Rapid Parallelization of a Multilevel Cloth Simulator Using OpenMP

R. Lario, C. Garcia, M. Prieto, F. Tirado

Abstract— Certain aspects of a computer-generated world have always been difficult to simulate. Cloth is one such example since, unlike a rigid object, it is flexible and subject to many internal and external forces which drive the fabric into a natural form. As a consequence of these difficulties, realistic simulations demand a significant computational cost, which makes parallel computing highly advantageous [1]. In this paper we analyze the OpenMP-based parallelization of a virtual cloth simulator based on multilevel techniques that attempts to simulate the manner in which cloth drapes. OpenMP not only offers a fast and direct way to treat the hierarchy of data structures employed in our simulator, but also achieves satisfactory efficiencies on the three different platforms studied: a SGI Origin 2000, a SUN HPC 6500 and an IBM SP2.

Keywords— **Cloth modeling, parallel multilevel techniques, OpenMP.**

I. INTRODUCTION

The goal of this paper is to analyze the OpenMP-based parallelization of a cloth simulator that attempts to reproduce the manner in which cloth drapes. Cloth modeling has received considerable attention in the computer graphics community over the last few years. The reasons are numerous, but just to cite a few examples (for an extensive survey of the current state of cloth modeling and its applications see reference [2]) we can mention the appearance of clothing worn by virtual actors, which is of considerable interest in the animation-entertainment industry. The need is even greater within the fashion industry, where computer-aided design tools have to generate (as accurately as possible) the forms of cloth objects so that the designer can easily experiment with a variety of fabrics and patterns (on a 3D virtual mannequin) before the garment is actually manufactured.

Departamento de Arquitectura de Computadores y Automatica
Facultad de C.C. Físicas. Universidad Complutense.
Ciudad Universitaria s/n 28040 Madrid.
{rlario,garsanca,mpmatias,ptirado}@dacya.ucm.es

Modeling realistic clothes can be divided into two separate problems; cloth motion modeling and collision detection to stop cloth from penetrating into other neighboring objects and to prevent collisions between different parts of the cloth itself [1,2]. Although our long-term goal is to study the whole process, in this paper we have not considered any collision detection algorithm, our analysis being limited to the first stage of the problem.

Among the different approaches to simulating flexible materials we have chosen a physical-based method that was first introduced by C. Feymann [3] and subsequently improved by H. Ng et al [4,5]. From a numerical point of view, this model involves the solution of an optimization problem. The simulator has to minimize an energy function, which depending on the configuration chosen, has to satisfy certain constraints. Following [5], the minimization technique that we have employed is based on a standard Polak-Ribiere method that is accelerated with a multilevel scheme [6]. Our algorithm is presented in section 2, along with a comparison with the H. Ng approach.

The idea of applying parallel computing to reduce the expensive computational cost of cloth simulators and to deal with more realistic scenarios is not new [1,5]. In section 3 we discuss some key factors that have been necessary to consider in order to achieve an efficient parallel implementation of our simulator. The main difficulties are caused by the multilevel treatment of the cloth, which accelerates the convergence of the optimization process but which limits the achievable parallel efficiency.

In section 4 we have investigated the performance of our simulator on a SGI Origin 2000 (O2K) [7] equipped with 400 MHz MIPS R12000 processors and 8 Mbytes of L2 cache, and two different UMA architectures: a SUN HPC 6500 SMP system [8] (equipped with 400 MHz UltraSPARC-II processors and 8 Mbytes of L2 cache) and one 16-way SMP node of an IBM SP2 system [9] based on the Nighthawk Power3 processor (running at 375 MHz and equipped with 8 Mbytes of L2 cache). Compared to the message-passing paradigm, OpenMP not only provides a straightforward way to deal with the hierarchy of grid levels, but also achieves satisfactory

efficiencies. This paper ends with some conclusions and hints about future research.

II. MODEL PROBLEM

The starting-point for our cloth simulator is an energy-based physical model that describes a piece of cloth as a 2D grid in a 3D space (see figure 1).

