Thread-level Speculative Parallelization

Diego R. Llanos[*]
Computer Science Department, University of Valladolid, Spain.
E-mail: diego@infor.uva.es.

The basic idea under speculative parallelization (also called *thread-level speculation*) [2, 6, 7] is to assign the execution of different blocks of consecutive iterations to different threads, running each one on its own processor. While execution proceeds, a software monitor ensures that no thread consumes an incorrect version of a value that should be calculated by a predecessor, therefore violating sequential semantics. If such a *dependence violation* occur, the monitor stops the parallel execution of the offending threads, discards iterations incorrectly calculated, and restart their execution using the correct values. Figure 1 shows an example of speculative parallel execution of a loop with dependences.

The detection of dependence violations can be done either by hardware or software. Hardware solutions [4, 5] rely on additional hardware modules to detect dependences, while software methods [2, 6, 7] augment the original loop with new instructions that check for violations during the parallel execution.

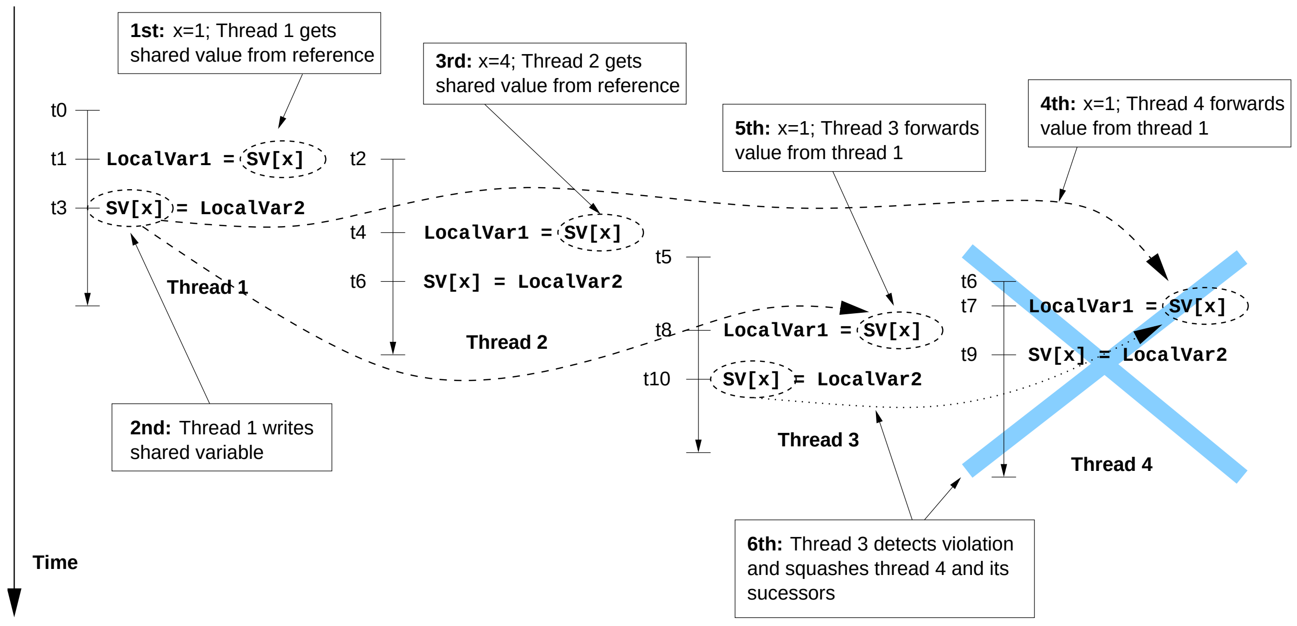
The author's visits to EPCC thanks to the TRACS and HPC-Europa programmes led to a successful collaboration with Dr. Marcelo Cintra, of the Division of Informatics, in the field of speculative parallelization. We have developed a new software-only speculative parallelization engine to automatically execute in parallel sequential loops with few or no dependences among iterations [1, 2, 3]. The main advantage of this solution is that it makes possible to parallelize an iterative application automatically by a compiler, thus obtaining speedups in a parallel machine without the cost of a manual parallelization. To do so, the compiler augments the original code with function calls to perform accesses to the structure shared among threads, and to monitor the parallel execution of the loop. The next section discusses the mechanism in more detail.

