Complete Domain Decomposition
A new approach to communication avoidance when solving boundary value problems in a parallel environment

Yoav Segev
Department of Computer Science
University of Maryland, College Park
segev@cs.umd.edu

November 30, 2015

Abstract

As the need for parallel computation for solving partial differential equations increases, so does the communication overhead, caused by the massage passing mechanism used in the parallel computation. In this paper, I suggest a domain decomposition style approach for parallel computation of elliptic partial differential equations that avoids nearly all communication between the computing nodes. The test results show the correctness of this quickly converging method for a one-dimensional problem. In the last sections I discuss ideas to extend this approach to two or three dimensions

1 Introduction

Solving partial differential equations has always been an important aspect of almost every scientific or engineering process. Since most differential equations do not have analytic solutions, our only resort is to solve them numerically. This entails a discretization, and solving a discrete problem that represents the original equation up to certain discretization error. For proper discretization process, this error gets smaller as the size of the discrete problem gets larger, by either increasing the number of discrete data points, increasing the dimension

of the finite dimensional functional space, or similar. Therefore, in order to get a result as accurate as possible, one need to solve problems of a constantly increasing size.

In order to solve these problems time efficiently, one needs to utilize more and more computing resources. This need for increasing computing power leads to the use of large-scale parallel computers. Since large-scale computers cannot use shared memory there is a need to communicate results from one computing node to another.

The division of work between the computing units is done by dividing the problem into a number of smaller sub-problems, usually by dividing the physical domain of the problem, and letting each processing unit solve the sub-problem assigned to it. Since the discrete problem usually relates values at certain grid points with its neighbors, for each computing unit to solve its assigned sub-problem, it needs to know the values at grid points adjacent to the sub-problems boundary. This means that each computing unit needs to communicate the values it computed to all of the computing units that compute neighboring subdomains. Also, as most current methods for solving partial differential equations, such as the Conjugate Gradient Method (CG), the Minimal Residual method (MinRes), Generalized Minimal Residual method (GMRES), just to name a few, are iterative methods, such that each iteration include a certain local part, as well as a global calculation that requires some knowledge of the entire problem, all computing units must communicate certain values to a central unit, which performs the calculation, and communicates the computation result back to the computing units.

As the number of computing units increases, for fixed problem size, the number of subdomains increase; hence the area of each subdomain decreases, and the number of boundary grid points increases. This translates to a decrease in the amount of work that each computing unit needs to do, and an increase in the amount of data it needs to communicate to other units. In other words, if more resources are utilized in the computation, the relative cost of the communication increases.

There are several known ideas to tackle this problem, either hiding the communication by approximating the communicated values, or decomposing the physical domain in a way that minimizes the amount of communication needed. I will focus on the latter ones.

2 Domain Decomposition Methods

Domain decomposition is a class of methods that apply the divide and conquer technique for solving partial differential equations in general, and specifically in

this context, boundary value problems. Chan and Mathew thoroughly survey many of these methods in [1].

2.1 Overlapping Subdomain Algorithms

These algorithms construct a decomposition of the domain Ω into k overlapping subdomains: $\Omega_1, \Omega_2, \ldots, \Omega_k$, usually by extending a non-overlapping decomposition to include all points up to a certain distance from a point within the non overlapping subdomain. Restriction matrices $\{R_i\}$, extension matrices $\{R_i^T\}$ and local matrices $\{A_i\}$ map data between the complete domain Ω and subdomains $\{\Omega_i\}$.

