Toward Microscopic Equations of State for Core-Collapse Supernovae from Chiral Effective Field Theory

THESIS

Presented to the Faculty of the Department of Physics and Astronomy
in Partial Fulfillment of the Major Requirements
for the Degree of

BACHELOR OF SCIENCE IN
PHYSICS

Toward Microscopic Equations of State for Core-Collapse Supernovae from Chiral Effective Field Theory

Signature of Author:

____

Department of Physics and Astronomy
May, 2018

Certified by:

____

Dr. Jeremy W. Holt
Professor of Physics & Astronomy
Thesis Supervisor

Accepted by:

____

Dr. Ronald Henderson
Professor of Physics & Astronomy
Chair, Physics & Astronomy

ABSTRACT

Chiral effective field theory provides a modern framework for understanding the structure and dynamics of nuclear many-body systems. Recent works have had much success in applying the theory to describe the ground- and excited-state properties of light and medium-mass atomic nuclei when combined with ab initio numerical techniques. Our aim is to extend the application of chiral effective field theory to describe the nuclear equation of state required for supercomputer simulations of core-collapse supernovae. Given the large range of densities, temperatures, and proton fractions probed during stellar core collapse, microscopic calculations of the equation of state require large computational resources on the order of one million CPU hours. We investigate the use of graphics processing units (GPUs) to significantly reduce the computational cost of these calculations, which will enable a more accurate and precise description of this important input to numerical astrophysical simulations.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

The mass of a star is a fundamental factor that determines its evolution. More massive stars burn through their fuel more quickly than less massive ones. Stars live by maintaining a balance between gravity, which acts to make the star collapse on itself, and pressure from the heat release by fusion in the core of the star, where hydrogen atoms are fused together to produce helium atoms. Stars that are 10 – 30 solar masses are able to fuse nuclei up to iron. However, the fusion process stops at iron because fusing iron is not energetically favorable. It requires the input of energy instead of releasing energy. After reaching an iron core, these 10 – 30 solar mass stars end their life in a violent and spectacular event we call a supernova [3].

Initially, the iron core in the star is supported against gravitational collapse by electron degeneracy pressure. Once the iron core reaches the Chandrasekhar limit, 1.4 solar masses, the star becomes unstable and gravity takes over and causes the star to collapse. Short-range nuclear forces halt the collapse of the star, which creates a shockwave that rebounds outward. In the core, electron capture occurs, where an electron and a proton come together to make a neutron and electron neutrino. These neutrinos carry the majority of the energy from the core outward and leave behind a dense core full of neutrons. A combination of the shockwave and the neutrinos travelling outward could yield to a potentially successful supernova. What remains behind is a neutron star or, in an extreme case, black hole [3].

The conditions that yield a successful supernova are still not well understood. We are not able to directly examine the physical processes inside the core of a star that produce a supernova. To gain a better understanding of these processes, we need to rely on accurate and efficient numerical simulations of core-collapse supernovae. Fortunately, we have achieved great advancement in numerical algorithms and computational resources that will help achieve accurate and efficient simulations. In our present time, graphics processing units (GPUs) are being used to optimize and speedup programs and simulations that require a long runtime.

Producing accurate simulations of core-collapse supernovae requires the input of many ingredients that are often very computationally intensive, such as neutrino transport, electron capture during core-collapse, etc. One important ingredient is the microphysical equation of state of nuclear matter. Simulations that use realistic nuclear equations of state have demonstrated promising results, which indicate the importance of these equations of state to the physics of core-collapse and even to times after the collapse. Microscopic calculations of the equation of state require large computational resources due to the large range of densities, temperatures, and proton fractions that need to be considered during stellar core collapse. To date, this has prohibited the use of microscopic equations of state in core-collapse simulations, and instead only phenomenological equations of state have been implemented [4].

