Computer Architecture Project Report

Lorenzo Catoni Leonardo Giovannoni Marco Lucio Mangiacapre

Contents

1 Introduction

The code for the whole project can be browsed here `https://github.com/PuffoTrillionarioGonePublic/CA-Project`. Here, we will reports the description of our project with the main results obtained during our study.

1.1 The Problem

Given a set of many metrics measured together (ex.: CPU usage, memory usage, ...) on the same entity (ex.: a server), determine if they are correlated in any way. Results can than be used to reduce the number of metrics measured in future samples.
In input we get a single csv file with an unknown number of rows and columns, the input file may be too large to fit in memory.
As output we will produce a single text file with all couple of columns indexes and the Pearson correlation coefficient [1] of the two columns.
Recall that the Pearson correlation coefficient is a measure of linear correlation between two series of data, so for example in output we may produce the line: `(0, 1):  0.0013`, meaning that column 0 and 1 are not correlated.

1.2 The Hardware Setup

All benchmarks and measures have been performed on a remote otherwise idle server with the following specs:

- **GPU**: GeForce GTX 1080 Ti [2]
- **CPU**: Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz, 8 core [3]
- **RAM**: 32GB
- **HDD**: ST2000DM001-1ER1 (2TB)

1.3 The Tools we used

The tools we used to benchmarking and performance analysis are the following:

- **time** [4]: a Linux utility command to get statistics about execution time of a process
- **nvprof** [5]: is part of the CUDA package, this tool allows you to collect and view profiling data of CUDA-related activities on both CPU and GPU, including kernel execution, memory transfers, etc.
- **perf** [6]: a Linux tool that is capable of statistical profiling of the entire system (both kernel and userland code)

That's how we used `perf` to profile respectively CPU and GPU executables:

- `$ sudo perf record ./multi inputs/dataset-256-1000000.csv >> /dev/null`
- `$ sudo perf record ./kernel inputs/dataset-256-1000000.csv >> /dev/null`

And to visualize the results: `$ sudo perf report`.
Note: since perf can't measure directly execution on GPU, the time spent on GPU the calculated as the missing percentage from the total time and the time spent on the GPU.

2 The Algorithm

2.1 One Pass Pearson Correlation Coefficient

By definition the Person correlation is just a normalized measurement of the covariance, the formula for the correlation coefficient is:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$$

where:

- cov is the covariance

- ρ_X is the standard deviation of X
- ρ_Y is the standard deviation of Y

But there is one problem: following the classical definition requires of the input data. With some substitutions it's possible to obtain a formula that requires only one pass, the formula is the following:

$$\rho_{X,Y} = \frac{E[XY] - E[X]E[Y]}{\sqrt{E[X^2] - E[X]^2}\sqrt{E[Y^2] - E[Y]^2}}$$

2.2 Implementation

The algorithm implementation can be summarized in the following steps:

- We read the input csv N lines at the time
- We calculate a partial result of this N lines for all the couples of columns
- Sum new partial result with the previous store
- Iterate until the whole file is read
- In one final computation convert the partial result into the correlation coefficient
- Save or print the results

2.3 I/O examples

Here we have reported a possible (despite reduced) input file (listing 1) with the corresponding output (listing 2).

```
"col1","col2","col3","col4","col5","col6","col7","col8"
"13562","10357","30754","26825","23508","30043","28938","10111"
"4808","13096","16702","21936","14933","6328","20732","10430"
"12335","25313","1066","15242","24547","15170","3110","13334"
"21891","10185","26028","24719","19499","23544","20108","13872"
"27943","31421","16439","8083","9641","20630","30652","17694"
"2940","20078","29753","26833","3845","4946","22617","24890"
"18711","32243","17611","13786","17079","25250","22190","25307"
"9993","32422","29676","29116","9166","24302","11250","32370"
```

Listing 1: Example of an input file

```
(0,1) 0.253106
(0,2) -0.170867
(0,3) -0.630296
(0,4) 0.225982
(0,5) 0.626356
(0,6) 0.342535
(0,7) -0.114682
(1,2) -0.294529
(1,3) -0.504529
(1,4) -0.375457
(1,5) 0.12661
(1,6) -0.228954
(1,7) 0.719545
(2,3) 0.728292
(2,4) -0.414364
(2,5) 0.239953
(2,6) 0.488402
(2,7) 0.32773
(3,4) -0.120147
```

```
(3,5) -0.0289303
(3,6) -0.123404
(3,7) 0.156105
(4,5) 0.447088
(4,6) -0.250147
(4,7) -0.65898
(5,6) 0.177765
(5,7) 0.0640271
(6,7) -0.180228
```

Listing 2: Example of an ouput file

3 Results

Initially we analyzed and compared CPU and GPU based solutions, varying the input parameters, then we studied the execution time and the relative speedup of the GPU solution with respect to the number of threads.

