Journal of Computational Physics **176,** 483–506 (2002)
doi:10.1006/jcph.2002.6999, available online at http://www.idealibrary.com on IDEAL®

A Parallelized Meshfree Method with Boundary Enrichment for Large-Scale CFD

Lucy T. Zhang, Gregory J. Wagner, and Wing K. Liu[1]

Department of Mechanical Engineering, Northwestern University, 2145 Sheridan Road, Evanston, Illinois 60208

E-mail: l-zhang2@northwestern.edu; g-wagner@northwestern.edu; w-liu@northwestern.edu

Received December 18, 2000; revised August 13, 2001

A parallel computational implementation of a meshfree method—the reproducing kernel particle method (RKPM)—is used for 3-D implicit CFD analysis. A novel procedure for implementing the essential boundary condition using the hierarchical enrichment method is presented. Using this enrichment along the essential boundaries produces results that more closely match experimental and analytical results for a flow past a cylinder problem than does either the finite-element method or other meshfree methods that require matrix inversion for the application of essential boundary conditions. This technique also allows the efficient parallelization of the algorithm and leads to higher parallel speedups than do other boundary condition implementations, many of which involve inherently serial steps; this is important, because the expense of meshfree computations makes parallelization crucial for large-size problems. The performance of the parallelization technique and the accuracy of the implicit CFD algorithm are demonstrated in two example problems.

Key Words: parallel meshfree methods; finite elements; boundary conditions; CFD.

1. INTRODUCTION

The reproducing kernel particle method (RKPM) is one of the meshfree computational methods that have been developed in recent years [1, 3, 5, 9, 15, 16]. These methods have advantages over traditional finite-element methods in their ability to handle large deformation problems without mesh distortion, and in their solution accuracy due to the large domain of influence covered by each node. Due to flexibility in constructing the conforming shape functions to meet specific needs for different applications, it has been proven that RKPM can enhance numerical performance in CFD problems [10, 20, 21]. The derivation of the RKPM interpolant and some of its characteristics are outlined in Section 2.

[1] Fax: (847)491-3915.

0021-9991/02 $35.00

The explicit dynamic analysis of the parallel computational implementation for RKPM was first presented by Danielson *et al*. [8]. By contrast, in the current paper an implicit analysis is presented. The algorithm includes computing residual vectors and using Newton's method iteratively at each time step. The time step can be chosen to be significantly larger than for explicit analysis for stability reasons.

One difficulty encountered with meshfree methods is the lack of a straightforward approach for applying essential boundary conditions. This stems from the fact that meshfree shape functions do not interpolate at the nodes, i.e., for a set of shape functions, $\Phi(x)$,

$$\Phi_I(x_J) \neq \delta_{IJ} \tag{1}$$

and the value of the approximation at a node is not given by nodal parameter values.

Although several methods for the application of essential boundary conditions have been suggested in the literature (e.g. [11, 20]), most of these are inherently serial operations and are unsuitable for large-scale parallel implementations. In [8], the essential boundary condition was implemented on a single processor instead of multiple processors. The parallel computations of compressible fluid flows by Günther *et al*. [11] also use just one processor to apply boundary conditions, due to the difficulty of parallelizing the d'Alembert's principle scheme used. In that paper, the difficulties were emphasized and a procedure was sketched for a parallel algorithm.

We outline in Section 3.2 a method for applying boundary conditions that utilize a projection of the meshfree shape functions onto a small set of finite-element shape functions. In this method, we use a complete set of meshfree shape functions that coexist with finite elements only on the boundary. This allows the numerical properties of the meshfree basis to extend throughout the domain, all the way to the boundary, while boundary conditions can be applied as for finite elements; a bridging term that we define handles the interaction between the two sets of basis functions.

The method was first presented by Wagner and Liu in [22] and was demonstrated in the solution of a 2-D Laplace equation. This paper extends the method to solve the 3-D Navier–Stoke equations. Its major advantage is the avoidance of matrix inversion or transformation for essential boundary condition application, as is required in many approaches. It provides an efficient algorithm especially for parallel implementation.

Because of the large influence domain of the nodes and the complexity of calculating meshfree shape functions, computation using RKPM and other meshfree methods is more intensive than with standard finite elements, especially for large-size problems. With the common availability of massively parallel computing systems, parallel algorithms for meshfree computations are needed to maximize their effectiveness. In this paper we introduce a parallel computation scheme. The procedure starts with a preprocessing phase that requires the partition of the domain nodes and integration points between the processors. Each processor calculates the residual of the equations of motion for its own portion of the domain based on the discretized Navier–Stokes equations derived in Section 3. Nodal data is communicated using a message passing interface (MPI) at each iteration to update shared data on all processors. Communication operations are needed in the parallel meshfree algorithms between each integration point and its neighboring nodes. After solving the equations on separate processors, the postanalysis gathers all the information into a single output file for postprocessing. The detailed communication structures and an outline of the parallel implicit CFD algorithm are described in Section 4.

Accuracy and speedup of the method are demonstrated using two example problems in Section 5.

2. THE REPRODUCING KERNEL PARTICLE METHOD FORMULATION

The kernel estimate was first introduced in the smooth particle hydrodynamics method (SPH) [18], in which the kernel estimate of a function $u(\mathbf{x})$ is

$$u^K(\mathbf{x}) = \int_\Omega \phi_a(\mathbf{y} - \mathbf{x}) u(\mathbf{y})\, d\mathbf{y}. \tag{2}$$

Note that if the kernel function $\phi_a(\mathbf{y} - \mathbf{x})$ is the delta function $\delta(\mathbf{y} - \mathbf{x})$, then $u^K(\mathbf{x}) = \mathbf{u}(\mathbf{x})$. Computation of $u^K(\mathbf{x})$ on a discrete set of nodes involves two approximations: the continuous integral is replaced by a summation over nodes, and the kernel function is chosen to be some approximation to a delta function (e.g., a spline or Gaussian) with compact support.

The kernel function $\phi_a(\mathbf{x})$ can be generated from a canonical function $\phi(\mathbf{x})$ through the dilation parameter a, which controls the width of the function's support:

$$\phi_a(\mathbf{x}) = \frac{1}{a}\phi\left(\frac{\mathbf{x}}{a}\right). \tag{3}$$

An example of the one-dimensional kernel function is the cubic spline function:

$$\phi(x) = \begin{cases} \frac{2}{3} - x^2\left(1 - \frac{|x|}{2}\right), & 0 \le |x| < 1, \\ \frac{1}{6}(2 - |x|)^3, & 1 \le |x| \le 2, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

The discrete form of (2) does not ensure an accurate estimate of $u(\mathbf{x})$. To impose higher order consistency conditions on the kernel estimate, Liu *et al.* [16] proposed a reproducing kernel approximation by introducing a correction function to the kernel estimate,

$$u^h(\mathbf{x}) = \sum_I C(\mathbf{x}; \mathbf{x}_I - \mathbf{x}) \phi_a(\mathbf{x}_I - \mathbf{x}) u(\mathbf{x}_I) \Delta V_I, \tag{5}$$

where $u^h(\mathbf{x})$ is the "reproduced" version of $u(\mathbf{x})$. The summation in (5) is taken over a set of discrete nodes I, and ΔV_I is an appropriate nodal weight, e.g., the spatial volume associated with node I. We find that the exact choice of ΔV_I has little effect on the computation as long as the sum of the nodal weights equals the total volume of the domain. The correction function $C(\mathbf{x}; \mathbf{x}_I - \mathbf{x})$ is included in (5) in order to recover the accuracy that is lost when (2) is approximated discretely on a bounded domain; it is further defined below. The superscript h represents a characteristic nodal spacing and is in general chosen to be proportional to a in Eq. (3). As h and a both go to zero, $u^h(\mathbf{x}) \to u(\mathbf{x})$.

The correction function $C(\mathbf{x}; \mathbf{y} - \mathbf{x})$ is chosen such that, given some order of accuracy N, $u^h(\mathbf{x}) = u(\mathbf{x})$ if $u(\mathbf{x})$ is a polynomial of order less than or equal to N. We take C to have the form

$$C(\mathbf{x}; \mathbf{y} - \mathbf{x}) = \mathbf{b}^T(\mathbf{x}) \mathbf{P}(\mathbf{y} - \mathbf{x}), \tag{6}$$

where $\mathbf{P}(\mathbf{y}-\mathbf{x})$ is the vector of monomial basis functions of order less than or equal to N and $\mathbf{b}$ is a vector of coefficients to be determined:

$$\mathbf{P}^T(\mathbf{y}-\mathbf{x}) = [1, x_1 - y_1, x_2 - y_2, x_3 - y_3, (x_1 - y_1)^2, \ldots, (x_3 - y_3)^N], \tag{7a}$$

$$\mathbf{b}^T = [b_1(\mathbf{x}), b_2(\mathbf{x}), \ldots]. \tag{7b}$$

The coefficients $b_i(\mathbf{x})$ are determined by imposing the Nth order completeness requirement. Substituting (6) into (5) and enforcing completeness,

$$u^h(\mathbf{x}) = u(\mathbf{x}) = \sum_I \mathbf{b}^T(\mathbf{x})\mathbf{P}(\mathbf{x}_I - \mathbf{x})\phi_a(\mathbf{x}_I - \mathbf{x})u(\mathbf{x}_I)\Delta V_I \tag{8}$$

when $u(\mathbf{x})$ is a polynomial of order $\leq N$. In this case, $u(\mathbf{x}_I)$ is exactly represented by its Nth order Taylor expansion about $\mathbf{x}$.

