An Interactive Problem Modeller and PDE Solver, Distributed on Large Scale Architectures

Nicolas Fressengeas, Hervé Frezza-Buet, Jens Gustedt, Stéphane Vialle

An Interactive Problem Modeller and PDE Solver, Distributed on Large Scale Architectures

Nicolas Fressengeas*, Hervé Frezza-Buet**, Jens Gustedt***, Stéphane Vialle**
* LMOPS and Metz University, France, *Firstname.Lastname@univ-metz.fr*
**SUPELEC, France, *Firstname.Lastname@supelec.fr*
***INRIA Lorraine and LORIA, France, *Firstname.Lastname@loria.fr*

Abstract

This paper introduces a research project and a software environment to speed up and size up problem modeling with Partial Differential Equations (PDE). These PDE are defined from a Mathematica™ interface, and are automatically solved by a devoted cellular automata program generated to be run on mainframes, clusters and Grids. Moreover, an interactive and graphic control of the cellular automata running allows to analyze the PDE model relevance. This environment improves large scale simulation usage in early stages of research projects.

1 Motivations and objectives

Simulations are currently used to study complex systems, and large simulations become mandatory in the early stages of any investigations as systems complexity increases. But large simulations are resource consuming at runtime, and require often dissuasive development effort on large mainframes, clusters and Grids. Moreover, end users like physicists are not expert in distributed programming, and prefer to use high level scientific tools like Mathematica™ to investigate problems.

This difficulty to early run large simulations is especially noticeable when investigating new real and complex problems: when no model is available and no specific simulation tool has been designed. Then, the research starting can be long, and the design of a relevant simulator can become a full research topic! Nevertheless, many problems can be modeled with Partial Differential Equations (PDE) difficult or impossible to solve and require large simulations.

The goal of the research project introduced in this paper is to design and implement a complete environment interfaced with usual scientific tools (like Mathematica™), allowing to easily model a new problem with PDE, and to generate automatically the corresponding interactive PDE solver on large scale distributed architectures. So the researchers can enter at once a friendly and interactive *modeling-evaluating loop* to design a relevant model of their real problem (see steps 1 and 2 in Figure 1). Then, they can generate a non-interactive simulator and run series of more classical simulations in batch mode (step 3 in Figure 1).

To reach this goal several scientific locks have been addressed in this project, and three main software tools have been developed:

Escapade: a generic PDE solver based on gradient computations and large cellular automata computations. Features of the cellular automata are designed by a Mathematica™ plug-in, and the automata are finally implemented on mainframes and large distributed architectures in C++.

parXXL: a framework to implement fine grained computations on large coarse grained architectures implemented on shared memory mainframes, cluster and Grids. It implements and manages the large cellular automata required to achieve PDE solving.

Grumpf: a framework to implement and control interactive fine grained computations, based on

