Basic Computational Biology

High Performance Computing Technology(1)

Introduction to parallel computing and systems

M. Sato

Contents

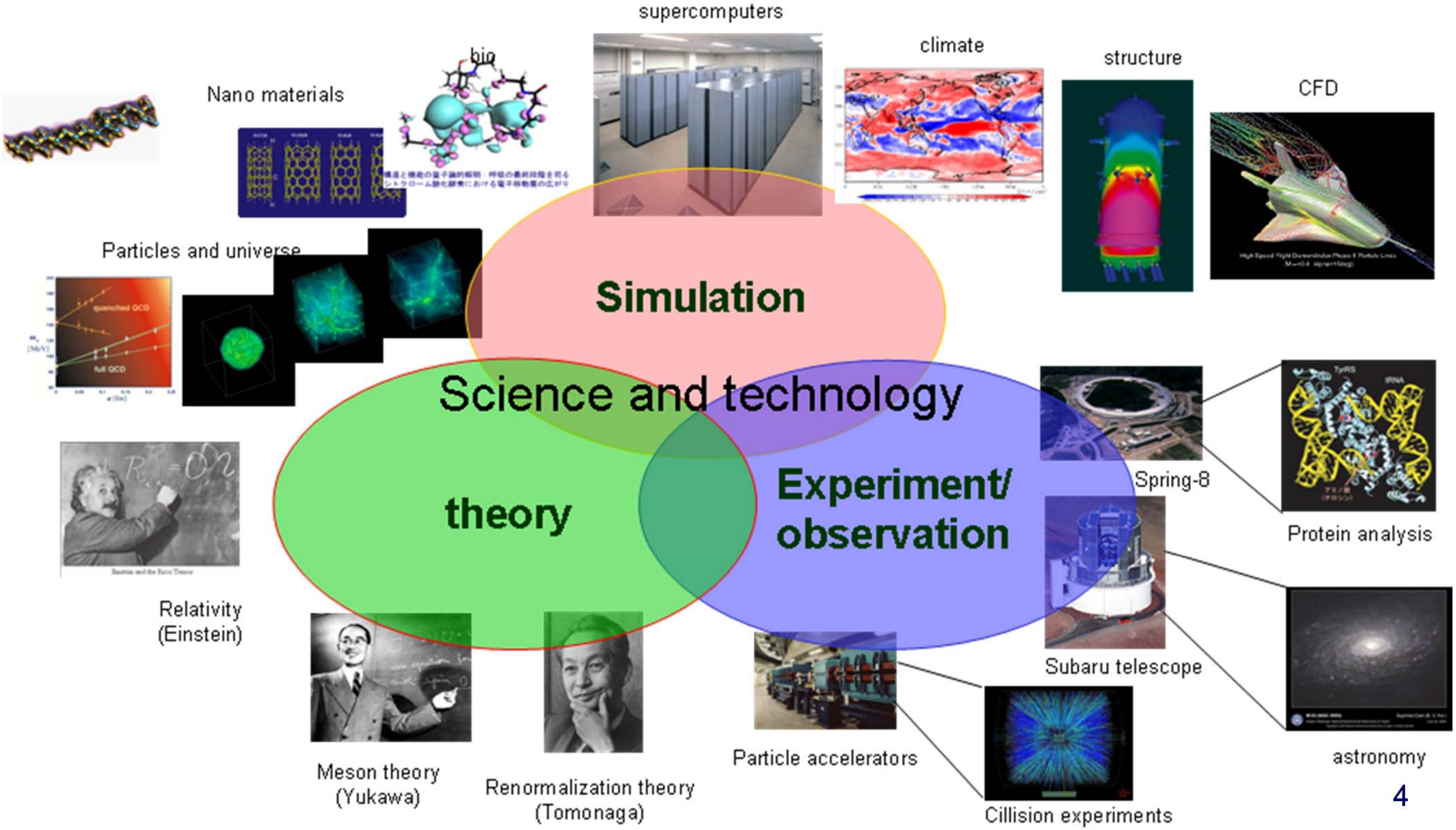
- What is HPC
- Parallel Computing
- Basics of Parallel computing
- Next
  - Parallel Programming
  - Trends of High Performance Comptuing

What is HPC ?

- Today’s science (domain science) is driven by three elements
  - – Experiment
  - – Theory
  - – Computation (Simulation)
- In many of these problems, computation performance and capacity are required to be larger and larger
  - – Floating point operation speed
  - – Memory capacity (amount)
  - – Memory bandwidth (memory speed)
  - – Network bandwidth (network speed)
  - – Disk (2nd storage) capacity
- “High Performance” does not mean only the speed but also capacity and bandwidth

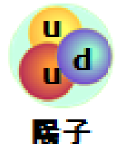
Computational science

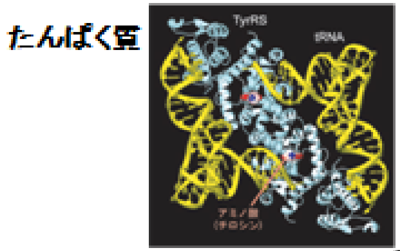
- Large-scale simulations using supercomputers
- Critical and cutting-edge methodology in all of science and engineering disciplines
- The third “pillar” in modern science and technology

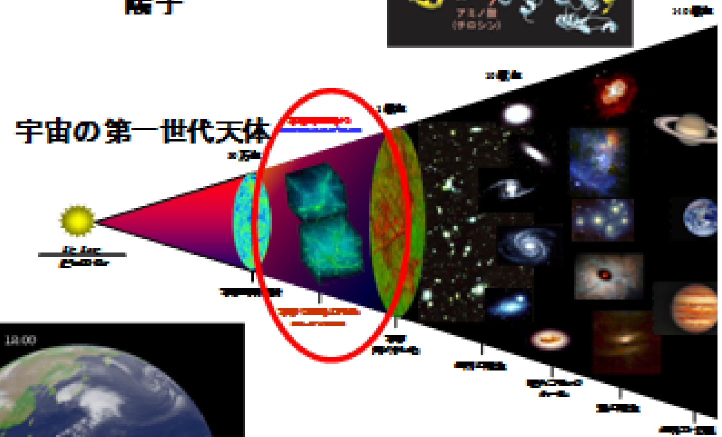


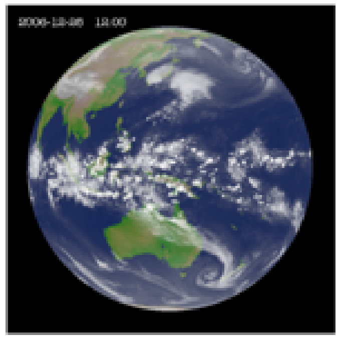
What computational science can do ...

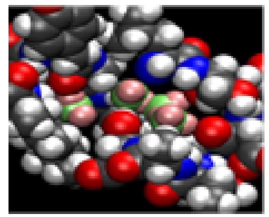
- To explore complex phenomenon which cannot be solved by "paper and pencil"
  - Particle physics to explore origin of materials
  - Phenomenon caused by aggregation of DNAs and protein
- To explore phenomenon which cannot be solve by experiment
  - Origin of universe
  - Global Warming of the earth
- To analyze a large set of data "big data"
  - Genome informatics
- To reduce the cost by replacing expensive experiments
  - car crash Simulation
  - CFD to design air craft



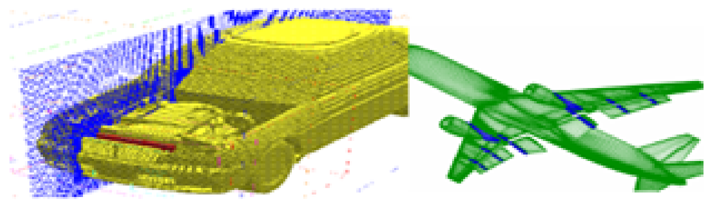








DNA



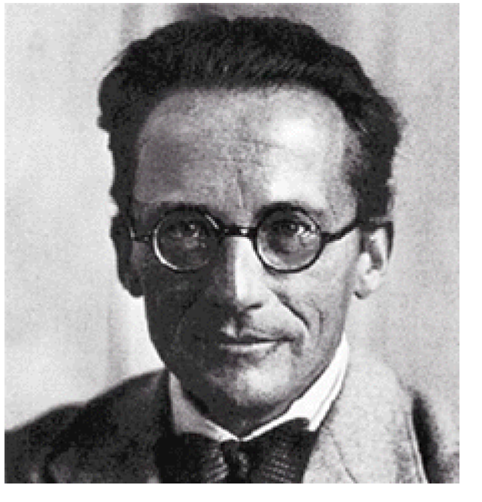
First principal method: computer simulation based on only computation without "experimental parameters. But it may require a huge computations

"First principal computation"...

- Schrödinger equation

$$\mathbf{H}\psi = E\psi \qquad i\hbar\frac{\partial \psi}{\partial t} = \left[-\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + V(x)\right]\psi$$

- "first principle calculation(computation)" in computational material



How to make Computer fast ...

