A Fast, Unstructured Panel Solver

John Moore*

18.337 Final Project, Fall, 2012

A parallel high-order Boundary Element Method accelerated by the Fast Multipole Method is presented in this report. The case of potential flow about arbitrary geometries will be investigated using this method. The goal of this application is to provide a "push-button" tool for the design of aircraft, where medium-fidelity solutions can be obtained in seconds and not hours or days required by Navier-Stokes solvers.

I. Introduction

PANEL methods are currently capable of rapidly solving the potential flow equation on rather complex geometries using only a workstation. These methods, which originated in the early 1960's, continue to be attractive since the governing equations only need to be solved at the boundary.[1–4] This eliminates the need for a volume mesh, as is needed when using Finite Difference, Finite Volume, or Finite Element methods, and results in a system which is of comparatively lower dimension. However, unlike these methods, panel methods result in a system of equations that is dense and memory requirements grow as the square of the number of degrees of freedom. In the late 1980's, Greengard et al.[5] developed the Fast Multipole Method (FMM) which reduced the computational complexity of performing matrix-vector products in particle simulations from $O(N^2)$ to $O(N)$, with a memory requirement that scales as $O(NlogN)$. In the early 1990's, the FMM was coupled with a Boundary Element Method (BEM) solver in order to rapidly simulate circuits.[6] In the late 1990's, this same approach was applied to aerodynamic cases with a computational complexity of $O(NlogN)$.[7–9]

This project seeks to develop a fast, parallel implementation of a potential flow solver that scales as $O(N)$ where N are the number of elements in the surface discretization. OpenMP will be used for the parallel environment, since the primary use of this code is seen as being a design tool used on a workstation or a laptop. This document will detail the formulation of a steady-state Boundary Element Method, followed by a detailed description of the Fast Multipole Method. Results will be presented for varying number of processors and various multipole expansion orders.

II. Incompressible Potential Flow

A. Scalar and Vector Potentials

Incompressible potential flow can be represented by a scalar potential field ϕ.[10] The fluid velocity at point $\vec{r}$ with respect to some origin is the superposition of the gradient of the scalar potential and the freestream velocity:

$$\vec{U}(\vec{r},t) = \nabla\phi(\vec{r},t) + \vec{V}_\infty \tag{1}$$

The continuity equation for an incompressible fluid can be written as:

$$\nabla \cdot \vec{U} = 0 \tag{2}$$

Equation 1 can be substituted into equation 2 to yield Laplace's equation which governs inviscid, incompressible, and irrotational flow:

$$\nabla^2\phi = 0 \tag{3}$$

The curl of the velocity field is the vorticity, $\vec{\omega}$, in the domain:

$$\vec{\omega} = \nabla \times \vec{U} \tag{4}$$

*MSc Sudent, Department of Aeronautics and Astronautics, MIT, 77 Massachusetts Ave., Cambridge, MA, 02139.

B. Boundary Conditions

In contrast to tratitional CFD methods where boundary conditions are only applied at boundaries, the BEM requires that a boundary condition be prescribed wherever a solution is needed. In potential flows, the boundary condition is a no-penetration condition, but not a no-slip condition since the flow is inviscid:

$$\vec{U}(\vec{r},t)\cdot\hat{n}(\vec{r},t)=\left(\nabla\phi(\vec{r},t)+\vec{V}_\infty\right)\cdot\hat{n}(\vec{r},t)=0 \tag{5}$$

where $\hat{n}(\vec{r},t)$ is the unit vector normal to the aerodynamic surface at $\vec{r}$. The no-penetration condition can be enforced explicitly with a Neumann boundary condition on the velocity or implicitly with a Dirichlet boundary condition on the potential inside the body, which will be discussed later.

C. Steady Bernoulli Equation

Pressure and velocity in an incompressible flow can be related by the steady Bernoulli equation:

$$\frac{1}{2}||\vec{U}||^2=\frac{1}{2}||\nabla\phi+\vec{V}_\infty||^2=\frac{p}{\rho} \tag{6}$$

The coefficient of pressure, C_p is defined as:

$$C_p=\frac{p-p_\infty}{p_\infty} \tag{7}$$

where $p_\infty=\frac{1}{2}V_\infty^2$ is the freestream dynamic pressure. The coefficient of pressure can be re-written using equation 6:

$$C_p=\frac{\frac{1}{2}||\nabla\phi+\vec{V}_\infty||^2}{\frac{1}{2}||\vec{V}_\infty||^2}-1 \tag{8}$$

D. Kutta Condition

Inviscid lifting flows require invoking the Kutta condition to enforce pressure continuity at the trailing edge and ensure the flow at the trailing edge leaves cleanly. Pressure continuity requires that the strength of the wake sheet potential must equal the jump in potential from the upper trailing edge surface to the lower trailing edge surface:[11] Note that the results in this project were obtained for non-lifting flow and hence no Kutta condition was applied, but the code can be easily modified to include the Kutta condition if needed.