$$u(\mathbf{x}_I) = u(\mathbf{x}) + (x_{I1} - x_1)\frac{\partial u(\mathbf{x})}{\partial x_1} + (x_{I2} - x_2)\frac{\partial u(\mathbf{x})}{\partial x_2} + (x_{I3} - x_3)\frac{\partial u(\mathbf{x})}{\partial x_3} + \cdots \tag{9a}$$

$$= \mathbf{P}^T(\mathbf{x}_I - \mathbf{x})\mathbf{w}(\mathbf{x}), \tag{9b}$$

where

$$\mathbf{w}(\mathbf{x}) \equiv \left[u(\mathbf{x}), \frac{\partial u(\mathbf{x})}{\partial x_1}, \frac{\partial u(\mathbf{x})}{\partial x_2}, \frac{\partial u(\mathbf{x})}{\partial x_3}, \cdots\right]^T. \tag{10}$$

Substituting (9b) into (8) and solving for $\mathbf{b}(\mathbf{x})$ gives

$$\mathbf{b}(\mathbf{x}) = \mathbf{M}^{-1}(x)\mathbf{P}(0), \tag{11}$$

where

$$\mathbf{M}(\mathbf{x}) \equiv \sum_I \mathbf{P}(\mathbf{x}_I - \mathbf{x})\mathbf{P}^T(\mathbf{x}_I - \mathbf{x})\phi_a(\mathbf{x}_I - \mathbf{x})\Delta V_I. \tag{12}$$

Using (6) and (11) in (5) allows the reproducing equation to be rewritten as

$$u^h(\mathbf{x}) = \sum_I \Phi_I(\mathbf{x})u_I, \tag{13}$$

where $u_I = u(\mathbf{x}_I)$, and the shape function $\Phi_I(\mathbf{x})$, which is independent of $u(\mathbf{x})$, is

$$\Phi_I(\mathbf{x}) = \mathbf{P}^T(0)\mathbf{M}^{-1}(\mathbf{x})\mathbf{P}^T(\mathbf{x}_I - \mathbf{x})\phi_a(\mathbf{x}_I - \mathbf{x})\Delta V_I. \tag{14}$$

In the next section we show how this approximation of a function through its values at a discrete set of nodes (13) can be used in a Galerkin solution of the Navier–Stokes equations.

3. IMPLICIT FORMULATION

3.1. The Governing Equations

We consider here incompressible, Newtonian fluids with constant density ρ and dynamic viscosity μ, flowing in a domain Ω with boundary Γ. The Navier–Stokes equations are written in index form in terms of velocity u_i and pressure p as follows.

1. Conservation of mass is

$$u_{i,i} = 0 \quad \text{in } \Omega. \tag{15}$$

2. Conservation of momentum is

$$\rho u_{i,t} + \rho u_j u_{i,j} = \sigma_{ij,j} + \rho b_i, \tag{16}$$

where b_i is the ith body force component. The Cauchy stress tensor σ_{ij} is defined as

$$\sigma_{ij} = -p\delta_{ij} + 2\mu e_{ij}, \tag{17a}$$

$$e_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i}) \quad \text{in } \Omega. \tag{17b}$$

3. Boundary conditions, in addition to the equations of motion, are stipulated as

$$u_i = g_i, \quad x \in \Gamma_{g_i}, \tag{18a}$$

$$\sigma_{ij} n_j = h_i, \quad x \in \Gamma_{h_i}, \tag{18b}$$

where g_i and h_i are given functions. The fluid boundary Γ is partitioned into Γ_{g_i} and Γ_{h_i} such that for each degree of freedom i, $\Gamma = \Gamma_{g_i} \cup \Gamma_{h_i}$ while $\Gamma_{g_i} \cap \Gamma_{h_i} = 0$ (i.e., the Dirichlet and Neumann condition boundaries span the entire fluid surface but do not intersect).

The transformation of the strong form of the conservation equations (15) and (16) into weak forms suitable for solution with RKPM requires two test functions: the velocity test function δu_i and the pressure test function δp. The velocity test function δu_i satisfies the homogeneous boundary condition:

$$\delta u_i = 0, \quad x \in \Gamma_{g_i}. \tag{19}$$

The standard Galerkin method for advection-dominated flows of incompressible fluids leads to undesirable oscillations in the velocity and pressure solutions. To reduce or eliminate these oscillations, the test functions are augmented with stabilization terms [4, 13],

$$\delta \tilde{v}_i = \delta u_i + \tau^m u_k \delta v_{i,k} + \tau^c \delta p_{,i}, \tag{20a}$$

$$\delta \tilde{p} = \delta p + \tau^c \delta u_{i,i}, \tag{20b}$$

where τ^m and τ^c are scalar stabilization parameters that are in general functions of the computational grid, the time-step size, and the flow variables [14].

The weak form of the continuity equation (15) is obtained by multiplying by the pressure test function $\delta \tilde{p}$ and integrating over Ω:

$$\int_\Omega (\delta p + \tau^c \delta u_{j,j}) u_{i,i} \, d\Omega = 0. \tag{21}$$

Similarly, the weak form of the momentum equation (16) is derived by multiplying by the velocity test function $\delta\tilde{v}_i$:

$$\int_\Omega (\delta u_i + \tau^m u_k \delta u_{i,k} + \tau^c \delta p_{,i})(\rho u_{i,t} + \rho u_j u_{i,j} - \sigma_{ij,j} - \rho b_i)\, d\Omega = 0. \tag{22}$$

By rearranging (22) and using integration by parts, the final weak form of the momentum equation becomes

$$\begin{aligned}
&\int_\Omega \rho(\delta u_i + \tau^m u_k \delta u_{i,k} + \tau^c \delta p_{,i})(\dot{u}_i + u_j u_{i,j})\, d\Omega - \int_{\Gamma_{h_i}} \delta v_i h_i \, d\Gamma_{h_i} \\
&\quad - \int_{\Gamma_g} \delta v_i \sigma_{ij} n_j \, d\Gamma_g + \int_\Omega \delta u_{i,j} \sigma_{ij} \, d\Omega + \int_\Omega (\tau^m u_k \delta v_{i,k} + \tau^c \delta p_{,i}) p_{,i} \\
&\quad - \int_\Omega (\tau^m u_k \delta v_{i,k} + \tau^c \delta p_{,i}) \mu (u_{i,jj} + u_{j,ij})\, d\Omega = 0.
\end{aligned} \tag{23}$$

Note that the last integral on the left-hand-side requires the calculation of the second derivative of the velocity, $u_{i,jj}$. For linear finite elements, the term is either equal to zero or approximately zero on element interiors, depending on the grid regularity; $u_{i,jj}$ is a delta function on element boundaries. For RKPM using cubic splines, $u_{i,jj}$ is a smooth function, and so this final term can in theory be computed. In practice, however, we neglect this term due to the cost of the computations and memory space required to calculate and store the second derivatives of the RKPM shape functions [21]. The neglect of these second-derivative stabilization terms is common practice in finite-element computations and has only a small effect on accuracy for most problems (see, for example, [19]). One reason for this is that the stabilization parameters τ^m and τ^c are designed to go to zero with decreasing nodal spacing; in many regions of the flow where second derivatives are large, such as the boundary layer, the nodal spacing and therefore the stabilization term are small.

The final weak form that is solved is the combination of the momentum equation (21) and the continuity equation (23):

$$\begin{aligned}
&\int_\Omega \rho(\delta u_i + \tau^m u_k \delta u_{i,k} + \tau^c \delta p_{,i})(\dot{u}_i + u_j u_{i,j})\, d\Omega - \int_{\Gamma_{h_i}} \delta v_i h_i \, d\Gamma_{h_i} \\
&\quad - \int_{\Gamma_g} \delta v_i \sigma_{ij} n_j \, d\Gamma_g + \int_\Omega \delta u_{i,j} \sigma_{ij} \, d\Omega + \int_\Omega (\tau^m u_k \delta v_{i,k} + \tau^c \delta p_{,i}) p_{,i} \, d\Omega \\
&\quad + \int_\Omega (\delta p + \tau^c \delta u_{j,j}) u_{i,i} \, d\Omega = 0.
\end{aligned} \tag{24}$$

3.2. Essential Boundary Conditions

In this paper, essential boundary conditions are applied using the "bridging scale hierarchical enrichment" method [17, 20]. As mentioned in the introduction, the Kronecker delta property is not satisfied for meshfree shape functions as it is for finite-element shape functions [Eq. (1)]. Because of this inequality, a function that is reproduced from a vector of nodal degrees of freedom via $u^h(x) = \sum_I \Phi_I(x) u_I$ does not take on the values of the nodal degrees of freedom when evaluated at the nodes, i.e., $u^h(x_I) \neq u_I$. Therefore, essential boundary conditions cannot be applied simply by enforcing individual values of the nodal degrees of freedom.

In order to efficiently enforce essential boundary conditions despite this inconvenience, we begin by decomposing the total solution u^h into two functions:

$$u^h(x) = g^h_{bound}(x) + v^h(x). \tag{25}$$

Here $g^h_{bound}(x)$ is a known function that is based in some way (to be defined) on the prescribed essential boundary condition $u(x) = g(x)$ on Γ_g, while $v^h(x)$ should depend on the nodal degrees of freedom u_I but should go to zero at nodes on the essential boundary.

The simplest way to construct $g^h_{bound}(x)$ is to interpolate using a set of shape functions that satisfy the Kronecker delta property on the essential boundary, e.g., the linear finite-element shape functions. Because only the shape functions associated with the essential boundary nodes are needed, we define B as the set of nodes on Γ_g and write

$$g^h_{bound}(x) = \sum_{I \in B} N_I(x) g_I, \tag{26}$$

where $N_I(x)$ is the finite-element shape function at node I and $g_I = g(x_I)$.

