ETH zürich

High-Performance Computing Lab for CSE **2023**

Student: Simon Aaron Menzi Discussed with: Lukas Martin Bühler

Solution for Project 1

Due date: 06 March 2023, 23:59

HPC Lab for CSE 2023 — 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.

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

1. Euler warm-up [10 points]

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

From Scicomp.ethz.ch: Modules configure your current computing environment (PATH, LD_LIBRARY_PATH, MANPATH, etc.) to make sure all required binaries and libraries are found.

1.2. What are the following terms: LSF, SLURM, and LoadLeveler? What do they have in common?

- **LSF** *LoadingSharingFacility* is a job scheduler used before on the Euler cluster.
- **SLURM** *SimpleLinuxUtilityForResourceManagement* is the job scheduler currently used on the Euler cluster.
- **LoadLeveler** is another job-scheduler developed by IBM.[1]

1.3. What software is installed on the Euler Cluster to manage workload and resources of multiple users?

Nowadays, SLURM is used to manage the workload of multiple users. Before that, LSF was used.

[1] for more information, see https://www.ibm.com/support/pages/tws-loadleveler-linux-multiplatform-v41-and-linux-power-v41

1.4. How can you execute programs on a cluster's compute node? Describe the interactive and batch modes.

Interactive modes do not produce output files but write the output directly into the terminal. This process can be started with the following command:

```
$ srun --pty bash
```

Jobs can be submitted with the following bash command

```
$ sbatch --wrap=command
```

More detailed information can be passed to the job scheduler.[2]

1.5. Using the batch mode, write a simple "Hello World" program which prints the host-name of the current machine.

For a single node, this command can be used:

```
$ echo "Hello World, the hostname is " && hostname
```

But as soon as multiple nodes are involved, one should use something like the following c++-program:

```
#include <iostream>
#include <unistd.h> // defines HOST_NAME_MAX
#include <limits.h> // defines gethostname

int main() {
    char hostname[HOST_NAME_MAX];
    gethostname(hostname, sizeof(hostname));

    std::cout << "Hello World from " << hostname << std::endl;

    return 0;
}
```

This code can be run with the following bash script:

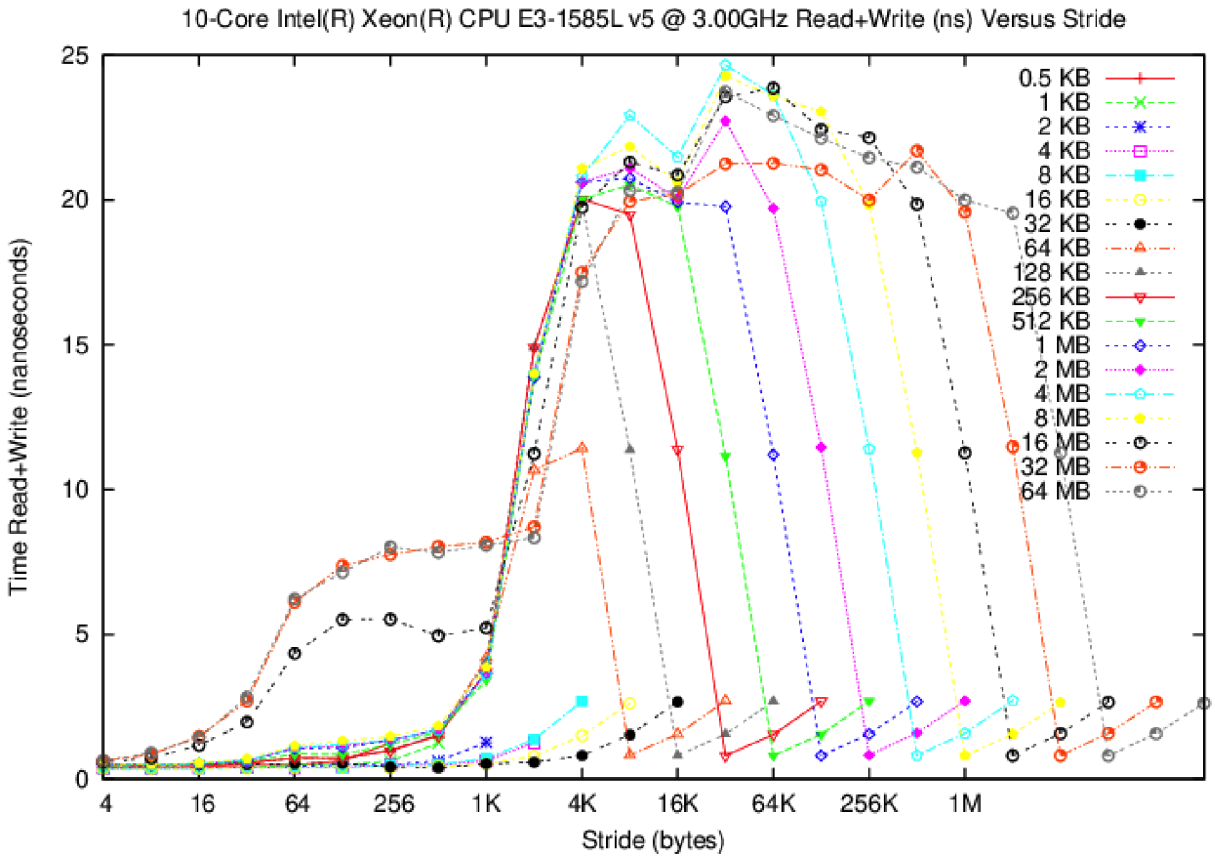
```
$ #!/bin/bash -l

$ #SBATCH --time=00:05:00
$ #SBATCH --ntasks 2
$ #SBATCH --nodes 2


$ srun ./a.out
```