3.1 Data collection methodology

All displayed results have been obtained by performing multiple executions of all the pairs (executable, input dataset) to get a statistically meaningful mean of the execution times. Hundreds of cases have been tested but only a bunch of them are here displayed for brevity.

3.2 Execution time for different configurations

3.2.1 Varying number of rows

Figure 1 shows the execution time of 4 different configurations with respect to the number of rows (the number of columns was fixed at 128). Table 1 shows the speedup of said configurations with respect to the single threaded version.

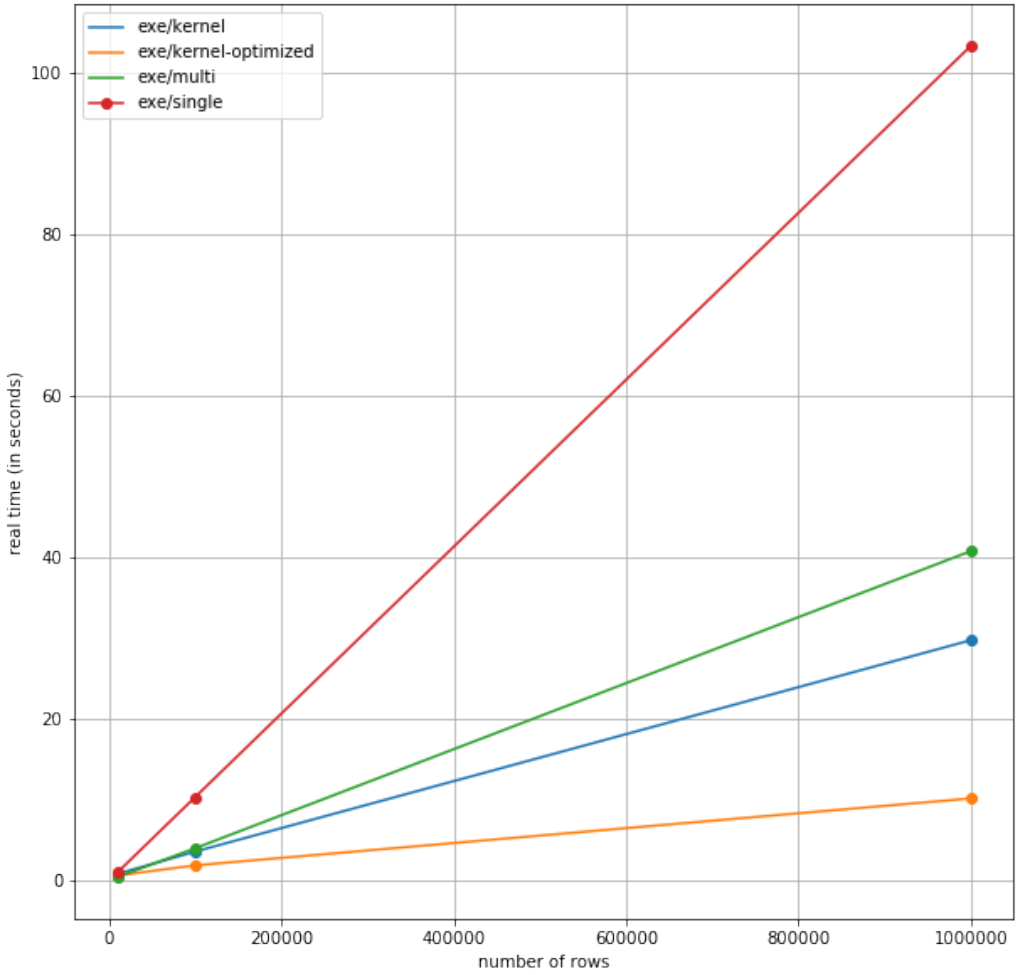

Figure 1: Execution time over number of rows with different configurations

Configuration	Speedup
exe/single	1.0
exe/multi	2.5315
exe/kernel	3.4680
exe/kernel-optimized	10.1147

Table 1: Speedup with respect to the single threaded version

3.2.2 Varying number of columns

Figure 2 shows the execution time of 4 different configurations with respect to the number of columns (the number of columns was fixed at 100k). Table 2 shows the speedup of said configurations with respect to the single threaded version.

