# CSC 548: Project Report
# CUDA Acceleration for BoomerAMG Implementation

http://www4.ncsu.edu/~ksivara/amg/index.html

Poonam Shidlyali
pashidly@ncsu.edu

Karthikeyan Sivaraj
ksivara@ncsu.edu

Keerthana Boloor
kboloor@ncsu.edu

## Introduction

AMG is a parallel algebraic multigrid solver for linear systems arising from problems on unstructured grids. The driver provided for this benchmark builds linear systems for various 3D problems from the given input, which is then solved by the AMG solver. It uses the HYPRE library(http://acts.nersc.gov/hypre/) built at Center for Applied Scientific Computing at Lawrence Livermore National Laboratory.

In this project, we have ported portions of the code in the multigrid solver to the NVIDIA CUDA platform. By identifying computational hotspots within the multigrid solver code and porting them to CUDA, we have taken advantage of the parallel processing capabilities of the GPU, which is able to run thousands of GPU threads simultaneously. Here, we discuss our approach, coding efforts and the performance results.

*Note: In this report we have indicated contributions of team members in relevant sections.

## Design/Approach

**Identification of hotspots:**

We have identified the functions within the solver that take up the most execution time using the gprof tool.

**Data and execution model:**

The data in this solver is partitioned according to the number of processors and processor topology, and each process works on its own chunk of data. The code also uses OpenMP regions and solves computation intensive regions with multiple threads.

**Our solution:**

We have ported these openmp regions along with some other normal loops that consume time. Hence, we replace openmp threads with GPU threads. In the previous report we have indicated some of the hotspots and issues that could be faced while porting. The next section contains the overview of the porting some of the hotspots.

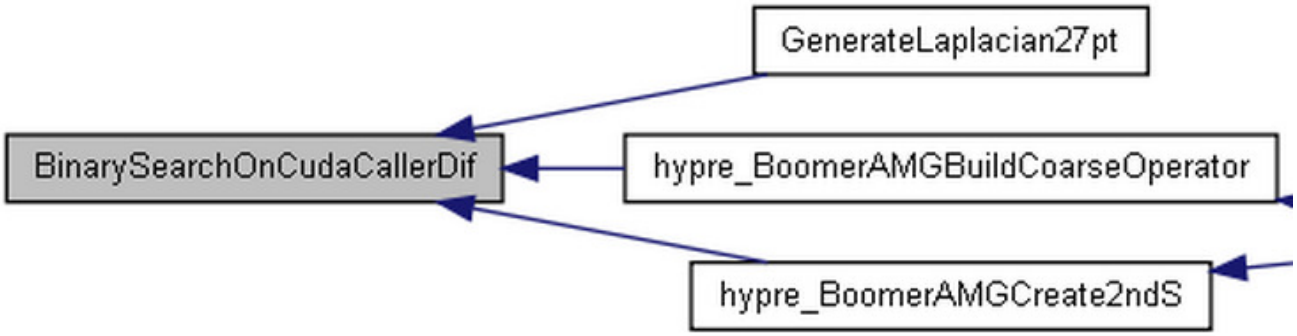
## Implementation

Here is a list of hotspots ported to CUDA:

1. hypre_BoomerAMGBuildInterp in par_interp.c


coarseshiftinterp contains the cuda calls.
**Ported by Keerthana**

2. GenerateLaplacian27pt in file par_laplacian_27pt.



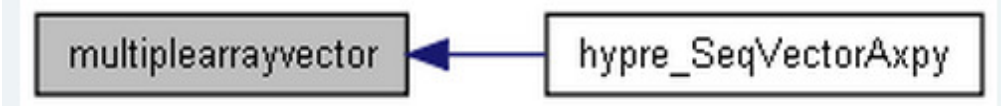

**Ported by Poonam**
The function BinarySearchOnCudaCallerDif contains the cuda calls.

3. hypre_SeqVectorInnerProd


We found that the above function was called frequently by the other two hotspots hypre_BoomerAMGCycle and Hypre_ParCSRMatrix repeatedly and so we ported this call to Cuda.

**Ported by Karthik**

4. hypre_BoomerAMGRelax




This is another function from the file seq_mv/vector.c which was ported to CUDA due to the large number of times it was called from the source of the hotspot.

