Journal of Computational Physics **169,** 151–192 (2001)
doi:10.1006/jcph.2001.6720, available online at http://www.idealibrary.com on IDEAL®

A Fast Parallel Algorithm for the Poisson Equation on a Disk

Leonardo Borges* and Prabir Daripa†,[1]

**Institute for Scientific Computation and †Department of Mathematics, Texas A&M University, College Station, Texas 77843*
E-mail: prabir.daripa@math.tamu.edu

Received January 10, 2000; revised September 6, 2000

A parallel algorithm for solving the Poisson equation with either Dirichlet or Neumann conditions is presented. The solver follows some of the principles introduced in a previous fast algorithm for evaluating singular integral transforms by Daripa *et al.* Here we present recursive relations in Fourier space together with fast Fourier transforms which lead to a fast and accurate algorithm for solving Poisson problems within a unit disk. The algorithm is highly parallelizable and our implementation is virtually architecture-independent. Theoretical estimates show good parallel scalability of the algorithm, and numerical results show the accuracy of the method for problems with sharp variations on inhomogeneous term. Finally, performance results for sequential and parallel implementations are presented. © 2001 Academic Press

1. INTRODUCTION

The Poisson equation is one of the fundamental equations in mathematical physics which, for example, governs the spatial variation of a potential function for given source terms. The range of applications covers magnetostatic problems to ocean modeling. Fast, accurate, and reliable numerical solvers play a significant role in the development of applications for scientific problems. In this paper, we present efficient sequential and parallel algorithms for solving the Poisson equation on a disk using Green's function method.

A standard procedure for solving the Poisson equation using Green's function method requires evaluation of volume integrals which define contribution to the solution resulting from source terms. However, the complexity of this approach in two-dimensions is $\mathcal{O}(N^4)$ for a N^2 net of grid points which makes the method prohibitive for large-scale problems. Here, we expand the potential in terms of Fourier series by deriving radius-dependent Fourier coefficients. These Fourier coefficients can be obtained by recursive relations

[1] To whom correspondence should be addressed.

which only utilize one-dimensional integrals in the radial directions of the domain. Also, we show that these recursive relations make it possible to define high-order numerical integration schemes in the radial directions without taking additional grid points. Results are more accurate because the algorithm is based on exact analysis. The method presents high accuracy even for problems with sharp variations on inhomogeneous term. On a single processor machine, the method has a theoretical computational complexity $\mathcal{O}(N^2 \log_2 N)$ or equivalently $\mathcal{O}(\log_2 N)$ per point which represents substantial savings in computational time when compared with the complexity $\mathcal{O}(N^2)$ for standard procedures.

The basic philosophy mentioned above has been applied previously in the context of developing fast algorithms for evaluations of singular integrals [8] in the complex plane. The mathematical machinery behind this philosophy is applied in Section 2 of this paper for the presentation of a theorem (Theorem 2.1) which outlines the fast algorithm for solving the Poisson equation in the real plane. The derivation of this theorem is straightforward and closely follows the analogous development elsewhere [7], except for the fact that it does not use the tools of single complex variable theory (such as Cauchy's residue theorem) as in Daripa and Mashat [8], and it involves a different equation.

We must state right at the outset that our main goal in this paper is the use of this theorem for the development of the very efficient serial and parallel algorithms and testing the performance of these algorithms on a host of problems. Thus, we could have merely stated Theorem 2.1 without its derivation, but the presentation of the derivation is necessary for completeness. Also, it is necessary for the purpose of extension of this fast algorithm to higher dimensions and to arbitrary domains which we will address in a forthcoming paper. It is worth pointing out that the statement of Theorem 2.1 follows the general format of a theorem recently introduced by the second author and his collaborators [8] in the context of singular integral transforms. Thus, part of this paper builds upon our earlier work.

We address the parallelization of the algorithm in some detail which is one of the main thrusts of this paper. The resulting algorithm is very scalable because of the fact that communication costs are independent of the number of annular regions taken for the domain discretization. It means that an increasing number of sample points in the radial direction does not increase overheads resulting from interprocessor coordination. Message lengths depend only on the number of Fourier coefficients in use. Communication is performed in a linear path configuration which allows overlapping of computational work simultaneously with data-exchanges. This overlapping guarantees that the algorithm is well suited for distributed and shared memory architectures. Here our numerical experiments show the good performance of the algorithm in a shared memory computer. Related work [2, 3] shows the suitability for distributed memory. It makes the algorithm architecture-independent and portable. Moreover, the mathematical formulation of the parallel algorithm presents a high level of data locality, which results in an effective use of cache.

At this point, it is worth mentioning that there now exists a host of fast parallel Poisson solvers based on various principles including the use of FFT and fast multipole method [5, 6, 16, 18]. The fast solver of this paper is based on Theorem 2.1, which is derived through exact analyses and properties of convolution integrals involving Green's function. Thus, this solver is very accurate because of these exact analyses which is demonstrated on a host of problems. Moreover, this solver is easy to implement and has a very low constant hidden behind the order estimate of the complexity of the algorithm. This gives this solver an advantage over many other solvers with similar complexity, which usually have a high value of this hidden constant. Furthermore, this solver can be very optimal for solving certain classes of problems involving circular domains or overlapped circular domains. This solver can also be used in arbitrary domains via spectral domain embedding technique. This work is currently in progress.

In Section 2 we begin presenting the mathematical preliminaries of the algorithm and deriving the recursive relations. In Section 3 we describe the sequential implementation and two variants of the integration scheme. In Section 4 we introduce the parallel implementation and its theoretical analysis. In Section 5 we present and discuss the numerical results on several test problems for accuracy and performance of the algorithm. Finally, in Section 6 we summarize our results.

2. MATHEMATICAL PRELIMINARIES

In this section we introduce the mathematical formulation for a fast solver for Dirichlet problems. Also, recursive relations are presented, leading to an efficient numerical algorithm. Finally, the mathematical formulation is extended to Neumann problems. Proofs are given in the Appendix.

2.1. *The Dirichlet Problem and Its Solution on a Disk*

Consider the Dirichlet problem of the Poisson equation

$$\begin{aligned} \Delta u &= f \quad \text{in } B \\ u &= g \quad \text{on } \partial B, \end{aligned} \tag{1}$$

where $B = B(0, R) = \{x \in IR^2 : |x| < R\}$. Specifically, let v satisfy

$$\Delta v = f \qquad \text{in } B, \tag{2}$$

and w be the solution of the homogeneous problem

$$\begin{aligned} \Delta w &= 0 \quad \text{in } B \\ w &= g - v \quad \text{on } \partial B. \end{aligned} \tag{3}$$

Thus, the solution of the Dirichlet problem (1) is given by

$$u = v + w. \tag{4}$$

A principal solution of Eq. (2) can be written as

$$v(x) = \int_B f(\eta)\, G(x, \eta)\, d\eta, \qquad x \in B, \tag{5}$$

where $G(x, \eta)$ is the free-space Green's function for the Laplacian given by

$$G(x, \eta) = \frac{1}{2\pi} \log|x - \eta|. \tag{6}$$

To derive a numerical method based on Eq. (5), the interior of the disk $B(0, R)$ is divided into a collection of annular regions. The use of quadrature rules to evaluate (5) incurs in poor accuracy for the approximate solution. Moreover, the complexity of a quadrature method is $\mathcal{O}(N^4)$ for a N^2 net of grid points. For large problem sizes it represents prohibitive costs in computational time. Here we expand $v(\cdot)$ in terms of Fourier series by deriving radius-dependent Fourier coefficients of $v(\cdot)$. These Fourier coefficients can be obtained by recursive relations which only utilize one-dimensional integrals in the radial direction. The fast algorithm is embedded in the following theorem.

THEOREM 2.1. *If $u(r,\alpha)$ is the solution of the Dirichlet problem* (1) *for $x = re^{i\alpha}$ and $f(re^{i\alpha}) = \sum_{n=-\infty}^{\infty} f_n(r)e^{in\alpha}$, then the nth Fourier coefficient $u_n(r)$ of $u(r,\cdot)$ can be written as*

$$u_n(r) = v_n(r) + \left(\frac{r}{R}\right)^{|n|}(g_n - v_n(R)), \qquad 0 < r < R, \tag{7}$$

where g_n are the Fourier coefficients of g on ∂B, and

$$v_n(r) = \int_0^r p_n(r,\rho)\,d\rho + \int_r^R q_n(r,\rho)\,d\rho, \tag{8}$$

with

$$p_n(r,\rho) = \begin{cases} \rho \log r \;\, f_0(\rho), & n = 0, \\ \frac{-\rho}{2|n|}\left(\frac{\rho}{r}\right)^{|n|} f_n(\rho), & n \neq 0, \end{cases} \tag{9}$$

and

$$q_n(r,\rho) = \begin{cases} \rho \log \rho f_0(\rho), & n = 0, \\ \frac{-\rho}{2|n|}\left(\frac{r}{\rho}\right)^{|n|} f_n(\rho), & n \neq 0. \end{cases} \tag{10}$$

2.2. *Recursive Relations of the Algorithm*

Despite the fact that the above theorem presents the mathematical foundation of the algorithm, an efficient implementation can be devised by making use of recursive relations to perform the integrations in (8). Consider the disk $\overline{B(0,R)}$ discretized by $N \times M$ lattice points with N equidistant points in the angular direction and M distinct points in the radial direction. Let $0 = r_1 < r_2 < \cdots < r_M = R$ be the radii defined on the discretization. Theorem 2.1 leads to the following corollaries.

COROLLARY 2.1. *It follows from* (8) *and* (10) *that $v_n(0) = 0$ for $n \neq 0$.*

COROLLARY 2.2. *Let $0 = r_1 < r_2 < \cdots < r_M = R$, and*

$$C_n^{i,j} = \int_{r_i}^{r_j} \frac{\rho}{2n}\left(\frac{r_j}{\rho}\right)^n f_n(\rho)\,d\rho, \qquad n < 0, \tag{11}$$

$$D_n^{i,j} = -\int_{r_i}^{r_j} \frac{\rho}{2n}\left(\frac{r_i}{\rho}\right)^n f_n(\rho)\,d\rho, \qquad n > 0. \tag{12}$$

If for $r_j > r_i$, *we define*

$$\begin{aligned} v_n^-(r_1) &= 0, && n < 0, \\ v_n^-(r_j) &= \left(\frac{r_j}{r_i}\right)^n v_n^-(r_i) + C_n^{i,j}, && n < 0, \end{aligned} \tag{13}$$

and

$$\begin{aligned} v_n^+(r_M) &= 0, && n > 0 \\ v_n^+(r_i) &= \left(\frac{r_i}{r_j}\right)^n v_n^+(r_j) + D_n^{i,j}, && n > 0, \end{aligned} \tag{14}$$

then for $i = 1, \ldots, M$ *we have*

$$v_n(r_i) = \begin{cases} v_n^-(r_i) + \overline{v_{-n}^+(r_i)}, & n < 0, \\ v_n^+(r_i) + \overline{v_{-n}^-(r_i)}, & n > 0. \end{cases} \tag{15}$$

COROLLARY 2.3. *Let* $0 = r_1 < r_2 < \cdots < r_M = R$, *and add* $n = 0$ *to the definitions in Corollary* 2.2 *as*

$$C_0^{i,j} = \int_{r_i}^{r_j} \rho\; f_0(\rho)\, d\rho \qquad \textit{and} \qquad D_0^{i,j} = \int_{r_i}^{r_j} \rho \log \rho\; f_0(\rho)\, d\rho, \tag{16}$$

then given $l = 1, \ldots, M$ *we have*

$$v_n(r_l) = \begin{cases} \log r_l \sum_{i=2}^{l} C_0^{i-1,i} + \sum_{i=l}^{M-1} D_0^{i,i+1}, & \textit{for } n = 0, \\ \sum_{i=2}^{l} \left(\frac{r_l}{r_i}\right)^n C_n^{i-1,i} + \sum_{i=l}^{M-1} \left(\frac{r_i}{r_l}\right)^n \overline{D_{-n}^{i,i+1}}, & \textit{for } n < 0, \\ \sum_{i=l}^{M-1} \left(\frac{r_l}{r_i}\right)^n D_n^{i,i+1} + \sum_{i=2}^{l} \left(\frac{r_i}{r_l}\right)^n \overline{C_{-n}^{i-1,i}}, & \textit{for } n > 0. \end{cases} \tag{17}$$

It is important to emphasize that M distinct points $r_1, \ldots, r_M$ need not to be equidistant. Therefore, the fast algorithm can be applied on domains that are nonuniform in the radial direction. This anisotropic grid refinement may at first seem unusual for elliptic problems. Even though it is true that isotropic grid refinement is more common with solving elliptic equations, there are exceptions to the rule, in particular with a hybrid method such as ours (Fourier in one direction and finite difference in the other). Since Fourier methods are spectrally accurate, grid refinement along the circumferential direction beyond a certain optimal level may not always offer much advantage. This is well known because of the exponential decay rate of Fourier coefficients for a classical solution (c^∞ function). This fact will be exemplified later in Example 1 (see Table I in Section 5.1) where we show that to get more accurate results one needs to increase the number of annular regions without increasing the number of Fourier coefficients participating in the calculation (i.e. anisotropic grid refinement with more grids in the radial direction than in the the circumferential direction is more appropriate for that problem).

TABLE I
Problem 1—Relative Errors in Norm $\|\cdot\|_\infty$ Using Distinct Values for N and M

Relative errors for Problem 1 (Dirichlet)						
$_N\backslash^M$	64	128	256	512	1024	2048
64	2.6e-5	6.4e-6	1.6e-6	3.9e-7	9.8e-8	2.5e-8
128	2.6e-5	6.4e-6	1.6e-6	3.9e-7	9.8e-8	2.5e-8
256	2.6e-5	6.4e-6	1.6e-6	3.9e-7	9.8e-8	—
512	2.6e-5	6.4e-6	1.6e-6	3.9e-7	—	—
1024	2.6e-5	6.4e-6	1.6e-6	—	—	—
2048	2.6e-5	6.4e-6	—	—	—	—

Note. The number of circles M is the dominant parameter.

2.3. *The Neumann Problem and Its Solution on a Disk*

The same results obtained for solving the Dirichlet problem can be generalized for the Neumann problem by expanding the derivative of the principal solution v in (5). Consider the Neumann problem

$$\begin{aligned} \Delta u &= f \qquad \text{in } B \\ \frac{\partial u}{\partial n} &= \psi \qquad \text{on } \partial B. \end{aligned} \tag{18}$$

The analogous of Theorem 2.1 for the Neumann problem is given by Theorem 2.2.

THEOREM 2.2. *If $u(r,\alpha)$ is the solution of the Neumann problem* (18) *for $x = re^{i\alpha}$ and $f(re^{i\alpha}) = \sum_{n=-\infty}^{\infty} f_n(r)e^{in\alpha}$, then the nth Fourier coefficient $u_n(r)$ of $u(r,\cdot)$ can be written as*

$$\begin{aligned} u_0(r) &= v_0(r) + \varphi_0, && n = 0 \\ u_n(r) &= v_n(r) + \left(\frac{r}{R}\right)^{|n|}\left(\frac{R}{|n|}\psi_n + v_n(R)\right), && n \neq 0, \end{aligned} \tag{19}$$

where ψ_n are the Fourier coefficients of ψ on ∂B, v_n are defined as in Theorem 2.1, *and φ_0 is the parameter which sets the additive constant for the solution.*

3. THE SEQUENTIAL ALGORITHM

An efficient implementation of the algorithm embedded in Theorem 2.1 is derived from Corollary 2.2. It defines recursive relations to obtain the Fourier coefficients v_n in (7) based on the sign of the index n of v_n. In the description of the algorithm, we address the coefficients with index values $n \leq 0$ as *negative modes*, and the ones with index values $n \geq 0$ as *positive modes*. Equation (13) shows that negative modes are built up from the smallest radius r_1 toward the largest radius r_M. Conversely, Eq. (14) constructs positive modes from r_M toward r_1. Figure 1 presents the resulting sequential algorithm for the Dirichlet problem. The counterpart algorithm for the Neumann problem similarly follows from Theorem 2.2 and Corollary 2.2.

Notice that Algorithm 3.1 requires the radial one-dimensional integrals $C_n^{i,i+1}$ and $D_n^{i,i+1}$ to be calculated between two successive points (indexed by i and $i+1$) on a given radial

ALGORITHM 3.1. (Sequential Algorithm for the Dirichlet Problem on a Disk). Given M, N, the grid values $f(r_l e^{2\pi ik/N})$ and the boundary conditions $g(Re^{2\pi ik/N})$, $l \in [1, M]$, and $k \in [1, N]$, the algorithm returns the values $u(r_l e^{2\pi ik/N})$, $l \in [1, M]$, $k \in [1, N]$ of the solution for the Dirichlet problem (1).

