Embedding Constraint Satisfaction using Parallel Soft-Core Processors on FPGAs

Prasad Subramanian, Brandon Eames
Department of Electrical and Computer Engineering
Utah State University, Logan, Utah – 84322
prasad.subramanian@aggiemail.usu.edu, beames@engineering.usu.edu

Introduction

Constraint satisfaction and optimization techniques are commonly employed in scheduling problems, industrial manufacturing and automation processes, borrowing concepts from Operations Research (OR) and Artificial Intelligence (AI). Constraint satisfaction has also been applied in the design, synthesis and optimization of embedded systems. A problem particularly suited to constraint satisfaction is static scheduling and resource allocation, subject to end-to-end timing and resource constraints. Typical applications of constraint satisfaction involve the design-time specification of both the problem and non-functional requirements as constraints, which are provided to a solver program to search for an optimal or near-optimal solution.

In recent years a few examples of applying constraint satisfaction at runtime have emerged for supporting dynamic system adaptation and reconfiguration. The runtime selection between various web services and components based on user-specified Quality of Service (QoS) requirements has been modeled as a constraint satisfaction problem [1]. Constraint satisfaction has been used to realize an on-board paper path controller of a digital printer [2].

In this work, we address the development of an embedded finite domain constraint solver targeting an FPGA. Constraint solvers are concurrent in nature, and lend themselves to parallel implementation. We exploit the spatial parallelism offered by COTS FPGA architectures via the instantiation of multiple soft-core processors, which collectively implement the constraint solver. Soft-core processors facilitate the development of flexible software-based algorithms for implementing individual constraints. The multi-core architecture realized on the FPGA facilitates tight inter-core synchronization required when solving constraints in parallel.

Overview of Constraint Satisfaction

A finite domain constraint satisfaction problem (CSP) consists of set of variables *x*, as well as a set of basic constraints. Each variable is restricted to a domain of finite cardinality, typically consisting of a set of integer values. The basic constraints represent relations between variables. A solution to the CSP consists of a binding of values to the variables in *x*, such that all the basic constraints are satisfied. A solver is a program which takes a set of variables and basic constraints and searches for a solution to the CSP. A solver can be configured to find a single solution, all solutions, or alternatively, to find the best solution, where "best" is characterized by a user-specified cost function.

Figure 1 illustrates the internal architecture of a finite domain constraint solver [4]. A solver is developed around the concept of a central repository called the constraint store. The constraint store contains information on each variable and its domain in the CSP. Computational entities called propagators implement the basic constraints in the problem, by reading domain information on their constituent variables from the constraint store, and attempting to shrink the domains through mathematics and deduction. Each propagator executes as a separate thread. The bounds analysis implemented via propagation is insufficient to converge to a solution. Therefore, propagation is followed by a step called distribution, which conveys new information to the constraint store by means of variable assignment. At each distribution step, the solver saves the state of the constraint store, in case the new information added leads to a contradiction, which would necessitate backtracking. Through alternating between propagation and distribution, the constraint solver can exhaustively traverse the solution space and determine a solution to the CSP.

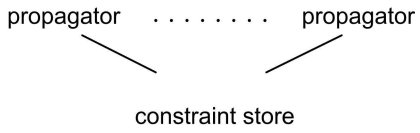

Figure 1: Constraint store.

Architecture of a Constraint Solver

Figure 2 represents the architecture employed for realizing an FPGA-based finite domain constraint solver. FPGAs lend themselves to spatial parallelism, offering large numbers of distributed memories and the ability to create numerous parallel processing elements. Current approaches to parallelization of finite domain constraint search focus on the partitioning of the CSP based on distribution steps [3]. Our approach seeks to exploit the inherent parallelism in the propagation step of the FD solver. Our architecture targets a Xilinx FPGA, and consists of a fixed set of soft-core Microblaze processors which share FSLs, fast point-to-point communication links native to the FPGA fabric. The model in Figure 1 implies the need for a globally shared memory, simultaneously accessible by all propagation elements for sharing variable information. Without such sharing, propagators cannot cooperate to jointly make progress toward a solution.

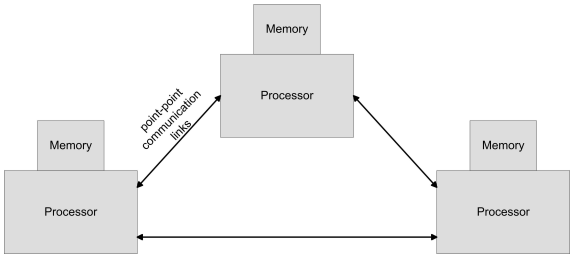

Figure 2: Distributed memory architecture.

