Optimizing a Boundary Elements Method implementations with GPU

Giuliano A. F. Belinassi[1], Rodrigo Siqueira[1], Ronaldo Carrion[2] Alfredo Goldman[1]

[1]Instituto de Matemática e Estatística (IME) – Universidade de São Paulo (USP)
Rua do Matão, 1010 – São Paulo – SP – Brazil

[2]Escola Politécnica (EP) – Universidade de São Paulo
Avenue Professor Mello Moraes, 2603 – São Paulo – SP – Brazil

Abstract. *This meta-paper describes the style to be used in articles and short papers for SBC conferences. For papers in English, you should add just an abstract while for the papers in Portuguese, we also ask for an abstract in Portuguese ("resumo"). In both cases, abstracts should not have more than 10 lines and must be in the first page of the paper.*

1. Introduction

Since the computer was proven a useful machine, there always has been an interest of building faster versions of it to solve bigger and more complex problems in a matter of time that no human could. One way archiving this is by building more complex sequential CPUs, with more transistors. Recently, the size of CPU components got so small that it is becoming difficult to create such faster sequential CPUs. As a consequence of this fact, CPU manufacturers are investing in multicore CPUs, capable of running one than one task asynchronously.

Parallel computing is hard to define, but intuitively, it is a computation method that allows data to be distributed and processed simultaneously. In Flynn's taxonomy [Pacheco 2011], there are two types of parallel computer archictectures:

1. Single Instruction, Multiple Data (SIMD); a processor that allows a chunk of data to be loaded and a single instruction to be used to process it. As example of SIMD, there is Intel's SSE [sse] and Graphics Processing Units (GPU) are examples it.
2. Multiple Instruction, Multiple Data (MIMD); a system consisting of multiple independent processing units, executing asynchronously. Multicore CPUs are a example of MIMD.

Since the computer was proven a useful machine, there always has been an interest of building faster versions of it to solve bigger and more complex problems in a matter of time that no human could. One way archiving this is by building more complex sequential CPUs, with more transistors. Recently, the size of CPU components got so small that it is becoming difficult to create such faster sequential CPUs. As a consequence of this fact, CPU manufacturers are investing in multicore CPUs, capable of running one than one task asynchronously.

Some graphical problems, such as dot product, can take advantage of SIMD instructions to reduce the required time to compute them. Here is invented the GPU.

Originally, GPU was designed to render graphics in real time and OpenGL and DirectX were the first libraries designed to access GPUs resources and provide graphics.

However, researchers realized that GPUs could be used for other applications different from graphics. Consequently, graphical libraries were used by engineers and scientists in their specific problems, however, they had to convert their problem into a graphical domain.

NVIDIA noticed a new demand for their products and created an API called CUDA to enable the use of GPUs in general purpose situation. CUDA has the concept of kernels, which are functions called from host to be executed in the GPU threads. Kernels are organized into a set of blocks wherein each block is a set of threads that cooperate with each other [Patterson and Hennessy 2007].

GPU's memory is divided into global memory, local memory, and shared memory. First, it is a memory that all threads can access. Second, it is a memory that is private to a thread. Third, it is a low-latency memory that is shared between all threads in the same block. [Patterson and Hennessy 2007].

The Boundary Elements Method (BEM) is a computational method of solving linear partial differentials equations that have been formulated as integral equations. This is used in many engineering areas, one of them is to analyze waves propagation on the ground and its effects on nearby structures. It requires high computational power and this article discusses how GPUs can be used to accelerate its calculations.

Given a sequential implementation of BEM by [**?**], the main objectives of this project includes the creation of automated tests to check if the modified program results are numerically compatible with the original version; somewhat modernize the legacy code, which was written in Fortran 77; optimize the code, removing repeated unnecessary calculations and managing a better usage of the Central Processing Unit (CPU) resources; identify the most time-consuming subroutines and paralellize them.

