# Implementation of Adaptive Cross Approximation on NVIDIA GPUs

Tyler N. Killian, Daniel L. Faircloth, R. Todd Lee, and James G. Maloney
Signature Technology Laboratory, Georgia Tech Research Institute
Atlanta, GA 30318, USA

PROBLEM. SOLVED.

## Introduction

- The Method of Moments (MoM) has been a popular choice among engineers for solving electrodynamic scattering problems for many years now.
- Here we discuss acceleration of the MoM scattering problem for conducting bodies. The governing integral equations are the Electric Field Integral Equation (EFIE) given by

$$\vec{E}^i(\vec{r})_{\tan} = \left[ j\omega \vec{A}(\vec{r}) + \nabla\Phi(\vec{r}) \right]_{\tan}$$

and the Magnetic Field Integral Equation given by

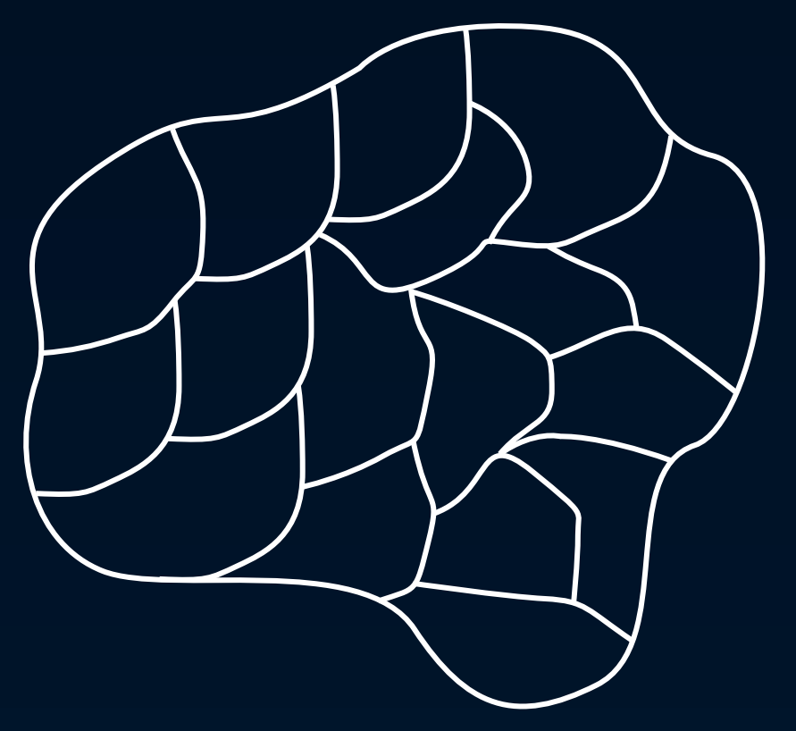
$$\hat{n} \times \vec{H}_i(\vec{r}) = \frac{\vec{J}(\vec{r})}{2} - \hat{n} \times \int_{S_c} \vec{J}(\vec{r}') \times \nabla' G(\vec{r},\vec{r}')\, dS', \quad \vec{r} \in S_c$$

