Accelerating Geophysics Simulation using CUDA

Brandon Holt and Daniel Ernst
holtbg@uwec.edu - ernstdj@uwec.edu
Department of Computer Science ● University of Wisconsin – Eau Claire

ABSTRACT

CitcomS, a finite element code modeling convection in the Earth's mantle, is used by many computational geophysicists to study the Earth's interior. In order to generate more accurate results, finer spatial resolutions are needed, so there is a constant push to increase the performance of the code to allow for more computations to complete in the same amount of time. In order to accomplish this, we leverage the massively parallel capabilities of graphics processors (GPUs), specifically those using nVidia's CUDA framework. We have begun translating existing functions to run in parallel on the GPU, starting with the functions where the most computing time is spent. Running on nVidia Tesla GPUs, initial results show an average speedup of 1.8 that stays constant with increasing problem sizes and scales with increasing numbers of MPI processes. As more of the CitcomS code is successfully translated to CUDA, and as newer general purpose GPU frameworks like Fermi are released, we should continue to see further speedups.

MOTIVATION

Causing a program to run faster doesn't warrant consideration as a project unless improving computation can better enable science to be done. Using tools like CitcomS, geophysicists hope to be able to simulate Earth with increasing accuracy. This accuracy is seen in using finer spatial grids to resolve complicated flows and running for more time steps to follow the system's evolution further. Enabling CitcomS to do the same computations in less time helps accomplish both goals. In order to do those computations in less time, they must be done in parallel.

CITCOMS PROGRAM STRUCTURE

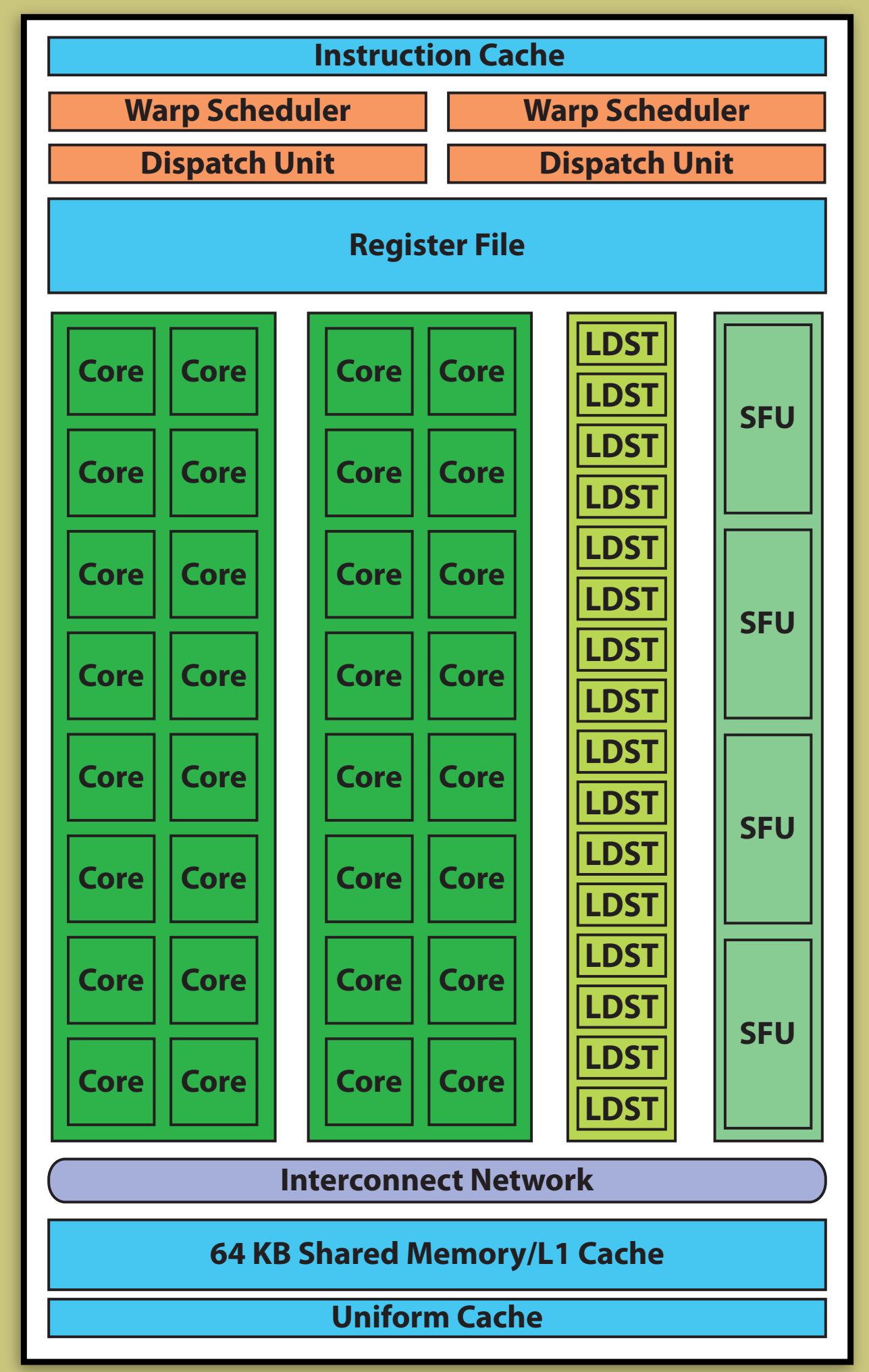
Finite element calculations like those in CitcomS rely principally on the values of nearby neighbors, allowing computation of the entire grid to be split up into chunks which can each be computed concurrently, with some communication at the boundaries. Existing CitcomS code has been made to run in parallel using MPI for that communication, but after each processor gets its large segment of the grid, it does calculations on that grid in serial. Scaling up the number of CPUs to decrease the amount of serial computation gets prohibitively expensive in power consumption.