2. Norms

In order to check if the final result obtained by the parallel program is numerically compatible with the original, the concept of vector and matrix norms are necessary. Let $A \in \mathbb{C}^{m \times n}$. [Watkins 2004] defines a matrix ∞-norm and matrix 1-norm as:

$$\|A\|_\infty = \max_{1 \le i \le m} \sum_{j=1}^{n} |a_{ij}| \quad \|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} |a_{ij}| \tag{1}$$

All norms have the propierty that $\|A\| = 0$ if and only if $A = 0$, and the same result asserts for matrix norms. Let f and g be two numerical algorithms that solves the same problem, but in a different fashion. Let now y_f be the result computed by f and y_g be the result computed by g. The *error* between those two values can be measured computing $\|y_f - y_g\|$.

3. Parallelization Technique

A parallel implementation of BEM began by analyzing and modifying a sequential code provided by Carrion. Gprof [bin], a profiling tool by GNU, showed that the most time-consuming routine was Nonsingd, a subroutine designed to solve small parts of the non-singular dynamic problem. Since most calls to Nonsingd were from Ghmatecd, most of the parallelization effort was focused on that last routine.

3.1. Ghmatecd Parallelization

Ghmatecd works in the following way: It assembles two complex matrices H and G by computing smaller 3×3 matrices returned by Nonsingd and Sigmaec. Let n be the number of mesh elements and m be the number of bondary elements. Algorithm 1 illustrates how Ghmatecd works.

Algorithm 1 Creates $H, G \in \mathbb{C}^{(3m)\times(3n)}$

```
1:  procedure GHMATECD
2:      for j := 1, n do
3:          for i := 1, m do
4:              ii := 3(i − 1) + 1; jj := 3(j − 1) + 1
5:              if i == j then
6:                  Gelement, Helement ← Sigmaec(i)   ▷ two 3 × 3 complex matrices
7:              else
8:                  Gelement, Helement ← Nonsingd(i, j)
9:              G[ii : ii + 2][jj : jj + 2] ← Gelement
10:             H[ii : ii + 2][jj : jj + 2] ← Helement
```

There is no interdependency between iteractions, so the two nested loop in the algorithm can be computed in parallel, hence, both matrices H and G can be built in a parallel fashion. If the number of processors is small, then parallelizing the two nested loops in line computing that nested loop in parallel is enough because even for small instances of the problem, $n\times m$ will be bigger than the number of processors. By other hand, GPUs have greater parallel capability than CPUs, and the above strategy alone would generate a waste of computational resources. Since Nonsingd is the cause of the high time cost of Ghmatecd, the main effort was to implement an optimized version of Ghmatecd, called Ghmatecd_Nonsingd, that only computes the Nonsingd case in the GPU, and leave Sigmaec to be computed in the CPU after the computation of Ghmatecd_Nonsingd is completed.

Let g be the number of Gauss quadrature points. The Algorithm 2 pictures this new strategy.

The Ghmatecd_Nonsingd can be implemented in CUDA kernel in the following way: Inside a block, create $g \times g$ threads to compute in parallel the two nested loop in lines 6 to 7, allocating *Hbuffer* and *Gbuffer* in shared memory. Since these buffers contain matrices of size 3×3, 9 of these $g \times g$ threads can be used to sum all matrices. Notice that $g \geq 3$ is necessary for that condition, and that g is also upper-bounded by the amount of shared memory available in the GPU. Launching $m \times n$ blocks to cover the two nested loops in lines 2 to 3 will generate the entire H and G.

The Ghmatecd_Sigmaec can be parallelized with a simple OpenMP Parallel for clause.

3.2. Ghmatece Parallelization

Ghmatece is a routine designed to create two real matrices H and G associated with the static problem, and it is very similar to Ghmatecd. That routine can be implemented in CUDA in the same way as described in Ghmatecd.

