Parallel tools for the bifurcation analysis of large-scale chemically reactive
dynamical systems

Gaetano Continillo, Artur Grabski, Erasmo Mancusi, Lucia Russo

Abstract

In this work we propose a set of tools for the parallel application of pseudo-arclength continuation to a class of systems for which the right hand side can be properly represented by a time numerically calculated evolution operator. For example, the reverse flow reactor and the reactors network with periodically switched inlet and outlet sections belong to this class of system. To conduct a dynamical analysis of these systems when the key parameters are changed, it is necessary to compute the eigenvalues of the Jacobian matrix many times. Since the Jacobian can only be obtained numerically, and this in turn takes away really significant computational power, running this operation in parallel saves real time of computation. Examples, solution lines and performance diagrams for selected systems are presented and discussed.

1. Introduction

There is an increasing interest in chemical engineering towards the application of bifurcation theory arsenal in order to obtain a complete dynamical characterization of the mathematical model of a system (for example a reactor plant). In order to properly design and control chemical reactors, it is necessary to accurately describe all the regime conditions when relevant design and operation parameters are changed. More generally, mathematical models of chemically reactive systems could exhibit complex regime transitions marked by catastrophic bifurcations (sudden changes in temperature and/or concentrations) (Balakotaiah, Dommeti, & Gupta, 1999; Mancusi, Merola, Crescitelli, & Maffettone, 2000). Bifurcation analysis and parametric continuation are the tools of choice in investigating dynamic features of non-linear systems and, particularly, in identifying multi-stability regions (e.g. Seydel, 1988). Numerical continuation is the process of solving systems of nonlinear equations \( F(x,p) = 0 \) (or \( x_{n+1} = F(x_n,p) = 0 \)) for various values of a real parameter \( p \). Such a technique requires at each step the computation of eigenvalues of the Jacobian matrix of the system in order to define the stability of the solution regime and the possible appearance of a bifurcation (e.g. Kuznetsov, 1998). The numerical computation of these eigenvalues is one of the main tasks in such continuation algorithms. However, standard and popular codes for automatic continuation, such as AUTO (Doedel et al., 1997, 2000), CONTENT (Kuznetsov et al., 1996), CONT (Schreiber & Marek, 1991), BIF-PACK (Seydel & Hlaváček, 1987) are generally unsuitable for large scale systems. Moreover, these software packages work efficiently only if the mathematical model of the system has an analytic expression for the Jacobian matrix.

Generally, the numerical computation of the eigenvalues is the slowest task in the execution of the continuation algorithm and, when the system under study has no analytic expression of the Jacobian matrix, it must be computed numerically and the whole process becomes even more time consuming. It is perhaps worth mentioning that there exist some methods of automatic differentiation and relevant software tools such as ADIFOR and ADIC (Bischof et al., 1992; Bischof, Roh, & Mauer-Oats, 1997). Main purpose of this software is to extract a computer code (usually in high-level language code statements such as Fortran and/or C) for the evaluation of the analytical form of the derivatives of a vector function expressed. This obviously assumes that an analytical form of the function exists. It should be made clear that we rather address our attention to systems for which an analytical form of the function does not exist and also the execution of the numerical derivatives requires huge computational effort.
Nomenclature

Symbols

- \( n \): order of reaction, dimension of the system, number of species
- \( m \): nr. of estimations of Jacobian matrix
- \( ns \): number of slaves
- \( P_{EM} = \frac{wL}{D} \): Pelet number of mass balance
- \( P_{ES} = \frac{wL_{DC}C_k}{\lambda_k} \): Pelet number of solid phase
- \( P_{FT} = \frac{wL_{DC}C_k}{\lambda_k} \): Pelet number of fluid phase
- \( p \): number of processors
- \( r = k^2 \text{A}^2 \): reaction speed (kmol/(m² s))
- \( R \): gas constant (kJ/(mol K))
- \( \text{RHS} \): right hand side of equation
- \( s \): serial CPU time/parallel CPU time: speedup
- \( St = \frac{Ak_k}{(\rho C_p^F)} \): Stanton number
- \( t \): time (s)
- \( T \): temperature (K)
- \( V \): reactor volume (m³)
- \( \mathbf{u} \): vector of variables
- \( w \): velocity (m/s)
- \( q \): fraction of parallel code
- \( \mathbf{x} \): vector of variables
- \( z \): spatial coordinate (m)

Greek symbols

- \( \alpha \): conversion
- \( \beta = (-\Delta H)C_{k0}/(T_0\rho C_p) \): dimensionless adiabatic temperature rise
- \( \delta = A_k k_{pp}/(\rho C_p^F) \): dimensionless heat transfer coefficient
- \( \gamma = E/(RT_0) \): dimensionless activation energy
- \( \varepsilon \): porosity
- \( \psi_i(x) \): evolution operator
- \( \lambda \): thermal conductivity (W/(mK))
- \( \theta \): dimensionless temperature
- \( \xi_R \): dimensionless spatial coordinate in reactor
- \( \xi_k \): dimensionless spatial coordinate in heat exchanger
- \( \rho \): density (kg/m³)
- \( \sigma = (1 - \varepsilon)\rho C_p^F/(\rho C_p) \): dimensionless heat volume of solid phase
- \( \sigma_p = V_p/V_R \): dimensionless volume of products pipe of heat exchanger
- \( \sigma_F = V_F/V_R \): dimensionless volume of feed pipe of heat exchanger

Subscripts and superscripts

- \( B \): backward perturbation
- \( E \): exchanger, end value
- \( F \): feed, forward perturbation
- \( G \): fluid (gas)
- \( i \): number of species
- \( H \): heat
- \( M \): mass
- \( P \): product
- \( R \): reactor
- \( S, K \): catalyst, solid, start
- \( k \): number of iterated point (maps)

