On Practical Modifications to the Barnes-Hut Multipole Method for Electromagnetic Scattering

Brian J. Driessen
Sandia National Laboratories, Albuquerque, NM 87185-0439, bjdries@sandia.gov
Joseph D. Kotulski
Sandia National Laboratories, Albuquerque, NM 87185-1152, jdkotul@sandia.gov

Abstract: This paper presents a simple methodology for quickly predicting and optimizing computer run time for the Barnes-Hut multipole method for boundary element electromagnetic scattering problems. The methodology is easily extended to other multipole methods (e.g., Greengard-Rokhlin) and to other physics. The idea is to simply count the number of element-cell interactions, number of direct element-element interactions, and the number of cell multipole expansion creations (each expansion weighted by the number of elements in the cell), and then finally combine these three results with the associated unit costs to obtain the total computer run-time to perform a single matrix-vector multiply. By counting operations instead of actually performing them, the time to predict the computer run time is orders of magnitude smaller than the time to actually perform the associated calculations. This allows for very quick optimization of parameters, such as the maximum number of elements in a final generation cell of the tree. Numerical examples are presented herein in which the rate of return (time saved over time spent finding optimal parameter values) is significantly more than two orders of magnitude.

1. Introduction

Electromagnetic scattering calculations by the boundary element method is one of many applications of multipole methods. The two main multipole methods are the Barnes-Hut method [1] and the Greengard-Rokhlin method [2, 4], which are $O(N \log N)$ and $O(N)$ respectively, where $N$ is the number of "particles" in the problem. For the boundary element method $N$ is the number of elements used to discretize the boundary. The multipole method serves the purpose of providing a close approximation to the matrix-vector product in $O(N \log N)$ or $O(N)$ time, the matrix being the dense one associated with the discretization of the boundary integral equation. This is in contrast to the direct matrix-vector multiply which costs $O(N^2)$. The fact that the matrix-vector product costs $O(N \log N)$ or $O(N)$ time together with the fact that the condition number of the matrix is $O(1)$ gives us the ability to iteratively solve the linear system in $O(N \log N)$ or $O(N)$ time.

In the following sections we will present and demonstrate a simple method of quickly predicting the time to calculate the matrix-vector product by the multipole method. The idea of the method is simple and is based upon simply counting the number of operations that would occur if one were actually calculating the product. This allows one to quickly find optimal values of the parameters so as to minimize the computer run-time for the actual calculations. The method is described in Section 2, numerical examples are provided in Section 3 to demonstrate the utility of the approach, and finally conclusions are given in Section 4.

2. The Method

We first describe a variant of the Barnes-Hut multipole method that uses a variable order of the multipole expansion. Each generation of the tree has its own order of expansion. This order is determined by the cell-size criterion that can be found in [4], namely,

$$p = k \rho + \ln(\pi + k \rho)$$

(2.1)
DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, make any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
where $p$ is the distance from the center of a cell, $k$ is the wavenumber, and $p$ is required order of expansion. For cubical cells, with edge length $a$, the maximum value of $p$ for a cell is $p_{\max} = (\sqrt{3}/2)a$. Thus (2.1) translates to

$$p = -\frac{\sqrt{3}}{2}ka + \ln(\pi + \frac{\sqrt{3}}{2}ka)$$

(2.2)

The obvious utility in using a different order of expansion for each generation of the tree is that cells of finer generations of the tree are smaller and do not require as large an expansion order as do cells of coarser generations. Moreover, the number of finer generation cells is relatively large so that the cost of using excessively large expansion orders for these cells would be significant. We note that if the initial domain is a cube with edge length $L$, then the edge length $a_n$ of the $n^{th}$ generation cells is $L(1/2)^n$. Then (2.2) gives us the associated expansion order.

We propose herein a simple and nearly costless method of predicting the total time to perform a matrix-vector multiply using the Barnes-Hut multipole method. The basic methodology extends easily to the Greengard-Rokhlin approach as well. The method is based upon the following simple observation. The total run-time of the matrix-vector multiply can be calculated by simply counting the number of times each operation occurs without actually performing any of the associated calculations; then, these counters can be used in a weighted sum where the weights are the time to perform one of each such operation. For example, to calculate the total time to perform element-cell interactions for the $m^{th}$ generation of the Barnes-Hut tree [1], we simply multiply the number of such interactions by the time it takes to perform one such operation. The other costs in the matrix-vector multiply are the costs to create cell expansions and the cost to perform direct element-element interactions. Similarly the total number of each such operation can be simply tallied and the resulting count can then be multiplied by the unit cost to obtain the total cost. The time to update these counters as one moves through the cells of the tree is orders of magnitude smaller than the time to actually perform all of the associated calculations.

The above approach allows us to predict very quickly how the total run time will vary as the maximum number of elements per cell allowed in a final generation cell is varied. This maximum number of cells will be denoted herein by $n_{\text{max}}$. This prediction takes only a small fraction, as we will see in Section 3, of the time it takes to actually perform the matrix-vector multiplies, thus allowing us to minimize the total computer run time with respect to the parameter $n_{\text{max}}$ before actually launching the computer job, while the time to calculate the optimal $n_{\text{max}}$ is virtually trivial. Then by using the optimal value, tremendous rates of return on time spent are obtained, as we will see in Section 3.

