A Parallel Algorithm for the Dynamic Partitioning of Particle-Mesh Computational Systems*

Jing-Ru C. Cheng and Paul E. Plassmann

Department of Computer Science and Engineering
The Pennsylvania State University, University Park, PA 16802, USA,
jccheng@cse.psu.edu, and plassman@cse.psu.edu

Abstract. Particle tracking methods are a versatile computational technique central to the simulation of a wide range of scientific applications. In this paper we present new parallel approach for the dynamic partitioning of particle-mesh computational systems. The approach uses a framework, the "in-element" particle tracking method, based on the assumption that particle trajectories are computed by problem data localized to individual elements. The parallel efficiency of such particle-mesh systems depends on the partitioning of both the mesh elements and the particles; this distribution can change dramatically because of movement of the particles and adaptive refinement of the mesh. To address this problem we introduce a combined load function that is a function of both the particle and mesh element distributions. We present experiment results that detail the performance of this parallel load balancing approach for a three-dimensional particle-mesh test problem on an unstructured, adaptive mesh.

1 Introduction

Parallel particle tracking methods have been employed to solve a variety of problems, such as Direct Simulation Monte Carlo (DSMC) methods to model the dynamics of dilute gas. Wong and Long [1] implemented parallel DSMC algorithm using a forward Euler explicit time marching scheme in the movement phase of the algorithm. They pointed out that two different levels of data parallelism (i.e., molecules and cells) cause some parallel processing difficulties. Nance et al. [2] parallelized DSMC using the run time library CHAOS [3] on a 3-D uniform discretized mesh. Robinson and Harvey [4] parallelized DSMC by use of a spatial mesh decomposition over the processors. The domain decomposition is based on a localized "load table" computed on each processor for its neighbors, which will receive cells donated by the processor if they have load less than itself. The demonstration problem was a 2-D driven cavity with approximately 10,000 uniform cells. Another field of application of particle tracking methods

* This work was supported by NSF grants DGE–9987589 and ACI–9908057, DOE grant DG-FG02-99ER25373, and the Alfred P. Sloan Foundation.

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2329, pp. 1020–1029, 2002.

is the streamline calculation, for which the most common technique is the integration of particle paths numerically or analytically. Often, for the numerical approach, a high-order ODE solver such as Runge-Kutta methods or Adam's method is required in terms of accuracy [5]. However, in 2-D and 3-D, the evaluation of velocity between grid points across processors required by the high-order ODE solver becomes costly and alternative methods should be considered. Analytic solutions for streamlines within tetrahedra were presented based on linear interpolation and therefore produces exact results for linear velocity fields [6]. However, the analytic method works for steady flow only and the parallel version has not developed yet.

The in-element particle tracking technique was developed by Cheng et al. [7] to accurately and efficiently trace fictitious particles in the velocity fields of real-world systems. Since in-element particle tracking methods are implemented with respect to the element basis, a natural approach is to develop a parallel framework based on specifying local, element-based operations. The key to the parallel algorithm is the correlated partitioning of the particle system and the unstructured element mesh. We partition particles to processors based on their element location—this approach ensures that the data required for the computation of the particle movement phase by the in-element method involves only data local to a processor. The correspondence between particles and elements is maintained through explicit references in the element and particle data structures. This correspondence is essential to ensure the correct reassignment of particles between processors. The reassignment occurs when particles move between elements owned by a processor and the ghost elements (elements with shared faces, edges, or vertices that are owned by another processor) [8].

With the goal of balancing the work load at each time step whether the explicit time-stepping method, the implicit time-stepping method, or streamline calculation, is employed, computational workload estimates need to consider the particle distribution to mesh elements. In general, the workload on each processor is a function of the number of particles and number of vertices assigned to a processor. This distribution can change significantly each time step because of the particle movement and adaptive mesh refinement. This dynamic character raises the issue of load balancing and the problem of determining representative load estimates to achieve a balancing of the assigned work. To address this problem, we define such a load function—a weighted function based on the assignment of particles and vertices to processors.

