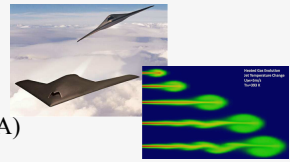
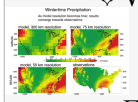
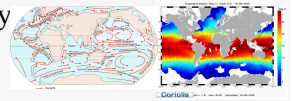
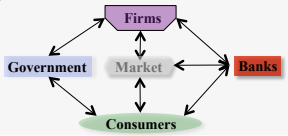
Parallel Computing - Introduction

- Parallelism
  - Ubiquity of parallelism
  - Single CPU - multi-tasks: shared resources
  - Hardware optimization
  - Multiple CPU: Distributed OS; Parallel computing
- Parallel computing
  - Goals: speed
    - Solve pb. in less time, solve larger pb., use of more precise model
  - New issues
    - Parallel tasks identification, pb. decomposition, tasks synchronization, communication, optimization, new bug types (deadlock, interference), starvation…

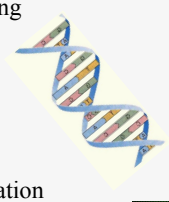
Parallel Computing

- Application areas
  - Simulation
    - Aerospace/Defense
      - Simulation of fluid flow (ONERA)
      - Nuclear weapons simulation
    - Weather forecast/Oceanography
      - Temperature, humidity, wind…
      - Oceanic circulation (IFREMER)
    - Economic & finance
      - Agent based models simulation

Parallel Computing

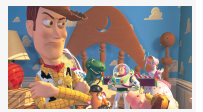
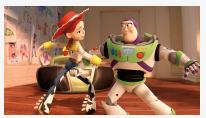
- Data analysis
  - Aerospace
    - Spatial image processing
  - Bio-info
    - Genome sequence
  - Cyber security
    - 10 000 person organization
      - 8.7 TB data stored each day
      - 6 GB streamed data each minute
  - Web services, big data …


Alex Kent (Los Alamos National Lab)
Cyber security defense using HPC

Parallel Computing

- Computer animation
  - Toy Story (1995)
    - 100 dual processors Sun renderfarm
  - Toy Story 2 (1999)
    - 1400 processors Sun renderfarm
  - Toy Story 3 (2010)
    - >3500 processors


Parallel Computing

- Objective
  - Use more than one process/thread to solve a problem
- Lectures
  - Chapter 1: Parallel computers
  - Chapter 2: Parallel computing - overview
  - Chapter 3: Communication between tasks
  - Chapter 4: Design of parallel programs
  - Chapter 5: Cases study
- Practicals
  - OpenMP, MPI

References

- A. Legrand et Y. Robert, *Algorithmique Parallèle*, Dunod, Paris, 2003.
- J. Duato, S. Yalmanchili and L. Ni, *Interconnection networks --- an Engineering Approach*, IEEE Computer society, Los Alamitos, California, 1997.
- Ian Foster, *Designing and Building Parallel Programs*, Addison-Wesley, 1995, http://www.mcs.anl.gov/dbpp.
- Georg Hager, Gerhard Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, Chapman & Hall / CRC Computational Science, 2010.

References

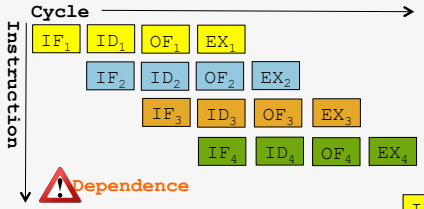
- B. Chapman, G. Jost, R. van der Pas, *Using OpenMP,* The MIT Press, 2008.
- W. Gropp, E. Lusk and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, The MIT Press, 2nd edition, 1999.
- W. Gropp, E. Lusk and R. Thakur, *Using MPI-2: Advanced Features of Message-Passing Interface*, The MIT Press, 1999.
- https://www.**mpich**.org/; https://www.**open-mpi**.org/
- D.B. Kirk, W.W. Hwu, *Programming Massively Parallel Processors*, Elsevier, 2010. *(CUDA)*

Chapter 1. Parallel Computers

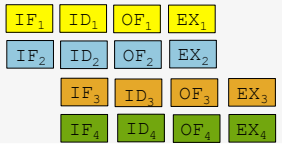
- Why
  - Request of computational power
  - Cost of the development of faster processors
  - Physical limits: floor print size, frequency/power consumption, cooling needed…
  - Faster → more power consumption to do so
  - **Conclusion:** Parallel computer is an more efficient solution than the development of faster processor

