Università
della
Svizzera
italiana

**Institute of
Computing
CI**

High-Performance Computing Lab **Institute of Computing**

Student: ZITIAN WANG Discussed with: None

Solution for Project 1

HPC Lab — Submission Instructions

**(Please, notice that following instructions are mandatory:
submissions that don't comply with, won't be considered)**

- Assignments must be submitted to iCorsi (i.e. in electronic format).
- Provide both executable package and sources (e.g. C/C++ files, Matlab). If you are using libraries, please add them in the file. Sources must be organized in directories called:
  Project_number_lastname_firstname
  and the file must be called:
  project_number_lastname_firstname.zip
  project_number_lastname_firstname.pdf
- The TAs will grade your project by reviewing your project write-up, and looking at the implementation you attempted, and benchmarking your code's performance.
- You are allowed to discuss all questions with anyone you like; however: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently.

In this project you will practice memory access optimization, performance-oriented programming, and OpenMP parallelizaton on the Rosa Cluster .

1. Rosa Warm-Up *(5 Points)*

1.1. Introduction of module system and Approaches to utilizing

The module system is a concept available on most supercomputers, simplifying the use of different software (versions) in a precise and controlled manner[1].In Rosa, it named as module concept, which allow user to use command to check or load all applications, tools, libraries, etc. For instance, user can use following commands[2].

- `module avail` - Lists all available modules on the current system.
- `module list` - Displays all currently loaded modules.
- `module load` - Loads specified modules from the existing list.

In contrast to Docker and Environment Modules, which I am more experienced with, both focus on streamlining environment configuration, controlling various program versions and dependencies, and enhancing user productivity in intricate settings. The distinction between distributed systems and HPC systems is also reflected in the differences between the two. While Environment Modules give customers flexible configuration in a shared operating system, Docker offers a more autonomous environment.

1.2. Introduction Slurm and its intended function

Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. Its intended functions.First, it allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work. Second, it provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes. Finally, it arbitrates contention for resources by managing a queue of pending work[3].Drawing an analogy with the YARN system that I am familiar with, both systems excel in managing and scheduling computing resources within a cluster. Their primary function is to dynamically allocate resources for computation. They both employ a task scheduler and facilitate scheduling based on job priority.

1.3. The programme prints words and host name

From the previous course, it is known that can use the getenv() method to read the node name of the currently running program. The specific implementation code is in the first part of the appendix, following is the result of the code.

```
[wangzi@icsnode21 1-Rosa-warm-up]$ ./hello_worldc
Hello World
host (node) name: icsnode21
```

Listing 1: Result of Hello World Program

1.4. The batch script shows nodes information

Use command `sinfo` check basic information of nodes. As demonstrated in Listing 3 in appendix, the status of certain nodes is identified as idle. Therefore, node 25 has been selected to execute the batch script. Add part of the script as follows:

```
#SBATCH --partition=slim
#SBATCH --nodelist=icsnode25
```

Listing 2: Specify the node to run

Based on the code presented in the course materials, the directive `#SBATCH --cpus-per-task=1` and `# SBATCH -- ntasks =1` indicates that the task is executed using a single CPU. I aim to enhance the clarity of this process by explicitly designating a specific CPU for task execution. Following the modifications in Listing 4, resulting in the output file reflecting the host name as icsnode25. The Result of this part can be checked in appendix section B.

1.5. The batch script allows run job in two nodes

Similar to the previous question, the existing script has declared that two nodes should perform this task. Set node 26 and 27 to run the batch script. The expected behavior is that the first function will be run twice, and run on the specified nodes. The Result of this part can be checked in appendix section B.

```
#SBATCH --partition=slim
#SBATCH --nodelist=icsnode26,icsnode27
```

Listing 3: Specify two nodes to run

2. Performance Characteristics *(30 Points)*

To construct a model for a single core of the Rosa nodes, we need to evaluate Peak FLOPS, Memory Bandwidth, Operational Intensity, and Measured Performance.