The implementation of a low-latency, globally shared memory accessible by several computational devices is not practical on an FPGA fabric, due to limitations on the number of read-write ports of internal memories. Our implementation emulates shared memory by making use of on-chip BlockRAM. The constraint store is partitioned and distributed among the local memories associated with each soft core processor. Data sharing across partition boundaries is realized through interrupt-driven communication along the FSL links.

Coherency and the Consolidator

Emulation of shared memory via distributed memories and communication links imposes issues of coherency. Local copies of shared data are cached on each processor requiring the data. Coherency is maintained through two steps. First, when a finite domain variable is updated by a remote processor, updates are sent to the owning processor. Second, all updates are routed through a hardware unit called a consolidator, illustrated in Figure 3. The consolidator is a comparator that acts as a check point for all updates to the constraint store, and only accepts updates which improve or tighten the currently stored bounds for the variable.. Since propagation approaches a solution through bounds analysis of finite domain variables, ordering between updates need not be maintained.

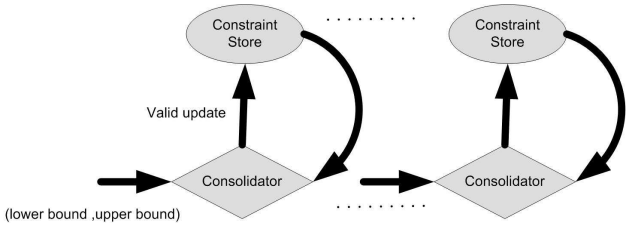

Figure 3: Consolidator

Distribution and Backtrack

When propagation stalls due to lack of new information in the constraint store, the solver must implement a distribution step. Distribution requires global barrier synchronization among all processors. To facilitate global synchronization, commands are broadcast to all processors. Command processing was modeled and implemented as a distributed set of finite state machines, which cause the collection and sharing of state information with a master node. Distribution causes all nodes to push their local state into a local configuration stack. The master node broadcasts a command targeting a single node in the network, to constrain a variable to a value. Once this is done, propagation proceeds. If a node determines that a constraint violation has occurred, a notification message is sent to the master node, which instructs all nodes to halt and backtrack to the most recently stored configuration on the configuration stack. It repeats the distribution step with a new value. The process halts when all variables have been bound to a value without encountering a constraint violation ("solution"), or no satisfying selection can be obtained ("failure").

Test Application

To evaluate our embedded finite domain constraint solver, we leveraged a hypothetical event graph from an autonomous space mission planning algorithm [5]. Our constraint model consists of enforcing temporal precedence constraints in order to derive a schedule of events in the graph. We employ a simple clustering algorithm to determine how to map propagators implementing constraints extracted from the graph model onto Microblaze processors. Measurements from the solver implementation executing on a VirtexII Pro Xilinx FPGA are provided in Table 1. Results indicate that a performance speed-up in propagation increases as the number of processors is increased.

Table 1: Speed-up in propagation for potential space application

Clusters	prop completion(ticks)	# distrib. Steps	First solution(ticks)	Cluster size	Prop speed-up
1	310209	60	FAILS	50/51	1.00
2	159608	47	2632804	34/34/33	1.94
3	109971	43	1668505	26/26/26/23	2.82
4	85914	65	2360598	26/23/26/26	3.61

The solver failed to converge in the single-processor case, due to lack of sufficient space in the local configuration stack. The number of processors involved in the problem, together with the number of variables in the constraint store dictates the storage requirements per configuration. Having fewer nodes imposes a larger per-configuration storage requirement, and with a large number of distribution steps, the solver exhausts the available stack space.

Conclusion

Online constraint satisfaction potentially opens the door to a variety of introspective dynamic optimizations to embedded systems. We have developed an approach for embedding a finite domain constraint solver on an FPGA, using a network of soft-core processors, distributed memories and point-to-point communication.

References

[1] M. Lin, J. Xie, H. Guo, and H. Wang, "*Solving QoS-driven web service dynamic composition as fuzzy constraint satisfaction*," in Proc. 2005 IEEE Intl. Conf. e-Technology, e-Commerce and e-Service. Wash., DC, 2005.

[2] M.Fromherz, "*Constraint-based scheduling,*" American Control Conference, vol. 4, pp. 3231–3244, 2001.

[3] Schulte, C. "Parallel Search Made Simple", Tech Rprt TRA9/00, Nat. Univ. Singapore, Sept. 2000.

[4] M. Henz and T. M¨uller, "*An overview of finite domain constraint programming,*" in Proc. 5th Conf. Assoc. of Asia-Pacific Operational Research Societies, 2000.

[5] A. Dasu and J. Phillips, "*Deriving FPGA based custom soft-core microprocessors for mission planning algorithms,*" in 21st AIAA/USU Conf. Small Satellites. Aug, 2007.