Parallel Programming

Instructor(s) :

Zhengxiong Hou (侯正雄)

Tianhai Zhao (赵天海)

houzhx,zhaoth@nwpu.edu.cn

Fall 2022

Basic information of this course

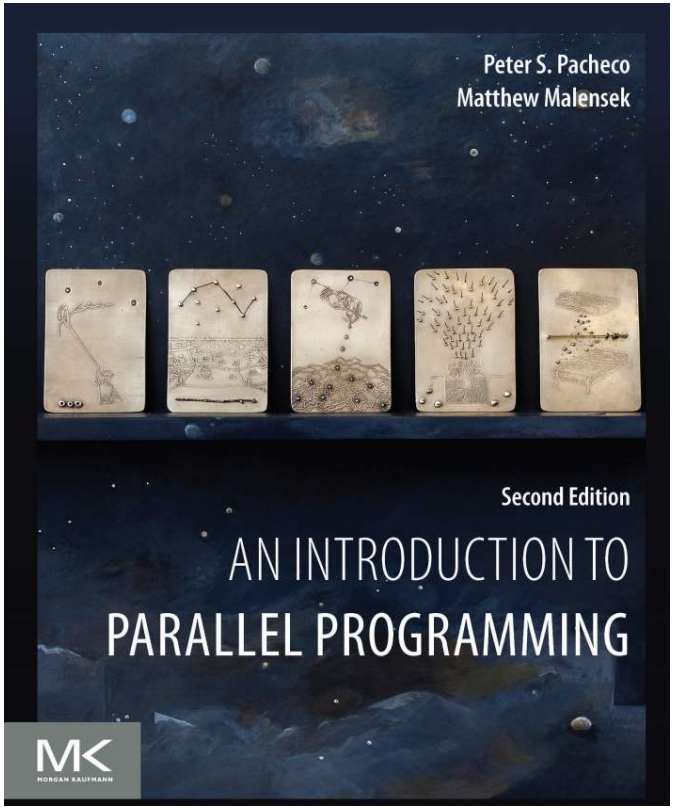
- **Credit: 3**
- **Class Hours: 48**
- **Weeks: 2-13**
- **Grading Policy and Class Rules**
  Daily Performance (10%)
  Homework Sets (30 %)
  Final Exam.-Open book (60 %)

TextBook

- **An Introduction to Parallel Programming**

  -Peter Pacheco, Matthew Malensek

  (University of San Francisco, 2021)



SOURCEBOOK OF PARALLEL COMPUTING

JACK DONGARRA
University of Tennessee

IAN FOSTER
Argonne National Laboratory

GEOFFREY FOX
Indiana University

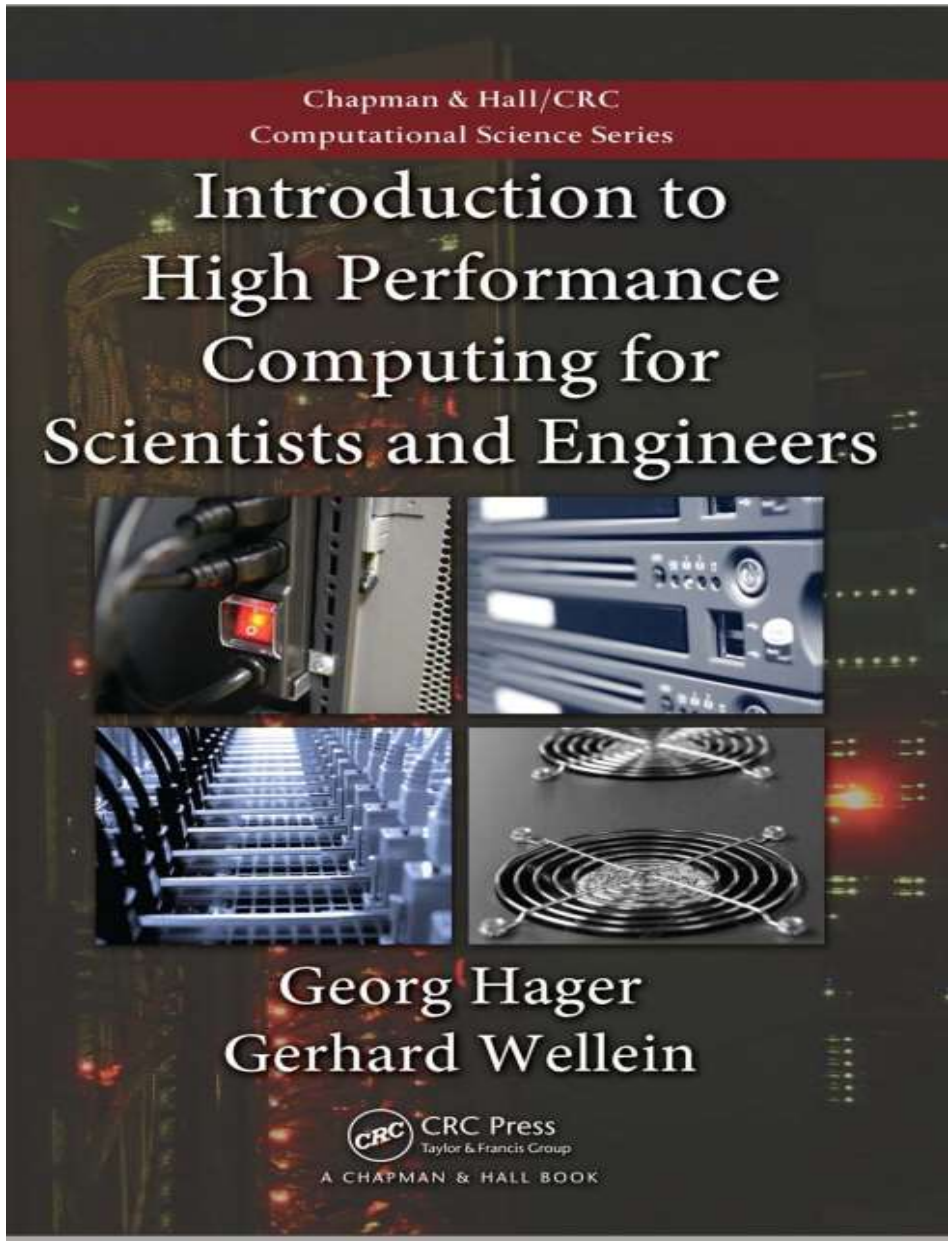
WILLIAM GROPP
Argonne National Laboratory

KEN KENNEDY
Rice University

LINDA TORCZON
Rice University

ANDY WHITE
Los Alamos National Laboratory

MK
MORGAN KAUFMANN PUBLISHERS
AN IMPRINT OF ELSEVIER SCIENCE
AMSTERDAM BOSTON LONDON NEW YORK
OXFORD PARIS SAN DIEGO SAN FRANCISCO
SINGAPORE SYDNEY TOKYO



并行计算系列丛书

并　行　计　算

——结构·算法·编程

（修订版）

陈国良　编著

高等教育出版社

Online-resources

-https://hpc.llnl.gov/training/tutorials#training_materials

-easyhpc.net

Contents

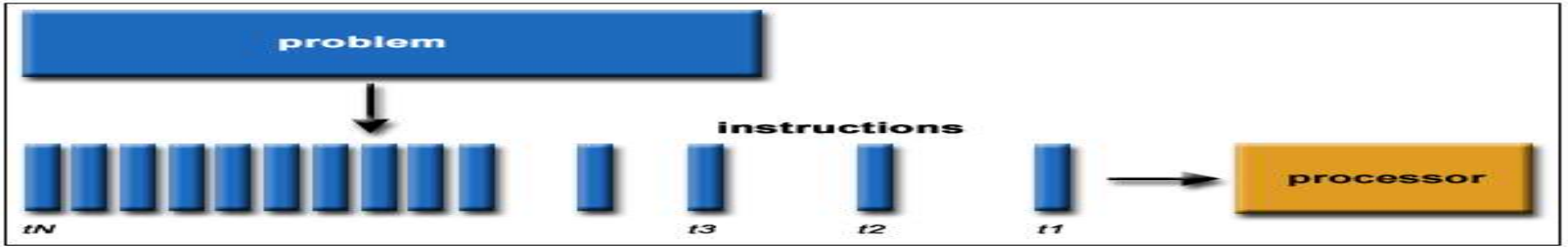
- Introduction of parallel computing (2 h.)
- Parallel computer architectures (4 h.)
- Parallel computation model (2 h.)
- Parallel algorithms (4 h.)
- Parallel programming models (2 h.)
- Parallel programming-MPI (8 h.)
- Performance Analysis and Tuning (2 h.)
- Parallel programming-Pthreads & OpenMP (12 h.)
- Parallel programming-applications development (12 h.)

I An Introduction to Parallel computing (HPC, SuperComputing)

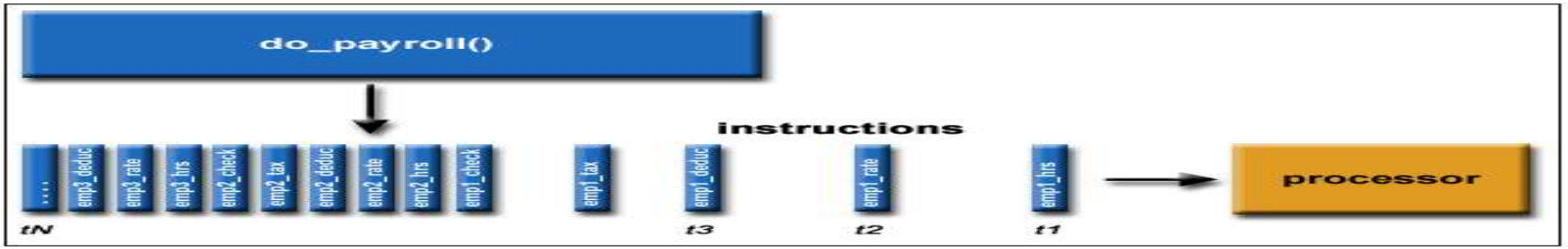
- What is parallel computing?
- Why parallel computing?
- Who is using parallel computing?
- Why we need to write parallel programs?
- Concepts and Terminology

Serial Computing

- Serial computing systems have been with us for more than seven decades since **John von Neumann** introduced digital computing in the 1950s
- Traditionally, software has been written for serial computation:
  - A problem is broken into a discrete series of instructions
  - Instructions are executed sequentially one after another
  - Executed on a single processor
  - Only one instruction may execute at any moment in time

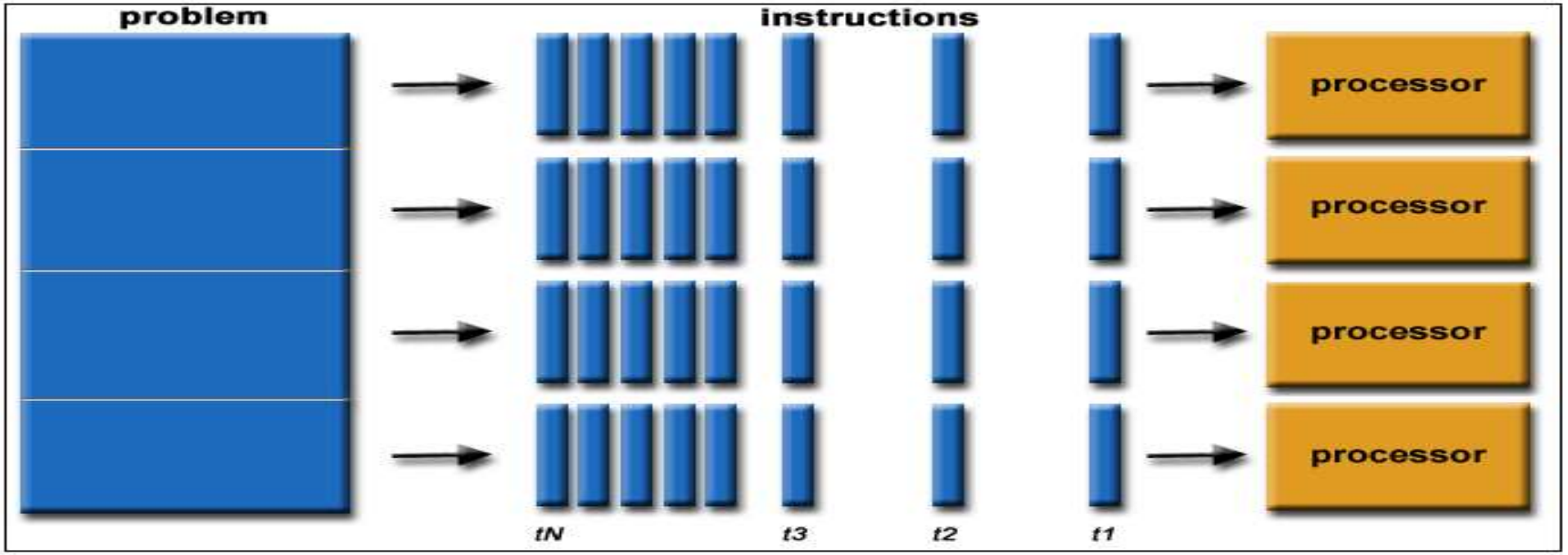


For example:

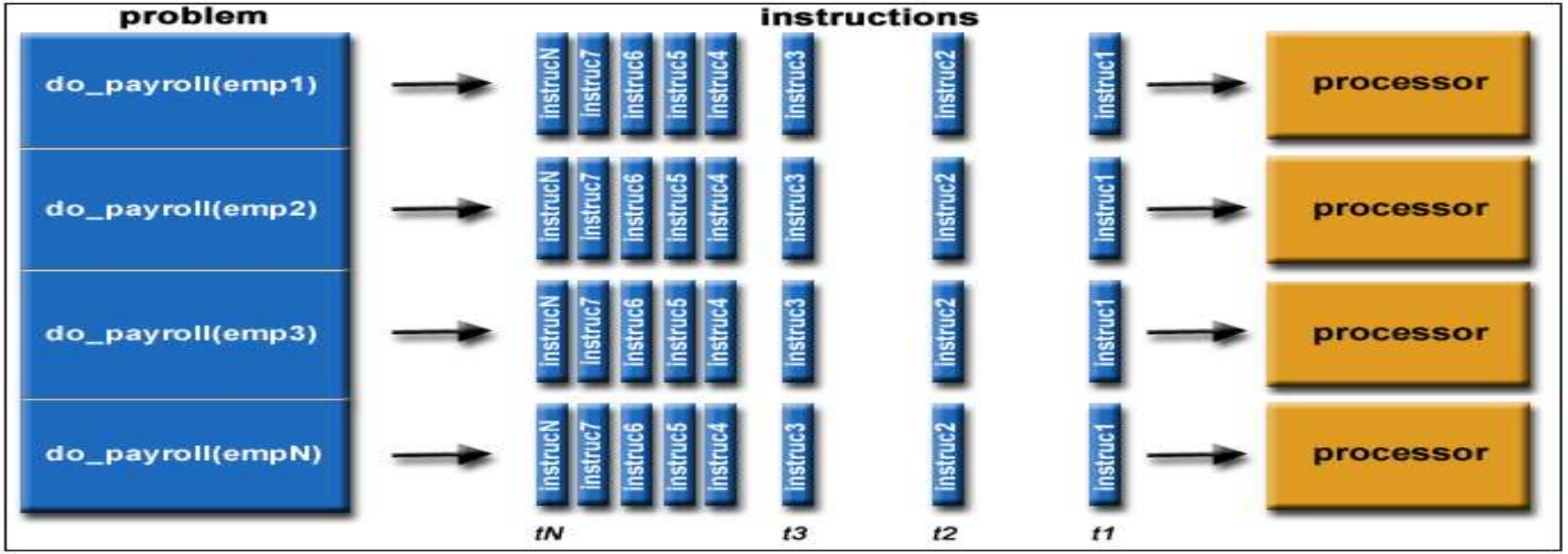


What is Parallel Computing?

- In the simplest sense, ***parallel computing*** is the simultaneous use of multiple compute resources to solve a computational problem:
  - A problem is broken into discrete parts that can be solved concurrently
  - Each part is further broken down to a series of instructions
  - Instructions from each part execute simultaneously on different processors
  - An overall control/coordination mechanism is employed

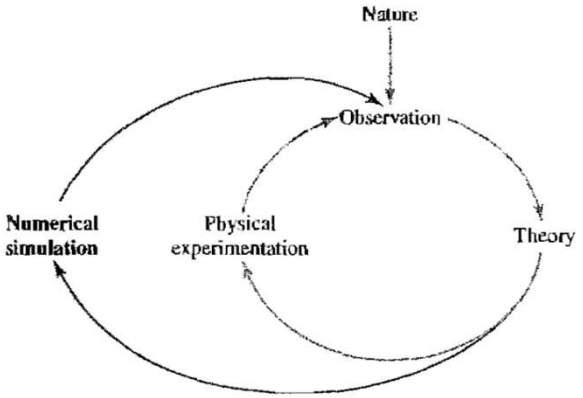

problem
instructions
processor
processor
processor
processor
tN
t3
t2
t1

For example:


problem
instructions
do_payroll(emp1)
do_payroll(emp2)
do_payroll(emp3)
do_payroll(empN)
instrucN
instruc7
instruc6
instruc5
instruc4
instruc3
instruc2
instruc1
processor
processor
processor
processor
tN
t3
t2
t1

- The computational problem should be able to:
  - Be broken apart into discrete pieces of work that can be solved simultaneously;
  - Execute multiple program instructions at any moment in time;
  - Be solved in less time with multiple compute resources than with a single compute resource.
- The compute resources are typically:
  - A single computer with multiple processors/cores
  - An arbitrary number of such computers connected by a network

Why parallel computing?



