

Workshop on Renegotiating the Levels of Abstraction for the Post Moore's Law Era @ ARM Research Summit

Interface, Communication, Approximation

Sarita Adve

**University of Illinois at Urbana-Champaign
sadve@illinois.edu**

w/ I-USHER ISAT study participants, Illinois HPVM, Spandex, approximate computing teams

Sponsors: NSF, ADA (JUMP center by SRC, DARPA), DARPA DSSOC program

Post-Moore Imperatives

- Interface
- Communication
- Approximation

Application-driven

Post-Moore Imperatives

- **Interface**
- **Communication**
- **Approximation**

Application-driven

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

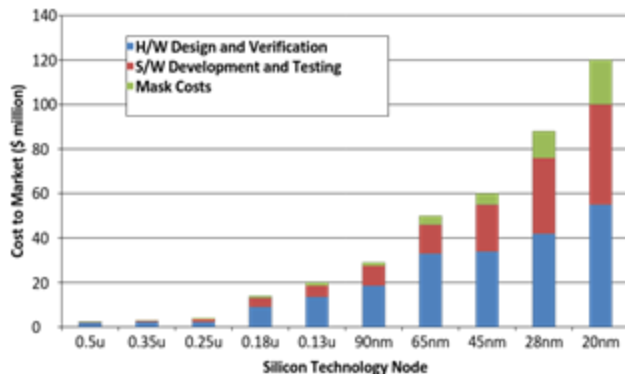
April 1, 2019

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA).

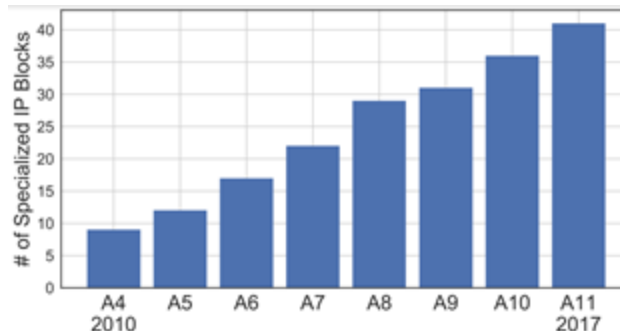
Approved for public release; distribution unlimited.

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



Source: International Business Strategies
Graph from Todd Austin's seminar @ UIUC, 8/17



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



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

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

Technology

CPUs

Databases

Datacenters

GPUs

Internet

Custom hardware

Enabling Interface

ISAs

Relational queries

MapReduce

CUDA

IP

???

Right interface can address cost

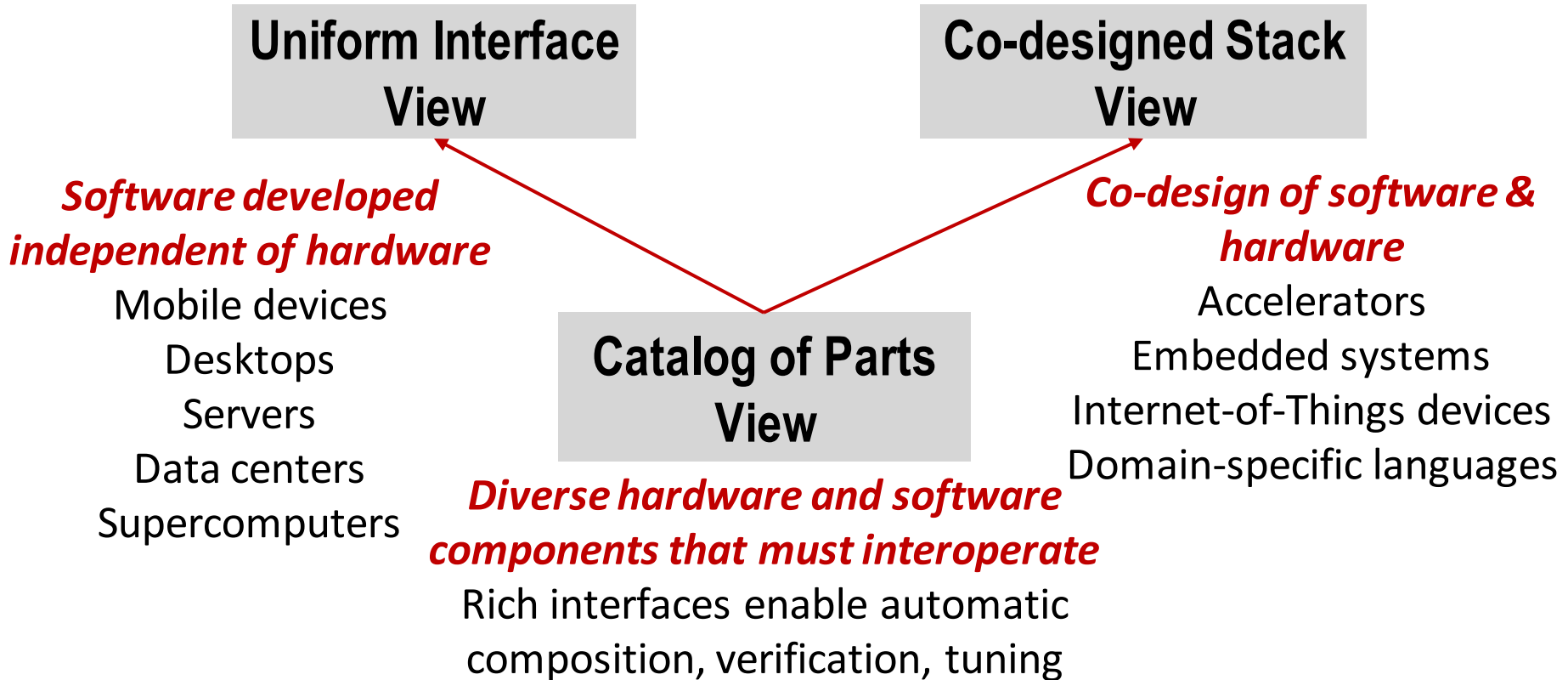
Free hardware/software designer to innovate

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

Why Now

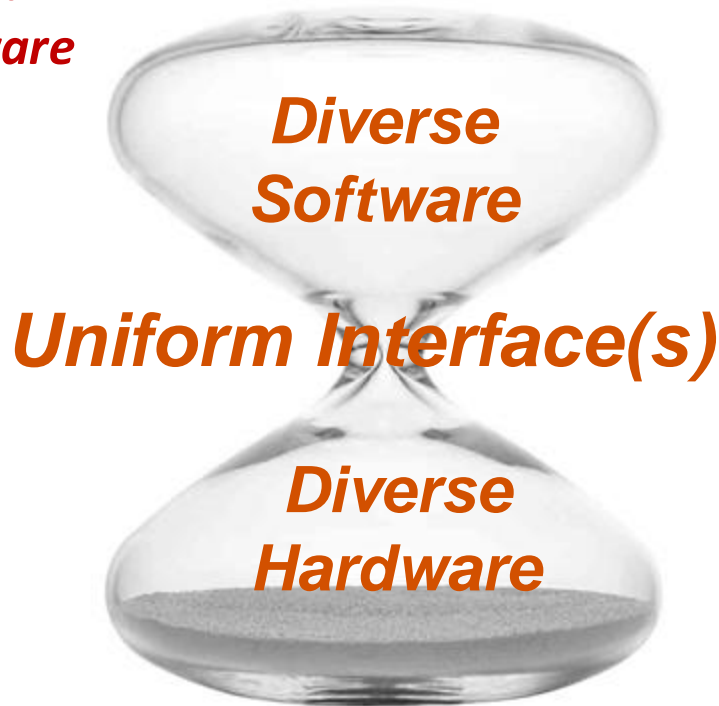
- **Explosion of accelerators**
 - Broaden accelerator applicability from kernels to apps and infrastructure
 - Accelerate memory and communication, too
- **Move to system view of specialization**
 - Focus on specialization of communication, to connect multiple hardware IPs
 - Solve composability and 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

