ITERATIVE AND PARALLEL PERFORMANCE ANALYSIS OF NON-BLOCKING COMMUNICATION ALGORITHMS IN THE MASSIVELY PARALLEL NEUTRON TRANSPORT CODE PIDOTS

Raffi Yessayan[1], Yousry Y. Azmy[1], and R. Joseph Zerr[2]

[1]North Carolina State University
Department of Nuclear Engineering
2146 Burlington Engineering Laboratory
Raleigh, NC, USA, 27603

[2]Los Alamos National Laboratory

rayessay@ncsu.edu, yyazmy@ncsu.edu, rzerr@lanl.gov

ABSTRACT

The PIDOTS neutral particle transport code utilizes a red/black implementation of the Parallel Gauss-Seidel algorithm to solve the S_N approximation of the neutron transport equation on 3D Cartesian meshes. PIDOTS is designed for execution on massively parallel platforms and is capable of using the full resources of modern, leadership class high performance computers. Initial testing revealed that some configurations of PIDOTS's Integral Transport Matrix Method solver demonstrated unexpectedly poor parallel scaling. Work at Idaho and Los Alamos National Laboratories then revealed that this inefficiency was a result of the accumulation of high-cost latency events in the complex blocking communication networks employed during each PIDOTS iteration. That work explored the possibility of minimizing those inefficiencies while maintaining a blocking communications model. While significant speedups were obtained, it was shown that fully mitigating the problem on general-purpose platforms was highly unlikely for a blocking code. This work continues that analysis by implementing a deeply interleaved non-blocking communication model into PIDOTS. This new model benefits from the optimization work performed on the blocking model while also providing significant opportunities to overlap the remaining un-mitigated communication costs with computation. Additionally, our new approach is easily transferable to other similarly spatially decomposed codes. The resulting algorithm was tested on LANL's Trinity system at up to 32,768 processors and was found at that processor count to effectively hide 100% of MPI communication cost – equivalently 20% of the red/black phase time. It is expected that the implemented interleaving algorithm can fully support far higher processor counts and completely hide communication costs up ~50% of total iteration time.

KEYWORDS: Deterministic Transport, Massively Parallel, Non-Blocking Communication, MPI

1. INTRODUCTION

Designing codes to fully utilize contemporary leadership class high-performance computers (HPC) requires the implementation of highly parallel, distributed or hybrid memory programs. While these codes gain access to dramatically increased computing resources, they also become more vulnerable to factors beyond the control of an end user. MPI communication among processes requires passing messages across the network interconnect. There, they can encounter a variety of issues including bandwidth limitations, resource contention with other programs, and varying communication costs based on allocation diameter. As many of these factors are dependent on both the type and location of other programs in the system, they are nearly impossible to fully isolate. Instead, for an end user, they must be treated as a combined, external, and unpredictable contribution to the communication cost. Previous work has shown that these costs, while minor for any one message, can become hugely impactful on the overall performance and efficiency of a massively parallel code [1]. Without mitigation, these impacts can quickly cap the maximum number of viable processors for a given code, thereby limiting the highest achievable speedup factor.

For a code developer, two primary MPI communication models are available– blocking and non-blocking. In a blocking model, messages do not resolve for either the sender or receiver until the message is fully received. This implicitly guarantees ordering between tasks, even across machines. However, it also more deeply couples the steps within those tasks. A slowdown on one processor will be propagated to another due to the synchronous nature of the tasks. In the non-blocking model, the dispatch and receipt of a message are decoupled. If the recipient of a message possesses sufficient work prior to receiving the message, this allows for the tasks to operate more independently and prevents some slowdowns from propagating. By overlapping or interleaving the communication and computation steps, portions of the total communication cost can be hidden. In general, a blocking model is easier to implement while a non-blocking model has better potential for parallel performance if an interleaved algorithm is possible. However, it is critically important to note one difference between the two communication models. While both decrease end user run time, improvements to a blocking routine *improve* communication performance but additional non-blocking interleave *hides* communication costs. This means that the best scalability comes from the combination of optimized algorithmic performance and increased available interleaving.

