Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

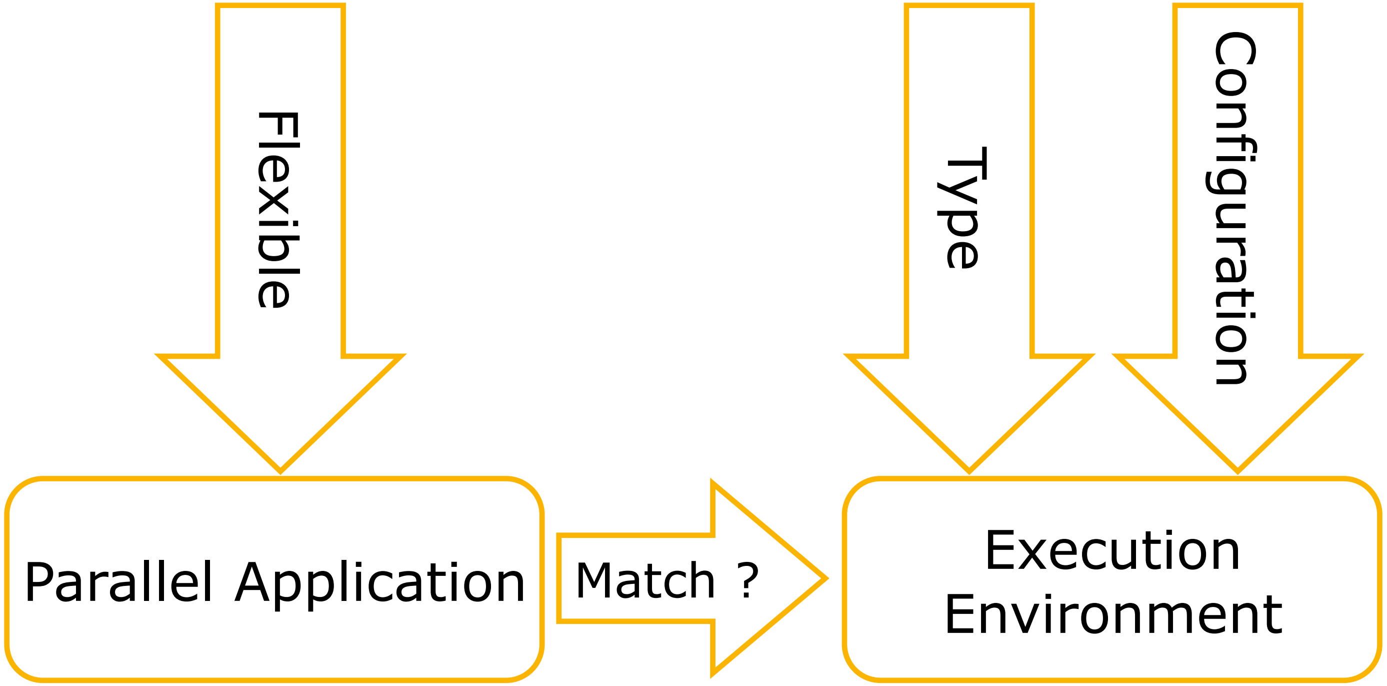
Metrics

Programmierung Paralleler und Verteilter Systeme (PPV)

Sommer 2015

Frank Feinbube, M.Sc., Felix Eberhardt, M.Sc.,
Prof. Dr. Andreas Polze

The Parallel Programming Problem



Flexible

Type

Configuration

Parallel Application

Match ?

Execution Environment

Which One Is Faster ?

- Usage scenario
  - Transporting a fridge
- Usage environment
  - Driving through a forest
- Perception of performance
  - Maximum speed
  - Average speed
  - Acceleration

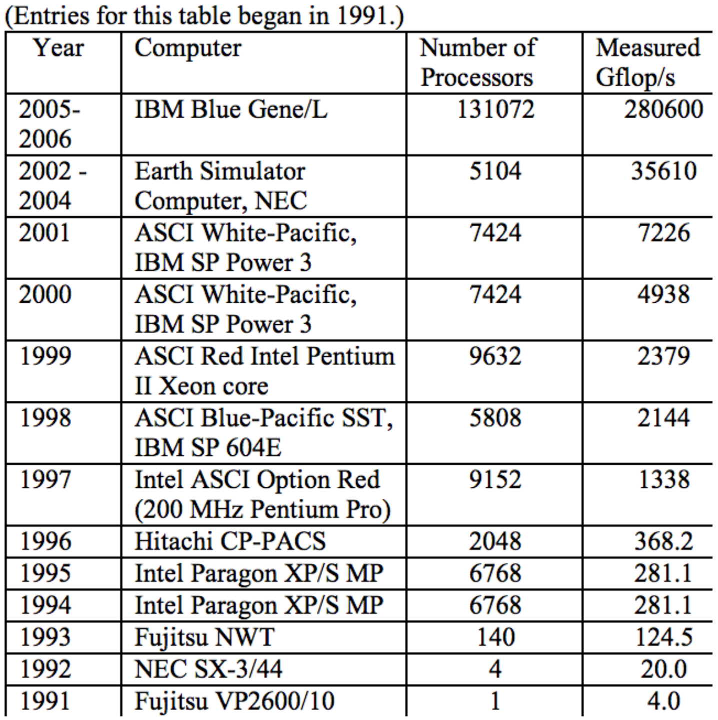
- We need some kind of application-specific benchmark

