Parallel Computing Applied to the Dynamics of One-Dimensional Cellular Automata

ALAN Z. LABES, SILVIA C. HIRATA, MANUEL E. CRUZ
Department of Mechanical Engineering
EE/COPPE/UFRJ
C.P. 68503, CEP 21945-970, Rio de Janeiro, RJ
BRAZIL

RITA M. ZORZENON DOS SANTOS
Institute of Physics
Universidade Federal Fluminense
C.P. 100296, CEP 24210-340, Niterói, RJ
BRAZIL

Abstract: Parallel computing is becoming increasingly important for solving technological and scientific problems. In fact, the most powerful computers in the world are now parallel machines. In this paper we utilize a low-cost parallel computing environment, based on the UNIX operating system and the PVM (Parallel Virtual Machine) and MPI (Message Passing Interface) communication protocols, to compute the dynamical evolution of one-dimensional cellular automata with large numbers of sites. Specifically, we develop and validate parallel binary PVM and MPI implementations, analyse their parallel performance measures, and use the relatively faster MPI implementation to obtain some results for cellular automata with large numbers of sites.

Key-Words: Parallel computing, PVM, MPI, cellular automata, complex systems.

1 Introduction

The importance of parallel and distributed computing for solving technological and scientific problems has been growing steadily over the last fifteen years (El-Rewini & Lewis, 1998; Leiss, 1995). Powerful parallel machines and clusters of machines are nowadays available to researchers and the scientific community at supercomputing centers in large research laboratories and universities throughout the world. In addition, general-purpose parallel routines are also available through the Internet (e.g., PETSc (Balay *et al.*, 1997)), which can be incorporated into the user's application code. In order to make appropriate use of such valuable computing resources, access to a parallel program development environment is necessary for the users, where they can produce and test parallel algorithms and codes.

In this paper we briefly describe a low-cost parallel computing environment that we have recently implemented (Dallalana *et al.*, 1998), and utilize it to compute the dynamical evolution of one-dimensional cellular automata (Wolfram, 1994) with large numbers of sites. Specifically, we develop *parallel binary* implementations using the PVM (acronym for Parallel Virtual Machine, Geist *et al*., 1994) and MPI (acronym for Message Passing Interface, Snir *et al*., 1996) communication protocols. We first present the binary and parallel algorithms, and describe how they are implemented under PVM and MPI. Next, we validate these implementations by comparing serial and parallel computations to benchmark results obtained by Wolfram (1994) with small numbers of sites. We then present and discuss the parallel performance measures of the PVM and MPI implementations. Finally, we perform parallel calculations with the relatively faster MPI implementation to obtain some results for cellular automata with large numbers of sites.

As detailed in Wolfram (1994), cellular automata are mathematical models for complex physical systems, in which the discrete values of the physical quantities at discrete spatial sites evolve in discrete time steps according to well-defined local or non-local rules. Cellular automata have been shown to model a wide range of physical (Rothman & Zaleski, 1997; Toffoli &

Margolus, 1987) and biological systems (Bernardes & Zorzenon dos Santos, 1997; Stauffer, 1991). Typically, the simulation of the behavior of real systems by cellular automata is much improved when the number of sites is increased. Therefore, the memory size of a machine executing a serial computation may become a severe limiting factor for the desired simulations. When the algorithm for the time evolution of a particular class of cellular automata can be efficiently parallelized, as we show here in the context of one-dimensional cellular automata, distributed-memory parallel computing becomes an attractive way to effect simulations with large numbers of sites, in order to obtain more relevant results.

2 Parallel Computing Environment

Our low-cost parallel computing environment (Dallalana *et al*., 1998) is based on the UNIX operating system and the cost-free communication protocol software packages PVM (Geist *et al*., 1994) and MPI (Snir *et al*., 1996), supported by a local Ethernet network which connects the participating machines. The UNIX system employed may be either the native operating system in workstations or the cost-free version Linux (Welsh, 1995) in personal microcomputers. The software packages PVM and MPI enable a cluster of homogeneous or heterogeneous computers to function as a single parallel machine; messages can be exchanged between processors through UNIX networking resources. Therefore, the environment supports the distributed-memory message-passing – MIMD loosely-coupled – computational model (El-Rewini & Lewis, 1998; Leiss, 1995). PVM and MPI come equipped with a collection of FORTRAN77 library subroutines, which are linked to the application object code and accessed at runtime.

