InterCell: a Software Suite for Rapid Prototyping and Parallel Execution of Fine Grained Applications

Jens Gustedt, Stéphane Vialle, Hervé Frezza-Buet, d'Havh Boumba Sitou, Nicolas Fressengeas

InterCell: a Software Suite for Rapid Prototyping and Parallel Execution of Fine Grained Applications*

Jens Gustedt[†1,3], Stéphane Vialle[‡2,3], Hervé Frezza-Buet[§2], D'havh Boumba Sitou[¶4], and Nicolas Fressengeas[‖4]

[1]*INRIA Nancy – Grand Est, France*
[2]*IMS Group, SUPELEC, France*
[3]*AlGorille INRIA Project Team, France*
[4]*LMOPS laboratory, Metz University and SUPELEC, France*

Abstract InterCell is an open, operational software suite for implementation, code generation and interactive simulation of fine grained parallel computational models. This article describes the software architecture, some use cases from physics and cortical networks as well as first performance measurements.

Keywords fine grained parallel models, interactivity, rapid prototyping

1 Introduction

The goal of the InterCell project is to help non-experts in parallel computing to use large scale parallel computers when developing models for physical phenomena, especially when these models are to be evaluated at large scale. To achieve this goal we developed a software suite that allows to rapidly design and implement parallel fine grained computing models on coarse grained parallel architectures, *e.g* clusters or mainframes. The InterCell development cycle typically has 4 stages: (1) rapid design of a mathematical model, (2) automatic implementation of a fine grained parallel simulator, (3) parallel run of large scale interactive simulations, and (4) rapid evaluation of the model.

Fine grained models of computation are widely adapted in different application domains. For our project this concerns two of these domains namely the modeling of physical phenomena that have some notion of 'locality' (spacial or timely) and the modeling and development of neuromimetic networks. But most likely InterCell could be useful for other domains as well.

Figure 1 introduces the InterCell software architecture. At top level users describe their problems with application domain tools, such as a PDE solver or a cortical inspired neural network simulator. These high level tools generate fine grained parallel simulators, using a C++ library named Booz. This library ensures the interactive control of the simulations and in turn uses the parXXL library. With that, it efficiently maps the fine grained computations on a coarse grained parallel architectures. The parXXL runtime hides the underlying parallel or distributed hardware. The final software has two parts: a parallel and interactive compute server and a set of easy-to-use control and visualization clients.

2 Fine grained parallel computations

Fine grained computations that act on statically structured data (generally matrices) are nowadays well mastered and can be parallelized on coarse grained architectures (typically multicore clusters) with good results.

The case we focus here is the computation on unstructured data for which the structure may even change occasionally and where the compute function that has to be executed may differ for each data point. Here an efficient mapping of computations to processors is not straightforward and good efficiency is generally difficult to achieve. parXXL provides a framework that facilitates programming under such constraints and draws good performances out of nowadays platforms.

The parXXL framework, see [1], includes several software layers, as shown in Fig. 1. Important for this project here are the following.

`par::cell:` a set of functionalities and a programming model to design and implement fine grained computations. This layer allows to dynamically create and connect *cells* to establish cellular networks that are *executed cyclically*.

When created, each cell is associated to four *cell behavior functions*: a function that is executed in each compute cycle, a query function that can be used to capture the state of the cell, a constructor and a destructor.

*Authors want to thank Region Lorraine.
†Email: Jens.Gustedt@loria.fr
‡Email: Stephane.Vialle@supelec.fr
§Email: Herve.Frezza-Buet@supelec.fr
¶Email: boumba_dha@metz.supelec.fr
‖Email: Nicolas.Fressengeas@univ-metz.fr

