Shared Memory System for Babel: a VHDL Specification*

F. Sáenz¹, W. Hans², J.J. Ruiz¹ and S. Winkler²

¹ Universidad Complutense de Madrid (UCM), Departamento de Informática y Automática, 28040 Madrid, Spain, {fernán.jruiz}@dia.ucm.es
² RWTH Aachen, Lehrstuhl für Informatik II, D-52056 Aachen, Germany, {hans,winkler}@zeus.informatik.rwth-aachen.de

In this work we present a shared memory abstract machine designed for the independent And-parallel execution of higher order innermost Babel [1] programs. We validate the abstract machine and study its performance with VHDL [3] tools.

Our approach is based on the exploitation of independent And-parallelism. Under this kind of parallelism, we allow the parallel execution of independent expressions on the right hand side of a Babel rule. In an innermost functional language, several expressions are said to be independent if no expression is a subexpression of any other expression. Since the Babel language embodies logical variables, the independence refers also to the sharing of common logical variables. This condition is imposed because the common variables may be bound to different values by the different functions.

The aim is to speed up the execution of Babel programs through the parallelization without altering the semantics. Therefore, we adjust the parallel execution scheme so that the sequential order of solutions is maintained, because a more liberal parallel execution order would yield an unpredictable behaviour. The forward computation reaches a fork point in the presence of a parallel call relating several expressions, consisting of the parallel evaluation of these expressions. The join point is reached when all the expressions to the left of the leftmost definitory expression³ are computed, if it exists, otherwise when all the (non-definitory) expressions are computed. The results of the expressions to the right of the definitory one are discarded. If a failure is computed for one of the expressions, the remaining evaluations to its right are discarded. If there is a definitory result to its left, the forward computation resumes. Otherwise, backward computation happens and one expression with pending alternatives is requested for further results in accordance with the sequential order. Eventually, the forward computation resumes. This scheme embodies semi-intelligent backtracking.

We propose an implementation of the above execution scheme, retaining the best features present in stack-based implementations [2] (e.g., the fast deallocation of memory during backtracking) despite the parallel extensions. Our

* This work was supported by the spanish PRONTIC project TIC92–0793–C02–01 and by the german DFG–grant IN 20/6–1.

³ For instance, the leftmost truth value true for a disjunction implies the result true for the disjunction. Hence we call such values definitory.
machine consists of several stack data structures so that topmost data frames are deallocated first during backtracking. In order to accomplish this criteria, we have imposed a precedence restriction on the data frames which can be pushed on top of the stacks. The forward as well as the backward computation are performed in parallel. The former due to the parallel scheduling of tasks, and the latter due to the parallel deallocation of data frames during backtracking.

The performance study of abstract machines is usually done by using the high level capabilities in the underlying system. These capabilities rely on a software layer (e.g., a concurrent operating system) which isolates the underlying system from the abstract machine. In this way, it is not possible to carry out the performance study of fine grain features present in the abstract machine which make use of low level capabilities in the underlying system. For instance, the features of the following different topics may prove interesting for future study: the interleaved memory model which assumes a segmented address space, the shared memory model which assumes a common memory and an access protocol policy, the distributed memory model which incorporates isolated memory pieces together with the message passing mechanism, and a cache memory system considering several coherence policies.

To deal with the functional validation of our parallel system as well as the latter topics, we have specified it with the hardware description language VHDL. This language provides a versatile set of description facilities for modelling systems from the behavioural level to the gate level. Moreover, it has a temporal model capable of the simulation of the fine grain features required to study the performance of our abstract machine. We have designed a VHDL model for the parallel system assuming a shared memory over a single bus and local memory to reduce the bus traffic. In this model we can specify several parameters of the system such as the memory access times (register, shared memory and local memory). The simulation provides the actual computation time of the parallel system, which can be compared with the computation time of the sequential system. Moreover, we can carry out the study of the factors which explain the way the parallel system behaves. For instance, we may evaluate the data bus contention, different cache memory policies, and others which we may analyze and adjust to try different alternatives leading to a better performance. We intend to compare our first design with other memory models and parallel system designs which will be derived from the first conclusions.

References
