Fourth Eurographics Workshop on Parallel Graphics and Visualization (2002)
D. Bartz, X. Pueyo, E. Reinhard (Editors)

Physical cloth simulation on a PC cluster

F. Zara[†] and F. Faure[‡] and J-M. Vincent[†]

[†] Laboratoire d'Informatique et de Distribution (ID-IMAG), projet APACHE (CNRS, INRIA, INPG, UJF), Grenoble, France.
[‡] Laboratoire GRAVIR, projet iMAGIS (CNRS, INRIA, INPG, UJF), Grenoble, France.

Abstract

Cloth simulation is of major interest in 3D animation, as it allows the realistic modeling of dressed humans. The goal of our work is to decrease computation time in order to obtain real time dynamics animation. This paper describes a cloth simulation and addresses the problem of parallelizing the implicit time integration and to couple a parallel execution with a standard visualization. We believe that this work could benefit to other applications based on a conjugate gradient solution and other applications of PC clusters.

Categories and Subject Descriptors (according to ACM CCS): G.4.5 [Mathematical software]: Parallel and vector implementations

1. Introduction

Cloth simulation is of major interest in 3D animation, as it allows the realistic modeling of dressed humans. Applications range from textile CAD to video games and interactive web assistants. In CAD softs, cloth simulation decreases productivity cost thanks to a direct vizualization of cloth appearance. In multimedia, cloth simulation adds a realistism to dressed humans.

Cloth is modeled as a network of particles connected by springs. Several thousands particles are typically needed for one piece of cloth. One of the biggest challenges of cloth simulation is to obtain interactivity (25 frames by second). There are two major steps in the simulation loop. First, we have to solve a differential equation over time in order to repeatedly compute the next state of the system as a function of the current state. This computation involves a large number of springs and particles. Spring stiffness requires either small time steps or sophisticated computations in order to prevent the system from diverging [3], [2], [20]. Then, collisions have to be detected and processed. This computation is basically of quadratic time complexity, which is expensive due to the high number of objects. Consequently, to obtain real time animation, a huge amount of computation time and memory are needed.

Now clusters of PCs are on the rise alternative architectures. They provide scalable computing power and are able to solve big problems of 3D simulation. So we choose to parallelize simulation algorithms in order to obtain real time, and we run our application on a PC cluster. This adds a difficulty to the cloth simulation because we have to perform the physical computations on a parallel architecture computer and to visualize the results on a separated graphics computer. Moreover our programs should be scalable to provide the best performances on several sizes of clusters and achieve portability in order to be executed on different types of processors.

The main problem concerning parallel 3D simulation is to transform a highly sequential program in a parallel code for a large number of processors. The nature of time-driven simulation implies a step by step computation. Parallelism could be exploited only inside a step of simulation, saying roughly between two global synchronization points.

Because of clusters development, scalability of parallel programs is of fundamental importance. In particular, for irregular applications (numerical treatment of large sparse matrices), it is not clear that classical parallel methods such as conjugate gradient are scalable. Parallel libraries, such as ScaLapack (http://www.netlib.org/scalapack/), seems to be inefficient for irregular applications, in particular when the structures of the sparse matrices are modified during the execution. This is typically the case for cloth simulation, because contacts modify the structure of the network of particles. To avoid this difficulty, we use a dynamic scheduler with load balancing policies. Depending on the evolution of the simulation, computation tasks should be automatically spread on the processors to guaranty a minimal throughput.

Our main contribution is the following. We present a decomposition of the physical simulation problem based on a structuration of the object space. This leads to a parallelization of the simulation step: an implicit time integration based on a conjugate gradient algorithm. Moreover, we present a practical solution to interactively visualize a simulation performed on the cluster using a separated graphics computer. We believe that this work could benefit to other applications of PC clusters. In this paper, we do not address the problem of collision detection [14], [18], [20], but the global architecture of the program allows modifications to implement collisions detection.

Our program is designed for clusters up to one hundred PCs. First experiments on our one hundred basic PC cluster run about 100,000 particles. We predict that this code could scale to a million of particles on the same architecture.

