A C to Hardware/Software Compiler

Kiarash Bazargan Ryan Kastner Seda Ogrenci Majid Sarrafzadeh

Department of Electrical and Computer Engineering
Northwestern University
2145 Sheridan Rd.
Evanston, IL 60208-3118
{kiarash,kastner,seda,majid}@ece.nwu.edu

ABSTRACT

Improvements in the FPGA technology have resulted in introduction of reconfigurable computing machines, where the hardware adapts itself to the running application to gain speedup. This paper presents a top-down compilation method, under development, for such systems. We compile a C program into hierarchical VHDL source files, and annotate them with the placement information of the hardware modules to be configured on the FPGA. Static scheduling combined with a fast, two-stage placement core reduces the compilation time of large programs to minutes.

1. INTRODUCTION

Reconfigurable computing systems (RCS) are the next promising alternatives to costly high-performance multi-processors. A typical RCS is a host processor, coupled with a reconfigurable "co-processor". The co-processor, or as we call it, *reconfigurable functional unit* (RFU), could be an FPGA, or some reconfigurable fabric on the same die as the CPU.

One of the challenging problems in realizing a general-purpose reconfigurable processor, is devising fast compilation/debugging methods. Ideally, a programmer should be able to write his/her code (e.g., DSP, encryption, compression, etc.) in a high-level language and compile it as we compile C programs today. The compilation/debug cycle should be short. Such a scenario does not leave room for logic synthesis, placement and routing of the circuit at the gate-level. Extensive use of versatile IP libraries, as well as fast physical design methods seem to be a necessity.

Future reconfigurable computing systems (RCS) tend to integrate the RFU and on-chip cache (possible even the CPU) on the same chip to overcome latencies in CPU/RFU communication. The data communication between CPU and RFU would be through fast on-chip memory blocks (e.g. Xilinx's Virtex chip [7]), as well as general- and special-purpose register files (Chimaera [4] and GARP [5]). Our method handles both communication schemes.

2. PREVIOUS WORK

There has been some work in compilation of programs onto reconfigurable computing systems in the last few years (e.g., Garp and [3]). Some of them target specific architectures, and some need the user to identify which parts of the code should be mapped to reconfigurable fabric (e.g., the PRISM compiler, Garp compiler).

To this date, none of the compilation methods deal with automatic placement or routing of the RFUOPs. We propose an integrated approach for compilation and simultaneous scheduling of the operations and placement of the hardware modules.

3. THE COMPILATION PROCESS

Our model runtime system is a statically configured RFU, (tightly) coupled with the CPU. The RFU works as a coprocessor to the CPU. At any given time, either of the CPU or RFU is executing parts of the program.

Selection of the RFUOPs to allocate from the IP library, as well as their locations on the chip, is done at compile time. Once the set of RFUOPs and their locations are decided, the configuration bit-stream could be generated and stored as a "hardware code". When the program is loaded into memory, the configuration should be streamed to RFU as well before the program starts running. Figure 1 shows the compilation flow, from a C program into an assembly and corresponding hardware code.

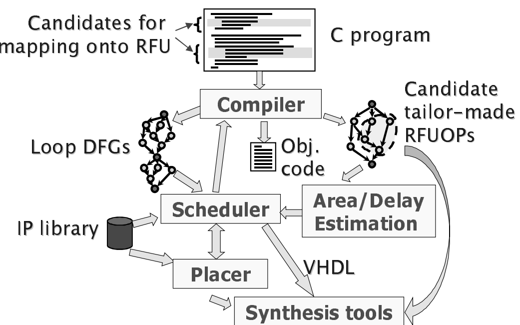

Figure 1 Compilation flow

The compiler in Figure 1 is a modification of *gcc* under development at Northwestern University [8]. Although it is targeted for the Chimaera architecture [4], it can be used for any reconfigurable architecture. In our current compilation to hardware, we do not consider tailor-made RFUOPs (Figure 1).

Loop bodies in a C program are compiled to data flow graphs (DFG). Loop unrolling is used to create more code parallelism. The scheduler (see Figure 1) allocates resources from the IP library for each of the loops and schedules each loop independently. Meanwhile, the placer generates a *local placement* for the RFUOPs implementing any given loop. We refer to the set of RFUOPs implementing a loop as a *loop block*.

After local placements are generated for each of the loop blocks, the scheduler decides on which one of the loops to actually map onto RFU. At first, the loop block with highest speedup to area ratio is chosen and *globally placed* on the RFU. Other loop blocks are placed in the order of their speedup to area ratio until there is not enough room on the RFU or all loop blocks with latency gain have been placed on the RFU.

The VHDL codes corresponding to datapaths are generated, and synthesized using the synthesis tools (Xilinx Foundation Series [7]). If the synthesis tool fails to place and route the whole

design, the scheduler starts again, using less area for placing hardware modules.

4. STATIC RCS SCHEDULING

Our scheduler (non-preemptive) has an internal loop in which resource allocation and operation scheduling are done repeatedly. Resources are added in the order of latency gain to area ratio. After all resources for a loop have been added, registers and multiplexers are added and a final local placement is generated for the loop. We employ heuristics to minimize congestion and high fanins, but for brevity, we will not discuss them in this paper.

