Programming Systems for Specialized Architectures

Interface, Data, Approximation

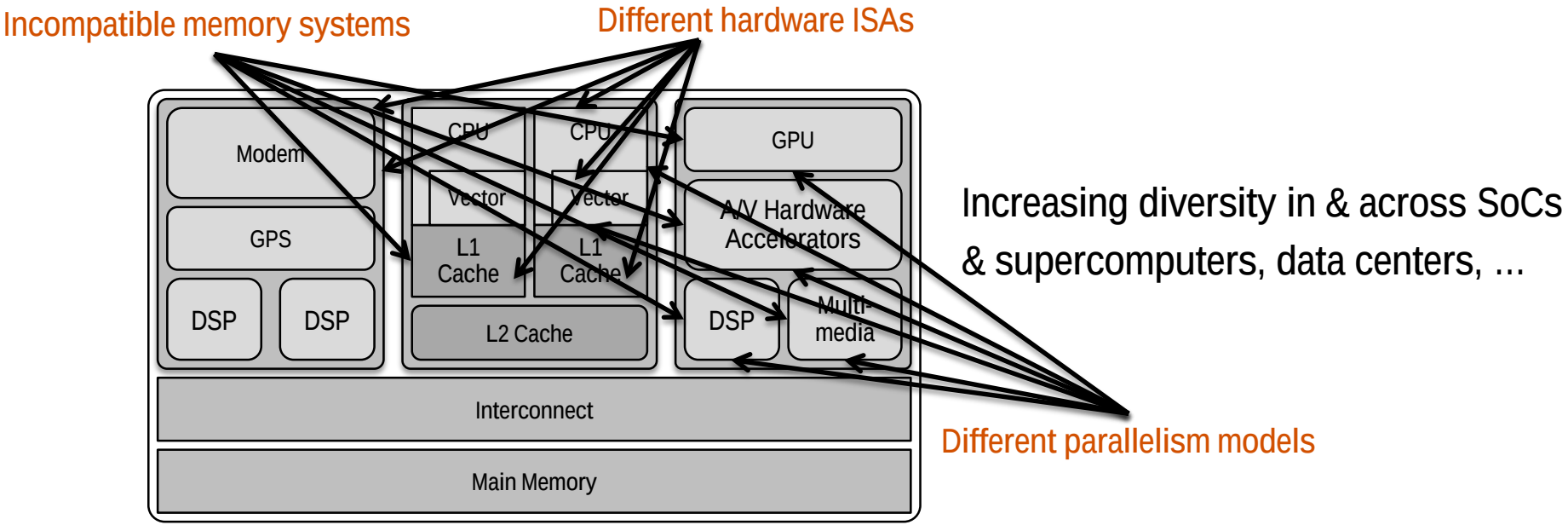
Sarita Adve

With: Vikram Adve, Johnathan Alsop, Maria Kotsifakou, Sasa Misailovic, Matt Sinclair, Prakalp Srivastava

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

Sponsors: NSF, C-FAR, ADA (JUMP center by SRC, DARPA)

A Modern Mobile SoC



Need common interface (abstractions): HW-independent SW development, “object code” portability

Data movement critical: Memory structures, communication, consistency, synchronization

Approximation: Application-driven solution quality trade off to increase efficiency

Interfaces: Back to the Future

April 7, 1964: IBM announced the 360

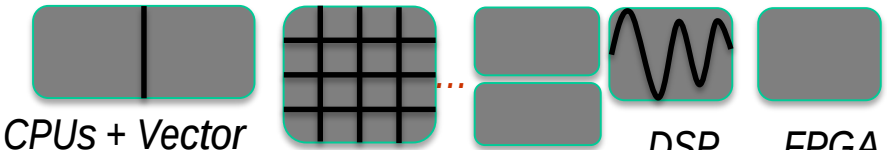
- Family of machines w/ common abstraction/interface/ISA
  - Programmer freedom: no reprogramming
  - Designer freedom: implementation creativity

