THE APPLICATION OF A THREADS PARALLEL COMPUTING MODEL TO THE U.S. NRC NEUTRON KINETICS CODE PARCS

Deokjung Lee and T. J. Downar
School of Nuclear Engineering
Purdue University
West Lafayette, IN 47907-1290 USA
lee100@ecn.purdue.edu ; downar@ecn.purdue.edu

ABSTRACT

The U.S. NRC Neutron Kinetics code PARCS has been coupled to the systems thermal hydraulic analysis codes RELAP5 and TRAC-M for best estimate analysis of Light Water Reactors. Several coupled code applications have been performed such as the OECD PWR Main Steam Line Break Benchmark and the OECD BWR Peach Bottom Turbine Trip Benchmark. In both these cases, it was evident that the computational burden needs to be reduced if coupled code analysis is to become an effective tool for LWR analysis. Efforts have been initiated to investigate more efficient numerical methods for solving the coupled temperature/fluid and neutron field equations. The work described here is a companion effort which investigates the use of parallel computing in PARCS to reduce the computational burden. Because the neutron diffusion equation is tightly coupled and because the target platforms for coupled codes has been multiprocessor workstations, the focus of attention in the work here has been on the use of multi-threading to achieve parallelism in PARCS.

The specific applications to be discussed here are the use of POSIX Threads and OpenMP on multiprocessor SUN, SGI, and LINUX PC workstations. The achievable parallel performance for practical applications is compared for each of the code modules using POSIX threads and OpenMP. A detailed analysis of the cache misses was performed to explain the observed performance. Considering the effort required for implementation, the directive based standard OpenMP appears to be the preferred choice for parallel programming on a shared memory address machine.

1. INTRODUCTION

The Purdue Advanced Reactor Core Simulator (PARCS) code[1] was developed by the United States Regulatory Commission for the safety analysis of nuclear reactors operating in the United States. The code solves the time-dependent Boltzmann transport equation for the neutron flux distribution in the nuclear reactor core. The code is written in FORTRAN and has been executed on a wide variety of platforms and operating systems. The code has been extensively benchmarked and results are documented in numerous reports[2] for a variety of Light Water Reactor (LWR) applications.

The computational burden in PARCS for solving practical reactor problems can be considerable even on the most efficient workstations. The execution time depends on the type of reactor transient simulated, but typically the code runs about an order of magnitude slower than real time. One of the primary objectives of the work performed here was to investigate multiprocessing as a means to reduce the computational burden with the ultimate goal of real time nuclear reactor simulation. The

target platform for this work is a typical shared memory space workstation and therefore threading was the preferred parallel model. This paper will describe experiences with using POSIX Threads [3] (Pthreads) and OpenMP[4] in achieving parallel computing with the PARCS code on SUN, SGI, and LINUX PC.

2. PHYSICS AND COMPUTATIONAL MODEL

During the analysis of reactor transients in PARCS, the primary equation solved is the transient fixed source problem which can be discretized using conventional methods resulting in an elliptic PDE. The physical core model contains about 200 fuel assemblies, each of which contains about 250 fuel pins. Because the mean free path of the neutron is about the same as the distance between fuel pins, the resulting three-dimensional spatial mesh could be on the order of a million. However, innovative "multi-level" type methods have been developed over the years in which the pin by pin flux is solved only at the local level and the global problem is solved with mesh spacing the same size as the fuel assembly. The resulting local/global iteration introduces some natural parallelism since the local problems (NODAL module) can all be solved simultaneously. Innovative domain decomposition methods were also introduced to solve the global (CMFD module) problem. The Coarse Mesh Finite Difference (CMFD) solution is based on a Krylov semi-iterative method, Bi-Conjugate Gradient Stabilized (BICGSTAB), accelerated with a preconditioning scheme based on a blockwise incomplete LU factorization. Parallelism is achieved using an incomplete domain decomposition preconditioning method [1]. In addition to the Boltzmann transport equation, the single phase fluid dynamics and one-dimensional heat conduction equations are solved in PARCS to provide the temperature/fluid field in the core. This is performed in the T/H module of PARCS. The feedback to the neutron field equations is provided through the cross sections which is performed in the XSEC module of PARCS.

