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

Incompatible memory systems
Different hardware ISAs

Increasing diversity in & across SoCs & supercomputers, data centers, ...

Different parallelism models

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Level</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>App. productivity</td>
<td>Domain-specific prog. language</td>
<td>TensorFlow, MXNet, Halide, ...</td>
</tr>
<tr>
<td>App. performance</td>
<td>General-purpose prog. language</td>
<td>CUDA, OpenCL, OpenAcc, OpenMP, Python, Julia</td>
</tr>
<tr>
<td>Language innovation</td>
<td>Language-level Compiler IR</td>
<td>Delite DSL IR, DLVM, TVM, ...</td>
</tr>
<tr>
<td>Compiler investment</td>
<td>Language-neutral Compiler IR</td>
<td>Delite IR, HPVM, OSCAR, Polly</td>
</tr>
<tr>
<td>Object-code portability</td>
<td>Virtual ISA</td>
<td>SPIR, HPVM</td>
</tr>
<tr>
<td>Hardware innovation</td>
<td>&quot;Hardware&quot; ISA</td>
<td>IBM AS/400, Transmeta, PTX, HSAIL, Codesigned Virtual Machines</td>
</tr>
</tbody>
</table>

Source: Vikram Adve, HPVM project, [https://publish.illinois.edu/hpvm-project/](https://publish.illinois.edu/hpvm-project/)
Which Interface Levels Can Be Uniform?

- **Domain-specific prog. language**
- **General-purpose prog. language**
- **Language-level Compiler IR**
- **Language-neutral Compiler IR**
- **Virtual ISA**
- **"Hardware" ISA**

**CPUs + Vector SIMD Units**

**GPU**

**Domain-specific Accelerators**

**DSP**

**FPGA**

Too diverse to define a uniform interface

Much more uniform

Also too diverse ...

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

Hierarchical Dataflow Graph with side effects

Vector

\[ V_A = \text{load} \langle L4 \times \text{float} \rangle^* A \]
\[ V_B = \text{load} \langle L4 \times \text{float} \rangle^* B \]
\[ \ldots \]
\[ V_C = \text{fmul} \langle L4 \times \text{float} \rangle V_A, V_B \]

or
Dataflow Graph with side effects

Hierarchical

Vector

- Task, data, vector parallelism
- Streams, pipelines
- Shared memory
- High-level optimizations
- FPGAs (more custom hw?)

$V_A = \text{load} \langle L4 x \text{float} \rangle^* A$
$V_B = \text{load} \langle L4 x \text{float} \rangle^* B$
...

N different parallelism models \rightarrow single unified model
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

<table>
<thead>
<tr>
<th>Granularity</th>
<th>MESI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line</td>
<td>Line</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stale data invalidation</th>
<th>MESI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Writer-invalidate</td>
<td>Writer-invalidate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Write propagation</th>
<th>MESI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ownership</td>
<td>Ownership</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Protocol properties</th>
<th>MESI</th>
<th>GPU coherence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Granularity</strong></td>
<td>Line</td>
<td>Reads: line writes: word</td>
</tr>
<tr>
<td><strong>Stale data invalidation</strong></td>
<td>Writer-invalidate</td>
<td>Self-invalidate</td>
</tr>
<tr>
<td><strong>Write propagation</strong></td>
<td>Ownership</td>
<td>Write-through</td>
</tr>
</tbody>
</table>

### 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

**Good for:**

- CPU
- GPU
DeNovo is good fit for CPU and GPU

<table>
<thead>
<tr>
<th>Protocol properties</th>
<th>MESI</th>
<th>GPU coherence</th>
<th>DeNovo</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Granularity</strong></td>
<td>Line</td>
<td>Reads: line writes: word</td>
<td>Reads: flexible Writes: word</td>
</tr>
<tr>
<td><strong>Stale data invalidation</strong></td>
<td>Writer-invalidate</td>
<td>Self-invalidate</td>
<td>Self-invalidate</td>
</tr>
<tr>
<td><strong>Write propagation</strong></td>
<td>Ownership</td>
<td>Write-through</td>
<td>Ownership</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Application</th>
<th>Access Pattern</th>
<th>Important Dimension</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pull-based PageRank</td>
<td>Read neighbor vertices, Update local vertex</td>
<td><strong>Flat LLC avoids indirection for read misses</strong></td>
<td>Spandex LLC ⇒ 37% better exec. time 9% better NW traffic</td>
</tr>
<tr>
<td>Push-based Betweenness Centrality</td>
<td>Read local vertex, Update (RMW) neighbor vertices</td>
<td><strong>Ownership-based write propagation</strong> exploits locality in updates</td>
<td>DeNovo at GPU ⇒ 18% better exec. time 61% better NW traffic</td>
</tr>
</tbody>
</table>
Looking Forward…

Software Innovations

- Synchronization locality
- Data locality, visibility
- Coarse-grain operations
- Producer/consumer relationships

Hardware Innovations

- hLRC adaptive laziness
- Coherent scratchpads
- Spandex dynamic caches
- HBM caches
- Hardware queues
- NVRAM

HPVM + DRF Consistency + ???
Approximation

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

Integrate approximation (quality) into the interface
Summary

• Interfaces

• Data

• Approximation