Recurrence quantification analysis in images with CUDA

Thiago de Lima Prado
Department of Physics, Federal University of Paraná, 81531-990, Curitiba, Paraná, Brazil.
(Date: 8 June 2012)

Abstract

This work has as first goal develop an parallel algorithm considerably faster than it serial pair. The algorithm is based on recurrence plots idea, that is a square matrix where it is possible detect many complex behaviours (e.g. with the recurrence quantification analysis technique) of dynamical systems. In the results and conclusion part, we will present the time saved using *CUDA* with *C* programming language in comparison with the serial program and the time relation with the number of kernels acess for some rates of threads per block.

Introduction

The physics nowadays needs faster computers for many fields of research. But we have reached a physical limit for increased clocks on *CPUs* due excessive heating, one of many solutions for this problem was increase the number of cores in the *CPUs* for parallel processing. This solution has increased considerably the processing with parallel algorithms, and has introduced some new technologies like *OMP* [1] that have boosted the power of processing for science. Whatever when the problem is huge enough or need to be extremely fast solved, we have used the so called supercomputers, but these big machines are so expensive that only few reaserch centers have acess to them. With this problem in mind, recently a new technology have been developed by *Nvidia* called *CUDA* [2], and it uses the power of the *GPUs* for achieving processing scales only previously reached by supercomputers, but with cost in the level of a modest *CPU*.

In this work we will present our work on physics, that was developed in *C* programming language for single core processing and we have compared it with one adapted to use *GPU-CUDA*. Was tried some conditions of number of threads, number of kernel acess and we have achieved the gain compared with this conditions and against single core solution of the problem in terms of processing time.

Finaly we have presented our observations and the knowledge acquired for the best processing time to solve our problem.

Methods and Tools

In the 90s reaserch have been developed for understand the non-linear phenomena in physics. One of the methods that have helped in many non-linear problems is the recurrence plots or RPs [3], this technique is based in the poincaré theorem that says about recurrent events in the phase space. This plot is a visual tool used in time or recently space series [4], and it gives you some insight of the dynamical behaviour of the system in study. We can mathematicaly write the RPs as in equation [Eq. 1]:

$$R_{i,j} = \theta\left(\epsilon - \left[\!\left[x_i - x_j \right]\!\right]\right) \tag{1}$$

It says that each point in the time/space serie (x_i and x_j) can be as close in the phase space as ε, when this happen the Heaviside function will atribute 1, when they are not close enough, it will be zero. When compared all points in the serie with each other, we will have a square matrix with side equal to the number of points in the serie. There is some variations of it, and we use one that gives a minimum threshold too, as the equation [Eq. 2]:

$$R_{i,j} = \theta(\epsilon_{max} - [\![x_i - x_j]\!]) \theta([\![x_i - x_j]\!] - \epsilon_{min}) \quad (2)$$

Some time later the reaserchers [5] have developed lots of tools for quantization of this plots, and one very important is the recurrence rate that is part of what we call recurrence quantification analysis (RQAs). The recurrence rate or commonly called RR can be described as follow [Eq. 3]

$$RR = (1/N^2) \sum_{i,j=1}^{N} R_{i,j} \quad (3)$$

With *N* normally as the number of points in the serie (in our case is slightly different). What we use is some variation of this two physical mechanisms of non-linear serie analysis, but we use it as a kind of filter in images that act like an edge detector, as we can see in the natural image to the left and the filtered one to the right on image [Fig 1]:

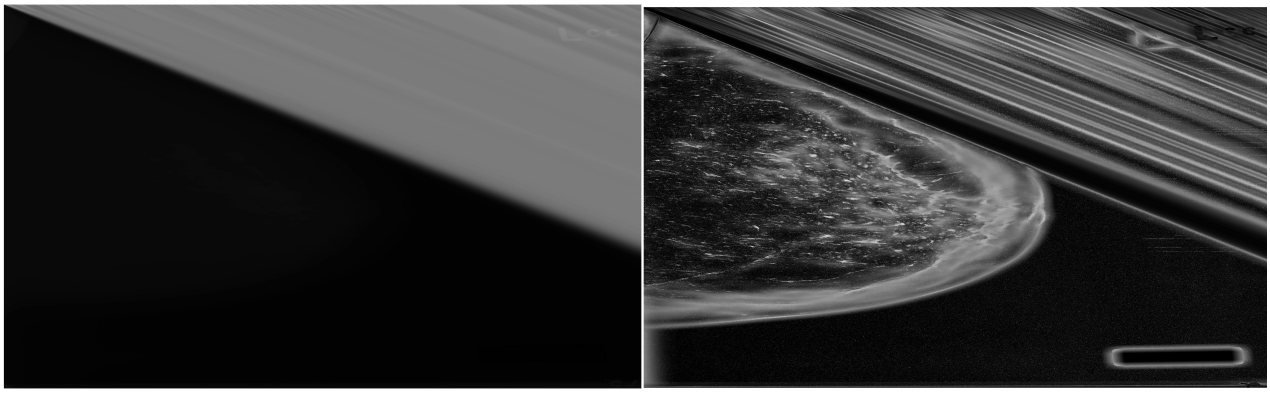

Fig 1: The image to the left is the original mammography (with *5928x3776* pixels) from the *USF* database [6], and to the right is the image after our algorithm.

The problem in this technique stays on the size of *N*, when *N* becomes larger (to have some kind of zoom) the number of operations increase with N^2 , since our image has more than 10 million points, we acquire a big problem with processing time.

We have two algorithms that have been adapted for *CUDA*, the first one is properly the image filter, the second one is the program that finds the best combination of threshold for the first program. The second one is considerably slower than the other, and automatically fits better when used in the *GPU*.

As ours equipaments of work, we have used one *Nvidia GTX 460* with *1Gb* of memory and as *CPU* was used an *Intel Core I7* with *3 GHz* of clock, all the programs was developed in *C* programming language.

Results

As the first part of this section we will present the results of the filter algorithm, next we will show the results for the optimization threshold algorithm and to finish we will show the variations of time due to multiple acess of kernels function.

The filter image algorithm depends a lot of the *N* factor as we have previously said, so we have done a plot (Fig 2) that compares the time spent of single core, eight cores with *OMP*, *CUDA* with *16x16* threads per block and *CUDA* with *32x32* threads per block algorithms in function of *N*. This number of threads per block was chosen propositely because there is a great difference in each other due the increase of *N*.