Parallelism is achieved in PARCS by first dividing the code into the four basic modules which solve distinct aspects of the coupled neutron and temperature/fluid field calculation: CMFD, Nodal, T/H, and XSEC. The last three of these modules are easily parallelizable since all calculations are performed on a node by node basis and can be evenly assigned to separate threads. Parallelism of the CMFD calculation is more difficult and is achieved by domain decomposition. In the analysis reported here domain decomposition is applied by dividing the reactor core into several axial planes. In the model used here the reactor core is divided into 18 axial planes. In some cases, load imbalance will contribute to the loss of parallel efficiency since the number of axial planes will not always be an even multiple of the number of processors.

3. IMPLEMENTATION OF POSIX THREADS AND OPENMP

The target architecture for multiprocessing with PARCS was a shared memory address space machine since much of the parallelism was expected to be very fine grain. In general, the overhead for message passing parallelism is most suitable for applications which are predominantly coarse grain. Pthreads and OpenMP are the standard techniques available for multithreaded programming. OpenMP is an Application Program Interface (API), and it is composed of a set of compiler directives and library routines. In fact, OpenMP can be implemented using Pthreads. In the work here, both Pthreads and OpenMP were implemented on SUN and SGI workstations. For the LINUX PC implementation, Pthreads is applied using the NAGWare FORTRAN compiler since it does not support OpenMP. The characteristics of the machines used here are shown in Table 1.

Table 1. Specifications of Multiprocessor Workstations

Platform	SUN ULTRA-80	SGI ORIGIN 2000	LINUX PC
Number of CPUs	2	32	4
CPU Type	ULTRA SPARC II 450 MHz	MIPS R10000 250 MHz 4-way superscalar	Intel Pentium-III 550 MHz
L1 Cache	16 KB D-cache 16 KB I-cache Cache-line : 32bytes	32 KB D-cache 32 KB I-cache Cache-line : 32bytes	16 KB D-cache 16 KB I-cache
L2 Cache	4MB	4MB per CPU Cache-line : 128bytes	512KB per CPU
Main Mem.	1GB	16GB	1GB
FORTRAN Compiler	SUN Workshop 6 - f90 version 6.1 & 6.2	MIPSpro Compiler - f90 version 7.2.1 & 7.3.1	NAGWare - f90 version 4.2

A decision had to be made regarding the general approach used to multithread a program. One approach is to fork each subroutine which contains a parallel region. The benefit of this approach is that it minimizes the global vs. local variable concerns. However, it introduces the possibility of performance loss due to the overhead creating a thread with Pthreads, which is particularly true if the subroutine is in a loop. It is very possible that the overhead associated with the creation of a thread could override any performance improvement by parallel computations. To avoid this problem and minimize the overhead in the application of Pthreads and OpenMP to PARCS, the main program was forked at the beginning of execution.

Since Pthreads does not support FORTRAN, a mixed programming was required to make the Pthreads library available to PARCS. To access Pthreads library functions, PARCS calls C functions in which Pthreads are available directly[5]. Pthreads required considerably more effort than OpenMP to implement within the existing PARCS code. Caution had to be used with Pthreads to avoid such problems as race conditions and false sharings. Conversely, the implementation of parallelism with OpenMP was much easier since it was simply a matter of inserting OpenMP directives for parallel regions and specifying which variables were shared and/or private.

4. APPLICATIONS AND RESULTS

The performance of Pthreads and OpenMP in PARCS was analyzed using a control rod ejection transient benchmark problem[6] which models the ejection of a control assembly from an initially critical core at hot, zero power conditions. There is a significant redistribution of power in the core and the possibility of a local energy deposition that could result in the melting of a fuel rod. The focus of the results here is on the neutron flux solution, but the transient also requires solution of the temperature/fluid field equations. The multithreaded solution of the OECD benchmark was performed with PARCS and as shown in Table 2, all threaded versions provide the consistent results with the serial execution.

