# Development Of A Parallel Performance Model For The THOR Neutral Particle Transport Code

## M&C 2017 International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering

Raffi Yessayan, Yousry Azmy,
Sebastian Schunert

February 2017

# Development Of A Parallel Performance Model For The THOR Neutral Particle Transport Code

Raffi Yessayan[1*], Yousry Azmy[1], Sebastian Schunert[2]

[1]*North Carolina State University Department of Nuclear Engineering*
*3140 Burlington Engineering Labs, 2500 Stinson Drive, Raleigh, NC, 27695*
[2]*Idaho National Laboratory, Nuclear Science &Technology Directorate, Idaho Falls, ID*
[*]*rayessay@ncsu.edu*

**Abstract** – *The THOR neutral particle transport code enables simulation of complex geometries for various problems from reactor simulations to nuclear non-proliferation. It is undergoing a thorough V&V requiring computational efficiency. This has motivated various improvements including angular parallelization, outer iteration acceleration, and development of peripheral tools. For guiding future improvements to the code's efficiency, better characterization of its parallel performance is necessary. A parallel performance model (PPM) can be used to evaluate the benefits of modifications and to identify performance bottlenecks. The PPM development incorporates an evaluation of network communication behavior over heterogeneous links that are present in the utilized High Performance Computer (HPC) and a functional characterization of the per-cell/angle/group runtime of each major code component. After evaluating several possible sources of variability, this resulted in a communication model and a parallel component model. The former's accuracy is bounded by the variability of communication on the HPC while the latter has an error on the order of 1%.*

## I. INTRODUCTION

The THOR neutral particle transport code is being developed to allow for the simulation of complex geometries for a variety of problem types across the fields of reactor simulation and nuclear non-proliferation. It implements the arbitrarily high order transport method of the characteristic type (AHOTC) on unstructured, tetrahedral meshes [1].

Currently, THOR is undergoing thorough Verification and Validation (V&V) as part of its development process. The need for computational efficiency in this process has motivated a variety of improvements to the code including angular parallelization, outer iteration acceleration, and the development of peripheral tools [2]. It has also been recognized that, for guiding future improvements to the efficiency of the code, a better characterization of its parallel performance is essential.

By developing a parallel performance model (PPM) for predicting THOR's parallel execution time, one can better identify the benefits and detriments of various modifications to the code. This may be used to evaluate the true effectiveness of future modifications as well as to identify bottlenecks in existing routines. Additionally, by also characterizing the system on which THOR runs, one can identify if sources of efficiency loss are related to the code or to features of the host system.

As such, a PPM provides a powerful tool not just for the evaluation of the existing code, but as a benchmarking method for future, more significant modifications to THOR, such as parallelization with spatial domain decomposition (SDD). In addition, the obtained characteristic of the host system constitutes valuable information for other users of the same system, the system's administrators, and future HPC procurement.

## II. DESCRIPTION OF THE ACTUAL WORK

This work details the development of a PPM for THOR executed on a representative leadership-class HPC whose specifications will be detailed in Sec. II-2. This model is intended to characterize the behavior of the code for an arbitrary problem on an arbitrary configuration executed on the targeted HPC. However, both THOR and the utilized hardware introduce unique challenges in terms of model characterization.

### 1. Challenges to Modeling THOR's Parallel Performance

As is typical, THOR implements a variety of solvers. These include an outer iteration procedure for external source problems and (accelerated) power iterations as well as JFNK solvers for eigenvalue problems. For simplicity, the developed model only reflects the behavior of the un-accelerated power iteration solver. This solver was chosen as it is the most mature of the existing options and because it forms the basis for both of the non-JFNK accelerated solvers. Since the currently implemented accelerations have a relatively small impact on the runtime of any given iteration, it is logical that the timing model will hold, roughly, on a per iteration basis for all of the sub-implementations of the power iteration solver.

Furthermore, development of THOR's PPM provides a unique challenge due to the nature of its characteristic solver. Instead of solving a given arbitrary tetrahedral (tet) cell directly, THOR subdivides the cells into *canonical tetrahedrons* each having a single incoming and outgoing face. This allows a single code section to implement AHOTC on each of the canonical tetrahedrons comprising an arbitrary cell. While this simplifies the solving of each sub-cell, it introduces a degree of uncertainty in the total number of sub-tets in a given mesh as their number varies with quadrature angle being swept and with cell orientation. The 6 possible canonical tetrahedron decompositions are shown in Fig. 1. The first three correspond to the incoming direction of particle motion, $\widehat{\Omega}$, entering the tetrahedron on a face, entering on an edge, or exiting on a face, while the latter three correspond to $\widehat{\Omega}$ laying along a tetrahedron edge or face with differing numbers of exposed faces.

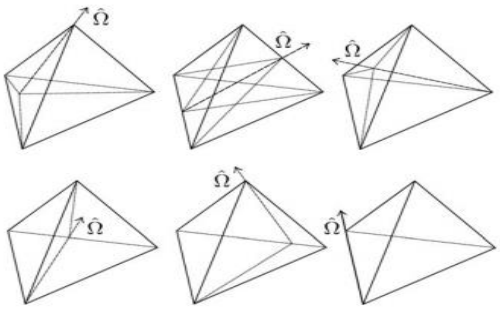

*Figure 1. Canonical tetrahedron decompositions during the THOR cell solve operation [1]*

After solving the characteristic equations in all of the generated canonical tets, the data from each is recombined into values representative of the entire cell. This mesh-cell level flux data on each outgoing face is then passed to the next downstream neighbor in the sweep operation and the single-cell solver starts over. This makes the cell-splitting process a black box. Outside of the single cell solver, all of the code components and transport operations only see the recombined mesh cells. The details of the characteristic cell solve are covered in detail in Ref. [1].

## 2. Challenges Resulting from HPC's Architecture

While larger systems would be required to truly test the efficiency of massively parallel schemes like SDD, the utilized HPC has proven an ideal testbed for the current developmental phase of THOR. It is sufficiently large that the serial and mid-scale parallel functions (10s-100s of processors) can be executed easily and frequently.