The parallel environment has been utilized in previous work on heat conduction in composite materials (Dallalana *et al.*, 1998), where it has been observed that parallel computations can be both faster and slower than the equivalent serial computations, depending on the parallel algorithm adopted and the size (or granularity) of the problem. Also, as we show here, the same parallel algorithm performs differently on the same hardware when different communication protocols are used.

In this work we use a machine cluster composed of three workstations IBM RISC System/6000 43P-133 running the AIX operating system, each with 64 Mb of RAM and a native Ethernet card which transmits and receives data at a rate of 10 Mbps. We have used Version 3 of PVM and Version 1.1.1 of MPI**.** The machines in this homogeneous cluster are interconnected through a hub, such that they topologically form a 2-D ring network (El-Rewini & Lewis, 1998).

3 One-Dimensional Cellular Automata

Cellular automata (Wolfram, 1994) are discrete mathematical models for complex physical systems. A cellular automaton consists of a uniform lattice of identical sites in *n*-dimensional space. At each site, discrete values (in a finite set) are assigned to each physical variable considered, which evolves in discrete time steps according to a well-defined local or non-local rule. The value of one variable at a site depends on the values of the variable at neighboring sites evaluated at previous time steps. Typically, the neighborhood of a site is taken as the site itself and a prescribed number of adjacent sites. The state of an automaton at an instant of time is completely specified by the values of all the variables at each site. During the physical evolution of the automaton, the update of the values of the variables is effected simultaneously, or synchronously, at all the sites.

Cellular automata are employed to model real systems constituted by a large number of elementary discrete components which interact locally or non-locally with each other (for example, a volume of gas with many molecules, or a living organism with many cells). Even when the interactions are simple to describe, the large numbers of components in these systems lead to very complex global behaviors. A comprehensive analysis of cellular automata is presented in Wolfram (1994), where applications to different fields are developed, and many important issues are addressed, such as classification, local and global statistical properties, topology and self-organization of configurations, irreversibility, and computational irreducibility, universality and undecidability.

Here we consider the dynamical evolution of finite one-dimensional cellular automata. In particular, one binary variable V is defined at N sites; the neighborhood of a site is simply taken as the site itself and the two adjacent sites to the right and left. Such cellular automata are called *elementary* (Wolfram, 1994), for which there are 2^3=8 possible configurations for the neighborhood of a site, 2^8=256 local rules, and 2^N possible states. The variable V may be subject to fixed or periodic

boundary conditions; since both types of boundary conditions lead to essentially the same global behavior, we impose here the latter type, so that the first and last sites are neighbors. The temporal evolution of an elementary cellular automaton may be written in the general form

$$V_i^{t+1} = F\left(V_{i-1}^t, V_i^t, V_{i+1}^t\right), \qquad (1)$$

where V_i^t, $V_i^t \in \{0,1\}$, represents the value of variable V at site i, $i \in \{1,2,\ldots,N\}$, at time instant t, $t \geq 0$. Rule F may be specified by a binary number with eight digits, one digit for each possible configuration for the neighborhood of a site; rule F may thus be labeled by an integer decimal number in the set {0,1,...,255}. To update the value of V at a site i, the configuration of the neighborhood of site i is first identified at the previous time, and then V takes on the value of the digit of F corresponding to that configuration.