Experimental results presented in this paper demonstrate the advantage of dynamic load balancing based on this load estimation approach. For a representative parallel particle tracking problem, we present experimental results showing the performance improvement using the load function discussed above. A three-dimensional rotation cone problem is solved using the particle tracking software developed and implemented for the SUMAA3d [9] programming environment.

The remainder of this paper is organized as follows. In Sect. 2, we review in-element particle tracking technology and the extensions required for the parallel implementation are described. In Sect. 3, we discuss the encountered workload

imbalance issues and present a repartitioning strategy. In Sect. 4, experimental results are presented to demonstrate the performance of our implementation. In Sect. 5, we summarize these results and discuss planned future work.

2 Particle Tracking Algorithms

Consider the following advection-dominated system as representative of many problems that are solved by particle tracking schemes. The advection equation for a scalar field $C(\mathbf{x}, t)$, for example the chemical concentration in transport equations, is represented as

$$\frac{\partial C(\mathbf{x}, t)}{\partial t} + \mathbf{u} \cdot C(\mathbf{x}, t) = 0 \ , \tag{1}$$

where the velocity is given by

$$\frac{d\mathbf{x}}{dt} = \mathbf{u}(\mathbf{x}, t) \ . \tag{2}$$

The value of $C(\mathbf{x}, t)$ is constant along the characteristic line defined by the solution to (2) (i.e., $C(\mathbf{x}, t_n) = C(\mathbf{x_o}, t_o)$) and

$$\mathbf{x} - \mathbf{x_o} = \int_{t_o}^{t_n} \mathbf{u}(\mathbf{x}, t) dt \ , \tag{3}$$

where $\mathbf{x_o}$ is the initial point of the particle. The accuracy of the solution to (1) depends on the accurate computation of (3). To address this problem, we have specifically developed the in-element algorithm.

2.1 The Sequential In-Element Algorithm

In Algorithm 2.1 introduced in [8], we give pseudocode describing a generic in-element particle tracking method. This algorithm takes advantage of a number of numerical features. First, the tracking follows the characteristic line element-by-element so that the computational overhead of locating the element in the mesh containing the departure point is reduced. Note that in this way the time-step size is controlled by the element size which preserves the accuracy and stability in the time integration scheme employed. Second, to increase the tracking accuracy within an element, the element can be adaptively refined into a desired number of sub-elements with the interpolated velocity computed at all vertices on sub-elements.

Algorithm 2.1 The In-Element Particle Tracking Algorithm [8]

Let P_0 be the set of particles
Set the residual time t_r to the time-step size
$n \leftarrow \|P_0\|$
Foreach ($p \in P_0$) do

```
        While (t_r > 0) do
            Refine the element M to the prescribed number of subelements
            Track p subelement by subelement until time is exhausted or
                    until the element boundary is hit
            Compute t_r, velocity, and identify the possible neighbor element
                    for next tracking
            if t_r = 0 do
                    Interpolate concentration
            Endif
        Endwhile
Endfor
```

2.2 A Parallel Particle Tracking Algorithm

The parallel particle tracking algorithm is based on the correlated partitioning of the particle system and the unstructured element mesh. Particles are partitioned to processors based on their element location—this approach ensures that the data required for the computation of the Lagrangian step by the in-element method involves only data local to a processor. The correspondence between particles and elements is maintained through explicit references in the element and particle data structures. This correspondence is essential to ensure the correct reassignment of particles between processors. The reassignment occurs when particles move between elements owned by a processor and the ghost elements (elements with shared faces, edges, or vertices that are owned by another processor) [8]. The caching of ghost elements ensures that when the target element found by the tracking algorithm is owned by another processor, the particle is correctly moved to that processor.

Based on the element partioning resulting from the partitioning heuristic, vertex data and element data are never changed during the particle tracking process. Instead, the particles are repartitioned by inheriting their resident element's partition. This approach neither affects the coherency of element neighbor data nor destroys the coherency of vertex data. The disadvantage is the imbalance of particles as the tracking process redistributes particles among processors. Later, we present a repartitioning strategy for the partitioner to balance the workload among the processors. Algorithm 2.2 is the implementation of parallel in-element particle tracking technique neglecting repartitioning at each tracking step.

