Scowl: A Tool for Characterization of Parallel Workload and its Use on Splash-2 Application Suite

Darko Marinov, Davor Magdić, Aleksandar Milenković,
Jelica Protić, Igor Tartalja, and Veljko Milutinović
Department of Computer Engineering
School of Electrical Engineering, University of Belgrade
POB 35-54, 11120 Belgrade, Yugoslavia
E-mail: marinov@sag.lcs.mit.edu, {emilenka, etartalj, VM@etf.bg.ac.yu}, jeca@sezampro.yu

Abstract

This paper concentrates on the problem of defining and measuring parameters that characterize typical behavior of parallel applications targeted to Distributed Shared Memory (DSM) systems and Shared memory MultiProcessors (SMP). These parameters can be used as input to various models for performance evaluation in this research area. Furthermore, typical application behaviors can be recognized, which can help to generate new ideas for improvements of memory consistency protocols, adapting them to specific application characteristics. Our study encompasses a variety of parameters, such as frequencies of operations of various access types (private read/writes, shared read/writes, lock operations, barrier operations), average number of accessed blocks per interval, average number of modified words, etc. Results presented in this paper are based on the SPLASH-2 application suite. The developed instrumentation tool Scowl, along with applied simulation environment, Limes, are publicly available and applicable for performing measurements on other parallel applications as well.

1. Introduction

At the starting point of any analytical modeling or simulation analysis with synthetic benchmarks, one is faced with a difficult problem of determining realistic values of input parameters of the model, especially in the research field of multiprocessors. Some authors mostly use educated guesses to choose appropriate values [2], the others concentrate on published values obtained by characterization of existing benchmarks such as [9], while some combine both approaches [8]. However, in some studies that concentrate on specific aspects of applications and protocols, parameters of interest can not be found in the open literature and have to be determined by measurements on existing benchmark suits.

There are several tools used to create an illusion of multiprocessor on a single processor machine, such as TangoLite [1] developed at the Stanford University. One of them is Limes [3], developed at the University of Belgrade. Limes runs on a PC with an i486 or higher, on a Linux operating system, and executes parallel applications written in C, using specific (ANL) macros to express parallelism. Limes is based on execution-driven approach, and can be used for architecture evaluation studies as well as parallel algorithms evaluation. The idea presented in this paper is to extend Limes with Scowl, a measuring instrumentation back-end, capable of providing a wide set of parameters relevant for evaluating SMP/DSM systems with relaxed memory consistency models [5]. Our main goal was to provide the software tool that enables: (a) execution-driven collecting of all relevant data (b) generation of customized, reduced traces suited to specific needs (c) simultaneous generation of traces and execution-driven measurements. This software should be modular in order to enable easy widening of the set of parameters to be measured.

Early research efforts in shared-address-space multiprocessing were based on small workloads consisting of a few simple programs. Using different programs and different problem sizes made comparisons across studies difficult. In a way, the problem was surpassed with SPLASH (Stanford ParalleL Applications for SHared Memory) [7]. Although SPLASH has provided a degree of consistency and comparability across studies, it has many limitations. First, SPLASH contains only a small number of programs and even more it does not cover all aspects of scientific and engineering computing. Also, the implementations of SPLASH programs are not optimized for modern memory system characteristics and for machines that scale beyond relatively small number of processors. To overcome these limitations the SPLASH suite has been expanded and modified to include several

0-7695-0728-X/00 $10.00 © 2000 IEEE

new programs as well as improved versions of the original SPLASH programs. The resulting SPLASH-2 suite [9] contains programs that (a) represent a wider range of computations in scientific, engineering and graphics domains; (b) use better algorithms and implementations; and (c) are more architecturally aware. Therefore, we used Splash-2 benchmark suit as a subject of measurements that we performed using Scowl.

2. The Essence of Scowl

The essence of our approach to characterization of parallel workload aimed to DSM/SMP systems is: (a) to define a list of parameters important for analytical or simulation study of DSM/SMP systems, according to our experiences in analyzing such systems [8], [6], [4], (b) to classify the parameters according to some common characteristics into semantic groups, (c) to design Scowl - a tool that extends Limes, event driven simulator of parallel applications, in order to enable measurements of relevant parameters, (d) to perform experiments that measure relevant parameters on a subset of SPLASH-2 applications, using Limes/Scowl running on Linux P5 based platform. Our approach is presented in Figure 1.