Parallel computation is the most promising approach to reduce the computation time in complex numerical problems. Presence of independent computation tasks that can be conducted in parallel is not the only condition to obtain successful implementation of parallel algorithms. Numerical problems in chemical engineering can often be cast in such a way as to benefit from parallel implementations of the algorithms. Simulation of distributed systems is a typical field of application, when Internal Domain Decomposition techniques are employed to generate decoupled sets of differential equations that can be integrated in parallel numerically (Kumar & Mazumder, 2010). Also, large scale parameter estimation problems have been tackled by decomposition methods (Liu & Wang, 2009) to convert large sets of ordinary differential equations into decoupled algebraic equations. Their method not only reduces computation time, but also generates a set of uncoupled algebraic equations that can therefore be efficiently solved in parallel. A similar property can be obtained in large scale optimization processes, when multiple shooting techniques are employed for the relevant DAE integrations (Leineweber, Schäfer, Boch, & Schlöder, 2003): computations in each interval are completely decoupled and hence can be performed in parallel. Parallel computations may be also efficiently applied when PDE equations governing macroscopic distributed processes are not known, whereas evolution rules at the microscopic/mesoscale level are well known. Indeed, in such cases short "bursts" of microscopic simulations may be run in parallel while system level tasks, like stability analysis and bifurcation calculations, are performed with the so-called "coarse timestep" approach without ever obtaining the equations in closed form (Armaou, Siettos, & Kevrekidis, 2004; Kevrekidis & Samaey, 2009; Siettos, Armaou, Makeev, & Kevrekidis, 2003). It should be noted that, even if such method might be able to perform computational tasks otherwise unaffordable, in the case of spatially distributed macroscopic processes the "coarse time stepper" may be still very high dimensional. In this framework, our approach may be used to perform efficiently stability and bifurcation analysis at a macroscopic level in a parallel computation environment when a time simulator is used as "black box".

Parallel computation can also change comparative performance of different algorithms which can be used as alternatives to perform the same task. An example can be found in the numerical solution of boundary value problems, for which orthogonal collocation methods are memory intensive whereas shooting methods are computationally intensive. Parallel shooting under proper circumstances may make solution of large scale boundary value problems affordable.

To obtain convenient parallel speedup, the parallel fraction of the whole algorithm must be very high (Amdahl, 1967). For example, when bi analysis requires extensive numerical work spent to integrate functions for independently changed integration bounds, and this task is dominant in the computing time, we are in a favorable condition for best performance of parallel algorithms. This condition is met for a wide class of systems, which include – but are not limited to – systems for which the Jacobian matrix must be computed numerically via repeated independent numerical integrations. Several examples are found in the literature. Discontinuous periodically forced reactors such as Reversed-Flow Reactors (RFR) and Reactor Networks (RN) were analyzed (Mancusi, Russo, Altimari, Maffettone, & Crescitelli, 2007; Mancusi, Russo, Altimari, & Crescitelli, 2010; Mancusi, Russo, Altimari, & Crescitelli, 2011; Russo, Altimari, Mancusi, Maffettone, & Crescitelli, 2006; Russo, Mancusi, Maffettone, & Crescitelli, 2002) by conducting a bifurcation analysis on a properly constructed discrete map, based on the system’s Poincaré map. This map is not available in analytic form and thus it must be computed numerically. As a consequence, there is no analytical expression for the Jacobian matrix. Mancusi, Russo, and Continillo (2003) pointed out that most of the computation time is spent during repeated time integrations of the map. By the way, they successfully used their method for continuation of various periodically forced systems for the construction of solution diagrams. Jacobsen and Berezowski (1998), in order to study the dynamic behavior of ideal homogeneous reactors with...
recycle, showed that the system described by a set of PDE equations can be efficiently approximated by a discrete map. Also in this case the discrete map is not available in analytical form and then it must be computed numerically. Analysis of static bifurcation is then possible.

In this work we implement and conduct parallel bifurcation analysis of systems with require extensive independent numerical integration work. The method is applied by means of versions of AUTO (Doedel et al., 1997, 2000) modified by the authors to operate in parallel. We report implementation details and results of two different applications, which are typical examples of chemical engineering problems, namely:

- Bifurcation analysis of a Reverse-Flow Reactor for the catalytic combustion of lean gas mixture;
- bifurcation analysis of a network of connected catalytic reactors with periodically switched inlet and outlet sections.

In the two cases the continuation software is applied to a suitable discrete system (Poincaré map). Results are presented and discussed which include solution diagrams and evaluation of parallel performance.

The article is structured as follows. First, two categories of systems are described for which bifurcation analysis may benefit of a parallel approach. Then, a key task in our problems with high parallelism is described, i.e. the numerical computation of the Jacobian matrix for systems requiring extensive independent numerical integrations. Then, application examples are illustrated in which parallel algorithms are implemented, essentially highly computationally intensive parameter continuation of model problems. Finally, performance results are collected and discussed.

2. Two kinds of systems involving many independent numerical integrations

2.1. Study of discontinuous periodically forced systems

Discontinuous periodically forced systems, like those arising from modeling periodically forced reactors, can be formulated in an abstract form as:

$$\mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{p}, g(T))$$

(1)

where $g(T)$ represents the discontinuous forcing with period $T$, $\mathbf{x}$ is the state vector and $\mathbf{p}$ the vector parameter. The continuous time system can be studied via its Poincaré map. The details of this approach are described in Russo et al. (2002). The authors adopt AUTO (Doedel et al., 1997) to conduct the bifurcation analysis. AUTO can trace the fixed point locus of a discrete map, compute branches of stable and unstable fixed points for a discrete system, and compute the Floquet multipliers that determine the stability along these branches. Standard use of AUTO requires that the user supplies an analytic expression of the discrete-time system. In their case, an analytic expression for the map is unavailable, thus they resort to numerical evaluation of the map. The technique consists of an interaction between AUTO (or any equivalent continuation software) and an ODE solver which efficiently evaluates the map. More explicitly, for the continuous-time forced system reported in Eq. (1), let the Poincaré map $\mathbf{P}$ be:

$$\mathbf{x}_{k+1} = \mathbf{P}(\mathbf{x}_k, \mathbf{p})$$

(2)

The map must be evaluated numerically. For periodically forced systems the Poincaré map is easily constructed by sampling the time trajectory at each period $T$. Then, starting from $\mathbf{x}_0$, the equations of the continuous-time system are integrated over a time period $T$ and the result can be assumed as initial condition of a new time integration of the equations of the continuous-time system, again on a time interval $T$. Numerically, the continuation of the fixed points of the map is conducted with calls from the AUTO main routine to an external integrator. The vector state of the system is sent to the integrator, which sends it back after a time equal to $T$, for the one-iterate.

The bifurcation analysis is conducted by solving the following algebraic vector equation:

$$\mathbf{x} - \mathbf{P}(\mathbf{x}, \mathbf{p}) = \mathbf{F}(\mathbf{x}, \varphi_T(\mathbf{x}), \mathbf{p}) = 0$$

(3)

where $\varphi_T(\mathbf{x})$ represents the time integration operator. Of course, since the map $\mathbf{P}$ has no analytical expression, the application of the Newton–Raphson method to Eq. (3) requires that both $\mathbf{F}$ and its Jacobian be computed numerically. This involves extensive numerical integrations in a number that grows with the square of the order of the reduced dynamical system.

