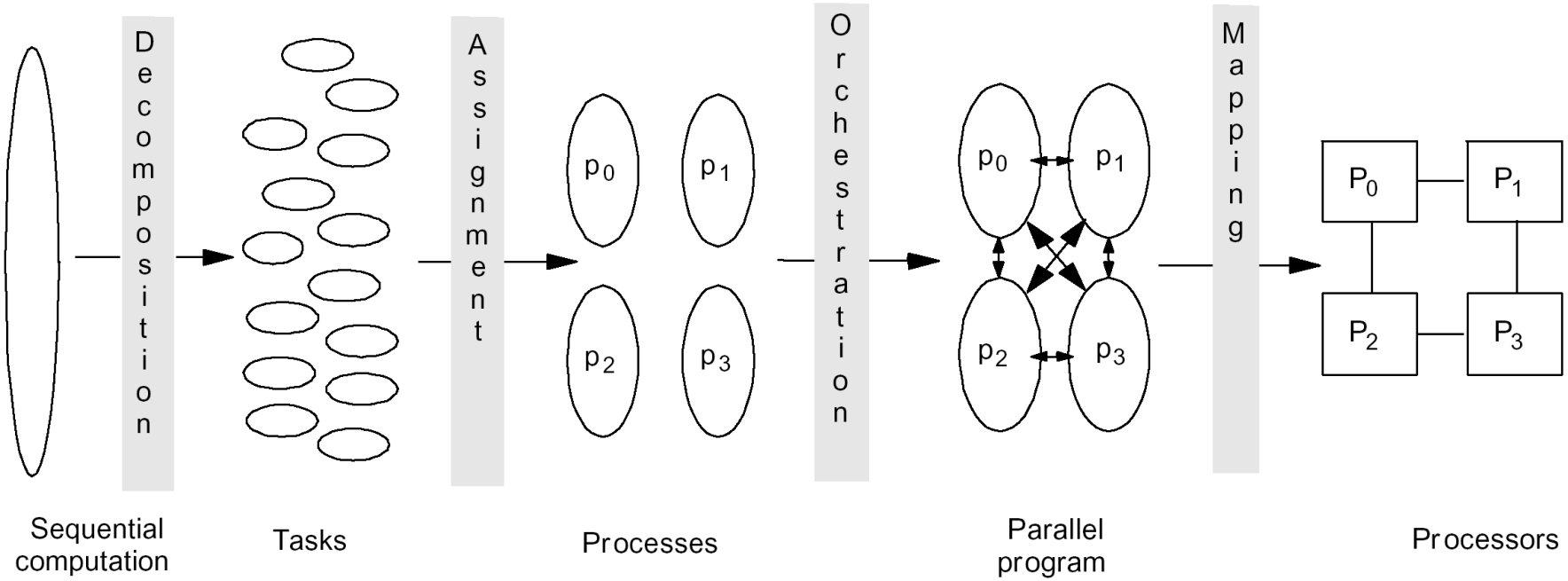
```
# pragma omp parallel  \
# shared (A, B, C, n) private (i, j, k)
for (i = 1;   i < n; i++)
  for (j = 1;  j < n; j++)
    for (k = 1;   k < n; k++)
      C[i][j] += A[i][k]*B[k][j];
```

Cell BE

```
B_gpu = sp_make2_f32(ln, lk,&r4work[rb],
        SP_MEMORY_NORMAL, sizeof(float), 0);
C_gpu = sp_make2_f32(lm, ln,&r4work[rc],
        SP_MEMORY_NORMAL, sizeof(float), 0);

sp_cblas_sgemm(CblasColMajor, CblasNoTrans,
               CblasNoTrans, lm, ln,lk,
               ALPHA_gpu, A_gpu, lm, B_gpu,
               lk, BETA_gpu, C_gpu, lm);
sp_hint_read_f32(C_gpu);
```

4 Tasks in parallel programming



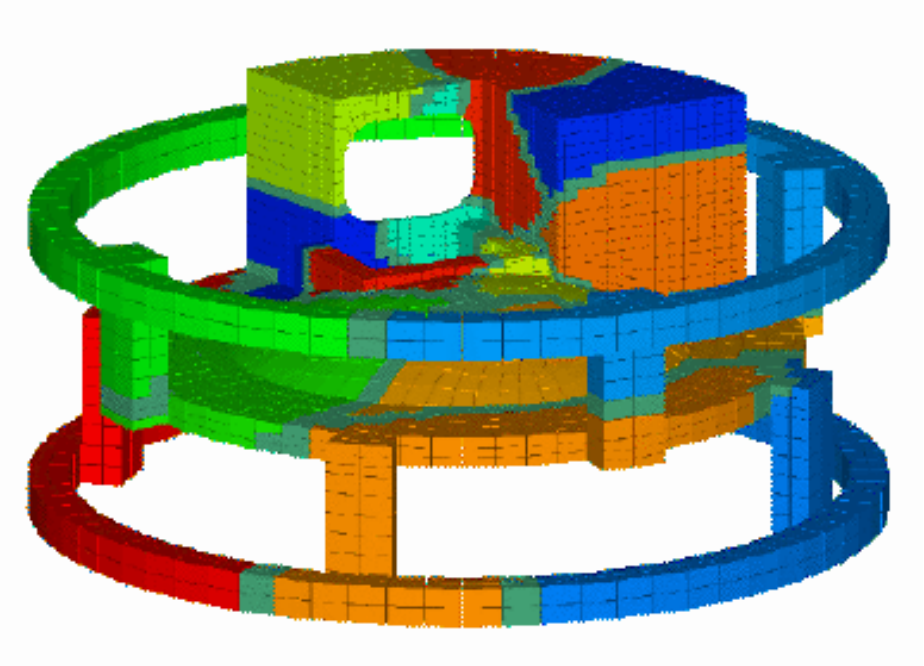
- **decomposition** – of the computation into subtasks
- **assignment** – of the tasks to processes/threads
- **orchestration** – Data access pattern, communication, synchronization.
- **mapping** – of processes to processors (hardware)

Partitioning

- Example of a partitioning of a 3-D tool (Bruce Hendrikson, Sandia National Laboratory)

communication:

Inertia	14652
Spektrale Bisection	3425
Kernighin-Lin	21312
Multilevel Bisection	1321



Partitioning algorithms are one important method to minimize communication on massively-parallel architectures.

A short view in the future

- The importance of parallel simulations is increasing:

the computed replaces the physical

- but we have to deal with research challenges in parallel computing in
  - algorithms
  - software technology
  - architectures and processor technology

in order to exploit the advantages of parallel HPC architectures **(and please remember that your laptops and handhelds will have thousands of cores in a few years from now).**