Homework Assignment 1

General information

Ugent HPC infrastructure

All source files required for this assignment can be found in the documents section on Minerva. All examples should be compiled and run on the 'Delcatty' cluster (part of the UGent HPC). You will need an account for this cluster, which can be requested by going to the following URL and following the instructions:

http://hpc.ugent.be -> "Toegang en beleid" (NL) / "Access policy"

Note that it might take one or two days before your request is approved.

After your request is approved, read the section on "Information for new users" on the wiki page. This explains how to connect to the HPC infrastructure and how to copy files between your system and the UGent HPC. A VPN connection to UGent is required when attempting to connect from outside the UGent network.

Once you connect to the HPC infrastructure, you will be logged onto one of the interface nodes. These nodes can be used to compile software and submit jobs to the different clusters (default = Delcatty), but running software on these interface nodes is generally considered bad practice. Therefore, we ask that you compile and run the benchmark examples below on one of the worker nodes. In order to get access to one of the worker nodes type:

```
qsub -I
```

If at least one core on a worker node is available, this command will open a new shell ('interactive session') on one of the worker nodes (nodeXXXX). In order to get access to a complete machine (16 cores), issue:

```
qsub -I -l nodes=1:ppn=16
```

This will give you exclusive access to one of the nodes which has the advantage that your timings will not be influenced by software from other users. Depending on the occupation of the cluster by other users, it might take some time before the scheduler is able to grant access. You can check the cluster usage this by issuing:

```
pbsmon
```

When you are finished, terminate your session by typing:

```
exit
```

Report

This assignment should be completed **individually**. While it is OK to discuss the solution with fellow students, it is strictly prohibited to exchange and/or copy solutions.

You should hand in your report in **.pdf format**. Your report should contain **(a) your name** and **(b) the answers to the questions using the format listed below**, i.e.

Q1.1: answer to this question

Q2.1: answer to this question

…

Additionally, hand in the modified **matmat.cpp** file. **Write your name in this source file as well (in comment).**

The maximum size for the report is **one A4 page, 10pt**: “In der Beschränkung zeigt sich erst der Meister”.

Your report should be named “assignment1_firstname_lastname.pdf”, the modified matmat.cpp filename should be renamed to “matmat_firstname_lastname.cpp”. Use the Minerva dropbox to submit your solutions. Do not zip or otherwise pack these two files!

The deadline is Friday, October 10th 12:00 (noon), CET. This deadline is firm and non-negotiable.

1 Influence of pipelining on software performance

Download the C++ source file `pipeline.cpp` from Minerva. Study the source code. Compile the code using the GCC compiler:

```
g++ -O2 pipeline.cpp -o pipeline
```

Note that we do not use -O3 because we do not want to enable SIMD instructions at this moment. Run the code. Each line of output shows the execution time for varying size of the inner loop size N. Note that the outer loop size (nIter) is taken such that the total number of floating point operations is constant, regardless of the choice of N (like in the vector triad example).

- Modify the program to also show the FLOPS/s, next to the execution time. Note that the inner loop in scaleVector starts at index = 6.
- Change the inner loop (vectorScale function) from `a[i] = s*a[i]` to `a[i] = s*a[i-1]` and benchmark the code again. Repeat the same experiment for the following offsets:

```
a[i] = s*a[i-2];
a[i] = s*a[i-3];
a[i] = s*a[i-4];
```

```
a[i] = s*a[i-5];
a[i] = s*a[i-6];
```

- **Q1.1: report the FLOPS/s (for e.g. N = 1018) for each of loop kernels and clearly explain the observed performance behavior.**

2 Influence of the cache on software performance

Download the C++ source file vectortriadICC.cpp from Minerva. Study the source code. Compile the code using the Intel compiler as follows (on Delcatty):

```
module load ictce/5.5.0
icc -O3 -xAVX vectortriadICC.cpp -o vectortriad
```

The first line will enable the use of a number of Intel tools, including the Intel compiler (icc) and the Intel MKL (Math Kernel Library).

