CAMELot: A Cellular Automata Simulation Environment

KOSTAS KAVOUSSANAKIS
Edinburgh Parallel Computing Centre
University of Edinburgh
James Clerk Maxwell Building, Mayfield Road, Edinburgh EH9 3JZ,
SCOTLAND

Abstract: - Soil Bioremediation is the process by which bacteria are used to accelerate and improve the natural degradation of soil contaminants. This process is currently applied in many European, American and Middle East countries. Bioremediation is environmentally safe and cheaper than other decontamination methods, nonetheless it is still expensive and forecasting the result is difficult. Although accurate modelling of the phenomena involved can reduce the cost and time factors and improve the quality of the process, very few bioremediation companies apply such methods. The ESPRIT funded CABOTO project, introduced the use of computers for modelling bioremediation. *Cellular Automata* were used to model the interaction of the bacteria with a specific contaminant. A purpose driven language for CA specification, called CARPET was developed to this end.

CABOTO finished in 1996 and was followed by COLOMBO. The major differences between the two projects include the simulation of a wider class of phenomena, involving a wider range of contaminants and real field testing. It was also desired to make the transputer-based simulation software portable across platforms and improve its usability and functionality.

EPCC is a leading UK High Performance Computing Centre with a strong track record of providing services and facilities to industry. It participates in COLOMBO providing expertise in Parallel Programming. The objective is to design and develop CAMELot, an improved environment for Cellular Automata programming and parallel execution, and implement the necessary extensions to CARPET so as to meet the end-users' needs. In this paper we give a short outline of the Cellular Automata modelling framework and focus on the design of the CAMELot system. We also give an overview of the supported functionality and discuss the most important implementation issues. IMACS/IEEE CSCC'99 Proceedings, Pages:5521-5526

Key-Words: - Parallel Execution, Computer Simulation, MPI, Sockets, Cellular Automata, Bioremediation.

1 Introduction

Bioremediation is a process by which bacteria are used to degrade the contaminant in polluted soils. This process improves and accelerates the natural decontamination process. Pilot studies are executed in laboratories to predict the outcome of the application of the method to the target grounds. In addition to those, the phenomena are modelled using mathematical computation methods. The physical, chemical and biological phenomena involved in the process are complex and require large computational power in order to be modelled. Parallel computers can provide this computational power in a cost-effective way.

Cellular Automata [1] is a framework than can be used to model the process. Cellular Automata are discrete space and time, dynamical systems. A Cellular Automaton (CA) consists of an *n*-dimensional grid of cells (*n* usually not exceeding 3), each of which can be in one of a finite number of *k* possible states. The state of the cells is updated synchronously in discrete time steps according to a local, identical interaction rule, called the *transition function*. Each cell is associated with a set of *neighbour* cells defined by their distance from it. The state of each cell depends on the previous state of its neighbouring cells (and itself) [2]. The CA approach to computational modelling involves describing complex, continuous systems by means of the interaction of simple cells which follow simple rules. CA models are inherently paralellisable because of the locality that they exhibit.

COLOMBO is an ESPRIT collaborative project which aims at «the application of parallel computing to the simulation of the bioremediation of contaminated soils». It is a follow-up of the previous ESPRIT project, *CABOTO*[8]. COLOMBO aims to improve the CABOTO products and results both on the bioremediation and on the computing side. The consortium includes British, Italian and German companies specialising in the areas of contaminated soils remediation and IT and parallel computing.

The software implemented during the CABOTO project is called CAMEL (Cellular Automata environMent for systEms modeLling) [3]. It was implemented on a PC-hosted transputer system. Model development was performed on a Windows based graphical interface and visualisations of the model evolution were displayed on a dedicated framebuffer. A purpose-driven language for CA programming called *CARPET* (CellulAR Programming EnvironmenT) was also developed [7].

