ELSEVIER

Comput. Methods Appl. Mech. Engrg. 145 (1997) 301–321

Computer methods in applied mechanics and engineering

3D simulation of fluid–particle interactions with the number of particles reaching 100

A.A. Johnson and T.E. Tezduyar*

Aerospace Engineering and Mechanics, Army HPC Research Center, University of Minnesota, 1100 Washington Avenue South, Minneapolis, MN 55415, USA

Received 4 June 1996

Abstract

A high performance computing research tool has been developed for 3D simulation of fluid–particle interactions with the number of particles reaching 100. The tool is based on a stabilized space–time finite element formulation for moving boundaries and interfaces and parallel computing. Other components of this tool include: fast automatic mesh generation with structured layers of elements around the particles and unstructured meshes elsewhere; an automatic mesh moving method combined with remeshing as needed; accurate and efficient projection of the solution between the old and new meshes after each remesh; surface mesh refinement as two spheres or a sphere and the tube wall get close; and multi-platform computing.

We apply this tool to the simulation of two cases involving 101 spheres falling in a liquid-filled tube. In both cases the initial distribution of the spheres in the tube is random. In the first simulation the size of the spheres is also random, whereas in the second case it is uniform. We demonstrate that the tool developed can be used for simulation of this class of problems with computing durations kept at acceptable levels.

1. Introduction

A 3D flow simulation capability for studying fluid–particle interactions and the application of this capability to several cases involving multiple spheres falling in a liquid-filled tube were reported earlier in [1–3]. The methods used for developing a parallel computational capability to simulate this class of problems were described in detail in [4]. These methods include: a stabilized space–time formulation for moving boundaries and interfaces; parallel implementation of this formulation; an automatic mesh generation method which produces structured layers of elements around the particles and unstructured meshes elsewhere in the domain; a mesh update method based on an automatic mesh moving method (with the structured layers of elements 'glued' to the particles) combined with remeshing as needed; and accurate projection of the solution from the old mesh to the new one after each remesh.

A detailed study of four cases of multiple spheres falling in a liquid-filled tube, with the number of spheres ranging from two to five, were reported in [5]. The methods employed for these simulations served their purpose well in being able to handle up to five spheres. It is our belief that these methods also have the capacity to handle cases with the number of spheres being around 10, and with computing durations still remaining within acceptable levels. Following the development and demonstration of computational tools satisfactory for simulations with this many spheres, we targeted the development of the methods required to carry out such simulations with the number of spheres reaching 100.

* Corresponding author.

PII S0045-7825(96)01223-6

Such a simulation capability can be used to help understand the behavior of fluid–particle mixtures which are used in many industrial applications. Examples are applications in materials processing such as drying, particle coating and controlling chemical reactions such as coal burning, and also in materials transport. See [6] for a discussion of industrial applications with flows involving fluid–particle mixtures.

In this paper we describe the methods we needed to develop to increase our fluid–particle simulation capacity by one order of magnitude in terms of the number of spheres. Our starting point was the methods developed earlier for a much fewer number of spheres. The new methods were developed by improving and optimizing the earlier methods, and for some of the methods by having a new way of looking at things. The main objectives in this re-development process were to have sufficient accuracy and robustness and to keep the computing durations to acceptable levels.

For simulations carried out earlier for 2–5 spheres, the size of the meshes ranged from 24 000 nodes to 55 000 nodes. With our automatic mesh moving scheme capable of reducing the frequency of remeshing, we were able to continue the computations with the average frequency of remeshing during a simulation ranging from roughly every 75 time steps to every 500 steps. Because of this, the computing time associated with resmeshing (computing time for automatic mesh generation, projecting the solution, and partitioning the mesh to parallel processors) during a simulation was not significant compared to the total computing time for the entire simulation. Consequently, reducing the computing time for the automatic mesh generation was not a major concern to us. The objective was more to have a functional, pilot version of a simulation tool for fluid–particle interactions. When the number of spheres reaches 100, however, the size of the meshes needed increases significantly. Furthermore, the frequency of remeshing also increases, rather dramatically, due to the dramatic increase in the frequency of collisions and spheres getting too close to each other. This motivated us to redesign our automatic mesh generation strategy to substantially reduce the memory and computing time demands of the mesh generation process.

We also made other modifications to the automatic mesh generation method. A new way of generating structured layers of elements around the spheres was introduced. In the first stage of this process, prismatic elements (triangular base) are generated in these structured layers. Using prismatic elements would result in better accuracy, especially as the height of these elements becomes much smaller compared to the dimensions of the base. This would be particularly desirable when very large aspect ratios become difficult to avoid. For the Reynolds number ranges we are dealing with, this does not become an issue [7], and therefore, for simplicity in the parallel implementation, we subdivide these prismatic elements into tetrahedra to have an all-tetrahedra mesh. Another change in the mesh generation process to increase the accuracy and efficiency of the computations was made in the way the surface meshes are refined as two spheres get close to each other or as a sphere gets close to the tube wall.

Other modifications and additions to the methods that were used in our previous fluid–particle interaction simulations include the following:

- To increase the scope and accuracy of the computations, a new collision model, which also accommodates multiple-sphere collisions, was introduced.
- To decrease the memory requirements of the computations in the iterative solution of the coupled, nonlinear equations, the strategy to compute the effect of the global matrix–vector multiplications was changed from element matrix-based to element vector-based (aka matrix-free [8]).
- To speed-up the computations, the algorithm used for projecting the solution from the old mesh to the new one after a remesh was changed to decrease substantially the computing time for the projection.
- To make the simulations more free from user interference, a more autonomous simulation strategy based on multi-platform computing is introduced. In this strategy, while the mesh partitioning, flow computations, and mesh movements are performed on the Thinking Machines CM-5, the mesh generation and projection is accomplished on a Silicon Graphics multi-processor (SGI-MP) system. The communication between the two platforms is sustained via a High Performance Parallel Interface (HiPPI) channel.

In Section 2, we describe these methods in more detail. Our simulations for 101 spheres falling in a

liquid-filled tube are reported in Section 3. The initial distribution of these spheres in the tube is random. In the first of the two simulations the size of the spheres is also random, while in the second simulation the sphere size is uniform. In Section 4 we present our concluding remarks.

2. Methods and implementation

In this section, we describe the improvements to the methods and computing strategies developed earlier for much fewer particles and the new methods and strategies developed for the class of simulations we are targeting here.

2.1. Mesh generation

A new version of the automatic mesh generator described in [4,7] is used here. The algorithm is based on Delaunay methods [9,10] and uses edge-swapping techniques [11,12] for the nodal insertion process. This new version was necessary because for this class of simulations we are faced with the need for repetitive generation (due to remeshing) of very large meshes with the number of elements of the order 1 million. The previous, pilot version suffered from both large memory requirements and exponential scaling of computational time with mesh size [7].

The problem with computing time was due to the fact that when a new node was inserted into the interior of the mesh, the location of that node was determined by searching all elements in the mesh for the one with the worst quality (largest circumcircle). As the mesh gets larger, this search becomes slower and slower. This problem was resolved by replacing the element search with a heap list [13]. In this heap list, the worst element is always stored at the top of the list, and any changes to this list (addition or removal of an element) can be effected in order $\log_2(N)$ operations on average.

This change, and other improvements in algorithmic efficiency and better memory management, resulted in a linear relationship between computational time, mesh size, and required memory. In this class of problems, a finite element mesh with around 1.2 million tetrahedral elements can now be generated on a single (150 MHz R4400) processor of an SGI workstation in 6 minutes with 90 MBytes of memory.

We also modified the mesh generation procedure to vary the surface mesh refinement for the spheres as a function of the distance between the spheres. If any two spheres get close to each other, the refinement increases so that there is always roughly three layers of elements between the spheres (plus the structured layers of elements created around each sphere to better model the boundary layer features of the flow). For example, Fig. 1 shows the surface refinement for a sphere when it is alone and when another sphere (not shown) is very close. Fig. 2 shows a slice of a mesh at a cross-section cutting two spheres that are close together. We also increase the refinement on the tube wall if the distance between a sphere and the tube wall becomes small. These increases in refinement are implemented to

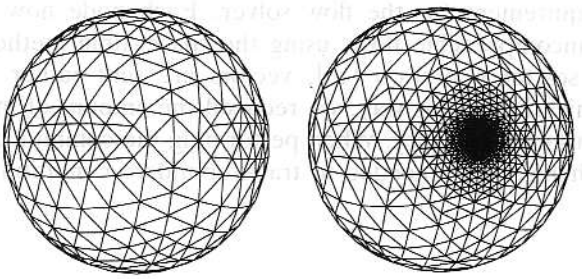

Fig. 1. Surface refinement for a sphere when it is alone (left) and when another sphere (not shown) is very close (right).