Not unique

- CPUs : ISAs; Internet : IP; GPUs : CUDA; Databases : SQL; …

Current Interface Levels

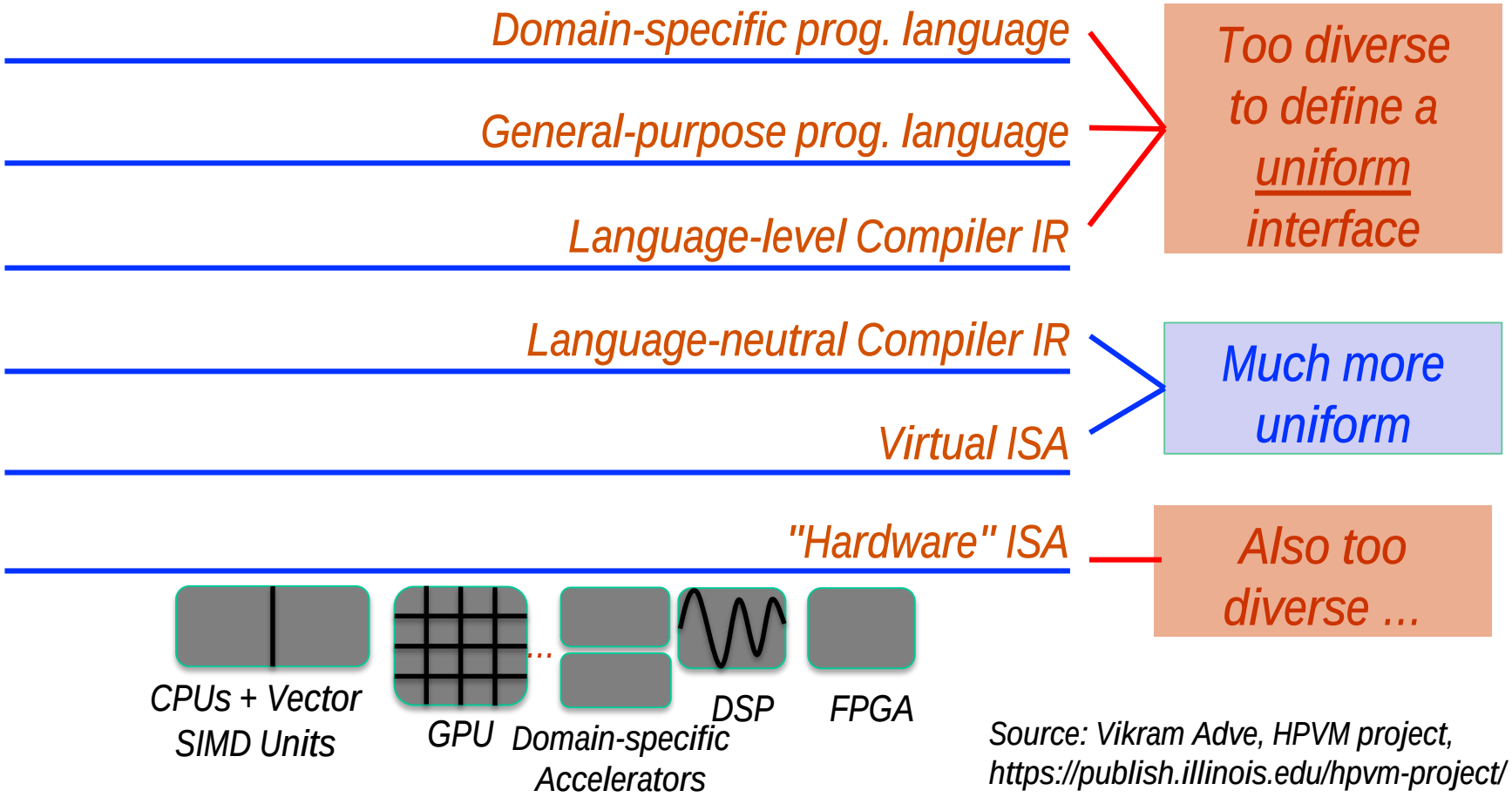
App. productivity	*Domain-specific prog. language*	*TensorFlow, MXNet, Halide, …*
App. performance	*General-purpose prog. language*	*CUDA, OpenCL, OpenAcc, OpenMP, Python, Julia*
Language innovation	*Language-level Compiler IR*	*Delite DSL IR, DLVM, TVM, …*
Compiler investment	*Language-neutral Compiler IR*	*Delite IR, HPVM, OSCAR, Polly*
Object-code portability	*Virtual ISA*	*SPIR, HPVM*
Hardware innovation	*"Hardware" ISA*	*IBM AS/400, Transmeta, PTX, HSAIL, Codesigned Virtual Machines*