Algorithm 2.2 The Parallel In-Element Particle Tracking Algorithm[8]

```
Let P_i be the set of particles on processor i
Set the residual time t_r to the time-step size
n ← Σ_i ||P_i||
While (n > 0) do
    n_i ← 0
    Foreach (p ∈ P_i ) do
        While (p ∈ P_i and status(p) ≠ finish) do
            Refine M to the prescribed number of subelements
            Track p subelement by subelement until time is exhausted or
                hit the boundary of the element M
            Compute t_r, velocity and identify the possible neighbor element M'
```

```
                for next tracking
            If t_r > 0 do
                If owner(M') ≠ i do
                    Pack p and remove p from P_i
                    n_i ← n_i + 1
                Endif
            Elseif t_r = 0 do
                Interpolate concentration
                status(p) ← finish
            Endif
        Endwhile
    Endfor
    n = Σ_i n_i
    Send and receive message, unpack messages to form new P_i
Endwhile
Update concentration on each vertex
```

3 A Repartitioning Strategy

For scientific simulations on high-performance parallel computers, it is essential that the partitioner is able to divide an irregular mesh into equal-sized pieces with as few interconnecting edges as possible. In general, the work performed by the simulation on each processor depends on the mesh vertices, the mesh elements, or a combination of of these objects. To balance the workload among processors, a mapping which decomposes the computational mesh onto the processors is commonly found by solving a graph partitioning problem. The graph to be partitioned is simple to construct if the computation is mainly performed on mesh vertices; and a number of adequate partitioning heuristics exist (although the graph partitioning problem itself is known to be NP-complete). However, for parallel particle tracking methods, the workload on each processor is dependent not only on mesh vertices or mesh elements, but also on particles, which move about the mesh dynamically. A different repartitioning strategy is thus required to adapt to these combined particle-mesh computational systems.

To avoid the problem of incurring unnecessary overhead in the repeated mesh partitioning in the in-element particle tracking, we first propose a balance estimator. Note that our approach is based on the assumption that the mesh is completely repartitioned, not "incrementally" repartitioned. Incremental approaches for geometric partitioning schemes have been suggested [10] and implemented for the unbalanced recursive bisection (URB) method [11]. However, the overhead in the data structure reconstructions and message-passing optimizations often makes this approach nearly as expensive (in practice) as a complete repartitioning. Incremental partitioning algorithms based on the "diffusion" of elements or vertices between processors have also been tried but suffer a similar practical downside [12].

Prior to defining the balance estimator, we define several performance metrics. At the k-th tracking step, let ${N_i}^k$ be the total number of particles on processor i, let A^k be the average number of particles on all processors, let M^k be the maximum number of particles, and let m^k be the minimum number of

particles on any processor. These metrics can be represented as

$$\begin{aligned} N_i^k &= \textstyle\sum_{j \in p_i} n_j \\ A^k &= \tfrac{1}{p} \textstyle\sum_{i=1}^{p} N_i^k \\ M^k &= \max\{N_i\}_{i \in [1,p]} \\ m^k &= \min\{N_i\}_{i \in [1,p]} \ , \end{aligned} \tag{4}$$

where p_i is the processor i, n_j is number of particles in element j, and p is number of processors. After obtaining the above measures, the repartitioning is determined to be necessary if the following tw o criteria are met

$$\sum_{i=1}^{p} N_i^k \geq \mathcal{N}_{threshold} \ , \tag{5}$$

and

$$\max\left\{\left(M^k - A^k\right), \left(A^k - m^k\right)\right\} \geq \gamma \mathcal{T} \ , \tag{6}$$

where $\mathcal{N}_{threshold}$ is the threshold number of particles for repartitioning, $\mathcal{T}$ is the total number of vertices of global mesh, and γ is the fraction of $\mathcal{T}$. We employ a load function which is a linear function of the total weight of particles and the total weight of vertices on each processor. Thus, the load of processor i, L_i, is

$$L_i \ = \ \sum_{j \in i} \omega_j \ , \tag{7}$$

where

$$\omega_j \ = \ \alpha + \beta \sum_{e \ni j} n_e \tag{8}$$

is the weight of local vertex j in the processor i, α and β are the weighting coefficients with respect to vertex and particle, and n_e is number of particles in the element e next to vertex j.

