2DV605

Parallel Computing

Architectures, Models, Performance

Sabri Pllana, PhD

Department of Computer Science

Building D, Room D2236C, LNU, Växjö

sabri.pllana@lnu.se, http://homepage.lnu.se/staff/saplaa/

Outline

- Parallel Computer Architectures
- Parallel Programming Models
- Performance
- Presentation Techniques

Selected Literature

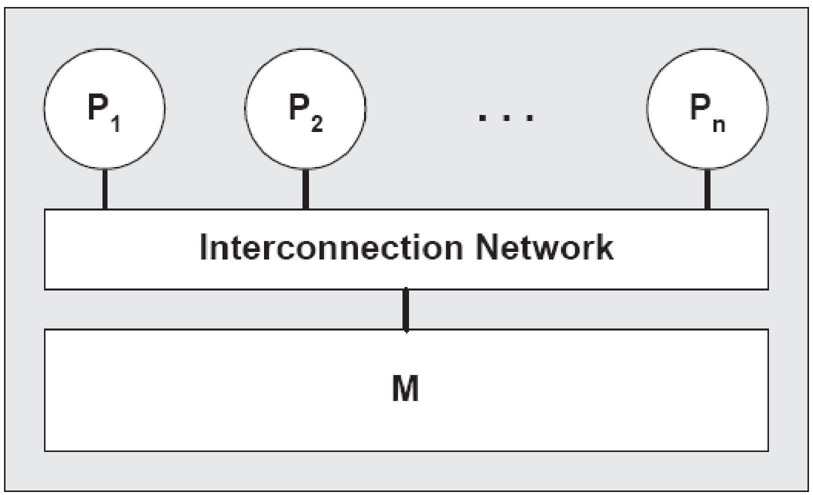
- **Textbooks**
  - J. Hennessy and D. Patterson. “Computer Architecture: A Quantitative Approach”. Morgan Kaufmann, 5th Edition, September 2011.
  - R. Jain. “The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling“. Wiley- Interscience, New York, NY, April 1991.
  - S. Pllana and F. Xhafa. "Programming multi-core and many-core computing systems", John Wiley & Sons, Inc., Hoboken, NJ, USA, 2017.

- **Performance Evaluation: Projects and Tools**
  - Paradyn Tools Project (www.cs.wisc.edu/paradyn/)
  - TAU (www.cs.uoregon.edu/research/tau/)
  - VI-HPS (http://www.vi-hps.org/tools/)

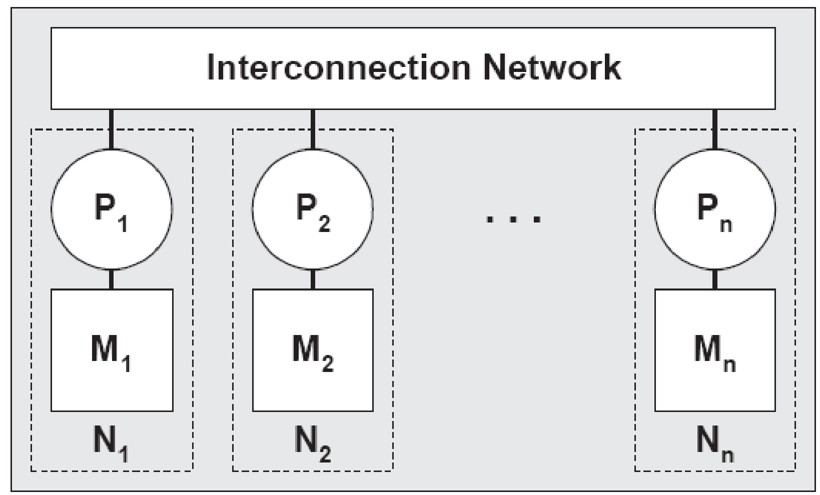
Parallel Computing Architectures

Shared and Distributed Memory Systems

- **Shared memory (SM)**
  - easy to program, but do not scale to a large number of processors
- **Distributed memory (DM)**
  - scalable, but harder to program than SM systems



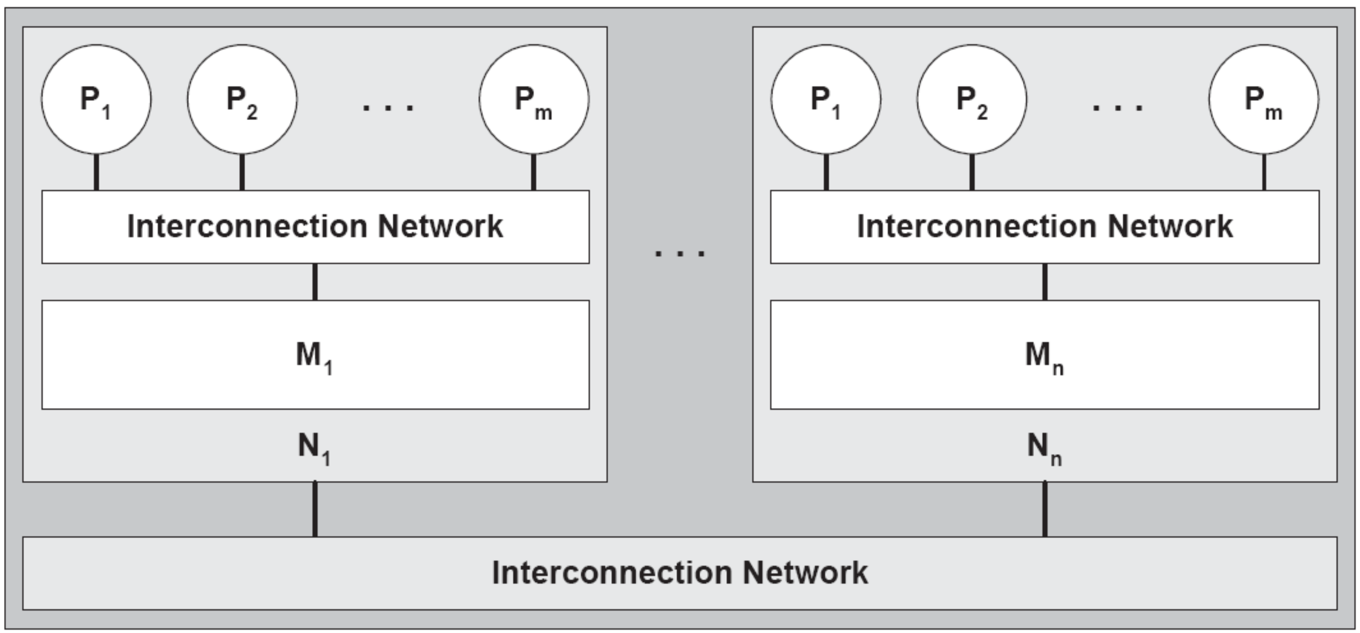
Shared memory



Distributed memory

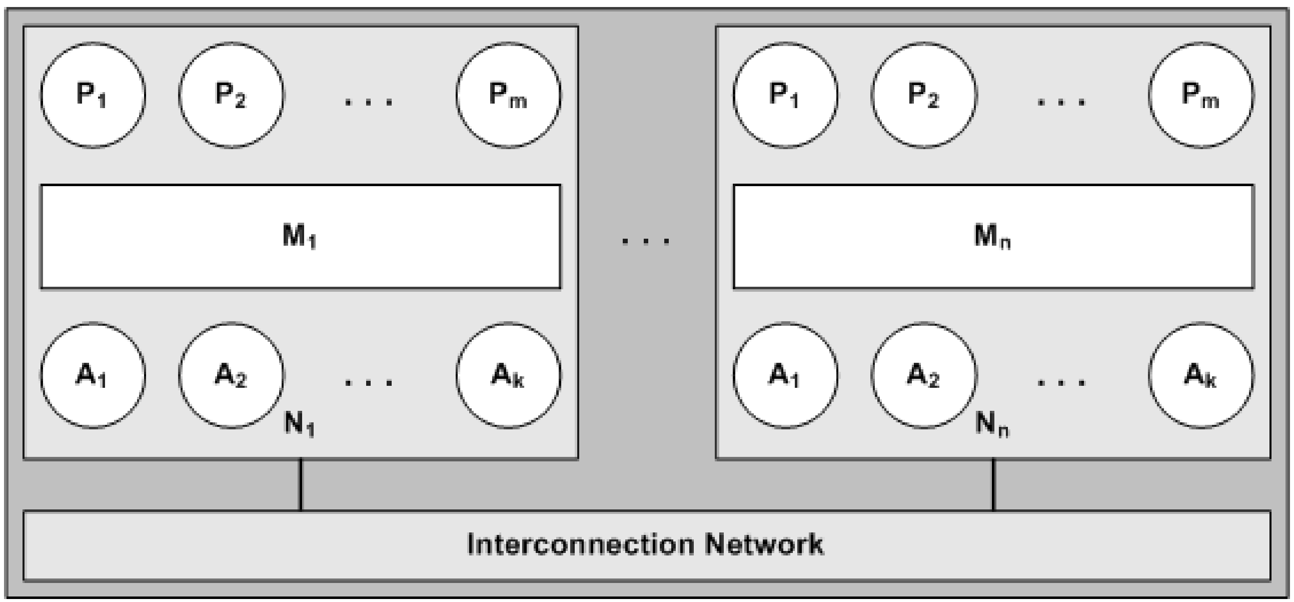
Clusters of SMPs

- **Clusters of symmetric multiprocessing (SMP) systems**
  - combine features of SM and DM systems



Accelerated Systems

- Also known as *heterogeneous* or *asymmetric systems*
- Common: GPU-accelerated systems



