FLORIDA STATE UNIVERSITY

Solving PDEs on GPUs with the Radial Basis Function Methods

Isaac Lyngaas

April 26, 2015

1 ABSTRACT

The radial basis function-generated finite difference (RBF-FD) method is a method for solving partial differential equations (PDEs) that is very well suited for parallel computation. In this paper, a description of how RBFs are used for solving PDEs including the RBF-FD method will be given. An implementation of the RBF-FD using a single graphics processing unit (GPU) will then be explored. This GPU implementation will be tested with different precision arithmetics to determine how this affects the amount of time until solution and the amount of storage required.

2 INTRODUCTION

In this paper, we will consider the RBF-FD approach for solving PDEs. This approach is a variant of the RBF direct method which is a very accurate method for solving PDEs that runs into issues when solving PDEs with large problem sizes. For large problem sizes, the RBF direct method needs to solve a large dense matrix which has high computational costs and memory requirements as well as stability issues. The (RBF-FD) sacrifices some of the accuracy of the direct method in order to solve the PDE with a sparse matrix rather than a dense one. This change from solving a dense matrix to a sparse matrix alleviates some of the main problems with the direct method. It turns out that this approach is viable due to its susceptibility to being parallelized. This susceptibility will be taken advantage of in attempt to speed up the method using a GPU. In section 3, background information on RBFs that is necessary for a description of the RBF-FD method will be given. In section 4, the RBF-FD method will be detailed. In section 5, a strategy for implementation of the RBF-FD on a GPU will be outlined. Finally in section 6, timing of the GPU implementation with different precision arithmetics will be done for a test two-dimensional domain.

3 RBF Background Information

3.1 Meshfree Method

Many scientific problems can be expressed as PDEs on some spatial domain. In order to solve these PDEs numerically, the domain is normally discretized into points scattered over the domain such as the two-dimensional example in Figure 3.1. In order to solve a PDE, traditional numerical methods such as finite element methods require the creation of a mesh, like the two-dimensional example in Figure 3.2, that connects discretized points in a well defined manner. This mesh creation can sometimes be a very costly computation and can be difficult to program especially for problems with irregular domains and with multiple dimensions. RBF methods are considered because they do not require a well-defined connection of discretized points. For this reason, RBF methods are called meshfree methods.

