45th International Scientific Colloquium
Ilmenau Technical University
October 04 - 06, 2000

A. Agueev / B. Däne / W. Fengler

Cellular Computing on a Linux Cluster

Abstract

Cellular computing is a special approach to massive parallelism. While the direct implementation of really large cellular computers is limited by technological constraints, good results can be obtained by simulation. This paper presents an example for such a simulation. By using a cluster of interconnected workstations the time consumed when simulating fairly large fields can be shortened. The paper mainly deals with the technology used for running the simulation in distributed manner.

1 Introduction

1.1 Cellular Computers

Parallel programming is not easy. Many years of research result in the knowledge that it's harder than it seems. Some tasks allow trivial parallelizing, some not, some don't allow any kind of it. The main problem here is the way we think and the way we write our algorithms and programs, which are actually sequential. The von Neumann's computer architecture, dominating nowadays, is also sequential. Mostly it's quite difficult to break a sequential task into different, but nevertheless sequential tasks, so that they could run in parallel and create performance benefit.

But there is a completely different approach, named cellular computing [1, 2]. Main principles of this approach are *simplicity*, *vast parallelizm* and *locality*, and using it we can forget about sequential programming. The core of a cellular computer is the cellular field, where every cell itself is a small computer. A cell calculates its state, depending on its current state and those of neighbourhood cells. Every cell sees only its local neighbours, there's nothing like hierarchy. State functions of the cells can be the same throughout the field (uniform cellular computer) or they can vary (non-uniform), and state changes can happen at the same time for all the cells (synchronous), or not (asynchronous). Although cellular computer programming is not easy again, they are capable of really parallel computation, and they can simply be ran on a workstation cluster, as we show in this article.

1.2 Workstation Clusters

Workstation Clusters (or Clusters of Workstations - COW, Networks of Workstations - NOW, Clusters of PCs - CoPs) are increasingly more popular in the last years [3, 4]. They penetrate more and more into the market of high-performance computing, once owned by mainframes. The main reason for this trend is that PCs, though attractively priced, are cluttering up „the performance mountain“ with incredible speed. The difference between supercomputers and personal computers is getting smaller. This means that one can build a supercomputer-equivalent with tens of PCs, having paid one fourth that supercomputer's price. Another reason is that high-performance computers are used only for special goals, most of the time staying idle. Workstation Clusters, on the opposite, can most of the time do their usual job as PCs, and only sometimes work together as a cluster. You have to *buy* a supercomputer, but chances are that you *already own* a cluster, since you have a network of PCs.

So what is a Workstation Cluster? A cluster consists of:

- **Hardware,** including PCs and communication (network) hardware. Every computer in a cluster has its processor(s), memory and disk(s);

- **Middleware,** which does all the cluster-management work, messaging, and other types of inter-process communication [5, 6, 7];

- and **Software,** that is written specifically to run on a cluster.

2 Going Cellular

We were considering a possibility of effective use of our workstation cluster. The thought that lead us to the idea of this experiment is following.

One of the main properties of a cellular computer is its vast parallelism, so we shouldn't have much problems distributing the load over a workstation cluster. Since the cells in a cellular computer only have local connections, why couldn't we just split the computer into a few parts? The only problem left then is border cells - the cells that lie on the cutting-border. We cannot calculate their values, since we do not have all their neighbours. But we can do the cutting in a „tricky" way (Fig. 1).

In this method the field parts slightly overlap. We take the column next to the border column as the last „right" calculated, and intentionally calculate the border column(s) false, because of partial neighbourhood's loss. The same way we do this to the other side of the cutted field. What we have to do now for the next step calculation is an exchange of „right" columns (shown in Fig. 1) between the field parts, so that they could be used as border columns. After that we have an correctly computed cellular field with the little overlapping, and we can go on again.

We have done this experiment in 2-dimensional space, but there's no limit on it. It could be done in n-dimensional space, performing n-dimensional cuts. There's always only a little overhead - two rows (or columns, or what else it can be) for every cut.

