Università
della
Svizzera
italiana

Institute of
Computing
CI

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

Student: Paolo Deidda Discussed with: FULL NAME

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 source files (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 .

Contents

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

1.1. exercise 1

The **module system** is a utility that allows the user to dynamically manage their software environment on the Rosa HPC cluster and to load different compilers, libraries and applications in order to modify environment variagles (like PATH, LD LIBRARY PATH, MANPATH, etc.) without creating conflicts between different software versions or dependencies.

As riported in the USI resource page the module system provides several commands to manage the environment.

- `module avail` – lists all available modules (on the current system)
- `module list` – lists all currently loaded modules
- `module show` – display information about
- `module load` – loads module
- `module switch` – unloads, loads
- `module rm` – unloads module
- `module purge` – unloads all loaded modules

1.2. exercise 2

The **Slurm** (Simple Linux Utility for Resource Management) is a job scheduler for Linux clusters. Main features of Slurm are:

- Job scheduling and resource management
- Framework for starting, executing, and monitoring work (jobs) on a set of allocated nodes
- Queuing management to handle multiple users and jobs

The two main components are:

- *sulurmd*: the deamnon that runs on each compute node responsible for launching, monitoring, and terminating jobs
- *slurmctld*: the central management daemon that manages job queues and allocates resources

Main commands:

- `srun`: submit a job for execution
- `sbatch`: submit a batch job
- `squeue`: view the status of jobs in the queue
- `scancel`: cancel a job
- `salloc`: allocate resources for an interactive job

1.3. exercise 3

Here below is a simple program in C that prints "Hello World" and the information about the system where it is executed.

```c
#include <stdio.h>
#include <unistd.h>

int main(void) {
    char hostname[256];
    gethostname(hostname, sizeof(hostname));
    printf("Hello World from host: %s\n", hostname);
}
```

Listing 1: Hello World C Program - *src/1-Rosa-warm-up/hello_ world.c*

We can process the script with the following command:

`srun -N1 --time=00:01:00 ./hello_worldc > hello_worldc.out 2> hello_worldc.err`

and we can see from the output that the program has been correctly compiled and executed on the cluster on node `icsnode22` from the *slim* partion.

```
Hello World from host: icsnode22
```

Listing 2: Output of the program `hello_world.c`

1.4. exercise 4

We can see the output of the command `sinfo` here below:

```
PARTITION  AVAIL  TIMELIMIT  NODES  STATE NODELIST
slim*         up 2-00:00:00      7    mix icsnode[22,27-28,32-35]
slim*         up 2-00:00:00     12  alloc icsnode[17-21,23-26,29-31]
slim*         up 2-00:00:00      2   idle icsnode[36-37]
lr-slim       up 30-00:00:0      1   idle icsnode38
gpu           up 2-00:00:00      8    mix icsnode[05-06,08-13]
gpu           up 2-00:00:00      2   idle icsnode[14-15]
fat           up 2-00:00:00      4   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   idle icsnode16
multi_gpu     up 2-00:00:00      2   idle icsnode[41-42]
```

Listing 3: Output of line command `sinfo`

As we can see nodes are divided in partitions with names that already give us some information about their characteristics. As explained in the sbatch guide on slurm website, we can use different flags on the sbatch command to specify the partition to use with different commands.

The flag that applies to our case is:

`sbatch –partition=fat job_script.sh`

Submit with bigMem partition to run on nodes with very large memory:

`sbatch –partition=bigMem job_script.sh`

Similarly, for GPU partitions:

`sbatch –partition=gpu job_script.sh`

For example, we can modify the script file `slurm_job_one.sh` to specify the partition with the gpu partition with the following line:

`#SBATCH --partition=gpu`

After inserting the line into the file and reprocessing the script, we can see the following result:

```
Loading gcc/13.2.0-gcc-8.5.0-5hqhkwo
  Loading requirement: gcc-runtime/8.5.0-gcc-8.5.0-7fyorqa
    gmp/6.2.1-gcc-8.5.0-lrpcvy5 mpfr/4.2.1-gcc-8.5.0-ybeybcx
    mpc/1.3.1-gcc-8.5.0-cv2gjfw zlib-ng/2.1.6-gcc-8.5.0-ztbc5xt
    zstd/1.5.5-gcc-8.5.0-azepnn7
Currently Loaded Modulefiles:
 1) gcc-runtime/8.5.0-gcc-8.5.0-7fyorqa   5) zlib-ng/2.1.6-gcc-8.5.0-ztbc5xt
 2) gmp/6.2.1-gcc-8.5.0-lrpcvy5           6) zstd/1.5.5-gcc-8.5.0-azepnn7
 3) mpfr/4.2.1-gcc-8.5.0-ybeybcx          7) gcc/13.2.0-gcc-8.5.0-5hqhkwo
 4) mpc/1.3.1-gcc-8.5.0-cv2gjfw
Model name:          Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz
Hello World from host: icsnode08
```

