ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

HPC Lab for CSE

FS 2024
Dr. R. Käppeli,
Dr. A. Schell, M. R. Herde,
C. Humbel, A. Mazzoleni,
N. A. Siegenheim

Project 1a

Due date: 11 March 2024, 11:59pm (midnight)

The first part of this project (a) will introduce you to using the Euler cluster and collecting some performance characteristics. The second part of the project (b) will be about the optimization of general matrix-matrix multiplication. Both project parts will be serial. Parallel programming will be the subject of the forthcoming projects.

1 Euler warm-up [10 points]

Review the Euler cluster documentation[1] to gain an understanding of its features and usage. Answer briefly the following questions in your report:

1. What is the module system and how do you use it?
2. What is Slurm and its intended function?
3. Write a simple "Hello World" `C`/`C++` program which prints the host name of the machine on which the program is running.
   function with some appropriate size
4. Write a batch script which runs your program on one node. The batch script should be able to target nodes with specific CPUs.
   Hint: Slurm hast the option `--constraint` to select some nodes with specific features. Possible options to select nodes featuring a certain CPU: `EPYC_7742`, `EPYC_7H12`, `EPYC_7763`, `EPYC_9654`.
5. Write another batch script which runs your program on two nodes. Check that the Slurm output file (`slurm-*.out` by default) contains two different host names.

Include the source code, batch scripts, and Slurm output files in your submission (see the notes at the end of the document).

2 Performance characteristics [50 points]

In the realm of HPC, understanding the performance limits of computing systems, particularly in relation to the memory system and floating-point operations, is crucial for designing efficient algorithms and optimizing their implementations on (a particular) hardware. The performance characteristics and concept of lightspeed estimates comes into play when considering the physical constraints that limit data transfer rates and computation speeds within a computing system. These constraints are not only dictated by the raw processing power of the CPU but also by the latency and throughput of the memory hierarchy. It is essential to grasp these concepts in order to appreciate the theoretical and practical limits of computing, including how data locality and memory bandwidth impact the achievable performance. This knowledge is fundamental in pushing the boundaries of what is computationally possible while being mindful of the inherent practical limitations posed by physics and current technology.

However, determining the performance characteristics of computing systems is often a complex and nuanced task that requires a multi-faceted approach. Manufacturer documentation provides a baseline of theoretical capabilities and specifications, but real-world performance can deviate significantly from these idealized scenarios due to a myriad of factors. Web resources, including insights from HPC centers, offer valuable empirical data and analyses that reflect the performance characteristics. Yet another source of information is experimentation in the form of low-level benchmarking. Thus, a combination of scrutinizing manufacturer documentation, leveraging the wealth of information available from HPC centers and other

[1] https://scicomp.ethz.ch

web resources, and conducting methodical benchmarking experiments is essential for a comprehensive understanding of computing system performance.

The goal of this task is to collect such performance characteristics and lightspeed estimates. As a refresher (or a concise introduction), we *strongly* encourage you to read the article and appendix [3], and Chapters One and Three of the book [1][2].

2.1 Peak performance

In this task, we ask you to compute the core, CPU, node and cluster peak performance for the Euler VII (phase 1 and 2) nodes. Please detail your computation and all sources in your report. In the following, we provide some context and give a step-by-step guide to complete and report the task.

In scientific computing, the focus often lies on floating-point (FP) data, typically utilizing *double precision*. The rate at which the CPU's floating-point unit (FPU) can produce results for multiplication and addition operations is quantified in terms of *floating-point operations per second* (Flops/s, or simply FLOPS). The maximum rate at which a system is capable of executing Flops/s is the so-called (theoretical[3]) *peak (FP) performance*. The peak performance P_{core} of a single core is computed as follows

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

where

- n_{super} is the superscalarity factor: the number of FP operations the FPU of a core can execute in parallel within a single clock cycle.
- n_{FMA} is the fused multiply-add factor: $n_{\text{FMA}} = 2$ if the FPU supports the FMA instruction (enabling it to perform a multiplication and an addition in a single operation), or $n_{\text{FMA}} = 1$ if the FPU does not support FMA (requiring separate instructions for multiplication and addition).
- n_{SIMD} is the SIMD (Single Instruction, Multiple Data) factor: the number of doubles the FPU can process concurrently with a single SIMD instruction (i.e., the width of the SIMD registers in units of doubles).
- f is the (base) clock frequency (modern multi-core CPUs may dynamically increase their clock frequency to make use of Thermal Design Power (TDP) more efficiently if fewer cores are active).

