

HYBRIDTHREADS COMPILER: GENERATION OF APPLICATION SPECIFIC HARDWARE THREAD CORES FROM C

Jim Stevens

Information and Telecommunication Technology Center - The University of Kansas
2335 Irving Hill Road, Lawrence, KS
jstevens@ittc.ku.edu

1. INTRODUCTION

The Hybridthreads system (hthreads) is a platform for hybrid CPU/FPGA chips that allows designers to create multi-threaded applications in which a thread can run in software or as a customized hardware core [2]. The hthreads system provides a framework for creating hardware thread cores that can interact with other hardware and software threads using a Pthreads-like API and shared memory.

Initially, hardware thread implementation required the use of hardware description languages such as VHDL. VHDL's level of abstraction requires the developer to understand RTL design principles and requires a significant amount of time and effort to successfully implement a hardware thread. The ability to migrate a thread from software to hardware without significant effort is desirable because it allows developers to explore more of the design space. Then, only if the effort is needed, the developer can create a hand-coded VHDL thread implementation to achieve more performance. Since the software threads for hthreads are developed in C, generating hardware threads from unmodified C satisfies this goal. Another benefit of compiling C into hardware threads is that software engineers can exploit the abilities of dedicated application specific thread cores.

The hthreads group is developing the Hybridthreads Compiler (HTC) to satisfy the need for a C compiler that can generate hardware threads. Compiling C-like languages to hardware has been studied a number of times [1, 4, 5]. The goal of past projects is different than the goal of HTC because past projects focused on creating and optimizing hardware based co-processors that interleave execution of a single thread with the CPU. Our goal is to compile unmodified C into hardware threads that can run without depending on the CPU, gaining our speedup from physical thread-level parallelism (TLP). We are not attempting to use C as a general-purpose hardware description language. Instead, we use the FPGA with hthreads as just another compiler target architecture.

HTC is divided into two independent modules: HIFGEN and HIF2VHDL. The HIFGEN module is an extension of

GCC that extracts the GIMPLE architecture independent intermediate form from the GCC compilation process [3] and generates a file format we call the Hardware Intermediate Form (HIF). HIF is a simplified linear representation of GIMPLE that includes the information needed to generate a hardware thread core. HIFGEN generates a HIF file for each compiled C file. The HIF2VHDL module then links the HIF files and completes the compilation from HIF to VHDL. More about HTC can be found in [6]. The remainder of this paper focuses on the design and implementation of HIF2VHDL.

2. REPRESENTATION OF C IN VHDL

HIF2VHDL must support four categories of constructs to implement the semantics of C in VHDL. The categories are primitive arithmetic and logical operations, basic control flow operations, a memory model, and a function call model.

Most of C's primitive arithmetic and logical operations are directly represented in synthesizable VHDL. More complicated primitive operations such as integer divide and floating point multiply are supported either by building them in terms of VHDL's available primitives or by instantiating vendor provided IP.

The control flow within a C function is represented by a state machine in the hardware thread. The state machine model supports all of C's control flow constructs including if-else statements, all forms of loops, and unconditional branches (goto, continue, and break). The state machine acts as a custom control unit for the hardware thread because it sequences the operations of the thread's computation and allows the thread to operate without an instruction stream at run-time.

Since a hardware thread core is running within the hthreads system, a global memory system already exists and the hardware thread simply makes bus transactions to read and write to the global main memory. This global memory model allows the hardware thread to communicate with the other hardware threads and with the CPU. The thread's lo-

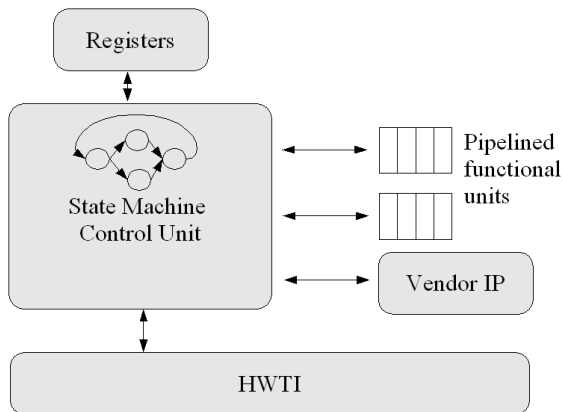


Fig. 1. Hardware Thread Block Diagram

cal memory is implemented efficiently with on-chip BRAM. This allows for very low latency access to local memory. Pointer accesses often cause difficulty when implementing C in hardware [1]. Our memory model removes the difficulty because pointers work exactly like in software.