- Metric of speed of computation: arithmetic operations (floating point) per second
- MFLOPS: Millions of FLoating Point OPerationS.
- GFLOPS：10^9 ops，TFLOPS：10^{12} ops，PFLOPS：10^{15} pos、Exa

- ① By making electric circuit work fast
  - Increasing clock speed

  (Frequency of processors used

  in PC: 2～3GHz)
  - Using fast transistor

History of hardware of supercomputers

- 1983: 1 GFLOPS, 1996:1 TFLOPS…
- Before 1990's, the main stream of "supercomputer" was vector supercomputer

- Rapid progress of microprocessor (all components in a chip) used for PC --- "killer micro"
  - Moore's Low: integration (density) of transistor increase double per 1.5 year
  - 4004(first microproessor、1971、750KHz) 8008(1972、500KHz、Intel) 8080(1974、2MHz、Intel)
  - Pentium 4 (2000、~3.2GHz)

  - <u>Clock speed increased from 1MHz to 1GHz in the last three decades</u>

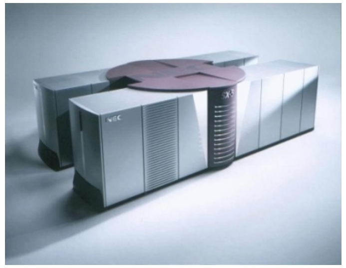
To make computer fast …

- ② By good mechanisms (architecture) in computer
  - mechanism to execute many instruction at a time (in one clock ...)
  - Vector supercomputer: a computer with computing unit to execute vector computation frequently used in scientific computation
    (1980's)

Fujitsu VPP500

Fujitsu VPP5000

NEC SX-4



NEC SX-5

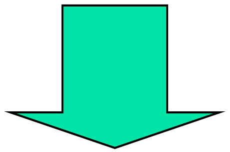
To make computer fast ...

- ③ by using many computer at a time
  - Parallel computers, parallel processing ...
  - This is a main stream in supercomputer !

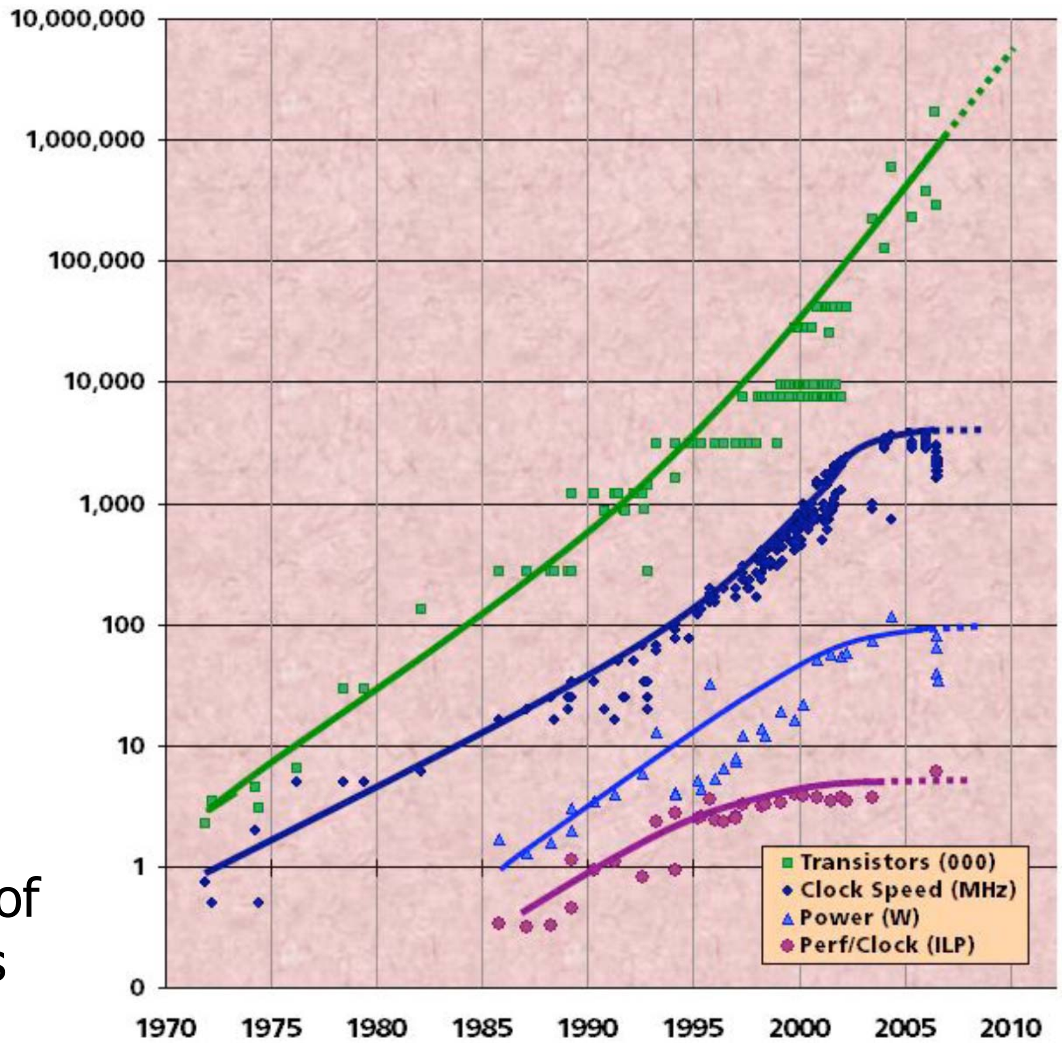
  - You can find 2 or 3 processors in a PC or "smart phone"!

Moore's Law re-interpreted

- Progress of clock speed stops after 2000's
- Still increasing the number of transistors



- Multicore
  - Core (computer) in onechip
  - double in the number of cores every 18 months



TOP 500 List: How to measure (rank) performance of supercomputers

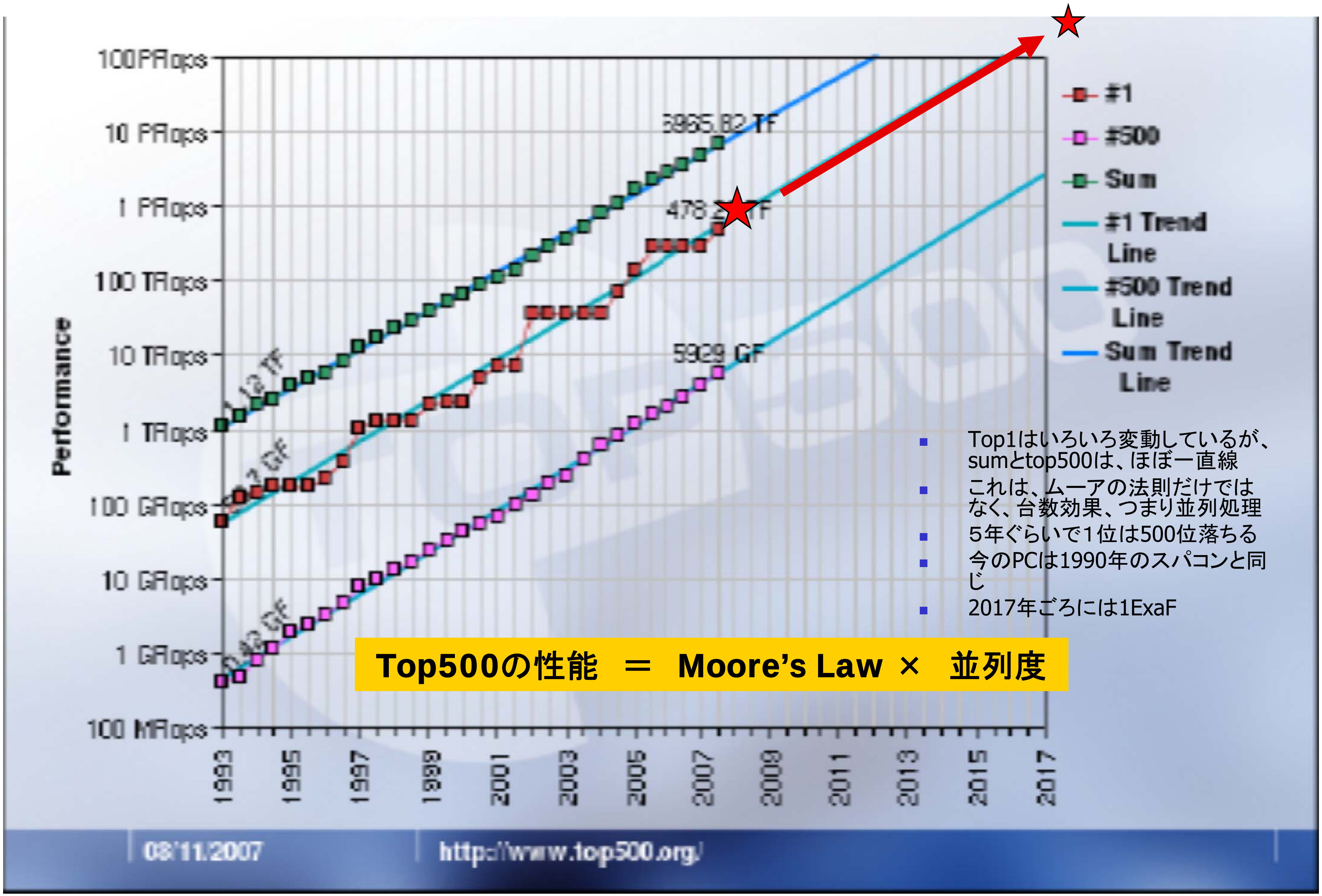
http://www.top500.org/

- Ranked by the performance of benchmark program "LINPACK"
  - LINPACK solves a huge size of linear equations
  - the size is more than 10 millions

- Different from the performance of "real" applications
  - It does not necessarily reflect the performance of "real" applications

- The power consumption is indicated since 2008
  - The power saving is import now !