The peak performance P_{CPU} of a CPU is then

$$P_{\text{CPU}} = n_{\text{cores}} \times P_{\text{core}},$$

where n_{cores} is the number of *physical* cores[4]. The peak performance P_{node} of a node with n_{sockets} identical CPUs

$$P_{\text{node}} = n_{\text{sockets}} \times P_{\text{CPU}}.$$

Finally, the peak performance P_{cluster} of a cluster consisting of n_{nodes} identical nodes is

$$P_{\text{cluster}} = n_{\text{nodes}} \times P_{\text{node}}.$$

Let's compute the peak performance of the Euler III (2016-2022) nodes as an example. The system comprised 1215 nodes (i.e., $n_{\text{nodes}} = 1215$), each equipped with a single Intel Xeon E3-1585Lv5 CPU (i.e., $n_{\text{sockets}} = 1$). According to the Euler webpage, which conveniently provides detailed specifications, each of these CPUs has four cores (i.e., $n_{\text{cores}} = 4$). Additionally, the link to Intel's ARK website offers further technical details, revealing that the CPU operates at a base clock frequency of $f = 3.00$ GHz and supports AVX2 SIMD instructions with 256-bit wide vector registers. This setup allows for processing four 64-bit double-precision FP numbers simultaneously, leading to $n_{\text{SIMD}} = 4$.

The remaining factors to consider for calculating peak performance are the superscalarity and FMA factors. However, these details are not as readily available and typically require a deep dive into technical documents provided by the CPU manufacturer or from technology review sites and academic research. In this specific instance, valuable information is found in Figure 2-8 of Intel's Optimization Reference Manual Volume 1 (around p. 84 in the PDF). This figure indicates that Ports 0 and 1 each can perform

[2] To gain access, you have to be within the ETH network, e.g., by using the VPN.

[3] There is a distinction between theoretical and measured peak performance, but we will ignore it for the time being (see here)

[4] As opposed to *logical* cores often present on modern CPUs with *threading* capabilities, such as Hyper-threading (HT) for Intel CPU or Simultaneous Multithreading (SMT) for AMD.

one vector FMA operation (i.e., $n_{\text{FMA}} = 2$), establishing the superscalarity factor $n_{\text{super}} = 2$. Hence, we can compute the (theoretical) peak performance of the Euler III nodes

$$\begin{aligned} P_{\text{core}} &= 2 \times 2 \times 4 \times 3 \text{ GHz} = 48 \text{ GFlops/s}, \\ P_{\text{CPU}} &= 4 \times P_{\text{core}} = 192 \text{ GFlops/s}, \\ P_{\text{node}} &= 1 \times P_{\text{CPU}} = 192 \text{ GFlops/s}, \\ P_{\text{Euler III}} &= 233'280 \text{ GFlops/s} = 233.28 \text{ TFlops/s} \end{aligned}$$

Here are some useful hints for the task:

- Euler VII reveals that each node has two AMD EPYC 7H12 CPUs and provides a link to AMD's general specifications. From AMD's specifications you get that the CPU is part of the AMD EPYC 7002 Series, from which you deduce that it is based on the Zen 2 microarchitecture (codenamed "Rome"). For example, from https://en.wikipedia.org/wiki/Epyc#History.
- Euler VII reveals that each node has two AMD EPYC 7742 CPUs and provides a link to AMD's general specifications. From AMD's specifications you get that the CPU is part of the AMD EPYC 7003 Series, from which you deduce that it is based on the Zen 3 microarchitecture (codenamed "Milan").
- The uops.info website will then provide you with the superscalarity and FMA factors. Instead, you could consult AMD's documentation for these factors. However, this approach is slightly more complex and can be avoided.
- Remember the vector SIMD register naming convention https://en.wikipedia.org/wiki/Advanced_Vector_Extensions#Advanced_Vector_Extensions.
- Remember that ThroughPut (TP) is defined by how many cycles it takes to execute one instruction.

In general (beyond this project and course), the following resources are valuable:

- Intel's software and optimization manuals: The easiest way to find them is to search for "Software Developer's Manual" or "Optimization Reference Manual" in your favorite web search engine.
- AMD's software and optimization manuals: The easiest way to find them is to search for "AMD documentation hub" in your favorite web search engine. Once on the hub, search for "Software optimization guide" specific to the CPU model you are interested.
- Intel Intrinsics Guide
- Agner Fog's software optimization resources: https://www.agner.org/optimize/
- Latency and throughput, listing: https://uops.info/.
- https://en.wikichip.org: Can be a useful resource (despite the ads).

