Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

High-Performance Computing Lab for CSE **2024**

Student: Yannick Ramic Discussed with: FULL NAME

Solution for Project 1

Due date: 11 March 2024, 23:59

HPC Lab for CSE 2024 — Submission Instructions
(Please, notice that following instructions are mandatory: submissions that don't comply with, won't be considered)

- Assignments must be submitted to Moodle (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.

1. Euler warm-up [10 points]

This answers come from the documentation website, hence I won't cite anything properly.

1.1. What is the module system and how do you use it?

While working on the cluster it's necessary to use software or work with programming languages like C++ or Python. To make use of the cluster and use it's potential to work with accelerated GPU and CPU, these dependencies need to be actually installed on the cluster. Thus, to configure the desired environment with a prefered software version, the cluster makes use of modules. For the EULER cluster, two types of modules exist. LMOD modules, which use a hierarchy of modules with three layers (Core -, Compiler - and MPI layer, according to the illustration on the documentation webpage). On the other hand, environment modules have the advantage to make use of pre-installed dependencies by updating the desired software stock. More important information, users are able to install additional applications in their home directory, also there are useful commands to check whether a dependency is installed.

1.2. What is SLURM and its intended function?

SLURM is a workload manager for the management of compute jobs on high-performance computing clusters. It's important to know that users can solely use the cluster ressources through the batch system. During the lecture it was made clear that for each job submitted we need ro

make a request at the cluster, which is done by the described batch system! Thus a job submission command consists of three parts:

1. sbatch (SLURM submit command)
2. SLURM options (requesting resources and job-related options)
3. Job (computing job to be submitted)

To summarize, the intended function is to efficiently delegate and utilize computing resources. Further information about how to submit a job, a parallel job, a GPU job, as well as, how to monitor a job and see the job output can be found on the EULER cluster documentation online!

1.3. Write a simple "Hello World" C/C++ program which prints the host name of the machine on which the program is running.

You can find the corresponding C++ file under the name `hello_world.cpp`.

1.4. Write a batch script which runs your program on one node. The batch script should be able to target nodes with specific CPUs

The resulting files are `slurm_job_one-49284262.out` and `slurm_job_one-49284262.err`.

1.5. Write another batch script which runs your program on two nodes

The resulting files are `slurm_job_two-49289041.out` and `slurm_job_two-49289041.err`.

2. Performance characteristics [50 points]

2.1. Peak performance

This task requires the computation of the core, CPU, node and cluster peak performance for the Euler VII (phase 1 and 2) nodes. Relevant information from the provided webpages can be found summarized in table 1. Also from the exercise sheet we get the following formulas for calculating the cluster peak perfromances:

$$\begin{aligned} P_{core} &= n_{super} \cdot n_{FMA} \cdot n_{SIMD} \cdot f \\ P_{CPU} &= n_{core} \cdot P_{core} \\ P_{node} &= n_{sockets} \cdot P_{CPU} \\ P_{cluster} &= n_{node} \cdot P_{node} \end{aligned}$$

In addition I want to note a few things. The value from n_{SIMD} comes from the fact that both AMD processors support AVX2 SIMD instructions with 256-bit wide vector registers. Since we are dealing with double-precision FP numbers the size of the vector register needs to be divided by 64 bits, this will result in the n_{SIMD} value. Also from the given optimization manual for each processor unit we have to get the information if FMA (Floating Multiply Add) is provided. Since both processors support FMA, the fused multiply-add factor $n_{FMA} = 2$ otherwise it would just be 1 and the multiplication and addition can't happen in a single operation. f is the base clock frequency and key characteristic of each CPU. In addition, it should be mentioned that a good approximation for the duration of one clock cycle is given by 1/f. This number helps measuring the execution time of instructions.

From table 1 we get the following theoretical values for the Euler VII Phase 1:

$$\begin{aligned} P_{core} &= 2 \cdot 2 \cdot 4 \cdot 2.6GHz = 41.6GFlops/s \\ P_{CPU} &= 64 \cdot 41.6GFlops/s = 2662.4GFlops/s \\ P_{node} &= 2 \cdot 2662.4GFlops/s = 5324.8GFlopts/s \\ P^{(1)}_{EulerVII} &= 292 \cdot 5324.8GFlops/s = 1554.8416TFlops/s \end{aligned}$$

Parameter	Phase 1	Phase 2
n_{nodes}	292	248
$n_{sockets}$	2	2
n_{cores}	64	64
n_{super}	2	2
n_{FMA}	2	2
n_{SIMD}	4	4
f in [GHz]	2,6	2.45

