High Performance Parallel Computing (1.)

overview, basics

Imre Szeberényi, Éva Geist

BME IIT

<szebi@iit.bme.hu>

MŰEGYETEM 1782

Course information

- Course site
  - https://www.iit.bme.hu/targyak/BMEVIIIMA06
  - teams + edu.vik.bme.hu
  - 2+1+0v (2 lecture +1 practice + 0 lab + exam)
- Schedule
  - Lecture: Monday, 10:15-11:45, IL408
  - Practice: Thursday: 10:15-11:45, IB408 (biweekly, first on Sept. 14.)
- Assessment
  - Should not miss more than 30% of the classes
  - One final test
  - Homework – has to be completed by the exam
  - Written exam and homework presentation
  - 5 small knowledge test (optional for offering)

Course information #2

Grading:

IF best 3 from the 5 small tests are better than 80% (each) and the midterm test is better than 80%

- offered mark is: 2 * midterm test + 1 * HW

ELSE IF the midterm test is better than 40%

- mark is: 2 * written part of the exam + 1 * HW

ELSE

- mark is "not absolved,,

HW

A parallel implementation and performance study of a problem or algorithm you are interested in. Implementations can use OpenMP, Pthreads, MPI, Charm, OpenCL, CUDA, …

HW has to be demonstrated in person.

Course information #3

Usefull sources

Parhami, B., *Introduction to Parallel Processing: Algorithms and Architectures*, Plenum Press, 1999

Rauber, T. and G. Runger, *Parallel Programming for Multicore and Cluster Systems*, 2nd ed., Springer, 2013

Lawrence Livermore National Laboratory: Introduction to Parallel Computing Tutorial
https://hpc.llnl.gov/training/tutorials/introduction-parallel-computing-tutorial

Free on-line book (Creative Commons License)
Matloff, N., *Programming on Parallel Machines: GPU, Multicore, Clusters and More*, 341 pp., PDF file
http://heather.cs.ucdavis.edu/~matloff/158/PLN/ParProcBook.pdf

Useful free on-line course, sponsored by NVIDIA
“Introduction to Parallel Programming,” CPU/GPU-CUDA
https://developer.nvidia.com/udacity-cs344-intro-parallel-programming

Short History

- In early 1950s Daniel Slotnick proposed a parallel machine, but John von Neumann dismissed because it reuires „too many tubes”
- 1967: ILLIAC IV (256 proc, 200MFlop)
- Thinking Machines CM-1, CM-2 (1980)
- Cray-1

What is Parallel Processing?

Parallelism = Concurrency

Doing more than one thing at a time

Has been around for decades, since early computers

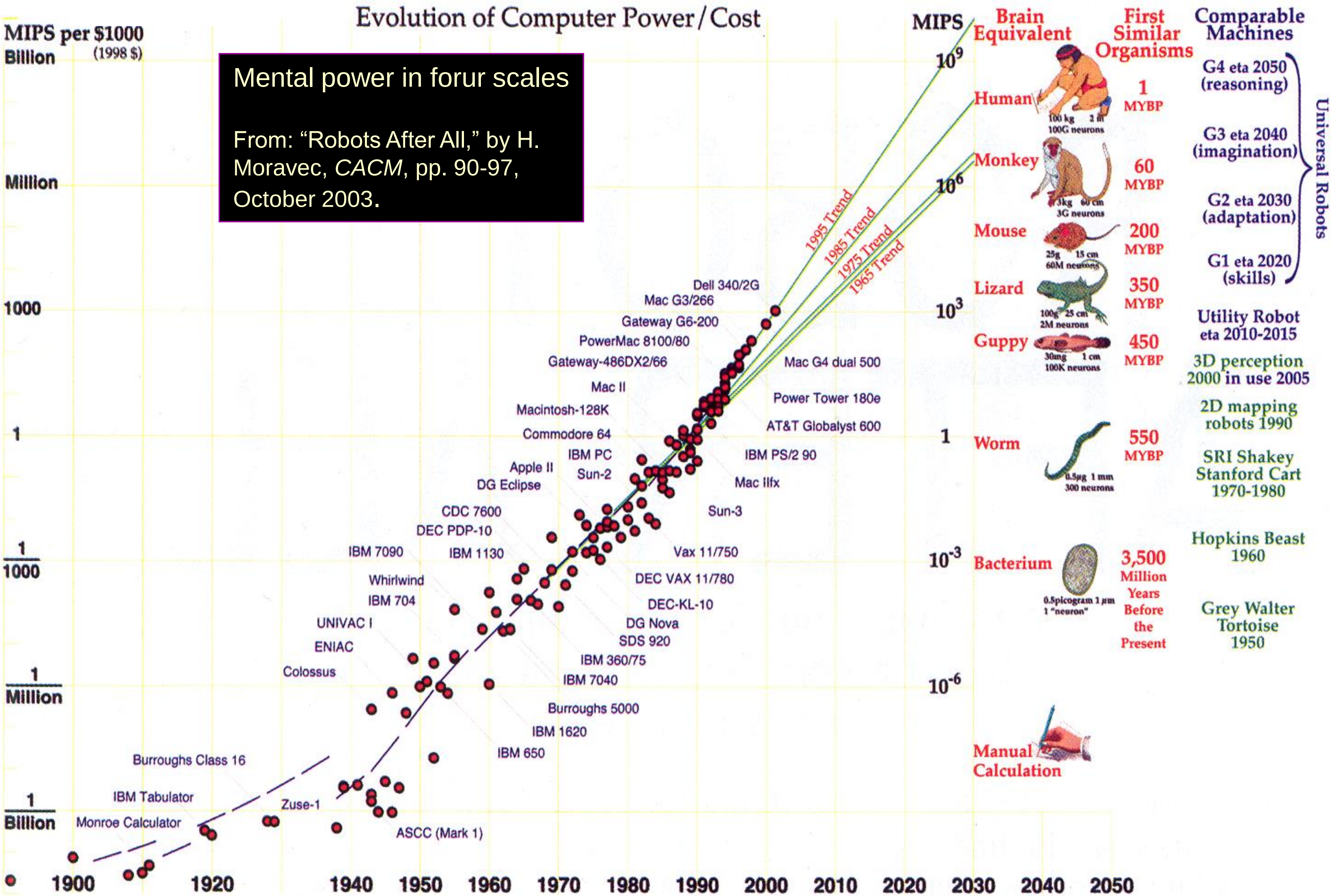
I/O channels, DMA, device controllers, multiple ALUs
Multiple CPUs, Multiple Cores, …

Why do we need big performance?

- Higher speed (solve problems faster)
- Higher throughput (solve more problems)
- Higher computational power (solve larger problems)
- Examples:
  - Modelling
    - weather, environment, drugs, dieses
    - forecasting (earthquake, volcanoes, global warming)
    - Mechanics, chemistry
  - Simulations
    - design, production
    - bioinformatics, drug discovery
    - particle physics

Mental power in forur scales

From: “Robots After All,” by H. Moravec, *CACM*, pp. 90-97, October 2003.



The Speed-of-Light Argument

The speed of light is about 30 cm/ns.

Signals travel at 40-70% speed of light (say, 15 cm/ns).