Let A be an $n \times n$ matrix representing the discrete problem on the entire domain Ω. Than for each $i = 1, \ldots, k$ define R_i as the matrix which entries are all zeros and ones, that restricts a vector of size n to a vector of size n_i by choosing only indices corresponding to interior points of Ω_i. Its transpose represents the extension matrix, as it extends vectors of size n_i to a vector of size n by putting zeros in all entries corresponding to points not in Ω_i. Lastly, define the local matrix $A_i = R_i A R_i^T$, the local stiffness matrix for subdomain Ω_i. Using these matrices we can define a preconditioner to the problem $Au = f$ by

$$M^{-1} = \sum_{i=1}^{k} R_i^T A_i^{-1} R_i$$

When this method is used as a preconditioner in conjunction with Krylov subspace method, it is known [2, 3] that the convergence rate deteriorates as the number of subdomains increases. This can be adjusted by adding a global coarse grid operator, which leads to a condition number bounded independently of either the mesh size or the number of subdomains.

2.2 Non-Overlapping Subdomain Algorithms

The main idea in these methods is to decompose the matrix A into four submatrices $A_{II}, A_{IB}, A_{BI}, A_{BB}$, where A_{II} corresponds to entries in A that deal with internal points of the subdomains. A_{BB} corresponds only to points on the boundary between subdomains, and A_{IB}, A_{BI} corresponds to realtions between boundary points and internal points. This way the equation $Au = f$ can be written as a block matrix equation:

$$\begin{bmatrix} A_{II} & A_{IB} \\ A_{BI} & A_{BB} \end{bmatrix} \begin{bmatrix} u_I \\ u_B \end{bmatrix} = \begin{bmatrix} f_I \\ f_B \end{bmatrix}$$

Formal block factorization of A gives:

$$A = \begin{bmatrix} A_{II} & A_{IB} \\ A_{IB}^T & A_{BB} \end{bmatrix} = \begin{bmatrix} I & 0 \\ A_{IB}^T A_{II}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{II} & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & A_{II}^{-1} A_{IB} \\ 0 & I \end{bmatrix}$$

One can easily note that A_{II} is a block-diagonal matrix, where each block is concerned with a single subdomain. It is a block diagonal, since internal points of different subdomain do not affect one another in the solving process. This means that basically one can solve for each subdomain internal points on a different computing node, without any communication needed. Still, though, there is a need for communication for the boundary points, and their relation to adjacent internal points, and that is where the solution with S is involved in the above block matrix representation. The efficiency of this task, and hence the efficiency of the entire computation relies greatly on finding a good preconditioner for the Schur complement S.

3 One-dimensional Complete Domain Decomposition method

The idea behind this method is to extend the idea of domain decomposition to a case where absolutely no communication is needed between computing nodes for each of the solver iterations. The basic idea relies on the fact that the sub-problem presented on each subdomain is actually the same as that of the original problem on the entire domain, so I tried to solve each of them independently using the same method that one might want to use to solve the original problem. There is still one problem though, and that is the fact that for solving a boundary value problem, one must have correct boundary conditions, but the problem statement only gives boundary conditions for $\partial\Omega$, and not for $\partial\Omega_i, \forall i = 1, \ldots, k$. In order to solve the local subdomain problem, one must find proper boundary conditions. To do so, I suggest to solve on a coarse grid consisting of only boundary points of the subdomains.

As an example, assume we want to solve an equation on the interval $[0, 1]$ given certain Dirichlet boundary conditions $u(0) = u_0$ and $u(1) = u_1$. We first discretize the problem to grid points $x_i = ih$ for some mesh width h. Then we divide the problem into subdomains: $\Omega_1 = [0, H], \ldots, \Omega_i = [(i-1)H, iH], \ldots, \Omega_k = [(k-1)H, 1]$ where $H = 1/k$ and k is the number of subdomains. In order to find the boundary conditions for each subdomain, I start by solving the same equation on the coarse grid defined by $\hat{x}_i = iH$, and use the values I get $\hat{u}_i$ as boundary conditions for the respective subdomain problem, which can be solved now completely independently of the other subdomains, hence implying no communication needed.

