Korea-Japan HPC Winter School 2016

“Fundamentals of HPC and Parallel Processing”

Taisuke Boku
taisuke@cs.tsukuba.ac.jp
Center for Computational Sciences
University of Tsukuba

Contents

- Performance metric on computation and communication
- What is parallel processing ?
- Requirement for parallel processing
- Parallel processing methods
- Communication & synchronization
- Parallel efficiency and Amdhal's Law
- Load balancing (advanced issue)

Performance metric on computation and communication

- computation performance (mainly floating point)
  - FLOP: (number of) Floating point Operations
    number of floating point operations in the processing
    ex) for(i=0; i<100; i++) a[i] = b[i] * c + b[i];
    ⇒ 200FLOP
  - FLOPS: Floating point Operations Per Second
    floating point operations per second -> Performance
    ex) computing the above calculation in 2 micro-sec.-> 100 MFLOPS
    K: 10^3 M: 10^6 G: 10^9 T: 10^{12} P: 10^{15} E: 10^{18}
- communication performance
  - B/s (Byte/sec):
    data transfer amount per second
    ex) theoretical peak performance of Infiniband 4xQDR = 4 GB/s
    sometimes, with bps (bit per second)
    Caution: not always 1Byte=8bit !!

What is parallel processing ?

- “Decomposing single problem with in a number of processes and solving it to enhance the performance and/or increase problem size”
  - “Solving single probem” ⇒ differs from distributed processing
  - “Problem decomposing (parallelizing)” ⇒ careful for efficiency
  - “Improved” issues ⇒ not just speed, but also problem size, computing accuracy, etc. (various metrics)
- parallel processing v.s. concurrent processing
  - solving parallelized processes in “pseudo” parallel ⇒concurrent processing
  - solving them in “physically” parallel (simultaneously) ⇒parallel processing
- resources to contribute for parallel processing
  - CPU, memory, disk, network, etc. ⇒ all the computation resources may contribute for improvement
  - hereafter, we call these processes to be mapped to multiple CPUs as “parallel processes”

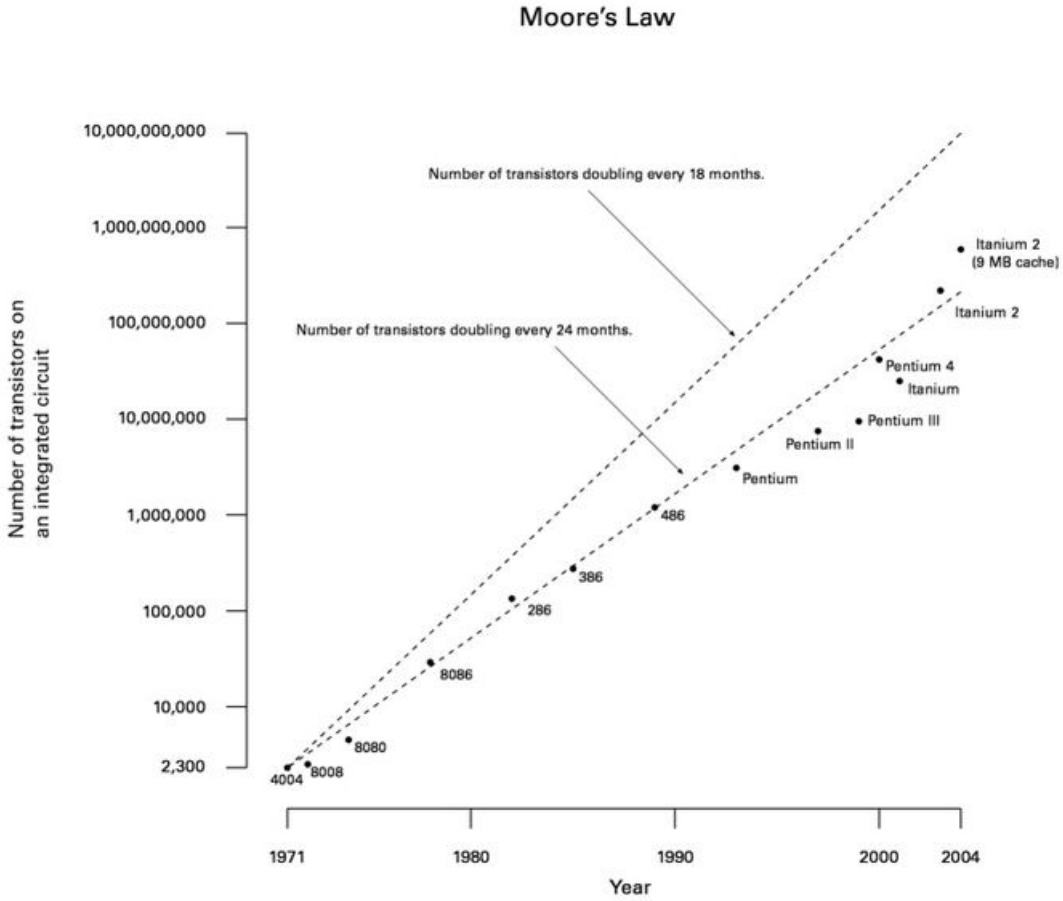
High performance computing (HPC) and parallel processing