GPU Example: NVIDIA Volta Architecture

NVIDIA TESLA V100
Credit: NVIDIA

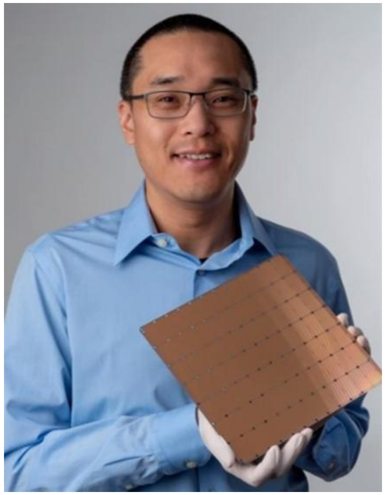
NVLink

PCIe

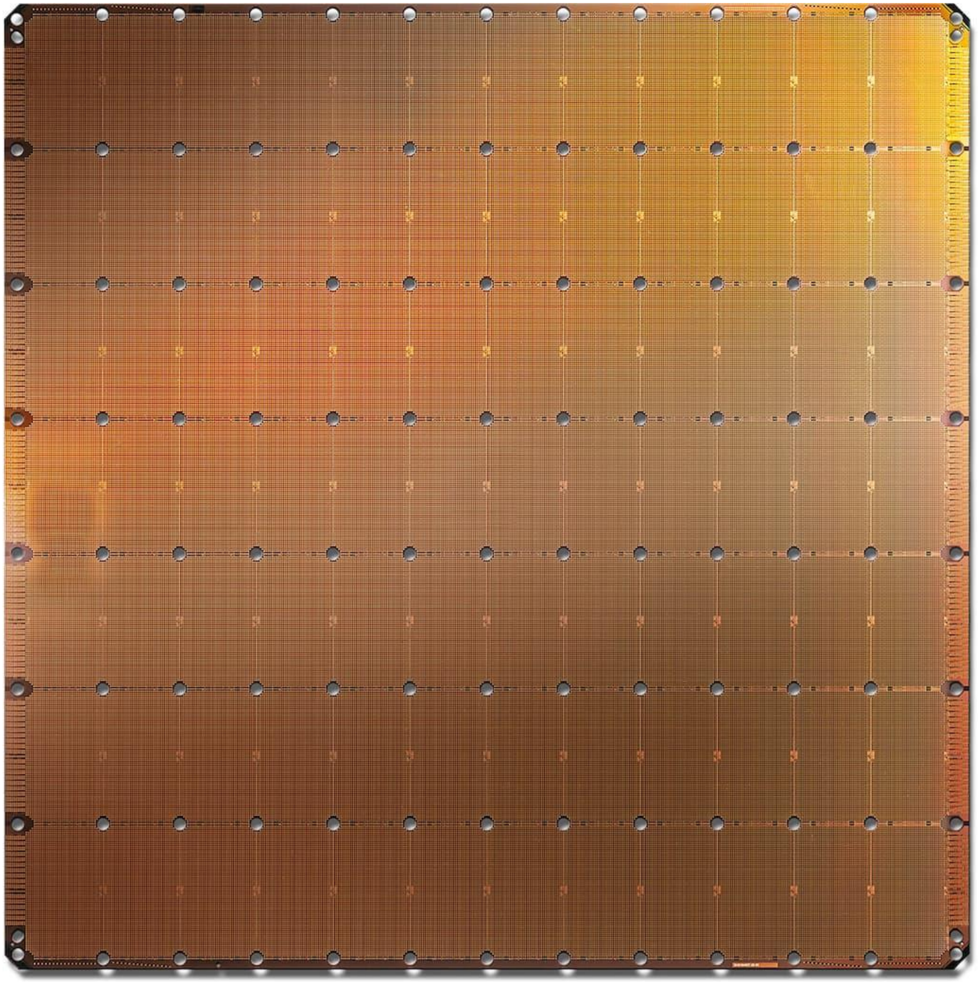
- *V100: 7.5 TF of double-precision and 15 TF of single-precision performance*
- *80 Streaming Multiprocessors (SMs)*
- *an SM has 64 FP32 cores, 64 INT32 cores, 32 FP64 cores, 8 tensor cores, 4 texture units*
- *NVLink enables communication among CPU and GPUs; 5 to 12x faster than the third generation of PCIe*

Cerebras Wafer-Scale Engine (WSE)

- *optimized for deep learning*
- *400 000 cores*
- *18 GB on-chip SRAM*
- *WSE = 56x GPU size*
- *100 – 1000 faster than NVIDIA V100 GPU*



WSE



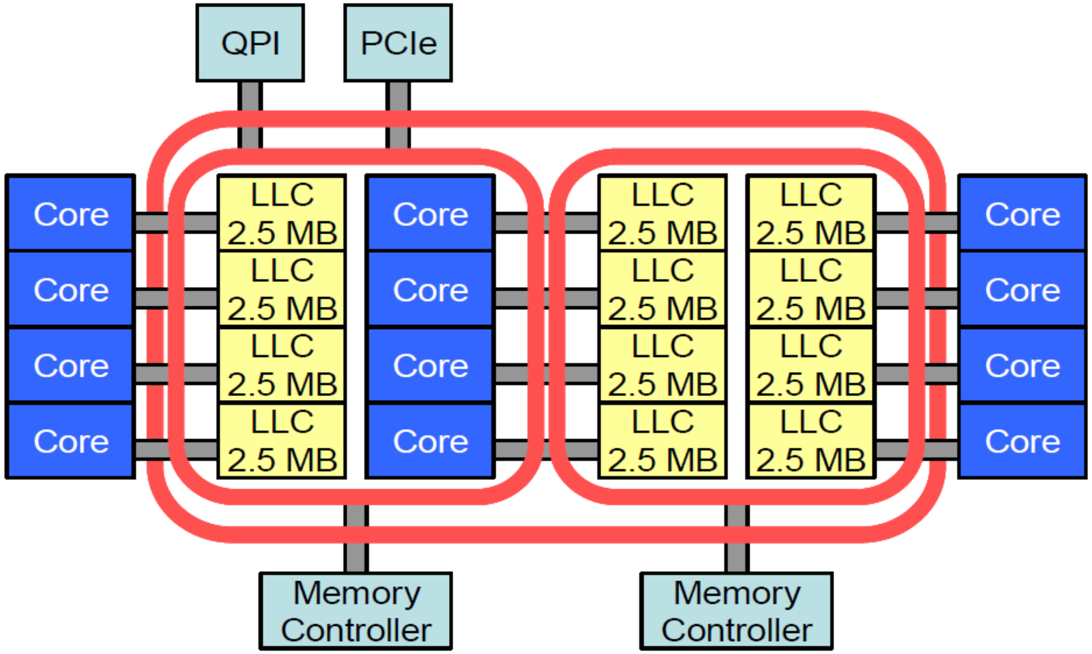
GPU

Credit: Cerebras

Intel Xeon Processor E5-2600 v2

- *Intel Xeon E5-2695v2 comprises 12 cores*
- *up to 30 MB of last level cache*

Intel QuickPath Interconnect (QPI)



Credit: IBM Systems and Technology Group

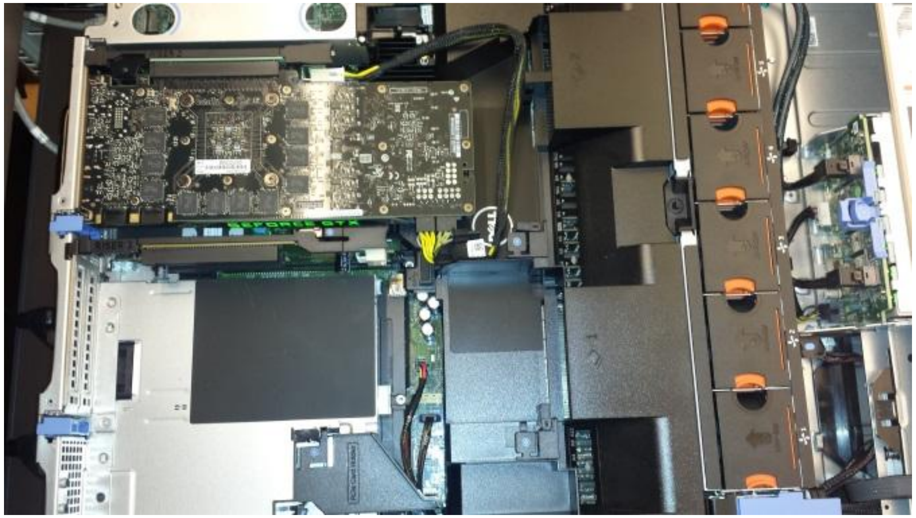
Examples: Old Clusters at the University of Vienna

Cluster of SMPs

GPU-accelerated System

LNU Parallel Computing Lab: Ida

	Ida	
Specs	Intel Xeon E5	GeForce GPU
Type	E5-2650 v4	GTX Titan X
Core Frequency	2.2 - 2.9 GHz	1 - 1.1 GHz
# of Cores	12	3072
# of Threads	24	/
Cache	30 MB	/
Mem. Bandwidth	76.8 GB/s	336.5 GB/s
Memory	384 GB	12 GB
TDP	105 W	250 W



LNU Parallel Computing Lab: DISA

- *Installed at LNU in 2018*
- *19 DELL EMC nodes:*
  - 14 PowerEdge R740
  - 5 PowerEdge R740XD
- *Various roles of nodes:*
  - homogeneous computing,
  - heterogeneous computing,
  - data storage,
  - system administration,
- *Heterogeneous nodes contain two CPUs and four GPUs*

Specs	Intel Xeon	NVIDIA GPU
Type	Gold 6148	Quadro P4000
Frequency (GHz)	2.4 - 3.7	1.48
# of Cores	20	1792
# of Threads	40	/
Cache (MB)	27.5	/
Memory (GB)	768	8
TDP (W)	150	105

Front

Inside a Node

Back

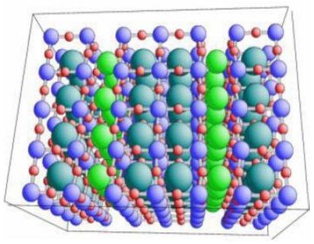
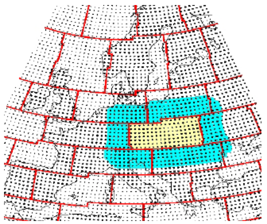
Examples: Summit at ORNL

- *Rank 1 in TOP500 list (June 2018)*
- *200 petaflops (max performance)*
- *250 petabytes of data storage*
  - 74 years of HD video
- *4608 computing nodes*
- *each node comprises*
  - two IBM Power9 22-core CPUs
  - six NVIDIA Volta GPUs
- *340 tons*
- *space of two tennis courts*

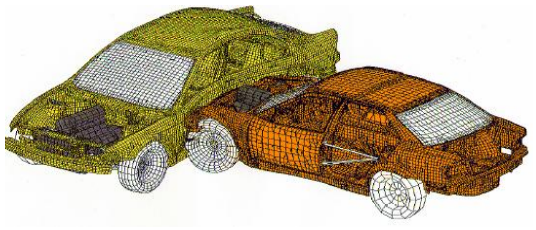
Parallel Programming Models

Software for Parallel Computing Systems

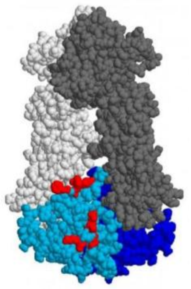
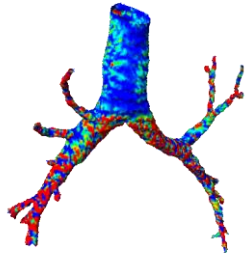
To exploit the performance of parallel computing systems, application programs have to be *parallelized*