Graphics processors are designed to do calculations on enormous amounts of data, so the calculations done across the entire grid in CitcomS should map nicely. GPUs have limited synchronization capability, so in order for all of the threads in a GPU to operate concurrently, the calculations must be largely independent of each other. The finite-element method in CitcomS consists of local computations based on immediate neighbors, making it theoretically well-suited for this.

COMPUTE UNIFIED DEVICE ARCHITECTURE (CUDA)

In order to allow programmers to leverage their graphics processors for general purpose computation instead of just using them for graphics, NVIDIA developed a parallel computing architecture for their GPUs called CUDA. Using "C for CUDA" (C with NVIDIA extensions), programmers are able to write programs that run hundreds of threads, each executing the same "kernel" function to process different data. Issues involved with programming for CUDA include:

- Getting maximum performance requires knowing the hardware.
- Computation is FREE compared to accessing memory, so values should typically be recomputed instead of stored in tables whenever it is feasible.
- The CUDA memory hierarchy is explicit, so caching must be handled by the programmer. Without addressing this, performance can suffer by orders of magnitude.
- Some algorithms written to be computed in serial must be refactored to run in parallel, as their assumptions about cost are not accurate for GPU hardware.



GPU Logical Hardware Layout

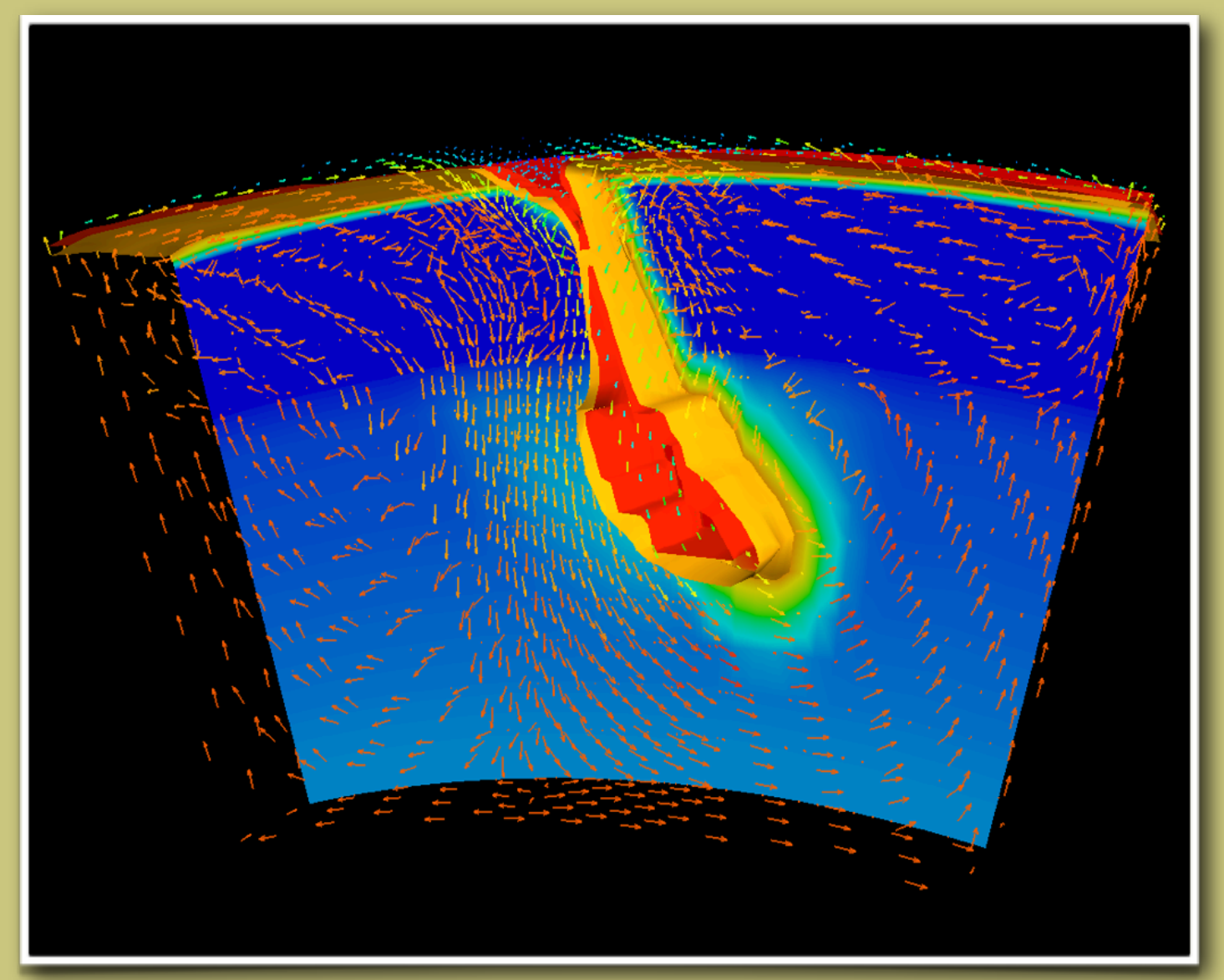
Regional Solver – Profile

% Time	Function Name
78.82	n_assemble_del2_u
4.70	conj_grad
4.06	global_vdot
3.49	assemble_div_u
2.82	get_elt_k
2.12	assemble_grad_p
1.17	regional_exchange_id_d
…	…

Profiling
Shown is a typical profile of CitcomS's Regional solver run with 8 processes via MPI. The most benefit from optimization will come from moving the code where the most compute time is spent onto the GPU.

