University of Tennessee, Knoxville

TRACE: Tennessee Research and Creative Exchange

Masters Theses

Graduate School

5-2014

Verification of the Shift Monte Carlo Code Using the C5G7 and CASL Benchmark Problems

Nicholas Cameron Sly
University of Tennessee - Knoxville, nsly@utk.edu

Follow this and additional works at: https://trace.tennessee.edu/utk_gradthes

 Part of the Nuclear Engineering Commons

Recommended Citation

Sly, Nicholas Cameron, "Verification of the Shift Monte Carlo Code Using the C5G7 and CASL Benchmark Problems. " Master's Thesis, University of Tennessee, 2014.
https://trace.tennessee.edu/utk_gradthes/2779

This Thesis is brought to you for free and open access by the Graduate School at TRACE: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Masters Theses by an authorized administrator of TRACE: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

To the Graduate Council:

I am submitting herewith a thesis written by Nicholas Cameron Sly entitled "Verification of the Shift Monte Carlo Code Using the C5G7 and CASL Benchmark Problems." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Nuclear Engineering.

Guillermo I. Maldonado, Major Professor

We have read this thesis and recommend its acceptance:

Lawrence H. Heilbronn, Ronald E. Pevey

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

University of Tennessee, Knoxville
Trace: Tennessee Research and Creative Exchange

Masters Theses | Graduate School

5-2014

Verification of the Shift Monte Carlo Code Using the C5G7 and CASL Benchmark Problems

Nicholas Cameron Sly
University of Tennessee - Knoxville, nsly@utk.edu

This Thesis is brought to you for free and open access by the Graduate School at Trace: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Masters Theses by an authorized administrator of Trace: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

To the Graduate Council:

I am submitting herewith a thesis written by Nicholas Cameron Sly entitled "Verification of the Shift Monte Carlo Code Using the C5G7 and CASL Benchmark Problems." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Nuclear Engineering.

Guillermo I. Maldonado, Major Professor

We have read this thesis and recommend its acceptance:

Lawrence H. Heilbronn, Ronald E. Pevey

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Verification of the Shift Monte Carlo Code Using the C5G7 and CASL Benchmark Problems

A Thesis Presented for

The Master of Science

Degree

The University of Tennessee, Knoxville

Nicholas Cameron Sly

May 2014

© by Nicholas Cameron Sly, 2014
All Rights Reserved.

I would like to dedicate this thesis to my parents for their invaluable encouragement and support.

Acknowledgements

I would like to thank the Shift team at Oak Ridge National Laboratory for the opportunity to work on this project and for all their help: John Wagner, Tom Evans, Scott Mosher, and Brenden Mervin. I'd also like to thank my advisor G. Ivan Maldonado for providing direction and sharing his experience. Shane Hart is also deserving of a great deal of gratitude for helping me learn my way around all of the computing systems and codes. I also want to thank Carolyn McGraw for her support and assistance. Finally, I'd like to thank my parents for their unerring support.

"When it is not in our power to determine what is true, we ought to follow what is most probable." –René Descartes

Abstract

While Monte Carlo simulation has been recognized as a powerful numerical method for use in radiation transport, it has required a mixture of methods development and hardware advancement to meet these expectations in practical applications. In an effort to continue this advancement for uses of Monte Carlo simulation in ever larger capacities, Oak Ridge National Laboratory is developing the Shift hybrid deterministic/Monte Carlo code to be massively-parallel for use on parallel computing systems of all sizes. As part of this development, verification of the Monte Carlo parts of the code is needed to confirm that the current version of the code is operating properly, by matching the results of similar, currently available codes, as well as allowing for testing of the code in the future, to ensure that subsequent code changes and the implementation of new capabilities don't adversely affect the results. This research starts that verification using some basic reactor criticality benchmarks. The Shift code has been shown to agree within three standard deviations with MCNP and KENO, two of the most widely used Monte Carlo criticality codes. Also investigated was the efficiency of the Shift code as it currently stands, scaling with the number of processors the code is run on as well as the number of particles being simulated. The code was found to scale well, as long as there are enough particles to make the transport take significantly more time than the inter-cycle communication between compute nodes.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

As early as the Manhattan Project, Monte Carlo simulation has been recognized as a powerful numerical method (3). However, at the time it was infeasible to use Monte Carlo for modeling radiation transport phenomena with any level of detail. As a consequence, many simplifications were made early in the history of radiation transport to accommodate the computational capabilities of the times. Generally, advances in nuclear simulation follow and take advantage of advances in computing (6). The availability of advanced computational methods notwithstanding, ever-increasing processing speeds have also influenced the utility of various radiation transport methods. However, as codes were borne out of one advancement in computing, they would subsequently need to be retrofitted to utilize later advancements or risk becoming irrelevant. This has been attempted several times with various codes and with mixed results. The ever-present alternative is to design a new code centered around fully utilizing the new capability.

This alternative approach has been explored by researchers at Oak Ridge National Laboratory (ORNL), giving rise to the Denovo project and, more specifically, the Shift hybrid deterministic/Monte Carlo code. The code is being built from the ground up to be massively parallel by utilizing the best available techniques while overcoming the challenges those techniques present (15). As part of the Denovo project, Shift

will combine Monte Carlo transport with a deterministic transport algorithm as a hybrid method in order to reduce the total computation time required to acquire an accurate result by utilizing the advantages of both techniques.

Once this new code has enough capabilities, these capabilities must be verified and reverified during later stages of development. This is done to demonstrate that each new capability performs its job appropriately while, at the same time, ensuring it does not break any of the previous capabilities of the code. For a Monte Carlo program, this generally involves utilizing benchmarks that have either been checked against experimental results or simulated using codes that have been previously validated against other experimental results. This research conducts such a verification on the eigenvalue estimation, flux estimation, and the scaling capabilities of the Monte Carlo code Shift.

Chapter 2

Background

2.1 Early Uses and Constraints of Monte Carlo Radiation Transport Codes

Monte Carlo has been regarded since the Manhattan Project as a powerful numerical method that would be useful in the simulation of radiation transport. This is due to its generality (2). So long as the geometry of a given simulation can be described mathematically, and the physical processes can be represented with equations or probability distributions, the Monte Carlo method can simulate almost any radiation transport problem. However, Monte Carlo has historically been considered "complicated, inefficient, and expensive" (3) and because of this, has been utilized as a "method of last resort" (10). Several reasons have been suggested for this (3). First, Monte Carlo was used when other methods couldn't be. Thus, it was used on problems that had complicated geometries and physics that were too difficult to describe with partial differential equations in deterministic calculations. Second, the use of statistical methods added a new source of numerical error. Variance reduction techniques were derived in order to reduce this error to acceptable levels during a practical run time; however, the application of these techniques has often been considered less of a science than an art. Third, the use of the law of large numbers

required that the number of histories run be increased according to the square of the desired reduction in statistical error. Thus, in order to have a great deal of certainty in a given result, the programs would need to run a very large number of histories.

2.2 Utilizing Advances in Computation

With these problems in mind, Monte Carlo was restricted in the areas and ways in which it was applied. Due to memory restrictions, complicated geometries could only be represented on large machines. Also, cross section files of limited size had to be utilized. Large and detailed geometries had the added complication of requiring more histories to get a reasonable degree of certainty in the results. Because of this, Monte Carlo methods were primarily utilized for performing detailed simulations at the pin cell level as a means of determining the flux spectrum, which would subsequently be used to collapse cross sections for use in less detailed diffusion simulations (1). For other reasons, including the inherent difficulty in accounting for thermal hydraulic feedback, a task easily handled by diffusion-based codes, deterministic radiation transport was also limited to this role. While diffusion simulations are faster and easier, they are inherently error prone due to the assumptions made to justify the use of the diffusion equation. This resulted in many years of research to find ways to reduce the errors in the diffusion results. At the same time, research and advances in computing methods and power have led to an enormous increase in the availability of Monte Carlo radiation transport methods to the average researcher, as well as its feasibility for use with larger problems. Aside from the general increases in processing speed, two main computational advances have led to the re-imagining of Monte Carlo codes in an attempt to utilize them: vectorization and parallelization.

While scalar processing power increased through the 1950s and 1960s, vectorization provided the next great leap in the 1970s (figure 2.1). Before vectorization, scalar processing computers would take one piece of data and one instruction of how to manipulate it, even if the same instruction was being used for multiple pieces of data in

a row. Vectorization allowed for a multiple pieces of data to be sent to the processors with a single instruction to perform on all of them, resulting in all of the calculations being completed much quicker. The key to optimizing processor efficiency is to fill the vector as much as possible before sending it through the processor. However, the way in which Monte Carlo codes had historically been built was not suited for use on vector processors. At the time, Monte Carlo codes were designed to model a single particle, following it from collision to collision until it was absorbed or left the boundary of the problem - an approach known as history-based Monte Carlo. Attempting to vectorize this would lead to starting with a full vector of particles, but having some particles absorbed before others and ending up with an incomplete vector going through the processor. Many attempts were made to vectorize many history-based programs, including KENO-IV. Almost all of them resulted in increased computation time and those that didn't only resulted in small decreases (4). In 1973, Martin and Brown published an alternative method they referred to as "event-based Monte Carlo" (2). In this approach, the simulation was divided into different processing events such as collision analysis, tallying, tracking, and boundary crossing. When a given event had a full vector, or whichever vector was closest to being full, it would be processed, contributing to the totals of other the vectors. Various codes were built using this method until a full general-geometry, general physics, continuous energy code was constructed around this method. Lessons were taken from the previous codes and combined them all into the production code called RACER3D. Speedups of more than an order of magnitude were reported using this code. Attempts at vectorizing old codes, such as KENO-IV, probably ended up with such poor results because other sections of the code were not specifically designed for vector processing.

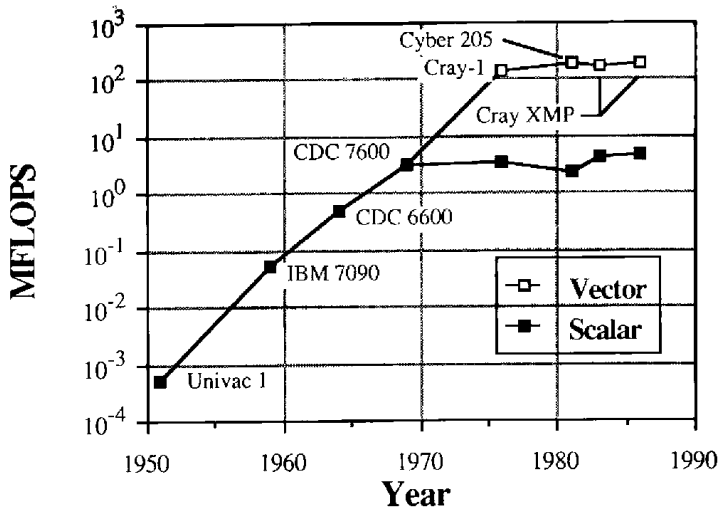

Figure 2.1: Historic computer performance versus time.(2)

The second major computational advancement was parallel processing. Monte Carlo designs have been even slower to capitalize on this technological development than was the case with vector processing. This is curious because it has been recognized since very early on that Monte Carlo was well-suited for parallel computers (3). This is because each particle in a given generation is processed independently of any other particles. Thus, co-generational particles can be run on separate processors at the same time. The memory requirements of Monte Carlo were still a major roadblock. Separate processes would either have to share memory and risk one process having to wait for another process, if they both try to access the same piece of memory at the same time, or to give each process its own complete set of memory, doubling the required amount of accessible memory. Having distributed memory, on the other hand, would require synchronization and communication between the processors after each cycle in order to calculate the results. Accepting this communication issue, this gave rise to domain decomposition in its many forms.

Domain decomposition takes a dimension of the problem and splits it into two or more sections each to be distributed to one processor. This can be done over any dimension: geometry, angle, and/or energy (6). Because of the inevitable exchange

of particles between the sub-domains of any of these decomposition dimensions, this leads to an increase in network communication. Thus, this generally leads to a reduction in efficiency with the notable exception of cases in which there are greater memory requirements for the complete problems than are available to the individual processors, which is alleviated by decomposing the domains. In this case, communicating particles that cross domains between different processors could be faster than having to read the data off of the hard disk. For example, energy decomposition could allow one processor to only concern itself with all high energy particles in fast groups and another could concern itself with thermal energy particles. This would lead to a great many particles scattering from the domain of the first processor to that of the second, but it would only require each processor to have the cross sections for those specific energy ranges. Far more popular is geometric domain decomposition, which divides the problem geometry between processors. This requires each processor to only keep part of the geometry in memory and communicate the necessary information about particles that cross a given boundary to another processor. A concern when attempting to combine this with vector processing is that by dividing up the number of particles simulated on a given processor, it will be more difficult to fill a vector for efficient processing.

While domain decomposition solves the problem of computers with limited memory, it does so with the aforementioned trade-offs. However, computers have been advancing such that containing a fairly large and complex problem geometry complete with continuous energy cross-sections for many isotopes on a single computer is becoming more common. With this, the main problem to be tackled by parallel processing is to process more particles more quickly. This has been achieved with domain replication. Domain replication takes the entire problem domain, including all cross sections, all angles, and the entire problem geometry, and puts it on every processor. With this method, the only communications are giving each processor the problem domain and the number of particles to be run at the beginning and each processor reporting to the master processor the results of each cycle at its end.

Thus, so long as the computation time involved in simulating the required number of particles on a given processor takes a significant amount of time with respect to the communication time between cycles, a speedup can be observed. Domain replication and domain decomposition can also be utilized at the same time, taking multiple instances of a decomposed domain and distributing them to various processors. To observe the benefits of both domain replication and domain decomposition, Lawrence Livermore National Laboratory developed the MERCURY Monte Carlo code (8). They were able to observe speedups for both methods.

Finally, another technique used to improve the efficiency of the Monte Carlo approach is to use it as part of a hybrid method (13). This hybrid method utilizes a deterministic code to generate importance functions over the problem in any dimension, generally space and energy. This importance function can then be used to generate a problem-wide map for weight windows to be used in the Monte Carlo simulation. Weight windows is a variance reduction technique used in Monte Carlo codes that establishes upper, lower, and target weights for a particle of a given energy in a given region of the problem. If a particle has too high of a weight, it is split into multiple particles of equal weight as close to the target weight as possible. If a particle has too low of a weight, it goes through roulette with an appropriate probability of survival such that the particle is either killed or survives with the target weight. By applying such a weight window map to a problem at the very beginning, this method ensures that areas that are generally slow to converge, and would otherwise require a great deal more histories to be run for confident tally estimates are able to converge at similar rates to other, more quickly converging areas of the problem.

2.3 Shift: A Massively Parallel, Hybrid Monte Carlo/Deterministic Radiation Transport Code

In order to take full advantage of all of these methods, Oak Ridge National Laboratory undertook the development of the Shift hybrid deterministic/Monte Carlo code as a part of the Denovo code-base. The entire code-base is being built to be utilized on massively parallel machines. As such, Shift is capable of serial runs, domain replication, domain decomposition, or both domain replication and decomposition. Once it is complete, the results from the deterministic calculation will be used to accelerate source convergence and drive hybrid variance reduction techniques used in the Monte Carlo simulation. By combining all of these, it should become feasible to run highly detailed, full reactor problems on massively parallel computers in reasonable amounts of time and to get well-converged tallies in all areas of the core.

As a starting point in building this code, the Monte Carlo module of the code is being built to be capable of stand-alone simulations. This is to verify that it is performing appropriately as it is being built (12). Such verification is necessary throughout the project to ensure that each newly added component is working and that its implementation does not break any of the other components already in place. One of the ways of accomplishing this is by building an input to test a given feature that has been added to the code under development and already exists in a reference code that has been previously verified and validated. Using different inputs with varying expected results, it can be shown that the method is responding appropriately to the different input parameters by showing similar changes between the new code and the old one. For instance, the transport section of a radiation transport code can be tested by utilizing the same problem at different temperatures and seeing if the result responds in similar relative direction and magnitude as the reference code. After new components are added, the same problems should be run again to verify that the old components work, and new problems should be run to verify the new component has been implemented correctly.

Chapter 3

Methodology

For this research, the results of the Monte Carlo module of Shift were compared against those of MCNP5-1.60, hereafter referred to as MCNP, and KENO-VI. The compared results include eigenvalue results as well as flux tallies for certain benchmarks. The benchmarks used include the C5G7-2D and -3D benchmarks as well as select benchmarks developed by the Consortium for Advanced Simulation of Light Water Reactors (CASL) at Oak Ridge National Laboratory. To test the parallel capabilities of Shift, the problems were all run on varying numbers of processors. To observe the efficiency of Shift, the problems were also run with varying numbers of histories.

3.1 Codes Used

MCNP is a general purpose Monte Carlo radiation transport code developed at Los Alamos National Laboratory. Capable of both criticality and shielding calculations, MCNP handles the transport of neutrons, photons, and electrons. While MCNP can utilize continuous energy cross sections, the benchmark for which this code was used in this research did not require it. However, part of the powerful tallying capabilities were utilized. MCNP is also capable of domain replication parallel processing, though

this was only used to achieve better results faster and not for timing comparisons in this research (11).

KENO-VI is the Monte Carlo criticality code that comes with the SCALE package, developed at Oak Ridge National Laboratory. With the update from KENO-Va to KENO-VI, the geometry package gained improved functionality while also making the generation and reading of the input file easier for the user. The update also added continuous energy functionality, though the code is equally capable in both continuous energy and multi-group modes. Recently, domain replication parallel capabilities have been added (14).

As previously noted, Shift is being constructed at Oak Ridge National Laboratory to be a massively-parallel, hybrid-method radiation transport code. This research concentrates on criticality calculations using the Monte Carlo method only. At this point in development, Shift is capable of utilizing multi-group and continuous energy cross sections for both criticality and shielding calculations. It can run in parallel using full domain decomposition, full domain replication, domain decomposition with overlapping domains, and multi-set overlapping domain (MSOD) decomposition (15). Shift also accepts several geometry input formats including those used in KENO-VI and MCNP.

At the time of this research, two variance reduction techniques were employed: weight roulette and implicit capture. Weight roulette involves tracking a particle's weight. Once the weight falls below a prescribed value, the particle is subject to Russian roulette. This results in either the particle being killed or the particle's weight increasing above the lower limit based on the probability of elimination (11). While this does not directly reduce the variance of a result, it allows for computation time to be spent on particles of greater significance to the answer, generally resulting in a reduction in the variance reported after a given period of computation time. Implicit capture occurs along every step of particle transport. Every time a particle experiences a scatter, its weight is reduced based on the probability of absorption and a partial absorption is tallied at that location (11). This results in absorption tallies

all along the particle path instead of just at the end. Also, this reduces the weight of the particle smoothly as it is transported, taking it more quickly towards the lower limit to be subject to rouletting.

3.2 Benchmark Problems

The C5G7 benchmark problems were developed by the Organization for Economic Co-operation and Development Nuclear Energy Agency in order to test the capabilities of radiation transport codes that do not use spatial homogenization above the fuel-pin level (7; 9). The benchmark problem is the C5 MOX fuel assembly problem specified by Cavarec et al. (5). It has quarter-core radial and half-core axial symmetry. The full core consists of 16 light-water-moderated 17x17 fuel assemblies containing either uranium oxide (UO2) or mixed oxide (MOX) fuel. Taking advantage of the symmetry, one octant is simulated with reflective boundary conditions on the three inward-facing surfaces. The fuel assemblies are surrounded by light-water acting as a reflector beyond which is a vacuum boundary condition (Figure 3.1). While the UO2 assembly only has one enrichment, the MOX assemblies have three enrichments of 4.3%, 7.0%, and 8.7% as shown in Figure 3.2. Each assembly contains 24 guide tubes for control rods and one instrument tube for a fission chamber in the center grid-cell. All pin cells have a radius of 0.54cm with a pitch of 1.26cm.

The two-dimensional case (2D) is a 1cm thick slice of the reactor with reflective boundary conditions on the top and the bottom, as well as the two inward facing radial surfaces. The fuel regions in the three three-dimensional (3D) cases are divided into three equal axial sections. The differentiation between the three inputs is by the axial sections that have control rods or water in the guide tubes. The "Rods Out" case only has control rods above the fuel assemblies in the reflector region. The "Rodded A" case has control rods only in the top section of the inner fuel assembly. The "Rodded B" case has control rods in the top section of all but the outer/corner assembly and the middle section of the inner assembly (Appendix A). As these cases are reflected

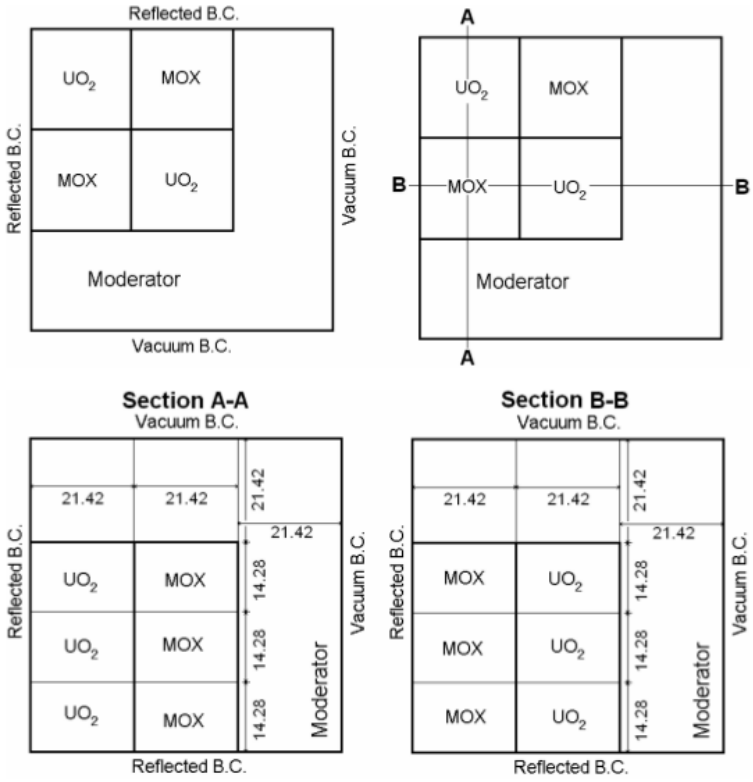

Figure 3.1: C5G7 boundary conditions (7).

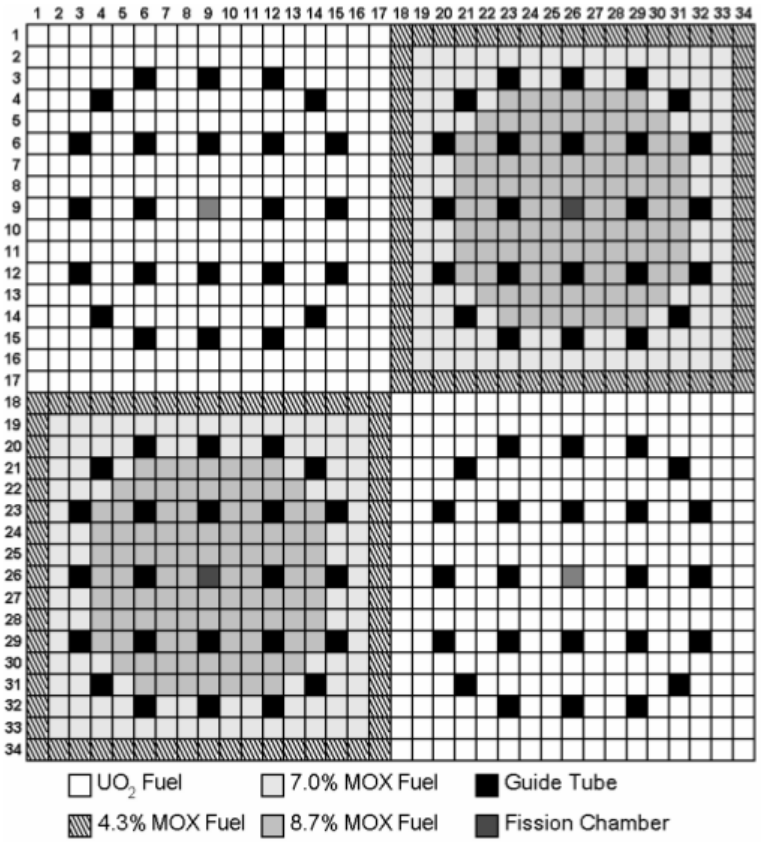

Figure 3.2: C5G7 pin pattern (7).

