COMMUNICATION STRATEGIES FOR ANGULAR DOMAIN DECOMPOSITION OF TRANSPORT CALCULATIONS ON MESSAGE PASSING MULTIPROCESSORS

Y. Y. Azmy

Oak Ridge National Laboratory*
P.O. Box 2008, MS-6363
Oak Ridge, Tennessee 37831-6363
Phone: (423) 574-8069  Fax: (423) 574-9619
yya@ornl.gov

"The submitted manuscript has been authored by a contractor of the U.S. Government under contract DE-AC05-84OR21400. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes."


*Research sponsored by the Oak Ridge National Laboratory managed by Lockheed Martin Energy Research Corporation for the U.S. Department of Energy under contract No. DE-AC05-96OR22464.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
COMMUNICATION STRATEGIES FOR ANGULAR DOMAIN DECOMPOSITION OF TRANSPORT CALCULATIONS ON MESSAGE PASSING MULTIPROCESSORS

Y. Y. Azmy

Oak Ridge National Laboratory
P.O. Box 2008, MS 6363
Oak Ridge, Tennessee, 37831-6363
(423) 574-8069, yya@ornl.gov

ABSTRACT

The effect of three communication schemes for solving Arbitrarily High Order Transport (AHOT) methods of the Nodal type on parallel performance is examined via direct measurements and performance models. The target architecture in this study is Oak Ridge National Laboratory's 128 node Paragon XP/S computer and the parallelization is based on the Parallel Virtual Machine (PVM) library. However the conclusions reached can be easily generalized to a large class of message passing platforms and communication software. The three schemes considered here are: (i) PVM's global operations (broadcast and reduce) which utilizes the Paragon's native corresponding operations based on a spanning tree routing; (ii) the Bucket algorithm wherein the angular domain decomposition of the mesh sweep is complemented with a spatial domain decomposition of the accumulation process of the scalar flux from the angular flux and the convergence test; (iii) a distributed memory version of the Bucket algorithm that pushes the spatial domain decomposition one step farther by actually distributing the fixed source and flux iterates over the memories of the participating processes. Our conclusion is that the Bucket algorithm is the most efficient of the three if all participating processes have sufficient memories to hold the entire problem arrays. Otherwise, the third scheme becomes necessary at an additional cost to speedup and parallel efficiency that is quantifiable via the parallel performance model.

I. INTRODUCTION

The working assumption in parallel computing in recent years has been that advances in communication speed and capacity will eliminate the need for sophisticated hardware connection topologies and communication schemes for highly demanding large applications. Indeed the hypercube topology aimed at compromising the largest distance between nodes and the complexity of constructing and maintaining a large number of inter-node connections has given way to simpler topologies such as two dimensional grids. The main motivation for this approach is to relieve the programmer from the generally tedious task of customizing the communication strategy for a given application by providing standard library routines that can perform these functions efficiently. To what extent has this promise been met is almost certainly application and platform dependent, and in some cases may even be sensitive to problem specifications, such as problem size.

The purpose of this paper is to examine this assumption for neutron transport calculations parallelized via an angular domain decomposition on distributed memory, message passing multiprocessors. Specifically, we implement the Arbitrarily High Order Transport (AHOT) code with the Nodal option (AHOT-N) on the Paragon XP/S 5 computer using the Parallel Virtual Machine (PVM) communication library. Angular domain decomposition for AHOT-N has been investigated earlier on a variety of multiprocessor architectures. In general it has been shown that parallelization along angular directions in Cartesian geometry possesses the following desirable properties:

1. Provides an intrinsic decomposition: hence intermediate results and the final solution in the parallel and sequential codes are identical to within arithmetic precision; also the number of iterations required to achieve convergence is independent of the number of participating processors.
2. Coarse grained parallelization: each concurrent process comprises a sweep of the entire mesh along one discrete ordinate in the positive octant of angular space, hence the low communication penalty and high parallel efficiency. [More concurrency can be achieved if reflective boundary conditions are excluded].
3. Perfect load balance: even on irregular grids, since each concurrent process covers all the cells, not necessarily in the same order, the total computational load per process is the same for all participating processors if their number
Based on these facts various implementations of angular domain decomposition have been demonstrated to produce large speedup at high parallel efficiency on shared and distributed parallel computers in the coarse to intermediate grain size range. It has also been observed that the most serious limitations of angular domain decomposition is the typically small number of discrete ordinates, at most a few hundreds for the largest applications, which sets a strict limit on the achievable speedup. This, however, is not a severe limitation in view of the fact that other domain decompositions that result in more abundant concurrency usually incur large parallelization penalties that limit them to the same general range of speedup factors albeit with a lot more processors. Also most angular domain decompositions reported so far do not divide the memory requirement among the participating processors thus curtailing the potential for scalability to very large problem sizes. In this paper we present a new scheme for distributing the largest arrays, namely the fixed source, and new and old iterates of the scalar flux spatial moments among the participating processors.