Listing 4: Output of the job script `slurm_job_one.sh` after specifying the partition

From the very last line we can assert that the job has been submitted to node `icsnode08`, and from the `sinfo` output before (3) we can see that this node belongs to the *gpu* partition instead of the default *slim* partition as before (2).

1.5. exercise 5

In order to run our program on two nodes is sufficient to add the following line of our script:

```
#SBATCH --nodes=2                          # Number of nodes
```

This line is already implemented in the script `slurm_job_two.sh` provided in the src folder. Once we process the script with the command we get the the message Submitted batch job 51103 and after a while we can check the output file slurm_job_two-51103.out (5) to see the result of our job.

```
Loading gcc/13.2.0-gcc-8.5.0-5hqhkwo
  Loading requirement: gcc-runtime/8.5.0-gcc-8.5.0-7fyorqa
    gmp/6.2.1-gcc-8.5.0-lrpcvy5 mpfr/4.2.1-gcc-8.5.0-ybeybcx
    mpc/1.3.1-gcc-8.5.0-cv2gjfw zlib-ng/2.1.6-gcc-8.5.0-ztbc5xt
    zstd/1.5.5-gcc-8.5.0-azepnn7
Currently Loaded Modulefiles:
 1) gcc-runtime/8.5.0-gcc-8.5.0-7fyorqa   5) zlib-ng/2.1.6-gcc-8.5.0-ztbc5xt
 2) gmp/6.2.1-gcc-8.5.0-lrpcvy5           6) zstd/1.5.5-gcc-8.5.0-azepnn7
 3) mpfr/4.2.1-gcc-8.5.0-ybeybcx          7) gcc/13.2.0-gcc-8.5.0-5hqhkwo
 4) mpc/1.3.1-gcc-8.5.0-cv2gjfw
Model name:          Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz
Hello World from host: icsnode22
Hello World from host: icsnode21
```

Listing 5: Output of the job script `slurm_job_two.sh`

From the output we can see that the job has been submitted to two different nodes `icsnode22` and `icsnode21` confirming that the command has been correctly processed twice on two different nodes.

2. Performance Characteristics *(30 Points)*

2.1. Peak performance

Rosa's nodes are equipped with dual-socket Intel Xeon E5-2650 v3 processors, 10 cores and AVX2 units that operates with 256-bit vectors[1, 3]. Haswell micro-architecture have two 256-bit FMA instructions per cycle meaning that each core performs four double-precision at 2.30 GHz [2].

The Rosa documentation (and also the output from command `sinfo` 3) shows 42 compute nodes (icsnode01–icsnode42).[3]
The calculations for the aggregate peak throughput for the partition is written in the provided file [INTEL_XEON_E5-2650.txt] in the 02 folder and reported here below:

```
Intel Xeon E5-2650 v3 @ 2.30GHz:

P_{core} = 4 lanes x 2 flop x 2 FMA x 2.30 GHz = 36.8 GFlops/s
P_{CPU}  = 10 cores x P_{core}                 = 368 GFlops/s
P_{node} = 2 sockets x P_{CPU}                 = 736 GFlops/s
P_{EVII} = 42 nodes x P_{node}                 = 30,912 GFlops/s
                                               = 30.912 TFlops/s
```

Listing 6: Peak throughput breakdown for Intel Xeon E5-2650 v3

2.2. Memory Hierarchies

In order to study the memory hierarchy of Rosa compute nodes, I used the commands: `lscpu`, `cat /proc/meminfo`, and `hwloc-ls`. The complete output files are available in the submission in the folder 02.

The results are summarized in Table 1.

Component	**Size**
Main memory (total)	62 GB
Main memory (per NUMA node)	31 GB
L3 cache (shared per socket)	25 MB
L2 cache (per core)	256 KB
L1d cache (per core)	32 KB
L1i cache (per core)	32 KB

Table 1: Memory hierarchy of Rosa compute node

The graphical representation of the memory hierarchy is shown in Figure 1.