A general algorithm to allow for the calculation of the evolution of an elementary cellular automaton according to any rule F in the set {0,1,...,255} may be described as follows: (i) first, we convert the decimal number specifying rule F to a binary number with eight digits, $d_7d_6d_5d_4d_3d_2d_1d_0$; (ii) next, for site i, $i \in \{1,...,N\}$, we determine the configuration of the neighborhood, C, using the expression

$$C = 4 * V_{i-1}^t + 2 * V_i^t + V_{i+1}^t, \qquad (2)$$

where C is an integer such that $C \in \{0,...,7\}$; (iii) finally, the update may be effected by setting $V_i^{t+1}=d_C$. The algorithm just described for the automaton evolution under rule F could be implemented in a parallel or serial machine; only one N-long integer vector array, $A=\{A_1,A_2,...,A_N\}$, would be required to store the values of V at the N sites (N^t sites with values at time t and N^{t+1} sites with values at time t+1, $N^t+N^{t+1}=N$). An auxiliary integer array with three entries would then be needed to store the values of V in the neighborhood of site i at time t; N^t, N^{t+1}, and the three entries would change after the update of each site. However, since V is a binary variable, such an implementation would not make optimal use of machine memory. In the next section we describe our general *binary algorithm*, which will allow for optimal use of machine memory in both parallel and serial implementations.

4 The Binary Algorithm

The binary algorithm is devised so that each bit of machine memory can be used to store the value of V at a site. A word of memory with N_{by} bytes contains $N_{bi}=8N_{by}$ bits; thus, N_W words can hold $N=8N_WN_{by}=N_WN_{bi}$ sites. A general binary algorithm to allow for the update of all the sites in a word according to *any* rule F in the set {0,1,...,255} cannot be constructed using a single binary operation; a sequence of binary operations is thus needed, as described next.

We consider a word with N_{bi} bits, which are numbered from 0 to N_{bi}-1, from right to left. Also, the decimal number specifying rule F is assumed to be converted to a binary number with eight digits, $d_7d_6d_5d_4d_3d_2d_1d_0$, and in the following we refer to the array A of the previous section and to the words W_1, W_2, W_3, which respectively store the words A_{k-1}, A_k, and A_{k+1}, $1 \leq k \leq (N/N_{bi})-1$. For a site i whose value of V is stored at bit b of the word W_2, $b \in \{0,...,N_{bi}-1\}$, and belonging to word A_k, the first step is to identify the corresponding configuration, C, of its neighborhood. If b=0, four steps are necessary to determine C: first, W_2 is shifted to the left by N_{bi}-2 places; second, the resulting word is shifted N_{bi}-3 places to the right; third, W_1 is shifted by N_{bi}-1 places to the right; fourth, the binary operation "or" operates on the words obtained in the second and third steps. If $b=N_{bi}$-1, four steps are also necessary to determine C: first, W_3 is shifted to the left by N_{bi}-1 places; second, the resulting word is shifted N_{bi}-3 places to the right; third, W_2 is shifted by N_{bi}-2 places to the right; fourth, the binary operation "or" operates on the words obtained in the second and third steps. Finally, if $b \in \{1,..., N_{bi}-2\}$, only two steps are necessary to determine C: first, W_2 is shifted by N_{bi}-2-b places to the left; second, the resulting word is shifted by N_{bi}-3 places to the right. The word obtained after executing the corresponding steps for each value of b contains the binary representation of the integer value of C, $C \in \{0,...,7\}$. The update of V at site i may now be effected by simply setting $V_i^{t+1}=d_C$. This new value must next be introduced at bit b of the word A_k. We start with bit b=0: we simply attribute the integer value d_C to the word A_k, thus erasing its contents at time t. For $b \in \{1,..., N_{bi}-1\}$, we introduce the new value at bit b through the binary operation "or," whose operands are the word A_k (at time t+1) and the word whose bits are all zero except the one at position b, whose value is d_C.

5 The Parallel Algorithm

The purpose of our parallel computing effort is to allow for cellular automata simulations with larger numbers of sites compared to serial simulations, for a given machine specification and a given spatial dimension *n* of the lattice. The synchronous update of the values of *V* at the *N* sites, based on the previous time step, is the characteristics of the dynamical evolution of the automata which permits efficient parallel processing in a network of computers. The parallel algorithm which we now describe, to simulate the evolution of one-dimensional elementary cellular automata as given by expression (1), has been devised for the distributed-memory message-passing − MIMD loosely-coupled − computational model (El-Rewini & Lewis, 1998; Leiss, 1995), in which there are *P* participating processors.

