Practical Work - Project : Optimization and Profiling

Master CSMI
Compilation & Performance
Bérenger Bramas

December 16, 2020

1 Summary

In this project, you will optimize a kernel using OpenMP and measure the benefit using Perf.

2 Warning

- You have until the 13th of January 2021 at 13:00to complete the test, and you must push all your code and your report before this deadline.
- You must write a report of 2 or 3 pages (see the related section to know more) and not a *report.md* file.
- You can ask me questions by email at any time!
- Coding style/quality will be evaluated too.
- Do not pin the process with *taskset*, as we are going to parallelize the code you might have troubles. Instead use *OMP_PROC_BIND*.

3 Ressources

- Perf example: `http://www.brendangregg.com/perf.html`
- Another Perf wiki: `https://perf.wiki.kernel.org/index.php/Main_Page`
- The command *perf list* provides all the possible event types
- OpenMP 4.0 API C/C++ Syntax Quick Reference Card: `https://www.openmp.org/wp-content/uploads/OpenMP-4.0-C.pdf`

4 Practical work organization (almost the same)

In the practical work, you will obtain the code from my repository and push it to your repository. Therefore, you will have to clone one branch per session and push it to your own repository.

In the rest of the document, we consider you have a repository named *cnp-tp-2020* on *git.unistra.fr* that is private but that I can access in read.

4.1 Get the practical work

Consider you are in your project directory do the following:

```
# Clone my repo
git clone https://git.unistra.fr/bbramas/csmi-tp-2020.git --branch=TP9 csmi-tp9
# If you use SSH replace [USER] and use:
```

```
# git clone git@git.unistra.fr:[USER]/csmi-tp-2020.git --branch=TP9 csmi-tp9
# Go in the newly created directory
cd csmi-tp9
```

4.2 Add your repository as remote

You will push on your own repository:

```
# Rename my remote
git remote rename origin old-origin
# Add your own remote
git remote add origin https://git.unistra.fr/[YOU LOGIN HERE]/cnp-tp-2020.git
# If you use an SSH key:
# git remote add origin git@git.unistra.fr:[YOU LOGIN HERE]/cnp-tp-2020.git
# Push the current branch and active the tracking
git push -u origin TP9
```

4.3 During the session and while you work on the project

After each question or important modification push the current changes:

```
# No matter where you are in the project directory
git commit -a -m "I did something"
git push
```

4.4 When you are done

You have fully finished your work (again the deadline is 13th of January 2021 at 13:00):

```
git commit -a -m "I did something"
git push
```

4.5 Compilation

To compile, we use CMake:

```
cd Code
mkdir build
cd build
cmake ..
make # Will make all
make something # Will build only something
VERBOSE=1 make # Will show the commands used to compile (including the flags)
```

Make sure you compile in release! This can be done using *ccmake* **or** *cmake.. – DCMAKE_BUILD_TYPE = Release*

5 Reminder

Profiling a code is mandatory before performing some optimizations. Looking at the execution time is a good measure, however it does not really help to know what should be optimize (instructions, memory, etc...) and timers that cover large scope of the code will not help to know the instructions that are not used correctly. In this context, several performance tools exist to help the programmer. These tools are able to access some counters in the CPU (consider them as registers). These counters record events such as cache misses (when a data needed by the CPU is not in the cache) or many other types of events.

6 Get an overview

On my pc I have:

```
$ OMP_PROC_BIND=TRUE ./matrix -check
>> Without Optim : 28.4301
>> With Optim : 0.0887636
```

7 Measure the performance of the unoptimize kernel

7.1 Get an overview

Compile the matrix.cpp file (make sure CMake CMAKE_BUILD_TYPE is setup to RelWithDebInfo, $cmake.. -DCMAKE\_BUILD\_TYPE = RelWithDebInfo$, mode to capture the debugging information that allows to link the binary to the source code). Then run it and look at the different counters that Perf measures. As you will see instruction per cycle is pretty low, and the number of cache misses is high (this could mean that there is space for optimization).

The original code (on a different computer):

```
$ perf stat -B -e cache-references,cache-misses,cycles,instructions,branches,\
faults,migrations,L1-dcache-load-misses,LLC-load-misses,LLC-store-misses \
./matrix -no-optim
>> Without Optim : 8.30243

 Performance counter stats for './matrix -no-optim':

     2 156 098 401      cache-references
        25 678 142      cache-misses              #    1,191 % of all cache refs
    21 720 440 900      cycles
     8 888 816 705      instructions              #    0,41  insn per cycle
     1 111 715 812      branches
             6 629      faults
                 3      migrations
     3 311 488 365      L1-dcache-load-misses
        25 201 230      LLC-load-misses
           431 115      LLC-store-misses

       8,549233264 seconds time elapsed
```