In order to construct $v^h(x)$, we first note that Eq. (26) is a projection of $g(x)$ onto the set of finite-element shape functions at the boundary. We therefore subtract from the RKPM approximation given in Eq. (13) its own projection onto the same set of shape functions, giving as desired a $v^h(x)$ that is zero on the boundary nodes. Temporarily defining $w^h(x) \equiv \sum_{I \in A} \Phi_I(x) u(x_I)$, where A is the set of all nodes (including those on the boundary),

$$\begin{aligned} v^h(x) &= w^h(x) - \sum_{I \in B} N_I(x) w^h(x_I) \\ &= \sum_{I \in A} \Phi_I(x) u_I - \sum_{I \in B} N_I(x) \left[\sum_{J \in A} \Phi_J(x_I) u_J \right] \\ &= \sum_{I \in A} \left[\Phi_I(x) - \sum_{J \in B} N_J(x) \Phi_I(x_J) \right] u_I. \end{aligned} \tag{27}$$

The projection of the RKPM approximation on the set of boundary finite-element shape functions is the "bridging scale" referred to in the name of the method [22].

Substituting (26) and (27) into (25),

$$u^h(x) = \sum_{I \in B} N_I(x) g_I + \sum_{I \in A} \tilde{\Phi}_I(x) u_I, \tag{28}$$

where

$$\tilde{\Phi}_I(x) \equiv \Phi_I(x) - \sum_{J \in B} N_J(x) \Phi_I(x_J). \tag{29}$$

It can be easily shown that for a node K on the essential boundary, $u^h(x_K) = g_K$. Thus the essential boundary conditions can be applied directly through the coefficients g_I. The summation in the definition of the modified shape functions (29) is nonzero only within

elements that contain one or more nodes lying on the essential boundary. Therefore, throughout most of the domain the usual meshfree shape functions can be used unchanged. In addition, there is no need to create an entire finite-element mesh, as only one layer of elements on the essential boundary is necessary.

An example of both unmodified and modified 1-D RKPM shape functions whose supports span seven nodes is shown in Fig. 1, along with the field emission microscopy (FEM)

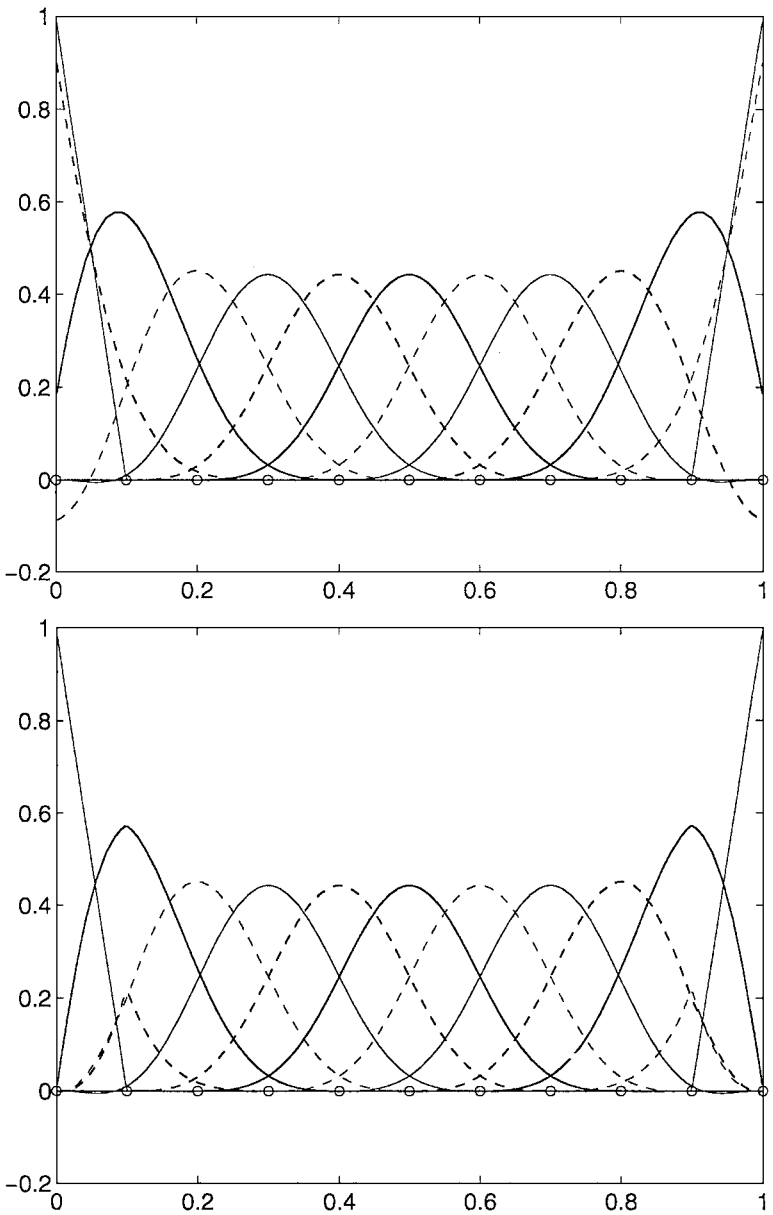


FIG. 1. Unmodified (above) and modified (below) 1-D RKPM shape functions and FEM boundary shape functions. FEM shape functions are the piecewise linear functions at either end. Different line styles are included only to distinguish neighboring shape functions. Note that only the finite-element shape functions are nonzero at the boundary in the modified plot.

boundary shape functions. Note that after modification, only the FEM functions are nonzero at the boundary nodes.

Remarks.

- This method of applying boundary conditions preserves the Nth-order accuracy of the RKPM shape functions [20].
- When used in a Galerkin solution, the order of convergence of the RKPM method is preserved as long as it is noted that boundary integrals such as $\int_{\Gamma_g} \delta u_i \sigma_{ij} n_j \, d\Gamma_g$ in the weak form (24) are nonzero, since the boundary conditions are enforced only on the boundary nodes and not on the entire boundary [12, 20].
- There is no matrix inversion or transformation necessary for the application of boundary conditions, as is needed for many methods [10, 20]. For this reason, this method provides better performance in parallel algorithms than do other methods, such as the corrected collocation method [20], in which the most memory-efficient means of solving (e.g., matrix LU decomposition and back-substitution) result in inherently serial procedures.
- The function $g^h_{bound}(x)$ is a known function, independent of the nodal degrees of freedom u_I, and will therefore contribute to the right-hand-side in a Galerkin solution. Furthermore, variations of $u^h(x)$ to be used as test functions reduce to

$$\delta u^h(x) = \delta u^h(x) = \sum_{I \in A} \tilde{\Phi}_I(x) \delta u_I. \tag{30}$$

3.3. Discretization of the Weak Form

The discretizations of the conservation equations are written using the modified shape functions $\tilde{\Phi}(\mathbf{x})$ to enforce essential boundary conditions. The velocity and pressure functions, $u_i(\mathbf{x})$ and $p(\mathbf{x})$, along with the test functions $\delta u_i(\mathbf{x})$ and $\delta p(\mathbf{x})$, are interpolated as

$$u^h_i(\mathbf{x}) = \sum_{I \in B_{u_i}} N_I(\mathbf{x}) g_{iI} + \sum_{I \in A} \tilde{\Phi}_I(\mathbf{x}) u_{iI}, \tag{31a}$$

$$\delta u^h_i(\mathbf{x}) = \sum_{I \in A} \tilde{\Phi}_I(\mathbf{x}) \delta u_{iI}, \tag{31b}$$

$$p^h(\mathbf{x}) = \sum_{I \in B_p} N_I(\mathbf{x}) s_I + \sum_{I \in A} \tilde{\Phi}_I(\mathbf{x}) p_I, \tag{31c}$$

$$\delta p^h(\mathbf{x}) = \sum_{I \in A} \tilde{\Phi}_I(\mathbf{x}) \delta p_I. \tag{31d}$$

Subscripts on the boundary node sets B are included as a reminder that the essential boundary conditions may be applied at different nodes for different variables. The coefficients s_I are included in Eq. (31c) for those cases in which an essential boundary condition for the pressure is enforced. For many simulations there is no such condition on the pressure; in these cases, the set B_p is empty, and $\tilde{\Phi}(\mathbf{x}) = \Phi(\mathbf{x})$ for the pressure interpolation.

Substituting Eq. (31b) and Eq. (31d) into Eq. (24) gives

$$\begin{aligned}
&\sum_{I\in A}\int_\Omega \rho\left(\delta u_{iI}\tilde{\Phi}_I + \tau^m u_k^h \delta u_{iI}\tilde{\Phi}_{I,k} + \tau^c \delta p_I \tilde{\Phi}_{I,i}\right)\left(\dot{u}_i^h + u_j^h u_{i,j}^h\right) d\Omega \\
&\quad + \sum_{I\in A}\int_\Omega \delta u_{iI}\tilde{\Phi}_{I,j}\sigma_{ij}^h \, d\Omega + \sum_{I\in A}\int_\Omega \left(\tau^m u_k^h \delta u_{iI}\tilde{\Phi}_{I,k} + \tau^c \delta p_I \tilde{\Phi}_{I,i}\right) p_{,i}^h \, d\Omega \\
&\quad - \sum_{I\in A}\int_{\Gamma_{h_i}} \delta u_{iI}\tilde{\Phi}_I h_i \, d\Gamma_{h_i} - \sum_{I\in A}\int_{\Gamma_{g_i}} \delta u_{iI}\tilde{\Phi}_I \sigma_{ij} n_j \, d\Gamma_{g_i} \\
&\quad + \sum_{I\in A}\int_\Omega (\delta p_I \tilde{\Phi}_I + \tau^c \delta u_{iI}\tilde{\Phi}_{I,i}) u_{i,i}^h \, d\Omega = 0,
\end{aligned} \tag{32}$$

where u_i^h and p^h are calculated from (31a) and (31c), respectively. We avoid making this substitution explicitly in order to emphasize that no large-system matrices are computed or stored in our formulation.