The utilized computing platform is a ~25,000 processor system built by SGI using a 7D enhanced Hypercube topology. Recent upgrades to the system have introduced a number of heterogenous compute nodes. To avoid complications from this, all work was done on homogenous allocations. The 7D enhanced Hypercube topology was developed by SGI to provide a high bandwidth low latency environment in the framework of a standard 7D hypercube architecture [3]. In addition to an interesting system architecture, the HPC implements a variety of hardware levels. Within the 7-dimensional hypercube, a single processor also belongs to a rack, a server, and a processor die. Each node has two processors of 12 (or, for the newer nodes, 18) cores each. This results in several different forms of communication. Two processing nodes located on the same die may communicate directly. Two nodes in the same server but on different chips may share a bus. Finally, two processors on completely different systems will be governed by the network communications interface. This hardware layout is common for large cluster systems and introduces a large number of unknowns regarding relative node locations in both physical and network space. Comparatively, on a personal computer or small server, parallel communication may be limited solely to on-die or on-board hardware communication buses.

Coupled with this hardware variation is the inability to request a specific subset of the system. Using a standard PBS scheduler, users may request exclusive use of a processor, a server, or a group of servers; but, the scheduling system assigns the resulting block of processors. To the authors' knowledge, the system makes no guarantee of locality or grouping during a standard allocation. So, when a communication network is established, the cost of traversing the links in the tree can vary as a function of network location in the full network hypercube. From a performance standpoint, the varying communication speed is never detrimental. The slowest communication path will yield the behavior expected by a network-interconnected hypercube. However, the uncertainty in the allocation can mean that code behavior is difficult to quantify with precision. A request for 6 processors each on 2 servers may yield any 6 of the 24 processors per server and 2 servers at any point in the hypercube. Even in this small case, there are multiple resulting combinations of on-die, on-board, and network communication and, consequently, difficult to predict latency and contention.

As quantifying this uncertainty in a model is nearly impossible, the model simply aims to determine whether the variance due to this phenomenon is significant when compared to both the aforementioned variability of the code and the total runtime of the other components.

## 3. The Parallel Performance Model

To develop the PPM, THOR was divided into 3 major sections – input, solver, and output. For any reasonably large problem, the read-in/write-out sections are negligibly small as they are not executed repeatedly. Hence, their contribution to execution time was ignored. For very large problems, like the ATR configuration in [2], the initial read-in and final write-out can be on the order of a half hour. Still, a single outer iteration's execution time is also on this order. So, the fraction of time consumed by I/O drops off with each iteration. However, as the number of processors increases, Amdahl's law states that the total run time of the code will approach that of the serial portion. This would include the serial I/O operations during initialization.

Within the solver section, the code was further subdivided into five major sections that represent the code's primary logical functions; these are nested as shown in Fig. 2.

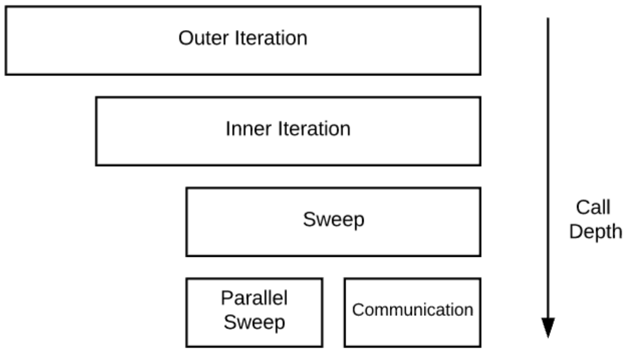

*Figure 2: Timing model's logical components*

Using the shown structure, the total run-time of each component is given as the sum of the run-time of its child component and the operations executed in that routine. At the lowest levels of the call tree, the sweep operation is subdivided into two parts to account for the angular domain decomposition (ADD) parallelization.

The parallel sweep contribution accounts for the time taken to sweep over $m$ ($m$+2) angles, with vacuum boundary conditions, in an $S_m$ quadrature set each of which comprises a sweep over $N$ tetrahedrons using $p$ processors. The time consumed for each cell/angle combination solve is referred to as the grind time. The communication portion represents the two spanning-tree communications used at the end of the sweep operation to accumulate flux angular moments and distribute their values to all processors. The AHOTC formalism comprises an arbitrary-order expansion of the flux variables within each cell and its bounding faces [1]. In this work, we consider the zero spatial-expansion order option in THOR but extension of the PPM to higher order expansions should be rather straightforward. Furthermore, all timing measurements were made using a one group problem under the assumption that a $g$ group problem's execution will scale with $g$. In problems with a very large number of groups, this assumption may not hold. But, it should also be a straightforward task to modify the model with a factor for time growth in terms of $g$. Also, THOR does not implement any parallelization in energy at this time due to that parallelization's asynchronicity, which is likely to increase the number of iterations thereby adversely affecting parallel efficiency.

Based on this description, we propose the timing model for a single outer running a single inner iteration:

$$T_{solve} = \tau_{\text{const}} + N * (\rho * [\tau_{outer} + \tau_{inner}] + \tau_{sweep} + \frac{m*(m+2)}{p} * \tau_{parallel}) + f(\tau_{comms}, p), \quad (1)$$

where $N$ is the number of cells, $\rho$ is the number of angular moments, and $\tau$ represents the time constant of each major function block exclusive to that routine (i.e. $\tau_{outer}$ is the time spent in the outer iteration that is not in the inner iteration). For multiple inner iterations, all values except $\tau_{outer}$ would be multiplied by the number of inner iterations.