The paper is organized as follows. Section 2 presents the previous work made in computer animation on deformable objects simulation and in parallel computation. Section 3 details our PC cluster architecture and parallel programming interface. Section 4 describes the physical model and its implementation as an ODE solution. Section 5 shows our associated parallel data structures and algorithmic schemes. Section 6 presents how we display particles positions computed on the cluster of PC. Then we show results in section 7. We finally discuss future work in section 8.

2. Previous work

In computer animation, particles systems have proven to be an appropriate model for fast physically based simulation of deformable objects [3], [2], [5], [7]. These models require an ordinary differential equation to be solved.

Meier and Eigenmann [13] have analyzed the computational structure of several different congugate gradient schemes for solving elliptic partial differential equations, using a parallel implementation on the Cedar hierarchical memory multiprocessor. Demmel, Heath and van der Vorst [8] have explained iterative algorithms for solving linear systems of equations considering dense, band and sparse matrices on parallel architectures.

With the work of Baraff and Witkin [3], implicit integration methods designed to solve differential equations, have proven to allow the use of large time steps without loss of stability. Desbrun *et al.* [9] have proposed a stable, real-time algorithm for animating cloth-like materials using a hybrid explicit/implicit algorithm.

Last year, Hauth and Etzmuss [10] have presented theorical analysis to exploit special properties of the mechanics of deformable objects. Romero and Zapata [15] have detailed a solution for cloth and other non-rigid solid simulations on parallel computers. They have developed an application, which combines data parallelism with task parallelism on 8 processors.

3. Experimental context

3.1. Hardware

The dynamic simulation of our 3D object consists in computing along time, positions and velocities of sample points and to display the associated surface. If the object has N vertices, we have to compute $3N+3N$ values, which are positions and velocities coordinates at each timestep. We then have to visualize $3N$ coordinates. When the object is complex, that is when N is nearby 100 000 (for more realistic simulation), the goal of real time becomes illusive because of the amount of computation to perform and moreover we have memory problems due to the amount of data. The memory used by our model is nearby $4 \times 3N \times 25$ bytes, this corresponds to a flow of 30 Mb/s for a frame rate of 25 Hz.

In order to decrease computation time of each frame of simulation, we chose to make parallel algorithms, using the parallel programming interface Athapascan developped in the ID-IMAG laboratory for the APACHE project. The application runs on a cluster of 216 HP e-vectra (pentium III, 733 MHz, 216 Mo, 15 Go) for the i-cluster (INRIA, HP) project and the LIPS (INRIA, BULL) project (see figure 1).

Computation is performed in parallel on the cluster, and the results are brought up on screen of a standard computer, using the library GLUT (OpenGL Utility Toolkit). Consequently we have to control the flow between the cluster and the computer used for visualization, in order to obtain real time (25 Hz).

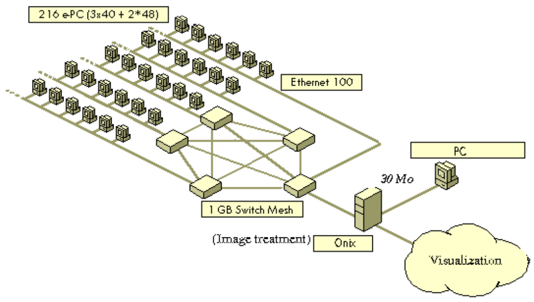


Figure 1: *Architecture: reality center and cluster together.*

3.2. Parallel environment programming Athapascan

Parallel hardware, like a cluster, provides a computation more power than using a mono or multi-processors computer and cheaper than using a Cray. In fact, in the eighties, parallel hardware corresponded to construction of vectorial processors (Cray), adapted to a kind of applications and more specialy to regular problems. However these system are very expensive. Therefore in the nineties, parallel computer have been elaborated using powerful standard processors and a fast interconnection network.