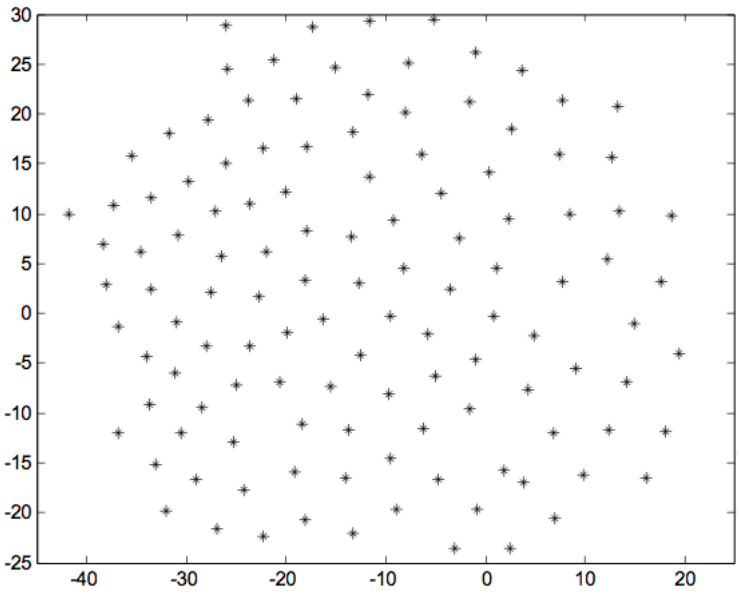

Figure 3.1: Discretized 2-D Domain

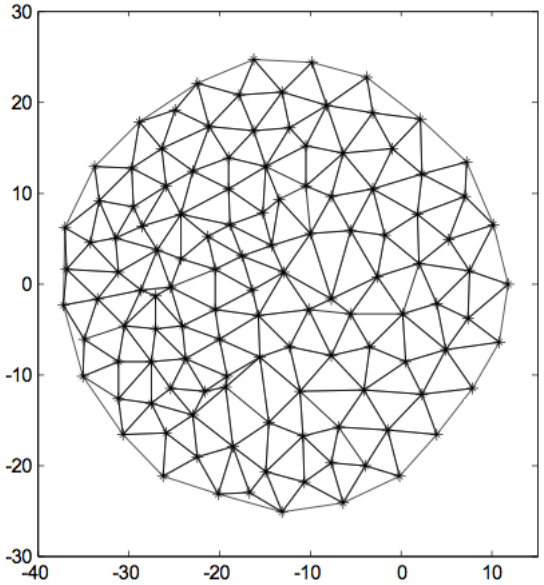

Figure 3.2: 2-D Mesh

3.2 RBF Interpolation

RBFs are real valued functions of the following form

$$\phi(r,\varepsilon)$$

where

$$r = \|\mathbf{x}-\mathbf{y}\|_2 = \sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\ldots+(x_d-y_d)^2}\,,\ \mathbf{x},\mathbf{y}\in\mathbb{R}^d$$

denotes the Euclidean distance between two locations in a domain and ε is a parameter that affects the shape of the RBF. The utility of this shape parameter will not be touched on in this paper but more information can be found in [2]. Two examples of radial basis functions that are commonly used are in Table 3.1.

Table 3.1: Commonly Used RBFs

Name	$\phi(r,\varepsilon)$
Multiquadric	$\sqrt{1+(r\varepsilon)^2}$
Gaussian	$\exp(-(r\varepsilon)^2)$

Before determining how a PDE can be approximated by these RBFs, a function must be shown how it can be interpolated using RBFs. The idea behind RBF interpolation is to take a linear combination of RBFs to make an approximation of the function which in this case will be name u. To do this, an interpolation function will be found for a given finite set of N data points (often called centers in RBF terminology) $\mathbf{x}_1,\ldots,\mathbf{x}_N \in \mathbb{R}^d$ that are scattered over the domain of interest and for which the value of $f(\mathbf{x})$ is known at these centers. An example of how these centers might look like in a two-dimensional domain are the points in Figure 3.1. The RBF interpolant becomes

$$s(\mathbf{x}) = \sum_{j=1}^{N} w_j \phi(\|\mathbf{x}-\mathbf{x}_j\|,\varepsilon) \tag{3.1}$$

where the goal will be to find the weights w_j for $j = 1,\ldots,n$. To find these weights, interpolation conditions are enforced by the following equations

$$s(\mathbf{x}_i) = f(\mathbf{x}_i) \; for \; i = 1,\ldots,N.$$

This results in the following NxN linear system

$$H\mathbf{w} = \mathbf{f} \tag{3.2}$$

where $H_{ij} = \phi(\|\mathbf{x}_i-\mathbf{x}_j\|,\varepsilon)$, for $i,j = 1,\ldots,N$ and $f_i = f(\mathbf{x}_i)$, for $i = 1,\ldots,N$. This system is then solved to find the weights w_i, for $i = 1,\ldots,N$ that makes up the approximation function

$$\hat{u}(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi(\|\mathbf{x}-\mathbf{x}_i\|,\varepsilon) \tag{3.3}$$

to the problem $u = f$.

3.3 RBF Derivatives

Now that it has been shown how an interpolation function can be found for $u = f$, one more thing that is needed in order to describe how PDEs can be solved with RBFs is how to find the derivatives of RBFs. Since we are normally interested in solving elliptic PDEs, RBF derivatives up to the second derivative will be found explicitly. The RBFs being used are sufficiently differentiable and can be written in the form $\phi(r(\mathbf{x}))$ where $r(\mathbf{x}) = \|\mathbf{x}\|$, so the chain rule can be used to find that

$$\frac{\partial \phi}{\partial x_i} = \frac{d\phi}{dx_i}\frac{\partial r}{\partial x_i}$$

and

$$\frac{\partial^2 \phi}{\partial x_i^2} = \frac{d\phi}{dr}\frac{\partial^2 r}{\partial x_i^2} + \frac{d^2\phi}{dr^2}\left(\frac{\partial r}{\partial x_i}\right)^2$$

where

$$\frac{\partial r}{\partial x_i} = \frac{x_i}{r},$$

$$\frac{\partial^2 r}{\partial x_i^2} = \frac{1 - (\frac{dr}{dx_i})^2}{r}$$

and $\frac{d\phi}{dr}$ and $\frac{d^2\phi}{dr^2}$ can easily be determined based on the RBF used. For more on the derivation of these equations refer to [7]. These derivatives will be necessary for use in the methods of the following section.

