Monte Carlo Domain Decomposition for Robust Nuclear Reactor Analyses

Nicholas Horelik[a,*], Andrew Siegel[b], Benoit Forget[a], Kord Smith[a]

[a]*Massachusetts Institute of Technology, Department of Nuclear Science and Engineering, 77 Massachusetts Avenue, Building 24, Cambridge, MA 02139, United States*
[b]*Argonne National Laboratory, Mathematics and Computational Sciences (MCS) Division, 9700 Cass Avenue, Argonne, IL 60439, United States*

Abstract

Monte Carlo (MC) neutral particle transport codes are considered the *gold-standard* for nuclear simulations, but they cannot be robustly applied to high-fidelity nuclear reactor analysis without accommodating several terabytes of materials and tally data. While this is not a large amount of aggregate data for a typical high performance computer, MC methods are only embarrassingly parallel when the key data structures are replicated for each processing element, an approach which is likely infeasible on future machines. The present work explores the use of spatial domain decomposition to make full-scale nuclear reactor simulations tractable with Monte Carlo methods, presenting a simple implementation in a production-scale code. Good performance is achieved for mesh-tallies of up to 2.39TB distributed across 512 compute nodes while running a full-core reactor benchmark on the Mira Blue Gene/Q supercomputer at the Argonne National Laboratory. In addition, the effects of load imbalances are explored with an updated performance model that is empirically validated against observed timing results. Several load balancing techniques are also implemented to demonstrate that imbalances can be largely mitigated, including a new and efficient way to distribute extra compute resources across coarse domain meshes.

Keywords: Monte Carlo, domain decomposition, load balancing, neutron transport, nuclear reactor analysis

1. Introduction

Next-generation high-performance-computing (HPC) architectures will increasingly utilize on-node parallelism to achieve improvements in peak FLOP rates at improved power efficiency ([1, 2, 3]). For many applications the increased processor performance will have a signficant impact on the fidelity of the physical models, potentially enabling the simulation of a much broader range of physical phenomenon for signicantly longer timescales or at much higher resolution. However, at the same time the total memory is growing at a slower rate than the aggregate available processing power, so that the amount of memory available to each individual processing unit is descreasing ([4, 5]). This new FLOP/memory balance will leave us in a regime quite distinct from what has become familiar over the past twenty years, requiring in many cases non-trivial adaptations of traditional methods in order to take advantage of the increased parallelism.

One prime example can be found in the field of nuclear reactor physics, where stochastic Monte Carlo (MC) particle transport methods have the potential to be a highly accurate, general-purpose tool for robust for full core reactor simulations ([6]). MC methods can sample directly from evaluated nuclear data to carry out random walks for individual particles through an arbitrarily complex geometry with minimal approximations. They can be used to determine power distributions, dose rates, and other quantities critical to assessing the safety and performance of nuclear systems. However, for certain classes of problems MC methods require immense computational effort to achieve good statistical convergence, making their routine use impractical ([7]). Thus, they have traditionally been relegated to a more limited set of applications, such as benchmarking and validation of lower-fidelity methods, or for the analysis of smaller systems with less complex physics. It is desirable to shift this paradigm, especially with the complexity of next-generation reactor designs presenting new challenges to the existing suite of highly-specialized reactor simulation tools.

While the performance improvements of next-generation systems might overcome the time-to-solution limitations of MC methods, a number of algorithmic challenges stand in the way of harnessing the increased processing power - especially in the presence of reduced-memory enviroments. The root of the problem is that effective parallelization of MC methods has traditionally been carried out by performing parallellization *over particles*, where each processing element tracks a subset of particle histories through (potentially) the entire domain. This approach requires the replication of domain meta-data on each distributed processing element. While this may be a relatively small memory footprint for simple problems, for a robust analysis of power reactors the required data structures are much too large for local memory. We note that this problem is present even on existing leadership class machines, and that a number of simplifications and approximations are made in an attempt to retain the traditional approach and minimize the negative performance impact. On future HPC systems the problem is expected to be exacerbated.

The details of the full-scale reactor problem are discussed in [8]. The basic idea is that a very fine spatial mesh is needed for a robust full-core reactor depletion analysis: several Ter-

*Corresponding author
Email addresses: nhorelik@mit.edu (Nicholas Horelik), siegela@mcs.anl.gov (Andrew Siegel), bforget@mit.edu (Benoit Forget), kord@mit.edu (Kord Smith)

abytes of memory are required to hold reaction rate tallies and isotopic abundances for several hundreds of nuclides in each region. For this problem, the authors of [7, 8] detail the several well-known performance hurdles that must be overcome before MC codes can be used routinely: (i) excessive overall time to solution for adequate statistical convergence; (ii) inadequate memory for reaction rate tallies and material composition data, and (iii) inadequate memory for temperature-dependent nuclear cross section data. Several recent works aim to address issues (i) and (iii) ([9, 10, 11, 12]), but issue (ii) requires additional attention.

Implementing domain decomposition seems like an obvious solution, but the routine use of this approach is made difficult by the large amount of particle communication that would be required between domains (billions of particles for real problems), and the accompanying potential for significant parallel inefficiencies due to load imbalances ([6]). Instead, traditional parallel MC codes employ full domain replication to take advantage of the embarrassingly-parallel nature of neutral particle tracking. With this scheme, all material and tally data is allocated on each distributed computational node ([13, 14]) and synchronized between fission source iterations: a relatively inexpensive operation ([15]). Given on-node memory constraints, it is clear that this parallelization scheme is not capable of tackling reactor problems on its own.

One approach for handling this memory limitation without domain decomposition is data decomposition. For example, the method described in [16] stores tally data on a set of distributed "server" processes, to which tracking processes send tally writes via asynchronous MPI sends. Indeed, reasonable performance was observed in [17] for a variety of typical computer parameters with tallies at a scale relevant to full-core analyses. In principle a similar concept can be applied to the treatment of materials data, but it is not clear what effect this will have on particle tracking rates.

On the other hand, domain decomposition immediately solves the data problems for both materials and tallies. Specifically, when memory pertaining to certain regions is allocated only on nodes that track those regions, the footprint on each is reduced inversely to the number of spatial domains used. This is not a new concept for MC codes ([18, 19, 20, 21, 22]), but to our knowledge it has not been applied to full-core 3D reactor analyses. For this class of problem it still needs to be demonstrated that the concept is feasible for realistic calculations that incur the true memory and particle communication burden.

The ability to model and predict the extent of particle communication costs and load imbalances has been investigated in [23, 24] for a range of machine and problem parameters with a simplified MC code. In these analyses it is demonstrated that these costs can be computed from the peaking factor and spatial statistics of the problem (*i.e.*, per-domain leakage fractions), and that only modest penalties are predicted for a typical reactor geometry.

These results motivated the present work, where we implement domain decomposition in a full-physics MC code (OpenMC, [14]) and explore the full-size reactor problem for both tally and particle communication performance characteristics.

2. Domain Decomposition Implementation

2.1. Model

The present work implements domain decomposition by adding particle synchronization *stages* during each generation of particles, as described in [23]. This is much simpler than the algorithms presented in [19], [20], [21], and [22] but it allows for the derivation of an explicit timing model that can be used to predict how well it will perform for the full-fidelity reactor problem. The changes to the outer loop of the OpenMC eigenvalue iteration routine are shown in lines 4, 5, 9-11, and 15-17 of Algorithm 1.