By the arbitrariness of the test function degrees of freedom δu_{iI} and δp_I, we have four equations at each node I,

$$r_{iI}^u = 0, \tag{33a}$$

$$r_I^p = 0, \tag{33b}$$

where the residual vectors r_{iI}^u and r_I^p are

$$\begin{aligned}
r_{iI}^u &= \int_\Omega \rho\left(\tilde{\Phi}_I + \tau^m u_k^h \tilde{\Phi}_{I,k}\right)\left(\dot{u}_i^h + u_j^h u_{i,j}^h\right) d\Omega - \int_{\Gamma_{h_i}} \tilde{\Phi}_I h_i \, d\Gamma_{h_i} - \int_{\Gamma_{g_i}} \tilde{\Phi}_I \sigma_{ij} n_j \, d\Gamma_{g_i} \\
&\quad + \int_\Omega \tilde{\Phi}_{I,j}\sigma_{ij}^h \, d\Omega + \int_\Omega \tau^m u_k^h \tilde{\Phi}_{I,k} p_{,i}^h \, d\Omega + \int_\Omega \tau^c \tilde{\Phi}_{I,i} u_{j,j}^h \, d\Omega,
\end{aligned} \tag{34a}$$

$$r_I^p = \int_\Omega \rho\tau^c \tilde{\Phi}_{I,i}\left(\dot{u}_i^h + u_j^h u_{i,j}^h\right) d\Omega + \int_\Omega \tau^c \tilde{\Phi}_{I,i} p_{,i}^h \, d\Omega + \int_\Omega \tilde{\Phi}_I u_{i,i}^h \, d\Omega. \tag{34b}$$

The residuals in Eq. (34) are evaluated at each iteration in our solution algorithm (see Section 4.3) by first computing u_i^h and p^h and their derivatives at every integration point according to (31a) and (31c). The integration points can be chosen based either on the elements of a corresponding finite-element grid or in a regular array unrelated to the nodal distribution. In our work, we use the former method; generally, we find that we require one integration point at the center of each tetrahedral element.

3.4. Time Integration Scheme

Velocity and pressure are to be solved at every time step using the residual equations (34) derived in the previous section. Specifically, we calculate Δu_{iI}^m and p_I^{m+1}, where the superscript m denotes the current time step and

$$\Delta u_{iI}^m = u_{iI}^{m+1} - u_{iI}^m. \tag{35}$$

The increment of the velocity at time m, $\Delta\mathbf{u}^m$, is used to calculate the time derivative of the velocity approximation, $\dot{u}_i^h$:

$$\dot{u}_i^h = \frac{u_i^{h,m+1} - u_i^{h,m}}{\Delta t} = \frac{1}{\Delta t}\sum_{I\in A}\tilde{\Phi}_I\,\Delta u_{iI}^m. \tag{36}$$

The value of the velocity approximation u_i^h in Eq. (34) is evaluated as

$$u_i^h = \alpha u_i^{h,m+1} + (1-\alpha)u_i^{h,m} \tag{37}$$

$$= \sum_{I\in B} N_I g_{iI} + \sum_{I\in A}\tilde{\Phi}_I\left(u_{iI}^m + \alpha\Delta u_{iI}^m\right), \tag{38}$$

where $0 \le \alpha \le 1$. In this work we use $\alpha = \frac{1}{2}$ for a central-difference scheme. Note that p is always computed at $m+1$; there is no central difference on p.

Newton's method is used at each time step for this implicit algorithm with initial guesses $\Delta\mathbf{u}^m = 0$ and $\mathbf{p}^{m+1} = \mathbf{p}^m$. The vectors of increments in velocity and pressure, $\Delta\mathbf{u}^{incr}$ and $\mathbf{p}^{incr}$, are solved iteratively to satisfy the nonlinear residual equations (34):

$$\mathbf{r}_i^w(\mathbf{u}^m, \Delta\mathbf{u}^m + \mathbf{u}^{incr}, \mathbf{p}^{m+1} + \mathbf{p}^{incr}) = 0, \tag{39a}$$

$$\mathbf{r}^q(\mathbf{u}^m, \Delta\mathbf{u}^m + \mathbf{u}^{incr}, \mathbf{p}^{m+1} + \mathbf{p}^{incr}) = 0. \tag{39b}$$

Taylor expanding (39) gives

$$\mathbf{r}_i^w(\mathbf{u}^m, \Delta\mathbf{u}^m, \mathbf{p}^{m+1}) + \mathbf{r}_{i,\mathbf{u}}^w(\mathbf{u}^m, \Delta\mathbf{u}^m, \mathbf{p}^{m+1})\Delta\mathbf{u}^{incr} + \mathbf{r}_{i,p}^w(\mathbf{u}^m, \Delta\mathbf{u}^m, \mathbf{p}^{m+1})\mathbf{p}^{incr} \approx 0, \tag{40a}$$

$$\mathbf{r}^q(\mathbf{u}^m, \Delta\mathbf{u}^m, \mathbf{p}^{m+1}) + \mathbf{r}_{,\mathbf{u}}^q(\mathbf{u}^m, \Delta\mathbf{u}^m, \mathbf{p}^{m+1})\Delta\mathbf{u}^{incr} - \mathbf{r}_{,p}^q(\mathbf{u}^m, \Delta\mathbf{u}^m, \mathbf{p}^{m+1})\mathbf{p}^{incr} \approx 0. \tag{40b}$$

The resulting matrix equation, linear in $\Delta\mathbf{u}^{incr}$ and $\mathbf{p}^{incr}$, is

$$\begin{bmatrix} \mathbf{r}_{i,\mathbf{u}}^w & \mathbf{r}_{i,p}^w \\ \mathbf{r}_{,\mathbf{u}}^q & \mathbf{r}_{,p}^q \end{bmatrix} \begin{Bmatrix} \Delta\mathbf{u}^{incr} \\ \mathbf{p}^{incr} \end{Bmatrix} = \begin{Bmatrix} -\mathbf{r}_i^w \\ -\mathbf{r}^q \end{Bmatrix}. \tag{41}$$

The unknowns $\Delta\mathbf{u}^m$ and $\mathbf{p}^m$ can be updated as

$$\Delta\mathbf{u}^m \to \Delta\mathbf{u}^m + \Delta\mathbf{u}^{incr}, \tag{42a}$$

$$\mathbf{p}^{m+1} \to \mathbf{p}^{m+1} + \mathbf{p}^{incr}. \tag{42b}$$

This Newton step is repeated for a fixed number of iterations or until the norms of the residual vector, $\mathbf{r}_i^w$ and $\mathbf{r}^q$, are below a given tolerance before proceeding to the next time step.

We solve the linear system of Eq. (41) using the generalized minimum residual (GMRES) algorithm with diagonal preconditioning. The minimization property of this method ensures that even an incomplete GMRES procedure decreases the residual. In combination with Newton's method, a very small Krylov subspace is enough to obtain fast

convergence to the solution of a nonlinear problem. In the solution of matrix equation $\mathbf{Ax} = \mathbf{b}$, the GMRES algorithm requires the repeated multiplication of a given vector by matrix $\mathbf{A}$. In our implementation, we compute each of these matrix–vector products without evaluating the individual elements of the matrix. Instead, the analytical expression for the residual derivatives is used to directly compute the product of the matrix on the left-hand-side of Eq. (41) with a given vector. This strategy greatly reduces the amount of memory necessary for storage, at the cost of a slight increase in the total number of computations.

4. PARALLEL COMPUTATION ANALYSIS

The CFD code is parallelized based on a domain-decomposition message-passing paradigm to be executed on a CRAY T3E using the message passing interface (MPI) library. Currently, we are using the 256-processor CRAY T3E-1200 at the Army HPC Research Center as our computing platform. Each processor has 512 Mbytes of dedicated memory.

4.1. Communication Structure

Because the physical domain is partitioned between processors, communication is required. There are two sets of geometrical entities that must be distributed. First, the nodes themselves are assigned to different processors. Each processor stores the "official" data for its own nodes, and updates at each time step. In addition, numerical computation of the integrals in the conservation equations must be done by evaluating at a discrete set of integration points. To compute efficiently, each processor should therefore also receive a subset of the integration points and be responsible for calculating the contribution to the residual vectors at those points (see Fig. 2). In order to evaluate a function at a given integration point, a processor must have information about all nodes under whose domain of influence that integration point falls. The integration point in turn contributes to the residual computed at each of those same influence nodes. These nodes are not necessarily among the ones "owned" by that processor, but are distributed among different processors, necessitating communication at each iteration. For every evaluation of the residual vectors, each processor seeks from the other processors the data at the nodes that are needed by its own integration points. This is known as a *gather* operation. Once this data has been used to compute the residual vectors, each processor has a piece of the residual vector at all nodes whose domains include any one of that processor's integration points. The goal, however, is for each processor to have the entire residual at each of its own nodes. So the reverse of the previous communication step must take place; each processor sends the residual it has computed at a given node to the processor that is responsible for that node, where the total residual is stored. This is called a *scatter* operation. Both gather and scatter operations, which are sometimes are referred to as a swap-and-add procedure, must be performed at every iteration. This process is shown in Fig. 3.

