PARALLEL MONTE CARLO PARTICLE TRANSPORT AND THE QUALITY OF RANDOM NUMBER GENERATORS: HOW GOOD IS GOOD ENOUGH?

Richard Procassini and Bret Beck
Lawrence Livermore National Laboratory
Mail Stop L-95, P. O. Box 808
Livermore, CA 94551
spike@llnl.gov; bbeck@llnl.gov

ABSTRACT

It might be assumed that use of a "high-quality" random number generator (RNG), producing a sequence of "pseudo random" numbers with a "long" repetition period, is crucial for producing unbiased results in Monte Carlo particle transport simulations. While several theoretical and empirical tests have been devised to check the quality (randomness and period) of an RNG, for many applications it is not clear what level of RNG quality is required to produce unbiased results.

This paper explores the issue of RNG quality in the context of parallel, Monte Carlo transport simulations in order to determine how "good" is "good enough". This study employs the MERCURY Monte Carlo code, which incorporates the CNPRNG library for the generation of pseudo-random numbers via linear congruential generator (LCG) algorithms. The paper outlines the usage of random numbers during parallel MERCURY simulations, and then describes the source and criticality transport simulations which comprise the empirical basis of this study. A series of calculations for each test problem in which the quality of the RNG (period of the LCG) is varied provides the empirical basis for determining the minimum repetition period which may be employed without producing a bias in the mean integrated results.

Key Words: Monte Carlo, particle transport, parallel computation, random numbers

1   INTRODUCTION

Recent advances in the speed of parallel computers, as well as the development of parallel algorithms designed to harness the power of these systems, have prompted discussion regarding the "quality" of the pseudo-random numbers required in order to produce accurate, unbiased results in Monte Carlo transport simulations. Several theoretical and empirical tests have been devised which are designed to test the quality of random number generator (RNG) algorithms [1]. However, such tests are not sufficient to determine the level of RNG quality that is required to produce unbiased results. In general, each application must diagnose the results of simulations in order to determine if the quality of the RNG being employed is sufficient.

One might assume that use of an RNG which produces a stream of pseudo-random numbers with a "long" repetition period is crucial in order to produce unbiased results. For some Monte Carlo applications, such as numerical integration, it is extremely important to avoid the reuse of pseudo-random numbers. These types of applications employ RNGs to randomly sample a small number of distributions, usually one. As such, it is important to avoid correlations in the sampled quantity.

In contrast, Monte Carlo transport applications use RNGs to randomly sample from several distributions, which may or may not be correlated: the spatial, energy or angular distributions of

sources, the distance to collision, the isotope with which the particle interacts, the resulting reaction, the energy and angle of any secondary particles, etc. While it is desirable to avoid reuse of pseudo-random numbers in Monte Carlo transport simulations, it is not clear, a priori, whether such reuse will produce a bias in the results. For example, a given pseudo-random number may be used to sample the energy of one particle which is created in an external source, and the next use of that pseudo-random number may occur many time steps and particles later during a collision sampling event.

The non-linear nature of particle transport, in concert with the multiple forms of distribution sampling encountered in Monte Carlo particle transport, *may* be beneficial in that the RNG period required for unbiased simulations *may* be shorter that that required for single sample applications such as numerical integration. In this paper, we explore the issue of RNG quality in the context of parallel, Monte Carlo transport simulations in order to determine how "good" is "good enough". This study employs the MERCURY Monte Carlo code [2], which incorporates the CNPRNG library [3] for the generation of pseudo-random numbers. A set of four source and criticality transport problems form the basis for comparing the quality of LCG RNGs with varying period.

The paper is organized as follows. The parallel programming model implemented in MERCURY is presented in Section 2, and the use of pseudo-random numbers in parallel MERCURY calculations is described in Section 3. Section 4 describes the four test problems used for this study, while Section 5 presents the methodology used for this study along with results for a series calculations of each problem in which the period of the RNG was varied. Finally, the conclusions of this study and recommendations for future work are presented in Section 6.

2   THE MERCURY PARALLEL PROGRAMMING MODEL

MERCURY is a modern, Monte Carlo transport code which has been developed over the last six years at the Lawrence Livermore National Laboratory (LLNL). The intent is for MERCURY to replace the legacy codes TART and COG as the next-generation radiation transport tool at LLNL. The design and development of MERCURY has been driven by the requirement that the code be able to operate on the wide variety of parallel computing platforms that are provided by the Advanced Simulation and Computing (ASC) program. This has led to the adoption of a three pronged approach to parallelism within MERCURY.

The first form of parallelism supported in MERCURY is *Domain Decomposition*, in which the problem geometry is spatially partitioned across multiple processors. This form of *spatial* parallelism allows MERCURY to transport particles through geometries which contain a large number of cells, such as calculations requiring high-resolution, multidimensional meshes. Figure 1 shows the 4-way spatial partitioning of a two-dimensional, block-unstructured mesh. The domains are color coded according to processor number. As particles track to a facet which lies on the boundary of a domain, it must be sent to the adjacent domain before it can continued its trajectory. This transfer of particles between adjacent domains is accomplished via point-to-point communication of particle buffers using message passing techniques on distributed-memory parallel computers, as illustrated by the red arrows in Figure 1.
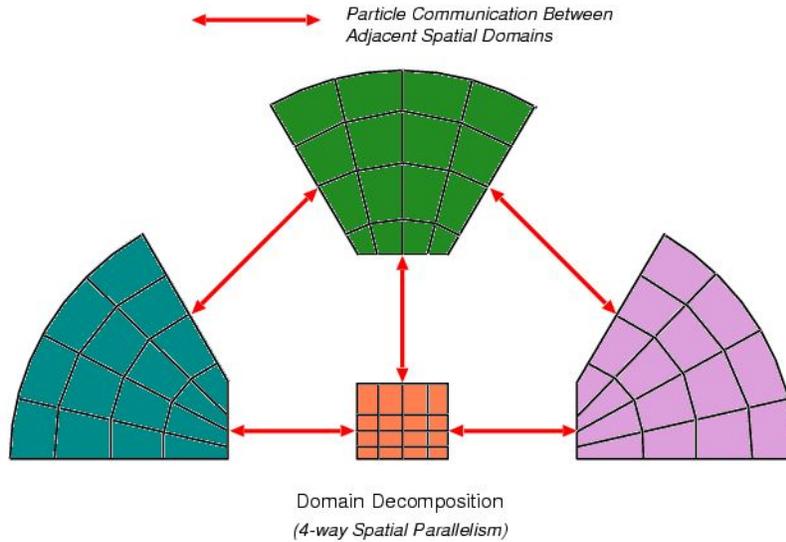
Figure 1. Spatial parallelism in MERCURY is achieved via domain decomposition (spatial partitioning) of the problem geometry. Particles must be transferred to adjacent domains when they reach a domain boundary. The communication of particle buffers between adjacent spatial domains is indicated via the red arrows.

