CONF-960415--25

Multitasking the Three-Dimensional Transport Code TORT on CRAY Platforms*

Y. Y. Azmy
Computational Physics and Engineering Division
Oak Ridge National Laboratory
P.O. Box 2008, Bldg. 6025
Oak Ridge, Tennessee 37831-6363

D. A. Barnett and C. A. Burre
Lockheed Martin Corporation
P.O. Box 1072
Schenectady, New York 12301

Paper submitted for oral presentation and publication in the *Proceedings of the 1996 Radiation Protection and Shielding Topical Meeting of the American Nuclear Society,* N. Falmouth, MA, April 21-25, 1996.

---

*Managed by Lockheed Martin Energy Research Corporation for the U.S. Department of Energy, under contract DE-AC05-96OR22464.

MULTITASKING THE THREE-DIMENSIONAL TRANSPORT CODE TORT ON CRAY PLATFORMS

Y. Y. Azmy
Oak Ridge National Laboratory
P.O. Box 2008, MS 6363
Oak Ridge, TN 37831

D. A. Barnett, and C. A. Burre
Lockheed Martin Corporation
P. O. Box 1072
Schenectady, New York 12301

ABSTRACT

The multitasking options in the three-dimensional neutral particle transport code TORT originally implemented for Cray's CTSS operating system are revived and extended to run on Cray Y/MP and C90 computers using the UNICOS operating system. These include two coarse-grained domain decompositions; across octants, and across directions within an octant, termed Octant Parallel (OP), and Direction Parallel (DP), respectively. Parallel performance of the DP is significantly enhanced by increasing the task grain size and reducing load imbalance via dynamic scheduling of the discrete angles among the participating tasks. Substantial Wall Clock speedup factors, approaching 4.5 using 8 tasks, have been measured in a time-sharing environment, and generally depend on the test problem specifications, number of tasks, and machine loading during execution.

I. INTRODUCTION

Accumulated experience with a variety of multiprocessing techniques and architectures for the neutron transport equation have demonstrated the benefit to parallel efficiency of employing the coarsest possible grain size. The downside of this is the resulting small number of concurrent processes which bounds the achievable speedup from above. The coarse grained architectural features of Cray multiprocessors, i.e., tens of supercomputer class CPUs, therefore provide an excellent opportunity to best utilize the available resources and at the same time produce significant speedup. The three-dimensional neutron transport code TORT[1] typically consumes several hours, sometimes days, to execute large application problems making it a good testbed for multitasking. In addition, the Cartesian geometry option is the one most commonly used in TORT applications due to the somewhat flexible grids it permits. As demonstrated earlier[2] angular domain decomposition is intrinsic in Cartesian geometry, meaning no degradation of the convergence rate with increasing number of participating processors, thereby enabling the highest possible parallel speedup and efficiency on coarse grained platforms. This follows from the fact that the processors on such platforms are so fast that a large computational load per concurrent process is crucial for good parallel performance, and because the typically small number of CPUs does not allow more concurrent processes than the number of discrete ordinates in the majority of applications.

Previous work on multitasking TORT's Linear Nodal method on a two-processor Cray X/MP running under the now obsolete CTSS operating system achieved limited speedup, if any, even for 100,000 cell problems.[3] In this work we revisit the implementation on the Cray Y/MP and C90 computers using two variants of angular domain decomposition, one along octants, the Octant Parallel (OP) scheme, the other along individual angles, the Direction Parallel (DP) scheme, and demonstrate

their high parallel efficiency for various test problems. In the remainder of this paper we briefly review the OP, and DP schemes in Secs. II, and III, respectively. Then we present and discuss measured performance for three test problems on 8-CPU Cray Y/MP and 16-CPU Cray C90 computers in Sec. IV, and we close with a brief summary and some conclusions in Sec. V.

