Light Curve Fitting on Heterogeneous Computers

Kevin J. P. Luecke

Table of Contents

1. Introduction

The field of computer science is one of the most recently created and rapidly expanding sciences. In the years since its inception in the mid 1900s, it has revolutionized the other sciences, businesses, and our personal lives. With increasing capabilities, however, comes increasing complexity.

This increasing complexity was quantified in 1965, when Gordon Moore predicted the rough doubling of the number of components per integrated function on a chip every ∼1.5 years (10). This trend has remained more or less true up to today and is expected to continue until at least 2015 (8). This miniaturization was expected to allow for "higher speed for the same power per unit area" (10). That is important, as it allowed the increasing complexity of the architecture involved to remain hidden from programmers. As keeping the

same, single core design promised better performance in the next generation of hardware without compromising anything, there was no need to target different platforms. This fact remained mostly true until 2004. On May 7th of that year, Intel canceled the next generation of Pentium 4 processors due to heat problems (7),(3). Above the 100nm level, power consumption had been a function of the size of the components so making them smaller was actually more power efficient. At smaller then 100nm, however, power consumption increases with miniaturization(3). In 2006, two years after this cancellation, Intel released Core Duo processor, marking a dramatic shift in computing–away from single core and towards parallel programing paradigms.

Intel's was not the first parallel architecture to be widely available, however. It was 5 years earlier in 1999 that NVIDIA released the the GeForce 256. This is one of the first examples of a wide spread specialized hardware for a single application. The first set of GPUs were specific to the task of rendering graphics to a a screen. However, in attempting to solve problems in the field of graphics, NVIDIA and ATI created devices with increasing program-ability. This flexibility granted GPUs the power to solve other, non-graphics related problems as well. The first GPGPU (General Purpose GPU) applications were mostly academic, notable only because they functioned at all (5). Responding to these simple applications ATI released the first unified shader GPU in 2005, which had dramatically improved programability (5). Ease of programming on these devices increased greatly when NVIDIA released CUDA in 2006, a C like language intended for GPGPU use (11).

These devices have gained mass appeal because in the right circumstances, they can run many more computations per second than a cpu, and with less power consumption (4). Unconventional hardware like this can "provide a significant benefit in performance from improved energy efficiency" with "parallelism in excess of 90%" (4). Parallel algorithms that can take advantage of this are often seen in scientific computing, and more and more in the private sector. Mobile devices are primarily limited by battery power, and "all U[nconventional]-cores, especially those based on custom logic, are more broadly useful when power or energy reduction is the goal rather than increased performance" (4). Because of this, we can expect to see much more heterogeneous hardware (containing traditional CPUs as well as unconventional hardware).

On a heterogeneous platform, the problem of minimizing run-time of a program becomes much more complicated then on a homogeneous platform (8). The speed of a program on heterogeneous systems is not simply a function of the speed of the CPU. Rather it is generally possible to increase performance by taking advantage of all available resources. This could be achieved by extensive fine tuning of programs; however, this tuning would be different for each machine you ran it on depending on its exact makeup. Even for programs designed

to work on a specific machine, if one of the components is upgraded, it could easily change how the work needs to be divided.

A solution to minimize this problem has been proposed in the form of GAMA, a joint project between the University of Texas and the University of Minho in Portugal. GAMA (GPU And Multi-core Aware) addresses this problem by providing a runtime system that determines the amount of work that should be sent to each available resource dynamically (8). It provides a simple interface so it can be included with minimal changes to the original algorithm (8). Further, it promises to add very little overhead to program execution and correctly schedule irregular programs.

In order to demonstrate the abilities of this approach, a sample program with real applications in astrophysics has been implemented. This program, Lcfit Theta, attempts to find the size of convection zones in white dwarf stars. Most white dwarfs pulsate or oscillates in temperature and density during their life cycles. The oscillations can be seen from earth telescopes which record a star's brightness as a function of time in a graph known as a 'light-curve'. These light-curves can be reproduced fairly well by a simple linear sum of sine functions corresponding to the frequencies and amplitudes at which it the star is oscillating. However, in stars where a convection zone is present at the surface, non-linear combinations of these sine functions appear in the light-curve. Lcfit Theta uses a model of these white dwarfs proposed by Montgomery which includes the effects of these convective regions, to generate synthetic light curves (9). By comparing these synthetic light curves to the measured stellar data, we can find the input parameters to Mongomery's model which create a synthetic light curve that matches the data. One of these 'fitted' input parameters is the size of the convection zone. This application is the focus of this thesis.