CPUs + Vector SIMD Units *GPU* *Domain-specific Accelerators* *DSP* *FPGA*

Source: Vikram Adve, HPVM project, https://publish.illinois.edu/hpvm-project/

Which Interface Levels Can Be Uniform?



Source: Vikram Adve, HPVM project, https://publish.illinois.edu/hpvm-project/

One Example

HPVM: Heterogeneous Parallel Virtual Machine [PPoPP'18]

Parallel program representation for heterogeneous parallel hardware

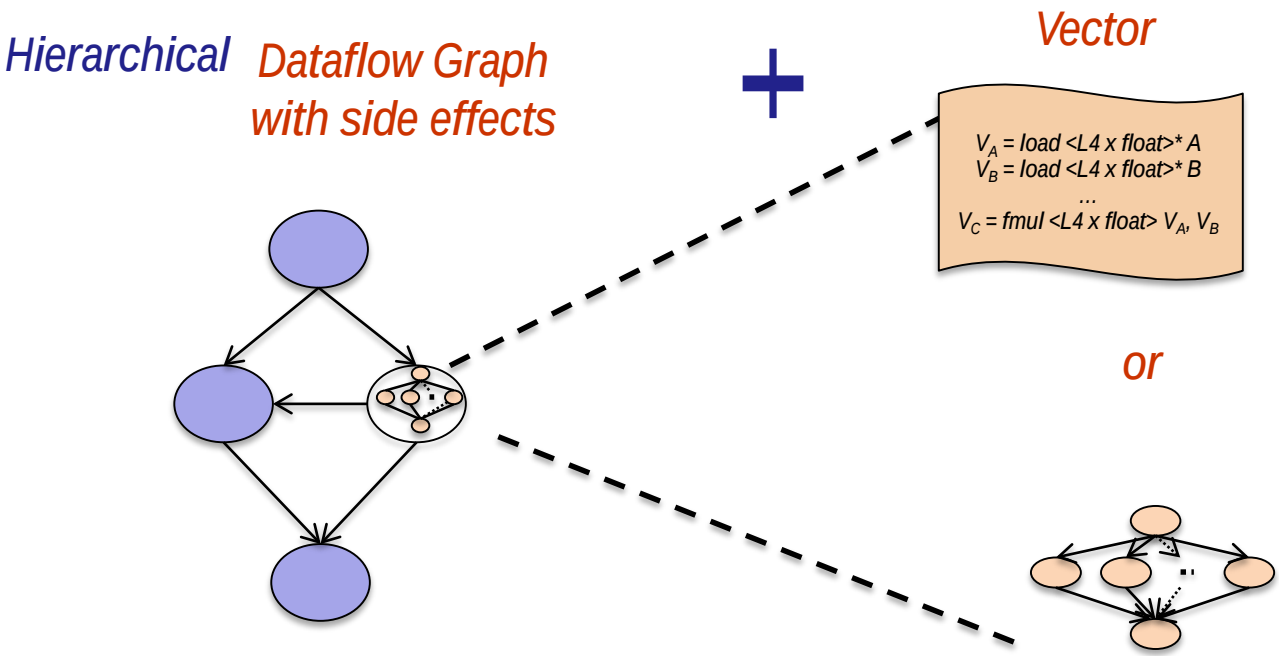
- Virtual ISA: portable virtual object code, simpler translators
- Compiler IR: optimizations, map diverse parallel languages
- Runtime Representation for flexible scheduling: mapping, load balancing

