UCRL- JC-105427
PREPRINT

JAN 16 1991

A KILLER MICRO ATTACK ON 3D NEUTRON TRANSPORT

Milo R. Dorr and James M. Ferguson
Massively Parallel Computing Initiative
Lawrence Livermore National Laboratory
Livermore, CA 94550

This paper was prepared for submittal to
CompCon Spring '91, San Francisco, California
February 25-March 1, 1991

November, 1990

Lawrence
Livermore
National
Laboratory

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California 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 products, 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 the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

A Killer Micro Attack on 3D Neutron Transport *

Milo R. Dorr and James M. Ferguson

Massively Parallel Computing Initiative
Lawrence Livermore National Laboratory
Livermore, California 94550

Abstract : We describe the deterministic solution of the neutron transport equation and the computation of the effective criticality of three-dimensional assemblies using the BBN TC2000 *killer micros*. We observe that the performance of our research code PTRAN running on 48 processors of the TC2000 is competitive with the partially vectorizable version running on a single Cray Y/MP processor. This performance scales well with the number of processors on real problems, including those that are not load balanced *a priori*. To obtain this performance, we explicitly specify and exploit data locality and data dependence using domain decomposition and dynamic job scheduling.

Keywords: Neutron transport, effective criticality, BBN TC2000, Cray Y/MP, shared memory, private memory, data locality, load balance, parallel efficiency, domain decomposition, dynamic job scheduling.

1 Introduction

In this paper, we describe the deterministic solution of the neutron transport equation in three spatial dimensions on the BBN TC2000 *killer micros*. The TC2000 incorporates a microprocessor-based parallel architecture whose performance, based on theoretical estimates of the aggregate performance of each of its parts, rivals that of the Cray machines currently used for production. Our goal in this effort is to determine whether this potential can in fact be harnessed and focussed on an important application requiring substantial computational resources. Based on the results described below, we believe that this question has an affirmative answer. Given the scalability designed into the TC2000 architecture as well the prospect of more powerful microprocessors than those in the cur-

rent model, these preliminary results indicate that a significant increase in performance could be realized in a production machine based on this architecture.

2 Effective Criticality

The neutron transport problem we consider here is the following *effective criticality* problem. Find k and $\{\phi_g\}$ such that, for $g = 1, 2, \dots, G$,

$$\begin{aligned} \vec{\Omega} \cdot \vec{\nabla}_x \psi_g(x, \vec{\Omega}) + \sigma_g(x) \psi_g(x, \vec{\Omega}) \\ = \frac{1}{k} \sum_{g'} \tau_{g' \rightarrow g}(x) \phi_{g'}(x) \end{aligned} \quad (1)$$

$$\psi_g(x, \vec{\Omega}) = 0 \quad \text{for } x \in \partial D, \vec{\Omega} \cdot \vec{n} < 0$$

$$\phi_g(x) = \int_{S^2} \psi_g(x, \vec{\Omega}) d\vec{\Omega}$$

$$k = \sum_{g, g'} \int_D \tau_{g' \rightarrow g}(x) \phi_{g'}(x) dx$$

where $x \in D$ with $D \subset \mathbb{R}^3$ the spatial domain, $\vec{n}$ is the outward unit normal to the boundary of D , $\vec{\Omega}$ is a unit vector in the neutron direction, ψ_g is the angular neutron flux, ϕ_g is the scalar neutron flux, σ_g is the total cross section, and $\tau_{g' \rightarrow g}$ are the scattering cross sections. In Eq. (1), a standard multigroup discretization of the energy variation has been made, where we assume that increasing values of the multigroup index g correspond to decreasing energies. Given convergence tolerances ϵ_{outer} and ϵ_{inner} , Eq. (1) is solved by iteration as follows:

*Work performed under the auspices of the U. S. Department of Energy by the Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

