SCALLOP: A Highly Scalable Parallel Poisson Solver in Three Dimensions

Gregory T. Balls
Department of Computer Science and Engineering
University of California, San Diego
9500 Gilman Drive
La Jolla, CA 92093-0114 USA
gballs@cs.ucsd.edu

Scott B. Baden
Department of Computer Science and Engineering
University of California, San Diego
9500 Gilman Drive
La Jolla, CA 92093-0114 USA
baden@cs.ucsd.edu

Phillip Colella
Applied Numerical Algorithms Group
Lawrence Berkeley National Laboratory
Berkeley, CA USA
pcolella@lbl.gov

ABSTRACT

SCALLOP is a highly scalable solver and library for elliptic partial differential equations on regular block-structured domains. SCALLOP avoids high communication overheads algorithmically by taking advantage of the locality properties inherent to solutions to elliptic PDEs. Communication costs are small, on the order of a few percent of the total running time on up to 1024 processors of NPACI's and NERSC's IBM Power-3 SP sytems. SCALLOP trades off numerical overheads against communication. These numerical overheads are independent of the number of processors for a wide range of problem sizes. SCALLOP is implicitly designed for infinite domain (free space) boundary conditions, but the algorithm can be reformulated to accommodate other boundary conditions. The SCALLOP library is built on top of the KeLP programming system and runs on a variety of platforms.

Keywords

computation-intensive applications, parallel and distributed algorithms, program optimization and performance programming

1. INTRODUCTION

If Terascale computing is to mature in the coming decade, then it must overcome the steadily increasing costs of interprocessor communication relative to arithmetic computation. One approach is to mask communication costs by overlapping communication with computation [17, 18, 4, 3, 2]. An alternative approach is to reduce the amount data communicated at the expense of additional computation. The fast multipole method [12], the method of local corrections [1], and the finite element of Bank and Holst [7] all take this second approach. Results presented by Holst [13] include a rigorous proof that these types of algorithms can produce accurate results with little communication.

We present SCALLOP, an elliptic partial differential equation solver in three dimensions that also employs the latter strategy, trading off the high cost of communication through algorithmic reformulation. SCALLOP employs a method of local corrections to mitigate distant numerical coupling inherent in elliptic partial differential equations, using a coarse grid approximation that effectively reduces the amount of information that must be moved among processors. The added computational cost is purely local work, and the cost remains a constant which is independent of the number of processors.

SCALLOP solves partial differential equations with infinite domain (free space) boundary conditions. Such boundary conditions are especially useful for calculations of astrophysics phenomena. SCALLOP can be modified for problems involving other boundary conditions, but in this paper we focus exclusively on the infinite domain case, and, for simplicity, we only discuss the solution to a particular partial differential equation, the Poisson equation.

Unlike domain decomposition methods such as [16] which require multiple iterations between the local and nonlocal descriptions, SCALLOP does not perform repeated iterations between coarse and fine levels or several communication steps. SCALLOP reaches a solution to the Poisson equation in three steps and communicates data only twice. First, coarse grid data are communicated to generate a global coarse grid charge field. Second, coarse and fine boundary condition data are communicated once among neighboring regions. These communication costs are low in practice–on the order of a few percent–and come at the expense of computational (numerical) overhead. We show that the extra computation involved is reasonable, and significantly, that the computational overhead is independent of the number of processors for a wide range of problem sizes. As a result, we are able to demonstrate scalability on up to 1024 processors of an IBM SP system, and we plan future computations on thousands of processors. Our goal is to utilize fully the resources of ever-growing computational platforms. Therefore, we define scalability as the ability to run problems scaled appropriately to the number of processors and available memory in roughly constant time. This

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SC '03 Phoenix, Arizona, USA
Copyright 2003 ACM 1-58113-695-1/03/0011 ...$5.00.

type of scalability is often called scaled speed-up.

SCALLOP is a three dimensional extension of the two-dimensional algorithm due to Balls and Colella [5, 6]. Our contribution is in two parts. The first part entails new numerical techniques required in three dimensions and to permit the computation to scale to thousands of processors. The second part is the theory behind performance tradeoffs needed to convert a terse description of a numerical algorithm into a highly scalable solver. Significant performance programming accompanied this latter contribution, and in the process, we devised a performance model that is shown to match our results well.

SCALLOP is representative of a class of algorithms that employ sophisticated numerical techniques to reduce communication costs. The techniques in turn require an appropriate software infrastructure to manage the underlying details, in particular the bookkeeping. To this end, SCALLOP was built with the KeLP programming system [10], a framework for implementing scientific applications on distributed memory parallel computers. KeLP provides geometric and communication abstractions that facilitated the development of SCALLOP without sacrificing performance.

2. PRELIMINARIES

The equation we solve is the Poisson equation in three dimensions with a charge distribution ρ with compact support, i.e., the charge is only nonzero in a finite region of space. Specifically, we seek the solution ϕ to

$$\triangle\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = \rho(x,y,z)$$

which has far-field behavior characterized by

$$\phi = -\frac{R}{4\pi|\vec{x}|} + o\left(\frac{1}{|\vec{x}|}\right), \qquad |\vec{x}| \to \infty,$$

where R is the total charge:

$$R = \int_\Omega \rho(\vec{x})\, d\vec{x},$$

and the region Ω contains the support of the charge ρ.

For many engineering calculations, methods which are accurate to $O(h^2)$ provide a good balance between accuracy and work required. We seek a solution which is accurate to $O(h^2)$ over the discretized computational domain Ω^h, where h is the uniform discretization, as illustrated in Figure 1. This computational domain corresponds to the index set of the discrete solution ϕ^h, i.e., the indices of the underlying discrete mesh.

Since our goal is to solve the problem on parallel processors, we partition Ω^h into a set of disjoint subdomains Ω^h_k:

$$\Omega^h = \bigcup_k \Omega^h_k.$$

Our method entails solving local problems on each of the Ω^h_k in parallel, as well as on a single coarsened global mesh Ω^H. The spacing of this coarsened mesh is $H = Ch$, where C is a specified *coarsening factor*.

