DeNovo: A Software-Driven Rethinking of the Memory Hierarchy

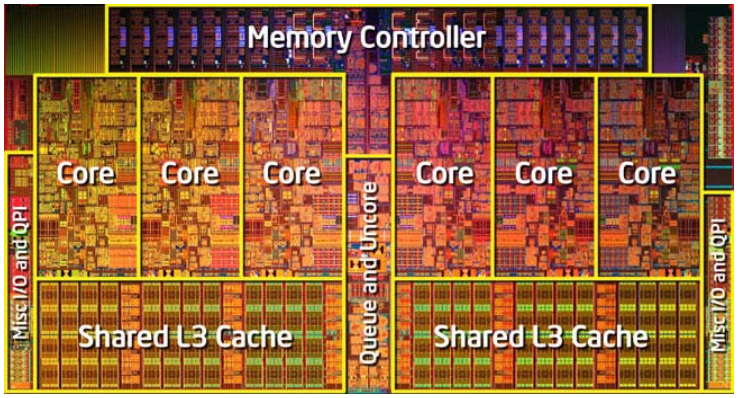
Sarita Adve, Vikram Adve,
Rob Bocchino, Byn Choi, Nima Honarmand, Rakesh Komuravelli, Maria Kotsifakou,
Matthew Sinclair, Robert Smolinski, Prakalp Srivastava, Hyojin Sung
University of Illinois at Urbana-Champaign

Nicholas Carter, Ching-Tsun Chou, Pablo Montesinos,
Tatiana Schpeisman, Adam Welc
Intel, Qualcomm

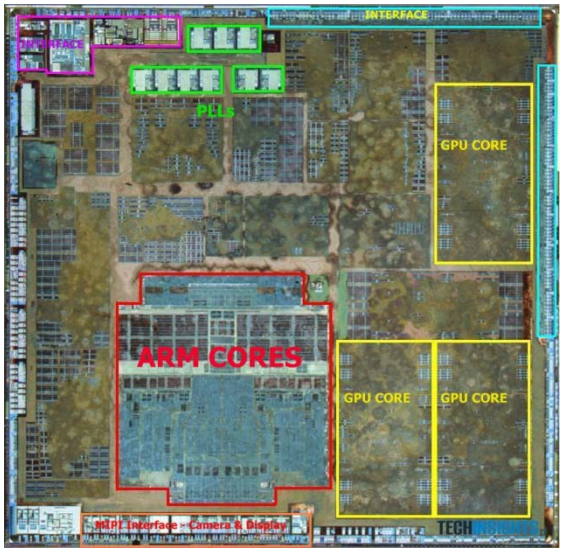
denovo@cs.illinois.edu

Silver Bullets for the Energy Crisis?