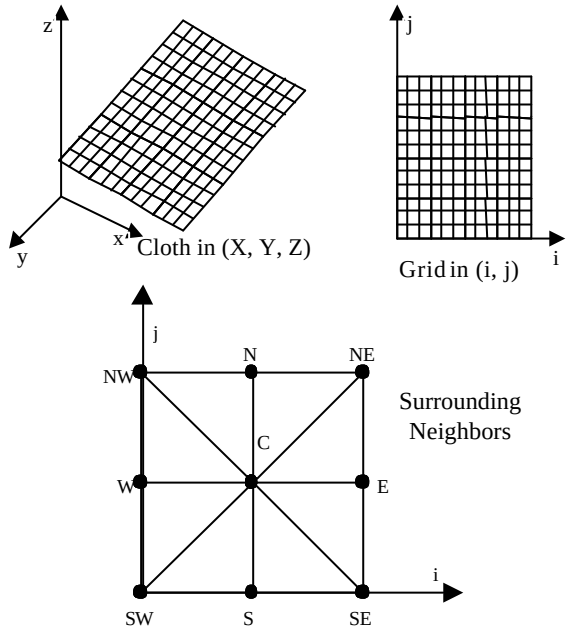

Fig 1. *A piece of cloth as a 2D grid in a 3D space (on the top) and surrounding neighbors (on the bottom).*

Every grid point is characterized by an individual energy related to certain physical parameters (elasticity, bending and density) that can be evaluated as a function of the relative position of each grid point with its eight surrounding neighbors [2,3,4,5] as described by the following equations:

$$\text{Energy}(C) = Ks \cdot \text{Strain}(C) + Kb \cdot \text{Bending}(C) + Kg \cdot \text{Gravitational}(C)$$

$$\text{Strain}(C) = (s - |C-E|)^2 + (s - |C-W|)^2 + (s - |C-N|)^2 + (s - |C-S|)^2 + r\left((\sqrt{2}s - |C-NE|)^2 + (\sqrt{2}s - |C-NW|)^2 + (\sqrt{2}s - |C-SW|)^2 + (\sqrt{2}s - |C-SE|)^2\right)$$

$$s = \text{equilibrium distance}$$

$$\text{Bending}(C) = (p - \text{angle}(C,E,W))^2 + (p - \text{angle}(C,N,S))^2 + (p - \text{angle}(C,NE,SW))^2 + (p - \text{angle}(C,NW,SE))^2$$

$$\text{angle}(p0,p1,p2) = \arccos\left(\frac{(p1-p0)\cdot(p2-p0)}{|p1-p0|\cdot|p2-p0|}\right)$$

$$\text{Gravitational}(C) = s^2 \cdot \text{density}(C) \cdot z(C)$$

The final equilibrium form of the cloth can be obtained by finding its energy minimum. As in [4], the core of the simulator is a point-by-point minimization algorithm that modifies (relaxes) one grid point at a time to minimize its energy as much as possible. The localness of this relaxation process allows an efficient parallelization of the algorithm, but from a numerical point of view it may present (in general) two important difficulties [6]:

1. Slow convergence: in general, point-by-point minimization does not handle large-scale features well.
2. False convergence: instead of converging to the true global minimum.

Fortunately, we have only observed the first of these in our particular problem. The minimization technique proposed in [4] copes with this first issue by combining a standard Polak-Ribiere method with a multilevel scheme (see figure 2). The intuitive idea is to employ coarse grids to determine the overall form of the draped cloth, while the finest grid or target grid is only necessary to obtain small-scale details in the form. Quoting A. Brand, *"the multilevel technique supplements the local processing with increasingly larger scale processing"* [6].

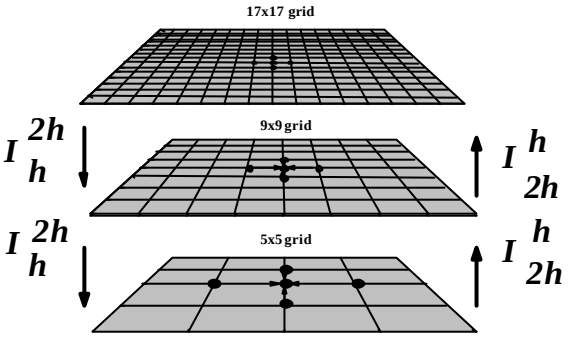

Fig 2. *Hierarchy of grids employed by the multilevel scheme.*

The combined algorithm, which we have denoted as MPR (Multilevel Polak-Ribiere) can be define as:

Algorithm 1 MPR (n^1, n^2, g)***:*** *Multilevel Polack-Ribiere V-cycle for a given target grid, where n^1 and n^2 denote the number of iterations of the Polack-Ribiere method on the descendant and ascendant parts of the cycle and g denotes the number of iterations on the coarsest grid.*

```
/* down */

for L=0..(num_levels-2)

  relaxation  Energy of grid[L] with
              Polak-Ribiere Method (n1
              times)
  restrict    grid[L] to grid[L+1]
```

```
  copy       grid[L] to grid_old[L]

/* coarsest level */

L = num_levels-1
store grid[L] in grid_old[L]
relaxation  Energy  of  grid[L]  with
           Polak-Ribiere  Method  (g
           times)
/* up */

for L=(num_levels-2)..0

  difference  diff[L+1] = grid[L+1] –
              grid_old[L+1]
  prolongation   diff[L+1] to diff[L]
  update      grid[L] = grid_old[L] +
              diff[L]
  relaxation  Energy of grid[L] with
              Polak-Ribiere Method (n²
              times)
```

The transfer operators are used to connect the grid levels. The restriction operator transfers values from a finer to a coarser level while the prolongation operator maps data from a coarser to a finer level.

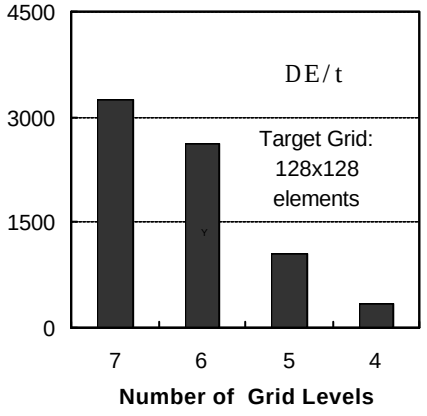

Fig 3. *Benefits of the multilevel technique using a MPR(1,1,5) cycle. DE/t denotes the reduction of energy obtained per unit time (seconds). These results have been obtained using a round table simulation on a SGI O2 workstation.*

Figure 3 shows the benefits of the multilevel technique using the average energy reduction per unit time (in seconds) as a metric. The results have been obtained with a MPR(1,1,5) cycle using a 128^2 round table simulation (see figure 3). The execution time measurements have been made on a SGI O2 workstation equipped with a MIPS R10000 microprocessor running at 250 MHz. As can be observed, going down to the coarsest grid dramatically improves the performance of the algorithm.

To reduce progressive asymmetry errors the point-by-point minimization employed in [4] was based on a Jacobi-like approach for the update of the grid point coordinates (following the analogy with the well-known Jacobi iterative method for solving linear systems of equations): it used two grids, one for the previous position and another for the new one, so that all the new positions were computed taking only the original ones into account.

We have improved the H. Ng approach by overwriting the point coordinates as soon as new positions are calculated. In this way, not only are the memory requirements of the optimization process reduced but so too is the execution time to find the solution. Nevertheless, we should remark that the order in which points are selected for relaxation has to be chosen carefully to avoid asymmetric errors. Among the different ordering strategies that we have investigated, the best results have been obtained using a four-color interleaved update and an alternating zebra-like ordering (red and black rows are not only interleaved but also processed in opposite directions).

Figure 4 compares the three different update strategies using a MPR(1,1,5) cycle for different grid sizes. Both the zebra and the four-color updates outperform Jacobi, the former being the best choice in most cases. Compared to Jacobi, the improvement varies from around 40% for a 128^2 cloth to more than 50% for a 1024^2 size.

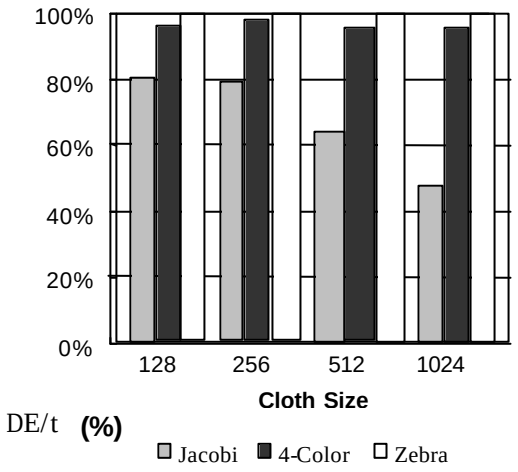

Fig 4. *Performance of three different update strategies using as a metric the reduction of energy (in percentage) obtained per second (DE/t). These results have been obtained using a round table simulation on a SGI O2 workstation.*

Finally, and just to give an example of the simulator capability, figure 5 shows cloth draped over a round table (top illustration) and over a pair of parallel planks (bottom illustration). As can be seen

from the figures, the appearance of the cloth is highly satisfactory.