2. Explaining the impact of memory hierarchies [10 points]

3. Auto-vectorisation [10 points]

The following plot results from using the given script.

[2] For more information, consult https://scicomp.ethz.ch/wiki/LSF$_{t}o_{S}lurm_{q}uick_{r}eference$



What can be observed is that for small enough csize, one will only access the second cache layer. If they are bigger, the 3rd cache layer will be used.

3.1. Why is it important for data structures to aligned?

If memory is aligned, more data can be fetched with a single SSE instruction, resulting in less time spent getting the data from main memory.

3.2. What are some obstacles that can prevent vectorization?

Non-contiguous memory access: If memory is accessed contiguously, you can get more data with a single SSE instruction, thus saving time. Most common case is non-unit memory strides, because those can rarely be vectorized by the compiler due to non-contiguous memory access. - Data dependencies: If the order seems to matter, the compiler won't vectorize, as this would reorder and therefore change the output.

3.3. Is there a way to assist the compiler through language extensions? If yes, plese give some examples. If not, explain why not.

Yes, there are ways to assist the compiler:

3.3.1. Pragmas

- #pragma ivdep: tells the compiler that it has not to worry about data dependencies.
- #pragma loop count (n): gives the compiler the trip count, with which he can decide if vectorization is worthwhile.
- #pragma vector always: asks the compiler to always vectorize the following loop.
- #pragma loop align: Tells the compiler that the data of the following loop is aligned (16 Byte for Intel SSE instruciton)

- #pragma novector: tells the compiler to not vectorize the following loop.
- #pragma vector nontemporal: Tells the compiler that the data will not be reused, therefore cache bypassing is possible.

3.3.2. Keywords

- restrict: needs compiler flags [Q]restrict or [Q]std=c99 to be used. It asserts that the word referenced by a pointer is not aliased.

3.3.3. Options/Switches

- Disambiguation of pointers: Telling the compiler that the same memory location is not accessed via different arrays or pointers
- Interprocedural Optimization (IPO)
- High-level optimization (HPO)

3.3.4. Which loop optimizations are performed by the compiler in order to vectorise and pipeline loops?

- Reordering
- Using accessible registers to perform multiple operations

4. The perfect DGEMM micro-kernel [30 points]

There are several things to consider when optimizing for a specific micro architecture. In this exercise, the AMD Epyc 7742 was looked at with the following specifications:

- 32 KiB L1d 32 KiB L1I cache per core
- 512 KiB L2 cache per core
- 256 MiB L3 cache, shared
- AVX2 and SSE instructions up to 4.2 are supported

Due to this knowledge, in the implementation, intrinsic AVX2 instructions were used. This did not result in a significant speedup but in a more convoluted code. No uncluttering measurements were taken due to time constraints. One could also unroll the loops even more, but the crux of the problem doesn't lie in the vectorization by hand in this problem. Memory layout could be managed more efficiently instead, which would probably result in better speed up than vectorization by hand. For further projects, this knowledge will be considered.

5. Optimize General Square Matrix-Matrix Multiplication [40 points]

The first implementations for this task were way to complex and gave wrong answers, so a more simplified version was put together. A simple triple for-loop is still being used but for smaller chunks of the matrices. Also no preloading was done.

Due to structure of the program, the simple approach to parallelize the most inner loop seems to be sufficient as all the computation is done in there and no critical reading or writing takes place.