The parallel sweep routine demonstrates the effects of parallelization. In serial codes, one can expect the sweep work to scale linearly with the number of cells and angles. Here, as a result of implementing ADD, the dependence on number of angles is modified. Now, instead of scaling simply with the number of angles, one also can expect to see a 1/$p$ relation. This represents the process of distributing the work across processors based on the angle. However, if $p$ is not a factor of the number of angles, work is unevenly assigned. In this case, this code section's execution time will behave as if $p$ were lowered to the nearest multiple of $m$($m$+2), i.e. $\left\lceil \frac{m(m+2)}{p} \right\rceil$.

Finally, there is the function representing the communication time component. This component is dependent not only on the implementation of the code, but also on the architecture of the system the code is running on and the topology of the subsystem assigned at run time for executing the specific case. A proposed model for this behavior that recognizes the underlying hypercube topology of the HPC is given by:

$$f(\tau_{comms}, p) = 2 * \log_2(p) * \left(\tau_{comms} + \frac{2N*\rho}{\beta}\right) \quad (2)$$

Here, $\beta$ is the system bandwidth in words/s and $\tau_{comms}$ is the time to initialize communication, i.e. communication latency. As discussed before, these constants will be highly dependent on the processor allocation at runtime. As such, it is unlikely that generally applicable explicit values of these parameters can be extracted. The $\log_2(p)$ represents the number of send components in a tree based communication system. This is the system implemented for communication in this topology. Next, as the communication functions are 2-way (i.e. they send and receive data), the latency and data size are doubled. The ratio of $N$ to the bandwidth gives the time consumed in actual data transfer. The entire equation is multiplied by two to represent the two communication operations that occur per iteration. These two operations are used to accumulate the scalar flux calculated on each processor and then to redistribute it back amongst all processors.

## III. RESULTS

Timing results were obtained for the communication variation, canonical cell variation, and major parts of the timing model. Data was collected by running one of THOR's standard test problems, a simple cube with various levels of mesh refinement with $S_2$ through $S_{16}$ quadrature. Where timing data was collected, each timing case was run 5 times and averaged to produce an approximation to the expected value. Additionally, for the evaluation of the variance in the number of canonical tetrahedrons, the C5G7 [4] and Godiva [5] benchmark configurations were used. After evaluation of the model coefficients, the PPM was validated against configurations and problems not included in the original measurement set used in estimating the model parameters.

### 1. Canonical Tetrahedron Variation

Based on the orientation of a tetrahedral cell and the incoming angle, any arbitrary tetrahedron can be subdivided into 2, 3, or 4 canonical tetrahedra [1]. These configurations were shown in Fig. 1. Because of this variability, it is theoretically possible for two meshes featuring the same number of cells to exhibit differing workload for sweeps along different angles and for a single mesh to feature varying workloads between regions with identical number of cells and also between sweeping directions.

To quantify this behavior, the simple cube test problem [1] was solved at 4 different mesh refinement levels and 5 different quadratures. This problem features a cubic domain with vacuum boundaries and a single energy group. Table 1 summarizes the results for these cases.

As can be seen, the degree of variation between the cases is negligible. All the cases demonstrate an average number of canonical tetrahedrons around 3.67. However, there does appear to be a slow upward trend as the number of angles increases. Regardless, as long as the variation stays on the order of $10^{-4}$ or $10^{-5}$, it is unlikely that it will contribute significantly to imprecision in the performance model.

While the reason for the extremely low variance is not known, we conjecture that it may result from the simple configuration of this test problem. The simple cube test presents a very regular geometry with a good aspect ratio. These factors could result in very little variation between input configurations. Additionally, given the regularity of the domain, it is likely that mesh-refinement produces similar results in all regions.

*Table 1: Simple Cube Test - Canonical Tet Variation*

| # Angles | # Cells | Avg. Subcells | Std. Dev |
|---|---|---|---|
| 8 | 8,859 | 3.66836 | 0 |
| 80 | 8,859 | 3.66836 | |
| 288 | 8,859 | 3.66836 | |
| 8 | 151,562 | 3.66812 | 1.53E-05 |
| 80 | 151,562 | 3.66814 | |
| 288 | 151,562 | 3.66815 | |
| 8 | 194,332 | 3.66802 | 1.15E-05 |
| 80 | 194,332 | 3.66804 | |
| 288 | 194,332 | 3.66804 | |
| 8 | 426,885 | 3.66813 | 4.73E-05 |
| 80 | 426,885 | 3.66820 | |
| 288 | 426,885 | 3.66822 | |

To address this concern, a small number of supplemental cases were run using an unfolded version of the 2x2 assembly C5G7 3D Benchmark. The unfolding was done to convert the reflective boundary conditions present in C5G7 to vacuum boundaries. The resulting mesh has approximately 20 million tetrahedrons and a much more complicated geometry than the cube test problem. However, only a single mesh was available for testing with two angular quadratures as reported in Table 2. Additional testing was performed using a very small model of the Godiva benchmark. These results are given in Table 3.

*Table 2: C5G7 - Canonical Tet Variation*

| # Angles | Mesh Size | Avg. Subcells | $\Delta Avg$ |
|---|---|---|---|
| 8 | 20617414 | 3.618 | |
| 24 | 20617414 | 3.640 | .022 |

In Table 2, we see that the difference even between the $S_2$ and $S_4$ quadratures is significantly larger – on the order of one percent. This is still a relatively small effect, compared to for example the effect on execution time from acquiring different processors at run time. But, it does indicate that there is the possibility for greater degrees of variation between cases than suggested by the simple cube case.

*Table 3: Godiva - Canonical Tet Variation*

| # Angles | Mesh Size | Avg. Subcells | $\Delta Avg$ |
|---|---|---|---|
| 8 | 274 | 3.66836 | 0 |
| 80 | 274 | 3.66836 | |
| 288 | 274 | 3.66836 | |

The Godiva mesh does not show the strong fluctuation present in the C5G7 mesh. Instead, it is more akin to the results obtained in the simple cube test. As these two tests are homogenous systems they are meshed into more regular tets across the entire geometry; so, it is likely that the fluctuations present in C5G7 are a result of mesh behavior along material boundaries. This would indicate that highly heterogeneous configurations or those with realistically shaped material interfaces may result in "biased" meshes and cause the average work to drift with mesh refinement or increasing number of angles. This would have to be evaluated on a case by case basis and used to modify the grind time to ensure that the PPM remains applicable.

