A parallel implementation of a high-order accurate solution technique for variable coefficient Helmholtz problems

Natalie N. Beams\textsuperscript{a}, Adrianna Gillman\textsuperscript{a,\,*}, Russell J. Hewett\textsuperscript{b,c}

\textsuperscript{a}Department of Computational and Applied Mathematics, Rice University
\textsuperscript{b}Total E&P Research & Technology USA
\textsuperscript{c}Department of Mathematics, Virginia Tech

Abstract

The recently developed Hierarchical Poincaré-Steklov (HPS) method is a high-order discretization technique that comes with a direct solver. Results from previous papers demonstrate the method’s ability to solve Helmholtz problems to high accuracy without the so-called pollution effect. While the asymptotic scaling of the direct solver’s computational cost is the same as the nested dissection method, serial implementations of the solution technique are not practical for large scale numerical simulations. This manuscript presents the first parallel implementation of the HPS method. Specifically, we introduce an approach for a shared memory implementation of the solution technique utilizing parallel linear algebra. This approach is the foundation for future large scale simulations on supercomputers and clusters with large memory nodes. Performance results on a desktop computer (resembling a large memory node) are presented.

Keywords: Numerical partial differential equations, Direct solver, High-order discretization, Nested dissection, OpenMP, shared-memory parallelization, MKL

1. Introduction

Consider the variable coefficient Helmholtz problem

\begin{equation}
\begin{cases}
-\Delta u(x) - \kappa^2 c(x) u(x) = s(x) & x \in \Omega, \\
\frac{\partial u}{\partial \nu} + i\eta u = t(x) & x \in \Gamma = \partial \Omega,
\end{cases}
\end{equation}

where $\Omega$ is a rectangle in $\mathbb{R}^2$, $\kappa$ is the wave number, $\nu$ is the outward facing normal on $\Gamma$, $\eta \in \mathbb{C}$ and $u(x)$ is the unknown solution. The functions $s(x)$, $t(x)$,
and $c(x)$ are assumed to be smooth. Solutions to this boundary value problem are oscillatory and the frequency at which the solutions oscillate is dictated by $\kappa$. In other words, as $\kappa$ grows, the solution becomes more oscillatory. The task of creating high-order approximate solutions to boundary value problems of the form [1], where the number of discretization points per wavelength is fixed has been a challenge for some time. The recently developed Hierarchical Poincaré-Steklov (HPS) method is a high-order discretization technique that comes with an efficient direct solver which does not, in numerical experiments, suffer from the so-called pollution effect [1]. For the HPS method to be useful for large scale computations, a high performance computing implementation of the method needs to be developed. This paper presents the first such implementation. The implementation is for a shared memory machine that is representative of the large memory nodes in upcoming supercomputers and clusters.

### 1.1. Overview of discretization technique

Roughly speaking, the discretization technique and construction of the direct solver can be broken into three steps:

**Step 1:** The geometry is partitioned into a collection of disjoint patches sized so that a boundary value problem on the patch can be solved to high accuracy via a classic spectral collocation method (e.g. [2]).

**Step 2:** Each patch is discretized using a high order spectral collocation technique. Approximate boundary (Poincaré-Steklov) and solution operators are constructed.

**Step 3:** In a hierarchical fashion, the patches are “glued” together two at a time by by enforcing a continuity conditions on the solution though the Poincaré-Steklov operators at the boundaries of each patch. For each merged patch, corresponding boundary and solution operators are constructed.

These three steps comprise the precomputation stage of the solution technique. Once the precomputation is complete, the task of finding the solution to [1] for any choice of body load $s(x)$ and boundary data $t(x)$ is simply a collection of small matrix vector multiplies involving the precomputed operators. This is called the solve stage.

The domain decomposing nature of the algorithm provides significant opportunities for parallelism. For two dimensional problems, the required dense linear algebra involves matrices corresponding to one dimensional interfaces, which keeps the computational cost in flops of the algorithm down. The distribution of the work while moving through the hierarchical tree in both stages of the algorithm are explored in this paper.

While the method can be employed with any Poincaré-Steklov operator, this paper uses the impedance-to-impedance ($\text{ItI}$) operator, which is ideal for Helmholtz problems. For general elliptic problems, the Dirichlet-to-Neumann operator is a suitable choice [3][4][5].
1.2. Related to prior work

The HPS method has evolved since it was first developed. The original method \cite{4} was designed for elliptic partial differential equations and the local discretization utilized classic spectral collocation techniques, which involved points at the corners of leaf boxes. These corner discretization points were not ideal for the "gluing" procedure in Step 2. In \cite{3, 5, 1} corner points were removed by using interpolation operators to represent the boundary operators only on edges of boxes. Most recently, in \cite{6}, a new spectral collocation scheme is presented which does not place any discretization points the corners of boxes. The parallelization of this latest version of the method is presented in this paper.