Set $k_0 = \sum_{g,g'} \int_D \tau_{g' \rightarrow g} \phi_{g'} dx = 1$

For $i = 1, \dots,$

For $g = 1, \dots, G$

Set $\tilde{k}_0 = \sum_{g,g'} \int_D \tau_{g' \rightarrow g} \phi_{g'} dx$

For $j = 1, \dots,$

Solve

$$\vec{\Omega} \cdot \vec{\nabla} \psi_g + \sigma_g \psi_g = \frac{1}{k_{j-1}} \sum_{g'} \tau_{g' \rightarrow g} \phi_{g'} \quad (2)$$

for ψ_g

$$\text{Compute } \phi_g = \int_{S^2} \psi_g d\vec{\Omega} \quad (3)$$

$$\text{Set } \tilde{k}_j = \sum_{g,g'} \int_D \tau_{g' \rightarrow g} \phi_{g'} dx \quad (4)$$

If $\frac{|\tilde{k}_j - \tilde{k}_{j-1}|}{|\tilde{k}_j + \tilde{k}_{j-1}|} < \epsilon_{\text{inner}}$, then next g

Next j

Next g

Set $k_i = \sum_{g,g'} \int_D \tau_{g' \rightarrow g} \phi_{g'} dx$

If $\frac{|k_i - k_{i-1}|}{|k_i + k_{i-1}|} < \epsilon_{\text{outer}}$, then stop

Next i

Clearly, the majority of the computation and the greatest parallelization opportunities occur in the "inner iteration" (2)–(4), which is where we focus most of our attention.

In addition to the multigroup treatment of the energy dependence, we employ a standard discrete ordinates collocation (with symmetric quadrature, etc.) of the angular variable $\vec{\Omega}$. The assumed logically rectangular domain D is mapped to a rectangular computational domain, which is then partitioned into rectangular zones. A Petrov-Galerkin finite element method is then applied to the transport equation. That is, a piecewise trilinear representation of the angular flux ψ_g is substituted into the transport expressed in Cartesian coordinates, and the result is tested against piecewise constants in each zone, assuming constant cross sections. The resulting balance equation then couples the values of ψ_g at the eight nodes of a zone. Given a discrete angular direction $\vec{\Omega}_k$ and starting from known boundary conditions, each nodal value of $\psi_g(\vec{\Omega}_k)$ can be obtained by solving the balance equation for this value in terms of the other seven (upstream relative to the direction $\vec{\Omega}_k$) nodal values of $\psi_g(\vec{\Omega}_k)$. Thus, for each discrete angular direction, all of the nodal angular fluxes can be determined by a *transport sweep* of the grid. After the nodal angular fluxes have been calculated for each discrete direction, they are averaged to get values at zone centers,

then summed over all discrete angular directions to get zone-centered values of the scalar flux ϕ_g .

3 A Parallelization Strategy for the BBN TC2000

The BBN TC2000 currently installed at LLNL contains 63 Motorola 88100 reduced instruction set microprocessors running at 20 MHz. Each processor is located on a separate board, or *node*, along with a 16 kilobyte cache and 16 megabytes of local memory. The nodes are interconnected by a scalable switch. At boot time, some of each processor's local memory is dedicated to an interleaved shared memory pool. Thus, allocated memory is either *private* (accessible by only one processor making a memory reference within its own node) or *shared* (accessible by all processors via a memory reference through the switch). It takes 40, 10 and 3 clock cycles to access a word of shared, non-cached private and cached private memory, respectively. Fully-pipelined operations fed from the (non-coherent) cache can produce one result per clock. It is therefore clear that in order to obtain the maximum performance offered by the TC2000, computationally-intensive portions of code must be organized in such a way that medium to large scale "chunks" of data are located in private memory when they are required for computation. The strategy we have therefore devised to parallelize the inner iterations (2)–(4) of the effective criticality problem is as follows.

First, we place the entire problem to be solved in shared memory. We then decompose the spatial domain into subdomains, with the intent that processors will be assigned one of two types of jobs to do on each subdomain. The first type of job corresponds to a subdomain all of whose incident fluxes relative to some angular direction are known, either from boundary conditions or from a previous sweep over the upstream subdomains (for a rectangular domain decomposition, each subdomain will in general have 3 upstream neighbors in a given direction). In such a subdomain, the incident nodal fluxes and the zone-centered sources and cross sections are copied from shared to private memory. The angular fluxes are then computed at each node in the subdomain by a transport sweep in the given angular direction. Once the nodal angular fluxes are known, the zone-centered angular fluxes required for the later integration into zone-centered scalar fluxes are computed by averaging. The nodal angular fluxes exiting the domain, as well as the zone-centered angular fluxes, are then copied to

shared memory.

