Catapult the Masses

James Larus
EPFL

Reconfigurable Computing for the Masses, Really?
A Workshop at and after FPL'15, 4th September 2015
My Perspective

• Masses == software developers
• Reconfigurable computing == FPGA

• Can SDEs program FPGAs without learning HW design or getting an EE degree?

• Can high-level programming languages be compiled down to FPGAs?
  • Not hardware description languages

• Can reconfigurable computing be made as easy as GPU programming?
Semantic Gap
Alternative View of Microsoft Catapult System

• Another way of telling the same story
• Design principles from this story suggest an alternative approach
Microsoft Catapult

Better: Use Programmable Accelerators

Eric Chung, *Programming a Reconfigurable Fabric for Large Scale Datacenter Services*
Accelerators == Non von Neumann Computers (NonvoN)

• Massively parallel
• Not general purpose
  • Not Turing complete (non-Turing)
• Instructions ≠ data

Catapult
• Simple to program directly from Bing language model
• Quickly reprogrammable as search model evolved
• “Easy” to implement
• High throughput at low clock speed
Catapult Feature Extractor

- 196 feature families
- 54 state machines
- 2.6K dynamic features extracted in less than 4us (~600us in SW)

Adrian Caulfield, A Reconfigurable Fabric for Large Scale Datacenter Services
Catapult Free-Form Expressions

Programmable FFE Soft Cores

- Soft processor for multi-threaded throughput
- 4 HW threads per core
- 6 cores share a complex ALU
- log, divide, exp, float/int conversions
- 10 clusters (240 HW threads) per FPGA

Eric Chung, Programming a Reconfigurable Fabric for Large Scale Datacenter Services
Seven representative ML techniques

- k-means
- k-nearest neighbors
- naive bayes
- support vector machine
- linear regression
- classification tree
- deep neural network
Why Was NonvoN Architecture a Good Idea?

• Small compiler-HW semantic gap
  • Some compilers (SM) could have been perl scripts
  • Others (FFE) were sophisticated (~llvm) compilers

• HW was easy to get right and to extend
  • Simple, regular, modular
  • Can track software evolution

• Computations were fine-grain parallel, HW effective at exploiting

• Easy to compose computations in a pipeline

• “Soft” programmability for alternative language models
  • < 200 ms
Limitations

• Lack of generality
  • Will not work as well when
    • No parallel implementation
    • Complex HW (eg GPU)
    • Too sophisticated compilation / programming model (eg GPU)

• Interpretation overhead
  • Probably could do better with ‘pure’ HW implementation
  • But, Bing language models change every 3 months

• Still requires HW designer to implement processors
  • One-time expense, primitives change rarely
  • More importantly, another topic for research
Vision

Domain-Specific Language Specification

Compiler Compiler

Compiler

Domain-Specific Program

FPGA

Domain-Specific Processor Specification

Compiler Compiler

FPGA Implementation
Open Problems

• High-level description of domain-specific languages (DSL)
  • Currently, DSL (mostly) described by imperative implementation

• Declarative techniques for implementing HL DSLs
  • Current, DSL implemented by writing compiler and optimizer (using framework)

• High-level description of domain-specific processors (DSProc)
  • Processor description is an old idea. Time to revive?
  • Possible to derive DSProc from DSL?

• Techniques for implementation HL DSPros
  • Processor compiler?

• Methodology for analyzing domain, designing DSL, co-designing DSProc
LMS: Program Generation and Embedded Compilers in Scala

- Used to build DSL like Delite, Spiral, LegoBase
  - DSLs are concise and expressive
  - Constructing a DSL is still complex and requires compiler expertise
- Type-directed meta/macro programming

```scala
var n: Double = 0.0
var i: Int = 0
val end = data.length
while (i < end) {
  val x = data(i)
  val c = x > 0
  if (c) n += x }
println(n)
```
Putting on Compiler Hat

• High-level description of domain-specific languages ✔
• Declarative techniques for implementing HL DSLs
• High-level description of domain-specific processors
• Techniques for implementation HL DSProcs
Programmer’s
Compiler Writer’s Nightmare