Contents

Chapter 1

Introduction

The process of solving partial differential equations (PDEs) using radial basis functions (RBFs) dates back to 1990 [42, 43]. At the core of all RBF methods lies the fundamental problem of approximation/interpolation. Some methods (e.g., global- and compact-RBF methods) apply RBFs to approximate derivatives directly. Others (e.g., RBF-generated Finite Differences) leverage the basis functions to generate weights for finite-differencing stencils, utilizing the weights in turn to approximate derivatives. Regardless, to track the history of RBF methods, one must look back to 1971 and R.L. Hardy's seminal research on interpolation with multi-quadric basis functions [37].

As "meshless" methods, RBF methods excel at solving problems that require geometric flexibility with scattered node layouts in d-dimensional space. They naturally extend into higher dimensions without significant increase in programming complexity [25, 75]. In addition to competitive accuracy and convergence compared with other state-of-the-art methods [22, 23, 25, 26, 75], they also boast stability for large time steps.

While most of the literature surrounding RBFs for PDEs involves collocation (see Chapter 2), the hot topic in the community today is RBF-generated Finite Differences (RBF-FD). RBF-FD is a hybrid of RBF scattered data interpolation and classical Finite Difference (FD). It shares many of the benefits from other RBF methods to generalize to scattered node layouts in any dimension, and allows for high order accurate solutions.

The idea behind classical FD is to express derivatives at a single node (center) as a weighted combination/difference of solution values from a small neighborhood (i.e., a stencil) around the center. Common approximations such as upwind differencing, center differencing, and other higher order approximations are of this form. In similar fashion, RBF-FD combines solutions values based on stencils, but it does so in a more generalized sense than standard FD stencils. For example, classical FD is typically restricted to regular meshes and often symmetric stencils in practice with the same set of weights for each stencil. Weights can be derived from polynomial expansion and obtained in 1D by solving a Vandermonde interpolation matrix [31]. Higher dimension FD stencils are composed from combinations of 1D formulas applied to each dimension. This implies restrictions on the shape/layout of stencils. In contrast to this, RBF-FD is designed for stencils with irregular node placement and can easily provide a unique set of weights for each stencil with no restrictions on stencil shape.

The concept of RBF-FD was first introduced by Tolstykh in 2000 [66], but it was the

simultaneous, yet independent, efforts in [59], [65], [74] and [9] that gave the method its real start. Introduced over a decade ago, the method is only recently showing signs that it has obtained the critical-mass following necessary for the method's use in large-scale scientific models. At the onset of this work, most of the literature considered RBF-FD for problem sizes up to a few thousand or tens of thousands of nodes. Similar to most RBF methods, RBF-FD is predominantly implemented within small-scale, serial computing environments. Under most circumstances the community at large continues investigation and extension development within MATLAB.

Our goal is to scale RBF-FD solutions on high resolution meshes across high performance clusters, and to lead the way for its adoption within HPC and supercomputing circles. Chapter 6 focuses on the problem of distributing RBF-FD across independent compute nodes, and demonstrates the scalability of RBF-FD to a thousand processors. As part of the push to HPC, leveraging Graphics Processing Units (GPUs) for computation is considered critical. GPUs, introduced in Chapter 5, are many-core accelerators capable of general purpose, embarrassingly parallel computations. Accelerators represent the latest trend in HPC, where compute nodes are commonly supplemented by one or more accessory boards for offload parallel tasks. Chapter 7 continues the discussion of RBF-FD on GPUs by tackling the problem of spanning a GPU cluster with an algorithm for overlapping communication and computation to hide the latency in data transfer between accelerators. Our effort leads the way for application of RBF-FD in an age when compute nodes with attached accelerator boards are considered key to breaching the exa-scale computing barrier [3].

1.1 On Parallel/Distributed RBF-FD

Parallel implementations of RBF methods rely on domain decomposition. Depending on the implementation, domain decomposition not only accelerates solution procedures, but can decrease the ill-conditioning that plague all global RBF methods [15]. The ill-conditioning is reduced if each domain is treated as a separate RBF domain, and the boundary update is treated separately. Domain decomposition methods for RBFs were introduced by Beatson et al. [4] in the year 2000 as a way to increase problem sizes into the millions of nodes.

This work leverages a domain decomposition, but not for the purpose of conditioning. Instead the focus is on decomposing the domain in order to scale RBF-FD across more than a thousand CPU cores of an HPC cluster. Add to this the twist of incorporating a novel implementation on the GPU with overlapping communication and computation. This combination is unmatched in related work. However, RBF methods do have a bit of history of parallel implementations.

In 2007, Divo and Kassab [15] used a domain decomposition method with artificial subdomain boundaries for their implementation of a local collocation method [15]. The subdomains are processed independently, with derivative values at artificial boundary points averaged to maintain global consistency of physical values. Their implementation was designed for a 36 node cluster, but benchmarks and scalability tests are not provided.

Kosec and Šarler [45] have the only known (to our knowledge) OpenMP implementation for RBFs. The authors parallelize coupled heat transfer and fluid flow problems on a single workstation. The application involves the local RBF collocation method, explicit

time-stepping and Neumann boundary conditions. A speedup factor of 1.85x over serial execution was achieved by executing on two CPU cores; no further results from scaling tests were provided.

Stevens et al. [64] mention a parallel implementation under development, but no document is available at this time.

Perhaps the most competitive parallel implementation of RBFs is the PetRBF [79] branch of PETSc [**?**]. The authors of PetRBF (also developers for PETSc) have implemented a highly scalable, efficient RBF interpolation method based on compact RBFs (i.e., they operate on sparse matrices). The authors demonstrate efficient weak scaling of PetRBF across 1024 processes on a Blue Gene/L, and strong scaling up to 128 processes on the same hardware. Additionally, strong scaling was tested on a Cray XT4. On the Blue Gene/L, PetRBF is demonstrated to achieve an impressive 74% parallel weak scaling efficiency on 1024 processes (operating on over 50 million points), and 84% strong scaling efficiency for 128 processes. For the Cray XT4, strong scaling tops out at 36% for 128 processes, a respectable number—and similar to observed results for our own code on 128 processes.

1.2 On GPU RBF Methods

Related work on RBFs and GPUs is sparse. In 2009, Schmidt et al. [57, 58] implemented a global RBF method for Tsunami simulation on the GPU using the AccelerEyes Jacket [1] add-on for MATLAB. Jacket provides a MATLAB interface to data structures and routines that internally call to the NVidia CUDA API. Their model was based on a single large dense matrix solve, and with the help of Jacket the authors were able to achieve approximately 7x speedup over the standard MATLAB solution on the then current generation of the MacBook Pro laptop. The authors compared the laptop CPU (processor details not specified) to the built-in NVidia GeForce 8600M GT GPU. Schmidt et al.'s implementation was the first contribution to the RBF community to leverage accelerators. The results were significant and promising, but no further contributions were made on the topic.

While both Schmidt et al.'s method and our's are based on RBFs, the two problems are only distantly related when it comes to implementation on the GPU. Dense matrix operations have a high computational complexity, are considered ideal (or near to) by linear algebra libraries like BLAS [**?**] and LAPACK [2], and were demonstrated to fit well on GPUs from the onset of General Purpose GPU (GPGPU) Computing. In fact, NVidia included CUBLAS [**?**] (a GPU based BLAS library for their hardware) with their initial public release of the game-changing CUDA development kit in 2006. In stark contrast to this, sparse matrix operations have minimal computational complexity and are less than ideal for the GPU.

Earlier this year (2013), Cuomo et al. [13] implemented RBF-interpolation on the GPU for surface reconstruction. Their implementation utilizes PetRBF [79], and new built-in extensions that allow GPU access within PETSc. PETSc internally wraps the CUSP project [**?**] for sparse matrix algebra on the GPU. With the help of these libraries, Cuomo et al. solve and apply sparse interpolation systems on the GPU for up to three million nodes on an NVidia Fermi C1060 GPU (4GB). They compare results to a single core CPU implementation on an Intel i7-940 CPU and demonstrate that the GPU accelerate their

solutions between 6x and 25x. Unfortunately, the authors do not show evidence of scaling the interpolation across multiple GPUs; so while evidence exists that PetRBF now has full GPU support, it remains to be seen how well the code can scale in GPU mode.

1.3 On Multi-GPU Methods

Multi-GPU Jacobi iteration for Navier stokes flow in cavity http://scholarworks.boisestate.edu/cgi/viewcontent.cgi?article=1003&context=mecheng_facpubs

Thibault et al. have multiple works on Multi-GPU and overlapping comm and comp.

Part I

Preliminaries

Chapter 2

RBF Methods for PDEs

The process of solving partial differential equations (PDEs) using radial basis functions (RBFs) dates back to 1990 [42, 43]. However, at the core of all RBF methods lies the fundamental problem of approximation/interpolation. Some methods (e.g., global- and compact-RBF methods) apply RBFs to approximate derivatives directly. Others (e.g., RBF-generated Finite Differences) leverage the basis functions to generate weights for finite-differencing stencils, utilizing the weights in turn to approximate derivatives. Regardless, to track the history of RBF methods, one must look back to 1971 and R.L. Hardy's seminal research on interpolation with multi-quadric basis functions [37].

As "meshless" methods, RBF methods excel at solving problems that require geometric flexibility with scattered node layouts in d-dimensional space. They naturally extend into higher dimensions without significant increase in programming complexity [25, 75]. In addition to competitive accuracy and convergence compared with other state-of-the-art methods [22, 23, 25, 26, 75], they also boast stability for large time steps.

This chapter is dedicated to summarizing the four-decade history of RBF methods leading up to the development of the RBF-generated Finite Differences (RBF-FD) method. Beginning with a brief introduction to RBFs and a historical survey, we attempt to classify related work into three three types: global, compact, and local methods. Following this, the general approximation problem is introduced, with a look at the core of all three method classificiations: RBF scattered-data interpolation.

Three global RBF collocation methods are presented: Kansa's method, Fasshauer's method and Direct collocation. Within the historical context of RBF methods we highlight extensions that lead to local interpolation matrices instead of a single global interpolation matrix. Additionally, the RBF-pseudospectral (RBF-PS) method is shown as an extension to fit global RBF methods into the framework of lower complexity pseudo-spectral methods.

By surveying related work in the field of RBF PDE methods, we frame the context in which RBF-FD was developed, and illustrate both the benefits and pitfalls inherited from its predecessors.

2.1 Survey of Related Work

In Radial Basis Function methods, radially symmetric functions provide a non-orthogonal basis used to interpolate between nodes of a point cloud. RBFs are univariate and a function of distance from a center point defined in $\mathbb{R}^d$, so they easily extend into higher dimensions without significant change in programming complexity. Examples of commonly used RBFs from the literature are provided in Table 2.1; 2D representations of the same functions can be found in Figure 2.1. Figure 2.2 illustrates the radial symmetry of RBFs—in this case, a Gaussian RBF—in the first three dimensions.

RBF methods are based on a superposition of translates of these radially symmetric functions, providing a linearly independent but non-orthogonal basis used to interpolate between nodes in d-dimensional space. The interpolation problem—referred to as *RBF scattered data interpolation*—seeks the unknown coefficients, $\mathbf{c} = \{c_j\}$, that satisfy:

$$\sum_{j=1}^{N} \phi_j(r(\mathbf{x}))c_j = f(\mathbf{x}),$$

where $\phi_j(r(\mathbf{x}))$ is an RBF centered at $\{\mathbf{x}_j\}_{j=1}^n$. In theory the radial coordinate, $r(\mathbf{x})$, could be any distance metric, but is most often assumed to be $r(\mathbf{x}) = ||\mathbf{x} - \mathbf{x}_j||_2$ (i.e., Euclidean distance), as it is here. The coefficients $\mathbf{c}$ result in a smooth interpolant that collocates sample values $f(\mathbf{x}_j)$. An example of RBF interpolation in 2D using 15 Gaussians is shown in Figure 2.3.

RBFs have been shown in some cases to have exponential convergence for function approximation [19]. It is also possible to reformulate RBF methods as pseudospectral methods that have generated solutions to ill-posed problems for which Chebyshev-based and other pseudospectral methods fail [17]. However, as with all methods, RBFs come with certain limitations. For example, RBF interpolation is—in general—not a well-posed problem, so it requires careful choice of positive definite or conditionally positive definite basis functions (see [19, 40] for details).

RBFs depend on a shape or support parameter ϵ that controls the width of the function. The functional form of the shape function becomes $\phi(\epsilon\, r(\mathbf{x}))$. For simplicity in what follows, we use the notation $\phi_j(\mathbf{x})$ to imply $\phi(\epsilon||\mathbf{x} - \mathbf{x}_j||_2)$. Decreasing ϵ increases the support of the RBF and in most cases, the accuracy of the interpolation, but worsens the conditioning of the RBF interpolation problem [56]. This inverse relationship is widely known as the *Uncertainty Relation* [40, 56]. Fortunately, recent algorithms such as Contour-Padé [29] and RBF-QR [30, 33] allow for numerically stable computation of interpolants in the nearly flat RBF regime (i.e., $\epsilon \to 0$) where high accuracy has been observed [34, 46].

RBF methods for interpolation first appeared in 1971 with Hardy's seminal research on multiquadrics [37]. In his 1982 survey of scattered data interpolation methods [35], Franke rated multiquadrics first-in-class against 28 other methods (3 of which were RBFs) [35]. Many other RBFs, including those presented in Table 2.1 have been applied in literature, but for PDEs in particular, few can rival the attention received by multiquadrics. Recently, however, Gaussian RBFs are on the rise due to recent advances in eigenvalue stabilization and new methods for investigating the $\epsilon \to 0$ regime (see e.g., [30, 31]).

Name	Abbrev.	Formula	Order (m)
Multiquadric	MQ	$\sqrt{1+(\varepsilon r)^2}$	1
Inverse Multiquadric	IMQ	$\frac{1}{\sqrt{1+(\varepsilon r)^2}}$	0
Gaussian	GA	$e^{-(\varepsilon r)^2}$	0
Thin Plate Splines	TPS	$r^2 ln\|r\|$	2
Wendland (C^2)	W2	$(1-\varepsilon r)^4(4\varepsilon r+1)$	0

Table 2.1: Examples of frequently used RBFs based on [19, 34]. ε is the support parameter. All RBFs have global support. For compact support, enforce a cut-off radius (see Equation 2.1).

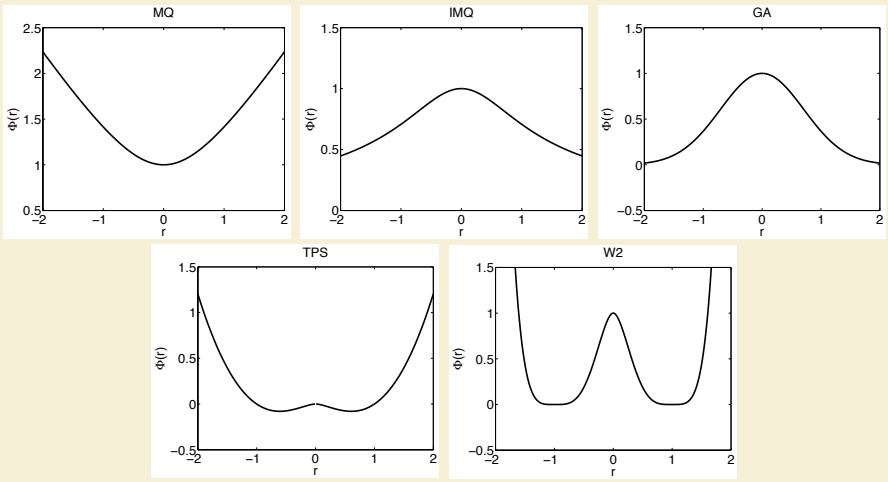


Figure 2.1: Example RBF shapes from Table 2.1 with parameter $\varepsilon = 1$.

By 1990, the understanding of the scientific community regarding RBFs was sufficiently developed for collocating PDEs [42, 43]. PDE collocation seeks a solution of the form

$$(\mathcal{L}u)(x_i) = \sum_{j=1}^{N} \phi_j(x_i)c_j = f(x_i)$$

where $\mathcal{L}$ is, in general, a nonlinear differential operator acting on $u(x)$. The solution $u(x)$ is expressed as a linear combination of N basis functions $\phi_j(x)$, not necessarily RBFs:

$$u(x) = \sum_{i=1}^{N} \phi_j(x)c_j$$

As in the problem of RBF scattered data interpolation, $\mathbf{c} = \{c_j\}$ is the unknown coefficient vector. Under the assumption that $\mathcal{L}$ is a linear operator, one can collocate the differential

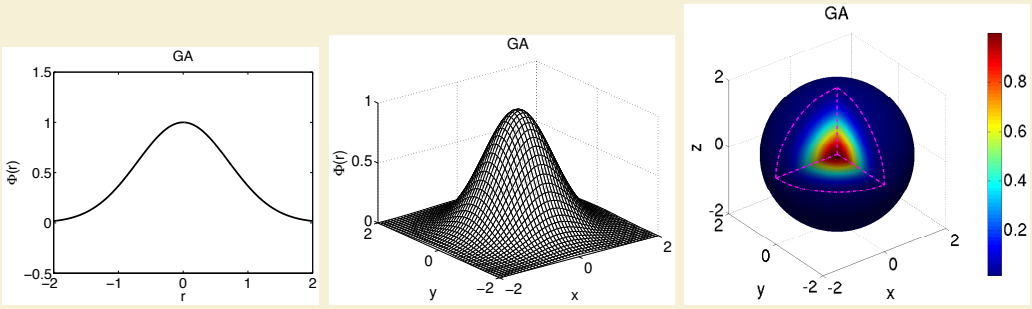


Figure 2.2: The Gaussian (GA) RBF (Table 2.1) with parameter $\varepsilon = 1$ and r in $D = 1$, 2 and 3.

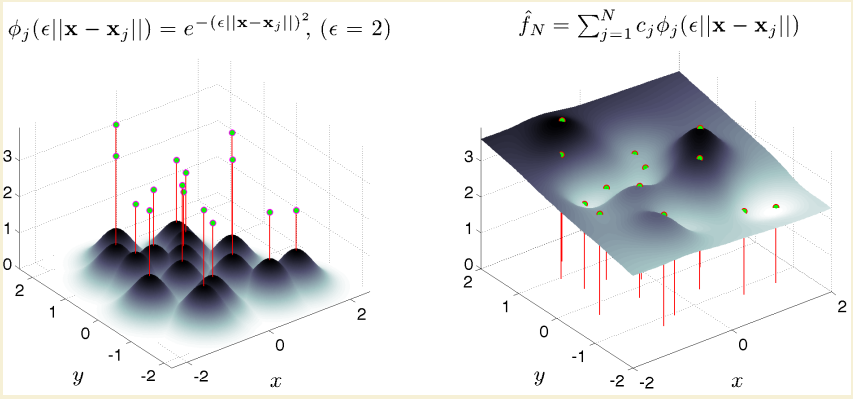


Figure 2.3: RBF interpolation using 15 translates of the Gaussian RBF with $\epsilon = 2$. One RBF is centered at each node in the domain. Linear combinations of these produce an interpolant over the domain passing through known function values.

equation. Alternatively, individual derivative operators can be expressed as linear combinations of the unknowns u_j (leading to the RBF-FD methods). In all cases, a linear system of equations arises, with different degrees of sparsity, dependent on the chosen basis functions and how the various constraints are enforced. While we restrict the $\phi_j(x)$ to RBFs or various operators applied to the RBFs, we note that spectral methods, finite-element or spectral-element methods can be formulated in a similar way with different choices of basis functions. Of course, u can be a vector of unknown variables ($\mathbf{c}$ then becomes a matrix).

Table 2.3 classifies references according to their choice of collocation method and RBF interpolation type. There are three main categories of RBF interpolation listed in Table 2.2. The first is *Global* in the case that a single, large ($N \times N$) and dense matrix corresponding to globally supported RBFs is inverted; second, *Compact* if compactly supported RBFs are used to produce a single, large, but *sparse* matrix; and third, *Local* if compactly supported RBFs are used to produce many small but dense matrices with one corresponding to each collocation point. In all three cases the matrices are symmetric and with the correct choice of RBF they are at least conditionally positive definite. The final row of Table 2.3 considers literature on the RBF-FD method and is discussed in depth in Chapter 3.

We note that three types of collocation occur throughout the RBF literature: Kansa's unsymmetric collocation method [42, 43], Fasshauer's symmetric collocation method [18],

Interpolation Type	Dense/Sparse A	Dim(A) ($N_S \ll N$)	# of A^{-1}	RBF Support
Global	Dense	$N \times N$	1	Global
Compact	Sparse	$N \times N$	1	Compact
Local	Dense	$N_S \times N_S$	N	Global/Compact

Table 2.2: RBF interpolation types and properties, assuming a problem with N nodes.

and the Direct collocation method [21].

We now turn to discussion of the benefits and shortcomings of each RBF method, before covering derivation of the methods.

2.1.1 Global RBF Methods

Kansa's method [42, 43] (a.k.a. unsymmetric collocation) was the first RBF method for PDEs, and is still the most frequently used method. The idea behind Kansa's method is that an approximate solution to the PDE can be found by finding an interpolant which satisfies the differential operator with zero residual at a set of *collocation points* (these coincide with the RBF centers). To find the interpolant, the differential equation is formulated as a two block (unsymmetric) linear system with: 1) the approximation of values at boundary points with boundary data only, and 2) the approximation of interior points by directly applying the differential operator. It was shown in [18, 38] that the unsymmetric linear system produced by Kansa's method does not guarantee non-singularity; although it is also noted that in practice singularities are rare encounters [46].