Handling data dependences

From the parallel execution point of view, in each iteration two different classes of variables can appear. Informally speaking, *private* variables will be those that are always written in each iteration before being used. On the other hand, values stored in *shared* variables are used among different iterations. It is easy to see that if all variables are private, then no dependences can arise and the loop can be executed in parallel. Shared variables may lead to dependence violations only if a value is written in a given iteration and a successor has consumed an outdated value. This is known as the Read-after-Write (RAW) dependence. In this case, the latter iteration and all its successors should be re-executed using the correct values. This is known as a *squash* operation.

To simplify squashes, threads that execute each iteration do not change directly the shared structure: instead, each thread maintains a *version* of the structure. Only if the execution of the iteration succeeds, changes are reflected to the original shared structure, through a *commit* operation. This operation should be done in order for each block of iterations, from the non-speculative thread (that is, the one executing the earliest block) to the most-speculative one. If the execution of the iteration fails, version data is discarded.

[*]Appeared in Alberigo, Erbacci and Garofalo (eds.) *Science and Supercomputing in Europe*, pp. 211–213, CINECA, Italy, 2005, ISBN 88-86037-15-5.



1st: x=1; Thread 1 gets shared value from reference
3rd: x=4; Thread 2 gets shared value from reference
5th: x=1; Thread 3 forwards value from thread 1
4th: x=1; Thread 4 forwards value from thread 1
t0
t1
t3
LocalVar1 = SV[x]
SV[x] = LocalVar2
Thread 1
t2
t4
t6
LocalVar1 = SV[x]
SV[x] = LocalVar2
Thread 2
t5
t8
t10
LocalVar1 = SV[x]
SV[x] = LocalVar2
Thread 3
t6
t7
t9
LocalVar1 = SV[x]
SV[x] = LocalVar2
Thread 4
2nd: Thread 1 writes shared variable
6th: Thread 3 detects violation and squashes thread 4 and its sucessors
Time

Figure 1: Speculative parallelization.

Results and future work

The experiments performed to measure the execution time of both sequential and parallel versions of the applications considered were done on the EPCC's Sun Fire 15K symmetric multiprocessor (SMP). Our results so far [1, 2, 3] show noticeable speedups in all the applications considered. This work can be downloaded from the author's website (see the Biography section). Our future work include the development of new scheduling alternatives to handle hot spots in the distribution of dependences inside the loop.

Conclusions

Parallel implementations of incremental algorithms are hard to develop and require an in-depth understanding of the problem, the language, the compiler and the underlying computer architecture. In our work we have shown how we can use speculative parallelization techniques to execute automatically in parallel sequential applications with a negligible implementation cost.

Acknowledgments

The author would like to thank the Edinburgh Parallel Computing Center (EPCC) for the main computing resources used in this work and its support staff, in particular Mark Bull and Catherine Inglis.

Biography

Diego R. Llanos received his MS and PhD degrees in computer science from the University of Valladolid, Spain, in 1996 and 2000, respectively. He is a recipient of the Spanish government's national award for academic excellence. Dr. Llanos is an associate professor of computer architecture at the University of Valladolid, and his research interests include parallel and distributed computation, computer system performance evaluation and automatic parallelization of sequential code. He is a member of the IEEE Computer Society. More information about his current research activities can be found at `http://www.infor.uva.es/˜diego`.

References

[1] Marcelo Cintra and Diego R. Llanos. Design space exploration of a software speculative parallelization scheme. *To appear in IEEE Trans. on Paral. and Distr. Systems*.

[2] Marcelo Cintra and Diego R. Llanos. Toward efficient and robust software speculative parallelization on multiprocessors. In *Proceedings of the SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, June 2003.

[3] Marcelo Cintra, Diego R. Llanos, and Belén Palop. Speculative parallelization of a randomized incremental convex hull algorithm. In *Proceedings of the Computational Geometry and Applications (CGA)*, May 2004.

[4] Marcelo Cintra, Jos F. Martnez, and Josep Torrellas. Architectural support for scalable speculative parallelization in shared-memory multiprocessors. In *Proc. of the 27th intl. symp. on Computer architecture (ISCA)*, pages 256–264, June 2000.

[5] Lance Hammond, Mark Willey, and Kunle Olukotun. Data speculation support for a chip multiprocessor. In *Proc. of the 8th Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 58–69, 1998.

[6] M. Gupta and R. Nim. Techniques for run-time parallelization of loops. *Supercomputing*, November 1998.

[7] L. Rauchwerger and D. A. Padua. The LRPD test: Speculative run-time parallelization of loops with privatization and reduction parallelization. *IEEE Transactions on Parallel and Distributed Systems*, 10(2):160–180, 1999.