1.1 PIDOTS Code Overview

The PIDOTS code [2] is designed as a testbed for the implementation of high-performance, massively parallel algorithms. It is a time-independent, 3D, Cartesian mesh, mono-energetic transport solver using a red/black Parallel Gauss-Seidel implementation of the Integral Transport Matrix Method (ITMM) to solve the S_N approximation of the neutron transport equation. This code allows for highly distributed, spatially decomposed meshes with a fine grain size. These problem configurations are ideal for testing the performance of a massively parallel routine before implementing it in a more complex environment.

To implement its spatially decomposed solver, PIDOTS relies on a three-level spatial decomposition scheme. This allows the global problem domain to be subdivided amongst processors and then further subdivided into red/black solve domains. A 2D analogue of this decomposition is shown in Figure 1 for a R/B-4, $p = 4$ configuration. In that example, a single problem domain is decomposed amongst $p = 4$ processors and each processor is split into 4 red/black domains per axis. This second refinement is referred to as the problem's R/B-#. As ITMM requires matrix operations, PIDOTS tends to favor high R/B-# configurations for solves.

During the initial verification of the PIDOTS code [2], several unexpected parallel efficiency issues were observed. While these did not impact the accuracy of the solve, they did indicate that significant unidentified costs were being incurred during the communication phase. Later work, performed at Idaho National Lab (INL), established that these unexpected efficiency losses were a result of the cost of latency spikes accumulating through the many blocking communications in a PIDOTS iteration. This made the original blocking algorithm highly vulnerable to unpredictable latency spikes. Measurement of system latency on INL's Falcon system was used to drive a discrete event simulation of PIDOTS's iterations which showed an up to 4x performance loss as a result of the blocking latency chains [1]. Further work performed at INL and Los Alamos National Laboratory (LANL) implemented a much more efficient blocking communications model that resulted in a ~2x speedup for the R/B-16 cases. Unfortunately, analysis of per-iteration performance on LANL's Trinity system indicated that further blocking speedup was unlikely. The algorithm had already been reduced to the minimum number of messages possible and communication costs still drove a large degree of variance between iterations [3].

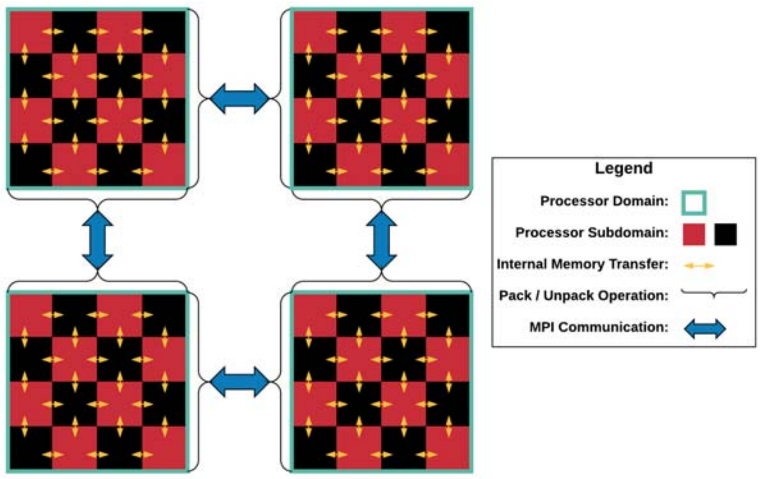

Figure 1: 2D Analogue of PIDOTS Decomposition for RB-4, *p*=4

As further blocking communication speedups were unlikely, it was necessary to explore the development of a non-blocking algorithm. However, several critical constraints existed as a result of PIDOTS's testbed status. Firstly, the new method had to be generalizable to other codes, not just applicable to PIDOTS. Second, the algorithm should interleave with the mathematics of the solver as little as possible because the ITMM solver is not commonly used in transport codes. So, inserting interleaving within ITMM to achieve speedups would limit the applicability of the algorithm. An ideal solution would be transferable to a broader class of spatially decomposed transport solvers. Finally, as the new algorithm is intended to be retrofit into existing codes, it should incur low overheads in both implementation and run time cost.

