DISTRIBUTED SYSTEMS ASSIGNMENT REPORT

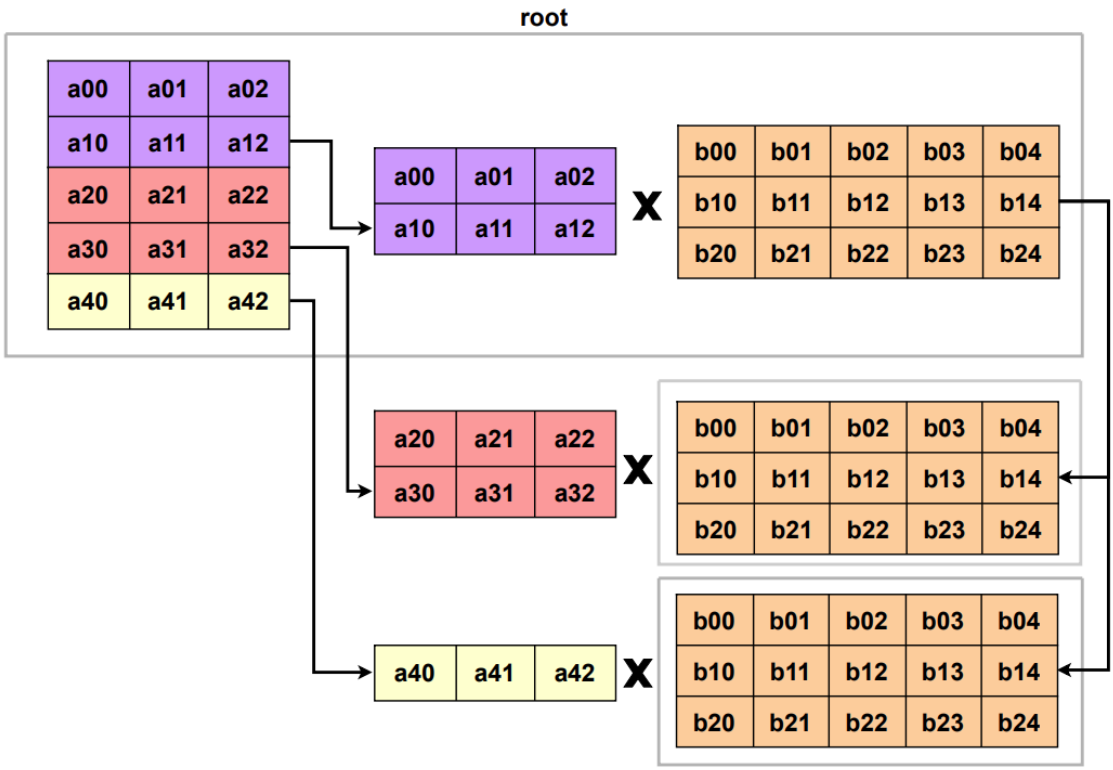
Assignment ID: 1

Student Name: 徐春晖 XU Chunhui

Student ID: 12110304

DESIGN

Distributed Idea



Just follow the instruction, I implemented my distributed idea in the following way: I divided the A matrix into rows and distributed some rows of A to different distributed computing units; and the entire content of the B matrix was distributed to the computing units.

For the computing node u, if it gets the rows in Matrix A from row u_s to u_e, its computing task is:

$$A[i][j] * B[j][k] = C[i][k](u_s \leq i \leq u_e, \forall j, k)$$

While Matrix C is the computation result.

Finally, combine the results from different computing units then get the whole result matrix C.

Because the focus of my exploration in this assignment is on distributed strategies and their performance comparison, I only ensured that each part used the same calculation strategy and **did not introduce various different matrix multiplication optimization methods**.

DESIGN IMPLEMENTATION

Initial Matrix Generation Strategy

I changed the strategy for generating the initial matrix elements so that each generated initial element has a float part e_i satisfies $e_i \in [0, 1]$, giving full play to the calculation characteristics of the C language `double` type.

```
// for (int i = 0; i < MAT_SIZE; ++i)
//     for (int j = 0; j < MAT_SIZE; ++j) {
//         a[i * MAT_SIZE + j] = i + j;
//         b[i * MAT_SIZE + j] = i * j;
//     }

void init_rand_mat_ptr(double *mat) {
    /* initialize a matrix with random values */
    for (int i = 0; i < MAT_SIZE; ++i) {
        for (int j = 0; j < MAT_SIZE; ++j) {
            mat[i * MAT_SIZE + j] = rand() + (double) rand() / RAND_MAX;
        }
    }
}
```

