Overview of ROCCC 2.0

Walid Najjar and Jason Villarreal

SUMMARY

FPGAs have been shown to be powerful platforms for hardware code acceleration. However, their poor programmability is the main impediment to their wider adoption. ROCCC (Riverside Optimizing Compiler for Configurable Computing) is a C to VHDL compilation toolset specifically designed for the efficient and rapid generation of high-performance code accelerators on FPGA platforms.

FPGAs As Hardware Accelerators

Over the past decade or so, FPGAs have become attractive platforms for hardware code acceleration because they provide a good tradeoff between the programmability of CPUs and GPUs and the performance of ASICs. This particularly true for streaming applications, such as signal, image and video processing as well as several high performance computing applications, where a circuit is configured in hardware and the data is streamed through it generating one set of results every cycle. In this same period we have witnessed a tremendous growth in the size of FPGAs and the bandwidth that can be supported in/out of the chip allowing very large and massively parallel circuits to be implemented.

Published papers report speedups ranging from one to four orders of magnitude on a wide range of applications. The primary reasons for such speedups are:

- The massive on-chip parallelism allowing multiple concurrent loop iterations
- Data re-use reduces the re-fetching of data from external storage. The custom configuration of on-chip registers allows the sharing of intermediate values.
- The elimination of memory off-loading: both CPUs and GPUs must fetch data from memory, hence the data has to be loaded into memory first. FPGAs can directly interface to networks, disks and other I/O devices.

However, their programmability remains a major barrier to their wider acceptance by application code developers. FPGAs are commonly programmed using low-level hardware description languages such as VHDL and Verilog. Such languages require complete timing information and low-level details that software designers are traditionally unfamiliar with. They also assume that the application developer is throughly trained as a hardware designer. Hardware implementations may provide critical speedup to software applications, necessitating a way to overcome the long development times and programming overhead normally required to create FPGA implementations.

ROCCC 2.0

The Riverside Optimizing Compiler for Configurable Computing (ROCCC) raises the level of abstraction for programming FPGAs to a subset of C, allowing developers to translate their code to an FPGA platform while remaining in a familiar programming language. ROCCC is *not* designed as a hardware description language and cannot be used to describe *arbitrary circuits* in C. Instead, the focus of ROCCC is on generating high performance hardware accelerators, from natural C code, that perform a wide range of computations on streams of data.

Through extensive optimizations, ROCCC transforms this subset of C into VHDL code that achieves very high throughput. The purpose of these transformations is to generate circuits that (1) maximize the throughput, (2) minimize the number of off-chip memory accesses, and (3) minimize the area used on the FPGA. The user is given fine-grained control over the extent of these transformations through an elaborate GUI developed as an Eclipse plug-in.

FPGA boards and platforms have widely varying characteristics such type, size and number of FPGAs, number of memory modules, bandwidth to memory, I/O interfaces (e.g. Ethernet, USB, PCI, SATA etc). Applications must be tuned to take advantage of the platform specifics, and the tuning traditionally would involve rewriting the VHDL to best utilize the particular resources of a given platform. ROCCC allows the user to tune a large number of optimizations to better utilize the particulars of a given platform without rewriting the source code. One of the most powerful controls that ROCCC supports is through the control of loops. Users may specify exactly which loops are unrolled, and by what amount, unlike in a traditional compiler where all of these decisions are made across the whole program. Unrolling will adjust the number of data elements required per loop iteration, which in turn affects the bandwidth required for maximum throughput. Additionally, the user can control the number of incoming channels on a stream-by-stream basis, ensuring that incoming and outgoing data rates are maximized.

In ROCCC, the same source code can be tuned for different FPGA platforms by varying the compiler optimizations in the GUI, without altering the original code.

ROCCC also provides optimizations that are specific to hardware platforms, including systolic array generation, temporal common subexpression elimination, and several low-level optimizations that control the amount of pipelining and fanout.

Design Space Exploration and Productivity

The ability to program FPGAs in a higher level language and through tuning of optimizations create many different physical implementations not only greatly increases user productivity, but also enables design space exploration which might have previously been out of reach due to time concerns.

One of the most critical features of a design environment is the ability to reuse sections of code. The ROCCC 2.0 compiler is built around the concept of reuse through the creation and integration of modules. Modules are concrete hardware blocks that have a known number of inputs and outputs, some internal computation, and a known latency. Modules written in C can be reused as both software functions or hardware functions and integrated directly into larger designs. Once compiled to VHDL, modules are available for integration in larger code through standard C function calls.

The same mechanism which allows the ROCCC compiler to reuse previously compiled modules allows for the importing of other VHDL codes or hardware cores into larger designs. ROCCC system code can call hardware modules, including those imported from other locations, and generates a hardware component that integrates these external cores into one cohesive data-path, handling all of the connections and overhead.