We choose the domain Ω^h to be a rectangular region, $\Omega^h = [\vec{l}, \vec{u}]$, where $\vec{l}$ and $\vec{u}$ are the integer vectors corresponding to the lower and upper corners of the region. The coarsened domain is then defined as

$$\mathcal{C}(\Omega^h, C) = \Omega^H = [\lfloor \vec{l}/C \rfloor, \lceil \vec{u}/C \rceil]$$

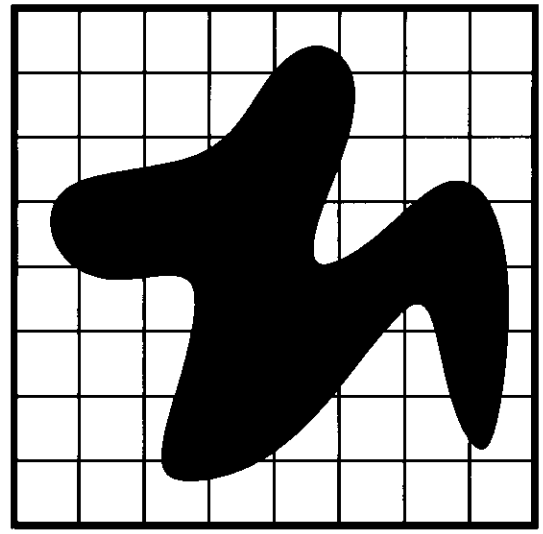

Figure 1: Discretization. The outer boundary of the grid shown here corresponds to Ω^h. The regular mesh, with spacing h represents the uniform discretization of Ω^h. The charge field is non-zero within the grey region shown, also called the support of the charge ρ. The support of ρ lies completely within Ω^h.

where the operators $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$ represent the *floor* and *ceiling* operators, respectively.

Because our meshes are node-centered, the points of Ω^H map directly onto corresponding points in Ω^h, as shown in Figure 2, and no averaging is required to coarsen the mesh data. Thus, we can coarsen the mesh by sampling the mesh without having to interpolate. In particular, we coarsen a fine grid representation using the *sample* operator $\mathcal{S}^H$: for each point $\vec{x}_C$, we can find the coarse grid value $\psi^H(\vec{x}_C)$ (where ψ^H has grid spacing H) by finding the fine grid point $\vec{x}$ at the corresponding position in ψ^h (with grid spacing $h = H/C$):

$$\psi^H(\vec{x}_C) = (\mathcal{S}^H(\psi^h))(\vec{x}/C) = (\psi^h)(\vec{x})$$

For the discussion that follows, we also need one more bit of notation. The *grow* operation extends or shrinks an index domain by a uniform amount in each direction. If $\Omega^h = [\vec{l}, \vec{u}]$ (where $\vec{l} = (\mathtt{l}_x, \mathtt{l}_y, \mathtt{l}_z)$ and $\vec{u} = (\mathtt{u}_x, \mathtt{u}_y, \mathtt{u}_z)$), we define *grow* as

$$grow(\Omega^h, g) = [\vec{l} - (g,g,g), \vec{u} + (g,g,g)].$$

When $g < 0$, grow returns a shrunken domain.

3. THE METHOD

The domain decomposition method presented here is built upon a method for solving single-processor infinite domain Poisson problems, as described in [5]. We will summarize the single-grid algorithm first, and then describe the domain decomposition algorithm.

3.1 A Serial Infinite Domain Poisson Solver

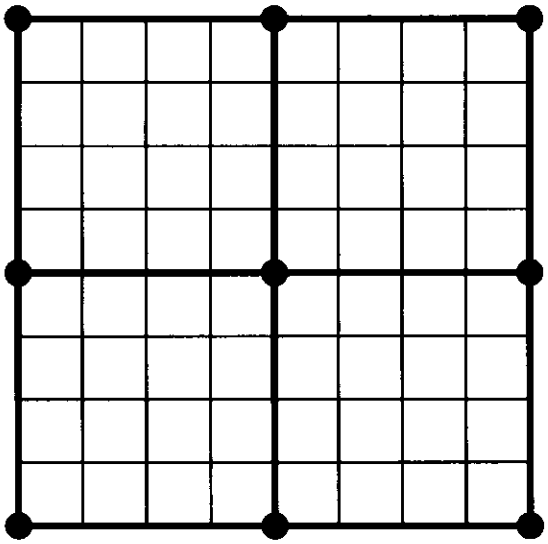

Figure 2: A sampling mesh for a coarsening factor of 4. Coarse grid points shown as bold face disks coincide with fine grid points, so sampling is sufficient to transfer data from fine to coarse: no averaging is necessary.

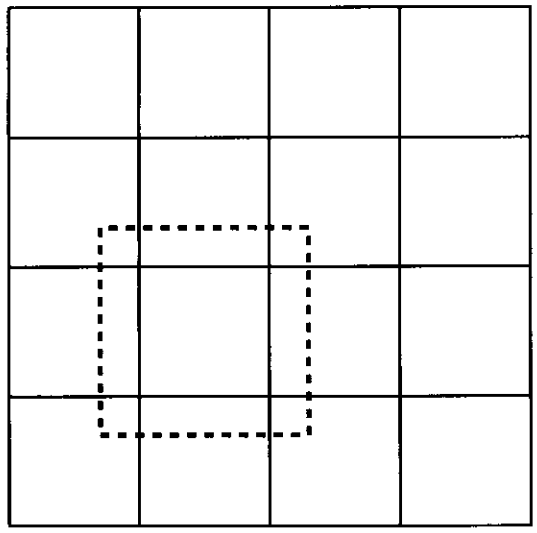

Figure 3: Extended Local Grids. The local solution to each of the 16 subdomains shown here is found on an extended region. One such region is indicated by the dashed box.

Following methods described in [14] and [15], we are able to calculate a solution to the Poisson equation with infinite domain boundary conditions in three steps, using two grids. The first grid, referred to here as the *inner grid* or $\Omega^{h,g}$, is 10% larger than the support of the charge in each dimension. The second grid, called the *outer grid* or $\Omega^{h,G}$, is 20% larger than the support of the charge in each dimension. Spacing the grids this way, such that the boundaries are sufficiently removed from the support of the charge, is necessary for accurate solutions. The three steps required to calculate the solution are as follows:

1. Find the solution to the Poisson equation on the inner grid, $\Omega^{h,g}$, using Dirichlet boundary conditions.

2. Calculate a charge along the inner grid boundary equal to the normal derivative of initial solution at the inner grid boundary, and integrate this charge numerically to calculate boundary conditions at each point on the outer grid.

3. Find the solution to the Poisson equation on the outer grid, $\Omega^{h,G}$, using the boundary conditions calculated in step 2.

This approach is identical to the approach described in [5] and [6] except in the way the integration is performed in step 2. In two dimensions, the integration of the charge requires $O(N^2)$ work, where N is the length of a side of the grid. In three dimensions, straightforward integration requires $O(N^4)$ work and would soon overwhelm the $O(N^3)$ work required for each Poisson solution. The work required can be reduced to $O(N^3)$ by taking advantage of the smoothness of the solution, however. We do this by integrating the charge from the inner grid onto a coarsened version of the outer grid, with mesh spacing $H = h/O(\sqrt{N})$. The integration can then be written as:

$$\phi_k^{H,G}(\vec{y})\Big|_{\partial\Omega_k^{H,G}} = \sum_{\vec{x}\in\partial\Omega_k^{h,g}} \frac{\partial\phi_k^{h,g}}{\partial\vec{n}}(\vec{x})G(\vec{x},\vec{y})$$

where $G()$ is the Green's function:

$$G(\vec{x},\vec{y}) = -\frac{1}{4\pi|\vec{x}-\vec{y}|}.$$

The integration step thus requires $O(N^3)$ work, and the subsequent interpolation required to set the outer fine grid boundary from the coarse grid data adds only $O(N^2)$ work.