TOP500：全世界のスパコンランキング500位

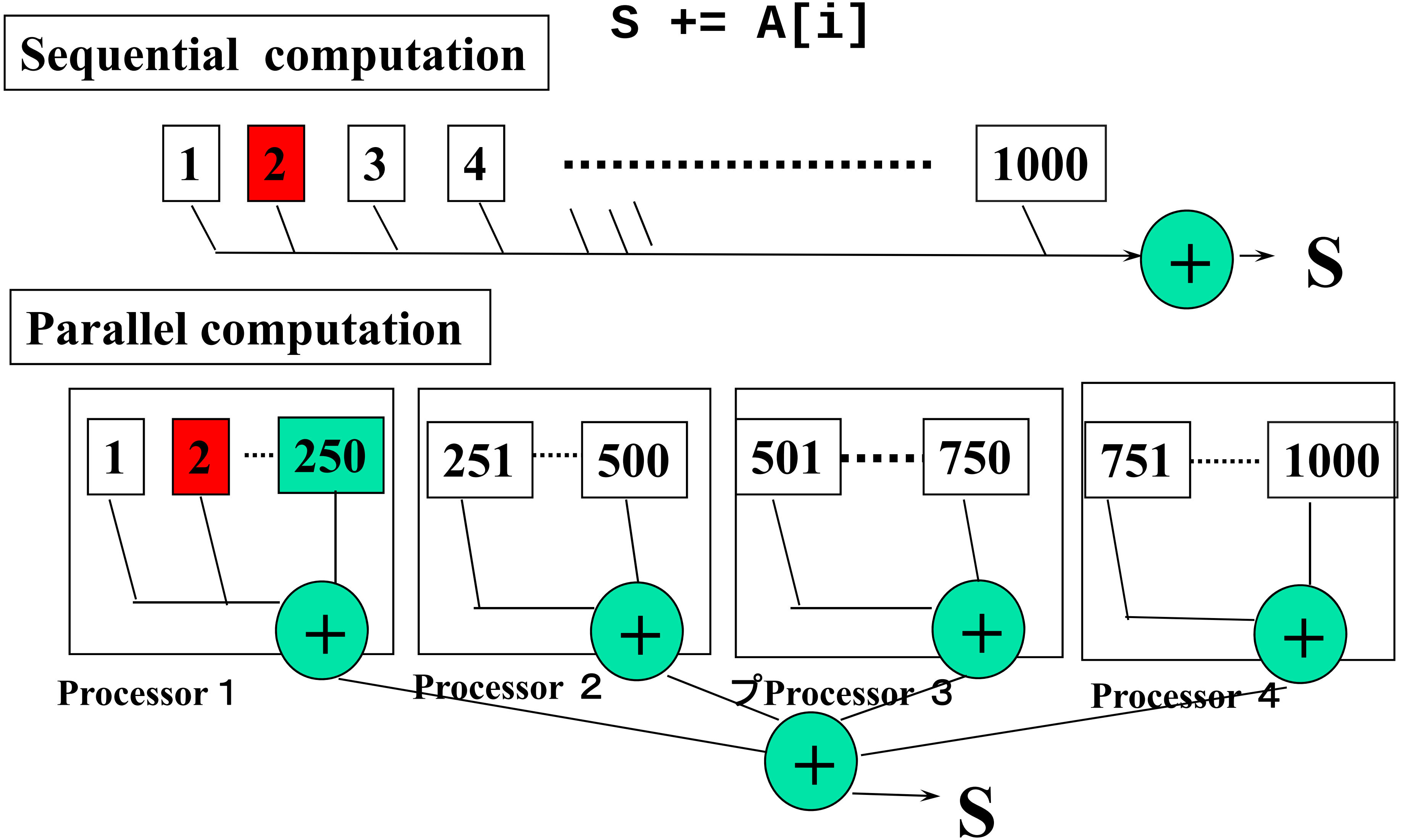


- Top1はいろいろ変動しているが、sumとtop500は、ほぼ一直線
- これは、ムーアの法則だけではなく、台数効果、つまり並列処理
- 5年ぐらいで1位は500位落ちる
- 今のPCは1990年のスパコンと同じ
- 2017年ごろには1ExaF

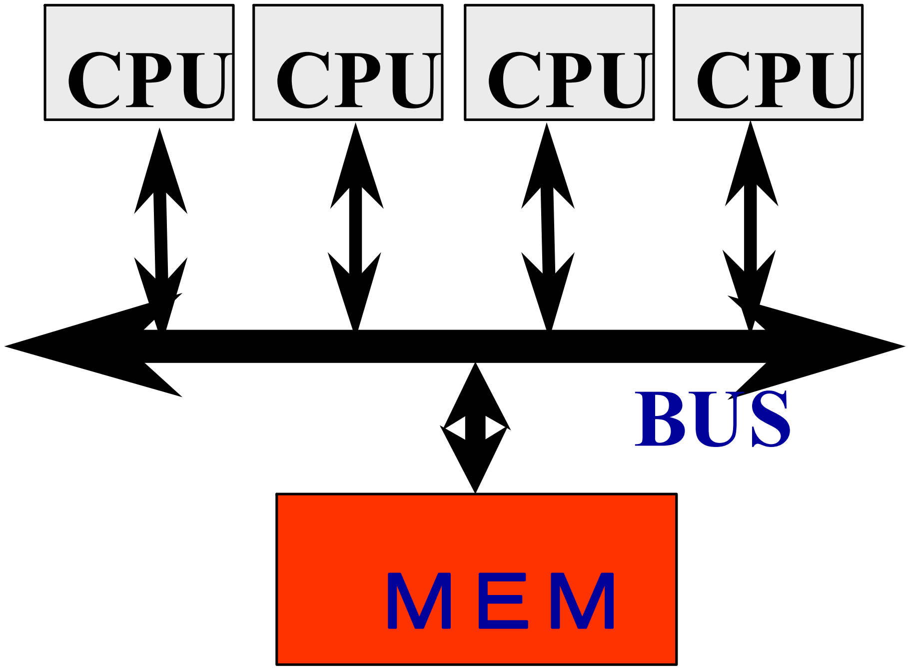
Top500の性能 ＝ Moore’s Law × 並列度

Very simple example of parallel computing for high performance

```
for(i=0;i<1000; i++)
    S += A[i]
```



Shared memory multi-processor system

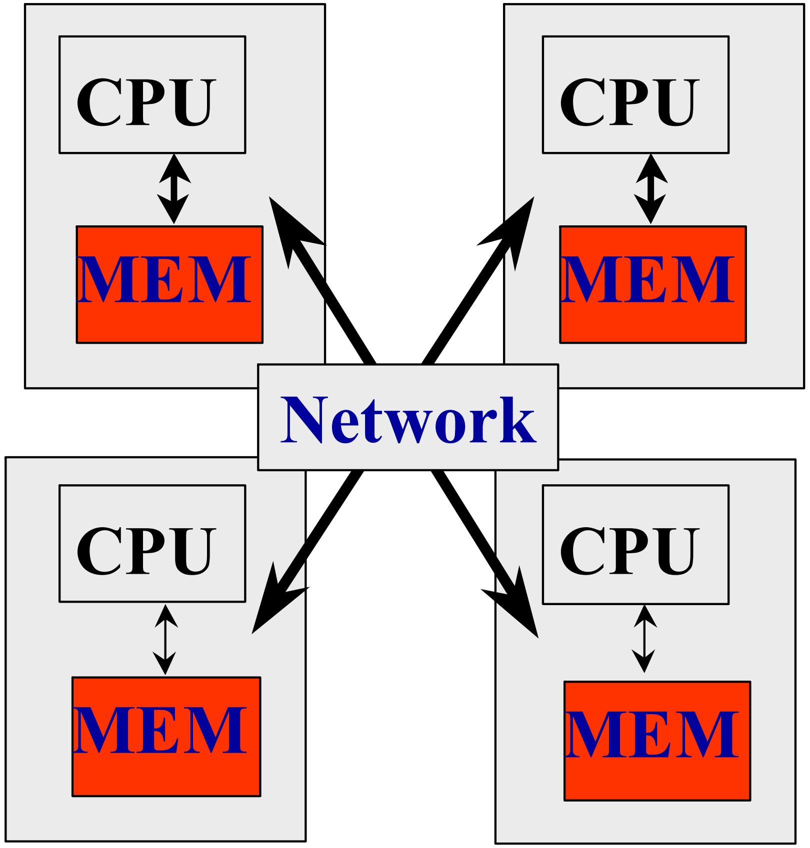


- Multiple CPUs share main memory

- Threads executed in each core(CPU) communicate with each other by accessing shared data in main memory.

- Enterprise Server
- SMP Multi-core processors

Distributed memory multi-processor



- ◆System with several computer of CPU and memory, connected by network.
- ◆Thread executed in each computer communicate with each other by exchanging data (message) via network.タ
- ◆PC Cluster
- ◆AMP Multi-core processor

京コンピュータ “The K computer"

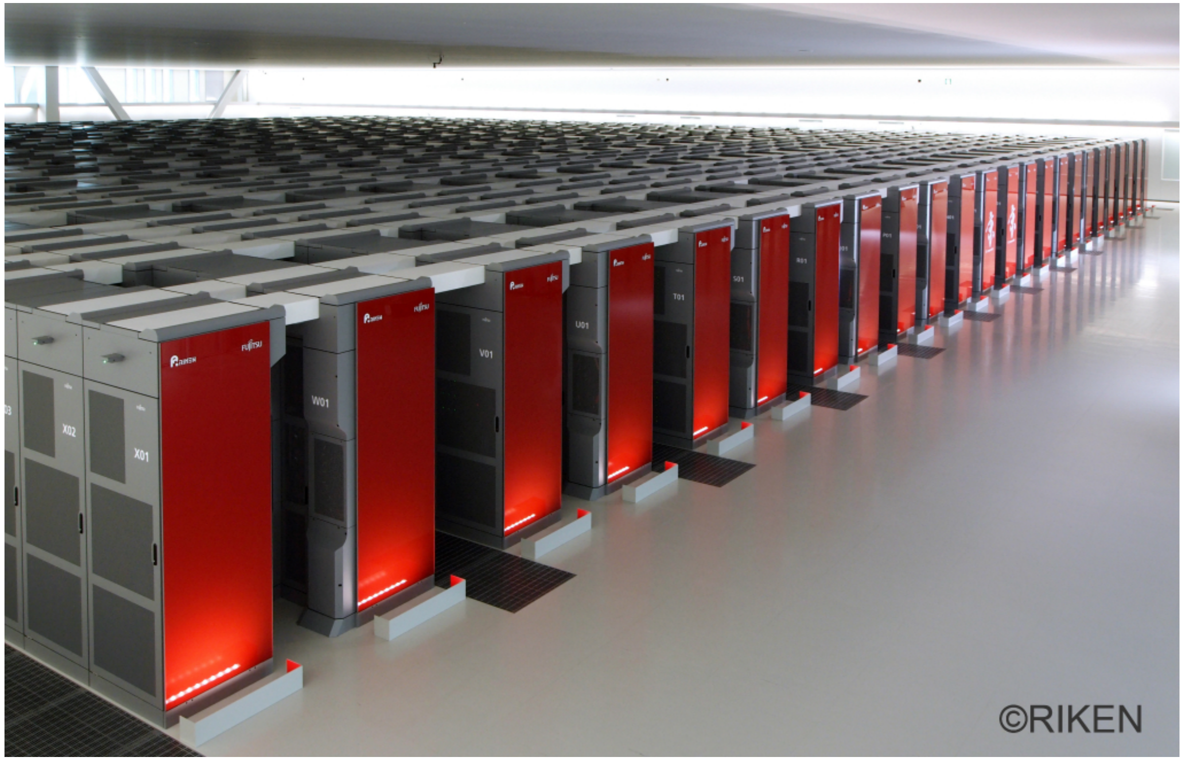
Facts of the K computer

- The number of racks (boxes) 864
- the number of chips 82,944
- The number of cores (computers) 663,552
- Linpack perf

