

Automatic Generation of Application Specific Processor Libraries from a High Level Description

Robert Dimond, Oliver Pell, Oskar Mencer and Wayne Luk
Department of Computing, Imperial College, London
{rgd,op,oskar,wl}@doc.ic.ac.uk

April 1, 2006

Application Specific Instruction Processors (ASIPs) provide enhanced performance by directly implementing application segments in hardware as custom instructions. Recent work [1, 2] automates generation of custom instructions from application source code. Speedups of up to 6 times [3] are attainable for certain benchmarks. However, instructions generated from C are often restricted to small clusters of operations due to limited inference of parallelism. For fundamental operations, e.g. Fast Fourier transform, an alternative approach is to provide libraries of optimised, parallel custom instruction implementations.

We automatically generate libraries comprising hardware custom instructions, software and their associated interfaces from a high-level description. Using a single, concise description in the Quartz [4] language, we generate parameterised libraries for performance critical operations. Parameters enable exploration of hardware/software partitioning and degree of pipelining, so that the library is customised to meet the required performance and area constraints.

The Quartz language is inspired by concepts from functional and relational programming. Quartz supports polymorphism and overloading with type inference, higher-order functions (functions that can be parameterised by other functions) and relational composition operators to facilitate clear and efficient description of highly parameterised hardware libraries. Our Quartz compiler generates both hardware custom instructions (in VHDL) and software (C) from a single specification. We use Quartz higher-order functions to control the partitioning between software and hardware. Thus, we produce multiple library implementations automatically, each with a different performance and area trade-off. Moreover, our approach requires only the single, concise Quartz code to be verified correct, as opposed to many hardware/software designs.

Our CUSTARD [3, 5] infrastructure automatically generates application specific processors from the Quartz hardware description. CUSTARD generates synthesisable RTL for parameterisable, embedded MIPS instruction set processors including custom instructions. We support various implementation options for CUSTARD: different pipeline depths, different coupling to custom instructions and optional hard-multiplier. For accurate cycle counts, frequency and

Hardware Width	Area (Slices)	
	Sorter	FFT
2	136	333
4	479	923
8	1645	5347
16	5127	-

Table 1: Area cost of generated Sorter and FFT hardware

area results we synthesise generated custom processors to a Xilinx VirtexIIPro FPGA. We compile the software part of each library using a CoSy compiler extended to support CUSTARD and custom instructions.

In this abstract we present results for two designs: a bitonic sorter and a fixed-point Fast Fourier transform (FFT). For each library, we generate multiple instances, varying the partitioning between hardware and software. We benchmark each instance across a range of problem sizes: the number of elements to sort for the sorter and number of points for the FFT. Figures 1 and 2 show performance results for sorter and FFT implementations respectively. Each line shows a different problem size implemented in hardware, e.g. Hardware 2 implements a two point FFT as a custom instruction and builds larger FFTs by reusing this instruction in software. Table 1 shows the area cost of custom instructions, out of 13696 slices available on the FPGA. For the largest (16 input) sorter implemented entirely as a custom instruction, we obtain nearly 38 times speedup over software, taking into account the overhead of calling the custom hardware. For the 8 input FFT, we obtain nearly 15 times speedup. For the largest problem size where computation is split between hardware and software, we obtain 53% speedup for sorting and 57% speedup for FFT.

References

- [1] K. Atasu, G. Dündar, C. Özturan An Integer Linear Programming Approach for Identifying Instruction-Set Extensions, In *CODES+ISSS 2005*, pages 172–177, Jersey City, NJ, Sept. 2005.
- [2] N. Clark, H. Zhong, S. Mahlke. Processor Acceleration Through Automated Instruction Set Customization. In *36th MICRO*, pages 184–88, San Diego, CA, Dec. 2003.
- [3] R. Dimond, O. Mencer, W.Luk CUSTARD - A Customisable Threaded FPGA Soft-processor and Tools In *FPL 2005*, August 2005.
- [4] O. Pell, W.Luk Quartz: A Framework for Correct and Efficient Reconfigurable Design. In *RECONFIG 2005*, September 2005.
- [5] R. G. Dimond, O. Mencer, W.Luk Combining Instruction Coding and Scheduling to Optimize Energy in System-on-FPGA. To appear in *FCCM 2006*, April 2006.