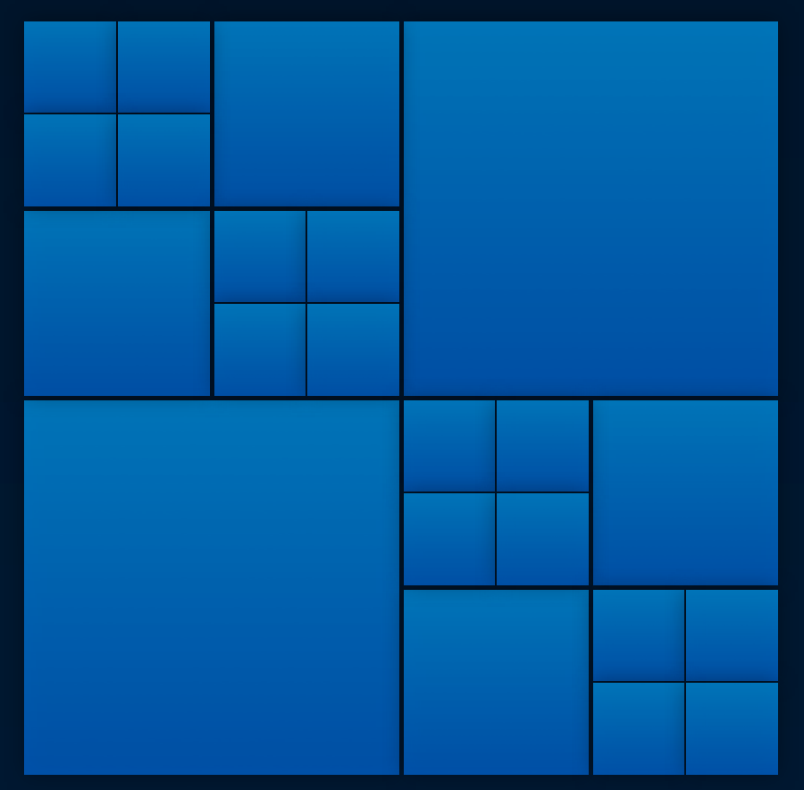
which can be combined to form the Combined Field Integral Equation

$$\alpha \text{EFIE} + \eta(1-\alpha)\,\text{MFIE}$$

- In traditional formulations, the solution of a dense linear system is required. The matrix fill time is $O(N^2)$ while the solution time is $O(N^3)$.
- Recently, several fast methods have been proposed to reduce the complexity of the fill and solve times such as Fast Multipole Method (FMM), Adaptive Integral Method (AIM), and Adaptive Cross Approximation (ACA).
- These methods can reduce solution complexity as low as $O(N\log N)$.
- GPU acceleration of these methods yields substantial gains in time now making it practical to solve very large systems on personal workstations.
- Here we propose a method for applying GPUs to the acceleration of the ACA including a novel procedure for direct solution of the system of equations via a compressed LDU decomposition.

## Overview of Adaptive Cross Approximation





- Matrix compression algorithm for matrices exhibiting low-rank sub-blocks
- Blocks formed by creating disjoint groups of basis functions
- Compressibility determined primarily by distance
- Blocks are compressed using incomplete pivoted LU decompostion
- Compression can be >99%
- Compressed form also yields reduced operation counts for *GEMV, *GEMM type operations
- Partitioned matrix due to grouping

$$Z = \left[\begin{array}{cc|cc} Z_{11} & Z_{21} & Z_{13} & Z_{14} \\ Z_{12} & Z_{22} & Z_{23} & Z_{24} \\ \hline Z_{13} & Z_{32} & Z_{33} & Z_{34} \\ Z_{14} & Z_{42} & Z_{43} & Z_{44} \end{array}\right]$$

- Low rank matrices can be compressed as

$$Z_{G2G1} = UV \longrightarrow U^{m\times k},\ V^{k\times n} \qquad k \ll [m,n]$$