Algorithm 1 OpenMC eigenvalue iteration outer loop with domain decomposition modifications, from the perspective of one spatial domain. Only particles in the present domain are run (the *local source*) at each stage. While transporting particles, if the distance to a collision $d_{collision}$ is larger than the distance to a domain boundary $d_{boundary}$, the particle is buffered for communication.

```
1:  Guess initial fission distribution
2:  for each generation of particles do
3:      Sample particles from fission distribution
4:      Compile local source
5:      repeat
6:          for each particle in the local source do
7:              while Neutron not yet absorbed do
8:                  Sample d_collision, find d_boundary
9:                  if d_collision > d_boundary then
10:                     Buffer particle for send, Break
11:                 end if
12:                 Sample physics, calc. tallies
13:             end while
14:         end for
15:         Communicate particle buffer
16:         Rebuild local source from incoming particles
17:     until All particles in generation are absorbed
18:     Synchronize tallies
19:     Calculate eigenvalue
20:     Rebuild fission source distribution
21: end for
```

This method blocks all processes at each synchronization stage (line 15 in Algorithm 1), which clearly presents a potential inefficiency for load-imbalanced problems. This effect can be predicted with the performance model, and we will show in later sections that it can be largely mitigated with load-balancing techniques.

As described in [24], the time τ needed to complete a perfectly load-balanced domain-decomposed Monte Carlo run using this simple scheme can be modeled as a combination of latency, bandwidth, and particle tracking components:

$$\tau = \tau_{latency} + \tau_{bandwidth} + \tau_{tracking} \tag{1}$$

It is straightforward to write this in terms of the number of synchronization stages M, the average number of particles $\bar{P}_i$ run in each domain at stage i, and the average fraction $\bar{\lambda}_i$ of particles that leak out of each domain at stage i:

$$\tau = 6\alpha M + \beta \sum_{i=0}^{M-1} \bar{\lambda}_i \bar{P}_i + \mu \sum_{i=0}^{M-1} \bar{P}_i \tag{2}$$

where α, β, and μ are the latency, inverse bandwidth, and inverse particle tracking rate coefficients of the system, with units of seconds per connection, seconds per particle transfered, and seconds per particle simulated, respectively. This applies to a rectilinear decomposition where each domain has six Cartesian neighbors.

For real problems with load imbalances we can write the number of particles run in each domain at each stage as a deviation from the perfectly load-balanced average. Since this approach requires all processes to wait at the synchronization points between stages, τ' for the load-imbalanced problem can be written as:

$$\begin{aligned} \tau' &= 6\alpha M + \beta \sum_{i=0}^{M-1} \lambda_i^{max} \left(\bar{P}_i + \delta p_i^{max}\right) + \mu \sum_{i=0}^{M-1} \left(\bar{P}_i + \delta p_i^{max}\right) \\ \tau' &\approx \tau + \beta \sum_{i=0}^{M-1} \lambda_i^{max} \delta p_i^{max} + \mu \sum_{i=0}^{M-1} \delta p_i^{max} \end{aligned} \tag{3}$$

where δp_i^{max} and λ_i^{max} denote the leakage and load deviation for the domains at stage i that take the longest to finish tracking and transmitting particles, respectively (these could be from the same domain, but not necessarily). The relation to τ is approximate because $\lambda_i^{max} \bar{P}_i \neq \bar{\lambda}_i \bar{P}_i \forall i$ in the bandwidth term. However, for nearly all practical cases the particle tracking term will be much larger than the bandwidth term, so this approximation should produce negligible error.

We can then quantify the magnitude of the *load imbalance penalty* Δ caused by the blocking synchronization points between stages as

$$\Delta \equiv \frac{\tau' - \tau}{\tau}. \tag{4}$$

In [24] the authors took this a step further, expanding the sums in Equations 2 and 3 to write them in terms of the load imbalance distribution at the initial particle tracking stage. For example,

$$\begin{aligned} \tau_{tracking} &= \mu \sum_{i=0}^{M-1} \bar{P}_i \\ &= \mu \left(\bar{P}_0 + \bar{P}_1 + ... + \bar{P}_{M-1}\right) \\ &= \mu \left(\bar{P}_0 + \bar{\lambda}_0 \bar{P}_0 + \bar{\lambda}_0 \bar{\lambda}_1 \bar{P}_0 + ... + \bar{\lambda}_0 \bar{\lambda}_1 ... \bar{\lambda}_{M-2} \bar{P}_0\right) \\ &= \mu \bar{P}_0 \left(1 + \lambda_0 + \bar{\lambda}_0 \bar{\lambda}_1 + ... + \bar{\lambda}_0 \bar{\lambda}_1 ... \bar{\lambda}_{M-2}\right) \\ &= \mu \bar{P}_0 \left(1 + \sum_{i=0}^{M-1} \prod_{k=0}^{i} \bar{\lambda}_k\right). \end{aligned} \tag{5}$$

We can expand the bandwidth term in a similar fashion and write Equation 2 as

$$\tau = 6\alpha M + \beta \bar{P}_0 \|\bar{\lambda}\| + \mu \bar{P}_0 \left(1 + \|\bar{\lambda}\|\right) \tag{6}$$

with

$$\|\bar{\lambda}\| \equiv \sum_{i=0}^{M-1} \prod_{k=0}^{i} \bar{\lambda}_k. \tag{7}$$

The authors next considered that in the worst case

$$\delta p_{i+1}^{max} \leq \lambda_i^{max} \delta p_i^{max}, \tag{8}$$

which with similar expansions allows Equation 3 to be written as

$$\tau' \lesssim \tau + \beta \delta p_0^{max} \|\lambda^{max}\| + \mu \delta p_0^{max} \left(1 + \|\lambda^{max}\|\right) \tag{9}$$

with

$$\|\lambda^{max}\| \equiv \sum_{i=0}^{M-1} \prod_{k=0}^{i} \lambda_k^{max}. \tag{10}$$

Finally, by substituting Equations 6 and 9 into Equation 4 and then rearranging we arrive at an upper bound for the load imbalance penalty as

$$\Delta \leq \frac{\delta p_0^{max}}{\epsilon + \frac{1}{C} \bar{P}_0} \tag{11}$$

for

$$C \equiv \frac{\mu \left(1 + \|\lambda^{max}\|\right) + \beta \|\lambda^{max}\|}{\mu \left(1 + \|\bar{\lambda}\|\right) + \beta \|\bar{\lambda}\|}. \tag{12}$$

Neglecting ϵ (a multiplicative combination of α, β, and M), we obtain the form presented in [24]:

$$\Delta \leq C \frac{\delta p_0^{max}}{\bar{P}_0} = \frac{C}{\Gamma_0} - C. \tag{13}$$

with *load balance* $\Gamma_i \equiv \bar{P}_i / P_i^{max}$.

Note that the treatment presented here represents a slightly different re-derivation of the previously-reported performance model with fewer approximations, and that the $||\lambda||$ terms presented here are not directly comparable to their previous definitions.

In this work we use data from real DD runs in Equations 2 and 3 to calculate the load imbalance penalty directly, as well as compute the upper-bound from Equation 13. Both of these are then compared to the observed load imbalance penalties.

2.2. Implementation

Support for domain decomposition was implemented in OpenMC for rectilinear meshes that can be laid over arbitrarily complex geometries. The initial focus was on the inter-domain particle communication routines to validate the performance model and demonstrate the feasibility of DD for full reactor problems. Future work will handle the domain-aware loading of tallies and materials in memory, which was not implemented here in a general way. However, domain-aware memory loading of rectilinear mesh-tallies was implemented, with the requirement that each mesh bin be fully encompassed inside a single domain. This means that memory will only be allocated for each tally mesh bin on compute nodes that track particles in the same region of space, and that no tally synchronization is required between domains.