II. THE OCTANT PARALLEL METHOD

The mesh sweeps are performed in TORT one plane at a time; for each plane every row in the x-direction is swept to the left then to the right in all angular directions within an octant in angular space. Hence, the OP is implemented by starting a slave task to perform the left sweep for all angles with $\mu<0$ while the master task completes the right sweep for all angles with $\mu>0$ then awaits the slave task before proceeding to the next row. Mutually exclusive locks are implemented within the parallel section of the code to avoid memory conflicts in the process of accumulating the angular flux contributions to the new iterate of the scalar flux and its higher angular and spatial moments. Great care must be exercised in setting up these locks not just to ensure correct operation of the code but also to avoid destroying the vectorizability of loops over a row for a given angle. After sweeping a row along discrete ordinates within two octants, TORT proceeds to sweep the next row over the same two octants. After both octants are solved in all rows of a plane, TORT solves the other two octants which have the same *eta*–level sign. Upon completing a plane, TORT steps to the next plane; a complete inner iteration consists of a downward and upward sweep through the computational mesh.

The OP strategy suffers two drawbacks. First, only two tasks, corresponding to the right and left sweep directions, can execute concurrently, thus bounding the potential speedup from above by 2. However, the intrinsic nature of the domain decomposition implies that there is a one-to-one correspondence between the sequential and concurrent algorithms. Thus, there is no deterioration in the spectral properties of the iteration process. This, along with the coarse granularity of the parallel algorithm, provides for as high a parallel efficiency as can be expected. Second, only fixed source (including vacuum) and periodic boundary conditions are permissible along the spatial dimension in which the sweep concurrency is realized, the x-dimension in TORT.

TORT uses zero weight angular directions to distinguish the various η levels which is important for proper computation of the redistribution term in curvilinear geometry. The μ values attached to the zero weight directions are always negative implying that all will be swept in the left sweep performed by the slave task in a multitasked run; this introduces some load imbalance. Of course, the zero weight directions are conceptually worthless in Cartesian geometry, and as such, it seemed possible to skip sweeping them in the loop over angular directions in subroutine *rowmp*. Hence, we examined the 5000 loop in *rowmp* and discovered, as well as numerically confirmed, that skipping the zero weight directions entirely results in erroneous fluxes because of the several indices that are updated in this loop.

III. THE DIRECTION PARALLEL METHOD

The Direction Parallel (DP) method is based on sweeping each row of cells in the mesh concurrently along the discrete ordinates in an octant of angular space. [This method was termed Row Parallel in Ref. 3]. As such it eliminates the two drawbacks of OP mentioned above thus providing a large number of independent tasks to be executed simultaneously and, since octants are processed successively, allowing all types of boundary conditions. The DP is an Angular Domain Decomposition (ADD) which in rectangular geometry, the primary focus of this work, is intrinsic. Its main drawback is the typically limited number of tasks available, essentially the number of discrete

ordinates per octant, compared to the very large number of computational cells in a spatial domain decomposition for example. Of course the number of concurrent tasks sets an upper bound on the potential for speedup, except in case of distributed memory architectures where the aggregate memory of the participating processors can reduce or eliminate the need for I/O. Typically for coarse-grained platforms supporting at most a few tens of supercomputer class CPUs such as the Cray Y/MP the real bound on speedup is set by the number of processors.