Benchmarks

- Parallelization problems are traditionally speedup problems
- Traditional focus of high-performance computing
- Standard Performance Evaluation Corporation (SPEC)
    - SPEC CPU – Measure compute-intensive integer and floating point performance on uniprocessor machines
    - SPEC MPI – Benchmark suite for evaluating MPI-parallel, floating point, compute intense workload
    - SPEC OMP – Benchmark suite for applications using OpenMP
- NAS Parallel Benchmarks
    - Performance evaluation of HPC systems
    - Developed by NASA Advanced Supercomputing Division
    - Available in OpenMP, Java, and HPF flavours
- Linpack

Linpack

- Fortran library for solving linear equations
- Developed for supercomputers of the 1970s
- Linpack as benchmark grew out of the user documentation
  - Solving of dense system of linear equations
  - Very regular problem, good for peak performance
  - Result in *floating point operations / s (FLOPS)*
  - Base for the TOP500 benchmark of supercomputers
  - Increasingly difficult to run on latest HPC hardware
  - Versions for C/MPI, Java, HPF
  - Introduced by Jack Dongarra

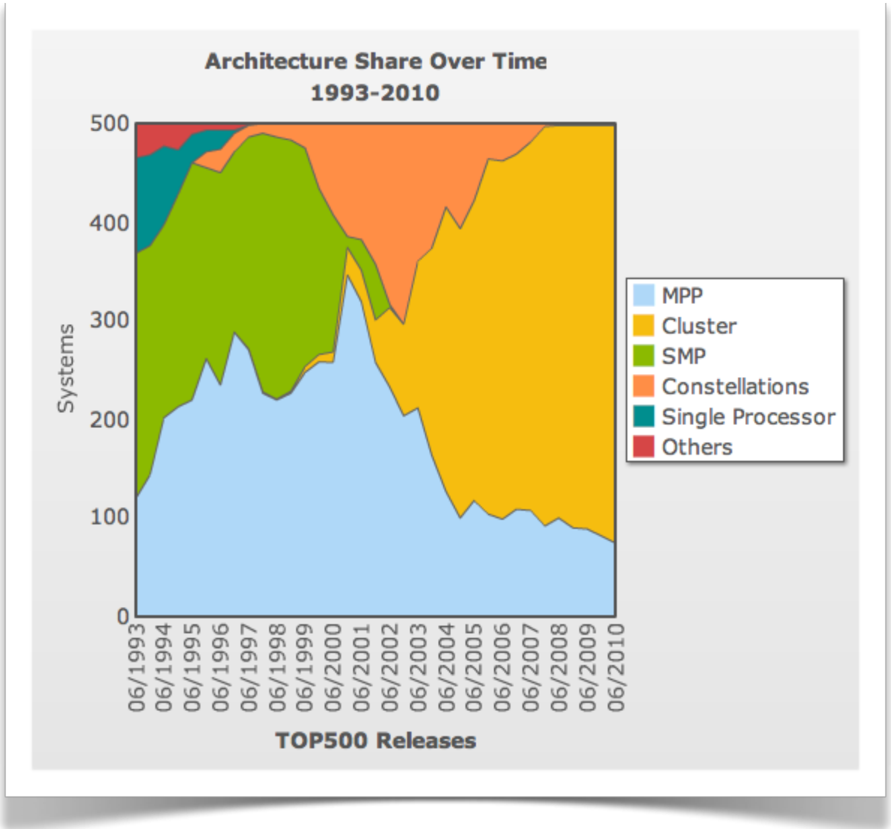


(Entries for this table began in 1991.)

Year	Computer	Number of Processors	Measured Gflop/s
2005-2006	IBM Blue Gene/L	131072	280600
2002 - 2004	Earth Simulator Computer, NEC	5104	35610
2001	ASCI White-Pacific, IBM SP Power 3	7424	7226
2000	ASCI White-Pacific, IBM SP Power 3	7424	4938
1999	ASCI Red Intel Pentium II Xeon core	9632	2379
1998	ASCI Blue-Pacific SST, IBM SP 604E	5808	2144
1997	Intel ASCI Option Red (200 MHz Pentium Pro)	9152	1338
1996	Hitachi CP-PACS	2048	368.2
1995	Intel Paragon XP/S MP	6768	281.1
1994	Intel Paragon XP/S MP	6768	281.1
1993	Fujitsu NWT	140	124.5
1992	NEC SX-3/44	4	20.0
1991	Fujitsu VP2600/10	1	4.0