The direct solver for the HPS discretization is related to the direct solvers for sparse systems arising from finite difference and finite element discretizations of elliptic PDEs, such as the classical nested dissection method of George \cite{7} and the multifrontal methods by Duff et al. \cite{8}. These methods can be viewed as a hierarchical version of the “static condensation” idea in finite element analysis \cite{9}. High-order finite difference and finite element discretizations lead to large frontal matrices, and consequently very high cost of the LU-factorization (see, e.g., Table 2 in \cite{3}). It has been demonstrated that the dense matrices that arise in these solvers have internal structure that allows the direct solver to be accelerated to linear or close to linear complexity, see, e.g., \cite{10, 11, 12, 13, 14}. The two dimensional HPS solution technique has one dimensional “dividers” independent of order and thus the direct solver only pays (in terms of computational complexity) the price of the high-order discretization at the lowest level in the hierarchical tree. The same ideas that accelerate the nested dissection and multifrontal solvers can be applied the HPS direct solver \cite{3}.

There are multiple widely-available libraries for high performance parallel implementations of direct factorization for sparse matrices. SuperLU \cite{15, 16, 17} takes either a left-looking (shared memory) or right-looking (distributed memory) approach to factorization. To minimize idle time when the number of independent tasks is less than the number of available processors, SuperLU implements pipelining, where portions of dependent tasks are computed simultaneously and waiting only occurs when a task cannot continue without receiving necessary data from a related task. The multifrontal solvers in WSMP \cite{18} and MUMPS \cite{19, 20} both employ multiple strategies for parallelism based on the hierarchical nature of the multifrontal algorithm. First, there is parallelism from the natural independent calculation of subtrees that do not depend on each other. As the elimination continues and the number of independent calculations is greater than the number of processes, the processes share the calculations through parallel linear algebra. This tailoring of parallelism to the algorithm’s tree structure is, in essence, our approach as well. Given the success of the approach in WSMP and MUMPS and the similarity of the multifrontal or nested dissection algorithm to the build stage of the HPS algorithm, we should expect the concept to be successful in accelerating the HPS solver as well.
1.3. Outline

This paper begins with a description of the HPS method in section 2. Techniques for finding the optimal shared-memory parallelization are presented in section 3. Section 4 illustrates the results of the optimization procedure and the speedup obtained when the HPS method is implemented on a desktop computer. Finally, the paper closes with remarks and future directions in section 5.

2. The HPS method

This section reviews the HPS method for solving the boundary value problem (1). The solution technique begins by partitioning the domain $\Omega$ into a collection of square (or possibly rectangular) boxes, called leaf boxes. Throughout this paper, we assume that the parameter for the order of the discretization $n_c$ is fixed ($n_c = 16$ is often a good choice). For a uniform discretization, the size of all leaf boxes is chosen so that any potential solution $u$ of equation (1), as well as its first and second derivatives, can be accurately interpolated from their values at the local discretization points on any leaf box.

Next, a binary tree on the collection of leaf boxes is constructed by hierarchically merging them. All boxes on the same level of the tree are roughly of the same size, cf. Figure 1. The boxes should be ordered so that if $\tau$ is a parent of a box $\sigma$, then $\tau < \sigma$. We also assume that the root of the tree (i.e. the full box $\Omega$) has index $\tau = 1$. We let $\Omega^{\tau}$ denote the domain associated with box $\tau$.

Recall, from section 1.1, that the solution technique is comprised of a precomputation stage and a solve stage. The precomputation stage discretizes the partial differential equation and builds a direct solver. The solve stage uses the precomputed direct solver information applied to body load $s(x)$ and boundary condition $t(x)$ information to construct an approximate solution the partial differential equation. The two major computational components of these stages involve leaf and merging computations.

The key to merging boxes is a Poincaré-Steklov operator. For variable coefficient Helmholtz problems such as (1), the impedance-to-impedance (ItI) operator is used. The ItI operator is defined as follows:

**Definition 1 (impedance-to-impedance map).** Fix $\eta \in \mathbb{C}$, and $R|\eta \neq 0$. Let

\begin{align*}
(2) \quad f & := u_n + i\eta u|_\Gamma \\
(3) \quad g & := u_n - i\eta u|_\Gamma
\end{align*}

be Robin traces of $u$. We refer to $f$ and $g$ as the “incoming” and “outgoing” (respectively) impedance data. For any $\omega > 0$, the ItI operator $R : L^2(\Gamma) \to L^2(\Gamma)$ is defined by

\begin{equation}
Rf = g
\end{equation}

for $f$ and $g$ the Robin traces of $u$ the solution of (1), for all $f \in L^2(\Gamma)$. 

When \( s(x) \) in (1) is nonzero, it is advantageous to write the solution \( u(x) \) as the superposition of the homogeneous solution \( \bar{u} \) and the particular solution \( \tilde{u} \); i.e. \( u = \bar{u} + \tilde{u} \) where \( \tilde{u} \) is the solution of

\[
-\Delta \tilde{u} - \kappa^2 c(x)\tilde{u} = s(x) \quad x \in \Omega
\]
\[
\frac{\partial \tilde{u}}{\partial \nu} + i\eta \tilde{u} = 0 \quad x \in \Gamma
\]

and \( \bar{u} \) is the solution of

\[
-\Delta \bar{u} - \kappa^2 c(x)\bar{u} = 0 \quad x \in \Omega
\]
\[
\frac{\partial \bar{u}}{\partial \nu} + i\eta \bar{u} = t(x) \quad x \in \Gamma.
\]

The remainder of the section presents the technique for discretizing leaf boxes (section 2.1) and merging neighboring boxes (section 2.2) via this superposition form. Specifically, a collection of approximate solution, impedance, and \( \text{ItI} \) operators \( R \) are constructed for each box. By constructing the operators via the superposition, they are independent of the body load \( s(x) \) and boundary condition \( t(x) \). An approximate solution can then be constructed for any body load \( s(x) \) and boundary condition \( t(x) \) by sweeping the tree twice. First, particular solution information is constructed, moving from the leaf boxes up the hierarchical tree. The approximation solution is then created by propagating homogeneous boundary information down the tree. Algorithm 2 presents the details of this procedure. When there is no body load (i.e. \( s(x) = 0 \)), the solution procedure needs only the downward sweep of the tree. The homogeneous solver is the same algorithm as presented in [1].