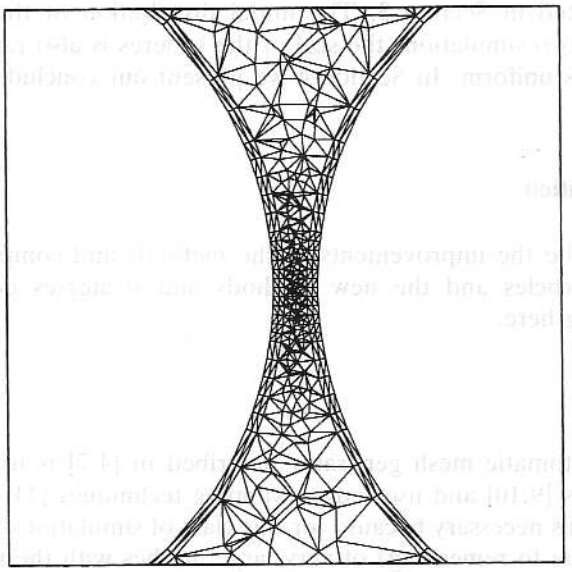

Fig. 2. Slice of a mesh at a cross-section cutting two spheres that are close together.

improve solution accuracy and to help in reducing the mesh distortion due to the movement of the spheres.

The final addition to this mesh generation program is an improvement to the way the method creates structured layers of elements around each sphere. In this new version, the layers are created algebraically by 'extruding' out the surface mesh of each sphere to construct prismatic elements (triangular base). These elements are then divided into tetrahedral elements, and proper care is given to make sure that element edges match across the boundaries of the prismatic elements.

2.2. Flow solver

The flow solver we use for these simulations is implemented on the CM-5 using the data-parallel programming model and is described in sufficient detail in [8,14]. In the iterative solution of the coupled, nonlinear equations, an element vector-based (matrix-free) strategy is used to compute the effect of the global matrix–vector multiplications. This significantly reduces the memory requirements of these large-scale simulations. In this strategy, no global matrix is formed, but we still solve a coupled system of equations using only residual vectors.

Unlike the earlier version of our parallel implementation, the communication traces are set up between element-level quantities and global nodal-level quantities rather than global equation-level quantities. The amount of memory needed to save these communication traces is rather large on the CM-5 using the Thinking Machine's Scientific Software Library (CMSSL) [15], and in almost all cases is the dominant memory requirement for the flow solver. Each node now represents *ndf* degrees of freedom ($ndf = 8$ for 3D incompressible flows using the space–time method), and when data is sent using either a gather or scatter operation [16], vectors are sent rather than scalar quantities. By implementing the communication in this way, we reduced the amount of memory needed to store the communication traces by a factor of $1/ndf$. When performing the communication steps in this manner, other minor algorithmic changes were needed to track boundary condition information properly.

2.3. Mesh movement

A combination of three mesh update methods are used to handle the motion of the mesh as the

spheres move around. These were described previously in [5,17] and are a global mesh translation, an automatic mesh moving scheme, and remeshing.

The entire mesh translates with the center of gravity of the system of particles. This way the spheres can fall for as long as we want them to. On top of this global translation, we use an automatic mesh moving scheme where the motion of the nodes is determined by solving the modified equations of linear elasticity. This requires the solution of a set of finite element equations for the unknown components of the velocity of the nodes governed by the elasticity equations. Since we have already set up the communication traces between the element-level and nodal-level quantities for the flow solver, we use the same communication traces for the mesh movement formulation, and this gives us more savings in memory over our earlier version. When the mesh becomes too distorted, we remesh by generating an entirely new set of nodes and elements and projecting the solution from the old mesh to the new one. The computations then proceed using an undeformed mesh.

Another feature we added to our methods for the simulations presented here is the capability for automatic exit from the flow solver. During the simulation, when the mesh distortion measures based on the volume and aspect ratio changes become too high [5], the flow solver terminates itself and generates restart data. After the flow solver/mesh motion code finishes its tasks, the remeshing process starts.

2.4. Collision model

In the four cases of fluid–particle simulations presented in [5], only one involved collision between the spheres, and that was only one collision during the entire simulation. In the fluid–particle simulations presented here, on the other hand, on the average, 1–2 collisions take place at every time step. We further improved our collision model over what was presented in [5] to better model this frequent occurrence.

Fig. 3 explains the notation used in the collision equation. In Fig. 3, V_A and V_B are the velocities of Spheres A and B, and n_A is a unit normal vector pointing from the center of Sphere A to the center of Sphere B, and $n_B = -n_A$.

The equations governing the motion of Spheres A and B due to the gravitational and fluid dynamical forces are

$$V_A^{n+1} = V_A^n + \Delta t\left[\frac{(F_A^{n+1} + F_A^n)}{2m_A} + g\right], \tag{1}$$

$$V_B^{n+1} = V_B^n + \Delta t\left[\frac{(F_B^{n+1} + F_B^n)}{2m_B} + g\right], \tag{2}$$

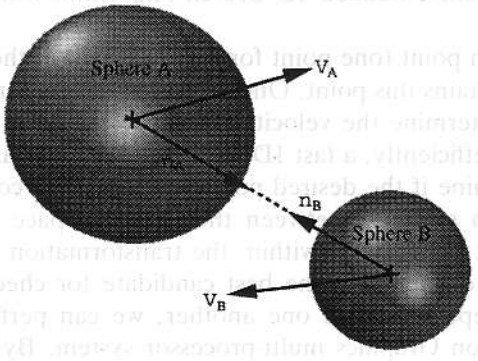

Fig. 3. Notation used in collision equations.

where n and $n+1$ denote the time levels, Δt is the time step size, m_A and m_B are the masses of Spheres A and B, $\boldsymbol{g}$ is the gravity vector, and $\boldsymbol{F}_A$ and $\boldsymbol{F}_B$ are the fluid forces acting on Spheres A and B.

When we determine that two spheres collide between time levels n and $n+1$, we use a second set of equations to correct, due to the collision, the most recent updated velocity at level $n+1$. This new set of collision equations is derived from the conservation of linear momentum and equations relating the relative velocities of the spheres before and after the collision. We assume that the tangential forces are zero and the correction is applied only in the normal direction. We write these equations in a way that we find to be convenient to implement:

$$\boldsymbol{V}_A^{n+1'} = \boldsymbol{V}_A^{n+1} + \left[V_{nA}^n - V_{nA}^{n+1} - \frac{(1+e)m_B(V_{nA}^n + V_{nB}^n)}{m_A + m_B} + g_{nA}\,\Delta t \right] \boldsymbol{n}_A , \tag{3}$$

$$\boldsymbol{V}_B^{n+1'} = \boldsymbol{V}_B^{n+1} + \left[V_{nB}^n - V_{nB}^{n+1} - \frac{(1+e)m_A(V_{nA}^n + V_{nB}^n)}{m_A + m_B} + g_{nB}\,\Delta t \right] \boldsymbol{n}_B , \tag{4}$$

where e is the coefficient of restitution, and quantities such as V_{nA} are defined as $V_{nA} = \boldsymbol{V}_A \cdot \boldsymbol{n}_A$. For collision of Sphere A with the tube wall, we set $m_B \to \infty$, and $V_{nB} = 0$.

Eqs. (3) and (4) are employed in an iterative fashion, where after a collision the positions of the spheres are updated based on these equations. We then check all over again for other collisions between the same time levels that might have been caused by the earlier ones. By handling these collision equations in this manner, sequential collisions between two time levels can be accounted for.

We believe that this collision model works satisfactorily within the scope of the simulations presented here, however, other enhancements to this model could be implemented to improve the accuracy even further.

2.5. Projection algorithm

For the class of problems we are interested in we need an accurate projection method and also an efficient implementation of it. In the simulations presented here, the projection actually takes place within the jump terms of the space–time finite element formulation (see [5]). Since the projection is carried out by using these least-squares type terms already present in our finite element formulation, we believe this is the most natural way of steering through a remesh while integrating in time. See [7] for a more comprehensive investigation on this subject.

The projection is handled numerically by first finding the values of the interpolated quantities (i.e. the components of the current velocity) within the old mesh at points corresponding to the numerical integration points of the new mesh. These values are then used in the jump terms of the finite element formulation for the first space–time slab following remesh. In the simulations reported here, we use one point integration in space. Computing the old values at the new integration points is the most computationally intensive part of this algorithm. To perform this most efficiently for very large meshes, we use a method we call the Multi-Threaded 1D Search Algorithm with Random Restart.

The task of this algorithm is:

For each numerical integration point (one point for each element in the new mesh), find the element within the old mesh which contains this point. Once this element is found, perform the linear element function space mapping to determine the velocity at this point.

To perform this element search efficiently, a fast 1D search algorithm is used. In a 1D search algorithm, an element is checked to determine if the desired point lies within. A convenient equation to perform this check is the transformation equation between the physical space and the element space for a tetrahedral element. If the point does not lie within, the transformation equation will specify which of the four neighbors (tetrahedral elements) is the best candidate for checking next (see Fig. 4). Since searches can be performed independently of one another, we can perform this search in the multi-threaded environment of a Silicon Graphics multi-processor system. By doing so, we have numerous element searches being performed on the same data structure (the old mesh), and we can achieve almost linear parallel speed-ups.