- Requirement for numerical computing performance in scientific computation and large amount of data processing
- Computing order to increase the problem size N is not O(N) (linear)
  - 3-dimensional fluid dynamic (climate simulation etc.)
    when spatial resolution on 1-dimension is N, computing operation's order is $O(N^3)$
  - matrix calculation (linear equation)
    for direct method (Gaussian elimination etc.) for N variables of equation, computing operation's order is $O(N^3)$
  - n-body problem (gravity calculation in astrophysics)
    force computation for N particles requires computing operations with $O(N^2)$
- There is no "enough performance nor amount" to the requirement
  ⇒ large scale scientific computation does no more exist without Parallel Processing
- Large amount of computation, data and communication requirements always need effective parallel processing with appropriate resource utilization
  ⇒ "High Performance" in any issue ⇒ Parallel Processing

Limit of computing performance

- Density of semiconductor integration is growing with a rate of twice/1.5year
  ⇒ Moor's Law
- If all the transistors on the silicon chip can contribute to the computation, it says "processor performance is growing as twice/1.5year of speed"
- Transistor count on a chip (by Intel web site)
- Memory capacity is also increased with this rate
- It is impossible to catch up the growth of computing performance requirement
- Also, Moor's Law itself is reaching to the limit

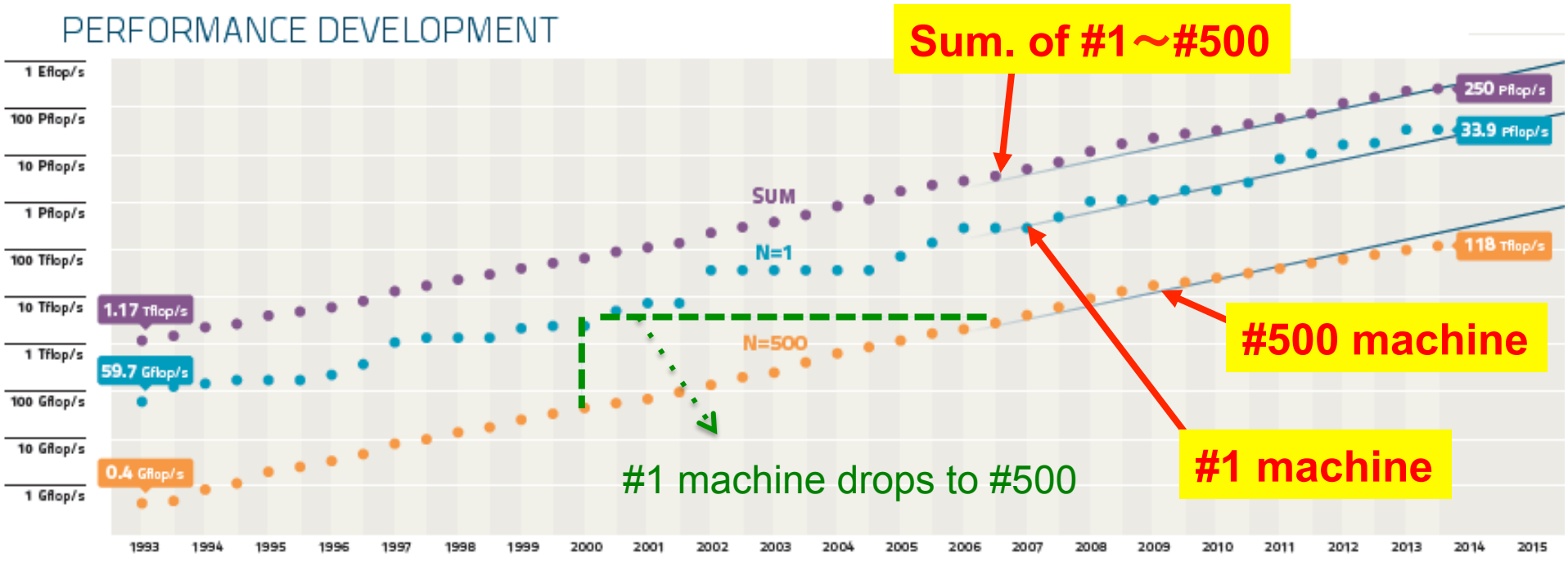
⇒ (Massively) Parallel



(from Intel's home page)

TOP500 List (HPL performance: ~ peak performance)

www.top500.org



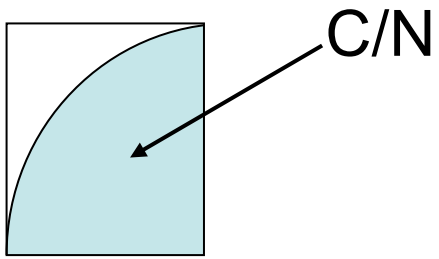
	NAME	SPECS	SITE	COUNTRY	CORES	R_{MAX} PFLOP/S	POWER MW
1	**Tianhe-2 (Milkyway-2)**	NUDT, Intel Ivy Bridge (12C, 2.2 GHz) & Xeon Phi (57C, 1.1 GHz), Custom interconnect	NSCC Guangzhou	China	3,120,000	**33.9**	17.8
2	**Titan**	Cray XK7, Operon 6274 (16C 2.2 GHz) + Nvidia Kepler GPU, Custom interconnect	DOE/SC/ORNL	USA	560,640	**17.6**	8.2
3	**Sequoia**	IBM BlueGene/Q, Power BQC (16C 1.60 GHz), Custom interconnect	DOE/NNSA/LLNL	USA	1,572,864	**17.2**	7.9
4	**K computer**	Fujitsu SPARC64 VIIIfx (8C, 2.0GHz), Custom interconnect	RIKEN AICS	Japan	705,024	**10.5**	12.7
5	**Mira**	IBM BlueGene/Q, Power BQC (16C, 1.60 GHz), Custom interconnect	DOE/SC/ANL	USA	786,432	**8.59**	3.95