1.1. The Expected Effects of this Application

Complex scientific computation is most often accomplished on massively parallel supercomputers like those at the Texas Advanced Computing Center (TACC). The highest performance supercomputer at TACC (Stampede) included GPUs in its architecture. This application is meant to run on that platform, so it includes a CPU and GPU implementation of the algorithm. Lcfit Theta is interesting because it involves two different models with substantially different complexity. First and foremost, this program contains a full treatment of the problem, which is believed to be more accurate, but computationally intensive. In order to save time, this program also contains a simplified model, which is intended to identify likely candidate values, reducing the workload of the more full model. These two models have varying ability to take advantage of CPU and GPU resources. The simple model runs

mostly on the GPU while the full treatment is much more CPU dependent. When this program is run for astronomy, it will switch off between these two models frequently. This means GAMA will need to react quickly, assigning more work to the GPU when running the simple model, and more to the CPU when running the complex model.

2. Astrophysical Context

White dwarfs are some of the simplest stars. By studying them, we can learn about their progenitors, which make up 97% of stars. They can also be used to find the age of the galactic disc (13). These determinations, however, rely on the accuracy of our models of white dwarfs. In white dwarf science and the field of Astrophysics in general, one of the most prevalent, and least understood phenomena is convection. The current method for predicting convection zone size, Mixing Length Theory, simply assumes it is proportional to a pressure scale height in the star (6). The reason for this uncertainty is the non-local nature of convection. One could argue that the beginning of modern science, (certainly classical physics) came when Newton developed calculus. Calculus is the system we use to describe change, and has been widely applied across the sciences. However, this technique, while incredibly useful, cannot describe non-local phenomena, or situations where any point can be effected by far off points. Convection is just such a situation. If you can't break these problems down to a simple, local approximation, we have no analytical tools to solve them. These type of problems went largely unsolved until recently. With massively parallel architectures, we can now employ a new technique to solve this problem, namely we can compute convective regions and their effects directly.

2.1. Interaction Between Pulsation and Convection

In his 2005 paper entitled "A New Technique for Probing Convection in Pulsating White Dwarf Stars", Montgomery builds on the work of Brickhill and that of Goldreich & Wu, to develop a technique to measure convection zone size by fitting observations (9), (1), (14). It is Montgomery's results which I have implemented as my full treatment. In this section I will follow his derivation. This technique takes advantage of the extreme oscillations in the physical properties of the atmospheres of pulsating white dwarfs.

As part of their evolutionary cycle, white dwarfs with hydrogen atmospheres (DAs) enter what is known as the instability strip, where they can change in brightness by 10% in a matter of minutes, as pictured in Figure 1. These brightness variations are a result of

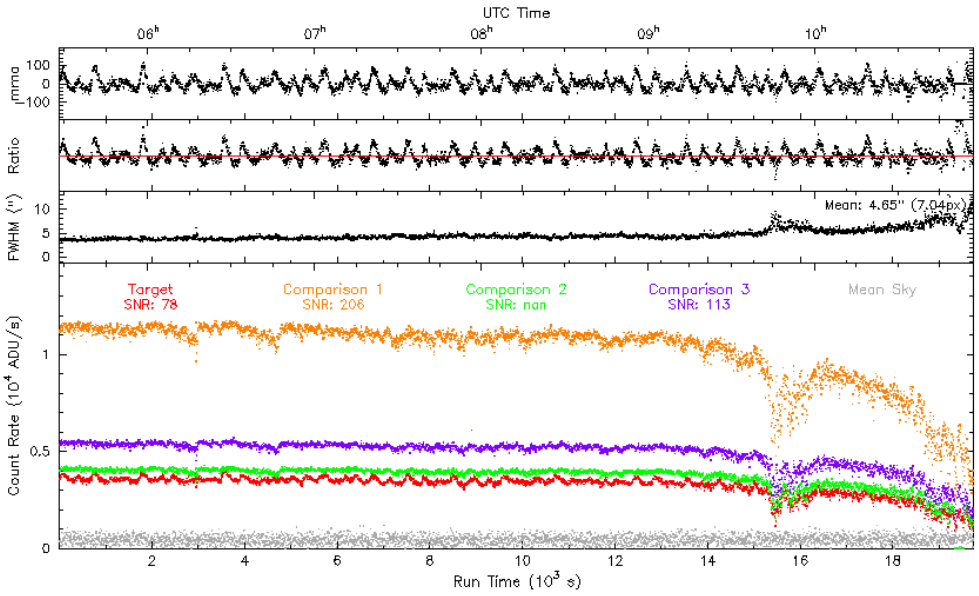

Fig. 1.— Observation of HS05057+0854 taken at the 36" telescope at McDonald Observatory on December 12^{th} 2012. The bottom graph is the measured values of the tracked stars. The top graph is the brightness of HS0507+0835 corrected for cloud cover and instrumental deficiencies, measured in $\frac{1}{1000}$ths of its average brightness.