3.2 Domain Decomposition

The domain decomposition algorithm in SCALLOP is a finite-difference analogue of Anderson's method of local corrections [1]. Our algorithm consists of three computational steps interspersed by two communication steps. We follow the general design of the two-dimensional algorithm, as described in [5] and [6].

1. In the first step we calculate a solution on each local subdomain, which has been augmented with an overlap region. As in the serial case, we need an expanded $\Omega_k^{h,G}$, at least 20% larger than Ω_k^h in each dimension, in order to calculate an accurate infinite domain solution. In the domain decomposition method, the effect of this requirement is to cause the individual subdomains to overlap, as shown in Figure 3.

   On each extended subdomain $\Omega_k^{h,G}$ we solve the Poisson equation

$$\triangle_{27}\phi_i^h = \rho_k^h$$

with infinite domain boundary conditions, evaluated as described for the serial case above. The $\triangle_{27}$ operator represents the Laplacian calculated with the 27-point stencil of nearest neighbor points. The 27-point stencil is necessary because of its error characteristics: the error associated with the 27-point stencil is essential for maintaining $O(h^2)$ accuracy overall by coordinating the precise interaction between the coarse and fine grid data.

2. We next couple the local solutions computed in the previous step by solving another Poisson equation on a grid covering the entire domain. Owing to well known locality properties of solutions to Poisson's equation, a crucial observation is that we may perform the far-field coupling with a reduced description of the data, i.e., on a coarsened version. We can therefore keep the cost of global coupling modest, both in terms of communication and computation. To maintain $O(h^2)$ accuracy, we will need to make local corrections to this far-field data in the third and final step.

   This step contains 4 sub-steps:

   (a) Coarsen each local solution computed in step 1, by applying the sampling operator $\mathcal{S}^H$ to the local fine meshes:

   $$\phi_k^{H,G} = \mathcal{S}^H(\phi_k^{h,G})$$

   (b) Apply the Laplacian operator, $\triangle_{27}$, to $\phi_k^{H,G}$, in order to generate local coarse charge fields:

   $$R_k^H = \begin{cases} \triangle_{27}\phi_k^{H,G}, & \text{on } grow(\Omega_k^{H,G}, -1) \\ 0, & \text{otherwise.} \end{cases}$$

   (c) Assemble the local coarse charge fields into a single global right hand side. Due to the extension of the local grids, as shown in Figure 3, the pieces will overlap in physical space. We accumulate the overlapping parts into the global right hand side by summation:

   $$R^H = \sum_k R_k^H$$

   (d) Solve the Poisson equation with infinite domain boundary conditions on the coarsened right hand side R^H:

   $$\triangle_{27}\phi^H = R^H \text{ on } \Omega^H$$

3. In the third step we compute the global fine grid solution. At the end of this step, each processor will have the solution for its assigned subdomain Ω_k^h.

   This step entails a calculation of a local solution on each patch, Ω_k^h, with Dirichlet boundary conditions:

   $$\triangle_7\phi_k^h = \rho_k^h.$$

   where $\triangle_7$ is the Laplacian operator using a 7-point stencil of nearest neighbors along the Manhattan directions[1]. Note that these calculations are carried out on the non-overlapping Ω_k^h subdomains.

[1] We do not use the more expensive 27-point stencil, since the accuracy required for this solution is not nearly as stringent as in the previous steps [5]

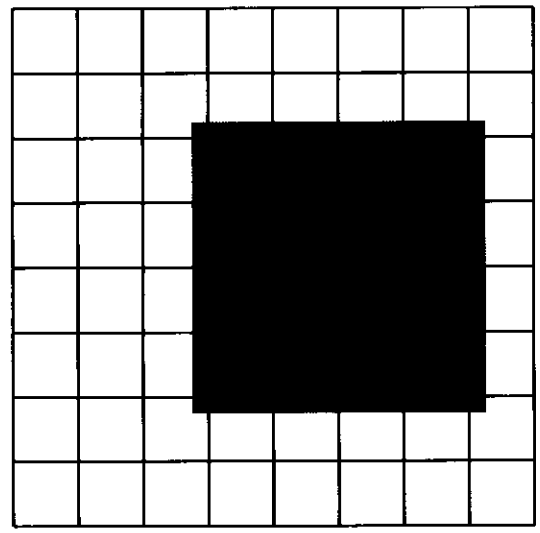

Figure 4: The domain of influence of a fine grid point $\vec{x}$. With the correction distance $D = 2$, the domain of influence for point $\vec{x}$, marked by an x in the figure, is the five-by-five set of coarse grid points within the shaded region.

Boundary conditions for this final step are determined by combining values interpolated from corresponding coarse patches computed in step 2 with data from nearby fine patches. We must correct the coarse data to remove local effects before interpolating, since these local effects will be added through the local fine grid data. Communication is required among nearest neighbors in order to share the fine grid data near the interfaces prior to this final solution pass. The boundary computation proceeds as follows.

To describe the computations, we introduce the notion of a *domain of influence*. The domain of influence of $\vec{x}$ is the set of coarsened points within D units of $\vec{x}$. The variable D represents a *correction distance* required for accurate interpolation. For our $O(h^2)$ accurate algorithm, $D = 2$. If C is the coarsening factor, as before, we can define the domain of influence as

$$\Omega_{\mathcal{D}(\vec{x})} = [\vec{x}/C - (D, D, D), \vec{x}/C + (D, D, D)],$$

as depicted in Figure 4. We use the notation $\mathcal{D}(\vec{x})$ to signify the set of indices corresponding to subdomains k' that contain the domain of influence of $\vec{x}$:

$$\mathcal{D}(\vec{x}) = \{k' | \Omega_{\mathcal{D}(\vec{x})} \subset \Omega_{k'}^{h,g}\}$$

For each boundary point $\vec{x}$ on the boundary of Ω_k^h, we compute

$$\phi_k^h(\vec{x}) = \sum_{k' \in \mathcal{D}(\vec{x})} \phi_{k'}^h(\vec{x}) + \mathcal{I}(\phi^{H,Correct})$$

where $\mathcal{I}$ is the interpolation operator used to transfer information from coarse grid to the fine grid, and $\phi^{H,Correct}$ is the locally corrected coarse grid.