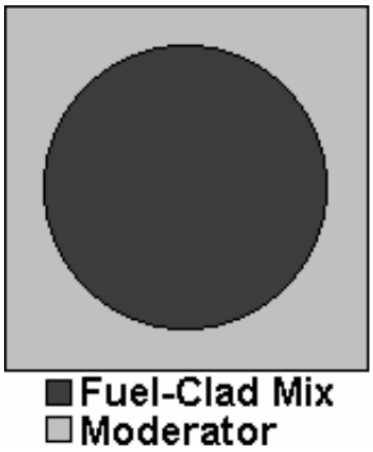

Figure 3.3: C5G7 fuel pin geometry (7).

across the axial mid-plane of the reactor, they simulate the control rods being inserted from the top and the bottom.

The materials in this benchmark include light-water used as the reflector and moderator, one UO2 fuel, three enrichments of MOX fuel, one guide tube, one instrument tube, and one type of control rod. While this benchmark test is intended for codes that avoid homogenization above the fuel-pin level, all of the pins are homogenized with the gap and cladding (Figure 3.3). The benchmark provides seven group macroscopic cross sections in an MCNP-format that are normalized such that the atom density of all of the materials is 1 atom/barn-cm. The benchmark provides eigenvalue results for all of the above configurations based on MCNP simulations. Although the benchmark also provides pin power results, this research utilized flux-tallies instead, due to the early stage of development Shift was in. All of these problems were run with 1200 cycles including 500 skipped cycles. For the eigenvalue results, 300,000 particles were simulated for every cycle.

At ORNL, as part of CASL, a series of reactor progression benchmark problems are being designed by Andrew Godfrey (17). These benchmark problems are based on the Watts Bar Nuclear 1 (WBN1) reactor on its first cycle. WBN1 is a Westinghouse designed 17x17 pressurized water reactor (PWR) built in the U.S. in the 1980s. This series of benchmarks is being designed to provide several problems of increasing complexity to test and demonstrate the capabilities of a reactor simulation code. Again, due to the early stage of development that Shift was in at the time of this research, some modifications were required for the utilization of these benchmark problems. While these benchmarks were designed to use continuous energy (CE) cross sections, at the time of this research Shift did not have CE capabilities. Instead, 238-group cross sections were used. Although the benchmark provides reference Monte Carlo results based on CE cross sections, multi-group cross sections were generated and used for comparisons in this research. Also, as Shift only utilized isotropic scattering, the KENO-VI runs were performed using only isotropic scattering. Problems 1 and 2 were run for 2300 cycles including 500 skipped cycles. For the eigenvalue results of Problem 1, 200,000 particles were simulated for every cycle in both KENO-VI and Shift. For Problem 2, KENO-VI used 200,000 particles while Shift used 100,000.

The first benchmark problem is an infinitely reflected 2D pin-cell based on the beginning of cycle. It has a UO2 fuel pin including 3.1% enriched fuel, a gap filled with Helium gas, zircalloy-4 cladding, and borated water as the moderator at 1300ppm. The radius of the fuel pellet and the outer radii of the gap and cladding are 0.4096cm, 0.418cm, and 0.475cm, respectively. The lattice pitch is 1.26cm. Due to the fact that one of the capabilities this benchmark tries to demonstrate is variation in the eigenvalue result with respect to temperature, several temperature combinations of the fuel and moderator are evaluated (Table 3.1). For all of the cases, the moderator temperatures, density, and boron concentration are based on the hot-zero-power (HZP) condition of WBN1.

Table 3.1: CASL benchmark problem 1 material temperatures.

Case	Fuel temperature [Kelvin]	Moderator temperature [Kelvin]
1a	565	565
1b	600	600
1c	600	900
1d	600	1200

The second benchmark problem from this suite expands the pin-cell of the first problem out to a 2D assembly lattice. To appropriately simulate an assembly, this benchmark includes 24 guide tubes and an instrument tube in the center-grid cell. Also included is an inter-assembly gap of 0.04cm. The lattice has reflective boundary conditions on all sides. Figure 3.4 shows a quarter representation of this lattice. Unlike the first problem, only one temperature was used to evaluate the eigenvalue results and scaling capabilities of Shift. Additional benchmark problems have been developed scaling up to several assemblies; however, these were not available at the time of this research.

3.3 Verification Methodology

These problems were all run on various computers at ORNL and the NECluster at the University of Tennessee's Department of Nuclear Engineering. The only computer on which timing runs were executed was the ORNL Institutional Cluster (OIC). This computer is composed of 40 nodes with 12 processors (dual hexacores) per node and distributes jobs using the TORQUE queuing system. Scaling runs were performed using the 3D C5G7 benchmarks and the CASL problem 1 and 2 benchmarks.

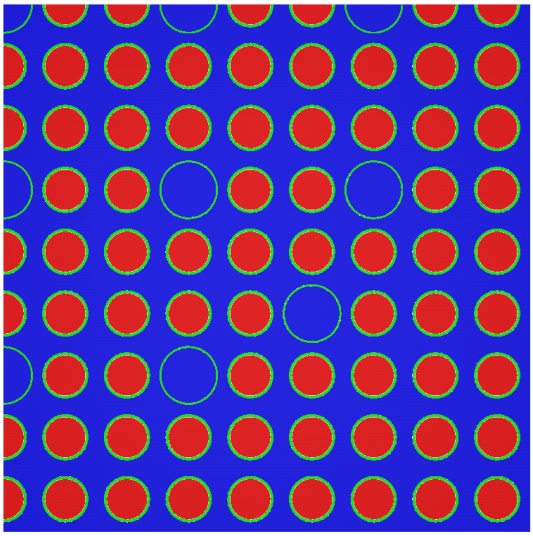

Figure 3.4: Quarter-lattice used in Problem 2 (17).

All of the inputs were generated by the author. The inputs for Shift used the python input format, an example of which can be seen in Appendix B. Cross sections were input manually for the 7-group C5G7 benchmarks in Shift. The benchmark came with its own MCNP-style cross section file to use with MCNP. For the C5G7 benchmarks, the geometry is also defined within the python input. For the 238-group CASL benchmarks, cross sections were generated using the CSASI sequence in the SCALE code package. CSASI uses CENTRM, PMC, and WORKER to generate problem-dependent, microscopic multi-group cross sections based on the geometry(14). It then uses ICE to mix the cross sections and generate an AMPX library of macroscopic cross sections. These can then be used by both KENO and Shift. The CASL benchmark geometries are defined in Shift using the KENO format to assure consistency. To avoid complications and assure that the codes are run in as similar a manner as possible, both KENO and Shift were run using isotropic scattering only.

The C5G7 MCNP and Shift runs were compared, using both the eigenvalue and flux tally results along with their respective standard deviations, to each other and to the reference results. In order to properly compare the results, propagation of error was used to determine the relative uncertainty between the results of the benchmark paper, the MCNP results, and the Shift results (Equation 3.1). Equation 3.1 shows that the propagated error is equal to the square root of the sum of the squares of the two errors being propagated - in this case from the benchmark paper, MCNP, or Shift results. As the flux tallies were performed on every pin cell on the 2D benchmark and for all three axial zones for every pin on the three 3D benchmark problems, the errors for the respective tallies in MCNP and Shift were also propagated. The flux tallies were compared using relative percent difference between the results of each tally (Equation 3.2). The CASL problems were only compared using the eigenvalue results from the benchmark, which used KENO-VI and Shift.

$$\sigma_{Propagated} = \sqrt{\sigma_1^2 + \sigma_2^2} \tag{3.1}$$

$$Rel.\%Diff. = 100 \times \left| \frac{\phi_1 - \phi_2}{\phi_1} \right| \tag{3.2}$$

To investigate the scaling capabilities at this point in development, the problems were run on OIC with varying numbers of processors and particles. The number of processors on which the benchmarks were run varied based on using complete nodes. Thus, each problem was run on 12, 24, 48, 72, 96, 120, 144, 168, 192, 216, and 240 processors for each number of particles. The number of particles was varied based on the number of particles per cycle starting at 1000 and increasing by steps of roughly double the previous number. The maximum number of particles per cycle varied based on the input. The second CASL problem was only scaled up to 1×10^5 particles per cycle, whereas the first CASL problems were scaled up to 1×10^6 particles per cycle, and the C5G7 problems up to 1×10^7 particles per cycle. For example, the second CASL problem was run with 1000, 2000, 5000, 10000, 20000, 50000, and

100000 particles per cycle. These runs allowed for observation of the speedup and efficiency of Shift for each problem. Speedup is generally defined as the time it takes for a problem to run on a single processor divided by the time it takes for the same problem to run on more than one processor. In this research, speedup is the time it takes for a problem to run on a single 12-processor node divided by the time it takes for the same problem to run on more than one 12-processor node. The efficiency is the ratio of how much speedup was achieved to how much additional power was used to solve the problem. For example, if a problem is found to run twice as fast when running on four times as many processors, the efficiency would be $\frac{2}{4}$ or 0.5. These results also provided an opportunity to observe the behavior of the variance of the eigenvalue as a function of particles simulated. The expectation was that the standard deviation should decrease as the inverse square root of the number of histories.

Chapter 4

Results

4.1 Eigenvalue Results

4.1.1 C5G7 Benchmark

Table 4.1 shows the eigenvalue results of all four C5G7 benchmark problems along with their respective standard deviations in percent mil or pcm (Equation 4.1). The table also shows the propagated standard deviations between the benchmark results and those from MCNP and Shift. The difference between the eigenvalue results between the benchmark and the two codes is shown in both pcm and as a fraction of the propagated standard deviations (Equation 3.1). All of the results show good agreement by falling within three standard deviations of each other, with MCNP and Shift each having one result that is over two standard deviations away from the respective benchmark value. This is followed by a direct comparison of the results of the two codes. The propagated standard deviations are presented again, in pcm, followed by the differences in the eigenvalue results in terms of pcm and fractions of the propagated standard deviations. The table shows that none of the eigenvalue results were more than two standard deviations apart.

$$pcm = 10^5 \times (k_1 - k_2) \tag{4.1}$$

Table 4.1: C5G7 Eigenvalue Results

Case	Source	Eigenvalue	Std. Dev. [pcm]	Error Propagated Std. Dev. [pcm]	vs. benchmark [pcm]	vs. benchmark [fraction of Std. Dev.]	Error Propagated Shift/MCNP [pcm]	vs. MCNP [pcm]	vs. MCNP [fraction of Std. Dev.]
2D	Benchmark	1.18655	3	-	-	-	-	-	-
	MCNP	1.18651	5	5.83	-4	-0.686	-	-	-
	Shift	1.18652	7	7.62	-3	-0.394	8.60	1	0.116
Rods Out	Benchmark	1.14308	3	-	-	-	-	-	-
	MCNP	1.14298	5	5.83	-10	-1.71	-	-	-
	Shift	1.14310	7	7.62	2	0.262	8.60	12	1.40
Rodded A	Benchmark	1.12806	3	-	-	-	-	-	-
	MCNP	1.12822	5	5.83	16	2.74	-	-	-
	Shift	1.12807	7	7.62	1	0.131	8.60	-15	-1.74
Rodded B	Benchmark	1.07777	3	-	-	-	-	-	-
	MCNP	1.07773	5	5.83	-4	-0.686	-	-	-
	Shift	1.07759	7	7.62	-18	-2.36	8.60	-14	-1.63

Table 4.2: C5G7 Flux Tally Results

Case	Range of relative % differences in flux tally	Tallies within 3σ
2D	-0.456 - 0.492	98.01%
Rods Out	-1.23 - 1.09	98.41%
Rodded A	-1.35 - 0.846	97.95%
Rodded B	-1.53 - 0.965	96.80%

The flux tally results were compared using relative percent differences as well as the propagated standard deviations. Table 4.2 shows the ranges of the relative percent differences for all four benchmark problems. Some negative bias with respect to the MCNP flux tally results can be observed in the ranges of the 3D benchmark problem results; however, the greatest relative difference is just over 1.5%, so the results have a reasonable level of agreement.

Table 4.2 also shows the percentage of flux tally results that fall within three propagated standard deviations of each other for each benchmark problem. A perfect Gaussian distribution would have 99.7% of the tallies fall within three standard deviations of each other, and these results have nearly met this criterion. It is reasonable to suspect that the observed variations could be attributed to cycle-to-cycle correlations or fission source distributions that have not completely converged. It is important to remember that the alternating fuel materials between assemblies will lead to an uneven fission source. Figure 4.1 shows a distribution of the relative percent differences between the flux tallies of MCNP and Shift for the Rodded B benchmark problem, which contains the greatest flux tally difference. In this figure, the center of the core is in the back corner at coordinate [35, 0, 0]. The figure shows that the tallies all fall within 2% of each other. The greatest difference is at the edges of the fuel nearest the reflector. These areas are also the most uncertain as they have

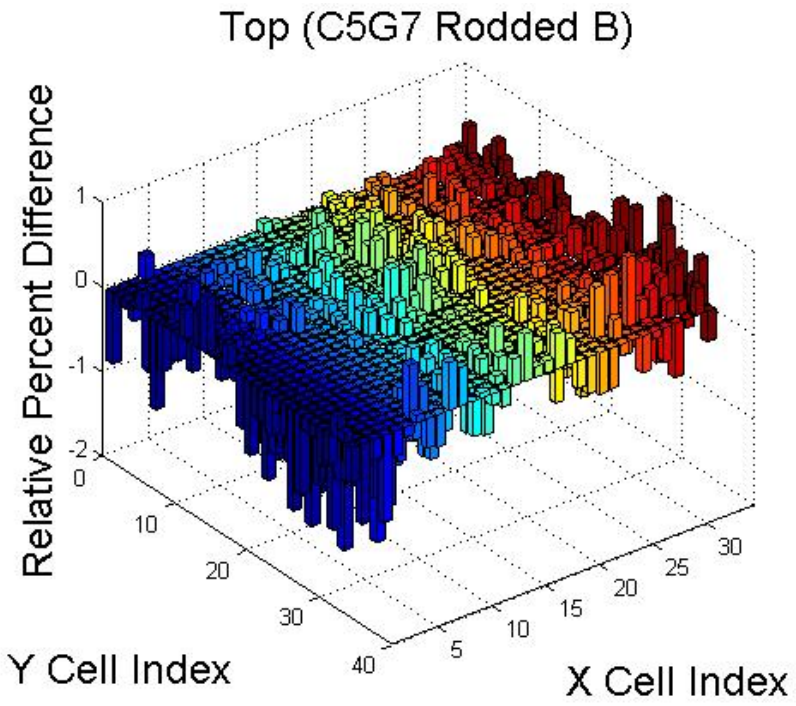

Figure 4.1: Relative percent difference of flux tallies in top section of Rodded B case between Shift and MCNP.

lower fluxes. The seemingly random nature of the tallies supports the conclusion that there are no significant systematic differences in the results of the two codes.

4.1.2 CASL Problems 1 and 2

Table 4.3 shows the eigenvalue results for both CASL benchmark Problems 1 and 2. The eigenvalue results for each case are given for both codes along with the resulting standard deviation, in terms of pcm. These were then combined using propagation of error. The differences in the eigenvalue results from the two codes are then presented in both pcm and as a fraction of the error propagated standard deviation. All of the results fall within two standard deviations, and most of the Problem 1 results fall

Table 4.3: CASL Eigenvalue Results

Case	Code	Eigenvalue	Std. Dev. [pcm]	Error Propagated Std. Dev. [pcm]	vs. KENO-VI [pcm]	vs. KENO-VI [fraction of Std. Dev.]
1a	KENO-VI	1.18234	2.8	5.22	-5.4	-1.03
	Shift	1.18229	4.4			
1b	KENO-VI	1.17763	2.8	5.13	-0.9	-0.175
	Shift	1.17762	4.3			
1c	KENO-VI	1.16644	2.9	5.27	-3.8	-0.721
	Shift	1.16640	4.4			
1d	KENO-VI	1.15700	3.1	5.22	-3.1	-0.594
	Shift	1.15697	4.2			
2a	KENO-VI	1.17858	2.9	6.75	16.1	2.39
	Shift	1.17874	6.1			

within one standard deviation. All of the Problem 1 results for Shift do fall below those of KENO-VI, however the result for Problem 2 from Shift is greater than that from KENO-VI. As with the C5G7 results, the standard deviation estimate reported by KENO-VI is smaller than that reported by Shift. This is most likely due to more variance reduction techniques being used by KENO-VI. Naturally, this is not the only reason for the results for Problem 2 as it should be noted that Shift was run with half as many histories as KENO-VI. But even after estimating the amount that the standard deviation would be reduced by doubling the number of particles, it still would not be as low as the estimate reported by KENO-VI.

4.2 Shift Scaling Results

Performing the scaling study allowed for the observation of two characteristics of the code. First, it allowed for the verification of the variance behavior as the number of histories varied across several orders of magnitude, the expected behavior being that it would decrease linearly with an increase in number of histories. Second, it allowed for an evaluation of the parallel speedup and efficiency of the code under various circumstances.

The variance behavior was evaluated by plotting all of the variances for a given benchmark on a log-log plot of variance vs. the number of particles per cycle. Generation of a best fit line as an exponential should result in the exponent being -1. This was performed using Microsoft Excel 2010. The resulting graph for the C5G7 Rodded B case can be seen in Figure 4.2. The trend line and its equation are included on the chart, and show that the exponential is -1.005 based on data from 10^3 to 10^7 particles per cycle. Figure 4.3 shows the same result for Problem 1a of the CASL benchmark. As the figure shows, this particular instance resulted in the expected exponent of -1. The exponentials for all of the benchmarks range from -0.984 to -1.009. The -0.984 value is for CASL Problem 2 with the next lowest being -0.998. These all show that the variance behavior in Shift with varying numbers of histories is appropriate for a Monte Carlo code.

The timing evaluation of Shift using the C5G7 Rodded B case is demonstrated in Figure 4.4, which shows that there is a reduction in run time as the problems is run on more processors for larger numbers of particles per cycle. As expected, once the number of histories per processor becomes small enough, it takes longer to complete the simulation as it is run on more processors. For clarity, the equation for the trend line for each data set, based on number of particles per cycle, is provided next to the legend. As the exponential decreases, the efficiency of adding more processors is decreasing. This culminates once the exponential becomes positive, showing that it takes longer to complete the simulation as more processors are added. The lines

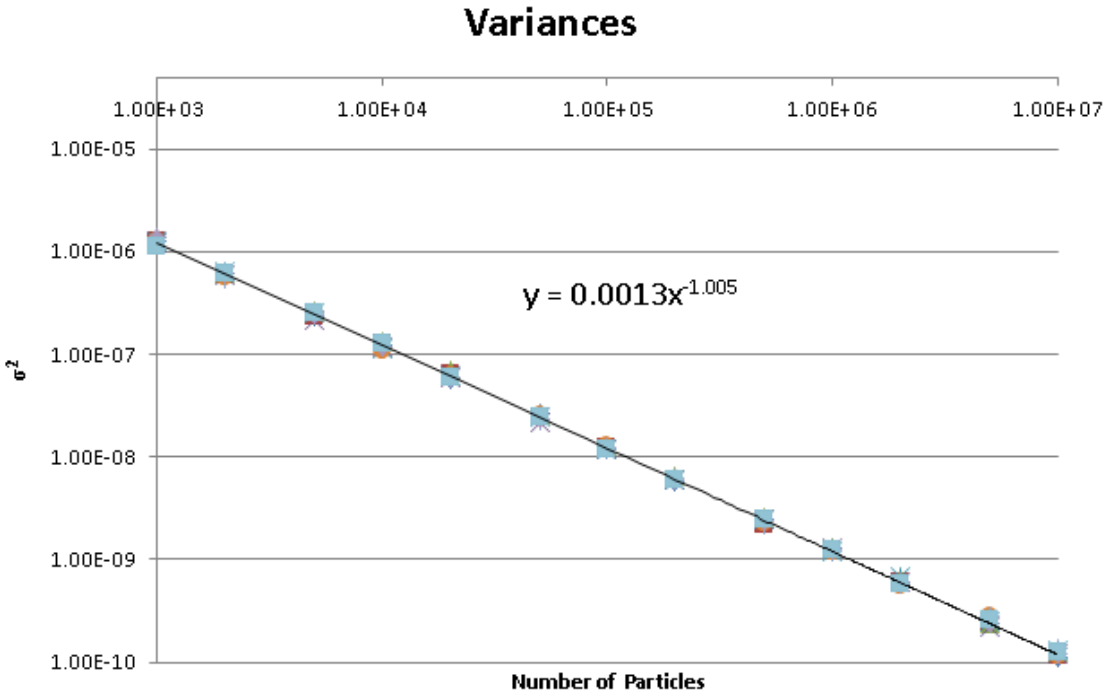

Figure 4.2: C5G7 Rodded B variances vs. number of particles per cycle.

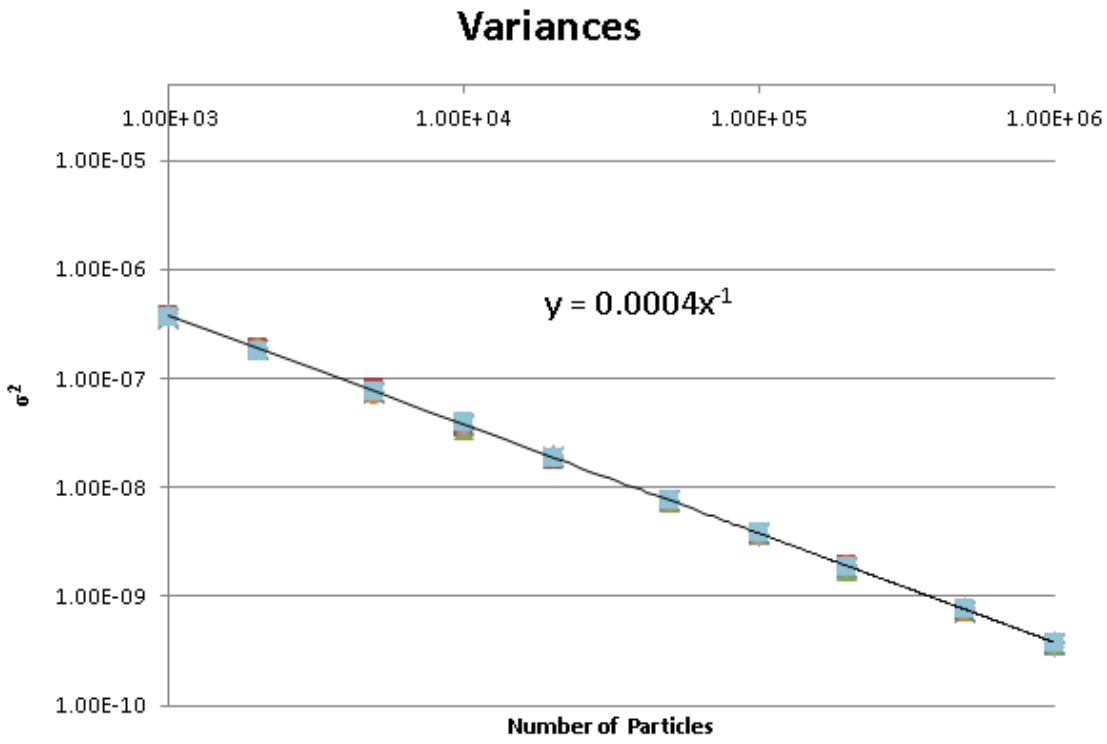

Figure 4.3: CASL Problem 1a variances vs. number of particles per cycle.