The remainder of this paper is organized as follows. In Sec. II we describe the standard angular domain decomposition based on PVM's global reduce and broadcast operations and we construct a simple parallel performance model for the Paragon. In Sec. III we briefly describe the Bucket algorithm, demonstrate its lower communication cost, then construct a parallel performance model for the Paragon. We present the new distributed memory version of the Bucket algorithm in Sec. IV and put its performance in perspective by constructing a model for its performance on the Paragon and comparing its communication penalty to that of the standard Bucket algorithm. Finally we summarize our findings in Sec. V.

II. THE STANDARD GLOBAL REDUCE

The most straightforward approach to implementing angular domain decomposition is to use global reduce operations built-in PVM, and most alternative communication libraries. The computation for a single Source Iteration (SI) proceeds like this. Each participating processor has the old iterate of the scalar flux and fixed source in its local memory, so each computes the scattering source over the entire mesh. Since the computational load comprising a mesh sweep along a single discrete ordinate is the same for all angles in the selected quadrature, these are statically scheduled to the participating processors without adversely affecting load balance (unless the number of angles is not divisible by the number of processors). Each participating processor then proceeds to sweep the mesh along the directions it owns accumulating the spatial moments of the angular flux in their private memories during the sweep. Upon concluding the mesh sweep all participating processors cooperate in a global reduce operation in which the private vector containing the weighted sum of the spatial moments of the new iterate of the scalar flux along owned directions is summed across processors. The algorithm with which PVM performs the global reduce operation depends on the target platform(s). On the Paragon it utilizes the native global sum which is based on earlier, i.e. hypercube, spanning tree routing of messages discussed shortly. The result of the global sum is to leave the summed vector, the new iterate of the spatial moments of the scalar flux, on a single, i.e. root, node which then proceeds to broadcast that vector to all participating processors via a group broadcast, also using a spanning tree route for messages. At this point all participating processors test pointwise convergence over the entire mesh thereby completing a full iteration.

As noted earlier, spanning tree algorithms are geared towards minimizing the number of messages exchanged between participating processors without regard to the volume of data traffic and processor idleness. This was motivated by the high communication latencies that characterized early message-passing multiprocessors, and is naturally suited for the hypercube connection topology aimed at limiting the distance between nodes while keeping the number of hardware connections reasonable. In a $d$-dimensional cube each node is connected directly to $d$ neighbors and the largest number of hops between two nodes is $d$. The spanning tree routing of messages is defined with respect to a root which collects the final result in a reduce operation, or initiates the sending sequence in a broadcast. In every stage of a global reduce, for example, half the nodes that possess partially reduced results send them to nearest neighbors which perform the reduce operation between the just received vector and their own vector, then proceed to the next stage until the root is reached. Analogously in a broadcast in every stage each node in possession of the broadcast vector, starting with the root in the first stage, sends it to a nearest neighbor, thereby doubling the number of nodes the message has reached in every stage.

The two main drawbacks of the spanning tree routing of messages are:

1. Full utilization of all participating processors requires their number to be a power of two; this is a severe constraint especially if the application seldom provides concurrency in powers of two, as indeed is the case with angular domain decomposition of discrete ordinates methods.
2. In every stage the number of idle processors is doubled or halved, to the extent that when only the root is engaged say in a reduce operation all other nodes are wasted, causing a substantial drag on parallel efficiency.

In recognition of the great reduction in communication latency, advanced algorithms for performing global operations that do not suffer these drawbacks were investigated, particularly in view of the shift in hardware topology from hypercube to simpler connection schemes. Among those is the Bucket algorithm discussed in the following section for the present application. An alternative philosophy with an optimistic outlook to future advances in communication technology presumes communication penalty will eventually become negligible to justify investing effort in attempting to reduce it. Perhaps this is the reason vendors, Intel in the case of the Paragon for example, still adopt the spanning tree routing for global operation. Clearly no matter how negligible the cost of communication is the effect of processor idleness on parallel efficiency will not diminish.