2.1. Peak FLOPS

This section focuses on identifying the parameters within the formula and calculating the Peak FLOPS value. Using the given formula, Peak FLOPS can be determined.

$$P_{\text{core}} = n_{\text{super}} \times n_{\text{FMA}} \times n_{\text{SIMD}} \times f \tag{1}$$

With command `lscpu`, it shows some basic information of the CPU as following.

Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
Byte Order	Little Endian
CPU(s)	20
On-line CPU(s) list	0-19
Thread(s) per core	1
Core(s) per socket	10
Socket(s)	2
NUMA node(s)	2
Vendor ID	GenuineIntel
CPU family	6
Model	63
Model name	Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz
Stepping	2
CPU MHz	3000.000
CPU max MHz	3000.000
CPU min MHz	1200.000
BogoMIPS	4599.85
L1d cache	32K
L1i cache	32K
L2 cache	256K
L3 cache	25600K
NUMA node0 CPU(s)	0-9
NUMA node1 CPU(s)	10-19
Flags	avx, avx2, fma ...

The CPU max MHz equals 3000.0000, then $f = 3.0$ GHz. According to the CPU Flags information, the CPU supports FMA, AVX, and AVX2 instructions. So n_{FMA} equals 2 and n_{SIMD} equals 4.

To get the value of SIMD factor (n_{FMA}), we need to check on the cpu-world.com website. The CPU of Rosa cluster is an Intel Xeon E5-2650, which has a core microarchitecture named Haswell.

Instruction	**Lat**	**TP**	**Uops**	**Ports**
VFMADD132PD (XMM, XMM, M128)	[5;≤11]	0.50 / 0.50	1 / 2	1*p01+1*p23
VFMADD132PD (XMM, XMM, XMM)	5	0.50 / 0.50	1 / 1	1*p01
VFMADD132PD (YMM, YMM, M256)	[5;≤12]	0.50 / 0.50	1 / 2	1*p01+1*p23
...				

The necessary data is available on the website, but it is too extensive to present in its entirety. The key point is that the TP values associated with the FMA operations remain consistent. As the Throughput (TP) is 0.50/0.50, then $n_{\text{FMA}} = \frac{1}{\text{TP}} = 2$.

Based on above information, the value of Peak FLOPS as following:

$$P_{\text{core}} = 2 \times 2 \times 4 \times 3.0\,\text{GHz} = 48\,\text{GFLOPS} \tag{2}$$

2.2. Cache and main memory size

With batch `hwloc-ls`, we can see the structure of memory. The conclusions drawn are consistent with the teaching materials provided. In the structure diagram, L1 is divided into two parts, L1d (for data) and L1i (for instructions)[4]. In this work, only L1d needs to be considered.