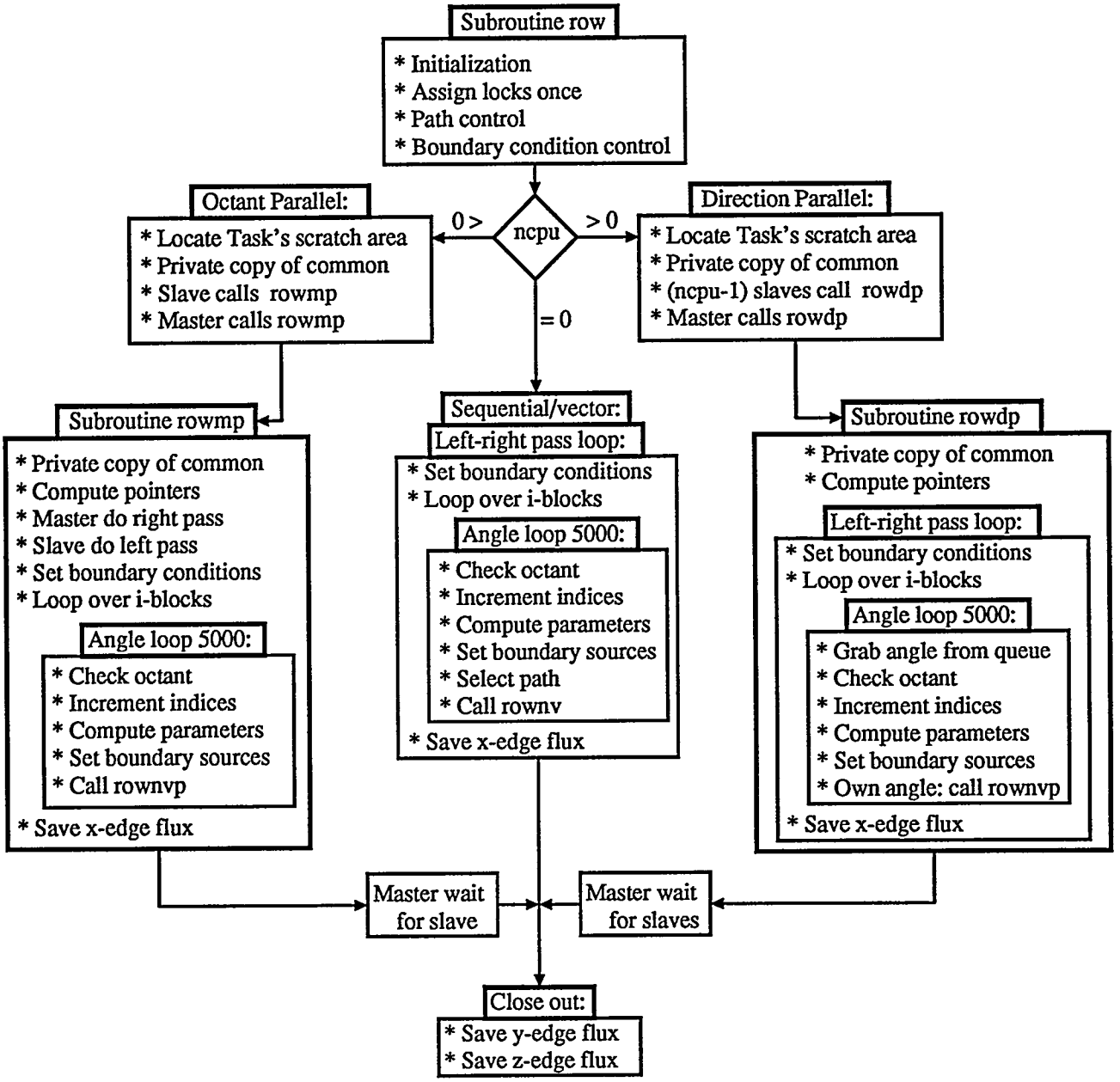
The original implementation of the DP in TORT was hardwired to utilize only two processors, perhaps because of the poor performance which was observed in early tests even on this few processors.[3] Indeed our effort to make this option run correctly, i.e., produce results identical to the sequential run to within roundoff error, progressed very rapidly but initial tests on two processors exhibited very poor performance. The Wall Clock time on two processors was larger than the sequential Wall Clock time, while the CPU time increased by as much as 100%. Upon examining the profiles for sample runs it became evident that while there are several sources of additional CPU time in the DP over the sequential run, much of it is spent by processors on hold waiting for tasks to be executed.

The major enhancements in the new DP implementation address the following issues:

1. Eliminating the overhead for assigning and releasing locks: In the original implementation locks were unnecessarily assigned and released every time subroutine *row* is called. This is avoided by assigning them only once, upon first entry into *row*.
2. Permitting the user to select parallelization scheme and number of concurrent tasks: Several changes were made to permit the user to select the number of concurrent tasks by setting the input parameter *ncpu*. If it is set to zero the original sequential (vector) path is executed in which the row sweeps are conducted by subroutine *rownv*. The OP is selected by setting *ncpu* to -1, or -2 to run with only a master, or one master and one slave task, respectively; if it is set to a value less than -2, TORT resets it to -2 since this is the most concurrency OP allows. In this case the row sweeps are conducted by subroutine *rownvp* which is called by the interface subroutine *rowmp*. Positive values of *ncpu* imply DP running on a master and $(ncpu-1)$ slave tasks also using *rownvp* to sweep a row along one angle, but here called by the interface subroutine *rowdp* described shortly. Generalizing the number of concurrent tasks required, among other things, creating sufficient scratch space for arrays used by *rownvp* and for the private copies of the common block *comrow*. The latter must be fixed at installation; we set the number of these copies to 16 in the installation script of the current version of TORT.
3. Increasing the useful computations per task creation: In the original implementation of DP each call to *rownvp* was performed by a new task implying a large number of task creations which contributes to overhead. By moving the task creation outside of the loop over angles, only as many tasks as requested by the user will be created to sweep each row. Of course this overhead can be reduced even further by moving the task creation to a higher level routine, e.g. *plane,* but this requires additional memory and major code restructuring. DP was implemented in a new subroutine *rowdp* that serves as an interface between *row* and *rownvp*. An additional improvement in the new implementation is the dynamic scheduling of discrete directions within existing tasks thereby reducing the undesirable effect of load imbalance.

The major features of the two parallel options in TORT are depicted in Fig. 1. Our new implementation of DP tests the value of *ncpu* early in subroutine *row,* similar to the OP case, and if positive makes private copies of the common block then calls the new subroutine *rowdp* once by the master task, and $(ncpu-1)$ times by the slave tasks via a call to *tskstart*. Once in *rowdp* each task unpacks its own copy of the common block and computes pointers to the scratch arrays. Execution

Figure 1. Flow Chart for Subroutine *row* with the Two Multitasking Options: OP and DP.



then proceeds as it would in the sequential/vector case, i.e. *ncpu* = 0, to the loop over the left-right passes then to loop 5000 over the discrete ordinates in the current octant. Each task spans loop 5000 for all values of its index *m* but sweeps only along the directions dynamically scheduled to it. Thus each task grabs a direction from the queue of all directions via the global counter *iselfsch* and calls the row sweep subroutine *rownvp* only if *m* is equal to the angle index it just grabbed. This keeps all indices incremented properly within each task but implies redundant operations performed repeatedly in loop 5000. Upon return from *rowdp* the master task waits for all slave tasks then proceeds as in the sequential case with close out activities.

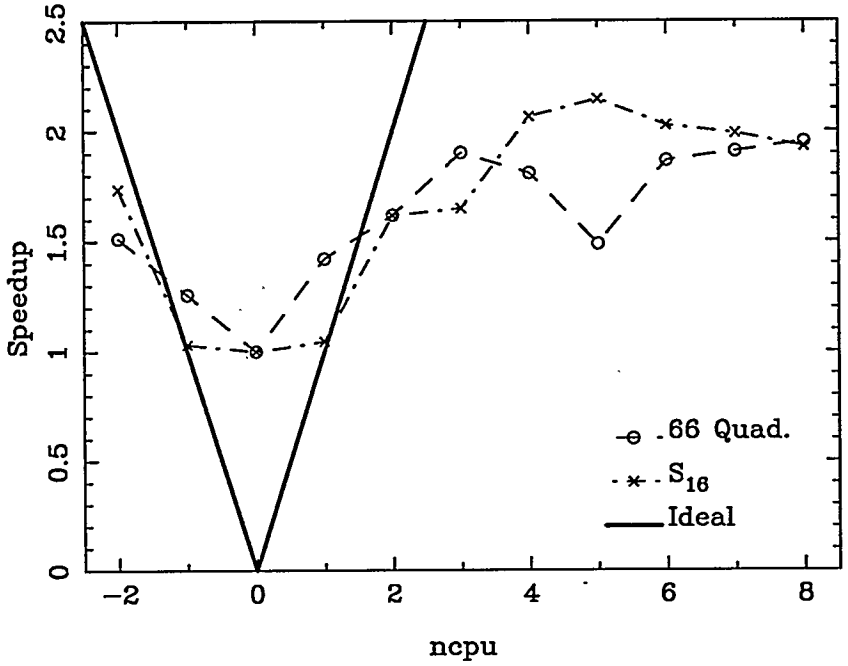
IV. MEASURED PERFORMANCE

We used three problem configurations to test the correctness of the OP and DP methods implementation in TORT and to characterize their performance. All test runs are conducted in the time sharing environment using the target platforms for this implementation, namely multitasking-capable Cray computers. The first two test problems were executed on Los Alamos National Laboratory's rho machine, an 8-CPU Y/MP running UNICOS 8.0, and the third test problem measurements were obtained on Lockheed Martin Corporation's 16-CPU C90, also running UNICOS 8.0.