The first component of the parallel algorithm is *data partition*: the line (i.e., one-dimensional lattice) with *N* sites is divided into *P* equal, or approximately equal, segments, each segment containing *N*/*P*, or approximately *N*/*P*, sites; next, the segments are distributed to the processors, as illustrated in Fig. 1. The sites are numbered locally in each processor, from 1 to *N*/*P* (or approximately *N*/*P*), from right to left. The other component of the parallel algorithm is *nearest-neighbor data communication*: in order to compute the evolution of all its resident (local) sites, each processor at each time step needs to exchange the values of *V* at its two extremal sites of the resident segment. This is accomplished through nearest-neighbor message-passing, as illustrated in Fig. 1; since we are imposing periodic boundary conditions, the

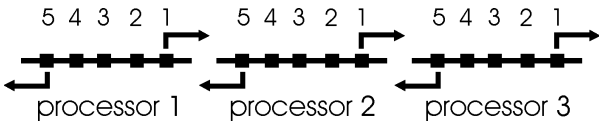

Figure 1. Illustration of the parallel algorithm showing data partition and nearest-neighbor data communication (*N*=15 sites, *P*=3 processors).

first and last processors are neighbors, and need to exchange data.

6 The Parallel Implementations

The binary and parallel algorithms presented in the previous two sections are implemented together as a single code, one for the PVM protocol and another one for the MPI protocol. The parallel codes are written in the FORTRAN77 programming language, and are essentially identical, except for the specific PVM and MPI calls which effect the parallel computing tasks. We use words of type INTEGER4 to store the values of *V* at the sites; thus, N_{by}=4 and N_{bi}=32. In both PVM and MPI implementations, the minimum amount of data that can be exchanged between processors is one word, or 32 bits in our case. Therefore, instead of exchanging data with one of its neighbors corresponding to one single site, a processor exchanges one word of data, corresponding to 32 sites; the processor thus uses just one bit of information, and disregards the remaining 31 bits.

6.1 The PVM Implementation

The PVM implementation is now described by indicating the FORTRAN77 PVM routines (Geist *et al.*, 1994) which are called from within the program to effect the various tasks of the parallel algorithm previously presented. It is first required to start up PVM and, next, to configure a parallel virtual machine with *P* machines of our cluster, $P \leq 3$; one of the *P* participating machines is the parent (the one where PVM is started), the other $P-1$ are the children (initialized by the parent with the `pvmfspawn` routine).

Data partition requires the labeling of each of the *P* participating machines of a virtual machine; the setup and identification of each of the *P* machines are effected with the `pvmfmytid` and `pvmfparent` routines. Nearest-neighbor data communication, required to update the extremal sites of the resident automaton segment in one processor, is effected in four steps: first, a message buffer is initialized by calling the `pvmfinitsend` routine; second, the word of data containing one of the extremal sites is exchanged by the call to the `pvmfpsend` routine, which must be matched at the receiving processor by the call to the `pvmfprecv` routine; third and fourth, respectively, the first and second steps are repeated for the other extremal site.

6.2 The MPI Implementation

The MPI implementation follows the same structure of the PVM implementation; we now just indicate the FORTRAN77 MPI routines (Snir *et al.*, 1996) needed to implement the parallel algorithm. Start up of MPI and the configuration of the parallel virtual machine are effected by writing the names of the *P* machines to an ASCII data file (in our case named `machines.rs6000`) and calling the MPI_INIT and MPI_COMM_SIZE

routines. The labeling of each of the *P* machines is effected by calling the MPI_COMM_RANK routine. Finally, calls to the routines MPI_SEND and MPI_RECV are necessary to send and to receive one word of data, respectively.