## 2. Communication Time Variation

As stated previously, communication on the utilized computing platform can be subdivided into two components, a communication tree building time and a data-transfer portion [2]. Below about $10^5$ words, the tree-building time dominates. The tree building time increases linearly with $\log_2(p)$ and the communication time increases linearly with the data size. This behavior is shown in Fig. 3. The figure was made using data from an external routine which implements MPI AllReduce operations with sizes similar to those found in THOR. Across all data sizes and processor counts, the total communication time can be decomposed into two parts. The first is a constant time region which scales with the size of the binary tree used in the communication. The second is the region in which time increases linearly with the size of the data being transmitted.

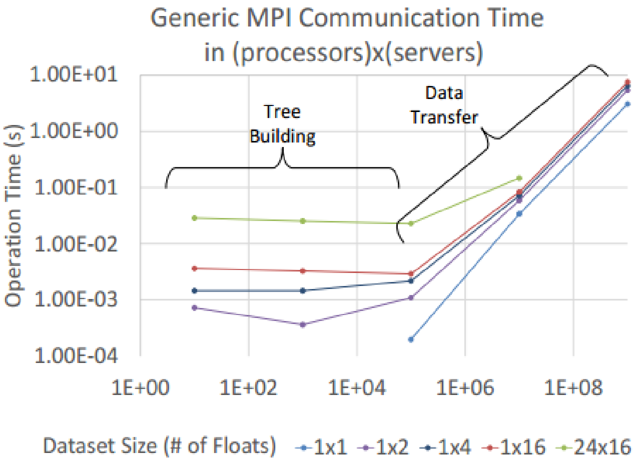

*Figure 3: Generic behavior of the HPC during MPI AllReduce operations [2]*

To evaluate this behavior for THOR, the total communication time was measured for processor counts of 1, 2, 4, 16, and 64 and mesh refinement levels of ~8,000, ~150,000, and ~200,000 tets on the simple cube problem. As expected, for the ~8,000 cell case, total communication times on the order of $10^{-3}$s to $10^{-2}$s were observed. These align well with the tree construction times shown in Fig. 3. For the more refined meshes, the results also fall in line with the data predicted by the communication testing. Unfortunately, even for the most refined case, the number of cells is still relatively small. This results in all the experimental cases occupying a very small region of predicted behavior.

The time variation between repetitions of identical cases was calculated. As expected, the run-time standard deviation between runs on the same processor allocation is very low, often on the order of $10^{-4}$ or $10^{-5}$s. However, occasional cases were seen where the standard deviation was on the order of 5-10% of the measured time. Rarely, more extreme spikes resulted in run to run differences of an order of magnitude. These outliers are likely the result of network contention and are unlikely to occur repeatedly over the course of a long-running problem. However, in the event that the system is heavily loaded, it is possible to see an unpredictable increase in the cost of communication.

Having established that communication times are generally consistent on constant allocations, the cases were rerun on several different allocations. Since, as discussed, one cannot request a specific (or different) allocation, this was accomplished by rerunning the cases over the course of several days. The presence of other users' jobs should guarantee that available allocations vary with time. Initially, under these conditions, it was observed that measured communication times were highly inconsistent between allocations. This resulted in not just varying communication times, but varying trends in communication times. To highlight this variability, two cases are presented in Figs. 4 and 5. The blue line represents an allocation large enough to run $S_{16}$ ($p$ = 288) regardless of the quadrature of the current case, while the orange line is an allocation where $p = m(m+2)$ for each attempted m value. The values along the horizontal-axis are the $ceil(\log_2(p))$ and correspond to the depth of the binary tree used in the reduce operation.

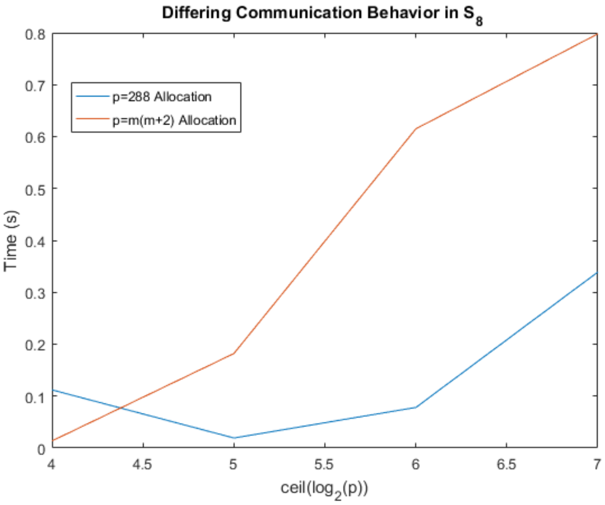

*Figure 4: Differing communication trends for p>m(m+2) and p=m(m+2) for $S_8$*

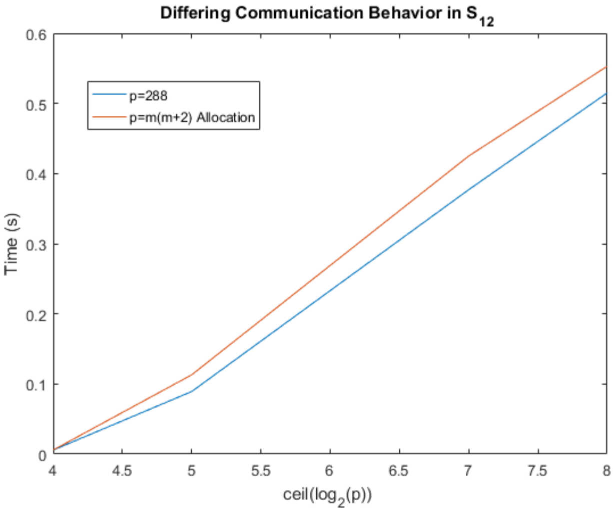

*Figure 5: Differing communication trends for p>m(m+2) and p=m(m+2) for $S_{12}$*

It is likely that the change in trend in Fig. 4 results from the 3 different loading patterns that must be considered when allocating processors. For the full $S_{16}$ cases, 288 processors were allocated and $p$ was selected to be a factor of *m(m+2)* for each case (maximizing efficiency). The resulting communication tree only depends on processors active in the case. As such, since $p$ is likely not a power of two, the tree has empty leaves. Still, all possible leaf positions are still the same distance from the root node. However, the allocated network sub-hypercube is sized based on the allocation. This means that processors assigned to the program do not necessarily come from the smallest encapsulating sub-cube and that distance from root can vary by multiple hops.

