A Parallel Domain Decomposition Method for the Helmholtz Equation

Elisabeth Larsson
Sverker Holmgren

Technical report 2000-006
March 2000
ISSN 1404-3203

Department of Information Technology
Uppsala University

A PARALLEL DOMAIN DECOMPOSITION METHOD FOR THE HELMHOLTZ EQUATION

ELISABETH LARSSON† AND SVERKER HOLMGREN†

Abstract. A parallel solver for the Helmholtz equation in a domain consisting of layers with different material properties is presented. A fourth-order accurate finite difference discretization is used. The arising system of equations is solved with a preconditioned Krylov subspace method. A domain decomposition framework is employed, where fast transform subdomain preconditioners are used. Three ways of treating the Schur complement of the preconditioner are investigated, and the corresponding preconditioned iterative methods are compared with a standard direct method. It is noted that the convergence rate of the iterative methods is closely related to how the Schur complement system for the preconditioner is formed, and how accurately it is solved. However, in almost all cases, the gain in both memory requirements and arithmetic complexity is large compared with the direct method. Furthermore, the gain increases with problem size, allowing problems with many unknowns to be solved efficiently. The efficiency is further improved by parallelization using message-passing, enabling us to solve even larger Helmholtz problems in less time.

Key words. Helmholtz equation, domain decomposition, preconditioned iterative method, Schur complement, parallelization

AMS subject classifications. 65N55, 65F10, 65N22, 65Y05

1. Introduction. The interest for numerical simulations of wave propagation phenomena is rapidly increasing. The fast development in the field of computational wave propagation has become possible both because of the development of effective and accurate numerical methods and more powerful computers. Today, a wide range of numerical methods are used for application problems, such as radar cross sections, electromagnetic compatibility, lightning, antennas, bioelectromagnetics, microwave components, aeroacoustics, underwater acoustics, and geophysics.

Normally, accurate solution of interesting application problems puts very high demands on the capacity of the computer, even when an effective numerical scheme is employed. Only high-performance parallel computers have sufficient computational speed and sufficiently large memories for performing the computations. Hence, it is important to study parallel implementations on such computers.

In this paper, a class of solution methods for the Helmholtz equation in a multi-layer domain is considered. Equations of this type describe wave propagation in the frequency domain. Applications in acoustics and electromagnetics are described, e.g., in [8] and [2]. For many wave propagation computations in the frequency domain, e.g., wave scattering and cross section determination, methods using boundary integral formulations are employed [15]. Such methods are often effective, and also simplify the treatment of infinite domains. However, it is not possible to employ methods of boundary integral type when the medium has smoothly varying material properties. In [5, and the references therein], fictitious domain methods for solving time-harmonic wave propagation problems are described. Methods in this class are effective if the material parameters are constant in large parts of the domain. For cases where the material properties vary a lot, fictitious domain methods are applicable, but it is not clear how effective they would be. The methods presented below are applicable to problems where the medium consists of layers with different material parameters, and also when the parameters vary smoothly within the layers. To handle discontinuous

†Department of Scientific Computing, Information Technology, Uppsala University, Box 120, SE-751 04 Uppsala, Sweden (bette@tdb.uu.se, sverker@tdb.uu.se).

material parameters at subdomain boundaries, a domain decomposition framework is employed and the resulting system of equations is solved using a preconditioned iterative method. A first version of the solution scheme is described in [10]. In the presentation below, we focus on new versions of the algorithm, suitable for solving very large problems using parallel computers.

2. The problem. We study the propagation of a time-harmonic wave in two space dimensions, where we assume that the physical domain is divided into a number of layers. For computational purposes, each layer may be further divided, yielding a total of n_d subdomains. A brief discussion on how to generalize the methods to three space dimensions is given in [14].

As an example of a typical problem setting, we have chosen to study acoustical wave propagation. Other types of wave propagation, e.g., electromagnetic wave propagation, yields similar computational problems that can be solved by the same class of methods [10].

2.1. Governing equation. Time-harmonic sound propagation in a medium with density $\rho(x_1,x_2)$ and sound speed $c(x_1,x_2)$ is governed by the Helmholtz equation

$$-\rho(x_1,x_2)\nabla\cdot\left(\frac{1}{\rho(x_1,x_2)}\nabla u(x_1,x_2)\right)-\kappa(x_1,x_2)^2u(x_1,x_2)=0, \tag{2.1}$$

where $\kappa=\frac{2\pi f}{c}(1+i\delta)$ is the wave number, $\delta(x_1,x_2)$ is an attenuation parameter, and f denotes the frequency of the sound. The acoustic pressure is given by $\mathbb{R}\mathrm{e}(ue^{-i2\pi ft})$.

We assume that each subdomain is mapped onto the unit square $0\le\xi_1,\xi_2\le1$, by an orthogonal transformation. Employing an orthogonal mapping is not critical, but it simplifies the boundary conditions and does not introduce additional terms in the transformed equation. The partial differential equation (2.1) is transformed into

$$-\rho\frac{\partial}{\partial\xi_1}\left(\frac{a}{\rho}\frac{\partial u}{\partial\xi_1}\right)-\rho\frac{\partial}{\partial\xi_2}\left(\frac{a^{-1}}{\rho}\frac{\partial u}{\partial\xi_2}\right)-\kappa^2eu=0, \tag{2.2}$$

where $a(\xi_1,\xi_2)$ and $e(\xi_1,\xi_2)$ are metric coefficients.

2.2. An application. As an example, we consider underwater sound propagation. In such a problem, there will typically be one or more water layers, possibly with abrupt changes in salinity or temperature at the layer interfaces, and one or more bottom sediment layers, where the sound absorption increases with depth. In Figure 2.1, a physical domain with two layers and five subdomains is shown.

We assume pressure release at Γ_0, and that the absorption in the sediment layers is large enough to make the sound level at Γ_{n_d} negligible, yielding the boundary conditions

$$u=0 \text{ at } \Gamma_0 \text{ and } \Gamma_{n_d}. \tag{2.3}$$

At the subdomain interfaces, the pressure, which is proportional to u, and the normal velocity are continuous. In the computational domain, the second condition is given by

$$\frac{1}{\rho\eta_1}\frac{\partial u}{\partial\xi_1} \text{ continuous across } \Gamma_j, \quad j=1,\ldots n_d-1, \tag{2.4}$$

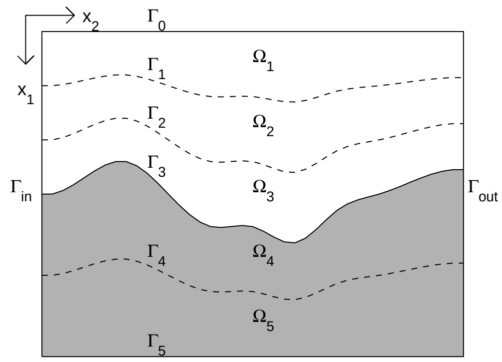

FIG. 2.1. *A physical domain divided into five subdomains. The problem has one water and one sediment layer.*

where $\eta_1(\xi_1, \xi_2)$ is a metric coefficient. The physical domain is assumed to be infinite in the ξ_2-direction, and at the artificial boundaries Γ_{in} and Γ_{out}, Dirichlet–to–Neumann map boundary conditions [9] are employed [10]. All waves are assumed to originate from a truncated point source located on Γ_{in}, yielding the transparent boundary conditions

$$\begin{aligned} -\frac{\partial u}{\partial \xi_2}(\xi_1, 0) - i \sum_{m=1}^{\mu_\ell} \sqrt{-\lambda_m} \langle \psi_m, u(\cdot, 0) \rangle \psi_m(\xi_1) \\ = -2i \sum_{m=1}^{\mu_\ell} A_m \sqrt{-\lambda_m} \psi_m(\xi_1), \end{aligned} \tag{2.5}$$

$$\frac{\partial u}{\partial \xi_2}(\xi_1, 1) - i \sum_{m=1}^{\mu_r} \sqrt{-\lambda_m} \langle \psi_m, u(\cdot, 1) \rangle \psi_m(\xi_1) = 0. \tag{2.6}$$

