Performance and Accuracy of Criticality Calculations Performed Using WARP - A Framework for Continuous Energy Monte Carlo Neutron Transport in General 3D Geometries on GPUs

Ryan M. Bergmann*, Kelly L. Rowland, Nikola Radnović, Rachel N. Slaybaugh, Jasmina L. Vujić

Department of Nuclear Engineering, 4155 Etcheverry Hall, University of California - Berkeley, Berkeley, CA 94720-1730

Abstract

In this companion paper to “Algorithmic Choices in WARP - A Framework for Continuous Energy Monte Carlo Neutron Transport in General 3D Geometries on GPUs” (doi:10.1016/j.anucene.2014.10.039), the WARP Monte Carlo neutron transport framework for graphics processing units (GPUs) is benchmarked against production-level central processing unit (CPU) Monte Carlo neutron transport codes for both performance and accuracy. Neutron flux spectra, multiplication factors, runtimes, speedup factors, and costs of various GPU and CPU platforms running either WARP, Serpent 2.1.24, or MCNP 6.1 are compared. WARP compares well with the results of the production-level codes, and it is shown that on the newest hardware considered, GPU platforms running WARP are between 0.8 to 7.6 times as fast as CPU platforms running production codes. The GPU platforms running WARP are also shown to be between 15% and 50% as expensive to purchase and between 80% to 90% as expensive to operate as equivalent CPU platforms performing at an equal simulation rate.

*Corresponding author. Tel.: +41.76.687.53.09.

Email addresses: ryanbergmann@gmail.com (Ryan M. Bergmann), krowland@berkeley.edu (Kelly L. Rowland), radnovicn@gmail.com (Nikola Radnović), slaybaugh@berkeley.edu (Rachel N. Slaybaugh), vujic@nuc.berkeley.edu (Jasmina L. Vujić)

Keywords: Monte Carlo, Neutron Transport, GPU, CUDA, OptiX, Supercomputing

1. Introduction

WARP [1], which can stand for “Weaving All the Random Particles,” is a three-dimensional (3D) continuous energy Monte Carlo neutron transport code developed at UC Berkeley to efficiently execute on NVIDIA graphics processing unit (GPU) platforms. WARP accelerates Monte Carlo simulations while preserving the benefits of using the Monte Carlo method, namely, that very few physical and geometrical simplifications are applied. WARP is able to calculate multiplication factors, neutron flux distributions (in both space and energy), and fission source distributions for time-independent neutron transport problems. It can run in both criticality or fixed source modes, but fixed source mode is currently not robust or optimized. WARP can transport neutrons in unrestricted arrangements of parallelepipeds, hexagonal prisms, cylinders, and spheres.

The goal of developing WARP is to investigate algorithms that can grow into a full-featured, continuous energy, Monte Carlo neutron transport code that is *accelerated* by running on GPUs. The crux of the effort is to make Monte Carlo calculations faster while producing accurate results. Modern supercomputers are commonly being built with GPU coprocessor cards in their nodes to increase their computational efficiency and performance. Compared to more common central processing units (CPUs), GPUs have a larger aggregate memory bandwidth, much higher rate of floating-point operations per second (FLOPS), and lower energy consumption per FLOP. GPUs execute efficiently on data-parallel problems, but most CPU codes, including those for Monte Carlo neutral particle transport, are predominantly task-parallel. Data-parallelism is parallelism that arises from a single task operating on many different pieces of data at one time, whereas task-parallelism is parallelism that arises from running many concurrent tasks that each act on a single piece of data. Figure 1 shows an illustration of the difference between a data-parallel and a task-parallel neutron transport loop. Therefore, to create Monte Carlo software that works well on GPUs, the transport algorithms had to be reconsidered.

In this paper, we start in Section 2 discussing background information about GPUs and CPUs in the context of Monte Carlo neutral particle transport. Then, in Section 3, the current features of WARP are summarized while

Data-Parallel

Task-Parallel

Figure 1: A data-parallel neutron transport loop vs. a task-parallel transport loop for transporting N neutrons in parallel.

35 highlighting the newest developments. The results of criticality calculations
 36 performed by WARP are compared against those from Serpent 2.1.24 [2, 3]
 37 and MCNP 6.1 [4], two widely-used production-level Monte Carlo neutron
 38 transport codes. This comparison is done to ensure the accuracy of WARP
 39 and to highlight performance differences. The test case models used to con-
 40 duct the comparisons are outlined in detail in Section 4, then the run times,
 41 multiplication factors, and neutron flux spectra calculated for the test cases
 42 are compared in detail in Section 5, after which a discussion is given sum-
 43 marizing comparisons and addressing any discrepancies shown in Section 6.
 44 Finally, conclusions about the initial development of WARP are drawn and
 45 future work is roughly outlined in Section 7.

46 **2. Background**

47 Typical Monte Carlo neutron transport algorithms are “task-based.” A
 48 thread will transport the neutron until it is terminated through absorption or
 49 by leaking out of the system. The sequence of events the neutron undergoes
 50 from start to finish is called a neutron “history,” and is the basic unit of work
 51 for a thread in a task-based algorithm. Neutron histories are independent
 52 of each other and the algorithm is parallelized by running many histories
 53 in independent, parallel threads. Thus, having as many parallel threads
 54 as possible should provide the greatest performance. This parallelization
 55 method requires that the threads can execute completely independently of
 56 each other, i.e. the actions of one thread do not affect the other parallel
 57 threads in any way. This is a MIMD (multiple instruction multiple data)
 58 way of executing threads. In MIMD execution model, different threads can
 59 execute different instructions on their data at a point in time. Threads
 60 appear to completely independent, and having threads at different points in
 61 the transport algorithm is not a problem since they are allowed to execute
 62 their own instructions.

63 At first glance, it seems that GPU cards would be good for running Monte
 64 Carlo neutron transport because they can run large numbers of concurrent
 65 threads to complete the large number of independent tasks required in tra-
 66 ditional Monte Carlo neutron transport. However, it soon becomes apprent
 67 that the situation is complicated by how instructions are carried out on these
 68 threads. The reported concurrent thread number is based on the width of the
 69 processor’s SIMD (Single Instruction Multiple Data) units. SIMD is an exe-
 70 cution model some processors use to lower the number of instructions needed

71 per amount of computation done, which increases both power and computa-
72 tional efficiency [5]. SIMD requires the same instructions to be carried out
73 over every element in a concurrently-processed data vector.

74 An NVIDIA GPU’s basic processing unit is a streaming multiprocessor
75 (SM). These multiprocessors house many individual computational cores and
76 can process jobs independently of each other. One instruction is carried out
77 over the entire SM, however, so it is the processing level above which SIMD
78 requirements are no longer present. The GPU programming model abstracts
79 SIMD execution by using threads, which can be thought of in the traditional
80 sense. Single-instruction multiple-thread (SIMT) and SIMD are similar in
81 the sense that identical instructions are carried out across different pieces of
82 data, but SIMT allows threads to act independently, albeit with a perfor-
83 mance penalty. From a thread standpoint, SIMT requires threads to execute
84 the same instruction at one point in time. If a set of threads does not ex-
85 ecute the same instruction at the same time (the data they act on can be
86 different), the GPU will serialize the operations. In this case, the subset
87 of *divergent* threads executing the first instruction will all execute together,
88 then the subset of divergent threads executing the second instruction will all
89 execute together after the first subset has completed. In the limit of each
90 thread executing a different instruction, the operations become completely
91 serial. The magnitude of the performance penalty depends on the rate at
92 which data is being delivered to the multiprocessor since work can only be
93 done if data is present. If the multiprocessor can complete all jobs, even
94 if it needs some serialization, before the next set of data arrives, then the
95 penalty will be invisible [6]. To most efficiently use SIMT and avoid se-
96 rialization, instruction regularity¹ is required. Unfortunately, Monte Carlo
97 typically breaks instruction regularity because of its conditional statements
98 based on random numbers. Therefore, if Monte Carlo algorithms are to be
99 used on GPUs, they must be implemented in a manner that carefully takes
100 into account the limitations of the GPU.

101 A further consideration is that the memory subsystems of GPUs also func-
102 tion in a SIMD-like way. In order to use the full memory bandwidth of the
103 device, more than one piece of data must be loaded and used per transaction,

¹“Instruction regularity” means some systematic behavior for the instructions that ad-
jacent threads need to execute. Ideally, they should all be executing the same instructions,
but operating on different pieces of data.

104 and the only way that multiple pieces of data can be loaded simultaneously
 105 is if they are adjacent in memory. In other words, if a program requests a
 106 single piece of data at location i and a piece at location $i + 10$, these requests
 107 will be split into two separate transactions that each yield one data element.
 108 Two data elements in two transactions produces an effective bandwidth of
 109 one element per transaction time. On the other hand, if the program requests
 110 data at i , $i + 1$, $i + 2$, and $i + 3$, the entirety of the requested data can be
 111 retrieved in a single transaction. Four data elements in a single transaction
 112 yields an effective memory bandwidth of four elements per transaction time,
 113 four times higher than the previous scenario. Having a memory subsystem
 114 that handles requests in this way is not ideal for Monte Carlo methods, how-
 115 ever. With Monte Carlo, data is accessed in a very random way because of
 116 the random nature of the simulation, and requested data is unlikely to be
 117 adjacent. This means the full memory bandwidth of the GPU will not be
 118 used unless this problem is mitigated in some way.

119 Another challenge of using GPUs for Monte Carlo is GPUs have very high
 120 global memory latency compared to a CPU. Memory latency is the number
 121 of clock cycles, or amount of time, it takes for a data request to be fulfilled.
 122 As Table 1 shows, the GPU’s global memory latency is about an order of
 123 magnitude higher than the CPU’s [7, 6]. GPUs try to eliminate the effect of
 124 large global latency by pipelining memory access. Pipelining means threads
 125 that have received their data can execute as other threads are waiting for
 126 their data to load. If many requests are known, the data can be continually
 127 loaded as threads start to execute their jobs. The hope is that the jobs
 128 take longer than the memory loads, eventually all data arrives, and the later
 129 threads appear to have zero latency for their memory access. This is why it
 130 is important for GPUs to have such a large number of concurrent threads.
 131 It allows them to pipeline data access and minimize the impact of memory
 132 latency. A CPU system is a like a fast, strong horse that can move some
 133 goods from A to B very quickly by itself. If only one cart of goods is allowed
 134 on the path from A to B, then a GPU system is like a large team of slower
 135 ponies that can move a larger load of goods in a larger time, but only if they
 136 work together in moving the same, large load. Averaged over time, the team
 137 of ponies can move more goods than the horse if there are enough of them
 138 working together.