Once the number of processors is explicitly allocated to be $m(m+2)$, the network uncertainty is reduced. There are still various possible configurations, but they all exist in the same dimensionality. As can be seen above, this results in the better behaved trends shown in orange.

Yet, as the $S_{12}$ case shows, there is still some degree of drift in communications time between allocations. Even when changing the allocation size does not dramatically change the shape of the communication trend, there are still frequently changes in the average time of any given case. This is demonstrated by all cases and likely results from day to day changes in system loading/contention.

Based on a subset of the data collected with $m(m+2)$ sized allocations, an approximate model of the communication was developed. As shown in Fig. 6, the fit demonstrates the $\log_2 p$ dependence predicted in the theoretical parallel performance model (Eq. 2). However, there are still significant outliers both at very low $p$ and at various points throughout the curve.

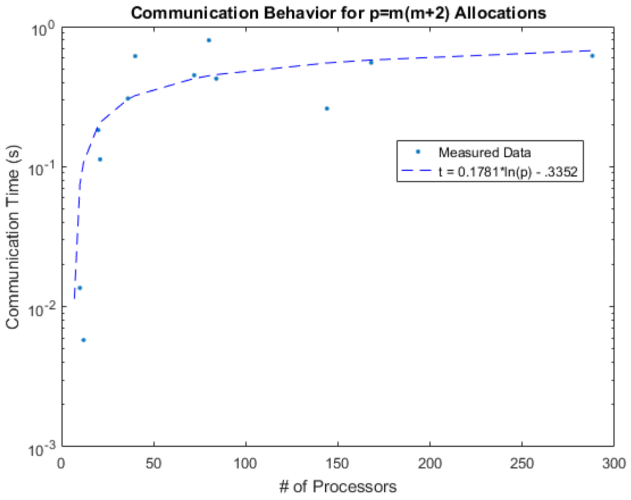

*Figure 6: THOR/HPC communication time fit*

## 3. Developing the Parallel Sweep Model

The innermost evaluated function is the parallel sweep routine. By proper selection of $p$ this routine divides the sweep operation evenly among processors based on the number of angles. Hence, it is expected that the execution times will exhibit a $1/p$ behavior while $p \leq m(m+2)$. To evaluate this, a set of cases was selected. These included combinations of $S_m$ $m$ = 2, 4, and 6 for the simple cube problem with mesh sizes of ~8k, 150k, and 200k tets. The Level-Symmetric quadrature set was used with $M = m(m+2)$ angles. A selection of these cases is shown in Fig. 7 with the dashed lines representing the regions where $p \leq m(m+2)$. Data was collected at $p = 1, 2, 4, 16, 64$.

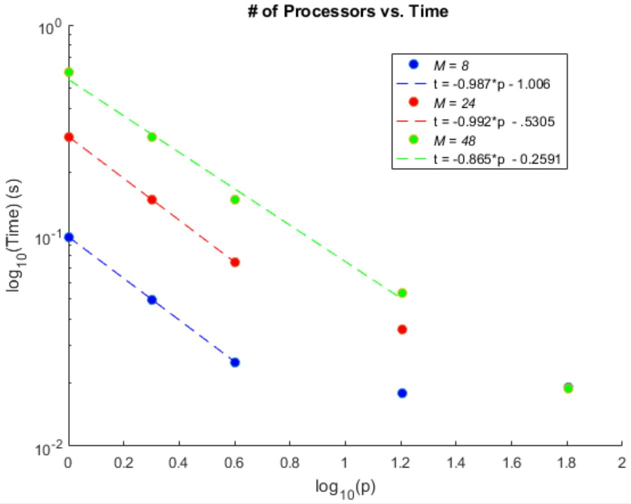

*Figure 7: 1/p relationship between processor count and parallel sweep time*

As expected, this plot shows a strong *1/p* trend where the number of processors is less than or equal to the number of angles. If there are more processors than angles, then some processors will sit idle and not contribute to parallel speedup. This results in the trend flattening out for higher *p*.

Further information can be extracted from the plot by analyzing the amount of work performed for any value of *p* as the number of angles changes. This shows the dependence of the execution time on increasing number of angles. Since the total amount of work scales approximately linearly with the number of angles, one would expect to see a linear relationship between execution time and number of angles for each *p*. Additionally, the ratio of the slopes between any two sets of points should yield the ratio of the number of processors used as depicted in Fig. 8.

From these values, we can extract the grind time. This is done by evaluating the time per cell per angle, a value we have defined as the grind time.

$$grind\ time = \frac{time(m\ angles,\ \ N\ cells\ )}{m*N} \qquad (3)$$

Equation 3 was evaluated using both sets of data plotted in Fig. 8 for angular dependency as well as with the equivalent calculation for cell-count dependency. These methods yielded grind times of ~2.30E-6 and ~2.33E-6, a difference of ~1%. For the sampled cases, the variance in runtime between case repetitions was typically ~0.5%.

