I-USHER: Interfaces to Unlock the Specialized Hardware Revolution

A DARPA Information Science and Technology (ISAT) Study

Leads:
Sarita Adve, University of Illinois
Ras Bodik, University of Washington

Steering Committee: Luis Ceze, University of Washington

April 1, 2019

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA).
Approved for public release; distribution unlimited.
The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official view of policies of the Department of Defense or the U.S. Government.
Post-Moore: Exploding Heterogeneity and Cost

How to build the software stack?
What is the hardware-software interface?

Right interface can address cost
Free hardware/software designer to innovate

Source: Brooks, Wei group, http://vlsiarch.eecs.harvard.edu/accelerators/die-photo-analysis

Source: Brooks, Wei group, http://vlsiarch.eecs.harvard.edu/accelerators/die-photo-analysis

CPU = Central Processing Unit, GPU = Graphics Processing Unit, ISA = Instruction Set Architecture, CUDA = Compute Unified Device Architecture
Why Now

- **Explosion of accelerators**
  - Broaden accelerator applicability from kernels to apps and infrastructure
  - Accelerate memory and communication, too

- **Move to system view of specialization**
  - Focus on specialization of communication, to connect multiple hardware IPs
  - Solve composability and portability, to co-develop accelerators
  - Manage software cost, to make system-wide specialization affordable

- **Develop next-generation interface methodologies**
  - Convey multiple properties: security, verifiability, accuracy, …
  - Inflection point in tools for verification, synthesis, machine learning, …

- **Open-source hardware and other Electronics Resurgence Initiative investments**
Three (Related) Views of Interfaces

**Uniform Interface View**
- Software developed independent of hardware
- Mobile devices
- Desktops
- Servers
- Data centers
- Supercomputers

**Co-designed Stack View**
- Co-design of software & hardware
- Accelerators
- Embedded systems
- Internet-of-Things devices
- Domain-specific languages

**Catalog of Parts View**
- Diverse hardware and software components that must interoperate
- Rich interfaces enable automatic composition, verification, tuning
Uniform Interface View

For software developed independent of hardware

Key: Uniform abstractions for diverse hardware

Diverse Software

Uniform Interface(s)

Diverse Hardware

Front-ends, tools for diverse languages

Back-ends, optimizers, autotuners, schedulers for high performance
Current Interface Levels: Which Can Be Uniform?

Application productivity: **Domain-specific language**

Application performance: **General-purpose language**

Language innovation: **Language-level Compiler IR**

Compiler investment: **Language-neutral Compiler IR**

Object-code portability: **Virtual ISA**

Hardware innovation: **"Hardware" ISA**

Too diverse to define a uniform interface

Much more uniform

Also too diverse ...

Source: Vikram Adve, HPVM project, https://publish.illinois.edu/hpvm-project/
### Current Interface Levels: Which Can Be Uniform?

<table>
<thead>
<tr>
<th>Category</th>
<th>Interface Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application productivity</td>
<td>Domain-specific language</td>
</tr>
<tr>
<td>Application performance</td>
<td>General-purpose language</td>
</tr>
<tr>
<td>Language innovation</td>
<td>Language-level Compiler IR</td>
</tr>
<tr>
<td>Compiler investment</td>
<td>Language-neutral Compiler IR</td>
</tr>
<tr>
<td>Object-code portability</td>
<td>Virtual ISA</td>
</tr>
</tbody>
</table>

**Too diverse to define a uniform interface**

**Much more uniform**

What should this uniform interface be?
How to represent software attributes to maximize efficiency on diverse hardware?
How to create front ends and tools for diverse languages?
How to create back-ends, optimizers, autotuners, schedulers for diverse hardware?
Uniform Interface View: Potential Surprise

Unlocks 100-1000x efficiency of heterogeneous hardware
Zero Hour SW Bring Up: Software ready as soon as hardware off fab

**Today**
- DSL 1
  - DSL1-HW compiler
  - h/w1
  - x86
- DSL 2
  - DSL2-HW compiler
  - h/w2
  - FPGA bitstream
- DSL 3
  - DSL3-HW compiler
  - h/w3
  - FPGA bitstream

**Tomorrow**
- DSL 1
  - DSL1-IF compiler
  - IF
  - LLVM 2.0
  - HW1
  - HW3
  - FPGA bitstream
  - x86
- DSL 2
  - DSL2-IF compiler
  - HW implementation of IF

DSL = Domain-Specific Language  
HW = Hardware  
SW = Software  
IF = Interface
Example 1: HPVM: Compiler IR and Virtual ISA [V. Adve et al.]