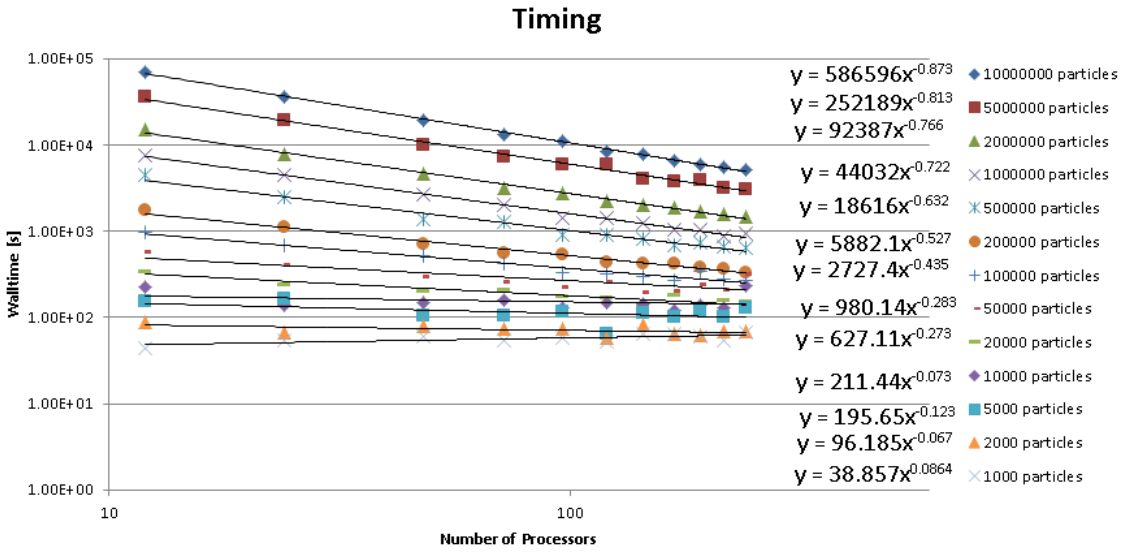

Figure 4.4: C5G7 Rodded B run time vs. number of processors.

representing the runs with the most histories show that running the simulation on 240 processors has resulted in completion in less than a tenth of the time it takes to run on 12 processors. However, the equation for the line representing the fewest histories has a positive exponetial, indicating that the simulation actually took longer when run on more processors than it did on fewer. This is the point at which the extra time required for communication between all of the processors is longer than the reduction in time due to each processor having to process fewer particles. The results from almost all of the benchmarks show that the inflection point is somewhere around 1000 particles per cycle.

Figure 4.5 shows the speedup for the C5G7 Rodded B case. As this research was done using complete nodes, this chart shows how much faster a simulation is completed vs. how many nodes were used to complete the simulation. The chart includes a line demonstrating perfect speedup. With 10 times the nodes, perfect

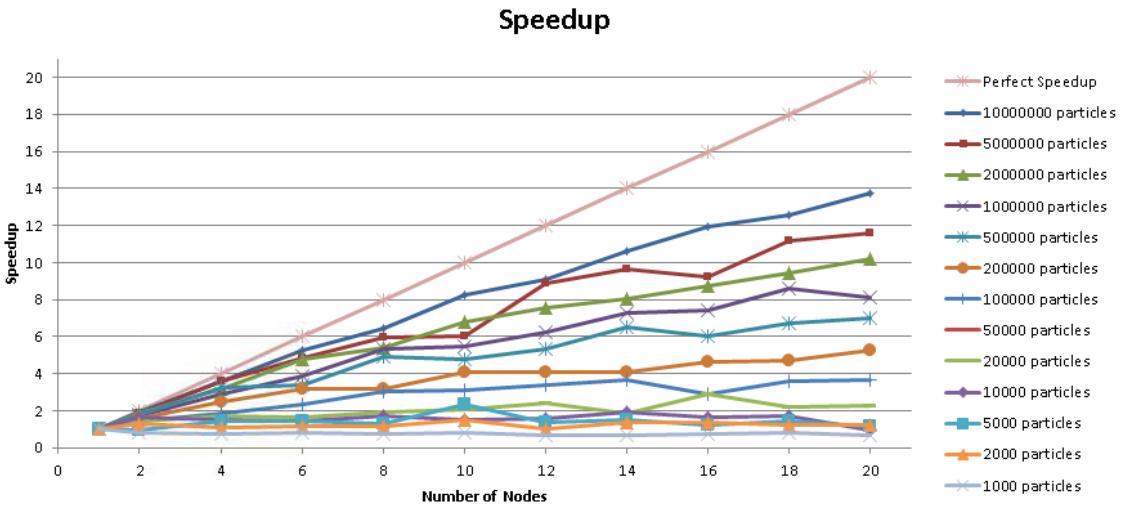

Figure 4.5: C5G7 Rodded B speedup vs. number of nodes.

speedup would result in the simulation finishing 10 times faster. Of course, this is not feasible because there are serial parts of the code and communication costs. Every line segment that has a positive slope shows that adding those processors caused the simulation to take less time to run overall. If the line segment is flat, i.e., has a slope of zero, it represents no increase or decrease in runtime associated with running the case on those extra processors. Naturally, if the line segment has a negative slope, running the case on those extra processors resulted in the simulation taking longer than it did without them. Some variation is to be expected as adding processors requires different sets of random numbers. This can lead to certain processors getting hung up on more particles requiring the entire process to wait for them to finish. Again, it is clear that the speedup is not as pronounced as the number of particles per cycle decreases, as the parallel sections of the code (particle transport) are completed significantly faster than the serial and communication sections.

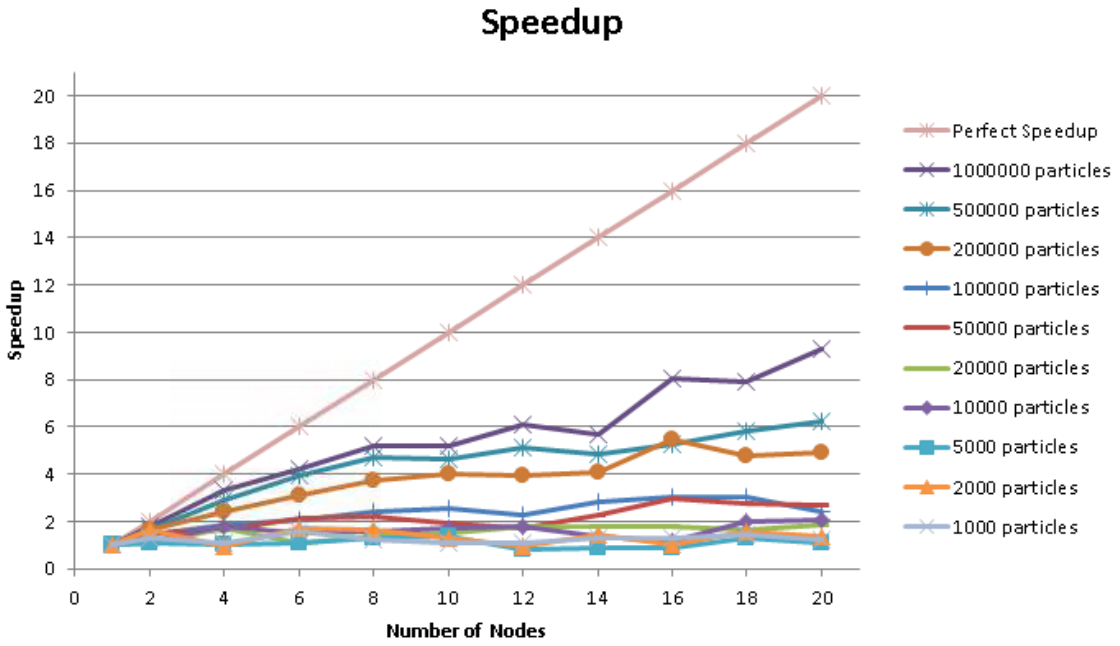

Figure 4.6: CASL Problem 1d speedup vs. number of nodes.

Figure 4.6 shows a similar speedup. This demonstrates that the speedup is not significantly impacted by the number of cross section groups used in the simulation.

Finally, Figure 4.7 shows the efficiency for the Rodded B case. This chart demonstrates the relationship between the lines for each case and the perfect scenario (a linear speedup with the addition of more processors) illustrated in Figure 4.5. In this chart, a perfect speedup would be represented by an efficiency of 1, or a flat line across the top. Again, as the particle transport section of the code is that which is parallelized, having more histories allows for a more efficient use of multiple processors. Graphs of the results for all of the benchmark problems can be found in Appendix C.

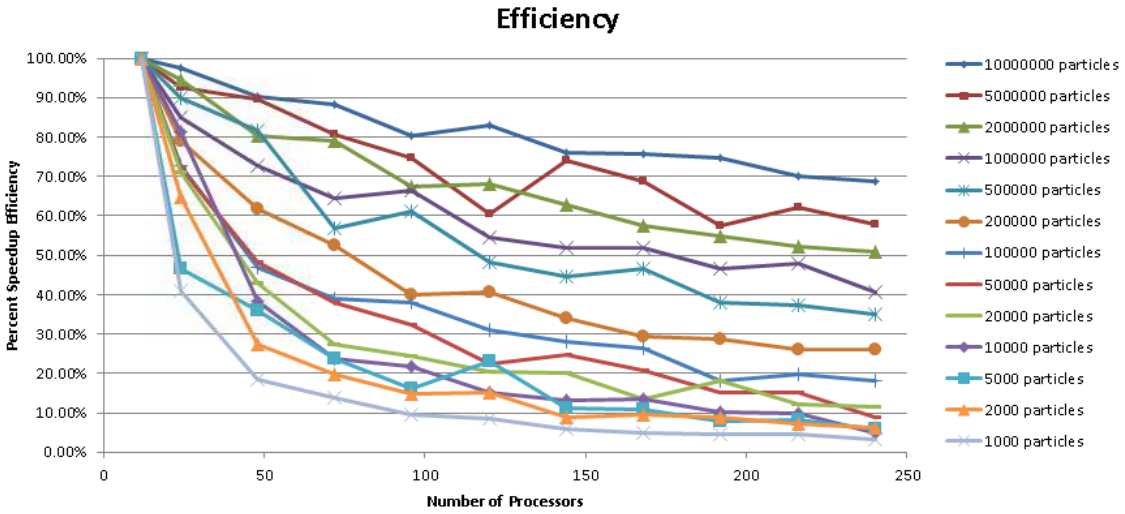

Figure 4.7: C5G7 Rodded B efficiency vs. number of processors.

Chapter 5

Conclusions

The results show that Shift is working properly at this stage of development. For all nine cases that were investigated, Shift has been shown to produce eigenvalue results in agreement, within three standard deviations, with results from published benchmarks, MCNP5, and/or KENO-VI. Shift was also shown to have good agreement with MCNP5 flux tallies for the C5G7 cases. While fewer than expected tallies fell within three standard deviations of each other, this may be due to cycle-to-cycle correlation or incomplete convergence of the fission source (16). The scaling study went on to demonstrate that the variance estimate behaves as expected for a Monte Carlo simulation, by decreasing as an exponential of negative one versus increased histories.

The scaling study then demonstrated the parallel capabilities of Shift. As expected, with many histories, increasing the number of processors on which the simulation is run results in significant run time reduction. However, with fewer histories to be divided up between the many parallel processors, further parallelization results in longer run times. This was further reflected in the speedup and efficiency results. The results show that Shift is capable of significant increases in speed of simulation through the use of domain replicated parallelization. For systems that are used by many people, a minimum desired degree of efficiency should be set to ensure

that the system is being utilized to as close to its full potential as possible. As shown in Figure 4.7, there comes a point at which using more processors results in all of the processors being used inefficiently. Whilt it is infeasible to expect experimenters to demonstrate that their use of a code was above the recommended efficiency, such scaling studies can be used as a basis for such assurance.

There is much work to do yet on the verification of the Shift code. As it is under development, these simulations will need to be repeated to ensure no future modifications or added capabilities have caused the initial code segments to behave differently. Also, the CASL benchmark problems will have to be compared once continuous energy physics has been fully implemented. Reaction rate tally comparisons will also need to be made. The rest of the suite of CASL benchmark problems will also have to be used to demonstrate Shift's ability to properly simulate more complicated environments. The larger of the CASL benchmarks and other such benchmarks will allow for the testing of the domain decomposition method of parallelization. Finally, once the code has been completed to the satisfaction of ORNL, it will need to be validated against measured experimental results to show that it can adequately predict real world processes.

Bibliography

[1] K. S. Smith (1986). Assembly Homogenization Techniques for Light Water Ractor Analysis. *Progress in Nuclear Energy*, 17(3):303–356. 4

[2] William R. Martin and Forrest B. Brown (1987). Status of Vectorized Monte Carlo for Particle Transport Analysis. *International of High Performance Computing Applications*, 1(2):11–32. x, 3, 5, 6

[3] F. Schmidt and W. Dax and M. Luger (1990). Experiences with the Parallelisation of Monte Carlo Problems. *Progress in Nuclear Energy*, 24:141–151. 1, 3, 6

[4] K. Asai and K. Higuchi and M. Akimoto and Y. Naito and Y. Komuro and H. Kadotani (1990). Monte Carlo Calculations on High Speed Machines. *Progress in Nuclear Energy*, 24:175–182. 5

[5] C. Cavarec and J. F. Perron and D. Verwaerde and J. P. West (1994). Benchmark Calculations of Power Distribution Within Assemblies. NEA/NSC/DOC(94)28. 12

[6] Yousry Y. Azmy (1997). Multiprocessing for Neutron Diffusion And Deterministic Transport Methods. *Progress in Nuclear Energy*, 31(3):317–368. 1, 6

[7] (2003). Benchmark on Deterministic Transport Calculations Without Spatial Homogenisation. OECD Report ISBN 92-64-02139-6, A 2-D/3-D MOX Fuel Assembly Benchmark. x, 12, 13, 14, 15

[8] Richard Procassini and Janine Taylor and Ivan Corey and John Rogers (2003). Design, Implementation, and Testing of Mercury: A Parallel Monte Carlo Transport Code. In *Nuclear Mathematical and Computational Sciences: A Century in Review, A Century Anew*. 8

[9] (2005). Benchmark on Deterministic Transport Calculations Without Spatial Homogenisation. OECD Report ISBN 92-64-01069-6, MOX Fuel Assembly 3-D Extension Case, NEA No. 5420. 12

[10] Forrest B. Brown (2005). Fundamentals of Monte Carlo Particle Transport. (LA-UR-05-4983). 3

[11] X-5 Monte Carlo Team (2005). *MCNP - A General N-Particle Transport Code, Version 5 - 1: Overview and Theory*. Los Alamos National Laboratory. 11

[12] Forrest B. Brown and Brian Kiedrowski and Jeffrey Bull and Matthew Gonzales and Nathan Gibson (2010). Verification of MCNP5-1.60. (LA-UR-10-05611). 9

[13] John C. Wagner and Scott W. Mosher and Thomas M. Evans and Douglas E. Peplow and John A. Turner (2011). Hybrid and Parallel Domain-Decomposition Methods Development to Enable Monte Carlo for Reactor Analyses. *Progress in Nuclear Science and Technology*, 2:815–820. 8

[14] (2011). *SCALE: A Modular Code System for Performing Standardized Computer Analysis for Licensing Evaluation*. Oak Ridge National Laboratory. Version 6.1. 11, 18

[15] Nicholas C. Sly and Brenden T. Mervin and Scott W. Mosher and Thomas M. Evans and John C. Wagner and G. Ivan Maldonado (2012). Verification of the Shift Monte Carlo Code. In *PHYSOR 2012 - Advances in Reactor Physics - Linking Research, Industry, and Education*. American Nuclear Society. 1, 11

[16] Brenden T. Mervin and Scott W. Mosher and John C. Wagner and G. I. Maldonado (2013a). Uncertainty Underprediction in Monte Carlo Eigenvalue Calculations. *Nuclear Science and Engineering*, 173(3):276–292. 33

[17] A. Godfrey (2013b). VERA Core Physics Benchmark Progression Problem Specifications. Technical report, Consortium for Advanced Simulation of LWRs. Revision 2 of CASL Technical Report: CASL-U-2012-0131-002. x, 16, 18

Appendix

Appendix A

C5G7 Control Rod Insertion Depth

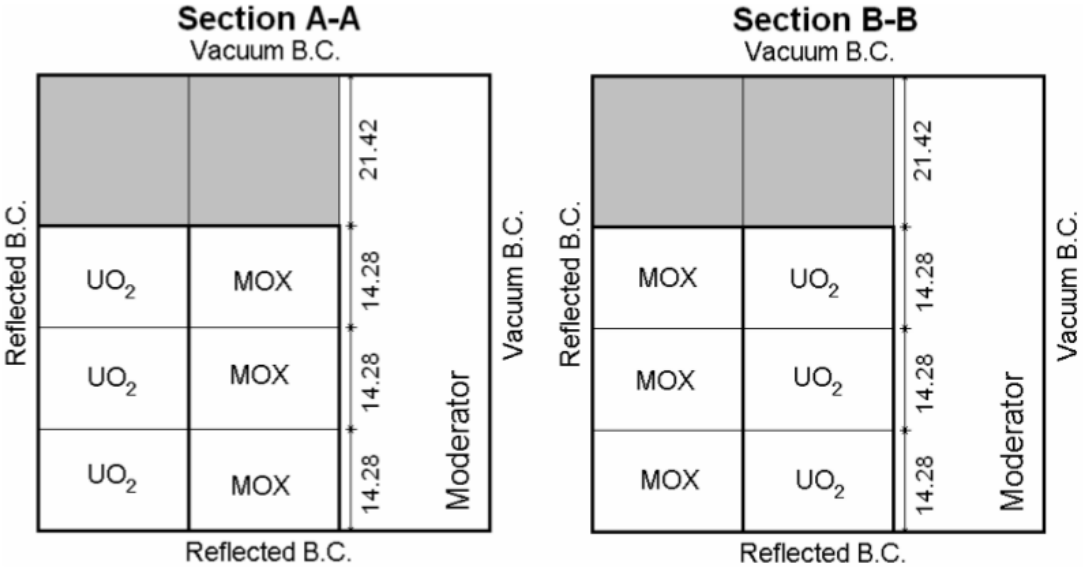

Figure A.1: C5G7 Rods Out control rod insertion depth (shaded sections).

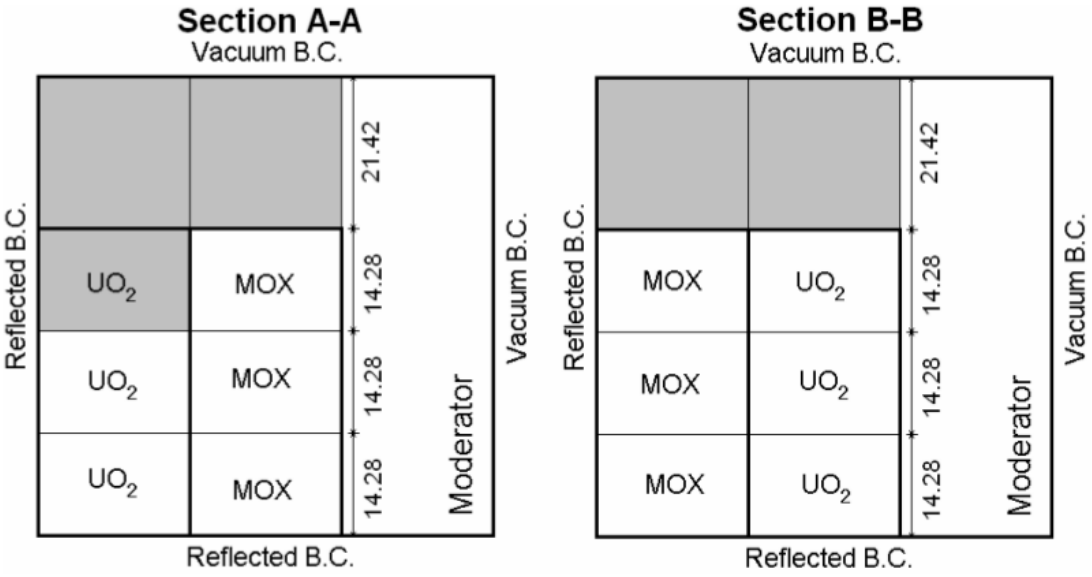

Figure A.2: C5G7 Rodded A control rod insertion depth (shaded sections).

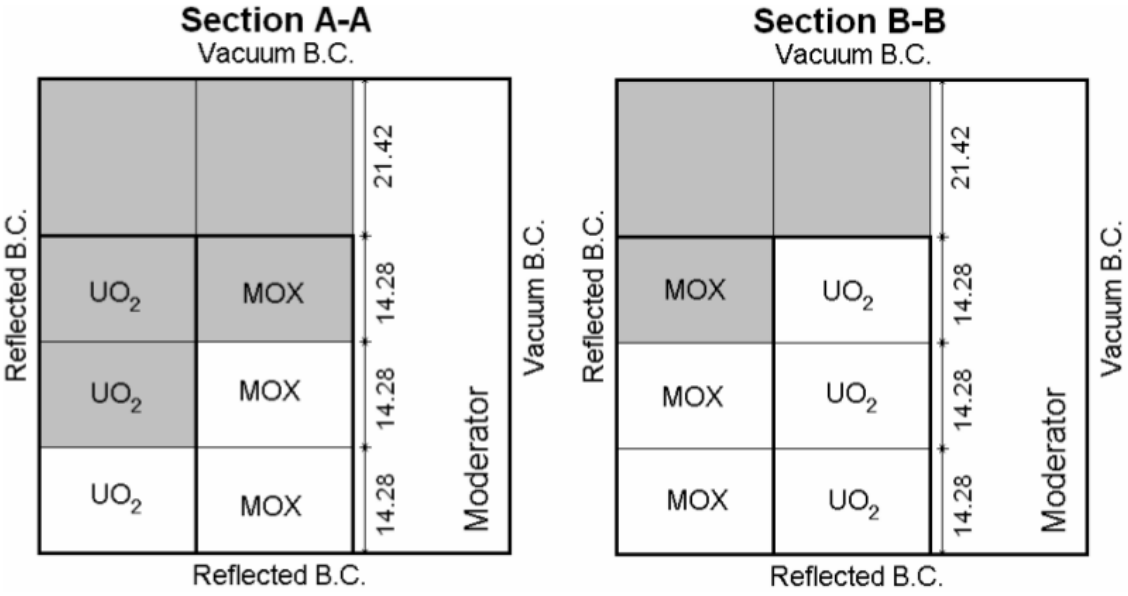

Figure A.3: C5G7 Rodded B control rod insertion depth (shaded sections).