Table 1: CPU Parameter

Analog for the Euler VII Phase 2 these are the results:

$$\begin{aligned} P_{core} &= 2 \cdot 2 \cdot 4 \cdot 2.45GHz = 39.2GFlops/s \\ P_{CPU} &= 64 \cdot 39.2GFlops/s = 2508.8GFlops/s \\ P_{node} &= 2 \cdot 2508.8GFlops/s = 5017.6GFlopts/s \\ P^{(2)}_{EulerVII} &= 248 \cdot 5017.6GFlops/s = 1244.3648TFlops/s \end{aligned}$$

2.2. Memory Hierarchies

The result of the command lscpu can be summarized in the following table:

Information	**Resulting Output**
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	1403.802
CPU max MHz	3800.0000
CPU min MHz	800.0000
BogoMIPS	5799.77
Virtualization	VT-x
L1d cache	32K
L1i cache	32K
L2 cache	256K
L3 cache	6144K
L4 cache	131072K
NUMA node0 CPU(s)	0-3

Table 2: Result of the command $ lscpu

The total available main memory with the command cat /proc/meminfo is:

$$MemoryTotal = 32871604kB$$

Now we want to gain information about the memory hierarchy, which can be achieved by the following command $ hwloc-ls –whole-system –no-io the result is the same as depicted in the exercise sheet. In the end I want to obtain the figure and all necessary commands are described as well in the exercise sheet. The resulting hierarchy is depicted in figure