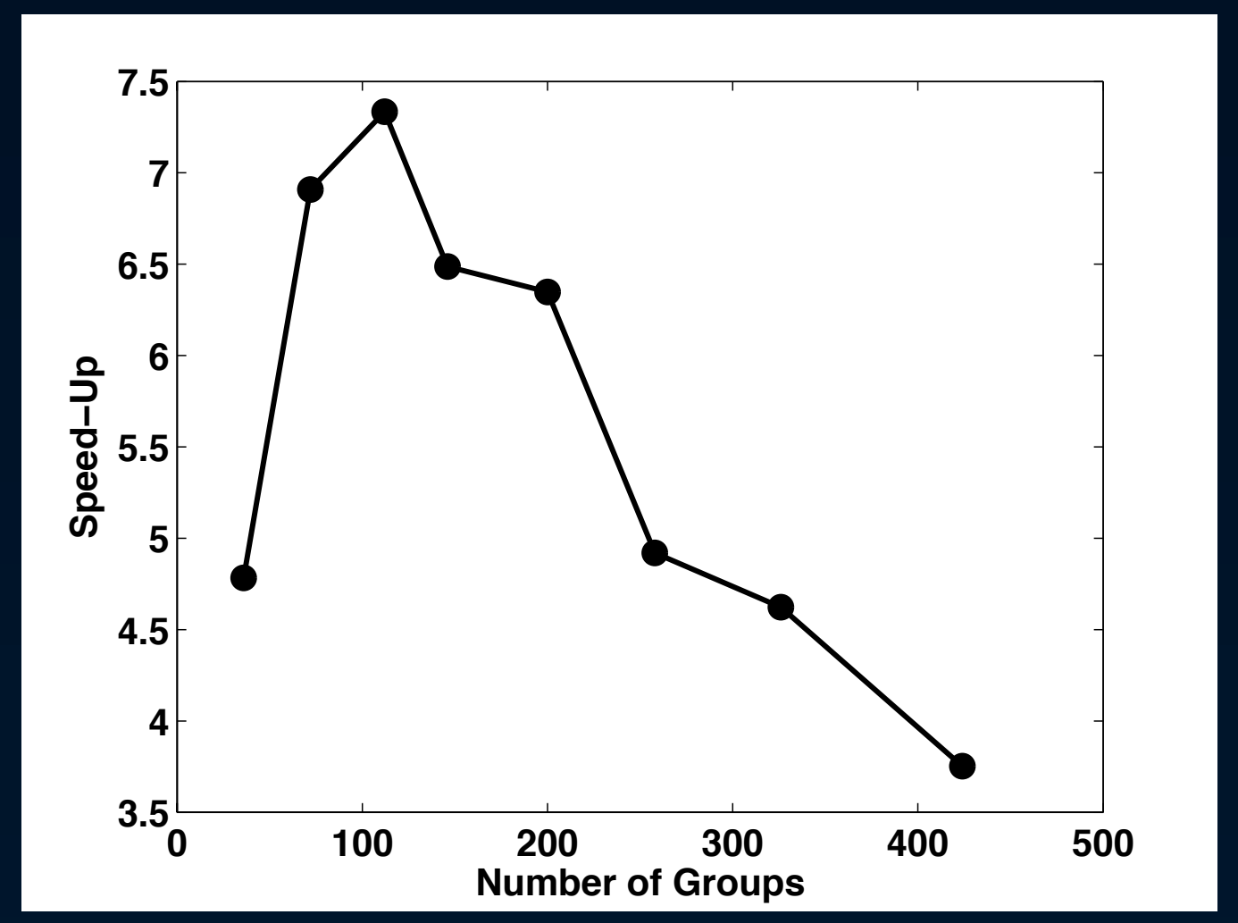
## GPU Acceleration of ACA Matrix Fill

- Goal is to accelerate matrix fill/compression while storing full compressed matrix in host memory
- ACA algorithm done on CPU
- On CPU, primary computation time is row and column computation
- Row/column generation done on GPU
- Calculate and ship back to CPU
- One thread per matrix element
- ACA requires many vector dot products, norms, abs, max value search, etc.
- Do this on CPU due to communication cost and small data size
- Favor computation over communication
- Each thread calculates necessary triangle areas, edge lengths, basis transformations, etc.
- Avoids multiple memory reads
- Unstructured meshes lead to difficulties in ensuring coalesced reads