The particle communication logic (line 15 of Algorithm 1) is shown in detail in Algorithm 2. It was designed for cases when the number of domains is less than or equal to the number of compute processes available. In other words: more than one compute process can be assigned to work on the same domain if extra resources are available, with the particle tracking load in that domain divided evenly among them (*i.e.*, traditional Monte Carlo parallelism). This is handled efficiently by conceptualizing the total number of particles entering a domain from any neighbor as belonging to an ordered array. Then, analogous to the fission bank treatment described in [15], the boundaries of slices "owned" by each process can be computed and used to buffer and send particles to the correct location. Figure 1 attempts to clarify this concept with an example.

It should be noted that each domain must receive information from all second-degree neighbors regarding the number of particles being sent to its direct neighbors (line 2 of Algorithm 2). For example, domains $d0$ and $d1$ in Figure 1 are not direct neighbors that communicate particles at each stage, but processes on each do need to know how many particles are being sent to their shared direct neighbor domain $d3$. Thus, at most each domain must communicate with a maximum of 24 domains at each stage (the 3D von Neumann neighborhood of range $r = 2$). This means that the computational complexity of this algorithm is not a direct function of the number of domains. However, it does scale linearly with the number of processes on neighboring domains, which relates to the number of domains depending on the load-balancing strategy employed.

2.3. Random Number Reproducibility

Random number reproducibility was retained in this implementation by adding the LCG random number seed to the particle data structure and communicating it with particles as they cross domain boundaries. New fission particles received initial seeds that were randomly sampled using the

Algorithm 2 Particle communication routine

1: Count no. of particles to send to each neighbor domain i: MPI_ALLREDUCE among processes in just this domain
2: Synchronize domain send info in local neighborhood (i.e. all $n\_send_{domain(j)\to domain(i)}$)

Determine sending process starts

3: **for** $i = 1 \to 6$ **do**
4: $\quad starts(i) = 0$
5: $\quad$ **for** $j = 1 \to 6$ **do**
6: $\quad\quad starts(i) = start(i) + n\_send_{domain(j)\to domain(i)}$
7: $\quad\quad$ **if** $domain(j) == my\_domain$ **then**
8: $\quad\quad\quad$ Break $\quad\quad \triangleright\ starts(i)$ is now the domain start
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: $\quad$ Find $proc\_offset$ with MPI_EXSCAN within domain
12: $\quad starts(i) = starts(i) + proc\_offset$
13: $\quad\quad\quad\quad \triangleright\ starts(i)$ is now the current process start
14: **end for**

Determine receiving process (pid) finishes

15: **for** $i = 1 \to 6$ **do**
16: $\quad mod = modulo(n\_send_{all\to domain(i)}, n\_procs(i))$
17: $\quad$ **for** $pid = 1 \to n\_procs(i)$ **do**
18: $\quad\quad finishes(i, pid) = i \times \frac{n\_send_{all\to domain(i)}}{n\_procs(i)}$
19: $\quad\quad$ **if** $i - 1 < mod$ **then**
20: $\quad\quad\quad finishes(i, pid) = finishes(i, pid) + i$
21: $\quad\quad$ **else**
22: $\quad\quad\quad finishes(i, pid) = finishes(i, pid) + mod$
23: $\quad\quad$ **end if**
24: $\quad$ **end for**
25: **end for**

Determine send info to each process (pid)

26: **for** $i = 1 \to 6$ **do**
27: $\quad pid = search(starts(i), finishes(i, :))$
28: $\quad$ **for** $s = 1 \to n_{buffered\_particles\to i}$ **do**
29: $\quad\quad$ Buffer site s for sending to process pid
30: $\quad\quad starts(i) = starts(i) + 1$
31: $\quad\quad$ **if** $starts(i) > finishes(i, pid)$ **then**
32: $\quad\quad\quad pid = pid + 1$
33: $\quad\quad$ **end if**
34: $\quad$ **end for**
35: **end for**

36: Send/Recv particles

Symbol Table

i, j	Local indices of first- and second-degree neighbor domains, respectively
$domain(i)$	Global domain index of local neighbor i
my_domain	Global domain index of the current process
$n\_send_{domain(j)\to domain(i)}$	No. of particles being sent to direct neighbor i from second-degree neighbor j
$starts(i)$	Starting index of the current process's slice in the total particle array going to domain i
$finishes(i, pid)$	Ending index of process pid's slice in the total particle array going to domain i
$proc\_offset$	Starting index of the current process's slice in the particle array from my_domain only
$n_{procs}(i)$	No. of processes working on domain i
$n_{buffered\_particles\to i}$	No. particles to send to domain i

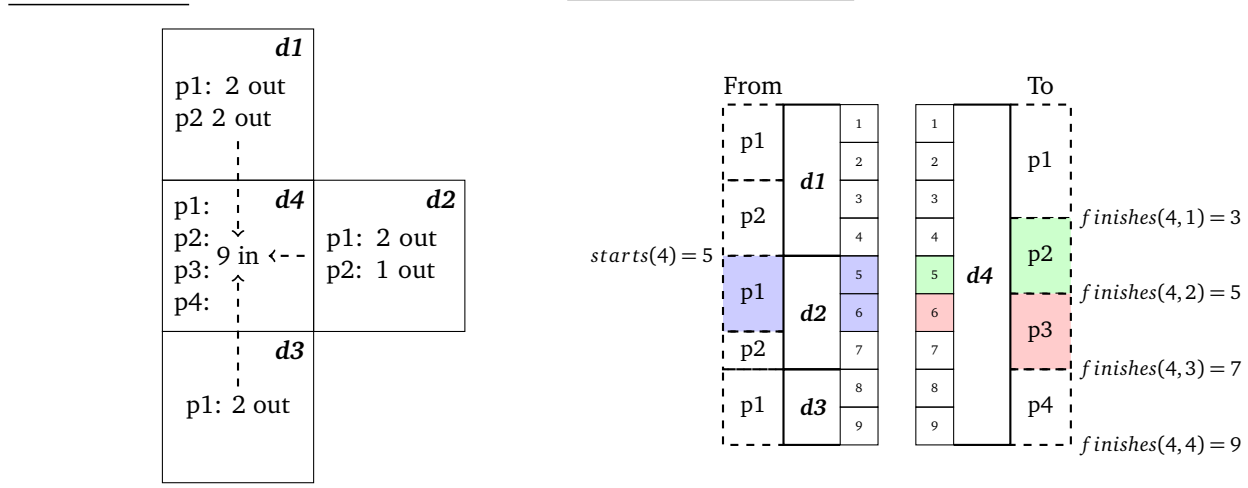

Fig. 1. *Left*: Scheme of four square domains, where processes on domains $d1$, $d2$, and $d3$ have a total of nine particles to send to processes on domain $d4$. *Right*: Particle communication scheme from the perspective of process $p1$ on domain $d2$, when deciding how to send particles 5 and 6 to domain $d4$. The total number of particles being sent to domain $d4$ are conceptualized as existing in a fictitious ordered array, where slices are 'owned' by different processes. The left is the particle array from the perspective of sending processes; the right is same array from the perspective of receiving processes. By comparing the starting and ending indices of the slices between the sending and receiving processes (the $starts$ and $finishes$ arrays in Algorithm 2), each process can decide where to send particles in a distributed fashion. In this example, this process sends one particle to process $p2$ and one particle to process $p3$.

random number stream of the parent, and source sampling between successive fission generations was done using these new streams. This ensures that eigenvalue and tally results will be identical regardless of the number of domains and compute processes used: a highly desirable quality of the code for debugging and validation purposes.

The starting source - typically sampled from some analytical spatial distribution - also needs to be consistently chosen to retain random number reproducibility. This is not entirely straightforward, since it is not known *a priori* for a given domain mesh how many particles will start in each region. In the current implementation this is handled by having processes on all domains sample all particles of the starting source in the same order with the same random number stream, keeping only those that fall in their domain (*i.e.*, rejection sampling). However, this will be extremely inefficient for runs with fine domain meshes and a large number of particles per generation. Future work will switch to a scheme where all processes sample a subset of the source using the appropriate section of the random number stream (like for the traditional parallel implementation), and then communicate particles to the appropriate domain depending on the sampled location of each.