- **Requirements of Applications**

-Many important scientific, engineering, and commercial applications/problems are so complex that solving them via numerical simulation requires extraordinarily powerful computers

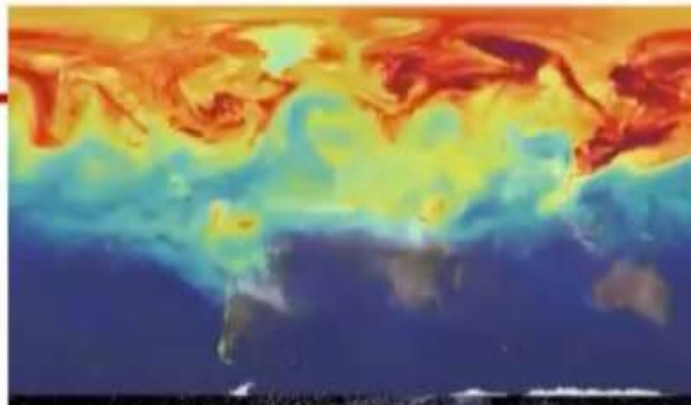
- **Development of (Parallel) Computers**

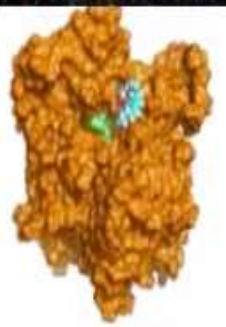
-From 1986 to 2002 the performance of microprocessors increased, on average, 50% per year (**Moore's Law)**

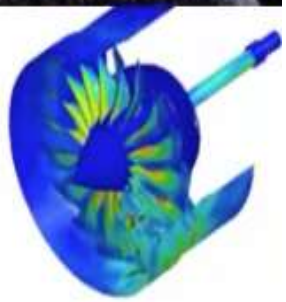
-By 2005, most of the major manufacturers of microprocessors had decided that the road to rapidly increasing performance lay in the direction of parallelism (**multi-core/many core**)

Simulation: The Third Pillar of Science

- Traditional scientific and engineering paradigm:
  1) Do theory or paper design.
  2) Perform experiments or build system.
- Limitations:
  - Too difficult -- build large wind tunnels.
  - Too expensive – to experiment with birds in a jet engine.
  - Too slow -- wait for climate or galactic evolution.
  - Too dangerous -- weapons, drug design.
- Computational science paradigm:
  3) Use high performance computer systems to simulate the phenomenon
     - » Base on known physical laws and numerical methods.



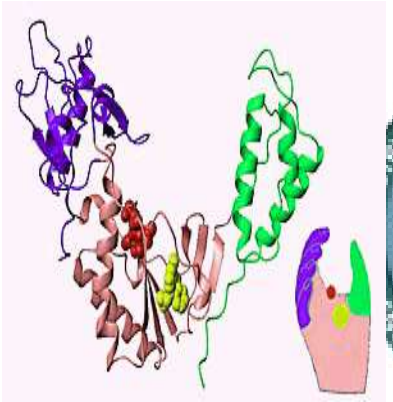




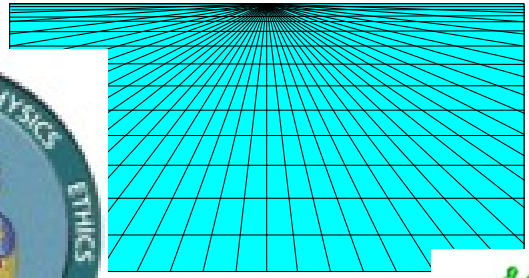
- **Courtesy to Prof. Jack Dongarra**

Why Parallel Computing? – Resource Hungry Applications

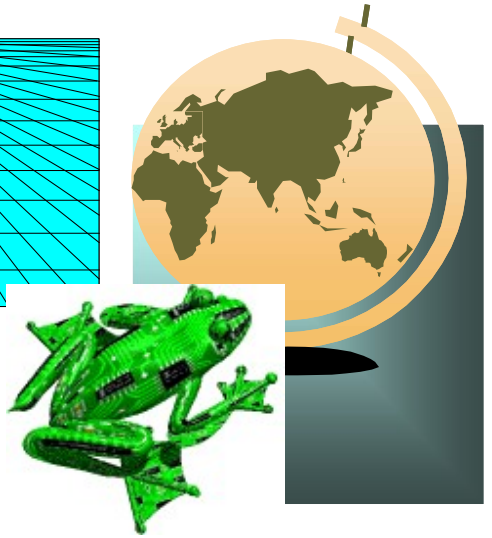
- Solving grand challenge applications using computer *modeling*, *simulation* and *analysis*



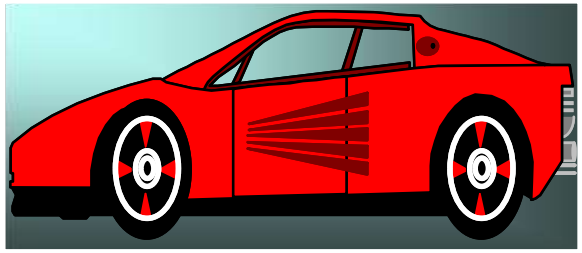
Life Sciences



Aerospace



Internet & Ecommerce



CAD/CAM

Digital Biology

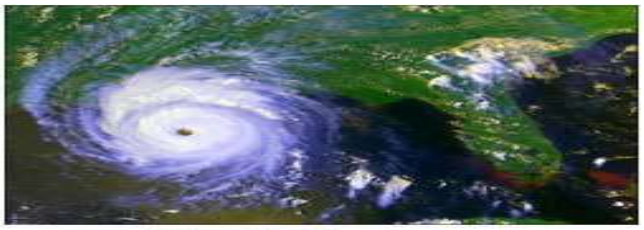
Military Applications

Why Parallel Computing? – Parallel Applications

- **The Real World is Massively Parallel:**
  - In the natural world, many complex, interrelated events are happening at the same time, yet within a temporal sequence.
  - Compared to serial computing, parallel computing is much better suited for modeling, simulating and understanding complex, real world phenomena.
  - For example, imagine modeling these serially:


Galaxy Formation


Planetary Movments


Climate Change

Rush Hour Traffic


Plate Tectonics


Weather

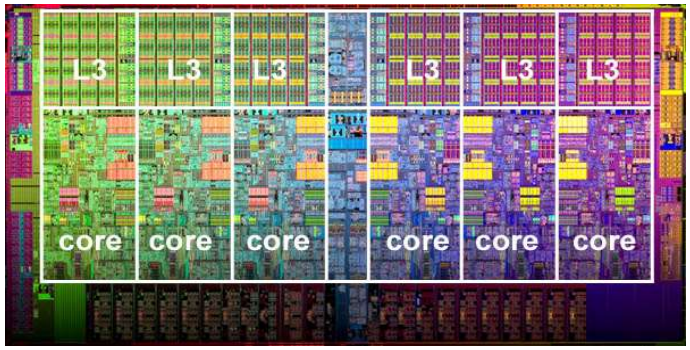

Auto Assembly


Jet Construction


Drive-thru Lunch

Main Reasons of parallel applications:

- Applications
  - ✓ solve problems **more quickly** than a single processor ("*strong scaling*")
  - ✓ solve **larger problems** in the same time as a single processor ("*weak scaling*")
  - ✓ solve problems with higher fidelity
  - ✓ irreplaceable simulations



- Resources
  - ✓ Make better use of underlying parallel hardware and distributed resources

    --Modern computers, even laptops, are parallel in architecture with multiple processors/cores.

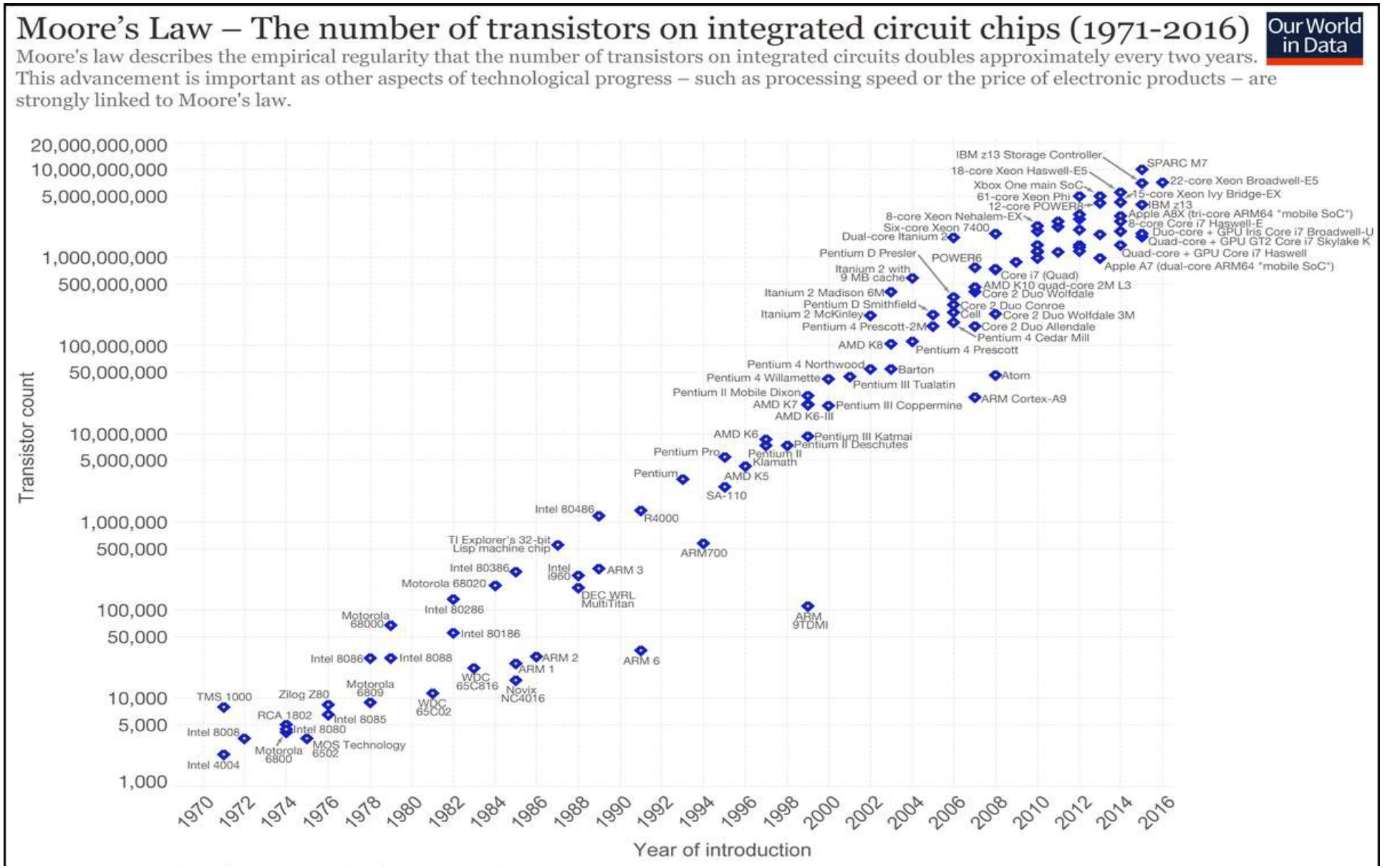
    --Parallel software is specifically intended for parallel hardware with multiple cores, threads, etc.

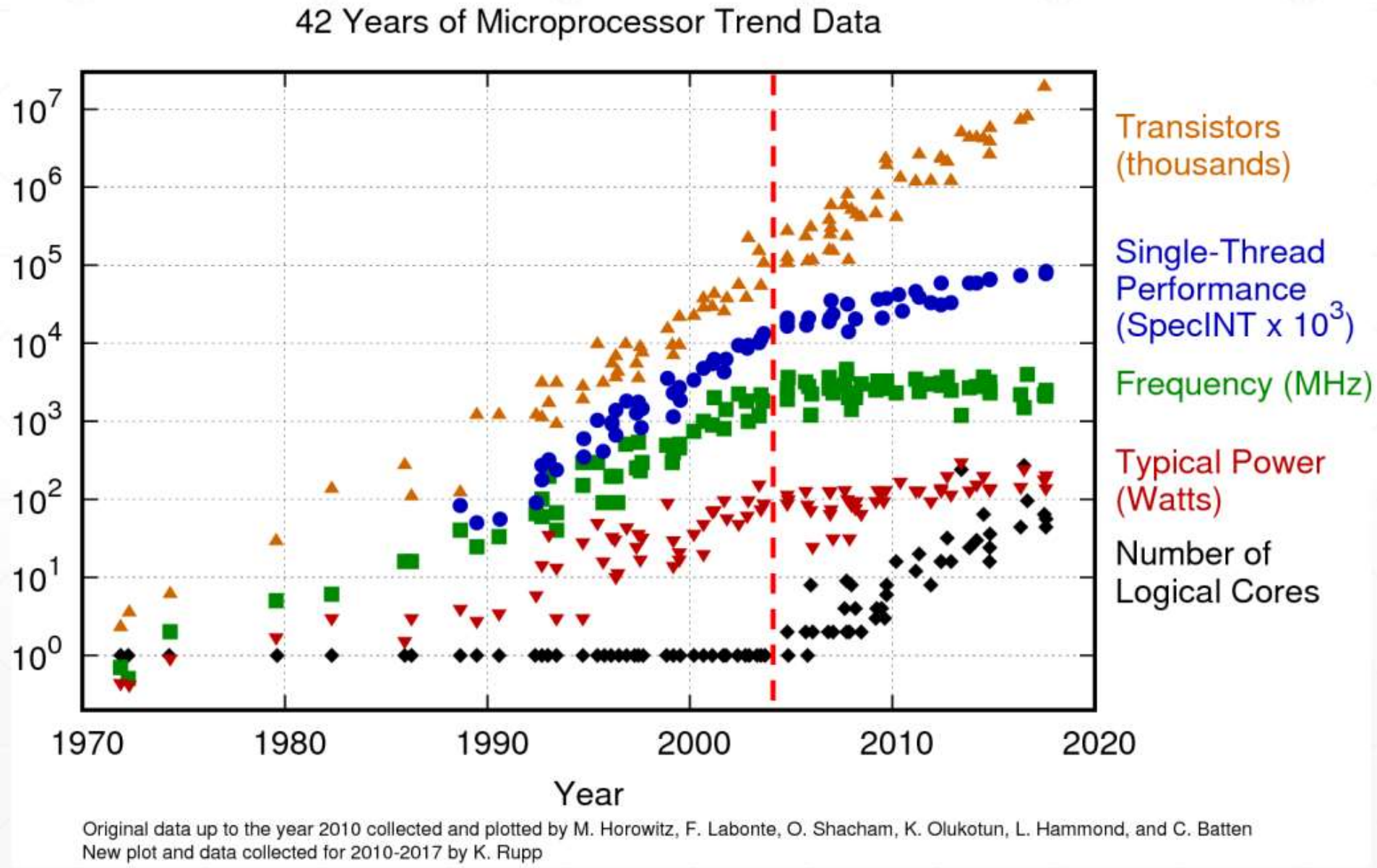
    --In most cases, serial programs run on modern computers "waste" potential computing power.

How to Run Applications Faster ?

- **There are 3 ways to improve performance:**
  - Work Harder
  - Work Smarter
  - Get Help
- **Computer Analogy**
  - Using faster hardware
  - Optimized algorithms and techniques used to solve computational tasks
  - Multiple computers to solve a particular task

Why Parallel Computing? Development of (Parallel) Computers

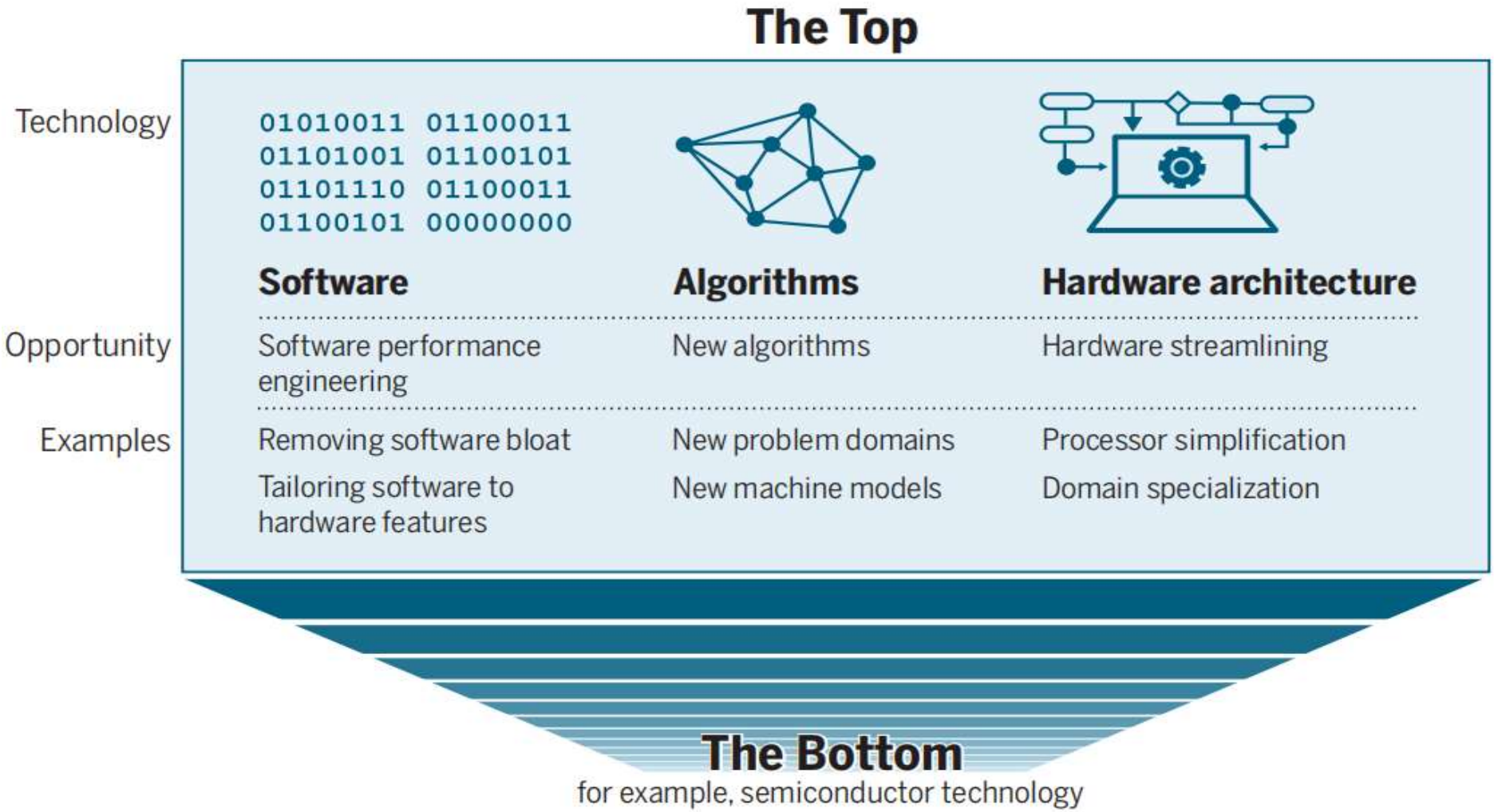



42 Years of Microprocessor Trend Data
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0
1970
1980
1990
2000
2010
2020
Year
Transistors (thousands)
Single-Thread Performance (SpecINT x 10^3)
Frequency (MHz)
Typical Power (Watts)
Number of Logical Cores
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

There's plenty of room at the Top: What will drive computer performance after Moore's law?