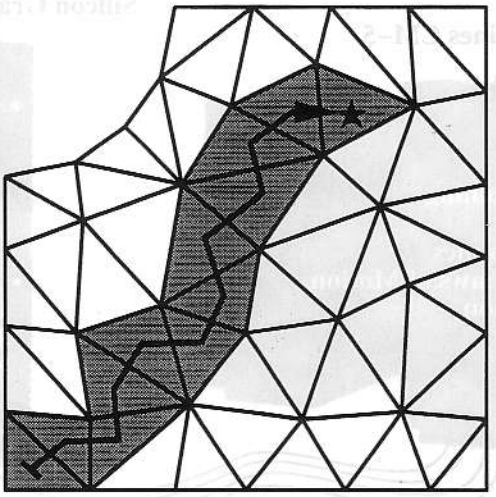

Fig. 4. A 1D search algorithm.

In multiply-connected domains such as those considered here, the 1D search algorithm may encounter a boundary and not be able to locate the containment element. When this happens, we re-initialize the search with a new, randomly chosen element. By doing this, we can ensure that the search algorithm will eventually find the containment element. Of course, other algorithms can be chosen to handle this problem, but this one is simple and works for all cases with minimal loss in performance. Once the containment element is found, the transformation equation in a slightly different form (replace coordinates with the components of velocity) is used to determine the velocity at that point.

Once the velocities are found at each of the integration points of the new mesh, these are written to a file to be used by the flow solver when the simulation is continued. We also solve a least-squares problem to generate an initial guess for the unknown velocities and pressure at the first space–time slab following remesh. This equation is written as

$$\int_{\Omega} \boldsymbol{w} \cdot (\boldsymbol{u} - \boldsymbol{u}^*)\, \mathrm{d}\Omega = 0\,, \tag{5}$$

where $\boldsymbol{u}$ is the velocity within the new mesh (the unknown), and $\boldsymbol{u}^*$ is the velocity within the old mesh. Eq. (5) is solved with a parallel Jacobi iteration technique. For meshes with the number of tetrahedral elements of the order 1.3 million, we can project the solution with our algorithm in roughly 5.5 minutes on a 10 processor Silicon Graphics Onyx (with 150 MHz R4400 processors).

2.6. Multi-platform computation

The simulations reported here are performed in a multi-platform computing environment using two different parallel platforms. One platform is the Thinking Machines CM-5 using the data-parallel programming model, and the other is a Silicon Graphics multi-processor (SGI-MP) using a shared-memory programming model (see Fig. 5). There are four individual tasks that need to be performed to carry out the simulation, and each is assigned to the platform more suitable for that task:

- *Automatic Mesh Generation:* Automatic mesh generators of the type we use here are inherently serial algorithms with relatively large memory requirements. We assign this task to the SGI-MP using only one processor.
- *Projection of Solution:* As we described previously, we do take advantage of multi-threaded parallelism in this algorithm, so this task is also assigned to the SGI-MP.

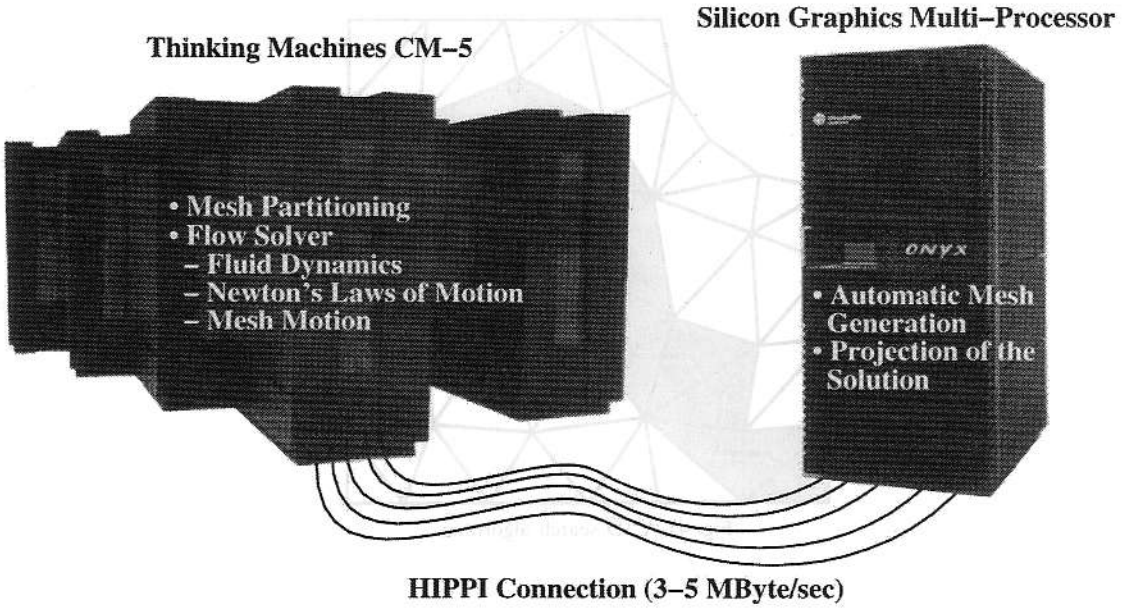

Fig. 5. Multi-platform computation schematic.

- *Mesh Partitioning:* Efficient mesh partitioning libraries are available on the CM-5 using CMSSL [15], so this task is assigned to the CM-5.
- *Flow Solver:* The scale of this application requires the computational power of a parallel supercomputer, and our implementation of the flow solver has been in place on the CM-5 using the data-parallel programming model for several years [14].

Data files are sent between these two computers using a High Performance Parallel Interface (HiPPI) connection which has a transfer rate, in this application, of roughly 3–5 MBytes/second.

The interaction and synchronization of all four of these tasks, and file management, is handled by a shell script. A typical cycle of the computation managed by this shell script is as follows:

1. Automatic exit from the flow solver on the CM-5 when the mesh gets too distorted.
2. Data files from the previous run organized with proper names and moved to proper directories. Pertinent data sent to the SGI-MP through the HiPPI connection.
3. SGI-MP generates a new finite element mesh with the automatic mesh generator.
4. SGI-MP projects the solution from the old mesh on to the new mesh that was just created.
5. New files organized with proper names and moved to proper directories. New mesh and restart information sent to the CM-5 through the HiPPI connection.
6. Partitioning of the new mesh performed on the CM-5.
7. Flow solver restarts on the CM-5 using the new mesh and restart information.
8. Repeat.

If all the hardware and the connections are working properly, the entire simulation can take place automatically by using this shell script without any user interaction.

This multi-platform computing environment seems to work fairly well, and here we have several observations about trying to use such a computing environment for production-type simulations:

- As stated earlier, managing the simulation using the shell script works well if all the hardware and the connections are up and working properly. In a typical multi-user environment, this is not guaranteed. System maintenance, hardware failures, and resource unavailability due to job managers and other users of the system, are all typical occurrences which would disturb this particular multi-platform computing arrangement.

- To perform a multi-platform computation, a very fast connection is required to transfer the large amounts of data between the two computers. The computations reported here would not have been possible without the fast HiPPI connection between the CM-5 and the SGI-MP.
- During a multi-platform computation, part of the system will have to remain idle while the other parts of the system are performing their tasks. In our calculation, the 512 processors of the CM-5 are idle while the remeshing stages are carried out by the SGI-MP. One could, of course, release these 512 processors to other users while the remeshing stages take place, but in our multi-user environment managed by a job manager, we would have to wait sometimes up to a day before computations can proceed on the CM-5 once we have released these processors.
- There are numerous files generated in such a multi-platform computation involving remeshing. We generate data files for the storage of the solution, files to store each mesh and restart information after a remesh procedure, mesh partitioning files, files to store the coordinates, velocities, and forces on each sphere at every time step, and so on. Also, we keep copies of many of these files on the two different file systems used. The management and organization of this many files becomes an important issue.

3. Numerical simulation

We carried out simulations for two different cases of 101 spheres falling in a liquid-filled tube with radius 5.0, and with the axis of the tube aligned with gravity. In the first case, the sizes of the spheres are chosen randomly with their radii being between 0.5 and 1.0. In the second case the radius of each sphere is 0.75. In both cases, the initial positions of the spheres are chosen randomly to fill up the segment of the tube between the horizontal planes $Z = -15.0$ and $Z = 15.0$. The computational domain, which translates downward in the tube with the center of mass of the system of particles, initially extends from $Z = -25.0$ to $Z = 35.0$. In both simulations, all spheres are initially at rest.

The base refinement on the surfaces of the spheres is set to an element edge-length of 0.2, and the base refinement of the tube wall is set to 1.0. However, as discussed in the previous section, the refinement level may increase when spheres become too close to each other or to the tube wall. Around each sphere we create two structured layers of elements, each with a thickness of 0.015.

Other parameters used in these simulations are the same in both cases and also the same as the four cases presented in [5]. These parameters, in their non-dimensional form, are set such that a single sphere with radius 1.0 will fall with a terminal velocity yielding a Reynolds number of approximately 100. The viscosity of the fluid μ is 0.02, the density of the fluid ρ is 1.0, the magnitude of gravity g is -1.0, and the mass density of each sphere ρ_s is 1.47. The moment of inertia for each sphere is computed based on the sphere's mass with the assumption that the sphere is solid. The size of the time step for both simulations is 0.05.

