11th World Congress on Computational Mechanics (WCCM XI)
5th European Conference on Computational Mechanics (ECCM V)
6th European Conference on Computational Fluid Dynamics (ECFD VI)
E. Oñate, J. Oliver and A. Huerta (Eds)

A COMPARISON OF PARALLELIZATION STRATEGIES FOR THE MATERIAL POINT METHOD

Kevin P. Ruggirello* **AND Shane C. Schumacher†**

* Sandia National Laboratories
PO Box 5800 MS-0836
Albuquerque, NM 87185-0836
e-mail: kruggir@sandia.gov

† Sandia National Laboratories
PO Box 5800 MS-0836
Albuquerque, NM 87185-0836
e-mail: scschum@sandia.gov

Key words: Material point method, parallel, high performance computing

Abstract. Recently the Lagrangian Material Point Method (MPM) [1] has been integrated into the Eulerian finite volume shock physics code CTH [2] at Sandia National Laboratories. CTH has the capabilities of adaptive mesh refinement (AMR), multiple materials and numerous material models for equation of state, strength, and failure. In order to parallelize the MPM in CTH two different approaches were tested. The first was a ghost particle concept, where the MPM particles are mirrored onto neighboring processors in order to correctly assemble the mesh boundary values on the grid. The second approach exchanges the summed mesh values at processor boundaries without the use of ghost particles. Both methods have distinct advantages for parallelization. These parallelization approaches were tested for both strong and weak scaling. This paper will compare the parallel scaling efficiency, and memory requirements of both approaches for parallelizing the MPM.

1 INTRODUCTION

The MPM was first introduced by Sulsky et. al. [1] to solve solid mechanics problems by representing the solid as a series of Lagrangian particles and utilizing a background grid to conveniently compute gradients and solve the conservation equations. In the original MPM formulations, piecewise constant linear basis functions were used to map the particle properties to the background grid and from the grid to the particles to update their properties. Since its development the MPM has been utilized to solve a variety of structural dynamics[3] and fluid-structure interaction problems[4, 5, 6].

Recently the MPM has been integrated into the Eulerian finite volume shock physics code CTH[2] at Sandia National Laboratories for improved structural response and to solve fluid-structure loading problems. Typical problems of interest can easily require upwards of tens of millions of computational cells and hundreds of millions of MPM particles. This necessities that the algorithms used in the method scale to several thousand processors to solve problems of this magnitude. This paper will introduce two parallelization strategies for the MPM that were explored in CTH. The first approach is a ghost particle concept, where the particles are ghosted onto neighboring processors and used to assemble the correct vertex data at mesh boundaries. The second method is a vertex ghost cell approach, where the summation of the vertex data is exchanged at mesh boundaries. The parallel efficiency and memory scaling will be analyzed for both approaches.

2 PARALLELIZATION STRATEGIES

2.1 Ghost Particles

The ghost particle approach was devised to try to minimize the number of communications and barriers within the MPM implementation. By having a ghost particle on each mesh block, the vertex data can be correctly computed when needed without any additional communications. Figure 1 illustrates the communication steps for ghosting a MPM particle for a 2D mesh. The communications are broken into separate X, Y, and Z communications. First the real particle is ghosted in the X direction to the neighboring processor, then the real particle and the additional ghost particles that was previously communicated are ghosted in the Y direction. In 3D the Z communication would ghost the real particle plus the 3 additional ghost particles. At the end of the split direction communications all the processors have the necessary ghost particles to correctly assemble all the background grid data.

In parallel a particle in a corner cell results in 3 additional ghost particles in 2D, and 7 in 3D. A particle along a mesh boundary, not at a corner cell, gives 1 additional ghost particle in 2D and 2 ghost particles in 3D. This requires additional memory to be allocated to store the ghost particles that are created during each communication.