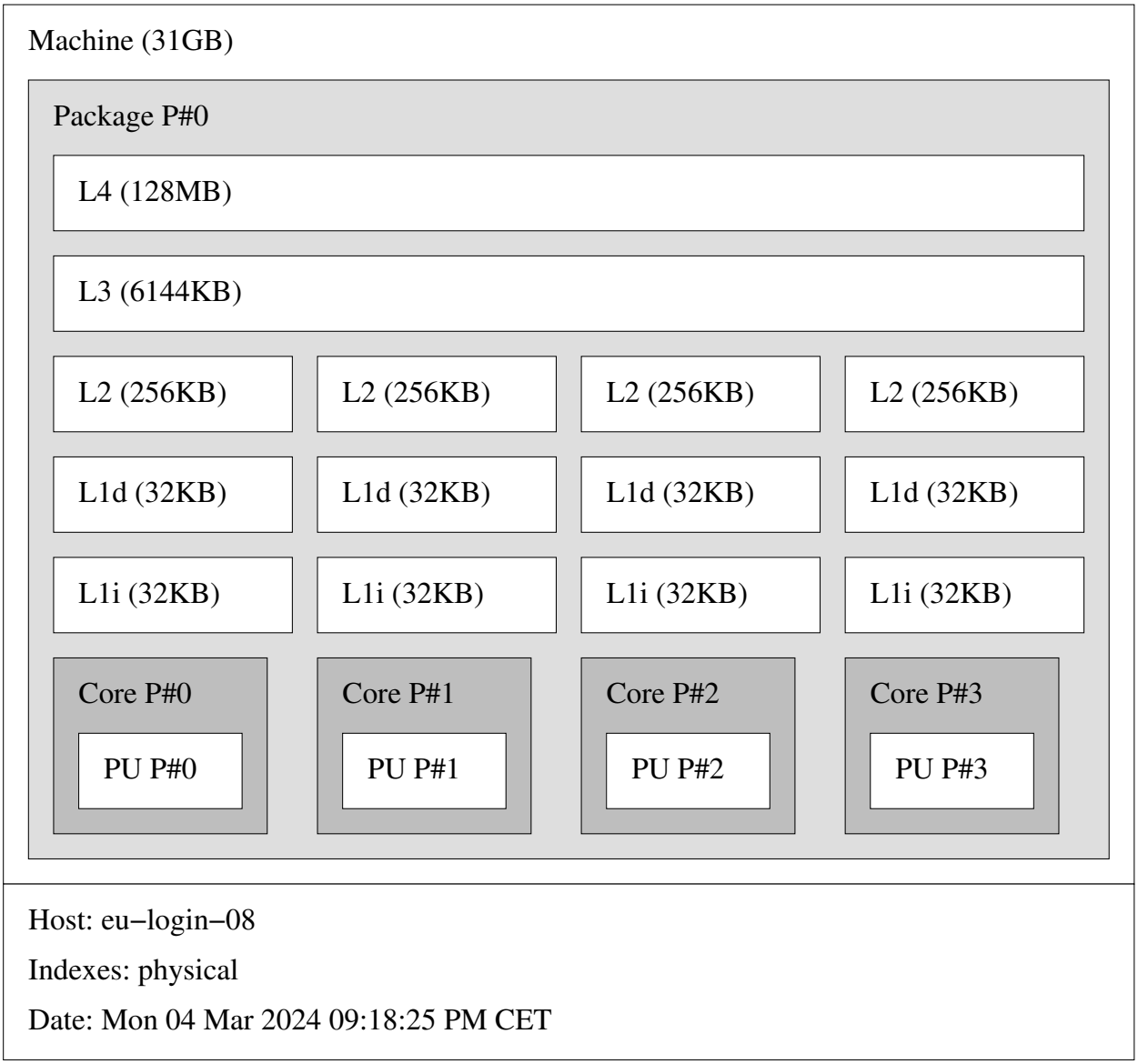

Figure 1: Schematic of an Euler login node with an Intel(R) Xeon(R) CPU E3-1248L v4 2.90GHz.

While the setup for the Intel(R) Xeon(R) CPU E3-1248L v4 2.90GHz is rather simple, lets summarize the result from Figure 2 and 3 in the following Table.

	AMD EPYC 7H12	**AMD EPYC 7763**
Main memory	31 GB	31 GB
L3 cache	16 MB	32 MB
L2 cache	512 KB	512 KB
L1 cache	32 KB	32 KB

Table 3: This table summarizes the information of one node for the underlying processor memory hierarchy.

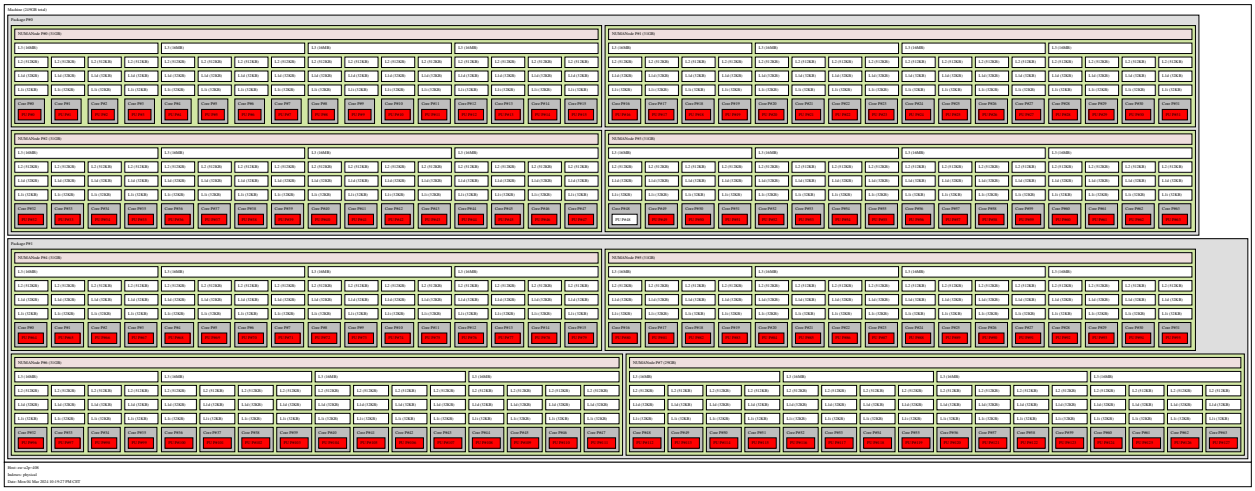

Figure 2: Schematic of an Euler login node with an AMD EPYC 7H12 CPU.