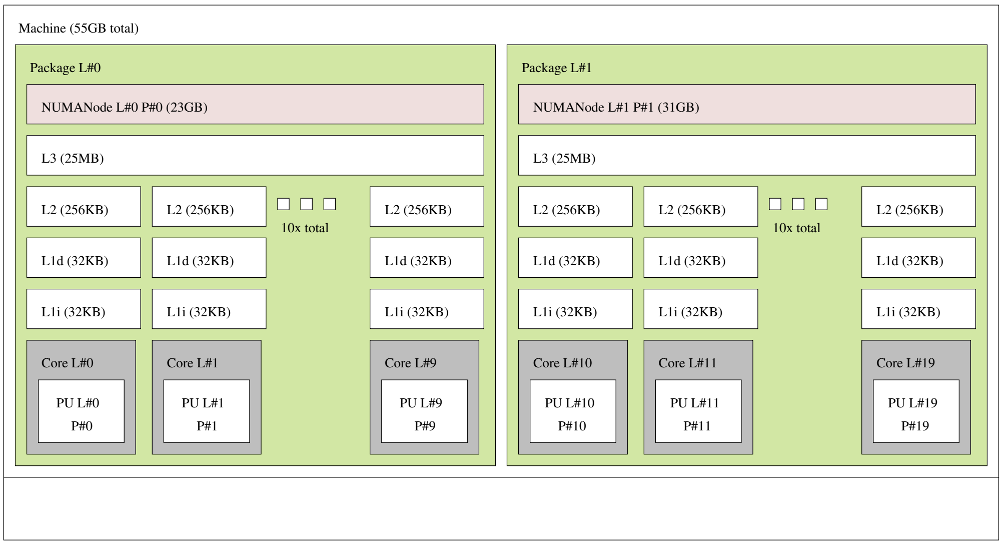


Figure 1: Memory structure

Component	Size
Main memory	23 GB
L3 cache	25 MB
L2 cache	256 KB
L1d cache	32 KB

Table 3: Memory and Cache Sizes

2.3. Bandwidth

First, we need confirm the value of $DSTREAM\_ARRAY\_SIZE$ is correct. Each array must be at least four times the size of the last cache level.

$$DSTREAM\_ARRAY\_SIZE \geq 4 \times \text{L3 Cache} = 4 \times 25\,\text{MB} = 100\,\text{MB} \tag{3}$$

So $DSTREAM\_ARRAY\_SIZE$ needs to exceed 104,857,600 B. It is reasonable to set `-DSTREAM_ARRAY_SIZE = 128000000` in the makefile.

```
CC = gcc
CFLAGS = -O3 -march=native -DSTREAM_TYPE=double -DNTIMES=20

all: stream_c_1

stream_c_1: stream.c
    ${CC} ${CFLAGS} -DSTREAM_ARRAY_SIZE=128000000 stream.c -o stream_c_1

.PHONY: clean
clean:
    rm -f stream_c_1
# code from course material
```

Listing 4: The makefile

The result after running is shown in the following table. As expected from the textbook, only the bandwidth of the copy operation shows a significant difference. It might because Copy kernel operations generally benefit from better cache utilization because of their simpler data access patterns. The data being copied may fit into the cache more effectively, minimizing cache misses and allowing faster memory access[5].

Function	Best Rate MB/s	Avg time (s)	Min time (s)	Max time (s)
Copy	19135.9	0.107083	0.107024	0.107272
Scale	11244.1	0.182227	0.182140	0.182403
Add	12286.6	0.250114	0.250029	0.250248
Triad	12287.9	0.250091	0.250003	0.250243

As shown in the above table, the single-core bandwidth is approximately 12GB/s.

2.4. Roofline model

The important parameters required to build the model are as table 4. Replace the parameters in the example python file with it to get the desired Roofline model diagram. Two of the parameters, $I_{\max}$ and $I_{\min}$, are obtained from experience[5].

Parameter	Value
$\mathrm{P}_{\mathrm{max}}$	48 GFLOPS
$\mathrm{b}_{\mathrm{max}}$	12 GB/s
$\mathrm{I}_{\mathrm{min}}$	1.e-2 FLOPS/B
$\mathrm{I}_{\mathrm{max}}$	1.e+3 FLOPS/B