HPVM Model
Hierarchical Dataflow Graph (with side effects)

$V_a = \text{load} <L4 \times \text{float}>* A$
$V_B = \text{load} <L4 \times \text{float}>* B$
$\ldots$
$V_C = \text{fmul} <L4 \times \text{float}> V_a, V_B$

HPVM comes close to separate hand-tuned code on GPU, vectors

8% slower on Vector
3% slower on GPU

HPVM enables highly flexible static or dynamic scheduling policies

Single program: $N^k$ mappings

$N$ graph nodes
Static OR
$K$ devices
dynamic mappings

HPVM = Heterogeneous Parallel Virtual Machine
Kotsifakou et al., PPoPP’18
Example 2: Delite IR: Parallel Pattern Lang. [Olukotun et al.]

Most data analytic computations can be expressed as functional data parallel patterns on collections (e.g. sets, arrays, tables, n-d matrices)

Parallel Patterns
Map, Zip, Filter, FlatMap, Reduce, GroupBy, Join, Sort, …

Key elements
- DSLs embedded in Scala
- IR created using type-directed staging
- Domain specific optimization
- General parallelism, locality optimizations using parallel patterns
- Optimized mapping to hardware targets

K. J. Brown et. al., PACT, 2011; K. J. Brown et. al., CGO 2016
Codesigned Stack View

Co-design of hardware and software

Key: Coordinated stack of codesigned interfaces

High-level interface for DSL construction
Low-level interface for hardware
Automated generation of stack
Coordinated Stack of Interfaces

Bottlenecks in accelerator design
- What to accelerate?
- What is the hardware/software interface?
- Developer tools and IR stack

New interfaces appear in a coordinated stack of interfaces, needing coordinated effort of experts

Takes years of design and implementation today, not reusable for other domains

Source: Olukotun, I-USHER workshop
Coordinated Stack of Interfaces

Bottlenecks in accelerator design
- What to accelerate?
- What is the hardware/software interface?
- Developer tools and IR stack

How to automate this process?
How to reuse across domains?
Modular, configurable IRs?
Retargetable toolchains for new IRs?
Leverage uniform interface view?
Example process

1. Collect **representative apps** or kernels
2. Automatically rewrite into **alternative algorithms**
3. Identify performance **bottlenecks**
4. **Map** hardware primitives to software dataflow graphs; select best hardware design
5. **Infer** hardware interface
6. **Synthesize** DSL spec
7. Automatically **construct compiler** from DSL to accelerator
8. **Design hardware** that implements the hardware interface
Example 1: Spatial: IR for Accel. Design [Olukotun et al.]

Simplify accelerator design

- IR that can be mapped to many hardware targets: FPGA, ASIC, ...
- Constructs to express:
  - Parallel patterns as parallel and pipelined datapaths
  - Hierarchical control
  - Explicit memory hierarchies
  - Explicit parameters
- Optimizes parameters for each target: parallelization, pipelining, memory size, memory banking

Allows programmers & high level compilers to focus on specifying parallelism and locality

D. Koeplinger et al. Al. PLDI 2018

```scala
val output = ArgOut[Float]
val vectorA = DRAM[Float](N)
val vectorB = DRAM[Float](N)

Accel {
    Reduce(output)(N by B){ i =>
        val tileA = SRAM[Float](B)
        val tileB = SRAM[Float](B)
        val acc = Reg[Float]

        tileA load vectorA(i :: i+B)
        tileB load vectorB(i :: i+B)

        Reduce(acc)(B by 1){ j =>
            tileA(j) * tileB(j)
        }{a, b => a + b}
    }
}
```

6500x Speedup Over HLS!
Example 2: TVM for Automated Hardware/Software Co-Design [Ceze et al.]