2.2. Study of the stationary solutions of distributed chemical reactors

The problem of finding stationary solutions of nonlinear distributed dynamical systems like those arising from distributed chemical reactors can be often formulated as a boundary value problem. For a reaction–diffusion–convection system, the abstract form can be:

$$\begin{align*}
\mathbf{f} \left( \frac{d^2 \mathbf{x}}{dt^2} + \frac{d \mathbf{x}}{dt}, \mathbf{x}, \mathbf{p} \right) &= 0 \quad \xi \in [0, 1] \\
\mathbf{B} \left( \frac{d \mathbf{x}}{d \xi} \bigg|_{1}, \frac{d \mathbf{x}}{d \xi} \bigg|_{0} \right) \mathbf{x}(1), \mathbf{x}(0), \mathbf{p} &= 0
\end{align*}$$

(4)

where $\mathbf{f}$ is a system of differential equations in the spatial variable and $\mathbf{B}$ are the associated boundary conditions. In some cases, systems of the type (4) can be transformed into:

$$\mathbf{F}(\mathbf{u}, \varphi_\xi(\mathbf{x}), \mathbf{p}) = 0$$

(5)

and treated like a two-point boundary value problem (Berezowski, 2000). In Eq. (5) $\mathbf{u}$ is the subset of the state variables which are not given at the, say, right boundary, $\varphi_\xi(\mathbf{x})$ represents the integration operator along the spatial variable $\xi$, and $\mathbf{p}$ is the parameter vector. Starting conditions for $\mathbf{u}$ are given as:

$$\begin{bmatrix}
x^1_0 \\
x^2_0 \\
\vdots \\
x^m_0
\end{bmatrix} =
\begin{bmatrix}
u_1 \\
u_2 \\
\vdots \\
u_m
\end{bmatrix}$$

(6)

The remaining state variables must be determined as solutions of Eq. (5) so that the full stationary vector state $\mathbf{x}^s$ is finally computed. The problem is thus recast as a continuation problem, starting from a known stationary steady–state, in which solution diagrams are computed via pseudo arc length continuation. This method involves many Newton–Raphson iterations and the Jacobian of $\mathbf{F}$ again is not available in analytical form and must be computed numerically.

3. Parallel numerical computation of the Jacobian matrix

A pseudo arc length continuation algorithm (Keller, 1977) consists essentially of the following steps:

- Prediction.
- Pseudo arc length continuation with step size control. During this task there are $m$ evaluations of the Jacobian matrix for the Newton solver, where $m > 1$ depends on the chosen continuation step size and on numerical tolerances.
- Correction.
- Detection of bifurcation point. During this task there is one evaluation of the Jacobian matrix. In general, each point of the solution
(or bifurcation) line takes two or more evaluations of the Jacobian matrix.

Whenever a Jacobian is to be evaluated for a vector function \( \mathbf{F}(\mathbf{x}, \mathbf{p}) \) for which no analytical expression is available, derivatives are to be computed numerically. In our examples, function evaluations include numerical integration of given expressions. Parallelism applies to the numerical computation of derivatives. As an example, the following steps are necessary in order to compute the Jacobian matrix by means of second order difference operators:

1. Prepare 2n perturbed vectors of starting conditions:
   \[
   \mathbf{x}_{1F} = \begin{bmatrix} x_1 + \varepsilon \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{x}_{1B} = \begin{bmatrix} x_1 - \varepsilon \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{x}_{nF} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n + \varepsilon \end{bmatrix}
   \]
   \[\mathbf{x}_{nB} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n - \varepsilon \end{bmatrix} \quad (7)\]

2. Compute values of right hand sides with suitable perturbed conditions. Each of these operations takes significant computation power.
   \[
   \mathbf{F}_B = \mathbf{F}(\mathbf{x}_B), \quad \mathbf{F}_F = \mathbf{F}(\mathbf{x}_F), \quad i = 1, 2, \ldots, n \quad (8)
   \]

3. Compute the numerical derivatives by second order finite difference operators. These operations do not require much computation power.
   \[
   \frac{\partial \mathbf{F}(\mathbf{x})}{\partial \mathbf{x}} = \frac{\mathbf{F}_B - \mathbf{F}_F}{2\varepsilon}, \quad i = 1, 2, \ldots, n \quad (9)
   \]

   where \( n \) is the dimension of the system (number of state variables) whereas \( \varepsilon \) is a small magnitude arbitrary perturbation, for example:
   \[
   \varepsilon = 10^{-7}(1 + \max(P_i))
   \]

   In Eq. (7) subscript \( F \) means forward, \( B \) means backward perturbation. It is worth to stress here that all the operations expressed by Eqs. (7)–(9) are intrinsically independent and might be run in parallel. However, tasks involved by Eqs. (7) and (9) would require too much communication compared to the amount of computation, therefore it is convenient to run them as serial on the master processor. Task (8) takes almost the entire computation time due to the significant time consumption required by the numerical integration operator \( \varphi \). These operations are therefore run in parallel, by distributing them among available processors, for example, \( \mathbf{F}_{1F} \) on processor 1, \( \mathbf{F}_{2F} \) on processor 2, \( \mathbf{F}_{3F} \) on processor 3, etc. It appears that, in this case, the maximum number of processors that can be usefully involved is limited by twice the dimension of the system.

4. Realization of the parallel software to conduct continuation analysis

We used MPI and PTHREADS parallel libraries for distributing tasks around the cluster. There are two possible computation regimes, namely synchronous and asynchronous. For each of the two regimes a set of parallel subroutines is prepared.

- Synchronous regime of computation.

   In the synchronous regime of computation, the performance of the algorithm is limited by the slowest processor. This regime is then designed for homogeneous clusters and for shared memory Symmetric Multi Processing (SMP) machines. Generally in such systems all processors have the same performance and therefore the job is equally divided among processors. Delivery of results is expected to occur synchronously. The synchronous regime gives speedup starting from a 2-processor system.

   For large systems it is expected that the delay of the network connection does not play a significant role. To investigate this aspect, a comparison between a shared memory Symmetric Multi Processor (SMP) machine and a distributed cluster is conducted. SMP machines are composed by few high-performance nodes and latency is very small, as it is governed by the memory bus speed, whereas typically distributed-memory machines (clusters) are made by many medium-performance nodes connected by a network. If the parallel speedup is comparable between the two kinds of architectures, it will be concluded that communication speed is not too important in our problems.