2.1. Leaf computation

This section presents a modified spectral collocation method for constructing the necessary operators for processing a leaf box \( \tau \). The modified spectral
collocation technique, first presented in [6], is ideal for the HPS method because it does not involve corner discretization points, for which Poincaré-Steklov operators are not always well defined.

The modified spectral collocation technique begins with the classic $n_c \times n_c$ product Chebychev grid and the corresponding differential matrices $D_x$ and $D_y$, as defined in [2]. Let $I^r_\tau$ denote the index vector corresponding to points on the interior of $\Omega^r$ and $I^b_\tau$ denote the index vector corresponding to points on the boundary of $\Omega^r$, not including the corner points, based on the classic tensor grid. Figure 2 illustrates the indexing of the points in terms of the classic discretization. Thus $\{x_j\}_{j=1}^{n^2_{c}-4}$ denotes the discretization points in $\Omega^r$ given by the union of the red and blue points in Figure 2. We order the solution vector $u$ according to the following: $u = \begin{bmatrix} u_b & u_i \end{bmatrix}$ where $u_b$ and $u_i$ denote the approximate values of the solution on the boundary and the interior, respectively. The homogeneous and particular solution vectors $\tilde{u}$ and $\bar{u}$ are ordered in a similar manner. The ordering of the entries related to the boundary corresponds to the interaction of the corner points with the points on the interior of $\Omega^r$ are zero. The directional basis functions for the other points on the boundary are not impacted by the removal of the corner points. Thus the differential operators from the classic pseudospectral discretization can be used to create the approximation of the local differential operator, the ItI operator, and all other necessary leaf operators.

The classic discrete approximation of the differential operator on $\Omega^r$ is given by

$$A = -D_x^2 - D_y^2 - C,$$

where $C$ is the diagonal matrix with diagonal entries $\{\kappa^2_c(x_k)\}_{k=1}^{n^2_{c}}$.

Likewise, operators can be constructed to approximate impedance operators. Let $N$ denote the matrix that takes normal derivatives of the basis functions. Then $N$ is given by

$$N = \begin{bmatrix} -D_x(I_s, I^r) & D_y(I_c, I^r) \\ D_x(I_n, I^r) & -D_y(I_w, I^r) \end{bmatrix}.$$  

The matrix for creating the incoming impedance data is

$$F = N + i\eta I_{n^2_{c}}(I_b, I^r)$$

and the matrix for creating the outgoing impedance data is

$$G = N - i\eta I_{n^2_{c}}(I_b, I^r)$$

where $I_{n^2_{c}}$ is the identity matrix of size $n^2_{c}$.
2.1.1. Homogeneous solution operators

To construct the homogeneous solution operators, we consider the discretized differential equation defined on $\Omega^\tau$. The discretized body load problem to find the approximation to $\bar{\mathbf{u}}$ at the collocation points takes the form

$$
B \begin{bmatrix} \bar{\mathbf{u}}_b \\ \bar{\mathbf{u}}_i \end{bmatrix} = \begin{bmatrix} \mathbf{F} \\ \mathbf{A}(i,b) \mathbf{A}(i,i) \end{bmatrix} \begin{bmatrix} \bar{\mathbf{u}}_b \\ \bar{\mathbf{u}}_i \end{bmatrix} = \begin{bmatrix} \mathbf{t} \\ \mathbf{0} \end{bmatrix}
$$

where $\bar{\mathbf{u}}$ is the vector with the approximate homogeneous solution at the collocation points, $\mathbf{A}_{i,i} = \mathbf{A}(I_{\tau}^i, I_{\tau}^i)$ is a matrix of size $(n_c - 2)^2 \times (n_c - 2)^2$, $\mathbf{A}_{i,b} = \mathbf{A}(I_{\tau}^i, I_{\tau}^b)$ is a matrix of size $(n_c - 2)^2 \times (4n_c - 8)$, and $\mathbf{t}$ is vector of length $4n_c - 8$ containing impedance boundary data.