3.1. Case 1: Random sized spheres

This simulation consists of 101 randomly placed spheres with random radii between 0.5 and 1.0. As a result of the spheres having random sizes, each sphere will have a different settling speed. For each sphere, the settling speed increases with the radius because the drag coefficient for a sphere decreases with Reynolds number for the range of Reynolds numbers we are dealing with here [18,19]. The effect of these different settling speeds is that the smaller spheres will slowly fall behind the larger spheres. We also expect several cases of drafting, kissing and tumbling (described in [5,20,21]) to occur.

The mesh sizes for this simulation were of the order 1.2 million tetrahedral elements and 240 000 nodes, however, these numbers varied significantly since they are highly dependent on the arrangement of the spheres. A view of a portion of the initial surface mesh[1] along with a vertical cross-section of this

[1] The number of elements starts below 1 million for the initial mesh, but this number increases to around 1.2 million as the simulation progresses.

mesh is shown in Fig. 6. A close up view of a portion of the initial mesh is shown in Fig. 7. We note in Fig. 6 the patches of higher surface mesh refinement along the tube wall in places where a sphere is close. In this simulation, we had a total of 766 time steps and we remeshed 98 times.

The spheres at four instants during the simulation can be seen in Fig. 8. We note in this figure that the smaller spheres (darker colored) are separating out from the larger ones. There are several smaller spheres trailing behind the rest at the final instant, and the smaller spheres that were initially at the very bottom of the group are now trailing behind several larger spheres.

More quantitative evidence of this separation of the larger spheres from the smaller ones can be seen in Fig. 9 where time histories of the average velocities and positions for the spheres are shown. Some of the quantities plotted in this figure are labeled Small Group, Medium Group and Large Group. The Small Group contains the smallest one third of the spheres, the Medium Group contains the middle one third, and the Large Group contains the largest one third. The noise in some of the curves in Fig. 9 is due to the numerous collisions taking place and, in some part, projections after remeshing. From these curves we can determine a rough number for the terminal velocity Reynolds number based on the average settling velocity and average sphere diameter. The approximate Reynolds number for the whole group is 40 and for the small, medium and large groups is 23, 40, and 57.

In the plot of the Z component of the grouped average velocity, it can be seen that the difference between the velocity in the two larger groups is smaller than the difference between the two smaller groups. We believe this might be due, in part, to the up-drafts in the flow that are created by the settling of the larger spheres. As the spheres settle in this self-contained tube, flow must rise in other areas, and by looking at the flow, it seems that many of the smaller spheres get caught in these up-drafts and some

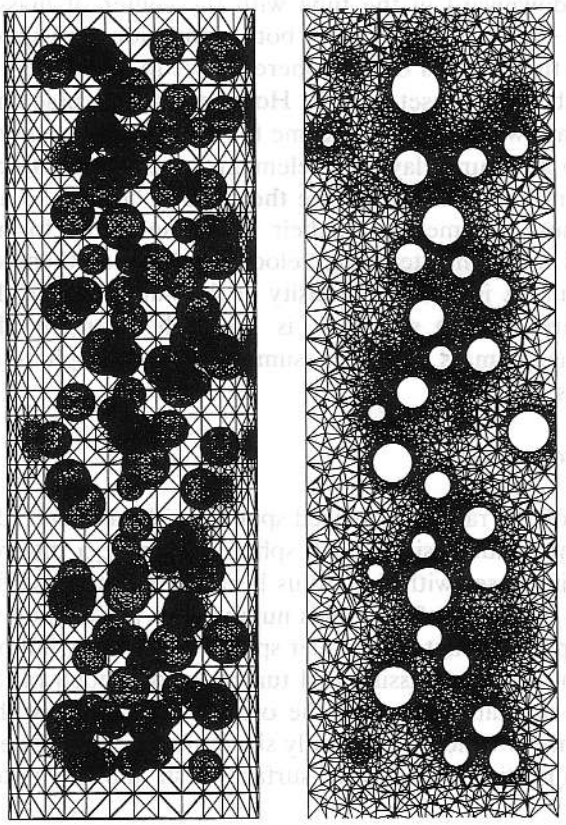

Fig. 6. Case 1: initial mesh with 166 877 nodes and 959 152 elements (as the simulation progresses, the mesh size increases to approximately 240 000 nodes and 1.2 million elements).

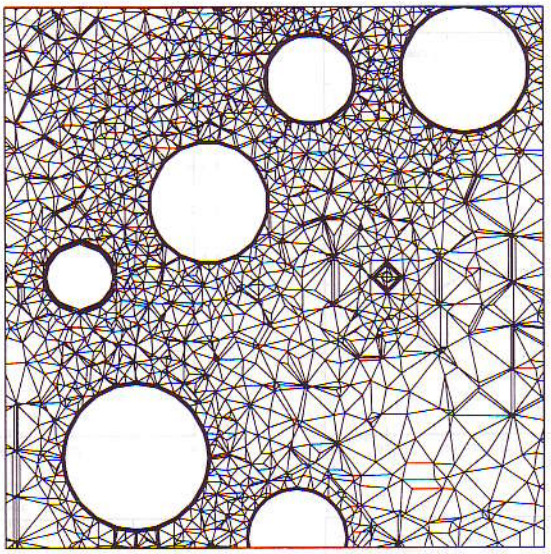

Fig. 7. Case 1: close up view of the initial mesh.

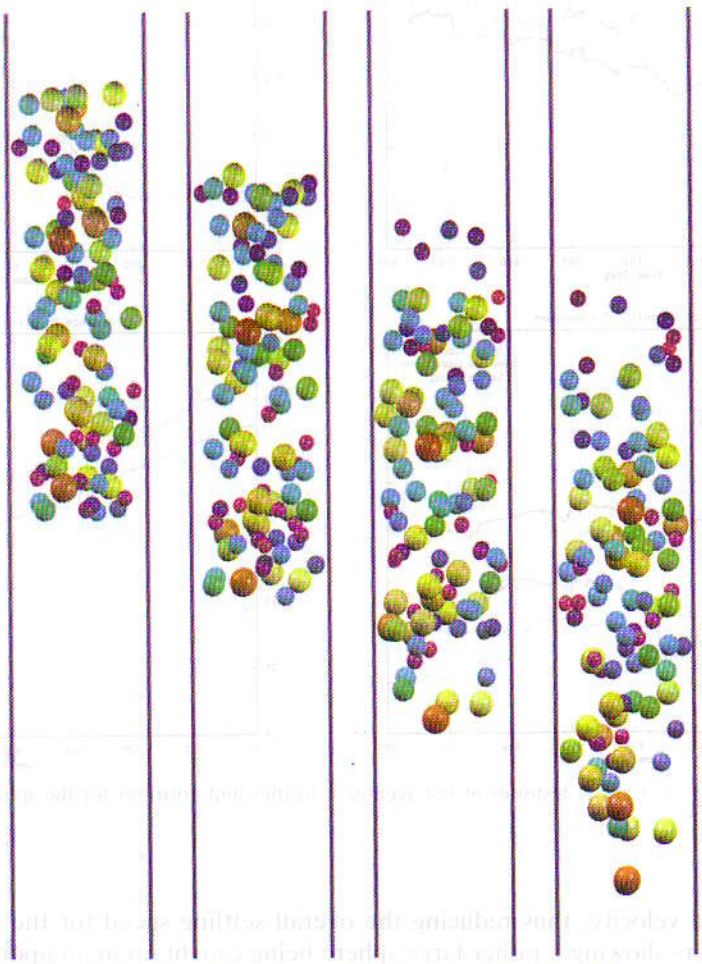

Fig. 8. Case 1: spheres at four instants during the simulation.