harmonic oscillations in temperature and density throughout the star. These variations in brightness (or energy flux), then should be described by standard harmonic oscillations of a spherical object:

$$F = \sum_j Re[A_j e^{i(\omega_j t + \delta_j)} Y_{l_j,m_j}(\theta, \phi)].$$

Here the characteristic sinusoidal behavior of a wave is represented by $A_j e^{i(\omega_j t+\delta_j)}$, where each of the j waves has an amplitude A, a frequency ω, and a phase δ. The effects of the spherical structure of the star are taken into account through Lapace's spherical harmonic equations, $Y_{l,m}(\theta, \phi)$, where θ and ϕ are spherical coordinates, and l and m are integers such that $-l < m < l$. This fits our observations fairly well, however the light curves show additional nonlinear effects. These effects have long been attributed to convection (1). As a star pulsates, the fundamental physical parameters of the star (e.g., its temperature and density) change. When these parameters change, so does the size of its convective envelope.

Three facts about these stars make it possible to parametrize this constantly changing convection zone. First of all, the time it takes a convection zone to relax into stable state (convective turnover timescale) is much less than the time is takes a pulsation to complete a cycle. Secondly, the radial size of the convection zone is much smaller than the radial

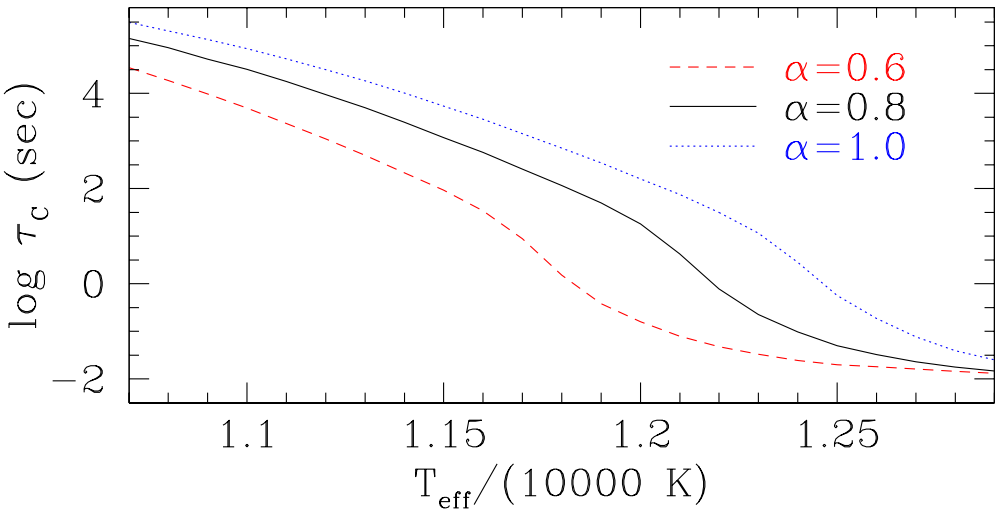

Fig. 2.— Values for τ_c from Mixing Length Theory

wavelength of a pulsation. This means at any given time we can assume that the convection zone on any section of the surface of the star is the same as a convection zone of a non-pulsating star in equilibrium at that temperature. Finally, as the convective envelope lies on the very outer edge of the star, we can assume that the energy flux underneath the convection zone strictly follows the equation above.

As a fluid becomes convective its method of energy transport changes, which can be thought of like a phase change. This means the energy flux leaving the star is the flux entering the convective region minus the energy it takes to "change the phase" of fluid elements from convective to non-convective:

$$F_{phot} = F_{base} - \tau_C \frac{dF_{phot}}{dt},$$

where τ_C is the convective turnover timescale and can be thought of as the heat capacity of the convection zone. The formula for the flux at the base is the standard model for white dwarfs mentioned earlier. Using mixing length theory, we can estimate the convective timescale as a function of temperature. Mixing length theory uses an order of magnitude approximation of the size of convection zones to approximate its effect on the stars brightness (6). The guess for the size of the convection zone is controlled by the mixing length parameter α. Even if you vary α, τ_C is fitted by a power of $T_{\rm eff}$. To see this more clearly we can plot $\log(\tau_C(T_{\rm eff}))$, and notice that it is nearly linear, with a negative slope (see Figure 2). From this we can approximate the convective timescale:

$$\tau_C(T_{\rm eff}) \approx \tau_0 \times \left(\frac{T_{\rm eff}}{T_{\rm eff,0}}\right)^{-N}$$

Here $T_{\text{eff},0}$ is the starting temperature, τ_0 is a constant depending on α, and N is a constant approximately between 90 and 95 in DAs. This supports the theory that τ_C is strongly dependent on T, and the convection zone strongly affects the energy flux at the photosphere (F_{phot}).

