Modeling Particle Systems Animations for Heterogeneous Clusters

Caroline Bellan Oliva, César A. F. De Rose
Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Porto Alegre, Brazil
carolineoliva@gmail.com, derose@inf.pucrs.br

Abstract

Parallel processing may help obtaining animations in shorter time or achieving better quality of the images generated through the simulation of particles systems, due to the parallel nature of particles. This work presents a model that aims to assist users on the use of clusters, especially heterogeneous ones, on the creation process of simulations with several particle systems. The model is destined to the animation of independent particles and provides a dynamic load balancing mechanism. The model is validated through the comparison of results (time taken to obtain the images) extracted from sequential and parallel executions of the library whose implementation was based on the model.

1. Introduction

Phenomena such as smoke, steam, fog, dust and wind are part of our daily visual perception and thus can significantly increase the realism of artificial renderings. As simple and ordinary as these phenomena may seem, as complex and difficult it is to simulate them [10].

Stochastic particle systems may be considered a general technique in the field of computer graphics, capable of creating a wide variety of effects and fuzzy objects such as water, smoke, grass, clouds, among others [14]. These effects happen in consequence of a series of physical aspects that may demand high computing power if applied all together.

Due to the growth in the availability of high speed networks and the parallel nature of the particles, it becomes possible to distribute the amount of data among several machines and achieve better performance of the applications [13]. The exploitation of parallel computing and characteristics of 64-bit architectures allow the reduction of the time necessary to obtain the images in particle systems animations. However, parallel programs are characterized by different concurrent activities, communication and synchronization between the processes executed on several machines, making the parallel programming a hard task [5].

In the recent past, clusters of off-the-shelf components gained importance in the field of computer graphics. As these machines become more available, the formulation of models to help users on the process of parallelizing their applications becomes interesting [12].

In this paper, we present a model for stochastic particle systems animations for heterogeneous clusters. The model deals with domain decomposition allowing the user to introduce efficient particle collision detection procedures and offers load balancing mechanism to compensate the different processing power of the cluster's machines.

The rest of the paper is organized as follows. The next section looks at related work. In Section 3 we describe our model. In Section 4 we present the validation of the model and some experimental results. In Section 5 we summarize and present some avenues of future research.

2. Related Work

The first parallel solution for particle systems simulation was conceived of by Karl Sims [13]. It was developed specifically for the Connection Machine CM-2. Each one of the processors receives a set of particles, independently of their localization in space. The application offers a small number of effects and the processes only communicate to decide the color of each pixel of the image.

In [16], the authors describe a process to perform cloth simulation on clusters using the parallel programming interface Athapascan. Their strategy splits the particles in particle blocks, which can be dynamically assigned to processors by Athapascan. The experiment was validated only on a multiprocessors machine so far, although the application was meant for clusters. The results show that the execution time depends both on the number of processors and on the size of the blocks.

Rodrigues *et al*. present a mechanism for parallel interactive graphical simulations of Molecular Dynamics [11].

Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)
1530-2075/05 $ 20.00 **IEEE**

It uses MPI (Message Passing Interface) for communication between the processes and the interaction with the user happens through the master process. The mechanism uses spatial decomposition of the particles and the processes communicate only with their immediate neighbors. The experiment was run on two different clusters and the results show the importance of high speed networks on these simulations, since the experiments achieved better performance in the slower cluster connected with gigabit Ethernet.

Henty in [7] presents a hybrid solution for the simulation of discrete elements such as sand grains that uses both message passing and shared memory. Again, the space is divided into blocks and to each processor it is assigned a set of blocks. The shared memory parallelization occurs lower down at the level of loops over particles within each block, so MPI communications never take place within a parallel region. Global quantities such as the energy are reduced in parallel within a block through the shared memory, summed over blocks by the MPI process and then accumulated over processes by an MPI collective call. The results show that the pure MPI code is always more efficient.

Particle systems are also used in Monte Carlo simulations. In [2], the authors present the parallelization of the package PENELOPE used in medical applications. The new package uses MPI for communication among the several processes and the results show a speed-up of 240.25 for 256 processors.

