論文 / 著書情報
Article / Book Information

Title	PetRBF–A parallel O(N) algorithm for radial basis function interpolation with Gaussians
Authors	Rio Yokota, Lorena Barba, Matthew Knepley
Citation	Computer Methods in Applied Mechanics and Engineering, Vol. 199, No. 25-28, pp. 1793–1804
Pub. date	2010, 3
DOI	http://dx.doi.org/10.1016/j.cma.2010.02.008
Note	このファイルは著者（最終）版です。 This file is author (final) version.
Creative Commons	See next page.

License

PetRBF—A parallel $\mathcal{O}(N)$ algorithm for radial basis function interpolation

Rio Yokota[a], L. A. Barba[*,b], Matthew G. Knepley[c]

[a]*Department of Mathematics, University of Bristol, England BS8 1TW*
[b]*Department of Mechanical Engineering, Boston University, Boston MA 02215*
[c]*Computation Institute, University of Chicago*

Abstract

We have developed a parallel algorithm for radial basis function (RBF) interpolation that exhibits $\mathcal{O}(N)$ complexity, requires $\mathcal{O}(N)$ storage, and scales excellently up to a thousand processes. The algorithm uses a GMRES iterative solver with a restricted additive Schwarz method (RASM) as a preconditioner and a fast matrix-vector algorithm. Previous fast RBF methods — achieving at most $\mathcal{O}(N \log N)$ complexity — were developed using multiquadric and polyharmonic basis functions. In contrast, the present method uses Gaussians with a small variance (a common choice in particle methods for fluid simulation, our main target application). The fast decay of the Gaussian basis function allows rapid convergence of the iterative solver even when the subdomains in the RASM are very small. The present method was implemented in parallel using the PETSc library (developer version). Numerical experiments demonstrate its capability in problems of RBF interpolation with more than 50 million data points, timing at 106 seconds (19 iterations for an error tolerance of 10^{-15}) on 1024 processors of a Blue Gene/L (700 MHz PowerPC processors). The parallel code is freely available in the open-source model.

Key words: radial basis function interpolation, domain decomposition methods, GMRES, order-N algorithms, particle methods, parallel computing

1. Introduction

There are innumerable applications in computational science where one needs to perform approximation of a function based on finite data. When the data are in a certain sense "scattered" in their domain, one very powerful technique is radial basis function (RBF) interpolation. Since the early 1980s, after a comparative study of various methods for scattered-data interpolation showed the superiority of RBF methods [25], a large amount of interest and research effort has been invested in this subject. For many years, however, the wide applicability of RBF interpolation was hindered by their numerical difficulty and expense. Indeed, in their mathematical expression, RBF methods produce a linear system of size equal to the number of data points. A direct solution of such systems — requiring $\mathcal{O}(N^3)$ operations and $\mathcal{O}(N^2)$ memory usage — becomes prohibitive for more than a few thousand data points. Great progress has been made in recent years towards alleviating this computational burden. We review this progress below. The present work aims to position itself as the fastest and most efficient algorithm to date for RBF interpolation. It moreover exhibits excellent parallel scalability, and we put it to the test with up to 50 million unknowns.

In addition to scattered-data interpolation, RBF methods are now included in the broad category of meshfree methods for the solution of partial differential equations. The conventional methods for solving PDEs rely on a discretization of the domain in terms of a mesh, *i.e.*, a set of inter-connected nodes. In meshfree methods, the aim is to provide a means of discretization that relies only on a set of nodes, but with

[*]Correspondence to: 110 Cummington St, Boston MA 02215, (617) 353-3883, labarba@bu.edu
Email addresses: Rio.Yokota@bristol.ac.uk (Rio Yokota), labarba@bu.edu (L. A. Barba), knepley@ci.uchicago.edu (Matthew G. Knepley)

no knowledge of the connectivity among them. One of the motivations is that the generation of a mesh (for example, as used in finite difference and finite element methods) can be a cumbersome task, especially for problems with complex geometry and moving boundaries. It is often cited that mesh generation can take up to 80% of the time required for computational analysis in industrial applications [30]. Indeed, this is the *greatest hurdle* for the wide use of computational mechanics as a design tool.

Meshfree methods for the solution of PDEs can be grouped into two broad categories. One, as already mentioned, includes methods based on RBF interpolation. The second is based on the least squares technique. To this second class of methods belong element-free Galerkin methods [12], reproducing kernel particle methods [40], *hp*-clouds methods [20], partition of unity methods [2], and meshless local Petrov-Galerkin methods [1]. We will not refer to least squares methods further. They differ from RBF methods in that they do not satisfy the interpolation conditions at the data. In certain applications, *e.g.*, when it is known that the data contain noise, the smoothing quality of least-squares methods may be desirable. We restrict this work to the interpolating RBF methods, which have superior approximation quality without smoothing.

RBF interpolation has a number of favorable characteristics. Madych *et al.* [41] have shown that multiquadric RBF interpolation schemes converge faster as the dimension increases, and converge exponentially as the density of the nodes increases. These favorable characteristics encouraged others to use it as a tool for solving PDEs. In the early 1990s, Kansa [32, 33] used RBF interpolation in a global collocation method with multiquadrics to solve PDEs. He found that RBFs could yield a very accurate solution for parabolic and elliptic PDEs. This method has been further extended to symmetric collocation by Fasshauer [22], to modified collocation by Chen [18] and to indirect collocation by Mai-Duy [42].

Amongst the *un*favorable characteristics of RBF interpolation, the most severe is the ill-conditioning of the linear system it produces. Preconditioning can result in significant clustering of eigenvalues and improves the condition numbers of the interpolation problem by several orders of magnitude [8]. Considerable progress in the iterative solution of multiquadric and polyharmonic RBF interpolation was made with the development of preconditioners using approximate cardinal functions based on a subset of the interpolating points. This approach was first proposed by Beatson and Powell [11]. Beatson *et al.* [8] coupled this method with the GMRES iterative solver and a fast matrix-vector algorithm for polyharmonic splines to construct a method with $\mathcal{O}(N)$ storage and $\mathcal{O}(N \log N)$ complexity. Ling and Kansa [39] showed that an approximate least squares cardinal function preconditioner can be used to transform an ill-conditioned system arising from RBF solution of PDEs into a well-conditioned system. Faul *et al.* [24] used a carefully selected set of q points to construct a preconditioner for which they were able to interpolate problems with $d = 40$ dimensions and $N = 10,000$ points in a few iterations. Gumerov and Duraiswami [28] used the fast multipole method (FMM) to accelerate the matrix-vector multiplication in Faul's method and reduced the calculation cost from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$. There have been a few other studies where fast summation methods were used to accelerate the matrix-vector multiplications for polyharmonic [10, 44, 27], multiquadric [19, 45], and Gaussian [26, 45] basis functions.

When handling problems with large numbers of data points (say, millions of points), the large amount of memory usage can also become a problem. As the problem size grows, parallelization on distributed memory architectures becomes necessary. Therefore, domain decomposition methods (DDM) [47] have been gaining interest. The idea behind DDM is to divide the considered domain into a number of subdomains and then try to solve the original problem as a series of subproblems that interact through the interfaces. Hardy [29] was the first to use DDMs for RBF interpolation. Beatson *et al.* [9] used the multiplicative Schwarz method along with a coarse-grid preconditioner, and a fast matrix-vector algorithm for polyharmonic splines. Their code requires $\mathcal{O}(N)$ storage and $\mathcal{O}(N \log N)$ time, and the timings are quite impressive. There have been a few related efforts using the truncated multiquadric function [34], multi-zone methods [50], and a comparison between overlapping and non-overlapping DDMs for matching and non-matching subdomain interfaces [37]. Ling and Kansa [38] used the additive Schwarz method and found that increasing the number of subdomains not only reduces the operation count but also increases the rate of convergence. However, none of these studies mention the parallelizability of these potentially highly-parallel methods, as they all view the DDMs merely as a way to deal with the ill-conditioning of the system.

All work with DDMs mentioned above involved global and conditionally positive-definite RBFs, such as multiquadrics and thin-plate splines. For Gaussian basis functions, the choice and effectiveness of the preconditioner, DDM, and fast matrix-vector algorithm differ substantially. In fact, for Gaussian basis functions with a small standard deviation σ, the resulting system is not so ill-conditioned, and preconditioning becomes a straightforward task of truncating the effect of the far field [7]. In such a case, DDMs will provide this function of preconditioning by localizing the interaction, *i.e.*, making the bandwidth of the matrix smaller. Furthermore, the matrix-vector multiplication could be performed by a fast Gauss transform [26] if σ were much larger than the inter-node spacing h. However, for problems that have $\sigma \approx h$, it is faster to perform a direct calculation for the neighboring nodes. These specializations for the domain decomposition strategy and matrix-vector multiplication have a large positive impact on the performance and, as we show in this work, result in a method which has $\mathcal{O}(N)$ storage and $\mathcal{O}(N)$ time.

An application that will benefit tremendously from fast RBF interpolation using Gaussian basis functions with small σ, is the vortex particle method. It is now a well-known fact that particle methods for continuum systems generally require some sort of spatial adaptation, and RBF interpolation can be used in a number of different ways to handle this. The use of RBF interpolation for spatial adaptation in the vortex particle method was proposed and demonstrated in [4]. It was shown that the core-spreading method [35] coupled with RBF interpolation was more accurate than the standard particle strength exchange scheme with remeshing. A parallel version of the vortex particle method with RBF interpolation for spatial adaptation was later developed using the PETSc library [3], [6]. The parallel vortex code with RBF interpolation was later used in a comprehensive study of vortex tripole evolution [5]. The same method was also used in a calculation of homogeneous isotropic turbulence in three dimensions, where the results of the vortex particle method agreed quantitatively with those of the reference calculation using a pseudo-spectral method with the same number of calculation points [51]. However, the calculation time for the RBF interpolation became dominant, compared to the other parts of the code. Although convergence of the Krylov method was generally observed with the parameters chosen in the bulk of experiments that were performed, no investigation was pursued at that time on the conditioning and convergence of the RBF interpolation for this particular type of basis function. The generality of RBF interpolation allows the possibility to extend this method to other meshfree methods — typically the ones that involve convection of the nodes, for which the basis functions must have asymptotically local influence (like the Gaussian).