The homogeneous solution operator $\Psi^\tau$ is defined by solving

$$
B \Psi^\tau = \begin{bmatrix} \mathbf{0}_{(n_c - 2)^2 \times 4n_c - 4} \\ \mathbf{I}_{4n_c - 4} \end{bmatrix}.
$$

To construct the approximate $HI$ operator $\mathbf{R}^\tau$, we simply need to apply $\mathbf{G}$ to $\Psi^\tau$; that is

$$
\mathbf{R}^\tau = \mathbf{G}\Psi^\tau.
$$

2.1.2. Particular solution operators

The particular solution operators are constructed in a similar manner. Specifically, the discretized version of (5) takes the form

$$
B \begin{bmatrix} \tilde{\mathbf{u}}_b \\ \tilde{\mathbf{u}}_i \end{bmatrix} = \begin{bmatrix} \mathbf{F} \\ \mathbf{A}(I_{\tau}^i, I_{\tau}^i) \mathbf{A}(I_{\tau}^i, I_{\tau}^b) \end{bmatrix} \tilde{\mathbf{u}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{s} \end{bmatrix}
$$

where $\mathbf{s}$ is a vector of length $(n_c - 2)^2$ of body load data.

Then the particular solution operator $\mathbf{Y}^\tau$ which can be used to approximate the particular solution $\tilde{\mathbf{u}}$ on the leaf $\tau$ when applied to any body load vector $\mathbf{s}$ is the solution of

$$
B \mathbf{Y}^\tau = \begin{bmatrix} \mathbf{0}_{1n_c - 4 \times (n_c - 2)^2} \\ \mathbf{I}_{(n_c - 2)^2} \end{bmatrix}.
$$

As with the homogeneous case, the operator constructing the approximation of the outgoing impedance data is constructed by applying the operator $\mathbf{G}$ to the particular solution operator $\mathbf{Y}^\tau$. Let

$$
\Gamma^\tau = \mathbf{G}\mathbf{Y}^\tau
$$

denote this particular solution outgoing impedance operator.

Remark 1. Once all the leaf operators are constructed for a box $\tau$, the solution vector $\mathbf{u}^\tau$ is given by

$$
\mathbf{u}^\tau = \Psi^\tau \mathbf{t} + \mathbf{Y}^\tau \mathbf{s} = \bar{\mathbf{u}}^\tau + \tilde{\mathbf{u}}^\tau$$
where $t$ is a vector whose entries are impedance boundary data at the boundary nodes on $\tau$ and $s$ is a vector whose entries are the evaluation of the body load $s(x)$ at the interior discretization points of $\tau$. The outgoing impedance data is given by

$$g^\tau = R^\tau t + \Gamma^\tau s = R^\tau t + h^\tau$$

where $h^\tau$ is the particular outgoing impedance boundary data.

**Figure 2**: Illustration of the discretization points for a leaf box $\tau$. The points in blue are the boundary points with indices $I^\tau = [I_s, I_e, I_n, I_w]$. The points in red are the interior points with indices $I^\iota$. The points in black are the omitted corner points.

### 2.2. Merging two boxes

This section presents the technique for constructing the necessary operators for the union of two boxes for which outgoing particular solution information and $\text{ItI}$ operators have already been constructed.

Let $\Omega^\tau$ denote a box with children $\Omega^\alpha$ and $\Omega^\beta$ so that

$$\Omega^\tau = \Omega^\alpha \cup \Omega^\beta.$$  

For concreteness, but without loss of generality, let us assume that $\Omega^\alpha$ and $\Omega^\beta$ share a vertical edge as shown in Figure 3. We partition the points on $\partial \Omega^\alpha$ and $\partial \Omega^\beta$ into three sets:

- $I_1$: Boundary nodes of $\Omega^\alpha$ that are not boundary nodes of $\Omega^\beta$.
- $I_2$: Boundary nodes of $\Omega^\beta$ that are not boundary nodes of $\Omega^\alpha$.
- $I_3$: Boundary nodes of both $\Omega^\alpha$ and $\Omega^\beta$ that are *not* boundary nodes of the union box $\Omega^\tau$.

The indexing for the points on the interior and boundary of $\Omega^\tau$ are $I^\iota = I_3$ and $I^{\tau}_b = [I_1, I_2]$, respectively.
For the box $\alpha$, let $t^\alpha$ denote the \textit{homogeneous solution} incoming impedance boundary data, $h^\alpha$ denote the \textit{particular solution} outgoing impedance boundary data, and $g^\alpha$ denote the \textit{total} outgoing impedance boundary data. Define the vectors $t^\beta$, $h^\beta$, and $g^\beta$ similarly.

Using the $\operatorname{ItI}$ operators $R^\alpha$ and $R^\beta$ and ordering according everything according to the boundary numbering in Figure 3, the outgoing impedance data for boxes $\alpha$ and $\beta$ are given by

\begin{equation}
\begin{bmatrix}
g_1^\alpha \\
g_3^\alpha \\
\end{bmatrix}
= 
\begin{bmatrix}
R_{11}^\alpha & R_{13}^\alpha \\
R_{31}^\alpha & R_{33}^\alpha \\
\end{bmatrix}
\begin{bmatrix}
t_1^\alpha \\
t_3^\alpha \\
\end{bmatrix}
+ 
\begin{bmatrix}
h_1^\alpha \\
h_3^\alpha \\
\end{bmatrix}
\tag{11}
\end{equation}

and

\begin{equation}
\begin{bmatrix}
g_2^\beta \\
g_3^\beta \\
\end{bmatrix}
= 
\begin{bmatrix}
R_{22}^\beta & R_{23}^\beta \\
R_{32}^\beta & R_{33}^\beta \\
\end{bmatrix}
\begin{bmatrix}
t_2^\beta \\
t_3^\beta \\
\end{bmatrix}
+ 
\begin{bmatrix}
h_2^\beta \\
h_3^\beta \\
\end{bmatrix}
\tag{12}
\end{equation}