In [6] and [3], the authors present dynamic load balancing techniques for applications where it is necessary to preserve the locality of the objects (*i.e.* neighbors should remain close to each other). The first one divides the data along the molecular chain, assigning work proportional to the processing power of the processor responsible for the subspace. The experiments on a Cray T3D resulted in a speed-up of 1.38 in comparison to the unbalanced application. The second one breaks the space in blocks in one or more dimensions, in equal sizes. This kind of division leads to unbalanced data among processors. In order to equilibrate the load, particles are exchanged between neighbors during the simulation. When one particle leaves or enters a process, it is necessary to redefine the block's dimensions of both processes. The major problem of this solution lies in the restriction of acceptable dimensions when the space is broken in more than one dimension, leading to unbalanced states. The results of the experiments on an IBM SP2 presented speed-up between 1.6 and 4.4.

In [9], McAllister presents the Particle System API, a library for the simulation of dynamic particle systems. The API has been implemented both on the parallel geometry processors of PixelFlow and on UNIX and Win32 systems. In the PixelFlow implementation, the API assigns a set of particles to each one of the processors and the rendering is done directly by the processors. The UNIX/Win32 implementation works on both uniprocessor and multiprocessor shared memory systems through the use of threads.

The solutions above show a lack of generic applications that can take advantage of the processing power offered by clusters, whether they are homogenous or heterogeneous. This deficiency led to the development of a parallel model for the simulation of independent particles on clusters. The new model uses massage passing for communication between processes and load dynamic balancing mechanisms based on some of the solutions above. The model is described in the next section.

3. The Model

The developed model is meant for the simulation of stochastic particle systems where one particle is totally independent of the others. The only foreseen interaction among particles is the possibility of collision detection between them.

The model was built using the parallel phases programming paradigm. According to this paradigm, an application consists of a certain number of steps, each step divided in two phases: one computing phase, when the processes process their local data independently, followed by an interaction phase, when the processes execute one or more synchronization operations such as barriers or blocking operations [15].

To maintain the work equilibrium among the processes during the simulation, the model uses local dynamic load balancing with a centralized manager, since one process can exchange particles with its neighbors during the balancing procedure. This restriction is due to the need of data locality preservation, because in case the user decides to include collision detection methods it is better to keep the particles close to their neighbors to reduce communication between processes.

The following items describe each one of the components of the model and the procedure used to simulating the particle systems.

3.1. Components of the Model

The model is composed by processes, particles, particle systems, domains and actions over particles. Each one of these components is described in the following items.

3.1.1. Processes The model has three kinds of processes, the calculators, the image generator and the manager. The manager is responsible for creating the particles and managing the load balance between the calculators. The image generator collects the particles sent by the calculators and renders each one of the frames of the animation. The calculators are responsible for applying the actions over the particles, moving them and detecting collision (according to

the procedure chosen by the user). All the processes in the model know the global information needed for the simulation. The particles are the only exclusive set of data of the process; however, all particle systems exist in all processes.

3.1.2. Particles There are four basic particle properties that are always needed, independently of the kind of animation we want to create. The properties are:

- Position in the space or on the plane (x, y or x, y, z);
- Orientation (x, y or x, y, z);
- Age;
- Velocity.

It is important to highlight that the model does not require each particle of a system to have a unique identifier, as long as particles of different systems are stored in different data structures.

3.1.3. Particle Systems The particle systems have the same properties of their particles except for the age. These properties do not change the state of the system; they are used to determine the initial values for the particle's properties. According to the structures used to store the systems, it may be necessary to give an identifier to each one of the systems to guarantee that the particles exchanged between processes remain in the original system. In case the systems are stored in a vector, the position in the vector can be used as the identifier, because the creation of the systems happens in the same order for all processes.

3.1.4. Domains Given the possibility of collision detection among particles and aiming at the reduction of amount of tests, the simulated space is divided into domains, along one of the axis of the plane or space. Each particle system has its own domains, *i.e.*, each system is divided in n slices, with n the number of calculator processes. Figure 1 shows a possible division of the space in four domains (four calculator processes, P1, P2, P3 and P4).

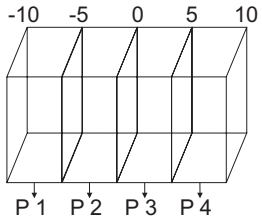

Figure 1. Example of domains, initially with the same size.