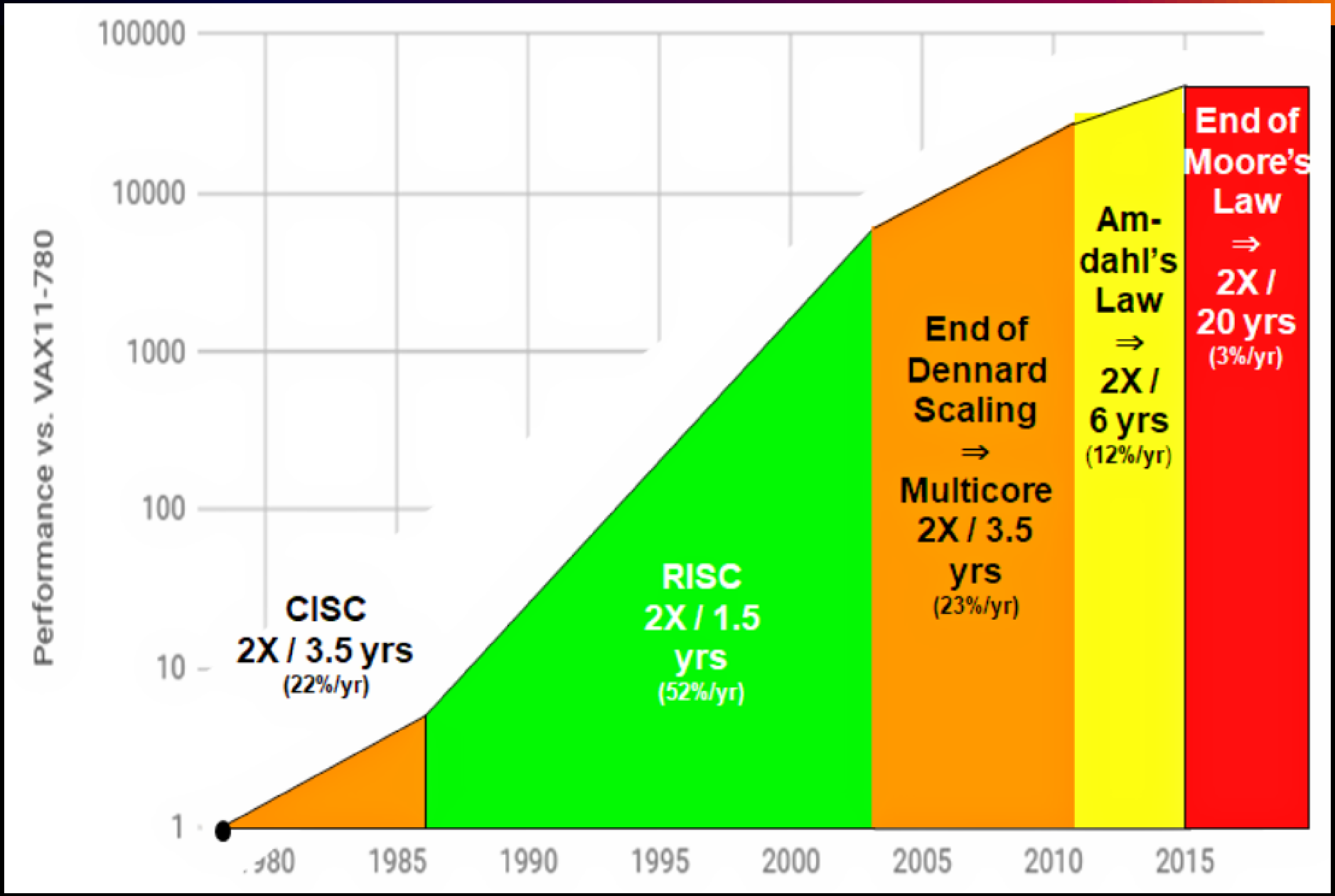
If signals must travel 1.5 cm during the execution of an instruction, that instruction will take at least 0.1 ns; thus, performance will be limited to 10 GIPS.

This limitation is eased by continued miniaturization, architectural methods such as cache memory, etc.; however, a fundamental limit does exist.

Performance development

- Increase the clock frequency
  - Physical limits (3.7GHz, 9GHz+liquid Nitrogen cooling)
  - energy requirement (cubic dependency)
- Parallelization
  - overhead
  - hard to debug

Processor Performance



Source: Hennessy and Patterson, *Computer Architecture: A Quantitative Approach, 6/e. 2018*

Reading: https://www.eejournal.com/article/fifty-or-sixty-years-of-processor-developmentfor-this/

TOP 500 2013 november

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
5	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945
6	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115,984	6,271.0	7,788.9	2,325
7	Texas Advanced Computing Center/Univ. of Texas United States	**Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462,462	5,168.1	8,520.1	4,510
8	Forschungszentrum Juelich (FZJ) Germany	**JUQUEEN** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	458,752	5,008.9	5,872.0	2,301
9	DOE/NNSA/LLNL United States	**Vulcan** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	393,216	4,293.3	5,033.2	1,972
10	Leibniz Rechenzentrum	**SuperMUC** - iDataPlex DX360M4, Xeon E5-2680 8C	147,456	2,897.0	3,185.1	3,423

Only the 6th is European

TOP 500 2017 june

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/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 E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT 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, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. 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, Cray Gemini interconnect, 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

Now the 3rd is European

TOP 500 2018 june

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,282,544	122,300.0	187,659.3	8,806
2	**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
3	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/NNSA/LLNL United States	1,572,480	71,610.0	119,193.6	
4	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
5	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2550 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR , Fujitsu National Institute of Advanced Industrial Science and Technology (AIST) Japan	391,680	19,880.0	32,576.6	1,649
6	**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

The 6th again.

TOP 500 2020 june

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,299,072	415,530.0	513,854.7	28,335
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	**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
5	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
6	**HPC5** - PowerEdge C4140, Xeon Gold 6252 24C 2.1GHz, NVIDIA Tesla V100, Mellanox HDR Infiniband, Dell EMC Eni S.p.A. Italy	669,760	35,450.0	51,720.8	2,252

The 6th again.

TOP 500 2021 june

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	**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
5	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	706,304	64,590.0	89,794.5	2,528
6	**Selene** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63,460.0	79,215.0	2,646