  10.51PF

  (Power 12.66MW)

  2011/11月



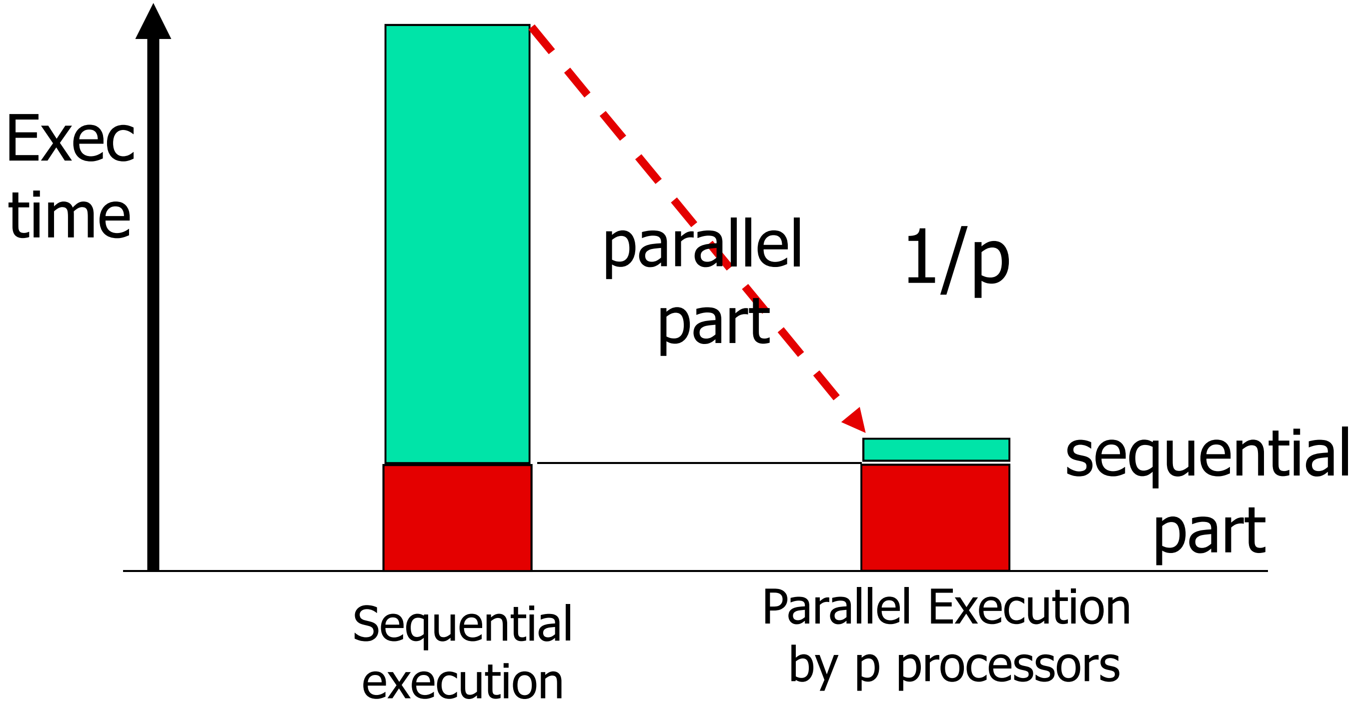
Amdahl's low

- Question: How much do parallel computers became fast by increasing the number of processors???

ジーン・アムダール（Gene Amdahl、1922年11月16日 - ）は、アメリカ人のコンピュータアーキテクトで、企業家ある。彼の業績はIBMおよび彼の創設した会社(特にアムダール社)における、メインフレームの設計である。並列コンピューティングの基本的な理論としてアムダールの法則がよく知られている。(wikipediaより)

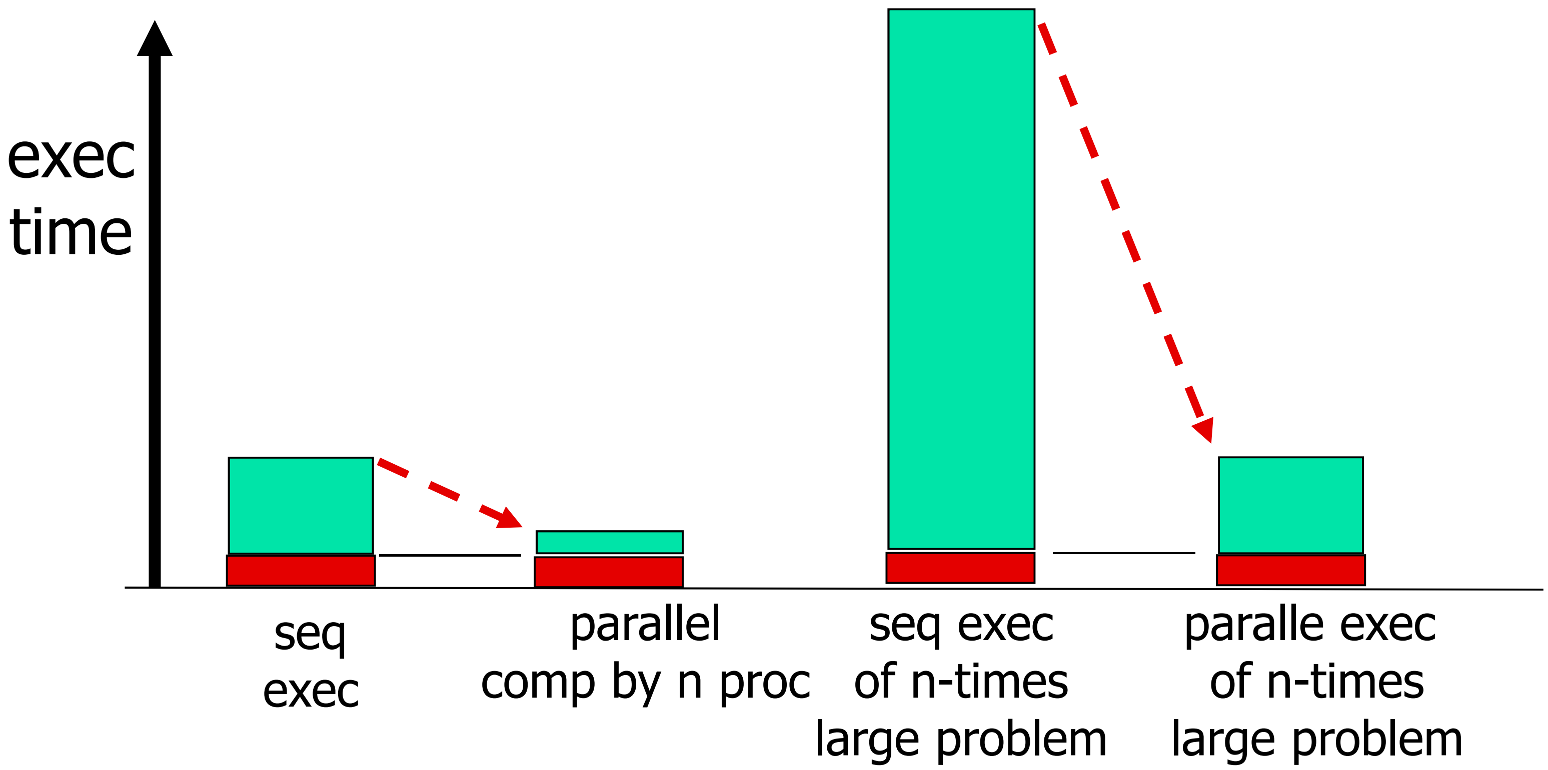
Speedup by parallel computing: "Amdahl's low"

- Amdahl's low
  - Suppose execution time of sequential part T_1, ratio of sequential part α, execution time by parallel computing using p processors T_p is (no more than) $T_p = \alpha * T_1 + (1-\alpha) * T_1/p$
  - Since some part must be executed sequentially, speedup is limited by the sequential part.



Breaking ”Amdahl’s low”

- "Gustafson's low"： what about performance of real apps?
  - The fraction of parallel part often depends on the size of problem
  - For example, n-times larger problem to be solve by n-times larger parallel computers.
  - Weak scaling – Scaling with constant size per processor ← in the case of large scale scientific applications
  - Strong scaling —Scaling with constant size problem ← We need fast one-processor.



How different between the K computer and your PC?

- The processors (computer) used are almost the same!
  - Even slow clock for the K computer, but some enhancement in computing unit.