Parallelism



Specialization, heterogeneity, …



BUT large impact on

- Software
- Hardware
- Hardware-Software Interface

Multicore Parallelism: Current Practice

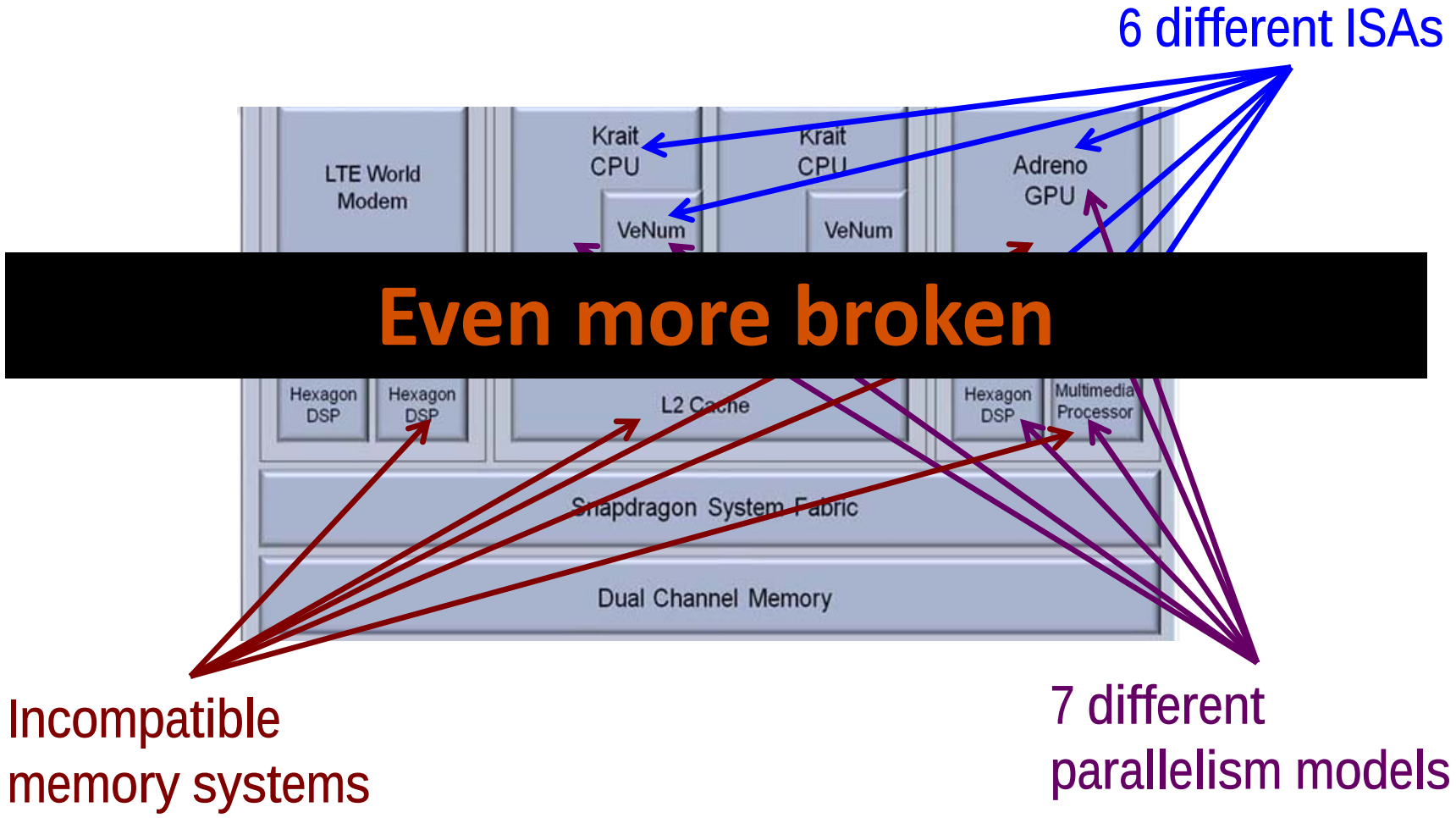
- Multicore parallelism today: shared-memory
  - Complex, power- and performance-inefficient hardware
    - Complex directory coherence, unnecessary traffic, ...
  - Difficult programming model
    - Data races, non-determinism, composability?, testing?
  - Mismatched interface between HW and SW, a.k.a memory model
    - Can't specify "what value can read return"
    - Data races defy acceptable semantics

Fundamentally broken for hardware & software

Specialization/Heterogeneity: Current Practice

A modern smartphone

CPU, GPU, DSP, Vector Units, Multimedia, Audio-Video accelerators



Energy Crisis Demands Rethinking HW, SW

How to (co-)design

- Software? *Deterministic Parallel Java (DPJ)*
- Hardware? *DeNovo*
- HW / SW Interface? *Virtual Instruction Set Computing (VISC)*

First focus on (homogeneous) parallelism

Multicore Parallelism: Current Practice

- Multicore parallelism today: shared-memory
  - Complex, power- and performance-inefficient hardware
    - Complex directory coherence, unnecessary traffic, ...
  - Difficult programming model
    - Data races, non-determinism, composability?, testing?
  - Mismatched interface between HW and SW, a.k.a memory model
    - Can't specify "what value can read return"
    - Data races defy acceptable semantics

Fundamentally broken for hardware & software

Multicore Parallelism: Current Practice

- Multicore parallelism today: shared-memory
  - Complex, power- and performance-inefficient hardware
    - Complex directory coherence, unnecessary traffic, …
  - Difficult programming model
    - Data races, non-determinism, composability?, testing?
  - Mismatched interface between HW and SW, a.k.a memory model
    - Can't specify "what value can read return"
    - Data races defy acceptable semantics

Banish shared memory?

Fundamentally broken for hardware & software

Multicore Parallelism: Current Practice

- Multicore parallelism today: shared-memory
  - Complex, power- and performance-inefficient hardware
    - Complex directory coherence, unnecessary traffic, …
  - Difficult programming model
    - Data races, non-determinism, composability?, testing?
  - Mismatched interface between HW and SW, a.k.a memory model
    - Complex, inefficient, and error prone
    - Data races defy acceptable semantics

Banish wild shared memory

Need disciplined shared memory!

