Aalto University

Aalto University
School of Science

Programming Parallel Computers

CS-E4580

P: Project

José González López: 893699

May 26, 2021

Contents

1 Introduction **2**

2 Files **2**

3 Section A **2**

4 Section B **3**

5 Section C **6**

6 Conclusions **6**

1 Introduction

This is my small research project done via remote access (with SSH) to the Aalto computers provided in Maari-A classroom. The chosen computer is the one with the name: Dodo.

2 Files

- benchmark.cc: Little program for testing
- benchmark.s: Assembly code of benchmark.cc
- cp_benchmark.cc: My cp3b version benchmarked the same as benchmark.cc
- report.pdf: This report

3 Section A

By typing on the command line

$ *cat /proc/cpuinfo*

We can access most of the CPU information such as:

- Vendor: GenuineIntel
- Name: Intel(R) Xeon(R) CPU E3-1230 v5 @ 3.40GHz
- Processors: 0-7
- Cache size: 8192 KB
- Bomips (which is supposed to be number of million of instrucctions when doing nothing): 6799.81

We can even type

$ *lscpu*

to get more information

- Thread(s) per core: 2
- Socket(s): 1

- CPU MHz: 833.056
- CPU max MHz: 3800.0000 (I may suppose this is theoretical maximum of clock frecuency)
- CPU min MHz: 800.0000
- L1d cache: 128 KiB
- L1i cache: 128 KiB
- L2 cache: 1 MiB
- L3 cache: 8 MiB

In order to get an idea of how many FLOPS this computer can do I used the CTP information provided by Intel itself to do the theoretical calculations and the following data and formula [1][2][3]:

- 128.000 million instructions per second
- Theoretical 3.8 Ghz
- 1 socket (one cpu per node)

Node performance in GFLOPS = (CPU speed in GHz) x (number of CPU cores) x (CPU instruction per cycle) x (number of CPUs per node)

$$\text{GFLOPS} : 3.8 * 8 * \frac{128 * 10^9}{3.8 * 10^9} * 1 = 30.4 * 32 = 972.8$$

So, in theory, we can execute around 973 Billion operations per second if I am not mistaken with these calculations.

Note: I simplified the division so it is power of 2 friendly because that result is instructions/cycle, which I think it should be a power of 2.

4 Section B

I made a very simple program:

```
int main(){
        float a = 1.1
        float b = 1.2
        float result=0.0
        for(int i=0;i<1000000000;++i){
                result=result+(a-b)*(a+b)+(a+b)/(a-b);
        }
}
```

In which we are making per iteration:

- 4 Sums
- 2 Subtractions
- 1 Multiplication
- 1 Division

and this is repeated 1 billion times. So we are having around 8 billion arithmetic operations in total. Then I compile the program [4][5]

```
$ gcc benchmark.cc -lstdc++ -S
```

And have the following results when executing:

- Number of clock cycles: 11825434760
- The program took 3.469921 seconds of wall time
- The program took 3.469806 seconds of cpu time

So, to get the number of FLOPS per second we do the following calculations

$$\frac{8 * 10^9}{3.469921} = 2,305 \text{ GFLOPS per second}$$

This number is very low compared to the one got on section A. I believe that, as it was discussed in IS, not all arithmetic operations performed in this program are equal (time-wise). Making divisions is far more expensive than doing additions, subtractions and multiplications. I also checked the assembly code produced by the compiler.

```
        ############LOOP############
# 0 "" 2
#NO_APP
        movl    $0, -156(%rbp)
.L5:
        cmpl    $999999999, -156(%rbp)
        jg      .L4
        movss   -152(%rbp), %xmm0
        movaps  %xmm0, %xmm1
        subss   -148(%rbp), %xmm1
        movss   -152(%rbp), %xmm0
        addss   -148(%rbp), %xmm0
        mulss   %xmm1, %xmm0
        movaps  %xmm0, %xmm2
        addss   -160(%rbp), %xmm2
        movss   -152(%rbp), %xmm0
        addss   -148(%rbp), %xmm0
        movss   -152(%rbp), %xmm1
        subss   -148(%rbp), %xmm1
        divss   %xmm1, %xmm0
        addss   %xmm2, %xmm0
        movss   %xmm0, -160(%rbp)
        addl    $1, -156(%rbp)
        jmp     .L5
```

And we have

- 4 ADDSS instructions
- 2 SUBSS instructions
- 1 MULSS instruction
- 1 DIVSS instruction

Which correspond to the arithmetic operations detailed earlier. So, my guess is that we could not perform all the theoretical FLOPS because some operations were more time-consuming than others. I have checked that this processor is from the skylake production, so thanks to the pdf shared on Zulip we can actually confirm this theory [6].

- Throughput ADDSS: 0.5

- Throughput SUBSS: 0.5
- Throughput MULSS: 0.5
- Throughput DIVSS: 3

Meaning that the division is 6 times more expensive than the others.

5 Section C

To avoid confusion in how the course files are computed I just created a copy of the original CP3b file I had and compile it using the grading tool while putting on the standard output the same information I calculated on the other file. And I will take the times when running the test without the address sanitize. The chosen one was benchmark2a.

- Number of clock cycles: 333261292
- The program took 0.097807 seconds of wall time
- The program took 0.744079 seconds of cpu time

According to the course material in benchmark2a there are 16.016.000.000 operations.

$$\frac{16.016.000.000}{0.097807} = 163.75\text{GFLOPS per second}$$

Which is far more impressive than the other case, still, we are quite far from the theoretical maximum (if my calculus is not wrong up to this point). This makes sense in the sense that in this program we have not made use of any divisions.

6 Conclusions

This little project work has shown me some interesting facts that we have been using in the course and up to this point those concepts sounded like a lot of jargon to me. However, now I understand the meaning of many of the things that the grader tool provides me when I am submitting my code and I even revisited some assembly code which in my opinion it is quite important to at least understand a little bit. In my opinion, I would suggest to keep this project as something optional at the end of the course because it is a very nice way to wrap up the contents and also quite relaxing after all the *difficult* assignments.

References

[1] CTP metrics for all Intel Processors. Available. Online: https://www.intel.com/content/dam/support/us/en/documents/processors/CTP-Metrics-for-all-Intel-Processors.pdf

[2] How to calculate peak theoretical performance Available. Online: http://www.novatte.com/our-blog/197-how-to-calculate-peak-theoretical-performance-of-a-cpu-based-hpc-system

[3] Benchmarking. Available. Online: https://cs.stackexchange.com/questions/9144/calculating-floating-point-operations-per-secondflops-and-integer-operations-p

[4] How to get clock cycles on C. Available. Online: https://helloacm.com/the-rdtsc-performance-timer-written-in-c/

[5] How to measure execution time in C. Available. Online: https://levelup.gitconnected.com/8-ways-to-measure-execution-time-in-c-c-48634458d0f9

[6] Optimize instruction table. Available. Online: https://www.agner.org/optimize/instruction_tables.pdf