Using this model as the basis for a fitting program, Montgomery has been able to reproduce the light curves of simple stars with extreme accuracy. However, this program is not able to fit all of the parameters quickly enough to be useful on complex stars. Using GAMA, we can improve the speed of this program enough to fit more complicated stars.

2.2. Lcfit Theta and GAMA

Lcfit Theta solves the following ordinary differential equation (ODE), reproduced in full:

$$F_{phot} = -\left[\tau_0 \times \left(\frac{T_{\text{eff}}}{T_{\text{eff},0}}\right)^{-N}\right] \frac{dF_{phot}}{dt} + \sum_j Re[A_j e^{i(\omega_j t+\delta_j)} Y_{l_j,m_j}(\theta,\phi)]$$

This equation has the free parameters of the convection zone, τ_0 and N, as well as A_j, ω_j, δ_j, l_j, and m_j for each mode of oscillation. Additionally, there is the final free parameter which measures the orientation of the axis of pulsation relative to earth (known as the inclination angle θ_I). This final parameter comes into play when we attempt to create the light curve predicted by this model. Fortunately, using the simple approximation of the star as a spherical oscillator we can find each ω_j, and we get a good starting guess for each δ_j and A_j. Using mixing length theory, we can also find a good starting guess for N (namely 90-95 for DAs) and τ_0.

Up to this point, Lcfit Theta has only been fast enough to refine the initial guesses for N, τ_0, θ_I, and A_j and δ_j for each pulsation frequency. So users have to put in their own guesses for the l_j and m_j of each frequency. This strategy is tractable if you are fitting a star with a small number of frequencies; however, for stars with many frequencies, this creates too many possibilities to test. Not trying all the possible combinations calls into question the values of the parameters we do fit, which is a shame as this technique is one of the only ways we can probe l and m values of the pulsation modes. Better determining both the convection zone size and spherical harmonics of the pulsation modes will help us improve our understanding of the interiors of these stars.

This leaves two possible areas of improvement. First of all, we can make this fit as it is implemented now run faster. To do this, we take advantage of the fact that, to accurately model the flux we see, we need to break up the surface of the star into a grid of points

small enough that we can assume they have uniform temperature. Since we are solving the same equation on all of these grid points, we can speed up the simulation by solving this equation for all the grid points concurrently on massively parallel architectures like multicore computers and GPUs.

The second way to improve this fitting program is to programmatically fit the values of l and m. To do this we would like run the entire fitting routine on a wide range of l and m values. This is impractical, however, as for stars with many modes (10+) this creates far too many possibilities. In order to reduce the number of models to run the full treatment on, I have implemented a simpler version which can isolate likely sets of l and m values. I took this formulation from the work of Wu in 2001 (14). This approximation takes the same form as the model of the flux at the base of the convection zone:

$$F = \sum_j Re[A_j e^{i(\omega_j t + \delta_j)} Y_{l_j,m_j}(\theta, \phi)]$$

It simply changes apparent amplitude (A) and phase (δ) of the witnessed modes to approximate the effects of a convection zone of base size τ_0 according to:

$$A_{phot} = \frac{A_{fbase}}{\sqrt{1 + (\omega_{fbase}\tau_0)^2}}$$

$$\delta_{phot} = \delta_{fbase} - \arctan(\omega\tau_0)$$

This dramatically simplified calculation does not always find the same values as 'best fit' as the full integration, however, it does seem to find better fits for the same values of l and m. In order to test this, I have run both methods on a few stars simple enough to preform an exhaustive search with the full integration method. So far Wu's method has identified all of the same the values that the full integration has identified. While this does not prove that Wu's method will identify the "best fit" according to Montgomery's method, it does give us a technique for intelligently narrowing the set of l and m values that we calculate fully.

3. Description of Implementation

This program starts with guessed values of the free parameters τ_0, N, θ_I, A_j, and δ_j, as well as a set of l_j, and m_j values to try for each mode. For each combination of possible l_j's and m_j's, we try to find the best fit of the other parameters.

When trying to find a best fit, we start by calculating a light curve detailing what we would have observed from a star with these guessed parameters. The difference (found by

the chi squared method) between this calculated light curve and our actual observed light curve tells us how good our guess was. In order to decide what to guess next, we employ a fitting algorithm known as mrqmin (12). This works by doing something like a partial derivative of the fit in terms of each parameter. To find this derivative, we simply change one parameter slightly, and re-calculate our light curve. The change in the chi-squared value from varying a parameter tells us to increase or decrease that parameter, and by how much. Following these partial derivatives in each parameter, we can find a local minimum of chi squared, or best fit light curve. In order to make sure we have not missed a better fit, we jump out of this minimum a few times, and follow the gradient from there.