Fundamentally broken for hardware & software

What is Shared-Memory?

Shared-Memory =

Global address space

+

Implicit, anywhere communication, synchronization

What is Shared-Memory?

Shared-Memory =

Global address space

+

Implicit, anywhere communication, synchronization

What is Shared-Memory?

Wild Shared-Memory =

Global address space

+

Implicit, anywhere communication, synchronization

What is Shared-Memory?

Wild Shared-Memory =

Global address space

+

~~Implicit, anywhere communication, synchronization~~

What is Shared-Memory?

Disciplined Shared-Memory =

Global address space

+

~~Implicit, anywhere communication, synchronization~~

Explicit, structured side-effects

Our Approach

Strong safety properties - Deterministic Parallel Java (DPJ)

- No data races, determinism-by-default, safe non-determinism
- Simple semantics, safety, and composability

explicit effects + structured parallel control

Disciplined Shared Memory

Efficiency: complexity, performance, power - DeNovo

- Simplify coherence and consistency
- Optimize communication and storage layout

Simple programming model AND

Complexity, performance-, power-scalable hardware

Current Hardware Limitations

- Complexity
  - Subtle races and numerous transient states in the protocol
  - Hard to verify and extend for optimizations
- Storage overhead
  - Directory overhead for sharer lists
- Performance and power inefficiencies
  - Invalidation, ack messages
  - Indirection through directory
  - False sharing (cache-line based coherence)
  - Bandwidth waste (cache-line based communication)
  - Cache pollution (cache-line based allocation)

Results *for Deterministic and Safe Non-Det Codes*

- Complexity
  - No transient states
  - Simple to extend for optimizations

Base DeNovo
20X faster to verify vs. MESI

- Storage overhead
  - Directory overhead for sharer lists

- Performance and power inefficiencies
  - Invalidation, ack messages
  - Indirection through directory
  - False sharing (cache-line based coherence)
  - Bandwidth waste (cache-line based communication)
  - Cache pollution (cache-line based allocation)

Results *for Deterministic and Safe Non-Det Codes*

- Complexity
  - No transient states
  - Simple to extend for optimizations

Base DeNovo
20X faster to verify vs. MESI

- Storage overhead
  - No storage overhead for directory information

- Performance and power inefficiencies
  - Invalidation, ack messages
  - Indirection through directory
  - False sharing (cache-line based coherence)
  - Bandwidth waste (cache-line based communication)
  - Cache pollution (cache-line based allocation)

Results *for Deterministic and Safe Non-Det Codes*

- Complexity
  - No transient states
  - Simple to extend for optimizations

Base DeNovo
20X faster to verify vs. MESI

- Storage overhead
  - No storage overhead for directory information

- Performance and power ineffic
  - No invalidation, ack messages
  - No indirection through director
  - No false sharing: region based coherence
  - Region, not cache-line, communication
  - Region, not cache-line, allocation (ongoing)

Up to 79% lower memory stall time
Up to 72% lower traffic

DPJ Overview

- Deterministic-by-default parallel language
  - Extension of sequential Java
  - Structured parallel control: nested fork-join
  - Novel region-based type and effect system
  - Speedups close to hand-written Java
  - Expressive enough for irregular, dynamic parallelism
- Supports disciplined non-determinism
  - Explicit, data race-free, isolated
  - Non-deterministic, deterministic code co-exist safely (composable)
- Ongoing work addresses unanalyzable parallelism/effects
- Focus on deterministic codes for next few slides

