Boundary Integral Solver Approaches for Particle Accelerator Simulation Problems and Deployment on NERSC Hardware

Julia Wei, M. Harper Langston, Pierre-David Letourneau, Matthew J. Morse, Larry Weintraub, Aimee Nogoy, Noah Amsel and Richard Lethin
Reservoir Labs Inc.
New York, NY 10012
Email: {wei,langston,letourneau,morse,weintraub,nogoy,amsel,lethin}@reservoir.com

Abstract—Reservoir Labs, through a project titled MACH-B (Multipole Accelerator Codes for Hadron Beams)\(^1\), is developing a Fast Multipole Method [1]–[7] (FMM)-based tool for higher fidelity modeling of particle accelerators for high-energy physics with an initial focus on the next generation of Fermilab’s Synergia simulation package [8]. MACH-B incorporates (1) highly-scalable, high-performance and generally-applicable FMM-based algorithms [5]–[7], [9] to accurately model space-charge effects in high-intensity hadron beams and (2) boundary integral approaches [10]–[12] to handle singular effects near the beam pipe using advanced quadratures. MACH-B will allow for more complex beam dynamics simulations that more accurately capture bunch effects and predict beam loss. By introducing an abstraction layer to hide FMM implementation and parallelization complexities, this project will also remove one of the key impediments to the adoption of FMMs by the accelerator physics community.

In this work, we focus on the following results for the boundary integral solver components of the MACH-B project:

- Study of the relative accuracies of the hedgehog [12] boundary integral solver when evaluating potential and gradient solutions to Laplace’s equation with Dirichlet boundary conditions
- Study of a single-bunch, Gaussian-distributed set of charges within a conducting pipe, using an embedded boundary solver

Results show the ability to simulate charge densities inside of a pipe-shaped object for accelerator simulations, running experiments on a collection of NERSC’s Cori Cray XC40 Intel Xeon “Haswell” processor nodes\(^2\) and Reservoir’s internal computational resources.

Index Terms—combinatorial algorithms, spectral support solver, linear systems, fast solvers, preconditioners, multigrid, graph laplacian, benchmarking, iterative solvers

I. INTRODUCTION

The majority of numerical approaches for accelerator multiparticle-tracking solve the macroscale problem by employing Particle-In-Cell (PIC) methods [8], [13]–[17]. These methods incorporate an Eulerian method for solving the necessary equations and Lagrangian techniques to advect particles through the domain (e.g., see Figure 1). The specific differences in PIC methods are in how mesh values and particle values are mapped back and forth.

Since space-charge modeling in high-intensity hadron beams for the accelerator physics community requires scalable and high-fidelity algorithmic approaches, all computational methods need to be (1) inherently multiscale, (2) exploit locality, (3) reduce expense of non-locality while handling accuracy, (4) guarantee high accuracy when needed, and (5) handle a variety of complex geometries.

Reservoir Labs’ MACH-B project addresses the above five key elements, maintaining the strengths of PIC codes and approaches while further improving upon some of their weaknesses, allowing domain experts to evaluate and optimize various scenarios for complex high-energy physics experiments. The MACH-B technology is based on existing and new mathematical frameworks, providing new scalable, high-performance algorithms that will assist in accurately and rapidly computing a variety of complex particle accelerator simulations; specifically (1) Fast Multipole Methods (FMM) and (2) Boundary Integral Solvers (BIS).

Combining these two technologies, MACH-B constructs an Embedded Boundary Solver (EBS) made up of (1) a multiscale, adaptive FMM solver for computing near- and far-
field interactions within the volume and (2) a boundary integral solver (BIS) based on Quadrature by Expansion (QBX) [12], [18], [19] for fast evaluation of layer potentials near and on a complex geometry. As an example, consider Figure 2: a complex shape with interior domain \( \omega \), boundary \( \delta \omega \) and an inhomogeneous interior PDE of the form: \( L(u)(x) = f(x) \in \omega \), \( u(x) = g(x) \in \delta \omega \), where \( L(u) \) represents the integral operator for the fundamental solution for the underlying PDE (in this case \( L(u) = \Delta u \)).

