Scheduling Speculative Work in MUSE

(Extended Abstract)

Khayri A. M. Ali and Roland Karlsson

Swedish Institute of Computer Science, SICS
Box 1263, S-164 28 Kista, Sweden
Phone: +46 8 752 15 00
Fax: +46 8 751 72 30
{khayri,roland}@sics.se

The Muse OR-parallel Prolog system has been designed and implemented on a number of NUMA (Non-Uniform Memory Access) and UMA (Uniform Memory Access) computers; a VME system, Sequent Symmetry, TP881V from Tadpole Technology, Sun Galaxy, BBN Butterfly I, and BBN Butterfly II [1, 2, 3, 4, 5, 6]. It supports the full Prolog language and programs run on it with almost no user annotations. It is based on a simple extension of the state-of-the-art sequential Prolog implementation (WAM).

The Muse model assumes a number of extended WAMs (called *workers*), each with its own local address space, and some global space shared by all workers. The model requires copying parts of the WAM stacks when a worker runs out of work or suspends its current branch. The copying operation is made efficient by utilizing the stack organization of the WAM. Workers make a number of choicepoints sharable, and they get work from those shared choicepoints (nodes) by the normal backtracking of Prolog. A worker consists of two components: the *engine*, which performs the actual Prolog work, and the *scheduler*. The schedulers, working together, schedule the work between engines and support the sequential semantics of Prolog.

The first Muse engine has been produced by extending the SICStus Prolog version 0.6. Extensions are carefully added to preserve the high efficiency of SICStus leading to a negligible overhead which is significantly lower than in other OR-parallel models.

The new Muse scheduler [5] supports efficient scheduling of speculative work (i.e., work which may later be pruned) in comparison with the earlier scheduler [2]. The new scheduler has two different scheduling strategies: one for busy workers actively seeking the leftmost available work, and the other for idle workers looking for work. The idea of the first strategy is based on dividing the Prolog tree into two sections: the right section contains voluntarily suspended work and the left section contains active work. The available workers concentrate on the available work in the left section. When the amount of parallelism in the left section is not enough for the workers, some of the leftmost part of the right section will be de-suspended (made active). The other strategy is based on the principle that when a worker is idle, its next piece of work will be taken from the bottommost (i.e. youngest) node in the richest branch of a set of active branches. When the work at the youngest node is exhausted, that worker will find more work by backtracking to the next youngest node. If the idle worker cannot find work in the system, it will de-suspend the leftmost part of the suspended section of the tree.

In the meeting, we present work submitted for publication in the *International Journal of Parallel Programming* [5]. We describe and evaluate our strategy of better scheduling of speculative work used in the new version of the Muse scheduler. We compare also the results of the new Muse scheduler with the new Aurora schedulers specially designed for efficient scheduling of speculative work. The comparison shows that our strategy performs quite well.

Table 1 and Table 2 show timing and speedup results of the current Muse system on

Sequent Symmetry for two sets of benchmarks used for the evaluation of OR-parallel Prolog systems. The first set (Table 1) contains programs with lots of speculative work and is used for producing the first (leftmost) solution. The other set (Table 2) contains programs with little or no speculative work and is used for producing all solutions. All results presented in Table 1 and Table 2 are the mean values of 8 runs.

Goals executed	Muse workers 1	5	10	15	20	25
queen2	46.96	11.15 (4.21)	5.98 (7.85)	4.23 (11.1)	3.23 (14.5)	2.62 (17.9)
solitaire	38.78	9.13 (4.25)	4.76 (8.15)	3.31 (11.7)	2.62 (14.8)	2.22 (17.5)
queen1	16.37	4.04 (4.05)	2.24 (7.31)	1.64 (9.98)	1.31 (12.5)	1.10 (14.9)
tst22	5.82	1.42 (4.10)	0.86 (6.77)	0.65 (8.95)	0.56 (10.4)	0.51 (11.4)
protein	4.38	1.34 (3.27)	0.90 (4.87)	0.75 (5.84)	0.66 (6.64)	0.65 (6.74)
flypan	2.48	0.71 (3.49)	0.46 (5.39)	0.42 (5.90)	0.42 (5.90)	0.44 (5.64)
transform	9.98	5.10 (1.96)	4.28 (2.33)	3.88 (2.57)	3.85 (2.59)	4.00 (2.50)
warplan	4.42	2.16 (2.05)	2.02 (2.19)	2.04 (2.17)	1.93 (2.29)	1.97 (2.24)
Σ	129.19	35.04 (3.69)	21.51 (6.01)	16.92 (7.64)	14.58 (8.86)	13.51 (9.56)

Table 1: Runtimes (in seconds) and speedups (in parentheses) of the Muse system for the set of benchmarks containing speculative work.

