CSC4005
Parallel Programming
Tutorial 2

Yangzhixin Luo, 119010224@link.cuhk.edu.cn

Outline of Tutorial 2

- How to connect to the server
- Secure your code
- How to build MPI projects
- How to run programs on multiple nodes
    - Interactive Approach
    - Batch Approach
    - Cancelling
    - Useful Commands
- MPI, OpenMP, pthread, and CUDA
- How to conduct experiments and write reports

How to connect to the server

ssh STUDENT_ID@10.26.200.21

```
(base) laureline@Laureline ~ % ssh 119010355@10.26.200.21
119010355@10.26.200.21's password:
Last login: Sat Sep 17 01:21:48 2022 from 10.30.120.43
[119010355@csc4005_slurm_master ~]$
```

Remarks:

This is just a test account. Accounts will be distributed to every student individually. Please use the account you are assigned with to run your programs.

Secure your code

Only put your code in the following directories:

- **/home/STUDENT_ID**
  - For use in the main node only
- **/nfsmnt/STUDENT_ID**
  - For use in all nodes

Or other users might be able to read your stuffs!

How to build MPI projects

Instead of gcc/clang/g++/clang++, Use:

- mpicc
- mpic++
- mpicxx

Demo

mpic++ mpi_hello.cpp -o mpi_hello

How to run programs on multiple nodes

Submit a job to Slurm:

- Interactive Approach
- Batch Approach

Interactive Approach

You can start an interactive session via **salloc**

The following command, for example, will prepare a session with 8 cores distributed on one node in the Debug partition (default) and the session can last for 5 mins:

salloc -N1 -n8 -t5

If you are acquiring resource more than allowed, then your shell will hang there.

Interactive Approach

Two partitions:

- Debug partition (default)

salloc -N1 -n8 -t5 -p Debug

- Project partition

salloc -N4 -n48 -t5 -p Project

```
[119010224@node21 mpi_demo]$ scontrol show partition
PartitionName=Project
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
   AllocNodes=ALL Default=NO QoS=N/A
   DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=4 MaxTime=00:05:00 MinNodes=0 LLN=NO MaxCPUsPerNode=UNLIMITED
   Nodes=node[22-29,31-40]
   PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO OverSubscribe=NO
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=720 TotalNodes=18 SelectTypeParameters=NONE
   JobDefaults=(null)
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
   TRES=cpu=720,mem=720000M,node=18,billing=720

PartitionName=Debug
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
   AllocNodes=ALL Default=YES QoS=N/A
   DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=1 MaxTime=01:00:00 MinNodes=1 LLN=NO MaxCPUsPerNode=8
   Nodes=node[01-09]
   PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO OverSubscribe=NO
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=360 TotalNodes=9 SelectTypeParameters=NONE
   JobDefaults=(null)
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
   TRES=cpu=360,mem=360000M,node=9,billing=360
```

Interactive Approach

sinfo

```
[119010224@node21 mpi_demo]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
Project      up       5:00     18   idle node[22-29,31-40]
Debug*       up    1:00:00      9   idle node[01-09]
```

```
[119010224@node21 mpi_demo]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
Project      up       5:00      1    mix node23
Project      up       5:00      1  alloc node22
Project      up       5:00     16   idle node[24-29,31-40]
Debug*       up    1:00:00      1    mix node01
Debug*       up    1:00:00      8   idle node[02-09]
```

State	Meaning
alloc	The node has been allocated and is in use.
idle	The node is idle and can be allocated.
mix	Part of the node is in use but still has allocatable resources.
down	The node is down.
drain	No further job will be scheduled on that node, but the currently running jobs will keep running.

