CS458 Term project proposal

Software-managed Cache Coherence

Sponsor: Sandhya Dwarkadas

Group Member:

XI JIN	propose the idea and write report
Yang Song	do the experiment

1 Introduction

In the conventional small-scale share-memory system, hardware snooping protocol can be used to maintain cache coherence. When the scale is increasing, snooping protocol creates great burden on the bus since snooping protocol requires broadcast messages to communicate. Hardware directory-based coherence protocol can be applied that the information of ownership and state for each cache line is maintained in directory and every message is passing between two nodes which releasing the pressure from broadcast style. The split-transaction bus can meet the requirement. However directory protocol increases hardware design complexity and brings in additional meta-data storage cost for maintaining sharing pattern bit. For instance, maintaining a 1024-core share memory system requires 1024 presence bit for every cache. Assume we have 128KB L1 cache with 64B block size, we have 2048 cache lines and directory need 2048*1024 bit = 256KB = 2 * single L1 size to record presence bit only. Another disadvantage of hardware-managed coherence is that people cannot do anything with the machine shipped. However software-managed cache coherence provides more scalability and it's more attractive because the overhead of detecting conflicting data is moving from run-time to compile time. The basic idea is that we can move cache coherence responsibility from low-level hardware to compiler, high level language and programmer.

2 Motivation

The idea is inspired by software-managed strategy used in DeNovo [2] and Rigel [3]. Both of them propose the Phase(DeNovo)/Domain(Rigel) idea that the task can be divided into different phases. Coherence is guaranteed crossing different phase but it's not ensured within the same phase. R->W and W->W is relaxed in one phase. Comparing to Rigel which requires more sophisticated and complex state transition, DeNovo proposes the idea that the code itself should ensure data-race-freedom. More disciplined language should be provided avoiding conflict data access within

one phase; therefore, a read should simply return the value of the last write that either from the reader's own task or from a task in a previous parallel phase.

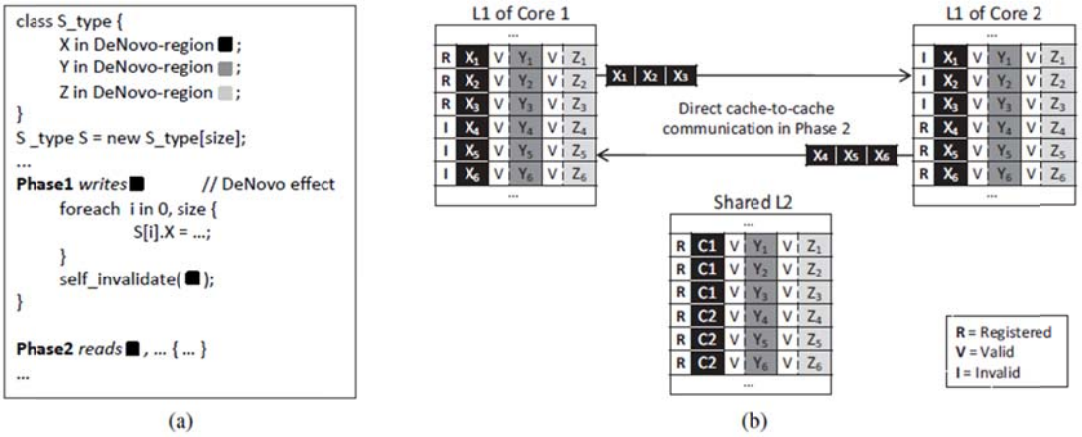

Figure1.Writing and reading from some address will be divided into two phases. At the beginning of phase1, X's tracing bit in shared L2 will be registered changing from Valid state (Data in L2 is up-to-date) to Registered state (Data in one of L1 is up-to-date). At the end of phase1, self-invalidation will be applied until cache line is "touched" in that particular L1. In phase2, according to the information in L2, direct cache-to-cache communication is possible.

DeNovo proposes both tracking state for single-word cache line and multiple-words cache line. The problem is that for application like red-black SOR:

[red]

[red] [black] [red]

[red]

In multiple-words cache line DeNovo, we need 4 phases to complete one iteration with 4 implicit synchronization at the end of each phase.

Phase1 read all red

Phase2 write to black

Phase3 read all black

Phase4 write to red

It costs synchronization overhead. Then we can let the fake conflict (false sharing) exists within one phase by inspection in compile time. The question is that is it worthy to elimination synchronization cost?

3 First Step

Firstly, measure the synchronization cost of a parallel computation (like red-black sor) on a conventional hardware managed coherence with share memory system. Try to find the leverage between least synchronization cost and speedup. Show how will be code working under DeNovo schema and what if we do increase the concurrent within one phase.

Reference

[1] Adve, Sarita V., et al. *Comparison of hardware and software cache coherence schemes*. Vol. 19. No. 3. ACM, 1991.

[2]Choi, Byn, et al. "DeNovo: Rethinking the memory hierarchy for disciplined parallelism." Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on. IEEE, 2011.

[3] Kelm, John H., et al. "A task-centric memory model for scalable accelerator architectures." Parallel Architectures and Compilation Techniques, 2009. PACT'09. 18th International Conference on. IEEE, 2009.

[4]Kelm, John H., et al. "Rigel: an architecture and scalable programming interface for a 1000-core accelerator." ACM SIGARCH Computer Architecture News. Vol. 37. No. 3. ACM, 2009.