2.2 Memory Hierarchies

2.2.1 Cache and main memory size

The next task is to identify the parameters of the memory hierarchy on a node of the Euler VII (phase 1 and 2) cluster. Follow the step-by-step guide provided below and detail your results in your project report. Please also mention if you observe any differences between the CPUs of phase 1 and 2.

To guide you, we show how this can be achieved for an Euler login node. A useful tool to determine the CPU architecture is `lscpu`:

```
[user@eu-login-39 ~]$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    4
Socket(s):             1
```

```
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 71
Model name:            Intel(R) Xeon(R) CPU E3-1284L v4 @ 2.90GHz
Stepping:              1
CPU MHz:               3700.048
CPU max MHz:           3800.0000
CPU min MHz:           800.0000
BogoMIPS:              5799.57
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              6144K
L4 cache:              131072K
... TRUNCATED ...
```

From this we obtain the CPU model (Intel(R) Xeon(R) CPU E3-1284L v4 @ 2.90GHz) featuring four cores and some basic information on the memory hierarchy. To obtain the total available main memory, one option is to look at the `/proc/meminfo`:

```
[user@eu-login-39 ~]$ cat /proc/meminfo
MemTotal:       32871584 kB
... TRUNCATED ...
```

In summary, we have collected so far the information listed in Table 1. Since this is a multi-core CPU, it is valuable to obtain more information on how the memory hierarchy is organized and shared among the cores. A (possible; there are others) tool for this is provided by `hwloc`. In particular, the following `hwloc-ls` commands will give us the desired information:

```
[user@eu-login-39 ~]$ hwloc-ls --whole-system --no-io
Machine (31GB) + Package L#0 + L4 L#0 (128MB) + L3 L#0 (6144KB)
  L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0 + PU L#0 (P#0)
  L2 L#1 (256KB) + L1d L#1 (32KB) + L1i L#1 (32KB) + Core L#1 + PU L#1 (P#1)
  L2 L#2 (256KB) + L1d L#2 (32KB) + L1i L#2 (32KB) + Core L#2 + PU L#2 (P#2)
  L2 L#3 (256KB) + L1d L#3 (32KB) + L1i L#3 (32KB) + Core L#3 + PU L#3 (P#3)
```

From the output, we conclude that each of the four cores has its own L1 instruction (L1i) and data (L1d) caches as well as an L2 cache. The L3 and L4 caches, along with the main memory, are shared among all cores. This observation has been noted in the caption of Table 1 to provide a complete overview. A graphical representation of the memory hierarchy is displayed in Fig. 1. To obtain such a figure, first ask `hwloc-ls` to output in the "fig" format

```
[user@eu-login-39 ~]$ hwloc-ls --whole-system --no-io -f --of fig XEON_E3-1284L.fig
```

Unless you have Xfig[5] installed on your machine, you can convert `XEON_E3-1284L.fig` by copying it on one of the `slab1` machines and converting it to a PDF file:

```
[user@eu-login-39 ~]$ scp XEON_E3-1284L.fig user@slab1.ethz.ch:
[user@eu-login-39 ~]$ ssh user@slab1.ethz.ch
[user@slab1vrt ~]$ fig2dev XEON_E3-1284L.fig XEON_E3-1284L.pdf
```

Of course, you will have to copy the resulting PDF to your own machine when writing your project report.

For more information on these commands and their options, please have a look at there man pages (e.g., `man lscpu`).

[5] https://www.xfig.org/

Main memory	32 GB
L4 cache	128 MB
L3 cache	6 MB
L2 cache	256 KB
L1 cache	32 KB

Table 1: Memory hierarchy of an Euler login node with an Intel(R) Xeon(R) CPU E3-1284L v4 @ 2.90GHz. Each core has its own L1 and L2 cache, while L3 and L4 caches are shared among all the four cores.