Usages of salloc (see **salloc --help**)

```
[119010355@csc4005_slurm_master ~]$ salloc -help
Usage: salloc [OPTIONS(0)...] [ : [OPTIONS(N)]] [command(0) [args(0)...]]

Parallel run options:
  -A, --account=name          charge job to specified account
  -b, --begin=time            defer job until HH:MM MM/DD/YY
      --bell                  ring the terminal bell when the job is allocated
      --bb=<spec>             burst buffer specifications
      --bbf=<file_name>       burst buffer specification file
  -c, --cpus-per-task=ncpus   number of cpus required per task
      --comment=name          arbitrary comment
      --container             Path to OCI container bundle
      --cpu-freq=min[-max[:gov]] requested cpu frequency (and governor)
      --delay-boot=mins       delay boot for desired node features
  -d, --dependency=type:jobid[:time] defer job until condition on jobid is satisfied
      --deadline=time         remove the job if no ending possible before
                              this deadline (start > (deadline - time[-min]))
  -D, --chdir=path            change working directory
      --get-user-env          used by Moab.  See srun man page.
      --gid=group_id          group ID to run job as (user root only)
      --gres=list             required generic resources
      --gres-flags=opts       flags related to GRES management
  -H, --hold                  submit job in held state
  -I, --immediate[=secs]      exit if resources not available in "secs"
  -J, --job-name=jobname      name of job
  -k, --no-kill               do not kill job on node failure
  -K, --kill-command[=signal] signal to send terminating job
  -L, --licenses=names        required license, comma separated
  -M, --clusters=names        Comma separated list of clusters to issue
                              commands to.  Default is current cluster.
                              Name of 'all' will submit to run on all clusters.
                              NOTE: SlurmDBD must up.
  -m, --distribution=type     distribution method for processes to nodes
                              (type = block|cyclic|arbitrary)
      --mail-type=type        notify on state change: BEGIN, END, FAIL or ALL
      --mail-user=user        who to send email notification for job state
                              changes
      --mcs-label=mcs         mcs label if mcs plugin mcs/group is used
  -n, --ntasks=N              number of processors required
      --nice[=value]          decrease scheduling priority by value
      --no-bell               do NOT ring the terminal bell
      --ntasks-per-node=n     number of tasks to invoke on each node
  -N, --nodes=N               number of nodes on which to run (N = min[-max])
  -O, --overcommit            overcommit resources
      --power=flags           power management options
      --priority=value        set the priority of the job to value
      --profile=value         enable acct_gather_profile for detailed data
                              value is all or none or any combination of
                              energy, lustre, network or task
  -p, --partition=partition   partition requested
  -q, --qos=qos               quality of service
  -Q, --quiet                 quiet mode (suppress informational messages)
      --reboot                reboot compute nodes before starting job
  -s, --oversubscribe         oversubscribe resources with other jobs
      --signal=[R:]num[@time] send signal when time limit within time seconds
      --spread-job            spread job across as many nodes as possible
      --switches=max-switches{@max-time-to-wait}
                              Optimum switches and max time to wait for optimum
  -S, --core-spec=cores       count of reserved cores
      --thread-spec=threads   count of reserved threads
  -t, --time=minutes          time limit
      --time-min=minutes      minimum time limit (if distinct)
      --uid=user_id           user ID to run job as (user root only)
                              smaller count
  -v, --verbose               verbose mode (multiple -v's increase verbosity)
      --wckey=wckey           wckey to run job under

Constraint options:
      --cluster-constraint=list specify a list of cluster constraints
      --contiguous            demand a contiguous range of nodes
  -C, --constraint=list       specify a list of constraints
  -F, --nodefile=filename     request a specific list of hosts
      --mem=MB                minimum amount of real memory
      --mincpus=n             minimum number of logical processors (threads)
                              per node
      --reservation=name      allocate resources from named reservation
      --tmp=MB                minimum amount of temporary disk
  -w, --nodelist=hosts...     request a specific list of hosts
  -x, --exclude=hosts...      exclude a specific list of hosts

Consumable resources related options:
      --exclusive[=user]      allocate nodes in exclusive mode when
                              cpu consumable resource is enabled
      --exclusive[=mcs]       allocate nodes in exclusive mode when
                              cpu consumable resource is enabled
                              and mcs plugin is enabled
      --mem-per-cpu=MB        maximum amount of real memory per allocated
                              cpu required by the job.
                              --mem >= --mem-per-cpu if --mem is specified.

Affinity/Multi-core options: (when the task/affinity plugin is enabled)
                              For the following 4 options, you are
                              specifying the minimum resources available for
                              the node(s) allocated to the job.
      --sockets-per-node=S    number of sockets per node to allocate
      --cores-per-socket=C    number of cores per socket to allocate
      --threads-per-core=T    number of threads per core to allocate
  -B  --extra-node-info=S[:C[:T]]  combine request of sockets per node,
                              cores per socket and threads per core.
                              Specify an asterisk (*) as a placeholder,
                              a minimum value, or a min-max range.

      --ntasks-per-core=n     number of tasks to invoke on each core
      --ntasks-per-socket=n   number of tasks to invoke on each socket
      --hint=                 Bind tasks according to application hints
                              (see "--hint=help" for options)
      --mem-bind=             Bind memory to locality domains (ldom)
                              (see "--mem-bind=help" for options)

GPU scheduling options:
      --cpus-per-gpu=n        number of CPUs required per allocated GPU
  -G, --gpus=n                count of GPUs required for the job
      --gpu-bind=...          task to gpu binding options
      --gpu-freq=...          frequency and voltage of GPUs
      --gpus-per-node=n       number of GPUs required per allocated node
      --gpus-per-socket=n     number of GPUs required per allocated socket
      --gpus-per-task=n       number of GPUs required per spawned task
      --mem-per-gpu=n         real memory required per allocated GPU


Help options:
  -h, --help                  show this help message
      --usage                 display brief usage message

Other options:
  -V, --version               output version information and exit
```