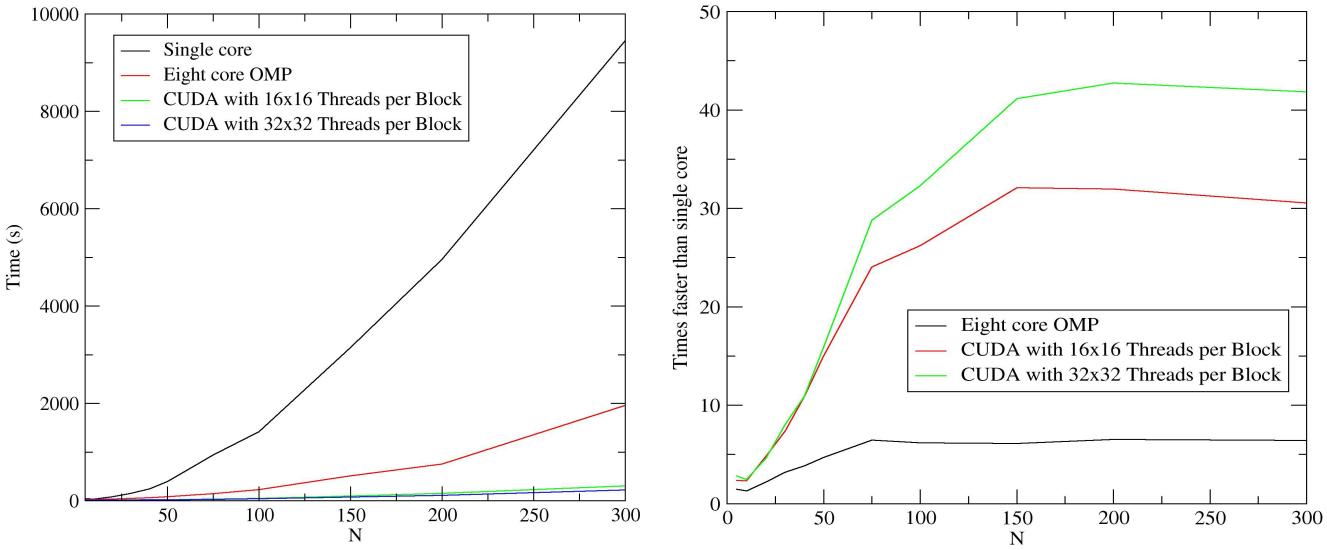

Fig 2: The left plot shows the time spent by the programs for different values of *N*, and the right plot says how much faster is the parallel programs for the different cases shown against the serial case.

It is really interesting notice that even parallel *OMP* isn´t so efficient as any of the *CUDA* cases. Other point to be mentioned is the increasing curve of gain, we notice the advantage of using parallelism growing with the number of operations needed. In this situation there is a limit where it stops to increase, for *OMP* is something near *N=75* that is the same region where the number of threads per block begin to become an really important issue, so important that it becomes near *36%* faster use *32x32* threads per block than use *16x16*.

The image (Fig 3) shows the same plots as before, but now for the algorithm responsible to find the best possible threshold for our problem.

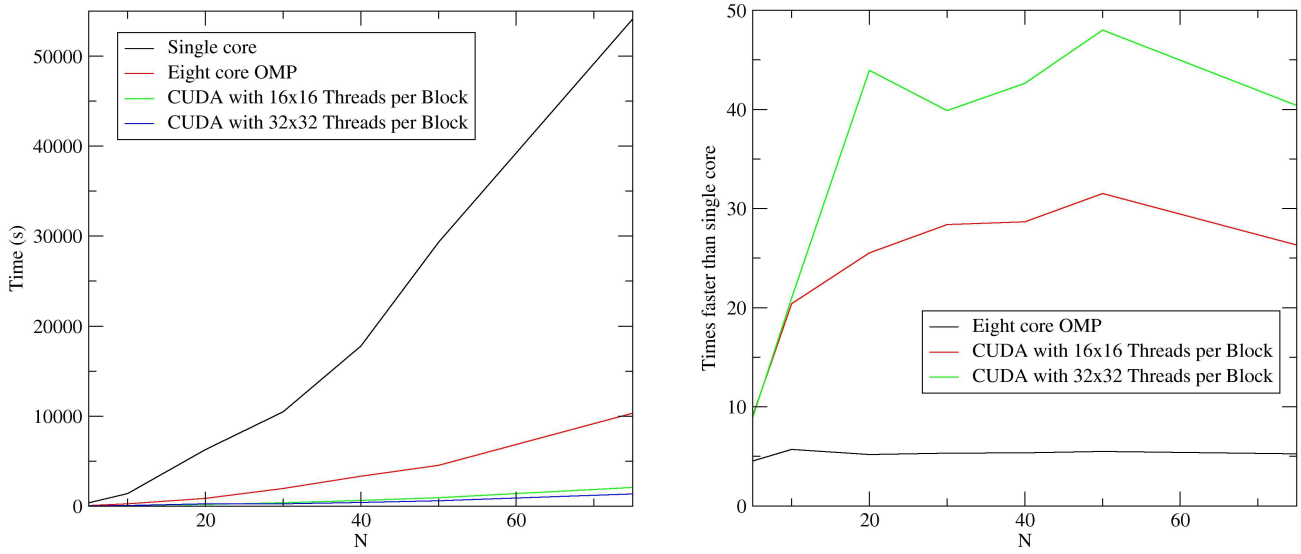

Fig 3: The left plot shows the time spent by the programs for different values of *N*, and the right plot says how much faster is the parallel programs for the different cases shown against the serial case.

