Fast Cloth Simulation with Parallel Computers

Sergio Romero, Luis F. Romero, and Emilio L. Zapata

Universidad de Málaga, Dept. de Arquitectura de Computadores, P.O. Box 4114, E-29080, Spain,
{sromero,felipe,ezapata}@ac.uma.es

Abstract. The computational requirements of cloth and other non-rigid solid simulations are high and often the running time is far from real time. In this paper, an efficient solution of the problem on parallel computer is presented. An application, which combines data parallelism with task parallelism has been developed, achieving a good load balancing and minimizing the communication cost. The execution time obtained for a typical problem size, its super-linear speed-up, and the iso-scalability shown by the model, will allow to reach real-time simulations in sceneries of growing complexity, using the most powerful multiprocessors.

1 Introduction

Cloth and flexible material simulation is an essential topic in computer animation of realistic virtual humans and dynamic sceneries. New emerging technologies, as interactive digital television and multimedia products, make necessary the development of powerful tools able to perform real time simulations. There are several approaches to simulate flexible materials. These methods can be classified in physically-based, geometrical and hybrid models (a combination of both). The former provide reliable representations of the behavior of the materials, while the others require a high degree of user intervention making them unusable for interacting applications. In this work, a physically-based method has been chosen. In a physical approach, clothes and other non-rigid objects are usually represented by interacting discrete components (finite elements, springs-masses, patches) each one numerically modeled by an ordinary differential equation: $\ddot{x} = M^{-1} f(x, \dot{x})$, where x is the vector of position of the masses M. The derivatives of x are the velocities $\dot{x}$ and the accelerations $\ddot{x}$.

The model presented considers both spring-mass and triangle mesh formulations. The former is usually applied with solids, while the later works better for clothes. Equations in most formulations contain non-linear components, that are typically linearized using a Newton method, generating a linear system of algebraic equations where positions and velocities are the unknowns. So, these different formulations can be merged, giving an unified system where 2D and 3D models are simultaneously solved, an essential topic when interactions among bodies occur. The use of explicit integration methods, such as forward Euler and Runge-Kutta, results in easily programmable code and accurate simulations [6]. They have been broadly used during the last decade, but recent works [1]

A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 491–499, 2000.

demonstrate that implicit methods overcome the performance of explicit ones, assuming a non visually perceptible lost of precision. In the composition of virtual sceneries, appearance, rather than accuracy, is required. So, in this work, an implicit technique, backward Euler method, has been used to solve equation (1) where $v = \dot{x}$.

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

The detection of collision between simulated objects is critical and the computational cost can be extremely high [7]. To avoid an exhaustive collision detection test that requires time $O(n^2)$, an spatial-temporal coherence strategy has been implemented. In the case of collision, additional forces are introduced to maintain the system in a legal state.

In this work, the key factors, like data distribution, load balancing and communication overhead, have been considered in order to obtain a high speed-up for the related problem. The application code has been implemented on a cache-coherent shared memory platform (SGI Origin 2000), and may be easily ported to other multiprocessors and multicomputers.

In section 2, a description of the used models and the implementation technique for the implicit integrator are presented. Also, the resolution of the resulting system of algebraic equations, by means of the Conjugate Gradient method, is analyzed. In section 3, the parallel algorithm and the data distribution technique for a scenery is shown. Finally, in section 4, some results and conclusions are presented.

2 Implementation

To create animations, a time-stepping algorithm has been implemented. Every step is mainly performed in three phases: computation of forces, determination of interactions and resolution of the system. The iterative algorithm is shown in figure 1.

The `Update State` procedure computes the new state of the system, which is made up of the position and velocity of each element, calculated from the previous one. `Compute Forces`, `Collision Detection` and `Solve System` stages are described below.