In this study, we aim to extend the application of chiral effective field theory to describe the nuclear equation of state required for supercomputer simulations of core-collapse supernovae. Chiral effective field theory is an approximation to quantum chromodynamics, which provides a modern framework for understanding the structure and dynamics of nuclear many-body systems. We use sophisticated nuclear two- and three-body forces to calculate the equation of state in many-body perturbation theory.

II. Methods

In this study, we investigate the use of GPUs to speed up codes for computing the third order perturbative contribution to the ground state energy density of nuclear matter. This is one of the inputs needed to generate an accurate equation of state of nuclear matter for simulations of core-collapse supernovae. The code before our study ran serially on central processing units (CPUs). It was optimized to utilize all the computing power CPU processing offers, but faster computational methods are more desirable.

GPU processing is explored in this project to potentially offer a significant speedup to the current best optimized version of the serial code. The main reason why GPUs could offer the speed up needed lies in the way that they differ from CPUs in processing computational tasks. CPUs consists of a few cores that are intended to process tasks sequentially, which could potentially take a very long time when dealing with millions of calculations. On the other hand, GPU's consist of many small cores that are designed to do tasks in parallel. This allows for multiple tasks to be executed simultaneously. However, GPUs are inefficient at accessing memory. Whereas CPUs can quickly access hundreds of Mbytes of RAM, GPUs are limited to the Kbytes of easily accessible memory. Here, we attempt to harness the potential speedup offered by the parallelism feature of GPUs to execute large amounts of computations needed for our serial code.

We are able to compute the energy density of nuclear matter using perturbation theory. Unlike simple systems such as the hydrogen atom and the harmonic oscillator, either there is not a Hamiltonian which we could use to find the eigenvalues and eigenstates of an interacting system, or the Hamiltonian for the system is too complicated to solve. In some cases, the interacting system is very close to a system whose Hamiltonian we know, such as the hydrogen atom. We are then able treat the small differences between the two systems as perturbations and approach them in a systematic way. The Hamiltonian of the system will then consist of two parts, the perturbed and unperturbed Hamiltonian. The eigenvalues and the eigenstates of the system could then be described as a power-series expansion, and for the expansion to be useful, successive terms in the series must grow smaller [5]. In our case, we know the Hamiltonian associated with the nuclear force. The problem is that the Hamiltonian is too complicated to solve for the eigenvectors and eigenvalues of an interacting many-body system. Therefore, we start with a system we can solve, noninteracting Fermions, and perturbatively build in the nuclear force as corrections. We then get the following free energy perturbation series

$$\bar{F}(T,\rho,\delta) = \bar{F}_o(T,\rho,\delta) + \lambda\bar{F}_1(T,\rho,\delta) + \lambda^2\bar{F}_2(T,\rho,\delta) + \lambda^3\bar{F}_3(T,\rho,\delta) + \mathcal{O}(\lambda^4)$$

where T is temperature, ρ is the total nucleon density, and δ is the isospin-asymmetry parameter [2]. The diagrammatic contributions to the ground-state energy density of isospin-symmetric nuclear matter is shown below

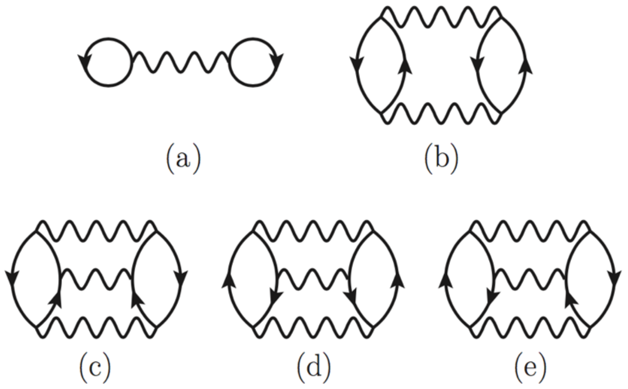

Figure 1: First-, second-, and third- order diagrammatic contributions to the ground state energy density of nuclear matter. The wavy lines indicate the (antisymmetrized) density-dependent NN interaction derived from chiral two- and three-body forces [1].