For testing purposes we decided to take a *Game of Life* program [8] - this program is very simple, it's principles are widely known, and hence this is a very good test-bed. After all, it also has many applications in many different areas [9].

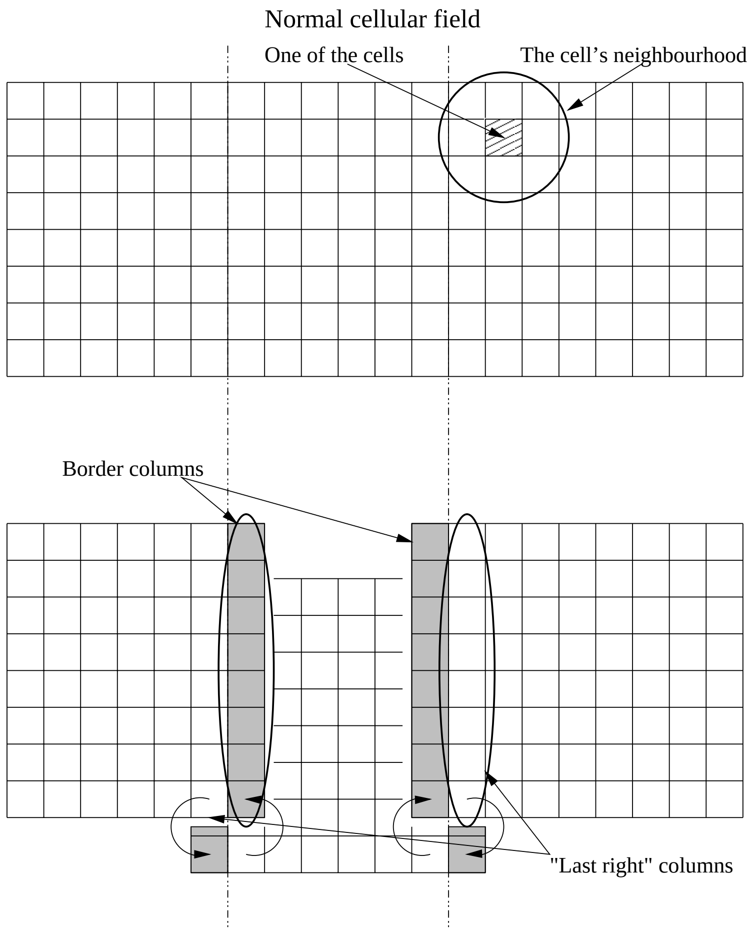


Figure 1: Cutting the field

2.1 Technical Environment

The cluster that we use in this experiment consists of:

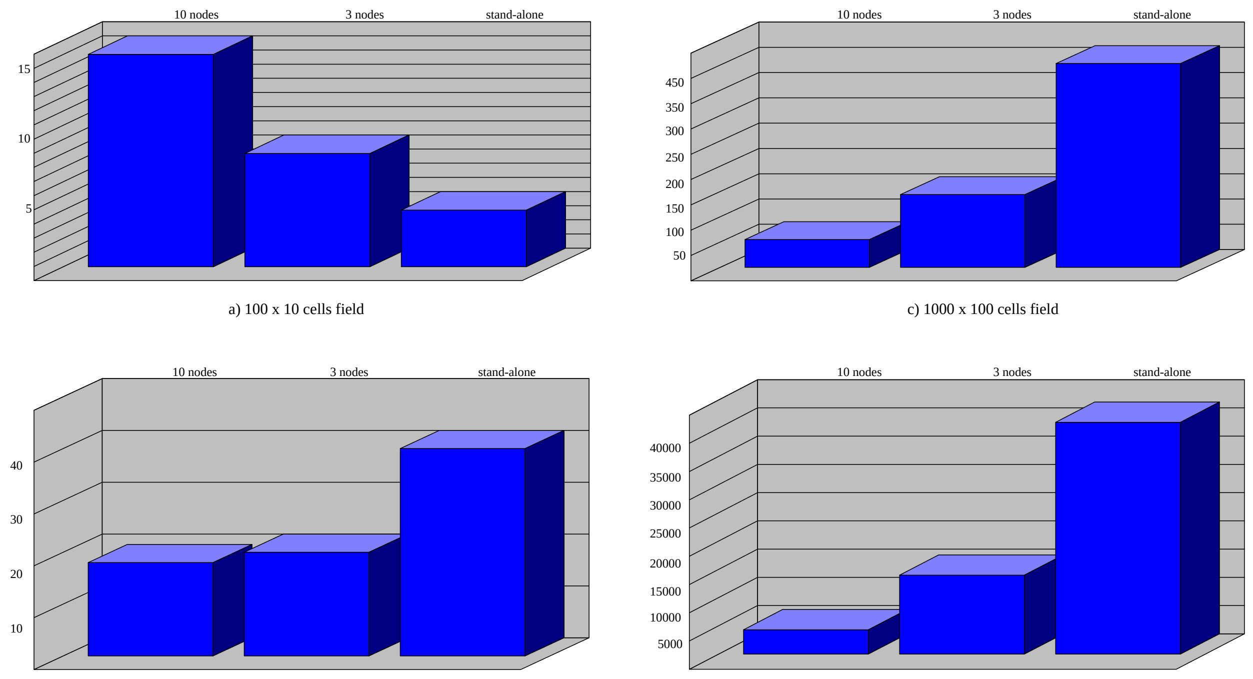
- **11 Node PCs** running Linux - Pentium III/500MHz, 512 Mb RAM, Linux kernel 2.2.10.

- **Gigabit Ethernet network** - full duplex, fully switched.

- **MPI middleware** - lam 6.2b. LAM (Local Area Microcomputer) is an MPI (Message Passing Interface) programming environment and development system from University of Notre Dame, USA [6, 10].

2.2 How we tested

We have made 8 different executables for the Game of Life model: 4 with different cellular field sizes for the cluster, with MPI code, and 4 with the same field sizes for one stand-alone computer, without any MPI instructions. For more precise results we have chosen 10 000 steps in every test performed.



a) 100 x 10 cells field

c) 1000 x 100 cells field

b) 100 x 100 cells field

d) 10000 x 1000 cells field

Figure 2: Experimental results (runtime for 10 000 steps, in seconds)

The test program for the cluster can adopt itself automatically to the cluster topology, i.e. the number of nodes in the cluster. It takes then one of the available computers as master, and the rest as slaves. The slaves perform the actual computations, while the master's work is to:

- Cut the field
- Initialize the slaves with their field parts
- Exchange the border cells for slaves in every step
- Get the results from slaves
- Glue the field parts together

We have used 4 and 11 machines in cluster configurations, but since one of the nodes (the master) actually doesn't take part in the calculations, we refer to the configurations as „3 node“ and „10 node“.

2.3. Experimental Results

You can see the test results (runtime in seconds to complete 10 000 steps, excluding the time to initialize the game field) in Table 1 and in Figure 2.

	field size				
	10^3	10^4	10^5	10^6	10^7
stand-alone	4	40	403	4 090	41 030
3+1 nodes cluster	8	20	144	1 400	13 980
10+1 nodes cluster	15	18	55	430	4 310

Table 1: Experimental results (runtime for 10 000 steps, in seconds)

In Fig. 2a we can see that for small tasks the field division, MPI initialization and interprocess communication are a big overhead, and the more nodes there are, the more overhead we have. Fig. 2b shows that for a 100x100 cells field the 3+1 nodes cluster is almost equal to the 10+1 nodes cluster, and they are only 2 times faster than the stand-alone computer.

As the field grows, the picture changes. With sizes of order 10^5 to 10^6 the cluster nodes get to almost the same performance as the stand-alone computer (Fig. 3), resulting in correspondingly 3 or 10 times better overall performance. This is actually one nearly ideal example of parallelization (speedup is proportional to number of nodes).