Table 4: Model parameters

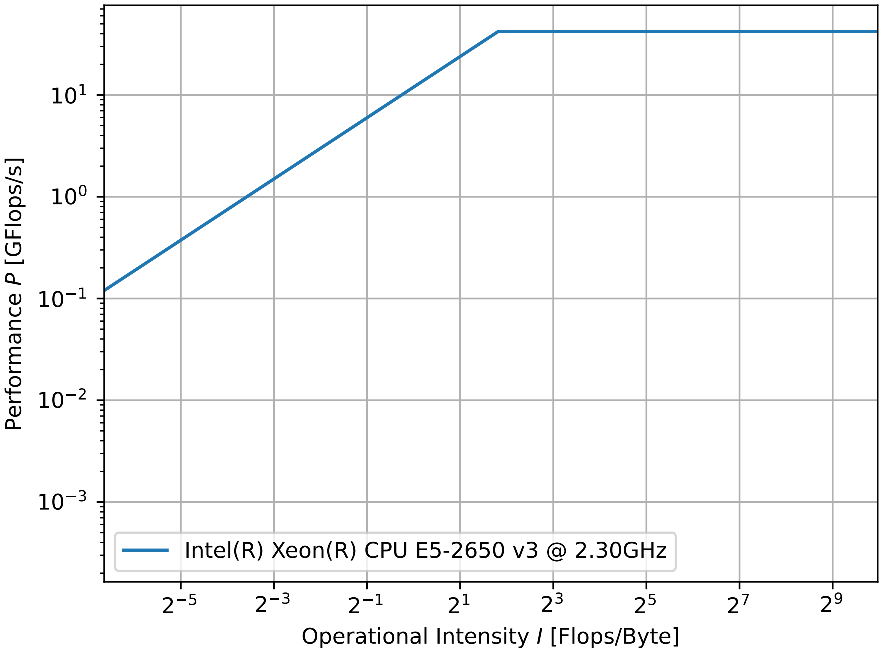


Figure 2: Picture roofline model

In the picture 2, there is an obvious inflection point in the function. Using Formula 4, we can get the operating intensity inflection point value based on system performance.

$$I_{\max} = b_{\max} \cdot P_{\max} = \frac{42\,\mathrm{GFlops/s}}{12\,\mathrm{GB/s}} = 3.5\,\mathrm{FLOPS/Byte} \tag{4}$$

- **Operation Intensity between 0.01 and 3.5 Flops/Byte**: As operation intensity increases, performance improves but is still limited by memory bandwidth.
- **Operation Intensity greater than 3.5 Flops/Byte**: When operation intensity approaches 12 Flops/Byte, the system may become compute-bound.

3. Optimize Square Matrix-Matrix Multiplication *(50 Points)*

3.1. Compiler options

In this section, I will compare the running time of the program compiled with `-O0`, `-O2`, and `-O3` to evaluate the effect of compiler optimization. Initially, revising the makefile to compile the updated executable file.

```
OPT_O0 = -O0 -march=native        # No optimization, useful as a baseline
OPT_O1 = -O1 -march=native        # Moderate optimization level 1
OPT_O2 = -O2 -march=native        # Higher optimization level 2
CFLAGS_O0 = -Wall -std=gnu99 $(OPT_O0) # standard compiler flags with no optimization
CFLAGS_O1 = -Wall -std=gnu99 $(OPT_O1) # standard compiler flags with level 1 optimization
CFLAGS_O2 = -Wall -std=gnu99 $(OPT_O2) # standard compiler flags with level 2 optimization
```

Listing 5: Makefile of Compiler options

Next step is to record new data time.

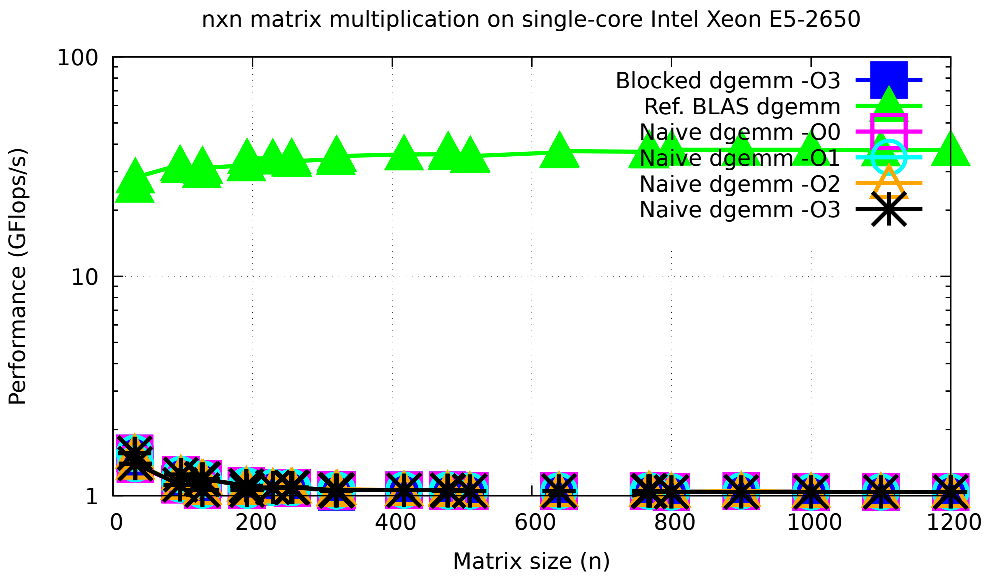
```
echo "====␣benchmark-naive␣-O0␣===================="
srun ./benchmark-naive-O0 | tee timing_O0.data
echo

echo "====␣benchmark-naive␣-O1␣===================="
srun ./benchmark-naive-O1 | tee timing_O1.data
echo

echo "====␣benchmark-naive␣-O2␣===================="
srun ./benchmark-naive-O2 | tee timing_O2.data
echo

echo "====␣benchmark-naive␣-O3␣===================="
srun ./benchmark-naive-O3 | tee timing_O3.data
echo
```