Since the normal vectors are opposite in each box, we know $t_3^\alpha = -g_3^\beta$ and $g_3^\beta = -t_3^\alpha$. Using this information in the bottom row equations in (11) and (12), $t_3^\alpha$ and $t_3^\beta$ can found in terms of $t_1^\alpha$, $t_2^\beta$, $h_3^\alpha$, and $h_3^\beta$. They are given by

\begin{equation}
t_3^\alpha = \Phi^\alpha 
\begin{bmatrix}
t_1^\alpha \\
t_2^\beta \\
\end{bmatrix}
+ \Upsilon^\alpha 
\begin{bmatrix}
h_3^\alpha \\
h_3^\beta \\
\end{bmatrix}
\tag{13}
\end{equation}

and

\begin{equation}
t_3^\beta = \Phi^\beta 
\begin{bmatrix}
t_1^\alpha \\
t_2^\beta \\
\end{bmatrix}
+ \Upsilon^\beta 
\begin{bmatrix}
h_3^\alpha \\
h_3^\beta \\
\end{bmatrix}
\tag{14}
\end{equation}
where
\[
\Phi^\alpha = W^{-1} \left[ R_{31}^\beta R_{31}^\alpha - R_{32}^\beta \right], \\
\Upsilon^\alpha = \left[ W^{-1} R_{33}^\beta \ | - W^{-1} \right], \\
\Phi^\beta = \left[ -R_{31}^\alpha - R_{34}^\beta W^{-1} R_{33}^\beta R_{31}^\alpha | R_{33}^\beta W^{-1} R_{32}^\beta \right], \\
\Upsilon^\beta = \left[ (I + R_{31}^\alpha W^{-1} R_{33}^\beta) | R_{33}^\beta W^{-1} \right], \text{ and} \\
W = I - R_{33}^\beta R_{33}^\alpha.
\]

Substituting (13) and (14) into the top row equations of (11) and (12) results in the following expression for the outgoing impedance data for the box \( \Omega^\alpha \cup \Omega^\beta \)

\[
\begin{bmatrix}
g_\alpha^1 \\
g_\beta^2
\end{bmatrix}
= R^\tau \begin{bmatrix}
t_\alpha^1 \\
t_\beta^2
\end{bmatrix} + \begin{bmatrix}
h_\alpha^1 \\
h_\beta^2
\end{bmatrix} + \Gamma^\tau \begin{bmatrix}
h_\alpha^3 \\
h_\beta^3
\end{bmatrix}
\]

where
\[
R^\tau = \begin{bmatrix}
R_{11}^\alpha & 0 \\
0 & R_{22}^\beta
\end{bmatrix} + \begin{bmatrix}
R_{31}^\alpha & R_{32}^\beta \\
R_{33}^\beta & R_{33}^\beta
\end{bmatrix} \begin{bmatrix}
\Phi^\alpha \\
\Phi^\beta
\end{bmatrix}
\]
is the homogeneous ItI operator and
\[
\Gamma^\tau = \begin{bmatrix}
R_{31}^\alpha & R_{32}^\beta \\
R_{33}^\beta & R_{33}^\beta
\end{bmatrix} \begin{bmatrix}
\Upsilon^\alpha \\
\Upsilon^\beta
\end{bmatrix}
\]
is the outgoing particular solution flux due to interior edge operator. The outgoing particular solution flux is

\[
\begin{bmatrix}
h_\alpha^1 \\
h_\beta^2
\end{bmatrix}
= R^\tau \begin{bmatrix}
t_\alpha^1 \\
t_\beta^2
\end{bmatrix} + \Gamma^\tau \begin{bmatrix}
h_\alpha^3 \\
h_\beta^3
\end{bmatrix}
\]

2.3. Computational cost

The cost of constructing the discretization and direct solver is dominated by inverting the matrix \( W \) of size \( \sim N^{1/2} \) at the top level in the tree, where \( N \) is the number of discretization points. Thus the computational cost of the precomputation stage is \( O(N^{3/2}) \). At any level in the solve stage, the cost of applying all the operators is \( O(N) \), and there are \( \log N \) levels in a uniform tree. It follows that the total cost of the apply the solver is \( O(N \log N) \) with a small constant.

3. General thread optimization technique

This section presents the proposed optimization technique for shared-memory parallelism via OpenMP and the Intel MKL library. Recall from section 2 the bulk of the computational cost in both stages of the algorithm is associated
This algorithm builds all the operator needed to construct an approximate solution to (1) for any choice of body load $s(x)$ and incoming impedance boundary condition $t(x)$.

It is assumed that if node $\tau$ is a parent of node $\sigma$, then $\tau < \sigma$.

Let $N_{\text{boxes}}$ denote the number of boxes in the tree.

\begin{algorithm}
\begin{algorithmic}
\State (Precomputation stage)
\State
\State This algorithm builds all the operator needed to construct an approximate solution to (1) for any choice of body load $s(x)$ and incoming impedance boundary condition $t(x)$.
\State It is assumed that if node $\tau$ is a parent of node $\sigma$, then $\tau < \sigma$.
\State Let $N_{\text{boxes}}$ denote the number of boxes in the tree.

