Stash: Have Your Scratchpad and Cache it Too

Matthew D. Sinclair
with:
Rakesh Komuravelli, Johnathan Alsop,
Muhammad Huzaifa, Maria Kotsifakou,
Prakalp Srivastava, Sarita V. Adve, and Vikram S. Adve

University of Illinois @ Urbana-Champaign
hetero@cs.illinois.edu
SoCs Need an Efficient Memory Hierarchy

- Energy-efficient memory hierarchy is essential
  - Heterogeneous SoCs use **specialized memories**
  - E.g., scratchpads, FIFOs, stream buffers, ...

<table>
<thead>
<tr>
<th></th>
<th>Scratchpad</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directly addressed: no tags/TLB/conflicts</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Compact storage: no holes in cache lines</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>
SoCs Need an Efficient Memory Hierarchy

- Energy-efficient memory hierarchy is essential
  - Heterogeneous SoCs use specialized memories
  - E.g., scratchpads, FIFOs, stream buffers, …

<table>
<thead>
<tr>
<th></th>
<th>Scratchpad</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directly addressed: no tags/TLB/conflicts</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Compact storage: no holes in cache lines</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Global address space: implicit data movement</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Coherent: reuse, lazy writebacks</td>
<td>X</td>
<td>✓</td>
</tr>
</tbody>
</table>

Can specialized memories be globally addressable, coherent?
Can we have our scratchpad and cache it too?
Can We Have Our Scratchpad and Cache it Too?

- Make specialized memories globally addressable, coherent
  - Efficient address mapping
  - Efficient coherence protocol

Focus: CPU-GPU systems with scratchpads and caches
  - Up to 31% less execution time, 51% less energy
Outline

- Motivation
- **Background: Scratchpads & Caches**
- Stash Overview
- Implementation
- Results
- Conclusion
Global Addressability

• Scratchpads
  – Part of private address space: not globally addressable
    ⇒ Explicit movement, pollution, poor conditional accesses support

• Cache
  + Globally addressable: part of global address space
    ⇒ Implicit copies, no pollution, support for conditional accesses
Coherence: Globally Visible Data

- **Scratchpads**
  - Part of private address space: not globally visible
    ➞ Eager writebacks and invalidations on synchronization

- **Cache**
  + Globally visible: data kept coherent
    ➞ Lazy writebacks as space is needed, reuse data across synch
### Stash – A Scratchpad, Cache Hybrid

<table>
<thead>
<tr>
<th></th>
<th>Scratchpad Cache</th>
<th>Stash</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Directly addressed:</strong> no tags/TLB/conflicts</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Compact storage:</strong> no holes in cache lines</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Global address space:</strong> implicit data move.</td>
<td>✗</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>Coherent:</strong> reuse, lazy writebacks</td>
<td>✗</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Related Work

• Caches:
  – Changing Data Layout [HPCA ’99, SC ’11]
  – Elide Tag Accesses [MICRO ’13, ISPLED ’14]

• Scratchpads:
  – Bypassing L1 cache [Southern Island ‘09]
  – Virtualizing Private Memories
    • [ISPLED ‘11, ISPLED ‘12, UC-B MS ’09, TACO ‘12]
  – Scratchpads with DMA support [SC ‘11, PACT ‘14]

• Compare stash to scratchpads with DMA support
• Motivation
• Background: Scratchpads & Caches
  • Stash Overview
• Implementation
• Results
• Conclusion
Stash: Directly & Globally Addressable

- Like scratchpad: directly addressable (for hits)
- Like cache: globally addressable (for misses)
  - Implicit loads, no cache pollution
Stash: Globally Visible

- Stash data can be accessed by other units
- Needs coherence support

- Like cache
  - Keep data around – lazy writebacks
  - Intra- or inter-kernel data reuse on the same core
Stash: Compact Storage

- **Caches**: cache line granularity storage ("holes" ⇒ waste)
  - Do not compact data

- **Like scratchpad, stash compacts data**
Outline

• Motivation
• Background: Scratchpads & Caches
• Stash Overview
  • Implementation
• Results
• Conclusion
Stash Software Interface

- Software gives a mapping for each stash allocation
  - One map entry (instruction) per stash array per thread block
  - Map 2D non-contiguous global regions to stash
stash_load[505, R_k];

Stash Hardware
Coherence Support for Stash

• Stash data needs to be kept coherent

• Extend a coherence protocol for three features
  – Track stash data at word granularity
  – Capability to merge partial lines when stash sends data
  – Modify directory to record the modifier and stash-map ID

• We choose to extend the DeNovo protocol
  – Simple, low overhead, hybrid of CPU and GPU protocols
• Read hit – don’t return stale data
  – Before next parallel phase, selectively self-invalidate
    • Needn’t invalidate data it accessed in previous phase

• Read miss – Find one up-to-date copy
  – Before end of phase, write miss registers at “directory” registry
  – Shared LLC data arrays double as directory
    • Keep valid data or registered core ID
    • Stash extension: store map ID at registry
• Assume (for now): Private L1, shared L2; single word line
  – Data-race freedom at word granularity

  ![Diagram showing coherence states: Invalid → Read → Valid → Write → Registered → Read, Write]

  No transient states
  No invalidation traffic
  No directory storage overhead
  No false sharing (word coherence)

• Line-based DeNovo: word coherence, line address/transfer
• Motivation
• Background: Scratchpads & Caches
• Stash Overview
• Implementation
  • Results
• Conclusion
Evaluation

• Simulation Environment
  – GEMS + Simics + Princeton Garnet N/W + GPGPU-Sim
  – Extend McPAT and GPUWattch for energy evaluations

• Workloads:
  – 4 microbenchmarks: implicit, reuse, pollution, on-demand
  – Heterogeneous workloads: Rodinia, Parboil, SURF

• 1 CPU Core (15 for microbenchmarks)
• 15 GPU Compute Units (1 for microbenchmarks)
• 32 KB L1 Caches, 16 KB Stash/Scratchpad
Evaluation (Microbenchmarks) – Execution Time

- **Scr** = Baseline configuration
- **C** = All requests use cache
- **Scr+D** = All requests use scratchpad w/ DMA
- **St** = Converts scratchpad requests to stash
Evaluation (Microbenchmarks) – Execution Time

No explicit loads/stores
Evaluation (Microbenchmarks) – Execution Time

No cache pollution
Evaluation (Microbenchmarks) – Execution Time

Only bring needed data
Evaluation (Microbenchmarks) – Execution Time

Data compaction, reuse
Evaluation (Microbenchmarks) – Execution Time

- Avg: 27% vs. Scratch, 13% vs. Cache, 14% vs. DMA
Evaluation (Microbenchmarks) – Energy

- Avg: 53% vs. Scratch, 36% vs. Cache, 32% vs. DMA
Evaluation (Apps) – Execution Time

Scr = Reqs use type specified by original app
C  = All reqs use cache
St = Converts scratchpad reqs to stash
Evaluation (Apps) – Execution Time

- Avg: 10% vs. Scratch, 12% vs. Cache (max: 22%, 31%)
  - Source: implicit data movement
- Comparable to Scratchpad+DMA
Evaluation (Apps) – Energy

- Avg: 16% vs. Scratch, 32% vs. Cache (max: 30%, 51%)
Conclusion

• Make specialized memories globally addressable, coherent
  – Efficient address mapping (only for misses)
  – Efficient software-driven hardware coherence protocol

• Stash = scratchpad + cache
  – Like scratchpads: Directly addressable and compact storage
  – Like caches: Globally addressable and globally visible

• Reduced execution time and energy

• Future Work:
  – More accelerators & specialized memories; consistency models