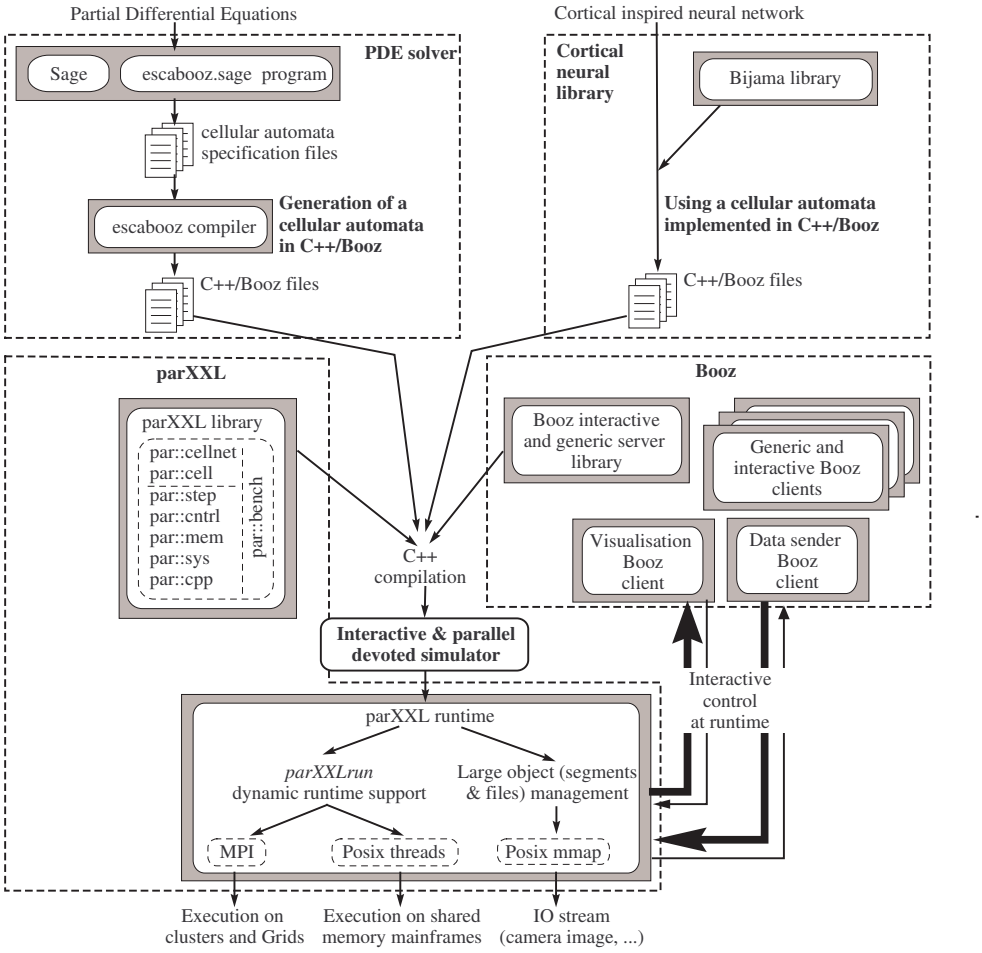


Figure 1: Global architecture of our interactive problem modeler and PDE solver on large parallel and distributed computers

Cells can communicate in a synchronous or quasi-asynchronous mode. If in synchronous mode, cell inputs are all updated at the end of each computation cycle. In quasi-asynchronous mode, cells are grouped in subsets and the communications of the different groups are routed at different times.

A network of cells can easily be controlled by a sequential program, using *missions*. Such a mission can create a set of cells, execute one compute cycle, or extract data running the query functions of the cells.

`par::mem`**:** an abstraction layer for handling large `chunks` of data. These allow for an efficient handling of large tables that are allocated on the heap or inside files and that can be resized dynamically. It allows to group the cell data and output and access them in order or through hashed indices.

`par::cntrl`**:** handles the basic communication functionalities. It abstracts from the underlying runtime, currently MPI or POSIX threads. In particular important for this project has been the `transfer` family of functions that implements a `all_to_all_v` communication. In combination with the resizability of the `mem::chunk`, `transfer` dispenses to specify communication sizes and to allocate buffers beforehand.

`par::bench`**:** is used to instrument the library and to collect various performance data. In particular it registers the number of communications and their size, wall-clock and CPU times.

3 Interactive parallel computations

One original feature of the InterCell software suite is that cellular computation is interactive. It allows visualization, writing and loading of snapshots, setting of cell values, all, while the cellular automaton is running in parallel. This feature is provided by the Booz library that includes a visualization client, see Fig. 2.

This interactivity allows to use a cluster for situated systems, like robots, where cellular computation models the inclusion of an artificial brain in some real robot

