Introduction to large-scale computing
大規模計算序論

Institute for NanoScience Design, Osaka University
大阪大学ナノサイエンスデザイン教育研究センター
Masaaki Geshi
下司　雅章

36th CMD workshop, supercomputer course
2020, February 17th (Mon.)

Contents

- Necessity of large-scale computing
- Methods of the large-scale computing(Mainly parallelization)
- Today's supercomputers
- Future direction of large-scale computing

Necessity of large-scale computing

- Systems we want to study have many atoms.
- The computational cost increases N^3, N^6, or exponentially.(N is the number of atoms, electrons,…)
- Some physical quantities needs a huge computational cost in spite of a small system(high accuracy).
- There is scientific and/or industrial significance obtained by large-scale computing.

Problems raised by large-scale computing

- Because of huge date must be handled, huge HDD and fast IO is needed. (~30PB in K computer, ~150PB in Fugaku(post-K))
- The problem of data analysis including visualization.
- In some cases, data analysis costs more time than computational time.
- …

When we perform large-scale computing,…

- It costs huge computational time.
- In some cases, huge physical memory of CPU is needed.

- In order to solve these problems, we have to improve software codes to reduce the computational time and memory.

- Parallel computing is definitely necessary.

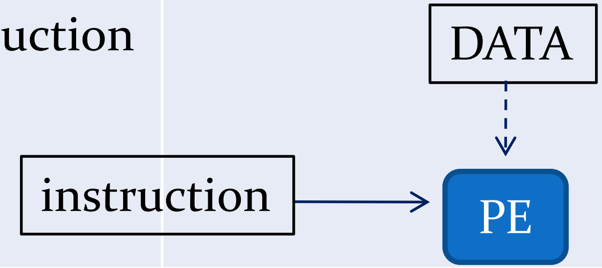
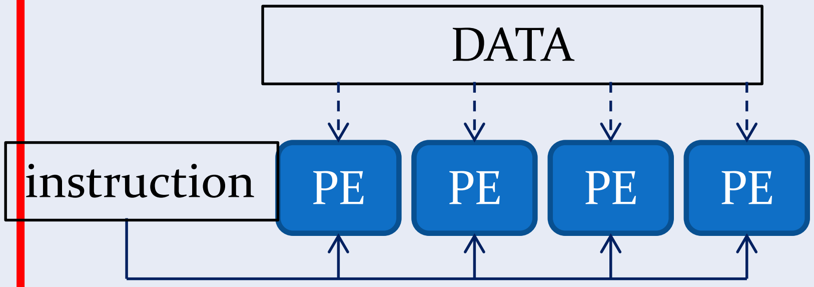
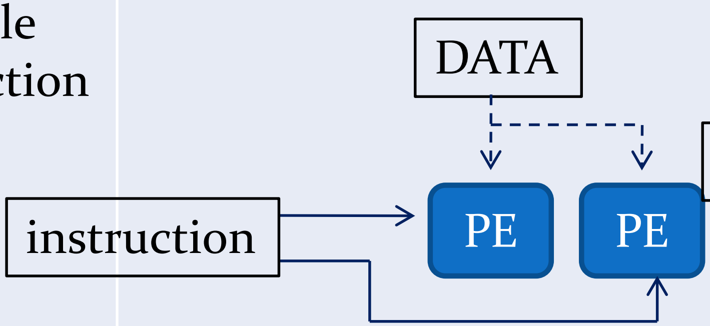
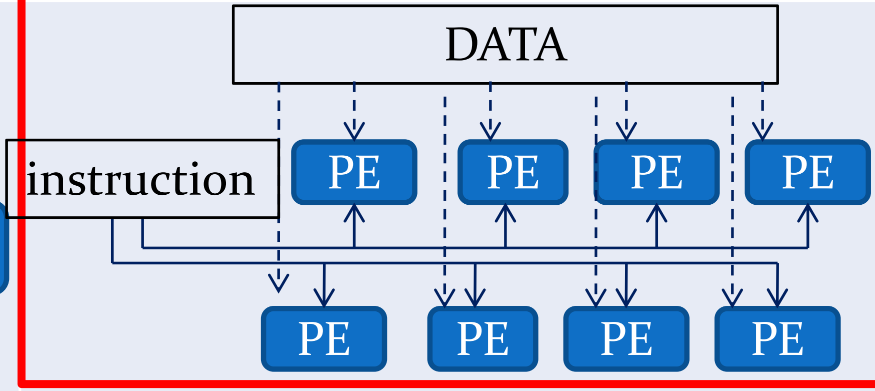
Methods for large-scale computing

- Speed-up techniques
  - Vectorization→SIMD(many cores)
  - Parallelization
  - Order-*N* method(reduce the cost o(N^p)→o(N)(p=1))
  - Use of specialized machine or accelerator(GPU).
- These are used in a code depending on one's necessity.

The easiest way to speed up is to be able to realize it by choosing proper compile options(vectorization, Automatic parallelization, optimization,…).

Since it is not possible to speed up the code automatically, we have to tune the code significantly.

Concept of Flynn’s taxonomy

	Single data	Multiple data
	SISD	SIMD
Single instruction		
	MISD	MIMD
Multiple instruction		 DATA instruction PE PE PE PE PE PE PE PE

Vectorization

- A vector processor, or array processor, is a CPU that implements an instruction set containing instructions that operate on one-dimensional arrays of data called *vectors*(row, column, or diagonal elements, etc…).
- Vectorization is the method that we program a code to construct regularly-arrayed data, and process the data at a time by the vector CPU.
- Vectorization realizes the reduction of a CPU time.
- Today, this concept is included in SIMD.

Vector processing

```
do i=1,imax
   a(i) = c(i) ＋d(i)
   b(i) = c(i) ×d(i)
enddo
```

We tune to increase the part being processed at the deepest do-loop in multi do-loop.

Scalar processing

```
a(1) = c(1) ＋d(1)
b(1) = c(1) ×d(1)
a(2) = c(2) ＋d(2)
b(2) = c(2) ×d(2)
:
:
a(imax) = c(imax) ＋d(imax)
b(imax) = c(imax) ×d(imax)
```

Vector processing

```
a(1) = c(1) ＋d(1)
a(2) = c(2) ＋d(2)
:
:
a(imax) = c(imax) ＋d(imax)
```

“a” array

```
b(1) = c(1) ×d(1)
b(2) = c(2) ×d(2)
:
:
b(imax) = c(imax) ×d(imax)
```

“b” array

Order-*N*

- Realized by algorithm

→No approximation Ex.: Screened-KKR

- Realized by approximation of Hamiltonian

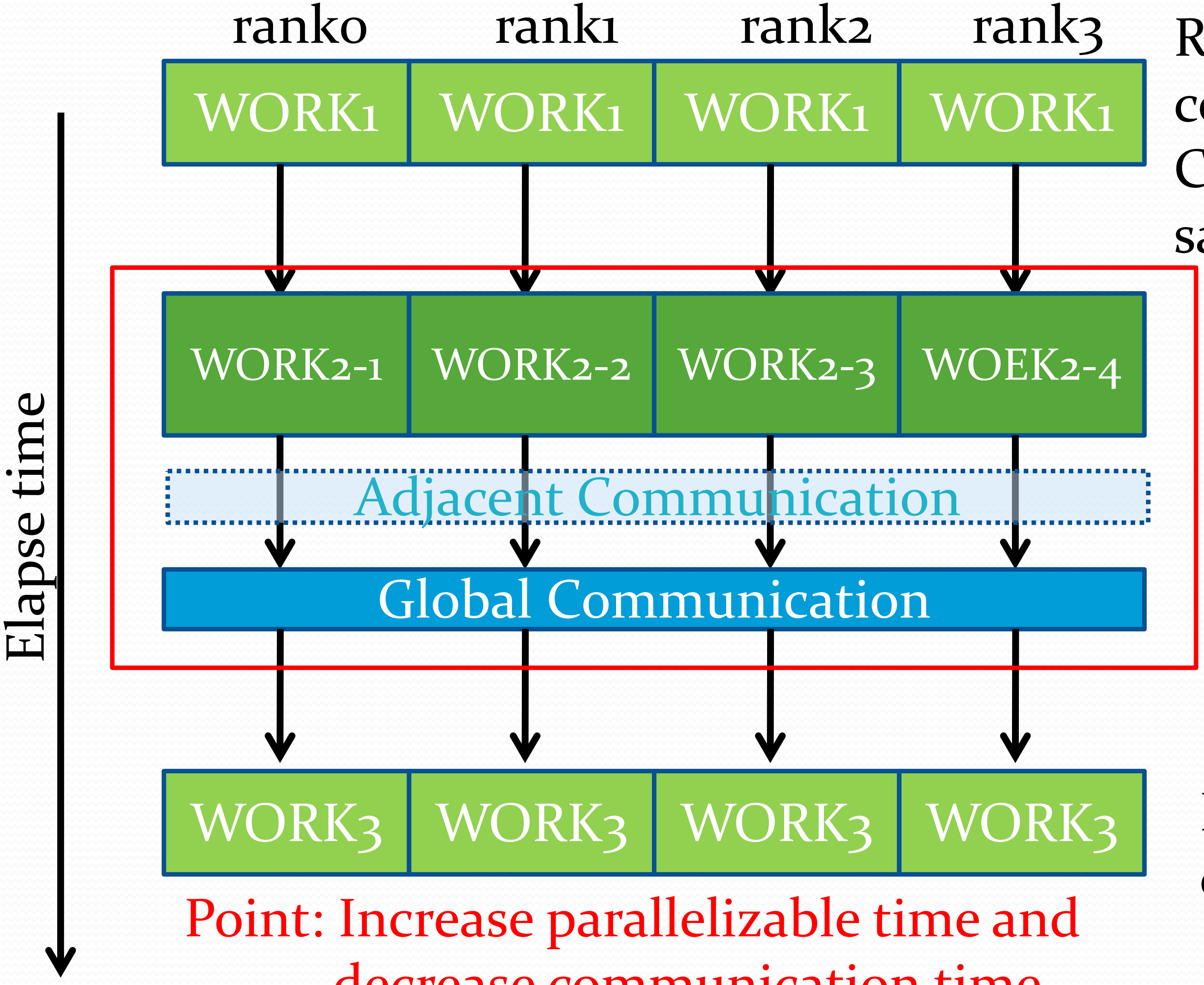
→Approximation by localization of interaction. This type of methods were actively studied in 90's.

  - Roughly speaking, matrices are block-diagonalized or including similar method.
  - $N \times N \rightarrow N \times N_{\mathrm{local}}$(fixed). If N increases, N_{local} not increase. (like a tight-binding approximation)
  - The accuracy of the approximation can be adjustable depending on a required accuracy.
  - One of the famous method is a Divide and conquer method（分割統治法）. This is used in "OpenMX(Ozaki)", "DC(Kobayashi and Nakai)",CONQUEST(Bowler, Miyazaki, Gillan)…

Parallelization

- This is the method to shorten the elapse time by sharing task with more than one CPU without changing the processing time of the task.
- This is used to share big memory(array).
- SIMD(Single instruction, multiple data), MIMD(Multiple instruction, multiple data)

Concept of parallelization



Redundant computation (all CPUs compute the same calculation)

Parallel computation (different CPU computes different calculation)

Redundant computation

Point: Increase parallelizable time and decrease communication time.

Problems in parallelization

- The ratio of parallelizable part in an elapse time on 1CPU sequential job is conclusive (Amdahl' law) .
- When we parallelize a software code, the data communication between nodes must occur. If the number of nodes increases, the communication time between nodes costs more than the computational time.(In the case of classical MD simulation, this is a crucial problem.)

Amdahl’s Law

Speed up ratio $\alpha_P \equiv \dfrac{t_1}{t_{N_\text{core}}}$

Elapse time of one job on 1 core

Elapse time of one job on N core

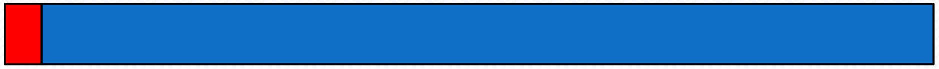
$$(\alpha_P)_\text{max} = t_1 \times \left(\frac{1}{t_{N_\text{core}}} \right)_\text{max} = \frac{1}{(1-P) + \dfrac{P}{N_\text{core}}}$$

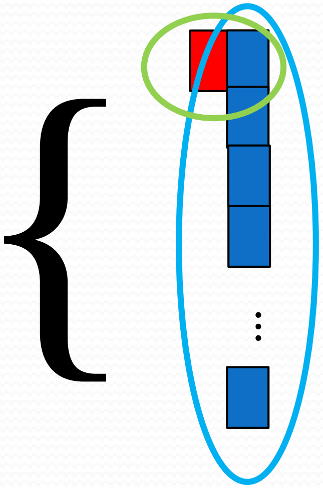
P: parallelizable time in a sequential job

Upper bound of parallelizable ratio

For examples,

Not parallelizable (1 sec.) parallelizable (99 sec.)





1+0.99=1.99 sec.(almost 50 times faster)

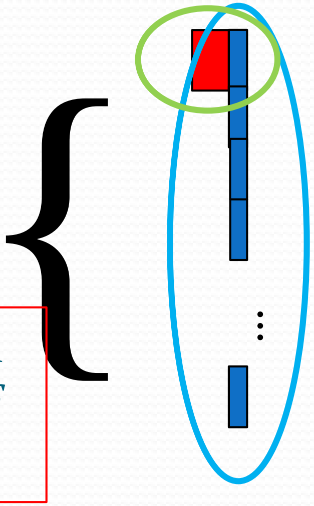
Even w

obtain

We have to increase the parallelizable part as possible.

We have to decrease the non parallelizable part as possible.