7 Validation Results

The validation of the PVM and MPI implementations has been effected in two steps: first, we reproduce some of the previous benchmark results presented in Wolfram (1994), utilizing only one processor of our cluster; second, for the same computation and for each implementation, we verify that the same evolution is obtained as the number of processors in the parallel virtual machine is increased to 2 and 3. We report here that both the PVM and MPI implementations are working properly: serial and parallel computations agree, and also the implementations agree with each other. As an illustrative example, in Fig. 2 we show 370 time steps of the evolution of cellular automata with N=320 sites, starting from a single initial nonzero site, according to rules 18, 45, 73, and 110. It is visually observed, to printing resolution, that these evolutions agree with those reported in Wolfram (1994, p. 501).

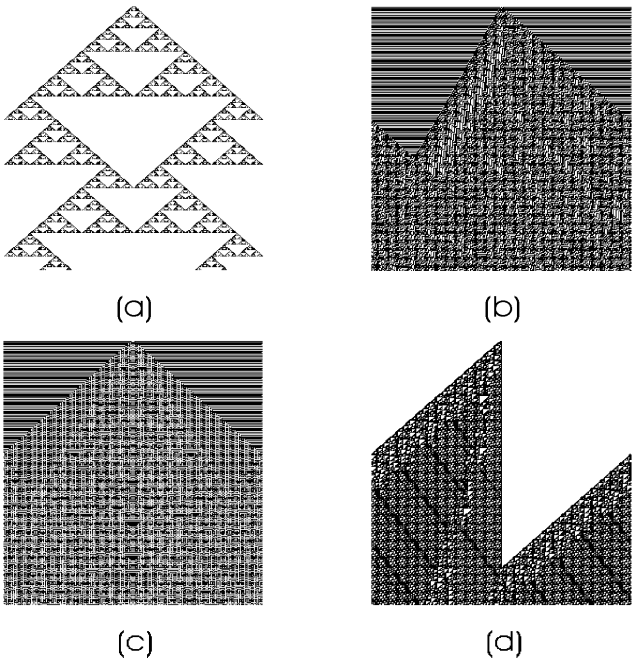

Figure 2. Illustration of 370 time steps of the evolution of cellular automata with N=320 sites, according to rules (a) 18, (b) 45, (c) 73, (d) 110.

8 Parallel Performance

We now present and discuss the parallel performance measures obtained with our PVM and MPI implementations. In Fig. 3 we show the total execution time, T, required to compute 500 time steps of the evolution of a cellular automaton with N sites, using P processors of our cluster; the execution time is measured with the Unix `time` command, and is given in seconds. In Fig. 4 the speed-up, S, defined as the execution time for one processor divided by the execution time for P processors, is shown as a function of P; the speed-up results are obtained for 500 time steps of the evolution of two cellular automata, one with N=1.5x10^6 sites and the other with N=4.5x10^6 sites. Several observations can be drawn from these figures. First, it is clear from Fig. 1 that parallel computation of cellular automata evolution with either the PVM or the MPI implementation is advantageous with respect to the corresponding serial computation, which has higher execution times for all N in the range 5.0x10^5 to 4.5x10^6. Second, it is observed from Fig. 2 that, for fixed P, the speed-up increases as the number of sites is increased; this behavior is explained by the fact that the amount of computation on one processor increases when N is increased, whereas the amount of communication remains fixed. Third, for fixed N, the speed-up increases as the number of processors is increased from 1 to 3; the speed-up curve for the PVM implementation, contrary to the curve for the MPI implementation, is beginning to level-off, indicating that the latter will benefit more than the former if the number of processors is increased beyond 3. Finally, we observe from Figs. 3 and 4 that the MPI implementation has a better parallel performance than the PVM implementation; this is mainly due to the 'communicators' used in the MPI protocol (Snir *et al.*, 1996), which may completely eliminate intermediate buffering when sending and receiving messages (as opposed to the PVM protocol). As a last comment, we obviously expect that the performance of both the PVM and MPI implementations would improve with better (i.e., faster) communication hardware (Geist *et al.*, 1994), such fast Ethernet devices.