$$\phi_u-\phi_l=\phi_w \tag{9}$$

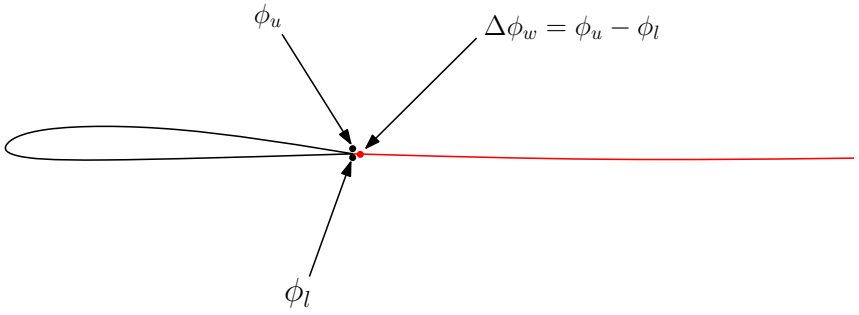

Figure 1. Potential shed into the wake in a direction that bisects the trailing edge with a strength equal to the jump in potential between the upper and lower trailing edge surfaces.

III. Boundary Integral Equations

Laplace's equation is discretized using the BEM based on the Morino formulation[2] where the *perturbation* potential is equal to zero inside the body, forcing the body to be a streamsurface. This results in a

Dirichlet boundary condition. The perturbation potential at some evaluation point $\vec{r}_i$ infinitesimally inside the aerodynamic body due to source and doublet panels distributed over surface S_j is:

$$\phi(\vec{r}_i) = \int_{S_j} (G_d(\mu_j, \vec{r}_i - \vec{r}_j) + G_s(\sigma_j, \vec{r}_i - \vec{r}_j)) = 0 \tag{10}$$

where G_d is the Green's function for double-layer potential satisfying Laplace's equaiton, G_s is the corresponding equation for the single-layer potential, μ is the double-layer (doublet) potential strength, and σ is the single-layer (source) potential strength. For the remainder of this document, i will be used to denote an "evaluation index" and j (and sometimes k) will be used as a "origin index". Green's function for single-layer potential is:

$$G_s(\sigma_j, \vec{r}_i - \vec{r}_j) = \frac{1}{4\pi} \frac{\sigma_j}{||\vec{r}_i - \vec{r}_j||} \tag{11}$$

Green's function for double-layer potential is:

$$G_d(\mu_j, \vec{r}_i - \vec{r}_j) = \frac{1}{4\pi} \frac{\partial}{\partial \hat{n}_j} \frac{\mu_j}{||\vec{r}_i - \vec{r}_j||} \tag{12}$$

The perturbation potential in equation 10 can be decomposed into contributions from the aerodynamic surface B_j and any panelled wake surface B_{W_j}:

$$\phi(\vec{r}_i) = \frac{1}{4\pi} \int_{B_j} \mu_j \frac{\partial}{\partial \hat{n}_{B_j}} \frac{1}{||\vec{r}_i - \vec{r}_j||} dB_j + \frac{1}{4\pi} \int_{B_j} \sigma_j \frac{1}{||\vec{r}_i - \vec{r}_j||} dB_j + \frac{1}{4\pi} \int_{B_{W_j}} \mu_{w_j} \frac{\partial}{\partial \hat{n}_{B_{W_j}}} \frac{1}{||\vec{r}_i - \vec{r}_j||} dB_{W_j} = 0 \tag{13}$$

At least one row of panelled wake needs to be shed from the trailing edge in order to enforce the Kutta condition. Note that μ_{w_j} is known from the Kutta condition. σ_j is chosen to equal the component of all velocity in the domain not generated by equation 13 in the direction normal to surface B_j:

$$\sigma_j(\vec{r}_j) = \vec{V}_\infty \cdot \hat{n}_j \tag{14}$$

σ can be thought of as a blowing term accounting for all the velocity not generated directly by the aerodynamic surface. The only unknown in the system are the doublet strengths, μ_j on surface B_j. The velocity at some point $\vec{r}_i$ is simply the gradient of the perturbation potential according to equation 1. Referring to equation 13, the velocity becomes

$$\begin{aligned} \vec{U}(\vec{r}_j) &= \tfrac{1}{4\pi} \nabla \textstyle\int_{B_j} \mu_j \frac{\partial}{\partial \hat{n}_{B_j}} \frac{1}{||\vec{r}_i - \vec{r}_j||} dB_j + \frac{1}{4\pi} \nabla \int_{B_j} \sigma \frac{1}{||\vec{r}_i - \vec{r}_j||} dB_j \\ &+ \tfrac{1}{4\pi} \nabla \textstyle\int_{B_{W_j}} \mu_w \frac{\partial}{\partial \hat{n}_{B_{W_j}}} \frac{1}{||\vec{r}_i - \vec{r}_j||} dB_{W_j} + \vec{V}_\infty \end{aligned} \tag{15}$$