The memory layout chosen for all the particle property arrays is shown in Fig. 2. The real particles for a given processor are stored contiguously in memory, followed by all the necessary ghost particles, and additional memory at the end of the array. The additional memory is necessary since the mesh decomposition is static during the simulation, and the real particles can move throughout the mesh which can result in localized memory loads and computational load imbalances. The additional memory required for the ghost particles exacerbates the issue. A linked list data structure could have been utilized, but the cost of transversing the list to assemble the background grid values would significantly increase the computational time. By having all the particles contiguous in memory the compiler is able to vectorize many of the loops efficiently.

To estimate the amount of memory required, the problem is initialized and the max-

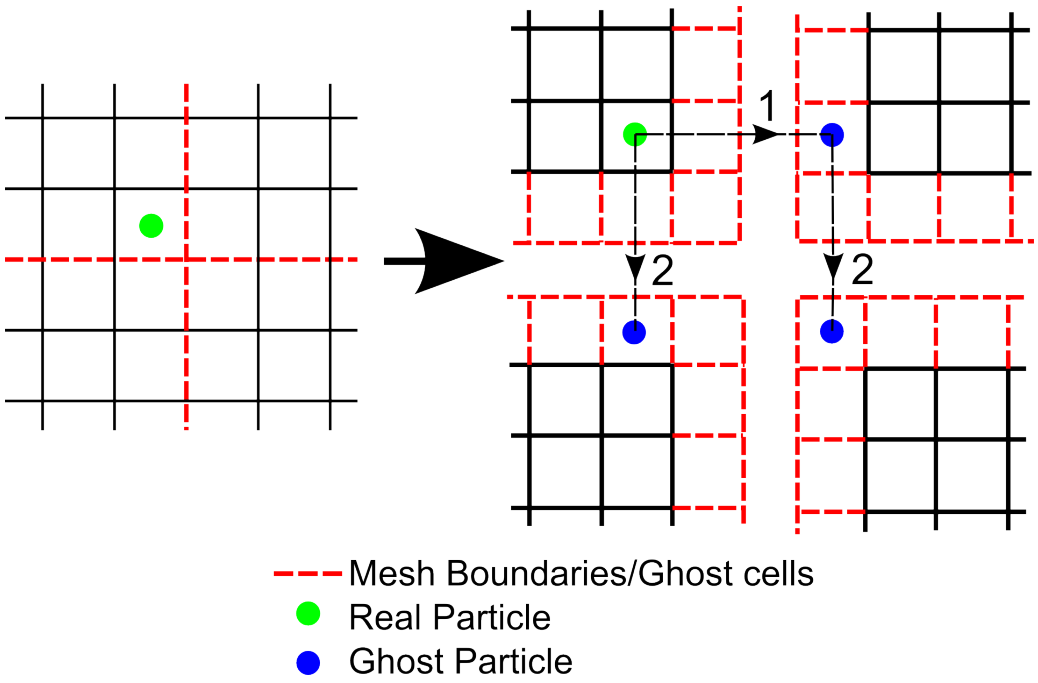


Figure 1: Illustration of communication order for ghost particle parallelization.

imum number of particles on a processor is multiplied by a constant. That number of particles is then allocated on all the processors. For most problems using ghost particles a factor of 8 was chosen for the additional memory factor. This was found to give enough memory throughout the problem for most problems, but will still fail for highly convergent problems where the particles become localized to a small number of mesh cells potentially residing on a single processor. To resolve this issue particle combining or dynamic load balancing of the particles must be implemented. These approaches were not explored in this study.

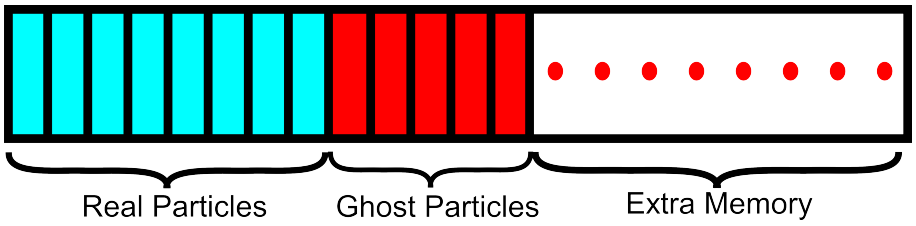


Figure 2: Memory layout for particle properties.

