Full Prolog and Scheduling
Or-Parallelism in Muse

Khayri A. M. Ali\textsuperscript{1} and Roland Karlsson\textsuperscript{1}

Received December 1990; Revised May 1991

Muse is a simple and efficient approach to Or-parallel implementation of the full Prolog language. It is based on having multiple sequential Prolog engines, each with its local address space, and some shared memory space. It is currently implemented on a number of bus-based and switch-based multiprocessors. The sequential SICStus Prolog system has been adapted to Or-parallel implementation with very low extra overhead in comparison with other approaches. The Muse performance results are very encouraging in absolute and relative terms.

The Muse execution model and its performance results on two different multiprocessor machines for a parallel version of Prolog, named Commit Prolog, have been presented in previous papers. This paper discusses supporting the full Prolog language and describes mechanisms being developed for scheduling Or-parallelism in Muse. It also presents performance results of the Muse implementation on Sequent Symmetry after supporting full Prolog. The results show that the extra overhead associated with supporting the full Prolog language is negligible.

\textbf{KEY WORDS: } Or-parallelism; full Prolog; multiprocessors; experimental results; scheduling.

1. INTRODUCTION

A variety of approaches toward exploitation of parallelism in Prolog programs are under current investigation. Many of these deal with efficient implementation of Prolog on multiprocessor machines by exploiting either Or-parallelism\textsuperscript{1-8} or Independent And-parallelism\textsuperscript{9-11} or a combination of both\textsuperscript{12-15}. The Muse approach is one of those that exploit only Or-parallelism\textsuperscript{1}. Execution of a Prolog program forms a search tree.

\textsuperscript{1} Swedish Institute of Computer Science, SICS, Box 1263, S-164 28 Kista, Sweden.
Or-parallel execution of a Prolog program means exploring branches of a Prolog search tree in parallel. In the Muse approach (as in other Or-parallel Prolog approaches, e.g., Aurora\(^7\) and PEPSys\(^2\)), Or-parallelism in a Prolog search tree is explored by a number of workers (processes or processors). [In this paper we try to be consistent with the Aurora terminology.\(^7\)] This paper describes the basic mechanisms used for exploring branches of a Prolog search tree by the Muse workers. It also describes mechanisms for maintaining the sequential semantics of cut, findat and side-effect constructs.

The Muse approach is based on having several sequential Prolog engines, each with its local address space, and some shared memory space. It is currently implemented on a bus-based shared memory machine TP881V, from Tadpole Technology, with 4 (88100) processors, a bus-based machine with local/shared memory with 7 (68020) processors constructed at SICS, a bus-based shared memory S81, Sequent Symmetry, with 16 (i386) processors, and switch-based shared memory machines, BBN Butterfly I (GP1000) and II (TC2000), with 96 (68020) and 45 (88100) processors respectively. The sequential SICStus Prolog,\(^{16}\) a fast, portable system, has been adapted to Or-parallel implementation. The extra overhead associated with this adaptation is very low in comparison with the other approaches. It is around 3% for TP881V, and 5% for the constructed prototype and Sequent Symmetry. The performance results of Muse on the BBN Butterfly machines will be reported by Shyam Mudambi at Brandeis University who has ported the Muse system into the BBN Butterfly machines. Mudambi preliminary results are very promising.\(^{17}\) The Muse execution model and its performance results on the constructed prototype and Sequent Symmetry machines for a parallel version of Prolog, named Commit Prolog, have been presented in previous papers.\(^1,^{18}\) Commit Prolog is a Prolog language with cavalier commit\(^2\) instead of cut, asynchronous (parallel) side-effects and internal database predicates instead of the synchronous (sequential) counterparts, and sequential and parallel annotations. Cut and sequential side-effect semantics could be obtained on Commit Prolog by annotating Prolog programs according to some rules.\(^1\) In this paper, we discuss supporting the full Prolog language without such annotation. Some parts of this paper has been presented in Ref. 20.

The paper is organized as follows. Section 2 briefly describes the Muse execution model. This helps for understanding the principles for scheduling work in Muse. Section 3 discusses principles for scheduling work in Muse.

\(^2\)Cavalier commit prunes branches both to the left and right of the committing branch, and is not guaranteed to prevent side-effects from occurring on the pruned branches.\(^{19}\)