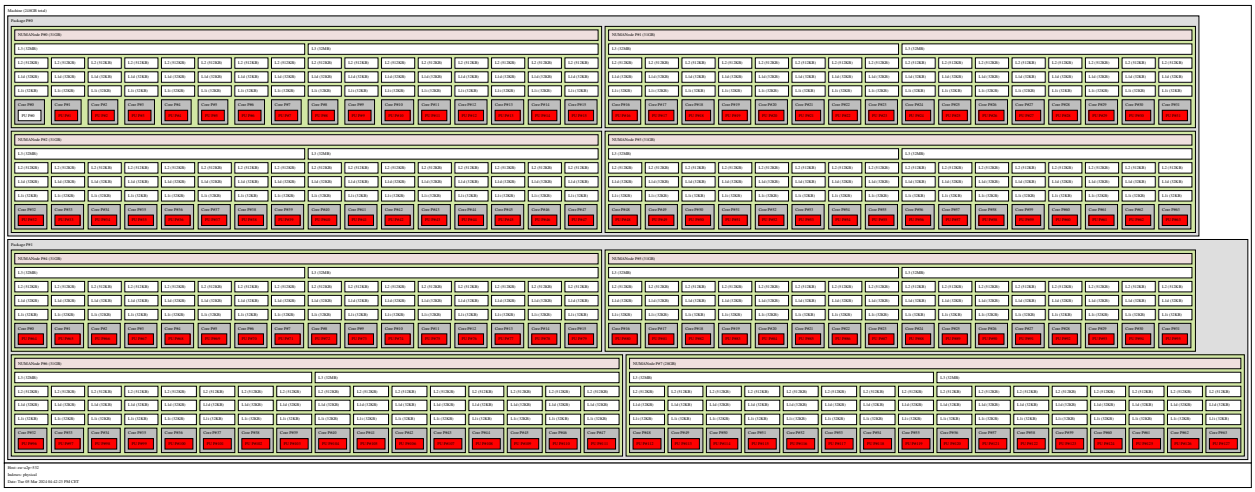

Figure 3: Schematic of an Euler login node with an AMD EPYC 7763 CPU.

2.3. Bandwidth: STREAM benchmark

For the STREAM benchmark we can use the provided file `stream.c`, where we need to adjust the STREAM_ARRAY_SIZE Parameter in the C-file for our system. In particular our two systems are, the Euler VII Phase 1 with the AMD EPYC 7H12 CPU and for Euler VII Phase 2 with the AMD EPYC 7763 CPU. According to the `stream.c` file provided by "Dr. Bandwidth", we have to calculate the STREAM_ARRAY_SIZE by considering the following two criteria.

1. The first criteria is that each array must be at least 4 times the size of the available cache memory. Note that it is important to considering from Figures 2 and 3 only one node.

2. The size should be large enough so that the 'timing calibration' output by the program is at least 20 clock-ticks. Furthermore, according to `stream.c` most versions of Windows have a 10 millisecond timer granularity. This assumption doesn't hold true for the euler cluster!

Retrieving the results from the previous task, namely Table3, we can see that we get for Euler VII Phase 1 the following results:

$$\bar{L}1 = 4 \cdot 32KB = 128KB$$
$$\bar{L}2 = 4 \cdot 512KB = 2.048MB$$
$$\bar{L}3 = 4 \cdot 16MB = 64MB.$$

While we would get for Euler VII Phase 2 these results:

$$\bar{L}1 = 4 \cdot 32KB = 128KB$$
$$\bar{L}2 = 4 \cdot 512KB = 2.048MB$$
$$\bar{L}3 = 4 \cdot 32MB = 128MB.$$

By respecting the second criteria the chosen parameter values can be extracted from Figure 4 and 5.

```
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 8500000 elements, Offset = 0 elements
Memory per array = 64.8 MiB = 0.1 GiB.
Total memory required = 194.5 MiB = 0.2 GiB.
Each kernel will be executed 20 times.
 The  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 5301 microseconds.
   = 5301 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:           26351.2     0.005233     0.005161     0.005424
Scale:          17845.9     0.007694     0.007621     0.007945
Add:            19750.2     0.010414     0.010329     0.010478
Triad:          20116.1     0.010250     0.010141     0.010537
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

Figure 4: STREAM benchmark result for Euler VII Phase 1

As done in the exercise sheet for a rough estimate we can assume for the Euler VII Phase 1 a maximum bandwidth $b_{STREAM} = 17$ GB/s, which corresponds to the scale function. Same thing is possible where the Euler VII Phase 2, where we can retrieve from Figure 5 a maximum bandwidth value of $b_{STREAM} = 27$ GB/s, which corresponds to the add function.

```
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 17000000 elements, Offset = 0 elements
Memory per array = 129.7 MiB = 0.1 GiB.
Total memory required = 389.1 MiB = 0.4 GiB.
Each kernel will be executed 20 times.
 The  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 5962 microseconds.
   = 5962 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:           38444.8     0.007299     0.007075     0.007544
Scale:          28665.3     0.009878     0.009489     0.010262
Add:            27657.4     0.015129     0.014752     0.015473
Triad:          28163.1     0.015124     0.014487     0.015558
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

Figure 5: STREAM benchmark result for Euler VII Phase 2

2.4. Performance model: A simple roofline model

As a result, the last question is to answer, at what operational intensity is a kernel or application memory/compute-bound? To answer this question the provided paper by Williams et al. demonstrates the roofline model, which can be described as a visual performance model, that can help in order to optimize code and software for floating point computations. Furthermore, the proposed model combines three important computer metrics which were already computed in the prior tasks. Parameters included are, the floating point performance, the operational intensity and the memory performance. The resulting 2D graph then consists of the following two parts:

1. Horizontal line: represents the peak floating point performance of the core.
2. Linear line: represents the peak memory bandwidth.