*For software developed
independent of hardware*

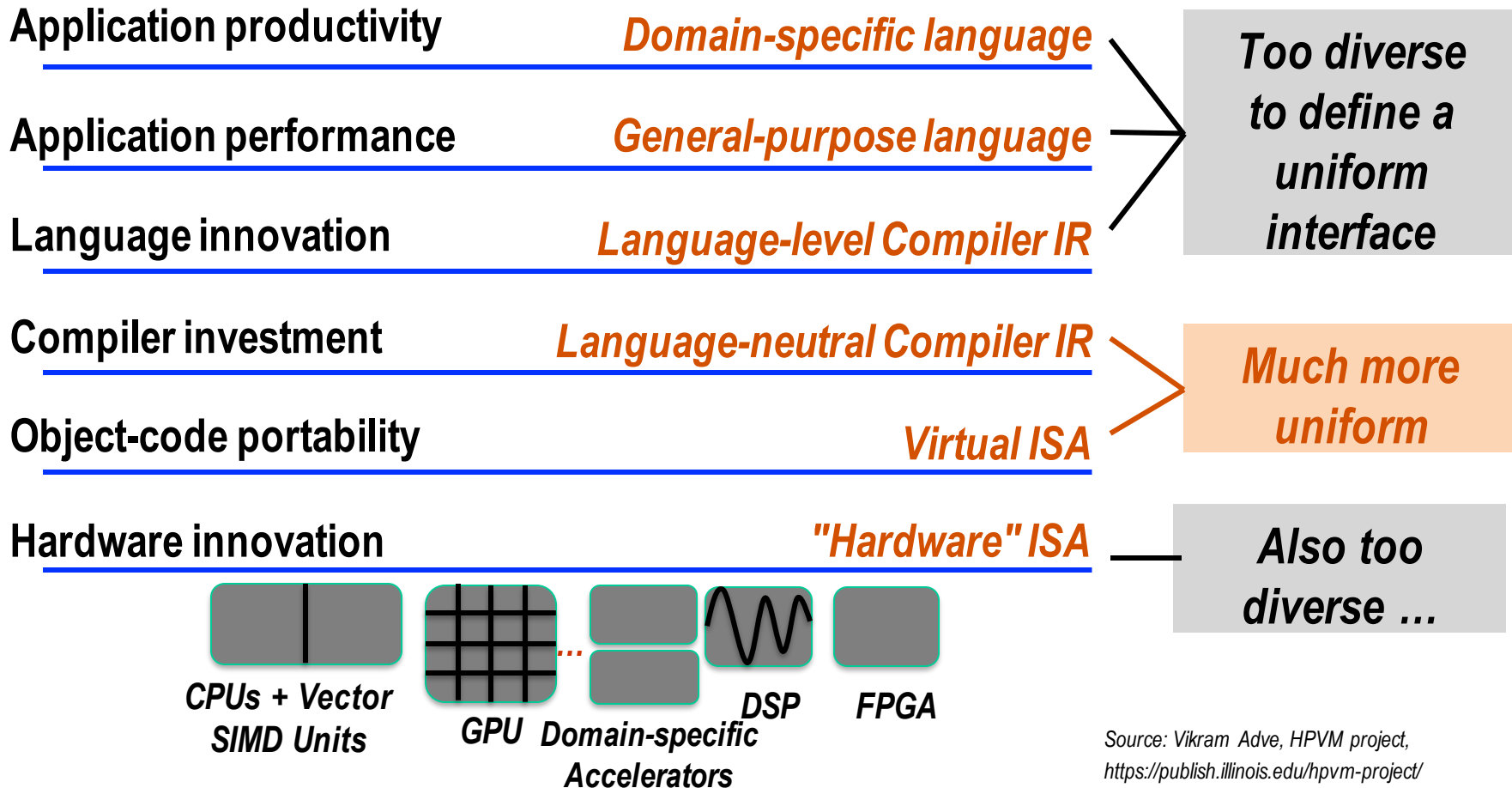


**Key: Uniform
abstractions for
diverse hardware**

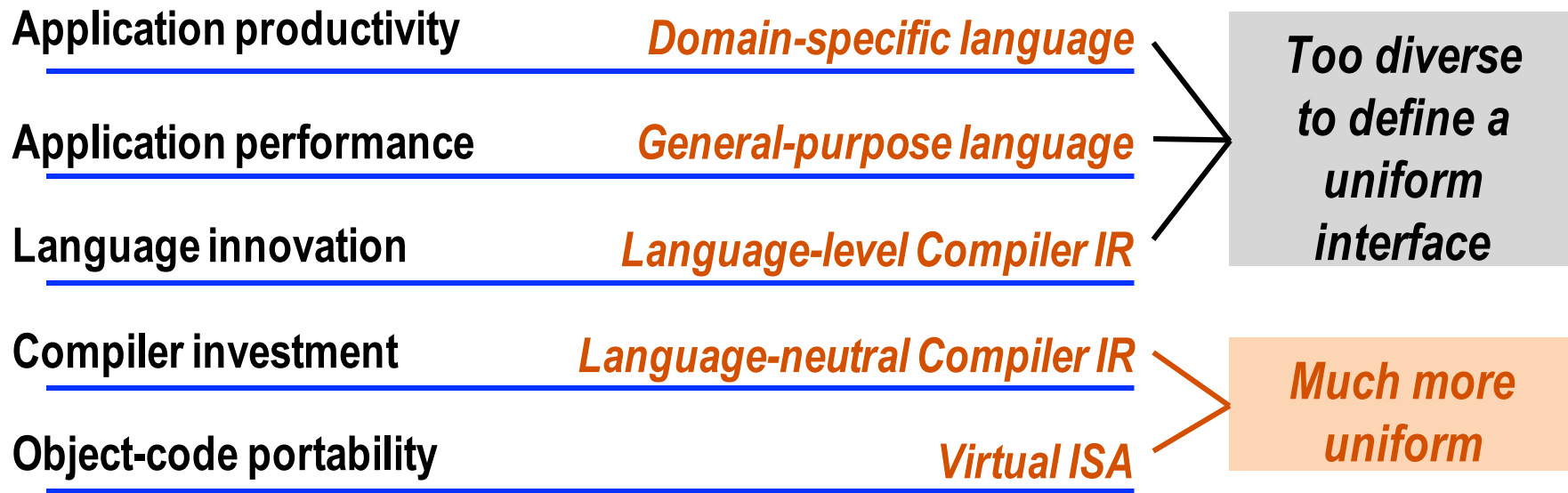
Front-ends, tools for
diverse languages

Back-ends, optimizers,
autotuners, schedulers
for high performance

Current Interface Levels: Which Can Be Uniform?



Current Interface Levels: Which Can Be Uniform?



What should this uniform interface be?

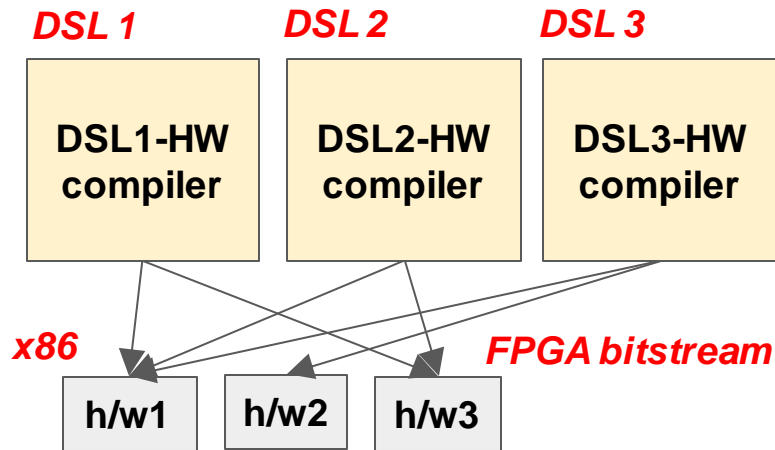
How to represent software attributes to maximize efficiency on diverse hardware?

How to create front ends and tools for diverse languages?

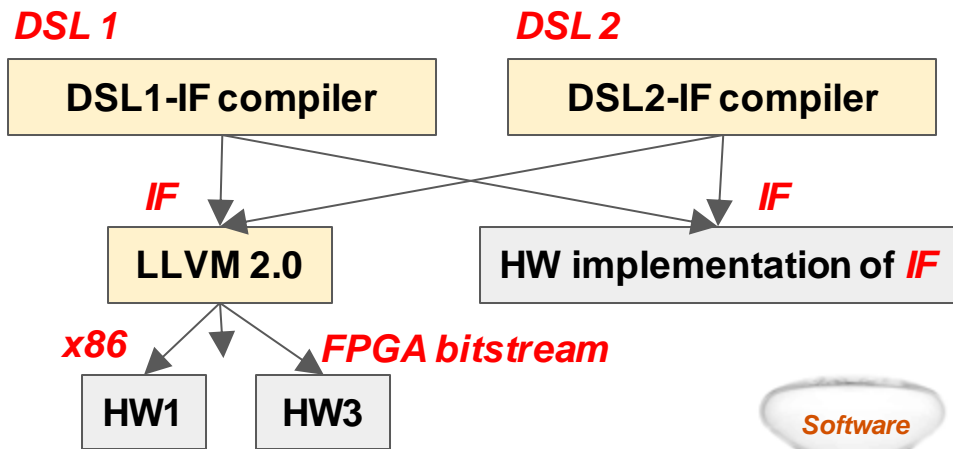
How to create back-ends, optimizers, autotuners, schedulers for diverse hardware?