The main goal of this paper is to develop a highly-parallel RBF interpolation algorithm by using GMRES with the restricted additive Schwarz method (RASM) [16] as a preconditioner along with a fast matrix-vector multiplication. We demonstrate that, for Gaussian basis functions with small σ, it is possible to eliminate global communications completely by exploiting the asymptotic locality of the basis function. This results in $\mathcal{O}(N/N_{procs})$ storage and close to $\mathcal{O}(N/N_{procs})$ complexity, where N_{procs} is the number of processes. A serial version of the present method was investigated by Torres and Barba [48]. Their results show excellent convergence rates for different types of data, but they only show a limited portion of what this method truly has to offer. Unlike the previous studies on domain decomposition-based RBF interpolation, our focus is on the optimum subdivision of the domain, and parallel scalability.

2. RBF interpolation with domain decomposition

2.1. Background on radial basis function interpolation

The technique of radial basis function (RBF) interpolation was introduced as a tool for solving multivariate scattered data interpolation problems. There has been a large production of research results in this field, with some excellent books being published in recent years [15, 23].

The problem of scattered data interpolation is that of approximating a function f at an arbitrary point $\vec{x}$, when its values are only known on a limited set of points $\vec{x}_j \in \mathbb{R}^d,\ j = 1, ..., N$. This is done by representing the function f by the interpolant $s(\vec{x})$:

$$s(\vec{x}) = \sum_{j=1}^{N} \lambda_j\ \phi(||\vec{x} - \vec{x}_j||) + \sum_{k=1}^{M} \gamma_k\ p_k(\vec{x}), \tag{1}$$

where ϕ is the radial basis function, $||\cdot||$ denotes the Euclidean norm, and $\{p_k(\vec{x})\}_{k=1}^{M}$ is a basis for Π_M^d (space of all d-variate polynomials with degree of less than M). Micchelli [43] showed that there is a direct

Table 1: Examples of radial basis functions

Name of RBF	$\phi(r)$
Gaussian	$e^{-\epsilon^2 r^2}$
Polyharmonic	$\begin{cases} r^\nu, \nu \in 2\mathbb{N}-1 \\ r^\nu \log r, \nu \in 2\mathbb{N} \end{cases}$
Multiquadric	$(1+\epsilon^2 r^2)^\nu, \nu > 0, \nu \notin \mathbb{N}$
Inverse multiquadric	$(1+\epsilon^2 r^2)^\nu, \nu < 0$

relationship between $\phi(r)$ being positive definite and the function $\phi(\sqrt{r})$ being completely monotone. If $\phi(\sqrt{r})$ is completely monotone, then the resulting system is positive definite on its own and does not need to be augmented by polynomials, which is the case for Gaussians and inverse multiquadrics. On the other hand, if $(-1)^M \phi^{(M)}(\sqrt{r})$ is completely monotone for some $M > 0$, then $\phi(r)$ is said to be *conditionally* positive definite of order M, and requires augmentation by polynomials of degree $M-1$ to become positive definite.

The coefficients λ_j and the polynomial functions $p_k(\vec{x})$ are chosen to satisfy the fitting conditions,

$$s(\vec{x}_j) = f(\vec{x}_j), \quad j = 1, ..., N \tag{2}$$

along with the additional constraints

$$\sum_{j=1}^{N} \lambda_j p_k(\vec{x}_j) = 0, \quad k = 1, ..., M \tag{3}$$

This results in a linear system

$$\underbrace{\begin{pmatrix} \Phi & P \\ P^T & O \end{pmatrix}}_{\mathcal{A}} \underbrace{\begin{pmatrix} \vec{\lambda} \\ \vec{\gamma} \end{pmatrix}}_{\vec{x}} = \underbrace{\begin{pmatrix} \vec{f} \\ 0 \end{pmatrix}}_{\vec{b}} \tag{4}$$

where:

$$\left\{ \{\Phi_{ij} = \phi(||\vec{x}_i - \vec{x}_j||)\}_{i=1}^{N} \right\}_{j=1}^{N}, \left\{ \{P_{ij} = p_j(\vec{x}_i)\}_{i=1}^{N} \right\}_{j=N+1}^{N+M},$$
$$\left\{ \vec{\lambda} = \lambda_j \right\}_{j=1}^{N}, \{\vec{\gamma} = \gamma_j\}_{j=N+1}^{N+K}, \text{ and } \left\{ \vec{f} = f_i \right\}_{i=1}^{N}.$$

For future reference of Equation (4) we denote the coefficient matrix as $\mathcal{A}$, the solution vector as $\vec{x}$, and the right hand side as $\vec{b}$. We use the symbol O to represent the zero matrix of appropriate size, while the symbol 0 is used for the zero vector/scalar. Popular choices for the radial basis function ϕ are summarized in Table 1.

2.2. Restricted additive Schwarz method (RASM)

The idea behind domain decomposition methods is to divide the considered domain into a number of subdomains and then try to solve the original problem as a series of subproblems that interact through the interfaces. A brief explanation is given below to introduce the main concepts of domain decomposition methods, for completeness. The key components that will be explained in this section are used to describe the parallelization procedure and analyze the results later on in this paper.

We consider a square domain Ω as shown in Figure 1, having a somewhat uniform distribution of calculation points within the domain. These calculation points could be the nodes of a finite difference/element mesh or they could be particles. Let us divide the entire calculation domain Ω, into overlapping subdomains

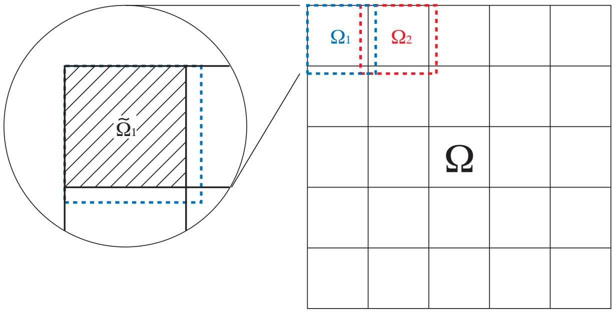


Figure 1: Illustration of Overlapping and Non-overlapping Domains

Ω_i. Also, let $\tilde{\Omega}_i$ denote the non-overlapping subdomains as shown in Figure 1. The solution of a linear system $\mathcal{A}\vec{x} = \vec{b}$ for the whole domain can be obtained by sequentially solving the individual overlapping subdomains $\mathcal{A}_i\vec{x}_{\Omega_i} = \vec{b}_{\Omega_i}$, where $\mathcal{A}_i$, $\vec{x}_{\Omega_i}$, and $\vec{b}_{\Omega_i}$ are the sub-elements associated with the domain Ω_i for $\mathcal{A}$, $\vec{x}$, and $\vec{b}$, respectively. The values $\vec{x}$ in the overlapping region can be overwritten when the next subdomain is solved. If the values $\vec{x}$ of the previous subdomains are used to update the solution of the present subdomain calculation, it is called a *multiplicative* Schwarz method. If each subdomain is solved individually and the solution of the entire domain is updated simultaneously at the end of each iteration step, it becomes an *additive* Schwarz method. Furthermore, when the values $\vec{x}$ outside of the subdomain $\tilde{\Omega}_i$ are discarded after the calculation of each subdomain Ω_i, it is called the *restricted additive* Schwarz method (RASM) [16]. The restricted additive Schwarz method is known to converge faster than the additive Schwarz method and requires less communication in parallel calculations [21]. As far as we know, RASM has not been used before within RBF interpolation with domain decomposition.

The mathematical formulae for the above methods can be written quite succinctly by defining a restriction matrix R_i,

$$\vec{x}_{\Omega_i} = R_i\vec{x} = (I \quad 0)\begin{pmatrix} \vec{x}_{\Omega_i} \\ \vec{x}_{\Omega\backslash\Omega_i} \end{pmatrix} \tag{5}$$

which restricts a vector in the whole domain $\vec{x}$ to one in the subdomain $\vec{x}_{\Omega_i}$. If the entire domain has N elements and the subdomain has N_i elements, R_i is a $N_i \times N$ matrix that restricts the vector $\vec{x}$ of size N into vector $\vec{x}_{\Omega_i}$ of size N_i. The transpose of the restriction matrix is the projection matrix,

$$\vec{x} = R_i^T\vec{x}_{\Omega_i} \tag{6}$$

which projects the vector in the subdomain onto the whole domain. Note that the R_i matrices are usually not formed in practice, but are introduced to express the different algorithms in a concise manner. Using the restriction and projection matrices, the multiplicative Schwarz method for the n-th iteration step with p overlapping subdomains can be written as

$$\begin{aligned} \vec{x}^{n+1/p} &= \vec{x}^n + R_1^T\mathcal{A}_1^{-1}R_1(\vec{b} - A\vec{x}^n) \\ \vec{x}^{n+2/p} &= \vec{x}^{n+1/p} + R_2^T\mathcal{A}_2^{-1}R_2(\vec{b} - A\vec{x}^{n+1/p}) \\ &\dots \\ \vec{x}^{n+1} &= \vec{x}^{n+(p-1)/p} + R_p^T\mathcal{A}_p^{-1}R_p(\vec{b} - A\vec{x}^{n+(p-1)/p}) \end{aligned} \tag{7}$$

where $\mathcal{A}_i = R_i\mathcal{A}R_i^T$. The n-th iteration step with p overlapping subdomains of the additive Schwarz can be expressed as

$$\vec{x}^{n+1} = \vec{x}^n + \sum_{i=1}^{p} R_i^T\mathcal{A}_i^{-1}R_i(\vec{b} - A\vec{x}^n) \tag{8}$$

The only difference between the multiplicative Schwarz method in Equation (7) and the additive Schwarz method in Equation (8) is that the multiplicative Schwarz method updates the residual after each subdomain is calculated, whereas the additive Schwarz method updates the residual only once per iteration.

