PCS5120 Homework

Giuliano A. F. Belinassi[1]

[1]Institute of Mathematics and Statistics (IME) – University of São Paulo (USP)
Rua do Matão, 1010 – São Paulo – SP – Brazil

In various research fields where numerical linear algebra is required eighter because of its facility or inexisence of analytical solutions, can take advantage of packages such as Lapack or BLAS. One major concern about these libraries is its performance, subject discussed in this report. Here we target the routine designed to multiply two matrices in single precision floating point (`SGEMM`).

Althrough implementing a matrix-matrix multiplication seems trivial by its concept, it is fairly difficult to provide an efficient code because of various reasons, such as cache usage. The use of techniques such as block tiling can yield better results because it improves cache usage, but it is important to select the size of the block that results in a best overall performance. Such size can be determined empirically.

1. The database

We used the database "*SGEMM GPU kernel performance Data Set*" provided by *UCI machine learning repository*[1]. Briefly, it has timings of multiplication of two matrices measured in *ms*, each one of size 2048×2048, using a combination of 14 parameters, totalizing 241601 lines in the database. All multiplications were computed using Graphic Processing Units (GPU).

2. Data analysis

We used Orange in our analysis. Our first objective was to find the distribuition of the average of the four executions per sample to check possible improvements or deterioration of the performance. For creating a column with the average of four executions, we used the Orange's Feature Constructor. Unfortunately, there is no average function implemented here, so we had to calculate such function by the definition $\left(\frac{1}{n}\sum x_i\right)$.

Once we had the average column, we plotted the distribution, as illustrated by Figure 1. Notice that most of the averages concentrate under 200ms, and there are some data around 2400ms. Such high timing can be caused by the parameters itself or be an outlier because there were other programs running on the computer.

Once we got the distribuition of timings, we focused on what parameters yielded best results. Orange's Data Table let us find the combination of parameters because how straightfoward the table sorting feature is implemented.

Maybe for pratical reasons, the showed results is enough to provide a setup for an efficient implementation, but we can explore the provided data in order to create projections for perhaps even better timings.

Using the scatterplot tool, we could plot how each parameter impacts the average of timing. Analysing such graphics, we could determine that `MWG`, the tiling line dimension of the matrix, is the parameter that mostly impacts performance, and such impact is

[1]`https://archive.ics.uci.edu/ml/datasets/SGEMM+GPU+kernel+performance`

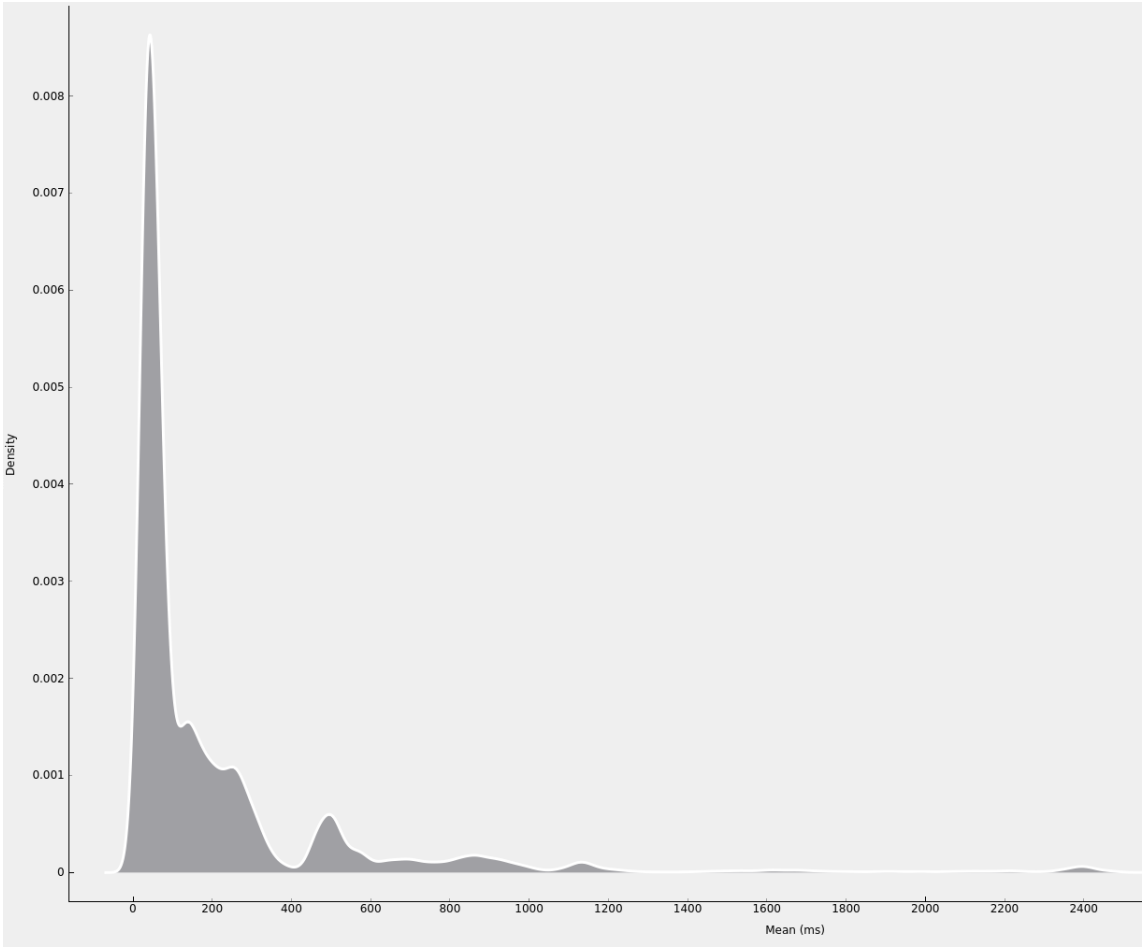

Figure 1. Distribution of the average of four samples per parameter.