4 RBF Methods for PDEs

Using the RBF interpolation and RBF derivatives from the previous section, methods for solving PDEs can now be derived. Two methods for solving PDEs with RBFs will be introduced. One is a dense matrix approach called the RBF direct method and the other is a sparse matrix approach called the RBF-FD method.

4.1 RBF Direct Method

The RBF direct method takes the approach of modifying the system from Equation 3.2 in order to solve the elliptic PDE $u'' = f$ that we are interested in solving. In order to do this, our interpolation problem from Equation 3.1 is modified to become

$$\frac{\partial^2}{\partial \mathbf{x}_i^2} s(\mathbf{x}) = \sum_{j=1}^{N} w_j \frac{\partial^2}{\partial (x_i^1)^2} \phi(\|\mathbf{x} - \mathbf{x}_j\|, \varepsilon) + \cdots + \sum_{j=1}^{N} w_j \frac{\partial^2}{\partial (x_i^d)^2} \phi(\|\mathbf{x} - \mathbf{x}_j\|, \varepsilon),\ \mathbf{x} \in \mathbb{R}^d. \tag{4.1}$$

A system of equations is then set up for the centers of the domain being considered $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$ where the equations are of the form

$$\sum_{j=1}^{N} w_j \left[\frac{\partial^2}{\partial (x_i^1)^2} \phi(\|\mathbf{x}_i - \mathbf{x}_j\|, \varepsilon) + \cdots + \frac{\partial^2}{\partial (x_i^d)^2} \phi(\|\mathbf{x}_i - \mathbf{x}_j\|, \varepsilon) \right] = f(\mathbf{x}_i) \tag{4.2}$$

if x_i is in the interior of the domain,

$$\sum_{j=1}^{N} w_j \left[\frac{\partial}{\partial x_i^1} \phi(\|\mathbf{x}_i - \mathbf{x}_j\|, \varepsilon) + \cdots + \frac{\partial}{\partial x_i^d} \phi(\|\mathbf{x}_i - \mathbf{x}_j\|, \varepsilon) \right] = g(\mathbf{x}_i) \tag{4.3}$$

if x_i is on the boundary of the domain and has the Neumann boundary condition g, or

$$\sum_{j=1}^{N} w_j \phi(\|\mathbf{x}_i - \mathbf{x}_j\|, \varepsilon) = h(\mathbf{x}_i) \tag{4.4}$$

if x_i is on the boundary of the domain and has the Dirichlet boundary condition h. Again,

an NxN linear system has been formed that can be solved to find $w_i \; for \; i = 1,\ldots,N$ which forms the approximation

$$\hat{u}(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi(\|\mathbf{x}-\mathbf{x}_i\|, \varepsilon) \tag{4.5}$$

to the problem $u'' = f$.

The RBF direct method is a very accurate method for solving PDEs, however the problem with using it is that it results in the need to solve a dense system of linear equations. If there are a large amount of centers, this method can be very computationally intensive and can potentially have stability problems. Refer to [2] for more information on the stability problems of the RBF direct method. In addition, the issue of solving the dense matrix becomes a huge problem in the case of a time dependent PDE where a new matrix would have to be solved at each time step. Due to these problems the need for the sparse matrix approach of the RBF-FD method becomes evident.

