

ECE 497NC: Unconventional Computer Architecture

Lecture 10: Reconfigurable Computing 5: Compilation

Challenges

- **Language Support**
 - Standard FPGA tools operate on Verilog/VHDL
 - Programmers want to write in C
- **Compilation Time**
 - Traditional FPGA synthesis often takes hours/days
 - Need compilation time closer to compiling for conventional computers
- **Programmable-Reconfigurable Processors**
 - Compiler needs to divide computation between programmable and reconfigurable resources
- **Non-uniform target architecture**
 - Much more variance between reconfigurable architectures than current programmable ones

Why Compiling C is Hard

- General Language
- Not Designed For Describing Hardware
- Features that Make Analysis Hard
 - Pointers

Approach 1: DIL Compiler for PipeRench

- **Simplify the Problem**
 - Single-assignment language
 - Closer to HDLs than C
 - Compile for PipeRench, an architecture optimized for multi-bit computations
 - Base block is >1 bit
 - Support for virtualizing large computations on smaller pipelines
- **Fast Compilation**
 - > 3000 operations/second
 - Trade some solution quality for compile time
- **Good Utilization**
 - Over 60% of hardware resources used
 - Over 55% of hardware resources used for useful computation

Approach

- Two Phases
 - Map program to reconfigurable logic elements
 - Place elements within array
- Mapping Program to Logic
 - Convert program to abstract syntax tree
 - Reasonably straightforward to convert syntax tree to basic operations
- Optimizations
 - Conventional compiler optimizations
 - Bit-width inference

Place and Route

- Traditionally a big consumer of time
 - Algorithms written for best solution quality
 - Many passes
 - Placement iterated to find best match
- Use greedy approach instead
 - Block not moved once placed
 - Greatly reduces placement time at some cost in quality
 - Requires guarantees of routability
- Ensure graph routable before starting placement
 - “Lazy NOPs” added to simplify difficult wiring
 - Placer only instantiates NOP when absolutely necessary

Garp Compiler

- Automatic compilation of complete C programs to Garp
 - Extract regions for reconfigurable execution
 - Use programmable processor to execute code they can't map to reconfigurable
 - Subroutine calls
 - Floating-point
 - Division/remainder
 - Manage communication between programmable processor and array
- We'll look at two aspects
 - Selection of regions for reconfigurable array
 - Mapping computation to modules

Selecting Regions for Array

- Problems
 - Branches
 - Unimplementable operations
 - Lack of parallelism
- Approach
 - Identify loops
 - Use hyperblock structure to convert most-common path into straight-line code with predication
 - Exclude basic blocks that perform unimplementable operations
 - Merge loopback branches to allow multiple iterations of loop to execute on array
- Advantages
 - Unimplementable operations often off of common path
 - Creates parallelism by removing control dependencies

Datapath Synthesis

- Conventional approach
 - Convert design into gate-level primitives
 - Map primitives onto hardware resources
- Problems
 - Long run time (many objects to place)
 - Loss of high-level information (e.g. carry chains)
- Other approach
 - Map design to pre-optimized modules
 - Allows use of high-level information
 - No opportunity for cross-module optimization

Gama Approach

- Start with graph-based representation at higher level
- Convert graph to “forest of trees”
- Iterate across nodes of trees, finding modules from library
 - Modules can span multiple nodes
 - Modules can represent many gates
- Convert modules into row structure
- Map row structure to hardware
 - Much faster than conventional synthesis
 - Requires fewer modules than traditional approaches
 - Yields faster critical path
- Note: not guaranteed to give optimal results

Parallelizing Applications Into Silicon

- View:
 - Delay of large memories is becoming critical
 - Communication time needs to be explicitly modeled in programs
- Hardware Assumptions:
 - System contains many small memories rather than one large one
 - Some processing resource associated with each memory (either CPU or reconf. logic)
 - Overall architecture divided into *tiles*, where the size of a tile is bounded by the wire length that can be traversed in a clock cycle.
 - Tiles are connected by pipelined wires
 - *Virtual Wires*: technique for multiplexing multiple communication streams across a single pipelined wire

High-Level Approach

- Divide data structures into *alias equivalence classes*, such that no pointer references data in multiple equivalence classes
- Assign equivalence classes to memories
 - Model restriction: no memory operation can access data in more than one memory.
- Use unrolling to distribute equivalence classes across memories, maintaining the restriction that no memory operation can access more than one memory
- Distribute program instructions across processing units
 - Any load or store must be executed on the unit associated with the memory containing the data it references.
 - Other operations assigned with goal of minimizing communication.
- Statically schedule communication between memories

High-Level Approach

- Convert instructions on each processing unit into control FSM and datapath
- Generate RTL language description of FSM and datapath on each node for synthesis.

- Apply customizing optimizations
 - Fold constants into logic
 - Use bit concatenation to eliminate address calculations for array elements.
 - Convert floating-point operations into sequences of micro-instructions that can be optimized.

Results

- Assuming equivalent clock rates, see speedups of 5-10x over designs that have a programmable processor at each memory
 - Equivalent clock rates probably feasible with ASIC design, less feasible with reconfigurable logic
- For applications they were able to parallelize, see better performance and similar rates of scaling to processor-based architectures.
- Use of virtual wires imposes significant hardware and performance costs compared to custom hardwired, but would allow one chip to implement multiple operations.