**Maximizing the Efficiency of the Newton-Raphson Solver Using GPU Technology within Climate Models**

**Abstract**

An accurate model of the Earth’s climate is essential to project future and past conditions of the planet. Creating these models requires an immense amount of computing power, which requires vast quantities of resources, mainly time and money. Currently, most of the methods involved in these models are run on Central Processing Units (CPUs), which are capable of processing only single threads of execution. In order to maximize the efficiency of the climate model subroutines, Graphical Processing Unit (GPU) computing, which entails multi-thread processing, is used as an alternative solution. This paper discusses the implementation of the Newton-Raphson method, the mathematical basis of some climate modeling subroutines, onto GPU processors. This alternate implementation resulted in a 500 time speed increase over similar solvers run on CPUs. By improving the efficiency of subroutines run within the climate model, additional data can be collected within a defined period of time, resulting in increasingly accurate projections of the Earth by the climate model.

**Maximizing the Efficiency of the Newton-Raphson Solver Using GPU Technology within Climate Models**

**Executive Summary**

In order to accurately study and model the Earth’s climate, climate models must continuously collect and calculate data to create exact projections of Earth and its atmosphere. The Community Earth System Model (CESM) is a climate model which has the capability to create projections of multiple aspects of the Earth. The Community Land Model (CLM), a submodel within the CESM, encompasses all aspects of Earth’s land, i.e. vegetation. Most methods currently used to collect data for these projections only utilize processors which can run merely one line of execution at a time within the entire model, thereby dominating a majority of the CLM’s runtime. The project focuses mainly on creating a more efficient way of solving systems of equations within the CLM. A program was created with the capability of executing multiple calculations at once, thus accelerating the program up to 500 times that of previous applications. This new method conserves both time and money as well as expected improved projections on the climate model.

**Maximizing the Efficiency of the Newton-Raphson Solver Using GPU Technology within Climate Models**

**Introduction**

Titan, the second largest supercomputer in the world, is capable of completing calculations at a speed of 20+ petaflops or floating point operations per second. Titan has reached these high levels of speed and efficiency by running a hybrid architecture with a combination of Central Processing Units (CPUs) and Graphical Processing Units (GPUs). Historically, supercomputers such as Jaguar consisted of large numbers of CPUs, each of which can only perform a single operation at a time. Because of this limitation, previous supercomputers required a large amount of energy and physical space to reach high levels of computing capacity. However, Titan’s use of GPUs allows it to execute multiple threads of code in parallel on each GPU. This capability reduces the number of processors needed and increases computing power to ten times that of Jaguar while using roughly the same amount of energy and space.

Despite the obvious benefits of using GPUs, many programs have not yet capitalized on this power. These computing capabilities have not been fully utilized because GPU computing has several drawbacks, namely that GPUs are unable to store data. This limitation requires that they are used in conjunction with CPUs where the data is passed for storage and output. A majority of the runtime on GPU centric programs is the passing back and forth data between the CPU and GPU. Therefore, when running the program only one or two times, a primarily CPU program is preferable because it does not require additional processing time to pass data between processors. Additionally, programs coded for CPUs must be reformatted to function on GPUs. By adapting algorithms currently run only on CPUs to function using a combination of CPUs and GPUs, the maximum computing capacity of Titan can be exploited [1].

One algorithm currently being modified to execute more efficiently using a combination of CPUs and GPUs is the Community Earth System Model (CESM). This model contains data about the Earth's climate, including information from both its past and present, which can be used to project future climate conditions. In order to capture and project the conditions of the Earth accurately, the CESM must be a comprehensive model, taking into account all factors that affect the climate. Because all aspects must be considered, the CESM is composed of smaller sub-models, including Atmosphere, Ocean, Land, River, Sea Ice, and Land Ice Models, which focus on these specific elements of the Earth’s surface. [2] These separate sub-components of the CESM must interact with each other on a larger scale, so they must be flexible to changes and improvements while also remaining compatible with each other. Multiple research facilities contribute to the CESM, resulting in an interactive community model allowing contributors to collaborate [3].

The Community Land Model (CLM) is the sub-model within the CESM used to represent aspects of the land surface, including land biogeography, ecosystem dynamics, the hydrologic cycle, biogeochemistry, and human dimensions affecting the land. The model subdivides the terrain into five distinct types of land covering, including vegetated cover. This subtype has an algorithm that partitions the vegetation area into smaller grids in order to calculate and store information on specific plant types in the region. The Canopy Fluxes subroutine located in the CLM calculates the latent heat and water vapor flux of an area’s vegetation based on the surrounding ground and vegetation temperature and the specific humidity.