4.2 RBF-FD Method

Rather than finding the interpolating function that approximates a PDE, the RBF-FD method takes the finite difference approach to solving PDEs with RBFs. Considering that the problem we are interested in solving is $u'' = f$ for the center points $\mathbf{x}_1,\ldots,\mathbf{x}_N \in \mathbb{R}^2$, at each center $\mathbf{x}_c$ we want to find a linear combination of functions values u_k at the neighboring n nearest locations to $\mathbf{x}_c$ that will form an approximation to $u''(\mathbf{x}_c)$. So the differentiation weights a_k need to be found that satisfy $\sum_{k=1}^{n} a_k u_k = u''(\mathbf{x}_c)$. These differentiation weights are calculated by enforcing that this linear combination should be exact for RBFs centered at each center location. This leads to the following system, where $\mathbf{x}_1,\ldots,\mathbf{x}_n$ denote the center points that are the n nearest in Euclidean distance to $\mathbf{x}_c$, that can be solved to find the RBF-FD local stencil weights for $\mathbf{x}_c$.

$$\begin{pmatrix} \phi(\|\mathbf{x}_1-\mathbf{x}_1\|,\varepsilon) & \phi(\|\mathbf{x}_1-\mathbf{x}_2\|,\varepsilon) & \cdots & \phi(\|\mathbf{x}_1-\mathbf{x}_n\|,\varepsilon) \\ \cdots & \ddots & \vdots & \\ \phi(\|\mathbf{x}_n-\mathbf{x}_1\|,\varepsilon) & \phi(\|\mathbf{x}_n-\mathbf{x}_2\|,\varepsilon) & \cdots & \phi(\|\mathbf{x}_n-\mathbf{x}_n\|,\varepsilon) \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \frac{\partial^2}{\partial x_1^2}\phi(\|\mathbf{x}_c-\mathbf{x}_1\|,\varepsilon) + \frac{\partial^2}{\partial x_2^2}\phi(\|\mathbf{x}_c-\mathbf{x}_1\|,\varepsilon) \\ \vdots \\ \frac{\partial^2}{\partial x_1^2}\phi(\|\mathbf{x}_c-\mathbf{x}_n\|,\varepsilon) + \frac{\partial^2}{\partial x_2^2}\phi(\|\mathbf{x}_c-\mathbf{x}_n\|,\varepsilon) \end{pmatrix}$$

For more information on the derivation of these RBF-FD local stencil systems refer to [4]. Solving one of these systems results in one row of the RBF-FD global stencil matrix H in Equation 4.6. In total, N systems will be solved to find a local stencil for each center point $\mathbf{x}_1,\ldots,\mathbf{x}_N \in \mathbb{R}^d$. These RBF-FD local stencil systems can all be solved simultaneously, so it is ultimately the thing that is attempted be computed on a GPU. Given that all of these local stencil weights have been found for each center point, the global stencil system

$$H\hat{\mathbf{u}} = \mathbf{f} \tag{4.6}$$

can be solved to find the solution to the PDE of interest. In this linear system, $\hat{u}_i$ is the approximate solution to the PDE $u'' = f$ at $\mathbf{x}_i$ for $i = 1,\ldots,N$ and $f_i = f(\mathbf{x}_i)$, for $i = 1,\ldots,N$. H is an $N \times N$ sparse matrix made up of n elements per row. Each row of H is made up of the RBF-FD local stencil weights found from the previous step where they are placed in the

correct position based on the location of the n neighboring points in $\mathbf{x}_1, \ldots, \mathbf{x}_N$. For an example of how these local stencils are placed into the global stencil matrix consider the case where $N = 101$ and 5 local stencil weights are found for the center $\mathbf{x}_1$ using the 5 nearest points. Suppose the nearest points to $\mathbf{x}_1$ in order are $\{\mathbf{x}_1, \mathbf{x}_{21}, \mathbf{x}_8, \mathbf{x}_{18}, \mathbf{x}_2\}$ then the first row of H will be filled with the local stencil weights $a_1, \ldots, a_5$ placed accordingly into the positions $\{1, 21, 8, 18, 2\}$ of the row and zeros will be placed everywhere else.

The linear system in 4.6 will benefit from using a sparse linear system solver as a large portion of the system is zero. Thus, the computational advantage of the RBF-FD over the RBF direct method has been found. By setting n the local stencil size smaller, the matrix H in Equation 4.6 becomes more sparse and thus easier easier to solve and the N RBF-FD weight systems become smaller making them easier to solve, however according to [4] this also decreases the accuracy of the method.