2. THE NON-BLOCKING ALGORITHM

A new non-blocking algorithm was designed for PIDOTS to meet the requirements outlined above. To achieve significant interleaving, it relies on reordering the spatial solves (each red/black cell shown in Figure 1) rather than interleaving into them. This makes our new algorithm completely transferable to other similarly decomposed codes, regardless of the underlying transport solver. Additionally, since the non-blocking algorithm sits above the solver level, it is straightforward to implement, requiring only a function to reorder solves and minor changes to the communication calls. Finally, the new method is numerically equivalent to its blocking counterpart, allowing for easy testing during an in-place upgrade for existing codes.

2.1 Iterative Flow

The algorithm maintains the general iterative flow of PIDOTS with no major algorithmic changes. This consists of, in order, the five phases illustrated in Figure 2: Red Solve, Red Send, Black Solve, Global Convergence Check, and Black Send. This means that the first interleaving occurs in an iteration and the second across two iterations. However, this is purely a descriptive construct since the same code is executed during each interleave. As a result, it may be easier to consider a slightly shifted iteration order in which red solve is moved to the last step in the iterative flow. In this model, a single red solve is performed without interleave and then the code iterates over pairs of interleaved phases separated by a convergence check. These two logical models are shown as “Traditional” and “Fully Implemented” in Figure 2. Again, these two models are purely for analysis and discussion purposes. They do not change the iterative flow of the original code.

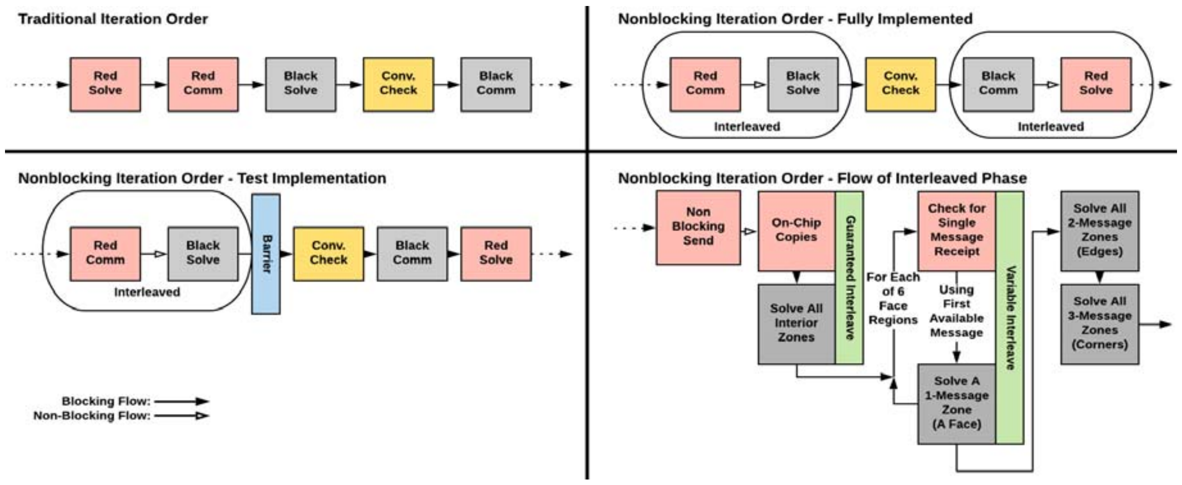
Traditional Iteration Order
Red Solve
Red Comm
Black Solve
Conv. Check
Black Comm
Nonblocking Iteration Order - Fully Implemented
Red Comm
Black Solve
Interleaved
Conv. Check
Black Comm
Red Solve
Interleaved
Nonblocking Iteration Order - Test Implementation
Red Comm
Black Solve
Interleaved
Barrier
Conv. Check
Black Comm
Red Solve
Blocking Flow:
Non-Blocking Flow:
Nonblocking Iteration Order - Flow of Interleaved Phase
Non Blocking Send
On-Chip Copies
Guaranteed Interleave
Solve All Interior Zones
For Each of 6 Face Regions
Check for Single Message Receipt
Using First Available Message
Solve A 1-Message Zone (A Face)
Variable Interleave
Solve All 2-Message Zones (Edges)
Solve All 3-Message Zones (Corners)