As seen in Figure 1, each domain is associated to one calculator process, sequentially, *i.e.*, the first process is responsible for the first domain, the second for the second and so on. This process is repeated for each one of the particle systems.

All the processes know the dimensions of all the domains, so they know which process to send a particle to when it needs to detect collision. If the space was not divided into domains, it would be necessary to test collision with all the particles of all the processes, because there would be no guarantee that close particles would remain in the same process during the simulation. Another reason for global knowledge of the domains is the fact that particles may change domains during simulation, allowing the particle to be sent only to its new process instead of broadcasting the particles to all processes.

The dimensions of the domains may be altered during simulation, due to the load balance between processes. Since it may be necessary to move particles from one process to another, it becomes necessary to change the domain's dimension, because a process can only hold the particles that belong to its domain.

3.1.5. Actions over Particles The model stipulates rules of behavior only for actions that create and move particles, since these actions change the spatial distribution of the particles. The actions that modify the properties of the particles without changing its positioning may be applied on the particles at any time during the simulation, because these changes do not need to be informed to the other processes.

All the particles are created by the same process and sent to the others according to their domains, as described before.

At the end of each frame of the animation, it is necessary to verify if the particles associated with a process remain within the process' domain, otherwise they must be sent to their new process.

Depending on the collision detection mechanisms chosen by the user, the particles that change domains may be exchanged between processes during the computation and validation of their new position. In this case, it is not necessary to exchange particles between processes at the end of each frame, because all the particles are already within the correct domain.

3.2. Simulation Procedure of One Particle System

The model allows the simulation of several particle systems simultaneously. For each one of the systems, the model keeps information about domains, amount of particles and time taken to process all the actions over the particles. Figure 2 shows the simulation procedure for one particle system.

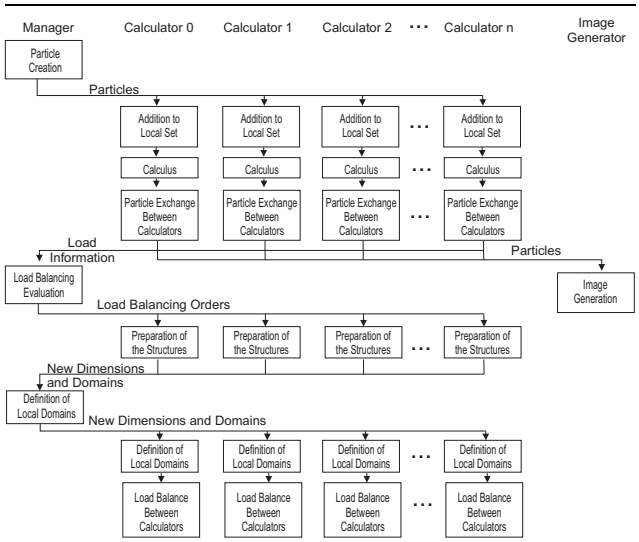

Figure 2. Simulation of one particle system.

The processes are launched and the particle system is created. During its creation, the space is divided in *n* domains, as described before. After the creation, the processing loop starts. Algorithm 1 exemplifies this loop.

Algorithm 1 Simulation of one particle system.

```
Do {
    Configure particle system
    Create n particles
    Simulate gravity over the particles
    Remove particles under the position (x, y, z)
    Simulate collision with object obj
    Move particles
    Generate the image
    frames = frames + 1
} While frames < maximum_amount
```

The external framework is the same for all the processes in order to facilitate the development of the animation, but the action is treated differently by the processes. The system configuration, that does not need to be necessarily inside of the loop, is an example of action that is not processed by the image generator.

If the user had chosen not to use the load balancing mechanism, it would have been necessary to include a synchronization step between the processes. When the balancing is activated, the synchronization happens through the exchange of information about domains. If the mechanism is deactivated, the image could be incomplete, because there would be no guarantee that the image generator would receive particles from all the processes. If one of the processes was faster, it could send the particles twice to the generator, ending the frame before the slower process started sending particles.