Interactive Approach

After entering the **salloc** environment, you can use the following command to run your MPI program (suppose you have compiled it):

mpirun ./mpi_hello

Or to specify the number of processors used to run the program:

mpirun -np 4 ./mpi_hello

Batch Approach

Another method to use slurm is **sbatch** . This is especially useful when you want to submit multiple jobs to queue.

We have prepared **template.sh** for you to test.

The script looks like:

```
#!/bin/bash
#SBATCH --job-name=parallel_job_test # Job name
#SBATCH --nodes=1                   # Run all processes on a single node
#SBATCH --ntasks=4                  # number of processes = 4
#SBATCH --cpus-per-task=4           # Number of CPU cores per process
#SBATCH --mem=600mb                 # Total memory limit
#SBATCH --time=00:05:00             # Time limit hrs:min:sec
#SBATCH --partition=Debug           # Partition name: Project or Debug (Debug is default)

mpirun -np 4 ./xxxxx
```

You can adjust the path, nodes, processes, and other parameters in it. Try and play around with it.

Batch Approach

Then you can submit it with:

sbatch template.sh

After the job is finished, you get a **slurm-XXX.out** file under the submitting directory.

For more options, see **sbatch --help** and for instance, you can add

#SBATCH --output=XXXX

to change your output.

Cancelling Jobs

To cancel a specific job, simply type:

scancel <JOBID>

To cancel all jobs submitted by you, you can use:

scancel --user=$(whoami)

Useful Commands

- salloc - Enter an interactive shell
- sbatch - Submit a job
- scancel - Cancel a job
- squeue - View current queue
- sacct - View submission history
- sinfo -a - View node information
- scontrol show job [JOB_ID] - View information for the job

and so on...

Additional Commands' Usages

Get help on these commands from these pages:

- PKU HPC (Chinese)

https://hpc.pku.edu.cn/_book/guide/slurm/slurm.html

- Harvard FASRC

https://docs.rc.fas.harvard.edu/kb/convenient-slurm-commands/

- HPC2N

https://www.hpc2n.umu.se/batchsystem

MPI, OpenMP, pthread, and CUDA

For the demo code and commands for compilation and execution, please refer to:

https://github.com/bokesyo/CSC4005_2022Fall_Demo

The latest version of the code has also been uploaded to blackboard.