CHARLES E. LEISERSON, NEIL C. THOMPSON, JOEL S. EMER, BRADLEY C. KUSZMAUL, BUTLER W. LAMPSON, DANIEL SANCHEZ, AND TAO B. SCHARDL Authors Info & Affiliations

SCIENCE · 5 Jun 2020 · Vol 368, Issue 6495 · DOI: 10.1126/science.aam9744



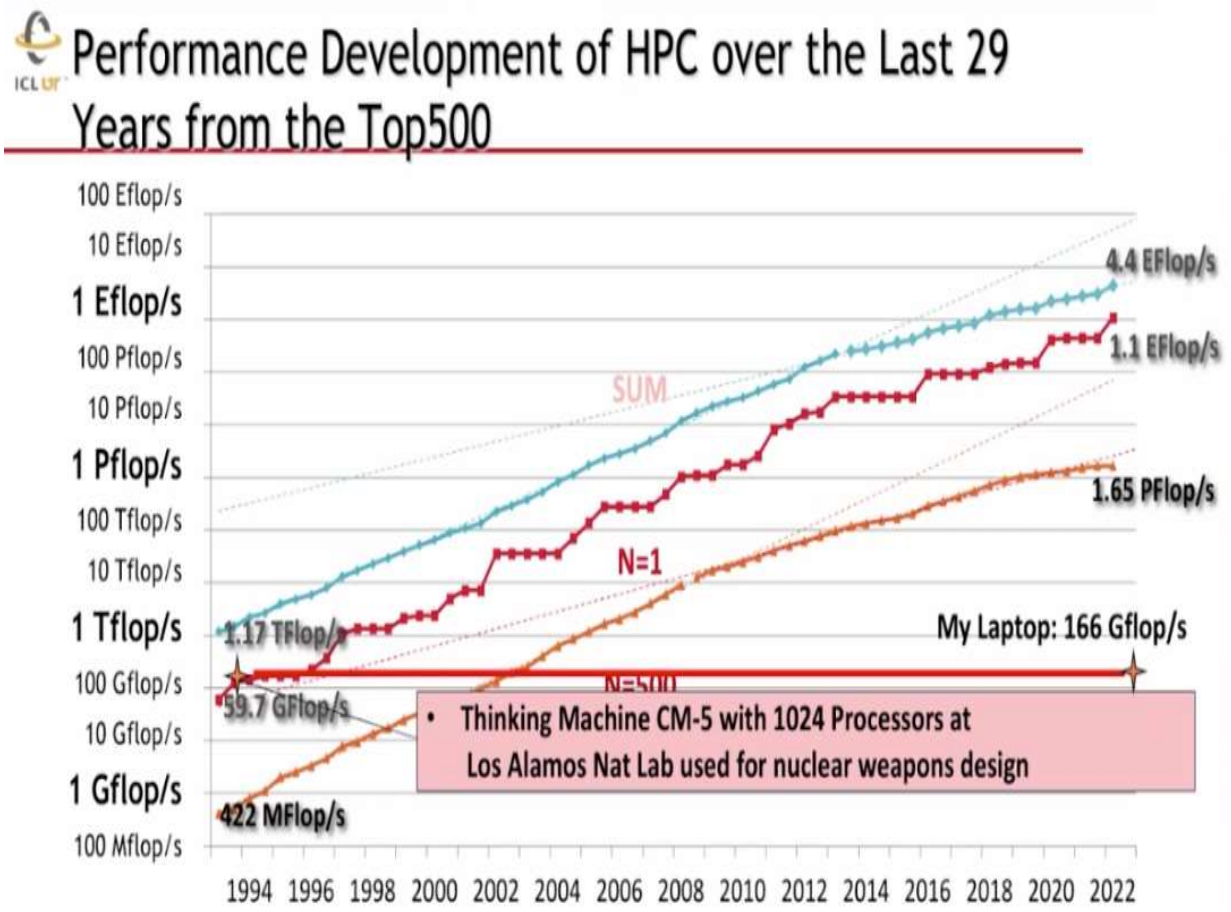
Available Computing Resources (Parallel Computers) www.top500.org

- During the past 20+ years, the trends indicated by ever faster networks, distributed systems, and multi-processor computer architectures (even at the desktop level) clearly show that

  parallelism is the future of computing.

- Exponential growth of supercomputing power as recorded by the **TOP500** list.
- *The race is already on for* ***Exascale Computing!***

  -Exaflop = 10^{18} calculations per second



- **Courtesy to Prof. Jack Dongarra**

2017.6-Top 10

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-2A** - 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/HPE 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/HPE 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/HPE 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, Intel Omni-Path, Fujitsu Joint Center for Advanced High Performance Computing Japan	556,104	13,554.6	24,913.5	2,719
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	12,660
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, Aries interconnect , Cray/HPE DOE/NNSA/LANL/SNL United States	301,056	8,100.9	11,078.9	4,233

2018.6-Top 10

#	Site	Manufacturer	Computer	Country	Cores	Rmax (Pflops)	Power (MW)
1	Oak Ridge National Laboratory	IBM	Summit IBM Power System, P9 22C 3.07GHz, Mellanox EDR, NVIDIA GV100	USA	2,282,544	122.3	8.8
2	National Supercomputing Center in Wuxi	NRCPC	Sunway TaihuLight NRCPC Sunway SW26010, 260C 1.45GHz	China	10,649,600	93.0	15.4
3	Lawrence Livermore National Laboratory	IBM	Sierra IBM Power System, P9 22C 3.1GHz, Mellanox EDR, NVIDIA GV100	USA	1,572,480	71.6	
4	National University of Defense Technology	NUDT	Tianhe-2A ANUDT TH-IVB-FEP, Xeon 12C 2.2GHz, Matrix-2000	China	4,981,760	61.4	18.5
5	National Institute of Advanced Industrial Science and Technology	Fujitsu	AI Bridging Cloud Infrastructure (ABCI) PRIMERGY CX2550 M4, Xeon Gold 20C 2.4GHz, IB-EDR, NVIDIA V100	Japan	391,680	19.9	1.65
6	Swiss National Supercomputing Centre (CSCS)	Cray	Piz Daint Cray XC50, Xeon E5 12C 2.6GHz, Aries, NVIDIA Tesla P100	Switzerland	361,760	19.6	2.27
7	Oak Ridge National Laboratory	Cray	Titan Cray XK7, Opteron 16C 2.2GHz, Gemini, NVIDIA K20x	USA	560,640	17.6	8.21
8	Lawrence Livermore National Laboratory	IBM	Sequoia BlueGene/Q, Power BQC 16C 1.6GHz, Custom	USA	1,572,864	17.2	7.89
9	Los Alamos NL / Sandia NL	Cray	Trinity Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries	USA	979,968	14.1	3.84
10	Lawrence Berkeley National Laboratory	Cray	Cori Cray XC40, Intel Xeons Phi 7250 68C 1.4 GHz, Aries	USA	622,336	14.0	3.94

2019.6-Top 10

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	2,414,592	148,600.0	200,794.9	10,096
2	**Sierra** - IBM Power System S922LC, 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
3	**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
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	**Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR , Dell EMC Texas Advanced Computing Center/Univ. of Texas United States	448,448	23,516.4	38,745.9	
6	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	387,872	21,230.0	27,154.3	2,384
7	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/NNSA/LANL/SNL United States	979,072	20,158.7	41,461.2	7,578
8	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2570 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
9	**SuperMUC-NG** - ThinkSystem SD650, Xeon Platinum 8174 24C 3.1GHz, Intel Omni-Path , Lenovo Leibniz Rechenzentrum Germany	305,856	19,476.6	26,873.9	
10	**Lassen** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, Dual-rail Mellanox EDR Infiniband, NVIDIA Tesla V100 , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	288,288	18,200.0	23,047.2	

2020.6-Top 10

- 1 Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu
- 2 Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM
- 3 Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox
- 4 Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC
- 5 Tianhe-2A - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT
- 6 HPC5 - PowerEdge C4140, Xeon Gold 6252 24C 2.1GHz, NVIDIA Tesla V100, Mellanox HDR Infiniband, Dell EMC
- 7 Selene - DGX A100 SuperPOD, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia
- 8 Frontera - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR, Dell EMC
- 9 Marconi-100 - IBM Power System AC922, IBM POWER9 16C 3GHz, Nvidia Volta V100, Dual-rail Mellanox EDR Infiniband, IBM
- 10 Piz Daint - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100, Cray/HPE

2021.6-Top 10

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
7	**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
8	**JUWELS Booster Module** - Bull Sequana XH2000 , AMD EPYC 7402 24C 2.8GHz, NVIDIA A100, Mellanox HDR InfiniBand/ParTec ParaStation ClusterSuite, Atos Forschungszentrum Juelich (FZJ) Germany	449,280	44,120.0	70,980.0	1,764
9	**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
10	**Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR, Dell EMC Texas Advanced Computing Center/Univ. of Texas United States	448,448	23,516.4	38,745.9	

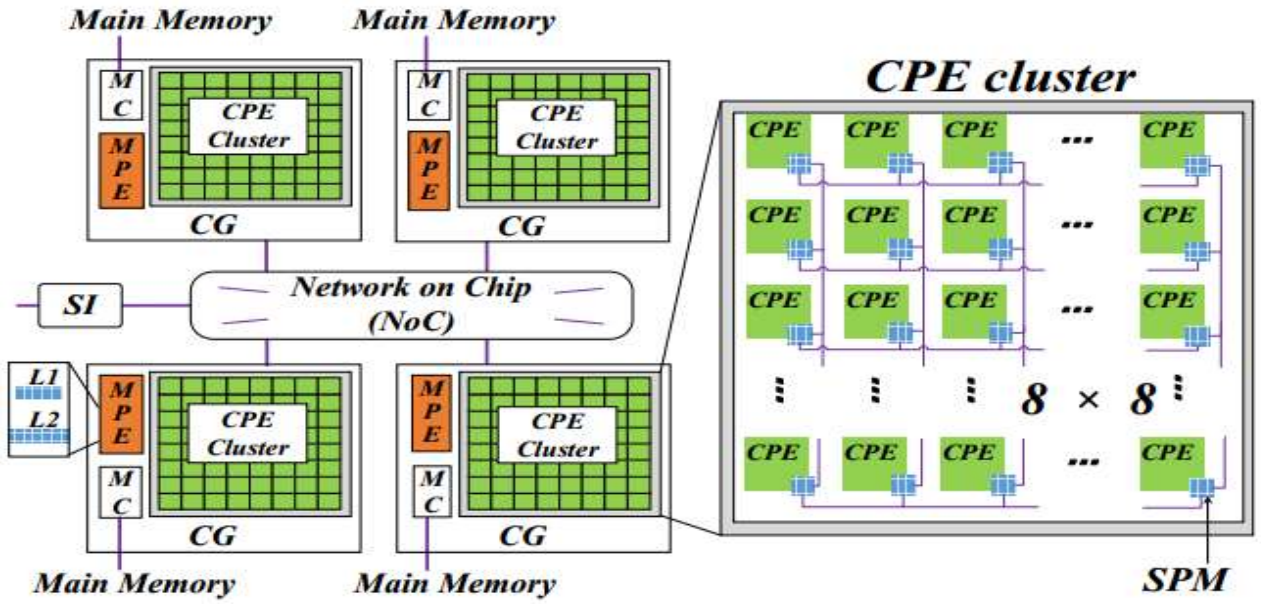
2022.6-Top 10

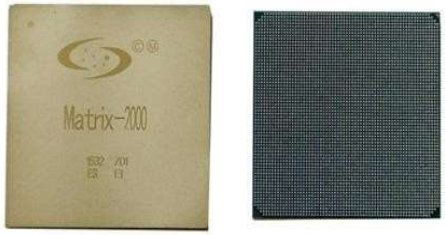
June 2022: The TOP 10 Systems (45% of the Total Performance of Top500)

Rank	Site	Computer	Country	Cores	Rmax [Pflops]	% of Peak	Power [MW]	GFlops/ Watt
1	DOE / OS Oak Ridge Nat Lab	Frontier, HPE Cray Ex235a, AMD 3rd EPYC 64C, 2 GHz, AMD Instinct MI250X, Slingshot 10	USA	7,733,248	1,102	55	21.1	52.2
2	RIKEN Center for Computational Science	Fugaku, ARM A64FX (48C, 2.2 GHz), Tofu D Interconnect	Japan	7,299,072	442.	82	29.9	14.8
3	EuroHPC /CSC	LUMI, HPE Cray EX235a, AMD 3rd EPYC 64C, 2 GHz, AMD Instinct MI250X, Slingshot 10	Finland	1,268,736	152.	71	2.94	51.6
4	DOE / OS Oak Ridge Nat Lab	Summit, IBM Power 9 (22C, 3.0 GHz), NVIDIA GV100 (80C), Mellonox EDR	USA	2,397,824	149.	74	10.1	14.7
5	DOE / NNSA L Livermore Nat Lab	Sierra, IBM Power 9 (22C, 3.1 GHz), NVIDIA GV100 (80C), Mellonox EDR	USA	1,572,480	94.6	75	7.44	12.7
6	National Super Computer Center in Wuxi	Sunway TaihuLight, SW26010 (260C), Custom Interconnect	China	10,649,000	93.0	74	15.4	6.05
7	DOE / OS NERSC - LBNL	Perlmutter HPE Cray EX235n, AMD EPYC 64C 2.45GHz, NVIDIA A100, Slingshot 10	USA	706,304	64.6	69	2.53	25.5
8	NVIDIA Corporation	Selene NVIDIA DGX A100, AMD EPYC 7742 (64C, 2.25GHz), NVIDIA A100 (108C), Mellanox HDR	USA	555,520	63.4	80	2.64	23.9
9	National Super Computer Center in Guangzhou	Tianhe-2A NUDT, Xeon (12C) , MATRIX-2000 (128C) + Custom Interconnect	China	4,981,760	61.4	61	18.5	3.32
10	Grand Equipment Nat de Calcul GENCI-CINES	Bull Sequana XH2000 , AMD EPYC 7402 (24C, 2.8GHz), NVIDIA A100 (108C), Mellanox HDR/ParTec ParaStation ClusterSuite	France	448,280	44.1	62	1.76	25.0

- Courtesy to Prof. Jack Dongarra

The general architecture of SW26010 processor



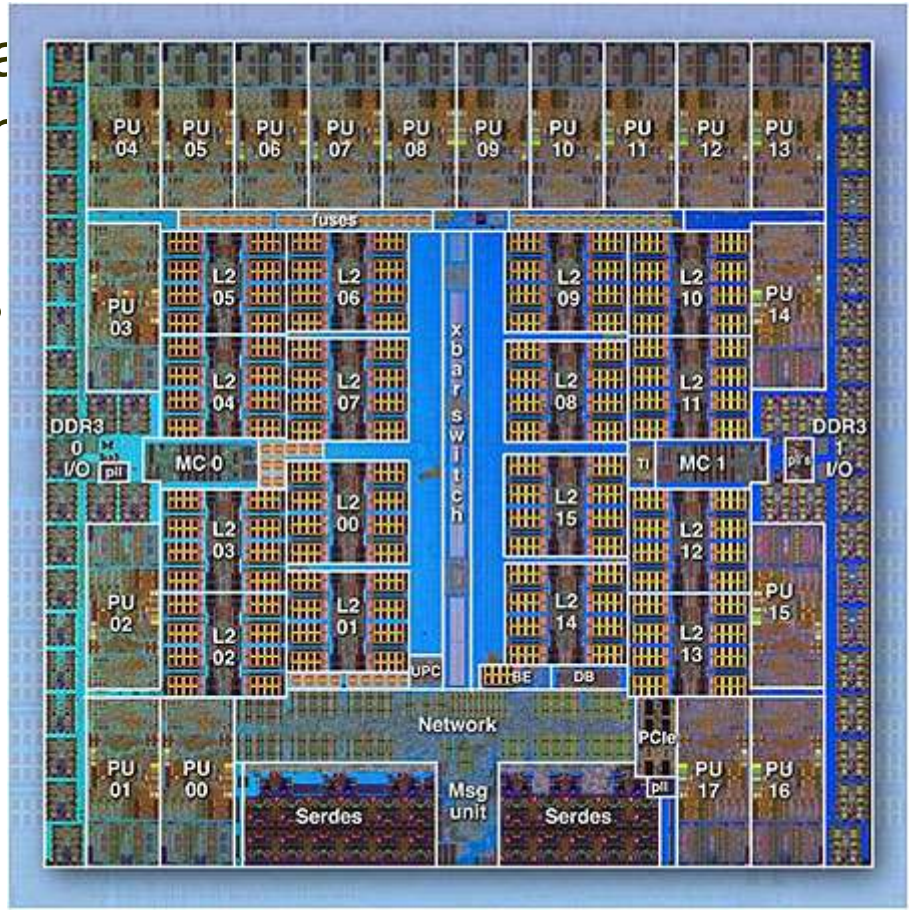


9	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou China	4,981,760	61.44	100.68	18,482