Figure 2: PIDOTS's Iterative Flow for Various Models

The third model shown in Figure 2, denoted “Test Implementation”, is the primary one that will be analyzed in the Results section. It provides the fully implemented interleave over the first pair of phases and the original blocking model over the last two. These are separated by a barrier to avoid cross-contamination of timing results from the two communication algorithms. This model, while limited to half the interleaved speedups of the full model, allows for direct comparison of the two algorithms under the same network contention and allocation quality thus eliminating external (user-independent) effects that may change between runs. Finally, the flowchart in the bottom-right quadrant shows the flow of a single interleaved phase. This behavior is detailed in the next section.

2.2 Solve Reordering

To avoid limiting the applicability of the new algorithm, interleaving was achieved through solve reordering rather than by moving receives into the computational section of the solver. For general meshes, Cartesian and unstructured, this requires a pre-processing step. However, it is executed only once, is performed in parallel, and can be stored along with mesh data, if necessary, for later use with runs on the same mesh.

As opposed to a more traditional pattern of sweeping across the processor domain from one corner to the opposite, this method relies on ordering the red/black solve regions by number of required MPI messages and message source. This means that domains requiring only on-chip data are solved first. The code then loops over MPI Test calls until any one message is received. Once received, zones requiring only that message are solved. The code then MPI Tests for a second message and solves the zone requiring only that message. This continues until all one-message zones are solved (i.e. all messages received). Then, the two-message zones and three-message zones are solved. A 2D analogue of this ordering scheme is shown in Figure 3. Note that while the same number of messages are required for each edge, the 4 edges are queued as four distinct zones in the solve order. The order of solving edge domains and corner domains is decided on-the-fly as messages arrive. However, all edge domains will be solved before the first corner domain is solved. This guarantees that no more than 1 additional message is ever required for the next group of work to proceed. Likewise, in 3D Cartesian geometry, the domain-solve order is interior, 6 faces, 12 edges, 8 corners. This is the pattern shown in Figure 2.

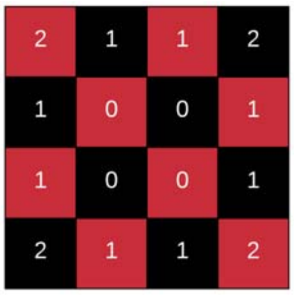

Figure 3: 2D Analogue of MPI Message Count Numbering that Dictates Solution Order

This method provides a variable degree of interleaving based on the solve domain refinement selected. This means that it is most effective in higher refinement configurations. The degree of achieved interleave in terms of percent of solve-phase time is shown in Table 1. The algorithm can hide at least $Interleave\ \% * T_{Solve}$ seconds of communication time per iteration. Further hiding can occur but will vary by execution based on how long the next messages take to arrive. Also note that R/B-2 has no guaranteed interleaving. Instead, it must directly wait for some usable set of 3 messages to arrive. While this is suboptimal, it is considered acceptable as R/B-2 is not a highly useful configuration due to higher resource and time costs when compared to a similar sized problem at a higher R/B-# refinement.

Increased refinement is more useful but also increases the number of on-chip memory transfers required (the zero MPI message region). This work is not interleaved and can be non-negligible if actual copies are used in place of pointers. PIDOTS currently uses the former, but we plan to transition to the latter. Further interleave can be achieved in more permanent implementations by interleaving into that code's specific solve routines in contrast to the current implementation that permits any solver matching the spatial decomposition pattern used by PIDOTS.

