Journal of Computational Physics 307 (2016) 164–188

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Mesh-free data transfer algorithms for partitioned multiphysics problems: Conservation, accuracy, and parallelism ☆

Stuart R. Slattery [1]

Oak Ridge National Laboratory, 1 Bethel Valley Rd., Oak Ridge, TN 37831, USA

ARTICLE INFO

Article history:
Received 19 May 2015
Received in revised form 25 November 2015
Accepted 26 November 2015
Available online 2 December 2015

Keywords:
Data transfer
Multiphysics
Parallel algorithms
Moving least square
Spline interpolation

ABSTRACT

In this paper we analyze and extend mesh-free algorithms for three-dimensional data transfer problems in partitioned multiphysics simulations. We first provide a direct comparison between a mesh-based weighted residual method using the common-refinement scheme and two mesh-free algorithms leveraging compactly supported radial basis functions: one using a spline interpolation and one using a moving least square reconstruction. Through the comparison we assess both the conservation and accuracy of the data transfer obtained from each of the methods. We do so for a varying set of geometries with and without curvature and sharp features and for functions with and without smoothness and with varying gradients. Our results show that the mesh-based and mesh-free algorithms are complementary with cases where each was demonstrated to perform better than the other. We then focus on the mesh-free methods by developing a set of algorithms to parallelize them based on sparse linear algebra techniques. This includes a discussion of fast parallel radius searching in point clouds and restructuring the interpolation algorithms to leverage data structures and linear algebra services designed for large distributed computing environments. The scalability of our new algorithms is demonstrated on a leadership class computing facility using a set of basic scaling studies. These scaling studies show that for problems with reasonable load balance, our new algorithms for both spline interpolation and moving least square reconstruction demonstrate both strong and weak scalability using more than 100,000 MPI processes with billions of degrees of freedom in the data transfer operation.

© 2015 Elsevier Inc. All rights reserved.

1. Introduction

When a partitioned approach is selected for multiphysics simulation, a data transfer strategy must be employed that considers the fact that both a mathematical framework and data structures are likely not to be shared amongst the various physics components used in the calculation. Furthermore, high fidelity modern physics simulations demand data transfer

☆ Notice: This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

E-mail address: slatterysr@ornl.gov.

[1] Computational Engineering and Energy Sciences Group, Computer Science and Mathematics Division.

http://dx.doi.org/10.1016/j.jcp.2015.11.055
0021-9991/© 2015 Elsevier Inc. All rights reserved.

techniques that demonstrate scalability on modern high performance computing hardware. When large scale calculations are targeted, data transfer algorithms that do not exhibit good parallel performance can destroy the scalability of the entire coupled calculation, regardless of the performance of the other components used in the simulation. In addition to scalability, we also require these methods to be both conservative and accurate to ensure stability of the calculations [1]. From the perspective of implementing partitioned physics coupling, algorithms that enable a simple set of inputs and outputs provide additional computational value and usability. Recently developed mesh-free algorithms based on surface reconstruction techniques using compactly supported radial basis functions potentially satisfy these requirements and may serve as an alternative to algorithms that explicitly utilize a computational mesh and the associated topological information.

For data transfers on an interface such as the transfer of loads in a fluid–structure-interaction calculation or the transfer of heat fluxes in a conjugate heat transfer calculation, additional complexities arise when the discretizations on either side of that interface do not match both in geometric structure and the numerical form chosen for functional support. When either side of the interface is discretized with a mesh, several groups have made contributions with methods that address both conservation and accuracy [2–5]. These methods are often characterized by a weighted residual problem with accuracy and conservation properties that rely on the method of numerical integration used to assemble the problem and the accuracy of the functional support in the underlying physics discretization. Parallelism in these methods has been addressed with much of the problem effectively relying on the tools and techniques of parallel finite element assembly [6].

As an alternative to methods that leverage the meshes discretizing a shared interface, significant work has been performed in recent years on mesh-free techniques that instead apply the data transfer between sets of point clouds that represent the discrete interface [7–13]. In these methods, one effectively removes the functional support provided by the underlying physics discretization and replaces it with support of often a higher order constructed from various basis functions. Two types of basis functions are generally used in the literature: those with global support such as a thin plate spline and those with compact support provided by radial basis functions. Those with global support are formulated such that all nodes provide support to all other nodes resulting in a dense coupling matrix. Methods leveraging compactly supported radial basis functions, however, only require support for a given node in its local spatial neighborhood, thus providing a natural element of sparsity and therefore potential scalability to the algorithm. Furthermore, their formulation also makes them a potentially viable option for shared domain coupling where physics fields are exchanged over a shared volume of space in addition to problems where information is exchanged on a shared interface.

This work has two purposes. The first is to provide a comparison of conservation and accuracy for shared interface problems between the weighted residual method, namely that leveraging the common-refinement technique of Jiao and Heath [4], and two mesh-free methods with compact support: the spline interpolation method of Beckert and Wendland [7] and the moving least square formulation of Quaranta, Masarati, and Mantegazza [8]. We do this because we have yet to observe this particular combination of algorithms compared in the numerous survey papers on these methods [4,14–18], and in these papers, each of these methods is shown to be amongst the most accurate and robust of those that are tested. Of equal importance, these survey papers, particularly those assessing the mesh-based methods, do not always provide three-dimensional comparisons and instead typically conduct investigations with one and two-dimensional problems. Therefore, we feel that it is warranted to directly compare these methods and to do so for three-dimensional problems so that situations in which any one of them are more effective can be identified. In addition, although not strictly conservative, it is not entirely clear from the literature how accurate interpolations using the mesh-free algorithms translate to conservative interpolations and how these compare to the inherent conservative properties of the common-refinement method. To address this, we have constructed a series of numerical experiments using various types of functions and geometries in which we measure both the levels of accuracy and conservation achieved by each data transfer method for a repeated sequence of transfers.

The second purpose of this work is to develop efficient parallel algorithms for the spline interpolation and moving least square reconstruction techniques to enable large scale multiphysics simulations in distributed computing environments. Some efficient parallel schemes have been developed outside of the context of multiphysics data transfer for the spline interpolation technique. These schemes use an additive Schwarz domain decomposition strategy and were demonstrated on problems of modest size for both distributed computing platforms [19] and GPUs [20]. However, these algorithms only focus on interpolation methods without compact support where the interpolation problem is inherently dense and modifications are made to the algorithm to achieve scalable, global solutions. For compactly supported interpolations, others do not report parallel results but indicate the possibility of parallelism for data transfers with, for example, the solution leveraging a partition of unity scheme in subdomains [21]. In the context of multiphysics data transfer, we have yet to observe massively parallel spline interpolation algorithms with compact support reported in the literature and no parallel results reported for the moving least square scheme. Therefore, we present algorithms that we have developed for compactly supported interpolations leveraging sparse linear algebra techniques that aim to achieve data transfer at high levels of concurrency with a large number of degrees of freedom. To demonstrate the performance of our algorithms, we perform scaling studies using a leadership class computing facility.

This paper is organized as follows. In §2 we define the data transfer problem and then provide details on the three algorithms used in this work. In §3 we directly compare the weighted residual technique using the common-refinement method with the two mesh-free algorithms in order to assess their conservation and accuracy properties for a set of test problems with data transfer on a shared interface. In §4 we extend the mesh-free methods by presenting algorithms to

parallelize them. In §5 we demonstrate the scalability of our parallel algorithms using a leadership class computing facility. Finally, in §6 we draw conclusions and indicate future work.

2. Data transfer algorithms

The purpose of a data transfer algorithm in a multiphysics simulation is to exchange solution fields or other response functions between physics kernels. In this section we first define the data transfer problem where the solution results in this exchange of fields and then present three techniques for solving this problem: a weighted residual method, a spline interpolation method, and a moving least square reconstruction algorithm.

2.1. The data transfer problem

To present a common framework for these algorithms, we define the data transfer problem following the notation of Jiao and Heath presented in [4]. The data transfer problem is defined as the translation or reconstruction of a function represented by a discretization on one grid to a representation on a potentially different grid with a different discretization. The grid on which the function is initially represented is defined as the *source grid* and the grid onto which the function will be transferred is defined as the *target grid*. In its representation on the source grid, the function will be defined as, f, the *source function*. The representation of the function on the target grid will be, g, the *target function*. We define the source function over M control points as:

$$f = \sum_{i=1}^{M} f_i \phi_i \,, \tag{1}$$

where:

$$f_i = f(\mathbf{s}_i) \,, \tag{2}$$

is the source function evaluated at the ith control point in the source grid, $\mathbf{s}_i$, and:

$$\phi_i = \phi(\hat{\mathbf{s}}_i) \,, \tag{3}$$

is the source basis function evaluated at the parametric coordinates of the ith control point in the source grid, $\hat{\mathbf{s}}_i$. Equivalently, we have the target function defined over N control points as:

$$g = \sum_{i=1}^{N} g_i \psi_i \,, \tag{4}$$

where:

$$g_i = g(\mathbf{t}_i) \,, \tag{5}$$

is the target function evaluated at the ith control point in the target grid, $\mathbf{t}_i$, and:

$$\psi_i = \psi(\hat{\mathbf{t}}_i) \,, \tag{6}$$

is the target basis function evaluated at the parametric coordinates ith control point in the target grid, $\hat{\mathbf{t}}_i$. In our definitions of ϕ and ψ, we make no assumption of the form of functional support per [3]. Rather, we only require the existence of some local or global support for the functions on the coupled domains.

The data transfer problem is then to find the values of the source function on the target grid given the function discretization on both grids and the source function values on the source grid control points. We define a *data transfer operator*, $\mathbf{H}$, such that:

$$g \leftarrow \mathbf{H}(\phi, \psi) f \,, \tag{7}$$

with $\mathbf{H} : \mathbb{R}^M \rightarrow \mathbb{R}^N$, $f \in \mathbb{R}^M$, and $g \in \mathbb{R}^N$ such that the transfer operator applied to the source function yields the target function and $\mathbf{H}(\phi, \psi)$ indicates that $\mathbf{H}$ is potentially constructed from the basis functions of the source and target discretizations. For each of the data transfer algorithms presented in this paper, the goal is to then form $\mathbf{H}$ and apply it to any given $f \in \mathbb{R}^M$.

The quality of a data transfer algorithm is indicated by two characteristics: *conservation* and *accuracy*. To identify these quantities we define the *residual*, r, of the data transfer operation:

$$r = g - f \,. \tag{8}$$

Accuracy is then defined in terms of some norm of the residual such that a perfectly accurate data transfer operation satisfies:

$$||r|| = 0 \,, \tag{9}$$

and conservation is defined in terms of some integral of the residual such that a perfectly conservative data transfer operation satisfies:

$$\int_{\Omega} |r| d\Omega = 0 \,, \tag{10}$$

where Ω is the shared domain or interface of interest.

2.2. Weighted residuals and common-refinement

If a mesh is available on either side of a shared interface, a weighted residual method can be used that leverages the underlying discretization of the field to be transferred. To construct the data transfer operator for this method, the weighted residual problem based on L_2 minimization is formed as follows per [4]. We minimize the 2-norm of the data transfer residual, $||f - g||_2$, by solving the following problem:

$$\frac{\partial}{\partial g_i}\left[\int_{\Omega} (g - f)^2 d\Omega\right] = 0 \,. \tag{11}$$

Expanding the squared term and substituting in Eq. (4) for g we have:

$$\frac{\partial}{\partial g_i}\left[\int_{\Omega} (g - f)^2 d\Omega\right] = \frac{\partial}{\partial g_i}\left[\int_{\Omega} \left(\left(\sum_{j=1}^{N} \psi_j g_j\right)^2 - 2\sum_{j=1}^{N} \psi_j g_j f + f^2\right) d\Omega\right] . \tag{12}$$

Performing the differentiation and separating the integral gives the following linear system to solve for the target function:

$$\sum_{j=1}^{N} \int_{\Omega} \psi_i \psi_j d\Omega g_i = \int_{\Omega} \psi_i f d\Omega \,, \tag{13}$$

or

$$\mathbf{M} g = \mathbf{b} \,, \tag{14}$$

where $\mathbf{M}$ is defined as the *mass matrix* with individual elements:

$$M_{ij} = \int_{\Omega} \psi_i \psi_j d\Omega \,, \tag{15}$$

$\mathbf{b}$ is defined as the *load vector* with individual components:

$$b_i = \int_{\Omega} \psi_i f d\Omega \,, \tag{16}$$

and g the vector of unknowns. The action of the data transfer operator on the source function is then:

$$\mathbf{H} f = \mathbf{M}^{-1} \mathbf{b}(f) \,, \tag{17}$$

where now a symmetric positive-definite linear system must be solved and $\mathbf{b}(f)$ indicates that the load vector is constructed at every application of the data transfer operator from the current values of the source function. Although the weighted residual problem can be weakly formulated as in [2], we find this definition to be more general as it permits minimizing the data transfer residual over other norms.[2]

Building the mass matrix only requires integrations of the target basis functions and therefore those integrations can always occur exactly over the target grid. However, constructing the load vector requires numerical integration of the target basis functions and the source function containing the source basis functions. As both of those functions are likely to be defined over grids of different topologies and/or mesh size, the question becomes how to perform those integrations in a way that is both accurate and conservative. A first choice for building the load vector is numerical integration over the source grid while a second choice is numerical integration over the target grid [2,3]. However, in both cases any numerical integrations carried out over one grid requires integration of the basis functions of the other grid. In general, one set of these functions will not be continuous across the elements of the integration grid and therefore accuracy and conservation is not

[2] Such as the Sobolev norm as in [4].

expected due to integrations over discontinuities.[3] By noting that the key issue of maintaining accuracy and conservation in a data transfer is exact numerical integrations of both the source function and the target basis functions, Jiao and Heath [4] introduced a third grid into the data transfer operation termed the *common-refinement*. Defined as the topological intersection of the source and target grids, consistency of the integrations is obtained by performing the numerical integrations over the common-refinement. Because the grids are geometrically intersected, this means that both the source function and the target basis functions will be continuous in each of its elements, therefore yielding both accuracy and conservation by providing exact integrations of both. Details on how to construct a common-refinement of two grids with linear elements on arbitrary three-dimensional interfaces with curvature and sharp features are given in [22] and [23].

