

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

R³ Institute for Newly-Emerging Science Design,
The University of Osaka
大阪大学エマージングサイエンスデザインR³センター

Masaaki Geshi
下司 雅章

47th CMD workshop, supercomputer course
2025, September 1st (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.

It may not be clear how much computation is required before something can be called "large-scale." There is no strict standard, but in practice, it is often considered large-scale once the problem size exceeds the capacity of the computers available within your group.

Problems caused by large-scale computing(1)

When performing large-scale computations, issues arise that do not occur in small-scale runs.



- Because huge amounts of data must be handled, very large storage and fast I/O are required (~30 PB for the K computer, ~150 PB for Fugaku. How about your lab system? Do you know the specifications of the system you normally use?).
- Even at the level of shared PC clusters, problems can arise when increasing the number of atoms, the number of k-point samples, the cutoff energy, or other parameters.
 - Data from wavefunctions in first-principles calculations can reach a few GB per job, and high-throughput calculations may require tens of TB or more.
 - Data from atomic positions in MD simulations can be ~300 GB or more (e.g., for one million atoms over 10 million MD steps).
 - You need to monitor the remaining capacity of your home directory or workspace—if the disk overflows, it will affect other users.
 - You also need to be aware of the memory capacity of compute nodes. If your job uses swap space, calculation speed on a PC cluster can slow down drastically (10x or more). On most supercomputers, such jobs are automatically terminated.

Problems caused by large-scale computing(2)

Issues in data analysis and visualization

- For atomic systems with over 1 million atoms, generating a single snapshot takes about 30 seconds. Producing 1,000 snapshots requires more than 8 hours.
- In some cases, data analysis takes longer than the computation itself.
- Data transfer becomes a serious issue when handling massive datasets generated on supercomputers outside the lab.
- Using paid cloud services such as AWS can be extremely costly, as transferring more than a few terabytes of data incurs high charges.
- ...

When we perform large-scale computing,...

- It requires enormous computational time.
 - In some cases, massive computational resources are needed (e.g., CPU or GPU memory, HDD, etc.).
- 
- To address these problems, software codes must be improved to reduce both computational time and memory usage. (The system also needs sufficient HDD capacity.)
- 
- Parallel computing is absolutely essential.

Many of you may not be very familiar with computing systems. However, to fully utilize HPC systems such as supercomputers, it is important to understand as much as possible about the system.

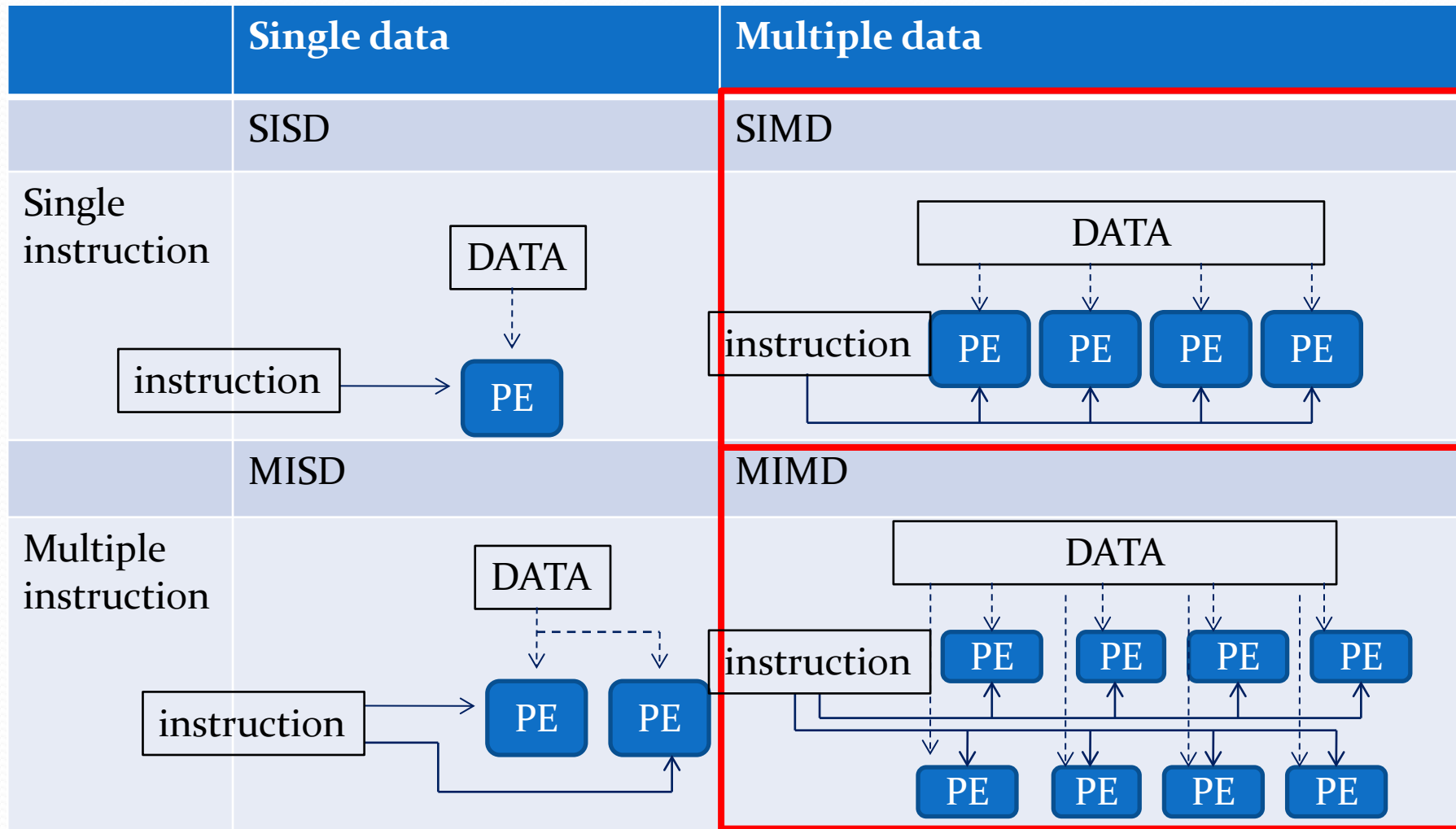
Methods for large-scale computing

- Speed-up techniques
 - Vectorization→SIMD(Single Instruction, Multiple Data)
 - Parallelization
 - Order- N method (reduce the cost from $O(N^p) \rightarrow O(N)$, with $p = 1$)
 - Use of specialized machines or accelerators (e.g. GPU)
- These methods are applied in codes depending on necessity.
- The easiest way to achieve speed-up is to use proper compile options (vectorization, automatic parallelization, optimization, etc.).

Since code cannot be automatically accelerated to its best performance, significant tuning is required..

You may not be very familiar with the details of the software. However, to fully exploit its capabilities, you should also understand the software in depth.

Flynn's taxonomy (フリンの分類)



SIMD: A single instruction is applied to multiple data sets.

MIMD: Multiple instructions are applied to multiple data sets.

Vectorization

- A **vector processor**, or array processor, is a CPU that implements an instruction set containing operations on one-dimensional arrays of data called **vectors** (rows, columns, diagonals, etc.).
- **Vectorization** is the method of programming a code to construct regularly arrayed data and process it all at once using the vector CPU.
- Vectorization reduces CPU time.
- It is basically the same concept as **SIMD**.

Vector processing

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

multiple operations at
once (innermost loop
is parallelized)