After a processor has performed a transport sweep of a subdomain in some direction, it then looks at each of the subdomain's downstream neighbors to see if the newly computed exiting nodal fluxes (which will be incident fluxes for the downstream neighbor) will allow the downstream subdomain to be swept. If so, the processor posts this information (essentially a subdomain number and an angle number) to a *sweep* queue; otherwise it simply increments a count of the number of upstream neighbors that have "checked in".

The second type of job corresponds to a subdomain whose zone-centered angular fluxes have all been computed. A processor assigned to such a subdomain copies these angular fluxes from shared memory into its private memory, integrates the angular fluxes to obtain scalar fluxes that are then copied back to shared memory. These new scalar fluxes enable the source for the next inner iteration and the corresponding contribution to the k update to be calculated as well. Subdomains on which this second type of job is ready to be performed are listed on an *integration* queue. Subdomains are posted to the integration queue by the last processor to calculate angular fluxes in the subdomain.

Note that in this strategy, do *not* statically assign processors to specific subdomains; any processor can perform any available work on any subdomain. We therefore avoid a host of load balancing problems that arise from making *a priori* assumptions about the availability of certain processors to do specific jobs. This is particularly important on the TC2000, where, because available resources are variably loaded during time-sharing, the execution times of identical code on different processors can vary slightly.

A second advantage of this procedure is that synchronization requirements are minimized. If a processor cannot find a subdomain to sweep (first job type), it looks for a subdomain whose scalar fluxes, updated sources and k contributions are ready to be calculated (second job type). The only critical regions are the accesses of the sweep and integration queues, which must be protected from multiple processors adding or removing jobs. These critical regions are very small however. Since all jobs on these queues are "ready", a queue access is simply the reading of a job index and the bumping of a pointer. The fact that processors do not have to search the subdomains to find work, but simply have to grab jobs from the queues (as well as post to the appropriate queue any work that it knows is ready to be done) is a crucial component of this algorithm.

4 Test Problems

Meaningful tests of this kind of code should contain all the important features of real, physical problems. These features include multiple materials, curved shapes mapped onto our Cartesian grid, abrupt material interfaces, true physical densities, and physically accurate nuclear cross sections. We have therefore calculated several reflected effective criticality problems that have these attributes. For convenience, we chose spherically symmetric assemblies. However, no symmetries were exploited in the numerical solution of these problems, which were run as truly 3D assemblies. In particular, the performance results reported below were obtained in the calculation of an assembly with 8.5 kg of plutonium surrounded by a 3.7 cm thick beryllium reflector [1].

All tests were performed with a set of 16-group cross sections, which is typical for these types of calculations. Since we are focussing only the inner iterations (2)-(4), using more or less groups would change the total problem running time, but would have no significant effect on the details of the timing. Also, for convenience while the multiprocessing algorithm was being developed, isotropic scattering was assumed as indicated in Eq. (1). Our experience with other 3D transport codes shows that the inclusion of P_1 scattering imposes some additional memory burden and a very moderate increase in CPU time in both the sweep and source routines. Inclusion of these terms would not change any of the conclusions of this study.

5 Performance Results

In this section, we present some performance results for the algorithm and test problem just described using a new version of our research code PTRAN. PTRAN was initially implemented on shared memory machines such as the Alliant FX/8 and the Cray X/MP and Y/MP [2]. PTRAN uses the PCP and PFP preprocessors [3] developed at LLNL to obtain parallel extensions to C and Fortran, respectively. We first consider PTRAN's absolute performance versus the current production supercomputer, which for us is the Cray Y/MP. We then investigate the *scalability* of the parallel algorithm described above.

In order to factor out as many problem-dependent variables as possible in the comparison of the TC2000 and Y/MP versions of PTRAN, we introduce the notion of a *phase space zone* (psz), which is one energy group, one angular direction, and one spatial zone. The times per psz reported in Table 1 are therefore

Table 1: Comparison of TC2000 and Y/MP versions of PTRAN.