100 processors



1+0.099=1.099 sec. (almost 91 times faster)

Although we uses 1000 processors, the efficiency does not become 100 times even!!!

1000 processors

We feel that the merit of parallelization does not increase even if the number of processors increases more and more.

Gustafson-Barsis' law

Speed-up ratio $\leq n + (1 - n)s$

n: the number of cores

s: the ratio of sequential computation(this part cannot parallelize.)

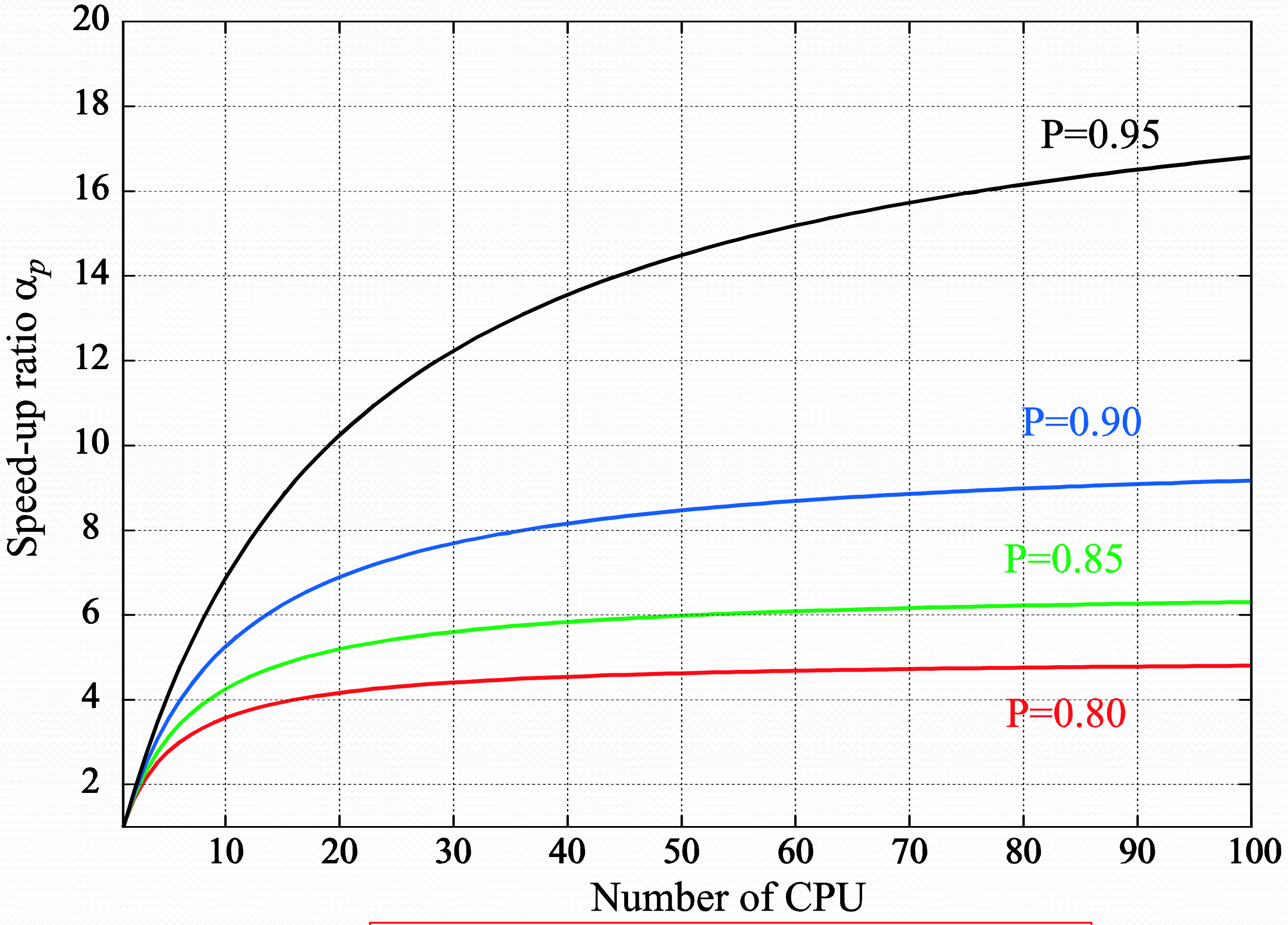
In this law, the communication time among nodes does not considered.

Indices of parallelization efficiency

- Strong scaling－The overall problem size (number of atoms or electrons etc…) is fixed and the number of processors is increased(based on Amdahl's law).
- Weak scaling－The problem size per processor is fixed and the overall problem size is increased with increasing processors(based on Gustafson's law). The processing of one processor is not changed. The ideal situation is the computation time should be constant even if the increase of the number of processors. If the computation time increases, we can see the non parallel part remains, and if the adjacent communication time increases, the method of the communication has a problem.

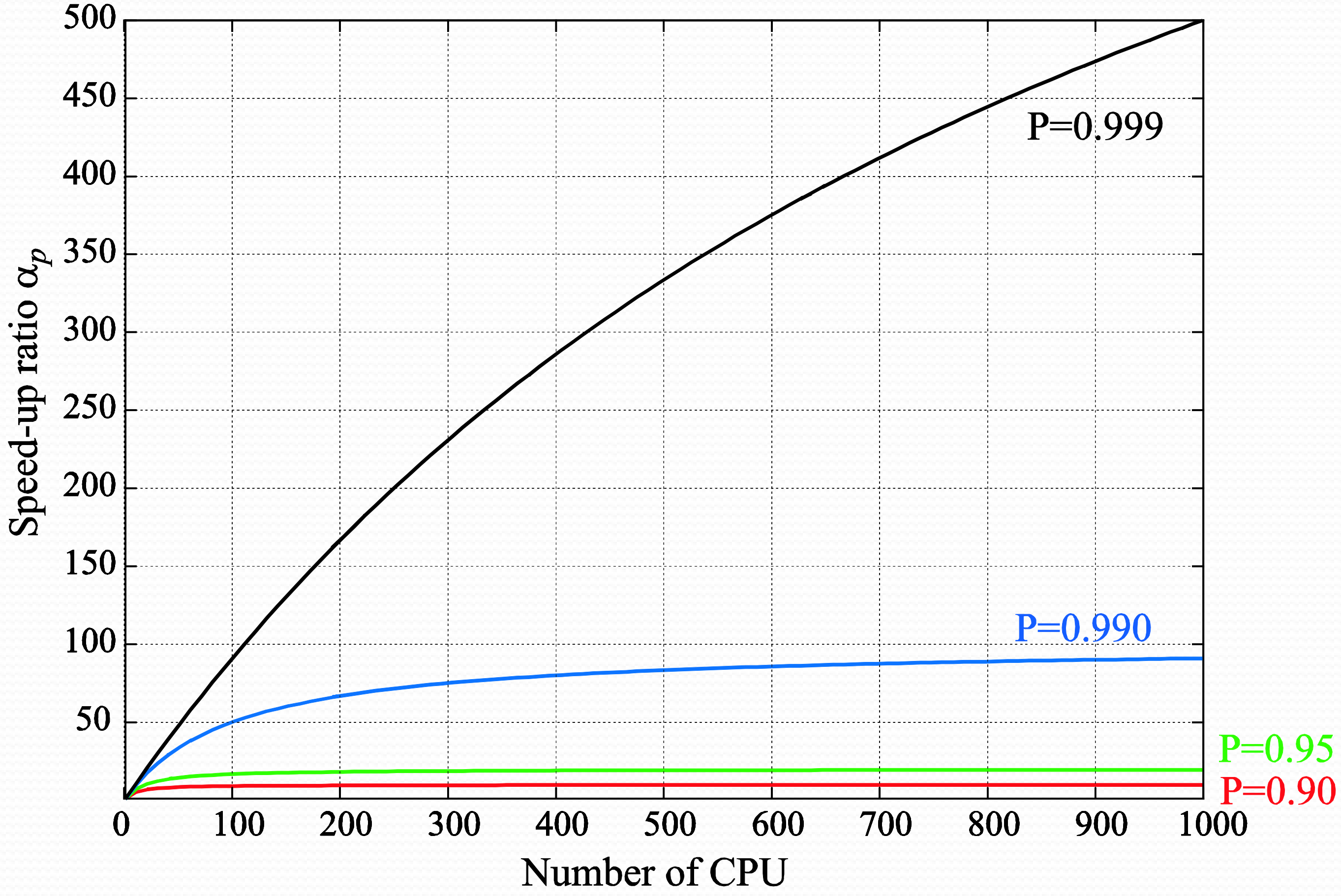
Reference: http://www.r-ccs.riken.jp/library/event/tokuronB_180406.html

Example of Amdahl's law(1)



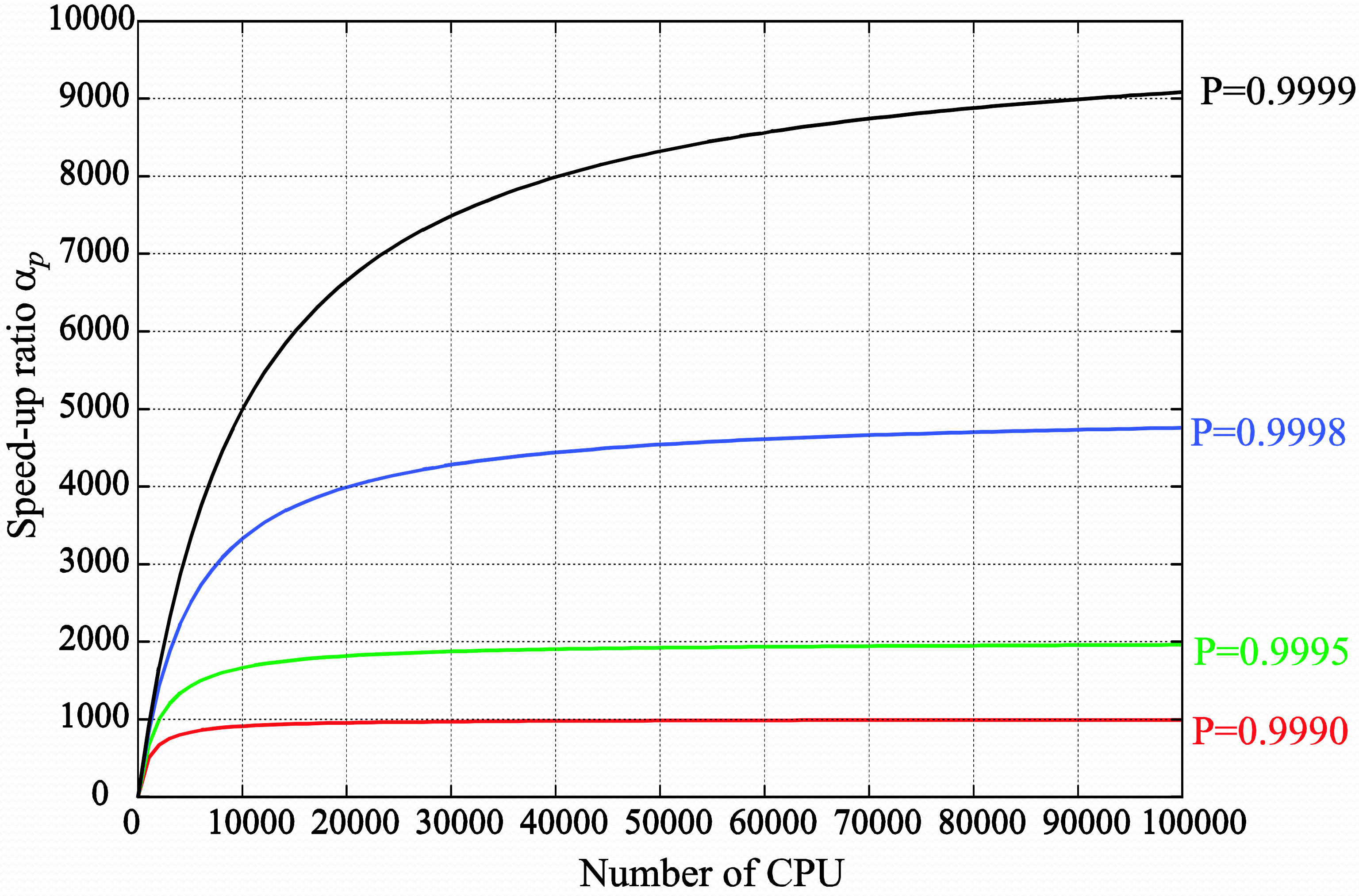
Ordinary calculations using PC cluster

Example of Amdahl's law(2)



Calculations using high-end PC cluster or supercomputers

Example of Amdahl's law(3)