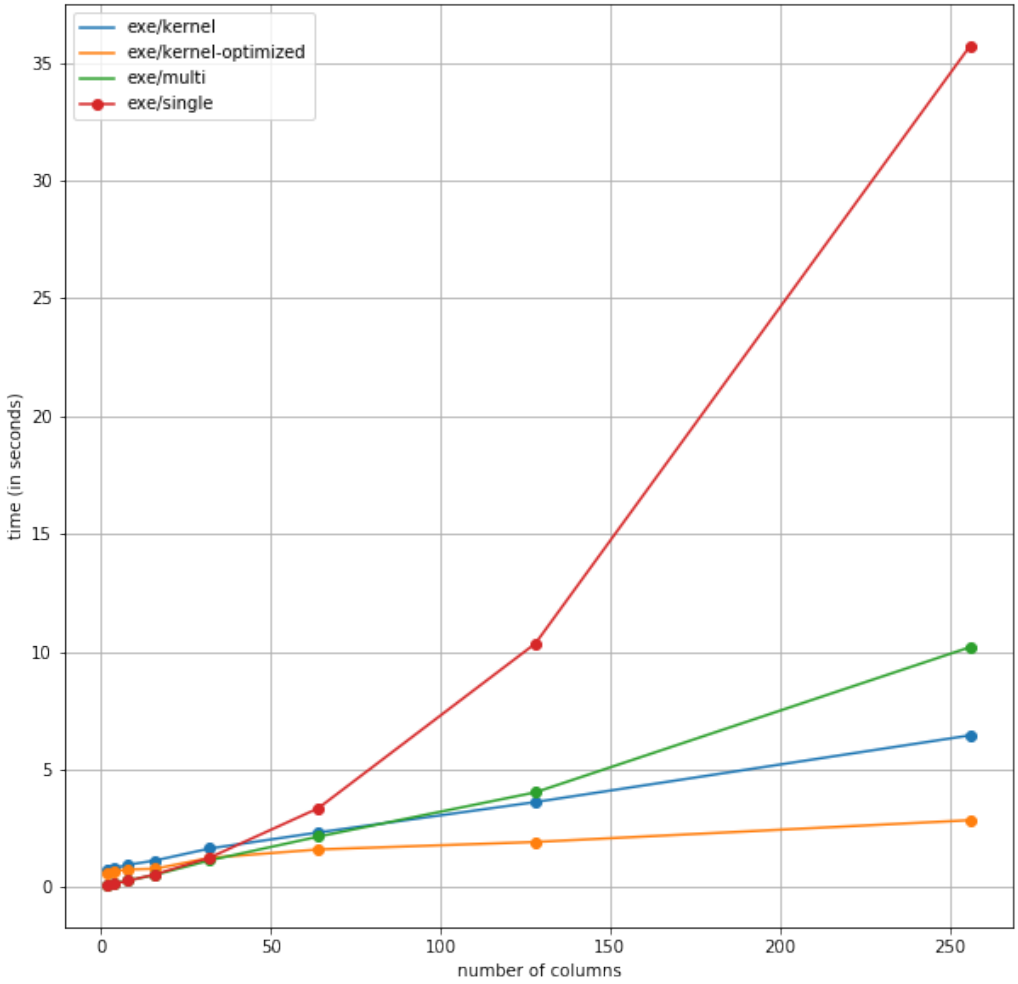

Figure 2: Execution time over number of rows with different configurations

Configuration	Speedup
exe/single	1.0
exe/multi	3.500
exe/kernel	5.530
exe/kernel-optimized	12.50

Table 2: Speedup with respect to the single threaded version

3.3 GPU Only

3.3.1 GPU Speedup

In figure 3 we see the speedup for the GPU solution over the number of GPU threads (Note that the scale is logarithmic on the x axis). The same information is shown in table 3. It's clearly visible how after roughly 16k thread the speedup hits a plateau. The benchmark has been executed with an input file of 256 columns and 100K rows.