**Ported by Poonam**

5.hypre_ParCSRMatrixMatvec


This function was a hotspot when run with 27pt and which has been reduced due to porting with CUDA.
**Ported by Keerthana**

Please refer to our website for more functions ported.

## Test cases

1. Verify that AMG with CUDA works
    1. We ran the code with CUDA with 1,4 and 8 processors. We could not run for higher number of unique processors as the number of machines in the cuda cluster available to us was 15.

2. Execution times
    1. We obtained the execution times for the PCG setup and PCG solvers individually with CUDA and compared with those when run without CUDA for "-laplace and -laplace27pt

3. CPU to GPU – GPU to CPU memory transfer time calculation
    1. We measured memory transfer times of the hotspot functions in CUDA. We found that memory transfer takes the maximum time in CUDA execution and the decrease in performance is caused by the large memory transfers to and from CUDA (shown in results).

4. Hotspot removal verification
    1. We ran gprof with CUDA to check if there has been a difference in the hotspots generated by the code. Some hotspots have moved to the lower functions calling CUDA when run with laplace27pt and laplace due to the increase in memory transfer times and we can see a removal of the hotspots when run with "jumps" option due to improvement in performance when run with 8 processors.

## Results

The results of the tests are as shown.
The main comparison is of performance variation with using CUDA.

| Prob with number of nodes | Before Porting PCG Setup (Wallclock in sec) | After Porting PCG Setup (Wallclock in sec) | Before Porting PCG Solver (Wallclock in sec) | After Porting PCG Solver (Wallclock in sec) |
|---|---|---|---|---|
| **Laplace** | | | | |
| **1** | 0.015625 | 0.054688 | 0.007812 | 0.035156 |
| **4** | 0.128906 | 1.179688 | 0.058594 | 0.183594 |
| **8** | 0.300781 | 1.289062 | 0.054688 | 0.71875 |
| **Laplace 27pt** | | | | |
| **1** | 0.023438 | 0.058594 | 0.007812 | 0.039062 |
| **4** | 0.117188 | 2.570000 | 0.042969 | 0.332031 |
| **8** | 0.347656 | 0.769531 | 0.183594 | 0.76953 |
| **Jumps** | | | | |
| **1** | 0.011719 | 0.050781 | 0.003906 | 0.042969 |
| **4** | 0.144531 | 1.996094 | 0.566406 | 3.332031 |
| **8** | 0.300781 | 0.570312 | 2.191406 | 1.687500 |

The above table shows the performance of AMG before and after porting it to CUDA. The measurements have been done for all three problem types Laplace, Laplace 27pt and Jumps. We found that with jumps there is a slight improvement with using CUDA. The reason we believe this is because of the decrease in the memory transfer sizes due to sparse matrices when run with the "Jumps" option. There is performance degradation in "Laplace" and "Laplace27pt" option. We found the amount of time the calls took at the modified hotspots and separated the time run inside cuda and the memory transfer time.

The graph depicts the time for coarseshiftinterp (hotspot 1) function when coded on host and with CUDA. It also shows the time spent in allocating and releasing memory on the GPU device. X-axis shows number of iterations for which the function call is made.