Goals executed	Muse workers 1	5	10	15	20	25
8queens1	6.70	1.37 (4.89)	0.71 (9.44)	0.50 (13.4)	0.39 (17.2)	0.33 (20.3)
8queens2	17.05	3.44 (4.96)	1.76 (9.69)	1.19 (14.3)	0.92 (18.5)	0.76 (22.4)
tina	14.32	2.99 (4.79)	1.60 (8.95)	1.14 (12.6)	0.92 (15.6)	0.78 (18.4)
saltmustard	2.07	0.44 (4.70)	0.23 (9.00)	0.17 (12.2)	0.15 (13.8)	0.13 (15.9)
parse2*20	5.85	1.64 (3.57)	1.24 (4.72)	1.20 (4.88)	1.24 (4.72)	1.30 (4.50)
parse4*5	5.42	1.27 (4.27)	0.85 (6.38)	0.75 (7.23)	0.75 (7.23)	0.76 (7.13)
parse5	3.82	0.85 (4.49)	0.52 (7.35)	0.44 (8.68)	0.41 (9.32)	0.40 (9.55)
db4*10	2.40	0.58 (4.14)	0.38 (6.32)	0.32 (7.50)	0.30 (8.00)	0.30 (8.00)
db5*10	2.92	0.70 (4.17)	0.46 (6.35)	0.38 (7.68)	0.36 (8.11)	0.34 (8.59)
house*20	4.32	1.30 (3.32)	0.97 (4.45)	0.87 (4.97)	0.85 (5.08)	0.86 (5.02)
parse1*20	1.55	0.61 (2.54)	0.62 (2.50)	0.65 (2.38)	0.66 (2.35)	0.66 (2.35)
parse3*20	1.33	0.63 (2.11)	0.62 (2.15)	0.64 (2.08)	0.64 (2.08)	0.65 (2.05)
farmer*100	3.11	1.68 (1.85)	1.77 (1.76)	1.79 (1.74)	1.79 (1.74)	1.79 (1.74)
Σ	70.86	17.50 (4.05)	11.73 (6.04)	10.04 (7.06)	9.37 (7.56)	9.06 (7.82)

Table 2: Runtimes (in seconds) and speedups (in parentheses) of Muse for the set of benchmarks containing little (or no) speculative work.

The degree of performance improvement for each benchmark in Table 1 depends on where in the Prolog search tree the first solution is and on the frequency of moving workers from right to left. For instance, the first solution for the transform, and warplan benchmarks is far to the left in comparison with the other benchmarks.

The degree of performance improvement for each benchmark in Table 2 depends on the amount of parallelism. For instance, the amount of parallelism in the first four benchmarks in Table 2 is much larger than that in the other benchmarks.

References

[1] Khayri A. M. Ali and Roland Karlsson. The Muse Approach to Or-Parallel Prolog. *International Journal of Parallel Programming*, 19(2): 129–162, April 1990.

[2] Khayri A. M. Ali and Roland Karlsson. Full Prolog and Scheduling Or-Parallelism in Muse. *International Journal of Parallel Programming*, 19(6): 445–475, December 1990.

[3] Khayri A. M. Ali and Roland Karlsson. OR-parallel Speedups in a Knowledge Based System: on Muse and Aurora. In *the Proceedings of the International Conference on Fifth Generation Computer Systems*, June 1992.

[4] Khayri A. M. Ali, Roland Karlsson, and Shyam Mudambi. Performance of Muse on Switch-Based Multiprocessor Machines. *New Generation Computing*, 11(1,4): 81–103, 1992.

[5] Khayri A. M. Ali and Roland Karlsson. Scheduling Speculative Work in Muse and Performance Results. Submitted for publication in *the International Journal of Parallel Programming*, March 1993.

[6] Roland Karlsson. *A High Performance OR-parallel Prolog System.* PhD thesis, The Royal Institute of Technology, Stockholm, March 1992.