The EBS approach performs the following steps in Algorithm 1, using FMMs for both major computational steps.

**Algorithm 1 Embedded Boundary Solver**

1. **Embed** the interior domain \( \omega \) in a regular (e.g., a rectangular prism) domain \( \Omega \) with boundary \( \delta \Omega \) and solve \( L(u_1)(x) = f(x) \) in \( \Omega \), where \( f \) is given at interior locations.
2. **Solve** a boundary integral problem, where border data is modified from step 1: \( L(u_2)(x) = 0 \), \( u_2 = g - u_1 \in \delta \omega \).
3. **Combine** \( u_1 \) and \( u_2 \) to obtain the final solution \( u \).

In Section I-A we briefly introduce Fast Multipole Methods (FMM) followed by a brief introduction to Boundary Integral Solvers (BIS) in Section I-B, followed by a deeper discussion of the background for FMM-based BIS and embedded solvers in Section II, a discussion of the code interface in Section III and results in Section IV.

**A. Fast Multipole Methods (FMM)**

Originally designed for Coulomb interactions and belonging to the class of tree codes, FMMs achieve linear scaling by separating near- and far-field interactions (e.g., see Figure 3) on a spatial hierarchy using tree data structures. As they achieve arbitrary precision at modest cost with straightforward error estimates [1]–[6], [9], [20]–[23], FMMs are well-suited for problems requiring high accuracy at large scales, such as in particle accelerator simulations.

FMMs are inherently multiscale, separating a regular domain into disjoint near- and far-field sets, using a tree structure to exploit locality as well as reduce the expense of non-locality through low-rank approximation multipole expansions [1], [4].

**B. Boundary Integral Solvers (BIS)**

For smooth/piecewise-smooth boundaries, such as those seen near particle accelerator pipe walls, boundary integral equation approaches (1) require no need for complex mesh generation for calculating potentials, (2) satisfy far-field boundary conditions, and (3) result in higher degrees of accuracy. At the beam pipe, a BIS can be specifically designed to couple with MACH-B’s proposed domain-based FMM solver to produce an embedded boundary solver (EBS). In cases where periodic or mixed boundary conditions may be required, FMMs can be tailored to handle these with minimal complexity [11], [12], [17]–[19], [22], [24]. Combining these two technologies, MACH-B constructs an Embedded Boundary Solver (EBS) consisting of (1) a multiscale, adaptive FMM solver for computing near- and far-field interactions within the volume (based on [6], [9], [25]) and (2) a Quadrature by Expansion (QBX)-based boundary integral solver (BIS) package, called hedgehog [12] for fast evaluation of layer potentials near and on a source geometry. More details follow in Section II along with a description of the interface and modifications to hedgehog in Section III.

**II. Boundary Integral Solver Background**

Given a smooth boundary \( \partial \Omega \) for domain \( \Omega \), and a smooth function \( g \) on \( \partial \Omega \), the interior Dirichlet problem is:

\[
\begin{align*}
\Delta u(x) &= 0 \quad \text{for } x \in \Omega \\
u(x) &= g(x) \quad \text{for } x \in \partial \Omega
\end{align*}
\]

(1)

(2)

Throughout this section, the free-space, three-dimensional Green’s function \( G \) is used, defined below. In potential theory, it is also referred to as the single-layer Laplace kernel.

\[
G(x, y) = \frac{1}{4\pi \|x - y\|}
\]

(3)

Other kernels used in this problem are the single-layer gradient kernel \( T_G \), the double-layer kernel \( D \), and the double-layer gradient kernel \( T_D \) [10]. These are defined below, where \( n(y) \) is the unit normal pointing outward from \( \partial \Omega \) at point \( y \).

\[
T_G(x, y) = -\frac{(x - y)}{4\pi \|x - y\|^3}
\]

(4)

\[
D(x, y) = \frac{\partial G(x, y)}{\partial n(y)} = \frac{(x - y) \cdot n(y)}{4\pi \|x - y\|^3}
\]