Generalization of LLVM IR for parallel heterogeneous hardware

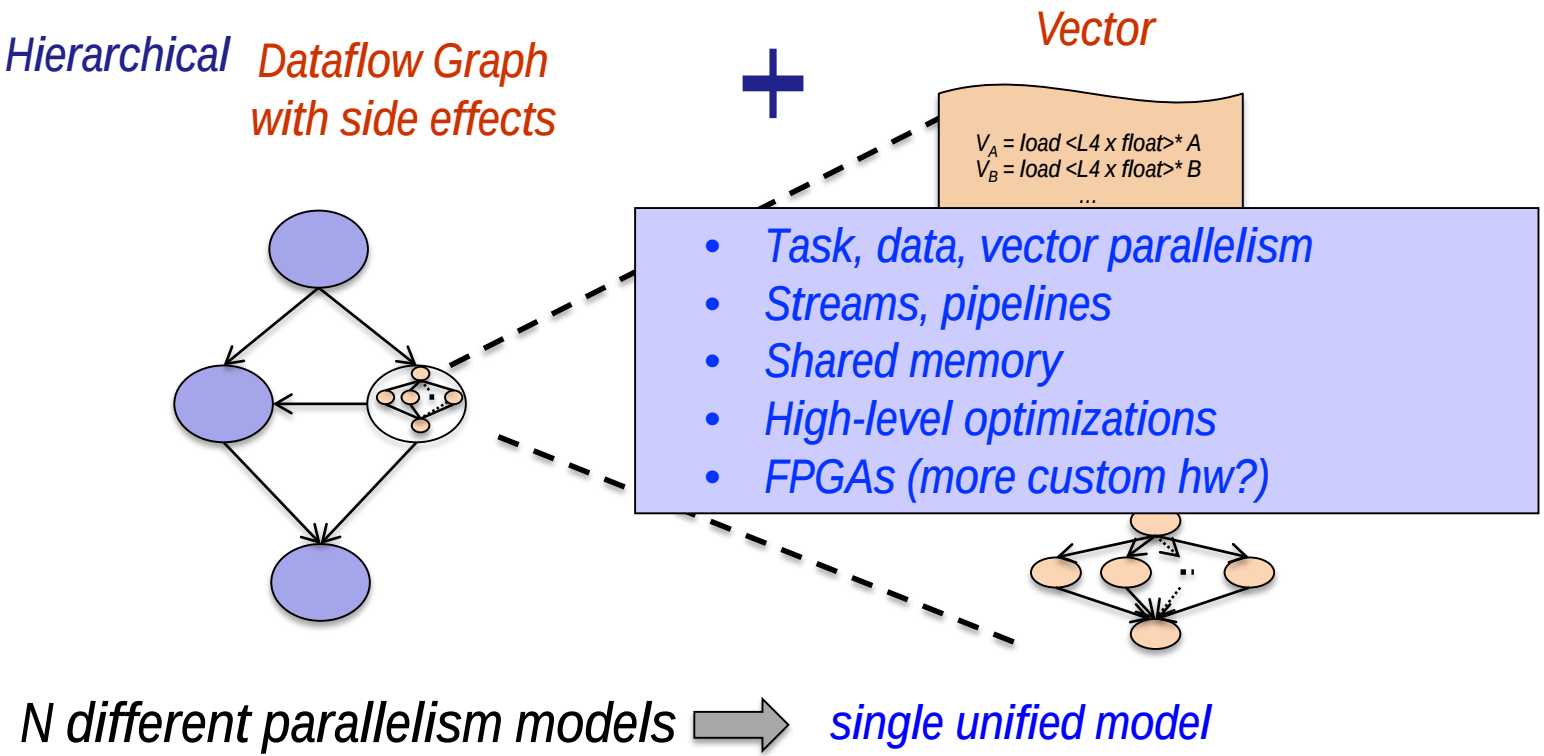
PPoPP'18: Results on GPU (Nvidia), Vector ISA (AVX), Multicore (Intel Xeon)

