Performance of the Compiler-based Andorra-I System

(Extended Abstract)

Rong Yang, Tony Beaumont, Ines Dutra, Vitor Santos Costa and David H.D. Warren

Department of Computer Science
University of Bristol
Bristol BS8 1TR, U.K.
{rong,beaumont,ines,costa,warren}@acrc.bristol.ac.uk

Andorra-I is an experimental parallel Prolog system for shared-memory multiprocessors, based on the Basic Andorra model. This model supports both dependent and-parallelism, by executing determinate goals in parallel, and or-parallelism, stemming from the nondeterminate goals. As far as we know, Andorra-I is the first and only fully implemented system integrating or-parallelism and dependent and-parallelism.

We have previously presented the main implementation issues and early performance results of the first, interpreter-based, version of Andorra-I. Since then, a much faster, compiler-based, version of Andorra-I has been developed. The compiler-based version preserves the interpreter's data-structures, whilst including a full compiler and emulator. The decision to preserve the interpreter's data-structures was based on the good performance of the interpreted version.

In this talk, we present the new compiler-based version and discuss its performance. We study the system with a wide set of real-life, non-trivial logic programming applications. As Andorra-I provides a powerful programming model, we are able to include Prolog programs, committed-choice programs, and Andorra-style programs which combine both implicit coroutining and search.

The results show that the new compiler-based system performs well, and is very effective at exploiting both forms of parallelism. Basic performance compares well with other compiler-based systems such as Sicstus Prolog and Parlog JAM. Although Andorra-I supports a more powerful execution model than either Prolog or Parlog, and the new compiler-based system would benefit from further optimisation, the uniprocessor version is typically only 2.5 times slower than Sicstus (but ranging from 1.7 to 4.5 times slower), and about the same speed as JAM. The actual overheads in supporting the multiprocessor version are also reasonable, about 40% on average. This means that we can exploit the parallelism contained within a program very efficiently. For the programs which have either and- or or-parallelism alone, we can obtain speedups that are as good as JAM and Aurora. For the programs which contain both and- and or-parallelism, we can obtain speedups which are greater than those obtainable by exploiting just one type of parallelism alone.