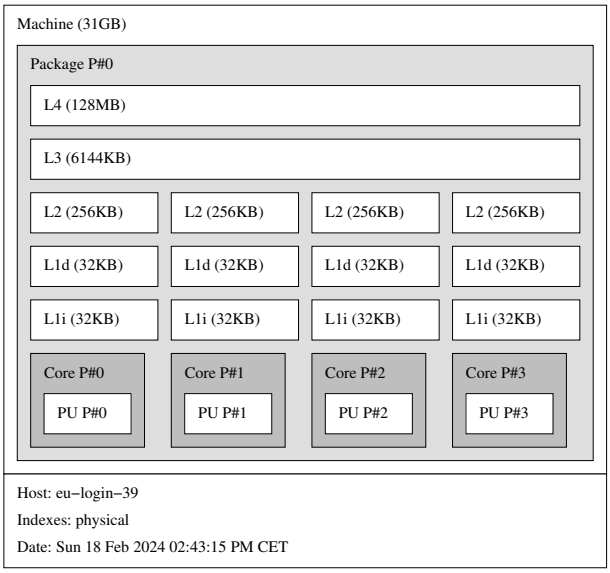

Figure 1: Schematic of an Euler login node with an Intel(R) Xeon(R) CPU E3-1284L v4 @ 2.90GHz.

2.3 Bandwidth: STREAM benchmark

Now that we have an overview of the peak performance, as well as the topology and size of the memory system, another key performance characteristic to consider is the speed of the memory system. The speed of the memory system is quantitatively measured in terms of its bandwidth, which indicates the amount of data that can be transferred within a specific time frame. McCalpin's STREAM benchmark [2] is a widely recognized tool used for measuring the memory bandwidth of a CPU. It consists of four operations or kernel — Copy, Scale, Add, and Triad — that evaluate the sustainable memory bandwidth and the corresponding computation rate for simple vector kernels (see, e.g, [1, Sec. 3.1.2]).

In this task, we ask you to run the STREAM benchmark on the Euler VII (phase 1 and 2) nodes in order to measure the single-core bandwidth. Find below a step-by-step guide:

1. Download the STREAM Benchmark: it can be obtained from the official website[6] in the source code directory[7]. You will need `stream.c` and `mysecond.c`.

2. In `stream.c`, you will find precise instructions on how to compile and run the benchmark. It is particularly important to tune the sizes of the arrays (used in the Copy, Scale, Add, and Triad kernels) to match the cache sizes of the system of interest: Each array must be at least four times the size of the last cache level. This is set by the `STREAM_ARRAY_SIZE` preprocessor macro and two examples are given in the `stream.c`. For the CPU studied in the example of Section 2.2.1, we would then

```
module load gcc # load a compiler (gcc/11.4.0)
gcc -O3 -march=native -DSTREAM_TYPE=double -DSTREAM_ARRAY_SIZE=68000000 \
    -DNTIMES=20 stream.c -o stream_c.exe # compile
sbatch --mem-per-cpu=2G --wrap "./stream_c.exe" # submit to queue and run
                                                # (require enough memory!)
```

[6] https://www.cs.virginia.edu/stream/
[7] https://www.cs.virginia.edu/stream/FTP/Code/