Our project focuses on optimizing this subroutine by adapting the primary mathematical algorithm used in Canopy Fluxes to run using GPUs. The calculations in the subroutine are performed using a form of the Newton-Raphson method, an iterative process that can be used to solve nonlinear systems of equations similar to those found in Canopy Fluxes subroutine. Because the objective is to implement the method in the Canopy Fluxes subroutine, our project focused primarily on constructing a solver for a system of 16 equations, the size of the system involved in the subroutine. Completing the calculations involved in the Newton-Raphson method requires roughly 68% of the CLM’s runtime, meaning that improvements to this subsection of code will drastically reduce the time required to run the model [4]. As improvements are made to increase the efficiency and functionality of this subroutine in the CLM, projections made by the CESM as a whole are improved [5].

**Newton-Raphson Method**

The Newton-Raphson method is based on a first degree Taylor Polynomial and its LaGrange error. The series expansion of the function f(p) at point can be expressed as

|  |  |
| --- | --- |
|  | (1) |

where *c* is between and . To find roots of the equation, is set equal to zero. To perform the iteration, the error term is negligible producing the equation

|  |  |
| --- | --- |
| . | (2) |

Solving for p produces the root-finding algorithm known as Newton's Method.

|  |  |
| --- | --- |
|  | (3) |

When an initial guess is inserted into the equation, Newton’s Method results in a new value, , closer to the true root than the initial guess. This process can be performed iteratively until the desired accuracy is obtained.

The general form of the method can then be modified in order to solve a system of nonlinear algebraic equations using a vector composed of the equations and its Jacobian matrix.

|  |  |
| --- | --- |
|  | (4) |

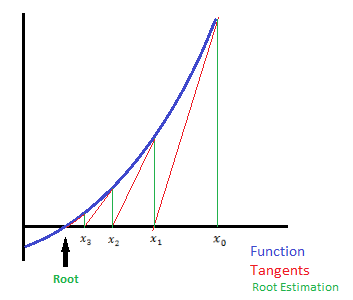
In this form, is a vector of values for the current iteration, is the Jacobian matrix formed from these values and is the vector of the values of the system of equations at . The matrix is formed by approximating the partial derivatives of each equation with respect to each variable.

Figure 1: Graphical representation of the root finding capabilities of the Newton-Raphson Method

In the past, other programs have used variations of the Newton-Raphson method on GPUs. However, these programs were created to run most efficiently for large systems of hundreds of equations. The Canopy Fluxes subroutine uses a system of 16 equations to be solved with the Newton-Raphson method, a relatively small number in comparison to other systems using the method with thousands of equations. Additionally, the Newton-Raphson method must be run for each grid cell on the land at each time-step of the CESM, resulting in thousands of solves per time-step.

**Methods**

**CPU Implementation**

A simple, one dimensional Newton Raphson solver, like the one in Equation 3, can be implemented as:

program iteration

real :: x=900, err=1e-4, it=0, dfx

real function fx(x)

fx=x\*\*2-1

do

dfx=(fx(x+err)-fx(x))/err

if(fx(x)<err) exit

x=x-fx(x)/dfx

end do

end program iteration

In this solver, the root of a stored function (fx) is determined from a starting point (x). The solver puts the initial guess into a loop which first approximates the derivative of the function at the point x, checks to see if the guess is correct within a certain accuracy (err), and then, if not within the allotted error, continues to set a new x through the Newton-Raphson equation. This loop repeats until the desired accuracy or maximum number of iterations is achieved.

For systems with higher dimensions, the Newton-Raphson solver takes an initial guess for the convergence of a system of 16 stored functions. For comparative purposes, a solver for a system of five equations was also created to observe the differences in program runtime depending on the number of equations. The solver uses a subroutine that creates an array of the values of each function at these estimated values. Then a Jacobian matrix is created using the partial derivatives of the functions at the estimated values. To solve for the next estimation, Gaussian Substitution is used to find the matrix inverse. As modeled in Figure 1, the solver then performs numerous iterations until a predefined maximum number of iterations is achieved.

Figure 1: A flow chart showing the Newton-Raphson iteration.

Initial guess

inputted

Array of function values created

Jacobian matrix built

Jacobian matrix inverted

calculated

New iteration values

stored

Newton-Raphson

Iteration

Because the subroutine requires the system to be run continuously, our solver was developed differently from previous solvers that required small numbers of runs. The method in the developed solver was designed to be most efficient for small systems that need to be solved large numbers of times.

