EPJ Web of Conferences **109**, 06003 (2016)
DOI: 10.1051/epjconf/201610906003
© Owned by the authors, published by EDP Sciences, 2016

Efficient GPU Acceleration for Integrating Large Thermonuclear Networks in Astrophysics

Mike Guidry[1,a]

[1] *Department of Physics and Astronomy, University of Tennessee, Knoxville TN 37996 USA*

Abstract. We demonstrate the systematic implementation of recently-developed fast explicit kinetic integration algorithms on modern graphics processing unit (GPU) accelerators. We take as representative test cases Type Ia supernova explosions with extremely stiff thermonuclear reaction networks having 150-365 isotopic species and 1600-4400 reactions, assumed coupled to hydrodynamics using operator splitting. In such examples we demonstrate the capability to integrate independent thermonuclear networks from ~250–500 hydro zones (assumed to be deployed on CPU cores) in parallel on a single GPU in the same wall clock time that standard implicit methods can integrate the network for a single zone. This two or more orders of magnitude increase in efficiency for solving systems of realistic thermonuclear networks coupled to fluid dynamics implies that important coupled, multiphysics problems in various scientific and technical disciplines that were intractable, or could be simulated only with highly schematic kinetic networks, are now computationally feasible. As examples of such applications I will discuss our ongoing deployment of these new methods for Type Ia supernova explosions in astrophysics and for simulation of the complex atmospheric chemistry entering into weather and climate problems.

1 Introduction

Many physically-important systems can be modeled by a fluid dynamics plus reaction kinetics approximation. For example, in stellar structure and stellar evolution one often uses a formalism where the dynamical evolution of the stellar fluid is described by multi-dimensional hydrodynamics coupled to a thermonuclear reaction network describing element production and energy release, and in atmospheric simulations the evolution of the atmospheric fluid is coupled to chemical kinetics equations describing the chemical reactions occurring in the atmosphere.

In physically-realistic systems the coupling of the fluid dynamics to a realistic kinetic network is not feasible with present algorithms and computational power, and often a highly-truncated kinetic network is coupled to the fluid dynamics evolution. For example, the current state of the art for published Type Ia supernova simulations is illustrated in figure 1. In this example we see that the current state of the art uses a schematic network that is 10 times smaller than the minimal realistic network. Similar considerations apply in many fields where a realistic kinetic network requires hundreds of species but the most realistic networks routinely coupled to fluid dynamics are much smaller than

[a]e-mail: guidry@utk.edu

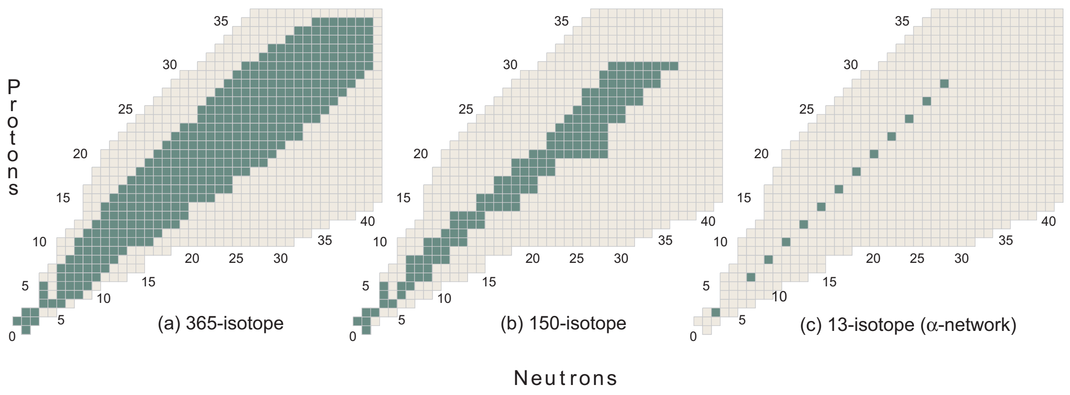

Figure 1. Typical thermonuclear networks that might be used for simulating Type Ia supernova explosions [1]. (a) A network containing the 365 isotopes that are populated with measurable intensity in the explosion. (b) A subset representing the 150 isotopes populated with sufficient intensity to influence the hydrodynamics. This is the minimal realistic network for coupling to hydrodynamical simulations of the Type Ia explosion. (c) The largest network that has typically been coupled to multidimensional hydrodynamics in a Type Ia simulation.