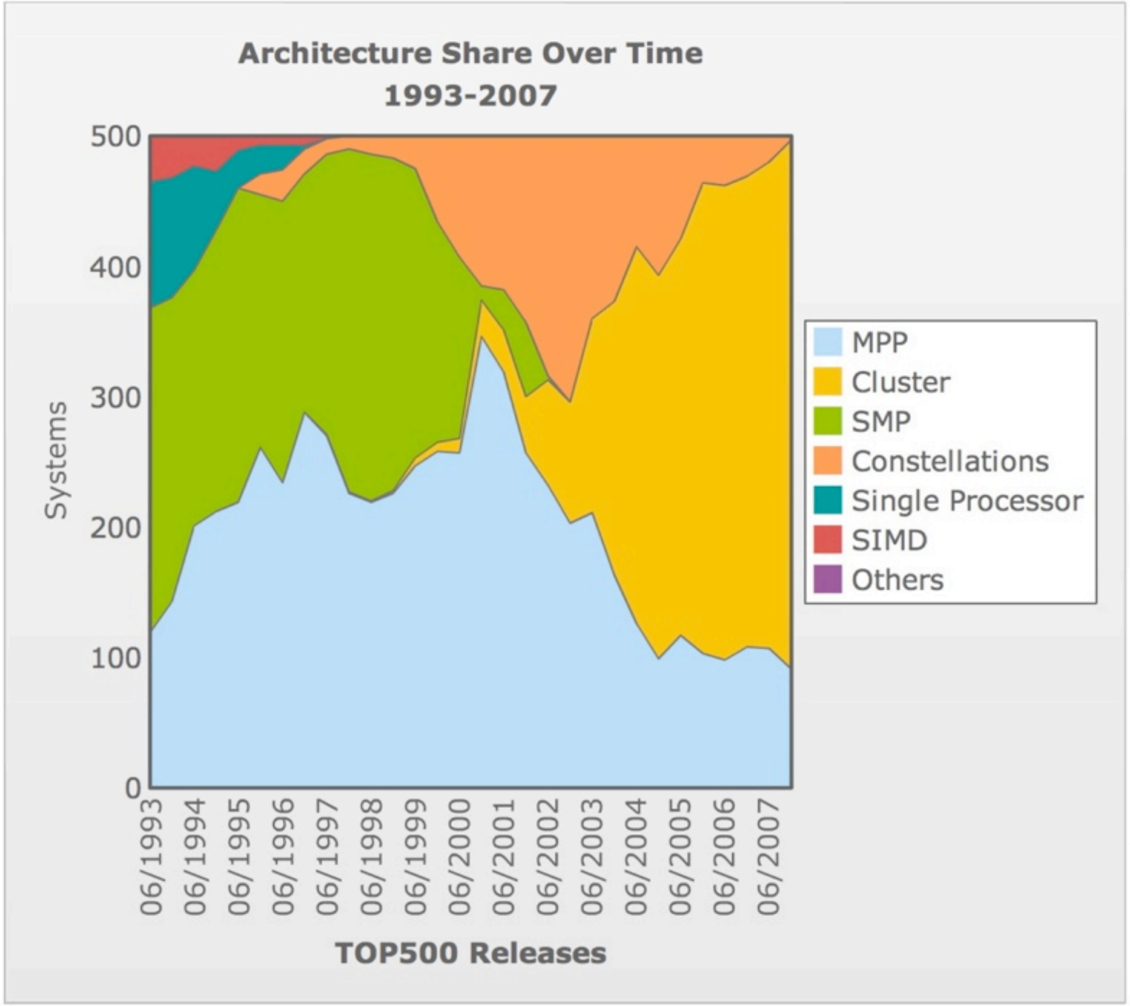
TOP 500

- It took 11 years to get from 1 TeraFLOP to 1 PetaFLOP
- Performance doubled approximately every year
- Assuming the trend continues, ExaFLOP by 2020
- Top machine in 2012 was the IBM Sequoia
  - □ 16,3 Petaflops
  - □ 1.6 PB memory
  - □ 98304 compute nodes
  - □ 1.6 Million cores
  - □ 7890 kW power