Parallelization Method

- Various parallelization methods for problems
  - Partitioning a problem
    ex）domain decomposition: partitioning the problem in spatial domain to dispatch them to parallel processes
  - Distributing a problem
    ex) parameter search: trying a problem with various parameters and getting a statistical result ⇒ a set of parameters are executed in one process, and there is a master process to collect and statistically process them
- Various parallel methods
  - EP (Embarrassingly Parallel): each of parallel processes is individual (such as parameter search) and the entire problem is naturally parallelizable
  - data parallel: parallelizing the processed data with the same procedure (ex: domain decomposition)
  - pipeline: each part of pipelined processes is dispatched to computational resource
  - master/worker: multiple workers and a management process

Example of EP

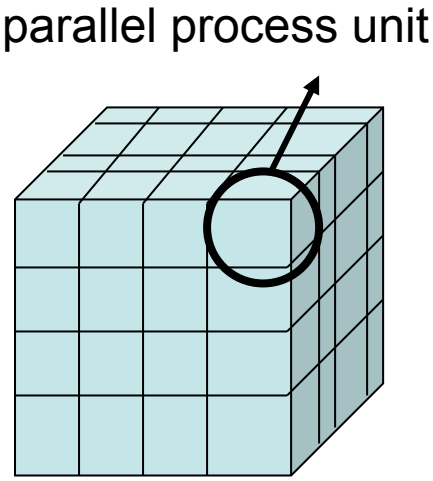
- Monte Carlo Simulation
  - Examining a number of cases with random parameters, then getting the result with statistical process
  - ex: calculating π with random numbers
    N pairs of *(x, y)* where $(0 \le x \le 1, 0 \le y \le 1)$
    When *C* is the number of pairs which satisfies $x^2 + y^2 < 1$,
    C/N is closing to *4/π*
  - The examination on each pair of (x, y) can be performed individually and simultaneously ⇒ completely in parallel
  - Finally, getting the summation of C from these parallel processes



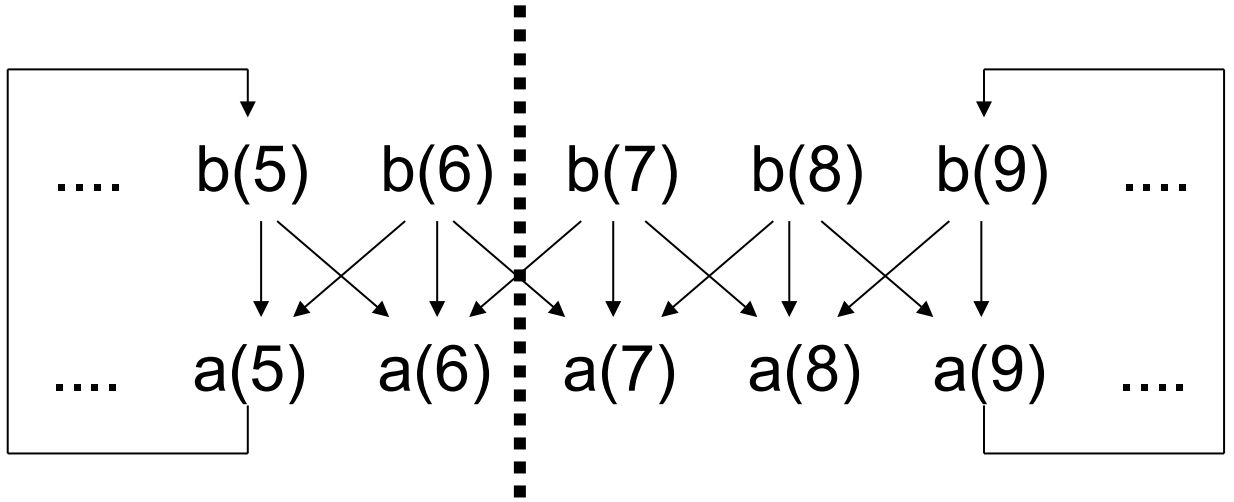
Example of data parallel case

- domain decomposition
  - Calculating points are uniformly distributed in some dimensions of space, and partitioning them into orthogonal blocks to be parallelized
  - There are some communication required to exchange data
    ex) for PDE with explicit method, surface points data are exchanged with neighbors
  - ex (1-dimension))

```
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];
}
```



domain decomposition (cont'd)