Equation (8) can be extended to the restricted additive Schwarz method (RASM) by defining a projection matrix

$$\vec{x} = \tilde{R}_i^T \vec{x}_{\tilde{\Omega}_i} \tag{9}$$

which projects only the values in the *non-overlapping* subdomain $\vec{x}_{\tilde{\Omega}_i}$ onto the whole domain. The n-th iteration step of the RASM is

$$\vec{x}^{n+1} = \vec{x}^n + \sum_{i=1}^{p} \tilde{R}_i^T \mathcal{A}_i^{-1} R_i (\vec{b} - A\vec{x}^n) \tag{10}$$

where the matrix R_i restricts the residual to the overlapping subdomain Ω_i, $\mathcal{A}_i^{-1}$ solves the problem on Ω_i, and $\tilde{R}_i^T$ projects the solution in the non-overlapping subdomain $\tilde{\Omega}_i$ onto the entire domain. The matrix

$$\mathcal{M}^{-1} = \sum_{i=1}^{p} \tilde{R}_i^T \mathcal{A}_{\Omega_i}^{-1} R_i \tag{11}$$

can be viewed as a preconditioner, and can be used along with any Krylov subspace method. In the present calculation we use the GMRES [46].

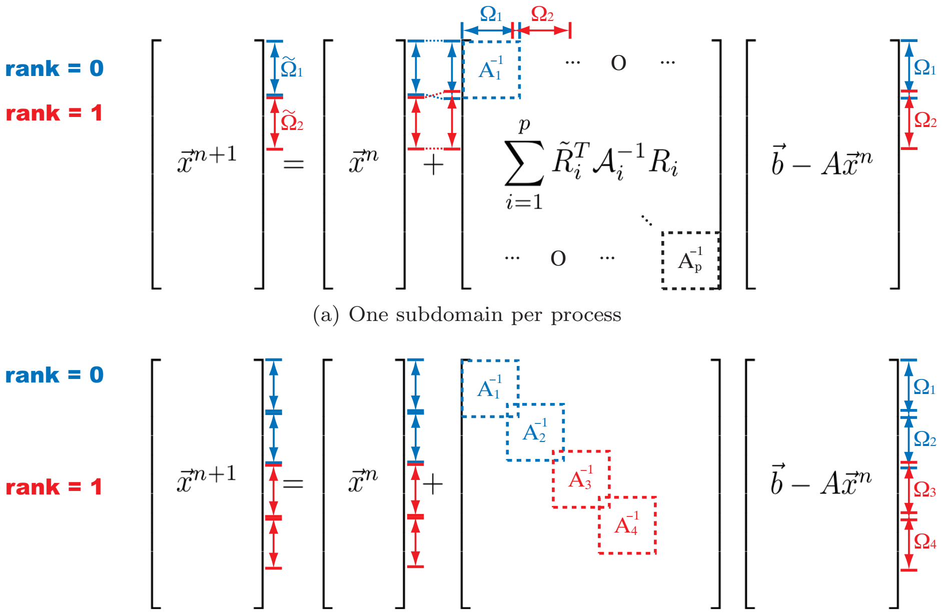
The calculation of inner products, norms, and scalar multiplication in the GMRES are not computationally intensive nor do they require large data transfer. Therefore, the efficiency and parallelism of the GMRES calculation depends on how the preconditioning is handled. When the preconditioning is done by RASM, the efficiency of the calculation is strongly affected by the size of the overlapping and non-overlapping subdomains. One needs to find the balance between having a large number of small subdomains against having a small number of large subdomains. Also, increasing the overlap region allows the GMRES to converge in fewer iterations, but each iteration will take longer because the direct solve will be performed for larger overlapping subdomains. We have performed a thorough investigation of the optimum subdomain size for our algorithm, and present the results in §3.

2.3. Parallelization of RASM

For the parallelization, we assume a distributed memory model that stores only the local components of each vector. Parallelization of the calculation of inner products, norms, and scalar multiplication are trivial, and the only noteworthy task when parallelizing the preconditioned GMRES is the preconditioning itself. When the RASM is used as the preconditioner, it can be seen that Equation (10) can be readily parallelized by assigning each subdomain to a separate process. A schematic of the parallelization of the RASM corresponding to Equation (10) is shown in Figure 2(a). The different colors represent separate processes. The "rank" refers to the rank of the MPI process. Each process owns a local portion of the global vectors $\vec{x}$ and $\vec{b}$. The partitioning of the vectors corresponds to the non-overlapping subdomains $\tilde{\Omega}_i$ shown in Figure 1. For the case of one subdomain per process, as seen in Figure 2(a), each process owns one submatrix $\mathcal{A}_i$. Each process would require a small amount of communication from other processes to calculate $\mathcal{A}\vec{x}^n$. Once this is done, the local residual for the overlapping subdomain $\vec{r}_i = \vec{b}_i - \mathcal{A}\vec{x}_i^n$ can be used as the right-hand-side to solve the local system. Then, the non-overlapping part of the solution to the local system is added to the vector $\vec{x}^n$. This is done by simply updating only the local values that belong to the process. This procedure is depicted in Figure 2(a).

For RBF interpolation using basis functions with negligible global effects, the matrix $\mathcal{A}$ can be considered to have a finite bandwidth. Thus, the calculation of $\mathcal{A}\vec{x}_i^n$ can also be done somewhat locally. Let us consider a normalized Gaussian basis function in 2D as an example. The matrix $\mathcal{A}$ has the following elements:

$$\mathcal{A}_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{|\mathbf{x}_i - \mathbf{x}_j|^2}{2\sigma^2}\right) \tag{12}$$



(a) One subdomain per process

(b) Multiple subdomains per process

Figure 2: Parallelization strategy of the RASM

Since the Gaussian function decays rapidly, the elements of matrix $\mathcal{A}$ corresponding to the interaction of distant points can be neglected. This “sparsity” of $\mathcal{A}$ depends on the relative size of the calculation domain compared to the standard deviation, σ, of the Gaussian function. For problems with relatively small σ, the calculation of $\mathcal{A}\vec{x}_i^n$ can be done by a sparse matrix-vector multiplication with controllable accuracy. If σ is kept constant while the size of the calculation domain is increased along with N, the calculation load will scale as $\mathcal{O}(N)$. The communication required to perform $\mathcal{A}\vec{x}_i^n$ is also limited to a constant number of elements in the vicinity. Therefore, the RASM becomes an extremely parallel algorithm with minimum communication for the RBF interpolation using Gaussian basis functions with small σ.

In the present work, domain decomposition is used not only to save storage, but also to increase the speed of convergence. Therefore, it is useful to have independent control over the subdomain size and the number of processes, as shown in Figure 2(b). In this case, each process stores multiple sets of $\vec{x}_i$ and $\vec{b}_i$, and loops over the the subdomain solves. The restriction from overlapping subdomains to the non-overlapping subdomains is not as straightforward as the case of one subdomain per process, because the local portion of the vector $\vec{x}$ does not necessarily correspond to the values that need to be updated. In the present algorithm, we perform the restriction by using a separate index list for the non-ovelapping subdomain.

2.4. Implementation details

The implementation of the algorithm developed here has been realized using the Portable, Extensible Toolkit for Scientific Computation (PETSc) [3]. This is a library developed over many years at Argonne National Laboratories[1]. It is fine-tuned and well-supported, and provides an interface between scientific application codes and low-level libraries such as BLAS, LAPACK, and MPI. All vectors and matrices are distributed among processes by PETSc and only the local portion is stored. Data types such as `Vec` and `Mat` allow the user to create these distributed objects and access them using global indices. The fact that all MPI communications happen “under the hood” allows users to manage parallel codes (almost) as if they were serial codes. Another feature of PETSc is that it can switch between different Krylov subspace solvers and preconditioners with just one command line option.

[1] http://www.mcs.anl.gov/petsc/

The current version of PETSc (3.0.0, released December 2008) provides routines for domain decomposition using additive Schwartz methods. The goal of these routines seems to have been using domain decomposition for parallelization only, and thus they were written to hold one subdomain per process. To fit our needs, one of the authors (Matthew G. Knepley, who is a member of the PETSc developer team) has extended the PETSc functions to handle multiple subdomains per process, and thus we are able to implement the restrictive additive Schwarz method as used in our algorithm. This extension to PETSc (currently only available in the developer version) enabled us to implement our RBF interpolation algorithm with excellent parallel scalability.

In our implementation, we use the following features of the PETSc library.

▷ vectors (for vectors $\vec{x}$ and $\vec{b}$)

▷ ghost vectors (for overlapping subdomains and matrix-vector multiplication)

▷ index sets (for setting overlapping and non-overlapping subdomains)

▷ shell matrices (for matrices $\mathcal{A}$ and $\mathcal{M}$)

▷ KSP linear solvers (GMRES, RASM)

The parallel GMRES-RASM method that has been described earlier in this section can be implemented in PETSc using the above features. The vectors are distributed among the processes and the local storage is N/N_{proc} for each vector, where N is the number of elements and N_{proc} is the number of processes. The calculation of inner products, norms, and scalar multiplication are done by calling PETSc routines. The ghost vectors handle the non-local values that are necessary for the direct solve for the overlapping subdomains and also the matrix vector multiplication of $\mathcal{A}\vec{x}$. One scatters the global vector into a local "ghosted" work vector, performs the computation on the local work vectors, and then scatters back into the global solution vector. The index sets are global indices that are used to define the elements in the overlapping and non-overlapping subdomain. Although their numbering is global, the index sets are distributed among the processes in the same way as the vectors. The shell matrices allow one to calculate the matrix vector multiplication $\mathcal{A}\vec{x}$ without actually storing the matrix $\mathcal{A}$. For the preconditioner $\mathcal{M}$, only the small matrices for the subdomain $\mathcal{A}_i$ are formed. The matrix $\mathcal{A}_i$ is then used to calculate Equation (10). After all these objects are defined, PETSc finally calculates the linear system solution using the `KSPSolve()` command, which takes the solver type GMRES (default) and the preconditioner type RASM from the command line options.

