MULTI-CORE AND MANY-CORE SPMD PARALLEL ALGORITHMS FOR CONSTRUCTION OF BASINS OF ATTRACTION

Marcos Silveira, Paulo J.P. Gonçalves, José M. Balthazar
São Paulo State University – UNESP, Faculty of Engineering, Bauru, Brazil
e-mail: m.silveira@feb.unesp.br

Construction of basins of attraction, used for the analysis of nonlinear dynamical systems which present multistability, are computationally very expensive. Because of the long runtime needed, in many cases, the construction of basins does not have any practical use. Numerical time integration is currently the bottleneck of algorithms used for the construction of such basins. The integrations related to each set of initial conditions are independent of each other. The assignment of each integration to a separate thread seems very attractive, and parallel algorithms which use this approach to construct the basins are presented here. Two versions are considered, one for multi-core and another for many-core architectures, both based on a SPMD approach. The algorithm is tested on three systems, the classic nonlinear Duffing system, a non-ideal system exhibiting the Sommerfeld effect and an immunodynamics system. The results for all examples demonstrate the versatility of the proposed parallel algorithm, showing that the multi-core parallel algorithm using MPI has nearly an ideal speedup and efficiency.

Keywords: basins of attraction, MPI, CUDA, Duffing equation, Sommerfeld effect, immunodynamics

1. Introduction

Usual tools for numerical analysis of nonlinear dynamical systems which present multistability and chaotic behaviour, such as construction of bifurcation diagrams, construction and visualisation of basins of attraction, and construction of stability maps, are essential for a thorough understanding of the underlying phenomena (Soliman and Thompson, 1989; Thompson and Stewart, 2002; Rega and Lenci, 2005). However, they are computationally very expensive and, sometimes, these tools may not be of any practical use.

Improvements of processor core raw performance have been decreasing for over a decade, and advances in computer hardware and software have been aimed towards parallel computation (Pacheco, 2011; Brodtkorb et al., 2013). A common technique for parallelisation is the domain partitioning as in the parallel algorithm for mixed time integration for large-scale structural dynamic analysis with nonlinearities, Rao (2002), which is based on the serial algorithm presented by Belytschko et al. (1979). Other examples are single step implicit algorithms (Modak and Sotelino, 2002), population dynamics problems (Ramachandranmurthi et al., 1997; Michaels and Zubik-Kowal, 2012) and molecular dynamics (Murty and Okunbor, 1999; Trobec et al., 1993; Aktulga et al., 2012). In other types of problems, a modular approach can be used for systems that can be subdivided, such as in vehicle dynamics (Agrawal et al., 1992), large articulated robotic systems (Bhalerao et al., 2012), and contact dynamics (Koziara and Bičanić, 2011). In these cases, however, the communication overhead limits the efficiency and overall gains of parallelisation.

Currently, the time integration is the bottleneck of algorithms used for simulation and dynamical stability analysis of nonlinear dynamical systems which present multistability. Although
parallel algorithms for time integration have been sought (Sommeijer, 1993; Cong, 1996), their performance in large scale simulations is usually limited, as they suffer from low scalability and large overhead (Bendtsen, 1996).

The construction of bifurcation diagrams, basins of attraction, stability maps and sensitivity analysis involve numerous time integrations for different combination of values of parameters and initial conditions. In analysis of dynamical integrity (Soliman and Thompson, 1989; Rega and Lenci, 2005; McDonald et al., 1985; Lenci et al., 2013), the total run-time is drastically increased, as the construction of basins of attraction is repeated many times. For a specific case of construction of basins, the integrations related to each set of initial conditions are independent of each other. This way, the assignment of each integration to a separate thread seems very attractive. Each process is computationally intensive, although the memory requirement is relatively low.

In view of the necessity to find faster ways to construct basins of attraction and to perform dynamical stability analysis of nonlinear dynamical systems fully utilising the available computing power, parallel algorithms for construction of basins of attraction are presented here. Two versions are considered, one for multi-core and another for many-core architectures, both based on a SPMD (single program multiple data) approach. The rest of the paper is organised as follows: Section 2 introduces the generic serial algorithm for construction of basins of attraction. Sections 3 and 4 present the multi-core algorithm using MPI and the many-core algorithm using CUDA. Sections 5-7 contain results based on the classical Duffing oscillator, a non-ideal oscillator and an immunodynamics model. These examples of application are used to verify the accuracy, scalability and efficiency while demonstrating the versatility of the proposed parallel algorithms.