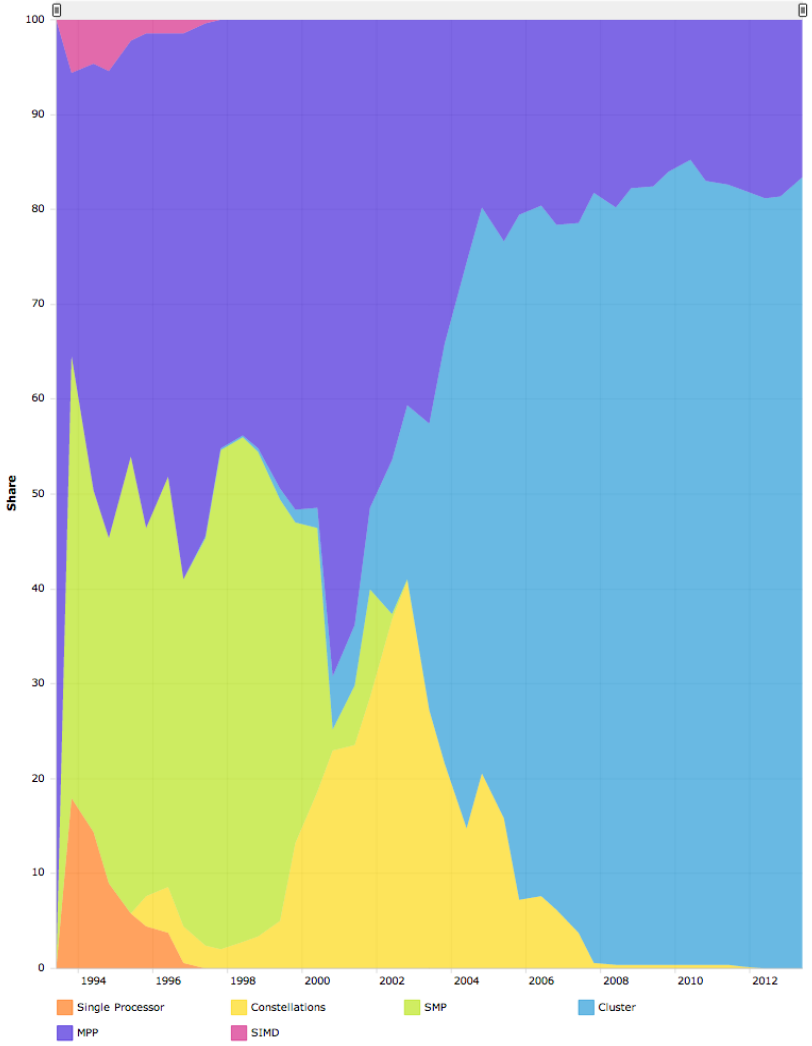
TOP 500 - Clusters vs. MPP (# systems)

- **Clusters** in the TOP500 have more nodes than cores per node
- **Constellation** systems in the TOP500 have more cores per node than nodes at all
- **MPP** systems have specialized interconnects for low latency