4.2. Partitioning Schemes

In the code, the nodes and integration points are assigned to the processors based on a domain-decomposition algorithm, in which each processor receives and is responsible

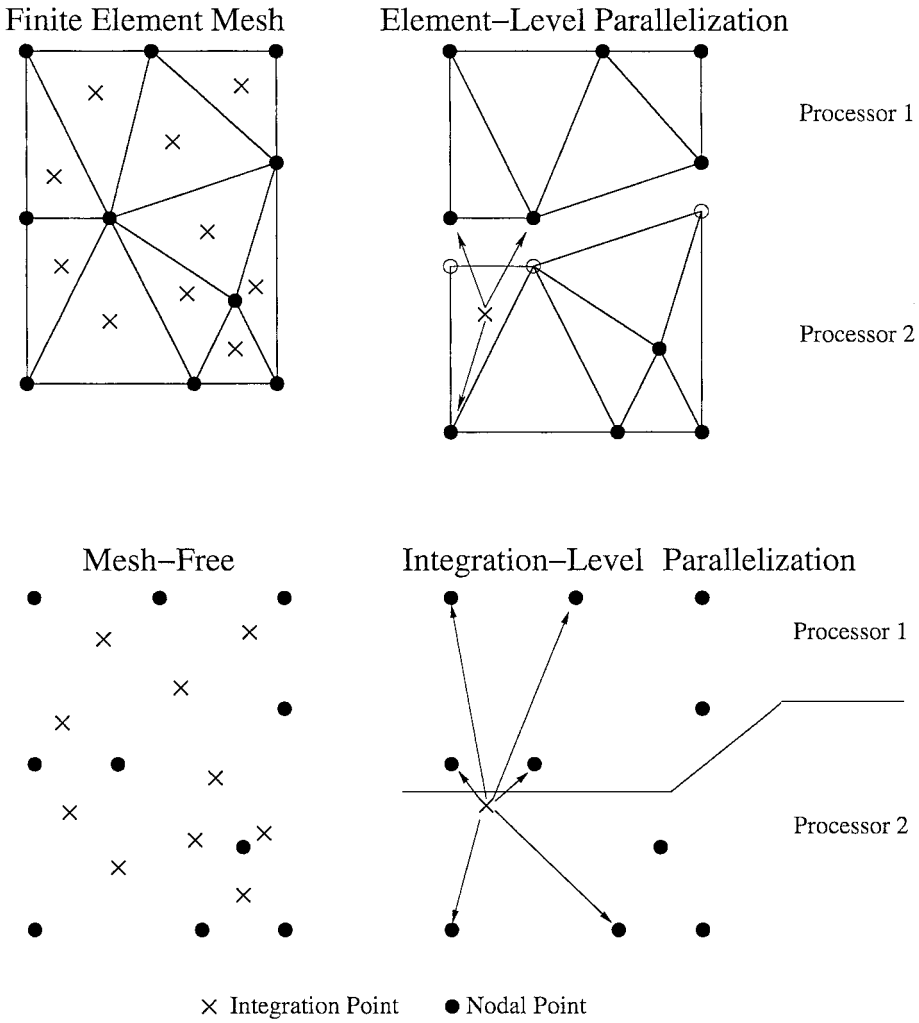


FIG. 2. Communication for finite-element and meshfree methods.

for computing on a partition of the domain Ω. Due to the large domain of influence of each node, meshfree methods require more communication time than similar-sized parallel finite-element computations. In order to minimize the required amount of communication, the domain is partitioned before the analysis in such a way that each processor "owns" a large number of integration points and nodes that neighbor each other. In our work thus far, the nodes and integration points are sorted according to their x-coordinates in ascending order. Using this algorithm, the nodes are numbered from left to right in the geometry domain that is to be solved, as in Fig. 4. This saves communication time between processors because in most cases a node does not need to communicate with neighbors that may belong to another processor; most often, neighbors belong to the same processor after partitioning.

This partitioning method is used in our code for simplicity. There are a variety of different partitioning codes or softwares, such as Metis, that can be easily used as a "black box" [8].

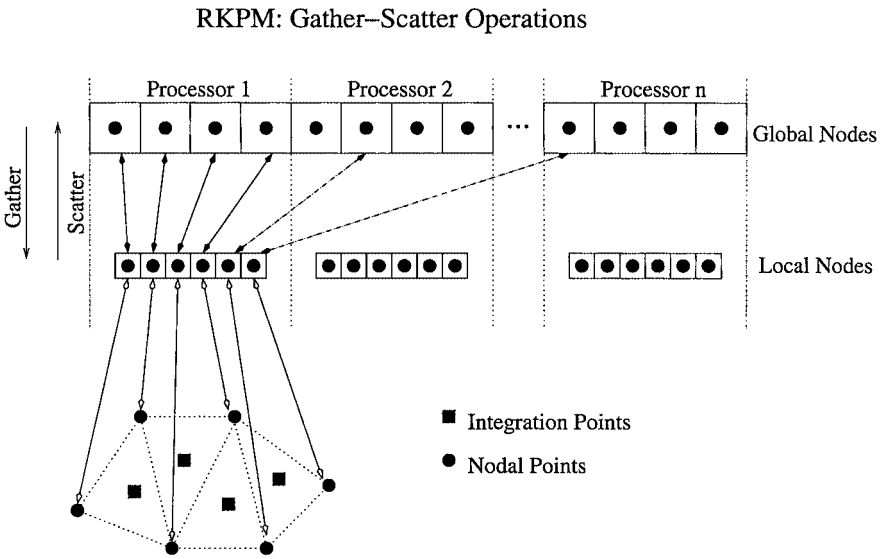


FIG. 3. Gather and scatter operation for the meshfree method.

4.3. Outline of Procedures

This section includes the algorithm for implementing RKPM implicit analysis.

1. Serial preprocessing
   i. Read geometry data
   ii. Partition the nodes and integration points onto each processor
2. Parallel analysis on each processor
   i. Read the input partitioned data
   ii. Set up data structure for local analysis
   iii. Calculate shape functions $\Phi_I(\mathbf{x})$ and their derivatives, Eq. (14)
   iv. Calculate the modified shape functions $\tilde{\Phi}(\mathbf{x})$ and their derivatives, Eq. (29)
3. Time increment loop
   i. Initialize variables: $\Delta\mathbf{d}_i^m = 0,\ \mathbf{p}^{m+1} = \mathbf{p}^m$
   ii. Newton iteration loop
      a. Calculate the residuals, Eqs. (34)
      b. Check residual, if converges, then go to step iii
      c. Solve for $\Delta\mathbf{d}^{incr}$ and $\mathbf{p}^{incr}$ using GMRES, Eq. (41)
      d. Update the variables, $\mathbf{d}^m$ and $\mathbf{p}^{m+1}$, Eqs. (42)
   iii. Output the results to files
4. Serial postprocessing
   i. Create a single output file for processing

5. NUMERICAL EXAMPLES

5.1. Simple 3-D Flow Past a Circular Cylinder

To validate the code, a uniform flow past a 3-D cylinder is simulated. This flow has been the subject of many theoretical, experimental, and computational investigations; two notable

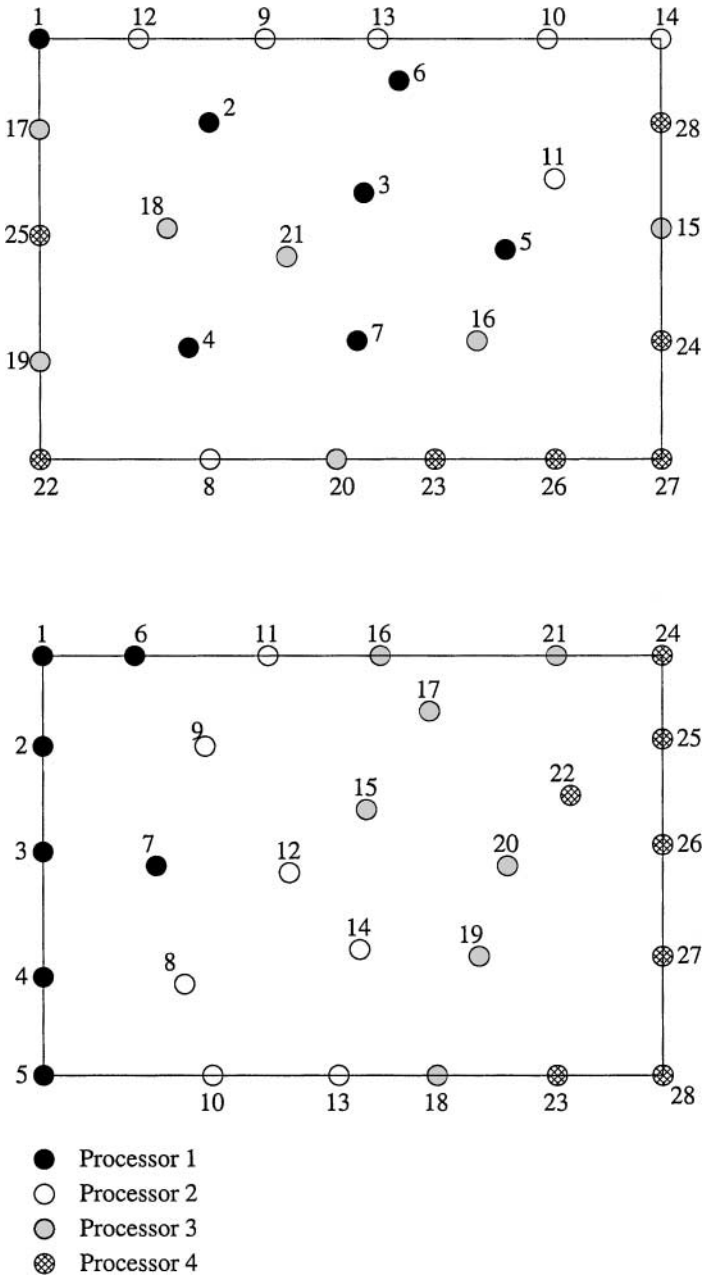


FIG. 4. Before (above) and after (below) nodal partitioning.

examples to which we compare our results are those of Collins and Dennis [6, 7] and Bar-Lev and Yang [2], who studied analytically the early time history of flow past a cylinder initially at rest. Our simulations employ parallel RKPM with the enrichment boundary condition implementation outlined in Section 3.2. For comparision, we have solved the same problem using both finite elements and RKPM with the "corrected collocation" method for boundary conditions proposed in [20].