- **Parallelization**
  - Data distribution
  - Work distribution
  - Communication/Synchronization

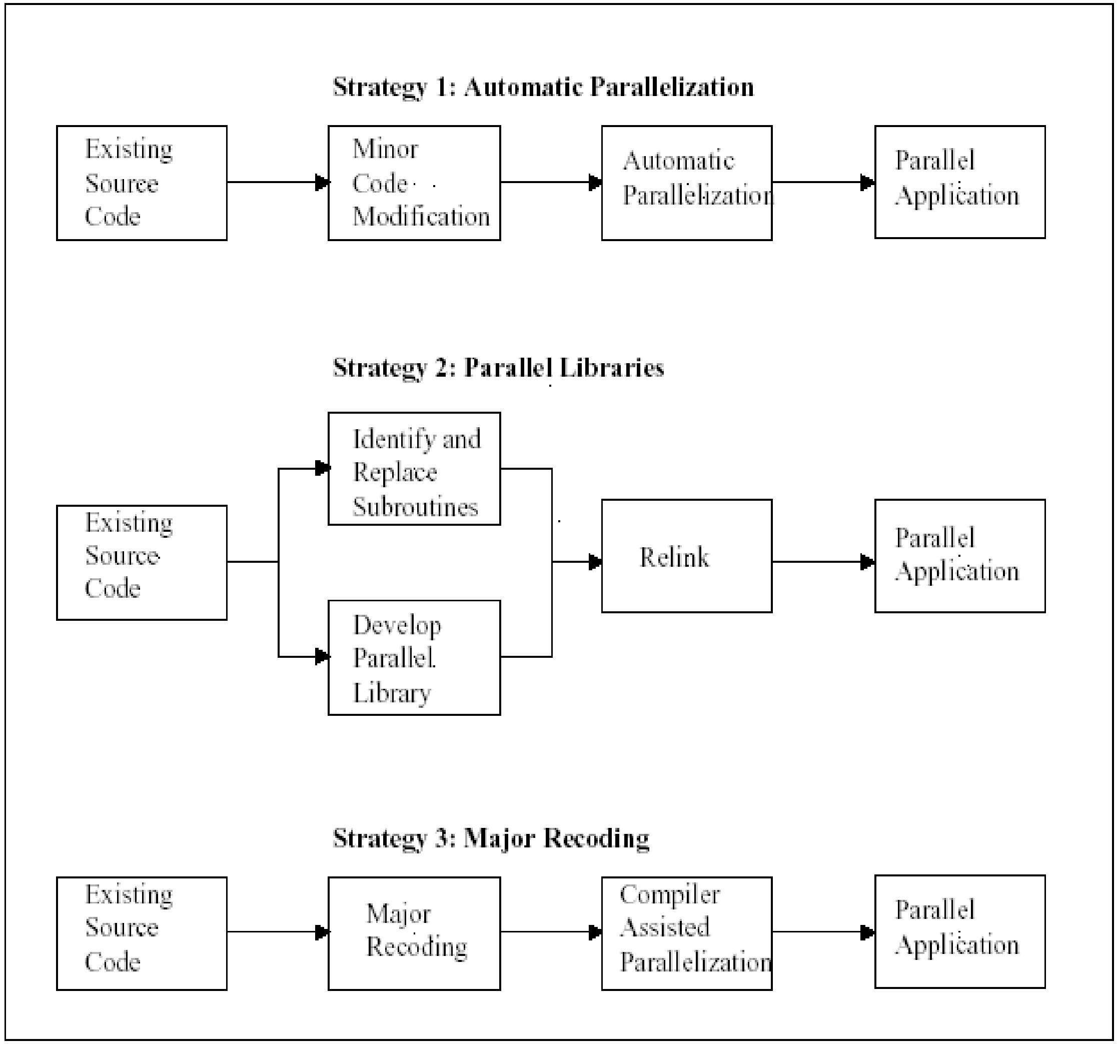


- **Parallel APIs and Languages**
  - MPI, OpenMP, OpenCL, …
- **Libraries**
  - LAPACK, ScaLAPACK, …

- **Programming parallel computing systems is complex**
  - *"no free lunch"*

Parallelization (1)



Parallelization (2)

- **Focus on two major resources**
  - processors
  - memory

- **Parallelization involves**
  - distributing work to processors
  - distributing data (if memory is distributed)

  and
  - synchronization of the distributed work
  - communication of remote data to local processor (if memory is distr.)

- **Programming models** offer a combined method for

distribution of work & data, synchronization and communication
  - Distributed-Memory Programming Models
  - Shared-Memory Programming Models

Programming Models

- **Distributed Memory Model (Message Passing Model)**
  - MPI
  - Applicable also to shared-memory machines
- **Shared Memory Model**
  - OpenMP, pThreads, Java threads,
  - Restricted to shared-memory machines
- **Hybrid Model**
  - Mixing shared and distributed memory model
  - Using OpenMP and MPI together
- **Other Programming Models**
  - **Accelerated or heterogeneous systems**: CUDA, OpenCL, OpenACC
  - Partitioned Global Address Space (PGAS): Chapel, X10
- **Object and Service Oriented Models**
  - OO: RPC/RMI
  - Web Services (Grid or Cloud computing)

Work & Data Distribution

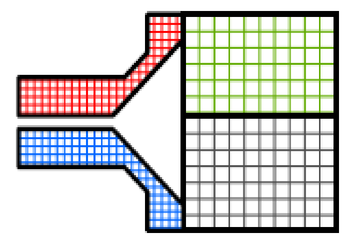
- **Work distribution**
  - usually based on loop decomposition

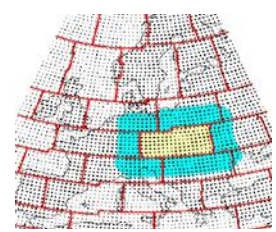
```
do i=1,100
→ i=1,25
  i=26,50
  i=51,75
  i=76,100
```

- **Data distribution**
  - all work for a local portion of the data is done by the local processor

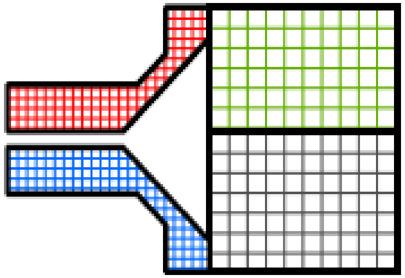
```
A( 1:20,  1: 50)
A( 1:20, 51:100)
A(21:40,  1: 50)
A(21:40, 51:100)
```

- **Domain decomposition**
  - decomposition of work and data is done in a higher model, reflecting reality





Single-Program Multiple-Data (SPMD)



Domain Decomposition

1. Divide domain (=data) into different parts.
2. Assign parts to different processors.

Owner-Computes Rule

- Each data point is owned by some processor
- The owner of a data point is responsible for computing its value.
- Communication (explicitly or implicitly) is performed, if computation of a data point requires data from other processors.

Domain decomposition typically leads to SPMD structure.

Single-Program Multiple-Data (SPMD)

- Each processor executes the same code, but on different data
- Scalar variables are typically replicated on all processors and redundantly computed on each of them.

Communication

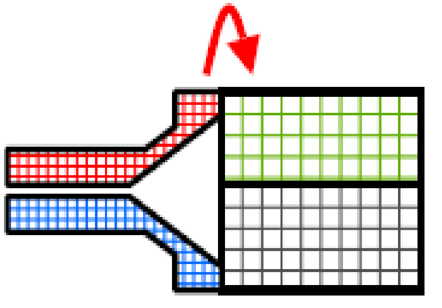
```
Do i=2,99
    b(i) = a(i) + f*(a(i-1)+a(i+1)-2*a(i))
Enddo
```

Communication is necessary on the boundaries

- **e.g.** b(26) = a(26) + f*(a(25)+a(27)-2*a(26))

```
A(1:25)
A(26:50)
A(51:75)
A(76:100)
```

- **e.g. at domain boundaries**



Synchronization

Synchronization

- is needed in some contexts
- may cause
  - idle time on some processors
  - overhead to execute the synchronization primitive

```
Do i=1,100                    i=1..25 | 26..50 | 51..75 | 76..100
    a(i) = b(i)+c(i)              execute on the 4 processors
Enddo
---------------------------- BARRIER synchronization
Do i=1,100
    d(i) = 2*a(101-i)         i=1..25 | 26..50 | 51..75 | 76..100
Enddo                             execute on the 4 processors
```

Dependence

- Dependence is a relationship that places constraints on the *execution order of computations*
  - determines wether or not computations can be executed in parallel.

- There are two types of dependences
  - Control dependence

S1: if (b ≠ 0) then
S2: c = a / b;

  - Data dependence, relation on the set of statements

S1: a = …
S2: c = a / b;

MPI and OpenMP

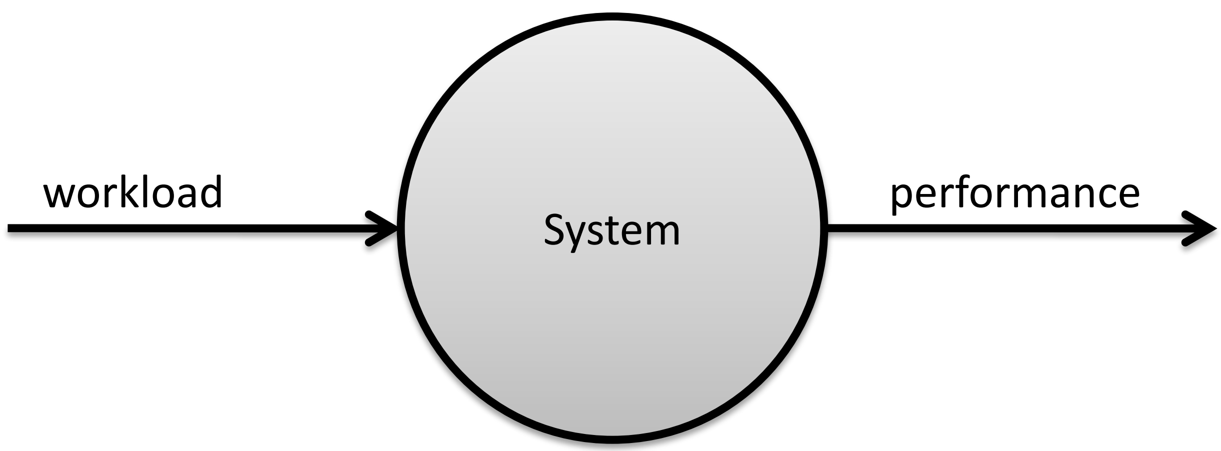
- **MPI (Message Passing Interface)**
  - user specifies how work & data is distributed
  - user specifies how and when communication/synchronization is done
  - by calling MPI communication library-routines
  - applicable to distributed- and shared-memory architectures