Advanced Techniques on Processor Design

- Scalar processor: *a*, *b*: real numbers a x b = axb
- Pipelined functional units: instruction/arithmetic pipeline



Floating point addition pipeline
Example: $3.5*10^3 + 9.7*10^4$
- Compare: 3<4
- Shift: $0.35*10^4$
- Add: 0.35+9.7=10.05
- Normalized: $1.005*10^5$



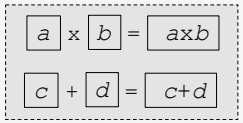
- Superscalar processor: multiple arithmetic/logic units

Advanced Techniques on Processor Design

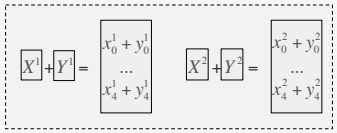
- Vector processor: *X*, *Y*: 5d vertices

$$X + Y = \begin{bmatrix} x_0 + y_0 \\ \dots \\ x_4 + y_4 \end{bmatrix}$$

- Multi-cores processor



scalar processor



vectoriel processor

- Hyper threading

Classification of Flynn

- Depends on
  - Flux of instruction and data

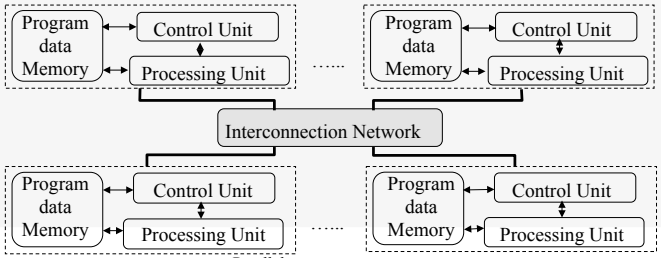
Instruction flux \ Data flux	Single	Multiple
Single	SISD	SIMD
Multiple	MISD	MIMD

Classification of Flynn

- Classification
  - SISD: 1 instruction works on 1 data; serial computer
  - SIMD: 1 instruction works on multiple data; vector processor, processor array, GPU
  - MISD: multiple instructions work in pipeline
  - MIMD: multiple instructions work asynchronously on multiple data;

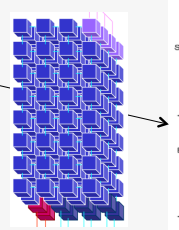
MIMD Architecture

- Characteristics
  - Asynchronous operation on multiple processors
  - Communication way: Shared or Distributed memory
- Distributed Memory Architecture: DM-MIMD
  - Private memory access

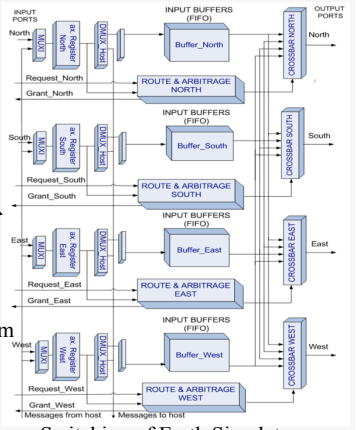


MIMD Architecture

- Characteristics
  - Interconnection networks
  - Swithing
  - Routing
  - Memory access
  - ...