A. Test Problem 5 (TP5)

The first test problem is a modified version of TORT's Test Problem 5 (TP5) wherein we use the Linear Nodal method with a 10^{-4} convergence criterion, and allow for 20 inner iterations. This problem, which we still denote TP5 in spite of the modifications mentioned above, is sufficiently small (1 group, 66 directions, and 3672 cells) that it fits entirely in core. We used the 66-angle quadrature of the original TP5, then to test the scaling of parallel performance with the number of discrete directions, we repeated the runs with a standard S_{16} angular quadrature. The higher quadrature order benefits parallel performance in two ways. First, it provides a larger pool of independent processes which for DP improves the potential for speedup on a platform supporting more CPUs and enhances the load balance across tasks. Second, since in OP and in the new DP implementation the tasks are created only once per row, then the higher quadrature results in a larger computational load per task thus diluting the effect of the task starting penalty. The measured Wall Clock speedup for these cases is plotted in Fig. 2 as a function of the input parameter *ncpu* with values ranging from -2 to 8, the number of processors on LANL's rho machine.

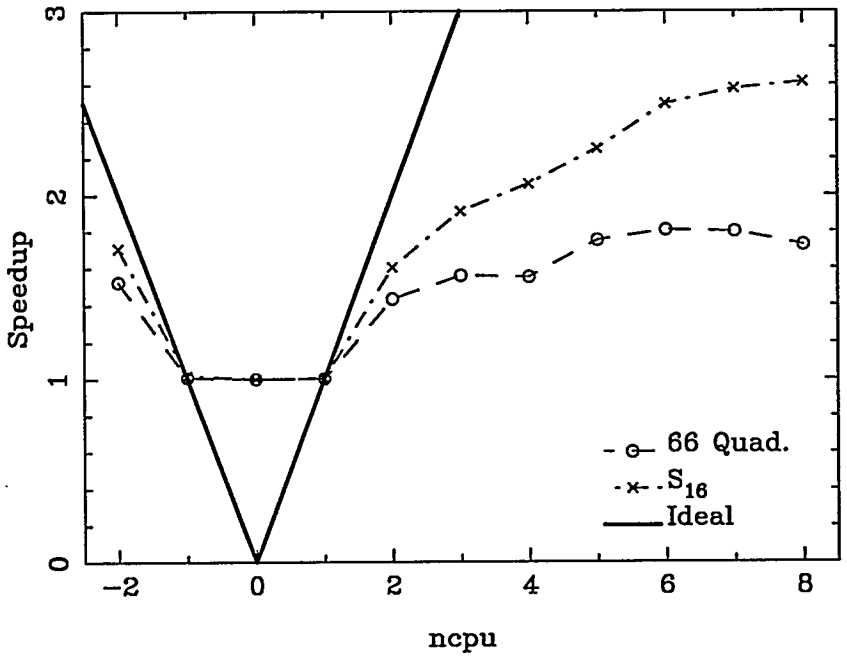
Figure 2. Measured Wall Clock Speedup for TP5 on Typically Loaded Machine.



The nondedicated environment of these runs implies dependence of the parallel performance on machine loading at the time of execution, hence the non-monotonic speedup with increasing *ncpu*. The results depicted in Fig. 2 were observed in single runs for each case shown (except when unusually high machine loading occurred) so they are representative of performance on a typically loaded

machine. In order to characterize the parallelization itself we repeated the run for each value of *ncpu,* until a sufficiently large and monotonically increasing machine utilization occurred. The measured Wall Clock speedup plotted in Fig. 3 is representative of machine performance on a lightly loaded machine. More importantly, it comes closer to characterizing performance in a dedicated environment, hence judging the success of the parallel algorithms employed in ideal circumstances.