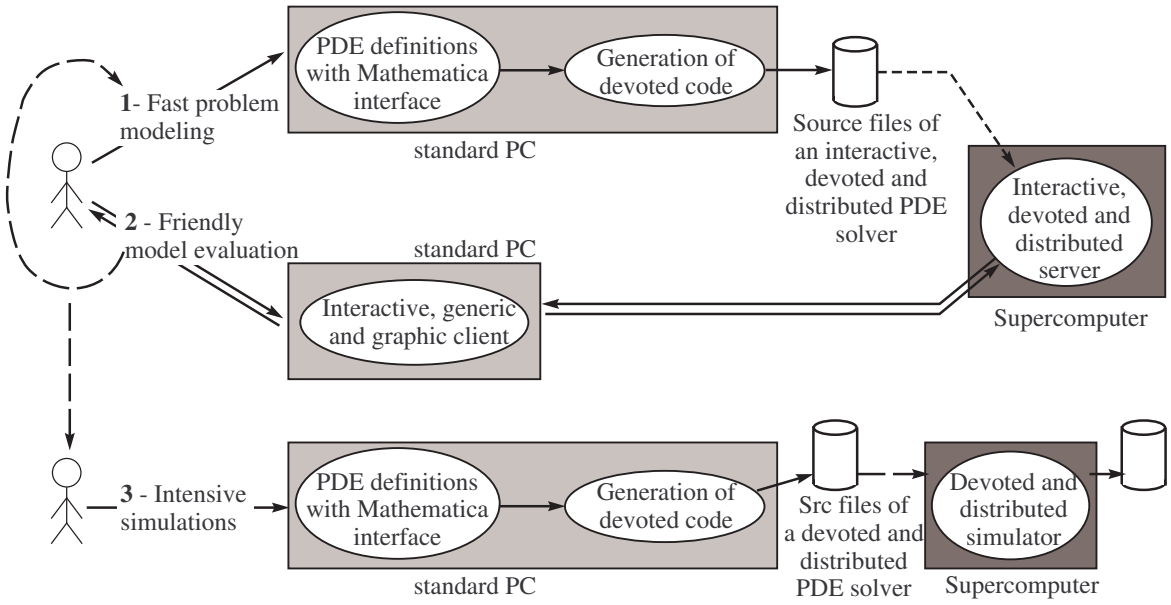

Figure 1: A complete environment for fast and friendly problem modeling and solving.

a client-server paradigm and including many graphical investigation tools. It implements the cellular automata computations required by Escapade on parXXL distributed framework.

2 Previous and related work

A previous version of the cellular automata generated by Escapade was implemented only on top of Grumpf, which in turn was implemented on top of ParCeL-6: a fine grained and parallel programming environment limited to shared memory multiprocessors. ParCeL-6 and Grumpf were initially designed to allow the development of biologically inspired neural networks [7], a complex kind of fine grained computations. Then ParCeL-6 and the optimized communication library SSCRAP [2] have been merged and have led to the more generic and powerful parXXL distributed framework, available on various parallel and distributed architectures [4]. So, the new version of the software suite Escapade +Grumpf +parXXL will process larger problems on large distributed architectures.

Many development environments and tools exist to distribute and scale scientific computations. GridMathematica^{TM} is a Mathematica^{TM} plug-in mainly designed to parallelize multi-parameter investigations. But it does not include a generic PDE solver, and does not allows to automatically speed up or size up PDE solving. *Network Enabled Servers* like DIET [1] have been designed to run programs including *Grid remote procedure call* to achieve large and time consuming computations, and have been interfaced with Mathematica^{TM}. But they do not automatically solve PDE, nor run fine grained computations (a Grid-RPC has to be a coarse computation to achieve efficiency). More complex Grid environments, such as GridLab [9], are generic high-level middleware. They address many issues, like scheduling, load balancing and fault tolerance, but delegate scientific computations to the different tools or *Problem Solving Environments* they support, and distribute coarse grained tasks. Finally, some Grid environments focuss on scientific domains and usual numerical computations, like OpenSees [5] for earthquake engineering using finite element methods, but do not supply generic PDE solvers.

Subsequently, our software suite could be interfaced with GridLab or DIET environments, to become a generic PDE solver to be called *on demand*

for various kinds of large Grid applications.

3 Architecture overview

Figure 2 illustrates the global software architecture of our environment to quickly model and solve various problems based on PDE.

Escapade is the acronym for Ergonomic Solver using Cellular Automata for solving of PArtial Differential Equations. It consists in two pieces of software called Dempf and Simpf which respectively stand for Differential Equation Mapping for SimPF and SIMulation with grumPF. Dempf is implemented as a Mathematica^{TM} package, the goal of which is the design of a continuous automaton specifically adapted to a given discretized differential problem given in a Mathematica^{TM} formalism. It works by transforming the differential problem solution seeking scheme into an error minimization one to be performed over the cell network. The calculations involved in the process are local only (see *e-print math-ph/0610037* [3]) and are thus to be implemented in a Cellular Automaton.

parXXL is a framework mainly designed to easily implement fine grained computations on coarse grained parallel architectures. It is based on 8 software layers and some portable runtime mechanisms (see the bottom of Figure 2). These different layers allows to choose a predefined *cellular network template*, to define some *cell behavior functions*, to run some *supersteps* with relaxed synchronization, and to ensure efficient and large memory management and communications (see Section 5). parXXL achieves to create and run the very large cellular automata required by Escapade, equally on shared memory machines, on clusters and on Grids. A dynamic runtime support optimizes the execution on any coarse grained architecture, choosing automatically the most efficient runtime. Moreover, a parXXL application can be a sequential server controlling a set of worker processors and interacting with a Grumpf client (Figure 2, on the right).