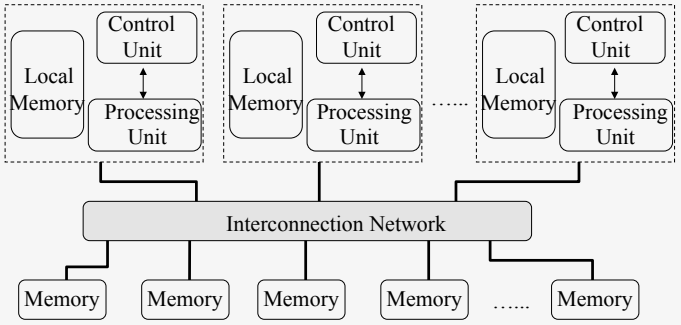
Interconnection of Red Storm



Switching of Earth Simulator

MIMD Architecture

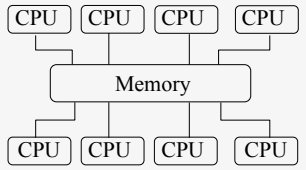
- Shared Memory Architecture: SM-MIMD



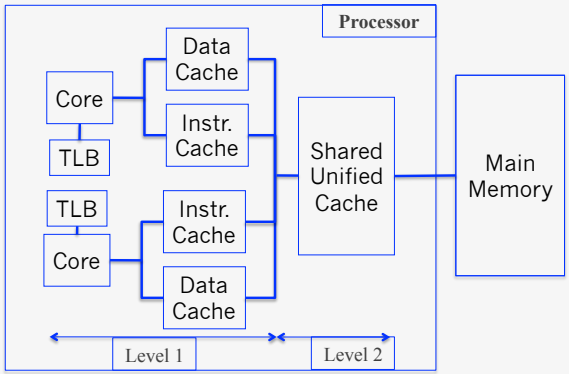
  - Memory accessible by all processors

MIMD Architecture

- SMP (Symmetric Multi-Processor)
  - Identical processors (2-64)
  - Shared memory
    - UMA / NUMA
  - Cc-NUMA
    - if one processor updates a location in shared memory, all the other processors know about the update
    - Example: SGI Origin 3800

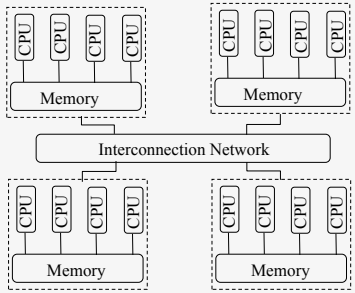


Memory of a processor



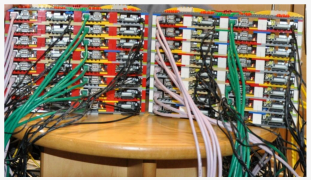
MIMD Architecture

- Cluster of SMP
  - Hybrid memory
  - Shared memory within each SMP
    - Shared memory access
  - Distributed memory between SMP
    - Message passing



MIMD Architecture

- Clusters
  - Collection of computers interconnected by :
    - a local network (Gigabit Ethernet)
    - or a dedicated interconnection network (infiniband)
  - Slower inter-processors communication
  - Distributed computing
  - MPI


Raspberry Pi – University of Southampton


Beowulf project – NASA 1994

MIMD Architecture

- SM-MIMD
  - Easy and quick parallel programming
  - Parallelization with compiler
  - Limitation of processors number
  - Programming library: OpenMP, pthread
- DM-MIMD
  - Very-large number of processors
  - Parallel programming difficult
  - Programming library: MPI

MIMD architecture – Example

- Sequoia – BlueGene/Q (TOP 1, June 2012; 4, Nov. 2016)
  - Soc custom: chip→module→node→ rack→system
  - IBM Power BQC 16C (1.60 Ghz), 1 572 864 cores
  - Network: 5-D torus – 2 GBps/link/direc., 2,5 μs latency
  - 1 GB Control Network (Boot, Debug, …)
  - Memory: 1 572 846 GB (SM within each node 16GB)
  - 8 I/O compute cards / rack
  - Lightweight proprietary kernel

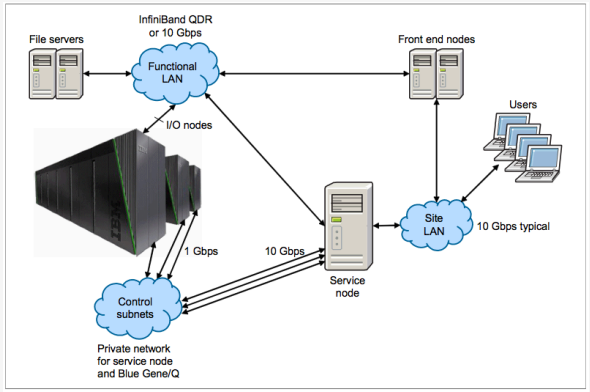
  - Cooling: water (90%, BQC), air (10%, I/O cards)

Sequoia --- BlueGene/Q

- Performance
  - Rpeak (theoretical): 20 132.7 TFlops
  - Rmax (Linpack): 16 324.8 TFlops
  - Power: 7 890.00 kW
- Softwares
  - Linux environment: gnu (glibc, pthreads, gdb…)
  - Fortran, C, C++, MPI, OpenMP, ESSL
- Location
  - DOE/NNSA/LLNL, USA