2.1 Forces

Forces and constraints are evaluated on every discrete element in order to compute the equation coefficients for the second Newton's law. For a practical and general application, both spring-mass discretization —of 2D–3D objects, as shown in figure 3— and triangular patches —for the special case of 2D objects like garments, as shown in figure 2— have been included. The particular forces considered are: Visco-Spring forces mapped on the grid for the former model; and Stretch, Shear and Bend forces for the later. In both cases, gravity and air

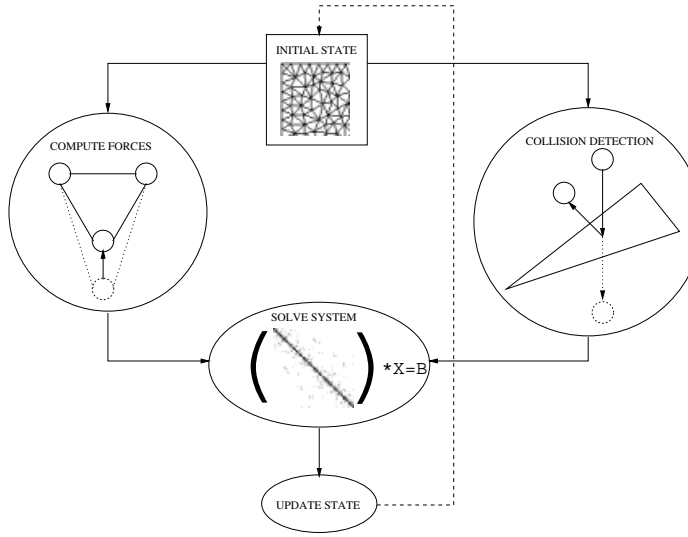

Fig. 1. Simulation Diagram.

drag forces have been also included. The backward Euler method approximates the second Newton's law by the equation (2),

$$\Delta v = \Delta t \cdot M^{-1} \cdot f(x_i + \Delta x, v_i + \Delta v) \tag{2}$$

being $\Delta x = \Delta t(v_i + \Delta v)$. This is a non-linear system of equations which has been time-linearized by one step of the Newton method as follows:

$$f_{i+1} = f(x_{i+1}, v_{i+1}) = f_i + \left|\frac{\partial f}{\partial x}\right|_i \Delta x + \left|\frac{\partial f}{\partial v}\right|_i \Delta v \tag{3}$$

An energy function E_α for every discrete element α is analytically described; the forces acting on particle i are derived from $f_i = -\partial E_\alpha / \partial x_i$ and the arising coefficients from the analytical partial derivations in equation (3) have been coded for its numerical evaluation. All above gives a large system of algebraic linear equations with a sparse matrix A of coefficients.

2.2 Collisions

In the case of cloth simulation, the self-collision detection and the human-cloth collisions may be critical and the computational cost can be extremely high [7]. In order to detect possible interactions and forbidden situations, like colliding surfaces or body penetrations, a hierarchical approach based on the use of bounding-boxes combined with a spatial-temporal coherence strategy have been implemented. In these cases, additional forces are introduced in the system matrix to keep the simulation in a legal state. These forces are included in the system as described above.

In the global scenery, every pair of objects have to be checked to detect if collisions occur. In a hierarchical approach each object is associated with a binary tree of Axis Aligned Bounding Boxes ($AABB$s), in which root node represents a box that enclose the whole object and the leaves enclose only single triangles of the surface. To detect when two objects collide and to determine which pairs of triangles are too close, a simple recursive algorithm can be used. In order to exploit the temporal and spatial coherence a more elaborated algorithm is necessary. In general, when searching collision between two subtrees, two possibilities may occur: the corresponding boxes overlap or not. In the first case the algorithm must go on down the trees until the compared nodes are both leaves, in this case, if the boxes overlap a pair of triangles can be very close or touching them selves, and then it is added to the *possible collision list*. In the other case, the non-overlapped boxes have a minimum distance, and this distance is added to another *non-collision list*. Pair of triangles sharing a vertex will never be an element in any list.

In the following steps, it is not necessary to check every pair of objects and all the subsequent hierarchy. The *possible collision list* is reviewed, if the $AABB$s overlap, the element remains in the list, otherwise, the element is deleted and the calculated distance between the $AABB$s is added to the *non-collision list*. Every element in the *non-collision list* is recomputed using a heuristic, in order to predict when the given $AABB$s do overlap, in a robust way.

