Performance of MP3D on the SB-PRAM Prototype

Roman Dementiev, Michael Klein, and Wolfgang J. Paul

Saarland University
Computer Science Department
D-66123 Saarbrücken, Germany
{rd,ogrim,wjp}@cs.uni-sb.de

Abstract. The SB-PRAM is a shared memory machine which hides latency by simple interleaved context switching and which can be expected to behave almost exactly like a PRAM if all threads can be kept busy. We report measured run times of various versions of the MP3D benchmark on the completed hardware of a 64 processor SB-PRAM. The main findings of these experiments are: 1) parallel efficiency is 79% for 32 processors and 56% for 64 processors. 2) Parallel efficiency is limited by the number of available threads.

1 Introduction and Previous Work

The SPLASH benchmark suite was designed in 1991 to test the performance of shared memory parallel machines[9]. MP3D is a program from this benchmark suite simulating the laminar flow of 40 000 particles in a wind tunnel around a space vehicle. As particles only interact by collisions a parallelization by spatial decomposition would be very efficient even on distributed memory machines. But MP3D partitions work among processors quite differently: *every particle is always processed by the same processor*.

On their way through the wind tunnel particles mix in an irregular way leading to non local communication patterns between processors. Most cache based machines react to this situation with thrashing. The DASH machine [8] is typical for this behaviour. Parallel efficiency is 23% for 16 processors and 16% for 32 processors.[1]. However, using a cache protocol optimised for 'migratory sharing' of data the MIT Alewife machine [4] achieves for optimized versions of MP3D an efficiency of up to 84% for 16 processors and 68% for 32 processors.

Frequent irregular accesses to shared memory do not inherently slow down parallel machines provided network congestion and memory module congestion can be avoided (e.g by address hashing and combining) and the latency of the network can be hidden (e.g. by multithreading). Various combinations of address hashing, combining and multithreading are used in machines like the NYU ultracomputer [7,11], the Fluent Machine [12], Cray T3D [1], TERA [5] and the SB-PRAM [2]. The latter machine is a reengineered version of the Fluent machine using address hashing, combining at the memory modules and in the network nodes as well as simple interleaved contex switching. With p processors it uses a certain number of threads $th(p)$ per processor in order to hide latency. Simulations suggest that for $th(p) \geq 3 \cdot \log p$ such a machine with cycle time τ

[1] MP3D is not present in the SPLASH 2 benchmark suite [16]

B. Monien and R. Feldmann (Eds.): Euro-Par 2002, LNCS 2400, pp. 132–136.

will look to the user almost exactly like a priority CRCW-PRAM with $p \cdot th(p)$ processors and cycle time $th(p) \cdot \tau$ processing one instruction per cycle *provided all threads can be kept busy*. [3]. The results reported here will use $p \leq 64$ processors, each with $th(p) = 32$ threads. Simulations of a 16 processor SB-PRAM predict for an optimized version of MP3D an efficiency of 72% [6]. In this note we present *measured* run times of optimized versions of MP3D on the completed prototype of a 64 processor SB-PRAM. On such a machine one tries to keep up to $32 \cdot 64 = 2048$ threads busy. The instruction set of the SB-PRAM includes multiprefix instructions. For details about the prototype see [10].

In section 2 we show that for such large numbers of threads the handling of collisions between particles in MP3D becomes a bottleneck. With 32 processors the efficiency drops to 59% and with 64 processors even to 27%, i.e. the machine is even slightly slower than with 32 processors. Parallelizing the handling of collisions brings the efficiency back up to 79% for 32 processors and 56% for 64 processors. Increasing the number of particles to 80 000 increases the efficiency to 71% for 64 processors. In section 3 we develop and validate a simple model of the run time based on threads and available tasks. The model shows that the less than perfect speedup figures are due to a shortage of tasks. We draw some conclusions in section 4.

2 Optimizing MP3D on the SB-PRAM

2.1 Parallel Prefix Operations and Parallel Task Arrays

If locality of memory access is not an issue and multiprefix instructions are supported by a combining network, then the method of choice for the parallelization and load balancing of many scientific codes is to make use of fine grained parallel task queues [15,14] or even more simply by two-dimensional *task arrays* where each row contains the data for one task. Parallel reading from or writing to different rows of a task array is easily realized with the help of multprefix adds.