2.4. Test Problems

The performance of this domain decomposition implementation was explored in two ways: 1) by scaling the number of domains to evaluate communications performance and compare load imbalance penalties to the analytical model and 2) by scaling the size of a mesh-tally on a fixed domain mesh to demonstrate the feasibility of accomplishing Terabyte-scale tallies.

Runs were carried out for two problems: 1) an infinite medium fuel and water mixture that serves as a perfectly load-balanced baseline, and 2) the BEAVRS Pressurized Water Reactor (PWR) benchmark, a realistic 3D full core nuclear reactor with a physically relevant spatial distribution.

The BEAVRS benchmark geometry is shown in Figure 2 - the specification describes a PWR with 193 standard 17x17-pin Optimized Fuel Assemblies ([25]). For both BEAVRS and the infinite medium problem, domain meshes consisting of evenly-spaced cubes were overlaid on top of the geometry for a variety of domain sizes - i.e., 2x2x2 = 8 domains total, 3x3x3 = 27 domains total, etc., up to the 8x8x8=512 domain case, which is shown in Figure 2.

Mesh-tallies were carried out for BEAVRS with the 8x8x8 DD mesh for a variety of meshbin sizes, shown in Table 1 and depicted in Figure 3. These were chosen to be structured Cartesian meshes across the entire geometry, sized so that each cubic mesh cell fits exactly into only one domain. These were each run separately for both tracklength flux and analog fission tallies. The reported tally size is calculated from the number of mesh bins and the fact that OpenMC requires 24 bytes per meshbin: three eight-byte double-precision numbers corresponding to the temporary batch accumulator, the total accumulator for the mean, and the total accumulator for the variance.

It should be noted that full-core depletion problems will not require these types of meshes - cell tallies as described in [8] are more appropriate. In the current work we use these meshes as a proxy for the real isotopic reaction rate tallies to demonstrate the efficiency of the OpenMC tally system at this scale while using domain decomposition.

All runs were carried out on the Mira IBM Blue Gene/Q supercomputer at the Argonne National Laboratory which has 16GB of memory on each compute node. As indicated in Table 1, the finest tally mesh used 30% of the memory on

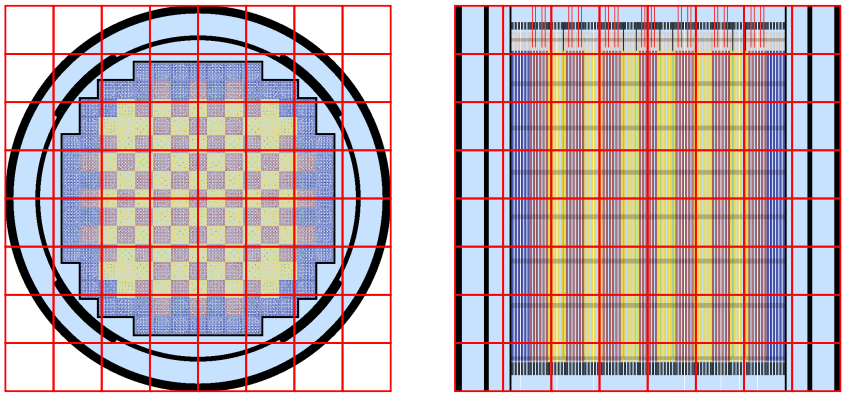

Fig. 2. Geometry of the BEAVRS PWR benchmark showing the 8x8x8 DD mesh used for tally results. *Left:* Radial top-down view. *Right:* Axial view.

each node when decomposed.

Table 1
Mesh-tallies used with the 512 domain mesh over the BEAVRS PWR geometry. All tally meshes comprised of cubes covering the entire geometry, sized so that no tally mesh bin crosses a domain boundary. For instance, mesh 1 had one tally bin per domain coinciding exactly with the domain mesh, mesh 2 had 8 tally bins per domain, etc.

Mesh	No. Tally Cells	Tally Size (GB)	% Memory
1	512	1.144×10^{-5}	0.00%
2	4096	9.155×10^{-4}	0.00%
3	5.316×10^{7}	1.188	0.01%
4	4.252×10^{8}	9.505	0.12%
5	3.402×10^{9}	76.04	0.93%
6	6.816×10^{9}	151.6	1.85%
7	1.369×10^{10}	305.9	3.73%
8	2.743×10^{10}	613.2	7.49%
9	5.487×10^{10}	1226	15.0%
10	1.095×10^{11}	2447	29.9%

3. ADDRESSING LOAD IMBALANCES

As discussed in [24], load imbalances are normally expected to occur as domains get smaller and leakage fractions increase, incurring significant parallel inefficiencies when scaling the number of domains. Several algorithmic strategies exist to address this problem. In practice, a combination of each method may be appropriate depending on the number of compute nodes and domains, the severity of the load imbalance, and the extent to which the spatial load distribution is known.

3.1. Matching Resources to Load

The current DD implementation allows for any number of processes to work on each domain, facilitating the use of simple Cartesian domain meshes. If the load distribution on such a mesh is known or can be approximated, the imbalance problem can be solved to the extent that resources can be deployed to match that distribution (see Figure 5). This approach works particularly well if all domains have some particles to transport, and the number of particles run does not vary widely in adjacent domains.

However, for the light water reactor problem of interest depicted in Figure 2 this is not sufficient to alleviate all load imbalances when using regular Cartesian domain meshes. For instance, it is clear that domains in the corners of the geometry would have significantly fewer particles to track than others. At least one node must still be assigned to each domain to handle any particles that travel there; in the worst case these nodes would spend all of their time idling between synchronization steps if they have no particles to track.

Some specifics for determining how to distribute resources according to load was discussed for Monte Carlo codes in [19] and demonstrated for small-scale problems in a code that performs re-balancing dynamically during a simulation. As the authors of [19] point out, the calculation of the optimal number of compute nodes for each domain is function of the amount of work done on each domain, which takes into account all work associated with interactions during particle transport.

For the purposes of this study a simpler strategy was employed to demonstrate this idea. Here, compute nodes were distributed based only on the number of starting particles in each domain: information that was tallied from non-domain-decomposed runs. Since the characteristics of particle transport are relatively homogeneous for this problem at the scale of the domains being used, this should work reasonably well. Algorithm 3 describes the routine used to determine this distribution for each domain mesh. This routine simply computes the number of nodes to assign to each domain using the fraction of the starting source in that domain, ensuring that all domains would be tracked by at least one. For domains where the fraction indicates that less than one node would be needed, nodes are symmetrically reassigned from domains that have the most nodes already working on them.

