A first attempt to solve the dynamics of a network of coupled oscillators in CUDA

P. P. Galuzio*[1] and S. R. Lopes[1]

[1] *Departamento de Física, Universidade Federal do Paraná, 81531-990, Curitiba, Paraná, Brazil*

Abstract

In this work, the CUDA technology has been used to solve the dynamics of a network of coupled chaotic oscillators, which is a very important problem in the field of non-linear oscillations. Furthermore, its very general structure allows one to generalize the results obtained to other more specific systems. The primary results show a significant decrease in the time of computation when compared to the serial version of the problem. Moreover the optimal number of threads per block has been found for different sizes of the problem.

1 Introduction

In several areas of physics the use of numerical simulations constitutes the most important tool for solving problems. However, the programs that represent physical systems more faithfully require a lot of computational work, leading to very large execution times. What turns to be a very important limiting factor for scientific research. In this way, the use of new technologies is very important to try to reduce the time of computation, allowing for physicists to explore more complex, and consequently more realistic and relevant problems. Among the tools available, the CUDA technology is one of the most prominent [1], due to both its high efficiency and its relative easiness of implementation.

The problem chosen to be studied in this work holds a great importance in the field of nonlinear dynamics due to its simplicity and generality, it is an one-dimensional network of coupled discrete chaotic oscillators [2, 3]. Mathematically, each oscillator is described by a state variable $x_n^{(j)}$ whose value is limited between 0 and 1, the index (j) distinguishes the N oscillators of the network. The time evolution of the system is given by:

$$x_{n+1}^{(j)} = (1-\epsilon) f(x_n^{(j)}) + \frac{\epsilon}{\eta(\alpha)} \sum_{\substack{i=1 \\ i \neq j}}^{M} \frac{1}{i^\alpha} \left[f(x_n^{(j+i)}) + f(x_n^{(j-i)}) \right], \tag{1}$$

where $M = (N-1)/2$, $\eta(\alpha) = 2\sum_{k=1}^{M} k^{-\alpha}$, and x_{n+1} stands for the variable x_n in the subsequent time $t = n+1$. This is a recursive process that is performed from an arbitrary initial value x_0, until very large values of t. The function $f(x)$ used in this work is the logistic equation, given by $f(x) = 3.6x(1-x)$. The parameter ϵ measures the intensity of the coupling and varies between the range $[0, 1]$, and $\alpha = 1.0$.

A preliminary version of a program in CUDA to solve this problem was made, and the results of time of execution are compared to a serial version of the code. Besides, the dependence of time of execution with the number of threads per block in the CUDA program has been established, allowing to determine for what value of this parameter the program is more efficient.

2 Methods

At each iteration of the system, a lot of numerical work is done, actually one has N independent summations over M different terms. The approach chosen to solve this problem in CUDA is the

*ppg06@fisica.ufpr.br

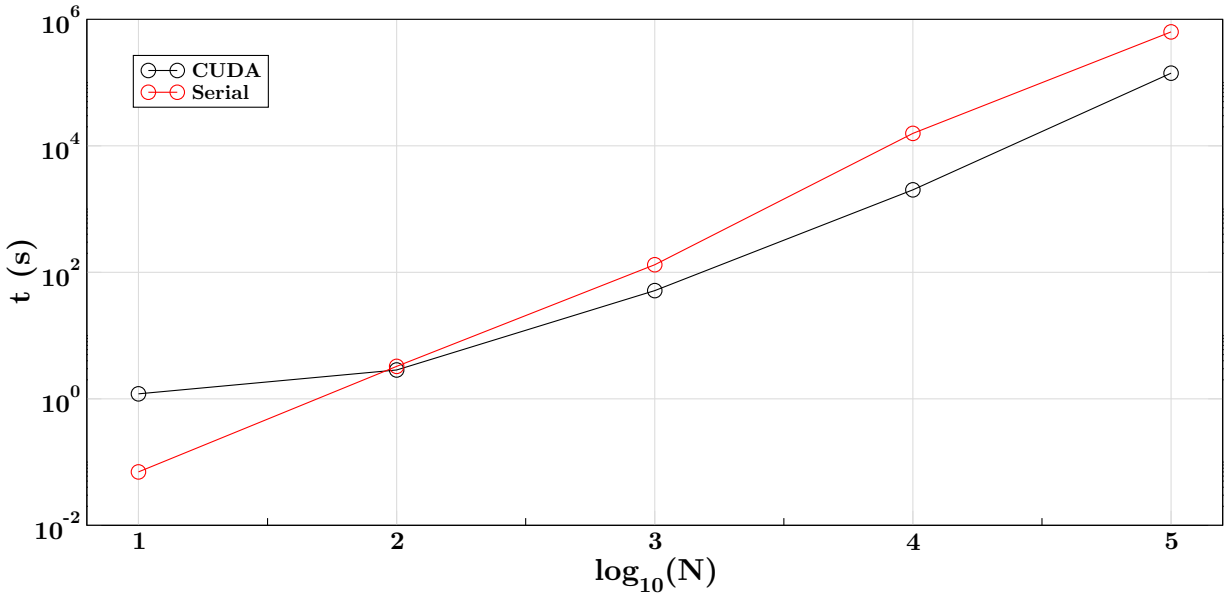

Figure 1: Time of execution (in seconds) as a function of the logarithm of the lattice size, for CUDA (black line) and the serial version (red line) of the program

simplest possible: each thread is responsible for an individual network site (j). An important limiting factor for this problem is that it requires complete synchronization among all the sites at each iteration step, which can only be assured by finishing a kernel. In pseudocode 1 the general algorithm used is represented, with a different kernel for each step that requires the problem to be synchronized.