Grumpf adds more utilities and interfaces to parXXL framework. At a programming point of view, it allows to define a set of cell updating procedures by inheritance from a general class. Instances of such subclasses are to be gathered within maps, that are 2D arrays of cells used for viewing purpose. A programmer thus defines first some cell classes, then creates instances of them that are put within maps, and connect them together. Once this is done, Grumpf allows to start a TCP/IP server where the computation of the whole network is managed, based on parXXL functionalities. Clients may trigger execution of the server step by step, but also file saving and visualisation. The advantage of Grumpf is that the programmer only cares about designing updating rules and how to link cells together. The TCP/IP part, including remote control and viewing, as well as parallelization by parXXL, are hidden. The *ad hoc* generated automata obtained from Escapade is then a parallel process that can be queried from remote clients for ergonomic viewing and manipulation.

4 Escapade: solving PDE with cellular automata

4.1 Principle of generic PDE solving

Our strategy for solving Partial Differential Equations builds upon the local nature of most such problems as follows: first, a differential problem is discretized by, for example, a finite difference method. This yields to an approximation of the problem by means of *cells of a discrete mesh* and a *relationship* that has to be verified among these cells. Finding the appropriate values in each cell that meet the discrete equations (*i.e.* which are consistent) solves the problem. The local nature of the differential problem implies that this consistency is expressed for each cell as a *function of its neighbors* only.

If the entire cell mesh is thought of as a unique vector Ω, a positive error function $\mathcal{E}(\Omega)$ can be designed so that the error is zero if and only if all the cells are filled with values that are actually consistent. In this way, the solution to the initial differential problem can be reached through the minimization of $\mathcal{E}(\Omega)$, its zero value being verified *a posteriori*. This minimization can then be carried out by an adaptation of the Newton's Method [8] as known from mathematical optimization.

4.2 Generated Cellular Automata

The design of the cellular automaton results from the implementation of Newton's Method. The com-