Once these lists are filled, every pair of triangles in the *possible collision list* is evaluated to determine if repulsion forces must be added in the matrix A.

The main problem is that the *non-collision list* can grow up to $O(n^2)$ because old collisions are always checked on the following steps. In practice, garments only collide with a small, fixed part of the body, so the lists remain in a manageable size. In any case, if the list grows above a given limit, the collision detection system can be restarted.

2.3 Solver

In the `Solve System` procedure, the unknowns Δv are computed. As stated above, implicit integration methods require the resolution of a large, sparse linear system of equations that must be simultaneously fulfilled. An iterative solver, the preconditioned conjugate gradient (PCG) method, has proven to work well, in practice. This method requires relatively few, and reasonably cheap, iterations to converge [1]. The choice of a good preconditioner can result in a significant reduction of the computational cost in this stage. The Block-Jacobi preconditioner has been chosen for the implementation, because of its good parallel behavior [5]. Due to the nature of the problem, blocks have been formed grouping the physical variables (Δv_x, Δv_y, Δv_z) of the particles, so the block dimension is 3×3. A minimization of the norm $(r^{(i)T} P^{-1} r^{(i)})^{1/2}$, being P the preconditioner matrix and $r^{(i)} = Ax^{(i)} - b$ the residual, is the chosen stopping criterion. Heavier particles are so enforced to be closer to the exact solution.

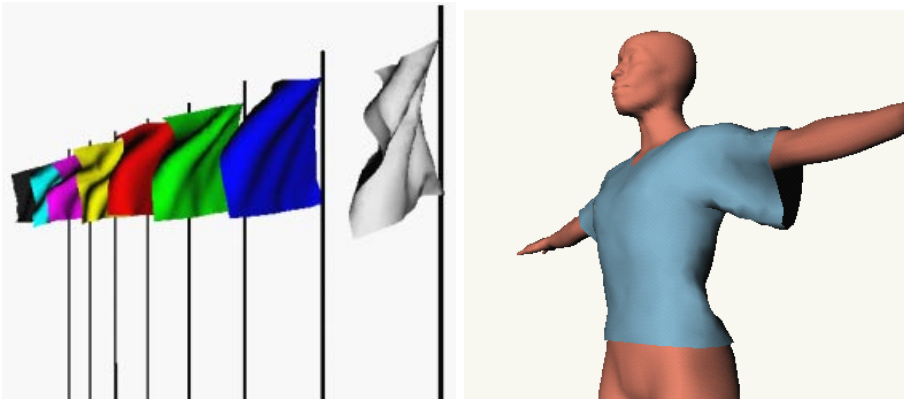

Fig. 2. Flags blowing in the wind and a virtual body wearing a shirt.

3 Parallelization

The parallelization of the model has been performed on a non-uniform memory access (NUMA) multiprocessor architecture. The sequential code (see section 3.1) exhibits irregular access patterns to the data that current parallelizing compilers [4] are insufficiently developed to deal with, leading to non efficient parallel codes. Irregular codes can be parallelized using the inspector-executor paradigm. In run time, the inspector locates non-local data for each processor. Afterwards, an executor must gather non-local data before operate and must scatter the result after it. This strategy introduces a significant overhead, proportional to the number of non-local data accesses. So, a shared memory model and a data parallelism strategy, usual techniques in the *state-of-the-art*, have been used, instead of the run time library. Task parallelism has been also considered for the collision detection stage.

The distribution of the objects in a scenery between the processors is performed using a proportional rule based on the number of elements (particles, triangles,...). The redistribution and reordering of the elements, inside an object among the assigned processors, have been performed using domain decomposition methods. The sparsity pattern of the matrix A is perfectly known, because every non-zero component, with about 12–15 in a row, are the neighbours affecting a given particle for a given tessellation of the object. A Compressed Row Storage (CRS) of the matrix is used in order to minimize memory usage. A striped ordering results in a thin banded diagonal which will produce the parallel distribution with less communication expenses. The Multiple Recursive Distribution (MRD), has higher locality, which will result in a better cache usage [2]. Both have been used in this work with good results, but the choice of the method will depend of the scenery and the computational platform.