139 Another notable feature is that the GPU has a greater FLOPS/byte of
 140 memory bandwidth ratio than the CPU. This implies that GPUs could be
 141 used to turn a compute-bound problem into a bandwidth-bound problem.

Table 1: A comparison of an NVIDIA GPU and an Intel CPU [8, 7, 9].

Processor	Intel i7 (Westmere-EP)	NVIDIA Tesla C2075 (Fermi)
Processing Elements	6 cores, 2 issue, 4-way SIMD	14 cores, 2 issue, 16-way SIMD
Frequency	3.46 GHz	1.15 GHz
Resident Strands / Threads (max)	48	21,504
Single Precision GFLOPS	166	1030
Mem. Bandwidth	32 GB/s	144 GB/s
Global Latency	~50 clocks	200-800 clocks
FLOPS / byte	5.2	7.2
Register File	6 kB	2 MB
Local Storage / L1 Cache	192 kB	896 kB
L2 Cache	1536 kB	0.75 MB
L3 Cache	12 MB	-

142 This may seem like a deficit, but the GPU has a higher maximum memory
143 bandwidth, so even though a problem is bandwidth-bound on a GPU, it may
144 still require less execution time than on a CPU.

145 The impetus for developing WARP was the research done by Martin and
146 Brown in 1984 [10] for Monte Carlo and by Vujić and Martin in 1991 [11] for
147 the collision probability method. In the 1984 paper, a method for mapping
148 the Monte Carlo problem onto SIMD vector computers is described. The
149 essential idea in the 1984 paper was to bank the neutrons into vectors based
150 on their required operation. If a neutron was sampled to scatter, its data were
151 placed into a buffer containing the data of other neutrons that have also been
152 sampled to undergo scattering interactions. If a neutron crosses a surface, it
153 was placed into the surface crossing buffer, and so on. Once a buffer became
154 full, it was processed in a SIMD fashion by the vector computer. Processing
155 the neutron data further after the vector operation caused the buffers to
156 contain non-uniform reactions, however, and a “shuffle” operation was done
157 that actually moved data back into contiguous blocks based on the reaction
158 type.

159 This new approach was named “event-based” Monte Carlo, since the
160 neutron events are tracked and processed as a group. This was a very dif-
161 ferent way of performing a Monte Carlo simulation at the time. Almost all

162 computers were strictly serial, and SIMD lanes were only available in super-
163 computers. Therefore, the pervasive method was the “task-based” method
164 in which neutrons are tracked for their entire lifetime in series. Since GPUs
165 are massively parallel and rely on SIMD, an event-based algorithm seemed
166 to be the appropriate approach to GPU-accelerated neutron transport.

167 Instead of trying to port an existing code to the GPU, the simplest way
168 to accommodate requirements for efficient GPU execution was to write a new
169 code from scratch. This project ultimately resulted in WARP. WARP uses
170 an event-based algorithm, but with some important differences. Vectorizing
171 the Monte Carlo algorithm should allow for efficient GPU execution, but a
172 paper by Zhang [12] also shows that by remapping data references, the thread
173 divergence in GPU warps (groups of threads) can be minimized. Moving data
174 is expensive, especially when the data set is large, so WARP uses a remapping
175 vector of pointer/index pairs to direct GPU threads to the data they need to
176 access. The remapping vector is sorted by reaction type after every transport
177 iteration using a high-efficiency parallel radix sort (via the CUDPP library
178 [13]), which serves to keep the reaction types as contiguous as possible and
179 removes completed histories from the transport cycle. Sorting reduces the
180 amount of divergence in GPU “thread blocks,” keeps the SIMD units as full
181 as possible, and eliminates using memory bandwidth to check if a neutron in
182 the batch has been terminated or not. Using a remapping vector means the
183 data access pattern is irregular, but this is mitigated by using large batch
184 sizes where the GPU can effectively reduce the high cost of irregular global
185 memory access. The details about the existing algorithms used in WARP
186 are discussed in [1] and will not be discussed in detail here. New algorithmic
187 changes added since the last publication are discussed in Section 3.

188 To use NVIDIA GPUs, parts of software must be written in a language
189 that facilitates interaction with the GPU, and CUDA [6], a set of extensions
190 for C/C++, was chosen. CUDA was first released in 2006 as NVIDIA’s
191 proprietary GPU programming platform. It makes minimal additions to
192 C/C++, and any C programmer would be very comfortable programming
193 in CUDA [6]. It was chosen over OpenCL, the open source GPU program-
194 ming platform, because of CUDA’s greater feature support, stability, ease of
195 programming, wider community usage, and ability to use new, cutting-edge
196 features in NVIDIA GPUs that OpenCL does not have.

197 3. Features of WARP

198 WARP has only existed since 2013, and it is not as fully-featured as more
199 mature Monte Carlo codes. It has the functionality necessary to compare its
200 performance against Serpent 2.1.24 and MCNP 6.1, however. This section
201 outlines the current set of features available in WARP. WARP is mainly
202 written in C/C++ and is compiled to a shared library. With the shared
203 library compiled, a user must write a `main()` function that calls the library
204 routines and directs program flow. In other words, there currently are no
205 input processing routines included with WARP to interpret text file inputs.
206 Instead, the simulations are configured by providing arguments to the library
207 routines themselves. Therefore, when one wants to change the simulation,
208 the arguments to the WARP routines must be changed, and this requires
209 the `main()` function to be recompiled. This is why WARP is termed a
210 “framework” rather than a “program.” Of course, there is flexibility in the
211 setup as well, and a user could write a main function that could handle
212 all conceivable input cases and would never have to recompile the `main()`
213 function.

214 Pythonic wrapping of the shared library has been done via SWIG [14,
215 15], which automatically wraps compiled languages like C/C++ in high-
216 level scripting languages. With the C++ classes exposed in Python, the
217 `main()` function can be replaced with a Python script, eliminating the need
218 to recompile a main function and link it to the WARP library when different
219 geometries or different run parameters are desired. Using a Python script to
220 call the WARP library is effectively equivalent to having a text input file,
221 but also allows users to use programming structures like loops and regular
222 expressions directly in their “input files.” Eliminating the text input file
223 removes a level of developer programming effort for input parsing, and allows
224 direct access to the results’ data arrays for easy post processing and plotting.
225 Also, eliminating files as the sole program output removes user error and
226 effort in parsing them for post processing and plotting.

227 The Python wrapping approach deviates from the standard flat text input
228 file structure that many Monte Carlo codes use. Flat text input/output
229 relies on keywords and/or regular formatting, which adds a layer between
230 simulation and analysis. Using Python to directly access the classes and
231 their data removes this layer and allows a user to build complex, custom
232 applications as desired. With Python, the results of a calculation are also
233 resident in a Python session and are therefore easily available to the user for

234 plotting with scripts or processing with other analysis tools. To process data
 235 in the same way from a text file output, the output would need to be parsed
 236 with a user-written function or processed by hand, which is time-consuming
 237 and can lead to human error.

238 3.1. Physics

239 Only neutrons are transported by WARP. Any other particles are not
 240 considered in any way. Cross section data compiled by the United States
 241 is distributed by the Department of Energy in *ENDF* files. ENDF stands
 242 for “Evaluated Nuclear Data File” and can contain data for nuclear decay,
 243 photons, atomic relaxation, fission yields, thermal neutron scattering, and
 244 charged particle reactions as well as neutron reactions [16, 17]. WARP loads
 245 ACE-formatted nuclear data libraries via the “ace” module in PyNE (Python
 246 for Nuclear Engineering) [18]. This module has been separated from PyNE
 247 and included as a standalone Python module with WARP. Including it as
 248 a separate module with WARP means the entire PyNE package does not
 249 need to be installed in order to compile and run WARP. ACE stands for “A
 250 Compact ENDF” and strips out much of the extra information unnecessary
 251 for neutron transport and formats the data into the specified tabulation [17].
 252 Since many Monte Carlo codes read ACE-formatted data rather than the
 253 original ENDF file, WARP can eliminate one potential cause for discrepancies
 254 by loading the same data as other codes. The cross section data are then
 255 reformatted to use a unionized energy grid² via NumPy [19]; the reformatted
 256 data are then passed to the C++ routines via the Python C API.

257 In its current state, WARP does not use $S(\alpha, \beta)$ thermal scattering tables
 258 or unresolved resonance parameters. It only uses the free-gas approximation
 259 to account for the motion of material nuclei. The free-gas approximation
 260 treats the target as part of a gas at a certain temperature and does not con-
 261 sider other modes of energy transfer that rise from inter- and intra-molecular
 262 phenomena. Thermal scattering and unresolved resonance data improve the
 263 physical fidelity of the simulation, but these features were turned off in pro-
 264 duction codes so that WARP could be directly compared to them.

265 WARP currently has an incomplete set of the ENDF sampling laws im-
 266 plemented: laws 3 (level scattering), 4 (continuous tabular distribution), 7

²A “unionized energy grid” is a common set of energy points where all cross sections have defined values as opposed to every cross section having its own individual set of energy points.

(simple Maxwell fission spectrum), 9 (evaporation spectrum), 11 (energy-dependent Watt spectrum), 44 (Kalbach-87 formalism), 61 (LAW=44 but tabular angular distribution), and 66 (N-body phase space distribution) [20]. These laws cover all the reactions present in the test problems presented here and most nuclides in the ENDF/B-VII.1 cross section data set, but some nuclei may have interactions that require the remaining neutron sampling laws: 1 (tabular equiprobable energy bins), 5 (general evaporation spectrum), 22 (tabular linear functions), 24 (UK law 6), and 67 (laboratory angle-energy law). Problems containing any such nuclides cannot be simulated with the current version of WARP, but could be added later. These unincluded laws are not nearly as common as the ones currently included in WARP, and excluding them at this point should not greatly impact the ability to benchmark WARP's performance and accuracy.