- **OpenMP**
  - shared memory directives to define the work decomposition
  - high-level multi-threading
  - restricted to shared-memory systems
  - synchronization is implicit (can be also user-defined)

Performance

System Performance

- **Performance = f (system, workload)**
- **In our case:**
  - workload ~ computer program
  - system ~ computer system
  - performance ~ program execution time, power consumption



Performance of Computing Systems

- **Expressed in floating-point operations per second (FLOPS)**

- **Peak performance**
  - theoretical maximal number of FLOPS
  - all components of the system are completely utilized

- **Sustained Performance**
  - obtained FLOPS when a certain program is executed

Performance of Processor

- **Clock frequency of processor**
  - speed at which are performed the fundamental operations
  - for instance: add operation
  - clock frequency is expressed in cycles per second
  - one cycle per second corresponds to one hertz (Hz)
  - 1GHz: 1 000 000 000 (or 10^9) cycles per second
  - clock frequency is also known as the speed of processor

- **Performance depends not only on clock frequency**
  - architecture of processor is important

Program Execution Time

- **Clock frequency: F**
- **Number of clock cycles: NC**
- **Number of instructions: NI**
- **Clocks per instruction: CPI = NC / NI**

- **Program execution time = NI * CPI * 1/F**
  - also known as Iron law

- **Average time per instruction: TI_{av} = CPI * 1/F**
  - program execution time = NI * TI_{av}

Memory Performance

- **Data transfer between the memory and processor**

- **Latency (also known as delay)**
  - time between the request and the retrieve of a data element from the memory

- **Bandwidth**
  - quantity of data that can be transferred between the processor and the memory within one second

Network Performance

- **Data transfer among the nodes via the interconnection network**

- **Latency (also known as delay)**
  - time that is needed to transfer a data element between two nodes of network

- **Bandwidth**
  - number of bits of data that can be transferred between two nodes within one second

Speedup

- Number of processors: NP
- Program execution time on one processor: T_1
- Program execution time on NP processors: T_{NP}

- Speedup: $$Speedup(NP) = \frac{T_1}{T_{NP}}$$

- Linear speedup: Speedup(NP) = NP

- Super linear: Speedup(NP) > NP

Efficiency

- **Number of processors: NP**

$$Efficiency(NP) = \frac{Speedup(NP)}{NP}$$

Scalability

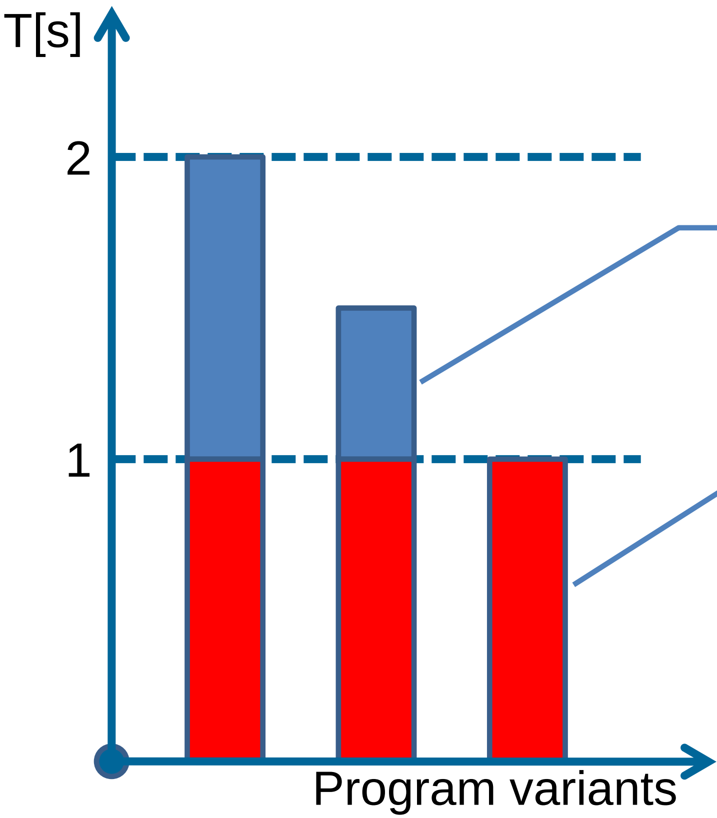
- **Performance improvement is proportional to the increase of the number of processors**
- **Theoretical maximal speedup can be estimated using Amdahl's law**
  - fraction of computation that is performed sequentially: σ
  - fraction of computation that is performed in parallel: 1- σ

$$Speedup(\sigma, NP) = \frac{1}{\sigma + \frac{1-\sigma}{NP}}$$

  - maximal speedup depends only on the fraction of computation that is performed sequentially

$$\lim_{NP \to \infty} \frac{1}{\sigma + \frac{1-\sigma}{NP}} = \frac{1}{\sigma}$$

Speedup and Amdahl's law



"*the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude*" - Gene M. Amdahl, 1967

Fraction improved by factor of 2 (parallel processing)

Fraction is not improved (sequential)

Speedup $= T_{\text{original program}} / T_{\text{improved program}}$

Performance Evaluation Methods

- **Measurement**
  - trustfulness: results obtained on a real system
  - requires that the system under study is available
- **Mathematical modeling**
  - represents the system as a set of symbolic expressions
  - lacks the structural information (type/relationship of SW/HW)
- **Simulation**
  - imitates the behavior of system under study
  - commonly implemented as a computer program
  - may be time-consuming

Benchmarks

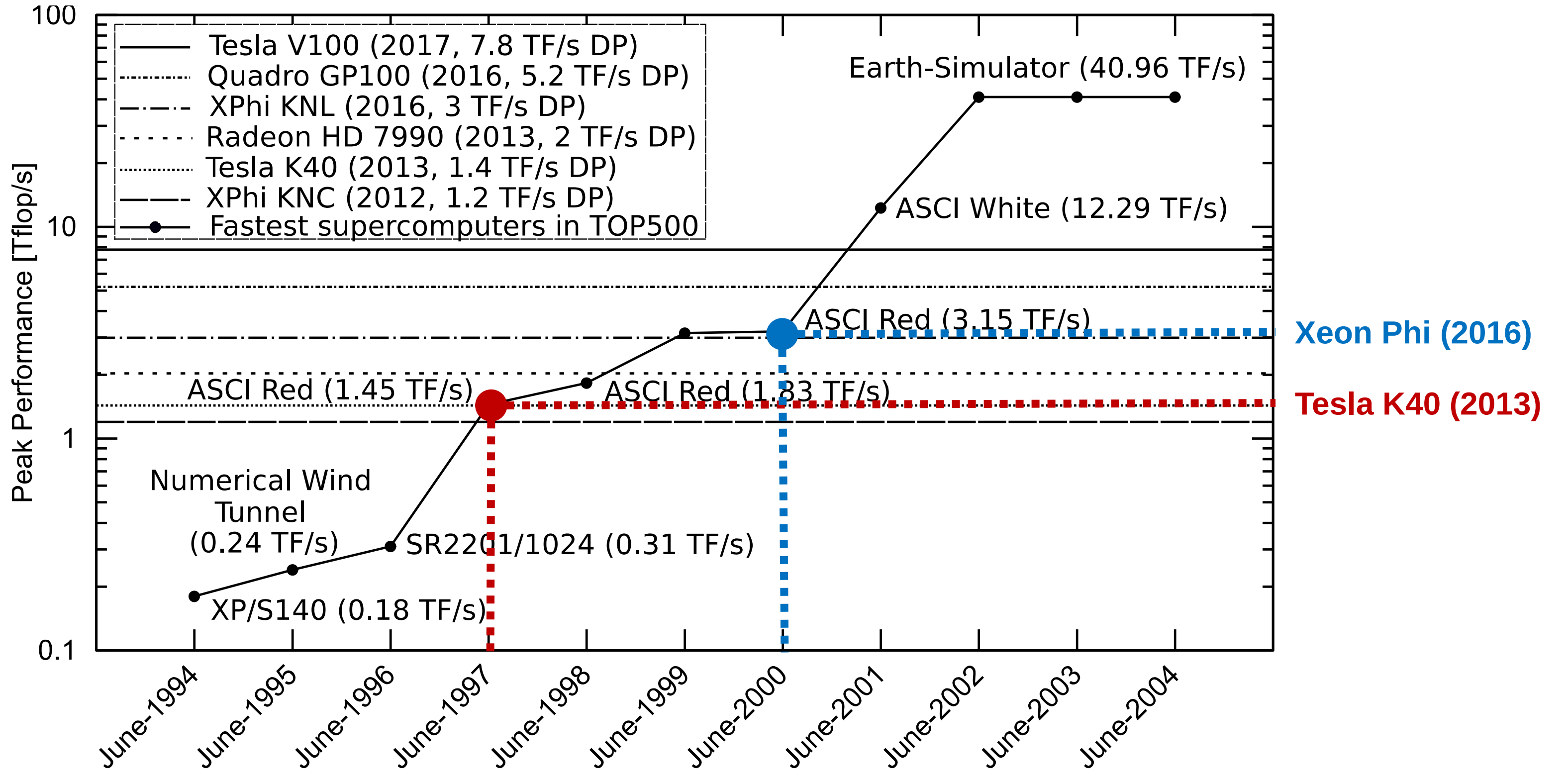
- **Benchmarks are computer programs**
  - artificial (instruction mixes) or real programs
  - measure various aspects (computation, communication, power)
- **LINPACK benchmark**
  - solves dense systems of linear equations
  - is used to rank 500 most powerful computing systems
  - TOP500 is updated two times per year (www.top500.org)
- **Standard Performance Evaluation Corporation (SPEC)**
  - SPEC benchmarks: CPU, MPI/OMP, Power, …
  - www.spec.org

June 2019
www.top500.org

- **Rmax** is maximal performance achieved in the High Performance Computing LINPACK benchmark
- **Pflop/s** or petaflop: 10^{15} FLOPS