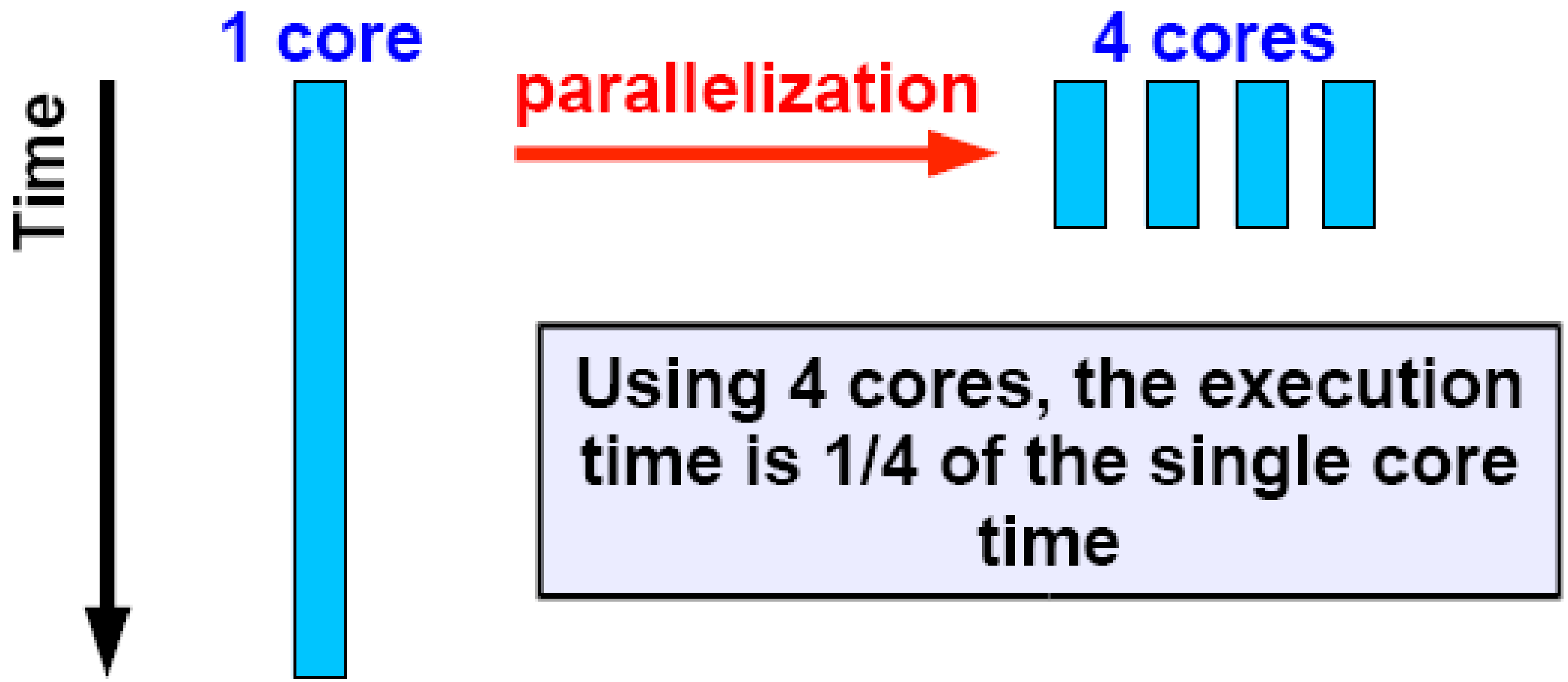
- The K computer consists of many "processors"
  - 80,000 chip、0.64 M cores
  - Fast network between processors is required!

- The programmer is forced to make parallel program to make use of many processors
  - The program running on the PC (sequential program) does not run fast !

Parallel computing

- For efficient parallel processing, certain "granularity" of parallel processing unit and enough degree of parallelisms are necessary
- Ordinary (non-scientific) applications are not sufficient to satisfy these conditions naturally
  - ex. "Word" or "Excel" applications do not have parallelism nor large amount of computation in a second
- Various scientific computations satisfy these conditions, and there are much requirement of solving these problems (especially for high-end domain science)
- Large scale parallel processing is naturally getting along with HPC
- So many numerical algorithms have been developed for scientific computation which is enable on parallel systems
- In many cases, matrix computation is essential, but direct solution is more effective in some cases

Why parallelization needs?
4 times speedup by using 4 cores!



Parallel Processing and Distributed Processing

- parallel processing is defined as a technology to process/compute faster by using many processors simultaneously
  - HPC(High Performance Computing)
    - scientific simulation "supercomputing"
  - HTC (High Throughput Computing)
    - processing a huge amount of data "big data"

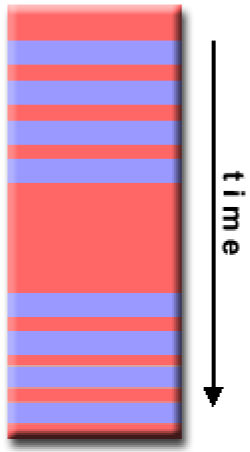
- Distributed processing is referred as a technology to process/compute by using many processors, but it federate several functions executed in different computers to provide high-level services.
  - Distributed objects ...
  - RMI , J2EE, Jini…

Some terminologies

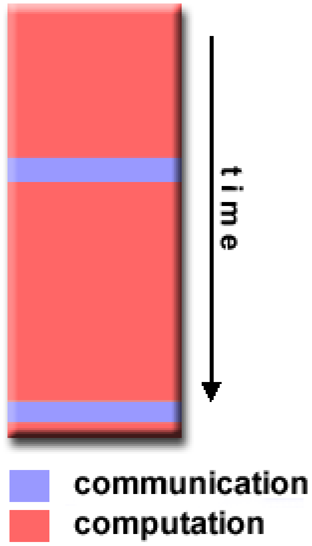
- Node – A standalone "computer in a box". Usually comprised of multiple CPUs/processors/cores. Nodes are networked together to comprise a parallel system.
- Task – A logically discrete section of computational work. A parallel program consists of multiple tasks running on multiple processors.
- Communications – Parallel tasks typically need to exchange data. There are several ways this can be accomplished, such as through a shared memory bus or over a network.
- Synchronization – The coordination of parallel tasks in real time, very often associated with communications. Often implemented by establishing a synchronization point with an applications where a task may not proceed further until another task(s) reaches the same or logically equivalent point.

Some terminologies

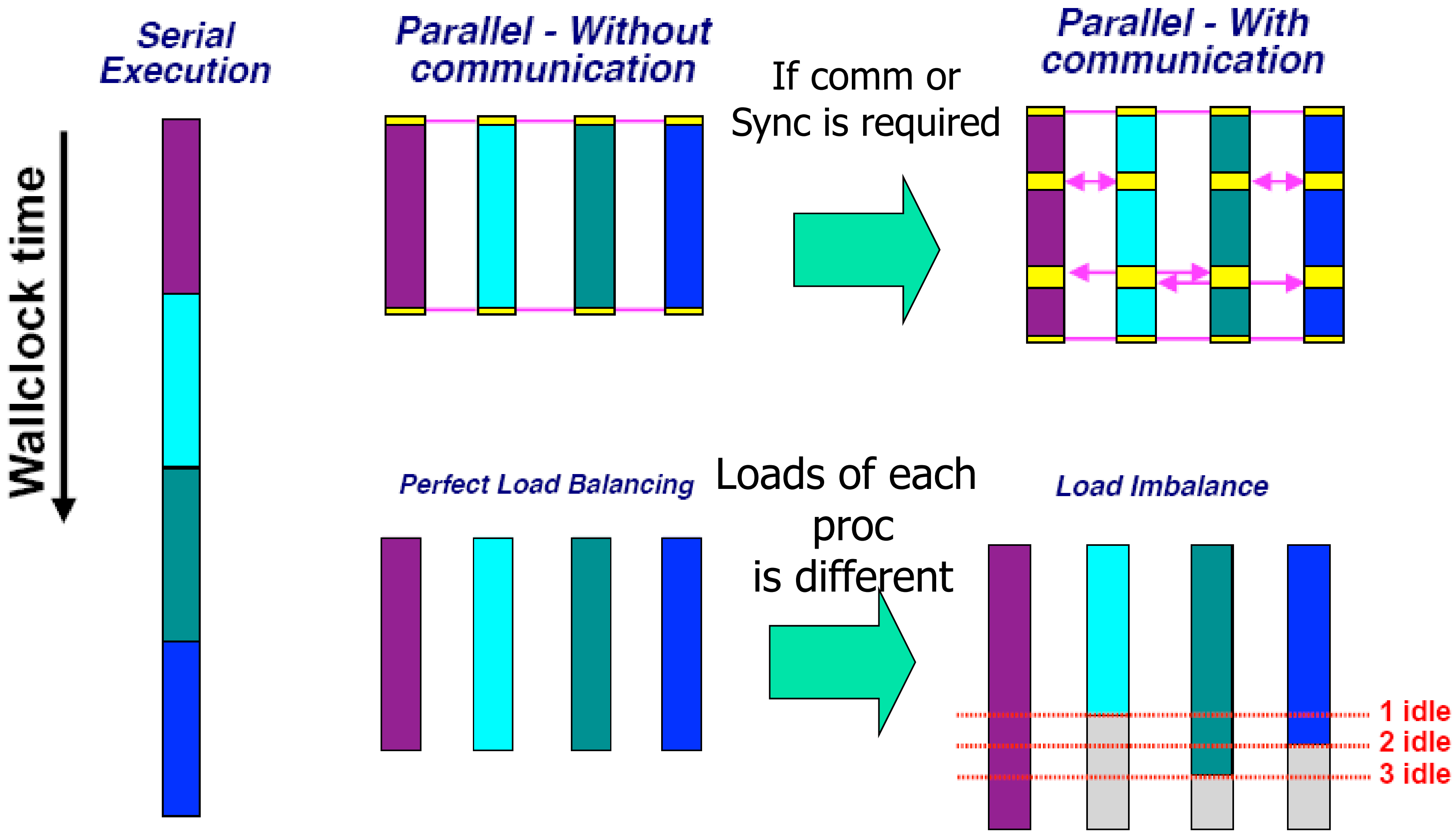
- Granularity – in parallel computing, granularity is a qualitative measure of the ratio of computation to communication.
  - Coarse : relatively large amount of computational work are done between communication events
  - Fine: relatively samll amount of computational work are done between communication events