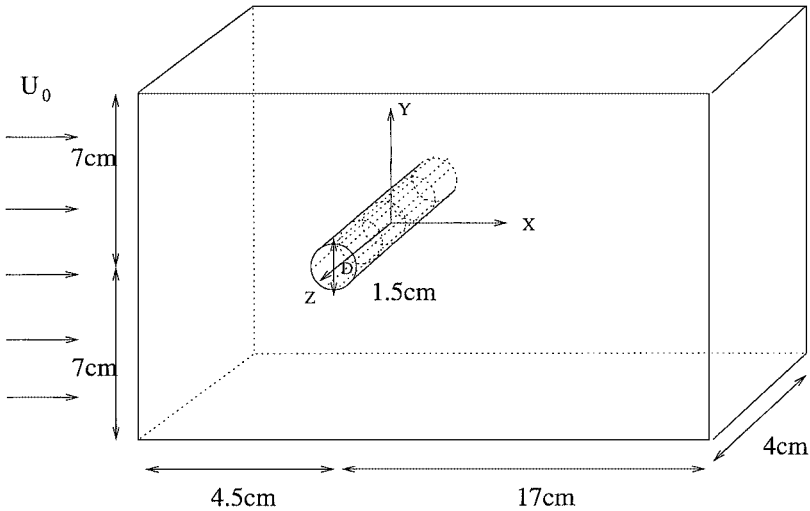


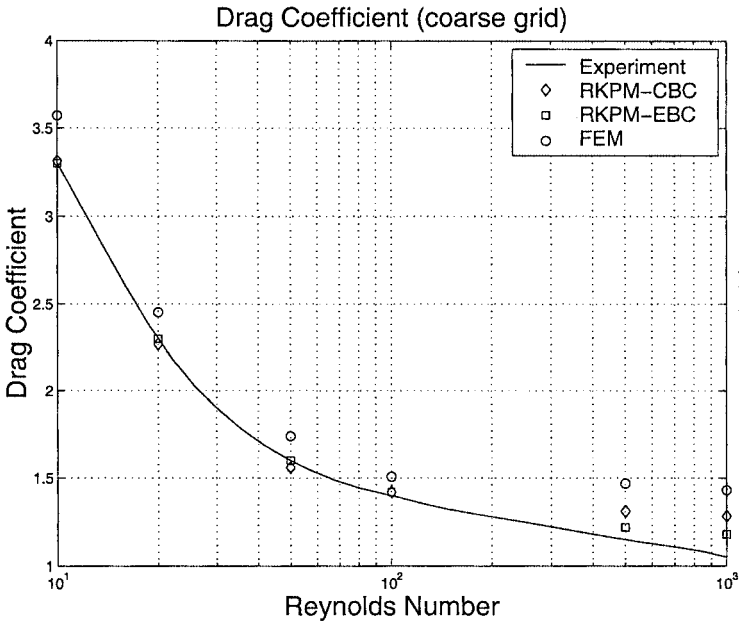
FIG. 5. Flow past a cylinder.

A cylinder with a diameter of 1.5 cm with its axis in the z-direction is placed in a uniform x-directional flow. The dimensions of the computational domain are $21.5 \times 14 \times 4$ cm, and the cylinder is located 4.5 cm downstream of the inflow (see Fig. 5). Two different discretizations are used: a coarse discretization with 2236 nodes and 11,628 integration points, and a fine discretization with 15,447 nodes and 87,646 integration points. For both nodal distributions, the discretization is finest near the cylinder surface in order to resolve the boundary layer.

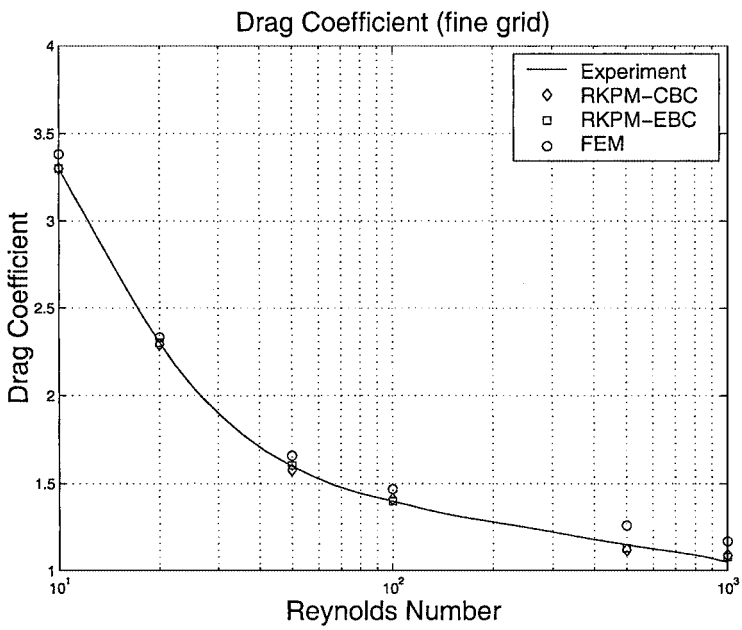
Initially, the velocity is uniform, with speed U_0 everywhere. For $t > 0$, $\mathbf{u} = 0$ is enforced on the surface of the cylinder. The inflow boundary condition at $x = -4.5$ cm is uniform flow of speed U_0, while the outflow at $x = 17$ cm is a zero-stress boundary. The top, bottom, and sides of the computational domain have zero penetration conditions but allow slip parallel to the walls.

Figure 6 shows computed drag coefficients for both nodal distributions plotted against Reynolds number ranging from 10 to 1000 compared with the values obtained from experiments (data from many sources collected by [23], p. 266). As shown on the plot, using the enrichment method discussed in Section 4.3 to implement the essential boundary condition yields results closer to the experimental values than does the method without enrichment, but both methods are more accurate than FEM. The values of drag coefficient for different Reynolds numbers are tabulated in Table I. Our results seem to indicate that even the coarse discretization is sufficiently fine to obtain accurate results using RKPM, while the FEM solution is not fully resolved even with the fine mesh.

The calculated drag coefficients for the early time history are plotted for Reynolds numbers of 40, 200, and 1000 in Fig. 7. The time $T = U_0 t/a$ is nondimensionalized based on the uniform velocity U_0 and the radius of the cylinder, a. The computational results using RKPM and FEM are compared with the analytical results of Bar-Lev and Yang [2] and Collins and Dennis [7]. It can be seen that RKPM more closely matches the theory than does FEM, most likely due to the meshfree shape function's ability to better represent the sharp boundary layer.



(a) Coarse Grid (2236 nodes, 11628 integration points)



(b) Fine Grid (15447 nodes, 87646 qudrature points)

FIG. 6. Drag coefficient of the cylinder vs Reynolds number. CBC, collocation boundary condition implementation; EBC, enrichment boundary condition.

TABLE I
Drag Coefficients for Different Reynolds Numbers

	Re = 10	Re = 20	Re = 50	Re = 100	Re = 500	Re = 1000
Experimental values	3.3	2.3	1.6	1.4	1.15	1.05
2256 nodes						
FEM	8.2%	6.6%	8.7%	7.7%	28%	36%
RKPM-CBC	0.38%	1.4%	2.5%	1.5%	14%	22%
RKPM-EBC	0.0%	0.0%	0.31%	1.1%	8.1%	12%
15,447 nodes						
FEM	2.5%	1.4%	3.7%	4.9%	14%	20%
RKPM-CBC	0.041%	0.45%	1.3%	0.79%	2.6%	3.8%
RKPM-EBC	0.0%	0.0%	0.25%	0.36%	1.7%	2.9%

Note. CBC, collocation boundary condition implementation; EBC, enrichment boundary condition.

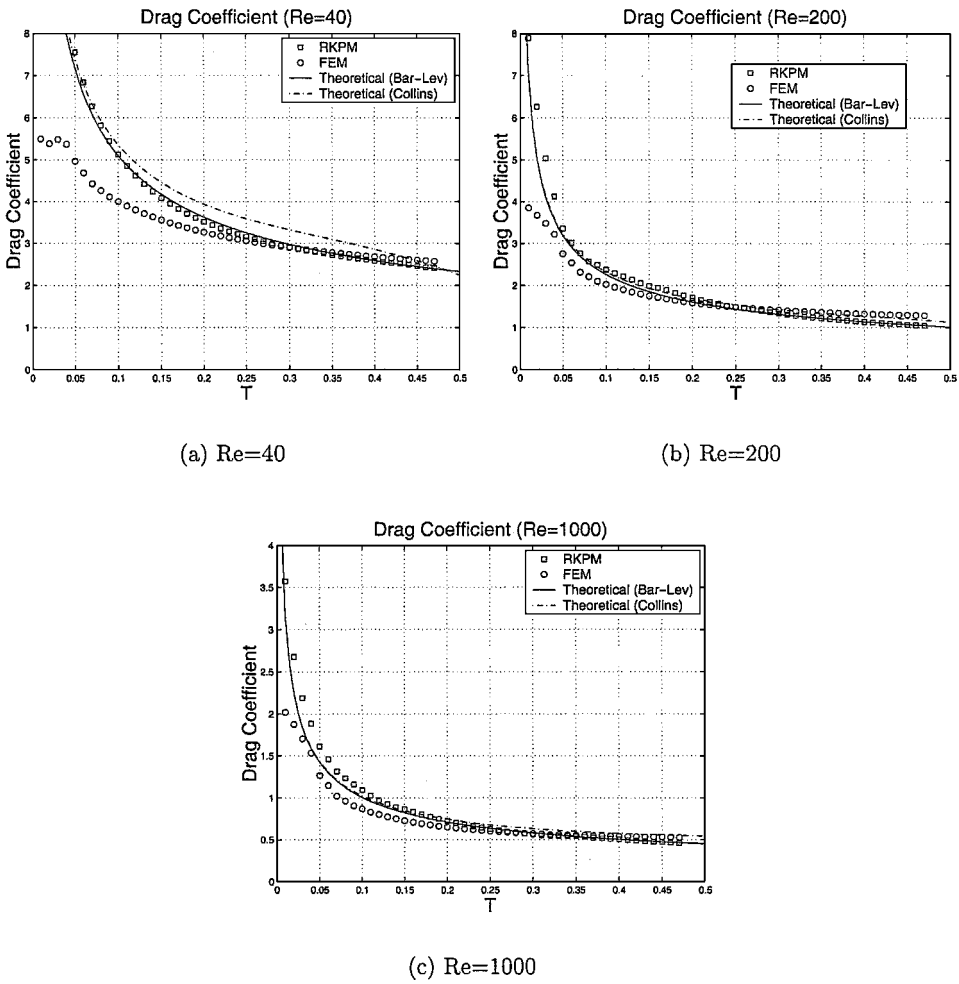


FIG. 7. Early time history of the drag coefficient for different Reynolds numbers (see Bar-Lev [2], Collins and Dennis [7]). $T = U_0 t/a$.