1. Compute the Fourier coefficients $f_n(r_l)$, $n \in [-N/2, N/2]$, for M sets of data at $l \in [1, M]$, and the Fourier coefficients g_n on ∂B.
2. For $i \in [1, M-1]$, compute the radial one-dimensional integrals $C_n^{i,i+1}$, $n \in [-N/2, 0]$ as defined in (11) and (16); and compute $D_n^{i,i+1}$, $n \in [0, N/2]$ as defined in (12) and (16).
3. Compute coefficients $v_n^-(r_l)$ for each of the negative modes $n \in [-N/2, 0]$ as defined in (13) and (17):

   (a) Set $v_n^-(r_1) = 0$ for $n \in [-N/2, 0]$.
   (b) For $l = 2, \ldots, M$

$$v_n^-(r_l) = \left(\frac{r_l}{r_{l-1}}\right)^n v_n^-(r_{l-1}) + C_n^{l-1,l}, \qquad n \in [-N/2, 0].$$

4. Compute coefficients $v_n^+(r_l)$ for each of the positive modes $n \in [0, N/2]$ as defined in (14) and (17):

   (a) Set $v_n^+(r_M) = 0$ for $n \in [0, N/2]$.
   (b) For $l = M-1, \ldots, 1$

$$v_n^+(r_l) = \left(\frac{r_l}{r_{l+1}}\right)^n v_n^+(r_{l+1}) + D_n^{l,l+1}, \qquad n \in [0, N/2].$$

5. Combine coefficients v_n^+ and v_n^- as defined in (15) and (17):
   For $l = 1, \ldots, M$

$$v_0(r_l) = \log r_l v_0^-(r_l) + v_0^+(r_l).$$
$$v_n(r_l) = \overline{v_{-n}(r_l)} = v_n^-(r_l) + \overline{v_{-n}^+(r_l)}, \qquad n \in [-N/2, -1].$$

6. Apply the boundary conditions as defined in (7):
   For $l = 2, \ldots, M$

$$u_n(r_l) = v_n(r_l) + \left(\frac{r_l}{R}\right)^{|n|}(g_n - v_n(R)), \qquad n \in [-N/2, N/2].$$

7. Compute $u(r_l e^{2\pi ik/N}) = \sum_{n=-N/2}^{N/2} u_n(r_l) e^{2\pi ikn/N}$, $k \in [1, N]$, for each radius r_l, $l \in [1, M]$.

FIG. 1. Description of the sequential algorithm for the Dirichlet problem.

direction (defined by n). One possible numerical method for obtaining these integrals would be to use the trapezoidal rule. However, the trapezoidal rule presents an error of quadratic order. One natural approach to increase the accuracy of the numerical integration would be to add auxiliary points between the actual points of the discretization of the domain to allow higher-order integration methods to obtain $C_n^{i,i+1}$ and $D_n^{i,i+1}$. This approach presents two major disadvantages: (1) it substantially increases computational costs of the algorithm because the fast Fourier transforms in step 1 of Algorithm 3.1 must also be performed for all the new circles of extra points added for the numerical integration; (2) in practical problems the values for function f may be available only on a finite set of points, which constrains the data to a fixed discretization of the domain, and no extra grid points can be added to increase the accuracy of the solver.

Here, we increase the accuracy of the radial integrals by redefining steps 2, 3, and 4 of Algorithm 3.1 based on the more general recurrences presented in Eqs. (13) and (14). Terms $C_n^{i,i+1}$ and $D_n^{i,i+1}$ are evaluated only using two consecutive points. In fact, for the case $n < 0$ one can apply the trapezoidal rule for (11) leading to

$$C_n^{i,i+1} = \frac{(\delta r)^2}{4n}\left(i\left(\frac{i}{i+1}\right)^{-n} f_n(r_i) + (i+1)\, f_n(r_{i+1})\right) \tag{20}$$

for a uniform discretization, where $r_i = (i-1)\delta r$. It corresponds to the trapezoidal rule applied between circles r_i and r_{i+1}. A similar equation holds for $D_n^{i,i+1}$. By evaluating terms of the form $C_n^{i-1,i+1}$ and $D_n^{i-1,i+1}$, three consecutive points can be used in the radial direction. It allows the use of the Simpson's rule

$$C_n^{i-1,i+1} = \frac{(\delta r)^2}{6n}\left((i-1)\left(\frac{i-1}{i+1}\right)^{-n} f_n(r_{i-1}) + 4i\left(\frac{i}{i+1}\right)^{-n} f_n(r_i) + (i+1)\, f_n(r_{i+1})\right), \tag{21}$$

which increases the accuracy of the method. In the algorithm, it corresponds to redefining step 3 for $n < 0$ as

$$\begin{aligned} v_n^-(r_1) &= 0, \\ v_n^-(r_2) &= C_n^{1,2}, \\ v_n^-(r_l) &= \left(\frac{r_l}{r_{l-2}}\right)^n v_n^-(r_{l-2}) + C_n^{l-2,l}, \qquad l = 3, \ldots, M, \end{aligned}$$

and step 4 for $n > 0$ as

$$\begin{aligned} v_n^+(r_M) &= 0, \\ v_n^+(r_{M-1}) &= D_n^{M-1,M}, \\ v_n^+(r_l) &= \left(\frac{r_l}{r_{l+2}}\right)^n v_n^+(r_{l+2}) + D_n^{l,l+2}, \quad l = M-2, \ldots, 1. \end{aligned}$$

It results in an integration scheme applied between three successive circles, say r_{i-1}, r_i and r_{i+1}, with computational costs practically similar to the trapezoidal rule but with higher accuracy. The above Simpson's rule presents an error formula of fourth order in the domain of length $2\delta r$. For sufficiently smooth solutions, it allows cubic convergence in δr as the numerical results show in Section 5.

4. THE PARALLEL ALGORITHM

Current resources in high-performance computing can be divided into two major models: distributed and shared memory architectures. The design of a parallel and portable application must attempt to deliver a high user-level performance in both architectures. In this section, we present a parallel implementation suited for the distributed and shared models. Although we conduct our presentation using the message-passing model, this model can also be employed to describe interprocessor coordination: Higher communication overhead corresponds to larger data dependency in the algorithm, which results in loss of data locality. Even though shared memory machines have support for coherence, good performance requires locality of reference because of the memory hierarchy. Synchronization and true sharing must be minimized [1]. Efficient parallelized codes synchronize infrequently and have little true sharing [22]. Therefore, a good parallelization requires no communication whenever possible. Using the data decomposition, which allows lower communication cost, also improves the data locality. The numerical results in Section 5 are obtained in a shared memory architecture. The performance of the parallel algorithm on distributed memory systems was addressed in [2]. There a variant of the algorithm was used for fast and accurate evaluation of singular integral transforms.

The recursive relations in Corollary 2.2 are very appropriate to a sequential algorithm. However, they may represent a bottleneck in a parallel implementation. In this section we use the results presented in Corollary 2.3 to devise an efficient parallel solver for the Poisson equation. Theoretical estimates for the performance of the parallel version of the algorithm are given below. We also show that this parallel solver has better performance characteristics than an implementation based on Corollary 2.2. Finally, we compare our parallel algorithm with other Poisson solvers.

4.1. *Parallel Implementation*

The fast algorithm for the Poisson equation requires multiple fast Fourier transforms (FFT) to be performed. There are distinct strategies to solve multiple FFTs in parallel systems [4, 11]. In [2] we have shown that an improved implementation of parallel calls to sequential FFTs is the best choice for the fast algorithm. For the sake of a more clear explanation, let P be the number of available processors and M be a multiple of P. Data partitioning is defined by distributing the circles of the domain into P groups of consecutive circles so that each processor contains the grid points for M/P circles. To obtain a more compact notation we define

$$\gamma(j) = jM/P.$$

Given P processors p_j, $j = 0, \ldots, P-1$, data is distributed so that processor p_j contains the data associated with the grid points $r_l e^{2\pi ik/N}$, $k \in [1, N]$, and $l \in [\gamma(j)+1, \gamma(j+1)]$. Figure 2 exemplifies the data distribution for a system with three processors ($P = 3$).

One optimized version of a sequential N-point FFT algorithm is available on each processor: Multiple Fourier transforms of the same length are performed simultaneously. The M sequences of values assumed on the N grid points belonging to a circle are distributed between processors so that each one performs one unique call to obtain M/P FFT transforms. Overall, the FFT transforms contribute the most to the computational cost of the algorithm and the above data-locality allows the intensive floating point operations to be

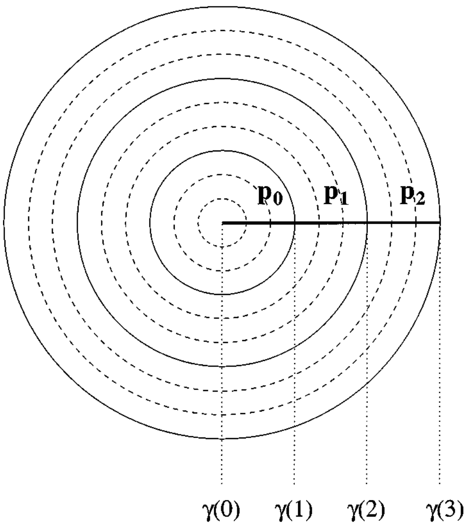


FIG. 2. Data distribution for the parallel version of the fast algorithm.

performed locally and concurrently. Thus, each FFT can be evaluated in place, without communication. Other strategies for solving the multiple FFTs required in the algorithm are discussed in [2].

Although Corollary 2.2 is formulated for the generic case $r_j > r_i$, the results in Corollary 2.3 only require consecutive radii (i.e., terms of the form $C_n^{l-1,l}$ and $D_n^{l,l+1}$, $l \in [\gamma(j)+1, \gamma(j+1)]$) in processor p_j. Therefore, the numerical integration for Eqs. (11), (12), and (16) can be performed locally if one guarantees that all necessary data is available within the processor. Notice that p_j already evaluates the Fourier coefficients $f_n(r_l)$, $l \in [\gamma(j)+1, \gamma(j+1)]$. In the case of a numerical integration based on the trapezoidal rule (20), only the Fourier coefficients for $l = jM/P$ and $l = (j+1)M/P + 1$ must be added to the set of known Fourier coefficients for processor p_j. That is, if the initial data is overlapped so that each processor evaluates coefficients for radii r_l, $l \in [\gamma(j), \gamma(j+1)+1]$, there is no need for communication. Similarly, if the modified Simpson's rule (21) is employed, processor p_j only needs to evaluate coefficients for radii r_l, $l \in [\gamma(j)-1, \gamma(j+1)+2]$. The number of circles whose data overlap between any two neighbor processors remains fixed regardless of the total number of processors in use. Consequently, this strategy does not compromise the scalability of the algorithm.

Algorithm 3.1 was described based on the inherently sequential iterations from Corollary 2.2 which are more suitable for a sequential implementation. In the case of a parallel algorithm, an even distribution of computational load is obtained by splitting the computational work when performing recurrences (16) and (17) as described in Corollary 2.3. We evaluate iterative sums q_l, $l \in [\gamma(j), \gamma(j+1)]$, concurrently on all processors p_j, $j = 0, \ldots, P-1$, as follows. For the case $n \leq 0$ let

$$\begin{aligned} q^-_{\gamma(j)}(n) &= 0, \\ q^-_l(n) &= \left(\frac{r_{l+1}}{r_l}\right)^n \left(q^-_{l-1}(n) + C_n^{l-1,l}\right), \quad l = \gamma(j)+1, \ldots, \gamma(j+1), \end{aligned} \tag{22}$$

where we have defined $r_{M+1} = 1$, and for the case $n \geq 0$ let

$$\begin{aligned} q^+_{\gamma(j+1)+1}(n) &= 0, \\ q^+_l(n) &= \left(\frac{r_{l-1}}{r_l}\right)^n \left(q^+_{l+1}(n) + D^{l,l+1}_n\right), \quad l = \gamma(j+1), \ldots, \gamma(j)+1. \end{aligned} \tag{23}$$

Since coefficients $C^{i-1,i}_n (n \leq 0)$ and $D^{i,i+1}_n (n \geq 0)$ are already stored in processor p_j when $i \in [\gamma(j)+1, \gamma(j+1)]$, partial sums t^-_j and t^+_j can be computed locally in processor p_j. In [2] we have shown that the above computations can be used to define the following *partial sums* for each processor p_j:

$$\begin{aligned} t^-_j(n) &= q^-_{\gamma(j+1)}(n), \qquad n \leq 0, \\ t^+_j(n) &= q^+_{\gamma(j)+1}(n), \qquad n \geq 0. \end{aligned}$$

Moreover, it follows from (22) and (23) that for $n \leq 0$

$$\begin{aligned} t^-_0(n) &= r^n_{\gamma(1)+1} \sum_{i=2}^{\gamma(1)} \left(\frac{1}{r_i}\right)^n C^{i-1,i}_n, \\ t^-_j(n) &= r^n_{\gamma(j+1)+1} \sum_{i=\gamma(j)+1}^{\gamma(j+1)} \left(\frac{1}{r_i}\right)^n C^{i-1,i}_n, \end{aligned}$$

and for $n \geq 0$

$$\begin{aligned} t^+_{P-1}(n) &= r^n_{\gamma(P_{-1})} \sum_{i=\gamma(P-1)+1}^{M-1} \left(\frac{1}{r_i}\right)^n D^{i,i+1}_n, \\ t^+_j(n) &= r^n_{\gamma(j)} \sum_{i=\gamma(j)+1}^{\gamma(j+1)} \left(\frac{1}{r_i}\right)^n D^{i,i+1}_n. \end{aligned}$$

Although sums as described above may seem to produce either fast overflows or fast underflows for large absolute values of n, partial sums t^-_j and t^+_j can be obtained by performing very stable computations (22) and (23) as described in [2]. Therefore, the algorithm proceeds by performing the partial sums in parallel as represented in Fig. 3.

To combine partial sums t_j and t^+_j evaluated on distinct processors, we define the *accumulated sums* $\hat{s}^-_j$ and $\hat{s}^+_j$, $j = 0, \ldots, P-1$. For $n \leq 0$ let

$$\begin{aligned} \hat{s}^-_0(n) &= t^-_0(n), \\ \hat{s}^-_j(n) &= \left(\frac{r_{\gamma(j+1)+1}}{r_{\gamma(j)+1}}\right)^n \hat{s}^-_{j-1}(n) + t^-_j, \end{aligned} \tag{24}$$

and for $n \geq 0$

$$\begin{aligned} \hat{s}^+_{P-1}(n) &= t^+_{P-1}(n), \\ \hat{s}^+_j(n) &= \left(\frac{r_{\gamma(j)}}{r_{\gamma(j+1)}}\right)^n \hat{s}^+_{j+1}(n) + t^+_j. \end{aligned} \tag{25}$$

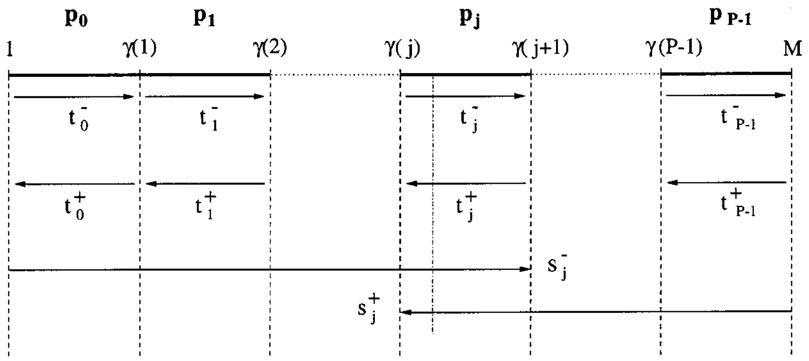


FIG. 3. Sums are evenly distributed across processors.

Therefore we have a recursive method to accumulate partial sums t_j^- and t_j^+ computed in processors p_j. Accumulated sums $\hat{s}_j^-$ and $\hat{s}_j^+$ can now be used to calculate coefficients C_n and D_n locally on each processor. Given a fixed radius r_l, the associated data belongs to the processor p_j such that $l \in [\gamma(j)+1, \gamma(j+1)]$. Computations in p_j only make use of accumulated sums from neighbor processors. For $n \leq 0$, local updates in processor p_0 are performed as described in Corollary 2.2. Local updates in processors p_j, $j = 1, \ldots, P-1$, use the accumulated sums $\hat{s}_{j-1}^-$ from the previous processor when obtaining terms v_n^- as defined in Eq. (13):

$$\begin{aligned} v_n^-(r_{\gamma(j)+1}) &= \hat{s}_{j-1}^-(n) + C_n^{\gamma(j),\gamma(j)+1} \\ v_n^-(r_l) &= \left(\frac{r_l}{r_{l-1}}\right)^n v_n^-(r_{l-1}) + C_n^{l-1,l}. \end{aligned} \tag{26}$$

For $n \geq 0$, local updates in processor p_{P-1} are also performed as described in Corollary 2.2. Local updates in processors p_j, $j = 0, \ldots, P-2$ use the accumulated sum $\hat{s}_{j+1}^+$ from the next processor to obtain terms v_n^+ from Eq. (14):

$$\begin{aligned} v_n^+(r_{\gamma(j+1)}) &= -\hat{s}_{j+1}^+(n) - D_n^{\gamma(j+1),\gamma(j+1)+1} \\ v_n^-(r_l) &= \left(\frac{r_l}{r_{l+1}}\right)^n v_n^+((r_{l+1}) + D_n^{l,l+1}. \end{aligned} \tag{27}$$

The advantage of using Eqs. (26) and (27) over original recurrences in Corollary 2.2 is that accumulated sums $\hat{s}_j^-$ and $\hat{s}_j^+$ are obtained using partial sums t_j^- and t_j^+. Since all partial sums can be computed locally (without message passing) and hence simultaneously, the sequential bottleneck of the original recurrences is removed. The only sequential component in this process is the message-passing mechanism to accumulate the partial sums.