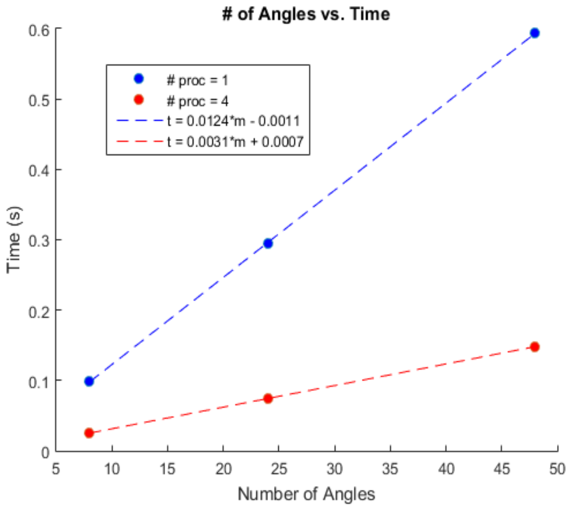

*Figure 8: Linear relation between time and number of angles for a fixed value of p*

Finally, the evaluated grind time was used to predict the data points used to generate the model. Since the variance in sweep times was rather small, the predictions were quite good – on the order of single percent error. Since this error sets the baseline model error, it should be as low as possible. Evaluating the runtime of other cases can never be expected to be higher accuracy than this.

## 4. Evaluating the Parallel Sweep Model

Having established a model for parallel sweeping using a limited subset of cases, it is necessary to validate the model for general cases not included in its own development. These tests can be broken down into 3 cases – interpolation, extrapolation, and external. Interpolation cases are those whose parameters are within the envelope defined in the previous section and which use the same geometry. Extrapolation uses the same geometry but parameters outside the envelope (i.e. higher order quadratures and mesh refinements). Finally, external problems use a different geometry at a wide set of parameter configurations. Based on these descriptions, one would expect that the interpolation results will have errors comparable to those used to develop the model.

First, for the interpolation cases, no other mesh refinements were available. And, as $S_{2\text{-}6}$ had been used, there were no unused quadratures in the interpolation set. To address this, a set of edge cases were selected instead. These edge cases are in the interpolation envelope in quadrature, but not in mesh refinement. They use a 500k tet refinement of the simple cube mesh. Since the fits in figures 7 and 8 are very clean, these cases were expected to conform to the baseline error established for the model cases. For all 3, the measured error reported in Table 4 was only slightly higher than that seen in the model cases. This satisfied the interpolation accuracy of the model.

*Table 4: Percent Error for Interpolation Cases*

| Case | Meas. Time | Model Time | Difference |
|---|---|---|---|
| *p=4* | (s) | (s) | (%) |
| S2, 500k tet | 1.53 | 1.5 | 1.9% |
| S4, 500k tet | 4.61 | 4.51 | 2.3% |
| S6, 500k tet | 9.2 | 9.02 | 2.1% |

Next, a series of extrapolation cases were tested using $S_{12}$ and $S_{16}$ for the 3 mesh refinements used earlier and $S_2$-$S_{16}$ for a 500k tet mesh refinement. As the number of processors, angles, and cells increases, any terms missing from the model will become increasingly evident. This is a critical step as the original model envelope used rather small problems to establish a trend. Figure 9 shows the full set of quadrature cases for the 500k tet mesh with lines representing the model and markers representing the measured parallel component times.

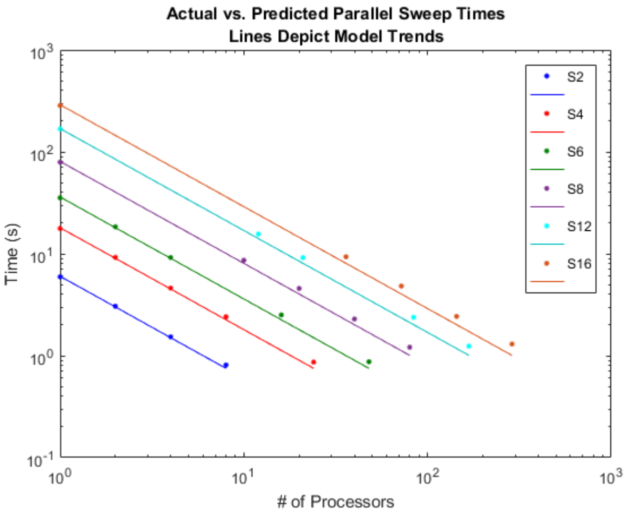

*Figure 9: Extrapolation cases on a 500k tet mesh*

Immediately, it is evident that there is a loss of model accuracy with increasing processor counts. This effect was suppressed in the earlier original development due to the low quadrature and correspondingly low numbers of processors. What had looked like ~2% error resulting from runtime noise was the beginning of a much larger deviation.

By evaluating up through $S_{16}$ on a larger mesh problem, several important details could be inferred. First, the error term continues to grow with $p$ (reaching almost 25% in Fig. 9); and, second, the error is independent of the quadrature. This can be determined by evaluating a "column" of points in the plot. For example, looking at the first data point right of $p$=64 for $S_8$, $S_{12}$, & $S_{16}$, one can observe that the relative error of each trace is approximately the same, about 20%. This relation holds true for all "columns" in the plot. Taken together, these two details indicate that the model is missing a term that grows purely in $p$.

To test this hypothesis, the difference between the model and actual data in Fig. 9 was converted to a $\Delta grind\ time$ and plotted against $\log_2 p$. As shown in Fig. 10, the resulting trend is highly linear. This indicates that a purely processor count dependent change in grind time is present and needs to be corrected globally in the model.

Based on these results, the grind time was modified from that shown in Eq. 3 to one which includes an error term with a processor count dependency.

$$grind\ time = \frac{time(m\ angles,\ N\ cells)}{m*N} + f(p) \tag{4}$$

As shown in Fig. 11, this modification greatly reduces the errors present in Fig. 9, but it also raises some concerns regarding the efficiency of the code. These concerns will be discussed in the conclusions section. This growth in grind time is likely the result of an inefficiency in the code which is causing unnecessary communication or access to a shared network resource, like file I/O.