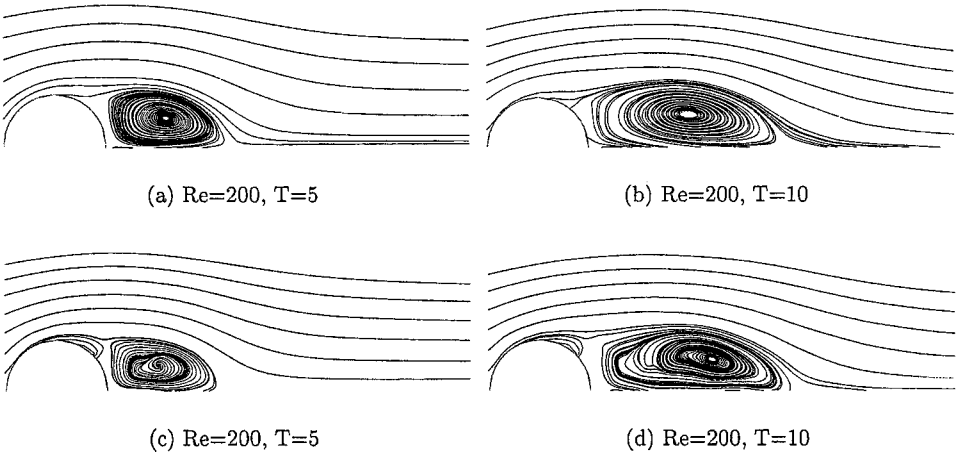


FIG. 8. Streamlines for Re $= 200$ at $T = 5$ and $T = 10$ using FEM (a, b) and the meshfree method (c, d).

The 3-D streamline contours of the flow field for Reynolds numbers of 200 and 1000 at different times are presented in Figs. 8 and 9 for both FEM and RKPM solutions. The vortices and their developments through the early times can be clearly seen in the figures. This shows that the method that has been used in this paper has the ability to capture the characteristics of the flow.

The parallel performance on the Cray T3E for this problem is shown in Fig. 10. The speedup is scaled based on the calculation time of four processors. Although the parallel performance cannot reach the ideal case of linear speedup, speedup is noticeable when running the fixed-size problem using up to 32 processors. Communication then dominates the running time as the number of processors is increased to 64. As mentioned in Section 3.2, using the enrichment scheme to implement the essential boundary conditions produces better performance in parallel by eliminating the LU decomposition procedure or inversion of the matrices necessary for most boundary condition implementations. In the case of RKPM

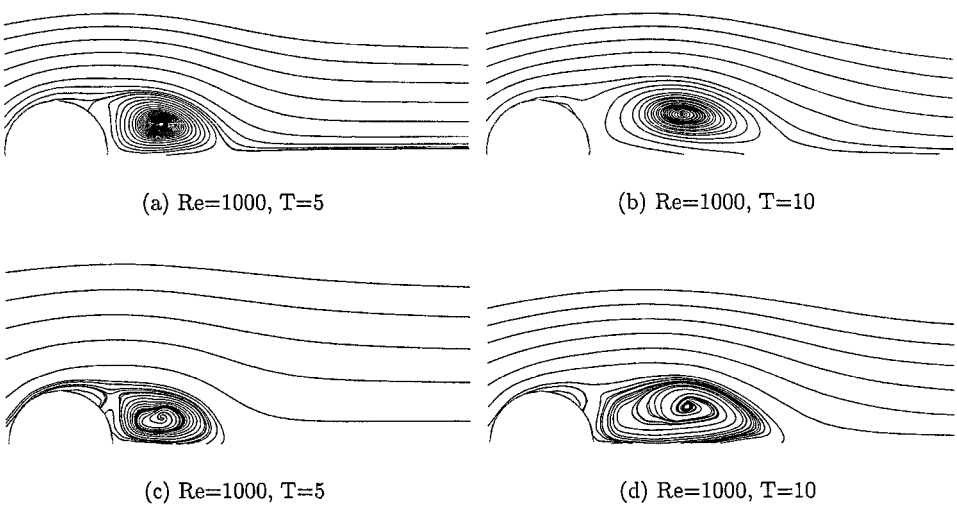


FIG. 9. Streamlines for Re $= 1000$ at $T = 5$ and $T = 10$ using FEM (a, b) and the meshfree method (c, d).

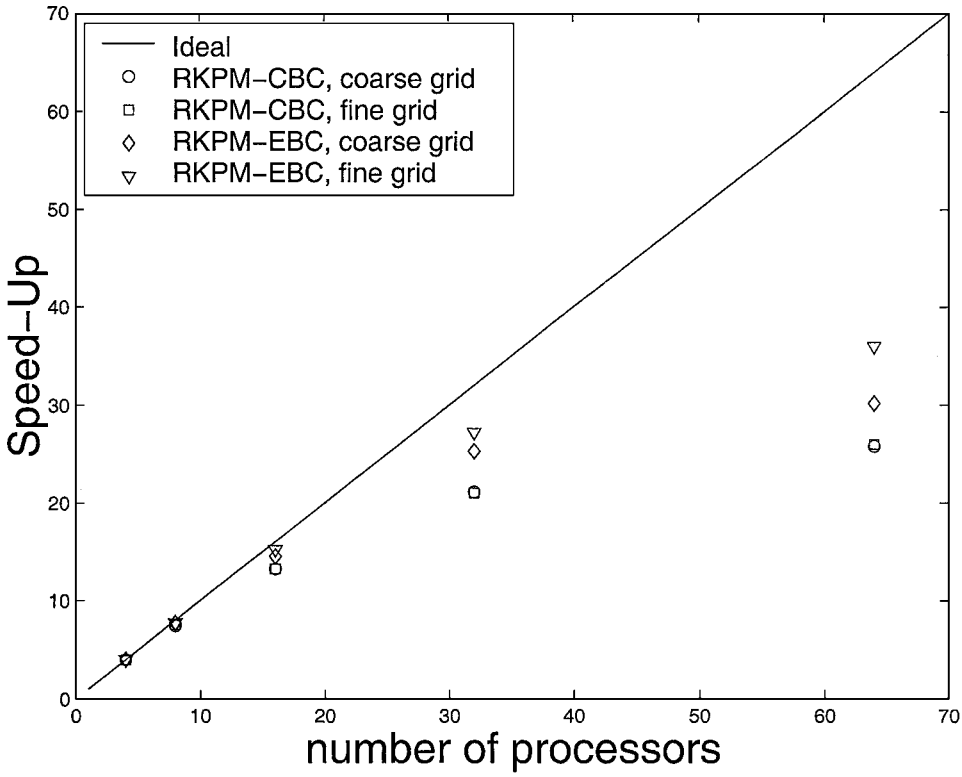


FIG. 10. Parallel performance on a CrayT3E of RKPM for the problem of flow past a cylinder (speedup is calculated based on the calculation time of using four processors). CBC, collocation boundary condition implementation; EBC, enrichment boundary condition implementation.

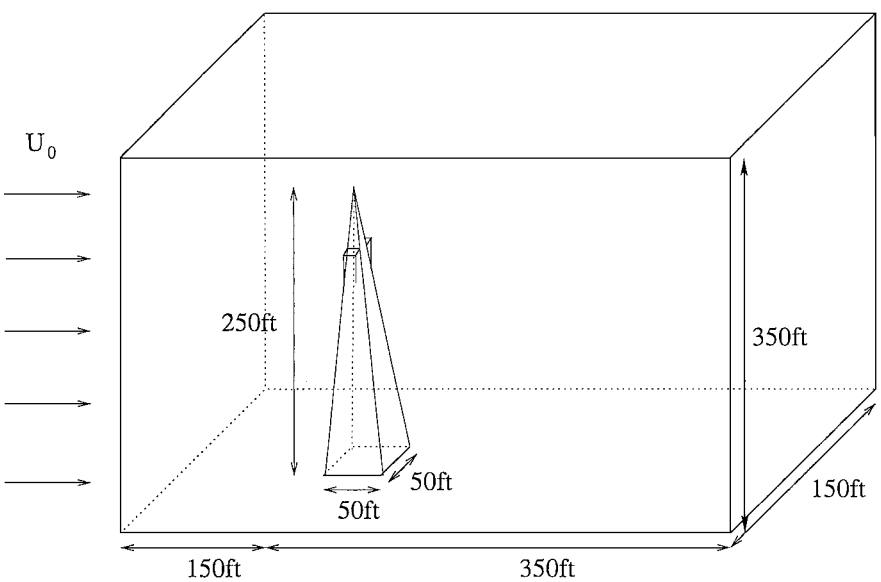


FIG. 11. Flow past a building.