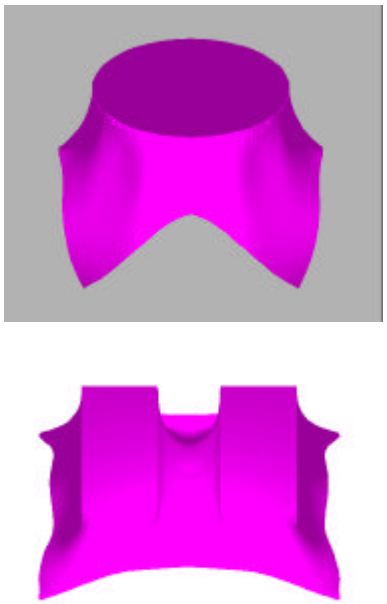

Fig 5. *Draped cloth over a round table (top illustration) and over a pair of parallel planks (bottom illustration).*

III. PARALLEL IMPLEMENTATION

One of the main advantages of the point-by-point minimization technique lies on its inherent degree of parallelism, which can be easily expressed using either OpenMP directives (distributing the iterations of the `for` loop that sweeps the different elements of the cloth across threads) or the message passing paradigm (applying the general principles of domain decomposition). Indeed, the relaxation process on the finest level, i.e. without considering the multilevel strategy, scales almost linearly on the investigated systems using both OpenMP and MPI. The contribution of the other components of the MPR algorithm (copy, update, difference and grid transfer operators) to the execution time are less significant than the minimization cost. But in any case, they are also parallel by nature and like the relaxation on the finest grid, they do not present any kind of complications either.

However, given a certain number of processors the parallel version is only worthwhile from a certain cloth size. Due to the multilevel treatment of the cloth, this means that from some MPR level, which we have denoted as the *critical level* (to be precise, in our code the critical level is the level in which all processes/threads have to process two lines), a parallel implementation cannot improve the execution time of its sequential counterpart. Indeed, it can deteriorate the performance due to an unsatisfactory communication-to-computation ratio (from a message-passing point of view) or the overheads associated with short loops (from an OpenMP point of view). This problem, which is very common in other multi-level algorithms [10,11], may be alleviated in some cases by setting the number of grid levels such that the maximum level is the critical one. However, for the MPR algorithm, the execution time improvement of this approach does not compensate the numerical deterioration of the algorithm (note that, as we have shown above, from a numerical point of view by far the most efficient strategy is to choose the coarsest level as coarse as possible).

.

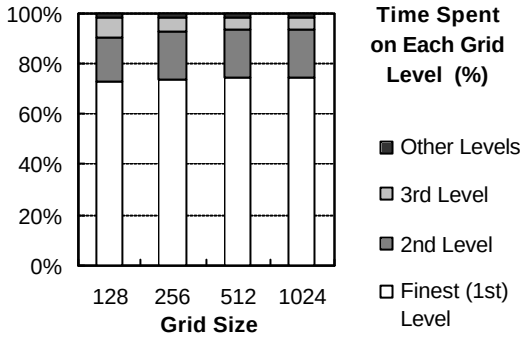

Fig 6. *Time spent by the MPR algorithm on each grid level for different grid sizes. The measurements have been made on a SGI O2 workstation.*

Fortunately, as figure 6 shows, the time spent by the MPR algorithm on the very coarse levels (below the critical one) is only a small fraction of the total execution time which makes the implementation of an agglomeration strategy attractive [10], i. e. adjusting the number of processors/threads as the problem size decreases and even using only one single processor/thread on very coarse levels.

From a message-passing point of view, this scheme makes implementation laborious since at sub-critical levels it is necessary to dynamically rearrange the communication patterns and grid distributions. However, it can be implemented easily with OpenMP, just by changing the number of threads below the critical level. Our first attempt consisted in controlling the number of threads with the `omp_set_num_threads` function, which seems to fit perfectly with this kind of hierarchical applications, but the experimental results have been disappointing so far. In this work, the agglomeration has been implemented through the `#pragma omp master` directive, i.e. from the critical level, only the master thread does the calculations.