The next step in the algorithm consists of combining coefficients v_n^+ and v_n^- to obtain the component v_n of the solution as described in step 5 of Algorithm 3.1. Notice that for a fixed radius r_l, coefficients $v_n^-(r_l)$ and $v_{-n}^+(r_l)$, $n \in [-N/2, 0]$ are stored in the same processor. Therefore, computations in (17) can be performed locally and concurrently, without any communication. Specifically, processor p_j evaluates terms $v_n(r_l)$, $n \in [-N/2, N/2]$, where $l \in [\gamma(j)+1, \gamma(j+1)]$. A final set of communications is employed to broadcast

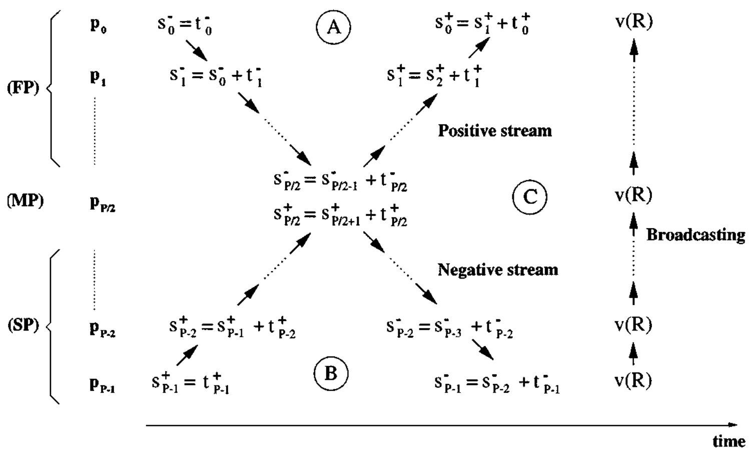


FIG. 4. Message distribution in the algorithm. Two streams of neighbor-to-neighbor messages cross communication channels simultaneously. Homogeneous and principal solution are combined after processor p_{P-1} broadcasts the boundary values of v.

the values $v_n(R)$, $n \in [-N/2, N/2]$, from p_{P-1} to all other processors so that the Fourier coefficients u_n of the solution can be evaluated by using Eq. (7), as represented in step 6 of Algorithm 3.1. This broadcast process is represented in Fig. 4 by the second set of upward arrows starting from processor p_{P-1}.

The notation in Eqs. (24) and (25) will be simplified to allow a clear exposition of the interprocessor communication present in our parallel implementation:

- Relation $s_j^- = s_{j-1}^- + t_j^-$ represents the updating process in recurrence (24), and
- Relation $s_j^+ = s_{j+1}^+ + t_j^+$ represents updating (25).

The parallel algorithm adopts the successful approach investigated in [2, 3]. Processors are divided into three groups: processor $p_{P/2}$ is defined as the *middle processor* (MP), processors $p_0, \ldots, p_{P/2-1}$ are the *first half* processors (FP), and $p_{P/2+1}, \ldots, p_{P-1}$ are in the *second half* (SP) as represented in Fig. 4.

We define a *negative stream* (negative pipe): A message started from processor p_0 containing the values $s_0^- = t_0^-$ and passed to the neighbor p_1. Generically, processor p_j receives the message s_{j-1}^- from p_{j-1}, updates the accumulated sum $s_j^- = s_{j-1}^- + t_j^-$, and sends the new message s_j^- to processor p_{j+1}. It corresponds to the downward arrows in Fig. 4. In the same way, processors on the second half start computations for partial sums s^+. A *positive stream* starts from processor p_{P-1}: processor p_j receives s_{j+1}^+ from p_{j+1} and sends the updated message $s_j^+ = s_{j+1}^+ + t_j^+$ to p_{j-1}. The positive stream is formed by the first set of upward arrows in Fig. 4. The resulting algorithm is composed of two simultaneous streams of neighbor-to-neighbor communication, each one with messages of length $N/2$. Note from Fig. 4 that negative and positive streams arrive at the middle processor simultaneously because of the symmetry of the communication structure. In [2, 3] we describe an efficient interprocessor coordination scheme which leads local computational work being performed simultaneously with the message-passing mechanism. In short, it consists of having messages arriving and leaving the middle processor as early as possible so that idle times are minimized. Any processor p_j in the first half (FP) obtains the accumulated sum s_j^- and immediately sends it to the next neighbor processor p_{j+1}.

Computations for partial sums t_j^+ only start after the negative stream has been sent. It corresponds to evaluating t_j^+ within region A of Fig. 4. Similarly, any processor p_j in the second half (SP) performs all the computations and message-passing work for the positive stream prior to the computation of partial sums t_j^- in region B. This mechanism minimizes delays because of interprocessor communication. In fact, in [2] we compare this approach against other parallelization strategies by presenting complexity models for distinct parallel implementations. The analysis shows the high degree of scalability of the algorithm.

The parallel algorithm presented here is certainly based on decomposing the domain into full annular regions and hence, it has some analogy with domain decomposition method. But this analogy is superficial because domain decomposition methods by its very name have come to refer to methods which attempt to solve the same equations in every subdomain, whereas our algorithm *does not* attempt to solve the same equation in each annular subdomain separately. Thus our algorithm is not a classical domain decomposition method. Interpreting otherwise would be misleading. In fact, decomposing a circular domain into full annular domains and then attempting to solve the equation in each subdomain in the spirit of domain decomposition method would not be very appealing for a very large number of domains because the surface-to-volume area becomes very large. Our algorithm is not based on this principle in its entirety, even though there is some unavoidable similarity.

4.2. *Complexity of the Parallel Algorithm*

To analyze the overhead resulting from interprocessor coordination in the parallel algorithm we adopt a standard communication model for distributed memory computers. For the timing analysis we consider t_s as the message startup time and t_w the transfer time for a complex number. To normalize the model, we adopt constants c_1 as the computational cost for floating point operations in the FFT algorithm, and c_2 to represent operation counts for the other stages of the algorithm. To obtain the model, we analyze the timing for each stage of the algorithm:

- Each processor performs a set of M/P Fourier transforms in $(c_1/2)(M/P)N\log_2 N$ operations.
- Radial integrals $C_n^{i,i+1}$ and $D_n^{i-1,i}$ are obtained using $(c_2/4)(M/P)N$ operations for the trapezoidal rule (and $(c_2 2/3)(M/P)N$ for Simpson's rule).
- Each group of M/P partial sums t^+ and t^- takes $(c_2/4)(M/P)(N/2)$ operations on each processor.
- Positive and negative streams start from processors p_{P-1} and p_0, respectively, and each processor forwards (receive and send) a message of length $N/2$ toward the middle node (see Fig. 4). The total time is $2((P-1)/2)(t_s + (N/2)t_w)$.
- The second group of M/P partial sums t^+ and t^- is performed in $(c_2/4)(M/P)(N/2)$ operations.
- Positive and negative streams restart from the middle node and arrive in p_0 and p_{P-1}, respectively, after $2((P-1)/2)(t_s + (N/2)t_w)$ time units for communication.
- Terms v^-, v^+, and v are computed in $(c_2/4)(M/P)N$ operations.
- Boundary conditions are broadcast in $(t_s + Nt_w)\log_2 P$ time units.
- Principal solution v and boundary conditions are combined in $(c_2/4)(M/P)N$ operations.
- $(c_1/2)(M/P)N\log_2 N$ operations are used to apply inverse Fourier transforms.

Therefore, the parallel timing T_P for the parallel fast algorithm is given by

$$T_P = \frac{MN}{P}(c_1 \log_2 N + c_2) + (2(P-1) + \log_2 P)t_s + N(P - 1 + \log_2 P)\, t_w. \tag{28}$$

To obtain an asymptotic estimate for the parallel timing, we drop the computational terms of lower order in (28) which leads to

$$T_P^{asymp} = c_1 \frac{MN}{P} \log_2 N + 2Pt_s + NPt_w. \tag{29}$$

The performance of the parallel algorithm can be observed by comparing Eq. (29) against the timing estimate for the sequential algorithm. In the case of a sequential implementation, we have the following stages:

- M Fourier transforms are performed in $(c_1/2)MN \log_2 N$ operations.
- Radial integrals $C_n^{i,i+1}$ and $D_n^{i-1,i}$ are obtained after $(c_2/4)MN$ operations.
- Terms v^-, v^+ and v are computed in $(c_2/4)MN$ operations.
- Principal solution v and boundary conditions are combined in $(c_2/4)MN$ operations.
- M inverse Fourier transforms take $(c_1/2)MN \log_2 N$ computations.

Summarizing, the sequential timing T_s is given by

$$T_s = c_1 MN \log_2 N + \frac{3}{4} c_2 MN, \tag{30}$$

with asymptotic model

$$T_s^{asymp} = c_1 MN \log_2 N. \tag{31}$$

From Eqs. (28) and (30) one can observe that most of the parallel overhead is attributed to the communication term in Eq. (28). An immediate consequence is that overheads are mainly the result of increasing the number of angular grid points N. No communication overhead is associated with the number of radial grid points M. We use the asymptotic estimates to obtain the speedup S for the parallel algorithm

$$S = \frac{T_s^{asymp}}{T_P^{asymp}} = \frac{c_1 MN \log_2 N}{c_1 \frac{MN}{P} \log_2 N + 2Pt_s + NPt_w} \tag{32}$$

$$= P \frac{c_1 MN \log_2 N}{c_1 MN \log_2 N + P^2(2t_s + Nt_w)} \tag{33}$$

and the corresponding efficiency

$$E = \frac{S}{P} = \frac{1}{1 + P^2(2t_s + Nt_w)/c_1 MN \log_2 N}, \tag{34}$$

which shows that the efficiency decays quadratically in the number of processors P.

Different problem sizes correspond to distinct levels of granularity, which implies that there is an optimal number of processors associated with each granularity. Since message lengths depend on N and computational work depends also on M, the theoretical model can be used to estimate the best performance for a given problem. The number of processors for which the asymptotic parallel running time T_P^{asympt} achieves its minimum is determined by $\frac{\partial T_P^{asympt}}{\partial P} = 0$. In the case of (29), we have

$$P_{opt}^{asymp} = \sqrt{\frac{c_1 MN \log_2 N}{2t_s + Nt_w}}, \tag{35}$$

which can be understood as an approximation for the optimal value of P which maximizes the efficiency (34) for given values of M and N.

4.3. *Comparison with a Matrix Transposition-based Algorithm*

Although the recursive relations in Corollary 2.2 are very appropriate to a sequential algorithm, these may introduce excessive communications on parallel implementation. The major difference is that if one attempts to evaluate recurrences (13) and (14), data must be reverted in all processors. In fact, steps **3** and **4** in Algorithm 3.1 show that each coefficient $v_n^-(r_l)$ depends on all terms $C_n^{i-1,i}$ with $i \in [2, l]$, and each coefficient $v_n^+(r_l)$ depends on all terms $D_n^{i,i+1}$ with $i \in [l, M-1]$. Consequently a message-passing mechanism must be used to exchange coefficients of the form $C_n^{i-1,i}$ and $D_n^{i,i+1}$ across processors. Figure 5 shows data being reverted in all processors for the case where $P = 4$. Initially each processor contains data for evaluating M/P Fourier transforms. It corresponds to each row on Figure 5(a). To calculate recurrences locally, each processor must exchange distinct data of size NM/P^2 with all $P-1$ remaining processors. At the end of the communication cycle, processor p_j contains all the terms $C_n^{i-1,i}$ and $D_n^{i,i+1}$ with $n \in [jN/P - N/2, (j+1)N/P - N/2]$. Figure 5(b) describes the communication pattern. Rows are divided into P blocks of size NM/P^2 so that processor p_j exchanges distinct data-blocks with different processors. The data-transfer pattern involves an all-to-all personalized communication as in a parallel matrix transposition procedure. For a mesh architecture the

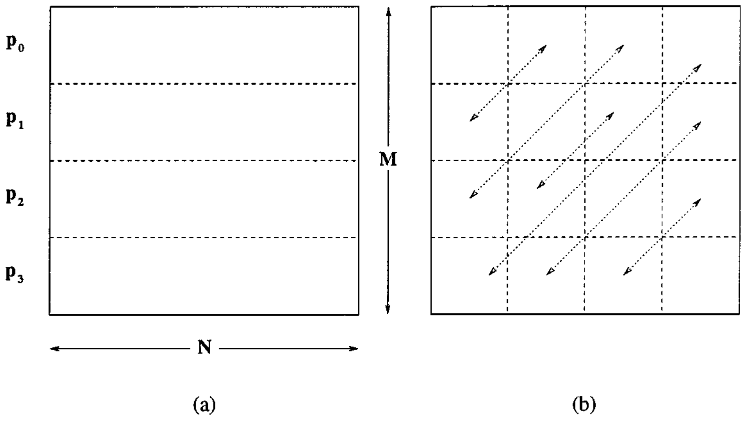


FIG. 5. Coordination pattern based on all-to-all personalized communication: (a) M/P Fourier transforms are evaluated locally; (b) each two processors exchange blocks of size MN/P^2.

estimated communication timing [15] is given by

$$T_{comm}^{transpose} = 2(\sqrt{P} - 1)\left(2t_s + \frac{MN}{P}t_w\right). \tag{36}$$

Therefore, interprocessor communication introduces a delay of order $4MN/\sqrt{P}$. Comparatively, the stream-based algorithm generates a delay of order PN. In a large-scale application, clearly $M \gg P$ because of practical limitations on the number of available processors which make $PN \ll 4MN/\sqrt{P}$. It implies that the stream-based algorithm must scale up better than the second approach because of a smaller communication overhead.

4.4. *Comparison with Other Methods*

Fourier analysis cyclic reduction (FACR) solvers encompass a class of methods for the solution of Poisson's equation on regular grids [12, 24, 25]. In two-dimensional problems, one-dimensional FFTs are applied to decouple the equations into independent triangular systems. Cyclic reduction and Gaussian elimination (or another set of one-dimensional FFTs and inverse FFTs) are used to solve the linear systems. In the FACR(ℓ) algorithm, ℓ preliminary steps of block-cyclic reduction are performed to decrease the number or the length of the Fourier coefficients. The reduced system is solved by the FFT method and by ℓ steps of block back-substitution. In particular; for $\ell = 0$ we have the basic FFT method, and $\ell = 1$ corresponds to a variant of the original FACR algorithm [12]. The basic idea of the FACR(ℓ) method relies on switching to Fourier analysis in the middle of cyclic reduction to reduce the operation count when compared with either pure Fourier analysis or cyclic reduction. Formally, the optimal choice $\ell \sim \log_2(\log_2 N)$ makes the asymptotic operation count for FACR(ℓ) be $\mathcal{O}(N^2 \log_2 \log_2 N)$ in an $N \times N$ grid, which is an improvement over the estimate $\mathcal{O}(N^2 \log_2 N)$ associated with the basic FFT method (FACR(0)) and cyclic reduction.

A parallel implementation of the FACR(ℓ) solver must take into account the effect of the choice of ℓ on the degree of parallelism of the algorithm [25]. At $\ell = 0$, the method performs a set of independent sine transforms and solves a set of independent tridiagonal systems, which makes the choice $\ell = 0$ ideally suited for parallel computations. The parallel implementation of the matrix decomposition Poisson solver (MD-Poisson solver) presented in [21] follows this concept: A block-pentadiagonal system is solved on a ring of P processors using Gaussian elimination without pivoting, so that only neighbor-to-neighbor communication is required. The complexity of the method on a ring of P processors is $\mathcal{O}(N^2/P \log_2 N)$ if one disregards communication overhead [21]. For $\ell > 0$, the degree of parallelism of the FACR(ℓ) algorithm decreases at each additional stage of cyclic reduction. For example, in [14] a parallel variant of the FACR(ℓ) algorithm exploits the numerical properties of the tridiagonal systems generated in the method. Factorization is applied based on the convergence properties of these systems. However, this approach can lead to severe load-imbalance on a distributed memory architecture because convergence rates may be different for each system, resulting in idle processors. Cyclic allocation must be used to diminish load-imbalance. Moreover, it is also known from [14] that any two-dimensional data partitioning would produce communication overhead because of the data transposition.

The previous observations show that our parallel Poisson solver is competitive with other current techniques. Typically, the best parallel solvers are defined using an one-dimensional processor array configuration because of the unbalanced communication requirements for the operations performed along the different coordinates of the grid.

5. NUMERICAL RESULTS

In this section, numerical results for the algorithms presented in the previous sections are given. To achieve portability, we used MPI [19] for the communication library. Currently, major computer vendors provide MPI implementations regardless of the memory model adopted on each platform. It allows easy implementation and portability.

Of particular importance to the following results is the accuracy of the methods for a given number of Fourier coefficients N and a number of circles M used for the domain discretization. For sufficiently smooth data only a few number of Fourier coefficients are needed to guarantee an accurate representation of the solution in a finite Fourier space. However, if the actual function presents rapid variations, then a high-frequency component may appear to be the same as a lower frequency component when using a limited number of samples. In other words, aliasing may occur. Similarly, the numerical integration method adopted to evaluate one-dimensional radial integrals presents an error term depending on the number of circles defined during the discretization of the domain. For instance, the trapezoidal rule presents an error of order $\mathcal{O}(\delta r^2)$, where $\delta r = R/M$ for a disk of radius R. If a three-point-based integration method is adopted, such as the variant of the Simpson's rule presented in Section 3, one would expect convergence of order $\mathcal{O}(\delta r^3)$. It suggests that there is a tradeoff when making a choice for the discretization parameters M and N. Numerical results in Section 5.1 demonstrate the accuracy of our solver.