Table 2. Comparison of Serial and Multi-threaded Code Execution

Peak	serial	Number of Threads			
		1	2	4	8
Time (sec)	0.275	0.275	0.275	0.275	0.275
Power (%)	434.1	434.1	434.0	434.2	434.2

4.1 PARALLEL PERFORMANCE ON A SUN WORKSTATION

The execution time and parallel performance on the SUN machine are summarized in Table 3 for the Pthreads and OpenMP versions of the code. The number of updates in the table indicates the number of times each module must be executed in order to solve the benchmark problem. As indicated in the table, the OpenMP performance on the SUN is poor, whereas the Pthreads performance is reasonably good.

Table 3. Summary of Parallel Execution Times on the SUN
(Fortran Compiler v6.1)

Module		serial	OpenMP			POSIX threads		
			1 th.	2 ths.	Speedup	1 th.	2 ths.	Speedup
Time (sec)	CMFD	36.7	107.0	59.0	0.62	32.1	20.8	1.77
	Nodal	11.5	22.8	17.4	0.66	11.3	6.4	1.78
	T/H	29.6	32.6	19.1	1.56	27.9	14.5	2.04
	Xsec	7.6	47.5	24.3	0.31	7.1	3.7	2.04
	Total	85.4	209.8	119.8	0.71	78.5	45.5	1.88
# of Updates	CMFD	445	445	456	-	445	456	-
	Nodal	31	31	33	-	31	33	-
	T/H	216	216	216	-	216	216	-
	Xsec	225	225	226	-	225	226	-

The speedup achieved with the parallel Pthreads version is considerably different for each module. For example, the T/H and XSEC module show superlinear speedup while the CMFD and NODAL module speedup is only 1.77 and 1.78 respectively. As noted earlier, T/H and XSEC modules are inherently parallel and therefore superlinear speedup is possible. Conversely, the solution of the elliptic PDE in the CMFD module uses domain decomposition methods and as indicated in the Table, there is a slight increase in the number of linear solutions required because of additional iterations. Also, the number of updates in the NODAL module increased due to the increased number of calls to the CMFD module. The increase of the number of updates for 2-threads case in CMFD and NODAL modules was small, just 2.5% and 6.5%, respectively.

The OpenMP execution time with 2-threads was actually slower than the execution time with a serial version of the code. This was primarily a result of the high overhead for OPENMP threads implementation as seen by the large increase in the time required for serial and 1-thread execution in Table 3. The reason for this high overhead was a deficiency in the OpenMP implementation in the SUN FORTRAN compiler version 6.1. For historical reasons of directive based multi-threading on the SUN compiler, it was assumed that all global variables were volatile and their value could change because of instructions performed by another thread. Therefore every time the data of a variable was required, it had to be fetched from memory even when the data already existed in a register due to the previous access. This overly conservative assumption about shared variables in

OpenMP resulted in considerably more memory accesses than necessary and consequently a much longer execution time[8]. This deficiency in the implementation of OPENMP was apparently fixed in version 6.2 of the SUN FORTRAN compiler. The results for execution of the rod eject problem are shown in Table 4 which shows a reasonably good speedup of 1.74 for 2 threads. However some performance degradation from serial to a single thread still exists (e.g., XSEC module).

Table 4. Summary of Parallel Execution Times on the SUN
(Fortran Compiler v6.2)

Module		serial	OpenMP		
			1 th.	2 ths.	Speedup
Time (sec)	CMFD	33.0	34.2	19.1	1.73
	Nodal	15.4	17.3	9.5	1.63
	T/H	28.9	28.9	14.9	1.94
	Xsec	6.3	8.4	4.5	1.40
	Total	83.7	88.7	48.0	1.74

4.2 PARALLEL PERFORMANCE ON A SGI WORKSTATION

The parallel performances of PARCS on the SGI are summarized in Tables 5 and 6 for implementation with versions 7.2.1 and 7.3.1 of the FORTRAN compiler. Parallel speedup could not be achieved with Pthreads on the SGI since all threads are scheduled to one CPU. This deficiency in the SGI compiler was confirmed by examining the status of CPU usage during runtime. The false scheduling of Pthreads on the SGI machine was observed only for FORTRAN and C mixed language programming; in C only program, Pthreads showed acceptable parallel performance.