- Parallel overhead – The amount of time required to coordinate parallel tasks, as opposed to doing useful work. Parallel overhead can include factors such as:
  - Task start-up time
  - Synchronization
  - Data communications
  - Software overhead imposed by parallel compiler, libs, tools, ...
  - Task terminations



Overhead of parallel execution

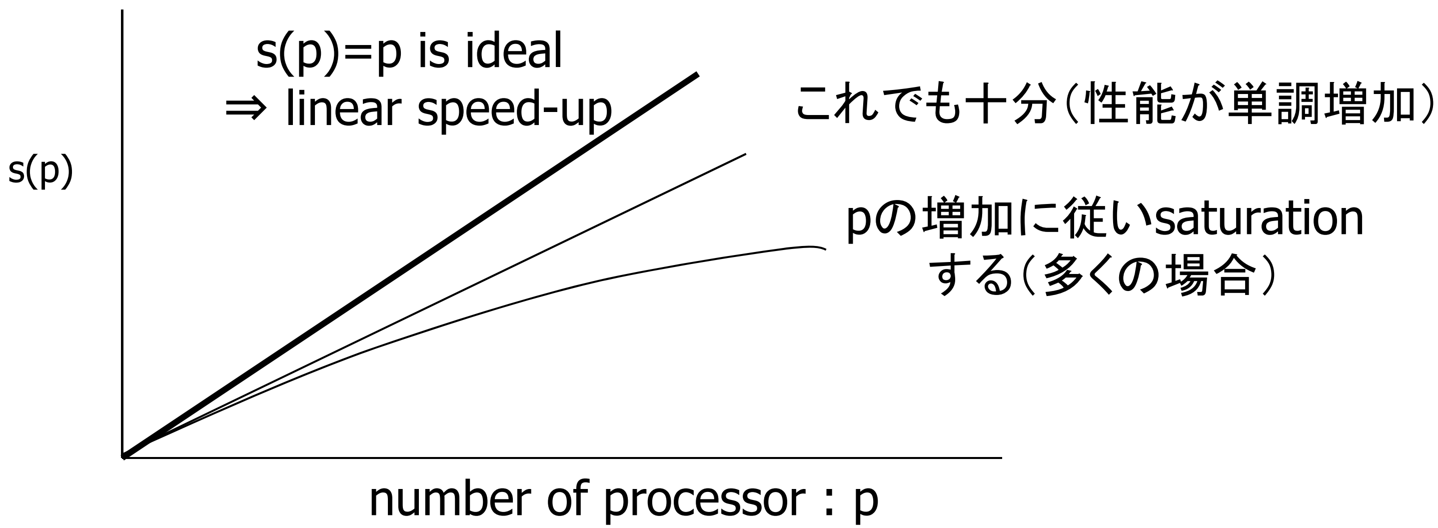


Some terminologies

- Scalability – Refers to a parallel system's (hardware and/or software) ability to demonstrate a proportionate increase in parallel speedup with the addition of more processors. Factors that contribute to scalability include:
  - Hardware – particularly memory-cpu bandwidth and network communications
  - Application algorithm
  - Parallel overhead related
  - Characteristics of your coding and apps.

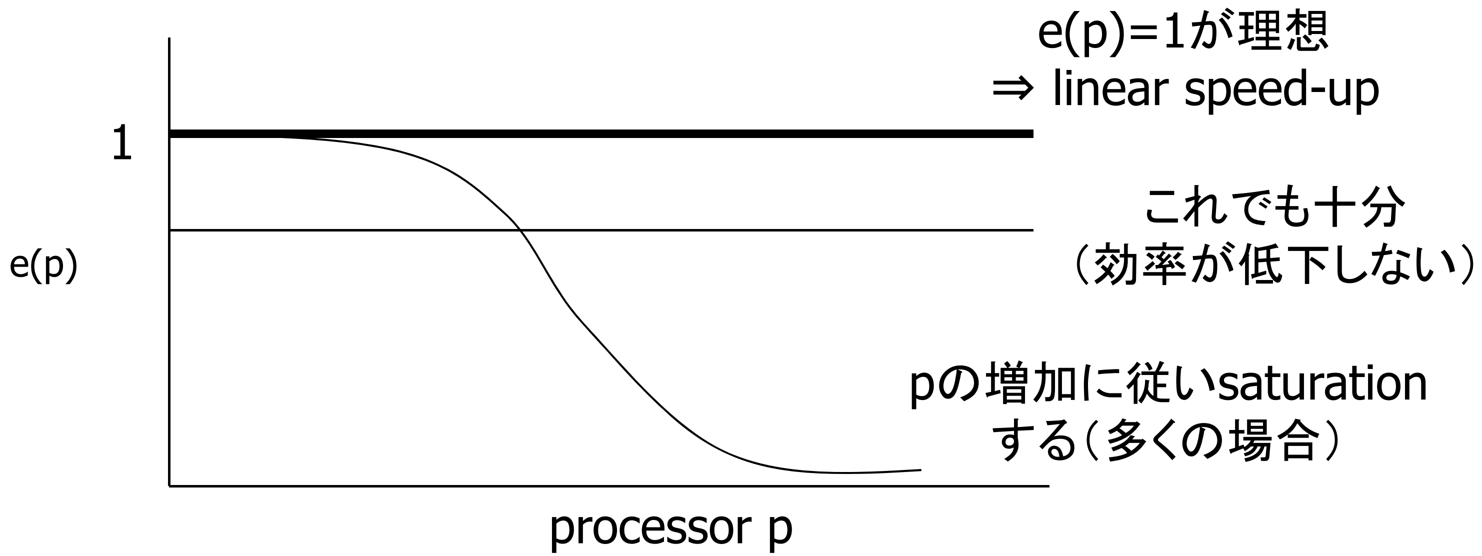
Metric of Performance of Parallel Systems

- Speed up
  - T : execution time by 1 processor
  - *T(p)* : execution time by *p* processors
  - *s(p)=T/T(p)*
    *s(p)*を: speedup by processor p. if *s(p)* is more than 1, the speed of computation increases
  - Ideally it should be *s(p)=p*（*p*台のプロセッサを投入した結果、*p*倍の速度が得られた）



Metric of Performance of Parallel Systems

- Efficiency
  - Speedup is not useful since *s(p)* depends on p
  - Suppose「*s(p)=p is ideal*」, this metric is defined as how much this ideal is archived.
  - *e(p)=s(p)/p*
    e(p) does not depend on *p*. It is good if it is close to 1

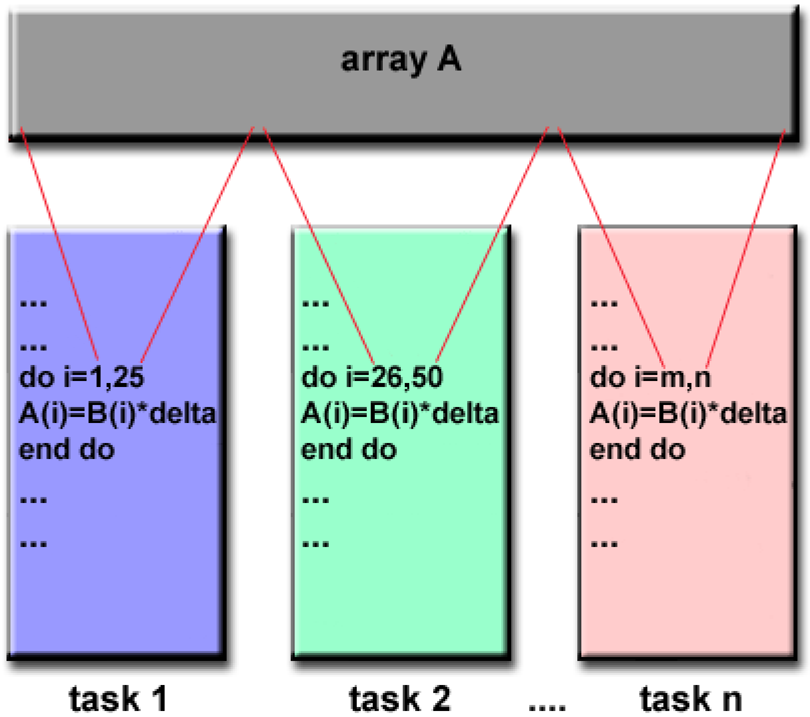


Data Parallel Model

- The data parallel models demonstrates the followings:
  - Most of the parallel work focuses on performing operations on a data sets. The data set is typically organized into common structure, such as an array or cube.
  - A set of task work collectively