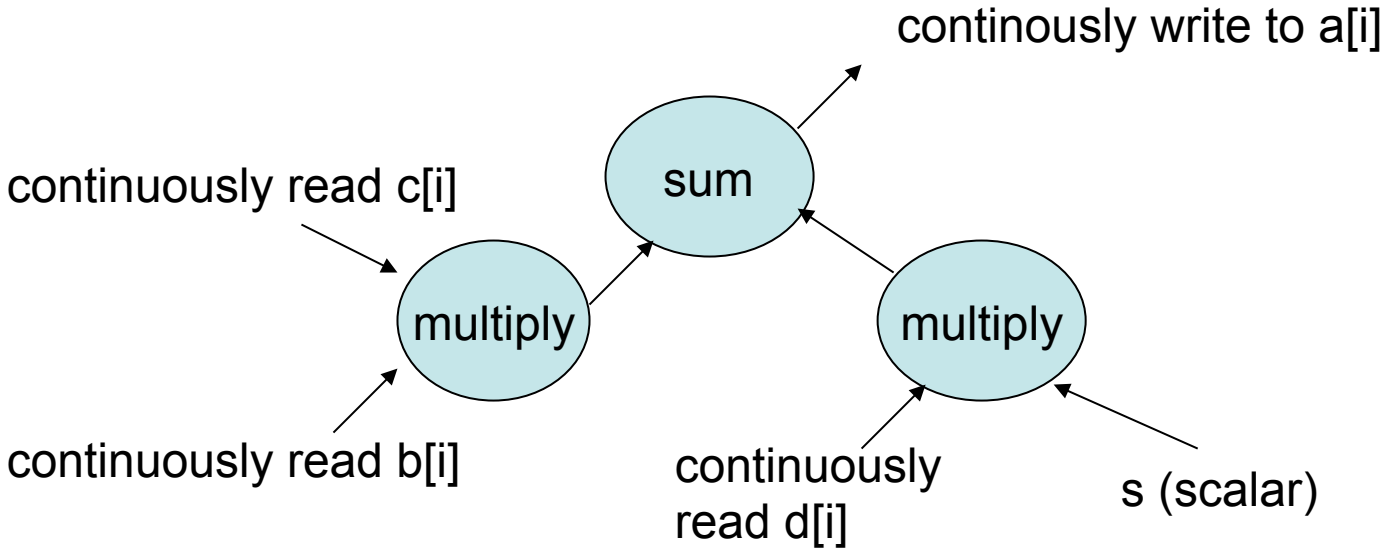
border of parallel processes

```
for(t=0; t < T; t++){
   for(i=0; i < N; i++)
      a(i) = b(i-1) + 2*b(i) + b(i+1);  // communication required
   for(i=0; i < N; i++)
     b(i) = a(i);                       // communication not required
}
```

Pipeline parallelism

- A set of data are processed in the same manner and same order, the entire data stream can be processed by each computation stage and these stages are connected
- Example of vector processing

```
for(i=0; i < N; i++){
    a[i]=b[i] * c[i] + s * d[i];
}
```



Master/worker parallelization

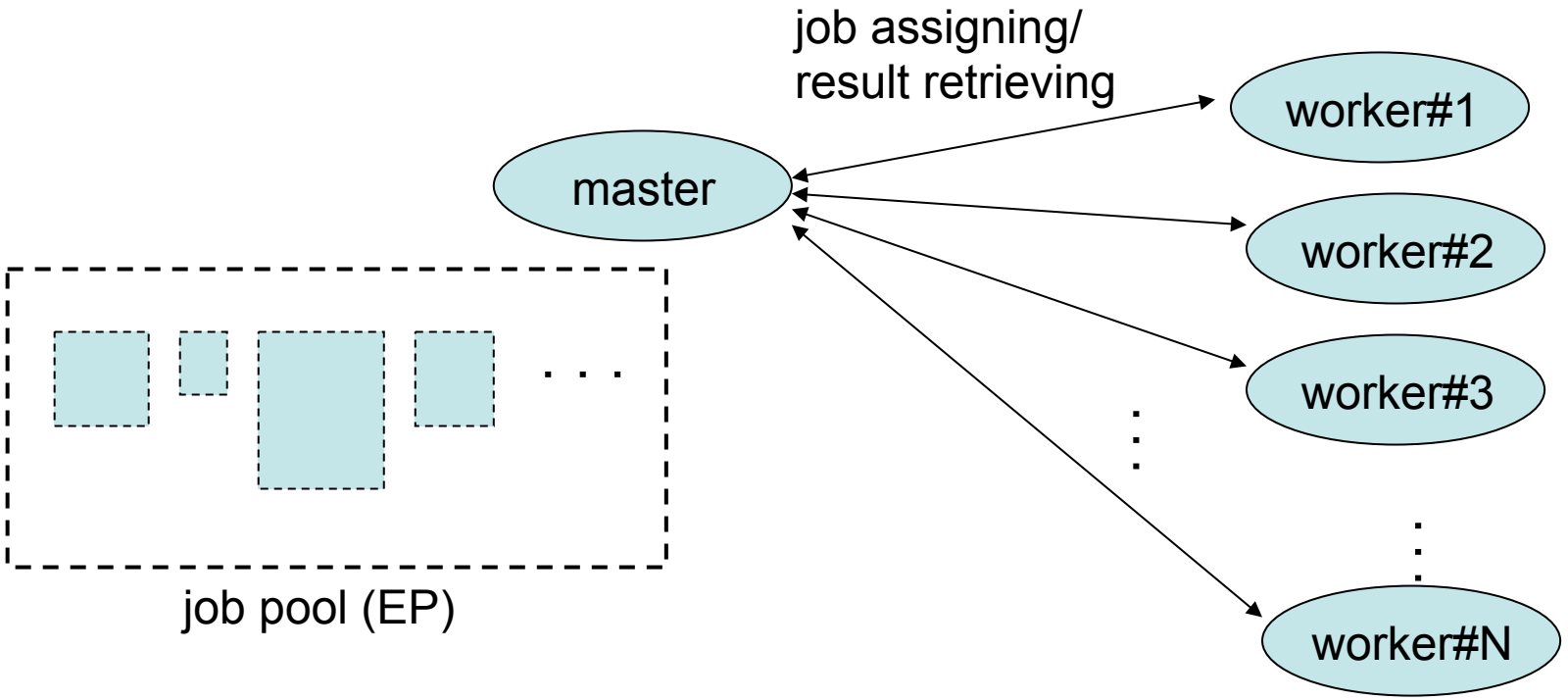
- One master and several workers exist, and the master maintains a “pool” of data to be processes (# of data sets >> # of workers)
- The master retrieves a set of data to dispatch to a worker, and repeats it while there is a data set in the pool
- A worker processes the dispatched data, and returns the result to the worker, then is assigned the next data