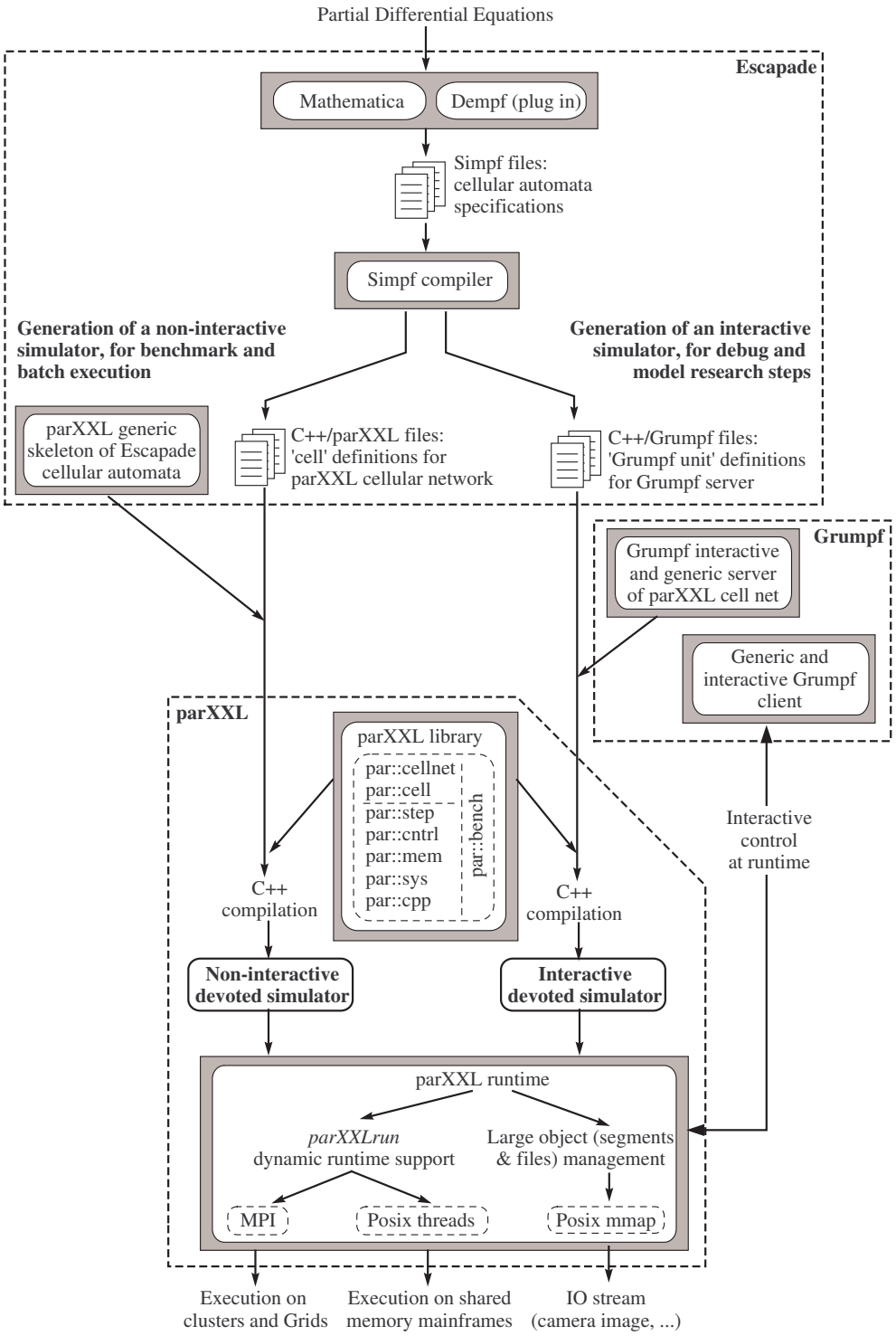

Figure 2: Global architecture of our interactive problem modeller and PDE solver on large parallel and distributed computers

putations that need to be carried out on Ω involve only the neighboring cells (see *e-print math-ph/0610037* [3]) and can be carried out independently without global knowledge. They can thus be described as cellular automata that are assigned to each cell of the mesh. The task of Dempf is to use the formal language of Mathematica^{TM} to automatically derive such an automaton from the equation. This automaton implements the problem specific optimization procedure and relaxes to the consistent state, which is the problem solution, as mentioned above. Our implementation of Dempf is currently restricted to use only 4 dimensions (time and space) but an extension to higher dimensions will be considered.

The output of Dempf is a set of text files that describe the cellular automata. These files are used by Simpf to produce C++ source files which use the Grumpf library or directly the parXXL library to implement the cellular automata.

4.3 Example of PDE solving

Let us now illustrate this with a simple academic example of solving a Poisson Equation for V in the three spatial dimensions and restricted to a cube, where the boundary conditions $\triangle V(x,y,z) = 0$ are set on the sides of the cube. The corresponding discrete problem is straightforward and is obtained through finite difference centered second derivatives on each dimensions of space, for the same space discretization δ. The automaton obtained from Dempf has 28 different update rules, to account for the boundary conditions. The most generic of them concerns the majority of the cells, namely those which are far enough from the boundaries (distance 3 or more). It is a centro-symmetric three dimensional convolution kernel, a two-dimensional slice of which is as follows:

$$V \longleftarrow \frac{1}{42} * V \left(\begin{array}{ccccc} & & -1 & & \\ & -2 & 12 & -2 & \\ -1 & 12 & 0 & 12 & -1 \\ & -2 & 12 & -2 & \\ & & -1 & & \end{array} \right)$$

When launched, the system converges to a fixed point. We obtain the same result as, in that simple case, can be obtained with more conventional methods. For a cube of $20 \times 20 \times 20$ cells, the *a posteriori* computed remaining error $\mathcal{E}$ (after normalization) is 7×10^{-3} for $\delta = 1$ and decreases with it.