In the example electromagnetic scattering problems considered herein in Section 3 while using the variable order of expansion approach discussed above, the total computer run time will be predicted and the optimal $n_{\text{max}}$ value obtained in a small fraction of the time to perform one matrix vector multiply. When there are 100s or 1000s of such matrix-vector multiplies (one for each iteration of a GMRES algorithm [3]), the cost to obtain the optimal value of $n_{\text{max}}$ is virtually zero.

3. Numerical Examples

We will consider two cases of electromagnetic scattering for a cone whose surface is meshed with 5823 and 10080 triangle elements, respectively. The wave number is 20.9. For both of these problem cases, we will present and discuss how the computer run-time varies with $n_{\text{max}}$ and also present and discuss the trivial computer run times required to quickly predict this variation of run time with $n_{\text{max}}$.

First the dependence of the different operation costs upon the expansion order $p$ must be created. Using only a small number of $p$ values and linear interpolation, this dependence can be quickly established. Figure 1 below illustrates how the cost to perform an element-cell interaction varies with $p$. The circles in Figure 1 denote the data points and the lines the linear interpolation.
Likewise the unit costs for creating a cell’s multipole expansion is shown in Figure 2 below.

The cost to perform one direct element-element interaction can be easily obtained. This value is $2.849 \times 10^{-4}$ seconds. The associated total cost is simply the total number of direct interactions times this value.

Figure 3 below shows the results of the timing prediction for $n_{\text{max}}$ values ranging from 1 to 201, with an increment of 10. It took only 17 seconds to generate these 21 values, for an average of less than one second of prediction time per value of $n_{\text{max}}$. Again this is a result of the fact that we are simply counting the number of operations without performing any of them.

From Figure 3, we find that the optimal value of $n_{\text{max}}$ is $n_{\text{max}} = 70$, which produces a matrix-vector multiply cost of 70 seconds, which is a factor of 1.6 smaller than the cost for $n_{\text{max}} = 1$. Suppose there were 100 GMRES iterations. Then the savings is 1.2 hours (out of 3.1 hours) while the time lost to obtain the optimal value of $n_{\text{max}} = 70$ was only 17 seconds. The rate of return for time spent (time saved over time spent finding the optimal value of $n_{\text{max}} = 70$) is therefore 253. In other words, by spending 17 seconds, we saved 1.2 hours. Likewise, if the number of GMRES iterations were 1000, spending 17 seconds saved us 12 hours (out of 31 hours) for a rate of return of 2530.

The same timing study was conducted for the 10,080 element cone. Figure 4 below shows the time for a single matrix-vector multiply versus $n_{\text{max}}$. 
Figure 3. Variation in Run-Time for One Matrix-Vector Multiply Versus Maximum Number of Elements in a Final Generation Cell, 10080 Element Cone Problem

From Figure 4 we see that the optimal value of $n_{max}$ is $n_{max} = 50$. The total time to create the 21 points in Figure 4 was 38 seconds. Thus the average time to predict a run-time was less than 2 seconds. The time for $n_{max} = 1$ is 211 seconds and that for $n_{max} = 50$ is 164. Thus, for 100 GMRES iterations, 1.3 hours (out of 5.8 hours) is saved for a rate of return of 124. For 1000 GMRES iterations, 13 hours (out of 58 hours) is saved for a rate of return of 1240.

Since we have at our disposal timing studies for two levels of discretization of the body’s surface, it is worthwhile to empirically check the anticipated $O(N \log N)$ complexity of the Barnes-Hut algorithm, where $N$ denotes the number of triangle elements of the surface mesh. Letting $T_{5823}$ and $T_{10800}$ denote the computer run times for the two respective meshes, the theoretical complexity predicts

$$\left( \frac{T_{10800}}{T_{5823}} \right)_{\text{theoretical}} = 1.84$$

(3.1)

and the actual value for $n_{max} = 1$ is 1.87, differing only by 1.5%. This check appears to show consistency with the theoretical complexity. However, more values of $N$ should of course be checked for thoroughness.

We note that the method presented in this paper also allows one to conduct empirical complexity studies in a fraction of the time that actually calculating the associated complete matrix-vector products would take.

4. Conclusions

This work presented and demonstrated a simple method for quickly predicting computer run-times for the Barnes-Hut multipole method for boundary element electromagnetic scattering problems. The method is based upon simply counting the number of element-cell interactions, direct element-element interactions, and cell multipole-expansion collapses that would occur without actually calculating any of them. Numerical examples demonstrated that this approach reduces the time to predict the total computer run time to a small fraction of the time required to actually perform the matrix-vector multiply. As demonstrated, this speed of prediction enables one to quickly find optimal parameter values and achieve rates of return (time saved over time spent in optimizing parameters) significantly greater than two orders of magnitude.

References