Figure 3. Measured Wall Clock Speedup for TP5 on Lightly Loaded Machine.



Figures 2 and 3 exhibit good speedup of the respective computations which appear to increase with the order of angular quadrature and CPU availability. The relatively low achievable speedup, ~ 2.6, is due to the small size of TP5, especially the short row-length, and thus the smaller the computational load per task compared to parallelization overhead.

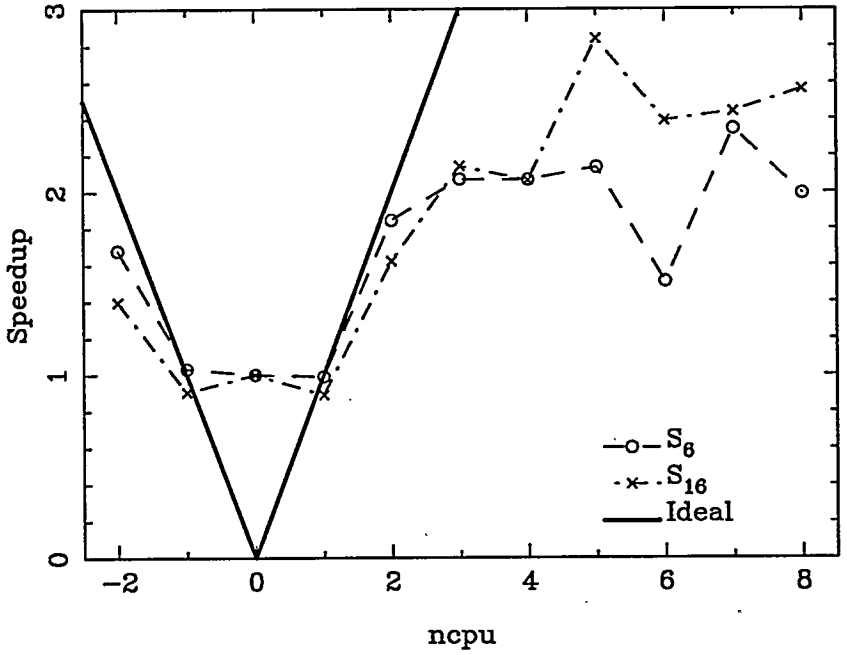
The output files containing the converged flux for each case shown in Figs. 2 and 3 were compared with the *ncpu*=0 case to verify correctness of the solution to all printed figures regardless of the number of tasks. Also the iteration histories agreed across runs for all but the first iteration in which the flux that undergoes the largest change is different between the single and multitasked runs. Upon examining this inconsistency we concluded that it is the result of finite arithmetic precision. More specifically, in the first iteration the previous iterate is a uniform 0 and the code computes the pointwise change in the scalar flux to be the ratio of the new iterate to itself multiplied by the zone importance. On an infinite precision computer this would produce a uniform value of 1 in zones with unit importance, and the point where the largest change is assigned would always be the last point tested in such zones. The finite precision of real computations, in contrast, perturbs this unity value by an arbitrary amount on the order of machine precision and the point of largest flux change in the first iteration is not necessarily the last one tested, but rather the one where the arbitrary perturbation is largest and positive. The parallel methods as coded in TORT accumulate the flux moments in the new iterate vector within a lock in subroutine *rownvp* in a random order depending on which task reaches, and arms, each lock first for the directions it owns. For example the sequential code, as well as the $|ncpu| = 1$ cases, accumulate all the angular flux contributions to the flux

moments for the right-to-left sweep first then for the left-to-right sweep, thus resulting in first, as well as later, iterates, that differ within machine precision from those obtained using multiple tasks. This difference translates into the observed difference described above in the first iterate only because in all later iterates the convergence errors are far more significant than roundoff error.