Timing performance is also a critical issue in scientific computing. To increase memory bandwidth and decrease latency of memory access, more recent computer architectures are based on memory hierarchy structures. Under the principle of locality of reference, the data most recently accessed is likely to be referenced again in the near future. Modern computers present a cache memory at the top of the hierarchy: A smaller and faster memory is connected to the processor to hold the most recently accessed data. The function of the cache is to minimize the number of accesses to other slower levels on the memory hierarchy. Understanding and exploiting the memory hierarchy is a fundamental issue when obtaining high performance for numerical applications. A good utilization of data cache depends not only on the data partitioning but also on how the computational work is performed. The fast algorithm was designed to take advantage of data cache. In Section 5.2, we present sequential and parallel timings for the fast algorithm.

5.1. *Accuracy of the Poisson Solver on Disks*

Seven problems were tested to determine the accuracy and efficiency of the Poisson solver for Dirichlet and Neumann problems defined on the unit disk $B = \overline{B(0; 1)}$. Problems 3 and 4 were also solved for disks $\overline{B(0; R)}$ with $R \neq 1$. Numerical experiments were carried out using double precision representation. The first four problems present solutions smooth enough to make the number of circles M as the dominant parameter for the accuracy of the method. The last three problems were taken to exemplify the importance of the number of Fourier coefficients N in use. For each problem, we present only the solution $u(x, y)$ in B so that the right hand side term f and the boundary conditions can easily be obtained from u. The only exception occurs in Problem 7.

TABLE II
Problem 1—Relative Errors in Norms $\|\cdot\|_\infty$ and $\|\cdot\|_2$ Using a Fixed Number of Fourier Coefficients $N = 64$

	Relative errors for problem 1							
	Trapezoidal rule				Simpson's rule			
	Dirichlet		Neumann		Dirichlet		Neumann	
M	$\|\cdot\|_\infty$	$\|\cdot\|_2$	$\|\cdot\|_\infty$	$\|\cdot\|_2$	$\|\cdot\|_\infty$	$\|\cdot\|_2$	$\|\cdot\|_\infty$	$\|\cdot\|_2$
64	2.6e-5	4.6e-5	7.0e-4	5.7e-4	4.4e-6	6.3e-6	4.4e-6	6.3e-6
128	6.4e-6	1.1e-5	1.7e-4	1.4e-4	5.5e-7	7.9e-7	5.5e-7	7.9e-7
256	1.6e-6	2.8e-6	4.3e-5	3.5e-5	6.9e-8	9.9e-8	6.9e-8	9.9e-8
512	3.9e-7	6.9e-7	1.1e-5	8.7e-6	8.6e-9	1.2e-8	8.6e-9	1.2e-8
1024	9.8e-8	1.7e-7	2.7e-6	2.2e-6	1.1e-9	1.5e-9	1.1e-9	1.5e-9
2048	2.5e-8	4.3e-8	6.7e-7	5.4e-7	1.3e-10	1.9e-10	1.3e-10	1.9e-10

PROBLEM 1. The solution of the first problem [13] is given by

$$u(x, y) = 3e^{x+y}(x - x^2)(y - y^2) + 5.$$

Table I presents relative errors in the norm $\|\cdot\|_\infty$ when solving the Dirichlet problem for distinct values of N and M. Specifically, each row corresponds to a fixed value of N taken as 64, 128, 256, 512, 1024, or 2048. Similarly, each column corresponds to a fixed value of M ranging from 64 to 2048. Entries marked with a dash represent no available data because of memory limitations. The trapezoidal rule was used for numerical integration in the radial direction.

Clearly, the dominant parameter is the number of circles M. Functions f and u are smooth on each circle of the discretization, and consequently 64 Fourier coefficients are enough to represent these functions. The only variations in Table I occurs when we increase the number of circles, which increases the accuracy of the numerical integration in the radial directions. The same behavior is observed for the relative errors in the norm $\|\cdot\|_2$ and for the associated Neumann problem. Table II summarizes relative errors in norm $\|\cdot\|_\infty$ and in norm $\|\cdot\|_2$ when the Dirichlet and Neumann problems are solved using a constant number of Fourier coefficients $N = 64$. Since the Fourier space representation presents high accuracy for u and f, convergence rates are determined by the numerical integration adopted in the radial direction. In fact, one can observe in Table II that the ratio between two consecutive errors in the same columns for the trapezoidal rule is constant and equals 4, that is, the two-points-based integration results in quadratic convergence. For the case of three-points-based integration derived from Simpson's rule, the ratio is constant and equals 8, which implies cubic convergence.

PROBLEM 2. The solution of this problem has a discontinuity in the "2.5" derivative [13]:

$$u(x, y) = (x + 1)^{5/2}(y + 1)^{5/2} - (x + 1)(y + 1)^{5/2} \\ - (x + 1)^{5/2}(y + 1) + (x + 1)(y + 1).$$

TABLE III
Problem 2—Relative Errors in Norms $\|\cdot\|_\infty$ and $\|\cdot\|_2$ Using a Fixed Number of Fourier Coefficients $N = 64$

	Relative errors for problem 2							
	Trapezoidal rule				Simpson's rule			
	Dirichlet		Neumann		Dirichlet		Neumann	
M	$\|\cdot\|_\infty$	$\|\cdot\|_2$	$\|\cdot\|_\infty$	$\|\cdot\|_2$	$\|\cdot\|_\infty$	$\|\cdot\|_2$	$\|\cdot\|_\infty$	$\|\cdot\|_2$
64	3.4e-5	1.7e-4	2.2e-4	5.4e-4	3.2e-6	1.3e-5	3.5e-6	1.2e-5
128	8.2e-6	4.2e-5	5.4e-5	1.3e-4	4.2e-7	1.5e-6	4.5e-7	1.5e-6
256	2.0e-6	1.0e-5	1.3e-5	3.3e-5	5.6e-8	1.9e-7	5.9e-8	1.9e-7
512	4.9e-7	2.6e-6	3.3e-6	8.1e-6	7.7e-9	2.3e-8	8.3e-9	2.3e-8
1024	1.2e-7	6.4e-7	8.2e-7	2.0e-6	1.4e-9	2.9e-9	1.7e-9	2.9e-9
2048	3.1e-8	1.6e-7	2.0e-7	5.1e-7	5.5e-10	4.2e-10	9.3e-10	4.6e-10

As in the previous problem, the dominant parameter is the number of circles M. Table III presents relative errors for the Dirichlet and Neumann problems in a discretization with a constant number of Fourier coefficients $N = 64$. Note that quadratic and cubic convergence, resulting from distinct integration schemes, still holds.

PROBLEM 3. This problem was originally designed for the ellipse centered at (0, 0) with major and minor axes of 2 and 1 [20]. One interesting property is the presence of symmetry for all four quadrants:

$$u(x, y) = \frac{e^x + e^y}{1 + xy}.$$

Relative errors for the Dirichlet and Neumann problems can be found in Table IV. The number of Fourier coefficients was kept constant $N = 64$. Again, the ratio between two consecutive errors in norm $\|\cdot\|_2$ is constant and equals either 4 or 8. The same problem was also solved for the disk $\overline{B(0; 0.5)}$, and the relative errors for $N = 64$ are presented in

TABLE IV
Problem 3—Relative Errors Using $R = 1$ and a Fixed Number of Fourier Coefficients $N = 64$

	Relative errors for problem 3 ($R = 1$)							
	Trapezoidal rule				Simpson's rule			
	Dirichlet		Neumann		Dirichlet		Neumann	
M	$\|\cdot\|_\infty$	$\|\cdot\|_2$	$\|\cdot\|_\infty$	$\|\cdot\|_2$	$\|\cdot\|_\infty$	$\|\cdot\|_2$	$\|\cdot\|_\infty$	$\|\cdot\|_2$
64	1.2e-4	1.3e-4	6.0e-4	3.0e-4	2.4e-5	1.9e-5	2.5e-5	2.0e-5
128	2.9e-5	3.2e-5	1.5e-4	7.6e-5	3.2e-6	2.5e-6	3.2e-6	2.5e-6
256	7.6e-6	8.0e-6	3.8e-5	1.9e-5	6.2e-7	3.1e-7	6.2e-7	3.2e-7
512	1.9e-6	2.0e-6	9.5e-6	4.7e-6	1.3e-7	4.0e-8	1.3e-7	4.0e-8
1024	5.1e-7	5.0e-7	2.3e-7	1.2e-6	3.0e-8	5.2e-9	3.0e-8	5.2e-9
2048	1.3e-7	1.2e-7	5.9e-7	2.9e-7	7.2e-9	7.2e-10	7.2e-9	7.2e-10

TABLE V
Problem 3—Relative Errors Using R = 0.5 and a Fixed Number of Fourier Coefficients N = 64

	Relative errors for problem 3 ($R = 0.5$)							
	Trapezoidal rule				Simpson's rule			
	Dirichlet		Neumann		Dirichlet		Neumann	
M	$\|\cdot\|_\infty$	$\|\cdot\|_2$	$\|\cdot\|_\infty$	$\|\cdot\|_2$	$\|\cdot\|_\infty$	$\|\cdot\|_2$	$\|\cdot\|_\infty$	$\|\cdot\|_2$
64	3.1e-5	1.0e-5	3.1e-5	1.0e-5	3.7e-6	6.0e-7	3.7e-6	6.0e-7
128	8.2e-6	2.4e-6	8.2e-6	2.5e-6	9.0e-7	1.0e-7	9.0e-7	1.0e-7
256	2.2e-6	5.9e-7	2.2e-6	6.1e-7	2.2e-7	1.8e-8	2.2e-7	1.8e-8
512	5.9e-7	1.5e-7	5.9e-7	1.5e-7	5.5e-8	3.7e-9	5.5e-8	3.2e-9
1024	1.6e-7	3.6e-8	1.6e-7	3.7e-8	1.4e-8	5.6e-10	1.4e-8	5.6e-10
2048	4.2e-8	9.0e-9	4.2e-8	9.3e-9	3.4e-9	9.9e-11	3.5e-9	9.8e-11

Table V. As it was expected, the accuracy is higher for $R = 0.5$ because of larger density of points in the domain discretization.

PROBLEM 4. In contrast with Problem 3, here we adopt a solution without symmetries:

$$u(x, y) = x^3 e^x (y + 1)\cos(x + y^3).$$

Table VI presents relative errors for the Dirichlet and Neumann problems in the disk $\overline{B(0;1)}$. The same problem was solved in the larger disk $\overline{B(0;2)}$, and the numerical results are shown in Table VII. Clearly, the solution in the larger domain (even using twice the number of Fourier coefficients) presents a lower accuracy when compared with the same number of circles for $\overline{B(0;1)}$.

TABLE VI
Problem 4—Relative Errors Using R = 1 and a Fixed Number of Fourier Coefficients N = 64

	Relative errors for problem 4 ($R = 1$)							
	Trapezoidal rule				Simpson's rule			
	Dirichlet		Neumann		Dirichlet		Neumann	
M	$\|\cdot\|_\infty$	$\|\cdot\|_2$	$\|\cdot\|_\infty$	$\|\cdot\|_2$	$\|\cdot\|_\infty$	$\|\cdot\|_2$	$\|\cdot\|_\infty$	$\|\cdot\|_2$
64	1.3e-4	2.5e-4	8.6e-4	1.3e-3	2.2e-5	5.5e-5	2.3e-5	5.5e-5
128	3.2e-5	6.1e-5	2.1e-4	3.3e-4	2.8e-6	6.7e-6	2.7e-6	6.8e-6
256	7.8e-6	1.5e-5	5.3e-5	8.2e-5	3.4e-7	8.4e-7	3.4e-7	8.4e-7
512	1.9e-6	3.7e-6	1.3e-5	2.0e-5	4.2e-8	1.0e-7	4.3e-8	1.0e-7
1024	4.8e-7	9.2e-7	3.2e-6	5.1e-6	5.3e-9	1.3e-8	5.3e-9	1.3e-8
2048	1.2e-7	2.3e-7	8.2e-7	1.2e-6	6.6e-10	1.6e-9	6.6e-10	1.6e-9

TABLE VII
Problem 4—Relative Errors Using $R = 2$ and a Fixed Number of Fourier Coefficients $N = 128$

	Relative errors for problem 4 ($R = 2$)							
	Trapezoidal rule				Simpson's rule			
	Dirichlet		Neumann		Dirichlet		Neumann	
M	$\lVert\cdot\rVert_\infty$	$\lVert\cdot\rVert_2$	$\lVert\cdot\rVert_\infty$	$\lVert\cdot\rVert_2$	$\lVert\cdot\rVert_\infty$	$\lVert\cdot\rVert_2$	$\lVert\cdot\rVert_\infty$	$\lVert\cdot\rVert_2$
64	6.1e-4	1.6e-3	3.6e-3	6.7e-3	2.9e-4	4.7e-4	3.1e-4	4.7e-4
128	1.4e-4	3.7e-4	9.0e-4	1.6e-3	3.6e-5	4.9e-5	3.7e-5	4.9e-5
256	3.4e-5	9.2e-5	2.3e-4	4.0e-4	4.4e-6	5.5e-6	4.4e-6	5.6e-6
512	8.3e-6	2.3e-5	5.8e-5	9.8e-5	5.4e-7	6.6e-7	5.5e-7	6.6e-7
1024	2.1e-6	5.7e-6	1.5e-5	2.4e-5	6.7e-8	8.0e-8	6.8e-8	8.1e-8
2048	7.3e-7	1.8e-6	4.3e-6	6.2e-6	8.4e-9	9.9e-9	8.4e-9	9.9e-9

PROBLEM 5. To analyze the effect of growing derivatives in our method we consider the solution

$$u(x, y) = \sin(\alpha\pi(x + y)).$$

This solution and the respective function $f(x, y) = -2\alpha^2\pi^2 \sin(\alpha\pi(x + y))$ present rapidly growing derivatives for large values of α [23]. In Tables VIII and IX we present relative errors in the norm $\|\cdot\|_\infty$ when solving the Dirichlet problem for $\alpha = 5$ and $\alpha = 20$, respectively. Here we have adopted the trapezoidal rule for evaluating the radial integrals. For the case $\alpha = 5$ the dominant parameter is the number of circles M regardless of the number of Fourier coefficients in use. In fact, quadratic convergence depending on M can be observed in Table VIII. For the larger value $\alpha = 20$ functions, u and f oscillate rapidly, and the derivatives increase in absolute value. The Fourier spaces of dimension $N = 64$ and $N = 128$ do not allow a good representation of u and f as one can observe on the first two rows of relative residual in Table IX. However, for $N = 256$ or larger, the Fourier space provides a good representation of these functions, and the quadratic convergence on M resumes (rows 3, 4, 5, and 6 in Table IX). This problem shows the importance of using Fourier representation when dealing with rapidly oscillating functions.

TABLE VIII
Problem 5—Relative Errors in Norm $\|\cdot\|_\infty$ Taking $\alpha = 5$

	Relative errors for Problem 5 (Dirichlet and $\alpha = 5$)					
$_N\backslash^M$	64	128	256	512	1024	2048
64	1.3e-2	3.4e-3	8.4e-4	2.1e-4	5.2e-5	1.4e-5
128	1.3e-2	3.4e-3	8.4e-4	2.1e-4	5.2e-5	1.3e-5
256	1.3e-2	3.4e-3	8.4e-4	2.1e-4	5.2e-5	—
512	1.3e-2	3.4e-3	8.4e-4	2.1e-4	—	—
1024	1.3e-2	3.4e-3	8.4e-4	—	—	—
2048	1.3e-2	3.4e-3 —	—	—	—	—

Note. The number of circles M is the dominant parameter.

TABLE IX
Problem 5—Relative Errors in Norm $\|\cdot\|_\infty$ Taking $\alpha = 20$

Relative errors for Problem 5 (Dirichlet and $\alpha = 20$)						
$N \backslash M$	64	128	256	512	1024	2048
64	2.5e+1	2.5e+1	2.5e+1	2.5e+1	2.5e+1	2.5e+1
128	2.2e+0	2.1e+0	2.1e+0	2.0e+0	2.0e+0	2.0e+0
256	2.7e-1	6.5e-2	1.6e-2	4.0e-3	1.0e-3	—
512	2.7e-1	6.5e-2	1.6e-2	4.0e-3	—	—
1024	2.7e-1	6.5e-2	1.6e-2	—	—	—
2048	2.7e-1	6.5e-2	—	—	—	—

Note. The number of Fourier coefficients is the dominant parameter for small values of N.