In this approach, a file with a kind of reduced address trace (each record represents aggregate data for multiple memory accesses) is generated, and then, parameters are computed using data from that file; some other parameters are dynamically computed during the same experiment. The organization of data in the file is convenient for obtaining wide range of parameters. The simulation of a multiprocessor system is provided by Limes. An application from SPLASH-2 suite is linked together with Limes and Scowl to produce a single executable file. Limes supplies Scowl with shared memory operations (read, write, lock, unlock, barrier). Scowl represents the PRAM shared memory simulator capable of collecting statistics of this operations. The results are stored in reduced trace file or used for dynamic computation of some parameters. The use of the trace file is convenient because it can be analyzed using simple programs, without knowing details of Limes and Scowl, in order to determine other parameters that were not analyzed in this paper. This method is also less time consuming than dynamic simulations.

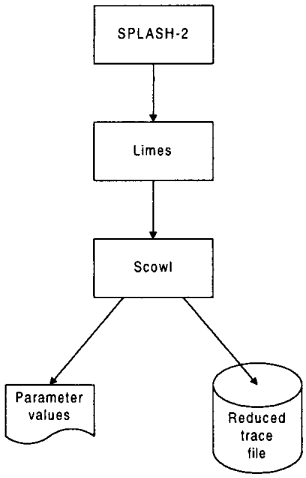

Figure 1. The structure and data-flow of designed measurement system.

3. Specification of parameters

The terms we will use in the rest of this paper include:

- *Epoch* is a sequence of instructions executed by a processor between two successive barriers.
- *Interval* is a sequence of instructions executed by a processor between two synchronization points. There are two types of intervals:
  - *CSI* (critical section interval) is a sequence of instructions executed by a processor between LOCK and UNLOCK primitives,
  - *NCSI* (non-critical section interval) is a sequence of instructions executed by a processor outside of a critical section (between BARRIER and LOCK primitives, UNLOCK and LOCK primitives, and UNLOCK and BARRIER primitives). There are no nested critical sections in SPLASH-2 programs.
- *Segment* is a contiguous part of shared address space allocated statically by the compiler or dynamically by the programmer using G_MALLOC (ANL macro for allocation of shared data).
- *Subsegment* is a contiguous part of a segment accessed in an interval.

Table 1 Scalar parameters group

par1.1	Total number of used locks on all processors
par1.2	Total number of used barriers on all processors
par1.3	Average number of processors that access the same lock
par1.4	Probability that given processor requires the same lock that it has just released
par1.5	Probability that given lock is acquired by the same processor that last released this lock
par1.6	Probability that given epoch has no critical sections
par1.7	Average number of CSI in epochs with at least one CSI
par1.8	Probability of CSI with no shared data writes
par1.9	Probability of NCSI with no shared data writes
par1.10	Average number of cycles between two consecutive requires for the same lock

In this work we have measured a set of parameters that we divide into several semantic groups. The first group of parameters (Table 1) contains some general behavioral scalar indicators such as number of initialized locks/barriers on all processors, probability that given processor requires the same lock that it has just released, etc. The common characteristic of parameters in this group is that each parameter represents only one value per application.

Table 2. Access-type-indexed array parameters group

par2.1	Total number of instructions executed by all processors
par2.2	Total number of instructions executed in all CSI on all processors
par2.3	Average number of instructions executed during one CSI
par2.4	Total number of instructions executed during all NCSI on all processors
par2.5	Average number of instructions executed during one NCSI
par2.6	Average number of instructions between two successive acquire operations for the same lock

The second group of parameters (Table 2) encompasses total and average numbers of executed instructions between particular synchronization points. The common characteristic of parameters in this group is that each parameter represents an array of separately measured values for each type of operation (no memory access, private/shared reads/writes of ordinary shared variables, and synchronizing primitives).

The third group of parameters (Table 3) is devoted to parameters considering physical coherence units of address space referred to as blocks (DSM pages or SMP cache lines). These parameters are measured for wide range of block sizes, and are given separately for CSI and NCSI intervals.