As shown in Figure 4, the quality of the matches produced by this algorithm can be improved upon to a large degree. Using the notation from Algorithm 3, the match discrepancy d is calculated for each domain i as

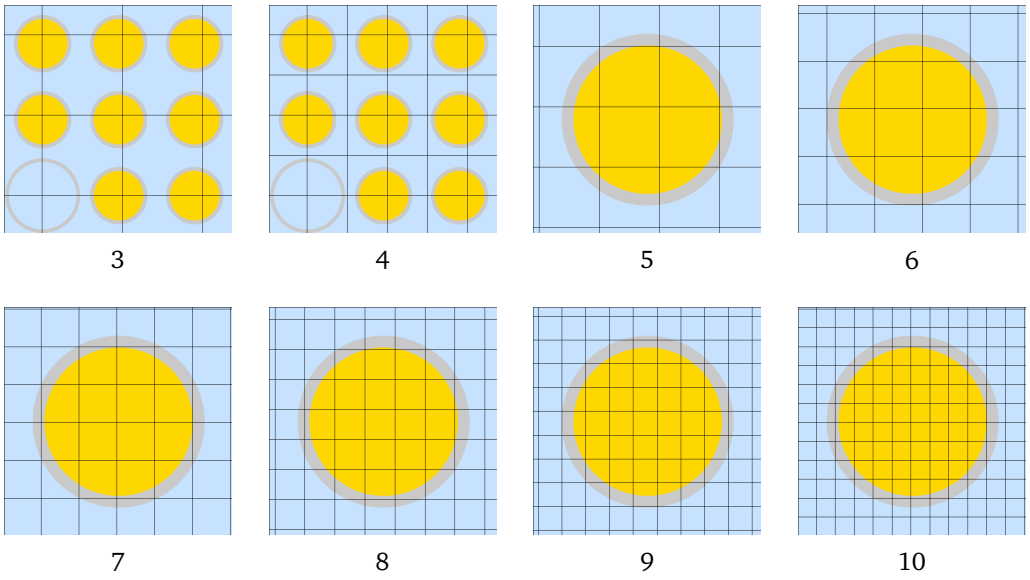

Fig. 3. Zoom of mesh-tally boundaries over a 3x3 section of BEAVRS pins (meshes 3 and 4) and a single pincell (meshes 5-10).

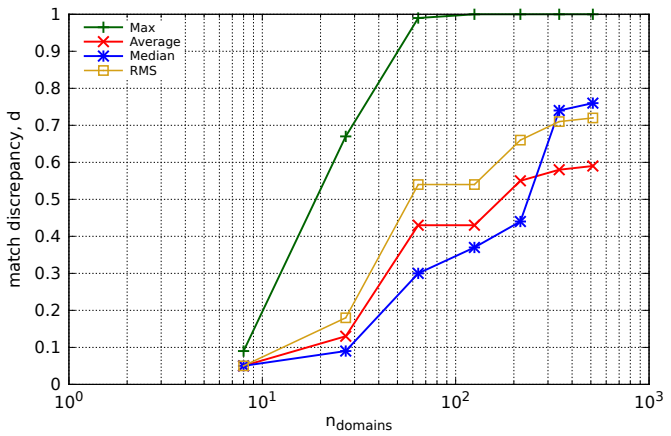

Fig. 4. Statistics describing the match quality of the resource maps produced by Algorithm 3 for the domain meshes described in Section 2.4.

$$d(i) = \frac{\text{abs}\left(nodes(i) - frac\_nodes(i)\right)}{nodes(i)}. \qquad (14)$$

Values of zero indicate that the number of nodes needed for that load was matched exactly. Since this algorithm assigns at least one node to all domains, drawing nodes away from domains with the highest loads, $frac\_nodes(i)$ will always be less than $nodes(i)$, and values of one indicate that a node was assigned to a domain that had an insignificant percentage of the total load. It is clear that this is the case for the finer meshes described in Section 2.4, where corner domains may not cover meaningful portions of the problem geometry (*e.g.*, see the corner domains in Figure 2).

Regardless of the poor quality, these resource mappings do lead to reasonable load balancing performance, as shown in Section 4.

Algorithm 3 Resource matching routine.

input $n\_domains$, $nodes\_available$, $source$
for $i = 1 \rightarrow n\_domains$ **do**
  $frac\_nodes(i) = \frac{source(i)}{\text{sum}(source)} \times nodes\_available$
  $nodes(i) = \max(1, \text{ceiling}(frac\_nodes(i)))$
end for
while $\text{sum}(nodes) > nodes\_available$ **do**
  for $i = 1 \rightarrow n\_domains$ **do**
    if $nodes(i) == \text{maxval}(nodes)$ **then**
      $nodes(i) = nodes(i) - 1$
      Break
    end if
  end for
  $nodes = \text{reverse}(nodes)$ ▷ Reverse order of the array
end while
Ensure $nodes$ is in forward order
output $nodes$

Symbol Table

$n\_domains$	Number of domains
$nodes\_available$	Total number of available compute nodes
i	Domain index
$source(i)$	Number of starting particles in domain i
$frac\_nodes(i)$	Fractional number of nodes needed for domain i
$nodes(i)$	Output actual number of nodes for domain i

3.2. Tailoring the Domain Mesh to Load

The other traditional way to manage load imbalances in DD simulations is to size the domain mesh according to the load. This requires some knowledge of the load distribution, as well as support for unstructured or irregular domain meshes. OpenMC does not currently support general unstructured meshes, so in order to demonstrate the effect on performance a modified run mode was added to allow the domain mesh to shrink (see Figure 5). In this mode materials outside the domain mesh are loaded on all nodes, so particles that travel there can continue to be tracked. However, tally decomposition using this capability was not implemented.

3.3. Particle Density Manipulation

Load imbalances might also be addressed by enforcing identical loads in each domain, adjusting weights of particles to preserve the fair game. This can be done during transport using traditional Monte Carlo variance reduction techniques such as splitting and Russian Rouletting, or between particle generations during source sampling in eigenvalue problems with, for instance, the Uniform Fission Site (UFS) method [26, 27]. The former can be used to achieve balance for all space, whereas the latter is only helpful over source regions (fuel assemblies in Figure 2).

It is also important to keep in mind that the effectiveness of particle density manipulation to alleviate load imbalances in DD simulations is not adequately described by a simple measure of parallel efficiency. Specifically, the choice of method can strongly affect the statistical convergence of tallies in each region of space. Indeed, this is why such methods are typically used for variance reduction in non-DD simulations. As a result, efficiency should be assessed using a more general figure of merit that includes the statistical convergence in regions of interest. In particular, splitting/rouletting could result in good load balance by manufacturing more work in reflector regions that normally have few particles to run. This improves the statistical quality of any tallies in those regions, but reduces the quality of tallies in fuel regions of interest. On the other hand, UFS serves to flatten the distribution of variances in the fuel region - a desirable quality of the method.

4. Results

4.1. Domain Decomposition Performance

Load imbalance penalties were calculated for both test problems from Equation 4 using observed run timings. The DD problem was run for each case to obtain τ' directly, and τ was approximated by running the problem with identical parameters and the traditional parallelization scheme. Since the traditional parallelization scheme evenly divides particles across all processes, these run timings will be equivalent to τ (the time for the DD run to finish if it were perfectly load-balanced) as long as the particle tracking term in Equation 2 is significantly larger than the latency and bandwidth terms. On Blue Gene/Q α and β are on the order of 10^{-8} - whereas $\mu \sim 10^{-5}$ - so this is a decent approximation to make. Note that