3. Parametric study of convergence rate

The rate of convergence, calculation time, storage requirements, and communication requirements of the present method depend strongly on five independent parameters. Figure 3 illustrates these parameters as they refer to the domain decomposition choices, and basis function choices. In this section, we present the results of a non-dimensionalized parametric study to demonstrate the sensitivity of the convergence rate and calculation time to these parameters.

All the calculations presented here were obtained on the Cray XT4 machine on the HECToR system at the UK National Supercomputing Service [2]. HECToR offers a total of 5664 *AMD* 2.3 GHz quad-core *Opteron* processors. The system offers 8 GB of memory per processor, which is shared between its four cores. The processors are connected with a high-bandwidth interconnect using *Cray SeaStar2* communication chips. Only *one processor* of this machine was used for obtaining the results in this section — parallel results are presented in the next section.

[2] http://www.hector.ac.uk/

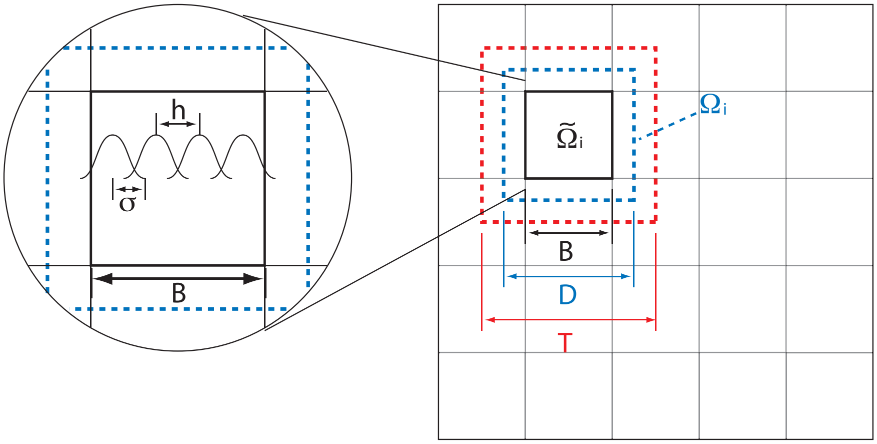


Figure 3: Illustration of parameters in the RASM for RBF interpolation using Gaussian basis functions

3.1. Calculation conditions

The whole domain is divided into non-overlapping subdomains $\tilde{\Omega}_i$ and overlapping subdomains Ω_i as shown in Figure 3. We define B as the size of the non-overlapping subdomain and D as the size of the overlapping subdomain. We also define an additional subdomain with size T, which defines the non-zero entries for the matrix-vector multiplication. Everything outside of this region is neglected when calculating $\mathcal{A}\vec{x}$ for $\tilde{\Omega}_i$. In the magnified circle in Figure 3, an illustration of the Gaussian basis functions is shown, where h is the average distance between the calculation points and σ is the standard deviation of the Gaussian distribution. When σ is large compared to h, the matrix $\mathcal{A}$ becomes ill-conditioned. In other words, the more global the basis functions are, the slower the convergence. On the other hand, if the locality was perfect and changing the coefficient λ_j in Equation 4 affects only the data at that point f_j, the linear system $\mathcal{A}$ would be diagonal, *i.e.* perfectly conditioned, but would suffer from poor approximation properties.

The following function by Franke is a standard test function for 2D scattered data fitting [25]:

$$\begin{aligned} f(x,y) &= \frac{3}{4}e^{-\frac{1}{4}((9x-2)^2+(9y-2)^2)} + \frac{3}{4}e^{-\frac{1}{49}(9x+1)^2-\frac{1}{10}(9y+1)^2} \\ &+ \frac{1}{2}e^{-\frac{1}{4}((9x-7)^2+(9y-3)^2)} - \frac{1}{5}e^{-(9x-4)^2-(9y-7)^2} \end{aligned} \tag{13}$$

We will use this function as a test case throughout the present paper. The points are distributed on a square lattice on the domain $[0,1]^2$. We also used quasi-scattered data, by shifting the position using a random number between 0 and $h/2$, where h is the average distance between points.

3.2. Parametric study of convergence rate

We first investigate the rate of convergence of the present method for different overlap ratio of the Gaussian basis function, h/σ. The size of the domain is set from 0 to 1 and $\sigma = 0.01$, which results in $N = (2/h+1)^2 = 10,201$ points for $h/\sigma = 1.0$. The size of the subdomains B is set to 5σ, the overlapping subdomains are of size $D = 1.9 \times B$, and the truncated domain for the mat-vec products has size $T = B + 4\sigma$ ($T = B + 6\sigma$ for $h/\sigma = 0.8$). These numbers are later shown to be the optimal set of parameters that result in the fastest calculation time. Upon execution of the code, we passed the following command line options to PETSc:

```
-pc_type asm -sub_pc_type lu -sub_mat_type dense
-vecscatter_alltoall -ksp_rtol 1e-13 -ksp_monitor -log_summary
```

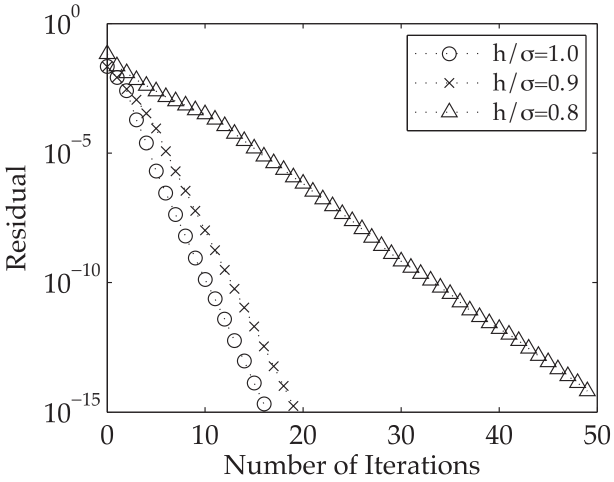


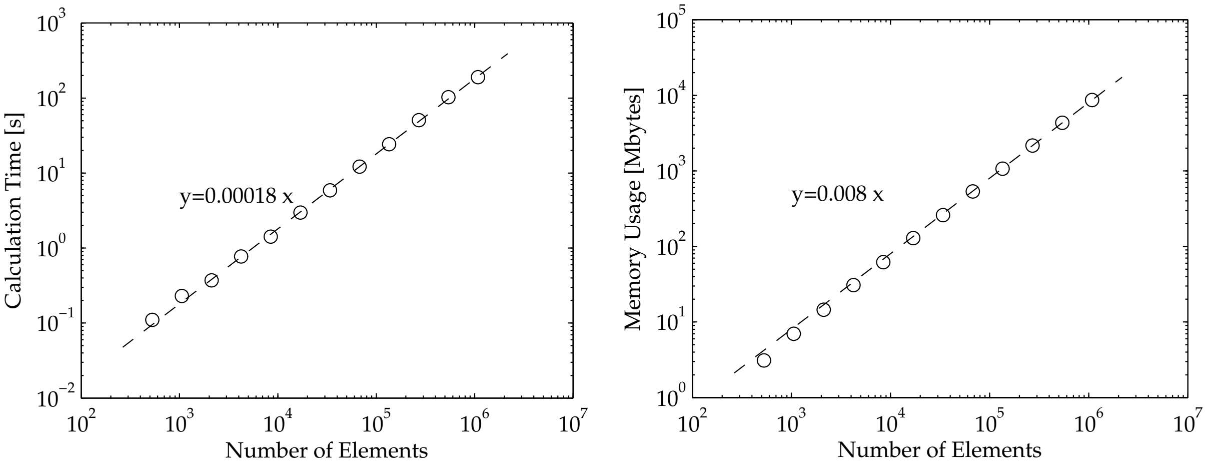
Figure 4: Convergence rate of the GMRES iteration

The l_2-norm of the residual $||r||_2$ of the GMRES iterations is plotted against the number of iterations in Figure 4. All cases converged to $||r||_2 = 10^{-15}$, but for $h/\sigma = 0.8$ the number of iterations required is significantly larger. This is due to the system $\mathcal{A}$ becoming more ill-conditioned. Nevertheless, the present method converges within 20 iterations for $h/\sigma \geq 0.9$. Furthermore, tests using larger N have shown that this convergence rate does not change with the problem size. This can be indirectly observed in the $\mathcal{O}(N)$ behavior of our method shown in Figure 5(a). Since each iteration takes $\mathcal{O}(N)$ time, the number of iterations must remain constant for the entire algorithm to scale as $\mathcal{O}(N)$. The dashed line in Figure 5(a) is the function $y = 0.00018x$, which is drawn in order to quantify the scaling exponent. The coefficient 0.00018 depends on the hardware, and implementation. The memory usage is plotted against the number of elements in Figure 5(b). The dashed line is the function $y = 0.008x$, which is drawn in order to quantify the scaling exponent. The storage requirements of the present method also scales as $\mathcal{O}(N)$.

The rate of convergence does not change with the problem size N as long as the ratio between h/σ, B/σ, and D/σ remain constant. The maximum number of N that has been calculated is $50,580,544$, and we discuss this result later in Section 4. Even for this case, the number of iterations it takes to converge to $||r||_2 = 10^{-15}$ remains exactly the same. This is a unique property of the basis function that we are using. For multiquadric and polyharmonics, some sort of hierarchical method would be required in order to have a constant convergence rate.

