Performance of a Data-Parallel Concurrent Constraint Programming System

Bo-Ming Tong* and Ho-Fung Leung

Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, New Territories, Hong Kong
email: bmtong@cs.cuhk.hk, lhf@cs.cuhk.hk

Abstract. Finite domain constraints [27] are very effective in solving a class of integer problems which find its applications in various areas like scheduling and operations research. The advantage over conventional approaches is that the problem is specified declaratively. The actual computation is carried out by logic inference and constraint satisfaction [24]. Solving a set of finite domain constraints is an intractable problem and we propose to use massively parallel computers to obtain satisfactory performance. In our previous papers, we have shown that finite domain constraint languages can be implemented on massively parallel SIMD machines. The resulting system, Firebird, has been implemented on a DECmpp 12000 Sx-100 massively parallel computer with 8,192 processor elements. In this paper, some preliminary performance results are given. A speedup of 2 orders of magnitude is possible when we compare the performance using 8,192 processor elements and the performance using a single processor element of the same machine. On the other hand, we measure the effects of several control strategies and optimizations on execution time and memory consumption in a data-parallel context.

1 Introduction

Finite domain constraint logic programming [27] is very effective in solving a class of integer problems which find its applications in various areas like scheduling and operations research. Examples include the car sequencing problem [5] and time-table scheduling [30]. The advantage over conventional approaches is that the problem is specified declaratively. The actual computation is carried out by logic inference and constraint satisfaction [24]. Solving a set of finite domain constraints is an intractable problem and we propose to use massively parallel computers to obtain satisfactory performance.

We have shown that finite domain constraint programming languages can be implemented efficiently on massively parallel SIMD computer systems. The result is Firebird [22, 23], which, to the best of our knowledge, is the first data-parallel concurrent constraint programming system.

In this paper, we present some preliminary performance results of our implementation. We also measure the effects of several control strategies and optimizations on execution.

* Present address: Department of Computer Science, University of Arizona, Tucson, AZ 85721, U.S.A. email: bmtong@cs.arizona.edu
time and memory consumption in a data-parallel context. The reader is referred to Van Hentenryck [27] for an introduction to finite domain constraints, and to Maher [13] and Saraswat et al. [16, 17] for concurrent constraint programming, but we give brief reviews in the rest of this section. In the next section, we give a brief review of our previous papers [22, 23]. Section 3 is the performance data and Section 4 is a comparison between SIMD MultiLog [18] and Firebird. We conclude with Section 5.

1.1 Concurrent Constraint Programming

ALPS [13] is a scheme to integrate constraint logic programming and concurrent logic programming. Saraswat [16, 17] develop the ideas further by introducing the concurrent constraint programming framework. Computation is modeled as the interaction of concurrent, cooperating agents exchanging information via a global store, which is a conjunction of constraints. An agent may assert (tell) new constraints to the store, as well as inquire (ask) whether a constraint is implied (entailed) by the store. The constraints in the store must be consistent (satisfiable), or the computation aborts.

Since each tell constraint is conjoined to the current store, the store is monotonically refined. As a result, a successful ask operation will remain successful throughout the rest of the computation. Thus, synchronization can be achieved by blocking ask—an agent blocks until the store is refined enough to entail or reject the constraint it wants to ask. It remains blocked until some other concurrently executing agents have added enough information to the store so that it is strong enough to entail the ask constraint.

1.2 Finite Domain Constraints

Recent treatment of finite domain constraints in the concurrent constraint programming framework, as in cc(FD) [28] and clp(FD) [4], represents a domain variable X with domain d as a constraint X \in d. As constraints are added to the store, the domain of each related variable shrinks, until it becomes a singleton or becomes empty. For example, X may take any value from 1 to 10 initially. A constraint X > 4 will rule out some of the values \{1...4\} in d. Now X can only range from 5 to 10. When a constraint X < 6 is added, the domain of X becomes a singleton, and we can deduce that X = 5.

The reader is referred to Van Hentenryck [27] for a full treatment of finite domain constraints in the traditional logic programming framework [12]. It is summarized as follows. Ordinary variables are termed h-variables (h stands for Herbrand). A d-variable X with domain d is denoted by X^d. The unification algorithm must be modified to support d-variables. The modified algorithm is termed d-unification. When an h-variable is unified with a d-variable, the former is bound to the latter. When a constant c is unified with a d-variable X^d, X is bound to c if c is in d. Otherwise the unification fails. When two d-variables, X^d and Y^e, are unified, both of them are bound to the d-variable Z^f where f = d \cap e. If f is a singleton \{c\}, both variables are bound to the constant c. If f is empty, the unification fails. SLD-resolution extended with d-unification is termed SLDD-resolution. However, the introduction of d-unification alone is insufficient to solve finite domain constraints efficiently. Disequality, inequality, (arithmetic) equality constraints,

\(^2\) An agent corresponds to an atom in traditional logic programming.