Each time step of MP3D can essentially be divided into 5 phases i, each with a number of tasks N_i in the following way :

1. 'Initialize data structures'. One task initializes for one cell the data structure handling collissions of particles in the cell. The number N_1 of tasks equals the number of cells, i.e. $N_1 = 840$.
2. 'move and collide particle'. One task updates for 1 particle the position. In case of collisions it also updates direction and velocity. The number of tasks N_2 equals the number of particles, i.e. $N_2 = 40000$. This phase consumes sequentially 95% of the time.
3. 'add to free stream'. One task is to insert a new particle from the reservoirs into the tunnel. The average number of tasks is $N_3 = 30$.
4. 'move reservoir particle'. One task moves one particle in the reservoir. The number of tasks equals the number N_4 of particles in the reservoir. The reservoir contains an extra 2% of particles, i.e. $N_4 = N_2 \cdot 2/100$.
5. 'collide particles in reservoir'. A single task collides a pair of particles in the reservoir. The number of tasks N_5 equals $N_4/2$.

Note that N_4 and N_5 depend on N_2.

Partitioning and parallelizing work in this way [13] leads to the simulated figures reported in [6] for up to 16 processors. Running the program on the real hardware instead of the simulator reproduced the predicted run times and speedups to within 1 %. Running the same program on 32 or 64 processors leads to the speed ups shown in the left column of Table 1. One sees that doubling the number of processors from 16 to 32 only increases the speed up by only 46%. Doubling the number of processors again from 32 to 64 processors even leads to slightly deteriorated performance.

Table 1. Measured speedups of MP3D implementations for SB-PRAM

procs.	old (40000 part.)	new (40000 part.)	new (80000 part.)
1	1.00	1.00	1.00
2	1.90	1.94	1.91
4	3.72	3.84	3.81
8	7.19	7.53	7.56
16	12.88	14.40	14.78
32	19.16	25.28	27.84
64	18.69	35.97	45.69

2.2 Parallelizing the Handling of Collisions

The deterioration of performance comes from the fact that in MP3D the processing of collisions of particles serializes for each cell. Collisions are handled in the following way: A processor processing particle u in cell c checks if there is a particle waiting in the cell. If no particle is waiting, the processor leaves particle u waiting in the cell and quits. Otherwise there is exactly one particle v waiting in the cell. The processor collides the pair of particles u and v. None of them is waiting any more. Locks are used in order to guarantee that at most 1 processor at a time is accessing the cell. As long as the number of cells is large compared with the number of threads this is not a problem, but for $p = 32$ processors the number $32 \cdot p = 1028$ of threads already exceeds the number of cells. Adding more processors does not speed up this part of the computation which happens to dominate the run time.

Fortunately the handling of collisions during one time step can be parallelized for each cell. The problem to pair the particles in the cell is easily solved: Each cell gets a task array with enough lines for all particles which can be in the cell simultaneously (200 lines suffice) and a pointer q indicating that already q processors have accessed the cell. Lines in the array are numbered 0,1,2,.... Pointer q is initialized with 0. Suppose processors $i_1, \ldots, i_k$ are simultaneously accessing the cell with particles $u_1, \ldots, u_k$. With a multiprefix add each processor i_j determines $q(j) = q + j - 1$. If $q(j)$ is even processor i_j leaves particle u_j in row $q(j)/2$ waiting and quits. If $q(j)$ is odd processor i_j waits until a particle y is waiting in row $\lfloor q(j)/2 \rfloor$ and then collides particles u_j and y.

With the handling of collisions parallelized in this way one gets the speed ups in the middle column of Table 1. Parallel efficiency is back up to 78% for 32 processors and 56% for 64 processors. For 64 and 32 processors this is the best speed up for MP3D

known to us. Increasing the number of particles to 80 000 improves things further as can be seen in the right column of Table 1. Efficiency is now 87% for 32 processors and 71% for 64 processors.

3 Speedup as a Function of the Number of Threads

The less than perfect speed up figures can be explained with a simple run time model. Let $t_i(N_i, th)$ be the run time of phase i with N_i tasks and th threads. Measurements show that with the constants C_i and T_i from Table 2 the formula $t_i(N_i, th) = C_i + T_i \cdot \lceil N_i/th \rceil$ models the run times of the phases with an error of at most 4% for the computation in for phases 2 to 5 and at most 7.6% for phase 1.