that. To incorporate realistic networks in astrophysical simulations we must improve (substantially) the speed and efficiency for computing kinetic networks coupled to fluid dynamics. There are two general approaches that we might take:

1. Improve the algorithms used to solve the kinetic networks.
2. Improve the hardware on which the algorithms are executed.

This presentation is about using both approaches to affect a dramatic improvement in the speed for solving large kinetics networks in astrophysics and other fields.

2 Stiff Equations

Two broad classes of numerical integration may be defined. In *explicit* numerical integration, to advance the solution from time t_n to t_{n+1} only information already available to the calculation at t_n is required. In *implicit* numerical integration, to advance the solution from t_n to t_{n+1} requires information at t_{n+1}, which is of course unknown. Thus implicit integration requires an iterative solution. Such solutions are expensive for large sets of equations because they involve matrix inversions.

Why then would one want to use implicit methods? The answer is *stability*, in particular for the integration of *stiff equations*. Systems of equations exhibit *stiffness* if multiple timescales differing by many orders of magnitude are an essential feature. Since most processes of physical importance involve changes occurring on more than one timescale, stiffness is a common feature in computer modeling of realistic physical systems. It is commonly understood that stiff systems cannot be integrated efficiently with explicit methods because the largest stable timestep is much too short, so the traditional view is that stiff systems require implicit numerical methods for their solution. Thus, in the standard view explicit methods are inherently simple, but potentially unstable, particularly for stiff systems, while implicit methods are inherently complicated but usually stable for stiff systems. We now wish to describe new explicit methods that have the potential to combine the desirable attributes

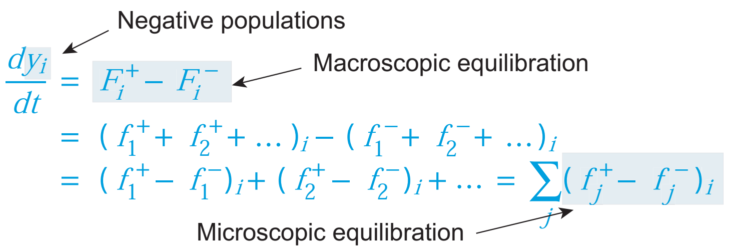
Negative populations
Macroscopic equilibration
$\frac{dy_i}{dt} = F_i^+ - F_i^-$
$= (f_1^+ + f_2^+ + \ldots)_i - (f_1^- + f_2^- + \ldots)_i$
$= (f_1^+ - f_1^-)_i + (f_2^+ - f_2^-)_i + \ldots = \sum_j (f_j^+ - f_j^-)_i$
Microscopic equilibration

Figure 2. Sources of stiffness in explicit integration of a set of coupled differential equations. *Negative probabilities* correspond to populations (which cannot be negative) being driven negative by numerical error because of a too-large timestep. This turns damped exponentials into growing exponentials and destabilizes the network. *Macroscopic equilibration* occurs when F_i^+ becomes approximately equal to F_i^-, so that one is taking numerically the tiny difference of two very large numbers. This too will destabilize the network if the explicit timestep is too large. *Microscopic equilibration* occurs when forward–reverse terms at the reaction level become almost equal. This again implies numerically taking the tiny difference of very large numbers, and will destabilize the network if the timestep is too large.

of both implicit and traditional explicit methods: These new *algebraically-stabilized explicit methods* will be shown to have the simplicity of an explicit method but with stability rivaling that of implicit methods [1–5].