Thus, each boundary point gets a local contribution from all overlapping fine meshes computed in step 1 plus a quantity derived from the coarse mesh, providing the effects of far-field charges.

To locally correct the coarse grid solution at a point $\vec{x}$, we subtract the coarsened values of the fine grids $\phi^h_{k'}$ for all grids $k' \in \mathcal{D}(\vec{x})$. Thus we compute $\phi^{H,Correct}$ as follows:

$$\phi^{H,Correct}(\vec{x}) = \phi^H - \sum_{k' \in \mathcal{D}(\vec{x})} \mathcal{S}^H(\phi^h_{k'})(\vec{x}/C),$$

where ϕ^H is the global solution computed in step 2, and $\mathcal{S}^H(\phi^h_{k'})(\vec{x}/C)$ is the coarsened version of the solution computed in step 1.

4. PERFORMANCE MODEL

As mentioned previously, the principle behind SCALLOP is to trade off communication against computation. We next discuss these tradeoffs and show that they are reasonable. We describe a performance model, and use it to show that in theory the overheads are reasonable. In the following two sections we reconcile our predictions with practice. Our model also motivates future extensions that should enable SCALLOP to scale to 256K processors, and we discuss these improvements later on.

In determining the computational overhead in SCALLOP, we will use the serial infinite domain Poisson solver as a baseline. We will first show that the cost of our initial fine grid solutions is similar to the cost of a serial solution, except that our initial solutions are calculated using a 27-point rather than a 7-point stencil. The computational overhead in SCALLOP, then, can be described as the sum of three costs: the extra time required for the 27-point stencil, the cost of the coarse grid solution, and the time required for the final local solutions on fine grid data. We will discuss each of these costs, and show that with the proper choice of a coarsening factor, SCALLOP should be able to scale to thousands of processors.

4.1 Serial Infinite Domain Poisson Solver

The SCALLOP algorithm is built from many of the same computational pieces as the serial infinite domain Poisson solver. We will examine the computational costs involved in the serial solver first and compare this cost with the cost of the initial solutions in SCALLOP.

SCALLOP was originally implemented using a multigrid solver [9] to calculate the solutions to the Dirichlet problems within our infinite domain solver. Very recently, SCALLOP was redesigned to use an FFT solver in place of multigrid since we found that the FFTW [11] library provided significantly faster uniprocessor performance. Most of the results in this paper are based on the original multigrid code, however, so we will base our analysis on that solver and then discuss the effects of switching to the FFTW solver.

For now, let us consider only cubical domains with edge length N, and let us define the time to solve a Dirichlet problem with a 7-point stencil on a grid of size N as $S_7(N)$.

Due to requirements of the infinite domain solver, we must solve two Dirichlet problems on domains larger than size N, one with edges of length $1.1N$ and another with edges of length $1.2N$. For our three dimensional problem,

$$S_7(1.1N) \approx 1.3S_7(N)$$

and

$$S_7(1.2N) \approx 1.7S_7(N)$$

The boundary condition calculation required for an infinite domain Poisson solution is dominated by an $O(N^3)$ problem, as discussed in Section 3.1. By taking advantage of symmetry in the calculation, we are able to reduce the number of times we evaluate the distance calculation within the Green's function by close to a factor of 10. After these optimizations, we find experimentally that the boundary calculation takes approximately as long as $1.2S_7(N)$.

The total time required, T_{SID}, for a serial infinite domain Poisson solution using the 7-point stencil is then approximately

$$\begin{aligned} T_{SID} &\approx 1.3S_7(N) + 1.7S_7(N) + 1.2S_7(N) \\ &\approx 4.2S_7(N). \end{aligned}$$

Thus the time required for a solution with infinite domain boundary conditions is significantly greater than the time required for a Dirichlet problem of similar size.

The initial fine grid solutions performed in SCALLOP are of the same aggregate size as an equivalent serial calculation. For a problem divided into q subdomains on a side, or q^3 total subdomains, each subdomain for an problem of size N will have an edge length of N/q. It is easy to see that the costs of calculating the inner and outer solutions for a collection of subdomains are the same as those found for the serial case above.

The only computational overhead introduced by our algorithm during the initial fine grid solutions involves the use of a 27-point stencil. The 27-point stencil is required for accuracy reasons. The total work required for $S_{27}(N)$ is approximately 3.8 times the work required for $S_7(N)$ in terms of floating point operations. Due to effects in the memory hierarchy of the machines we have used, however, the difference in the time required for $S_7(N)$ and $S_{27}(N)$ is not as great as the operation count would suggest. For our runs on Blue Horizon and Seaborg[2], we have found that $S_{27}(N)$ is very nearly 2.0 times more expensive than $S_7(N)$. The cost of the initial local fine grid calculations, T_{local}, in SCALLOP can be estimated as

$$\begin{aligned} T_{local} &\approx 2.0(1.3S_7(N) + 1.7S_7(N)) \\ &\quad +1.2S_7(N) \\ &\approx 7.2S_7(N) \\ &\approx 1.7T_{SID}. \end{aligned}$$

As we will see, this factor of 1.7 represents the majority of the computational overhead in our current implementation.

The change from our original multigrid solver to our newer FFTW-based solver alters these estimates somewhat. The main motivation for using FFTW is the uniprocessor performance it provides. We find that we are able to solve an equivalent problem with FFTW approximately 8 times faster than with our multigrid code. As a result, we have a new benchmark for comparison:

$$S_{7-fftw}(N) \approx 0.13S_7(N).$$

Another benefit of the FFTW solver is that it does not impose a performance penalty for solving the 27-point stencil, that is, $S_{27-fftw}(N) \approx S_{7-fftw}(N)$. Compared with the

[2]Both Blue Horizon and Seaborg are IBM SP systems with Power3 processors.

FFTW Poisson solver, however, the infinite domain calculation appears significantly slower, requiring approximately $9.6S_{7-fftw}(N)$. In total, these changes result in an estimate for $T_{local-fftw}$ of

$$\begin{aligned} T_{local-fftw} &\approx 3.0S_{7-fftw}(N) \\ &\quad +9.6S_{7-fftw}(N) \\ &\approx 13S_{7-fftw} = T_{SID-fftw} \end{aligned}$$

In this case, we see that the cost of the initial fine grid solutions in SCALLOP will be the same as for a corresponding serial solution, but that the cost of calculating the infinite domain boundary conditions swamps the cost of calculating the solutions to the simpler Dirichlet boundary condition problems.

4.2 The Coarse Grid Solution

In comparing the computational cost of SCALLOP to a serial infinite domain solver, the cost of computing the solution on the global coarse grid is overhead. We want to determine what range of problems can be solved with minimal computational overhead due to the coarse grid calculation. Our goal is to keep the cost of the coarse grid solution small enough that this overhead can be ignored.