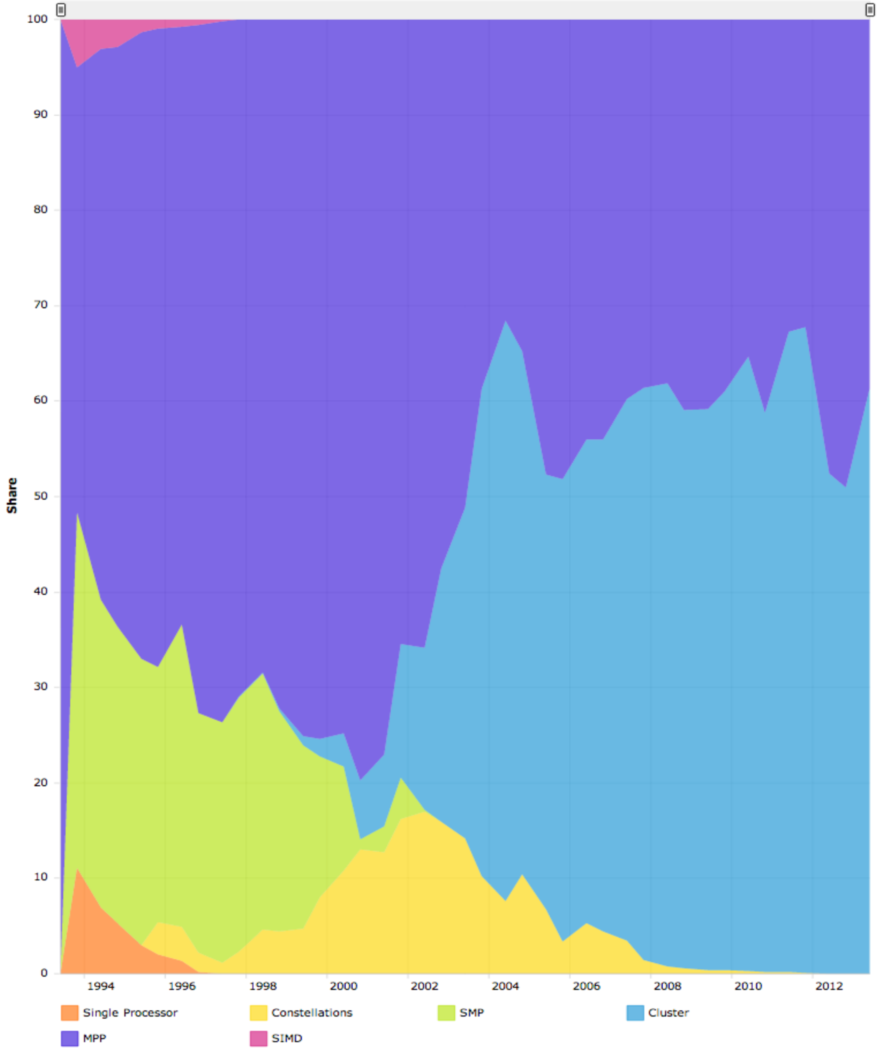


TOP 500 - Clusters vs. MPP

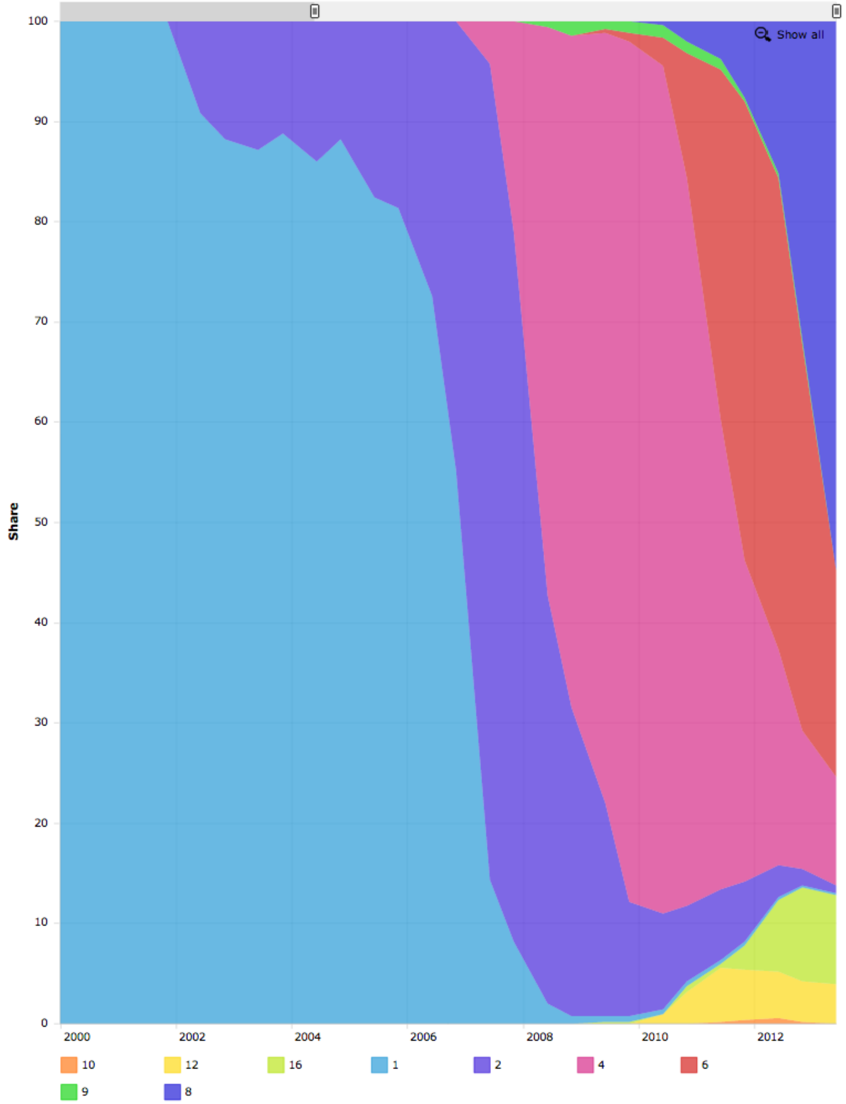
Systems share



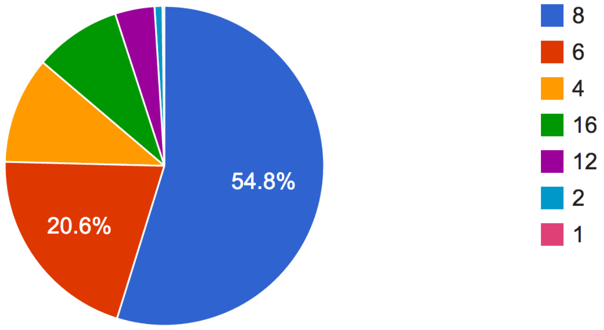
Performance share



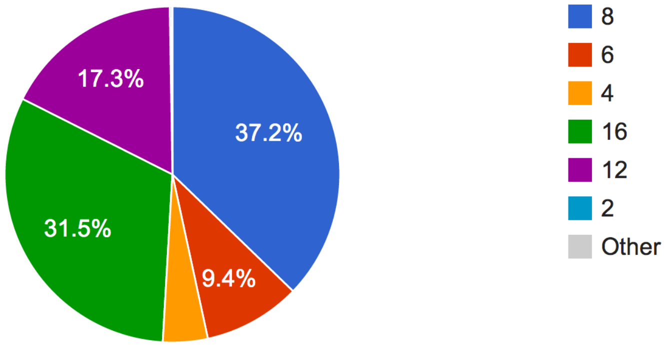
TOP 500 – Cores per Socket



Cores per Socket System Share



Cores per Socket Performance Share



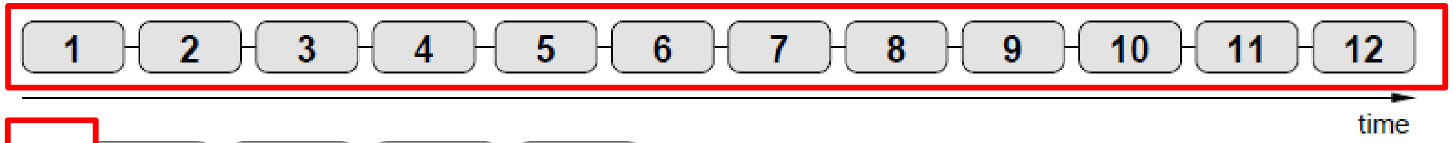
[top500.org, June 2013]

Metrics

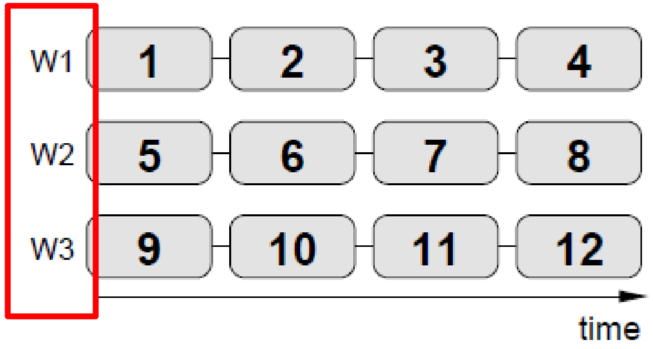
- Parallelization metrics are application-dependent, but follow a common set of concepts
  - **Speedup**: More resources lead less time for solving the same task