<u>GPU Hardware</u>
Al-Salam (Earlham):
Tesla C1060
Lincoln (NCSA):
Tesla S1070

Visualization of CitcomS Data



TRANSLATING CITCOMS

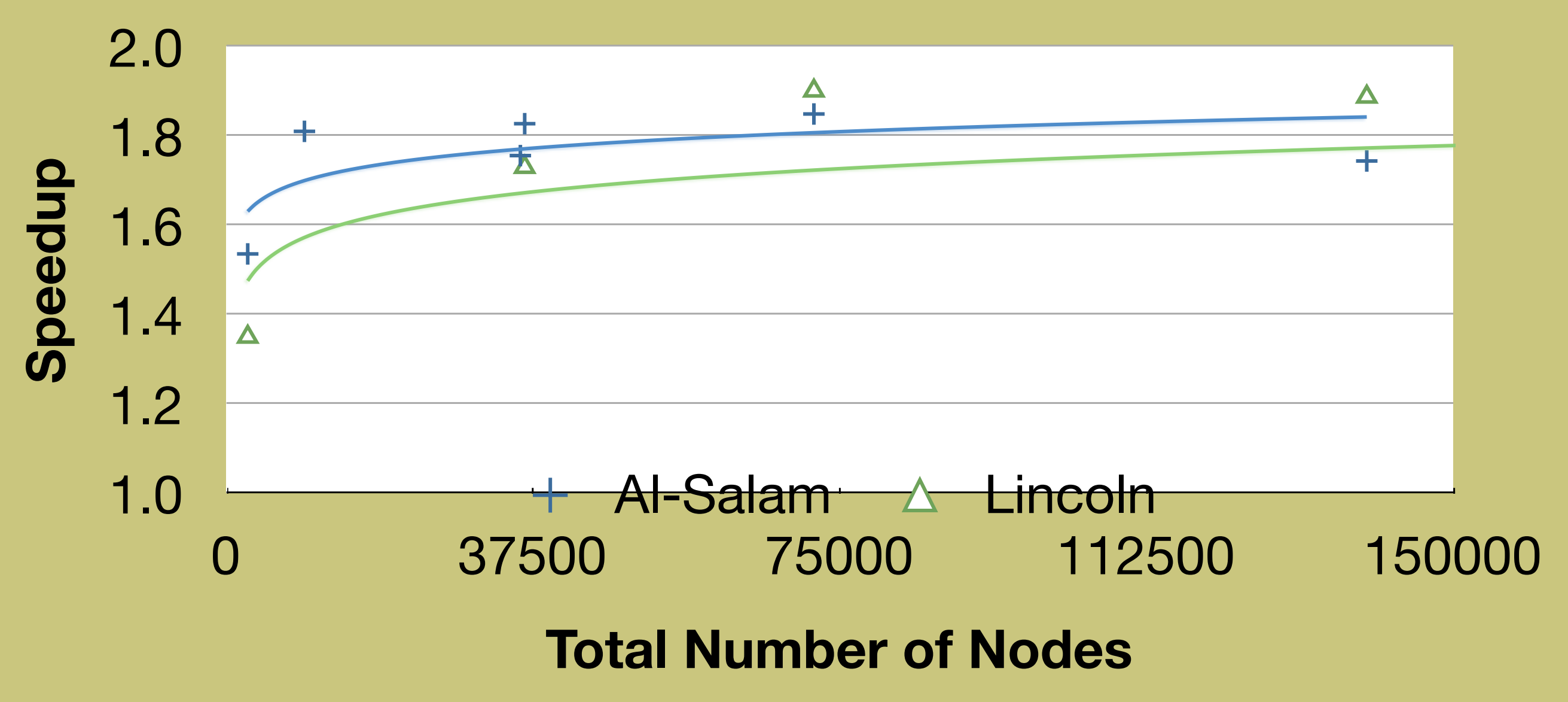
The function n_assemble_del2_u, where the majority of the compute time was being spent, was translated literally computation for computation. This function is used to calculate the Laplacian of the velocity matrix in order to solve for the flow using the Conjugate Gradient solver. Instead of iterating over the nodes as in the serial function, the CUDA version has each thread do one node's calculations, and all the threads are executed concurrently.

- *Algorithm refactoring*: In the original loop, each iteration wrote to neighboring nodes' variables, so in order to run in parallel, calculations were re-ordered so that each node instead read values from each of its neighbors and modified only itself.
- *Recomputing values*: Instead of using an array in memory which mapped out each node's neighbors, the CUDA version calculates each node's neighbors every time. Before this change, the CUDA version was an order of magnitude slower.