Table 5. Summary of Execution Time on SGI
(FORTRAN compiler v7.2.1)

Module		serial	OpenMP					POSIX threads		
			1	2	Speedup	4	Speedup	1	2	Speedup
Time (sec)	CMFD	19.8	19.3	12.1	1.63	8.93	2.21	19.4	20.9	0.95
	Nodal	9.0	9.2	5.8	1.55	3.56	2.53	9.2	9.7	0.92
	T/H	26.6	25.3	12.3	2.17	8.92	2.99	25.2	25.5	1.05
	Xsec	4.8	4.4	2.4	2.01	1.37	3.53	4.8	5.0	0.97
	Total	60.2	58.1	32.6	1.85	22.78	2.64	58.6	61.1	0.99

The OpenMP performance on the SGI is very good. For compiler version 7.2.1, the speedup (1.85) achieved with 2 threads using OpenMP is comparable to the 2-thread performance achieved with Pthreads on the SUN (1.88). In particular, the T/H and XSEC modules show a superlinear speedup, whereas the CMFD and NODAL modules show a relatively low speedup.

For version 7.3.1, OpenMP performance is slightly worse than version 7.2.1 (1.77 vs. 1.85) and also each module shows different performance; T/H module shows the highest speedup in 7.2.1 and CMFD in 7.3.1. As will be discussed in the section 4.4, this can be attributed in part to the efficiency of the cache utilization in each module. The speedup of 4 processors is poor, primarily because of problems balancing the load. As noted earlier, domain decomposition was implemented on a plane-wise basis and the 18 planes in the model are not evenly assignable to 4 processors.

Table 6. Summary of Execution Time on SGI
(FORTRAN compiler v7.3.1)

Module		serial	OpenMP					POSIX threads		
			1	2	Speedup	4	Speedup	1	2	Speedup
Time (sec)	CMFD	45.3	39.6	23.3	1.95	17.06	2.66	47.4	48.4	0.94
	Nodal	9.6	9.5	5.9	1.64	3.78	2.54	9.6	10.1	0.96
	T/H	24.3	24.3	15.6	1.56	8.23	2.95	25.3	25.3	0.96
	Xsec	3.6	3.5	2.1	1.75	1.27	2.82	4.9	4.9	0.73
	Total	82.8	76.9	46.8	1.77	30.35	2.73	87.1	88.7	0.93

4.3 PARALLEL PERFORMANCE ON LINUX PC

In Table 7, the parallel performance of PARCS on the LINUX PC is summarized for the Pthreads version of the code. The numbers of updates of each module are consistent with the those obtained on the SUN machine as noted in Table 3. However, the parallel performance achievable on the LINUX PC is poor, particularly for the CMFD module does not show any speedup for 2 threads. The low parallel performance can be directly attributed to the inefficiency of memory allocation for the arrays in a subroutine.

Table 7. Summary of Execution Time on the LINUX PC

Module		serial	POSIX threads				
			1	2	Speedup	4	Speedup
Time (sec)	CMFD	46.9	47.3	45.6	1.03	69.31	0.68
	Nodal	7.0	9.0	5.3	1.32	3.62	1.92
	T/H	25.0	32.2	16.3	1.53	8.56	2.92
	Xsec	5.0	5.6	3.0	1.67	1.92	2.60
	Total	83.8	94.0	70.2	1.19	83.40	1.00
# of Updates	CMFD	445	445	456	-	497	-
	Nodal	31	31	33	-	38	-
	T/H	216	216	216	-	216	-
	Xsec	225	225	226	-	228	-