3.1 Forces

In the core of the forces evaluation stage, loops like the one presented below[1] are found.

```
for (i=0;i<NumForces;i++) {
   partic0 = List[i][0];
   partic1 = List[i][1];
   force = computeForce(partic0,partic1);
   AccForce[partic0] = AccForce[partic0] + force;
   AccForce[partic1] = AccForce[partic1] - force;
}
```

where `computeForce` does not contain any reference to the array `AccForce[]`. As there are more forces than particles, this is a typical vector reduction operation, and `AccForce[]` is known as *reduction array*. The subscript array `List[][]` depends on the loop index `i`, appearing in both sides in the assignment sentence, through the variables `partic`*i*. In the implicit integration model, both the *rhs* and the system matrix are handled as reduction arrays.

In a parallel environment, two or more processors must synchronize themselves and invalidate their cache copies to actuallize the same memory position (`AccForce`). To overcome this overhead, those particles are replicated in the different processors *id* in a private data structure A_{id}. The loop can now be parallelized, and the final result is the accumulation of the contributions from the processors, $A = \Sigma A_{id}$. A new overhead appears in the accumulation stage, which depends on the number of replicated particles. A good reordering of both `AccForce` and `List` structures reduces the number of replications and minimizes the number of processors where each particle has been hosted.

3.2 Collisions

The lists involved in collisions are distributed among the processors, which compute new contributions to the coefficients of matrix A. As above, such coefficients are replicated in A_{id} to avoid write dependences, and after this step the accumulation into A is done. When collision occurs, the matrix A has a new sparsity pattern because additional coefficients have to be included in unexpected positions. A practical solution is to store them in an additional matrix A^c, also in compressed format.

If lists become longer than a given limit, new lists have to be recomputed from the *AABB* trees. Dealing with hierarchical data structures, it is difficult to use the data parallellism and it is better to keep the sequential code. An additional processor is used to perform this task, without increasing the simulation time. Resulting data are not inmediately required, so the simulation can go on while this task is completed.

[1] Note that this code corresponds with an explicit integration method, but the discussion can be extended to an implicit one.

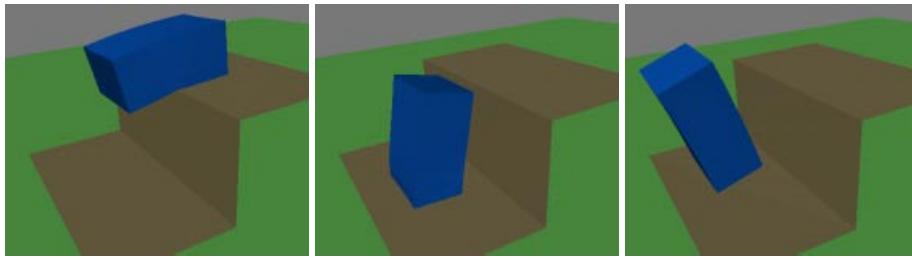

Fig. 3. Some frames of a sponge falling downstairs.

3.3 Conjugate Gradient

The PCG algorithm has been parallelized following a well-known strategy in which the successive parts of the vectors and the properly aligned rows of the matrices are distributed among the processors. This data partition matches with the distribution of the particles in the mesh. Computations inside a processor have been performed using sequential BLAS libraries, which are specially optimized for the underlying hardware.

Using this scheme of PCG [5], very few accesses to remote memories and synchronization points are required along the iterative process. In particular, three global synchronization points, one for every inner product and one for the computation of the convergence criterion, are required. Accesses to remote memories are carried out in these steps and also during the matrix-vector product.