In order to use this kind of parallel architecture easily, we have used the parallel programmation environment Athapascan. Athapascan is an environment designed for programming parallel computers. It implements a parallel programming model based on a shared memory. A parallel code can be generated that allows parallel and distributed executions on a wide class of parallel architectures (multi-processors computers, cluster of PC). Thus we only have to write once our program, and then it can be executed on all kind of parallel architecture. This environment consists of two modules:

- Athapascan-0 [6] is a multi-threaded, portable, parallel programming runtime system for distributed architectures, which can be available for all platforms where a POSIX threads kernel and a MPI communication library have been installed. It is designed to solve efficently large irregular problems.
- Athapascan-1 [17] is the application programming interface of Athapascan. It is structured in three layers. The first one is the application programming interface which is seen by user. It enables to generate parallel or sequential code at compile time, depending on the scheduling annotations. The second layer is the parallel library which implements the programming interface to achieve parallel and distributed executions. The third layer is the scheduling library which provides various scheduling policies and facilities to develop new ones. The parallel and the scheduling libraries are implemented upon Athapascan-0. Athapascan-1 is a **high level** interface in the sense that no reference is made to the execution support: the synchronization, communication and scheduling operations are entirely supported by Athapascan-1. Moreover it is an **explicit parallelism language**: the programmer indicates himself the parallelism of its algorithm performing asynchronous procedure calls via Athapsacan-1 keywords. The granularity of the computation and the data are also specified by user through classical procedure and type definition.

An Athapascan-1 program is a set of tasks dynamically created which share some objects. In order to use this parallel environment, we only have to split our simulation in a set of computation tasks and to find which objets have to be shared by processors of the cluster. The Athapascan-1 interface provides a **data-flow language**: the execution is data driven and determinated by the availability of the shared data, according to the access mode. In a parallel context, this implies that synchronizations between concurrent computations are made in order to ensure that the access to the shared data are consistent with the algorithm. That is to say that date of execution of a task is automatically determined by the system in order to ensure that the versions of the shared objects it owns are sementically coherent.

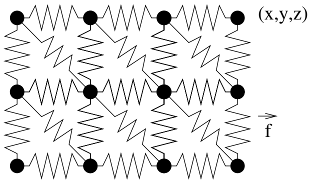
4. Physical cloth simulation

In this part, we shortly describe the physical simulation of cloth [2], [11], [5], [7]. It consists in computing the motion of a large number of particles in interaction. The principal stages of the animation are:

- Ordinary differential equation (ODE) integration,
- collision processing.

Here we focus on the first stage. In order to apply large time step, we use an implicit integration method [3].

4.1. Physical model



Cloth is modeled as a triangular mesh of particles connected by springs (figure 2). We have N particles in a 3D-space. A force f is applied to each particle.

Figure 2: *Triangular particles mesh connected by springs*

The acceleration of the i^{th} particle of our simulation is given by the fundamental dynamics rule, $x_i'' = f_i/m_i$, where f_i is the force applied to this particle and m_i its mass. The forces exerted are the gravity, the springs forces and the airdamping. If we define the diagonal matrix M by $M = diag(m_1, m_1, m_1, ..., m_N, m_N, m_N)$, where $m_1, ..., m_N$ are the masses of the particles, we have:

$$x'' = M^{-1} f(x, x').$$

Euler's method can be used to calculate system state evolution during time:

$$\begin{aligned} x(t_0+h) &= x_0 + hx_0' \\ x'(t_0+h) &= x_0' + hx_0'', \end{aligned}$$

where x_0, x_0', x_0'' are position, velocity, and acceleration of particles system at time t_0, respectively and h the stepsize parameter.

This method can be unstable when h is big or when the stiffness is large. Thus we use another method called implicit Euler method. To calculate the state of system and its velocity by an implicit method, we define velocity v as $v = x'$. Hence we have:

$$\frac{d}{dt}\begin{pmatrix} x \\ x' \end{pmatrix} = \frac{d}{dt}\begin{pmatrix} x \\ v \end{pmatrix} = \begin{pmatrix} v \\ M^{-1} f(x, v)) \end{pmatrix}.$$

To simplify notation, we define $\Delta x = x(t_0+h) - x_0$, $\Delta v = v(t_0+h) - v_0$ and force $f_0 = f(x_0, v_0)$.