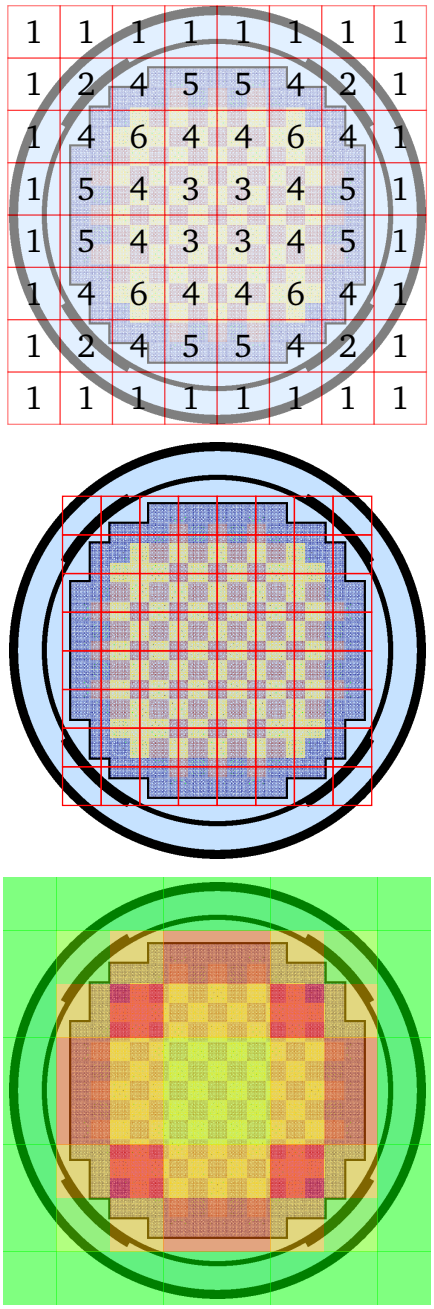

Fig. 5. Radial view of the BEAVRS geometry showing the 8x8x8 DD for three load-balancing strategies. *Top*: Resource matching; numbers indicate a potential distribution for the number of processes working on each domain. *Middle*: Restricted mesh; the domain is decomposed only over high-load regions. *Bottom*: Particle density manipulation; colors indicate a potential source biasing or particle re-weighting distribution.

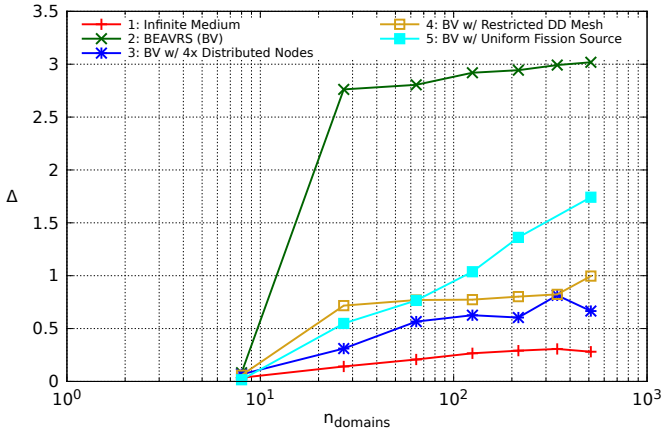

Fig. 6. Timing penalties (Equation 4) for the OpenMC domain decomposition implementation for two eigenvalue test problems and several load-balancing strategies, calculated from transport time only (total time - initialization and finalization time). Domain meshes were evenly-spaced cubes overlaid on top of the geometry, *i.e.*, 2x2x2, 3x3x3, ..., 8x8x8. Each mesh was run with one node per domain, with the exception of curve 3, where four times as many nodes as domains were distributed across domains according to the converged fission source distribution. Curve 4 was run with domain meshes where the outer boundaries were located on the edge of the fuel assemblies, as depicted in Figure 5. All runs were conducted as a weak-scaling study, where the total number of particles transported was equal to 10000 per batch per node (*i.e.*, for all 512-domain runs, 5.12 million particles per batch).

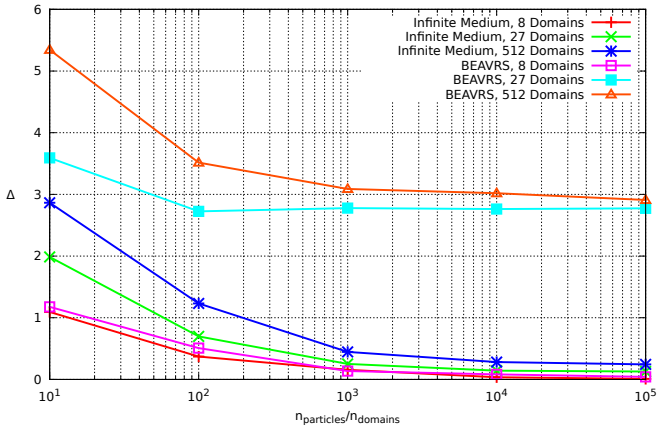

Fig. 7. Timing penalties for the OpenMC domain decomposition implementation for two eigenvalue test problems when varying the number of particles tracked per processor per batch.

all timings used in these equations refer to particle transport only, and exclude initialization and finalization time. This is consistent with the scaling study in [14], which demonstrates excellent parallel efficiency for particle transport in OpenMC using domain replication.

Figure 6 shows results for Δ as the domain mesh was refined. Curves 1 and 2 show results for the BEAVRS and infinite medium problems, and curves 3-5 demonstrate the effect of each of the three previously-discussed load balancing strategies on the DD BEAVRS runs. All runs used one compute node per domain with the exception of those in curve 3, which distributed 4 times as many nodes as domains across the space according to the converged particle source. Curve 1 is intended to demonstrate the overhead introduced by the domain decomposition mechanics (*i.e.*, the entire process of buffering and sending particles) as distinct from load imbalances, since the problem is naturally load-balanced in that case.

Clearly the load imbalance penalty can be mitigated to a significant degree. As expected, the efficacy of UFS decreases as the number of domains that cover corner regions increases: the method only flattens the particle density over the fuel assembly region.

It should be noted that the domain boundaries in the 8-domain BEAVRS run split the particle load along lines of symmetry. As a result, for those runs it is expected that the load imbalance penalties mirror the infinite medium case.

All of the runs in Figure 6 were carried out in a weak-scaling manner with 10k particles per domain. Figure 7 shows how the implementation behaves when this parameter is varied. As can be expected, performance degrades as the amount of work to do between synchronization points decreases and the latency and bandwidth terms in Equation 2 start to dominate. However, this result is encouraging when we note that this regime does not begin until we get below 1k particles per domain. This means that we can safely use very fine domain meshes and achieve similar performance without needing to increase total particle counts. It should also be noted that this plot was created without any tallies, so in real cases particle tracking will be slower and the situation more favorable.

4.2. Model Validation

The domain particle sources and leakage fractions were tallied for all runs in curves 2 and 4 in Figure 6. These allowed for the calculation of load imbalance penalties predicted by the model using Equation 2, the unapproximated Equation 3, and Equation 4. The same data can also be used to calculate the approximate upper-bound penalties using Equation 13. These results are shown in Figures 8 and 9. Consistent with [28], the values of α and β were estimated as 10^{-6} seconds per connection and 10^{-8} seconds per particle respectively (for β, 2 GB/s links with roughly 200 bytes per particle transfered). The particle tracking rate coefficient μ was set to 10^{-5} seconds per particle, which is consistent with tracking rate values observed for all runs on this machine.

The approximate upper-bounds appear valid for these results, but direct calculation of penalties with the unapproximated equations fall much closer to observed values. Note that since these equations account for only particle transport time and the latency/bandwidth times associated with sending particles - missing the additional communication and computation steps carried out in Algorithm 2 - it is not unexpected for the direct calculation to underestimate the penalties. However, as observed we expect this additional overhead to stay small, since the communication stencil inherent in Algorithm 2 does not grow with finer domain meshes.

The magnitude of the upper-bound is very sensitive to $\|\lambda^{max}\|$, which is a combination of the maximum domain

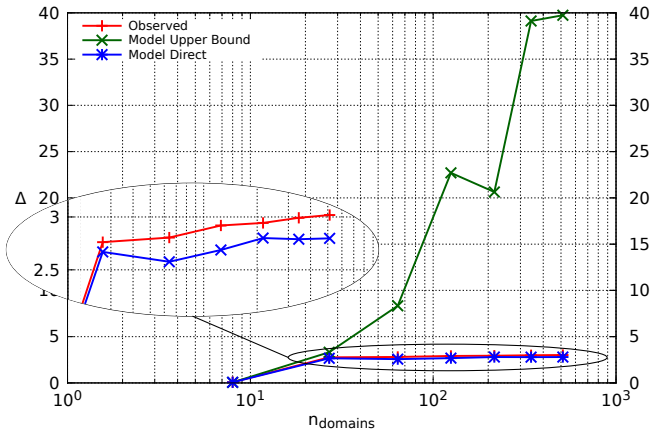

Fig. 8. Load imbalance penalties for the BEAVRS problem without any load balancing techniques (curve 2 in Figure 6) compared to direct calculation using the performance model as well as predictions for the upper bound.