Check Strategy

Different calculation methods may lead to inconsistent results for double type operations. I changed the way to determine whether the values are equal: the equality comparison was changed to the absolute value of the error being less than $1e^{-6}$ (in fact, if matrix mutiplication optimization algorithms are not used, the values could be completely consistent, which is also the case in my assignment).

```c
#define EPS (1e-6)

#define abs(x) ((x) > 0 ? (x) : (-(x)))

int checkRes_ptr(const double *target, const double *res) {
    /*
      check whether the obtained result is the same as the intended target;
      if true return 1, else return 0
    */
    for (int i = 0; i < MAT_SIZE; ++i) {
        for (int j = 0; j < MAT_SIZE; ++j) {
            // if (res[i][j] != target[i][j]) {
            //     return 0;
            // }
            double diff = target[i * MAT_SIZE + j] - res[i * MAT_SIZE + j];
            if (abs(diff) > EPS) {
                return 0;
            }
        }
    }
    return 1;
}
```

Task Allocation Strategy

I use `s_g.c` , `sv_gv.c` . They have different task allocation strategies.

`s_g.c` Strategy

I used `MPI_Scatter()` and `MPI_Gather()` to distribute matrix calculation tasks. This broadcast method is convenient, but there is one thing that is not ideal: it can only assign fixed-length calculation content to each node. When the tasks are not easy to be evenly distributed, the root node is directly used to calculate the remaining content.

For example, 8 tasks are divided into 3 nodes, each node processes 2 tasks, and the root node processes the remaining 2 tasks. Root node will perform 4 tasks, which has more workloads and latency.

```c
int local_size = MAT_SIZE / mpiSize;

// ... Other preparation and initialization

MPI_Scatter(a, local_size * MAT_SIZE, MPI_DOUBLE, local_a, local_size * MAT_SIZE, MPI_DOUBLE, 0,
            MPI_COMM_WORLD);

// ... Perform distributed calculation

```

```
MPI_Gather(local_c, local_size * MAT_SIZE, MPI_DOUBLE, c, local_size *
MAT_SIZE, MPI_DOUBLE, 0, MPI_COMM_WORLD);

if (rank == root) {
    // ... Calculate rest part
}
```

`s_g.c` Strategy

In this file, I used `MPI_Scatterv()` and `MPI_Gatherv()` to distribute matrix calculation tasks .I allocate computing tasks of varying length to each node, so that the computing tasks are as similar as possible.

For example, 8 tasks are divided into 3 nodes, each node processes 2 tasks. There are remaining 2 tasks. Then we select 2 of the whole 3 nodes, let them have 1 more task. This ensures that the task volume between each node differs by at most 1 unit.

```
int quo_sz = MAT_SIZE / mpiSize;
int rem_cnt = MAT_SIZE % mpiSize;

for (int i = 0; i < rem_cnt; ++i) {
    local_szs[i] = quo_sz + 1;
}
for (int i = rem_cnt; i < mpiSize; ++i) {
    local_szs[i] = quo_sz;
}

int offset = 0;
for (int i = 0; i < mpiSize; ++i) {
    displs[i] = offset;
    offset += local_szs[i] * MAT_SIZE;
    scounts[i] = local_szs[i] * MAT_SIZE;
}

// ... Other preparation and initialization

MPI_Scatterv(a, scounts, displs, MPI_DOUBLE, local_a, scounts[rank],
MPI_DOUBLE, root, MPI_COMM_WORLD);

// ... Perform distributed calculation

MPI_Gatherv(local_c, scounts[rank], MPI_DOUBLE, c, scounts, displs,
MPI_DOUBLE, root, MPI_COMM_WORLD);
```

Data Collection Strategy

I use a bash script `test.sh` to run `mpirun` command 100 times and get the average time.

If there's something wrong in result, it will stop automatically and report. Otherwise, it write the final result to `test_record.csv`.

Finally I use a python script `plot.py` . to draw the chart.