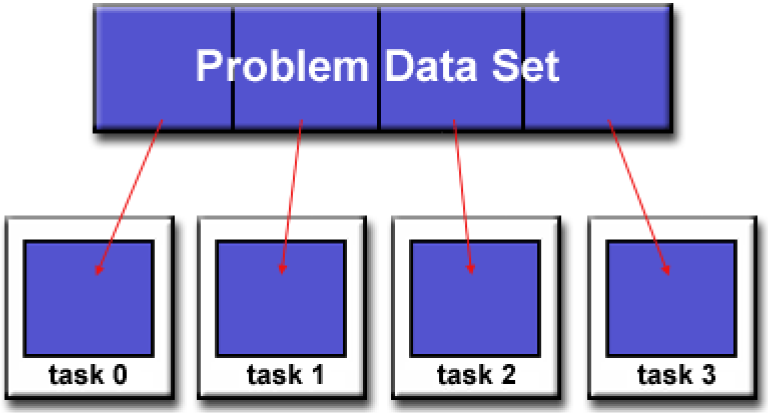
  on the same data structure, however, each task works on different partition of the same data structure.
  - Tasks perform the same

  operation on their partition of work

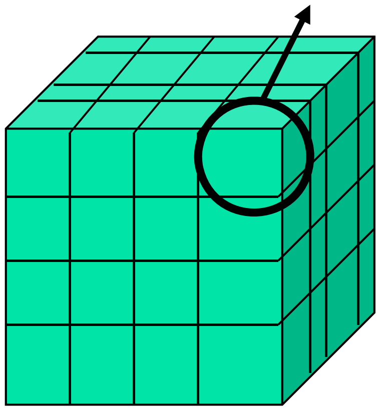


Example of data parallel model

- domain decomposition
  - Divide the space of simulation into uniform grids
  - Perform the same computation on each gird, sometimes with interaction of neighbor
  - example:



grid for computational unit



simulation space

```
for(t=0; t < T; t++){
   for(i=0; i < N; i++)
      a[i] = b[i-1] + 2*b[i] + b[i+1];
   for(i=0; i < N; i++)
      b[i] = a[i];
}
```

b[...] の部分で自分 (i) 以外のインデックスが出てくる

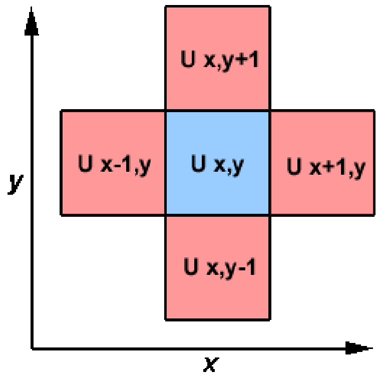
Simple Heat Equation

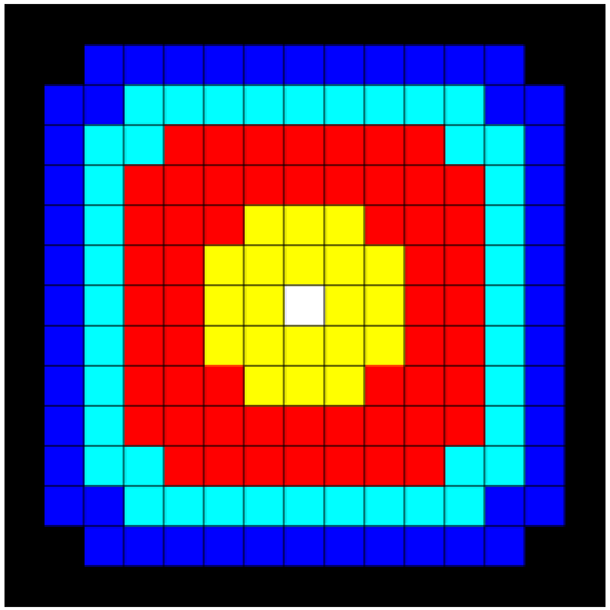
- Most problems in parallel computing require communication among the tasks. A number of common problem require communications "neibhbor" task. (stencil computations)
- A finite difference scheme is employed to solve the heat equations numerically on a square regions.
- For the fully explicit problem, a time stepping algorithm is used. The element of a 2-dimensional array represent the temperature at the point on the square.

$$U_{x,y} = U_{x,y}$$

$$+ C_x * (U_{x+1,y} + U_{x-1,y} - 2 * U_{xy})$$

$$+ C_y * (U_{x,y+1} + U_{x,y-1} - 2 * U_{x,y})$$

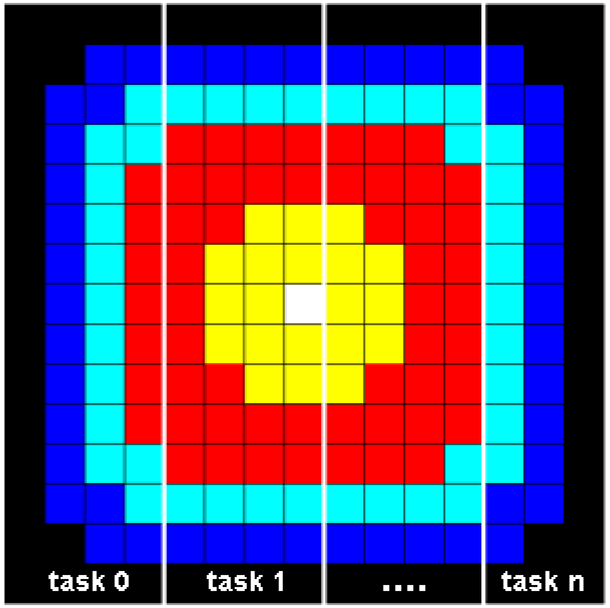




Simple Heat Equation

- The entire array is partitioned and distributed as subarray to all task. Each task owns a portion of the total array.
  - send slave read of u1 to neighbor processor
  - receive u1
  - compute u2 at each processor
  - update u1 with u2
  - repeat the above computation until

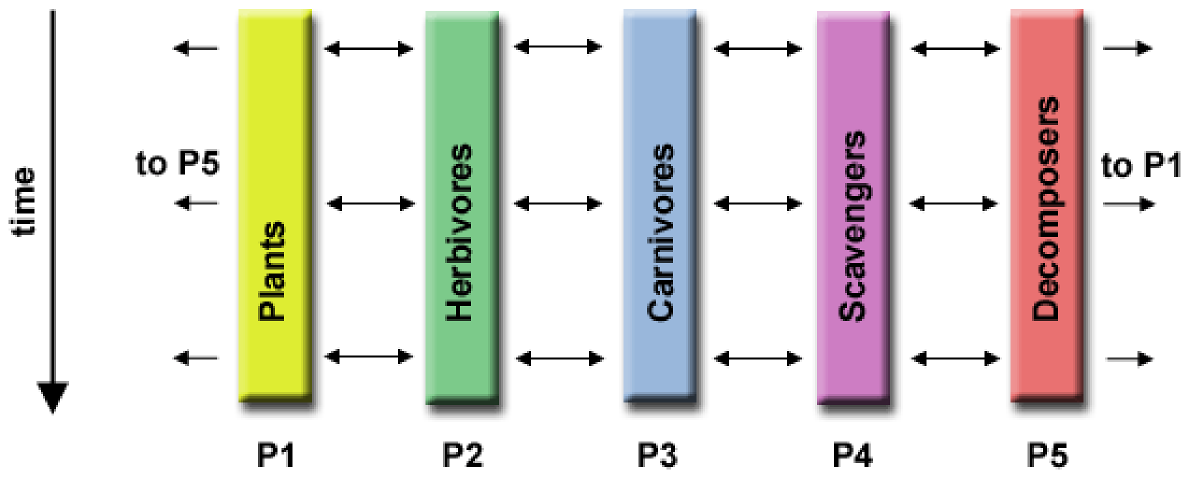
    the condition is satisfied.

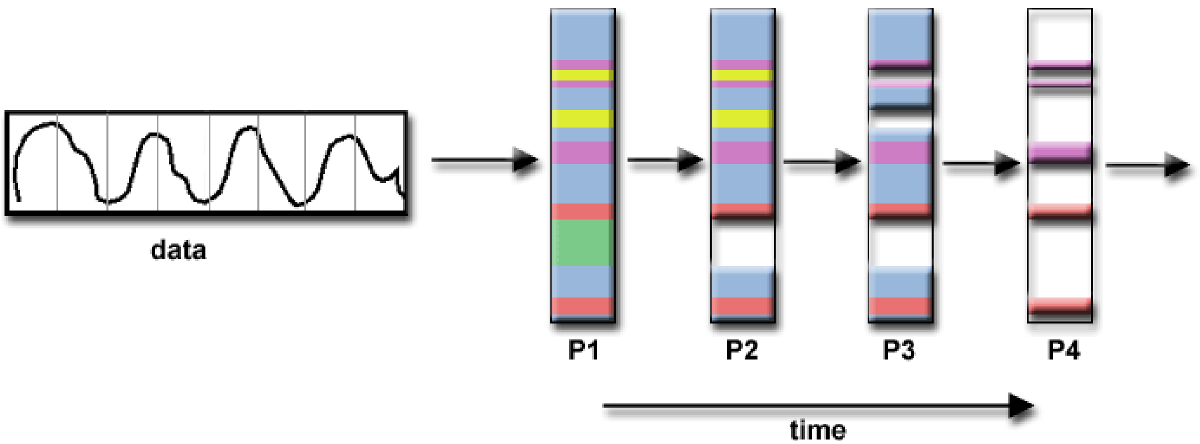



```
do iy = 2, ny-1
do ix = 2, nx-1
  u2(ix,iy) = u1(ix,iy)+
      cx*(u1(ix+1),y)+u1(ix+1,iy)-2*u1(ix,iy))+
       cy*(u1(ix,iy+1)+u1(ix,iy-1)-2*(ix,iy))
end do
end do
```