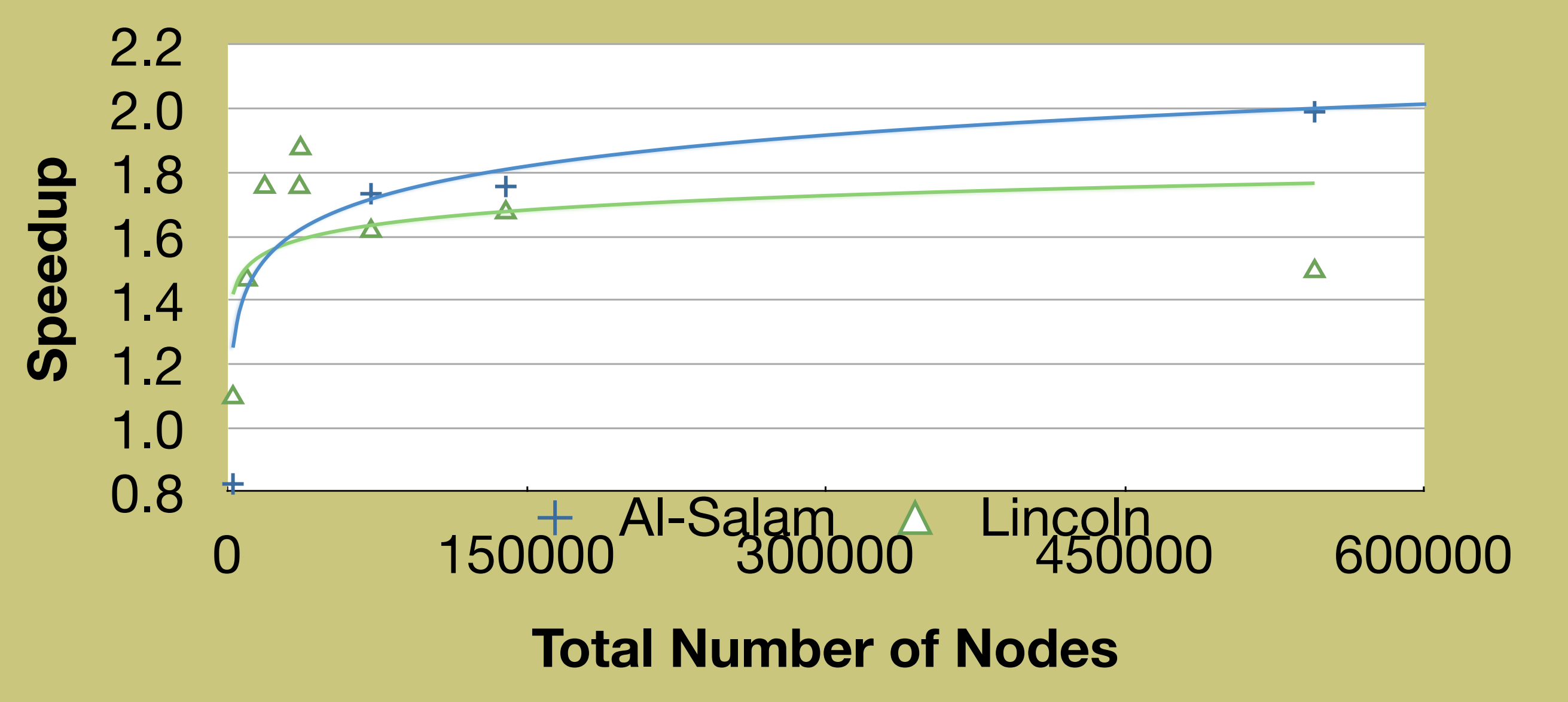
SPEEDUP

Comparing just the time for one function call for the CPU and GPU versions, we saw an average speedup of 2.5. Due to Amdahl's Law, because we're only parallelizing part of the computation, we expect a speedup slightly less than 2. Actual results showed an average speedup of 1.8 that appears relatively flat for typical model grid sizes.

Speedup - Regional



Speedup - Regional (2 processes)



LOOKING FORWARD

These results should also scale with MPI (a flat speedup for every CPU paired with a GPU). Graphics processor accelerators are already available on some high-performance systems, and many new systems are being designed to incorporate accelerators.

In addition, these times were conducted using cards with the Tesla architecture. The newer Fermi architecture released in March 2010 has many improvements, including faster double-precision math and better caching, leading to further speedups.

These results are preliminary; currently only one function has been translated. A translation of the Multigrid solver, which is more generally used, is currently under way.

Finally, the process for building CitcomS with CUDA enabled will be streamlined to make it simple for users to install on their GPU-accelerated clusters or CUDA-capable workstations.