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



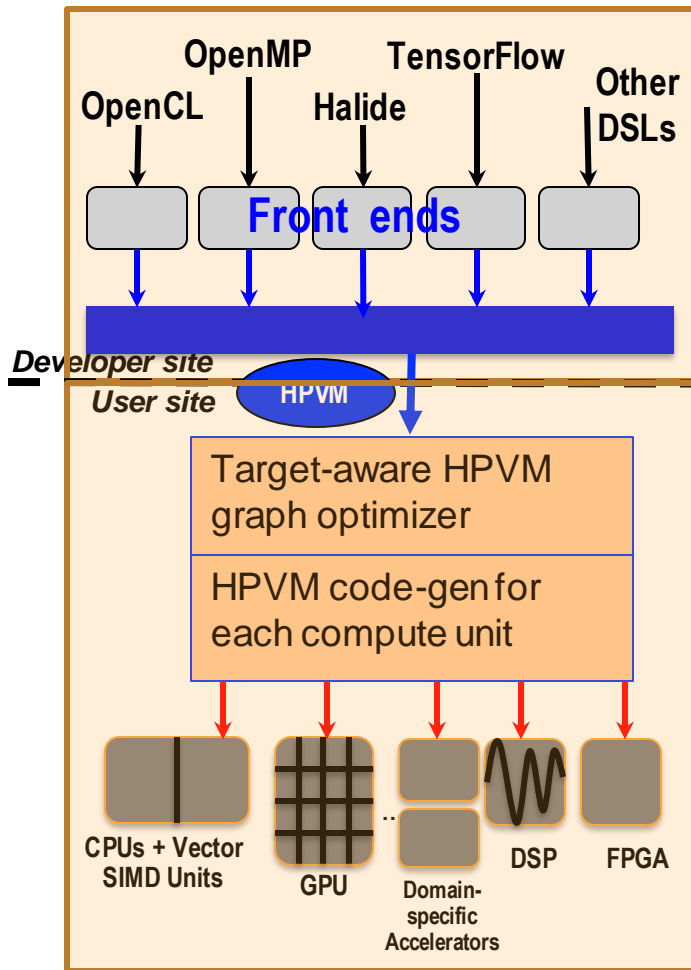
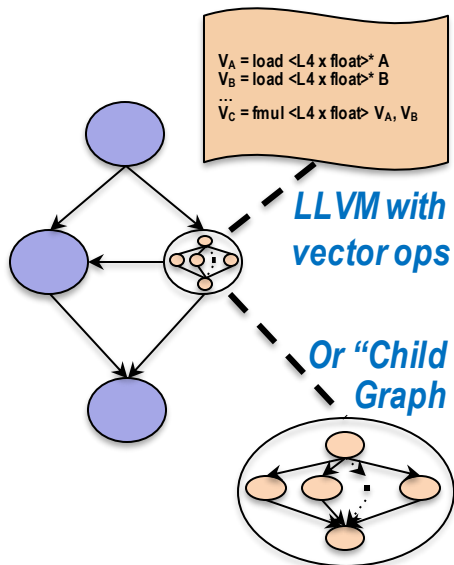
Tomorrow

DSL = Domain-Specific Language HW = Hardware SW = Software IF = Interface

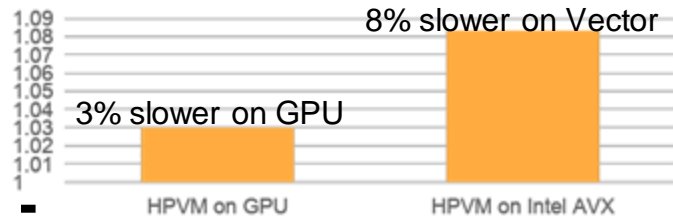


Example 1: HPVM: Compiler IR and Virtual ISA [V. Adve et al.]

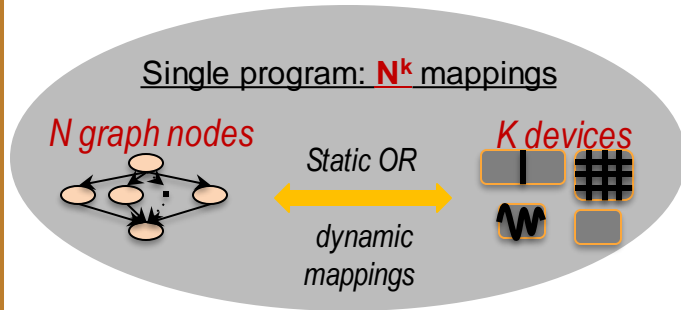
HPVM Model
Hierarchical
Dataflow Graph
(with side effects)



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



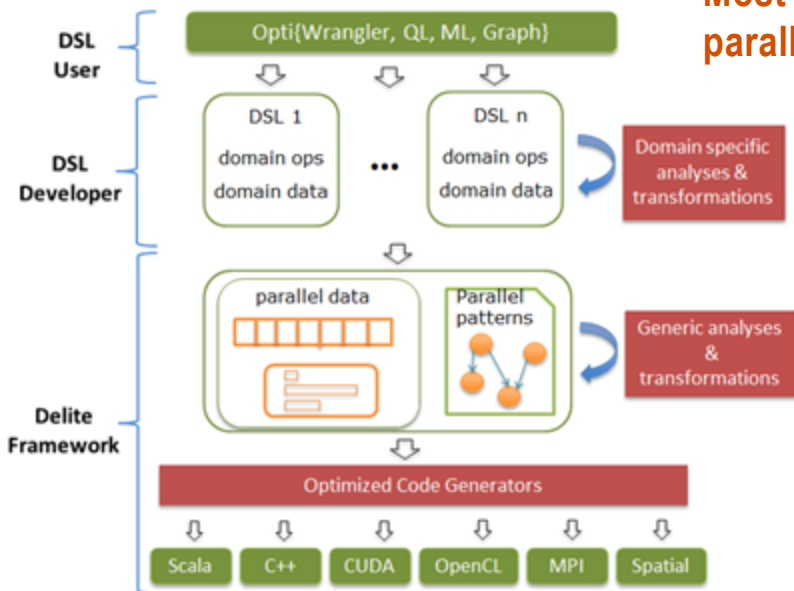
HPVM enables highly flexible static
or dynamic scheduling policies



HPVM = Heterogeneous Parallel Virtual Machine
Kotsifakou et al., PPOPP'18

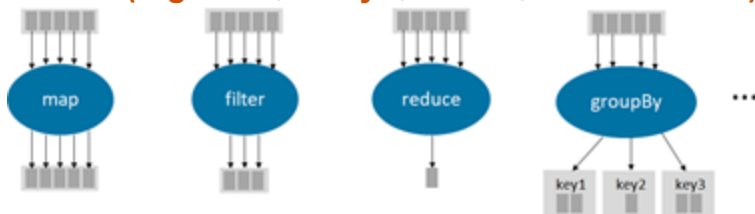
Example 2: Delite IR: Parallel Pattern Lang. [Olukotun et al.]

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



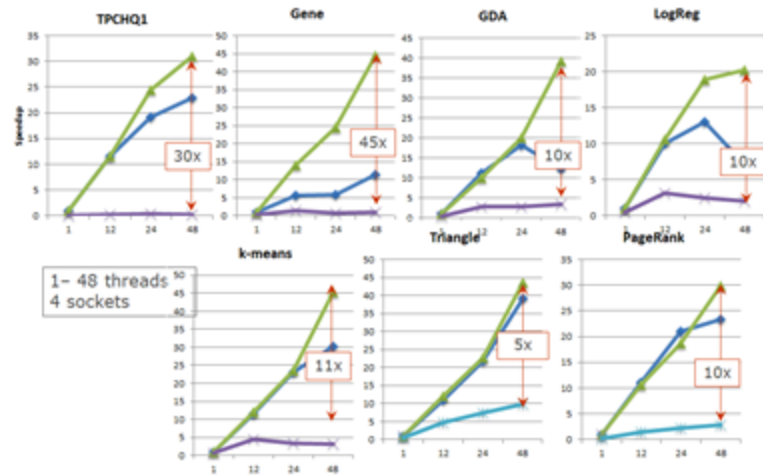
Parallel Patterns

Map, Zip, Filter, FlatMap, Reduce, GroupBy, Join, Sort, ...