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)
```

one operation at a time

Vector processing

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

“a” array

“b” array

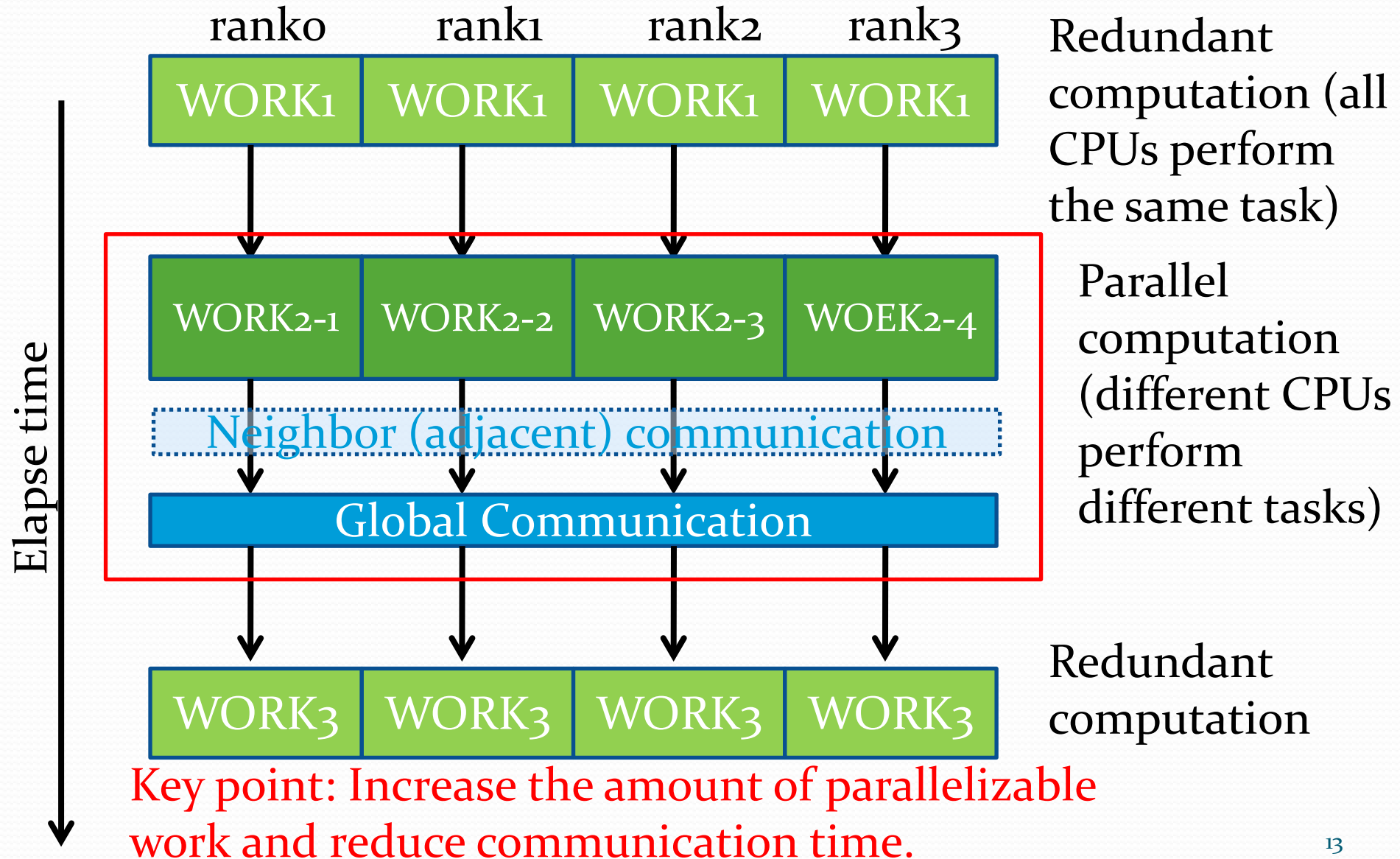
Order- N

- Realized by algorithm
 - No approximation example: Screened-KKR
- Realized by approximation of Hamiltonian
 - Approximation based on localization of interactions. These methods were actively studied in the 1990s.
 - Roughly speaking, matrices are block-diagonalized or treated with similar methods.
 - $N \times N \rightarrow N \times N_{\text{local}}(\text{fixed})$. If N increases, N_{local} does not increase. (similar to a tight-binding approximation)
 - The accuracy of the approximation can be adjusted depending on the required level of accuracy.
 - One famous method is the Divide-and-Conquer approach (分割統治法). It is implemented in codes such as “OpenMX(Ozaki)”, “DC(Kobayashi and Nakai)”, CONQUEST(Bowler, Miyazaki, Gillan), ...

Parallelization

- This method reduces the elapsed time by distributing tasks across multiple CPUs, without changing the processing time of each individual task.
- It is also used to share large memory arrays.

Concept of parallelization



Problems in parallelization

- The ratio of the parallelizable portion of the elapsed time in a CPU sequential job is critical (Amdahl's law).
- When a program is parallelized, data communication between nodes becomes necessary. As the number of nodes increases, **the communication time can exceed the computation time.** (In classical MD simulations, this becomes a crucial issue.)

Amdahl's Law

Speed up ratio $\alpha_P \equiv \frac{t_1}{t_{N_{\text{cores}}}}$

where t_1 = elapsed time of a job on 1 core,
 $t_{N_{\text{cores}}}$ = elapsed time of a job on N cores.

$$(\alpha_P)_{\max} = \frac{1}{(1 - P) + \frac{P}{N_{\text{cores}}}}$$

where P = fraction of execution time that can be parallelized.

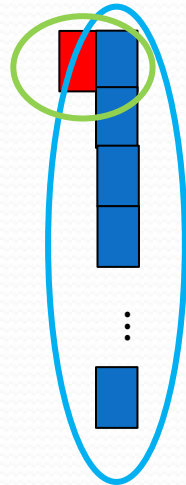
Upper bound of speedup due to parallelization
Optimize the program so that P approaches 1.

For examples,

Non-parallelizable (1 sec.) parallelizable (99 sec.)



100 processors {



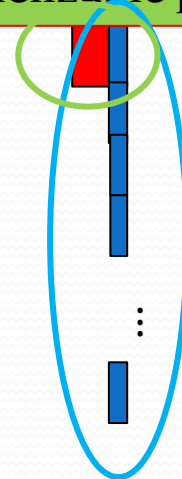
Execution time = $1 + 0.99 = 1.99$ sec. (almost $50\times$ speedup)

Even with
does not

We must **increase the parallelizable part** and **minimize the non-parallelizable part** as much as possible.

We need to decrease the non-parallelizable part as much as possible.

1000 processors {



Execution time = $1 + 0.099 = 1.099$ sec. (almost $91\times$ speedup)

Even with 1000 processors, the speedup does not reach $100\times$!!!

Conclusion: The benefits of parallelization diminish as the number of processors increases.

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.)

This law does not consider the communication time between nodes.

Indices of parallelization efficiency

- Strong scaling — **The overall problem size (e.g., number of atoms or electrons) is fixed**, while 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 increases with the number of processors (based on Gustafson's law).
 - The workload per processor remains unchanged.
 - Ideally, the computation time should remain constant as the number of processors increases.
 - If the computation time increases, it indicates the presence of non-parallelizable parts.
 - If the communication time between processors increases, it suggests inefficiencies in the communication method.

Reference: <https://www.r-ccs.riken.jp/outreach/schools/20240411-0725/>
<https://www.docswell.com/s/2300203199/K4QX1J-2024-04-16-133335>

Strong scaling and Weak scaling measurement

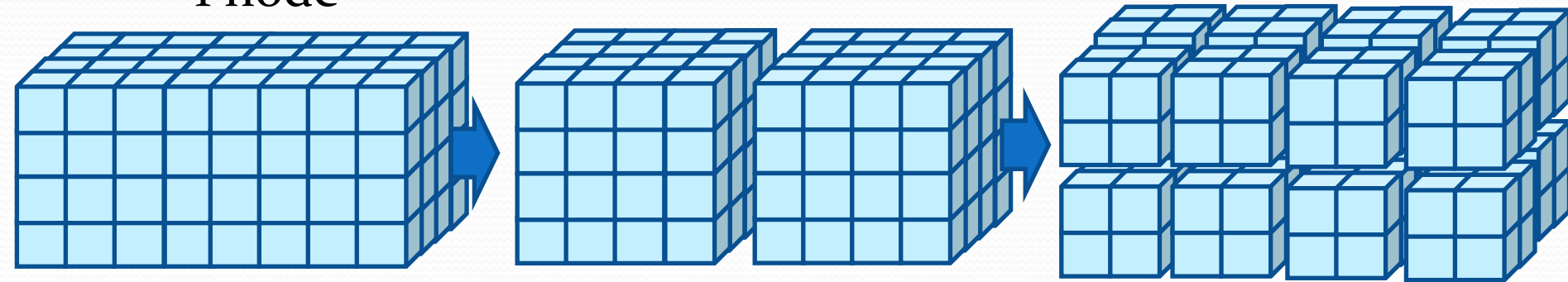
Changing only the degree of parallelization

Strong scaling

1 node

2 nodes

16 nodes

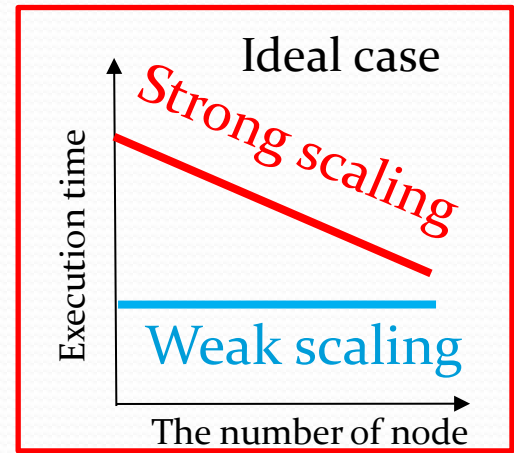
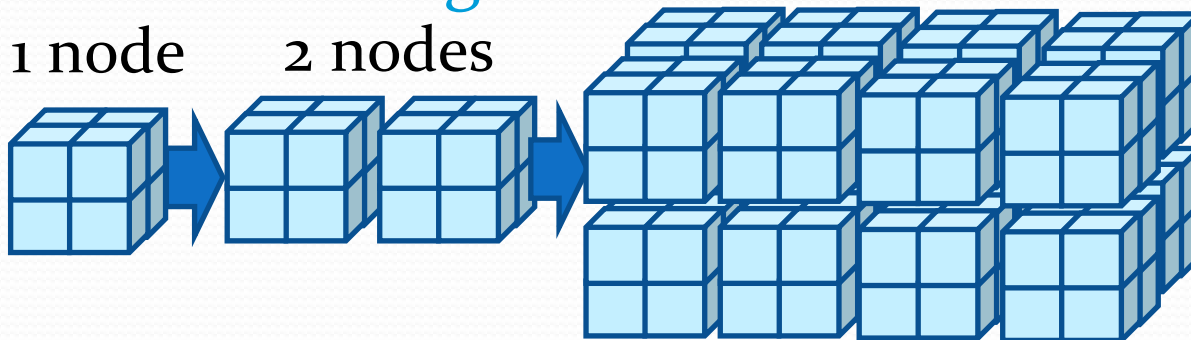


Weak scaling

1 node

2 nodes

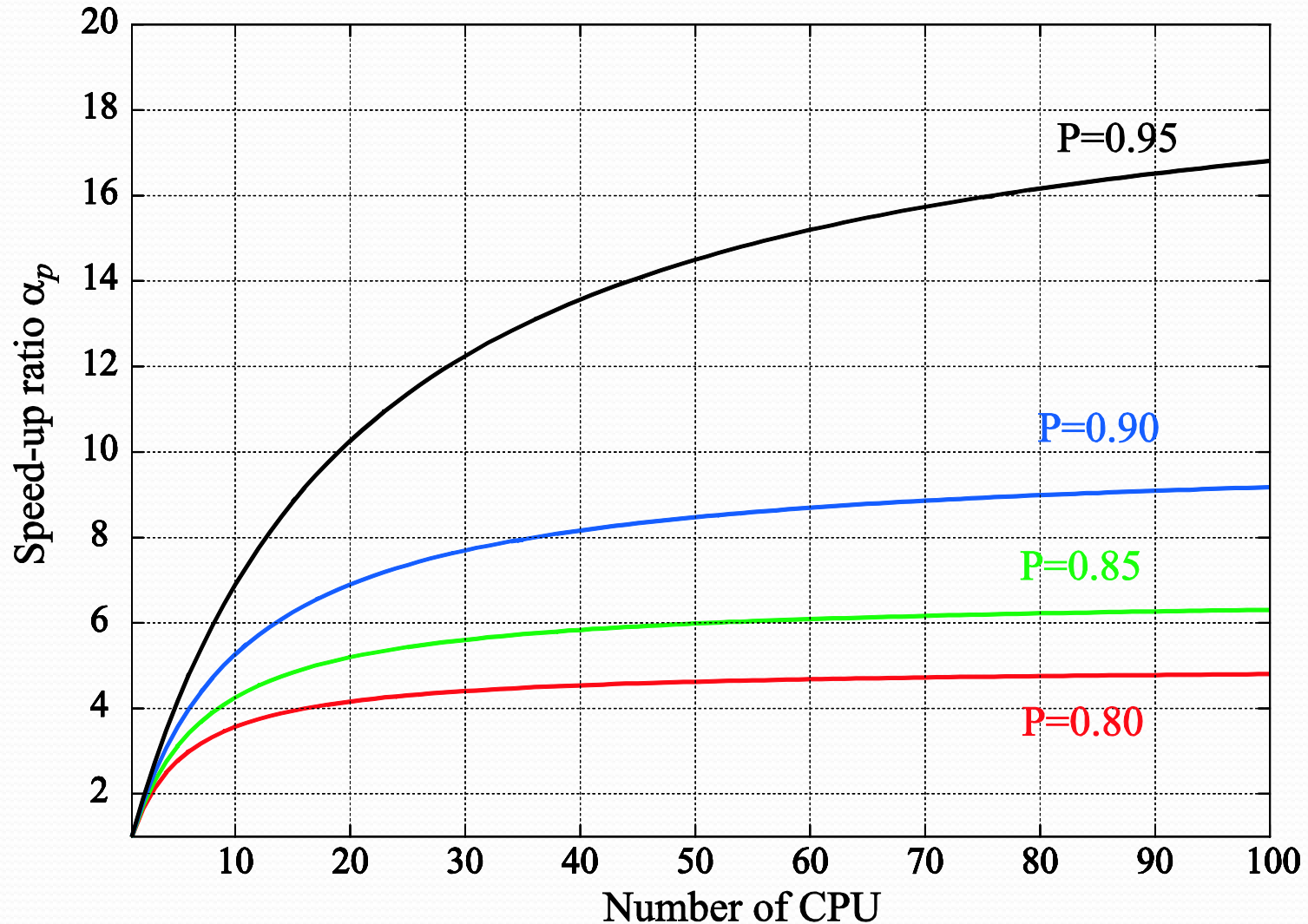
16 nodes



Prepare one input dataset for each parallelization case.