Listing 6: Script of Compiler options

Finally, update recorded data to draw the picture.

```
"timing_O0.data" using 2:4 title "Naive␣dgemm␣-O0" with linespoints pointtype 4 pointsize 2 linewidth 3 linetype 4 linecolor rgb "magenta", \
"timing_O1.data" using 2:4 title "Naive␣dgemm␣-O1" with linespoints pointtype 6 pointsize 2 linewidth 3 linetype 5 linecolor rgb "cyan", \
"timing_O2.data" using 2:4 title "Naive␣dgemm␣-O2" with linespoints pointtype 8 pointsize 2 linewidth 3 linetype 6 linecolor rgb "orange", \
"timing_O3.data" using 2:4 title "Naive␣dgemm␣-O3" with linespoints pointtype 3 pointsize 2 linewidth 3 linetype 7 linecolor rgb "black"
```

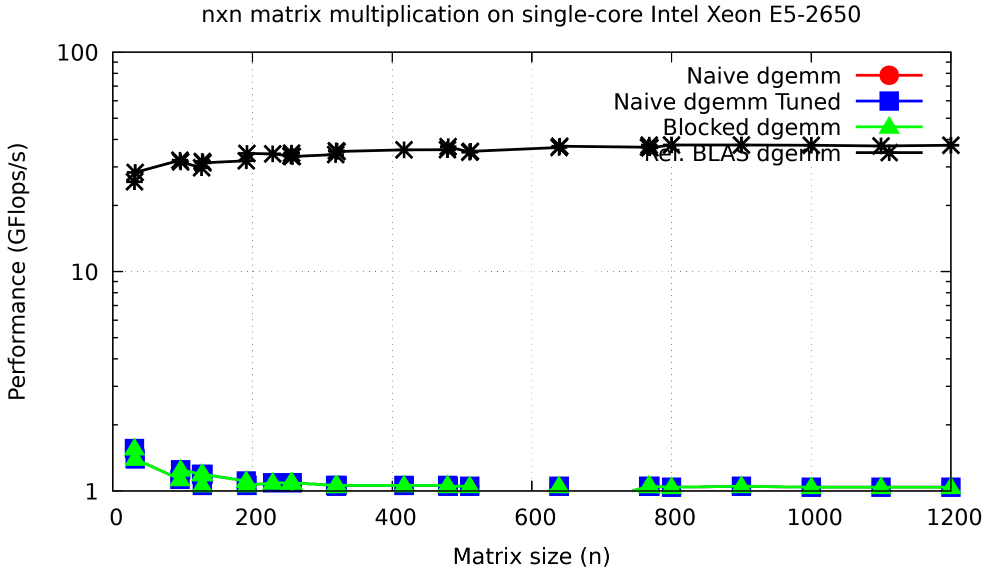
Listing 7: Plot of Compiler options



Due to the minimal impact of the compilation options on the final results, numerous data points overlap in the generated figure. Therefore, it is necessary to adjust the scale of the vertical axis. Then get the final result.