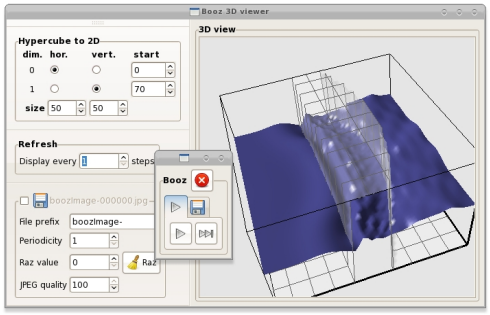


Figure 2: Example of interactive Booz client

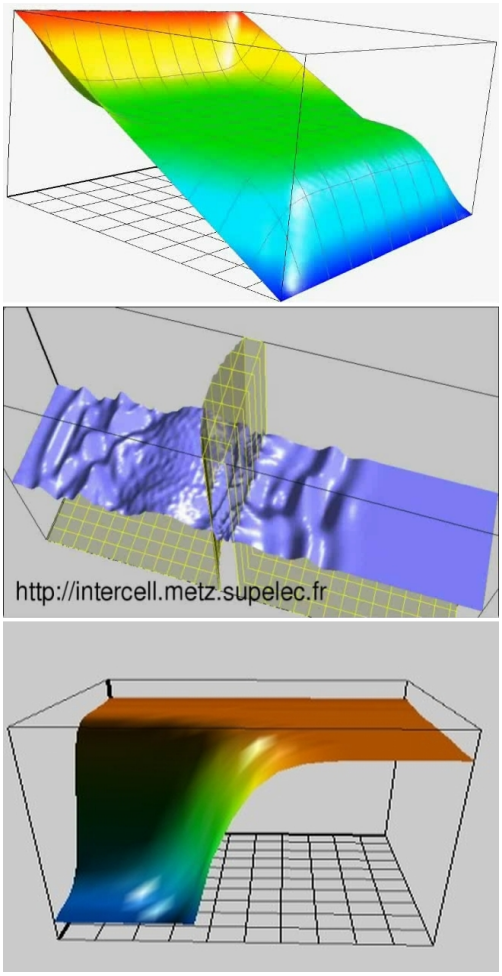


Figure 3: Three examples of InterCell simulations

perceptivo-motor loop. It also provides a real-time view of the running process, that allows to detect convergence problems of the cellular models. Such an on-line availability allows programmers of cellular automata to prototype their model at a large scale, from the very first design stage. This is of primary importance since properties of large scale discrete dynamical systems are not easily predictable from small prototypes.

4 Examples of InterCell usage

Fig. 3 shows 3 examples of InterCell simulations. On top is a classic 2D-Jacobi relaxation, where each *cell* just computes the average value of its four neighbors. It has been implemented to test the functionality of our software suite. The middle image shows a 2D grid of cells, each connected to its 8 neighbors. Each cell represents the elongation of a spring that is coupled to neighboring springs, in order to create 2D waves along the grid surface.

Fig. 3 bottom is a more complex simulation from semiconductor physics. It shows the result of a simulation of the electrostatic potential in a 2D P-N junction whose N-doped side is the square upper part while the P-doped part is the rest. This computation is done through the `sage/escapade/escabooz` part as described in Fig. 1.

The numerical method is based on a modified version of the Least Squares Finite Element Method (LSFEM) [2]. From LSFEM, we have derived a "*local only*" recursive rule. It allows for each point in a mesh to be considered as an independent automaton, which is particularly well suited for fine grained parallel computing. The initial problem is a partial differential system of equations involving a Poisson equation and the field expressions from the doping of the material. Added to this system is a set of boundary conditions: Dirichlet type where ohmic contacts are present, and Neumann type elsewhere.

The complete modeling and development process is thus as follows: the physicist (non-expert in parallelism) programs his equations in the SAGE[3] language, see Fig. 4, focusing entirely on physical and mathematical issues. Then, the `escabooz.sage` software suite, formally derives an update rule for each point of a given discretization mesh. Thereby it describes a complete cellular automaton. The SAGE program applies Newton's minimization method to a global error term. This error results from a discretized form of the initial partial differential problem. Following Newton's method, an approximate solution is fed to the automaton. Once run in asynchronous mode, the automaton eventually stabilizes around a fixed point. This is the nearest minimum of the error term and corresponds to the solution to the discretized problem.

5 Experimental performances

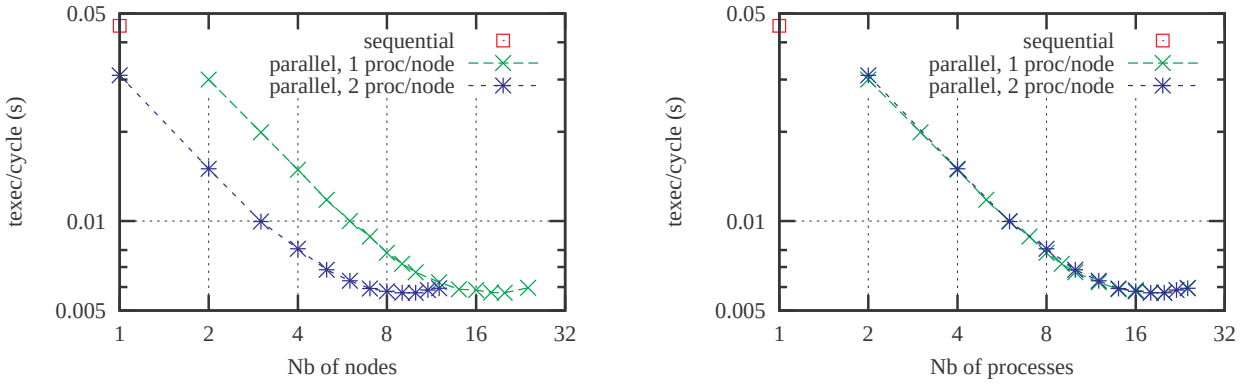
In Fig. 5 we show the results of a first performance evaluation of the second application of Section 3 on the InterCell cluster (Fig. 3 bottom). This cluster consists of 256 2.66 GHz Xeon bi-core nodes that are connected via standard Gbit Ethernet.