PROBLEM 6. To better understand the importance of the use of Fourier representation for functions with rapid variations, let

$$u(x, y) = 10\,\phi(x)\phi(y),$$

where $\phi(x) = e^{-100(x-1/2)^2}(x^2 - x)$. The solution has a sharp peak at $(0.5, 0.5)$, and it is very small for $(x - 0.5)^2 + (y - 0.5)^2 > 0.01$ [20]. Figure 6 shows the analytical solution u. For a small number of Fourier coefficients $N = 64$ aliasing occurs and errors of order 10^{-4} dominate the circle of radius $r = 0.5$ even if large values of M are used. In fact, Fig. 7 presents the function error for $N = 64$ and $M = 256$ when solving the

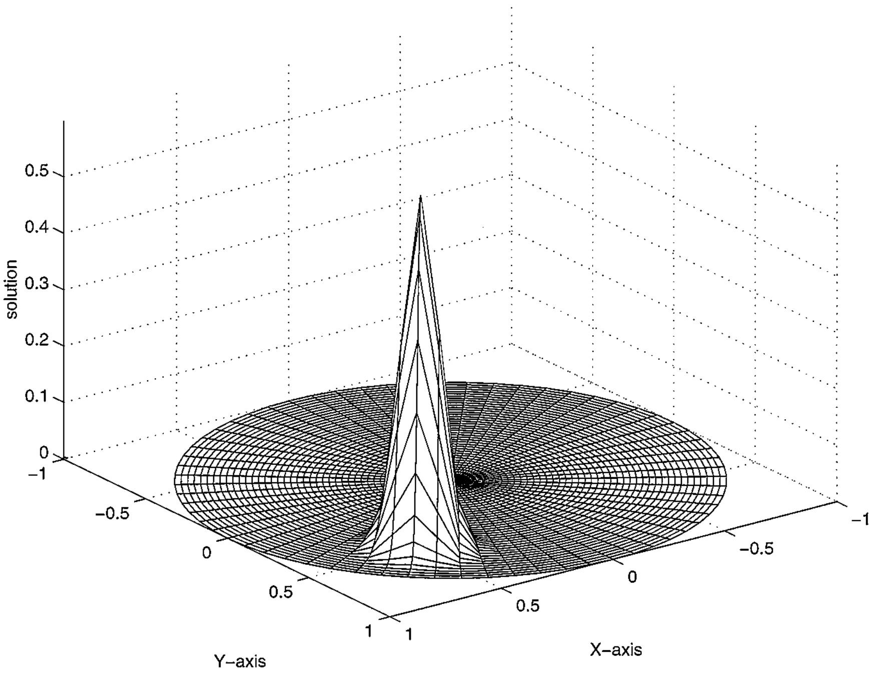


FIG. 6. Problem 6—Analytical solution.

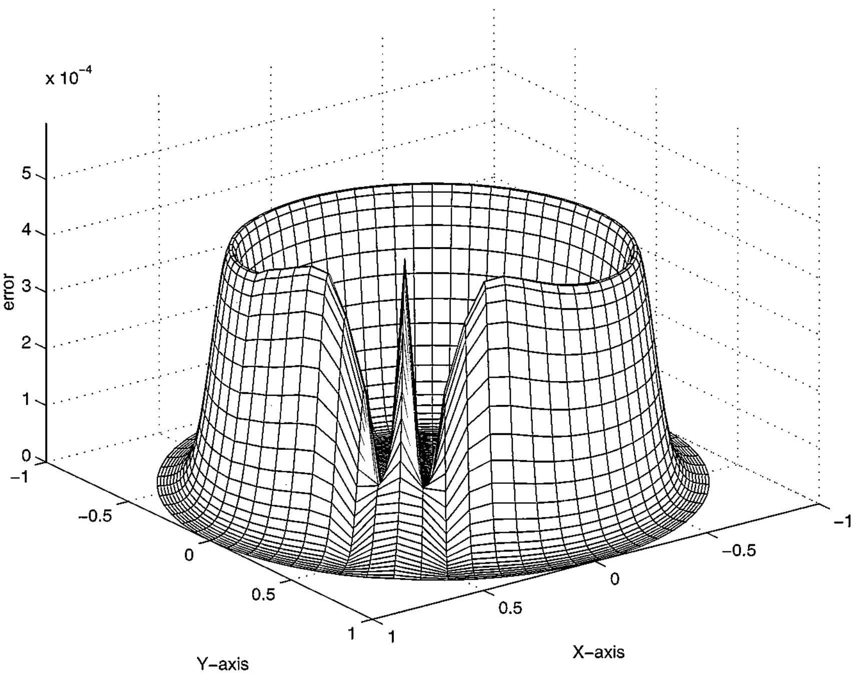


FIG. 7. Problem 6—Errors for 64 Fourier coefficients and 256 circles.

Dirichlet problem using the trapezoidal rule. If the number of coefficients is increased to $N = 128$, the Fourier space provides a better approximation, and the aliasing effect decreases drastically as one can observe in Fig. 8. Although the maximum error persists with order 10^{-4} in a neighborhood of (0.5, 0.5), globally it decreases for the larger value $N = 128$: Figure 9 contains the errors when only observing the grid points in $\overline{B(0; 1)}$ on the segment $(-\sqrt{2}/2, -\sqrt{2}/2)$ to $(\sqrt{2}/2, \sqrt{2}/2)$. Specifically, we say that the radial position is equal to -1 for the point $(-\sqrt{2}/2, -\sqrt{2}/2)$, and it is 1 for the point $(\sqrt{2}/2, \sqrt{2}/2)$. The linear plot of the errors presented in Figure 9(a) shows that for $N = 128$ the local error at (0.5, 0.5) persists in the same order but the aliasing effect is negligible at (–0.5, –0.5). Moreover, the log-scale shown in Fig. 9(b) shows the global convergence of the algorithm. Similar results hold for the Neumann problem as shown in Fig. 10.

PROBLEM 7. The last problem presents discontinuities on the boundary conditions. The formulation is best described in polar coordinates

$$\begin{aligned} \Delta u &= f, \quad \text{in } B = B(0; 1), \\ u &= g, \quad \text{on } \partial B, \end{aligned}$$

where

$$f(re^{i\alpha}) = -4r^3(\cos^2\alpha \cdot \sin\alpha + \sin^3\alpha)\sin(1 - r^2) - 8r\sin\alpha\cos(1 - r^2),$$

and

$$g(e^{i\alpha}) = \begin{cases} 0, & \alpha \in (0, \pi), \\ 1, & \alpha \in (\pi, 2\pi), \\ \frac{1}{2} & \alpha \in \{\pi, 2\pi\}, \end{cases}$$

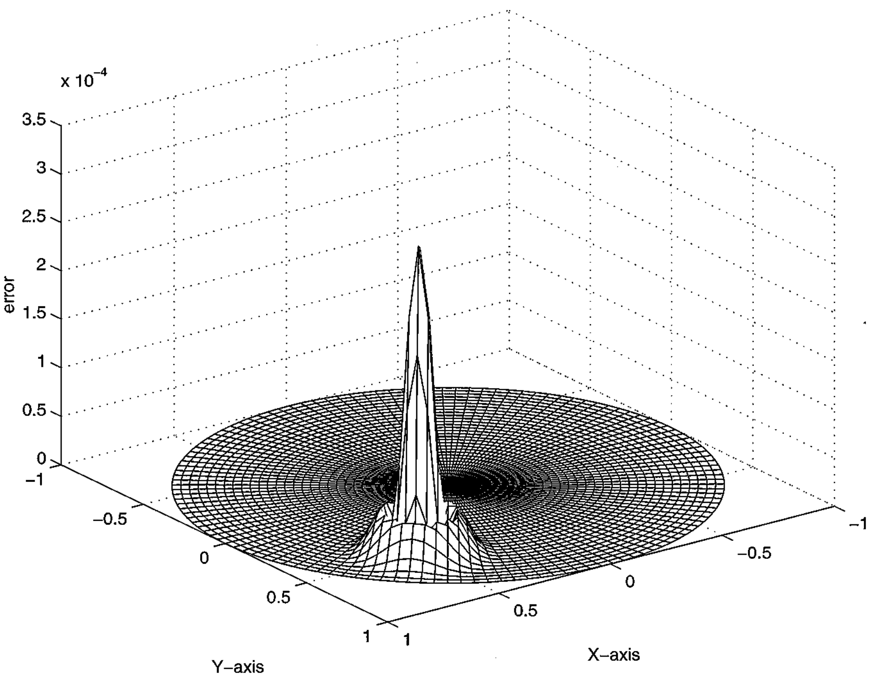


FIG. 8. Problem 6—Errors for 128 Fourier coefficients and 256 circles.

In this case we have the solution u given by

$$u(re^{i\alpha}) = \frac{1}{2} + \sin(r(1-r^2)\sin(\alpha)) - \frac{2}{\pi}\sum_{k=1}^{\infty} r^{2k-1}\frac{\sin(2k-1)\alpha}{2k-1}, \tag{37}$$

and the actual input data is expressed in Cartesian coordinates as

$$f(x, y) = -4(x^2y + y^3)\sin(1 - x^2 - y^2) - 8y\cos(1 - x^2 - y^2).$$

Figure 11 presents the actual solution of Problem 7 obtained by expanding the summation in (37) up to the machine precision on each point of the $M \times N$ discretization of the domain $\overline{B(0; 1)}$. The rapid variations in the points (1, 0) and $(-1, 0)$ produce considerable errors when the Dirichlet problem is solved using 64 Fourier coefficients and 256 circles, as shown in Fig. 12. Nevertheless, the use of a larger number of Fourier coefficients for representing the solution preserves the locality of the errors caused by rapid variations of the solution: Fig. 13 contains the errors when increasing the number of coefficients to 128; and Fig. 14 presents errors for 256 Fourier coefficients. Although the magnitude of the maximum error remains constant, the solution obtained by the algorithm converges globally. As an example, Fig. 15 contains the errors when only observing the grid points in $\overline{B(0; 1)}$ laying on the segment from $(0, -1)$ to (0, 1). In this case we say that the radial position is equal to -1 for the point $(0, -1)$, and it is 1 for the point (0, 1). The linear plot of the errors presented in Fig. 15(a) shows convergence as the number of Fourier coefficients increases from 64 to 128, and to 256. The log-scaling in Fig. 15(b) shows the rate of convergence. Global convergence can also be assessed by evaluating the global error without considering the points close to $(-1, 0)$ and (1, 0). Table X presents the relative errors in the domain

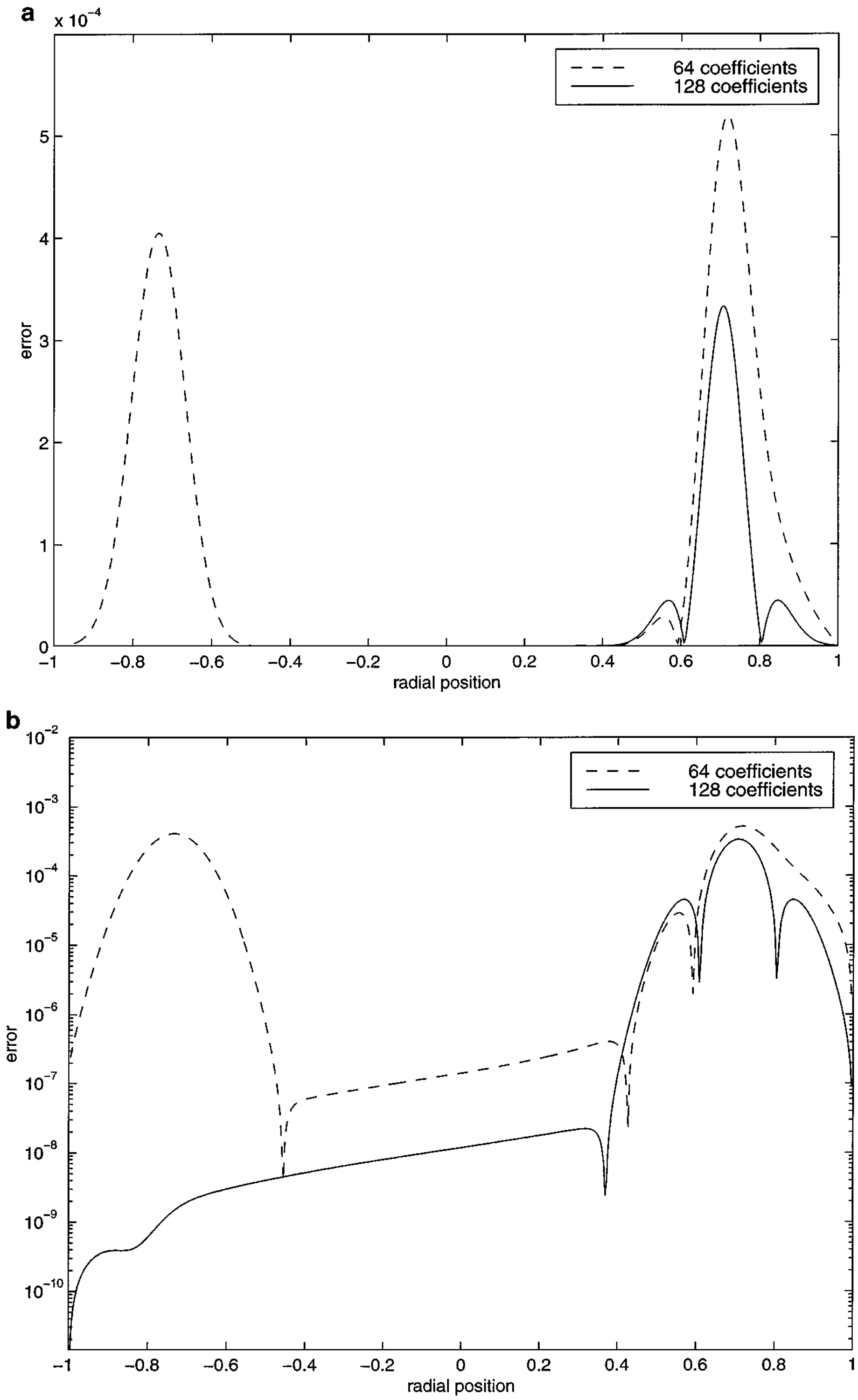


FIG. 9. Problem 6—Errors for the Dirichlet problem when considering the one-dimensional section of the disk $\overline{B(0;1)}$ from $(-\sqrt{2}/2, -\sqrt{2}/2)$ to $(\sqrt{2}/2, \sqrt{2}/2)$: (a) The aliasing effect disappears for $N = 128$; (b) global convergence also occurs as it can be noticed at the center of the graph.

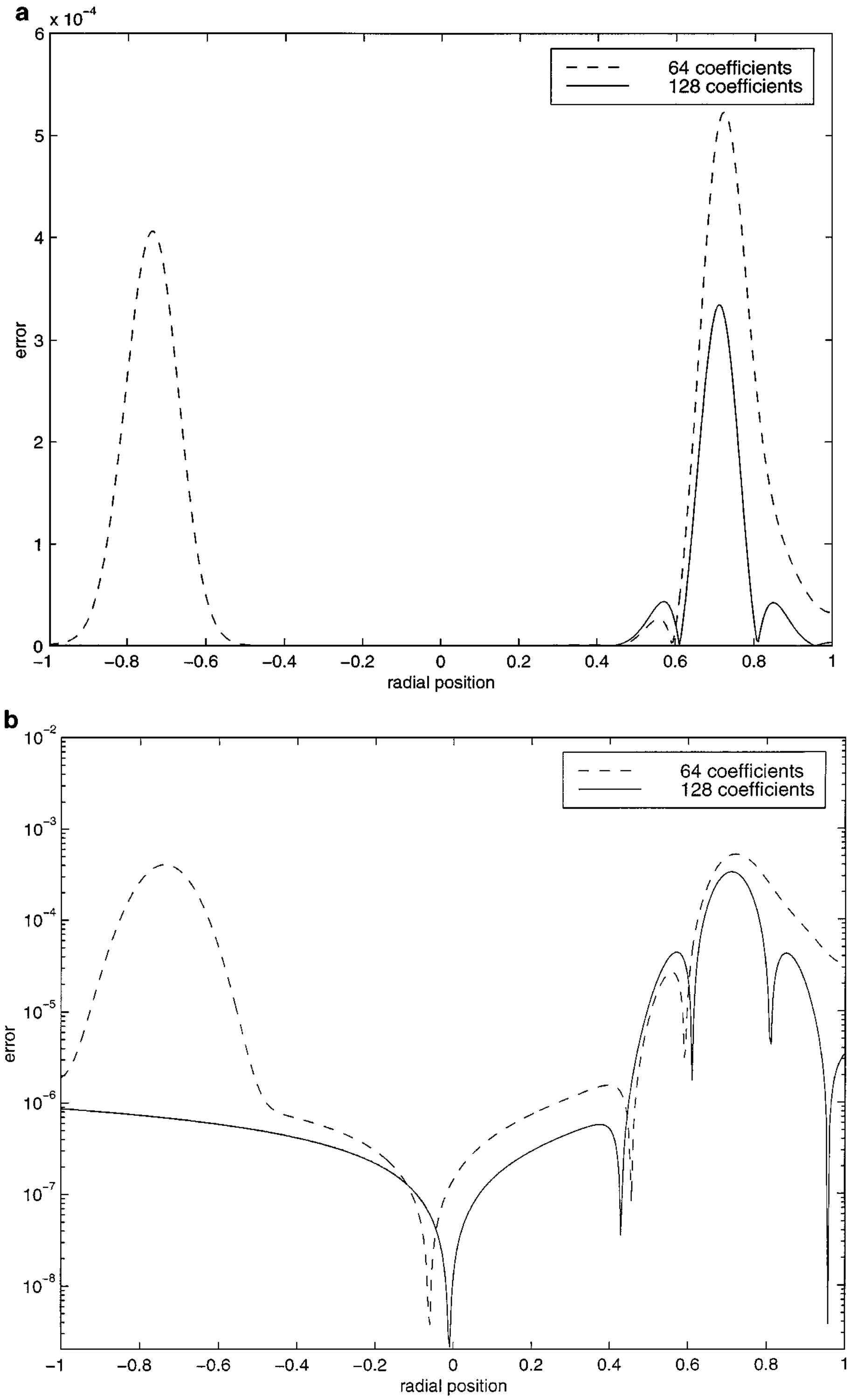


FIG. 10. Problem 6—Errors for the Neumann problem when considering the one-dimensional section of the disk $\overline{B(0;1)}$ from $(-\sqrt{2}/2, -\sqrt{2}/2)$ to $(\sqrt{2}/2, \sqrt{2}/2)$: (a) the aliasing effect disappears for $N = 128$; (b) global convergence also occurs as it can be noticed at the center of the graph.