MPI Sample

```c
#include <mpi.h>
#include <stdio.h>
#include <math.h>

int main(int argc,char** argv)
{
    int myid,numproces;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Comm_size(MPI_COMM_WORLD,&numproces);
    MPI_Get_processor_name(processor_name,&namelen);
    fprintf(stdout,"hello world! Process %d of %d on %s\n",
            myid,numproces,processor_name);
    MPI_Finalize();

    return 0;
}
```

MPI Sample

Compile demo code:

mpic++ mpi_hello.cpp -o mpi_hello

Run demo code:

mpirun -np 4 ./mpi_hello

The output should be like:

```
hello world! Process 2 of 4 on csc4005_slurm_node_01
hello world! Process 3 of 4 on csc4005_slurm_node_01
hello world! Process 0 of 4 on csc4005_slurm_node_01
hello world! Process 1 of 4 on csc4005_slurm_node_01
```

pthread Sample

```
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <pthread.h>

using namespace std;

void *welcome(void *arg) {
    cout << "Id: " << pthread_self() << endl;
    cout << "Welcome to Pthreads Programming" << endl;
    return (void *)0;
}

int main() {
    int ret;
    int *stat;
    pthread_t tid;

    // Create a thread within the process to execute welcome

    if ((ret = pthread_create(&tid, NULL, welcome, NULL)) != 0) {
        cout << "Error creating thread: " << strerror(ret) << endl;
        exit(1);
    }

    cout << "Created Thread " << tid << endl;
    pthread_join(tid, (void **)&stat);
    cout << "Thread " << tid << " terminated, Status = " << stat << endl;
    exit(0);

}
```

pthread Sample

Compile demo code:

g++ -o pthread_hello pthread_hello.cpp -lpthread

Run demo code:

./pthread_hello

The output should be like:

```
Created Thread 140185512527616
Id: 140185512527616
Welcome to Pthreads Programming
Thread 140185512527616 terminated, Status = 0
```

OpenMP Sample

```
#include <omp.h>
#include <iostream>

using namespace std;

int main(){
    #pragma omp parallel num_threads(5)
        {
                cout << "Hello, world!" << endl;
        }
}
```

OpenMP Sample

Compile demo code:

g++ -o openmp_hello openmp_hello.cpp -fopenmp

Run demo code:

./openmp_hello

The output should be like:

```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
```

CUDA Sample

```cpp
// nvcc cuda-hello.cpp
#include <cstdio>
#include <cuda.h>
#include <cuda_runtime.h>

int main() {
  int deviceCount = 0;
  cudaError_t error_id = cudaGetDeviceCount(&deviceCount);
  printf("%d CUDA devices detected\n");
  return 0;
}
```

CUDA Sample

Compile demo code:

nvcc -o cuda_hello cuda_hello.cpp

Run demo code:

./cuda_hello

The output should be like:

4291104 CUDA devices detected

How to conduct experiments and write reports

- For conducting experiments, adjust data size, number of cores, processors, threads, blocks, etc.
- Compare performances among different methods.
- Visualize the results.
- Analyze the results retrieved from different experiments and derive the reasons behind.

