A massively parallel implementation of the Optimal Transportation Meshfree method for explicit solid dynamics

B. Li¹,³, M. Stalzer² and M. Ortiz¹,*,†

¹Graduate Aerospace Laboratories, California Institute of Technology, Pasadena, CA 91125, USA
²Center for Advanced Computing Research, California Institute of Technology, Pasadena, CA 91125, USA
³Department of Mechanical and Aerospace Engineering, Case Western Reserve University, Cleveland, OH 44106, USA

SUMMARY

Presented is a massively parallel implementation of the Optimal Transportation Meshfree (pOTM) method Li et al., 2010 for explicit solid dynamics. Its implementation is based on a two-level scheme using Message Passing Interface between compute servers and threaded parallelism on the multi-core processors within each server. Both layers dynamically subdivide the problem to provide excellent parallel scalability. pOTM is used on three problems and compared to experiments to demonstrate accuracy and performance. For both a Taylor-anvil and a hypervelocity impact problem, the pOTM implementation scales nearly perfectly to about 8000 cores. Copyright © 2014 John Wiley & Sons, Ltd.

Received 8 October 2013; Revised 8 May 2014; Accepted 12 May 2014

KEY WORDS: optimal transportation meshfree method; meshfree methods; parallel computing; dynamic load balancing; explicit solid dynamics

1. INTRODUCTION

The Optimal Transportation Meshfree (OTM) method [1] is a meshfree updated Lagrangian methodology, which combines concepts from Optimal Transportation theory with material-point sampling and local max-ent meshfree approximation, and overcomes the essential difficulties in grid-based numerical methods like Lagrangian and Eulerian finite element methods. The rationale behind the approach is as follows. We resort to the Benamou–Brenier [2] differential formulation of optimal mass transportation problems and its connection to the Wasserstein distance [3] to discretize the inertial action in space and time within a strictly variational framework. The resulting discretization may be regarded as the result of restricting the inertial action to mass measures concentrated on material points undergoing piecewise rectilinear motions. The density of such mass measures and the constrained minimization structure of the problem may be expected to confer the discretization robust convergence properties. The optimal transportation variational framework also results in proper mass matrices and inertia forces in the presence of continuously varying spatial interpolation, geometrically exact mass transport and satisfaction of the continuity equation, and exact linear and angular momentum conservation. Finally, fields requiring differentiation, such as deformation and velocity fields, are interpolated from nodal values using max-ent shape functions. These shape functions are reconstructed continuously from the nodal set and have the key property of possessing a Kronecker-delta property at the boundary, which enables the direct imposition of displacement boundary conditions. An energy-based material-point erosion algorithm is also applied for

*Correspondence to: M. Ortiz, Graduate Aerospace Laboratories and Center for Advanced Computing Research, California Institute of Technology, Pasadena, CA 91125, USA.
†E-mail: ortiz@aero.caltech.edu

Copyright © 2014 John Wiley & Sons, Ltd.
simulating fracture and fragmentation phenomena. The convergence of the material-point erosion algorithm to Griffith-fracture solutions is ensured by means of $\Gamma$-convergence [4].

The main contribution of this article is a massively parallel implementation of the OTM method called pOTM. As described previously, there are two closely related discretizations of the objects: a set of material points and a set of nodes. In pOTM, these are divided into a subset for each compute server and Message Passing Interface (MPI) is used to communicate between the servers. This division, or parallel decomposition, is carried out in a way to keep each set physically compact and so many calculations can be carried out locally on a server. For the interactions between neighboring subsets, we developed a shadow point/node scheme that is analogous to the well known ghost cell technique [5]. For some problems, a static domain decomposition is sufficient to maintain good parallel balance. However, with highly deforming objects, a dynamic load balancing scheme is required to maintain balance.

Using MPI between cores on a single server is not optimal in terms of storage and overhead [6]. For on-server parallelism, we shift to a threaded model where the computations of the point/node subsets on each server are dynamically allocated to threads. The full pOTM parallelism model is then a dynamically load balanced MPI-based layer between servers with a dynamic thread layer on each server. This makes effective use of large clusters of many-core processors as will be shown later.

The performance and accuracy of pOTM is illustrated by comparing with in-house experiments. The first is a Taylor-anvil impact test where an aluminum alloy rod is impacted axially against a rigid boundary, which causes rod deformation at the impact site. The second is a terminal ballistics problem consisting of a stainless steel sphere impacting an aluminum plate: the sphere velocity, materials, and plate thickness are such that the sphere perforates the plate. The final problem is similar but using a cylindrical nylon projectile at higher (5.8 km/s) velocities. Each of these experiments illustrates different aspects of pOTM.

The outline of the article is as follows. Section 2 contains a brief description of (sequential) OTM and the material-point erosion algorithm. Section 3 describes pOTM at both the MPI and thread levels of parallelism. Results from applying pOTM to the three problems are presented in Section 4.

2. METHODOLOGY

2.1. The Optimal Transportation Meshfree method

We begin by briefly summarizing the OTM method of spatial and temporal discretization. A class of semi-discrete actions that is well-suited to computation is

$$S_d(\varphi_1, \ldots, \varphi_{N-1}) = \sum_{k=0}^{N-1} \left\{ \frac{1}{2} d_W^2(\rho_k, \rho_{k+1}) - \frac{1}{2} [U(\varphi_k) + U(\varphi_{k+1})] \right\} (t_{k+1} - t_k)$$

(1)

where $\varphi_k : B \subset \mathbb{R}^n \rightarrow \mathbb{R}^n$ is the deformation mapping at time $t_k$, $\rho_k$ is the corresponding mass density at time $t_k$,

$$U(\varphi) = \int_B f(\nabla \varphi) \, dx$$

(2)

is the free energy of the solid, $f(\nabla \varphi)$ is the local free-energy density per unit volume, and

$$d_W^2(\rho_a, \rho_b) = \inf_{T : B \rightarrow B} \int_{T=\text{det}(V)} |T(x) - x|^2 \rho_a(x) \, dx$$

(3)
is the Wasserstein distance between consecutive densities. The term $\frac{1}{2} \sum\nolimits_{k} d^2_W(\rho_k, \rho_{k+1})(t_{k+1} - t_k)$ in (1) supplies a measure of the inertial action between times $t_k$ and $t_{k+1}$. We also note that in writing (1), we have restricted attention to elastic behavior and unforced systems for simplicity. Extensions accounting for forcing, for example, in the form of body forces, boundary traction, and extensions to inelasticity may be found in [1].

