Replicate and Migrate Objects in the Runtime –not Cache Lines or Pages in Hardware

Manolis Katevenis

FORTH-ICS and Univ. of Crete, Greece

BMW – October 2010
Acknowledgements

• Alex Ramirez
• Dimitris Nikolopoulos
• Dionisios Pnevmatikatos
• Georgi Gaydadjiev
• Panagiota Fatourou
• Polyvios Pratikakis

• Vassilis Papaefstathiou
• Stamatis Kavadias
• Spyros Lyberis

This version of the slides includes post-talk comments, based on discussions at the end of the talk – see (new) slide 19 – as well as an improved slide 22
Outline

• Opportunity: Reduce Network Traffic (⇒ Energy too) by:
  – Transferring data if and when needed by the application
  – Transferring data in units of “objects” – not cache lines or pages
  – Knowing where each object version currently resides

• New Parallel Programming Models and Runtime Systems know how to achieve these

⇒ Cache Coherence and Paging duplicate the effort of the Runtime!

• Hierarchical Data Structures and Algorithms are needed
Parallel Computation: Graph of Producers-Consumers

- Producer (writer) - Consumers (readers) pattern is universal – not just in stream processing
Recent work on Task-based models where programmer identifies input & output data sets, and Runtime manages their replication / migration: bring local copy before starting up task execution
Live – Dead Words or Lines: Opportunity for Optimizations

- Task input & output data buffer areas have live and dead periods
If we know that the cache line being evicted is dead, we don’t need to write it back, even if marked “dirty”
If we know that we are writing into a dead line, we do not need to:

- have flushed it before
- invalidate other (knowingly dead) copies
- fetch old (dead) contents from last valid holder

- Kaxiras e.a. “tear-off copies”
Fetch Block ("Object") versus Fetch Lines

Coherent Caches

Local (Scratchpad) Memories or Non–Coherent (Expl. Mng’d) Caches

- Large blocks ⇒ save ~ 50% of the network packets
  - although saved packets are small, routing decisions consume energy
Know Where to fetch from, versus ask a Directory

Coherent cache directories tell—in case you don’t already know it:

– where is the most recent (currently valid) copy
– where are all other copies—for invalidation, if unaware of being dead
SARC: Local Mem & RDMA vs Coherent Caches & Prefetch

• SARC project (FET IP – 2006-10) - IEEE Micro Magazine, Oct. 2010:
  • GEMS-based simulation with up to 64 in-order cores
  • MOESI directory-based coherent caches (distributed directories)
    – Hardware strided prefetcher
  • vs. Local (Scratchpad) Memories and (our optimized) Remote DMA
  • GARNET NoC models (concentrated 2D mesh – 4 cores/router)
  • ORION 2.0 NoC power models (65nm)
• Four benchmarks kernels with diverse communication patterns
  – Algorithm and data layout separately tuned for each architecture
  – data set fits on-chip, and stays fixed when # of cores increases
  – Smith-Waterman (64 cores): RDMA 40% faster, vs destructive early prefetching
  – Bitonic Sort (64 cores): RDMA 40% faster, vs prefetcher cannot predict pattern
  – Jacobi (64): RDMA 13% faster; FFT (64): RDMA 16% faster – RemSt 25% faster

Manage Objects, not Lines or Pages - M. Katevenis - BMW'10
• RDMA close to “zero” control volume
• Jacobi (64 cores): RDMA: \textit{4x less volume}
  – caches: cache-lines ping-pong among caches
• FFT (64 cores): RDMA: \textit{2.8 x less volume} – Remote Stores: \textit{1.8 x less volume}
  – caches: barrier synchronization contributes considerable traffic
NoC Energy Analysis

- **Jacobi (64 cores):** RDMA *60% less* NoC energy than prefetching
- **FFT (64 cores):** RDMA *35% less* NoC energy than prefetching
NoC Power Analysis

RDMA reduces total NoC power while prefetching increases it!
- 15% - 30% (64 cores) compared to plain caches
- 20% - 50% (64 cores) compared to prefetching
- higher gains in dynamic power

Injecting less packets clearly improves NoC energy and power consumption

Manage Objects, not Lines or Pages - M. Katevenis - BMW'10
Outline

• Transfer Objects when & where needed: Reduce Network Energy

• New Parallel Programming Models and Runtime Systems know how to achieve these

⇒ Redundant Hardware –avoid duplication of effort:
  – Coherence Directories (where each cache line currently resides)
  – Page Tables (where each “logical” page currently resides)