EPCC is a leading UK HPC research centre. The mission of the centre is to «accelerate the effective exploitation of high performance parallel computing systems throughout academia, industry and commerce». EPCC was not involved in the CABOTO project. Its participation in COLOMBO is to redesign CAMEL, so as to make it portable across various architectures. It also aims at extending the CARPET language capabilities.

This paper discusses the design of the emerging CAMELot (CAMEL with Open Technology) software and gives an overview of the functionality of the software. It also highlights the most important features of the implementation.

2 The CAMELot Software Design

CAMELot is an environment for the programming and parallel execution of Cellular Automata. It offers a programming environment and a Graphical User Interface to interact with the system while running a simulation, view visualisations of the simulated data and post-processing of the output of the run. The system supports the CA specification language CARPET.

Although most of the CAMELot code has been designed and written from scratch, the structure of the product was dictated by that of CAMEL. It consists of the following three major components:

- *The CA Engine*, comprising the CA Engine *Harness* and the CARPET-defined CA model;
- *The Graphical User Interface (GUI)*, including the GUI/CA Engine communication library;
- *The CARPET Parser*, integrated with the GUI and invoked by means of a button.

The main design aim for CAMELot was to come up with a system intended to run on a Quadrics (formerly MEIKO) CS-2, a massively parallel computer, yet able to run on smaller, even single process computers. Portability of the GUI was also an issue.

It was thus decided to implement the CA Engine harness using MPI-1 [4]. MPI stands for *Message Passing Interface*. It is a standard for writing message-passing programs, developed with the aim to provide a «practical, portable, efficient and flexible standard». It is available on most UNIX platforms and Microsoft Windows.

The CAMELot GUI is a Motif application written using Imperial Software Technology's X-Designer 4.6 GUI builder tool, which is a tool for designing GUIs graphically. This tool generates in this case C code which implements a GUI using the Motif library.

This approach was taken to allow rapid prototyping and development. Since X-Designer also has facilities for generating C++ (using Motif or Microsoft MFC libraries) or Java (using the AWT library), this may also ease any future porting of CAMELot to other platforms.

The communication between the GUI and the CA Engine is done using BSD sockets. The reason for choosing sockets is that they provide a simple programming interface. MPI-1 could not be used because its specification does not allow processes to start at different times, which is essential for the application since the GUI spawns the macrocell processes. The reason for choosing Berkeley sockets instead of TLI is that they have been established over the past years and are widely supported across platforms [6]. Our protocol was implemented over TCP, which provides a bi-directional, connection oriented channel of communication

The CARPET parser translates CARPET programs into C programs that define the transition function of the CA. It is composed of a tokeniser and a parser generated using the UNIX tools `flex` and `yacc`

(or `bison`). This parser is derived from the one developed in the CABOTO project, yet the language and therefore the parser are continuously being updated to meet the users' needs.

3 Functionality of the System

In order to execute a CARPET program, the user must do the following:

- *Compile* it, so as to invoke the parser and translate it to C;
- *Build* it, so as to define the size of the model and its decomposition to a number of processes and create the CA executable;
- *Run* it.

The user can initialise the state of the CA model using appropriate files. They can also instruct the system to perform periodic or one-off file dumps of the state of the system, which can be uploaded at a later time or post-processed. Periodic visualisations of the substates of the system are also available for the whole model or a subset of the cells. The user may initiate an infinite loop or define a number of iterations to be run and can pause or terminate the run using buttons. Off-line execution not involving the GUI is also supported but with limited functionality, since visualisations and interaction with the user are not available.

The CAMELot environment supports 3 different types of Windows:

- The *Development* Window;
- The *Simulation* Window;
- The *Visualisation* Window.

3.1 Development Window

The Development Window consists of a Menu Bar, the CAMELot Editor and a 3-Button Bar. The Menu Bar contains the following options:

- *File*, which enables the user to open and save a CARPET file. The user may also exit the Development Window, thus quitting the program;
- *Edit*, which involves the usual Cut, Copy and Paste functionality[1];
- *Configure*, which allows the user to define the size of the model and its decomposition, and set the C compiler command line and the MPI run command arguments.