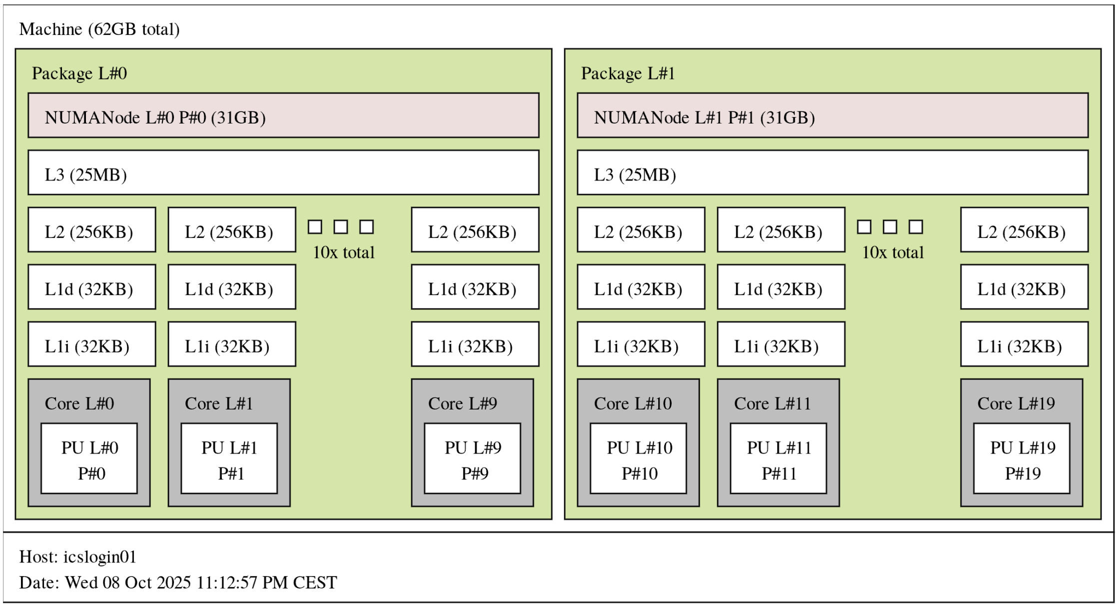


Figure 1: Graphical representation of Rosa node topology generated by command `hwloc-ls`.

2.3. Bandwidth: STREAM benchmark

The STREAM benchmark was executed on a single core of a Rosa consisting of four simple vector operations: Copy, Scale, Add, and Triad.

As required I runned the command:

```
gcc -O3 -march=native -DSTREAM_TYPE=double -DSTREAM_ARRAY_SIZE=128000000
    -DNTIMES=20 stream.c -o stream_c.exe
```

So the array size is configured to 128,000,000 elements (3GB) which significantly exceeds the L3 cache size (25MB), ensuring that the operations always "fall" out of the cache and therefore forcing the use main memory. This is done to **measure the bandwidth of main memory rather than cache performance.** Each kernel has been executed 20 times, and the best time (excluding the first iteration) I used to compute the reported bandwidth.