  - **Linear speedup:** n times more resources → n times speedup
  - **Scaleup:** More resources solve a larger version of the same task in the same time
  - **Linear scaleup:** n times more resources → n times larger problem solvable
- The most important goal depends on the application
  - Transaction processing usually heads for **throughput** (scalability)
  - Decision support usually heads for **response time** (speedup)

Speedup

W=12
'timesteps'



N = 3
Workers

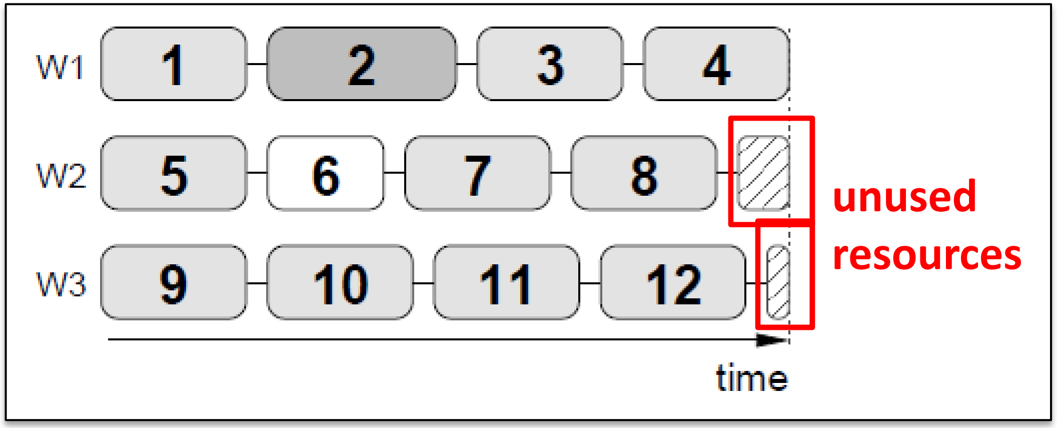


T = 'timesteps', here 12

N = # workers, here 3

Speedup:

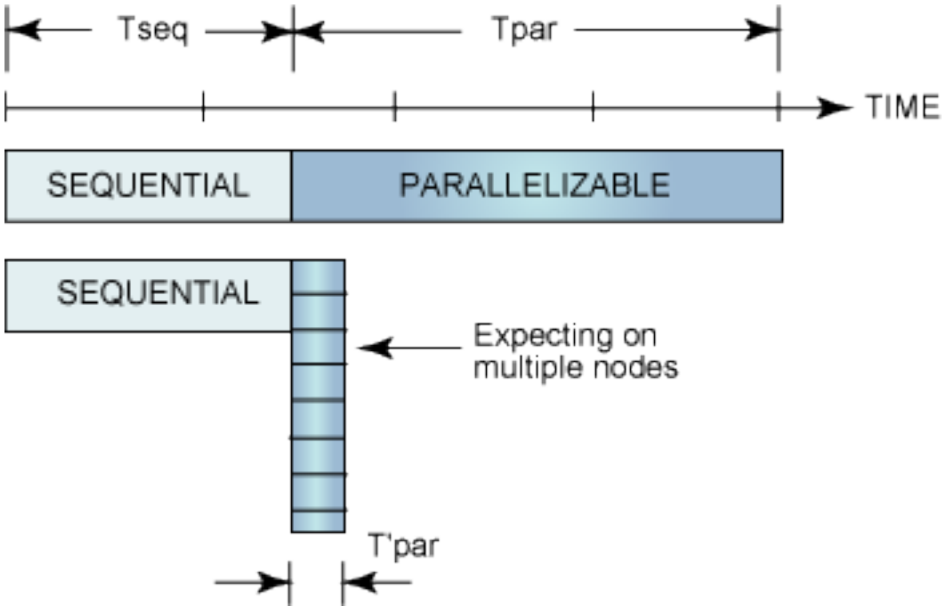
T/N = 12/3 = 4 'timesteps'