To calculate these light curves, we must divide the part of the star's surface we can see into a grid of points. These points need to be small enough that we can ignore spatial variations in temperature within each gridpoint to satisfy the assumptions made by our models. We first calculate the perturbations due to spherical harmonics across the grid as this is not time dependant. We then calculate the flux of each gridpoint at each time our observed light curve has a value, and then sum across the grid to get our calculated light curve. Our target for parallelization through GAMA is the calculation of this flux over the discretized surface of the star.

To calculate these light curves with Montgomery's model, for each grid-point, we integrate over the ordinary differential equation from a starting point (chosen to be the beginning of the first run) through the end of the final run. This is known as the initial value problem, as all you know is an initial starting point to your function and an expression to calculate the derivative of the function at any given point. This problem can be solved by approximating the solution incrementally from the starting value using the derivatives. A class of algorithms to solve this was proposed by Runge and Kutta at the end of the nineteenth century (2), where the "order" of the algorithm is the number of derivatives used to approximate each step. It is in this integration that the implementations on the CPU and GPU differ.

In the CPU version, I run a Runge-Kutta 4 integration technique on each mesh point individually, taken from numerical recipes (12). For the GPU version, we needed an implementation that could compile in CUDA. To accomplish this, we employed a library entitled "ODEINT" that was included in the BOOST c++ libraries at the end of 2012. This implementation only runs one integration, but it concurrently finds the derivatives of each gridpoint. One other difference is that currently, on the GPU we use the Dormand-Prince method, which is a fifth order Runge-Kutta style integration method. Although we find that the fourth order solution is all that is necessary, the 5th order solution is the only one in the ODEINT library that currently implements all the features we use. The library is in active development, however, and we intend to switch to the fourth order implementation when

that becomes available.

To calculate the derivative of the flux from our section of the star, we need to first calculate the flux at the base of the convection zone. This means simply evaluating the expression:

$$\sum_j Re[A_j e^{i(\omega_j t+\delta_j)} Y_{l_j,m_j}(\theta,\phi)]$$

for our current values of A_j, δ_j, l_j, and m_j at the current time t. The values for θ and ϕ are determined by the gridpoint (k). This calculation is relatively simple, as we have already calculated the static $Y_{l,n}$ values. However, we found significant speed improvements in the CPU version by calculating a grid of $F_{base}(k,t)$ values and interpolating between them in the dxdt step. On the GPU version, however, calculating this flux directly at each call to dxdt performs surprisingly well. A simple linear interpolation scheme produces a maximum speed-up of approximately 10% over the integration in the GPU case with minor loss of accuracy (four times as inaccurate, but still of order 10^{-5}). Any higher order interpolation schemes take longer than simply calculating the flux directly.

Wu's model simplifies things by removing the ODE entirely. For this model we simply modify the apparent amplitude and phase of each mode to reflect the effects of the convection zone and calculate the flux for each gridpoint at each time we have an observed measurement of the star's flux.

Both methods leave us with the calculated F_{phot} at each gridpoint for each time we have an observed measurement. From here we have to sum up the contributions from each grid-point to get our simulated light curve. This process is complicated by an effect known as limb darkening. This effect is due to the curvature of the stellar surface we are simulating. Because of this curvature less flux from the edges of the area we are simulating contributes than from the center. Limb darkening effects can be parametrized as:

$$F_{obs} = \sum_k \mu(k, F_{phot}(k,t)) \times F_{phot}(k,t)$$

To account for this effect, we read in a grid of computed values for $\mu(k, F_{phot})$ and interpolate between them over the two dimensional space. Unfortunately, this grid is to coarse for a bi-linear interpolation to give accurate results. I therefore employed a cubic spline for this interpolation as given in numerical recipes for both the CPU and GPU (12).

This fitting technique works with all the values that can take on any real value, namely τ_0, N, θ_I, A_j, and δ_j. On the other hand l and m can only take on integer values. Also, any change to a mode's l or m value can radically change its contribution to the total flux. This means that for every combination of the possible values for l and m of each mode, we have

to run a separate mrqmin fitting routine, finding the best values of τ_0, N, θ_I, A_j, and δ_j. This is where we take advantage of speed of Wu's method. We first run mrqmin using Wu's model and only when it finds a reasonable fit do we run Montgomery's model.

4. Results

To test this application I ran it on four computers with different hardware configurations, Ix, Scar, Morpheus and Stampede. Ix is the computer of Dr. Montgomery of The University of Texas, who will likely use this software in the future. It contains two 2.93 Ghz 6-core Intel Xeon processors with hyperthreading. Scar is my development computer, it has a 6-core 3.2Ghz Intel i7 processor with hyperthreading and a NVIDIA GeForce GTX 660Ti GPU. Morpheus belongs to the graphics group at the University of Texas, and has two 6-core 2.54 GHz Intel Xeon processors with hyperthreading and two NVIDIA Tesla C2070 GPUs. Finally, Stampede is the newest super computer at the Texas Advanced Computing Center. Each node on Stampede has a 16 core Intel Xeon Phi processor and an NVIDIA Kepler K20 GPU.