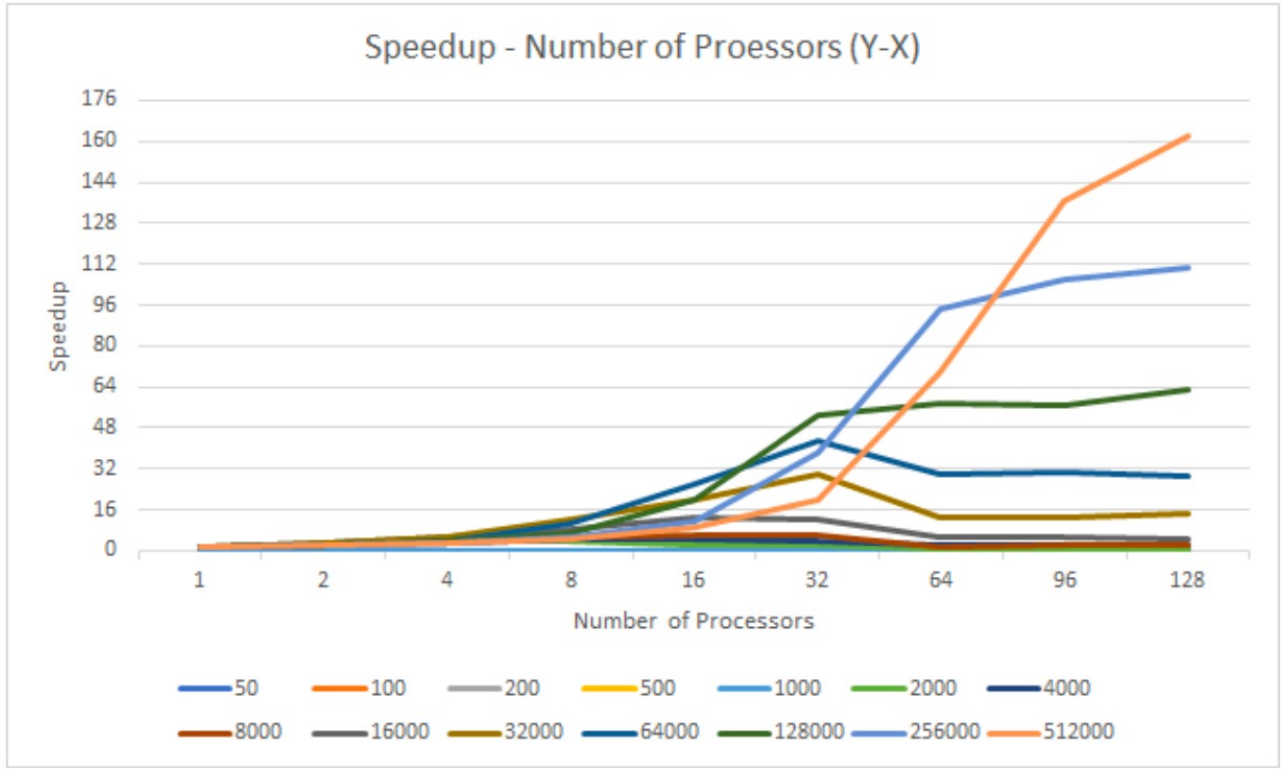
Sample Reports

Speedup of version 2 implementation (Super-linear Speedup is bolded)

Data Size\Cores	1	2	4	8	16	32	64	96	128
32000	1	2.619	5.549	**11.769**	**19.473**	29.750	13.222	12.904	14.092
64000	1	2.238	3.709	**10.455**	**26.081**	**42.929**	29.944	30.898	29.048
128000	1	2.103	3.314	6.420	**19.609**	**53.261**	57.847	57.055	62.538
256000	1	2.014	3.069	5.378	11.129	**38.216**	**94.751**	**106.240**	110.811
512000	1	1.941	2.538	4.790	9.054	19.807	**70.123**	**136.929**	**162.003**