2. Tasks for the construction of basins of attraction

The construction of basins of attraction involves finding all attractors of a dynamical system and identifying regions in the state space corresponding to initial conditions that lead to each attractor. Details concerning the search and classification of attractors among static or dynamic equilibria are not the focus of this work, and analytical and numerical methods can be found elsewhere (Thompson and Stewart, 2002; Parker and Chua, 1992; Nayfeh and Balachandran, 1995). Once all the attractors are known, the dynamical equations can be numerically evaluated with one set of initial conditions to determine to which attractor this set leads. This is repeated for all initial conditions, identifying the basins of attraction.

An algorithm to determine the basins consists in cycling search through all desired initial points, integrating a trajectory from each point to determine to which attractor it leads, and grouping the points that belong to the same attractor. As a consequence of the uniqueness of solutions, all points touched by a trajectory can be marked as pertaining to the same attractor. This eliminates the necessity to evaluate a large quantity of grid points. Therefore, as a grid point is selected, it must be checked if this point already belongs, or is close with a certain degree of accuracy, to a trajectory. A serial algorithm to perform such tasks is depicted in Fig. 1a.

3. Multi-core parallel algorithm using MPI

From the previous Section, it can be seen that the detection of points belonging to calculated trajectories can significantly reduce the run-time. However, this detection can lead to unacceptable communication overhead. In order to minimise communication between threads, information of points already hit and of points hit by the current trajectory are sent at the beginning and end of the process. In a region of the space state with an attractor at its center, trajectories tend
Fig. 1. Serial algorithm used for construction of basins of attraction (a) and the multi-core parallel algorithm using MPI (b). The processes in blue are executed by the master thread, the processes in orange are executed in parallel by all other threads.

The serial algorithm for computation of the basins of attraction from Section 2 was modified for use in x86 multi-core processors using MPI (message passing interface), as shown in Fig. 1b. MPI was chosen as it is one of the most used frameworks for parallel scientific computing. The master thread is responsible for broadcasting all parameters to the slave threads, for scattering each subgrid of initial conditions to the respective thread, and for collecting and consolidating the results. As the threads do not communicate with each other, there is no concurrency problem.
When the number of cores $c$ is much lower than the number of grid points $p_g$, the grid is subdivided in fewer regions. The number of points in each region is given by

$$p_{ra} = \frac{p_g}{c} \quad (3.1)$$

For an increasing number of cores, there is a larger number of regions, although they become smaller, and the amount of checked, and possibly eliminated, grid points diminishes. Usually, one uses from 30,000 to 300,000 grid points, which can be 3 to 4 orders of magnitude greater than the available number of x86 cores. This ensures a good load balancing and that the detection of used points eliminates a satisfactory portion of points. The maximum number of eliminated grid points $p_e$ is given by

$$p_e^{\text{max}} = c(p_r - 1) = p_g - c \quad (3.2)$$

The serial run-time $T_s$ is proportional to the number of calculated trajectories. Note that the minimum $T_s$ will be the time of evaluation of one trajectory, and this happens when $p_e = p_e^{\text{max}}$. The maximum $T_s$ will be the time of evaluation of $p_g$ trajectories ($p_e = 0$). These relations are given by

$$T_s \propto (p_g - p_e) \quad T_p \propto \left( p_r - \frac{p_e}{c} \right)$$

Although the minimum number of evaluated trajectories equals $c$ in the parallel case, the minimum $T_p$ ($p_e = p_e^{\text{max}}$) will be the time of evaluation of one trajectory, and the maximum $T_p$ ($p_e = 0$) corresponds to $p_r$ trajectories. The time for communication must be considered in these relations. When the number of eliminated grid points tends to $p_e^{\text{max}}$, the run-time of the multi-core parallel algorithm tends to be similar to the serial algorithm. When the number of eliminated grid points tends to 0, the run-time of the multi-core parallel algorithm tends to be $c$ times faster than the serial algorithm.

### 4. Many-core parallel algorithm using CUDA

The use of GPUs for general computing in scientific problems may be very advantageous when the problem can be divided in several light processes. In the problem of construction of basins of attraction, the lightest independent process is the time integration of trajectories. Using the CUDA (compute unified device architecture) platform, the algorithm for computing basins of attraction was modified in order to assign the integration of each trajectory to one computing core of the GPU. The CUDA platform was chosen as it is one of the most frequently used technologies for GPU computing. Within the CUDA programming model, threads are grouped in blocks, and the blocks are grouped in grids. The number of threads in a block and the number of blocks in a grid depend on the device being used.

