社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2004－HPC－98　(2)
2004／4／13

An approach towards fast simulation of virtual cloth with adaptive mesh refinement and coarsening on Fujitsu HPC2500

Alam MUJAHID,[†] Koh KAKUSHO,[††] Michihiko MINOH,[††]
Yasuhiko NAKASHIMA,[†††] Shin-ichiro MORI[†]
and Shinji TOMITA[†]

Parallel simulation of shape and force acting on cloth is proposed for Fujitsu HPC2500 by using OpenMP. Parallelization is feasible due to iterative and higher computations needed for cloth model. Adaptive mesh refinement together with coarsening (AMRC) is employed to obtain the optimum mesh density of cloth. Mesh size changes at runtime due to AMRC, which initiates the necessity of load-balancing. Therefore, a load balancing scheme based on 'Active-Lists' has been implemented. Comparison with built-in scheduling constructs in OpenMP shows its good performance for simulating the shape as well as force of virtual cloth.

1. Introduction

Fast as well as realistic simulation of cloth is an interesting issue in computer graphics. Cloth has been represented by the mass-spring model, finite element model and particle-based model. Its shape has been simulated by using the force integration[3),10)] or energy minimization[1),5),9)]. However, only visual information is insufficient for virtual manipulation and shape as well as force are necessary to represent the cloth in virtual environment. We have considered, for the first time, the relation of force with shape deformation for cloth[5)].

A realistic cloth can only be represented by a denser mesh, which is computationally expansive. A closer look on variety of cloth applications indicates that some parts of cloth have active role in simulation while others contribute very little. This fact expresses the necessity of adaptive mesh. Adaptive refinement has been employed in different ways and for different applications in cloth simulation[3),4),7)]. Villard[10)] has modified the mechanical model for more accuracy. Volkov[11)] has introduced the refinement and simplification for cloth meshes but without adjusting the mechanical model. We have already implemented a combination of adaptive mesh refining with coarsening (AMRC) that adjusts mechanical model accordingly[6)]. Simulation begins with finest mesh and mesh density varies adaptively during the course of simulation.

AMRC reduces the sequential cost but larger, complex and real time applications require more speed. Developing a parallel cloth simulator can solve this problem and OpenMP is a good choice due to its easier implementation. Romero[8)] has implemented fast simulation of flag represented by a very small size. Cloth is simulated as multi-level meshes using OpenMP by Lario[4)] similar to multi-grid model[7)]. Its efficiency may degrade because some regions of cloth are refined or simplified unnecessarily.

Division of work can be decided easily for uniform meshes but AMRC generates uneven mesh density that needs to manage the load balancing. Therefore, we are developing a load balancing scheme using **Active-Lists**. Lists of elements are created according to the run-time density of mesh and equal number of elements are assigned to each processor. This parallel algorithm works well for draping of cloth.

2. Cloth Model

Cloth is a deformable object by stretching, bending and shearing to describe the basic property of cloth. Simulation is performed by using particle-based model, empirical data captured by Kawabata Evaluation System(KES) and minimization algorithm. KES curves give the unidirectional and hysteretic relation between force and shape of cloth. KES data has been utilized by[1),9)] without considering the all

† Graduate School of Informatics, Kyoto University
†† Academic Center for Computing and Media Studies, Kyoto University
††† Graduate School of Economics, Kyoto University

conditions applicable during the KES measurement process. We are assuming three conditions to develop the model that are 1) Initial state at each simulation step serves as internal variable that keeps the track of previous history. 2) KES curve works as boundary value. 3) All other hysteretic cycles lie within this boundary.

Cloth is modeled as a mesh of **I X J** particles and each particle represents the crossing of warp and weft thread. We are defining three types of elements; particles, edges and triples in our model. An edge connects two adjacent particles and its length describes the stretch. Similarly three adjacent particles make a triple to compute the bending angle. $\mathbf{X_{ij}}$ is the position of a particle, f^s is the internal stretching force of an edge and f^b is the bending force for a triple. All other variables are function of above variables and detail is given in the work[5)].

Three energy/cost functions, motion & gravitation (E^n), stretching (E^s) and bending (E^b), are considered for simulation. Since KES stretch curve is a relation from force to shape while KES bend curve describes the relation from shape to force, so stretching and bending properties are incorporated to have the relations in both directions. On the other hand, Newton's law describes the bilateral relation between force and shape. Therefore, we are able to involve the force as well as shape at the same time. Other properties may be added in the model in similar way. Each cost function is defined based on the KES data and represents the amount of violation from KES data. The cost is zero when the calculated data lies within or on the KES curve and cost increases outside the KES curve. The total cost function is the sum of individual cost functions.