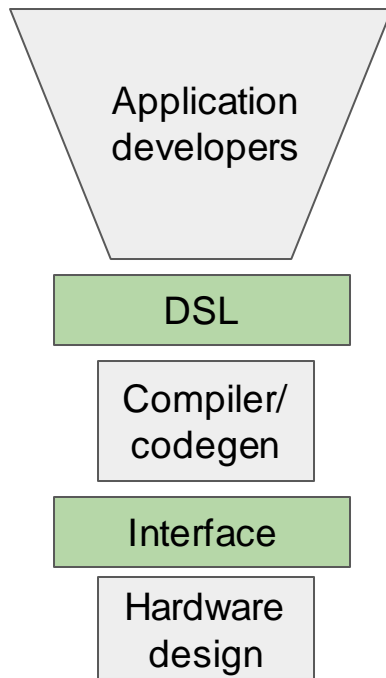
Key elements

- 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



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

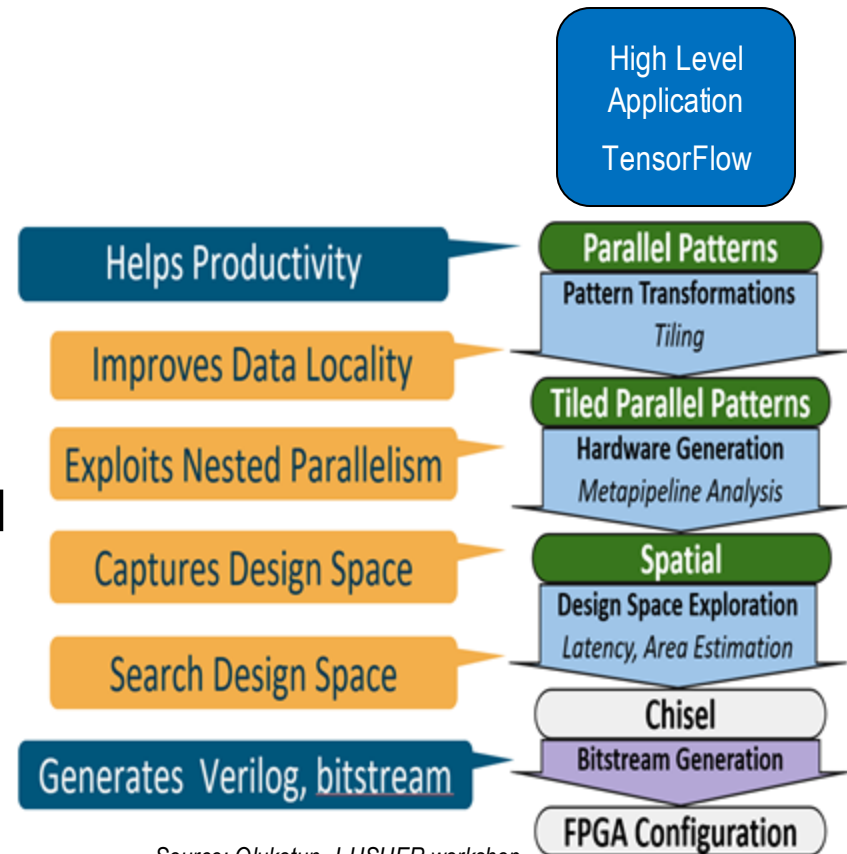
Coordinated Stack of Interfaces

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



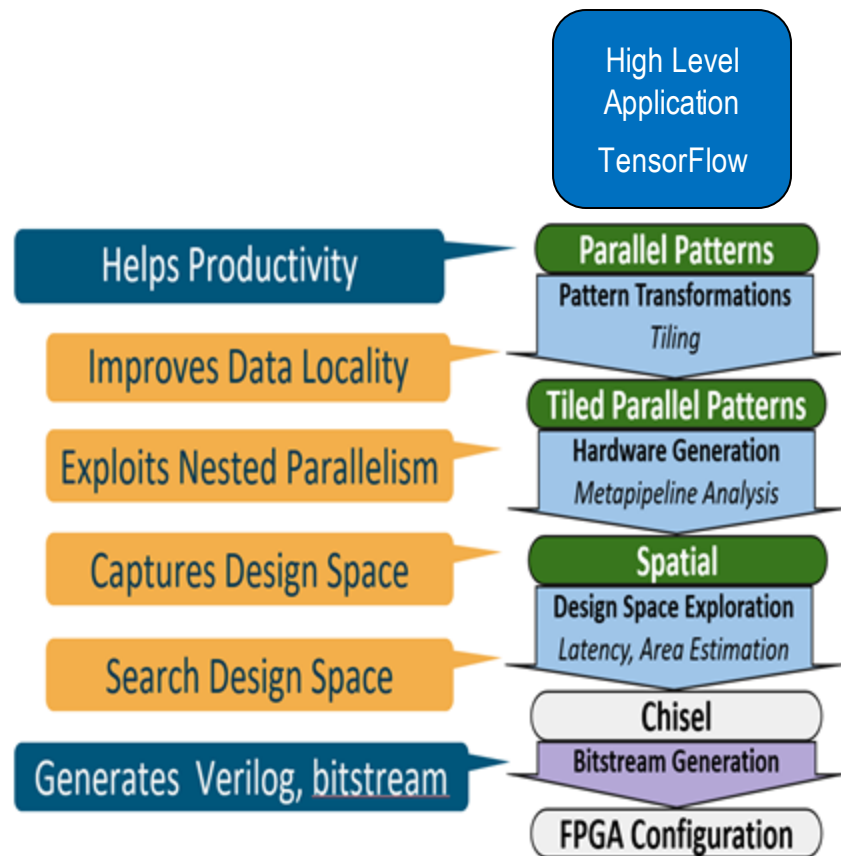
Source: Olukotun, I-USHER workshop

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?

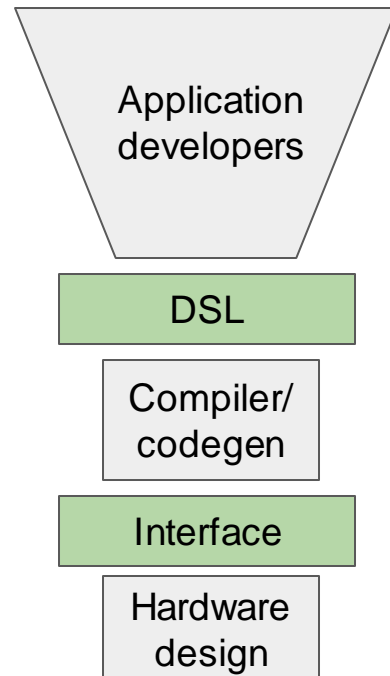


Codesigned Stack View: Potential Surprise

Semi-automatic generation of co-designed hardware interface and DSL for chosen domain

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. *PLDI 2018*

```
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}
  }{a, b => a + b}
}
```

Spatial: ~30 lines

Benchmark	Designs	Search Time
Dot Product	5,426	5.3 ms / design
Outer Product	1,702	30 ms / design
TPCH		
Black		
Matrix Multiply	70,740	11 ms / design
K-Means	75,200	20 ms / design
GDA	42,800	17 ms / design

6500x Speedup Over HLS!