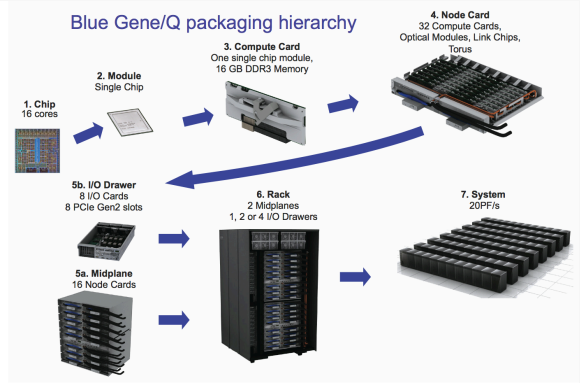
Sequoia --- BlueGene/Q

- System overview



Sequoia --- BlueGene/Q

- Soc Based

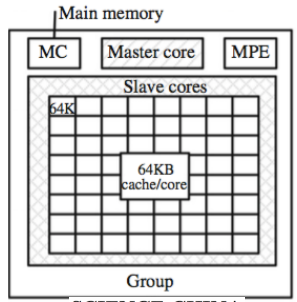


MIMD architecture – Example

- Sunway MPP –(TOP 1, 2016, TOP 3 en Green500)
  - Proc.: Sunway SW26010 260C (1.45 Ghz), 10 649 600 cores;
  - Perf.: Rmax-93 014.6 Tflops/s; Rpeak-125 436 Tflops/s
  - Memory: 1 310 720 GB
  - System: Linux like
  - C / C++ /Fortran
  - MPI / OpenMP / OpenACC
  - NSCC-Wuxi – China

Sunway MPP

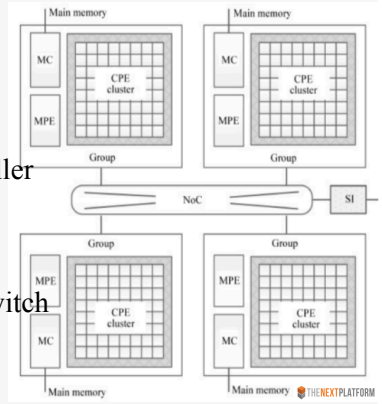
- Core Group
  - 1 Management PE + cache
  - 64 Computing PE + cache
  - PE:
    - 64 bits RISC
    - vector instruction
  - 1 Memory Controller



SCIENCE CHINA
Information Sciences

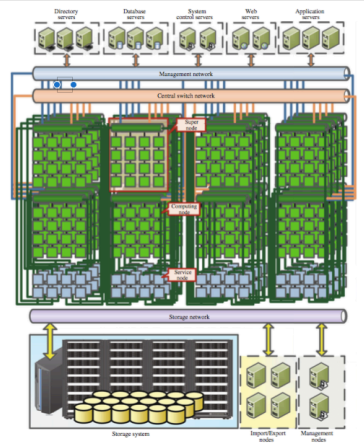
Sunway MPP

- Node (processor)
  - 4 core groups
  - 32MB memory
  - system interface
  - 1 node management controller
- Super node
  - 256 nodes
  - fully-connected crossing switch
- Cabinet
  - 4 super nodes

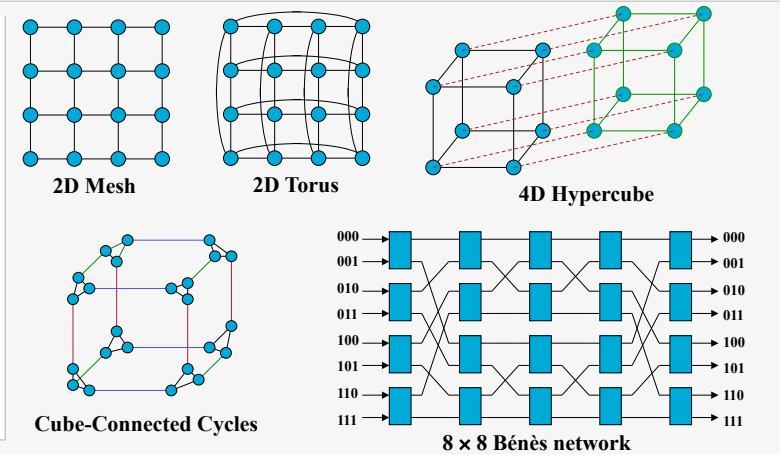


Sunway MPP

- System
  - 40 cabinets
  - Central switching network
    - between super nodes
  - Super node network
  - NoC
  - Diameter: 7



Interconnection Networks --- Examples



Direct Networks

- Basic properties: Topology
  - *Node degree*: number of channels connecting this node to its neighbors
  - *Diameter*: the maximum distance between two nodes in the network
  - *Regularity*: all the nodes of a network have the same degree
  - *Symmetry*: a network looks alike from every node
- Physical limits
  - Length, number and intersection of links

Direct Networks

- Characterization
  - **Latency** is the time elapsed between the generating of a 0 byte message at a node and its delivering to another node
  - **Bandwidth** is the amount of information that can be delivered on a link per unit of time
  - **Scalability** and partitionability
  - **Reliability** and reparability