IV. Discretization and Implementation

The boundary integral equations are discretized with a Galerkin formulation of the Boundary Element Method. Assume the aerodynamic surface S_j in the previous section is discretized into N_E elements and the wake surface S_{W_j} is discretized into and N_W wake elements. This results in N_E aerodynamic panels ,S_j, and N_W wake panels, $S_{W\,j}$. The perturbation potential at some evaluation point $\vec{r}_i$ just inside the aerodynamic surface can be expressed as:

$$\begin{aligned} \phi(\vec{r}_i) &= \textstyle\sum_{j=1}^{N_E} \left(\frac{1}{4\pi} \int_{S_j} \mu_j \frac{\partial}{\partial \hat{n}_{S_j}} \frac{1}{||\vec{r}_i - \vec{r}_j||} dS_j + \frac{1}{4\pi} \int_{S_j} \sigma_j \frac{1}{||\vec{r}_i - \vec{r}_j||} dS_j \right) \\ &+ \textstyle\sum_{j=1}^{N_W} \frac{1}{4\pi} \int_{S_{W\,j}} \mu_{w_j} \frac{\partial}{\partial \hat{n_{S_j}}} \frac{1}{||\vec{r}_i - \vec{r}_j||} dS_{W\,j} = 0 \end{aligned} \tag{16}$$

The integrals over the triangular panels S_j and S_{W_j} have been analytically derived by Neumann[12] for arbitrarily high order source and doublet distributions. Currently, panels with linearly varying source and doublet strengths are implemented. The Galerkin method minimizes the residual over the aerodynamic

surface by multiplying each target panel with a test basis function and integrating over the panel. For each target panel i with surface S_i, a residual can be written in terms of the Boundary Integral Equation:

$$R_i = \int_{S_i} a_i \phi_i(\vec{r}_i) dS_i = \int_{S_i} a_i \Bigg[\sum_{j=1}^{N_E} \Big(\frac{1}{4\pi} \int_{S_j} \mu_j \frac{\partial}{\partial \hat{n}_{S_j}} \frac{1}{||\vec{r}_i - \vec{r}_j||} dS_j + \frac{1}{4\pi} \int_{S_j} \sigma_j \frac{1}{||\vec{r}_i - \vec{r}_j||} dS_j \Big) + \sum_{j=1}^{N_W} \frac{1}{4\pi} \int_{S_{W_j}} \mu_{w_j} \frac{\partial}{\partial \hat{n}_{S_{W_j}}} \frac{1}{||\vec{r}_i - \vec{r}_j||} dS_{W_j} \Bigg] dSi = 0 \tag{17}$$

The basis functions a_i are chosen to be linear, and the integrals over each target element surface S_i are computed using Gaussian quadrature.

V. Fast Multipole Method

This section presents the theory of the Fast Multipole Method and describes how it is used to accelerate the flow solver. The equations presented below closely follow those developed by Greengard[5] for use in particle simulations. A brief explanation of the fast multipole method will be given, and will be followed by a more detailed description of the translation operators. The FMM is composed of three primary routines: the octree domain decomposition, the upward pass, and the downward pass. Each of the three routines are detailed below. Figure 3 depicts the primary steps of the FMM algorithm.

A. Octree Decomposition

A cubic domain is generated spanning all the sources, doublets, and vortons in space. The cube is recursively divided into eight boxes, until there are no more than N_{MAX} elements in a box. Boxes with no elements in them are deleted. The FMM method requires that each cell know which its *nearest neighbors* and *second nearest neighbors* are. A nearest neighbor is defined as a box who touches another box, even if they are just touching at a point. Nearest neighbors can be easily computed by traversing down the octree, searching through the *children* of each box's *parent*. The second nearest neighbors can be computed by searching through the nearest neighbors of each box's nearest neighbors. The computational cost of the octree domain composition can range from $O(NlogN)$ to $O(N)$, depending on the homogeneity of the singularity distributions where N is the number of elements in the domain.

B. Upward Pass

The upward pass is composed of two steps: (1) Multipole Generation and (2) the Multipole-to-Multipole ($M \rightarrow M$) translation. In the Multipole Generation step, multipole expansion coefficients are computed due to all sources and dipoles in each *childless* box. This will result in two expansion coefficients for each box, corresponding to the source and doublet potentials. The complexity of this step is $O(Np^2)$, where p is the order of the multipole expansion. The Multipole-to-Multipole translation is the recursive translation of the multipole coefficients up the tree, starting with the childless boxes. The computational complexity of this step is $O(Np^4)$.

C. Downward Pass

The downward pass is composed of four steps: (1) the Multipole to Local ($M \rightarrow L$) translation,(2) the Local to Local translation ($L \rightarrow L$), (3) the Direct Evaluation, and (4) the Local Expansion evaluation. The ($M \rightarrow L$) translation involves the conversion of all the multipole expansions of the boxes in a box's *interaction list* into a Taylor series expansion about the box's center. The interaction list can be defined as all the second nearest neighbors of a box, or a more complicated criteria.[5] This is the most expensive part of the FMM algorithm due the large number of boxes in the second nearest neighbor list and has a complexity that scales as $O(Np^4)$. The $L \rightarrow L$ translation is a recursive translation of local coefficients down the octree from each box to it's child. The complexity of this operation scales as $O(Np^4)$. The Direct Evaluation step computes the interaction between all sources, dipoles, and vortons in a box and all sources, dipoles, and vortons in the box's nearest neighbors. The complexity of this operation is $O(Ns)$, where s is a measure of the FLOPS required to analytically integrate the source and dipole distribution over a element. s is approximately 100 for a linear panel. The final step is the evaluation of the Local Expansions in each