   In both cases it makes sense to involve the master processor in parallel computations, since otherwise it would be running idle while awaiting for results to be delivered. For SMP systems (relatively small number of processors) the computing power employed with inclusion of the master is increased significantly.

- Asynchronous regime of computation.

   This regime is best suited for heterogeneous clusters or GRID computing, because it distributes the job dynamically for each node. So for example when slower and faster computers are used or the external load on the cluster is unknown/unpredictable, the algorithm behaves similarly to peer-to-peer networks. The speed in general is not limited by the slowest processor. This concept requires at least three processors to yield speedup (one master and two slaves). Since the work is partitioned into 2n independent simulations, this concept works well when the dimension of the system is larger than half of the number of available nodes, otherwise, all jobs would be allocated at the first distribution, dynamic allocation would not be possible and speedup would again be limited by the slowest processor.

   The software was prepared and tested for a distributed-memory architecture. However, the same concepts can be used for shared memory architectures. In fact, for the AUTO2000 version of the package, a parallel version was prepared by means of the PTHREADS library for shared memory machines. The software was run on a Linux cluster equipped with ROCKS 3.2.0 operating system and LAM-MPI 7.0.6 (Burns, Daoud, & Vaigl, 1994). For tests we used a system built by eight processor shared memory machine Intel® Xeon™ MP CPU 2.80 GHz 8 Gb, memory plus 6 individual nodes 1.8 GHz 500 Mb RAM. The network cards are 100 GbE.

4.1. Synchronous approach

For the synchronous approach, the master process realizes the main continuation process. During the execution of the Jacobian subroutine, the master prepares perturbed starting conditions and broadcasts them to available slaves. Slaves do the simulations. During these simulations the master also does simulation. After simulations are completed, the master collects the vector of variables and prepares next round of job for slaves and for itself. This is done until all 2n simulations are done, then the master computes the Jacobian. As shown in Fig. 1, all nodes do the same number of jobs. The same number of nodes (particularly, evaluations of the RHS) are done on the master. Fig. 1 reports a schematic of the synchronous algorithm.
4.2. Asynchronous approach

In the asynchronous approach (Fig. 2) the master process, as opposed to the synchronous approach, does not work on simulations, but rather waits for results from the simulations conducted by the slaves. Thus, the master process prepares suitable perturbed starting conditions \((2n)\) and broadcasts them to available \((ns)\) slaves. As soon as the result from slave \(k\) is received, the master sends a new job to the same slave \((k)\). This is done until all \(2n\) simulations are done (value of \(m\) achieves \(2n\)). Next, the master process computes the Jacobian matrix. This procedure works well for distributed-memory machines, heterogeneous clusters and also when there is large latency among some nodes, because speed is not limited by the slowest processor or the slowest connection. To obtain speedup, it needs more than two processors. On the figure: \(n\) – dimension of the system, \(m\) – work counter, \(ns\) – number of slaves, \(k\) – slave number.

4.3. PTHREADS approach

We used also the PTHREADS library in order to make it possible to use the parallel AUTO2000 on shared memory computers. The corresponding software can be useful on a multiprocessor computer where MPI is not installed. This subroutine is asynchronous in the sense that all slaves are created at the beginning and allocated on available processors by the system. In this implementation there is no control on which processor will realize each particular task. Tests run on a 4-processor SMP machine showed that speedup of this approach is very close to that obtained on synchronous MPI-based subroutines run on the same shared memory machine. It should be pointed out that the speedup depends on number of PTHREADS processes – threads.

4.4. Subroutine MASTER

This subroutine is run on the main computer of the cluster or on any of the processors on multiprocessor shared memory machines. This subroutine runs the main computational task (in our examples, parameter continuation) as the master process. A block diagram is presented in Fig. 3. Note that the subroutine for Jacobian matrix, both for the synchronous and asynchronous approach, is run by the master process, which in turn spawns parallel integration jobs to slaves. After the computation is finished, the master process sends an “end” tag to slaves, collects the working data (number of RHS calls, name of slave processor) and possibly outputs computation statistics.

4.5. Subroutine slave

Subroutines “slave” are run on slave nodes. They work in parallel and, obviously, independently from each other. The block diagram is given in Fig. 4. This subroutine is designed as a loop which can be terminated only by the master process. When the “work” tag is received, the subroutine receives a vector of variables and a vector

**Fig. 1.** Sketch of synchronous Jacobian subroutine.

**Fig. 2.** Sketch of asynchronous Jacobian subroutine.

**Fig. 3.** Sketch of subroutine “master”.

**Fig. 4.** Sketch of subroutine “slave”.
of parameters from the master process. Next, the RHS subroutine is called and the result – vector of RHS values is sent to the master process. When a slave receives an “end” tag, it sends statistics data (number of calls of RHS subroutine and name of processor) to the master and exits.

5. Speedup

Parallel speedup is defined as the ratio between the real time of a serial computation and real time of a parallel computation. Speedup in general depends on the system, especially on:

- difficulty of numerical evaluation of the RHS;
- scale of the system.

A general rule of thumb is that larger systems exhibit better speedup, and systems more difficult (time consuming) to integrate also lead to better speedup. This is obvious with respect to the fact that larger and more difficult systems typically lead to a larger fraction of computation work to be conducted on each processor before data intercommunication is required.

The simplest definition of speedup is due to Amdahl (1967). For a given, fixed size problem, and assuming infinite intercommunication speed among processors (no latency), Amdahl derived the ideal speedup $s$ that can be obtained on $p$ processors by an algorithm with fraction $q$ of parallel tasks and $(1 - q)$ of serial tasks, in the simple form:

$$s = \frac{p}{q + (1 - q)p}\quad (11)$$

The reverse form of Eq. (11),

$$q = \frac{p(s - 1)}{s(p - 1)}$$

can be used to estimate the parallel fraction $q$ of a given algorithm from the measured value of the speedup $s$.

Fig. 5 shows parallel performance comparison as a function of the scale of the system for one case of the Reverse-Flow Reactor model problem described later.