Table 1: Achievable Total Iteration Interleave for Various R/B-# Before First Message Required

R/B-#	# of Solve Domains					Interleave
	Total	Interior	Face	Edge	Corner	(%)
2	8	0	0	0	8	0
4	64	8	24	24	8	12.5
8	512	216	216	72	8	42
16	4,096	2,744	1,176	168	8	67
$\boldsymbol{N}$	$\boldsymbol{N^3}$	$\boldsymbol{(N-2)^3}$	$\boldsymbol{6(N-2)^3}$	$\boldsymbol{12(N-2)}$	**8**	$\boldsymbol{\frac{(N-2)^3}{N^3}}$

3. RESULTS

Performance of the interleaved algorithm was evaluated using LANL's Trinity [4] system for processor counts from 512 to 32,768 at R/B-2 through R/B-16. The one-group test problem employed in this process comprised 4,096 cells per processor and S_8 quadrature. We noted contributions to wall-clock time per iteration from the various phases of the algorithm using the five variants of the code described above:

1. **Original** – The optimized blocking algorithm presented in Ref. 3
2. **Color Non-Blocking (CNB)** – A non-blocking algorithm but interleaving only with the on-chip copies, not the solves
3. **Interleaved (INT) –** The "Test Implementation" described in Figure 2

https://doi.org/10.1051/epjconf/202124703016

4. **No Barrier (NBAR) –** Version 3 with the barrier removed
5. **Interleaved v2 (INTv2) –** An optimized (see below) version 3

Each version was intended to isolate and measure a specific component of the algorithm change. CNB measures the impact of switching to non-blocking without significant interleave as well as the in-iteration overhead of the reordered solve phases. INT tests the effectiveness of the interleaved model. Finally, NBAR and INTv2 address questions raised during discussion with LANL HPC staff regarding certain optimal OpenMPI [5] behaviors. As the name suggests, NBAR removes the inter-phase barrier to test machine jitter. INTv2 implements several optimizations to the MPI call logic to match best practices.

As shown in Figure 4 and Figure 5, the CNB and INTv2 variations are the most effective in terms of total iteration time. While CNB is equally effective at *p*=32K, it lacks the further potential interleaving available to INTv2 and, we conjecture, will not demonstrate massive scaling to the same degree. Nor will it be as effective if pointer operations are used in place of on-chip copies. The new algorithm is observed to operate cleanly as the solve time remains constant across the red and black phases, even while only the Red Comms phase is interleaved. Additionally, by comparing the Red/Blk Comms columns, one can see that significant reductions in communication time occur in the interleaved (red) communications phase. Note that the All Reduce time is included in Black Comms. The figures also present it isolated purely for ease of reference. The remaining red communication time is on-chip memory copies that, as expected, increase with increasing R/B-#. Finally, it is interesting to note that, unexpectedly, NBAR is the worst performing of the non-blocking variants. This feature can reasonably be attributed to the increased jitter of the non-blocking routines increasing the total time of the convergence check phase. By placing a barrier before the All Reduce operation in this phase, the total delay between entering and leaving the reduction is minimized. Overall, we observe an ~6 and 10% run time reduction in R/B-16 and R/B-4, respectively. We anticipate that these gains will approximately double to ~12-20% in the "Full Implementation" model interleaving both phases.