3.2.1. Actions that Create Particles During the particle creation step, the manager generates the amount of particles determined in the action, positioning the particles according to the behavior stipulated for the particle system. After the creation of each particle, it is stored in the structure corresponding to its domain. By the end of the creation action, the manager sends the particles to the processes, according to their domains, informing the end of transmission to each one of the calculator processes.

The calculators wait for the particles to start processing the following actions. It is mandatory that these processes receive notification about the end of transmission; otherwise they will remain blocked inside the creation action and will not allow the processing of the next frame of the animation.

3.2.2. Actions that Change Properties of Particles Actions that simulate gravity, eliminate or bounce particles that collided with external objects do not change the positioning of the particles. These actions can take place at any moment of the simulation, since there is no need of interaction between processes.

3.2.3. Actions that Change Positioning of Particles During the actions that alter the positioning of the particles, there is no need of communication between the processes. However, when moving a particle, the process must verify whether the particle left its domain. When it happens, the process must store the particle in a different structure for future exchange of particles with other processes.

3.2.4. Actions that Generate the Animation Frame During the execution of this action, processes exchange particles due to domains changes, balance their load and generate the frame of the animation.

Initially, particles which changed domains are exchanged between processes, so that from this point on the processes only manage their own particles. Once again, it is important to all the processes to be notified about the end of transmission, otherwise they will remain blocked waiting for particles.

After finishing the exchange of particles, the calculators must inform the manager about their load (amount of particles under control of the process) and time taken to process the particles.

The time is measured from the beginning of the first action to the end of the last one, but since it is not possible to determine which one is the last action, the time is updated in all of the actions. After retrieving the amount of time, the process must recalculate it, since the amount of particles of the process changed due to the exchange of particles between processes. The new time must be proportional to the new amount of particles held by the process.

While the manager evaluates the load balancing, the calculators send the particles to the image generator. Upon particles reception, the image generator applies the necessary actions to obtain the image. It is also its responsibility to render external objects that exist in the simulation.

3.2.5. Load Balancing Evaluation After receiving the information about the load of each calculator process, the manager evaluates the load balancing for each pair of neighbors, *i.e.*, the manager evaluate the equilibrium of load between x and x+1, x+1 and x+2, x+2 and x+3, and so on. For some frame of the animation, with the need to balance the load between the pair x and x+1, the pair x+1 and x+2 will not be evaluated. The next pair to be evaluated will be x+2 and x+3. This happens due to the following rules:

- The load balancing only takes place between neighbors due to the domains;
- Each process only sends or receives particles, but never both operations to avoid alignment of processes;
- The load balancing happens only between two processes, *i.e.*, the process x, for example, cannot receive particles from both processes x-1 and x+1, once again to avoid alignment.

To avoid performing balancing always between the same pair of processes, at every execution of the load balancing evaluation, the manager alternate the identifier of the first process to be evaluated (1 or 2).

For each pair, if the difference between their processing times is bigger than a certain value, the manager will redistribute their particles. The new load will be proportional to the processing power of the processes. Depending on the amount of particles to be moved from one process to another, it may not be interesting to perform the transmission of the particles.

After finalizing the evaluation, the manager sends the load balancing orders to the calculators, informing the amount of particles to be moved, the neighbor with whom the balancing will take place and the operation to be performed by the calculator (sending or receiving of particles).

Upon reception of the orders, the process responsible for sending particles during the load balancing must select the particles to be donated. The process cannot simply remove the specified amount of particles because the new redistribution must keep obeying the domains. The particles must be ordered in accordance to the axis chosen for the division of the domains. If the particles are to be send to the left process, the particles with lower x values are the ones to be donated; otherwise the particles with higher x values are the ones to be transferred. Based on the ordering and selection of the particles, it is possible to define the new dimensions of the domains.

After the definition of the new dimensions, the calculator processes send the new values to the manager, which will update its local information and send the dimensions back to all the calculators. Only after receiving the new domains the calculators effectively start the donation and reception of particles.

3.3. Simulation Procedure of Two or More Particle Systems

The simulation process of two or more particle systems follows the same steps previously presented, when analyzing the systems individually. The difference exists when the simulation is seen as a whole, because there are different ways to combine the processing of more than one system. Depending on the form used, the processing may be more or less efficient.

4. Validation of the Model

The model was validated using David McAllister's [9] library as base for the development of a new parallel library, because it offers a wide variety of effects and can process more than one particle system.