As a reference, we have compared the performance of the OMP version with a MPI

implementation based on a 1-D decomposition and a data replication operation (see figure 7), so that from the critical level all the processes can independently perform the rest of the computation. Although it could be argued that a better MPI implementation may be obtained by adjusting the number of processors as the cloth size decreases, this implementation is conceptually more similar to the investigated OpenMP version.

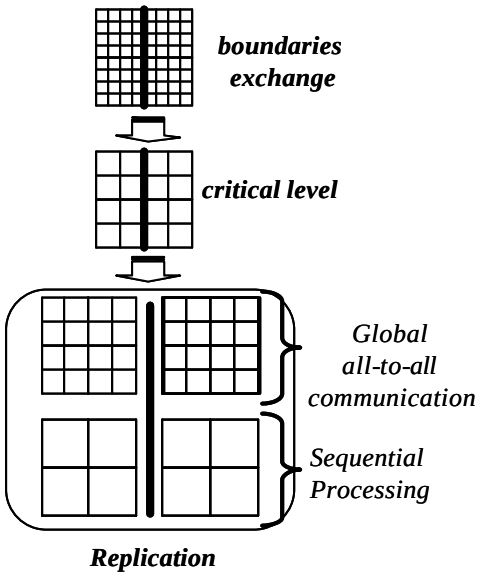

Fig 7. Scheme of the replication-based MPR simulator.

IV. EXPERIMENTAL RESULTS

A. SGI Origin 2000

Figure 8 shows the parallel efficiencies achieved with the OpenMP and the MPI simulators in the SGI O2K. As can be seen, its NUMA architecture does not impose any important performance degradation in the OpenMP version, whose efficiencies can be even better than those obtained with MPI.

Focusing on the OpenMP simulator, data distribution can be effectively done when data are initiated through the Origin first-touch policy using an appropriate parallel loop whose iterations are distributed conveniently among the processors. Nevertheless, we should remark that this technique only allows a page-based distribution (there is not element granularity but page granularity), which may be inadequate, especially when coarse levels are processed. Obviously, instead of forcing a reduction in the degree of parallelism with the `master` directive in order to steer clear from this problem, it can be avoided by padding the array to separate data blocks by at least one page of memory [12]. However, it is our personal opinion that this technique requires a level of manual tuning not easily justifiable with the easy-to-parallelize principles of OpenMP.

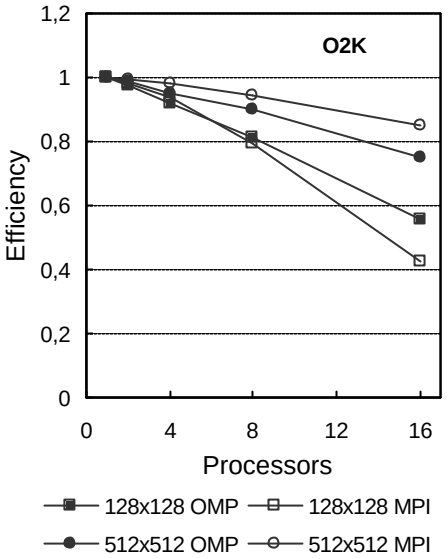

Fig 8. Parallel efficiency obtained by the OpenMP and MPI versions of the MPR algorithm on the SGI O2K.

Figure 9 shows an execution time profile of the MPI-based simulator for the 128^2 cloth size (the smallest size considered). The communication cost shown in this figure only accounts for the boundary exchange up to the critical level, while the replication cost accounts for the global communications required to replicate data and the computations below the critical level (i.e. computations than cannot be done in parallel). As can be seen, replication cost is the main reason behind the poor performance exhibited in the sixteen-processor case. This measure explains why the OpenMP version achieves better performance in this case, despite the problems mentioned above.

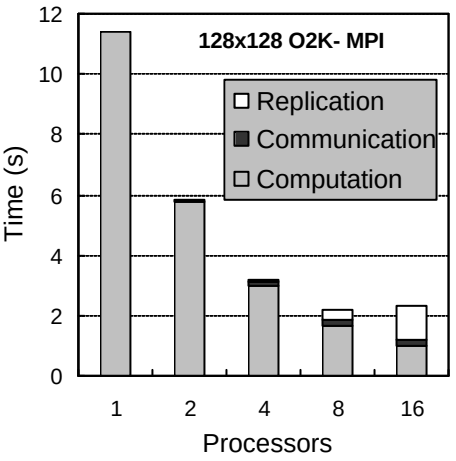

Fig 9. Execution time profile obtained with the MPI-based simulator on the SGI O2K.