The eigenfunctions ψ_m and the corresponding eigenvalues λ_m are solutions of Sturm–Liouville problems at Γ_{in} and Γ_{out}. The eigenmodes are conjugate with respect to the bilinear form $\langle \cdot, \cdot \rangle$, defined by

$$\langle \psi_m, \psi_n \rangle \equiv \int_\Gamma \frac{\eta_1}{\rho} \psi_m \psi_n \, d\xi_1, \tag{2.7}$$

where Γ is one of the vertical boundaries. The coefficients A_m define the truncated point source, and μ_ℓ and μ_r are the cutoff limits. High modes are strongly evanescent and neglected in the boundary conditions, since the bottom is assumed to be locally flat near the boundaries.

3. The discretized problem. For discretizations of the Helmholtz equation, the number of gridpoints per wavelength required for computing a solution of given accuracy grows with the frequency. This makes it hard to solve high frequency problems. Employing standard second-order accurate discretization methods often results in unacceptable memory requirements and arithmetic complexities, and only high-order accurate discretizations are viable [1], [18], [7], [14], and [13]. In the work presented below, a fourth order accurate finite difference discretization is utilized.

Employing a finite difference method results in a spatial difference operator that corresponds to a highly structured matrix, which makes it easier to design effective preconditioners. If the geometry is too complicated, finite difference discretizations cannot be easily used. However, one of the major reasons for employing a domain decomposition framework is that complicated geometries can be divided into a number of geometrically simpler subdomains. Accordingly, only problems with subdomains that make finite differences feasible are considered.

In subdomain d, a grid with m_2 points in the horizontal direction and $m_1^{(d)}$ interior points in the vertical direction is defined. On each subdomain interface, there are m_2 gridpoints. The equation (2.2) is discretized by a fourth-order Numerov scheme [13], and the boundary condition (2.4) is discretized by fourth-order one-sided differences. The eigenproblems at the vertical boundaries are solved by a fourth-order finite element method, and the radiation boundary conditions (2.5) and (2.6) are discretized by using Simpson's rule for the integrals and fourth-order one-sided differences for the derivatives. By ordering the discretized equations in a domain decomposition fashion, we arrive at a blocked system of equations $Bu = g$, or

$$\begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \end{pmatrix} = \begin{pmatrix} g_0 \\ g_1 \end{pmatrix}. \tag{3.1}$$

Let $m_1 \equiv \sum_{d=1}^{n_d} m_1^{(d)}$. In (3.1), the $m_2(n_d-1)$-vector u_0 represents unknowns on the interfaces, and the $m_2 m_1$-vector u_1 represents unknowns in the interior of the subdomains. In Figure 3.1, the structure of the matrix B is shown. The dense diagonal blocks in B_{11} are results of the global coupling of the unknowns at the in- and outflow boundaries introduced by the Dirichlet–to–Neumann map boundary conditions.

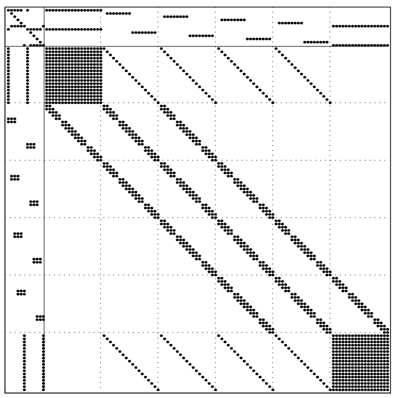

FIG. 3.1. *The structure of B for a small problem, where $n_d = 3$, $m_2 = 6$ and $m_1 = (5, 8, 5)$.*

Note that, when employing an iterative method for solving (3.1), matrix–vector multiplications,

$$\begin{pmatrix} y_0 \\ y_1 \end{pmatrix} = \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}, \tag{3.2}$$

are performed in the iteration process.

4. The iterative method. The coefficient matrix in (3.1) is normally very large and sparse. A viable approach for solving the system of equations is to use a preconditioned iterative method, employing a preconditioner matrix M. It is important to recall that the objective of preconditioning is not to minimize the number of iterations, but to solve the system of equations in minimal time. The extreme case $M = B$, corresponding to a direct solution method, results in prohibitive memory requirements and arithmetic complexity. On the other hand, using $M = I$, corresponding to no preconditioning, results in an iteration that does not converge in a reasonable number of iterations, see, e.g., [14]. Using a standard preconditioner like SSOR often yields unacceptable convergence properties [14]. The challenging balance between using an expensive, almost exact, or a cheap, less accurate, preconditioner is further complicated by the fact that their parallelization properties may be very different. One of the purposes of this paper is to investigate this.

Right preconditioning is utilized, and in iteration i, the preconditioner M_i is given by

$$M_i = \begin{pmatrix} B_{00} + \Delta_i & B_{01} \\ B_{10} & M_{11} \end{pmatrix}. \tag{4.1}$$

The preconditioner solve,

$$\begin{pmatrix} z_0 \\ z_1 \end{pmatrix} = M_i^{-1} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}, \tag{4.2}$$

is performed using the Schur complement algorithm:

1. Solve $M_{11} w_1 = x_1$
2. Solve $C_i z_0 = x_0 - B_{01} w_1$
3. Solve $M_{11} v_1 = B_{10} z_0$
4. $z_1 = w_1 - v_1$

Here, the matrix C_i is the Schur complement of the preconditioner,

$$C_i = (B_{00} + \Delta_i) - B_{01} M_{11}^{-1} B_{10}. \tag{4.3}$$

When using a preconditioner where $\Delta_i \equiv \Delta$, we employ the standard restarted GMRES iteration [17]. When the preconditioner changes during the iteration process, we instead use the flexible, restarted GMRES iteration [16]. The work performed in one iteration using either of these two methods consists of one matrix–vector multiplication (3.2), one preconditioner solve (4.2), and some vector operations. As initial guess, we use $M_0^{-1} g$, which is a good approximation to the solution u if M_0 is a good preconditioner. The convergence criterion used for the experiments presented in Section 7 is

$$\frac{\|Bu_i - g\|_2}{\|g\|_2} < 10^{-5}. \tag{4.4}$$

4.1. The subdomain preconditioner. The largest block in the coefficient matrix in (3.1) is B_{11}, corresponding to the subdomains. Neglecting the couplings introduced by the non-reflecting boundary conditions at the vertical boundaries, B_{11} corresponds to n_d independent Helmholtz problems. This property is employed when constructing M_{11} [10]. The decoupled problems in M_{11} correspond to difference approximations of Helmholtz problems with constant coefficients in the ξ_1-direction, decoupled radiation boundary conditions at the vertical boundaries, and Dirichlet boundary conditions at the horizontal boundaries.