As before, let N be the length of a side, thus N^3 is the total number of points. Let q be the number of subdomains on a side. Then q^3 is the total number of subdomains, or processors, and

$$N_f = \frac{N}{q}$$

is the length of a local fine subdomain.

Let C be the coarsening factor, as defined previously, such that the size of coarse grid, N_c, is N/C. Again, as with the fine grid solutions, the coarse grid solutions are calculated on domains larger than this, but since both the fine and coarse grids are enlarged by the same ratio, we will use these definitions of N_f and N_c for this analysis.

In order to minimize the overhead due to calculating the coarse grid solution, we want $N_c < N_f$ and therefore

$$\frac{N}{C} < \frac{N}{q}$$

or

$$q < C.$$

Since the accuracy of the solution is only weakly dependent on our choice of C, we prefer to have $q \leq C/2$. This choice makes the coarse grid calculation 1/8 as large each fine grid calculation and therefore nearly an order of magnitude less expensive. Thus, the cost can safely be ignored.

The ratio $N/(qC)$ represents the number of coarse grid cells per fine grid subdomain. The case where $qC = N$ corresponds to the limiting case, representing both the maximum amount of parallelism and the maximum coarsening factor possible. In this limit there is exactly one coarse cell per fine grid subdomain, or a single coarse point on each corner of a subdomain. When $qC < N$ we have multiple coarse grid points per fine domain, including some on the interior.

Given the amount of memory available per processor, it is straightforward to calculate the maximum possible problem size. At the limit of $qC = N$, we have $C = N/q$ which is the same as N_f. If we choose $q = C/2$, as suggested above, then we can determine the maximum number of subdomains in terms of the size of the local fine grid:

$$q = \frac{N_f}{2}.$$

As an example, given a machine with sufficient memory for 128^3 points per processor ($N_f = 128$), the algorithm would allow scaling up to $q = 64$ subdomains per side, or $q^3 = p = 262,144$ processors without introducing excessive computational overhead due to the coarse grid calculation. Platforms in this regime will be available soon[3].

4.3 Summary of Computational Overhead

We have discussed the first two sources of computational overhead in SCALLOP: the extra time required for the 27-point stencil and the cost of the coarse grid solution. The only source of overhead not addressed so far is the cost of the final fine grid solutions. We will briefly discuss the cost of this final step and then sum up these contributions to get an estimate of the overall computational overhead of SCALLOP.

The final fine grid solutions are standard Dirichlet problems calculated using a 7-point stencil. These are precisely the type of problem on which we based the estimates of the infinite domain solutions. Thus the overhead due to the final fine grid solutions is $S_7(N)$, or approximately $0.24T_{SID}$, for the multigrid version of SCALLOP.

At this point we can sum up the various calculation costs. In the current implementation, the time spent on the initial fine grid solutions is $1.7T_{SID}$. By choosing the coarsening factor, C, and the number of subdomains per side, q, carefully, we can limit the time required for the solution on the coarse grid to 1/8 the cost of the initial fine grid solutions, or approximately $0.21T_{SID}$. Adding in the cost of the final fine grid solutions, we arrive at a total cost estimate of

$$\begin{aligned} T_{SCALLOP} &\approx 1.7T_{SID} + 0.21T_{SID} + 0.24T_{SID} \\ &\approx 2.2T_{SID}. \end{aligned}$$

While a factor of 2 in computation time may seem large, we believe it is a reasonable tradeoff for a method which scales to large numbers of processors with very little communication overhead.

For the FFTW version of SCALLOP we found that the initial fine grid solution takes only as long as the equivalent serial solution. In the FFTW version, the coarse grid still requires 1/8 as much work as the initial local fine grid solutions. We expect the final fine grid solutions to require only about 1/13, or 0.08 as much time as $T_{SID-fftw}$, however, since the infinite domain solutions are bound by the infinite domain boundary condition calculation in this case. Collecting all these factors, we find

$$\begin{aligned} T_{SCALLOP-fftw} &\approx T_{SID-fftw} + 0.13T_{SID-fftw} \\ &\quad +0.08T_{SID-fftw} \\ &\approx 1.2T_{SID-fftw}. \end{aligned}$$

For the FFTW version of SCALLOP, the computational overhead introduced is quite small. We note that a large part of the reason for the low overhead is the inefficiency

[3]BlueGene/L will have 128K processors with 128 MB of memory per processor. This amount of memory is sufficient to store 4 8-byte field variables at each of 128^3 points per processor with room left over for overhead.

of the infinite domain boundary condition calculation. At this point more effort needs to be focused on optimizing that piece of the algorithm. Even if we manage to reduce the cost of $T_{SID-fftw}$ to nearly S_{7-fftw}, the computational overhead of SCALLOP will still be only slightly more than a factor of 2.

4.4 Further Implementation Details

There are a few details of our implementation that affect the performance of SCALLOP and therefore must be discussed in order to understand the results that follow.

In the current implementation, the global coarse grid problem is solved redundantly on all the processors. This phase of the computation could be made more efficient by using a parallel solver, but we have chosen this approach since all the processors will need a copy of the result anyway. We have taken this redundant computation into account implicitly in our discussion of the scaling requirements in section 4.2. Since the coarse grid computation will ideally take a small amount of time, we have not incorporated a parallel coarse grid solver into SCALLOP.

Also, our implementation currently only creates cubical subdomains, but it allows for multiple subdomains to be computed per processor. In our test cases, we attempt to keep the ratio of fine grid points and coarse grid points nearly constant. We do this by adjusting both the coarsening factor and the number of subdomains per processor. The number of fine grid points is equivalent to N^3/q^3, while the number of coarse grid points is N^3/C^3, so for our tests we keep the ratio between q and C as close to constant as possible.

Another important detail relates to the size of the initial fine grid subdomains. In the first step described in Section 3.2, we described the overlap of subdomains due to the requirements of the infinite domain boundary conditions. This requirement can be specified as

$$\Omega_k^{h,G} = grow(\Omega_k^h, (0.1(len(\Omega_k^h))).$$

For the third step described in Section 3.2, we need an annulus of D coarse grid points around each subdomain. The values at these coarse grid points need to represent the same solution as the initial fine grid solution, sampled onto a coarser mesh.

In order to generate these coarse grid points, we solve the initial fine grid problems on domains large enough to allow us to sample the necessary coarse grid points from the fine grid in step 2a. This implementation choice leads to an additional requirement on $\Omega^{h,G}$:

$$\Omega_k^{h,G} = grow(\Omega_k^h, max(0.1(len(\Omega_k^h)), CD)),$$

where $len(\Omega_k^h)$ represents the length of the longest side of Ω_k^h, C represents the coarsening factor, and D represents the correction distance. For large coarsening factors, the size of $\Omega_k^{h,G}$ is determined by CD.