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

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

Master/Worker (cont'd)

- Especially effective when the loads of processes are not balanced and it is difficult to keep load balance
- Each process should be in EP manner



Communication and Synchronization

- All processes have some interaction with each other and need to communication at certain point in the processing
- These communication may be the overhead which is not required in sequential processing
  ex) surface data exchanging in domain decomposition
- Effect of **communication** to the computation efficiency
  - Time to be spent for communication itself:
    Overhead which does not exist in sequential process
  - Time for **synchronization (waiting)** to stall the process:
    Load imbalance causes the stall to wait for the communication partner
- It is required to minimize the overhead caused by communication and synchronization for efficient parallel processing

Communication patterns and costs

- It depends on the parallel processing architecture
  - Distributed Memory Architecture:
    each process explicitly communicates with each other by data sending/ receiving
    ⇒message passing (send, receive, ...)
  - Shared Memory Architecture:
    each process read/write the data from/to the shared address space
    ⇒shared memory access (write read, ...)
- Communication cost
  - For message passing, the distance and geometrical relation of communicating processes is important
    ⇒ “neighboring communication” (physically close distance) requires a low cost and less impact on entire system communication
  - For shared memory access, the geometrical relation of shared memory to be accessed is important
    ⇒ for NUMA architecture, the distance to memory differs
  - In both cases, the bottom-line hardware performance strongly affects on the performance

Cost of synchronization

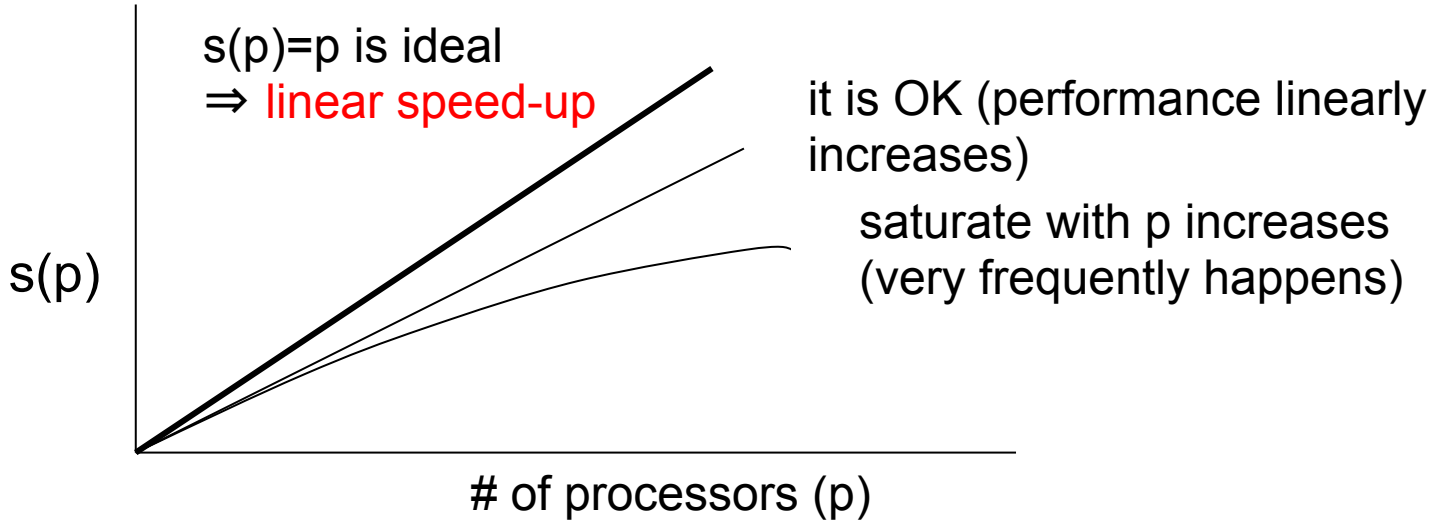
- Distributed memory system case
  - Network topology for parallel communication and synchronizing algorithm affect on the performance
  - System size (# of parallel processors) is essential
- Shared memory system case
  - Whether the hardware supports synchronization (primary) or not is important (ex: memory lock feature)
  - Process number to be synchronized is essential

Metric for parallel processing efficiency

- Most important purpose of parallel processing is SPEED
- It is strongly expected to reduce the time for solution when we introduce the parallel processing... but
- It happens that the actual speed does not increased (very often!)
- Especially, it will be the problem when the system size (# of parallel processes) increases
  ⇒ "**scalability" in parallel processing**
- The metric to examine the efficiency of parallel processing is important
- **degree of parallelism** is defined as:
  - parallelism in the problem: how much natural parallelism (or degree of parallelisms) exist in the problem
  - parallelism in the system: how much hardware resources (# of processors, etc.) exist in the system