In order to obtain a fully discrete action for computations, we begin by approximating the usual Lebesgue measure $\mathcal{L}$ of the volume at $t_k$ by discrete measures of the form

$$\mathcal{L}_{k,h} = \sum_{p=1}^{M} v_{p,k} \delta(x - x_{p,k}), \quad (4)$$

concentrated at material points $x_{p,k}$, each of which is assigned a discrete volume $v_{p,k}$, Figure 1. The discretization of the mass densities $\rho_{k,h}$ may be achieved simply by identifying the discrete mass distributions as measures that are absolutely continuous with respect to the discrete volume measure $\mathcal{L}_{h,k}$, with Radon–Nykodim density $\rho_{h,k}$, namely,

$$\rho_{h,k}(x) = \sum_{p=1}^{M} \rho_{p,k} v_{p,k} \delta(x - x_{p,k}). \quad (5)$$

The quantity $m_p = \rho_{p,k} v_{p,k}$ may be regarded as the mass carried by material-point $p$. Li et al. [1] have shown that the constancy of the material-point masses $m_p$ is indeed equivalent to the weak satisfaction of the continuity equation. To complete the spatial discretization, we approximate the incremental deformation mapping as

$$\varphi_{h,k \rightarrow k+1}(x) = \sum_{a=1}^{N} x_{a,k+1} N_{a,k}(x), \quad (6)$$

Figure 1. Spatial discretization scheme used in the Optimal Transportation Meshfree method. The figure describes one step of the application of the method between times $t_k$ and $t_{k+1}$. The open circles represent the nodal points and the red circles the material points. $B_k$ and $B_{k+1}$ are the configurations of the body at times $t_k$ and $t_{k+1}$, respectively, and $\varphi_{k \rightarrow k+1}$ is the corresponding incremental deformation mapping. $N_{p,k}$ is the local neighborhood of material point $x_{p,k}$ that supports the local max-ent meshfree interpolation.
where \( \{x_{a,k+1}, a = 1, \ldots, N\} \) is an array of nodal coordinates at time \( t_{k+1} \) and \( N_{a,k}(x) \) are conforming shape functions defined over the configuration at time \( t_k \). In calculations, we specifically use max-ent shape functions [7] computed from the array \( \{x_{a,k}, a = 1, \ldots, N\} \) of nodal coordinates at time \( t_k \). Because max-ent shape functions are strongly localized, the interpolation at a material-point \( x_{p,k} \) depends solely on the nodes contained in a small local neighborhood \( N_{p,k} \) of the material point, Figure 1. In calculations, the local neighborhoods are continuously updated using range searches [8] to account for the relative motion between material points and nodes. The volume that encloses all of the ranges for a collection of points is important for the parallel implementation as described in the next section.

Inserting these approximations into (1), we obtain the fully discrete action

\[
S_h(\varphi_{h,1}, \ldots, \varphi_{h,N-1}) = \sum_{k=0}^{N-1} \sum_{p=1}^{M} \left\{ m_p \frac{|x_{p,k+1} - x_{p,k}|^2}{(t_{k+1} - t_k)^2} \right\}
- \frac{1}{2} \left[ \sum_{p=1}^{M} m_p f(\nabla \varphi_{h,k}(x_{p,k})) + \sum_{p=1}^{M} m_p f(\nabla \varphi_{h,k+1}(x_{p,k+1})) \right]\left(t_{k+1} - t_k\right),
\]

where we again consider the unforced elastic case for simplicity. The discrete trajectories now follow from the discrete Hamilton’s principle

\[
\delta S_h = 0
\]

de of stationary action.

**Algorithm 1 OTM time step**

| Require | Initial and final times for time step, \( t_k, t_{k+1} \). |
|---------------------------|
| Require | Initial nodal coordinates, \( x_k = \{x_{a,k}, a = 1, \ldots, N\} \). |
| Require | Initial material-point coordinates, \( \{x_{p,k}, p = 1, \ldots, M\} \). |
| Require | Initial shape functions \( \{N_{a,k}(x_{p,k}), a \in N_{p,k}, p = 1, \ldots, M\} \). |
| Require | Ibid gradients \( \{\nabla N_{a,k}(x_{p,k}), a \in N_{p,k}, p = 1, \ldots, M\} \). |
| Require | Initial deformation gradients, \( \{F_{p,k}, p = 1, \ldots, M\} \). |
| Require | Initial material state at all material points. |

1: Compute mass matrix \( M_k \), nodal forces \( f_k \), accelerations \( a_k = M_k^{-1} f_k \).
2: Update nodal coordinates: \( x_{k+1} = x_k + (t_{k+1} - t_k)(v_k + \frac{1}{2}(t_{k+1} - t_{k-1})a_k) \).
3: Update nodal velocities: \( v_{k+1} = (x_{k+1} - x_k)/(t_{k+1} - t_k) \).
4: Update material-point coordinates: \( x_{p,k+1} = \varphi_{k\rightarrow k+1}(x_{p,k}), p = 1, \ldots, M \).
5: Update deformation gradients: \( F_{p,k+1} = \nabla \varphi_{k\rightarrow k+1}(x_{p,k})F_{p,k}, p = 1, \ldots, M \).
6: Effect constitutive updates at material points.
7: Update local material-point neighborhoods \( N_{p,k+1} \), \( p = 1, \ldots, M \).
8: Compute shape functions \( \{N_{a,k+1}(x_{p,k+1}), a \in N_{p,k+1}, p = 1, \ldots, M\} \).
9: Ibid gradients \( \{\nabla N_{a,k+1}(x_{p,k+1}), a \in N_{p,k+1}, p = 1, \ldots, M\} \).