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

Parallel computers are basically everywhere

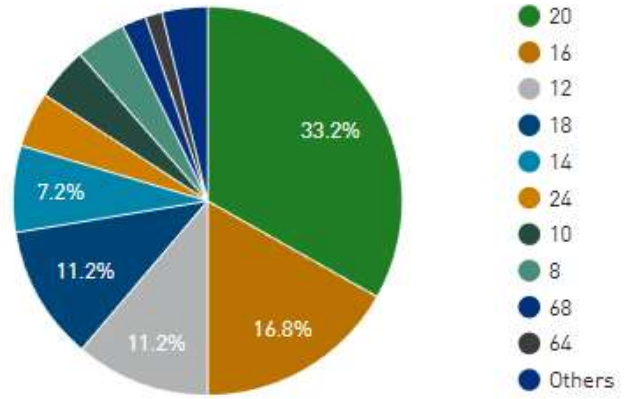
- Virtually all stand-alone computers today are parallel from a hardware perspective:
  - Multiple functional units (L1 ca
    prefetch, decode, floating-poir
    (GPU), integer, etc.)
  - Multiple execution units/cores
  - Multiple hardware threads



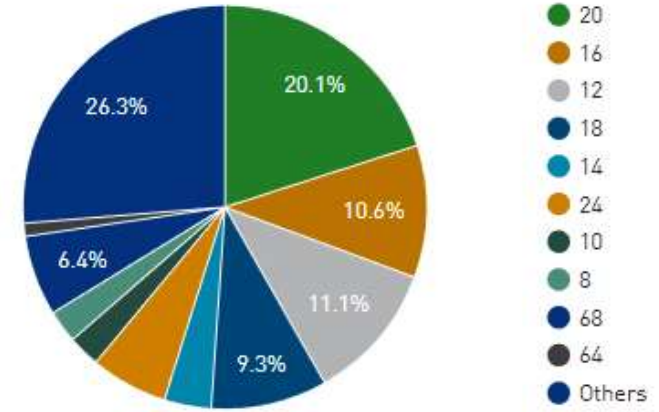
➢ IBM BG/Q Compute Chip with 18 cores (PU) and 16 L2 Cache units (L2)

Cores per socket of TOP500 Supercomputers

Cores per Socket System Share



Cores per Socket Performance Share



Cores per Socket	Count	System Share (%)	Rmax (GFlops)	Rpeak (GFlops)	Cores
20	166	33.2	312,717,084	478,122,278	8,858,072
16	84	16.8	165,100,590	280,612,384	7,242,844
12	56	11.2	173,828,326	308,958,193	9,682,734
18	56	11.2	145,115,861	204,441,480	3,768,648
14	36	7.2	58,642,962	117,849,925	2,011,308
24	23	4.6	96,630,718	146,894,409	1,899,456
10	22	4.4	40,057,390	76,701,397	1,687,988
8	21	4.2	40,800,395	68,595,479	2,440,568
68	10	2	100,520,050	190,813,155	4,228,348
64	7	1.4	16,406,620	28,792,934	692,656
22	6	1.2	264,924,540	353,808,349	4,346,464
6	4	0.8	7,648,000	15,494,585	437,040
32	4	0.8	12,768,000	15,480,288	449,104
28	3	0.6	29,564,250	47,574,374	663,152
4	1	0.2	1,836,000	4,297,421	48,384
260	1	0.2	93,014,594	125,435,904	10,649,600

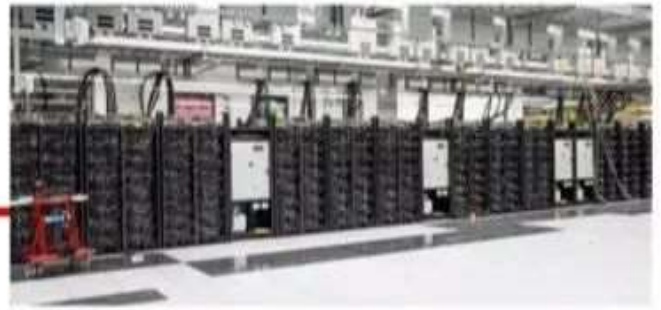
Accelerator/Co-Processor

Accelerator/Co-Processor	Count	System Share (%)	Rmax (GFlops)	Rpeak (GFlops)	Cores
NVIDIA Tesla V100	52	10.4	99,558,070	196,283,310	2,331,252
NVIDIA Tesla P100	47	9.4	95,683,760	162,862,507	2,195,396
NVIDIA Tesla V100 SXM2	7	1.4	45,658,000	73,589,705	872,064
NVIDIA Tesla K20x	4	0.8	24,615,000	38,926,835	731,712
NVIDIA Tesla K40	3	0.6	8,824,090	14,612,320	201,328
NVIDIA Tesla K80	3	0.6	7,480,570	12,110,830	247,070
NVIDIA Volta GV100	3	0.6	261,100,000	351,532,690	4,278,096
Intel Xeon Phi 5120D	2	0.4	2,571,200	4,099,204	122,512
NVIDIA 2050	2	0.4	3,837,000	7,685,300	307,008
NVIDIA Tesla P100 NVLink	2	0.4	9,147,000	13,611,069	153,468
NVIDIA Tesla K20m	1	0.2	1,029,000	3,024,959	12,592
NVIDIA Tesla P40	1	0.2	1,036,000	2,112,000	210,000
Intel Xeon Phi 7120P	1	0.2	1,457,730	2,011,641	76,896
Intel Xeon Phi 31S1P	1	0.2	2,071,390	3,074,534	174,720
NVIDIA Tesla K40m	1	0.2	2,478,000	4,946,790	64,384
Intel Xeon Phi 5110P	1	0.2	2,539,130	3,388,032	194,616
NVIDIA Tesla K40/Intel Xeon Phi 7120P	1	0.2	3,126,240	5,610,481	152,692
Deep Computing Processor	1	0.2	4,325,000	6,134,170	163,840
Matrix-2000	1	0.2	61,444,500	100,678,664	4,981,760

NVIDIA Accelerator Specification Comparison

	A100	V100	P100
FP32 CUDA Cores	6912	5120	3584
Boost Clock	~1.41GHz	1530MHz	1480MHz
Memory Clock	2.4Gbps HBM2	1.75Gbps HBM2	1.4Gbps HBM2
Memory Bus Width	5120-bit	4096-bit	4096-bit
Memory Bandwidth	1.6TB/sec	900GB/sec	720GB/sec
VRAM	40GB	16GB/32GB	16GB
Single Precision	19.5 TFLOPs	15.7 TFLOPs	10.6 TFLOPs
Double Precision	9.7 TFLOPs (1/2 FP32 rate)	7.8 TFLOPs (1/2 FP32 rate)	5.3 TFLOPs (1/2 FP32 rate)
INT8 Tensor	624 TOPs	N/A	N/A
FP16 Tensor	312 TFLOPs	125 TFLOPs	N/A
TF32 Tensor	156 TFLOPs	N/A	N/A
Interconnect	NVLink 3 12 Links (600GB/sec)	NVLink 2 6 Links (300GB/sec)	NVLink 1 4 Links (160GB/sec)
GPU	A100 (826mm2)	GV100 (815mm2)	GP100 (610mm2)
Transistor Count	54.2B	21.1B	15.3B
TDP	400W	300W/350W	300W
Manufacturing Process	TSMC 7N	TSMC 12nm FFN	TSMC 16nm FinFET
Interface	SXM4	SXM2/SXM3	SXM
Architecture	Ampere	Volta	Pascal

Example of a Typical Supercomputer



Combination of shared memory and distributed memory programming

- **Courtesy to Prof. Jack Dongarra**

IBM POWER5

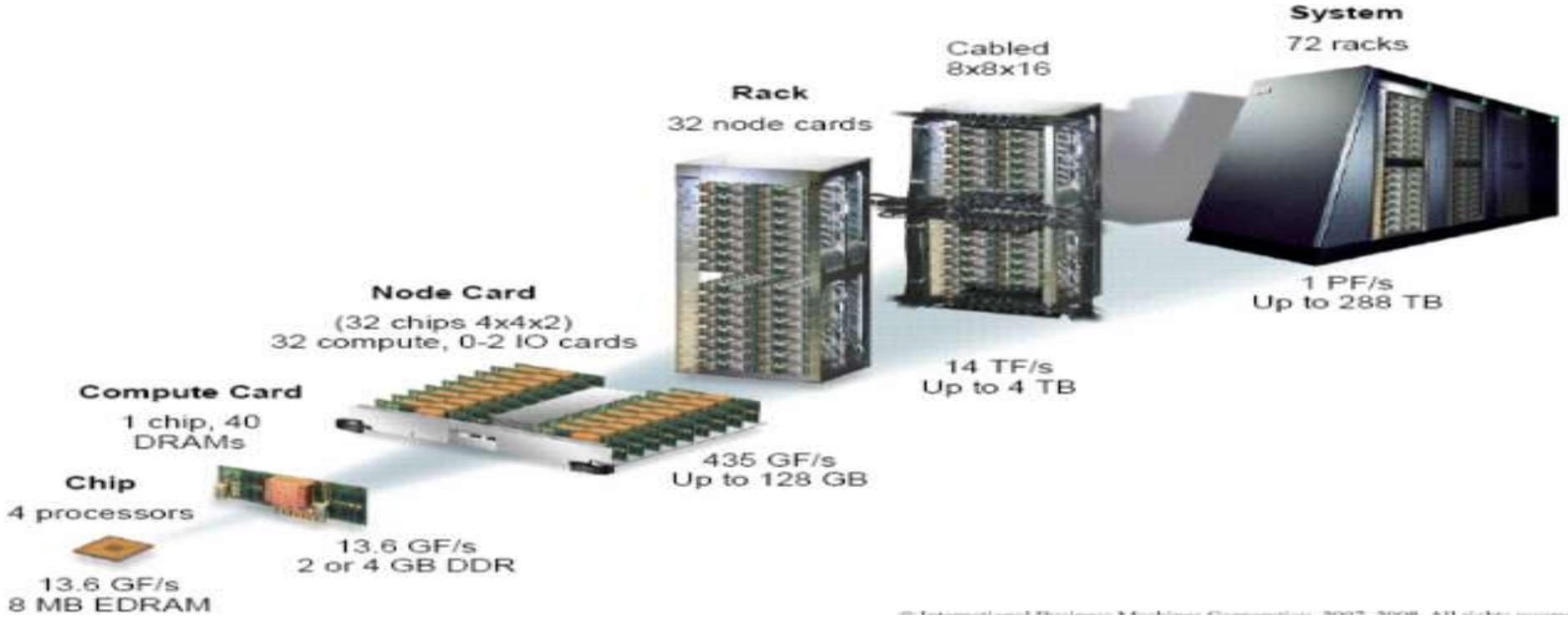
HP/Compaq Alphaserver

Intel IA32

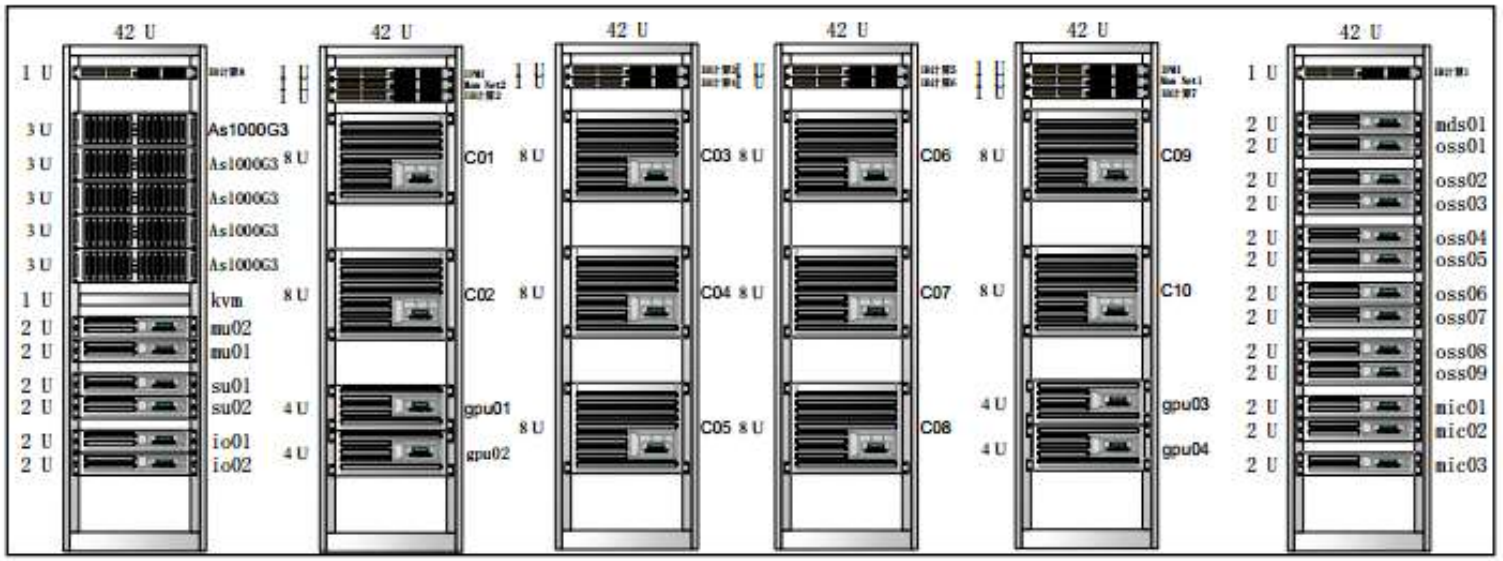
AMD Opteron

Cray XT3

IBM BG/L

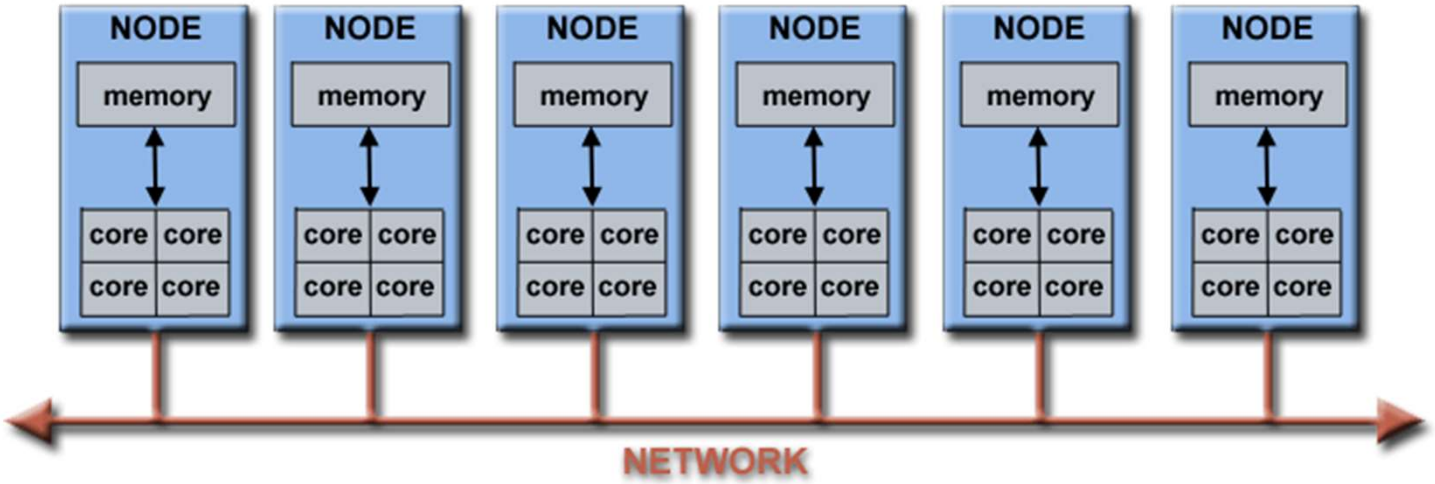


Inspur Cluster @ NPU



HPC Clusters

- Networks connect multiple stand-alone computers (nodes) to make larger parallel computer clusters.