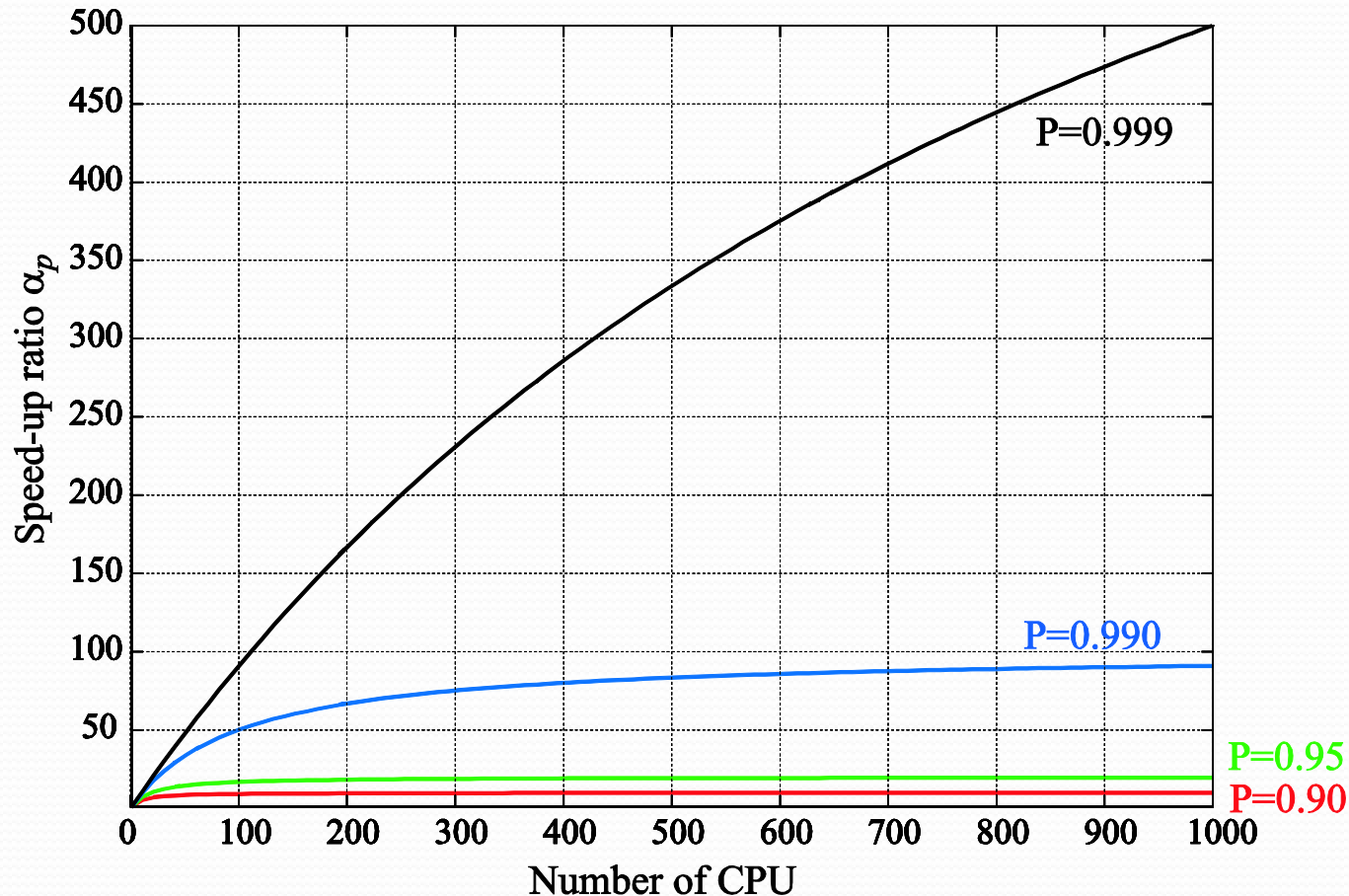
Example of Amdahl's law(1)

Caution! The communication time is not included.



Ordinary calculations using PC cluster

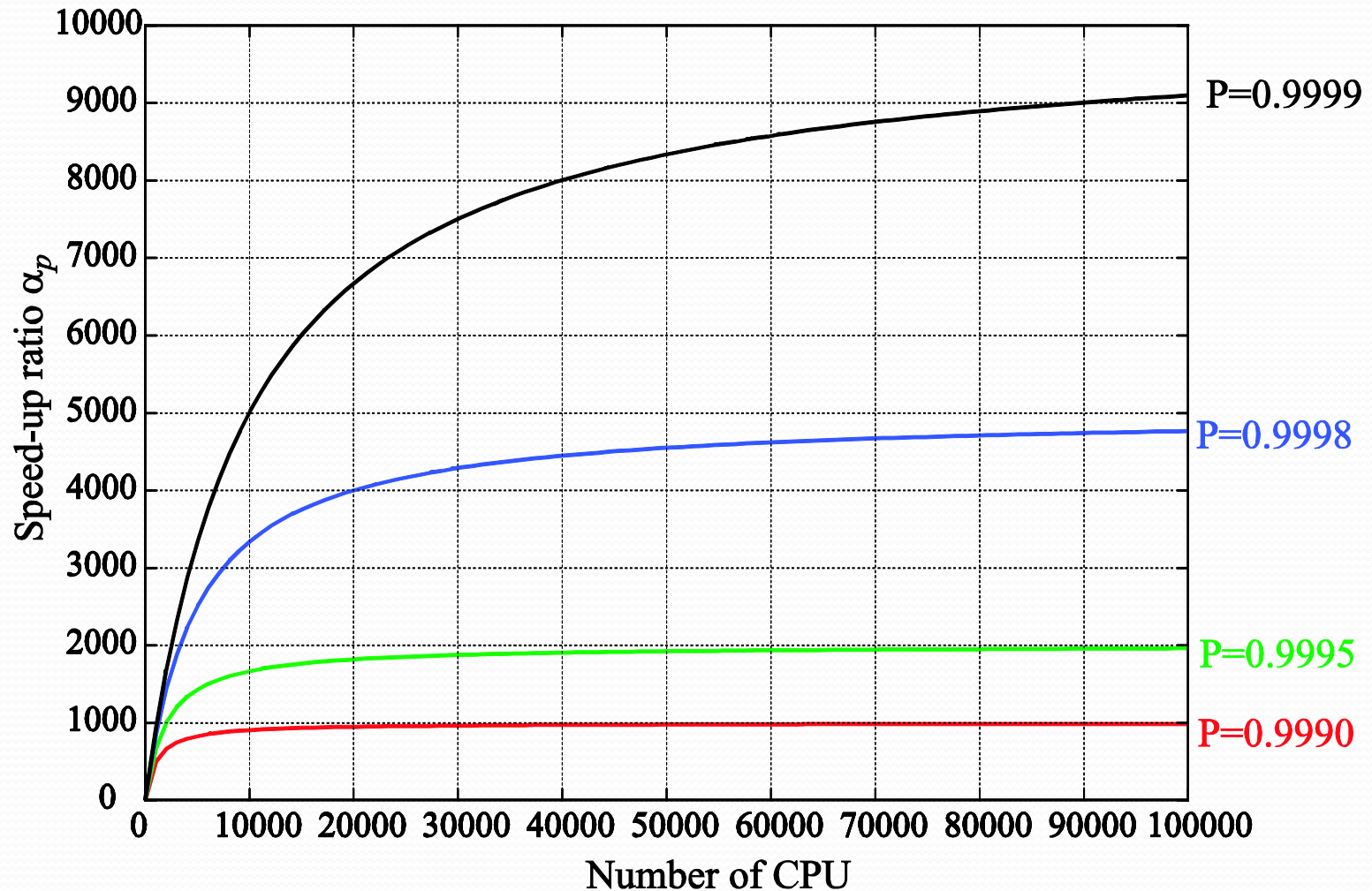
Example of Amdahl's law(2)



Calculations using high-end PC cluster or supercomputers

The maximum number of cores available in the supercomputer queue is about 1,000.

Example of Amdahl's law(3)

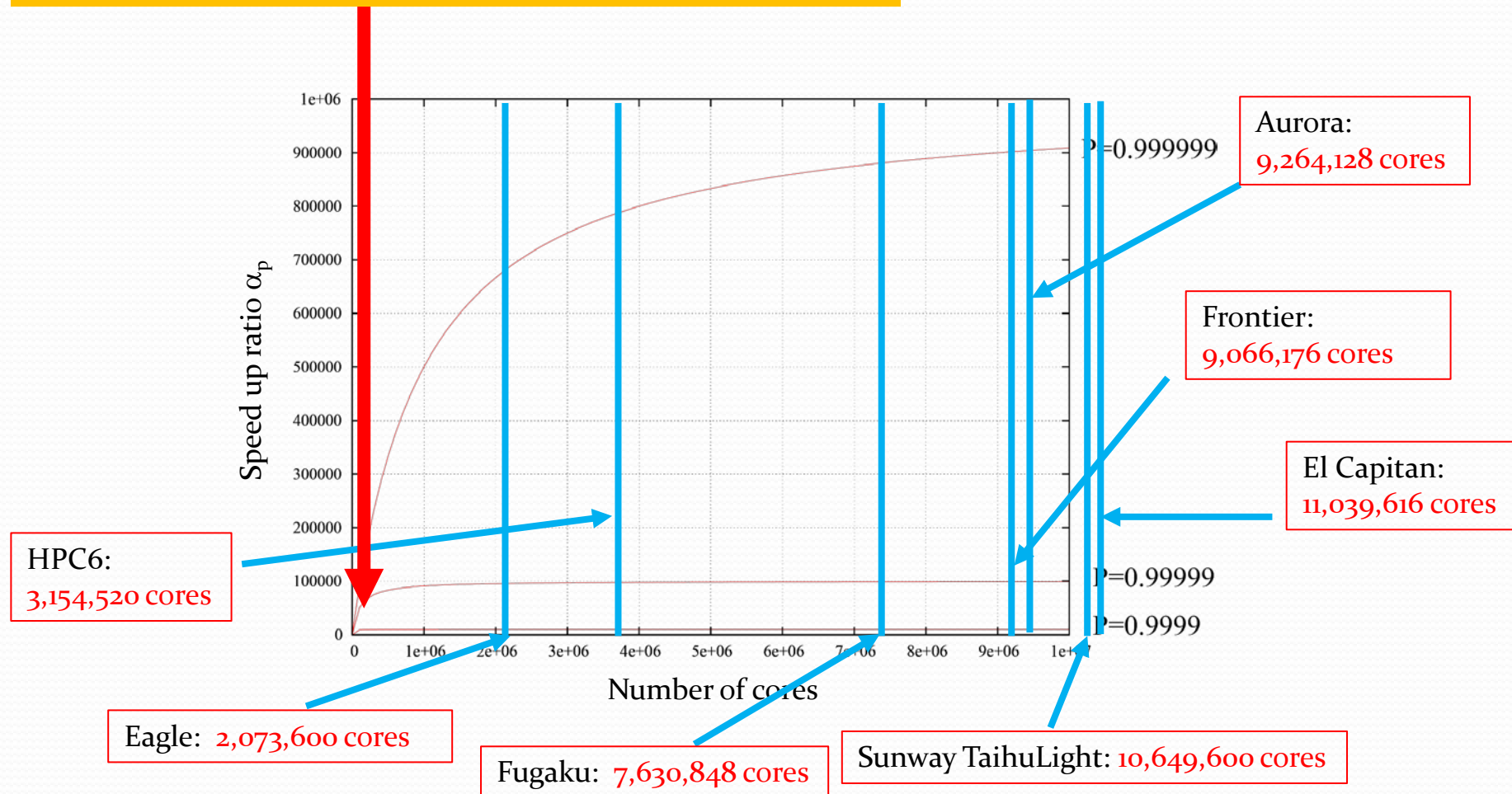


Fugaku: : 7,630,848 cores

Calculations using high-end supercomputers

Example of Amdahl's law(4)

Perfect parallel efficiency is extremely rare and too costly to achieve. It is more practical to aim for reasonable efficiency and focus on collecting data.



Differences in policy between K and the Fugaku

- With the K computer, software had to be tuned for a high degree of parallelism (this was also a requirement for use).
 - Parameter parallelism was not allowed.
- In the Fugaku project, the use of parameter parallelism is encouraged, reflecting the growing importance of data science.
- Development costs for achieving very high parallelization efficiency are substantial and provide limited benefits; therefore, excessive pursuit is not recommended.
 - When data communication time between nodes becomes too large, the benefits of increasing the number of parallel processes are lost.

Even if you are just a user,

- You need to be aware of the parallelization efficiency of the software you use, since you must make effective use of machine time (machine points or usage fees).
- You need to understand the characteristics of the computer and make efforts to improve computational efficiency.
- You need to choose the best compile options to generate efficient executable files.
e.g., for Intel compiler: -xHost, -axCORE-AVX2, ...
- You need to choose the best parameters to achieve the highest performance of the software and to obtain reliable scientific results.

e.g., in DFT calculations: cutoff energy, k-point sampling, model size,...

Parallel programming models

- MPI(message passing interface)
 - The most versatile method.
 - Supports both core-to-core and node-to-node communication.
 - Programming can be complex.

CPU
- OpenMP
 - Limited to communication within shared memory (core-to-core).
 - Programming is generally easier than MPI.

CPU/GPU
- OpenACC (primarily for NVIDIA GPUs, easier to use for non-complex cases)
- CUDA (NVIDIA's native GPU programming model)

GPU
- Kokkos (A C++ library designed to enable efficient and portable parallel programming across diverse hardware architectures, including CPUs, GPUs, and other accelerators.)

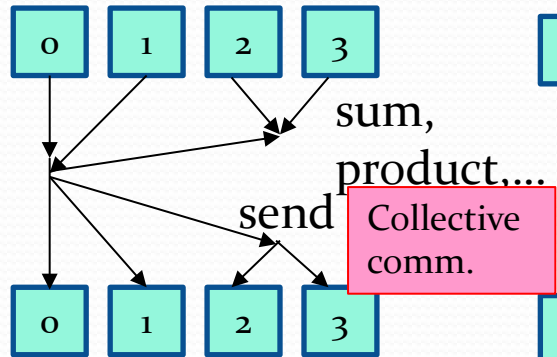
Ideally, users should be able to write parallel programs without considering hardware parallelism, but this remains a major challenge.

Message Passing Interface(MPI)

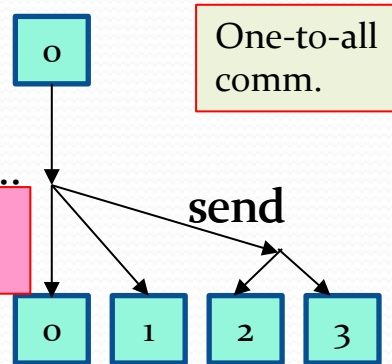
- A standardized and portable message-passing standard for parallel computing architectures.
- **MPI enables communication between nodes and cores**, making it applicable to both distributed and shared memory systems.
- Typical HPC programs are parallelized primarily using MPI.
- In practice, only about 10 types of functions are commonly used.
- Programs often need to be fundamentally rewritten to take advantage of MPI.
- The format, size, destination array, and other details of the data to be communicated must be explicitly specified.

Data Behavior in typical MPI function

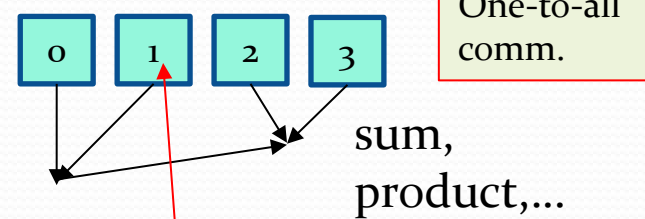
MPI_Allreduce



MPI_Bcast

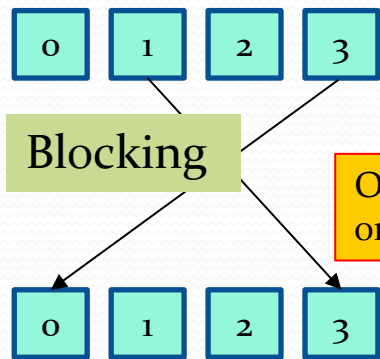


MPI_Reduce



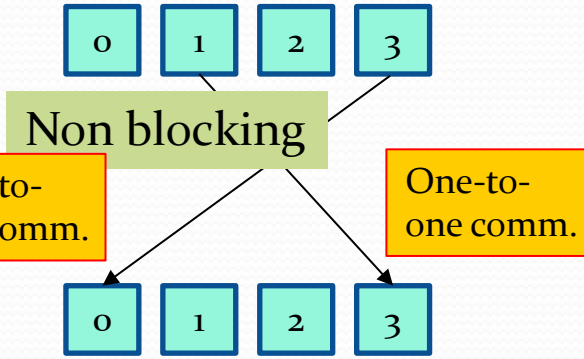
Rank: an identifier number of process in each MPI process

MPI_Send



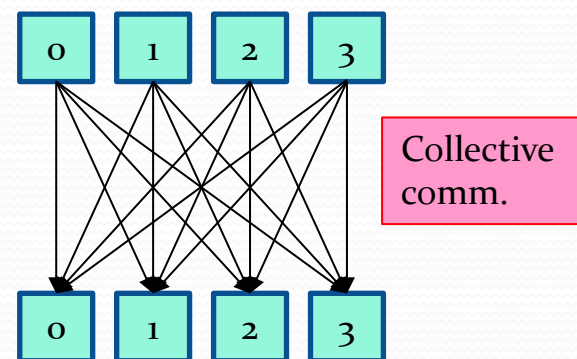
MPI_Recv

MPI_Isend



MPI_Irecv

MPI_Alltoall



Ex. FFT

The difference between blocking and non-blocking communication lies in whether the operations are synchronized or not.

Example of parallelization with MPI