Now only the 8th is European

TOP 500 2022 june

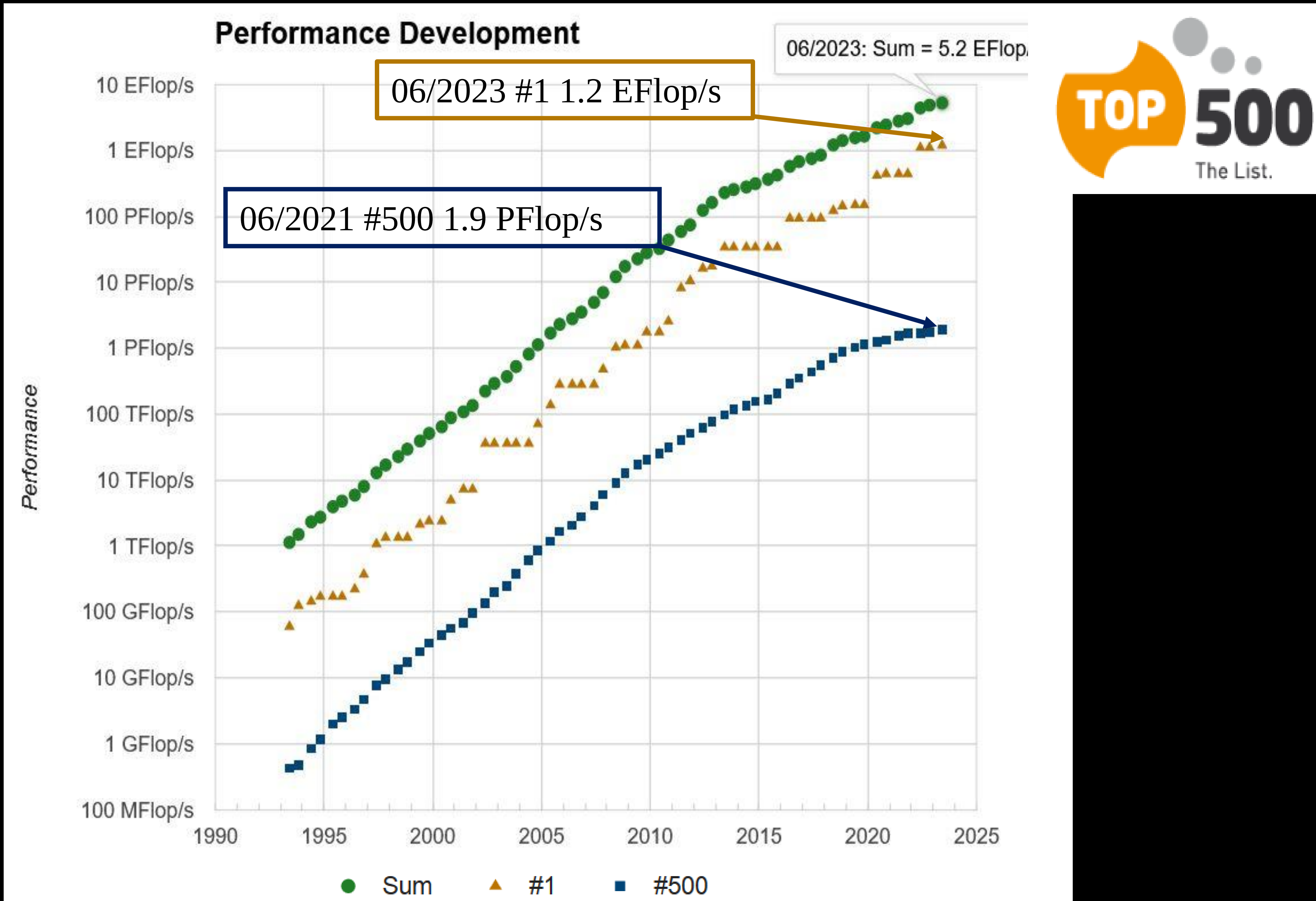
Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,730,112	1,102.00	1,685.65	21,100
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	1,110,144	151.90	214.35	2,942
4	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096
5	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94.64	125.71	7,438
6	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93.01	125.44	15,371

Now the 3th is European

TOP 500 2023 june

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,824,768	238.70	304.47	7,404
5	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096
6	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94.64	125.71	7,438

Now the 3th and the 4th is European


Performance Development
06/2023: Sum = 5.2 EFlop/
06/2023 #1 1.2 EFlop/s
06/2021 #500 1.9 PFlop/s
10 EFlop/s
1 EFlop/s
100 PFlop/s
10 PFlop/s
1 PFlop/s
100 TFlop/s
10 TFlop/s
1 TFlop/s
100 GFlop/s
10 GFlop/s
1 GFlop/s
100 MFlop/s
Performance
1990
1995
2000
2005
2010
2015
2020
2025
Sum
#1
#500
TOP 500
The List.

Behind the development

- The density of element
  - 1971: 10μm, 1984:1μm, 2001: 130nm, 2020: 5nm, 2022: 3nm, 2024: 2nm
  - https://en.wikipedia.org/wiki/2_nm_process
  - https://youtu.be/YIkMaQJSyP8
- Development of network technology
- Development of storage technology
- Development of memory
- Many cores technology

Overview of parallel programming

Why is parallel processing is important?

- For greater performance? Only for performance?
  - Parallelism in this case is just technology
  - not important for the application designer
  - Must be covered (such as hardware registers, cache, ...)
- A possible tool for modelling reality
- Simplify design work

Problems with parallel processing

- It is often harder to manage than serial processing
- fragile (non deterministic behaviour)
- deadlock problems can occur
- resource allocation issues can occur
- Error detection is not easy

These are real experiences, but not a necessity!

Parallel machine models

- Several models have emerged.
- The simplest is Flynn's model (taxonomy), which considers the parallel machine as the extension of the Neumann model.
- Another commonly used model is the idealized parallel computer model

Flynn's taxonomy

DATA

INSTRUCTIONS	**Single**	**Multiple**
Single	**S**ingle **I**nstruction **S**ingle **D**ata **SISD** (serial machines)	**S**ingle **I**nstruction **M**ultiple **D**ata **SIMD** (vector processors)
Multiple	**M**ultiple **I**nstruction **S**ingle **D**ata **MISD** (pipelines)	**M**ultiple **I**nstruction **M**ultiple **D**ata **MIMD** (multiprocesszors)

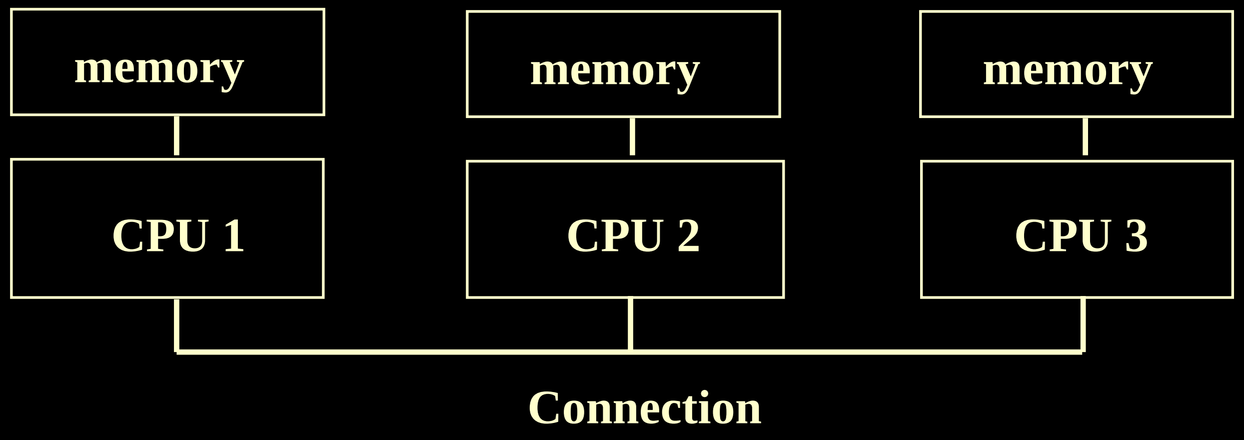
Johnson's expansion
see next slide

Johnson's expansion of MIMD

Comunication

Memory	**Shared variables**	**Message passing**
Global	**GMSV** Shared-memory multicomputers	**GMMP** **rarely used**
Distributed	**DMSV** Distributed shared memory multicomputers	**DMMP** Distributed memory multicomputers