\For{$\tau = N_{\text{boxes}}, N_{\text{boxes}} - 1, N_{\text{boxes}} - 2, \ldots, 1$}
\If{$(\tau$ is a leaf$)$}
\State Construct $R^{\tau}, Y^{\tau}, \Psi^{\tau}$ and $\Gamma^{\tau}$ via the process described in Section 2.1.
\Else
\State Let $\sigma_1$ and $\sigma_2$ be the children of $\tau$.
\State Split $I_6^{\sigma_1}$ and $I_6^{\sigma_2}$ into vectors $I_1, I_2,$ and $I_3$ as shown in Figure 3.
\State $W = (I - R_{33}^{\sigma_1} R_{33}^{\sigma_2})^{-1}$
\State $\Phi^{\sigma_1} = W^{-1} [R_{33}^{\sigma_1} R_{31}^{\sigma_1} - R_{33}^{\sigma_2}]$
\State $\Phi^{\sigma_2} = -[R_{33}^{\sigma_1} - R_{33}^{\sigma_2} W^{-1} R_{33}^{\sigma_2} R_{31}^{\sigma_1} R_{33}^{\sigma_1} W^{-1} R_{33}^{\sigma_2}]$
\State $\Phi^{\sigma_2} = -[I + R_{33}^{\sigma_1} W^{-1} R_{33}^{\sigma_2} R_{33}^{\sigma_1} W R_{33}^{\sigma_2} R_{33}^{\sigma_1} W^{-1} R_{33}^{\sigma_2}]$
\State $R^{\tau} = \begin{bmatrix} R_{11}^{\sigma_1} & 0 \\ R_{13}^{\sigma_1} & R_{22}^{\sigma_2} \end{bmatrix}$
\State $\Gamma^{\tau} = \begin{bmatrix} \Phi^{\sigma_1} \\ \Phi^{\sigma_2} \end{bmatrix}$
\State Delete $R^{\sigma_1}$ and $R^{\sigma_2}$.
\EndIf
\EndFor
\end{algorithmic}
\end{algorithm}

with matrix inversion and matrix-matrix multiplication. Since the algorithm is domain decomposing, all boxes on a given level in the tree are independent of each other. As a result, there are two types of parallelism that can be exploited: dividing boxes among threads and utilizing the multi-threaded linear algebra in MKL. We propose a hybrid of these approaches. In the bottom of the tree where the matrices involved are small, it is best to use a “divide-and-conquer” approach which distributes boxes among all available threads. At the top of the tree, letting the parallel linear algebra have full reign is ideal. The best distribution of work on the intermediate levels depends on the available computational resources and number of boxes on a given level. This section presents a technique for distributing work in the hybrid parallelism setting. Let $\theta_t$ denote the total number of available threads, $\theta_o$ denote the number of outer threads dedicated to a divide-and-conquer distribution of boxes, and $\theta_i$ denote the num-
Algorithm 2 (Solve stage)

This algorithm constructs an approximate solution \( u \) to (1) given the body load \( s(x) \) and incoming impedance boundary condition \( t(x) \).

It is assumed that if node \( \tau \) is a parent of node \( \sigma \), then \( \tau < \sigma \).

Let \( N_{\text{boxes}} \) denote the number of boxes in the tree.

All operators are assumed to be precomputed.

Upward pass

\begin{verbatim}
(1) for \( \tau = N_{\text{boxes}}, N_{\text{boxes}} - 1, N_{\text{boxes}} - 2, \ldots, 1 \)
(2) if (\( \tau \) is a leaf)
(3) Compute \( \tilde{u}^\tau = \Upsilon^\tau s \) and \( h^\tau = \Gamma^\tau s \) for the leaf.
(4) else
(5) Let \( \sigma_1 \) and \( \sigma_2 \) be the children of \( \tau \).
(6) Compute \( \tilde{t}^\sigma_1 = \Upsilon^\sigma_1 \begin{bmatrix} h^\tau_{\sigma_1} \\ h^\tau_{\sigma_2} \end{bmatrix} \) and \( \tilde{t}^\sigma_2 = \Upsilon^\sigma_2 \begin{bmatrix} h^\tau_{\sigma_1} \\ h^\tau_{\sigma_2} \end{bmatrix} \).
(7) Compute \( h^\tau \) via (16).
(8) end if
(9) end for
\end{verbatim}

Downward pass

\begin{verbatim}
(10) for \( \tau = 1, 2, 3, \ldots, N_{\text{boxes}} \)
(11) if (\( \tau \) is a leaf)
(12) Let \( J^\tau \) denote the indices for the discretization points in \( \tau \).
(13) \( u(J^\tau) = \Psi^\tau t^\tau + \tilde{u}^\tau \).
(14) else
(15) Let \( \sigma_1 \) and \( \sigma_2 \) be the children of \( \tau \).
(16) \( t^\sigma_1 = \Phi^\sigma_1 t^\tau + \tilde{t}^\sigma_1, \quad t^\sigma_2 = \Phi^\sigma_2 t^\tau + \tilde{t}^\sigma_2 \).
(17) end if
(18) end for
\end{verbatim}

The number of \textit{inner threads} given to each outer thread for parallel linear algebra. This means \( \theta_t \geq \theta_o \times \theta_i \) and \( \theta_o, \theta_i \in \mathbb{Z} \).