Load Imbalance

Speedup

- Each application has inherently serial parts in it
  - Algorithmic limitations
  - Shared resources acting as bottleneck
  - Overhead for program start
  - Communication overhead in shared-nothing systems



[IBM DeveloperWorks]

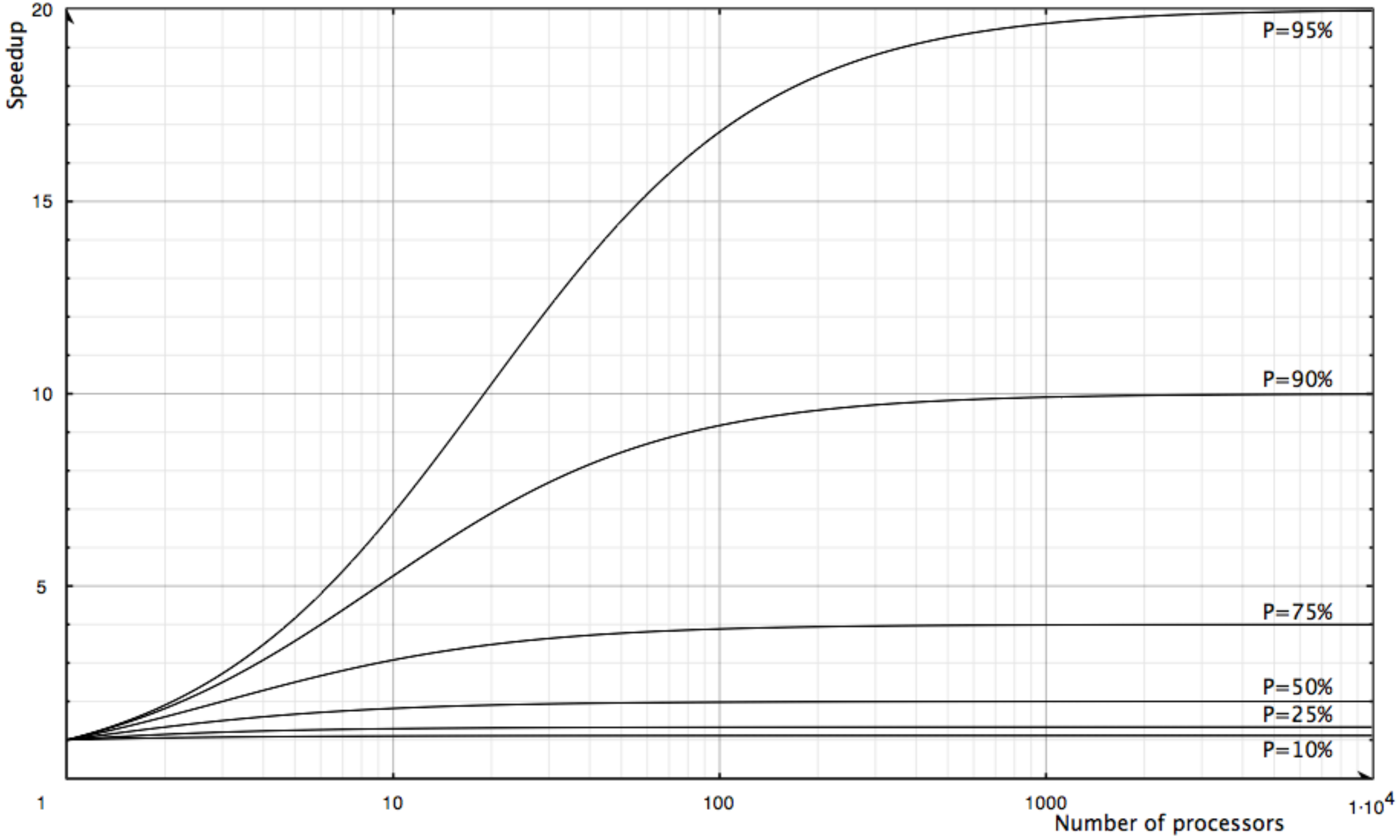
Amdahl’s Law (1967)