The block structure in M_{11} is the same as in B_{11}, and each block in M_{11} is a normal block approximation [6] of the corresponding block in B_{11}. Block (r, c), $1 \le r, c \le m_2$, in M_{11} has the form

$$M_{11}^{(r,c)} = Q\Lambda^{(r,c)}Q^H, \tag{4.5}$$

where

$$\Lambda^{(r,c)} = \mathrm{diag}(Q^H B_{11}^{(r,c)} Q). \tag{4.6}$$

The normal matrix Q contains n_d sine transform matrices, one for each domain,

$$Q = \begin{pmatrix} S_{m_1{}^{(1)}} & & \\ & \ddots & \\ & & S_{m_1{}^{(n_d)}} \end{pmatrix}. \tag{4.7}$$

Here,

$$S_m(j,k) = \sqrt{\frac{2}{m+1}} \sin\left(jk\frac{\pi}{m+1}\right), \quad 1 \le j, k \le m. \tag{4.8}$$

The choice of $\Lambda^{(r,c)}$ in (4.6) minimizes $\|B_{11} - M_{11}\|_F$ over the matrix class defined in (4.5) [12].

Let Λ denote a matrix where block (r, c) is given by $\Lambda^{(r,c)}$. By employing a factorization of M_{11},

$$M_{11}^{-1} = (I_{m_2} \otimes Q)\Lambda^{-1}(I_{m_2} \otimes Q^H), \tag{4.9}$$

the subdomain preconditioner solve $x = M_{11}^{-1}y$ can be performed as

1. $v = (I_{m_2} \otimes Q^H)y$,
2. Solve $\Lambda z = v$,
3. $x = (I_{m_2} \otimes Q)z$.

The work in these three steps can be done for each subdomain independently. For subdomain d, m_2 sine transforms and inverse sine transforms of length $m_1^{(d)}$, and one solution of $m_1^{(d)}$ narrow banded systems of equations of size m_2 are performed. In the implementation, the matrix M_{11} is not explicitly formed, instead it is represented by the coefficients used for the fast sine transforms, and the factorizations of the narrow banded systems.

4.2. The Schur complement system. The Schur complement matrix C_i in (4.3) is of size $m_2(n_d - 1) \times m_2(n_d - 1)$, and the work and storage requirements for solving the corresponding system of equations can not be neglected, especially when n_d is large. The Schur complement system contains the coupling between the different subdomains. Hence, solving it may be a nontrivial task to parallelize in a setting where the subdomains are treated by different parallel processes. Below, three different choices of C_i, resulting in different convergence and parallelization properties for the iterative method, are studied.

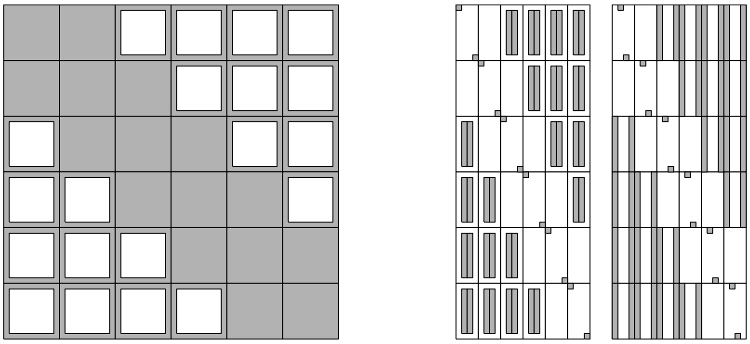

FIG. 4.1. *Structure of C for $n_d = 7$ (left), and U and V in (4.10) (right). The shaded areas represents nonzero elements.*

Method 1. Note that, in the definition of the preconditioner in (4.1), Δ_i corresponds to an "error". In [10], a preconditioner using $\Delta_i \equiv 0$ is studied. In this case, C is explicitly formed by applying M_{11}^{-1} to each column in B_{10}. The basic operation utilized is the parallel subdomain solve described in Section 4.1, but by exploiting the sparsity of B_{10} and B_{01}, the arithmetic complexity can be reduced. The matrix C has a block structure with blocks of size m_2, shown in the left part of Figure 4.1. In the matrix, the block tridiagonal entries are dense, and the first and last rows and columns in every block are also nonzero.

In [10], C is stored as a dense matrix, and standard Gaussian elimination for dense matrices is employed for solving the Schur complement system. Below, a more sophisticated scheme is employed, primarily in order to reduce the memory requirements. Note that C may be written as a blocktridiagonal matrix $\hat{C}$ plus a rank $4(n_d - 1)$ correction,

$$C = \hat{C} + UV^T. \tag{4.10}$$

The structure of the $m_2(n_d - 1) \times 4(n_d - 1)$-matrices U and V is shown in the right part of Figure 4.1. Using the Sherman–Morrison–Woodbury formula, we have

$$(\hat{C} + UV^T)^{-1} = (I - \hat{C}^{-1}US^{-1}V^T)\hat{C}^{-1}, \tag{4.11}$$

where

$$S = I + V^T\hat{C}^{-1}U. \tag{4.12}$$

Here, S is a $4(n_d - 1) \times 4(n_d - 1)$-matrix. By employing (4.11), it is clear that the Schur complement solve $Cy = g$ can be performed as

2.1 Solve $\hat{C}w = g$,
2.2 Solve $Sz = V^Tw$,
2.3 Solve $\hat{C}v = Uz$,
2.4 $y = w - v$.

In the implementation of Method 1, this algorithm is employed. A natural approach for simplifying Method 1 is to ignore the low-rank corrections U and V, and use $\hat{C}$ as an approximation for C. However, numerical experiments show that this does not yield acceptable convergence properties for the iterative method.

Method 2. In Method 1, forming C involves $(n_d-1)m_2$ subdomain solves. This implies that the arithmetic complexity is large. In [11], an alternative approach is considered. A modified matrix $\tilde{M}_{11}$ is employed when forming the Schur complement, yielding $\Delta_i \equiv \Delta = B_{01}(M_{11}^{-1} - \tilde{M}_{11}^{-1})B_{10}$.

In the solver for $\tilde{M}_{11}$, each subdomain is approximated by a collection of rectangular slices of constant depth, and with constant (frozen) coefficients, see Figure 4.2. Within each rectangular slice, the Helmholtz equation is separable, and an analytic solution is available. The different slices are connected via one-dimensional boundary conditions that ignore mode couplings, resulting in a small system of equations that has to be solved for each one of the propagating modes.

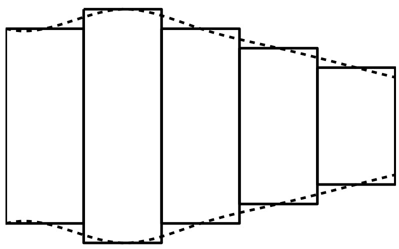

FIG. 4.2. *The boundaries of a subdomain are approximated as piecewise constant. Within each slice, material properties are averaged.*

For large problems, the total work needed for computing C is smaller when $\tilde{M}_{11}$ is used in place of M_{11}. However, since the matrix $\tilde{M}_{11}$ is a less accurate preconditioning matrix, the convergence rate for the iterative method will also be reduced, as shown in Section 7.

The nonzero structure of C is the same as for Method 1, and also here, the Sherman–Morrison–Woodbury formula is used. Note that the approximation $\tilde{M}_{11}$ is only employed for constructing the Schur complement C. To use $\tilde{M}_{11}$ also as subdomain preconditioner in the iterative method does not result in a competitive method.