We now investigate the effect of changing B and D, to optimize the performance of the RASM. The number of iterations is takes to converge to $||r||_2 = 10^{-15}$ is shown in Figure 6 for different B, D and h/σ. The different plots are for different values of h/σ. The standard deviation of the Gaussian is $\sigma = 0.01$ for all cases, B/σ is changed from 3 to 7, while D is changed from $1.5B$ to $2.3B$. These parameters were selected by performing a parameter study for a larger range and determining the optimum values $B = 5\sigma$ and $D = 1.9B$, and then choosing the values close to it to see how the behavior changes. The optimum value for T is also determined by performing a parameter study for a larger range, but will not be changed here for brevity. The optimum value for T was $B + 4\sigma$ for $h/\sigma \geq 0.9$ and $B + 6\sigma$ for $h/\sigma = 0.8$, regardless of B and D.

It is useful to view Figure 6 in comparison with Figure 7, where the calculation time is shown instead of the number of iterations. Keeping B/σ constant and increasing D results in the same number of subdomains, but larger overlapping subdomains, so each direct solve for the subdomain takes longer. Since the number of subdomains remains constant, the total calculation time to go through all subdomains increases proportional to each direct solve. On the other hand, a larger overlapping subdomain gives a better convergence rate so fewer iterations will be required to achieve the same accuracy. Thus, there exists an optimum value of D



(a) Calculation time

(b) Memory usage

Figure 5: Scaling of the calculation time and memory usage against the problem size.

for which the calculation time per iteration is balanced with the required number of iterations to yield the best calculation time. Similarly, increasing B/σ results in fewer number of subdomains, but the size of each subdomain will increase along with the time it takes to solve for it. Thus, there exists an optimum value of B/σ for which the calculation time per subdomain is balanced with the number of subdomains. It can be seen from Figure 7 that the optimum set of parameters is $B = 5\sigma$ and $D = 1.9B$. This optimum set of B and D seems to be consistent throughout the present range of h/σ.

4. Scaling of the parallel implementation

There are two common measures for evaluating the performance of parallel codes: the *strong scaling* and the *weak scaling*. Strong scaling reflects how the solution time varies with the number of processes for a fixed total problem size. Weak scaling, in contrast, shows how the solution time varies with the number of processes for a fixed problem size *per process*. Throughout this paper, we strictly differentiate between processors and processes. Modern processors have multiple cores and can run multiple MPI processes without loss of performance. Thus, the proper unit for parallelization is the MPI process and not the processor.

In the previous section we showed that our algorithm scales as $\mathcal{O}(N)$ in a serial implementation. In this section, we demonstrate that in parallel our algorithm and implementation scale as $\mathcal{O}(N/N_{procs})$ by keeping N constant and changing N_{procs} (strong scaling), and keeping N/N_{procs} constant while changing N_{procs} (weak scaling).

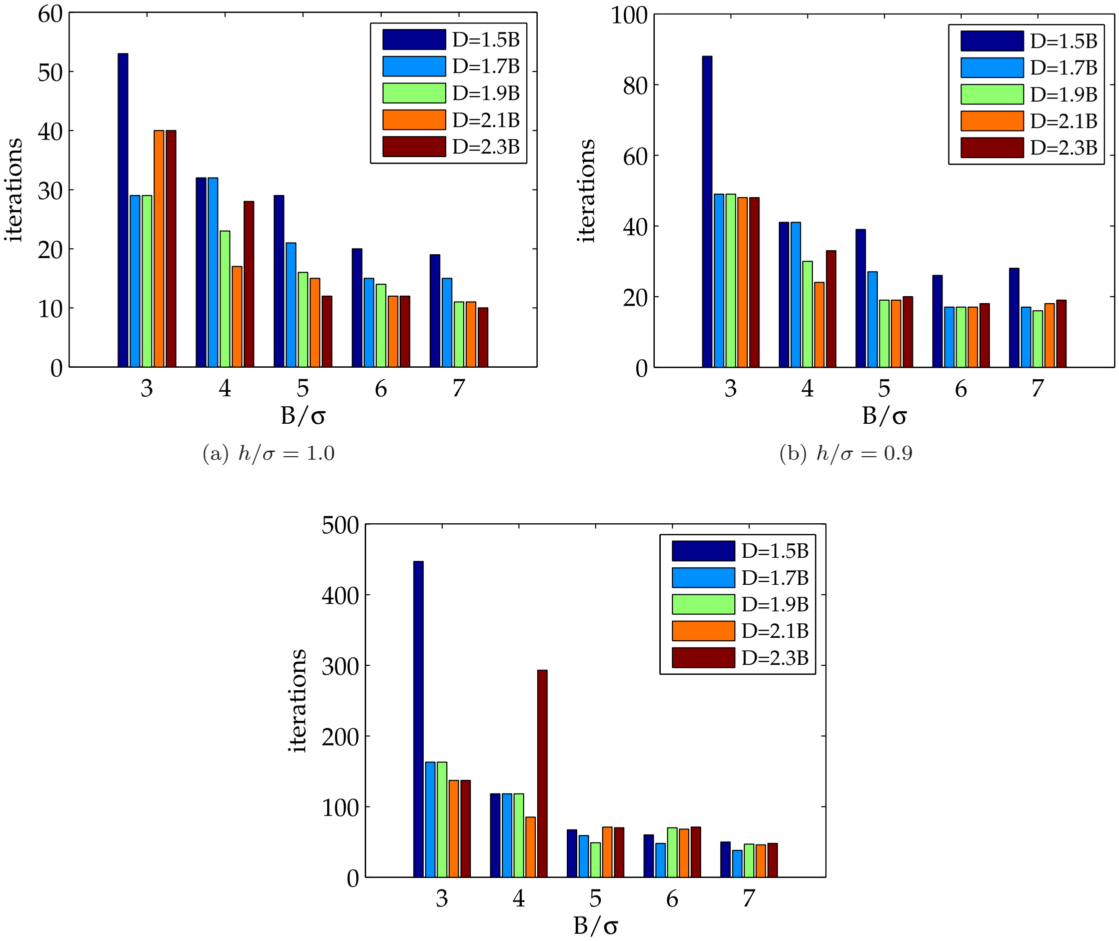
4.1. Strong scaling

For the parallel calculations in this section we used the Blue Gene/L machine at the Center for Computational Science (CCS), Boston University [3], along with the Cray XT4 machine mentioned in the previous section. The Boston University Blue Gene is a single rack system and has 1024 nodes with dual core 32-bit PPC440 processors (700 MHz) and 512 MB of main memory[4].

In the present RBF interpolation method using Gaussian basis functions with small σ, the problem is highly parallel. This is so because in addition to the fact that the RASM itself is highly parallel, the matrix vector multiplication for $\mathcal{A}\vec{x}$ can be done using only the information within the influence distance of the

[3] http://ccs.bu.edu/

[4] See http://scv.bu.edu/computation/bluegene

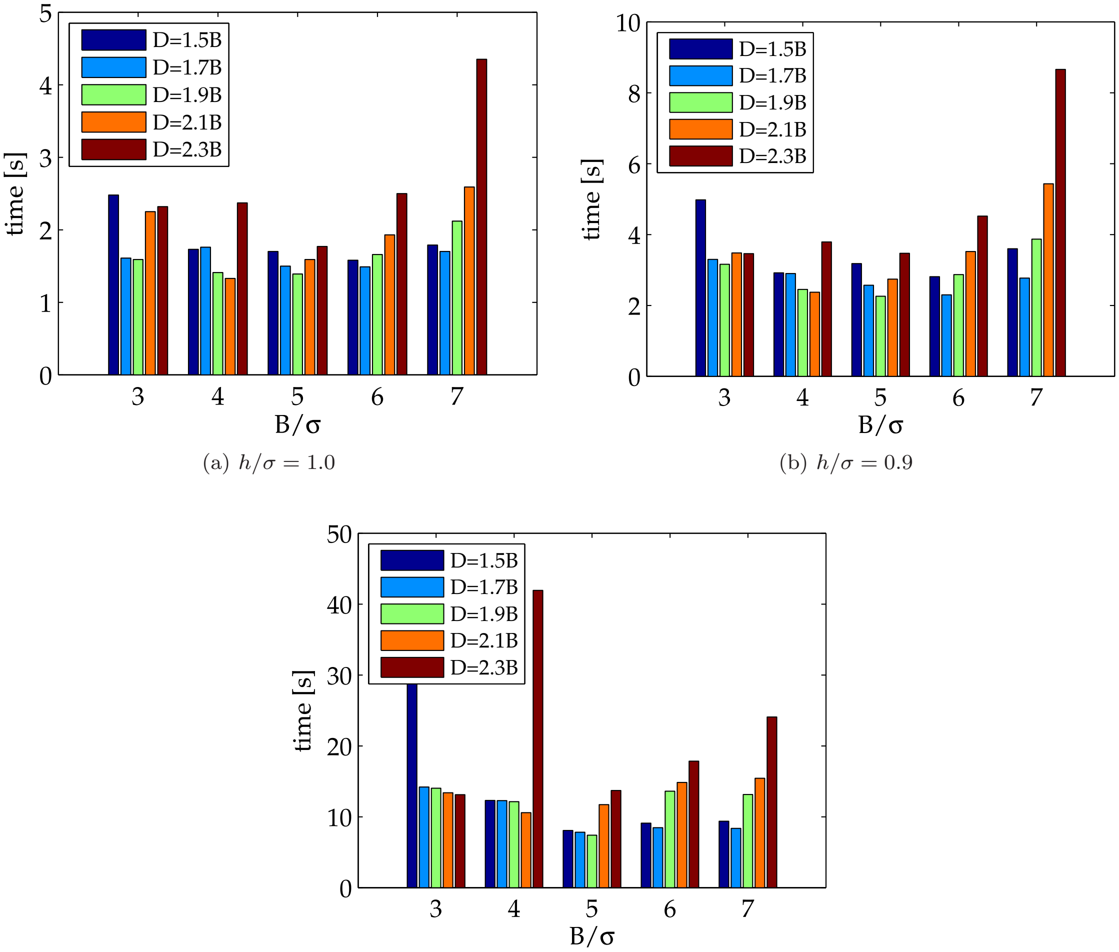


(a) $h/\sigma = 1.0$

(b) $h/\sigma = 0.9$

(c) $h/\sigma = 0.8$

Figure 6: Number of iterations for different B and D (data points are on a square lattice).