9 Results for Cellular Automata with Large Numbers of Sites

In order to illustrate the enhancement of computing power afforded by parallel processing to simulate the time evolution of cellular automata, we now

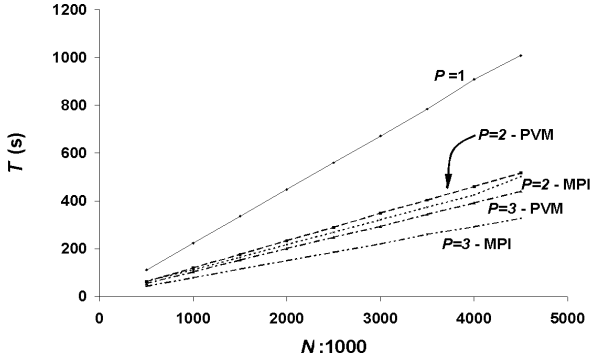

Figure 3. Total execution time, T (s), as a function of the number of sites, N, required to compute 500 time steps using P processors.

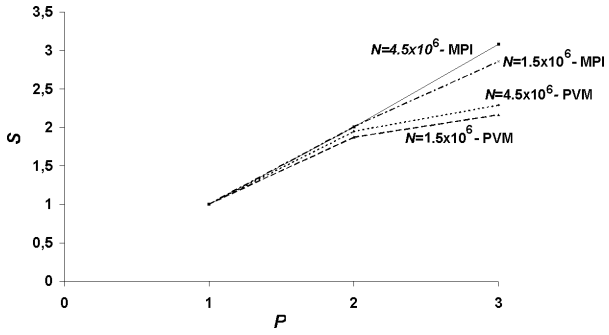

Figure 4. Speed-up, S, as a function of P obtained for 500 time steps of the evolution of two automata (N=1.5x10^6 and N=4.5x10^6 sites).

show some results obtained for the density of automata with one billion sites, N=10^9, for 100 time steps. For the machines of our cluster, serial simulations with N=10^9 are impossible due to memory limitations. Density D is a statistical local property defined as the number of sites at which the value of V is 1 divided by N, $0 \le D \le 1$ (Wolfram, 1994; local and global properties characterize, respectively, individual and the ensemble of all possible configurations of a cellular automaton).

In Fig. 5 we show a plot of D versus time t for the complex rules 18, 90, 110, and 182; at time t=0, the state is disordered and the density is D_0=0.5 for all rules. We observe that all rules lead to asymptotic values for D as time is increased, as predicted by Wolfram (1994). Finally, in Fig. 6 the density D is shown in the course of the time evolution of cellular automata according to rule 22, starting from three different disordered initial states, with D_0 taking on the values of 0.1, 0.2, and 0.3. We note that, irrespective of the value of D_0, the density D for rule 22 approaches the same asymptotic value of 0.351. As pointed out by Wolfram (1994, p. 23), no simple domain structure appears with rule 22, rendering impossible a rule transformation approach to derive the exact asymptotic value of D; Wolfram (1994) thus estimates, based on simulations of automata with relatively small numbers of sites, such asymptotic value to be 0.35±0.02. Based on our simulations of cellular automata with large numbers of sites, we are able to confirm his estimate to within 0.3%.

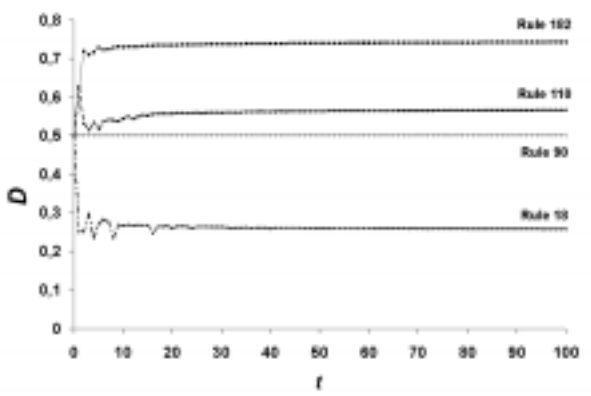

Figure 5. Density D versus time t for the complex rules 18, 90, 110, and 182; at time t=0, the state is disordered and the density is D_0=0.5 for all rules.