Table 3 Block-size-indexed array parameters group

par3.1	Average number of blocks modified inside one CSI/NCSI with shared data writes
par3.2	Maximum number of blocks modified inside one CSI/NCSI with shared data writes
par3.3	Minimum number of blocks modified inside one CSI/NCSI with shared data writes
par3.4	Average number of blocks only read inside one CSI/NCSI with one read only block at least
par3.5	Maximum number of blocks only read inside one CSI/NCSI with one read only block at least
par3.6	Minimum number of blocks only read inside one CSI/NCSI with one read only block at least
par3.7	Probability that CSI/NCSI contains one read only block at least

Except for par1.3 and par1.5 which have to be measured from the lock point of view, all the others were measured from the processor point of view, i.e. the parameter was measured on each processor, and then averaged per processor. For instance, par1.9 (probability of NCSI with no shared data writes) is figured out by counting total number of NCSIs on a processor, and the number of NCSIs with no shared data writes. This statistics is measured for each processor separately, and at the end the ratio representing probability is determined by dividing the two numbers. Finally, the parameter par1.9 is obtained as mean value of the ratios on all processors.

3.1. Conditions and assumptions

The measurements were performed under the following set of conditions:

- All the simulations are performed for a 32-processor system.
- The PRAM architectural model is simulated, as it best characterizes the inherent behavior of the workload.
- Applications operate on problem sizes that were suggested as suitable for an up to 64 multiprocessor system.
- The gcc-i486 (GNU C compiler for Intel architectures) is used for compiling the SPLASH-2 applications. All the parallel applications were compiled with -O2 switch for maximum optimization level.
- The resulting machine code is a mixture of both integer and floating point machine instructions.
- One machine word is 32 bits wide. Memory operands are either 1, 2, 4, 8 or 10 bytes wide, whereas the former three are counted as one-word operands, while 10-byte operands are treated as being three-word long.
- Any memory reference is counted as one regardless of the operand size.
- An operation that both reads and writes memory (such as incrementing a variable) is treated as a read instruction followed by a write instruction.
- An access is considered to be shared if it accesses memory within a shared segment. Shared segments are: (a) segment of static data, and (b) parts of heap allocated with G_MALLOC. These shared segments are not coherence units, but rather logical units allocated by programmer. Since code segment is not considered to be shared, some reads of constants from it are neglected.

In addition, the following assumptions are made:

- Every instruction is executed within a single cycle. That gives, on the average, the execution rate of one instruction per cycle.
- There are as many processors in the simulated system as there are threads in a parallel application; or rather, there are as many available processors in the simulated systems as there are threads in an application.
- Threads are pinned to its processors; i.e. no migration is modeled.
- No process other than the parallel application exists in the simulated system, and the activities of the operating system (i.e. its kernel) during the execution are negligible.

4. Limes and Scowl

Limes is a software tool for simulation of multi-processors [3], running on a P5-based PC, under the Linux operating system. Limes performs execution-driven simulation. The parallel application and the simulation kernel are linked into a single executable. The host CPU is executing machine instructions of the workload, as long as they do not attempt to access memory. The moment it happens, the execution of the native workload instructions is stopped, and the control is transferred to the simulation kernel, passing all the relevant parameters: address, data size, type of operation, and the operation time-stamp. Simulation kernel calls are inserted through the process called code instrumentation. The C or C++ code of a parallel application is converted to assembler by the compiler. The resulting assembly code is then instrumented for intercepting memory references and counting the time. In the instrumentation process, a kernel

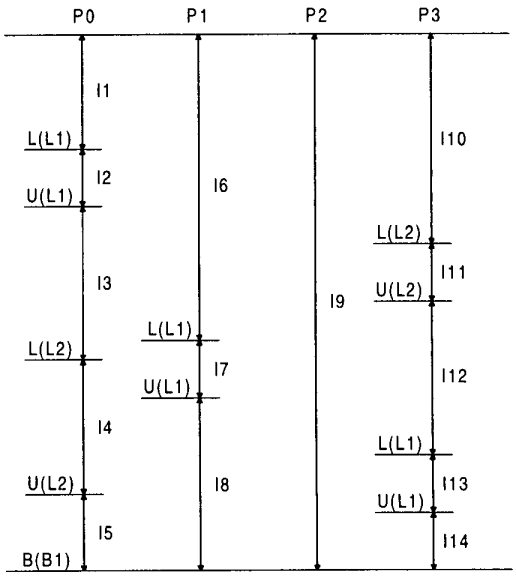

Figure 2. An example of an epoch executed on 4 processors. Legend. P-processor, Li-lock identifier, Bi-barrier identifier, L()-lock acquire, U()-lock release, B()-barrier operation, Ii-interval identifier.

call-out is inserted before each instruction that accesses shared memory.