Based on the load function $\{L_i\}_{i=1}^{i=p}$ and the distribution of $\{\omega_j\}_{j=1}^{j=\mathcal{T}}$, the partitioner repartitions the mesh based on the graph partitioning theorem and the associated heuristics. Clearly, this partitioning is based only on vertices if $\beta = 0$ and $\alpha = 1$.

4 Experimental Results

In this section we present results on the scalability and accuracy of the parallel in-element approach in solving for the advection of a sharped-peaked concentration cone in a three-dimensional domain. Backward particle trac king is performed under the following flow field:

$$\mathbf{u} = (V_x, V_y, V_z) \ = \ (-\vartheta y, \vartheta x, 2), \quad \vartheta \ = \ \frac{\pi}{500} \tag{9}$$

where V_x, V_y, and V_z are the velocity components in x-, y, and z-directions, respectiv elyA region of [-3000,3000]×[-3000,3000]×[0,3000] is discretized to tetrahedral elements. The fictitious particles to be backward tracked are originally the

domain vertices. After a tracking time period of 500, the fictitious particles can be analytically determined by using the relationship (3) with time integration from 0 to 500. The initial conditions for this problem are specified to demonstrate the capability of parallel in-element particle tracking method to solve transport equations for a system with infinite Peclet number and for Courant numbers C_r in excess of 1 over the entire grid.

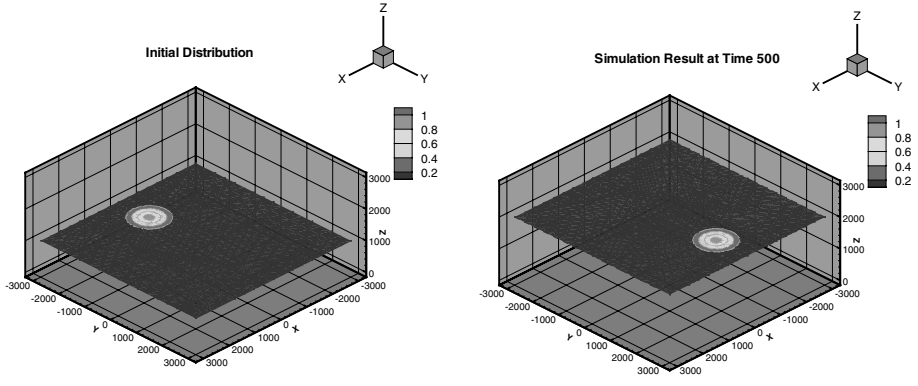

Fig. 1. The initial distribution (left) and the simulation result at time 500 (right) of the 3-D rotation cone problem

This problem is the 3-D version of "rotating cone" problem, a standard benchmark case for numerical advection algorithms [13]. The governing equation for a pure advection of concentration hills in a three-dimensional regime is given by (1). The initial and boundary conditions are given by

$$\begin{aligned} C(x,y,z,0) &= C_0(x,y,z) \\ C(x,y,z,t) &\to 0 \qquad \text{as} \quad x^2+y^2+z^2 \to \infty \ , \end{aligned} \tag{10}$$

in which $C_0(x,y,z)$ is given by

$$C_0(x,y,z) = \begin{cases} H(1-\frac{|\mathbf{x}-\mathbf{x_0}|}{r_0}) & \text{if } \mid \mathbf{x}-\mathbf{x_0} \mid \leq r_0 \ , \quad \text{and } z = 1,000 \\ 0 & \text{otherwise} \end{cases} \ , \tag{11}$$

where r_0 is the radius of the initial cone-hill distribution. The exact solution for this system is

$$C(x,y,z,t) = C_0(x - V_x t, y - V_y t, z - V_z t) \ . \tag{12}$$

The domain is discretized to 612,821 tetrahedrons and 113,847 vertices. The initial distribution based on (11) and the simulation result from the parallel

in-element tracking algorithm (i.e., Algorithm 2.2) are presented in Fig. 1. The maximum absolute pointwise error of this simulation is in the order of 10^{-6}.