Amdahl’s law expresses the obvious concept that, for a given problem, no matter how fast one performs the parallel fraction of some algorithm, the total time cannot be any shorter than the time required by the sequential (serial) part of the algorithm. This reflects into the fact that, no matter how large $p$, $s$ can never be larger than $s_\infty$, given by

$$s_\infty = \lim_{p \to \infty} s = \frac{1}{1 - q}$$

This observation has led to skepticism towards massive parallelism. However, as observed by Gustafson (1988), when given a more powerful processor, programmers rather use their control over grid resolution, number of time steps, difference operator complexity and other parameters that are usually adjusted, to allow the program to run in some desired amount of time. The size of the problem then expands to make use of the increased facilities. Hence, it may be more realistic to assume that run time, not problem size, is constant. In this view, predictions of speedup based on Amdahl’s law are not too realistic. To incorporate this observation, Gustafson introduced the concept of “scalable speedup” in which the fraction of serial code (scaled serial fraction) is no longer fixed but scales with the size of the system. Gustafson speedup $s_G$, again under the zero-latency assumption, is then given by Gustafson (1988):

$$s_G = p - (p - 1)q_G\quad (12)$$

where $q_G$ is the serial fraction of the code, also called Gustafson bottleneck, which can be estimated by means of the reverse form of Eq. (12):

$$q_G = \frac{p - s_G}{p - 1}$$

Thus, by running a test we can simply estimate the Gustafson bottleneck by knowing the number of processors and the measured speedup. Following Gustafson, the test is run by increasing the size of the problem proportionally to the number of available nodes.

Table 1 shows a timing of a sample computation of a scalable reverse flow reactor model. As expected, the estimated serial fraction of the code decreases (parallel performance increase) as the problem size is increased. The average Gustafson serial fraction of code is $q_G = 0.2160$. This value might be used to predict the possible speedup of larger systems run on correspondingly larger parallel machines (Eq. (12)). However, hereafter we chose to test functionality of our parallel approaches following Amdahl’s paradigm (fixed size of problem, increasing size of cluster) since it poses more emphasis on parallel efficiency.

6. Examples

Non-ideal (distributed-parameter) models of chemical reactors are necessary to take into account in order to accurately described dynamical behavior. Therefore, chemical reactors models are frequently described by sets of nonlinear PDEs. A set of PDEs represents an infinite dimensional problem. The most
used approach to model reduction in chemical engineering is to discretize the spatial domain (e.g. Canuto, Hussaini, Quarteroni, & Zang, 1988). Traditional reduction schemes (finite elements, weighted residuals, etc.) lead to the determination of large sets of ODEs which can be written in abstract form as the following dynamical system:

\[
\begin{cases}
\dot{x} = f(x, p, g(t)) \\
B(x, p) = 0
\end{cases}
\]

(13)

where \( \mathbf{f} \) is the right hand side of PDE system, \( \mathbf{B} \) are the boundary conditions, \( x \) is the state vector and \( p \) is the parameters vector. Finally \( g(t) \) is time function that can represent a disturbance or more generally a periodic forcing.

Any non-autonomous \( T \)-periodic system can be studied with the Poincaré map \( \mathbf{P} \) (e.g. Kuznetsov, 1998):

\[
\mathbf{P} : \mathbf{u}_0 \in \Sigma \rightarrow \Phi_T(\mathbf{u}_0, 0) \in \Sigma
\]

where \( \mathbf{u} \) is the vector of state variables, \( \Sigma \) is a hyperplane (the so-called Poincaré section) orthogonal to the whole system orbits:

\[
\Sigma : = \{ x \in R^n : u_{m+1} = \tilde{u} \}, \quad \tilde{u} \in [0, 2\pi], \quad \text{with} \quad X = [0, 2\pi] \times R^m.
\]

In the following we always refer to the section with \( \tilde{u} = 0 \), and \( \Phi_T \) is the evolution operator of the dynamical system at \( t = T \). Any trajectory intersects orthogonally the Poincaré section \( \Sigma \) every \( T \)-time, and the Poincaré map \( \mathbf{P} \) merely tracks any initial conditions \( \mathbf{u}_0 \in \Sigma \) for \( t = 0 \) after a period \( T \). Thus, one can study the continuous-time periodic system with the globally equivalent Poincaré map. Therefore, the continuous-time system is “equivalent” (e.g. Kuznetsov, 1998) to the discrete-time system:

\[
\Sigma : (P^k_{1k}, z),
\]

where \( P^k \) is the \( k \)-iterate of the map, and:

\[
u_k = P^k(u_0 - 1) = P^k u_0 = \Phi_{kT}(u_0, 0)
\]

(14)

In general, a periodic orbit of a continuous time system may intersect a Poincaré section \( k \) times before closing onto itself: fixed points of \( \mathbf{P} \) correspond univocally to periodic orbits of period \( T = n \cdot T \); fixed points of \( P^k \) correspond to \( k \)-order subharmonic solutions of the continuous-time system. The stability of a periodic orbit of the continuous-time system is determined by studying the stability of the corresponding fixed point of the associated map according to the Floquet theory. Details of the construction of the discrete dynamical system (13) are described in Russo et al. (2002).

6.1. Discrete dynamical system: Reverse-Flow

We consider a first order exothermic reaction occurring in a fixed catalytic reactor where the flow direction is periodically reverted. The fixed-bed reactor is modeled as a heterogeneous system with heat and mass transfer resistance between the gas and the solid phase, axial dispersion in the gas phase, axial heat conduction in the solid phase, and cooling at the reactor wall. The mathematical model for the RFR reads (Reháček, Kubiček, & Marek, 1998):

Mass balance in the gas phase

\[
\frac{\partial \alpha}{\partial t} + (2g(t) - 1) \frac{\partial \alpha}{\partial x} = \frac{1}{Pe_M} \frac{\partial^2 \alpha}{\partial x^2} + \delta_M(\alpha_s - \alpha)
\]

(15)

Mass balance in the solid phase

\[
0 = D\alpha(1 - \alpha_s) \exp \left( \gamma \frac{\beta \alpha}{1 + \beta \alpha_s} \right) + \delta_S(\alpha_s - \alpha_s)
\]

(16)

Heat balance in the gas phase:

\[
\frac{\partial T}{\partial t} + (2g(t) - 1) \frac{\partial T}{\partial x} = \frac{1}{Pe_H} \frac{\partial^2 T}{\partial x^2} + \delta_H(T_s - T) + \delta_G(T_H - T) + \delta_\dot{\theta}(\dot{\theta}_s - \dot{\theta})
\]

(17)

Heat balance in the solid phase:

\[
\frac{\partial \theta}{\partial t} + \frac{1}{Pe_S} \frac{\partial^2 \theta}{\partial x^2} + \phi(\alpha_s, \theta) + \delta_G(\theta_s - \theta) + \delta_G(\theta_H - \theta)
\]

(18)

The periodic inversion of the flow direction is modeled by the following square wave function:

\[
g(t) = \begin{cases} 1 & \text{if } \frac{T}{\Gamma} \mod(2) < 1 \\ 0 & \text{if } \frac{T}{\Gamma} \mod(2) > 1 \end{cases}
\]

(19)

It is apparent that the vector field changes discontinuously in time, and it recovers the same form after a time \( \Gamma \). Indeed, \( g(t) \) is a discontinuous periodic function with minimum period \( \Gamma \) and the non-autonomous system Eqs. (15)–(19) is \( T \)-periodic.