5 parXXL: distributing fine grained computations

5.1 Cellular programming model

The parXXL framework includes 8 software layers, as shown in Figure 2. The top of parXXL is composed of 2 *cellular* layers:

`par::cellnet:` a library of optimized and generic cellular network templates. It allows to quickly implement ad hoc networks with optimized cell distribution on any number of processors. Processors are load balanced and they host neighbor cells in order to limit inter-processor communications. One of these templates is used to implement the 4 dimensional cubic network required by Escapade cellular automata.

`par::cell:` a set of functionalities and a programming model to design and implement fine grained computations. This layer allows to define *cell behavior functions*, to create *cells*, and to dynamically connect these cells to establish cellular networks. This last operation can be fastidious and is usually done by the `par::cellnet` layer.

Cell nets are executed cyclicly, and cells can communicate in a synchronous mode: cell inputs are all updated at the end of each computation cycle (*buffered mode*), or in a more asynchronous mode (*hybrid* mode): cells are split in different subsets and their cell outputs are routed to connected cell inputs at different time during each computation cycle. As the Dempf resolution method of Escapade is based on a stochastic gradient descent, a pure asynchronous evaluation should be chosen but would need *on demand* cell input update and would lead to poor efficient processor communications. At the opposite, synchronous evaluation can be implemented with efficient processor communication steps (at the end of each cell computation cycles) but could be inappropriate for Escapade computations. The *hybrid cell communication mode* has been designed both to satisfy Escapade computation

requirements and to allow efficient processor communications. Its efficiency will be tested during the next phase of the project.

5.2 Efficient implementation of BSP-like programming

The next 3 parXXL layers define some *strategic functionalities* to support any application development.

`par::step`: A set of utilities that help for the organization of supersteps (as in the BSP model) and implement fundamental algorithms such as, *e.g*, an efficient sorting routine.

`par::cntrl`: The control layer implements the basic control and communication task on top of the different run-times. Currently supported run-times are POSIX threads and MPI. They are generally not compiled into the executable but flexibly choosen through dynamic linkage at the startup of the program.

`par::mem`: The memory layer efficiently implements an abstraction from the address space of the underlying system that eases access to large objects (several GiB) independently whether or not they are realized on the heap (`malloc`), as shared memory segments (`shm_open`) or within a disk or NFS file. This functionality allows to integrate remote applicative IO (*e.g* user interaction or camera images) and transparent check-pointing.

Then parXXL includes 2 other layers implementing *basic functionalities* and insuring *portability*: `par::sys` for system-level interfaces and `par::cpp` for C++ interfaces.

Finally, to ease *benchmarking* in the development process, `par::bench` transversal layer supplies a set of tools to easily implement some performance measurements on parallel and distributed architectures.

Beyond these functionalities the parXXL cell programming model includes a client-server model. Each `par::cell` program is controlled by a sequential `main` routine executed on only one processor: the *master processor*, that installs the cell network and runs cell computation steps on all processors. This master processor can be specified explicitly such that the application may use it as a server interface (like a classic TCP server). This parXXL feature is used to implement the Grumpf server, running interactive PDE solvers controlled by the graphic Grumpf client and the user (see right part of Figure 2).

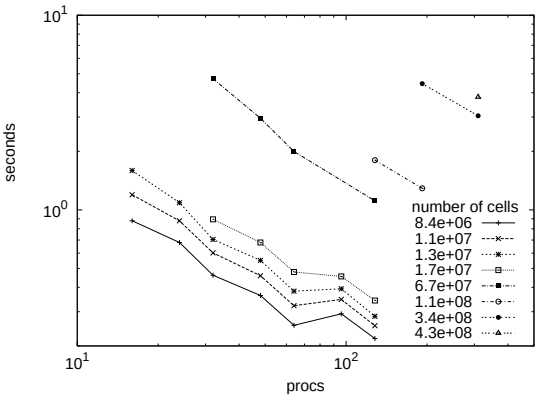

Figure 3: parXXL performances on a 3D Jacobi relaxation on "Grid-eXplorer" PC cluster.