Pseudocode 1 Iteration of the maps using CUDA, the prefix $\mathcal{K}$ in front of the statement lines represents that these instructions will be executed in parallel within different kernels.

for $n = 0 \to n_{max}$ **do**
  $\mathcal{K}1$: calculate $f(x_n^{(j)})$
  $\mathcal{K}2$: $\mathcal{S}_j \leftarrow \sum_{i=1}^{M} \frac{f(x_n^{(j+i)}) + f(x_n^{(j-i)})}{i^{\alpha}}$
  $\mathcal{K}3$: $x_{n+1}^{(j)} \leftarrow (1-\epsilon) f(x_n) + \epsilon * \mathcal{S}_j/\eta$
end for

Within each Kernel there are exactly N independent serial operations to be performed. The kernels are launched with an one-dimensional grid of blocks. The dimension of each block, i.e., the number of threads per block ($\mathcal{N}_{TpB}$) is provided by the programmer, while the number of blocks ($\mathcal{N}_B$) is calculated through the formula: $\mathcal{N}_B = N/\mathcal{N}_{TpB} + 1$. Each thread is then distinguished by an index `j`, given by:

```
j=blockIdx.x*blockDim.x+threadIdx.x;
```

where `blockIdx.x`, `blockDim.x` and `threadIdx.x` are built-in variables of CUDA. It is important to notice that `blockDim.x`= $\mathcal{N}_{TpB}$.

3 Results

The results presented here were obtained with a GeForce GTX 460 for the CUDA version, and with a Intel® Core™ i7-950 processor for the serial code.

In figure 1, the time of execution is displayed as a function of the logarithm of the lattice size N, for the CUDA and serial version of the program. It is clear that for lattice sizes $N \lesssim 100$ the serial version is more efficient, while for larger values of N the CUDA program becomes faster. This is an expected behavior, since it takes a long time to copy the relevant variables to the GPU memory. For

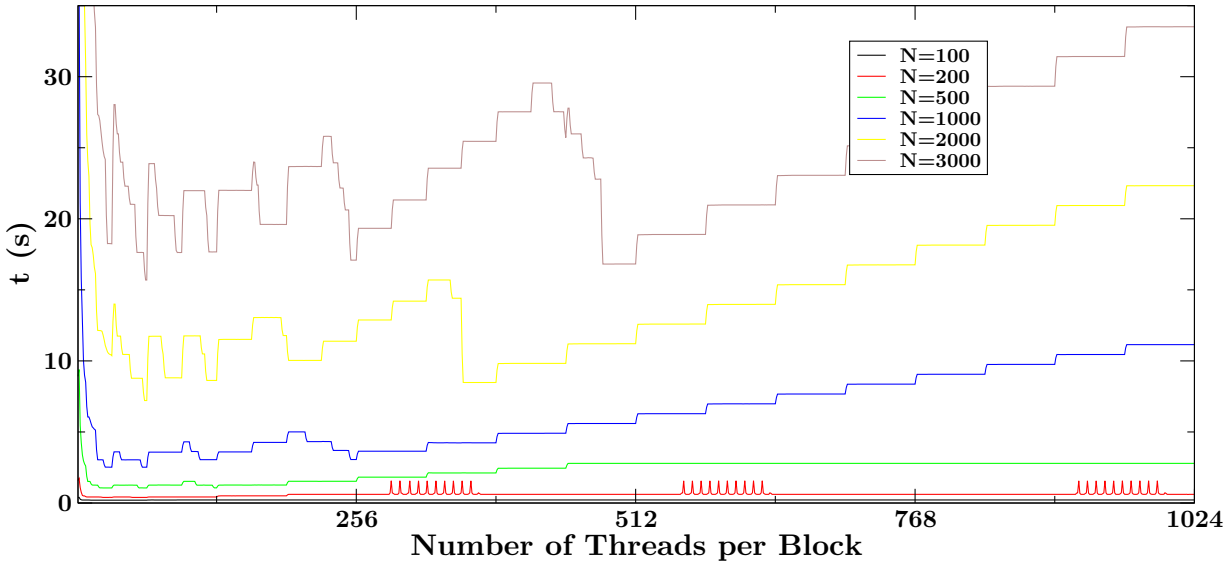

Figure 2: Time of execution (in seconds) as a function of the number of threads per block, for different lattice sizes (N)

small N, this time is comparable to the time of execution. For all the results the number of threads per block used is the maximum allowed ($\mathcal{N}_{TpB}^{max} = 1024$).