The second alternative for RBF collocation, is based on Hermite scattered data interpolation (see [77]). The so-called *Fasshauer* or *Symmetric Collocation* method ([18]) performs a change of basis for the interpolant by directly applying the differential operator to the RBFs. It then collocates using the same approach as Kansa's method [46, 62]. The resulting block structure of the linear system is symmetric and guaranteed to be non-singular [18]. In comparison to Kansa's method, the disadvantages of Fasshauer's method include: a) requirement of higher order differentiability of the basis functions (to satisfy double application of the differential operator) and b) the linear system is larger and more complex to form [19]. As [38] points out, the possible existence of a singularity in Kansa's method is not enough to justify the added difficulties of using Fasshauer's method.

The last collocation method, *Direct Collocation*, was introduced by Fedoseyev, Friedman and Kansa [21] and satisfies the differential operator on the interior and the boundary. Larsson and Fornberg [46] observe that this third method has a matrix structure similar to that found in Kansa's method; however, it is noted that the dimensions of the matrix blocks for each method differ. This is due to collocation constraints added for the differential operator applied to the boundary. Aside from the survey on RBF collocation presented by Larsson and Fornberg [46], no related work was found that applied, or investigated, this method further.

Both Kansa's method and Fasshauer's methods were shown in [17] to fit well in the gen-

eralized framework of pseudo-spectral methods with a subtle change in algorithm. While collocation methods explicitly compute the coefficients for a continuous derivative approximation, their alternates, referred to in literature as RBF-pseudospectral (RBF-PS) methods, never explicitly compute the interpolant coefficients. Instead, a differentiation matrix (DM) is assembled and used to approximate derivates at the collocation points only [20]. Since most computational models are simply concerned with the solution at collocation points, the change to assemble DMs as in RBF-PS is organic.

Following the evolution of the RBF-PS algorithm, applications of global RBFs in the classic collocation sense (i.e., without the RBF-PS DMs) become impractical. This statement stems from the algorithmic complexity of each method. Global RBF methods result in full matrices [19]. The global collocation methods then scale on the order of $O(N^3)$ floating point operations (FLOPs) to solve for weighting coefficients on a given node layout, plus $O(N^2)$ to apply the weights for derivatives. If time-stepping is required, global collocation methods must recompute the time-dependent coefficients with additional cost dominated by $O(N^3)$ operations. RBF-PS methods have similar requirements for $O(N^3)$ operations to assemble the differentiation matrix and $O(N^2)$ to apply for derivatives. However, by avoiding time-dependent coefficients, the differentiation matrix application at each time-step is only $O(N^2)$ operations. As an aside, the $O(N^3)$ complexity for each method—typically due to an LU-decomposition, with subsequent forward- and back-solves—could be reduced. While not in mainstream use by the RBF community, [49] correctly points out that iterative solvers could be employed for $O(N^2)$ complexity.

Hon et al. [39] employed Kansa's method to solve shallow water equations for Typhoon simulation. In [26], Flyer and Wright employed RBF-PS (Kansa method) for the solution of shallow water equations on a sphere. Their results show that RBFs allow for longer time steps with spectral accuracy. The survey [22] by Flyer and Fornberg showcases RBF-PS (Kansa's method) out-performing some of the of the best available methods in geosciences, namely: Finite Volume, Spectral Elements, Double Fourier, and Spherical Harmonics. When applied to problems such as transport on the sphere [25], shallow water equations [26], and 3D mantle convection [75], RBF-PS consistently required fewer time steps, and a fraction of the nodes for similar accuracy [22].

2.1.2 Compactly Support RBFs

Thus far, all cases of collocation and interpolation mentioned have assumed globally supported RBFs. While global RBFs are well-studied and have nice properties, a major limitation is the large, dense system that must be solved. One alternative to global support is to use a set of compactly supported RBFs (CSRBFs) that are defined as:

$$\phi(r) = \begin{cases} \varphi(r) & r \in [0,1] \\ 0 & r > 1 \end{cases} \tag{2.1}$$

where a cut-off radius is defined past which the RBF (in this case $\varphi(r)$) has no influence on the interpolant. Note that the radius can be scaled to fit a desired support. Methods that leverage CSRBFs produce a global interpolation matrix that is *sparse* and therefore results in a system that is more efficiently assembled and solved with smaller memory requirements [19]. The actual complexity estimate of the CSRBF method depends on the sparsity of the

	RBF Interpolation Type		
Method	Global (Dense)	Compact (Sparse Global)	Local
Kansa's Method	[17, 22, 25, 26, 29, 34, 35, 38, 39, 42, 43, 46, 50, 58, 75]	[34, 70]	[15, 45, 67, 68]
Fasshauer's Method	[17, 18, 46]	[48]	[62, 63, 64]
Direct Collocation	[21, 46]		
RBF-FD	N/A	N/A	[9, 10, 14, 23, 24, 30, 31, 59, 60, 73, 74, 76]

Table 2.3: Classification of references based on choice of RBF interpolation types and method for solving PDEs. References may appear in multiple cells according to the breadth of their research.

problem as well as the ordering of the assembled system. Assuming $n \ll N$ where n represents the number of nodes in support, [80] lists the complexity as dominated by $O(N)$ for properly structured systems within MATLAB, and the investigation in [49] found $O(N^{1.5})$ consistent with the estimate provided by their choice of general sparse solver package. A multi-level CSRBF method, introduced by Fasshauer [19], collocates solutions over multiple grid refinements to achieve reduced $O(N)$ complexity, but the method is plagued by poor convergence. It is also worth noting that in the context of CSRBFs, analogues to Kansa's method and Fasshauer's method are known by the names *radial point interpolation method (RPIM)* [70] and *radial point interpolation collocation method (RPICM)* [48], respectively. A more thorough survey of CSRBF history can be found in [19, 40].

CSRBFs have attracted a lot of attention in applications. For example, in the field of dynamic surface and image deformation, compact support allows for local transformations which do not induce global deformation (see e.g., [12, 47, 78]).

2.1.3 Local RBF Methods

Around 2005, Šarler and Vertnik [67, 68] demonstrated that if compactly supported RBFs are chosen, the traditional global collocation matrix from Kansa's method, can be avoided altogether in favor of small localized collocation matrices defined for each node. Local collocation still faces possible ill-conditioning and singularities like global collocation, but make it easier to distribute computation across parallel systems. Also, the smaller linear systems can be solved with less conditioning issues. In [68], the authors consider 2D diffusion problems. Divo and Kassab [15] employ the method for Poisson-like PDEs including fluid flow and heat transfer. Kosec and Šarler [45] apply the same technique to solve coupled heat transfer and fluid flow problems.

In similar fashion, Stevens et al. [64] introduced a local version of Fasshauer's method called *local Hermitian interpolation*. The authors have applied their method to 3D soil problems based on transient Richards' equations [62, 63, 64].

2.2 Comparison of RBF Methods

We now detail RBF methods for PDEs leading up to the derivation of RBF-FD.

Following [50], consider a PDE expressed in terms of a (linear) differential operator, $\mathcal{L}$:

$$\begin{aligned} \mathcal{L}u &= f \quad \text{on } \Omega \\ u &= g \quad \text{on } \Gamma \end{aligned}$$

where Ω is the interior of the physical domain, Γ is the boundary of Ω and f, g are known explicitly. In the case of a non-linear differential operator, a Newton's iteration, or some other method, can be used to linearize the problem (see e.g., [76]); of course, this increases the complexity of a single time step. Then, the unknown solution, u, which produces the observations on the right hand side can be approximated by an interpolant function u_ϕ expressed as a linear combination of radial basis functions, $\{\phi_j(x) = \phi(||x - x_j||)\}_{j=1}^N$, and polynomial functions$\{P_l(x)\}_{l=1}^M$:

$$u_\phi(x) = \sum_{j=1}^{N} \phi_j(x) c_j + \sum_{l=1}^{M} P_l(x) d_l, \qquad P_l(x) \in \Pi_p^d \tag{2.2}$$

where $\phi_j(x) = ||x - x_j||_2$ (Euclidean distance). The second sum represents a linear combination of polynomials that enforces zero approximation error when $u(x)$ is a polynomial of degree less than or equal to p. The variable d is the problem dimension (i.e., $u_\phi(x) \in \mathbb{R}^d$). To eliminate degrees of freedom for well-posedness, p should be greater than or equal to the order of the chosen RBF (see Table 2.1) [40]. Note that Equation 2.2 is evaluated at $\{x_j\}_{j=1}^N$ data points through which the interpolant is required to pass with zero residual. We refer to the x_j's as *collocation points* (a.k.a. trial points), taken as the RBF centers. The test points, x, usually coincide with collocation points, although this is not a requirement.

To clarify the role of the polynomial part in Equation 2.2, it is necessary to put aside the PDE for the moment and consider only the problem of *scattered data interpolation* with Radial Basis Functions.

2.2.1 RBF Scattered Data Interpolation

Borrowing notation from [19, 40], we seek an interpolant of the form

$$f(x) = \sum_{j=1}^{N} \phi_j(x) c_j$$

where $f(x)$ is expressed as a scalar product between the unknown coefficient weights c_j and the radial basis functions $\phi_j(x)$.

To obtain the unknown coefficients, c_j, form a linear system in terms of the N RBF centers:

$$\begin{aligned} f(x) &= \sum_{j=1}^{N} c_j \phi_j(x) \quad \text{for } x = \{x_j\}_{j=1}^{N} \\ \left(\ f \ \right) &= \left[\ \phi \ \right] \left(\ c \ \right) \end{aligned}$$

The invertibility of this system depends on the choice of RBF, so one typically chooses a function that is positive definite to avoid issues. It has been shown (see [19, 40]) that some choices of RBFs (e.g. multiquadrics and thin-plate splines [38]) are not positive definite and therefore there is no guarantee that the approximation is well-posed. A sufficient condition for well-posedness is that the matrix be *conditionally positive definite*. In [19], Fasshauer demonstrates that conditional positive definiteness is guaranteed when Equation 2.2 exactly reproduces functions of degree less than or equal m. For RBF scattered data interpolation in one dimension, this can be achieved by adding a polynomial of order m with $M = \binom{m+1}{1}$ terms (e.g., $x^0, x^1, \cdots, x^m$). In $\mathbb{R}^d$, $M = \binom{m+d}{d}$ [40], giving

$$\begin{aligned} \sum_{j=1}^{N} c_j \phi_j(x) + \sum_{l=1}^{M} d_l P_l(x) &= f(x), \qquad P_l(x) \in \Pi_m^d \\ \left[\ \phi \quad P \ \right] \begin{pmatrix} c \\ d \end{pmatrix} &= \left(\ f \ \right) \end{aligned} \tag{2.3}$$

where the second summation (referred to as *interpolation conditions* [40]) ensures the minimum degree of the interpolant. Refer to Table 2.1 for a short list of recommended RBFs and minimally required orders of m. This document prefers the Gaussian RBF. Notice, in Equation 2.3, that the interpolation conditions add M new degrees of freedom, so we must provide M additional constraints to square the system. In this case:

$$\sum_{j=1}^{N} c_j P_l(x_j) = 0, \qquad l = 1, ..., M$$

or

$$P^T c = 0. \tag{2.4}$$

It is now possible again to write the interpolation problem as a complete linear system using Equations 2.3 and 2.4:

$$\underbrace{\begin{bmatrix} \phi & P \\ P^T & 0 \end{bmatrix}}_{A} \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix} \tag{2.5}$$

Equation 2.5—typically a dense system except in the case of RBFs with compact support—can be solved efficiently via standard methods like LU-decomposition. With the coefficients,

the interpolant can be sampled at any test points, $\{x_i\}_{i=1}^n$, by substitution into Equation 2.3:

$$\begin{aligned} f(x_i) &= \sum_{j=1}^{N} c_j \phi_j(x_i) + \sum_{l=1}^{M} d_l P_l(x_i) \\ &= \underbrace{\left[\begin{array}{cc} \phi & P \end{array}\right]}_{B} \left(\begin{array}{c} c \\ d \end{array}\right) \Bigg|_{x=x_i} \end{aligned} \tag{2.6}$$

2.2.2 Reconstructing Solutions for PDEs

In the next few subsections, we will consider collocation equations based on this general form:

$$\begin{aligned} \mathcal{L}u_\phi(x) &= f(x) \qquad \text{on } \Omega \\ \mathcal{B}u_\phi(x) &= g(x) \qquad \text{on } \Gamma \end{aligned}$$

where the methods presented below will apply the differential operators, $\mathcal{L}$ and $\mathcal{B}$, to different choices of u_ϕ and different sets of collocation points. In many applications $\mathcal{L}$ is chosen as a differential operator (e.g., $\frac{\partial}{\partial x}$, ∇, ∇^2) and $\mathcal{B} = I$ (i.e. identity operator for Dirichlet boundary conditions) for PDEs. For RBF scattered data interpolation, $\mathcal{L} = I$. There are also applications where $\mathcal{L}$ is a convolution operator (see e.g., [7, 8]) capable of smoothing/de-noising a surface reconstructed from point clouds.

For all the methods that follow a linear system is generated:

$$A_\mathcal{L} \left(\begin{array}{c} c \\ d \end{array}\right) = \left(\begin{array}{c} f \\ 0 \end{array}\right)$$

$$\left(\begin{array}{c} c \\ d \end{array}\right) = A_\mathcal{L}^{-1} \left(\begin{array}{c} f \\ 0 \end{array}\right)$$

where matrix $A_\mathcal{L}$ depends on the choice of collocation method.

Once the linear system is solved, the value $u(x)$ is reconstructed at the test points following Equation 2.6:

$$\begin{aligned} u(x) &\approx \left[\begin{array}{cc} \phi & P \end{array}\right] \left(\begin{array}{c} c \\ d \end{array}\right) \Bigg|_{x=x_i} \\ &\approx B A_\mathcal{L}^{-1} \left(\begin{array}{c} f \\ 0 \end{array}\right) \end{aligned} \tag{2.7}$$

Likewise, to obtain differential quantities we have:

$$\begin{aligned} \mathcal{L}u(x) &\approx \left[\begin{array}{cc} \phi_\mathcal{L} & P_\mathcal{L} \end{array}\right] \left(\begin{array}{c} c \\ d \end{array}\right) \Bigg|_{x=x_i} \\ &\approx B_\mathcal{L} A_\mathcal{L}^{-1} \left(\begin{array}{c} f \\ 0 \end{array}\right) \end{aligned}$$

2.2.3 PDE Methods

Now, since $u_\phi(x)$ from Equation 2.2 cannot (in general) satisfy the PDE everywhere, we enforce the PDE at a set of collocation points, which are distributed over both the interior and the boundary. Again, these points do not necessarily coincide with the RBF centers, but it is convenient for this to be true in practice. Also, for each of the methods the choice of RBF can be either global, resulting in a large dense system, or compact, resulting in a large sparse system.

Kansa's Method

The first global RBF method for PDEs, *Kansa's method* [42, 43], collocates the solution through known values on the boundary, while constraining the interpolant to satisfy the PDE operator on the interior. This is equivalent to choosing u_ϕ according to Equation 2.2. The resulting system is given by [50]; assuming that $\mathcal{L}$ is a linear operator,

$$\mathcal{L}u_\phi(x_i) = \sum_{j=1}^{N} c_j \mathcal{L}\phi_j(x_i) + \sum_{l=1}^{M} d_l \mathcal{L}P_l(x_i) \quad = \quad f(x_i) \qquad i = 1, ..., n_I \tag{2.8}$$

$$\mathcal{B}u_\phi(x_i) = \sum_{j=1}^{N} c_j \mathcal{B}\phi_j(x_i) + \sum_{l=1}^{M} d_l \mathcal{B}P_l(x_i) \quad = \quad g(x_i) \qquad i = n_I + 1, \cdots, n \tag{2.9}$$

$$\sum_{j=1}^{N} c_j P_l(x_j) \quad = \quad 0 \qquad l = 1, \cdots, M \tag{2.10}$$

where n_I are the number of interior collocation points, with the number of boundary collocation points equal to $n - n_I$. First, observe that the differential operators are applied directly to the RBFs inside summations, rather than first solving the scattered data interpolation problem and then applying the operator to the interpolant. Second, since the basis functions are known analytically, it is possible (although sometimes painful) to derive $\mathcal{L}\phi$ (refer to [19] for RBF derivative tables); the same is true for the polynomials P_l.

We can now reformulate Kansa's method as the linear system:

$$\underbrace{\begin{bmatrix} \phi_\mathcal{L} & P_\mathcal{L} \\ \phi_\mathcal{B} & P_\mathcal{B} \\ P^T & 0 \end{bmatrix}}_{A_\mathcal{L}} \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} f \\ g \\ 0 \end{pmatrix} \tag{2.11}$$

where $\phi_\mathcal{L} = \mathcal{L}\phi$, $P_\mathcal{L} = \mathcal{L}P$ are the interior components (Equation 2.8), $\phi_\mathcal{B}$ and $P_\mathcal{B}$ are the boundary components (Equation 2.9), and $P^T = \begin{bmatrix} P_\mathcal{L}^T & P_\mathcal{B}^T \end{bmatrix}$ are constraints for both interior and boundary polynomial parts (Equation 2.10). From Equation 2.11 it should be clear why Kansa's method is also known as the *Unsymmetric* collocation method.

Recall that the matrix in Equation 2.11 has no guarantee of non-singularity [18]; however, singularities are rare in practice [46].

Fasshauer's Method

Fasshauer's method [18] addresses the problem of singularity in Kansa's method by assuming the interpolation to be Hermite. That is, it requires higher differentiability of the basis functions (they must be at least C^k-continuous if $\mathcal{L}$ is of order k). Leveraging this assumption, Fasshauer's method chooses:

$$u_\phi(x_i) = \sum_{j=1}^{N_I} c_j \mathcal{L}\phi_j(x_i) + \sum_{j=N_I+1}^{N} c_j \mathcal{B}\phi_j(x_i) + \sum_{l=1}^{M} d_l P_l(x_i) \tag{2.12}$$

as the interpolant passing through collocation points. Note N_I is used here to specify the number of RBF centers in the interior of Ω. Here the interpolant is similar to Equation 2.2, but a change of basis functions is used for the expansion: $\mathcal{L}\phi_j(x)$ on the interior and $\mathcal{B}\phi_j(x)$ on the boundary.

Substituting Equation 2.12 into Equations 2.8-2.10 we get:

$$\begin{aligned}
\sum_{j=1}^{N_I} c_j \mathcal{L}^2\phi_j(x_i) + \sum_{j=N_I+1}^{N} c_j \mathcal{L}\mathcal{B}\phi_j(x_i) + \sum_{l=1}^{M} d_l \mathcal{L}P_l(x_i) &= f(x_i) \qquad i = 1, ..., n_I \\
\sum_{j=1}^{N_I} c_j \mathcal{B}\mathcal{L}\phi_j(x_i) + \sum_{j=N_I+1}^{N} c_j \mathcal{B}^2\phi_j(x_i) + \sum_{l=1}^{M} d_l \mathcal{B}P_l(x_i) &= g(x_i) \qquad i = n_I + 1, ..., n \\
\sum_{j=1}^{N_I} c_j \mathcal{L}P_l(x_j) + \sum_{j=N_I+1}^{N} c_j \mathcal{B}P_l(x_j) &= 0 \qquad l = 1, ..., M
\end{aligned} \tag{2.13}$$

which becomes the following:

$$\underbrace{\begin{bmatrix} \phi_{\mathcal{L}\mathcal{L}} & \phi_{\mathcal{L}\mathcal{B}} & P_{\mathcal{L}} \\ \phi_{\mathcal{B}\mathcal{L}} & \phi_{\mathcal{B}\mathcal{B}} & P_{\mathcal{B}} \\ P_{\mathcal{L}}^T & P_{\mathcal{B}}^T & 0 \end{bmatrix}}_{A_{\mathcal{L}}} \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} f \\ g \\ 0 \end{pmatrix} \tag{2.14}$$

Note that $\phi_{\mathcal{L}\mathcal{L}}$ represents the first summation in Equation 2.13.

The symmetry of Fasshauer's (*symmetric collocation*) method is apparent in Equation 2.14. Likewise, it is clear that the symmetric method requires more storage and computation to solve compared to Kansa's method. However, based on the assumption that collocation points coincide with RBF centers, the symmetry reduces storage requirements by half.

Direct Collocation

In *Direct collocation* (see [21, 46], the interpolant is chosen as Equation 2.2 (the same as Kansa's method). However, the Direct method collocates both the interior and boundary

operators at the boundary points:

$$\begin{aligned}
\sum_{j=1}^{N} c_j \mathcal{L}\phi_j(x_i) + \sum_{l=1}^{M} d_l \mathcal{L}P_l(x_i) &= f(x_i) \qquad i = 1, ..., n \\
\sum_{j=1}^{N} c_j \mathcal{B}\phi_j(x_i) + \sum_{l=1}^{M} d_l \mathcal{B}P_l(x_i) &= g(x_i) \qquad i = 1, ..., n_B = n - n_I \\
\sum_{j=1}^{N} c_j P_l(x_j) &= 0 \qquad\quad l = 1, ..., M
\end{aligned} \tag{2.15}$$

Reformulating as a linear system we get:

$$\begin{bmatrix} \phi_{\mathcal{L}} & P_{\mathcal{L}} \\ \phi_{\mathcal{B}} & P_{\mathcal{B}} \\ P^T & 0 \end{bmatrix} \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} f \\ g \\ 0 \end{pmatrix} \tag{2.16}$$

While the final system in Equation 2.16 is structured the same as Kansa's method (Equation 2.11), careful inspection of the index i in Equations 2.8 and 2.15 reveals that Direct collocation produces a larger system.