150+ contributors, several production industrial users.

Mapping ML code to diverse hardware typically requires a significant amount of hand-tuning over a space with billions of possibilities.

A solution is to use learning techniques to make tuning automatic. Recent advances such as automatic optimization in the TVM stack show significant improvement compared to hand-tuned implementations.

This technique is now being applied to automatic hardware/software co-design.

AutoTVM Conv2d example on TitanX

Source: UW SAMPL group (sampl.ai)
**Example 3: Stream Dataflow Execution [Sankaralingam et al.]**

**5 common principles for domain specific architecture (DSA)**

- **Concurrency**
- **Computation**
- **Communication**
- **Data Reuse**
- **Coordination**

**Stream-Dataflow ISA**

- **Set-up Interface**
  - `SD_Config` - Configuration data stream for dataflow computation fabric (CGRA)

- **Control Interface**
  - `SD_BARRIER_SCRATCH_RD`, `SD_BARRIER_SCRATCH_WR`, `SD_BARRIER_ALL`

- **Stream Interface** → `SD_[source]_[dest]`

  *Source/Dest Parameters: Address (memory or local storage), DFG Port number, Pattern Parameters: access_size, stride_size, num_strides*

**Stream-Dataflow Execution Model**

- **Computation abstraction** – Dataflow Graph (DFG) with input/output vector ports
- **Data abstraction** – Streams of data fetched from memory and stored back to memory
- **Reuse abstraction** – Streams of data fetched once from memory, stored in local storage (programmable scratchpad) and reused again
- **Communication abstraction** – Stream-Dataflow data movement commands and barriers

- **Stream-Dataflow Acceleration, ISCA-2017**
- **Domain Specialization is generally unnecessary for accelerators, HPCA 2016 & Top-Picks**
- **Analyzing Behavior Specialized Acceleration, ASPLOS-2016**
- **Exploring the Potential of Heterogeneous Von Neumann/Dataflow Execution Models, ISCA-2015, Top-Picks, CACM RH**
Catalog of Parts View

For plug-and-play hardware and software

Hardware and Software Component Specs for Heterogeneous Computing

Key: Rich, formal, composable interfaces

Automated, verified composition

Communication

Tuning

The TTL Data Book for Design Engineers Second Edition
Author: The Engineering Staff of Texas Instruments, 1976

In this 832-page data book, Texas Instruments is pleased to present important technical information on the industry's broadest and most advanced families of TTL integrated circuits. — You'll find complete specifications on standard-technology TTL circuits (Series 54/74, Series 54H/74H, Series 54L/74L) and on TI's high-technology TTL circuits such... more »
Towards Formal Interfaces for Universal Plug and Play

Different cadence of innovation between hardware and software, between accelerators

To deploy new parts ASAP, need clean interfaces to “plug and play”

Today’s parts

- Interfaces in English
- Glue logic explosion
  - Linux: 12M of 15M LOC in drivers
- Inefficiencies of driver-driver interactions
- Bugs in inter-IP block interactions
- No composability, build from scratch rather than reuse
Towards Formal Interfaces for Universal Plug and Play

Different cadence of innovation between hardware and software, between accelerators

To deploy new parts ASAP, need clean interfaces to “plug and play”

How to specify formal, machine checkable spec

- Operational spec for part + how parts connect
  - **Shim** to connect parts is also a part
  - **Communication/memory** first order
- Express performance, accuracy, resource use, security, ...

TI OMAP4 SoC
Catalog of Parts View: The Surprise

Reusable, verifiable, secure, market-driven ecosystem of parts that can composably interoperate and has checkable performance+semantic properties

Source: Sharad Malik, I-USHER workshop
Example 1: Instruction-Level Abstraction (ILA) [Malik et al.]