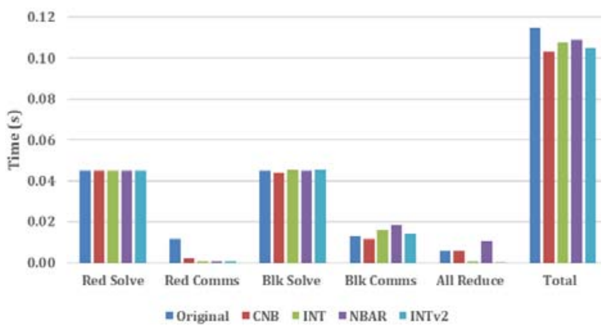

Figure 4: Version Comparison at p = 32K, RB4

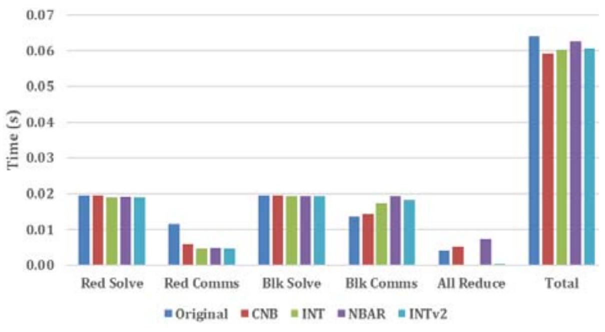

Figure 5: Version Comparison at p = 32K, RB16

Figure 6 shows the performance of the INTv2 algorithm at R/B-16 for processor counts ranging from 512 to 32,768 under a weak scaling regime. As expected, the two solve phases remain constant, while the blocking communication time increases with *p*. The apparent plateau at 32K is likely caused by the network topology resulting in 21K and 32K processors having a nearly identical average communication distance. This plateau should not persist as the processor count increases further. In the red (non-blocking) phase, the communication time is drastically shorter than the corresponding blocking time in the black phase, thus indicating that a significant fraction of MPI communication overhead is effectively hidden via interleaving. The remaining time in the Red Comms column is the on-chip memory copying. Finally, and as expected, in the INTv2 algorithm the All Reduce time is practically negligible for all *p*. These results clearly demonstrate the communication time improvements obtained through interleaving for higher processor counts. They also show that lower processor counts are not adversely affected by the new algorithm.

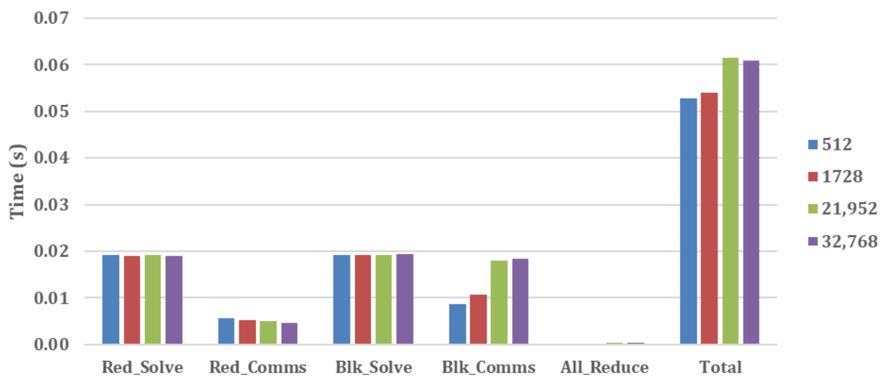

Figure 6: INTv2 Performance for Various Processor Counts at R/B-16

Figure 7 shows the weak scaling growth rate for the Original and "Test Implementation" model (see Figure 2) of INTv2. As can be seen, by interleaving half of each iteration, the growth in iteration time is reduced by a factor of 2. This strongly suggests that the overwhelming majority of iteration-time inefficiency (and, consequently, growth) results from the communication phase, thus confirming findings in Ref. 3. While it is expected that the full interleave model will further reduce these growth rates, it is unlikely that it will fully eliminate them. Rather, a small growth rate will likely remain to account for various overheads including maintaining more MPI members in collective operations such as the Barrier and the All Reduce incurred in each iteration.

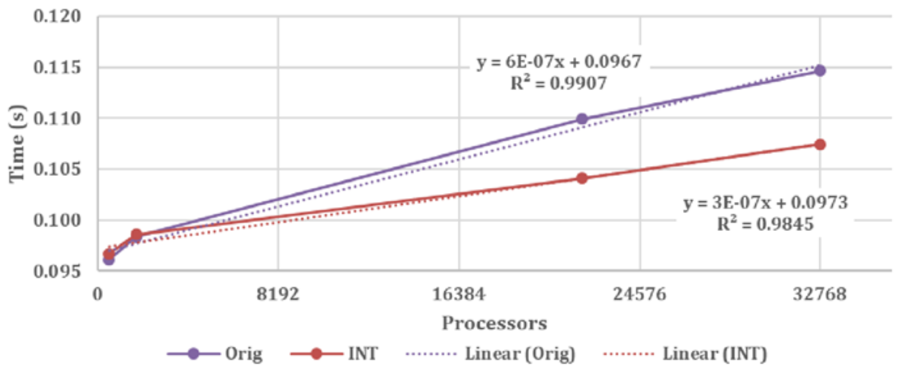

Figure 7: Iteration Time Growth Rate for R/B-4 Weak Scaling (Test Implementation Model)