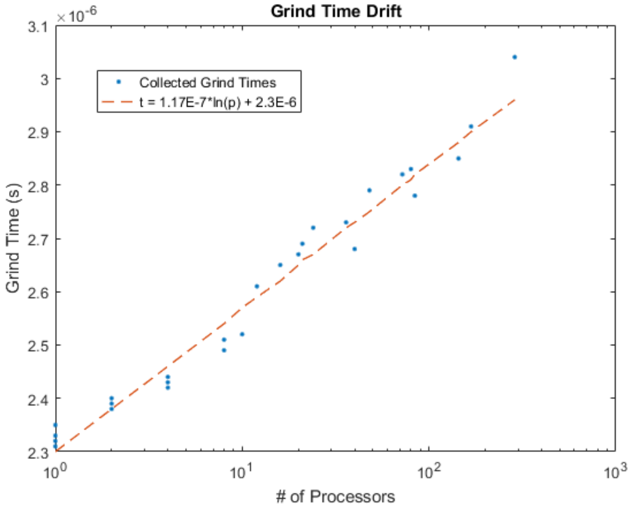

*Figure 10: Per processor change in grind time*

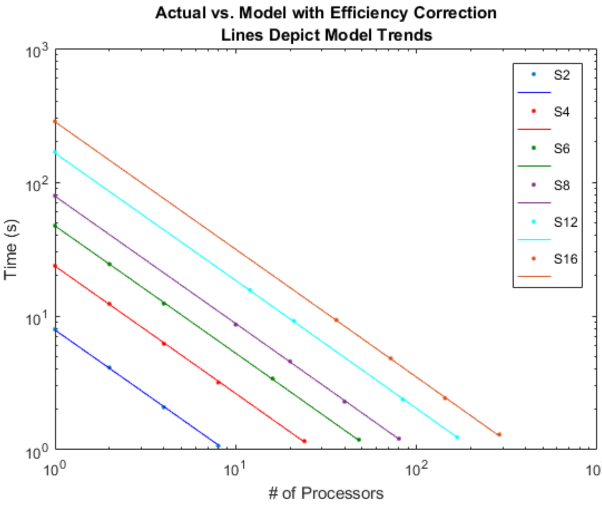

*Figure 11: Updated model extrapolation cases on a 500k tet mesh*

With the correction applied, see Fig. 11, the error drops from a maximum of ~25% to a maximum of ~3% for all the cases shown in Fig. 9. This correction can be further applied to the interpolation and foundational cases to improve their accuracy to similar levels. However, since the number of operations is much smaller in those cases, the total effect of the correction is smaller.

Finally, the model was used to predict a set of external cases. For these, the Takeda-IV [6] and Godiva benchmarks were selected. These benchmarks provide both a homogenous and heterogeneous material test case on a geometry dissimilar from the one used in constructing the model.

The range of quadratures was run against a ~15k tet Takeda-IV mesh and a ~3k tet Godiva mesh. As this marks a step away from the previous model problems both in mesh size and geometry, it is expected that the results will not be as clean as the ~3% error seen previously. However, as the solver treats all power iteration problems in the same fashion, the model should still serve as an accurate predictor of problem runtime.

*Table 5: Actual and Model Results for External Tests*

| Case | | Time (s) | Model Time (s) | Difference (%) |
|---|---|---|---|---|
| Godiva, 3k tet | | | | |
| S8 | $p$=40 | 1.17E-02 | 1.36E-02 | 16% |
| | $p$=80 | 5.88E-03 | 6.78E-03 | 15% |
| S12 | $p$=84 | 1.24E-02 | 1.36E-02 | 10% |
| | $p$=168 | 6.28E-03 | 6.78E-03 | 8% |
| S16 | $p$=144 | 1.24E-02 | 1.36E-02 | 10% |
| | $p$=288 | 6.40E-03 | 6.79E-03 | 6% |
| Takeda, 15k tet | | | | |
| S8 | $p$=40 | 6.71E-02 | 7.28E-02 | 9% |
| | $p$=80 | 3.44E-02 | 3.65E-02 | 6% |
| S12 | $p$=84 | 6.80E-02 | 7.29E-02 | 7% |
| | $p$=168 | 3.55E-02 | 3.65E-02 | 3% |
| S16 | $p$=144 | 7.04E-02 | 7.30E-02 | 4% |
| | $p$=288 | 3.70E-02 | 3.65E-02 | 1% |

As is shown in Table 5, the PPM predicts the parallel runtime of these two problems reasonably well. Even the shortest measured time is well above the clock precision, 1E-6 s, as reported by MPI_Wtick. Discounting the smallest Godiva case, the maximum error across the two problems is about 10%. However, for the Takeda-IV cases, the error is often equivalent to that seen in the extrapolation cases, ~1-4%.  This difference in prediction error, especially as the problems grow smaller, may result from several sources such the relative memory footprints of the two problems or hardware effects. The Takeda mesh is about double the size of the smallest mesh used to establish the model, while the Godiva mesh is almost three times smaller. Due to the very small size of the problem, the Godiva mesh could be benefitting from caching effects. Regardless of these effects, the model provides a good estimate of the runtime for problems outside of the original model set. Additionally, the accuracy is highest in high processor count cases. As it is most effective to use THOR’s angular domain decomposition with $p = m(m+2)$ , this is the more relevant region for high accuracy estimates.

## 5. Evaluating the Combined Model

With the communication and parallel operations modelled, the vast majority of the work in any given iteration has been quantified. The remaining work in the inner, outer, and sweep operations is almost entirely composed of variable management and I/O, both to the screen and files. It was determined that, were the I/O to be removed, the remaining work would be negligible. And, the I/O behavior is rather small (e.g. printing a one line summary to the screen / a file). Based on this, it was decided to mark the constants of the remaining sections as negligible and to roll the resulting time differences into the error already present in the model. This avoids having to characterize time contributions, like file access time, which are dependent on a huge number of hardware and network parameters. This gives an effective PPM expression of:

$$T_{solve} \approx \frac{m*(m+2)}{p} * N * grind\ time(p) + f(comm) \qquad (5)$$

Where the communication function is given by the fit in Fig. 6 and the grind time is 2.3E-6 s plus the processor based error given in Fig. 10. For problems with large numbers of groups, a term may need to be added to address the increasingly non-negligible cost of constructing the group sources.

