Interfacing Reconfigurable Logic with a CPU

Kip Walker, Mihai Budiu, Seth Copen Goldstein
{kwalker,mihaib,seth}@cs.cmu.edu
School of Computer Science
Carnegie Mellon University

1 Introduction

Reconfigurable computing devices have achieved substantial performance improvements over conventional processors on some computational kernels. These benefits derive from hardware customization which avoids the mismatch between the basic requirements of the algorithms and the architectures of the processors. A reconfigurable fabric alone is not sufficient for general-purpose computing since it can be ill-suited to executing entire programs due to space limitations, dataflow-centricity, and inefficiency at implementing some operations (e.g. floating-point arithmetic). These observations have led to the appearance of numerous designs which place some form of reconfigurable logic under the control of a general-purpose processor.

In this abstract we explore the ways in which a reconfigurable fabric can be interfaced with a general-purpose processor. While off-chip reconfigurable fabrics have proven to be quite effective at performing streaming, data-intensive computations, they require large streams of data to overcome the latency between the devices. Here we explore the design space for an on-chip fabric, i.e., a *reconfigurable function unit* (RFU). An RFU allows smaller portions of application to be mapped to the fabric in the form of *custom instructions*. Though the speedups achieved for stream-based computations will in general be much larger than those for custom instructions, they are limited to a smaller class of applications. Custom instructions, however, can be found in a larger class of programs, and compiler techniques can automatically create them.

The basic tradeoff in augmenting a processor with a reconfigurable function unit is to maximize both the opportunity for customization and the speedups possible while minimizing hardware costs and runtime overheads. To broaden the range of computations that may be mapped to the fabric, many features are worth considering such as maintaining state in the fabric, supporting simultaneous direct access to many registers, and making memory accessible to the RFU. However, each of these options carries an associated cost in terms of implementation and overhead: saving fabric state on context switches, increasing the number of ports on the register file, and dealing with memory consistency between the processor and the fabric.

Several designs have proposed the integration of a given reconfigurable fabric with a general-purpose fixed processor and have reported some exciting performance results. Some important questions have not been answered by these projects, such as: Are there enough opportunities for custom instructions to warrant including them in a hybrid processor design? Which fabric features are essential to attaining speedups for different benchmarks? How many inputs and outputs are needed to support useful custom instructions?

2 The RFU Design Space

The designer of a hybrid processor faces numerous decisions. In this abstract we outline the major axis of the design space. Table 1 shows the design choices made in previous work.

Access method Access to data is one of the fundamental considerations in any function unit. Primary sources and sinks of data are the register file and main memory or the first-level data cache. While streaming operations obviously need efficient access to memory, there are many opportunities for finding custom instructions (CIs) that read and write only data from a register file. Register ports are an expensive component of processors; one way to avoid overloading the main register file(s) is to employ a *shadow register file*. Alternatively a designer may choose to require explicit moves of the data using coprocessor instructions.

Number of I & Os In order to increase the opportunities for creating custom instructions a designer can choose to increase the number of inputs and outputs allowed for a custom instruction. RFUs which require explicit moves allow unlimited numbers of inputs and outputs, without requiring additional register ports.

Input selection When inputs come from the register file, the registers may be read by the processor and passed to the fabric, or the configuration might hard-code the indices of the inputs. The latter can be attractive since minimal control overhead is required from the CPU to invoke the custom instruction. However, it complicates the job of the code scheduler and register allocator.

Input width The width of the input and output buses are critical to accessing the power of a reconfigurable logic array. Many of the existent designs include a fabric that is closely matched to the CPU's natural word size. While wider inputs to the RFU increase flexibility, they require extra hardware to arrange the data.

0-7695-0871-5/00 $10.00 © 2000 IEEE

Feature	*Design*				
	PRISC [3]	Garp [2]	Chimaera [1]	NAPA1000 [4]	OneChip-98 [5]
Access method	registers	coproc+mem	shadow reg	memories	memory
Number of I & Os	2-to-1	n/a/	9-to-1	n/a	n/a
Input selection	instruction	instruction	configuration	instruction	n/a
Input width	natural	natural	natural	wide	natural
Transactions	atomic	split	implicit/split	split	atomic
Autonomy	none	loop	none	yes	loop?
Clocking	sync	async	sync	async	sync
Exit method	fall-through	cond+branch	fall-through	cond+branch	fall-through
Pipelining	no	yes	no	yes	no
State	no	yes	no	yes	?

Table 1: *Taxonomy of previously published CPU+RFU designs.*

Transactions A direct-style invocation appears as an atomic operation in the instruction stream, although the latency may be large and other instructions may execute concurrently. The coprocessor-style invocation serves as a split-transaction model. This model would favor the implementation of software-pipelined operations.

Autonomy A truly datapath-centric RFU requires the core to handle all control flow. A more autonomous design would allow for the loop control to be executed by the fabric itself, freeing the CPU to execute independent instructions. A more complex design would allow for completely autonomous RFU.

Clocking It has been argued that to be useful, the RFU must run at a rate that is reasonably matched with that of the processor core. Whatever clock rate is chosen for the fabric, there is a choice between synchronous and asynchronous clocking.

Exit method Mapping larger sections of program code to the RFU may require support for hyperblocks: code segments with a single entry point and multiple exits. When the RFU completes, the CPU must determine which exit was taken in order to continue executing.

One solution is for the RFU to set a condition value for the CPU to check. The processor can switch on this value and jump to the correct continuation code. A simpler implementation option is to always fall-through to the next instruction, but have the fabric return a continuation address which the CPU jumps to.

Pipelining Efficient mapping of small loops without fabric autonomy requires the ability to pipeline RFU operations.

State Some of the fabrics need to keep internal state in registers: for instance loop-carried dependencies and for double-pipelining. Allowing state, however, can cause runtime overheads when the configuration needs to be switched out.

3 Conclusion

Closely coupling a reconfigurable fabric with a conventional processor enables application speedups through both stream-based computations and custom instructions. While more potential speedup is available from streaming instructions, the custom instructions can easily be synthesized automatically from unannotated C code, and can applied to a wider range of applications. If the RFU can be designed to provide even a modest speedup for all applications through custom instructions, then it will be available to bring the tremendous speedups shown for streaming functions.

Acknowledgements

This work was supported by DARPA contract DABT63-96-C-0083 and an NSF CAREER award.

References

[1] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao. The Chimaera reconfigurable functional unit. In *IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '97)*, pages 87–96, April 1997.

[2] John R. Hauser and John Wawrzynek. GARP: A MIPS processor with a reconfigurable coprocessor. In J. Arnold and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 12–21, Napa, CA, April 1997.

[3] R. Razdan and M. D. Smith. A high-performance microarchitecture with hardware-programmable functional units. In *Proceedings of the 27th Annual International Symposium on Microarchitecture*, pages 172–80. IEEE/ACM, November 1994.

[4] C. Rupp, M. Landguth, T. Garverick, E. Gomersall, H. Holt, J. Arnold, and M. Gokhale. The NAPA adaptive processing architecture. In *IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '98)*, April 1998.

[5] R. Wittig and P. Chow. OneChip: An FPGA processor with reconfigurable logic. In *IEEE Symposium on FPGAs for Custom Computing Machines*, 1996.