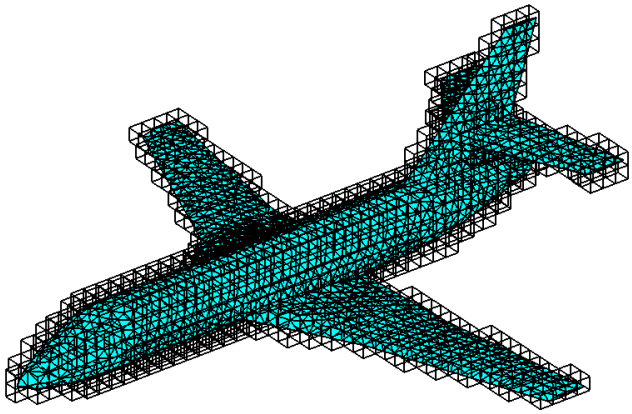

Figure 2. All childless boxes for an octree about a busness jet

childless box. This step has a complexity of $O(Np^2)$ since it is not a translation operation, only a local evaluation.

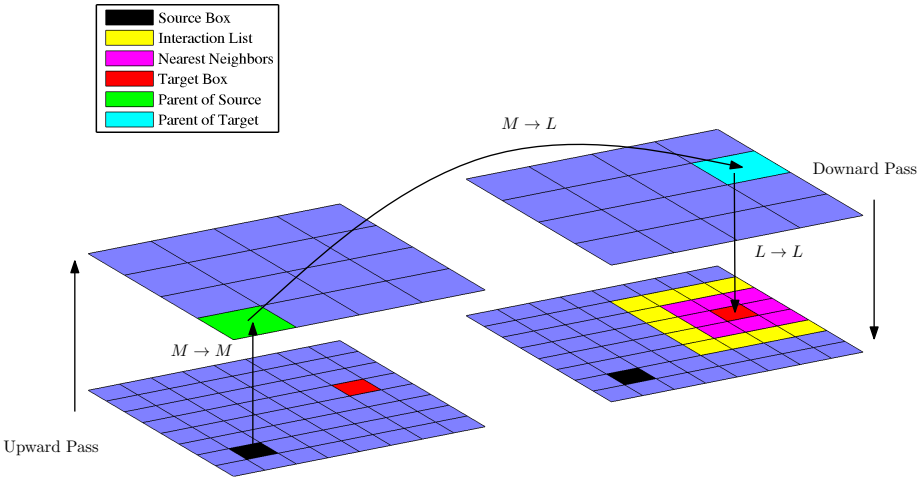

Figure 3. Diagram of FMM translations and interaction lists between two levels of a 2D grid. Note that the grids on the right and left are identical, and are duplicated for the purpose of visualizing the upward and downward passes.

D. Translation and Evaluation Operators

In three dimensions, multipole expansions are expressed in terms of spherical harmonics. The spherical harmonics have a degree n and order m and are defined by the following formula:[5]

$$Y_n^m = \sqrt{\frac{(n-|m|)!}{(n+|m|)!}} P_n^{|m|} cos(\theta) e^{im\phi} \tag{18}$$

where P_n^m are associated Legendre functions. A truncated multipole expansion of order p about a box origin representing a collection of K sources of strength q_s located at spherical coordinates $(\rho_1, \alpha_1, \beta_1)...(\rho_K, \alpha_K, \beta_K)$ with respect to the box centroid is defined by the formula:

$$\Phi(r,\theta,\phi) = \sum_{n=0}^{p} \sum_{m=-n}^{n} \frac{M_n^m}{r^{n+1}} Y_n^m(\theta,\phi), \quad M_n^m = \sum_{s=1}^{K} \sigma_s \rho_s^n Y_n^{-m}(\alpha_s, \beta_s) \tag{19}$$

where (r,θ,ϕ) are the spherical coordinates of the point where the multipole expansion is evaluated. Similarly, the multipole coefficients M_n^m of the multipole expansion for a collection of K dipoles with strength μ_s and orientation $\hat{n}_s$ is

$$M_n^m = \sum_{s=1}^{K} \mu_s \frac{\partial}{\partial n_s} \left(\rho_s^n Y_n^{-m}(\alpha_s, \beta_s) \right) \tag{20}$$

Each surface panel is represented by a collection of point sources and dipoles, and their strength and location on each panel is given by the weights and spacing of a numerical quadrature rule. In Multipole to Multipole step of the algorithm, multipole expansions of child boxes must be transferred to their parent box. Let (ρ, α, β) be spherical coordinates of the parent box's centroid with respect to the child box's centroid. Then the multipole coeffecients MT of the parent box are given by the following multipole translation formula

$$MT_j^k = \begin{cases} \sum_{n=0}^{j} \sum_{m=-n}^{n} \frac{M_{j-n}^{k-m} \cdot i^{|k|-|m|-|k-m|} \cdot A_n^m \cdot A_{j-n}^{k-m} \cdot Y_n^{-m}(\alpha,\beta)}{A_j^k} x & |k-m| <= p \\ 0 & |k-m| > p \end{cases} \tag{21}$$