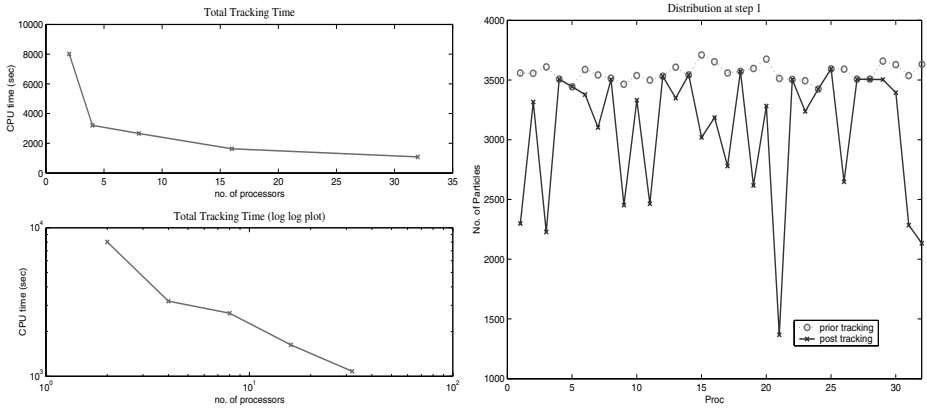

Fig. 2. On the left, the performance of the particle tracking benchmark as a function of the number of processors on an IBM SP2. On the right, we show the imbalance of particle numbers as a function of processor number resulting from a tracking step.

Concerning the performance, Fig. 2 depicts the CPU time spending in particle tracking, both in normal and log-log scale. As expected, the slope of the log-log plot in Fig. 2, i.e., speedup in tracking, close but less than 1 except a kink between 4 processors and 8 processors. To further investigate the workload across the processors, on the right we plot number of particles versus processor number and find that the distribution of particles across the processors has changed significantly after a tracking step. Thus, the presented repartitioning strategy described in Sect. 3 is applied to achiev ethe goal of load balance for parallel particle tracking methods.

The parameters required in the repartitioning strategy have been set to $\alpha = 1$, $\beta = 1, 10, 50, 100$, or 500, $\mathcal{N}_{threshold} = 500 \times p$, where $p = 32$ in this experiment, and $\gamma = 0.02$. Fig. 3 shows the performance improvement with $\beta > 0$, i.e., the repartitioning strategy is employed. Although the partitioning overhead incurs approximately 10% of the total tracking time, the speedup due to the balanced particle tracking outperforms the required overhead. To demonstrate the partitioner does the mesh partitioning as expected, Fig. 4 shows the balanced distribution of ω_i and total number of particles among the processors as $\beta = 10$.

5 Summary and Future Plans

We have developed a dynamic repartitioning method that is appropriate for particle-mesh methods based on in-element particle tracking. We have implemented this method within the SUMAA3d unstructured mesh programming

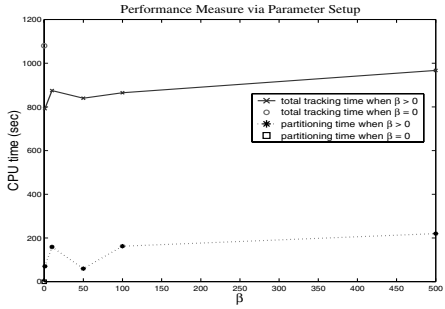

Fig. 3. The performance improvement and the partitioning overhead with the repartitioning strategy implemented. Note the relatively poor performance without any repartitioning, (i.e., $\beta = 0$).

environment, a system that includes the functionality of mesh generation, mesh optimization, adaptive mesh refinement, and sparse matrix solution. Testing of this approach for an advection-dominated test problem on 32 nodes of an IBM SP computer indicates that it significantly improves the parallel performance for this benchmark problem. From these experimental results, we observe that the proposed load function and balance estimator do improve the balance of particles to processors, in addition to the performance, for a wide range of parameter values. Additional issues, such as including an appropriate error estimator in the mesh refinement, particle tracking applications in additional areas, and experiments with more complicated flow fields are topics for further study.