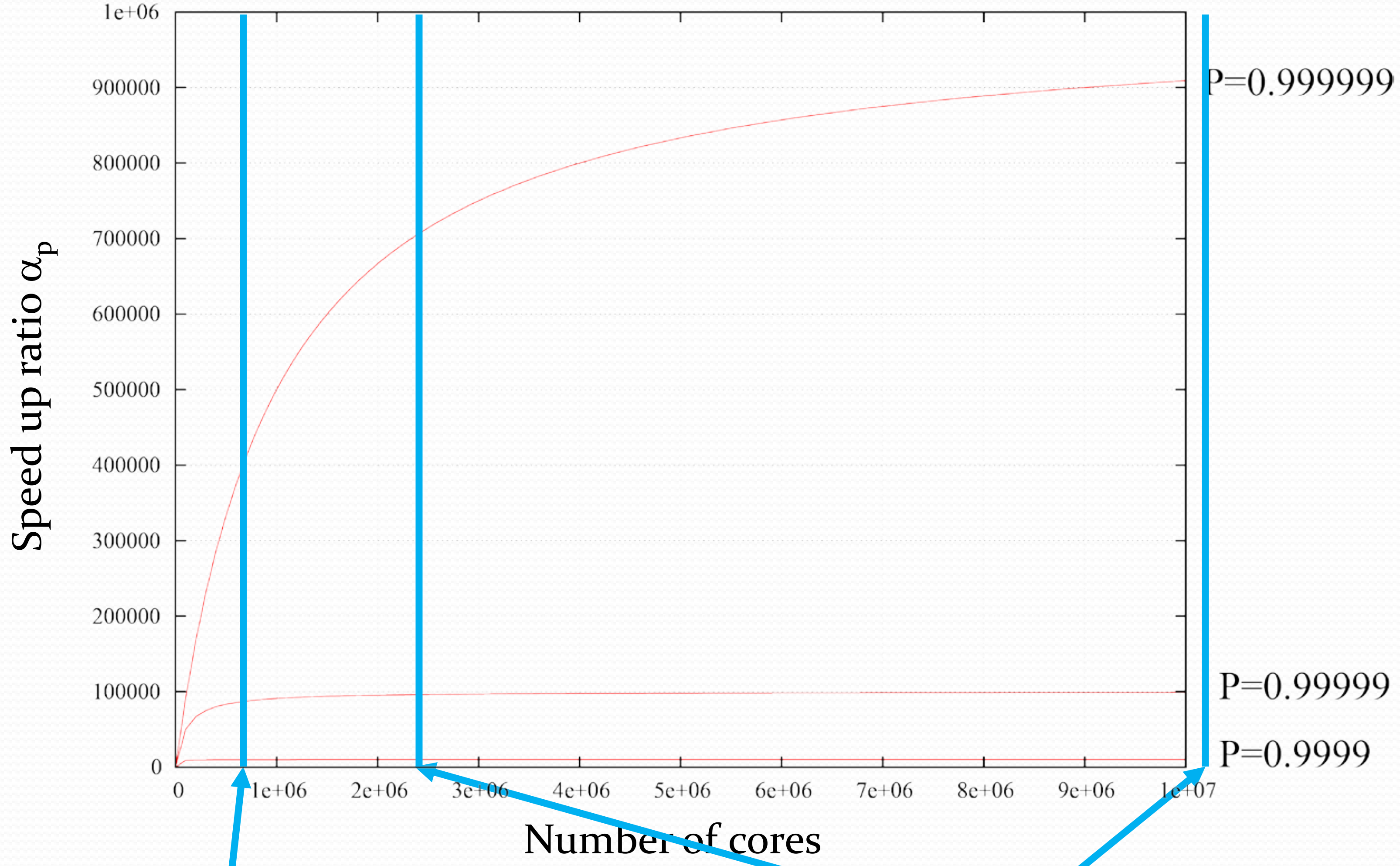


K computer: 705,024 cores

Calculations using high-end supercomputers

Example of Amdahl’s law(4)

Calculations using top class supercomputers in the world



K computer: 705,024 cores

Summit: 2,397,824 cores

Sunway TaihuLight: 10,649,600 cores

Even if you are just a user,

- you need to be aware of the parallelization efficiency of the software you use because you must effectively use machine time(machine point, or cost(usage fee)) you can use.
- you need to understand the characteristics of the computer and make efforts to increase the calculation efficiency.
- you need to choose the best compile options to make executable files.

For Intel compiler, -xHost, -axCORE-AVX2,...

- you need to choose the best parameters for the best performance of the software to obtain the best scientific results.

In DFT calculations, E_{cut}, k-point sampling,...
Fundamentally a model size,...

Parallel programming languages

- MPI(message passing interface)
  - The most versatile method.
  - Both core-to-core and node-to-node communication can be handled.
  - The programming may not be easy.
- OpenMP
  - Only core-to-core communication.
  - The programming may be easier than that of MPI.

(within non-complex case)

- PVM
- HPF(High Performance Fortran)

} Now, these are seldom used.

Example of parallelization with MPI

```
program main
include (mpif.h)
        …
call MPI_INIT(IERR)
call MPI_COMM_SIZE(MPI_COMM_WORLD,NPROCS,IERR)
call MPI_COMM_RANK(MPI_COMM_WORLD,MYRANK,IERR)
        …
call para_range(1, n, nprocs, myrank, ista, iend)
do i=ista, iend
        …
   a(i)=a(i)+value
enddo
 call MPI_ALLREDUCE(a,a1,n_element,
     & MPI_DOUBLE_PRECISION,
     & MPI_SUM, MPI_COMM_WORLD, IERR)
     …
call MPI_FINALIZE(IERR)
     …
end
```

ista, iend are the first array numbers allocated to each processor.

Each processor has only allocated elements of the array a() .

Sum up all elements of the array a() for all processors and distributed it for all processors (all processors have the same value of a()).

It is necessary to rewrite so as to calculate only the part allocated to each node and to communicate the data.

Execution: mpirun (or mpiexec) –np 4(# of parallel) ./a.out

Example of parallelization with OpenMP

```
program main
implicit none
integer omp_get_thread_num,I
double precision z(100), a, x(100), y
 do i= 1, 100
    z(i) = 0.0
    x(i) = 2.0
  end do
 a = 4.0
 y = 1.0
print *, “Welcome to the parallel world”
!$omp parallel
print *, “Excuted on”, omp_get_thread_num()
 call daxpy(z, a, x, y)
!$omp end parallel
end program main

subroutine daxpy(z, a, x, y)
integer I
double precision z(100), a, x(100), y
!$omp parallel do
 do i = 1, 100
    z(i) = a * x (i) + y
 enddo
 return
end
```

For simple parallelization, we only need to insert directives, so if we compile without parallelization options the directives are commented out, so basically we do not need to change the original code.

Directive of the start of parallel processing

Directive of the end of parallel processing

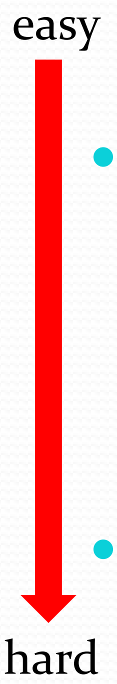
Do parallel of just below do loop

Execution: We do not use a special command.
Set environmental value
“OMP_NUM_THREADS=4” (# of parallel)

The difference between MPI and OpenMP

- OpenMP can be parallelizable only for cores which share the memory. We cannot make a large-scale parallelization code by using OpenMP only.
- The typical software codes which correspond to the large-scale parallelization are tuned by using MPI for its main part.
- The role of OpenMP is to assist the speed up of the code which is parallelized by using MPI.
- Even if you do not develop the software code, you have to know the details of the code in order to get maximum performance.

The other environments to develop parallel computing codes using GPU



- CUFFT, CUBLAS,…
  - Only call libraries to accelerate specific parts. The other parts do not be accelerated.
- OpenACC
  - a programming standard for parallel computing developed by Cray, CAPS, NVIDIA and PGI, designed to simplify parallel programming of heterogenious CPU/GPU system.
  - Very similar with OpenMP. In near future, OpenMP and OpenACC may be merged.
  - Fortran and C are supported.
- CUDA
  - a parallel computing platform and programming model created by NVIDIA and implemented by the graphics processing units (GPUs) that they produce. (only PGI Fortran can used.))
- OpenCL
  - a framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), DSPs and other processors.

OpenACC(Open Accelerator)

```
program picalc
    implicit none
    integer, parameter :: n=1000000
    integer :: i
    real(kind=8) :: t, pi
    pi = 0.0
  !$acc parallel loop
    do i=0, n-1
      t = (i+0.5)/n
      pi = pi + 4.0/(1.0 + t*t)
    end do
  !$acc end parallel loop
    print *, 'pi=', pi/n
  end program picalc
```

As the development environment, the best results can be obtained by improving according to the message issued by the compiler.

http://www.nvidia.co.jp/object/openacc-gpu-directives-jp.html
http://www.cms-initiative.jp/ja/events/2014-haishin
http://www.r-ccs.riken.jp/library/event/tokuronB_180406.html
(The 14th lecture given by Dr. A. Naruse is useful.)

The cutting-edge techniques

- Avoid memory wall problem(cache control)
  - Computing power>data transfer from memory to computing unit(CPU)→reuse the data on caches
- Pipeline processing
- Continuous access in do loop
- loop unrolling
- Divide data into blocks
- Use the highly-optimized libraries
- …

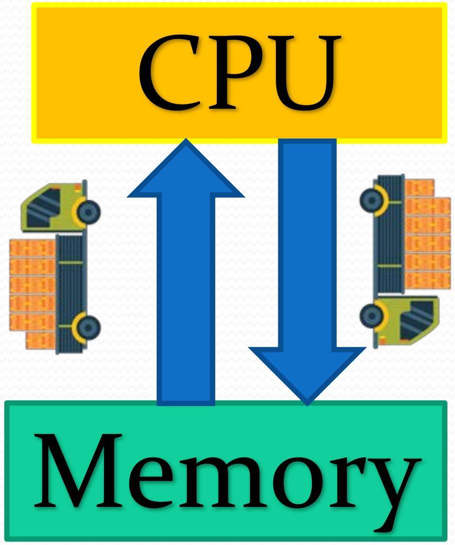
We have to know details of hardware to make highly optimized software codes.

Please see these sites; (only Japanese)

http://www.r-ccs.riken.jp/library/event/tokurona_170406.html
http://www.r-ccs.riken.jp/library/event/tokuronB_180406.html

Memory wall problem

Old computer



CPU

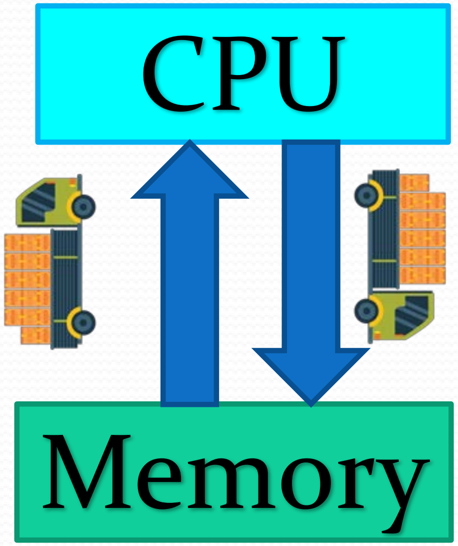
Memory

Computation performance (number of clocks) was slow (~10MHz)

The memory can not be accessed until a certain time passes

Sending data from memory to the CPU was slow, and the computation speed was also slow, so it was not the problem at “data transfer speed ≈ calculated speed”.

Today’s computer



CPU

Memory

The computation performance becomes extremely fast (~a few GHz)

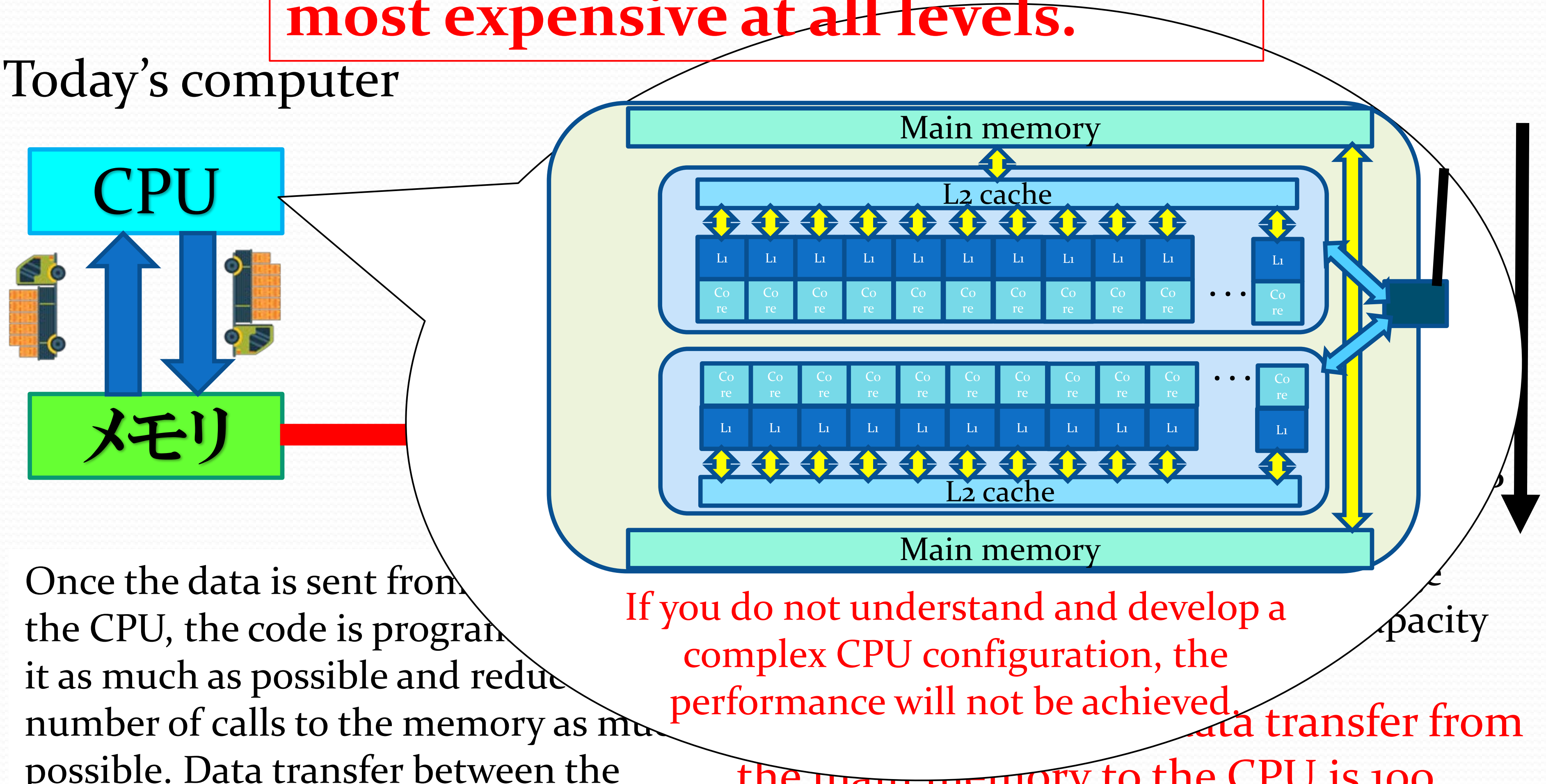
The fact that memory cannot be accessed only for a fixed time does not change much.