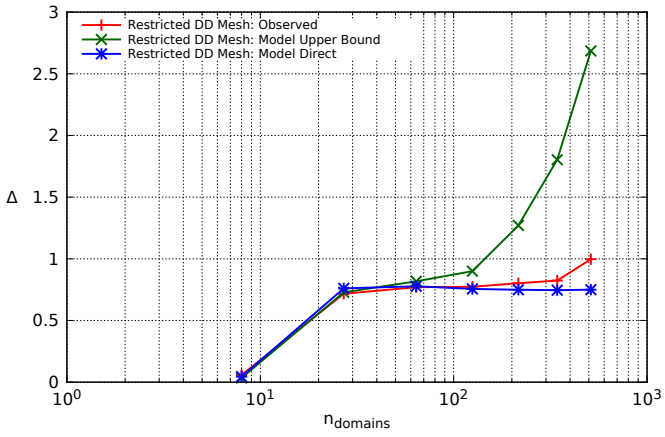

Fig. 9. Load imbalance penalties for the BEAVRS problem using the restricted mesh load balancing strategy (curve 4 in Figure 6) compared to direct calculation using the performance model as well as predictions for the upper bound.

leakage fractions over all stages. In many cases, it was observed that later stages consisted of a small number of neutrons bouncing back and forth between domains (especially common in reflector regions). In these cases the maximum leakage fraction λ_i^{max} tended to be unity, serving to inflate the averaged leakage fraction considerably, and leading to the over-estimation of the load imbalance penalty. This also explains the apparent noise in the upper-bound curve of Figure 8: results are highly sensitive to the stochastic movements of a very small number of particles. Overlapping-domain strategies discussed in the literature (*e.g.* [22]) have been shown to reduce this kind of particle movement, which would lessen the impact of the latency and bandwidth terms.

To verify that the present derivation of the model upper-bound is equivalent to that presented in [24], the previously-reported leakage fraction and particle source data tallied from full-scale OpenMC runs on the Monte Carlo Performance Benchmark problem ([29]) were re-processed by the present

Table 2
Upper-bound model parameters computed using previously-tallied data, presented for verification. Dual table entries denote: (value reported in [24]), (value calculated by the present authors).

Mesh	C	$\frac{1}{\Gamma_0}$	$\Delta \leq$
Full Assembly	1.13, 1.13	3.24, 3.25	3.67, 3.68
Quarter	2.58, 2.53	3.28, 3.31	8.46, 8.38
Ninth	6.98, 7.06	3.45, 3.44	24.08, 24.25

Table 3
Directly-calculated load imbalance penalties predicted by the unapproximated model equations from previously-tallied data. The full case used all data (compare to the mesh in Figure 2), whereas the restricted case used data only from assembly regions and stages where particles were run in assembly regions (compare to the restricted mesh in Figure 5).

Mesh	No. Domains	Δ, restricted	Δ, full
Full Assembly	5780	2.67	3.42
Quarter	46240	2.39	3.06
Ninth	156060	2.44	3.12

authors. These results are presented in Table 2, confirming the equivalence between the two derivations. Note that for these calculations tally data was only considered from assembly regions and "active stages" (stages that had particles running in assemblies, as opposed to reflector regions).

The previous tally data were also used to compute the load imbalance penalties directly with Equations 2, 3, and 4, presented in Table 3. These results are consistent with the load imbalance penalties observed in Figure 6 for coarser domain meshes.

4.3. Full-Core Tallies

Timing results for mesh-tallies 4-10 from Table 1 and Figure 3 are presented in Figure 10 and Table 4. All were done with analog fission tallies on the 8x8x8 domain mesh in Figure 2. Tracklength flux tallies took marginally longer, but identical trends were observed.

As expected, the amount of time needed to write output files grows with tally size. However, for longer production runs this will be constant and essentially negligible compared to particle transport time. We also observe from Table 4 that the equivalent non-DD run for mesh 3 is 3.3x faster than the same DD run after subtracting finalization time. This is consistent with the observed load imbalance penalty, and indicates the small magnitude of the additional overhead incurred by doing tallies. It should be noted that the finalization time for mesh 3 is larger in the non-DD case because OpenMC does not normally finalize tallies in parallel, instead consolidating all results to a master process for output writing.

As indicated in Table 1, tally result files were approximately 4.8GB per domain, or $\sim$ 30% of node memory. Post-processing was manageable with standard shell commands, requiring for instance only 30 seconds of additional computation to parse and extract an axial slice of tally data from mesh 10.

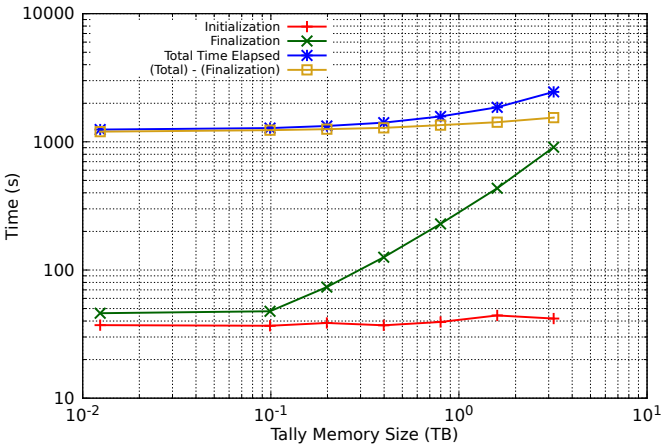

Fig. 10. Timing results for full-core analog fission tallies on meshes 4-10 in Table 1. All runs were carried out for 10 batches (5 active) of 5.12 million particles per batch on a half rack of Blue Gene/Q (512 compute nodes).

Table 4
Timing results in seconds for full-core tallies on the meshes in Table 1, as depicted in Figure 10. I: Initialization time; F: finalization time; T: total run time. For comparison, timings are also shown for the three smallest mesh-tallies run with traditional parallelization (no domain decomposition), as well as DD and non-DD versions of the same problem without any tallies. No domain load balancing was employed.

	Tally Mesh	I	F	T
No DD no Tally	N/A	41	0	278
No DD and Tally	3	37	267	626
DD no Tally	N/A	40	0	977
DD and Tally	1	39	21	1205
	2	40	21	1207
	3	46	25	1222
	4	37	46	1247
	5	37	48	1280
	6	39	74	1329
	7	37	126	1412
	8	39	229	1578
	9	44	435	1859
	10	42	908	2453

5. Conclusions

A simple version of domain decomposition was implemented in the OpenMC Monte Carlo neutron transport code, and an updated performance model was presented. Performance was explored with a suite of runs carried out on a half-rack of IBM Blue Gene/Q, which provided timing results that quantify the magnitude of load imbalance penalties for a detailed nuclear reactor geometry and empirically validate the performance model. In addition, several basic load-balancing strategies were explored and observed to mitigate these slowdowns considerably, including a new and efficient way to distribute extra compute nodes across the domain mesh. Finally, good performance was observed while carrying out mesh-tallies of up to 2.4TB in size, distributed across 512 spatial domains.