RUNNING RESULT

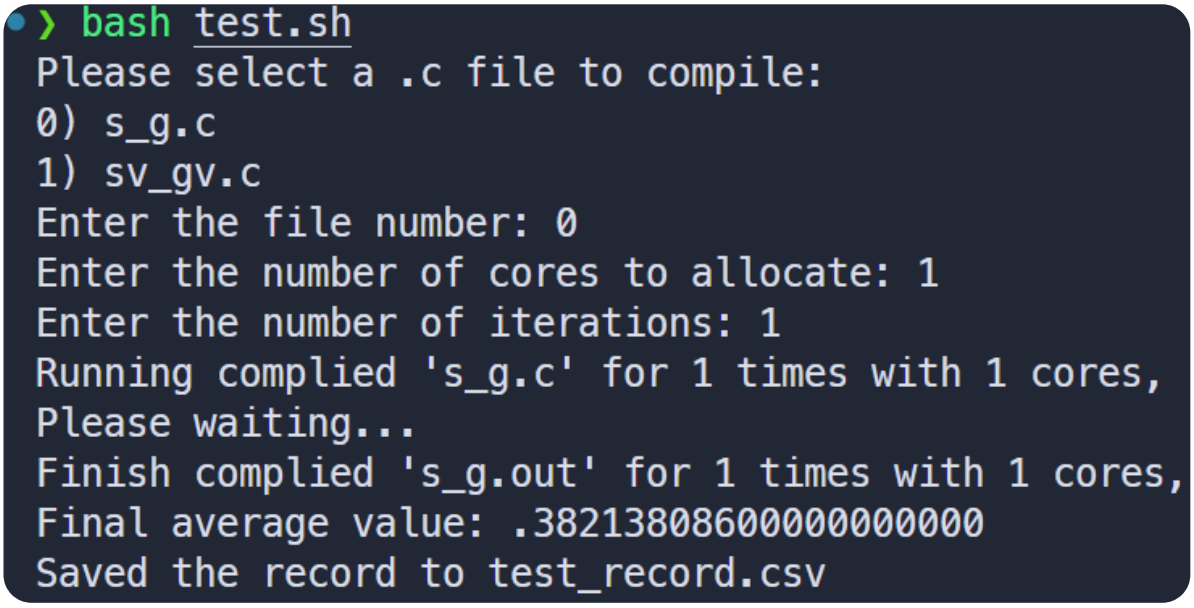
Environment

Components	Parameters
CPU	AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz (8 cores 16 threads)
RAM	Samsung DDR4 3200MHz 8G × 2
OS	Ubuntu 22.04 in WLS2 with Windows 10 22H2

Local Test

Screenshot

Bash Script Running Result:

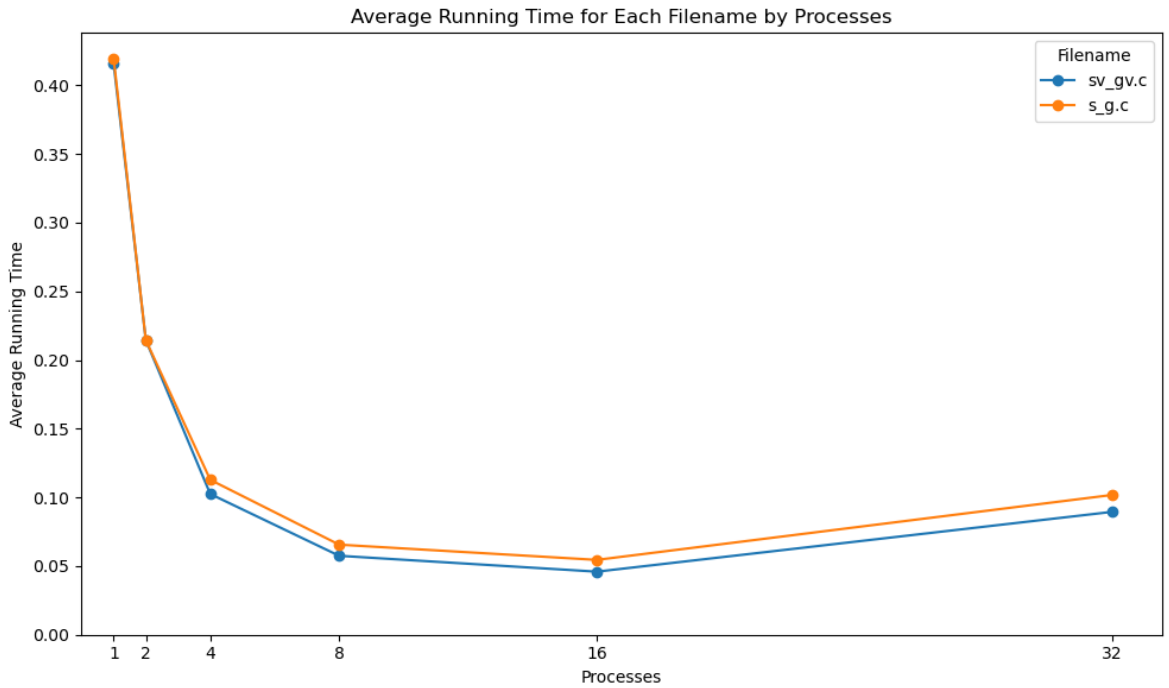


Test Record Table:

Assignment1 > src > test_record.csv > data

```
Filename,Proc,Running Time,Time,Iterations
sv_gv.c,1,0.419864952,2024/10/16 19:49,100
sv_gv.c,2,0.214915833,2024/10/16 19:51,100
sv_gv.c,4,0.102479292,2024/10/16 20:52,100
sv_gv.c,8,0.057490792,2024/10/16 20:57,100
sv_gv.c,16,0.045920487,2024/10/16 20:58,100
sv_gv.c,32,0.089484178,2024/10/16 21:01,100
sv_gv.c,1,0.412157277,2024/10/16 21:04,100
sv_gv.c,2,0.213920324,2024/10/16 21:07,100
s_g.c,1,0.419455944,2024/10/16 21:11,100
s_g.c,2,0.214426436,2024/10/16 21:13,100
s_g.c,4,0.112797895,2024/10/16 21:16,100
s_g.c,8,0.065604851,2024/10/16 21:19,100
s_g.c,16,0.054498454,2024/10/16 21:22,100
s_g.c,32,0.101804487,2024/10/16 21:24,100
s_g.c,1,0.381711411,2024/10/17 17:16,1
```

Chart:



Feature

As the number of processes increases, the parallel computation time becomes shorter. However, once a certain number of processes is reached, the execution time reaches a minimum. Beyond that point, as more processes are added, the execution time begins to increase again.

With the growth in the number of processes, the speedup factor also increases gradually until it reaches its peak. But as the number of processes continues to rise, the speedup begins to decline.

The efficiency of the program doesn't increase as the number of processes increases.

Use `MPI_Gatherv()` , `MPI_Gatherv()` is faster.

Analyze

We can see that when the number of MPI processes increases, the computing efficiency does not continue to increase, but instead decreases in the end. The reason is that in the local test environment, the CPU has 8 cores and 16 threads.

When the processes increases, we need to use the MPI `--oversubscribe` command to run multiple tasks on 1 core. At this time, the actual CPU multi-process utilization rate is not increase, so the acceleration effect is no longer obvious. (Computation gain < communication cost)

Instead, it will lead to repeated execution of more common code parts, introduce more steps to distribute tasks and collect results, and lead to a decrease in overall efficiency.

And when I use `MPI_Gatherv()` , `MPI_Gatherv()` , the tasks are distributed more equally and the computation is more efficient (especially when the number of nodes increases and the `MAT_SIZE` cannot be simply divided by MPI processes number).

Distributed Test

Screenshot

Docker Build:

```
[+] Building 178.9s (12/12) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 1.69kB
 => [internal] load metadata for docker.io/library/ubuntu:22.04
 => [internal] load .dockerignore
 => => transferring context: 2B
 => CACHED [1/8] FROM docker.io/library/ubuntu:22.04@sha256:58b87898e82351c6cf9cf5b
 => [2/8] RUN sed -i s:/archive.ubuntu.com:/mirrors.tuna.tsinghua.edu.cn/ubuntu:g /
 => [3/8] RUN apt-get install -y     build-essential
 => [4/8] RUN apt-get install -y     openssh-server     openmpi-bin     openmpi-com
 => [5/8] RUN echo 'root:root' | chpasswd
 => [6/8] RUN useradd -rm -d /home/mpi -s /bin/bash -g root -G sudo -u 1001 mpi &&
 => [7/8] RUN mkdir -p /home/mpi/.ssh &&     ssh-keygen -q -t rsa -N '' -f /home/mp
 => [8/8] WORKDIR /home/mpi
 => exporting to image
 => => exporting layers
 => => writing image sha256:9594a48511c35cbb3c9122a8865f17efa94e67f064df599c5a64bd4
 => => naming to docker.io/library/mpi1

ons → docker scout quickview
```