The problem that arises from the algorithm as described above is that the new internal boundary conditions $u(\hat{x}_i) = \hat{u}_i$ are not calculated precisely, but up to a discretization error, and therefore force an error on the local subdomain problem. One may argue that this is alright, as the continuous problem can never be solved exactly, but only up to some discretization error, but unfortunately this is not the case, as the coarse grid discretization error is of order $O(H)$ which leads to a total error of the entire solution of order $O(H)$ on top of the much smaller order of $O(h)$ discretization error for the fine grid.

In order to solve this problem, I introduced an iterative process similar to the one used in multigrid methods to reduce the error. To describe it, define the following:

- A_h the matrix representing the fine grid discrete problem.
- $A_{h,i}$ the matrix representing the local problem on subdomain i.
- A_h the matrix representing the coarse grid discrete problem.
- u_h the desired fine grid solution.
- u_H the coarse grid solution.

First solve $A_H u_H = f_H$ for $\hat{u}_H$ to get the coarse grid values to use as subdomain boundary values. Than solve each subdomain j separately by solving $A_{h,j} u_j = f_j$ to get $\hat{u}_{h,j}$ and collect all of them into one vector $\hat{u}_h$. Now define the residual of the process by

$$r_h = f_h - A_h \hat{u}_h$$

Note that the error $e_h = u_h - \hat{u}_h = A_h^{-1} f_h - A_h^{-1} A_h \hat{u}_h = A_h^{-1} r_h$ or in other words $r_h = A_h e_h$, so if we could solve this equation for e_h we could find the exact u_h by $u_h = \hat{u}_h + e_h$. Instead we can approximate it by iteratively solve for the error using the same complete domain decomposition method and get $u_i = u_{i-1} + e_i$, $r_i = f - A u_i$, and solve for e_{i+1}: $r_i = A e_{i+1}$

4 Test Problem

To test this method for correctness I ran it on the one-dimensional Poisson equation: $-u'' = f$ on the domain $\Omega = [0, 1]$. After discretization I got that A_h is a tridiagonal matrix which entries are 2 along the diagonal and -1 along both the subdiagonal and superdiagonal multiplied by a constant $\frac{1}{h^2}$ if finite differences discretization is used, or $\frac{1}{h}$ if finite elements. It is clear to see that $A_H, A_{h,i}$ have the exact same structure only different by the multiplied constant and dimension. In my tests I used discretizations of five different functions for f, described below. For simplicity I used only Dirichlet boundary conditions $u(0) = u_0$ and $u(1) = u_1$.

The test function I used as the right hand side, with respective boundary conditions and exact solution found analytically were:

Table 1: Test Functions

#	f	u(0)	u(1)	u
1	$100 * \cos(\pi x)$	0	0	$\frac{100*(\cos(\pi x)+2x-1)}{\pi^2}$
2	$192 - 384x$	-6	6	$(4x-1)(4x-2)(4x-3)$
3	$900\pi^2 \sin(30\pi x)$	1	1	$\sin(30\pi x) + 1$
4	$-e^x$	1	e	e^x
5	$\frac{\sin(10\pi x)(100\pi^2 x^2-2)}{10\pi x^3} + \frac{2\cos(10\pi x)}{x^2}$	1	0	$\frac{sin(10\pi x)}{10\pi x}$

The underlying algorithm I used to solve this problem with this novel domain decomposition scheme was by Gaussian elimination (or in other words LU factorization of the matrix), since I wasn't testing for performance, only for correctness. The performance advantage of complete communication avoidance is clear.

5 Results

The metrics used to examine the quality of the results were the L_2 and H^1 norms of the error $u - u_h$. For the test functions I used, I know the exact continuous solution u, and the continuous version of the discrete solution u_h is basically a linear interpolation between the calculated values for the grid points. The L_2 norm is defined as follows:

$$||u - u_h||_{L_2} = \sqrt{\int_\Omega |u(x) - u_h(x)|^2 dx}$$

And the H^1 norm:

$$||u - u_h||_{H^1} = \sqrt{\int_\Omega (|u(x) - u_h(x)|^2 + |u'(x) - u_h'(x)|^2) dx}$$

5.1 Non Iterative Method

At first I ran the original method without an iterative process and got the results listed in the following table. The first column is the function serial

number, the second shows the number of grid points used in the fine grid, the third is the error of a numeric solution found using a regular solver, the other columns named "CDD - n" are the errors of the solutions found using the complete domain decomposition method with "n" subdomains. The content of each cell in the table is either the L_2 norm of the error or the H^1 norm of the error (respectively) for that specific run.