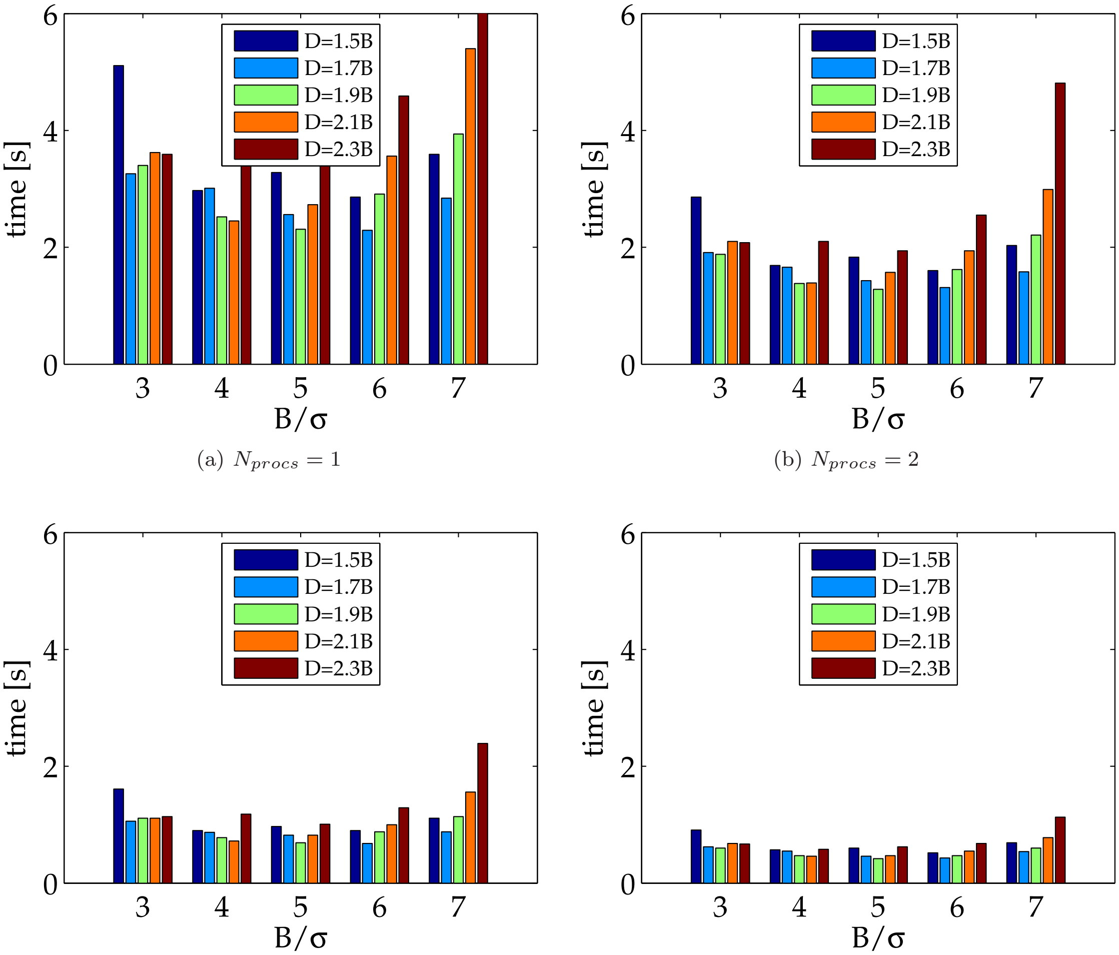


(a) $h/\sigma = 1.0$

(b) $h/\sigma = 0.9$

(c) $h/\sigma = 0.8$

Figure 7: Calculation time for different B and D (data points are on a square lattice).



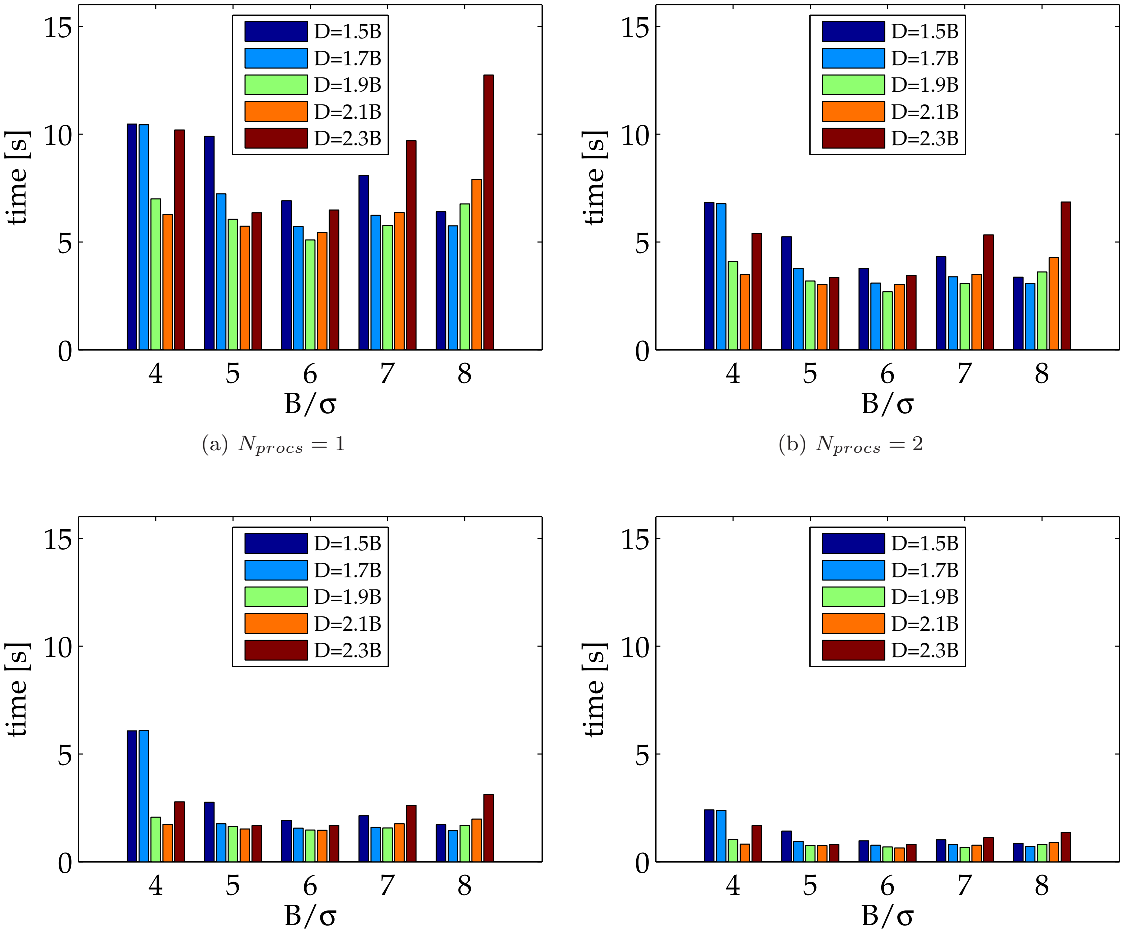
(a) $N_{procs} = 1$

(b) $N_{procs} = 2$

(c) $N_{procs} = 4$

(d) $N_{procs} = 8$

Figure 8: Calculation time for different number of processes with varying B and D ($h/\sigma = 0.9$, data points are on a square lattice) on Blue Gene/L.



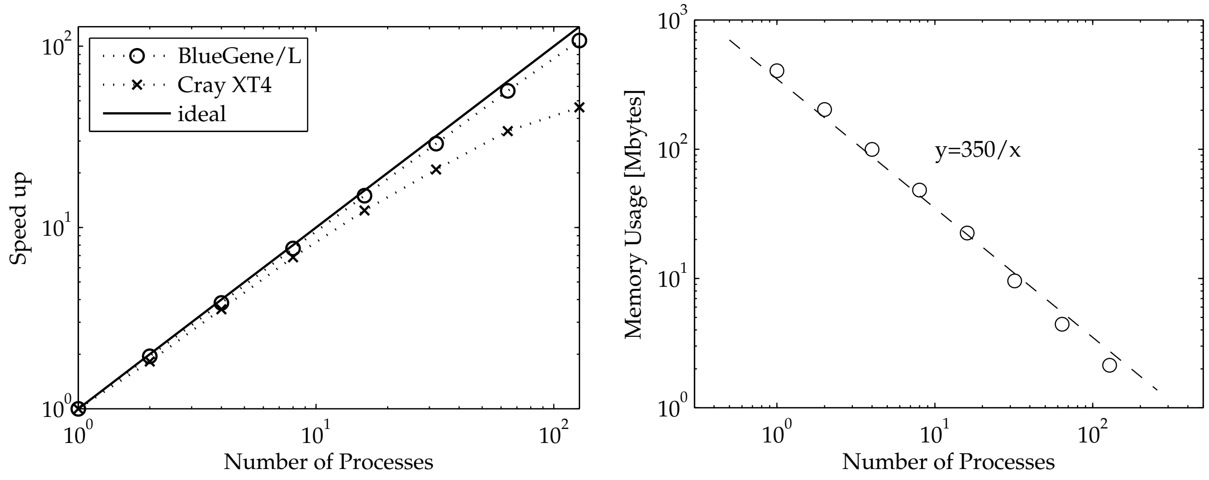
(a) $N_{procs} = 1$

(b) $N_{procs} = 2$

(c) $N_{procs} = 4$

(d) $N_{procs} = 8$

Figure 9: Calculation time for different number of processes with varying B and D ($h/\sigma = 0.9$, data points are scattered) on Blue Gene/L.



(a) Strong scaling

(b) Memory usage

Figure 10: Strong scaling and memory usage for different number of processes.