	TC2000	Y/MP
CPU type	scalar	vector
cycle time (ns)	50	6
# of CPUs used	48	1
time/psz (μ s)	3.2	2.8

the times in microseconds required for the TC2000 and Y/MP to compute the angular and scalar fluxes in one angle and spatial zone within an inner iteration. As indicated in the table, 48 TC2000 processors were used in the comparison versus one Y/MP processor. It must be noted that the Y/MP version of PTRAN is unable to exploit the full vector performance of the Y/MP due to the fact that vector lengths can become very short during some parts of a transport sweep. We also remark that, due to some structural differences in the TC2000 and Y/MP versions of PTRAN that make a direct comparison of times difficult, the TC2000 time includes the time to compute the source and k updates whereas the Y/MP time does not. However, this accounts for less than approximately 10% of the time reported for the TC2000, and therefore the times are very close.

PTRAN is instrumented using an event logging facility provided with the TC2000. When a processor encounters a specified event point in the code, a microsecond clock is read and the time-stamped event is recorded in a memory resident event log. When the code completes, each processor writes its event log to a file. An X Windows client named *gist* is then used to read the event log file and present the data in graphical form. Using this tool, we are able to view the performance of the job scheduling strategy described in the previous section. In particular, we can see when a processor is sweeping a subdomain (i.e., performing a job of the first type) or is integrating angular fluxes and performing source and k updates (i.e., performing a job of the second type). We can also visualize processor idle time and the degree of load balancing achieved by the job scheduling algorithm.

Figure 1 contains the gist plot for the test problem with 8 angles running on 32 processors, using 40 zones and 4 subdomains in each coordinate direction. A subdomain is therefore a $10 \times 10 \times 10$ cube of zones. Each horizontal line is the process trace of a single processor computing one inner iteration (2)-(4). The

Figure 1: Gist plot for test problem with 8 angles running on 32 processors. The dimension of each subdomain is $10 \times 10 \times 10$ zones.

darkly-shaded regions denote time intervals in which the processor is sweeping a subdomain. The smaller, lightly-shaded regions denote time intervals in which the processor is integrating angular fluxes and performing source and k updates. Because there are 8 angles in the problem, at the beginning of the inner iteration only 8 subdomains have work to be done on them. Since the sweeping of each of these initial subdomains enables the sweeping of 3 more subdomains, we see 24 subdomains being acted upon in the second "wave". Finally, there is sufficient work for all processors. At the end of the inner iteration, the reverse situation occurs. That is, after a point there is less and less parallel work available, even though our job scheduling strategy is allowing some processors that could not find work on the sweep queue to grab work from the integration queue.

One way to reduce the amount of time processors sit idle waiting for work is to reduce the size of the subdomains. This correspondingly reduces the amount of time that a single processor spends on a subdomain. The 8 processors that have the first 8 subdomains will finish their work more quickly, resulting in a more rapid expansion of parallel work. Moreover, the smaller *granules* allow work to be distributed more smoothly across the processors, thereby improv-

Figure 2: Gist plot for test problem with 8 angles running on 32 processors. The dimension of each subdomain is $5 \times 5 \times 5$ zones.

ing load balancing. Figure 2 contains the gist plot of the same problem as before, except for the fact that each subdomain is a $5 \times 5 \times 5$ cube of zones. As can be seen in this figure, the processor idle time is greatly reduced and the load balance has indeed improved.

If load balancing is improved by using smaller subdomains, then the question arises as to how small subdomains can be made. Since a subdomain must contain at least one zone, this places a clear lower bound on subdomain size. However, considering the fact that in performing work on a subdomain, a processor must copy subdomain data from shared to private memory, perform some computations and copy results back to shared memory, it is clear that there must be a diminishing return in reducing subdomain size assuming that one does not want to spend more time "communicating" data than computing with it. Table 2 verifies this speculation, showing the communication (copy) and computation (real work) times for one transport sweep of subdomains of various sizes. If a subdomain contains 1000 or more zones, then the time to move data back and forth to shared memory is less than 20% of the time spent doing real work. As subdomains are made smaller, the percentage of time spent communicating with shared memory increases. Note also that the communication percentage does not drop appre-

Table 2: Communication (copy) versus computation times for one transport sweep of various-sized subdomains.

zones per subdomain	compute time (ms)	copy time (ms)	copy compute
8000	855	162	18.9%
1000	107	21.1	19.5%
125	13.5	3.60	26.0%
27	2.97	1.26	42.7%
1	0.226	0.271	120%

ciably in going from 1000 to 8000 zone subdomains. Therefore, as a general rule we use subdomains with 1000 zones in order achieve a suitable compromise between good load balancing and low communication overhead.