Algorithm 1 : Simulation

```
for(Simulation Loop) {
  do{ Itr ++;     /* Minimization Loop */
      SaveVariables(position, force);
      ComputeGradient(clothMesh);
      ComputeMinima(clothMesh, gradient);
      UpdateVariables(position, force);
      ComputeCost(clothMesh);
  } while(Cost> Tolerance && Itr< MaxItr);
  DisplayCloth();
 } \* End Simulation */
```

Simulation process, in sequential order, is described as pseudo code in Algorithm 1. The current variables are saved and gradient is computed numerically. The values of desired variables corresponding to the equilibrium state of the cloth are obtained through cost minimization by using the conjugate gradient method. Then cost is computed after updating the variables. This process continues until cost reduces to tolerance or maximum iterations are reached.

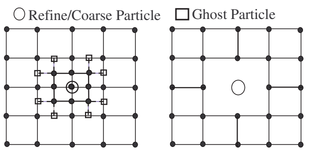

Fig. 1 Mesh refining(left) and coarsening(right)

3. Adaptive Meshes

We are implementing the adaptive refining together with coarsening by utilizing bending cost as criterion to refine or coarse the mesh. Flat region of cloth is represented with coarser mesh and deformed region is represented with denser mesh. Adaptive mesh implementation needs to adjust the mass conservation, force distribution and mechanical model.

Adaptive mesh refinement adds eight active and eight ghost particles in deformed region of cloth. Ghost particles just maintain the topology of mesh. Length of new edge reduces to half and a new particle represents 1/4 of the area as compared to one level coarser particle. On the other hand, adaptive coarsening has maximum density for initial mesh and omits particles that have very small bending cost. Omission of a particle removes the four edges and six triples connected with this particle as shown in Fig. 1.

3.1 Adjusting Mechanical Model

KES characteristics, for both bend and stretch, are available for a specific size of cloth. Therefore, our model uses KES data interpolated for adaptive size of meshes.

In contrast to mass-spring model, we calculate bending angle formed by three particles of a triple that permits to add or omit a particle from mesh. Eliminating a particle merges neighboring triples and edges, for example see Fig. 2. Triple T_2 is merged in the triples T_1 and T_3, as a result force corresponding to T_2 is divided equally between T_1 and T_3. Similarly merging of edges E_2 and E_3, adds forces to resultant edge.

Refinement splits an edge into two with half

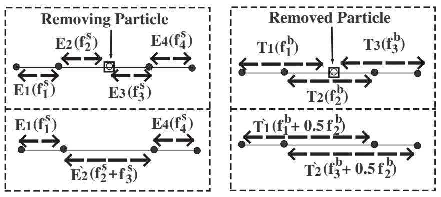

Fig. 2 Force adjustment for removing a particle

length and space having four edges is represented by sixteen edges after refinement as shown in Fig. 1. Therefore, uniform force distribution over the area assigns $f^s/4$ to the finer edges. The bending parameters in refining can be adjusted in the similar manner as explained above for coarsening but in opposite direction.

As refinement reduces the area represented by a particle to $1/4$, so mass should be $m/4$ for finer particle to preserve the total mass. However, different masses show different response that should be tackled carefully. To have the same response, one way is to take the heavier mass as for coarser mesh and adjust it when calculating the stretching and bending. The other way is to take the lighter masses (like concept of mass density) as for finest mesh by considering that whole mesh is refined and coarser area contains more non-active particles. We are using the latter concept that also remains valid for adaptive coarsening.

3.2 Adaptive Mesh Refinement and Coarsening (AMRC)

We have already adopted the adaptive refining and coarsening separately and together to cloth simulation[6]. Both of these change mesh density in one direction that may represent the complex shape inaccurately. Thus implementing AMRC produces optimum mesh density and flexible model. Initial cloth with flat shape is configured as maximum resolution for simulation. When it drapes over an object, deformation takes place around the boundary of object. Therefore, mesh needs coarsening to lower the density in the region away from the boundary of object. Elements, removed during coarsening, are not completely deleted from data structure but those are just made inactive. This information is later used for refinement. Similarly, in case of refining, previously in-activated elements are made active in data structure. As elements are neither deleted nor added by AMRC, memory location remains unchanged. Mesh refinement and coarsening are employed accordingly if criteria for refining/coarsening is satisfied. This process continues throughout the simulation and few possible mesh transitions as an example are shown in Fig. 3.