Table 2: Noniterative Method - L_2 Errors

#	n	Solution	CDD - 2	CDD - 4	CDD - 8	CDD - 16	CDD - 32
1	64	3.070E-04	3.070E-04	6.440E-02	1.880E-02	4.860E-03	1.230E-03
	256	1.920E-05	1.920E-05	6.430E-02	1.870E-02	4.860E-03	1.220E-03
	1024	1.200E-06	1.200E-06	6.420E-02	1.870E-02	4.860E-03	1.220E-03
2	64	2.950E-15	1.530E-15	2.040E-16	4.350E-16	4.860E-16	8.520E-16
	256	2.110E-14	1.620E-14	1.010E-14	4.230E-15	3.780E-16	8.590E-16
	1024	1.080E-13	6.240E-14	4.870E-14	2.240E-14	1.260E-14	4.790E-15
3	64	1.430E-01	1.430E-01	1.600E+02	1.580E+02	1.580E+02	6.130E-01
	256	8.040E-03	8.040E-03	1.600E+02	1.580E+02	1.580E+02	4.990E-01
	1024	4.990E-04	4.990E-04	1.600E+02	1.580E+02	1.580E+02	4.910E-01
4	64	3.140E-06	2.500E-03	7.600E-04	1.980E-04	5.010E-05	1.260E-05
	256	1.960E-07	2.500E-03	7.590E-04	1.980E-04	5.010E-05	1.260E-05
	1024	1.230E-08	2.500E-03	7.590E-04	1.980E-04	5.010E-05	1.260E-05
5	64	8.850E-02	3.030E-01	2.66	5.000E-01	1.160E-01	9.010E-02
	256	4.420E-02	2.920E-01	2.66	4.930E-01	8.740E-02	4.760E-02
	1024	2.210E-02	2.900E-01	2.66	4.920E-01	7.850E-02	2.820E-02

Table 3: Noniterative Method - H^1 Errors

#	n	Solution	CDD - 2	CDD - 4	CDD - 8	CDD - 16	CDD - 32
1	64	2.280E-03	2.280E-03	4.490E-01	1.230E-01	3.070E-02	7.270E-03
	256	1.430E-04	1.430E-04	4.500E-01	1.240E-01	3.160E-02	7.890E-03
	1024	8.910E-06	8.910E-06	4.500E-01	1.240E-01	3.160E-02	7.950E-03
2	64	3.910E-03	3.910E-03	3.910E-03	3.910E-03	3.910E-03	3.910E-03
	256	2.440E-04	2.440E-04	2.440E-04	2.440E-04	2.440E-04	2.440E-04
	1024	1.530E-05	1.530E-05	1.530E-05	1.530E-05	1.530E-05	1.530E-05
3	64	6.43	6.43	1.120E+03	1.030E+03	1.010E+03	4.940E+01
	256	3.780E-01	3.780E-01	1.120E+03	1.030E+03	1.010E+03	5.330E+01
	1024	2.350E-02	2.350E-02	1.120E+03	1.030E+03	1.010E+03	5.360E+01
4	64	1.850E-05	9.010E-03	2.580E-03	6.600E-04	1.640E-04	4.110E-05
	256	1.150E-06	9.010E-03	2.580E-03	6.650E-04	1.670E-04	4.170E-05
	1024	7.210E-08	9.010E-03	2.580E-03	6.650E-04	1.670E-04	4.190E-05
5	64	9.220E-02	1.04	1.360E+01	5.03	8.230E-01	2.020E-01
	256	4.420E-02	1.04	1.360E+01	5.04	8.410E-01	2.090E-01
	1024	2.210E-02	1.04	1.360E+01	5.05	8.410E-01	2.070E-01