My optimized version:

```
$ perf stat -B -e cache-references,cache-misses,cycles,instructions,branches,\
faults,migrations,L1-dcache-load-misses,LLC-load-misses,LLC-store-misses ./matrix -no-ch
>> With Optim : 1.59968

 Performance counter stats for './matrix -no-check':

       137 599 498      cache-references
           633 985      cache-misses              #    0,461 % of all cache refs
     4 320 670 959      cycles
     5 639 639 250      instructions              #    1,31  insn per cycle
       569 871 753      branches
             2 478      faults
                 0      migrations
       137 468 160      L1-dcache-load-misses
            16 179      LLC-load-misses
           402 373      LLC-store-misses

       1,668058527 seconds time elapsed
```

$cache - misses$ is reduced. *instructions* is slightly reduced. But remember that it also includes all the matrix initialization stages.

7.2 Record in details

It is also possible to record the information related to the execution and to generate reports to see the result per function (or even per instruction).

```
$ perf record -e cache-misses ./matrix -no-optim
>> Without Optim : 8.34207
[ perf record: Woken up 2 times to write data ]
[ perf record: Captured and wrote 0.335 MB perf.data (8298 samples) ]
```

Then

```
# To get the symbol
perf report --stdio
# To get the details
perf annotate
PercentI
       I     main():
       I       mov    %r10,%rsi
       I             for(long int k = 0 ; k < N ; ++k){
       I                 for(long int j = 0 ; j < N ; ++j){
       I                     for(long int i = 0 ; i < N ; ++i){
       I                         C[i*N+j] += A[i*N+k] * B[j*N+k];
       I210:   movsd  (%rsi),%xmm1
  0,06 I       lea    -0x800000(%rcx),%rax
       I       mov    %r8,%rdx
       I       xchg   %ax,%ax
  0,85 I220:   movsd  (%rdx),%xmm0
  9,59 I       add    $0x2000,%rax
  0,76 I       add    $0x2000,%rdx
  5,01 I       mulsd  %xmm1,%xmm0
  3,00 I       addsd  -0x2000(%rax),%xmm0
 75,89 I       movsd  %xmm0,-0x2000(%rax)
       I                     for(long int i = 0 ; i < N ; ++i){
  3,84 I       cmp    %rax,%rcx
       I     ↑ jne    220
       I       add    $0x8,%rcx
       I       add    $0x2000,%rsi
       I                for(long int j = 0 ; j < N ; ++j){
       I       cmp    %rcx,%r9
       I     ↑ jne    210
       I       add    $0x8,%r8
       I       add    $0x8,%r10
       I             for(long int k = 0 ; k < N ; ++k){
       I       cmp    %r8,%r11
       I     ↑ jne    20a
```

So no surprise, all the cache misses are in our matrix product.

8 Improve the kernel

8.1 Improve the memory access

From the profiling the cache misses are significant and this means that the memory is not accessed correctly. Look at the access pattern and reorder the loops to improve the access pattern. See the improvement in the execution time.

8.2 Parallelize the code

Add the OpenMP parallel pragma to parallelize the kernel. As the loops are regular, do it on a single loop only.

8.3 Vectorize

Use the OpenMP SIMD pragma to vectorize the inner loop. Do not forget reduction, safelen, simdlen and aligned keywords. You might need to use pointers over the matrices to make OpenMP life easier (that is, instead of using directly $\&array[idx]$ in the pragma, first declare a pointer $ptr = \&array[idx]$ and then use it in the pragma).

8.4 Anything else if you want

Feel free to update the code as you like to apply other optimizations, or to show pathological configurations, or to compute something else if you want. This is not needed, but if you have some ideas or something you did test on your side, then do not hesitate to include it in the report.

For example, one could imagine to compare *float* vs. *double*, to do blocking, to evaluate the performance with other counters (cache misses, etc.), compare with Blas, etc...

8.5 Report

This should be a 2 or 3 pages report:

- You must use the *report/report.tex* file and the *report/report.bib* file for the bibliography (if needed).
- You must write an abstract and a conclusion, the rest of the document is free. It is up to you to decide if you want to describe step after step what you did and what are the improvements, or if you prefer to have a section implementation and a section performance study.
- You should show me results and how things were improved based on your modifications. It could be with tables, with plot figures or both.
- To measure the improvement, you could use some of the counters given by Perf or you could use execution time. You could also change the size of the matrix or the number of threads (imagine a plot where y is execution time and x is the number of threads, and where you will put three lines for small/medium/large matrices... that would be nice, isn't it?).
- Please let me know when you pin the threads or apply any specific flag (and pinning is advised). Also, for each result, tell me if this is the average of several execution time, etc.
- You must push your generated pdf in addition to the updated latex.