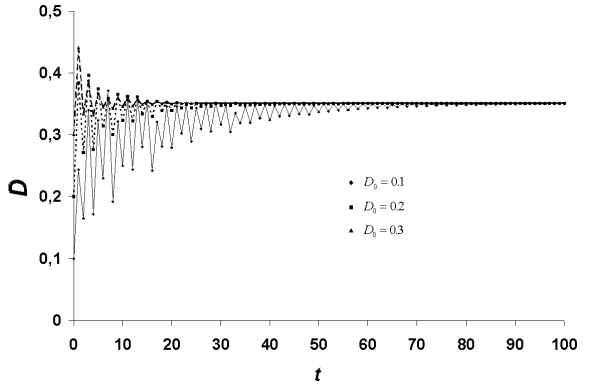

Figure 6. Density D versus time t for the complex rule 22, starting from three disordered initial states with D_0 equal to 0.1, 0.2, and 0.3.

We would like to conclude by stating that we have successfully utilized a low-cost parallel computing environment, based on the UNIX operating system and the communication protocols PVM and MPI, to compute the time evolution of one-dimensional cellular automata with large numbers of sites. Extensions of this work to multi-dimensional cellular automata is possible and promising.

Acknowledgements: The support of this work by the Brazilian agencies CNPq, through Grants 521002/97-4 and PIBIC/UFRJ-1998/0063, and

FAPERJ, through Grant E-26/170.889/97-APQ1, is gratefully acknowledged. All the runs were conducted at LEPAC/DEM-EE/COPPE/UFRJ.

References:

[1] Balay, S., Gropp, W., McInnes, L. C. and Smith, B., "*PETSc 2.0 Users Manual*," freely available at http://www.mcs.anl.gov/petsc, 1997.

[2] Bernardes, A. T. and Zorzenon dos Santos, R. M., "Immune Network at the Edge of Chaos," *J. Theor. Biol.*, Vol. 186, 1997, pp. 173-187.

[3] Dallalana, A., Costa, B. J. and Cruz, M. E., "Parallel Microcomputing; Application to Heat Conduction in Thermal Composites," *Proceedings of the V North-Northeast Mechanical Engineering Congress (CEM-NNE)*, Vol. 3, 1998, pp. 315-322, Fortaleza, CE, Brazil.

[4] El-Rewini, H. and Lewis, T. G., "*Distributed and Parallel Computing*," Manning Publications Co., CT, 1998.

[5] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V., "*PVM - Parallel Virtual Machine, A user's guide and tutorial for networked parallel computing*," The MIT Press, Cambridge, MA, 1994.

[6] Leiss, E. L., "*Parallel and Vector Computing, A Practical Introduction*," McGraw-Hill, Inc., New York, NY, 1995.

[7] Rothman, D. H. and Zaleski, S., "*Lattice-Gas Cellular Automata, Simple Models of Complex Hydrodynamics*," Cambridge University Press, Cambridge, 1997.

[8] Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W. and Dongarra, J., "*MPI: The Complete Reference*," The MIT Press, Cambridge, MA, 1996.

[9] Stauffer, D., "Computer Simulations of Cellular Automata," *J. Phys. A: Math. Gen.*, Vol. 24, 1991, pp. 909-927.

[10] Toffoli, T. and Margolus, N., "*Cellular Automata Machines*," The MIT Press, Cambridge, MA, 1987.

[11] Welsh, M., "*Linux Installation and Getting Started*," Version 2.2.2, freely available at http://www.unix-ag.uni-kl.de/docs, 1995.

[12] Wolfram, S., "*Cellular Automata and Complexity*," Collected Papers, Addison-Wesley Publishing Co., Reading MA, 1994.