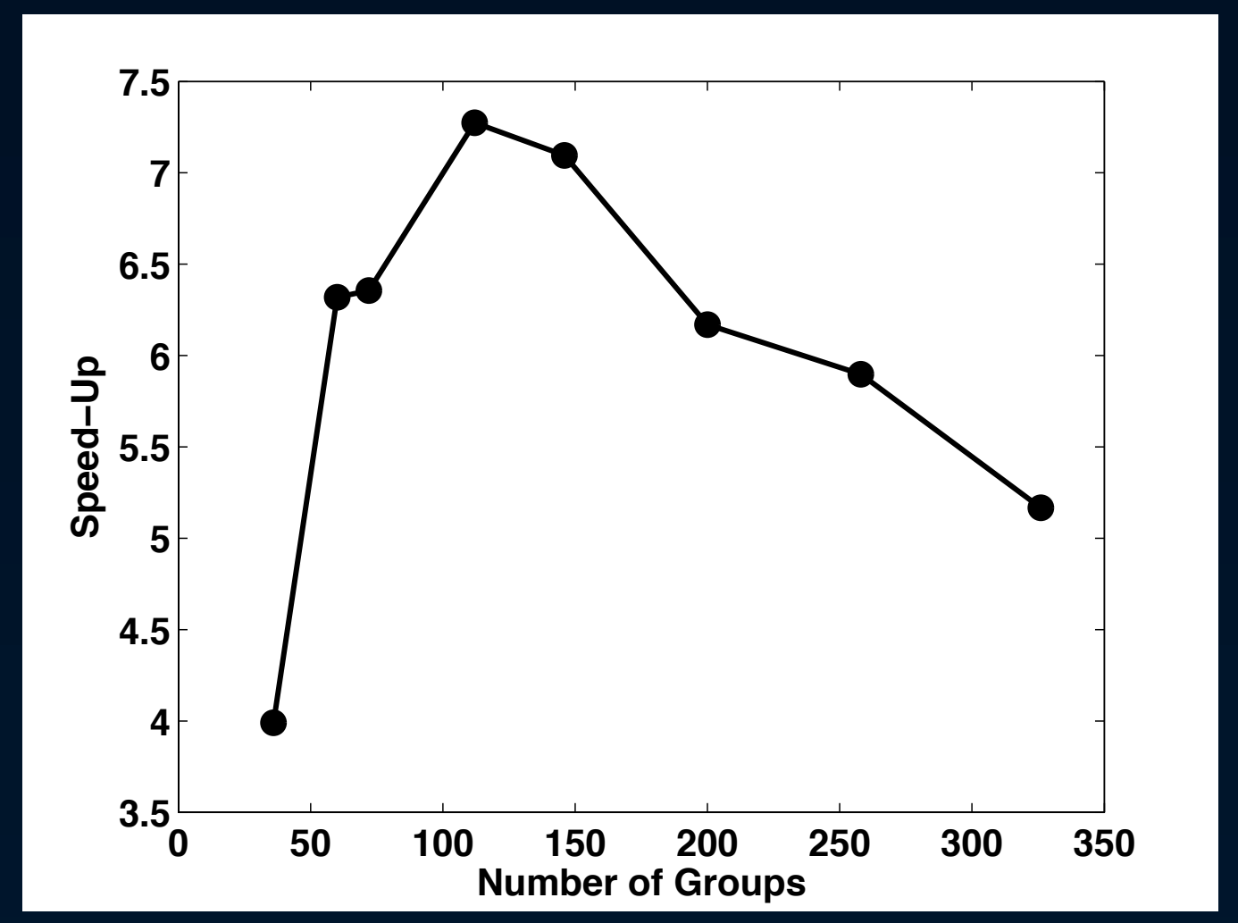
## ACA Matrix Fill Results

- Comparisons based on Core i7 920 with ATLAS BLAS+LAPACK and NVIDIA GeForce GTX 480 (Fermi architecture) using CUDA 3.1
- It should be noted that variation in average group size for the ACA may yield different speed-up results. Larger groups result in less communication overhead for the GPU whereas this is not such a concern for the CPU.
- Results for a 1m radius sphere illuminated by a plane wave. Results are speed-up for matrix fill only.