Ongoing: FPGA, novel domain-specific SoCs

HPVM Abstractions



HPVM Abstractions



N different parallelism models ⇒ *single unified model*

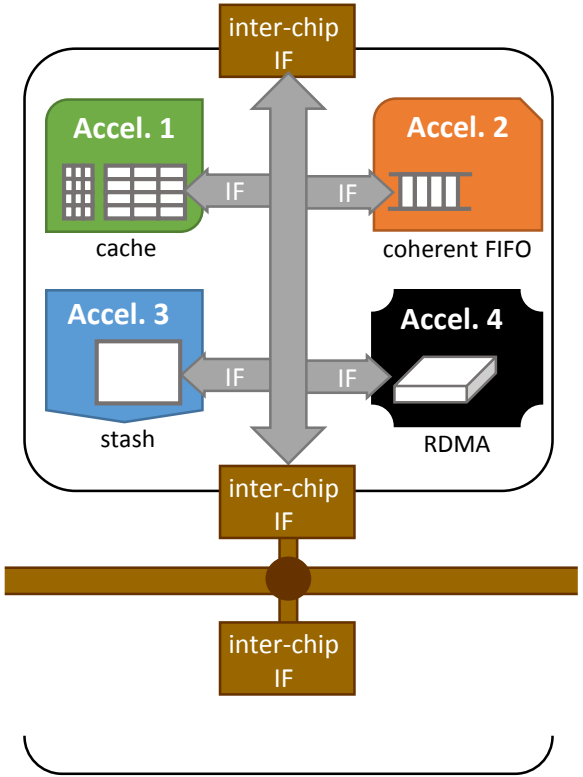
Data

Data movement critical to efficiency

- Memory structures
- Communication
- Coherence
- Consistency
- Synchronization

Uniform communication interface for hardware

Abstract to software interface



Application-Customized Accelerator Communication Arch

Problem: Design + Integrate
Multiple accelerator memory systems + Communication

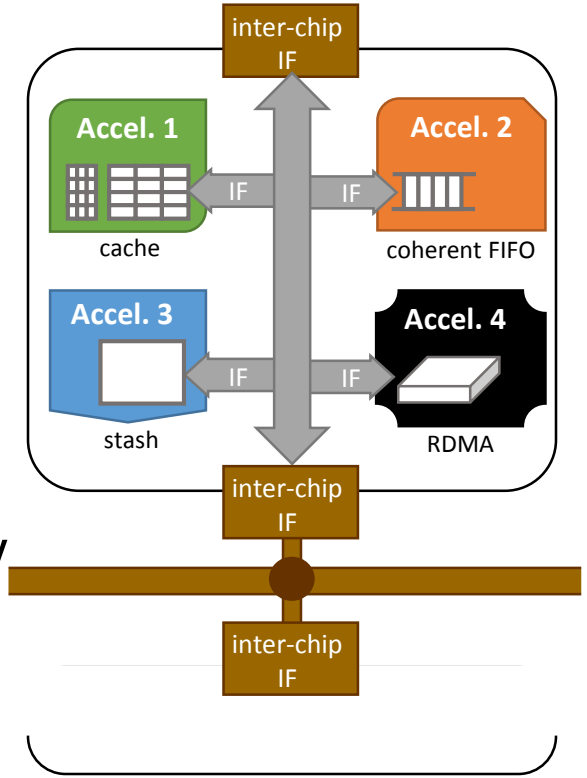
Challenges:

- Friction between different app-specific specializations
- Inefficiencies due to deep memory hierarchy
- Multiple scales: on-chip to cloud

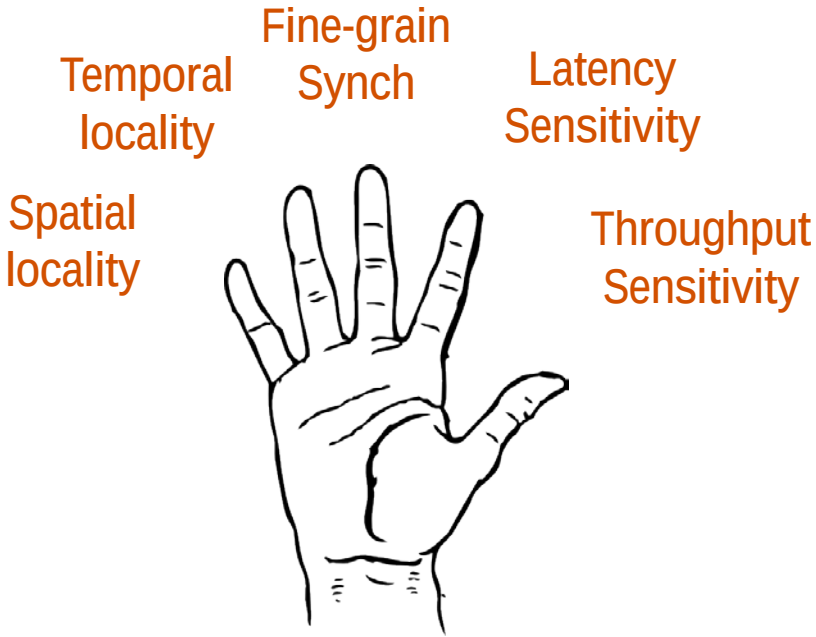
New accelerator communication architecture

