Carte++: An LLVM Based Compiler Targeting FPGAs

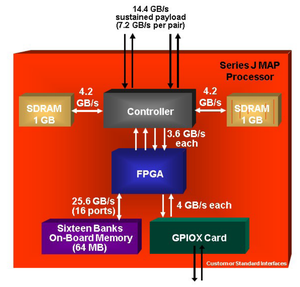
Authors: Lisa Krause, Matt O'Connor, Jon Steidel, and Jeffrey Hammes

SRC Computers, LLC

SRC Computers, LLC

- Founded by Seymour Cray in 1996
  - Creation of a new, highly efficient computer architecture
  - Reconfigurable processor that adapts to the needs of the application.
- History of successful execution
  - Invention and deployment of innovative computer systems
  - Three generations of SRC® reconfigurable systems
  - Twelve generations of SRC reconfigurable MAP® processors

SRC Series J MAP Processor



What is an FPGA?

- Field Programmable Gate Arrays
  - In essence, a digital logic "blank canvas"
  - A large mesh of configurable logic elements, memories, and interconnect system
  - "Programmed" using hardware description languages (HDLs) such as Verilog and VHDL
  - Takes hours to place and route, producing a bitstream
  - Takes milliseconds to configure a FPGA chip at runtime with an existing bitstream

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?

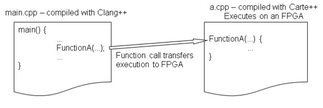
- Unconstrained by fixed ISA
- Compiler must bridge wide gulf between C++ and digital logic gates
- Clock frequencies are typically in hundreds of MHz, so performance must come from parallelism
- FPGAs have incredible parallelism (everything can execute on every clock tick) but C++ is sequential

Compilation Modes

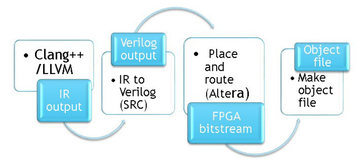
- Pure CPU mode:
  - Compiled with Clang++/LLVM to CPU executable
  - No MAP resources
  - Provides a fast development environment running entirely on the cpu
- Simulation mode:
  - Compiled with Carte++ to instantiations of Verilog modules, which are then executed by Verilog simulator.
  - No MAP resources
  - Allows easier performance analysis of code compared to MAP hardware
- Production mode:
  - Compiled with Carte++ to instantiations of Verilog modules then Altera place-and-route tools, creating a MAP bitstream which is combined with the CPU object files to create a unified executable
  - Executes on CPU and MAP 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



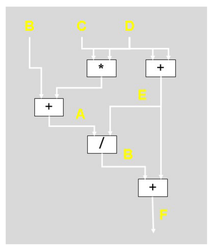
Carte++ MAP Compilation Process



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

```
A = B + C * D;
E = C + D;
B = A / E;
F = B + E;
```



Hand-Written Verilog Components

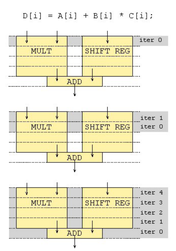
- The library of Verilog modules is hand-crafted for performance and efficient FPGA routing
- Two kinds of Verilog modules:
  - Functional units (e.g. float adder, etc.)
  - Infrastructure components (e.g. directs control flow, etc.)
- The functional units correspond to LLVM IR instructions
- The infrastructure 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

Carte++ Pipelining

- Hardware pipelining uses pipelined Verilog modules
- On each clock tick, a new iteration's values are dropped in
- In this example, the MULT operation takes four clocks, and the Add takes one
- The four-stage shift register is needed for path balancing
- Results begin to appear on the output after five clock ticks



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

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

Code Example - MAP Functions

```
static int64_t Sum, N;

void subr(const uint64_t InN, int64_t *Result,
int _MapNum) {

 Sum = 0;
 N = InN;

 stream<bits288> S;

 pthread_t T0, T1;

 pthread_create(&T0, 0, writer, &S);
 pthread_create(&T1, 0, reader, &S);

 pthread_join(T0, 0);
 pthread_join(T1, 0);

 *Result = Sum;
}
```

```
void *writer(void *arg) {
 stream<bits288> *s = static_cast<stream<bits288> *>(arg);
 int64_t WriterSum = 0;
 for (int64_t i = 0; i < N; ++i) {
  bits288 D = { i, i * 2, i * 3, i * 4, -1};
  WriterSum += D[0] + D[1] + D[2] + D[3];
  S->put(D);
 }
 S->put({0, 0, 0, 0, 0});
 return 0;
}
void *reader(void *arg) {
 stream<bits288> *s = static_cast<stream<bits288> *>(arg);
 int64_t ReaderSum = 0;
 bits288 D = S->get();
 while (D[4]) {
  ReaderSum += D[0] + D[1] + D[2] + D[3];
  D = S->get();
 }
 Sum = ReaderSum;
 return 0;
}
```


Future Work

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

SRC®
SRC COMPUTERS, LLC