This work demonstrates the feasibility of using spatial domain decomposition with high-fidelity Monte Carlo particle transport methods to tally quantities at the Terabyte scale, which is needed to conduct robust nuclear reactor depletion analyses. Of particular interest is the observation (both empirical and through the updated performance model) that the load imbalance penalties for fine domain meshes may not be as large as previously thought for these types of problems. Furthermore, the good performance observed for Terabyte-scale tallies in the present results imply that it may not be necessary to use domain meshes much finer than those employed here, especially since these tallies used only 40% of the available node memory. The situation is expected to improve with the implementation of advanced DD methods discussed in the literature ([18, 19, 20, 21, 22]).

Future work should complete the implementation of cell tallies that use an arbitrary depletion mesh. A particularly desirable feature of this implementation will be the ability to insulate the user from deciding on which domains materials and tallies need to be allocated for any given spatial decomposition. Furthermore, since we speculate that the optimal configuration might entail utilizing data decomposition among processers within each spatial domain, a marriage of the two methods will be explored.

Once these final pieces are added, a more complete exploration of the full-core fully-detailed depletion problem will be possible with OpenMC, which is expected to perform well on next-generation distributed computing architectures.

Acknowledgments

This work was sponsored by the DOE's Center for Exascale Simulation of Advanced Reactors (CESAR), and used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

References

[1] S. Dosanjh, R. Barrett, D. Doerfler, S. Hammond, K. Hemmert, M. Heroux, P. Lin, K. Pedretti, A. Rodrigues, T. Trucano, J. Luitjens, Exascale

design space exploration and co-design, Future Generation Computer Systems 30 (2014) 46 – 58.
[2] N. Attig, P. Gibbon, T. Lippert, Trends in supercomputing: The european path to exascale, Computer Physics Communications 182 (9) (2011) 2041 – 2046.
[3] N. Rajovic, L. Vilanova, C. Villavieja, N. Puzovic, A. Ramirez, The low power architecture approach towards exascale computing, Journal of Computational Science 4 (6) (2013) 439 – 443.
[4] J. Shalf, S. Dosanjh, J. Morrison, Exascale Computing Technology Challenges, in: 9th International Conference on High Performance Computing for Computational Science, Berkeley, CA, USA, 2011, pp. 1–25.
[5] C. Engelmann, Scaling to a million cores and beyond: Using light-weight simulation to understand the challenges ahead on the road to exascale, Future Generation Computer Systems 30 (2014) 59 – 65.
[6] F. B. Brown, Recent Advances and Future Prospects for Monte Carlo, Prog. in Nuc. Sci. and Tech. 2 (2011) 1–4.
[7] W. R. Martin, Challenges and Prospects for Whole-Core Monte Carlo Analysis, Journal of Nuclear Engineering Technology 44 (2012) 151–160.
[8] K. Smith, B. Forget, Challenges in the Development of High-Fidelity LWR Core Neutronics Tools, in: Proc. Int'l Conf. on Math. and Comp. Methods Appl. to Nucl. Sci. and Eng., Sun Valley, ID, USA, 2013.
[9] S. Yun, N. Z. Cho, Acceleration of source convergence in monte carlo k-eigenvalue problems via anchoring with a p-cmfd deterministic method, Annals of Nuclear Energy 37 (12) (2010) 1649 – 1658.
[10] E. R. Wolters, E. W. Larsen, W. R. Martin, Hybrid monte carlo-cmfd methods for accelerating fission source convergence, Nuclear Science and Engineering 174 (3) (2013) 286–299.
[11] B. Becker, R. Dagan, C. H. M. Broeders, G. Lohnert, An Alternative Stochastic Doppler Broadening Algorithm, in: Int'l Conf. on Math., Comp. Methods Appl. and Reactor Phys., Saratoga Springs, New York, 2009.
[12] B. Forget, S. Xu, K. Smith, Direct doppler broadening in monte carlo simulations using the multipole representation, Annals of Nuclear Energy 64 (2014) 78 – 85.
[13] X-5 Monte Carlo Team, MCNP - A General Monte Carlo N-Particle Transport Code, Version 5, Tech. Rep. LA-UR-03-1987, Los Alamos National Laboratory (Feb. 2008).
[14] P. K. Romano, B. Forget, The OpenMC Monte Carlo Particle Transport Code, Annals of Nuclear Energy 51 (2013) 274 – 281.
[15] P. K. Romano, B. Forget, Parallel fission bank algorithms in monte carlo criticality calculations, Nucl. Sci. Eng. 170 (2012) 125–135.
[16] P. K. Romano, A. R. Siegel, B. Forget, K. Smith, Data decomposition of monte carlo particle transport simulations via tally servers, Journal of Computational Physics 252 (2013) 20 – 36.
[17] P. K. Romano, B. Forget, K. Smith, A. Siegel, On the user of tally servers in monte carlo simulations of light-water reactors, in: Proc. Joint Int. Conf. on Supercomputing in Nuc. App. and Monte Carlo, Paris, France, 2013.
[18] H. Alme, G. Rodrigue, G. Zimmerman, Domain decomposition models for parallel monte carlo transport, The Journal of Supercomputing 18 (1) (2001) 5–23.
[19] R. Procassini, M. Obrien, J. Taylor, Load balancing of parallel monte carlo transport calculations, in: International Topical Meeting on Mathematics and Computations, 2005.
[20] T. A. Brunner, T. J. Urbatsch, T. M. Evans, N. A. Gentile, Comparison of four parallel algorithms for domain decomposed implicit Monte Carlo, Journal of Computational Physics 212 (2) (2006) 527 – 539.
[21] T. A. Brunner, P. S. Brantley, An efficient, robust, domain-decomposition algorithm for particle Monte Carlo, Journal of Computational Physics 228 (10) (2009) 3882 – 3890.
[22] J. Wagner, S. Mosher, T. Evans, D. Peplow, J. Turner, Hybrid and parallel domain-decomposition methods development to enable monte carlo for reactor analyses, Progress in Nuclear Science and Technology 2 (2011) 815 – 820.
[23] A. Siegel, K. Smith, P. Fischer, V. Mahadevan, Analysis of communication costs for domain decomposed monte carlo methods in nuclear reactor analysis, Journal of Computational Physics 231 (8) (2012) 3119–3125.
[24] A. Siegel, K. Smith, P. Romano, B. Forget, K. Felker, The effect of load imbalances on the performance of monte carlo algorithms in lwr analysis, Journal of Computational Physics 235 (2013) 901–911.
[25] N. Horelik, B. Herman, B. Forget, K. Smith, Benchmark for Evaluation and Validation of Reactor Simulations (BEAVRS), v1.0.1, in: Proc. Int. Conf. Mathematics and Computational Methods Applied to Nuc. Sci. & Eng., Sun Valley, Idaho, 2013.
[26] D. J. Kelly, T. M. Sutton, S. C. Wilson, MC21 analysis of the Nuclear Energy Agency Monte Carlo performance benchmark problem, in: PHYSOR – Advances in Reactor Physics – Linking Research, Industry, and Education, Knoxville, Tennessee, 2012.
[27] J. L. Hunter, T. M. Sutton, A method for reducing the largest relative errors in monte carlo iterated-fission-source calculations, in: Proc. Int'l Conf. on Math. and Comp. Methods Appl. to Nucl. Sci. and Eng., Sun Valley, ID, USA, 2013.
[28] D. Chen, N. A. Eisley, P. Heidelberger, R. M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. L. Satterfield, B. Steinmacher-Burow, J. J. Parker, The IBM Blue Gene/Q Interconnection Network and Message Unit, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, ACM, New York, NY, USA, 2011, pp. 26:1–26:10.
[29] J. E. Hoogenboom, W. R. Martin, A proposal for a benchmark to monitor the performance of detailed monte carlo calculation of power densities in a full size reactor core, Proc. Int. Conf. Mathematics, Computational Methods, and Reactor Physics, Saratoga Springs, New York, 2009.