Since the linear algebra for processing a given box is roughly sequential, the distribution of work is based on the most expensive operation in processing a box called the \textit{representative action}. In the build stage, the representative action is inverting a matrix. For leaf boxes, this corresponds to the inverting the approximate differential operator. For merging two boxes, the inverse of the matrix defined on the interface is the representative action. In the solve stage, the representative action in both sweeps of the hierarchical tree is matrix vector multiplication (\textit{matvec}). Table 1 details the matrix size of the representative action for each stage based on level in the tree. We call the time for computing the representative action on level \( l \) the \textit{representative time}. Since this time depends on the number of threads given the parallel linear algebra, we denote the representative time for level \( l \) with \( j \) inner threads by \( r^l_j \). Since the representative
times are machine dependent and the order of discretization order $n_c$ can be
fixed for a variety of problems, the representative times need only be found once
for a machine. The representative times can then be used in the application of
the HPS method to any boundary value problem of the form (1).

**Table 1:** Representative actions used to estimate computation time on a level.

| Stage          | Level type | task      | matrix size       |
|----------------|------------|-----------|-------------------|
| Build          | Leaf       | matrix inversion | interior $\times$ interior |
|                | Others     |            | interface $\times$ interface |
| Upward pass    | Leaf       | matvec    | [interior + exterior] $\times$ interior |
|                | Others     |            | exterior $\times$ interface |
| Downward pass  | Leaf       | matvec    | [interior + exterior] $\times$ exterior |
|                | Others     |            | interface $\times$ exterior |

For a uniform discretization the number of boxes on a given level is $N_l = 4^l$. For a given representative action, the choice of inner and outer thread pair is selected so that

$$
\epsilon_l = \text{ceiling}(N_l/\theta_o) r_l^{\theta_i}
$$

is minimized over all feasible pairs of $\theta_o$ and $\theta_i$.

**Remark 2.** When the hierarchical tree in the HPS method is non-uniform, the size of the largest matrix on a level $l$ can be used to create the calibration data. If the tree is highly non-uniform and is going to remain fixed for a large number of simulations, it may be advantageous to build calibration data for each possible matrix size and adjust the definition of $\epsilon_l$ accordingly.

4. Results

This section illustrates the performance of the parallelization technique when implemented on a dual 2.3 GHz Intel Xeon Processor desktop workstation with 256 GB of RAM and $\theta_t = 56$ threads. The algorithm was implemented in Fortran 95.

Figures 4-5 illustrate the choices of inner $\theta_i$ and outer $\theta_o$ threads obtained via the optimization procedure in the previous section for the build and solve stages of the algorithm applied to uniform trees with different number of levels.

For the build stage, when the number of boxes on a level $N_l$ is greater than 10, it is advantageous to let $\theta_i = 1$ and distribute the boxes amongst the threads. At the top several levels of the tree when the number of boxes on the level is less than or equal to 10, it is advantageous to assign each thread a box, i.e. $\theta_o = N_l$, and divide the remaining threads evenly for parallel linear
algebra. Linear algebra is not parallelized in the lower levels of the tree since the matrices are too small to benefit from it. A similar behavior is observed for the solve stage, though the parallel linear algebra is utilized earlier in the hierarchical tree; when there are less than roughly 15 boxes on a level.

Figure 4: The optimal combinations per level of (a) outer and (b) inner threads in the build stage of the HPS method with 9 (●), 11 (♦), 13 (▲), and 15 (■) total levels in the tree and 56 available threads.

Figure 5: Optimal inner threads on each level of the tree for upward (a) and downward (b) passes of the solve stage with 56 available threads. Total tree depths are 9 (●), 11 (♦), 13 (▲), and 15 (■). The optimal number of outer threads on each level is identical to the build stage, shown in Figure 4(a).

To illustrate the speed of the parallel implementation, the HPS method was applied to (1) with different levels of uniform refinement. The number of discretization points on a leaf ($n_c = 16$) is fixed. Let $N$ denote the number of discretization points where the solution is unknown. Figure 6 reports the execution time, in seconds, for the HPS method with the serial and parallel implementations. Figure 7(a) illustrates the corresponding speedup gained by moving from the serial to parallel implementation of the algorithm. For the largest problem considered with over two million unknowns, the build stage of the algorithm takes roughly 10 minutes via the serial implementation while the parallel implementation takes 36 seconds. This is 17.5 times speedup. Figure 7(b) reports the speedup split between the leaf and merge computations in the
build stage of the algorithm. As expected, the bulk of the speed up is gained on the leaf level where the algorithm is perfectly parallelizable. The speedup of the merge computations is limited by the speedup of the parallel linear algebra inversion provided by MKL. The constant prefactor for the solve stage is small since it is a collection of BLAS3 matvec operations involving small matrices. These efficient operations are precisely why the expected speedup is small. In fact, only a factor of two speedup is obtained.

![Graph](image1)

**Figure 6:** Time in seconds versus the number of discretization points where the solution is unknown $N$ for the (a) serial and mixed threaded implementation for the precomputation stage (■) and solve stage (●). The values $N \log N$ is plotted (---) for reference.