Appendix B

Example Shift Input for C5G7 Rodded B Case

```
#########################################################################
## c5g7_roddedb.py
## nsk
#########################################################################

import os, sys, math, string, copy

# pykba equation type
from rtk_mg import *

##-------------------------------------------------------------------##
## MAIN
##-------------------------------------------------------------------##

initialize(sys.argv)
```

```
if node() == 0:
    print "Denovo - pykba Python Front-End"
    print "-------------------------------"
    print "Release      : %16s" % (release())
    print "Release Date : %16s" % (release_date())
    print "Build Date   : %16s" % (build_date())
    print

timer = Timer()
timer.start()

##-----------------------------------------------------------##
## MATERIAL CROSS SECTIONS
##-----------------------------------------------------------##
## The following material data is taken directly from the
   published benchmark
## The total cross sections are actually transport-corrected
   cross sections
## According to the benchmark paperwork, only the transport-
   corrected total
## cross section should be used, not the actual total cross
   section.
## NOTE:  The tables in the benchmark are transposed from the
    way Denovo reads
##        the scattering cross sections...

num_groups = 7
```

```
######## UO2 Fuel-Clad Macroscopic Cross Sections ##########
## Transport-corrected Total Cross Sections
T_UO2 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
T_UO2[0] = 1.77949e-1
T_UO2[1] = 3.29805e-1
T_UO2[2] = 4.80388e-1
T_UO2[3] = 5.54367e-1
T_UO2[4] = 3.11801e-1
T_UO2[5] = 3.95168e-1
T_UO2[6] = 5.64406e-1

## Fission Cross Section
F_UO2 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
F_UO2[0] = 7.21206e-3
F_UO2[1] = 8.19301e-4
F_UO2[2] = 6.45320e-3
F_UO2[3] = 1.85648e-2
F_UO2[4] = 1.78084e-2
F_UO2[5] = 8.30348e-2
F_UO2[6] = 2.16004e-1

## Nu
N_UO2 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
N_UO2[0] = 2.78145
N_UO2[1] = 2.47443
N_UO2[2] = 2.43383
N_UO2[3] = 2.43380
```

```
N_UO2[4] = 2.43380
N_UO2[5] = 2.43380
N_UO2[6] = 2.43380

## Chi
C_UO2 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
C_UO2[0] = 5.87910e-1
C_UO2[1] = 4.11760e-1
C_UO2[2] = 3.39060e-4
C_UO2[3] = 1.17610e-7
C_UO2[4] = 0.00000000
C_UO2[5] = 0.00000000
C_UO2[6] = 0.00000000

## Scattering Matrix for UO2 Fuel-Clad (Macroscopic)
S_UO2 = [ [[]], [[]], [[]], [[]], [[]], [[]], [[]]]
S_UO2[0] = [[1.27537e-1]]
S_UO2[1] = [[4.23780e-2], [3.24456e-1]]
S_UO2[2] = [[9.43740e-6], [1.63140e-3], [4.50940e-1]]
S_UO2[3] = [[5.51630e-9], [3.14270e-9], [2.67920e-3],
    [4.52565e-1], [1.25250e-4]]
S_UO2[4] = [[0.00000000], [0.00000000], [0.00000000],
    [5.56640e-3], [2.71401e-1], [1.29680e-3]]
S_UO2[5] = [[0.00000000], [0.00000000], [0.00000000],
    [0.00000000], [1.02550e-2], [2.65802e-1], [8.54580e-3]]
S_UO2[6] = [[0.00000000], [0.00000000], [0.00000000],
    [0.00000000], [1.00210e-8], [1.68090e-2], [2.73080e-1]]
```

```
## Upscattering Matrix
U_UO2 = [ [], [], [], [], [], [], [] ]
U_UO2[0] = []
U_UO2[1] = []
U_UO2[2] = []
U_UO2[3] = [4]
U_UO2[4] = [5]
U_UO2[5] = [6]
U_UO2[6] = []

######## 4.3% MOX Fuel-Clad Macroscopic Cross-Sections
   ############
## Transport-corrected Total Cross Sections
T_MOX43 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
T_MOX43[0] = 1.78731e-1
T_MOX43[1] = 3.30849e-1
T_MOX43[2] = 4.83772e-1
T_MOX43[3] = 5.66922e-1
T_MOX43[4] = 4.26227e-1
T_MOX43[5] = 6.78997e-1
T_MOX43[6] = 6.82852e-1

## Fission Cross-Sections
F_MOX43 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
F_MOX43[0] = 7.62704e-3
F_MOX43[1] = 8.76898e-4
F_MOX43[2] = 5.69835e-3
F_MOX43[3] = 2.28872e-2
```

```
F_MOX43[4] = 1.07635e-2
F_MOX43[5] = 2.32757e-1
F_MOX43[6] = 2.48968e-1

## Nu Cross-Sections
N_MOX43 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
N_MOX43[0] = 2.85209
N_MOX43[1] = 2.89099
N_MOX43[2] = 2.85486
N_MOX43[3] = 2.86073
N_MOX43[4] = 2.85447
N_MOX43[5] = 2.86415
N_MOX43[6] = 2.86780

## Chi Cross-Sections
C_MOX43 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
C_MOX43[0] = 5.87910e-1
C_MOX43[1] = 4.11760e-1
C_MOX43[2] = 3.39060e-4
C_MOX43[3] = 1.17610e-7
C_MOX43[4] = 0.00000000
C_MOX43[5] = 0.00000000
C_MOX43[6] = 0.00000000

## Scattering Matrix for 4.3% MOX Fuel-Clad (Macroscopic)
S_MOX43 = [ [[]], [[]], [[]], [[]], [[]], [[]], [[]] ]
S_MOX43[0] = [[1.28876e-1]]
S_MOX43[1] = [[4.14130e-2], [3.25452e-1]]
```

```
S_MOX43[2] = [[8.22900e-6], [1.63950e-3], [4.53188e-1]]
S_MOX43[3] = [[5.04050e-9], [1.59820e-9], [2.61420e-3],
    [4.57173e-1], [1.60460e-4]]
S_MOX43[4] = [[0.00000000], [0.00000000], [0.00000000],
    [5.53940e-3], [2.76814e-1], [2.00510e-3]]
S_MOX43[5] = [[0.00000000], [0.00000000], [0.00000000],
    [0.00000000], [9.31270e-3], [2.52962e-1], [8.49480e-3]]
S_MOX43[6] = [[0.00000000], [0.00000000], [0.00000000],
    [0.00000000], [9.16560e-9], [1.48500e-2], [2.65007e-1]]

## Upscattering Matrix
U_MOX43 = [ [], [], [], [], [], [], [] ]
U_MOX43[0] = []
U_MOX43[1] = []
U_MOX43[2] = []
U_MOX43[3] = [4]
U_MOX43[4] = [5]
U_MOX43[5] = [6]
U_MOX43[6] = []

########## 7.0% MO Fuel-Clad Macroscopic Cross-Sections
    ##########
## Transport-corrected total cross section
T_MOX70 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
T_MOX70[0] = 1.81323e-1
T_MOX70[1] = 3.34368e-1
T_MOX70[2] = 4.93785e-1
T_MOX70[3] = 5.91216e-1
```

```
T_MOX70[4] = 4.74198e-1
T_MOX70[5] = 8.33601e-1
T_MOX70[6] = 8.53603e-1

## Fission cross section
F_MOX70 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
F_MOX70[0] = 8.25446e-3
F_MOX70[1] = 1.32565e-3
F_MOX70[2] = 8.42156e-3
F_MOX70[3] = 3.28730e-2
F_MOX70[4] = 1.59636e-2
F_MOX70[5] = 3.23794e-1
F_MOX70[6] = 3.62803e-1

## Nu
N_MOX70 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
N_MOX70[0] = 2.88498
N_MOX70[1] = 2.91079
N_MOX70[2] = 2.86574
N_MOX70[3] = 2.87063
N_MOX70[4] = 2.86714
N_MOX70[5] = 2.86658
N_MOX70[6] = 2.87539

## Chi
C_MOX70 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
C_MOX70[0] = 5.87910e-1
C_MOX70[1] = 4.11760e-1
```

```
C_MOX70[2] = 3.39060e-4
C_MOX70[3] = 1.17610e-7
C_MOX70[4] = 0.00000000
C_MOX70[5] = 0.00000000
C_MOX70[6] = 0.00000000

## Scattering Matrix for 7.0% MOX Fuel-Clad (Macroscopic)
S_MOX70 = [ [[]], [[]], [[]], [[]], [[]], [[]], [[]] ]
S_MOX70[0] = [[1.30457e-1]]
S_MOX70[1] = [[4.17920e-2], [3.28428e-1]]
S_MOX70[2] = [[8.51050e-6], [1.64360e-3], [4.58371e-1]]
S_MOX70[3] = [[5.13290e-9], [2.20170e-9], [2.53310e-3],
    [4.63709e-1], [1.76190e-4]]
S_MOX70[4] = [[0.00000000], [0.00000000], [0.00000000],
    [5.47660e-3], [2.82313e-1], [2.27600e-3]]
S_MOX70[5] = [[0.00000000], [0.00000000], [0.00000000],
    [0.00000000], [8.72890e-3], [2.49751e-1], [8.86450e-3]]
S_MOX70[6] = [[0.00000000], [0.00000000], [0.00000000],
    [0.00000000], [9.00160e-9], [1.31140e-2], [2.59529e-1]]

## Upscattering Matrix
U_MOX70 = [ [], [], [], [], [], [], [] ]
U_MOX70[0] = []
U_MOX70[1] = []
U_MOX70[2] = []
U_MOX70[3] = [4]
U_MOX70[4] = [5]
U_MOX70[5] = [6]
```

```
U_MOX70[6] = []

################ 8.7% MOX Fuel-Clad Macroscopic Cross-sections
    ################
## Total cross sections
T_MOX87 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
T_MOX87[0] = 1.83045e-1
T_MOX87[1] = 3.36705e-1
T_MOX87[2] = 5.00507e-1
T_MOX87[3] = 6.06174e-1
T_MOX87[4] = 5.02754e-1
T_MOX87[5] = 9.21028e-1
T_MOX87[6] = 9.55231e-1

## Fission cross sections
F_MOX87 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
F_MOX87[0] = 8.67209e-3
F_MOX87[1] = 1.62426e-3
F_MOX87[2] = 1.02716e-2
F_MOX87[3] = 3.90447e-2
F_MOX87[4] = 1.92576e-2
F_MOX87[5] = 3.74888e-1
F_MOX87[6] = 4.30599e-1

## Nu
N_MOX87 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
N_MOX87[0] = 2.90426
N_MOX87[1] = 2.91795
```

```
N_MOX87[2] = 2.86986
N_MOX87[3] = 2.87491
N_MOX87[4] = 2.87175
N_MOX87[5] = 2.86752
N_MOX87[6] = 2.87808

## Chi
C_MOX87 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
C_MOX87[0] = 5.87910e-1
C_MOX87[1] = 4.11760e-1
C_MOX87[2] = 3.39060e-4
C_MOX87[3] = 1.17610e-7
C_MOX87[4] = 0.00000000
C_MOX87[5] = 0.00000000
C_MOX87[6] = 0.00000000

## Scattering Matrix for 8.7% MOX Fuel-Clad (Macroscopic)
S_MOX87 = [ [[]], [[]],[[]], [[]], [[]], [[]], [[]] ]
S_MOX87[0] = [[1.31504e-1]]
S_MOX87[1] = [[4.20460e-2], [3.30403e-1]]
S_MOX87[2] = [[8.69720e-6], [1.64630e-3], [4.61792e-1]]
S_MOX87[3] = [[5.19380e-9], [2.60060e-9], [2.47490e-3],
    [4.68021e-1], [1.85970e-4]]
S_MOX87[4] = [[0.00000000], [0.00000000], [0.00000000],
    [5.43300e-3], [2.85771e-1], [2.39160e-3]]
S_MOX87[5] = [[0.00000000], [0.00000000], [0.00000000],
    [0.00000000], [8.39730e-3], [2.47614e-1], [8.96810e-3]]
```

```
S_MOX87[6] = [[0.00000000], [0.00000000], [0.00000000],
    [0.00000000], [8.92800e-9], [1.23220e-2], [2.56093e-1]]

## Upscattering Matrix
U_MOX87 = [ [], [], [], [], [], [], [] ]
U_MOX87[0] = []
U_MOX87[1] = []
U_MOX87[2] = []
U_MOX87[3] = [4]
U_MOX87[4] = [5]
U_MOX87[5] = [6]
U_MOX87[6] = []

######### Fission Chamber Macroscopic Cross-sections
    ###############
## Transport corrected Total Cross-Section
T_FISCH = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
T_FISCH[0] = 1.26032e-1
T_FISCH[1] = 2.93160e-1
T_FISCH[2] = 2.84250e-1
T_FISCH[3] = 2.81020e-1
T_FISCH[4] = 3.34460e-1
T_FISCH[5] = 5.65640e-1
T_FISCH[6] = 1.17214

## Fission Cross-section
F_FISCH = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
F_FISCH[0] = 4.79002e-9
```

```
F_FISCH[1] = 5.82564e-9
F_FISCH[2] = 4.63719e-7
F_FISCH[3] = 5.24406e-6
F_FISCH[4] = 1.45390e-7
F_FISCH[5] = 7.14972e-7
F_FISCH[6] = 2.08041e-6

## Nu
N_FISCH = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
N_FISCH[0] = 2.76283
N_FISCH[1] = 2.46239
N_FISCH[2] = 2.43380
N_FISCH[3] = 2.43380
N_FISCH[4] = 2.43380
N_FISCH[5] = 2.43380
N_FISCH[6] = 2.43380

## Chi
C_FISCH = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
C_FISCH[0] = 5.87910e-1
C_FISCH[1] = 4.11760e-1
C_FISCH[2] = 3.39060e-4
C_FISCH[3] = 1.17610e-7
C_FISCH[4] = 0.00000000
C_FISCH[5] = 0.00000000
C_FISCH[6] = 0.00000000

## Scattering Matrix for Fission Chamber (Macroscopic)
```

```
S_FISCH = [ [[]], [[]], [[]], [[]], [[]], [[]], [[]] ]
S_FISCH[0] = [[6.61659e-2]]
S_FISCH[1] = [[5.90700e-2], [2.40377e-1]]
S_FISCH[2] = [[2.83340e-4], [5.24350e-2], [1.83425e-1]]
S_FISCH[3] = [[1.46220e-6], [2.49900e-4], [9.22880e-2],
    [7.90769e-2], [3.73400e-5]]
S_FISCH[4] = [[2.06420e-8], [1.92390e-5], [6.93650e-3],
    [1.69990e-1], [9.97570e-2], [9.17420e-4]]
S_FISCH[5] = [[0.00000000], [2.98750e-6], [1.07900e-3],
    [2.58600e-2], [2.06790e-1], [3.16774e-1], [4.97930e-2]]
S_FISCH[6] = [[0.00000000], [4.21400e-7], [2.05430e-4],
    [4.92560e-3], [2.44780e-2], [2.38760e-1], [1.09910    ]]

## Upscattering Matrix
U_FISCH = [ [], [], [], [], [], [], [] ]
U_FISCH[0] = []
U_FISCH[1] = []
U_FISCH[2] = []
U_FISCH[3] = [4]
U_FISCH[4] = [5]
U_FISCH[5] = [6]
U_FISCH[6] = []

############## Guide Tube Macroscopic Cross-Sections
    ###############
## Transport Cross-Section
T_GUIDT = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
T_GUIDT[0] = 1.26032e-1
```

```
T_GUIDT[1] = 2.93160e-1
T_GUIDT[2] = 2.84240e-1
T_GUIDT[3] = 2.80960e-1
T_GUIDT[4] = 3.34440e-1
T_GUIDT[5] = 5.65640e-1
T_GUIDT[6] = 1.17215

## Scattering Matrix for Guide Tube (Macroscopic)
S_GUIDT = [ [[]], [[]], [[]], [[]], [[]], [[]], [[]] ]
S_GUIDT[0] = [[6.61659e-2]]
S_GUIDT[1] = [[5.90700e-2], [2.40377e-1]]
S_GUIDT[2] = [[2.83340e-4], [5.24350e-2], [1.83297e-1]]
S_GUIDT[3] = [[1.46220e-6], [2.49900e-4], [9.23970e-2],
    [7.88511e-2], [3.73330e-5]]
S_GUIDT[4] = [[2.06420e-8], [1.92390e-5], [6.94460e-3],
    [1.70140e-1], [9.97372e-2], [9.17260e-4]]
S_GUIDT[5] = [[0.00000000], [2.98750e-6], [1.08030e-3],
    [2.58810e-2], [2.06790e-1], [3.16765e-1], [4.97920e-2]]
S_GUIDT[6] = [[0.00000000], [4.21400e-7], [2.05670e-4],
    [4.92970e-3], [2.44780e-2], [2.38770e-1], [1.09912    ]]

## Upscattering Matrix
U_GUIDT = [ [], [], [], [], [], [], [] ]
U_GUIDT[0] = []
U_GUIDT[1] = []
U_GUIDT[2] = []
U_GUIDT[3] = [4]
U_GUIDT[4] = [5]
```

```
U_GUIDT[5] = [6]
U_GUIDT[6] = []

################ Control Rod Macroscopic Cross-Sections
    ###################
## Transport Cross-Section
T_CONTROL = [0.0, 00, 0.0, 0.0, 0.0, 0.0, 0.0]
T_CONTROL[0] = 2.16768e-1
T_CONTROL[1] = 4.80098e-1
T_CONTROL[2] = 8.86369e-1
T_CONTROL[3] = 9.70009e-1
T_CONTROL[4] = 9.10482e-1
T_CONTROL[5] = 1.13775
T_CONTROL[6] = 1.84048

## Scattering Matrix for Control Rod (Macroscopic)
S_CONTROL = [ [[]], [[]], [[]], [[]], [[]], [[]], [[]] ]
S_CONTROL[0] = [[1.70563e-1]]
S_CONTROL[1] = [[4.44012e-2], [4.71050e-1]]
S_CONTROL[2] = [[9.83670e-5], [6.85480e-4 ], [8.01859e-1]]
S_CONTROL[3] = [[1.27786e-7], [3.91395e-10], [7.20132e-4],
    [5.70752e-1], [6.55562e-5]]
S_CONTROL[4] = [[0.00000000], [0.000000000], [0.00000000],
    [1.46015e-3], [2.07838e-1], [1.02427e-3]]
S_CONTROL[5] = [[0.00000000], [0.000000000], [0.00000000],
    [0.00000000], [3.81486e-3], [2.02465e-1], [3.53043e-3]]
S_CONTROL[6] = [[0.00000000], [0.000000000], [0.00000000],
    [0.00000000], [3.69760e-9], [4.75290e-3], [6.58597e-1]]
```

```
## Upscattering Matrix
U_CONTROL = [ [], [], [], [], [], [], [] ]
U_CONTROL[0] = []
U_CONTROL[1] = []
U_CONTROL[2] = []
U_CONTROL[3] = [4]
U_CONTROL[4] = [5]
U_CONTROL[5] = [6]
U_CONTROL[6] = []

################## Moderator 1 Macroscopic Cross-Sections
   ##################
## Transport-corrected Total Cross Section
T_MOD1 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
T_MOD1[0] = 1.59206e-1
T_MOD1[1] = 4.12970e-1
T_MOD1[2] = 5.90310e-1
T_MOD1[3] = 5.84350e-1
T_MOD1[4] = 7.18000e-1
T_MOD1[5] = 1.25445
T_MOD1[6] = 2.65038

## Scattering Matrix for Moderator (Macroscopic)
S_MOD1 = [ [[]], [[]], [[]], [[]], [[]], [[]], [[]] ]
S_MOD1[0] = [[4.44777e-2]]
S_MOD1[1] = [[1.13400e-1], [2.82334e-1]]
S_MOD1[2] = [[7.23470e-4], [1.29940e-1], [3.45256e-1]]
```

```
S_MOD1[3] = [[3.74990e-6], [6.23400e-4], [2.24570e-1],
    [9.10284e-2], [7.14370e-5]]
S_MOD1[4] = [[5.31840e-8], [4.80020e-5], [1.69990e-2],
    [4.15510e-1], [1.39138e-1], [2.21570e-3]]
S_MOD1[5] = [[0.00000000], [7.44860e-6], [2.64430e-3],
    [6.37320e-2], [5.11820e-1], [6.99913e-1], [1.32440e-1]]
S_MOD1[6] = [[0.00000000], [1.04550e-6], [5.03440e-4],
    [1.21390e-2], [6.12290e-2], [5.37320e-1], [2.48070    ]]

## Upscattering Matrix
U_MOD1 = [ [], [], [], [], [], [], [] ]
U_MOD1[0] = []
U_MOD1[1] = []
U_MOD1[2] = []
U_MOD1[3] = [4]
U_MOD1[4] = [5]
U_MOD1[5] = [6]
U_MOD1[6] = []

###################### Create nuf vectors
NUF_UO2 = []
NUF_MOX43 = []
NUF_MOX70 = []
NUF_MOX87 = []
NUF_FISCH = []
for i in range(0, 7):
    NUF_UO2.append( N_UO2[i] * F_UO2[i] )
    NUF_MOX43.append( N_MOX43[i] * F_MOX43[i] )
```

```
    NUF_MOX70.append( N_MOX70[i] * F_MOX70[i] )
    NUF_MOX87.append( N_MOX87[i] * F_MOX87[i] )
    NUF_FISCH.append( N_FISCH[i] * F_FISCH[i] )

##-------------------------------------------------------------##
## DB
##-------------------------------------------------------------##

db = DB("pykba")

# mg cross section data
db.insert("downscatter", 0, 1)
db.insert("num_groups", num_groups)
db.insert("Pn_order", 0)

# groups
n_bnd = [2.0e7, 1.0e6, 5.0e5, 3.0, 0.625, 0.1, 0.02, 1.0e-5]
db.insert("neutron_bnd", n_bnd)

# initial fission source box
fs = [0.0, 42.84, 0.0, 42.84, 0.0, 42.84]

# k-code database
db.add_db("kcode_db", "kcode")
db.insert("kcode_db", "init_fission_src", fs)
db.insert("kcode_db", "keff_init", 1.07777)
db.insert("kcode_db", "num_cycles", 1200)
db.insert("kcode_db", "num_inactive_cycles", 500)
```

```
db.insert("kcode_db", "Np", 10000000)

# tally database
db.add_db("tally_db", "tally")

# general input
db.insert("mc_diag_frac", 1.1)
db.insert("seed", 32442)

# boundary mesh
xb = [-0.000001, 64.26001]
yb = [-0.000001, 64.26001]
zb = [-0.000001, 64.26001]

db.insert("x_bnd_mesh", xb)
db.insert("y_bnd_mesh", yb)
db.insert("z_bnd_mesh", zb)

##-----------------------------------------------------------##
## GEOMETRY
##-----------------------------------------------------------##
## MOD    -> 0
## UO2    -> 1
## MOX 43 -> 2
## MOX 70 -> 3
## MOX 87 -> 4
## GT     -> 5  Guide tubes in the core (no guide tubes above
    the core)
```

```
## FC      -> 6  Fission chambers in the core
## CR_1    -> 7  Control rods in the core
## FC_1    -> 8  In the top reflector region
## CR      -> 9  In the top reflector region
## mod     -> 0  moderator block above fuel pins
## REF     -> 0  Moderator block that is the reflector region
## TOPREF -> 0  Moderator block above the reflector region

pitch  = 1.26
height = 14.28

# pins (measurements in cm)
uo2 = RTK_Cell(1, 0.54, 0, pitch, height,1)
m43 = RTK_Cell(2, 0.54, 0, pitch, height,1)
m70 = RTK_Cell(3, 0.54, 0, pitch, height,1)
m87 = RTK_Cell(4, 0.54, 0, pitch, height,1)
gt  = RTK_Cell(5, 0.54, 0, pitch, height,1)
fc  = RTK_Cell(6, 0.54, 0, pitch, height,1)
cr_1= RTK_Cell(7, 0.54, 0, pitch, height,1)
fc_1= RTK_Cell(8, 0.54, 0, pitch, 21.42,1)
cr  = RTK_Cell(9, 0.54, 0, pitch, 21.42,1)
mod = RTK_Cell(0, 1.26, 21.42,1)

# not really pins, but blocks of moderator/reflector
ref = RTK_Cell(0, 21.42, height,1)
topref = RTK_Cell(0, 21.42, 21.42,1)

# reflector lattice
```

```
ref_1 = Lattice(1, 1, 1, 1)
ref_1.assign_object(ref, 0)
ref_1.complete(0.0, 0.0, 0.0)

# top reflector not over core but over other reflector (see
    previous)
topref_1 = Lattice (1, 1, 1, 1)
topref_1.assign_object(topref, 0)
topref_1.complete(0.0, 0.0, 0.0)

# top reflector over core (lattice w/ rods, tubes, and
    chambers)
topref_2 = Lattice(17, 17, 1, 3)
topref_2.assign_object(mod, 0)
topref_2.assign_object(cr, 1)
topref_2.assign_object(fc_1, 2)

topref_2.set_id(5,  2,  0, 1)
topref_2.set_id(8,  2,  0, 1)
topref_2.set_id(11, 2,  0, 1)

topref_2.set_id(3,  3,  0, 1)
topref_2.set_id(13, 3,  0, 1)

topref_2.set_id(2,  5,  0, 1)
topref_2.set_id(5,  5,  0, 1)
topref_2.set_id(8,  5,  0, 1)
topref_2.set_id(11, 5,  0, 1)
```

```
topref_2.set_id(14, 5,  0, 1)

topref_2.set_id(2,  8,  0, 1)
topref_2.set_id(5,  8,  0, 1)
topref_2.set_id(8,  8,  0, 2)
topref_2.set_id(11, 8,  0, 1)
topref_2.set_id(14, 8,  0, 1)

topref_2.set_id(2,  11, 0, 1)
topref_2.set_id(5,  11, 0, 1)
topref_2.set_id(8,  11, 0, 1)
topref_2.set_id(11, 11, 0, 1)
topref_2.set_id(14, 11, 0, 1)

topref_2.set_id(3,  13, 0, 1)
topref_2.set_id(13, 13, 0, 1)

topref_2.set_id(5,  14, 0, 1)
topref_2.set_id(8,  14, 0, 1)
topref_2.set_id(11, 14, 0, 1)

topref_2.complete(0.0, 0.0, 0.0)

# UO2 lattice
uo2_l = Lattice(17, 17, 1, 3)
uo2_l.assign_object(uo2, 0)
uo2_l.assign_object(gt, 1)
uo2_l.assign_object(fc, 2)
```

```
uo2_l.set_id(5,  2,  0, 1)
uo2_l.set_id(8,  2,  0, 1)
uo2_l.set_id(11, 2,  0, 1)

uo2_l.set_id(3,  3,  0, 1)
uo2_l.set_id(13, 3,  0, 1)

uo2_l.set_id(2,  5,  0, 1)
uo2_l.set_id(5,  5,  0, 1)
uo2_l.set_id(8,  5,  0, 1)
uo2_l.set_id(11, 5,  0, 1)
uo2_l.set_id(14, 5,  0, 1)

uo2_l.set_id(2,  8,  0, 1)
uo2_l.set_id(5,  8,  0, 1)
uo2_l.set_id(8,  8,  0, 2)
uo2_l.set_id(11, 8,  0, 1)
uo2_l.set_id(14, 8,  0, 1)

uo2_l.set_id(2,  11, 0, 1)
uo2_l.set_id(5,  11, 0, 1)
uo2_l.set_id(8,  11, 0, 1)
uo2_l.set_id(11, 11, 0, 1)
uo2_l.set_id(14, 11, 0, 1)

uo2_l.set_id(3,  13, 0, 1)
uo2_l.set_id(13, 13, 0, 1)
```

```
uo2_l.set_id(5,  14, 0, 1)
uo2_l.set_id(8,  14, 0, 1)
uo2_l.set_id(11, 14, 0, 1)

uo2_l.complete(0.0, 0.0, 0.0)

# UO2 lattice with control rods in
uo2_2 = Lattice(17, 17, 1, 3)
uo2_2.assign_object(uo2, 0)
uo2_2.assign_object(cr_1, 1)
uo2_2.assign_object(fc, 2)

uo2_2.set_id(5,  2,  0, 1)
uo2_2.set_id(8,  2,  0, 1)
uo2_2.set_id(11, 2,  0, 1)

uo2_2.set_id(3,  3,  0, 1)
uo2_2.set_id(13, 3,  0, 1)

uo2_2.set_id(2,  5,  0, 1)
uo2_2.set_id(5,  5,  0, 1)
uo2_2.set_id(8,  5,  0, 1)
uo2_2.set_id(11, 5,  0, 1)
uo2_2.set_id(14, 5,  0, 1)

uo2_2.set_id(2,  8,  0, 1)
uo2_2.set_id(5,  8,  0, 1)
```

```
uo2_2.set_id(8,  8,  0, 2)
uo2_2.set_id(11, 8,  0, 1)
uo2_2.set_id(14, 8,  0, 1)

uo2_2.set_id(2,  11, 0, 1)
uo2_2.set_id(5,  11, 0, 1)
uo2_2.set_id(8,  11, 0, 1)
uo2_2.set_id(11, 11, 0, 1)
uo2_2.set_id(14, 11, 0, 1)

uo2_2.set_id(3,  13, 0, 1)
uo2_2.set_id(13, 13, 0, 1)

uo2_2.set_id(5,  14, 0, 1)
uo2_2.set_id(8,  14, 0, 1)
uo2_2.set_id(11, 14, 0, 1)

uo2_2.complete(0.0, 0.0, 0.0)

# MOX lattice
mox_l = Lattice(17, 17, 1, 5)
mox_l.assign_object(m43, 0)
mox_l.assign_object(m70, 1)
mox_l.assign_object(m87, 2)
mox_l.assign_object(gt, 3)
mox_l.assign_object(fc, 4)

for j in xrange(17):
```

```
    for i in xrange(17):
        mox_l.set_id(i, j, 0, 1)

for i in xrange(17):
    mox_l.set_id(i, 0, 0, 0)
for i in xrange(17):
    mox_l.set_id(i, 16, 0, 0)
for j in xrange(17):
    mox_l.set_id(0, j, 0, 0)
for j in xrange(17):
    mox_l.set_id(16, j, 0, 0)

for j in xrange(5, 12):
    for i in xrange(3, 14):
        mox_l.set_id(i, j, 0, 2)
for i in xrange(4, 13):
    mox_l.set_id(i, 4, 0, 2)
for i in xrange(5, 12):
    mox_l.set_id(i, 3, 0, 2)
for i in xrange(4, 13):
    mox_l.set_id(i, 12, 0, 2)
for i in xrange(5, 12):
    mox_l.set_id(i, 13, 0, 2)

mox_l.set_id(5,  2,  0, 3)
mox_l.set_id(8,  2,  0, 3)
mox_l.set_id(11, 2,  0, 3)
```

```
mox_l.set_id(3,  3,  0, 3)
mox_l.set_id(13, 3,  0, 3)

mox_l.set_id(2,  5,  0, 3)
mox_l.set_id(5,  5,  0, 3)
mox_l.set_id(8,  5,  0, 3)
mox_l.set_id(11, 5,  0, 3)
mox_l.set_id(14, 5,  0, 3)

mox_l.set_id(2,  8,  0, 3)
mox_l.set_id(5,  8,  0, 3)
mox_l.set_id(8,  8,  0, 4)
mox_l.set_id(11, 8,  0, 3)
mox_l.set_id(14, 8,  0, 3)

mox_l.set_id(2,  11, 0, 3)
mox_l.set_id(5,  11, 0, 3)
mox_l.set_id(8,  11, 0, 3)
mox_l.set_id(11, 11, 0, 3)
mox_l.set_id(14, 11, 0, 3)

mox_l.set_id(3,  13, 0, 3)
mox_l.set_id(13, 13, 0, 3)

mox_l.set_id(5,  14, 0, 3)
mox_l.set_id(8,  14, 0, 3)
mox_l.set_id(11, 14, 0, 3)
```

```
mox_1.complete(0.0, 0.0, 0.0)

# MOX lattice with control rods in
mox_2 = Lattice(17, 17, 1, 5)
mox_2.assign_object(m43, 0)
mox_2.assign_object(m70, 1)
mox_2.assign_object(m87, 2)
mox_2.assign_object(cr_1, 3)
mox_2.assign_object(fc, 4)

for j in xrange(17):
    for i in xrange(17):
        mox_2.set_id(i, j, 0, 1)

for i in xrange(17):
    mox_2.set_id(i, 0, 0, 0)
for i in xrange(17):
    mox_2.set_id(i, 16, 0, 0)
for j in xrange(17):
    mox_2.set_id(0, j, 0, 0)
for j in xrange(17):
    mox_2.set_id(16, j, 0, 0)

for j in xrange(5, 12):
    for i in xrange(3, 14):
        mox_2.set_id(i, j, 0, 2)
for i in xrange(4, 13):
    mox_2.set_id(i, 4, 0, 2)
```

```
for i in xrange(5, 12):
    mox_2.set_id(i, 3, 0, 2)
for i in xrange(4, 13):
    mox_2.set_id(i, 12, 0, 2)
for i in xrange(5, 12):
    mox_2.set_id(i, 13, 0, 2)

mox_2.set_id(5,  2,  0, 3)
mox_2.set_id(8,  2,  0, 3)
mox_2.set_id(11, 2,  0, 3)

mox_2.set_id(3,  3,  0, 3)
mox_2.set_id(13, 3,  0, 3)

mox_2.set_id(2,  5,  0, 3)
mox_2.set_id(5,  5,  0, 3)
mox_2.set_id(8,  5,  0, 3)
mox_2.set_id(11, 5,  0, 3)
mox_2.set_id(14, 5,  0, 3)

mox_2.set_id(2,  8,  0, 3)
mox_2.set_id(5,  8,  0, 3)
mox_2.set_id(8,  8,  0, 4)
mox_2.set_id(11, 8,  0, 3)
mox_2.set_id(14, 8,  0, 3)

mox_2.set_id(2,  11, 0, 3)
mox_2.set_id(5,  11, 0, 3)
```

```
mox_2.set_id(8,  11, 0, 3)
mox_2.set_id(11, 11, 0, 3)
mox_2.set_id(14, 11, 0, 3)

mox_2.set_id(3,  13, 0, 3)
mox_2.set_id(13, 13, 0, 3)

mox_2.set_id(5,  14, 0, 3)
mox_2.set_id(8,  14, 0, 3)
mox_2.set_id(11, 14, 0, 3)

mox_2.complete(0.0, 0.0, 0.0)

# core
core = Core(3, 3, 4, 7)
core.assign_object(ref_1, 0)
core.assign_object(uo2_l, 1)
core.assign_object(mox_l, 2)
core.assign_object(topref_1, 3)
core.assign_object(uo2_2, 4)
core.assign_object(mox_2, 5)
core.assign_object(topref_2, 6)

core.set_id(0, 0, 0, 1)
core.set_id(1, 0, 0, 2)
core.set_id(2, 0, 0, 0)
core.set_id(0, 1, 0, 2)
core.set_id(1, 1, 0, 1)
```

```
core.set_id(2, 1, 0, 0)
core.set_id(0, 2, 0, 0)
core.set_id(1, 2, 0, 0)
core.set_id(2, 2, 0, 0)
core.set_id(0, 0, 1, 4)
core.set_id(1, 0, 1, 2)
core.set_id(2, 0, 1, 0)
core.set_id(0, 1, 1, 2)
core.set_id(1, 1, 1, 1)
core.set_id(2, 1, 1, 0)
core.set_id(0, 2, 1, 0)
core.set_id(1, 2, 1, 0)
core.set_id(2, 2, 1, 0)
core.set_id(0, 0, 2, 4)
core.set_id(1, 0, 2, 5)
core.set_id(2, 0, 2, 0)
core.set_id(0, 1, 2, 5)
core.set_id(1, 1, 2, 1)
core.set_id(2, 1, 2, 0)
core.set_id(0, 2, 2, 0)
core.set_id(1, 2, 2, 0)
core.set_id(2, 2, 2, 0)
core.set_id(0, 0, 3, 6)
core.set_id(1, 0, 3, 6)
core.set_id(2, 0, 3, 3)
core.set_id(0, 1, 3, 6)
core.set_id(1, 1, 3, 6)
core.set_id(2, 1, 3, 3)
```

```
core.set_id(0, 2, 3, 3)
core.set_id(1, 2, 3, 3)
core.set_id(2, 2, 3, 3)

reflect = Vec_Int(6, 1)
reflect[1] = 0
reflect[3] = 0
reflect[5] = 0
core.set_reflecting(reflect)
core.complete(0.0, 0.0, 0.0)

# geometry
geometry = Geometry()
geometry.build_geometry(core)

# tally mesh
## ([number of planes],[initial fill])

x= Vec_Dbl(35, 0.0)
y= Vec_Dbl(35, 0.0)
z= Vec_Dbl(4, 0.0)

## x-planes
for n in xrange(34):
    x[n+1] = x[n] + pitch


## y-planes
```

```
for o in xrange(34):
    y[o+1] = y[o] + pitch

## z-planes
for n in xrange(3):
    z[n+1] = z[n] + height

db.insert("tally_db", "tally_mesh_x", x)
db.insert("tally_db", "tally_mesh_y", y)
db.insert("tally_db", "tally_mesh_z", z)

##-----------------------------------------------------------##
## MANAGER
##-----------------------------------------------------------##

# make manager, material, and angles
manager = Manager()
manager.partition(db)

##-----------------------------------------------------------##
## MATERIAL SETUP
##-----------------------------------------------------------##
## MOD     -> 0
## UO2     -> 1
## MOX 43 -> 2
## MOX 70 -> 3
## MOX 87 -> 4
```

```
## GT      -> 5  Guide tubes in the core (no guide tubes above
    the core)
## FC      -> 6  Fission chambers in the core
## CR_1    -> 7  Control rods in the core
## FC_1    -> 8  In the top reflector region
## CR      -> 9  In the top reflector region
## mod     -> 0  moderator block above fuel pins
## REF     -> 0  Moderator block that is the reflector region
## TOPREF -> 0  Moderator block above the reflector region

# make xs db
xsdb = XS_DB(db)

# physics
physics = Physics()

xsdb.set_num(10)

for g in xrange(0, num_groups):
    xsdb.assign_upscatter(0, g, T_MOD1[g], U_MOD1[g], S_MOD1[
        g])
    xsdb.assign_upscatter(1, g, T_UO2[g], U_UO2[g], S_UO2[g])
    xsdb.assign_upscatter(2, g, T_MOX43[g], U_MOX43[g],
        S_MOX43[g])
    xsdb.assign_upscatter(3, g, T_MOX70[g], U_MOX70[g],
        S_MOX70[g])
    xsdb.assign_upscatter(4, g, T_MOX87[g], U_MOX87[g],
        S_MOX87[g])
```

```
    xsdb.assign_upscatter(5, g, T_GUIDT[g], U_GUIDT[g],
        S_GUIDT[g])
    xsdb.assign_upscatter(6, g, T_FISCH[g], U_FISCH[g],
        S_FISCH[g])
    xsdb.assign_upscatter(7, g, T_CONTROL[g], U_CONTROL[g],
        S_CONTROL[g])
    xsdb.assign_upscatter(8, g, T_FISCH[g], U_FISCH[g],
        S_FISCH[g])
    xsdb.assign_upscatter(9, g, T_CONTROL[g], U_CONTROL[g],
        S_CONTROL[g])

## Assign fission data
xsdb.assign_fission(1, NUF_UO2, C_UO2)
xsdb.assign_fission(2, NUF_MOX43, C_MOX43)
xsdb.assign_fission(3, NUF_MOX70, C_MOX70)
xsdb.assign_fission(4, NUF_MOX87, C_MOX87)
xsdb.assign_fission(6, NUF_FISCH, C_FISCH)
xsdb.assign_fission(8, NUF_FISCH, C_FISCH)

physics.build_physics(xsdb)

##------------------------------------------------------------##
## SOLVE
##------------------------------------------------------------##

manager.set_geometry(geometry)
manager.set_physics(geometry, physics)
manager.setup_kcode()
```

```
db.output()

tally = Tally()

manager.solve(tally)

##------------------------------------------------------------------##
## OUTPUT
##------------------------------------------------------------------##

tally_field = tally.get_tally_field()
tally_var = tally.get_variance_field()

if node() == 0:
    for j in xrange(34):
        for i in xrange(34):
            for k in xrange(3):
                cell = k + (i + j * 34) * 3
                print "%5i %12.7f %7.5e" % (cell+1,
                   tally_field(cell), tally_var(cell))

if tally_field.assigned() and tally_var.assigned():

    silo = Tally_Mesh_SILO()

#     silo.open("c5g7-shift-3-1")
```

```
    silo.add("flux_tally", tally_field)

    silo.add("tally_var", tally_var)

    silo.close()

##------------------------------------------------------------##
## TIMING
##------------------------------------------------------------##

timer.stop()
time = timer.wall_clock()

if node() == 0:
    print "\n"
    print "TIMING : Problem ran in %16.6e seconds." % (time)

    keys = timer_keys()
    if len(keys) > 0:
        print
          "_____________________________________________________"

        for key in keys:
            print "%30s : %16.6e" % (key, timer_value(key) /
                time)
        print
          "_____________________________________________________"
```

```
##-------------------------------------------------------------------##

manager.close()
finalize()

#######################################################################
## end of unrodded_2D.py
#######################################################################
```