RBF-PS

The extension of global collocation to traditional pseudo-spectral form was introduced by Fasshauer in [17]. Dubbed RBF-PS, the method utilizes the same logic from Kansa's and Fasshauer's collocation methods to form matrix $A_{\mathcal{L}}$ (i.e., $A_{\mathcal{L}}$ can be either Equation 2.11 or 2.14). However, RBF-PS subtly assumes the solution, $u(x)$, is only required at collocation points (i.e., $\{x_i\} = \{x_c\}$) [17, 19]. Then, extending Equation 2.7, RBF-PS gives:

$$\begin{aligned}
u(x) &= \left(B A_{\mathcal{L}}^{-1}\right) \begin{pmatrix} f \\ 0 \end{pmatrix} \\
&= D_{\mathcal{L}}^T \begin{pmatrix} f \\ 0 \end{pmatrix}.
\end{aligned} \tag{2.17}$$

where $D_{\mathcal{L}}$ is a discrete differentiation matrix (DM) for the operator $\mathcal{L}$. Here, $D_{\mathcal{L}}$ is independent of the function $f(x)$ and is assembled by solving the system:

$$D_{\mathcal{L}} = A_{\mathcal{L}}^{-T} B^T \tag{2.18}$$

An LU-decomposition ($O(N^3)$) in preprocessing with forward- and back-solves ($O(N^2)$) are fitting to efficiently solve the multiple RHS system[19, 75].

Since matrix $D_{\mathcal{L}}$ is independent of functions $u(x)$ and $f(x)$, the matrix requires update only if the RBF centers move—a compelling benefit for time-dependent problems on stationary nodes. The complexity of RBF-PS for time-dependent solutions is then reduced to a matrix-vector multiply ($O(N^2)$) for each time-step. In contrast, classic RBF collocation methods also construct LU factors of $A_{\mathcal{L}}^{-1}$ in preprocessing, but delay application of forward- and back-solves to acquire time-dependent weighting coefficients at each time-step. This is then followed by the pre-multiply of B (i.e., additional $O(N^2)$) to complete the time-step.

2.2.4 Local Methods

Another trend in RBF methods is to use compact support to produce local linear systems defined at each collocation point. Examples of this include [67, 68] for Kansa's method, [62, 63, 64] for Fasshauer's method. To our knowledge no one has considered local Direct collocation. Also, instead of specifying a cut-off radius for RBF support, some authors specify the exact stencil size (i.e., number of neighboring points to include); see e.g., [15, 62].

After observing the general structure of the symmetric and unsymmetric collocation methods above, it is necessary only to present the symmetric (i.e. Fasshauer's) local method and note that in the unsymmetric case certain blocks will be zero allowing the system to shrink.

The formula for the interpolant local to the (k)-th collocation point (i.e., RBF center) is given by:

$$u_\phi^{(k)}(x_i) = \sum_{j(k)=1}^{N_I} c_j^{(k)} \mathcal{L}\phi_j(x_i) + \sum_{j(k)=N_I+1}^{N_S} c_j^{(k)} \mathcal{B}\phi_j(x_i) + \sum_{l=1}^{M} d_l^{(k)} P_l(x_i)$$

where N_S represents the number of points that defines the local stencil; N is possibly a function of the cut-off radius in the RBF, N_I is the number of interior stencil points (those points of the stencil that lie in the interior of Ω). The index j is a function of the stencil center k allowing the system to include a local neighborhood of stencil points.

This results in a linear system with similar structure to the global collocation problem, but the dimensions are much smaller:

$$\underbrace{\begin{bmatrix} \phi_{\mathcal{L}\mathcal{L}} & \phi_{\mathcal{L}\mathcal{B}} & P_{\mathcal{L}} \\ \phi_{\mathcal{B}\mathcal{L}} & \phi_{\mathcal{B}\mathcal{B}} & P_{\mathcal{B}} \\ P_{\mathcal{L}}^T & P_{\mathcal{B}}^T & 0 \end{bmatrix}}_{A_{\mathcal{L}}} \begin{pmatrix} c^{(k)} \\ d^{(k)} \end{pmatrix} = \begin{pmatrix} f \\ g \\ 0 \end{pmatrix} \tag{2.19}$$

Solving this system gives an interpolant locally defined around the stencil center. Note that approximating the PDE solution $u(x)$ requires finding the stencil center nearest x, then using the local interpolant for that stencil. Since interpolation is local (i.e., $c_j^{(k)}$'s are unique to each RBF center), reconstructing the derivatives with Equation 2.8 is limited to an inner product for each center rather than the matrix-vector grouping possible with global RBFs. This approach decomposes the problem into smaller and more manageable parts. However, because the interpolants are local, there is no notion of global continuity/smoothness of the solution.

2.3 Recent Advances in Conditioning

The most limiting factor in the success of RBF methods has not been the complexity of the methods, nor the task of approximating derivatives. Rather, it is the support parameter, ϵ, and the dilemma one faces in the *Uncertainty Relation* [56]. Recall that as $\epsilon \to 0$, ill-conditioning of the RBF interpolation matrices increases, but so too does the approximation accuracy—that is, assuming a stable solution can be found. Likewise, as the number of collocation points increases, the range of ϵ for which the linear system has acceptable

conditioning narrows. In [30], the authors observe that much of the literature on RBF methods seek to find optimal values of the support parameter ϵ for the highest accuracy in applications. Occasionally the optimal values lie within a range of acceptable conditioning to solve the linear systems directly (a.k.a. RBF-Direct solutions). More often, one must compromise between the accuracy loss for large ϵ and accuracy loss in RBF-Direct solutions due to lower values of ϵ. Many attempts to express the optimal ϵ as a function of problem size have also been thwarted as the impact on the optimal ϵ values in the face of small node perturbations is still not fully understood. This makes refinements a challenge to manage.

In an effort to overcome limitations due to conditioning, Fornberg and Wright [29] presented the *Contour–Padé* algorithm, which allows for numerically stable computation of highly accurate interpolants with nearly flat RBFs (i.e., $\epsilon \to 0$). Larsson and Fornberg [46] applied the algorithm to all three methods of collocation (Kansa's, Fasshauer's and Direct Collocation) with considerable gain in accuracy over solutions from classical second-order FD and a pseudospectral method. The Contour-Padé algorithm is not overly competitive due to the fact that it only supports fewer than a hundred in 2D and slightly more in 3D [32].

The *RBF-QR* method, was later introduced by Fornberg and Piret [33] in context of a sphere to let $\epsilon \to 0$ for a few thousand nodes. It was later extended to general 1D, 2D and 3D problems in [30]. The RBF-QR method uses a truncated expansion of RBFs in terms of spherical harmonics or Chebyshev polynomials and leverages QR factorization to create create a new well-conditioned set of basis functions to reproduce the original RBF space. The well-conditioned basis set allows stable solution independent of the value ϵ. The cost of the method is demonstrated to increase as ϵ increases. Benchmarks in [30] show that double precision RBF-QR is between 3x-7x slower than RBF-Direct for the same values of ϵ. Fornberg, Larsson and Flyer [30] successfully implemented the 2D method in less than 100 lines of MATLAB code and apply RBF-QR to problems with 6000 quasi-unifrom nodes and globally supported RBFs.

Between Contour-Padé and RBF-QR, global RBF methods overcame many conditioning issues for small to mid-sized problems. The lack of support for large problem sizes is discouraging, but it leads to an argument in favor of local methods like RBF-FD, which decrease the problem size to fit nicely within the scope of stable methods. To our knowledge, no application of a local method has required more than a few hundred nodes per local solution.

Most recently, Fornberg et al. [32] introduced a new method called RBF-GA, which performs a similar change of basis as RBF-QR, but the method avoids truncated infinite expansions by expressing the new basis functions in terms of an incomplete Gamma function. Unlike RBF-QR, this method is limited to Gaussian basis functions only. Interestingly, benchmarks provided in [32] rank stable methods for RBFs from fastest to slowest as: RBF-Direct, RBF-GA, and then RBF-QR. Similar to the other methods, RBF-GA is effective for a small number of nodes: a few hundred in 2D, and at least 500 in 3D. Unlike RBF-QR, which performs a change of basis on the interpolating matrix only, the RBF-GA method additionally requires a complicated change of basis for the RHS.

Part II

RBF-FD for HPC Environments

Chapter 3

Introduction to RBF-FD

RBF-generated Finite Differences (RBF-FD) were first introduced by Tolstykh in 2000 [66], but it was the simultaneous, yet independent, efforts in [59], [65], [74] and [9] that gave the method its real start. The RBF-FD method (and the RBF-HFD, "Hermite" equivalent [76]) is similar in concept to classical finite-differences (FD), but differs in that the underlying differentiation weights are exact for RBFs rather than polynomials. The method contrasts with global RBF methods in the sense that it does not collocate the PDE. Instead, RBF-FD provides a set of generalized FD weights representing the discrete differential operator for a small neighborhood of nodes.

RBF-FD share many advantages with global RBF methods, like the ability to function without an underlying mesh, easily extend to higher dimensions and afford large time steps; however spectral accuracy is lost. Other advantages of RBF-FD include lower computational complexity together with high-order accuracy (6th to 10th order accuracy is common). As in FD, increasing the stencil size, n, increases the order accuracy of the approximation. While not a panacea for PDEs, the method is simple to code, easily extensible to higher dimensions, and powerful in its ability to avoid singularities introduced by the coordinate systems that might impact other methods (see e.g., [25, 31]).

In some ways, RBF-FD and global RBF methods are plagued by the same difficulties. For example, as the number of nodes in the stencil increases, so too does the ill-conditioning of the linear systems to be inverted. Similarly, the most accurate weights occur when $\epsilon \to 0$, but values in that regime beget additional ill-conditioning problems—a recurrence of the *Uncertainty Relation* [56]. One key difference in the multiple independent RBF-FD origins was that Wright [74] focused on bypassing ill-conditioning of RBF-FD and investigated its behavior in the limit as $\epsilon \to 0$ by means of the Contour-Padé algorithm.

Given N total nodes in the domain, N linear systems, each of size $(n+1) \times (n+1)$, are solved to calculate the differentiation weights for derivatives at each node. With $n \ll N$, the RBF-FD preprocessing complexity can behave as $O(N)$; significantly lower than the global RBF or RBF-PS methods ($O(N^3)$). Additionally, the cost per time step is also dominated by $O(N)$.

RBF-FD have been successfully employed for a variety of problems including Hamilton-Jacobi equations [9], convection-diffusion problems [10, 62], incompressible Navier-Stokes equations [11, 59], transport on the sphere [31], and the shallow water equations [24]. Shu et al. [60] compared the RBF-FD method to Least Squares FD (LSFD) in context of 2D

incompressible viscous cavity flow, and found that under similar conditions, the RBF-FD method was more accurate than LSFD, but the solution required more iterations of an iterative solver. RBF-FD was applied to Poisson's equation in [73]. Chandhini and Sanyasiraju [10] studied it in context of 1D and 2D, linear and non-linear, convection-diffusion equations, demonstrating solutions that are non-oscillatory for high Reynolds number, with improved accuracy over classical FD. An application to Hamilton-Jacobi problems [9], and 2D linear and non-linear PDEs including Navier-Stokes equations [59] have all been considered.

The RBF-FD method is similar to classical Finite Differences in that RBF-FD allows derivatives of a function $u(x)$ to be approximated by weighted combinations of n function values in a small neighborhood (i.e., $n \ll N$) around a *center* node, x_c. That is:

$$\mathcal{L}u(x) \mid_{x=x_c} \approx \sum_{j=1}^{n} c_j u(x_j) \tag{3.1}$$

where $\mathcal{L}u$ again represents a differential quantity over $u(x)$ (e.g., $\mathcal{L} = \frac{\partial}{\partial x}$). We refer to the n nodes around x_c as a *stencil* with size n. While not required, in practice one considers stencils to include the center, x_c, plus the $n-1$ nearest neighboring nodes. The definition of "nearest" depends the choice of distance metric; here, Euclidean distance ($||x - x_c||_2$) is preferred.

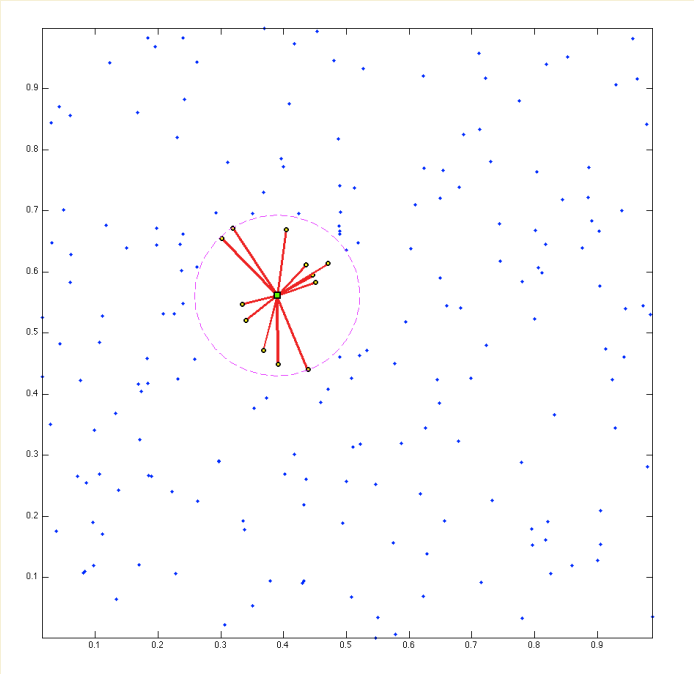
Typically, one needs derivatives at every node in the discretized domain to solve PDEs. To achieve this with RBF-FD, stencils are generated around each node in the domain. Stencils need not have the same size (n), but this is assumed here for simplicity in discussion. Furthermore, the number of stencils need not match the number of nodes in the domain, but this is also assumed.

Figure 3.1 provides two examples of RBF-FD stencils. First, Figure 3.1a illustrates a single stencil of size $n = 13$ in a domain of randomly distributed nodes. The stencil center, x_c, is represented by a green square, with the 12 neighboring nodes connected via red edges. The purple circle, the minimum covering circle for the stencil, demonstrates that the stencil contains only the 12 nearest neighbors of the center node. In Figure 3.1b, a larger RBF-FD stencil of size $n = 75$ on the unit sphere is shown as red and blue disks surrounding the center represented as a square. Green disks are nodes outside of the stencil. The radii and color of the red and blue disks represent the magnitude and alternating sign of coefficients, c_j, determined to calculate a derivative quantity at the stencil center.

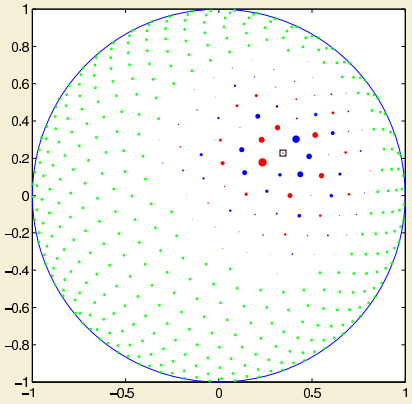
3.0.1 Stencil Weights

To approximate $\mathcal{L}u(x)$, one requires the stencil *weights* (coefficients), c_j. Stencil weights are a discrete representation of the differential operator at the stencil center and may vary by node location (e.g., nodes at the boundary are usually governed by another operator, $\mathcal{B}$). Weights are obtained by enforcing that they be exact within the space spanned by the RBFs centered at stencil nodes (i.e., $\phi_j(x) = \phi(\epsilon||x - x_j||_2)$; an RBF centered at x_j). Various studies [24, 27, 31, 76] show that better accuracy is achieved when the interpolant can exactly reproduce a constant, p_0, such that

$$\mathcal{L}\phi_i(x) \mid_{x=x_c} = \sum_{j=1}^{n} c_j \phi_j(x_i) + c_{n+1} p_0 \qquad \text{for } i = 1, 2, ..., n$$



(a) A 13 node RBF-FD stencil of randomly distributed nodes. The stencil centered at the green square contains the 12 nearest neighbors contained within the minimum covering circle drawn in purple.



(b) A 75 node RBF-FD stencil with blue (negative) and red (positive) differentiation weights to approximate advective operator at the square. Stencils weights indicated by scale of disk radii. (Image courtesy of Bengt Fornberg and Natasha Flyer)

Figure 3.1: Examples of stencils computable with RBF-FD

with $\mathcal{L}\phi_i$ provided by analytically applying the differential operator to the RBF. Assuming $p_0 = 1$, the constraint $\sum_{i=1}^{n} c_i = \mathcal{L}p_0|_{x=x_c} = 0$ completes the system:

$$\begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_n(x_1) & 1 \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_n(x_2) & 1 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ \phi_1(x_n) & \phi_2(x_n) & \cdots & \phi_n(x_n) & 1 \\ 1 & 1 & \cdots & 1 & 0 \end{bmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \\ c_{n+1} \end{pmatrix} = \begin{pmatrix} \mathcal{L}\phi_1(x)\,|_{x=x_c} \\ \mathcal{L}\phi_2(x)\,|_{x=x_c} \\ \vdots \\ \mathcal{L}\phi_n(x)\,|_{x=x_c} \\ 0 \end{pmatrix} \tag{3.2}$$

$$\begin{bmatrix} \phi & P \\ P^T & 0 \end{bmatrix} \begin{pmatrix} c_{\mathcal{L}} \\ d_{\mathcal{L}} \end{pmatrix} = \begin{pmatrix} \phi_{\mathcal{L}} \\ 0 \end{pmatrix}.$$

The choice of $\mathcal{L}$ can be any linear operator. As an example, if $\mathcal{L}$ is the identity operator, then the above procedure leads to RBF-FD weights for interpolation. If $\mathcal{L} = \frac{\partial}{\partial x}$, one obtains the weights to approximate the first derivative in x. Refer to [19] for a table of commonly used RBF derivatives. Section 3.0.4 provides a list of derivatives used in this work.

The small $(n+1) \times (n+1)$ system in Equation 3.2 is dense, and is solved at a cost of $O(n^3)$ floating point operations (FLOPs) using direct methods like LU-decomposition. The resulting stencil weights, $c_{\mathcal{L}} = \{c_j\}_{j=1}^{n}$ can be substituted into Equation 3.1 for the derivative approximation at x_c. Coefficient c_{n+1} ($d_{\mathcal{L}} = c_{n+1}$), included in the solution of Equation 3.2, is of no use and discarded once the system has been solved.

Based on the choice of support parameter, ϵ, the Equation 3.2 may suffer problems with conditioning. In such cases, stable methods like Contour–Padé [74] or RBF-QR [14, 30] may be preferred.

3.0.2 Differentiation Matrix

Note that Equation 3.2 resolves the weights only for the stencil x_c. The small system solve is repeated N times—once for each stencil—to obtain a total of $N \times n$ stencil weights.

For PDEs, it is common practice to assemble a *differentiation matrix* (DM); a discrete representation of the PDE operator on the domain. Given the set of nodes in the domain $\{x_k\}_{k=1}^{N}$, the c-th row of the DM represents the discrete PDE operator for the stencil centered at node x_c with stencil nodes $\{x_j\}_{j=1}^{n}$:

$$\mathcal{L}u(x) \approx D_{\mathcal{L}}u$$
$$D_{\mathcal{L}}^{(c,k)} = \begin{cases} c_j & x_k = x_j \\ 0 & x_k \neq x_j \end{cases}$$

where (c, k) represents the (row, column) index of $D_{\mathcal{L}}$ and vector $u = \{u(x_k)\}_{k=1}^{N}$. Equation 3.1 can be rewritten as:

$$\mathcal{L}u(x) \mid_{x=x_c} \approx D_{\mathcal{L}}^{(c)}u \quad .$$

In the solution of PDEs the DMs are utilized in explicit and implicit modes. Here explicit implies evaluating the matrix-vector mulitply to get derivative values, u', from explicitly known vector of solution values u:

$$u' = D_{\mathcal{L}}u \tag{3.3}$$

whereas implicit solves for unknown u:

$$D_{\mathcal{L}}u = f \tag{3.4}$$

An example RBF-FD DM is illustrated in Figure 3.2. In this example, assume operator $\mathcal{L} = \frac{\partial}{\partial x}$ is approximated at all N stencil centers of an arbitrary domain. RBF-FD weights assemble the rows of the differentiation matrix, D_x. On each row, weights are indicated by blue dots. The sparsity of rows reflects the subset of $\{x_k\}_{k=1}^{N}$ included in corresponding stencils of size n. On the right hand side, discrete derivative values $\frac{du}{dx}$ are approximated at all stencil centers.

Differentiation matrices are assembled at a cost of $O(n^3 N)$ FLOPs. However, since the goal of RBF-FD is to keep stencil neighborhoods small ($n \ll N$), the cost of assembly scales as $O(N)$. Furthermore, RBF-FD weights are independent of function values ($u(x)$) and rely only on stencil node locations. The implications of this are as profound as in the context RBF-PS for time-dependent PDEs: the stencil weights are constant so long as the nodes are stationary. Thus, the DM assembly is part of a one-time preprocessing step.

The sparsity exhibited by the DM in Figure 3.2 is typical for RBF-FD due to $n \ll N$. Best practices dictate that the DMs be stored in a compressed sparse storage format to retain only non-zeros and their corresponding indices in memory.