Run Docker Compose:

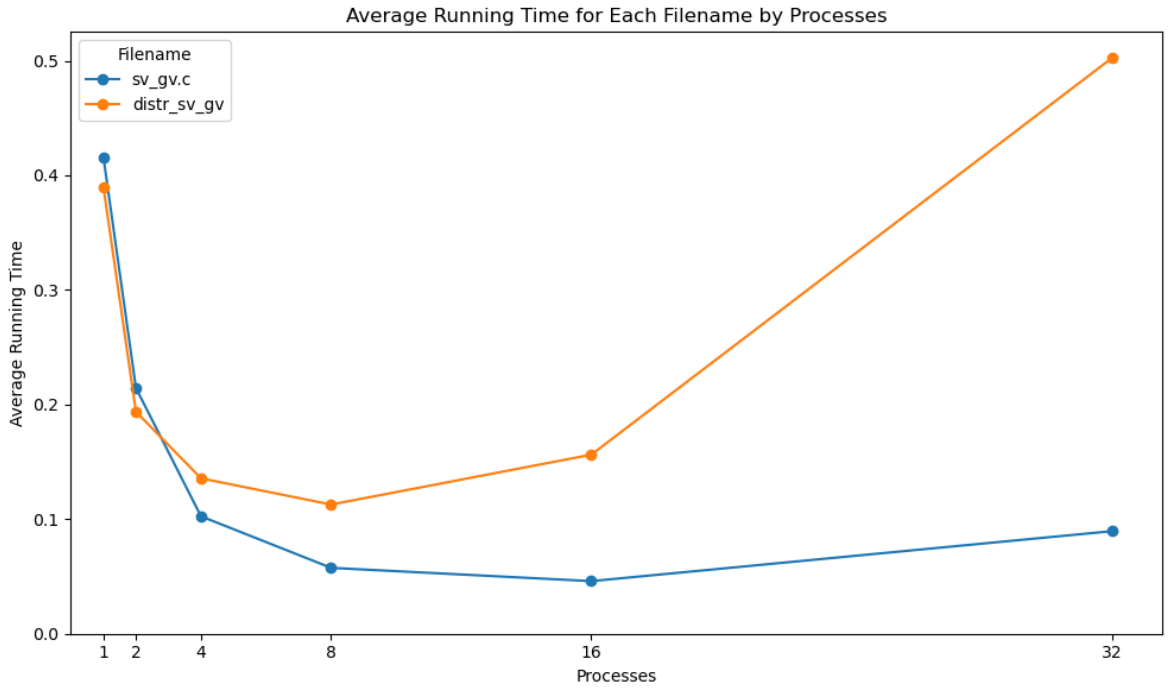
```
E:\SUSTech_UGCS\SUSTech_CS328-Distributed_2024f_Works\Assignment1\src  main
❯ docker compose up -d
[+] Running 4/4
 ✔ Network src_mpi-network  Created
 ✔ Container node1          Started
 ✔ Container node2          Started
 ✔ Container node3          Started
E:\SUSTech_UGCS\SUSTech_CS328-Distributed_2024f_Works\Assignment1\src  main
❯ docker exec -it node1 /bin/bash
root@node1:/home/mpi# su mpi
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

mpi@node1:~$ sudo echo node1 slots=2 > host
[sudo] password for mpi:
mpi@node1:~$ sudo echo node2 slots=2 >> host
mpi@node1:~$ sudo echo node3 slots=2 >> host
mpi@node1:~$ cat host
node1 slots=2
node2 slots=2
node3 slots=2
mpi@node1:~$ sudo chmod 777 ./mpi_files/sv_gv.out
mpi@node1:~$ mpirun -n 4 --hostfile host ./mpi_files/sv_gv.out
Warning: Permanently added 'node3' (ED25519) to the list of known hosts.
Warning: Permanently added 'node2' (ED25519) to the list of known hosts.
0.1331753870
```

Distributed Running Result (only once):

```
mpi@node1:~$ mpirun -n 1 --hostfile host ./mpi_files/sv_gv.out
0.3808874470
mpi@node1:~$ mpirun -n 2 --hostfile host ./mpi_files/sv_gv.out
0.1937590950
mpi@node1:~$ mpirun -n 4 --hostfile host ./mpi_files/sv_gv.out
0.1356105160
mpi@node1:~$ mpirun -n 8 --oversubscribe --hostfile host ./mpi_files/sv_gv.out
0.1127286420
mpi@node1:~$ mpirun -n 16 --oversubscribe --hostfile host ./mpi_files/sv_gv.out
0.1562446530
mpi@node1:~$ mpirun -n 32 --oversubscribe --hostfile host ./mpi_files/sv_gv.out
0.5022422540
```