where

$$A_n^m = \frac{(-1)^n}{\sqrt{|n-m|! \cdot |n+m|!}} \tag{22}$$

The Multipole to Local translation is detailed below. Let (ρ, α, β) be the spherical coordinates of a centroid of a box in a box's interaction list with respect to the centroid of the box itself. Let M_n^m be the coefficients of the multipole expansion of a box in the interaction list of this same box. Then the corresponding local expansion coefficients about the center of the box is:

$$L_j^k = \sum_{n=0}^{p} \sum_{m=-n}^{n} \frac{M_n^m \cdot i^{|k-m|-|k|-|m|} \cdot A_n^m \cdot A_j^k \cdot Y_{j+n}^{m-k}(\alpha, \beta)}{(-1)^n A_{j+n}^{m-k} \cdot \rho^{j+n+1}} \tag{23}$$

The Local to Local translation is detailed below. Let (ρ, α, β) be the spherical coordinates of a child box's centroid with respect to its parent's centroid. If L_n^m are the coefficients of the parent box's local expansion, then the corresponding coefficients of the local expansion about the child box's centroid are:

$$LT_j^k = \sum_{n=j}^{p} \sum_{m=-n}^{n} \frac{L_n^m \cdot A_{n-j}^{m-k} \cdot A_j^k \cdot Y_{n-j} m-k(\alpha, \beta) \cdot \rho^{n-j}}{(-1)^{n+j} \cdot A_n^m} \tag{24}$$

The Local Evaluation is detailed below. The potential resulting from the evaluation of the local expansion at spherical coordinates (r, θ, ϕ) with respect to a box centroid is computed as follows:

$$\Phi(r, \theta, \phi) = \sum_{j=0}^{p} \sum_{k=-j}^{j} LT_j^k \cdot Y_j^k(\theta, \phi) \cdot r^j \tag{25}$$

where LT_j^k are the coefficients of the local expansion representing the potentials of all cells which are not nearest neighbors of the box. Note that there will be two sets of LT_j^k coefficients per childless box, corresponding to the contributions from the source and doublet elements.

VI. Implementation and Results

This section will discuss the implementation of the FMM-accelerated BEM and present results of both serial and parallel FMM-accelerated non-lifting potential flow solvers.

A. Parallel Implementation

Several options were considered for the parallel implementation of the FMM-accelerated BEM. Initially, the code was written in MATLAB, but I soon realized that MATLAB is far too slow for an efficient implementation of the Fast Multipole Method. I decided to go ahead and write the code in C++, sine an object-oriented framework is really needed for the FMM algorithm. Upon writing the serial C++ code, speed-ups of as many as four orders of magnitude were realized. The fact that the switch from MATLAB to C++ increased performance so much steered me away from using Julia for this project, since I initially viewed Julia as a scripting language. However, upon seeing some more Julia features as the class progressed and listening to the project presentations, I would re-consider implementing my method in Julia.

I ended up choosing to implement my method in OpenMP, since it is easy to take a serial code and add shared-memory parallel functionality and because I only envision seeing this type of code being run on a laptop or workstation due to the fast runtimes. Another possible option would have been a GPU implementation, but many laptops and workstations are not equipped with good computing GPUs, so the GPU programming option was thrown out.

It took a good bit of effort to just get the serial code working, as the serial code is composed of 6,500 lines of code. However, once the serial code was validated, adding parallel functionality was relatively straightforward. OpenMP only requires that the programmer add preprocessor directives to a serial code, and almost everything else is handled internally. An example of the parallelized $M2L$ translation function is shown below:

```
// Multipole to local
for(int lev = 2; lev < Nlevel; lev++){
#pragma omp parallel for
        for(int i=0; i<level_index[lev].idx.size(); i++){
        int bx = level_index[lev].idx[i];
```

```
            if(boxes[bx].isevalbox){
            for(int j=0; j<boxes[bx].ilist.size(); j++){
          if(boxes[boxes[bx].ilist[j]].issourcebox){
           multipole_to_local(boxes[boxes[bx].ilist[j]],boxes[bx],Bnm,Anm,ilist_consts[bx][j],
                        }
                    }
      }
    }
}
```

In this function, the only change from the serial case was the addition of the #pragma omp parallel for line.

B. Solving the System of Equations

The discretized version of the Galerkin BEM given in equation 17 can be written in matrix form as a linear system of equations:

$$A\mu = b \tag{26}$$

where the matrix A will be referred to as the influence matrix, and b is the right hand side representing the monopole contributions. In the full (non FMM-accelerated) BEM, the system is generally solved with an iterative solver such as GMRES. Since we are solving a Fredholm equation of the second kind, the system of equations is usually well conditioned. For most cases that have been considered, GMRES converges within 100 iterations. However, this is sub-optimal and better convergence can be obtained with a preconditioner. In the FMM-accelerated BEM, the system must be solved with an iterative solver, since the influence matrix is never explicitly calculated. It would seem that since we don't have the influence matrix,it would be very difficult to generate a good preconditioner. However, if we consider what the FMM algorithm does, we can make use of the FMM method itself to create a sparse preconditioning matrix.