2.3. Mesh-free algorithms

We next define two general mesh-free algorithms for data transfer prefixed by a discussion on the compactly supported radial basis functions leveraged by both.

2.3.1. Compactly supported radial basis functions

Unlike the weighted residual technique, mesh-free methods for data transfer do not consider the discretization of fields when constructing the interpolant. Instead, the domain is represented by arbitrary point clouds with topology-independent functional support defined only for means of interpolation. Radial basis functions permit the construction of functional support in arbitrary point clouds by using the Euclidean distance, r, between a point, i, and any one of its supporting points, j:

$$r = ||\mathbf{x}_i - \mathbf{x}_j||_2 \,, \tag{18}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the physical coordinates of the points i and j respectively. These functions typically come in two forms, those with global support and those with compact support. As an example of a radial basis function with global support, consider the thin plate spline function:

$$\phi(r) = r^2 \log(r) \,. \tag{19}$$

To construct support for a point using this function, all control points in the coupling domain would be required and those that are further away give a greater contribution. As an alternative, radial basis functions with compact support are highly desirable for large interpolation problems because of the sparsity that they introduce into the problem by requiring only a subset of the control points in the coupling domain to construct support. As an example, consider Wu's C^4 function:

$$\phi(r) = \left(5\left(\frac{r}{\rho}\right)^5 + 30\left(\frac{r}{\rho}\right)^4 + 72\left(\frac{r}{\rho}\right)^3 + 82\left(\frac{r}{\rho}\right)^2 + 36\left(\frac{r}{\rho}\right) + 6\right)\left(1 - \frac{r}{\rho}\right)_+^6 \,, \tag{20}$$

where ρ is the physical support radius of the function and the $(\cdot)_+$ notation on the second term indicates that term evaluates to zero if $(1 - r/\rho) < 0$, thus truncating support at the boundary of the physical support radius. The choice of ρ will dictate how many control points are used for support with more points typically providing a better interpolation. For cases where the two grids in the data transfer have non-uniform spacing, such as cases with local refinement, a fixed support radius may not be the best choice. Locations with little refinement may not have enough support while locations with heavy support may be computationally intensive. In these cases, one can instead fix the size of the local support and calculate the local radius from the supporting points. All calculations in this paper will use a fixed radius of support. As a guideline, a support radius or number of support points should be chosen such that local gradients can be captured. We refer the reader to [24] for more details and examples of these functions.

2.3.2. Spline interpolation

Beckert and Wendland introduced a mesh-free data transfer technique for fluid–structure interaction problems based on spline interpolation [7] which was further generalized by Rendall and Allen [9]. How to construct the method is presented in detail in those references but we briefly present their algorithms here in order to later clarify certain choices when we parallelize the algorithm in §4. The method is based on the assumption that conservation of energy is achieved through a linear transformation operator, $\mathbf{H}$. In our more general case, this is simply the target function reconstructed from applying a data transfer operator to the source function:

$$\mathbf{g} = \mathbf{H}\mathbf{f} \,, \tag{21}$$

where the functional discretization of $\mathbf{f}$ and $\mathbf{g}$ is ignored and the control points in both grids are treated as point clouds without topological information.

[3] See [4,15,16] for examples of spurious oscillations and loss of conservative properties due to this effect.

To build the operator, we begin by constructing a continuous interpolant of the source function by using the known values at the M source control points and support them with radial basis functions, ϕ:

$$f(\mathbf{x}) = p(\mathbf{x}) + \sum_{i=1}^{M} \alpha_i \phi(||\mathbf{x} - \mathbf{s}_i||_2) \,, \tag{22}$$

where $p(\mathbf{x})$ is defined as a linear polynomial in terms of the spatial coordinates of the evaluation point, $\mathbf{x}$:

$$p(\mathbf{x}) = \beta_0 + \beta_1 \mathbf{x}_x + \beta_2 \mathbf{x}_y + \beta_3 \mathbf{x}_z \,, \tag{23}$$

with polynomial coefficients β_i and α_i are the basis coefficients constrained by:

$$\sum_{i=1}^{M} \alpha_i p(\mathbf{x}_i) = 0 \,. \tag{24}$$

We can find the coefficients by solving the following linear system given by the combinations of Eqs. (22) and (24):

$$\mathbf{C}\mathbf{a} = \begin{bmatrix} \mathbf{0} \\ \mathbf{I_M} \end{bmatrix} \mathbf{f} \,, \tag{25}$$

with $\mathbf{f}$ the source function, $\mathbf{I_M} \in \mathbf{R}^{M \times M}$ the identity matrix, $\mathbf{a}$ the vector of coefficients:

$$\mathbf{a} = \begin{bmatrix} \beta_0 & \beta_1 & \beta_2 & \beta_3 & \alpha_0 & \alpha_1 & \dots & \alpha_{M-1} \end{bmatrix}^T \,, \tag{26}$$

and $\mathbf{C}$ a matrix defined as:

$$\mathbf{C} = \begin{bmatrix} \mathbf{0} & \mathbf{P}^T \\ \mathbf{P} & \mathbf{M} \end{bmatrix} \,. \tag{27}$$

In Eq. (27), $\mathbf{C}$ is divided into a polynomial component:

$$\mathbf{P} = \begin{bmatrix} 1 & x_{s_1} & y_{s_1} & z_{s_1} \\ 1 & x_{s_2} & y_{s_2} & z_{s_2} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{s_M} & y_{s_M} & z_{s_M} \end{bmatrix} \,, \tag{28}$$

with x_{s_i}, y_{s_i}, and z_{s_i} the coordinates of source control point i and a basis component:

$$\mathbf{M} = \begin{bmatrix} \phi_{s_1 s_1} & \phi_{s_1 s_2} & \cdots & \phi_{s_1 s_M} \\ \phi_{s_1 s_2} & \phi_{s_2 s_2} & \cdots & \phi_{s_2 s_M} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{s_M s_1} & \phi_{s_M s_2} & \cdots & \phi_{s_M s_M} \end{bmatrix} \,, \tag{29}$$

with $\phi_{s_i s_j} = \phi(||\mathbf{s}_i - \mathbf{s}_j||_2)$ indicating the value of the supporting radial basis function between source control points i and j.

Once we have obtained the coefficients with the solution of Eq. (25) we can then reconstruct the target function by evaluating the interpolating function, Eq. (22), at the target control points via:

$$\mathbf{g} = \mathbf{A}\mathbf{a} \,, \tag{30}$$

where the matrix $\mathbf{A}$ is defined as:

$$\mathbf{A} = \begin{bmatrix} \mathbf{Q} & \mathbf{N} \end{bmatrix} \,, \tag{31}$$

with polynomial component:

$$\mathbf{Q} = \begin{bmatrix} 1 & x_{t_1} & y_{t_1} & z_{t_1} \\ 1 & x_{t_2} & y_{t_2} & z_{t_2} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{t_N} & y_{t_N} & z_{t_N} \end{bmatrix} \,, \tag{32}$$

and basis component:

$$\mathbf{N} = \begin{bmatrix} \phi_{t_1 s_1} & \phi_{t_1 s_2} & \cdots & \phi_{t_1 s_M} \\ \phi_{t_2 s_1} & \phi_{t_2 s_2} & \cdots & \phi_{t_2 s_M} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{t_N s_1} & \phi_{t_N s_2} & \cdots & \phi_{t_N s_M} \end{bmatrix} . \tag{33}$$

By combining Eq. (25) and Eq. (30) we can then define the data transfer operator as:

$$\mathbf{H} = \mathbf{A}\mathbf{C}^{-1} \begin{bmatrix} \mathbf{0} \\ \mathbf{I_M} \end{bmatrix} . \tag{34}$$

This technique is applicable to both scalar and vector forms of $\mathbf{f}$ and $\mathbf{g}$ provided that the scheme for solving Eq. (25) can do so with multiple right hand sides (i.e. $\mathbf{f} \in \mathbb{R}^{M \times \nu}$ and $\mathbf{g} \in \mathbb{R}^{N \times \nu}$ with ν being the number of vector components). In addition, we do recognize here that as an alternative to requiring a global linear solve at each application of the data transfer operator to produce the coefficients in Eq. (25), one can instead explicitly construct the entries of $\mathbf{H}$ through a partition of unity as presented in [25] and [10] at the cost of a potential loss of accuracy. In this case, applying $\mathbf{H}$ is simply a matrix–vector multiplication with the entries of $\mathbf{H}$ assembled through a weighted sum of local spline interpolations.

2.3.3. Moving least square reconstruction

As an alternative to spline-based interpolation, Quaranta, Masarati, and Mantegazza presented a function reconstruction technique for arbitrary point clouds based on a moving least square discretization [8]. In this method, support and subsequently the data transfer operator is constructed through solutions to local least square kernels defined by compactly supported radial basis functions. First, the solution at each target point is formulated as a polynomial sum:

$$\mathbf{g}_i = \sum_{i=1}^{m} p_i(\mathbf{t}_i) a_i(\mathbf{t}_i) , \tag{35}$$

where the a_i are the polynomial coefficients and the $p_i(\mathbf{t}_i)$ are quadratic polynomials in terms of spatial coordinates In vector form for an arbitrary three-dimensional point, q, these polynomials are:

$$\mathbf{p}(q) = \begin{bmatrix} 1 & x_q & y_q & z_q & x_q^2 & x_q y_q & y_q^2 & y_q z_q & z_q^2 & z_q x_q \end{bmatrix}^T . \tag{36}$$

We can create the interpolant by minimizing the data transfer residual through a weighted least square procedure at each individual target point, $\mathbf{t}_i$:

$$\textit{Minimize} \; \frac{\partial}{\partial a_i(\mathbf{t}_i)} \int_{\Omega} \phi(\mathbf{t}_i - \mathbf{s})(\mathbf{g} - \mathbf{f})^2 d\Omega(\mathbf{s}) , \tag{37}$$

where the $a_i(\mathbf{t}_i)$ serve as the free parameters in the minimization, $\Omega(\mathbf{s})$ is the domain of interest specifically defined by the supporting source control points, and compactly supported radial basis functions defined at the target control point and supported by the source control points, $\phi(\mathbf{t}_i - \mathbf{s})$, serve as the weights in the minimization. To close the problem, Eq. (35) provides a natural constraint for the minimization procedure. Solving the minimization problem gives the following kernel to be evaluated at each target control point[4]:

$$\xi(t_i) = \mathbf{p}(t_i)^T \left[\mathbf{P}(s_{t_i})^T \mathbf{\Phi}(s_{t_i}) \mathbf{P}(s_{t_i}) \right]^{-1} \mathbf{P}(s_{t_i})^T \mathbf{\Phi}(s_{t_i}) , \tag{38}$$

with s_{t_i} the set of n source points within the support radius of the ith target point, $\mathbf{t_i}$. The polynomial component is defined as:

$$\mathbf{P}(s_{t_i}) = \begin{bmatrix} \mathbf{p}(s_{t_1}) \\ \mathbf{p}(s_{t_2}) \\ \cdots \\ \mathbf{p}(s_{t_n}) \end{bmatrix} , \tag{39}$$

[4] See [8] for a derivation of this kernel.

and the basis component:

$$\mathbf{\Phi}(s_{t_i}) = \begin{bmatrix} \phi_{s_{t_1} t_i} & 0 & \cdots & 0 \\ 0 & \phi_{s_{t_2} t_i} & \cdots & 0 \\ \cdots & \cdots & \ddots & \cdots \\ 0 & 0 & \cdots & \phi_{s_{t_n} t_i} \end{bmatrix} . \tag{40}$$

We can then reconstruct the target function through independent evaluations of the moving least square kernel in Eq. (38) at each target control point. Alternatively, we can store the solutions to the kernels as a matrix to form the data transfer operator:

$$H_{ij} = \xi(\mathbf{t}_i)_j, \iff \mathbf{s}_j \in s_{t_i} , \tag{41}$$

where i is the index of the target control point and j the index of the source control point providing support within the radius of $\mathbf{t}_i$. Compared to the spline interpolation technique presented in §2.3.2, in addition to a naturally local formulation without the need for a global linear solve or a partition of unity method to enable locality, the method also has no coupling amongst target control points. In other words, the structure of the resulting linear operator, $\mathbf{H}$, in Eq. (41) is sparser than that in Eq. (34) because radial basis support only links target control centers to neighboring source control centers rather than a linking to both neighboring source and target control points. Furthermore, $\mathbf{f}$ and $\mathbf{g}$ may again have multiple dimensions with the resulting application of $\mathbf{H}$ occurring once for each dimension.