975 MHz - 48741 Unknowns

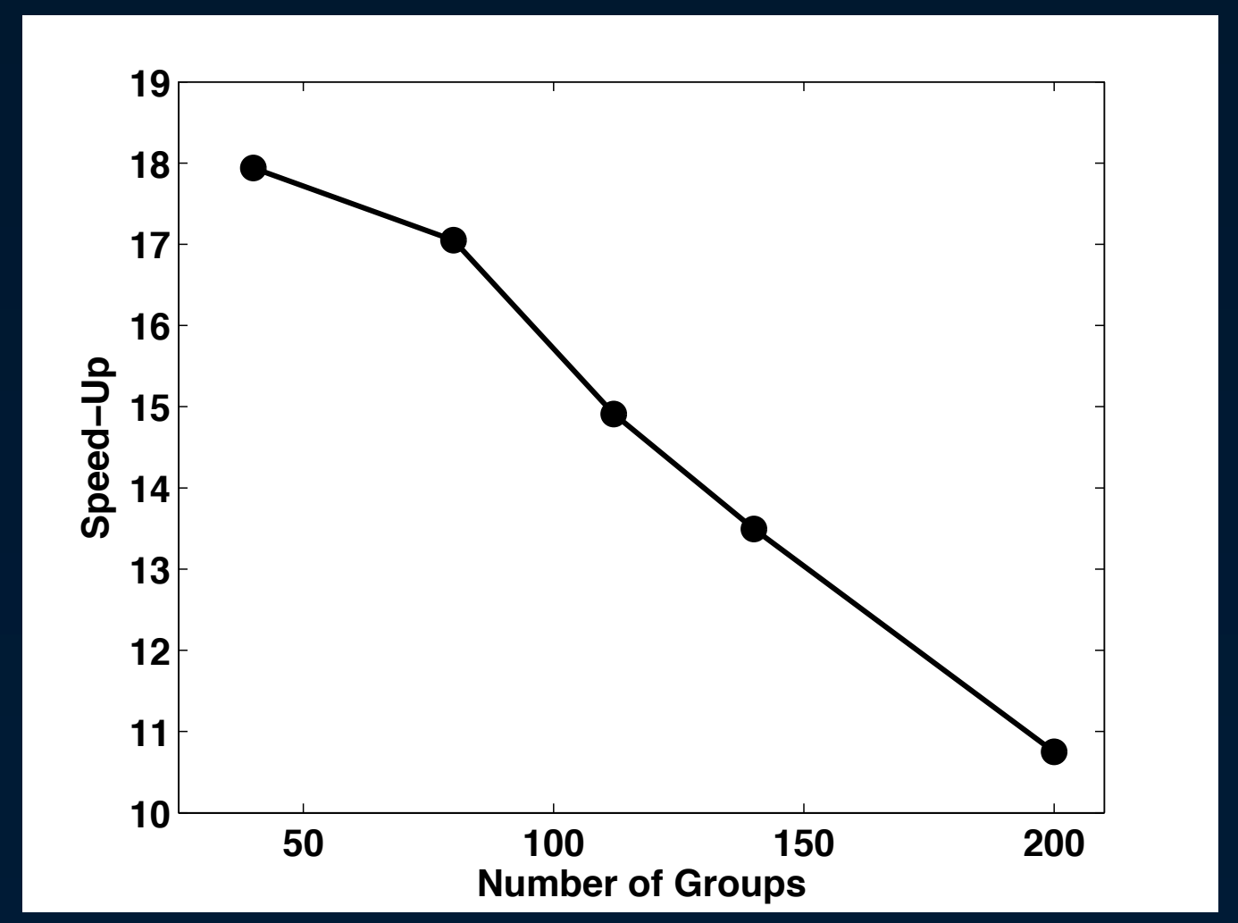


1075 MHz - 59508 Unknowns



- For spheres, maximum speed-up is achieved for ~120 groups at both frequencies.
- Results for a cylinder (shown at right) illustrate differences compressibility and how this affects GPU performance versus CPU.
- Cylinder geometry is "less-coupled" than sphere, and so GPU benefits from larger group sizes (fewer groups) with maximum speed-up ~25 groups.