The Editor Window is an 80x24-character window with scrollbars in either dimension which allows the user to modify their program.

The Button Bar consists of 3 buttons which are associated with the Compile, Build and Run functions respectively. The Compile and Build buttons pop up a window which contains feedback concerning the respective actions, whereas the Run button spawns the CA processes and pops up the Simulation Window.

3.2 Simulation Window

The Simulation Window consists of a Menu Bar, a Display Sub-window and a 5-Button Bar.

The Menu Bar contains the menus *State* and *Setup*. The former allows the Initialisation of the state of the system from files on the user's filesystem and one-off saves of such files. It also contains a Close option to close the Window and terminate CA Engine execution. The *Setup* menu options enable the user to set the number of iterations to be run and the periodic storage interval and to edit the substate of a single cell. It also allows the global modification of the CARPET program *parameters*. The user may also set the minimum and maximum values for the colour mapping and the first and last *active fold*; these two facilities are discussed later in section 4 of this paper.

The Display Sub-window provides online information about the size of each of the dimensions of the model, the current CA iteration, the storage interval and the number of *folds*.

The Button Bar contains the following buttons:

- *Go*, to start a CA Engine run with a given number of iterations;
- *Loop*, to start an infinite run;
- *Pause*, to temporarily stop the CA Engine execution;
- *Resume*, to restart the execution after Pause has been pressed, without resetting the current and final iteration as Go and Loop would do;
- *Visualise*, to define a visualisation entity and fire

[1] At the time of writing a text search facility is designed.

up a Visualisation Window;

- *IVT*, reserved for post processing of data using AVS/Express.

3.3 Visualisation Window

This Window consists of a Visualisation Space and a 2-Button Bar. The spatial specification, substate name and visualisation interval of the visualised entity which have been defined by the user when clicking on the Simulation Window *Visualise* button are displayed on the X-Windows title bar of the Visualisation Window.

The Visualisation space contains a 640x640-pixel visualisation area, a strip displaying the colour palette currently used and a strip relating the minimum and maximum values of the substate visualised with the palette strip. If the visualised entity is larger than the available area, it is sampled at regular spatial intervals, without averaging over the interval.

The Button bar contains a *Colours* button enabling the user to load a palette from a file following a specified format, and a *Close* button which closes the Window.

4 CA Engine Harness Internals

The CA Engine component of the program performs the evolution of the model specified in the CARPET program. As mentioned in section 2, the CA Engine is implemented as a parallel program adhering to the *Single Program Multiple Data* paradigm. Each process thus created is called a *macrocell* and can apply the transition function of the model locally to a subset of the model, under the assumption that it holds locally all the data that it requires. This suggests the introduction of boundary data which are maintained in neighbouring macrocells and communicated to the process after each evolution.

This communication is implemented using MPI. The CA Engine also needs to communicate with the GUI. This communication is performed between one CA Engine process and the GUI using our purpose-built sockets protocol.

4.1 Data Representation

Each cell can be thought of as a 3-D (x, y, z) triplet of co-ordinates with an associated set of substate values. The CA Engine is represented as an array of cells, each of which is implemented as a C struct having as members the substates. This *contiguous cell* approach aims at improving the performance of the transition function application, benefiting from data caching and read-ahead optimisations exhibited by modern processors. It should be noted that because the state of a cell depends on the previous state of its neighbourhood, two such copies of the model are maintained.

4.2 Data Decomposition

If the program is run on more than one process we arrange them in a linear Cartesian topology. In this case each process contains a fraction of the model data. The data are decomposed across the x-axis of the model. No padding mechanism has been implemented.