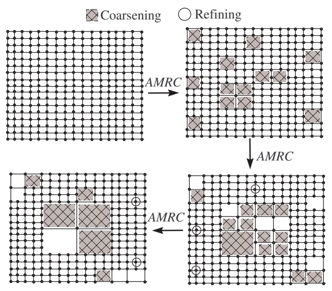

Fig. 3 Adaptive mesh refinement and coarsening

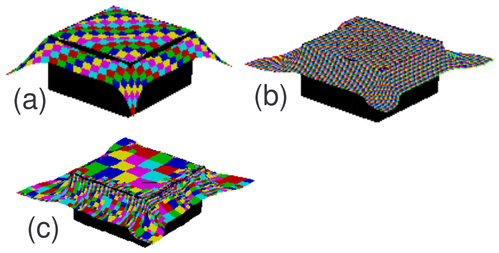

Fig. 4 Simulated image with (a)coarser mesh (b)finer mesh and (c)AMRC

A rectangular cloth, taking 77×77 as maximum and 20×20 as minimum mesh density, draping over a box is simulated on the pentium personal computer without using the adaptive meshes. The simulated images with coarser and finer mesh are shown in Fig. 4. Simulation with coarser mesh is faster but cloth is penetrating in the box and requires denser mesh for realistic simulation. On the other hand, simulation with finer mesh gives the best quality but needs maximum time. Then sequential algorithm for simulation with AMRC is executed and simulated image is shown in Fig. 4. Algorithm observes criteria for refining or coarsening after three iterations by assuming that there is no abrupt change in the model in few iterations. Coarsening is dominant initially because most of the cloth is flat. Later high deformation occurs near the edges of the box that need refinement.

A cost function represents the violation of cloth parameters from experimental (KES) data. Simulation would be realistic when cost function approaches to zero. Therefore, minimized value of cost function indicates the qual-

Table 1 Comparison for different meshes

Mesh	Time [sec]	No. of Particles	Quality Error
Finer Mesh	2940	5939	0.0
AMRC	829	5939 – 1378	+0.057
Coarser Mesh	176	400	+0.071

ity of simulation by considering that KES data corresponds to best quality. The cost value of finest mesh, achieved in experiment, is used as zero quality error for reference. Comparison among different simulations proves that AMRC produces good speed and quality (see Table 1).

4. Parallel Simulation

OpenMP, a shared memory based API, is utilized for parallel programming to have faster cloth simulation. When a parallel region is called in OpenMP, a master thread creates slave threads, divides iterations and synchronizes threads at the end of parallel region. It requires extra cost, called *Parallel Overhead*, which increases with number of threads[2]. Some performance tuning is necessary to run the program acceptably fast. However, over-all performance depends on the percentage of code that can be made parallel, granularity, load balancing, locality and synchronization among processors.

4.1 Synchronization

Cloth mesh is divided into blocks of 50×50 particles. Block size may vary depending on the application. Memory is allocated block-wise to particles, edges and triples respectively in cyclic fashion. There are three cost functions relating a particle to its eight neighbors through edges and triples in cloth structure. These tasks may use the same data and cannot be computed without some synchronization. This factor limits the parallel efficiency that must be resolved by keeping the overhead minimum. Therefore, we utilize the work-sharing directive of OpenMP to keep the all tasks in one parallel region separated by event synchronization and parallelism is exploited within a task. **ComputeGradient()** is parallelized as shown in Fig 5 and same idea is applied to other functions.

4.2 Load Balancing in OpenMP

Variation in mesh density by AMRC during execution, raises the necessity of dynamic load balancing. There are static, dynamic and guided scheduling available in OpenMP to balance the load. Static scheme simply divides the

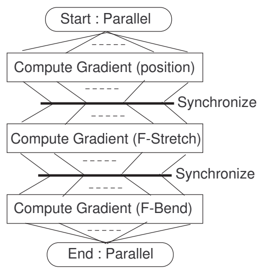
Start : Parallel
Compute Gradient (position)
Synchronize
Compute Gradient (F-Stretch)
Synchronize
Compute Gradient (F-Bend)
End : Parallel

Fig. 5 Parallelization of **ComputeGradient()**

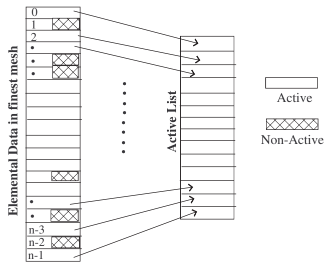

Fig. 6 Active List creation

work equally among processors. As distribution is decided at the start, it has the minimum overhead but faster processor has to wait for slower one at the end.