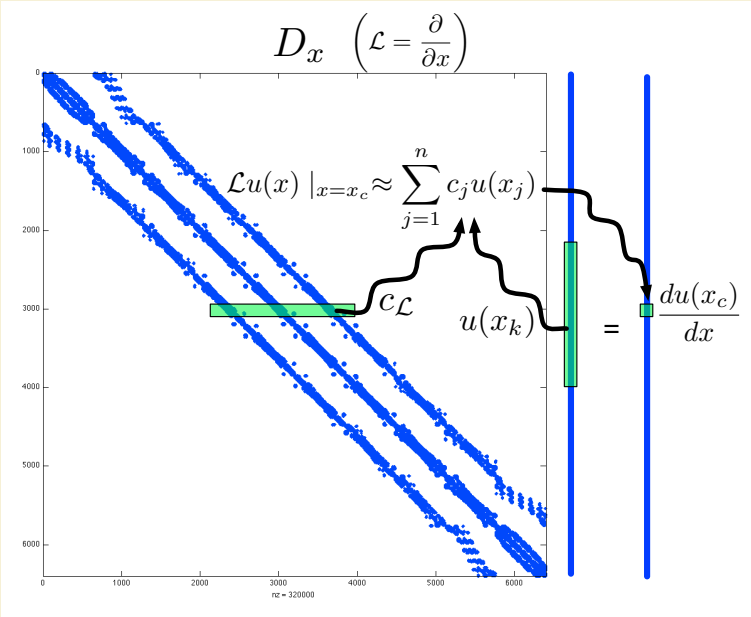


Figure 3.2: Differentiation matrix D_x is applied to the solution values $u(x)$ to obtain derivative approximations, $\frac{du}{dx}$.

3.0.3 Multiple Operators

In many cases, multiple derivatives (e.g., $\mathcal{L} = \nabla^2$, $\frac{\partial}{\partial x}$, $\frac{\partial}{\partial y}$, etc.) are required at stencil centers. This is common, for example, when solving coupled PDEs. For RBF-FD, acquiring weights for each additional operator can be both straight-forward and computationally efficient. For each change of differential operator, observe that only the RHS of Equation 3.2 is modified. Thus, new operators amount to extending Equation 3.2 to solve

$$\begin{bmatrix} \phi & P \\ P^T & 0 \end{bmatrix} \begin{bmatrix} c_{\nabla^2} & c_x & c_y & \cdots \\ d_{\nabla^2} & d_x & d_y & \ldots \end{bmatrix} = \begin{bmatrix} \phi_{\nabla^2} & \phi_x & \phi_y & \cdots \\ 0 & 0 & 0 & \cdots \end{bmatrix}.$$

where multiple sets of weights (c_∇, c_x, c_y) are obtained simultaneously. This dense, symmetric, multiple RHS linear system is considered ideal by linear algebra packages, and many highly optimized routines exist to solve them (e.g., LAPACK "dgesv") [2].

3.0.4 Weight Operators

In the course of this work we work with a variety of PDEs. This section enumerates a list of relevant operators and their corresponding equations for the RHS of Equation 3.2. Whenever possible the general form of $\mathcal{L}\phi$ is provided; otherwise the Gaussian RBF ($\phi(r) = e^{-(\epsilon r)^2}$) is assumed.

First and Second Derivatives ($\frac{1}{r}\frac{\partial\phi}{\partial r}, \frac{\partial^2\phi}{\partial r^2}$)

The following are used in subsequent derviatives:

$$\frac{1}{r}\frac{d}{dr}\phi(r) = -2\epsilon^2\phi(r)$$
$$\frac{\partial^2\phi}{\partial r^2} = \epsilon^2(-2 + 4(\epsilon r)^2)\phi(r)$$

Cartesian Gradient (∇)

The first derivatives in Cartesian coordinates ($\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}$) are produced via the chain rule:

$$\frac{\partial\phi}{\partial x} = \frac{\partial r}{\partial x}\frac{\partial\phi}{\partial r} = \frac{(x - x_j)}{r}\frac{\partial\phi}{\partial r}$$
$$\frac{\partial\phi}{\partial y} = \frac{\partial r}{\partial y}\frac{\partial\phi}{\partial r} = \frac{(y - y_j)}{r}\frac{\partial\phi}{\partial r}$$
$$\frac{\partial\phi}{\partial z} = \frac{\partial r}{\partial z}\frac{\partial\phi}{\partial r} = \frac{(z - z_j)}{r}\frac{\partial\phi}{\partial r}$$

where $\frac{\partial\phi}{\partial r}$ for the Gaussian RBFs is given above.

Cartesian Laplacian (∇^2)

Fasshauer [19] provides the general form of ∇^2 in 2D as:

$$\nabla^2 = \frac{\partial^2}{\partial r^2}\phi(r) + \frac{1}{r}\frac{\partial}{\partial r}\phi(r)$$

For Gaussian RBFs in particular we have the following operators:

- 1D:
$$\nabla^2 = \epsilon^2(-2 + 4(\epsilon r)^2)\phi(r)$$
- 2D:
$$\nabla^2 = \epsilon^2(-4 + 4(\epsilon r)^2)\phi(r)$$
- 3D:
$$\nabla^2 = \epsilon^2(-6 + 4(\epsilon r)^2)\phi(r)$$

which all fit $\nabla^2 = \frac{\partial^2}{\partial r^2}\phi(r) + \frac{d-1}{r}\frac{\partial}{\partial r}\phi(r)$ for dimension d.

Laplace-Beltrami (Δ_S) on the Sphere

The ∇^2 operator can be represented in spherical polar coordinates for $\mathbb{R}^3$ as:

$$\nabla^2 = \underbrace{\frac{1}{r}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}\right)}_{\text{radial}} + \underbrace{\frac{1}{r^2}\Delta_S}_{\text{angular}},$$

where Δ_S is the Laplace-Beltrami operator—i.e., the Laplacian operator constrained to the surface of the sphere. This form nicely illustrates the separation of components into radial and angular terms.

In the case of PDEs solved on the unit sphere, there is no radial term, so we have:

$$\nabla^2 \equiv \Delta_S. \tag{3.5}$$

Although this originated in the spherical coordinate system, [75] introduced the following Laplace-Beltrami operator for the surface of the sphere:

$$\Delta_S = \frac{1}{4}\left[\left(4 - r^2\right)\frac{\partial^2 \phi}{\partial r^2} + \frac{4 - 3r^2}{r}\frac{\partial \phi}{\partial r}\right],$$

where r is the Euclidean distance between nodes of an RBF-FD stencil and is independent of our choice of coordinate system.

Constrained Gradient ($P_x \cdot \nabla$) on the Sphere

Following [24, 26], the gradient operator can be constrained to the sphere with this projection matrix:

$$P = I - \mathbf{x}\mathbf{x}^T = \begin{pmatrix} (1 - x_1^2) & -x_1 x_2 & -x_1 x_3 \\ -x_1 x_2 & (1 - x_2^2) & -x_2 x_3 \\ -x_1 x_3 & -x_2 x_3 & (1 - x_3^2) \end{pmatrix} = \begin{pmatrix} P_{x_1} \\ P_{x_2} \\ P_{x_3} \end{pmatrix} \tag{3.6}$$

where $\mathbf{x}$ is the unit normal at the stencil center.

The direct method of computing RBF-FD weights for the projected gradient for $\mathbf{P} \cdot \nabla$ is presented in [26]. When solving for the weights, we apply the projection on the right hand side of our small linear system. We let $\mathbf{x} = \left(x_1, x_2, x_3\right)$ be the stencil center, and $\mathbf{x}_k = \left(x_{1,k}, x_{2,k}, x_{3,k}\right)$ indicate an RBF-FD stencil node.

Using the chain rule, and assumption that

$$r(\mathbf{x}_k - \mathbf{x}) = ||\mathbf{x}_k - \mathbf{x}|| = \sqrt{(x_{1,k} - x_1)^2 + (x_{2,k} - x_2)^2 + (x_{3,k} - x_3)^2},$$

we obtain the unprojected gradient of ϕ as

$$\nabla \phi(r(\mathbf{x}_k - \mathbf{x})) = \frac{\partial r}{\partial \mathbf{x}}\frac{\partial \phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r} = -(\mathbf{x}_k - \mathbf{x})\frac{1}{r(\mathbf{x}_k - \mathbf{x})}\frac{\partial \phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r}.$$

Applying the projection matrix gives

$$\begin{aligned}
\mathbf{P}\nabla \phi(r(\mathbf{x}_k - \mathbf{x})) &= -(\mathbf{P} \cdot \mathbf{x}_k - \mathbf{P} \cdot \mathbf{x})\frac{1}{r(\mathbf{x}_k - \mathbf{x})}\frac{\partial \phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r} \\
&= -(\mathbf{P} \cdot \mathbf{x}_k - 0)\frac{1}{r(\mathbf{x}_k - \mathbf{x})}\frac{\partial \phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r} \\
&= -(I - \mathbf{x}\mathbf{x}^T)(\mathbf{x}_k)\frac{1}{r(\mathbf{x}_k - \mathbf{x})}\frac{\partial \phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r} \\
&= \begin{pmatrix} x\mathbf{x}^T\mathbf{x}_k - x_k \\ y\mathbf{x}^T\mathbf{x}_k - y_k \\ z\mathbf{x}^T\mathbf{x}_k - z_k \end{pmatrix}\frac{1}{r(\mathbf{x}_k - \mathbf{x})}\frac{\partial \phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r}
\end{aligned}$$

Thus, we directly compute the weights for $P_x \cdot \boldsymbol{\nabla}$ using these three RHS in Equation 3.2:

$$P\frac{\partial}{\partial x_1} = (x_1\mathbf{x}^T\mathbf{x}_k - x_{1,k})\frac{1}{r(\mathbf{x}_k - \mathbf{x})}\frac{\partial\phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r}|_{\mathbf{x}=\mathbf{x}_j}$$
$$P\frac{\partial}{\partial x_2} = (x_2\mathbf{x}^T\mathbf{x}_k - x_{2,k})\frac{1}{r(\mathbf{x}_k - \mathbf{x})}\frac{\partial\phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r}|_{\mathbf{x}=\mathbf{x}_j}$$
$$P\frac{\partial}{\partial x_3} = (x_3\mathbf{x}^T\mathbf{x}_k - x_{3,k})\frac{1}{r(\mathbf{x}_k - \mathbf{x})}\frac{\partial\phi(r(\mathbf{x}_k - \mathbf{x}))}{\partial r}|_{\mathbf{x}=\mathbf{x}_j}$$

Hyperviscosity Δ^k for Stabilization

When explicitly solving hyperbolic equations, differentiation matrices encode convective operators of the form

$$D = \alpha\frac{\partial}{\partial\lambda} + \beta\frac{\partial}{\partial\theta} \tag{3.7}$$

The convective operator, discretized through RBF-FD, has eigenvalues in the right half-plane causing the method to be unstable [24, 31]. Stabilization of the RBF-FD method is achieved through the application of a hyperviscosity filter to Equation (3.7) [31]. By using Gaussian RBFs, $\phi(r) = e^{-(\epsilon r)^2}$, the hyperviscosity (a high order Laplacian operator) simplifies to

$$\Delta^k\phi(r) = \epsilon^{2k}p_k(r)\phi(r) \tag{3.8}$$

where k is the order of the Laplacian and $p_k(r)$ are multiples of generalized Laguerre polynomials that are generated recursively ([31]):

$$\begin{cases} p_0(r) & = 1, \\ p_1(r) & = 4(\epsilon r)^2 - 2d, \\ p_k(r) & = 4((\epsilon r)^2 - 2(k-1) - \frac{d}{2})p_{k-1}(r) - 8(k-1)(2(k-1) - 2 + d)p_{k-2}(r), \qquad k = 2, 3, ... \end{cases}$$

where d is the dimension of the problem. The application of hyperviscosity in Chapter **??**, utilizes the operator as a filter to shift eigenvalues and stabilize advection equations on the surface of the unit sphere. In that case, $d = 2$ is assumed ksince individual RBF-FD stencils can be viewed as (nearly) lying on a plane. For small N, the diameter of the stencil may not be sufficiently small compared to the radius of the sphere, and hyperviscosity might not work as advertised.

In the case of parabolic and hyperbolic PDEs, hyperviscosity is added as a filter to the right hand side of the evaluation. For example, at the continuous level, the equation solved takes the form

$$\frac{\partial u}{\partial t} = -\mathcal{L}u + Hu, \tag{3.9}$$

where $\mathcal{L}$ is the PDE operator, and H is the hyperviscosity filter operator. Applying hyperviscosity shifts all the eigenvalues of L (the discrete form of $\mathcal{L}$) to the left half of the complex plane. This shift is controlled by k, the order of the Laplacian, and a scaling parameter γ_c, defined by

$$H = \gamma\Delta^k = \gamma_c N^{-k}\Delta^k.$$

It was found in [24], and verified in our own application, that $\gamma = \gamma_c N^{-k}$ provides stability and good accuracy for all values of N considered here. It also ensures that the viscosity vanishes as $N \to \infty$ [24]. In general, the larger the stencil size, the higher the order of the Laplacian. This is attributed to the fact that, for convective operators, larger stencils treat a wider range of modes accurately. As a result, the hyperviscosity operator should preserve as much of that range as possible. The parameter γ_c must also be chosen with care and its sign depends on k (for k even, γ_c will be negative and for k odd, it will be positive). If γ_c is too large, the eigenvalues move outside the stability domain of our time-stepping scheme and/or eigenvalues corresponding to lower physical modes are not left intact, reducing the accuracy of our approximation. If γ_c is too small, eigenvalues remain in the right half-plane [24, 31].

3.1 Implementation

This section provides an overview of how one implements RBF-FD. Consider Algorithm 3.1. The algorithm is partitioned into two phases: preprocessing and application. The complexity of each phase depends on the choice of algorithms/method utilized for each task.

Preprocessing encompasses tasks such as grid setup/generation, stencil generation and stencil weight calculations. As output from these tasks one expects one or more DMs representing the discrete differential operators. Assuming the grid nodes do not move, the DMs remain constant for the duration of the second phase. Additionally, DMs can be loaded from disk on subsequent runs to effectively bypass the cost of all preprocessing.

Algorithm 3.1 A High-Level View of RBF-FD

Preprocessing:

$\{x\}_{j=1}^{N}$ = GenerateGrid()
for $j = 1$ to N **do**
  Stencil $\{S_{j,i}\}_{i=1}^{n}$ = QueryNeighbors(x_j)
end for
for $j = 1$ to N **do**
  $\{w_{j,i}\}_{i=1}^{n}$ = SolveForWeights($\{S_j\}$)
  $D_{\mathcal{L}}^{(j)}$ = AssembleDM($\{w_j\}$)
end for

Application:

$t = t_{min}$
while $t < t_{max}$ **do**
  $\{u'\}$ = SolvePDE($D_{\mathcal{L}}$, $\{u\}$)
  $\{u\}$ = UpdateSolution($\{u\}$, $\{u'\}$, Δt)
  t += Δt
end while

The constructed DMs are applied in the Application phase to solve a PDE either explicitly or implicitly. Note that regardless of the explicit or implicit method, this phase reduces to a Sparse Matrix-Vector Multiply (SpMV) within the SolvePDE step. For explicit solutions, SolvePDE computes the SpMV in Equation 3.3. In the implicit case, SolvePDE must

invert the DM in Equation 3.4 via a direct or iterative linear system solve. In the case of time-dependent PDEs, RBF-FD is applied the same as any other finite difference method within a time-stepping scheme (represented by UpdateSolution).

Tuning the performance of RBF-FD requires one to focus on the Application phase. This is especially true for time-dependent PDEs where an increase in grid size results in proportional increase in the number time-steps. The recurring cost of computing an SpMV each time-step quickly amortizes the one-time cost of preprocessing.

On the other hand, preprocessing tasks have more impact on accuracy, stability and conditioning of the method. As each of these properties improve, the overall time to solution can decrease thanks to larger stable time-steps, smaller required grid size, and faster convergence (i.e., fewer iterations) within iterative linear system solves.

Here we discuss various design decisions in implementing the preprocessing tasks for RBF-FD and consider potential impacts on performance (if any). Chapter **??** will discuss Application and SpMV performance in more detail.

3.1.1 Grid

An implementation of RBF-FD begins with the grid. The method has no requirement for structured grids, or for a well-refined mesh/lattice that limits connectivity between nodes. It operates both on structured grids and random point clouds; although, the choice of grid does impact the accuracy of the method. This freedom to function on domains of any shape, dimension, and granularity is a major selling point for RBF-FD and often impossible for many other PDE methods.

The choice of grid is only relevant to the extent that it impacts the connectivity between nodes in stencils. Connectivity translates to non-zeros in rows of the DMs, so the various grid distributions result in a number of sparsity patterns. For all other intents and purposes, the choice of grid is only pertinent to ensure consistency with the PDEs to solve.

In the course of this work, we applied RBF-FD to a number of PDEs and grid distributions. Here we focus on the subset utilized in Chapters **??** and **??**. This subset is chosen for two reasons: a) to easily construct refinements in 2D/3D for consistent benchmarking and convergence studies, and b) to verify our solutions against existing methods.

Regular Grid

For basic debugging and benchmarking purposes the most natural choice is to start with a regular or Cartesian grid. Equally spaced nodes in multiple dimensions are simple to generate. Additionally, refinements—for convergence tests—are direct subsamples.

In theory, RBF-FD functions the same whether nodes are uniformly spaced or random. However, regular grids do not fully exercise advantages that RBF-FD has over other methods with its ability to operate on scattered nodes.

Maximum Determinant Nodes

In Chapter **??** we verify our RBF-FD implementation by solving PDEs on the unit sphere. For consistency with respect to related investigations (e.g., [28, 31, 33]), we choose the Maximum Determinant (MD) node sets [61, 72].

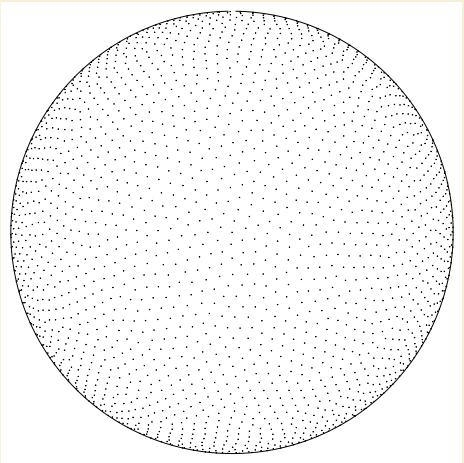


Figure 3.3: Example of $N = 4096$ maximum determinant (MD) node sets on the unit sphere.

MD node sets were introduced to the RBF community due to their success in spherical harmonics interpolation, where the seemingly irregular node distributions gain an order of magnitude accuracy compared to regular looking (Minimum Energy) node distributions [72]. RBF interpolation tends to reproduce spherical harmonics interpolants on the sphere when $\epsilon \to 0$, and RBF methods have been shown to benefit similarly from a subtle irregularity in node locations [33].

The MD node files are available for download on the authors' web site (`http://web.maths.unsw.edu.au/~rsw/Sphere`), and range in size from $N = 4$ up to N=27,556 nodes on the sphere. Figure 3.3 plots the $N = 4096$ node set to illustrate the irregularity in distribution. Node sets greater than N=27,556 are not available. Unlike regular grids, each MD node set is a refinement of the sphere, but not a subdivision, so extending beyond N=27,556 nodes would require complete regeneration.

Centroidal Voronoi Tessellations

While the MD nodes suffice for verification against related work, our objective is to scale RBF-FD to problem sizes never attempted. To this end, we sought approximately regular grids on the sphere on the order of hundreds of thousands and even millions of nodes. To this end we leverage Spherical Centroidal Voronoi Tessellations (SCVTs) to generate approximately regular node distributions on the sphere [16, 71].

SCVTs come with a sense of "optimality" in node locations due to energy minimizing properties [16]. The process to generate SCVTs involves constructing a Voronoi diagram, computing the mass centroids for each Voronoi partition, and updating node locations to the mass centroids projected onto the sphere. After a number of iterations the nodes coincide with the projected mass centroids and a converged SCVT is produced. In most large-scale applications, this iterative process leverages a probabilistic Lloyd's method, where integrals to compute mass centroids are approximated through random sampling [16, 71]. While SCVTs in theory converge to a regular node distribution, the probabilistic nature of the centroid calculation introduces irregularities in distribution reminiscent of MD nodes.

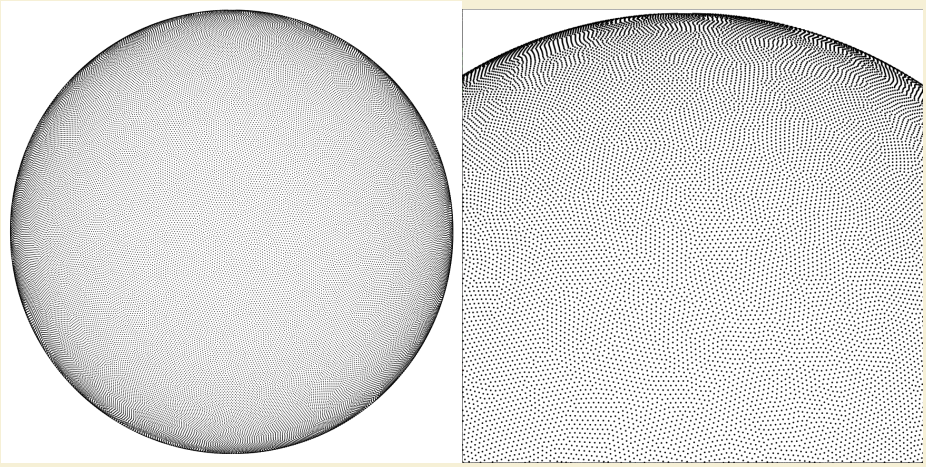


Figure 3.4: (Left) N=100,000 Spherical Centroidal Voronoi Tessellation nodes. (Right) Close-up of the same N=100,000 nodes to illustrate the irregularities in the grid.