- Gene Amdahl expressed that speedup through parallelism is hard
  - Total execution time = parallelizable part (P) + serial part
  - Maximum speedup *s* by *N* processors:

$$s_{Amdahl} = \frac{(1-P)+P}{(1-P)+\frac{P}{N}}$$

  - Maximum speedup (for N → inf.) tends to ***1/(1-P)***
  - Parallelism only reasonable with small ***N*** or small ***(1-P)***
- Example: For getting some speedup out of 1000 processors, the serial part must be substantially below 0.1%
- Makes parallelism an all-layer problem
  - Even if the hardware is adequately parallel, a badly designed operating system can prevent any speedup
  - Same for middleware and the application itself

Amdahl's Law


Speedup
20
15
10
5
1
10
100
1000
1·10^4
Number of processors
P=95%
P=90%
P=75%
P=50%
P=25%
P=10%

Amdahl’s Law

- 90% parallelizable code leads to not more than speedup by factor 10, regardless of processor count
- Result: Parallelism is useful for small number of processors, or highly parallelizable code
- What’s the sense in big parallel / distributed machines?
- *“Everyone knows Amdahl’s law, but quickly forgets it.”* [Thomas Puzak, IBM]
- Relevant assumptions
  - Maximum theoretical speedup is N (linear speedup)
  - Assumption of **fixed problem size**
  - Only consideration of **execution time** for one problem

Gustafson-Barsis’ Law (1988)

- Gustafson and Barsis pointed out that people are typically not interested in the shortest **execution time**
  - □ Rather solve the **biggest problem** in reasonable time
- Problem size could then scale with the number of processors
  - □ Leads to larger parallelizable part with increasing N
  - □ Typical goal in simulation problems
- Time spend in the sequential part is usually fixed or grows slower than the problem size → linear speedup possible
- Formally:
  - □ P_N: Portion of the program that benefits from parallelization, depending on N (and implicitly the problem size)
  - □ Maximum **scaled speedup** by N processors:

$$s_{Gustafson} = \frac{(1-P)_N + N * P_N}{(1-P)_N + P_N} = (1-P)_N + N * P_N$$

Karp-Flatt-Metric

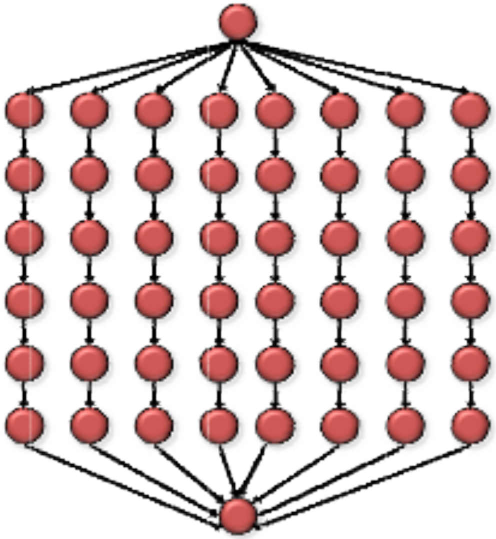
- Karp-Flatt-Metric (Alan H. Karp and Horace P. Flatt, 1990)
  - □ Measure degree of code parallelization,
    by determining serial fraction through experimentation
  - □ Rearrange Amdahl‘s law for sequential portion
  - □ Allows computation of empirical sequential portion, based on measurements of execution time, without code inspection
  - □ Integrates overhead for parallelization into the analysis
- First determine **speedup s** of the code with **N** processors
- Experimentally determined **serial fraction e** of the code:

$$e = \frac{\frac{1}{s} - \frac{1}{N}}{1 - \frac{1}{N}}$$

- If **e** grows with **N**, you have an overhead problem

Another View [Leierson & Mirman]

- DAG model of serial and parallel activities
  - □ Instructions and their dependencies
- Relationships: *precedes, parallel*
- Work *T*: Total time spent on all instructions
- Work Law: With *P* processors,
  $T_P >= T_1/P$
- **Speedup**: T_1 / T_P
  - □ **Linear**: P proportional to T_1 / T_P
  - □ **Perfect Linear**: P = T_1 / T_P
  - □ **Superlinear**: P > T_1 / T_P
  - □ **Maximum possible:** T_1 / T_{inf}



Work: $T_1 = 50$

Span: $T_\infty = 8$

Parallelism: $T_1/T_\infty = 6.25$

Examples

- Fibonacci function $F_{K+2}=F_K+F_{K+1}$
  - □ Each computed value depends on earlier one
  - □ Cannot be obviously parallelized
- Parallel search
  - □ Looking in a search tree for a ‚solution‛
  - □ Parallelize search walk on sub-trees
- Approximation by Monte Carlo simulation
  - □ Area of the square $A_S = (2r)^2 = 4r^2$
  - □ Area of the circle $A_C = pi * r^2$, so $pi = 4 * A_C / A_S$
  - □ Randomly generate points in the square
  - □ Compute A_S and A_C by counting the points inside the square vs. the number of points in the circle
  - □ Each parallel activity covers some slice of the points