For $O(h^2)$ accuracy, $D = 2$. Our implementation therefore requires an extended domain of at least $2C$ points in each direction, or an additional $4C$ points per side. As a result, the fine grids may need to be extended beyond the 10% growth in each direction (20% more points per side) required for a serial infinite domain calculation. Thus, the numerical constraints behind the SCALLOP algorithm do not predict the performance constraints required to achieve scalability. Specifically,

$$N_{f,G} = max\{1.2N_f, N_f + 4C\}.$$

To avoid introducing additional overhead for our implementation, we need $4C < 0.2N_f$ so that fine grid sizes are not determined by the $4C$ term in the max above. If we also have $C = 2q$ as before, to limit the overhead due to the coarse grid solution, then we require

$$4C = 8q < 0.2N_f$$

or

$$q < 0.025N_f.$$

As a result, the requirements imposed by our implementation force us to increase computational costs in both the local fine solutions and the global coarse solutions in order to scale to larger numbers of processors (in the next section). First, we violate the constraint that $q \leq C/2$, mentioned earlier in this section. We set $C \approx 1.25q$, thus making the coarse grid solution approximately one-half as computationally expensive as the fine grid solutions, rather than one-eighth, as was our original goal. This effect is implementation-dependent, and we have a strategy for removing this obstacle, which we will discuss below. Second, we allow the fine grid subdomains to be expanded slightly more than $0.2N_f$ in some cases, allowing for larger values of C, but creating additional computational overhead on the fine grid solutions.

In order for SCALLOP to be efficient and scalable up to and beyond the limits of current large scale machines, it is important to remove this limitation on the relationship between C and N_f. We believe it is possible to generate the necessary coarse grid points in the overlap region using a Green's function, in the same way that we calculate the infinite domain boundary conditions for the initial fine grid solutions. The calculation of the coarse grid data for the overlap regions could then be accomplished with little additional computation and without enlarging the corresponding fine grids beyond the requirements of the infinite domain boundary calculation. We plan on implementing these changes in the near future.

5. RESULTS

In this section we present computational results which demonstrate the low communication overhead of SCALLOP. We also compare our performance results with the estimates presented earlier. We ran the multigrid implementation of SCALLOP on up to 1024 processors of an IBM SP system. While generating these data, we found that SCALLOP ran out of memory on larger problem sizes toward the end of calculations. Consequently, for runs with more than 128 processors, we ran on twice as many nodes as necessary, using only half the processors on each node in order to have sufficient memory to complete the calculations. We have since isolated the problem, which was due to allocating memory in a way which scaled very poorly. By running on twice as many nodes as should have been necessary, we believe we penalized communication even more than in the ideal case, but the inefficient memory allocation routine also causes some of the later serial stages of the SCALLOP algorithm to run significantly more slowly than expected. We have corrected the faulty code in the FFTW version of SCALLOP, and we show a few of those results later, but we were unable to apply

the corrections and reproduce the results for the multigrid version of SCALLOP in time for this paper.

5.1 Hardware

We ran on NPACI's Blue Horizon IBM SP system[4], located at the San Diego Supercomputer Center. Blue Horizon contains 144 POWER3 SMP High Nodes (model number 9076-260) interconnected with a "Colony" switch. Each node is an 8-way Symmetric Multiprocessor (SMP) based on 375 MHz Power-3 processors[5], sharing 4 Gigabytes of memory, and running AIX 5L. Each processor has 8 MB of 4-way set associative L2 cache, and 64 KB of 128-way set associative L1 cache. Both caches have a 128-byte line size. Each CPU has 1.5 GB/sec bandwidth to memory.

We also ran on NERSC's Seaborg IBM SP system, which is similar to Blue Horizon, but with 16 processors and 16 Gigabytes of memory per node, and a total of 146 nodes.

On both machines, we used the installed IBM C++ and Fortran 77 compilers, mpCC and xlf. C++ code was compiled with compiler options `-O2 -qarch=pwr3 -qtune=pwr3`. Fortran 77 was compiled with compiler option `-O2`.

5.2 Performance Measurement Technique

We used the standard environment variable settings on Seaborg and Blue Horizon, and we collected timings in batch mode using *loadleveler*. The timings reported are based on wall-clock times, obtained with `MPI_Wtime()`.

Each calculation was performed 3 times. Variation among the runs was less than 10%. The times reported are for the runs with the shortest total times.

5.3 Communication and Computation Overhead

The run parameters and timing results for the performance tests on Blue Horizon and Seaborg are shown in Tables 1 and 2.

In order to measure performance, we scaled the work linearly with the number of processors. Ideally, computation time would remain constant. The scaled speed-up tests shown in Figure 5 demonstrate that SCALLOP scales well up to 512 processors and that the running times vary by a reasonable amount from the ideal. Communication overhead is low in SCALLOP. As shown in Figure 6, communication overhead is less than 3% on up to 512 processors.

As can be seen in Tables 1 and 2, time spent on the coarse grid solutions is approximately one half to one quarter the time spent on fine grid solutions. Ideally, the time required for coarse grid solutions would be negligible, but these results match our expectations since we chose C values of approximately $1.25q$ and $1.5q$. Reducing the fine grid overlap requirement as described in Section 4.4 will consequently allow us to reduce the global coarse grid size, thereby reducing this part of the computational overhead.

For runs on up to 64 processors, the time required for the first pass solutions on the fine grid data ranges from 6.3 to 10.45 times the time required for the final pass, which roughly matches the estimated factor of 7.2 predicted in Section 4.3. For the same set of runs, the ratio of total time to the time required for the final solution ranges from 9.4 to 15.0. In Section 4.3, we predicted a ratio between the total

[4] `http://www.npaci.edu/BlueHorizon/`

[5] http://www.rs6000.ibm.com/resource/technology/sppw3-tec.html

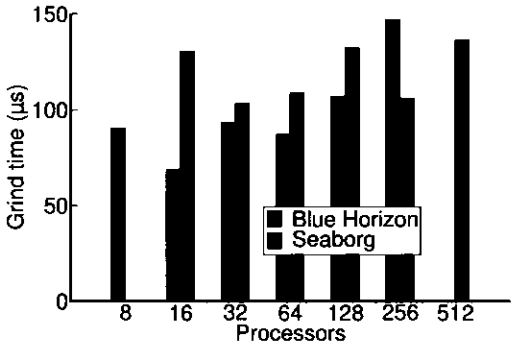

Figure 5: Computation times per point on Blue Horizon and Seaborg. Simulations on Blue Horizon have approximately 100^3 solution points per processor. Simulations on Seaborg have approximately 160^3 solution points per processor.