The complete output is available in the submission folder. The key results are shown below:

```
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 128000000 (elements), Offset = 0 (elements)
Memory per array = 976.6 MiB (= 1.0 GiB).
Total memory required = 2929.7 MiB (= 2.9 GiB).
Each kernel will be executed 20 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 116773 microseconds.
   (= 116773 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           19174.9     0.106875     0.106806     0.107031
Scale:          11261.5     0.181919     0.181859     0.182046
Add:            12303.3     0.249789     0.249689     0.249905
Triad:          12309.1     0.249664     0.249572     0.249827
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

Listing 7: STREAM benchmark output on Rosa compute node (single-core)

The Copy operation shows substantially higher bandwidth (approximately 19GB/s) compared to the others.

The **Triad kernel bandwidth** of approximately 12.3 GB/s will be the average representative as memory bandwidth for a single core on Rosa compute nodes.

2.4. Performance model: A simple roofline model

Using the values obtained from the previous sections:

- Peak performance per core: $P_{max} = 36.8$ GFlops/s (Section 2.1)
- Memory bandwidth per core: $b_{max} = 12.3$ GB/s (Section 2.3)

The ridge point (performance transitions from memory-bound to compute-bound) occurs at:

$$I_{ridge} = \frac{P_{max}}{b_{max}} = \frac{36.8}{12.3} \approx 2.99 \text{ Flops/Byte}$$

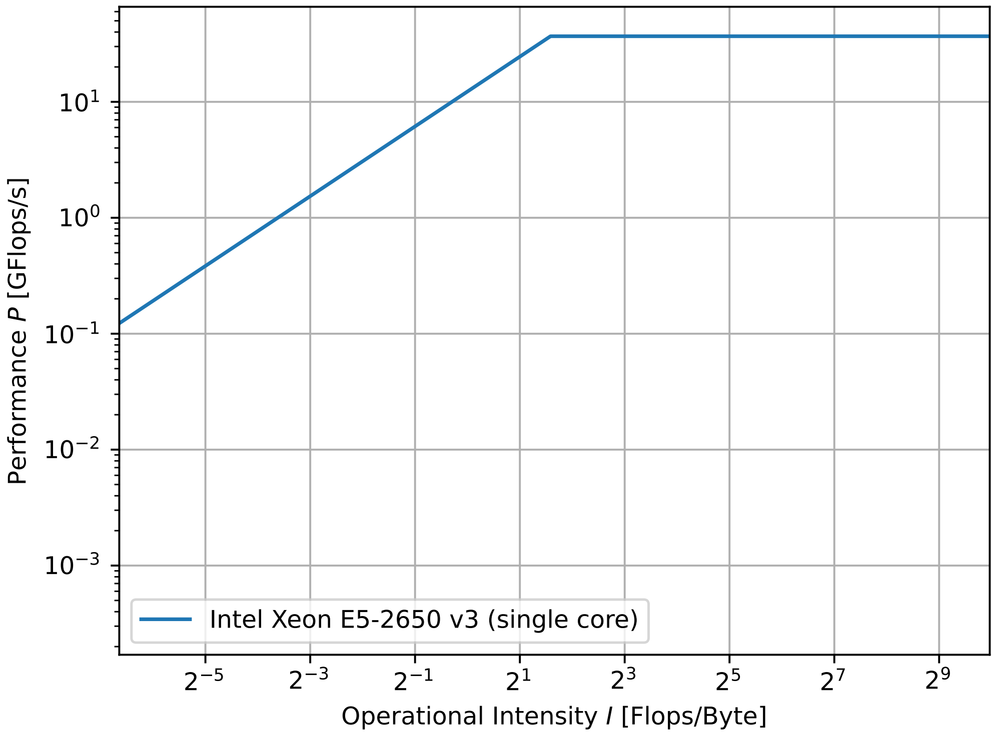
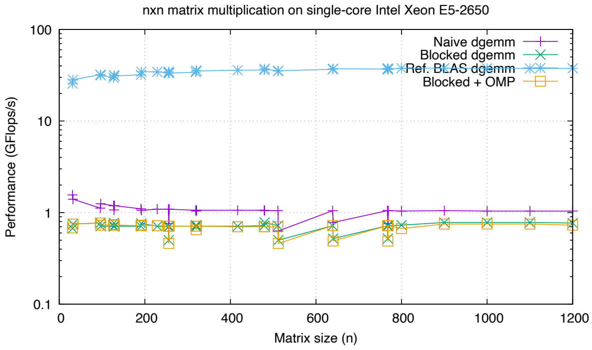


Figure 2: Roofline model for Intel Xeon E5-2650 v3 (single core) on Rosa compute node.

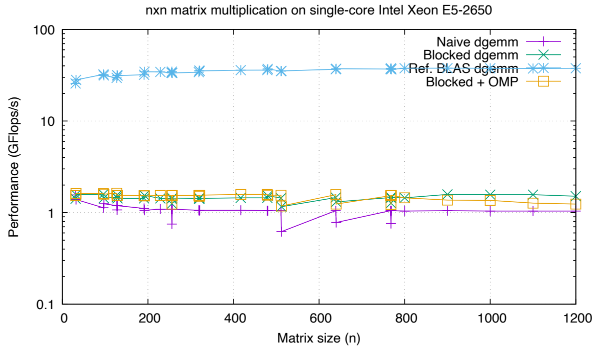
3. Optimize Square Matrix-Matrix Multiplication *(50 Points)*

I implemented the blocked DGEMM as prescribed in the assignment and processed it on the cluster with different block sizes; the plots in Figure 3 collect the timing traces.