4. CONCLUSIONS

Prior work provided an optimized blocking algorithm for the PGS transport solver implemented in the PIDOTS code and suggested that further efficiency gains could only be obtained using a non-blocking communication algorithm. This stemmed from the fact that remaining inefficiency was attributed to communication effects beyond the user's direct control such as allocation quality and contention.

This work proposed, implemented, and tested a new non-blocking scheme for PIDOTS which is deeply interleaved, easily implemented, and solver agnostic. These traits make it well-suited for implementation in other codes seeking a high-efficiency, massively scalable communications scheme. Work is currently ongoing to transfer the lessons learned in optimizing the PIDOTS communication structure to an unstructured mesh spatial domain decomposition parallelization in the THOR code [6].

The new communication scheme derives its interleaving work from the reordering of on-processor solve domains. By solving domains in sequence of increasing number of message-receives required, the need for the first message is pushed significantly further down in time into the solve phase. From there, subsequent solves are ordered to require the minimum number of additional messages. This method can be transferred to any arbitrary mesh, including unstructured, as the ordering phase can be pre-computed and stored. Each processor simply needs to sweep its owned solve domains and determine how many neighbors it has within its assigned subdomain. This order is then stored for use as a queue during the solve phase in all iterations.

Testing variants of the interleaved algorithm on LANL's Trinity HPC indicated that the algorithm is highly effective in reducing the growth of communication time with increasing p. It fully suppressed the cost of MPI communication for tests up to 32,768 processors for the interleaved half of the problem. Analysis of the available work indicates that the method should be able to fully suppress the communication costs of far larger processor counts. However, this has not yet been tested due to user resource limitations on the system. For the tested cases, this half-interleaved model demonstrated an ~10% speedup over the optimized blocking code. For a fully interleaved model, this is expected to rise to ~20%. Finally, analysis of the weak scaling growth rate indicated that the introduction of a half-interleaved model reduced the growth rate in iteration time by 50%. This constitutes the maximum available speedup and no higher reduction can be achieved by only improving half of the code. Work is ongoing to implement and test the fully interleaved model and to assess its parallel performance in the massively parallel regime.

ACKNOWLEDGMENTS

This material is based upon work by the first two authors (RY, YYA) supported by the Department of Energy National Nuclear Security Administration under Award Number(s) DE-NA0002576. This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

REFERENCES

1. R. Yessayan, Y. Y. Azmy, S. Schunert, C. Garvey, "Analysis of Communication Performance Degradation of The Radiation Transport Code PIDOTS On High-Utilization, Multi-User HPC Systems", *Proceedings of PHYSOR 2018*, Cancun, Mexico, April 22-26, 2018
2. R. Zerr, Y. Azmy, "Solution of the Within-Group Multidimensional Discrete Ordinates Transport Equations on Massively Parallel Architectures," *Transactions ANS* **105**, 429 (2011)
3. R. Yessayan, Y. Y. Azmy, R. J. Zerr, S. Schunert, "Mitigation of Communication Latency Based Slowdowns in the PIDOTS Massively Parallel S_N Transport Code", *Proc. of M&C* 2019, Portland, OR, Aug 25-29, 2019
4. "Trinity: Technical Specifications," https://www.lanl.gov/projects/trinity/specifications.php
5. E. Gabriel, et al., "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation," Proceedings of 11th European PVM/MPI User's Group Meeting, Budapest, Hungary, Sept. 2004
6. R. M. Ferrer, Yousry Azmy, "A Robust Arbitrarily High-Order Transport Method of the Characteristic Type for Unstructured Grids", *Nuclear Science and Engineering*, Vol. 172, Number 1, pp. 33-51 (2009)