The modified algorithm (Fig. 2) consists of dividing the total number of grid points $p_g$ in $b$ blocks, with $t$ being the number of available threads per block. All blocks are cycled, selecting a subset of grid points, evaluating trajectories from each point, and consolidating the results from the subset. The CPU is the master thread and is responsible for initialising the simulation, selecting the subsets of points, checking for used points, assigning the points to the GPU and consolidating the results. The processes executed in the GPU consist of the actual numerical integrations of each point. If one or more points are detected as used, a new subset must be selected in order to have exactly one point for each thread in the block, with the objective of achieving a good load balancing.
The number of blocks and the maximum number of points which can be eliminated are given by

\[ b = \frac{pg}{t} \quad p_{e}^{\text{max}} = (b - 1)t = pg - t \]

in which \( t \) is the number of threads per block, \( pg \) is the number of grid points. The check for used points is only made at the beginning and the end of the process. The equivalent of relation (3.3) for the run-time of the many-core parallel algorithm is given by

\[ T_p \propto \frac{pg}{t} - \left\lfloor \frac{pg}{b} \right\rfloor \]

in which \( \lfloor \cdot \rfloor \) is the notation for the floor function. The minimum number of evaluated trajectories equals to \( b \) in this case. The minimum \( T_p \) (\( p_e = p_{e}^{\text{max}} \)) will be the time of integration of one
trajectory, and the maximum $T_p (p_e = 0)$ corresponds to $b$ trajectories. When $p_e$ tends to $p_e^{max}$, the run-time of the many-core algorithm tends to be similar to the serial algorithm, and when $p_e$ tends to 0, the run-time of the many-core parallel algorithm tends to be $b$ times faster than the serial algorithm.

5. Multi-core and many-core algorithms used on the Duffing oscillator

In order to verify the accuracy, scalability and efficiency of the proposed parallel algorithm, the Duffing equation is used in this Section as an example of application, given by

$$m\ddot{x} + c\dot{x} + k_1 x + k_2 x^3 = B \cos(\omega t)$$  \hspace{1cm} (5.1)

in which $m$ is mass, $k_1$ and $k_2$ are linear and nonlinear coefficients of stiffness, $c$ is damping coefficient, $B$ and $\omega$ are amplitude and frequency of the excitation force. The values of the parameters used here are $m = 2.56 \text{kg}$, $k_1 = 1 \text{N/m}$, $k_2 = 0.05 \text{N/m}^3$, $c = 0.32 \text{Ns/m}$, $B = 2.5 \text{N}$ and $\omega = 1 \text{rad/s}$, which result in the existence of two co-existing limit-cycles, one with a large and another with small amplitude (Thompson and Stewart, 2002). Figure 3 shows the basins of the two attractors.

![Fig. 3. Basins of attraction of the Duffing oscillator, Eq. (5.1). The red dot is the small amplitude limit-cycle, with its basin in green, and the yellow dot is the large amplitude limit-cycle, with its basin in blue. The attractors are represented by their Poincaré sections.](image)

The multi-core algorithm was executed on a x86 processor with 16 physical cores. The many-core algorithm was executed on a GPU with 448 CUDA cores, limited to a maximum of 512 threads per block. All timings shown are the arithmetic mean of at least 5 runs. The desired resolution of the figure of basins of attraction is from $256 \times 256$ to $512 \times 512$ points, although lower resolution basins were calculated for comparison of different problem sizes. The numerical algorithm used to calculate the time response is the 4-th order Runge-Kutta method with a fixed step. The fixed step is used to enable direct comparisons between different architectures.

The performance is measured by the run-time, speedup and efficiency. The run-time of an ideal parallel implementation $T_p$, the speedup $S$ and efficiency $E$ are given by

$$T_p^{\text{ideal}} = \frac{T_s}{c} \hspace{1cm} S = \frac{T_s}{T_p} \hspace{1cm} E = \frac{S}{c} = \frac{T_s}{cT_p}$$  \hspace{1cm} (5.2)

This is only reached when no overhead is incurred by parallelisation. In an ideal parallel implementation, $S_{\text{ideal}} = c$.

