Cover Page

Mapping Applications onto Reconfigurable Architectures using Dynamic Programming [1]

Kiran Bondalapati, George Papavassilopoulos and Viktor K. Prasanna

Department of Electrical Engineering Systems, EEB-200C
University of Southern California
Los Angeles, CA 90089-2562
USA

Contact Author
Viktor K. Prasanna
Email: prasanna@usc.edu
Tel: +1-213-740-4483
Fax: +1-213-740-4480
http://ceng.usc.edu/˜prasanna/

[1] This work was supported by the DARPA Adaptive Computing Systems Program under contract DABT63-96-C-0049 monitored by Fort Hauchuca.

Mapping Applications onto Reconfigurable Architectures using Dynamic Programming

Kiran Bondalapati, George Papavassilopoulos and Viktor K. Prasanna

Introduction

Reconfigurable architectures vary from systems which have FPGAs and glue logic attached to a host computer to systems which include configurable logic on the same die as a microprocessor. Automatic compilation of applications onto reconfigurable architectures involves not only configuration generation, but also configuration management. Currently, there is no unified methodology for mapping applications to configurable hardware.

In this paper we describe algorithmic techniques for automatic mapping of applications in a platform independent fashion. We have developed an abstract model of reconfigurable architectures. This parameterized abstract model is general enough to capture a wide range of configurable systems. These include board level systems which have FPGAs as configurable computing logic to systems on a chip which have configurable logic arrays on the same die as the microprocessor.

Configurable logic is very effective in speeding up regular, repetitive computations. Loop constructs in general purpose programs are one such class of computations. In this paper, we address the problem of mapping a loop construct onto configurable architectures. The Hybrid System Architecture Model(HySAM) that we have developed is utilized to define the mapping problems. Efficient techniques based on dynamic programming are used to develop an optimal schedule for important variants of the problem. The problem of utilizing on-chip reconfiguration cache resources is addressed in this paper. The techniques are illustrated by mapping an example FFT loop onto the Berkeley Garp architecture.

Hybrid System Architecture Model(HySAM)

To realize a formal framework for algorithm development, we developed the Hybrid System Architecture Model of reconfigurable architectures. The *Hybrid System Architecture* is a general architecture consisting of a conventional microprocessor with an additional Configurable Logic Unit(CLU). The architecture consists of a conventional microprocessor, standard memory, configurable logic, configuration memory and data buffers communicating through an interconnection network. Key parameters of the Hybrid System Architecture Model(HySAM) are outlined below.

F	: Set of functions $F_1 \ldots F_n$ which can be performed on configurable logic.
C	: Set of possible configurations $C_1 \ldots C_m$ of the Configurable Logic Unit.
A_{ij}	: Set of attributes for implementation of function F_i using configuration C_j (execution time, precision etc.).
R_{ij}	: Reconfiguration cost in changing configuration from C_i to C_j.
G	: Set of generators which abstract the composition of configurations to generate more configurations.
B	: Bandwidth of the interconnection network(bytes/cycle).

The parameterized HySAM models a wide range of systems from board level architectures to systems on a chip. The values for each of the parameters establish the architecture and also dictate the class of applications which can be effectively mapped onto the architecture. For example, a system on a chip architecture would have potentially faster reconfiguration times than a board level architecture.

Mapping Loop Statements

Scheduling a general sequence of tasks with a set of dependencies to minimize the total execution time is known to be an NP-complete problem. We consider the problem of generating this sequence of configurations for loop constructs which have a sequence of statements to be executed in linear order. There is a linear data or control dependency between the tasks. Most loop constructs, including those which are mapped onto high performance pipelined configurations, fall into such a class.

The total execution time includes the time taken to execute the tasks in the chosen configurations and the time spent in reconfiguring the logic between successive configurations. We have to not only choose configurations which execute the given tasks fast, but also have to reduce the reconfiguration time. It is possible to choose one of many possible configurations for each task execution. Also, the reconfiguration time depends on the choice of configurations that we make.

Problem: Given a sequence of tasks of a loop, T_1 through T_p to be executed in linear order($T_1\ T_2\ \ldots\ T_p$), where $T_i \in F$, for N number of iterations, find an optimal sequence of configurations S (=$C_1\ C_2\ \ldots\ C_q$), where $S_i \in C$ (=$\{C_1, C_2, \ldots, C_m\}$) which minimizes the execution time cost E. E is defined as

$$E = \sum_{i=1}^{q} (t_{S_i} + R_{i-1i})$$

where t_{S_i} is execution time in configuration S_i and R_{i-1i} is reconfiguration cost.

Optimal Solution for Mapping Loops

A simple greedy approach of choosing the best configuration for each task will not work since the reconfiguration costs for later tasks are affected by the choice of configuration for the current task. We outline our dynamic programming based approach below without proofs:

Lemma 1: Given a sequence of tasks $T_1'T_2' \ldots T_p'$, an optimal sequence of configurations for executing these tasks **once** can be computed in $O(pm^2)$ time.

Lemma 1 provides a solution for an optimal sequence of configurations to compute one iteration of the loop statement. But repeating this sequence of configurations is not guaranteed to give an optimal execution for N iterations.