The speedup obtained is $\mathbf{S}_P \approx 8$. This is not a very high value for the speed-up. However if the extremely simplicity of the algorithm is taken into account, leading to a relatively short time of development, then this result can't be considered bad in the area of computational physics.

In the way this problem was implemented, there is still one parameter to determine: the number of threads per block. In fact the dependence of the time of execution with $\mathcal{N}_{TpB}$ is far from trivial, as shown in figure 2. Opposed to what is expected, the value for $\mathcal{N}_{TpB}$ that provides the minimum time of execution is not the maximum allowed $\mathcal{N}_{TpB}^{max}$, and it is also a different value for each lattice size.

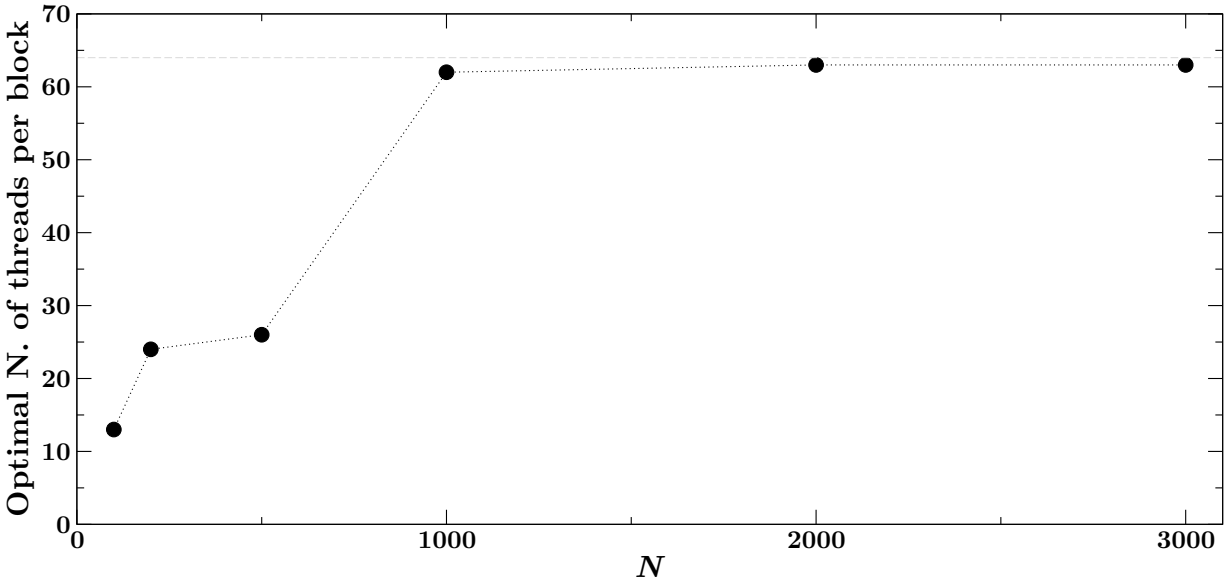

Figure 3: The optimal number of threads per block as a function of the lattice size N, the grey dashed line stand for $\mathcal{N}_{TpB} = 64$

As apparently there is no easy way to decide which value of $\mathcal{N}_{TpB}$ provides the optimal time of execution, a graph with the optimal value of $\mathcal{N}_{TpB}$ as a function of the lattice size was built (figure 3). For values of N smaller than $\mathcal{N}_{TpB}^{max}$ the optimal value of threads per block is always smaller than the lattice size. For values of $N \gtrsim \mathcal{N}_{TpB}^{max}$, then the optimal number of threads per block seem to tend asymptotically to 64.

4 Conclusions

CUDA is one of the most prominent technologies regarding parallel programing. In this work, it has been used to solve the dynamics of a network of N coupled chaotic oscillators. When compared to a serial version of the code a speedup $\mathbf{S}_P \approx 8$ was observed.

The dependence of the time of execution with the number of threads per block was also studied for different values of the lattice size, and found to be extremely non trivial. However, it was possible to find an optimal value of threads per block as a function of N, and for large enough lattice sizes, $\mathcal{N}_{TpB} = 64$ seems to give the smallest time of execution.

The problem chosen to be implemented is of great importance in the field of nonlinear dynamics, in fact it can be easily generalized for more general systems, with applications from physics to neurosciences. In this way, the speedup obtained, despite small when compared to other more complex parallel algorithms, is still very satisfactory, since it can be very easily applied to a very large range of scientific problems.

References

[1] Nvidia cuda programming guide version 4.0. *Nvidia Corporation* (2011).

[2] dos Santos, A. M., Viana, R. L., Lopes, S. R., de S. Pinto, S. E., Batista, A. M. Collective behavior in coupled chaotic map lattices with random perturbations. *Physica A 387* (2008), 1655–1668.

[3] Szmoski, R. M., Pinto, S. E. D. S., Kan, M. T. V., Batista, A. M., Viana, R. L., Lopes, S. R. Synchronization and suppression of chaos in non-locally coupled map lattices. *PRAMANA - Journal of Physics 73*, 6 (2009), 999–1009.