The ROCCC GUI is a plugin designed for the Eclipse IDE that works on both Linux and Mac systems. The GUI provides the user with full control over the programming, compilation, and interfacing of the ROCCC code. It provides an integrated way to manage the instantiation of modules and cores into C code, control both high and low level transformations (including pipelining), interface with platforms, and generate test-benches for verification.

An example

The Viola Jones Face Detection algorithm detects the existence of faces in an image by searching the sub-windows of that image and running a multi-stage classifier cascade agains that sub-window. Each stage of the classifier cascade can detect the presence of faces with a low rate of false negatives and false positives. Each stage of the classifier is more rigorous than the previous one and hence require more computations. The serial cascading of stages makes sense in a software execution since it allows the rapid elimination of sub-windows with no faces. In hardware, the area is allocated on the FPGA for all stages, conceptually they could all be executed in parallel. ROCCC code that implements the Viola Jones algorithm automatically detects this parallelism and implements a parallel data-path with a high throughput.

The ROCCC implementation of 5 stages of the classifier takes 587 lines of modularly written C code and generate 14,321 lines of VHDL, a ratio of 24.4. Assuming a bloating factor of 2x, because the VHDL code is machine generated, this translates in a 12x productivity increase. The resulting circuit can process one sub-window per cycle and achieves 150 MHz.

ROCCC Optimizations

In addition to providing an environment that allows the development of ROCCC modules and systems, the ROCCC GUI provides access to all of the compiler's controls. As such, the GUI gives the user control over all of the specific high and low level optimizations and allows the user to select how to apply these optimizations on a much finer-grained level than provided by other compilers.

High level optimizations

High level optimizations control transformations that affect the overall structure of the generated hardware architecture. These include loop optimizations such as unrolling, inlining optimizations, and redundancy specification. Unlike traditional compilers, each of these optimizations can be specified on individual elements of the ROCCC code and applied in different configurations so as

to produce different hardware architectures. For example, loop unrolling can be specified on a loop-by-loop basis, with different loops being unrolled different amounts in order to generate a hardware structure that uses an amount of data that can be supported by the I/O of the target hardware platform. Similarly, inlining and triple modular redundancy can be specified differently for different individual modules used in the design.

Low level optimizations

Low level optimizations give users control over the specifics of the generated data-paths. As an example, fine-grained control over the amount of pipelining of the data-path: a slider controls the amount of operations placed into each pipeline stage providing a convenient control over whether frequency or area is the limiting factor. Other low level optimizations control whether arithmetic instructions are balanced or not (important when matching software results with floating point operations) and precision control that can limit rounding issues.

Different hardware platforms have completely different interfaces to memory, including different bit-widths and multiple channels. In order to allow ROCCC generated code to be general, we do not target any one specific platform but instead give control of the memory interface generation to the user. The ROCCC GUI allows the user to specify the number of channels and bit width on a stream-by-stream basis without altering the ROCCC C code so the generated hardware can best connect to a multitude of different hardware platforms.

The ROCCC GUI provides access to the database of modules available for use in ROCCC development. These modules may have been previously compiled with ROCCC or can be imported from other sources in the GUI, allowing ROCCC code to utilize IP cores or HDL code that was acquired from another source. Once the user specifies the input and output ports of an external IP along with its latency, the ROCCC compiler will use them as appropriate. Instantiations of these available modules appear as function calls in the ROCCC C code.

Certain operations in C do not have native hardware capable of computing them on certain FPGAs, and any design that requires them must include extra hardware cores to compensate. These operations include floating point arithmetic, integer division, and integer to floating point translation. Different hardware implementations of these functions will lead to different size and speed characteristics of the final design, and the choice should be left to the user and not the compiler. The ROCCC GUI exports control of these intrinsic operations to the user and allows data-paths to be generated with a variety of cores, providing control for activating and deactivating individual intrinsics, along with a set of default parameters.

Mapping to FPGA platforms

ROCCC users have the option of generating “generic” code or platform specific codes. In a generic code the data is transferred to/from the circuit via on-chip buffers (BRAM) that can later be connected by the user to a specific set of memories and/or I/O ports. The connections generated

are either scalar inputs or outputs that are read or written only once, or streaming connections that have a simple memory interconnect.

The ROCCC GUI supports the generation of platform specific code for the Convey Computers HC series and the Pico Computing M-500 series.

In addition to generating portable hardware, the ROCCC GUI also facilitates the creation of a test-bench for each compiled component. The generation of a test-bench requires sample input and output data in order to verify functionality. Since all ROCCC code is also a subset of C, this data can be generated by compiling the ROCCC code in a C compiler, e.g. gcc, and running it in software.

Genesis of ROCCC

ROCCC started as a research project at UC Riverside in 2002. Dr. Najjar had previously run the Cameron Project (at Colorado State University, funded by DARPA within the ARS Program). In Cameron, a new language was developed, single assignment C, along with a compiler. The objective of ROCCC was to compile a true subset of C.

Jacquard Computing was founded to develop ROCCC as a commercial grade tool. I was funded by AFRL through SBIR Phase 1 and Phase 2 (Contract FA9453-09-C-0173).