In dynamic scheduling, a fixed amount of job is allocated to a processor on first come first served basis. When a thread finishes its work, it goes to the system runtime manager of OpenMP and asks for a next job. The fastest thread never has to wait for the slowest thread to compute more than fixed amount of job. As this scheme allocates job to processors at runtime, it is more expansive than static scheme.

Guided schedule is same as dynamic schedule except that next job size is not fixed. It assigns larger job size to first processor and job size decreases exponentially until it reaches the defined minimum size. Size of next job is equal to the remaining work divided by the number of threads. Computing the job size at run time, increases its overhead than dynamic scheduling. Its efficiency depends on the combination of job size and number of processors.

4.3 Load Balancing Using Active Lists

We propose the distribution of data on the basis of active elements in the mesh. Elements are stored in data structure for finest mesh. Omitting an element during coarsening sets its status as non-active. Similarly status of an element is

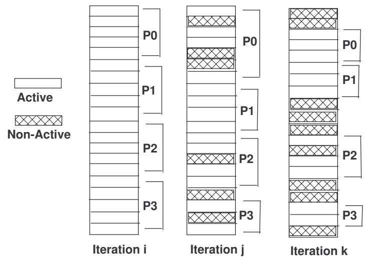

Fig. 7 Work distribution among processors

changed from non-active to active while refining. The lists of active elements are created in such a way that non-active elements are removed from the list after coarsening and active elements are inserted in the list after refining as shown in Fig. 6. There are three types of elements that demands to make three active lists. Therefore, three threads are employed to update the each list in sequential way. Then total number of elements in each list are divided among processors like static scheduling. The above procedure for maximum 20 elements is elaborated by an example in Fig. 7. Only active elements are redistributed among four processors at different iterations.

5. Experimental Results

Parallel algorithm based on OpenMP is designed to simulate the cloth with 1000 × 1000 particles, draping over a box using the Fujitsu PRIMEPOWER HPC2500 machine. The HPC2500 contains 96 CPUs distributed over 12 system boards that are connected with each other via a crossbar network supporting 133 GB/s. A system board has 8 SPARC64 V processors (1.3GHz) and 16*GB* of memory. All experiments are executed in batch mode to ensure the equal number of threads and processors. C-compiler **fcc** with *KOMP, Kfast_GP2=2, V9, Klargepage=2 and Khardbarrier* options is used to compile our source code. The performance evaluation in terms of computational time for 100 iterations and number of processors is expressed by the graph in Fig. 8.

Static scheduling is a simplest scheme with minimum parallel overhead. It has the poorest performance due to uneven distribution of load among processors as shown in Fig. 8.

Dynamic scheduling is employed for one, two and three blocks as minimum job size to see

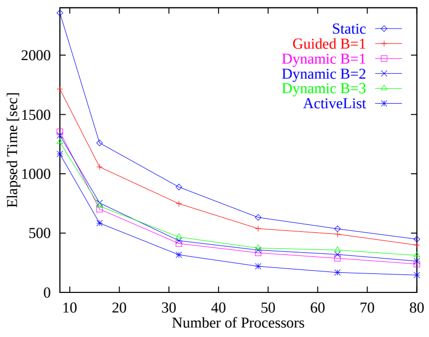

Fig. 8 Performance evaluation

Table 2 Timing analysis

No. of proce-ssors	Seq. over-head	Para. over-head	Time [sec]	Speed Up
1	11.5	0.000	10459.70	1.00
4	11.5	0.050	2498.97	4.18
8	11.5	0.080	1167.61	8.96
16	11.5	0.131	583.80	17.91
32	11.5	0.205	316.86	33.01
48	11.5	0.382	220.16	47.50
64	11.5	0.457	168.06	62.24
80	11.5	0.588	145.58	71.85

the effect of job size. The choice of job size is a compromise between quality of load balancing and synchronization cost. Load balancing improves for smaller size because fastest processor has to wait for less time but small size of job increases the number of job allocations at run time. So, synchronization cost increases because one synchronization per job allocation is required. Simulation results in Fig. 8 show that smaller job size is better for large number of processors while larger job size is better for less number of processors. Anyhow, difference in performance corresponding to the number of blocks is not prominent in our simulation.

Guided scheduling is done with minimum size equal to one block. At the end, job size becomes very small specially for large number of processors. Due to this reason, its performance lies between static and dynamic scheduling.