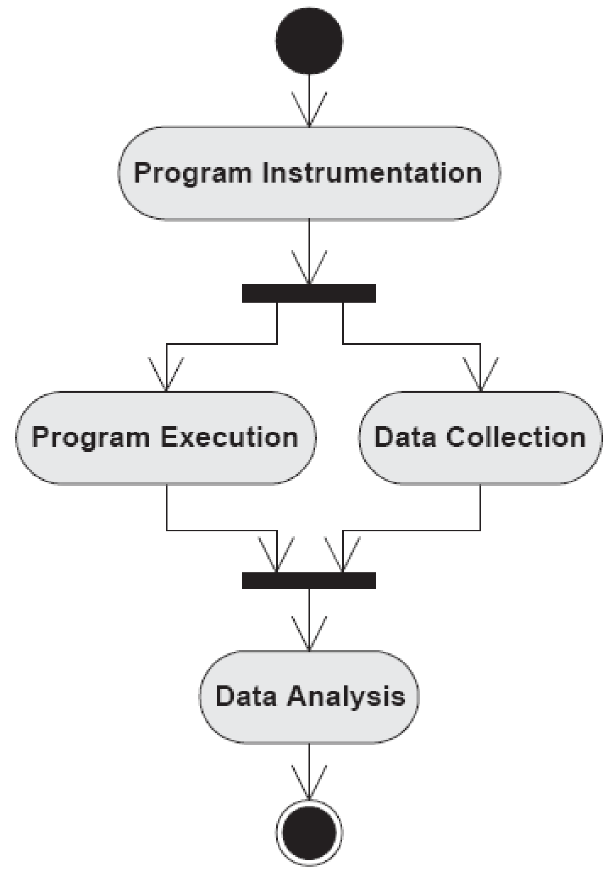
		SPECS	SITE	COUNTRY	CORES	RMAX PFLOP/S	POWER MW
1	**Summit**	IBM POWER9 (22C, 3.07GHz), NVIDIA Volta GV100 (80C), Dual-Rail Mellanox EDR Infiniband	DOE/SC/ORNL	USA	2,414,592	148.6	11.4
2	**Sierra**	IBM POWER9 (22C, 3.1GHz), NVIDIA Tesla V100 (80C), Dual-Rail Mellanox EDR Infiniband	DOE/NNSA/LLNL	USA	1,572,480	94.6	7.44
3	**Sunway TaihuLight**	Shenwei SW26010 (260C, 1.45 GHz) Custom Interconnect	NSCC in Wuxi	China	10,649,600	93.0	15.4
4	**Tianhe-2A (Milkyway-2A)**	Intel Ivy Bridge (12C, 2.2 GHz) & TH Express-2, Matrix-2000	NSCC Guangzhou	China	4,981,760	61.4	18.5
5	**Frontera**	Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR	TACC/U of Texas	USA	448,448	23.5	-

Performance of Accelerators and #1 in TOP500



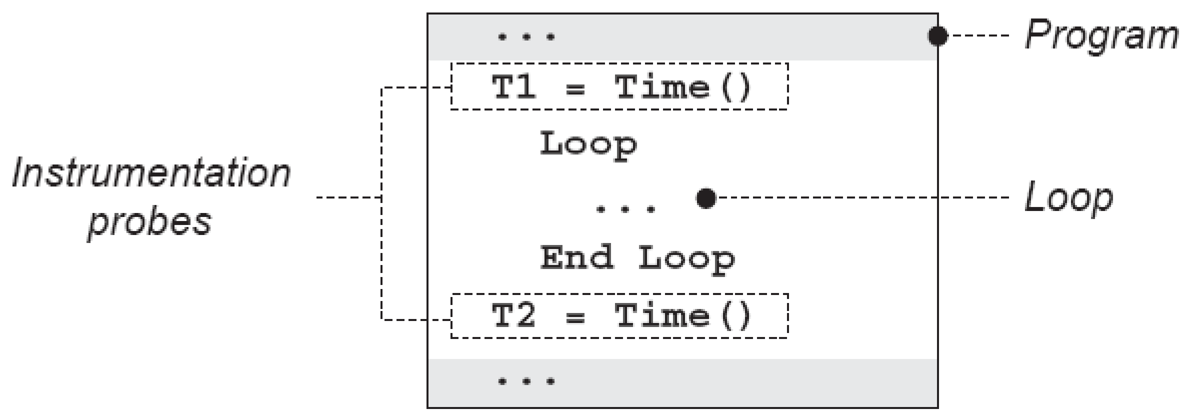
Program Performance Measurement

- **Common activities**
  - instrumentation
  - execution
  - data collection
  - analysis



Program Instrumentation

- **Instrumentation probes**
  - program statements
  - read and store the current time
  - may be done manually or automatically
  - loop execution time: $T_{Loop} = T_2 - T_1$



Data Collection

- **Examples of the collected data**
  - CPU time
  - number of floating point instructions
  - number of times a function is executed
  - number of load/store instructions

- **Data sources**
  - instrumented program
  - operating system (CPU utilization)
  - hardware performance counters (L2 requests)

Data Analysis: Average Value

- **Multiple program executions**
  - same program on the same machine
  - may produce various performance results
  - outliers: data values that differ significantly from other values
  - outliers are ignored if there is no reasonable explanation

- **Mean M of a set of measurements {x_1, ..,x_N}**

$$M = 1/N \sum_{i=1}^{N} x_i$$

Data Analysis: Variance and Standard Deviation

- **Variance**

$$S^2 = 1/(N-1)\sum_{i=1}^{N}(x_i - M)^2$$

- **Standard deviation**

$$S = \sqrt{1/(N-1)\sum_{i=1}^{N}(x_i - M)^2}$$

Data Analysis: Standard Error

- **Various sets of measurements**
  - may result with various means M

- **Standard error**
  - is the standard deviation of the mean M

- **Standard error of a set of measurements $\{x_1, .., x_N\}$**

$$S_M = S/\sqrt{N}$$

Data Analysis: Systematic Error

- **Biased measurements**
  - are reproducible
  - are always biased in the same direction (too large or too small)
- **Possible sources of systematic errors**
  - instrumentation issues: overhead, imperfection,..
  - methodological mistakes
- **Systematic errors are hard to detect and correct**
  - can not be corrected by simply repeating the experiment
  - may be helpful to try various measurement tools and methods or use calibration

Data Analysis: Confidence Interval (1)

- Assume normal distribution of measurement results
  - mean: μ
  - standard deviation: σ

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- Confidence interval of $k\sigma$ is calculated as follows

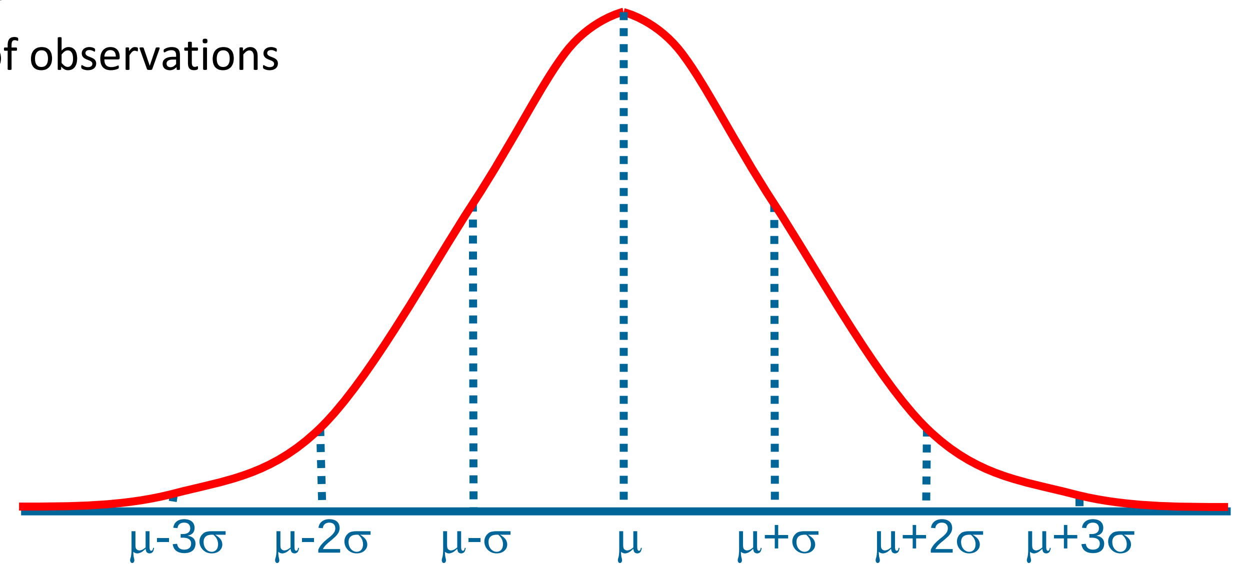
$$P(\mu - k\sigma < x < \mu + k\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \int_{\mu-k\sigma}^{\mu+k\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\, dx$$

- Commonly used intervals
  - 95% corresponds to 1.96σ
  - 99% corresponds to 2.58σ
  - higher levels of confidence imply larger confidence intervals

Data Analysis: Confidence Interval (2)

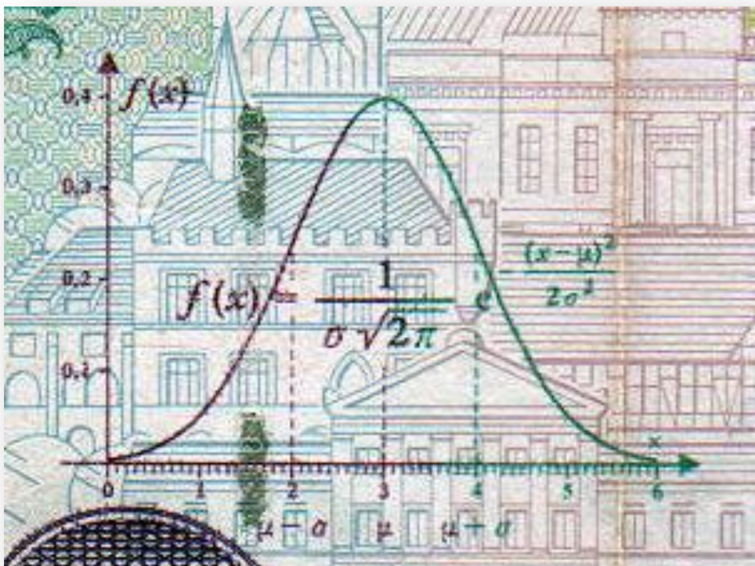
- **Data analysis**
  - repeat experiments to deal with performance fluctuation
  - average value, outliers, variance, standard deviation and error

- **Confidence intervals**
  - $\mu \pm \sigma$, 68.3% of observations
  - $\mu \pm 2\sigma$, 95.4%
  - $\mu \pm 3\sigma$, 99.7%



Normal Distribution (or Gaussian Distribution)

- Carl Friedrich Gauß (1777 – 1855)
- central limit theorem: *the mean of a large set of independent random variates is distributed approximately normally*



Math is everywhere!