We can make the assumption that near-field interactions are going to play a larger role than far-field interactions. Therefore, in the full BEM, entries in the influence matrix corresponding to near-field interactions will have a much larger magnitude than matrix entries corresponding to far-field interactions. This suggests that generating a sparse matrix including only near-field interactions will represent the structure of the original, non-FMM matrix well. LU factorization can then be performed on this near-field matrix to create the preconditioner $P = LU$. The near-field matrix is shown below for the case of a business jet:

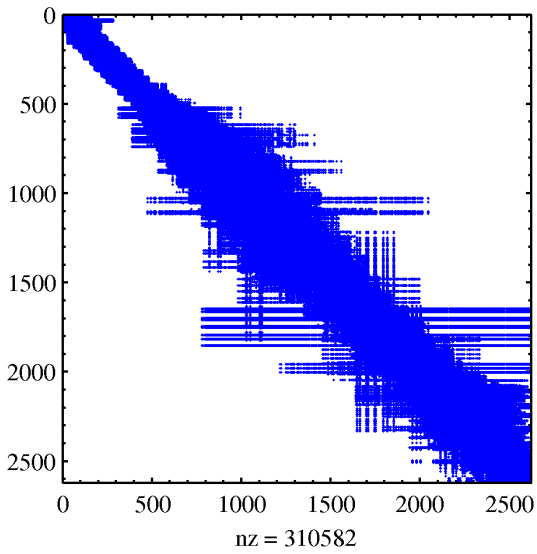

Figure 4. Near-field influence matrix.

MATLAB's `gmres` function was used to solve solve the linear system. `gmres` was provided with a function handle to a C++ MEX-file that returns the matrix-vector product and also returns the sparse near-field

matrix. Using a preconditioner reduced the number of iterations required from ~ 50 to ~ 5, resulting in a decrease in computational time of an order of magnitude.

C. Serial FMM Accelerated BEM Accuracy

A sphere consisting of 855 elements was used as a test case for the FMM solver since the analytical solution is known. The FMM solution to the sphere was compared to the direct BEM solver. Both the FMM and BEM solvers were solved with GMRES[13] to machine precision, and the only difference was in the approximate matrix-vector product of the FMM compared to the machine precision matrix-vector product of the full BEM solver. The potential on the sphere using a $p = 2$ multipole expansion and L_2 norm between the two solutions for increasing values of p are shown beow.

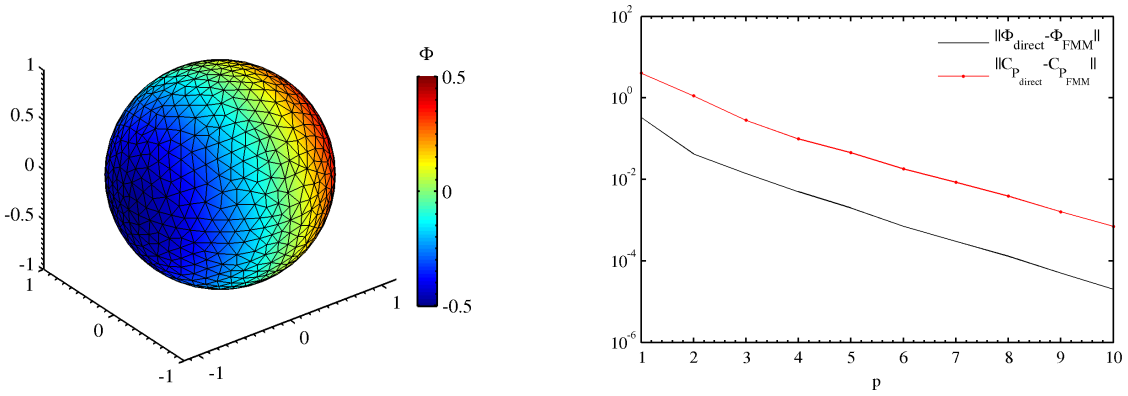

Figure 5. Potential over a sphere for $p = 2$ and $V_\infty = [1, 0, 0]$ (L) and error between the FMM and direct BEM (R).

Even with $p = 2$, the error in potential is small, however the error in pressure is larger. This is due to the fact that the pressure is obtained by differentiation of the potential field, leading to a reduction in pressure accuracy. If a higher pressure accuracy is needed, the multipole order can be increased to obtain the desired fidelity.

D. Parallel Results

A mesh of a falcon buisness jet was used as a test case for the parallel FMM-accelerated BEM implementation. The mesh consisted of 5234 elements and 2619 nodes. The potential distribution obtained using one core and a multipole expansion order of $p = 2$ is shown below:

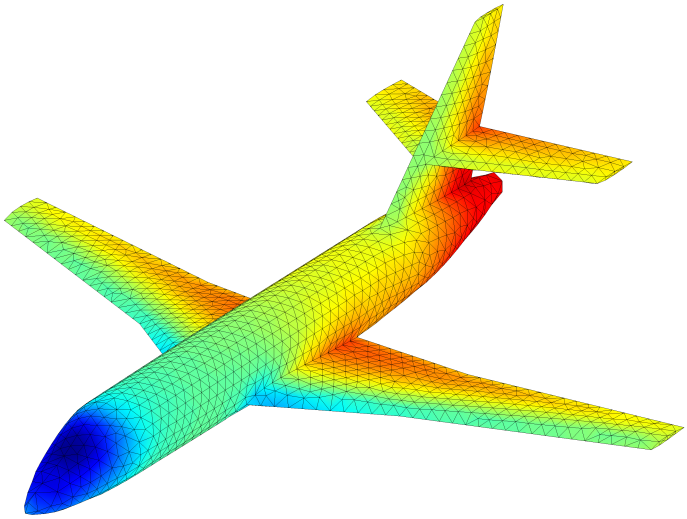

Figure 6. Falcon business jet

Simulations were run on my laptop which has an intel i7-3610QM 2.3GHz quad-core CPU for increasing number of processors and multopole orders. The maximum number of elements allowed in each childless box was set to 10. The code was tested for up to $p = 5$, which is far higher an order than is needed to obtain accurate results. Accurate results are obtained with only $p = 2$. Below is a table comparing the parallel solution time to that of the serial code.

Table 1. Speedup compared to 1CPU

p	1 CPU (s)	2 CPUs	3 CPUs	4 CPUs
1	5.2414	1.24	1.36	1.37
2	8.6618	1.39	1.59	1.70
3	23.8976	1.65	2.04	2.34
4	52.4548	1.73	2.26	2.59
5	105.9322	1.76	2.38	2.79

Table 1 shows that only a slight speed-up is observed for $p = 1$, but as the multipole expansion order increases the parallel efficiency also increases. A maximum speed-up of a factor of 2.8 is realized using a fifth order multipole expansion and 4 CPUs. Some of the multipole translation functions are more easily parallelized than others, and this is likely the reason the performance increase is not constant for all multipole expansion orders. For example, the $L2P$ translation will never have multiple threads trying to read from the same memory address. However, this can occur somewhat frequently in the $M2M$ translation. The implementation could be improved by creating multiple instances of some of the data arrays, but this would also increase the memory required.

E. Large Test Case

The FMM scalability,ideally $O(N)$, was tested by simulating a four-engine jet with a fine discretization. The surface consisted of $\sim 180,000$ elements. The simulation was run on 4 CPUs using a $p = 2$ expansion order, and the resulting potential distribution is shown below:

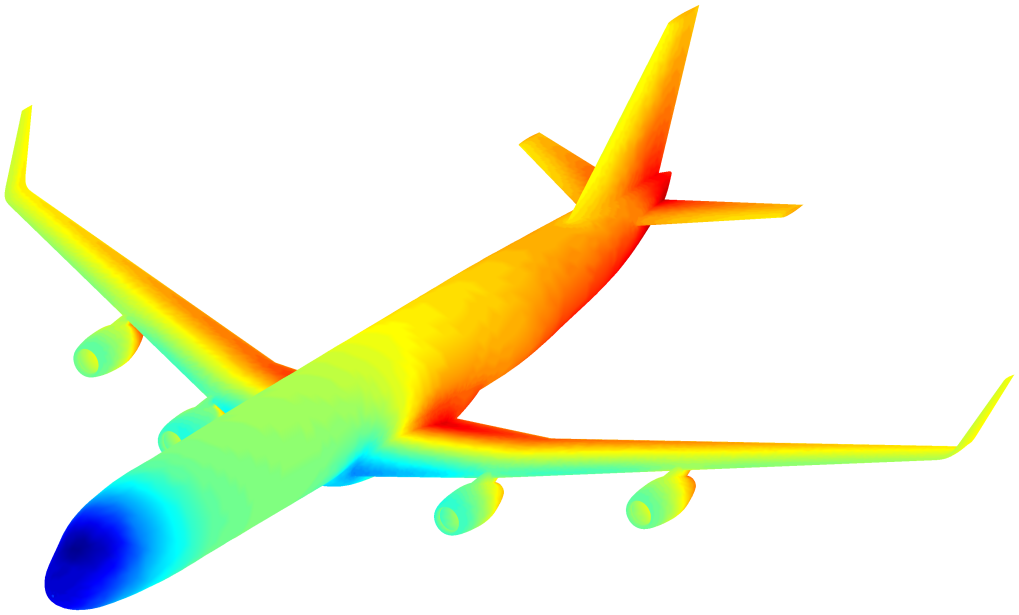

Figure 7. Potential Flow Solution. Runtime time: 4 minutes on 4 CPUs

The mesh for the four-engine case and resulting pressure distribution are shown below.