The code for computing the third order perturbative contribution to the ground state energy density of nuclear matter needs to compute a lot of integrals using Gaussian Quadrature Numerical Integration Method. This method involves using many nested for-loops to compute the integrals, which could potentially make the code run for hours. The amount of time the code runs depends on the precision desired for the integrals being computed. We aim to develop a basic algorithm that will offer a considerable computational speedup.

The serial program contains thousands of lines of code. We were only interested in a group of nested for-loops, that were used within the code. These nested for-loops needed to traverse and perform computations on arrays that were up to six dimensions, and each array could have hundreds of thousands of elements. This is the type of problem that GPUs could handle efficiently. Our algorithm needs to parallelize nested for-loops, which would allow access to thousands of elements at the same time and performing mathematical operations on them simultaneously.

CUDA, a parallel processing platform invented by NVIDIA, is used to parallelize portions of the code. Our aim is to create a subroutine in CUDA that contains the parallelization algorithm. This subroutine could then be called in the serial code by passing the right parameters into it. In doing so, we have a generalized subroutine that could be called in any program we want with the appropriate modifications.

Most of this project focused on parallelizing Guassian Quadrature Numerical Integration Method. We wanted to approach parallelizing the numerical integration method by starting with a simple case and then gradually move on to more complicated cases. This allowed us to enhance our understanding of how the CUDA platform works, and the program was easier to troubleshoot. These troubleshooting skills were then extended to more complicated versions of the Gaussian Quadrature Numerical Integration Method.

The first and easiest case we investigated was numerically integrating a one-dimensional function. This simply means that we had to only worry about a function that depended on only one variable, which in computer language means that we only needed a one-dimensional array. This simplified parallelizing the integration method since the parallelization process was reduced from being multidimensional to only one-dimensional.

As mentioned before, parallelization is about how to use the GPU memory available to accomplish a task in an efficient manner. The CUDA memory structure consists of grids which contain blocks of memory, which contain memory threads. Now, the grids could be two-dimensional, x- and y- dimensions. The blocks in each grid could be three-dimensional, x-, y-, and z-dimensions. This corresponds to having three different types of threads in a block, x-, y-, and z-threads. This leads to a very intricate way of accessing data on the GPU and carrying out the calculations needed.

The above brief layout of the memory structure in CUDA will allow us to understand what a one-dimensional parallelization of the integration method entails. Since we are only dealing with an array that has one-dimension, we only need memory blocks that are one dimensional to access the data within the array. So, once the array of interest is copied on the GPU, we could access its element via threads. This is possible by using functions that are pre-defined in the CUDA platform. For example, in the one-dimensional case, we could use blockIdx.x and threadIdx.x to traverse through the one-dimensional array. BlockIdx.x tells us which memory block we are accessing, and the threadIdx.x tells us which thread has access to which element in the memory block we are accessing.

Theoretically, the one-dimensional algorithm is sufficient enough to be implemented in the program of interest. There are two main reasons why we would want to expand our algorithm to more than one-dimension. The first reason is that the program with which we are working has up to six-dimensional arrays. It would be a much easier implementation if we are able to handle multidimensional arrays. The second reason is that by using all the available dimensions in a block, we might be able to get a much more efficient algorithm. As of right now, we are only using the x-dimension and there are two more dimensions available for us to use.

We started to investigate implementing a two-dimensional algorithm for the numerical integration method. This was a challenging task, since now we have to also worry about a y-dimension in addition to the x-dimension. It took a lot of time to figure out how to properly traverse a two-dimensional array on GPU. This requires thinking about how to make thread and block indices of each dimension communicate with each other and read information from the array properly and in the necessary order. Many ideas were attempted and finally we were able to accomplish the goal of developing a two-dimensional algorithm. The next thing was to introduce a new dimension to the algorithm. At this point things started to get a little abstract and thinking about the memory structure became more complex and less intuitive. A three-dimensional algorithm was developed, but the third dimension was computed serially. So, we did not anticipate a large speedup in the code.