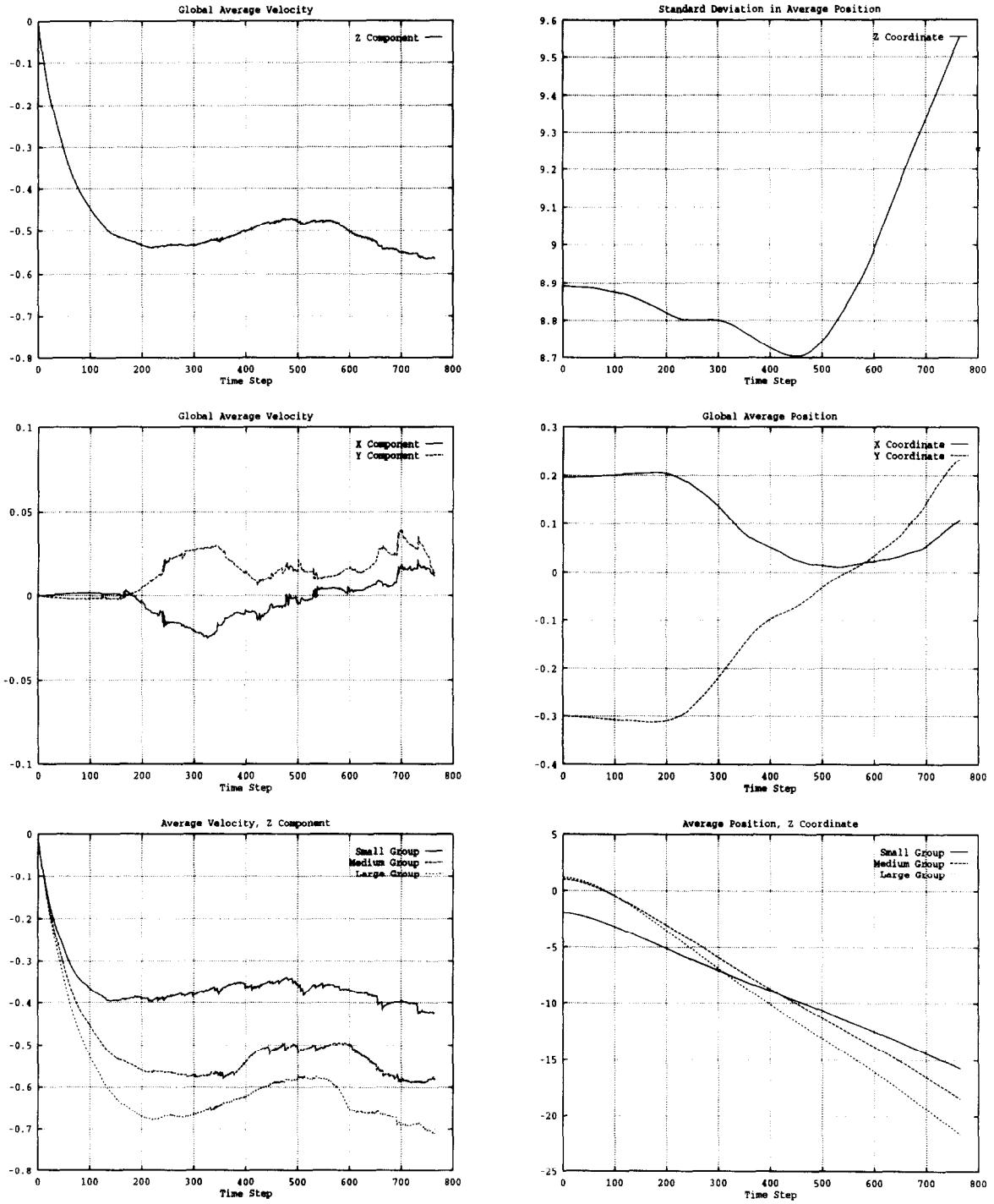

Fig. 9. Case 1: histories of the average velocities and positions for the spheres.

even have an upward velocity, thus reducing the overall settling speed for the small sphere group. A view of velocity vectors showing a rather large sphere being caught up in an updraft is shown in Fig. 10. A more global view of these up-drafts can be seen in Fig. 11 where the Z component of velocity is plotted at two cross-sections (orange and red colors denote an upward velocity).

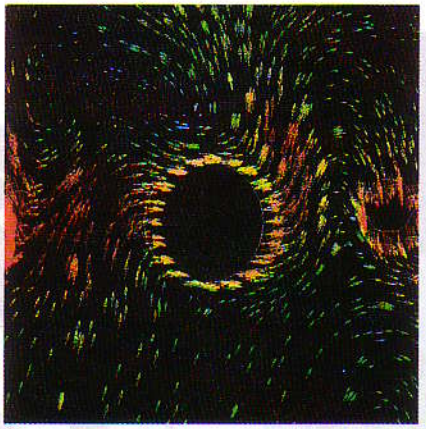

Fig. 10. Case 1: velocity vectors at a cross section.

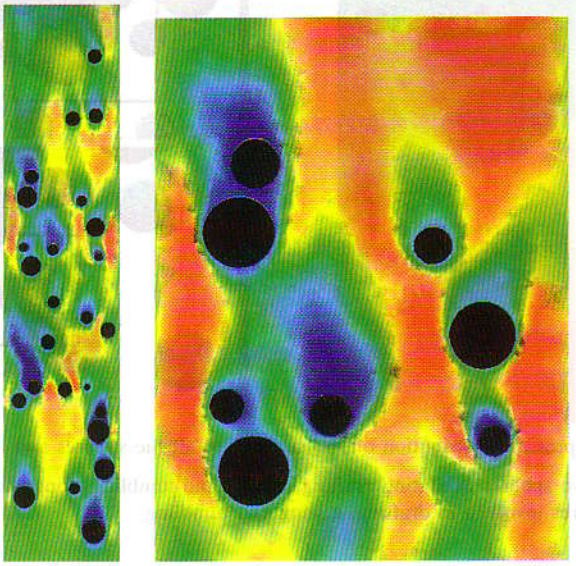

Fig. 11. Case 1: Z component of the velocity at two cross-sections.

Fig. 12 shows, towards the end of the simulation, pressure distribution on the surfaces of all the spheres and at a cross-section.

We observe several cases of spheres drafting, kissing and tumbling in this case. A sequence of pictures for one of such cases is shown in Fig. 13. Also in the same figure, we show a view of all the spheres at a later time with several pairs of spheres involved in drafting, kissing and tumbling highlighted. Views of the Z component of the velocity at cross-sections where two spheres are drafting (one sphere located in the wake of another) are shown in Fig. 14.

3.2. *Case 2: Uniform sized spheres*

This simulation consists of 101 randomly placed spheres each with a radius of 0.75. We chose to carry out this simulation with uniform sized spheres to remove the effect of the spheres separating out from one another due to different settling speeds, and we can get a clearer view of other effects that are taking place.

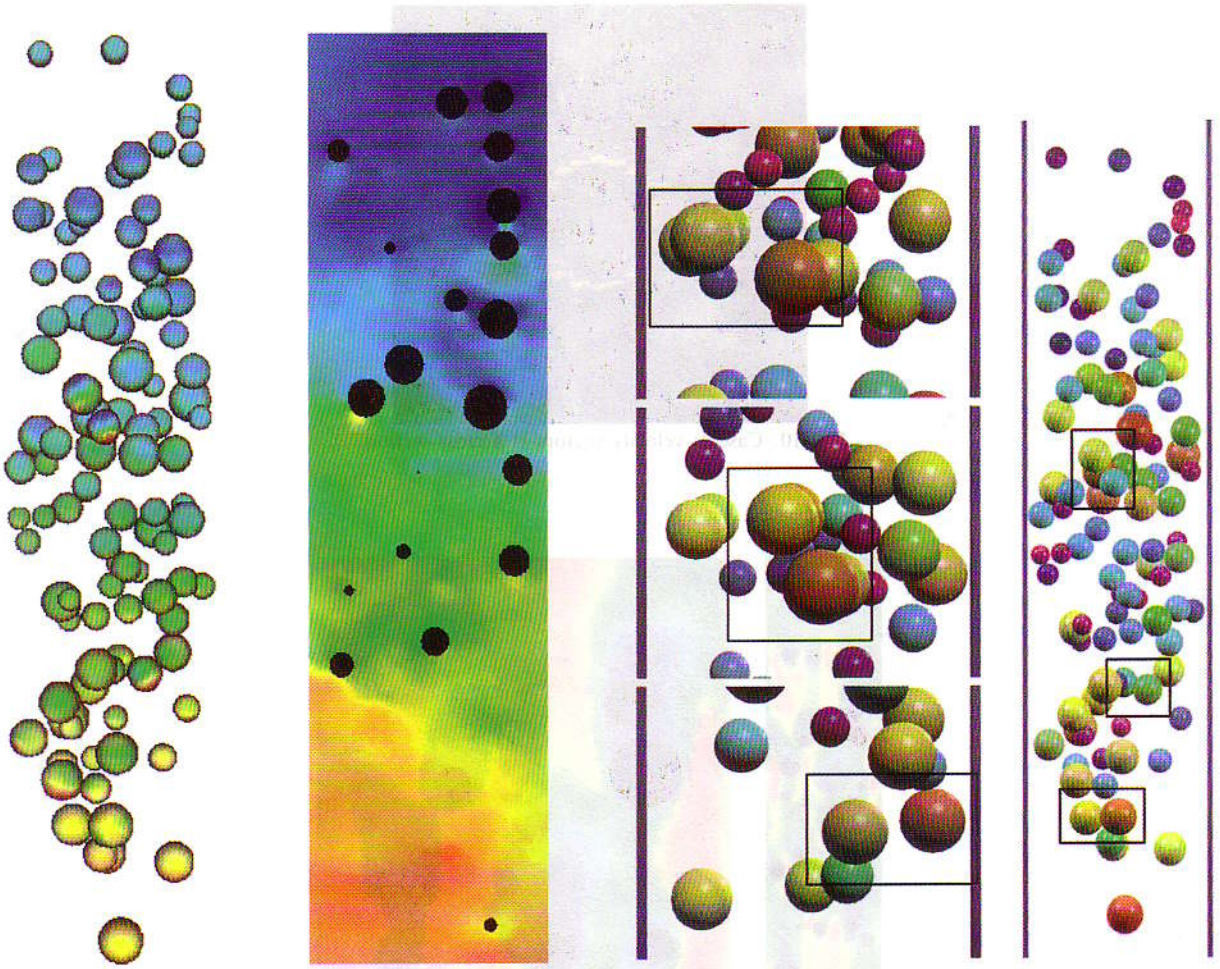

Fig. 12. Case 1: pressure distribution on the surfaces of all the spheres and at a cross-section.