The CABOTO project introduced a form of block-cyclic decomposition aiming to reduce load imbalance between processes. The idea, which was implemented in CAMELot as well, was to split the model in *folds* and then assign equal parts of each fold to each of the processors. This can lead to load balancing under the condition that the resulting granules (further referred to as *strips*) are fine enough to ensure uneven load distribution across folds is insignificant statistically across processes. It should be noted though, that the more the strips, the bigger the communication overhead among the processing elements.

4.3 Boundary Replication

It was decided to implement cyclic boundaries. This means that data are replicated across the axes to ensure cyclic interaction and execution of cells. Moreover, because of the data decomposition, physically neighbouring strips are allocated to different processes. Duplicate cells are annexed on either x-side of the strip to enable local execution of the CA rule.

The above approach suggests four types of halo:

- before the first and after the last real element (z-axis);
- between planes (y-axis);
- between lines (x-axis);
- between strips (folded data).

The first three halos conceptually form a shell

around the model, whereas the last increases its *x* dimension. There are two different kinds of boundary replication. Across the *y* and *z* axes the data can be replicated internally in macrocells. We call this a *boundary copy*. Across strips (this effects *x*-axis halo replication as well) the data must be exchanged between consecutive processes using MPI. We call this *boundary swap*. Boundary data are copied across axes and swapped between strips after the execution of a CA Engine generation.

4.4 Execution of the Transition Function

The transition function is applied to all the cells of a process without any interruption for servicing User request, thus preventing race conditions in the calculation of the state values. The processes communicate after the transition function has been applied to *all* the cells for boundary communication. Each macrocell loops over the strips skipping the halos and applies the parser-generated transition function to its cells. During this phase the cells in strips are examined in order to decide their activity status, as explained in section 4.5.

4.5 Automatic Inactive Strip Detection

CAMELot contains an *automatic inactive strip detection mechanism*, used to isolate inactive regions and avoid applying the function to idle strips. The block cyclic decomposition suggests that load imbalance emanating from this strategy will be insignificant in the general case, given that contiguous areas of the model are transparently distributed to processes. Automatic inactive strip detection can be disabled by manually choosing a set of active folds from the corresponding Simulation Window *Setup* menu option. The finest grain in this case is the fold, which is generally larger than the strip, and the strategy is error prone as it depends on the user's vigilance. Moreover, manual fold selection cannot isolate inactive regions located in the middle of the model even if the granularity suffices, because the active range defined is continuous.

Initially, all the strips are considered active. The system attempts automatic inactive fold detection under the condition that the user has characterised the transition function as deterministic (i.e. the transition function is not applied with a random rule) through the CARPET statement `deterministic`. Additionally, the user may define a `threshold` condition to denote if a cell has reached an insignificant state.

After the transition function has been applied to a cell, the cell is checked against the CARPET parser generated `cpt_threshold()` function and the previous values of all its substates. If a substate has changed and `cpt_threshold()` returns false, then the whole strip is characterised as active for the next generation and the check is not performed for any other cells of the strip.

Even if all the cells of a strip are classed as inactive, the strip is not considered inactive unless the boundary cells to be received by each of the neighbours are also inactive. During boundary swap the processes exchange information with their neighbours about the activity status of the incoming boundaries and combine the results with those emanating from the internal cell check to decide on the activity status of their strips.

5 Visualisation

The CA Engine can be instructed to transmit periodically Visualisation data for a defined entity. The implementation of the visualisation functionality required the introduction of various data structures on the GUI and the CA Engine. The protocol for the maintenance of the visualisation entities is slightly complicated because of the variety of possible events. Moreover, a colour mapping strategy was devised.

5.1 Data Structures

The finest granule of the visualisation entities is a 2-dimensional plane, except if the model is linear, in which case the granule is a line. The representation of planes includes the substate and the visualisation step so as to enable detection of equal planes or planes that could be visualised more than once in a given step. The former are rejected from the CA Engine, the latter are only transmitted once.