Method 3. Using Method 1 or 2, the Schur complement, or an approximation of it, is explicitly formed. Then, the corresponding system of equations is solved using a direct solution method. An alternative is to use an iterative method also for solving the Schur complement system, resulting in a two-level iteration. Utilizing this approach, the Schur complement matrix $C = B_{00} - B_{01}M_{11}^{-1}B_{10}$ is never formed, instead the parallel subdomain solve is employed for computing the matrix–vector multiplication $x_0 = Cy_0$. There is no initial work required for forming C, and the work in one inner iteration is essentially equal to one subdomain solve. In the implementation, the standard restarted GMRES method is used for the inner iteration. The initial guess is the zero vector, the stopping criterion is of the same type as (4.4) with tolerance 0.1, and the maximal number of iterations is set to be equal to the restarting length. Note that, because of the error introduced in the inner iteration, Δ_i will in general be different for different outer iterations, implying that here the FGMRES scheme must be employed as the outer iterative method.

In Figure 4.3, the spectrum of the Schur complement matrix C is shown for a problem with four subdomains and frequency $f = 32$ Hz. For this problem, the Schur complement system has 2202 unknowns.

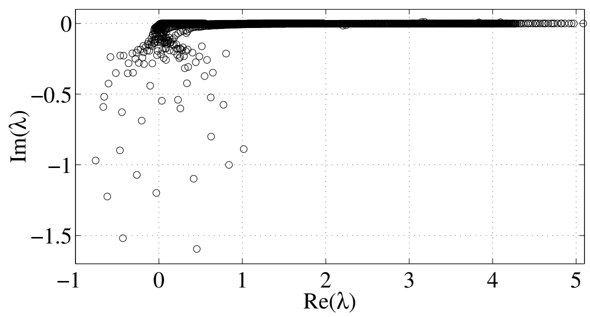

FIG. 4.3. *The eigenvalues of C for a problem where $f = 32$ Hz and $n_d = 4$. The absolute value of the eigenvalue closest to the origin is approximately* 0.0093.

As shown in Figure 4.3, some eigenvalues of C are very small in magnitude. Further computations show that when the frequency or the number of subdomains is increased, eigenvalues move even closer to the origin. This indicates that the convergence rate for the inner iterations will decrease, and this behavior is also verified in numerical experiments. Hence, as long as the maximal number of inner iterations is kept constant, the number of outer iterations will probably grow because the Schur complement system is solved less and less accurately. It is difficult to find an effective preconditioner for the Schur complement system in this setting, since the coefficient matrix C is not explicitly available.

5. Parallel implementation. It is important to recognize that the main objective when writing the parallel code was not to maximize scalability, but to solve as large Helmholtz problems as possible in minimal wall-clock time. If the original system of equations (3.1) is solved using standard Gaussian elimination for dense matrices, the performance will essentially be determined by that of the Linpack benchmark. For a reasonable computer architecture, scalability will then be close to optimal. However, this solution method is impossible to use for large-scale problems because of the prohibitive memory requirements and arithmetic complexity.

The code is written in Fortran 90, and the parallelization is performed using the message passing library MPI, employing n_p MPI processes, where $n_p \leq n_d$. The system of equations is solved using 32-bit arithmetics, whereas the grid generation uses 64-bit computations. The computations are performed on a 32 processor Sun WildFire system* using the Sun Workshop 5.0 F90 compiler and Sun ClusterTools 3.0 MPI.

5.1. Partitioning and data distribution. The resolution criteria used when determining the discretization [13] yields that the number of gridpoints in a subdomain is closely related to the quotient

$$\frac{\max(\text{thickness of subdomain})}{\min(\text{velocity of sound in subdomain})}.$$

*The Sun WildFire [4] system at the Department of Scientific Computing, Uppsala University, consists of two E6000 servers with 16 UltraSparc II processors (250 MHz) and 4 GByte memory each. The servers are connected using the WildFire interconnect, creating a single shared memory of 8 GByte.

The partitioning of the physical layers into n_d computational subdomains is performed so that the maximal difference in this quotient is as small as possible. Since the arithmetic work for a subdomain is approximately proportional to the number of gridpoints, this yields the best load balance possible. Note that, since the layer boundaries normally cannot be moved, there might in some situations be significant load balancing problems.

The parallelization scheme originates from the observation that the subdomain solves in the domain decomposition method are inherently parallel. In essence, all processes are assigned data for at least $\lfloor n_d/n_p \rfloor$ subdomains, and the first $n_d - \lfloor n_d/n_p \rfloor$ processes get data connected to one subdomain more than the other processes.

The guideline when determining the data distribution was to minimize the memory overhead introduced by the parallelization. However, to avoid excessive communication, some replication of data is introduced, resulting in a small amount of memory overhead and some replicated computation. For example, in the matrix–vector multiplication (3.2), the subdomain data in x_1 and y_1 are distributed, whereas the interface data in x_0 and y_0 are replicated in all processes.

5.2. Parallel algorithms. In the GMRES and FGMRES iterations, the dot products require all-reduce communication, otherwise the computations are performed locally.

The matrix B_{11} is distributed over the processes, with the exception of $\mathcal{O}(m_1)$ entries representing the radiation boundary conditions, which are replicated together with the very sparse matrices B_{00}, B_{10}, and B_{01}. The communication required for performing the matrix–vector multiplication (3.2) consists of an all-reduce operation when computing $B_{01}x_1$, and two all-reduce communications for computing the radiation boundary condition part of $B_{11}x_1$.

The representation of the matrix M_{11} in (4.1) is distributed over the processes that form their parts locally. The subdomain solves in Phases 1 and 3 in the preconditioner solve algorithm, as well as the update in Phase 4, are also performed locally, and no communication is required.

The Schur complement system in the preconditioner solve contains the global couplings in the problem, and a natural storage scheme would be to replicate all data involved. This scheme was also employed in [3], where a first version of the parallel algorithm for forming the Schur complement was presented. However, since the size of C grows with n_d, replicating the matrix leads to very poor memory scalability. Also, replication leads to a large amount of communication when forming C.

In the refined parallel scheme used for Methods 1 and 2 here, $\hat{C}$ in (4.10) is partitioned and stored in a block-diagonal format that is distributed over the processes, while the matrices U and V are replicated. Using this storage scheme, only some neighbor-exchange communication is required when forming C. In Figure 5.1, the partitioning of $\hat{C}$ is shown for a problem where $n_d = 6$ and $n_p = 3$.

The LU-factorization of $\hat{C}$ is computed using a column-oriented block algorithm with pivoting. The algorithm is adapted to that $\hat{C}$ is distributed over the processes, and the block size is chosen so that a good balance between the memory requirements for message buffers and the amount of communication startups is achieved. Also, using a block algorithm improves cache utilization. For performing the forward and back substitutions in Phases 2.1 and 2.3 in the Schur complement solve, the same type of block algorithm for distributed data is utilized. Using this type of algorithms implies that the available parallelism is limited. Only at most three processes are simultaneously participating in the work. A scalable algorithm could only be introduced at the

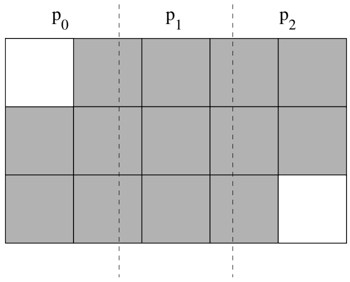

FIG. 5.1. *The storage scheme for $\hat{C}$, $n_d = 6$ and $n_p = 3$. Block column j contains block $(j-1, j)$ in the first block row, block (j, j) in the second block row, and block $(j+1, j)$ in the last block row.*

expense of arithmetic and memory overhead.

