Virtualizing Local Stores

Enabling Software-Managed Memory Hierarchies in Mainstream Computing

Henry Cook

PROBLEM STATEMENT

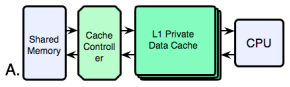
Software-managed local stores are more efficient than hardware-managed caches for some apps, yet their use has been confined to embedded systems. Local stores are problematic in general-purpose systems because they add to process state on context switches, and because they require fast data memory close to the processor that might be better spent as cache. We propose the use of virtualized local stores to provide the benefits of a software-managed memory hierarchy in a general-purpose system. A VLS is mapped into the virtual address space of a process to allow software management, but is kept in a partition of the hardware-managed cache when active.

RELATED WORK

Smart Memories, TRIPS, ALP all provide heavyweight reconfigurability. Various embedded processors have used way-based partitioning and locking, but generally are uni-purpose/uni-process.

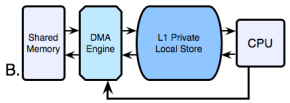
POSSIBLE MEMORY CONFIGURATIONS

HW-MANAGED CACHES



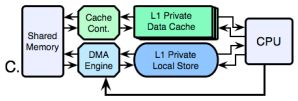
Hide locality from software by automatically moving data. Require prefetch engines for performance. Inflexible mappings and replacement policies. Productive.

SW-MANAGED LOCAL STORES



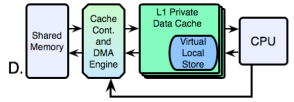
Expose locality to software, requiring explicit DMA control. Allow macroscopic prefetching and other optimizations. Better flexibility and simple high-bandwidth transfers. Efficient.

STATIC ALLOCATION OF BOTH



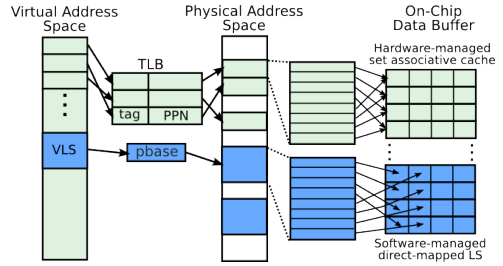
Provides both A and B within a single system. Presence of some cache improves programmability. Benchmarks cannot use local store until they have been rewritten.

VIRTUAL LOCAL STORES



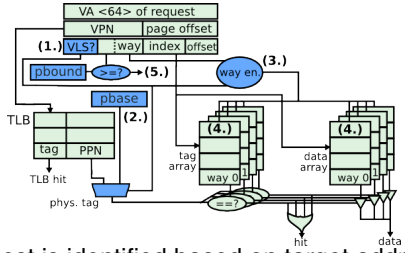
Provides benefits of C, but is lightweight and dynamically reconfigurable. Programmers have power to choose between A and B on a per routine basis. Fits OS/multiprocessor goal.

ADDRESS SPACE MAPPINGS

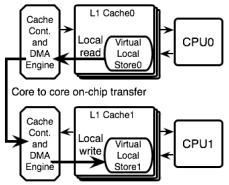


Mapping of virtual local stores from the VA space to physical pages, and how data in those pages is indexed in the on-chip memory hierarchy.

VLS MECHANISMS



1. Request is identified based on target address as being a VLS access
2. `pbase` holds physical base address, used instead of TLB lookup
3. Multiple way lookups avoided on a VLS access
4. Replacement policy respects partitioning
5. `pbound` holds number of pages given to VLS
6. Special network allows direct VLS-to-VLS data streaming via DMA engines



FUTURE WORK

- More multithreaded application use cases
- Use of VLS in multiple levels of hierarchy
- VLS-to-VLS communication
- Energy efficiency

METHODOLOGY

We used a combination of Virtutech Simics and Wisconsin GEMS to evaluate the detrimental effects of a fixed allocation between hardware and software-managed local memories on various computational kernels. General-purpose multiprocessor workloads will consist of all of these kernels and more.

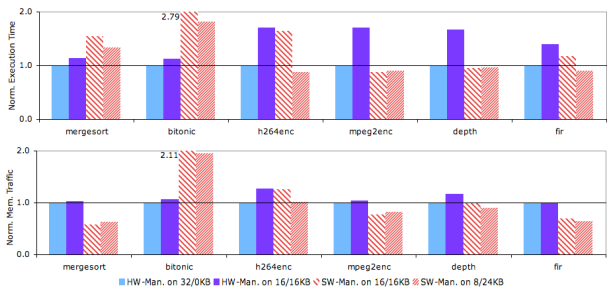
BENCHMARKS

Hand-tuned versions of several kernels designed to run on pure cache or LS machines.
Parallelized with pthreads.
Run to completion.

MACHINE

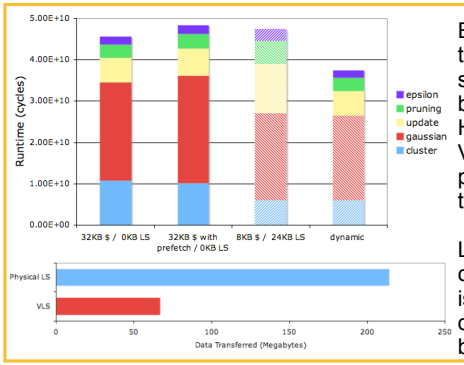
16 cores, 800MHz
16 KB 2-way L1 I-Cache
32 KB 4-way L1 D-Cache
with VLS up to 3-way
512 KB 16-way unified L2
1 DMA engine per core

MICROBENCHMARK RESULTS



Some kernels perform better with local stores, some do not. Static allocations that limit the size of each partition are detrimental to both.

SPEECH APPLICATION RESULTS



Each phase of the algorithm sees different benefits under HW or SW-man. VLS allows us to pick between them per-phase.

Load caused by context switches is reduced: VLS data is backed by phys. mem.