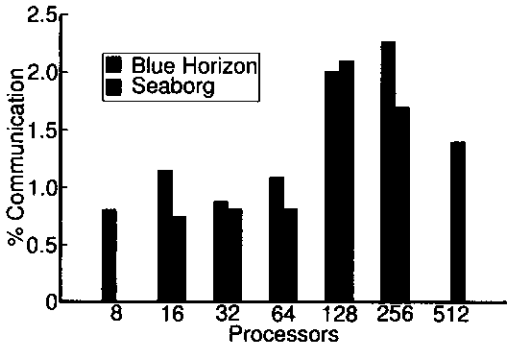

Figure 6: Communication overhead is small.

Input Parameters				Times for Each Stage (seconds)					
P	q	C	N	Local	Reduction	Global	Boundary	Final	Total
8	2	3	192^3	57.6	0.06	15.3	0.59	6.3	80.1
64	4	6	384^3	56.7	0.14	11.0	0.70	8.2	77.1
16	4	4	256^3	51.9	0.11	12.7	0.72	6.6	72.3
128	8	8	512^3	73.2	0.32	12.6	1.93	23.4	112.0
32	4	5	320^3	75.6	0.11	11.6	0.73	7.3	95.5
256	8	10	640^3	109.0	1.28	12.3	2.14	25.3	150.4

Table 1: Input parameters and timing breakdowns for runs performed on NPACI's Blue Horizon. The Local and Global solutions employ a free space solver using the $\triangle_{27}$ operator, and the Final calculation solves the Dirichlet problem with the $\triangle_7$ operator. Reduction accumulates the coarsened local solutions into a single coarse grid for the Global solve. Boundary corrects the result of the Final solution. P is the number of processors, q is the number of subdomains on a side, and C is the coarsening factor. The lines are sorted such that runs with the same number of points per processor appear next to each other, but all the runs have approximately 100^3 points per processor. Only the runs for P=8 and P=64 employ just one domain per processor.

Input Parameters				Times for Each Stage (seconds)					
P	q	C	N	Local	Reduction	Global	Boundary	Final	Total
16	4	3	384^3	234.5	0.54	190.7	2.91	30.7	461.5
128	8	6	768^3	229.5	4.29	136.8	5.72	89.8	467.8
32	4	4	512^3	237.1	0.80	153.4	2.86	37.8	433.8
256	8	8	1024^3	214.9	2.35	127.3	5.33	90.9	443.9
64	4	5	640^3	263.7	0.81	136.8	2.86	40.6	445.3
512	8	10	1280^3	311.37	1.98	108.3	6.04	127.7	558.0

Table 2: Input parameters and timing breakdowns for runs performed on NERSC's Seaborg. As in Table 1, the lines are sorted such that runs with the same number of points per processor appear next to each other. For these runs, each processor is responsible for approximately 160^3 points.

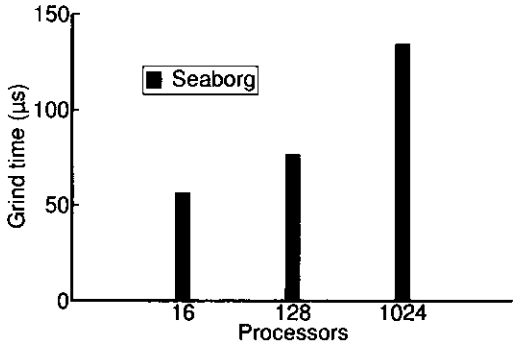

Figure 7: Grind times for the FFTW-based version of SCALLOP. For these runs, each processor is responsible for $96 * 192^2$ fine grid points.

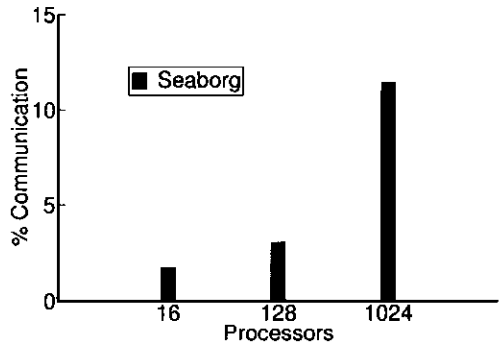

Figure 8: Communication overhead for the FFTW-based version of SCALLOP. For these runs, each processor is responsible for $96 * 192^2$ fine grid points.

running time and the time of the final solution of 2.15 to 0.24, or approximately 9.0. Our results show some overhead above the computational overhead inherent in the algorithm, mostly due to the tradeoffs we discussed in Section 4.4. In terms of the computation required for a serial solution, our running times represent 2.2 to 3.6 times T_{SID}, while we predicted a factor of 2.15.

On more than 64 processors, the final pass takes longer than expected, reducing the apparent computational overhead as a result. We have not yet determined the reason for this anomaly, but it may be related to the memory issues discussed previously.

5.4 Recent Results with FFTW-based SCALLOP

We have recently implemented the FFTW-based version of SCALLOP. As discussed earlier, the main benefit of using FFTW within SCALLOP is improved uniprocessor performance: the FFTW solver is approximately 8 times faster than the implementation of multigrid that it replaced.

In addition, this new version of SCALLOP has a smaller memory footprint, which also contributes to its better performance, especially for larger problem sizes.

Results for a limited number of runs of this new version of the code are shown in Table 3 and Figures 7 and 8. With this version of SCALLOP we see are able to scale a problem up from 16 to 1024 processors with only a factor of 2 increase in running time. The initial local solutions require significantly more time than we had expected, and we are currently investigating possible causes. A likely explanation is that the odd size grids created necessary for the calculation of the infinite domain boundary conditions are inefficient sizes for FFTW. It is also clear, however, that the boundary condition calculation itself is now dominating the total solution time, and we will need to find a more efficient algorithm for that part of SCALLOP.

Our analysis of the computation costs in Section 4.3 predicted that the best possible solution time for a serial solution would be equal to the solution time for the initial local calculations. By this metric, the FFTW-based version of SCALLOP shows an overhead of only 45% to 80%. This result falls short of the ideal 20% overhead we had predicted, but it is still quite reasonable.

5.5 Comparison to Other Options

Implementing a parallel finite-difference infinite domain solver is difficult, and, to our knowledge, there are no similar efforts with which we can compare ourselves. To get an approximate idea of the cost of communication for a more conventional method, we examined the scaling properties of a parallel FFT over a similar range of processors and problem sizes. We chose this comparison for two reasons. First, our latest implementation of SCALLOP uses FFT-based subdomain solvers, so an implementation using a parallel FFT would be the most natural competition. Second, vendor-supplied parallel FFT solvers exist and provide a fairer benchmark than code written by us.