Pipeline

- Breaking a task into steps performed by different processors unit, with inputs streams through, much like assemble lines
- Example: signal processing





master/worker parallel processing

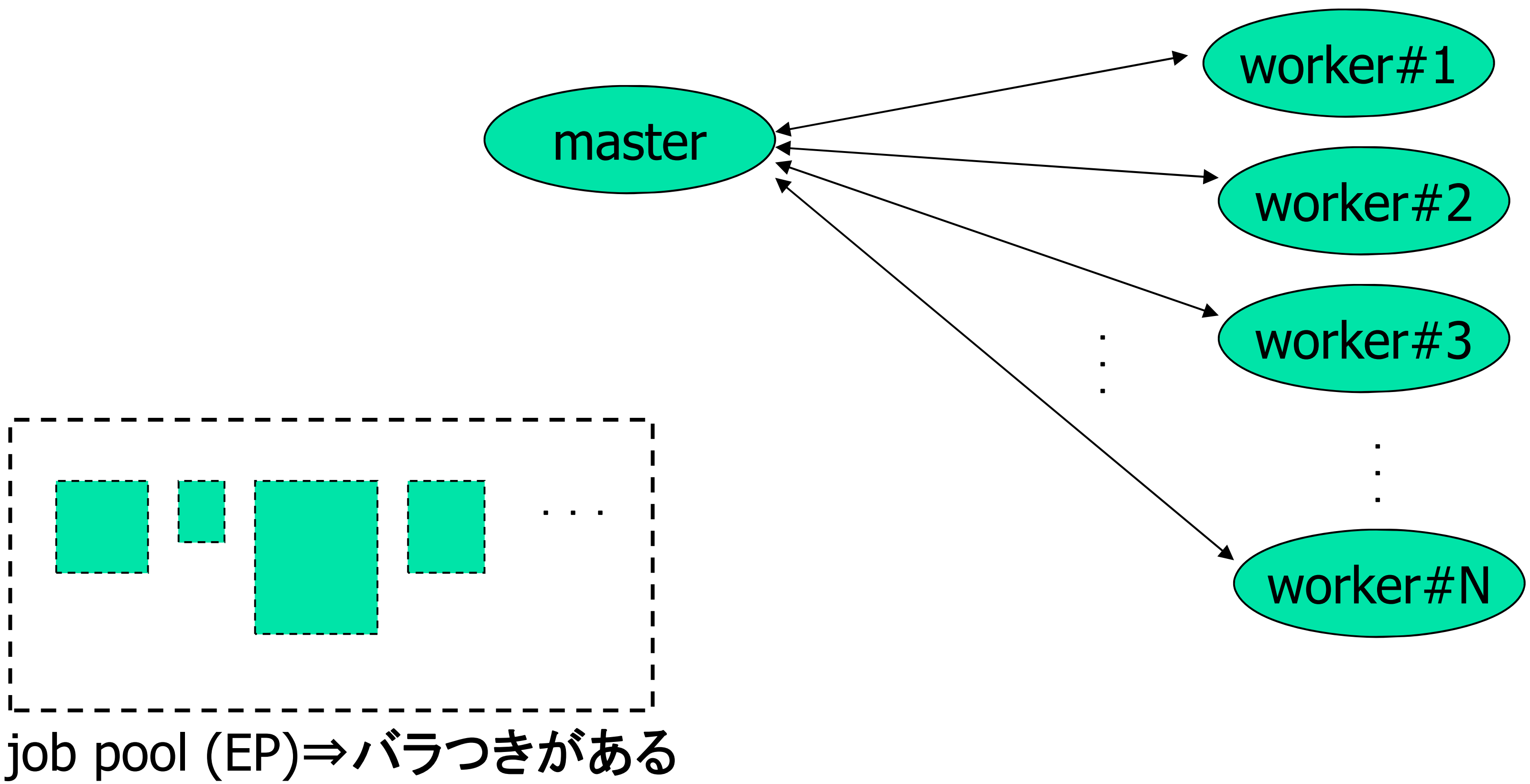
- one master processor and several worker processors
- A pool of work in master processor.
- master pick up one work to send the work to a worker.
- When worker finish the given work, then it return the result and receive next work

```
master::
// give a job to each worker
while(1){
    // receive a worker's result
    // give the next job to that worker
}
```

```
worker::
while(1){
    // receive a job from master
    // process the job
    // send the result to master
}
```

master/worker parallel processing

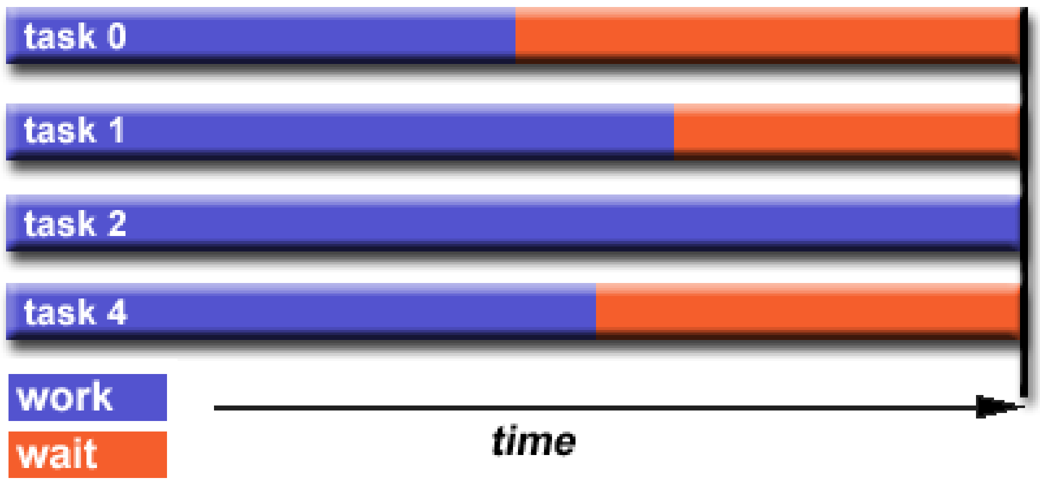
- It is effective parallel processing when each work have different load --> load balancing



Load Balancing

- Load Balancing refers to the practice of distributing work among tasks so that all tasks kept busy all of the time. It can be considered a minimization of task idle time.
- Load balancing is important to parallel programs for performance. For example, if all tasks are subject to a barrier sync point, the slowest task will determine the overall performance.
- How to achieve load balance:
  - Equally partition the work

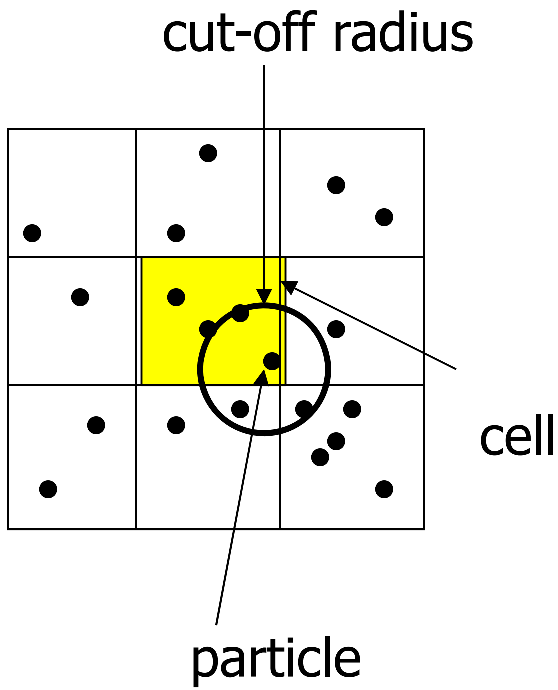
    each tasks receive.
  - Use dynamic work assignment
    - Master-Worker



Example: Molecular Dynamics with cut-off radius

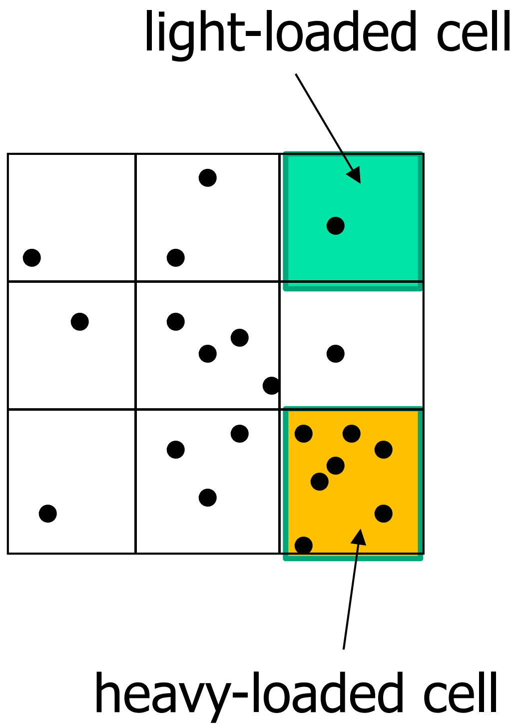
- MD (Molecular Dynamics)
  - Compute interaction between P particles in n-dimensional space.
  - Interaction may be force between particles
  - If the force is effective only within near fields, cut-off distance can be assumed.

- To save the computations, only computation within the cut-off radius should be done, not all-to-all interaction,
  - Space is divided into cell by "domain decomposition" assigned to each node, and each node computes particles within the assigned cell ⇒ If the size of cell is larger than the cut-off radius, nodes may communicate only with nodes of neighbor cells（cell mapping method）



Example: Molecular Dynamics with cut-off

- Particles moves by the force of interaction from other particles, as steps go. As a result, it may happens that many particles moves into a certain cell. (condense)
- In the case that cell is assigned to nodes in one-to-one manner, load imbalance may occur.
- In order to keep load balance, the number of particles computed by nodes should be balanced rather than the number of cells.
- Methods:
  - Method 1) Periodically, the number of cells are re-assigned (adjusted) according to the density of particles in the cell (the number of particle/cell)
  - Method 2) If the number of cells is far more than the number of nodes, use cyclic mapping rather than block mapping.
  - Method 3) Use particle mapping, not cell mapping.



Example：Molecular Dynamics with cut-off

- Method 1)
  To re-assign cells to nodes, a large amount of data should be exchanged (needs much comm) Since assignment will be irregular, the communication pattern is not neighbor communication.
- Method 2)
  Cyclic mapping is a simple way to take a good load balance. But, the communication pattern is not neighbor communication.
- Method 3)
  To keep track which nodes each particle is assigned to, the table to manage the index table of the assignment between particles and nodes, resulting a complicated and expensive computation and communications.

  ⇒ No best solution for all cases.
- Depends on the characteristics of phenomenon to be solved (how particle behaves, or what potential force.).
- It may important to keep load balancing in the case of heavily load imbalance.