Embedded Compilation for Multimedia Applications

Nathaniel Daw, Seth Goldstein, and Dennis Strelow
Carnegie Mellon University
Pittsburgh, PA 15217

1 Introduction

Reconfigurable computing obtains its performance advantage over fixed processors by creating hardware configurations specialized for a particular application. In some cases this advantage can be pushed even further, by creating hardware specialized to a particular instance of an application. For many problems where this approach is applicable, such as automatic target recognition, template matching, and encryption, the problem parameters can change often even within a single program execution, requiring periodic, and potentially expensive, hardware reconfigurations.

To support these applications, we propose a method for on-chip configuration generation, or embedded compilation, for use with CMU's PipeRench reconfigurable processor. We describe PipeRench's performance in detail for one problem, template matching, relative to the newest general-purpose processors, and show how embedded compilation can be used to support multiple problem instances for a second problem, IDEA encryption.

2 PipeRench

PipeRench [2] is an instance of the class of pipelined reconfigurable fabrics [4]. From the point of view of implementing runtime specialized hardware the three most important characteristics of PipeRench are: it has an on-chip configuration cache to support *hardware virtualization*; it has an on-chip controller to manage the partial runtime configuration; and its configuration bitstream is compact and regular. Hardware virtualization allows PipeRench to efficiently execute configurations larger than the size of the physical fabric, which allows runtime specialized hardware the ability to shrink or grow without requiring any of the non-specialized parts of the configuration to change. The on-chip configuration cache, which is managed by a small controller, means that there is already a source for the templates which will be modified at runtime.

3 Template Matching

Template matching, in which an image is searched for the regions that best match a given intensity pattern (i.e., template), is a basic mechanism for tracking and motion estimation. Template matching is one application where hardware specialization for a particular problem instance greatly speeds computation and reduces the amount of required hardware. Here, we briefly compare PipeRench's performance for this problem against the newest general purpose processors, which contain considerable instruction set support for template matching.

Reports from Intel describe the efficient implementation of template matching for MPEG encoding using MMX and Streaming SIMD [3]. Timing the assembly language routines from these reports on a 500 MHz Pentium III gives the results in Table 1. To determine the effect of cache misses, we have performed timings by matching over entire input images, which produces cache misses we expect from a normal application, and over a single location in an image, which produces no cache misses. Table 1 shows performance without cache misses.

The estimated times for our PipeRench implementation, also shown in Table 1, compare very favorably with the Intel architecture measurements. The current implementation of PipeRench, fabricated using .35 micron technology, is more than twice as fast as the Pentium. A hypothetical PipeRench chip fabricated using .18 micron technology, similar to that used for the Pentium III, with corresponding increases in cycle time and pipeline stages, would be almost 20 times as fast.

4 Embedded Compilation

For applications like template matching, IDEA encryption, and ATR, problem parameters can change often within a single execution. Template matching and ATR, for instance, may match several hundred templates for each input image. For IDEA encryption, it is the key that changes. In each case, the problems parameters are smaller than the resulting PipeRench configuration, so it makes sense to download only the parameters and to generate the correponding configuration on the chip.

This embedded compilation can be achieved by adding a small amount of hardware to PCI-PipeRench and corresponding functionality to PCI-PipeRench's

architecture	matches per second
500 MHz MMX	1.78×10^6
500 MHz Streaming SIMD	7.31×10^6
.35 micron PipeRench	14.8×10^6
.18 micron PipeRench	124.6×10^6

Table 1: Error function computations per second for the Intel and PipeRench arch itecutures

SwordAPI [1]. In this scheme, runtime specializable configurations are represented as collections of 640-bit stripe configurations, and are stored in PipeRench's virtual stripe cache, just as normal PipeRench programs are. In addition to the configuration, the host supplies a program for a newly introduced microcontroller, and precomputed reconfiguration information to be stored in, and accessed via, a newly introduced lookup table. When problem parameters are received, once before and potentially many times after execution of the application begins, the microcontroller uses the supplied microcontroller program, the parameters, and the lookup table information to overwrite the appropriate fields of the PipeRench configuration template, constructing a functional PipeRench program.

The SwordAPI is augmented to allow the host to specify the microcontroller program, lookup table data and program parameters. SwordAPI currently specifies four packet types, including "data to be processed," which specifies data intended as input to the pipeline, and "configuration," which includes data for the configuration controller. To these four types we add the new types "microcontroller code," "lookup table data," and "program parameters," which are directed by the input controller to the microcontroller, the lookup table, and the microcontroller, respectively.

Our most demanding example application with respect to reconfiguration is IDEA encryption. The problem parameters for IDEA are 34 16-bit constant multiplication operands, and each constant is used to configure a multiplier spanning three stripes. Each 16×16-bit multiplication is broken into two 8×16-bit multiplications by splitting the 16-bit constant into its upper and lower bytes. To limit the number of partial products required to implement each 8-bit multiplication to a maximum of five, the 8-bit constants are converted to CSD form, which uses 1, 0, and -1, rather than just 0 and 1, as the digits of the word.

Filling the IDEA template configuration requires the determination of CSD forms, shifts, and ALU operations from each byte of the 16-bit word. Since these are complex operations, we precompute this information for each possible byte value and store the results in the lookup table. When a new 16-bit operand is received from the host, the CSD forms, shifts, and ALU operations for each can be extracted from the appropraite lookup table word, and placed into the correct configuration word locations. Using this scheme, the required complexity of the controller can be greatly reduced at the cost of a small lookup table.

5 Conclusion

Reconfigurable architectures can create hardware specialized not only to a particular problem, but to particular instances of a problem. However, if problem parameters change frequently, communicating configuration information between the host and the fabric can create a bottleneck that wastes the advantage of specialization. Here, we have shown that embedded compilation can be performed on the chip, drastically reducing the amount of reconfiguration information that must be sent by the host, and preserving reconfigurable computing's advantage for multimedia applications like template matching and encryption.

Acknowledgements

Many thanks to Mihai-Dan Budiu, Srihari Cadambi, Matthew Moe, and Reed Taylor; and to Mark Ollis and Sanjiv Singh for many helpful discussions on template matching implementations and applications.

References

[1] R. Laufer, et al., "PCI PipeRench and the SwordAPI – a system for streaming-based reconfigurable computing," in *Proceedings of the Symposium on Field-Programmable Computing Machines*, 1999.

[2] S.C. Goldstein, et al., "Piperench: A coprocessor for streaming multimedia acceleration," in *Proceedings of the 26th Annual International Symposium on Computer Architecture*, pages 28–39, May 1999.

[3] Intel Corporation, "Using streaming SIMD extensions in a motion estimation algorithm for MPEG encoding," Intel Application Note 818.

[4] H. Schmit, "Incremental reconfiguration for pipelined applications," in J. Arnold and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 47–55, Napa, CA, April 1997.