```
program main
include (mpif.h)
```

The program must be rewritten so each node calculates only its share and exchanges the necessary data.

```
...
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
```

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

```
...
a(i)=a(i)+value
enddo
```

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

```
call MPI_ALLREDUCE(a,a1,n_element,
& MPI_DOUBLE_PRECISION,
& MPI_SUM, MPI_COMM_WORLD, IERR)
```

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()).

```
...
call MPI_FINALIZE(IERR)
```

```
...
end
```

Compiler: mpif90, mpicc, mpicxx (or **mpiifx**, **mpicx**, **mpicpx** (Intel), ...)

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

OpenMP

- An application programming interface (API) for **shared-memory** multiprocessing in Fortran, C, and C++.
- Since it can only be used for intra-node communication, it cannot be applied to massively parallel computing.
- As it works by inserting directives into the program, there is generally no need to rewrite the code.
- Compiler options such as `-openmp` (depending on the compiler) must be specified. Otherwise, the directives will be ignored.

Example of parallelization with OpenMP

```
program main
implicit none
integer omp_get_thread_num,i
double precision z(100), a, x(100), b

...
```

For simple parallelization, we only need to insert directives. If the code is compiled without parallelization options, the directives are ignored, and there is no need to modify the original code.

```
!$omp parallel do private(i) ← Directive of the start of parallel
do i= 1, 100                    processing
    z(i) = a * x(i) + b
end do
!$omp end parallel do ← Directive of the end of parallel
                        processing
.....
```

```
end program main
```

Execution: We do not use a special command.
Set environmental value
“OMP_NUM_THREADS=4” (# of parallel) . In the bash environment,
\$ export OMP_NUM_THREADS=4
\$./a.out

Even if more cores are available, good performance is typically achieved with 8 or 16 cores, but not more. Although Fugaku consists of 48 cores per node, using all 48 cores in OpenMP parallel computing often results in the worst performance. (See <https://www.r-ccs.riken.jp/outreach/schools/tokurona-2021/> 5th lecture of Prof. Katagiri.) PyTorch scales well with OpenMP.

Differences between MPI and OpenMP and their relationship to GPUs

- OpenMP can be parallelized **only across cores that share the same memory**. Therefore, large-scale parallelization cannot be achieved using OpenMP alone.
- Typical large-scale software codes are primarily parallelized using MPI.
- The role of OpenMP is to accelerate the parts of the code that are parallelized with MPI.
- Even if you are not developing the software code yourself, you still need to understand its details in order to achieve maximum performance.
- OpenMP supports GPUs. **In particular, Intel and AMD GPUs** are recommended to use either their own proprietary methods or OpenMP
- While MPI itself does not **directly support** GPUs, technologies such as CUDA-Aware MPI and GPUDirect can be used in conjunction with GPUs for efficient parallel computing.

The environment to develop parallel computing codes using (NVIDIA) GPU

easy

- CUFFT, CUBLAS,...

- These are libraries used to accelerate specific parts of a program.
- Only the targeted routines are accelerated; the rest of the code is not.

- OpenACC

- A programming standard for parallel computing developed by Cray, CAPS, NVIDIA, and PGI, designed to simplify heterogeneous CPU/GPU programming.
- OpenACC is primarily supported on NVIDIA GPUs. Support for Intel and AMD GPUs is currently limited or almost non-existent.
- Very similar to OpenMP. In the near future, OpenMP and OpenACC may be merged.
- Supported programming languages: **Fortran** and **C**.

- CUDA

- A parallel computing platform and programming model created by NVIDIA.
- It allows direct programming of NVIDIA GPUs and provides fine-grained control over GPU parallelism.

hard

Example of parallelization with OpenACC

```
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
```

Best results can be achieved by following the compiler's messages and feedback.

<https://www.olcf.ornl.gov/wp-content/uploads/2012/08/OpenACC.pdf>

<http://www.nvidia.co.jp/object/openacc-gpu-directives-jp.html>

<http://www.cms-initiative.jp/ja/events/2014-haishin> (Given by Dr. A. Naruse.)

<https://www.r-ccs.riken.jp/outreach/schools/20220407-1/> (Given by Dr. A. Naruse.)

Hybrid parallelization

- Hybrid parallel execution can be performed with
 - MPI + OpenMP (MPI (nodes), OpenMP (cores))
 - MPI + OpenACC/CUDA (CPU and GPU hybrid)
 - MPI + X (new architecture?)
- The parallelization of MPI only is called Flat (or Pure) MPI parallelization.
- Much of the well-known software is parallelized in such a scheme.
- Kokkos allows users to perform parallel programming without having to consider hardware parallelism details. The development cost is likely to be reduced.
- Using GPUs has become important, but not all software will be faster.

Kokkos Overview

What is Kokkos?

- C++ Performance Portability Library for HPC
- Single source code runs on **CPU, GPU, and future architectures**
- Unified model for parallel execution & memory management

Key Benefits

- Simple `parallel_for` / `parallel_reduce` abstraction
- `Kokkos::View` manages host/device memory seamlessly
- Supports CUDA, HIP, OpenMP, SYCL backends
- Improves portability & maintainability

```
#include <Kokkos_Core.hpp>
int main(int argc, char* argv[]) {
    Kokkos::initialize(argc, argv);
    {
        const int N = 1000000;
        Kokkos::View<double*> a("a", N);