The following boundary conditions for concentration in the gas phase and temperature in the solid and gas phases are assumed:

\[
\begin{align*}
\alpha &= \frac{1}{Pe_M} \frac{\partial \alpha}{\partial x} = 0 \\
\theta &= \frac{1}{Pe_S} \frac{\partial \theta}{\partial x} = 0
\end{align*}
\]

(20)

Eqs. (15)–(20) are dimensionless with axial coordinate defined as \( \xi = x/L \), time \( t = \Gamma \), gas phase conversion \( \alpha = (C_0 - C)/C_0 \), conversion on the catalyst \( \alpha_s = (C_0 - C_s)/C_0 \), gas phase temperature \( \theta_s = E(T - T_0)/RT_s^2 \), and catalyst temperature \( \theta_s = (T_s - T_0)/RT_s^2 \). The definition of all dimensionless parameters and their values are:

\[
\begin{align*}
Pe_M &= 317.0, \quad Pe_H = 644.0, \quad Pe_S = 115.08, \quad \theta_H = -0.79, \\
Da &= 0.56, \quad \sigma = 1251.00, \quad \delta = 0.72, \quad \gamma = 16.68, \\
\beta &= 0.7282, \quad \delta_M = 17.5, \quad \delta_S = 22.9, \quad \delta_H = 28.4, \quad \delta_\dot{\theta} = 28.4.
\end{align*}
\]

For subsequent numerical investigation, the infinite dimensional PDE system Eqs. (15)–(20) is reduced to a set of 36 ODEs by orthogonal collocation technique on finite elements (Villadsen & Michelsen, 1978). The domain of each reactor has been divided into three blocks, and four collocation points are used in each sub domain.

The stability of a periodic orbit of the system Eqs. (15)–(20) is assessed by studying the stability of the corresponding fixed point of the associated map (Eq. (14)).

A typical solution diagram is represented in Fig. 6. Each point represents a T-periodic regime, stable periodic regimes are shown as solid lines, and unstable regimes are shown as dashed lines. The computation of the stability characteristics of each point on the diagram reported in Fig. 6 needs several estimations of the Jacobian matrix of the discrete system. The corresponding performance diagrams and computing times are presented in Figs. 7 and 8.

It should be underlined that large scale systems with integration operator are characterized by little needs of intercommunication among processors. The measure on our homogeneous cluster compared with a shared memory 4-processor machine shows that speedup in the two cases is similar. The average estimated fraction of parallel code in the case of distributed memory machine is 0.9725 (Table 2). The estimated fraction of parallel code in the case of shared memory machine is 0.9866. Difference in the estimates is an indication of the deviation from ideal behavior of processors,
Fig. 6. Solution diagram, Reverse-Flow Reactor model. Conversion in the first node $\alpha$ vs. switch period $r$.

Fig. 7. Speedup, RFR model, distributed cluster.

Fig. 8. Speedup comparison, RFR model, distributed cluster and shared memory machine.

assumed to derive Amdahl’s law in the simple form of Eq. (11), as a consequence of latencies due to intercommunication processes on distributed systems. The difference visible in Fig. 8 corresponds to about 1.4%.

Fig. 9 shows the prediction of Amdahl’s speedup on larger computers by means of Eq. (11). The two lines correspond to the estimated fraction of the parallel code from shared memory machine and distributed cluster. As a general remark, it is seen that ideal performance estimated by Amdahl’s law for larger configurations is highly sensitive to even small changes in the fraction $(1 - q)$ of serial code. For a given size problem, increasing the number of nodes beyond a certain value does not provide significant benefits. However, if the size of the problem is correspondingly increased (Gustafson, 1988), parallel performance can also increase.

Previous computation results all refer to the synchronous approach run on a shared memory machine or on a homogeneous cluster. The asynchronous computation approach mentioned earlier along with the relevant software was also developed and tested. Table 3 presents a sample computation of a portion part of the solution line reported in Fig. 6.

Table 2

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Elapsed time (h)</th>
<th>Speedup</th>
<th>Estimated fraction $q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (serial)</td>
<td>10.81</td>
<td>1.000</td>
<td>0.974</td>
</tr>
<tr>
<td>2</td>
<td>5.551</td>
<td>1.949</td>
<td>0.971</td>
</tr>
<tr>
<td>4</td>
<td>2.939</td>
<td>3.680</td>
<td>0.973</td>
</tr>
<tr>
<td>6</td>
<td>2.057</td>
<td>5.278</td>
<td>0.973</td>
</tr>
<tr>
<td>8</td>
<td>1.614</td>
<td>6.703</td>
<td>0.972</td>
</tr>
</tbody>
</table>

Table 3

<table>
<thead>
<tr>
<th>Processor</th>
<th>State</th>
<th>Number of jobs</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Free</td>
<td>528</td>
<td>105.4</td>
</tr>
<tr>
<td>2</td>
<td>Free</td>
<td>529</td>
<td>131.0</td>
</tr>
<tr>
<td>3</td>
<td>Free</td>
<td>527</td>
<td>151.1</td>
</tr>
<tr>
<td>1</td>
<td>Busy</td>
<td>499</td>
<td>105.4</td>
</tr>
<tr>
<td>2</td>
<td>Busy</td>
<td>495</td>
<td>131.0</td>
</tr>
<tr>
<td>3</td>
<td>Busy</td>
<td>499</td>
<td>151.1</td>
</tr>
</tbody>
</table>

The computing platform employed has different available processors power. More precisely, a heterogeneous cluster was simulated by a homogeneous four processor system in which one or more processors were loaded with additional serial computation tasks, so that the available computation power of a processor is lower than that of the free processor. The resulting final number of jobs completed for each processor and the corresponding timing are reported in Table 3. In the table, “free” state means that the entire computation power of the processor is dedicated to the