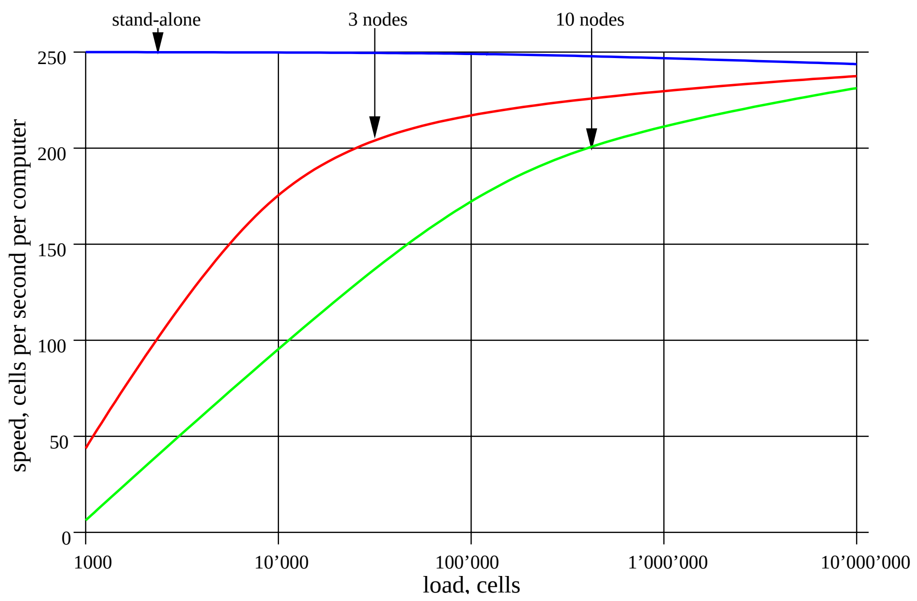


Figure 3: Computers‘ performance

3. Conclusion

Using our approach cellular computers are very easy to run on a cluster. The experiment shows that performance benefits can be high. It is trivial to adjust existing cellular computer models to run on a cluster. For cellular computer programmers it makes no difference in programming methods.

Cellular computers do their best when they run on hardware designed specifically for them. But design and testing phases are usually done with universal computers. That's where the clusters can be of great help.

The approach as it is discussed in the article and tested in the experiment can be applied to n-dimensional synchronous and uniform cellular machines. We suppose to do some additional research to extend it to non-uniform and then to asynchronous computers.

References:

[1] Sipper, Moshe: The Emergence of Cellular Computing. IEEE Computer, vol. 32, no. 7, July 1999.

[2] Sipper, Moshe: Evolution of Parallel Cellular Machines: The Parallel Programming Approach. Springer-Verlag, Heidelberg, 1997.

[3] Buyya, Rajkumar: High Performance Cluster Computing. 2 Vol., Prentice Hall, 1999.

[4] Sterling, Thomas L., and others: How to build a Beowulf: A Guide to the Implementation and Application of PC Clusters. MIT Press, 1999.

[5] Geist, Al: PVM: Parallel Virtual Machine: A Users Guide and Tutorial for Networked Parallel Computing. MIT Press, 1997.

[6] Snir, Marc: MPI: The Complete Reference. MIT Press, 1997.

[7] Pacheco, Peter: Parallel Programming with MPI. Morgan Kaufmann Publishers, 1996.

[8] Gardner, Martin: Mathematical Games. Scientific American 233(4), pp. 120-123, October 1970.

[9] Sigmund, Karl: Games of Life: Explorations in Ecology, Evolution and Behaviour. Penguin, 1995 (reprint).

[10] http://www.mpi.nd.edu/lam/

The Authors:

Dipl.-Ing. Alexei Agueev
Dr.-Ing. Bernd Däne
Prof. Dr. Ing. habil. Wolfgang Fengler
Ilmenau Technical University, P.O. Box 100565
98684 Ilmenau, Germany
Phone: +49-3677-69-2825
Fax: +49-3677-69-1614
E-mail: {ageyev;bdaene;wfengler}@theoinf.tu-ilmenau.de