Full Or-Parallelism and Restricted And-Parallelism in BTM

Zheng Yuhua (郑宇华), Xie Li (谢立) and Sun Zhongxiu (孙钟秀)

Department of Computer Science, Nanjing University, Nanjing 210008

Received May 21, 1992.

Abstract

BTM is a new And/Or parallel execution model for logic programs which exploits both full Or-parallelism and restricted And-parallelism. The advantages of high parallelism and low run time cost make BTJ, an experimental execution system of BTM implemented on a nonshared-memory multiprocessor system, achieve significant speedup for both And-parallel and Or-parallel logic programs.

Keywords: Full Or-parallelism, restricted And-parallelism, multi-transputer system, nonshared-memory; extra-predicates.

1 Introduction

Generally speaking, there are two kinds of parallelism that can be exploited in logic programs: Or-parallelism and And-parallelism.

*Or-parallelism:* two kinds of Or-parallelism.

(1) The parallel execution of all the multiple unified clauses that define one predicate is called the first type of Or-parallelism.

(2) Assume that: \( g(X,Y) : a(X), b(X,Y) \). If several solutions of \( a(X) \) exist, say \( a(x_1), a(x_2), \ldots, a(x_n) \), then we call the parallel execution of \( b(z_1,Y), \ldots, b(z_n,Y) \) the second type of Or-parallelism.

*And-parallelism:* the parallel execution of all the subgoals of one clause-body is called And-parallelism.

The control scheme of Or-parallel model is relatively simple compared with that of And-parallel model. It can generate large scale computation and can achieve high efficiency. But for And-parallel model, it is very complex in that it involves the parallel execution of subgoals which have shared variables. When two subgoals try to bind the same shared variable with different values, the binding conflict appears. Now several approaches have been proposed to solve this problem, including the consistency check method in full And-parallel models. But the runtime cost is very high. With the development of parallel execution model for logic programs, the "And/Or" model has been proposed recently in order to increase parallelism. It tries to exploit both And- and Or-parallelism in logic programs.

This paper also proposes a new "And/Or" parallel execution model BTM. It supports both And- and Or-parallel execution of logic programs. Based on BTM, an experimental parallel inference mechanism called BTJ is implemented. The case study result indicates that this new parallel execution model BTM can achieve significant speedup for both And- and Or-parallel execution of logic programs.
2 Parallel Execution Model BTM

2.1 Control of Parallelism

2.1.1 Control of And-Parallelism

In order to avoid the binding conflict and to lower the high communication cost among subgoals, we exploit only restricted And-parallelism in BTM. This means: (1) independent subgoals which have no shared variables can be executed in parallel; (2) dependent subgoals which have shared variables must be executed sequentially; (3) for subgoals that must be executed sequentially, they can turn to be executed in parallel as soon as the variable they share gets a binding value.

In BTM we precompile user's logic programs into the following 5 types of execution graph expressions (EGEs) at compile time:

\[ G \]
\[ (SEQ E_1, E_2, \ldots, E_n) \]
\[ (PAR E_1, E_2, \ldots, E_n) \]
\[ (GPAR(X_1, \ldots, X_k) E_1, \ldots, E_n) \]
\[ (IPAR(X_1, \ldots, X_k) E_1, \ldots, E_n) \]

The \( G \) expression indicates that the single goal \( G \) is executed. The \( SEQ \) expression indicates that the subexpression \( E_1 \) through \( E_n \) are executed sequentially in order. The \( PAR \) expression indicates that the following subexpressions can be executed in parallel. The \( GPAR \) expression tests its \( k \) variables, and if all are ground, the \( n \) subexpressions can be executed in parallel; if at least one of the \( k \) variables is not ground, then the subexpressions must be executed sequentially. The \( IPAR \) expression tests if the \( k \) variables are all mutually independent; if so, then the subexpressions \( E_1 \) through \( E_n \) can be executed in parallel; if any two of them are interdependent, then the \( n \) subexpressions must be executed sequentially.

Then at run time, test the type codes of some variables and analyze the dependency among the variables to determine whether the subgoals can be executed in parallel.

2.1.2 Control of Or-Parallelism

As discussed above, there are two types of Or-parallelism. To exploit the first one, we employ the method of parallel execution of all the clauses that match a goal predicate. To exploit the second one, for example, \( g(X,Y) :- a(X), b(X,Y) \), we first execute \( a(X) \). The execution of \( b(X,Y) \) is delayed until \( a(X) \) ends since subgoal \( b \) shares the variable \( X \) with subgoal \( a \). When \( a(X) \) receives a solution \( x_1 \), \( b(x_1, Y) \) can be executed at once. Henceforth, upon receiving each solution of \( a(X) \), say \( x_i, b(x_i, Y) \) can be executed at once. That means \( b(x_1, Y), \ldots, b(x_n, Y) \) can be executed in parallel. In this way the second type of Or-parallelism can be exploited.

2.2 Hierarchical Parallel Representation of a Logic Program

In BTM, we use a BTM-tree to represent a logic program. See Fig.1.

Here node 'SEQ' indicates all its sons should be executed sequentially; node 'PAR' betokens that all its sons can be executed in parallel. For node 'GPAR(X)', first check variable \( X \), if it is ground then all its sons can be executed in parallel; otherwise sequentially. Or-parallel branches are denoted by node ':-'.

From Fig.1 we see that in this representation method the parallel control information obtained at compile time is inserted into the representation tree and the layered represen-