Sending data from the memory to the CPU is slow, but since the computing speed has become extremely fast, the CPU is waiting for a long time because of “the transfer speed << compute speed” (supplement with cache etc…).

Memory wall problem

Important!: Data transfer is the most expensive at all levels.

Today's computer



If you do not understand and develop a complex CPU configuration, the performance will not be achieved.

Once the data is sent from the CPU, the code is program it as much as possible and reduc number of calls to the memory as mu possible. Data transfer between the memory and the register or cache is fast, but the capacity is small.

e pacity

ata transfer from the main memory to the CPU is 100 times longer than access to the data on the register.

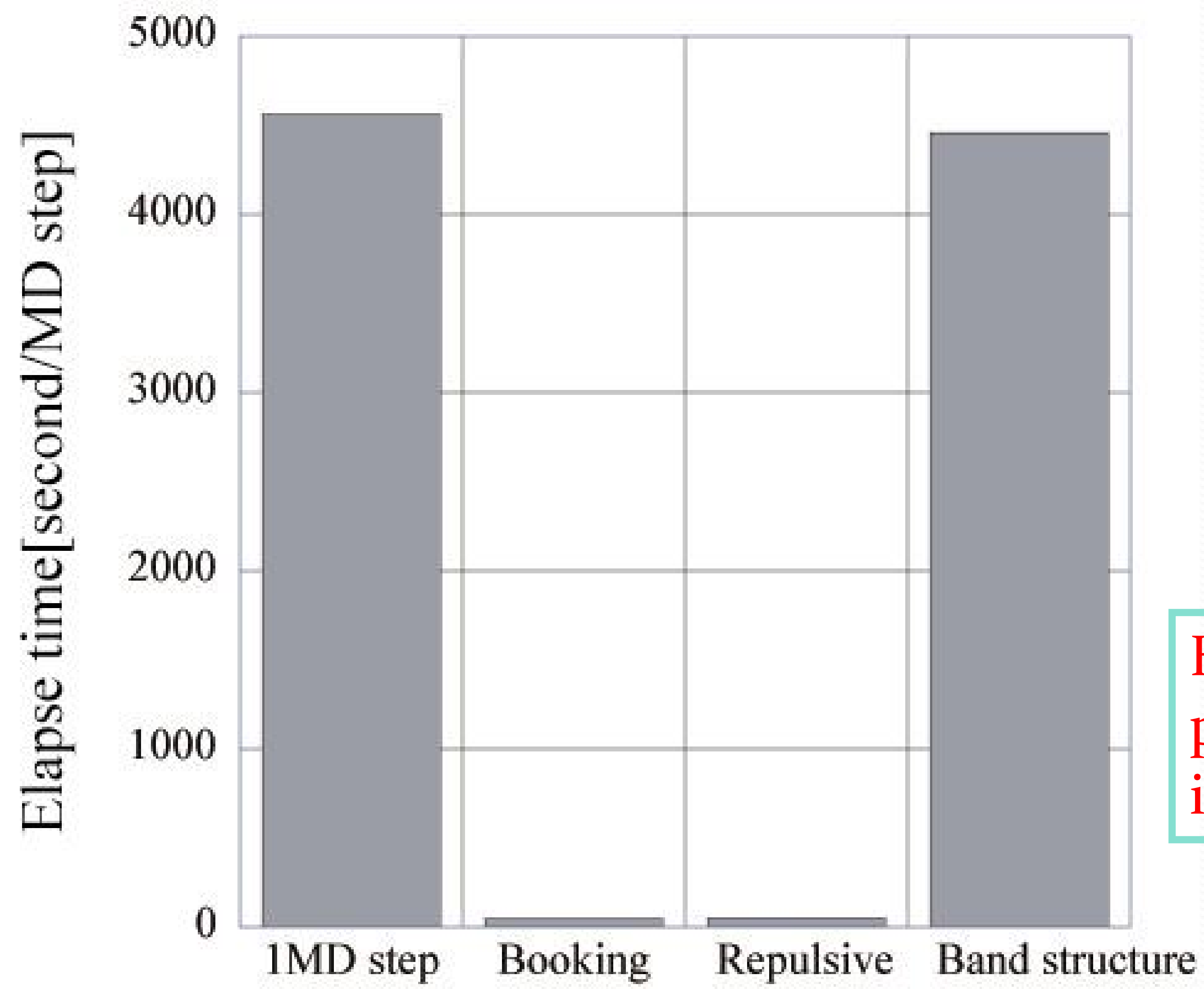
High performance Python

- In place of Fortran and C / C ++, which have been used in scientific computing in the past, HPC software development has been advanced in the USA by using Python.
- Python has overwhelming code readability compared to Fortran and C / C ++.

→It is more likely that the maintenance of the code will be continued without reading by experts.

- We do not need to write everything in Python. If we know that another language is faster than Python for a certain part, we should use the part in the original. We combine Python and the other language and improve the ease of maintenance. (It is said that Python is a "glue language" because it is a language that "sticks" programs in multiple languages.)

Analysis of the computational time in order-*N* tight-binding method



A electronic structure part occupies 97.6% of total elapse time in a sequential job.

↓

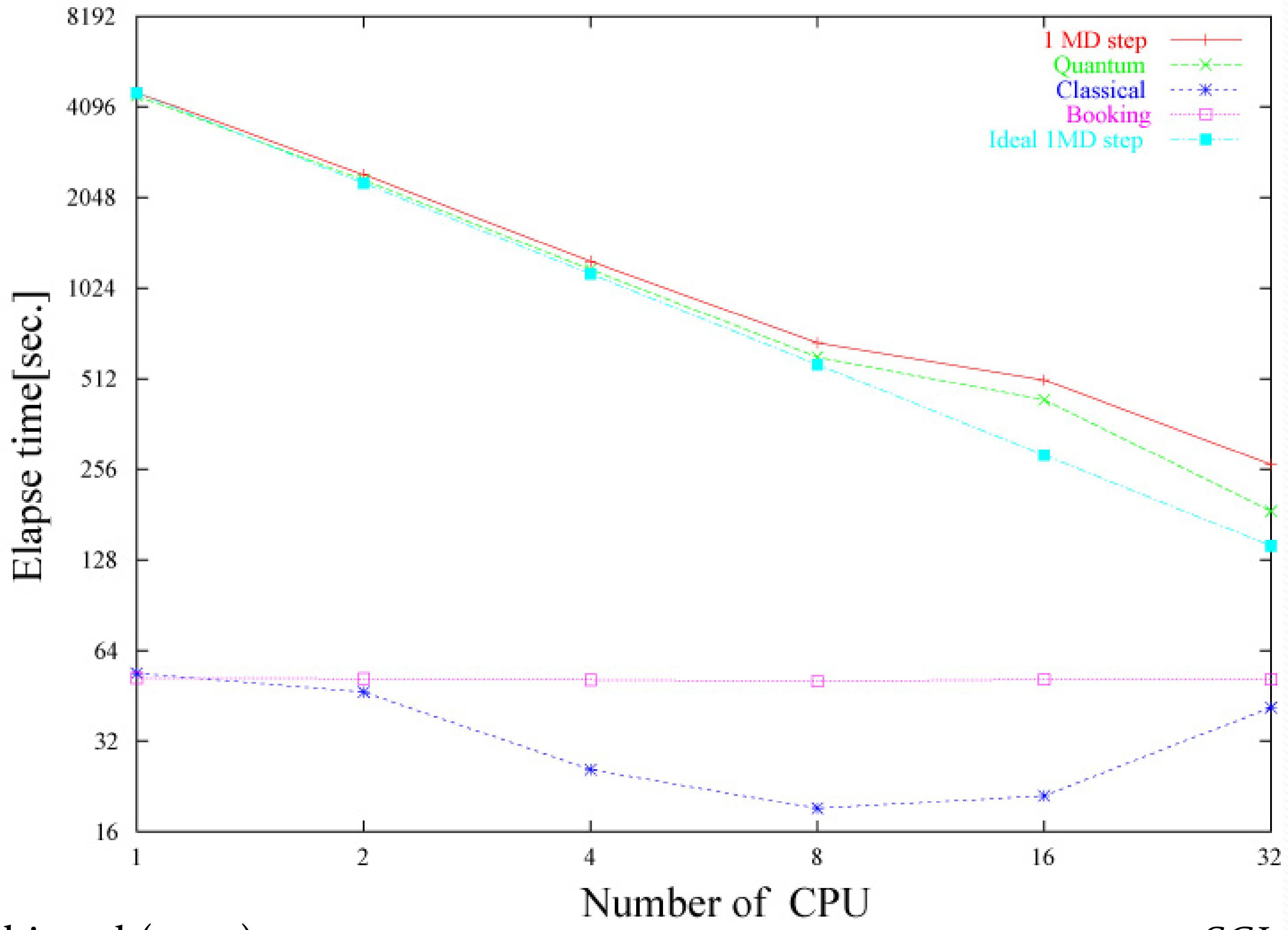
High parallelizability is expected.

Geshi et al.(2003)

Parallelization by means of MPI

- We used MPI_ALLREDUCE and MPI_BCAST to communicate data. (Although this is quite simple and includes useless data communication, the speed-up ratio was good.)
- We divided do-loop as outer as possible.
- In this method, we can make all matrix elements as long as we determine the atomic positions. These data determined from 1 MD step are stored in CPU memory of all node. Although almost all data are useless, we do not need to communicate the data among nodes and there is no loss of data communication.

Elapse time of each part as a function of the number of CPU

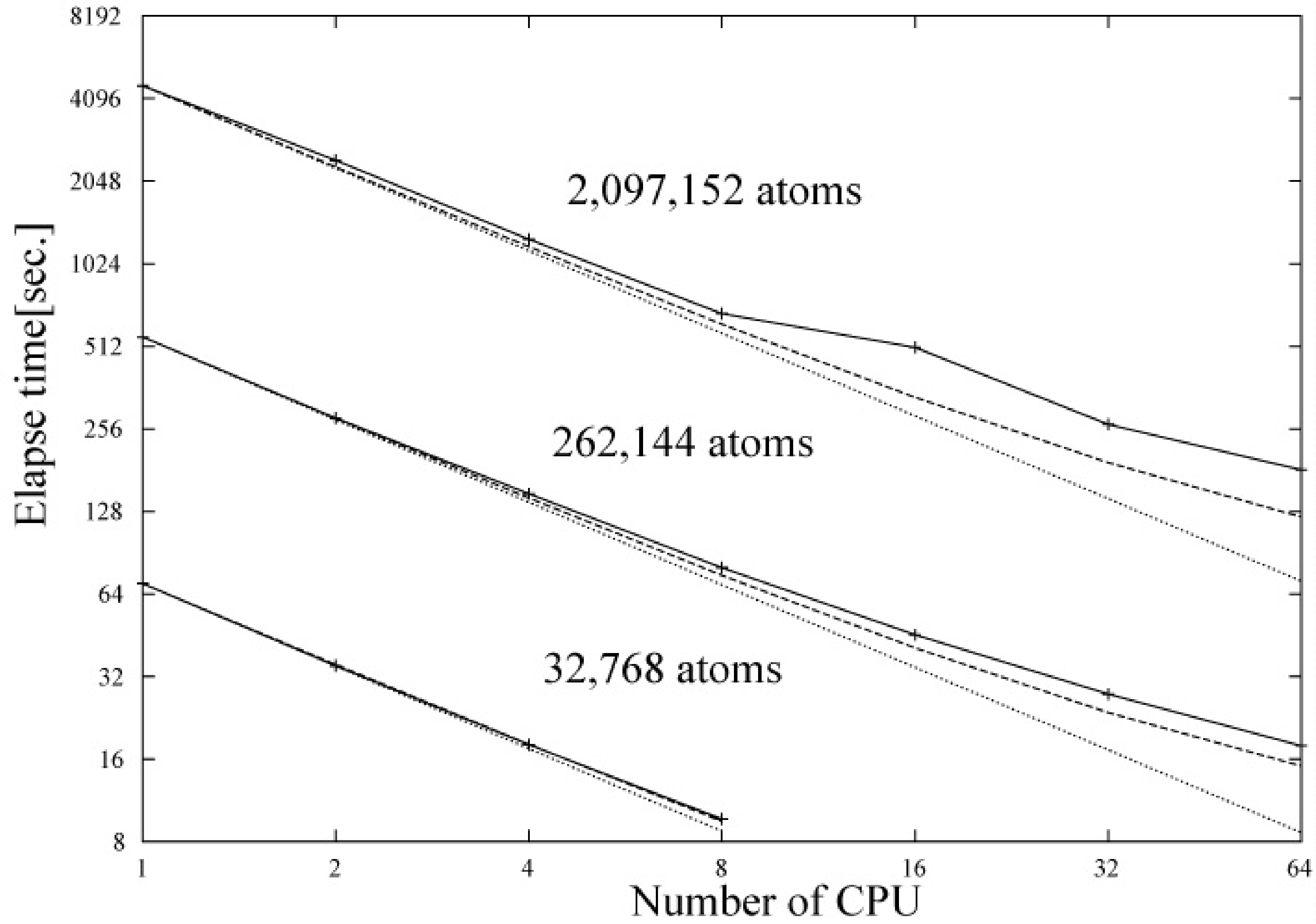


Geshi et al.(2003)