1m Radius - 35m Length
300 MHz - 78618 Unknowns



## GPU Acceleration of ACA-based LDU Decomposition

- ACA matrix fill typically followed by iterative solution
- Unacceptable for applications involving many right hand sides (RCS)
- Using ACA algorithm, possible to form compressed LDU decomposition
- Block Equations

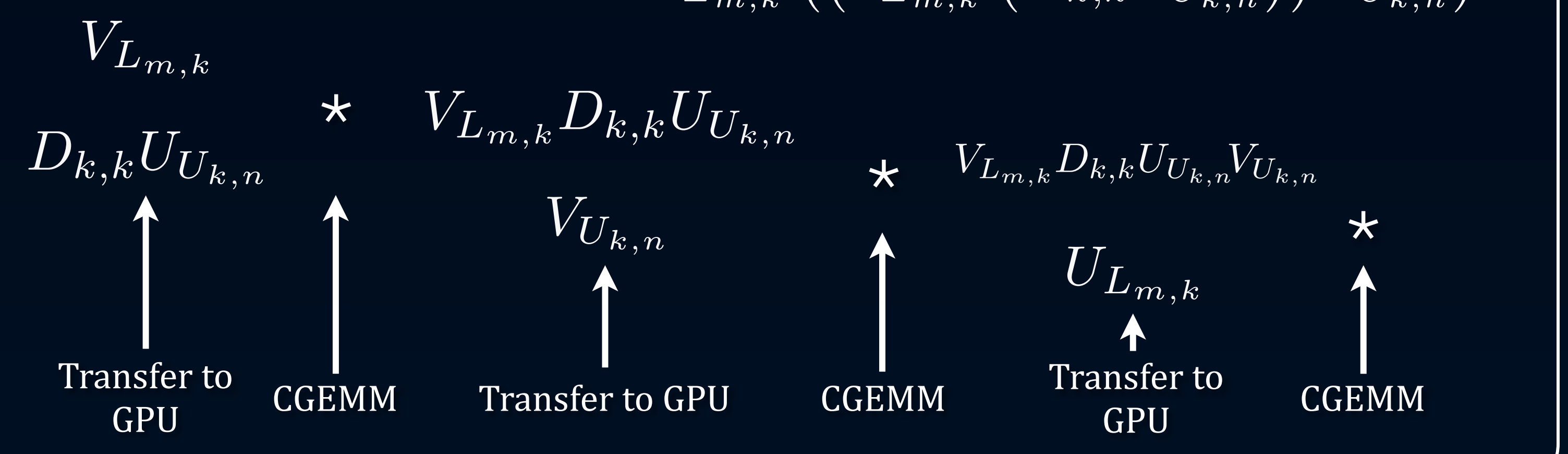
$$D_{m,m} = Z_{m,m} - \sum_{k=1}^{m-1} L_{m,k} D_{k,k} U_{k,m}$$

$$L_{m,n} = \left[ Z_{m,n} - \sum_{k=1}^{n-1} L_{m,k} D_{k,k} U_{k,n} \right] D_{n,n}^{-1} \qquad U_{m,n} = D_{m,m}^{-1} \left[ Z_{m,n} - \sum_{k=1}^{m-1} L_{m,k} D_{k,k} U_{k,n} \right]$$

- Computation dominated by summation term
- Compressed matrix representation

$$L_{m,k} D_{k,k} U_{k,n} = U_{L_{m,k}} V_{L_{m,k}} D_{k,k} U_{U_{k,n}} V_{U_{k,n}}$$