For larger cloth sizes, the impact of the data replication is not so important since its overhead (in

absolute values) is independent of the cloth size (note that the size of the critical level only depends on the number of processors). For the sixteen-processor simulation for example, replication cost ranges from the significant 45% illustrated in figure 9 to a mere 5% for the 512^2 cloth size.

Finally, it is also interesting to note that the boundary-exchange related cost is not an important issue in this case. For small problem sizes, it is insignificant compared to the replication cost (for the 128^2 case, it just represents a mere 7% in the sixteen-processor simulation), while for large problem sizes they are of the same order.

B. SUN HPC 6500

Figure 10 shows the efficiency data obtained in the SUN HPC 6500 system. Results are (qualitatively) very similar to those obtained in the SGI O2K, i.e. for small problem sizes OpenMP outperforms MPI due to the replication cost (see figure 11), while for large problem sizes, the overhead associated with data replication becomes insignificant and the MPI simulator achieves better performance.

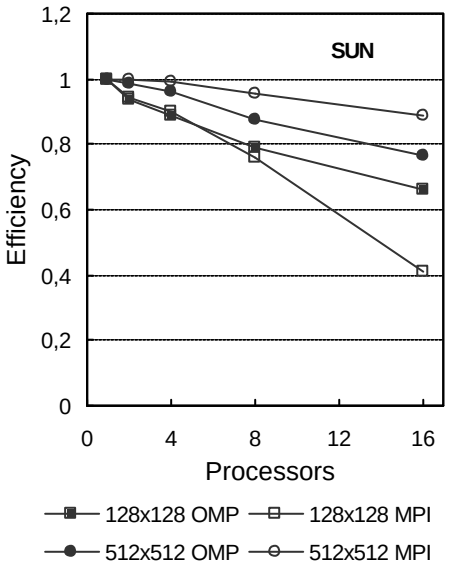

Fig 10. Parallel efficiency obtained by the OpenMP and MPI versions of the MPR algorithm on the SUN HPC 6500.

Nevertheless, as could be expected, OpenMP scales better in this system than in SGI O2K due to the benefits of the UMA architecture: even for the 128^2 problem, OpenMP achieves a satisfactory efficiency of about 0.67 in the sixteen-processor case. In addition, the benefits of the agglomeration technique (i.e. using the `master` directive) experienced in the SGI O2K are in this case almost negligible.

Regarding the MPI version and compared to the SGI O2K, the communication and replication costs are slightly better in this system although we should note that the computation cost is around 50-70 % lower in the SGI O2K (see figure 15).

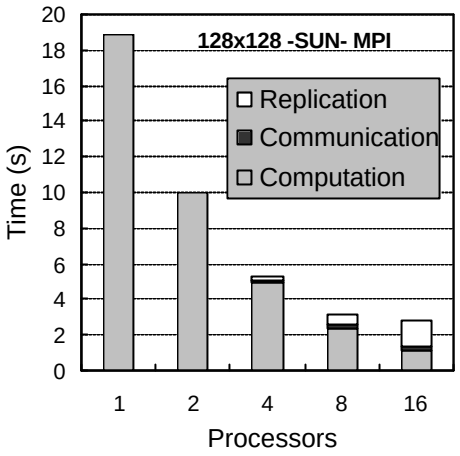

Fig 11. Execution time profile obtained with the MPI-based simulator on the SUN HPC 6500.

C. IBM SP2

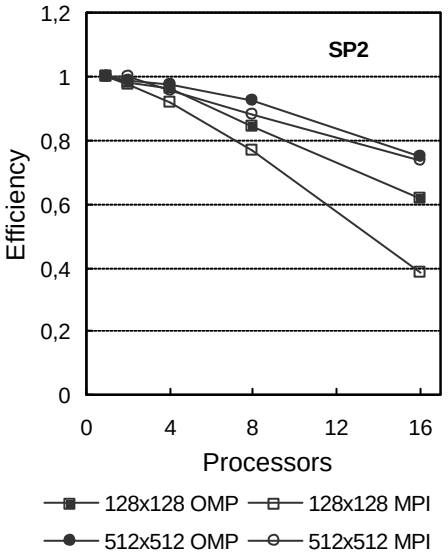

Fig 12. Parallel efficiency obtained by the OpenMP and MPI versions of the MPR algorithm on the IBM SP2.