TABLE X
Problem 7 - Relative Errors in Norm $\|\cdot\|_\infty$

Relative errors for Problem 7						
$N \backslash M$	64	128	256	512	1024	2048
64	3.1e-3	3.0e-3	3.1e-3	3.1e-3	3.1e-3	3.1e-3
128	5.6e-4	5.5e-4	5.6e-4	5.5e-4	5.6e-4	5.6e-4
256	1.4e-4	1.4e-4	1.4e-4	1.4e-4	1.4e-4	—
512	3.7e-5	3.5e-5	3.5e-5	3.5e-5	—	—
1024	1.7e-5	9.3e-6	8.6e-6	—	—	—
2048	1.6e-5	4.3e-6	—	—	—	—

Note. Errors were taken only over the points in $B(0;1) - (B_{0.01}(1;0) \cup B_{0.01}(-1;0))$.

$B(0;1) - (B_{0.01}(1,0) \cup B_{0.01}(-1,0))$. As the number of Fourier coefficients increases, convergence is observed.

5.2. *Timing Performance of the Fast Algorithm*

The computational results in this section were obtained on the HP V-Class [10] which is supported on the HP PA-8200 processor. The PA-8200 is based on the RISC Precision Architecture (PA-2.0) and runs at speeds of 200 or 240 MHz with 2 MBytes of data cache and 2 MBytes of instruction cache.

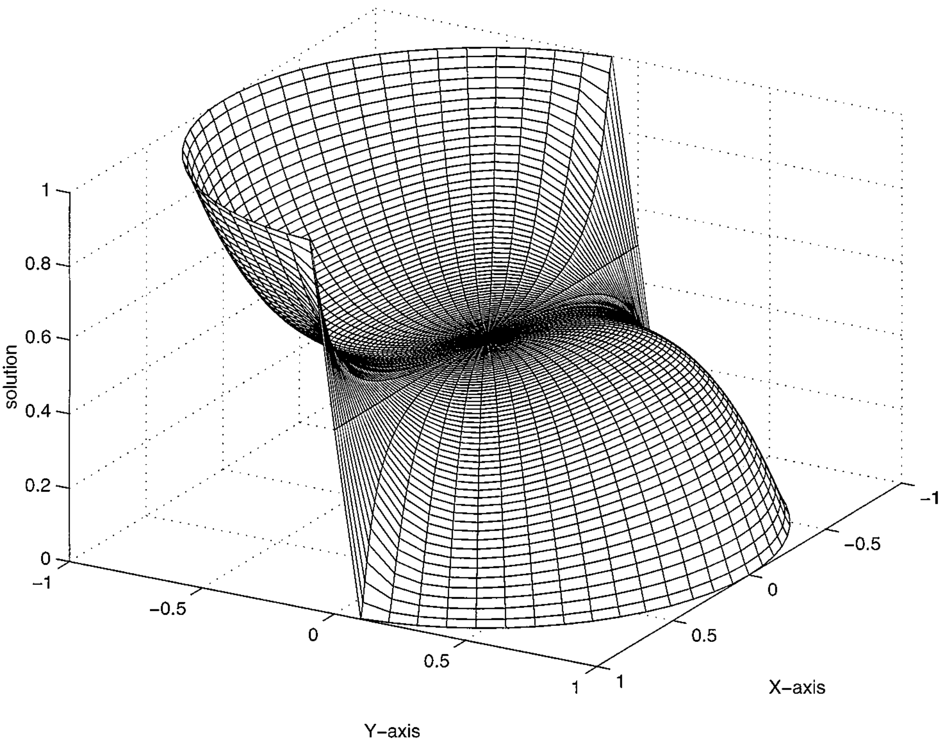


FIG. 11. Problem 7—Analytical solution.

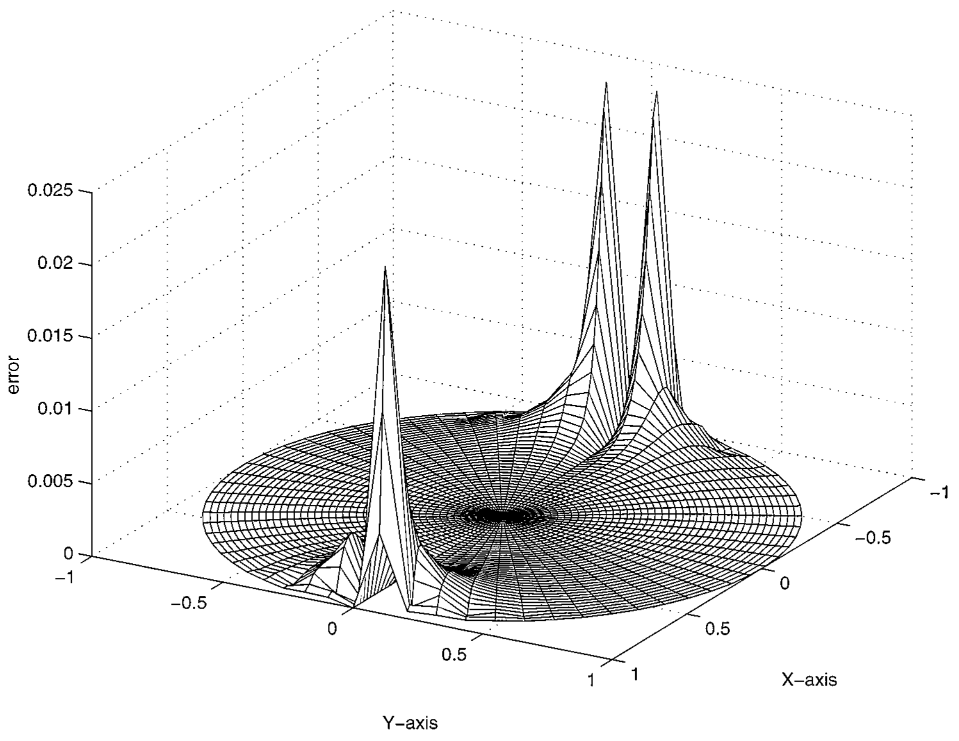


FIG. 12. Problem 7—Errors for 64 Fourier coefficients and 256 circles.

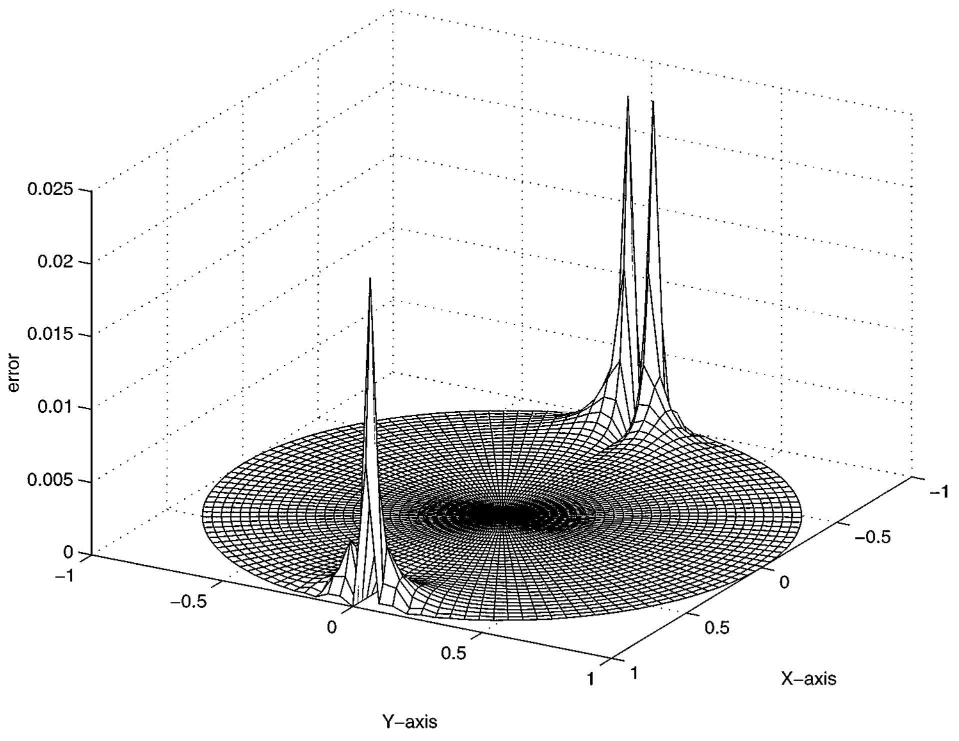


FIG. 13. Problem 7—Errors for 128 Fourier coefficients and 256 circles.

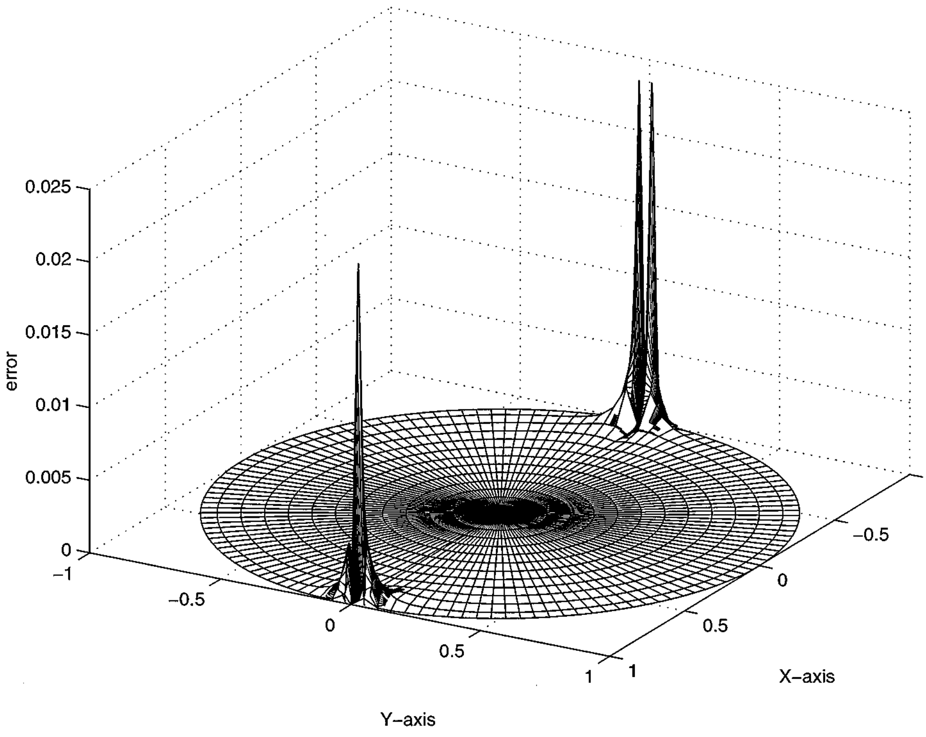


FIG. 14. Problem 7—Errors for 256 Fourier coefficients and 256 circles.

To observe the computational complexity of the fast algorithm, we ran the sequential code in a single node of the V-Class using seven distinct problem sizes. Table XI presents sequential timings when solving the Dirichlet and Neumann problems. Each row corresponds to $M = N$ taken as 32, 64, 128, 256, 512, 1024, or 2048. Results are shown for the two numerical integration schemes discussed in Section 3: the trapezoidal rule and the

TABLE XI
Timings and Estimates for the Constant c_1 for the Sequential Algorithm When Using Either Trapezoidal or Simpson's Rule

	Sequential timings and estimated constant c_1							
	Trapezoidal rule				Simpson's rule			
	Dirichlet		Neumann		Dirichlet		Neumann	
M = N	Time (sec.)	c_1	Time (sec.)	c_1	Time (sec.)	c_1	Time (sec.)	c_1
32	6.6e-4	1.2e-7	6.4e-4	1.2e-7	8.0e-4	1.5e-7	7.9e-4	1.5e-7
64	3.5e-3	1.4e-7	3.2e-3	1.3e-7	3.8e-3	1.5e-7	3.3e-3	1.3e-7
128	1.5e-2	1.3e-7	1.3e-2	1.1e-7	1.5e-2	1.3e-7	1.4e-2	1.2e-7
256	7.1e-2	1.3e-7	7.0e-2	1.3e-7	7.4e-2	1.4e-7	7.1e-2	1.3e-7
512	2.0e+0	8.7e-7	3.2e+0	1.3e-6	1.9e+0	8.3e-7	1.9e+0	8.1e-7
1024	1.5e+1	1.5e-6	1.5e+1	1.5e-6	1.5e+1	1.4e-6	1.5e+1	1.4e-6
2048	7.8e+1	1.6e-6	7.6e+1	1.6e-6	7.8e+1	1.7e-6	7.6e+1	1.6e-6

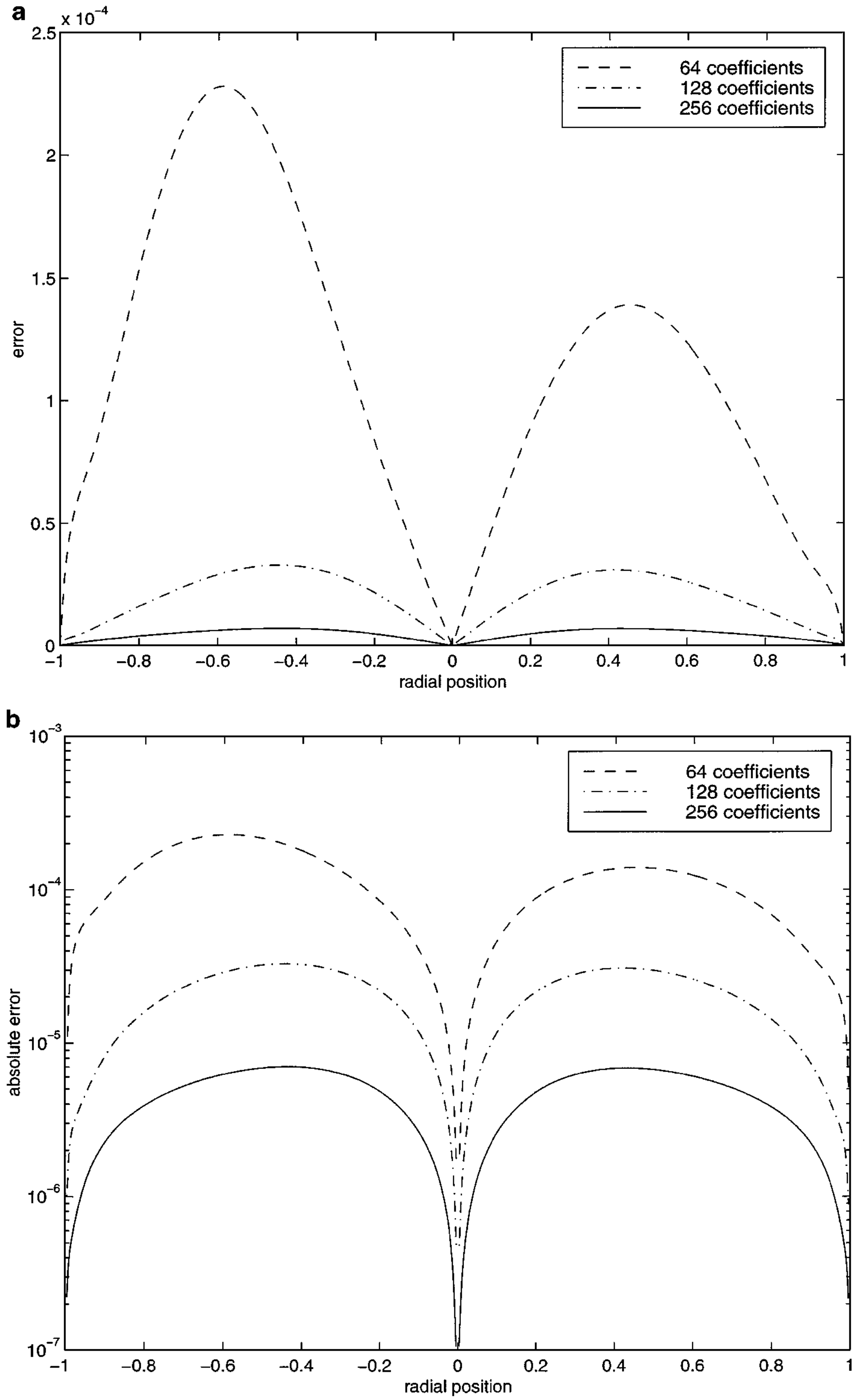


FIG. 15. Problem 7—Errors when considering the one-dimensional section of the disk $\overline{B(0; 1)}$ from $(0, -1)$ to $(0, 1)$: (a) convergence is observed as the number of Fourier coefficients increases; (b) the same errors observed in log-scaling.

modified Simpson's rule. Additionally, for each running time we estimate the constant c_1 in (31) which determines normalized timing per grid point spent on the sequential algorithm. Specifically,

$$c_1 = \frac{t}{N^2 \log_2 N},$$

where t represents the running times shown on the table. Overall, it shows an extremely low constant associated with the complexity of the algorithm. In fact, one can observe that c_1 is $\mathcal{O}(10^{-7})$ as observed for $N = 32$, 64, 128, and 256. It results from the data locality in Algorithm 3.1: It presents a low ratio of memory references over float point operations. For the larger cases $N = 512$, 1024 and 2048, one can observe slightly increasing values of c_1 because of the fact that all data cannot be stored in cache. It is due to the fact that some steps of Algorithm 3.1 basically involve two data structures formed by MN complex numbers in double precision. For the case in which $N = M = 256$ we have $256^2 \times 16 \times 2$ bytes, which can be maintained into the 2 MBytes of data cache. Conversely, for the cases $N = 512$, 1024, and 2048, multiple accesses between data cache and shared memory are expected.