Evaluation of parallel computer

- Test programs for performance
- Macro benchmark: PARKBENCH, SPEC, …
  - Procedures, Applications
  - Global performance
- Micro benchmark: STREAM, LINPACK
  - Procedures
  - CPU, memory, I/O…
- LINPACK: linear algebra; HPLinpack - HPL
- Other than linear algebra? HPC Challenge Benchmark

Chapter 2. Parallel Computing – Overview

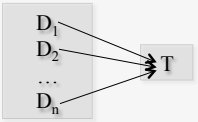
- Parallelism - What
  - Parallelism refers to the simultaneous occurrence of events on a computer
- Parallelism - Sources
  - Do simultaneously several works
  - Repeat one action on multiple data
  - Work in pipeline

Multiple jobs

- Arithmetical operation
- Logical operation
- Accessing memory
- I/O

Types of Parallelism

- Control parallelism → Functional decomposition
  - Decomposition of the computation
  - ≠ codes <-> process/thread
  - T -> T_1; T_2; T_3; T_4 …
- Data parallelism → Data decomposition
  - Same computation on different data
  - Partition of the data
  - ≠ data <-> process/thread
- Pipeline



Parallelism - Example

- Computation on polynomial vectors

(book « Initiation au parallélisme, concepts, architectures et algorithmes », Marc GENGLER, Stéphane UBÉDA & Frédéric DESPREZ)

for i=0 to n−1 do
  $w[i] = a+b.v[i]+c.v[i]^2+d.v[i]^3+e.v[i]^4+f.v[i]^5+g.v[i]^6$
end for

Parallelism – Example

- Data parallelism
  - Same computation on different data

for i=0 to n−1 do in parallel
  $w[i] = a+b.v[i]+c.v[i]^2+d.v[i]^3+e.v[i]^4+f.v[i]^5+g.v[i]^6$
end for

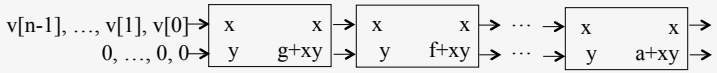
Parallelism – Example

- Control parallelism
  - Different computation

```
for i=0 to n−1 do
    parallel tasks
        x=a+b.v[i]+c.v[i]^2+d.v[i]^3
        y=e+f.v[i]^1+g.v[i]^2
        z=v[i]^4
    end parallel tasks
    w[i] = x+y.z
end for
```


Parallelism – Example

- Pipeline
  - 7 stages for 1 data stream



Parallel Computing --- Software

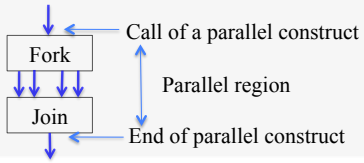
- Scalar processor

```
sum=0;
for (i=0; i<n; i++) sum += X[i]*Y[i];
```

- Vector processor

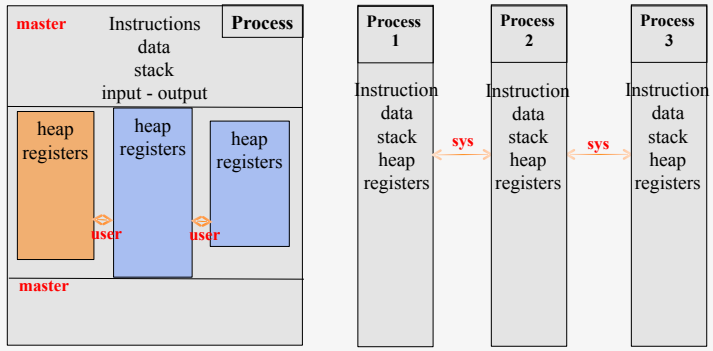
  compiling with option `-vec` $X \times Y = \begin{bmatrix} x_0 \times y_0 \\ \dots \\ x_4 \times y_4 \end{bmatrix}$

- Multi-cores processor / shared memory: OpenMP



Parallel Computing

- Tasks - multithreading - multiprocessing



Parallel Computing --- Software

- Multi-cores processor / shared memory: OpenMP

```
#include "omp.h"
#define V_SIZE_GL 100/* export OMP_NUM_THREADS=10    */
double sum=0.;       /* or omp_set_num_threads(10);  */
int main(int argc, char *argv[])
{
    double a[V_SIZE_GL], b[V_SIZE_GL];  /* shared */
    int i;
    for (i=0; i<V_SIZE_GL; i++) {
        a[i] = i*0.5;b[i] = 2.0;
    }
    #pragma omp parallel for reduction(+:sum)
    for (i=0; i<V_SIZE_GL; i++) sum += a[i] * b[i];
}
```