Scowl is implemented as an extension of Limes. Basically, Limes provides for the hardware simulations, so in order to facilitate a mostly software based study it was necessary to change the simulation kernel. It was achieved by introducing some new simulation system calls, which allow efficient control of synchronization primitives. Two different simulators were written in building the tool capable of measuring specified groups of parameters. One

Table 4. Reduced trace file

Reduced trace file for an execution shown in Figure 2.	*Definition of accesses.*

`L(L1) P0: accesses(I1)`
`U(L1) P0: accesses(I2)`
`L(L2) P3: accesses(I10)`
`U(L2) P3: accesses(I11)`
`L(L1) P1: accesses(I6)`
`L(L2) P0: accesses(I3)`
`U(L1) P1: accesses(I7)`
`L(L1) P3: accesses(I12)`
`U(L2) P0: accesses(I4)`
`U(L1) P3: accesses(I13)`
`B(B1) P0: accesses(I5)`
`B(B1) P1: accesses(I8)`
`B(B1) P2: accesses(I9)`
`B(B1) P3: accesses(I14)`
	`accesses(In)` ::= W: *list of segments* R: *list of segments*
	list of segments ::= *{segment}*
	segment ::= Si: *list of subsegments*
	list of subsegments ::= *subsegment {subsegment}*
	subsegment ::= `<sa,nosab>`
	Legend:
	In - interval identifier,
	W - write,
	R - read,
	Si - segment identifier,
	sa - starting address,
	nosab - number of successive accessed blocks.

Table 5. The first group of parameters for all applications

Code	Problem size	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	1.10
Barnes	4K particles	87	6	16.05	65.62%	47.91%	47.06%	62.21	3.47%	8.33%	172400
Cholesky	tk15.O	67	1	14.48	26.50%	69.92%	50.00%	890.12	0.08%	11.14%	112246
FFT	64K points	1	1	32.00	0.00%	0.00%	85.71%	1.00	0.00%	65.63%	3
FMM	4K particles	347	1	4.58	38.62%	56.51%	38.24%	21.36	0.41%	35.52%	2904818
LU	512x512 matrix	1	1	32.00	0.00%	0.00%	98.51%	1.00	0.00%	46.20%	795438
Ocean	258x258 ocean	4	20	32.00	92.75%	0.00%	76.89%	1.00	90.05%	47.64%	16051
Radix	256K integers	33	2	32.00	0.00%	0.00%	62.50%	21.67	0.00%	90.41%	9139
Watersq	512 molecules	518	3	17.08	0.54%	0.00%	29.41%	92.08	0.00%	98.83%	239306
Watersp	512 molecules	6	3	26.67	33.28%	16.80%	23.53%	1.46	0.31%	64.56%	3360132

simulator is used for parameters of groups 1 and 2 which can be measured in execution-driven simulation. The other simulator serves as generator of reduced trace files needed for figuring out parameters of group 3. Certain post-simulation analysis using specially written program is needed to really compute them. The first simulator works with both private and shared references and it has its own structure describing which parts of memory are shared. Except from structure for shared data, the simulator manages information about a critical section passes on each processor and critical section passes of each lock, which are at the end used to find out the average values.

The second simulator deals only with shared data. Since there are still too many memory references it is not intended to calculate the end parameters, but instead it is used to produce the reduced trace files (a kind of intermediate form). Output from this simulator is used as input to the next pass which calculates requested parameters. Currently, only one post-simulation analyzer exists for calculating parameters of group 3. However, the reduced traces can be used as input to another post-simulation analyzers.

Figure 2 shows an execution of application on processors P0 to P3 between two barriers. The example shows total of 14 intervals: 5 CSIs (intervals I2, I4, I7, I11, and I13), and 9 NCSIs (I1, I3, I5, I6, I8, I9, I10, I12, and I14). Reduced trace file produced by this execution is shown in Table 4.