We should also remark that one further potential hazard in using smaller subdomains to achieve good load balancing is that it increases the probability that the critical regions containing the job queues will become bottlenecks. With processors spending less time on smaller subdomains, they will be accessing the job queues more frequently which can result in memory contention and subsequent delays. This situation has not been observed for problems we have run to date, except in the case of single zone subdomains, which have already been discounted for the reasons cited above. Nevertheless, Amdahl's Law states that we will eventually have to face this problem if the number of processors is sufficiently increased. One remedy is to break the job queues up into subqueues in a hierarchical fashion, allowing each processor to access only a particular subqueue. At the price of potentially having processors unable to easily "see" available work on other subqueues when there is no work on their own, the number of potential critical region collisions can thereby be controlled.

Finally, we consider the parallel processing efficiency of our algorithm. We define the efficiency using N processors as the time to compute an inner iteration (2)-(4) on one processor divided by the time to compute an inner iteration on N processors divided by N . In other words, the efficiency is the "speedup" in the time to compute an inner iteration using N processors instead of just one, divided by N . Figure 3 shows efficiencies observed for a problems with (i) 2 subdomains and 20 zones in each coordinate direction, (ii) 4 subdomains and 40 zones in each coordinate direction, and (iii) 8 subdomains and 80 zones in each

Figure 3: Parallel efficiencies for some 8 angle test problems.

coordinate direction. Thus, the subdomain dimension is $10 \times 10 \times 10$ zones for all three problems. Only eight angles are used in these problems, which are therefore *not* ideally load-balanced. With the exception of the 2×20 problem, in which there are only 8 subdomains and therefore the available parallelism is quickly exhausted, the efficiencies are still above 80% when as many as 32 processors are used. Figure 4 shows a comparison of parallel efficiencies obtained on test problems with 40 zones and 4 subdomains in each coordinate direction, using 8 and 48 angles. In both Fig. 3 and Fig. 4, we believe that the apparent slow linear degradation in efficiency is due to the fact that, even when only one processor is used, *all* nodes are accessed over the switch. Thus, the processor-to-switch bandwidth effectively decreases (linearly) as the number of processors increases.

6 Summary

In conclusion, we have seen that the performance of our PTRAN code running on 48 processors of the BBN TC2000 is competitive with the partially vectorizable version running on a single Cray Y/MP processor. This performance scales well with the number of processors on real problems, including those that are not load balanced *a priori*. Obtaining this performance required the explicit specification and exploitation of data locality and data dependence using domain de-

Figure 4: Parallel efficiencies for 8 and 48 angle test problems.

composition and dynamic job scheduling. From these results, it would seem that, for this application, a production machine based on the TC2000 architecture with more powerful processors and a commensurate increase in switch speed could yield a significant gain in our design capability.

References

- [1] C. C. Cremer, R. E. Hunter, J. H. Beljin and D. R. Worlton, "Comparison of Calculations with Integral Experiments," LA-3529, Los Alamos Scientific Laboratory, Los Alamos, NM, 1969.
- [2] M. R. Dorr, J. M. Ferguson and A. Greenbaum, "Solving the Neutron Transport Equation in Three Dimensions with a Micro-Tasked Finite Element Method," in *Proceedings of the 1988 Nuclear Explosives Code Developers' Conference*, Boulder, CO.
- [3] Eugene D. Brooks III, "PCP: A Parallel Extension of C that is 99% Fat Free," UCRL-99673, Lawrence Livermore National Laboratory, Livermore, CA, 1988.

- END -

DATE FILMED

02 / 13 / 91