3. The output (`sluem-jobid.out`) produced:

```
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 68000000 (elements), Offset = 0 (elements)
Memory per array = 518.8 MiB (= 0.5 GiB).
Total memory required = 1556.4 MiB (= 1.5 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 61118 microseconds.
   (= 61118 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:           19934.4     0.055412     0.054579     0.056215
Scale:          12020.9     0.092785     0.090509     0.094412
Add:            12872.5     0.129225     0.126782     0.131574
Triad:          12780.8     0.129039     0.127692     0.130467
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

   We observe that the bandwidths resulting from the Scale, Add, and Triad kernels are roughly consistent, while the bandwidth for the Copy kernel appears to be substantially higher. There may be several explanations for this discrepancy, but they are beyond the scope of this project (and course). For a rough estimate, we can assume a maximum bandwidth $b_{\text{STREAM}} = 12$ GB/s.

Of course, you will need to adapt some these steps to the task. In your report, follow a similar style to the above description. Include source code (including build files), batch scripts, and Slurm output files in your submission.

For further context and advice regarding the STREAM benchmark, we recommend looking at the official website.

2.4 Performance model: A simple roofline model

The *roofline* model is a visual representation that serves as a powerful tool for evaluating the performance potential of computing systems, especially in the context of high-performance computing [3]. It plots achievable performance (in GFlops/s) against operational intensity (in Flops/Byte), delineating the maximum performance given by the system's peak performance and the memory bandwidth constraints. The "roofline" itself consists of two main segments: the *memory-bound* region, where performance is limited by the system's memory bandwidth, and the *compute-bound* region, where performance is capped by the peak computational power of the processor. The frontier between both regimes is called the *ridge point*. This model allows developers and researchers to identify whether their algorithms are memory-bound or compute-bound and to understand the performance implications of hardware limitations, guiding optimizations towards achieving maximum efficiency within the given hardware constraints.

A simple (or sometimes called naive) roofline model for a hypothetical system combining Sections 2.1 and 2.2.1 is displayed in Fig. 2. We observe that the ridge point is roughly at an operational intensity of $I_{\text{ridge}} \approx 4$. Hence, a given kernel or application with an operational intensity below I_{ridge} is memory-bound. Similarly, a given kernel or application with an operational intensity above I_{ridge} is compute-bound.

Your tasks are:

1. With your performance data collected in the previous tasks, create a roofline model for both single cores of the Euler VII (phase 1 and 2) nodes.
2. Visualize your roofline models in a plot similar to Fig. 2.
3. At what operational intensity is a kernel or application memory/compute-bound?

Include plotting scripts in your submission.

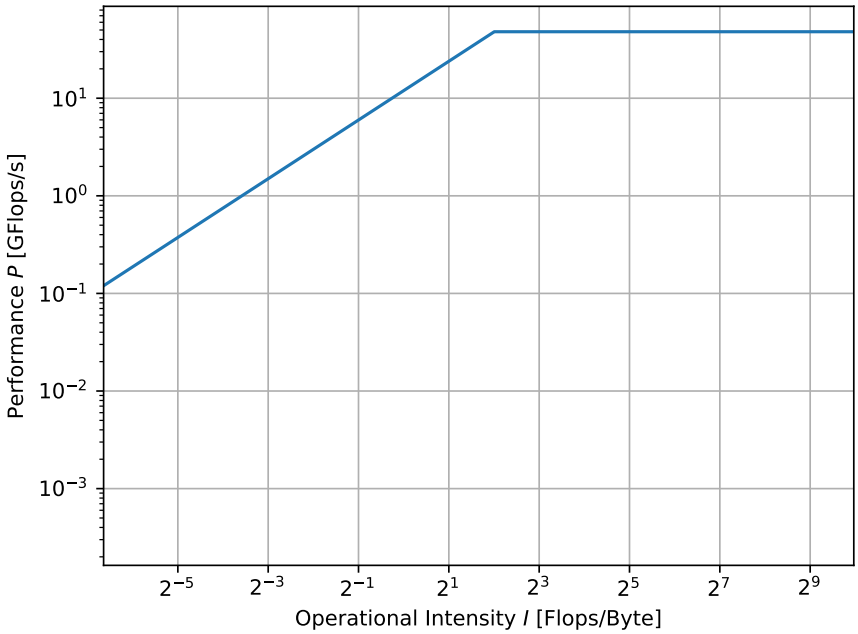

Figure 2: Simple (or naive) roofline model for a single core of a hypothetical CPU.

Additional notes and submission details

Submit **all the source code files** together with your used build files (e.g., `Makefile`(s)) and other scripts in an archive file (e.g., tar, zip, etc.) and summarize your results and observations for all sections by writing a detailed LaTeX report. Use the LaTeX template from the webpage and upload the report as a PDF to Moodle.

- Your submission should be an archive, formatted like `project_number_lastname_firstname.zip/tgz`. It must contain:
  - All the source codes of your solutions.
  - Build files an scripts. If you have modified the provided build files or scripts, make sure they still build the sources an run correctly. We will use them to grade your submission.
  - `project_number_lastname_firstname.pdf`, your report with your name.
  - Follow the provided guidelines for the report.
- Submit your archive file through Moodle.

Please follow these instructions and naming conventions. Failure to comply results in additional work for the TAs, which makes the TAs sad...

References

[1] Georg Hager and Gerhard Wellein. Introduction to high performance computing for scientists and engineers. *Chapman & Hall/CRC Computational Science*, July 2010. URL: http://dx.doi.org/10.1201/EBK1439811924, doi:10.1201/ebk1439811924.

[2] John D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE computer society technical committee on computer architecture (TCCA) newsletter*, 2, December 1995. URL: https://www.cs.virginia.edu/~mccalpin/papers/balance/.

[3] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65, April 2009. doi:10.1145/1498765.1498785.