At this point, we only have data at the extremes: on a single processor and on 512 processors of NERSC's Seaborg machine. A single real FFT problem of size 128^3 ran in approximately 1.1 seconds on one processor. (A full Poisson solution with Dirichlet boundary conditions requires both forward and reverse FFT solutions in addition to a modest amount of computation on the transformed data.) A 1024^3 problem on 512 processors (which represents 128^3 points per processor) required 5.5 seconds. This represents an an increase of running time by a factor of 5.0 in scaling from 1 to 512 processors. This estimate of the slowdown also ignores the additional cost of communication that would be necessary in a parallelized version of the calculation of the infinite domain boundary condition, which currently dominates the total calculation. The communication required for the boundary condition calculation itself would be quite significant, since the calculation requires each processor to receive $O(N^2)$ data from the other processors. In comparison to a slowdown of at least a factor of 5.0, the factor of 2.2 to 3.6 between the ideal serial cost and the actual cost of SCALLOP appears quite good.

6. CONCLUSIONS AND FUTURE WORK

We have presented a communication-tolerant approach for solving certain types of elliptic equations that trades off communication against computation. Our strategy reflects current technological constraints in which computation is a relatively cheap commodity compared with communication.

We described the design of the SCALLOP solver, which

Input Parameters				Times for Each Stage (seconds)					
P	q	C	N	Local	Reduction	Global	Boundary	Final	Total
16	4	3	384^3	130.1	0.53	60.9	2.95	3.70	198.8
128	8	6	768^3	187.7	1.89	67.3	6.42	4.42	270.7
1024	16	12	1536^3	265.4	31.74	141.5	22.72	4.63	474.9

Table 3: Input parameters and timing breakdowns for runs of the FFTW-based version of SCALLOP, performed on NERSC's Seaborg. For these runs, each processor is responsible for $96 * 192^2$ fine grid points.

realizes our strategy. In practice, the performance of SCALLOP matches the expectations of our performance model quite well. Communication costs are on the order of a few percent of the total running time, and total computation time can be predicted by the time required for the initial fine grid calculations. The benefit of little communication comes at the expense of added computation, but this overhead is reasonable, and more importantly does not grow much with the number of processors.

Most of the computational overhead in the multigrid version of SCALLOP is due to the extra computation required for the 27-point stencil in our multigrid solver. This large portion of the overhead is removed when the subdomain solutions are computed using an FFT-based solver, and the preliminary tests which we have conducted with a solver based on FFTW appear promising. In addition to reducing the overhead involved in using the 27-point stencil, the use of FFTW also greatly improves the raw performance of SCALLOP. Currently the largest portion of computational effort is devoted to calculating the infinite domain boundary condition. We are investigating ways to optimize this computation.

One important benefit of SCALLOP over other possible parallel solutions to the Poisson equation, especially purely FFT-based methods, is the ability to modify our algorithm and create an adaptive method [8]. A simple two-level adaptive method can be constructed whereby much computation is avoided in regions of zero charge. The computational overhead incurred by our method may then quickly become negligible for problems in which the charge field covers only a fraction of the computational domain.

As stated earlier, the most important limitation in the current implementation of SCALLOP is the extra overlap introduced on the fine grid subdomains. We believe this limitation can be removed by generating the outlying coarse grid data without calculating the corresponding find grid data. This improvement will allow SCALLOP to scale up to many thousands of processors, filling the largest machines available today and in the next several years.

7. ACKNOWLEDGMENTS

Greg Balls and Scott Baden were supported by the National Partnership for Advanced Computational Infrastructure (NPACI) under NSF contract ACI9619020. Phillip Colella's research is supported by the Mathematical, Information, and Computational Sciences Division of the Office of Science, U.S. Department of Energy under contract number DE-AC03-76SF00098. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC03-76SF-00098. Scott Baden dedicates this portion of his work to Ruben Perlman (1909-2002). Thanks also to Ryan Szypowski for his help with data on FFT solvers. SCALLOP is publicly available at `http://www-cse.ucsd.edu/groups/hpcl/scg/scallop/`.

8. REFERENCES

[1] C. R. Anderson. A method of local corrections for computing the velocity field due to a distribution of vortex blobs. *Journal of Computational Physics*, 62:111–123, 1986.

[2] S. B. Baden and S. J. Fink. Communication overlap in multi-tier parallel algorithms. In *Proc. of SC '98*, Orlando, Florida, November 1998.

[3] S. B. Baden and S. J. Fink. A programming methodology for dual-tier multicomputers. *IEEE Trans. Software Engineering*, 26(3):212–26, March 2000.

[4] S. B. Baden and D. Shalit. Performance tradeoffs in multi-tier formulation of a finite difference method. In *Proc. 2001 International Conference on Computational Science*, San Francisco, CA, May 2001.

[5] G. T. Balls. *A Finite Difference Domain Decomposition Method Using Local Corrections for the Solution of Poisson's Equation.* PhD thesis, University of California, Berkeley, 1999.

[6] G. T. Balls and P. Colella. A finite difference domain decomposition method using local corrections for the solution of Poisson's equation. *Journal of Computational Physics*, 180(1):25–53, July 2002.

[7] R. Bank and M. Holst. A new paradigm for parallel adaptive meshing algorithms. *SIAM Journal on Scientific Computing*, 22(4):1411–1443, 2000.

[8] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82:64–84, 1989.

[9] W. L. Briggs. *A Multigrid Tutorial.* SIAM, 1987.

[10] S. J. Fink, S. R. Kohn, and S. B. Baden. Efficient run-time support for irregular block-structured applications. *Journal of Parallel and Distributed Computing*, 50(1-2):61–82, April-May 1998.

[11] M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *ICASSP Conference Proceedings*, volume 3, pages 1381–1384. ICASSP, 1998.

[12] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *The Journal of Computational Physics*, 73:325–348, 1987.

[13] M. Holst. Applications of domain decomposition and partition of unity methods in physics and geometry. In I. Herrera, D. E. Keyes, O. B. Widlund, and R. Yates, editors, *Proceedings of the Fourteenth International*

Conference on Domain Decomposition Methods, January 2002.

[14] R. A. James. The solution of Poisson's equation for isolated source distributions. *Journal of Computational Physics*, 25(2):71–93, October 1977.

[15] K. Lackner. Computation of ideal MHD equilibria. *Computer Physics Communications*, 12(1):33–44, 1976.

[16] B. F. Smith and O. B. Widlund. A domain decomposition algorithm using a heirarchical basis. *SIAM Journal on Scientific and Statistical Computing*, 11(6):1212–1220, November 1990.

[17] A. Sohn and R. Biswas. Communication studies of DMP and SMP machines. Technical Report NAS-97-004, NAS, 1997.

[18] A. K. Somani and A. M. Sansano. Minimizing overhead in parallel algorithms through overlapping communication/computation. Technical Report 97-8, ICASE, February 1997.