David Baraff and Andrew Witkin show [3] by applying a Taylor series, that this system becomes:

$$\left(M - h\frac{\partial f}{\partial v} - h\frac{\partial f}{\partial x}\right)\Delta v \;=\; h\left(f_0 + h\frac{\partial f}{\partial x} v_0\right), \qquad (1)$$

which we have to solve to obtain Δv. We then easily compute $\Delta x = h(v_0 + \Delta v)$.

To sum up, when we use Euler's implicit method we have to evalue f_0, $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial v}$, to construct a linear sparse system, to solve it in order to compute Δv, and then to update x and v.

To solve the linear system, we apply the conjugate gradient method which easily uses the sparsity of the matrix, since it only adresses the matrix through its product with a vector.

Implementation 1 shows the computations made at each step of the simulation.

Implementation 1 Simulation step

```
ComputeForces();
ComputeAccelerations();
CreateEquationSystem();
SolveSystem();
UpdateVelocitiesAndPositions();
CollisionsProcessing();
```

5. Parallel cloth simulation

In this part, we first present two strategies to parallelize a simulation of particles (a spatial split of space simulation and a space dividing). Second, we describe the data structure. Third, we explain the conjugate gradient method to solve the sparse linear system.

5.1. Spatial split of space simulation *vs* space dividing

The currently traditionnal parallel programming approaches, as communicating processes and data parallelism, can easily exploit hardware parallelism when the computation can be split into smaller equally weighted computation processes, with good data locality. This type of parallel computation is called **regular**. Simple mapping and scheduling algorithms can be used to exploit satisfactorily the hardware parallelism. The same does not happen if the computations do not have equivalent sizes (or they are not previously known) or do not have good data locality properties. Parallel computations like that are called **irregular**.

Our cloth simulation is an irregular problem. Indeed using an implicit integration method, we have to solve a sparse linear system to compute velocities. Moreover in processing collisions, weighted computation is not the same according to the cloth zone considered. For this reason, we use the parallel programming environment Athapascan which permits to parallelize irregular problem. Indeed we can create parallel task which have not the same computation size.

There are two main strategies to parallelize a simulation of particles. The first one is to split space in boxes which contain parts of the object which we want to simulate [4] (see figure 3.a). These boxes are distributed to the processors. But the particles are not attached to a given processor.

The second strategy is to split the particles in the object space (see figure 3.b). The particles system is split in particles blocks, which can be dynamically assigned to processors [15] by Athapascan.

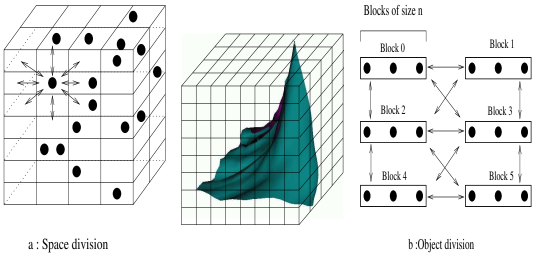


Figure 3: *Spatial split of space simulation vs space dividing.*

We chose to use the second strategy in order to parallelize our cloth simulation. This strategy is more adaptated to irregular problems. The number of particles is the same in each block, so we can assume that we have the same computation load for each block.

We don't lose any information in splitting the system in blocks. Indeed we have a data structure which binds blocks by knowing neighbouring particles of each system particle (see figure 3.b). This structure allows us to compute forces applied to particles which requires position, velocity and acceleration of all its neighbouring.

5.2. Parallelization by data blocks

Using parallel simulation, we can compute the position, velocity and acceleration of many particles at the same time. At time t, positions, velocities and accelerations of all particles are stored in 3 different arrays. By splitting each of these arrays into several blocks, each block being an Athapascan-1 shared object, we perform computations on several particles at the same time.

Let us first focus on position computation. Implementation 2 describes an algorithmic scheme for the Euler's method, used for determining new particles positions.

Implementation 2 General scheme in Athapascan for independent operations

```
// Method declaration
struct Euler {
  void operator()(Shared_r_w<Vect3D> BlockPos,
                  Shared_r<Vect3D> BlockVeloc) {
        // Computation of x_i(t+h) = x_i(t) + h v_i(t) }
};
// Parallel computation of the blocks
for(int i=0; i<Number_of_blocks; i++)
   Fork<Euler>()(Position[i], Velocity[i]);
```

In this example, “Position” refers to the array of particles position. Practically, positions are stored as blocks in this array. Thus Position[i] is a block containing the position of n particles. These blocks are Athapascan-1 shared objects, on which the “Euler” function is applied.