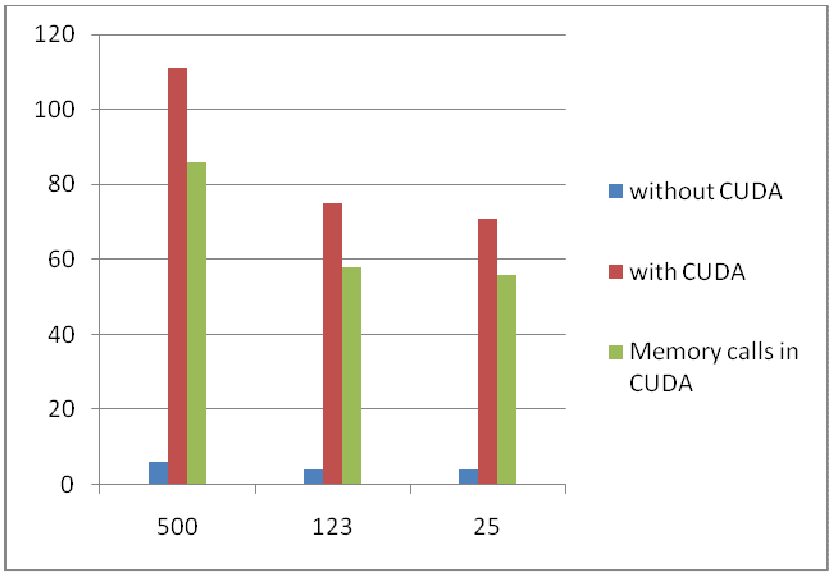



*Figure* 1 Execution Times (sec) for the coarseshiftinterp function

We have also ported the Binary search on CUDA (hotspot 2). The results below shows the wallclock times of the function call before and after porting to CUDA.

| Iterations | without CUDA | with CUDA |
|---|---|---|
| 100 | 8 | 425 |
| 50 | 5 | 213 |

Inner Product of matrix is also ported on CUDA (hotspot 3). For 1000 iterations, kernel call takes 664microsec as compared to 3microsec on the host machine.

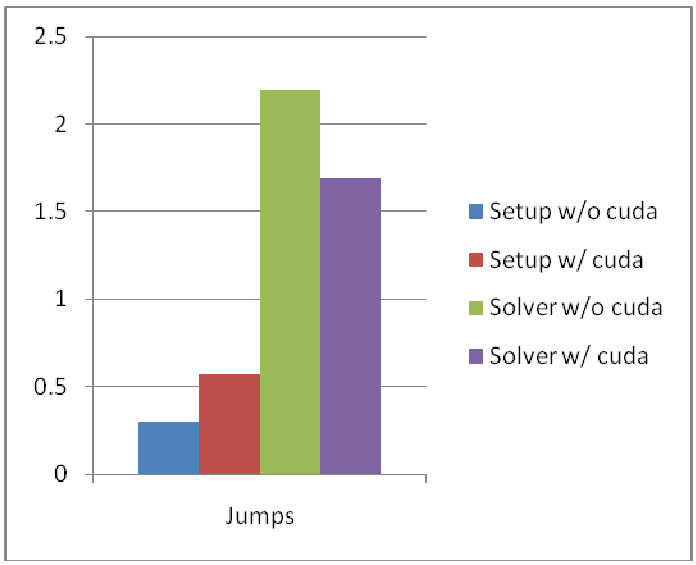



*Figure* 2 Execution Times (sec) for AMG for Jumps

*Run by: Poonam*

The graph above shows the execution times (Wallclock) for the Jumps problem before and after porting the AMG code to CUDA.

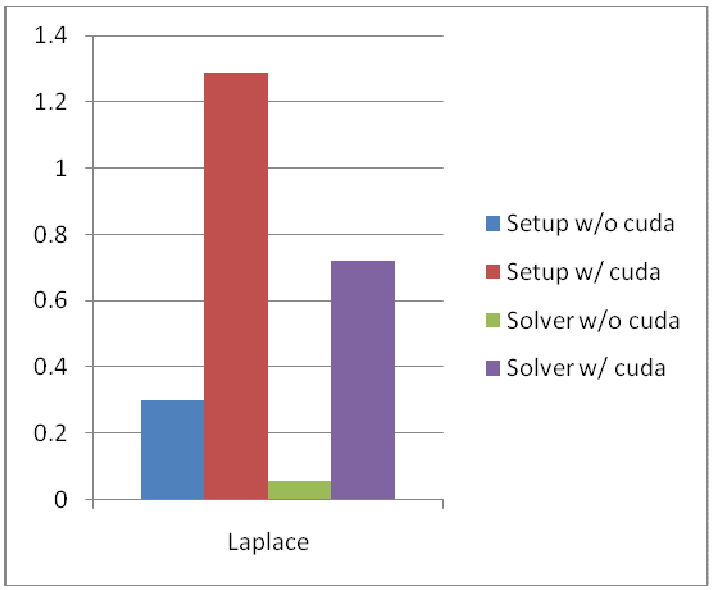



*Figure 3* Execution Times (sec) for AMG for Laplace

*Run by:Keerthana*

The graph above shows the execution times (wallclock) for the Laplace problem before and after porting the AMG code to CUDA.