The example application consists of a 100×300 grid of cells that is split evenly among the parXXL processes. We experimented several different splitting strategies to find out that (for this example) the difference in performance is negligible. Thus, here we only give the values for a split along the long side (300) of the grid, $1 \times 2, 1 \times 3, \ldots$. In one series of experiments we placed one parXXL process per compute *node* and in a second series we placed two,

```
#Poisson's equation for semiconductor devices
eq1 = (lmbda*nlap(phi(x,y),r,dr) == ni*( exp(phi(x,y)) - exp(-phi(x,y))- dop(x,y)).substitute(x=0,y=0)

#Newman contidions on one border
anp=(lmbda*nd2(phi(x,y),y,dy)== ni*( exp(phi(x,y)) - exp(-phi(x,y))) - dop(x,y)).substitute(x=0,y=0)
```

Figure 4: Extract of a SAGE file to specify the electrostatic potential of a 2D P-N junction.



(a) As a function of the number of used computing nodes

(b) As a function of the number of run parXXL processes

Figure 5: Execution time per compute cycle

i.e one parXXL process per compute *core*.

Each data point in the figures represents an average over several runs of batches of 1000 compute cycles. Variances where low, so usually we only performed 3 runs to do the averaging. Printed are the run times broken down to the time for one compute cycle for the network. Fig. 5(a) shows the time against the number of nodes, Fig. 5(b) the time against the number of parXXL processes.

We see that both series show an optimal speedup in the range of 2–8 processes, and up to 16 processes the speedup is still reasonable. From thereon the addition of additional processes / nodes doesn't accelerate the computation. So for the given problem, a number of about 2000 cells is a reasonable minimal requirement per parXXL process. Fig. 5(a) also demonstrates that this setting is well suited to take advantage of the two cores in each node.

6 Conclusion and perspectives

In this paper we presented InterCell, an open, operational software suite published under the GPL, see `http://ims.metz.supelec.fr/spip.php?rubrique13`. This *development tool* is currently used by researchers in in optics, photonics, and cortical inspired neural nets. The later models are generally large and require interactive execution on large parallel systems. InterCell allows them an easy-to-use model implementation on a large, realistic scale, the automatic generation of the code and parallel interactive simulation. First performance measurements are satisfying and show a good potential to attack problems on a larger scale.

The next step in the development of InterCell will thus be to tackle applications of a larger scale: complex models are under investigations and large scale simulations are being implemented. Therefore, the Sage program that is currently used to specify the cellular automaton (which is still sequential) has to be parallelized to be able to process large problems rapidly. Also, some serial optimizations remain possible in the parXXL cell management.

References

[1] J. Gustedt, S. Vialle, and A. De Vivo, "The parXXL environment: Scalable fine grained development for large coarse grained platforms," in *PARA-06*, vol. 4699. Umeå, Sweden: Springer, 2007, pp. 1094–1104.

[2] B.-N. Jiang, *The Least-squares Finite Element Method: Theory and Applications in Computational Fluid Dynamics and Electromagnetics*. Springer, 1998.

[3] W. A. Stein *et al.*, *Sage Mathematics Software (Version 4.3)*, The Sage Development Team, 2009. [Online]. Available: http://www.sagemath.org

[4] D. Boumba Sitou, S. Ould Saad Hamady, N. Fressengeas, H. Frezza-Buet, S. Vialle, J. Gustedt, and P. Mercier, "Cellular based simulation of semiconductors thin films," in *ITFPC 09*, France Nancy, 2009.

[5] N. Fressengeas, H. Frezza-Buet, J. Gustedt, and S. Vialle, "An interactive problem modeller and PDE solver, distributed on large scale architectures," in *DFMA '07*. France Paris: IEEE, 2007.