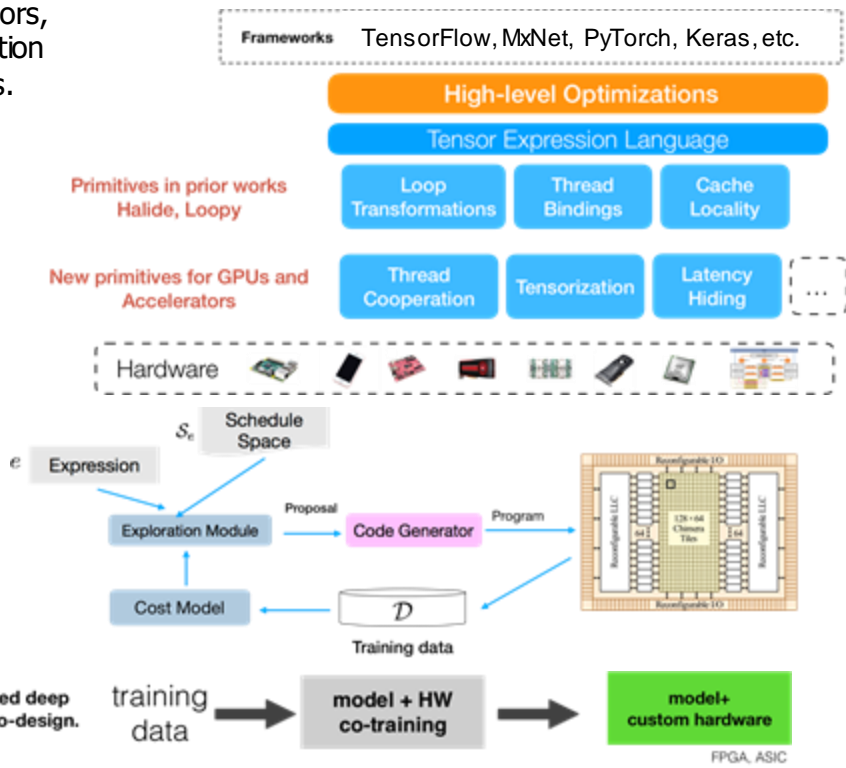
Vivado HLS	Designs	Search Time
GDA	250	1.85 min / design

[illegible]

Chisel: ~3200 lines

Example 2: TVM for Automated Hardware/Software Co-Design [Ceze et al.]

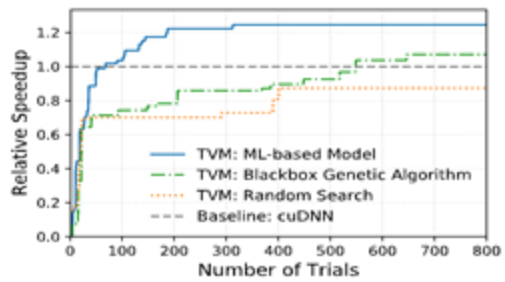
150+ contributors, several production industrial users.



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.

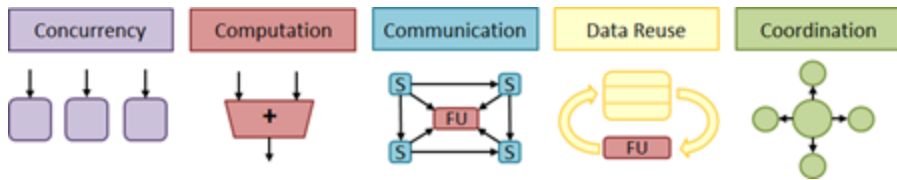


AutoTVM Conv2d example on TitanX

Source: UW SAMPL group (sAMPL.ai)

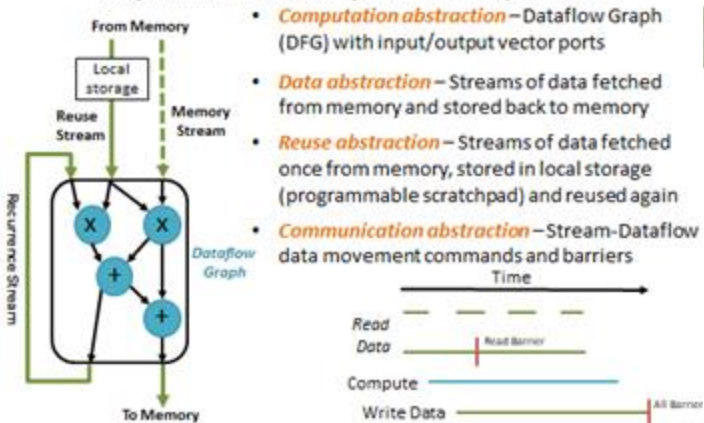
Example 3: Stream Dataflow Execution [Sankaralingam et al.]

5 common principles for domain specific architecture (DSA)

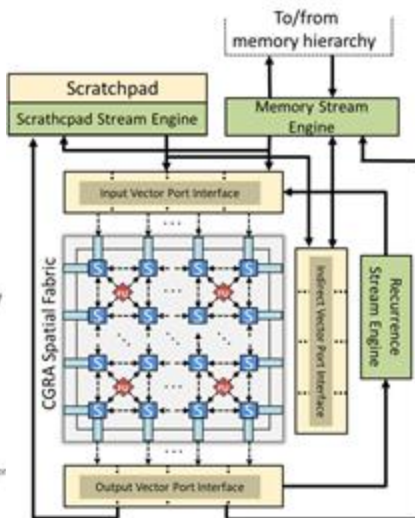


Stream-Dataflow Execution Model

Programmer Abstractions for Stream-Dataflow 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 ISA

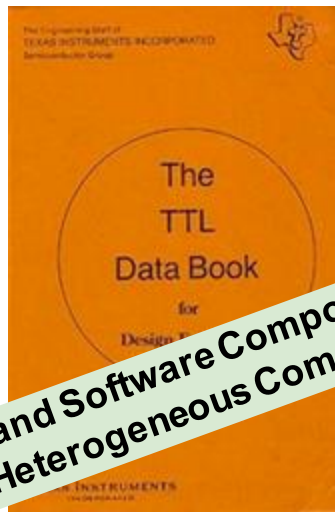
- Set-up Interface:**
 - SD_Config** – Configuration data stream for dataflow computation fabric (CGRA)
- Control Interface:**
 - SD_Barrier_Scratch_Rd**, **SD_Barrier_Scratch_Wr**, **SD_Barrier_All**
- Stream Interface** → **SD_[source]_[dest]**
 Source/Dest Parameters: Address (memory or local storage), DFG Port number
 Pattern Parameters: access_size, stride_size, num_strides

Command Name	Parameters	Description
SD_Config	Address, Size	Stream CGRA configuration from given address
SD_Mem_Scratch	Source Mem Address, Stride, Access Size, Num Strides, Dest, Scratch Address	Read from memory with pattern to scratchpad
SD_Scratch_Port	Source Scratch Address, Stride, Access Size, Strides, Input Port #	Read from scratchpad with pattern to input port
SD_Mem_Port	Source Mem Address, Stride, Access Size, Num Strides, Input Port #	Read from memory with pattern to input port
SD_Const_Port	Constant Value, Num Elements, Input Port #	Send constant value to input port
SD_Out_Port	Num Elements, Output Port #	Throw away extra elements from output port
SD_Pat_Port	Output Port #, Num Elements, Input Port #	Issue recurrences between input-output port pairs
SD_Pat_Scratch	Output Port #, Num Elements, Scratch Address	Issue recurrences between input-output port pairs
SD_Pat_Mem	Output Port #, Stride, Access Size, Num Strides, Dest, Mem Address	Write from port to memory with pattern
SD_Mem_Indirect	Source Mem Address, Stride, Access Size, Num Strides, Indirect Port #	Read the addresses from memory with pattern to indirect port
SD_Indirect_Port	Indirect Port #, Offset Address, Input Port #	Indirect load from addresses present in indirect port
SD_Indirect_Mem	Indirect Port #, Output Port #, Dest, Offset Address	Indirect store to addresses present in indirect port
SD_Barrier_Scratch_Rd	-	Barrier for scratchpad reads
SD_Barrier_Scratch_Wr	-	Barrier for scratchpad writes
SD_Barrier_All	-	Barrier to wait for all commands completion

- 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*



**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»](#)

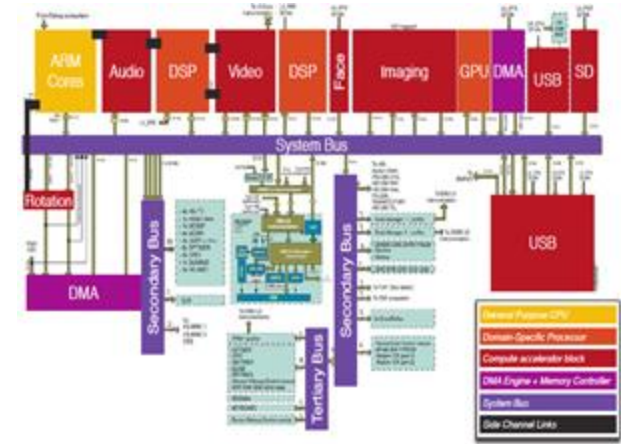
Towards Formal Interfaces for Universal Plug and Play