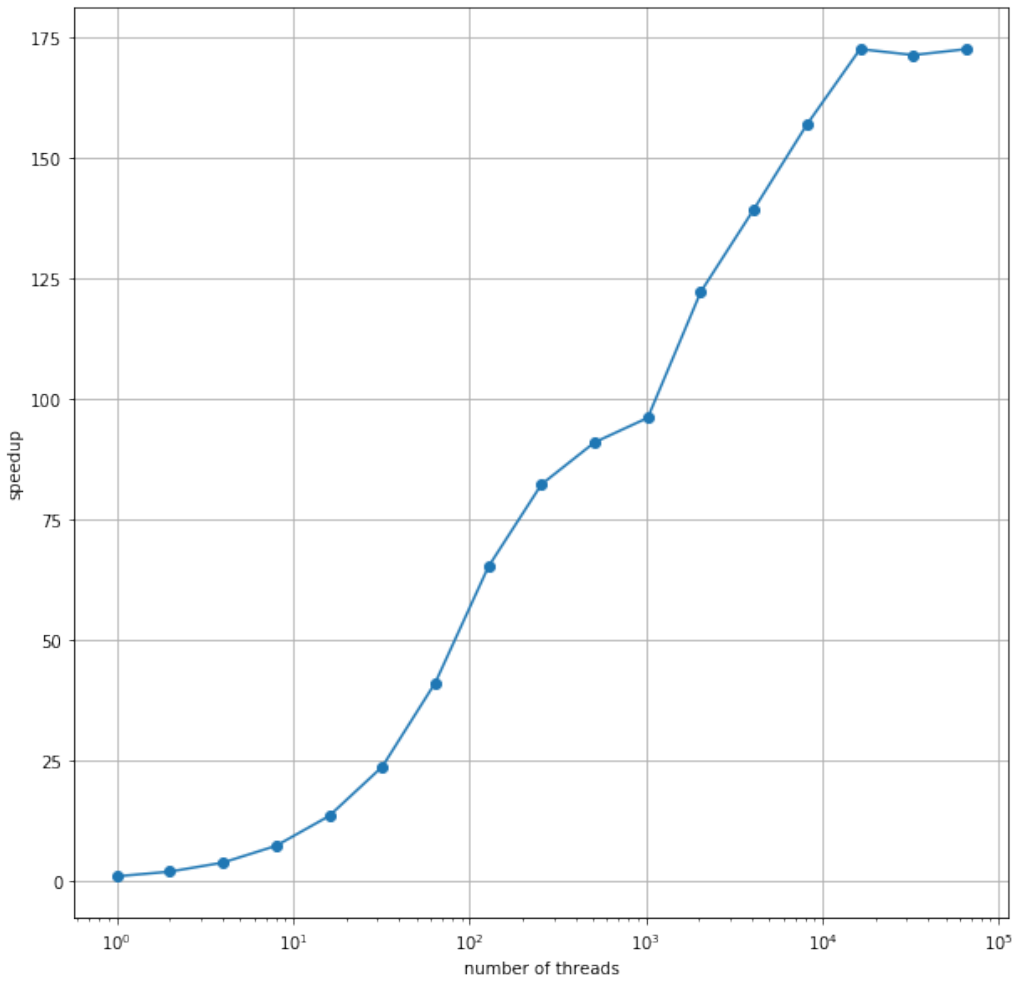

Figure 3: GPU speedup over the number of threads

number of GPU threads	Speedup
1	1.0
2	1.977180
4	3.849154
8	7.328337
16	13.518071
32	23.729698
64	41.179231
128	65.242591
256	82.319421
512	91.024612
1024	96.112646
2048	122.257789
4096	139.391881
8192	156.984066
16384	172.601079
32768	171.362547
65536	172.594878

Table 3: GPU speedup over the number of threads

3.3.2 GPU Isolated Execution Time

With the tool `nvprof` [5] we measured GPU only execution time over a varying input size, CUDA kernel start-up require a varying amount of time, causing the initial bent in the graph, see figure 4. The same GPU-time is

achieved both with or without compiler optimizations, that affect only the sequential part of the task.

Measured performed with `perf` confirmed the fraction of execution time spent on GPU was about 14%. The result is coherent with the ones obtained with the other profiling tools.