Figure 12 shows the parallel efficiency obtained on a 16-way SMP node of an IBM SP2 system. Unlike the other systems under study, OpenMP has always outperformed MPI, although the performance becomes similar as the problem size increases. The results for the OpenMP version are very similar to those obtained on the SUN HPC system, the benefits of the agglomeration technique also being negligible in this case.

Regarding the MPI version (see figure 13), replication also limits the performance for small cloth sizes. However, the communication cost of the MPI version is larger than in the other systems. For

example, for the 512^2 size, the sixteen-processor simulation expends 20% of the execution time in communication above the critical level, while for other systems the percentage is only around 10%.

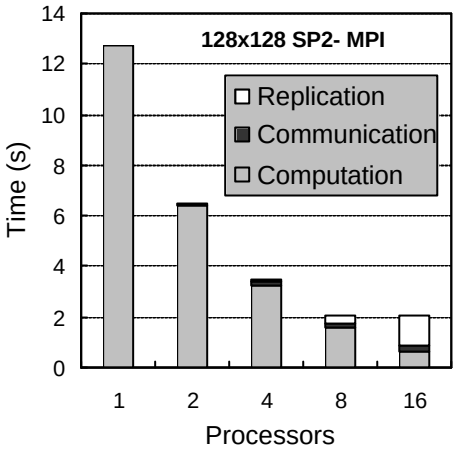

Fig 13. Execution time profile obtained with the MPI-based simulator on a SMP node of an IBM SP2.

D. Performance Comparison

Finally, we have compared the results obtained from the three systems using as a metric the execution time gain with regard to the SUN HPC system:

$$Gain(system) = \frac{T(SUN\_HPC\_6500)}{T(system)}$$

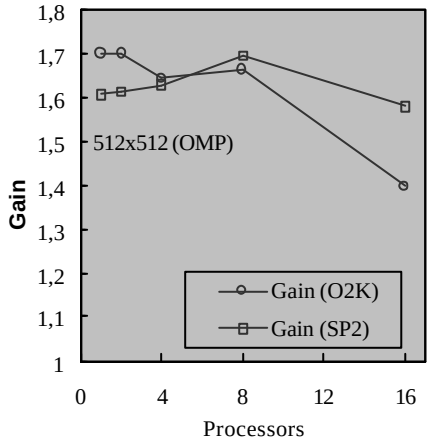

Fig 14. Gains achieved in the SGI O2K and the IBM SP2 with regard to the SUN HPC system using OpenMP.

As figure 14 shows, the performance of the SP2 system using OpenMP is qualitatively similar to the SUN HPC but with an improvement in the execution time of about 60-70%. In the SGI O2K the gain also starts as 70%, but it drops with the number of processors, reaching 40% for the sixteen-processor case.

For the MPI simulator, the results are just the opposite. In the SGI system the gain remains above 60%, while in the SP2 it drops linearly with the number of processors due to the communication cost.

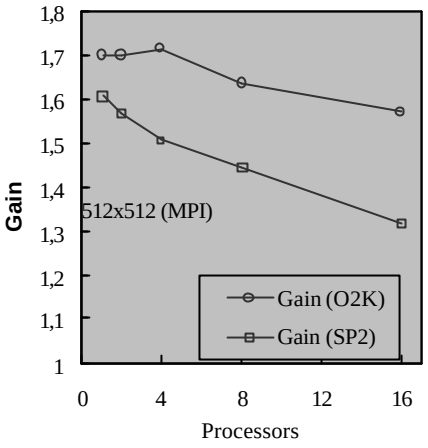

Fig 15. Gains achieved in the SGI O2K and the IBM SP2 with regard to the SUN HPC system using MPI.

V. CONCLUSIONS AND FUTURE RESEARCH

To sum up the previous sections, the overall objective of our research is to develop a realistic cloth simulator. Parallel computing is highly recommended for this problem in order to reduce the expensive computational cost and deal with more realistic scenarios.