3 Algebraically-Stabilized Explicit Integration

The key to stabilizing explicit integration is to understand that there are three basic sources of stiffness for a typical reaction network. We have termed these

1. Negative populations
2. Macroscopic equilibration
3. Microscopic equilibration

and they are illustrated in figure 2. We have developed a systematic set of algebraic constraints within the context of explicit numerical integration that remove these sources of stiffness and permit the algebraically-stabilized explicit method to take stable and accurate timesteps that are competitive with that of standard implicit methods; details may be found in Refs. [1–5]. Since the stabilized explicit method can execute each timestep faster for large networks (no matrix inversions), the resulting algorithm is intrinsically faster than implicit algorithms.

In figure 3 and 4 we illustrate the new method in action for a nova simulation. Because the timestepping for the algebraically-stabilized explicit integrations (labeled Asy and QSS for two somewhat different implementations) is competitive with that of the implicit-code reference, the algebraically-stabilized explicit method exhibited a speed $\sim 5-10$ times faster than that of the implicit code for this problem. In Refs. [1–5] we have documented a number of such tests of the new explicit approach and the intrinsic speedup of the algorithm over state of the art implicit methods is summarized in figure 5.[1]

[1]We use a standard implicit code running on a CPU as the benchmark because this is the current state of the art for large-scale astrophysical simulations. It remains to be seen whether implicit codes can be made substantially more efficient with GPU acceleration. As of this writing the implicit code we are using for comparison [6] runs at most several times faster on a GPU than a CPU for a 150-isotope network [7].)

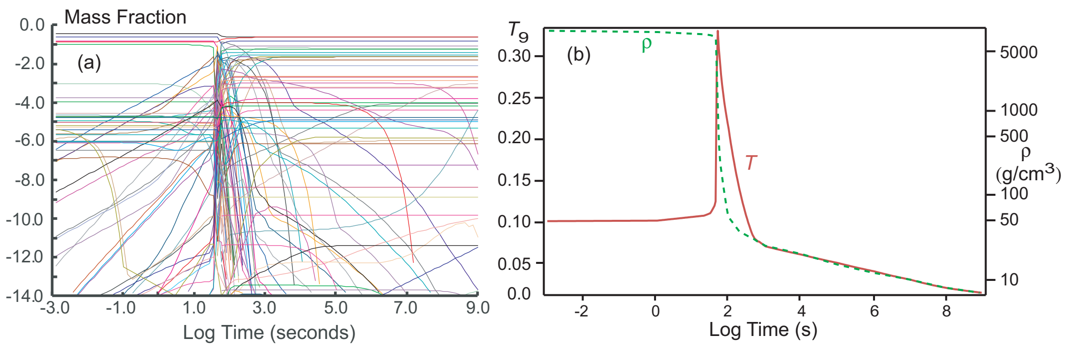

Figure 3. A nova simulation [1]. (a) Mass fractions as a function of time. (b) The hydrodynamical temperature and density profile used for the simulation.

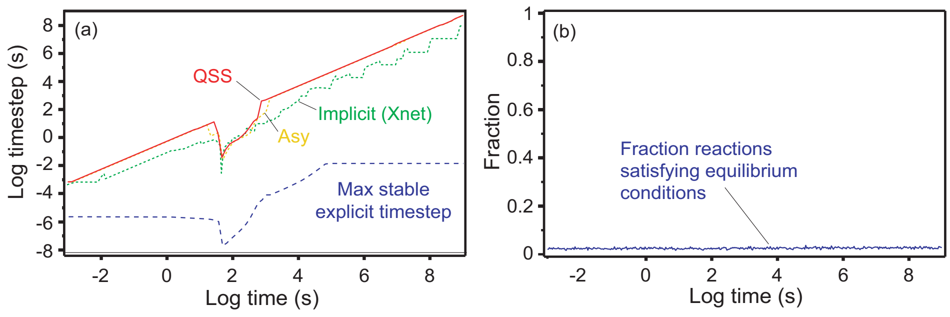

Figure 4. (a) The integration timestepping for the nova simulation of figure 3 for a standard implicit method (in this and other discussion in this paper we use the implicit backward Euler code Xnet [6] for comparison), and for two implementations of the algebraically-stabilized explicit method (labeled Asy and QSS) [1]. (b) Fraction of reactions satisfying the microscopic equilibration condition in the calculation.