The original library was completely rewritten to make it easier to include the new functions of the model. Still, the major modifications to the original code are related to the storage structures and inclusion of the communication operations.

All the particle systems are created by all process in the same order. It allows us to use the position in the particle systems vector as the identifier of the system. This identifier is used to define the current particle system and is also necessary for particle exchanges between processes.

Instead of storing all the particles of a system within a domain (region of the space that belongs to the process) in the same vector, we now break the domain in sub domains and store each one in a separate vector. We chose this approach to accelerate the load balancing process and particle exchanges between processes. If we kept storing all the particles in the same vector, it would be necessary to compare the particles' position to the domain's edges by the end of each frame in order to discover which ones should be sent to other processes. During the load balancing procedure, the division of the domains reduces the amount of particles that must be ordered because the process must discover the new dimensions of the domain, since data locality must be preserved.

The modifications were applied accordingly to the model. We used the sequential execution time as the comparison measure of processing power of the different machines of the cluster to perform load balance.

5. Experimental Results

The parallel version of the library was tested on a heterogeneous cluster composed of the following machines:

- 8 HP NetServer E60 (Dual Pentium III 550 MHz, 256 Mbytes of RAM) nodes (type *A* node);
- 8 HP NetServer E800 (Dual Pentium III 1 GHz, 256 Mbytes of RAM) nodes (type *B* node);
- 2 HP Workstation zx2000 (Itanium II 900 MHz, 1 Gbyte of RAM) nodes (type *C* node).

The Pentium III nodes are connected by two networks, Myrinet [1] and Fast-Ethernet, while the Itanium nodes are only connected by Fast-Ethernet.

We developed two experiments, both with the same amount of particle systems and particles per system but with different behavior. The first one simulates snow while the second simulates water fountains. The results from each one of the experiments are presented in the following items. By the end of this section we present some comparisons between the results.

5.1. Snow Simulation

For each frame of this simulation, we create new particles, apply a random acceleration on the particles, simulate collision, eliminate old particles and finally move the particles through the space. The particles tend to remain in their original domain since their movement is mainly vertical.

We simulated eight particles systems with 400.000 particles in each one of them. We chose to use eight particle systems because we simulate eight fountains in the other experiment, so we can have an equal comparison of the experiments.

During the simulation, at the end of each frame, each process has approximately 560 particles that belong to another calculator. For the total amount of process, it means 613 Kbytes of data to be exchanged.

Table 1 shows some of the results obtained through the execution of this experiment using Myrinet and GNU/GCC Compiler on eight E800 nodes. The speed-up is calculated using the time of the sequential execution on the same type of node using the same compiler, since the E800 nodes presented the best performance for this compiler.

When using infinite space (IS) together with static load balancing (SLB), depending on the size of the simulated space only a few processors might actually be given work. This factor can be seen in the results presented in the second column of Table 1. When we use odd numbers of processors, only the process responsible for the central domain receives particles during the simulation. If we use finite space, we increase the chances of load distribution among processes.

Nodes vs. Processes	Speed-Up			
	IS-SLB	FS-SLB	IS-DLB	FS-DLB
4*B /4 P.	1.74	1.74	1.73	1.75
5*B /5 P.	0.82	2.49	2.9	2.5
6*B /6 P.	1.74	3.12	2.99	3.11
7*B /7 P.	0.92	3.63	3.15	3.65
8*B /8 P.	1.74	4.14	3.37	4.14
8*B /16 P.	1.73	6.47	3.75	6.37

Table 1. Snow Simulation using Myrinet and GNU/GCC Compiler.

When using static load balancing with restriction of the simulated space (FS, *i.e* finite space), the experiment presented speed-up of 6.47 for the same machines as above. The use of sixteen processes allowed better performance because each E800 node has two processors. This speed-up is better than the one obtained through the use of dynamic load balancing (DLB) with finite space, because of the extra communication costs, since the distribution of the particles through the space is homogenous and there is no need to balance the load between neighbors.

However, we cannot always restrict the size of the simulated space to fit exactly the portion that we are using, making the use of dynamic load balancing necessary. This factor can be viewed by the increase in performance when comparing the results from the experiments using infinite simulated space.