The “Fork” keyword creates an Athapascan-1 task. The loop “for” of implementation 2 creates “Number_of_blocks” parallel tasks on processors. Each processor deals with all particles contained in one block “Position[i]”.

Let us now focus on the determination of velocities and accelerations. As shown in section 4, they depend on several forces. The force applied to a particle is the sum of the forces exerted by neighbouring particles and the external forces (like gravity, wind, ...).

Consequently, we should store the mesh as a data structure providing the connection data. Each particle points (using particles identifier) to its neighbouring particles and to the associated spring properties (see figure 4).

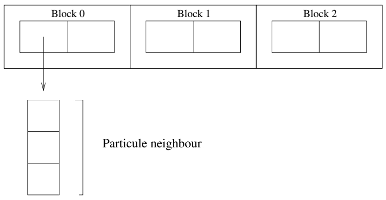


Figure 4: *Mesh matrix structure.*

Similarly with positions, this structure is split into shared blocks. We store the particle forces in the split array named “Force”.

To determine the forces applied to each particle we first initialize the forces at zero and then accumulate the contribution of each spring. Force blocks are processed in parallel (see figure 5). The force applied to a particle in a given block results from interactions within the block, and possibly from interactions with particles in other blocks. We process them separately. Local interactions are processed locally within a computation mode. Block-to-block interactions are processed by parallel tasks, each one dealing with a pair of blocks.

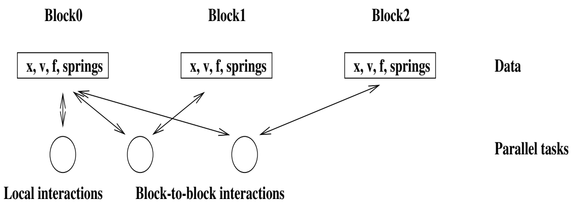


Figure 5: *Computation scheme of forces applied to block 0.*

To compute interactions between the blocks i and j, we only have to put Mat[i] in parameter, giving carrying all information about connections. We set the access rights of the force vector to “cumulative write” (suffix cw) (see implementation 3). It allows us to update the force applied to each particle of the two blocks, by adding the tasks results as they become available, without prior scheduling. This can be done because addition is an associative and commutative operation. The interactions between two blocks are computed only once and then there is no computation redundancy.

Implementation 3 Parallel computation of force

```
// Method declaration
struct Task_Force {
    void operator()(Shared_r<Vect3D> BlockPos1,
                    Shared_r<Vect3D> BlockPos2,
                    Shared_r<Vect3D> BlockVeloc1,
                    Shared_r<Vect3D> BlockVeloc2,
                    Shared_cw<Vect3D> BlockF1,
                    Shared_cw<Vect3D> BlockF2,
                    Shared_r<VectorMatrP> BlockMat) {
            // Interaction computation between block1 and block2}
};
// Iteration in main
for (i,j) ∈ B_i × B_j do // Considere only cases where i linked to j
    Fork<Task_Force>()(Pos[i], Pos[j], Veloc[i], Veloc[j],
                       Force[i], Force[j], Mat[i]);
```

5.3. Parallel conjugate gradient

We have to solve a linear system (equation 1) to update particles velocities. We define **A** as

$$A = \left(M - h\frac{\partial f}{\partial v} - h\frac{\partial f}{\partial x}\right),$$

and vector **b** as

$$b = h\left(f_0 + h\frac{\partial f}{\partial x}v_0\right).$$

Since the matrix A is symetric and positive, we use the conjugate gradient algorithm [16], [19] to solve the linear system $A\Delta v = b$. The algorithm terminates when the error parameter is smaller than the desired accuracy (see implementation 4).