Figure 3.4 provides an example SCVT grid with N=100,000 nodes. On the left, the full sphere; on the right, a close-up of the same node set. The close-up perspective clearly demonstrates the random artifacts/scarring of irregularly distributed nodes. For benchmarking purposes we use node sets N=100,000, N=500,000 and N=1,000,000 generated by the SCVT library from [71]. These SCVT grids are publicly available from: https://github.com/bollig/sphere_grids.git.

Chapter 4

Generating Stencils for RBF-FD

TODO:

- Finish KDTree description
- Hashing neighbor query algorithm description
- Cite refs on multi-dim spatial data structures
- Complexity of algorithms
- impact from ordering on matrix sparsity. Bandwidth impact. Bandwidth impact on condition considered in future chapter.
- what is best overlay resolution? based on time to generate. choose resolution as n/2, n/9? etc?
- perform neighbor queries on raster order because its easiest to jump cells. then each cell can be reordered according to z,x,u, etc. for better memory locality. Matlab script to do this (can be ported to C)

With a grid present, stencils are generated by querying n neighbors for each N total nodes in the domain. While not required, one typically assumes a stencil of n nodes must be the nearest neighbors to a stencil center. Afterall, as the distance between stencil nodes and the stencil center decreases, the accuracy of derivatives increases.

Brute force searching for neighbors—computing the distance between every pair of nodes and then selecting the n nearest—is discouraged due to its $O(N^2)$ complexity. Common practice in the RBF community is to construct kD-Trees to decrease the cost of queries (e.g., [19, 24, 31]).

Author's Note: Incomplete here to end of section

Many algorithms exist to query the k-nearest neighbors (equivalently all nodes in the minimum/smallest enclosing circle). Some algorithms overlay a grid similar to Locality Sensitive Hashing and query such as... [**?**].

RBF-FD is designed to handle irregular node distributions. Therefore, it is not essential that stencils contain only nearest neighbors. Instead, one can acquire the *approximate nearest neighbors*. Figure **??** demonstrates a case where a does not contain all nearest

neighbors. As illustrated in the Figure, the ANN stencil and true nearest neighbor stencil differ by one node. THis is not dire

Leveraging kD-Trees involves two costs: a) the initial tree construction, and b) k-nearest neighbor queries.

GPU version of Locality Sensitive Hashing could reduce complexity further [51]

This can be done efficiently using neighbor query algorithms or spatial partitioning data-structures such as Locality Sensitive Hashing (LSH) and kD-Tree. Different query algorithms often have a profound impact on the DM structure and memory access patterns. We choose a Raster (ijk) ordering LSH algorithm [**?**] leading to the matrix structure in Figures **??** and **??**. While querying neighbors for each stencil is an embarrassingly parallel operation, the node sets used here are stationary and require stencil generation only once. Efficiency and parallelism for this task has little impact on the overall run-time of tests, which is dominated by the time-stepping. We preprocess node sets and generate stencils serially, then load stencils and nodes from disk at run-time. In contrast to the RBF-FD view of a static grid, Lagrangian/particle based PDE algorithms promote efficient parallel variants of LSH in order to accelerate querying neighbors at each time-step [36, 51].

At the onset of our work on RBF-FD, the most commonly used KDTree implmentation used by the RBF community was [**?**]. Recently, improvements were made to the KDTree algorithm to reduce the cost of building the KDTree to $O(Nlog^2N)$.

Figure **??** compares the total time to generate N stencils of size $n = 50$ with three methods: [**?**], our hash-based neighbor query, and the improved KDTree from [**?**]. Until the new release of KDTree, our algorithm was a major improvement to the performance of stencil generation. The hash-based approach achieved greater than

RBF-FD operates on general node distributions. Historically, stencils are uniform in size (n) and generated by selecting the $(n-1)$ true nearest neighbors to a node x_c. This is a k-NN query.

Alternative queries are possible: ball query and approximate nearest neighbor. The approximate is of particular interest because nodes closest to the stencil will always be selected, whereas the nodes further away have minimal influence so swapping out cant hurt. The justification in altering the selection is for reduced complexity in neighbor queries.

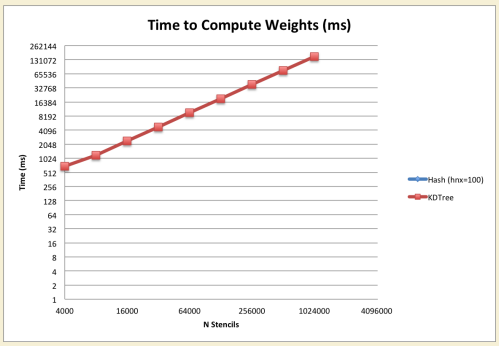
For example, in general brute force is inefficient The author of [19] queries n nearest neighbors for a compact-support RBF partition of unity example with a k-D tree. In [24, 31] a k-D Tree is leveraged for all neighbor queries for RBF-FD.

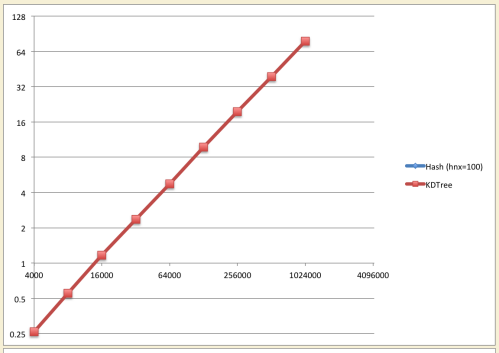
Our work in [6] leveraged an alternative to k-D tree, based loosely on space-filling curve orderings common in Lagrangian schemes like Smoothed Particle Hydrodynamics (e.g., [**?**], [**?**]).

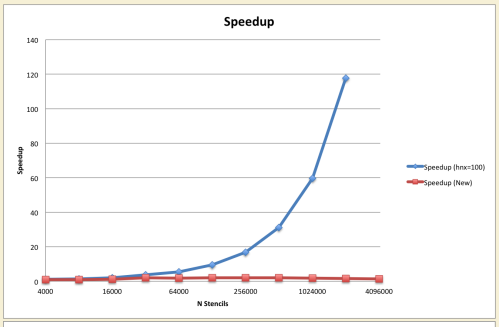
Rather than iterate through all N nodes to find the true neighbors, or step through a k-D tree in something like $O(logN)$ that requires extra built-out, ANN allows us to use a set of nodes that satisfy

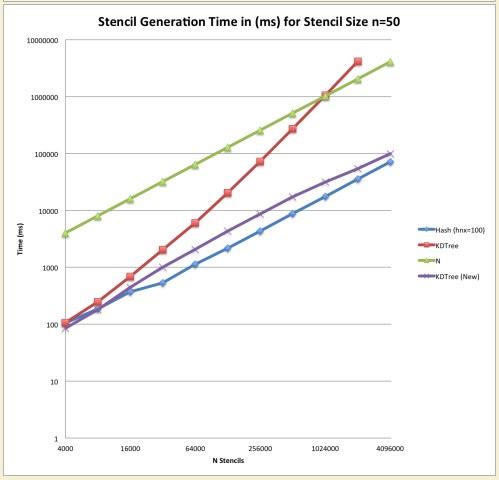
KDTree

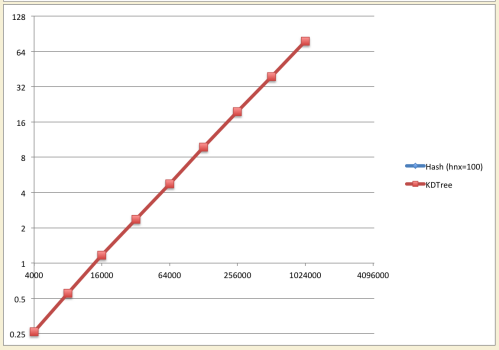
Most of the RBF community leverages the k-D tree, due to its low computational complexity for querying neighbors and its wide availability as standalone software in the public domain (e.g., matlab central has a few implementations for download, and the MATLAB Statistics

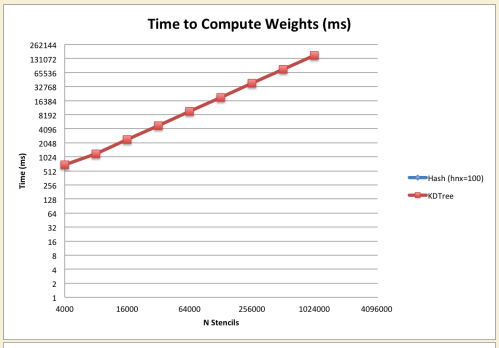

Time to Compute Weights (ms)
Time (ms)
N Stencils
Hash (hnx=100)
KDTree

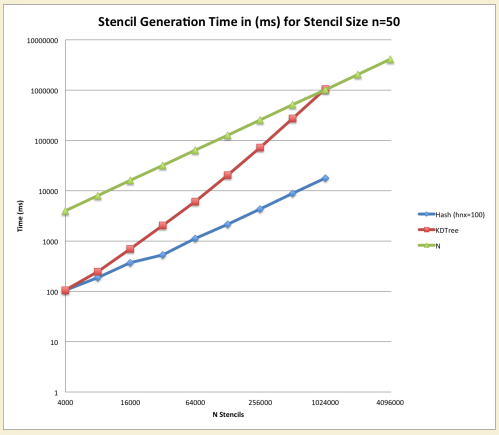

Hash (hnx=100)
KDTree

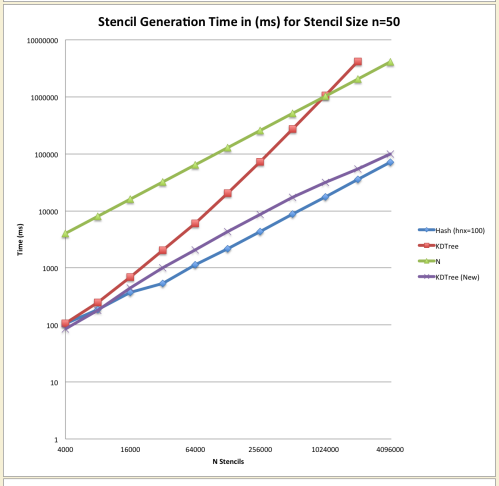

Speedup
Speedup
N Stencils
Speedup (hnx=100)
Speedup (New)

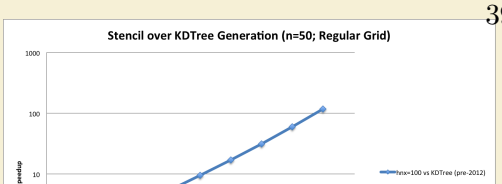

Stencil Generation Time in (ms) for Stencil Size n=50
Time (ms)
N Stencils
Hash (hnx=100)
KDTree
N
KDTree (New)

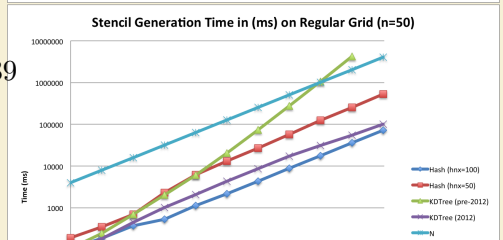

Hash (hnx=100)
KDTree


Time to Compute Weights (ms)
Time (ms)
N Stencils
Hash (hnx=100)
KDTree


Stencil Generation Time in (ms) for Stencil Size n=50
Time (ms)
N Stencils
Hash (hnx=100)
KDTree
N


Stencil Generation Time in (ms) for Stencil Size n=50
Time (ms)
N Stencils
Hash (hnx=100)
KDTree
N
KDTree (New)


Stencil over KDTree Generation (n=50; Regular Grid)
hnx=100 vs KDTree (pre-2012)


Stencil Generation Time in (ms) on Regular Grid (n=50)
Time (ms)
Hash (hnx=100)
Hash (hnx=50)
KDTree (pre-2012)
KDTree (2012)
N

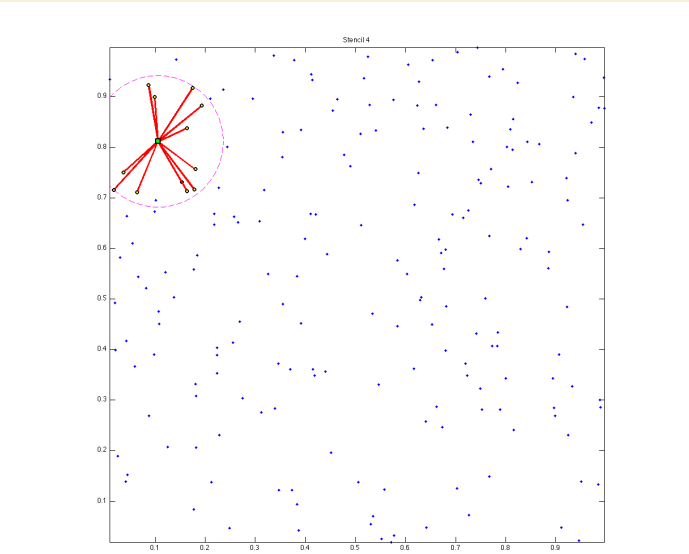


Figure 4.2: Example of an Approximate Nearest Neighbor (ANN) stencil, with all but one of the true nearest neighbors included in the stencil.

Toolbox includes an efficient k-D Tree).

The complexity of assembling he tree is

The Matlab central k-D Tree is MEX compiled and efficient. We integrated the standalone C++ code into our library.

While the k-D Tree functions well for queries, its downfall is a large cost in preprocessing to build the tree. For moving nodes, such as in Lagrangian schemes, this cost is prohibitively high. In an attempt to reduce the cost, lagrangian schemes introduced approximate nearest neighbor queries based on

Hashing

Approximate nearest neighbors will be nearly balanced. We observe that RBF-FD functions as well on stencils of true nearest neighbors as it does on approximate nearest neighbors.

Consider Figure 4.2 in which an Approximate Nearest Neighbor stencil is constructed. For this stencil, all but one of the nearest neighboring nodes are chosen.

Hashing, shown in Figure **??** overlays a regular grid. This is equivalent to an axis aligned bounding box AABB, with refinement. In other words, we form a quad-tree in 2D, an octree in 3D. The neighbor query starts with the cell in which x_c resides. Since we use an axis aligned bounding box, this cell index is easily calculated given the coordinate and number of subdivisions in each dimension. Once the cell index is resolved, the stencil is populated by taking the n nearest neighbors from within the current cell. If the cell does not contain sufficient number of nodes to fill the stencil, the search for neighbors expands to include the cells immediately adjoining the center cell, taking only the nearest nodes in the provided cells. The search continues to expand outward in a rasterized circle/sphere until n is satisifed. This search is considered approximate because it can happen that a

true nearest neighbor would lie in a cell that is is not included in the rasterized circle, and other nodes are substituted from the far reaches of the discretized grid.

The complexity of the method is still higher than the more efficient implementations used by Lagrangian methods, but as demonstrated in Figure **??** the savings are significant. Generating stencils for RBF-FD is a preprocessing cost, so we do not dedicate an excessive amount of attention to this algorithm. However, a few ideas that would improve: hilbert ordering, choose AABB resolution based on N not user parameters, faster sorting, GPU implementation

To demonstrate the savings in choice of stencil generation method, we provide Figure **??**.

The impact of our neighbor query also extends influence on the structure of the RBF-FD DMs. has is to To quantify the sparsity of a Differentiation Matrix we consider the ratio of non-zeros ($N * n$) to total elements in the matrix (N^2). For example, a problem of size $N = 10,000$ with stencil size $n = 31$ has a ratio of 0.0031 and is 99.69% empty.

Querying neighbors requires searching at least the immediate cell one layer of neighbors. by including one extra layer we ensure that small stencils near the border of the immediate cell can pick up neighbors in adjacent cells.

The KDTree implementation used in this work is from

Original data showed our algorithm as wildly successful against a version

Obviously, the ideal case for bandwidth is when all rows contain the $\frac{n}{2}$ nodes corresponding to solution value to either side of u_j. In 1-D this corresponds to every node containing the $\frac{n}{2}$ nodes to the left and right of x_j. In 2-D this is only possible if the nodes in the domain are properly indexed such that stencils contain the proper set of neighbors—a stringent requirement that will

4.0.2 On Choosing the Right ϵ

If solving for the weights directly (i.e., inverting Equation 3.2), one must carefully choose ϵ to prevent ill-conditioning. Numerous attempts exist in literature to provide "good" functions for ϵ based on node spacing, h, stencil size n, etc. In general, the values provided are particular to the specific problem and/or grid under consideration. No fool-proof method exists to select the support parameters, so the prospect of hassle-free application of the RBF-FD method is still out of reach.

Modern algorithms introduced in the last couple years provide methods for acquiring weights. Contour-Padé, RBF-QR, and RBF-GA are all options for weights corresponding to the $\epsilon \to 0$. These methods appear to resolve many issues including conditioning of implicit systems for more accurate solutions [14] [**?**]

In this work, we have attempted a variety of scalings on ϵ. We forego discussion of these attempts since they are outside our current scope of investigation. In the end, we find that the most effective method for choosing ϵ was to adopt the approach introduced in [24], wherein ϵ is expressed as a function of the number of nodes N and desired mean condition number, $\bar{\kappa}_A$.

The optimal ϵ for general node distributions is out of the scope of this research and not necessary for our purposes. For now we assume that nodes are more or less regularly distributed, either via some algorithm such as Lloyd's method to create Centroidal Voronoi Tessellations, repelling springs (distmesh), minimum energy or minimum determinant, or

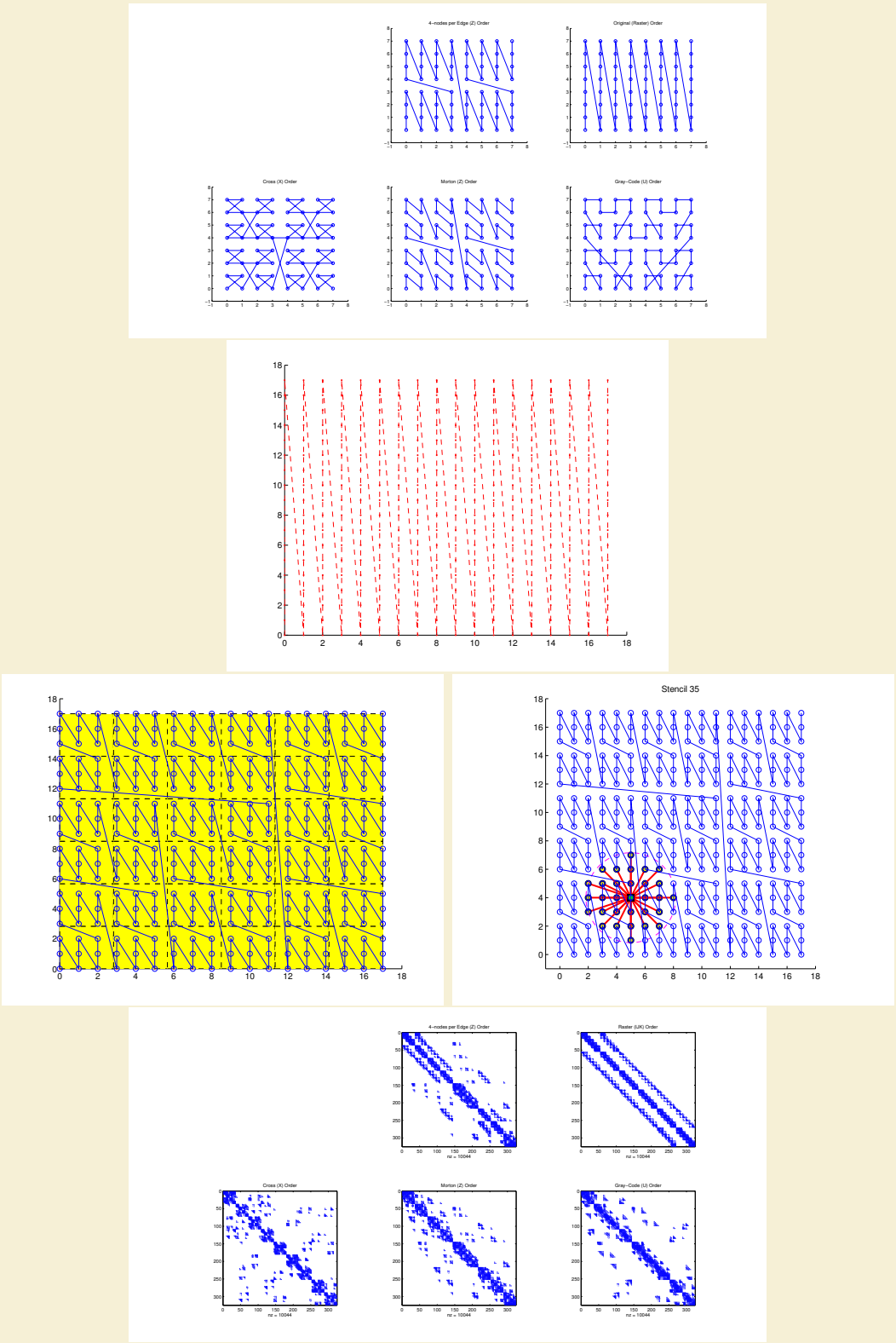


Figure 4.3: In order: a) node ordering test cases; b) original ordering of regular grid (raster); c) coarse grid overlay for hash functions ($hnx = 6$); d) example stencil ($n = 31$) spanning multiple Z's; e) spy of DM after orderings.

some other algorithm. Except in cases where convergence on PDE solutions can not be attained via RBF-FD, we find that a direct solve for RBF-FD weights is sufficient. To a certain extent, this work considers parallelization of the method so it does not matter if our weights are precise or not.

