Software Defined Memory

Chenzhuo Zhu
Christos Kozyrakis
Bill Dally
06/12/2020
Motivation

- LLC miss costs 100s of cycles – can use software to handle with little additional latency
  - Opportunity to add features (intelligence)
- Applications run across many nodes of a data center
  - Want a consistent view of memory across nodes
- Applications operate on persistent objects in storage
  - Want a consistent view of memory and storage – avoid translation costs
Impact of NUMA

- Machine A: 16 cores, 4 nodes
- Machine B: 24 cores, 4 nodes
- Machine C: 48 cores, 8 nodes

Figure 1.8 – Impact of remote memory accesses on performance of various applications.

Figure 1.9 – Impact of contention on performance of various applications.

Architectural Design

Enabling Software Defined Memory
Design Goal

Tradition Memory Architecture

- Core
- L1: 91.4% Hit Rate, 4 Cycles
- L2: 3.0% Hit Rate, 12 Cycles
- LLC: 3.1% Hit Rate, 46 Cycles

Software Defined Memory

- Core
- L1
- L2
- LLC

Directory Coherence Protocol

Local DRAM: 1.0% CPI, 150 cycles
Remote: 1.5% CPI, 250-1000 cycles
Global Memory Manager
- Local DRAM: >1.0% CPI, 150 cycles
- Remote DRAM: <1.5% CPI, 250-1000 cycles

Expected CPI of Mem Ops
Global Memory Manager and Virtual Cache

- **Global Memory Manager (GMM)**
  - Software layer between cache and DRAM
  - Optimize data placement on DRAM
  - Translate VA to PA

- **Virtual Cache**
  - Data cached under virtual address

- **Protection Lookaside Buffer (PLB)**
  - Reject illegal mem access before cache
  - Larger granularity, hence cheaper
Global Memory Manager with TLB

- Cache translation with TLB
  - Fast path bypasses GMM
  - Avoid unnecessary GMM invocation
GMM Control and Cache Bypassing

- Control and data path from user core to GMM core
  - One sided communication
  - With synchronization
- Enable dynamic policy control
- Bypass cache on critical data
GMM Policy Programming Model

- Message handling model
- Memory policy as collection of message handlers working collaboratively
- Non preemptable for efficiency and simplicity
- Interleaved message handling to exploit parallelism

GMM Hardware

- Input Msg Queue
- Policy Engine
- GMM Core
- Output Msg Queue
- Policy Lookup
- Policy Execution

GMM Software

- Core
- Core Req
- TLB Req
- TLB
- TLB Msg
- Cache Msg
- DRAM Msg
- GMM Msg
- Interconnect
- DRAM Cntlr
- Cache Cntlr

While (true) {
    handleNextMsg(type, addr, ...);
}
Speeding Up Multi-threaded Applications
by exploiting application specific knowledges
Distinct and Dynamic Access Patterns

• Parallel algorithms involve data structures with distinct and dynamic access patterns

• Graph topology is read only (Policy I)
  • Potential for replication

• Access pattern of data and state is dynamic (Policy III)
  • Read from data of current iteration
  • Write to data of next iteration
  • Swap at end of an iteration

• Traditional memory system treats them the same

Policy I: DRAM Replication

• Target static data accessed globally
  • Graph topology
• Use DRAM as extra read-only cache on each NUMA node
• Do not synchronize data
  • Data is never modified anyway
• Avoid cache-line granular metadata as with directory coherence
Policy I: DRAM Replication

- **System Settings:**
  - Machine A: 16 cores, 4 nodes
  - Machine B: 32 cores, 4 nodes
  - Machine C: 32 cores, 8 nodes

- **Application:**
  - BFS: Breadth First Search

Normalized Performance with Replication Policy

Avg Cycles per Mem Inst
Policy II: Subscription Update

• Target data frequently read but rarely written (Canneal, PARSEC)

• Traditional architecture:
  • Invalidate all copies
  • Read cache line in exclusive state and modify the line

• Subscription policy:
  • Data replicated at subscriber nodes
  • Updates go to home node and serialize
  • Home node publishes updates to all subscribers
Policy II: Subscription Update

• System Settings:
  • Machine A: 16 cores, 4 nodes
  • Machine B: 32 cores, 4 nodes
  • Machine C: 32 cores, 8 nodes

• Application:
  • Canneal from PARSEC
Policy III: Atomic Update – Phase 1

- Target graph runtime data during update
  - Frequently updated, never used
- Traditional architecture:
  - Write back modified copy
  - Read cache line in exclusive state and modify
- Atomic policy:
  - Data always resides at home node
  - User cores send messages to home node
  - Home GMM update data locally
Policy III: Atomic Update – Phase 2, Publish

• Target graph runtime data for read
• User core sends message to start new phase
  • User core messages have software defined behaviors
  • Stall user core for synchronization
• Data is published to all nodes
• GMM makes updated data visible
Policy III: Atomic Update

- **System Settings:**
  - Machine A: 16 cores, 4 nodes
  - Machine B: 32 cores, 4 nodes
  - Machine C: 32 cores, 8 nodes

- **Applications:**
  - PR: Page Rank
  - CC: Connected Components

- Incremental performance gain over replication policy
- Overhead due to atomic update contention can be completely eliminated
Policy IV, V, VI...

Identify Critical Data Structures → Analysis Access Patterns → Design Software Policy
Takeaways

• NUMA systems suffer from remote access latency and protocol overhead
• Different data structures have different access patterns that can be exploited
• With such application specific knowledge, software defined memory can optimize coherence protocol to access pattern
Questions?