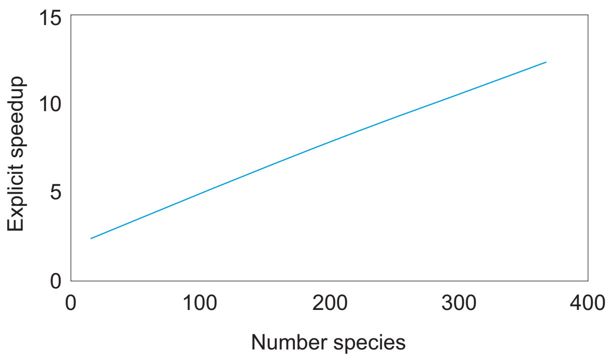

Figure 5. The approximate intrinsic speedup factor as a function of network size for the new explicit methods relative to standard implicit methods. Adapted from results presented in Refs. [1–4].

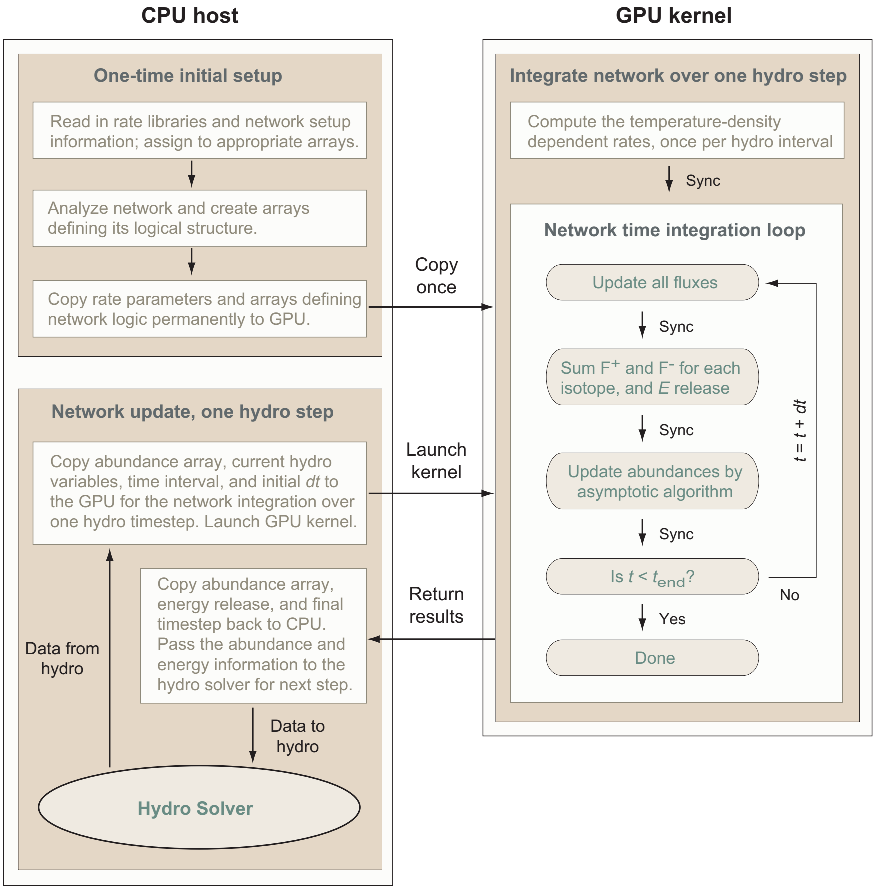
CPU host
GPU kernel
One-time initial setup
Read in rate libraries and network setup information; assign to appropriate arrays.
Analyze network and create arrays defining its logical structure.
Copy rate parameters and arrays defining network logic permanently to GPU.
Copy once
Integrate network over one hydro step
Compute the temperature-density dependent rates, once per hydro interval
Sync
Network time integration loop
Update all fluxes
Sync
Sum F+ and F- for each isotope, and E release
Sync
Update abundances by asymptotic algorithm
Sync
Is t < tend?
No
Yes
Done
t = t + dt
Network update, one hydro step
Copy abundance array, current hydro variables, time interval, and initial dt to the GPU for the network integration over one hydro timestep. Launch GPU kernel.
Launch kernel
Copy abundance array, energy release, and final timestep back to CPU. Pass the abundance and energy information to the hydro solver for next step.
Return results
Data from hydro
Data to hydro
Hydro Solver