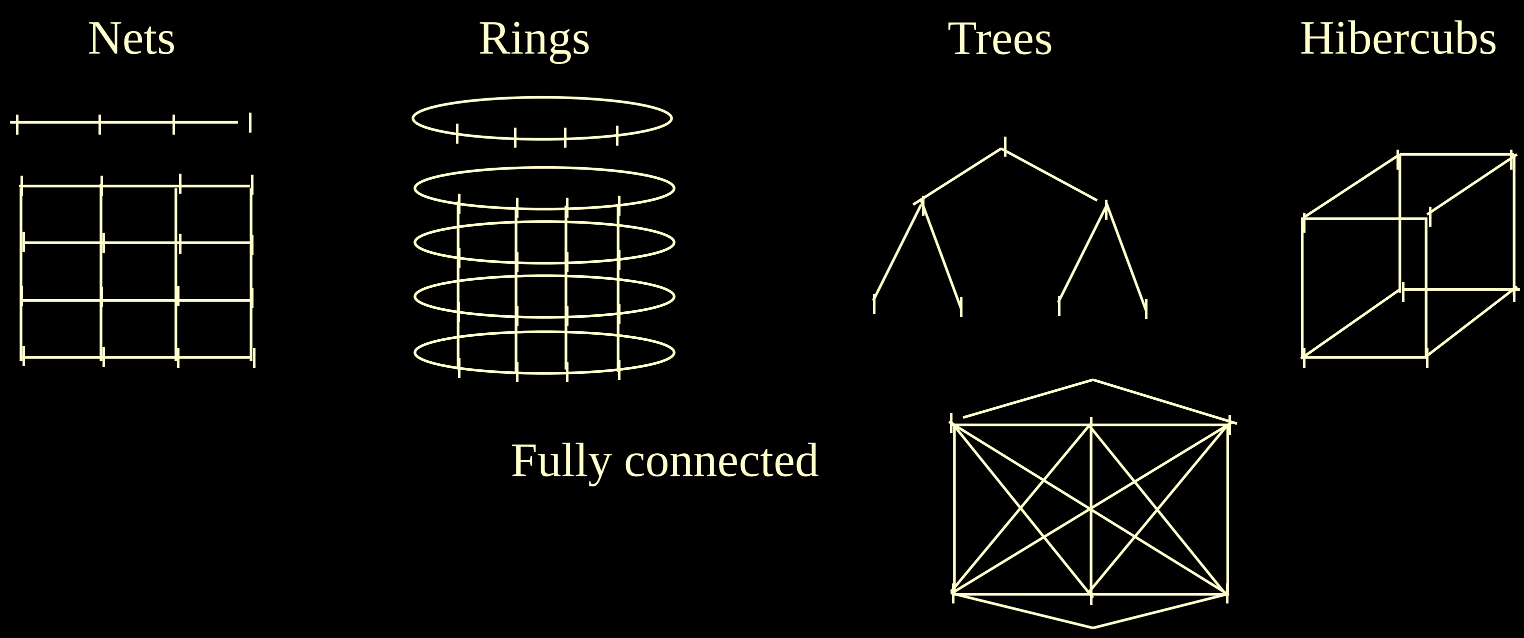
Idealized parallel computer



- Several processors are working on the same problem.
- Each processor has its own memory and address space.
- They coordinate with messages and can handover data.
- Local memory is faster to access.
- Transmission speed is independent of channel traffic.

Features of architectures

- Distribution of processors
- Homogeneous or heterogeneous
- Interconnect bandwidth and latency
- Topology



Programming model

- Shared memory
- Distributed shared memory
- Message passing

In fact, none of the models is closely tied to the actual physical architecture

Features of shared memory principle

- Benefits of using shared memory
  - Unified access
  - Simplified programming
  - Depending on the characteristics of hw, good speed-up ratings are available
- Disadvantages of using shared memory
  - Memory access may be a bottleneck
  - Not well scalable
  - Solving cache problems requires separate hardware
  - Debugging is difficult

Features of the distr. mem. principle

- **Benefits of using distributed memory**
  - Scalable
  - cost-effective
  - Increasing redundancy can increase reliability
  - It also works with special tools, axelators
- **Disadvantages of using distributed memory**
  - Demands a higher communication
  - Not all algorithms can be parallelised this way
  - Existing serial programs and applications using shared memory must be redrawn
  - Good speed ups are difficult to achieve
  - Debugging problems

Classes of parallel machines

- Machines with Vector processors, array proc.
  - optimized operations for vector data
  - one operating system instance
- Symmetric Multiprocessor (SMP)
  - many of same processor have common memory
  - one operating system instance
  - NUMA, ccNUMA
- Massively Parallel (MPP)
  - many processors with a fast internal network
  - distributed memory
  - there are many instances of operating system
- Cluster
  - many machines connected with a fast (commerce) network
  - distributed memory
  - in many instances it may be a heterogeneous operating system

Performance Measurement

Speed Up: $S_n = T_s / T_n$

where:
- S_n speed up gained with N processor
- T_s run time in Serial Execution
- T_n run time in case of N processor

Efficiency: $E_n = S_n / N$

where:
- E_n efficiency gained with N processor
- S_n speed up gained with N processor
- N number of processors

Redundancy (redundancy): $r = C_p / C_s$

where:
- r is the redundancy of the parallel program
- C_p The number of operations in the parallel program
- C_s The number of operations in the serial program

Speed Up limit

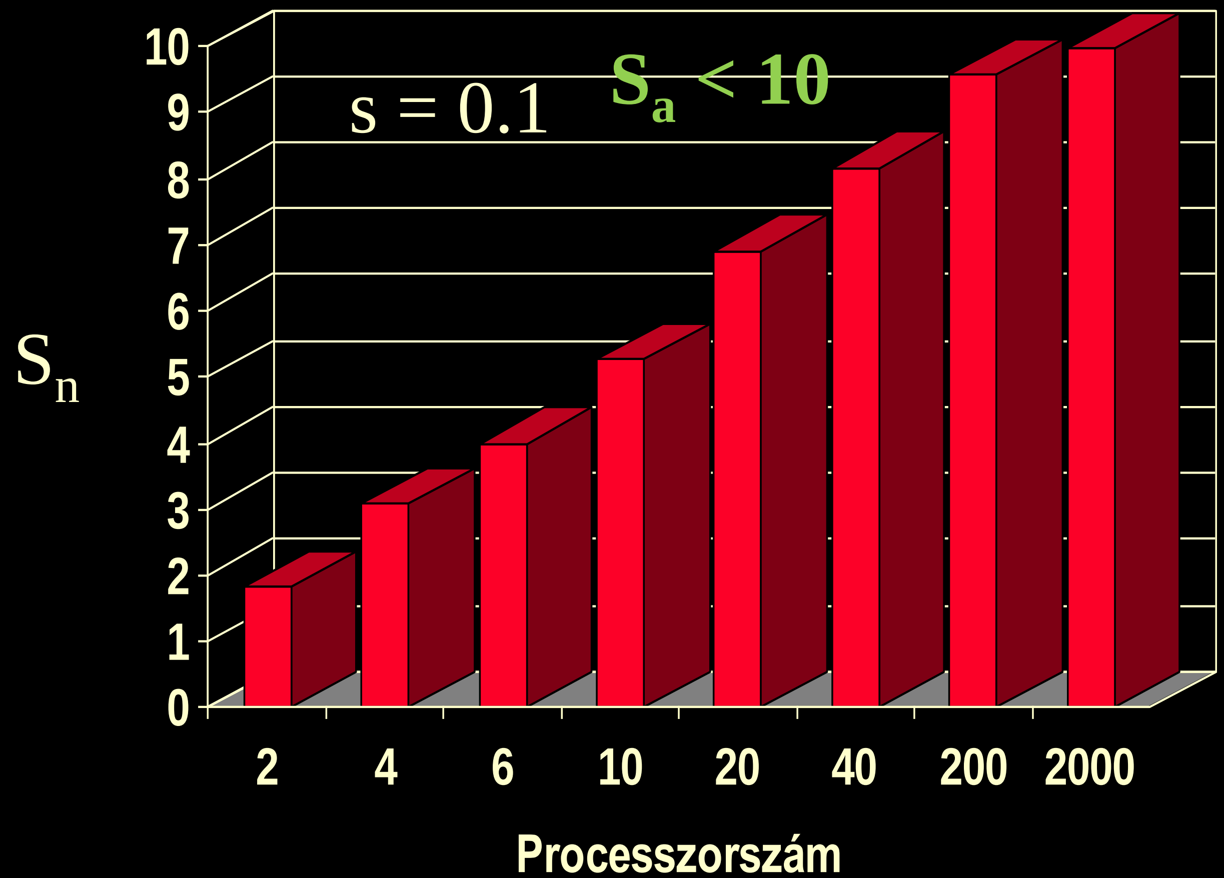
- **Amdahl's upper limit:** $\mathbf{S_a = 1 / (s + (1-s) / N)}$