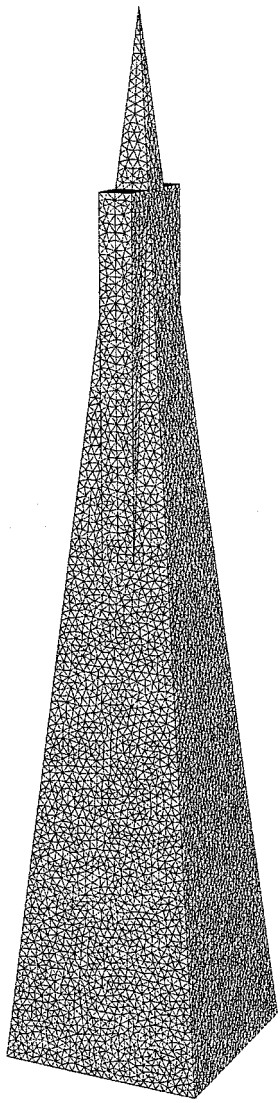


FIG. 12. The finite-element mesh of the building surface; RKPM nodes are placed at the FEM node locations.

using collocation boundary condition implementation, there is no change in speedup from the coarse to the fine mesh due to the limited ability of parallelization. However, the parallel algorithm continues to show improved performance for large-size problems, as also shown in Fig. 10. The results indicate that the enrichment method is more parallelizable than the collocation method.

5.2. 3-D Flow Past a Building

The RKPM flow solution method is applied to the uniform flow past a building to show its parallel application to even larger size problems. The shape of the building is shown in Fig. 11. The structure is 250 ft in height and has a square base with sides of 50 ft.

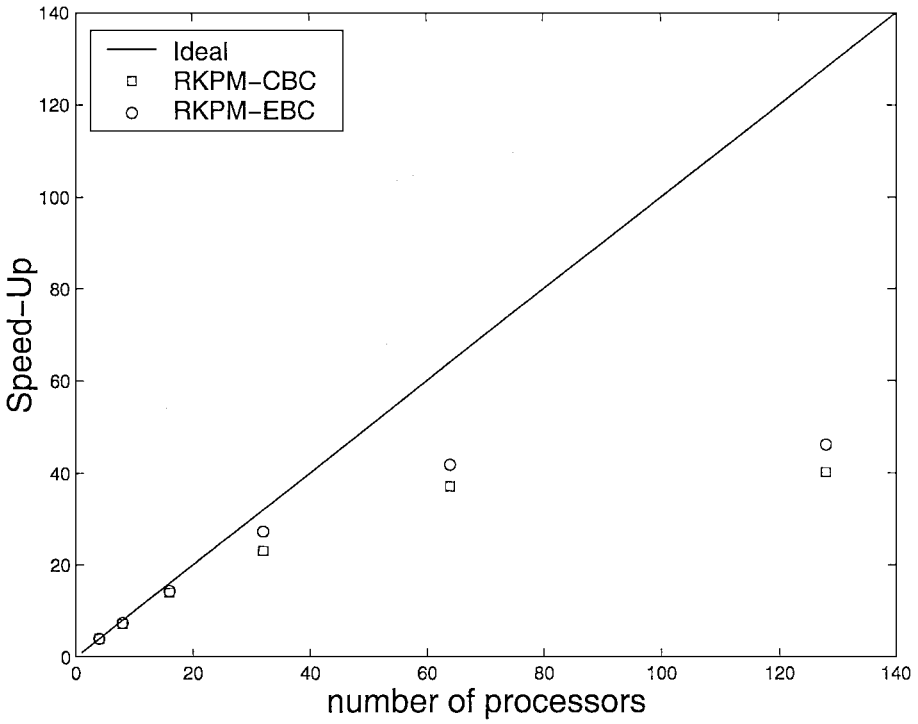


FIG. 13. Parallel performance on a CrayT3E of RKPM for the problem of flow past a building, using 38,359 nodes and 845,456 integration points (speedup is calculated based on the calculation time of using four processors). CBC, collocation boundary condition implementation; EBC, enrichment boundary condition implementation.

The computational domain extends 350 ft downstream of the building, and 150 ft upstream and to each side. It extends vertically from the base of the building to a point 100 ft above the top of the building. The domain is discretized using 38,359 nodes and 845,456 integration points. The discretization is much finer at and near the surface of the building than it is far away. The finite-element mesh on the building surface on which the meshfree nodal distribution is based is shown in Fig. 12. The boundary conditions are the same as in the previous example.

The parallel performance on the Cray T3E for this problem is shown in Fig. 13. Communication takes a large percentage of the total time when using 128 processors; therefore, speedup cannot be seen past 64 processors. Comparing the performance of the RKPM method with both boundary condition implementations shows that the enrichment boundary condition implementation is more parallelizeable.

6. CONCLUSIONS

The parallel computational implementation of the meshfree method that we have presented, together with the enrichment boundary condition scheme, has shown advantages over both the collocation boundary condition scheme and the finite-element method. In the flow past a cylinder problem, the enrichment boundary condition implementation yields results closer to the experimental values, presumably because of a better representation of the boundary layer. It has also captured the characteristics of the basic flow, such as the

formation of the vortices and vortex shedding. Another advantage is that there is no matrix inversion or transformation necessary for this application of boundary conditions. Therefore, this parallel algorithm provides better performance than do other methods. The 3-D CFD simulation examples have shown promising results in terms of accuracy and speedup of the algorithm.

ACKNOWLEDGMENTS

We would like to thank Prof. Tezduyar's computational group at Rice University for providing the original parallel finite-element code. Help from Prof. Aliabadi at Clark Atlanta University is greatly appreciated. The authors are sponsored by the Air Force Office of Scientific Research, the National Science Foundation, and the Army High Performance Computing Research Center, which provided time on its Cray T3E machine.

REFERENCES

1. I. Babuska and J. M. Melenk, *The Partition of Unity Finite Element Method*, Technical Note BN-1185 (University of Maryland, 1995).
2. M. Bar-Lev and H. T. Yang, Initial flow field over an impulsively started circular cylinder, *J. Fluid Mech.* **72**, 625 (1975).
3. S. Beissel and T. Belytschko, Nodal integration of the element-free Galerkin method, *Comput. Methods Appl. Mech. Eng.* **139**, 49 (1996).
4. A. N. Brooks and T. J. R. Hughes, Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier–Stokes equations, *Comput. Methods Appl. Mech. Eng.* **32**, 199 (1982).
5. M. A. Christon and T. E. Voth, Results for von neumann analyses for reproducing kernel semidiscretizations, *Int. J. Numer. Methods Eng.* **47**, 1285 (2000).
6. W. M. Collins and S. C. R. Dennis, The initial flow past an impulsively started circular cylinder, *Q. J. Mech. Appl. Math.* **26**, 53 (1973).
7. W. M. Collins and S. C. R. Dennis, Flow past an impulsively started circular cylinder, *J. Fluid Mech.* **60**, 105 (1973).
8. K. T. Danielson, S. Hao, W. K. Liu, A. Uras, and S. Li, Parallel computation of meshless methods for explicit dynamic analysis, *Int. J. Numer. Methods Eng.* **47**, 1323 (1999).
9. R. Gingold and J. Monaghan, Kernel estimates as a basis for general particle methods in hydronomics, *J. Comput. Phys.* **46**, 429 (1982).
10. F. C. Günther, A Meshfree Formulation for the Numerical Solution of the Viscous, Compressible, Navier-Stokes Equations, Ph.D. thesis. (Northwestern University, Chicago, 1998)
11. F. C. Günther, W. K. Liu, and M. A. Christon, Multi-scale meshfree parallel computations for viscous, compressible flows, *Comput. Methods Appl. Mech. Eng.* **190**, 279 (2000).
12. W. Han, G. J. Wagner, and W. K. Liu, Convergence analysis of a hierarchical enrichment of dirichlet boundary condition in a meshfree method. *Int. J. Numer. Methods Eng.* **53**(6), 1323 (2002).
13. T. J. R. Hughes and L. P. Franca, A new finite element formulation for computational fluid dynamics. VII. The Stokes problem with various well-posed boundary conditions: Symmetric formulations that converge for all velocity/pressure spaces, *Comput. Methods Appl. Mech. Engr.* **65**, 85 (1987).
14. T. R. Hughes, L. P. Francs, and M. Balestra, A new finite element formulation for computational fluid dynamics. V. Circumventing the babuska-brezzi condition: A stable Petrov-Galerkin formulation of the stokes problem accommodating equal-order interpolations, *Comput. Methods Appl. Mech. Eng.* **59**, 85 (1986).
15. W. K. Liu and Y. Chen, Wavelet and multiple scale reproducing kernel methods, *Int. J. Numer. Methods Fluids* **21**, 901 (1995).
16. W. K. Liu, S. Jun, and Y. F. Zhang, Reproducing kernel particle methods, *Int. J. Numer. Methods Fluids* **20**, 1081 (1995).

17. W. K. Liu, R. A. Uras, and Y. Chen, Enrichment of the finite element method with the reproducing kernel particle method, *J. Appl. Mech. ASME* **64**, 861 (1997).
18. J. J. Monaghan, Why particle methods work, *SIAM J. Sci. Stat. Comput.* **3**(4), 422 (1982).
19. T. E. Tezduyar and Y. Osawa, Finite element stabilization parameters computed from element matrices and vectors, *Comput. Methods Appl. Mech. Eng.* **190**, 411 (2000).
20. G. J. Wagner and W. K. Liu, Application of essential boundary conditions in mesh-free methods: a corrected collocation method, *Int. J. Numer. Methods Eng.* **47**, 1367 (2000).
21. G. J. Wagner, *A Multi-Scale Approach to Large Eddy Simulation via RKPM*, Master's thesis (Northwestern University, Chicago, 1999).
22. G. J. Wagner and W. K. Liu, Hierarchical enrichment for bridging scales and meshfree boundary conditions, *Int. J. Numer. Methods Eng.* **50**, 507 (2000).
23. F. White, *Fluid Mechanics* (McGraw–Hill, New York, 1986).