Table 2. Model parameters for different stages

i	phase	C_i (cycles)	T_i (cycles)	$N_i(40000)$	$N_i(80000)$
1	reset	296237	159203	840	840
2	move	1457627	306214	40000	80000
3	add to free stream	511190	69054	30	30
4	reservoir move	304000	28800	800	1600
5	reservoir collide	236406	701946	400	800

In all phases $i \neq 2$, i.e. except phase 'move', we have $\lceil N_i(40000)/1024 \rceil = \lceil N_i(40000)/2048 \rceil = 1$. Hence these phases do not become faster as we double the number of processors from 32 to 64. For N_2 particles and th threads the model predicts a run time of $t(N_2, th) = \sum_{i=1}^{5} t_i(N_i(N_2), th)$. In particular $t(N_2, 1)$ gives the run time of a single thread running on a single processor Therefore with respect to threads the parallel efficiency is $eff(N_1, th) = t(N_1, 1)/(th \cdot t(N_1, th))$. The efficiency for p processors is obtained by $eff(N_1, 32 \cdot p)$ which matches the measured values up to 7.3%. [2]

4 Conclusions

MP3D is a benchmark with highly irregular and non local access patterns to the memory. We have optimized this benchmark for the SB-PRAM, an architecture designed to deal effortlessly with irregular access patterns. We have measured parallel efficiencies of 79% for 32 processors and 56 % for 64 processors. We have shown that on the SB-PRAM architecture the efficiency is bounded by the number of available threads. Increasing the number of particles from 40 000 to 80 000 increases the efficiency to 71% for 64 processors.

[2] The 5 phases do not cover the entire work done in a computation step

References

1. Cray Research, Inc. Cray T3D System Architecture Overview, March 1993.
2. F. Abolhassan, R. Drefenstedt, J. Keller, W. J. Paul, and D. Scheerer. On the physical design of PRAMs. *The Computer Journal*, 36(8):756–762, 1993.
3. F. Abolhassan, J. Keller, and W. J. Paul. On the cost-effectiveness of PRAMs. *Acta Informatica*, 36(6):463–487, 1999.
4. A. Agarwal, R. Bianchini, D. Chaiken, K. Johnson, D. Kranz, J. Kubiatowicz, B.-H. Lim, K. Mackenzie, and D. Yeung. The MIT alewife machine: Architecture and performance. In *Proc. of the 22nd Annual Int'l Symp. on Computer Architecture (ISCA'95)*, pages 2–13, 1995.
5. R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. The Tera computer system. In *Proc. of the 1990 International Conference on Supercomputing*, pages 1–6, 1990.
6. A. Formella, T. Grun, J. Keller, W. Paul, T. Rauber, and G. Runger. Scientific applications on the SB-PRAM. In *Proc. of International Conference on Multi-Scale Phenomena and Their Simulation, pages 272–281. World Scientific, Singapore*, 1997.
7. A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir. The NYU ultracomputer - designing an MIMD shared memory parallel computer. *IEEE Transactions on Computers*, 32(2):175–189, 1983.
8. D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta, and J. Hennessy. The DASH prototype: Logic overhead and performance. In *IEEE Transactions on Parallel and Distributed Systems, 4(1):41–61*, 1993.
9. J. Pal Singh, W. Weber, and A. Gupta. SPLASH: Stanford Parallel Applications for Shared-Memory. Technical Report CSL-TR-91-469, Stanford University, 1991.
10. W. J. Paul, P. Bach, M. Bosch, J. Fischer, C. Lichtenau, and J. Roehrig. Real PRAM Programming. In *Proc. of the Europar'02*, 2002.
11. G. Pfister, W. Brantley, D. George, S. Harvey, W. Kleinfelder, K. McAuliffe, E. Melton, V. Norton, and J. Weiss. The IBM research parallel processor prototype. In *Proc. Int.l Conf. on Parallel Processing 764–771*, 1985.
12. A. G. Ranade. The Fluent Abstract Machine. Technical Report TR-12 BA87-3, 1987.
13. T. Rauber, G. Runger, and C. Scholtes. Shared-memory implementation of an irregular particle simulation method. In *Proc. of the EuroPar'96, number 1123 in Springer LNCS.*, pages 822–827, 1996.
14. J. Roehrig. Implementierung der P4-Laufzeitbibliothek auf der SB-PRAM. Master's thesis, Universitaet des Saarlandes, Saarbruecken, 1996.
15. J. Wilson. *Operating System Data Structures for Shared-Memory MIMD Machines with Fetch-and-Add.* PhD thesis, 1988.
16. S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proc. of the 22nd International Symposium on Computer Architecture, pages 24–38*, 1995.