Figure 6. Flow for calculating a single kinetic network assumed to be coupled to hydrodynamical integration by operator splitting [5]. Kinetic network integration over a time interval corresponding to one hydro timestep is executed entirely on the GPU. Once the problem is set up, the only communication between CPU and GPU is to copy data from the last hydro timestep to the GPU (a few hundred floating point numbers), launch the network integration kernel, and then copy the network integration results back to the CPU (a few hundred floating point numbers). At points labeled "Sync" in the GPU kernel, all threads must be synchronized before proceeding.

4 Leveraging Modern Hardware: GPU Acceleration

The simplicity of the new explicit algorithms makes them particularly amenable to optimization for modern hardware such as Graphics Processing Unit (GPU) accelerators. GPUs are characterized by the availability of many lightweight threads with access to limited amounts of fast memory. If algorithms can be adapted to that environment, substantial parallel speedup may be achieved. The new explicit algorithms can be formulated in a highly parallel way and require minimal CPU–GPU communication, so they may be attractive candidates for GPU acceleration.

We have implemented the algebraically-stabilized explicit asymptotic algorithm [2] using CUDA C/C++ on a variety of GPUs with both Fermi and Tesla architectures. The schematic implementation for a single network is shown in figure 6, and timing for sample GPU calculations is illustrated in

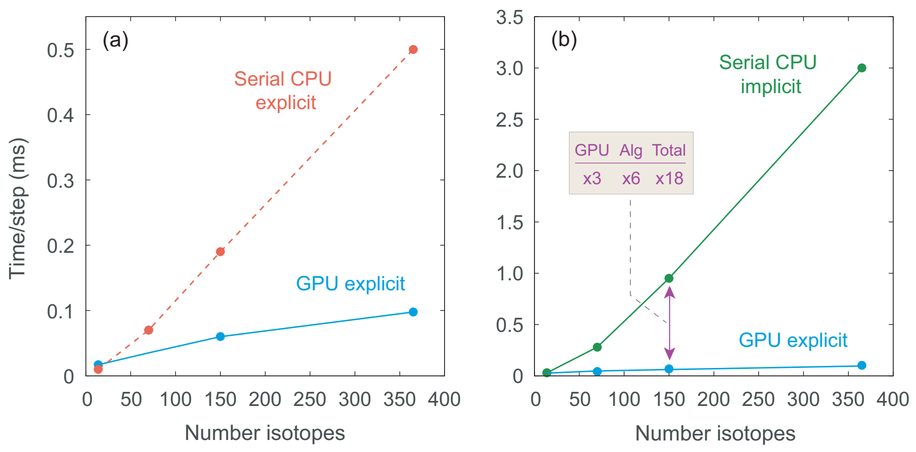

Figure 7. Time to execute a single integration step in milliseconds for one network as a function of the number of isotopes in the network. (a) GPU implementation of the explicit asymptotic algorithm (solid blue curve) versus CPU serial implementation of the same algorithm (dashed red curve). (b) GPU implementation of the explicit asymptotic algorithm (solid blue curve) versus serial implementation of a standard backward Euler implicit method (dashed green curve). The implicit curve was estimated using the scaling factors F determined in Ref. [2] to scale the explicit serial CPU curve in (a). See the text for further explanation. All calculations assumed Type Ia supernova conditions with a constant temperature of 7×10^9 K and a constant density of $10^8 \mathrm{g\,cm^{-3}}$. GPU calculations were run on a Tesla M2090 Fermi architecture; CPU calculations were run on a 3 GHz Intel processor.

figure 7. There we see that for a representative 150-isotope network the GPU implementation runs about 18 times faster than a standard implicit code on a CPU. In the figure we have broken that factor of 18 down into the product of a factor of approximately 6 coming from the new algorithm and a factor of approximately 3 coming from the increased parallelism of the GPU implementation.