Compiling:
gcc –fopenmp prog.c –o prog

Parallel Computing --- Software

- Multi-cores processor: multi-threading

```
/* Compiling : gcc -pthread ...
    typedef unsigned long int pthread_t; */

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define V_SIZE_LC 25

pthread_mutex_t my_mutex;
double          sum = 0.0;
```


Parallel Computing --- Software

- Multi-cores processor: multi-threading

```
/* Function performed by each thread */
void *reduction(void *arg) {
    unsigned long my_num = (unsigned long)arg, i;
    double        my_tab[V_SIZE_LC];
    double        my_sum = 0.;

    for (i=0; i<V_SIZE_LC; i++) {
       my_tab[i] = my_num*V_SIZE_LC + i; /* Initialization */
       my_sum += my_tab[i];       /* Compute local sum */
    }

    pthread_mutex_lock(&my_mutex);
    sum += my_sum;               /* Add to global sum */
    pthread_mutex_unlock(&my_mutex);
}
```


Parallel Computing --- Software

- Multi-cores processor / shared memory: multi-threading

```
int main(int argc, char **argv) {
    unsigned long nb_thread=4, t;
    pthread_t    *tab_th;
    void         *ret; /* 1st arg.->nb. of threads */

    if (argc > 1) nb_thread = atoi(argv[1]);

    tab_th=(pthread_t *)malloc(nb_thread*sizeof(pthread_t));
    if (!tab_th) {
        fprintf(stderr, "malloc error for tab_th\n");
        return 1;
    }

    pthread_mutex_init(&my_mutex, NULL);
```

Parallel Computing --- Software

- Multi-cores processor / shared memory: multi-threading

```
    ……
    for (t=0; t<nb_thread; t++) {
        if (pthread_create(tab_th+t, NULL, reduction,
(void *)t) < 0) {
            fprintf(stderr, "pthread_create error for
write_process\n"); exit(2);
        }
    }

    for (t=0; t<nb_thread; t++) {
        (void) pthread_join(tab_th[t], &ret);
        if (ret) free(ret);
    }
    printf("sum = %f\n", sum);
    free (tab_th);
    return 0;
}
```

FIN CM2

Parallel Computing --- Software

- Distributed memory processing: MPI

```
#include <stdlib.h>
#include <stdio.h>
#include "mpi.h"

#define V_SIZE_LC 25

int main(int argc, char **argv) {
    int  i, taskid, /* task ID-also used as seed number */
         numtasks,  /* number of tasks */
         mtype=10;  /* message type */

    MPI_Status status;

    double my_tab[V_SIZE_LC],
           my_sum=0, /* local sum calculated by current task */
           sum=0;    /* global sum */
```


Parallel Computing --- Software

- Distributed memory processing: MPI

```
    ……

    MPI_Init(&argc,&argv);

    /* Obtain number of tasks and task ID */
    MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD,&taskid);

    for (i = 0; i < V_SIZE_LC; i++) {
        my_tab[i]=taskid*V_SIZE_LC+i;/* Initialization */
        my_sum += my_tab[i];/* Computation of local sum */
    }
```


Parallel Computing --- Software

- Distributed memory processing: MPI with send/recv

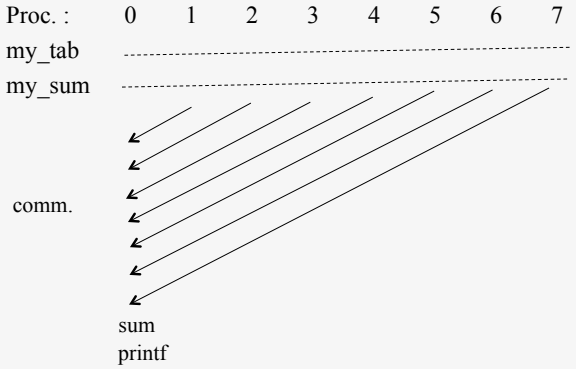
```
    ……
    if (taskid) /* workers-process other than 0 */
        MPI_Send(&my_sum, 1, MPI_DOUBLE, 0, mtype,
                 MPI_COMM_WORLD);
    else { /* Master receives messages from all workers */
        double received_sum;
        sum = my_sum;
        for (i = 1; i < numtasks; i++) {
            MPI_Recv(&received_sum, 1, MPI_DOUBLE,
unorganized              i, mtype,
communication !          MPI_COMM_WORLD, &status);
            sum += received_sum;  } /* for */
        printf(" Global sum = %f\n", sum); } /* else */
    MPI_Finalize();
    return 0; } /* main */
```