The second form of parallelism employed in MERCURY is *Domain Replication.* In this method, particles are distributed across multiple copies of the problem geometry, each of which is assigned to a specific processor. This form of *particle* parallelism allows MERCURY to efficiently transport large numbers of particles. Figure 2 shows the 2-way replication of the same two-dimensional, block-unstructured mesh discussed above. Each copy of the geometry (mesh) is color coded according to processor number. Once the calculation is complete on each copy of the geometry (domain), the partial results from each copy of the domain must be summed in order to produce the complete result. This transfer of data is accomplished via collective communication between the multiple copies of the domain using message passing techniques on distributed-memory computers, as indicated by the blue, curved arrows in Figure 2.

The final form of parallelism supported by MERCURY is *Task Decomposition.* This method decomposes the main particle loop by assigning individual particles, or *tasks*, to threads on a shared-memory parallel computer. This represents another form of particle parallelism.

3   USE OF RANDOM NUMBERS IN PARALLEL MERCURY CALCULATIONS

The complex, multifaceted parallel programming model that has been developed for MERCURY allows for a wide variety of operating modes, from serial calculations to parallel calculations using any of the three forms of parallelism individually or in combination. A major design requirement is that the code be capable of reproducing the same result, regardless of the chosen mode of operation.

In an attempt to guarantee such reproducibility, a hierarchy of random number seeds has been implemented in MERCURY. This approach makes use of the spawning feature of the CNPRNG library, which has been developed in the Computational Nuclear Physics Group at LLNL. At the top of this hierarchy is the *Simulation Seed*, which represents the primordial seed

from which all other seeds are spawned. This 64-bit seed is initialized via the bitwise combination of two 32-bit, user specified random number seed fragments.
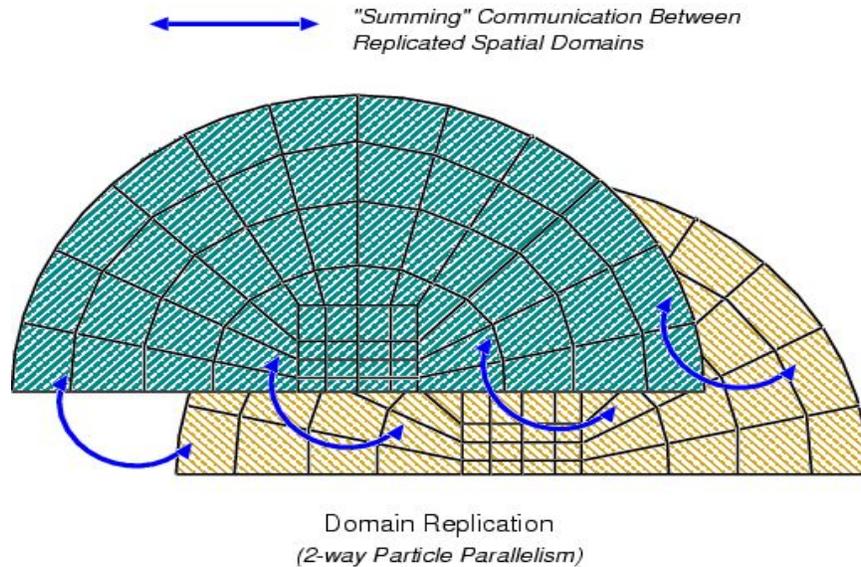
Figure 2. Particle parallelism in MERCURY is achieved via domain replication (multiple copies) of the problem geometry. The particle workload is distributed across the copies of the domain. The summing communication of partial results is indicated by the blue, curved arrows.

The Simulation Seed is then used to spawn two seeds at the next level in the hierarchy: the *Source Seed* and the *Spatial Domain Seed(s)*. The former is used to spawn the seeds for all particles that are generated in external sources, while the latter is (are) defined for each spatial domain in the problem, and is (are) used to *(a)* spawn the seeds of all particles generated in cell-based, physics sources (such as a fusion source) and *(b)* produce pseudo-random numbers which are required for a specific cell and/or domain.

At the lowest level of the hierarchy is the *Particle Seed*. Once spawned from the Source Seed or a Spatial Domain Seed, the Particle Seed controls the behavior of the particle. The Particle Seed is an inherent attribute of the particle, along with other attributes such as spatial coordinates, velocity components, kinetic energy, weight, time to census, cell, facet, etc. As a result, the particle carries its random number seed along with it as it tracks through the problem geometry and is communicated between adjacent spatial domains. This approach to parallelism and random numbers is different from others in which only the processors are assigned uniques random number seeds.

Ideally, the random number seed should be a small fraction of the overall storage cost of each particle. In MERCURY, the storage required for all of the particle attributes except the random number seed amounts to 120 bytes: ten 64-bit floating point (80 bytes), one 64-bit integer (8 bytes) and eight 32-bit integers (32 bytes). The seed of a pseudo-random number stream in the CNPRNG library is defined by a single 64-bit integer (8 bytes). This increases the cost of storing each particle by only 6.67%. The compact nature of this seed is the reason that this RNG library was chosen over others, such as the SPRNG library [4], which can require more than 100 bytes to

define the seed of each random number stream. Compact storage of the random number seed is crucial for applications using per particle seeds.

This hierarchical approach to the definition of random number seeds has allowed MER-CURY to achieve reproducibility of parallel calculations *in most cases*. The only case where such calculations are not reproducible is when domain decomposition involving more than two spatial domains is used. In this case, there exists the possibility of a race condition between processors on many types of parallel computers that could change the order in which results are accumulated.

Consider the case of three spatial domains assigned to three processors. Assume that each domain is adjacent to the remaining two domains. In this case Processor A will receive particle buffers from Processors B and C. The point-to-point data transfer between processors is coded using asynchronous, non-blocking communications routines. This method involves posting receive operations prior to send operations, and periodically checking to see if any messages have been received from an adjacent processor.

During one run, it is possible that a message from Processor B will arrive prior to a message from Processor C, and Processor A will check for messages from its neighbors in the interim. A second run may reverse the order of arrival of the messages from Processors B and C. While the particles carry their random number seeds with them as they are transferred between processors, and as such experience the same random sampling of distributions, the *results* or tallies of the calculation will be accumulated in a different order. This can lead to a different set of results for the two runs since finite-precision, floating point mathematics (the type used by all computer systems) is *not* associative.

The possible solutions for this problem require either *(a)* temporarily accumulating results as 64-bit integers or *(b)* conversion of the communications to use a synchronous, blocking methodology. The former solution requires significant recoding to achieve, while the latter could result in a potential deadlock that would stall all future computations. Therefore, neither option is very appealing. The good news is that while race-condition induced differences are possible, they appear only infrequently, and the magnitude of the observed differences is within the statistical variability of the results as set by the number of particle histories.