Based on the above description of the spanning tree algorithm we can construct a parallel performance model in terms of the various problem parameters. Consider the case of a two dimensional grid of \( I \times J \) cells, and an order \( \Lambda \geq 0 \) AHOT-N calculation utilizing an \( S_n \) angular quadrature, i.e. \( n(n+2)/8 \) angles per octant in angular space. The execution time per iteration can be divided into three broad categories:

1. Serial time \( (T^S) \) is independent of the number of processors, and includes the problem setup time, input reading and output printing, etc. A good representation of this component of execution time is,

\[
T^S = \tau^S_0 + I J (\Lambda + 1)^2 \tau^S_1 ,
\]

where \( \tau^S_i \), \( i = 0, 1 \) are model parameters that can be estimated from measured time components.

2. Parallel time \( (T^P) \) is the time consumed in the mesh sweep on \( P \) participating processors and is represented by,

\[
T^P = I J (\Lambda + 1)^2 \left[ \tau^P_0 + (\Lambda + 1)^2 \tau^P_1 \right] \left[ \frac{n(n+2)}{8P} \right] ,
\]

where \([\cdot]\) is the ceiling function, and \( \tau^P_i \), \( i = 0, 1 \) are model parameters representing the time required to set up, and solve the \( \Lambda + 1 \) coupled equations per cell, respectively.

3. Communication and global reduce time \( (T^C) \) is modeled according to,

\[
T^C = \left\lceil \log_2 P \right\rceil \left[ 2 \tau^C_0 + I J (\Lambda + 1) (2 \tau^C_1 + \tau^p) \right] ,
\]

where \( \tau^C_0 \) and \( \tau^C_1 \) are the communication latency and volumetric communication rate, respectively, and \( \tau^p \) is the time per reduce operation, addition in the present application.

The total time per iteration is the sum of equations (1-3). These three components were measured on Oak Ridge National Laboratory’s XPS 5 Paragon with 128 compute nodes for a simple homogeneous problem on a 36 \( \times \) 36 mesh and an \( S_{16} \) angular quadrature using the Order AHOT. The results of these measurements were fitted to the model constructed above and verified in Fig. 1 for each of the three components, as well as the total execution time. Agreement between the model and measured times is excellent for the parallel component and the total execution time, which is dominated by the former. The serial component is in good agreement given its small magnitude and the fact that it is comprised of many code sections making it highly susceptible to imprecision. The global reduce component exhibits generally correct behavior relative to the model, but is less accurate than the other two components. This is probably due to the fact that the actual connection topology of the nodes in the XPS 5 Paragon is not a hypercube so the \( \left\lceil \log_2 P \right\rceil \) behavior is only approximate. It is evident from the total execution plot in Fig. 1 that on such small problems high parallel efficiency and speedup factors can be achieved via the standard communication schemes.

III. THE BUCKET ALGORITHM

The primary motivation for the Bucket algorithm is to reduce processor idleness by distributing the arithmetic operations in the global reduce over the participating processors and exchanging a larger number of shorter messages in the broadcast stage. In the Bucket algorithm the participating processors are connected via a ring topology so that each participating processor communicates only with an adjacent neighbor on either side,

\[
p_+ = \text{mod}_P (p + 1) , \quad p_- = \text{mod}_P (P + p - 1) .
\]

Here also an iteration starts with all processors having local access to the previous iterate of the scalar flux and fixed source, which are used to compute the scattering source for the present iteration. The discrete ordinates in the employed quadrature are distributed among the participating processors in a static scheduling mode, and the mesh sweep proceeds exactly as in the case of the spanning tree case. The global reduce operation is comprised of two stages: the combine
stage, and the convergence test and broadcast stage.

In the combine stage the distributed vector $V$ which on each processor $p$ contains the contribution to the scalar flux spatial moments from the directions assigned to $p$ is summed across processors in $P-1$ steps. This is accomplished by first dividing $V$ into $P$ nonintersecting subvectors $V_{p,q}$, $q = 1, \cdots, P$, each of length $\lceil \text{dim}(V)/P \rceil$ except $V_{p,p}$ is of length $\text{dim}(V)/P - (P-1) \lceil \text{dim}(V)/P \rceil$. Then in each step $v = 1, \cdots, P-1$ each processor $p$ sends the vector $V_{p,q,v}$ to processor $p+$, receives the subvector $V_{p,q,v}$ from processor $p-$ into a temporary buffer then combines it, i.e. sum, with its local subvector $V_{p,q,v+1}$, where $q_{v+1} = \text{mod}_P (P + q_v - 1)$, $v > 1$, with $q_1 = p$. Note that $q_v$ depends implicitly on $p$ so that the initially nonoverlapping subvectors $V_{p,q}$ remain so throughout the combine stage. At the conclusion of the combine stage each processor $p$ will have in its local memory the globally summed subvector $V_{p,-1}$.