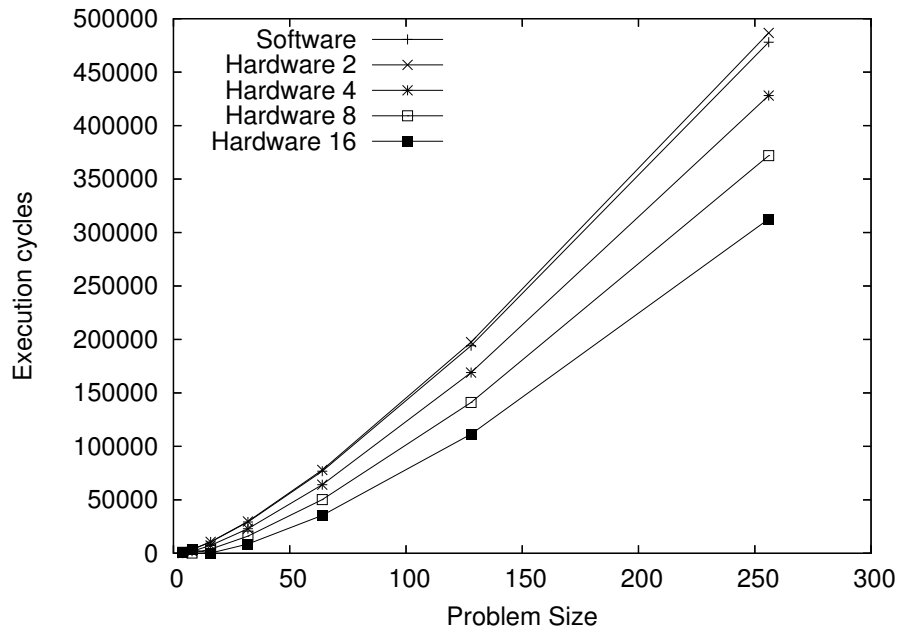


Figure 1: Execution cycles for Sorter implementations

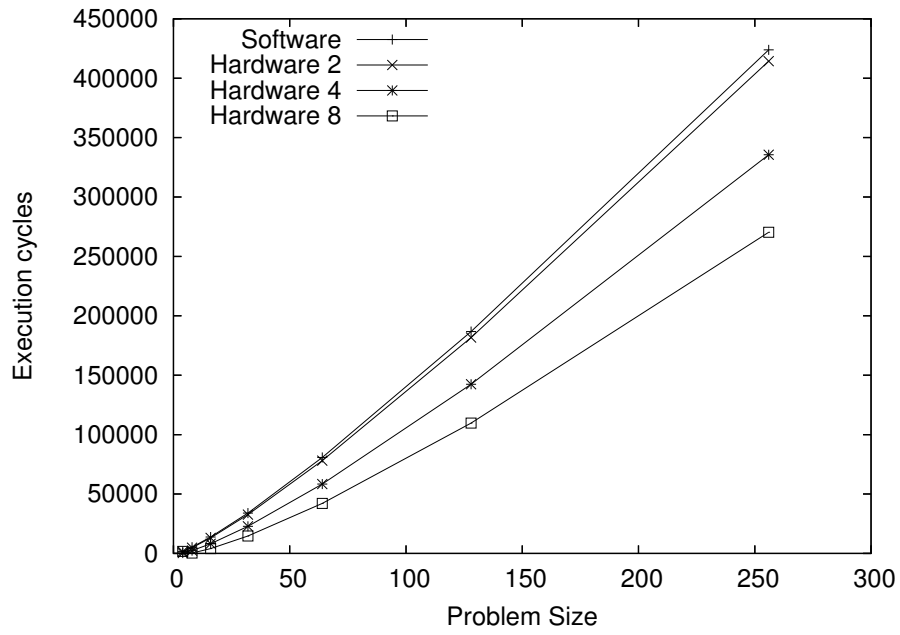


Figure 2: Execution cycles for FFT implementations