Compiling:
mpicc prog.c –o prog

Parallel Computing --- Software

- Run the program with 8 process

Running:
mpiexec –np 8 ./prog



Parallel Computing --- Software

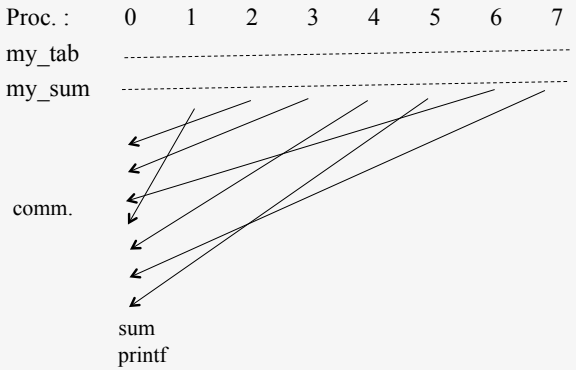
- Distributed memory processing: MPI with send/recv

```
……
if (taskid) /* workers-process other than 0 */
    MPI_Send(&my_sum, 1, MPI_DOUBLE, 0, mtype,
             MPI_COMM_WORLD);
else { /* Master receives messages from all workers */
    double received_sum;
    sum = my_sum;
    for (i = 1; i < numtasks; i++) {
        MPI_Recv(&received_sum, 1, MPI_DOUBLE,
                 MPI_ANY_SOURCE, mtype,
                 MPI_COMM_WORLD, &status);
        sum += received_sum;  } /* for */
    printf(" Global sum = %f\n", sum); } /* else */
MPI_Finalize();
return 0; } /* main */
```

unorganized communication !

Parallel Computing --- Software

- Run the program with 8 process



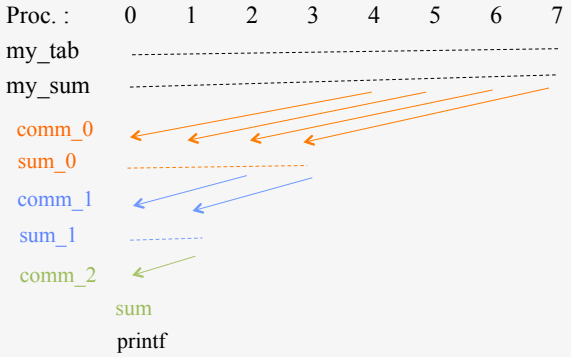
Parallel Computing --- Software

- Distributed memory processing: MPI with collective comm.

```
……
int main(int argc, char **argv)
{
    ……
    MPI_Reduce(&my_sum, &sum, 1, MPI_DOUBLE,
               MPI_SUM, 0, MPI_COMM_WORLD);
    if (taskid==0)
        printf("Global sum = %lf\n", sum);
    MPI_Finalize();
    return 0;
}
```

Parallel Computing --- Software

- Run the program with 8 process



Parallel Computing --- Software

- Hybrid programming:
  MPI / OpenMP, multi-threading
  - Cluster of SMP nodes
  - Shared memory: OpenMP + Distributed memory: MPI
  - Reduce MPI communication overhead
    - Use threads to limit number of MPI process
    - Reduce cross node communication
  - Minimize memory consumption
    - Replicated data in MPI replaced by data in shared memory

Parallel Computing --- Software

- Hybrid programming:
  MPI / OpenMP, multi-threading
  - Overlapping communication and computation:
    - Use separate communication / computation thread

```
if ( /* communicating thread */ )
{
        MPI_Send/Recv....
}
else {
       /* computation thread */
}
```

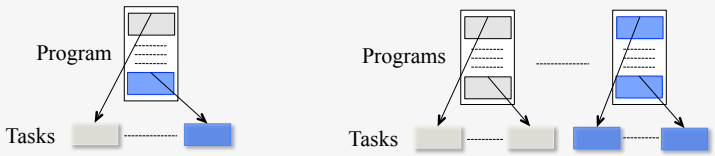
  - Add levels of parallelism
  - Optimal rate between the number of threads and process

Parallel Computing --- Software

- Hybrid programming:
  MPI / OpengMP, multi-threading
  - Compiling:
    `mpicc -openmp, -xopenmp, -mp, -qsmp=openmp`
    - Set `OMP_NUM_THREADS`
    - Link with thread safe MPI library
  - Run:
    `mpirun -n <#nodes> -np <#MPI procs per node> a.out`