5.3 Current performances

We have experimented parXXL on a 3D-mesh Jacobi relaxation run on a large PC cluster (the Grid-eXplorer machine). We have processed meshes up to 430 millions of points on 310 processors. Computations were simpler than required by Escapade cellular automata. But the cell network was defined in the `par::cellnet` library and used the same processor mapping principle than the 4D cell network defined for Escapade cellular automata.

Figure 3 shows the regular execution time decrease we achieved up to 310 processors, for different problem size. Finally it was possible to maintain the execution time per cycle and per cell using more processors constant while problem increased: our parXXL *scaled* up to 310 processors and 430 millions of points.

6 Grumpf: allowing interactive computations

6.1 Parallel programming model

The Grumpf programming model has initially be designed for the modeling of neural network-based cognitive systems and is still used for that purpose

[6]. In such a context, interactive visualization of computation is crucial, since emerging effects and relaxation dynamics are difficult to handle and debug. In the present framework, using Grumpf allows to benefit from neural network system design requirements, since Grumpf offers visualization, cell net serialization, and many other convenient utilities. It addresses three main issues:

Parallel cellular computations compatible with the fine grained cellular model of parXXL and the requirements of the Escapade 4D cellular automata.

Organization of the graphical rendering of the cell network activity, based on 2D maps grouping some network cells. Maps do not influence cell computation nor connectivity, but allow easy visualization and interpretation of the running cell network.

Client-server architecture to implement cellular computing servers, running fine grained computations *on demand*, and user friendly clients, allowing graphical visualization and interactive control of the cellular computations.

To set up a simulation, some cells have to be created, as well as the connections between them. The resulting graph is static, but may be arbitrary. A cell exposes an arbitrary (but constant) number of floating values, called activities. This number can change from one kind of cell to the other. A cell can access all the activities of a remote cell to which it is connected, but it is important to note that this is done read-only. In this framework the role of a cell is to perform successive evaluation steps, consisting of reading activities of connected cell, and writing consequently its own activities according to some updating procedure. Within the automaton, cells are referred by their position in some bidimensional array. The programmer defines then several arrays, called maps, and locates newly created cells in some free position in one of the maps. Maps do not influence the cell network, but organize graphical rendering of its activity.

6.2 Parallel implementation

Grumpf fully encapsulates the parXXL library to offer a more restricted programming model, and an

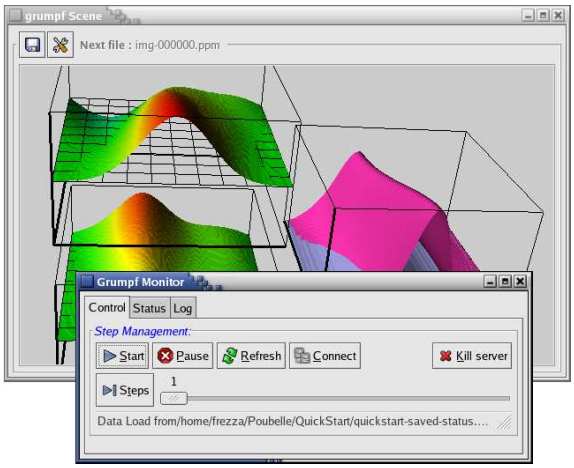

Figure 4: Grumpf interactive clients for execution control and visualization. The layout of visualization client is custom defined using glade from the Gnome project.

extended client-server mechanism, improving interactive visualization and control of the cell network activity. The Grumpf library offers a C++ interface for designing cells by setting some activities and updating rules. In the present framework, different cell classes correspond to the different updating rules computed by Dempf, due to limit conditions as mentioned in Section 4.3. Connections are kept local, as a consequence of the locality of the considered differential problem.