Table 4 shows a part of reduced trace file corresponding to the execution shown in Figure 2. Global order of executed synchronization operations is always preserved in the file, i.e. for any execution like the one shown in Figure 2, the file will have the same structure. R/W accesses performed during particular interval are listed according to the structure of accesses(In). It is also possible to generate lists not for each interval, but one list for all NCSIs in an epoch. Such feature of simulator is to be used for some kind of analysis which do not need partial data by intervals.

5. Results

Applications used as workload in our analysis are listed in column Code of Table 5. As mentioned before, a 32-processor system employing the PRAM memory model was simulated. Problem sizes used (given for each application in Table 5) are "default problem sizes for up to 64 processor machines" as defined in README files of SPLASH-2 suite, except in two cases, namely Barnes and FMM, where we had to decrease problem size from 16K particles to 4K particles due to limited resources of our simulations. The other parameters which can be changed for some applications were left at their default values, again bearing in mind that our intention was to characterize SPLASH-2 suite "as is". This implies executing FFT and LU (which are aware of cache) with

Table 6. Parameters of group 2 for Barnes

	na	rp	wp	rs	ws	la,lr	b
par2.1	24148652	87626723	72806593	103861978	45688170	17916	544
par2.2	1552861	473515	319922	359641	294367	0	0
par2.3	86	25	17	19	15	0	0
par2.4	239933391	87153208	72486671	103502337	45393803	17916	544
par2.5	13151	4780	3975	5669	2486	0	0
par2.6	232474	83910	69279	98495	43147	28	0

Table 7. Parameters of group 3 for Barnes

block size [B]	CSI						NCSI					
	write			read only			write			read only		
	avg	max	min	avg	max	min	avg	max	min	avg	max	min
4	14.97	204.84	1.00	7.98	79.00	1.00	32.42	5049.16	2.06	107.63	9640.91	1.03
8	9.74	115.56	1.00	6.34	50.38	1.00	18.28	2789.91	1.53	66.48	5307.88	1.03
16	7.03	71.19	1.00	5.12	30.41	1.00	11.20	1658.81	1.25	45.61	3144.66	1.03
32	5.42	48.28	1.00	4.14	18.22	1.00	7.06	958.78	1.12	33.96	2066.47	1.03
128	3.66	24.94	1.00	2.80	8.88	1.00	3.32	339.53	1.00	18.52	915.41	1.00
512	3.25	19.66	1.00	2.21	5.31	1.00	2.20	168.88	1.00	12.05	405.16	1.00
1024	3.18	18.25	1.00	2.10	4.25	1.00	1.95	129.03	1.00	10.71	251.28	1.00
2048	3.13	17.91	1.00	2.08	4.00	1.00	1.73	94.91	1.00	9.82	148.97	1.00
4096	3.08	17.31	1.00	2.08	3.62	1.00	1.51	62.97	1.00	9.27	92.91	1.00

cache of 64K lines of 16B, and with total cache size of 16KB, respectively.

The first group of parameters is shown in Table 5. It can be divided in several subgroups: par1.1 and par1.2 are basic indicators (number of synchronization variables); par1.3, par1.4, and par 1.5 present the relations of processors and locks (critical sections); par1.6 and par1.7 present the relations of epochs (barriers) and critical sections (locks); par1.8 and par1.9 give information about shared data writes and intervals (of both type); and par1.10 is an advanced indicator of behavior of parallel applications.

Parameters in the second group have one value for each type of operation: na (instructions which do not access memory), rp (reads of private data), wp (writes to private data), rs (reads of shared data), ws (writes to shared data), la,lr (lock acquire (LOCK), and lock release (UNLOCK) operations which execute the same number of times due to the way SPLASH-2 programs were written), and b (barrier operations, BARRIER). A memory reference is treated as private or shared (only semantically shared data is treated as shared – for example, it excludes the data allocated on heap with malloc and not G_MALLOC). All accesses to memory are counted as 1, no matter of the actual size of the data being accessed (mostly 4B (long words), relatively often 8B (double long words) in computation, and rarely 1B, 2B, or 10B). An example of results for this group of parameters for application Barnes in presented in Table 6.