Remaining serial sections and inter-thread communication overhead are the contributing factors for lower efficiency of real implementations. Note that the speedup and efficiency are
Multi-core and many-core SPMD parallel algorithms...

be calculated based on the reference $T_s$, which can be obtained by the fastest known algorithm to solve the problem, or by the parallel algorithm run on a single core (adopted here). The reference $T_s$ for the many-core algorithm is taken from the run on a single CUDA core.

Figure 4a shows the run-time as a function of the number of used cores. The number of trajectories is given as the size of the grid of initial conditions. Note that there is a rapid decrease in the run-time and fast saturation. However, incremental gains continue to be observed. Figure 4b shows the run-time as a function of the number of calculated trajectories. An increase in the run-time is linearly proportional to the number of calculated trajectories, indicating excellent scalability. Figure 4c shows that the speedup is very close to the ideal line (in red). Figure 4d confirms that the efficiency is nearly ideal. A small variation is seen for different grid sizes, indicating that this is not a limiting factor. Besides that, the excellent scalability shown demonstrates that the techniques applied to the algorithm are appropriate for the problem being studied.

![Fig. 4. Run-time as a function of the number of used cores (a) and as a function of the number of calculated trajectories (b), speedup (c) and efficiency (d) as a function of the number of applied cores using the multi-core algorithm with MPI for system (5.1).](image)

The results of computation of basins of attraction using the many-core parallel algorithm show that GPU technology also provides good time gains. Figure 5a shows a decrease in the run-time as the number of used cores increases. Again, the saturation tendency of the run-time is observed. For example, using a grid of initial conditions of $128 \times 128$ points, the difference between using 128 or 256 cores is less than 1%. Hence, it is possible to identify an optimal number of cores to be used for a given number of trajectories. Figure 5b shows that the run-time is also linearly proportional to the number of calculated trajectories. In terms of the speedup and efficiency (Fig. 5), the many-core algorithm is shown to be inferior to the multi-core version. Although there are substantial run-time gains compared to the performance on
a single core, they are obtained via the much larger number of used cores, resulting in a low efficiency. For direct comparison, the run-time of a $256 \times 256$ grid using the multi-core algorithm is 1.87 faster than the many-core version.

![Fig. 5. Run-time as a function of the number of used cores (a) and as a function of the number of calculated trajectories (b), speedup (c) and efficiency (d) as a function of the number of applied cores using the many-core algorithm with CUDA for system (5.1)](image)

6. Application to a non-ideal system exhibiting the Sommerfeld effect

A simplified spring-mass-damper oscillator model with a non-ideal excitation source was defined by Gonçalves et al. (2014, 2015), Balthazar et al. (2018), in which a cart moves horizontally without friction, with a non-ideal DC motor fixed to it. The motor has an unbalanced mass that provides excitation to the system. It was found that the Sommerfeld effect depends on some system parameters and the motor operational procedures. The dynamic equations for motion of the cart and of the motor shaft are written as

$$\ddot{x}(M + m_r) + kx + c\dot{x} = m_r r (\dot{\phi}^2 \cos \phi + \ddot{\phi} \sin \phi)$$

$$\dot{\phi}(J_0 + m_r r^2) = \ddot{x} m_r \sin \phi + M_0 \left(1 - \frac{\dot{\phi}}{\Omega_0}\right)$$

(6.1)

The values used for the parameters are given in Gonçalves et al. (2014). Two distinct responses of the system are related to two co-existent period-1 limit-cycle attractors, with a large amplitude limit-cycle related to the resonance capture. With low $\Omega_0$, only the large amplitude limit-cycle exists. This indicates the presence of the Sommerfeld effect, as the motor in this
configuration is not able to provide enough energy to the system to escape from the resonance. Increasing $\Omega_0$, two limit-cycle attractors co-exist, with the state space divided in two regions: one with initial conditions leading to the large amplitude limit-cycle (Sommerfeld effect), and another leading to the small amplitude limit-cycle (Fig. 6).

Fig. 6. Basins of attraction of the non-ideal oscillator, Eq. (6.1), using $\Omega_0 = 1.625\omega_0$ (a), $\Omega_0 = 2.167\omega_0$ (b) and $\Omega_0 = 2.438\omega_0$ (c). The red dot represents the small amplitude limit-cycle, with its basin in green, and the yellow dot represents the large amplitude limit-cycle, with its basin in blue.