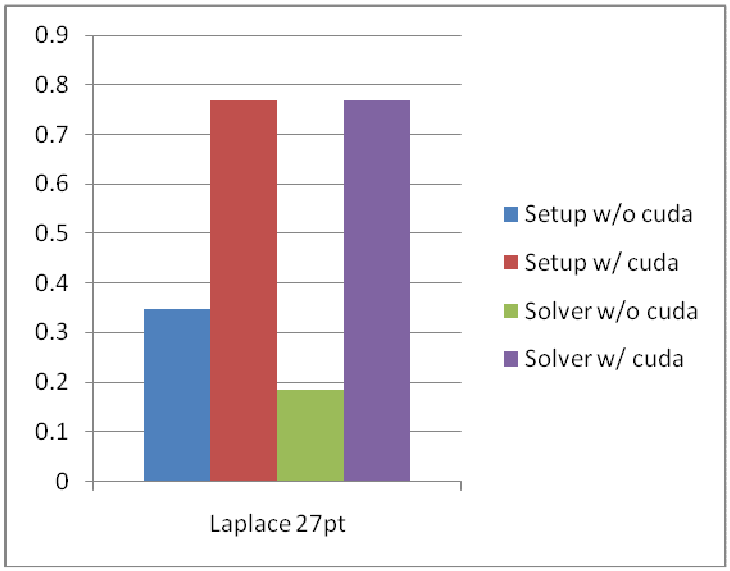



*Figure* 4 Execution Times (sec) for AMG for Laplace 27pt

Run by: Karthik

The graph above shows the execution times (wallclock) for the Laplace27pt problem before and after porting the AMG code to CUDA.

## Future work

1. Understanding the matrix dataset requirements for migration to CUDA for "-laplace" and "-laplace27pt" options of the AMG solver.

As shown in the above results, we observe a large overhead in memory transfers between CPU and GPU in functions like "coarseshiftinterp" in "-laplace functions. We need to figure out the optimum matrix dataset and input matrix for the "-laplace" and "-laplace27pt" operations for improved CUDA performance.

2. Compare performance with varying CUDA threads and block numbers

Once we find the optimum input set for improved performance with CUDA, we need to figure out the relation between the CUDA configuration and the input set dimensions.

3. Modifications to the AMG program model to suit CUDA requirements.
    1. No more dynamic memory allocations in threads
    2. Improved coherence to existing data sets / increased in memory operations in stack instead of using the heap frequently.
    3. Reduction in nested input dimensions (grid dimensions in the current AMG code are modeled in such a way that the space for data can be created with pointer heads part of another matrix with several levels of indirection). We need to change the program model to suit the CUDA implementation. [6] Illustrates the need and provides a starting step for this kind of porting.

**References**

[1] Van Emden Henson and Ulrike Meier Yang, "BoomerAMG: A Parallel Algebraic Multigrid Solver and Preconditioner", Appl. Num. Math. 41 (2002), pp. 155-177. Also available as LLNL technical report UCRL-JC-141495.

[2] Hans De Sterck, Ulrike Meier Yang and Jeffrey Heys, "Reducing Complexity in Parallel Algebraic Multigrid Preconditioners", SIAM Journal on Matrix Analysis and Applications 27 (2006), pp. 1019-1039. Also available as LLNL technical reports UCRL-JRNL-206780.

[3] Hans De Sterck, Robert D. Falgout, Josh W. Nolting and Ulrike Meier Yang, "Distance-Two Interpolation for Parallel Algebraic Multigrid", Numerical Linear Algebra with Applications 15 (2008), pp. 115-139. Also available as LLNL technical report UCRL-JRNL-230844.

[4] Hypre library (http://acts.nersc.gov/hypre/)

[5] ASC Sequoia benchmark (https://asc.llnl.gov/sequoia/benchmarks/#amg)

[6] Dominik Goddeke and Robert Strzodka and Jamaludin Mohd-Yusof and Patrick McCormick and Hilmar Wobker and Christian Becker and Stefan Turek, “Using GPUs to Improve Multigrid Solver Performance on a Cluster”, International Journal of Computational Science and Engineering (IJCSE)(2008), pp36-55 v4.