With the notable exception of the processor dependent grind time, this model matches the form of the one proposed at the beginning of this paper. Several of the parameters proved negligible or inseparable, however the general behavior with regards to *N*, *m*, and *p* is preserved.

As a final test of the model, the communications model and parallel sweep model were integrated to create a general parallel performance model. This model was then compared to several cases of the simple cube test to show that the two major components are accurate and represent the total run time of a THOR outer iteration.

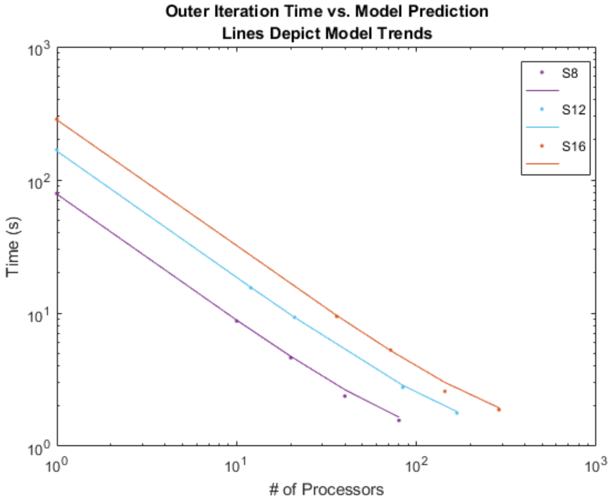

*Figure 12: Comparing outer iteration measured time to model time from Eq (5)*

As figure 12 shows, the time represented by the Eq (5) model is a very good fit for the total time of an outer iteration. This result does benefit somewhat from cancellation of errors. In the low $p$ regime, communication time is often poorly estimated. But, it has a very small contribution to the problem. Elsewhere, overestimation of the communication time can negate errors incurred by not explicitly accounting for some operations in the outer and inner loops.

## IV. CONCLUSIONS

As this paper has shown, a parallel performance model can be a powerful tool for characterizing the behavior of complex codes such as the THOR deterministic transport code and the computing platform they are executed on. As the capabilities of THOR grow, it becomes increasingly important that each addition to the code is well characterized and well implemented. The addition of a single inefficient routine can hamper the efficiency of the entire solver. To address this need, the PPM can be used as a tool for both characterization and evaluation. Any piece of code can be functionally characterized based on its run-time behavior and it can be evaluated by comparing the resulting functional description to the description expected by a theoretical model.

The PPM developed here does a good job of characterizing the behavior of THOR over a broad spectrum of input configurations. However, due to the multiple sources of uncertainty found in communication and system level factors, there is an irreducible amount of noise. Yet, even given this limitation, the model shows percentage point order agreement (~1-3%) for both interpolated and extrapolated cases on the simple cube mesh and ~5-10% error on the Godiva and Takeda-IV meshes.

The PPM has already been applied in a preliminary evaluative role as well. As shown in the results section, the final model was modified with a $p$ term in the grind time. While this does not invalidate the model, it does mark a deviation from the expected behavior. This deviation indicates the possibility of sub-optimal or inefficient implementation in the code and narrows down both the location and type of the possible problem. Work is ongoing to identify and, if necessary, correct the exact source of this divergence.

Identifying these opportunities for improvement requires an understanding of both the theoretical and actual behavior of the code. But, as identified issues are addressed, the actual behavior should begin to align increasingly closely with the theoretical. This provides a metric by which code implementation and efficiency can be verified.

The PPM discussed in this paper is a foundational tool designed to model one of THOR's primary solving routines. For it to remain applicable, the PPM must grow alongside THOR and represent new functionality as it is added. This process can be done modularly. As was done here, each new piece of functionality can be modeled individually and then integrated into the total model. Ideally, this process will allow for modeling and evaluation of code before it is finalized into THOR. In this way, the PPM serves as another integration test intended to keep the THOR codebase as efficient and well implemented as possible.

## ACKNOWLEDGMENTS

This material is based upon work by the first two authors (RAY, YYA) supported by the Department of Energy National Nuclear Security Administration under Award Number(s) DE-NA0002576.

## REFERENCES

1. R. FERRER, Y. AZMY, "A Robust Arbitrarily High-Order Transport Method of the Characteristic Type for Unstructured Grids," *Nucl. Sci. and Eng.*, **172**, 33 (2012)
2. R. YESSAYAN, Y. AZMY, S. SCHUNERT, "Verification and Validation of the Tetrahedral Radiation Transport Code THOR based on the Advanced Test Reactor Benchmark," *Proc. PHYSOR 2016*, Sun Valley, Idaho, May 1–5, 2016, American Nuclear Society (2016).
3. R. WOLFF, "Nasa Pleiades Infiniband Communications Network", High Performance Distributed Computing; http://www.hpdc.org/2009/PDF/vendor_sgi.pdf (current as of Feb. 17, 2017)
4. E. LEWIS, "Benchmark specification for Deterministic 2-D/3-D MOX fuel assembly transport calculations without spatial homogenization (C5G7 MOX)," NEA/NSC-2001, Nuclear Energy Agency (2008).
5. D. MOSTELLER, J. GODA, "Analysis of Godiva-IV Delayed-Critical and Static Super-Prompt-Critical Conditions," Proc. M&C 2009, Saratoga Springs, NY, May 3-7, 2009
6. T. TAKEDA, H. IKEDA, "3-D Neutron Transport Benchmarks," Journal of Nuclear Science and Technology, **28**, 7 (1991)