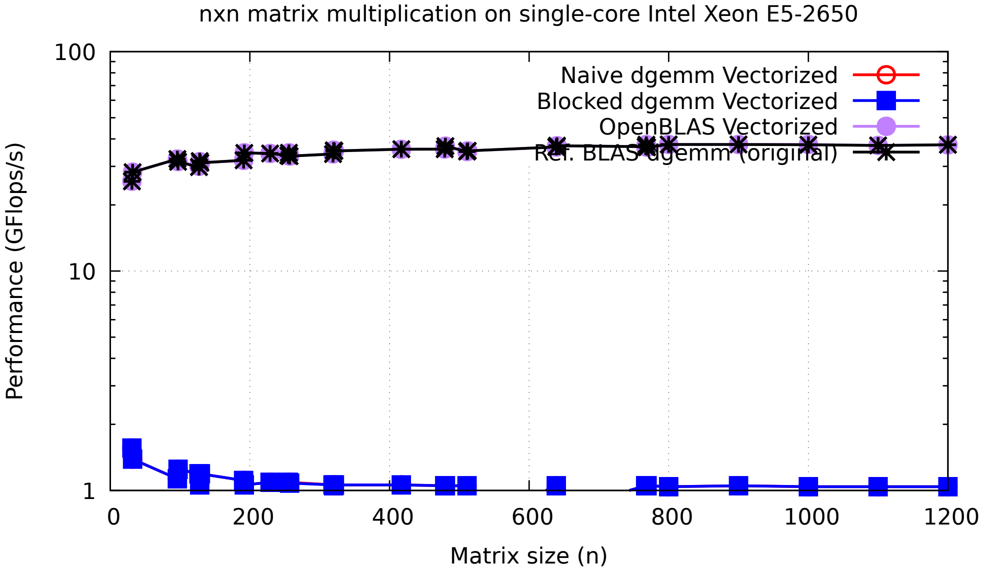
3.2. Tuning data access patterns

In this section, I compare four approaches: the original OpenBLAS implementation (benchmark-blas) as a control group, an unoptimized naive implementation (benchmark-naive) for baseline comparison, an optimized naive implementation (benchmark-naive-tune) with data access pattern optimizations, and a blocked matrix multiplication implementation (benchmark-blocked) using block-level optimization techniques. As the procedure is similar to the previous case, detailed explanations will not be provided here.



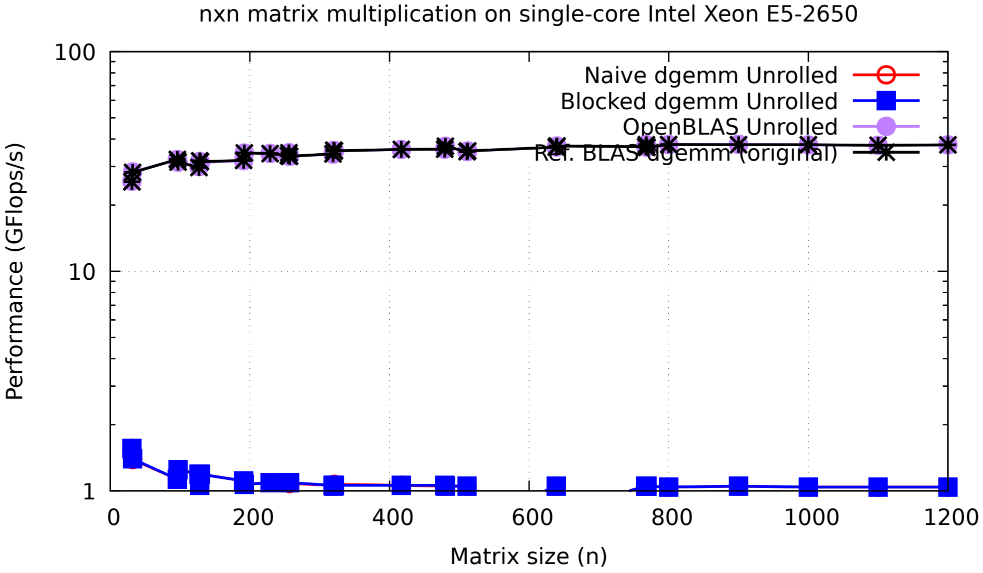
3.3. Vectorization

In this section, I compare four methods: Naive dgemm Vectorized, a vectorized version of the naive implementation; Blocked dgemm Vectorized, a vectorized version of the blocked matrix multiplication; OpenBLAS Vectorized, a vectorized implementation of OpenBLAS; and Ref. BLAS dgemm (original), the unmodified OpenBLAS implementation used as a control group. As the procedure is similar to the previous case, detailed explanations will not be provided here.