Fig. 13. Case 1: a sequence of two spheres exhibiting drafting, kissing and tumbling along with a global view of all the spheres highlighting several pairs of spheres involved in drafting, kissing and tumbling.

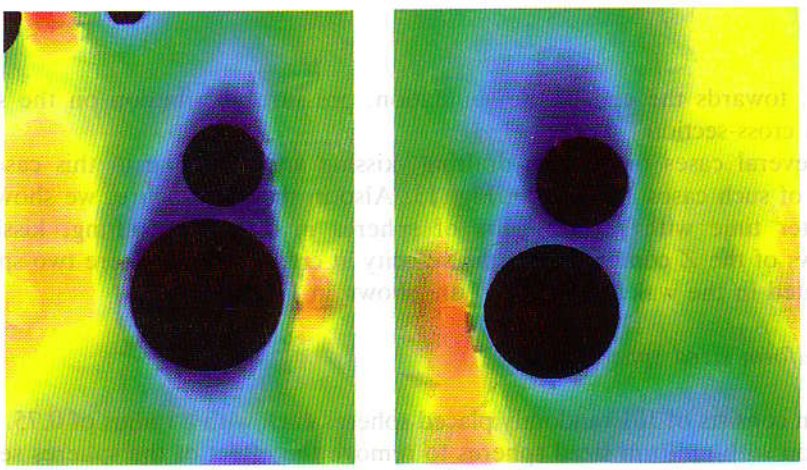

Fig. 14. Case 1: two views of the Z component of the velocity at cross-sections where spheres are drafting.

The mesh sizes for this simulation were again of the order 1.2 million elements and 240 000 nodes. A view of a portion of the initial surface mesh along with a vertical cross-section of this mesh is shown in Fig. 15. A horizontal cross-section of the initial mesh is shown in Fig. 16. In this simulation, we had a total of 869 time steps and we remeshed 98 times.

The spheres at four instants during the simulation can be seen in Fig. 17. Time histories of the average velocities and positions for the spheres are shown in Fig. 18. From these curves we can determine a rough number for the terminal velocity Reynolds number based on the average settling velocity. This approximate Reynolds number is 38.

A global view of the Z component of the velocity at vertical and horizontal cross-sections can be seen in Fig. 19. We can see in this figure the same types of up-drafts as those that were present in the previous simulation. A close-up view of the Z component of the velocity at a cross-section is shown in Fig. 20 along with the velocity vectors colored with their magnitudes at the same cross-section. We can see in this figure three spheres drafting with one another, as well as several up-drafts.

Fig. 21 shows, towards the end of the simulation, pressure distribution on the surfaces of all the spheres and at a cross-section. A closer view of the pressure distribution at this cross-section is shown in Fig. 22. The location of the cross-section in Fig. 22 is the same as the location of the cross-sections in Fig. 20.

We observe several cases of spheres drafting, kissing and tumbling in this case. We observe this behavior in more cases than in the previous simulation with randomly sized spheres. A sequence of pictures for one of such cases is shown in Fig. 23. Also in the same figure, we show a view of all the spheres at a later time with several pairs of spheres involved in drafting, kissing and tumbling

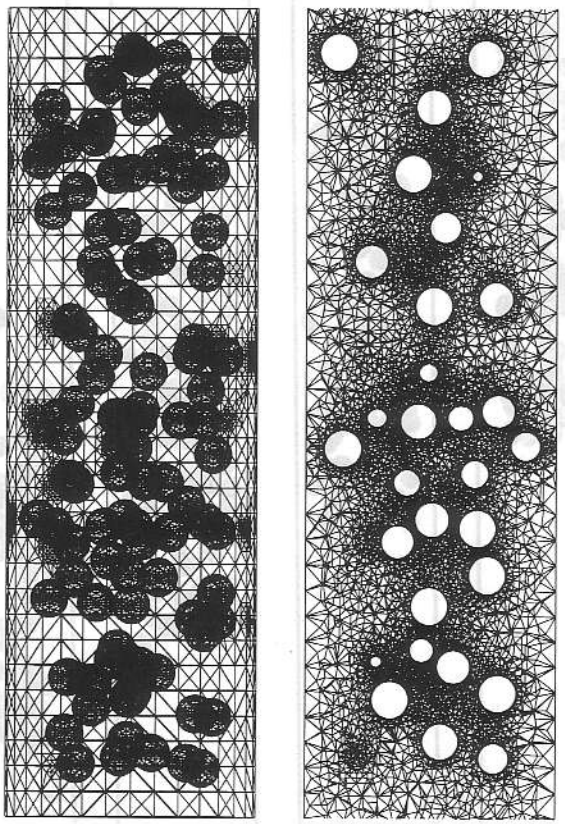

Fig. 15. Case 2: initial mesh with 157 907 nodes and 897 188 elements (as the simulation progresses, the mesh size increases to approximately 240 000 nodes and 1.2 million elements).

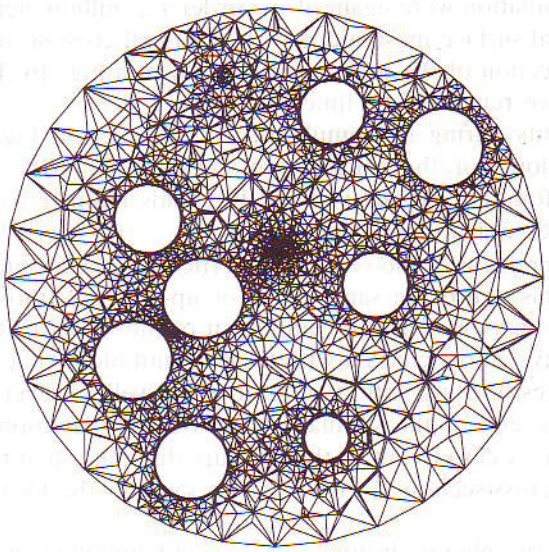

Fig. 16. Case 2: horizontal cross-section of the initial mesh.

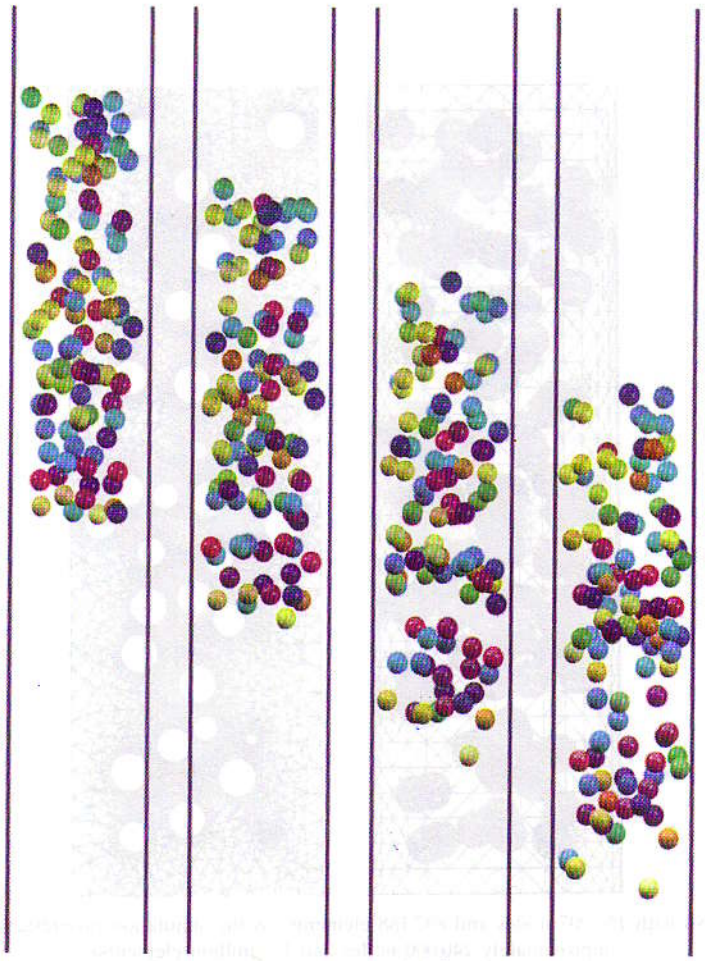

Fig. 17. Case 2: spheres at four instants during the simulation.