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



TI OMAP4 SoC

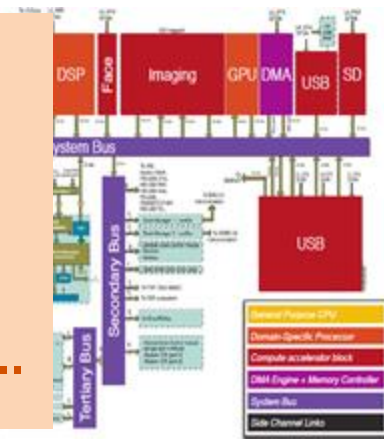
Towards Formal Interfaces for Universal Plug and Play

Different cadence of innovation between hardware and software, between accelerators

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

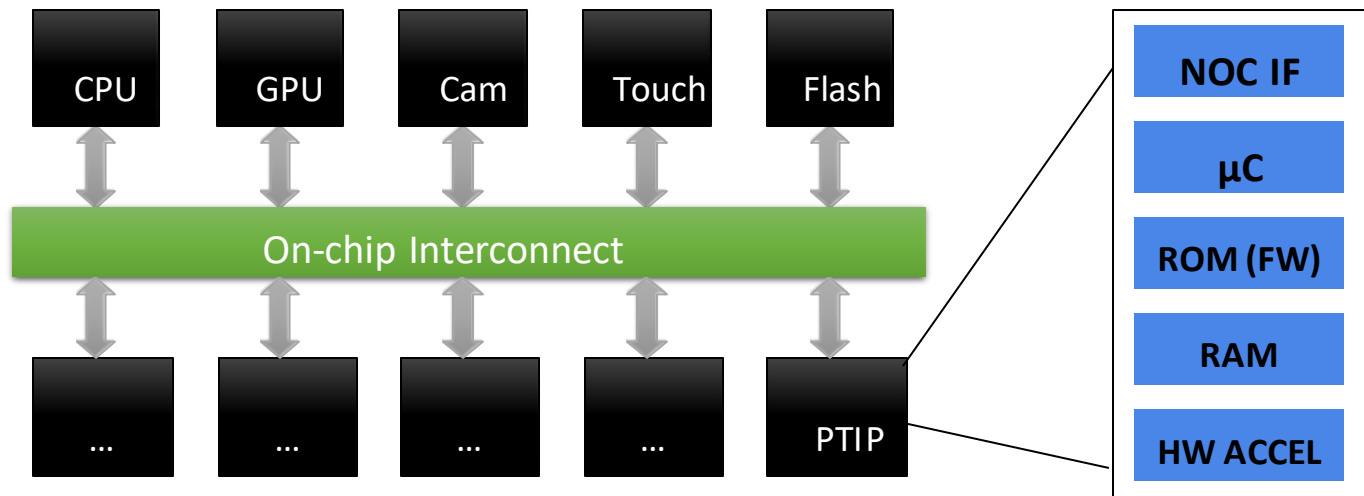
How to specify formal, machine checkable spec

- 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, ...

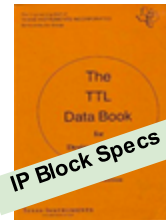


Catalog of Parts View: The Surprise

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



Source: Sharad Malik, IUSHER 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**



START_ENCRYPT	Write, 0xff00, 0x1	A sequence of operations
STORE_LENGTH	Write, 0xff10, data	Length := data

Insight: treat commands at interface as **instructions**

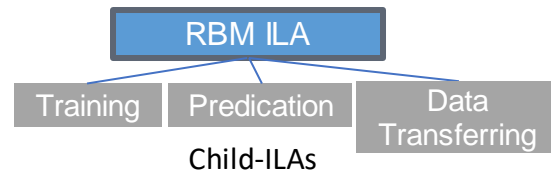
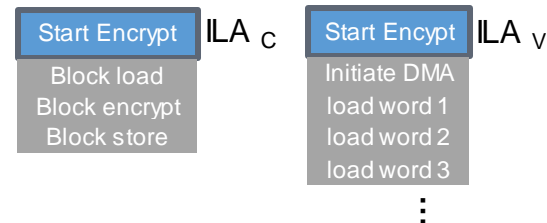
Modeling Accelerators

- 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

Halide description	High-level ILA
C++ for HLS	Low-level ILA
RTL implementation	

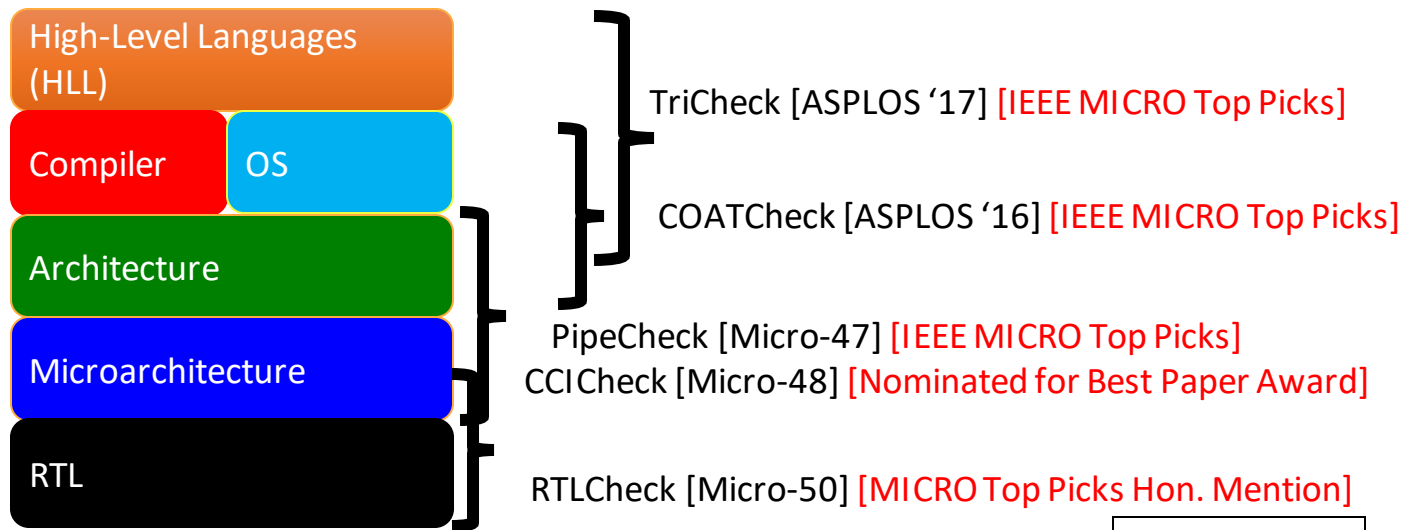


Verification

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

Example 2: CheckSuite [Martonosi et al.]

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

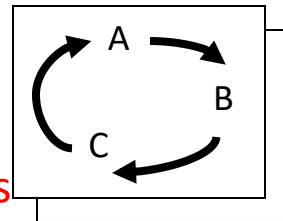


Tools found bugs in:

- 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

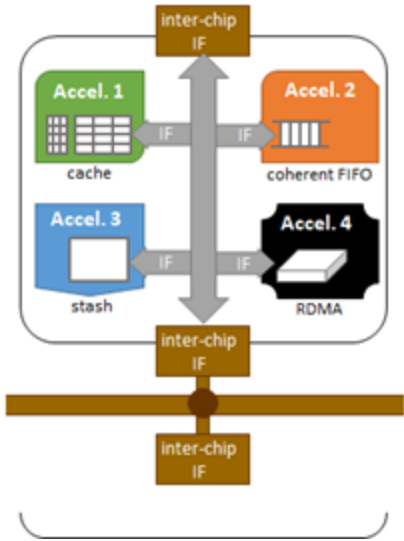
Approach

- Formal specifications \rightarrow Happens-before graphs
- Check **Happens-Before Graphs** via **Efficient SMT solvers**
 - Cyclic $\Rightarrow A \rightarrow B \rightarrow C \rightarrow A \dots$ **Can't happen**
 - Acyclic \Rightarrow Scenario is **observable**

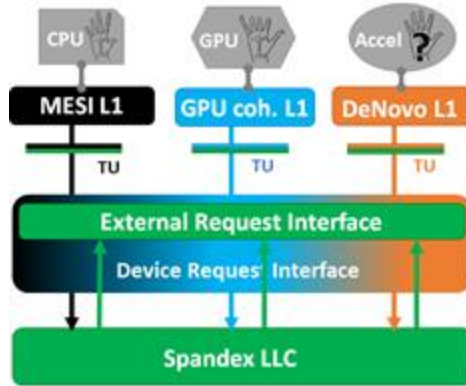


Example 3: Spandex [S. Adve et al.]