We have only one difference that is worthy relate in this case to the previously, in this one it

is needed much smaller *N* for optimum performance, this is because the program naturally needs an imense amount of calculations to be done.

As a last result (Fig 4) we have the number of acess in the *CUDA* kernel per time spent to execute this last algorithm with *N=20*.

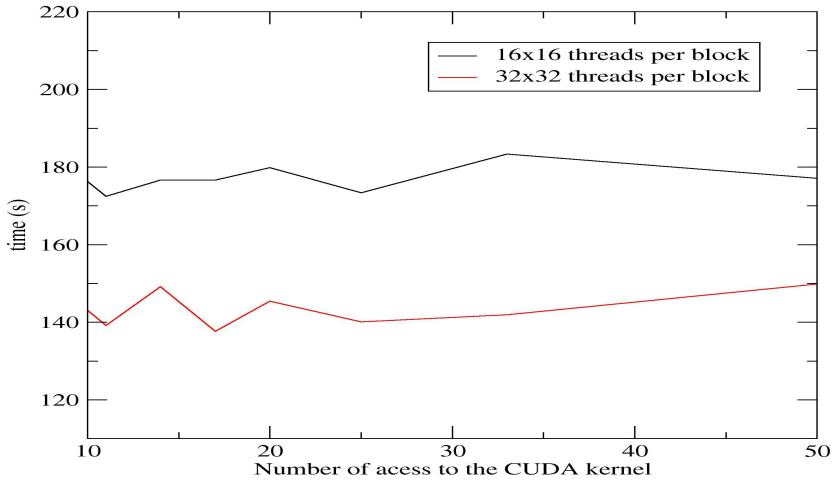

Fig 4: This plot shows the relation of time with the amount of acess needed to do some constant work

We can see in this plot that we have a strange variation of the performance with the acess to the *CUDA* kernel. Obviously we have fewer acesses to the kernel, but we have increased the number of processes that the kernel must execute. We have used *10* acesses to the *CUDA* kernel in all previsouly cases, that is a median case of performance.

Conclusion

In this work we have done some comparisons of our work that was all based in single core processing with the same work but now did with parallel computing. Was presented two possibilities to solve the problem and use considerably less computational time, *OMP* and *Nvidia CUDA*. We have seen an performance very good for both cases, but is evidently that for our case *CUDA* is much faster, something like seven or eight times faster than *OMP*, that had already been six or seven times faster than single core processing for some cases.

In pratically all cases here presented *CUDA* using 32x32 threads per block get the best performance. As future works we want to try use better the shared memory and registers for boost even more the performance of our algorithm.

Bibliography

(1) Barbara Chapman, Gabriele Jost, Ruud van der Pas: Using OpenMP, MIT Press, Cambridge, Massachusetts, London, England. (2008)

(2) David B. Kirk and Wen-mei W. Hwu: Programming Massively Parallel Processors – A Hands-on Approach, Morgan Kauffmann Publishers, Burlington, MA, USA (2010)

(3) J. P. Eckmann, S. O. Kamphorst, D. Ruelle: Recurrence plots of dynamical systems, *Europhys. Lett.*, **5**, 973-977 (1987).

(4) D. B. Vasconcelos, S. R. Lopes, R. L. Viana, J. Kurths: Spatial recurrence plots, Physical Review E, 73, 056207 (2006)

(5) N. Marwan, M. C. Romano, M. Thiel, J. Kurths: Recurrence Plots for the Analysis of Complex Systems, *Physiscs Reports*, 436 (5-6), 237-329 (2007).

(6) http://marathon.csee.usf.edu/Mammography/Database.html