The algorithm resulting from the preceding scheme is listed in Algorithm 1. We see from Figure 1 that the OTM scheme can be solved forward explicitly. This forward solution has the usual structure of explicit time-integration and updated Lagrangian schemes. In particular, all the finite kinematics of the motion, including the mass density and volume updates, are geometrically exact. In addition, the continuous reconstruction of the local material-point neighborhoods and shape functions has the effect of automatically reconnecting the material points and the nodal set, at no cost of remapping the local states carried by the material points. This property of the method is particularly convenient for inelastic materials whose local material state often includes additional internal variable information. A particularly convenient feature of OTM, which is common to other material-point based methods [9], is that seizing contact is automatically accounted for. This is so because
of the cancelation of linear momentum that naturally occurs when colliding nodes come within the local neighborhoods of material points, Figure 2. Thus, the linear momentum of node $a$ follows as [1]

$$l_{a,k} = \sum_{p=1}^{M} m_p \frac{x_{p,k} - x_{p,k-1}}{t_k - t_{k-1}} N_{a,k}(x_{p,k}).$$

(9)

2.2. The variational material-point erosion algorithm

Finally, we briefly outline the material-point erosion algorithm proposed to simulate the fracture and fragmentation phenomena in terminal ballistics calculations [10]. Within the context of OTM calculations, fracture can be modeled simply by eroding or failing material points according to an energy-release criterion. When the material points are eliminated from the model, gaps are left in the solid that approximate the presence of cracks. Following Ortiz et al. [11] and Fraternali et al. [4], we compute the energy-release rate attendant to the removal of material-point $p$ as

$$G_{p,k+1} = \frac{C\epsilon}{m_{p,k+1}} \sum_{q,k+1 \in B_{\epsilon}(x_{p,k+1})} m_q f_k(F_{q,k+1}).$$

(10)

where $B_{\epsilon}(x_{p,k+1})$ is the ball of radius $\epsilon$ centered at $x_{p,k+1}$, or $\epsilon$-neighborhood of the material point (cf. Figure 3),

$$m_{p,k+1} = \sum_{q,k+1 \in B_{\epsilon}(x_{p,k+1})} m_q$$

(11)

is the mass of the $\epsilon$-neighborhood and $C$ is a normalizing constant.
The averaging radius $\epsilon$ defines a length scale intermediate between the discretization size and the macroscopic size of the bodies. The material point is failed when

$$G_{p,k+1} \geq G_c,$$

where $G_c$ is a critical energy-release rate that measures the material-specific energy required to create a fracture surface of unit area. For linear elasticity, Fraternali et al. [4] have shown that criteria (10) and (12) result in approximations that converge to Griffith fracture in the limit of an infinitely fine discretization.

3. PARALLELIZATION

The OTM method is an incremental updated Lagrangian meshfree method, and considerable savings on computational time can be realized with the successful utilization of parallel computers. In this section, we describe the hybrid multiprocessing and multithreading implementation of the OTM method at the levels of distributed memory and shared memory systems, respectively.

3.1. The shadow scheme for distributed memory parallelism

Similar to explicit finite element codes for the simulation of highly nonlinear dynamic problems, the majority of workload of the OTM method is in the determination of internal nodal forces at each time step (Section 2, Algorithm 1). Among which, most of the computational time is spent on the evaluation of the material response at the material points. The reconstruction of the neighborhood of each material point followed by the recalculation of their shape functions and derivatives is another time-consuming process. Nevertheless, all these main operations are performed at every material point locally. In addition, material points in the OTM method do not communicate to each other directly. Therefore, the parallelization is made with regard to the determination of the nodal forces and reconstruction of the neighborhood of material points. The domain decomposition for the distributed memory parallelism is performed by distributing material points across processors, and only those nodes, which are in their neighborhoods, are carried on the same processor. A variety of domain decomposition algorithms that are based on graph theory or geometric techniques exist [12, 13]. In the OTM method, the initial domain of analysis is set up by triangulation with material points placed at the barycenters of the tetrahedral elements. The initial mesh defines a graph with material points as vertices and element interfaces as edges. Hence, a graph-based partitioner is employed.
to compute the initial static decomposition. In particular, open source codes like Metis [12] and Jostle [13] are available and generally efficient for this purpose. This is carried out once at problem initialization, the initial mesh is then jettisoned and the calculations subsequently proceed in a mesh-free manner. Nevertheless, a graph-based domain decomposition is not required. In applications where the geometry of the spatial domain is regular, a Cartesian decomposition can be defined in such a way that each process has (approximately) the same number of material points and minimal communicating regions.

Because each processor carries a subset of material points and nodes, the information of nodes that belong to the neighborhood of material points on multiple processors needs to be synchronized through the network connections using MPI. In addition, the amount of communication changes dynamically in the OTM method as the material points are transported by the flow and their supports are reconstructed ‘on the fly’. As a result, the amount of computation will vary among material points as well. To this end, we developed a shadow scheme to dynamically construct the

Figure 4. Shadow scheme for the construction of communication map among distributed partitions.

Figure 5. Synchronization of the range boxes information where \( L_i \) and \( U_i \) are the coordinates of the lower and upper corner of the range box defined for the \( i \)th partition, respectively.

Figure 6. Construction of the communication map where \( P_i \) and \( N_i \) are the id of the destination and the number of nodes sending to processor \( P_i \), respectively.
communication map between distributed processors. Figure 4 shows the MPI parallelization of the OTM method schematically. In this approach, a range box is defined for each subdomain. For simplicity, a cuboid is chosen as the shape of the range box. Thus, the coordinates of the lower and upper corners of the range boxes are recalculated at every iteration based on the current location of the material points as well as their support size. The lower and upper bounds of the range boxes of all the partitions are then gathered at every processor, as shown in Figure 5. Shared nodes between partitions are identified as the nodes located in the intersections of the range boxes of the partitions. Based on the lower and upper bounds, the number of shared nodes sending to and receiving from other processors, that is, the communication map, can be calculated locally at each MPI process. Similarly, the local communication map of all the partitions is then gathered at every processor shown in Figure 6.