ILA: ISA-like Abstraction
- **Uniform**: accelerator & processor
- ** Hierarchical**: multi-level
- Enables formal software/hardware co-verification
- ILA compatibility for accelerator replacement

Modeling Accelerators
- Gaussian Blur Image Processing (Horowitz Group)
  - Different levels of abstractions
- AES Block Encryption (OpenCores.org)
  - One spec, two implementations
- Restricted Boltzmann Machine (Carloni Group)
  - Decomposition of computation from interface protocol

Processor ISA
- RISC-V RV32I base instruction set w. privilege instructions

Verification
- Accelerator upgrades
- Found RISC-V Rocket MRET/SRET bug
- Verified AES/RBM/GB accelerators

Insight: treat commands at interface as instructions
Example 2: CheckSuite [Martonosi et al.]

An ecosystem of tools to verify cross-layer consistency, coherence interfaces

<table>
<thead>
<tr>
<th>High-Level Languages (HLL)</th>
<th>Compiler</th>
<th>OS</th>
<th>Architecture</th>
<th>Microarchitecture</th>
<th>RTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TriCheck [ASPLOS ‘17] [IEEE MICRO Top Picks]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COATCheck [ASPLOS ‘16] [IEEE MICRO Top Picks]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PipeCheck [Micro-47] [IEEE MICRO Top Picks]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCICheck [Micro-48] [Nominated for Best Paper Award]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTLCheck [Micro-50] [MICRO Top Picks Hon. Mention]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tools found bugs in:
- Widely-used Research simulator
- Cache coherence paper
- IBM XL C++ compiler (fixed in v13.1.5)
- In-design commercial processors
- RISC-V ISA specification
- Compiler mapping proofs
- C++ 11 mem model

Approach
- Formal specifications -> Happens-before graphs
- Check Happens-Before Graphs via Efficient SMT solvers
  - Cyclic => A->B->C->A... Can’t happen
  - Acyclic => Scenario is observable
Example 3: Spandex [S. Adve et al.]

Goal: Accelerator communication, coherence interface

Spandex Coherence Interface

Key Components
- Flexible device request interface
- External request interface
- DeNovo-based LLC
- Device may need translation unit

<table>
<thead>
<tr>
<th>Request</th>
<th>Generated for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>ReqV: Self-invalidating read</td>
</tr>
<tr>
<td></td>
<td>ReqS: Writer-invalidated read</td>
</tr>
<tr>
<td>Write</td>
<td>ReqWT: Write-through store</td>
</tr>
<tr>
<td></td>
<td>ReqO: Write-only ownership store</td>
</tr>
<tr>
<td>Read+Write</td>
<td>ReqWT+data: Atomic for WT cache</td>
</tr>
<tr>
<td></td>
<td>ReqO+data: Read-for-ownership store, Atomic for ownership cache</td>
</tr>
<tr>
<td>Writeback</td>
<td>ReqWB: Owned data eviction</td>
</tr>
</tbody>
</table>

Alsop et al. ISCA’18
Zero hour software bring up  + Rapid HW-SW codesign + Machine checked plug and play
Unlock usable specialization for embedded devices to planetary scale computing
Address performance, efficiency, portability, HW & SW design productivity, verifiability, security
Measuring Success

Time to Market (HW+SW)

- Months to Years
- Days to Weeks

Graphs showing the increase in the number of specialized IP blocks from 2010 to 2017 and the cost to market in million dollars across different silicon technology nodes from 0.5u to 20nm.

Bar graphs indicating the breakdown of costs into HW Design and Verification, SW Development and Testing, and Mask Costs.
Appendix: I-USHER Workshop Participants (March 5-6, 2018)

Sarita Adve, Illinois/ISAT
Vikram Adve, Illinois
Ras Bodik, Washington/ISAT
David Brooks, Harvard
Luis Ceze, Washington/ISAT
David Doermann, DARPA
Chris Fletcher, Illinois
Vinod Grover, NVIDIA
Priscilla Guthrie, ISAT
Mark Hill, Wisconsin
Shan Lu, U. Chicago
Sharad Malik, Princeton
Margaret Martonosi, Princeton
Sasa Misailovic, Illinois
Sandeep Neema, DARPA
Kunle Olukotun, Stanford
Chris Ramming, VMware/ISAT
Partha Ranganathan, Google
Jonathan Ragan-Kelley, Berkeley
Tatiana Shpeisman, Google
Michael Taylor, Washington
Kathy Yelick, Berkeley