Figure 7 shows that the run-time for this system is considerably longer than for the Duffing system, as this non-ideal oscillator takes longer to settle into steady-state behaviour. The algorithm provides the same level of time gains, the speedup remains very close to the ideal line, and the efficiency is maintained closer to ideal than in the Duffing system. The results of
computation of basins of attraction using the many-core parallel algorithm with CUDA present the same tendencies as shown in the previous Section with reasonable speedups at the cost of lower efficiency. However, the run-time is more than 2.1 times slower when compared to the multi-core version.

7. Application to an immunodynamics system

Lang and Li (2012) studied the presence of different types of equilibria on a model of Human T cell leukemia virus type I (HTLV-I) infection. The model describes uninfected CD4+ T cells \((x)\), infected CD4+ T cells \((y)\) and CTL \((z)\), and is given by

\[
\begin{align*}
\dot{x} &= \lambda - \beta xy - \mu_1 x \\
\dot{y} &= \sigma \beta xy - \gamma yz - \mu_2 y \\
\dot{z} &= \nu y \frac{z^n}{z^n + a} - \mu_3 z \tag{7.1}
\end{align*}
\]

This system has three types of equilibria (Lang and Li, 2012), with the first related to an organism without infection, the second related to an asymptomatic carrier state, and the third related to the development of HAM/TSP. Using the parameters related to CD4+ T cell production rate \(\lambda\) and CTL production rate \(\nu\) as bifurcation parameters, the system undergoes Hopf bifurcations, and regions with co-existing attractors arise. In the context of immune dynamics, the basins represent biological conditions at the time of infection which will result in one specific manifestation of a disease and, therefore, indicates the long-term health state of an individual. With the base set of parameters given in Lang and Li (2012), the system presents one equilibrium point, representing evolution of the infection in which the levels of healthy and infected cells reach fixed levels, and a limit-cycle, representing a condition in which the levels of cells oscillate over time.

The two attractors are shown in Fig. 8. On the left, the equilibrium point attractor is in yellow, and the limit-cycle attractor is in red. The blue region represents the basin of the equilibrium point, and the green region is the basin of the limit-cycle. On the right, the CTL effectiveness is decreased. As a result, the red attractor becomes an equilibrium point, and its basin (green region) is enlarged. The blue region is now the basin of an equilibrium point at another region of the state space, not shown in this figure.

![Fig. 8. Co-existing attractors and respective basins of model (7.1). With base parameters (a), there is one equilibrium point (yellow), with its basin in blue and one limit-cycle (red), with its basin in green. With \(\gamma = 1\) (b), there is one equilibrium point (red).](image)
Figure 9 shows that the run-time is consistently lower using the multi-core algorithm incorporating MPI. Intra-function timings show that the lack of trigonometric functions contribute to this. The algorithm provides the same level of time gains as observed in the previous Sections. The speedup and efficiency are less close to ideal. The run-time of the many-core algorithm with CUDA is more than 1.7 times slower when compared to the multi-core version.

8. Concluding remarks

Examples of application of the algorithm to the classic nonlinear Duffing system, to a non-ideal system exhibiting the Sommerfeld effect and to an immunodynamics system are shown. The results for all examples indicate that the multi-core algorithm using MPI has nearly an ideal speedup and efficiency, and perform consistently on different types of systems.

The use of GPUs for general computing in scientific problems may be very advantageous when the problem can be divided in several light processes. Although each integration process has a very small memory footprint, the complexity of the dynamical system being analysed may result in a heavy processing load, which hinders the advantage of using a GPU. In processing intensive cases, the execution in the CPU is usually more advantageous, even considering a lower number of computing cores compared to GPUs. This happens because of different purposes and architectures of GPUs (massive number of light threads) and CPUs (small number of intensive threads). Consequently, the use of a many-core GPU proved to be less satisfactory for constructing basins of attraction, with the multi-core algorithm run in the CPU shown to be faster, more efficient and more scalable.
Acknowledgements
The authors thank FAPESP (grant #2012/19774-3) and CNPq (grant #448560/2014-3) for financial support during this project.

References
14. McDonald S.W., Grebogi C., Ott E., Yorke J.A., 1985, Fractal basin boundaries, Physica D, 17, 125153, DOI: 10.1016/0167-2789(85)90001-6


*Manuscript received March 6, 2018; accepted for print June 25, 2019*