When employing Method 3 for the Schur complement solve, all vectors in the inner iterative method are replicated. Each inner iteration will primarily consist of a matrix–vector multiplication,

$$y_0 = Cx_0 = \left(B_{00} - B_{01} M_{11}^{-1} B_{10}\right) x_0. \tag{5.1}$$

These computations are performed locally, and the only communication required is an all-reduce operation for the multiplication with B_{01}.

6. Arithmetic complexities and memory requirements. Most of the arithmetic operations in the algorithms presented above are performed on subdomain data. If $n_p > 1$, these operations are inherently parallel. Let $a(\cdot)_{\mathrm{par}}^{(d)}$ denote the parallelized arithmetic operations connected to subdomain d, while $a(\cdot)_{\mathrm{ser}}$ denotes the arithmetic operations performed serially or on replicated data. A complex addition or subtraction corresponds to two arithmetic operations, a complex multiplication corresponds to six, and a complex division corresponds to nine arithmetic operations. A complex number is assumed to occupy two memory positions.

Ignoring communication and computer architecture effects, but not load imbalance, a crude prediction of the execution time is given by

$$\max_{1 \le p \le n_p} \left(\sum_{d \,\epsilon\, p} a(\cdot)_{\mathrm{par}}^{(d)} \right) + a(\cdot)_{\mathrm{ser}}. \tag{6.1}$$

For describing interface related operations, define

$$w^{(d)} = \begin{cases} 1 & : \quad d = 1 \text{ or } n_d, \\ 2 & : \quad d = 2, \ldots, n_d - 1. \end{cases} \tag{6.2}$$

Also introduce $\mu = (\mu_\ell + \mu_r)/2$ and $\mu_2^{(d)}$ that represents the number of propagating modes used by Method 2 in domain d. Finally, for describing the work for Methods 1 and 2, introduce the variable n_S, defined by

$$n_S = \begin{cases} 0 & : \quad n_d \le 3, \\ 2 & : \quad n_d = 4, \\ n_d - 1 & : \quad n_d \ge 5. \end{cases} \tag{6.3}$$

The arithmetic complexity for solving the system of equations (3.1), using the iterative method described in Section 4, is given by

$$a(\text{init}) + n_{\text{it}}\left(a(B) + a(M^{-1}) + a((\text{F})\text{GMRES})\right). \tag{6.4}$$

Here, $a(\text{init})$ denotes the complexity for setting up the preconditioner solve and starting the GMRES or FGMRES method, n_{it} denotes the number of iterations required for convergence, $a(B)$ denotes the complexity for the matrix–vector multiplication (3.2), $a(M^{-1})$ denotes the complexity for the preconditioner solve (4.2), and $a((\text{F})\text{GMRES})$ denotes the average number of arithmetic operations per iteration for the vector operations in the iterative method.

The initialization phase consists of forming the subdomain preconditioners, setting up for the Schur complement solution, computing the initial guess, and starting the iterative method, i.e.,

$$a(\text{init}) = a(M_{11}) + a(C_*) + a(\text{guess}) + a(\text{itstart}), \tag{6.5}$$

where $* = 1, 2$ or 3, depending on the choice of method for solving the Schur complement system. The arithmetic complexities for the initialization phase are given in Table 6.1.

TABLE 6.1
Arithmetic complexities for the initialization phase.

Operation	$a(\cdot)_{\text{par}}^{(d)}$	$a(\cdot)_{\text{ser}}$
M_{11}	$60m_2m_1^{(d)}\log_2 m_1^{(d)} + 377m_2m_1^{(d)}$	0
C_1	$7(m_1^{(d)})^2 + w^{(d)}((105m_1^{(d)} + 44)m_2^2$ $+(16m_1^{(d)} - 40)m_1^{(d)}m_2)$ $+(n_d - 1)((64\mu + 32)(m_1^{(d)})^2$ $+(210m_2 - 128)m_1^{(d)} + w^{(d)}88m_2)$	$12m_2(n_d - 1)$ $+(20(n_d - 1) + 16m_2)(n_d - 1)/n_p$ $+(n_d - 1)(\frac{32}{3}m_2^3 + 32m_2^2 - 23m_2)$ $-8m_2^3 + n_S((32n_d - 48)m_2^2$ $+(8n_S - 7)(n_d - 1)m_2) + \frac{8}{3}n_S^3$
C_2	$w^{(d)}(m_2 - 10)((162\mu_2^{(d)} + 92)m_2$ $+(16\mu_2^{(d)} + 28611)m_1^{(d)}$ $+4293 - 82\mu_2^{(d)}) + 7(m_1^{(d)})^2$ $+w^{(d)}((1050m_1^{(d)} + 440)m_2$ $+(160m_1^{(d)} - 400)m_1^{(d)})$ $+(n_d - 1)((64\mu + 32)(m_1^{(d)})^2$ $+(210m_2 - 128)m_1 + w^{(d)}88m_2)$	$16\mu m_1(m_2 - 10)(n_d - 1)$ $+(20(n_d - 1) + 16m_2)(n_d - 1)/n_p$ $+(n_d - 1)(\frac{32}{3}m_2^3 + 32m_2^2 - 23m_2)$ $-8m_2^3 + n_S((32n_d - 48)m_2^2$ $+(8n_S - 7)(n_d - 1)m_2) + \frac{8}{3}n_S^3$
C_3	0	0
itstart	$24m_2m_1^{(d)}$	$24m_2(n_d - 1)$

In Table 6.1, the arithmetic work for the factorization of $C_{1,2}$ is considered to be completely serial. This is more pessimistic than necessary, but the maximum parallelism is limited to three, as described in Section 5.

The arithmetic complexity for the preconditioner solve is given by

$$a(M^{-1}) = a(M_{11}^{-1}) + a(C_*^{-1}), \tag{6.6}$$

where $a(M_{11}^{-1})$ denotes the complexity required for Steps 1, 3, and 4 in the preconditioner solve algorithm, and $a(C_*^{-1})$ denotes the number of arithmetic operations for Step 2.

The arithmetic complexities for one iteration are given in Table 6.2. Here, ℓ is the restarting length in the outer iteration method and ℓ_{in} is the restarting length for the inner iterations in Method 3.