        Kokkos::parallel_for("Init", N, KOKKOS_LAMBDA(int i) {
            a(i) = i * 0.001;
        });

        double sum = 0.0;
        Kokkos::parallel_reduce("Sum", N,
            KOKKOS_LAMBDA(int i, double& local_sum) {
                local_sum += a(i);
            }, sum);
    }
    Kokkos::finalize();
}
```

👉 In short: Write one C++ code that runs efficiently on both CPUs and GPUs.

“Kokkos is designed not for peak, hand-tuned performance, but for performance portability: saving developer time while still achieving high efficiency across platforms.”

<https://www.r-ccs.riken.jp/outreach/schools/20250410-0724/>

(Dr. Nitadori, Introduction to Kokkos)

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
- ...

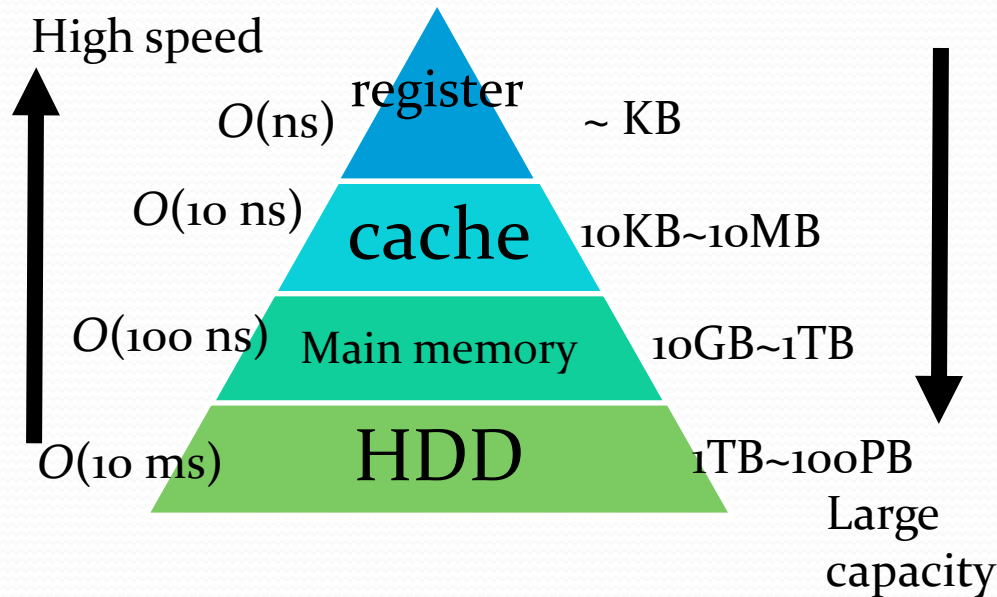
We need to know details of hardware to make highly optimized software codes.

Please see these sites;
(only Japanese)

<https://www.r-ccs.riken.jp/outreach/schools/20250410-0724/>

<https://www.r-ccs.riken.jp/outreach/schools/20240411-0725/>

Hierarchical memory



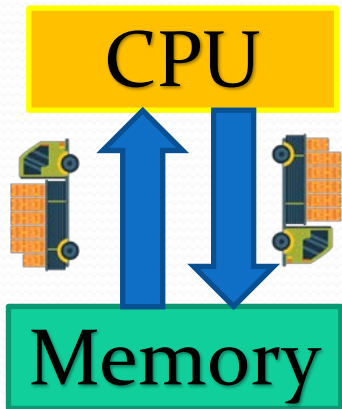
CAUTION!!!

- If memory exceeds physical capacity → system uses HDD swap → extremely slow!
- Common mistake for beginners

- To perform a computation, data must be fetched from memory.
- However, the development of memory bandwidth has not kept pace with the growth of computing power — this is known as the *memory wall problem*.
- To mitigate this, memory is organized in a hierarchical structure.
- Modern CPUs include multiple levels of cache, such as L1, L2, and L3 caches.
- The access speed of registers and caches is much faster than that of main memory, but their capacity is limited.
- For example, accessing data from the L1 cache takes on the order of 10 ns, whereas accessing data from main memory takes about 100 ns. This means that if cache usage is not considered, computations can be up to 10 times slower.

Memory wall problem(1)

Old computer

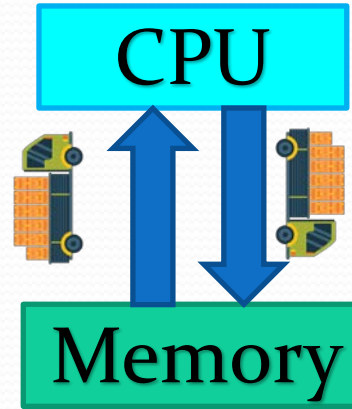


Early CPUs ran at ~10 MHz (slow by today's standards).

Memory access is delayed due to latency.

In the past, both memory-to-CPU transfer speed and computation speed were slow, so data transfer was not a bottleneck (data transfer speed \approx computation speed).

Today's computer



Computation speed: ~a few GHz (today's CPUs)

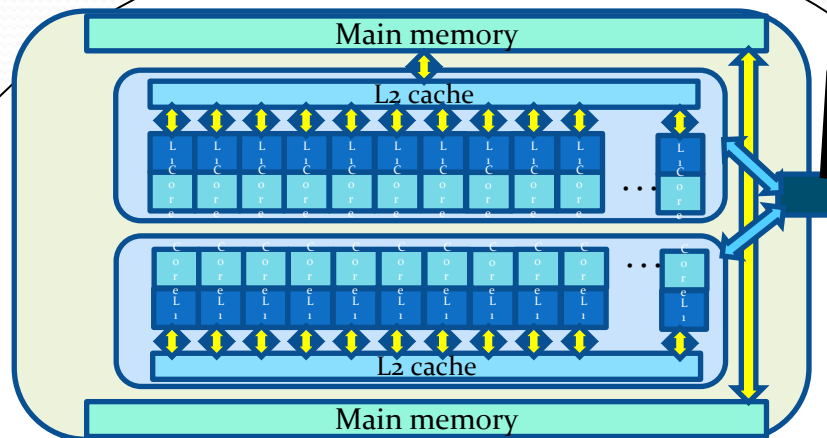
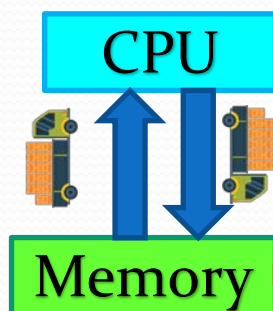
Memory access still involves a fixed latency, which has not improved significantly.

Data transfer from memory to the CPU is slow, while computation speed has become extremely fast. As a result, the CPU spends **a long time idling ('transfer speed \ll compute speed')**, partly mitigated by cache.

Memory wall problem(2)

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

Today's computer



Without understanding CPU architecture, program performance cannot be optimized.

Accessing data from main memory takes about 100 times longer than accessing data from registers.

After data are loaded from memory to the CPU, programs aim to reuse them and minimize memory accesses. Registers and caches are fast, but limited in capacity.

High performance Python and Julia

- In place of Fortran and C/C++, which have long been used in scientific computing, HPC software development in the USA has advanced significantly by adopting Python.
- Python offers outstanding code readability compared to Fortran and C/C++, making maintenance easier and less dependent on domain experts.
- We do not need to write everything in Python. If another language performs better for certain parts, we can keep those parts in the original language.
- By combining Python with other languages, we can improve performance while maintaining readability. (Python is often called a *glue language* because it connects programs written in different languages.)
- **Julia** is a high-level, general-purpose dynamic programming language. Its design makes it particularly suitable for numerical analysis and computational science.



Today's supercomputers

Classes of parallel computers

- Multicore computing MPI OpenMP
 - A processor that integrates multiple execution units (“cores”) on the same chip.
- Symmetric multiprocessing MPI OpenMP
 - A computer system with multiple identical processors that **share memory** and are connected via a common bus.
- Distributed computing MPI
 - A distributed-memory computer system in which the processing elements are connected through a network. (e.g., **cluster computing, massively parallel processing, grid computing**)
- Specialized parallel computers OpenACC OpenMP CUDA
 - Within parallel computing, there are specialized devices that serve niche applications. (e.g., **GPGPUs, application-specific integrated circuits (ASICs), vector processors**)

In the past, ‘many-core’ and ‘multi-core’ were distinguished by the large difference in the number of cores, but now the distinction has almost disappeared.

The latest top 500 supercomputer list (2025.6)(Latest)

<https://www.top500.org/lists/top500/list/2025/6/>

Rank	System	Cores	Rmax [PFlop/s]	Rpeak [PFlop/s]	Power [kW]
1	El Capitan - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/Argonne National Laboratory United States	11,039,616	1,742.00	2,746.38	29,581
2	Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607
3	Aurora - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
4	JUPITER Booster - BullSequana XH2000, GH Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, RedHat Enterprise Linux, EVIDEN EuroHPC/FZJ Germany	4,801,344	793.40	930.00	13,088
5	Eagle - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA InfiniBand NDR, Microsoft Azure United States	2,073,600	561.20	846.84	
6	HPC6 - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, RHEL 8.9, HPE Eni S.p.A. Italy	3,143,520	477.90	606.97	8,461
7	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
8	Alps - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Cray OS, HPE Swiss National Supercomputing Centre (CSCS) Switzerland	2,121,600	434.90	574.84	7,124
9	LUMI - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	531.51	7,107
10	Leonardo - BullSequana XH2000, Xeon Platinum 8358 32C 2.2GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 InfiniBand, EVIDEN EuroHPC/CINECA Italy	1,824,768	241.20	306.31	7,494

HPE Cray

GPU

USA

HPE Cray

GPU

USA

HPE Cray

GPU

USA

GPU

GPU

USA

HPE Cray

GPU

Japan

HPE Cray

GPU

HPE Cray

GPU

GPU

NVIDIA and AMD are fiercely competing.

Japanese supercomputers ranked 10th or lower

15th ABCI 3.0 (AIST), 22nd CHIE-3 (softbank), 24th CHIE-2 (softbank), 27th ABCI-Q(AIST), 36th FPT AI

Factory Japan(FPT AI Factory Japan), 37th Miyabi-G(JCAHPC(U

Tokyo and U Tsukuba)), 46th TSUBAME4.0 (Science Tokyo), 49th

SAKURAONE(SAKURA Internet),

73rd Wisteria /BDEC-01(Odyssey,

93rd AOBA-S(Tohoku U), 96th TOKI-SORA (JAXA)

Highlights from the List (1)2025.6

Highlights from the List

- A total of 237 systems on the list are using accelerator/co-processor technology, up from 210 six months ago. 82 of these use 18 chips, 68 use NVIDIA Ampere, and 27 systems with NVIDIA Volta.

	Count	System Share (%)	Rmax (TFlops)	Rpeak (TFlops)	Cores
1 NVIDIA A100	24	4.8	322,630	485,605	3,273,680
2 NVIDIA H100 SXM5 80GB	24	4.8	789,678	1,160,905	2,964,912
3 NVIDIA H100	16	3.2	936,113	1,425,210	3,179,696
4 NVIDIA Tesla V100	16	3.2	119,452	176,230	2,137,032
5 NVIDIA A100 SXM4 40 GB	16	3.2	180,776	253,436	1,836,152
6 Nvidia H100 SXM5 94Gb	13	2.6	186,828	294,349	766,448
7 AMD Instinct MI250X	13	2.6	2,393,739	3,448,993	16,274,880
8 NVIDIA GH200 Superchip	13	2.6	1,871,704	2,492,377	10,402,368
9 NVIDIA H100 80GB	12	2.4	206,409	320,171	737,424
10 AMD Instinct MI300A	10	2	2,140,937	3,300,102	13,239,576
11 NVIDIA A100 SXM4 80 GB	10	2	70,584	84,814	587,424
12 NVIDIA H200 SXM5 141 GB	8	1.6	462,985	681,883	1,572,016
13 NVIDIA Tesla V100 SXM2	8	1.6	64,025	120,227	1,456,144
14 NVIDIA Tesla A100 40G	7	1.4	56,591	91,126	596,644
15 NVIDIA Tesla A100 80G	6	1.2	114,409	148,289	963,520
16 NVIDIA Tesla P100	5	1	42,903	60,653	839,200
17 NVIDIA Volta GV100	3	0.6	120,839	161,770	1,993,504
18 NVIDIA H200	3	0.6	66,460	90,742	287,744
19 Intel Data Center GPU Max 1550	2	0.4	24,848	66,435	199,872
20 NVIDIA HGX H100	2	0.4	12,911	25,123	85,248
21 NVIDIA A100 80GB	2	0.4	47,940	55,198	369,024
22 NVIDIA Tesla K40	2	0.4	7,154	12,264	145,600
23 AMD Instinct MI210 64 GB	2	0.4	63,207	118,469	580,608
24 NVIDIA H100 PCIe 80GB	2	0.4	8,205	12,641	77,752
25 NVIDIA A100 SXM4 64 GB	2	0.4	244,705	311,293	1,849,824
26 Intel Data Center GPU Max	2	0.4	1,029,191	2,007,963	9,413,888
27 NVIDIA H100 80GB HBM3	1	0.2	4,363	6,545	14,208
28 AMD Instinct MI300X	1	0.2	3,109	5,271	20,096
29 NVIDIA H100 64GB	1	0.2	175,300	249,436	663,040
30 NVIDIA Tesla K80	1	0.2	2,592	3,799	66,000
31 Deep Computing Processor	1	0.2	4,325	6,134	163,840
32 Intel Xeon Phi 5110P	1	0.2	2,539	3,388	194,616
33 NVIDIA Tesla K20x	1	0.2	3,188	4,605	72,000