3. Algorithm comparison

In this section, a set of numerical experiments informed to some extent by those conducted in [4] and [16] are performed to demonstrate certain properties of each of the data transfer algorithms. Specifically, these experiments test the algorithms in the context of surface transfer where a surface mesh or point clouds representative of surfaces are used in the transfer. For each numerical experiment, a three-dimensional function is presented and then repeated data transfers are performed over the shared interface of the given geometries. For the discretization of the interfaces, finite elements are used with either triangular or quadrilateral faces supported by underlying tetrahedral and hexahedral meshes respectively with linear Lagrangian basis functions of degree 1. At each iteration, the function is simply transferred from the quadrilateral faces to the triangular faces and then back again without any modifications. When using the mesh-free methods, the nodes of the surface meshes are extracted and used as the representative point clouds. Doing so permits the data transfer to reconstruct the function on the same set of nodes as the common-refinement technique allowing for a direct comparison between the methods.

For each test case, we will directly measure accuracy and conservation on both grids as a function of iteration. Given a reference function for each problem, $u(x)$, accuracy will be measured by $||\mathbf{e}||_2$ where $\mathbf{e}$ is the error vector defined as:

$$\mathbf{e}_i = \frac{g(x_i) - u(x_i)}{u(x_i)} , \tag{42}$$

where x_i is the ith control point in the starting grid, $g(x_i)$ is the function evaluated at the ith control point in the starting grid, and $u(x_i)$ the reference solution evaluated at the ith control point in the starting grid. For conservation, numerical integrations will be computed over the grids such that:

$$G = \sum_{j=1}^{N} G^j , \tag{43}$$

where N is the number of finite elements in the starting grid and G^j is the integral of the transferred function in the jth element defined as:

$$G^j = \int_{\Omega_j} \psi_j g_j d\Omega_j , \tag{44}$$

where Ω_j is the domain of the jth element and ψ_j and g_j the basis functions and function values in that element. Given a reference function, $u(x)$, for each problem the reference integral will then be:

$$U = \sum_{j=1}^{N} U^j , \tag{45}$$

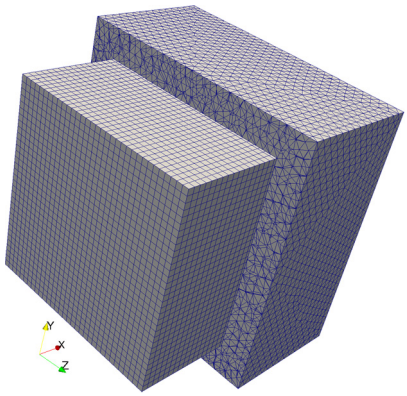


Fig. 1. Nested cube meshes used for the function order test. The outer tetrahedral mesh has been clipped to show the interior mesh of hexahedrons.

Table 1
Nested cube mesh sizes for the function order test. The sizes given are for the surfaces of the cubes in contact.

	Nodes	Faces
Inner cube (Hex)	6536	6534
Outer cube (Tet)	3656	7308

constructed numerically in the same way as G with element-wise integrals as:

$$U^j = \int_{\Omega_j} \psi_j u_j d\Omega_j \,, \tag{46}$$

where the same basis function from Eq. (44) is used for the in integration. Using these, the conservation error, E, is defined as:

$$E = \frac{|G - U|}{|U|} \,. \tag{47}$$

A quadrature rule was used with a degree of 2 to exactly integrate the linear basis functions supporting the functions in each test. Per the advice in [16], we choose to form the common-refinement by projecting onto the target surface instead of a computed reference surface as this provides the best accuracy results.

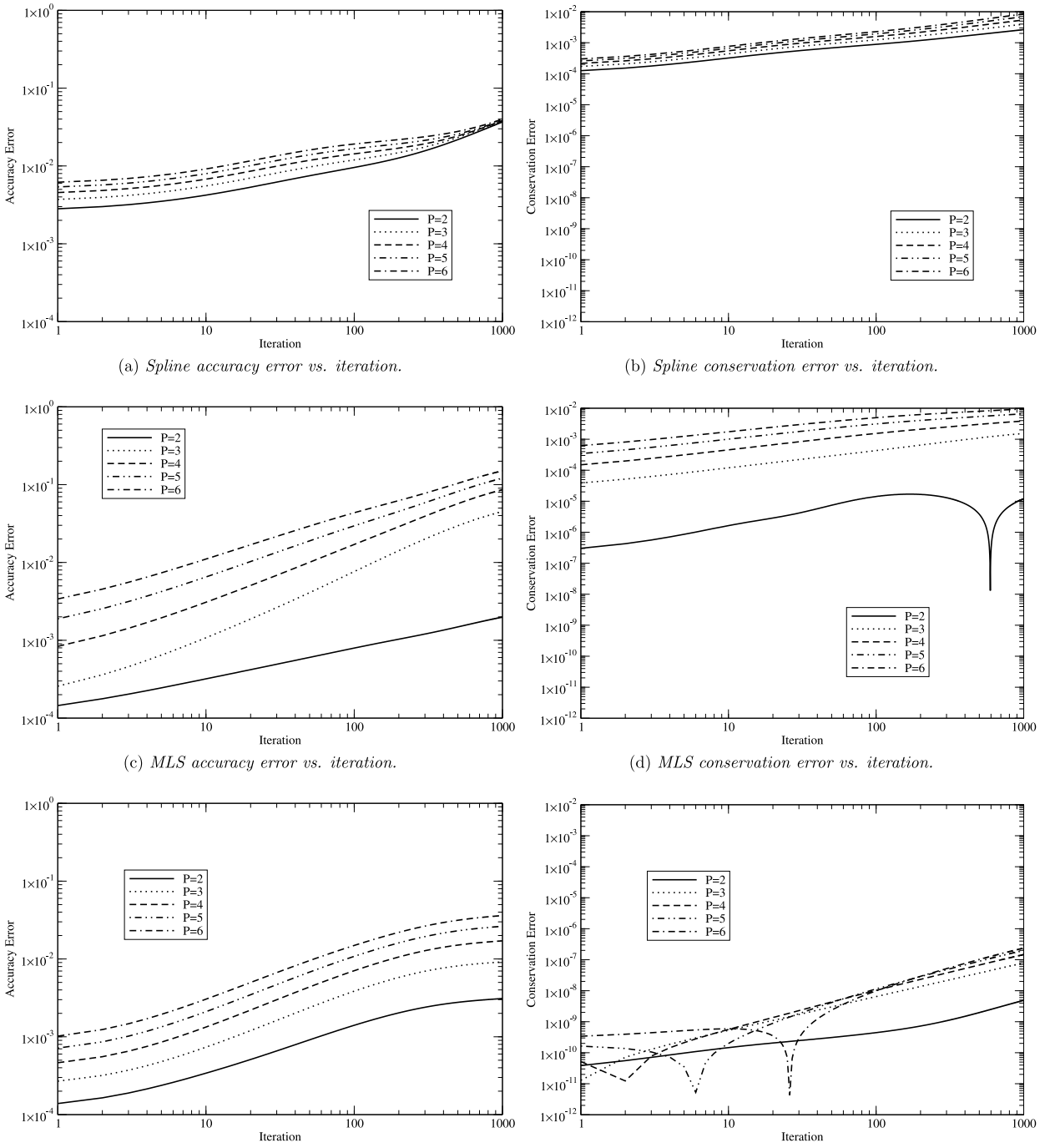
3.1. Function order test

In general, interpolation methods for data transfer are sensitive to the high order derivatives of the data. In the case of linear interpolation, for example, Appendix A of [4] indicates the magnitude of the second derivative dominates the interpolation error. For our first numerical experiment, we will assess the data transfer methods for functions of varying order so that the effect of higher order derivatives on the transfer results may be observed. This test is performed on a shared interface between the two nested cubes shown in Fig. 1 with the inner cube meshed with hexahedrons and the outer cube meshed with tetrahedrons. The size of these meshes is given in Table 1. For the reference function, we use the following:

$$u(x,y,z) = 10\left(\left|\frac{x}{3}\right|^P + \left|\frac{y}{3}\right|^P + \left|\frac{z}{3}\right|^P\right) + \left|\frac{x}{5}\right| + \left|\frac{y}{5}\right| + \left|\frac{z}{5}\right| \,, \tag{48}$$

where P is an integer specifying the order of the function. For support, the mesh-free methods used a radius of 1.5 with the mesh spacing $\approx$0.3 and Wu's function given by Eq. (20).

The function was defined first on the inner cube mesh and then 1000 repeated transfer iterations were performed. Fig. 2 gives the accuracy and conservation errors defined by the two-norm of Eq. (42) and Eq. (47) respectively for both mesh-free algorithms and the common-refinement technique at the end of each iteration on the inner cube surface. In general, these results indicate that for all methods the accuracy is dependent on the order of the function with higher order functions reproduced less accurately. Based on the results given in [4] for various one and two-dimensional problems for spline interpolation without compact support, the compactly supported interpolations perform in a similar manner with the spline interpolation algorithm giving measures of accuracy competitive with the common-refinement technique. In addition,



(a) *Spline accuracy error vs. iteration.*

(b) *Spline conservation error vs. iteration.*

(c) *MLS accuracy error vs. iteration.*

(d) *MLS conservation error vs. iteration.*

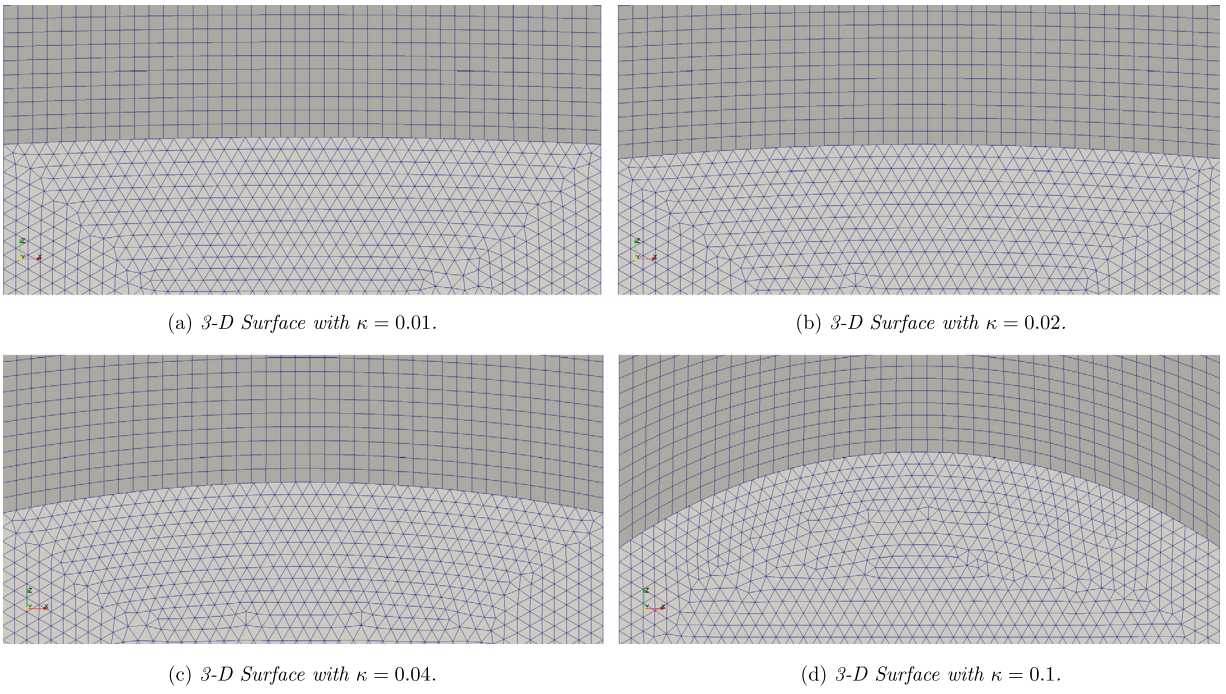
(e) *CR accuracy error vs. iteration.*

(f) *CR conservation error vs. iteration.*

Fig. 2. Function order test conservation and accuracy results. All methods indicated a sensitivity to function order with respect to accuracy. The common-refinement method (CR) conservation results did not have a strong dependence on function order while the moving least square algorithm (MLS) demonstrated better conservation error than spline interpolation.

for larger numbers of transfers the spline interpolation data indicates that the accuracy error becomes less dependent on function order as a function of data transfer iteration.