A factor of 18 speedup over current state of the art for thermonuclear networks coupled to hydrodynamics is impressive, but in reality the GPU running a single kinetic network is highly underutilized: most of its threads sit idle while it integrates one network. Therefore, we have investigated launching many networks running in parallel on a single GPU, as illustrated schematically in figure 8 [5]. Results are displayed in figure 9. There we see that the new algorithms implemented on a modern GPU are capable of running $\sim 250 - 500$ realistic (150-isotope) networks in the same length of time that a standard implicit code can run one such network on a CPU. The reasons for this massive increase in efficiency are documented partially in Refs. [1–5], and will be documented fully in forthcoming publications, but in essence the increased efficiency stems from

1. The greater speed of the algebraically-stabilized explicit algorithms relative to implicit algorithms for large networks.

2. Leveraging GPU parallelism for computing each network.

3. Maximizing GPU thread occupancy by launching many independent networks in parallel on the same GPU.

4. A restructuring of storage on the GPU to optimize thread memory access.

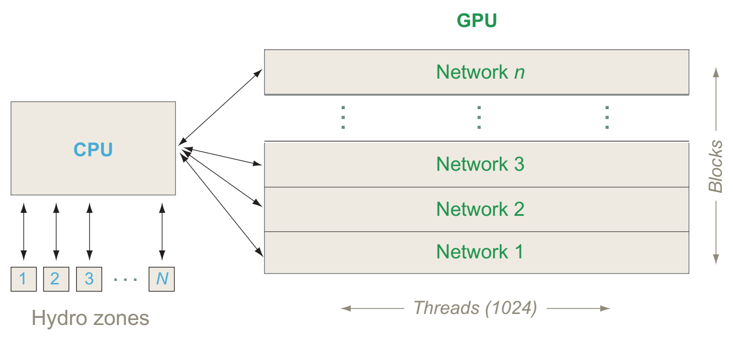

Figure 8. Integrating multiple kinetic networks in parallel on a GPU by stacking one network per block [5].