Based on parXXL functionalities introduced in Section 5.1, evaluation of the whole network can be done partially asynchronously using the *hybrid* cell communication mode of parXXL (as for the Hopfield neural model), or synchronously, using buffers and *buffered* mode (as for the Game of Life).

6.3 Interaction with a parallel run

The most significant extension to Escapade and parXXL offered by Grumpf is interactive control of the cellular automata running on large parallel and distributed architectures. The developer may directly control it from his work station. At the client side, no programming effort is required. Viewing tools are designed to visualize bidimensional maps of cells. The user can set up a visualization by using some utilities that define what is to be shown. For example, as maps at the server side are bidimen-

sional collection of cells, a query of activity number a in map m returns a mesh of floating values, that can be displayed either with 2D or 3D rendering (see Fig. 4). This display is updated when server state changes. In the present case, this rendering allows to view 2D slices of the 4D simulation.

7 Conclusion

This paper has introduced 3 software tools designed to solve PDE using cellular automata (Escapade), to run these fine grained computations on coarse grained architectures (parXXL), and to interactively control and visualize the cellular automata computations (Grumpf). These 3 tools have already been experimented and validated, and previous versions have been successfully integrated on share memory multiprocessors in 2004.

In order to achieve the integration of our new software suite, we are currently implementing the Simpf compiler: the Escapade module generating parXXL code for non-interactive computations or Grumpf code for interactive ones. In parallel, the design and implementation of the new Grumpf server have started, while the graphic Grumpf client remains unchanged. Moreover, some new minor functionalities of parXXL have been implemented to improve its integration with Escapade and Grumpf. Implementation and test of the *hybrid* cell communication mode (required for Escapade cell automata computations) remains the more sensitive issue.

We plan to achieve a non-interactive version before summer 2007 and a first complete suite for the end of 2007, allowing to quickly design a model based on PDE with Mathematica^{TM} and to automatically solve these equations and study the model using a mainframe or a large cluster.

Acknowledgment:

Authors want to thank Region Lorraine that has supported a part of this research.

References

[1] Eddy Caron and Frédéric Desprez. DIET: A scalable toolbox to build network enabled servers on the grid. *International Journal of High Performance Computing Applications*, 20(3), 2006.

[2] Mohamed Essaïdi and Jens Gustedt. An experimental validation of the PRO model for parallel and distributed computation. In *14th Euromicro Conference on Parallel, Distributed and Network based Processing*, 2006.

[3] Nicolas Fressengeas and Hervé Frezza-Buet. Generic method for solving partial differential equations through the design of problem-specific continuous automata. Technical Report math-ph/0610037, ArXiv.org, Nov 2006.

[4] Jens Gustedt, Stéphane Vialle, and Amelia De Vivo. parXXL: A fine grained development environment on coarse grained architectures. In *PARA-06: Worshop on state-of-the-art in scientific and parallel computing*, June 2006. Umeå, Sweden.

[5] F. McKenna and G. Fenves. The OpenSees command language manual: version 1.2. Technical report, Pacific Earthquake Engineering Center, Univ. of Calif., Berkeley, 2001.

[6] Olivier Ménard and Hervé Frezza-Buet. Model of multi-modal cortical processing: Coherent learning in self-organizing modules. *Neural Networks*, 18(5-6), 2005.

[7] Olivier Ménard, Stéphane Vialle, and Hervé Frezza-Buet. Making cortically-inspired sensorimotor control realistic for robotics: Design of an extended parallel cellular programming models. In *International Conference on Advances in Intelligent Systems - Theory and Applications*, 2004.

[8] Isaak Newton. De analysi per aequationes numero terminorum infinitas. In William Jones, editor, *Analysis per Quantitatum Series, Fluxiones, ac Differentias: cum Enumeratione Linearum Tertii Ordinis*. London, 1711.

[9] Ed Seidel, Gabrielle Allen, André Merzky, and Jarek Nabrzyski. GridLab: a grid application toolkit and testbed. *Future Generation Computer Systems*, 18(8), october 2002.