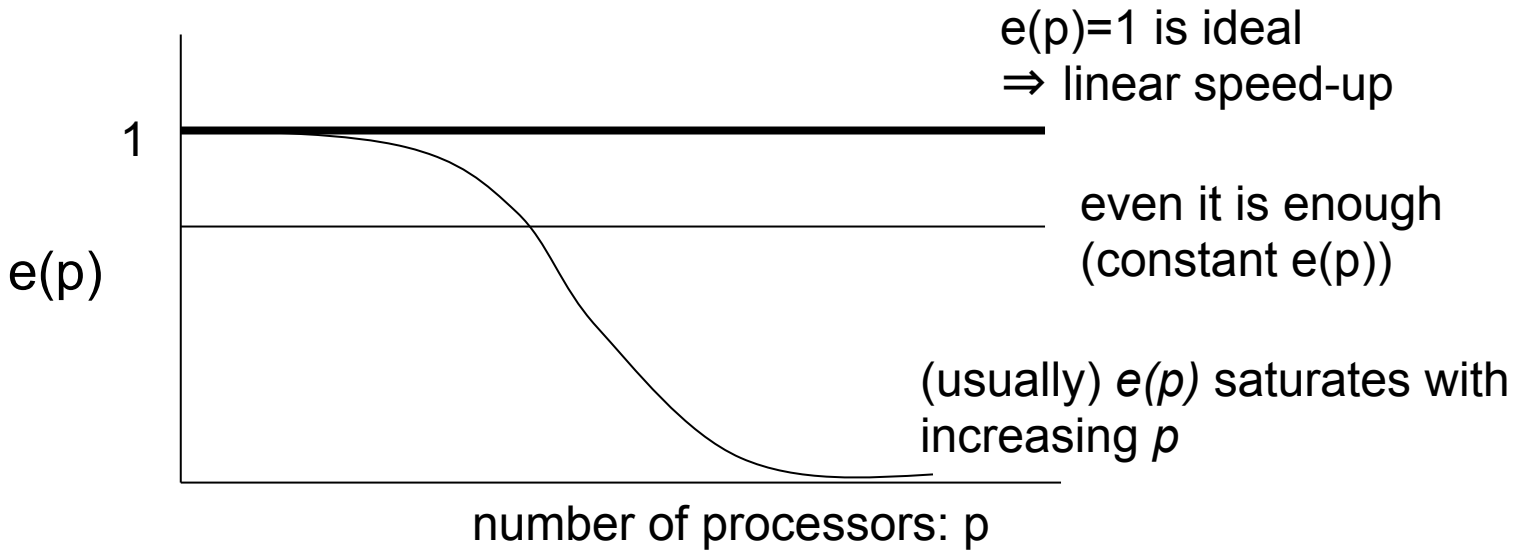
Parallel processing performance (1)

- Speed-up ratio
  - Let define the time required with one process as T_1
  - Let define the time required with *p processes as T(p)*
  - $s(p)=T_1 / T(p)$
    s(p) is called as "Speed-up Ratio with *p* processors"
    If *s(p) > 1, it means the speed is increased*
  - Ideally, $s(p)=p$ ⇒ "linear speed-up"
    (when *p* processors are used, there is a gain of *p* times)



Parallel processing performance (2)

- Parallel efficiency
  - Inconvenience of *s(p)*: it depends on *p*, and not an absolute value
  - "It is ideal when *s(p)=p"* ⇒ "How is it achieved ?" as the efficiency
  - *e(p)=s(p)/p ⇒ "linear speed-up" : e(p) = 1*
    e(p) does not depend on *p*, and it is better to achieve to 1 (normally it is lower than 1)



Amdahl's Law and parallel processing

- Amdahl's Law
  - "Process efficiency is determined just by the efficiency of inefficient part rather than the average efficiency of all parts"
- Amdahl's Law in parallel processing
  - Let assume the sequential execution time as T_1, and it can be devided into T_p as the part which can be parallelized and T_s as the part which cannot be (only executable as sequential process)

    $T_1 = T_p + T_s$
  - If the part for T_p is executed completely in parallel (ideally), the total execution time with p processors $T(p)$ is:

    $T(p) = T_s + T_p / p$
  - When p is infinite:

    $$\lim_{p \to \infty} T(p) = T_s$$

    and

    $$\lim_{p \to \infty} e(p) = T(p) / p = T_s / p = 0$$

    Thus **"although increasing the number of processors *p* toward infinite, the parallel efficiency *e(p)* becomes 0 with the bottleneck of T_s"**