where: S_a the upper limit of the speed up which can be gained by N processors
s part of the task that can NOT be parallelized
N Number of processors

Leaving (1-s) / N: $\mathbf{S_a < 1 / s}$
gives the above context which provides an upper limit.

For example, 10% non-parallelizable part gives $1 / 0.1 = 10$ for the upper limit of speed up.

It can not be accelerated unlimited



Programming languages

- Linda - a shared memory model, Tuple Space
  - 1986 – (C, later: Java, Python, Ruby, Lua ...)
  - out - copies a tuple to the shared area
  - in, inp - inports a tuple (will be removed)
  - rd, rdp – reads a tuple (p – probe, non blocking)
  - eval - executes a function as a process

In fact, there is a language supplement for a simple model but there are implementation difficulties, especially in message passing architectures.

Linda example #1 (hello)

```
#define NPROC 8
int hello(int id) {
  int j;
  in("count", ?j);
  printf("Hello World from proc%d: %d\n", id, j);
  return j+1;
}
int real_main() {
  int i;
  out("count", 0);
  for (i = 0; i < NPROC; ++i)
    eval("count", hello(i));
  in("count", NPROC);
  printf("All processors done\n");
  return 0;
}
```

Linda example #2 (queue)

```
int init() {
  out("head", 0);
  out("tail", 0);
}
```

```
void push(int elem) {
  int tail;
  in ("tail", ?tail);
  out("elem", tail, elem);
  out("tail", tail+1);
}

int pop() {
  int head, elem;
  in ("head", ?head);
  in ("elem", head, ?elem);
  out("head", head+1);
  return elem;
}
```

Programming languages/2

- Express - distributed memory model using message passing technique. Portable, supports logical connection of nodes.

~160 Routine callable from C and Fortran:

  - start, shutdown
  - structure of logical communication topology
  - communication between programs,
  - Synchronization
  - file operations
  - graphic operations
  - performance analysis, debugging

Programming languages/3

- PVM – (Parallel Virtual Machine) Distributed Memory Model
  - 1989 – 1993

~70 Routines callable from C and Fortran:
  - start, shutdown
  - communication between programs, synchronization

What is PVM used for?

- The Poor Man's supercomputer

  free CPU capacities can be collected from workstations and PCs

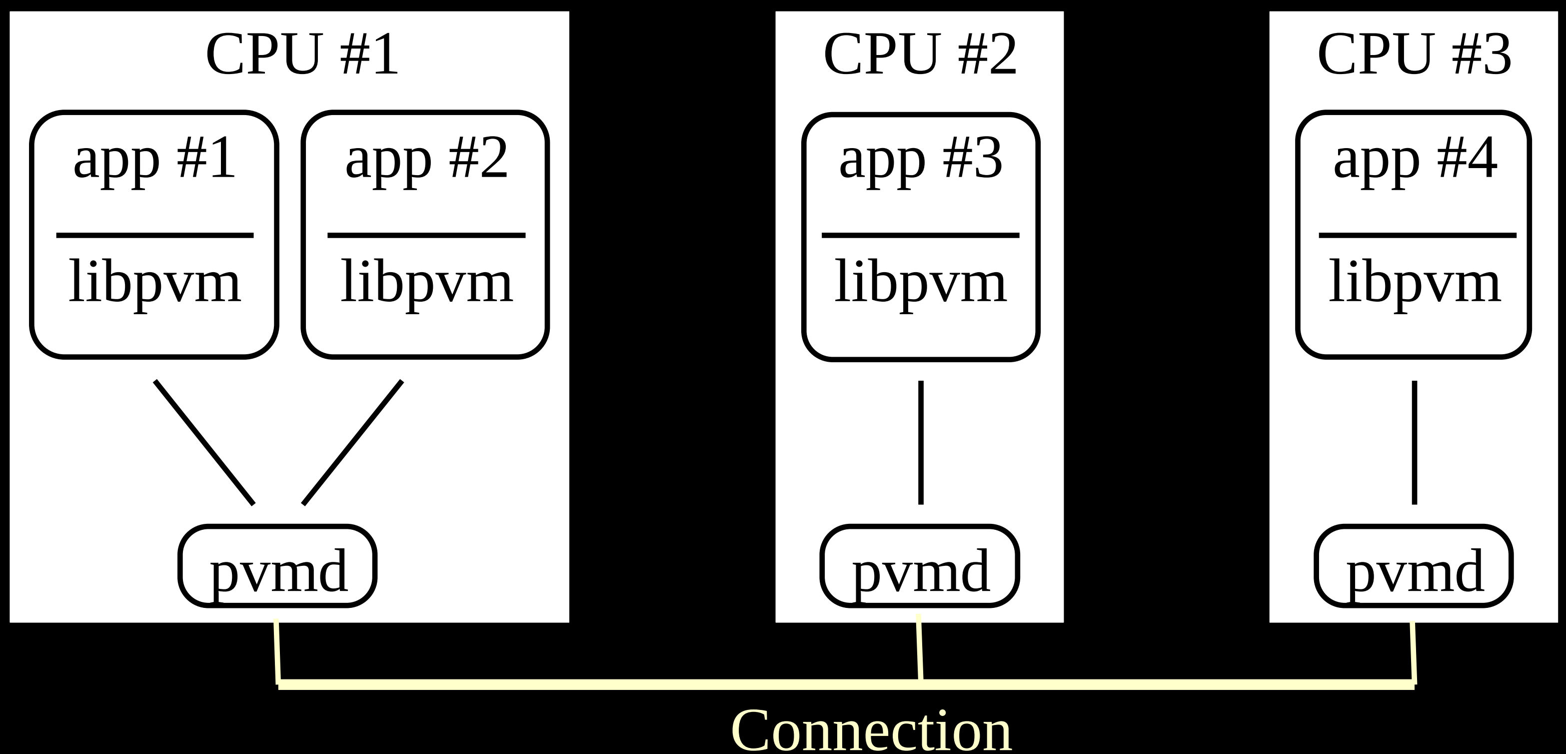
- Combining multiple supercomputers can create incredible computing capacity
- Educational tool