Compare Result Chart:



Images Description

I used the built `mpi1` docker image, using the docker compose configuration to created `node1`, `node2`, and `node3` 3 docker containers, and placed them in the same virtual subnet. I allocated 2 MPI slots to each node, with node1 as the host.

Feature & Analyze

We can see that in the distributed environment, the efficiency curves in the early stage and the local environment are similar, but as the number of MPI processes increases, the decrease in efficiency becomes very significant.

I only allocated 6 MPI slots in total. As the number of processes increases, more processes need to run on each slot. And its performance is slightly affected by docker virtualization, and its performance optimization is not as good as WSL2.

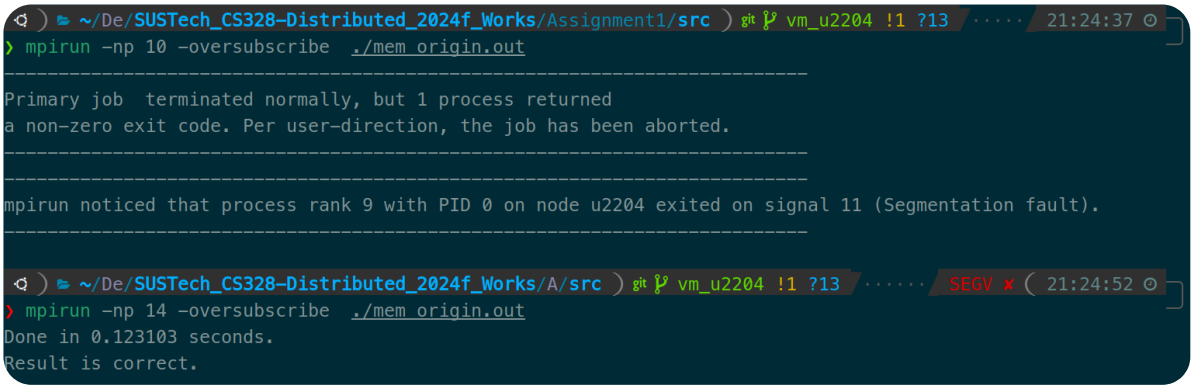
The most important thing is that in a distributed environment, the cost of communication will become relatively large. This will make the communication loss caused by the increase in processes more obvious, resulting in a significant decline in computing efficiency.

PROBLEMS

Memory Allocation

I first try my assignment in VMware Ubuntu 22.04 virtual machine. I allocated 2 CPUs × 4 cores and 4GB RAM for it.

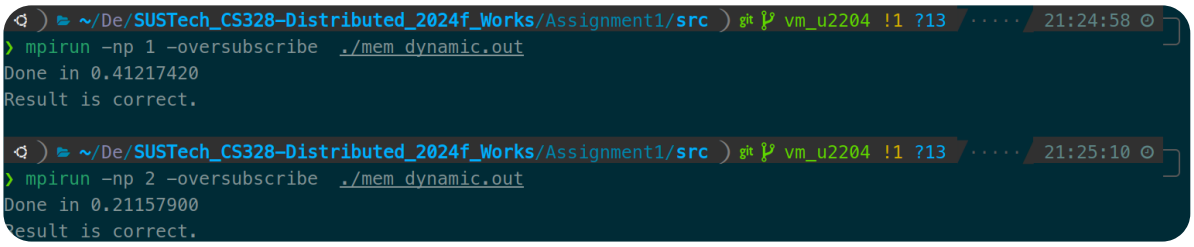
Because the stack memory of `OpenMPI` slot is limited, when the number of computing nodes decreases, each node will need more stack memory to allocate its own matrix for the computing process. When the memory is insufficient, it is very easy to cause `Segmentation Fault` (my test test shows that it will happen when there are less than 12 processes).