34 Matrix-2000	1	0.2	61,445	100,679	4,981,760
35 NVIDIA H100/H200	1	0.2	15,050	24,986	34,592
36 NVIDIA A40	1	0.2	4,016	8,665	206,304
37 NVIDIA Tesla K40m	1	0.2	2,478	4,947	64,384

- Intel continues to provide the processors for the largest share (58.80 percent) of TOP500 systems, down from 61.80 % six months ago. 173 (34.60 %) of the systems in the current list used AMD processors, up from 32.40 % six months ago.
- The entry level to the list moved up to the **2.44 Pflop/s** mark on the Linpack benchmark.
- The last system on the newest list was listed at position **456** in the previous TOP500.
- Total combined performance of all 500 exceeded the Exaflop barrier with now **13.84 exaflop/s** (Eflop/s) up from **11.72 exaflop/s** (Eflop/s) 6 months ago.
- The entry point for the TOP100 increased to **16.59 Pflop/s**.

Highlights from the List (2)2025.6

2024年1月時点の世界の国内総生産（GDP）ランキングは、次のとおりです。

- 1位：アメリカ（25兆4627億ドル）
- 2位：中国（17兆8863億ドル）
- 3位：ドイツ（4兆857億ドル）
- 4位：日本（4兆2375億ドル）
- 5位：インド（3兆3897億ドル）
- 6位：イギリス（3兆819億ドル）
- 7位：フランス（2兆7801億ドル）

The GDP ranking matches the TOP500 ranking. However, Germany does not produce CPUs. As for Chinese-made CPUs, only the former No. 1 machine remains within the top 100, and machines with new CPUs are ranked below 200.

General Trends

Installations by countries/regions:

	Count	System Share (%)	Rmax (TFlops)	Rpeak (TFlops)	Cores
1 United States	175	35	6,698,605	10,642,950	58,129,200
2 China	47	9.4	281,270	471,807	18,920,568
3 Germany	41	8.2	1,185,267	1,519,299	8,943,756
4 Japan	39	7.8	1,229,010	1,631,301	12,787,024
5 France	25	5	327,544	508,630	4,676,480
6 Italy	17	3.4	874,884	1,128,394	7,730,480
7 South Korea	15	3	323,115	442,277	2,799,412
8 Canada	13	2.6	78,990	138,730	1,051,472
9 United Kingdom	13	2.6	394,528	540,015	3,152,784
10 Brazil	9	1.8	67,912	122,168	874,720
11 Norway	9	1.8	44,075	62,066	642,528
12 Sweden	9	1.8	69,450	98,506	491,648
13 Taiwan	8	1.6	116,601	170,134	753,216
14 Poland	7	1.4	63,413	101,087	401,120
15 Netherlands	7	1.4	291,308	489,421	1,403,296
16 Russia	6	1.2	71,457	98,726	721,488
17 Saudi Arabia	6	1.2	93,713	149,854	2,435,392
18 India	6	1.2	29,412	38,541	785,400
19 Singapore	5	1	38,288	62,498	315,424
20 Australia	4	0.8	55,227	73,106	498,032
21 United Arab Emirates	4	0.8	122,281	201,731	816,800
22 Switzerland	4	0.8	471,169	628,265	2,663,712
23 Czechia	3	0.6	17,991	21,785	360,192
24 Spain	3	0.6	221,873	306,103	1,542,016
25 Finland	3	0.6	391,388	546,193	3,116,992
26 Thailand	2	0.4	21,996	35,450	134,656
27 Ireland	2	0.4	6,697	13,160	152,320
28 Slovenia	2	0.4	6,918	10,047	156,480
29 Turkey	2	0.4	7,366	10,101	69,760
30 Bulgaria	2	0.4	7,045	9,154	164,224
31 Austria	2	0.4	34,586	57,518	199,200
32 Israel	1	0.2	41,500	52,679	74,880
33 Denmark	1	0.2	66,590	100,630	223,088
34 Iceland	1	0.2	10,530	17,015	36,208
35 Morocco	1	0.2	3,158	5,015	71,232
36 Luxembourg	1	0.2	10,520	15,288	99,200
37 Belgium	1	0.2	2,776	3,967	23,200
38 Portugal	1	0.2	3,956	5,014	78,336
39 Argentina	1	0.2	5,390	12,583	43,008
40 Hungary	1	0.2	3,105	4,508	27,776

Highlights from the List (3)2025.6

HPC manufacturer:

	Count	System Share (%)	Rmax (TFlops)	Rpeak (TFlops)	Cores
1 Lenovo	136	27.2	713,299	1,202,393	14,072,636
2 HPE	132	26.4	6,626,535	9,844,161	51,398,848
3 EVIDEN	55	11	1,733,742	2,302,186	15,902,136
4 DELL	41	8.2	353,816	560,317	4,339,744
5 Nvidia	27	5.4	758,695	1,113,657	3,645,104
6 Fujitsu	17	3.4	681,177	853,739	10,090,888
7 NEC	14	2.8	90,975	125,879	847,688
8 Microsoft Azure	8	1.6	757,160	1,127,999	3,540,480
9 Penguin Computing, Inc.	7	1.4	23,425	38,351	775,672
10 MEGWARE	7	1.4	76,141	119,349	444,656
11 Inspur	5	1	13,638	24,842	237,216
12 Supermicro	4	0.8	123,770	190,324	938,608
13 ASUSTeK / ASUS Cloud / Taiwan Web Service Corporation	3	0.6	22,716	31,587	122,064
14 IBM	3	0.6	29,387	40,663	493,024
15 Sugon	3	0.6	9,307	14,368	262,784
16 ACTION	2	0.4	5,979	10,730	135,168
17 HPE, GALAXY	2	0.4	19,802	33,334	80,128
18 xFusion	2	0.4	5,617	7,609	62,672
19 Liqid	2	0.4	5,393	7,518	102,144
20 IBM / NVIDIA / Mellanox	2	0.4	112,840	148,759	1,860,768
21 YANDEX, NVIDIA	2	0.4	37,550	50,051	328,352
22 Nebius AI	2	0.4	248,940	425,282	937,728
23 ASUSTeK	2	0.4	92,470	136,785	285,600
24 Self-made	1	0.2	3,307	4,897	60,512
25 Format sp. z o.o.	1	0.2	5,051	7,709	47,616
26 T-Platforms	1	0.2	2,478	4,947	64,384
27 NEC / DELL	1	0.2	6,708	7,627	125,440
28 Intel	1	0.2	1,012,000	1,980,006	9,264,128
29 Nscale	1	0.2	7,401	12,380	66,528
30 ClusterVision / Hammer	1	0.2	2,969	4,335	64,512
31 Lenovo/IBM	1	0.2	2,814	3,578	86,016
32 Quanta Computer / Taiwan Fixed Network / ASUS Cloud	1	0.2	9,000	15,208	170,352
33 Amazon Web Services	1	0.2	9,950	15,107	172,692
34 MEGWARE / Supermicro	1	0.2	9,087	19,344	96,768
35 NRCP	1	0.2	93,015	125,436	10,649,600
36 Netweb Technologies	1	0.2	8,500	13,170	81,344
37 Huawei Technologies Co., Ltd.	1	0.2	3,534	5,880	63,360
38 Koi Computers	1	0.2	4,856	6,107	107,648
39 NVIDIA, Inspur	1	0.2	12,810	20,029	130,944
40 Alipa Technology	1	0.2	2,539	3,388	194,616
41 eSlim Korea	1	0.2	15,940	17,815	115,072
42 NUDT	1	0.2	61,445	100,679	4,981,760
43 Fujitsu / Lenovo	1	0.2	9,264	15,142	204,032
44 ParTec/EVIDEN	1	0.2	4,504	5,132	19,584

Interconnect Technologies:

	Count	System Share (%)	Rmax (TFlops)	Rpeak (TFlops)	Cores
1 Infiniband	271	54.2	5,576,009	7,976,777	43,262,416
2 Gigabit Ethernet	164	32.8	7,224,400	11,272,703	61,736,268
3 Omnipath	33	6.6	175,546	251,438	3,833,596
4 Custom Interconnect	21	4.2	271,786	388,803	19,515,736
5 Proprietary Network	8	1.6	530,839	641,021	9,105,408
6 Ethernet	3	0.6	58,951	82,417	253,536

Processor Technologies:

	Count	System Share (%)	Rmax (TFlops)	Rpeak (TFlops)	Cores
1 Intel Cascade lake	88	17.6	375,457	679,419	8,589,184
2 Intel Sapphire Rapids	79	15.8	3,455,733	5,644,045	20,999,464
3 AMD Zen-3 (Milan)	71	14.2	2,804,208	4,048,496	25,466,576
4 AMD Zen-2 (Rome)	58	11.6	675,088	945,194	10,691,984
5 Intel Skylake	44	8.8	200,027	314,458	3,956,644
6 AMD Zen-4 (Genoa)	38	7.6	2,531,981	3,916,261	17,044,328
7 Intel Ice Lake	33	6.6	418,408	601,647	4,930,172
8 Intel Broadwell	17	3.4	61,314	74,758	1,912,180
9 NVIDIA Grace CPU	13	2.6	1,871,704	2,492,377	10,402,368
10 Intel Emerald Rapids	10	2	359,483	467,788	1,323,168
11 Fujitsu ARM	9	1.8	534,795	646,035	9,183,744
12 Intel Haswell	9	1.8	47,788	65,588	1,381,020
13 Intel IvyBridge	6	1.2	80,882	129,536	5,666,452
14 NEC Vector Engine	5	1	57,893	72,429	239,168
15 AMD Zen-5 (Turin)	5	1	35,599	48,086	530,672
16 Intel Xeon Phi	5	1	46,554	84,556	1,901,172
17 Power	4	0.8	139,039	184,817	2,281,792
18 Intel Granite Rapids	2	0.4	10,608	18,545	47,616
19 AMD Instinct	1	0.2	31,096	44,165	151,200
20 X86_64	1	0.2	4,325	6,134	163,840
21 ShenWei	1	0.2	93,015	125,436	10,649,600

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.

<https://www.top500.org/project/linpack/>

LINPACK

- **LINPACK** is a software library for performing numerical linear algebra on digital computers.
- It is widely used to assess the performance of supercomputers. In practice, the **High-Performance Linpack (HPL)** benchmark is employed.
- HPL relies on the **BLAS libraries** to perform basic vector and matrix operations.
- The LINPACK benchmark measures performance by solving a **dense system of linear equations**.

Only one aspect of supercomputer performance

<https://www.top500.org/project/linpack/>

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:

LINPACK: Dense matrices

- 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.

<https://www.top500.org/lists/hpcg/2017/11/>

HPCG ranking(2025.6)Latest

<https://www.top500.org/lists/hpcg/2025/6/>

HPE Cray

USA GPU

HPE Cray

Japan CPU

HPE Cray

USA GPU

HPE Cray

USA GPU

HPE Cray

GPU

GPU

HPE Cray

GPU

GPU

HPE Cray

USA GPU

USA GPU

Rank	TOP500 Rank	System	Cores	Rmax (PFlop/s)	HPCG (TFlop/s)
1	1	El Capitan - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, HPE DOE/NNSA/LLNL United States	11,039,616	1,742.00	17406.90
2	7	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	16004.50
3	2	Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	14054.00
4	3	Aurora - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	5612.60
5	9	LUMI - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	4586.95
6	8	Alps - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Cray OS, HPE Swiss National Supercomputing Centre (CSCS) Switzerland	2,121,600	434.90	3671.32
7	10	Leonardo - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA H800 InfiniBand, EVIDEN EuroHPC/CINECA Italy	1,824,768	241.20	3113.94
8	15	ABCI 3.0 - HPE Cray XD670, Xeon Platinum 8558 48C 2.1GHz, NVIDIA H200 SXM5 141 GB, Infiniband NDR200, Rocky Linux 9.4, HPE National Institute of Advanced Industrial Science and Technology (AIST) Japan	479,232	145.10	2445.67
9	25	Perlmutter - HPE Cray EX 235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-11, HPE DOE/SC/LBNL/NERSC United States	888,832	79.23	1905.00
10	20	Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94.64	1795.67

