Explicit Loop Specialization

&

Polymorphic Hardware Specialization

Christopher Batten and Zhiru Zhang

Computer Systems Laboratory
School of Electrical and Computer Engineering
Cornell University

June 2015
Explicit Loop Specialization & Polymorphic Hardware Specialization

- Energy Efficiency (Tasks per Joule)
- Performance (Tasks per Second)

- Embedded Architectures
- Programmable Accel
- General Purpose Processor
- High-Performance Architectures

Design Performance Constraint
App Specific w/ HLS
Flexibility vs. Specialization
Design Power Constraint

Programmable Accel
void ordered_merge( int* out, int* in_0, int* end_0,
                   int* in_1, int* end_1 ) {
    while ( (in_0 != end_0) && (in_1 != end_1) ) {
        *out++ = ( *in_0 < *in_1 ) ? *in_0 : *in_1;
        ++in_0; ++in_1;
    }
}

Programmable Accelerator

Application-Specific HLS
XLOOPS: Explicit Loop Specialization [MICRO’14]

```c
#pragma xloops unordered
for ( i=0; i<N; i++ )
  C[i] = A[i] * B[i]
loop:
  lw       r2, 0(rA)
  lw       r3, 0(rB)
  mul      r4, r2, r3
  sw       r4, 0(rC)
  addiu.xi rA, 4
  addiu.xi rB, 4
  addiu.xi rC, 4
  addiu     r1, r1, 1
  xloop.uc r1, rN, loop
```

- Unordered Atomic
- Ordered-Through-Registers
- Ordered-Through-Memory
- Fixed vs Dynamic Bound
XLOOPS Energy-Efficiency vs. Performance Results

- XLOOPS vs. Simple Core: Similar energy efficiency, higher power
- XLOOPS vs. OOO 2-way: Higher energy efficiency, mixed power
- XLOOPS vs. OOO 4-way: Higher energy efficiency, lower power
- Adaptive execution trades energy efficiency for performance
- Profiling and migration cause minimal performance degradation
Software engineers also want to create specialized yet flexible pieces of software to improve code efficiency and reduce design complexity.

Software engineers develop carefully crafted libraries of algorithms and data structures that are composable and polymorphic over the types of values and/or functors.

template < typename Itr0, typename Itr1, typename Itr2, typename Cmp >
void ordered_merge( Itr0 out, Itr1 in_0, Itr1 end_0,
                   Itr2 in_1, Itr2 end_1, Cmp cmp ) { 
  while ( (in_0 != end_0) & (in_1 != end_1) ) {
    *out++ = ( cmp( *in_0, *in_1 ) ) ? *in_0 : *in_1;
    ++in_0; ++in_1;
  }
}

How can we systematically (and automatically?) generate hardware specialization at design time that supports compile-time polymorphism?
PolyHS Methodology

Poly-HS Synthesis

Software Stubs

Library of RTL for Poly-ASUs/DSUs

Library of Algorithms and Data Structures

code

HW Toolflow

SW Toolflow

Poly-HS Architecture

Full-Chip RTL and Simulators

Standard ASIC CAD Toolflow

Poly-HS Compilation

Application Binary

Poly-HS Run-Time System

Poly-HS Chip

Application code

SW Toolflow

Poly-HS Chip
Carefully designed iterator-based abstraction enables composition of algorithm and data-structure specialization units

Specialization units configured with metadata describing data types and functors
Explicit Loop Specialization & Polymorphic Hardware Specialization

- Design Performance Constraint
- Embedded Architectures
- PolyHS
- Programmable Accel
- App Specific w/ HLS

- Flexibility vs. Specialization
- Design Power Constraint

- Energy Efficiency (Tasks per Joule)
- Performance (Tasks per Second)

General Purpose Processor

High-Performance Architectures

Embedded Architectures

Programmable Accel

App Specific w/ HLS

PolyHS

Cornell University

Christopher Batten