SGI Origin 3800

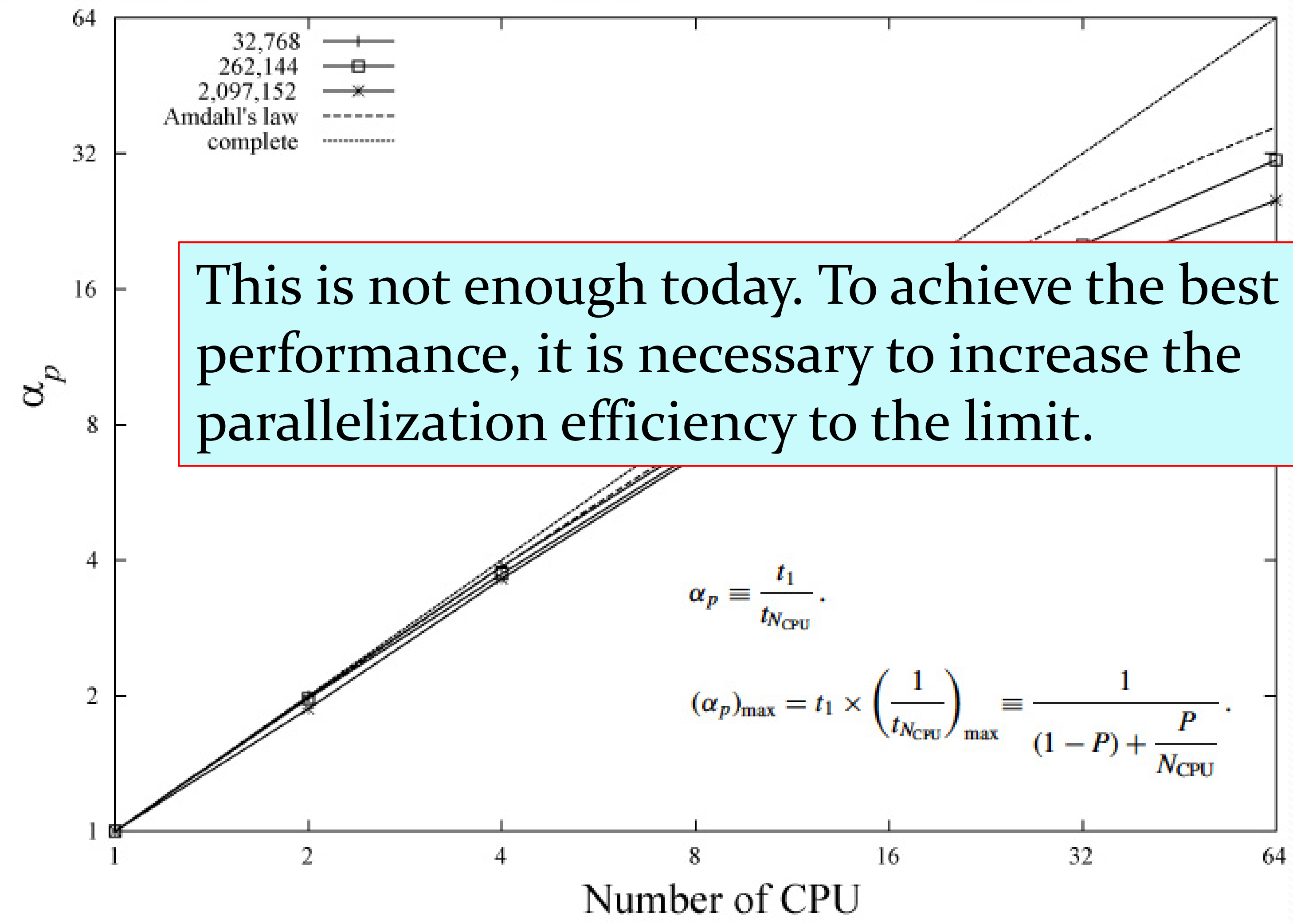
Elapse time as a function of the number of CPU

M. Geshi *et al*, J. Phys. Soc. Jpn. **72** ,2880 (2003).



Parallelization ratio

M. Geshi *et al*, J. Phys. Soc. Jpn. **72** ,2880 (2003).



$$\alpha_p \equiv \frac{t_1}{t_{N_{\mathrm{CPU}}}}.$$

$$(\alpha_p)_{\max} = t_1 \times \left(\frac{1}{t_{N_{\mathrm{CPU}}}}\right)_{\max} \equiv \frac{1}{(1-P)+\frac{P}{N_{\mathrm{CPU}}}}.$$

P=0.988

This is not enough today. To achieve the best performance, it is necessary to increase the parallelization efficiency to the limit.

Today’s supercomputers

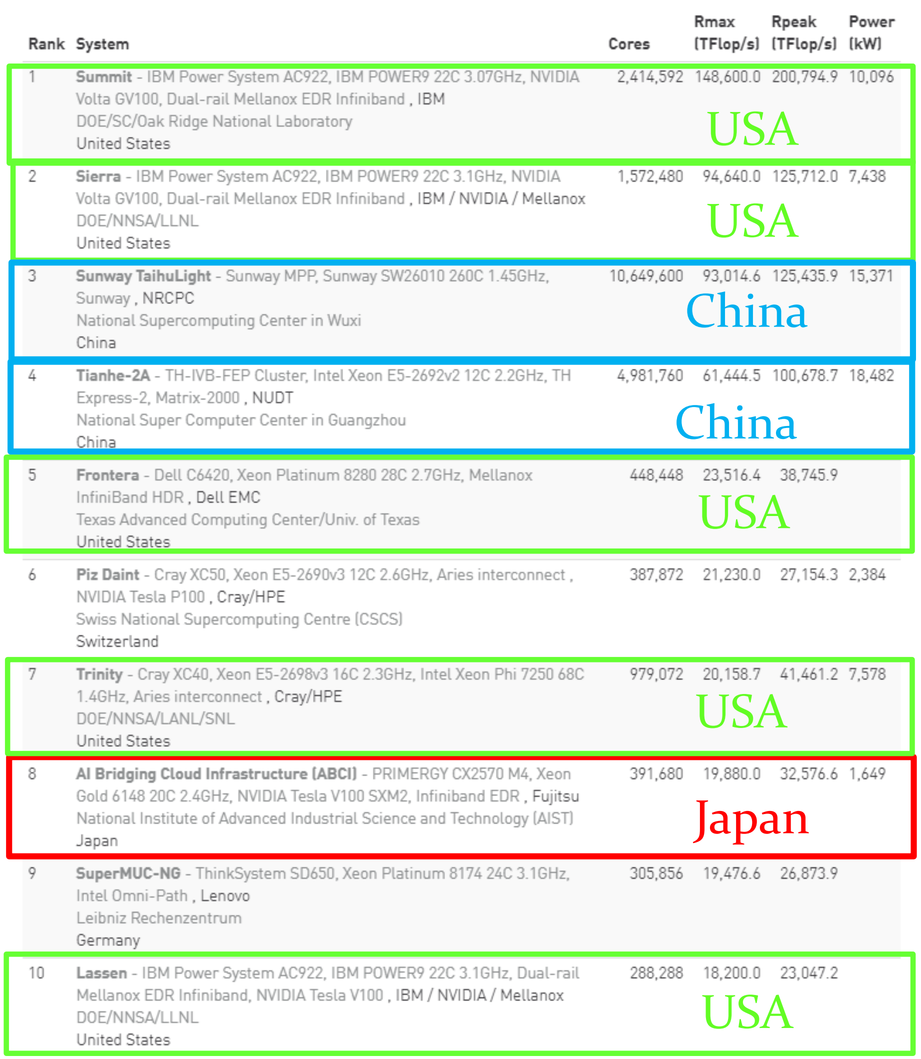
Classes of parallel computers

- Multicore computing
  - a processor that includes multiple execution units ("cores") on the same chip.
- Symmetric multiprocessing
  - a computer system with multiple identical processors that **share memory** and connect via a bus.
- Distributed computing
  - a distributed memory computer system in which the processing elements are connected by a network.(**Cluster computing, Massive parallel processing, Grid computing**)
- Specialized parallel computers
  - Within parallel computing, there are specialized parallel devices that remain niche areas of interest. (**GPGPU,Application-specific integrated circuits, Vector processors**)

From http://en.wikipedia.org/wiki/Parallel_computing

The latest top 500 supercomputer list (2019.11)(Latest)

http://www.top500.org/



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 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
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/HPE 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/HPE 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 AC922, 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	

By the way, Oakforest-PACS is 15, and TSUBAME3.0 is 23. In total, 14 Japanese supercomputers are in the top 100.

Intel XeonPhi
(This architecture was gone.)

LINPACK

- A software library for performing numerical linear algebra on digital computers.
- It is used for the performance assessment of supercomputers. In practice, High-Performance Linpack(HPL) is used.
- It makes use of the BLAS libraries for performing basic vector and matrix operations.
- The benchmark used in the LINPACK Benchmark is to solve a dense system of linear equations.

This benchmark shows nothing more than one side of performance of supercomputers.

HPC Challenge

The HPC Challenge benchmark consists of basically 7 tests:

- HPL - the Linpack TPP benchmark which measures the floating point rate of execution for solving a linear system of equations.
- DGEMM - measures the floating point rate of execution of double precision real matrix-matrix multiplication.
- STREAM - a simple synthetic benchmark program that measures sustainable memory bandwidth (in GB/s) and the corresponding computation rate for simple vector kernel.
- PTRANS (parallel matrix transpose) - exercises the communications where pairs of processors communicate with each other simultaneously. It is a useful test of the total communications capacity of the network.
- RandomAccess - measures the rate of integer random updates of memory (GUPS)
- FFT - measures the floating point rate of execution of double precision complex one-dimensional Discrete Fourier Transform (DFT).
- Communication bandwidth and latency - a set of tests to measure latency and bandwidth of a number of simultaneous communication patterns; based on b_eff (effective bandwidth benchmark).

http://www.hpcchallenge.org/

HPCG Benchmark(From 2014)

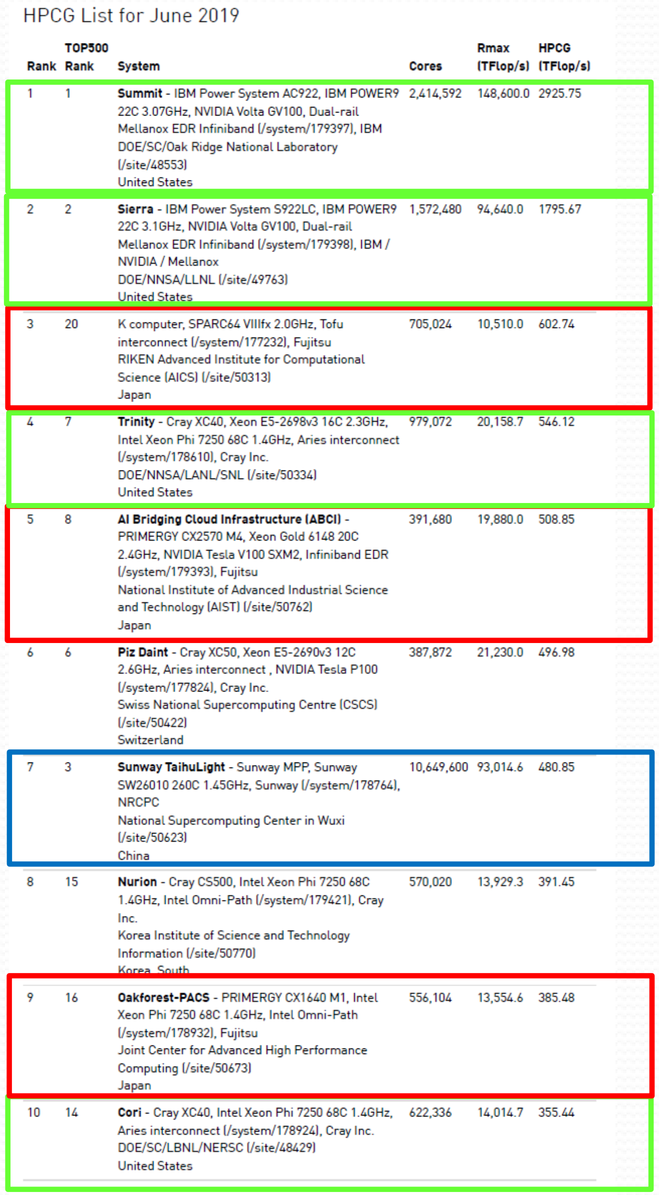
The High Performance Conjugate Gradients (HPCG) Benchmark project is an effort to create a new metric for ranking HPC systems. HPCG is intended as a complement to the High Performance LINPACK (HPL) benchmark, currently used to rank the TOP500 computing systems. The computational and data access patterns of HPL are still representative of some important scalable applications, but not all. HPCG is designed to exercise computational and data access patterns that more closely match a different and broad set of important applications, and to give incentive to computer system designers to invest in capabilities that will have impact on the collective performance of these applications.