This was confirmed by performing a simple experiment. In Table 8, the time for a dummy subroutine call (i.e., does nothing but return) is measured on SUN and LINUX PC. The size of an array in the subroutine is determined at runtime. Therefore when the subroutine is called, there is a call to the memory allocating function to assign the desired amount of memory in the heap region of the memory. For multi-threaded execution, each thread will call this subroutine and multiple executions of memory allocation will occur at the same time. In an ideal case, the multiple memory allocations will occur concurrently without any measurable increase of running time. On the SUN machine, the time increases just 0.5 % from 1 thread to 2 threads. However, the memory allocation time increases by more than a factor of 10 on the LINUX PC. Even though not all of the subroutines in PARCS have such arrays, once a subroutine is called a sufficient number of times, a serious performance problem will occur for the multi-threaded run. The memory allocation function is performed in the subroutine "sol1d" of the CMFD module in PARCS. It is called about 1e6 times in the transient benchmark problem and is specifically accountable for the poor multithread performance on the LINUX PC.

Table 8. Time for Single Dummy Subroutine Call

Unit: sec

Platform	Number of Threads	
	1	2
SUN	3.66E-07	3.68E-07
LINUX	1.66E-06	7.60E-05

4.4 CACHE PERFORMANCE ANALYSIS

The processors on the platforms used in the analysis here are relatively fast, which suggests that data access times will be particularly important to improve overall code performance. Previous studies[7] have shown that cache utilization, particularly the L2 cache hit ratio, can be important for achieving good performance on modern workstations.

Cache performance was analyzed for each PARCS module using the hardware counter library available on the SGI platform. For the analysis shown here a consistent level 2 optimization was used for both serial and threaded version of code. The data cache misses for each module are summarized in Table 9 and Table 10 shows the cache miss ratios which means normalized to the cache misses of serial execution. The normalized L2 cache misses for 2 threads are 1.71 for XSEC, 1.66 for CMFD, and 1.60 for NODAL. This correlates reasonably well the speedups shown in Table 5 for XSEC, CMFD and NODAL of 2.01, 1.63 and 1.55, respectively. This suggests that for these three modules the overall performance is determined primarily by the L2 cache utilization. This is reasonable since, as shown in Table 11, an unsatisfied L2 cache miss can result in an order of magnitude penalty in machine cycles.

Table 9. Data Cache Misses on SGI

Compiler	Module	cache	Serial	Number of Threads	
				1	2
f90 v7.2.1	CMFD	L1	477,691	479,474	258,027
		L2	28,242	29,650	17,007
	NODAL	L1	857,744	853,866	444,849
		L2	54,163	55,534	33,846
	TH	L1	165,133	60,587	39,419
		L2	9,551	9,512	9,673
	XSEC	L1	62,324	57,462	29,845
		L2	9,456	9,518	5,517
f90 v7.3.1	CMFD	L1	546,275	545,431	290,093
		L2	28,637	29,248	17,041
	TH	L1	63,499	53,650	31,474
		L2	9,502	10,098	9,360

Table 10. Data Cache Misses on SGI (Normalized)

Compiler	Module	cache	Serial	Number of Threads	
				1	2
f90 v7.2.1	CMFD	L1	1.00	1.00	1.85
		L2	1.00	0.95	1.66
	NODAL	L1	1.00	1.00	1.93
		L2	1.00	0.98	1.60
	TH	L1	1.00	2.73	4.19
		L2	1.00	1.00	0.99
	XSEC	L1	1.00	1.08	2.09
		L2	1.00	0.99	1.71
f90 v7.3.1	CMFD	L1	1.00	1.00	1.88
		L2	1.00	0.98	1.68
	TH	L1	1.00	1.18	2.02
		L2	1.00	0.94	1.02

Table 11. Typical Memory Access Cycles on the SGI

Memory Access Type	Cycles
L1 cache hit	2
L1 cache miss satisfied by L2 cache hit	8
L2 cache miss satisfied from main memory	75

The behavior of the T/H module does not appear to follow this trend since the normalized L2 data cache misses for 2 threads is very low (0.99) while the parallel performance is excellent (2.17). For the T/H module it appears that the L1 data cache plays a more important role. As shown in Table 10, the L1 data cache miss ratio is about 20 times larger than L2 cache data cache miss ratio. As also shown in Table 10, the L1 data cache misses for the threaded code are reduced by a factor of 4, which is very large compared with the other modules. Examination of the assembly code revealed some fundamental differences in the way the compiler treated the serial and threaded versions of the T/H subroutine. For example, the threaded version of the code had considerably more pre-fetch commands.