4   DESCRIPTION OF THE SOURCE AND CRITICALITY TEST PROBLEMS

Four problems have been chosen for this study of the effect of RNG period (quality) on the results produced by Monte Carlo calculations. This set includes two shielding calculations and two criticality calculations. These test problems have been derived from real world applications that are of interest to many in the nuclear engineering and transport communities.

4.1 Problem 1: A Spherical, Neutron Shielding Experiment

The first source problem used for this study is a spherical representation of one configuration of a fusion-neutron shielding experiment that was performed at Oak Ridge National Laboratory (ORNL) in 1980 [5]. This problem incorporates only the shield materials from configuration 7 of the ORNL experiment: 35.56 cm of stainless steel, followed by two sets of alternating layers of borated polyethylene and stainless steel, each of which is 5.08 cm thick (see Figure 3). Monoenergetic 14.1 MeV neutrons are isotropically injected at the center of the sphere during a 2 μ sec flat top pulse. The problem is run out to a simulation time of 200 μ sec.
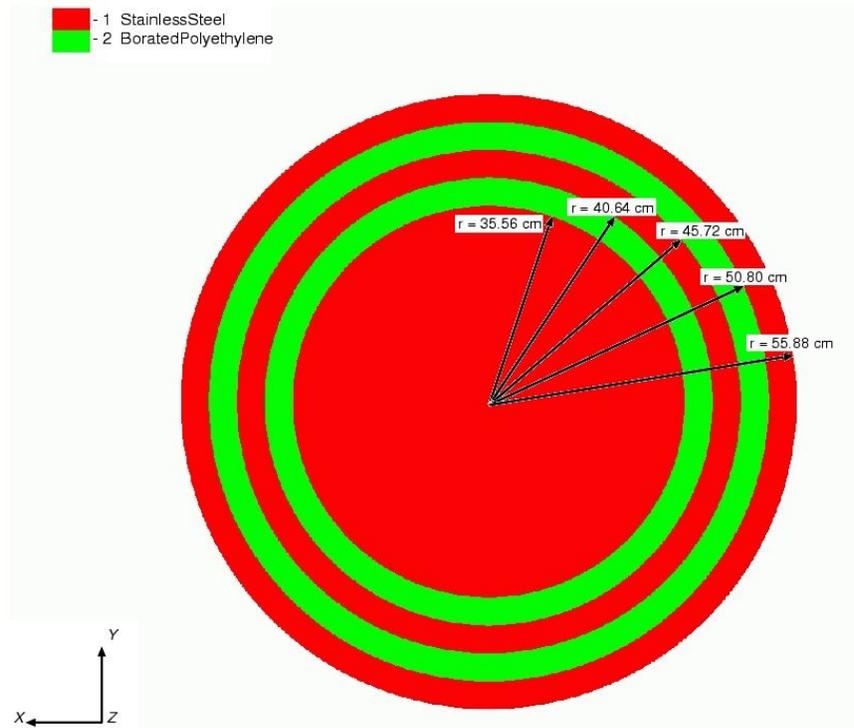
Figure 3. The geometry for Problem 1: A spherical, neutron shielding experiment.

For this test problem the study metrics include the following cumulative particles tallies: the number of particle collisions N_{coll} , as well as the number of particles leaked N_{leak} from the system and the number of particles absorbed N_{abs} , captured (terminal absorption) N_{cap} , and produced N_{prod} within the system. This problem was run in time-dependent transport mode.

4.2 Problem 2: A Fusion-Neutron Shielding Test Facility

The second source problem is a complete mock-up of the ORNL fusion-neutron shielding experiment discussed in the previous section. This problem includes all structural materials, such as the concrete floor, ceiling and walls, as well as the concrete shield box, iron pipe housing the beam line, stainless-steel thermal shield and the 30.48 cm thick stainless steel shield (ORNL configuration 3). The geometry for this test problem is shown in Figure 4. A parallel-ray disk source of monoenergetic 14.5 MeV neutrons are injected at the location of the tritiated titanium target during a 10 μ sec flat top pulse. The problem is run for for a total of 100 μ sec.

The study metrics for this test problem include the cumulative particles tallies used for Problem 1, as well as the counts of particles passing through two regions of the problem during the time interval 90 - 100 μ sec. These regions are *(a)* a 5 cm radius spherical "detector", filled with air, which is located 154.5 cm downstream of the source target and (b) a 5.08 cm thick planar thermal shield, made of stainless steel, which is located 204.5 cm downstream of the source target. This problem was also run in time-dependent transport mode.
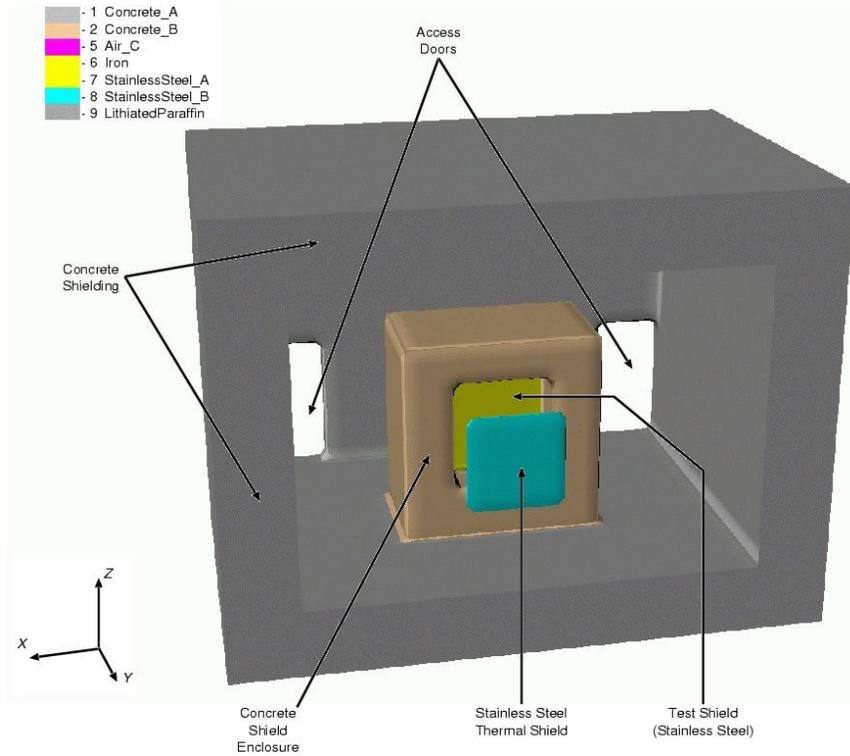
Figure 4. The geometry for Problem 2: a fusion-neutron shielding test facility.