Large systems are typically solved iteratively in the form

|  |  |
| --- | --- |
|  | (5) |

in order to avoid inverting the Jacobian matrix. This time efficiency of the iterative methods is expressed by the function O() where *m* is the number of iterations required to solve the equation and *n* is the size of the system. Because of the differences in size, the Newton-Raphson method for the Canopy Fluxes subroutine was approached by taking the inverse directly, as shown in Equation 4, with an efficiency measured by the function O(). Typically, in larger systems of equations, making the iterative approach more efficient, however in the small system used in Canopy Fluxes, the number of iterations required will usually be greater than the number of equations, making direct inversion more efficient.

**GPU Translation**

Due to its iterative nature, the Newton-Raphson method functions most efficiently in GPUs which allow 1024 solves to be performed simultaneously. The Jacobian matrix is built, inverted and multiplied, in parallel on the GPU [6]. Once all of these calculations are performed, the data is then transferred back to the CPU for storage and the process is repeated for subsequent iteration reference points.

We developed a flexible code that was translated to run on GPUs with CUDA FORTRAN that can be easily modified to solve various sizes of systems, including systems of 16 equations such as the system found in the Canopy Fluxes subroutine. To demonstrate the flexibility of the Newton-Raphson solver, several systems of different sizes were tested with varying numbers of runs. The solver was run first on a CPU, then on a GPU in order to compare the runtimes of the two processing units and assess the efficiencies of each processor.

As Titan is a large, expensive supercomputer, its runtime is dedicated mainly to five large programs, including the CESM. Because of this exclusivity, data for our Newton-Raphson solver was collected using Yona, a GPU capable computer. While Yona does not have as much computing capacity as Titan, testing the solver alone only requires one computing node. Thus, yielding the same time ratio on any GPU capable computer. Therefore the runtimes of the solver were not greatly affected by the device used.

**Equations**

For testing purposes, systems consisting of five and 16 equations were created. In initial tests, these equations were all composed of simple polynomials with shared roots of 0. The relative simplicity of the equations made them possible to easily check the accuracy by hand. After testing the initial equations, systems of higher complexity, such as multidimensional trigonometric functions, were used in order to ensure the solving capabilities of the algorithm would be viable for the varying complexity within the climate model. These equations were also created to ensure they had a shared root of zero in order to calculate the accuracy of the solver.

**Results**

After testing the program on Yona with various numbers of runs, the times shown in Table 1 were gathered for the system of 16 equations and the times in Table 2 for the system of five equations.

Table 1: Comparison of Newton-Raphson solver run times on CPU and GPU for a five equation system

|  |  |  |
| --- | --- | --- |
|  | System of 5 Equations |  |
| Runs | CPU Time  (s) | GPU Time  (s) |
| 1 | 4.9829483E-05 | 2.2060871E-03 |
| 512 | 1.6713142E-02 | 2.1691322E-03 |
| 1024 | 3.6828995E-02 | 2.1739006E-03 |
| 2048 | 7.9938889E-02 | 2.1860600E-03 |
| 4096 | 0.2209630 | 2.1381378E-03 |
| 8192 | 0.6169291 | 2.1519661E-03 |
| 16384 | 1.304934 | 2.3169518E-03 |
| 32768 | 2.585130 | 2.0940304E-03 |

Table 2: Comparison of Newton-Raphson solver run times on CPU and GPU for a 16 equation system

|  |  |  |
| --- | --- | --- |
|  | System of 16 Equations |  |
| Runs | CPU Time  (s) | GPU Time  (s) |
| 1 | 9.7203255E-04 | 4.2075872E-02 |
| 512 | 0.5471780 | 6.9380999E-02 |
| 1024 | 1.402217 | 8.4589005E-02 |
| 2048 | 4.954849 | 8.7203026E-02 |
| 4096 | 10.23711 | 0.1056719 |
| 8192 | 24.83107 | 0.1401691 |
| 16384 | 77.83984 | 0.2924194 |
| 32768 | 246.8770 | 0.4987183 |

The data in Figures 2 and 3 show the rapidly increasing runtimes of the CPU program while the GPU times remain relatively stable regardless of the number of runs required. The graphs are each shown on with a y-axis of log base 10.