From these results one can read several interesting conclusions. In almost all cases, the error, both L_2 and H^1, depends solely on the number of subdomains,

or in other words it is proportional to the coarse grid width H and is not at all proportional to number of fine grid points, and fine grid width h. This makes sense, since the boundary conditions created by the coarse solver have introduced an error proportional to H, so there was no realistic hope to get anything better than that.

A few exceptions to the above conclusion are (a) the polynomial function got a very good solution in every configuration. That is probably due to the fact that polynomials are very easy to solve accurately enough even with the coarse grid width, so it didn't really matter, and I got good results for every mesh width. And (b) for the two trigonometric functions (labeled 1 and 3) the decomposition to two subdomains produced an accurate result as well, that does depend on the number of fine grid points. The reason for that is probably pure luck, as these functions happen to have both boundary conditions and mid point value (hence boundary for subdomains) that have the same value, therefore the coarse grid solver, probably solve it as a constant, which was coincidentally correct for the mid point. That leads to an accurate subdomain boundary condition, leading to very good final results.

5.2 Iterative Method

These results lead me to the iterative version of the method as described above. Here are the L_2 and H^1 error results I got for the iterative method runs, with the exact same configurations.

Table 4: Iterative Method - L_2 Errors

#	n	Solution	CDD - 2	CDD - 4	CDD - 8	CDD - 16	CDD - 32
1	64	3.070E-04	3.070E-04	3.070E-04	3.070E-04	3.070E-04	3.070E-04
	256	1.920E-05	1.920E-05	1.920E-05	1.919E-05	1.920E-05	1.920E-05
	1024	1.200E-06	1.200E-06	1.200E-06	1.199E-06	1.200E-06	1.200E-06
2	64	2.950E-15	3.670E-16	8.170E-16	1.824E-15	1.710E-15	5.810E-15
	256	2.110E-14	4.860E-15	3.030E-15	2.410E-15	1.510E-14	2.270E-14
	1024	1.080E-13	7.850E-15	3.950E-14	2.464E-14	9.930E-15	3.540E-14
3	64	1.430E-01	1.430E-01	1.430E-01	1.429E-01	1.430E-01	1.430E-01
	256	8.040E-03	8.040E-03	8.040E-03	8.041E-03	8.040E-03	8.040E-03
	1024	4.990E-04	4.990E-04	4.990E-04	4.994E-04	4.990E-04	4.990E-04
4	64	3.140E-06	3.140E-06	3.140E-06	3.142E-06	3.140E-06	3.140E-06
	256	1.960E-07	1.960E-07	1.960E-07	1.964E-07	1.960E-07	1.960E-07
	1024	1.230E-08	1.230E-08	1.230E-08	1.227E-08	1.230E-08	1.230E-08
5	64	8.850E-02	8.850E-02	8.850E-02	8.849E-02	8.850E-02	8.850E-02
	256	4.420E-02	4.420E-02	4.420E-02	4.420E-02	4.420E-02	4.420E-02
	1024	2.210E-02	2.210E-02	2.210E-02	2.210E-02	2.210E-02	2.210E-02