5.2 Plane Visualisation

The CA Engine maintains a list of planes ordered with respect to the next iteration in which they are due to be visualised. When a plane is to be visualised, it is popped from the list, the substate data are gathered in the root process and are transmitted to the GUI using a dedicated socket. The plane is finally reinserted in the list.

5.3 Colour Mapping

CARPET supports all C datatypes and arrays of these as possible substate types. It is therefore essential to perform colour-mapping of these before visualising them.

The colour mapping is performed in all the processes before gathering the data at the root process so as to take advantage of the parallel execution. In order to perform the colour mapping the processes traverse the cells which are going to be visualised according to the plane specification, and seek the minimum and maximum values for the substate visualised. The minimum and maximum values are stored as double precision numbers globally in the processes. Their values are updated every time the substate is visualised and they are maintained *throughout the life* of the program. By doing this we generally make the mapping consistent for the planes throughout the life of the program and indicate how the substate changes with respect to time.

It is worth noting that, because the granule of visualisation is the plane, 3-D models are broken down to planes on the GUI side in order to visualise them. Therefore, in the first step of the visualisation the first plane of the cube visualised possibly sets the minimum and maximum values to something different than the next planes and could be displayed erroneously; in the next visualisation the minimum and maximum values and therefore the colour mapping, are updated, «converging» to the correct values. The minimum value of the substate for the visualised plane is mapped to 1 and the maximum is mapped to 255. The intermediate values are linearly projected to the [1,255] interval. The above mapping leaves 0 as the background colour for 3-D visualisations. In the palettes distributed with CAMELot, this corresponds to Black.

This automatic colour mapping strategy can be overridden. The automatic strategy implies that the whole range of values is of equal importance. Nonetheless, in some cases the user needs to focus on a specific range of substate values ignoring the rest. The system thus provides an option under the *Setup* menu of the Simulation Window, by means of which the user can set the range of its interest. All values below the user-specified minimum are mapped to the 1 and all those above the maximum are mapped to 255, thus visualising the intermediate range in greater detail.

6 Conclusion

CAMELot is an environment for Cellular Automata programming and parallel execution. It is developed in the context of soil bioremediation; nonetheless it is a general problem-solving tool. The system contains a purpose-built CA programming language, called CARPET.

In this paper we gave a list of the necessary functionality for such an environment. We also discussed details of its design and implementation and highlighted the most interesting features of the language and the system.

Approximately 75% of the two-year duration of the project has elapsed at the time of writing, and the programme is in the phase of soliciting and implementing extensions. Frequent feedback from the end users also helps to improve the stability and robustness of the product.

References:

[1] J. von Neumann, *Theory of Self-Reproducing Automata*, University of Illinois Press, Illinois, 1966. Edited and completed by A.W. Burks.

[2] T. Toffoli and N. Margolus, *Cellular Automata Machines*. The MIT Press, Cambridge, Massachusetts, 1987.

[3] S. di Gregorio and R. Rongo and W. Spataro and G. Spezzano and D. Talia, A Parallel Cellular Tool for Interactive Modeling and Simulation, *IEEE Computational Science and Engineering*, Vol.3, No.3, pp.33-43.

[4] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, Version 1.1, June 1995.

[5] K Kavoussanakis, S D Telford and S P Booth, *COLOMBO WP3: CAMELot Implementation*, Deliverable DI3.3.3, Version 1.0, March 1999. Restricted Availability.

[6] W.Richard Stevens, *UNIX Network Programming*, Prentice-Hall Software Series, 1990.

[7] G. Spezzano and D. Talia, Designing parallel models of soil contamination by the CARPET language, *Future Generation Computer Systems*, Vol. 13, 1998, pp 291-302.

[8] D. Talia, Cellular Automata Thrive on Parallel Systems, *Scientific Computing World*, October 1998.

[9] Project COLOMBO, *Project Programme*, Version 2.1. Restricted Availability.