Using Active-Lists, each processor has same number of elements for processing. As a result, it has the best performance as compared with static, guided and dynamic scheduling schemes.

Elapsed time for cloth simulation by Active-Lists scheme is elaborated in the Table 2 for further analysis. It gives the best performance for 8 to 32 processors. Parallel overhead is very small as compared with total elapsed time up to 32 processors and each processor has large enough percentage of total work to compute, which matches with the size of cache memory. On the other hand, performance slows down with larger number of processors. The main reasons are sequential and parallel overheads, which become gradually obvious as the number of processors increases over 48.

Sequential overhead is the time required to compute the sequential part of the code that is approximately 10 seconds and time needed to update the active lists is 1.5 seconds. As there are three active lists, so always three threads are used to update these lists. Parallel overhead is the cost needed for thread creation, synchronization among threads and event synchronization. Table 2 shows that these costs increase with the number of processors.

If the cost of overheads, mentioned above, is deducted from the total elapsed time then result shows the linear speed up. For example, net time spent by 80 processors in parallel computation is $(145.58 - 11.5 - 0.588) = 133.4$ seconds. In case of single processor, parallel execution part spending $(10459.7 - 11.5) = 10448.2$ seconds might be executed in parallel. Ideally this time would be reduced to $(10448.2/80) = 130.6$ seconds if we employ the 80 processors. These two values 133.4 and 130.6 seconds are approximately the same. This fact means that Active-List scheme works well to achieve the best load balancing and the crossbar network of HPC2500 has not yet become a bottleneck.

6. Conclusion and Future Work

We have successfully implemented the parallel algorithm in OpenMP for simulating the realistic force as well as shape of cloth using AMRC. AMRC reduces the sequential time while a scheme based on the Active-Lists is implemented to balance the work load. Its comparison with the scheduling schemes available in OpenMP reflects a good speed up.

We have simulated the cloth containing $1000X1000$ particles that is large enough for a real size application. This algorithm may employ to complex applications like dressed human after improving the speed in future.

Scalability for a very large set of CPUs is a future consideration to get the real time simulation. Since number of CPUs are limited for OpenMP because of shared memory API, it requires switching to MPI or cluster-based systems. Such systems need mesh decomposition and boundary exchange. It demands the parallel code for AMRC and Active-Lists scheme. Later scheme can be decomposed by assigning a smaller private list to each distributed part of memory. When AMRC changes mesh density, data is migrated and individual lists are updated. Parallel AMRC and data movement are big limitation and need more attention.

References

1) D.E Breen, et al.: Predicting the drape of woven cloth using interacting particles; *Computer Graphics (SIGGRAPH Proceedings)*, pp. 365-372 (1994).
2) J.M. Bull: Measuring synchronization and scheduling overheads in OpenMP, *1st European Workshop on OpenMP*, Lund, Sweden (1999).
3) D. Hutchinson, M. Preston and W.T. Hewitt: Adaptive refinement for mass/ spring simulations, *7th Eurographics Workshop on Animation and Simulation*, pp. 31-45 (1996).
4) R. Lario, et al.: Rapid parallelization of a multilevel cloth simulator using OpenMP, *3rd European Workshop on OpenMP*, Spain (2001).
5) Mujahid, et al.: Modeling virtual cloth to display realistic shape and force based on physical data, *Journal of System, Control and Information, Japan*, Vol. 47(4), pp. 183-191 (2003).
6) Mujahid, et al.: Simulating realistic force and shape of virtual cloth with adaptive meshes and its parallel implementation in OpenMP, *IASTED Int'l Conf. on PDCN*, Austria, pp. 386-391 (2004).
7) H.N. Ng, R.I. Grimsdale and W.G. Allen: A system for modeling and visualization of cloth material, *Computer and Graphics*, Vol. 19(3), pp. 423-430 (1995).
8) S. Romero, et al.: Fast cloth simulation with parallel computers, *6th Int'l Euro-Par Conf. (Euro-Par2000), Germany*, pp. 491-499 (2000).
9) Y. Sakaguchi, et al.: Party: A numerical calculation method for a dynamically deformable cloth model, *System and Computers in Japan*, Vol. 26(8), pp. 75-87 (1995).
10) Jullian Villard and Houman Borouchaki: Adaptive meshing for cloth animation, *11th Int'l Meshing Roundtable*, pp. 243-252 (2002).
11) Vasily Volkov and Li Ling: Adaptive local refinement and simplification of cloth meshes, *1st Int'l Conference on Information Technology and Application, Australia* (2002).