For the operation scheduling, we have modified the well-know list-scheduling algorithm to account for different running times of the operations on different resources (e.g., CPU vs. RFUOP). Our scheduler also tries to minimize routing/congestion costs by binding an operation and its successor to two resources which already are connected due to other node/successor pairs. Furthermore, to decide on the order of the operations to schedule in the list-scheduling algorithm, we not only look at how time-critical a node is, but we also consider how promising it is in allowing its successor nodes to be scheduled earlier.

5. PLACEMENT OF RFUOPS

RFUOPs in the same data path are packed into larger *blocks* which do not communicate with other *blocks* significantly, hence no routing is needed between thesm. Therefore, the placement problem at the level of hierarchical blocks can be solved much faster than a flat model.

For the local placement, we employed the method suggested in [2], which is a very fast linear placement algorithm for datapaths. For the global placement, when the area of currently selected modules is less than a fixed threshold (70% of the chip area in our experiments), then we assume that the loop blocks can be globally placed. Otherwise, the global placer sorts the loop blocks on their area (decreasing), and applies the KAMER-BF online placement method [1] to place the modules. If the global placer fails to place all the loop blocks, the most recently added loop block will be removed and the next candidate loop for hardware implementation is considered. The advantage of using KAMER-BF is that it is extremely fast and of reasonable quality so it can be used in the inner loop of the scheduling.

6. EXPERIMENTAL RESULTS

Although our methods do not target any specific FPGA products available in the market, we have chosen Xilinx's Virtex family [7] as the RFU, and Virtex IP-Cores [7] as the library. We assumed that the configurable component runs at 150MHz (Virtex's grade 5-6 speed) while the CPU runs at 300MHz. Since we target loops for optimization, we used the pipelined version of the operations in the IP library.

Table 1 shows the test programs [6] we used for scheduling. One loop from each program is shown in the table. In the first two rows, mapping the loop bodies onto hardware results in more number of CPU cycles than the software implementation of the code, hence the scheduler chooses not to map the blocks to hardware. In decompress_unroll2 and _unroll3, one and two additional loop iterations of decompress have been unrolled. As can be seen in the table, the scheduler has well utilized the potential parallelism in the "image" code.

7. CONCLUSION AND FUTURE WORK

We proposed an integrated static scheduling and placement method that can compile C programs to hardware/software in minutes. The novelty of our method is in exploiting a two-phase fast placement method as well as congestion-aware scheduling heuristic.

We can improve our compiler in many directions. An example is automatic memory placement (i.e., assigning program arrays to on-chip memory blocks). The way memory accesses are scheduled gravely affects the performance. It also puts restrictions on the placement of the RFUOPs (a loop block should be placed close to the memory blocks it accesses).

	Orig. # cycles	Sched. # cycles	RFUOPs used
diffeq	11	15	In software
decompress	17	20	In software
decompress _unroll2	34	30	2 mult, 4 add, 1 and, 1 reg 1 cmp, 2 shft, 5 mux
decompress _unroll3	51	32	4 mult, 5 add, 1 and, 4 shft, 3 cmp, 11 mux, 3 reg
image	24	10	4 add, 2 shft, 1 reg, 5 mux

Table 1 Original number of cycles and scheduled number of cycles per unrolling of the loops

8. ACKNOWLEDGEMENTS

This work was funded by DARPA under contract number DABT63-97-C-0035.

We would like to thank Alex Z. Ye and Candice McGrew for their helps in providing us with the modified gcc compiler [8] and the interface to our scheduler respectively.

9. REFERENCES

[1] Bazargan, K., and Sarrafzadeh, M., "Fast Online Placement for Reconfigurable Computing Systems", In *IEEE Symp. Field Programmable Custom Computing Machines*, pp. 300-302, 1999.

[2] Callahan, T. J., Chong, P., DeHon, A. and Wawrzynek, J., "Fast Module Mapping and Placement for datapaths in FPGAs", in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 1998.

[3] Gokhale, M. and Schott, B., " Data-Parallel C on a Reconfigurable Logic Array", The Journal of Supercomputing, 9 (3), pp. 291-313, 1995.

[4] Hauck, S., Fry, T. W., Hosler, M. M., and Kao, J. P., " The Chimaera Reconfigurable Functional Unit", In *IEEE Symp. Field Programmable Custom Computing Machines*, pp.87-96, 1997.

[5] Hauzer, J. R., and Wawrzynek, J., "GARP: A MIPS Processor with a Reconfigurable Coprocessor", In *IEEE Symp. Field Programmable Custom Computing Machines*, pp. 12-21, 1997.

[6] Honeywell benchmarks http://www.htc.honeywell.com/projects/acsbench/

[7] http://www.xilinx.com/

[8] Ye, A., Shenoy, N. and Banerjee, P., " A C Compiler for a Processor with a Reconfigurable Functional Unit", to appear in *ACM International Symposium on Field-Programmable Gate Arrays*, 2000.