HPCG is a complete, stand-alone code that measures the performance of basic operations in a unified code:

- Sparse matrix-vector multiplication.
- Vector updates.
- Global dot products.
- Local symmetric Gauss-Seidel smoother.
- Sparse triangular solve (as part of the Gauss-Seidel smoother).
- Driven by multigrid preconditioned conjugate gradient algorithm that exercises the key kernels on a nested set of coarse grids.
- Reference implementation is written in C++ with MPI and OpenMP support.

http://www.hpcg-benchmark.org/index.html

HPCG ranking(2019.6)



HPCG List for June 2019

Rank	TOP500 Rank	System	Cores	Rmax (TFlop/s)	HPCG (TFlop/s)
1	1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband (/system/179397), IBM DOE/SC/Oak Ridge National Laboratory (/site/48553) United States	2,414,592	148,600.0	2925.75
2	2	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband (/system/179398), IBM / NVIDIA / Mellanox DOE/NNSA/LLNL (/site/49763) United States	1,572,480	94,640.0	1795.67
3	20	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect (/system/177232), Fujitsu RIKEN Advanced Institute for Computational Science (AICS) (/site/50313) Japan	705,024	10,510.0	602.74
4	7	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect (/system/178610), Cray Inc. DOE/NNSA/LANL/SNL (/site/50334) United States	979,072	20,158.7	546.12
5	8	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2570 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR (/system/179393), Fujitsu National Institute of Advanced Industrial Science and Technology (AIST) (/site/50762) Japan	391,680	19,880.0	508.85
6	6	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 (/system/177824), Cray Inc. Swiss National Supercomputing Centre (CSCS) (/site/50422) Switzerland	387,872	21,230.0	496.98
7	3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway (/system/178764), NRCPC National Supercomputing Center in Wuxi (/site/50623) China	10,649,600	93,014.6	480.85
8	15	**Nurion** - Cray CS500, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path (/system/179421), Cray Inc. Korea Institute of Science and Technology Information (/site/50770) Korea, South	570,020	13,929.3	391.45
9	16	**Oakforest-PACS** - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path (/system/178932), Fujitsu Joint Center for Advanced High Performance Computing (/site/50673) Japan	556,104	13,554.6	385.48
10	14	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect (/system/178924), Cray Inc. DOE/SC/LBNL/NERSC (/site/48429) United States	622,336	14,014.7	355.44

USA

USA

Japan

USA

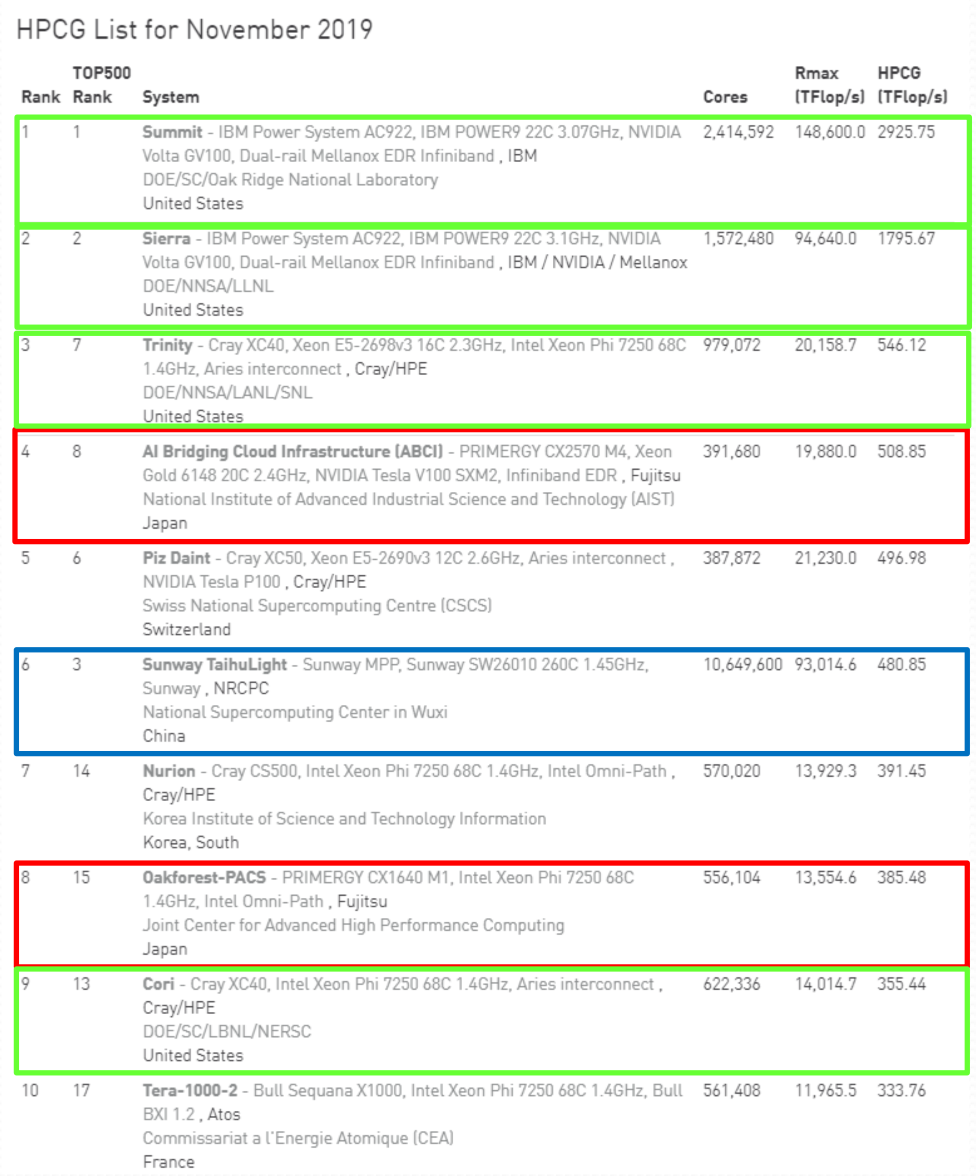
Japan

China

Japan

USA

HPCG ranking(2019.11)(Latest)



HPCG List for November 2019

	TOP500			Rmax	HPCG
Rank	Rank	System	Cores	(TFlop/s)	(TFlop/s)
1	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	2925.75
2	2	**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	1795.67
3	7	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray/HPE DOE/NNSA/LANL/SNL United States	979,072	20,158.7	546.12
4	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	508.85
5	6	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray/HPE Swiss National Supercomputing Centre (CSCS) Switzerland	387,872	21,230.0	496.98
6	3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	480.85
7	14	**Nurion** - Cray CS500, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path , Cray/HPE Korea Institute of Science and Technology Information Korea, South	570,020	13,929.3	391.45
8	15	**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	385.48
9	13	**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	355.44
10	17	**Tera-1000-2** - Bull Sequana X1000, Intel Xeon Phi 7250 68C 1.4GHz, Bull BXI 1.2 , Atos Commissariat a l'Energie Atomique (CEA) France	561,408	11,965.5	333.76

USA

USA

USA

Japan

China

Japan

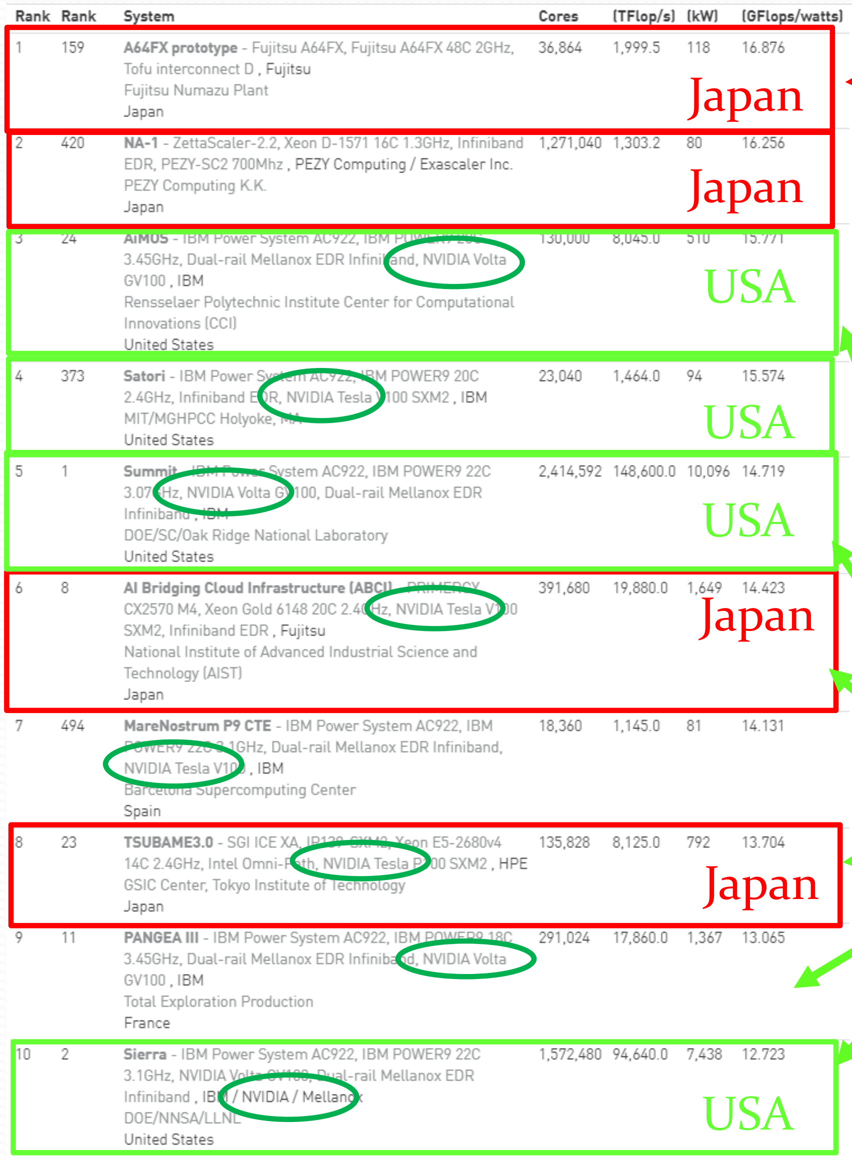
USA

Green500(省エネスパコンリスト)

- The purpose of the Green500 is to provide a ranking of the most energy-efficient supercomputers in the world. For decades, the notion of "performance" has been synonymous with "speed" (as measured in FLOPS). This particular focus has led to the emergence of supercomputers that consume egregious amounts of electrical power and produce so much heat that extravagant cooling facilities must be constructed to ensure proper operation. In addition, the emphasis on speed as the ultimate metric has caused other metrics such as reliability, availability, and usability to be largely ignored. As a result, there has been an extraordinary increase in the total cost of ownership (TCO) of a supercomputer.

http://www.green500.org/

Green 500(2019.11)(Latest)



Japan
Japan
USA
USA
USA
Japan
Japan
USA

Rank	Rank	System	Cores	(TFlop/s)	(kW)	(GFlops/watts)
1	159	**A64FX prototype** - Fujitsu A64FX, Fujitsu A64FX 48C 2GHz, Tofu interconnect D , Fujitsu Fujitsu Numazu Plant Japan	36,864	1,999.5	118	16.876
2	420	**NA-1** - ZettaScaler-2.2, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 700Mhz , PEZY Computing / Exascaler Inc. PEZY Computing K.K. Japan	1,271,040	1,303.2	80	16.256
3	24	**AiMOS** - IBM Power System AC922, IBM POWER9 20C 3.45GHz, Dual-rail Mellanox EDR Infiniband, NVIDIA Volta GV100 , IBM Rensselaer Polytechnic Institute Center for Computational Innovations (CCI) United States	130,000	8,045.0	510	15.771
4	373	**Satori** - IBM Power System AC922, IBM POWER9 20C 2.4GHz, Infiniband EDR, NVIDIA Tesla V100 SXM2 , IBM MIT/MGHPCC Holyoke, MA United States	23,040	1,464.0	94	15.574
5	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	10,096	14.719
6	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	1,649	14.423
7	494	**MareNostrum P9 CTE** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, Dual-rail Mellanox EDR Infiniband, NVIDIA Tesla V100 , IBM Barcelona Supercomputing Center Spain	18,360	1,145.0	81	14.131
8	23	**TSUBAME3.0** - SGI ICE XA, IP139-SXM2, Xeon E5-2680v4 14C 2.4GHz, Intel Omni-Path, NVIDIA Tesla P100 SXM2 , HPE GSIC Center, Tokyo Institute of Technology Japan	135,828	8,125.0	792	13.704
9	11	**PANGEA III** - IBM Power System AC922, IBM POWER9 18C 3.45GHz, Dual-rail Mellanox EDR Infiniband, NVIDIA Volta GV100 , IBM Total Exploration Production France	291,024	17,860.0	1,367	13.065
10	2	**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	7,438	12.723

GPU

GPU

GPU

GPU

GPU

GPU

GPU

GPU

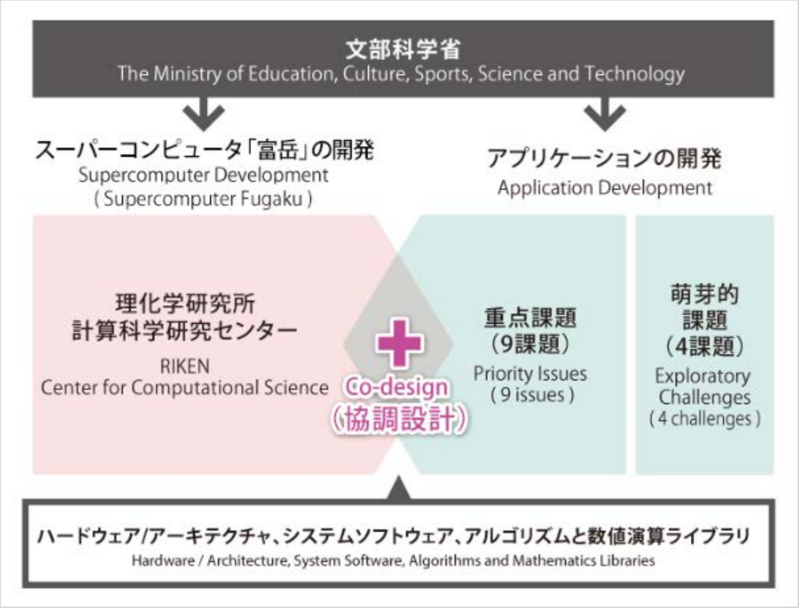
The prototype of Fugaku.