  It is an effective tool for teaching parallel programming

- Research tool

  scalable and cost-effective

The basic concept of PVM



app # → Part of the application

libpvm → Language interface

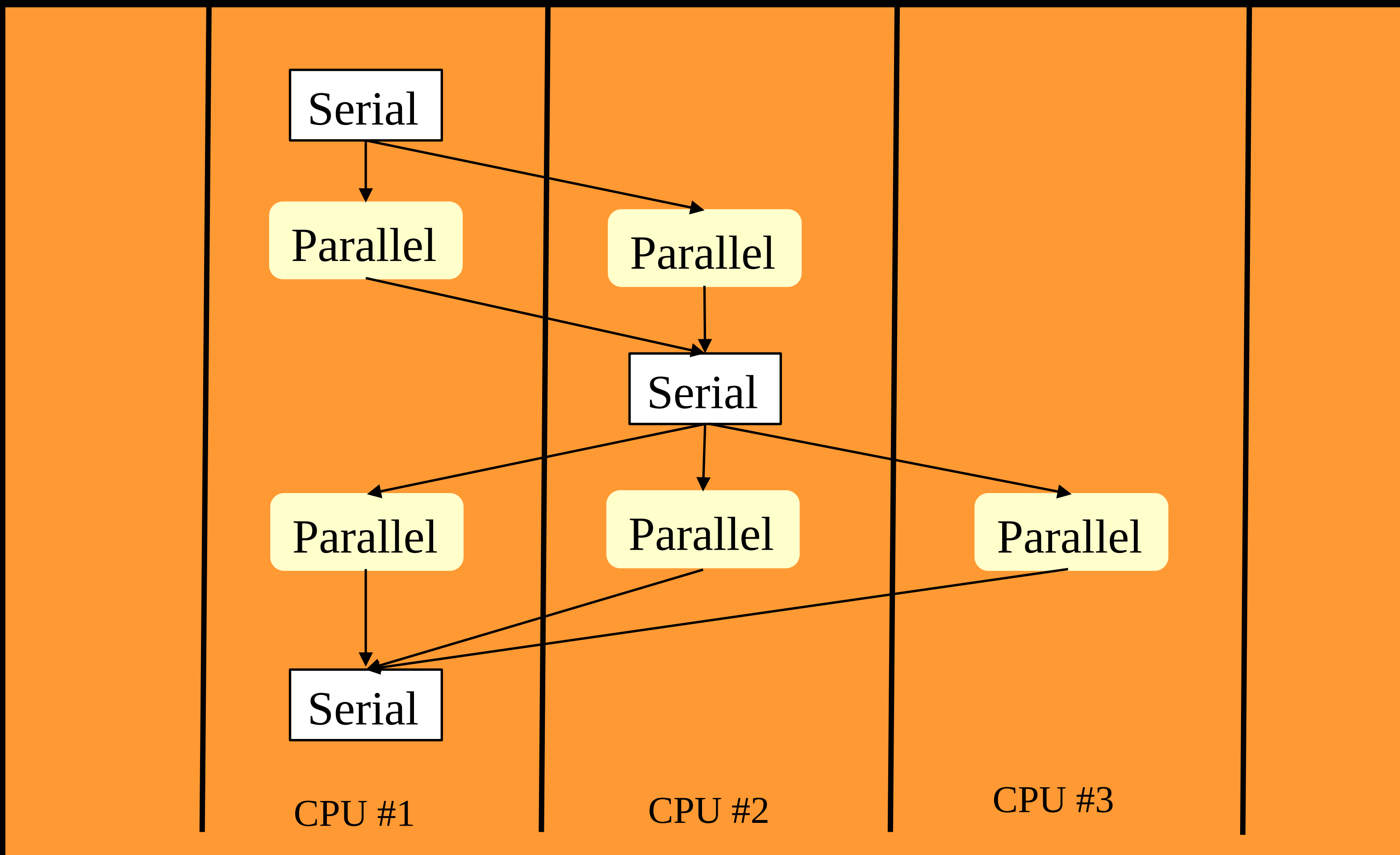
pvmd → PVM daemon

libpvm

The features provided by libpvm can be divided into four groups:

- Administrative functions
  virtual machine start, stop, add a new node, status query,
- Process Manager Features
  Launching and stopping processes
- Data Transmission Features
  Composing, sending, receiving, packing messages
- Synchronization features
  By sending a message or using a barrier

Structure of a parallel program



Parallelization strategies

Embarrassingly parallel

– We run the serial version of the program in parallel with different data.

– It is only a satisfactory method if the serial version has a tolerable running time.

Parallelization of cycles

– It can be applied if each iteration is independent of each other

Divide & conquer (master / slave)

– A supervisor task runs on one node

– It can be used if the tasks of the supervisor program are simpler than those of other tasks.

– If the tasks are independent of each other, they can be scaled well by varying the number of tasks.

Parallelization strategies/2

Consecutive

– Each node passes the partially processed data to the next node.
– It can be used if the serial part of the processing is considerably shorter than the parallel part.
– Usually every node runs the same code.
– Particularly suitable for ring topology.

Parallelization of regions

– Data dependency can be localized into regions.
– It can be used with serial execution time greater than parallel.
– Usually, it needs high volume of communication.
– Most complicated.

Integral calc. example

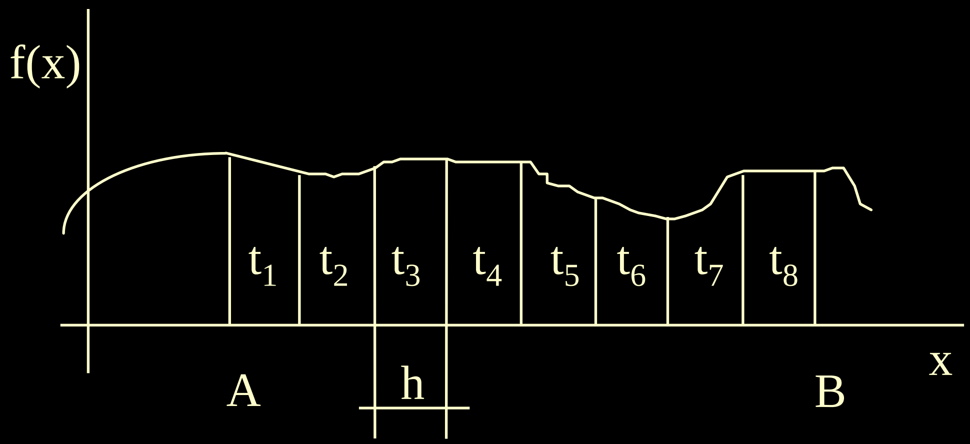
Calculate the

$$\int_A^B f(x)\,dx$$

integer value by simple numerical approximation (sum of area of rectangles)!

$$\int_A^B f(x)\,dx = h \cdot \sum_{i=1}^{N} f\left(A - \frac{h}{2} + i \cdot h\right) \qquad \text{ahol } h = \frac{B-A}{N}$$

In case N=8 eg:



Calculation of area of individual rectangles can be performed independently, in parallel.