4 Results and Conclusions

Figure 2 shows a human body wearing a shirt and several flags under different windy conditions and figure 3 shows some frames of a simulation of a sponge falling downstairs. In figure 4, the execution time in seconds, of one second of simulated time, and their corresponding efficiency, as a function of the number of processors, for three example models, are shown respectively. These figures correspond to simulations, under windy conditions, of flags of different complexity, with 599, 2602 and 3520 particles respectively. Each curve in a graph corresponds to the original unsorted data, MRD and striped sort distributions. These results has been obtained using an SGI Origin2000 computer with R10000-250Mhz processors. A real time simulation is obtained for the former model, with six processors using striped ordering. The efficiency of the third model shows a superlinear speed-up, which is a consequence of the increment of the ratio computation/communication. It can be observed that striped distribution is clearly faster than MRD for more than two processors, due to the minimization of the accumulation stage overhead.

The computational load of this problem is heavy enough to obtain good efficiencies, even for the simplest models. For larger models, the speed-up grows and gets linear, when problems of typical complexity are dealt with. A proportional

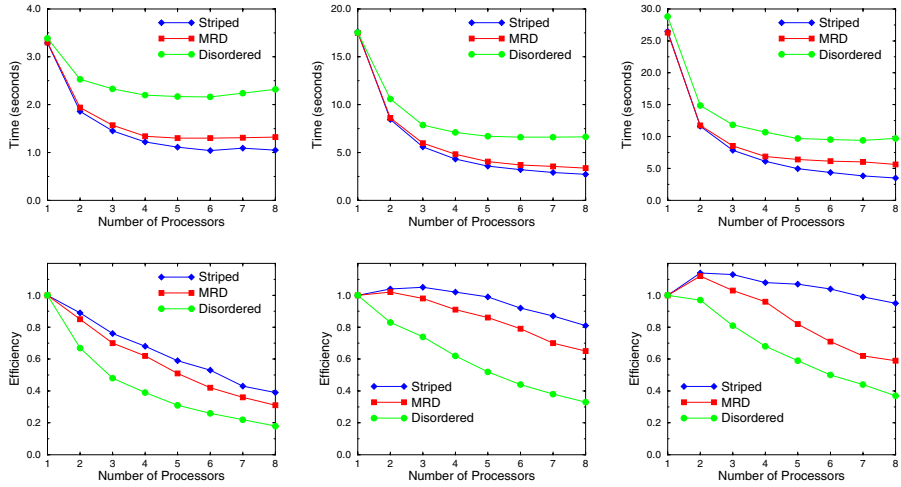

Fig. 4. Execution Time, Speed-up and Efficiency graph for 599, 2602 and 3520 particles.

increment of the number of processors and the size of the problem keeps the efficiency in an almost constant value. This property (isoefficiency, [3]) ensures the validity of the presented model for large simulations.

The use of more recent computers and a higher number of processors for models with more particles/triangles will allow real time simulations. The scenery complexity, considering interaction between several objects, will be improved as the speed of the microprocessors increases.

References

1. Baraff, D., Witkin A.: Large Steps in Cloth Simulation. In Michael Cohen, editor, Computer Graphics (SIGGRAPH 98 Conference Proceeding), pages 43–54. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.
2. Romero, L.F., Zapata E.L.: Data Distribution for Sparse Matrix Vector Multiplication. Parallel Computing Vol. 21, pp. 583–605, 1995.
3. Gupta, A., Kumar, V., Sameh, A.: Performance and Scalability of Preconditioned Conjugate Gradient Methods on Parallel Computers. IEEE Trans. on Parallel and Distr. Systems, Vol. 6, No. 5, pp. 455–469, 1995.
4. Silicon Graphics Inc. MIPSpro Auto-Parallelizing Option Programmer's Guide.
5. Dongarra, J., Duff, I.S., Sorensen, D.C., Van der Vorst, H.A.: Numerical Linear Algebra for High-Performance Computers Software, Environments and Tools series. SIAM, 1998.
6. Volino, P., Courchesne, M., Thalmann, N.: Versatile and efficient techniques for simulating cloth and other deformable objects. Computer Graphics, 29 (Annual Conference Series):137–144, 1995.

7. Volino, P., Thalmann, N.: Collision and Self-collision detection: Efficient and Robust Solutions for Highly Deformable Objects. Computer Animation and Simulation'95: 55–65. Eurographics Springer-Verlag, 1995.