Therefore, for the matrix part, I adopted the method of dynamic memory allocation, allocating the matrix memory on the heap, and assigning the first address to a one-dimensional pointer.

```c
// double a[MAT_SIZE][MAT_SIZE],    /* matrix A to be multiplied */
// b[MAT_SIZE][MAT_SIZE],       /* matrix B to be multiplied */
// c[MAT_SIZE][MAT_SIZE],       /* result matrix C */
// bfRes[MAT_SIZE][MAT_SIZE];   /* brute force result bfRes */

double *a = (double *) malloc(MAT_SIZE * MAT_SIZE * sizeof(double));
double *b = (double *) malloc(MAT_SIZE * MAT_SIZE * sizeof(double));
double *c = (double *) malloc(MAT_SIZE * MAT_SIZE * sizeof(double));
double *bfRes = (double *) malloc(MAT_SIZE * MAT_SIZE * sizeof(double));
```



And use WSL2 will not cause this problem.

Docker Build without specified Ubuntu Version

Following the steps in the lab tutorial, I wrote the `Dockerfile` to built a Docker image. When I used apt to install related file packages, network failures occurred in some file packages and the build was terminated. I simply guessed that the domestic network connection was not smooth.

```
11  RUN apt-get update
12
13  RUN apt-get install -y \
14      gcc \
15      openssh-server \
16      openmpi-bin \
17      openmpi-common \
18      libopenmpi-dev
    Connection failed [IP: 185.125.190.83 80]
    Get:118 http://archive.ubuntu.com/ubuntu noble/main amd64 cpp-13-x86-64-linux-gnu amd64 13.2.0-23ubuntu4 [11.2 MB]
    Err:152 http://archive.ubuntu.com/ubuntu noble/main amd64 libllvm17t64 amd64 1:17.0.6-9ubuntu1
    Connection failed [IP: 91.189.91.83 80]
    E:Fetched 154 MB in 20min 11s (127 kB/s)
    Failed to fetch http://archive.ubuntu.com/ubuntu/pool/main/i/icu/libicu74_74.2-1ubuntu3.1_amd64.deb  Connection
    failed [IP: 185.125.190.83 80]
    E: Failed to fetch http://archive.ubuntu.com/ubuntu/pool/main/l/llvm-toolchain-17/libllvm17t64_17.0.6-
    9ubuntu1_amd64.deb  Connection failed [IP: 91.189.91.83 80]
    E: Unable to fetch some archives, maybe run apt-get update or try with --fix-missing?
```

So I added a command, trying to change the source to Tsinghua University version:

```
1  RUN sed -i s:/archive.ubuntu.com:/mirrors.tuna.tsinghua.edu.cn/ubuntu:g
   /etc/apt/sources.list && \
2      sed -i s:/security.ubuntu.com:/mirrors.tuna.tsinghua.edu.cn/ubuntu:g
   /etc/apt/sources.list
```

But after that, the network failure still existed.

Then I check the log, and found that the The download source was not switched correctly.

```
✓ 2/9 RUN sed -i s:/archive.ubuntu.com:/mirrors.tuna.tsinghua.edu.cn/ubuntu:g /etc/apt/sources.list && sed -i s:/secu··· 0.0s
1  # Ubuntu sources have moved to the /etc/apt/sources.list.d/ubuntu.sources
2  # file, which uses the deb822 format. Use deb822-formatted .sources files
3  # to manage package sources in the /etc/apt/sources.list.d/ directory.
4  # See the sources.list(5) manual page for details.
```

Search on the Internet: the Ubuntu has change to use more modern ways to manage the source list in 2024 fall. It use `/etc/apt/sources.list.d/` folder instead of use single `/etc/apt/sources.list` file.

So I use a new command to specify the Ubuntu version to 22.04 instead of default latest 24.04.

```
1  # FROM ubuntu
2  FROM ubuntu:22.04
```

Then the problem has been solved.

Nested Virtualization

I try to enable CPU in VMware to use docker in virtual machine. But there's somthing wrong.

Virtualization Engine
Virtualization Intel VT-x/EPT or AMD-V/RVI(V)
Virtualization CPU performance counters(U)

I need to enable the two options to use virtualization in virtual machine, but there's another problem:

Ubuntu22.04-VMware Workstation
This platform does not support virtualized AMD-V/RVI.
Do you want to continue without using virtualized AMD-V/RVI?
Yes(Y) No(N)

It seems that I need to configure the Windows Hyper-V service correctly and have it work with the VMware settings. But the performance loss may be much more serious. And I have already completed the experiment with WSL and Windows Docker, so I gave up trying nested virtualization.