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.

<https://www.top500.org/lists/green500/2013/06/>

Green 500(2025.6)(Latest)

Green500 Data

Rank	TOP500 Rank	System	Cores	Rmax [PFlop/s]	Power [kW]	Energy Efficiency (GFlops/watts)
1	259	JEDI - BullSequana XH3000, Grace Hopper Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, ParTec/EVIDEN EuroHPC/FZJ Germany	19,584	4.50	67	72.733
2	148	ROMEO-2025 - BullSequana XH3000, Grace Hopper Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, Red Hat Enterprise Linux, EVIDEN ROMEO HPC Center - Champagne-Ardenne France	47,328	9.86	160	70.912
3	484	Adastra 2 - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, RH-EL, HPE Grand Equipement National de Calcul Intensif - Centre Informatique National de l'Enseignement Supérieur (GENCI-CINES) France	16,128	2.53	37	69.098
4	183	Isambard-AI phase 1 - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE University of Bristol United Kingdom	34,272	7.42	117	68.035
5	255	Onus (GPU only) - ThinkSystem SD645-N V3, AMD EPYC 9655 96C 2.6GHz, NVIDIA H100 50x5 80GB, InfiniBand NDR, Rocky Linux 9.4, Lenovo Universität Paderborn - PC2 Germany	19,440	4.66		68.177
6	66	Capella - Lenovo ThinkSystem SD645-N V3, AMD EPYC 9334 32C 2.7GHz, Nvidia H100 50x5 94Gb, InfiniBand NDR200, AlmaLinux 9.4, MEGWARE TU Dresden, ZIH Germany	85,248	24.06	445	68.053
7	304	SSC-24 Energy Module - HPE Cray XD470, Xeon Gold 6430 32C 2.1GHz, NVIDIA H100 50x5 80GB, InfiniBand NDR400, RH-EL 9.2, HPE Samsung Electronics South Korea	11,200	3.82	69	67.251
8	85	Helios GPU - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Dyfronet Poland	89,760	19.14	317	66.948
9	399	AMD Ouranos - BullSequana XH3000, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, InfiniBand NDR200, Red Hat Enterprise Linux, EVIDEN Aeos France	16,632	2.99	48	66.464
10	412	Hemri - ThinkSystem SR670 V2, Intel Xeon Platinum 8362 32C 2.8GHz, NVIDIA H100 80GB PCIe, InfiniBand HDR, Lenovo Flatiron Institute United States	8,288	2.88	44	65.396

<https://www.top500.org/lists/green500/2025/6/>

Germany

France

France

Germany

Germany

France

USA

This list changes dramatically every time. The focus is on technological development.

Graph 500(2025.6)(Latest)

<https://graph500.org/>

Top Ten from June 2025 BFS

RANK	MACHINE	VENDOR	INSTALLATION SITE	LOCATION	COUNTRY	YEAR	NUMBER OF NODES	NUMBER OF CORES	SCALE	GTEPS
1	Supercomputer Fugaku	Fujitsu	RIKEN Center for Computational Science (R-CCS)	Kobe Hyogo	Japan	2020	152064	7299072	43	204068
2	datacentre in Dallas Texas									
3	Wuhan Supercomputer	HUST	Wuhan Supercomputing Center	Wuhan	China	2023	252	6999552	41	115357.6
4	Aurora	Intel/HPE	DOE/SC/Argonne National Laboratory	Argonne IL	United States	2023	8192	51183616	42	69373.38
5	EOS NVIDIA DGX SUPERPOD	NVIDIA	NVIDIA Corporation	Santa Clara	United States	2023	416	485888	38	39085.4
6	Frontier	HPE	DOE/SC/Oak Ridge National Laboratory	Oak Ridge TN	United States	2021	9248	8730112	40	29654.6
7	Pengcheng Cloudbrain-II	HUST-Pengcheng Lab-HUAWEI	Pengcheng Lab	ShenZhen	China	2022	488	93696	40	28463.1
8	Sunway TaihuLight	NRCPC	National Supercomputing Center in Wuxi	Wuxi	China	2015	40768	10599680	40	23755.7
9	Wisteria/BDEC-01 (Odyssey)	Fujitsu	Information Technology Center The University of Tokyo	Kashiwa Chiba	Japan	2021	7680	368640	37	16118
10	MareNostrum 5 ACC	Eviden	Barcelona Supercomputing Center	Barcelona	Spain	2024	1120	680960	35	15737.43

Breadth-First Search (幅優先探索)

Explore the entire graph and determine the level (distance) of each vertex

- Purpose:** Visit all vertices in the graph and record the order of visitation.
- Traversal method:** Start from the source; visit all adjacent vertices at the same level before moving on to the next level.
- Output:** Return each vertex's visit level (distance from the source)—i.e., which vertices were visited and in what order.
- Time complexity:** $O(V+E)O(V+E)O(V+E)$, where V is the number of vertices and E is the number of edges.

<https://graph500.org/>

Top Ten from June 2025 SSSP

RANK	MACHINE	VENDOR	INSTALLATION SITE	LOCATION	COUNTRY	YEAR	NUMBER OF NODES	NUMBER OF CORES	SCALE	GTEPS
1	Wuhan Supercomputer	HUST	Wuhan Supercomputing Center	Wuhan	China	2023	252	6999552	41	15335.9
2	Pengcheng Cloudbrain-II	HUST-Pengcheng Lab-HUAWEI	Pengcheng Lab	ShenZhen	China	2022	488	93696	40	11529.7
3	Supercomputer Fugaku	Fujitsu	RIKEN Center for Computational Science (R-CCS)	Kobe Hyogo	Japan	2020	82944	3981312	39	2126.45
4	Hainan exascale Prototype Upgrade System	National University of Defense Technology	National Supercomputer Center in Tianjin	Tianjin	China	2021	2048	131072	34	2054.35
5	SuperMUC-NG	Lenovo	Leibniz Rechenzentrum	Garching	Germany	2018	4096	196608	37	1053.93
6	NERSC Cori - 1024 haswell partition	Cray	NERSC/LBNL	DOE/SC/LBNL/NERSC	United States	2017	1024	32768	36	558.833
7	Nurion	Cray	Korea Institute of Science and Technology Information	Daejeon	Korea Republic Of	2018	1024	65536	36	337.239
8	NERSC Cori - 512 KNL partition	Cray	NERSC/LBNL	DOE/SC/LBNL/NERSC	United States	2017	512	32768	35	229.188
9	Lise	Atos	Zuse Institute Berlin (ZIB)	Berlin	Germany	2019	1270	121920	38	197.7
10	Undisclosed Cray XE6	Cray	National Computing Facility	University	United States	2013	512	16384	34	134.173

Japan

Japan

Single-Source Shortest Path (単一始点最短経路)

- Purpose:** Find the shortest path from the source to each vertex.
- Traversal method:** Typically, algorithms such as Dijkstra's algorithm or the Bellman-Ford algorithm are used to compute the shortest distance to each vertex.
- Output:** Return the shortest distance and the actual shortest path from the source to each vertex.
- Time complexity:** Depends on the algorithm. For example, Dijkstra's algorithm runs in $O(V^2)O(V^2)O(V^2)$, or $O((V+E)\log V)O((V+E)\log V)$ when using a heap.

Find the shortest distance from the starting point to each vertex.

HPL-MxP (HPL-AI)

HPL-MXP MIXED-PRECISION BENCHMARK

The HPL-MxP benchmark seeks to highlight the emerging convergence of high-performance computing (HPC) and artificial intelligence (AI) workloads. While traditional HPC focused on simulation runs for modeling phenomena in physics, chemistry, biology, and so on, the mathematical models that drive these computations require, for the most part, 64-bit accuracy. On the other hand, the machine learning methods that fuel advances in AI achieve desired results at 32-bit and even lower floating-point precision formats. This lesser demand for accuracy fueled a resurgence of interest in new hardware platforms that deliver a mix of unprecedented performance levels and energy savings to achieve the classification and recognition fidelity afforded by higher-accuracy formats.

HPL-MxP strives to unite these two realms by delivering a blend of modern algorithms and contemporary hardware while simultaneously connecting to the solver formulation of the decades-old HPL framework of benchmarking the largest supercomputing installations in the world. The solver method of choice is a combination of LU factorization and iterative refinement performed afterwards to bring the solution back to 64-bit accuracy. The innovation of HPL-MxP lies in dropping the requirement of 64-bit computation throughout the entire solution process and instead opting for low-precision (likely 16-bit) accuracy for LU, and a sophisticated iteration to recover the accuracy lost in factorization. The iterative method guaranteed to be numerically stable is the generalized minimal residual method (GMRES), which uses application of the L and U factors to serve as a preconditioner. The combination of these algorithms is demonstrably sufficient for high accuracy and may be implemented in a way that takes advantage of the current and upcoming devices for accelerating AI workloads.

HPL-MxP

「TOP500」と「HPCG」では、連立一次方程式を解く計算性能でランクをつけてきました。どちらも科学技術計算や産業利用の中で多く用いられてきた倍精度演算（10進法で16桁の浮動小数点数）のみで計算することがルールに定められていました。近年、GPUや人工知能向けの専用チップで低精度演算（10進法で5桁、もしくは10桁）の演算器を搭載し、高性能化した計算機が多数現れています。これらの高性能演算能力が「TOP500」に反映されないとの実情があり、ジャック・ドンガラ博士を中心にLINPACKベンチマークを改良し低精度演算で解くことを認めた新しいベンチマーク「HPL-AI」（現在「HPL-MxP」）が2019年11月に提唱されました。「HPL-MxP」はLINPACKが連立一次方程式をLU分解^[5]を用いて解く際に低精度計算で実施することを認めています。しかしながら、倍精度計算よりも計算精度が劣ってしまうため、引き続き反復改良^[6]と呼ばれる技術で倍精度計算と同等の精度にすることを求めています。つまり、2段階の計算過程で構成されたベンチマークです。

[HPL-MxP用のベンチマークのプログラム](#) は、ランキングの規則に従って理研計算科学研究センター（R-CCS）が開発したものであり、オープンソース化し公開しています。

https://www.riken.jp/pr/news/2023/20230522_3/index.html

This is a ranking to come, as there are not many machines participating yet.

HPL-MxP(2025.6)(Latest)

<https://hpl-mxp.org/>
<https://hpl-mxp.org/results.md>

June 2025

Rank	Site	Computer	Cores	HPL-MxP (Eflop/s)	TOP500 Rank	HPL Rmax (Eflop/s)	Speedup
1	DOE/SC/LLNL	El Capitan	11,039,616	16.680	1	1.7420	9.6
2	DOE/SC/ANL	Aurora	8,159,232	11.643	3	1.0120	11.5
3	DOE/SC/ORNL	Frontier	8,560,640	11.390	2	1.3530	8.4
4	AIST	ABCI 3.0	479,232	2.363	15	0.1451	16.3