2.2 Vertex Ghost Data

To avoid the additional memory requirements from ghost particles, a vertex ghost data method was implemented. First the particle properties are summed on the background

grid at the vertices, and the sum is communicated and added to the neighboring processors ghost vertices. All processors now have the correct vertex data to solve the conservation equations, and interpolate the updated states back to the particles. This approach is very similar to the standard ghost cell parallel approach typically utilized in explicit computational fluid dynamics methods, except we are communicating sums instead of directly overwriting the ghost cell data.

Compared to the ghost particle approach, this method reduces the amount of memory required but adds additional communication steps since both the numerator and denominator sums have to be communicated separately then normalized after the communication. Additional memory is still required since the particles can still localize to a small region of the problem, but additional storage for ghost particles is not required so a smaller memory factor of 2 is found to be sufficient for most problems.

3 RESULTS

For all the scaling studies performed a single material 3D elastic Reimann problem is solved. The problem starts with a high pressure in all but 1/8th of a 20cm cube, as shown in Fig. 3. The high pressure, shown in red, is initially at 1E10 dynes/cm^2 while the low pressure region, shown in blue, is at 1E6 dynes/cm^2. The material is chosen to be copper and is modeled with a Mie-Grüneisen equation of state and an elastic-plastic model with a Poisson ratio of 0.27 and a yield strength of 1E30 dynes/cm^2 so that the material does not undergo any plastic deformation during the problem. All the scaling studies are performed with no I/O or visualization enabled, and are run for 10,000 computational cycles.

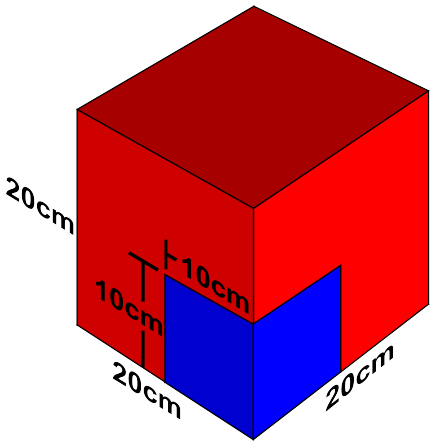


Figure 3: Initial problem setup for all scaling studies.

3.1 Strong Scaling

Strong scaling refers to the case where a problem is first run on a single processor and then the processor number is scaled keeping the problem size constant. The maximum speed up is given by Amdahl's Law as,

$$\text{Speed Up} = \frac{1}{\alpha + \frac{1-\alpha}{P}} \tag{1}$$

where α is the portion of the code that does not execute in parallel, and P is the number of processors. Ideally the serial portion of the code will be almost zero and the speed up will be equal to the number of processors. As the amount of work per processor decreases with increasing processor counts, the proportion of time the code spends doing parallel communications will increase and at some point it is expected that any additional processors will not speed up the computation anymore. The parallel speed up and efficiency for strong scaling are calculated as,

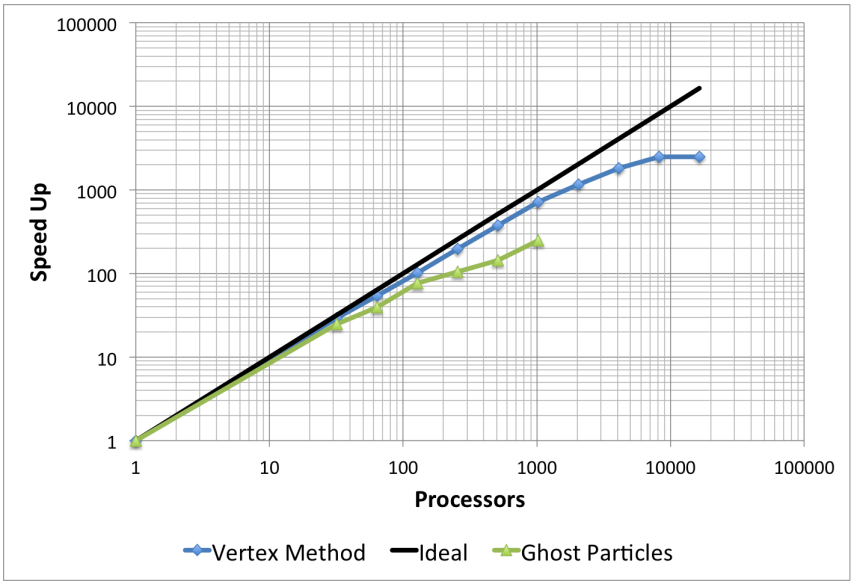
$$\begin{aligned} \text{Speed Up}_{\text{strong}} &= \frac{t_1}{t_P} \\ \text{Eff}_{\text{strong}} &= \frac{t_1}{t_P P} \end{aligned} \tag{2}$$