- Coherent, global address space
- App-specialized coherence, comm, storage, soln quality

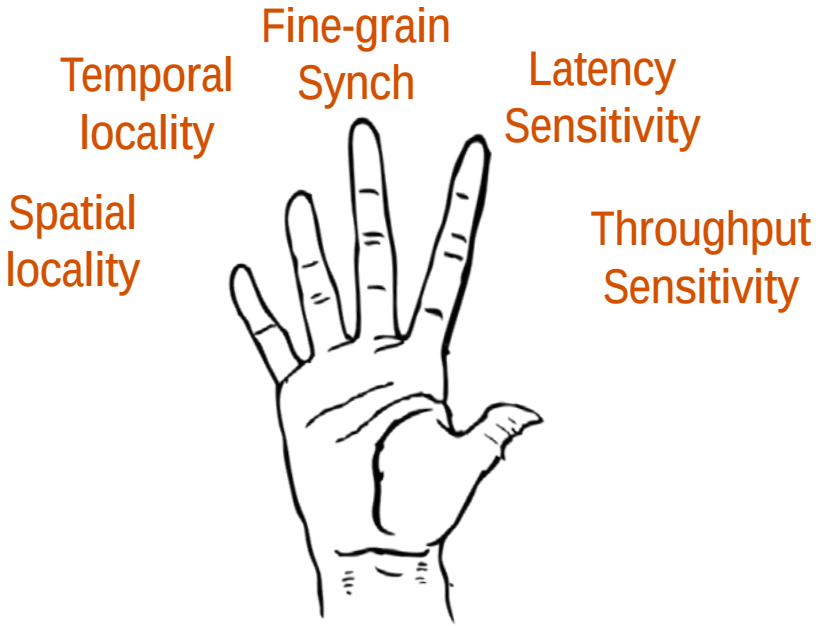
One example next focused on coherence: Spandex [ISCA'18]



Heterogeneous devices have diverse memory demands

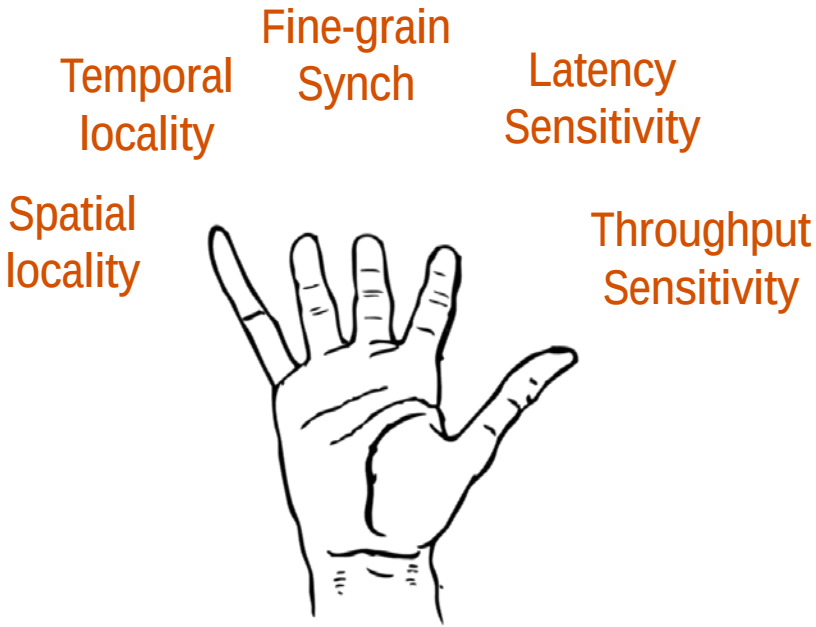


Heterogeneous devices have diverse memory demands



Typical CPU workloads:
fine-grain synch, latency sensitive

Heterogeneous devices have diverse memory demands



Typical GPU workloads:
spatial locality, throughput sensitive

MESI coherence targets CPU workloads

Protocol properties	MESI
Granularity	Line
Stale data invalidation	Writer-invalidate
Write propagation	Ownership