(5)

\[
T_D(x, y) = \frac{1}{4\pi} \left( \frac{3(x - y) \cdot n(y)}{|x - y|^5} - \frac{n(y)}{|x - y|^3} \right)
\]

(6)
Using these kernels, we can express both the potential $u$ and gradient $\nabla u$ as a boundary integral on $\partial \Omega$. The potential $u$ is first re-written supposing an unknown double-layer density $\phi$, which is continuous on $\partial \Omega$ [12], [24] (in the absence of resonance):

$$u(x) = D[\phi(x)] = \int_{\partial \Omega} D(x, y) \phi(y) dy_{\partial \Omega} \text{ for } x \in \Omega$$  \hspace{1cm} (7)

The gradient can be written in an analogous way:

$$\nabla u(x) = \int_{\partial \Omega} T_D(x, y) \phi(y) dy_{\partial \Omega} \text{ for } x \in \Omega$$  \hspace{1cm} (8)

Thus, after first solving for $\phi$, we can then evaluate the quantities $u$ and $\nabla u$ in $\Omega$ using Equations 7 and 8. To find $\phi$, we note that Equation 7 is singular for $x \in \partial \Omega$, resulting in a singular integral. Depending on whether the limit is taken from the interior or exterior of the boundary, the limiting value is given by Equation 10 [24], where the integral is interpreted in the Hadamard sense.

$$u_\pm(x) = \lim_{h \to +0} u(x \mp hn(x))$$  \hspace{1cm} (9)

$$u_\pm(x) = \int_{\partial \Omega} D(x, y) \phi(y) dy_{\partial \Omega} \pm \frac{1}{2} \phi(x) \text{ for } x \in \partial \Omega$$  \hspace{1cm} (10)

Setting the interior limit, $u_+$, equal to the boundary condition $f$ yields the following equation, where $I$ is the identity operator. In practice, Equation 11 is discretized and used to solve for $\phi$.

$$(\frac{1}{2}I + D)[\phi](x) = g(x), x \in \partial \Omega$$  \hspace{1cm} (11)

Ultimately, we are interested in addressing Poisson’s equation, which assumes a charge density $f \in \Omega$, which can be discrete or continuous. Using Dirichlet boundary conditions and assuming a perfectly conducting boundary, the Poisson problem is:

$$\Delta u(x) = f(x) \text{ for } x \in \Omega$$  \hspace{1cm} (12)

$$u(x) = 0 \text{ for } x \in \partial \Omega$$

To evaluate $u \in \Omega$ in the Poisson problem, we split $u$ into $u_1$ and $u_2$, where $u_1$ is a boundary integral evaluation and $u_2$ is a free-space evaluation. The convolution operator is $\ast$.

The steps, as highlighted in Figure 2, are:

1) Find the free-space solution, $u_1 = G * f$.
2) Find $u_1$ for the following Laplace problem:
   $$\Delta u_2(x) = 0 \text{ for } x \in \Omega$$
   $$u_2(x) = -u_1(x)(x) \text{ for } x \in \partial \Omega$$  \hspace{1cm} (13)
3) Evaluate the full solution $u = u_1 + u_2$.

An analogous set of steps can be used to evaluate $\nabla u$ (substituting $\nabla u$ for $u$ and $T_G$ for $G$).

III. BIS-QBX SOFTWARE INTERFACE

We provide high-level descriptions of the libraries used in the boundary integral and embedded boundary solvers. The main C++ libraries used are hedgehog [12], a boundary integral solver, and PVFMM [6], a free-space solver.

The hedgehog library is used to solve the Laplace equation with Dirichlet boundary conditions: it solves Equation 11 and evaluates Equations 7 and 8. It provides the following classes and functionality:

- **PatchSurfFaceMap** class. A C++ class to store surface information. This class implements the following methods:
  - **setup()**: Stores a surface as a set of quadrilateral patches.
  - **refine_uniform()**: Quadrisects each surface patch.