WARP only has collision estimators implemented for both flux and multiplication factor estimation. The relative errors of these estimations are calculated, but any additional convergence estimation beyond relative error has not been implemented. Variance reduction is not implemented in WARP. Neutrons have statistical weight in WARP, and currently the only reactions that adjust neutron weight are multiplicity reactions, like (n,2n), where the neutron weight is multiplied by the exiting neutron number in a manner similar to OpenMC [21]. This means that new neutron histories do not have to be created mid-batch in criticality calculations, and makes program flow simpler. Analog capture was turned on in Serpent and MCNP in all simulations considered to ensure a fair comparison of the results.

3.2. Geometry

The geometry representation in WARP is handled by the NVIDIA OptiX ray tracing framework [22]. OptiX provides good performance on the GPU, but imposes some limitations on how geometries must be input. For the current algorithm used, the geometrical limitations are:

1. Cells are the basic building blocks of a model, not surfaces.
2. All cells must be finite, closed, and non-overlapping.
3. Implicit nesting determines what material exists inside of a cell. In other words, since cells are defined directly, the material of cell A is only that space where cell A is the lowest nested cell. If cell B is within cell A, the entire space within cell B is excluded from cell A simply because cell B resides within cell A. It does not need to be excluded explicitly.

- 304 4. The only cell types available are spheres, cylinders, right rectangular
305 prisms, and right hexagonal prisms.
- 306 5. Only translational transforms have been implemented.
- 307 6. Vacuum (i.e. infinitely absorbing) and specular reflection (i.e. mirror)
308 boundary conditions are available.

309 An improved tracking routine, which determines which cell a neutron
310 resides in based on its spatial coordinates, has been implemented in WARP
311 since the last publication [1]. Previously, a ray was traced from the neutron
312 position to the outer cell, and a list of cell numbers was stored, in order, for
313 each intersection. The double entries were removed as the list was written,
314 which, at the end of the trace, yielded a list of cells the neutron was nested
315 in. This resulted in very poor performance in deeply-nested cases due to this
316 list becoming large. For example, WARP had very significant performance
317 decreases in the Jezebel spherical criticality test (discussed later) if the sphere
318 was segmented into many spherical shells. Doing this caused WARP to store
319 a long list of surface numbers and made the simulation very slow. WARP
320 now uses “cell sense” to reduce the memory impact of determining in which
321 cell/material a neutron resides. “Surface sense” is a basic feature of many
322 computational solid geometry (CSG) systems, and is simply the sign of the
323 remainder left after evaluating a set of coordinates in a surface equation.
324 “Cell sense” is like surface sense in that it is positive if a neutron is outside a
325 cell and is negative if a neutron is inside the cell. Cell sense can be calculated
326 by taking the product of the surface senses of the cell’s constituent planes.
327 In the new and improved tracking routine, the cell sense (which is either 1 or
328 -1) is calculated and summed as the query ray traverses the geometry. When
329 the sum becomes negative, the last intersected surface is the cell in which
330 the neutron is located. Figure 2 shows an illustration of the new geometry
331 query algorithm. Using the new surface sense traversal method results in
332 high performance even in the previously mentioned segmented Jezebel case.

333 When the cumulative sense becomes negative, the containing cell has
334 been intersected and tracing can stop. This approach can be used because
335 all bodies must be closed. This scheme takes advantage of the acceleration
336 structure in OptiX while still using the simplicity of combinatorial solid ge-
337 ometry. It essentially is the same scheme as MCNP/Serpent, except that the
338 acceleration structures allow surfaces that are part of cells that are far away
339 to be skipped. Also, compared to the past scheme of storing a list of doubles,
340 this method requires a single integer to be stored and operated on instead of

Figure 2: The improved geometry query algorithm showing how the cell number is calculated through ray tracing. A ray is iteratively drawn from the neutron starting position to the outer cell, and the surface normal values are summed along the way. When the sum reaches -1, the cell in which the neutron resides has been found and the trace stops.

341 a potentially large and slow list.

342 Since OptiX allows variables to be attached to individual geometrical
 343 objects, this feature has been exploited in order to create an efficient general
 344 tally routine. After the geometry has been input and desired tallies have
 345 been specified, WARP creates an array of tally data structures and attaches
 346 the tally index directly to each geometrical object that the tally is associated
 347 with. When OptiX is called to calculate the nearest intersection points and
 348 determine which cell a neutron is in, it also reports back the index of the tally
 349 (if any) that should be scored at a collision. This way, the high performance
 350 of the OptiX library is leveraged to perform the hash between cell number
 351 and tally index, and a separate search does not have to be performed. The
 352 material index is returned by OptiX in an identical way.

353 4. Tests

354 A small set of CPU and GPU platforms were chosen for use in testing
 355 WARP. These platforms were chosen simply because they were easily accessi-
 356 ble from UC Berkeley. For the CPU platforms, a single cluster node of both
 357 Berkelium (Bk), the departmental cluster, and Savio2, the shared campus
 358 high performance computing cluster, were used. For the GPU platforms, a
 359 consumer-level graphics card (Titan) and two compute-specific cards (Tesla)
 360 were used. Descriptions of each piece of hardware can be found in Table 2.
 361 For the GPUs, the costs include the retail price paid by the department plus
 362 that of a minimal host computer with at least as much memory as the card
 363 (~\$400). The costs of the The “physical processor” count shown in Table 2
 364 for the GPUs is the number of SMs present in each card.

Table 2: Platforms used in the benchmark cases and their specifications. Cost are those incurred when purchasing the systems and are approximately 2014 values.

Platform	Cost (USD)	Total Processor Power (Watts)	Local Memory	Memory Frequency
Bk PSSC PowerWulf Blade	9,310	460	96 GB	1.6 GHz
Savio2 Lenovo NeXtScale nx360 M5	6,770	210	64 GB	2.133 GHz
NVIDIA Tesla K20	3,325	225	4.8 GB	2.6 GHz
NVIDIA Titan Black	1,521	250	6.14 GB	3.5 GHz
NVIDIA Tesla K80	5,000	300	12 GB	2.505 GHz

Platform	Physical Processors	Processor Frequency	Maximum Threads
Bk PSSC PowerWulf Blade	4x AMD Opteron 6172	2.1 GHz	48
Savio2 Lenovo NeXtScale nx360 M5	2x Intel Xeon E5-2670 v3	2.3 GHz	24
NVIDIA Tesla K20	13	705.5 MHz	$2^{31} - 1$
NVIDIA Titan Black	15	1071.5 MHz	$2^{31} - 1$
NVIDIA Tesla K80	13	823.5 MHz	$2^{31} - 1$

365 All tests on all codes use ENDF/B-VII cross sections that are distributed
 366 with the Serpent 1.1.7 release from RSICC (Radiation Safety Information
 367 Computational Center). RSICC regulates the licensing and distribution of
 368 the MCNP and ENDF data worldwide, and also distributes Serpent 1.1.7
 369 [23]. The cross sections distribtued with Serpent are used since they contain
 370 fewer energy grid points than the ENDF/B-VII.1 cross sections distributed
 371 with the MCNP 6.1. The Serpent 1.1.7 data therefore require less storage

³⁷² space, and using them allows more isotopes to fit into card memory when
³⁷³ running WARP.

374 4.1. Test 1 - “Jezebel” Bare Pu Sphere

375 The “Jezebel” criticality test is a bare plutonium/gallium sphere with
 376 vacuum boundary conditions, which can be seen in Figure 3. The fission
 377 neutron rate from ^{239}Pu is balanced by the leakage rate from the 6.6595 cm
 378 radius to give a k_{eff} of approximately 1. Since this system is so leaky, pro-
 379 ducing results consistent with MCNP and Serpent ensures that the boundary
 380 conditions are correctly being enforced. The Jezebel test is a standard test
 381 used to validate neutron transport codes and is described in the Interna-
 382 tional Handbook of Evaluated Criticality Safety Test Experiments under the
 383 name “Pu-MET-FAST-001” [24]. The geometry and materials are outlined
 384 in Table 3. All cross sections used were processed at 273.5 K.

Figure 3: Horizontal slice of the geometry of the “Jezebel” test case. The purple color represents the material in the single cell, and the black color represents space outside the outermost boundary.

Table 3: Geometry and materials used in the “Jezebel” test case.

Cells	Isotopes (Atm. %)		Density
1 sphere, r=6.6595 cm	^{239}Pu (0.7381)	^{240}Pu (0.1942)	15.73 g/cm ³
	^{241}Pu (0.0299)	^{242}Pu (0.0038)	
	^{69}Ga (0.0203)	^{71}Ga (0.0135)	

385 *4.2. Test 2 - Homogenized Fuel Block*

386 The homogenized block criticality test is a bare cube with vacuum bound-
 387 ary conditions as seen in Figure 4. This test keeps the same boundary con-
 388 dition, temperature, and number of cells as test 1, but is larger (reducing
 389 leakage), contains light isotopes (which readily produce thermal neutrons),
 390 and introduces new isotopes. Introducing new isotopes into the simulation is
 391 significant in that new isotopes may use any of the various sampling laws out-
 392 lined earlier. Comparing spectra from calculations containing many different
 393 isotopes simply shows that WARP can handle sampling the laws correctly.
 394 The geometry and materials are outlined in Table 4. All cross sections used
 395 were processed at 273.5 K.

Figure 4: Horizontal slice of the geometry of the homogenized fuel block test case. The purple color represents the material in the single cell, and the black color represents space outside the outermost boundary.

Table 4: Geometry and materials used in the homogenized fuel block test case.

Cells	Isotopes	(Atm. %)	Density
1 cube, 100x100x50 cm	²³⁸ U (0.90)	²³⁵ U (0.10)	5.50 g/cm ³
	¹⁶ O (3.00)	² H (2.0)	
	⁹⁰ Zr (0.5145)	⁹¹ Zr (0.1122)	
	⁹² Zr (0.1715)	⁹⁴ Zr (0.1738)	
	⁹⁶ Zr (0.0280)		

396 4.3. Test 3 - Zr-Clad UO_2 Pin in Heavy Water

397 This criticality test, shown in Figure 5, consists of a UO_2 cylinder clad
 398 in zirconium surrounded by a large block of heavy water. This test increases
 399 complexity by having three materials, each with multiple isotopes, and three
 400 cells. The water block has vacuum boundary conditions. Its dimensions are
 401 relatively large and the absorption is low, so the mean neutron lifetime should
 402 be large. This test serves to demonstrate that all the processing routines work
 403 simultaneously, the effect of introducing more than one cell, and the effect
 404 of long-lived neutrons. The geometry and materials are outlined in Table 5.
 405 All cross sections used were also processed at 273.5 K.