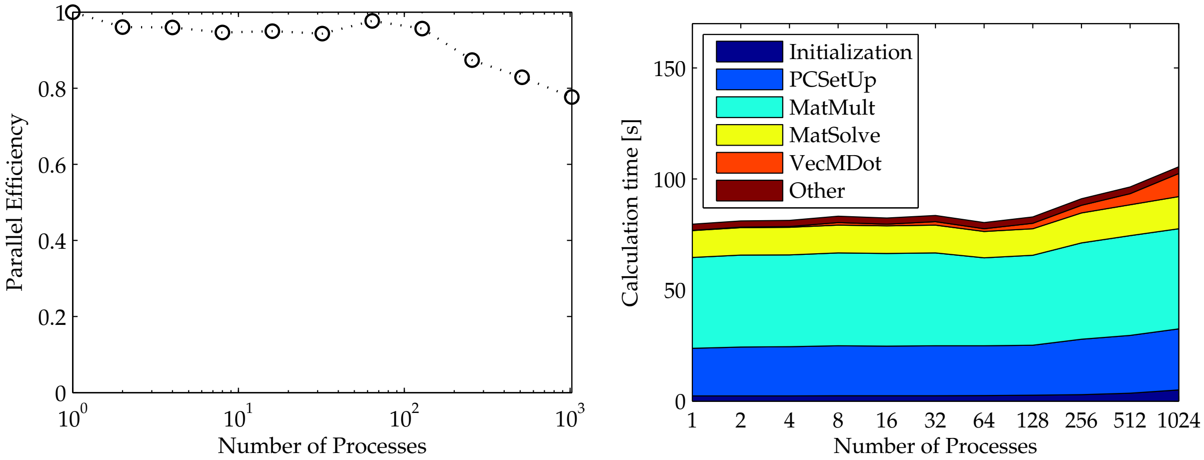
Gaussian function (T in Fig. 3). Note that this is only possible for rapidly decaying basis functions, where the matrix $\mathcal{A}$ is banded. Fast summation methods for global basis functions will involve more communication and may not scale as well for parallel calculations — this is the case in [28].

The same calculation as the one shown Figure 7 was performed in parallel. The results for 1, 2, 4, and 8 processes are shown in Figure 8. Note that the results in Figure 8(a) are the same results as the one in Figure 7(b), but the axis has been rescaled. The axes of the four plots in Figure 8 have been aligned so that the speed-up is better represented. The optimum parameter choices of $B = 5\sigma$ and $D = 1.9B$ do not change when the number of processes is increased. The scaling is not quite what we expect it to be because the problem size $N = (2/h + 1)^2 = 12,544$ is rather small, causing the otherwise insignificant serial parts of the program to degrade the scalability.

The results shown so far are for equally spaced data points on a square lattice. We now consider the effect of having scattered data points. The same calculation as in Figure 8 is performed for the same number of scattered data points. The results are shown in Figure 9. For scattered data, the optimum size of the B, D, and T domain sketched in Figure 3 changed slightly to $B = 6\sigma$, $D = 1.9B$, and $T = B + 6\sigma$ for $h/\sigma = 0.9$. Compared to the case using a square lattice, the optimum value for B increases by σ and T increases by 2σ, while D/B remains the same. The domain must be slightly larger than the square lattice case, because the random positioning of data points results in a slightly more ill-conditioned system. The calculation time also increases due to the number of iterations increasing. However, in our target application we use RBF interpolation to interpolate from scattered points onto an equally-spaced lattice distribution. Therefore, the performance of the interpolation onto scattered data points is not of primary importance. We will note that the difference is small, and proceed with our analysis for the parallel performance using the data points on a square lattice.

We now select the optimum parameter $B = 5\sigma$ and $D = 1.9B$ and test the speed-up for a larger number of processes. The calculation is performed on the Cray XT4 used above, and also a Blue Gene/L. The strong scaling results are shown in Figure 10(a). Preliminary calculations have shown that smaller h/σ gives slightly better scaling, but only the results for $h/\sigma = 0.9$ will be shown here. Also, σ is set to 0.005 for which case $N = 49,729$. The memory limitation on the Blue Gene/L 512 $MB/node$ prevented the strong scaling tests for larger calculations. The parallel efficiency using for 128 processes is 84% for the Blue Gene/L and 36% for the Cray XT4, using this test problem.

The memory usage for the strong scaling tests in shown in Figure 10(b). The dashed line is the function $y = 350/x$. It can be seen that the memory usage decreases inversely proportional (or even better) to the



(a) Parallel Efficiency

(b) Breakdown of calculation time

Figure 11: Weak scaling on the Blue Gene/L.

number of processes N_{procs}. Thus, it is safe to say that we have a storage requirement of $\mathcal{O}(N/N_{procs})$. For the weak scaling, the $\mathcal{O}(N/N_{procs})$ storage of our method allows the calculation of large-scale problems. This will be shown in the next subsection.

4.2. Weak scaling

The weak scaling on Blue Gene/L is shown in Figure 11(a). The parallel efficiency is the speed-up divided by the number of processes. Parallel efficiency equal to unity means that the parallel implementation scales perfectly with the serial implementation. The number of calculation points grows from $49,729$ (1 process) to $50,580,544$ (1024 processes). Even under the contraints of 512 $MB/node$, the $\mathcal{O}(N/N_{procs})$ storage of our code enabled the calculation of large problems. The weak scaling is almost perfect until 128 processes and gradually declines after that. The parallel efficiency for 1024 processes is 78%.

The breakdown of the calculation time for the weak scaling test is shown in Figure 11(b). “Initialization” corresponds to the allocation and initialization of all variables, “PCSetUp” is the setup of the RASM preconditioner, “MatMult” is the matrix-vector multiplication, “MatSolve” is the matrix solver for the subdomains, “VecMDot” is the dot product of vectors, and “Other” is the total of everything else. The total calculation time is the sum of all colors, which is around 80 seconds for 1 process and around 106 seconds for 1024 processes. The curve in Figure 11(a) is a vertical flip of the curve at the top of Figure 11(b). Fig. 11(b) gives a clear view of which part of the code is degrading the weak scaling performance. It is mainly the communication time in “PCSetUp” and “VecMDot” that is increasing the overall calculation time. Further tuning of these communication routines could allow the present code to scale up to tens of thousands of processors. However, such tuning is too hardware-specific to be discussed in the present paper.

5. Conclusions

This work positions itself among a series of recent efforts to provide fast algorithms for the solution of radial basis function (RBF) interpolation problems. The subject has gained much momentum in recent years, due to growing enthusiasm over meshfree methods, reflected in new books, such as [23], and various international conferences dedicated to the topic. Meshfree methods are attractive due to the infamous hurdle of mesh generation and maintenance for many applications, especially those involving complex or moving geometries. Moreover, RBF interpolation is an important technique for approximation of unknown functions based on unordered data. Applications include solution of partial differential equations [32, 33], reconstruction of surfaces for design of medical implants based on imaging [17], interpolation of geophysical data [13], modeling of groundwater contaminant transport [36], to name just a few.

In view of the multitude of important applications, the effort to offer computational efficiency has been strong. There have persistently been two challenges for this effort to succeed: the need to solve a dense and ill-conditioned linear system, and the cost of evaluating the RBF interpolant once a solution is found. This last challenge impacts on the first when using iterative methods of solution, as the internal iterations require an RBF evaluation (a dense matrix-vector multiplication). Until recently, the only way to address these challenges was to opt for using basis functions of compact support [49, 14]. The development of such compact support bases had a significant influence. It is known, however, that the approximation properties of compact support bases are inferior to global bases [15, p.150]. For this reason, several authors have continued to investigate ways to provide fast RBF interpolation with global bases.

Our contribution consists of an RBF interpolation algorithm with $\mathcal{O}(N)$ complexity, and $\mathcal{O}(N)$ storage. Moreover, we have formulated the algorithm *in parallel*. As far as we could find in the published literature, the only other parallel RBF method was presented in [31]. That work used polyharmonic basis functions, and parallel efficiency was rather modest; the largest problem reportedly solved involved $20,000$ interpolation points.

Two other RBF algorithms of note are the one developed by Beatson and others [9] and the accelerated version by Gumerov and Duraiswami [28] of a method developed in [24]. Both of these were implemented for serial computations only (as far as we are aware), and both have $\mathcal{O}(N \log N)$ complexity. In the first case, a multiplicative domain decomposition approach was used with polyharmonic basis functions. The numerical demonstrations there represent the largest problem size we could find reported, with 5 million interpolation points in 2D. In the second case, a preconditioner based on approximate cardinal functions is applied [11, 8], and both multi-quadric and polyharmonic bases are used. There, the largest calculation reported interpolated 1 million data points in 2D. As the work in [28] is very recent, the hardware used is comparable to the one used in our results. The reported timing for 1 million points was over 6000 seconds (in one 3.2 GHz processor). Although the basis function used is different, as well as the method of solution, a casual comparison can be made with our results. As shown in Figure 5(a), we have timed a solution with 1 million points at about 200 seconds (with a slower 2.3 GHz processor). Moreover, the accuracy we imposed in our experiment was higher by 7 orders of magnitude.

In light of the publications cited above, we are persuaded that the present work positions itself as the fastest and most efficient RBF algorithm to date. We aim to multiply the impact of our algorithm by realizing an implementation that is based on a well-tuned, well-supported parallel library for scientific computing, PETSc, and by offering the software free of charge, and in the open model.

In summary, our algorithm and its implementation have demonstrated good asymptotic behavior in the following aspects:

- ▷ Complexity: $\mathcal{O}(N)$ for the observed range of $N = 10^3 - 10^6$
- ▷ Storage: $\mathcal{O}(N)$ for the observed range of $N = 10^3 - 10^6$
- ▷ Convergence Rate: $\mathcal{O}(1)$ for the observed range of N (less than 20 iterations for most cases)
- ▷ Strong scaling: parallel efficiency of 84% on 128 processes for $N = 49,729$
- ▷ Weak scaling: parallel efficiency of 78% on 1024 processes for $N/N_{procs} \approx 49,729$
- ▷ Parallel storage: $\mathcal{O}(N/N_{procs})$ for the range $N_{procs} = 1 - 128$

The largest RBF interpolation problem we have calculated so far had over 50 million data points, and this calculation was timed at 106 seconds (19 iterations, for an accuracy of 10^{-15}) using 1024 processors on a Blue Gene/L supercomputer. Considering the fact that the processors of the Blue Gene/L are significantly slower than the high-end CPUs available today (about 5 times) and have only 512 $MB/node$ of memory, the performance is quite remarkable.