Lemma 2 An optimal configuration sequence can be computed by unrolling the loop only m times.

Theorem 1 The optimal sequence of configurations for N iterations of a loop statement with p tasks, when each task can be executed in one of m possible configurations, can be computed in $O(pm^3)$ time. $\odot$

Theorem 1 is derived from Lemma 1 and Lemma 2 and the complexity of the algorithm is $O(pm^3)$. This approach can also be used when the number of iterations N is not known at compile time and is determined at runtime. The decision to use this sequence of configurations to execute the loop can be taken at runtime from the statically known loop setup and single iteration execution costs and the runtime determined N.

Multiple Contexts and Configuration Caches

The performance achievable on reconfigurable architectures is limited by the costs involved in reconfiguring the logic. Currently, this overhead is very high and discourages the reconfiguration of the logic during the execution of a single application. To address this problem architectures which support configuration caches and multiple contexts on the devices are being developed. We extend the above approach for these devices with the following assumptions regarding the HySAM model:

1. N_c number of configurations can be loaded on to the device at the start of the computation.
2. There is one active context which can be configured from any of the N_c configurations with a cost k_c.
3. The pre-loaded configurations can not be modified during the execution of the complete application. Only the active context can be reconfigured externally.

We define an additional variable X_{ij}, $1 \leq j \leq 2*m$, which is the set of contexts which are cached for executing tasks T_1 to T_i with T_i being executed using configuration C_j. The E_{ij} and the X_{ij} ($1 \leq i \leq 2*m$) values are computed using *dynamic programming*. The recursive equations for computing them are given below(δ_{kj} denotes the reconfiguration cost):

$$
\begin{aligned}
&mink = k \; s.t. \; min[E_{ik} + \delta_{kj}] \;\; 1 \leq k \leq 2*m \\
&\quad if \; (C_j \in X_{ik}) \\
&\qquad \delta_{kj} = k_c \\
&\quad else \; if \; (|X_{kj}| < N_c \;\; and \;\; 1 \leq j \leq m) \\
&\qquad \delta_{kj} = k_c \\
&\quad else \\
&\qquad \delta_{kj} = R_{ij}
\end{aligned}
$$

Given the value of $mink$, the E_{i+1j} and the X_{i+1j} values are computed as follows:

$$
\begin{aligned}
E_{i+1j} &= t_{i+1j} + E_{i\,mink} + \delta_{mink\; j} \\
X_{i+1j} &= X_{i\,mink} \cup C_j \\
&\qquad if \; |X_{imink}| < N_c \;\; and \;\; 1 \leq j \leq m) \\
&= X_{i\,mink} \quad otherwise
\end{aligned}
$$

The minimum execution cost E and the corresponding set of contexts X for executing tasks T_1 to T_p are given by:

$$
\begin{aligned}
minj = j \; s.t. \; min[E_{pj}] \;\; 1 \leq j \leq 2*m \\
E = E_{p\,minj} \\
X = X_{p\,minj}
\end{aligned}
$$

The required optimal execution cost and the set of contexts can be computed by using *dynamic programming*. $\odot$

Illustrative Example

We illustrate the techniques by mapping the loop containing FFT butterfly operations. The butterfly operation consists of one complex multiply, one complex addition and one complex subtraction. First, the loop statements were decomposed into functions which can be executed on the CLU, given the list of functions in Table 1. One complex multiplication consists of four multiplications, one addition and one subtraction. Each complex addition and subtraction consist of two additions and subtractions respectively. The statements in the loop were mapped to multiplications, additions and subtractions which resulted in the task sequence T_m, T_m, T_m, T_m, T_a, T_s, T_a, T_a, T_s, T_s. Here, T_m is the multiplication task mapped to function F_1, T_a is the addition task mapped to function F_2 and T_s is the subtraction task mapped to function F_3.

The optimal sequence of configurations for this task sequence, using our algorithm, was C_1,C_3,C_4,C_3,C_4 repeated for all the iterations. The most important aspect of the solution is that the multiplier configuration in the solution is actually the slower configuration. The reconfiguration overhead

Function	Operation	Configuration	Configuration Time	Execution Time
F_1	Multiplication(Fast)	C_1	14.4 μs	37.5 ns
	Multiplication(Slow)	C_2	6.4 μs	52.5 ns
F_2	Addition	C_3	1.6 μs	7.5 ns
F_3	Subtraction	C_4	1.6 μs	7.5 ns
F_4	Shift	C_5	3.2 μs	7.5 ns

Figure 1: Representative Model Parameters for Garp Reconfigurable Architecture

is lower for C_2 and hence the higher execution cost is amortized over all the iterations of the loop. The total execution time is given by $N * 13.055$ μs where N is the number of iterations.

Conclusions

Mapping of applications in an architecture independent fashion can provide a framework for automatic compilation of applications. Loop structures with regular repetitive computations can be speeded-up by using configurable hardware. We developed dynamic programming based approaches to efficiently map tasks in a loop to a sequence of configurations. We illustrated our approach by developing algorithms for some variants of the mapping problem.