Virtualizing Local Stores

Henry Cook, Krste Asanovic, Dave Patterson

June 2, 2009
Par Lab Research Overview

Easy to write correct programs that run efficiently on manycore

Applications

Productivity Layer

Efficiency Layer

Diagnosing Power/Performance

Personal Health | Image Retrieval | Hearing, Music | Speech | Parallel Browser

Design Patterns/Motifs

Composition & Coordination Language (C&CL)

C&CL Compiler/Interpreter

Parallel Libraries | Parallel Frameworks

Efficiency Languages

Sketching

Autotuners

Legacy Code | Schedulers | Communication & Synch. Primitives

Efficiency Language Compilers

Legacy OS | OS Libraries & Services | Hypervisor

Multicore/GPGPU | ParLab Manycore/RAMP

Static Verification

Type Systems

Directed Testing

Dynamic Checking

Debugging with Replay

Diagnosing Power/Performance
Easy to write correct programs that run efficiently on manycore
Memory Hierarchy: Two Views

- A transparent system designed to leverage data locality automatically, allowing the programmer to be blissfully ignorant.

- A blackbox of interacting mechanisms that must be inevitably be overcome by the programmer to get peak performance.
Hardware-managed caches

- Surprising (to the programmer) behavior
  - Causes are hidden, but perf. effects apparent
- Reactivity leads to prefetching: a gamble
- Less programmer effort upfront
- Handles certain access patterns well
Software-managed Local Stores

- SW has explicit (re)placement control
  - Proactive movement, direct-mapped
- High BW transfers, overlapped with comp.
- Increased programmer effort
- Some memory usages are not amenable
Physically Hybrid Systems

- HW & SW management both have viable use cases
  - Why not provide hardware for both?

- In a general-purpose system, this is actually a very poor solution...
Worst of both worlds?

- Divides up available capacity resources
- Fixes partitioning at design time
- Can’t update all legacy code
- Runtime composability issues
- Adds to user context even if unused
Best of both worlds?

- Virtual Local Stores
  - Repurpose cache resources to provide “local store”-like characteristics to the programmer on demand
Some Notable VLS Features

- Low overhead dynamic repartitioning
- Minimal hardware changes
- Integrates with mainstream environments
  - multiprogrammed, virtual memory, multicore, legacy code
Outline

- Intro
- Programmer’s View
- VLS Mechanisms
- Case Study
- Future Work
The Programmer’s View

- Region of virtual address space has special “local store”-like characteristics
  - Direct mapped, explicitly managed, etc.

- If you copy data to that region, it will behave like it was in a real, physical LS

- If you do not use it, no impact on normal performance of HW-managed cache
Simple example

\[ DMA\text{Gather}(V_{Src}, V_{Dest}, \text{Stride}) \]

\[ VLS\text{Allocate}() \]

\[ \text{Do\ Computation}(V_{Dest}) \]

Virtual Memory
Physical Memory
On-chip Local Memory
How VLS Maps into Memory Space
How VLS Maps into Memory Space
Memory Access Pipeline Integration
Memory Access Pipeline Integration
Partitioning via Replacement Policy

- Don’t ever let HW-managed data evict SW-managed data when VLS is enabled
- More complicated replacement policy not on critical path
- Way-based partitioning
  - Other possibilities
- Dynamic reallocation
  - Low overhead
  - Lazy eviction
- Transparent to OS
OS implications of VLS

- If OS swaps an application using VLS, must save and restore control registers, but not data
  - Data is not part of the user process state thanks to backing physical memory
  - Data in non-resident VLS may be evicted (restored on demand)

- If a thread is migrated to a new core, hardware coherence can migrate data in VLS on demand

- Compatible with Tessellation view of scheduling
Application Use Cases

- Traditional use cases for LS
  - Streaming
  - Locking
- VLS can do both, and supports per-routine solutions (phases and library calls)
- Encourages incremental adoption of SW memory management
  - Only deploy it where deemed beneficial
Case Study: Speech Inference

- Simulated using Simics/GEMS, with VLS support added
Future Work

- Evaluation on more applications
- Investigate energy tradeoffs
- Using a shared LS in outer level cache
- VLS-to-VLS transfer via DMA transfers
  - Horizontal and vertical transfers
- Integration with Productivity Layer
The End

☐ Eager for collaboration
  ☐ Please send questions or requests for paper
  ☐ hcook@eecs.berkeley.edu

☐ Questions?
<table>
<thead>
<tr>
<th>Real LS</th>
<th>VLS</th>
<th>HW cache (w/ phys. tags)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No tag checks</td>
<td>One way check</td>
<td>N-way tag check</td>
</tr>
<tr>
<td>No TLB lookup</td>
<td>No TLB lookup</td>
<td>TLB lookup</td>
</tr>
<tr>
<td>Flush on context switch</td>
<td>No flush on context switch</td>
<td>No flush on context switch</td>
</tr>
<tr>
<td>No coherence in hardware</td>
<td>Coherence on migration</td>
<td>Coherence on “shared” data</td>
</tr>
</tbody>
</table>
Context switch overhead

- How much effort is saved by not having to save and restore the LS on every switch?

- Still need to calibrate energy cost relative to single tag check for VLS, memory accesses, etc.
Example of VLS Usage (1)
Example of VLS Usage (2)
Example of VLS Usage (3)
Example of VLS Usage (4)

STORE #99, 103;

Physical Memory

<table>
<thead>
<tr>
<th>100</th>
<th>101</th>
<th>102</th>
<th>103</th>
<th>104</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>218</td>
<td>0</td>
<td>11</td>
<td>219</td>
<td>22</td>
</tr>
<tr>
<td>21A</td>
<td>22</td>
<td>33</td>
<td>22B</td>
<td>44</td>
</tr>
<tr>
<td>22C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

On-chip Data Buffer

<table>
<thead>
<tr>
<th>Tag Array</th>
<th>Data Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>X 101</td>
<td>11</td>
</tr>
<tr>
<td>X 102</td>
<td>22</td>
</tr>
<tr>
<td>X 103</td>
<td>99</td>
</tr>
<tr>
<td>X 104</td>
<td>44</td>
</tr>
</tbody>
</table>

enabled? true
pbase 101
Example of VLS Usage (5)

VLS_UNMAP();
LOAD 218, R1;

Physical Memory

On-chip Data Buffer

<table>
<thead>
<tr>
<th>Tag Array</th>
<th>Data Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>X 101</td>
<td>11</td>
</tr>
<tr>
<td>X 102</td>
<td>22</td>
</tr>
<tr>
<td>X 104</td>
<td>0</td>
</tr>
<tr>
<td>218</td>
<td>218</td>
</tr>
<tr>
<td>219</td>
<td>22</td>
</tr>
<tr>
<td>21A</td>
<td>33</td>
</tr>
<tr>
<td>22B</td>
<td>44</td>
</tr>
</tbody>
</table>

(enabled?) false

pbase