We adopt similar appraoch as [24] by choosing the same epsilon values specified in Table **??**.

I also produced a Matlab script to generate the contours for any stencil size and produced the following figures.

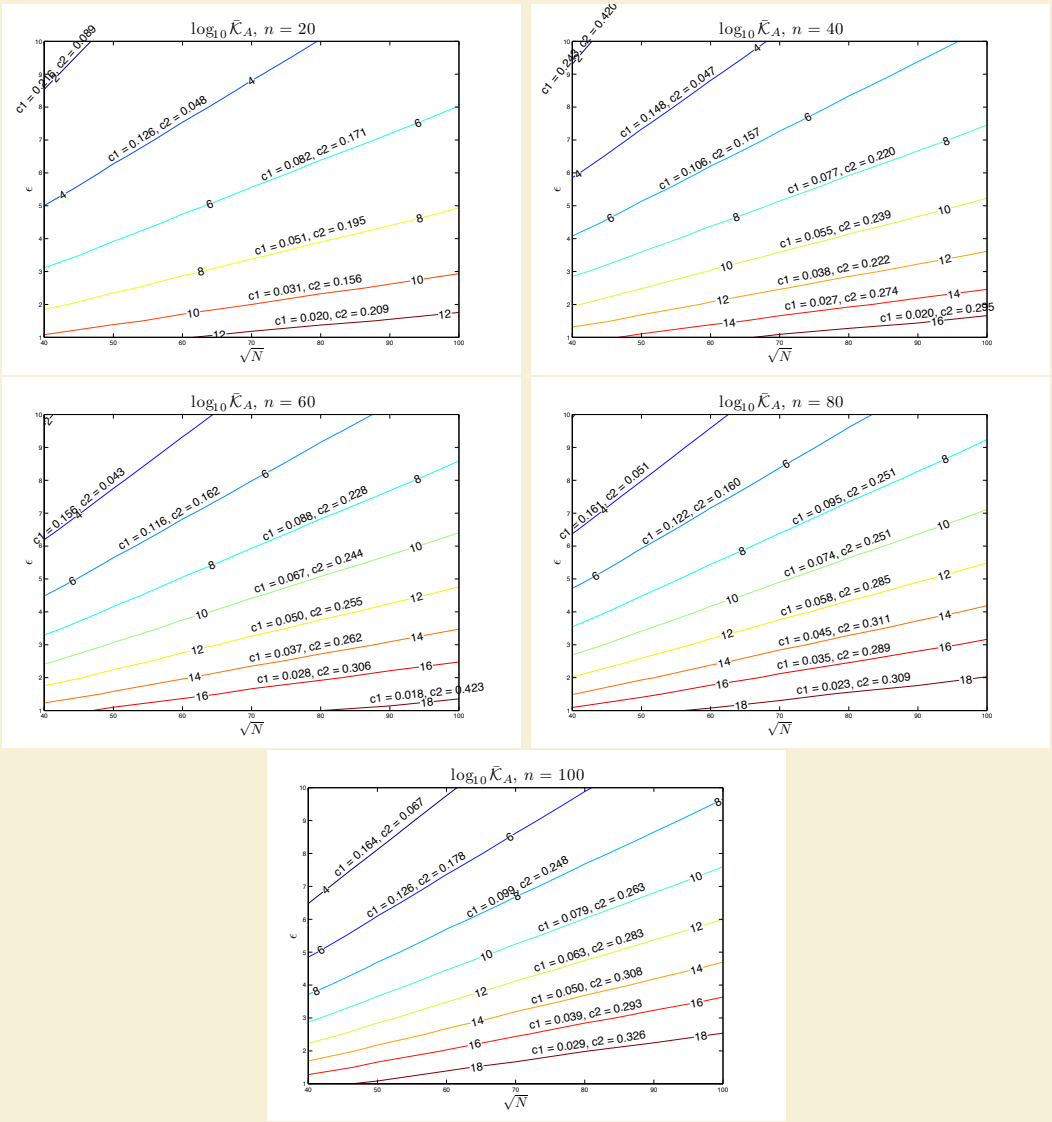


Figure 4.4: Contours for choosing ϵ for $n = 20$, 40, 60, 80 and 100 on the unit sphere as a function of $\sqrt{N}$. Contours assume near uniform distribution of nodes (e.g., maximum determinant nodes, icosahedral grids and spherical centroidal voronoi tessellations). Superimposed above each contour are parameters for the linear regression of the line, $c_1\sqrt{N} - c_2$.

Chapter 5

GPU SpMV

5.1 Related Work

[5] [**?**] [69] etc.

5.2 GPGPU

GPGPU evolution

5.2.1 OpenCL

OpenCL is chosen with the future in mind. Hardware changes rapidly and vendors often leapfrog one another in the performance race. By selecting OpenCL, we hedge our bets on the functional portability

5.2.2 Hardware Layout

Modern GPUs have a memory hierarchy and hardware layout.

5.3 Performance

5.3.1 GFLOP Throughput

In order to quantify the performance of our implementation, we can measure two factors. First, we can check the speedup achieved on the GPU relative to the CPU to get an idea of how much return of investment is to be expected by all the effort in porting the application to the GPU. Speedup is measured as the time to execute on the CPU divided by the time to execute on the GPU.

The second quantification is to check the throughput of the process. By quantifying the GFLOP throughput we have a measure that tells us two things: first, a concrete number quantifying the amount of work performed per second by either hardware, and second because we can calculate the peak throughput possible on each hardware, we also have a measure of how occupied our CPU/GPU units are. With the GFLOPs we can also

determine the cost per watt for computation and conclude on what problem sizes the GPU is cost effective to target and use.

Now, as we parallelize across multiple GPUs, these same numbers can come into play. However we are also interested in the efficiency. Efficiency is the speedup divided by the number of processors. With efficiency we have a measure of how well-utilized processors are as we scale either the problem size (weak) or the number of processors (strong). As the efficiency diminishes we can conclude on how many stencils/nodes per processor will keep our processors occupied balanced with the shortest compute time possible (i.e., we are maximizing return of investment).

5.3.2 Expectations in Performance

Many GPU applications claim a 50x or higher speedup. This will never be the case for RBF-FD for the simple reason that the method reduces to an SpMV. The SpMV is a low computational complexity operation with only two operations for every one memory load.

5.4 Targeting the GPU

5.4.1 OpenCL

5.4.2 Naive Kernels

5.4.3 SpMV Formats/Kernels

5.5 Performance Comparison

5.5.1 Performance of Cosine CL vs VCL

5.5.2 VCL Formats Comparison

Our assumption with RBF-FD in this manuscript is that all stencils will have equal size. Due to this, the ELL format is preferred as the default.

We are investigating optimizations that target both GPUs and Phi cards for a class of numerical methods based on Radial Basis Functions (RBFs) to solve Partial Differential Equations. RBF methods are increasingly popular across disciplines due to their low complexity, natural ability to function in higher dimension with minimal requirements for an underlying mesh, and high-order—in many cases, spectral—accuracy. RBF methods can be viewed as generalizations of many traditional methods such as Finite Difference and Finite Element to allow for truly unstructured grids. This generalization allows one to reuse many of the same techniques (e.g., sparse matrices, iterative solvers, domain decompositions, etc.) to efficiently obtain solutions. The variety of hardware available on Cascade will help us establish a clear argument in the choice of accelerator type and resolve the dilemma between choosing Phi vs GPU for our method. Since RBFs generalize other methods, our results should have broad reaching impact to answer similar questions for related methods.

With the generalization of RBF-FD derivative computation formulated as a sparse matrix multiplication, we can consider the various sparse formats provided by CUSP and ViennaCL.

Compare formats:

- ELL
- COO
- CSR
- Other formats such as HYB, JAD, DIA are considered on the GPU

How is communication overlap handled with each format?

Conclude: sparse containers allow increased efficiency compared to our custom kernels. The custom kernels compete with CSR and COO.

From the definition of RBF-FD we can formulate the problem computationally in two ways. First, stencil operations are independent. Therefore, we can write kernels with perfect parallelism by dedicating a single thread per stencil or a group of threads per stencil.

Unfortunately, perfect concurrency does not imply perfect or even ideal concurrency on the GPU.

We first demonstrate the case where one thread is dedicated to each stencil. This is followed by dedicating a group of thread to the stencil. In each case we are operating under the assumption that each stencil is independent on the GPU.

To further optimize RBF-FD on the GPU, we formulate the problem in terms of a Sparse Matrix-Vector Mulitply (SpMV). When we consider the problem in this light we generate a single Differentiation Matrix that can see two optimizations not possible with our stencil-based view:

- First, the sparse containers used in SpMV allow for their own unique optimizations to compress storage and leverage hardware cache.
- Evaluation of multiple derivatives can be accumulated by association into one matrix operation. This reduces the total number of floating point operations required per iteration.

We compare the performance of our custom kernel to ViennaCL kernels (ELL, CSR, COO, HYB, DIAG), UBlas (COO, CSR) and Eigen (COO, CSR, ELL)

Chapter 6

Distributed RBF-FD

Parallelizing RBF-FD in a distributed environment requires three design decisions [55]. First, the problem is partitioned in some fashion to distribute work across multiple processes. Intelligent partitioning impacts load balancing of processors and the ratio of computation versus communication; imbalanced computation can result in excessive delay per iteration as some processors tackle larger problem sizes with others sitting idle. Second, one must determine whether processes have access to all or a subset of node information, solution values, etc. and establish index mappings that translate between a local context and the global problem. Third, the local ordering of indices is established to improve solver efficiency and/or simplify operations. Node ordering is also significant in the future discussion of offloading computation to GPUs as it can help to minimize data transfer between CPU and GPU.

The following sections detail the approach to distributed computing and a few optimizations that allow RBF-FD to scale over a thousand processes. In later chapters these same decisions will tie into the resulting performance of the distributed multi-GPU implementations.

6.1 Partitioning

For ease of development and parallel debugging, partitioning is initially assumed to be linear within one physical direction (typically the x-direction). Figure 6.1 illustrates a partitioning of $N = 10,201$ nodes on the unit sphere onto four CPUs.

Each partition, illustrated as a unique color, contains many *stencil centers*. Although *stencil centers* are contained within a partition, there is no requirement for all *stencil nodes* to be contained within the same partition. As a result, many stencils require information updates from neighboring partitions for nodes that are referred to as *ghost nodes* [**?**]. In many cases, *ghost nodes* are treated the same as any other stencil node. The CPU process in charge of a partition is fully aware of the ghost node coordinate, current solution value(s), etc.. However, values at ghost nodes are modified by another process, so changes must be explicitly synchronized via an MPI collective for dependent processes to maintain consistency.

In Figure 6.1, alternating representations between node points and interpolated surfaces

illustrates the overlap regions where ghost nodes reside. Due to stencil dependencies in each partition, the overlap region representations are double-wide—i.e., they contain a set of ghost nodes for both the left and right partitions.

As the stencil size increases, the width of the overlap regions relative to total number of nodes on the sphere proportionally increases. In the case of the unit sphere from Figure 6.1, the width of the overlap is roughly $\sqrt{n}$ for stencil size n. Figure 6.1 shows the case of $n = 31$ nodes per stencil. Higher order RBF-FD stencils (i.e., larger stencil sizes) exacerbate the situation by further increasing the number of bytes that must be sent via MPI. Observe that since stencils need not have symmetric dependencies (i.e., if stencil s_1 depends on s_2, s_2 need not depend on s_1), the number of ghost nodes for each partition can vary.

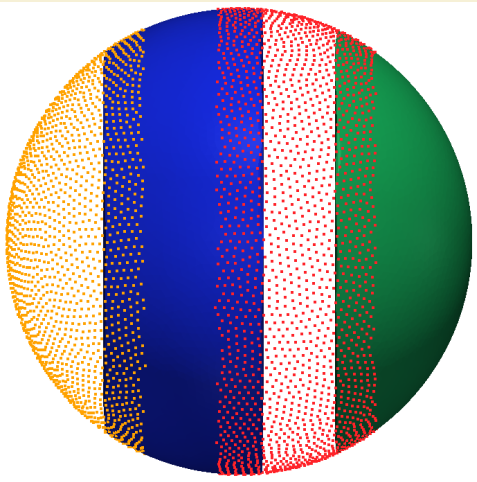


Figure 6.1: Partitioning of $N = 10,201$ nodes to span four processors with stencil size $n = 31$.

The choice for a linear partitioning is simple and easy to code. Each MPI process has a left and right neighbor, so communication is straightforward. On a high number of processors, there are two issues: 1) increasing the number of processors quickly reduces the width of each partition and can result in stencils dependent on more than one partition in each direction introducing the need for more complex collectives; and 2) with near uniform node distributions the resulting partitions are of unequal size and processors are improperly balanced. Thus, in the case of the sphere, linear partitioning is not ideal.

Many other options for partitioning the sphere exist. In atmospheric and geophysical communities for example, one often finds the cubed-sphere [41**?**], which transcribes a subdivided cube onto the sphere and assigns projected rectangular elements to individual processors. Another option is the icosahedral geodesic grid [52], which evenly balances the computational load by distributing equal sized geodesic triangles across processors. The options for partitioning the sphere are endless, and are outside the scope of this work.

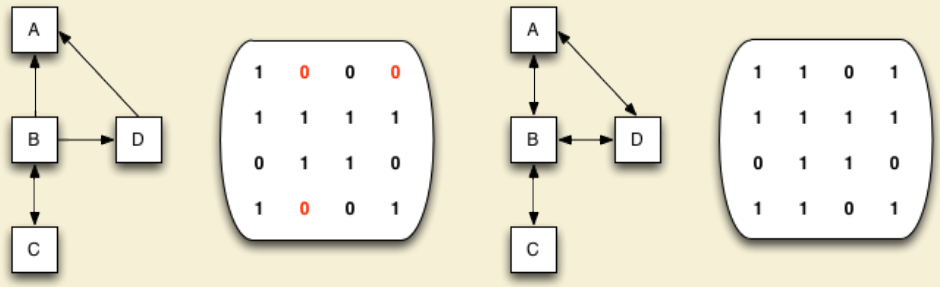
Other interesting partitionings can be generated with software libraries such as the METIS [44] family of algorithms, capable of partitioning and reordering directed graphs produced by RBF-FD stencils.

In order to partition our nodes, METIS requires an undirected adjacency graph repre-

senting the edges that connect nodes. In this case the adjacency graph represents edges connecting nodes in a mesh. For RBF-FD there is no well-defined mesh. Rather, every node is connected to multiple stencil centers. of connecting stencil nodes to stencil centers. An undirected To produce this we generate

The undirected graph is used only for partitioning and subsequently discarded.

METIS divides stencils into contiguous partitions of nearly equivalent size.



(a) Directed (Unsymmetric) (b) Undirected (Symmetric)

Figure 6.2: A simple adjacency graph and corresponding matrices. Edges connecting nodes of RBF-FD stencils produce (a) a directed adjacency matrix. To partition RBF-FD stencils, METIS requires conversion to (b) an undirected graph/matrix.

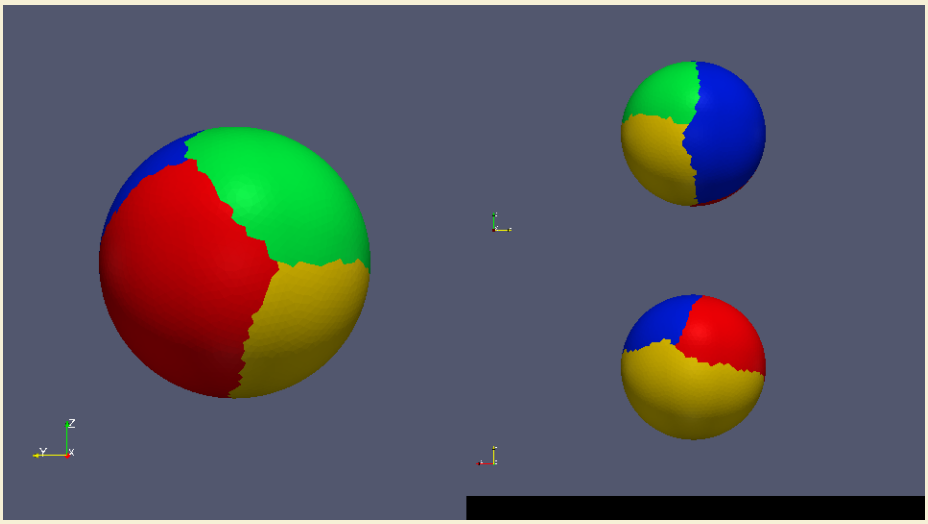


Figure 6.3: METIS partitioning of $N = 10,201$ nodes to span four processors with stencil size $n = 31$.

6.2 Index Mappings and Local Node Ordering

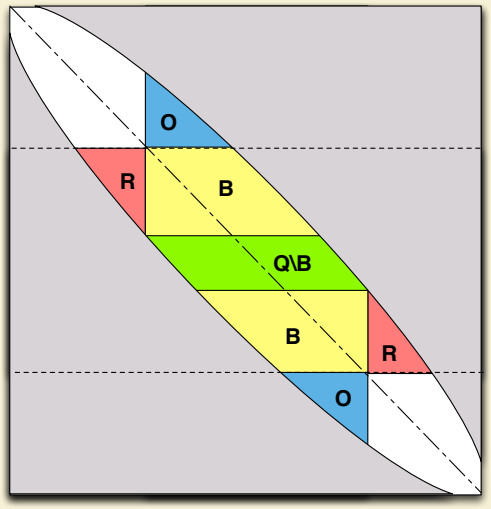


Figure 6.4: Decomposition for one processor selects a subset of rows from the DM. Blocks corresponding to node sets $\mathcal{Q}\backslash\mathcal{B}$, $\mathcal{O}$, and $\mathcal{R}$ are labeled for clarity. The subdomain for the processor is outlined by dashed lines.

6.3 Local node ordering

After partitioning, each CPU/GPU is responsible for its own subset of nodes. To simplify accounting, we track nodes in two ways. Each node is assigned a global index, that uniquely identifies it. This index follows the node and its associated data as it is shuffled between processors. In addition, it is important to treat the nodes on each CPU/GPU in an identical manner. Implementations on the GPU are more efficient when node indices are sequential. Therefore, we also assign a local index for the nodes on a given CPU, which run from 1 to the maximum number of nodes on that CPU.

It is convenient to break up the nodes on a given CPU into various sets according to whether they are sent to other processors, are retrieved from other processors, are permanently on the processor, etc. Note as well, that each node has a home processor since the RBF nodes are partitioned into multiple domains without overlap. Table 6.1, defines the collection of index lists that each CPU must maintain for both multi-CPU and multi-GPU implementations.

Figure 6.6 illustrates a configuration with two CPUs and two GPUs, and 9 stencils, four on CPU1, and five on CPU2, separated by a vertical line in the figure. Each stencil has size $n = 5$. In the top part of the figures, the stencils are laid out with blue arrows pointing to stencil neighbors and creating the edges of a directed adjacency graph. Note that the connection between two nodes is not always bidirectional. For example, node 6 is in the stencil of node 3, but node 3 *is not* a member of the stencil of node 6. Gray arrows point to stencil neighbors outside the small window and are not relevant to the following discussion, which focuses only on data flow between CPU1 and CPU2. Since each CPU is responsible for the derivative evaluation and solution updates for any stencil center, it is clear that some nodes have a stencil with nodes that are on a different CPU. For example,

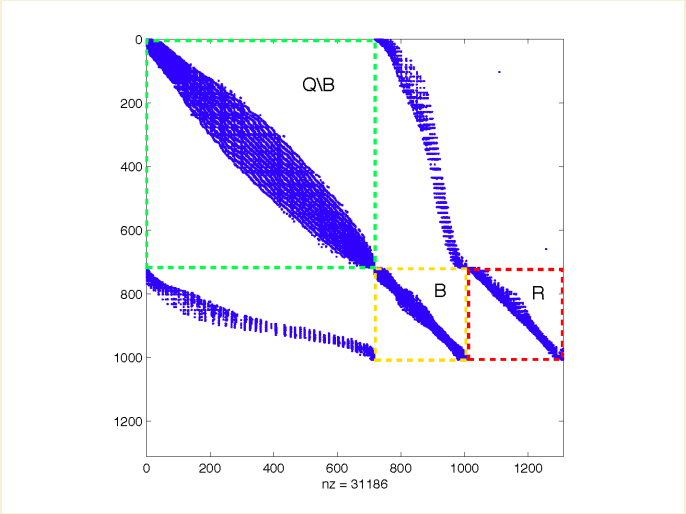


Figure 6.5: Spy of the sub-DM view on processor 2 of 4 from a METIS partitioning of $N = 10,201$ nodes with stencil size $n = 31$. Blocks are highlighted to distinguish node sets $\mathcal{Q}\backslash\mathcal{B}$, $\mathcal{O}$, and $\mathcal{R}$. Stencils involved in MPI communications have been permuted to the bottom of the matrix.

node 8 on CPU1 has a stencil comprised of nodes 4,5,6,9, and itself. The data associated with node 6 must be retrieved from CPU2. Similarly, the data from node 5 must be sent to CPU2 to complete calculations at the center of node 6.