Parallel Computing --- Software

- Programming models
  - SPMD
  - MPMD



Times of a parallel program

- Times
  - Computation time: T_{comp}
  - Communication time: T_{comm}
  - Time for processes synchronization & management: T_{other}
- Wall clock time: time elapsed between the end and the beginning of a process
  - Include all times: `$ time my_prog`
  - Internal measurement is more accurate than the external one: `start=time(); … texec=time()-start;`
  - Parallel library functions are more accurate than the sequential ones: `MPI_Wtime(); omp_get_wtime();`

Granularity of parallel program

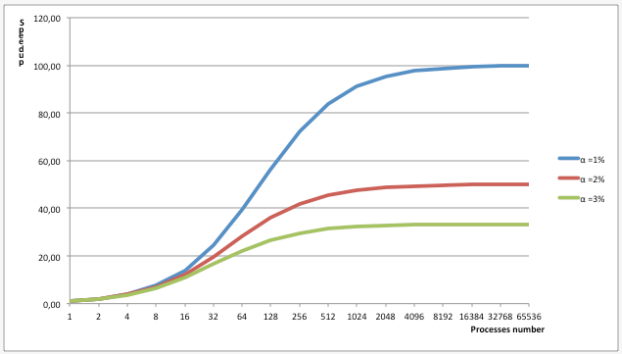
- Definition : grain=T_{comp}/T_{comm}
- Fine-grain
  - Small T_{comp}/T_{comm}, High communication overhead
  - May affect the performance of parallel program
  - Ease load balancing
- Coarse-grain
  - Large T_{comp}/T_{comm}, Communication cost ↘
  - Control depth ↘
  - Increase the performance of parallel program
  - Difficult to apply load balancing strategies

Performance Evaluation

- Amdhal's laws
  - N: number of process; M: size of problem
  - *Second law*: speedup $S_N(M) = \frac{T_1(M)}{T_N(M)} \le N$
    Sequential execution of the parallel program
    The best sequential algorithme?
    Same compiler
    Same computer?
  - *First law*: α the percent of sequential part of program $S_N(M) \le S_N = \frac{1}{\left(\alpha + \frac{(1-\alpha)}{N}\right)}$ Upper bound of speedup
  - Efficiency: $E_N(M) = \frac{S_N(M)}{N}$ Measurement of process activity

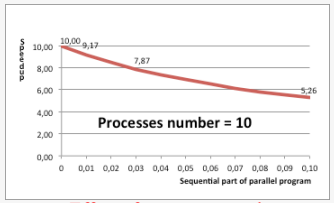
Performance Evaluation

- Amdhal's first law

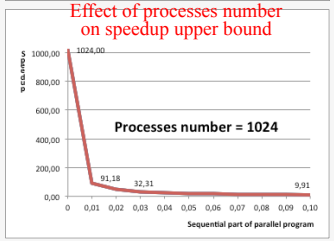


Performance Evaluation

- Effect of Amdahl's law
  - S: speed-up
  - α: percent of serial code
  - α =1%
    - S=9.17 when N=10
    - S=91.2 when N=1024
- Q
  - E=80%, N= 10000 → α?



Effect of processes number on speedup upper bound



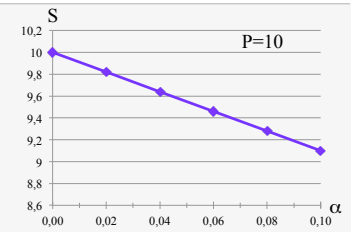
Performance Evaluation

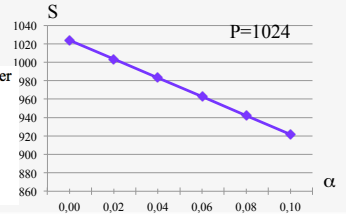
- Amdahl's laws don't take into account
  - Time for communication
  - Time for processes synchronization & management
- Amdahl's laws predict limited performance of parallel program for big processes number
- The facts
  - Good performance en Massively parallel computers
- What happened?

Performance Evaluation

- Gustafson's law
  - S(P) = P - α(P-1)
  - P: number of processors
  - S: speedup
  - α: no-parallelizable part of the process
  - Reasoning:

a + b execution time of the program on parallel computer
The execution time on a sequential computer is a + P.b
S =(a+Pb)/(a+b)
denote alpha=a/(a+b),
S=alpha + P.(1-alpha)) = P – alpha(P-1)





Performance Evaluation

- Gustafson's law
  - α =1%