EPCC OpenMP Benchmark

- **Comparison of OpenMP implementations**
- **EPCC: Edinburgh Parallel Computing Center**
- **Estimation of OMP overheads: T_{ovr}**
  - parallel execution: T_{par}
  - sequential execution: T_{seq}
  - number of processors: NP

$$T_{ovr} = T_{par} - \frac{T_{seq}}{NP}$$

EPCC OpenMP Benchmark: Example

- **PARALLEL DO overhead**
  - specifies that iterations of the loop should be executed in parallel

nthreads = 4

```
      do k = 0, outerreps
        start = getclock()
        do j = 1, innerreps
!$OMP PARALLEL DO
          do i = 1, nthreads
            call delay(dl)
          enddo
!$OMP END PARALLEL DO
        end do
        time(k) = (getclock() - start) * 1.0e6 / dble (innerreps)
      end do
```

Code example

overhead of PARALLEL DO is 5.16 ± 0.21 microseconds

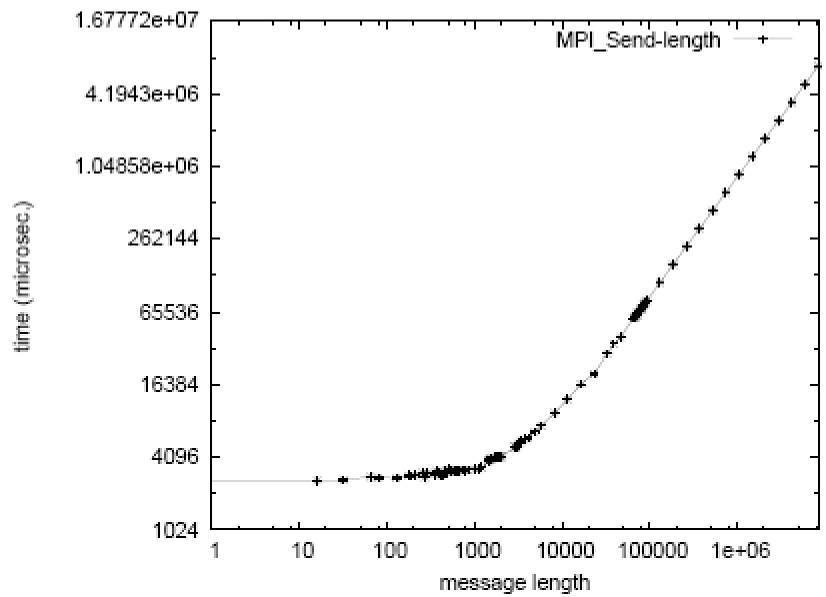
```
Sample_size       Average     Min         Max         S.D.       Outliers
         50       11.10000    11.00000    11.20000    0.05714           0

PARALLEL_DO time =                       11.10 microseconds +/-   0.112

PARALLEL_DO overhead =                    5.16 microseconds +/-   0.210
```

Result

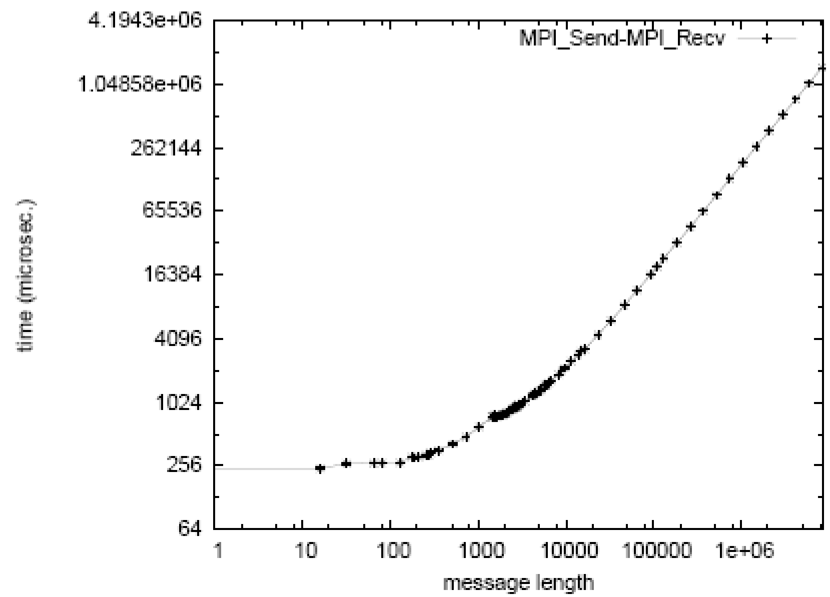
SKaMPI Benchmark

- **Assessment of various MPI implementations**
- **SKaMPI: Special Karlsruher MPI Benchmark**
- **Support**
  - point-to-point communication
  - collective communications

SKaMPI Benchmark: MPI Send, MPI Receive



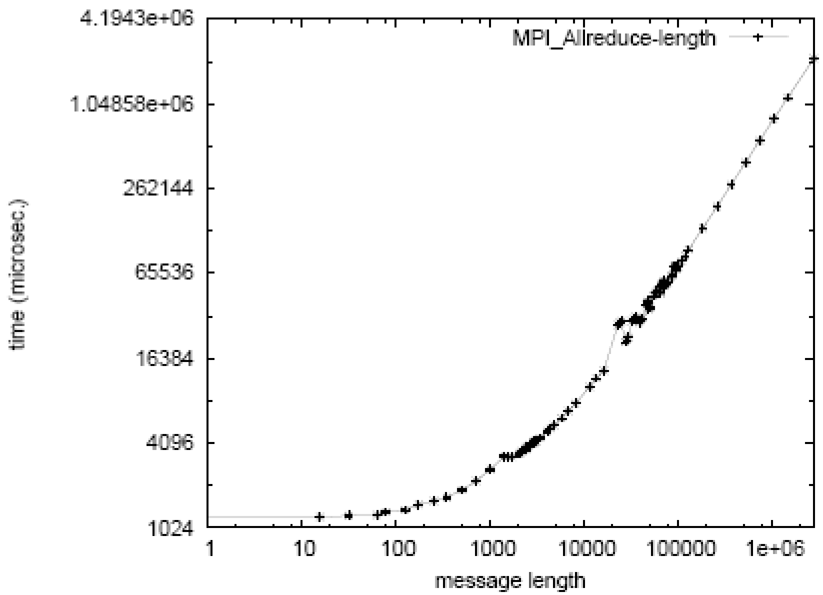
execution time of MPI SEND operation for various message lengths



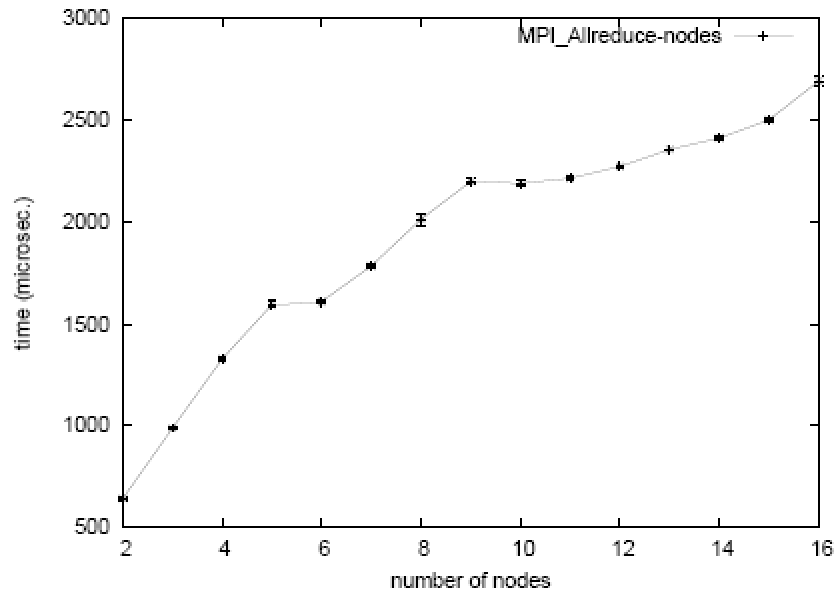
time needed to SEND and RECEIVE messages of various lengths

SKaMPI Benchmark: MPI Allreduce

performs a specific *operation (such as sum, min, max) on supplied data,* and broadcasts the result back to all involved processes



MPI Allreduce for various message lengths



MPI Allreduce for various number of nodes

Presentation Techniques

Presentation of Results

- **We provide several recommendations**
  - based on the Chapter 10 of the book "*The Art of Computer Systems Performance Analysis*" of Raj Jain
- **We expect that you will use these techniques for presenting your results**
  - quality and completeness of your presentation will be assessed
- **Worth to read**
  - "*Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers*" by David H. Bailey
  - Slides: http://www.davidhbailey.com/dhbtalks/dhb-12ways.pdf
  - Paper: http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/twelve-ways.pdf

Graphic Charts

- **Line chart**
  - x-axis: cause or independent variable
  - y-axis: effect or dependent variable
- **Bar chart**
- **Pie chart**
- **Histograms**
  - visualizes distribution of data
- **Gantt chart**
  - may be used to visualize activities of parallel processes
- **Kiviat chart**
  - two-dimensional representation of more than two variables

Variables

- **Qualitative**
  - ordered: type of computers (notebook, supercomputer)
  - unordered: types of programs (entertainment, education)

- **Quantitative**
  - discrete: integer number (number of processes or threads)
  - continuous: real number (execution time)

- **Type of variable determines the type of graphic chart**
  - line chart is used when x and y are continuous variables
  - bar chart is suitable for discrete x variables

Guidelines for Graphic Charts

- **Minimize the reading effort**
  - should be easy to understand the chart
- **Maximize the conveyed information**
  - use understandable key words; label properly axes
- **Minimize ink**
  - consider removing grid lines from the graph
- **Use common practices**
  - show dependent variables using y-axis
- **Avoid ambiguity**
  - identify/label individual elements of the chart