where t_1 is the processing time for a single work unit with 1 processor, and t_P is the processing time for a work unit with P processors. The total number of work units for a problem is defined as,

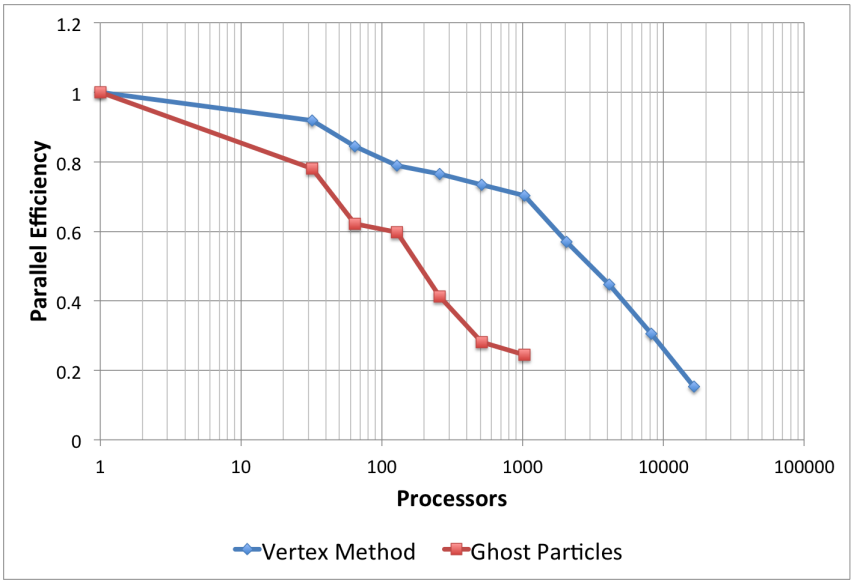
$$\text{Work Units} = \text{Cycles} \times (\text{Mesh Cells} + \text{Particles}) \tag{3}$$

The strong scaling study is performed with 120^3 grid cells with 9 MPM particles per cell, resulting in 3.0375E7 particles in the problem. This is the largest possible problem that would fit in the memory of a single node and complete within the wall clock limit of the queueing system. The number of processors is then scaled.

The parallel speed up and efficiency for both methods are shown in Fig. 4. For the vertex ghost data method, the parallel efficiency is 0.57 and 0.15 at 2,048 and 16,384 processors, respectively. Going from 8,192 to 16,384 processors only decreased the computational time by approximately 3%, so at this point parallel communications are dominating the problem time. With 16,384 processors there are approximately 1,850 particles per processor and approximately 216 mesh cells. The ghost particle method is only scaled up to 1,024 processors because the parallel efficiency falls off rapidly past this point due the additional memory operations required for ghost particles. At 1,024 processors the ghost particle method has a parallel efficiency of 0.24 compared to 0.70 for the vertex ghost data approach.



(a)



(b)

Figure 4: Strong scaling study showing (a) parallel speedup, and (b) parallel efficiency for ghost particle and vertex ghost data methods.