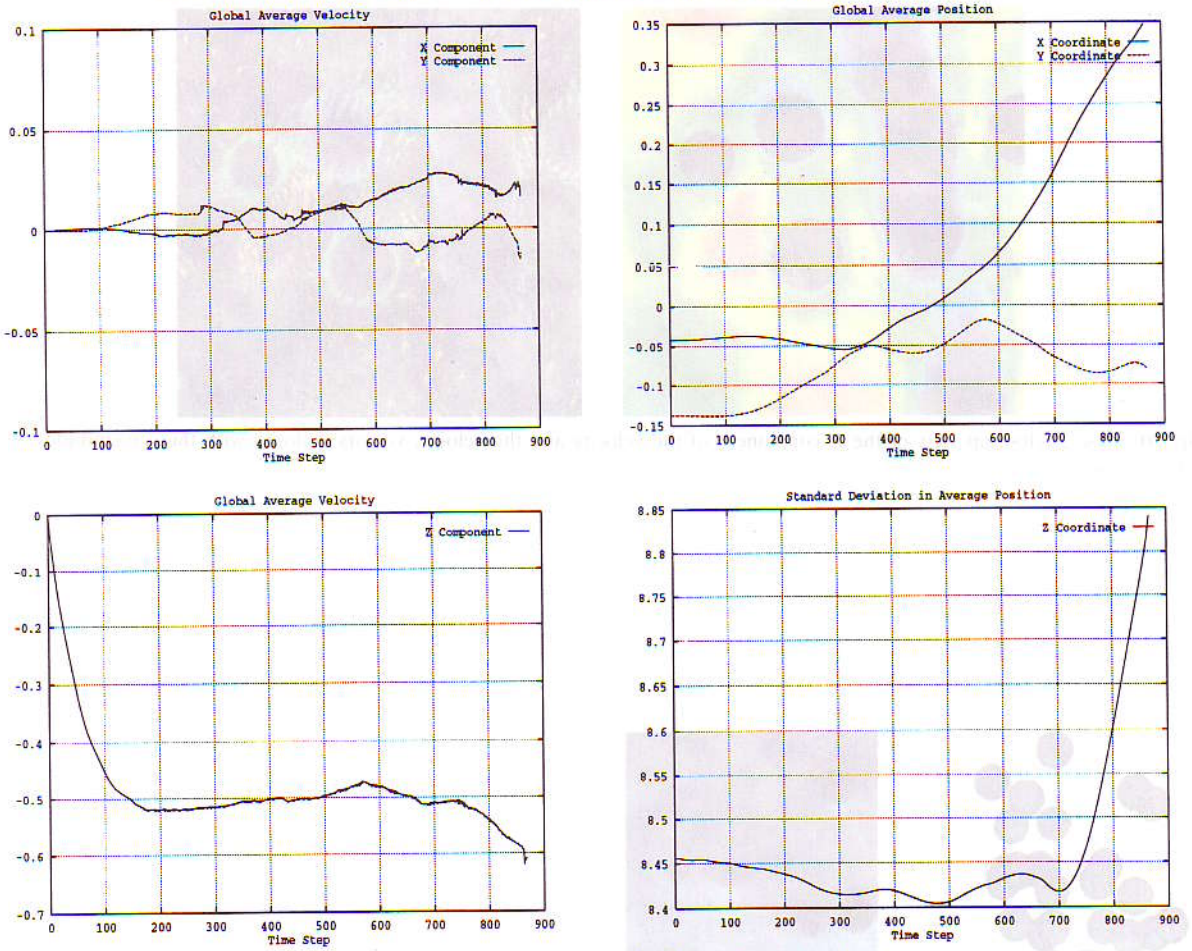

Fig. 18. Case 2: histories of the average velocities and positions for the spheres.

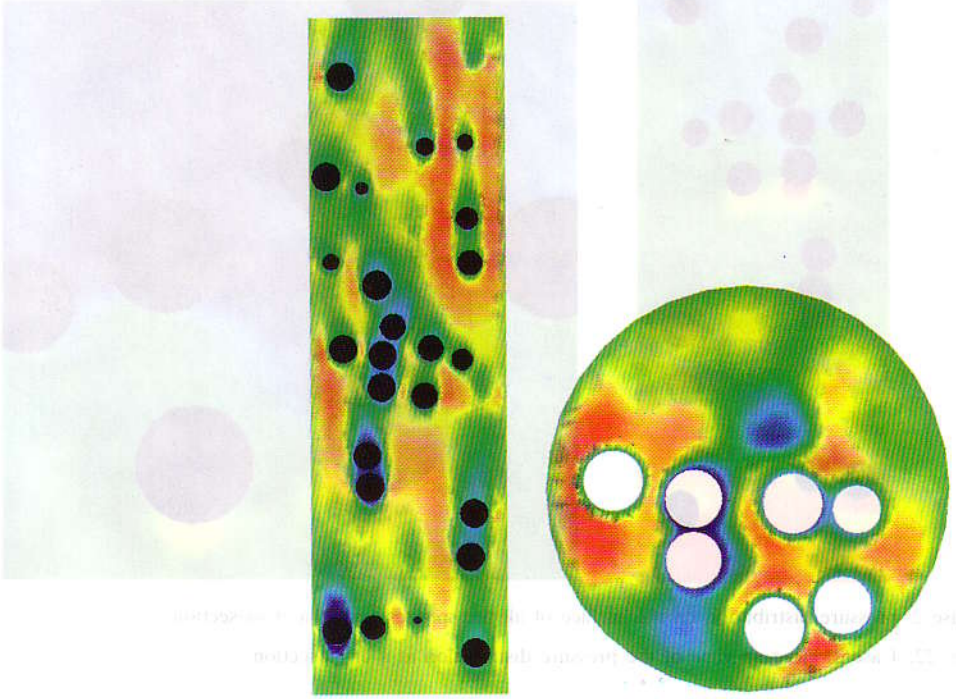

Fig. 19. Case 2: Z component of the velocity at horizontal and vertical cross-sections.

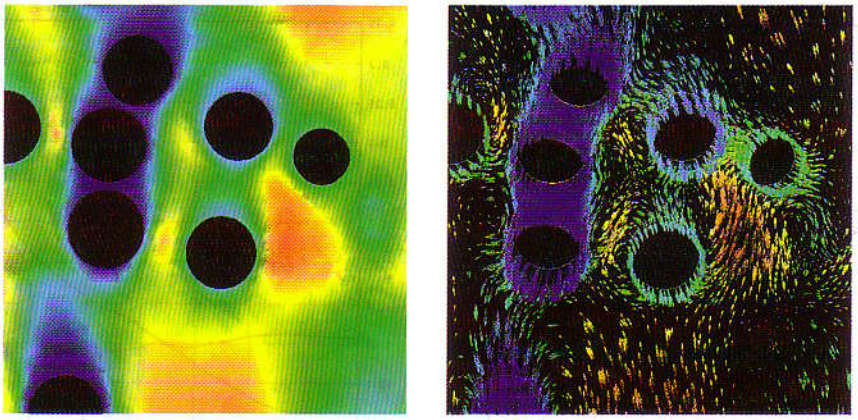

Fig. 20. Case 2: close-up view of the Z component of the velocity and the velocity vectors (colored with their magnitude) at a cross-section.

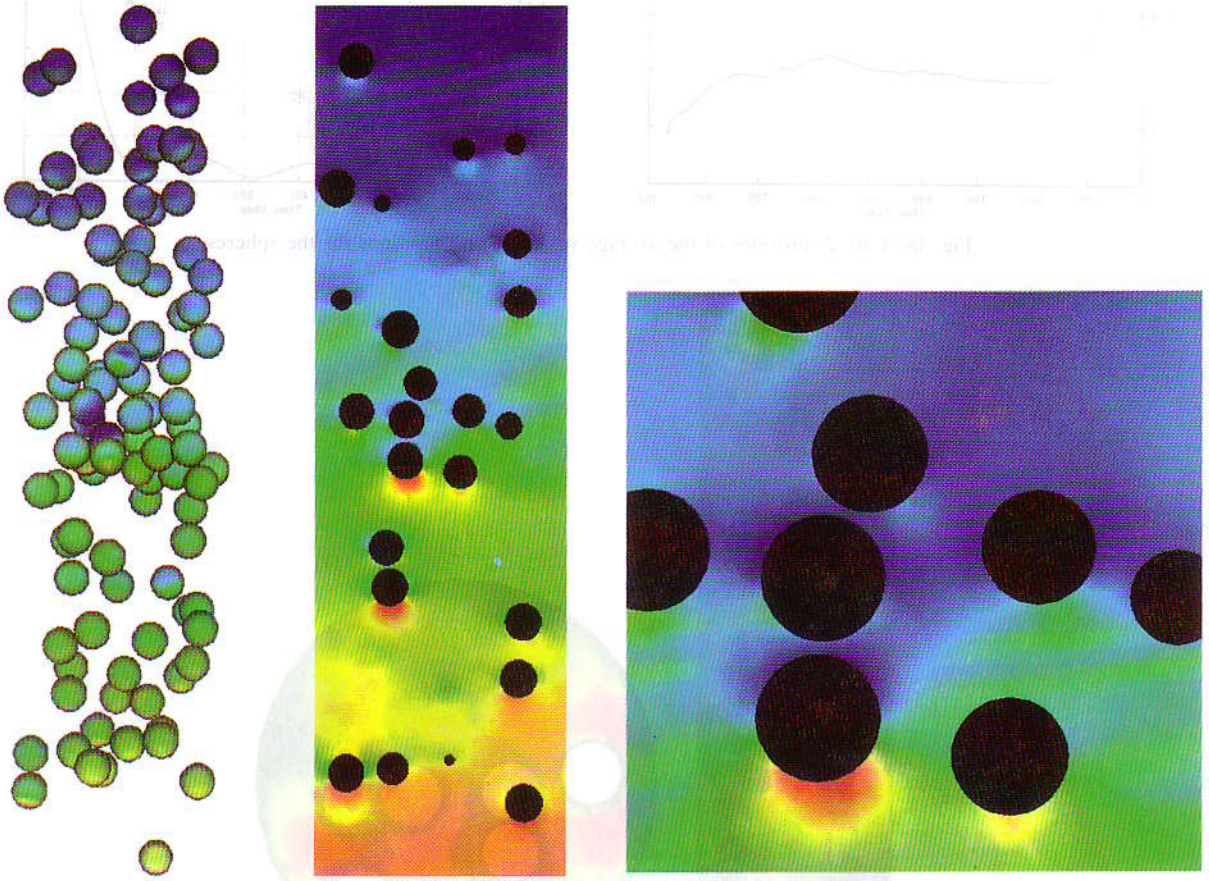

Fig. 21. Case 2: pressure distribution on the surface of all the spheres and at a cross-section.