Before the synchronization step, nodal forces are computed at each MPI process as if they were independent analysis. As the communication map is constructed via an MPI_Allgather, the required nodal information including coordinates, velocity, acceleration, and mass is sent to and received from other processors by asynchronous send/receive (MPI_Isend and MPI_Irecv) to avoid possible deadlocks and overcreating global force vectors. An MPI data structure is defined by calling MPI_Type_struct to pack the nodal information into a serial array. The code segment of the construction of the data type is shown in the Listing 1. The size of the message for a single node is maintained as 10 × sizeof(double precision floating number) + sizeof(integer) in three-dimensional analysis, for instance 84 bytes on a 64-bit system.

```c
MPI_Datatype node_type;
int block_lengths[2];
MPI_Aint offsets[2];
MPI_Datatype typelist[2];

// First specify the types
typelist[0] = MPI_INT;
.typelist[1] = MPI_DOUBLE;

// Specify the number of elements of each type
block_lengths[0] = 1; // node id
block_lengths[1] = 10; // mass, coordinates, velocity and acceleration

// Calculate the displacements of the members
offsets[0] = 0;
MPI_Aint extent;
MPI_Type_extent(MPI_INT, &extent);
offsets[1] = block_lengths[0] * extent;

// Create the derived type
MPI_Type_struct(2, block_lengths, offsets, typelist, &node_type);
```

Listing 1. Code segment of the MPI data structure for a single node

Finally, the nodal forces and kinematic information of shared nodes are corrected at each partition locally. The algorithm resulting from the preceding scheme is listed in Algorithm 2. One of the advantage to this computational structure is that an original serial code can be made to run in parallel with slight modifications, basically adding a synchronization step after each iteration, as shown in the flowchart Figure 7. In addition, because the neighbor search and dynamic contact is performed at each material point, and only those nodes on the local processor are accounted for by the search algorithm, it automatically results in a parallel neighbor search and parallel dynamic contact scheme.

### 3.2. Pthreads with dynamic load balancing for shared memory parallelism

Nevertheless, the scalability of a non-embarrassingly MPI parallelized program to a very large number of processors is cursed by the MPI overheads as well as load imbalance between distributed processors. In this paper, the majority of the work load of each MPI process is further parallelized using the shared memory multithreading parallelism to overcome this barrier. We develop
Algorithm 2 Parallel OTM time step

For process $P^I$, $I = 1, \ldots, P$:

**Require:** Initial and final times for time step, $t_k$, $t_{k+1}$.

**Require:** Initial nodal set, $S^I_k = \{x_{a,k} \in P^I, \ a = 1, \ldots, N\}$ at $t_k$.

**Require:** Initial material-point set, $\{x_{p,k}^I, \ p = 1, \ldots, M\}$ at $t_k$.

**Require:** Initial neighborhood of material points, $N_{p,k} = \{x_{a,k}, \ ||x_{a,k} - x_{p,k}^I|| \leq d_{p,k}\}$, where $d_{p,k}$ is the dynamic support size of material point $x_{p,k}^I$, and condition $\bigcup_p N_{p,k} \subseteq S^I_k$ must be satisfied.

**Require:** Initial shape functions $\{N_{a,k}(x_{p,k}^I), \ a \in N_{p,k}\}, \ p = 1, \ldots, M$.

**Require:** Ibid gradients $\{\nabla N_{a,k}(x_{p,k}^I), \ a \in N_{p,k}\}, \ p = 1, \ldots, M$.

**Require:** Initial deformation gradients, $\{F_{p,k}, \ p = 1, \ldots, M\}$.

**Require:** Initial material state at all material points on $P^I$.

**Require:** Initial communication map, $C_k$.

1. Compute local mass matrix $M^I_k$ and local nodal forces vector $F^I_k$.
2. Compute the local acceleration: $\ddot{x}^I_k = (M^I_k)^{-1} F^I_k$.
3. Asynchronous send/receive information of shared nodes based on $C_k$.
4. Correct mass matrix $M^I_k$: for shared nodes $x_{a,k}, m_{a,k} = \bar{m}_{a,k} + \sum_{\gamma \in C_k} \bar{m}_{\gamma,k}^Q$.
5. Correct nodal forces $F^I_k$: for shared nodes $x_{a,k}, f_a = \bar{f}_{a,k} + \sum_{\gamma \in C_k} \bar{f}_{\gamma,a,k}^Q \bar{m}_{\gamma,k}^Q$.
6. Compute accelerations $\ddot{a}^I_k = (M^{-1}_k)^{-1} F^I_k$.
7. Update nodal coordinates: $x^I_{k+1} = x^I_k + (t_{k+1} - t_k) \left( v^I_k + \frac{1}{2}(t_{k+1} - t_k) \ddot{a}^I_k \right)$.
8. Update nodal velocities: $v^I_{k+1} = (x^I_{k+1} - x^I_k) / (t_{k+1} - t_k)$.
9. Update material-point coordinates: $x^I_{p,k+1} = \varphi_{k \rightarrow k+1} \left( x^I_{p,k} \right), \ p = 1, \ldots, M$.
10. Update deformation gradients: $F_{p,k+1} = \nabla \varphi_{k \rightarrow k+1} \left( x^I_{p,k} \right) F_{p,k}, \ p = 1, \ldots, M$.
11. Effect constitutive updates at material points.
12. Update local material-point neighborhoods $N_{p,k+1}, \ p = 1, \ldots, M$.
13. Compute shape functions $\{N_{a,k+1}(x^I_{p,k+1}), \ a \in N_{p,k+1}\}, \ p = 1, \ldots, M$.
14. Ibid gradients $\{\nabla N_{a,k+1}(x^I_{p,k+1}), \ a \in N_{p,k+1}\}, \ p = 1, \ldots, M$.
15. Update range box of $P^I$ and gather information of the range boxes from all other processes.
16. Update communication map $C_{k+1}$.

A hierarchical model: MPI parallelization occurring at the top level, and POSIX threads (Pthreads) parallelization occurring below, shown in Figure 8. In the shared memory model, all threads of a MPI process have access to the same global memory space, that is, there is only one copy of the collection of material points and nodes. Therefore, no communication is needed for the synchronization of nodal information between threads, which gives a perfect situation to speed up the program ideally.

In addition, the workload distribution is carried out via both functional and data decomposition. In the functional decomposition, threads are assigned different tasks to perform in parallel. Especially, we use threads to overlap communications with calculations. When the nodal coordinates are updated in Algorithm 2 step (7), we invoke a thread to update the range boxes and reconstruct the communication map in Algorithm 2 steps (15) and (16). The other concurrent threads are assigned to finish the rest of the steps in Algorithm 2 in parallel. It is worthy to mention that in our implementation, we employ the fork, join, and quit model. A thread pool is initialized at the beginning of the computation, which takes charge of the scheduling, execution, and collection of the threads, and destroyed at the exit of the simulation. The use of a thread pool management avoids the cost of creation and destruction of threads in our incremental updated Lagrangian approach, which may result in better performance. A schematic representation of the model is shown in Figure 9.
Figure 7. Flowchart of one iteration in the multiprocessing parallelization of the Optimal Transportation Meshfree method.