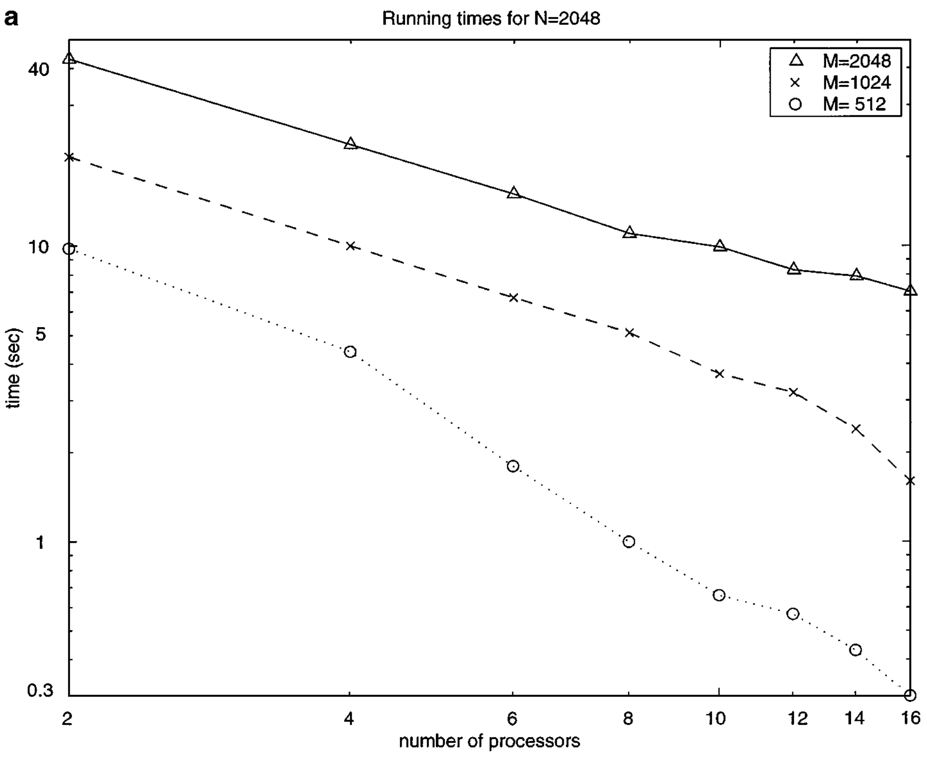
Estimate (31) can also be understood as the computational complexity of the algorithm based on floating point operations counting. In our current implementation, computations taken into account in (31) correspond to two sets of $4MN/2 \log_2 N + 3 \log_2 N + 4(2N - 1)$ multiplications and $6MN/2 \log_2 N + 4(2N - 1)$ additions. It leads to a total of $20MN/2 \log_2 N + 16(2N - 1) + 6 \log_2 N$ operations. Asymptotically, the sequential algorithm presents computational complexity

$$10MN \log_2 N$$

floating point operations, which essentially correspond to the the metric of two radix-2 Cooley–Turkey FFT implementations [17] applied over M data sets of size N.

To observe the scalability of the algorithm, we ran the parallel solver for the Dirichlet problem using the trapezoidal rule for numerical integration. Timings were taken for two sets of data. For a fixed number $N = 2048$ of angular grid points, three distinct numbers of radial grid points were employed: $M = 512$, 1024, and 2048. Fig. 16(a) presents plots for the actual running times when allocating 2, 4, 6, 8, 10, 12, 14, and 16 processors. For the second set, Fig. 16(b) contains the timings for three distinct numbers of angular grid points ($N = 512$, 1024, and 2048) on a discretization with a fixed number of radial grid points $M = 2048$. An immediate observation is that larger levels of granularity correspond to more computational work performed locally on each processor and, therefore, better performance for the algorithm. In fact, the problem of size $M = N = 2048$ scales better than the smaller cases. Nevertheless, savings in computational timings for an increasing number of processors can be observed even for the smaller problems because of the low overhead for interprocessor communication through the shared memory.

To infer the degree of parallelism of our implementation of the fast Poisson solver, we present speedups in a coarse-grained data distribution. Note that the algorithm takes advantage of data cache for small or even medium problem sizes. It means that comparing the running time for a single processor against the time obtained in a multiprocessor architecture may indicate super-linear speedups as a result of smaller amounts of data assigned to each node of the multiprocessor system. Data may reside on cache for a



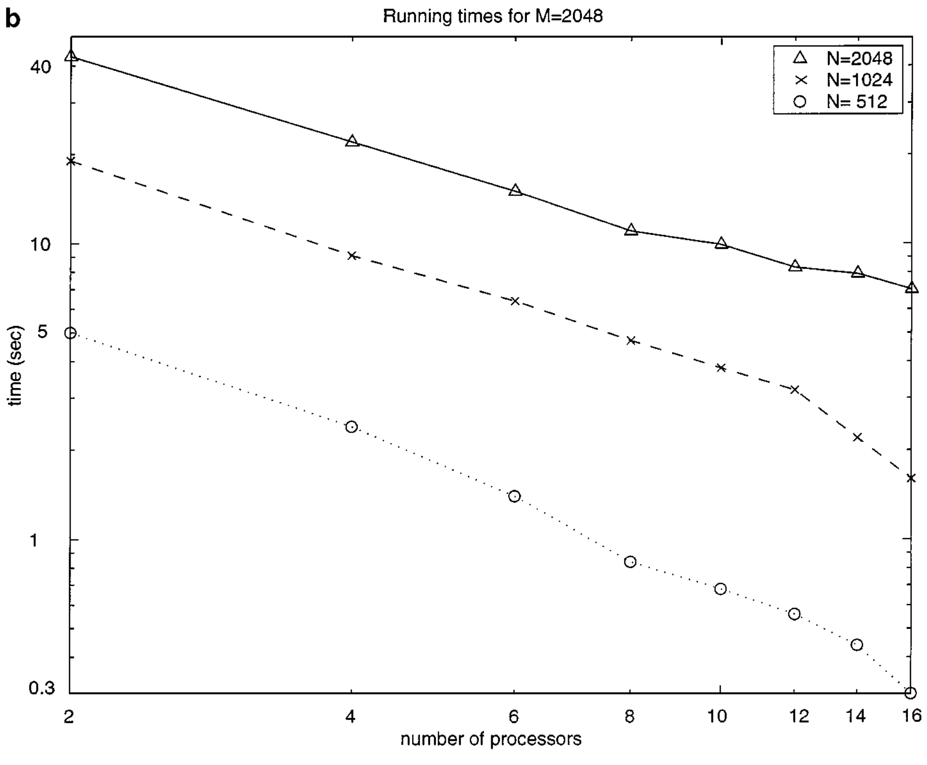


FIG. 16. Scalability of the parallel implementation of the fast solver for the Dirichlet problem: (a) timings for a fixed number of angular points $N = 2048$ and distinct number of radial points $M = 512$, 1024, and 2048; (b) timings for a fixed number of radial points $M = 2048$ and distinct number of angular points $N = 512$, 1024, and 2048. All plots in (a) and (b) are log–log plots.

TABLE XII
Speedups for the Parallel Algorithm for a Problem of Size $M = N = 2048$

Speedups for $M = N = 2048$			
Number of processors	Timing (sec.)	Speedup	Efficiency
1	78	1.0	1.00
2	43	1.8	0.91
4	22	3.5	0.89
6	15	5.2	0.87
8	11	7.1	0.88
10	9.7	8.0	0.81
12	8.3	9.4	0.78
14	7.8	10	0.71
16	7.0	11	0.69

sufficiently large number of processors. To overcome this problem, we compare running times for problem size $M = N = 2048$ to guarantee that multiple accesses occur between data cache and shared memory even when 16 processors are in use. Table XII presents the timings for the parallel algorithm using up to 16 processors. The timing for a single processor was extracted from Table XI. Speedup S is defined as the ratio of the time required to solve the problem on a single processor, using the purely sequential Algorithm 3.1, to the time taken to solve the same problem using P processors. Efficiency E indicates the degree of speedup achieved by the system and is defined as $E = S/P$. The lowest admissible value for efficiency $E = 1/P$ corresponds to leaving $P - 1$ processors idle and having the algorithm executed sequentially on a single processor. The maximum admissible value for efficiency $E = 1$ indicates all processors devoting the entire execution time to perform computations of the original Algorithm 3.1 without any overlapping. Speedup and efficiency are shown in Table XXII. These results demonstrate that the additional computational work introduced by using partial sums, as described in Section 4.1, does not increase the complexity of the algorithm. By comparing the asymptotic estimate for the parallel running time (29) against the full estimate (28), one can observe that this extra computational work does not increase the asymptotic estimate.

We see that efficiency and speedup of the parallel algorithm gradually decrease with an increasing number of processors, which is quite expected. However, at the rate it does so may raise some questions about whether our method scales well or not. This issue can be properly addressed by looking at how the parallel algorithms for this class of problems perform in general. We have already addressed this issue in Section 4.2 where Eq. (34) shows that the efficiency is approximately $\mathcal{O}(1/(1 + cP^2))$ which is consistent with the data in Table XXII. It is worth pointing out that an efficiency of 69% or speedup of 11 for an approximate four million points (see last line in Table XII) for this class of problems is not atypical. This is because the algorithm (see Section 4) uses two sets of data: Data set in the radial direction need to be constructed from the data set in the circumferential direction, and this requires communication among various processors. This communication cost is perhaps somewhat large, but this is not so unusual with problems of this kind. In fact, we have shown in Section 4.4 that FACR-based methods also present the same behavior. Table

XII shows that our algorithm scales well and is very competitive when compared with other current approaches.

6. CONCLUSIONS

In this paper, we presented a fast algorithm for solving the Poisson equation with either Dirichlet or Neumann conditions. The resulting algorithm presents a lower computational complexity when compared against standard procedures based solely on numerical integration. The method is based on exact analysis which provides a more accurate algorithm. The representation of the solution using Fourier coefficients and convolution properties provide a very accurate numerical solution, even for problems with sharp variations on inhomogeneous terms. We also have shown that the mathematical foundation of the algorithm allows us to define high-order one-dimensional integration schemes without increasing the number of grid points on the domain.

From a computational point of view, data locality was preserved leading to an efficient use of cache. By reformulating the inherently sequential recurrenœs present in the sequential algorithm, we were able to obtain a parallel version of the solver characterized by a reduced amount of communication, and message lengths depending only on the number of Fourier coefficients being evaluated. We have shown that the new approach can be defined in a way that it presents the same numerical stability as in the sequential algorithm. The parallel solver is very suited for distributed and shared memory systems. A timing model for the algorithm was presented to provide a better understanding of the algorithm and to provide performance prediction.

APPENDIX: MATHEMATICAL PROOFS

Proof of Theorem 2.1. We recall that the solution w of the homogeneous problem (3) can be derived by using the Poisson integral formula [9]

$$w(r,\alpha) = \frac{1}{2\pi}\int_0^{2\pi} \varphi(\tau)\, K\left(\frac{r}{R}, \alpha - \tau\right) d\tau, \quad 0 < r < R,$$

where the boundary conditions are defined by

$$\varphi(\tau) = g(\tau) - v(R,\tau), \tag{38}$$

and the Poisson kernel is

$$K(\rho,\tau) = \frac{1-\rho^2}{1+\rho^2 - 2\rho\cos\tau}, \quad 0 \le \rho < 1. \tag{39}$$

A Fourier representation of w is obtained by considering

$$K(\rho,\tau) = \frac{1-\rho^2}{(1-\rho\cos\tau)^2 + (\rho\sin\tau)^2} = \frac{1-|z|^2}{|1-z|^2}$$

for $z = \rho e^{i\tau} = \rho(\cos\tau + i\sin\tau)$. Thus,

$$K(\rho,\tau) = \frac{1 - z\bar{z}}{(1-z)(1-\bar{z})} = \frac{Re(1 - z\bar{z} - \bar{z} + z)}{(1-z)(1-\bar{z})}$$
$$= Re\left(\frac{(1+z)(1-\bar{z})}{(1-z)(1-\bar{z})}\right) = Re\left(\frac{1+z}{1-z}\right).$$

Since $|z| < 1$

$$K(\rho,\tau) = Re((1+z)(1+z+z^2+\cdots))$$
$$= Re(1 + 2(z + z^2 + \cdots)) = \sum_{n=-\infty}^{+\infty} \rho^{|n|} e^{in\tau}.$$

Consequently, for the Fourier representation $K(\rho,\tau) = \sum_n K_n(\rho) e^{in\tau}$ we have

$$K_n(\rho) = \rho^{|n|}. \tag{40}$$

Let $w(r,\alpha) = \sum_n w_n(r) e^{in\alpha}$, where

$$w_n(r) = \frac{1}{2\pi}\int_0^{2\pi} \varphi(\tau)\left[\frac{1}{2\pi}\int_0^{2\pi} K\left(\frac{r}{R}, \alpha - \tau\right) e^{-in\alpha}\, d\alpha\right] d\tau$$
$$= \frac{1}{2\pi}\int_0^{2\pi} \varphi(\tau)\left[\frac{1}{2\pi}\int_0^{2\pi} K\left(\frac{r}{R}, \alpha - \tau\right) e^{-in(\alpha-\tau)}\, d\alpha\right] e^{-in\tau}\, d\tau$$
$$= \frac{1}{2\pi}\int_0^{2\pi} \varphi(\tau) K_n\left(\frac{r}{R}\right) e^{-in\tau}\, d\tau = \varphi_n K_n\left(\frac{r}{R}\right),$$

and

$$\varphi_n = \frac{1}{2\pi}\int_0^{2\pi} \varphi(\tau) e^{-in\tau}\, d\tau. \tag{41}$$

Equation (40) leads to

$$w_n(r) = \varphi_n K_n\left(\frac{r}{R}\right) = \left(\frac{r}{R}\right)^{|n|} \varphi_n. \tag{42}$$

The principal solution v defined by the integral in (5) can be evaluated by considering the splitting of the domain defined by

$$_0\Omega_r = B(0; r),$$
$$_{r-\varepsilon}\Omega_{r+\varepsilon} = B(0; r+\varepsilon) - B(0; r-\varepsilon),$$
$$_{r-\varepsilon}\Omega^*_{r+\varepsilon} = {}_{r-\varepsilon}\Omega_{r+\varepsilon} - B(x; \varepsilon),$$

and

$$ {}_r\Omega_1 = B(0;1) - B(0;r). $$

Therefore,

$$
\begin{aligned}
v(x) &= \int_B f(\eta)\frac{1}{2\pi}\log|x-\eta|\,d\eta = \frac{1}{2\pi}\lim_{\varepsilon\to 0}\int_{B-B(x;\varepsilon)} f(\eta)\log|x-\eta|\,d\eta \\
&= \frac{1}{2\pi}\lim_{\varepsilon\to 0}\left\{\int_{{}_0\Omega_{r-\varepsilon}} f(\eta)\log|x-\eta|\,d\eta + \int_{{}_{r-\varepsilon}\Omega^*_{r+\varepsilon}} f(\eta)\log|x-\eta|\,d\eta \right.\\
&\qquad \left. + \int_{{}_{r+\varepsilon}\Omega_R} f(\eta)\log|x-\eta|\,d\eta\right\},
\end{aligned}
$$

and for $x = re^{i\alpha}$, the Fourier decomposition is given by

$$
\begin{aligned}
v_n(r) &= \frac{1}{4\pi^2}\left\{\int_{{}_0\Omega_r} f(\eta)\int_0^{2\pi}\log|x-\eta|\,e^{-in\alpha}d\alpha\,d\eta + \int_{{}_r\Omega_R} f(\eta)\int_0^{2\pi}\log|x-\eta|\,e^{-in\alpha}d\alpha\,d\eta\right\}\\
&\quad + \frac{1}{2\pi}\int_0^{2\pi}\left(\frac{1}{2\pi}\lim_{\varepsilon\to 0} I_\varepsilon\right)e^{-in\alpha}\,d\alpha,
\end{aligned}
$$

where

$$ I_\varepsilon = \int_{{}_{r-\varepsilon}\Omega^*_{r+\varepsilon}} f(\eta)\log|x-\eta|\,d\eta. $$