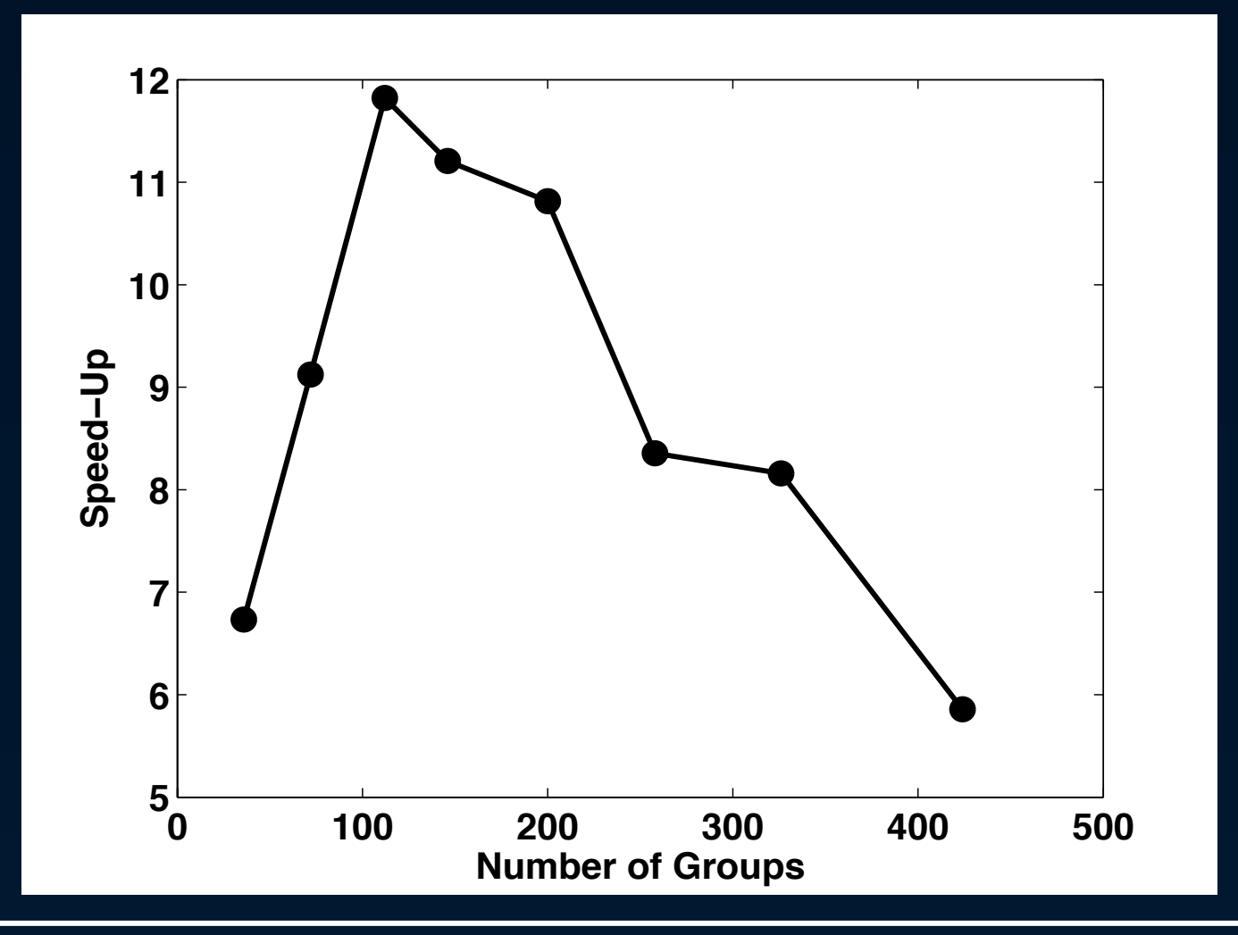
- CPU approach relies on matrix vector multiplies to form rows and columns for ACA in order to minimize memory and computation complexity.
- CGEMM better "bang for the buck" than CGEMV in terms of GFLOPs
- Can we just form the fully decompressed L or U blocks on GPU, ship back to CPU, and run ACA algorithm by picking and choosing rows/columns as needed?
- Minimum operations count: $U_{L_{m,k}} \left( \left( V_{L_{m,k}} \left( D_{k,k} U_{U_{k,n}} \right) \right) V_{U_{k,n}} \right)$