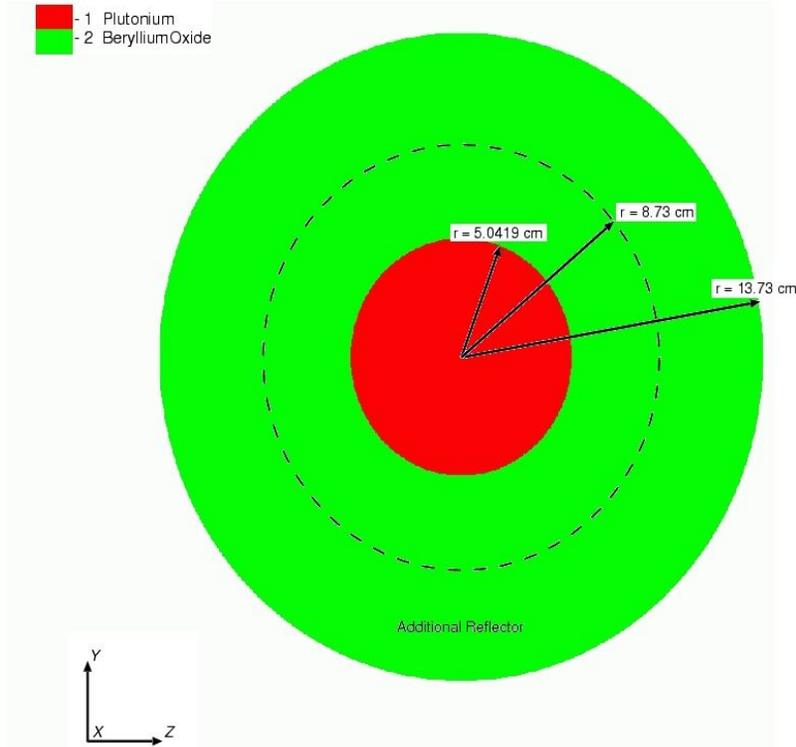### 4.3 Problem 3: A Cylindrical, Highly-Enriched Uranium Critical Assembly

The first criticality problem used in this study is a cylindrical critical assembly consisting of alternating discs of highly enriched uranium (HEU) and beryllium oxide. The specifications for this system were obtained from the *International Handbook of Evaluated Criticality Safety Benchmark Experiments* ("The Handbook") [6], which gives this critical assembly the moniker HEU-MET-FAST-017-001 (HEU metal, fast spectrum, assembly 17, case 1). The geometry of this test problem is illustrated in Figure 5.

The study metrics this for criticality test problem include the k_{eff} eigenvalue and the neutron removal lifetime τ_{rem}. This problem was run using a static k_{eff} eigenvalue method, in which particles are tracked for some number of generations. Each generation ends when all the particles that started the generation have either been absorbed of leaked from the system. A certain number of generations are run in a transient phase to allow the system to approach equilibrium, followed by a second set of equilibrium phase generations over which the results are averaged.

4.4 Problem 4: A Spherical, Plutonium Supercritical System

The second criticality problem is a modified version of a spherical, plutonium critical assembly taken from "The Handbook" [6]. The actual system consists of a fissile core (a 5.0419 cm sphere of d-phase plutonium) surrounded by a reflector (a 3.6881 cm thick spherical shell of beryllium oxide). This system is given the moniker PU-MET-FAST-018-001 (plutonium metal, fast spectrum, assembly 18, case 1). The modified version of this systems adds a second spherical shell of reflecting material (a 5.0 cm thick spherical shell of beryllium oxide) on the outside

of PU-MET-FAST-018-001 to produce a *gedanken* supercritical system with an α eigenvalue (the logarithmic growth rate of the neutron population) of $\alpha \simeq 6.6$ generations/μ sec (see Figure 6 for a graphical representation of this system).

Figure 5. The geometry of Problem 3: a cylindrical, highly-enriched uranium critical assembly.

For this test problem the study metrics include those used in Problem 3, along with the α eigenvalue. This problem was run using a pseudo-dynamic α eigenvalue method, in which particles are tracked through a fixed geometry and medium for some number of time steps (cycles). A certain number of cycles are run in a transient phase to allow the system to approach equilibrium, followed by a second set of equilibrium phase cycles over which the results are averaged.

5 METHODOLOGY AND RESULTS

The aim of this study is to determine the length of the RNG period required to prevent correlations in the pseudo-random number streams, and thus, biases in the metric result that were discussed in the previous section. Each of the four problems described above are run with several RNGs of varying period. For each pairing of problem and RNG period, an ensemble of 25 calculations is run where the initial random number seed is changed for each calculation, in order to determine the mean and standard deviations of each metric result.

Figure 6. The geometry of Problem 4: a spherical, plutonium supercritical system.

The CNPRNG library [3] that is used within MERCURY is a collection of Linear Congruential Generators (LCGs) of the form:

$$x_{n+1} = \left(a \cdot x_n + b\right) \ mod \ m \tag{1}$$

where x_n and x_{n+1} are successive pseudo-random numbers in the sequence, a and b are constants, and m is the modulus of the RNG. The LCGs provided by the CNPRNG library have moduli that are either a power of 2:

$$m = 2^p \tag{2}$$

or prime numbers. For this portion of the study, only power-of-2 modulus LCGs are used for the following set of powers $p = 4, 8, 12, 16, 24, 32, 48,$ and 64. The modulus, and hence period, of each of these LCGs, which is shown in Table I, range from a low of 16 ($p = 4$) to a high of 1.8×10^{19} ($p = 64$).

One of the primary motivation for performing this study was to determine if MERCURY required an RNG with a longer period than that currently supported by the CNPRNG library. If this turns out to be the case, modification of both the RNG library and the Monte Carlo transport code would be required. In particular, it was proposed that a 128-bit version of the power-of-2 LCG could be coded within the CNPRNG library. The resulting pseudo-random number stream would have a period of 3.4×10^{38}, which is long enough to avoid repetitions for more than 10^{20} years on 10^5 processors of the BlueGene/L computer at LLNL, assuming 10^6 random numbers are required per second. Since there are currently no computer systems that support native 128-

Table I. LCG powers and moduli (periods)

p	LCG Modulus (Period)	
4	16	$(1.6\textbf{x}10^1)$
8	256	$(2.6\textbf{x}10^2)$
12	4,096	$(4.1\textbf{x}10^3)$
16	65,536	$(6.6\textbf{x}10^4)$
24	16,777,216	$(1.7\textbf{x}10^7)$
32	4,294,967,296	$(4.3\textbf{x}10^9)$
48	281,474,976,710,656	$(2.8\textbf{x}10^{14})$
64	18,446,744,073,709,551,616	$(1.8\textbf{x}10^{19})$

bit floating point mathematics, these modifications would require that each random number seed be stored as two 64-bit integers.