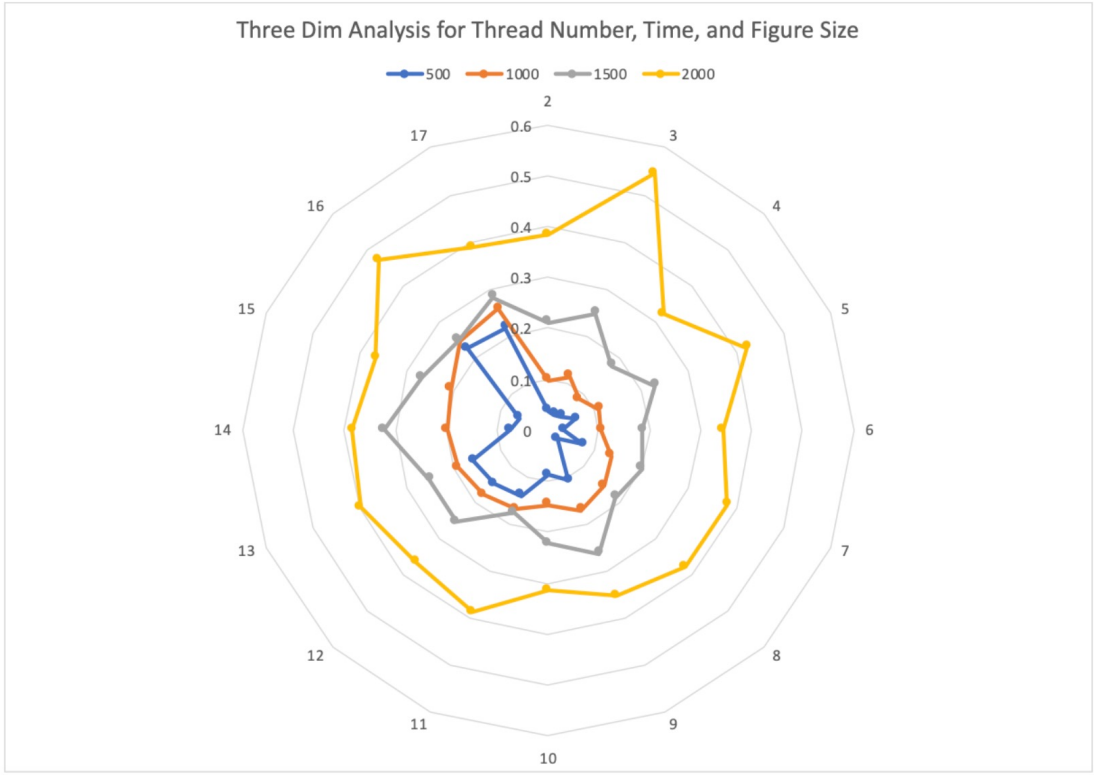
"More tests are added to get the super-linear speedup for $n < 128$"

Data Size\Cores	1	64	96	128
768000 (time / ms)	7.5×10^5	17032	7169	4196
768000 (speedup)	1	43.6	**103.571**	**176.954**
1024000 (time / ms)	1.4×10^6		18688	9664
1024000 (speedup)	1		70.6	**137**
2048000 (time / ms)	5.3×10^6			70884
2048000 (speedup)				75