3.4. Loop unrolling

This section compares four methods: benchmark-naive-unrolled, a naive implementation with loop unrolling; benchmark-blocked-unrolled, a blocked matrix multiplication with loop unrolling; benchmark-blas-unrolled, an OpenBLAS implementation with loop unrolling; and benchmark-blas, the original OpenBLAS implementation used as a control group. As the procedure is similar to the previous case, detailed explanations will not be provided here.



References

[1] *Modules.* URL: `https://hpc-wiki.info/hpc/Modules`.

[2] *Environment Modules.* URL: `https://www.ci.inf.usi.ch/research/resources/`.

[3] *overview.* URL: `https://slurm.schedmd.com/quickstart.html`.

[4] *History*. URL: https://en.wikipedia.org/wiki/CPU_cache.

[5] Georg Hager and Gerhard Wellein. *Introduction to High Performance Computing for Scientists and Engineers*. CRC Press, 2010.

A. Appendix: Code

```cpp
#include <cstdlib>
#include <iostream>

using namespace std;

int main() {
    const char *nodes = std::getenv("SLURM_JOB_NODELIST");
    cout << "Hello World" << '\n';
    cout << "host(node) name:" << nodes << '\n';
    return 0;
}
//for each print, read the host name
```

Listing 8: Hello World Program

```bash
#!/bin/bash
#SBATCH --job-name=slurm_job_one      # Job name    (default: sbatch)
#SBATCH --output=slurm_job_one-%j.out # Output file (default: slurm-%j.out
    )
#SBATCH --error=slurm_job_one-%j.err  # Error file  (default: slurm-%j.out
    )
#SBATCH --ntasks=1                    # Number of tasks
#SBATCH --cpus-per-task=1             # Number of CPUs per task
#SBATCH --time=00:01:00               # Wall clock time limit
#SBATCH --partition=slim
#SBATCH --nodelist=icsnode25

# load some modules & list loaded modules
module load gcc
module list

# print CPU model
lscpu | grep "Model name"

# run (srun: run job on cluster with provided resources/allocation)
srun ./hello_worldc
```

Listing 9: Tmakefile of Compiler options

```bash
#!/bin/bash
#SBATCH --job-name=slurm_job_two      # Job name    (default: sbatch)
#SBATCH --output=slurm_job_two-%j.out # Output file (default: slurm-%j.out
    )
#SBATCH --error=slurm_job_two-%j.err  # Error file  (default: slurm-%j.out
    )
#SBATCH --nodes=2                     # Number of nodes
#SBATCH --ntasks=2                    # Number of tasks
#SBATCH --cpus-per-task=1             # Number of CPUs per task
#SBATCH --time=00:01:00               # Wall clock time limit
#SBATCH --partition=slim
#SBATCH --nodelist=icsnode26,icsnode27

# load some modules & list loaded modules
```

```
module load gcc
module list

# print CPU model
lscpu | grep "Model␣name"

# run (srun: run job on cluster with provided resources/allocation)
```

Listing 10: The batch script of two node

```
#!/bin/bash
#SBATCH -A es_math                          # Account (es_math or ls_math)
#SBATCH --job-name=slurm_stream_c_1         # Job name    (default: sbatch)
#SBATCH --output=slurm_stream_c_1-%j.out # Output file (default: slurm-%j.
    out)
#SBATCH --error=slurm_stream_c_1-%j.err  # Error file  (default: slurm-%j.
    out)
#SBATCH --ntasks=1                          # Number of tasks
#SBATCH --cpus-per-task=1                   # Number of CPUs per task
#SBATCH --time=00:01:00                     # Wall clock time limit

# load some modules & list loaded modules
module list
module load gcc

# print CPU info
lscpu | grep "Model␣name"

# compile & run
make clean
make
srun ./stream_c_1
# code from course material
```