3.2 Weak Scaling

In weak scaling the problem size (work load) per processor is kept constant as the number of processors is scaled. Ideally the time per work unit should stay constant throughout the scaling. The parallel efficiency is calculated as,

$$\text{Eff}_{\text{weak}} = \frac{t_1}{t_P} \tag{4}$$

The parallel efficiency for the weak scaling study is shown in Fig. 5 for both methods. A constant workload of approximately 156,000 work units per cycle is used. The vertex ghost data method shows improved scaling over the ghost particle approach due to the additional memory operations from the ghost particles. At 1,024 processors the efficiency for the vertex ghost data method is 0.89 while the ghost particle method has an efficiency of 0.31. The efficiency of the vertex method starts to fall off past 2048 processors, but still has a value of 0.75 at 16,384 processors. At 16,384 processors there are approximately 2.3 billion particles in the problem.

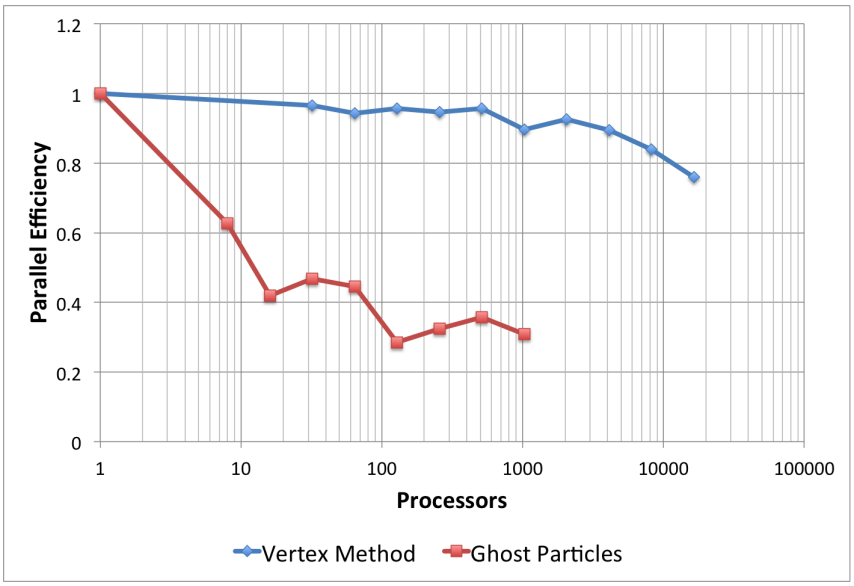
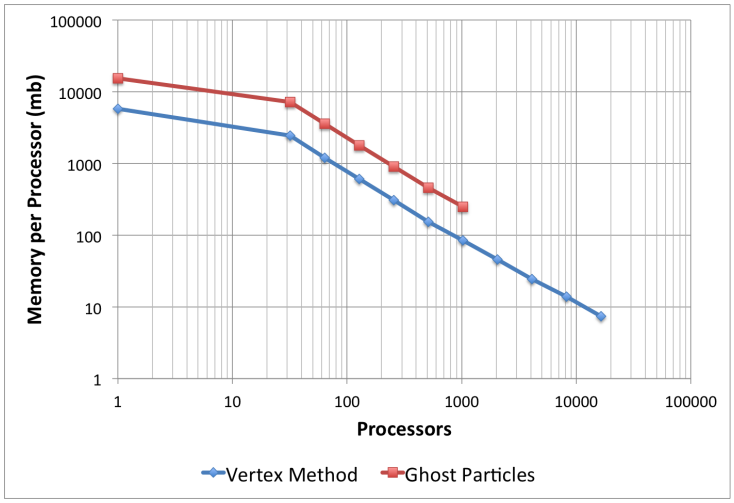


Figure 5: Weak scaling study showing parallel efficiency for ghost particle and vertex ghost data methods.