Figure 5: Horizontal slice of the geometry of the Zr-clad UO_2 pin in heavy water test case. Colors represent cell numbers and black represents space outside the outermost boundary.

Table 5: Geometry and materials used in the single UO_2 pin in D_2O test case.

Cells	Isotopes (Atm. %)	Densities
1 cylinder, $r=2.0$ cm $z=\pm 20$	^{238}U (0.90) ^{235}U (0.10) ^{16}O (2.00)	10.97 g/cm ³
1 cylinder, $r=2.2$ cm $z=\pm 20.2$	^{90}Zr (0.5145) ^{91}Zr (0.1122) ^{92}Zr (0.1715) ^{94}Zr (0.1738) ^{96}Zr (0.0280)	6.52 g/cm ³
1 box, 50x50x50 cm	^2H (2.0) ^{16}O (1.0)	1.11 g/cm ³

406 4.4. Test 4 - Homogenized Fuel Pebble in FLiBe

407 This criticality test consists of a single sphere of homogenized UO_2 and
 408 C surrounded by molten FLiBe (Li_2BeF_4) salt, shown in Figure 6. The
 409 outer cell is a right hexagonal prism with specular reflective boundary condi-
 410 tions. This test uses two temperatures simultaneously and tests the reflective
 411 boundary condition setting. The geometry and materials are outlined in Ta-
 412 ble 6, where “r” shown for the hex prism is the length of the apothem. The
 413 FLiBe cross sections used were processed at 900 K and the pebble cross
 414 sections at 1200 K.

Figure 6: Horizontal slice of the geometry of the homogenized fuel pebble in FLiBe test case. Colors represent cell numbers and black represents space outside the outermost boundary.

Table 6: Geometry and materials used in the pebble in molten FLiBe test case.

Cells	Isotopes (Atm. %)	Densities
1 sphere, r=5.0 cm	^{238}U (0.90) ^{235}U (0.10) ^{16}O (2.00) ^{12}C (1.978) ^{13}C (0.022)	8.75 g/cm ³
1 right hex prism, r=5.1 cm	^6Li (0.15) ^7Li (1.85) ^9Be (1.00) ^{19}F (4.00)	1.94 g/cm ³

415 4.5. Test 5 - Stainless Steel Clad Metallic Uranium Pin in Liquid Sodium

416 The criticality test in Figure 7 consists of a single cylinder of metallic ura-
417 nium surrounded by molten sodium. The outer cell is a right hexagonal prism
418 with specular reflective boundary conditions. This test serves to benchmark
419 a fast system with a coolant as well as introduce more and different isotopes.
420 The geometry and materials are outlined in Table 7. The fuel and clad cross
421 sections used were processed at 900 K, and the sodium coolant cross sections
422 were processed at 600 K. In order to keep the specification tenable, only the
423 major components of 316 stainless steel were included in the clad material.

Figure 7: Horizontal slice of the geometry of the stainless steel clad metallic uranium pin in liquid sodium test case. Colors represent cell numbers and black represents space outside the outermost boundary. The material inside the pin cells is the same (UO_2 and Zr), but in this plot the colors represent cell numbers, and therefore each pin has a slightly different color.

Table 7: Geometry and materials used in the 316 stainless steel clad metallic uranium single pin test case.

Cells	Isotopes (Atm. %)	Densities
1 cylinder, r=1.0 cm	^{238}U (0.90) ^{235}U (0.10)	19.1 g/cm ³
1 cylinder, r=1.2 cm	^{54}Fe (0.0435) ^{56}Fe (0.6879) ^{57}Fe (0.0165) ^{58}Fe (0.0021) ^{50}Cr (0.0065) ^{52}Cr (0.1257) ^{53}Cr (0.0143) ^{54}Cr (0.0035) ^{58}Ni (0.0681) ^{60}Ni (0.0262) ^{62}Ni (0.0036) ^{64}Ni (0.0009)	7.99 g/cm ³
1 right hex prism, r=1.8 cm	^{23}Na (1.00)	0.927 g/cm ³

4.6. Test 6 - Zr-Clad Hexagonal UO_2 Pin Cell Lattice in Light Water

This criticality test consists of 631 Zr-clad UO_2 cylinders laid out in a hexagonal lattice surrounded by light water, as shown in Figure 8. The material compositions, densities, and cylinder dimensions are similar to the pin cell test case, but, since this test has two orders of magnitude more objects, it serves to highlight the effect of introducing many geometric objects into the problem and further validates that the geometry processing routines work correctly. The lattice is in the x-y plane, has a pitch to diameter ratio of 1.164, and has 15 elements on a side. The geometry and materials are outlined in Table 8. All cross sections used were processed at 273.5 K.

Table 8: Geometry and materials used in the Zr-clad UO_2 pin hexagonal lattice in light water test case.

Cells	Isotopes (Atm. %)	Densities
631 cylinders, r= 1.0 cm	^{238}U (0.90) ^{235}U (0.10) ^{16}O (2.00)	10.97 g/cm ³
631 cylinders, r= 1.2 cm	^{90}Zr (0.5145) ^{91}Zr (0.1122) ^{92}Zr (0.1715) ^{94}Zr (0.1738) ^{96}Zr (0.0280)	6.52 g/cm ³
1 box, 48x48x48 cm	^1H (2.0) ^{16}O (1.0)	1 g/cm ³

Figure 8: Horizontal slice of the geometry of the Zr-clad hexagonal UO_2 pin cell lattice in light water test case. Colors represent cell numbers and black represents space outside the outermost boundary.

5. Results

The accuracy of the WARP calculations were measured by comparing the multiplication factors and neutron spectra of the aforementioned geometries against those calculated by Serpent 2.1.24 and MCNP 6.1. The multiplication factor differences shown in this section are reported in “per cent mille” (PCM), which is a thousandth of a percent, or 10^{-5} . This is a standard way of reporting differences in the multiplication factor, as any small change in it can cause a significant change in system behavior. The flux spectra are normalized per fission neutron and per unit lethargy. “Lethargy” is defined as the logarithm of the neutron energy, $\ln(E_0/E)$, where E_0 is the highest energy possible in a system and E is another energy [25]. Normalizing the flux per unit lethargy in conjunction with plotting on a logarithmic scale yields a plot where the area under the curve gives the fraction of neutrons flux within the bounding energy range.

The performance of WARP is measured by comparing the runtimes of the WARP simulations against those of the production codes. The runtimes do not include the time it takes to load and process cross sections before the simulations. In other words, it is only the time it takes to complete all transport cycles. The difference in runtimes are reported in speedup factors (t/t_{WARP}). The times reported are either for simulations performed on a single card or a single node. This was done to compare the platforms fairly. Resources are shared at the single card/node level, and therefore increasing

456 process and/or thread count does not necessarily scale linearly as resource
457 contention increases. It can be optimistically assumed that scaling occurs
458 linearly above this level if little inter-card/node communication is needed,
459 and therefore speedup ratios comparing single-node/card performance should
460 be equivalent to multi-node/card performance.

461 It must be noted that WARP uses single-precision data and math, while
462 the production CPU codes both use double-precision data and math. The
463 theoretical throughput of single-precision math is typically double that of
464 double-precision math on modern CPU platforms as well as the Tesla GPU
465 cards [6, 26, 27], but this kind of speedup is very problem- and platform-
466 dependent. For dense linear algebra operations, the factor of two is usually
467 realized, but for sparse multiplications or Monte Carlo simulations where
468 data access is more irregular, a less dramatic speedup is seen by switching to
469 single precision math [28, 29, 30, 31]. This implies that the production codes
470 aren't at a huge disadvantage compared to WARP simply because they must
471 use double-precision math. The codes are complex and overall performance
472 depends on many factors, but single precision math is used in WARP since
473 using it means the consumer-level cards have similar performance parameters
474 as the compute-specific cards, and therefore can provide substantial hardware
475 investment savings. For instance, a Tesla K40 card is theoretically capable
476 of 5040 FLOPs in single precision or 1680 FLOPs in double precision. A
477 GeForce GTX 780 Ti card is theoretically capable of 5046 FLOPs in single
478 precision, but only 210 FLOPs in double precision [32]. When they each were
479 first released, the Tesla cost \$5,500 whereas the GeForce only cost \$700. If
480 used in single-precision mode, the GeForce is the clear winner due to its much
481 lower price and equivalent performance. In contrast, a Titan Black card is
482 theoretically capable of 5121 FLOPs in single precision or 1707 FLOPs in
483 double precision and only cost \$1,000 when it was released. Compared to
484 the K40, it seems the Titan only lacks ECC and RDMA capabilities which
485 could be important for numerically sensitive or communication-intensive cal-
486 culations. In the case of the Titan Black, where double precision performance
487 is similar to the K40, it would be interesting to see the real-world impact of
488 switching to double precision data and math in WARP and could be an area
489 of future investigation.

490 The benchmark cases were run with 6.5×10^6 neutrons per criticality batch
491 with 20 initial batches discarded and 40 batches with statistics accumulated
492 (60 batches total). From the scaling results shown next in this section, the
493 neutron processing rate of the GPU cards all saturated near this number of

neutrons per batch. The CPU codes were run with identical batch parameters. The CPU cases were run with the optimal number of MPI processes and threads per process for the specific node to fairly measure the performance of the unit as a whole. Since the PowerWulf PSSC node contains 4 processors that each contain 2 non-uniform memory access (NUMA) nodes, the most efficient configuration for running the CPU codes was to use 8 MPI processes (one for each NUMA node), which each ran 6 threads for the 6 cores in each NUMA node. The Lenovo Savio2 runs were done in a similar way, but with 4 MPI processes that each ran 6 threads. The MCNP runs were launched with one additional MPI process since the master process does not perform any neutron tracking in MCNP. All codes were set to not use thermal scattering or unresolved resonance tables and to use analog capture only.

5.1. *Scaling*

Larger neutron datasets allow the GPU to schedule threads better, but the size of the dataset is limited by the available on-card memory. Cross sections are also stored on-card and compete with the neutrons for space in memory. It is of interest to know where performance saturates so a user can choose to run the number of neutrons per cycle that gives the best performance. Figure 9 shows the neutron processing rate of WARP compared to the neutron dataset size.