The convergence test and broadcast stage also is performed in $P-1$ steps. Prior to the first step each processor computes the maximum magnitude relative difference between the old and new iterates of the scalar flux spatial moments for the subvector $V_{p,-1}$. Then in each step $v = 1, \cdots, P-1$ of this stage each processor $p$ sends $V_{p,q,v}$ amended with the convergence error for that subvector computed earlier to processor $p+$, and receives $V_{p,-q,v}$ and its convergence error from $p-$. At the end of this stage each processor will have the globally summed vector $V$ in its local memory and the global supremum of the convergence error, based on which all processors simultaneously start a new iteration or terminate execution.

The serial and parallel components of the execution time for the Bucket algorithm have essentially the same form as Eqs. (1) and (2), respectively, except the $T$ superscript converts to a $B$ superscript to indicate the difference in the specifics of each algorithm. For example the convergence test included in the serial component earlier is performed in parallel in the Bucket algorithm so that $\tau_{1,1}^B \neq \tau_{1,1}^S$ and $\tau_{p,0}^B \neq \tau_{p,0}^S$. Based on the above description of the Bucket algorithm each of the $P-1$ steps of the two stages is comprised of sending two messages of length $\lceil \text{dim}(V)/P \rceil$ and summing an equal number of elements of the global vector. Using the same communication parameters as in Eq. (3), since these are a function of the target platform only, the model for the communication time is given by:

$$T_c^B = (P - 1) \left[ 2 \tau_{G,0} + \left[ \frac{1}{P} \left( \frac{IJ}{(A+1)^2} \right) (2 \tau_{G,1} + \tau_v) \right] \right].$$  

(5)
The three components of the Bucket algorithm execution time were measured on the XP/S 5 Paragon for the same test problem described in Sec. II, and the results were fitted to the model constructed above and is verified in Fig. 2. Here also the parallel component and total execution time exhibit excellent agreement between the model and measured values, while the serial component is only in good agreement for the same reasons noted in Sec. II. In contrast to the spanning tree scheme, however, the global reduce component is also in very good agreement perhaps because the ring topology is easier and more natural to approximate on a two-dimensional grid than a hypercube. Even though for this small problem the speedup and efficiency are practically the same as for the standard reduce, the global reduce component here is about 15% smaller.

IV. DISTRIBUTED MEMORY ALGORITHM

The most obvious drawback of the Bucket algorithm as described above is that it replicates the entire vector containing the spatial moments of the new iterate of the scalar flux on all participating processors. The size of this vector grows as $IJ(A+1)^2$ potentially limiting the scalability of the Bucket algorithm to very fine grids and high orders. Hence we introduce a modified version of the Bucket algorithm which results in distributing the fixed source, and new and old iterates of the scalar flux over the private memories of the individual processors. Obviously this results in additional communication that is not incurred in the Bucket algorithm, therefore the new algorithm should be viewed as a complement to the Bucket algorithm to be used only when the available memory constrains problem size.

The distributed memory algorithm also utilizes a ring connection topology among the participating processors to route messages, except here messages are passed in both directions of the ring simultaneously in order to overlap communication and cut its time penalty in half. At the beginning of an iteration each processor owns $\lfloor J/P \rfloor$ rows of the grid, keeping the fixed source, and new and old iterates of the scalar flux spatial moments in its private memory. The ownership scheme is static and is designed to maximize the overlap of communication activities at the end of processing a block of rows and at the beginning of processing the following block. More specifically, the odd-numbered blocks of rows are assigned to $p = 0, 1, \ldots, \lfloor P/2 \rfloor - 1$, and even-numbered blocks of rows are assigned to processors $p = \lfloor P/2 \rfloor, \ldots, P$. All participating processors loop over the $P$ blocks of rows. For each block the owner broadcasts the fixed source and the old iterate of the scalar flux spatial moments by sending them to its two adjacent neighbors who forward the messages along the same direction until all participants have local access to this data. By using nonblocking
communication nodes are able to proceed with the mesh sweep along the directions assigned to them statically as in the previous algorithms as soon as the messages are sent. When all processors have finished sweeping all directions assigned to them for the present block of rows the global reduce stage of the algorithm commences. In this stage all nodes participate in a global sum identical to the combine stage described above for the Bucket algorithm thereby producing the subvectors of the final result for the present block distributed over all participating processors. In order for the owner of the block to collect these subvectors the two nodes opposite it on the ring send their subvectors towards the owner. Each intermediate node receives a cumulative subvector from its neighbor, amends its own subvector to it, then forwards the resulting subvector towards the owner in the same direction. As before, since communication is nonblocking, the owner of the next block in the sequence, which must be one of the two nodes opposite the owner of the present block, starts broadcasting the fixed source and old iterate of the scalar flux spatial moments. The purpose of the block assignment scheme is now evident, namely to overlap the communications comprising the collection and broadcast steps thereby reducing the communication penalty.