- Run the code on a Delcatty worker node. The CPUs are octa-core Intel Xeon E5-2670 with 32 KByte L1 cache (data), 256 KByte L2 cache per core (unified) and 20 MByte L3 cache (unified, shared among 8 cores). **(Q2.1) Explain the benchmark results in function of the cache sizes of the Xeon E5-2670.**
- Generate assembly code for this program

  ```
  icc -O3 -xAVX vectortriadICC.cpp -S
  ```

  Output will be written to vectortriadICC.s. To find the code of the inner loop, look for a line that starts with `_Z11vectorTriad`. This line denotes the entry point of the vectorTriad function. Locate the inner loop code. Normally, this code should be properly vectorized using aligned loads and stores.
- Recall that the `__restrict` keyword hints the compiler that the memory regions pointed to by a, b, c, and d are guaranteed not to overlap. Remove (or comment out) the `__restrict` keywords in the source file. **(Q2.2) Do you think the compiler can still generate SIMD instructions in case the memory regions overlap? Explain why (not).**
- Regenerate the assembly code (without the restrict keyword) and again look at the source code. Note that the code became much larger. **(Q2.3) Did the compiler use SIMD instructions? (yes or no).**
- Run the vector triad again. Note that a, b, c and d still do not overlap in practice, the only difference is that the compiler cannot make this a priori assumption at compile time**. (Q2.4) Is there a significant difference in FLOPS/s (yes or no)**, compared to the use of the restrict keyword?
- Now, still without the restrict keyword, explicitly overlap a and b by replacing

  ```
  posix_memalign(((void**)(&b)), 32, ...);
  ```

by

```
b = a;
```

Also remove the line:

```
free(b);
```

to avoid freeing the same memory region twice. Compile and the run this code. **(Q2.5) Is there a big difference in runtime compared to the case where a, b, c and d did not overlap? (yes or no).**

- **(Q2.6) Can you now try to explain how the compiler handled the absence of the `__restrict` keyword?**

3 Influence of memory access patterns on software performance

Download the C++ source file matvec.cpp and the CMakeList.txt file and put them in the same directory. The matvec code provides several implementations of a matrix-vector multiplication that is stored in column-major format:

1. a strided-memory access implementation (row by row traversal of the matrix)
2. a linear memory access implementation (column by column traversal of the matrix)
3. a BLAS-based implementation

Linking with the BLAS library is more involved. We use the MKL implementation of the BLAS, since it is optimized for Intel processors. To make life easier, CMake can automatically detect BLAS and create a Makefile:

```
module load ictce/5.5.0                (this will load several Intel tools, including MKL)
module load CMake/2.8.12-ictce-5.5.0      (this loads the latest version of CMake)
module load Valgrind                              (this load the Valgrind software)
cmake .
make
```

Note that again no SIMD instructions are allowed ("-O2" optimization level; see "CMakeLists.txt" file), since we want to isolate the influence of memory access patterns on the software performance.

- Run the program and compare the FLOPS/s for each of the different implementations. **(Q3.1) What is the measured FLOPS/s for the 1000x1000 matrix for each of the implementations?**

- Change the matrix dimensions to 256x256. Comment out the calls to matvecLinear and matvecBLAS and recompile. Your program will now only perform the strided-memory access matrix-vector multiplication. **(Q3.2) How many FLOPS/s do you obtain?**
- Run this program in valgrind (a CPU emulator that can –among many other things- keep track of cache hits/misses):

  ```
  valgrind --tool=cachegrind ./matvec
  ```

  After 10 to 20 seconds, press <ctrl+C> to stop valgrind from executing the complete algorithm (might take a lot of time) **(Q3.3) What is the percentage of L1 data read misses (D1 miss rate) + explain this percentage.**
- Repeat the same experiment for a 255X255 matrix. **(Q3.4) Give the D1 miss rate + explain this percentage.**
- Repeat the same experiment for a 64x64 matrix. **(Q3.5) Give the D1 miss rate + explain this percentage.**

4 Influence of cache reuse on software performance

Download the C++ source file matmat.cpp and the CMakeList.txt file and put them in the same directory. Study the source code. Compile using:

```
module load ictce/5.5.0                (this will load several Intel tools, including MKL)
module load CMake/2.8.12-ictce-5.5.0     (this loads the latest version of CMake)
module load Valgrind                             (this load the Valgrind software)
cmake .
make
```

The source code only contains code for a naive matrix-matrix implementation and the BLAS implementation.

- Run the program (this will take a minute or so). This should give some insights in the power of the BLAS.
- **Implement a matrix-matrix multiplication by interchanging the loops such that the memory access is linear** (similar to what was done for a matrix-vector multiplication). An empty function prototype is already provided for this.
- Implement, from this linear access algorithm, a blocked algorithm. Use a block size 64. **Hand in the modified source code.** Note that you will probably not beat BLAS, but you should beat the linear access algorithm.
- **(Q4.1) Report the FLOPS/s for all algorithms**
- Use Valgrind to take a look at the cache miss ratios for all algorithms individually (simply comment 3/4 algorithms, compile, and run the software in Valgrind. After 10-20 seconds or so,

press <ctrl+C> to stop Valgrind from executing the complete algorithm (might take several hours). **(Q4.2) Report and explain the results.**

5 Statistics

(Q5.1) How much time did you spend on this assignment?