Performance increases until about 5×10^6 neutrons per cycle, then flattens above this for all cards. This curve was calculated using the hexagonal pin lattice test case, and may change if a different model were input. Despite this, 5×10^6 neutrons per cycle is a rough estimate of where maximum performance occurs for WARP, and 6.5×10^6 neutrons per cycle was used for the test cases. There is no data point for the Titan Black at 10^7 neutrons per cycle since a crash always occurred and the simulation could not be completed. The hexagonal pin lattice case was chosen for the scaling study because it is the most complicated geometry considered, and was assumed to be the most sensitive to thread saturation and therefore would require the most neutrons per cycle for the neutron processing rate to level out.

5.2. *Multiplication Factors*

Table 9 shows the multiplication factor deviations for the six criticality tests compared to MCNP 6.1 and Serpent 2.1.24 for WARP running on an NVIDIA Titan Black, K20, and K80 cards. The “ Δ MCNP” rows contains

Figure 9: Neutron processing rate vs. number of neutrons per cycle for the hexagonal lattice test case.

529 WARP's difference (in PCM) from MCNP, and the “ Δ Serpent” rows con-
 530 tains WARP's difference from Serpent. These difference rows also contain 1σ
 531 standard deviation values in terms of relative error. The standard deviation
 532 values indicate the level at which the multiplication factors calculated by
 533 WARP agree with either Serpent or MCNP, and have been used to produce
 534 the “y” and “n” flags shown after the standard deviation value. A “y” in the
 535 1σ position indicates that the multiplication factor values are mutually within
 536 1 standard deviation of each other, i.e. that $\Delta\text{MCNP} < \sigma_{\text{MCNP}} + \sigma_{\text{WARP}}$. In
 537 other words, this is the space where the estimated distributions of the two
 538 codes overlap and is the interval where they could produce the same results.
 539 For one standard deviation, there is a 68% chance that the next calculated
 540 value will be within a single standard deviation, a 32% chance it will be out-
 541 side, and a 16% chance it will be outside and on a single side of the mean.
 542 Therefore, there is a $0.16^2 = 2.6\%$ probability that the codes could produce
 543 values that are outside a the sum of their standard deviations. Outside two
 544 standard deviation is even more unlikely (.05%), so any results that are out-
 545 side two mutual standard deviations are most likely not statistically similar
 546 at all. The results are presented in this way to incorporate the uncertainties
 547 of both codes into the results, and to give an easy way of seeing which test
 548 cases are problematic. If a problem is within a single mutual standard de-

549 viation, it is assumed the solutions are equivalent. If the results are outside
550 two mutual standard deviations, it is assumed the solutions are different.

Table 9: The test case multiplication factors for WARP versus those of Serpent 2.1.24 and MCNP 6.1.

	Jezebel	Homogenized Block	Pin Cell
Serpent 2.1.24	0.9997840 $\pm$ 9.50E-5	0.5934090 $\pm$ 1.20E-4	0.2750510 $\pm$ 1.80E-4
MCNP 6.1	0.9999600 $\pm$ 7.00E-5	0.5933000 $\pm$ 4.00E-5	0.2751500 $\pm$ 7.00E-5
WARP Titan Black	1.0000620 $\pm$ 9.58E-5	0.5933724 $\pm$ 1.24E-4	0.2750294 $\pm$ 1.46E-4
Δ Serpent	27.80, 1 σ n, 2 σ y	-3.66, 1 σ y, 2 σ y	-2.16, 1 σ y, 2 σ y
Δ MCNP	10.20, 1 σ y, 2 σ y	7.24, 1 σ y, 2 σ y	-14.06, 1 σ n, 2 σ n
WARP K20	1.0000212 $\pm$ 7.82E-5	0.5934539 $\pm$ 1.66E-4	0.2751048 $\pm$ 2.07E-4
Δ Serpent	23.72, 1 σ n, 2 σ y	4.49, 1 σ y, 2 σ y	5.38, 1 σ y, 2 σ y
Δ MCNP	6.12, 1 σ y, 2 σ y	15.39, 1 σ n, 2 σ y	-6.52, 1 σ y, 2 σ y
WARP K80	0.9999602 $\pm$ 9.58E-5	0.5932266 $\pm$ 1.24E-4	0.2749884 $\pm$ 1.56E-4
Δ Serpent	17.62, 1 σ y, 2 σ y	-18.24, 1 σ n, 2 σ y	-6.36, 1 σ y, 2 σ y
Δ MCNP	0.02, 1 σ y, 2 σ y	-7.34, 1 σ y, 2 σ y	-18.26, 1 σ n, 2 σ n

	FLiBe Cell	Sodium Pin Cell	Hex Assembly
Serpent 2.1.24	0.8804900 $\pm$ 8.70E-5	1.0987100 $\pm$ 2.20E-4	1.0503300 $\pm$ 7.20E-5
MCNP 6.1	0.8805100 $\pm$ 6.00E-5	1.0987700 $\pm$ 6.00E-5	1.0506500 $\pm$ 9.00E-5
WARP Titan Black	0.8807005 $\pm$ 9.58E-5	1.0986860 $\pm$ 7.82E-5	1.0510795 $\pm$ 9.58E-5
Δ Serpent	21.05, 1 σ n, 2 σ y	-2.40, 1 σ y, 2 σ y	74.95, 1 σ n, 2 σ n
Δ MCNP	19.05, 1 σ n, 2 σ y	-8.40, 1 σ y, 2 σ y	42.95, 1 σ n, 2 σ n
WARP K20	0.8807086 $\pm$ 9.58E-5	1.0987912 $\pm$ 5.53E-5	1.0510620 $\pm$ 9.58E-5
Δ Serpent	21.86, 1 σ n, 2 σ y	8.12, 1 σ y, 2 σ y	73.20, 1 σ n, 2 σ n
Δ MCNP	19.86, 1 σ n, 2 σ y	2.12, 1 σ y, 2 σ y	41.20, 1 σ n, 2 σ n
WARP K80	0.8807546 $\pm$ 9.58E-5	1.0985898 $\pm$ 5.53E-5	1.0511035 $\pm$ 1.24E-4
Δ Serpent	26.56, 1 σ n, 2 σ y	-12.02, 1 σ y, 2 σ y	77.35, 1 σ n, 2 σ n
Δ MCNP	24.56, 1 σ n, 2 σ y	-18.02, 1 σ y, 2 σ y	45.35, 1 σ n, 2 σ n

551 From Table 9, it can be seen that WARP agrees with either MCNP *or*
552 Serpent (they sometimes do not exactly agree with each other!) at the 1 σ
553 level for all cases except the FLiBe cell and the hex assembly cases. The
554 FLiBe cell case agrees on the 2 σ level, but the hex assembly case does not,
555 indicating that there is a problem with the way WARP is solving the hex
556 assembly case. Speculation on what could be causing this is discussed in
557 Section 6.

558 5.3. Runtimes

559 Table 10 shows the transport cycle runtimes for the six criticality tests
560 compared to MCNP 6.1 and Serpent 2.1.24 for WARP running on an NVIDIA
561 Titan Black, K20, and K80 cards. The values in the time rows are reported in
562 minutes. The “ Δ MCNP” row contains WARP’s speedup factor over MCNP,
563 and the “ Δ Serpent” row contains WARP’s speedup factor over Serpent.

564 It can be seen from Table 10 that WARP runs fastest on the K80 card,
565 and both Serpent and MCNP run fastest on the Savio2 node. This makes

Table 10: The test case transport cycle run times (in minutes) for WARP, Serpent 2.1.24, and MCNP 6.1. Values in the “ Δ ” rows contain the speedup factor of WARP compared the the corresponding production code running on a Berkelium (Bk) PSSC node and a Savio2 node.

		Jezebel	Homogenized Block	Pin Cell
Serpent 2.1.24	Bk PSSC	7.24	39.41	146.15
	Savio2	6.32	24.10	76.51
MCNP 6.1	Bk PSSC	30.57	98.40	131.27
	Savio2	4.95	44.03	64.18
WARP Titan Black		1.47	7.53	26.47
Δ Serpent (Bk, S2)		4.91, 4.29	5.23, 3.20	5.52, 2.89
Δ MCNP (Bk, S2)		20.74, 3.36	13.07, 5.85	4.96, 2.42
WARP K20		2.45	12.60	42.55
Δ Serpent (Bk, S2)		2.96, 2.58	3.13, 1.91	3.44, 1.80
Δ MCNP (Bk, S2)		12.49, 2.02	7.81, 3.50	3.09, 1.51
WARP K80		1.18	5.79	20.91
Δ Serpent (Bk, S2)		6.11, 5.34	6.81, 4.17	6.99, 3.66
Δ MCNP (Bk, S2)		25.80, 4.18	17.01, 7.61	6.28, 3.07
		FLiBe Cell	Sodium Pin Cell	Hex Assembly
Serpent 2.1.24	Bk PSSC	79.16	120.86	81.54
	Savio2	45.08	68.46	44.32
MCNP 6.1	Bk PSSC	269.47	320.40	160.68
	Savio2	81.03	199.15	112.25
WARP Titan Black		25.78	81.18	34.91
Δ Serpent (Bk, S2)		3.07, 1.75	1.49, 0.84	2.34, 1.27
Δ MCNP (Bk, S2)		10.45, 3.14	3.95, 2.45	4.60, 3.22
WARP K20		39.83	121.34	56.03
Δ Serpent (Bk, S2)		1.99, 1.13	1.00, 0.56	1.46, 0.79
Δ MCNP (Bk, S2)		6.77, 2.03	2.64, 1.64	2.87, 2.00
WARP K80		24.68	81.40	35.63
Δ Serpent (Bk, S2)		3.21, 1.83	1.48, 0.84	2.29, 1.24
Δ MCNP (Bk, S2)		10.92, 3.28	3.94, 2.45	4.51, 3.15

566 sense since these are the two newest pieces of hardware considered. It is
567 important to note that the Titan Black card performs almost as well as the
568 K80 card, however. Comparing the K80 card to a Savio2 node (the most
569 relevant comparison), WARP runs 0.84 to 5.34 times as fast as Serpent and
570 2.45 to 7.61 times as fast as MCNP. The largest speedup over Serpent is
571 for the Jezebel test, the largest speedup over MCNP is for the homogenized
572 block test, and the smallest speedup is for the sodium pin cell test over both
573 Serpent and MCNP. This speedup factor is calculated over an entire compute
574 node, so WARP on a GPU gives a performance equivalent to 0.84 to 7.61
575 compute nodes, depending on the type of simulation hardware considered.
576 For the sodium pin case, where there is a relatively large amount of cross
577 section processing, a Savio2 node running Serpent actually performs better
578 than WARP on a K80 card. For problems where there is less cross section

579 processing (higher geometry processing fraction), WARP tends to perform
580 better than a Savio2 node running Serpent. WARP always performs better
581 than a Savio2 node running MCNP, even with relatively heavy cross section
582 processing, but this isn't a great surprise considering that MCNP doesn't use
583 a unionized energy grid structure when processing cross sections.