The conjugate gradient algorithm performs simple algebraic operations: vector sums, dot products and matrix/vector products. Figure 6 illustrates the product of the sparse matrix $\left[\frac{\partial \vec{f}}{\partial x}\right]$ by a vector. This matrix is symmetric and its structure is given by the graph. Matrix entries are blocks of dimension 3 × 3. Diagonal elements correspond to graph nodes (particles), and non-zero entries correspond to edges (springs). We thus store the matrix entries similarly with particle values, for the diagonal entries, or with spring values, for the other entries. Matrix and vectors are thus split into blocks, allowing parallelization.

Implementation 4 Conjugate gradient algorithm for solving $A\Delta v = b$

$\beta \leftarrow 0$; // Error factor initialisation
$\Delta v \leftarrow 0$; // Solution initialisation
$R \leftarrow b - A\Delta v$; // Residual vector initialisation

$\alpha \leftarrow R^T R$; // Step initialisation

$If(\beta \neq 0)$
$T \leftarrow R + (\frac{\alpha}{\beta})T$; // New direction vector
$Else$
$T \leftarrow R$; // Director vector initialisation

$\beta \leftarrow T^T AT$; // Error factor update
$R \leftarrow R - (\frac{\alpha}{\beta})AT$; // Residual vector update
$\Delta v \leftarrow \Delta v + (\frac{\alpha}{\beta})T$; // Solution update
$\beta \leftarrow \alpha$; // Error factor

Until $(\beta < \varepsilon)$ // Iteration until desired precision

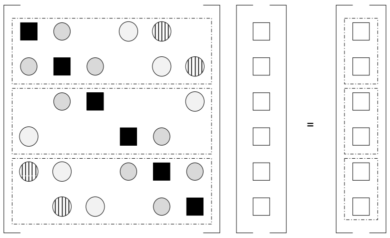


Figure 6: $\left[\frac{\partial \vec{f}}{\partial x}\right] \times$ *direction vector product.*

Figure 7 illustrates the parallel algorithm scheme for this product. This scheme is similar with the one used to compute forces described in figure 5 and has therefore, the same parallel implementation. Only the detail of the computations changes: linear for matrix $\times$ vector product, non-linear (spring's elongation) for force computation. The product between matrix blocks and vector blocks are made in parallel, in the same way as the forces exerted on points by springs were computed (see implementation 3). Access rights are set again to "cumulative write" to allow the accumulation of these forces. Therefore a unique data structure is needed for the simulation. The terms of the equation system, as well as the internal vectors of the algorithm, are stored in the blocks with the positions, velocities and forces.

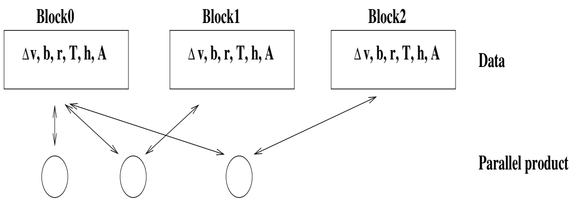


Figure 7: *Parallel algorithm scheme for product of* $\left[\frac{\partial \vec{f}}{\partial x}\right] \times$ *direction vector.*

6. Visualization

One of the difficulties of our cloth simulation, is that it combines a parallel execution with a visualization on a single computer. The visualization is made using the Glut library (OpenGL Utility Toolkit).

The whole difficulty rests in the fact that at each simulation step, data are distributed over all computer and must be gathered on the computer dedicated to visualization.

We make two programs: an Athapascan-1 program (implementation 6) which computes particles states in parallel, and a Glut program (implementation 5) which visualizes them (see figure 9).

Implementation 5 States particles visualization

```
Vect3D Position; // Global variable of particles positions
Socket sock; // Socket descriptor

void update() {
   // Positions array update with received buffers
   Gather(Position, sock);
   glutPostRedisplay(); // Request for redraw
}

int main () {
   // Get the socket descriptor
   Init(sock);
   glutDisplayFunc(display); // Visualization
   glutIdleFunc(update); // Particles positions update
   glutMainLoop(); // Start main loop
}
```

Implementation 6 States particles computation

```
Vect3D_Shared Position; // Positions particles array
Socket sock; // Socket descriptor

int main (){
   // Get pending socket connection
   Init(sock);

   while(1) {
     SimulationStep();
     Send(Position, sock);
   }
}
```

These two programs are connected together using sockets. The Athapascan-1 program opens a socket and waits for the visualization program to connect. Then at each simulation step, the Athapascan-1 program sends particles positions to the visualization program. A notification receipt is sent as soon as the whole data is received. Thus only positions transit through the network.