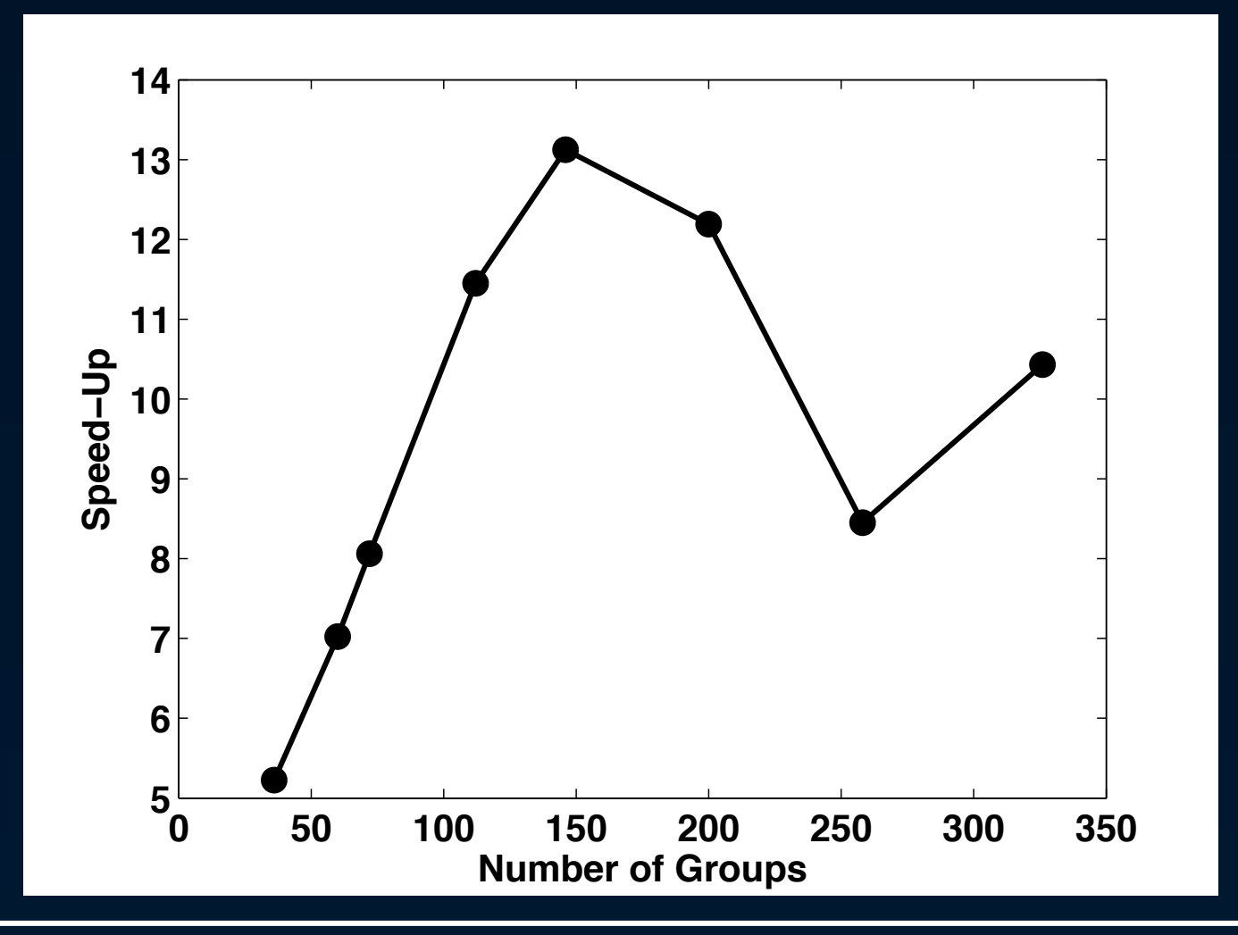
## ACA LDU Decomposition Results

- Maximum speed-up for LDU decomposition of 1m radius sphere occurs ~120-150 groups. This is similar to matrix fill case although speed-up is much higher in the case of LDU decomposition.

975 MHz - 48741 Unknowns



1075 MHz - 59508 Unknowns



## Conclusions

- GPU acceleration of ACA MoM makes solution of "real world" problems practical on personal workstations.
- As an example, the notional cruise missile has 118242 unknowns, ACA+GPU fill: 15 mins 25 secs, ACA+GPU LDU: 35 mins 20 secs