Good for:

CPU

MESI

- Coarse-grain state
  - ✓ Spatial locality
  - ✗ False sharing
- Writer-initiated invalidation
  - ✓ Temporal locality for reads
  - ✗ Overheads limit throughput, scalability
- Ownership-based updates
  - ✓ Temporal locality for writes
  - ✗ Indirection if low locality

GPU coherence fits GPU workloads

Protocol properties	MESI	GPU coherence
Granularity	Line	Reads: line writes: word
Stale data invalidation	Writer-invalidate	Self-invalidate
Write propagation	Ownership	Write-through

Good for:

CPU

GPU

GPU Coherence

- Fine-grain writes
  - ✓ No false sharing
  - ✗ Reduced spatial locality
- Self invalidation
  - ✓ Simple, scalable
  - ✗ Synch limits read reuse
- Write-through caches
  - ✓ Simple, low overhead
  - ✗ Synch limits write reuse

DeNovo is good fit for CPU and GPU

Protocol properties	MESI	GPU coherence	DeNovo
Granularity	Line	Reads: line writes: word	Reads: flexible Writes: word
Stale data invalidation	Writer-invalidate	Self-invalidate	Self-invalidate
Write propagation	Ownership	Write-through	Ownership

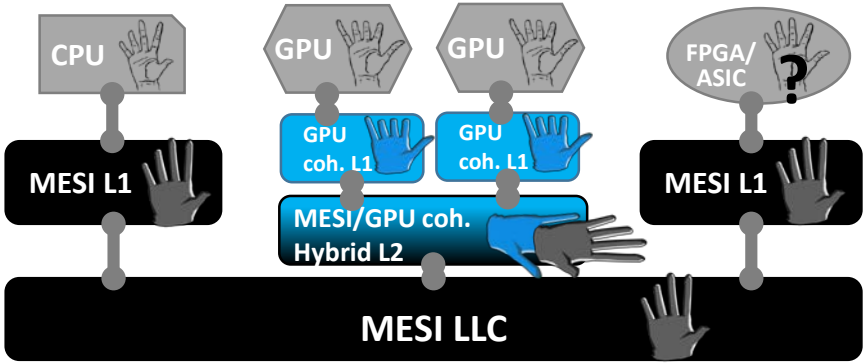
Good for:

CPU

GPU

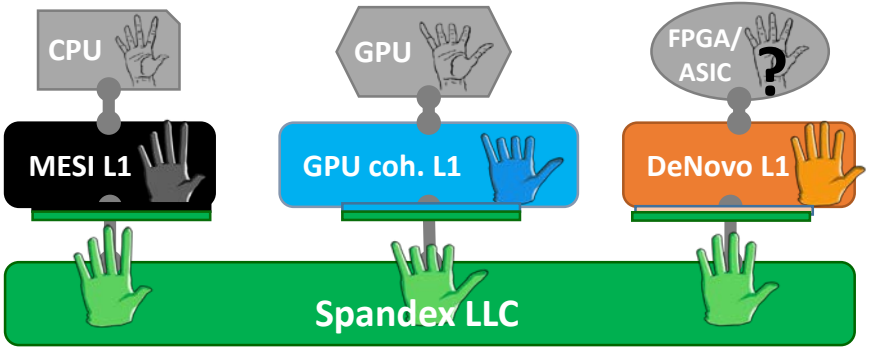
CPU or GPU

Integrating Diverse Coherence Strategies



Existing Solutions: MESI-based LLC

- Accelerator Requests forced to use MESI
- Added latency for inter-device communication
- MESI is complex: extensions are difficult



