Carte++: An LLVM Based Compiler Targeting FPGAs
Authors: Lisa Krause, Matt O’Connor, Jon Steidel, and Jeffrey Hammes
SRC Computers, LLC

C/C++ Source Language
- Previous compiler allowed a restricted subset of C
- Clang++ provides us with C and C++
- Goal is to implement as much of C as possible
- Templates in C++ offer a clean way to express variable width data streams with a single interface
- Not yet supported:
  - Recursion
  - C & C++ standard library
  - Variable length arrays (VLAs)
  - Unrestricted function pointers

Why is Compiling C++ to FPGAs Hard?
- Unconstrained by fixed ISA
- Compiler must bridge wide gulf between C++ and digital logic gates
- Clock frequencies are typically in hundreds of MHz, so performance must come from parallelism
- FPGAs have incredible parallelism (everything can execute on every clock tick) but C++ is sequential

Compilation Modes
- Pure CPU mode:
  - Compiled with Clang++/LLVM on CPU executable
  - No AIF resources
  - Provides a fast development environment running entirely on the cpu
- Simulation mode:
  - Compiled with Carte++ is instantiated with Verilog modules, which are then executed by avrgr65 simulators
  - No AIF resources
  - Allows testing performance analysis of code compared to AIF hardware
- Production mode:
  - Compiled with Carte++ is instantiated with Verilog modules then plases into Altera Stratix or FPGA HW, which is connected with the CPU via the AIF interface
  - Execution of the AIF code is the same as the simulation code
  - Compile time on CPU and AIF hardware

Anatomy of a Program
- Each MAP function goes in its own file
- Then during compilation, Carte++ generates a wrapper function around the MAP function that sets up and transfers control to the FPGA
- This makes the execution of a MAP function call look like any other function call

LLVM IR to Dataflow Graph
- LLVM IR is transformed to dataflow nodes
- SSA representation makes transformation easier
- Each node instantiated as an independent Verilog module

Hand-Written Verilog Components
- The library of Verilog modules is hand-crafted for performance and efficient FPGA routing
- Two kinds of Verilog modules:
  - Functional units (e.g., float adder, etc.)
  - Infrastructure components (e.g., direct connect control flow, etc.)
- The functional units correspond to LLVM IR instructions
- The infrastructure components determine what kind of concurrency can take place (concurrent basic blocks, pipelined loops, etc.)
- The compiler’s job is to instantiate modules from this library and interconnect them according to the dataflow graph

Parallelism
- Implicit
  - Overlapping instructions
  - Overlapping basic blocks
  - Simultaneous accesses to different memories
  - Loop pipelining
- Explicit
  - Threading via Pthreads
  - Data streams allow users to asynchronously connect loop pipelines

Data Streams
- A thread-safe queue
- Provides a clear way to communicate data between threads
- Allows simultaneous loads and stores
- Can produce a data value on every clock tick to support loop pipelining
- Flow control allows a stream’s consumer and producer to be loosely-coupled, i.e., they can accept or produce data values at their own speeds with a buffer in the FPGA coupling them

Future Work
- Pipelining loops
- Space optimizations
- More complete Pthread implementation
- Better memory allocation/deallocation support
- Components of the C & C++ standard libraries
- Other languages

Carte++ MAP Compilation Process
- Clang++/LLVM
- Pass and assemble
- AIF compiler
- FPGA
- Verilog
- Simulation

Code Example - MAP Functions

Pointer & Memory Analysis
- Goal is to connect loads/stores only to potentially accessed memories
- Requires pointer tracing and analysis
- Requires visibility into the entire program being compiled
- Statically assign an address range for each external and local memory
- LLVM IR and supporting infrastructure (e.g., Value) simplifies the analysis