Appendix C

Benchmark Scaling Results Graphs

C5G7 Rods Out Graphs

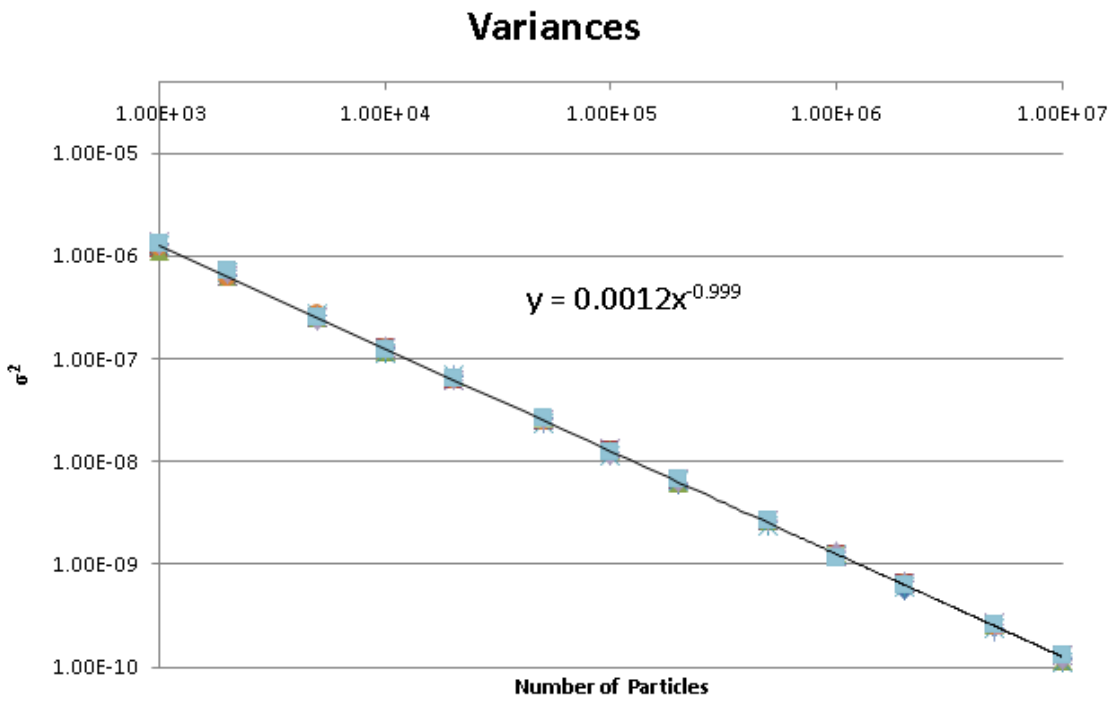

Figure C.1: C5G7 Rods Out variances vs. number of particles per cycle.

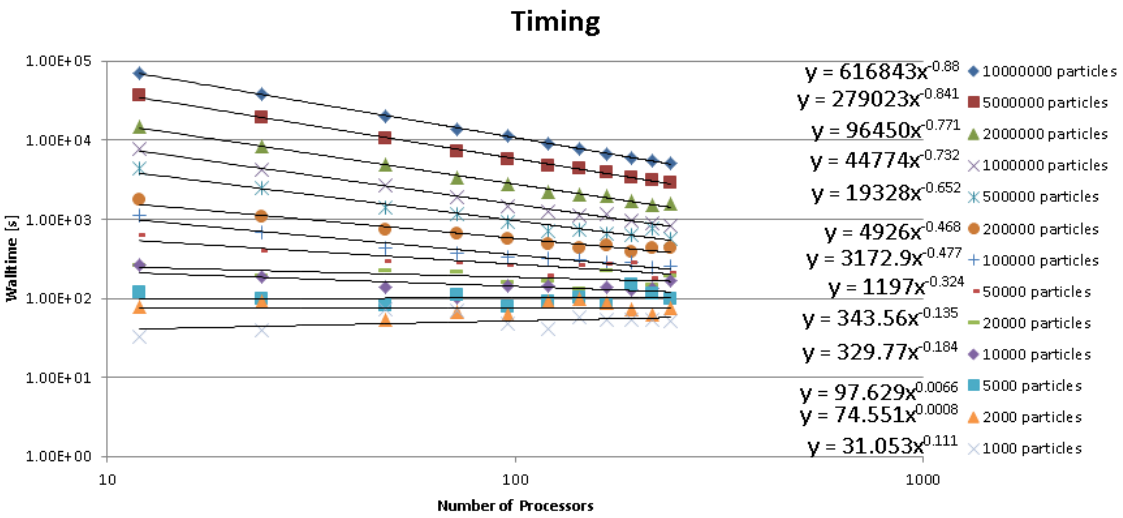

Figure C.2: C5G7 Rods Out run time vs. number of processors.