The set of all nodes that a CPU interacts with is denoted by $\mathcal{G}$, which includes not only the nodes stored on the CPU, but the nodes required from other CPUs to complete the calculations. The set $\mathcal{Q} \in \mathcal{G}$ contains the nodes at which the CPU will compute derivatives and apply solution updates. The set $\mathcal{R} = \mathcal{G}\backslash\mathcal{Q}$ is formed from the set of nodes whose values must be retrieved from another CPU. For each CPU, the set $\mathcal{O} \in \mathcal{Q}$ is sent to other CPUs. The set $\mathcal{B} \in \mathcal{Q}$ consists of nodes that depend on values from $\mathcal{R}$ in order to evaluate derivatives. Note that $\mathcal{O}$ and $\mathcal{B}$ can overlap, but differ in size, since the directed adjacency graph produced by stencil edges is not necessarily symmetric. The set $\mathcal{B}\backslash\mathcal{O}$ represents nodes that depend on $\mathcal{R}$ but are not sent to other CPUs, while $\mathcal{Q}\backslash\mathcal{B}$ are nodes that have no dependency on information from other CPUs. The middle section Figure 6.6 lists global node indices contained in $\mathcal{G}$ for each CPU. Global indices are paired with local indices to indicate the node ordering internal to each CPU. The structure of set $\mathcal{G}$,

$$\mathcal{G} = \{\mathcal{Q}\backslash\mathcal{B} \;\; \mathcal{B}\backslash\mathcal{O} \;\; \mathcal{O} \;\; \mathcal{R}\}, \tag{6.1}$$

is designed to simplify both CPU-CPU and CPU-GPU memory transfers by grouping nodes of similar type. The color of the global and local indices in the figure indicate the sets to which they belong. They are as follows: white represents $\mathcal{Q}\backslash\mathcal{B}$, yellow represents $\mathcal{B}\backslash\mathcal{O}$, green indices represent $\mathcal{O}$, and red represent $\mathcal{R}$.

The structure of $\mathcal{G}$ offers two benefits: first, solution values in $\mathcal{R}$ and $\mathcal{O}$ are contiguous in memory and can be copied to or from the GPU without the filtering and/or re-ordering normally required in preparation for efficient data transfers. Second, asynchronous communication allows for the overlap of communication and computation. This will be considered

$\mathcal{G}$	: all nodes received and contained on the CPU/GPU g
$\mathcal{Q}$	: stencil centers managed by g (equivalently, stencils computed by g)
$\mathcal{B}$	: stencil centers managed by g that require nodes from another CPU/GPU
$\mathcal{O}$	: nodes managed by g that are sent to other CPUs/GPUs
$\mathcal{R}$	: nodes required by g that are managed by another CPU/GPU

Table 6.1: Sets defined for stencil distribution to multiple CPUs

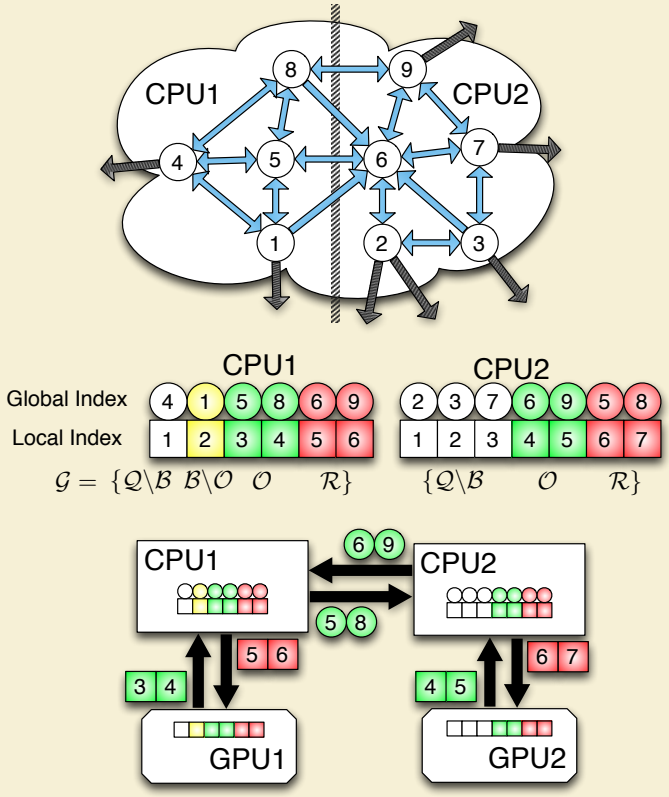


Figure 6.6: Partitioning, index mappings and memory transfers for nine stencils ($n = 5$) spanning two CPUs and two GPUs. Top: the directed graph created by stencil edges is partitioned for two CPUs. Middle: the partitioned stencil centers are reordered locally by each CPU to keep values sent to/received from other CPUs contiguous in memory. Bottom: to synchronize GPUs, CPUs must act as intermediaries for communication and global to local index translation. Middle and Bottom: color coding on indices indicates membership in sets from Table 6.1: $\mathcal{Q}\backslash\mathcal{B}$ is white, $\mathcal{B}\backslash\mathcal{O}$ is yellow, $\mathcal{O}$ is green and $\mathcal{R}$ is red.

as part of future research on algorithm optimization. Distinguishing the set $\mathcal{B}\backslash\mathcal{O}$ allows the computation of $\mathcal{Q}\backslash\mathcal{B}$ while waiting on $\mathcal{R}$.

Author's Note: The local index set is ordered as QmB, BmO, O, R

Author's Note: Domain boundary nodes appear at beginning of the list

Figure 6.1 illustrates a partitioning of $N = 10,201$ nodes on the unit sphere onto four CPUs. Each partition, illustrated as a unique color, represents set $\mathcal{G}$ for a single CPU.

Alternating representations between node points and interpolated surfaces illustrates the overlap regions where nodes in sets $\mathcal{O}$ and $\mathcal{R}$ (i.e., nodes requiring MPI communication) reside. As stencil size increases, the width of the overlap regions relative to total number of nodes on the sphere also increases.

When targeting the GPU, communication of solution or intermediate values is a four step process:

1. Transfer $\mathcal{O}$ from GPU to CPU
2. Distribute $\mathcal{O}$ to other CPUs, receive R from other CPUs
3. Transfer $\mathcal{R}$ to the GPU
4. Launch a GPU kernel to operate on $\mathcal{Q}$

The data transfers involved in this process are illustrated at the bottom of Figure 6.6. Each GPU operates on the local indices ordered according to Equation (6.1). The set $\mathcal{O}$ is copied off the GPU and into CPU memory as one contiguous memory block. The CPU then maps local to global indices and transfers $\mathcal{O}$ to other CPUs. CPUs send only the subset of node values from $\mathcal{O}$ that is required by the destination processors, but it is important to note that node information might be sent to several destinations. As the set $\mathcal{R}$ is received, the CPU converts back from global to local indices before copying a contiguous block of memory to the GPU.

This approach is scalable to a very large number of processors, since the individual processors do not require the full mapping between RBF nodes and CPUs.

By scalable here we imply total problem size and processor count. The performance scalability of the code depends on the problem size and the MPI collective. In Figure **??** the strong scaling of $N = 10^6$ nodes is tested on Itasca, a supercomputer at the Minnesota Supercomputing Institute.

6.4 Test Case

To test and demonstrate scaling of our method, we consider an idealized regular grid in three dimensions.

verification here is only significant to ensure we are applying all weights. We apply weights to calculate derivatives of a test function in X, Y, Z, and the Laplacian. the grid is regular and 3D. We test strong scaling on a $N = 160^3$ grid, and weak scaling with $N_p = 4000$. This way at $p = 1024$ processes we have weak scaling testing the full $N = 160^3$ grid.

6.5 Communication Collectives

MPI collectives allow information sharing between processes. Our code leverages three collectives: MPI_Alltoall, MPI_Alltoallv and MPI_Isend/MPI_Irecv.

The collective operation is essentially transposing information as seen in Figure 6.8.

MPI_Alltoall requires that all processors send and receive an equivalent number of bytes to one another. Since the size must be equivalent for all processors, the send and receive

buffers are padded to the maximum message size for any one connection between processors. MPI_Alltoallv reduces the number of bytes sent and received by allowing processors to specify variable message sizes when communicating. For a small number of processors the variable message size will function well. However, MPI_Alltoallv requires all processes to connect with every other process, even in the event that 0 bytes are to be sent. Based on the grid decomposition, processors compute on contiguous partitions with a small number of neighboring partitions. By replacing the MPI_Alltoallv with a MPI_Isend/MPI_Irecv combination, the number of collective connections are truncated such that processors only connect to and communicate with essential neighbors that need/provide data.

The actual implementation of MPI_Alltoall and MPI_Alltoallv likely use Isend and Irecv internally.

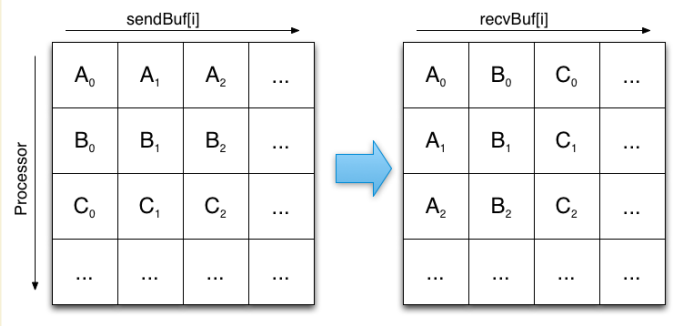


Figure 6.7: The MPI_Alltoall collective allows processors to interchange/transpose data by passing an equivalent number of bytes to every other processor.

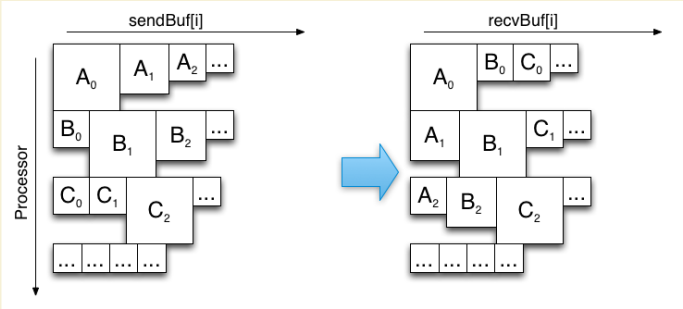


Figure 6.8: The MPI_Alltoallv collective compresses the interchange from MPI_Alltoall by allowing for variable message sizes between all processors. Assume message sizes are proportional to square size in figure.

MPI_Isend/MPI_Irecv also allows for overlapping communication and computation by posting receives early

6.5.1 Alltoallv

As a baseline for scaling we start with MPI_Alltoallv.

Author's Note: figure: alltoall visual

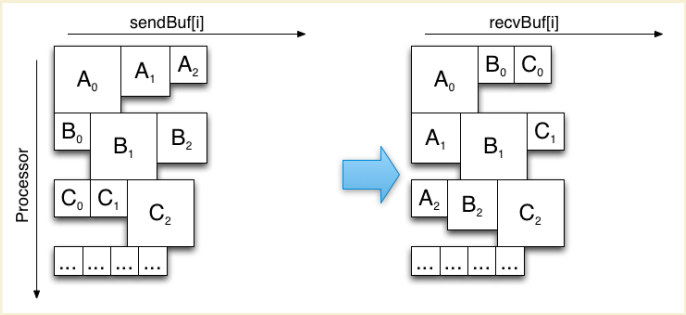


Figure 6.9: The MPI_Isend/MPI_Irecv collective allows for variable message sizes, and truncates the number of connections between processors to only required connections.

6.5.2 Isend/Irecv

The first improvement on Alltoallv collectives is to truncate the number of connections made between processes. Compact stencils implies an overlap region for each processor that draws values from a limited number of neighboring processors.

Author's Note: figure: isend visual

6.5.3 No Decode

Author's Note: figure: per iteration stacked bar for n=50 and 16 processes to show cost of decode Author's Note: figure: algorithm for collective

Author's Note: figure: alltoall to isend improvement. justify comm_combo for up to 16 procs. Author's Note: figure: comm_combo gains Author's Note: figure: algorithm for collective

6.5.4 Immediate Isend on Encode

Author's Note: figure: algorithm for collective

Author's Note: back to section: figure: improvement on all CPU collectives (n=50)

Author's Note: table: show percentage of comm time for actual mpi time. busy network can cause slower comm times. but the decode cost is gone. it can also be an issue if we have saturated comm pipes

6.6 CPU Scaling

Author's Note: Show the strong and weak scaling here

To demonstrate the effectiveness of our decomposition and indexing, we perform scaling experiments.

6.6.1 Strong Scaling

Strong scaling tests the growth in time for a fixed total problem size, and a variable number of processors.

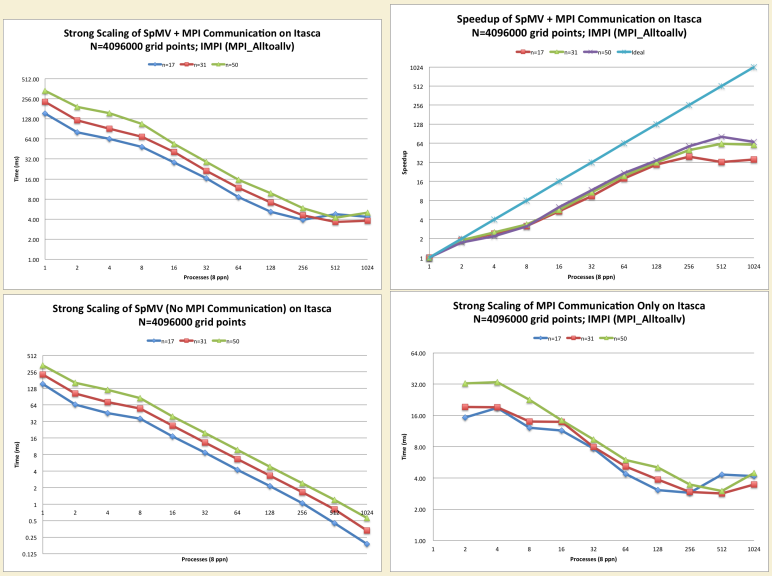


Figure 6.10: Strong scaling the distributed SpMV for $N = 4096000$ nodes (i.e., a 160^3 regular grid) and various stencil sizes. Here the MPI_Alltoallv collective operation is used. (Left) Strong scaling of SpMV (including cost of communication). (Center) Strong scaling of computation only. (Right) Strong scaling of communication only.

6.6.2 Weak Scaling

Weak scaling considers the amount of time for a fixed problem size per process and variable number of processors. That is to say, each processor has roughly the same amount of work, so as we scale to a large number of processors, changes in time will be the result of increased communication overhead.

Although our weak scaling results are promising, they also contain a problem. First, since we are subsampling a 160^3 regular grid to get the first $N = p * 4000$ nodes, many of the tests consider domains that are "L" shaped and have odd partitions with limited connectivity.

Author's Note: Here and strong scaling: table showing the min and max Osize,Rsize

6.6.3 Bandwidth

To understand the impact of MPI on these benchmarks we calculate the average and aggregate collective bandwidths. The average bandwidth considers the MPI throughput from the perspective of one processor.

The aggregate bandwidth reveals when processes saturate the interconnects.

We consider a simple idealized problem where derivatives are computed over a regular grid generated in 3-D. The experiment computes the SpMV one thousand times. At the end of each SpMV the MPI_Alltoallv collective is used to synchronize the local derivative vectors. After one thousand iterations, each process computes the local norm of the resulting vector and an MPI_Reduce collective dra

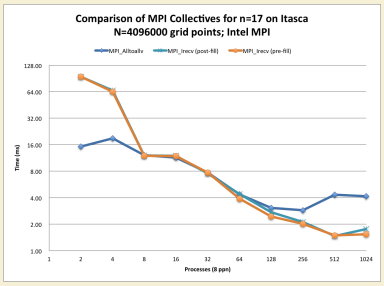


Figure 6.11: Scaling comparison of MPI_Alltoallv and two types of MPI_Isend/MPI_Irecv collectives: one with MPI_Irecv issued after filling the MPI_Isend send buffer (post-fill), and the other issued before filling the MPI_Isend buffer (pre-fill).

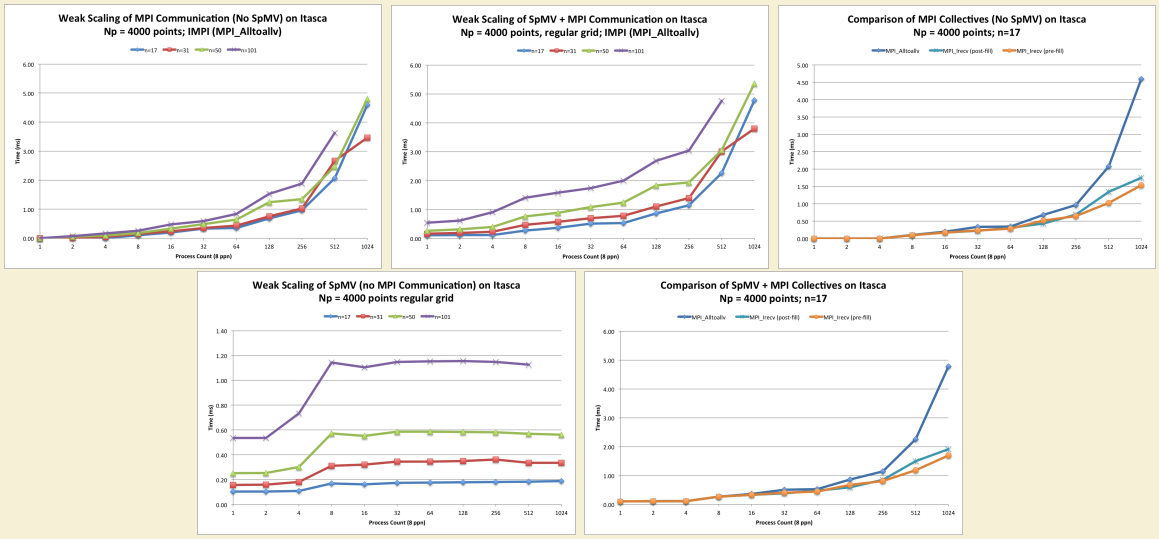


Figure 6.12: Weak scaling of the SpMV

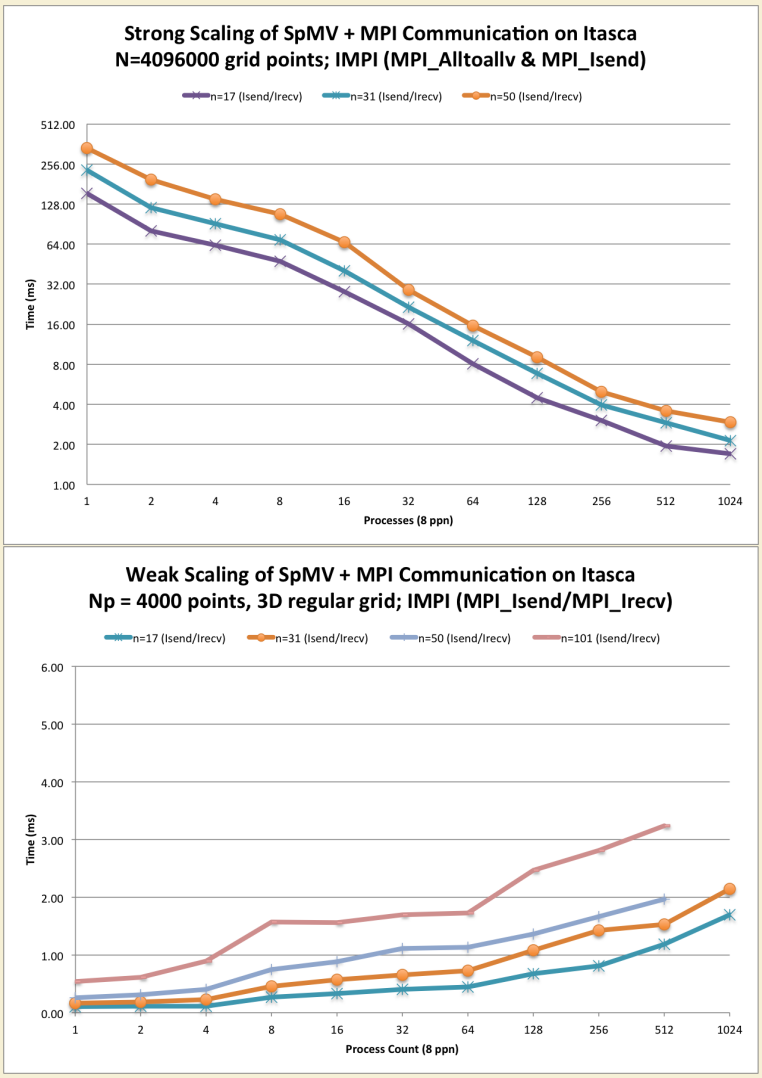


Figure 6.13: Scaling of SpMV with MPI_Isend/MPI_Irecv

Chapter 7

Distributed GPU SpMV

Distributing SpMV across multiple GPUs poses a new problem: as previous mentioned, the data sent and received via MPI collectives must be copied from device to host and vice-versa. To amortize this cost we introduce a novel overlapping algorithm to hide the cost of communication behind the cost of a concurrent SpMV on the GPU.

7.1 Overlapped Queues

7.2 Avoiding Copy Out

7.2.1 Avoiding Copy-Out on CPU

7.3 Scaling

We scale the SpMV across the GPUs on Cascade.

7.3.1 Fermi

7.3.2 Kepler

7.3.3 Shared K20s

Chapter 8

ViennaCL

Author's Note: 4 pages

The ViennaCL project [53, 54] is a sparse matrix library built on OpenCL that provides templated C++ API to easily and efficiently solve large sparse matrix problems.