Scalable Problems

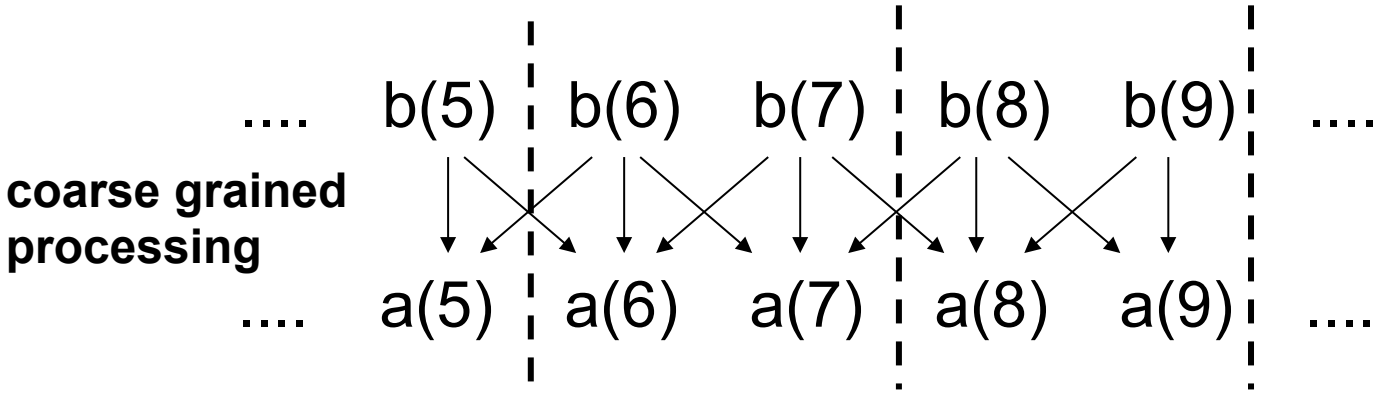
- Actually, there exists T_s in any problem, then the scalability of parallel efficiency is limited
- Does it pay to challenge “large scale parallel” or “massively parallel” actually ?
  ⇒ **It’s OK if we assume a problem with large portion of T_p where T_s is negligible**
  ⇒ **It is called as “scalable problem”**
- There are so many scalable problems in most of scientific computation, however it is always required to consider how T_s is large when we increase the system size p

- Another factor is how the communication/synchronization costs large because these parts also becomes the bottleneck like T_s
  ⇒ **granularity: how much is the “grain” of parallel execution ?**

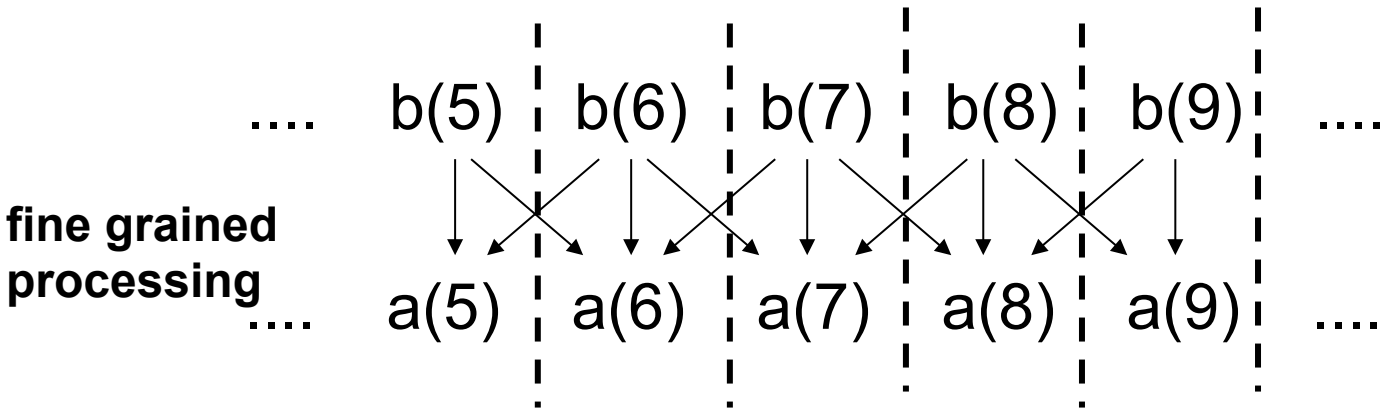
Granularity: parallel execution grain

- The dominant reason to decrease the parallel efficiency is the overhead for communication/synchronization
  - They are just the overhead which does not exist in sequential process
  - Synchronization causes some idling status without any processing
- It corresponds to the case where all other parallel processes stop, thus it seems to be a temporal sequential execution
  ⇒ Shorter time for this makes higher efficiency
- How long time is spent just for computation without communication/ synchronization ?
  - **Long ⇒ "coarse grain"**
  - **Short ⇒ "fine grain"**
- Naturally, when the parallelism increases for a certain size of problem, the granularity becomes fine
  ⇒ difficulty in scalability

Example in domain decomposition



- each process treats TWO points of computation
- exchanges TWO data sets with neighboring processes



- each process treats ONE point of computation
- exchanges TWO data sets with neighboring processes

Factors to determine parallel efficiency

- It is easy to achieve high efficiency when the problem size increases according to the system size increase
- The concept of granularity is relative, so it is required to examine the communication/synchronization overhead in the target application
  ⇒ ex) recording the wall clock time before/after the communication
- Always taking care of the parallel efficiency and resource utilization when the system size increases, based on the concept of granularity
- In some case, the application (algorithm) itself must be fine grained, and it is strongly required to care of the efficiency and trying to make it with coarser granularity
  ex) combining multiple communication in one
- However, also be careful of changing the communication pattern when combining the communications for coarser granularity

"Strong" vs "Weak" Scaling

- When increasing the system size:
  - If the problem size is fixed and only the system increases
    ⇒ ***"Strong Scaling" (fixed problem size)***
  - If the problem size is increased according to the system size increasement
    ⇒ ***"Weak Scaling" (fixed computation time)***
- It is relatively easy to achieve Weak Scalability
  - process granularity is not changed
  - only the communication complexity (communication pattern) is changed and system must have the capacity to accept it
- It is relatively hard to achieve Strong Scalability
  - process granularity is finer
  - the system must have the capacity to accept higher late of communication
    ⇒ ***shorter latency is required for communication/synchronization***