Moreover, the intersection between the two lines results in a point, which describes the limits for the underlying system. As a result to determine at what operational intensity a kernel or application is memory-bound or compute-bound, we have to look at where the performance of the kernel intersects (the described critical point) with the memory bandwidth line on the graph. Hence, on one hand, if the kernel's performance is below this line, it's liekly memory-bound. On the other hand, if it's above the line it's compute-bound.

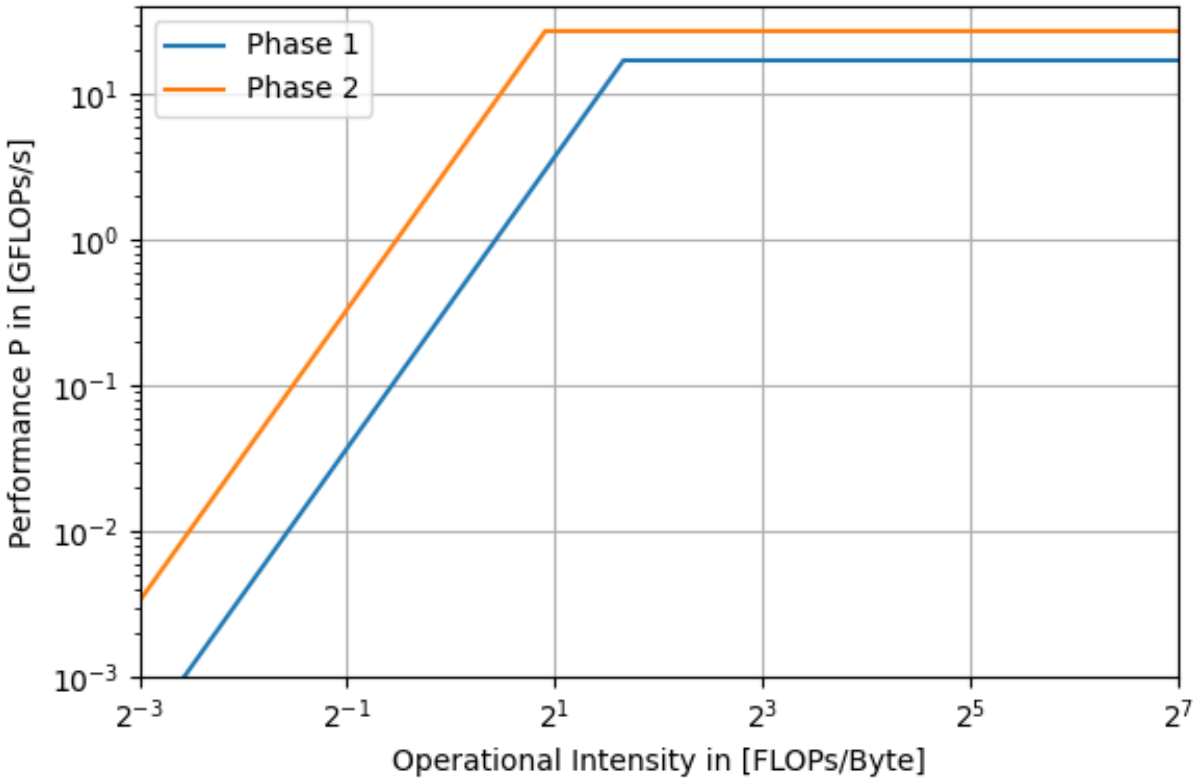

Figure 6: Simple roofline model for a single Euler VII Phase 1 and Phase 2 core.

As a result if we look at Figure 6, it is clear that the kernel or application is memory-bound at operational intensity I values below 3.2 and 1.9 for the Euler VII Phase 1 and 2 core respectively. If values are above this critical point the kernel and application is compute-bound.

Note that on the next page the solution for the Project 1b starts.

3. **Auto-vectorization [10 points]**

4. **Matrix multiplication optimization [30 points]**