In our tests, the use of eight E60 nodes (sixteen processes) was only justified when the amount of E800 nodes was lower than seven. Testes executed using four nodes of each type resulted in speed-up of 2.76 and 2.93, for eight and sixteen processes respectively.

The executions using Fast-Ethernet and ICC Intel Compiler on eight E800 nodes (sixteen processes) presented speed-up of 2.56 in comparison to the sequential execution on the Itanium machine using the same compiler.

Once again, when using static load balancing with restriction of the simulated space, the experiment presented speed-up of 2.65 for the same machines and processes as above.

Table 2 shows the results of heterogeneous experiments using dynamic load balancing and finite space. The speed-up for the heterogeneous environment is calculated using the time of the sequential execution on the Itanium processor together with the ICC Intel Compiler, since this combination presented the best performance.

The highest speed-up, 3.15, was achieved when using dynamic load balancing with restriction of the simulated space on two E800 nodes together with two Itanium nodes.

Nodes vs. Processes	Speed-Up
4*B (4 P.) + 4*A (4 P.) = 8 P.	1.36
4*B (8 P.) + 4*A (8 P.) = 16 P.	1.5
8*B (8 P.) + 8*A (8 P.) = 16 P.	2.4
8*B(16 P.) + 8*A (16 P.) = 32 P.	2.02
2*B (2 P.) + 2*C (2 P.) = 4 P.	2.67
2*B (4 P.) + 2*C (2 P.) = 6 P.	3.15
4*B (4 P.) + 2*C (2 P.) = 6 P.	2.84
4*B (8 P.) + 2*C (2 P.) = 10 P.	2.61

Table 2. Snow Simulation using Fast-Ethernet and ICC Intel Compiler.

5.2. Fountain Simulation

For each frame of this simulation, we create new particles, apply gravity and acceleration on the particles, simulate collision, eliminate old particles and finally move the particles through the space. Differently to the previous experiment, the particles tend to change domains during the simulation since their movement is both horizontal and vertical.

Once again, we simulated eight particles systems with 400.000 particles in each of them. The particle systems were distributed through the simulated space, so it becomes harder to restrict the space.

At the end of each frame, each process has approximately 4000 particles that belong to another calculator. For the total amount of process, it means 4375 Kbytes of data to be exchanged.

Table 3 shows some of the results obtained through the execution of this experiment using Myrinet and GNU/GCC Compiler on eight E800 nodes. The speed-up is calculated using the time of the sequential execution on the same type of node using the same compiler.

Nodes vs. Processes	Speed-Up			
	IS-SLB	FS-SLB	IS-DLB	FS-DLB
4*B /4 P.	0.98	1.09	1.49	1.49
5*B /5 P.	0.92	1.19	1.76	1.76
6*B /6 P.	0.98	1.31	2.02	2.05
7*B /7 P.	0.92	1.54	2.34	2.36
8*B /8 P.	0.98	1.86	2.66	2.67
8*B /16 P.	0.98	2.66	3.74	3.82

Table 3. Fountain Simulation using Myrinet and GNU/GCC Compiler.

The use of dynamic load balancing in this kind of simulation always resulted in better speed-up when comparing to the static load balancing strategy. This is due to the non uniform distribution of particles through the space.

Opposing to the previous experiment, the use of 16 nodes (eight E800 and eight E60) resulted in a speed-up of 4.28. The increase in availability of processing power compensates the loss of performance in function of the increase of communication between the processes.

The executions using Fast-Ethernet and C Intel Compiler did not result in gain of performance. The highest speed-up, 1.26, was achieved when using dynamic load balancing with restriction of the simulated space on two E800 nodes together with two Itanium nodes.

5.3. Comparison between the Simulations

The results from both simulations lead to the following considerations:

- When the load is uniform and the user is capable of restricting the simulated space, the use of dynamic load balancing mechanism is only necessary in heterogeneous clusters. In homogeneous clusters, if the simulated space has the same dimensions of the particle system, the equilibrium of the load happens naturally, because each process will be given a domain with the approximate same amount of particles of its neighbors;
- When the load is irregular, the use of dynamic load balancing becomes necessary also in homogeneous clusters, because there is no guarantee that the particles given to a process will remain inside of its original domain during the simulation. However, in order to gain performance, it is necessary to use high speed networks or machines which processing power can compensate the loss by the intense communication among the processes.