Perhaps the most interesting outcome of this test is the fact that the conservation error of the common-refinement scheme is not strongly linked to the function order at higher orders. For $P = 2$, we see the best results with the common-refinement method. We expect this because the function in this case is quadratic and the mass and stiffness matrices have a quadratic form and therefore there are sufficient degrees of freedom for the reconstruction. The conservation results also



(a) *3-D Surface with* $\kappa = 0.01$.

(b) *3-D Surface with* $\kappa = 0.02$.

(c) *3-D Surface with* $\kappa = 0.04$.

(d) *3-D Surface with* $\kappa = 0.1$.

Fig. 3. Side view of curved interface meshes used for the curvature test. The geometries were constructed by subtracting a sphere with the given radius of curvature from a box.

often exhibit a single oscillation during the transfer iterations, indicative of a shift from a positive to negative or negative to positive difference in the total integral. Because Eq. (47) is in terms of the absolute value of these integrals, this shift is indicated in the plots by a sharp point in the curve. In general, these results indicate that if accuracy is the only metric of concern for the data transfer, the mesh-free methods or weighted residuals with common-refinement may be used without significant trade-off. If strict conservation is of concern, the common-refinement method is the clear choice with conservation errors measured to be 3–4 orders of magnitude smaller after 1000 iterations for higher order functions. If only a mesh-free approach is feasible, lower order functions are best preserved by the moving least square algorithm while spline interpolation should be used for higher order functions.

3.2. Radius of curvature test

The previous numerical experiment analyzed the performance of the algorithms using a geometry with flat interfaces and sharp features. However, in a multiphysics calculation many geometries in practice will contain some degree of curvature and the resulting approximation of the curved surface by the discrete mesh must be considered when performing the data transfer. In the case of the common-refinement, approximations will be introduced during the construction of the integration mesh when the meshes on either side of the shared interface are projected onto one another. In the case of the mesh-free algorithms, the curvature will affect the interpolation by altering the relationship between the coordinates used to construct the spatial polynomial components. In [16], the authors assessed the common-refinement method for two-dimensional problems with curvature. We propose a similar experiment here, this time in three dimensions and also leveraging the mesh-free algorithms.

This test assesses the performance of the methods as a function of the radius of curvature, κ, of the shared surface where κ is defined as the inverse of the radius of the sphere that fits the surface. Shown in Fig. 3, four problems with identical mesh sizes and varying curvature were generated by intersecting a sphere of the prescribed radius with a cube. Including a flat reference calculation of $\kappa = 0.0$ values of 0.01, 0.02, 0.04, and 0.1 were used for the calculations. The size of these meshes is given in Table 2. As the radius of curvature increases, we expect more errors in both conservation and accuracy for the data transfer operation as meshes on the interface produce a less accurate representation of the curved surface.

For the test, we applied the following cubic function on the tetrahedral mesh (shown in Fig. 3 with the convex surface):

$$u(x, y, z) = 1 + xyz - xy \, . \tag{49}$$

For support, the mesh-free methods used a radius of 1.0 with the mesh spacing ≈0.25 with Eq. (20) for support. Using this reference function, 1000 iterations were performed with the conservation and accuracy errors on the tetrahedral mesh surface at the end of each iteration given in Fig. 4. In general, the common-refinement method performed as expected in

Table 2
Curved interface mesh sizes for the curvature test. The sizes given are for the surfaces in contact.

	Nodes	Faces
$\kappa = 0.0$ Hex	1764	1681
$\kappa = 0.0$ Tet	2024	3882
$\kappa = 0.01$ Hex	1764	1681
$\kappa = 0.01$ Tet	2085	4000
$\kappa = 0.02$ Hex	1764	1681
$\kappa = 0.02$ Tet	2085	4000
$\kappa = 0.04$ Hex	1764	1681
$\kappa = 0.04$ Tet	2160	4146
$\kappa = 0.1$ Hex	1764	1681
$\kappa = 0.1$ Tet	2348	4506

this test with errors on the order of magnitude those reported in [16] for similar radii of curvature in two-dimensions. In addition, we also observed the common-refinement performance to have a strong dependence on the radius of curvature as expected. As the curvature increases, errors in representing the surface geometry grow due to the use of linear Lagrangian elements of degree 1 for the meshes to represent a second-order curved surface. As the geometric errors increase in the surface representation, common-refinement errors grow due to inexact interpolation locations during the assembly of the mass matrix and load vector in the weighted residual problem as well as differences in the surface area of the two meshes on either side of the interface.

The moving least square algorithm demonstrated the best performance in this case with smaller errors in both conservation and accuracy at most levels of curvature. In addition, even the case without curvature had outstanding performance. This is in contrast to the previous experiment where all surfaces had no curvature. However, that experiment also included sharp features whereas these calculations did not. The addition of sharp features, as in the nested cubes problem, results in ill-conditioned least square systems, giving very large errors at those locations and polluting the global error measures. In addition, other than the flat interface case for the moving least square algorithm, the mesh-free errors were relatively insensitive to the curvature of the interface. These results indicate that for data transfer problems with strong curvature on the shared interface, the moving least square algorithm is the best choice with accuracy errors 1–3 orders of magnitude smaller than the other two methods and conservation errors 2–3 orders of magnitude smaller after 1000 iterations for large values of curvature. This can be explained by the fact that unlike the common-refinement method with linear elements, the mesh-free schemes use a second-order representation of the surface, thus accurately representing the curvature.

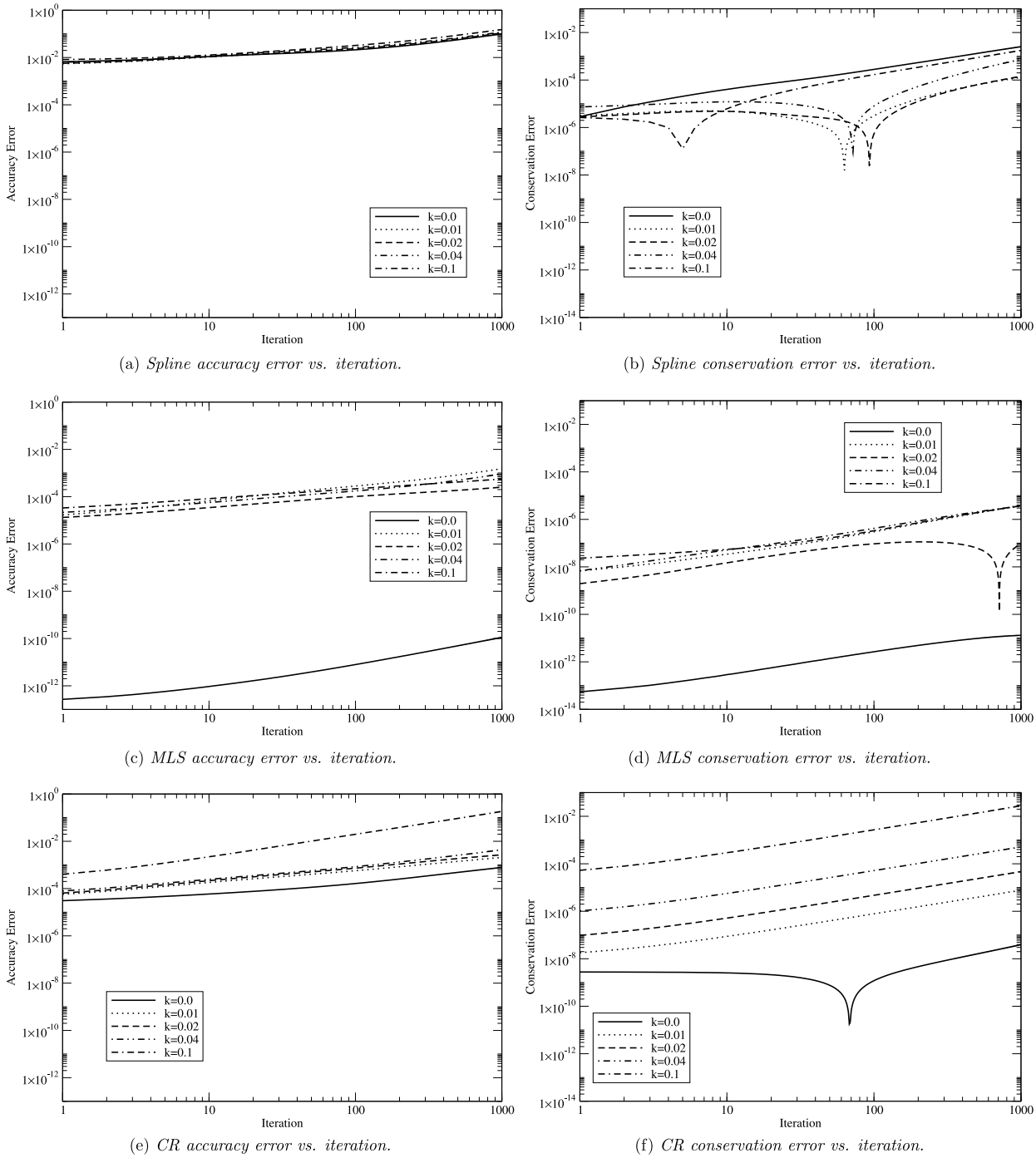
3.3. Discontinuous function test

Functions with effective discontinuities will occur, for example, in problems where a shock is incident on a shared interface or in cases of multiphase flow where the underlying fields are discontinuous due to changing material properties. Depending on how this type of data transfer is performed, large errors may be introduced into the coupled calculation (see [15] for an example of these errors in a coupled shock calculation). Most often, derivative information is not appropriately captured in the data transfer resulting in the appearance of Gibbs' phenomenon [4]. To mitigate this, the authors of [4] introduce Sobolev norm minimization into the weighted residual problem which reduces the effect of Gibbs' phenomenon by instead introducing some numerical diffusion into the problem. As a result, although not guaranteed to be monotone the potentially non-physical over and under-prediction of the function near the discontinuity is reduced with a smoothing of the discontinuity.

Because they do not incorporate extra derivative information into the function, we do not expect the mesh-free methods presented here to accurately transfer a discontinuous function. To test this, we define the following step function:

$$u(x, y, z) = \begin{cases} 1 & : x < 1.0 \\ 2 & : x \geq 1.0 \end{cases} . \tag{50}$$

We used the same tetrahedral and hexahedral grids as the flat interface case from the curvature test in §3.2 but this time with a smaller support radius of 0.5 for the mesh-free methods to better localize the interpolation near the step for a mesh spacing of $\approx$0.25. Again, 1000 data transfer iterations were performed with the results given in Fig. 5 for the mesh free algorithms and the common-refinement scheme with L_2 minimization. For the mesh-free cases, the moving least square method provides a better reconstruction while common-refinement gives the best results in terms of accuracy and conservation. We expect the moving least square scheme to perform better than spline interpolation to a certain extent because moving least square is a relatively local formulation compared to spline interpolation. In the case of spline interpolation, although the support is compact, the coupling matrix includes contributions from all points on the surface. In the moving least square kernels, the independence of each kernel permits a better local representation of the function and thus a better representation of the discontinuity in this test.