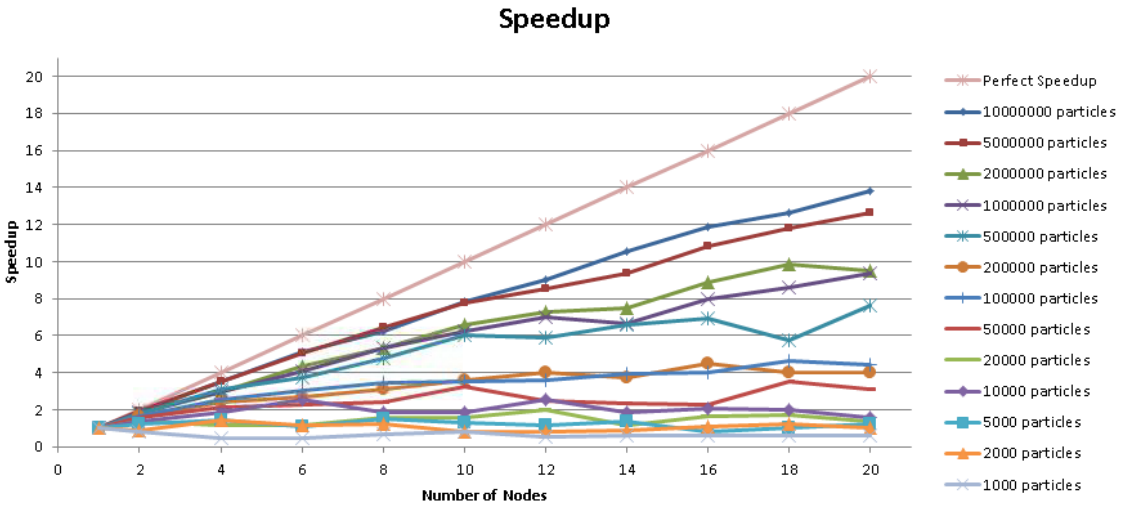

Figure C.3: C5G7 Rods Out speedup vs. number of nodes.

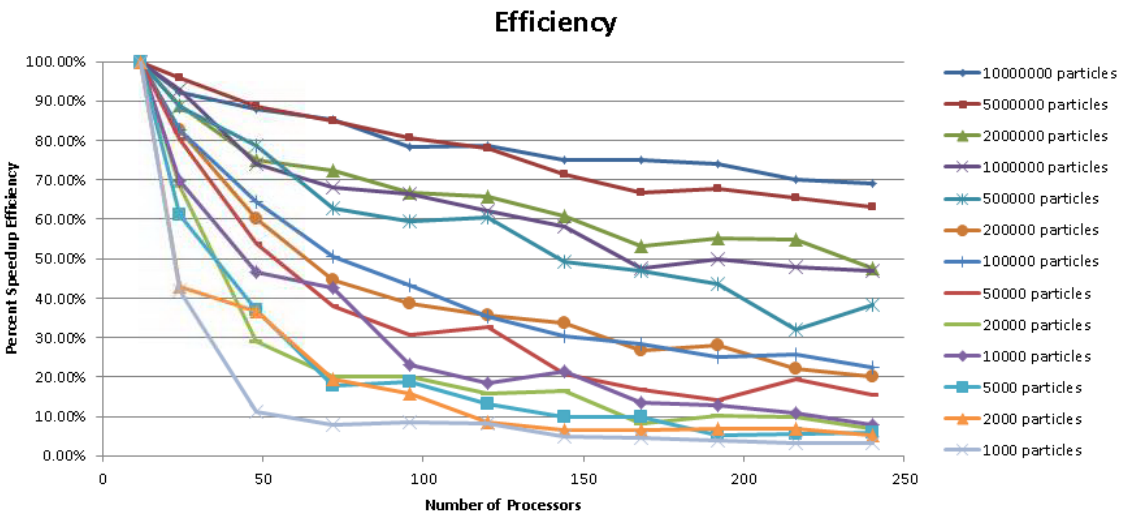

Figure C.4: C5G7 Rods Out efficiency vs. number of processors.

C5G7 Rodded A Graphs

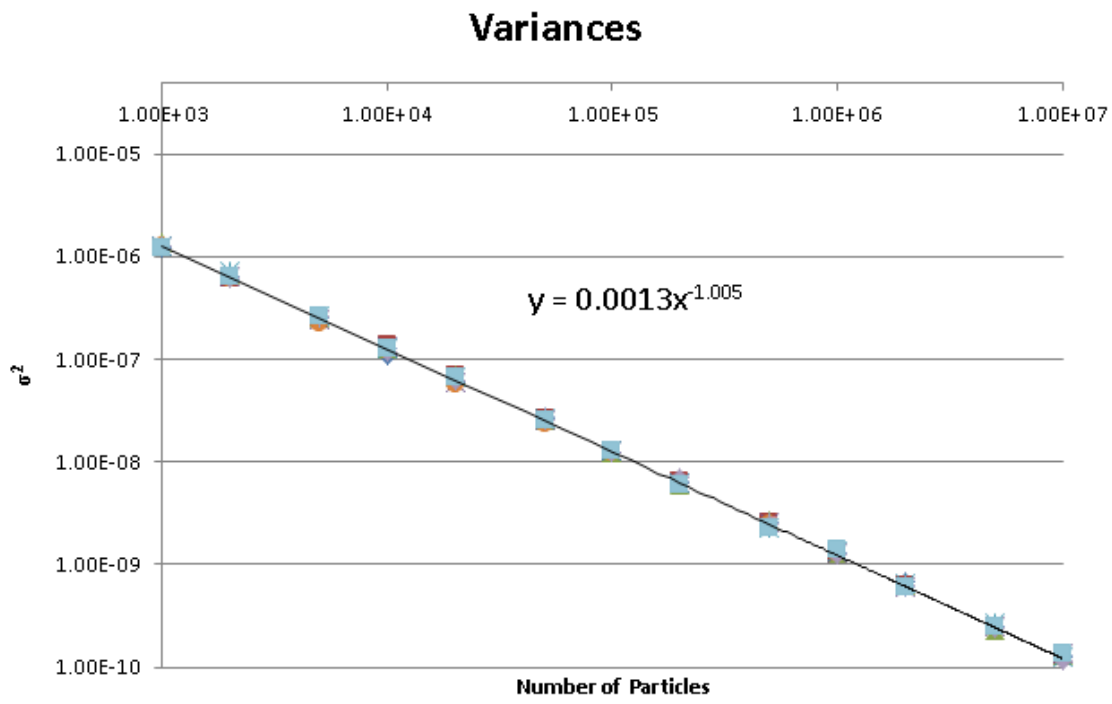

Figure C.5: C5G7 Rodded A variances vs. number of particles per cycle.

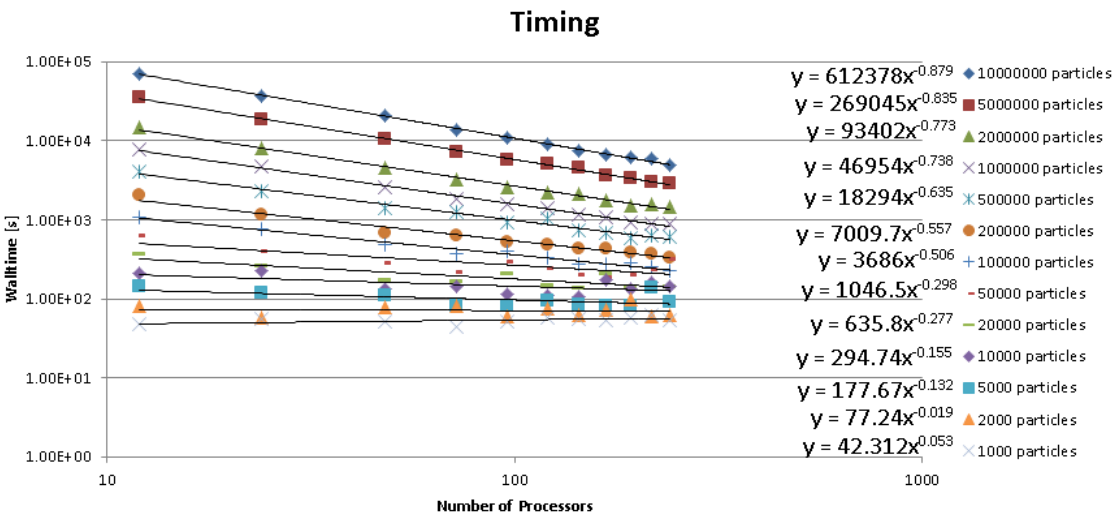

Figure C.6: C5G7 Rodded A run time vs. number of processors.

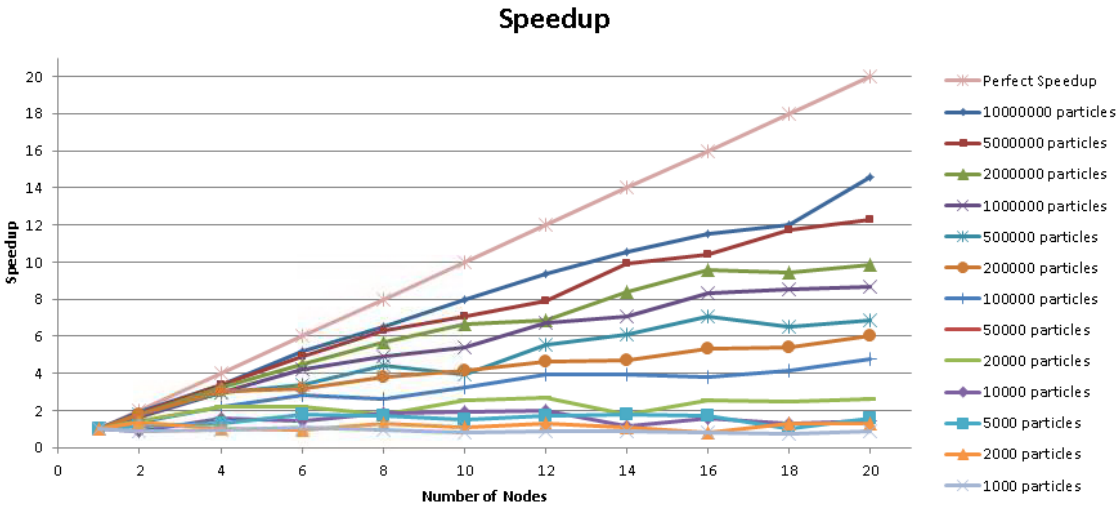

Figure C.7: C5G7 Rodded A speedup vs. number of nodes.

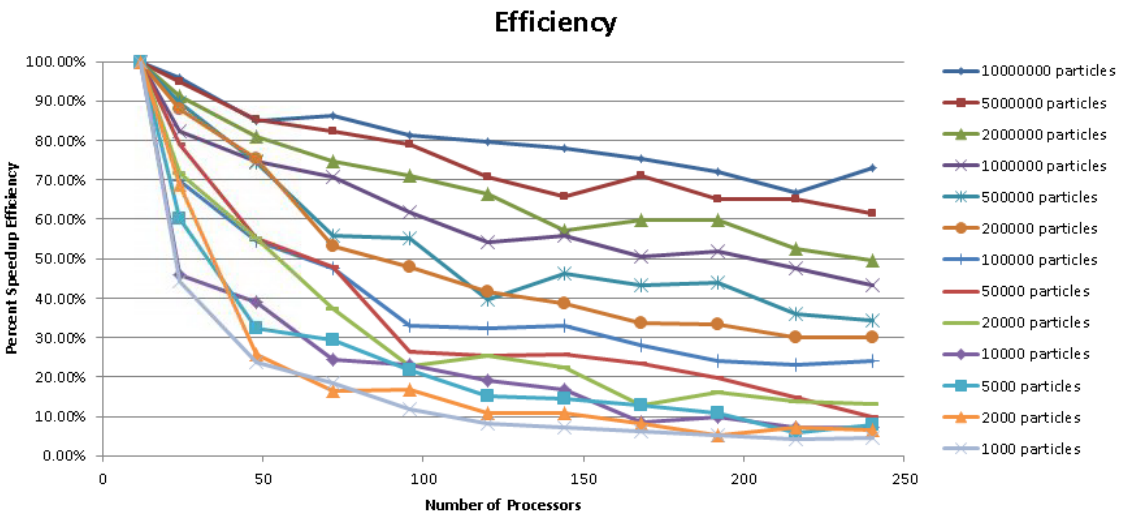

Figure C.8: C5G7 Rodded A efficiency vs. number of processors.

C5G7 Rodded B Graphs

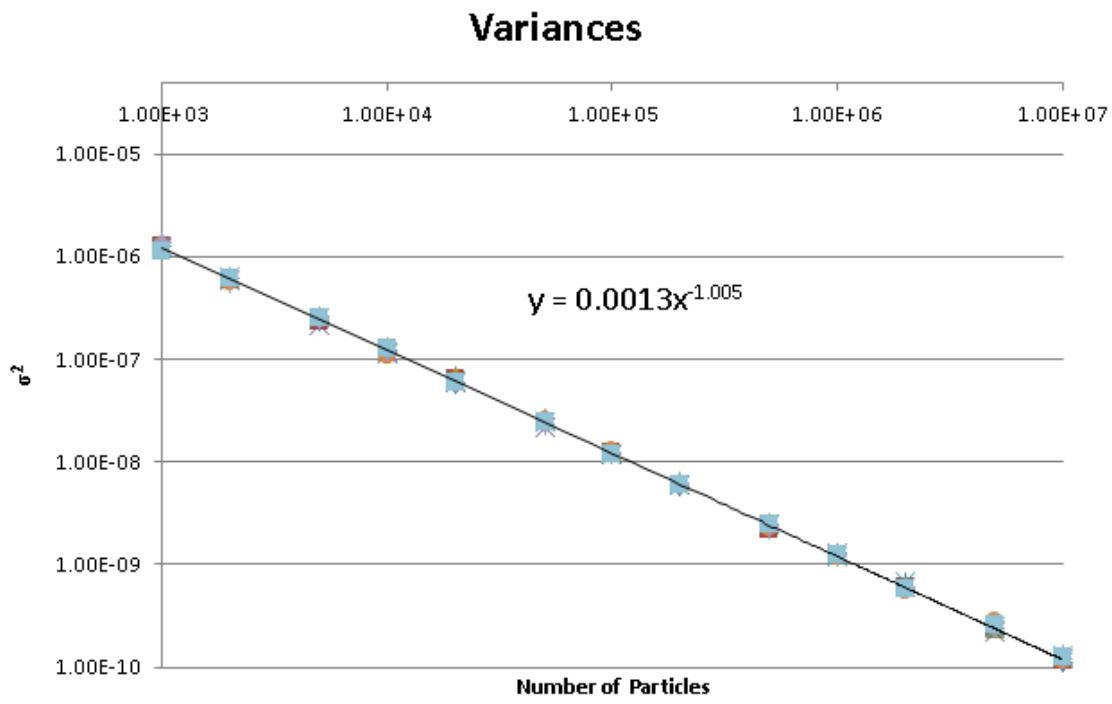

Figure C.9: C5G7 Rodded B variances vs. number of particles per cycle.

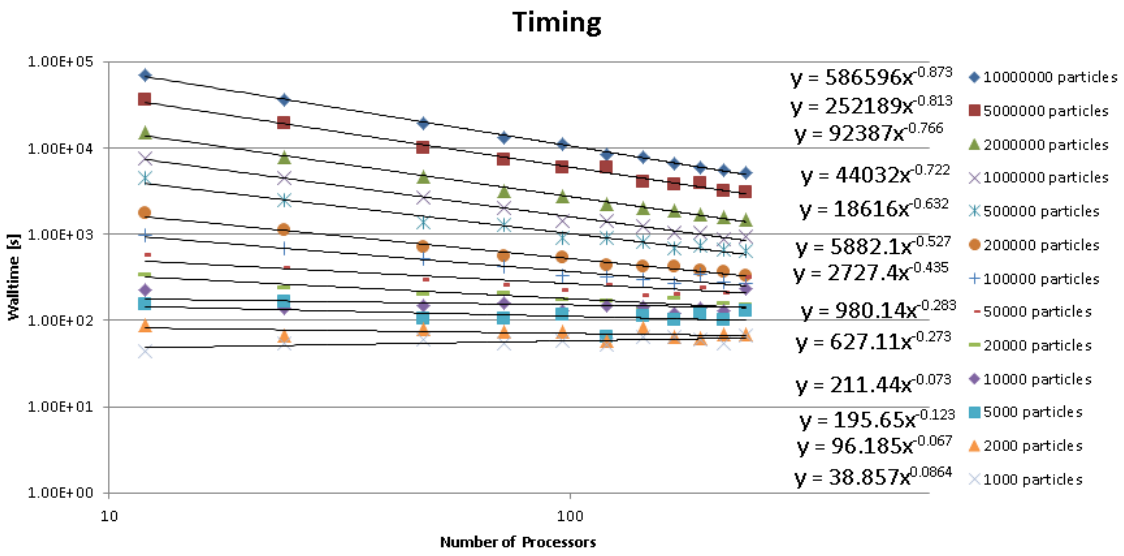

Figure C.10: C5G7 Rodded B run time vs. number of processors.

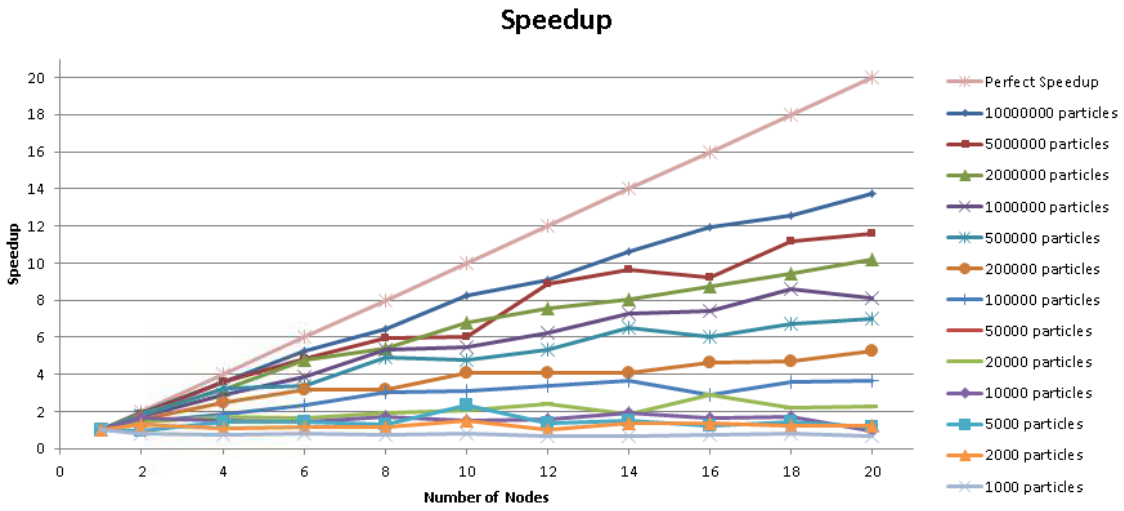

Figure C.11: C5G7 Rodded B speedup vs. number of nodes.

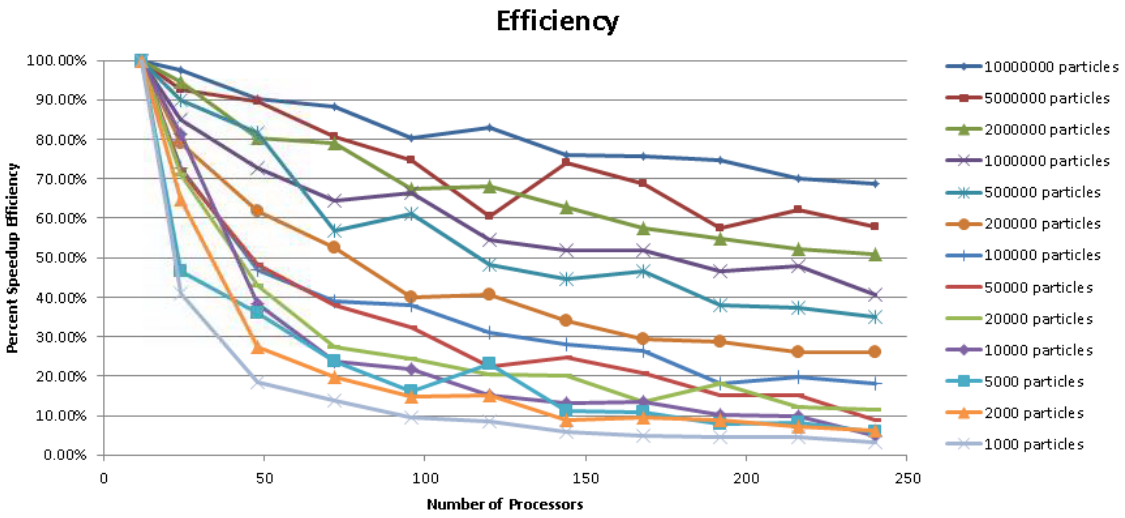

Figure C.12: C5G7 Rodded B efficiency vs. number of processors.

CASL 1a Graphs

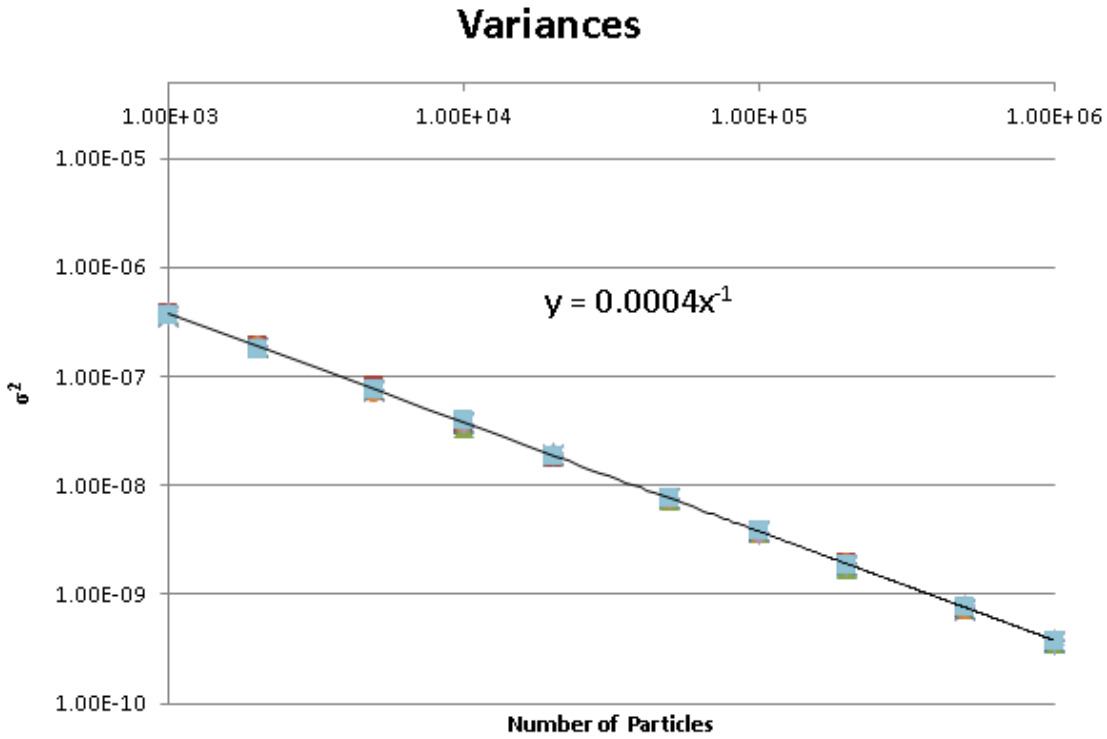

Figure C.13: CASL 1a variances vs. number of particles per cycle.