5 GPU Parallelization

In [3] and [6], multi-GPU implementations of the RBF-FD are used to solve time dependent PDEs, however these implementations are focused more on the parallelization of the sparse linear system from Equation 4.6 rather than parallelizing the stencil weight calculation. The justification behind this is that they are solving PDEs on a stationary domain where the stencil weight is a one time calculation. However, it is conceivable for a PDE to have a varying domain for an example of this refer to [1]. In the case of a varying domain, it would be very beneficial to parallelize the RBF-FD stencil weight calculation as it would have to been done multiple times. In this section, the focus will be on offloading the calculations of the RBF-FD local stencil weight calculation to a GPU in order to speed up computation.

5.1 Parallelization of RBF-FD Stencil Calculation

The GPU implementation of the RBF-FD stencil calculation used in this paper is dependent on several kernel functions that will operate on all of the data points of a given domain discretization of points. Two of the main kernels were adapted from code written by [5] for a K nearest neighbors algorithm. The first kernel which will be called the global distance kernel takes advantage of shared memory on a GPU to find the pairwise distances between each of the N data points in a given data set. This kernel results in an $N \times N$ matrix that holds distances between each possible combination of data points. The second kernel is one that takes the distance matrix calculated in the global distance kernel and sorts it to find the K nearest data points to each data point and returns the indices to these K nearest data points.

Using the calculations from the first two kernels, two more kernels were made that set up the RBF-FD local stencil systems for all N data points. These kernels benefit from the fact that all of the distance and index computations are already calculated and exist in GPU memory. With the local stencil matrix systems set up, CUBLAS batched matrix kernels are then used to solve the N local stencil systems simultaneously.

For ease of implementation, all GPU kernels were ran so that they did calculations on all of the data points simultaneously. An emphasis of this implementation is on minimizing the amount of time spent transferring data to and from the GPU and maximizing the number of

data points that can be done at one time. This implementation has a potential to use a large amount of the GPU memory if many data points are given, so the operations are ordered in such a way that the amount of GPU memory that is needed at one time is minimized and data on the GPU is is freed when its no longer needed to avoid overallocation of memory on the GPU. The workflow for the GPU implementation can be shown as the following list of tasks given that there are N two-dimensional data points and K nearest neighbors are specified for local stencil weight calculations.

1. Memory copy of data points to GPU
   a) $2 \times N$ matrix to hold x and y coordinates for each data point
2. 3 Global Distance Kernel calls to calculate pairwise distances for all data points
   a) Euclidean Distance
   b) Distance in x dimension
   c) Distance in y dimension
3. Kernel call to find K nearest neighbors
4. Kernel call to calculate RHS vectors for the RBF-FD local stencil systems
5. Kernel call to calculate matrices for the RBF-FD local stencil systems
6. CUBLAS kernel call to calculate LU factorization of each RBF-FD local stencil matrix
7. CUBLAS kernel calls to perform forward and backward triangular solves that result in the local stencil weights of each RBF-FD local stencil system
8. Return local stencil weights and their indices into the global stencil system back to the host

6 Application

In this section, the RBF-FD stencil weight calculation is done for a discretized two-dimensional square domain with evenly spaced points in $(0,2) \times (0,2)$. Timing will be done for the GPU implementation of the stencil weight calculations with single and double precision implementations to see how they compare.

6.1 Timing

Timing for the RBF-FD stencil weight calculation is tested on a Tesla M2050 with 2.82 GB of global memory and a clock cycle of 1.147 GHz. CUDA code for the implementations were compile using CUDA version 6.5. The plot in Figure 6.1 shows timings with a varying number of data points and the local stencil size fixed at 16. One thing to notice from this plot is that the number of data points for the single precision implementation stops at approximately 13,000 and the number of data points for the double precision implementation stops at approximately 9,000. The number of data points do not go past these amounts due to the GPU running out of memory. As can be seen by the plot, very little difference in timings is noticed between the single and double precision implementations. This makes us believe that it is not detrimental to use double precision numbers if the number of data points is

less than 9,000 and the stencil size is small. Figure 6.2 shows timings where the number of data points is fixed at 6,400 and the local stencil size varies. From this plot, it is seen that as the stencil size begins to increase there is an increasing advantage to using single precision over double precision.