(a) *Spline accuracy error vs. iteration.*

(b) *Spline conservation error vs. iteration.*

(c) *MLS accuracy error vs. iteration.*

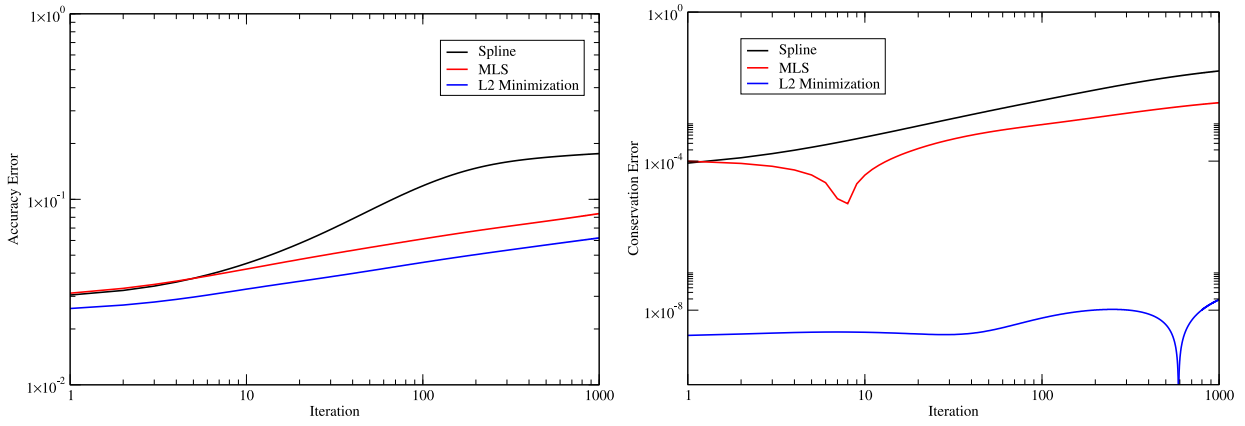
(d) *MLS conservation error vs. iteration.*

(e) *CR accuracy error vs. iteration.*

(f) *CR conservation error vs. iteration.*

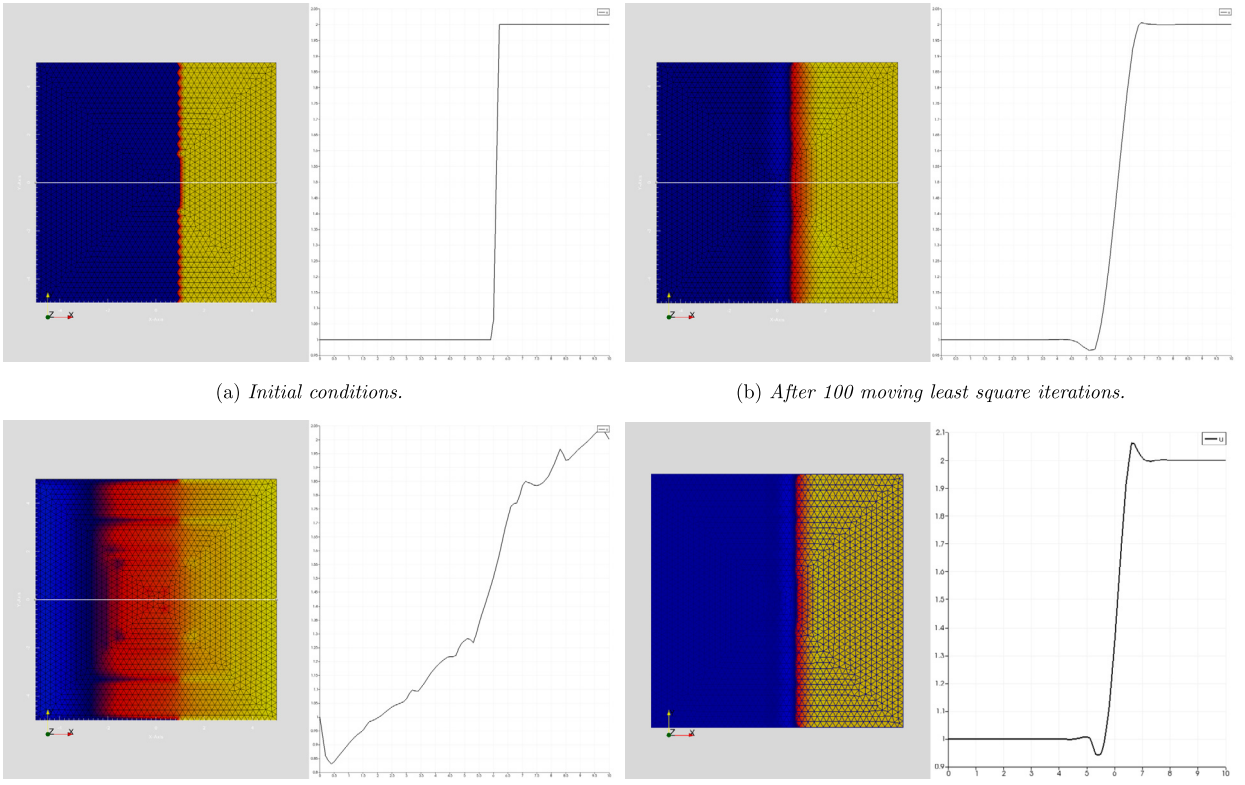
Fig. 4. Curvature test conservation and accuracy results. The common refinement (CR) method indicated a strong sensitivity to radius of curvature for both conservation and accuracy error while the spline interpolation and moving least square (MLS) algorithms did not.

To explore the spatial effects of the transfer algorithms on the step function, the initial conditions of the test along with the mesh-free and common-refinement results after 100 iterations are plotted in tandem with a corresponding line plot in Fig. 6. First, the moving least square method does a surprisingly good job of maintaining the discontinuity after 100 iterations although we begin to see the over and underestimation of the function value indicative of Gibbs' phenomenon. The spline results in this case are more surprising. We observe the discontinuity to have effectively diffused across the interface after 100 iterations. In addition, a banded structure emerges in the solution plot on the left resulting from closely aligned nodes on the surfaces of the triangular and quadrilateral meshes. For the results presented here, common-refinement with



(a) *Accuracy error vs. iteration.* (b) *Conservation error vs. iteration.*

Fig. 5. Discontinuous function test conservation and accuracy results. Moving least square reconstruction (MLS) produced a more accurate a conservative representation of the function than the spline interpolation method.



(a) *Initial conditions.* (b) *After 100 moving least square iterations.*

(c) *After 100 spline interpolation iterations.* (d) *After 100 L_2 minimization iterations.*

Fig. 6. Discontinuous function test. The plots on the right are constructed over the line at $y = 0$ shown on the plots on the left. The moving least square formulation produces less drift in the discontinuity while Gibbs' phenomenon is still observed. Using Sobolev minimization with the common-refinement scheme would mitigate the observed Gibbs' phenomenon in lieu of numerical diffusion.

L_2 minimization gives less drift in the solution than the moving least square scheme but gives slightly larger values from Gibbs' phenomenon. In addition, common-refinement provides a globally conservative transfer independent of the presence of Gibbs phenomenon

Although functional, the current formulation of the moving least square method cannot resolve Gibbs' phenomenon. In situations where this is important and the resulting numerical diffusion of the solution is acceptable or at least preferred to violating the bounds of the function, the common-refinement method should be used in tandem with a Sobolev-minimized

weighted residual scheme for problems of this type. If a mesh-free algorithm is the only option, the moving least square technique significantly outperforms spline interpolation.

4. Mesh-free algorithm parallelization

We next parallelize the spline interpolation and moving least square algorithms. When sparse linear algebra data structures are used, parallelizing these algorithms is fairly straightforward. We begin by first discussing a fast parallel radius search method for arbitrarily distributed point clouds which will serve as the basis for constructing support groups in the interpolation algorithms. We then present the parallel algorithms for both mesh-free methods with a discussion of some implementation details.

4.1. Parallel radius search

The data transfer methods presented in §2.3.2 and §2.3.3 rely on constructing support groups from an arbitrary point cloud within some radius defining the size of that support during the setup phase of the algorithm. In most multiphysics applications, these point clouds will come from nodes of a computational mesh or other similar data structures and will therefore be decomposed in parallel for a large problem. However, we may also assume without loss of generality that for a given parallel process these points will be contained within some bounding box encapsulating a subdomain in the decomposition. We expect that the bounding boxes from all subdomains will overlap only with a small subset of the bounding boxes of other subdomains or perhaps not at all. For example, if the point cloud in a subdomain were constructed from nodes in a mesh partition in that subdomain, these nodes would cover a continuous region of space not occupied by other subdomains in the mesh. This idea immediately leads us to the notion of sparsity in the parallel problem created by the need to only communicate information with those subdomains who are geometric nearest neighbors in the point cloud. For mesh-free interpolation, any given subdomain will then communicate with neighboring subdomains who may potentially contain either source or target control points that provide support to the local subdomain. The end result of this sparsity is a solution to the $m \times n$ problem where it is determined which n target subdomains a source subdomain will communicate with, which m source subdomains a target domain will communicate with, and how much information will be communicated amongst them.

To create support for a given point in the local subdomain, we first gather points from adjacent subdomains that may lie in that point's radius of support and add them to the set of local points. This extended set can then be searched for those points that lie within the support radius of the given point. Algorithm 1 gives a high-level scheme for the parallel radius search. In line 1, the algorithm input is the points, x, for which to determine support, the points from which to construct support, $xSup$, and the support radius, ρ. The inputs x and $xSup$ may be either source or target control points in the interpolation algorithms and it is possible for x and $xSup$ to be the same set of points. In line 2, we determine which subdomains we will be communicating with by solving the $m \times n$ problem, resulting in a communication plan that is then executed to send supporting points to their potentially multiple neighboring subdomains. This occurs within the function **GatherSupportPoints**(), defined in Algorithm 2, which returns a list of support points that were previously in another subdomain. In line 3, this list is combined with the already existing set of support points in the local subdomain and a static kD-tree [26] is constructed from the combined list in line 4 with **CreateKDTree**(). For each local point, we are then able to find those support points that lie within the support radius in logarithmic time using this tree with **RadiusSearch**().

Scalability of the search is dependent on how efficiently supporting points from other subdomains are gathered to the local subdomain. Given in Algorithm 2, **GatherSupportPoints**() achieves this by first computing the bounding box (we use an axis-aligned Cartesian box for simplicity) of the local set of points for which we want to determine support in line 2. In line 3, we expand the local box in all directions by the support radius of the interpolation problem. If any support points are in this box, then they have the potential to contribute to the support of the local domain. We gather all expanded subdomain bounding boxes in line 4. We note here that this is a potential scalability bottleneck for very large problems as we must construct an array the size of the number of subdomains (although we did not observe this for the problems presented in §5). In addition, we note that a Cartesian bounding box data structure in three dimensions consists of six floating pointing numbers, one for the minimum and maximum value in each dimension. After constructing the bounding box around the local support points in line 6, we proceed to check which of the gathered subdomain boxes intersects the local support box. As we may intersect an arbitrary number of these boxes, we must check all of them giving us a linear time operation in terms of the number of subdomains. If we do find an intersection, we store this information such that

Algorithm 1 Parallel radius search.

```
1: Input = x, xSup, ρ
2: neighborPoints = GatherSupportPoints(x, xSup, ρ)
3: xSup = xSup + neighborPoints
4: tree = CreateKDTree(xSup)
5: for all i in x do                          ▷ For each local point, find those supporting it
6:     supportSets[i] = RadiusSearch(tree, ρ, x[i])
7: end for
8: return supportSets                         ▷ Return the points supporting each input point
```

Algorithm 2 GatherSupportPoints().

```
1: Input = x, xSup, ρ
2: localBox = GetLocalBoundingBox(x)
3: expandedBox = ExpandBox(localBox,ρ)
4: subdomainBoxes = Allgather(expandedBox)
5: localSupportBox = GetLocalBoundingBox(xSup)
6: for all p in subdomainBoxes do
7:     if subdomainBoxes[p] intersects localSupportBox then
8:         neighborBoxes = neighborBoxes + subdomainBoxes[p]
9:         neighborIds = neighborIds + p
10:    end if
11: end for
12: for all i in xSup do
13:    for all n in neighborBoxes do
14:        if xSup[i] in neighborBoxes[n] then
15:            sendPoints[n] = sendPoints[n] + xSup[i]
16:            sendDomains[n] = sendDomains[n] + neighborIds[n]
17:        end if
18:    end for
19: end for
20: numNeighborPoints = CreateCommunicationPlan(sendDomains)
21: neighborPoints = ExecuteCommunicationPlan(sendPoints)
22: return neighborPoints
```

we now have a list of subdomains to which we will potentially be sending local support points. We next take this reduced set of subdomains in line 12 and check if each of the local support points is contained within them. Again, a local support point may be in none or all of the adjacent subdomain boxes and therefore we must check all of them. For each support point and subdomain combination, we store the support point and subdomain ids in a flat array. Once we know which local support points must be communicated to which subdomains, we construct a communication plan from this information in line 20 where each subdomain determines the other subdomains from which it will receive information. We execute the communication plan in line 21 to gather the set of points from other subdomains that may potentially provide support in the local subdomain.

Although we use a convenient set of data structures in the Tpetra package of the Trilinos library to build the communication plan in line 20 [27], it is fairly straightforward to construct this plan. As an alternative to the data structures in Trilinos, we could repeat the bounding box intersection procedure but this time gather bounding boxes of the support points (*xSup*) and check intersection with the local bounding box of points for which we desire support (x). We would then have two lists for each subdomain: a list of subdomains to which we will send information and a list of subdomains from which we will receive information. We can then send a single message between pairs of subdomains containing the number of support points that will be sent followed by another message containing the points themselves. In addition, we point out the need for effective geometric sparsity as previously described to maintain scalability (i.e. n $<<$ the number of subdomains in line 13 of Algorithm 2). With this geometric sparsity a particular subdomain will communicate with a group of subdomains much smaller than the total number of subdomains, thus significantly reducing the time complexity of the algorithm and the number of communications required. If this sparsity did not exist, every subdomain would potentially communicate with every other subdomain, resulting in a quadratic algorithm in terms of time complexity, space complexity, and number of communications.