584 5.4. *Spectra*

585 In the following subsections, neutron flux spectra calculated by WARP are
586 compared to those calculated by Serpent and MCNP for identical geometries
587 and materials. Every spectrum is binned into 1024 equi-log bins from 10^{-11}
588 to 20 MeV. In tests where there is a large region of little to no flux (like in
589 the fast spectrum inside the Jezebel sphere), the energy bin structure is not
590 changed, but the energy range of the plot is limited to the region where there
591 is nonzero flux. The relative difference compared to the MCNP spectrum is
592 shown in the top subplot below the main spectrum plot, and the relative
593 difference from the Serpent spectrum is shown in the lower subplot. The
594 green shaded area in the error subplots shows the space within 2 standard
595 deviations of the statistical uncertainty of the production code. Unlike the
596 intervals reported in the previous subsection, the interval shown in green is
597 only for the production code results. In other words, the codes should have
598 results within the 2σ interval 95.45% of the time (as opposed to 99.95% of
599 the time for two mutual standard deviations).

600 5.4.1. Test 1 - “Jezebel” Bare Pu Sphere

601 Figure 10 shows the volume-averaged flux spectra in the Jezebel sphere.
 602 The relative difference of WARP compared to Serpent and MCNP is very low,
 603 with the normalized tally bins being less than 0.5% from each other in regions
 604 where the flux is large. Of course, when the flux is small, the statistical
 605 uncertainty becomes much higher, and the relative difference becomes noisy.
 606 The relative error is almost always inside the 2σ confidence interval as well,
 607 indicating that the simulations are statistically identical.

Figure 10: Volume-averaged flux spectra inside the sphere of the Jezebel test case.

5.4.2. Test 2 - Homogenized Fuel Block

Figure 11 shows the volume-averaged flux spectra in the homogenized fuel block. The relative difference compared to MCNP and Serpent is again shown in the lower subplots. The relative difference compared to MCNP is again generally less than 0.15% and appears to have a zero mean, but the error compared to Serpent has a constant positive offset, indicating a slightly different normalization. This kind of offset appears in other spectra as well, but only when comparing to Serpent. This indicates a possible normalization discrepancy between MCNP and Serpent. There also appears to be some deviations with similar structure from 0.1 to 0.3 MeV, indicating that some kind of reaction sampling may not be treated exactly the same way as in Serpent and MCNP.

Figure 11: Volume-averaged flux spectra inside the homogenized fuel block.

5.4.3. Test 3 - Zr-Clad UO_2 Pin in Heavy Water

Figure 12 shows the volume-averaged flux spectra inside the fuel rod in the Zr-clad UO_2 pin in the heavy water test case. Again, the relative difference is generally less than 1% where the flux is large. Compared to Serpent, the WARP spectrum again appears to have a small, constant offset. The entire energy range agrees well with MCNP, however, with the relative error is almost always inside the 2σ confidence interval. This indicates that all the reaction laws were processed identically to MCNP for the nuclides present in this test case. The 2σ confidence interval is larger here than the previous cases since the tally volume is much smaller than previously and it simply has fewer scores per source particle.

Figure 12: Volume-averaged flux spectra inside the oxide fuel of the Zr-clad pin in the pin in heavy water test case.

5.4.4. Test 4 - Homogenized Fuel Pebble in FLiBe

Figure 13 shows the volume-averaged flux spectra in the fuel pebble for the reflective FLiBe test case. Here, the relative difference is generally less than 0.1% where the flux is large. The WARP spectrum again has a constant offset compared to the Serpent spectrum, but compares well to MCNP apart from the significant deviations around large resonances in the 20 keV to 1 MeV range.

Figure 13: Volume-averaged flux spectra inside the fuel pebble of the reflective FLiBe test case.

5.4.5. Test 5 - Stainless Steel Clad Metallic Uranium Pin in Liquid Sodium

Figure 14 shows the volume-averaged flux spectra in the fuel for the reflective sodium cooled, steel clad, metallic fuel test case. The relative difference is generally less than 0.1% where the flux is large. The WARP spectrum again has a constant offset compared to the Serpent spectrum, but compares well to MCNP apart from the significant deviations around large resonances in the 20 keV to 1 MeV range. This again highlights that there is probably a reaction law that isn't processed identically to MCNP.

Figure 14: Volume-averaged flux spectra inside the fuel of the reflective, steel clad, sodium cooled pin test case.

5.4.6. Test 6 - Zr-Clad Hexagonal UO_2 Pin Cell Lattice in Light Water

Figure 15 shows the volume-averaged flux spectrum in the center pin of the Zr-clad hexagonal oxide fuel pin lattice in light water test case. The relative difference is generally less than 5% where the flux is large, but this time the WARP results are always within the statistical error of both MCNP and Serpent. There is no constant offset compared to Serpent or MCNP and no large deviations, indicating that all the reaction laws were processed identically to both Serpent and MCNP for the nuclides present in this test case. This is also the only test case other than Jezebel where MCNP and Serpent give identical spectra. Again the 2σ confidence interval is large compared to other test problems because the tally volume is small compared to the overall geometry.

Figure 15: Volume-averaged flux spectra inside the center fuel pin of the hexagonal pin lattice test case.

5.5. History Power

For productivity, the “history power,” or number of neutron histories processed per unit time, is of primary interest when purchasing a system since it determines how quickly results can be produced. In addition to this, the history power per hardware cost is of interest for initial capital investment, and the history power per unit energy is of interest for operational costs. Table 11 shows the history power and relative costs of GPU systems running WARP compared to MCNP and Serpent running on the aforementioned CPU platforms.

Table 11: The history power and costs of performing Monte Carlo neutron transport on GPU and CPU systems.

		H. Power	H.P. / Capital (USD)	H.P. / Watt
Serpent 2.1.24	Bk PSSC	2.32E+6	249	5,033
	Savio2	8.84E+6	1,305	42,082
MCNP 6.1	Bk PSSC	2.32E+6	249	5,033
	Savio2	4.63E+6	684	22,039
WARP	Titan Black	1.32E+7	8,675	52,778
	K20	8.52E+6	2,561	37,847
	K80	1.38E+7	2,760	45,995

The history powers shown in Table 11 were calculated by summing the total number of neutrons processed in all six test cases and dividing this number by the sum of the runtimes of the six cases for each code/platform combination (i.e. 3.9×10^8 /sum of run times). This way, the average is not biased towards a special case that allows a code to process neutrons especially quickly. The history power per capital is simply the averaged neutron processing rate divided by the hardware cost of the platform. The history power per Watt is similar, except divided by the platform’s total maximum electrical power consumption rather than the cost. The costs and electrical power consumptions used to calculate these values were shown in Table 2.

Table 11 shows that, on average, for the test cases considered, a new NVIDIA GPU card can process neutrons about 1.6 times as fast as a new multiprocessor/multicore CPU node running Serpent and about 3 times as fast as a CPU node running MCNP. From a capital investment standpoint, a K80 card costs about half as much as a Savio2 node per neutron processing power and consumes about the same amount of electricity. The Titan Black, however, performs about as well as a K80 from the neutron processing rate

standpoint, but costs about a third as much and consumes 20% less electricity. Since WARP uses single-precision data and math, consumer-level GPUs have similar performance compared to compute-specific ones when running WARP. Because of this, the capital and electrical costs of running WARP are greatly reduced compared to a case where double precision data and math are required.

6. Discussion

The differences seen in the multiplication factors and neutron flux spectra calculated by WARP compared to those calculated by Serpent and MCNP are most likely due to either a reaction law being inexactly processed or from a type of reaction incorrectly sampled from time to time. Whether this is due to a software “bug”, or if it has to do with single precision math has not been determined. Some sampling laws potentially require division by small numbers when comparing or adding the result to other numbers. Roundoff error from single precision math could make these operations inaccurate and ultimately lead to visible differences in the neutron spectra and multiplication factor. The sampling routines have been spot tested to produce very similar results to the production code routines given narrow incoming energy bands, but rigorous testing of all energies has not been done. There has been some effort in implementing the “Google Test” testing framework [33] in WARP, and it has been used in older versions to performing unit testing. Since WARP has been in heavy development, creating standard unit tests was difficult since the inputs and outputs of routines were constantly changing. A full testing suite to ensure proper data sampling is an area of future development that can be done once WARP becomes more stable. Until then, WARP is sufficiently accurate to be a good testbed for Monte Carlo neutron transport calculations on GPU platforms.

Monte Carlo is a great candidate simulation method to take advantage of single precision math since solution convergence does not normally depend on derivatives, except perhaps in a cross section processing law, and the calculated results are typically not required to be more precise than single-precision numbers. The major places roundoff error can affect the solution is in tally accumulation and cross section processing. Currently, WARP calculates the total macroscopic cross section of a material on the fly, and roundoff could become a problem around resonances where the cross section of one nuclide (or several) is much larger than the that of the other nuclides

720 in the material. In this case, the additions from the minor cross section
721 nuclides could be neglected and could lead to oversampling their reactions
722 in the subsequent routines (since the material's calculated total macroscopic
723 cross section would be smaller than it should be). This problem could be
724 circumvented by preprocessing the material's total macroscopic cross section
725 using double precision math on the host before the data is given to the
726 GPU as single precision numbers. This, however, would increase the memory
727 footprint of the data.

728 Another place roundoff error could cause problems is in tally accumula-
729 tion. WARP calculates the flux in a cell by calculating the inverse of the
730 macroscopic cross section at a collision site and adding it to the sum of all
731 previous collisions. If a large resonance lies within an energy bin, very dis-
732 similar values could be added together and precision could be lost. This
733 could be circumvented by using double precision data for the tally arrays.
734 The cost of doing so would have to be measured, but hopefully it would be
735 small since tally accumulation only happens once per transport cycle and is
736 therefore a relatively infrequent operation.

