EPOCHS: EFFICIENT PROGRAMMABILITY OF COGNITIVE HETEROGENEOUS SYSTEMS
SARITA ADVE

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
SADVE@ILLINOIS.EDU
EPOCHS OVERVIEW

Quick design & implementation of domain-specific SoC + compiler + scheduler for cognitive connected vehicles

Distribution Statement A. Approved for Public Release
Key to heterogeneous system design: Interfaces for hardware & software
- Specializable
- Programmable
- Efficient

This talk: Two interfaces
- **Spandex**: Accelerator communication interface
- **Heterogeneous Parallel Virtual Machine (HPVM)**: Hardware-software interface

*Enable specialization with programmability and efficiency*
HETEROGENEOUS SYSTEMS COMMUNICATION

Traditional heterogeneity:
- CPU
- Accelerator
- CPU Mem Space
- Accelerator Mem Space

- Wasteful data movement
- No fine-grain synchronization (atomics)
- No irregular access patterns

Coherent shared memory:
- CPU
- Accelerator
- Shared Mem Space
- Coherent data

- Implicit data reuse
- Fine-grain synchronization (atomics)
- Irregular access patterns

Current solutions: complex and inflexible
HETEROGENEOUS DEVICES HAVE DIVERSE MEMORY DEMANDS

- Spatial locality
- Temporal locality
- Fine-grain Synch
- Latency Sensitivity
- Throughput Sensitivity
- Sensitivity
- Throughput Synch
HETEROGENEOUS DEVICES HAVE DIVERSE MEMORY DEMANDS

Typical CPU workloads: fine-grain synchronization, latency sensitive
HETEROGENEOUS DEVICES HAVE DIVERSE MEMORY DEMANDS

Typical GPU workloads: spatial locality, throughput sensitive
### Key Properties of Coherence Protocols

<table>
<thead>
<tr>
<th>Properties</th>
<th>Granularity</th>
<th>Invalidation</th>
<th>Updates</th>
</tr>
</thead>
</table>

Distribution Statement A. Approved for Public Release
## KEY PROPERTIES OF COHERENCE PROTOCOLS

<table>
<thead>
<tr>
<th>Properties</th>
<th>CPU MESI</th>
<th>GPU GPU coh. (for CPU or GPU)</th>
<th>DeNovo</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Invalidation</strong></td>
<td>Writer-invalidate</td>
<td>Self-invalidate</td>
<td>Self-invalidate</td>
</tr>
<tr>
<td><strong>Updates</strong></td>
<td>Ownership</td>
<td>Write-through</td>
<td>Ownership</td>
</tr>
</tbody>
</table>

How to integrate different accelerators with different protocols?
CURRENT SOLUTIONS: INFLEXIBLE, INEFFICIENT
CURRENT SOLUTIONS: INFLEXIBLE, INEFFICIENT

- CPU
  - MESI L1

- Accel 1
  - GPU coh. L1

- Accel 2
  - DeNovo L1

- GPU
  - GPU coh. L1

MESI Last Level Cache (LLC)
CURRENT SOLUTIONS: INFLEXIBLE, INEFFICIENT

CPU

MESI L1

DeNovo L1

GPU coh. L1

Accel 1

MESI Last Level Cache (LLC)

Accel 2

GPU
CURRENT SOLUTIONS: INFLEXIBLE, INEFFICIENT

Cpu
MESI L1

Accel 1
MESI L1

Accel 2
MESI L1

GPU
GPU coh. L1

MESI Last Level Cache (LLC)
CURRENT SOLUTIONS: INFLEXIBLE, INEFFICIENT
SPANDEX: FLEXIBLE HETEROGENEOUS COHERENCE INTERFACE [ISCA’18]

- Adapts to exploit individual device’s workload attributes
- Better performance, lower complexity
⇒ Fits like a glove for heterogeneous systems!

Supported by ADA JUMP and C-FAR STARNET centers
SPANDEX OVERVIEW

Key Components

- Flexible device request interface
- DeNovo-based LLC
- External request interface

Device may need a translation unit (TU)
**SPANDEX OVERVIEW**

**Key Components**

- Flexible device request interface
- DeNovo-based LLC
- External request interface
- Device may need a translation unit (TU)
### SPANDEX DEVICE REQUEST INTERFACE

<table>
<thead>
<tr>
<th>Action</th>
<th>Request</th>
<th>Indicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>ReqV</td>
<td>Self-invalidation</td>
</tr>
<tr>
<td></td>
<td>ReqS</td>
<td>Writer-invalidation</td>
</tr>
<tr>
<td>Write</td>
<td>ReqWT</td>
<td>Write-through</td>
</tr>
<tr>
<td></td>
<td>ReqO</td>
<td>Ownership only</td>
</tr>
<tr>
<td>Read+Write</td>
<td>ReqWT+data</td>
<td>Atomic write-through</td>
</tr>
<tr>
<td></td>
<td>ReqO+data</td>
<td>Ownership + Data</td>
</tr>
<tr>
<td>Writeback</td>
<td>ReqWB</td>
<td>Owned data eviction</td>
</tr>
</tbody>
</table>

- Requests also specify granularity and (optionally) a bitmask
**DYNAMIC & NEW COHERENCE SPECIALIZATIONS**

- Spandex flexibility & simplicity enables dynamic & new coherence specializations

- Directed by compiler or runtime
- Applied to neural networks: enables new programming patterns

*Dynamic Spandex request selection*
*Producer-consumer forwarding*
*Extended granularity flexibility*
**DATA PARALLEL VS. PIPELINED NEURAL NETWORKS**