4.2. Parallel spline interpolation

The spline interpolation algorithm defined in §2.3.2 can be parallelized by constructing an efficient parallel representation of the data transfer operator, $\mathbf{H}$, using parallel compressed row storage matrices. In this case, we cannot explicitly construct the entries of the operator due to the inverse operation required in Eq. (34). Rather, we write a function that efficiently produces the action of $\mathbf{H}$ on $\mathbf{f}$ as in Eq. (34).

To begin, we first construct the $\mathbf{C}$ matrix given by Eq. (27) in two parts in the setup phase of the algorithm. First, the input source center coordinates are used to construct the transpose of the polynomial component $\mathbf{P}^T$. We store the transpose because it is row sparse with each row containing a linear polynomial for a given source center. Next, the basis component $\mathbf{M}$ is constructed by first assembling all source centers within the support radius of any source center already on the local process using the parallel radius search in Algorithm 1. The entries of $\mathbf{M}$ are filled out through the basis evaluations using compactly supported functions such as Eq. (20). The structure of $\mathbf{M}$ is inherently sparse due to the compactness of the basis and we only perform the evaluation for those source centers that are within the radius of support.

Using these operators we form the action of $\mathbf{C}^{-1}$ on a vector in the apply stage of the algorithm:

$$\mathbf{y} = \mathbf{C}^{-1}\mathbf{x}\,, \tag{51}$$

which is achieved by solving the following linear system using GMRES [28]:

$$\mathbf{C}\mathbf{y} = \mathbf{x}\,. \tag{52}$$

To leverage GMRES, we only need to provide the action of the operator **C** on a vector. Using the polynomial and basis components of **C** we build the matrix–vector product as:

$$\mathbf{Cx} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{P} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}_\beta \\ \mathbf{x}_\alpha \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{M} \end{bmatrix} \begin{bmatrix} \mathbf{x}_\beta \\ \mathbf{x}_\alpha \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{P} & \mathbf{0} \end{bmatrix}^T \begin{bmatrix} \mathbf{x}_\beta \\ \mathbf{x}_\alpha \end{bmatrix}, \tag{53}$$

where again only **P** has been stored and we have broken **x** into parts. The use of GMRES here is motivated by a few reasons. First, **C** is symmetric positive-indefinite and therefore schemes for symmetric positive-definite systems may not be used. Second, scalable parallel implementations of the algorithm are available in many libraries including the implementation in the Belos package of Trilinos used for this work [27]. We note that for symmetric-indefinite operators such as **C**, MINRES may also be considered for problems of this type. However, our initial experience indicates that GMRES converges in $O(10)$ iterations without preconditioning for random point clouds and therefore we did not explore alternative solution options in this work. However, if iterative performance of the solve inverting C became an issue for certain problems in the future, we would explore both alternative solvers as well as preconditioners appropriate for these types of saddle point problems.

The operator **A** given in Eq. (31) is constructed in identical fashion with polynomial component **Q** assembled from the local target center coordinates and basis component **N** assembled from the corresponding basis evaluations during the setup phase of the algorithm. Note here that we must use the parallel radius search a second time to assemble all source centers that are within the support radius of any given target center contained in the local process. Again, **Q** is sparse in terms of non-zero entries per row and **N** is sparse due to the compact support of the basis. The action of **A** on a vector is then the composite action of the polynomial and basis parts:

$$\mathbf{Ax} = \mathbf{Qx}_\beta + \mathbf{Nx}_\alpha . \tag{54}$$

For a given set of input source and target centers, the polynomial and basis components of **A** and **C** can be constructed once before any interpolations are performed and reused repeatedly as long as the positions of the source and target centers do not change. At each interpolation, the vector **F** is constructed by wrapping the input values of the source function **f** at the given source centers in a parallel vector and then used as the right-hand-side for the GMRES solve in Eq. (52). Eq. (54) is then used to apply **A** to the solution obtained by GMRES resulting in the target function **g** defined at the given target centers.

Because of the structure of the resulting operators, one must be careful when constructing the matrix–vector products to ensure scalability. In particular, we note here that there are two ways to treat parallel matrix–vector products using the polynomial components. The first (version 1) is to store the polynomial components in a compressed row storage matrix because they are row-sparse and to let the underlying matrix data structure handle the multiplication. This approach is very straightforward because no parallel infrastructure is required in the implementation. However, because the matrix is dense in the first few columns, scalability of the matrix–vector product will likely be poor without decomposing it into blocks and treating the polynomial components separately.

Anticipating this issue, we construct a second version of the matrix–vector multiply where we store the polynomials as a set of parallel vectors and carry out the multiplications ourselves in a way that is potentially more efficient. For both **P** and **Q** we generalize a polynomial matrix–vector multiply to have the following form:

$$\begin{bmatrix} \mathbf{R} & \mathbf{0} \end{bmatrix} \mathbf{x} = \mathbf{y} , \tag{55}$$

where the entries in **R** containing the polynomial coefficients occupy the first few columns of the matrix. We store these polynomial coefficients as sets of parallel vectors, one vector for each non-zero column of the matrix. To perform the multiplication we then execute the procedure given in Algorithm 3. We broadcast the first few entries of the vector **x** corresponding to the number of coefficients in the polynomial so that all processes have this information. The rest of the matrix–vector multiply can then proceed locally as long as **R** and **y** have the same row-wise parallel decomposition. If this is not the case, the parallel vector data structure can be used to permute the decomposition of **R** so that they are the same. To complete the construction of **C** given by Eq. (53) we also require the action of a transpose polynomial matrix on a vector. Given in Algorithm 4, we first permute the parallel decomposition of **x** so it is the same as the set of vectors composing **R**. We then locally perform a subset of the matrix–vector multiply and execute a collective reduce to both sum the resulting products across all processes and move the result to the process containing the first few entries of **y**.

Algorithm 3 PolynomialMatrixVectorMultiply().

```
1: Input = R, x, y
2: For a polynomial with p coefficients, broadcast the first p entries in x to all processes
3: for all i in y do
4:     for j = 0..p do
5:         y[i] = R[i][j] × x[j]
6:     end for
7: end for
```

Algorithm 4 TransposePolynomialMatrixVectorMultiply().

```
Input = R, x, y
Permute the parallel decomposition of x to be the same as y
For a polynomial with p coefficients allocate a vector z of size p
for all i in x do
    for j = 0..p do
        z[j] = R[i][j] × x[i]
    end for
end for
Reduce and sum z to the process containing the first p entries of y
for j = 0..p do
    y[j] = z[j]
end for
```

4.3. Parallel moving least square reconstruction

The moving least square algorithm is more easily parallelized than spline interpolation as a global linear solve and composite linear operators are not required. Rather, per the formulation in §2.3.3 the entries of the data transfer operator **H** are explicitly constructed through the solution of local least square problems defined by Eq. (38) in the setup phase of the algorithm and then the operator is simply applied during the apply phase. To construct this problem, we again use the parallel radius search to build the set of source control points supporting each target control point, s_{t_i}, by gathering all source points that are within the support radius of those target points that are owned by the local process. Each moving least square kernel is then computed from the support set of each target center with the solution to that problem containing the non-zero entries of the corresponding row in **H**. We formulate each kernel as a small, dense linear algebra problem and utilize LAPACK and BLAS routines for manipulating the resulting matrices [29]. To construct the kernel in Eq. (38), we solve the following linear system:

$$\mathbf{P}(s_{t_i})^T \mathbf{\Phi}(s_{t_i}) \mathbf{P}(s_{t_i}) \mathbf{X} = \mathbf{P}(s_{t_i})^T \mathbf{\Phi}(s_{t_i}) \,, \tag{56}$$

where now:

$$\mathbf{X} = \left[\mathbf{P}(s_{t_i})^T \mathbf{\Phi}(s_{t_i}) \mathbf{P}(s_{t_i}) \right]^{-1} \mathbf{P}(s_{t_i})^T \mathbf{\Phi}(s_{t_i}) \,, \tag{57}$$

and the kernel for the ith target center t_i is then:

$$\xi(t_i) = \mathbf{p}(t_i)^T \mathbf{X} \,. \tag{58}$$

Rank deficiency is also an issue for the linear problem in Eq. (56) and we use a singular value decomposition with a singular value threshold set to the floating point precision to generate the solution because of this.

The resulting sparsity pattern of **H** is dictated be the number of source control points supporting each target point with the values in entry H_{ij} equal to the component of the solution to the least squares problem for the ith target control point corresponding to the jth source control point. We also note here that although the resulting operator is non-square, it does not have the dense banded structure resulting from the polynomial components of the operators in the spline interpolation method. The entries of **H** are also independent of the data to be transferred and can therefore be constructed once for a given set of source and target control points and reused until those sets of points are modified. For each data transfer operation, **H** is then simply applied to one or multiple vectors depending on the dimensionality of **f**. Although it is more efficient to construct **H** using the spline method because of the small linear problem that must be solved to assemble each row of the moving least square matrix, the data transfer operation is significantly more efficient as no global Krylov solve is required. In addition, the data transfer operation using the moving least square technique does not require the construction of the extra vector **F** as needed by the spline interpolation method and can instead operate directly on a user-provided **f** defined at the source control points.

5. Parallel algorithm results

To demonstrate the effectiveness of the parallel algorithms presented in §4 we perform two simple scaling studies using the spline interpolation and moving least square techniques; one strong and one weak. Although the analysis in §3 was concerned only with the numerical properties of the algorithms for surface transfers, they may be equally applied to volumetric transfers [12]. For both scaling studies, the problem setup consists of random point clouds for both the source and target centers defined over a 3-dimensional volume. Each process defines the Cartesian bounds of its local domain and then that domain is sampled randomly for the requested number of control points with different random sets for the source and target centers. To ensure communication amongst the domains and to attempt construct a communication scenario representative of a typical partitioned multiphysics problem, the domains containing the source points were expanded such that they intersected with up to 27 of their nearest neighbors and fewer if they were on the boundary of the problem. This creates a relatively sparse communication pattern that we might expect from physics of different partitioning. In addition,

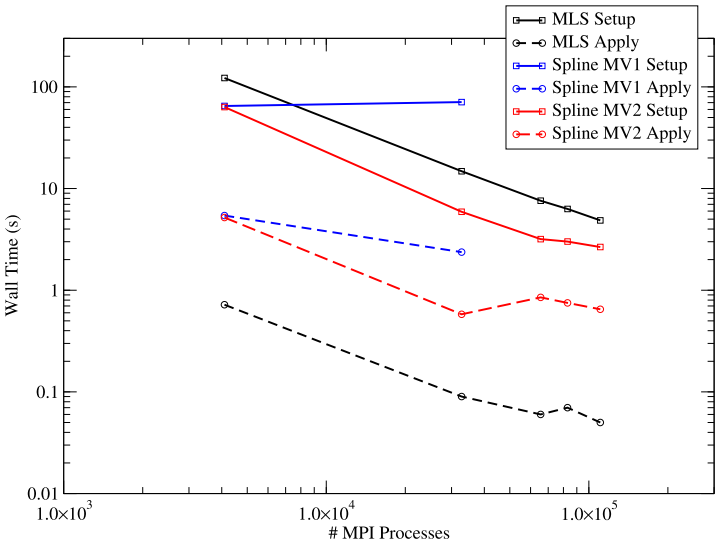


Fig. 7. Timing results for strong scaling study with 1×10^9 global points in the source and target center clouds. Timings reported are the maximum wall time in seconds over all processes. Spline MV1 indicates timings for spline interpolation with parallel-matrix vector multiply version 1 and Spline MV2 with version 2.

we chose to expand the source domains to ensure communication amongst domains for both parallel search operations required in the spline interpolation method. To further ensure a minimal amount of communication between identical processes, the process ids were inverted such that the source domain on process 1 would cover the same region of space as the target domain on process P for a problem with P parallel processes. For both data transfer methods, Wu's C^4 function given in Eq. (20) was used as the supporting compact radial basis with a supporting radius of $1.5\times$ the average spacing of the random source and target points in the domain.

We acknowledge here that point clouds resulting from arbitrary physics discretizations will likely not be as load balanced and may have communication patterns that are more sparse (communicating with fewer than 27 neighboring domains) or dense (communicating with more than 27 neighboring domains). However, these scaling studies will indicate the performance of the algorithms in typical strong and weak scaling environments as well as indicate how little work a process can do before latency begins to dominate when using an MPI implementation of the algorithms [30]. In addition, we emphasize in our discussion of the results the performance of the setup portion of the algorithms as this is always the dominating cost when compared to applying the data transfer operator. All scaling studies were performed on the Titan Cray XK7 machine at the Oak Ridge Leadership Computing Facility using an implementation of the algorithms contained in the DataTransferKit library [31,32]. Results for the scaling study are reported in terms of the number of MPI processes with one MPI process per core and all 16 cores leveraged per node on Titan. The GPU accelerators on Titan were not enabled for this work. Parallel efficiencies reported here are calculated based on the analysis techniques provided in [33]. A single calculation was performed for each data point in the scaling studies.