DPJ Overview

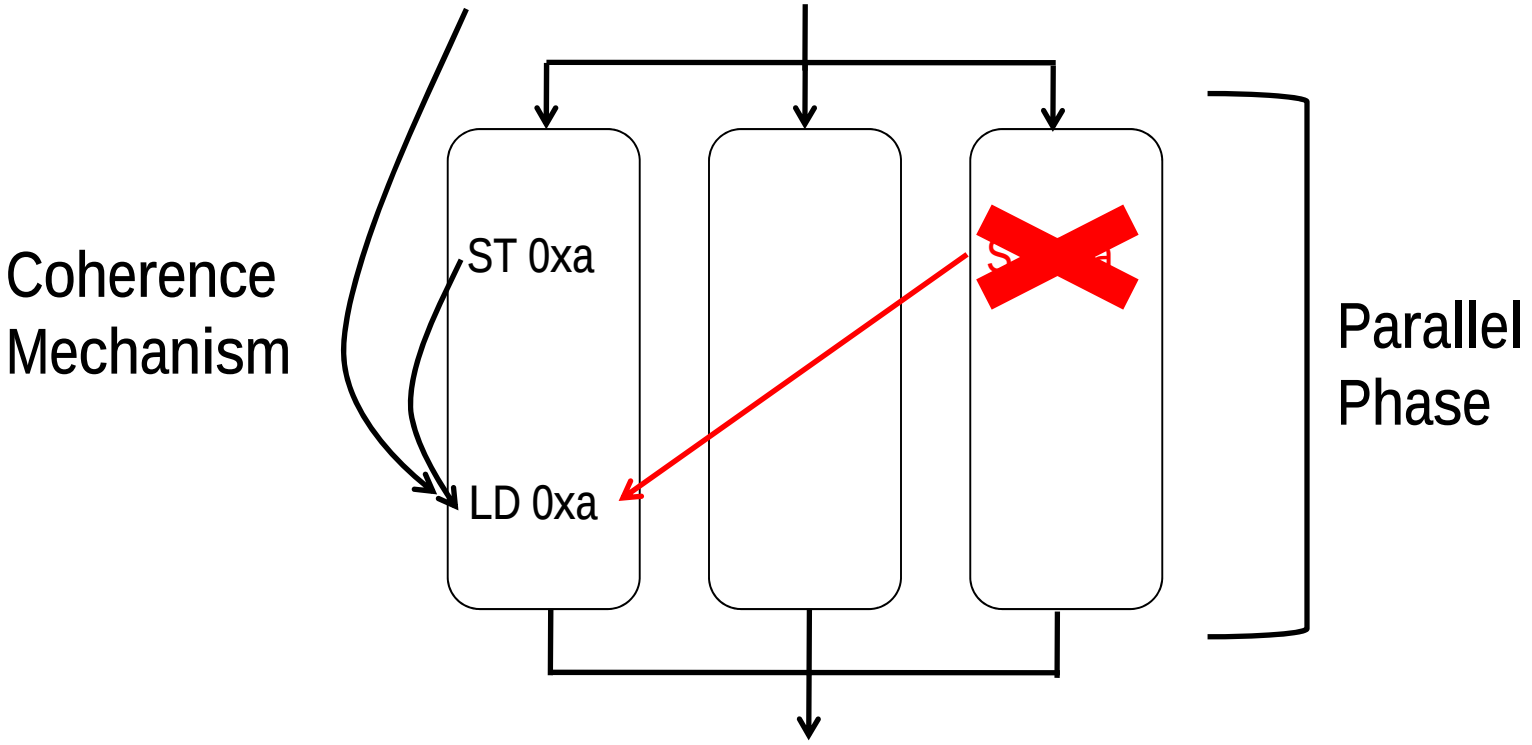
- Deterministic-by-default parallel language [OOPSLA'09]
  - Extension of sequential Java
  - Structured parallel control: nested fork-join
  - Novel region-based type and effect system
  - Speedups close to hand-written Java
  - Expressive enough for irregular, dynamic parallelism
- Supports disciplined non-determinism [POPL'11]
  - Explicit, data race-free, isolated
  - Non-deterministic, deterministic code co-exist safely (composable)
- Ongoing work addresses unanalyzable parallelism/effects
- Focus on deterministic codes for next few slides

Regions and Effects

- Region: a name for a set of memory locations
  - Programmer assigns a region to each field and array cell
  - Regions partition the heap
- Effect: a read or write on a region
  - Programmer summarizes effects of method bodies
- Compiler checks that
  - Region types are consistent, effect summaries are correct
  - Parallel tasks are non-interfering (no conflicts)
  - Simple, modular type checking (no inter-procedural ….)
- Programs that type-check are guaranteed determinism-by-default

Memory Consistency Model

- Guaranteed determinism
  ⇒ Read returns value of *last* write in sequential order
  1. Same task in this parallel phase
  2. Or before this parallel phase