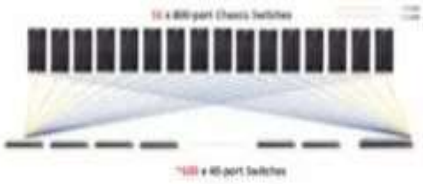
Today's HPC Environment for Scientific Computing

- Highly parallel
  - Distributed memory
  - MPI + Open-MP programming model

ORNL Frontier, 2 Eflop/s,
8.8 x 10^6 Cores, 9408 nodes, 30 MW
(node = 1-AMD CPU + 4-AMD GPUs)
> 98% of performance from GPUs

- Heterogeneous
  - Commodity processors + GPU accelerators

- Communication between parts very expensive compared to floating point ops

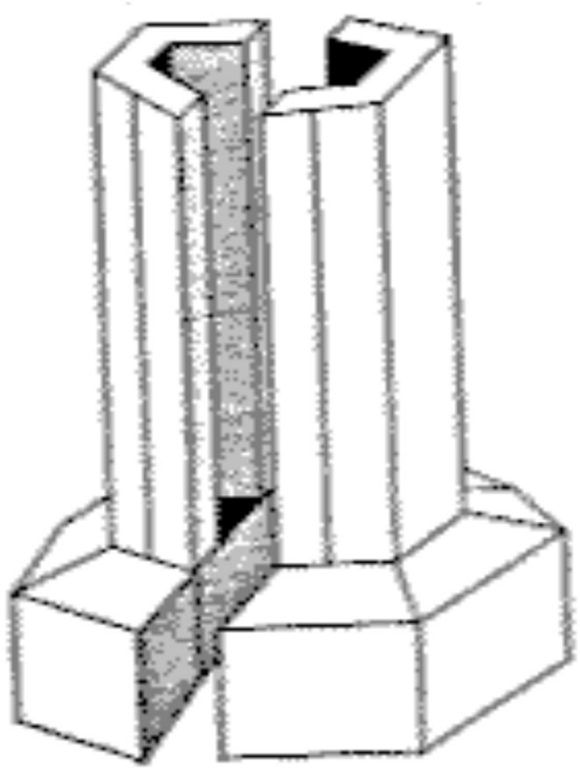


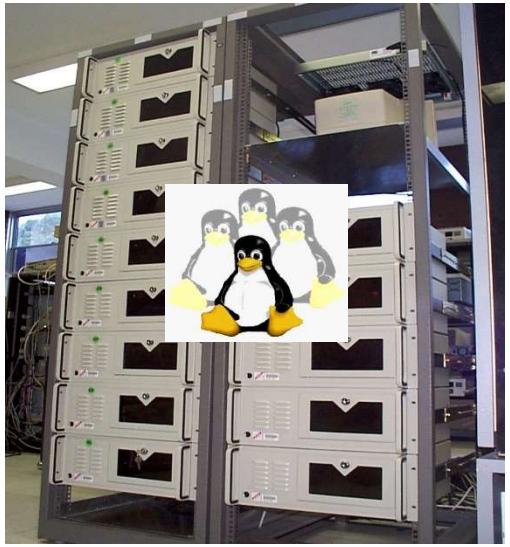
- Floating point hardware at 64, 32, 16, & 8 bit levels

Type	Size	Range	$u = 2^{-t}$
half	16 bits	$10^{\pm 5}$	$2^{-11} \approx 4.9 \times 10^{-4}$
single	32 bits	$10^{\pm 38}$	$2^{-24} \approx 6.0 \times 10^{-8}$
double	64 bits	$10^{\pm 308}$	$2^{-53} \approx 1.1 \times 10^{-16}$
quadruple	128 bits	$10^{\pm 4932}$	$2^{-113} \approx 9.6 \times 10^{-35}$

- **Courtesy to Prof. Jack Dongarra**

Towards Personal Clusters





amazon web services

阿里云计算
Alibaba Cloud Computing

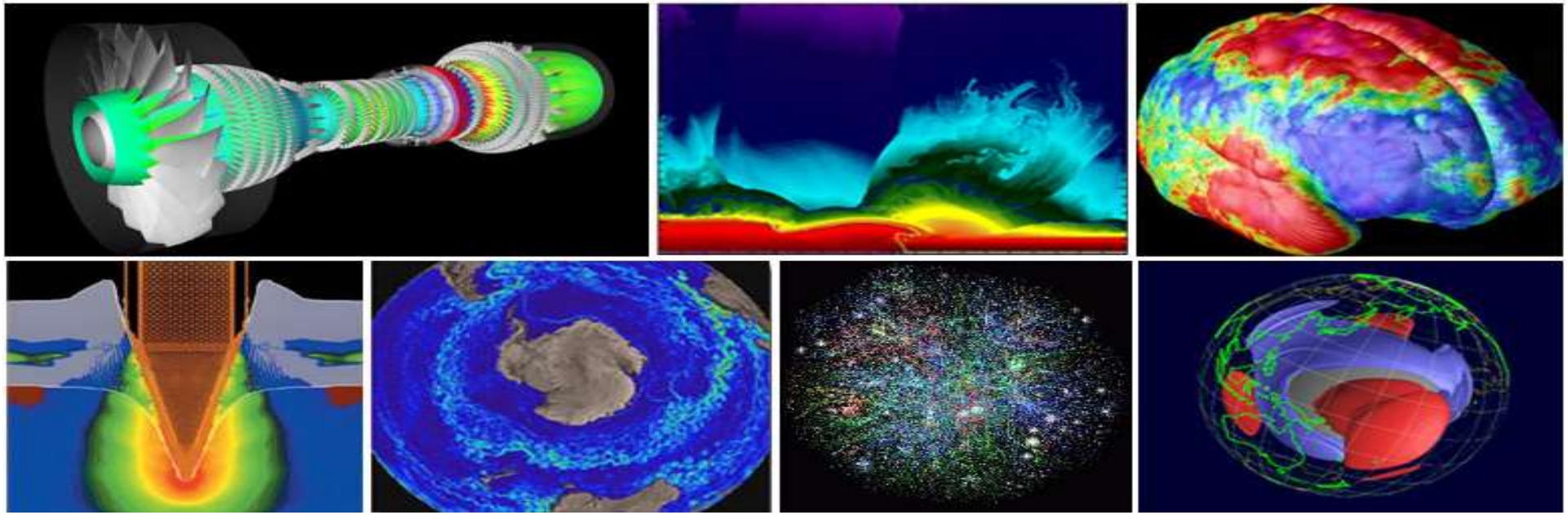
The promise of supercomputing to the average PC User ?

Who is using parallel computing/applications?

- **Science and Engineering, (graduated) students and teachers in universities/research institute**

  -Historically, parallel computing has been considered to be "the high end of computing", and has been used to model difficult problems in many areas of science and engineering:

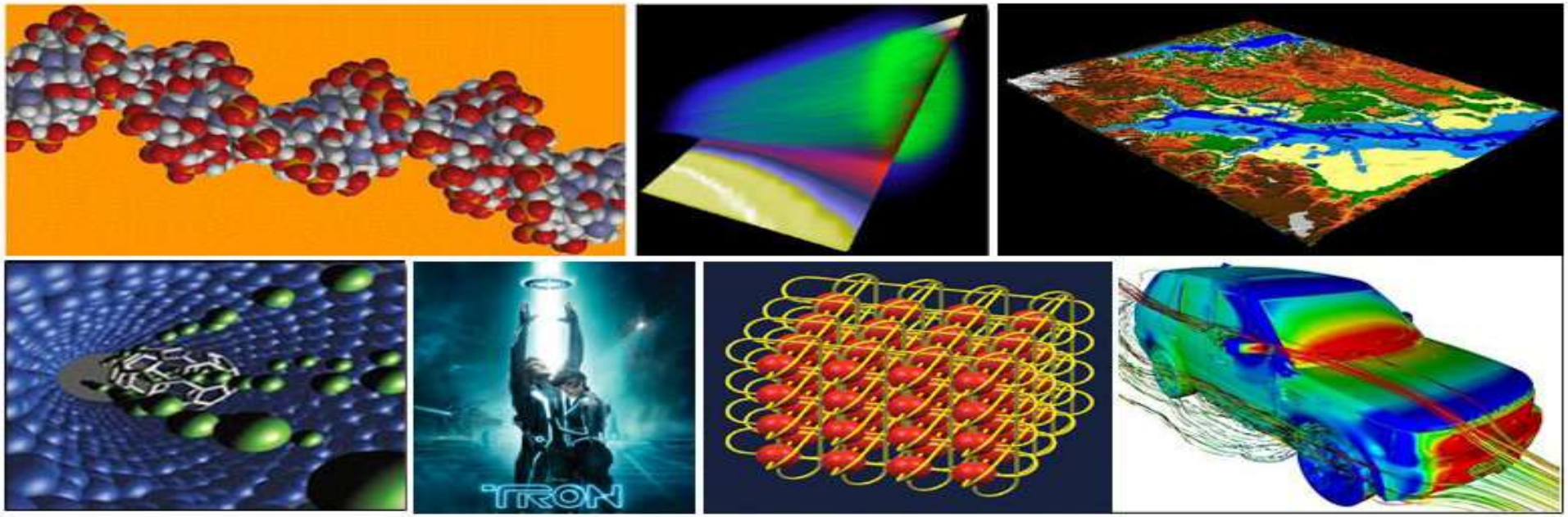
- Atmosphere, Earth, Environment
- Physics - applied, nuclear, particle, condensed matter, high pressure, fusion, photonics
- Bioscience, Biotechnology, Genetics
- Chemistry, Molecular Sciences
- Geology, Seismology
- Mechanical Engineering - from prosthetics to spacecraft
- Electrical Engineering, Circuit Design, Microelectronics
- Computer Science, Mathematics
- Defense, Weapons

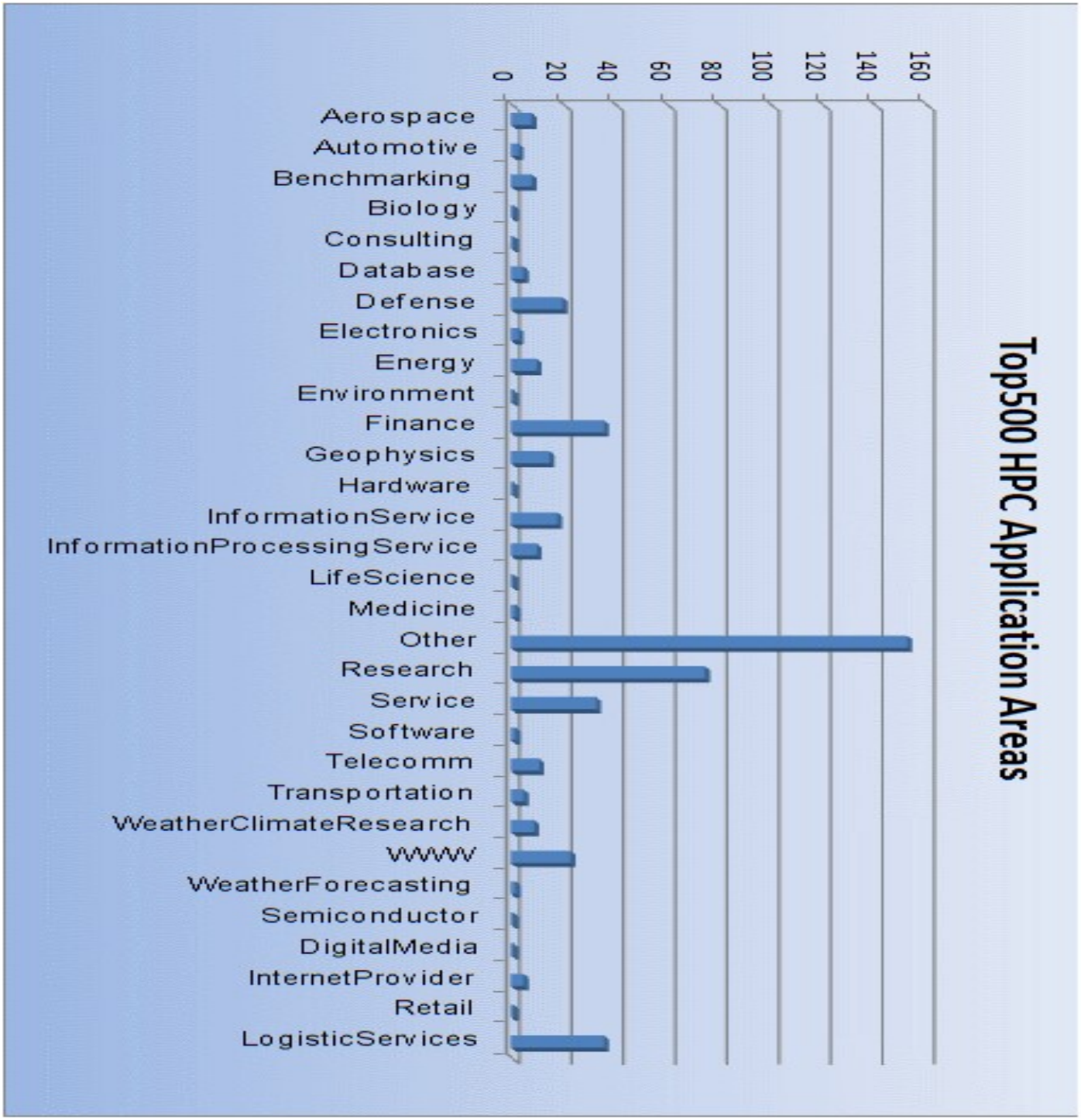


Industrial and Commercial:

- Today, commercial applications provide an equal or greater driving force in the development of faster computers. These applications require the processing of large amounts of data in sophisticated ways. For example: *AI and Deep learning*

  - "Big Data", databases, data mining
  - Oil exploration
  - Web search engines, web based business services
  - Medical imaging and diagnosis
  - Pharmaceutical design
  - Financial and economic modeling
  - Management of national and multi-national corporations
  - Advanced graphics and virtual reality, particularly in the entertainment industry
  - Networked video and multi-media technologies
  - Collaborative work environments




Top500 HPC Application Areas
0
20
40
60
80
100
120
140
160
Aerospace
Automotive
Benchmarking
Biology
Consulting
Database
Defense
Electronics
Energy
Environment
Finance
Geophysics
Hardware
InformationService
InformationProcessingService
LifeScience
Medicine
Other
Research
Service
Software
Telecomm
Transportation
WeatherClimateResearch
WWW
WeatherForecasting
Semiconductor
DigitalMedia
InternetProvider
Retail
LogisticServices

Why we need to write parallel programs?

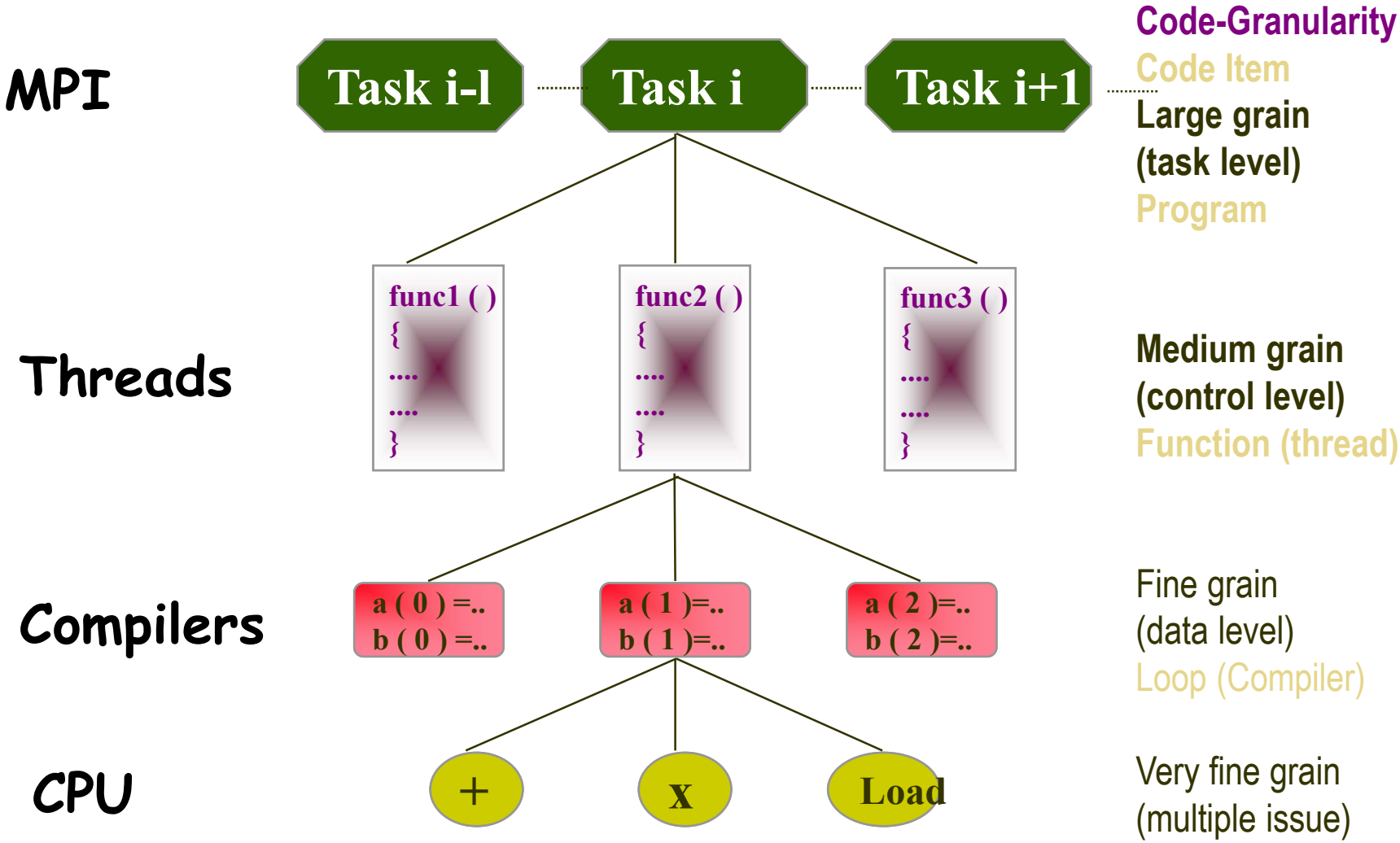
- Most programs that have been written for conventional, single-core systems cannot exploit the presence of multiple cores
- We can run multiple instances of a program on a multicore system, but this is often of little help
- We need to either rewrite our serial programs so that they're parallel, and they can make use of multiple cores;

  or write translation programs, that is, programs that will automatically convert serial programs into parallel programs (very hard)

How to write parallel programs?