• Hierarchical Data Structures and Algorithms are needed
Task Input/Output Data Sets Managed by Runtime

• New, promising, task-based parallel programming paradigm:
• Programmer/compiler identifies input & output data sets of tasks;
• Runtime compares these to detect dependencies/parallelism;
• Runtime uses this info to schedule tasks to processors;
• Runtime issues commands to replicate locally the input data set, allocate space for the output data set, run task, notify next tasks.
• E.g.: StarSs, OpenMPT, CellMP, TPC, CellGen, Sequoia, Prometheus

⇒ When these are available – when I/O data sets are known:
• Cache coherence and directories are superfluous, unnecessary
  – runtime explicitly replicates/migrates/invalidatesthe “objects” that constitute the input/output data sets – *data flow style*
Puzzled: Is Virtual-to-Physical Address Translation needed?

...when *Objects* are replicated/migrated by the Runtime...

Virtual Memory is used to:

1. Protection among processes
   - Can solve this in less expensive ways – see next slide
2. Swap pages to disk
   - Runtime knows what it has swapped where, and when to bring back
3. Load and run code at addresses ≠ address compiled at
   - Dynamically-linked libraries have already solved this
4. Migrate pages among various localities of physical memory
   - Task receives pointers from runtime to current I/O data set positions
   - Sub-arrays: index-to-address calculation uses current base address
   - Pieces of large data structures with internal pointers: *Problem!*...
SARC Protection Model without Address Translation

(a) Traditional; (b) SARC, e.g. 8 allowed ranges (base-bound reg’s) per domain
Discussion: Fragmentation, Mem.Space Revocation

1. Aspects of proposed protection (slide 18) and hierarchical data structures (slide 22) remind the MULTICS operating system

2. Paging (slide 17) also resolves the fragmentation problem:
   – malloc large virtually contiguous address space when the free physical space is fragmented
   – counter-arguments:
     – avoid fragmentation anyway, to economize on TLB size/efficiency
     – input/output data sets of “reasonable” size avoid fragmentation
     – data sets of “few” tasks in local memory, at any given time

3. How does the OS (quickly) revoke physical memory space from a process in order to give that to another process that needs it?
   – using access rights (protection mech.), after notifying the runtime

4. Physical memory fragmentation (in-node, across nodes) increases number of entries in hardware protection table
   – response: entire, contiguous nodes allocated to each prot. domain
**Outline**

• Transfer Objects when & where needed: Reduce Network Energy

• New Parallel Programming Models and Runtime Systems know how to achieve these

⇒ Cache Coherence and Paging duplicate the effort of the Runtime!

• Hierarchical Data Structures and Algorithms are needed:
  – Worth the effort – do not expect everything to be done automatically
  – Like Data Base community: disk-resident data structures & algorithms
Small records, randomly linked, scattered all over the memories...

Tasks operate from a distance, using locks & coherent caching

Unknown task data set, except for either tiny task (single record) or huge data set (entire data structure) – hence non replicatable
Need new Pointer Data Structures & Algorithms

local processor for each memory operates onto its substructure(s)

• Like disk-resident data bases: specific data structures & algorithms
• Intra-pointers stay valid upon migration, like relocatable code
• Inter-pointers must go thru runtime tables & dependence checks
Conclusions

• “Object” = unit of task input or output data set (variable size)

• Let the Runtime System keep track of Objects – not pages

• Let the hardware transfer Objects under runtime control – not cache lines under control of simplistic hardware protocols

• Non-hierarchical data structures, with small records allocated at random places, do not scale to massively parallel systems
Backup Slides
Data Transport and Synchronization (1/2)

- RDMA follows closely the “PERFECT” case (1cc memory accesses)
- Smith-Waterman (64cores): RDMA 40% faster
  - HW Prefetcher: early prefetching (destructive)
- Jacobi (64cores): RDMA 13% faster
  - HW Prefetcher: directories contention

Manage Objects, not Lines or Pages  - M. Katevenis  - BMW’10
• Bitonic Sort (64cores): RDMA 40% faster
  – HW Prefetcher: cannot predict the pattern
• FFT (64cores): RDMA 16% faster – Remote Stores 25% faster
  – HW Prefetcher: learning period not amortized
  – RDMA: massive initiation of short RDMAs

Manage Objects, not Lines or Pages - M. Katevenis - BMW'10