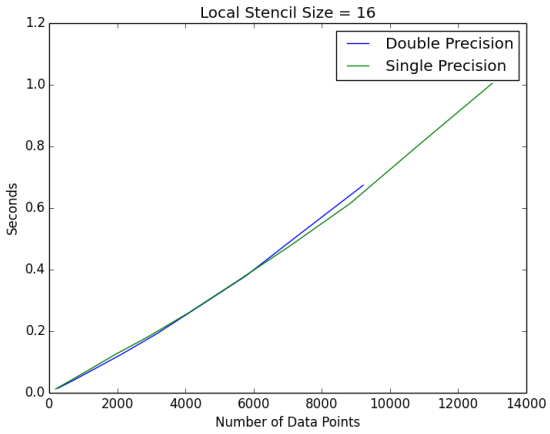

Figure 6.1: RBF-FD timings on a GPU with varying number of data points

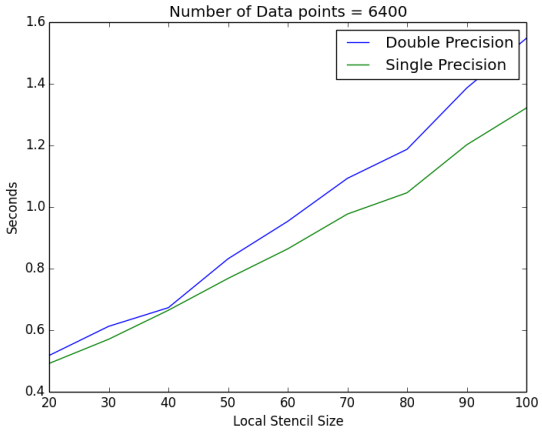

Figure 6.2: RBF-FD timings on a GPU with varying stencil size

7 Conclusion

In this paper, a GPU implementation was used to calculate the local stencil weights for the RBF-FD method. Through time testing, it is seen that using different precision accuracy data types does not have much effect on the time it takes for the GPU implementation with a small local stencil size. However as stencil size is increased, it becomes beneficial to use single precision data types to speed up the calculation. Also, it becomes necessary to use single precision data types if the number of data points becomes too large. Unfortunately not all of the necessary information is available at this point in order to completely write off using double precision data types. If it were found that the stencil weights from the single precision implementation were not accurate enough then the double precision implementation would need to be used. In future work, the RBF-FD implementations with different precision data types and stencil sizes will need to be tested in order to find configurations

that will get good speed while obtaining adequate accuracy needed from the method. Other future work would be to devise parallelization strategies for a multi-CPU implementation in order to have a comparison with this GPU implementation.

REFERENCES

[1] Antonios Armaou and Panagiotis D Christofides. Robust control of parabolic pde systems with time-dependent spatial domains. *Automatica*, 37(1):61–69, 2001.

[2] Elisabeth Larsson Bengt Fornberg and Natasha Flyer. Stable computations and gaussian radial basis functions. *SIAM Journal on Scientific Computing*, 33(2):869–892, 2011.

[3] 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*, 231(21):7133–7151, August 2012.

[4] Natasha Flyer, Grady B Wright, and Bengt Fornberg. Radial basis function-generated finite differences: A mesh-free method for computational geosciences. *Handbook of Geomathematics. Springer, Berlin*, 2014.

[5] Vincent Garcia, Eric Debreuve, and Michel Barlaud. Fast k nearest neighbor search using gpu. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–6. IEEE, 2008.

[6] G Kosec and P Zinterhof. Local strong form meshless method on multiple graphics processing units. *CMES-Comp Model Eng*, 91:377–396, 2013.

[7] Scott A Sarra and Edward J Kansa. Multiquadric radial basis function approximation methods for the numerical solution of partial differential equations. *Advances in Computational Mechanics*, 2:2, 2009.