Algorithm 2 Creates $H, G \in \mathbb{C}^{(3m)\times(3n)}$

1: **procedure** GHMATECD_NONSINGD
2: **for** $j := 1, n$ **do**
3: **for** $i := 1, m$ **do**
4: $ii := 3(i-1)+1; jj := 3(j-1)+1$
5: Allocate *Hbuffer* and *Gbuffer*, buffer of matrices 3×3 of size g^2
6: **if** $i \neq j$ **then**
7: **for** $y := 1, g$ **do**
8: **for** $x := 1, g$ **do**
9: *params* $\leftarrow$ ComputeParameters(i, j, x, y)
10: *Hbuffer*$(x, y) \leftarrow$ GenerateMatrixH$(x, y, params)$
11: *Gbuffer*$(x, y) \leftarrow$ GenerateMatrixG$(x, y, params)$
12: $Gelement \leftarrow$ SumAllMatricesInBuffer(*Gbuffer*)
13: $Helement \leftarrow$ SumAllMatricesInBuffer(*Hbuffer*)
14: $G[ii : ii+2][jj : jj+2] \leftarrow Gelement$
15: $H[ii : ii+2][jj : jj+2] \leftarrow Helement$
16: **procedure** GHMATECD_SIGMAEC
17: **for** $i := 1, m$ **do**
18: $ii := 3(i-1)+1$
19: $Gelement, Helement \leftarrow$ Sigmaec(i)
20: $G[ii : ii+2][ii : ii+2] \leftarrow Gelement$
21: $H[ii : ii+2][ii : ii+2] \leftarrow Helement$
22: **procedure** GHMATECD
23: Ghmatecd_Nonsingd()
24: Ghmatecd_Sigmaec()

4. Methodology

The error between CPU and GPU versions of H and G matrices were computated by calulating $\|H_{cpu} - H_{gpu}\|_1$ and $\|G_{cpu} - G_{gpu}\|_1$. An automated test check if this value is bellow 10^{-4}.

The elapsed time was computed in seconds with the OpenMP library function OMP_GET_WTIME. This function calculates the elapsed wall clock time in seconds with double precision.

The GPU time results are a sum of the time elapsed to move the data to GPU, launch and execute the kernel, wait for the result, and move the data back to computer's main memory.

For each experiment, there were 4 samples of data, with 240 mesh elements and 100 boundary elements; 960 mesh elements and 400 boundary elements; 2160 mesh elements and 900 boundary elements; 4000 mesh elements and 1600 boundary elements. All experiments set the Gauss Quadrature Points to 8.

For each experiment execution, it was collected the total time elapsed by the serial code, with OpenMP and CUDA. In order to assure the results, all experiments are run 30 times for each sample. It is important to highlight that before running each experiment, a warmup procedure is executed, that consists of running the application with the sample 3 times without collecting any result.

Two computers were used in the experiments. Both machines have a AMD A10-7700K with 4 cores, but one have a NVIDIA GeForce GTX980 and another have a GeForce GTX750.

5. Results

6. General Information

All full papers and posters (short papers) submitted to some SBC conference, including any supporting documents, should be written in English or in Portuguese. The format paper should be A4 with single column, 3.5 cm for upper margin, 2.5 cm for bottom margin and 3.0 cm for lateral margins, without headers or footers. The main font must be Times, 12 point nominal size, with 6 points of space before each paragraph. Page numbers must be suppressed.

Full papers must respect the page limits defined by the conference. Conferences that publish just abstracts ask for **one**-page texts.

7. First Page

The first page must display the paper title, the name and address of the authors, the abstract in English and "resumo" in Portuguese ("resumos" are required only for papers written in Portuguese). The title must be centered over the whole page, in 16 point boldface font and with 12 points of space before itself. Author names must be centered in 12 point font, bold, all of them disposed in the same line, separated by commas and with 12 points of space after the title. Addresses must be centered in 12 point font, also with 12 points of space after the authors' names. E-mail addresses should be written using font Courier New, 10 point nominal size, with 6 points of space before and 6 points of space after.

The abstract and "resumo" (if is the case) must be in 12 point Times font, indented 0.8cm on both sides. The word **Abstract** and **Resumo**, should be written in boldface and must precede the text.