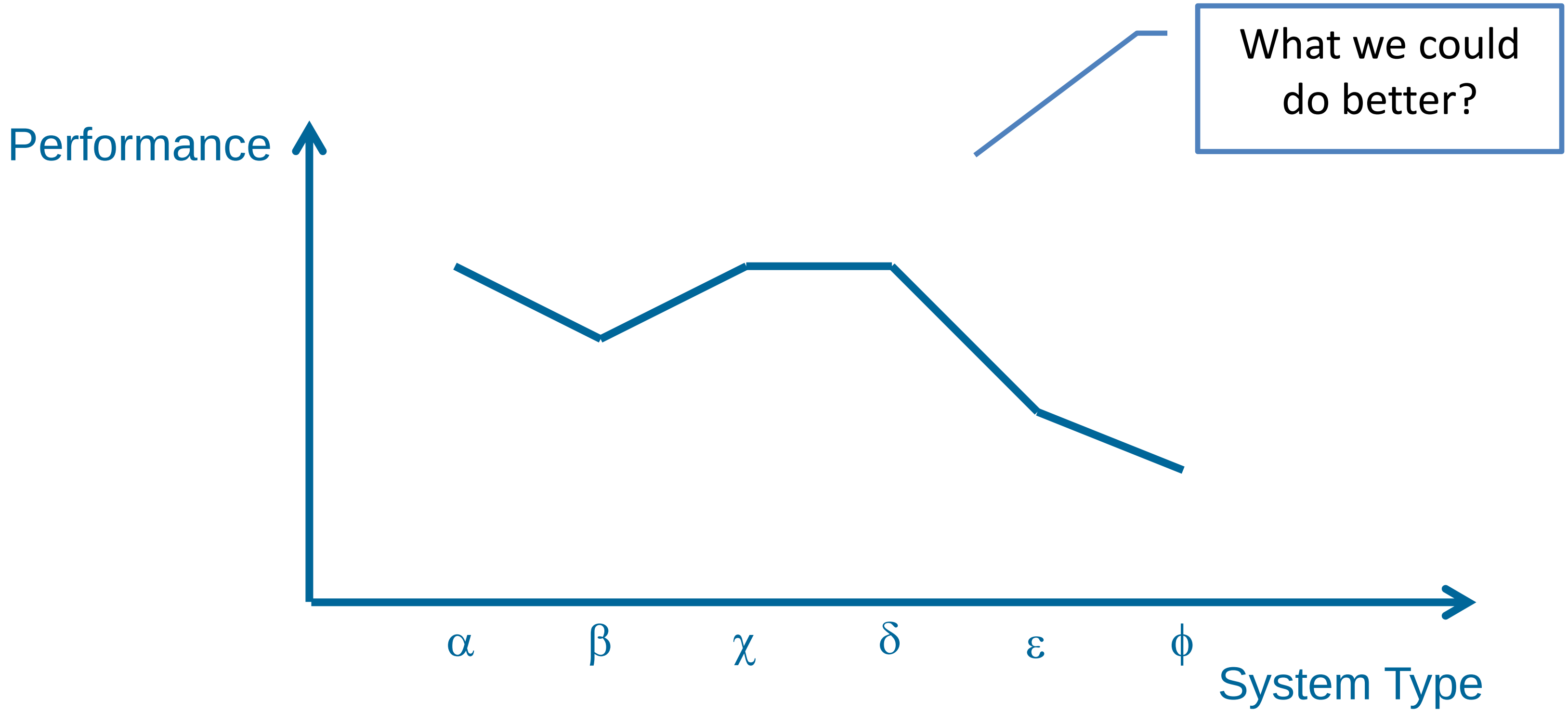
Checklist for Graphical Presentation of Results

- **A selection from Page 143, Chapter 10**
  - Are both axes labeled
  - Are the labels self-explanatory
  - Are the scales and divisions shown on both axes
  - Are the curves on a line chart individually labeled
  - Are the units of measurement indicated
  - Is the horizontal scale increasing from left to right
  - Is the vertical scale increasing from bottom to top
  - Are the grid lines aiding in reading the curves
  - Does this whole chart add to information available to the reader

Common Mistakes in Preparing Graphic Charts

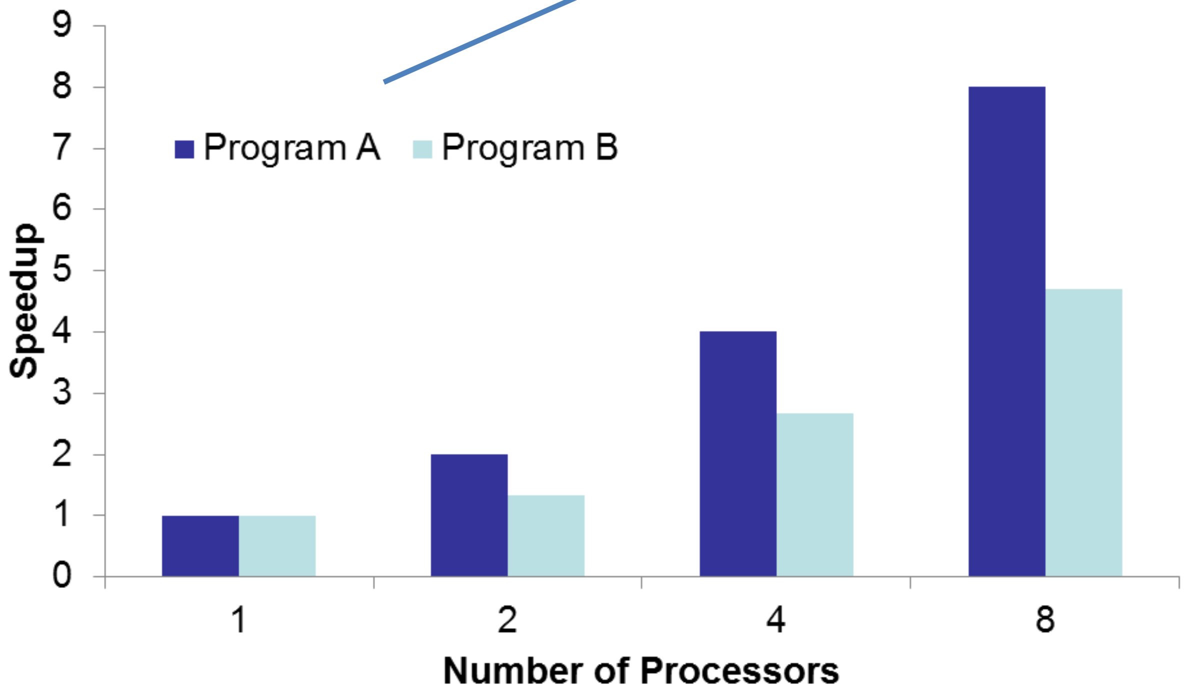
- **Presenting too many alternatives on a single chart**
  - avoid too many curves or bars ("*small is beautiful*")
- **Using symbols in place of text**
  - the reader should not be forced to decipher the meaning of symbols to understand the chart
- **Placing extraneous information on the chart**
  - avoid any detracting elements (grid lines)
- **Selecting scale ranges improperly**
  - select appropriate minimum and maximum values
- **Using not the appropriate type of chart**
  - avoid using a line chart instead of bar chart

Common Mistakes: Example



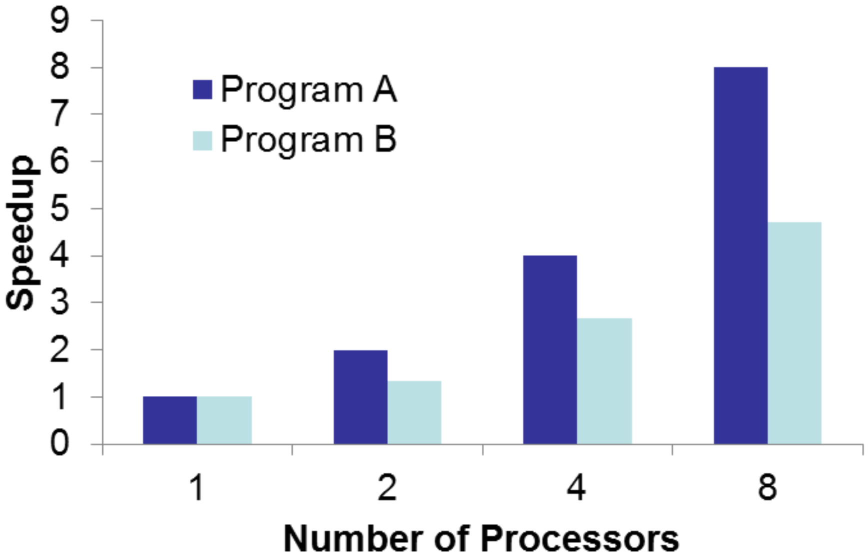
Showing Only Speedup Results is not Enough

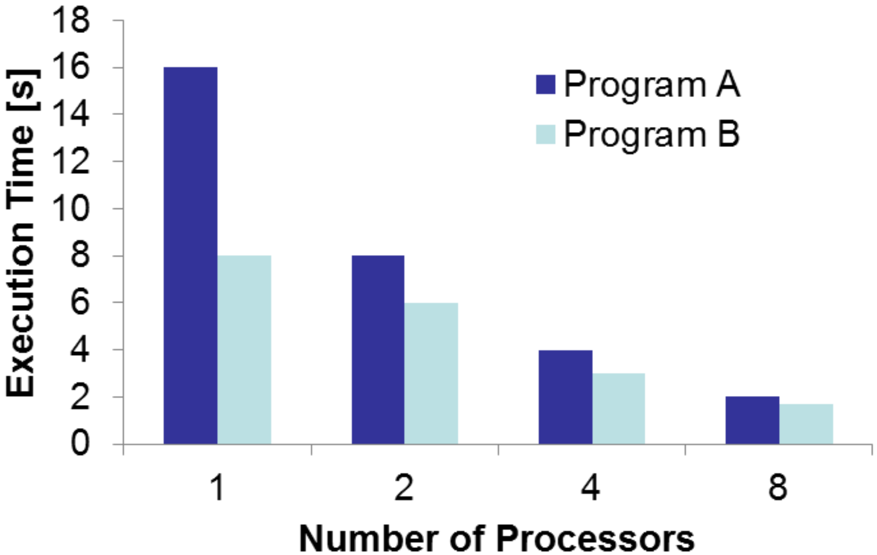
Which program is better?



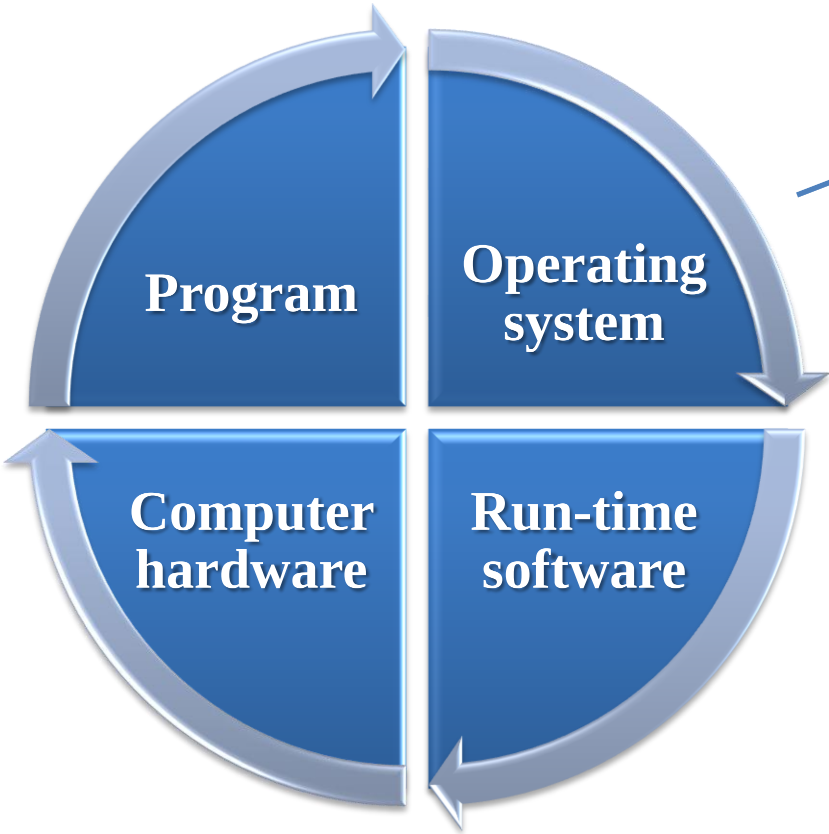
Speedup and Execution Time Results

Which program is better?