I ran tests with mesh sizes of 256 and 1152, which bounded the expected use cases. All the numbers shown are averages of 30 tests, with one standard deviation as the error bars. The averaged results for a mesh size of 1152 are plotted in figures 3, and 4.

Unfortunately Montgomery's model did not extend as well as I had hoped to the GPU. The fastest of the GPUs tested on was faster than a single CPU core, but only just. The ODE solver only uses the GPU for the dxdt calculation, which means it makes a lot of kernel calls in a single light curve calculation. This means GPU calls use some CPU power. When using more than a few cores, not enough data is sent to the GPU for it to be efficient. These device calls actually take up time on the CPUs enough that they are actually detrimental if you are using enough CPU cores.

Wu's model, on the other hand runs much faster on the GPU than on the CPU. Hyperthreading on this model was ineffective. The calculation has little logic, so hyperthreading shouldn't have provided any benifit. There is some overhead in creating a new thread, specifically allocating temporary arrays for the data, which seems enough to make hyperthreading actually slow things down.

5. Discussion

This project started simply as an attempt to make Montgomery's light curve fitting program faster by allowing it to take full advantage of all of the computer's resources. The problem was an easy target for parallelization as each of the gridpoints was computed separately. With this increased speed, we might be able to actually try all the different possible l and m values for stars for the first time. I thought that I could make this happen by simply throwing all of the cores available in the CPU and GPU at the problem. In reality, however, this didn't work.

In order to really take advantage of the GPU, I needed to employ some algorithmic refinement, namely using a simpler model. When calculating this simpler model, the characteristics of the algorithm are completely flipped so that it runs significantly faster on a GPU. Furthermore, the simple model on the slowest tested GPU runs 19 times faster then the full treatment on the fastest CPU. The disparity between these two cases shows that any homogeneous hardware could not have performed as well on this program. Neither Hardware component preforms optimally for both tasks. When this program is run for Astronomical purposes, it will need to run both models. This poses a significant challenge for GAMA. As the program is currently designed, it will simply run one model at a time, switching between them frequently. GAMA handles this well, sending more work to the CPU for the complex model and more work to the GPU for the simple model. This shows that GAMA can effectively deal with irregular workloads.

GAMA was an extremely useful tool in parallelizing this application. It has a simple interface that allows users to relatively easily incorporate it into existing code. Once I included it, I was able to see how optimisatons in the GPU or CPU implementations of my algorithm effected its overall speed without having to work out new partitions of the workload across the CPU and GPU. It also allowed me to easily port this program to other computers, taking full advantage of their diverse hardware. Finally, GAMA was flexible enough for me to use it. It was designed to launch CUDA kernels directly, but it had an option to let me launch my own kernels, which I needed for the BOOST ODE solver.

With the new forms of hardware becoming more and more common, its important to know which algorithms can effectively use them. Languages like CUDA are still somewhat incomplete and it can be cumbersome to extend applications to the GPU. Fortunately, tools like Thrust and AMP, which attempt to allow users to include GPU code directly in C++, are beginning to allow the average programmer to include GPU calls in their algorithms. GAMA, takes this a step further, not only allowing users to write code that will run on multiple devices, but helping users figure out which devices their algorithm should run on. This kind of simplifying library is going to be key for programmers to continue to write

modular, platform independent, and above all, understandable programs on modern diverse and dynamically changing hardware.

REFERENCES

(1) Brickhill, A. The pulsations of zz ceti stars. v-the light curves. vi-the amplitude spectra. *Monthly Notices of the Royal Astronomical Society 259* (1992), 519–535.

(2) Butcher, J. A history of runge-kutta methods. *Applied numerical mathematics 20*, 3 (1996), 247–260.

(3) Davis, R. I., and Burns, A. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv. 43 4*, 35 (October 2011).

(4) Eric S. Chung, Peter A. Milder, J. C. H., and Mai., K. Single-chip heterogeneous computing: Does the future include custom logic, fpgas, and gpgpus? In *In Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 43* (2010), pp. 225–236. Washington, DC, USA.

(5) John D. Owens, Mike Houston, D. L. S. G. J. E. S., and Phillips, J. C. Gpu computing: Graphics processing units–powerful, programmable, and highly parallel–are increasingly targeting general-purpose computing applications. In *Proceedings of the IEEE* (May 2008), vol. 96.