Fig. 9. Prediction of the speedup on larger computers based on estimates of the parallel fraction of the code.
Table 4
Dimensionless parameters with $T_w = 100$ °C, $T_0 = 200$ °C, $u_w = 1$ m/s, $r_0 = 1$ m (Mancusi et al., 2010).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B = 16$</td>
<td></td>
</tr>
<tr>
<td>$\theta_w = -8.25$</td>
<td></td>
</tr>
<tr>
<td>$Re = 413$</td>
<td></td>
</tr>
<tr>
<td>$Pr = 390$</td>
<td></td>
</tr>
<tr>
<td>$Da = 0.0017$</td>
<td></td>
</tr>
<tr>
<td>$Le = 27$</td>
<td></td>
</tr>
<tr>
<td>$\psi = 1.64$</td>
<td></td>
</tr>
<tr>
<td>$\psi = 3.28^3$</td>
<td></td>
</tr>
<tr>
<td>$L = 1/2$</td>
<td></td>
</tr>
</tbody>
</table>

6.2. Discrete dynamical system: a network of four reactors

A Reactor Network (RN) of four connected catalytic reactors with periodically switched inlet and outlet sections is selected as representative process. The forcing strategy consists of periodically switching the feed to the second reactor of the reactors sequence (Mancusi et al., 2010, 2011). Namely, the reactors are fed according to the sequence 1-2-3-4 during the time interval $[0, \tau]$. $\tau$ being the switch (or cycle) time, and after the first switch, within the interval $[\tau, 2\tau]$, the reactors are fed according to the sequence 4-3-2-1 and so on until to recover the initial feed configuration 1-2-3-4. It is worth to note that the forced network is $T$-periodic, that is the network recovers the initial configuration after a time $T = 4\tau$.

The mathematical model of each fixed bed of the RN consists of a three-dimensional pseudo-homogeneous model taking into account:

$$ CO + 2H_2 \rightarrow \text{CH}_3\text{OH} $$

Under the assumption of first order reversible reaction, the dimensionless mass and energy balances for the $i$-th reactor of the RN read as follows:

$$ \frac{d\theta_i}{dt} + \frac{1}{Pe_m} \frac{d\theta_i}{dx} = \frac{1}{Pr_h} \frac{d^2\theta_i}{dx^2} + Br(x_i, \theta_i) $$

$$ \frac{d\xi_i}{dt} + \frac{1}{Pe_h} \frac{d\xi_i}{dx} = \frac{1}{Pe_m} \frac{d^2\xi_i}{dx^2} - r(x_i, \theta_i) $$

$$ r(x_i, \theta_i) = \frac{Da \exp \left( \frac{\theta_i}{(\theta_i + \gamma)} \right) \left( 1 - x_i \left( 1 + \psi \left( \frac{(1 - \mu)^2}{\theta_i + \gamma} \right) \right) \right)}{\mu} ) \left( 1 - x_i \left( 1 + \psi \left( \frac{(1 - \mu)^2}{\theta_i + \gamma} \right) \right) \right)$$

with the following definitions for dimensionless variables and parameters:

$$ \mu = \frac{E_2}{E_1} ; \psi = \frac{A_2}{A_1} ; \xi = \frac{z}{z_0} ; \tau = \frac{t}{t_0} ; \gamma = \frac{T - T_0}{T} ; \psi = \frac{T - T_0}{T} ; \mu = \frac{C}{C_0} ; $$

$$ \gamma = \frac{E}{RT} ; \psi = \frac{u}{u_0} ; B = \left( \frac{\Delta H}{C_0 \gamma} \right) ; Da = \frac{A_2 z_0}{u_0} \exp(-\gamma) ; $$

$$ Le \left( \rho c_p' \right) \frac{\partial \xi_i}{\partial t} + Pe_h \left( \rho c_p' \right) \frac{\partial \theta_i}{\partial t} = \left( \rho c_p' \right) \frac{z_0 u_0}{k_e} ; Pe_m = \left( \rho c_p' \right) \frac{z_0 u_0}{D_f}. $$

The infinite dimensional PDE system (Eqs. (21)–(24)) has been reduced to a set of 128 ordinary differential equations (ODEs) by orthogonal collocation on finite elements (Villadsen & Michelsen, 1978). A coarser grid has been used giving rise to a smaller set of equations (80 ordinary differential equations) to study the system dimensions effect on the speedup.

A typical solution diagram of the system presented above is reported in Fig. 10. The diagram was obtained by performing the parameter continuation of the full 128 equations system. The ignited (high conversion) solutions form an isoala bounded by two catastrophic saddle node bifurcation points, while the washout solution is not reported. As the switch time is varied within the range $\tau_1 < \tau < \tau_2$, stable high conversion $T$-periodic regimes coexist with stable non ignited $T$-periodic regimes.

The size of the system will not affect, qualitatively and quantitatively, the dynamic behavior. That is, the solution diagrams of the coarser system (80 equations) coincide with the solution diagram reported in Fig. 10. The computation time required by a single processor to obtain the solution diagram reported in Fig. 10 is 56.4 h.
when the 128 ODEs model is considered and 11.1 h for the 80 ODEs model.

The corresponding performance diagrams are presented in Fig. 11a and b for 128 and 80 equations systems respectively.

In Fig. 11 comparison between speedups of two algorithms are presented – asynchronous and synchronous – for both 128 and 80 equations cases. It is worth to note that, independently on the system dimension, for a number of processors lower than 8 the synchronous algorithm has better performance. A better insight can be obtained by analyzing the data reported in Table 5 where the exact observed values of time consumption and speedup for both cases are presented.