As other hierarchical applications, what limits the scalability of the MPR algorithm is the multilevel treatment of the cloth. In any case, for the model problem studied, OpenMP offers an effortless way to get a parallel version of the code and also achieves satisfactory efficiencies using up to 16 processors in the three different investigated platforms.

In the IBM SP2, the OpenMP version has always achieved better results than the MPI simulator. However, in the SGI O2K and the SUN HPC 6500, the message-passing version of the simulator outperforms its OpenMP counterpart for large problem sizes. The page-based decomposition of the SGI O2K only limits its performance of the OpenMP version when the ratio between the cloth size and the number of threads is small. The main problem of the MPI version is the cost of data replication, but this cost only depends on the number of processors, which makes this overhead insignificant for large problem sizes.

Nevertheless, we should remark that for moderate-size parallelism, which is the target of this research (the size of current departmental servers, which is the most common computing platform on CAD-CAM environments, ranges from 4-way to 16-way SMP

system) the differences in performance do not justify the extra coding effort required by MPI.

Finally, we should note that the algorithm presented in this paper has to be considered as the building block of a more general simulator that has to include collision detection and to allow multi-block simulations. Indeed, we are currently studying a multi-block MPR algorithm where the cloth is divided into a set of pieces that are treated by the MPR algorithm independently (apart from the block connections). In this way, the simulator can deal with geometric complexities and can be applied to more general cloth samples than rectangular grids.

VI. ACKNOWLEDGMENTS

This work has been supported by the Spanish research grant TIC 99-0474 and by the European Community programme "*Access to Research Infrastructure action of the Improving Human Potential Programme (contract No HPRI-CT-1999-00026)* ". We would also like to thank Centro de Supercomputación Complutense (CSC), Centre Europeu de Parallelisme de Barcelona (CEPBA) and EPCC for providing access to the parallel systems employed in this work.

VII. REFERENCES

[1] S. Romero, L. F. Romero and E.L. Zapata."Fast cloth simulation with parallel Computers". Proc. 6th Int'l Euro-Par Conference (Euro-Par'2000). Munich, Germany, 2000.

[2] D. H. House and D. E. Breen. "Cloth Modeling and Animation". Published by A.K. Peters, Ltd. June 2000.

[3] C. Feynman. "Modelling the appearance of Cloth". Master's thesis, Dept. of EECS, Massachusetts Inst. of Technology, Cambridge, Mass., 1986.

[4] H. N. Ng, R L. Grimsdale and W. G. Allen. "A system for modelling and visualization of cloth material". Computers and Graphics, Vol. 19 (3), pp. 423-430, 1995.

[5] H. N. Ng and R. L. Grimsdale. "Computer Graphics Techniques for Modeling Cloth". IEEE Computer Graphics and Applications, Vol. 16 (5), pp. 28-41, September 1996.

[6] A. Brandt. "Multiscale Scientific Computations: Review 2000". Proc. 10th Copper Mountain Conferences on Multigrid Methods, Copper Mountain, CO, USA. April 2001. Available at http://ww.mgnet.org.

[7] J. Laudon and D. Lenoski. *"The SGI Origin: A ccNUMA Highly Scalable Server"*, in Proceeding of ISCA'97.May 1997.

[8] Sun Microsystems, Inc. "Sun Enterprise 6500". Information available at www.sun.com/servers.

[9] G. Mojtabaeezamani. "RS/6000 SP 375 MHz POWER3 SMP, Thin and Wide Node Architecture". February 2000. RS/6000 SP Node Hardware Development. Poughkeepsie, NY.

[10] M. Prieto, R. Santiago, D. Espadas, I. M. Llorente and F.Tirado. "Parallel Multigrid for Anisotropic Elliptic Equations". Journal of Parallel and Distributed Computing, vol. 61, no. 1, pp. 96,114. Academic Press, 2001.

[11] M. Prieto, R. Santiago, I. M. Llorente and F.Tirado. "A Multigrid Solver for the Incompressible Navier-Stokes Equations on a Beowulf-class System". To be published by the IEEE Computer Society in the Proc. of the Icpp 2001 (International Conference on Parallel Processing). Valencia, September 2001.

[12] SGI, Inc. "Origin 2000 and Onyx2 Performance Tuning and Optimization Guide". Available at SGI's Techpubs library: http://techpubs.sgi.com.