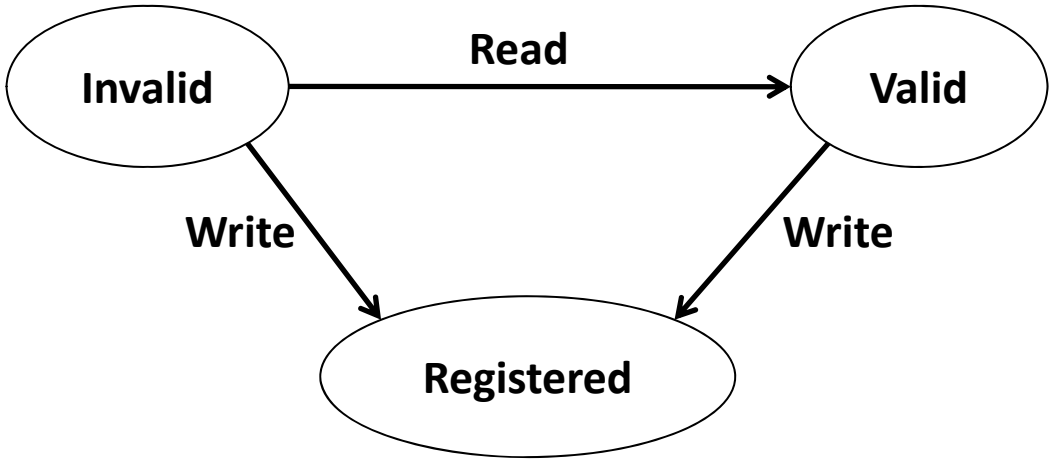


Cache Coherence

- Coherence Enforcement
  1. Invalidate stale copies in caches
  2. Track one up-to-date copy
- Explicit effects
  - Compiler knows all regions written in this parallel phase
  - Cache can self-invalidate before next parallel phase
    - Invalidates data in writeable regions not accessed by itself
- Registration
  - Directory keeps track of one up-to-date copy
  - Writer updates before next parallel phase

Basic DeNovo Coherence [PACT'11]

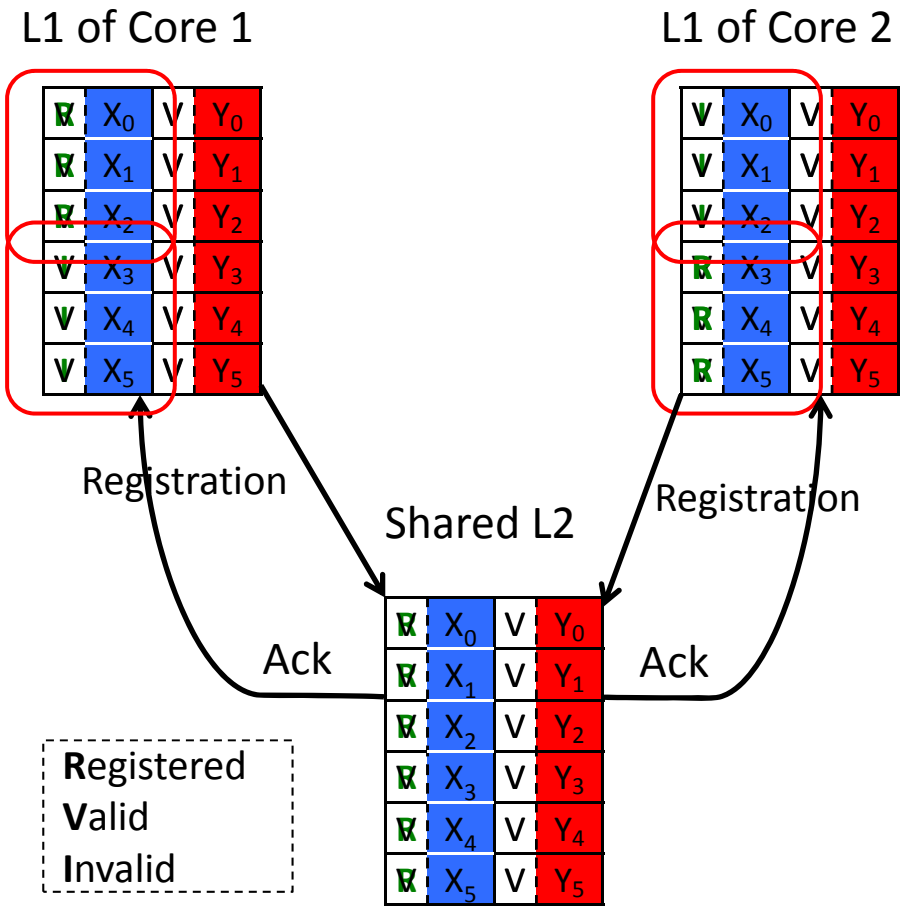
- Assume (for now): Private L1, shared L2; single word line
  - Data-race freedom at word granularity
- L2 data arrays double as ~~directory~~ registry
  - Keep valid data or registered core id, no space overhead
- L1/L2 states