- Most of the answers depend on the basic idea of *partitioning* the work to be done among the cores
  - **Task parallelism**
  - **Data parallelism**

Levels of Parallelism



Concepts and Terminology

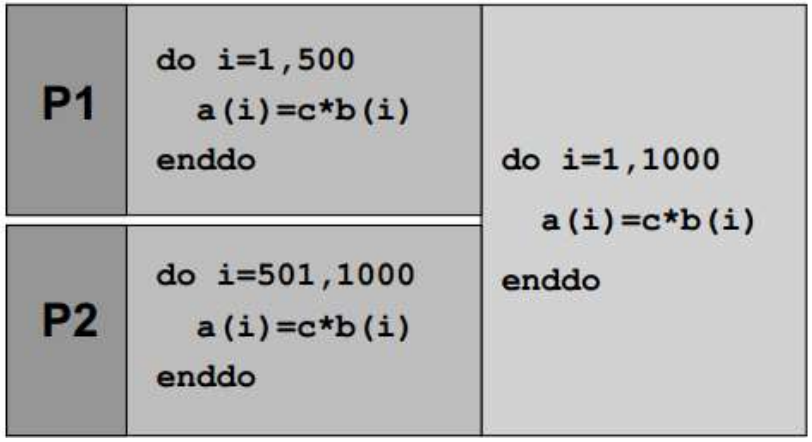
- Parallelism (data parallelism and task parallelism)
- Flynn's Classical Taxonomy
- Some General Parallel Terminology

Parallelism

- Writing a parallel program must always start by identifying the parallelism inherent in the algorithm at hand. Different variants of parallelism induce different methods of parallelization.

- **Data parallelism**
  -Many problems in scientific computing involve processing of large quantities of data stored on a computer. If this manipulation can be performed in parallel, i.e., by multiple processors working on different parts of the data, we speak of ***data parallelism***.

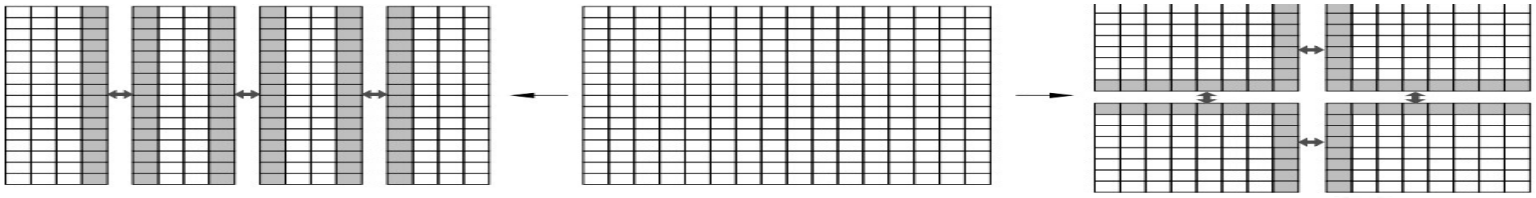
- As a matter of fact, data parallelism is the dominant parallelization concept in scientific computing on MIMD-type computers.
- It also goes under the name of *SPMD* (Single Program Multiple Data), as usually the same code is executed on all processors, with independent instruction pointers. It is thus not to be confused with SIMD parallelism.



-An example for medium-grained parallelism: The iterations of a loop are distributed to two processors P1 and P2 (in shared memory) for concurrent execution.

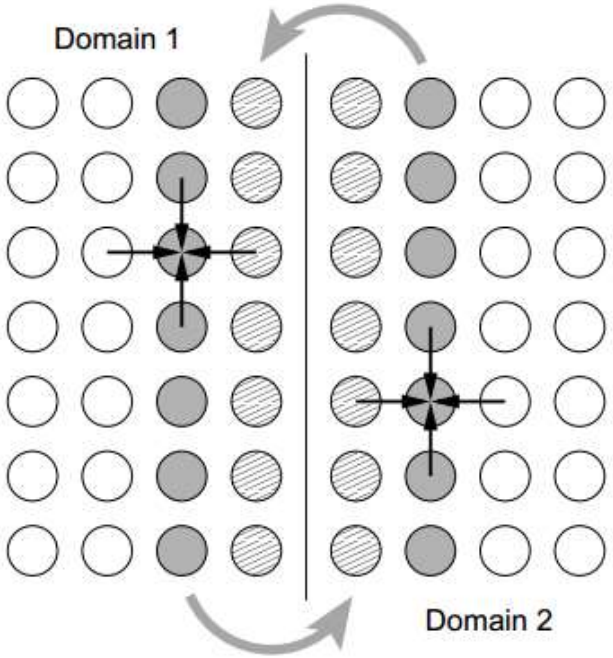
Coarse-grained parallelism by domain decomposition

- Simulations of physical processes (like, e.g., fluid flow, mechanical stress, quantum fields) often work with a simplified picture of reality in which a *computational domain*, e.g., some volume of a fluid, is represented as a *grid* that defines discrete positions for the physical quantities under consideration
- The goal of the simulation is usually the computation of observables on this grid. A straightforward way to distribute the work involved across workers, i.e., processors, is to assign a part of the grid to each worker. This is called ***domain decomposition***.



Jacobi Iteration

$$B(i,j) = 0.25 * (A(i-1,j) + A(i,j-1) + A(i+1,j) + A(i,j+1))$$



- Using halo (“ghost”) layers for communication across domain boundaries in a distributed-memory parallel Jacobi solver. After the local updates in each domain, the boundary layers (shaded) are copied to the halo of the neighboring domain (hatched)

Functional/Task parallelism

- Sometimes the solution of a “big” numerical problem can be split into more or less disparate subtasks, which work together by data exchange and synchronization.
- In this case, the subtasks execute completely different code on different data items, which is why functional parallelism is also called *MPMD* (Multiple Program Multiple Data). This does not rule out, however, that each subtask could be executed in parallel by several processors in an SPMD fashion.

Examples of functional parallelism

- **Master-worker scheme**

  -Reserving one compute element for administrative tasks while all others solve the actual problem is called the *master-worker* scheme. The master distributes work and collects results.

- **Functional decomposition**

  -Multiphysics simulations are prominent applications for parallelization by functional decomposition. For instance, the airflow around a racing car could be simulated using a parallel **CFD (Computational Fluid Dynamics)** code. On the other hand, a parallel **finite element** simulation could describe the reaction of the flexible structures of the car body to the flow, according to their geometry and material properties. Both codes have to be coupled using an appropriate communication layer.

Some cases that can not be parallelized

- Recursion

```
for (i = 1; i < N; i++)
  a[i] = a[i-1] + b[i];
```

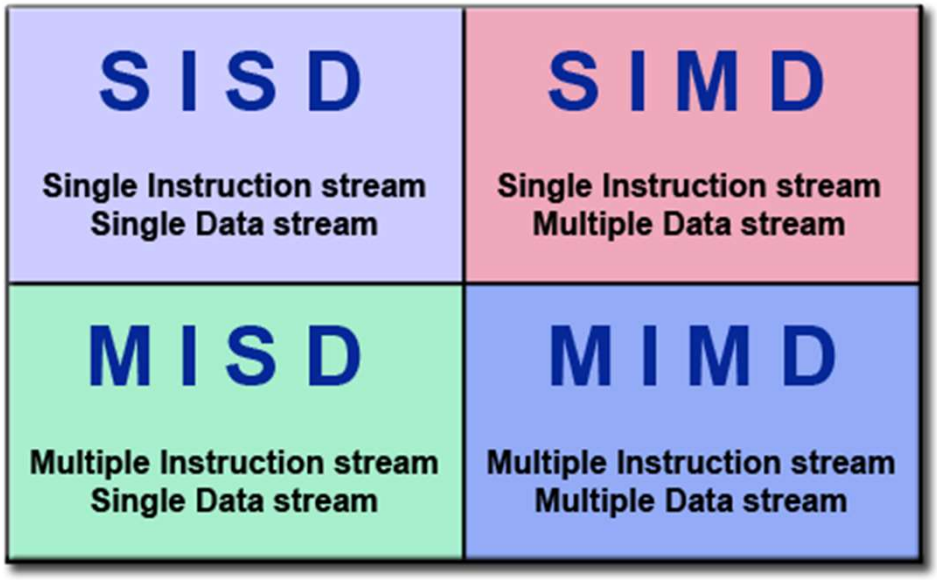
Fibonacci number

- Loop-Carried Dependence

```
for (k = 5; k < N; k++) {
  b[k] = DoSomething(k);
  a[k] = b[k-5] + MoreStuff(k);
}
```

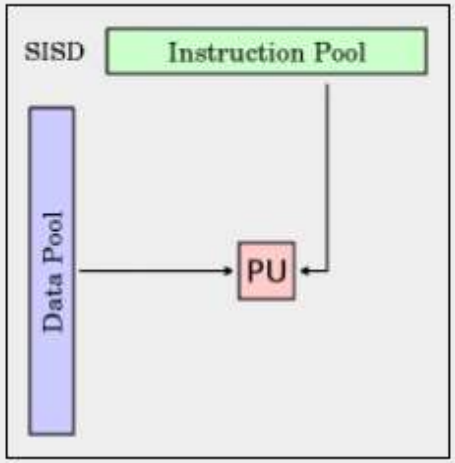
Flynn's Classical Taxonomy

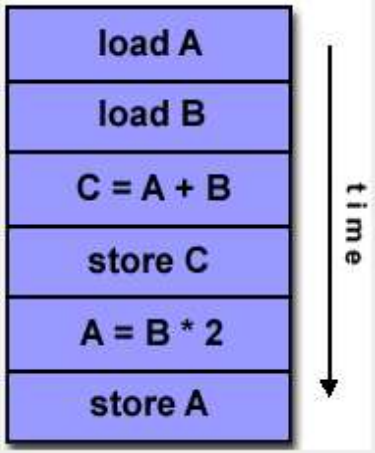
- There are different ways to classify parallel computers. One of the more widely used classifications, in use since 1966, is called **Flynn's Taxonomy.**
- Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of ***Instruction Stream*** and ***Data Stream***. Each of these dimensions can have only one of two possible states: ***Single*** or ***Multiple***.



Single Instruction, Single Data (SISD)

- A serial (non-parallel) computer
- **Single Instruction:** Only one instruction stream is being acted on by the CPU during any one clock cycle
- **Single Data:** Only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest type of computer
- Examples: older generation mainframes, minicomputers, workstations and single processor/core PCs.





UNIVAC1

IBM 360

CRAY1

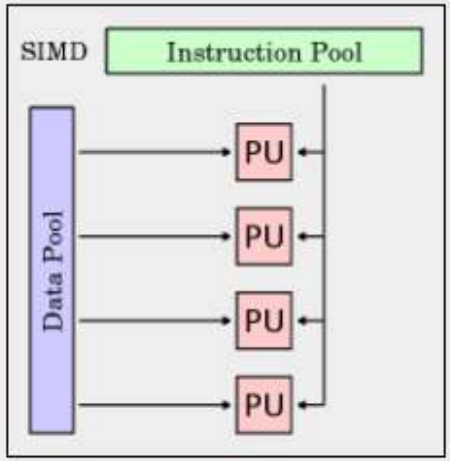
CDC 7600

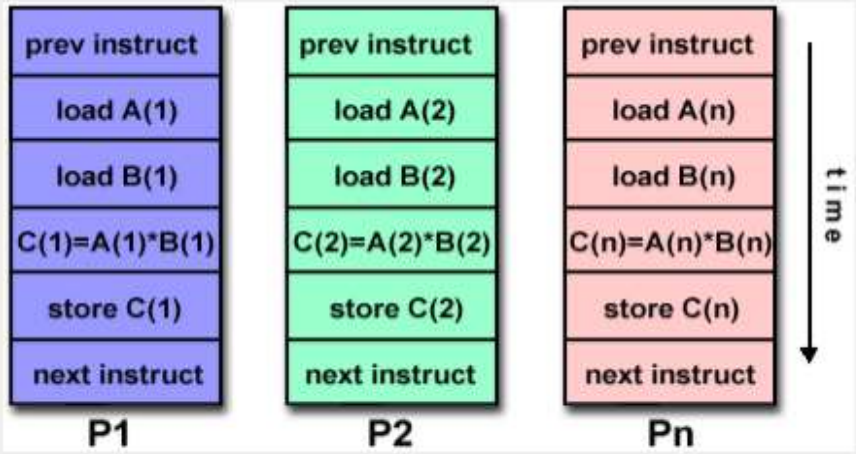
PDP1

Dell Laptop

Single Instruction, Multiple Data (SIMD):

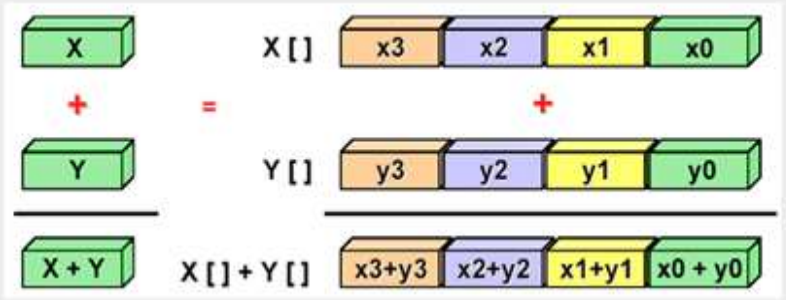
- A type of parallel computer
- **Single Instruction:** All processing units execute the same instruction at any given clock cycle
- **Multiple Data:** Each processing unit can operate on a different data element
- Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines
- Examples:
  - Processor Arrays: Thinking Machines CM-2, MasPar MP-1 & MP-2, ILLIAC IV
  - Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10
- Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.





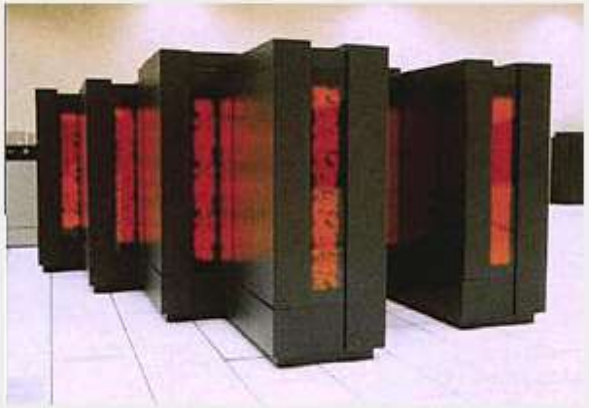
ILLIAC IV

MasPar

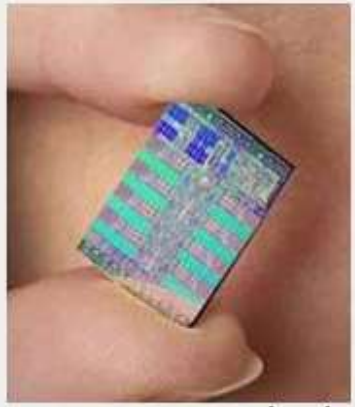


Cray X-MP

Cray Y-MP



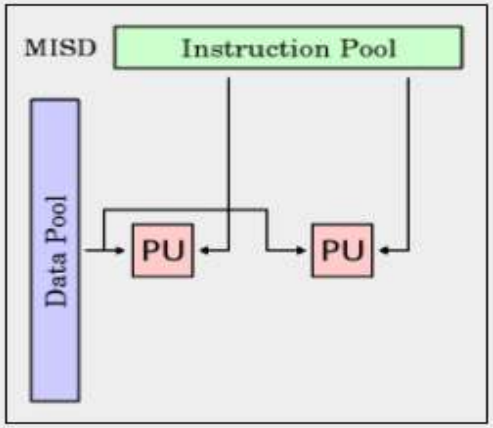
Thinking Machines CM-2

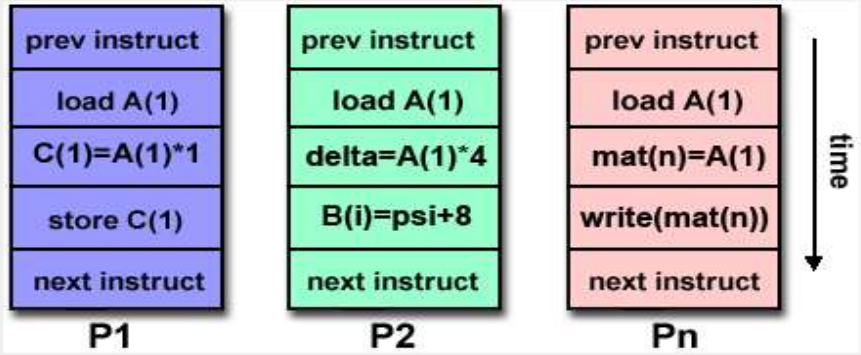


Cell Processor (GPU)

Multiple Instruction, Single Data (MISD):

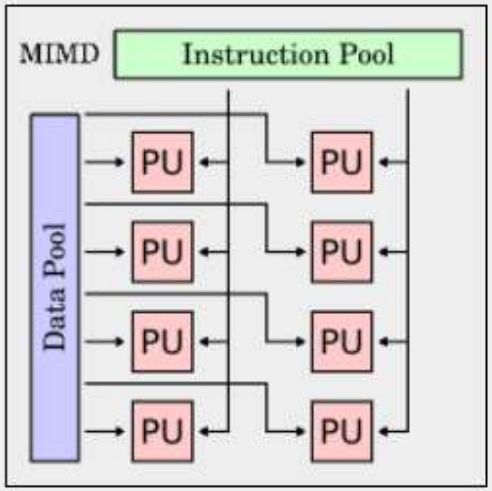
- A type of parallel computer
- **Multiple Instruction:** Each processing unit operates on the data independently via separate instruction streams.
- **Single Data:** A single data stream is fed into multiple processing units.
- Few (if any) actual examples of this class of parallel computer have ever existed.
- Some conceivable uses might be:
  - multiple frequency filters operating on a single signal stream
  - multiple cryptography algorithms attempting to crack a single coded message.