This list fluctuates violently every time. That is why emphasis is placed on the development of this technology.

These machines balance energy saving and computing performance.

https://www.top500.org/green500/

post-K “Fugaku”project on-going



文部科学省
The Ministry of Education, Culture, Sports, Science and Technology

スーパーコンピュータ「富岳」の開発
Supercomputer Development
(Supercomputer Fugaku)

アプリケーションの開発
Application Development

理化学研究所
計算科学研究センター
RIKEN
Center for Computational Science

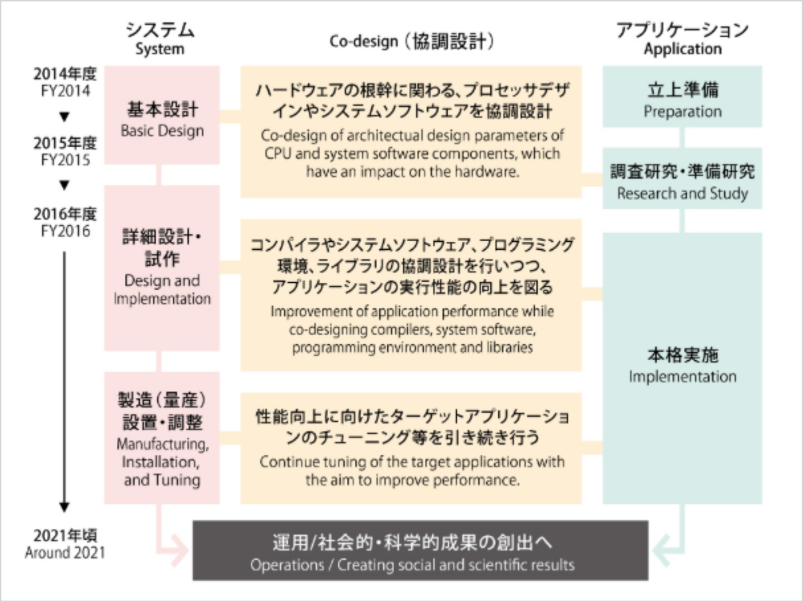
Co-design
（協調設計）

重点課題
（9課題）
Priority Issues
(9 issues)

萌芽的
課題
（4課題）
Exploratory
Challenges
(4 challenges)

ハードウェア/アーキテクチャ、システムソフトウェア、アルゴリズムと数値演算ライブラリ
Hardware / Architecture, System Software, Algorithms and Mathematics Libraries

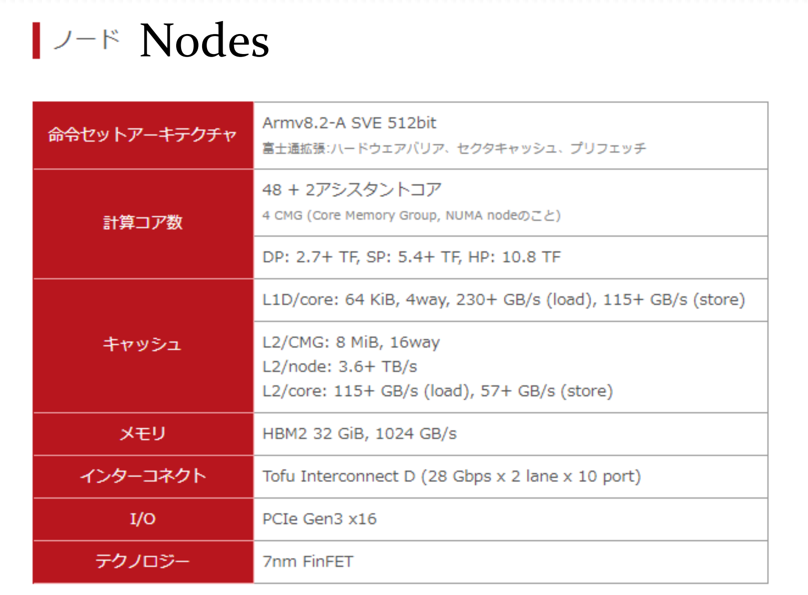
タイムライン



	システム System	Co-design（協調設計）	アプリケーション Application
2014年度 FY2014 ▼ 2015年度 FY2015	基本設計 Basic Design	ハードウェアの根幹に関わる、プロセッサデザインやシステムソフトウェアを協調設計 Co-design of architectual design parameters of CPU and system software components, which have an impact on the hardware.	立上準備 Preparation 調査研究・準備研究 Research and Study
▼ 2016年度 FY2016	詳細設計・試作 Design and Implementation	コンパイラやシステムソフトウェア、プログラミング環境、ライブラリの協調設計を行いつつ、アプリケーションの実行性能の向上を図る Improvement of application performance while co-designing compilers, system software, programming environment and libraries	本格実施 Implementation
	製造（量産）設置・調整 Manufacturing, Installation, and Tuning	性能向上に向けたターゲットアプリケーションのチューニング等を引き続き行う Continue tuning of the target applications with the aim to improve performance.	
2021年頃 Around 2021	運用/社会的・科学的成果の創出へ Operations / Creating social and scientific results		

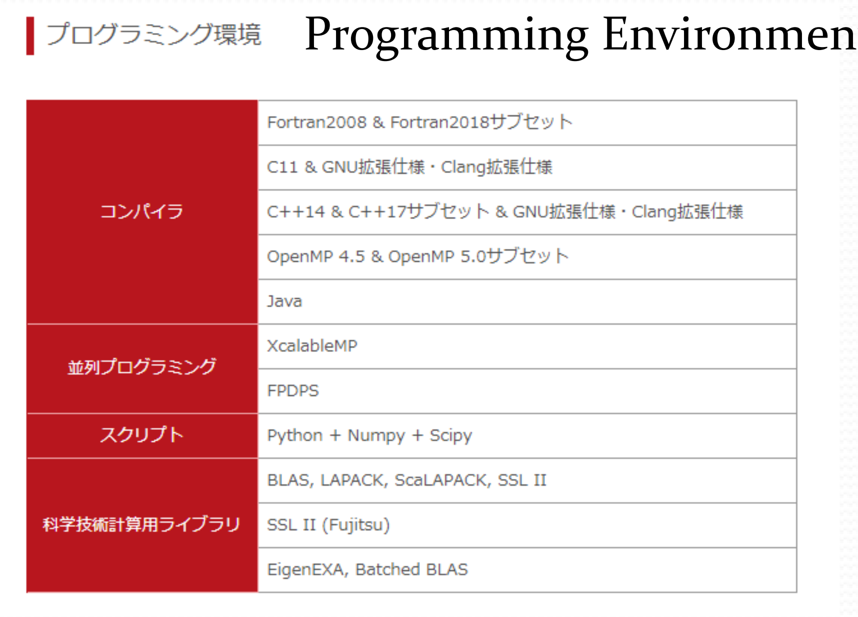
https://www.r-ccs.riken.jp/jp/post-k/project.html

Published specifications of Fugaku



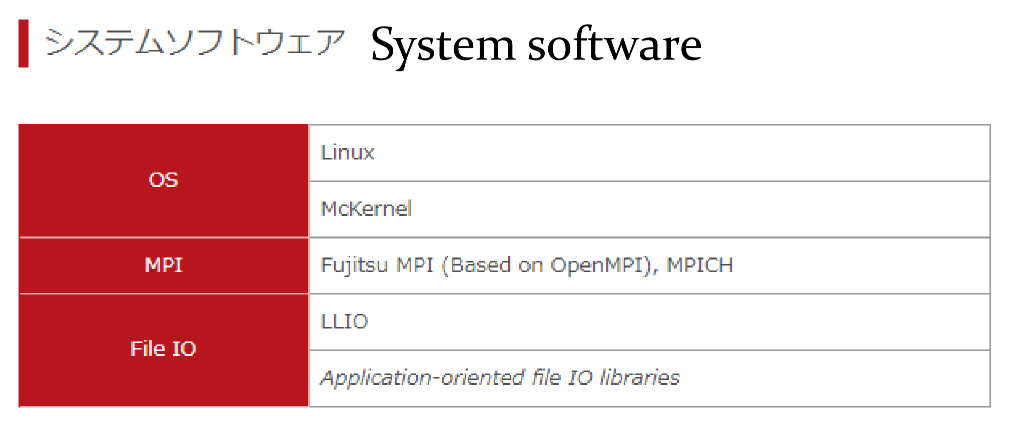
ノード Nodes

命令セットアーキテクチャ	Armv8.2-A SVE 512bit 富士通拡張:ハードウェアバリア、セクタキャッシュ、プリフェッチ
計算コア数	48 + 2アシスタントコア 4 CMG (Core Memory Group, NUMA nodeのこと)
	DP: 2.7+ TF, SP: 5.4+ TF, HP: 10.8 TF
キャッシュ	L1D/core: 64 KiB, 4way, 230+ GB/s (load), 115+ GB/s (store)
	L2/CMG: 8 MiB, 16way L2/node: 3.6+ TB/s L2/core: 115+ GB/s (load), 57+ GB/s (store)
メモリ	HBM2 32 GiB, 1024 GB/s
インターコネクト	Tofu Interconnect D (28 Gbps x 2 lane x 10 port)
I/O	PCIe Gen3 x16
テクノロジー	7nm FinFET



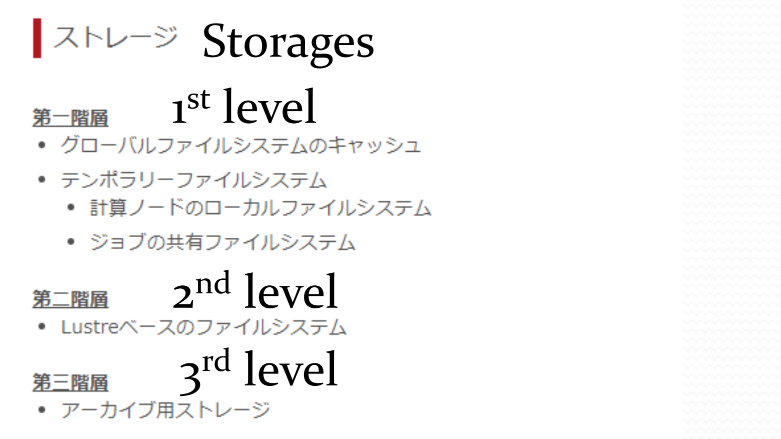
プログラミング環境 Programming Environment

コンパイラ	Fortran2008 & Fortran2018サブセット
	C11 & GNU拡張仕様・Clang拡張仕様
	C++14 & C++17サブセット & GNU拡張仕様・Clang拡張仕様
	OpenMP 4.5 & OpenMP 5.0サブセット
	Java
並列プログラミング	XcalableMP
	FPDPS
スクリプト	Python + Numpy + Scipy
科学技術計算用ライブラリ	BLAS, LAPACK, ScaLAPACK, SSL II
	SSL II (Fujitsu)
	EigenEXA, Batched BLAS



システムソフトウェア System software

OS	Linux
	McKernel
MPI	Fujitsu MPI (Based on OpenMPI), MPICH
File IO	LLIO
	Application-oriented file IO libraries



ストレージ Storages

第一階層 1st level

- グローバルファイルシステムのキャッシュ
- テンポラリーファイルシステム
  - 計算ノードのローカルファイルシステム
  - ジョブの共有ファイルシステム

第二階層 2nd level

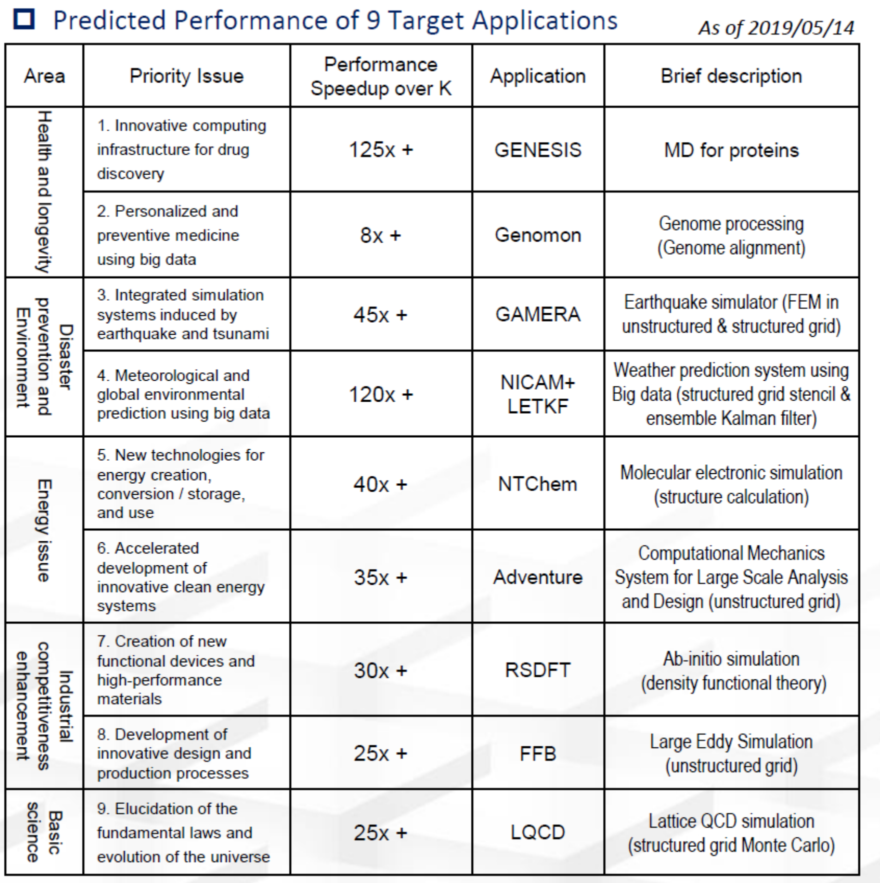
- Lustreベースのファイルシステム

第三階層 3rd level

- アーカイブ用ストレージ

Comparison between K and Fugaku

		Fugaku	K
CPU Architecture		A64FX (Armv8.2-A SVE +Fujitsu Extension)	SPARC64 VIIIfx
Node	Cores	48	8
	Peak DP performance	3.0720 TF (3.3792 TF)	0.128 TF
	Main Memory	32 GiB	16 GiB
	Peak Memory Bandwidth	1024 GB/s	64 GB/s
	Peak Network Performance	40.8 GB/s	20 GB/s
Rack	Nodes	384	102
	Peak DP performance	1.2/1.3 PF	< 0.013PF
Process Technology		7 nm FinFET	45 nm