- **SolverGMRESDoubleLayer** class. A C++ class that implements the boundary integral solver for a given kernel. This class implements the following methods:
  - **setup()**: Generates surface points representing a coarse, discretized boundary, and a refined, discretized boundary, using a **PatchSurfFaceMap** object [12].
  - **solve()**: Solves for the double-layer density given a set of boundary condition values.
  - **evaluate()**: Evaluates the boundary integral at a set of target points given the double-layer density.

- **PVFMM** class. A C++ class that acts as a wrapper to the PVFMM library. This class implements the following methods:
  - **evaluate()**: Evaluates the kernel sum given source values, source locations, and target locations.

The $G$ and $D$ kernels are available in hedgehog. The $T_G$ and $T_D$ kernels were implemented in order to evaluate the gradient, needed for modeling the electric field.

IV. NUMERICAL RESULTS

We begin with a discussion of our comparison methods and visualization pipelines in Section IV-A, followed by results in Sections IV-B and IV-C.

A. BIS Methods and Workflow for Solutions and Visualization

Both the BIS and EBS require as inputs a geometry file, a configuration file, and a set of evaluation points. For comparison testing, the analytical solution can be provided if known. The BIS also requires a boundary value evaluation method, while the EBS requires a charge density (discrete or continuous). The outputs are a set of VTK files: a VTU file containing the surface patches, and VTP file(s) containing the solution at the evaluation points, and the relative error if the analytical solution is known.

The geometry input file uses the WRL format, which consists of vertex locations and faces. For now, only meshes with quadrilateral faces are valid geometries. Blender\(^3\) is

\(^3\)http://www.blender.org
used to create quadrilateral meshes (typically in the OBJ file format), which can then be easily converted to the VRML format. The output VTK files can be read by Paraview for data visualization.

**B. Results: BIS examples**

We study the behavior of the boundary solver for two boundary geometries: a cube with side-length 1 (located at $[-0.5,0.5]^3$), and a cylinder with radius 0.5 and length 10 (located within $[-0.5,0.5]^2 \times [-5,5]$). For a fixed distance $\delta$ and a point $y$ on the discretized boundary, we have the interior point $s(y)$:

$$s(y) = y - \delta n(y)$$  \hspace{1cm} (15)

In this example, we fix a unit source point at $[1,0,0]$. After generating the set of interior evaluation points for a fixed $\delta$ and boundary discretization, we find the relative errors $\epsilon_{rel}$ of the boundary integral solver, shown below for the potential and gradient in Equations 16 and 17. The PVFMM library is used to evaluate $f$, the boundary condition, as well as $u_{true}$ and $\nabla u_{true}$, the true potential and gradient. The $\infty$-norm is taken over the set of evaluation points.

$$\epsilon_{rel} = \frac{||u(x) - u_{true}(x)||_\infty}{||u_{true}(x)||_\infty}$$  \hspace{1cm} (16)

$$\epsilon_{rel} = \frac{||\nabla u(x) - \nabla u_{true}(x)||_\infty}{||\nabla u_{true}(x)||_\infty}$$  \hspace{1cm} (17)

Using this setup, we characterize the boundary integral solver’s accuracy by varying the number of patches representing the boundary (see Figures 4 and 5), and the distance from the boundary for the cube and cylinder meshes. The key fixed parameters for hedgehog, as well as their definitions and values, are given in Table I.

**TABLE I**

**PARAMETER DESCRIPTIONS AND VALUES FOR HEDGEHOG.**

<table>
<thead>
<tr>
<th>Usage</th>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface Settings</td>
<td>patch_order $q$</td>
<td>Polynomial surface order Quad. order</td>
<td>20 25</td>
</tr>
<tr>
<td></td>
<td>m</td>
<td>Multipole order</td>
<td>12 1000</td>
</tr>
<tr>
<td></td>
<td>ptmax</td>
<td>Num. points per leaf box</td>
<td></td>
</tr>
<tr>
<td>GMRES Settings</td>
<td>max_it</td>
<td>Max num. GMRES iter</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>rtol</td>
<td>Relative tolerance</td>
<td>1e-12</td>
</tr>
<tr>
<td>Extrapolation</td>
<td>a</td>
<td>Prop. constant for distance between surface and the first checkpoint; typically set to $b/p$</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>Prop. constant for distances between checkpoints</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>Num. interpolation points</td>
<td>6</td>
</tr>
</tbody>
</table>