Each of the calculations performed for this study are run with 10 million particle histories: 10 million particles are injected into the system in each of the source problems, while 25000 particles are run for 400 "settle cycles" (generations or time steps) in each of the criticality problems. This number of histories was chosen a reasonable balance between accuracy and run time. All of the calculations were run on 16 processors in the form of eight 2-way SMP nodes on a parallel Linux cluster. These parallel calculations employed only Domain Replication which provides 16-way particle parallelism.

5.1 Problem 1 Results

The metric results from the calculations of the Problem 1 are plotted as a function of RNG period $(m = 2^p)$ in Figure 7. In the following figures, the red stars represents the ensemble average of 25 calculations, each of which was initialized with a unique set of two 32-bit random number seed fragments. The ensemble standard deviation of each point is indicated by the blue error bars. Note that for Problem 1, the error bars are much smaller than the size of the marker.

The figures clearly indicate that the variation of the average results with p is negligible for $16 \le p \le 64$, but the results can deviate substantially for $p \le 12$ $(m \le 4.1 \times 10^3)$. Once the average results start to deviate as p is made smaller, the trend is not always clear. In some cases the results increase with decreasing p (Figures 7a, 7b and 7c), while the opposite effect can also be found (Figures 7e and 7f). Figure 7 also shows that the deviation from the converged results is not monotonic as p is decreased.

At first glance, one may be surprised to learn that an RNG period of only $m = 6.5 \times 10^4$ is sufficient for this problem, especially when one considers that more than 915 million particle collisions occurred for during these calculations. However, this corresponds to an average of only 91.5 collisions per particle. When one takes into account that there are many random numbers
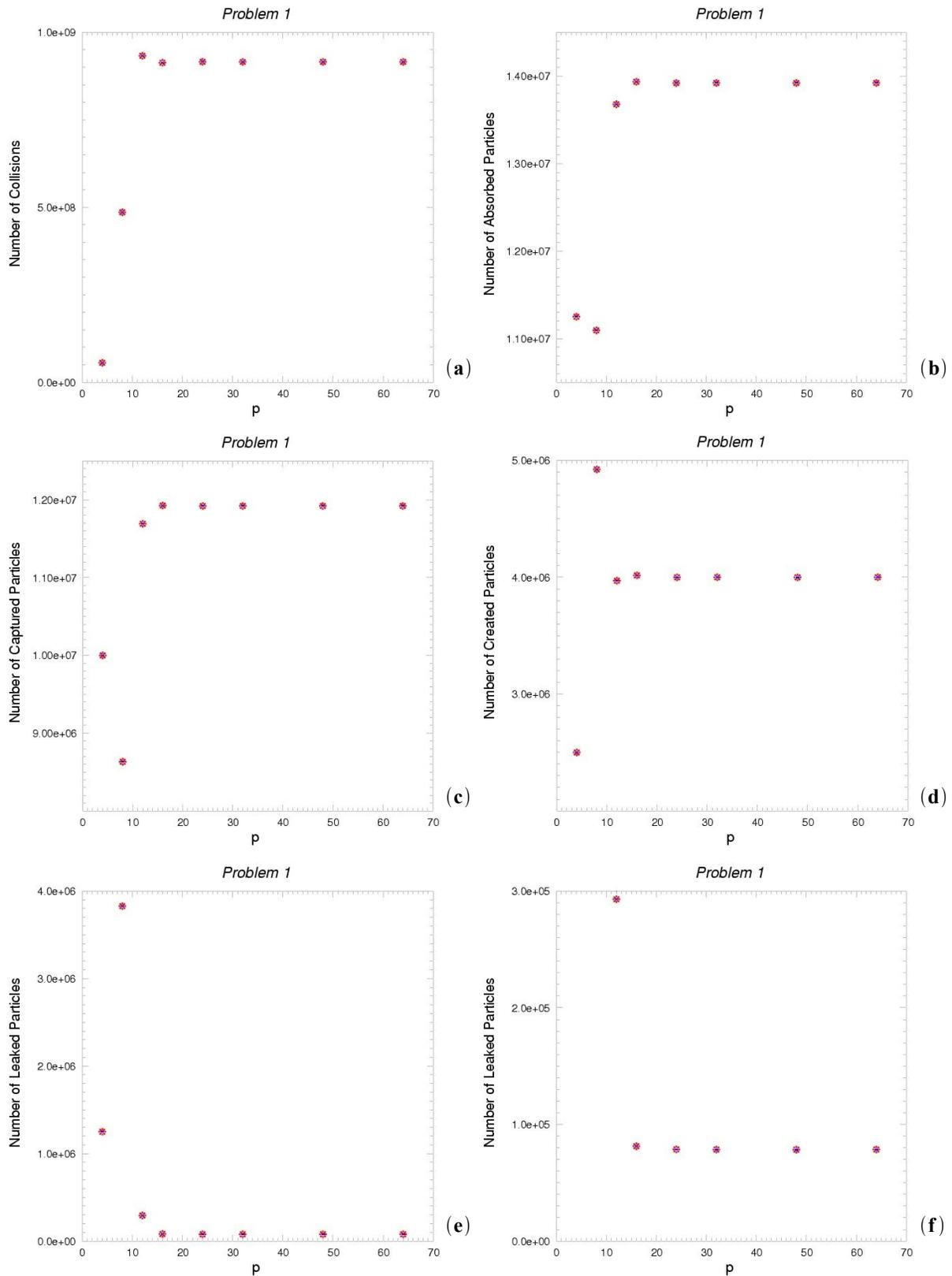
Figure 7. Metric results as a function of RNG period $(m = 2^p)$ **for Problem 1: (a) number of collisions, (b) number of absorbed particles, (c) number of captured particles, (d) number of produced particles, (e) and (f) number of leaked particles (different scales).**

used per collisions, and many particle segments per collision (each of which uses at least one random number), it is not surprising that the results begin to deviate from the converged values for RNG periods $m \leq 4.1 \times 10^3$, or about 45 times the average number of collisions per particle.

This finding illustrates the power of per-particle random number seeds: each particle can undergo a large number (m) of randomly-sampled events before repetitions of pseudo-random numbers will occur, leading to possible correlations in histories and biases in results. Consider the alternative technique of using independent random number streams per processor. For Problem 1, there is an average of 625 thousand particles injected into the system on each processor. This means that the average number of collisions per processor is approximately 57.2 million. Based upon the discussion in the previous paragraph, one might assume that an RNG period on the order of 2 billion $(p \approx 31)$ would be required to prevent repetition of random numbers when the per-processor random number stream method is used.

5.2 Problem 2 Results

The variation of the metric results for Problem 2 with RNG period are shown in Figure 8. Once again, the blue error bars are smaller than the size of the marker, except for Figures 8g and 8h, which show the number of particles passing through the detector and thermal shield regions for times between 90 - 100 μ sec.

For Problem 2, the variation of the average results with p is negligible for $24 \leq p \leq 64$, but the results can deviate significantly for $p \leq 16$ $(m \leq 6.6 \times 10^4)$. As was the case for Problem 1, the variation of the average results with RNG period for $p < 24$ is neither in the same direction nor is it monotonic.