Integral calc. example/2

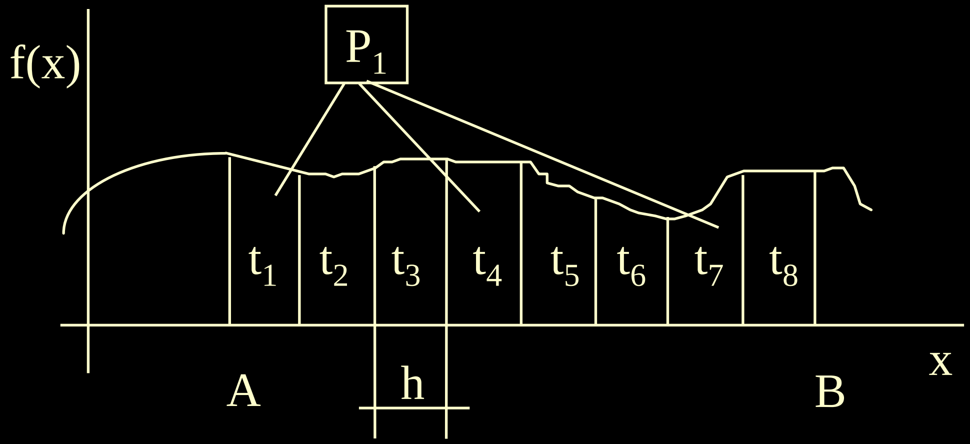
Calculate the

$$\int_A^B f(x)\,dx$$

integer value by simple numerical approximation (sum of area of rectangles)!

$$\int_A^B f(x)\,dx = h\cdot\sum_{i=1}^{N} f\left(A-\frac{h}{2}+i\cdot h\right) \qquad \text{ahol } h=\frac{B-A}{N}$$

In case N=8 eg:



Calculation of area of individual rectangles can be performed independently, in parallel. For example, each task only counts every Mth area and then sums up the results.

Integral calc. example/3

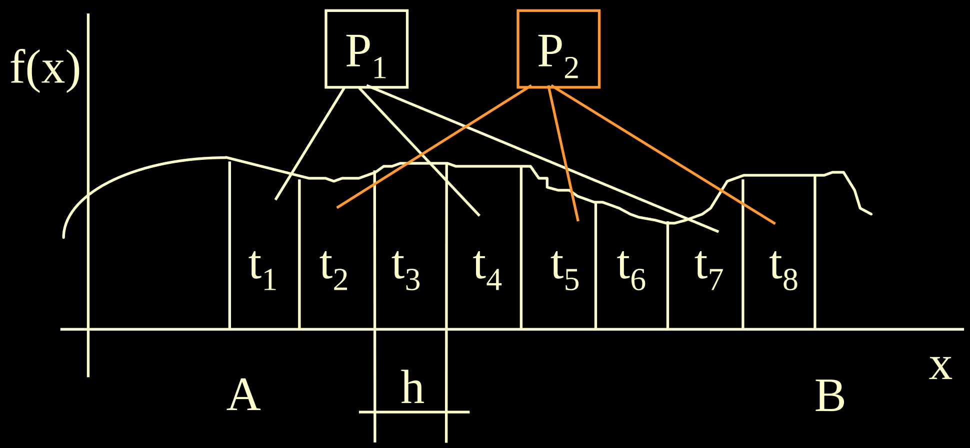
Calculate the

$$\int_A^B f(x)\,dx$$

integer value by simple numerical approximation (sum of area of rectangles)!

$$\int_A^B f(x)\,dx = h \cdot \sum_{i=1}^{N} f\left(A - \frac{h}{2} + i \cdot h\right) \qquad \text{ahol } h = \frac{B - A}{N}$$

In case N=8 eg:



Calculation of area of individual rectangles can be performed independently, in parallel. For example, each task only counts every M^{th} area and then sums up the results.

Integral calc. example/4

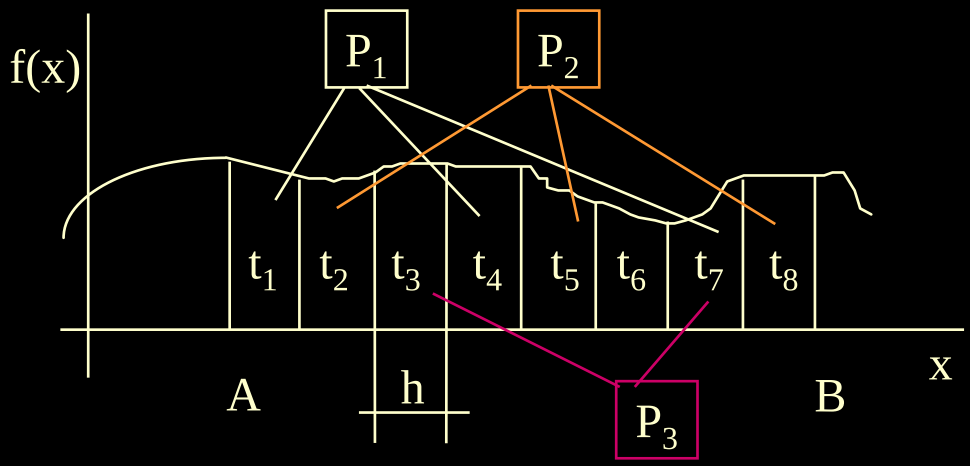
Calculate the

$$\int_A^B f(x)\,dx$$

integer value by simple numerical approximation (sum of area of rectangles)!

$$\int_A^B f(x)\,dx = h \cdot \sum_{i=1}^{N} f\left(A - \frac{h}{2} + i \cdot h\right) \qquad \text{ahol } h = \frac{B-A}{N}$$

In case N=8 eg:

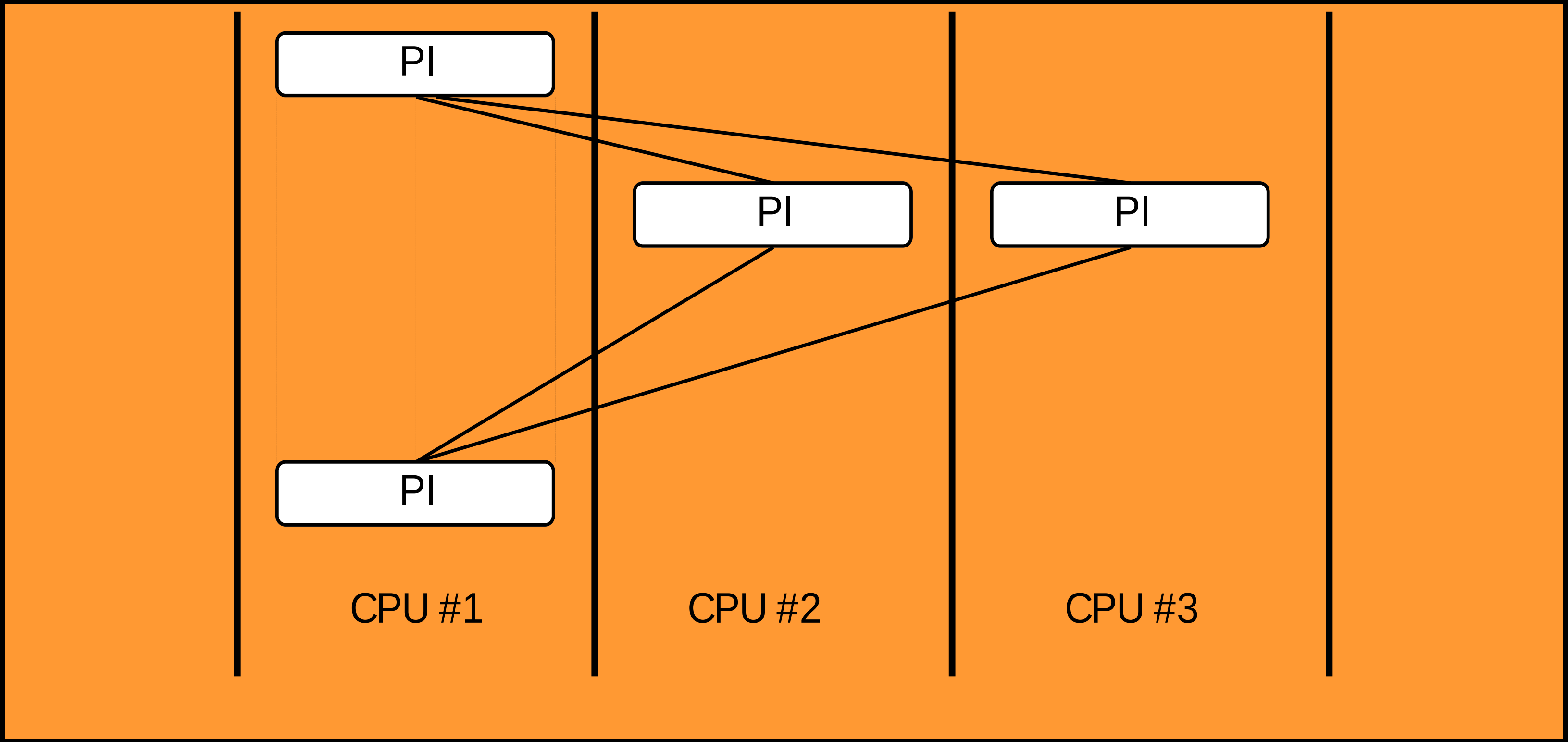


Calculation of area of individual rectangles can be performed independently, in parallel. For example, each task only counts every Mth area and then sums up the results.

PI example

Calculate the PI value of a $\int_0^1 \frac{4}{1+x^2}\,dx$ integral numerical integration!

Using the above method, we write SPMD program. The first instance of the program prompts the step and starts the other instances. Each instance only counts all M^{th} rectangle area, and the results are sent back to the first instance.



pi.c program

```
#include "pvm3.h"                    /* PVM 3 include file */
#define f(x) ((float)(4.0/(1.0+x*x))) /* function */
...
// startup function: each intance executes (SPMD)
//     The first one enters to the PVM and starts the other instaces
//     Returns: mynum, nprocs, and tids
void  startup(int *pmynum, int *pnprocs, int tids[]) {
    int  i, mynum, nprocs, info, mytid, numt, parent_tid;
    mytid = pvm_mytid();
    if (mytid < 0) {
      printf("Can't enter to PVM\n");
      exit(0);
    }
    parent_tid = pvm_parent();
```

pi.c program /2

```
if (parent_tid  ==  PvmNoParent) {
      mynum = 0;  tids[0] = mytid;
      printf ("How many instance (1-%d)?\n", MAXPROCS);
      scanf("%d", &nprocs);
      numt = pvm_spawn("pi", NULL, PvmTaskDefault, "", nprocs, &tids[1]);
      if (numt  !=  nprocs) { printf ("Error numt != nporcs);  exit(0); }
     *pnprocs = nprocs;                          /* instance numbers */
      info = pvm_initsend(PvmDataDefault);      /* broadcasts tid info to all */
      info = pvm_pkint(&nprocs, 1, 1);
      info = pvm_pkint(tids, nprocs+1, 1);
      info = pvm_mcast(&tids[1], nprocs, TAG_TIDS);
    } else {
      info = pvm_recv(parent_tid, TAG_TIDS);
      info = pvm_upkint(&nprocs, 1, 1);
      info = pvm_upkint(tids, nprocs+1, 1);
      for (i = 1; i  <=  nprocs; i++) if (mytid  ==  tids[i])  mynum=i;
    }
    *pmynum = mynum;
}
```

pi.c program /3

```
// solicit function: each intance executes (SPMD)
//  Returns the N (numner of rectangles). The first instance asks as a console input
//      The others reads from the first instance
void  solicit(int *pN, int *pnprocs, int mynum, int tids[]) {
    int info;
    if (mynum == 0) {
      printf("N: (0 = end)\n");
      if (scanf("%d", pN) != 1) *pN = 0;
      info = pvm_initsend(PvmDataDefault);
      info = pvm_pkint(pN,1,1);
      info = pvm_pkint(pnprocs,1,1);
      info = pvm_mcast(&tids[1], *pnprocs, TAG_N);
    } else {
      info = pvm_recv(tids[0], TAG_N);
      info = pvm_upkint(pN, 1, 1);
      info = pvm_upkint(pnprocs, 1, 1);
    }
}
```

pi.c program /4

```
int main() {  // SPMD
    float err, sum, w, x;
    int   i, N, M, info, mynum, nprocs, tids[MAXPROCS+1];
    startup(&mynum, &nprocs, tids);
    for (;;) {
       solicit (&N, &nprocs, mynum, tids);
       if (N <= 0) {
            printf("Instance %d exit\n", mynum);
           pvm_exit(); exit(0);
       }

    // Integral approximation
      M = nprocs+1; w = 1.0/(float)N;
      sum = 0.0;
      for (i = mynum+1; i <= N; i += M)
          sum = sum + f(((float)i-0.5)*w);
      sum = sum * w;
```

pi.c program /5

```
// main...
   if (mynum  ==  0) {       /* if it is the first instance */
     printf ("First instance sum = %7.5f\n", sum);
     for (i = 1; i <= nprocs; i++) {
       info = pvm_recv(-1, TAG_SUM);
       info = pvm_upkfloat(&x, 1, 1);
       printf ("First instance got x = %7.5f \n", x); fflush(stdout);
       sum = sum+x;
     }
     printf("sum = %12.8fn", sum); fflush(stdout);
   } else {              /* other instances  */
     info = pvm_initsend(PvmDataDefault);
     info = pvm_pkfloat(&sum, 1, 1);
     info = pvm_send(tids[0], TAG_SUM);
     printf("A %d. sent his data: %7.2f \n", mynum, sum);
    fflush(stdout);
   }
} // main
```

Message Passing Interface (MPI)

- It basically developed with other goals:
  - 1991 -> today
  - Standard, manufacturer-approved, special hw. environment supported by the development environment.
  - long delightful development
  - initially only static process management
  - The communication channels does not set up separately like in the PVM where the daemons must be started before the application.
  - The entire communication layer is linked to the application.