Hardware threads implement C's function call model by using part of the thread's local memory as a function call stack. The hardware thread function call model uses a calling convention similar to standard calling conventions in commercial processors. The state machine that handles control flow in the thread manipulates the stack to call and return from functions. A side effect of using the standard stack based model in the hardware thread is that it is trivial to implement recursive function calls, which have proved difficult in the past [1, 5].

Figure 1 shows a block diagram of the hardware thread model. The register set and functional units vary depending on the application. The hthreads hardware thread interface (HWTI) contains the BRAM that acts as local memory and a bus connection to access the rest of the hthreads system.

3. HIF2VHDL MODULE

The HIF2VHDL module requires the user to specify the set of .hif files that contain the thread's functions, the thread's start function, and an output file for the resulting VHDL. Using this information, HIF2VHDL parses the .hif files and performs a number of transformations to prepare the thread for hardware. Since HIF2VHDL can control all aspects of the hardware core that implements a thread, we choose to delay VHDL code generation until link time, when all information about a thread is known. Complete knowledge of a thread's computation allows for better decisions about how many registers and functional units to include in the core. HIF2VHDL generates code by instantiating VHDL

templates for each HIF instruction in the control flow graphs of the thread's functions.

An initial prototype of HIF2VHDL has been created as a proof of concept for hardware thread compilation. Combined with the prototype of the HIFGEN module, the prototype of HTC is operational and can compile threads from C to VHDL. The prototype already has the ability to handle constructs in C such as recursive functions and pointers. We have been able to execute compiled hardware threads on a Xilinx Virtex 2 Pro running hthreads. The prototype is currently limited to using 32-bit signed integers and does not support function pointers. The prototype HIF2VHDL performs a call graph pruning optimization to remove functions that cannot be reached from the thread's start function and performs a simple register allocation algorithm. The prototype currently does not perform other optimizations.

4. FUTURE WORK

The next step in HTC development is to complete the compiler's support of ANSI C. After HTC's ANSI C support is mature, the project will focus on optimizing the hardware thread compilation process. We will focus on optimizing the hardware thread compilation process using both traditional compiler optimizations and by researching new optimizations that are specific to our hardware thread model. The ILP and loop optimizations explored in the coprocessor-based hardware compilation systems can also be applied by HTC [4]. Other areas of future investigation include memory coherency for hardware thread cores, applying network-on-chip (NoC) concepts to hthreads, message passing with hthreads, and the use of partial reconfiguration to schedule and load hardware thread cores at run time.

5. REFERENCES

- [1] SPARK Project. <http://mesl.ucsd.edu/spark/>.
- [2] E. Anderson, J. Agron, W. Peck, J. Stevens, F. Baijot, E. Komp, D. Andrews, and R. Sass. Enabling a Uniform Programming Model Across the Software/Hardware Boundary. In *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, California, Apr. 2006.
- [3] Free Software Foundation, Inc. GNU Compiler Collection (GCC) Internals. <http://gcc.gnu.org/onlinedocs/gccint/>.
- [4] Z. Guo, B. Buyukkurt, W. Najjar, and K. Vissers. Optimized Generation of Data-Path from C Codes. In *Proceedings of the ACM/IEEE Design Automation and Test Europe*, Munich, Germany, March 2005.
- [5] D. Lau, O. Pritchard, and P. Molson. Automated Generation of Hardware Accelerators with Direct Memory Access from ANSI/ISO Standard C Functions. In *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, California, Apr. 2006.
- [6] J. Stevens and F. Baijot. Hybridthreads Compiler. <http://www.ittc.ku.edu/hybrdithreads/downloads>. Technical Report.