Table 5: Iterative Method - H^1 Errors

#	n	Solution	CDD - 2	CDD - 4	CDD - 8	CDD - 16	CDD - 32
1	64	2.280E-03	2.280E-03	2.280E-03	2.280E-03	2.280E-03	2.280E-03
	256	1.430E-04	1.430E-04	1.430E-04	1.430E-04	1.430E-04	1.430E-04
	1024	8.910E-06	8.910E-06	8.910E-06	8.910E-06	8.910E-06	8.910E-06
2	64	3.910E-03	3.910E-03	3.910E-03	3.910E-03	3.910E-03	3.910E-03
	256	2.440E-04	2.440E-04	2.440E-04	2.440E-04	2.440E-04	2.440E-04
	1024	1.530E-05	1.530E-05	1.530E-05	1.530E-05	1.530E-05	1.530E-05
3	64	6.43	6.43	6.43	6.43	6.43	6.43
	256	3.780E-01	3.780E-01	3.780E-01	3.780E-01	3.780E-01	3.780E-01
	1024	2.350E-02	2.350E-02	2.350E-02	2.350E-02	2.350E-02	2.350E-02
4	64	1.850E-05	1.850E-05	1.850E-05	1.850E-05	1.850E-05	1.850E-05
	256	1.150E-06	1.150E-06	1.150E-06	1.150E-06	1.150E-06	1.150E-06
	1024	7.210E-08	7.210E-08	7.210E-08	7.210E-08	7.210E-08	7.210E-08
5	64	9.220E-02	9.220E-02	9.220E-02	9.220E-02	9.220E-02	9.220E-02
	256	4.420E-02	4.420E-02	4.420E-02	4.420E-02	4.420E-02	4.420E-02
	1024	2.210E-02	2.210E-02	2.210E-02	2.210E-02	2.210E-02	2.210E-02

One can easily see that the iterative process converges to the same results up to machine precision as one gets by solving the original fine grid problem without any decomposition, which is obviously exact solution up to discretization error which cannot be avoided. Yet, this doesn't tell the whole story. Not only does this process converge, but for all tests ran the process reached convergence within two iterations, regardless of number of grid points, number of subdomains or function type.

6 Higher Dimensions

In higher dimension the process gets complicated, as the domains and grids geometry affects the results. I tried several different approaches for the two dimensional problem; each can be easily extended to higher dimensions.

The problem I used for testing is the immediate extension of the above one-dimensional problem to two dimensions:

$$-\Delta u = f \Leftrightarrow -(u_{xx} + u_{yy}) = f \qquad \text{on domain } \Omega = [0,1] \times [0,1]$$

6.1 Linear Interpolation

For this approach, the idea was to discretize the grid with fine width h in both directions, and as the coarse grid consider the grid with width H in both directions. Now, I encountered a new problem. When I solved the coarse grid problem I didn't get an approximation for the boundary condition of the entire subdomain, but only an approximation to the values at the subdomain's four

corners. To get boundary conditions for the fine grid subdomain problems, I used linear interpolation along the subdomain's (parallel to the axis) boundary.

The results of this approach had one thing in common with those of the one-dimensional approach, and that was that the error estimates have converged after exactly two iterations. Unlike the one-dimensional case, it did not converge to zero. In other words, I got a stagnating process.

The reason for these results is not entirely clear. My assumption is that the iterative domain decomposition worked exactly like in the one-dimensional case, however another type of error was introduced by the linear interpolation for the boundaries, which led to an error that wouldn't go away even with further iterations.

6.2 Finite Differences With Nonuniform Mesh Widths

Another approach was to discretize the coarse grid differently. As I did not want to use interpolation, I had to discretize the coarse grid in a way that a solution to the coarse-grid problem will yield the entire boundary conditions for the fine subdomain problem. To do that I used as a coarse grid, the entire "wire basket" of the union of subdomain boundaries. I tried two different approaches here, one where the domain is cut into k stripes as wide as the entire domain Ω and as high as Ω's height divided by k. The other approach was to divide Ω into k squares (assuming that k is a perfect square, and Ω is a square as well, Otherwise, there is a need to find subdomains as close in shapes to squares as possible in order to minimize the boundary length with respect to the interior area, hence minimizing the boundary's affect on the result). Both approaches lead to similar results, so I chose to present them together here.