7. Results

In order to evaluate the efficiency of the cloth simulation, we make several simulations with, each time, a different number of particles and processus and a new size for the blocks. We fix the number of iterations used to compute the conjugate gradient to 5.

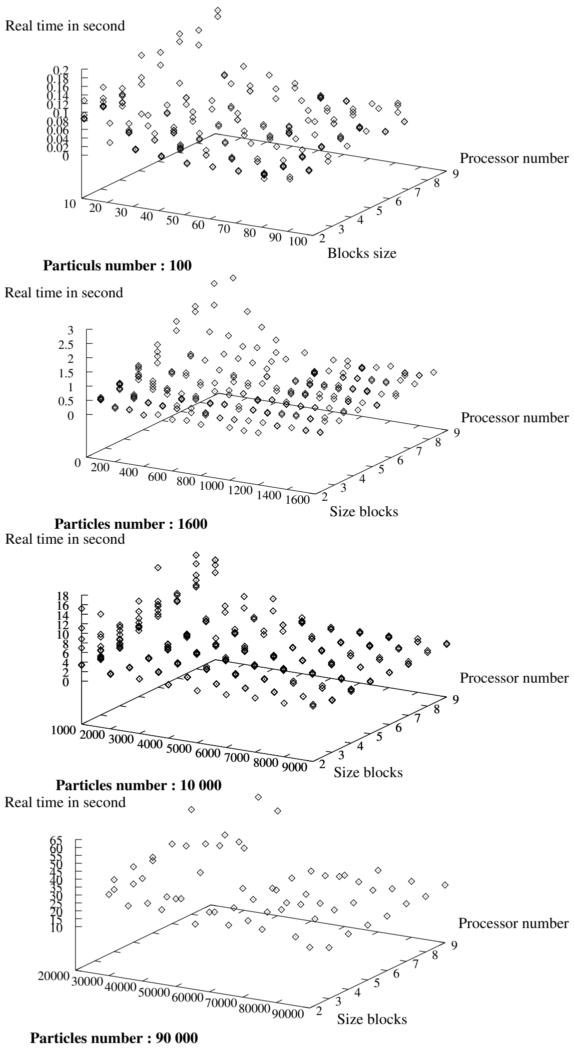


Figure 8: *Real time by frame.*

We can see that the execution time depends both on the number of processors and on the size of the blocks. These experimentations permit to find the suitable size of blocks, that is to say the granularity of parallelism, so as to optimize the execution time.

The Athaspacan version we use, as we are writting this paper, can only run our simulation on a multiprocessors machine, as can be noticed on 7. This is of course a first step and we will run the simulation on a cluster of 216 HP e-vectra PC (pentium III, 733 MHz, 216 Mo, 15 Go) as soon as Athapascan permits (the use of a newer version of Athapascan will not require any modification of our program). However, these results show that a great amount of time is spent in communication for too small blocks size. To us, a better distribution of particles in the blocks coupled with a suitable scheduler would consequently improve the results.

8. Future work

Thus we have presented a parallel cloth simulation using a decomposition based on a structuration of the object space. Our implementation uses the parallel environment Athapascan, which permits to execute our simulation on every kind of parallel architecture. This work suggests a number of areas for future research, such as:

Collisions detection and treatment [20] [12]**:** Collision detection between particles has not been implemented yet, but will soon be added to the application.

Using of the parallel vizualisation interface Net Juggler [1]**:** We plan to use the parallel vizualisation interface Net Juggler [1] which permits a parallel displaying of our animation. This interface exploits the power of many computers graphics cards, each bringing up on screen a part of the image. This allows the parallel visualization of a high number of particles, without using an expensive graphic computer as an SGI Onix (Silicon Graphics).