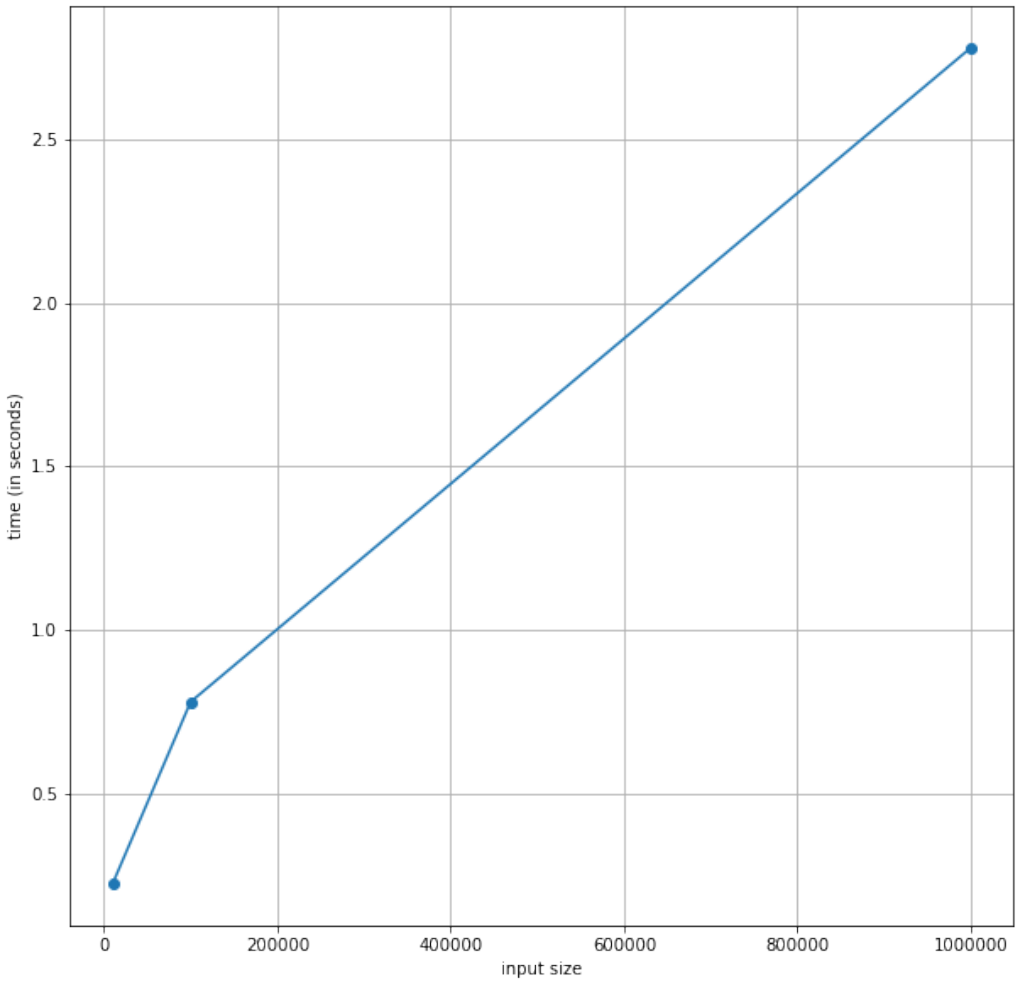

Figure 4: GPU only execution time over a varying input size

4 Conclusions

As a consequence of our study we can say that:

- The given task is a naturally parallelizable and the limiting factor is the sequential part of the execution.
- GPU-based solutions outperform CPU-based solution with large datasets (many rows) of many (¿=64) columns with highly optimized code (i.e. CPU relying on vectorised instructions).
- Without relying on CPU compiler optimizations, GPU-based solutions outperform CPU-only solution with large datasets of at least 32 columns.
- To process a dataset of 256 columns and 1'000'000 rows GPU requires only about 2.7s, while disk read time requires about 10s.
- Looking for further optimizations is worthless to reduce execution time because the IO+parsing bottleneck would make them imperceptible.
- It is not surprising that we had no way to improve GPU execution time since the whole task require no more than simple products and sums of floating point values, a natural task for a GPU processor.

References

[1] https://en.wikipedia.org/wiki/Pearson_correlation_coefficient.

[2] https://www.techpowerup.com/gpu-specs/geforce-gtx-1080-ti.c2877.

[3] https://www.intel.com/content/www/us/en/products/sku/80807/intel-core-i74790k-processor-8m-cache-up-to-4-40-ghz/specifications.html.

[4] https://man7.org/linux/man-pages/man1/time.1.html.

[5] https://docs.alliancecan.ca/wiki/Nvprof.

[6] https://perf.wiki.kernel.org/index.php/Main_Page.