The use of dynamic load balancing and Fast-Ethernet was not satisfactory, possibly due to the amount of data exchanged between processes at the end of each frame. For the rest of the simulations, the gain of performance justifies the use of clusters. The time to simulate snow with Myrinet was reduced by 84% and with Fast-Ethernet by 68%. The second simulation's time was reduced by 66% when using Myrinet.

6. Conclusions and Future Work

This work presented a model to generate particle systems animations on heterogeneous clusters, although it may also be used on homogeneous ones without modifications. The model allows the user to include collision detection mechanisms since it preserves data locality. Due to the spatial

domain decomposition, the model uses dynamic load balancing mechanisms that maintain the particles close to their neighbors.

The experiments with the library whose implementation was based on the model allow a gain of performance even for applications with irregular data distribution, as long as high speed networks or machines whose processing power can compensate the costs of communication are available.

The performance of the Itanium nodes was not satisfactory, but it is important to notice that the workstations used in the simulations are no longer manufactured. We hope that new machines, such as HP Integrity rx2600 (Dual Itanium II 1.5 GHz) allow more satisfactory gain of performance.

As future work, we intend to use remote image generation mechanisms such as WireGL [8] or Pomegranate [4], to include ways of interconnecting particles to allow the simulation of fabric, for example and to decentralize the load balancing management.

Acknowledgments

This work was done in collaboration with HP-Brazil.

References

[1] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: a gigabit-per-second local-area network. *IEEE Micro*, 15(1):29–36, February 1995.

[2] R. B. Cruise, R. W. Sheppard, and V. P. Moskvin. Parallelization of the penelope monte carlo particle transport simulation package. *Nuclear Mathematical and Computational Sciences: A Century in Review, A Century in Anew*, 2003. [CDROM]. 11p.

[3] E. Deelman and B. K. Szymanski. Dynamic load balancing in parallel discrete event simulation for spatially explicit problems. *Proceedings of the 20th Workshop on Parallel and Distributed Simulation*, pages 46–53, 1998.

[4] M. Eldridge, H. Igehy, and P. Hanraham. Pomegranate: A fully scalable graphics architecture. *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 443–454, 2000.

[5] I. Foster. Compositional parallel programming language. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 18(4):454–476, July 1996.

[6] D. F. Hegarty and M. T. Kechadi. Topology preserving dynamic load balancing for parallel molecular simulations. *Proceedings of the 1997 ACM/IEEE conference on Supercomputing*, 1997. [CDROM]. 20p.

[7] D. S. Henty. Performance of hybrid message-passing and shared-memory parallelism for discrete element modeling. *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, 2000. [CDROM]. 10p.

[8] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanraham. Wiregl: A scalable graphics system for clusters. *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 129–140, 2001.

[9] D. K. Mcallister. The design of and api for particle systems. *University of North Carolina, Chapel Hill, NC, Tech Rep. UNC-CH TR 00-007*, 2000, 8p.

[10] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. *Proceedings of the 2003 ACM SIGGRAPH Eurographics Symposium on Computer Animation*, pages 154–159, 2003.

[11] E. R. Rodrigues, A. J. Preto, and S. Stephany. A parallel engine for graphical interactive molecular dynamics simulations. *16th Symposium on Computer Architecture and High Performance Computing*, pages 150–157, 2004.

[12] J. P. Schulze and U. Lang. The parallelization of the perspective shear-warp volume rendering algorithm. *4th Eurographics Workshop on Parallel Graphics and Visualization*, pages 61–69, 2002.

[13] K. Sims. Particle animation and rendering using data parallel computations. *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, pages 405–413, 1990.

[14] D. Tonnessen. Particles systems for artistic expression. *4th Annual Subtle Technologies Conference*, pages 17–20, 2001.

[15] B. Wilkinson and M. Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice-Hall, 1999.

[16] F. Zara, F. Faure, and J.-M. Vincent. Physical cloth simulation on a pc cluster. *4th Eurographics Workshop on Parallel Graphics and Visualization*, pages 105–112, 2002.