The third group of parameters for an example application Barns is shown in Table 7. Parameters par3.1 to par3.6 are named in a semantic fashion (average(avg) /maximum(max)/minimum(min) write/read only). This group of parameters is dependent on the layout of memory being allocated from the heap. Presented results were obtained for the layout generated by calling the allocation function malloc, provided by the compiler environment (GCC 2.6.3.).

We have also measured the same parameters for G_MALLOC which allocates each segment on the block boundary, but the results have turned out to be similar; actually, the difference has been less than 1%. It should be noted that simulation kernel and the simulator itself allocate space at separated part of heap, so there is no interference of data allocated by the application and the simulation environment. By comparing the value of par2.3/par2.5 (average number of instructions executed during one CSI/NCSI) for shared writes and the value of par3.1 (average number of blocks modified inside one CSI/NCSI) for block size of 4B (one long word), one can get an approximation concerning multiple writes to the same location within one CSI/NCSI. The measure is not accurate since par2.3/par2.5 is not aware of the actual size of the accessed data (it merely counts the accesses), whereby par3.1 deals with actual size. Another conclusion may be drawn from tables representing group 3. It is that ratio of par3.1 for the different block sizes compared to the ratio of the block sizes can represent a measure of the spatial locality of shared data written within a CSI/NCSI. Similarly, ratio of par3.4 for the different block sizes gives the presentation of spatial locality of data which is only read.

6. Conclusion

In this paper we have presented an originally developed measuring tool Scowl which cooperates with Limes, a previously developed execution-driven simulator of multiprocessors. Parameters that we measured were classified into semantic groups. We also generated reduced traces that can ease the analysis of application behavior. Presented parameters obtained using Scowl and Limes are to assist people who devise new methods aimed to improvement of SMP/DSM systems and people who use analytical approach for quantitative evaluation and comparison of existing systems, especially in the area of the memory consistency models. Reduced traces can be used to obtain values for user-defined parameters with less programming efforts and execution time. Our future plans include the development of various analytical models for evaluation of algorithms and architectures in the field of

DSM/SMP systems, using obtained parameter values. We also plan to reevaluate memory consistency models and their implementations according to the results obtained in this study.

7. References

[1] Herrod, S.A., "TangoLite: Introduction and User's Guide," *Technical report*, Stanford University, Palo Alto, California, USA, November 1993.

[2] Kessler, R. E., Livny, M., "An Analysis of Distributed Shared Memory Algorithms," *Proceedings of the 9th International Conference on Distributed Computing Systems*, June 1989, pp. 498-505.

[3] Magdic, D., "Limes: A Multiprocessor Simulation Environment," *TCCA Newsletters*, March 1997, pp 68-71.

[4] Milutinovic, V., Milenkovic, A., "Cache Injection Control Architecture," *Proceedings of the 5th MASCOTS*, January 1997, Haifa, Israel.

[5] Protic, J., Tomasevic, M., Milutinovic, V., "Distributed Shared Memory: Concepts and Systems," *IEEE Parallel and Distributed Technology*, Vol. 5, No. 1, Summer 1996.

[6] Protic, J., Milutinovic, V., "Entry Consistency versus Lazy Release Consistency in DSM Systems: Analytical Comparison and a New Hybrid Solution," *Proceedings of the 5th IEEE Workshop on Future Trends of Distributed Computing Systems FTDCS'97*, October 1997, Tunis, Tunisia.

[7] Singh, J. P., Weber, W. D., Gupta,A., "SPLASH: Stanford parallel applications for shared-memory," *Computer Architecture News*, Vol. 20, No. 1, March 1992, pp. 5-44.

[8] Tartalja, I., Milutinović, V., "An approach to dynamic software cache consistency maintenance based on conditional invalidation," Proceedings of the 25th Annual Hawaii International Conference on System Sciences, IEEE Computer Society Press, Los Alamitos, California, January 1992, pp. 457-466.

[9] Woo S. C., Ohara M., Torrie E., Singh J. P., Gupta A., "The SPLASH-2 Programs: Characterization and Methodological Considerations," Proceedings of the 22nd Annual International Symposium on Computer Architecture, pp. 24-36, June 1995.