One should note that when discretizing a wire-basket grid as displayed here, every grid point might have different mesh widths in different directions, and might have different widths than other grid points. For example, in the squares approach, some of the grid points are on horizontal boundaries, leading to width h in the horizontal direction, and width H in the vertical direction. Grid points on the vertical boundaries on the other hand have horizontal width H and vertical width h. Grid points on the intersection of horizontal and vertical boundaries (namely the subdomains' corners) have width h in both directions.

This approach adds a layer of complexity as the coarse grid problem, the fine grid problem, and fine grid sub-problems, all have different matrices, leading to a code that is harder to write and debug and generally less clear.

The results of this approach were completely different than those of the linear interpolation approach. Unlike the interpolation approach, this time I actually got a converging process, and was able to get to the solution up to any

given tolerance (greater than machine precision of course), but unfortunately the convergence rate was extremely slow.

6.3 Alternating Directions

One thing that was easy to note about this complete domain decomposition, was that after each iteration (regardless of number of dimensions, or approach used), when looking at the values of the residual, it was all zeros (up to machine precision of course), except for the grid points of the coarse grid. This was rather obvious as the fine subdomain solve enforces these residuals to zero, but it says nothing about the final error, since the error satisfies: $e_h = A_h^{-1} r_h$ and although A_h has a very local structure (only have non zero terms for adjacent points), A_h^{-1} has no such property, and the residual on the coarse grid points to errors all over the fine grid points.

This note leads to a totally different approach. Since it is known that after each iteration the residual outside the coarse grid points is set to zero, and if I assume that after each iteration, if a residual set to zero it will stay zero after the following iterations, it suggest that I can try to alternate directions between iterations. Namely run one iteration with horizontal stripes decomposition (described above), zeroing the residual everywhere, other than the horizontal boundaries, then run similar iteration with vertical stripes, zeroing all but the vertical boundaries, and if the zeros from the previous iteration are preserved, there would only be non zeros on the coarse grid corners. Now this one is easy to solve, as it is a simple solve on a course grid with width H.

Unfortunately, when attempting this approach, very quickly I found that the basic assumption behind this is false, rendering this method as non converging.

6.4 Future Work - Finite Elements on wire basket

The last approach I had in mind, which I didn't yet implement due to its complexity was to solve a coarse grid using a finite elements discretization on the wire-basket. It is my hope that this approach will converge just as the finite differences with nonuniform mesh width did, but maybe in a much higher rate. The added complexity in this scenario is the need to apply finite element discretization using nonuniform mesh sizes and shapes. One discouraging fact regarding the convergence of such approach, is that common finite elements error analysis (i.e. [4]) is assuming shape regularity of the elements, which is not observed in this method.

7 Conclusions

I have developed a domain decomposition method that avoids communication between parallel computing nodes and demonstrated that it works correctly and very efficiently for one-dimensional elliptic differential equations. So far, all my attempts to extend this method to higher dimensions either did not converge to the correct solution, or converged with a very slow convergence rate. Since one-dimensional problems are usually small enough to solve without parallel computation, this method will only be useful for real scientific computation if a fast converging extension to higher dimensions will be found.

References

[1] Tony F. Chan and Tarek P. Mathew (1994). Domain decomposition algorithms. Acta Numerica, 3, pp 61-143. doi:10.1017/S0962492900002427.

[2] M. Dryja and O.B. Widlund (1989). Some domain decomposition algorithms for elliptic problems. Iterative Methods for Large Linear Systems, pp 273-291

[3] M. Dryja and O.B. Widlund (1992). Additive Schwartz methods for elliptic finite element problems in three dimensions. Fifth Conference on Domain Decomposition Methods for Partial Differential Equations, SIAM

[4] Nochetto, Ricardo H. and Siebert, Kunibert G. and Veeser, Andreas(2009). Theory of adaptive finite element methods: An introduction. Multiscale, Nonlinear and Adaptive Approximation, pp 409-542. doi: 10.1007/978-3-642-03413-8_12. http://dx.doi.org/10.1007/978-3-642-03413-8_12