A High Level Programming Model for Large-Scale Reconfigurable Computing Platforms

Corey Olson, Elliott Broussard

Faculty: Carl Ebeling, Scott Hauck, Larry Ruzzo
System Model
FPGA-Accelerated Computing Platforms

- Construct specialized computing structures that fit the application
  - e.g. systolic arrays, specialized memory structures,…

- Performance advantage
  - Extensive fine-scale parallelism in a spatial architecture

- Power advantage
  - No overhead for instruction fetch, decode, issue
Programming FPGA Accelerators

• Reasonably good tools for module level
• Very difficult to construct large applications
• Lack of high-level programming models

This is the problem we want to solve
System Model
Programming Model Overview

- A system is comprised of Objects
- Objects are created and connected at compile time
- Objects have
  - Data/State that object manages
    - e.g. memory, variables...
  - Methods
    - Manipulate the data, perform computation

```c
int accum;
void init() {
    accum = 0;
}
void sum(int value) {
    accum += value;
}
```
Methods

• Modules interact via method calls
  – Send parameters from source method to destination object
• Asynchronous methods
  – No return – caller continues immediately
  – Almost all methods asynchronous
• Synchronous methods
  – Caller blocks until method completes (return value)
Implementing Method Calls

Local
- Point-to-point connection with wires
- Standard way to implement hardware

Remote
- Communicating objects connected by network
- Remote procedure calls (RPCs)
- Synchronous calls require return message

Diagram:
- Object 1
- Network
- Object 2
- Object 1
- Object 2
int numCALs;
char read [READLENGTH];

init(){
    numCALs = 0;
}

nextRead(long readID, char read [READLENGTH], int numCALs){
    this.numCALs = numCALs;
    this.read = read;
    SWCollect::numCALs(readID,numCALs);
}

nextCAL(long readID, long CAL){
    numCALs--;
    Reference::nextCAL(readID,CAL);
    SW::nextRead(readID,CAL,read);
}
Recall Algorithm

1) Break Read into subsequences and find locations in reference genome

Precompiled Index Table

Reference

47 174 822
47 822
133 722
722

Filter

4) Compare Read to the Reference at all the Candidate Alignment Locations using Smith-Waterman alignment

47
174
822
722
133

3) Filter the CALs to remove repeats

822
174
722
47
133
Now with Object Model
Simple Solution
Dynamic Objects

• Automatic allocation
  – Object is allocated when method with new ID is called
  – All calls with same ID go to the same object
  – Object deallocates itself when done

• Implementation
  – Pool of objects (static)
  – Object manager
    • Allocates and maps objects
    • Directs method calls to appropriate object
More Parallelism
Distributed Objects

- Partitioned memory arrays
  - Concurrent accesses
  - Higher data bandwidth
- Partitioning memory causes associated objects to be copied and distributed with partitions
- Method calls automatically directed to right partition
Distributed Solution
Summary

- FPGAs great benefit to data hungry applications
  - Power efficient
  - Massively parallel
- Write applications using object model
  - Compile to a distributed accelerator system
  - Programmer describes the parallelism in the system
  - Compiler handles memory partitioning, object duplication and distribution, and communication between objects
Questions?

• Visit us at the poster session in:

CSE 407
5-8pm

Corey Olson, Elliott Broussard

Faculty:
Carl Ebeling and Scott Hauck
Object Model Solution – Phase 1

- **Reference Object**
  - Read reference sequence from memory
  - Construct seeds
  - Send seeds to Index Table
  - Distribute Reference over M locales
    - Generate seeds in parallel

- **Index Table Object**
  - Build Index Table
  - Distribute Index Table over N locales
    - Handle calls to table in parallel
Object Model Solution – Phase 2

- **Reads Object**
  - For each Read, generate seeds and send to Index Table Object
  - Send number of Reads to Filter Object

- **Index Table Object**
  - Look up seed in table
  - Send number of positions to Filter
  - Send positions to Filter Object

- **Filter Object**
  - Collect and counts positions for a read ID
  - Send CALs to SWDispatch Object when all positions have been received
Object Model Solution – Phase 2 cont’d

• **SWDispatch Object**
  – Allocate SW unit for each read ID, CAL pair

• **Smith-Waterman Object**
  – Initialized with read and reference
  – Operate in parallel
  – Send score to SW Collect Object

• **SW Collect Object**
  – Gather all scores for a read
  – Report top score to host
Compiling System Descriptions

• Implementing Hardware Objects

• Compiling Method Calls
  – Sender and receiver interface

• Implementing Dynamic Objects
  – Efficient handling of concurrent calls

• Implementing Distributed Objects
  – Memory objects
  – Broadcast methods

• Controlling work creation

• Debugging and Performance Monitoring
Compiling Method Calls

• Using Wired or Network Connection
  – Wires
    • Static call – callee object known at compile time
    • “Local” call – direct wires available
      – Same FPGA/neighbor FPGA
  – Network
    • Distributed object method calls
    • Efficient use of bandwidth – streaming data

• Efficient receiver interface
  – FIFOs
  – Exclusive calls
  – Streaming parameters
Implementing Hardware Objects

- Prototype objects using C++/Java
- Serves as specification for hardware implementation
- Options for generating HW:
  - Manually write HDL
    - Optimized implementations
    - Standard interface – Reusable modules
  - C-to-Hardware compiler
    - Straight from spec to hardware
  - HDL Synthesis tools
    - Translate C to HDL
    - Constrains input language
Implementing Dynamic Objects

- Mapping object IDs
- Efficient allocation and deallocation of objects
- Efficient handling of concurrent calls
  - Multiple concurrent calls to multiple objects
  - Concurrent call forwarding
Implementing Distributed Objects

- Memory objects
  - Partitioning and mapping the address space
- Memory replication
  - Full and partial
    - e.g. stencil computations
  - Sharing and coherency
- Distributed object method calls
  - Determining whether static and/or local
- Broadcast method calls
Reductions and Synchronization

• Simple and Prefix Reductions
  – Programmer provides function
  – Compiler generates reduction over distributed object set

• Synchronization:
  – Reference counting
    • Count work as it is created and as it is completed
      – Programmer implements
  – Barrier synchronization
    • AND reduction and broadcast over set of objects
Controlling Work Creation

- Creating work faster than it can be done clogs the network
- Rate limiting
  - Simple open-loop
  - Works only for simple cases
- Network-level flow-control
  - Messages allowed only if there is room at the receiver
- Application-level flow-control
  - Application keeps track of work as it is created and completed
  - Credit-based flow control based on buffer sizes
    - Compiler-supplied constants
Smith-Waterman Units

- Reads that match perfectly don’t need SW alignment
- CALs grouped into a bucket to allow for gaps
- Reads compared against more than one bucket