□ Predicted Performance of 9 Target Applications *As of 2019/05/14*

Area	Priority Issue	Performance Speedup over K	Application	Brief description
Health and longevity	1. Innovative computing infrastructure for drug discovery	125x +	GENESIS	MD for proteins
	2. Personalized and preventive medicine using big data	8x +	Genomon	Genome processing (Genome alignment)
Disaster prevention and Environment	3. Integrated simulation systems induced by earthquake and tsunami	45x +	GAMERA	Earthquake simulator (FEM in unstructured & structured grid)
	4. Meteorological and global environmental prediction using big data	120x +	NICAM+ LETKF	Weather prediction system using Big data (structured grid stencil & ensemble Kalman filter)
Energy issue	5. New technologies for energy creation, conversion / storage, and use	40x +	NTChem	Molecular electronic simulation (structure calculation)
	6. Accelerated development of innovative clean energy systems	35x +	Adventure	Computational Mechanics System for Large Scale Analysis and Design (unstructured grid)
Industrial competitiveness enhancement	7. Creation of new functional devices and high-performance materials	30x +	RSDFT	Ab-initio simulation (density functional theory)
	8. Development of innovative design and production processes	25x +	FFB	Large Eddy Simulation (unstructured grid)
Basic science	9. Elucidation of the fundamental laws and evolution of the universe	25x +	LQCD	Lattice QCD simulation (structured grid Monte Carlo)

今後の HPCI システムの構築とその利用
に関する意見交換会
2020/01/27

K computer is working on 24h/365days

- K computer was active for public use on 24hours/365 days for general users.
  - Different from K-computer, almost all the top class machines of the top500 ranking are for closed users and are used for limited purposes.
- The several rankings showed that K computer is almighty.
- Users may optimize their code for K computer easier than for the other ones(in the sense that it does not use a special architecture like GPU.).

Fugaku will be operated in the same way as K-computer.

Future direction of large-scale computing

From the above rankings

- From now on, both performance and energy-saving are important (Summit makes energy saving and computing performance highly compatible).
- The trend of many-core system with not so high frequency CPU is still going on. →Fugaku has 48 cores.
- Or many supercomputers have GPU and realize energy saving.
- As GPU is prominent in the field of AI and machine learning, this architecture will continue to be adopted in the future.
- However, I do not know whether supercomputers made up of GPU are effective in all fields or not.
- Probably, many supercomputers designed for data analysis and AI will appear after now.

However,…

- Effective performances measured by LINPACK or HPCG benchmarks are not necessarily accordance with the performance of our software codes. Usually, we spend so much effort to get better performance on supercomputers than typical PC clusters.
  →Fugaku, will be constructed for getting better performance for scientific application than benchmark programs (Co-design). (However, it may be the national mission to get the first prize of top 500 for taxpayers.)
- The effort goes beyond knowledge and techniques of physicists, chemist, biologists,… We need those of computer scientists and numerical mathematicians and need to collaborate them.
- Fugaku will be also a computer for general purpose. It may be inferior to a dedicated machine.(cf. Anton (supercomputer for performing classical MD) is 100~1000 faster than normal supercomputers.)

Future direction1

- The tendency of architecture is multi cores or many cores (including SIMD,GPU, …).
- It is not easy to use all cores effectively.

(As personal opinion, I think that I do not have to worry about using all the cores and it is enough to get the best efficiency to obtain the calculation result.)

- Is it reasonable and proper to devote many researchers' effort to develop parallel efficiency? (very serious problem!!!)
- For DFT calculations, it may be difficult to speed-up the software by using GPU without breakthrough.
- We have to perform feasibility studies continuously to follow a leading-edge architecture.

Future direction2

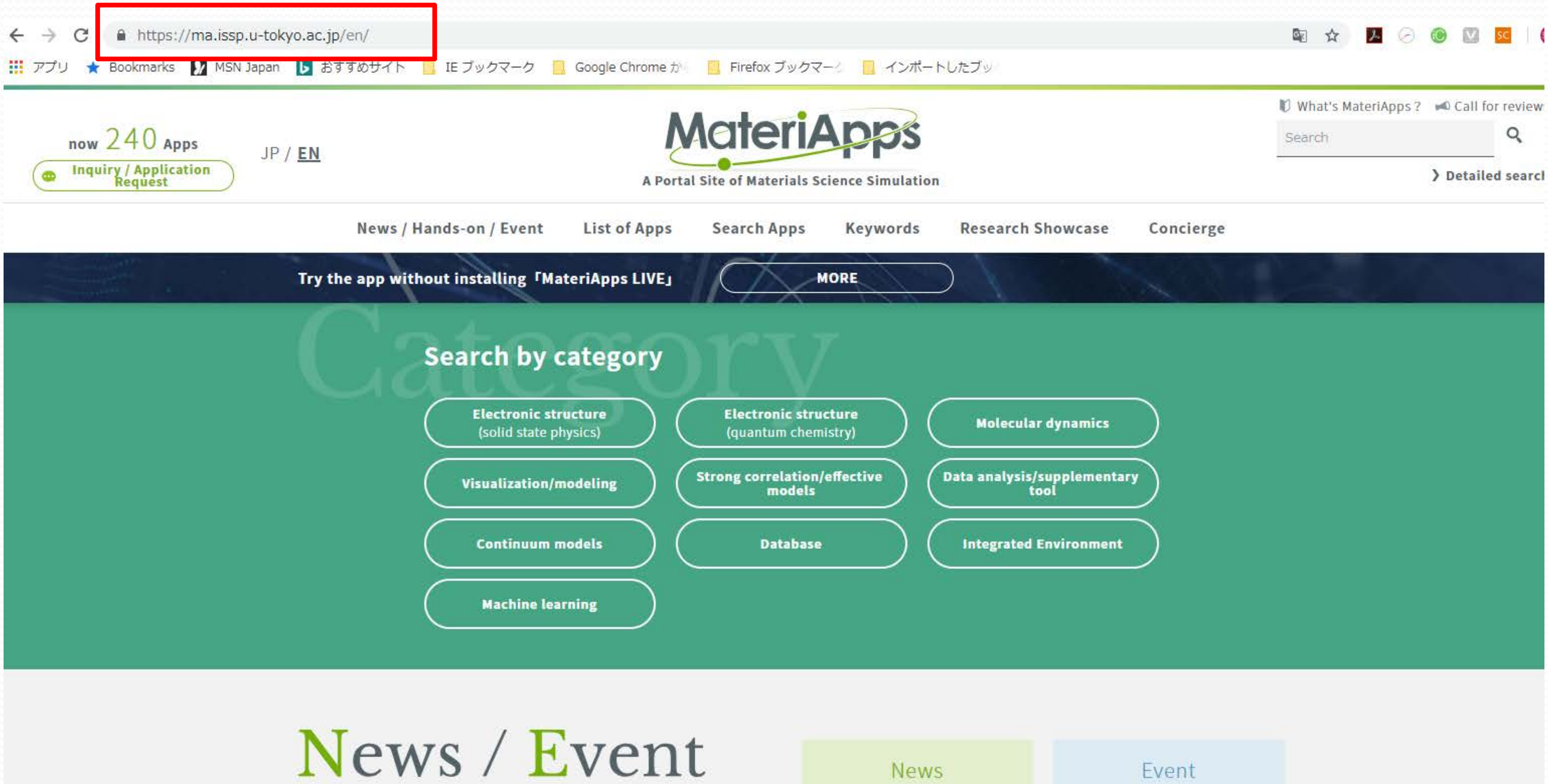
- The United States develops both architectures and programming language frameworks. We should contribute those.
- It is too much hard-pressed for physicists and chemists to cover computer science or numerical analysis.

  →it is needed to establish a system to promote to work together with computational scientists and computer scientists or experts of numerical analysis.
- Still now, the guarantee of accuracy of calculation is inadequacy. There is no standard for summation of data from cores (threads) on MPI or OpenMP,… We must treat it carefully, otherwise our calculation may become meaningless.

Future direction3

- Today’s amassed skills through K computer can be used during the next 10 years. (In Japan, Fugaku project, and the next project)
- However, the development of supercomputer must be stop in near future.(Prof. Hiraki said it was 2029. The reason is the necessity of huge electric power, failure of Moors’ law, etc…)
- We should consider a new idea to get high performance of our own calculations for both software and hardware as soon as possible. We may undergo a paradigm shift before 2029.
- **For the time being, data transfer will continue to be the most costly at all levels.**
- The research on quantum computers is also advancing, and that will be a focus in the future.

MateriApps



https://ma.issp.u-tokyo.ac.jp/en/

Try the app without installing「MateriApps LIVE」

Search by category

News / Event

This portal site provides the information of applications that are used in materials science. You can find your desired application. “MateriApps LIVE” gives you an environment to perform open-source software codes without any installation process.

Finally,

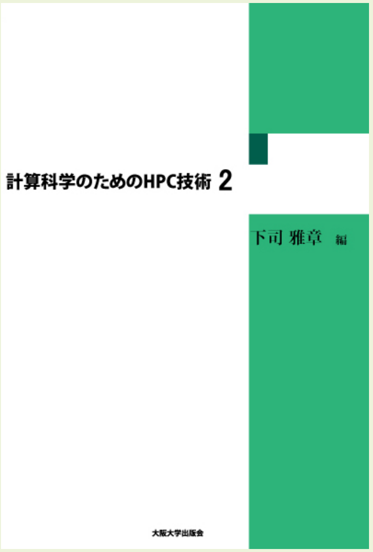
- We should advance the development of human resources to be able to use supercomputers and/or HPC machines effectively.
- The relation between the ranking of top500 and the software code we use is not necessarily clear. We need to have the knowledge of the architecture of the supercomputers and/or the HPC machines if we'd like to maximize the performance of our software codes.
- We should cultivate people who can develop the software code.
- We should cultivate people who can effectively use scientific software codes to progress the researches. (CMD-WS, distance learning etc…)

Related texts

計算科学のためのHPC技術 1

下司 雅章 編

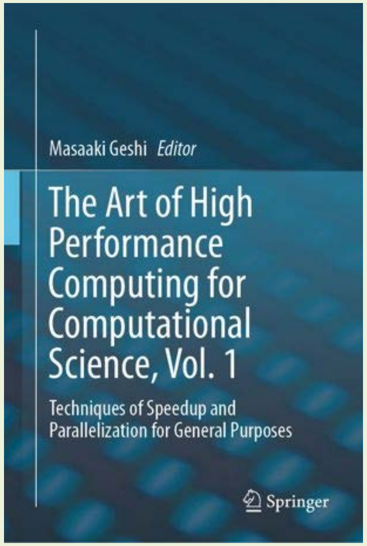
大阪大学出版会



計算科学のためのHPC技術 2

下司 雅章 編

大阪大学出版会

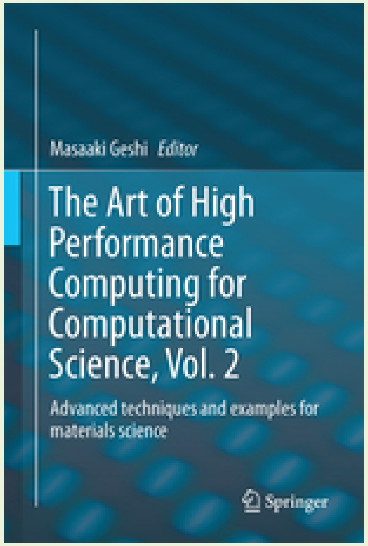


Masaaki Geshi *Editor*

The Art of High Performance Computing for Computational Science, Vol. 1

Techniques of Speedup and Parallelization for General Purposes

Springer



Masaaki Geshi *Editor*

The Art of High Performance Computing for Computational Science, Vol. 2

Advanced techniques and examples for materials science

Springer

- These books are for developers.
- Integrates techniques for development on massively parallel computers such as supercomputers.
- Lecture videos and texts are available on the following site(Japanese only).

https://www.r-ccs.riken.jp/library/event/tokuronA_2019.html

https://www.r-ccs.riken.jp/library/event/tokuronB_180406.html