References

[1] Wong, B.C., Long, L.N.: A data-parallel implementation of the dsmc method on the connection machine. Computing Systems in Engineering Journal **3** (1992)

[2] Nance, R.P., Wilmoth, R.G., Moon, B., Hassan, H.A., Saltz, J.: Parallel dsmc solution of three-dimensional flow over a finite flat plate. In: Proceedings of the ASME 6-th Joint Thermophysics and Heat Transfer Conference, Colorado Springs, Colorado, AIAA (June 20-23, 1994)

[3] Moon, B., Uysal, M., Saltz, J.: Index translation schemes for adaptive computations on distributed memory multicomputers. In: Proceedings of the 9-th International Parallel Processing Symposium, Santa Barbara, California, IEEE Computer Society Press (April 24-28, 1995) 812–819

[4] Robinson, C.D., Harvey, J.K.: A fully concurrent dsmc implementation with adaptive domain decomposition. In: Proceedings of the Parallel CFD Meeting, 1997. (1997)

[5] Malevsky, A.V., Thomas, S.J.: Parallel algorithms for semi-Lagrangian advection. International Journal for Numerical Methods in Fluids **25** (1997) 455–473

[6] Diachin, D., Herzog, J.: Analytic streamline calculations for linear tetrahedra. In: 13th AIAA Computational Fluid Dynamics Conference. (1997) 733–742

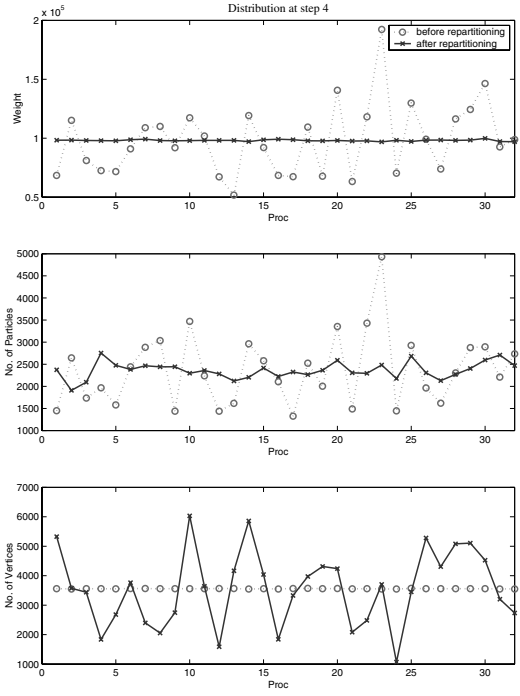

Fig. 4. The distribution of measured metrics before and after a tracking step

[7] Cheng, H.P., Cheng, J.R., Yeh, G.T.: A particle tracking technique for the Lagrangian-Eulerian finite-element method in m ulti-dimensions. International Journal for Numerical Methods in Engineering **39** (1996) 1115–1136

[8] Cheng, J.R.C., Plassmann, P.E.: The accuracy and performance of parallel in-element particle trac king methods. In: Proc. 10th SIAM Conf. on Parallel Processing, SIAM (2001) 252–261

[9] Freitag, L., Jones, M., Ollivier-Goodh, C., Plassmann, P.: SUMAA3dWeb page. (http://www.mcs.anl.gov/sumaa3d/, Mathematics and Computer Science Division, Argonne National Laboratory)

[10] Berger, M., Bokhari, S.H.: A partitioning strategy for nonuniform problems on m ultiprocessors.IEEE Transactions on Computers **C-36** (1987) 570–580

[11] Jones, M.T., Plassmann, P.E.: Computational results for parallel unstructured mesh computations. Computing Systems in Engineering **5** (1994) 297–309

[12] Karypis, G., Kumar, V.: A performance study of diffusive vs. remapped load-balancing schemes. In: Proceedings of the 11th International Conference on Parallel and Distributed Computing Systems. (1998)

[13] Convection-Diffusion Forum: Specification of five covection-diffusion benchmark problems. 7th International Conference on Computational Methods in Water Resources, Massach usetts Institute of Technology (1988)