Problem 2 is more complicated than Problem 1, both in terms of the number of zones (about 170 versus 5) and geometrically (see Figures 3 and 4). As a result, it is not unexpected that the period required to prevent biased results is larger for Problem 2 than it is for Problem 1, even though the cumulative number of collisions in both problems is comparable (915 million for Problem 1 and 972 million for Problem 2).

5.3 Problem 3 Results

The metric results from the calculations of the Problem 3 are plotted as a function of RNG period in Figure 9. The variation of the average the k_{eff} eigenvalue with p is shown to be comparable to the error bars for any one p value for $16 \leq p \leq 64$, and the results deviate from the converged eigenvalue for $p \leq 12$ $(m \leq 4.1 \times 10^3)$ (see Figures 9a and 9b). Similarly, the neutron lifetime (τ_{rem}) results are consistent for $24 \leq p \leq 64$, but deviate for $p \leq 16$ (Figure 9c). The average k_{eff} eigenvalue falls off monotonically for $p \leq 12$, but the deviation of the removal lifetime is not monotonic with p. Note that k_{eff} and τ_{rem} are independent tallies from the static k_{eff} eigenvalue method.

5.4 Problem 4 Results

The variation of the metric results for Problem 4 with RNG period are shown in Figure 10. The figure indicates that the average results are converged for $24 \leq p \leq 64$. The average results deviate significantly for $p \leq 16$ $(m \leq 6.6 \times 10^4)$, as was seen for the previous problems.
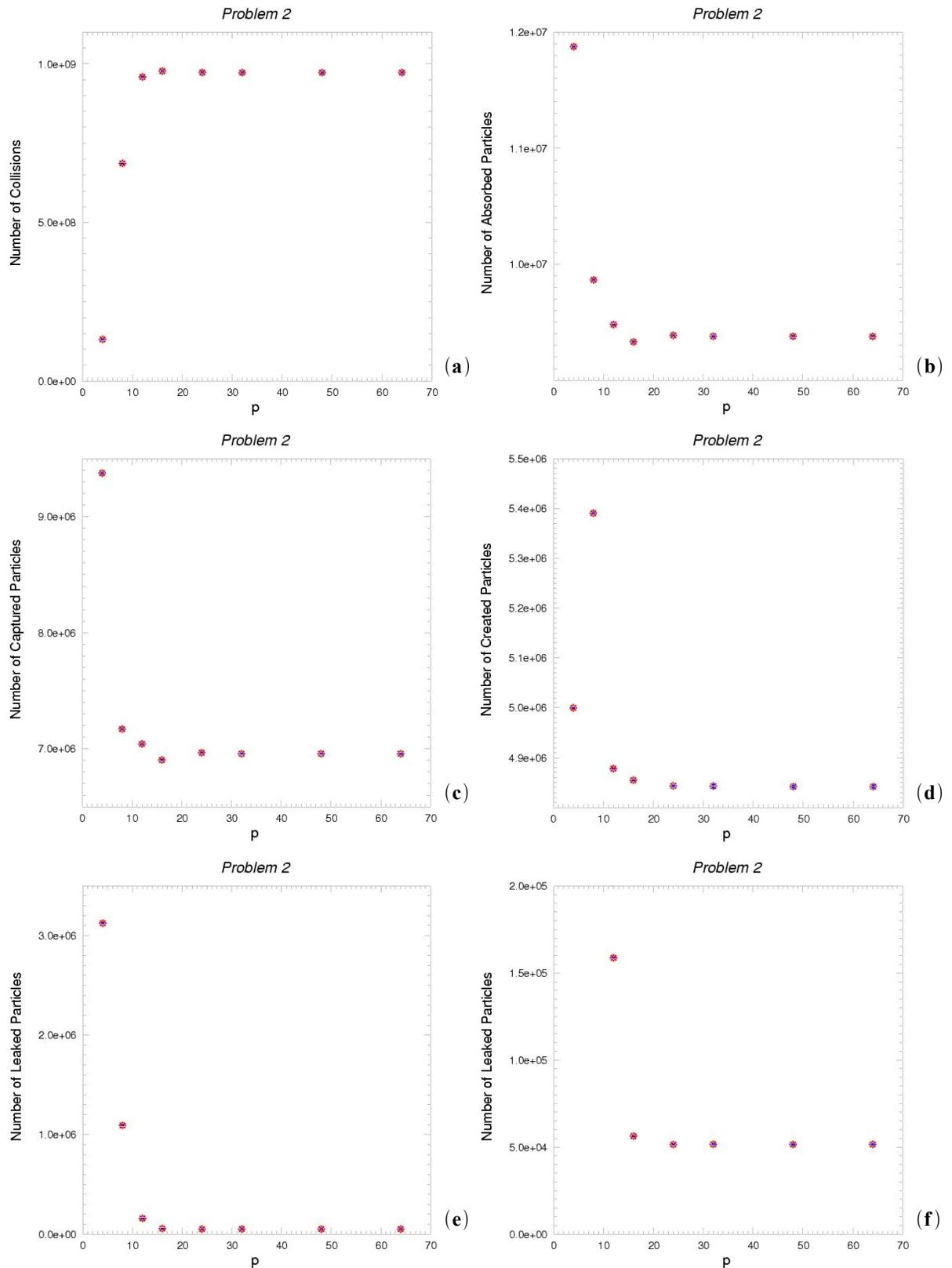
Figure 8. Metric results as a function of RNG period $(m = 2^p)$ for Problem 2: (a) number of collisions, (b) number of absorbed particles, (c) number of captured particles, (d) number of produced particles, (e) and (f) number of leaked particles (different scales).
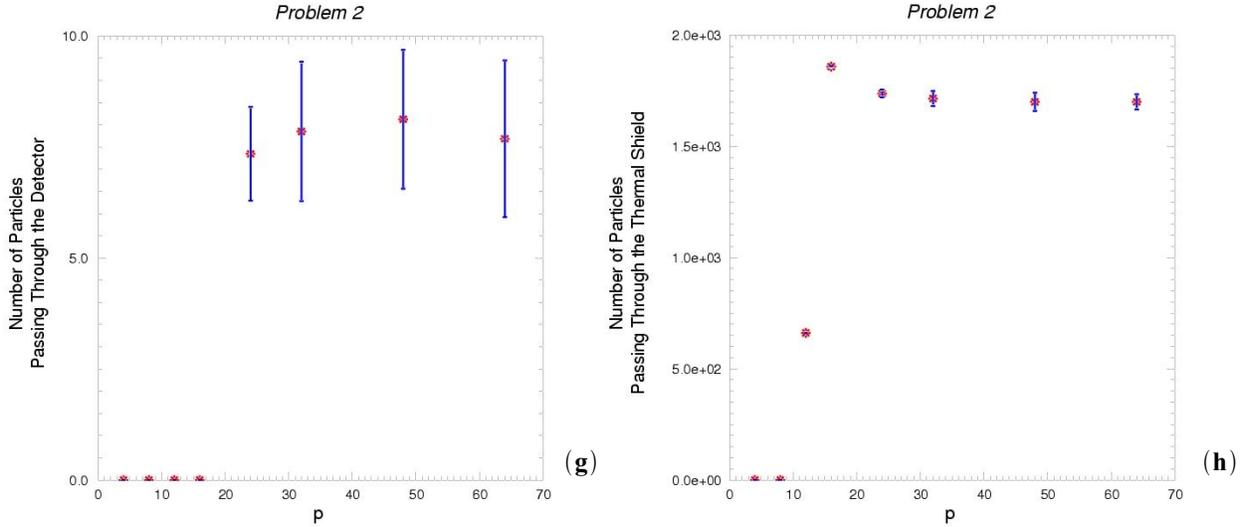
Figure 8 (continued): Metric results as a function of RNG period $(\mathbf{m} = \mathbf{2^p})$ for Problem 2: (g) number of particles passing through the detector region, and (h) number of particles passing through the thermal shield region.

