Christopher Batten
Assistant Professor
Computer Systems Laboratory
School of Electrical and Computer Engineering
Cornell University

Research Interests: energy-efficient parallel computer architecture; parallel programming methodologies; hardware specialization; interconnection networks; VLSI chip-design methodologies

2010 : PhD @ MIT
2000 : MPhil @ University of Cambridge
1999 : BS @ University of Virginia
Rapid Exploration of Accelerator-Rich Architectures: Automation from Concept to Prototyping

- Performance (Tasks per Second)
- Energy Efficiency (Tasks per Joule)
- General Purpose Processor
- Embedded Architectures
- Programmable Accel
- Composable Accel
- App Specific Accel

Design Power Constraint vs. Specialization

Flexibility vs. Specialization

General Purpose Processor
High-Performance Architectures
Rapid Exploration of Accelerator-Rich Architectures: Automation from Concept to Prototyping

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

General Purpose Processor
Embedded Architectures
High-Performance Architectures

Design Power Constraint
Flexibility vs. Specialization

App Specific Accel
PolyHS
XLOOPS

Programmable Accel
Composable Accel

Cornell University Christopher Batten
#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
Cycle-Level Evaluation

Normalized Energy Efficiency

Normalized Performance

Normalized Performance

OOO 2-way + LPSU vs. OOO 2-Way

gem5 + PyMTL + McPAT

XLOOPS
RTL/VLSI Evaluation

In-Order + LPSU

Verilog RTL + Synopsys EDA Toolflow

TSMC 40nm
PolyHS: Polymorphic Hardware Specialization

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.

```cpp
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?

Joint work with Prof. Zhiru Zhang @ Cornell University

[ MICRO 2014 ]
https://github.com/cornell-brg/pymtl

Pydgin: Generating Fast Instruction Set Simulators from Simple Architecture Descriptions with Meta-Tracing JIT Compilers

[ ISPASS 2015 ]
https://github.com/cornell-brg/pydgin
1. What accelerators have you designed or plan to design?
   - XLOOPS: Explicit loop specialization
   - PolyHS: Polymorphic hardware specialization

2. What is the process to select kernels for acceleration?
   - XLOOPS: Focus on challenging mix of regular and irregular loops
   - PolyHS: Focus on common template libraries of algos and data structures

3. How do you estimate the acceleration potential?

4. What is your methodology for accelerator design?

5. How do you validate your accelerator design?
   - Multi-level modeling using PyMTL/Pydgin: FL, CL, RTL
   - Not clear application-specific accelerator estimation methodologies can apply in general to programmable and composable accelerators?
   - Currently only focusing on decoupled in-core accelerators
   - Powerful unified framework for test-driven design