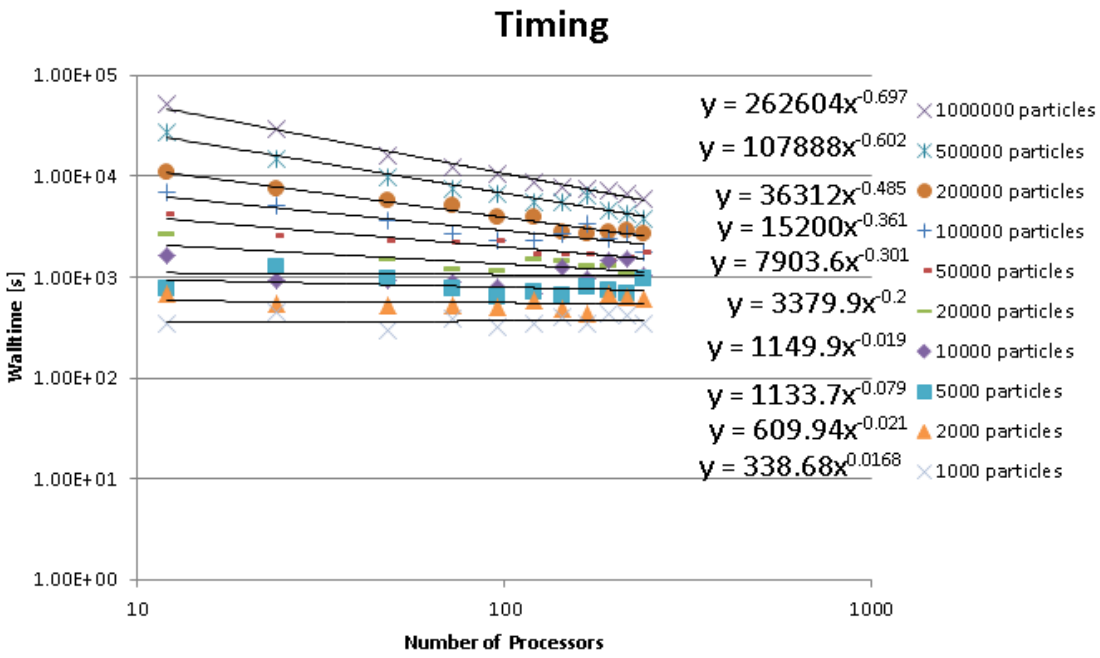

Figure C.14: CASL 1a run time vs. number of processors.

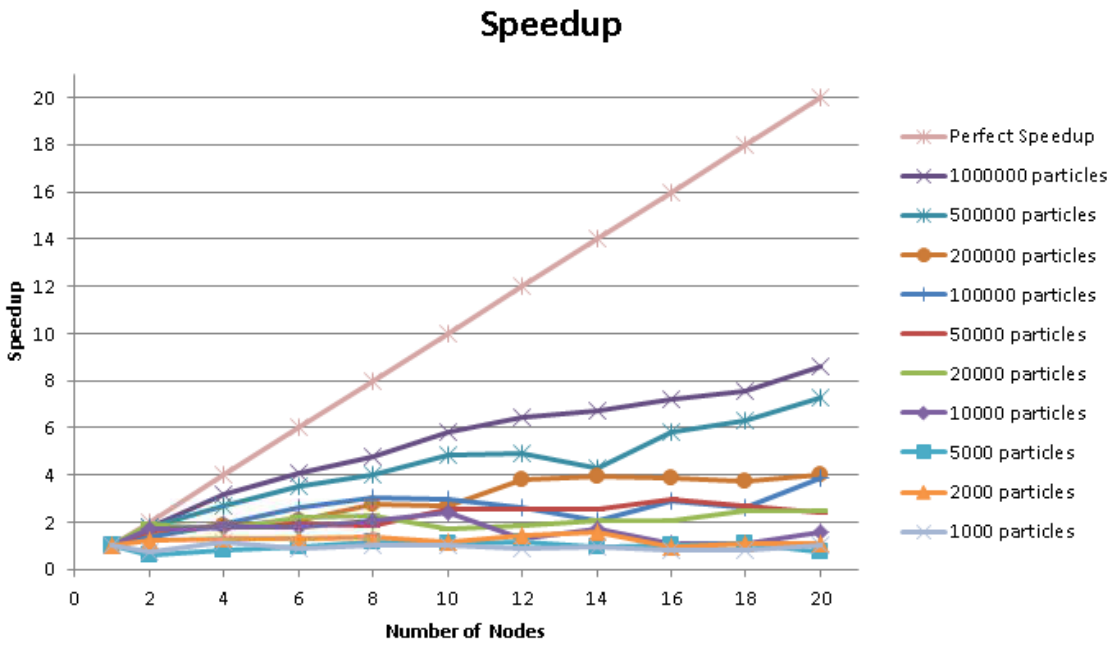

Figure C.15: CASL 1a speedup vs. number of nodes.

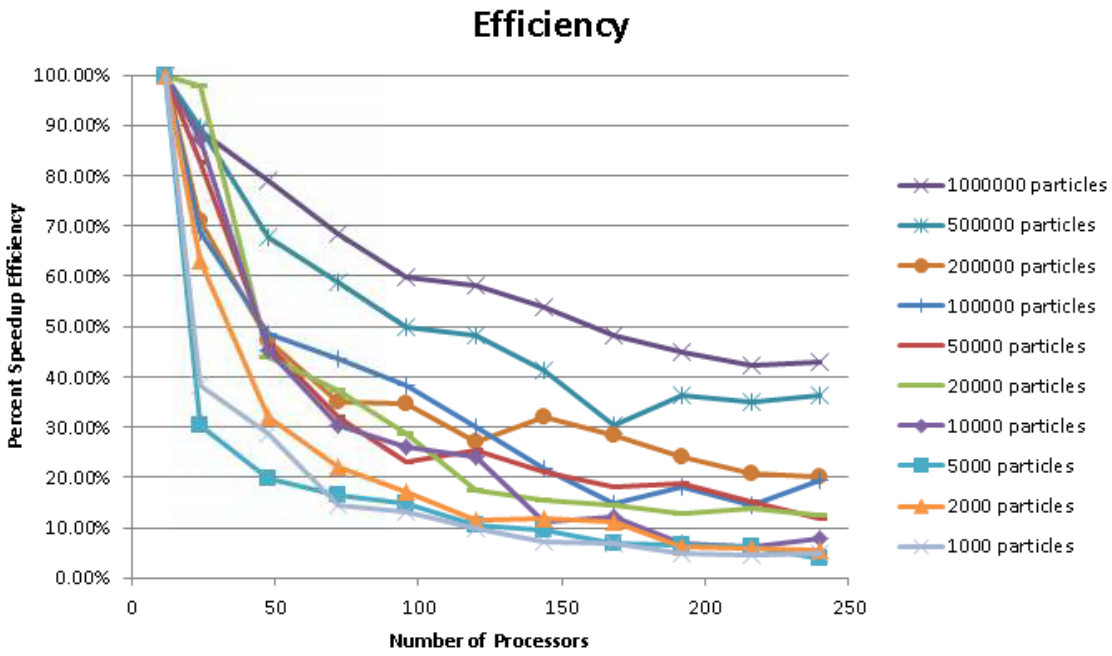

Figure C.16: CASL 1a efficiency vs. number of processors.

CASL 1b Graphs

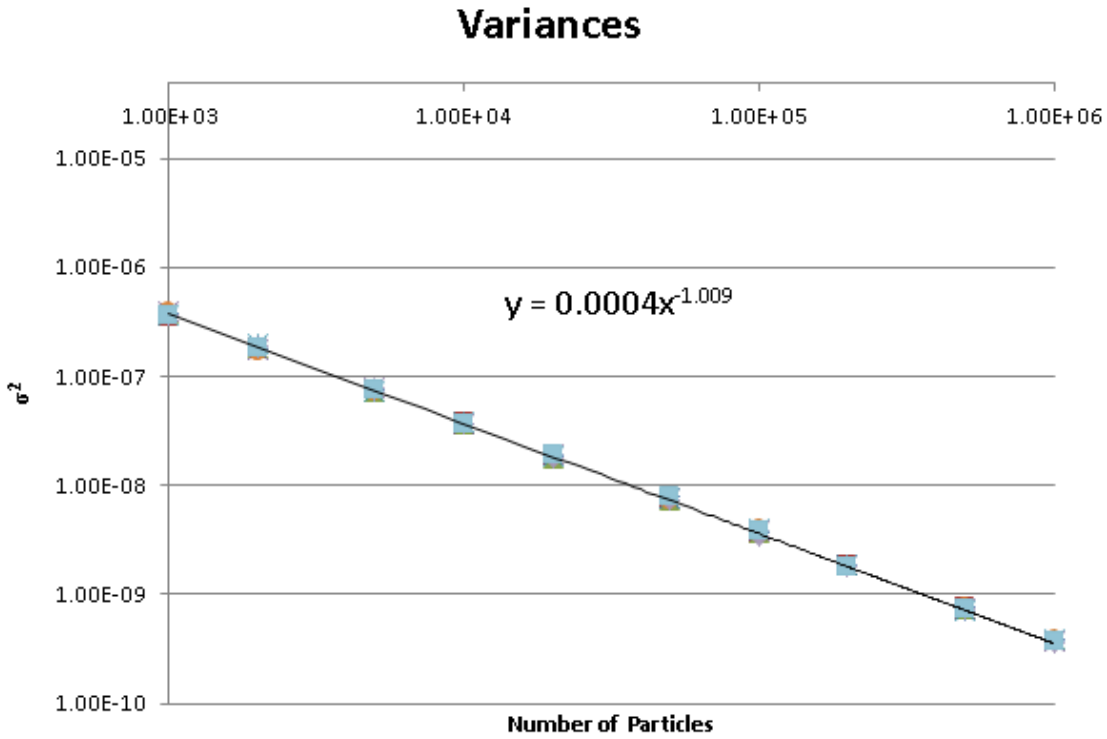

Figure C.17: CASL 1b variances vs. number of particles per cycle.

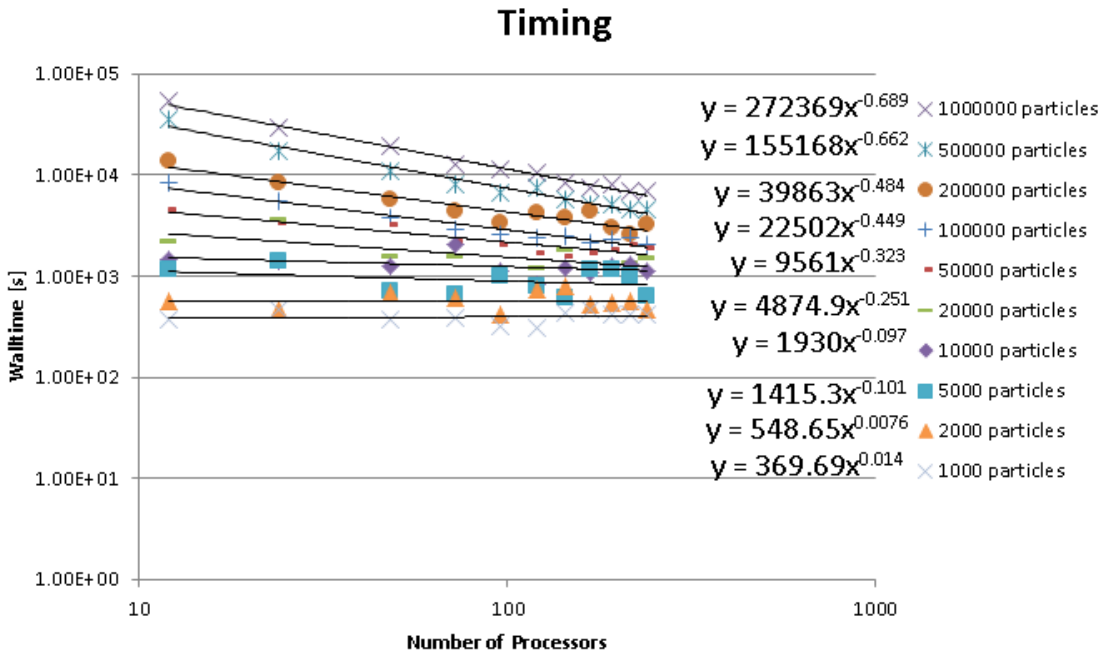

Figure C.18: CASL 1b run time vs. number of processors.

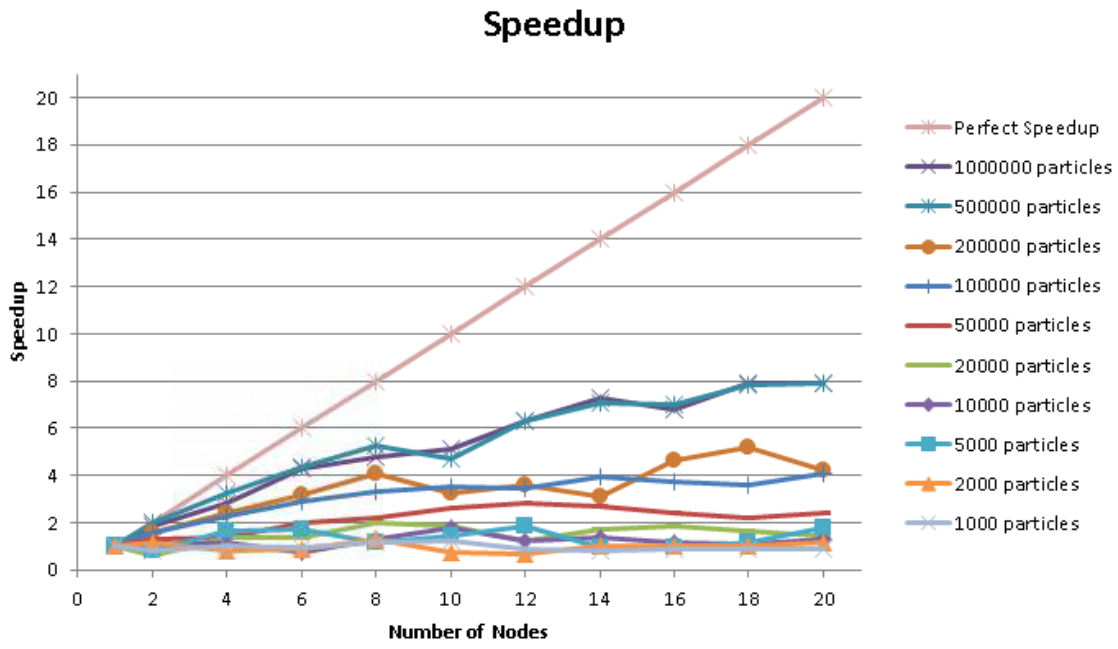

Figure C.19: CASL 1b speedup vs. number of nodes.

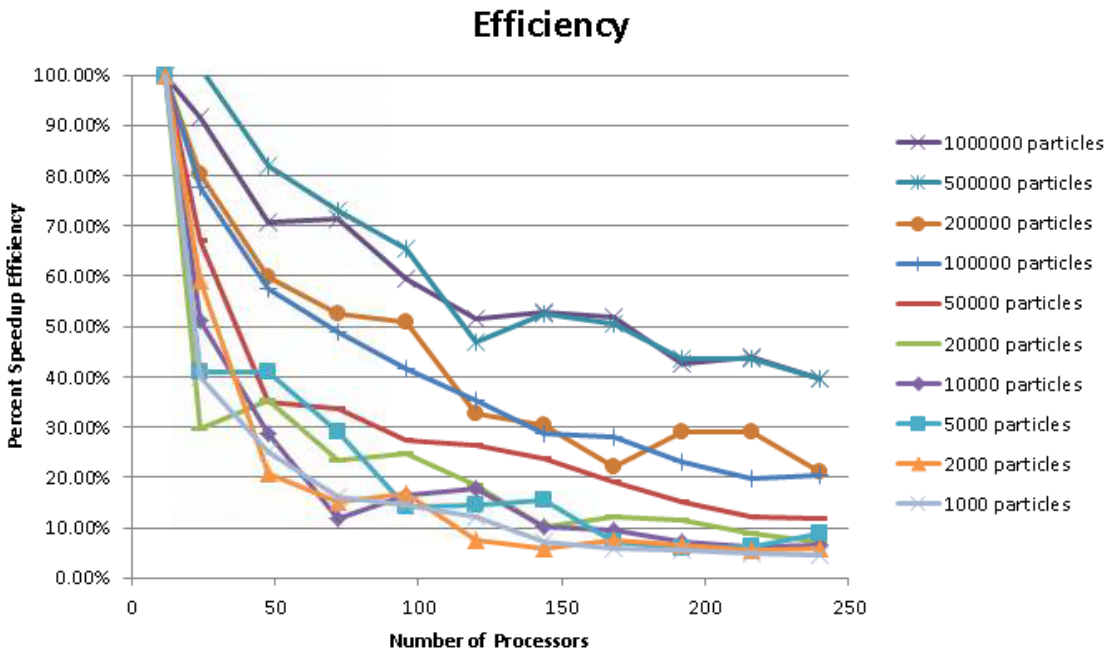

Figure C.20: CASL 1b efficiency vs. number of processors.

CASL 1c Graphs

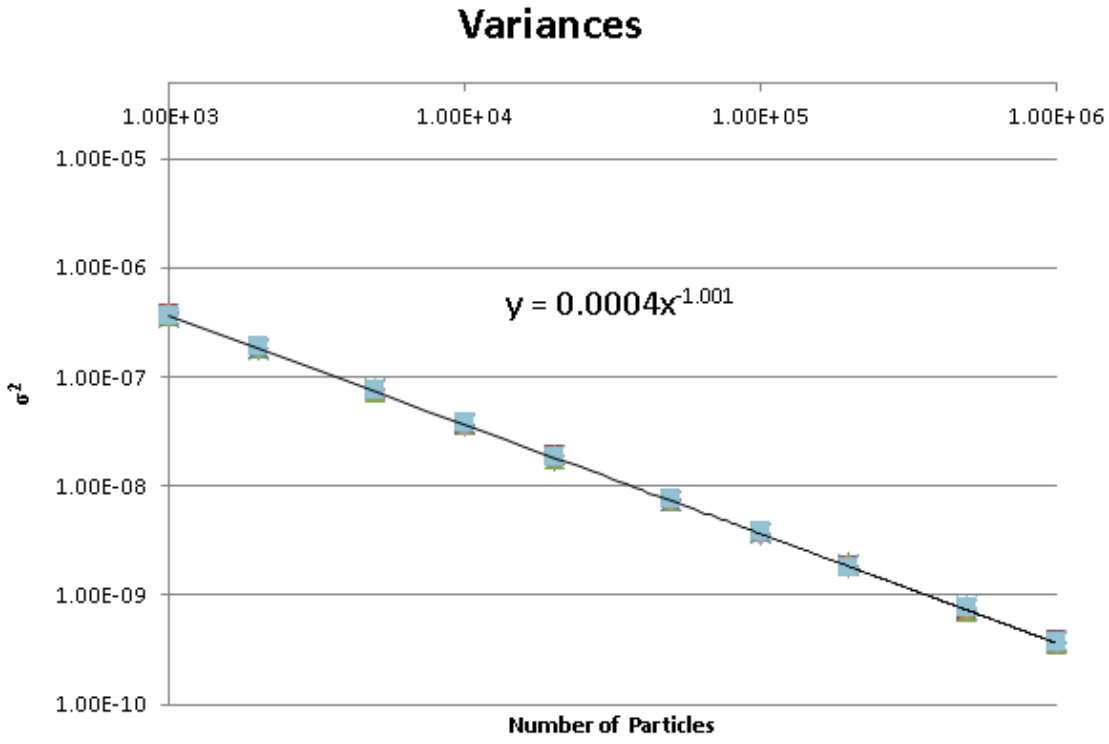

Figure C.21: CASL 1c variances vs. number of particles per cycle.

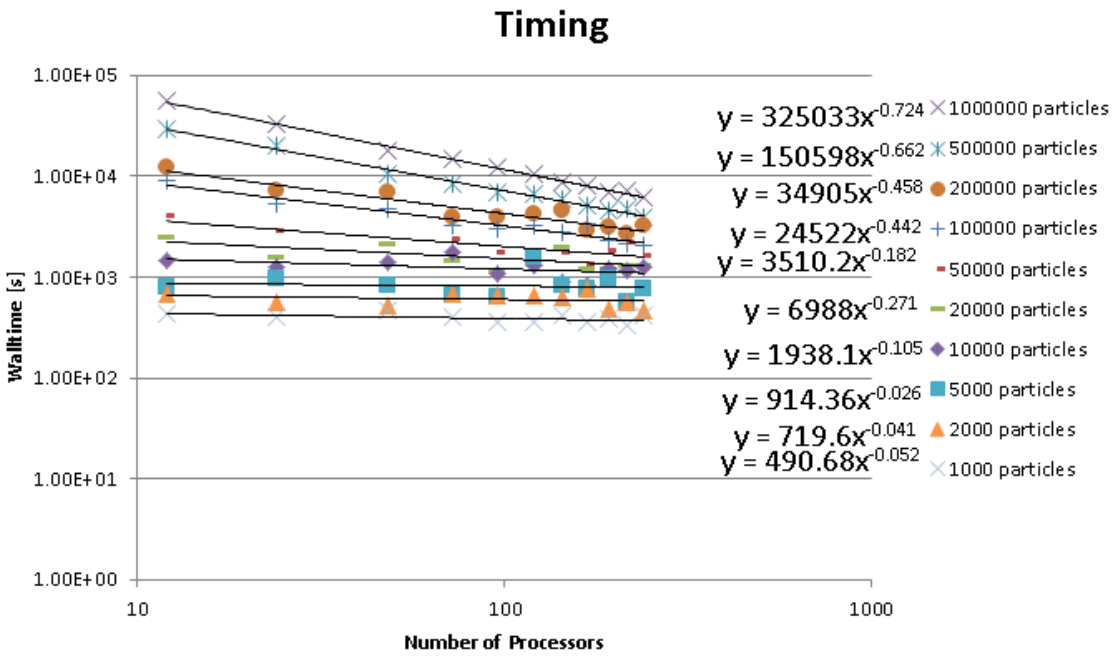

Figure C.22: CASL 1c run time vs. number of processors.

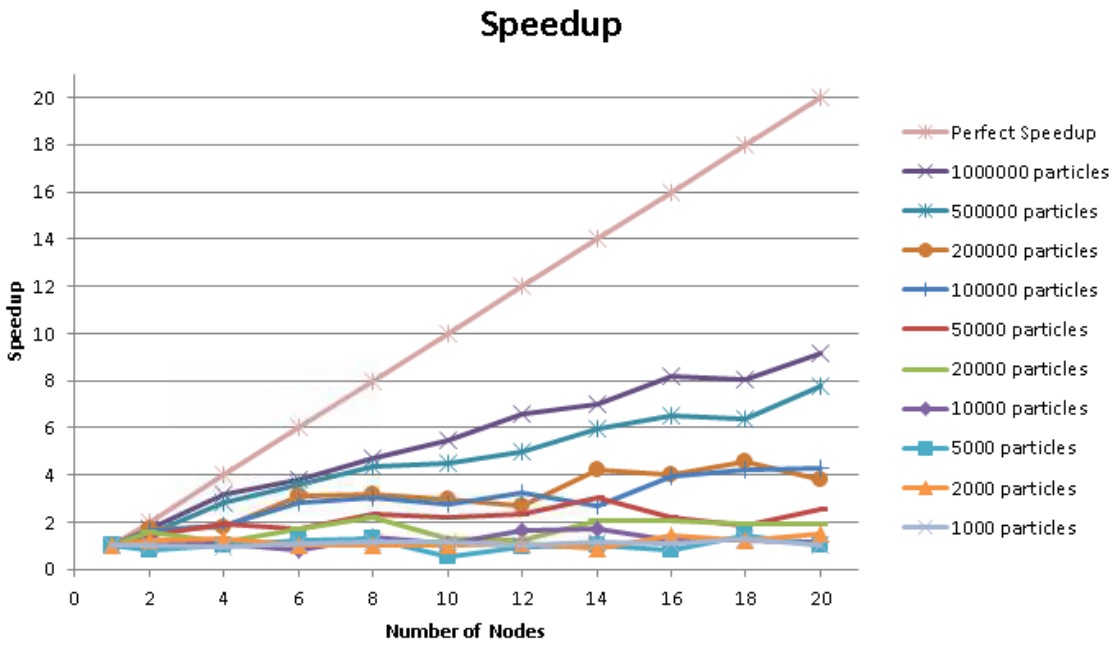

Figure C.23: CASL 1c speedup vs. number of nodes.