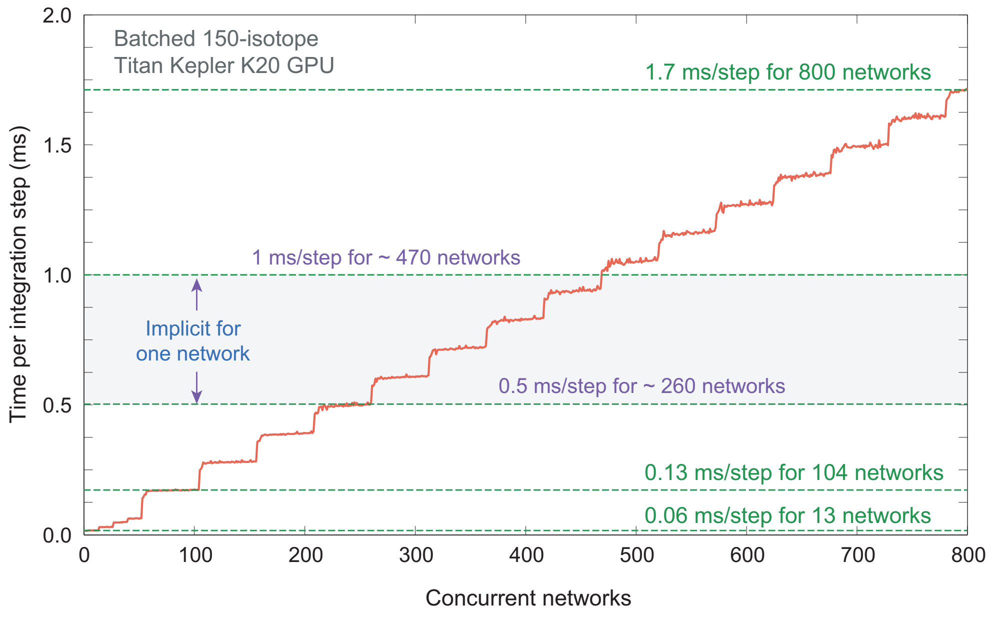

Figure 9. Massively parallel integration of many 150-isotope networks on a Tesla GPU. The approximate range of integration times for integrating a single network on a CPU with current implicit codes is indicated by the horizontal band. In this example the GPU has available 13 streaming multiprocessors and we have used batched methods to launch 4 networks per multiprocessor. This accounts for the step structure with width $4 \times 13 = 52$ networks.

5. The use of batched algorithms [8] to launch multiple network integration kernels for each GPU streaming multiprocessor.

We believe that there remains another factor of two or more in optimization yet to be realized in the algorithms discussed here. For example, we have shown that the present examples, which were done is double precision, are stable and accurate enough for coupling to fluid dynamics in single precision.

Thus we expect that with further improvement we can run $\sim$ 1000 networks in parallel on a single GPU in the time it presently takes for one implicit simulation of the same network to run on a CPU.

5 Summary

In summary, our new algebraically-stabilized explicit algorithms are intrinsically faster than standard implicit algorithms by factors of 5-10 for networks with several hundred species, primarily because they require neither matrix inversions nor interations. For a single network, GPU acceleration increases this to a factor of 20-40 for networks with several hundred species. The GPU is capable of running 250-500 networks in parallel in about the same length of time that a standard implicit code can run one such network on a CPU (this is also true presently within a factor of two or so for an implicit code accelerated with a GPU). These potential orders of magnitude increases in computational efficiency imply that much more realistic kinetic networks coupled to fluid dynamics are now feasible in a broad range of large problems in astrophysics and other disciplines. Presently we are investigating applications of these new methods to large-scale computer simulation for Type Ia supernovae, neutrino transport in core-collapse supernovae, real-time atmospheric forecasting, climate science, and materials science.

I would like to thank my collaborators: Azzam Haidar, Ben Brock, Daniel Shyles, Jay Billings, and Andrew Belt. Their contributons were essential in developing the material that I have presented in this paper.

References

[1] Mike Guidry, J. Comp. Phys. **231**, 5266-5288 (2012). [arXiv:1112.4778].

[2] M. W. Guidry, R. Budiardja, E. Feger, J. J. Billings, W. R. Hix, O. E. B. Messer, K. J. Roche, E. McMahon, and M. He, Comput. Sci. Disc. **6**, 015001 (2013) [arXiv: 1112.4716].

[3] M. W. Guidry and J. A. Harris, Comput. Sci. Disc. **6**, 015002 (2013) [arXiv: 1112.4750]

[4] M. W. Guidry, J. J. Billings, and W. R. Hix, Comput. Sci. Disc. **6**, 015003 (2013) [arXiv: 1112.4738]

[5] B. Brock, A. Belt, J. J. Billings, and M. W. Guidry, accepted for publication, J. Comp. Phys. (2015) [arXiv:1409.5826]

[6] Hix W R and Thielemann F-K 1999 Computational methods for nucleosynthesis and nuclear energy generation *J. Comp. Appl. Mathematics* **109** 321-51

[7] J. A. Harris; private communication

[8] Dong, Tingxing, Haidar, Azzam, Luszczek, Piotr, Harris, J. Austin, Tomov, Stanimire, Dongarra, Jack. (2014). LU Factorization of Small Matrices: Accelerating Batched DGETRF on the GPU. Paper presented at the 16th IEEE International Conference on High Performance Computing and Communications (HPCC 2014), Paris, France, August 20-22, 2014.