As the RNG period is decreased below $m = 1.7 \times 10^7$ $(p = 24)$, the α and k_{eff} eigenvalues each increase before falling rapidly. In contrast, the removal lifetime (τ_{rem}) results drop, then rise rapidly before falling rapidly.

While k_{eff} and τ_{rem} are independent tallies from the pseudo-dynamic α eigenvalue method, the α eigenvalue tally is related to the others via:

$$\alpha = \left(\frac{1}{\tau_{rem}} \right) \left(\frac{N_{prod}}{N_{leak} + N_{abs}} - 1 \right) = \left(\frac{1}{\tau_{rem}} \right) \left(k_{eff} - 1 \right) \tag{3}$$

where N_{prod} is the number of particles produced during the settle cycle, and $N_{leak} + N_{abs}$ is the number of particles removed (leaked plus absorbed) during the time step. As a result, the deviations from the converged values of k_{eff} and τ_{rem} compound to produce the large observed deviations in α.

6   SUMMARY, CONCLUSIONS AND FUTURE DIRECTIONS

This paper has studied the accuracy of Monte Carlo particle transport calculations in the context of a variable period ("quality") random number generator (RNG). The code MERCURY was used to model two source problems and two criticality problems that are of general interest to the transport community. For each problem, a series of calculations were made by varying the modulus, and hence the period, of the linear congruential generator (LCG) in the range $2^4 \le m \le 2^{64}$. For each of these problem-period pairs, an ensemble of 25 runs were performed by varying the initial random number seeds. Each of these runs employed 10 million particle histories. The average results of several tallies were then compared as a function of m in order
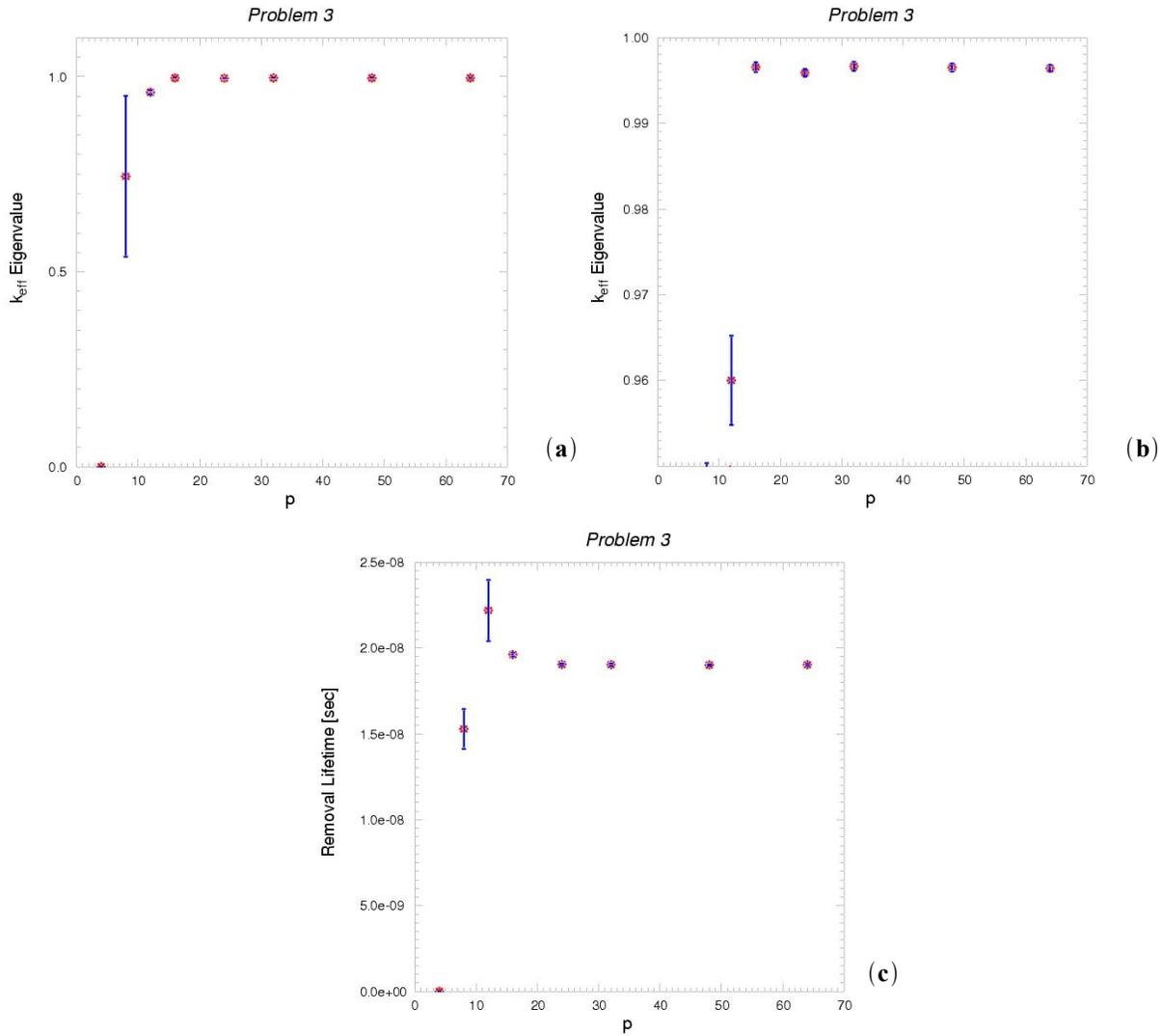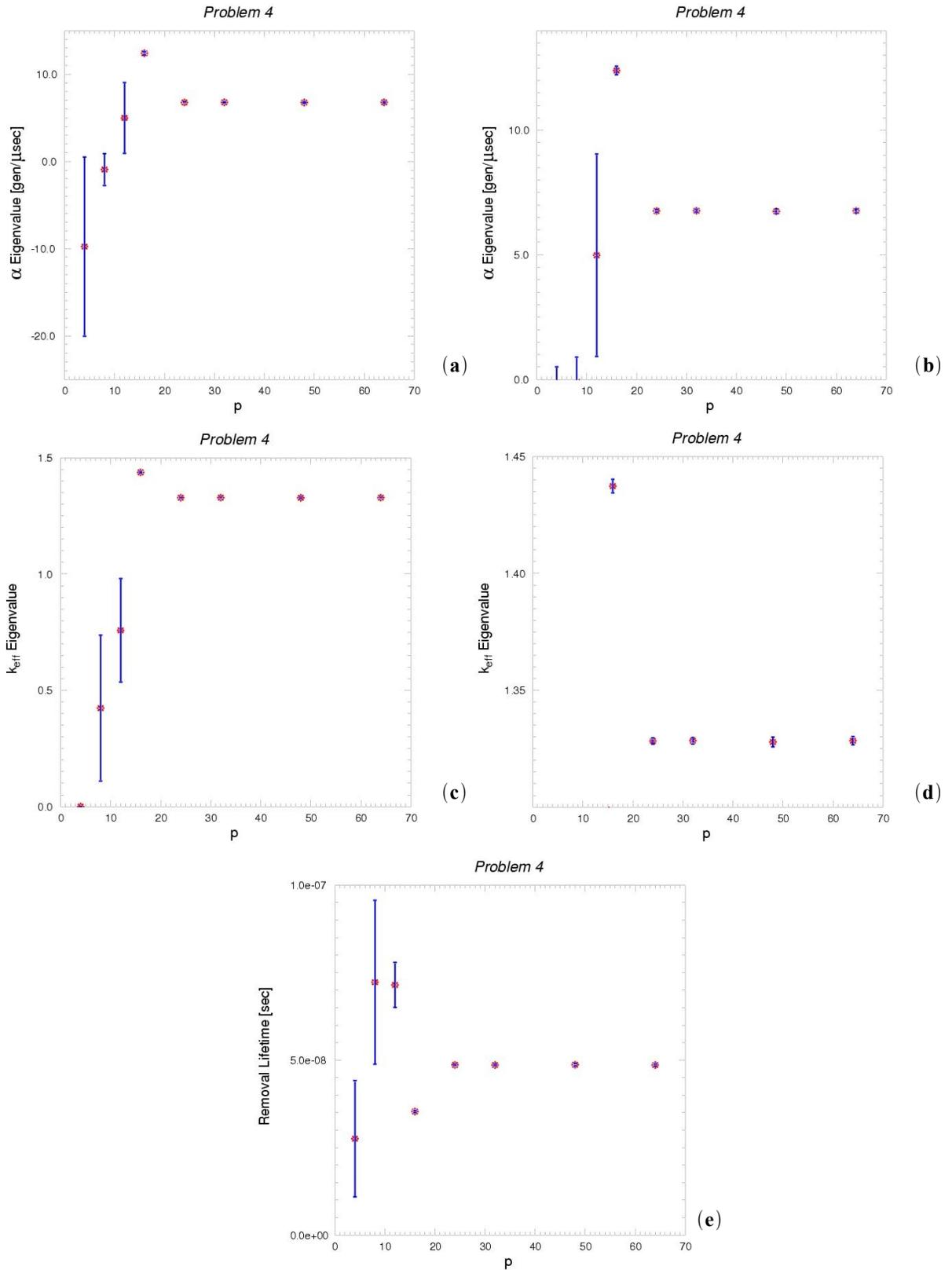
Figure 9. Metric results as a function of RNG period $(m = 2^p)$ for Problem 3: (a) and (b) k_{eff} eigenvalue (different scales), and (c) removal lifetime.