(6) Kippenhahn, R., Weigert, A., and Weiss, A. *Stellar structure and evolution*. Springer, 2013.

(7) Lammers, D. Intel cancels tejas, moves to dual-core designs. *EETimes* (May 2004).

(8) Mariano, A. M. M. *Scheduling (ir)regular applications on heterogeneous platforms*. PhD thesis, University of Minho, Department of Computer Science, 2012.

(9) Montgomery, M. H. A new technique for probing convection in pulsating white dwarf stars. *The Astrophysical Journal* (November 2005).

(10) Moore, G. E. Cramming more components onto integrated circuits. *Electronics 11*, 8 (August 1991).

(11) Nvidia, C. Programming guide, 2012.

(12) Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. *Numerical Recipes in FORTRAN 77: Volume 1, Volume 1 of Fortran Numerical Recipes: The Art of Scientific Computing*, vol. 1. Cambridge university press, 1992.

(13) Winget, D., Hansen, C., Liebert, J., Van Horn, H., Fontaine, G., Nather, R., Kepler, S., and Lamb, D. An independent method for determining the age of the universe. *The Astrophysical Journal 315*, 2 (1987), L77–L81.

(14) Wu, Y. Combination frequencies in the fourier spectra of white dwarfs. *Monthly Notices of the Royal Astronomical Society 323*, 1 (2001), 248–256.

This preprint was prepared with the AAS LaTeX macros v5.2.

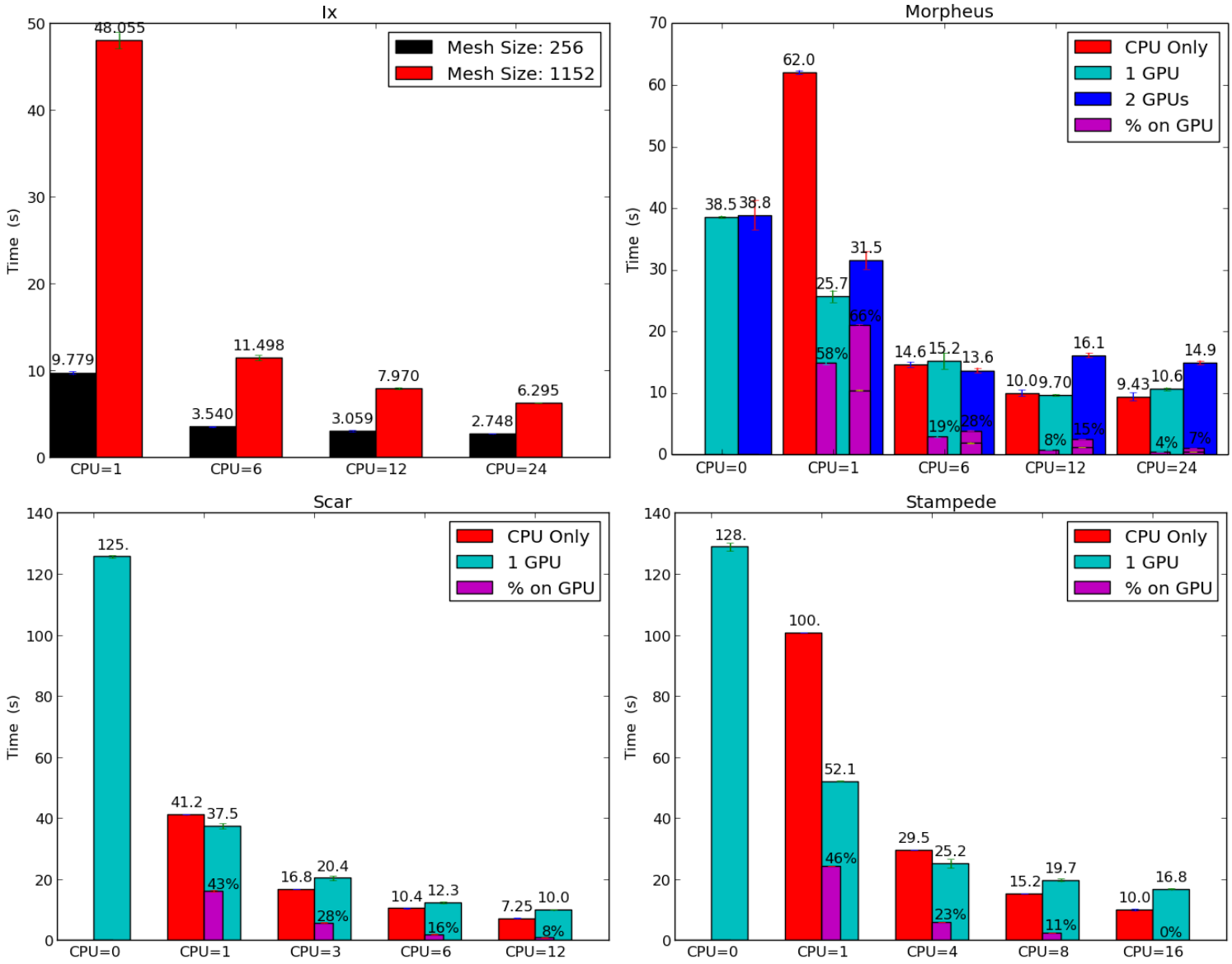

Fig. 3.— Run times for a full light curve computation using Montgomery's model on the four test computers. For the three with GPU(s), a mesh size of 1152 was used. Each test was ran thirty time and averaged