Load imbalance and parallel efficiency

- Load balance: ***It is ideal that the computation and communication amount for each process is equal***
  - ex) when taking the global synchronization, all processes are ready for it at the same time, and there is no time to wait
- For domain decomposition on uniformly distributed data, the load is naturally balanced
- How the load imbalanced:
  - Non-uniformly distributed domain decomposition
  - In parameter search, each job execution time depends on its parameter

Problem decomposition caring load imbalance

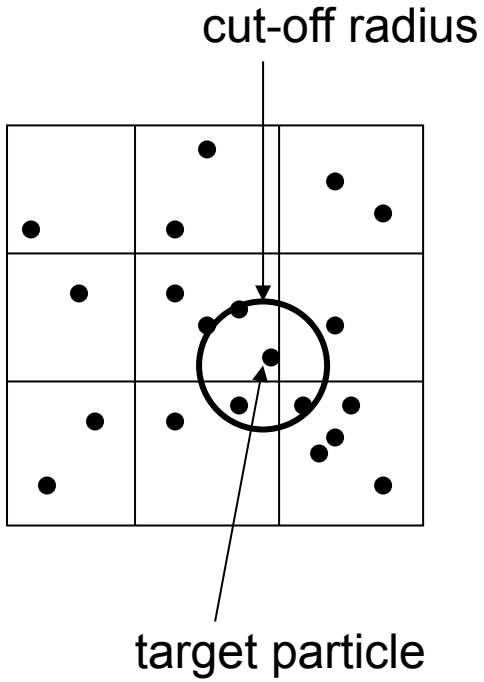
- Basically, it is ideal in domain decomposition to partition the problem space in large chunk for coarse grained processing
- For nonuniform problem space, the coarse grained decomposition causes the load imbalance
- It is important in some case to partition the problem space regardless the shape, to achieve load balancing
  ⇒ It may cause another imbalance in communication (pattern)
  ⇒ It may cause finer granularity
- These effects may trade-off with each other

Example: MD with cut-off

- MD (Molecular Dynamics)
  - There are p particles in n-dimension and simulating inter-particle force
  - The potential energy is not like Coulomb potential, but with rapidly decreasing potential according to the distance
    ⇒ "well" style potential and can be processed with "cut-off"

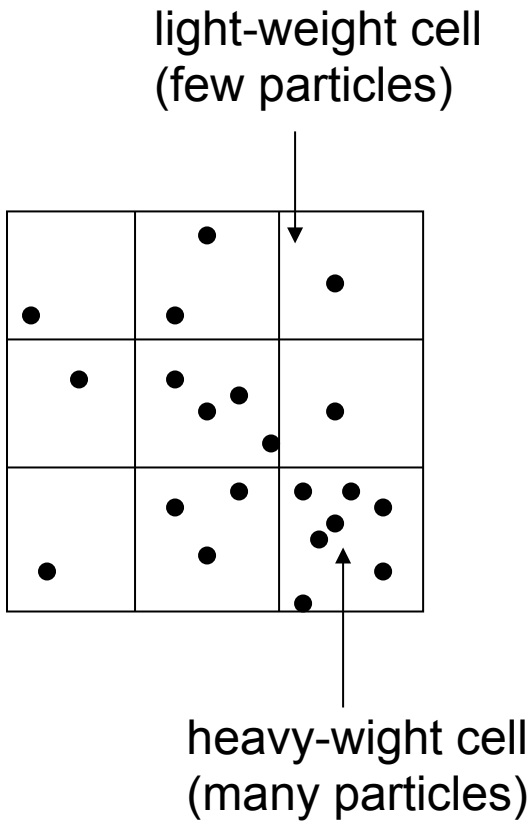
$$f(r) = \alpha \frac{1}{r^6} - \beta \frac{1}{r^{12}}$$

- It is possible to reduce the communication amount to limit the partners in certain domain
  - Partitioning the space in "cells" as like as domain decomposition, and processing the particles in a cell by a process
    ⇒ if the size of cell is set *"slightly larger than cut-off radius"*, it is necessary to communicate with neighboring cells only



MD with cut-off (cont'd)

- Particles move around in the time step by the interaction with other particles
- Load balance may be broken while the computation proceeds in cell mapping manner
- It is required to keep the number of particles within a cell to keep the load balance
- Methods
  - 1) remapping the cells to parallel processes according to the density of particles per cell, in every certain steps
  - 2) if the number of cells largely exceeds the number of processes, mapping the cells to processes in cyclic manner
  - 3) giving up cell-mapping method, and directly mapping the particles to processes



MD with cut-off (cont'd)

- Method-1)
  When remapping the cell to process, a large amount of data should be moved, and also the neighboring cells are not mapped to the neighboring processes
- Method-2)
  Cyclic mapping of cells to processes realizes easy load balancing, but the neighboring cells are not mapped to the neighboring processes (always) to cause long distance communication
- Method-3)
  It is required to refer the table of particle to process mapping, and the communication distance may so long
  ⇒ There is no best way in any case
- It depends on the problem characteristics (ex: how the particles concentrate/distribute, potential energy, time length), and there is no general best solution
- For heavy load imbalancing, it pays dealing even with heavy communication cost

Summary

- Importance of parallel processing on large scale scientific computation
- Parallel processing methodology and parallelization scheme
- Concept of speed-up and efficiency on parallelization
- Communication cost
- Amdahl's Law and scalability
- Scalability and granularity
- Load balancing