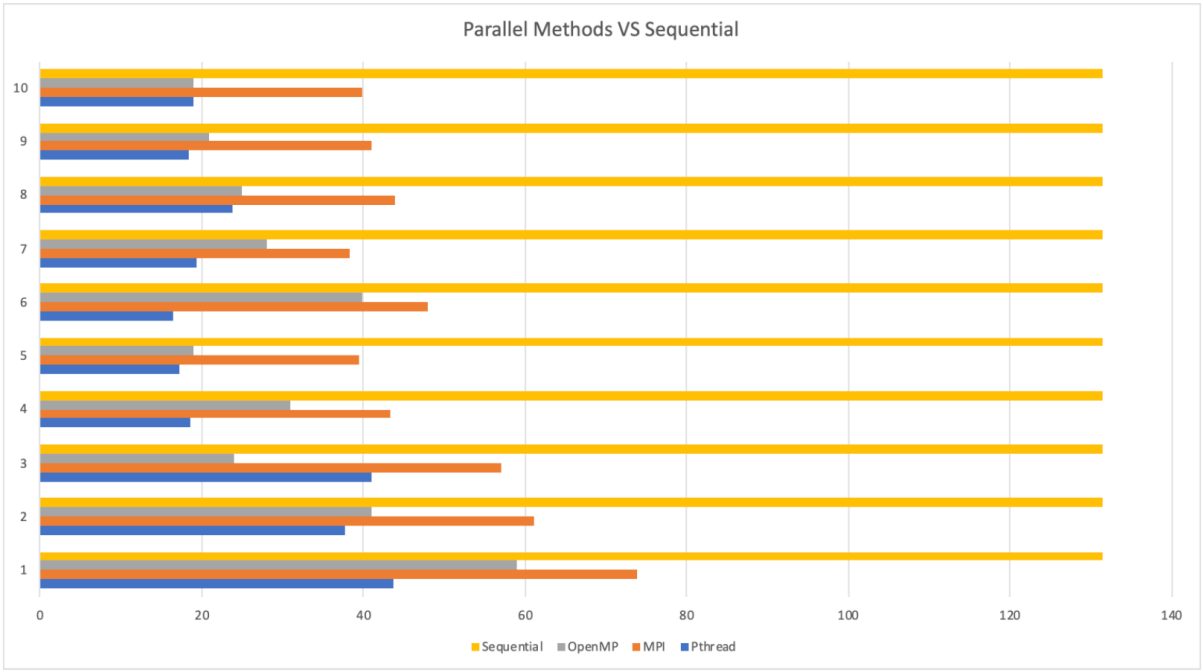


Sample Reports

We can see that as the number of threads used increases, the time used for simulation decreases. It decreases faster in the beginning, and coverges to a value as the thread number has reached some value. There is a diminishing effect in the speedup provided by OpenMP parallelization.

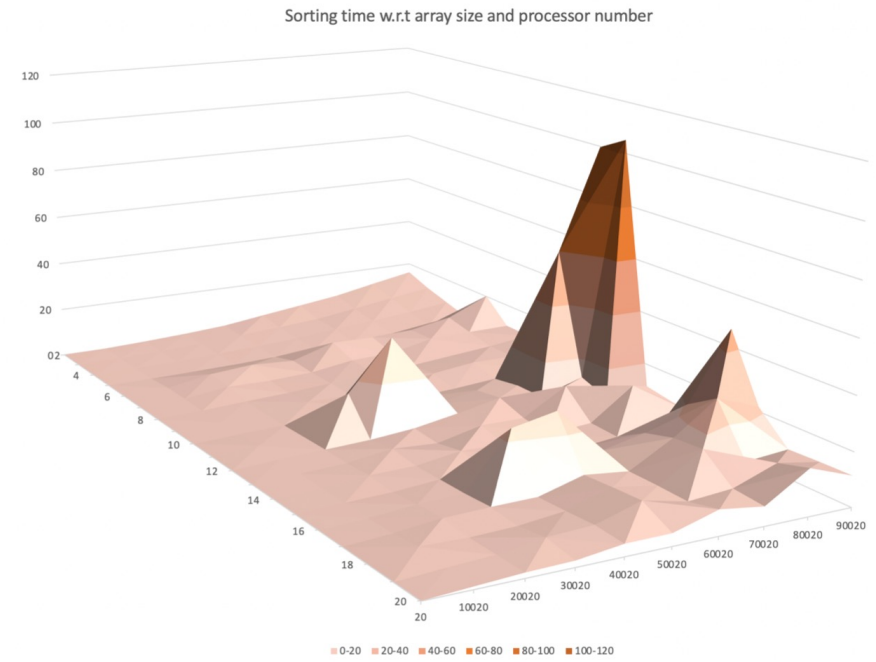


For a better comparison of the parallel speedup with the sequential simulation, here is a bar chart showing the different runtime records of 2000 iterations with different parallel methods and different level of parallelization:



Sample Reports

To have a complete overview of the experimental data, I have included the 3D graph, with x axis the process number, y axis the array size, and z axis the sorting time for seeing the optimal solution.



(Figure 12. Planar representation of sorting time with respect to sorting processors and sorting array size)