For the distances $\delta$ considered, $\delta = 0.05$ can be interpreted as “near” the boundary, while $\delta = 0.2$ is roughly in-between the center and the boundary. In Table II, we observe a decrease in error as the number of patches increases, and as the target points move further away from the boundary (increasing $\delta$). If the user provides a relative accuracy and an expected minimum distance $\delta$ from the boundary, mesh refinement can be used to achieve the requested accuracy. To run the calculations, Reservoir computing resources (256 GB RAM, single-processor) were used for the cases of 96 and 384 patches. For the case of 1536 patches, NERSC computing resources and MPI were used (6 Cori Haswell nodes, 1 MPI process/node).

**TABLE II**

**RELATIVE ERROR OF THE POTENTIAL AND GRADIENT FOR THE CUBE-SHOPHERIAL MESH FOR DIFFERENT DISTANCES $\delta$ AND DIFFERENT LEVELS OF MESH REFINEMENT.**

<table>
<thead>
<tr>
<th>Kernel</th>
<th>$\delta$</th>
<th>96</th>
<th>384</th>
<th>1536</th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential</td>
<td>0.05</td>
<td>$6.38 \times 10^{-6}$</td>
<td>$1.30 \times 10^{-6}$</td>
<td>$2.46 \times 10^{-7}$</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>$4.50 \times 10^{-6}$</td>
<td>$8.55 \times 10^{-7}$</td>
<td>$1.45 \times 10^{-7}$</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>$2.21 \times 10^{-6}$</td>
<td>$4.15 \times 10^{-7}$</td>
<td>$7.03 \times 10^{-8}$</td>
</tr>
<tr>
<td>Gradient</td>
<td>0.05</td>
<td>$5.57 \times 10^{-3}$</td>
<td>$1.24 \times 10^{-9}$</td>
<td>$2.59 \times 10^{-6}$</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>$2.88 \times 10^{-5}$</td>
<td>$5.94 \times 10^{-6}$</td>
<td>$1.13 \times 10^{-6}$</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>$1.68 \times 10^{-5}$</td>
<td>$3.19 \times 10^{-6}$</td>
<td>$5.42 \times 10^{-7}$</td>
</tr>
</tbody>
</table>

The same trend holds for Table III using the cylinder mesh, so that a refined mesh can be used to accurately evaluate the potential and gradient for the Laplace equation. In Figure 6, we observe that the relative error increases for evaluation points closer to the source. Again, Reservoir computing resources were used to evaluate 42 and 168 patches, while NERSC computing resources were used to evaluate 672 patches (14 Cori Haswell nodes, 1 MPI process/node).
TABLE III
RELATIVE ERROR OF THE POTENTIAL AND GRADIENT FOR THE CYLINDER MESH FOR DIFFERENT DISTANCES \( \delta \) AND DIFFERENT LEVELS OF MESH REFINEMENT.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>( \delta )</th>
<th>168</th>
<th>672</th>
<th>2688</th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential</td>
<td>0.05</td>
<td>6.27 × 10^{-3}</td>
<td>4.68 × 10^{-6}</td>
<td>8.87 × 10^{-7}</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>1.70 × 10^{-5}</td>
<td>3.56 × 10^{-6}</td>
<td>6.54 × 10^{-7}</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>9.07 × 10^{-6}</td>
<td>1.70 × 10^{-6}</td>
<td>3.20 × 10^{-7}</td>
</tr>
<tr>
<td>Gradient</td>
<td>0.05</td>
<td>8.43 × 10^{-3}</td>
<td>8.17 × 10^{-9}</td>
<td>5.84 × 10^{-6}</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>8.49 × 10^{-5}</td>
<td>1.96 × 10^{-5}</td>
<td>3.86 × 10^{-6}</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>5.53 × 10^{-5}</td>
<td>1.20 × 10^{-5}</td>
<td>2.25 × 10^{-6}</td>
</tr>
</tbody>
</table>

Fig. 6. Relative error for the potential (top) and gradient (bottom) using the cylinder mesh for the \( \delta = 0.05 \) and 168 patches case. The cylinder is oriented so that the +x axis faces outward, and the +z axis faces the right.