As it was previously mentioned, the synchronous process uses all processors for Jacobian calculation so the entire resource is exploited. But this regime of work is characterized by synchronous job distribution and synchronous feedback from all slave nodes. It means that, in the case that any slave process temporary works slower, the entire process awaits for it. This evidently leads to decreasing speedup for higher number of nodes. Therefore, the synchronous algorithm works more efficiently with a small number of clustered processors or with a shared memory machine. By default, current shared memory machines are characterized by a relatively small number of processors. The asynchronous algorithm is characterized by dynamic work distribution by one (master) process to available slave nodes regardless of computing power characterizing each node. The slowest nodes do not limit the entire computation speed. The master process is excluded from calculating the Jacobian matrix. The synchronous algorithm works very efficiently for large scale distributed cluster because is characterized by optimal job distribution. In the analyzed case, the asynchronous algorithm

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>128 equations case</th>
<th>80 equations case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Asynchronous</td>
<td>Synchronous</td>
</tr>
<tr>
<td></td>
<td>Elapsed time (h)</td>
<td>Speedup</td>
</tr>
<tr>
<td>1 (serial)</td>
<td>56.39</td>
<td>1.97</td>
</tr>
<tr>
<td>2</td>
<td>64.49</td>
<td>0.87</td>
</tr>
<tr>
<td>3</td>
<td>32.49</td>
<td>2.73</td>
</tr>
<tr>
<td>4</td>
<td>21.89</td>
<td>3.82</td>
</tr>
<tr>
<td>5</td>
<td>16.56</td>
<td>3.39</td>
</tr>
<tr>
<td>6</td>
<td>13.36</td>
<td>4.22</td>
</tr>
<tr>
<td>7</td>
<td>10.54</td>
<td>5.35</td>
</tr>
<tr>
<td>8</td>
<td>8.71</td>
<td>6.48</td>
</tr>
<tr>
<td>9</td>
<td>7.47</td>
<td>7.55</td>
</tr>
<tr>
<td>10</td>
<td>6.52</td>
<td>8.65</td>
</tr>
<tr>
<td>11</td>
<td>5.85</td>
<td>9.64</td>
</tr>
<tr>
<td>12</td>
<td>5.29</td>
<td>10.67</td>
</tr>
<tr>
<td>13</td>
<td>4.83</td>
<td>11.68</td>
</tr>
<tr>
<td>14</td>
<td>4.51</td>
<td>12.50</td>
</tr>
<tr>
<td>15</td>
<td>4.16</td>
<td>13.56</td>
</tr>
<tr>
<td>16</td>
<td>3.91</td>
<td>14.41</td>
</tr>
<tr>
<td>17</td>
<td>3.67</td>
<td>15.38</td>
</tr>
<tr>
<td>18</td>
<td>3.47</td>
<td>16.23</td>
</tr>
<tr>
<td>19</td>
<td>3.29</td>
<td>17.13</td>
</tr>
<tr>
<td>20</td>
<td>3.11</td>
<td>18.14</td>
</tr>
<tr>
<td>21</td>
<td>3.01</td>
<td>18.74</td>
</tr>
<tr>
<td>22</td>
<td>2.91</td>
<td>19.35</td>
</tr>
<tr>
<td>23</td>
<td>2.80</td>
<td>20.15</td>
</tr>
<tr>
<td>24</td>
<td>2.64</td>
<td>21.39</td>
</tr>
<tr>
<td>25</td>
<td>2.56</td>
<td>22.04</td>
</tr>
<tr>
<td>26</td>
<td>2.50</td>
<td>22.54</td>
</tr>
<tr>
<td>27</td>
<td>2.43</td>
<td>23.19</td>
</tr>
<tr>
<td>28</td>
<td>2.33</td>
<td>24.21</td>
</tr>
<tr>
<td>29</td>
<td>2.29</td>
<td>24.62</td>
</tr>
<tr>
<td>30</td>
<td>2.27</td>
<td>24.88</td>
</tr>
</tbody>
</table>

Fig. 11. Speedup of distributed cluster for synchronous and asynchronous continuation: (a) 128 ODEs system; (b) 80 ODEs system.
works better than the synchronous for 8 and more processors. It is necessary to point out that parallel execution is more effective for larger problems – the 128 equations case express higher speedup than the 80 equations one.

The stair step shape of the synchronous speedup curves for large number of nodes is caused by the splitting of the natural number of jobs (simulations) by the natural number of processors. For example here we examine the case of 80 equations case, 16 and 17 processors. All simulations are done by several rounds of calculation. Therefore, in order to calculate derivatives of 80 equations it is necessary to make 160 computations. Thus, if 160 computations are distributed among 16 processors, it is necessary to make 10 rounds of calculations. When the same problem is calculated by 17 processors, it would be necessary to make 9.411 rounds – more precisely 17 processors will calculate 9 rounds, and the remaining 7 jobs need a last, 10th round. For 18 processors 8.88 rounds would be necessary but, more precisely, 18 processors work for 8 rounds, the remaining 16 jobs need a last, 9th round. So, 16 and 17 processors work for 10 rounds, 18 processors for 9 rounds. Note that the asynchronous algorithm is also partially affected by the mentioned phenomenon, nonetheless we can affirm that the asynchronous algorithm is more efficient than the synchronous algorithm for a large scale distributed cluster.

7. Conclusions and future work

We have proposed a parallel computation approach for the bifurcation analysis of a class of systems. The systems are characterized by:

- Numerically expensive evaluation of the right hand side (RHS).
- Impossibility of obtaining analytical derivatives of the RHS.
- High degree of parallelism in the computational opera related to sensitivity matrix or parameter derivatives of the systems.

We propose the parallel computation of these derivatives and illustrate our proposal by running parallel bifurcation analysis and/or parameter continuation of mathematical models based on difference equations and algebraic equations. We found that, for relatively large systems (our largest case shown consists of a system of 128 difference equations), speedup is almost linear and does not depend much on the computer architecture. One test was conducted by following Gustafson’s paradigm (size of problem scaled with size of machine) while all other tests were conducted by following Amdahl’s paradigm (fixed size of problem for varied size of machine). We obtained similar speedup on a distributed cluster and a shared memory, four-processor machine, with a difference of about 1.4% estimated by means of the reverse Amdahl’s law. Our results indicate a host of potential application: there exists a wide variety of systems which are characterized by significant time consumption in computing the right hand side of the model equations and that could greatly benefit from parallel computing. Parallelism will be useful in conducting computations which need numerical derivatives of right hand sides of such systems, for example algorithms of optimization, estimation of Lyapunov exponent for discrete dynamical systems, models involving integro-differential equations etc. Finally, parallelization was developed, implemented and tested with two different parallel versions of the software, allowing alternatively synchronous and asynchronous regime of computation. Generally, synchronous subroutines perform better on homogeneous shared memory ma and asynchronous subroutines on heterogeneous clusters. The corresponding software is available as modified versions of AUTO97 and AUTO2000.

Acknowledgments

We are indebted with: Constantinios Siettos for helpful discussions and comments, and Francesco Saverio Marra for kindly hosting and assisting with many computations on his Linux cluster at Istituto di Ricerca sulla Combustione, CNR, Naples, Italy.

Appendix A. Supplementary data


References


Websites

Lam MPI www.lam-mpi.org.

AUTO97 and AUTO2000 (modified parallel versions) at: http://www.ing.unisannio.it/continillo/AUTO.