5	EuroHPC/CSC	LUMI	2,752,704	2.350	9	0.3797	6.2
6	RIKEN	Fugaku	7,630,848	2.000	7	0.4420	4.5
7	EuroHPC/CINECA	Leonardo	1,824,768	1.842	10	0.2412	7.6
8	CII, Institute of Science	TSUBAME4	172,800	0.641	46	0.0396	16.2
9	NVIDIA	Selene	555,520	0.630	30	0.0635	9.9
10	DOE/SC/LBNL	Perlmutter	888,832	0.590	25	0.0792	7.4
11	FZJ	JUWELS BM	449,280	0.470	43	0.0441	10.7
12	SAKURA/Prunus	SAKURAONE	11,760	0.340	49	0.0340	10.0
13	GENCI-CINES	Adastral	319,072	0.303	40	0.0461	6.6
14	Pawsey Supercomputing Centre	Setonix - GPU	181,248	0.175	58	0.0272	6.4
15	University of Florida	HiPerGator	138,880	0.170	94	0.0172	9.9
16	SberCloud	Christofari Neo	98,208	0.123	125	0.0120	10.3
17	DOE/SC/ANL	Polaris	256,592	0.114	60	0.0258	4.4
18	ITC	Wisteria	368,640	0.100	73	0.0221	4.5
19	NSC	Berzelius	59,520	0.050	240	0.0053	9.5
20	Cyfronet	Athena	47,616	0.050	248	0.0051	9.9

Japan

Japan

Japan

Japan

Japan

Fugaku is working on 24h/365days

- K computer was operated 24hours/365 days for public use. Fugaku is also being done in the same way.
 - Different from Fugaku, 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 Fugaku is almighty.
- Users could optimize their code for Fugaku easier than for the other ones(in the sense that it did not use a special architecture like GPU).
- However, as Post-Fugaku will be equipped with accelerators, applications must be GPU-enabled at minimum to make use of them.



Future direction of large-scale computing

From the above rankings

- From now on, **both performance and energy efficiency** will be important.
- Energy efficiency = Lower-frequency CPUs → Compensated by parallelization techniques for effective use.
- The trend toward many-core (or multi-core) systems with relatively low-frequency CPUs continues.
 - The number of cores per CPU is increasing, and the distinction between many-core and multi-core has largely disappeared.
- **Following this trend, many supercomputers are adopting GPUs, and their number is growing.**
- **The computers we use on a daily basis will also likely be equipped with GPUs. → *Who will rewrite programs for GPUs?***
- Since GPUs are prominent in AI and machine learning, this architecture is likely to continue being adopted in the future.
- However, it is not clear whether GPU-based supercomputers are effective for all scientific fields.
- It is likely that many supercomputers specifically designed for data analysis and AI will appear in the future.

However,...

- The effective performance measured by LINPACK or HPCG does not necessarily reflect the performance of real scientific applications.
- In practice, researchers invest significant effort to achieve higher performance on supercomputers than on standard PC clusters.
- Fugaku was **co-designed** to deliver better performance for scientific applications, not just for benchmark programs. (At the same time, winning first place in the Top500 may have been a national mission for taxpayers—and this was **achieved**.)
- Such performance optimization requires not only physicists, chemists, and biologists, but also collaboration with computer scientists and numerical mathematicians.
- Fugaku is a general-purpose system and can be less efficient than domain-specific machines (e.g., Anton for molecular dynamics, which is 100–1000 \times faster than general-purpose supercomputers).
- **Should users really be required to perform so much tuning themselves? NVIDIA is actively optimizing GPU software, which clearly contributes to the growing number of GPU users.**

Future direction1

- For the foreseeable future, architectures with computing accelerators such as GPUs will remain mainstream.
- It is difficult to fully utilize all cores, nor is that the goal anymore (unlike during the era of the K computer).
- What is important now is to develop highly efficient hybrid parallelism between CPUs and GPUs (which may yield at most a few-fold speedup).
- The next flagship supercomputer will undoubtedly be equipped with accelerators such as GPUs.
- Software developers themselves must make effective use of GPUs to accelerate their codes.
 - This is not trivial: many well-known software packages overseas are tuned for GPUs by NVIDIA. Can other vendors achieve the same level of support?
- Consistent and systematic research is required to keep up with the latest architectures.
- To make systems more user-friendly, cloud services and GUI-based operations are being promoted (with hardware and software details increasingly concealed).
 - While this is beneficial for users, training developers remains a different and equally critical challenge.

Future direction2

- The United States is rapidly advancing both architectures and programming frameworks, consolidating its position as a global leader. If we do not actively engage, we risk falling behind and losing influence in this critical field.
- It is unrealistic to expect physicists and chemists to fully cover computer science or numerical analysis on their own.
→ We need to establish systems that facilitate collaboration with computational scientists, computer scientists, and numerical analysis experts. This is currently underway, as exemplified by the establishment of *the Computational Materials Science Forum* (<https://cms-forum.jp/>).
- Even now, ensuring numerical precision remains insufficient. For example, there is no universal standard for data summation across cores (threads) in MPI or OpenMP. Such issues must be handled carefully; otherwise, the reliability of calculations may be compromised.

Future direction3

- The expertise accumulated through the K computer and Fugaku can be utilized for the next decade.
- However, expertise in GPU programming and optimization has not yet been fully established.
- The development of supercomputers may reach its limits in the near future (Prof. Hiraki suggested around 2029), due to factors such as enormous power consumption and the end of Moore's law.
- We must explore new ideas to achieve high performance in both software and hardware as soon as possible. A paradigm shift may occur in the near future.
- For the time being, data transfer will remain the most costly factor at all levels.
- Research on quantum computers is also progressing and will become an important focus in the future.

MateriApps

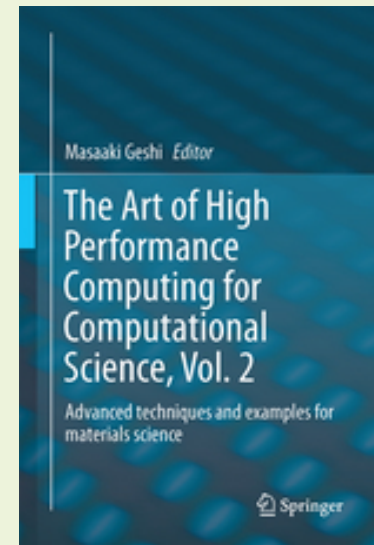
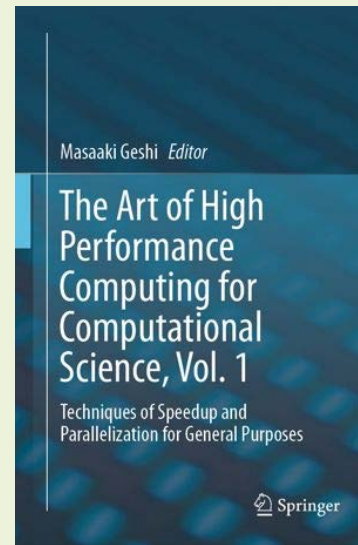
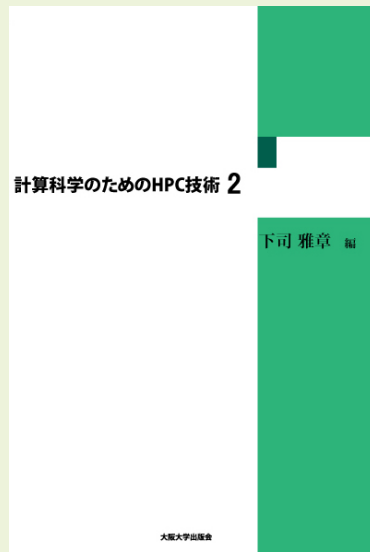
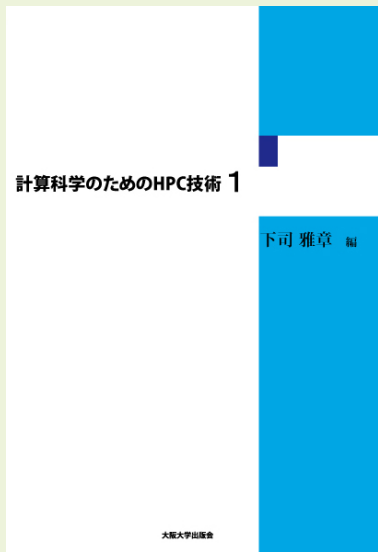
The screenshot shows the MateriApps website in a web browser. The address bar at the top displays the URL <https://ma.issp.u-tokyo.ac.jp/en/>, which is highlighted with a red rectangle. The website header features the MateriApps logo, a navigation menu with links like 'News / Hands-on / Event', 'List of Apps', 'Search Apps', 'Keywords', 'Research Showcase', and 'Concierge', and a search bar. Below the header, a banner promotes 'Try the app without installing 「MateriApps LIVE」' with a 'MORE' button. The main content area is titled 'Search by category' and displays a grid of buttons for various materials science topics: Electronic structure (solid state physics), Electronic structure (quantum chemistry), Molecular dynamics, Visualization/modeling, Strong correlation/effective models, Data analysis/supplementary tool, Continuum models, Database, Integrated Environment, and Machine learning. At the bottom, there is a 'News / Event' section with tabs for 'News' and '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 promote the development of human resources to effectively use HPC machines, including supercomputers.
- The relationship between the Top500 rankings and the performance of the software we actually use is not necessarily clear.
- To maximize the performance of our codes, it is essential to understand the architecture of supercomputers and HPC systems.
- We should cultivate specialists capable of developing advanced software to drive cutting-edge research.
- We should also train users who can effectively utilize scientific software packages to advance their research (through CMD workshops, distance learning, etc.).

Related texts



- 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).

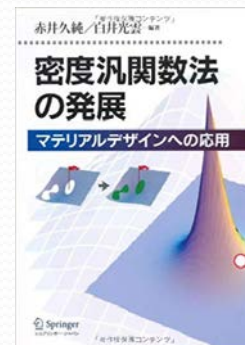
<https://www.r-ccs.riken.jp/outreach/schools/20230413-1/>

<https://www.r-ccs.riken.jp/outreach/schools/20240411-0725/>

- The introductory lecture is given in edX.

<https://www.edx.org/bio/Masaaki-Geshi>

第一原理計算書籍

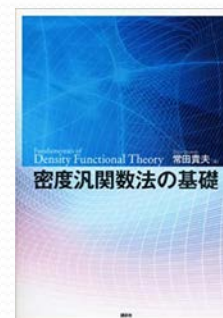
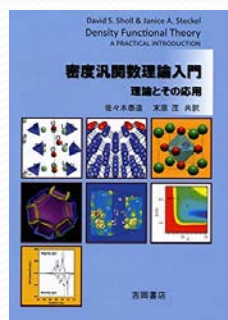
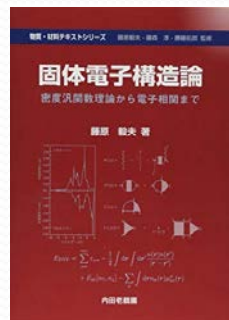


CMDワークショップ
の教科書。

笠井阪大名誉教授
による先端研究事
例の紹介

阪大産研小口教授
による第一原理計算
の初歩的なテキスト

赤井阪大名誉教授代
表の特定領域研究の
成果をまとめたもの。

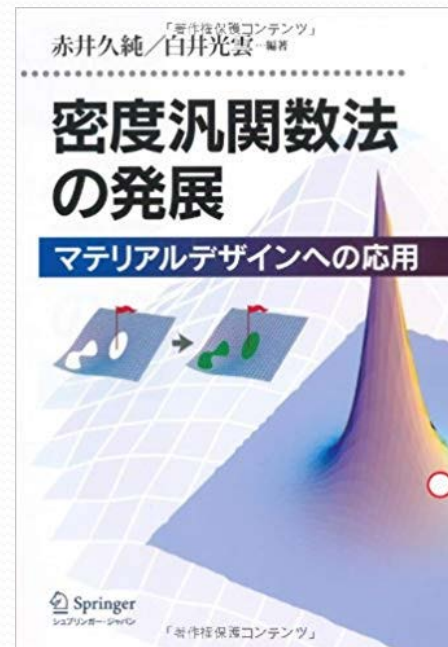


その他のいくつかの教科書。

特におすすめ



CMDワークショップ
でやる内容そのまま
が載せてあり、同じ
数値が再現できる。
定価も2500円(+
税)と400ページを超
えるのに専門書とし
ては格安。



10年以上前のプロ
ジェクトでやってい
たことが、今の標準
になっており、理論
の詳細は置いてお
けば、DFT計算を試
してみたい方の適用
例のガイドブックとし
ても最適。(4500円
+税)(現在は丸善
から販売)