B. Test Problem 6 (TP6)

The second test problem is TORT's Test Problem 6 (TP6) except here we allow the slow group also to converge. This problem is large enough (2 groups, 60 directions, and 104,247 cells) to exceed the core memory objective and require I/O of the flux and source to a scratch file. It also tests the correctness of the parallel implementation when a fixed, but non-zero, boundary source is imposed on an x-edge which is particularly important to verify in the OP method. As with TP5, here also we measured the parallel performance on typically and lightly loaded machines as shown in Figs. 4, and 5, respectively, using S_6 and S_{16} angular quadratures. These results demonstrate the beneficial effect of problem size on parallel speedup in general and on lightly loaded machines in particular where the Wall Clock speedup exceeds 4.4.

Figure 4. Measured Wall Clock Speedup for TP6 on Typically Loaded Machine.

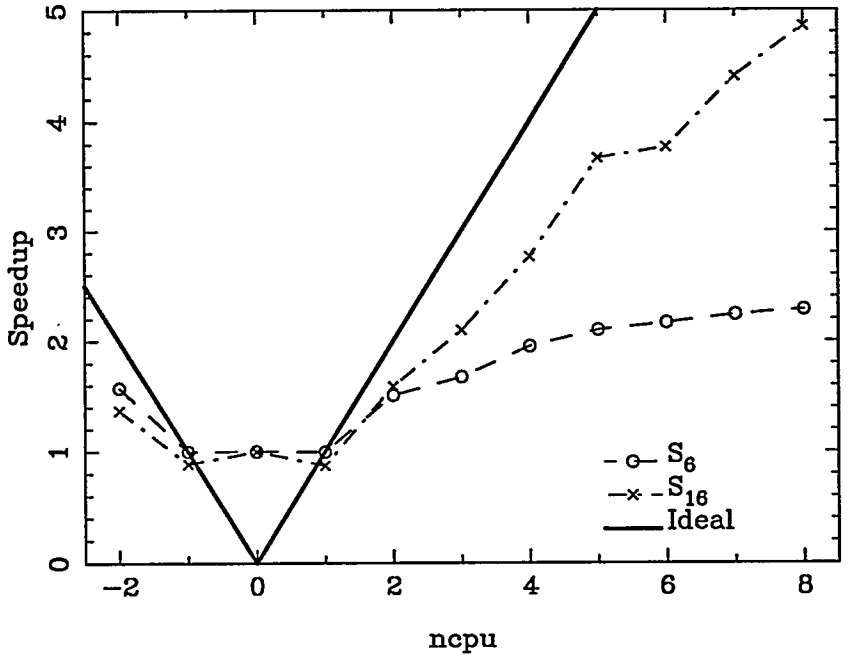


Comparison of the converged flux and iteration history for TP6, however, did not agree in the sixth, and rarely fifth, printed figure for all *ncpu* > 1 cases. We conjecture this is a roundoff effect that appears in TP6 because its solution is comprised of fluxes that are very small to the extent that they can be contaminated by roundoff errors resulting from the unpredictable sequence of arithmetic operations performed within the locks as discussed earlier. The full agreement between the printed solutions for the serial and multitasked cases for the other test problems attempted so far confirms the correctness of the parallel algorithm and its implementation.

C. DLVN Test Problem

The DLVN[4] problem used an S_{16} quadrature, P_3 flux moments expansion, 25 energy groups and 207,360 cells. The *ncpu* = 0 solution requires 910 min. of CPU time and 1125 min. of Wall Clock

Figure 5. Measured Wall Clock Speedup for TP6 on Lightly Loaded Machine.