The serial and parallel components are modeled exactly as in the standard global reduce case of Sec. II except here the model parameters are distinguished by a $D$ superscript to reflect the difference in the operations they comprise from the earlier case. The global reduce is performed in $P$ steps, one block of rows at a time. For each block the summation stage is identical to that modeled in Sec. III except here the vectors are only $1/P$ as long so it consumes,

$$(P - 1) \left[ \tau_{G,0} + \left[ \frac{I J (A + 1)^2}{P^2} \right] (\tau_{G,1} + \tau_o) \right].$$

The collection and overlapping broadcast stage is dominated by the latter since it involves a longer message resulting in,

$$(P - 1) \left[ \tau_{G,0} + \left[ \frac{2 I J (A + 1)^2}{P} \right] \tau_{G,1} \right].$$

It follows that the total global operation time for the distributed memory algorithm is modeled by,

$$T^D = 2 (P - 1) \left[ 2 \tau_{G,0} P + 2 I J (A + 1)^2 \tau_{G,1} + \left[ \frac{I J (A + 1)^2}{P} \right] (\tau_{G,1} + \tau_o) \right]. \quad (6)$$

The factor of 2 appearing on the right hand side of Eq. (6) accounts for the $\eta < 0$ and $\eta > 0$ sweeps which must be conducted separately, thus requiring separate rounds of communication.

The three components of the distributed memory algorithm execution time were measured on the XP/S 5 Paragon for the same test problem, and the results were fitted to the model constructed above and is verified in Fig. 3. The parallel and global reduce components, and the total execution time for this case exhibit excellent agreement between the model and measured values while the serial component is only in good agreement for the same reasons discussed in Sec. II. The communication penalty here, nevertheless, grows much faster than the previous two algorithms considered in this work. This is a consequence of the $O(P^2)$ term in Eq. (6) which is significant in the present test problem where the distributed vector length is relatively short. It must be noted from Eq. (6) though that even for high order, fine mesh problems the communication cost for this algorithm will grow as $O(P)$, much faster than either of the other two schemes. It is evident from the total execution time plot that in spite of the rapid increase in communication penalty the distributed memory algorithm is still capable of delivering respectable speedup at reasonable efficiency at a lower demand on individual processor memory than the previous schemes.

V. DISCUSSION AND CONCLUSION

We presented three schemes for conducting the global reduce operations necessary for parallelizing transport calculations via angular domain decomposition. These are PVM's built in global reduce operation which on the Paragon utilizes a spanning tree route for message exchanges; the Bucket algorithm; and a new distributed memory modification of the Bucket algorithm. These three schemes were implemented in the AHOT code on the XP/S 5 Paragon computer and used to solve a simple test problem, where we measured the three main components of execution time, namely the serial, parallel, and global reduce components. For each of these schemes we constructed a performance model for these three components as a function of the number of participating processors and verified the model against measured values. We conclude that for small problems such as the one used in this work all three schemes provide reasonable speedup, 10, 11, and 6, at high efficiency, 85%, 92%, and 50%, respectively. Focusing on the communication component we observe that the Bucket algorithm is the least penalizing followed by the standard reduce, while the distributed memory algorithm is far more penalizing. Hence we conclude that for problems that fit entirely in each processor's memory the Bucket algorithm is the best choice, otherwise only the distributed memory can relieve the demand on memory at a cost to
Fig. 3. Measured and model components of execution time for the distributed memory algorithm.

In order to explore the performance of these three algorithms for very large applications we plot in Fig. 4 the model of the communication penalty for each against $P$ for an $S_{16}$ angular quadrature but with $IJ(A+1)^2 = 10^5$. These results clearly demonstrate the vast advantage of the Bucket algorithm over the Paragon’s native global reduce operation, but it also casts doubt over the utility of the distributed memory algorithm for the large problems it is designed to handle. It is worth observing that the rapid increase in this curve is mainly due to the volume of traffic which, we conjecture, will characterize any spatial distribution of the large flux and source arrays over the memory of the participating processors.

Another way to view these results, is to compare the communication penalty of the spanning tree and Bucket algorithms to the total execution time which immediately justifies the tendency to ignore communication cost. We note however that squeezing the last bit of performance still requires fancy communication schemes some perhaps not developed yet.

REFERENCES


Fig. 4. Model of the communication penalty component of execution time for the three schemes for a problem with $10^6$ scalar flux spatial moments.