In order to relate code performance to machine specific characteristics, the speedup was estimated using the data access times:

$$S \approx \frac{T_{total}^{serial}}{T_{total}^{2threads}} \tag{1}$$

where

T_{total}^{serial} = Total data access time for serial execution

T_{total}^{2th} = Total data access time for 2 threads execution.

And the total data access time is approximated by L2 cache and main memory access time:

$$T_{total} \approx T_{L2} + T_{mem} = n_{L2} \cdot t_{L2} + n_{Mem} \cdot t_{Mem} \tag{2}$$

where

T_{L2} = Total L2 cache access time
T_{mem} = Total memory access time
n_{L2} = Number of L1 data cache misses satisfied by L2 cache hit
n_{Mem} = Number of L2 data cache misses satisfied from main memory
t_{L2} = L2 cache access time for 1 word
t_{Mem} = Main memory access time for 1 word.

Based on the above simple model and the data shown in Tables 9 and 11, the speedup for each module were estimated in Table 12. As indicated, the results are in reasonable agreement with the actual measured results. As noted in the previous section, Although CMFD and TH modules show much different parallel performance depending on the compiler version, the speedup estimation model of Eq. (1) still works well for both versions of compiler as shown in the table.

Table 12. Estimated 2-Thread Speedup
Based on Data Cache Misses for OpenMP on the SGI

Compiler	Module	Speedup	
		Measured	Predicted*
f90 v7.2.1	CMFD	1.63	1.78
	NODAL	1.55	1.80
	TH	2.17	2.04
	XSEC	2.01	1.86
f90 v7.3.1	CMFD	1.95	1.82
	TH	1.56	1.30

* : Predicted by Eq. (1)

4. CONCLUSIONS

In an effort to reduce the computational burden for estimate reactor analysis, the threads parallel computing model was applied to the U.S. NRC neutronics code PARCS. Two parallel versions of the nuclear reactor transient analysis code PARCS were developed using Pthreads and OpenMP. The parallel performance was analyzed on SUN, SGI, and LINUX PC workstations. Some unresolved issues remain regarding implementation of Pthreads on the SGI and on the LINUX PC.

The overall performance of OpenMP on the SGI was comparable to Pthreads on the SUN. The performance varied depending on the type of calculations performed in each module. A simple predictive model was developed based on cache access times and the results agreed reasonably well with measured performance. Considering the effort required for implementation, the directive based standard OpenMP appears to be the preferred choice for parallel programming on a shared memory address machine.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the support of the U.S. NRC during the performance of this research.

REFERENCES

1. H.G. Joo and T.J. Downar, "An Incomplete Domain Decomposition Preconditioning Method for Nonlinear Nodal Kinetics Calculations," *Nuc. Sci. Eng.*, **123**, p.403 (1996).

2. H.G. Joo and T.J. Downar, "PARCS: Purdue Advanced Reactor Analysis Code", *International ANS Reactor Physics Conference*, Mito, Japan (1996).

3. Bil Lewis and Daniel J. Berg. *THREADS PRIMER*. Californaia, USA: SunSoft Press (1996).

4. OpenMP. A proposed industry standard api for shared memory programming. *http://www.openmp.org/*.

5. D.J. Lee and T.J. Downar, "The Application of POSIX Threads and OpenMP to the U.S. NRC Neutron Kinetics Code PARCS," *Proc. Intl. Workshop on OpenMP Applications and Tools*, WOMPAT 2001, p. 90, West Lafayette, USA (2001).

6. H. Finnemann, et al., "Results of LWR Core Transient Benchmarks," *Proc. Intl. Conf. Math. And Supercomp. In Nuc. App.*, p.243, Karlsruhe, Germany (April, 1993).

7. Q. Wang, "A Parallel Computing Model for the TRAC-M Code," M.S. Thesis, School of Nuclear Engineering, Purdue University, December, 1999.

8. Larry Meadows, SUN Microsystems Inc., personal communication