Figure 8. The hierarchical model of the hybrid parallelism. A collection of material points in 2D domain has been divided geometrically between two Message Passing Interface (MPI) processes $P_1$ and $P_2$. These subsets of material points have been further divided between four threads $T_1$, $T_2$, $T_3$ and $T_4$ randomly for each process. Note that the subdivision of material points to threads follows the order of the data stored in the computer memory rather than their geometrical locations in space.

Similar to the MPI parallelization, the data decomposition in the Pthreads parallelization is performed on the collection of material points stored in the local memory space. While no synchronization is needed in the shared memory scheme, we may randomly decompose the collection of material points into small groups and select available threads from the thread pool to execute on the subset. Nevertheless, a thread load imbalance occurs exclusively if the data are itself unbalanced either by its quantity or its processing requirements. In this paper, we demonstrate an incremental dynamic load balancing scheme to address unequal thread workloads.
Figure 9. Schematic representation of the functional decomposition and thread pool management, where workers are working threads and communicators are communication threads.

The dynamic load balance is realized by dynamic domain decomposition at each iteration. Therefore, subsets of material points requiring equal processing time are created dynamically and assigned to the available threads. Suppose the number of threads is fixed and a thread $T_i$ for $i = 1, 2, \ldots, N$ is forked for the computation of material points in a subset $M_{i,k}^I$ at time $t_k$, where $M^I = \bigcup M_{i,k}^I$ is the total material points in the MPI process $\Pi^I$ and $M_{i,k}^I \cap M_{j,k}^I = \emptyset$ for $i \neq j$. Let $C(x_{p,k})$ be a measurement of the computational cost of the material point $x_{p,k}$ at $t_k$, the total computational cost assigned to thread $T_i$ at $t_k$ is given by

$$C_k(T_i) = \sum_{x_{p,k} \in M_{i,k}^I} C(x_{p,k}), \quad (13)$$

and the total computational cost of process $\Pi^I$ at $t_k$ can be calculated as,

$$C_k(\Pi^I) = \sum_{x_{p,k} \in M^I} C(x_{p,k}) = \sum_{i=1}^{N} C_k(T_i). \quad (14)$$

In addition, we assume the computational cost for each material point varies slightly from $t_k$ to $t_{k+1}$. Thus, at time $t_{k+1}$, we can reconstruct the subsets $M_{i,k+1}^I \subset M^I$ based on the total computational cost of $\Pi^I$ at $t_k$, such that

$$C_{k+1}(T_i) = \sum_{x_{p,k+1} \in M_{i,k+1}^I} C(x_{p,k+1}) = C_k(\Pi^I) / N \quad (15)$$
The dynamic domain decomposition does not require any data migration in the physical memory space as long as the pointers or indices of the material points are grouped in a way that satisfies Equation (15). Therefore, the extra cost of the incremental dynamic load balancing algorithm is negligible.

Nevertheless, multiple threads may access the same memory space simultaneously, which can lead to unpredictable behaviors. One solution is using mutex locks whenever a modification operation involves shared memory. However, mutex locks may cause another problem of thread competition and seriously affect the parallel performance of the program. In the Pthreads parallelization of the OTM method, it happens when a global nodal force matrix is assembled. In our implementation, we allocate a temporary memory space to store the nodal forces for each thread and assemble the global nodal force matrix in the main thread, which may totally remove the locks from the code.

4. NUMERICAL EXPERIMENTS

In this section, we aim to assess the parallel approach and demonstrate its scalability performance under a broad range of conditions. Three problems are solved using the pOTM method, namely, a concurrent multiscale computation of ductile failure in metals, a ballistic impact problem of 440C stainless steel spherical projectile on aluminum alloy 6061-T6 plate in the velocity range of 2–3 km/s, and a hypervelocity impact problem of Nylon 6/6 cylindrical projectile on aluminum alloy 6061-T6 plate in the velocity range of 5–7 km/s.

4.1. Taylor-anvil impact

The first example is a typical Taylor bar impact test designed to validate the micromechanical constitutive model of internal damage formation by void growth within ductile materials at high strain rate [14]. A three-dimensional right circular rod of aluminum alloy 1100-0 impacting axially against a rigid boundary at velocity between 100 and 300 m/s is simulated using the MPI parallelization of OTM described in Section 3.1. The rod has an initial radius of 6.295 mm and a length of 25.71 mm.

Figure 10. Initial configuration of the 15° slice of an aluminum alloy 1100-0 rod: 8599 nodes and 30,448 material points. Each material point represents a hollow sphere in the microscale which is discretized by 120 degrees of freedom and 270 elements. (a) nodal set; (b) material-point set and the representative volume element.
Because of the axisymmetric deformation, only a $15^\circ$ slice of the cylinder was simulated, as shown in Figure 10. As a concurrent FE$^2$ calculation, two levels of discretization in the macro and micro scale are performed for the initialization of the model, respectively. The initial domain of analysis in the macroscale is set up by triangulation with the material points placed at the barycenters of the tetrahedral elements. However, the initial mesh is subsequently jettisoned, and the calculations proceed in a meshfree manner. The model contains 8599 degrees of freedom and 30,448 material points for the $15^\circ$ slice at the continuum level, shown in Figure 10(a and b). In the microscopic scale, each material point represents a single hollow sphere [14]. Therefore, the material response at a material point is calculated by solving a boundary value problem using the Finite Element Method. In our computations, each hollow sphere is modeled with number of layers in the radial direction $N_r = 9$ and the order of the expansion in the spherical harmonics $N_l = 1$, which results in 120 degrees of freedom and 270 elements, Figure 10(b). Hence, the proposed discretization leads to totally 3.09 million degrees of freedom and 8.22 million elements for the entire domain. In our calculations, the set of material points at the continuum level is sorted according to their $z$-coordinates, that is, coordinates along the axis of the cylinder. The domain decomposition is performed by distributing an equal number of material points across processors in order from the sorted material-point set. The initial void volume fraction is taken as 3.7%. The $J_2$-viscoplasticity models with power-law hardening, rate-sensitivity and thermal softening (cf., e.g., [15, 16]) are used for the matrix of the hollow sphere. The material parameters have been chosen according to the related literature [17] and collected in Tables I and II. Figure 11 shows a sequence of snapshots of the collision at times $t = 0.5$ and $3 \mu s$. The final distribution of the void volume fraction agrees well with the experimental results presented in [18]. Similar to conventional FE$^2$ approaches, the main computational cost of the simulation is to