TABLE 6.2
Arithmetic complexities for the iterations

Operation	$a(\cdot)^{(d)}_{\text{par}}$	$a(\cdot)_{\text{ser}}$
B	$74m_2m_1^{(d)} + 84m_1^{(d)} + w^{(d)}68m_2$	$(14m_2 + 52\mu n_d + 80)(n_d - 1) + (48\mu + 16)m_1$
M_{11}^{-1}	$80m_2m_1^{(d)} \log_2 m_1^{(d)} + 328m_2m_1^{(d)} + w^{(d)}68m_2$	$2m_2(n_d - 1) + 52\mu(n_d - 1)$
C_1^{-1}	0	$(1 + \frac{n_S}{\max(1,n_S)})((32n_d - 48)m_2^2 - 7(n_d - 1)m_2) + n_S(64m_2(n_d - 1) + 32n_S - 23) + \frac{n_S}{\max(1,n_S)}2m_2(n_d - 1)$
C_2^{-1}	0	$a(C_1^{-1})$
C_3^{-1}	$(1 + \ell_{\text{in}})(40m_2m_1^{(d)} \log_2 m_1^{(d)} + 163m_2m_1^{(d)} + w^{(d)}68m_2)$	$24m_2(n_d - 1) + (1 + \ell_{\text{in}}) ((52\mu n_d + 14m_2 + 80)(n_d - 1) + 16\mu m_1 + 2m_2(n_d - 1)) + \ell_{\text{in}}(8\ell_{\text{in}} + 38)m_2(n_d - 1)$
GMRES	$(8\ell + 38)m_2m_1^{(d)} + \frac{1}{\ell}a(M^{-1})^{(d)}_{\text{par}}$	$(8\ell + 38)m_2(n_d - 1) + \frac{1}{\ell}a(M^{-1})_{\text{ser}}$
FGMRES	$(8\ell + 38)m_2m_1^{(d)}$	$(8\ell + 38)m_2(n_d - 1)$

The memory required for data corresponding to subdomain d is given by $m(\cdot)^{(d)}_{\text{par}}$, and $m(\cdot)_{\text{ser}}$ corresponds to replicated or non-subdomain related data. The total memory requirement is given by

$$m(\cdot) = \sum_{d=1}^{n_d} m(\cdot)^{(d)}_{\text{par}} + n_p m(\cdot)_{\text{ser}}. \tag{6.7}$$

In Table 6.3, the memory requirements for all significant data structures are given. Here, $m(B)$ includes memory for the coefficient matrix, the right-hand side, and the solution. The vectors stored in the iterative methods are represented by $m(\text{GMRES})$ and $m(\text{FGMRES})$, respectively.

7. Numerical simulation of underwater sound propagation. To illustrate the properties of the solution methods, we study two underwater sound propagation problems described in Table 7.1. In the examples, there are four different types of layers; water, soft sediment, hard sediment, and absorbing sediment. The latter type is artificial, representing an absorbing boundary condition at the bottom boundary. This is achieved by letting the absorption δ_1 is increase linearly with the depth, thus introducing a smoothly varying material parameter. The attenuation parameter δ in the wave number is computed as $\delta = \frac{\delta_1}{40\pi \log_{10}(e)}$, and the shape factor is computed as $\max_{x_2} T(x_2)/\min_{x_2} T(x_2)$, where $T(x_2)$ is the thickness of the layer. For simplicity, the density and sound speed have been chosen as constants within the different layers. This does not imply that we solve constant coefficient problems, since the metric

TABLE 6.3
Memory requirements

Data structure	$m(\cdot)_{\text{par}}^{(d)}$	$m(\cdot)_{\text{ser}}$
B	$24m_2m_1^{(d)}$	$22m_2(n_d-1)+4(\mu m_1+m_1+mu)$
M_{11}	$35m_2m_1^{(d)}$	$2m_2(n_d-1)$
C_1	0	$6m_2^2(n_d-1)/n_p+2m_2(n_d-1)(8n_s+1)+32n_s^2$
C_2	0	$6m_2^2(n_d-1)/n_p+2m_2(n_d-1)(8n_s+1)+32n_s^2$
C_3	0	$2m_2(n_d-1)(l_{\text{in}}+1)$
GMRES	$2m_2m_1^{(d)}(\ell+2)$	$2m_2(n_d-1)(\ell+2)$
FGMRES	$2m_2m_1^{(d)}(2\ell+1)$	$2m_2(n_d-1)(2\ell+1)$

TABLE 7.1
Properties of the two problems.

The first problem				
Type of layer	Density kg/m^3	Sound speed m/s	Absorption dB/λ	Shape factor
Water	1000	1500	0	2.3
Hard	2700	4000	0.8	1.6
Absorbing	2700	4000	0.8–10	1.0
The second problem				
Water	1000	1500	0	7.5
Soft	1200	1700	0.8	1.5
Hard	2700	4000	0.8	1.5
Absorbing	2700	4000	0.8–10	1.0

transformation yields smoothly varying coefficients in (2.2). In the computations, the effect of the geometry transformation is not possible to distinguish from the effect of variable material properties.

Typically, for an iterative Helmholtz solver, the number of iterations will grow with frequency. Other factors that will affect the convergence rate is the geometry, the number of layers, and their material properties. The effects of changing the geometry have been investigated in [14] and [10]. A small shape factor results in small variation in the metric coefficients, which is favorable for preconditioning of the type described in this paper. The second problem has larger shape factors, and also more layers with more interaction, than the first problem. Hence, the number of iterations required will probably be larger for the second problem than for the first.

In Figure 7.1, the computed sound intensity for frequency $f = 16$ Hz is displayed for the two problems.

7.1. Complexity results. In Table 7.2, the number of iterations required for Methods 1–3 are shown. Note that, when the frequency is doubled, the number of gridpoints must be increased more than 4 times to fulfill the resolution criteria [13]. For the first problem, the number of unknowns ranges from 1 960 to 787 446, and for the second problem, the range is 2 632 to 6 657 540.

When determining the restarting lengths for the iterative methods, it was first noted that the FGMRES method requires twice the storage of the GMRES method, implying that choosing $\ell_1 = \ell_2 = 2\ell$ and $\ell_3 = \ell$ yields a fair comparison. Then, several

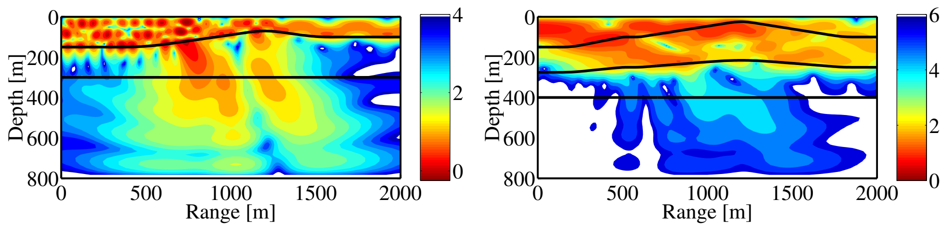

FIG. 7.1. *Solutions for $f = 16$ Hz. The first problem (left) and the second problem (right). The intensity scale is inverse logarithmic.*

numerical experiments were performed, showing that choosing $\ell = 30$ results in close to minimal arithmetic complexity. In the implementation, memory is dynamically allocated for the basis vectors, implying that the memory requirement is reduced if convergence is achieved before the iteration is restarted. For the inner iteration in Method 3, $\ell_{\text{in}} = 100$ is used.

TABLE 7.2
The number of iterations as a function of frequency.

The first problem			
Frequency [Hz]	Method 1	Method 2	Method 3
4	8	24	9
8	8	36	11
16	10	36	14
32	12	43	14
64	25	146	25
The second problem			
4	9	37	11
8	11	83	14
16	11	100	21
32	15	594	36
64	19	>10000	52
128	43	–	–

From Table 7.2, it is clear that the growth of the number of iterations with frequency is modest for Methods 1 and 3. However, for large problems Method 3 needs either a longer outer restarting length or a smaller tolerance for the inner iterations to perform well. Method 2 is sensitive, and not feasible for high frequencies.