For information about downloading the parallel software, which has been dubbed PetRBF (for 'parallel, extensible toolkit for RBF interpolation), please visit the website at http://barbagroup.bu.edu/.

Acknowledgments

Computing time provided by the Center for Computational Science (CCS) of Boston University, on BlueGene/L, and the UK National Supercomputing Service, on the HECToR supercomputer. LAB acknowledges partial support from EPSRC under grant contract EP/E033083/1, and from Boston University College of Engineering.

References

[1] S. N. Atluri and T. Zhu. A new meshless local Petrov-Galerkin (MLPG) approach in computational mechanics. *Computational Mechanics*, 22:117–127, 1998.

[2] I. Babuska and J. M. Melenk. The partition of unity method. *International Journal for Numerical Methods in Engineering*, 40:727–758, 1997.

[3] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. Knepley, L. Curfman-McInnes, B. F. Smith, and H. Zhang. PETSc User's Manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2002.

[4] L. A. Barba. Computing high-Reynolds number vortical flows: a highly accurate method with a fully meshless formulation. In *Parallel Computational Fluid Dynamics — Multidisciplinary Applications*, pages 305–312. Elsevier B. V., 2005.

[5] L. A. Barba and A. Leonard. Emergence and evolution of tripole vortices from net-circulation initial conditions. *Phys. Fluids*, 19(1):017101, 2007.

[6] L. A. Barba, A. Leonard, and C. B. Allen. Advances in viscous vortex methods – meshless spatial adaption based on radial basis function interpolation. *Int. J. Num. Meth. Fluids*, 47(5):387–421, 2005.

[7] L. A. Barba and L. F. Rossi. Global field interpolation for particle methods. *J. Comput. Phys.*, 2009. To appear.

[8] R. K. Beatson, J. B. Cherrie, and C. T. Mouat. Fast fitting of radial basis functions: Methods based on preconditioned GMRES iteration. *Adv. Comp. Math.*, 11(2–3):253–270, November 1999.

[9] R. K. Beatson, W. A. Light, and S. Billings. Fast solution of the radial basis function interpolation equations: Domain decomposition methods. *SIAM J. Sci. Comput.*, 22(5):1717–1740, 2000.

[10] R. K. Beatson and G. N. Newsam. Fast evaluation of radial basis functions: I. *Comp. Math. Applic.*, 24(12):7–19, 1992.

[11] R. K. Beatson and M. J. D. Powell. An iterative method for thin plate spline interpolation that employs approximations to Lagrange functions. In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis*, pages 17–39. Longman Sci. Tech., 1993.

[12] Ted Belytschko, Y. Y. Lu, and L. Gu. Element-free Galerkin methods. *Int. J. Num. Meth. Eng.*, 37(2):229–256, 1994.

[13] S. D. Billings, R. K. Beatson, and G. N. Newsam. Interpolaton of geophysical data using continuous global surfaces. *Geophysics*, 67(6):1810–1822, 2002.

[14] M. D. Buhmann. Radial functions on compact support. *Proc. Edinburgh MathṠoc.*, 41(33–46), 1998.

[15] M. D. Buhmann. *Radial Basis Functions. Theory and Implementations*. Cambridge University Press, 2003.

[16] X.-C. Cai and M. Sarkis. A restricted additive schwarz preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, 21:792–797, 1997.

[17] J. C. Carr, W. R. Fright, and R. K. Beatson. Surface interpolation with radial basis functions for medical imaging. *IEEE Trans. on Medical Imaging*, 16(1):96–107, 1997.

[18] W. Chen. New RBF collocation schemes and kernel RBFs with applications. *Lecture Notes in Computational Science and Engineering*, 26:75–86, 2002.

[19] J. B. Cherrie, R. K. Beatson, and G. N. Newsam. Fast evaluation of radial basis functions: Methods for generalized multiquadrics in R^n. *SIAM J. Sci. Comput.*, 23(5):1549–1571, 2002.

[20] C. A. Duarte and J. Tinsley Oden. An *h-p* adaptive method using clouds. *Comp. Meth. Appl. Mech. Engrg.*, 139(1–4):237–262, 1996.

[21] E. Efstathiou and M. J. Gander. Why restricted additive Schwarz converges faster than additive Schwarz. *BIT Numerical Mathematics*, 43:945–959, 2003.

[22] G. E. Fasshauer. Solving partial differential equations by collocation with radial basis functions. In Le Mehaute A., Rabut C., and Schumaker L. L., editors, *Surface Fitting and Multiresolution Methods*, pages 131–138. Vanderbilt University Press, 1997.

[23] Gregory E. Fasshauer. *Meshfree Approximation Methods with MATLAB*. World Scientific, 2007.

[24] A. C. Faul, G. Goodsell, and M. J. D. Powell. A Krylov subspace algorithm for multiquadric interpolation in many dimensions. *IMA J. Numer. Anal.*, 25:1–24, 2005.

[25] R. Franke. Scattered data interpolation: Tests of some methods. *Math. Comp.*, 38(157):181–200, 1982.

[26] L. Greengard and J Strain. The fast Gauss transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, January 1991.

[27] N. A. Gumerov and R. Duraiswami. Fast multipole method for the biharmonic equation. Technical Report UMIACS-TR-2005-29, Technical Reports of the Computer Science Department, University of Maryland, 2005.

[28] N. A. Gumerov and R. Duraiswami. Fast radial basis function interpolation via preconditioned Krylov iteration. *SIAM J. Sci. Comput.*, 29(5):1876–1899, 2007.

[29] R. L. Hardy. Theory and applicatoins of the multiquadric-biharmonic method. *Computers and Mathematics with Applications*, 19(8/9):163–208, 1990.

[30] T. J. R. Hughes, J. A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Comput. Methods Appl. Mech. Engrg.*, 194(39–41):4135–4195, 2005.

[31] M. S. Ingber, C. S. Chen, and J. A. Tanski. A mesh free approach using radial basis functions and parallel domain decomposition for solving three-dimensional diffusion equations. *Int. J. Num. Meth. Eng.*, 60:2183–2201, 2004.

[32] E. J. Kansa. Multiquadrics —A scattered data approximation scheme with applications to computational fluid-dynamics, I. Surface approximations and partial derivative estimates. *Computers Math. Applic.*, 19(8/9):127–145, 1990.

[33] E. J. Kansa. Multiquadrics —A scattered data approximation scheme with applications to computational fluid-dynamics, II. Solutions to parabolic, hyperbolic and elliptic partial differential equations. *Computers Math. Applic.*, 19(8/9):147–161, 1990.

[34] E. J. Kansa and Y. C. Hon. Circumventing the ill-conditioning problem with multiquadric radial basis functions: Applications to elliptic partial differential equations. *Comp. Math. Appl.*, 39:123–137, 2000.

[35] A. Leonard. Vortex methods for flow simulation. *J. Comp. Phys.*, 37:289–335, 1980.

[36] J. Li, Y. Chen, and D. Pepper. Radial basis function method for 1-D and 2-D groundwater contaminant transport modeling. *Comput. Mech.*, 23:10–15, 2003.

[37] J. Li and Y. C. Hon. Domain decomposition for radial basis meshless methods. *Numerical Methods for Partial Differential Equations*, 20(3):450–462, 2004.

[38] L. Ling and E. J. Kansa. Preconditioning for radial basis functions with domain decomposition methods. *Mathematical and Computer Modeling*, 40:1413–1427, 2004.

[39] L. Ling and E. J. Kansa. A least-squares preconditioner for radial basis functions collocation methods. *Advances in Computational Mathematics*, 23:31–54, 2005.

[40] Wing Kam Liu, Sukky Jun, Shaofan Li, Jonathan Adee, and Ted Belytschko. Reproducing kernel particle methods for structural dynamics. *Int. J. Num. Meth. Eng.*, 38(10):1655–1679, 1995.

[41] W. R. Madych and S. A. Nelson. Multivariate interpolation and conditionally positive definite functions: II. *Math. Comp.*, 54(189):211–230, 1990.

[42] N. Mai-Duy and T. Tran-Cong. Indirect RBFN method with thin plate splines for numerical solution of differential equations. *Computer Modeling in Engineering and Sciences*, 4(1):85–102, 2003.

[43] C. A. Micchelli. Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constr. Approx.*, 2:11–22, 1986.

[44] M. J. D. Powell. Truncated Laurent expansions for the fast evaluation of thin plate splines. *Numerical Algorithms*, 5:99–120, 1993.

[45] G. Roussos and B. J. C. Baxter. Rapid evalutation of radial basis functions. *Journal of Computational and Applied Mathematics*, 180(1):51–70, 2005.

[46] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Co., Boston, 1996.

[47] B. F. Smith, P. E. Bjorstad, and W. D. Gropp. *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

[48] C. E. Torres and L. A. Barba. Fast radial basis function interpolation with Gaussians by localization and iteration. *J. Comp. Phys.*, 228:4976–4999, 2009.

[49] H. Wendland. Piecewise polynomial, positive definite and compactly supported radial basis functions of minimal degree. *Adv. Comp. Math.*, 4:389–396, 1995.

[50] S. M. Wong, Y. C. Hon, T. S. Li, and S. L. Chung. Multi-zone decomposition for simulation of time-dependent problems using the multiquadric scheme. *Computers and Mathematics with Applications*, 37:23–43, 1999.

[51] R. Yokota, T. K. Sheel, and S. Obi. Calculation of isotropic turbulence using a pure Lagrangian vortex method. *J. Comp. Phys.*, 226:1589–1606, 2007.