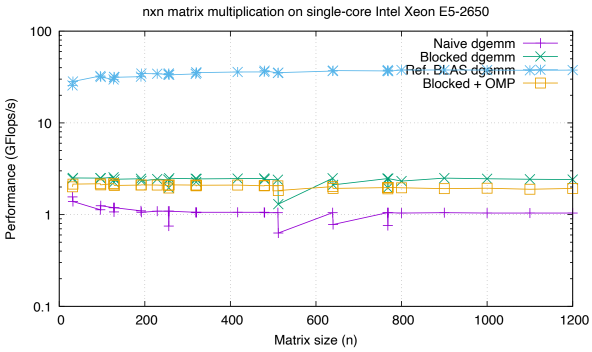
I produced dgemm-blocked-omp.c implementing the requested OpenMP parallelism, compiler flags, and the same block-size sweep to compare the tuned path. Unfortunately, the OpenMP version **did not** outperform the baseline in any configuration and I'm not sure why.



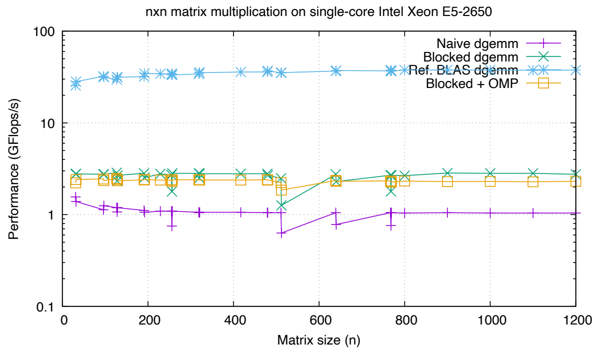
(a) Block size = 2



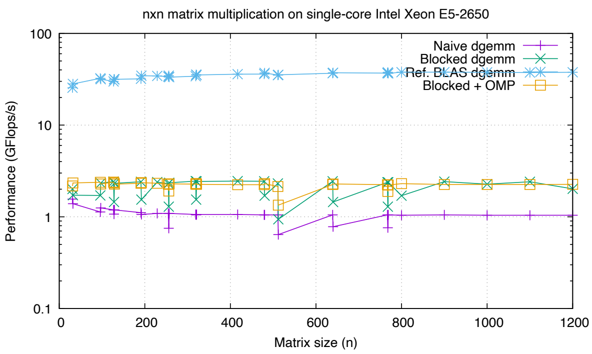
(b) Block size = 4



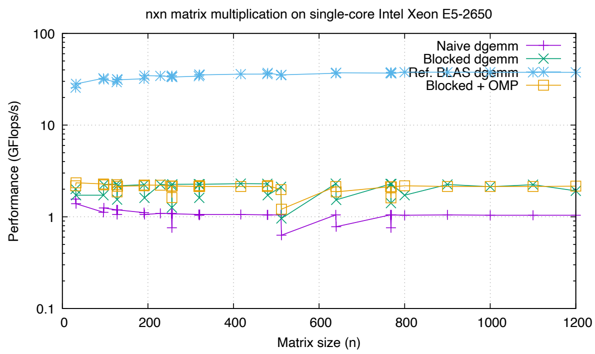
(c) Block size = 8



(d) Block size = 16



(e) Block size = 32



(f) Block size = 64

Figure 3: Performance comparison of different DGEMM implementations with different block sizes

I summarized each run from the `matrixmult-*.out` logs in results/ and reported them in Table 2. This includes the averages for each model and each block size. The rough binary search still points to block size 16 as the best baseline (7.06% of peak). A more refined search could still reveal a stronger configuration.

Implementation	Block Size	Blocked Avg. %	Blocked + OMP Avg. %
Naive DGEMM	–	2.88%	–
BLAS DGEMM	–	93.77%	–
Blocked DGEMM	2	1.91%	1.87%
	4	3.94%	4.01%
	8	6.41%	5.53%
	16	7.06%	6.35%
	32	5.55%	6.04%
	64	5.36%	5.66%

Table 2: Average Percentage of Peak Performance

The tuned OpenMP path gives the higher block sizes a lift (for example +0.5 points at block size 32), so it is the version I would keep scaling even if the single-node sweet spot stays around block size 16.

4. Quality of the Report *(15 Points)*

References

[1] Intel Corporation. Intel® xeon® processor e5-2650 v3 (25m cache, 2.30 ghz) product specifications, 2014.

[2] Intel Corporation. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*, 2023.

[3] Università della Svizzera italiana, Advanced Computing Lab. Rosa cluster hardware overview, 2024.