737 WARP uses integer math to calculate multiplication factors. Long integer
738 values for the total number of fissions produced in a simulation are stored on
739 the host, and integer yield values from each batch are accumulated into it. At
740 the end of the simulation, the long integer value containing the total number
741 of fission neutrons produced is divided by the total number of source neutrons
742 run, yielding the final value for the multiplication factor. Since integer math
743 is used during this whole process, roundoff error from accumulation can not
744 affect it as long as overflow doesn't occur (hence using the long integer data
745 type). Roundoff error in the reaction selection and processing routines could
746 affect the individual yield values and their frequency, however.

747 It is worth noting that MCNP and Serpent calculated somewhat different
748 solutions in these studies, which shows that getting exactly the same answer is
749 quite difficult since each code handles things slightly differently. This makes
750 it difficult to know what is really correct, but as long as the results are under
751 the error level of what can be detected in the physical world, any differences in
752 the codes should be irrelevant. One potential niche use for WARP would be
753 to do cheap and fast calculations where large total throughput is needed, not
754 precision. For example, in a large parametric study, once an optimal region
755 has been found in the parameter space, WARP's job would be done. This
756 knowledge could then be used as a starting point for more precise calculations
757 with CPU codes.

758 7. Conclusions and Future Development

759 It has been shown that WARP is able to accelerate high-fidelity neutron
760 transport on GPUs. Depending on the problem type and the card used,
761 WARP can achieve performance equivalent to that of 0.84 to 7.61 of modern
762 CPU nodes on a single GPU card, depending on the CPU code and type
763 of calculation being considered. The low capital and electrical cost of GPU
764 platforms then lead to about 2 to 3 times more histories calculated per dollar
765 spent. This clearly shows that GPUs can be very cost effective for continuous
766 energy Monte Carlo neutron transport. This efficiency comes at the cost
767 of precision, however, but WARP could still be a useful tool for scenarios
768 where high computational throughput and/or low-cost calculations are of
769 importance.

770 WARP, which is developmental, may still have some “bugs”, as evidenced
771 by the slight deviations of the multiplication factors and spectra compared to
772 MCNP and Serpent. A large part of future development will be eliminating
773 these discrepancies via rigorous testing. The initial goals of WARP have
774 been completed, but there is much work still to be done if it is going to
775 be of real use to the nuclear engineering community. Basic functionality
776 is currently good enough to assure that GPUs can accelerate high-fidelity
777 Monte Carlo neutron transport calculations, but many capabilities need to
778 be expanded and ensured to scale well to large numbers of neutrons, isotopes,
779 and geometric zones. The rest of this section goes over the major areas where
780 WARP could be improved in order to ensure good scaling in these dimensions.

781 Geometry is currently handled by NVIDIA OptiX, which provides a con-
782 venient way to obtain high-performance results. However, OptiX had to be
783 coerced into providing WARP with the extra information it needed beyond
784 the distance to the nearest intersection, namely the material and tally in-
785 dices. The way OptiX is used to determine these numbers is not efficient
786 since it must be done iteratively using OptiX’s native functions instead of
787 calculating the sum of surface normals in a single trace. NVIDIA has released
788 “OptiX Prime” with OptiX 3.5 [34], which promises to provide a more “to
789 the metal” ray tracing experience, and might be leveraged to provide more
790 efficient single-traverse functionality. OptiX could also be replaced by Ray-
791 force [35], a high-performance GPU ray tracing library developed by VSL
792 that has the desired functionality built-in and is currently available free of
793 charge for noncommercial use.

794 The OptiX geometry routines could also be replaced by handwritten rou-

795 tines that use combinatorial solid geometry like Serpent and MCNP. This
 796 would make writing input for WARP more like what most nuclear engineers
 797 are already used to and, more importantly, could provide a potential perfor-
 798 mance increase. A universe-based CSG representation may map very well
 799 to the GPU and may even be able to fit inside of shared memory for small
 800 numbers of surfaces. Using an efficient CSG method would further lend itself
 801 to using Woodcock delta-tracking for the neutrons and thus getting rid of
 802 the tracing algorithms and libraries altogether [36].

803 WARP also suffers from “tail effect,” where performance drops off as the
 804 number of remaining active neutrons in a batch decreases [1]. Developing a
 805 method that keeps the active neutron number high would greatly improve
 806 performance, but could become quite complicated as generations would start
 807 to overlap and calculation of the multiplication factor and converging the
 808 source distribution could not be done between cycles as it is now.

809 If an entire overhaul of the WARP transport algorithm is feasible, using
 810 a streaming multiprocessor (SM)-based algorithm might be investigated for
 811 replacing the current global one. This type of algorithm would treat each SM
 812 as an independent processor and would provide each a bank of neutrons to
 813 transport, as is done by Liu and Henderson [37, 38]. This way, neutron data
 814 could be stored in very fast shared memory, though this would compete with
 815 storing geometric information. Also, since a smaller set of neutrons could
 816 be stored, the SMs would need to communicate to determine which of the
 817 next neutrons they would take out of the global bank, or they would need
 818 to periodically rendezvous to shared source information and ensure that the
 819 distributions they use are each converged. This type of transport algorithm
 820 would also preclude using OptiX, since it does not have SM-level functionality
 821 [22].

822 An efficient way to handle situations where there are many different mate-
 823 rials and isotopes present needs to be explored. The work done by Scudiero
 824 on porting OpenMC’s macroscopic cross section processing benchmarking
 825 tool, “xsbench,” may elucidate this endeavor [21, 39]. Since global memory
 826 comes at a premium on GPUs, an on-the-fly temperature treatment for nu-
 827 clides would likely be required if more than a handful of isotopes are desired
 828 at more than one temperature. Methods like those used in Serpent could be
 829 adapted for use on the GPU [3]. On-the-fly methods reduce the amount of
 830 storage needed, but they require more computation per data element loaded
 831 since the loaded value is adjusted according to the temperature of the ma-
 832 terial. This kind of method may work well on the GPU since GPUs have a

833 larger FLOP/byte ratio than CPUs and the additional work may cost little.
 834 A grid thinning routine that discards closely-spaced energy grid points could
 835 also be added to WARP, saving memory but potentially costing accuracy.
 836 Fractional cascading may also help reduce the memory footprint of the cross
 837 section data while keeping energy search cost low [40].

838 WARP would gain usability if more features were incorporated as well.
 839 Importance cutoffs could be used to terminate neutrons, leading to shorter
 840 runtimes; cell importances (easily attached to cell primitives), track length
 841 tallies, and implicit absorption could help improve tally statistics. Develop-
 842 ing an efficient way to include many reaction rate tallies would also make
 843 WARP useful for performing depletion analysis. It would further be help-
 844 ful for WARP to have statistical tests like Shannon entropy to ensure the
 845 fission source is fully converged before tallies and multiplication factors are
 846 accumulated [41]. Accuracy would also be improved by adding $S(\alpha, \beta)$ ther-
 847 mal scattering tables and unresolved resonance parameters. Neutrons have
 848 statistical weight in WARP, so simple variance reduction techniques like im-
 849 plicit capture should be a straight-forward part of future development. Path
 850 lengths are already calculated in OptiX, so implementing track length tallies
 851 could also be a straight-forward variance reduction technique to introduce
 852 into WARP. Multi-GPU support should also be added so that WARP can
 853 be used effectively on computers with more than one GPU.

854 In conclusion, the initial development of WARP has been completed. It
 855 is the first code able to perform Monte Carlo neutron transport calculations
 856 in general 3D geometries entirely on GPUs using continuous energy, ACE-
 857 formatted cross sections. Using high-end consumer graphics cards, WARP
 858 can perform these calculations in very cost-effective way in a timeframe com-
 859 parable to using multiple supercomputer nodes. The results calculated by
 860 WARP compare well with production codes in many cases, and are at least
 861 accurate enough to provide a realistic testbed for further GPU development
 862 in the field of Monte Carlo neutron transport, or provide fast precalculations
 863 for more accurate production code runs.

864 WARP will be released as open source software, pending DOE approval,
 865 at <http://github.com/weft>. In a weave of cloth, The “warp” is the high-
 866 tension, longitudinal thread, whereas the “weft” is the low tension, transverse
 867 thread. On a loom, the warp threads are all moved in sync, then the weft
 868 is passed through them. Naming the repository for the single thread that
 869 defines the phase of the warp seemed appropriate.

870 Acknowledgements

871 This research is based upon work partially supported by the U.S. De-
872 partment of Energy National Nuclear Security Administration under Award
873 Number DENA0000979 through the Nuclear Science and Security Consor-
874 tium: <http://nssc.berkeley.edu>, as well as upon work supported under an
875 Integrated University Program Graduate Fellowship. This research also used
876 the Savio computational cluster resource provided by the Berkeley Research
877 Computing program at the University of California, Berkeley (supported by
878 the UC Berkeley Chancellor, Vice Chancellor of Research, and Office of the
879 CIO).

880 Disclaimer

881 This report was prepared as an account of work sponsored by an agency
882 of the United States Government. Neither the United States Government
883 nor any agency thereof, nor any of their employees, makes any warranty,
884 express or limited, or assumes any legal liability or responsibility for the
885 accuracy, completeness, or usefulness of any information, apparatus, product,
886 or process disclosed, or represents that its use would not infringe privately
887 owned rights. Reference herein to any specific commercial product, process,
888 or service by trade name, trademark, manufacturer, or otherwise does not
889 necessarily constitute or imply its endorsement, recommendation, or favoring
890 by the United States Government or any agency thereof. The views and
891 opinions of authors expressed herein do not necessarily state or reflect those
892 of the United States Government or any agency thereof.

893 References

- 894 [1] R. M. Bergmann, J. L. Vujić, Algorithmic choices in WARP—a frame-
895 work for continuous energy monte carlo neutron transport in general 3D
896 geometries on GPUs, *Annals of Nuclear Energy* 77 (2015) 176–193.
- 897 [2] J. Leppänen, Development of a New Monte Carlo Reactor Physics Code,
898 2007. D.Sc. Dissertation.
- 899 [3] Serpent website, <http://montecarlo.vtt.fi/>, 2014.