Figure 2: This graph shows the runtime of five equations using the Newton-Raphson Method

Figure 3: This logarithmic graph depicts the runtime of 16 equations using the Newton-Raphson Method

Figure 4: This graph depicts the exponential increase of the CPU runtime and static GPU runtime

**Discussion**

Tables 1 and 2 show that for 16 equations, while running the solver 32,768 times on the CPU takes over 250,000 times as long as running the program once, the program can be run 32,768 times on the GPU while only taking about ten times as long. Additionally, for 32,768 runs, the GPU ran roughly 500 times faster than the CPU with the equations tested. This performance demonstrates the comparative efficiency of running the Newton-Raphson solver developed on the GPU rather than on a CPU. From the times obtained from the GPU and CPU implementations of the Newton-Raphson subroutine, it can be determined that the GPU is many times more effective than the CPU­­ for the systems studied, which were small and solved many times simultaneously. These conditions are similar to those present in the Canopy Fluxes subroutine, demonstrating the benefits of adapting this subroutine to use GPU processors.

From the data shown in Figures 2 and 3, it can be observed that, for the system of five equations, the GPU time shows negligible variation between the runtimes of one and 32,768 runs. This consistency is a result of the time cost which for a system of five equations would result in a number less than 1024, the maximum number of simultaneous calculations on a GPU. For the same reason, there is a slight increase in time required for the system of 16 equations as 163 would exceed the maximum number and cause escalation in time. Additionally, these figures demonstrate that for small numbers of runs, the CPU is more efficient. Although the point at which the two processors are equally efficient is unknown, for the large numbers of runs needed by the climate model, the GPU is the more effective method.

**Solution Accuracy**

For the simple systems of polynomial expressions, the Newton-Raphson solver found the root at zero for all equations to machine accuracy of nine decimal places. In the comparison between the GPU and CPU for the more complex system, the results found by the two different processors were not in agreement, and additionally were not equal to the intended root of zero. Because the system of equations entered into the solver on each processor consisted of the same equations and initial guess, both processors should theoretically produce the same solutions. We believe this inexactitude arises because, as the equations used become more complex or have more possible roots, more accurate initial guesses are also required to ensure that the solver converges to the correct root. Additionally, the incongruence between the solutions found by the two processors may arise from machine errors during the passing of information between the GPU for calculation and the CPU for storage. While for individual calculations, this machine error appears small, it is compounded as the system is solved an increasing number of times. A small amount of machine error is unavoidable, however more accurate initial guesses as well as more precise calculations of partial derivatives within the Jacobian matrix will minimize this inexactitude.

**Conclusion**

Currently, the CLM runs primarily on CPUs. Because of the inefficiency of solving each thread individually, the program dedicates a majority of its runtime to the inefficient CPU-based Newton-Raphson solver. By harnessing the power of GPU computing, we were able to develop a Newton-Raphson solver that will increase the efficiency of the Canopy Fluxes subroutine in the CESM. Our approach can potentially accelerate the rate at which the solver runs to approximately 500 times faster, cut computing times, and subsequently decrease computing cost. In order to obtain increasingly exact results with future versions of the Newton-Raphson solver, we hope to investigate and implement more accurate methods of taking the derivative as well as more exact initial starting values for the subsequent iterations. Additionally, the current Newton-Raphson method is restricted only to systems of equations which yield invertible matrices. The ultimate goal of further research will be to apply this technique directly to the Canopy Fluxes subroutine within the CESM on Titan.

**References**

[1] “Introducing Titan | The World's #1 Open Science Supercomputer.” *OLCF Titan*. Oak Ridge National Laboratory, 2012. Web. 12 July 2013.

[2] “CESM Models.” *Community Earth System Model (CESM)*. N.p., 2013. Web. 12 July 2013.

[3] Neale, Richard B., et al. “Description of the NCAR Community Atmosphere Model (CAM 5.0).” *NCAR/TN-486+STR NCAR Technical Note*(2010): 1-5. Web.

[4] Wang, Dali. *A Sensitive Analysis Framework for the Global Offline Community Land Model Simulation*. *ORNL*. ORNL, n.d. Web. 26 Sept. 2013.

[5] Oleson, Keith W., et.al. “Technical Description of Version 4.0 of the Community Land Model (CLM).” *NCAR/TN-478+STR NCAR TECHNICAL NOTE* (2010): 61-99. Print.

[6] Pang, Tao. “Subroutine MIGS.” *Physics University of Nevada*. University of Nevada, 4 Oct. 2001. Web. 5 Aug. 2013.