<table>
<thead>
<tr>
<th>Table I. Mechanical material constants.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$ (kg/m$^3$)</td>
</tr>
<tr>
<td>2700</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table II. Thermal constants.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c$ (J/kgK)</td>
</tr>
<tr>
<td>904</td>
</tr>
</tbody>
</table>

Figure 11. Calculation of Taylor impact of a 1100-0 aluminum cylinder at 300 m/s. The left half of the bar shows the evolution of the stress wave, the right half shows the distribution of the void volume fraction in a sequence. (a) $t = 0.5 \mu s$; and (b) $t = 3.0 \mu s$. Copyright © 2014 John Wiley & Sons, Ltd. Int. J. Numer. Meth. Engng 2014; 100:40–61 DOI: 10.1002/nme
determine the macroscopic material response from the representative volume elements by solving a boundary value problem at each material point using the finite element method. The communication cost between distributed processors is due to the synchronization of nodal information at the macroscale. In a typical OTM calculation, each material point has less than $O(10^2)$ nodes as its support, and that is less than 8 Kbytes nodal data in a three-dimensional problem. Therefore, the communication cost of a single material point is trivial comparing to its computational cost, which is a proper scenario for obtaining the ideal scalability using the MPI parallelization. To this end, a weak scaling performance analysis of the concurrent multiscale OTM computation is performed on Hera at Lawrence Livermore National Laboratory (LLNL). Hera is a large capacity computing resource with 864 nodes connected by InfiniBand DDR and totally 13,824 AMD Opteron 8356 cores. The theoretical system peak performance is about 127.2TFlop/s. Figure 12 shows the parallel performance for our analyses using different number of cores with four material points per core. The speedup is significant for this moderately sized problem and nearly perfect up to about 8000 cores. In addition, the code runs at about 5% sustained and 8% peak of the theoretical Flops rate. This rate may be improved by additional tuning for the wide vector pipelines of the cores. It is evident that ideal scalability may be obtained from the MPI version of the pOTM method in the case that the computational cost of each partition is much heavier than the communications on distributed memory systems.

4.2. High velocity impact

Next, we investigate the robustness and scalability performance of the multithreading parallelization of the pOTM method on a shared memory system described in Section 3.2. To this end, a terminal ballistic impact problem of 440C stainless steel spherical projectiles on aluminum alloy 6061-T6 plates is solved by the pOTM method. The fidelity of the model is quantitatively assessed in an extended data-on-demand uncertainty quantification (UQ) protocol based on concentration-of-measure inequalities and martingale theory [19]. Driven by the UQ protocol, thousands of three-dimensional pOTM simulations of terminal ballistics are conducted for plates with dimensions $152.4 \times 152.4$ mm, thicknesses in the range of 1.6–3.2 mm, and impact velocity in the range of 2–3 km/s in the normal direction. The radius of the spherical projectile is 1.778 mm. A typical discretization of half of the domain of analysis is shown in Figure 13. As may be seen from the figure, the distribution of nodes and material points is highly adaptive – dense in the region of impact and decreased away from it – in order to provide the highest possible resolution of the penetration and perforation process with a minimum of degrees of freedom. Both the plate and projectile are modeled by $J_2$-viscoplasticity models with power-law hardening, rate-sensitivity, and thermal softening (cf., e.g., [15, 16]). In addition, fracture is simulated by the variational material-point failure algorithm. A detailed description of the material models and the failure algorithm may be found in [20] and [10]. Tables III–V list the material parameters used in our calculations.
Figure 13. Typical discretization of half of the domain of analysis (highly adaptive distribution of nodes).

| Table III. Mechanical material constants for the plate. |
|-----------------|---------|---------|---------|---------|---------|---------|
| $\rho$ (kg/m$^3$) | $E$ (GPa) | $v$ | $\sigma_0$ (MPa) | $\epsilon_0^p$ | $n$ | $\dot{\epsilon}_0^p$ | $m$ |
| 2700 | 69 | 0.33 | 276 | 0.001 | 13.5 | 1000 | 11.5 |

| Table IV. Mechanical material constants for the projectile. |
|-----------------|---------|---------|---------|---------|---------|---------|
| $\rho$ (kg/m$^3$) | $E$ (GPa) | $v$ | $\sigma_0$ (MPa) | $\epsilon_0^p$ | $n$ | $\dot{\epsilon}_0^p$ | $m_1$ | $m_2$ | $\dot{\epsilon}_t$ (s$^{-1}$) |
| 7650 | 200 | 0.3 | 1090 | 0.001 | 22 | 1 | 100 | 5 | $2 \times 10^5$ |

| Table V. Thermal constants. |
|-----------------|-------|-------|------|------|
| Material | $c$ (J/kgK) | $T_0$ (K) | $T_m$ (K) | $l$ | $\beta$ |
| Al6061-T6 | 896 | 298 | 853 | 0.5 | 0.9 |
| Stainless steel 440C | 477 | 298 | 1777 | 1.17 | 0.9 |

Figure 14. Calculation of ballistic impact of a 1.6 mm thick 6061-T6 aluminum plate struck by a 440C stainless steel spherical projectile at 2.4 km/s. Color shows the von-mises level in the target. (a) $t = 0.5\mu$s; (b) $t = 1.0\mu$s; (c) $t = 2.0\mu$s; and (d) $t = 4.0\mu$s.
A sequence of the perforation process simulated by the pOTM method is shown in Figure 14. Certifications of the model may be found in [19]. A strong scalability analysis is performed on a single node of LLNL’s Hera and SNL’s Glory using 1–16 threads. As stated earlier, LLNL’s Hera has four quad-core AMD Opteron 8356 processors (2.3 GHz) and 16 GB of memory per node. Similarly, SNL’s Glory has four quad-core AMD Opteron 2.2 GHz processors and 16 GB of memory per node. Recall that in the multithreaded pOTM method, all material points are stored in a continuous vector regardless of their spatial locations. The domain decomposition is conducted by simply assigning a subset of the vector to different threads. Thus, no geometrical domain decomposition is necessary. Two strategies, static and dynamic domain decomposition, are studied in our analysis. In the static case, the vector is subdivided into continuous subsets at the initialization of the calculation, and the subdivision remains unchanged in the entire simulation. Alternatively, in the dynamic approach, the decomposition of the vector is dynamically reconstructed at every iteration according

Figure 15. Parallel performance analysis for the multithreaded parallel implementation of the Optimal Transportation Meshfree simulations of terminal ballistics.