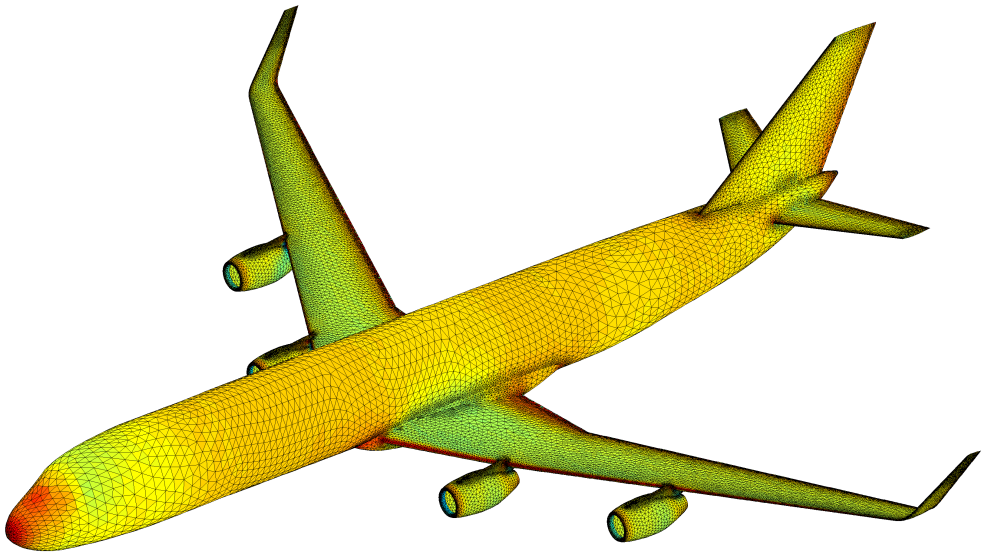

Figure 8. Coefficient of Pressure on surface

The solution time for the four-engine jet was 240s, compared to 5.1 seconds for the falcon mesh using 4 CPUs. The ideal $O(N)$ sped-up would have resulted in a runtime of 175 seconds for the four-engine jet. Therefore, the FMM method did not scale quite as $O(N)$, but came close. The reason it may not have

reached $O(N)$ scaling is that the memory access was more intensive for the larger case, and may have slowed memory reads across the board. Another possibility is that the geometry of the two aircraft, and hence the two octree decompositions, were entirely different from one another.

VII. Conclusions

A high-order panel method has been developed capable of predicting the potential flow about arbitrarily-shaped closed surfaces. The potential flow equation was discretized using the Boundary Element Method, which was accelerated by the Fast Multipole Method. The code was parallelized using OpenMP, resulting in an implementation which can see speed-ups of a factor of 2.8 for large p, and a factor of 2 for moderate expansion orders on 4 processors.

It is unlikely that an ideal linear speed-up will be obtained using OpenMP since it implements shared-memory parallelism. Unless large data arrays are explicitly copied, there will always be times when multiple threads will try to read the same data from memory simultaneously, which decreases performance. It would be interesting to implement the code in Julia to see how a Julia version of the code would scale.

References

[1]Hess, J. and Smith, A., "Calculation of Potential Flow About Arbitrary Bodies," *Progress in Aeronautical Sciences*, Vol. 8, 1967.

[2]Morino, L. and Kuo, C., "Subsonic Potential Aerodynamics for Complex Configurations. A General Theory," *AIAA Journal*, Vol. 12, 1974, pp. 191–197.

[3]Magnus, A. and Epton, A., "PAN AIR- A Computer Program for Predicting Subsonic or Supersonic Linear Potential Flows about Arbitrary Configurations Using a High Order Panel Method," Theory document, nasa cr-3251, 1980.

[4]Ashby, D., Dudley, M., and Iguchi, S., "Development and Validation of an Advanced Low Order Panel Method," NASA Technical Memorandum 101024, NASA Ames Research Center, Moffett Field, CA, Oct. 1988.

[5]H. Gheng, L. G. and Rokhlin, V., "A Fast Adaptive Multipole Algorithm in Three Dimensions," *Journal of Computational Physics*, Vol. 155, 1999, pp. 468–498.

[6]Nabors, K. and White, J., "FastCap: A Multipole Accelerated 3-D Capacitance Extraction Program," *IEEE Transactions on Computer-Aided Design*, Vol. 10, No. 11, 1991, pp. 1447–1459.

[7]Boschitsch, A., Burbishley, T., Quackenbush, T., and Teske, M., "A Fast Method for Potential Flows About Complex Geometries," *34th AIAA Aerospace Sciences Meeting*, 1996.

[8]Vassberg, J. C., "A Fast Surface-Panel Method Capable of Solving Million-Element Problems," *35th AIAA Aerospace Sciences Meeting*, 1997.

[9]Boschitsch, A. and Epstein, R., "Fast Lifting Panel Method," *14th Computational Fluid Dynamics Conference*, 1999.

[10]Winckelsmans, G. and Leonard, A., "Contributions to Vortex Particle Methods for the Computation of Three-Dimensional Incompressible Unsteady Flows," *Journal of Computational Physics*, Vol. 109, 1993, pp. 247–273.

[11]Katz, J. and Plotkin, A., *Low-Speed Aerodynamics*, Cambridge University Press, 2nd ed., 2001.

[12]Newman, J., "Distribution of Sources and Normal Dipoles over a Quadrilateral Panel," *Journal of Engineering Mathematics*, Vol. 20, No. 2, 1986, pp. 113–126.

[13]Saad, Y. and Schultz, M., "GMRES: A Generalized Minimal Residual Algorithm For Solving Nonsymmetric Linear Systems," *SIAM Journal of Scientific and Statistical Computing*, Vol. 9, 1986, pp. 856–869.