Our decision to utilize ViennaCL stems from its

The API provides containers for sparse and dense matrices, vectors and views. The sparse formats available are COO, CSR, (...) ELL and HYB

Matrix and Vector Views provide slices and subranges of containers. Subviews were recently added in version 1.2 with full functionality in v1.3. These views are essential to our work on multi-GPUs since we need to operate on views with full vectors containing the ghost values. (Show example of view).

Algorithms are provided for GMRES, CG, etc.

To build multi-GPU algorithms for GMRES we discard the ViennaCL arnoldi process in favor of the Givens rotations. Then we add MPI communication.

8.1 ViennaCL Limitations

ViennaCL is a young library. When we began working with the library, it supported only CSR and COO formats. It has no support for multi-GPU computing. According to the author there were no intentions to extend it multiple GPUs due to the limited return of investment.

8.2 ViennaCL Additions

Added support for distributed SpMV and SAXPY with MPI communication.

Extended library to support rectangular systems rather than just square systems.

Introduced alternative GMRES algorithm based on Givens Rotations

Introduced preconditioner ILU0.

8.3 ViennaCL Matrix Formats

ViennaCL supports numerous sparse matrix formats.

Assemble once in CSR format using direct access notation (e.g., mat(row,col) = val), internally convert to more efficient sparse representation.

8.4 Numerical Libraries vs Low Level Kernels

When research began on this project, GPU computing was still in a stage that could be considered fresh. Porting our problem to the GPU, we faced the same challenge that most applications of the GPU face: can it be done better with an existing library or should we write low-level code. We argue that leveraging existing libraries is preferred when possible. There is no need to reproduce what already exists except in the cases where we seek to improve performance.

Having chosen to leverage numerical libraries where possible, we make attempts to generalize the problem of RBF-FD on Multiple GPUs in terms of simple existing primitives. As Figure **??** illustrates, the bulk of computation can be reduced to simple sparse matrix-vector operations, with vector updates and a few vector reduce steps. From this perspective, then, RBF-FD is incredibly simple to code.

8.5 ViennaCL

The ViennaCL library was chosen to

is leveraged for its sparse matrix and dense vector representations on the GPU, as w its efficient

Chapter 9

GMRES and Preconditioning

The first thing we need to discuss about the GMRES method of the basics of the algorithm. Once we have the basics we can discuss things such as our choice of pre-conditioner. Our work in RBF-FD is the first such test of pre-conditioners for RBS finite differences.

The the best plan of attack for today will be to write up related works as I find them. This dictation mode actually allows me to move faster. Simple things like spelling of the RB FFT method in the GM rice method will be hiccups however it should be obvious when it Dragon Dictate has not made the correct spelling and I'll be able to go back through on my second pass and actually revise the phrases into whatever words I need.

Would really like to know is where my computer is not really work. If I had that people put Dragon on the work computer connected Sydney office and dictated all night. It's easier to create by dictating than it is to type it out at least with dictation I can go back and edit later but it's all written in one text file in 90 percent of it is that.

Chapter 10

On the Future of GPU Computing

In 2006, NVidia released CUDA to the general public—a milestone representing the moment GPU computing reached the critical mass necessary for uptake into widespread and prolonged use. Since then attempts have been made to leverage the GPU for nearly ever field of scientific research.

The question, however, is frequently asked: is GPU computing a buzzword that will die out, or is it here to stay?

Fine tuning of kernels is the way of the past. Anyone fine tuning kernels will be operating at a lower level, attempting to optimize library routines underlying large scale codes. The fine tuning can ultimately be replaced by auto-tuned kernels (e.g., see work in ViennaCL to auto-tune similar to libATLAS).

Already this is seen in attempts to port applications to the GPU in so-called "minimally invasive" fashion.

So the short answer is: yes. And it is from this assumption that we proceed with our efforts to target the GPU with RBF-FD. However, we have made a few predictions on the future of GPUs and these predictions guided our efforts to port RBFs onto GPUs.

First, we are proponents for OpenCL over CUDA. Although CUDA has a large following due to its early release, we believe in functional portability of code enabled by OpenCL over the performance provided by CUDA.

10.1 OpenCL vs CUDA

We support opencl. The difference lying in the end-goal of supporting a wide range of hardware with a compiler as support for porting.

10.2 Pragmas

10.2.1 PGI Accelerator Pragmas

During a 2009 summer internship at NCAR, we investigated the use of Portland Group Inc (PGI) Accelerator pragmas to accelerate a subgrid physics module for resolving clouds in the Community Climate Simulation Models (CCSM). The results were sketchy(*) for numerous reasons. First, as early adopters, we were attempting to

10.2.2 OpenACC

10.3 MPI and GPU Computing

CUDA 5 introduced support for MPI directly from the GPU. But we are proponents for OpenCL, not CUDA. The difference lying in the

10.4 OpenMP

The latest version of OpenMP (v4.0) introduces pragmas for offloading computation to accelerators. These pragmas will function similarly to pragmas from PGI, OpenACC, and the Intel MIC.

Bibliography

[1] AccelerEyes. *Jacket User Guide - The GPU Engine for MATLAB*, 1.2.1 edition, November 2009. 6

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. 6, 29

[3] Sylvie Barak. Gpu technology key to exascale says nvidia. http://www.eetimes.com/electronics-news/4230659/GPU-technology-key-to-exascale-says-Nvidia, November 2011. 5

[4] 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. 5

[5] Nathan Bell and Michael Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis - SC '09*, (1):1, 2009. 44

[6] Evan F. Bollig, Natasha Flyer, and Gordon Erlebacher. Solution to pdes using radial basis function finite-differences (rbf-fd) on multiple gpus. *Journal of Computational Physics*, (0):–, 2012. 38

[7] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and Representation of 3D Objects with Radial Basis Functions. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76, New York, NY, USA, 2001. ACM. 18

[8] J. C. Carr, R. K. Beatson, B. C. McCallum, W. R. Fright, T. J. McLennan, and T. J. Mitchell. Smooth Surface Reconstruction from Noisy Range Data. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 119–ff, New York, NY, USA, 2003. ACM. 18

[9] Tom Cecil, Jianliang Qian, and Stanley Osher. Numerical Methods for High Dimensional Hamilton-Jacobi Equations Using Radial Basis Functions. *JOURNAL OF COMPUTATIONAL PHYSICS*, 196:327–347, 2004. 5, 15, 25, 26

[10] G Chandhini and Y Sanyasiraju. Local RBF-FD Solutions for Steady Convection-Diffusion Problems. *International Journal for Numerical Methods in Engineering*, 72(3), 2007. 15, 25, 26

[11] P P Chinchapatnam, K Djidjeli, P B Nair, and M Tan. A compact RBF-FD based meshless method for the incompressible Navier–Stokes equations. *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment*, 223(3):275–290, March 2009. 25

[12] CD Correa, D Silver, and M Chen. Volume Deformation via Scattered Data Interpolation. *Proceedings of Eurographics/IEEE VGTC Workshop on Volume Graphics*, pages 91–98, 2007. 15

[13] Salvatore Cuomo, Ardelio Gallettiy, Giulio Giuntay, and Alfredo Staracey. Surface reconstruction from scattered point via rbf interpolation on gpu. *CoRR*, abs/1305.5179, 2013. 6

[14] Oleg Davydov and Dang Thi Oanh. On the optimal shape parameter for gaussian radial basis function finite difference approximation of the poisson equation. *Computers Mathematics with Applications*, 62(5):2143 – 2161, 2011. 15, 28, 41

[15] E Divo and AJ Kassab. An Efficient Localized Radial Basis Function Meshless Method for Fluid Flow and Conjugate Heat Transfer. *Journal of Heat Transfer*, 129:124, 2007. 5, 15, 22

[16] Qiang Du, Max D. Gunzburger, and Lili Ju. Voronoi-based Finite Volume Methods, Optimal Voronoi Meshes, and PDEs on the Sphere. *Computer Methods in Applied Mechanics and Engineering*, 192:3933–3957, August 2003. 35

[17] G. E. Fasshauer. RBF Collocation Methods and Pseudospectral Methods. Technical report, 2006. 10, 13, 15, 21

[18] Gregory E. Fasshauer. Solving Partial Differential Equations by Collocation with Radial Basis Functions. In *In: Surface Fitting and Multiresolution Methods A. Le M'ehaut'e, C. Rabut and L.L. Schumaker (eds.), Vanderbilt*, pages 131–138. University Press, 1997. 12, 13, 15, 19, 20

[19] Gregory E. Fasshauer. *Meshfree Approximation Methods with MATLAB*, volume 6 of *Interdisciplinary Mathematical Sciences*. World Scientific Publishing Co. Pte. Ltd., 5 Toh Tuck Link, Singapore 596224, 2007. 10, 11, 13, 14, 15, 16, 17, 19, 21, 27, 30, 37, 38

[20] Gregory E. Fasshauer and Jack G. Zhang. On choosing "optimal" shape parameters for rbf approximation. *Numerical Algorithms*, 45(1-4):345–368, 2007. 14

[21] A. I. Fedoseyev, M. J. Friedman, and E. J. Kansa. Improved Multiquadric Method for Elliptic Partial Differential Equations via PDE Collocation on the Boundary. *Computers & Mathematics with Applications*, 43(3-5):439 – 455, 2002. 13, 15, 20

[22] Natasha Flyer and Bengt Fornberg. Radial basis functions: Developments and applications to planetary scale flows. *Computers & Fluids*, 46(1):23–32, July 2011. 4, 9, 14, 15

[23] Natasha Flyer and Erik Lehto. Rotational transport on a sphere: Local node refinement with radial basis functions. *Journal of Computational Physics*, 229(6):1954–1969, March 2010. 4, 9, 15

[24] Natasha Flyer, Erik Lehto, Sebastien Blaise, Grady B. Wright, and Amik St-Cyr. Rbf-generated finite differences for nonlinear transport on a sphere: shallow water simulations. *Submitted to Elsevier*, pages 1–29, 2011. 15, 25, 26, 31, 32, 33, 37, 38, 41, 43

[25] Natasha Flyer and Grady B. Wright. Transport schemes on a sphere using radial basis functions. *Journal of Computational Physics*, 226(1):1059 – 1084, 2007. 4, 9, 14, 15, 25

[26] Natasha Flyer and Grady B. Wright. A Radial Basis Function Method for the Shallow Water Equations on a Sphere. In *Proc. R. Soc. A*, volume 465, pages 1949–1976, December 2009. 4, 9, 14, 15, 31

[27] B Fornberg, T Driscoll, G Wright, and R Charles. Observations on the behavior of radial basis function approximations near boundaries. *Computers & Mathematics with Applications*, 43(3-5):473–490, February 2002. 26

[28] B Fornberg, N Flyer, and JM Russell. Comparisons Between Pseudospectral and Radial Basis Function Derivative Approximations. *IMA Journal of Numerical Analysis*, 2009. 34

[29] B. Fornberg and G. Wright. Stable computation of multiquadric interpolants for all values of the shape parameter. *Computers & Mathematics with Applications*, 48(5-6):853 – 867, 2004. 10, 15, 23

[30] Bengt Fornberg, Elisabeth Larsson, and Natasha Flyer. Stable Computations with Gaussian Radial Basis Functions. *SIAM J. on Scientific Computing*, 33(2):869—-892, 2011. 10, 15, 23, 28

[31] Bengt Fornberg and Erik Lehto. Stabilization of RBF-generated finite difference methods for convective PDEs. *Journal of Computational Physics*, 230(6):2270–2285, March 2011. 4, 10, 15, 25, 26, 32, 33, 34, 37, 38

[32] Bengt Fornberg, Erik Lehto, and Collin Powell. Stable calculation of gaussian-based rbf-fd stencils. Technical Report 2012-018, Uppsala University, Numerical Analysis, 2012. 23

[33] Bengt Fornberg and Cécile Piret. A Stable Algorithm for Flat Radial Basis Functions on a Sphere. *SIAM Journal on Scientific Computing*, 30(1):60–80, 2007. 10, 23, 34, 35

[34] Bengt Fornberg and Cécile Piret. On Choosing a Radial Basis Function and a Shape Parameter when Solving a Convective PDE on a Sphere. *Journal of Computational Physics*, 227(5):2758 – 2780, 2008. 10, 11, 15

[35] Richard Franke. Scattered Data Interpolation: Tests of Some Method. *Mathematics of Computation*, 38(157):181–200, 1982. 10, 15

[36] Prashant Goswami, Philipp Schlegel, Barbara Solenthaler, and Renato Pajarola. Interactive sph simulation and rendering on the gpu. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 55–64, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association. 38

[37] R Hardy. Multiquadratic Equations of Topography and Other Irregular Surfaces. *J. Geophysical Research*, (76):1–905, 1971. 4, 9, 10

[38] Y. C. Hon and R. Schaback. On unsymmetric collocation by radial basis functions. *Appl. Math. Comput.*, 119(2-3):177–186, 2001. 13, 15, 17

[39] Yiu-Chung Hon, Kwok Fai Cheung, Xian-Zhong Mao, and Edward J. Kansa. A Multiquadric Solution for the Shallow Water Equations. *ASCE J. Hydraulic Engineering*, 125:524–533, 1999. 14, 15

[40] A. Iske. *Multiresolution Methods in Scattered Data Modeling*. Springer, 2004. 10, 15, 16, 17

[41] L. Ivan, H. De Sterck, S. A. Northrup, and C. P. T. Groth. Three-Dimensional MHD on Cubed-Sphere Grids: Parallel Solution-Adaptive Simulation Framework. In *20th AIAA CFD Conference*, number 3382, pages 1325–1342, 2011. 48

[42] 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):127–145, 1990. 4, 9, 11, 12, 13, 15, 19

[43] 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):147–161, 1990. 4, 9, 11, 12, 13, 15, 19

[44] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999. 48

[45] G Kosec and B Šarler. Solution of thermo-fluid problems by collocation with local pressure correction. *International Journal of Numerical Methods for Heat & Fluid Flow*, 18, 2008. 5, 15

[46] Elisabeth Larsson and Bengt Fornberg. A Numerical Study of some Radial Basis Function based Solution Methods for Elliptic PDEs. *Comput. Math. Appl*, 46:891–902, 2003. 10, 13, 15, 19, 20, 23

[47] Yuxu Lin, Chun Chen, Mingli Song, and Zicheng Liu. Dual-RBF based surface reconstruction. *Vis Comput*, 25(5-7):599–607, May 2009. 15

[48] X. Liu, G.R. Liu, K. Tai, and K.Y. Lam. Radial point interpolation collocation method (RPICM) for partial differential equations. *Computers & Mathematics with Applications*, 50(8-9):1425 – 1442, 2005. 15

[49] Bryan S. Morse, Terry S. Yoo, Penny Rheingans, David T. Chen, and K. R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA, 2005. ACM. 14, 15

[50] C.T. Mouat and R.K. Beatson. RBF Collocation. Technical Report UCDMS2002/3, Department of Mathematics & Statistics, University of Canterbury, New Zealand, February 2002. 15, 16, 19

[51] Jia Pan and Dinesh Manocha. Fast GPU-based Locality Sensitive Hashing for K-Nearest Neighbor Computation. *Proceedings of the 19th ACM SIGSPATIAL GIS '11*, 2011. 38

[52] DA Randall, TD Ringler, and RP Heikes. Climate modeling with spherical geodesic grids. *Computing in Science & Engineering*, pages 32–41, 2002. 48

[53] Karl Rupp, Florian Rudolf, and Josef Weinbub. ViennaCL-A High Level Linear Algebra Library for GPUs and Multi-Core CPUs. In *Proc. GPUScA*, pages 51—56, 2010. 60

[54] Karl Rupp, Josef Weinbub, and Florian Rudolf. Automatic Performance Optimization in ViennaCL for GPUs Categories and Subject Descriptors. In *Proceedings of the 9th Workshop on Parallel/High-Performance Object-Oriented Scientific Computing*, pages 6:1–6:6. ACM, 2010. 60

[55] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial Mathematics, second edition, 2003. 47

[56] Robert Schaback. Multivariate Interpolation and Approximation by Translates of a Basis Function. In C.K. Chui and L.L. Schumaker, editors, *Approximaton Theory VIII–Vol. 1: Approximation and Interpolation*, pages 491–514. World Scientific Publishing Co., Inc, 1995. 10, 22, 25

[57] J. Schmidt, C. Piret, B.J. Kadlec, D.A. Yuen, E. Sevre, N. Zhang, and Y. Liu. Simulating Tsunami Shallow-Water Equations with Graphics Accelerated Hardware (GPU) and Radial Basis Functions (RBF). In *South China Sea Tsunami Workshop*, 2008. 6

[58] J. Schmidt, C. Piret, N. Zhang, B.J. Kadlec, D.A. Yuen, Y. Liu, G.B. Wright, and E. Sevre. Modeling of Tsunami Waves and Atmospheric Swirling Flows with Graphics Processing Unit (GPU) and Radial Basis Functions (RBF). *Concurrency and Computat.: Pract. Exper.*, 2009. 6, 15

[59] C. Shu, H. Ding, and K. S. Yeo. Local radial basis function-based differential quadrature method and its application to solve two-dimensional incompressible Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 192(7-8):941 – 954, 2003. 5, 15, 25, 26

[60] C Shu, H Ding, and N Zhao. Numerical Comparison of Least Square-Based Finite-Difference (LSFD) and Radial Basis Function-Based Finite-Difference (RBFFD) Methods. *Computers and Mathematics with Applications*, 51(8):1297–1310, 2006. 15, 25

[61] Ian H. Sloan and Robert S. Womersley. Extremal systems of points and numerical integration on the sphere. *Adv. Comput. Math*, 21:107–125, 2003. 34

[62] D Stevens, H Power, M Lees, and H Morvan. The use of PDE centres in the local RBF Hermitian method for 3D convective-diffusion problems. *Journal of Computational Physics*, 2009. 13, 15, 16, 22, 25

[63] David Stevens, Henry Power, Michael Lees, and Herve Morvan. A Meshless Solution Technique for the Solution of 3D Unsaturated Zone Problems, Based on Local Hermitian Interpolation with Radial Basis Functions. *Transp Porous Med*, 79(2):149–169, Sep 2008. 15, 16, 22

[64] David Stevens, Henry Power, and Herve Morvan. An order-N complexity meshless algorithm for transport-type PDEs, based on local Hermitian interpolation. *Engineering Analysis with Boundary Elements*, 33(4):425 – 441, 2008. 6, 15, 16, 22

[65] A. I. Tolstykh and D. A. Shirobokov. On using radial basis functions in a "finite difference mode" with applications to elasticity problems. In *Computational Mechanics*, volume 33, pages 68 – 79. Springer, December 2003. 5, 25

[66] A.I. Tolstykh. On using RBF-based differencing formulas for unstructured and mixed structured-unstructured grid calculations. In *Proceedings of the 16 IMACS World Congress, Lausanne*, pages 1–6, 2000. 4, 25

[67] Robert Vertnik and Božidar Šarler. Meshless local radial basis function collocation method for convective-diffusive solid-liquid phase change problems. *International Journal of Numerical Methods for Heat & Fluid Flow*, 16(5):617–640, 2006. 15, 22

[68] B. Šarler and R. Vertnik. Meshfree Explicit Local Radial Basis Function Collocation Method for Diffusion Problems. *Computers and Mathematics with Applications*, 51(8):1269–1282, 2006. 15, 22

[69] Richard Vuduc, James W Demmel, and Katherine A Yelick. Oski: A library of automatically tuned sparse matrix kernels. In *Institute of Physics Publishing*, 2005. 44

[70] J. G. Wang and G. R. Liu. A point interpolation meshless method based on radial basis functions. *Int. J. Numer. Methods Eng.*, 54, 2002. 15

[71] Geoffrey Womeldorff. Spherical Centroidal Voronoi Tessellations: Point Generation and Density Functions Via Images. Master's thesis, Florida State University, 2008. 35, 36

[72] Robert S. Womersley and Ian H Sloan. How good can polynomial interpolation on the sphere be?, 2001. 34, 35

[73] Grady Wright and Bengt Fornberg. Scattered node mehrstellenverfahren-type formulas generated from radial basis functions. In *The International Conference on Computational Methods*, December 15-17 2004. 15, 26

[74] Grady B. Wright. *Radial Basis Function Interpolation: Numerical and Analytical Developments*. PhD thesis, University of Colorado, 2003. 5, 15, 25, 28

[75] Grady B. Wright, Natasha Flyer, and David A. Yuen. A hybrid radial basis function–pseudospectral method for thermal convection in a 3-d spherical shell. *Geochem. Geophys. Geosyst.*, 11(Q07003):18 pp., 2010. 4, 9, 14, 15, 21, 31

[76] Grady B. Wright and Bengt Fornberg. Scattered node compact finite difference-type formulas generated from radial basis functions. *J. Comput. Phys.*, 212(1):99–123, 2006. 15, 16, 25, 26

[77] Z M Wu. Hermite-Birkhoff interpolation of scattered data by radial basis functions. *Approx. Theory Appl*, (8):1–10, 1992. 13

[78] Xuan Yang, Zhixiong Zhang, and Ping Zhou. Local Elastic Registration of Multimodal Medical Image Using Robust Point Matching and Compact Support RBF. In *BMEI '08: Proceedings of the 2008 International Conference on BioMedical Engineering and Informatics*, pages 113–117, Washington, DC, USA, 2008. IEEE Computer Society. 15

[79] Rio Yokota, L.A. Barba, and Matthew G. Knepley. PetRBF — A parallel O(N) algorithm for radial basis function interpolation with Gaussians. *Computer Methods in Applied Mechanics and Engineering*, 199(25-28):1793–1804, May 2010. 6

[80] Hao Zhang and Marc G. Genton. Compactly supported radial basis function kernels, 2004. 15