### Fully connected DNN

**Data Parallel**
- Each core has independent image
- Each core processes all network layers
- No communication/synch overhead
- But need weights of all layers for reuse

**Pipelined**
- Each core processes one layer
- Communication/synch overhead
- But need weights of only one layer for reuse

### Recurrent neural nw (RNN)

**Pipelined (no simple data parallel)**
- No simple communication-free data parallel version since computation depends on previous input

**Spandex flexibility enables fine-grained communication and efficient pipelines**
COHERENCE SPECIALIZATION FOR PIPELINED NEURAL NETWORKS

• Dynamically select coherence strategy for different data
  – Use ownership for weight data (reuse), writethrough for input feature data
COHERENCE SPECIALIZATION FOR PIPELINED NEURAL NETWORKS

• Dynamically select coherence strategy for different data
  – Use ownership for weight data (reuse), writethrough for input feature data

• Optimization1: Producer $\rightarrow$ Consumer forward via last level cache (LLC)
  – LLC forwards feature output to next consumer
COHERENCE SPECIALIZATION FOR PIPELINED NEURAL NETWORKS

- Dynamically select coherence strategy for different data
  - Use ownership for weight data (reuse), writethrough for input feature data

- Optimization1: Producer → Consumer forward via last level cache (LLC)
  - LLC forwards feature output to next consumer

- Optimization2: Direct Producer → Consumer forward w/ owner prediction
  - Producer directly forwards feature data to consumer cache, with no LLC lookup
EVALUATION METHODOLOGY

• Baseline system: CPU + Multiple GPU compute cores on a network on chip
• Neural network (NN) computation on different compute cores of GPUs
• NNs based on standard networks or information from literature
• Cycle accurate architectural simulation: GEMS+GPGPUllSim+Garnett+Spandex
RESULTS FOR NEURAL NETWORKS

Large reduction in execution time and/or network traffic
Next steps for Spandex: Apply to EPOCHS application and integrate with compiler
Key to heterogeneous system design: Interfaces for hardware & software
  • Specializable
  • Programmable
  • Efficient

This talk: Two interfaces
  ✓ Spandex: Accelerator communication interface
  ➡️ Heterogeneous Parallel Virtual Machine (HPVM): Hardware-software interface

Enable specialization with programmability and efficiency
### CURRENT INTERFACE LEVELS

<table>
<thead>
<tr>
<th>Category</th>
<th>Interface Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>App. productivity</td>
<td>Domain-specific prog. language</td>
</tr>
<tr>
<td>App. performance</td>
<td>General-purpose prog. 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>
<tr>
<td>Hardware innovation</td>
<td>&quot;Hardware&quot; ISA</td>
</tr>
</tbody>
</table>

- CPUs + Vector SIMD Units
- GPU
- DSP
- Custom accelerators
- FPGA
WHERE SHOULD WE ABSTRACT HETEROGENEITY?

- **Domain-specific prog. language**
  - Too diverse to define a uniform interface

- **General-purpose prog. language**
  - Much more uniform

- **Language-level Compiler IR**
  - Also too diverse...

- **Language-neutral Compiler IR**
  - HPVM

- **Virtual ISA**
  - "Hardware" ISA

- **"Hardware" ISA**

**CPUs + Vector SIMD Units**

**GPU**

**DSP**

**Custom accelerators**

**FPGA**

Distribution Statement A. Approved for Public Release
**HETEROGENEOUS PARALLEL VIRTUAL MACHINE**

**Key to Programmability:**
Common abstractions for heterogeneous parallel hardware

Use HPVM for:
1. Portable *object* code
2. Retargetable parallel compiler IR and system
3. Run-time scheduling

Kotsifakou et al., PPOPP 2018
ABSTRACTION OF PARALLEL COMPUTATION

Hierarchical Dataflow Graph with side effects

N different parallelism models

Single unified parallelism model

Captures
- coarse grain task parallelism
- streams, pipelined parallelism
- nested parallelism
- SPMD-style data parallelism
- fine grain vector parallelism

Supports high-level optimizations as graph transformations
Two key aspects

1. Any node ↔ Any device
2. Reuse vendor back ends for high performance code-gen
APPROX-HPVM: ACCURACY AWARE OPTIMIZATION

Relaxing accuracy can enable higher efficiency (2X to 50X in our work)

Can we make these techniques easier to use?

Goal 1: Applications should only specify high-level accuracy goals
  • Maximum acceptable loss in quality, e.g., inference error, PSNR
  • End-to-end metrics, not per function or pipeline stage or …

Goal 2: (Often) Want object-code portability
  • Approximation choices are highly system-dependent
  • Can make orders-of-magnitude difference in performance, energy

**ApproxHPVM: Accuracy-aware representation and optimizations**
Implemented for Keras: Neural networks in TensorFlow

Sharif et al., OOPSLA 2019, accepted with revisions
DNN SPEEDUP AND ENERGY SAVINGS

Target system: NVIDIA Tegra TX2 + PROMISE ML accelerator
TX2: FP32 or FP16, PROMISE: 7 voltage levels in SRAM bitlines

1-2% loss of inference accuracy gives
2X-9X speedup, 2X-11X energy savings in most networks
Key to heterogeneous system design: Interfaces for hardware & software
  • Specializable
  • Programmable
  • Efficient

This talk: Two interfaces
  • Spandex: Accelerator communication interface
  • Heterogeneous Parallel Virtual Machine (HPVM): Hardware-software interface

Enable specialization with programmability and efficiency