Generally, there are a couple of digits of loss in precision between the potential and gradient for a given \( \delta \) and number of patches. Empirical studies of the quadrature and extrapolation methods used can better guide parameter choices, in \( q_r \) and in \( a, b, \) and \( p \), respectively. Moreover, the gradient presents additional challenges due to the evaluation of a hypersingular boundary integral. For target points near the boundary, methods development beyond one-dimensional Legendre interpolation may be needed to achieve a more accurate or stable evaluation. The memory and time requirements of the code have not been studied, and the code has not been optimized for distributed-memory parallelization, nor for process or thread affinities. Optimizing the code and scripts for use in larger-scale simulations will be part of future research.

C. Results: EBS example

We use the EBS to model the potential of a single-bunch, Gaussian distribution of charges within a cylindrical conducting pipe (same dimensions as the BIS example). By extending the length of the pipe, this model can also be used to approximate open boundary conditions at the ends.

The interior charges are Gaussian-distributed with a density at \( \mathbf{x} \) of \( p(\mathbf{x}) \):

\[
p(\mathbf{x}) = \frac{1}{\sigma^3(2\pi)^{3/2}} \exp(-\frac{||\mathbf{x}||^2}{2\sigma^2})
\]  

(18)

For \( \sigma = 0.1 \), we generate \( N = 10^4 \) source points by randomly sampling from the Gaussian, and discarding points where \( x^2 + y^2 > 0.025 \), or where \( |z| > 0.5 \) (truncated Gaussian), until \( N \) points are obtained.

The \( x-y \) and \( x-z \) planes from the EBS are below, showing an expected decay away from the origin.

Fig. 7. Cylinder cross-section at \( z = 0 \).

Fig. 8. Zoom-in of the cylinder cross-section at \( y = 0 \), showing only the +x half-plane.

V. CONCLUSIONS AND FUTURE WORK

We have exhibited the ability to use Quadrature by Expansion approaches using hedgehog for successfully solving boundary integral equations with FMM-based underpinnings on NERSC hardware. We have further shown that we can use simple and complex geometry representations and achieve high levels of accuracy for both simple kernels and the more complex kernels to implement for the problems of interest to the particle accelerator community. We anticipate that we are ready to begin incorporating these approaches into CAMPA codes using more specific geometries of interest.

Beyond continued theoretical developments of the BIS, MPI/OpenMP parallelization, and ease-of-use software/documentation improvements, next steps and additional features could include:

- Enabling longitudinal, periodic boundary conditions for modeling bunched beams, and comparisons to [17]. Similar to the EBS setup, and could be done by replacing the PVFMM with the STKFMM library [9] within hedgehog, and then approximating discontinuous boundary conditions with radial basis functions.
- Examples using more complex geometries relevant to the particle accelerator community.

\(^5\)Consortium for Advanced Modeling of Particle Accelerators: https://campa.lbl.gov/codes/
• An interface to Cubit, or other mesh generation software relevant to the particle accelerator community.

In particular, we have had ongoing discussions with Synergia engineers at Fermilab and the Accelerator Simulation Group about approximating a geometry of the quadrupole in the Debuncher machine as seen in Figure 9.

In preparation for incorporating such boundaries, in figure 10 (Left), we approximated a closed, non-periodic version of the aperture for a simple geometry. For an approximation of the boundary at \( h_s = 0.024 \) and a sample nonuniform force distribution, we perform a sample discretization of the shape in figure 10 (Right) as in [11].

Through the tests we have performed with hedgehog and the introduction of complex shapes, we have shown that we can approach the problems in HEP particle accelerator simulations with FMM-based boundary integral solvers. Further parallelization, acceleration, incorporation into particle accelerator codes and further deployment on NERSC hardware are planned for the immediate future.

REFERENCES