- 900 [4] D. B. Pelowitz, J. T. Goorley, M. R. James, T. E. Booth, F. B. Brown,
901 J. S. Bull, L. J. Cox, J. W. Durkee, J. S. Elson, M. L. Fensin, R. A.
902 Forster, J. S. Hendricks, H. G. Hughes, R. C. Johns, B. C. Kiedrowski,
903 R. L. Martz, S. G. Mashnik, G. W. McKinney, R. E. Prael, J. E. Sweezy,
904 L. S. Waters, T. A. Wilcox, A. Zukaitis, MCNP6 USERS MANUAL
905 VERSION 1.0, Los Alamos National Laboratory,, Los Alamos, NM,
906 rev. 0 edition, 2013.
- 907 [5] Y. Lee, R. Avizienis, A. Bishara, R. Xia, D. Lockhart, C. Batten,
908 K. Asanović, Exploring the tradeoffs between programmability and ef-
909 ficiency in data-parallel accelerators, SIGARCH Comput. Archit. News
910 39 (2011) 129–140.
- 911 [6] CUDA C Programming Guide, NVIDIA Co, Santa Clara, CA, pg-02829-
912 001_v5.5 edition, 2013.
- 913 [7] J. Ruggiero, Measuring Cache and Memory Latency and CPU to Mem-
914 ory Bandwidth, Technical Report, Intel Co., 2008.
- 915 [8] B. Catanzaro, An Introduction to CUDA/OpenCL and Manycore
916 Graphics Processors, Lecture slides, 2011. University of California,
917 Berkeley CS267 - Applications of Parallel Computers.
- 918 [9] The AMD Opteron processor is the ideal solution for High Performance
919 Computing, Technical Report, Advanced Micro Devices, Inc., 2011.
- 920 [10] F. B. Brown, W. R. Martin, Monte Carlo methods for radiation trans-
921 port analysis on vector computers, Progress in Nuclear Energy 14 (1984)
922 269 – 299.
- 923 [11] J. L. Vujic, W. R. Martin, Vectorization and parallelization of a pro-
924 duction reactor assembly code, Progress in Nuclear Energy 26 (1991)
925 147 – 162.
- 926 [12] E. Z. Zhang, Y. Jiang, Z. Guo, X. Shen, On-the-fly elimination of
927 dynamic irregularities for GPU computing, SIGPLAN Not. 46 (2011)
928 369–38.
- 929 [13] M. Harris, S. Sengupta, J. D. Owens, Parallel Prefix Sum (Scan) with
930 CUDA, in: H. Nguyen (Ed.), GPU Gems 3, Addison Wesley, 2007, pp.
931 851 – 876.

- 932 [14] Python Language Reference, Python Software Foundation, 2.7 edition,
933 2016. <http://www.python.org>.
- 934 [15] D. M. Beazley, Automated Scientific Software Scripting with SWIG,
935 Future Gener. Comput. Syst. 19 (2003) 599–609.
- 936 [16] M. Chadwick, M. Herman, P. Obloinsk, M. Dunn, Y. Danon, A. Kahler,
937 D. Smith, B. Pritychenko, G. Arbanas, R. Arcilla, R. Brewer, D. Brown,
938 R. Capote, A. Carlson, Y. Cho, H. Derrien, K. Guber, G. Hale, S. Hoblit,
939 S. Holloway, T. Johnson, T. Kawano, B. Kiedrowski, H. Kim, S. Ku-
940 nieda, N. Larson, L. Leal, J. Lestone, R. Little, E. McCutchan, R. Mac-
941 Farlane, M. MacInnes, C. Mattoon, R. McKnight, S. Mughabghab,
942 G. Nobre, G. Palmiotti, A. Palumbo, M. Pigni, V. Pronyaev, R. Sayer,
943 A. Sonzogni, N. Summers, P. Talou, I. Thompson, A. Trkov, R. Vogt,
944 S. van der Marck, A. Wallner, M. White, D. Wiarda, P. Young,
945 ENDF/B-VII.1 Nuclear Data for Science and Technology: Cross Sec-
946 tions, Covariances, Fission Product Yields and Decay Data, Nuclear
947 Data Sheets 112 (2011) 2887 – 2996.
- 948 [17] L. A. N. Laboratory, <http://t2.lanl.gov/nis/endl/>, 1998. An Intro-
949 duction to the ENDF Formats.
- 950 [18] A. Scopatz, S. Johnson, P. Romano, P. Wilson, N. Touran, K. Huff,
951 C. Dembia, E. Relson, E. Biondo, M. Gidden, C. Bates, K. Manalo,
952 PyNE: The Nuclear Engineering Toolkit, pyne.io, 2014.
- 953 [19] S. van der Walt, S. C. Colbert, G. Varoquaux, The NumPy Array: A
954 Structure for Efficient Numerical Computation, Computing in Science
955 Engineering 13 (2011) 22–30.
- 956 [20] X.-. M. C. Team, MCNP - A General Monte Carlo N-Particle Transport
957 Code, Version 5, Los Alamos National Laboratory,, Los Alamos, NM,
958 volume i: overview and theory edition, 2003. (Revised 2/1/2008).
- 959 [21] OpenMC Documentation, Massachusetts Institute of Tech-
960 nology, Cambridge, MA, 0.5.3 edition, 2013. [Http://mit-
961 crpg.github.io/openmc/index.html](http://mit-crpg.github.io/openmc/index.html).
- 962 [22] OptiX Programming Guide, NVIDIA Co, Santa Clara, CA, v3.0 edition,
963 Nov. 2012.

- 964 [23] RSICC website, [https://rsicc.ornl.gov/codes/dlc/dlc2/](https://rsicc.ornl.gov/codes/dlc/dlc2/dlc-249.html)
965 [dlc-249.html](https://rsicc.ornl.gov/codes/dlc/dlc2/dlc-249.html), 2016.
- 966 [24] O. N. E. Agency, International Handbook of Evaluated Criticality Safety
967 Benchmark Experiments, Nuclear Energy Agency, OECD, 1995.
- 968 [25] J. J. Duderstadt, L. J. Hamilton, Nuclear Reactor Analysis, John Wiley
969 & Sons, Inc., New York, NY, 1976.
- 970 [26] J. Langou, J. Langou, P. Luszczek, J. Kurzak, A. Buttari, J. Dongarra,
971 Exploiting the performance of 32 bit floating point arithmetic in obtain-
972 ing 64 bit accuracy (revisiting iterative refinement for linear systems),
973 in: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing,
974 SC '06, ACM, New York, NY, USA, 2006.
- 975 [27] L. Itu, C. Suci, F. Moldoveanu, A. Postelnicu, Comparison of single and
976 double floating point precision performance for Tesla architecture GPUs,
977 Bulletin of the Transilvania University of Br ov Series I: Engineering
978 Sciences 4 (2011).
- 979 [28] LAMMPS Benchmarks, <http://lammps.sandia.gov/bench.html>, ac-
980 cessed 10/2016.
- 981 [29] Roy Longbottoms PC Benchmarks Colleciton, Linux PC benchmarks,
982 <http://www.roylongbottom.org.uk/linux%20benchmarks.htm>, ac-
983 cessed 10/2016.
- 984 [30] A. Buttari, J. J. Dongarra, J. Kurzak, P. Luszczek, S. Tomov, Using
985 mixed precision for sparse matrix computations to enhance the perfor-
986 mance while achieving 64-bit accuracy, ACM Trans. Math. Softw. 34
987 (2008).
- 988 [31] Y. Chan, Benchmarks for Intel MIC Architecture, [http://www.](http://www.clustertech.com/wp-content/uploads/2014/01/MICBenchmark.pdf)
989 [clustertech.com/wp-content/uploads/2014/01/MICBenchmark.](http://www.clustertech.com/wp-content/uploads/2014/01/MICBenchmark.pdf)
990 [pdf](http://www.clustertech.com/wp-content/uploads/2014/01/MICBenchmark.pdf), 2013.
- 991 [32] List of Nvidia Graphics Processing Units, [https://en.wikipedia.](https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units)
992 [org/wiki/List_of_Nvidia_graphics_processing_units](https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units), accessed
993 01/2017.

- 994 [33] Google Test, <https://github.com/google/googletest>, accessed
995 10/2016.
- 996 [34] OptiX Programming Guide, NVIDIA Co, Santa Clara, CA, v3.5 edition,
997 Nov. 2012.
- 998 [35] C. Gribble, L. A. Butler, Advances in High-Performance GPU Ray
999 Tracing for Physics-Based Simulation, GPU Technology Conference,
1000 San Jose, CA.
- 1001 [36] J. Leppänen, Performance of Woodcock delta-tracking in lattice physics
1002 applications using the Serpent Monte Carlo reactor physics burnup cal-
1003 culation code, *Annals of Nuclear Energy* 37 (2010) 715–722.
- 1004 [37] T. Liu, A. Ding, W. Ji, X. G. Xu, C. D. Carothers, F. B. Brown, A
1005 Monte Carlo Neutron Transport Code For Eigenvalue Calculations on
1006 a Dual-GPU System and CUDA Environment, PHYSOR, Knoxville,
1007 Tennessee.
- 1008 [38] N. Henderson, K. Murakami, K. Amako, M. Asai, T. Aso, A. Dotti,
1009 A. Kimura, M. Gerritsen, H. Kurashige, T. S. J. Perl, A CUDA Monte
1010 Carlo simulator for radiation therapy dosimetry based on Geant4, Su-
1011 percomputing in Nuclear Applications and Monte Carlo (SNA+MC),
1012 Paris, France.
- 1013 [39] T. Scudiero, Monte Carlo Neutron Transport - Simulating Nuclear Re-
1014 actions One Neutron at a Time, GPU Technology Conference 2014, San
1015 Jose, California.
- 1016 [40] A. L. Lund, A. R. Siegel, B. Forget, C. Josey, P. K. Romano, Using
1017 fractional cascading to accelerate cross section lookups in Monte Carlo
1018 neutron transport calculations, in: Proceedings of Joint International
1019 Conference on Mathematics and Computation (M&C), Supercomputing
1020 in Nuclear Applications (SNA), and the Monte Carlo (MC) Method,
1021 Nashville, Tennessee.
- 1022 [41] T. Ueki, F. B. Brown, Stationarity Modeling and Informatics-Based
1023 Diagnostics in Monte Carlo Criticality Calculations, *Nuclear Science
1024 and Engineering* 149 (2005) 38–50.