Describe the Experimentation Platform



When we report performance results, we usually specify the experimentation platform: program, OS, HW, RT software, compiler type and parameters

Selected Linux Commands

- Environment information
  - **env**
- Operating system information
  - **uname –a**
- CPU information
  - **cat /proc/cpuinfo**
- Memory information
  - **cat /proc/meminfo** or
  - **free -m**

Command "time"

- **A UNIX command**
  - measures program execution time
- **Displays the following information**
  - user time (time CPUs spent for executing the program in user space)
  - system time (time CPUs spent for system calls in kernel mode)
  - real time (or wall clock time) between program invocation and termination
- **Example**
  - time mpirun -np 4 matvec_1000
  - 0.32u 0.10s 0:00.19

Why the user time plus system time is larger than the real time?

Command “perf”

- **A Linux performance measurement program**
  - example: perf stat -e cache-misses ./matmul
  - example: perf stat ./matmul

```
-bash-4.1$ perf stat -e cache-misses ./matmul
Execution time: 0.146337 [s] // matmul output

 Performance counter stats for './matmul':

            90,633 cache-misses

       0.155192113 seconds time elapsed
```

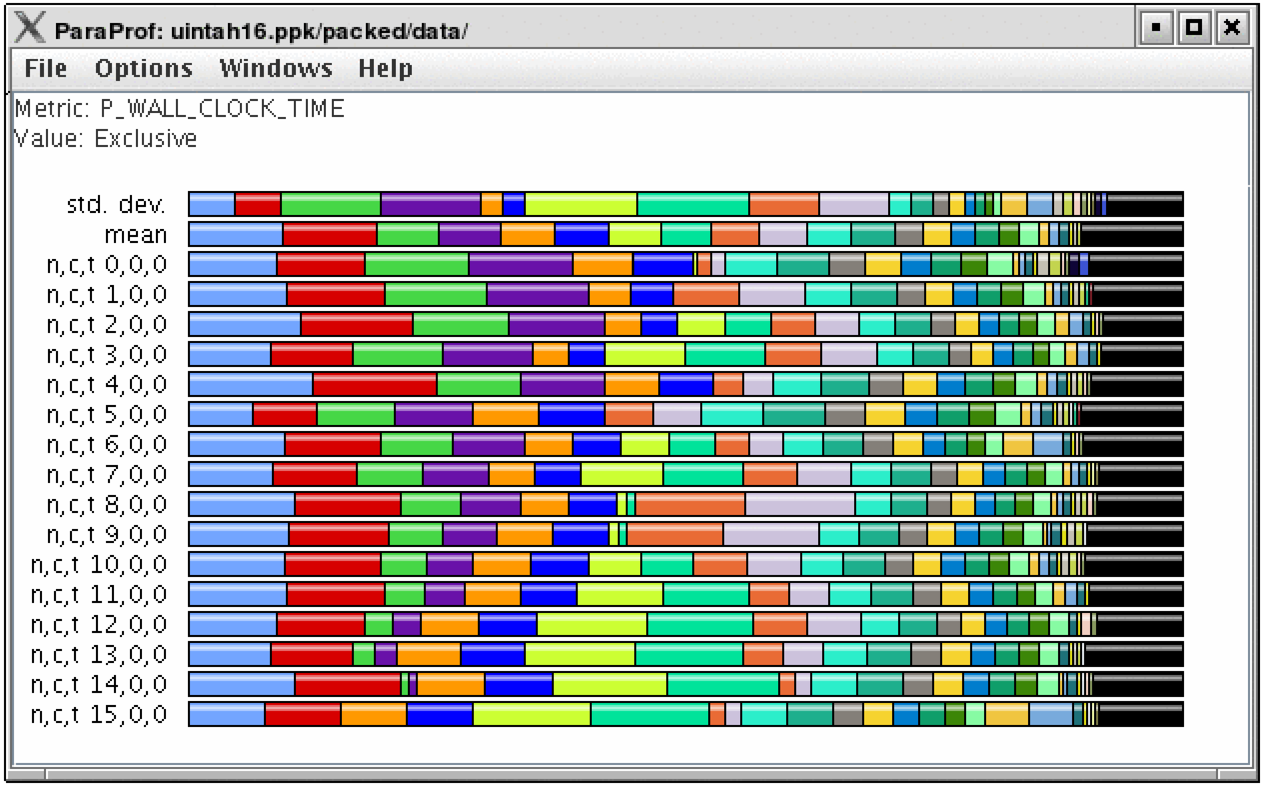
```
-bash-4.1$ perf stat ./matmul
Execution time: 0.164323 [s] // output of matmul program

 Performance counter stats for './matmul':

       6222.036262 task-clock                #   36.071 CPUs utilized
               188 context-switches          #    0.030 K/sec
                60 cpu-migrations            #    0.010 K/sec
             1,074 page-faults               #    0.173 K/sec
     7,459,088,185 cycles                    #    1.199 GHz
     4,647,863,937 stalled-cycles-frontend   #   62.31% frontend cycles idle
   <not supported> stalled-cycles-backend
     6,683,693,162 instructions              #    0.90  insns per cycle
                                             #    0.70  stalled cycles per insn
       672,748,725 branches                  #  108.124 M/sec
           292,465 branch-misses             #    0.04% of all branches

       0.172492173 seconds time elapsed
```

Tuning and Analysis Utilities (TAU)

- **Support**
  - code instrumentation
  - profiling (distribution of execution time across routines)
  - tracing (shows when and where an event occurred)
  - performance analyzing (textual and graphical display of results)

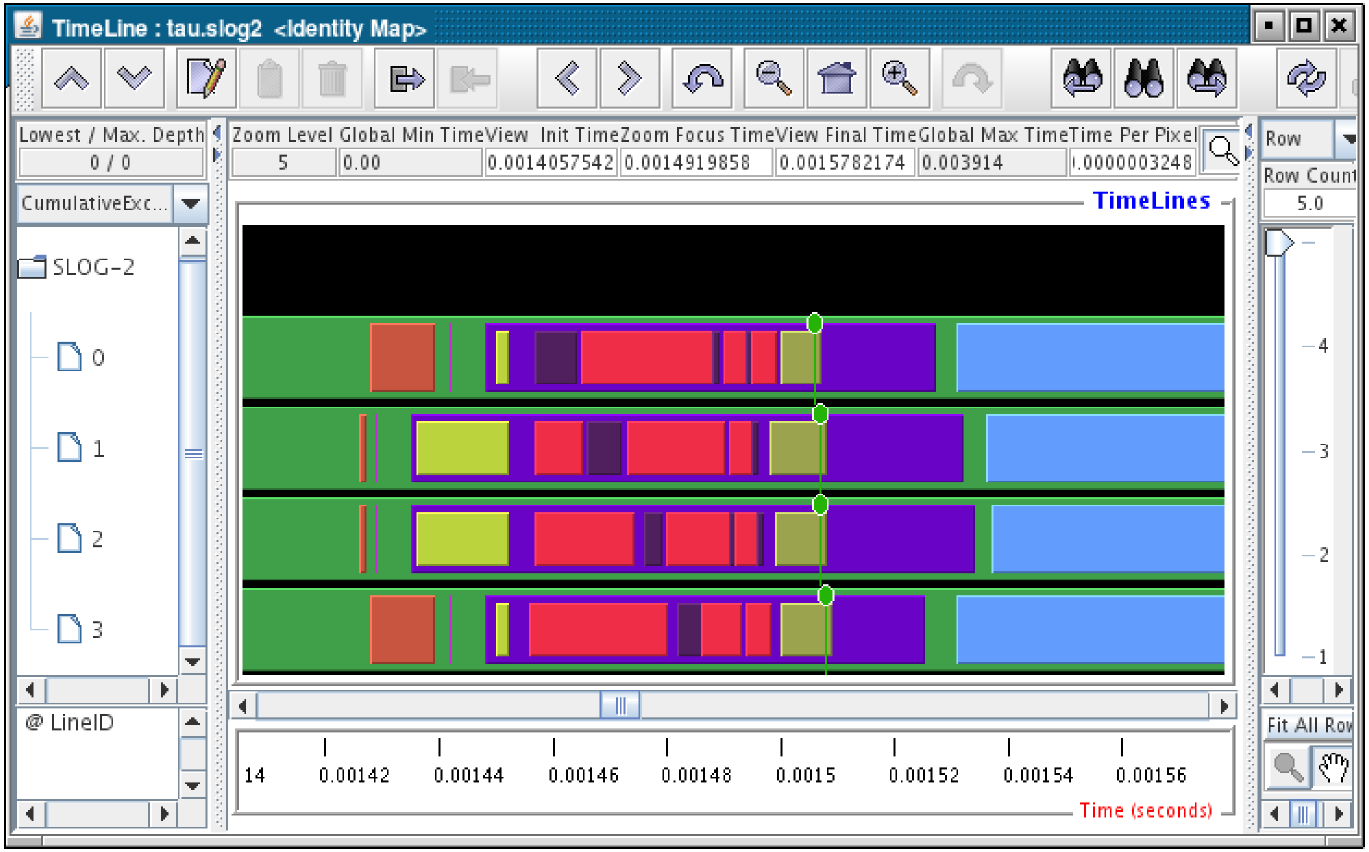
- **Documentation is available online**
  - http://www.cs.uoregon.edu/research/tau/docs.php

TAU: ParaProf



Visualization of the profile data with ParaProf.
Source: http://www.cs.uoregon.edu/research/tau/docs/newguide/bk01ch04s02.html

TAU: Jumpshot



Visualization of trace data (trace of events in time-line) with Jumpshot
Source: http://www.cs.uoregon.edu/research/tau/docs/newguide/bk01ch04s03.html