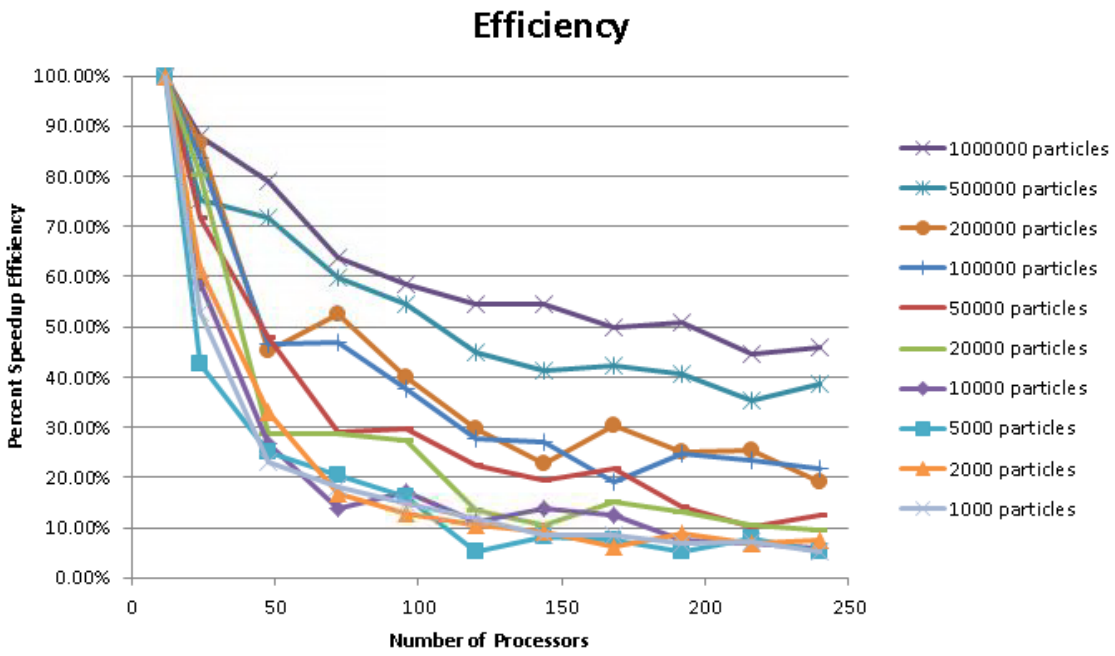

Figure C.24: CASL 1c efficiency vs. number of processors.

CASL 1d Graphs

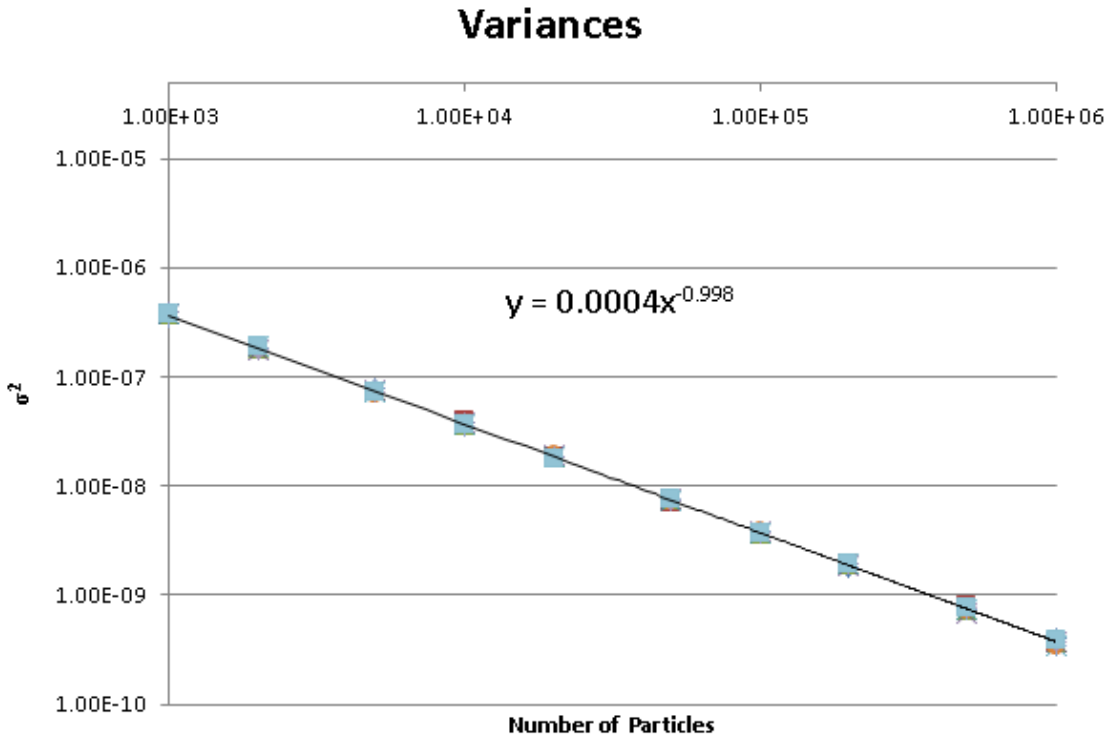

Figure C.25: CASL 1d variances vs. number of particles per cycle.

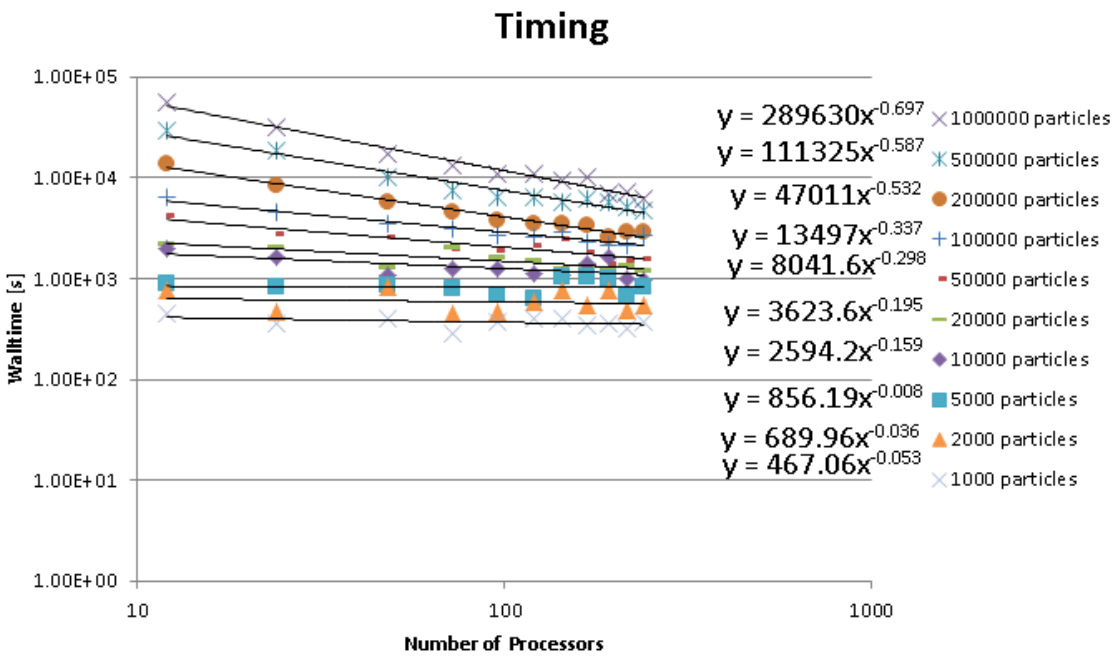

Figure C.26: CASL 1d run time vs. number of processors.

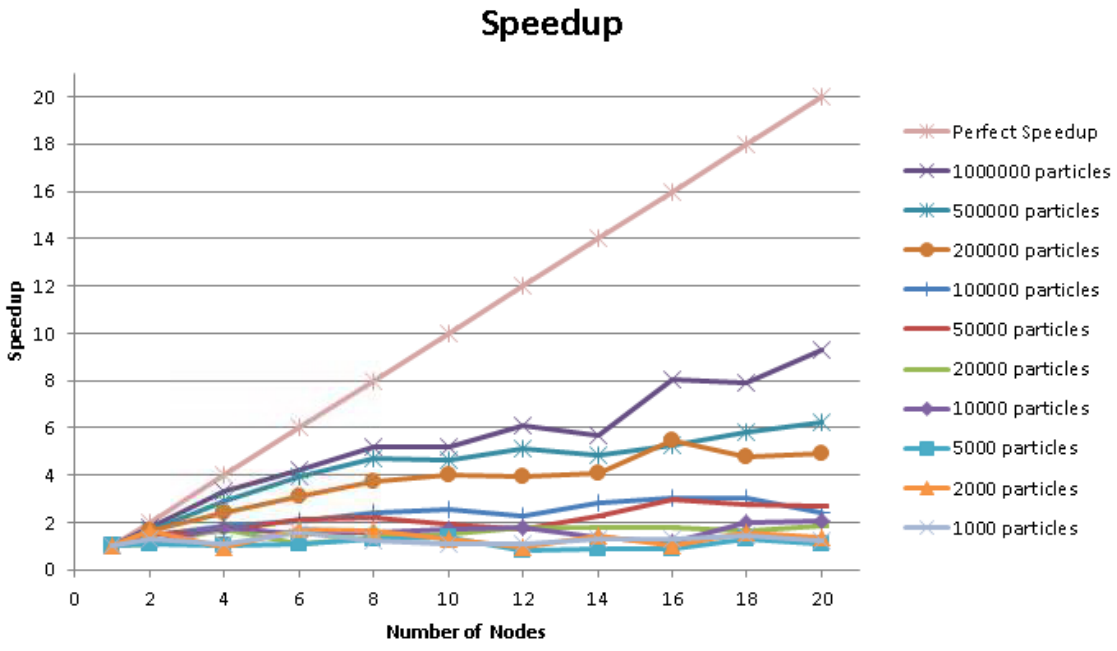

Figure C.27: CASL 1d speedup vs. number of nodes.

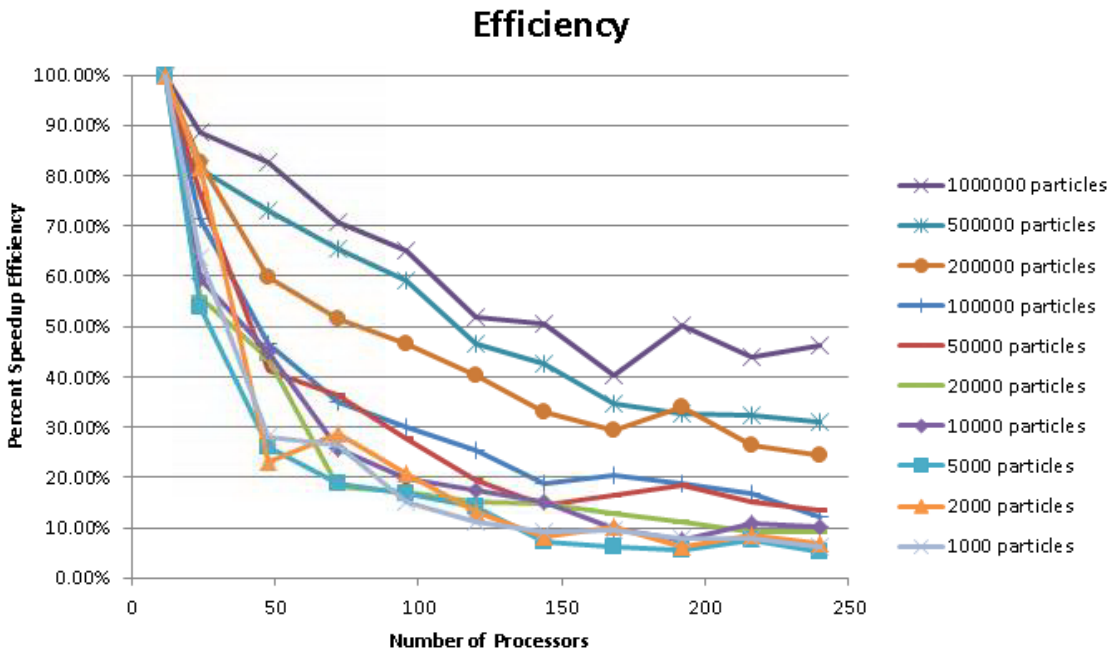

Figure C.28: CASL 1d efficiency vs. number of processors.

CASL 2a Graphs

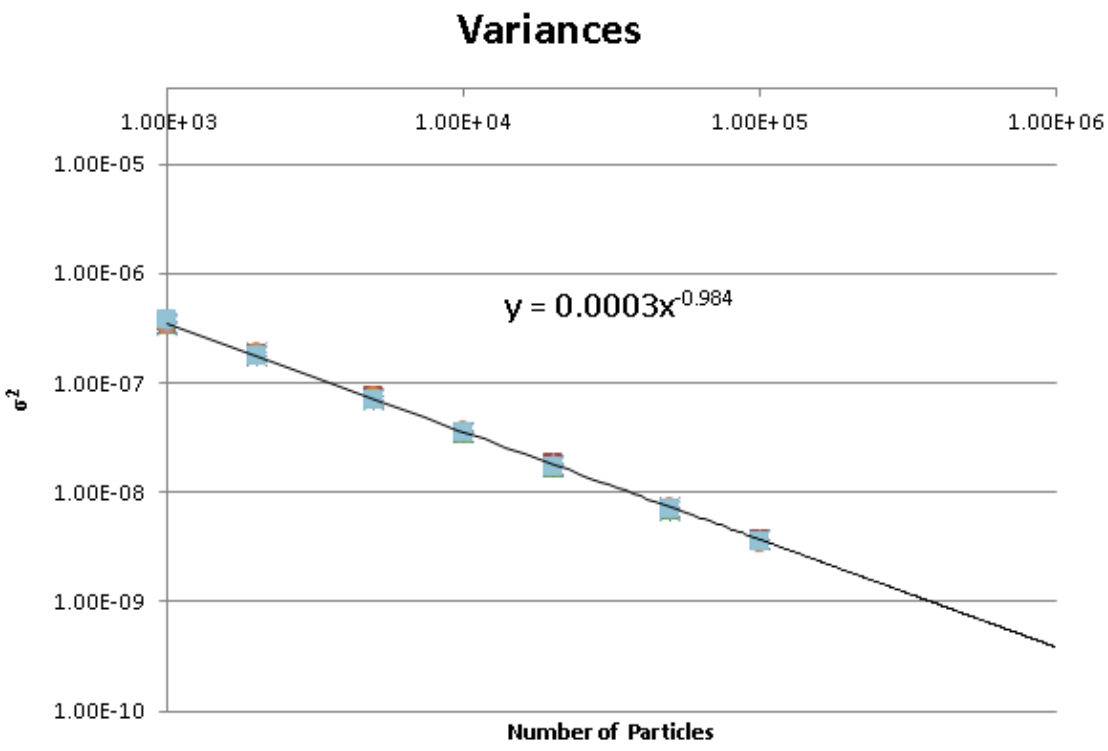

Figure C.29: CASL 2a variances vs. number of particles per cycle.

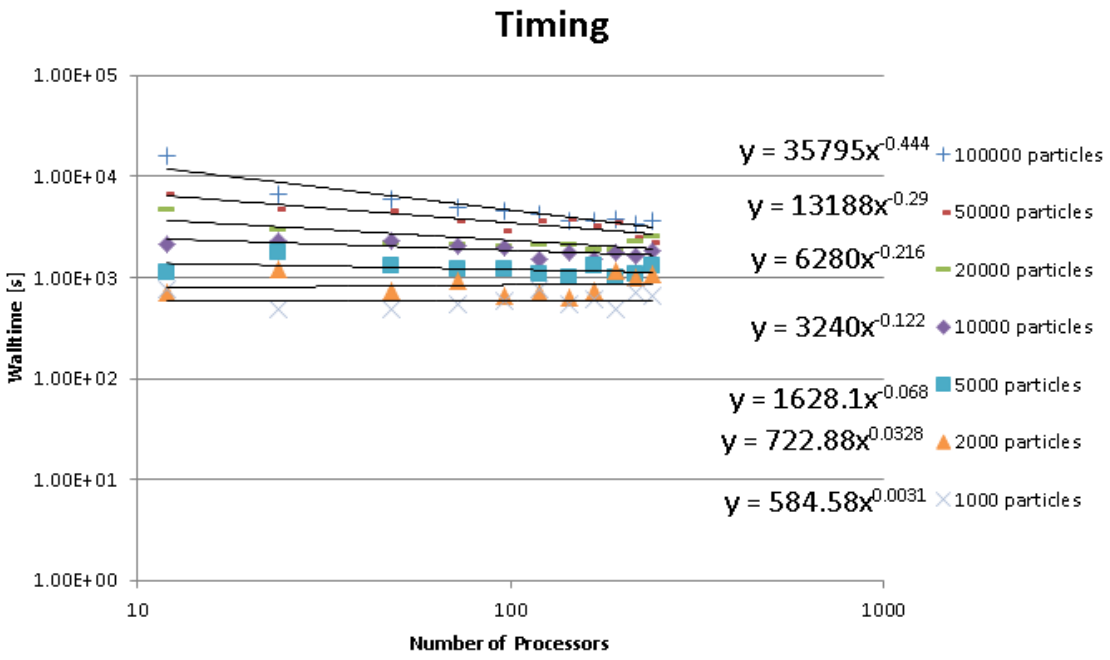

Figure C.30: CASL 2a run time vs. number of processors.

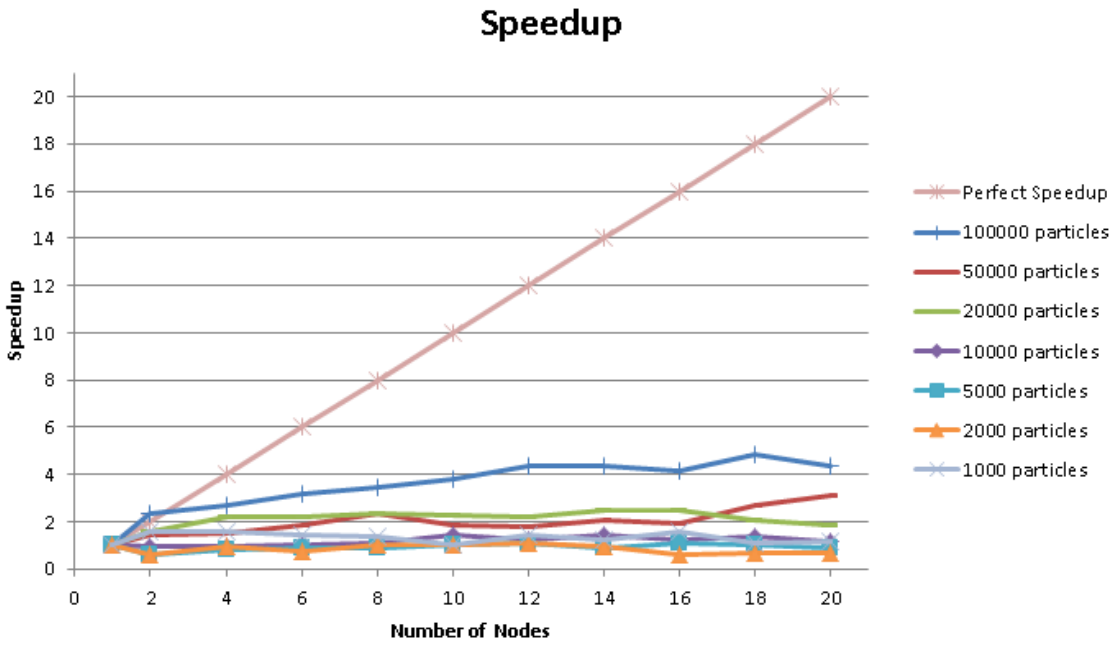

Figure C.31: CASL 2a speedup vs. number of nodes.

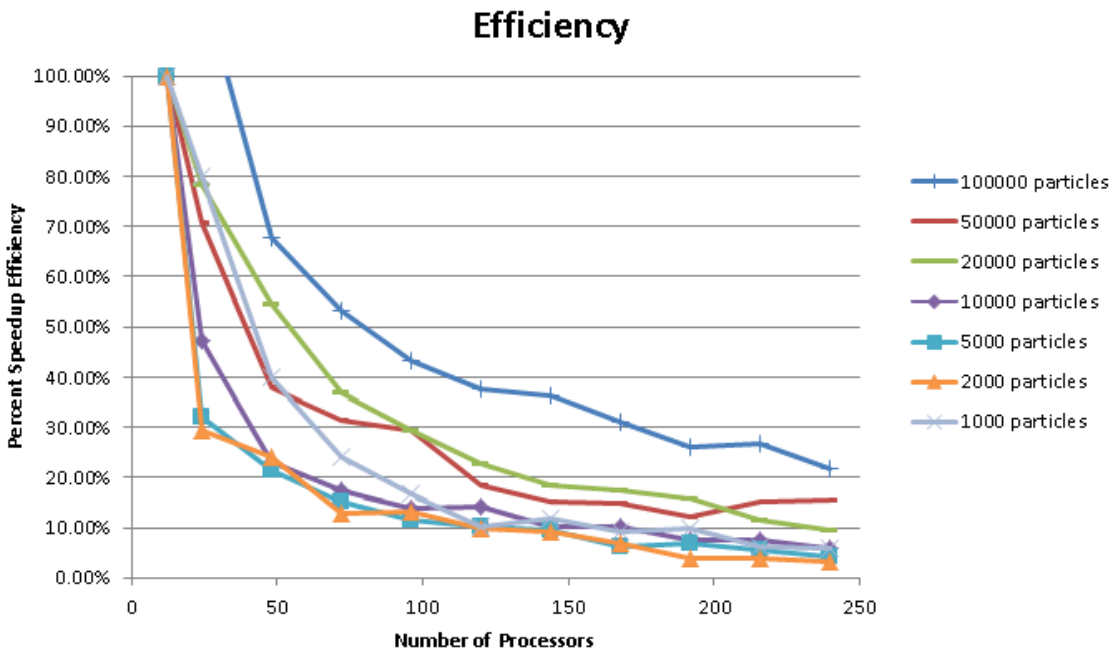

Figure C.32: CASL 2a efficiency vs. number of processors.

Vita

Nicholas Sly was born on December 11, 1987 in Dallas, Texas. He attended Gainesville High School in Gainesville, Florida and graduated in June 2006. He went on to attend the University of Florida (UF) studying nuclear engineering. While at UF, he worked at the University of Florida Training Reactor (UFTR) assisting with radiological surveys, upkeep up of the Neutron Activation Analysis Laboratory, and assistance in the upkeep and repair of the reactor. Nicholas graduated in August of 2010. He then went directly to and is currently attending The University of Tennessee at Knoxville to study for his graduate degree in Nuclear Engineering. There he works as a research assistant on collaborative projects between Oak Ridge National Laboratory and The University of Tennessee. He will obtain his Masters Degree in the Spring of 2013 and continue working towards his Ph.D. Nicholas endeavors to proceed from there into a successful career of research.