Goal: Accelerator communication, coherence interface



Spandex Coherence Interface

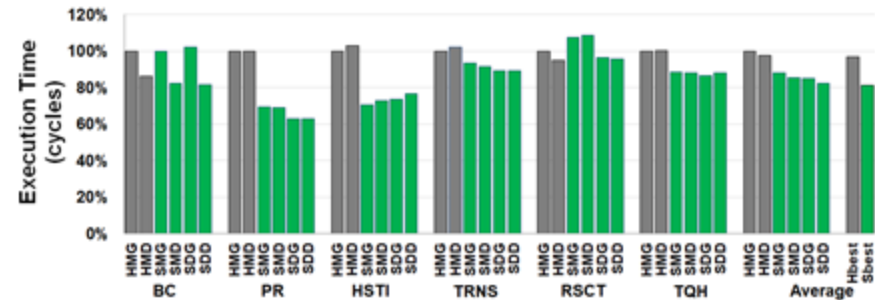


Key Components

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

	Request	Generated for
Read	ReqV	Self-invalidating read
	ReqS	Writer-invalidated read
Write	ReqWT	Write-through store
	ReqO	Write-only ownership store
Read+Write	ReqWT+data	Atomic for WT cache
	ReqO+data	Read-for-ownership store, Atomic for ownership cache
Writeback	ReqWB	Owned data eviction

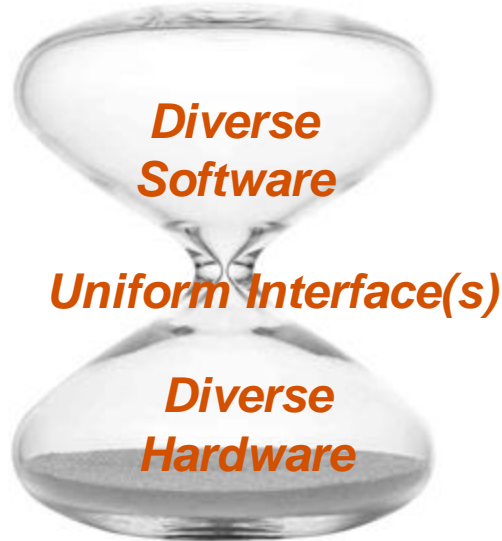
+ granularity



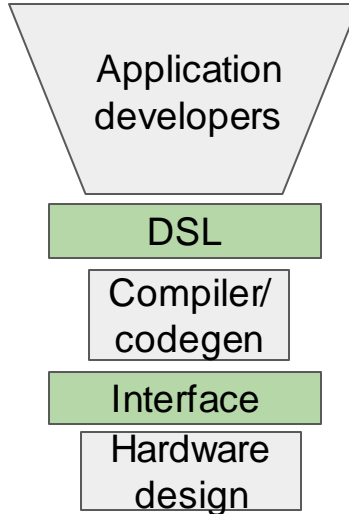
Alsop et al. ISCA'18

The Three Interface Views Together

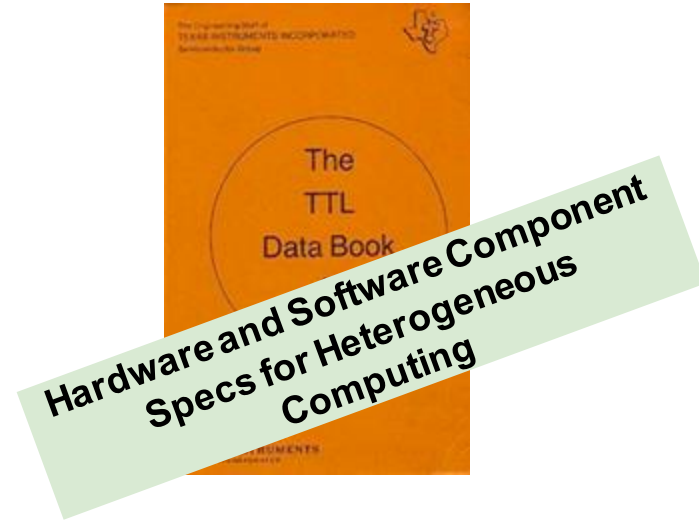
Uniform Interface



Codesigned stack



Catalog of Parts

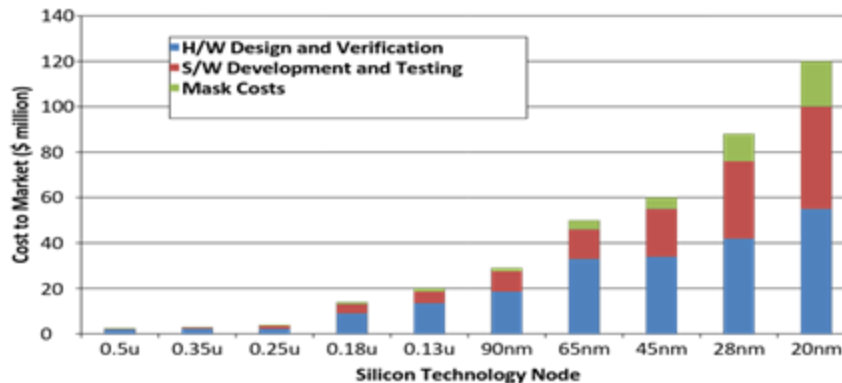
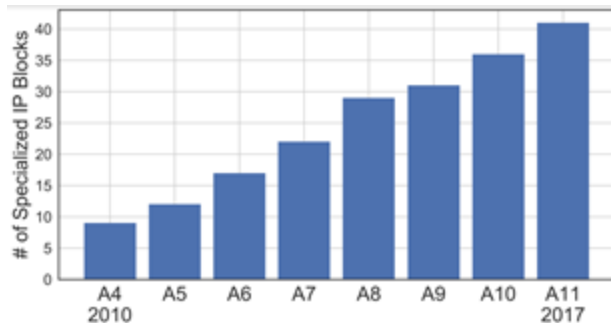


Zero hour software bring up + Rapid HW-SW codesign + Machine checked plug and play

Unlock usable specialization for embedded devices to planetary scale computing

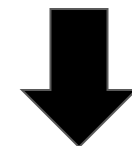
Address performance, efficiency, portability, HW & SW design productivity, verifiability, security

Measuring Success

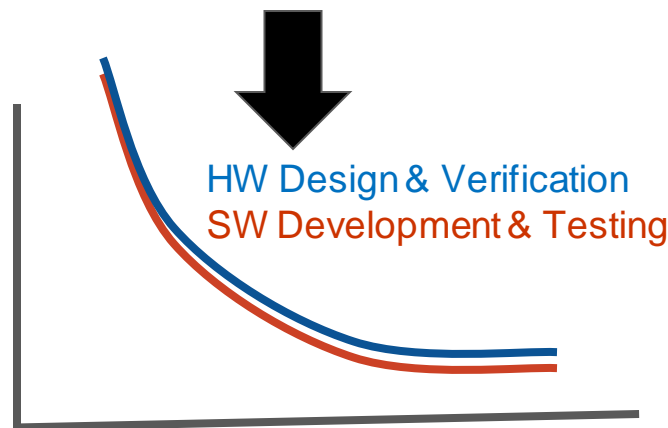
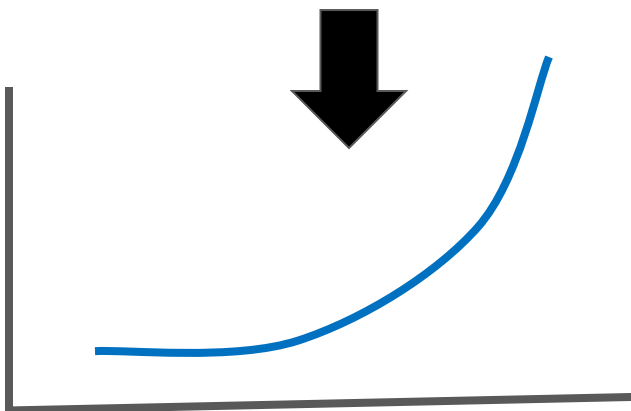


Time to Market (HW+SW)

Months to Years



Days to Weeks



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

Post-Moore Imperatives

- Interface
- **Communication**
- Approximation

Application-driven

Communication

Communication critical to efficiency

- **Memory structures**
- **Data movement**
- **Coherence**
- **Consistency**
- **Synchronization**

Accelerator Communication Interface

SoC will have many programmable and fixed fn accelerators

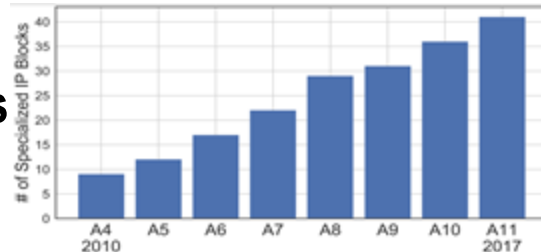
Each accelerator w/ specialized memory system

- Caches, buffers, persistent storage, ...