(a) static decomposition

(b) dynamic decomposition

Figure 16. Thread concurrency analysis of the simulations of terminal ballistics using eight threads by Intel(R) VTune(TM) Amplifier XE for the static and dynamic decomposition schemes. The horizontal green lines show idle threads and the black lines are active threads. With dynamic decomposition (b), the idle time is cut significantly; (a) static decomposition and (b) dynamic decomposition.
to the computational cost of the material points measured from the previous step. Figure 15 shows the scalability performance and comparison of the static and dynamic strategies. Because of the assembling operation at the global level, this part of the computation is handled by the main thread. Therefore, the theoretical limit of the speedup factor is estimated in terms of Amdahl’s Law. In addition, the thread concurrency is analyzed by Intel(R) VTune(TM) Amplifier XE, Figure 16. It is noteworthy that the dynamic decomposition scheme shows nearly perfectly balanced computational load between threads, and excellent scaling performance up to 16 threads is obtained in the dynamic approach as well.

4.3. Hypervelocity impact

In the last example, we demonstrate the scalability performance of the multiprocessing and multithreading hybrid pOTM approach in the numerical solutions of high energy density dynamic response of materials. In this type of applications, a large amount of energy is deposited into the material within a very short time such as in microseconds. As a consequence, extremely large deformations occur; distributed damage nucleates and propagates in the material; fragments or debris cloud forms and travels in space at a very high velocity. This problem serves as a extreme condition to test our parallel scheme. To this end, the hypervelocity impact (hvi) tests of Aluminum 6061-T6 plates struck by Nylon 6/6 cylindrical projectiles are simulated by the pOTM method. In order to quantitatively assess the modeling error and uncertainty, an Optimal UQ analysis [21] of the OTM hvi model is performed at Caltech’s PSAAP center, and in-house hvi experiments are conducted at Caltech’s Small Particle Hypervelocity Range Facilities for plates with dimensions $152 \times 152$ mm and thickness from 0.5 to 3.0 mm, impact velocity in the range of 5–7 km/s, and obliquity $0^\circ$ to $70^\circ$.

Again, the distribution of nodes and material points in a typical discretization is highly adaptive. The ratio of the size of large elements and small elements is in $O(10^2)$. Because of the octahedral symmetry, the domain is bisected into four partitions along the $xz$ and $yz$ planes across the center of the plate. Each partition is further decomposed by Metis as shown in Figure 17.

A typical hypervelocity impact process of a 0.5 mm Al6061-T6 plate by a nylon cylindrical particle at 5.8 km/s is shown in Figure 18. The material models and validation of the pOTM calculations can be found in [22]. The scalability study was carried out maintaining the same mesh size and increasing the number of processors and threads used on LLNL’s Cab. Cab consists of 1296 nodes connected by InfiniBand QDR and each node has 16 Intel Xeon E5-2670 cores and 32 GB of memory. In the analysis, the plate and projectile is modeled by 4.77 million material points and 2.49 million degrees of freedom. All calculations were conducted for a simulated time of 1 $\mu$s such that the important physical phenomena are captured including extremely large inelastic deformation, high pressure and temperature, multiphase transition, and mixing such as melting and vaporization, and fracture and fragmentation. Table VI outlines the scalable performance of the pOTM method.
Figure 18. Calculation of hypervelocity impact of a 0.5mm thick 6061-T6 aluminum plate struck by a nylon 6/6 cylindrical projectile at 5.8 km/s. (a) initial configuration; (b) $t = 0.5\mu s$; (c) $t = 1.0\mu s$; (d) $t = 2.0\mu s$; and (e) $t = 3.0\mu s$.

*Time per iteration* is a measure of the average wallclock time spent in computing a single timestep in Algorithm 2. Figure 19 shows the speedup factor as a function of the number of cores used. Because of the limitation of wallclock time and the total memory of a single node in the parallel system, the pOTM calculation with 256 MPI ranks and a single thread per rank is used as the baseline calculation. Thus, the *Time per iteration* for ideal performance decreases by the same ratio as the number of cores used divided by 256. It is evident that super-linear speedup is obtained in the study. Our explanation for this apparent super-linear effect follows.
Table VI. Parallel performance of the parallel implementation of the Optimal Transportation Meshfree simulation of hypervelocity impact tests using 4.77 million material points and 2.49 million degrees of freedom.

<table>
<thead>
<tr>
<th>Number of MPI ranks × threads per rank</th>
<th>Time per iteration (s)</th>
<th>Speedups</th>
<th>Efficiency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 × 1</td>
<td>162</td>
<td>1.0</td>
<td>100</td>
</tr>
<tr>
<td>512 × 1</td>
<td>43.38</td>
<td>3.73</td>
<td>186.5</td>
</tr>
<tr>
<td>1024 × 1</td>
<td>17.04</td>
<td>9.50</td>
<td>237</td>
</tr>
<tr>
<td>2048 × 1</td>
<td>7.55</td>
<td>21.45</td>
<td>268</td>
</tr>
<tr>
<td>4096 × 1</td>
<td>5.30</td>
<td>30.56</td>
<td>191</td>
</tr>
<tr>
<td>8192 × 1</td>
<td>4.74</td>
<td>34.17</td>
<td>106</td>
</tr>
<tr>
<td>4096 × 2</td>
<td>4.02</td>
<td>40.30</td>
<td>126</td>
</tr>
</tbody>
</table>

MPI, Message Passing Interface.