Spandex: DeNovo-based interface [ISCA'18]

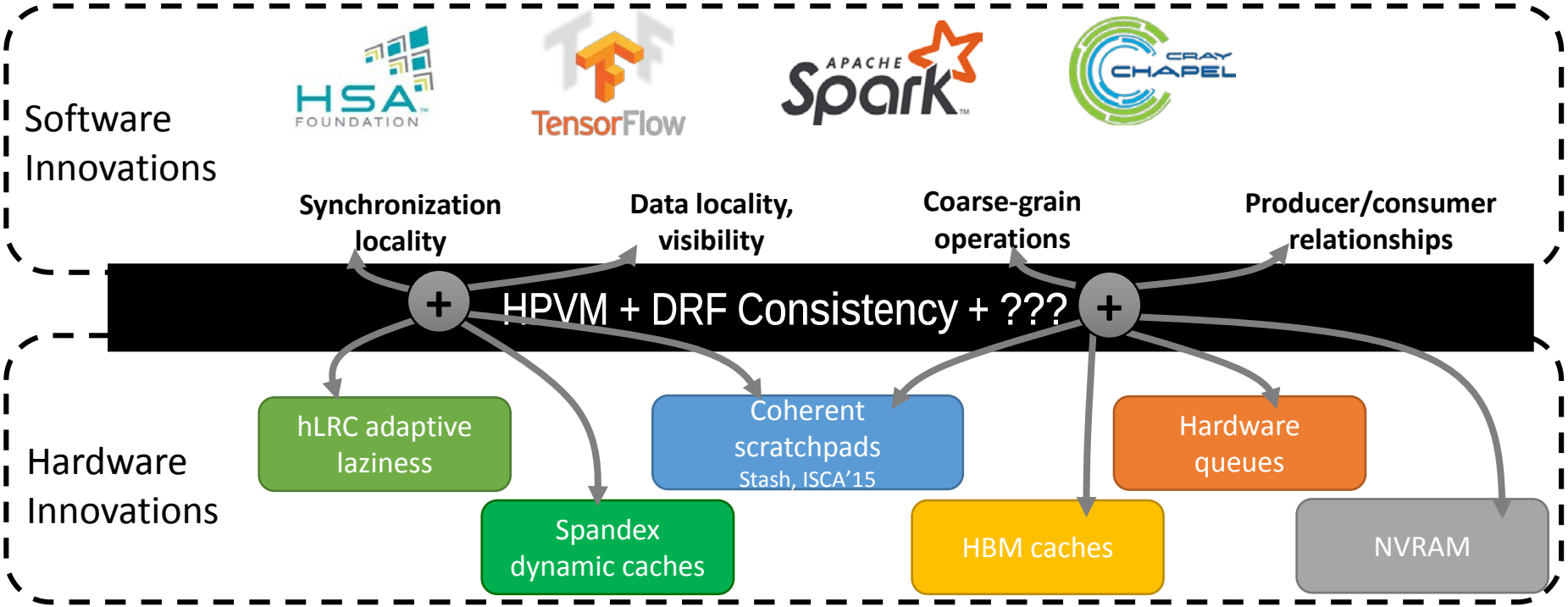
- Supports write-through and write-back
- Supports self-invalidate and writer-invalidate
- Supports requests of variable granularity
- Directly interfaces MESI, GPU coherence, hybrid (e.g. DeNovo) caches

Example: Collaborative Graph Applications

Vertex-centric algorithms: distribute vertices among CPU, GPU threads

Application	Access Pattern	Important Dimension	Results
Pull-based PageRank	Read neighbor vertices, Update local vertex	**Flat LLC** avoids indirection for read misses	Spandex LLC ⇒ **37%** better exec. time **9%** better NW traffic
Push-based Betweenness Centrality	Read local vertex, Update (RMW) neighbor vertices	**Ownership-based write propagation** exploits locality in updates	DeNovo at GPU ⇒ **18%** better exec. time **61%** better NW traffic

Looking Forward…



Approximation

How to express quality of solution from the application to the hardware?

Integrate approximation (quality) into the interface

Summary

- Interfaces
- Data
- Approximation