- *Touched* bit set only if read in the phase

Example Run

```
class S_type {
      X in DeNovo-region ■   ;
      Y in DeNovo-region ■;
}
S _type S[size];
...
Phase1 writes ■  {  // DeNovo effect
      foreach  i in 0, size {
            S[i].X = …;
      }
      self_invalidate( ■ );
}
```



Current Hardware Limitations

- Complexity
  - ✔ – Subtle races and numerous transient sates in the protocol
  - – Hard to extend for optimizations
- Storage overhead
  - ✔ – Directory overhead for sharer lists
- Performance and power inefficiencies
  - ✔ – Invalidation, ack messages
  - – Indirection through directory
  - ✔ – False sharing (cache-line based coherence)
  - – Traffic (cache-line based communication)
  - – Cache pollution (cache-line based allocation)

Flexible, Direct Communication

Insights

1. Traditional directory must be updated at every transfer

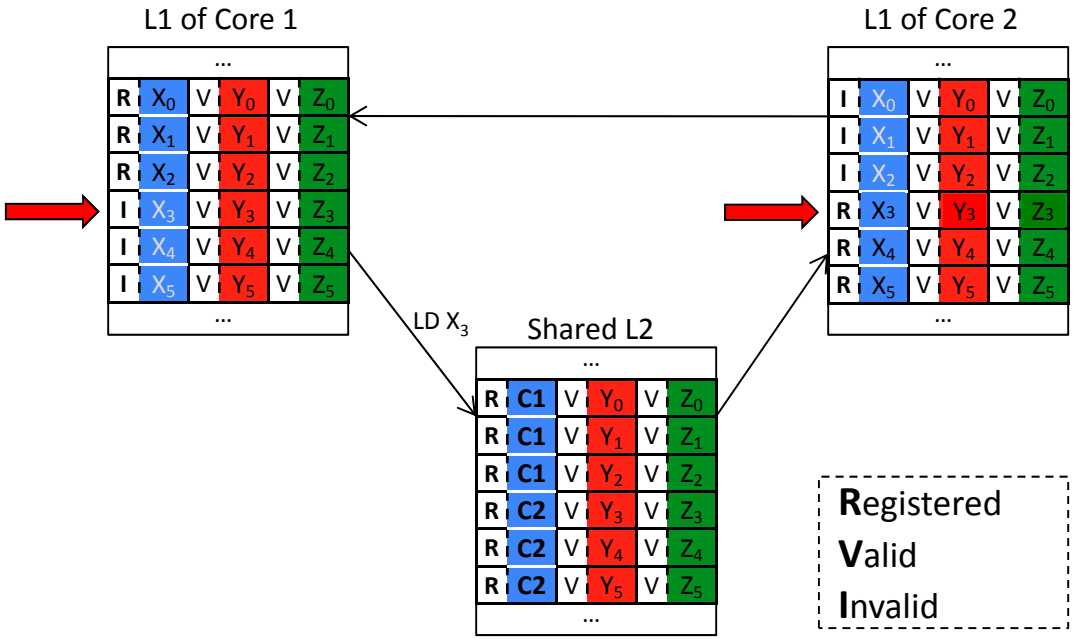
   DeNovo can copy valid data around freely