Figure 19. Scalable performance analysis for the multiprocessing and multithreaded pOTM simulations of hypervelocity impact tests.

Suppose the Time per iteration $t_{\text{tot}}$ may be decomposed additively,

$$t_{\text{tot}}(N) = t_{\text{comp}}(N) + t_{\text{comm}}(N) + t_{\text{assm}}(N)$$  \hspace{1cm} (16)

where $N$ is the number of cores, $t_{\text{comp}}$ denotes the actual computation time per iteration as listed in the Algorithm 2, $t_{\text{comm}}$ is the communication time caused by MPI to receive and send nodal information across distributed processors, and $t_{\text{assm}}$ indicates the extra computational cost due to the search, editing and insertion of elements to the data structure of nodal information during assembling the shadow nodes described in Section 3.1. Suppose the number of $N$ MPI processes is used and the computation on each MPI rank is perfectly balanced, the Time per iteration of the sequential computation can be calculated as

$$t_{\text{seq}} = N \times t_{\text{comp}}(N)$$  \hspace{1cm} (17)

and the speedup is measured as

$$\text{Speedup} = \frac{t_{\text{seq}}}{t_{\text{tot}}(N)}$$  \hspace{1cm} (18)
Because of MPI overheads, the speedup factor will be less than $N$. However, the $t_{seq}$ is unknown in our analysis, and we used $t_{tot}(256)$ as the baseline calculation measurement. Therefore, the speedup is calculated as

$$\text{Speedup} = \frac{t_{tot}(N)}{t_{tot}(256)} = \frac{t_{comp}(256) + t_{comm}(256) + t_{assm}(256)}{t_{comp}(N) + t_{comm}(N) + t_{assm}(N)}$$ (19)

for $N \geq 256$. If the computational load is perfectly balanced and $t_{comm} + t_{assm} \ll t_{comp}$, ideal speedup performance can be obtained.

Nevertheless, the assembling is a barrier operation in our MPI parallel algorithm and its time cannot be neglected. Using the standard domain decomposition strategy, with increasing MPI process count, the ratio of local subdomain surface and volume gets worse and at the same time the average message size is reduced, which results in the number of shadow nodes in the parallel computation with 256 cores much more than the one in the computations with $N \geq 512$. As a consequence, more memory operations are required to assemble the shadow nodes in our baseline calculation. For instance, in our analysis, $t_{assm}(256)$ is $O(10)$ times more than $t_{assm}(1024)$. Furthermore, a large $t_{assm}$ causes load imbalance between processors as well. According to Equation (19), it is possible to obtain a super-linear speedup factor in case that the communication cost is negligible and the assembling cost reduces much faster than the computational cost. We do see, however, that the efficiency of the pOTM approach with a single thread per MPI rank begins to drop off severely as the number of cores approaches to 4096 or the size of the problem per MPI rank approaches several hundreds of material points. In this case, the communication cost can no longer be amortized by large amounts of computation. Our solution is activating the multithreading parallelization, that is, using multiple threads per MPI process. As a shared memory scheme, the communication costs will not change but the computational costs are further distributed to threads. The effectiveness of hybrid parallel approach is noteworthy in Figure 19 where an extra 20% improvement of the scalability performance is achieved.

5. DISCUSSIONS

We have presented in this work a massively pOTM method [1], a multiprocessing and multithreading hybrid parallel implementation of the OTM method using MPI and Pthreads. The set of material points is partitioned using a graph-based or geometric domain decomposition algorithm and distributed across MPI processes at the initialization of the computation. A shadow scheme is developed for the synchronization of nodes that belong to the support of material points on different MPI processes at each iteration. The results of our performance analysis indicate that up to $O(10^4)$ MPI processes, the overhead introduced because of message passing is very small compared to the execution time required for the computations of the complex physics in high energy density dynamic response of materials. Moreover, for applications with multiscale material models, the pure MPI parallelization scales almost perfectly up to 8000 MPI processes.

Because of the static domain decomposition for MPI parallelization, with increasing process count, the ratio of local subdomain surface and volume gets worse, but the average message size does not reduce in the same rate in our shadow scheme. In this case, a multithreading parallelization based on the Pthreads library is proposed to help reduce the number of subdomains and improve the rate of convergence. In addition, a thread pool management and dynamic load balance approach are developed to reduce the overhead in Pthreads parallel regions. The scalable performance of our hybrid parallelization with MPI and Pthreads demonstrates significant speedup of the computation up to 8192 cores in our numerical experiments of hypervelocity impact tests.

Although our analysis shows trivial communication cost across processors compared to the computational cost for each MPI process in the multiprocessing parallelization, the assembling of the information of the shadow nodes becomes the major bottleneck in our implementation of the parallel program. In particular, because of the highly adaptive distribution of the material points and the dramatic change of their neighborhood in the simulations of materials response in extreme dynamic...
environment, overlaps between the cubical range boxes increases dynamically, which results on demand of the assembling. In our implementation of the hybrid MPI/Pthreads parallelization, the improvement of the scalability performance of the program converges with only four threads per MPI processes because of the barrier of the assembling operation in the MPI parallelization. A profiling of the code shows the searching, editing, and insertion operation to the data structure in assembling the nodal information can cause communications stalls that appear as MPI overhead. Therefore, a better implementation of the data structure with efficient search and modification operations may reduce the MPI overhead and improve the concurrency between MPI processes. In addition, the assembling operations may be reduced by using polygonal range boxes. Further investigation of the hybrid MPI/Pthreads approach to evaluate the parallelization with respect to the number of threads per MPI process and the number of MPI processes per node is required. The mapping between MPI processes and threads to sockets and cores within a compute node will also strongly influence the performance of the hybrid approach.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of the Department of Energy National Nuclear Security Administration under Award Number DE-FC52-08NA28613 through Caltech’s ASC/PSAAP Center for the Predictive Modeling and Simulation of High Energy Density Dynamic Response of Materials. M. Stalzer acknowledges the hospitality of the Aspen Center for Physics, which is supported by the National Science Foundation Grant No. PHY-1066293. The large-scale pOTM simulations are courtesy of Sharon Brunett.

REFERENCES