Using the arithmetic complexity formulas in Section 6, it is possible to study the efficiency of the methods without introducing effects of the implementation or the computer architecture. Figure 7.2 shows the arithmetic complexity per unknown for the first problem. The solid lines describe the total complexity, whereas the dashed and dash-dot lines show the work for the initialization phase and the iterations, respectively. For comparison, the complexity for standard band Gaussian elimination is also included. Here, the unknowns are assumed to be ordered by a lexicographical single-domain ordering. The initialization phase consists of the LU-factorization, and the substitution is the single iteration required. For this direct solution method, the factorization completely dominates the work, whereas for Method 3 the complexity for the initialization phase is negligible compared with that of the iterations. For Methods 1 and 2, the work is more evenly partitioned between the initialization phase and the iterations, and the relative proportion depends more on the number of unknowns. If no preconditioner is used, there is no convergence in 10 000 iterations, even for $f = 4$ Hz. Hence, a comparison with the unpreconditioned iteration is not feasible.

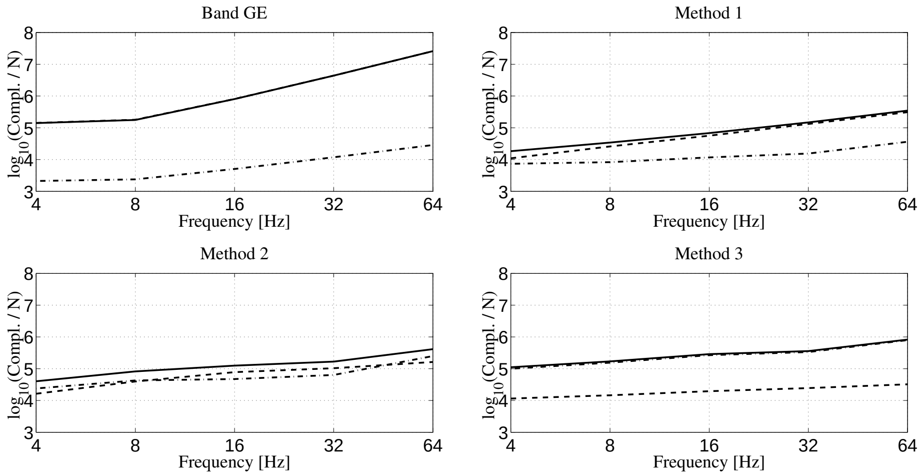

FIG. 7.2. *The arithmetic complexity per unknown for the different methods. The total complexity (solid line), the complexity for the initialization phase (dashed line), and the complexity for the iterations (dash-dot line) is shown.*

In Figure 7.3, the total complexity is shown for both problems. From the figure, it is clear that Method 1 results in the smallest amount of work, but asymptotically if ℓ is increased Method 3 may win.

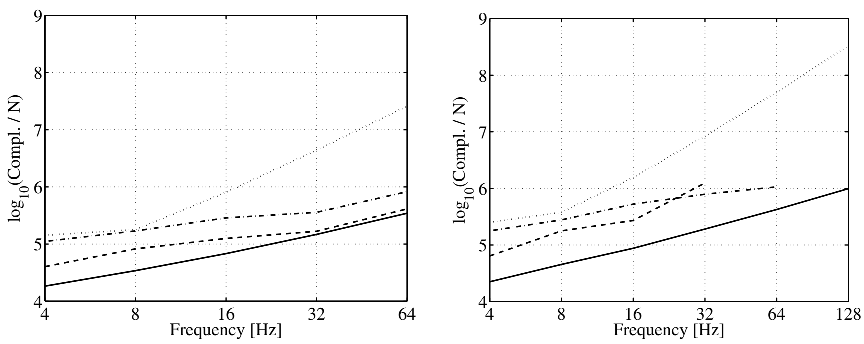

FIG. 7.3. *The arithmetic complexity per unknown for the first problem (left) and the second problem (right). The solid line represents Method* 1*, the dashed line Method* 2*, the dash-dot line Method* 3*, and the dotted line band Gaussian elimination.*

Since we want to solve as large problems as possible, memory requirement is critical. In Table 7.3, the total memory requirement per unknown for the second problem is shown. It is clear that, for all of the iterative methods, the gain compared with band Gaussian elimination is very large and increases with problem size. Also, for large problems, Method 3 uses the smallest amount of memory, since the Schur complement matrix is not stored.

7.2. Parallel computations. First, we consider the second problem for $f = 32$ Hz, solved using different values of $n_p = n_d$. Method 2 is not included in the study,

TABLE 7.3
The memory requirement per unknown.

Frequency [Hz]	4	8	16	32	64	128
Method 1	103	127	135	144	151	196
Method 2	159	225	233	234	233	230
Method 3	116	126	150	183	182	181
Gaussian elimination	1060	1300	2640	6150	15000	38500

since it is not competitive. In Table 7.4, the number of iterations is given for different numbers of subdomains.

Table 7.4, shows that, for Method 1, the number of iterations rather quickly decreases to about 10 when n_d is increased. This type of behavior is expected, since dividing the subdomains leads to less variable coefficients in the subdomain problems, implying that the subdomain preconditioners are more accurate. The Schur complement is treated exactly, and increasing the size of the Schur complement system does not result in deteriorated convergence. In fact, if Method 1 is used, it is easy to show that $M^{-1}B$ has $m_2(n_d-1)$ eigenvalues $\lambda_j \equiv 1$. For Method 3, the situation is different. The iteration count grows when the number of subdomains is increased. This is due to the decreasing accuracy of the solution computed in the inner iteration, as discussed in Section 4.2.

TABLE 7.4
The number of iterations as a function of n_d.

n_d	4	5	6	7	8	9	10
Method 1	15	11	10	9	9	9	10
Method 3	37	16	39	38	53	65	67

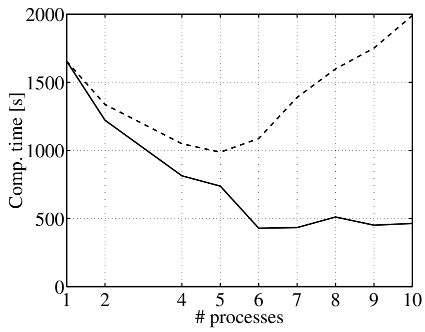

FIG. 7.4. *The computational time for Method 1. Measured (solid line) and predicted (dashed line) values.*

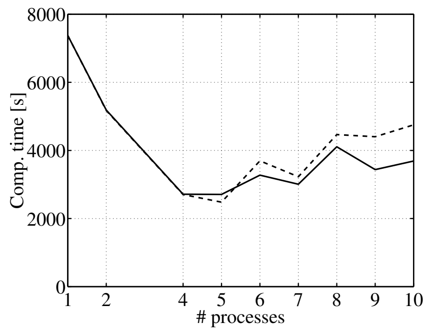

FIG. 7.5. *The computational time for Method 3. Measured (solid line) and predicted (dashed line) values.*

The solid lines in Figures 7.4 and 7.5 show the measured computational times for Method 1 and 3, respectively. Here, $n_d = 4$ when $n_p \leq 4$, otherwise, $n_d = n_p$. For the problem studied here, Method 1 is about four times faster than Method 3 using one process. For Method 3, the number of iterations grows with the number of subdomains. This growth is large enough to cancel the gain from having smaller subdomain problems. For Method 1, the number of iterations does not grow, and the cpu-time only grows slowly when the number of processes is increased.