inverse proportional to the timing. So, for a better performance, we must minimize the value of `MWG`. Figure 2 illustrates the impact of this parameter.

Regarding other parameters, we could insert the graphics for each one, but we chose to not do so to not extend this report unnecessarily.

3. Conclusions

Analysing the information in the database in order to extract how the parameters impact the overall performance of the routine can aid the development of increasingly efficient numerical libraries, and such analysis was made possible by Orange in a visually intuitivelly manner. There was no need to write extensive code to prepare the data for being processed, nor to plot graphics.

Table 1. Combination of parameters that yielded the 5# best results

MWG	NWG	KWG	MDIMC	NDIMC	MDIMA	NDIMB	KWI	VWM	VWN	STRM	STRN	SA	SB	Mean (ms)
128	128	16	16	32	32	32	8	2	4	1	0	1	1	13.3175
128	128	16	16	32	32	32	8	2	4	1	1	1	1	13.42
128	128	16	16	32	32	32	2	2	2	1	1	1	1	13.74
128	128	16	16	32	32	32	8	2	2	1	1	1	1	14.4475
128	64	16	16	16	16	32	2	2	2	1	0	1	1	14.6325

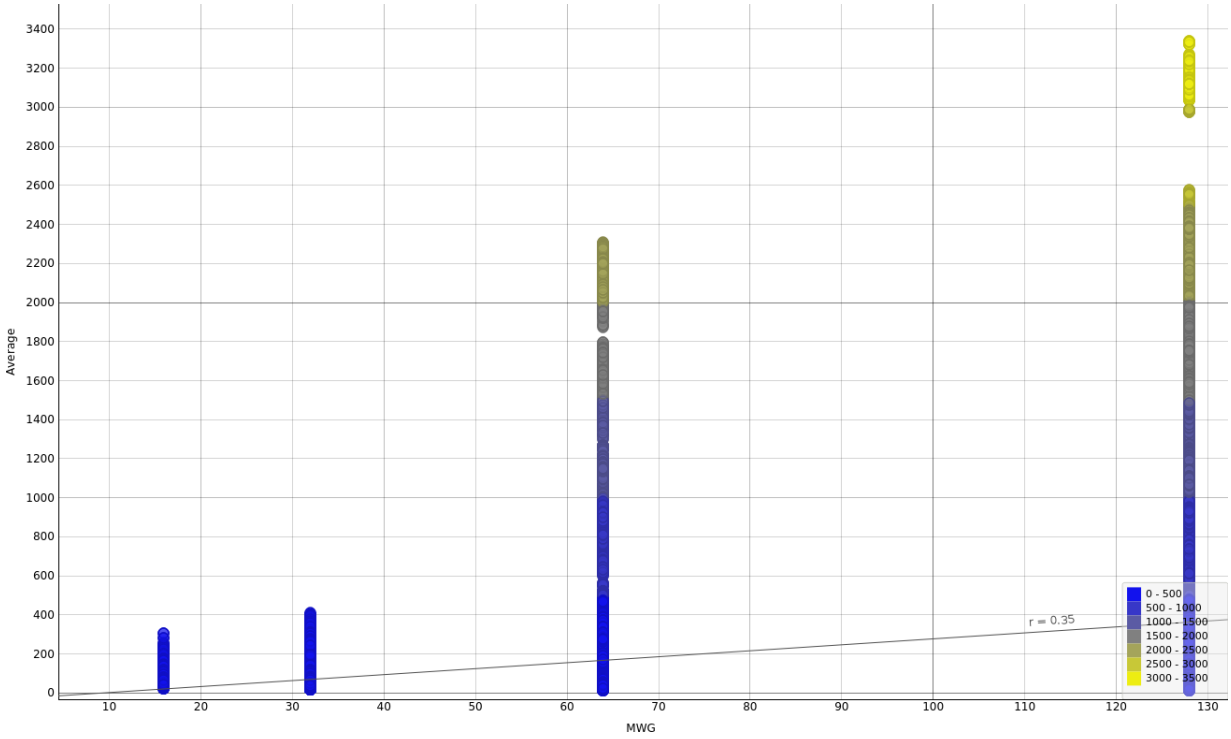

Figure 2. Plot of `MWG` versus Timing Average.

On the other hand, Orange is not a fairly efficient tool performance-wise. For the database selected, it used around 12Gb of memory just for loading it, and every step took very long to compute and display any result. The use of more efficient data structures and parallel computing may address these issues.