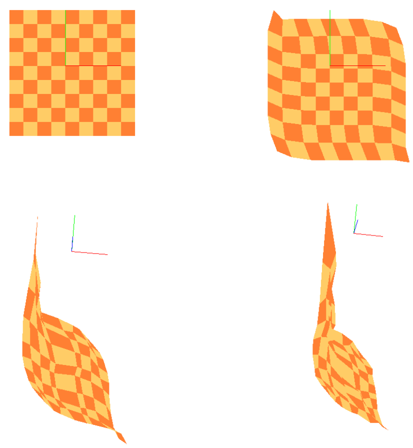


Figure 9: *100 particles large cloth maintained at a corner.*

9. Acknowledgements

This work was partly financed by the contract of the "Thématique Prioritaire $n^o 4$ *"Sciences et technologies de* *l'information, outils et applications"* de la région Rhône-Alpes."

References

1. J. Allard, L. Lecointre, V. Gouranton, E. Melin, and B. Raffin. Net juggler guide. Technical Report RR-LIFO-2001-02, LIFO, Orléans, France, June 2001.

2. W. Stasser B. Eberhardt, A. Weber. A fast, flexible, particle-system model for cloth. *IEE Computer Graphics and Applications*, 16:52–59, 1996.

3. D. Baraff and A. Witkin. Large steps in cloth simulation. In *Computer Graphics Proceedings, Annual Conference Series*, pages 43–54. SIGGRAPH, 1998.

4. P.-E. Bernard. *Parallélisation et multiprogrammation pour une application irrégulière de dynamique moléculaire opérationnelle*. PhD thesis, Institut National Polytechnique de Grenoble, France, October 1997.

5. D. Breen, D. House, and P. Getto. A particle-based model for simulating the draping behavior of woven cloth. *Textile Research Journal, Vol. 64*, 11:663–685, nov 1994.

6. J. Briat and M. Pasin I. Ginzburg. *Athapascan-0 User Manual*. Projet APACHE, Grenoble.

7. M. Kass D. Terzopoulos, A. Witkin. *Computer graphics techniques for modeling cloth*. 1988.

8. J. Demmel, M. Heath, and H. van der Vorst. Parallel numerical linear algebra. In *Acta Numerica 1993*, pages 111–198. Cambridge University Press, Cambridge, UK, 1993.

9. M. Desbrun, M. Meyer, and A. H. Barr. Interactive animation of cloth-like objects for virtual reality.

10. M. Hauth and O. Etzmuss. A high performance solver for the animation of deformable object using advanced numerical methods. In *European Association for Computer Graphics (EUROGRAPHICS'2001)*, Manchester, UK, September 2001. ACM.

11. D. Crochemore J. Louchet, X. Provot. Evolutionary identification of cloth animation models. In Dimitri Terzopoulos and Daniel Thalmann, editors, *Computer Animation and Simulation'95*, pages 44–54. Springer-Verlag, 1995.

12. A. Mir M. Mascaro and F. Perales. Elastic deformations using finite element methods in computer graphic publications. In H. H. Nagel and F. J. Perales, editors, *LNCS*, volume 1899, pages 38–47, 2000.

13. U. Meier and R. Eigenmann. Parallelization and performance of conjugate gradient algorithms on the cedar hierarchical-memory multiprocessor. In *Proceedings of the 3rd ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, volume 26, pages 178–188, Williamsburg, VA, April 1991.

14. T. Moller. A fast triangle-triangle intersection test. *JGTOOLS: Journal of Graphics Tools*, 2, 1997.

15. S. Romero, L. F. Romero, and E. L. Zapata. Fast cloth simulation with parallel computers. In *Euro-Par 2000 European Conference on Parallel Processing*, pages 491–499, Munich, August 2000.

16. Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, 1996.

17. Athapascan-1 team. *Athapascan-1*. Projet APACHE, Grenoble.

18. N. Magnenat Thalmann. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum*, 13(3):155–166, 1994.

19. A. Gupta V. Kumar, A. Grama and G. Karypis. *Introduction to parallel computing, design and analysis of algorithms*. Benjamin/Cummings, 1994.

20. P. Volino and N. Magnenat Thalmann. Implementing fast cloth simulation with collision response. In *CGI'00 Computer Graphics International*, Geneva, June 2000.