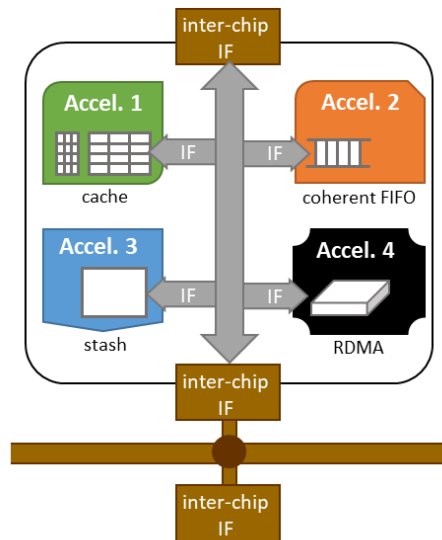
How to connect accelerators efficiently in programmable way?

- Unified address space, messaging, other models?
- Specialize communication, coherence, consistency?
- What is the interface?

Need accelerator communication interface for hardware
Abstract to software interface



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

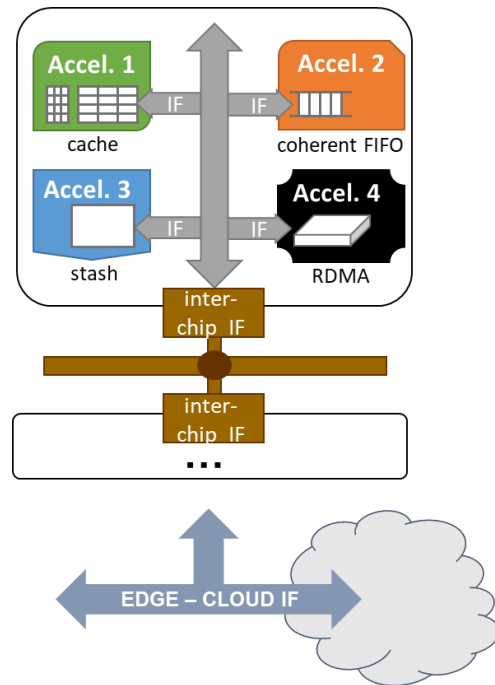


Accelerator Communication Interface: Not Just SoC

Chip scale to planetary scale acceleration

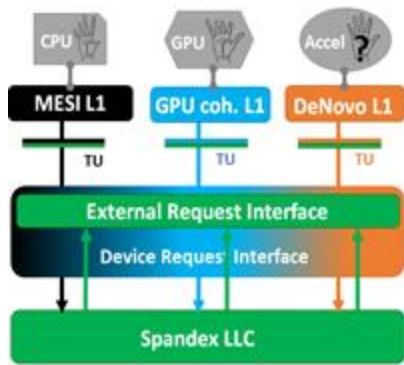
- Edge to cloud as a system

Challenge: Frictionless distribution of work and communication across diverse scales



Spandex Accelerator Communication Interface

- **Talk tomorrow: Spandex interface for SoC**
 - Seamless integration of devices with heterogeneous coherence
 - Dynamic application-customized coherence specialization
 - Consistency model: Familiar data-race-free



Key Components

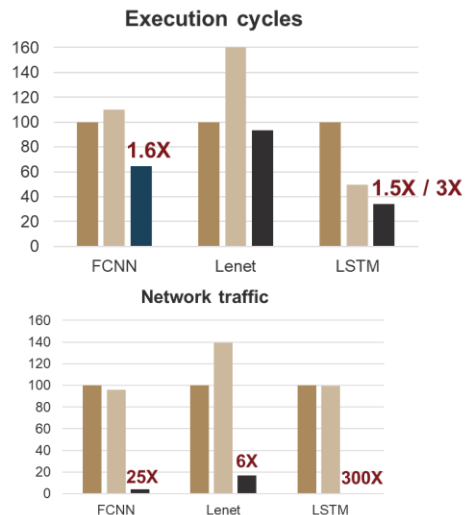
Flexible device request interface

External request interface

DeNovo-based LLC

	Request	Generated for
Read	ReqV	Self-invalidating read
	ReqS	Writer-invalidated read
Write	ReqWT	Write-through store
	ReqO	Write-only ownership store
Read+Write	ReqWT+data	Atomic for WT cache
	ReqO+data	Read-for-ownership store, Atomic for ownership cache
Writeback	ReqWB	Owned data eviction

+ granularity



Post-Moore Imperatives

- Interface
- Communication
- Approximation

Application-driven

Approximation

Judiciously exploiting approximations can provide orders of magnitude efficiency benefit with little impact on quality of solution

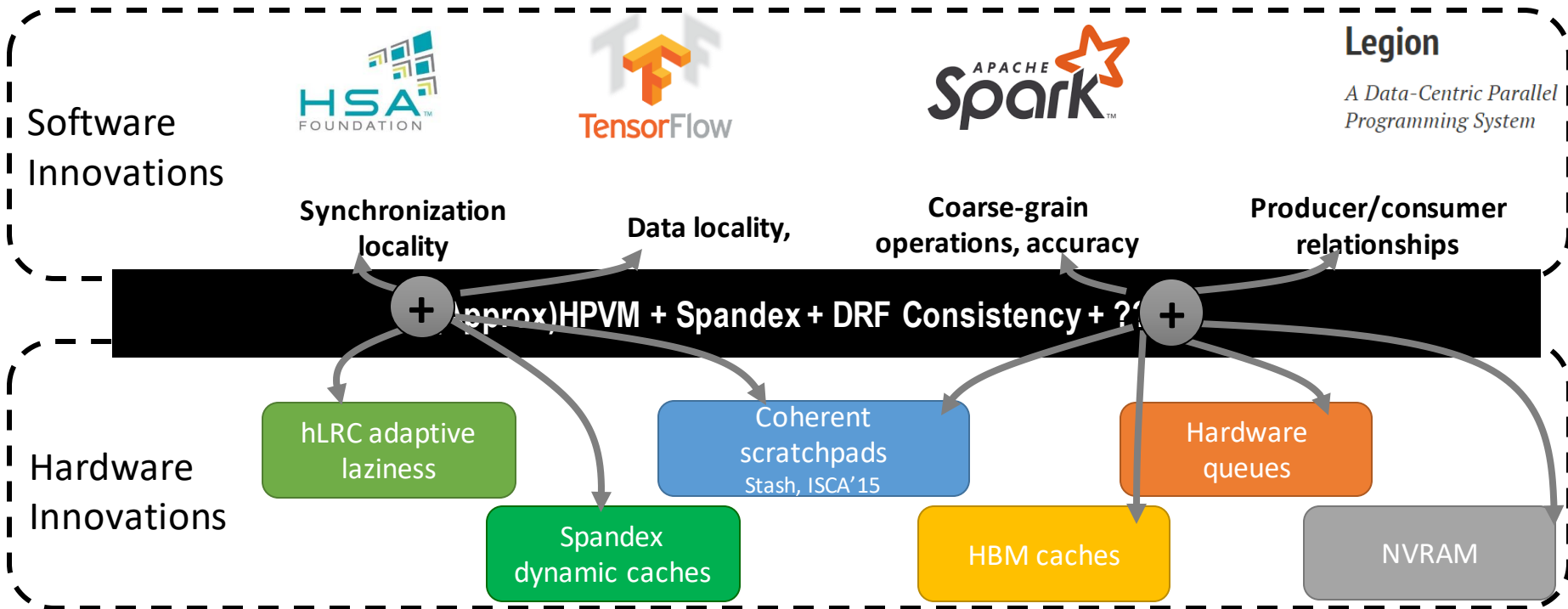
But how to express quality of solution from the application to the hardware?

- Talk tomorrow on **ApproxHPVM** [OOPSLA'19]

How to understand impact of errors?

- Gem5-Approxilyzer [DSN'19]
- HarDNN: Quantifying impact of errors on DNNs [in review]

Looking Forward...



Interface, communication, approximation

Application-driven: See AR/VR benchmark suite talk tomorrow