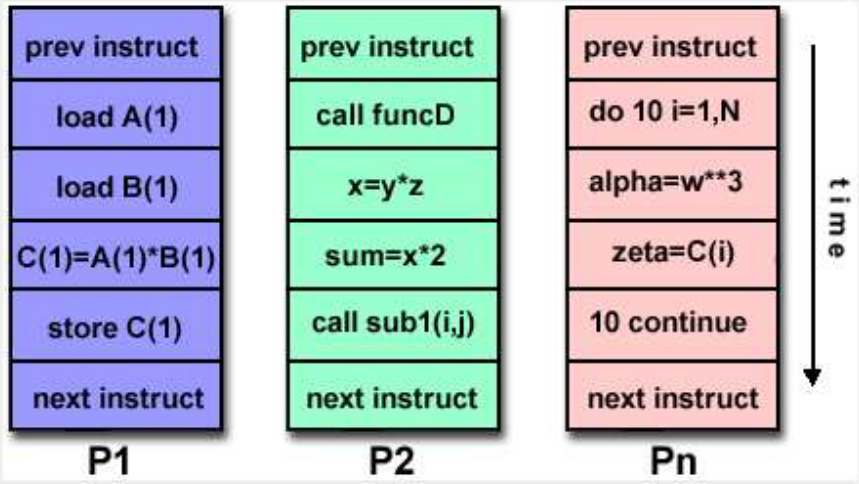




Multiple Instruction, Multiple Data (MIMD):

- A type of parallel computer
- **Multiple Instruction:** Every processor may be executing a different instruction stream
- **Multiple Data:** Every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- **Currently, the most common type of parallel computer - most modern supercomputers fall into this category.**
- Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.
- Note: many MIMD architectures also include SIMD execution sub-components





IBM POWER5

HP/Compaq Alphaserver

Intel IA32

AMD Opteron

Cray XT3

IBM BG/L

Some General Parallel Terminology-hardware

- **Supercomputing / High Performance Computing (HPC)**
  Using the world's fastest and largest computers to solve large problems.
- **Node**

  A standalone "computer in a box". Usually comprised of multiple CPUs/processors/cores. Nodes are networked together to comprise a supercomputer.
- **CPU / Socket / Processor / Core**

  This varies, depending upon who you talk to. In the past, a CPU (Central Processing Unit) was a singular execution component for a computer. Then, multiple CPUs were incorporated into a node. Then, individual CPUs were subdivided into multiple "cores", each being a unique execution unit. CPUs with multiple cores are sometimes called "sockets" - vendor dependent. The result is a node with multiple CPUs, each containing multiple cores. The nomenclature is confused at times. Wonder why?

Supercomputer - each blue light is a node

Node - standalone Von Neumann computer

CPU / Processor / Socket - each has multiple cores / processors.

- **Shared Memory** From a strictly hardware point of view, describes a computer architecture where all processors have direct (usually bus based) access to common physical memory. In a programming sense, it describes a model where parallel tasks all have the same "picture" of memory and can directly address and access the same logical memory locations regardless of where the physical memory actually exists.
- **Symmetric Multi-Processor (SMP)** Hardware architecture where multiple processors share a single address space and access to all resources; shared memory computing.
- **Distributed Memory** In hardware, refers to network based memory access for physical memory that is not common. As a programming model, tasks can only logically "see" local machine memory and must use communications to access memory on other machines where other tasks are executing.
- **Massively Parallel** Refers to the hardware that comprises a given parallel system - having many processors. The meaning of "many" keeps increasing, but currently, the largest parallel computers can be comprised of processors numbering in the hundreds of thousands.

Some General Parallel Terminology-tasks

- **Task** A logically discrete section of computational work. A task is typically a program or program-like set of instructions that is executed by a processor. A parallel program consists of multiple tasks running on multiple processors.
- The tasks of an application are coded in a parallel programming language or environment and are assigned to **processes** or **threads** which are then assigned to physical computation units for execution.
- The assignment of tasks to processes or threads is called **scheduling** and fixes the order in which the tasks are executed
- The assignment of processes or threads onto the physical units, processors or cores, is called **mapping** and is usually done by the runtime system but can sometimes be influenced by the programmer

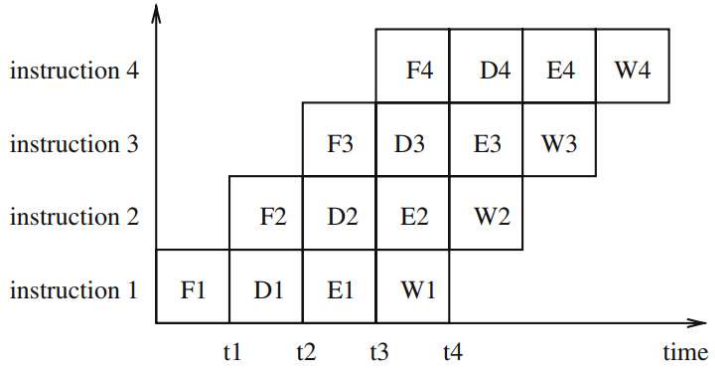
- **Load balancing** In general, it's clear that we want the processors or cores all to be assigned roughly the equal workload/tasks
- **Communications** Parallel tasks typically need to exchange data. There are several ways this can be accomplished, such as through a shared memory bus or over a network, however the actual event of data exchange is commonly referred to as communications regardless of the method employed.
- **Synchronization** The coordination of parallel tasks in real time, very often associated with communications. Often implemented by establishing a synchronization point within an application where a task may not proceed further until another task(s) reaches the same or logically equivalent point. Synchronization usually involves waiting by at least one task, and can therefore cause a parallel application's wall clock execution time to increase.

- **Granularity** In parallel computing can be defined as the size of tasks (e.g., in terms of the number of instructions)

  -***Coarse:*** relatively large amounts of computational work are done between communication events

  -***Fine:*** relatively small amounts of computational work are done between communication events

- **Embarrassingly Parallel** Solving many similar, but independent tasks simultaneously; little to no need for coordination between the tasks.
- **Pipelining** Breaking a task into steps performed by different processor units, with inputs streaming through, much like an assembly line; a type of parallel computing.



- **Parallel Overhead** The amount of time required to coordinate parallel tasks, as opposed to doing useful work. Parallel overhead can include factors such as:
  - Task start-up time
  - Synchronizations
  - Data communications
  - Software overhead imposed by parallel compilers, libraries, tools, operating system, etc.
  - Task termination time

Some General Parallel Terminology-Evaluation

- **parallel execution time**

  -The parallel execution time is the time elapsed between the start of the application on the first processor and the end of the execution of the application on all processors

  -it consists of the time for the computation on processors or cores and the time for data exchange or synchronization

  -The parallel execution time should be smaller than the sequential execution time on one processor so that designing a parallel program is worth the effort

- **speedup and efficiency**

  -They are used for a quantitative evaluation of the execution time of parallel programs , which compare the resulting parallel execution time with the sequential execution time on one processor

Speedup and Parallel Efficiency

- **Observed Speedup** Observed speedup of a code which has been parallelized, defined as:

$$\frac{\text{wall-clock time of serial execution}}{\text{wall-clock time of parallel execution}}$$

- **One of the simplest and most widely used indicators for a parallel program's performance.**

- **Parallel Efficiency = Speedup/# of processors**

Iso-efficiency

- The two-parameter efficiency function is defined as efficiency with two parameters *p* and *n*.

$$E(p,n) = \frac{S_p(n)}{p}$$

- The isoefficiency, which binds together the core count and the input size for a specific, **constant efficiency**

Suppose a parallel system exhibits efficiency $\varepsilon(n, p)$, where n denotes problem size and p denotes number of processors. Define $C = \varepsilon(n, p)/(1 - \varepsilon(n, p))$. Let $T(n, 1)$ denote sequential execution time, and let $T_o(n, p)$ denote parallel overhead (total amount of time spent by all processors performing communications and redundant computations). In order to maintain the same level of efficiency as the number of processors increases, problem size must be increased so that the following inequality is satisfied:

$$T(n, 1) \geq CT_o(n, p)$$

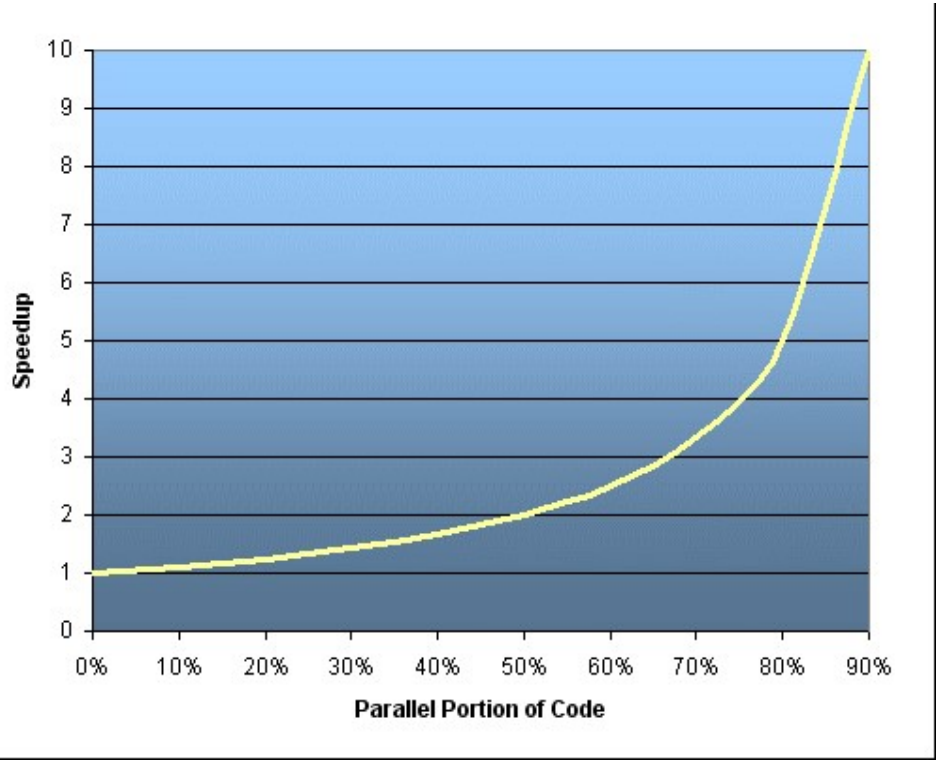
Scalability:

- **Scalability** Refers to a parallel system's (hardware and/or software) ability to demonstrate a proportionate increase in parallel speedup with the addition of more processors.
  - The ability of a parallel program's performance to scale is a result of a number of interrelated factors. Simply adding **more processors** is rarely the answer.
    - ✓ Hardware - particularly memory-cpu bandwidths and network communications
    - ✓ Application algorithm
    - ✓ Parallel overhead related
    - ✓ Characteristics of your specific application and coding
- Two types of scaling based on time to solution:
  - **Strong scaling**: The total problem size stays fixed as more processors are added.
  - **Weak scaling**: The problem size *per processor* stays fixed as more processors are added.

Scalability(II)

- The **algorithm** may have inherent limits to scalability. At some point, adding more resources causes performance to decrease. Many parallel solutions demonstrate this characteristic at some point.
- **Hardware** factors play a significant role in scalability. Examples:
  - Memory-cpu bus bandwidth on an SMP machine
  - Communications network bandwidth
  - Amount of memory available on any given machine or set of machines
  - Processor clock speed
- Parallel support **libraries** and subsystems software can limit scalability independent of your application.

Amdahl's Law



- **Amdahl's Law** states that potential program
  that can be parallelized:

$$\text{speedup} = \frac{1}{1 - P}$$

- If none of the code can be parallelized, P =
- If all of the code is parallelized, P = 1 a
- If 50% of the code can be parallelized, maximum speedup = 2, meaning the code will run twice as fast.
- Introducing the number of processors performing the parallel fraction of work, the relationship can be modeled by:

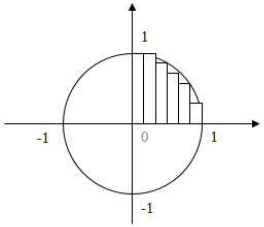
$$\text{speedup} = \frac{1}{\frac{P}{N} + S}$$

where P = parallel fraction, N = number of processors and S = serial fraction.

Parallel fraction

- 0<=P<1

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
double f(double x);/* Definition of f(x) */
{
    return(4.0/(1.0+x*x));
}
int main (int argc,char * argv[])
{
    int done =0,n,myid,numprocs,i;
    double PI25DT=3.14159265358979323846 2643;
    double mypi,pi,h,sum,x;
    double startwtime=0.0,endwtime;
    int namelen;
    char processor_nam
    MPI_Status status;
```

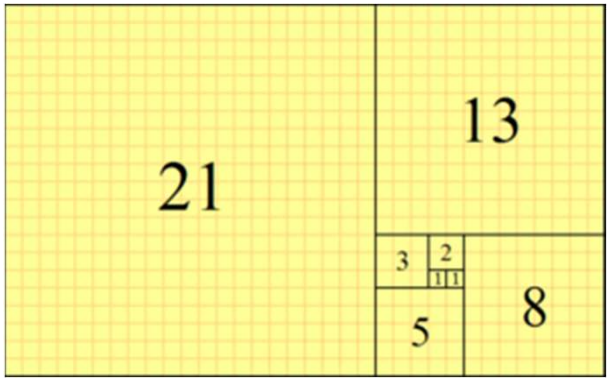


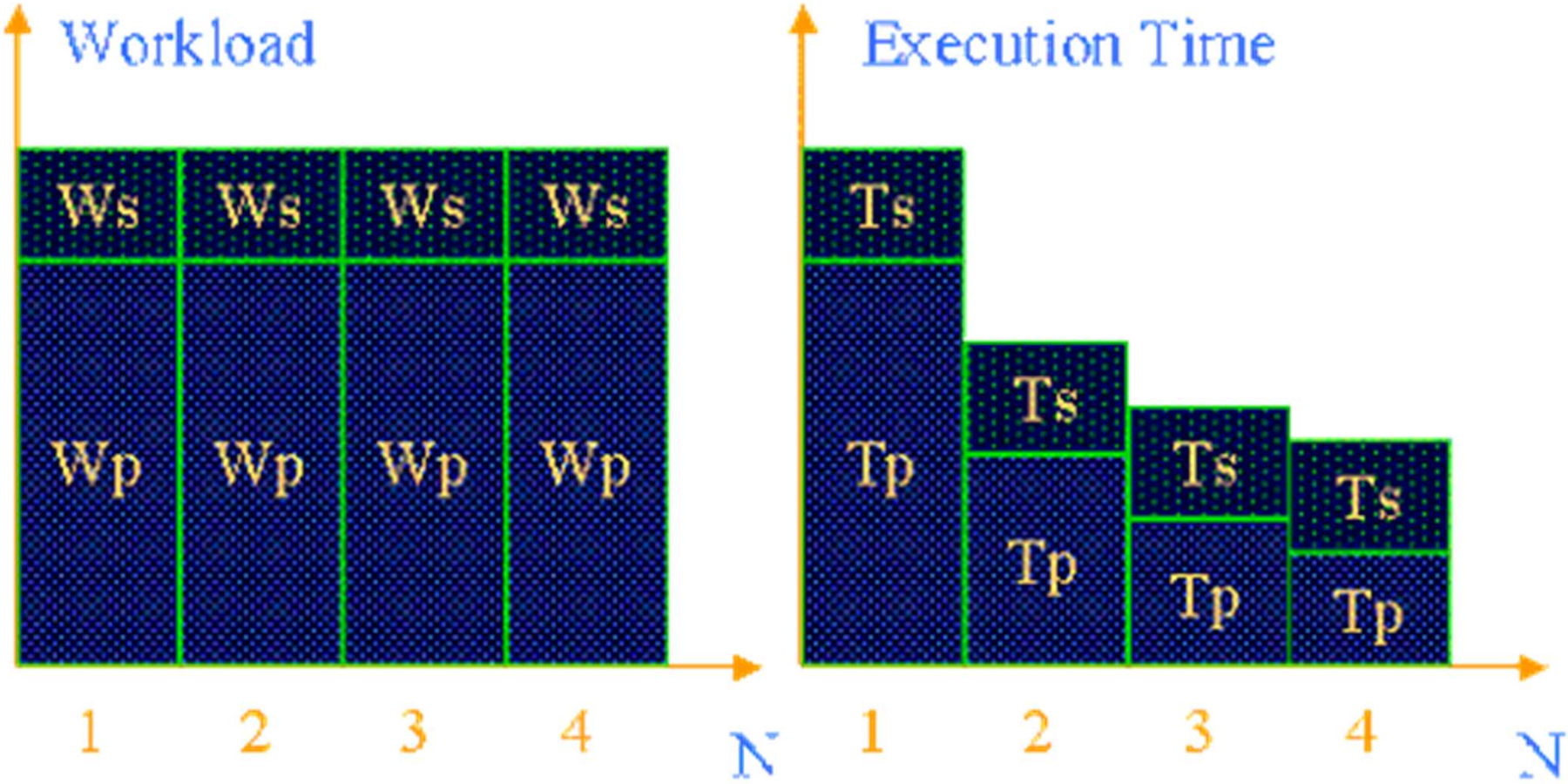