Fig. 22. Case 2: close-up view of the pressure distribution at a cross-section.

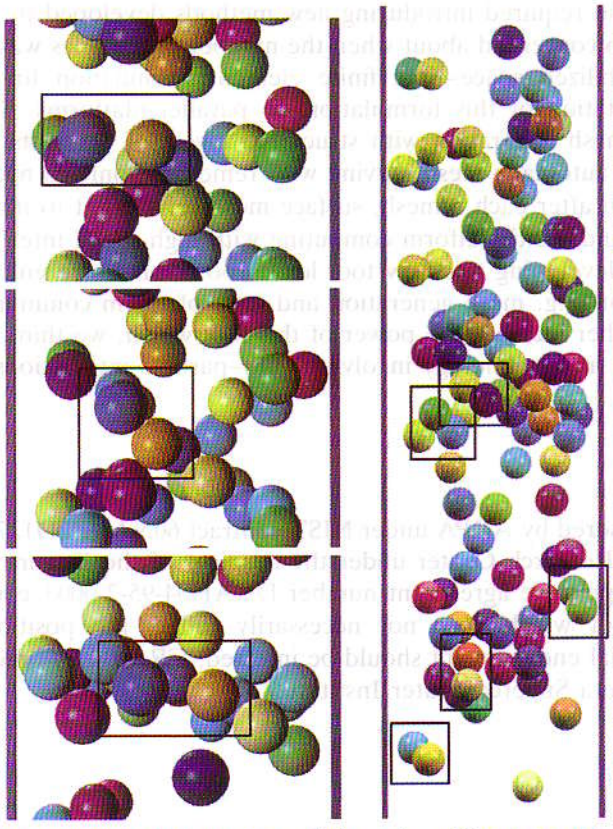

Fig. 23. Case 2: a sequence of two spheres exhibiting drafting, kissing and tumbling along with a global view of all the spheres highlighting several pairs of spheres involved in drafting, kissing and tumbling.

highlighted. By analyzing this simulation, we observe that the drafting behavior in these instances is somewhat exaggerated due to our model of the collision process. During this process, spheres tend to fall while in continued contact with each other for a small period of time, and our current collision model was not specifically designed to model spheres that may be in continued contact with each other. In future simulations we plan to add this capability to our collision/contact modeling equations.

4. Concluding remarks

We presented a new high performance computing tool for 3D simulation of fluid–particle interactions with as many as 100 particles. This tool is capable of keeping the computing durations involved in this class of simulations to acceptable levels. To test and demonstrate this new capability, we applied it to the simulation of 101 spheres falling in a liquid-filled tube. The spheres, in addition to interacting with the fluid, interact and sometimes collide with each other and with the tube wall. We simulated two cases with 101 spheres: with the size of the spheres random in one case and uniform in the second. In both cases, the simulation is started with the spheres distributed randomly in the tube. These particular simulations are not meant to be in depth studies of this class of fluid–particle interactions but are intended to show how the advanced computational methods developed, together with modern parallel computing platforms, enable us to carry out this difficult class of simulations at magnitudes that would have been unthinkable a few years ago.

In developing this tool, although we started with the set of methods we developed earlier for simulations involving an order of magnitude less particles and used some of these methods, bringing the

capability to this level also required introducing new methods developed by taking into consideration issues that we were not so concerned about when the number of particles was much fewer. At the core of this tool are the stabilized space–time finite element formulation for moving boundaries and interfaces and implementation of this formulation on parallel platforms. The surrounding methods include: fast automatic mesh generation with structured layers of elements around particles, a mesh update method based on automatic mesh moving with remeshing only as needed, an efficient method for projecting the solution after each remesh, surface mesh refinement to increase accuracy when two solid surfaces get close, and multi-platform computing with high-speed inter platform communication.

The methods used in developing this new tool leave room for future enhancements in quite a few directions we can think of (e.g. mesh generation and inter-platform communication), and we plan to pursue those later to further increase the power of this tool which, we think, have just opened a new door to exploring science and technology involving fluid–particle interactions.

Acknowledgements

This research was sponsored by ARPA under NIST contract 60NANB2D1272, and by the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. CRAY C90 time was provided in part by the University of Minnesota Supercomputer Institute.

References

[1] A.A. Johnson and T.E. Tezduyar, Mesh generation and update strategies for parallel computation of 3D flow problems, in Computational Mechanics '95, Proceedings of International Conference on Computational Engineering Science, Mauna Lani, Hawaii, 1995.

[2] T.E. Tezduyar and A.A. Johnson, The world of flow simulation, in Lecture Notes on Finite Element Simulation of Flow Problems, Japan Society of Computational Fluid Dynamics, Tokyo, Japan, 1995.

[3] A.A. Johnson and T.E. Tezduyar, Numerical simulation of fluid–particle interactions, in: Proceedings of the Ninth International Conference on Finite Elements in Fluids—New Trends and Applications, Venezia, Italy, 1995.

[4] A.A. Johnson and T.E. Tezduyar, Parallel computation of incompressible flows with complex geometries, Int. J. Numer. Methods Fluids (1997) ~~to appear.~~ **24** (1997) 1321-1340.

[5] A.A. Johnson and T.E. Tezduyar, Simulation of multiple spheres falling in a liquid-filled tube, Comput. Methods Appl. Mech. Engrg. 134 (1996) 351–373.

[6] D. Joseph, R. Glowinski, G. Golub, H. Hu and A. Sameh, Direct simulation of the motion of particles in flowing liquids. WWW html document. Address http://www.aem.umn.edu/Solid–Liquid-Flows/.

[7] A.A. Johnson, Mesh generation and update strategies for parallel computation of flow problems with moving boundaries and interfaces, Ph.D. Thesis, University of Minnesota, 1995.

[8] J.G. Kennedy, M. Behr, V. Kalro and T.E. Tezduyar, Implementation of implicit finite element methods for incompressible flows on the CM-5, Comput. Methods Appl. Mech. Engrg. 119 (1994) 95–111.

[9] P.L. George, Automatic Mesh Generation, Application to Finite Element Methods (John Wiley & Sons, 1991).

[10] P.L. George, F. Hecht and E. Saltel, Automatic mesh generator with specified boundary, Comput. Methods Appl. Mech. Engrg. 92 (1991) 269–288.

[11] T.J. Barth, Aspects of unstructured grids and finite-volume solvers for the Euler and Navier–Stokes equations, in: Special Course on Unstructured Grid Methods for Advection Dominated Flows, Advisory Group for Aerospace Research and Development (1992) 6-1–6-61.

[12] B. Joe, Construction of three-dimensional Delaunay triangulations using local transformations, Comput. Aided Geomet. Des. 8 (1991) 123–142.

[13] R. Löhner, Finite element methods in CFD: Grid generation, adaptivity and parallelization, in: Special Course on Unstructured Grid Methods for Advection Dominated Flows, Advisory Group for Aerospace Research and Development (1992) 8-1–8-58.

[14] M. Behr, A. Johnson, J. Kennedy, S. Mittal and T.E. Tezduyar, Computation of incompressible flows with implicit finite element implementations on the Connection Machine, Comput. Methods Appl. Mech. Engrg. 108 (1993) 99–118.

[15] Thinking Machines Corporation, CMSSL for CM Fortran: CM-5 Edition, Version 3.1, July 1993.

[16] Z. Johan, K.K. Mathur, S.L. Johnsson and T.J.R. Hughes, An efficient communications strategy for finite element methods on the Connection Machine CM-5 system, Comput. Methods Appl. Mech. Engrg. 113 (1994) 363–387.
[17] A.A. Johnson and T.E. Tezduyar, Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces, Comput. Methods Appl. Mech. Engrg. 119 (1994) 73–94.
[18] H. Schlichting, Boundary-Layer Theory, 7th edition, McGraw-Hill, New York, 7th edition (1979).
[19] R. Clift, J.R. Grace and M.E. Weber, Bubbles, Drops, and Particles (Academic Press, 1978).
[20] A.F. Fortes, D.D. Joseph and T.S. Lundgren, Nonlinear mechanics of fluidization of beds of spherical particles, J. Fluid Mech. 177 (1987) 467–483.
[21] H.H. Hu, D.D. Joseph and M.J. Crochet, Direct simulation of fluid particle motions, Theoret. Comput. Fluid Mech. 3 (1992) 285–306.