Listing 11: The batch scripts of STREAM benchmark

```
import matplotlib.pyplot as plt
import numpy as np


def plot_roofline(Pmax, bmax, Imin, Imax, N=1000, ax=None, **plt_kwargs):
    if ax is None:
        ax = plt.gca()
    I = np.logspace(np.log(Imin), np.log(Imax), N)
    P = bmax * I
    P = np.minimum(P, Pmax)
    ax.loglog(I, P, **plt_kwargs)
    ax.set_xscale(’log’, base=2)
    # ax.set_yscale(’log’, base=2)
    ax.grid(True)
    ax.set_xlim(Imin, Imax)
    ax.set_xlabel(rf"Operational␣Intensity␣$I$␣[Flops/Byte]")
    ax.set_ylabel(rf"Performance␣$P$␣[GFlops/s]")
    return ax


if __name__ == "__main__":
    fig, ax = plt.subplots()
    ax = plot_roofline(Pmax=42, bmax=12, Imin=1.e-2, Imax=1.e+3, ax=ax,
                       label="Intel(R)␣Xeon(R)␣CPU␣E5-2650␣v3␣@␣2.30GHz")
    ax.legend()
    plt.savefig("roofline.pdf")
    plt.show()
* code from course material
```

Listing 12: Draw the picture roofline model

B. Appendix: Terminal Output

```
wangzi@icslogin01 ~ (0.055s)
sinfo
PARTITION  AVAIL   TIMELIMIT     NODES    STATE NODELIST
slim*      up      2-00:00:00    8        plnd icsnodel [19-261]
slim*      up      2-00:00:00    7        mix icsnode[17-18, 32-361]
slim*      up      2-00:00:00    2        alloc icsnode[37-381]
gpu        up      2-00:00:00    1        idle icsnode[27-31]
gpu        up      2-00:00:00    1        mix icsnode05
gpu        up      2-00:00:00    8        alloc icsnode06
gpu        up      2-00:00:00    4        idle icsnodel [08-15]
bigMem     up      2-00:00:00    2        idle icsnode[01-04]
bigMem     up      2-00:00:00    2        idle icsnode[07, 15]
debug-slim up      4:00:00       1        idle icsnode39
debug-gpu  up      4:00:00       1        plnd icsnode16
multi_gpu  up      2-00:00:00    2        idle icsnode[41-42]
```

Listing 13: Basic information of nodes

```
wangzi@icslogin01 ~/src2/1-Rosa-warm-up (0.0295)
cat slurm_job_one-10854.out
Currently Loaded Modulefiles:
1) gec-runtime/8.5.0-gcc-8.5.0-7fyorqa
2) gmp/6.2.1-gcc-8.5.0-1rpcvy5
3) mpfr/4.2.1-gcc-8.5.0-ybeybcx
4) mpc/1.3.1-gcc-8.5.0-cv2gjfw
5) zlib-ng/2.1.6-gcc-8.5.0-ztbc5xt
6) zstd/1.5.5-gcc-8.5.0-azepnn7
7) gcc/13.2.0-gcc-8.5.0-5hqhkwo
Model name:
Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz
Hello World
host(node) name: icsnode25
```

Listing 14: The Result of batch script of one node

```
wangzi@icslogin01 ~/src2/1-Rosa-warm-up (0.0295)
cat slurm_job_two-10873.out
Currently Loaded Modulefiles:
1) gec-runtime/8.5.0-gcc-8.5.0-7fyorqa
2) gmp/6.2.1-gcc-8.5.0-1rpcvy5
3) mpfr/4.2.1-gcc-8.5.0-ybeybcx
4) mpc/1.3.1-gcc-8.5.0-cv2gjfw
5) zlib-ng/2.1.6-gcc-8.5.0-ztbc5xt
6) zstd/1.5.5-gcc-8.5.0-azepnn7
7) gcc/13.2.0-gcc-8.5.0-5hqhkwo
Model name:
Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz
Hello World
host(node) name: icsnode[26-27]
```

```
Hello World
host(node) name: icsnode[26-27]
```

Listing 15: The Result of batch script of two node