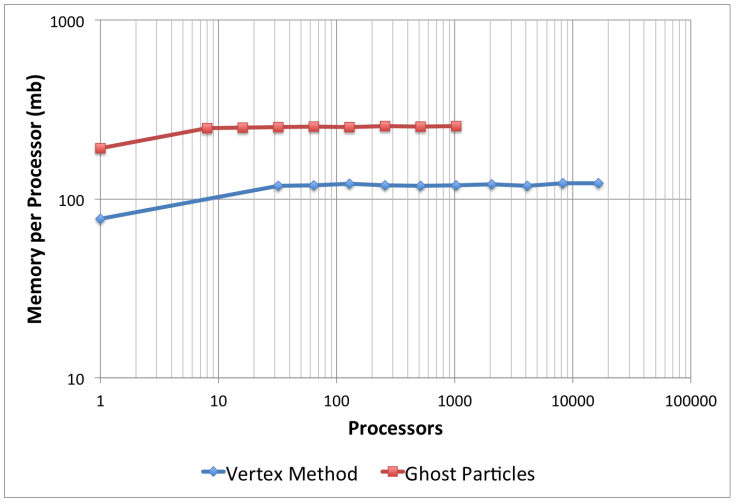
3.3 Memory Scaling

The memory scaling is also examined for both methods in the strong and weak scaling studies. The memory reported does not include any MPI buffers or other system memory, only the initial memory allocated for the computational mesh and particles. The code avoids any large dynamic memory allocation for efficiency, so the memory load is approximately constant throughout the simulation. Ideally the memory per processor should decrease linearly with increasing processors for the strong scaling study, and should remain approximately constant during the weak scaling study.

The memory scaling is shown in Fig. 6. Both the ghost particle and vertex ghost data methods show the expected memory scaling behavior past a single processor. The change in slope past a single processor is due to additional data arrays which are allocated to keep track of particles being moved to new processors and scratch storage for computing sums in ghost cells. For all the cases the ghost particle method required more memory than the vertex ghost data approach due to the additional ghost markers created, but both methods scaled as expected beyond a single processor.



(a)



(b)

Figure 6: Total memory scaling for ghost particle and vertex ghost data methods for (a) strong scaling, and (b) weak scaling.

4 CONCLUSIONS

Two parallelization approaches for the MPM were presented and implemented. Both approaches strong and weak scaling characteristics were examined, along with their memory usage. The vertex ghost data method showed superior strong and weak scaling characteristics compared to the ghost particle approach, and also had reduced memory usage. The results show that the MPM is able to be parallelized and run efficiently at processor counts up to 16,384 for problems with billions of degrees of freedom. In the future the scaling characteristics of the vertex ghost data method will be examined in conjunction with the AMR in CTH, along with ways to improve the scalability past 16,384 processors.

ACKNOWLEDGEMENTS

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

REFERENCES

[1] Sulsky, Deborah, Chen Z. and Schreyer, Howard L., "A Particle Method for History-Dependent Materials," *Computational Methods Applied Mechanical Engineering*, Vol. 118, pp. 179–196 (1994).

[2] McGlaun, J. M., F. J. Zeigler and S. L. Thompson, CTH: A Three-Dimensional, Large Deformation, Shock Wave Physics Code, APS Topical Conference on Shock Waves in Condensed Matter, Monterey, CA, July 20-23, (1987).

[3] Sulsky, Deborah, and Howard L. Schreyer. "Axisymmetric form of the material point method with applications to upsetting and Taylor impact problems." Computer Methods in Applied Mechanics and Engineering 139, pp. 409–429 (1996).

[4] Guilkey, James E., Todd Harman, Amy Xia, Bryan Kashiwa, and Patrick McMurtry. "An Eulerian-Lagrangian approach for large deformation fluid structure interaction problems, Part 1: algorithm development." Advances in Fluid Mechanics 36, pp. 143–156 (2003).

[5] Harman, Todd, James E. Guilkey, Bryan Kashiwa, John Schmidt, and Patrick McMurtry. "An Eulerian-Lagrangian approach for large deformation fluid structure interaction problems, Part 2: Multi-physics simulations within a modern computational framework." Advances in Fluid Mechanics 36, pp. 157–166 (2003).

[6] Guilkey, J. E., T. B. Harman, and B. Banerjee. "An EulerianLagrangian approach for simulating explosions of energetic devices." Computers & structures 85, pp. 660–674 (2007).