to determine the minimum RNG period necessary for unbiased results from parallel, Monte Carlo transport calculations.

While the conclusions differ slightly between the four problems, the main conclusion of this study is that RNG periods as low as $m = 2^{16} = 65,536$ are sufficient to produce unbiased results. These results were obtained from a parallel code that utilizes per-particle random number seeds, and should not be considered a universal guideline. In particular, a parallel transport code which employs per-processor random number seeds may require RNG periods that are larger by the ratio of the number of particles per processor. An important conclusion of this study is that the current form of random number parallelism used within MERCURY does *not* require us to develop a new RNG library that supports 128-bit LCGs.

Two areas of future research come to mind. The first involves repeating this series of calculations with the prime-modulus LCGs that are available within the CNPRNG library. The prime-number-modulus LCGs have been shown [7] to produce streams of pseudo-random numbers

Figure 10. Metric results as a function of RNG period $(m = 2^p)$ for Problem 4: (a) and (b) α eigenvalue (different scales), (c) and (d) k_{eff} eigenvalue (different scales), and (e) removal lifetime.

with fewer correlations in the bit patters than comparable period power-of-two-modulus LCGs. It would be interesting to determine if the added cost of calculating random numbers with the prime-modulus LCGs would produce superior results to those obtained in the current study.

The second involves modifying the CNPRNG library to allow MERCURY to use random number streams that are assigned on a per-processor basis. This technique would allow us to determine if the per-particle-seed approach to Monte Carlo transport is indeed superior to the per-processor-seed approach for a given RNG period by making a third set of calculations.

7   ACKNOWLEDGMENTS

8   REFERENCES

1. J. E. Gentle, *Random Number Generation and Monte Carlo Methods (Second Edition)*, Springer-Verlag, New York, USA (2003).

2. R. J. Procassini and J. M. Taylor, *MERCURY User Guide (Version b.6)*, Lawrence Livermore National Laboratory, Report UCRL-TM-204296 (2004).

3. B. R. Beck and E. D. Brooks III, *The RNG Random Number Library*, Lawrence Livermore National Laboratory, Internal Report (2000).

4. "SPRNG: Scalable Parallel Pseudo Random Number Generators Library", http://sprng.cs.fsu.edu (2002).

5. R. T. Santoro, R. G. Alsmiller, J. M. Barnes and G. T. Chapman, "Calculation of Neutron and Gamma Ray Spectra for Fusion Reactor Shield Design: Comparison with Experiment", *Nucl. Sci. and Eng.*, **78**, pp. 259-272 (1981).

6. International Criticality Safety Benchmark Evaluation Program, *International Handbook of Evaluated Criticality Safety Benchmark Experiment*, Nuclear Energy Agency, Report NEA/NSC/DOC(95)03/I (2003).

7. B. R. Beck, *Linear Congruential Random Number Generators with Prime Moduli*, Lawrence Livermore National Laboratory, Report UCRL-JC-145424 (2000).