2. Traditional systems send cache line at a time

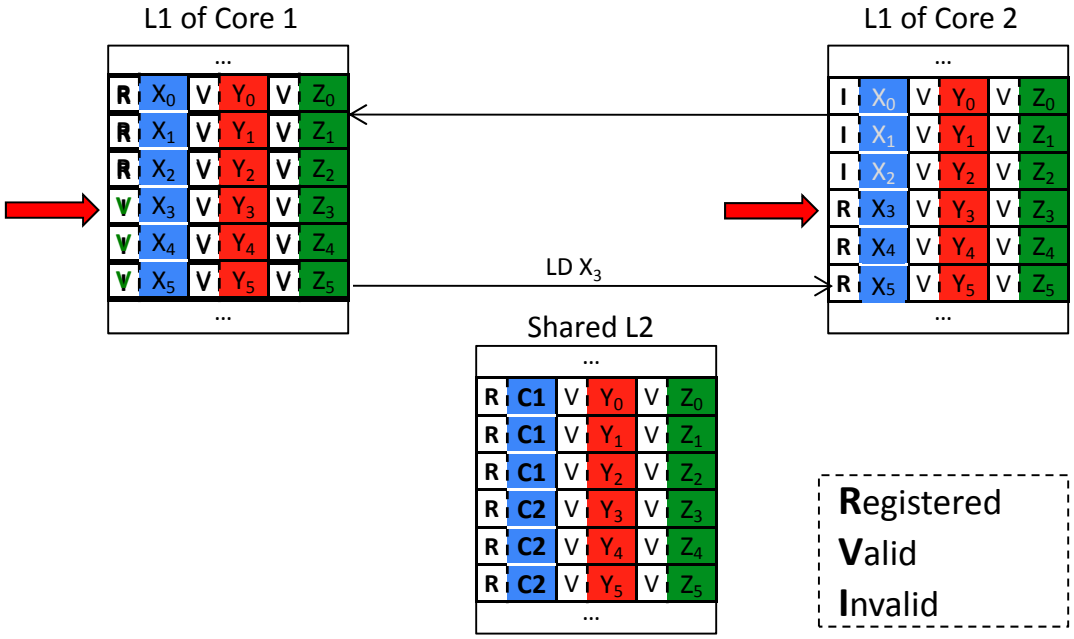
   DeNovo uses regions to transfer only relevant data

   Effect of AoS-to-SoA transformation w/o programmer/compiler

Flexible, Direct Communication



Flexible, Direct Communication



Current Hardware Limitations

- Complexity
  - ✔ – Subtle races and numerous transient sates in the protocol
  - ✔ – Hard to extend for optimizations

Base DeNovo
20X faster to verify vs. MESI

- Storage overhead
  - ✔ – Directory overhead for sharer lists

- Performance and power inefficie
  - ✔ – Invalidation, ack messages
  - ✔ – Indirection through directory
  - ✔ – False sharing (cache-line based coherence)
  - ✔ – Traffic (cache-line based communication)
  - – Cache pollution (cache-line based allocation) *ongoing*

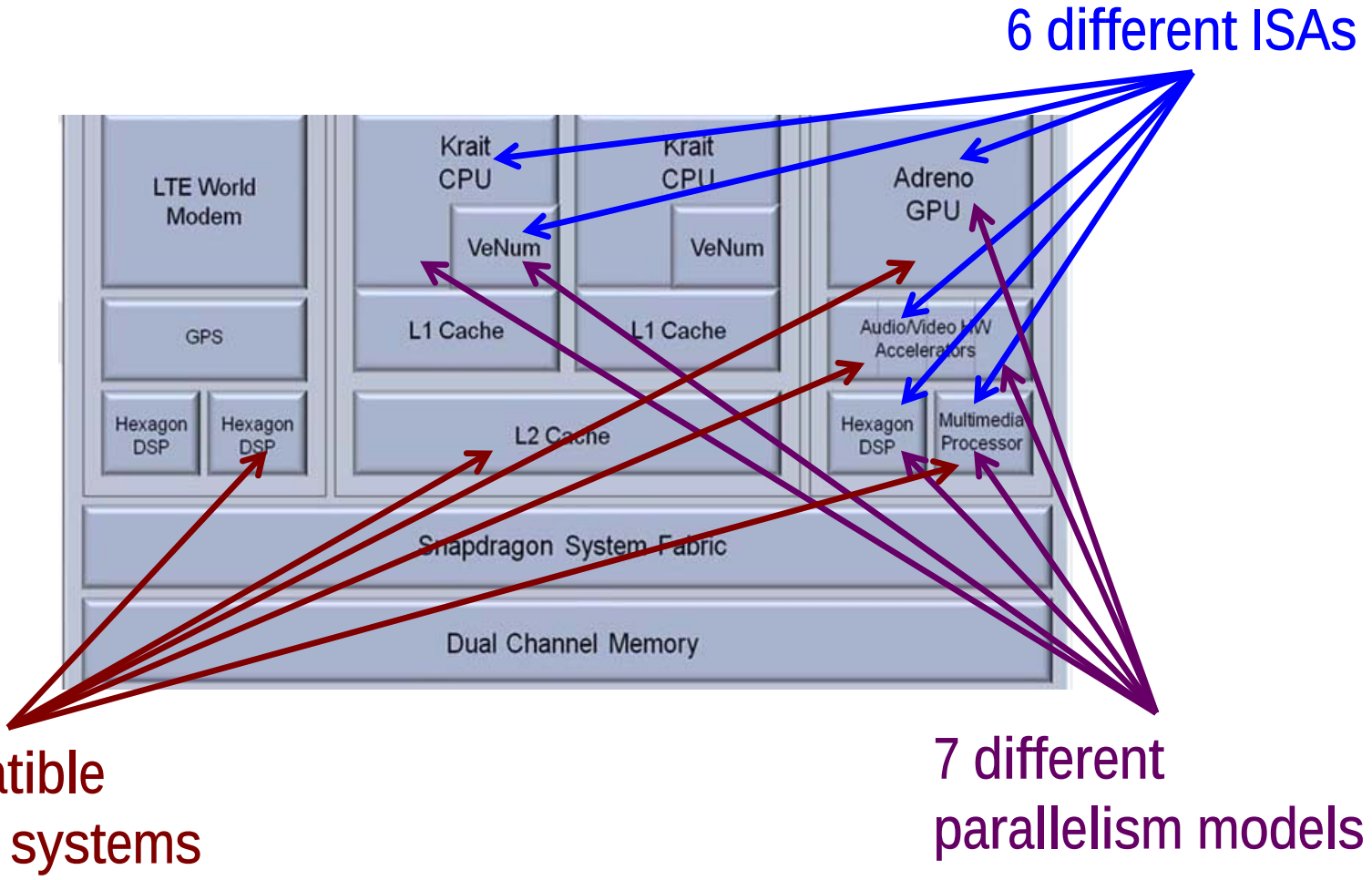
Up to 79% lower memory stall time
Up to 72% lower traffic