5.1. Strong scaling

Strong scaling indicates the parallel efficiency of an algorithm for a problem of fixed global size. As the number of parallel processes used to solve the problem is increased, the local problem size decreases. Eventually, the local problem size will be small enough that latency costs from communication will dominate and the algorithm is no longer efficient. For the first strong scaling study we choose a global size of 1×10^9 for the source and target control point clouds. Fig. 7 plots the timing results for the data transfer operator construction (setup) and the data transfer operator apply phases of the algorithm with the time given the maximum reported over all processes in seconds. The raw data for these calculations is tabulated in Table 3, Table 4, and Table 5. As is typical for these types of algorithms, construction of the data transfer operator and subsequent communication plan in the setup phase is 1–2 orders of magnitude larger than the apply operation. The results for version 1 of the spline interpolation are indicative of a poorly scaling algorithm with 5% efficiency reported for the setup calculation and 28% efficiency reported for the apply calculation at only 32,769 MPI processes. Due to this poor efficiency at lower core counts, we elected not to extend the analysis to higher core counts for the spline interpolation algorithm. Version 2 of the spline interpolation algorithm demonstrated excellent strong scaling with over 88% efficiency reported at 110,592 MPI processes. Efficiency results for the moving least square algorithm were excellent as well with a strong scaling efficiency over 93% at 110,592 MPI processes for the setup operation. For some of the data points, strong scaling efficiencies above 100% are observed. This behavior is quite common for scaling studies of this type and is a result of the local problem shrinking and more efficiently utilizing the cache, thus decreasing the on-process runtime and improving efficiency [33].

Table 3
Strong scaling data for the spline interpolation method using version 1 of the polynomial parallel matrix–vector multiply. Timings reported are the maximum wall time in seconds over all processes.

# MPI processes	Local size	Global size	Setup time	Apply time	GMRES iters
4096	244,140	999,997,440	64.90	5.42	8
32,768	30,517	999,981,056	70.80	2.37	8

Table 4
Strong scaling data for the spline interpolation method using version 2 of the polynomial parallel matrix–vector multiply. Timings reported are the maximum wall time in seconds over all processes.

# MPI processes	Local size	Global size	Setup time	Apply time	GMRES iters
4096	244,140	999,997,440	63.3	5.19	8
32,768	30,517	999,981,056	5.91	0.58	8
65,536	15,258	999,948,288	3.18	0.85	10
82,944	12,056	999,972,864	3.01	0.75	10
110,592	9042	999,972,864	2.66	0.65	8

Table 5
Strong scaling data for the moving least square reconstruction method. Timings reported are the maximum wall time in seconds over all processes.

# MPI processes	Local size	Global size	Setup time	Apply time
4096	244,140	999,997,440	122.18	0.72
32,768	30,517	999,981,056	14.8	0.09
65,536	15,258	999,948,288	7.59	0.06
82,944	12,056	999,972,864	6.30	0.07
110,592	9042	999,972,864	4.86	0.05

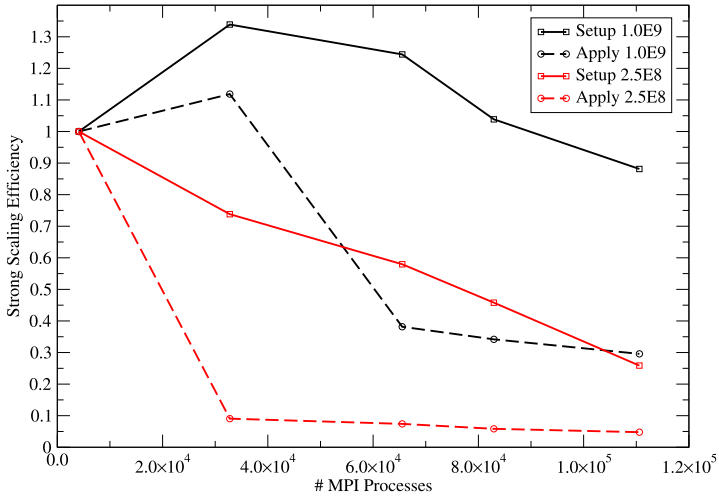


Fig. 8. Spline interpolation version 2 strong scaling efficiencies for global sizes of 1×10^9 and 2.5×10^8 points in the source and target center clouds. Scaling degrades with a smaller global problem size due to latency overcoming the lack of on-process work while larger problem sizes with more local work scale well.

A second strong scaling study was performed with a smaller global problem size to demonstrate the loss of efficiency of the algorithms with less work per process. A global size of 2.5×10^8 for the source and target center clouds was used for these calculations. The strong scaling efficiency results are plotted in Fig. 8 and Fig. 9 for both point cloud sizes. These results indicate that decreasing the global problem size by a factor of four results in a reduction of parallel efficiency to only 54% for the moving least square algorithm and 25% for the spline interpolation algorithm at 110,592 MPI processes for the setup operation. For this case the local cloud size was 2260 points and therefore we expect good strong parallel scalability for problems at least 2×–4× larger with local cloud sizes of 5,000–10,000 points or more.

5.2. Weak scaling

Weak scaling is used to indicate the parallel of efficiency of an algorithm when the local problem size is fixed and the global size increases as a function of processor count. Efficiency is poor in weak scaling environments when the parallel

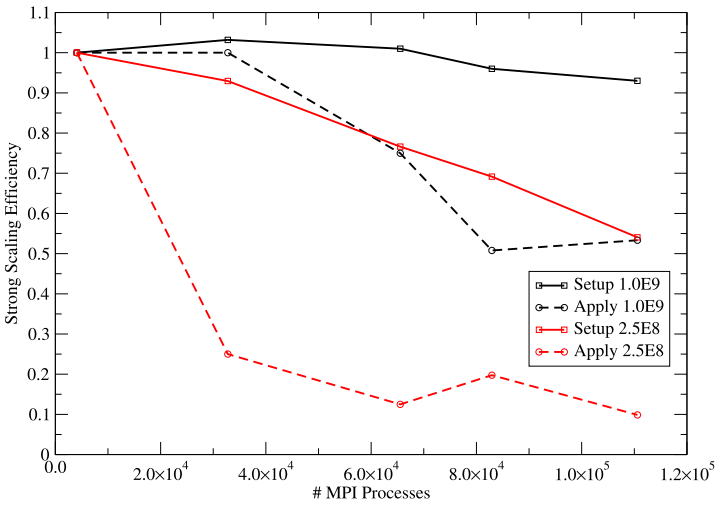


Fig. 9. Moving least square algorithm strong scaling efficiencies for global sizes of 1×10^9 and 2.5×10^8 points in the source and target center clouds. Scaling degrades with a smaller global problem size due to latency overcoming the lack of on-process work while larger problem sizes with more local work scale well.

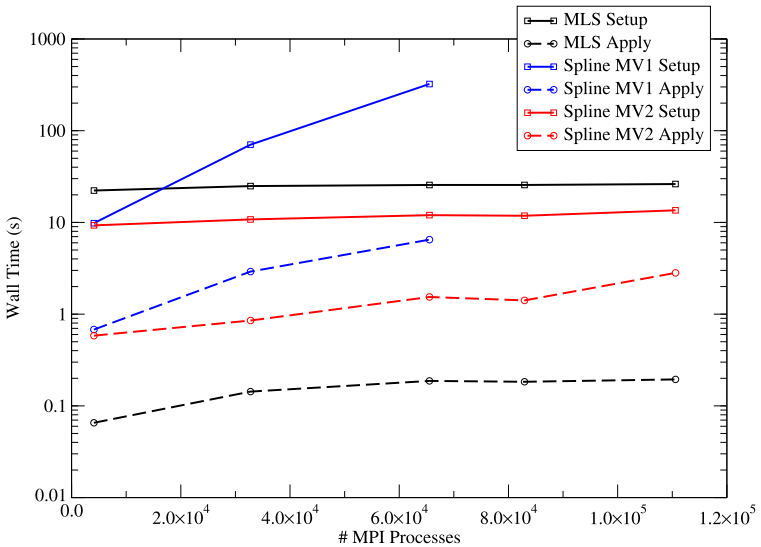


Fig. 10. Timing results for weak scaling study with 5×10^4 local points in the source and target center clouds. Timings reported are the maximum wall time in seconds over all processes. Spline MV1 indicates timings for spline interpolation with parallel-matrix vector multiply version 1 and Spline MV2 with version 2.

Table 6
Weak scaling data for the spline interpolation method using version 1 of the polynomial parallel matrix–vector multiply. Timings reported are the maximum wall time in seconds over all processes.

# MPI processes	Local size	Global size	Setup time	Apply time	GMRES iters
4096	50,000	204,800,000	9.81	0.67	8
32,768	50,000	1,638,400,000	70.48	2.92	8
65,536	50,000	3,276,800,000	322.82	6.49	10

algorithm does not scale logarithmically or better with processor count. The first weak scaling study fixes the local size of the source and target clouds at 50,000 points. Fig. 10 gives the timing results for these calculations with the raw data tabulated in Table 6, Table 7, and Table 8. Timings for the moving least square algorithm and version 2 of the spline algorithm again indicate excellent scaling for the setup operation with a weak scaling efficiency of more than 84% and 68% respectively at 110,592 MPI processes with over 10 billion global points in the combined source and target center point clouds.

Table 7
Weak scaling data for the spline interpolation method using version 2 of the polynomial parallel matrix–vector multiply. Timings reported are the maximum wall time in seconds over all processes.

# MPI processes	Local size	Global size	Setup time	Apply time	GMRES iters
4096	50,000	204,800,000	9.289	0.583	8
32,768	50,000	1,638,400,000	10.779	0.853	8
65,536	50,000	3,276,800,000	12.012	1.539	10
82,944	50,000	4,147,200,000	11.828	1.412	10
110,592	50,000	5,529,600,000	13.55	2.82	8

Table 8
Weak scaling data for the moving least square reconstruction method. Timings reported are the maximum wall time in seconds over all processes.

# MPI processes	Local size	Global size	Setup time	Apply time
4096	50,000	204,800,000	22.28	0.07
32,768	50,000	1,638,400,000	24.91	0.14
65,536	50,000	3,276,800,000	25.60	0.19
82,944	50,000	4,147,200,000	25.62	0.18
110,592	50,000	5,529,600,000	26.24	0.20

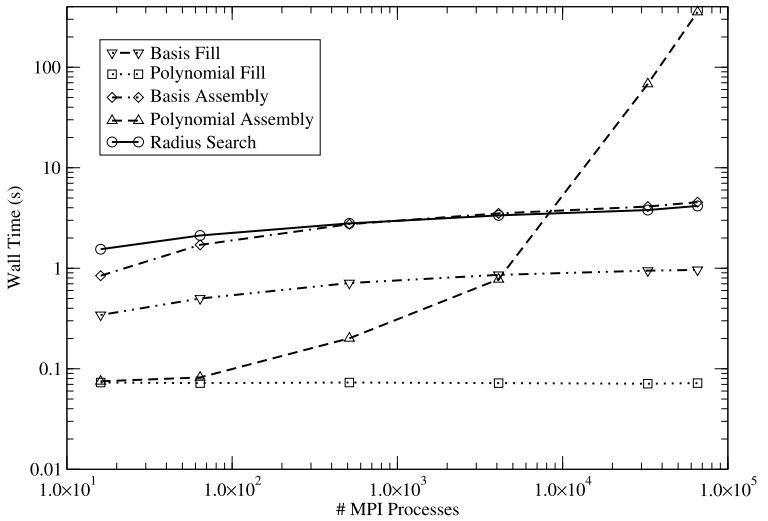


Fig. 11. Timing results for individual components of the spline interpolation algorithm setup operation with matrix–vector multiply version 1 for the polynomial basis components. Basis fill is the time required to fill the entries of the basis components of the operators. Polynomial fill is the time required to fill the polynomial components of the operators. Basis assembly is the time required to do parallel assembly and construct a parallel matrix–vector multiply communication plan for the basis components of the operators. Polynomial assembly is the time required to do parallel assembly and construct a parallel matrix–vector multiply communication plan for the polynomial components of the operators. Radius search is the time required to perform the parallel radius search and support group construction for both operators.

The spline interpolation method with version 1 of the matrix–vector multiply again performs poorly with a 3% weak scaling efficiency reported at 65,536 MPI processes. We elected not to test the implementation at higher core counts due to the poor efficiency. To address why the scaling is poor for this version of the algorithm we timed it with more granularity. Fig. 11 reports the wall times for individual pieces of the spline interpolation algorithm setup operation computed in an additional scaling study. As expected, the main scalability bottleneck here is the parallel assembly and construction of the parallel matrix–vector multiply communication plan for the polynomial components of the interpolation operators, **P** and **Q**. Looking at Eq. (28) and Eq. (32), although these matrices are sparse by construction with very few non-zero entries per row, they are extremely inefficient in parallel when using a compressed row storage format because each non-zero entry occupies the first four columns of the row. Because of this, in order to perform a matrix–vector multiplication each process containing a row of the matrix must communicate with the process containing the first four entries of the vector on which the matrix is acting. This effectively produces a one-to-all communication pattern and leads to the lack of scalability. Furthermore, this issue also manifests itself in the apply operation where numerous matrix–vector multiply operations are performed during the course of the GMRES solve required for each data transfer. Version 2 of the algorithm bypasses this bottleneck by taking advantage of the structure of the polynomial operators and replacing the one-to-all communication pattern with a single collective communication.