The serial code contained up to six dimensional arrays, which allowed for better efficiency when run on CPU. As mentioned above, our parallelization algorithm could only handle three dimensional arrays, with the third dimension computed serially. We faced difficulty when we tried to implement our three-dimensional algorithm after reducing the arrays in the serial code to a maximum of three dimensions. Dr. Jeremy Holt flattened all array to only one dimension, which allowed for a much easier implementation of our one-dimensional algorithm.

III. Results and Conclusions

The results of the parallelized program vs. the serial program are summarized in figure 3. The plot shows the time it takes a program to run on CPU vs. GPU as a function of mesh points. These mesh points could be thought of as precision points. The more precise you would like the final answer to be, the more mesh points you need. Also, the higher precision points require more computational time. We could see that the GPU code performs much better than the CPU code at any given number of mesh points, and especially when we get to larger mesh points. This is partially due to the fact that GPU can accommodate to higher memory demand than CPU can.

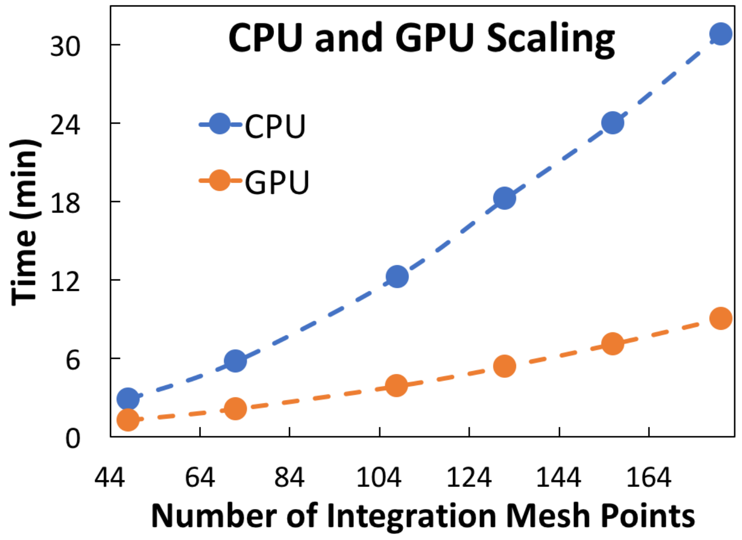

Figure 3: Comparison between CPU and GPU performance

Toward the end, we tried to see if we can make the program more efficient by using more sophisticated parallelization algorithms. These algorithms were complicated, and a lot of time would have been needed to understand them. We were approaching the end of our research time, so we decided that the results we obtained were good, and future optimization to the code would be needed.

IV. REFERENCES

1. J. W. Holt and N. Kaiser, "**Equation of State of Nuclear and Neutron Matter at Third-Order in Perturbation Theory from Chiral Effective Field Theory**", Physical Review C **95** , 034326 (2017).

2. C. Wallenhofer, J. W. Holt, and N. Kaiser, "**Thermodynamics of Isospin-Asymmetric Nuclear Matter form Chiral Effective Field Theory**", Physical Review C **92**, 015801 (2015).

3. Müller B., Hüdepohl L., Marek A., Hanke F., Janka HT. (2012) The SuperN-Project: Neutrino Hydrodynamics Simulations of Core-Collapse Supernovae. In: Nagel W., Kröner D., Resch M. (eds) High Performance Computing in Science and Engineering '11. Springer, Berlin, Heidelberg

4. Hix W. R., Lentz E. J., Endeve E. *et al* 2014 *AIPA* **4** 041013

5. "**Time-Independent Perturbations.**" *A Modern Approach to Quantum Mechanics*, by John S. Townsend, University Science Books, 2012, pp. 381–382.