The dashed lines in Figures 7.4 and 7.5 show the computational times as predicted by (6.1), normalized so that the predicted and measured times coincide for $n_p = 1$. For Method 3, (6.1) predicts the parallelization properties quite well. For large n_d, the measured computational times are a bit smaller than estimated, since the performance for the subdomain solves increases when the size of the subdomain decreases because of improved cache utilization. For Method 1, Figure 7.4 shows that, for large n_d, (6.1) results in a large over-estimation of the computational time. The reason for this is a combination of two effects: For large values of n_d, factorizing the Schur complement system is a time-consuming operation. In (6.1), this is considered to be a serial computation, but experiments show that in reality the speedup is approximately 2. Also, the computational performance of the Schur complement factorization routine is about six times that of the performance in the other routines, again because of improved cache utilization in the block algorithm used for the LU-factorization. Since the Schur complement factorization stands for a relatively small part of the computations for $n_d = 4$, this difference will result in over-estimating the computational time for large values of n_d.

We now study the memory requirements for the parallel implementation. In Figure 7.6, a comparison between the total memory requirement per unknown for the serial and parallel codes is shown. From Figure 7.6, we first note that the memory

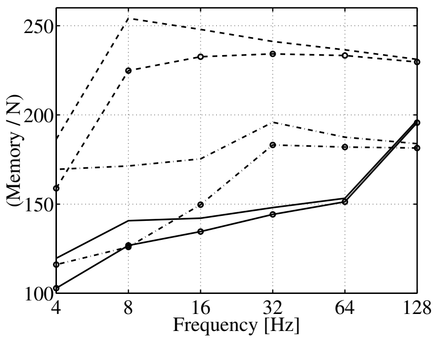

Fig. 7.6. *Total memory requirements for the serial (marked with circles) and parallel codes for Method* 1 *(solid line), Method* 2 *(dashed line), and Method* 3 *(dash-dot line).*

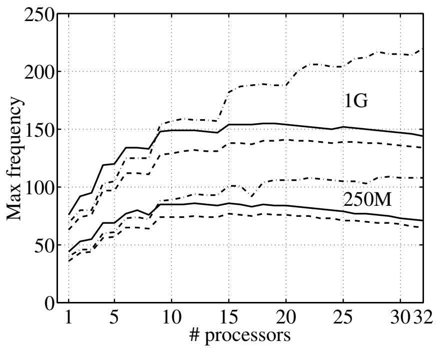

Fig. 7.7. *The highest frequency for which a solution can be computed as a function of the number of processes. Two cases with different sizes of the local memory in each processing element are shown for Method* 1 *(solid line) , Method* 2 *(dashed line), and Method* 3 *(dash-dot line).*

overhead from the parallelization is negligible for large problem sizes. If the code is run on a multiprocessor computer with a single, shared memory, this implies that we do not lose the ability of solving large problems by parallelizing the computations. If the code is run on a computer where the processing elements consist of processors and local memory, we get access to more memory by using more processors. For this situation, Figure 7.7 shows the maximal frequency for which a solution could be computed using $n_p = 1, \ldots, 32$. Two cases are presented, in the first case each processor is assumed to have 250 MBytes of memory and in the second case, the local memory in each processor is 1 GByte.

From Figure 7.7, it is clear that, for Methods 1 and 2, there is nothing to be gained by increasing n_d above a certain number. The reason is that, even if the frequency is fixed, the size of the Schur complement matrix C increases when we increase n_d. Each process stores a part of C that is essentially independent of n_d, which will dominate the memory requirement. Method 3 does not suffer from this drawback, since C is not stored.

The plateaus on the curves in Figure 7.7, especially for Method 3, are results of load imbalance. The size of the largest subdomain is not significantly decreased until all physical layers have been partitioned at least one more time.

8. Conclusions. The motivation for the work presented here is to solve as large Helmholtz problems as possible, in the shortest possible time. Three preconditioned iterative methods, Methods 1–3, have been presented. All three methods show a large gain both in arithmetic complexity and memory requirement compared with a standard direct solution method. The gain increases with problem size.

On a computer architecture with processing elements consisting of local memory/processor nodes, the parallelization of the code enables us to solve very large problems with Method 3 using many processors. Method 1 shows very good performance on a small number of processors, but the scalability is limited. Method 2 is probably too sensitive to be of any practical use.

Method 1 scales best in computational time when the number of subdomains increases. The convergence rate is improved, whereas for the other methods it decreases. On a shared memory computer, Method 1 or Method 3 using a rather small number of processors is the best solution method.

REFERENCES

[1] A. Bayliss, C. I. Goldstein, and E. Turkel, *On accuracy conditions for the numerical computation of waves*, J. Comput. Phys., 59 (1985), pp. 396–404.

[2] D. K. Cheng, *Field and Wave Electromagnetics*, Addison–Wesley, Reading, MA, 2nd ed., 1989.

[3] R. Escovar, *Parallelization of the creation of a Schur complement matrix*, M.Sc. thesis, Internal Report, Dept. of Scientific Computing, Uppsala Univ., Uppsala, Sweden, 1999.

[4] E. Hagersten and M. Koster, *WildFire: a scalable path for SMPs*, in Proc. 5th International Symposium on High Performance Computer Architecture, 1999.

[5] E. Heikkola, Y. A. Kuznetsov, P. Neittaanmäki, and J. Toivanen, *Fictitious domain methods for the numerical solution of two-dimensional scattering problems*, J. Comput. Phys., 145 (1998), pp. 89–109.

[6] T. Huckle, *Fast transforms for tridiagonal linear equations*, BIT, 34 (1994), pp. 99–112.

[7] F. Ihlenburg and I. Babuška, *Finite element solution of the Helmholtz equation with high wave number Part II: The h-p version of the FEM*, SIAM J. Numer. Anal., 34 (1997), pp. 315–358.

[8] F. B. Jensen, W. A. Kuperman, M. B. Porter, and H. Schmidt, *Computational Ocean Acoustics*, AIP Press, New York, 1994.

[9] J. B. Keller and D. Givoli, *Exact non-reflecting boundary conditions*, J. Comput. Phys., 82 (1989), pp. 172–192.

[10] E. Larsson, *A domain decomposition method for the Helmholtz equation in a multilayer domain*, SIAM J. Sci. Comput., 20 (1999), pp. 1713–1731.

[11] E. Larsson and L. Abrahamsson, *Helmholtz and PE solutions to a benchmark problem in ocean acoustics*, Report 210, Dept. of Scientific Computing, Uppsala Univ., Uppsala, Sweden, 1998.

[12] K. Otto, *A unifying framework for preconditioners based on fast transforms*, Report 187, Dept. of Scientific Computing, Uppsala Univ., Uppsala, Sweden, 1996.

[13] ———, *Iterative solution of the Helmholtz equation by a fourth-order method*, Boll. Geof. Teor. Appl., 40 suppl. (1999), pp. 104–105.

[14] K. Otto and E. Larsson, *Iterative solution of the Helmholtz equation by a second-order method*, SIAM J. Matrix Anal. Appl., 21 (1999), pp. 209–229.

[15] A. F. Peterson, S. L. Ray, and R. Mittra, *Computational Methods for Electromagnetics*, IEEE Press, New York, 1998.

[16] Y. Saad, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM J. Sci. Comput., 14 (1993), pp. 461–469.

[17] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[18] L. L. Thompson and P. M. Pinsky, *Complex wavenumber Fourier analysis of the p-version finite element method*, Comput. Mech., 13 (1994), pp. 255–275.