8. CD-ROMs and Printed Proceedings

In some conferences, the papers are published on CD-ROM while only the abstract is published in the printed Proceedings. In this case, authors are invited to prepare two final versions of the paper. One, complete, to be published on the CD and the other, containing only the first page, with abstract and "resumo" (for papers in Portuguese).

9. Sections and Paragraphs

Section titles must be in boldface, 13pt, flush left. There should be an extra 12 pt of space before each title. Section numbering is optional. The first paragraph of each section should not be indented, while the first lines of subsequent paragraphs should be indented by 1.27 cm.

9.1. Subsections

The subsection titles must be in boldface, 12pt, flush left.

10. Figures and Captions

Figure and table captions should be centered if less than one line (Figure 1), otherwise justified and indented by 0.8cm on both margins, as shown in Figure 2. The caption font must be Helvetica, 10 point, boldface, with 6 points of space before and after each caption.

Figure 1. A typical figure

In tables, try to avoid the use of colored or shaded backgrounds, and avoid thick, doubled, or unnecessary framing lines. When reporting empirical data, do not use more decimal digits than warranted by their precision and reproducibility. Table caption must be placed before the table (see Table 1) and the font used must also be Helvetica, 10 point, boldface, with 6 points of space before and after each caption.

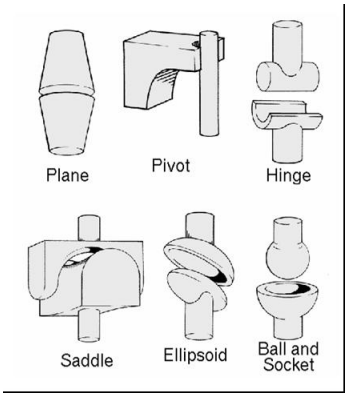

Figure 2. This figure is an example of a figure caption taking more than one line and justified considering margins mentioned in Section 10.

Table 1. Variables to be considered on the evaluation of interaction techniques

	Chessboard top view	Chessboard perspective view
Selection with side movements	6.02 ± 5.22	7.01±6.84
Selection with in-depth movements	6.29±4.99	12.22±11.33
Manipulation with side movements	4.66± 4.94	3.47±2.20
Manipulation with in-depth movements	5.71 ±4.55	5.37 ±3.28

11. Images

All images and illustrations should be in black-and-white, or gray tones, excepting for the papers that will be electronically available (on CD-ROMs, internet, etc.). The image resolution on paper should be about 600 dpi for black-and-white images, and 150-300 dpi for grayscale images. Do not include images with excessive resolution, as they may take hours to print, without any visible difference in the result.

12. References

Bibliographic references must be unambiguous and uniform. We recommend giving the author names references in brackets, e.g. [Knuth 1984], [Boulic and Renault 1991], and [Smith and Jones 1999].

The references must be listed using 12 point font size, with 6 points of space before each reference. The first line of each reference should not be indented, while the subsequent should be indented by 0.5 cm.

References

Gnu binutils. `https://www.gnu.org/software/binutils/`. Accessed: 2017-05-08.

Intel sse. https://www.intel.com/content/www/us/en/support/processors/000005779.html?_ga=2.114110593.729918000.1502128033-2045315159.1502128033. Accessed: 2017-07-08.

Boulic, R. and Renault, O. (1991). 3d hierarchies for animation. In Magnenat-Thalmann, N. and Thalmann, D., editors, *New Trends in Animation and Visualization*. John Wiley & Sons ltd.

Knuth, D. E. (1984). *The TEX Book*. Addison-Wesley, 15th edition.

Pacheco, P. (2011). *An introduction to parallel programming*. Elsevier.

Patterson, D. A. and Hennessy, J. L. (2007). Computer organization and design. *Morgan Kaufmann*.

Smith, A. and Jones, B. (1999). On the complexity of computing. In Smith-Jones, A. B., editor, *Advances in Computer Science*, pages 555–566. Publishing Press.

Watkins, D. S. (2004). *Fundamentals of matrix computations*, volume 64. John Wiley & Sons.