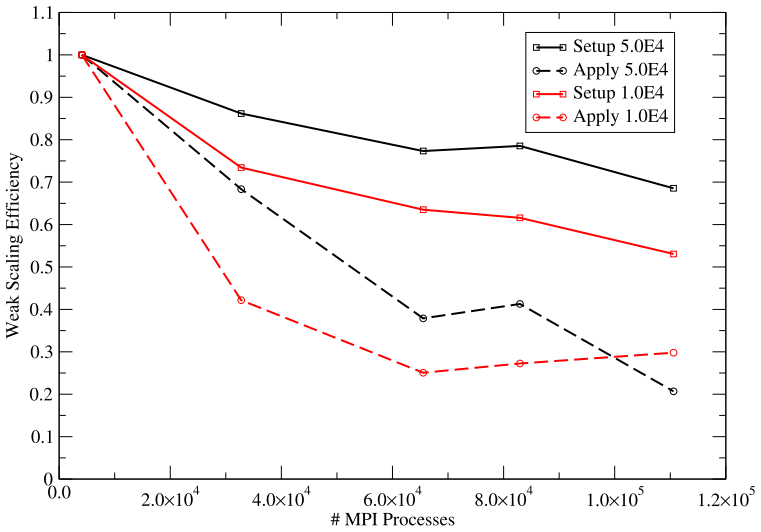


Fig. 12. Spline interpolation version 2 weak scaling efficiency for problem sizes of 5×10^4 and 1×10^4 points in the local source and target center clouds. Latency affects the apply operation scaling more significantly in a weak scaling environment than a strong scaling environment.

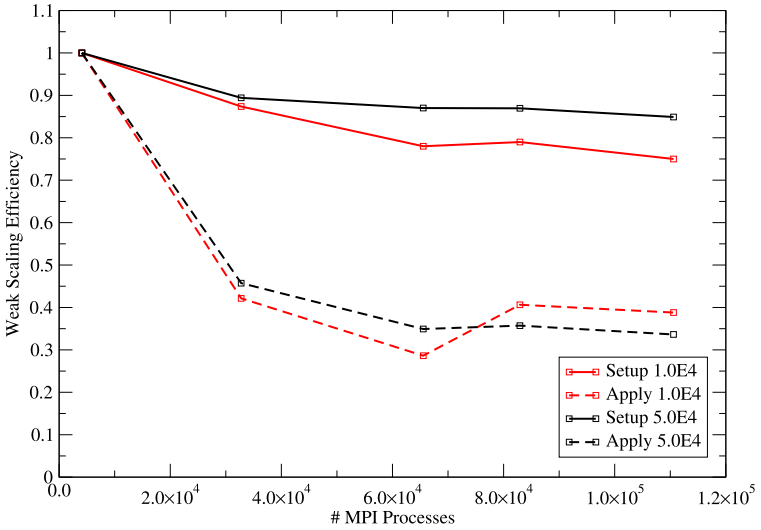


Fig. 13. Moving least square algorithm weak scaling efficiency for problem sizes of 5×10^4 and 1×10^4 points in the local source and target center clouds. Latency affects the apply operation scaling more significantly in a weak scaling environment than a strong scaling environment.

A second weak scaling study was performed with a smaller local problem size of 10,000 points to study how much latency was hidden by a larger amount of on-process work. Fig. 12 and Fig. 13 give the weak scaling efficiencies for these calculations along with those computed from the first set of calculations with a local problem size of 50,000 points. We report an efficiency of over 75% for the moving least square calculation and over 53% for the spline interpolation at 110,592 MPI processes with a global size of over 1.1 billion points for both the source and target clouds. The loss of efficiency here is not as significant as that observed for strong scaling case, although we did not shrink the local problem size to the level of that in the strong scaling study. These results also indicate good weak scaling at large core counts for problems with $O(10{,}000)$ points in the local clouds.

6. Conclusion

In this paper, we have analyzed mesh-free data transfer algorithms for several cases that are common in partitioned multiphysics problems. In addition, we have compared these methods to the common-refinement method – a variant of the weighted residual technique with conservative and accurate properties. Our studies indicate that mesh-free algorithms, in particular spline interpolation and moving least square reconstruction, are complementary to weighted residual methods in three dimensions. For data transfer across a flat shared interface with sharp features, the mesh-free methods are competitive

in terms of data transfer accuracy while the common-refinement scheme is superior in measures of conservation. When fields are transferred on a curved interface, the moving least square algorithm should be chosen with better conservation and accuracy properties observed at large values of curvature. If large gradients or even discontinuities are present in the solution it can be advantageous to choose the common-refinement scheme due to the numerical flexibility of the weighted residual formulation and the relative invariance of the conservation measure to function order.

We have also parallelized the compactly supported mesh-free algorithms by formulating them in a way that we can leverage sparse linear algebra data structures designed for high performance computing environments. Using a set of scaling studies and a leadership class computing facility, our results indicate that the algorithms we have developed are scalable in both strong and weak scaling environments using $O(100{,}000)$ MPI processes and point clouds with billions of degrees of freedom. In addition, we learned that these algorithms are scalable only to a point with a minimum local problem size of at least 10,000 points required for good performance. The mesh-free algorithms are also straightforward to use without the requirement of a computational mesh and the associated topological information. Instead, they require a minimal input of point clouds without knowledge of the global parallel decomposition and the corresponding function values defined on the points. This reduced set of inputs can greatly facilitate coupling in a multiphysics calculation when using a partitioned approach. As an area of future work, it is desirable to extend the mesh-free methods to more accurately resolve Gibbs' phenomenon as observed in §3.3.

Acknowledgements

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

This research was supported by the Consortium for Advanced Simulation of Light Water Reactors (www.casl.gov), an Energy Innovation Hub (http://www.energy.gov/hubs) for Modeling and Simulation of Nuclear Reactors under U.S. Department of Energy Contract No. DE-AC05-00OR22725.

References

[1] C. Michler, E. van Brummelen, S. Hulshoff, R. de Borst, The relevance of conservation for stability and accuracy of numerical methods for fluid–structure interaction, Comput. Methods Appl. Mech. Eng. 192 (37) (2003) 4195–4215.
[2] J.R. Cebral, R. Lohner, Conservative load projection and tracking for fluid–structure problems, AIAA J. 35 (4) (1997) 687–692.
[3] C. Farhat, M. Lesoinne, P. LeTallec, Load and motion transfer algorithms for fluid/structure interaction problems with non-matching discrete interfaces: momentum and energy conservation, optimal discretization and application to aeroelasticity, Comput. Methods Appl. Mech. Eng. 157 (1998) 95–114.
[4] X. Jiao, M.T. Heath, Common-refinement-based data transfer between non-matching mesh in multiphysics simulation, Int. J. Numer. Methods Biomed. Eng. 61 (2004) 2402–2427.
[5] P. Farrell, M. Piggott, C. Pain, G. Gorman, C. Wilson, Conservative interpolation between unstructured meshes via supermesh construction, Comput. Methods Appl. Mech. Eng. 198 (2009) 2632–2642.
[6] A. Jain, X. Jiao, Overlaying surface meshes: extension and parallelization, in: M. Brewer, D. Marcum (Eds.), Proceedings of the 16th International Meshing Roundtable, Springer, Berlin, Heidelberg, 2008, pp. 347–363.
[7] A. Beckert, H. Wendland, Multivariate interpolation for fluid–structure-interaction problems using radial basis functions, Aerosp. Sci. Technol. 5 (2) (2001) 125–134.
[8] G. Quaranta, P. Masarati, P. Mantegazza, A conservative mesh-free approach for fluid–structure interface problems, in: Int. Conf. on Computational Methods for Coupled Problems in Science and Engineering, 2005.
[9] T. Rendall, C. Allen, Unified fluid–structure interpolation and mesh motion using radial basis functions, Int. J. Numer. Methods Biomed. Eng. 74 (2008) 1519–1559.
[10] T. Rendall, C. Allen, Improved radial basis function fluid–structure coupling via efficient localized implementation, Int. J. Numer. Methods Biomed. Eng. 78 (2009) 1188–1208.
[11] W. Costin, C. Allen, Radial basis function interpolation for data transfer across a mesh interface, in: 20th AIAA Computational Fluid Dynamics Conference, 2011.
[12] T. Rendall, C. Allen, Evaluation of RBFs for volume data interpolation in CFD, Int. J. Numer. Methods Biomed. Eng. 72 (2013) 748–769.
[13] M. Cordero-Gracia, M. Gomez, E. Valero, A radial basis function algorithm for simplified fluid–structure data transfer, Int. J. Numer. Methods Biomed. Eng. (2014).
[14] H.B.A. de Boer, A.H. van Zuijlen, Review of coupling methods for non-matching meshes, Comput. Methods Appl. Mech. Eng. 196 (2007) 1515–1525.
[15] R. Jaiman, X. Jiao, P. Geubelle, E. Loth, Assessment of conservative load transfer for fluid–solid interface with non-matching meshes, Int. J. Numer. Methods Biomed. Eng. 64 (2005) 2014–2038.
[16] R. Jaiman, X. Jiao, P. Geubelle, E. Loth, Conservative load transfer along curved fluid–solid interface with non-matching meshes, J. Comput. Phys. 218 (2006) 372–397.
[17] R. Johnson, G. Hansen, C. Newman, The role of data transfer on the selection of a single vs. multiple mesh architecture for tightly coupled multiphysics applications, Appl. Math. Comput. 217 (2011) 8943–8962.
[18] G. Wellmer, L. Reimer, H. Flister, M. Behr, J. Ballmann, A comparison of fluid/structure coupling methods for reduced structural models, in: Management & Minimization of Uncertainty & Errors, vol. 122, 2013, pp. 181–218.
[19] R. Yokota, L. Barba, M. Knepley, PetRBF – a parallel $O(N)$ algorithm for radial basis function interpolation with Gaussians, Comput. Methods Appl. Mech. Eng. 199 (2010) 1793–1804.
[20] S. Cuomo, A. Galletti, G. Giunta, A. Starace, Surface reconstruction from scattered point via RBF interpolation on GPU, in: 2013 Federated Conference on Computer Science and Information Systems, FedCSIS, IEEE, 2013, pp. 433–440.
[21] M. Cordero-Gracia, M. Gomez, J. Ponsin, E. Valero, An interpolation tool for aerodynamic mesh deformation problems based on octree decomposition, Aerosp. Sci. Technol. 23 (2012) 93–107.
[22] X. Jiao, M.T. Heath, Overlaying surface meshes, Part I: Algorithms, Int. J. Comput. Geom. Appl. 14 (6) (2004) 379–402.
[23] X. Jiao, M.T. Heath, Overlaying surface meshes, Part II: Topology preservation and feature matching, Int. J. Comput. Geom. Appl. 14 (6) (2004) 403–419.
[24] M. Buhmann, Radial basis functions, Acta Numer. (2000) 1–38.

[25] R. Ahrem, A. Beckert, H. Wendland, A meshless spatial coupling scheme for large-scale fluid–structure-interaction problems, Comput. Model. Eng. Sci. 12 (2) (2006) 121.
[26] J. Bentley, Multidimensional binary search trees used for associative searching, Commun. ACM 18 (1975) 509–517.
[27] M.A. Heroux, R.A. Bartlett, V.E. Howle, R.J. Hoekstra, J.J. Hu, T.G. Kolda, R.B. Lehoucq, K.R. Long, R.P. Pawlowski, E.T. Phipps, A.G. Salinger, H.K. Thornquist, R.S. Tuminaro, J.M. Willenbring, A. Williams, K.S. Stanley, An overview of the Trilinos project, ACM Trans. Math. Softw. 31 (Sept. 2005) 397–423.
[28] Y. Saad, M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (July 1986) 856–869.
[29] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK Users' Guide, third ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999.
[30] M.P. Forum, MPI: a message-passing interface standard, Tech. rep., Knoxville, TN, USA, 1994.
[31] S. Slattery, P. Wilson, R. Pawlowski, The Data Transfer Kit: a geometric rendezvous-based tool for multiphysics data transfer, in: International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering, M&C 2013, 2013, pp. 5–9.
[32] DataTransferKit GitHub website, https://github.com/ORNL-CEES/DataTransferKit.
[33] D.E. Keyes, How scalable is domain decomposition in practice?, in: Proceedings of the 11th International Conference on Domain Decomposition Methods, 1999.