```
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
MPI_Get_processor_name(processor_name,&namelen);
fprint(stdout,"Process %d of %d on % s\n",myid,numprocs,
   processor_name);
n=0;
if (myid==0)
   {
      printf("Please give N=");
      scanf(&n);
      startwtime=MPI_Wtime();
      for (j=1;j<numprocs;j++)
     {
        MPI_Send(&n,1,MPI_INT,j,99,MPI_COMM_WORLD);
      }
   }
```

- Fibonacci number (P=0)

$$F_0 = 0, \quad F_1 = 1,$$

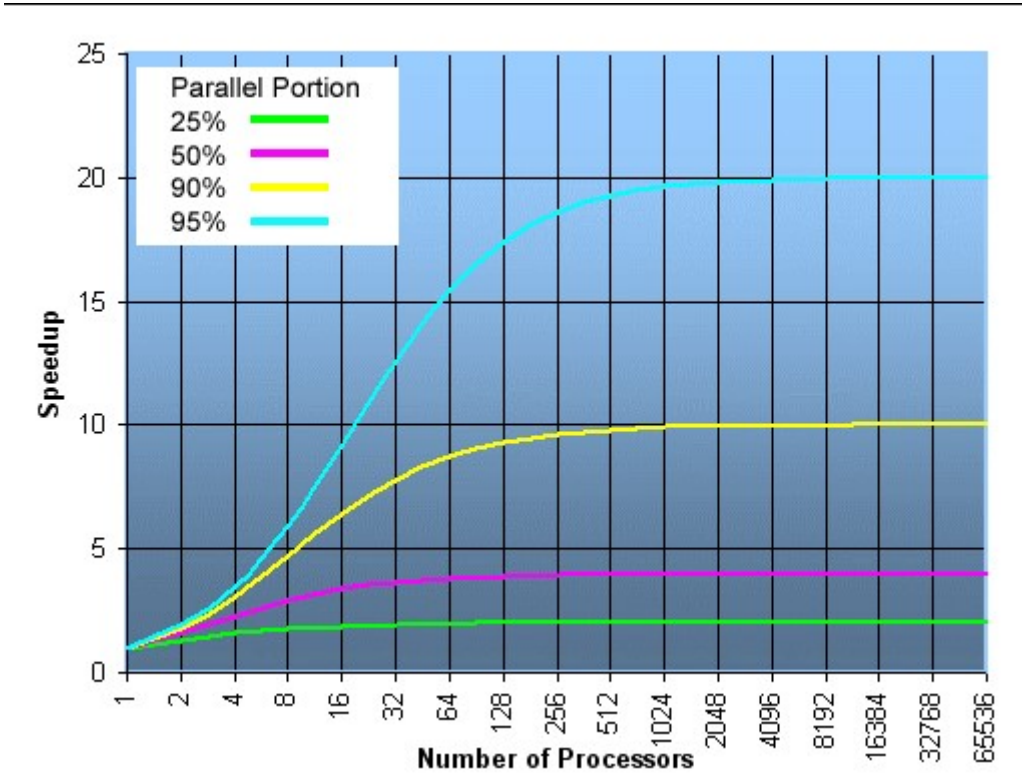
(for n >1) $$F_n = F_{n-1} + F_{n-2},$$




Workload
Ws
Ws
Ws
Ws
Wp
Wp
Wp
Wp
1
2
3
4
N
Execution Time
Ts
Ts
Ts
Ts
Tp
Tp
Tp
Tp
1
2
3
4
N

- It soon becomes obvious that there are limits to the scalability of parallelism. For example:

	speedup		
N	P = .50	P = .90	P = .99
10	1.82	5.26	9.17
100	1.98	9.17	50.25
1,000	1.99	9.91	90.99
10,000	1.99	9.91	99.02
100,000	1.99	9.99	99.90



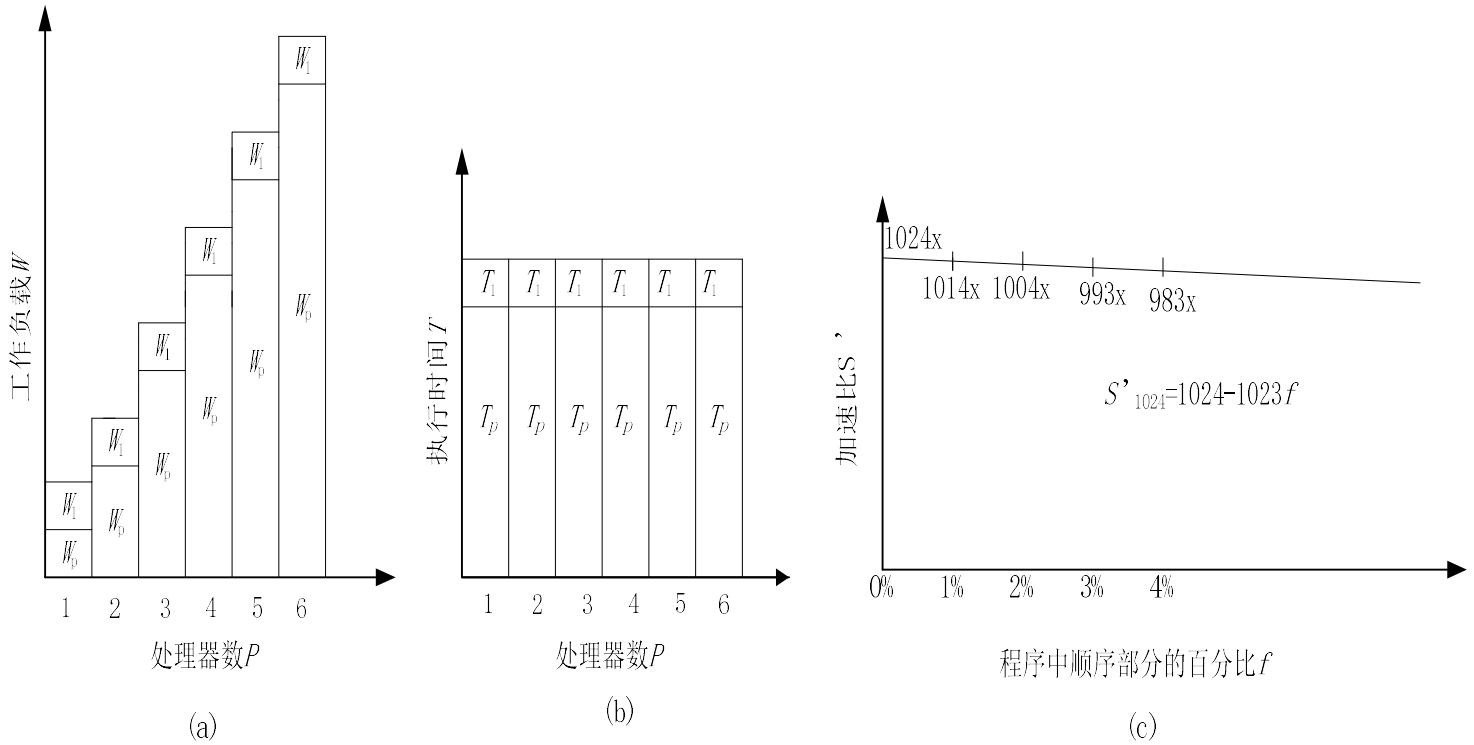
Gustafson's Law

$$S(N) = N - (1 - P)(N - 1)$$

where:

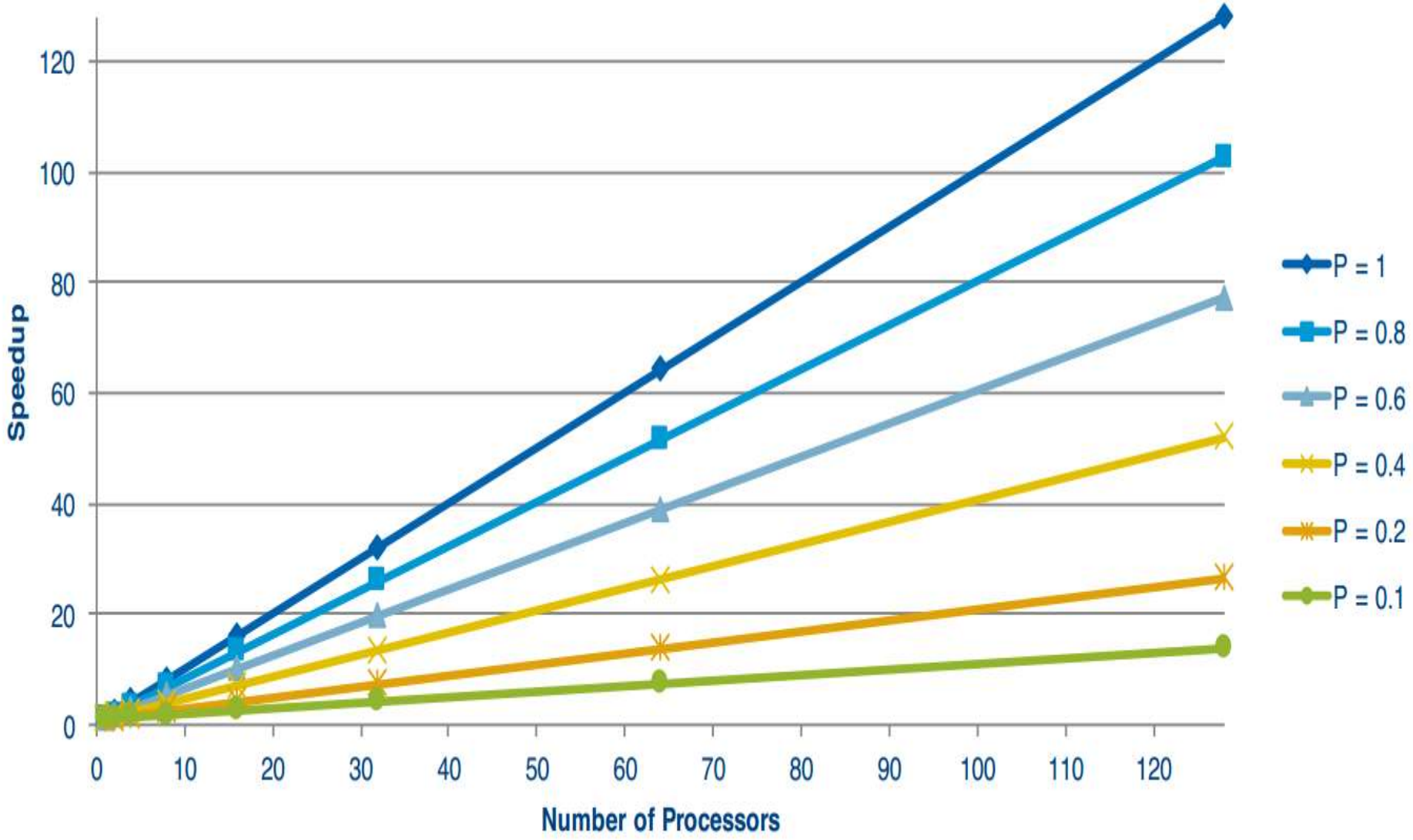
- $S(N)$ = speedup on N processors
- P = fraction of code that can be parallelised
- N = number of processors

- The speedup of "weak scaling" applications is governed by Gustafson's Law. As $N \to \infty, S(N) \to PN$



Weak scaling

Impact of Gustafson's Law



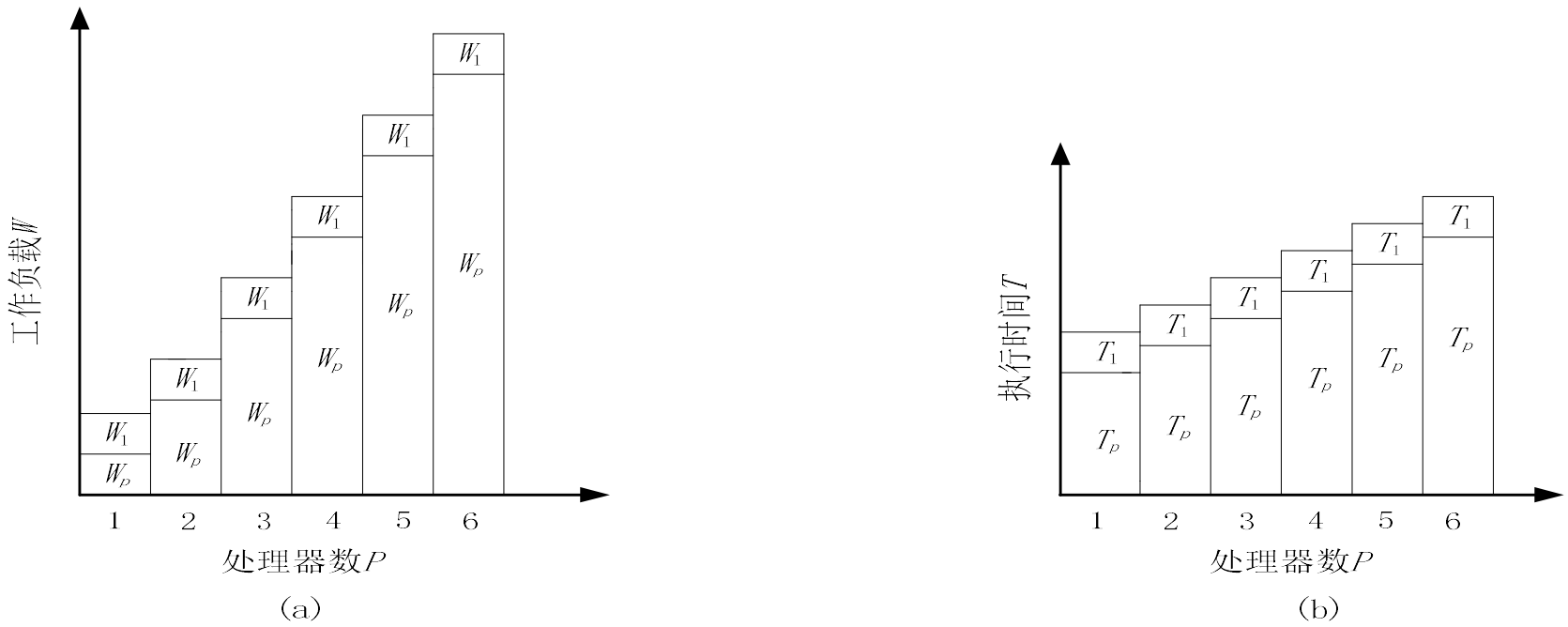
Sun-Ni law– Memory bounded speedup

- **Basic idea： As long as there is enough memory, we can increase the problem size to produce a better and more accurate solution.（Execution time may be increased.)**
  - **Single node, with full Memory, W-execution time，workload W= fW +（1-f）W。**
  - **P nodes can solve large scale of problem due to increased memory (pM). G（p）is defined as increased workload with pM.**

    W = fW +（1-f）G（p）W
- **Memory-bounded speedup**

$$S'' = \frac{fW + (1-f)G(p)W}{fW + (1-f)G(p)W/p} = \frac{f + (1-f)G(p)}{f + (1-f)G(p)/p}$$

Sun-Ni law(cont'd)



- G（p）=1 → Amdahl's law -- fixed-size speedup
- G（p）=p → f + p（1-f）→ Gustafson's law – fixed-time speedup
- G（p）>p，workload increases faster than memory, The speedup of Sun-N I is higher than Amdahl's and Gustafson's.

Some issues (1) - Complexity

- In general, parallel applications are much more complex than corresponding serial applications, perhaps an order of magnitude. Not only do you have multiple instruction streams executing at the same time, but you also have data flowing between them.
- The costs of complexity are measured in programmer time in virtually every aspect of the software development cycle:
  - Design
  - Coding
  - Debugging
  - Tuning
  - Maintenance
- Adhering to "good" software development practices is essential when working with parallel applications - especially if somebody besides you will have to work with the software.

Some issues (2) - Portability

- Thanks to standardization in several APIs, such as MPI, POSIX threads, and OpenMP, portability issues with parallel programs are not as serious as in years past. However...
- All of the usual portability issues associated with serial programs apply to parallel programs. For example, if you use vendor "enhancements" to Fortran, C or C++, portability will be a problem.
- Even though standards exist for several APIs, implementations will differ in a number of details, sometimes to the point of requiring code modifications in order to effect portability.
- Operating systems can play a key role in code portability issues.
- Hardware architectures are characteristically highly variable and can affect portability.

Some issues (3) - Resource Requirements

- The primary intent of parallel programming is to decrease execution wall clock time, however in order to accomplish this, more CPU time is required. For example, a parallel code that runs in 1 hour on 8 processors actually uses 8 hours of CPU time.
- The amount of memory required can be greater for parallel codes than serial codes, due to the need to replicate data and for overheads associated with parallel support libraries and subsystems.
- For short running parallel programs, there can actually be a decrease in performance compared to a similar serial implementation. The overhead costs associated with setting up the parallel environment, task creation, communications and task termination can comprise a significant portion of the total execution time for short runs.

Thank you!