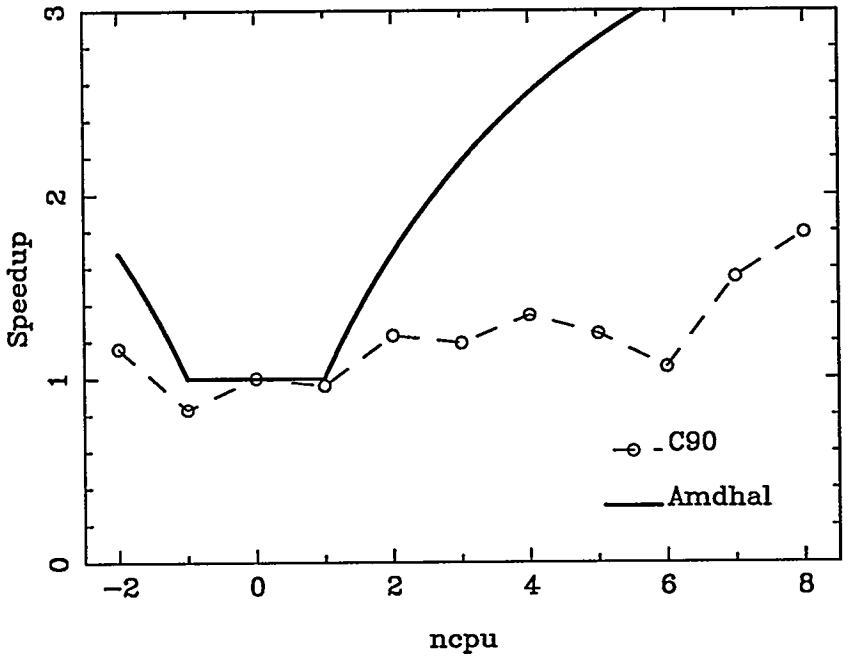
time on the C90 computer. The difference between the elapsed time and the CPU time is due in part to the typically high load on this machine, and in part to the significant amounts of I/O required by the problem.

The parallel speedup for the DLVN problem is shown in Fig. 6. The maximum speedup obtained was 1.8 for *ncpu* = 8. For comparison, Amdahl's Law is also plotted in Fig. 6 for a code with 81% parallelism. This is typical of the time TORT spends in the row sweep portion of the code for the DLVN problem. When *ncpu* = 8, the maximum theoretical speedup is 3.4; TORT achieves 52% of this value. The degradation from the theoretical speedup arises from two sources: the load on the machine and the increase in the parallel overhead with increasing number of tasks. While the load on the machine is not directly quantifiable, the increase in CPU time for the *ncpu* = 8 case was approximately 32% over the *ncpu* = 0 case, i.e., the parallel run on 8 processors required 1201 min. of CPU time versus 910 min. for the serial run. In the worst case, if one-eighth of the extra 291 min. of overhead are added to the sequential part of the *ncpu* = 0 run, then only 78% of TORT would be parallelized and Amdahl's Law would give a theoretical speedup on 8 processors of 3.1. TORT achieves approximately 57% of this value when *ncpu* = 8. Of course this assumes that the overhead occurs entirely in the parallel portion of code which is known not to be the case; some overhead is serial. Therefore, even on a lightly loaded machine the parallel speedup of the DLVN problem is bounded from above by 3.1.

V. CONCLUSION

We revived and fine-tuned the two multitasking options for the Linear Nodal method available in the TORT code to execute in parallel on Cray platforms running the UNICOS operating system. The two options represent coarse-grained angular domain decompositions in octants (OP) and individual discrete ordinates (DP) in the process of sweeping a row of computational cells. Our results exhibit speedup factors that approach 4.5 for some problems on a lightly loaded machine. However,

Figure 6. Measured Wall Clock Speedup for DLVN on Typically Loaded C90 at Lockheed Martin Corporation.



the observed performance demonstrates the high sensitivity of speedup to operating conditions beyond the user's control.

REFERENCES

1. W. A. Rhoades, R. L. Childs,"The TORT Three-Dimensional Discrete-Ordinates Neutron/Photon Transport Code," ORNL-6268 (November 1987).

2. Y. Y. Azmy, *Transport Theory and Statistical Physics,* **22,** 359 (1993).

3. W. A. Rhoades and R. E. Flanery, "3-D Discrete Ordinates Calculations with Parallel-Vector Processors," *Proc. ANS Topical Meeting on Advances in Nuclear Engineering Computation and Radiation Shielding,* Santa Fe, New Mexico, April 9-13, 1989, **69,** American Nuclear Society, LaGrange Park, IL (1989).

4. Douglas Allen Barnett, Jr., "Benchmark Measurements and Calculations of a Three-Dimensional Neutron Streaming Experiment," *Proc. ANS Topical Meeting on Advances in Mathematics, Computations, and Reactor Physics,* Pittsburgh, PA, April 28 - May 2, 1991, Vol. 2, **9.1 1,** American Nuclear Society, LaGrange Park, IL (1991).

DISCLAIMER