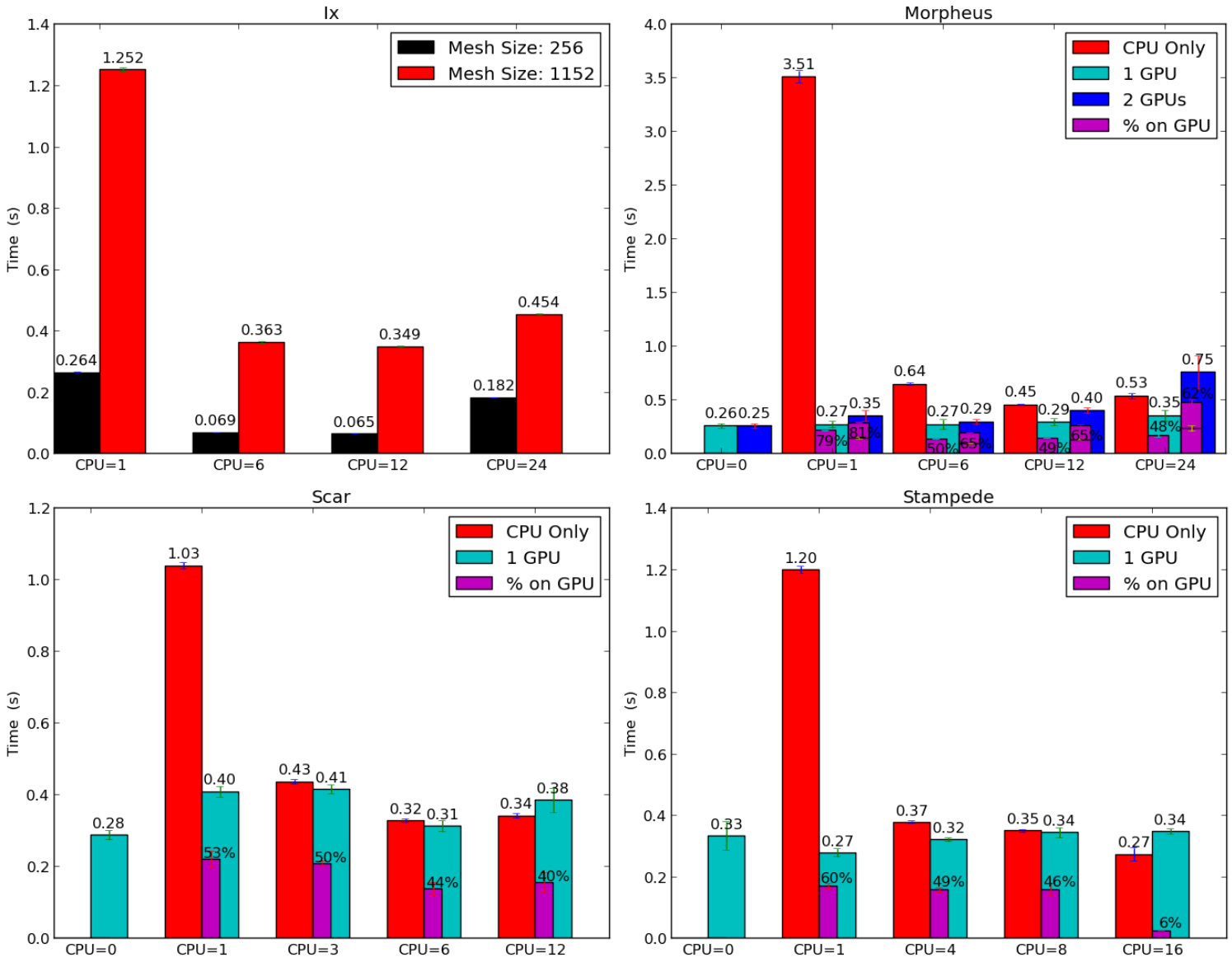

Fig. 4.— Run times for a full light curve computation using Wu model on the four test computers. For the three with GPU(s), a mesh size of 1152 was used. Each test was ran thirty time and averaged