![Graph](image2)

**Figure 7:** (a) Speedup in build (■) and solve (●) stages portions of algorithm for 56 total threads. (b) Breakdown of the speedup between the leaf (■) and the merge (●) in precomputation stage.

5. Conclusions

This paper presented a simple technique for parallelizing the two dimensional HPS method in a shared memory setting with access to parallel linear algebra. The most expensive stage (build) observes a 17.5 times speedup over a serial implementation on a desktop computer which is comparable to a modern super computing node. This corresponds to discretizing a problem with over 2 million
unknowns and building the corresponding direct solver in approximately 30 seconds.

While the techniques presented are applied in a shared memory setting, they are the foundations for full parallelism on super computers and clusters with large memory nodes that are becoming more prevalent. These machines will require two level parallelism where MPI divides portions of the geometry amongst the large memory nodes. Then each node can utilize the shared memory techniques presented in this paper.

The three dimensional version of the HPS method has much larger matrices even close to the leaf level. Thus parallel linear algebra will be utilized earlier in the build stage and the solve stage will see more benefits (i.e. larger speedup) from having access to it, as well.

6. Acknowledgements

The authors wish to thank Total US E&P for permission to publish. The work by A. Gillman is supported by the Alfred P. Sloan foundation and the National Science Foundation (DMS-1522631). A. Gillman and N. Beams are supported in part by a grant from Total E&P Research and Technology USA.

References

[1] A. Gillman, A. Barnett, P. Martinsson, A spectrally accurate direct solution technique for frequency-domain scattering problems with variable media, BIT Numerical Mathematics 55 (1) (2015) 141–170.
[2] L. Trefethen, Spectral methods in MATLAB, SIAM, 2000.
[3] A. Gillman, P. Martinsson, A direct solver with $O(n)$ complexity for variable coefficient elliptic pdes discretized via a high-order composite spectral collocation method, SIAM Journal on Scientific Computing 36 (4) (2014) A2023–A2046.
[4] P. G. Martinsson, A direct solver for variable coefficient elliptic pdes discretized via a composite spectral collocation method, Journal of Computational Physics 242 (0) (2013) 460–479.
[5] T. Babb, A. Gillman, S. Hao, P. G. Martinsson, An accelerated poisson solver based on a multidomain spectral discretization.
[6] P. Geldermans, A. Gillman, An adaptive high order direct solution technique for elliptic boundary value problems, URL https://arxiv.org/abs/1805.01980
[7] A. George, Nested dissection of a regular finite element mesh, SIAM Journal on Numerical Analysis 10 (1973) 345–363.
[8] I. S. Duff, A. M. Erisman, J. K. Reid, Direct methods for sparse matrices, 1989.

[9] E. L. Wilson, The static condensation algorithm, International Journal for Numerical Methods in Engineering 8 (1) (1974) 198–203.

[10] J. Xia, S. Chandrasekaran, M. Gu, X. S. Li, Superfast multifrontal method for large structured linear systems of equations, SIAM Journal of Matrix Analysis and Applications 31 (3) (2009) 1382–1411.

[11] A. Gillman, Fast direct solvers for elliptic partial differential equations, Ph.D. thesis, University of Colorado at Boulder, Applied Mathematics (2011).

[12] S. L. Borne, L. Grasedyck, R. Kriemann, Domain-decomposition based H-lu preconditioners, in: O. B. Widlund, D. E. Keyes (Eds.), Domain Decomposition Methods in Science and Engineering XVI, Vol. 55, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 667–674.

[13] P. G. Martinsson, A fast direct solver for a class of elliptic partial differential equations, Journal of Scientific Computing 38 (3) (2009) 316–330.

[14] P. Schmitz, L. Ying, A fast direct solver for elliptic problems on general meshes in 2D, Journal of Computational Physics 231 (4) (2012) 1314–1338.

[15] X. Li, J. Demmel, J. Gilbert, L. Grigori, M. Shao, I. Yamazaki, SuperLU Users’ Guide, Tech. Rep. LBNL-44289, Lawrence Berkeley National Laboratory, http://crd.lbl.gov/~xiaoye/SuperLU/. Last update: August 2011 (September 1999).

[16] J. W. Demmel, J. R. Gilbert, X. S. Li, An asynchronous parallel supernodal algorithm for sparse gaussian elimination, SIAM J. Matrix Analysis and Applications 20 (4) (1999) 915–952.

[17] X. S. Li, J. W. Demmel, SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems, ACM Trans. Mathematical Software 29 (2) 110–140.

[18] A. Gupta, A shared- and distributed-memory parallel general sparse direct solver, Applicable Algebra in Engineering, Communication and Computing 18 (3) (2007) 263277. doi:10.1007/s00200-007-0037-x

[19] P. R. Amestoy, I. S. Duff, J.-Y. L’Excellent, J. Koster, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM J. Matrix Anal. Appl. 23 (1) (2001) 1541.

[20] P. R. Amestoy, A. Guermouche, J.-Y. L’Excellent, S. Pralet, Hybrid scheduling for the parallel solution of linear systems, Parallel Computing 32 (2) (2006) 136–156, parallel Matrix Algorithms and Applications (PMAA04). doi:https://doi.org/10.1016/j.parco.2005.07.004