Extended to non-deterministic codes; pipelined/unstructured codes ongoing

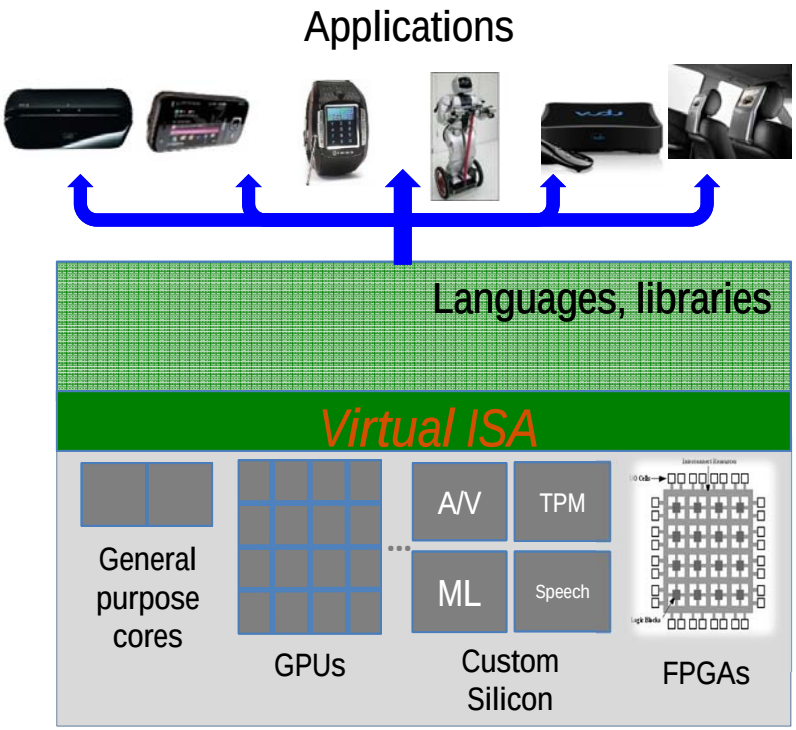
Specialization/Heterogeneity

A modern smartphone

CPU, GPU, DSP, Vector Units, Multimedia, Audio-Video accelerators



Our Approach [HotPar'12]



Virtual Instruction Set Computing (VISC)

- Low-level, language neutral virtual ISA (LLVM++)
- Abstract away hardware differences
  - Few parallelism models
  - Uniform memory abstractions

Region-driven heterogeneous memory

- (Reconfigurable) scratchpads, caches, FIFOs, …
- Unified address space
- Coherent a la DeNovo
- Flexible data transfer a la DeNovo

Goal: Higher energy efficiency + Simpler programming model

Summary

- Parallelism and specialization may solve energy crisis, but
  - Require rethinking software, hardware, interface
- Disciplined parallel programming allows
  - Safe programming
  - Complexity-, performance-, power-efficient hardware
  - DPJ, DeNovo
- Virtual instruction set computing allows
  - Easier programmability, portability
  - Effiicient memory system
  - LLVM++, DeNovo++