Since

$$
\begin{aligned}
|I_\varepsilon| &\le \sup_{\eta\in{}_{r-\varepsilon}\Omega^*_{r+\varepsilon}} f(\eta) \sup_{\eta\in{}_{r-\varepsilon}\Omega^*_{r+\varepsilon}} |\log|x-\eta||\,\pi((r+\varepsilon)^2-(r-\varepsilon)^2)\\
&= -\sup_{\eta\in{}_{r-\varepsilon}\Omega^*_{r+\varepsilon}} f(\eta)\,4\pi\varepsilon\log\varepsilon,
\end{aligned}
$$

we have $\lim_{\varepsilon\to 0}|I_\varepsilon| = 0$. If $\eta = \rho e^{i\tau}$ and $\theta = \alpha-\tau$, then

$$
\begin{aligned}
v_n(r) &= \frac{1}{2\pi}\left\{\iint_{{}_0\Omega_r} \rho f(\rho,\tau)e^{-in\tau}\left(\frac{1}{2\pi}\int_{-\tau}^{2\pi-\tau}\log|x-\eta|e^{-in\theta}d\theta\right)d\tau\,d\rho\right\}\\
&\quad + \frac{1}{2\pi}\left\{\iint_{{}_r\Omega_R} \rho f(\rho,\tau)e^{-in\tau}\left(\frac{1}{2\pi}\int_{-\tau}^{2\pi-\tau}\log|x-\eta|e^{-in\theta}d\theta\right)d\tau\,d\rho\right\}.
\end{aligned}
$$

It implies

$$v_n(r) = \int_0^r f_n(\rho)\,\tilde{G}_n(r,\rho)\,\rho\,d\rho + \int_r^R f_n(\rho)\,\tilde{G}_n(r,\rho)\,\rho\,d\rho \qquad 0 \le r \le R, \tag{43}$$

where f_n and $\tilde{G}_n$ are the nth Fourier coefficients of f and $\tilde{G}$, respectively, with

$$\tilde{G}(r,\rho,\theta) = \log|x-\eta| = \log|r^2+\rho^2-2r\rho\cos\theta|^{1/2}.$$

To derive $\tilde{G}_n$ we define $\zeta = \frac{\rho}{r}e^{-i\theta}$ for $r > \rho$, and $\xi = \frac{r}{\rho}e^{i\theta}$ for $r < \rho$, then

$$\log|x-\eta| = \begin{cases} \log(r|1-\zeta|), & r > \rho, \\ \log(\rho|1-\xi|), & r < \rho. \end{cases} \tag{44}$$

Moreover, since $|\zeta| < 1$ for $r > \rho$

$$\begin{aligned} \log|1-\zeta| &= \frac{1}{2}(\log(1-\zeta) + \log(1-\bar{\zeta})) = -\sum_{n=1}^{\infty}\frac{\zeta^n}{2n} - \sum_{n=1}^{\infty}\frac{\bar{\zeta}^n}{2n} \\ &= -\sum_{n=1}^{\infty}\frac{1}{2n}(\zeta^n+\bar{\zeta}^n) = -\sum_{n=1}^{\infty}\frac{|\zeta|^n}{2n}(e^{-in\theta}+e^{in\theta}) \\ &= -\sum_{n\ne 0}\frac{|\zeta|^{|n|}}{2|n|}e^{in\theta}. \end{aligned}$$

A similar result hold for $\log|1-\xi|$ when $r < \rho$. Equation (44) leads to

$$\log|x-\eta| = \begin{cases} \log r - \sum_{n\ne 0}\frac{1}{2|n|}\left(\frac{\rho}{r}\right)^{|n|}e^{in\theta}, & r > \rho, \\ \log\rho - \sum_{n\ne 0}\frac{1}{2|n|}\left(\frac{r}{\rho}\right)^{|n|}e^{in\theta}, & r < \rho. \end{cases} \tag{45}$$

Using the above decomposition, Eq. (43) can be rewritten as in (8) with p_n and q_n as defined in (9) and (10).

Recall that the solution of the Dirichlet problem (1) is given by

$$u = v + w.$$

Therefore, the Fourier coefficients $u_n(r)$ of $u(r,\cdot)$ are obtained from (42) as

$$u_n(r) = v_n(r) + w_n(r) = v_n(r) + \left(\frac{r}{R}\right)^{|n|}\varphi_n, \tag{46}$$

and from (38) we obtain (7).

Proof of Corollary 2.2 We first want to show that recurrences (13) and (14) evaluate

$$v_n^-(r) = \int_0^r \frac{\rho}{2n}\left(\frac{r}{\rho}\right)^n f_n(\rho)\,d\rho, \quad n < 0, \tag{47}$$

$$v_n^+(r) = -\int_r^R \frac{\rho}{2n}\left(\frac{r}{\rho}\right)^n f_n(\rho)\,d\rho, \quad n > 0. \tag{48}$$

If $n < 0$, then

$$v_n^-(r_j) = \left(\frac{r_j}{r_i}\right)^n \int_0^{r_i} \frac{\rho}{2n}\left(\frac{r_i}{\rho}\right)^n f_n(\rho)\,d\rho + C_n^{i,j} = \int_0^{r_j} \frac{\rho}{2n}\left(\frac{r_j}{\rho}\right)^n f_n(\rho)\,d\rho,$$

which implies (47). If $n > 0$, the same argument holds for (48). From Theorem 2.1 we have

$$v_n(r) = \begin{cases} \int_0^r \frac{\rho}{2n}\left(\frac{r}{\rho}\right)^n f_n(\rho)\,d\rho + \int_r^R \frac{\rho}{2n}\left(\frac{\rho}{r}\right)^n f_n(\rho)\,d\rho, & n < 0, \\ \int_0^r \rho \log r \;\; f_0(\rho)\,d\rho + \int_r^R \rho \log\rho \;\; f_0(\rho)\,d\rho, & n = 0, \\ \int_0^r \frac{-\rho}{2n}\left(\frac{\rho}{r}\right)^n f_n(\rho)\,d\rho + \int_r^R \frac{-\rho}{2n}\left(\frac{r}{\rho}\right)^n f_n(\rho)\,d\rho, & n > 0. \end{cases} \tag{49}$$

Consequently, if $n < 0$ we have from (47)

$$v_n(r_i) = v_n^-(r_i) - \int_{r_i}^R \frac{\rho}{-2n}\left(\frac{r_i}{\rho}\right)^{-n} f_n(\rho)\,d\rho,$$

and since $f_n = \overline{f_{-n}}$, Eq. (48) leads to

$$v_n(r_i) = v_n^-(r_i) + \overline{v_{-n}^+(r_i)}$$

as in (15). The proof is similar for $n > 0$ since $v_n(r_i) = \overline{v_{-n}(r_i)}$.

Proof of Corollary 2.3. For $n = 0$ we have

$$\sum_{i=2}^{l} C_0^{i-1,i} = \int_0^{r_l} \rho f_0(\rho)\,d\rho \quad \text{and} \quad \sum_{i=l}^{M-1} D_0^{i,i+1} = \int_{r_l}^R \rho \log\rho f_0(\rho)\,d\rho,$$

thus $v_0(r_l)$ is given as in (49). If $n < 0$, Eq. (11) and (47) give

$$\sum_{i=2}^{l} \left(\frac{r_l}{r_i}\right)^n C_n^{i-1,i} = \int_0^{r_l} \frac{\rho}{2n}\left(\frac{r_l}{\rho}\right)^n f_n(\rho)\,d\rho = v_n^-(r_l),$$

and Eqs. (12) and (48) lead to

$$\sum_{i=l}^{M-1} \left(\frac{r_i}{r_l}\right)^n \overline{D_{-n}^{i,i+1}} = -\int_{r_l}^R \frac{\rho}{-2n}\left(\frac{r_l}{\rho}\right)^{-n} \overline{f_{-n}(\rho)}\,d\rho = \overline{v_n^+(r_l)}.$$

From Corollary 2.2 the above terms sum up

$$v_n(r_l) = v_n^-(r_l) + \overline{v_n^+(r_l)}.$$

The proof is similar for the case $n > 0$.

Proof of Theorem 2.2. Consider the normal derivative of the principal solution v obtained from the gravitational potential (5)

$$\alpha(x) = \frac{\partial v}{\partial \vec{n}}(x) = \int_B f(\eta) \frac{\partial}{\partial \vec{n}} G(x, \eta)\, d\eta, \qquad x \in \partial B,$$

where the normal derivative for the Green's function (6) on ∂B is given by

$$\frac{\partial}{\partial \vec{n}} G(x, \eta) = \frac{1}{2\pi} \frac{\partial}{\partial \vec{n}} \log|x - \eta| = \frac{\langle 2(x_1 - \eta_1, x_2 - \eta_2), \frac{1}{|x|}(x_1, x_2)\rangle}{4\pi |x - \eta|^2} = \frac{\langle x, x - \eta\rangle}{2\pi R|x - \eta|^2},$$

with $x = (x_1, x_2)$ and $\eta = (\eta_1, \eta_2)$. One can rewrite the above derivative as

$$\begin{aligned}
\frac{\partial}{\partial \vec{n}} G(x, \eta) &= \frac{1}{4\pi R} \frac{2\langle x, x - \eta\rangle + \langle \eta, \eta\rangle - \langle \eta, \eta\rangle}{\langle x - \eta, x - \eta\rangle} \\
&= \frac{1}{4\pi R} \frac{\langle x, x - \eta\rangle - \langle \eta, x - \eta\rangle + \langle x, x\rangle - \langle \eta, \eta\rangle}{\langle x - \eta, x - \eta\rangle} \\
&= \frac{1}{4\pi R} \frac{\langle x - \eta, x - \eta\rangle + \langle x, x\rangle - \langle \eta, \eta\rangle}{\langle x - \eta, x - \eta\rangle} \\
&= \frac{1}{4\pi R}\left(1 + \frac{R^2 - |\eta|^2}{|x - \eta|^2}\right) = \frac{1}{4\pi R} + \frac{1}{4\pi R} \frac{R^2 - |\eta|^2}{|x - \eta|^2}
\end{aligned}$$

for $x \in \partial B$. Therefore, $\frac{\partial}{\partial \vec{n}} G(x, \eta)$ can be expanded in the same way as the Poisson kernel (39) by noticing that

$$\frac{\partial}{\partial \vec{n}} G\left(\frac{x}{R}, \frac{\eta}{R}\right) = \frac{1}{4\pi R} + \frac{1}{4\pi R} \frac{1 - \gamma^2}{1 + \gamma^2 - 2\gamma \cos\theta}$$

for $x = Re^{i\alpha}$ and $\eta = \rho e^{i\tau}$, where $\theta = \alpha - \tau$, $\gamma = \frac{|\eta|}{R}$, and $0 < \gamma \leq 1$. That is,

$$\frac{\partial}{\partial \vec{n}} G\left(\frac{x}{R}, \frac{\eta}{R}\right) = \frac{1}{4\pi R} + \frac{1}{4\pi R} \sum_{n=-\infty}^{+\infty} \left(\frac{\rho}{R}\right)^{|n|} e^{in\theta}.$$

Therefore, for $n \neq 0$ we have

$$\alpha_n = 2\pi \int_0^R \frac{\rho}{4\pi R} \left(\frac{\rho}{R}\right)^{|n|} f_n(\rho)\, d\rho = \frac{1}{2R} \int_0^R \rho \left(\frac{\rho}{R}\right)^{|n|} f_n(\rho)\, d\rho,$$

and comparing the above equation against the Fourier coefficients of v given in (8) as

$$v_n(R) = \int_0^R \frac{-\rho}{2|n|} \left(\frac{\rho}{R}\right)^{|n|} f_n(\rho) \qquad \text{for } n \neq 0$$

we have

$$\alpha_n = -\frac{|n|}{R} v_n(R) \quad \text{for } n \neq 0. \tag{50}$$

To obtain the solution u of the Neumann problem (18) in the form $u = v + w$ as before, we need to define boundary conditions φ of the homogeneous Dirichlet problem

$$\begin{aligned} \Delta w &= 0 \quad \text{in} \quad B \\ w &= \varphi \quad \text{on} \quad \partial B, \end{aligned}$$

which corresponds to the Neumann conditions

$$\frac{\partial}{\partial \vec{n}} w(R) = \psi - \frac{\partial}{\partial \vec{n}} v(R) = \psi - \alpha \tag{51}$$

on ∂B. From the relation given in (42) we obtain

$$\frac{\partial}{\partial \vec{n}} w_n(R) = \frac{|n|}{R} \varphi_n,$$

and from equations (50) and (51) we have

$$\varphi_n = \frac{R}{|n|} \frac{\partial}{\partial \vec{n}} w_n(R) = \frac{R}{|n|}(\psi_n - \alpha_n) = \frac{R}{|n|}\psi_n + v_n(R), \quad \text{for} \quad n \neq 0.$$

Consequently, to solve the Neumann problem (18) the above definition for the boundary condition φ is used in (46) leading to the Fourier coefficients described in (19) for $n \neq 0$.

For the case $n = 0$, ψ_0 is uniquely defined by the compatibility condition derived from Green's theorem:

$$\psi_0 = \frac{1}{2\pi} \int_0^{2\pi} \psi(\tau)\, d\tau = \frac{1}{2\pi} \int_B \Delta u(\eta)\, d\eta = \frac{1}{2\pi} \int_B f(\eta)\, d\eta = \int_0^R \rho f_0(\rho)\, d\rho.$$

ACKNOWLEDGMENTS

This material is based in part upon work supported by the Texas Advanced Research Program under Grant TARP-97010366-030. We sincerely thank the referees for their constructive criticisms.

REFERENCES

1. J. Anderson, S. Amarasingle, and M. S. Lam, Data and computation transformations for multiprocessors, in *Proceedings 5th Symposium on Principles and Practice of Parallel Programming* (ACM SIGPLAN, July 1995).
2. L. Borges and P. Daripa, A parallel version of a fast algorithm for singular integral transforms, *Numer. Algor.* **23**, 71 (2000).
3. L. Borges and P. Daripa, A parallel solver for singular integrals, in *Proceedings of PDPTA'99—International Conference on Parallel and Distributed Processing Techniques and Applications* (Las Vegas, Nevada, June 28–July 1, 1999), Vol. III, pp. 1495–1501.
4. W. Briggs, L. Hart, R. Sweet, and A. O'Gallagher, Multiprocessor FFT methods, *SIAM J. Sci. Stat. Comput.* **8**, 27 (1987).

5. W. Briggs and T. Turnbull, Fast Poisson solvers for mimd computers, *Parallel Comput.* **6**, 265 (1988).
6. T. F. Chan and D. C. Resasco, A domain-decomposed fast Poisson solver on a rectangle, *SIAM J. Sci. Stat. Comput.* **8**, S14 (1987).
7. P. Daripa, A fast algorithm to solve the Beltrami equation with applications to quasiconformal mappings, *J. Comput. Phys.* **106**, 355 (1993).
8. P. Daripa and D. Mashat, Singular integral transforms and fast numerical algorithms, *Numer. Algor.* **18**, 133 (1998).
9. M. D. Greenberg, *Application of Green's Functions in Science and Engineering* (Prentice Hall, New York, 1971).
10. Hewlett-Packard, *HP 9000 V-Class Server Architecture*, 2dn ed. (Hewlett-Packard, March 1998).
11. R. Hockney and C. Jesshope, *Parallel Computers: Archithecture, Programming and Algorithms* (Hilger, Bristol, 1981).
12. R. W. Hockney, A fast direct solution of Poisson equation using Fourier analysis, *J. Assoc. Comput. Mach.* **8**, 95 (1965).
13. E. Houstis, R. Lynch, and J. Rice, Evaluation of numerical methods for ellipitic partial differential equations, *J. Comput. Phys.* **27**, 323 (1978).
14. L. S. Johnsson and N. P. Pitsianis, Parallel computation load balance in parallel FACR, in *High Performance Algorithms for Structured Matrix Problems*, edited by P. Arbenz, M. Paprzycki, A. Sameh, and V. Sarin (Nova Science Publishers, Inc., 1998).
15. V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing* (Benjamin/Cummings, Redwood City, CA, 1994).
16. J.-Y. Lee and K. Jeong, A parallel Poisson solver using the fast multipole method on networks of workstations, *Comput. Math. Appl.* **36**, 47 (1998).
17. C. V. Loan, Computational Frameworks for the Fast Fourier Transform (Soc. for Industr. & Appl. Math., Philadelphia, 1992).
18. A. McKenney, L. Greengard, and A. Mayo, A fast Poisson solver for complex geometries, *J. Comput. Phys.* **118**, 348 (1995).
19. P. Pacheco, *Parallel Programming with MPI* (Morgan Kaufmann, San Francisco, CA, 1997).
20. J. Rice, E. Houstis, and R. Dyksen, A population of linear, second order, elliptic partial differential equations on rectangular domains, I, II, *Math. Comput.* **36**, 475 (1981).
21. A. Sameh, A fast Poisson solver for multiprocesors, in *Elliptic Problem Solvers II*,edited by G. Birkhoff and A. Schoenstadt (Academic Press, Orlando, 1984) pp. 175–186.
22. J. Singh, W. Weber, and A. Gupta, SPLASH: Stanford parallel applications for shared-memory, *Comput. Arch. News* **20**, 5 (1992).
23. G. Sköllermo, A Fourier method for the numerical solution of Poisson's equation, *Math. Comput.* **29**, 697 (1975).
24. P. N. Swarztrauber, The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle, *SIAM Rev.* **19**, 491 (1977).
25. C. Temperton, On the FACR(l) algorithm for the discrete Poisson equation, *J. Comput. Phys.* **34**, 315 (1980).