Implementing I-structures At Cache Coherence Level

Xiaowei Shen and Boon S. Ang*
MIT Laboratory for Computer Science
(xwshen, hahaha@abp.lcs.mit.edu)

1 Background

$*\mathcal{T}$-$\mathcal{NG}$ is a PowerPC 620-based multiprocessor system supporting both user-level message passing and cache-coherent shared memory [3]. With very limited hardware assistance, the machine implements global cache coherence mostly in firmware/software. This also allows considerable flexibility for experimentation with various protocols [5]. Fine-grain synchronization is provided by incorporating I- [1] and M-structures [2], with their synchronization protocol integrated with the cache coherence protocol.

Figure 1 shows the abstract configuration of a processing site, which contains an application processor (AP), a protocol processor (PP), a site cache, shared memory and a network interface unit (NIU). User applications are performed on the AP, which allows multiple outstanding split-phased memory accesses. The PP is responsible for servicing memory accesses issued from the AP and maintaining the global cache coherence.

Logically there are two components in the PP, a site engine (SEng) and a home engine (HEng). The SEng maintains the cache lines' states in a site directory (SDir) and runs a site state transition engine to process transactions regarding the site cache. The HEng maintains the memory blocks' states in a home directory (HDir) and runs a home state transition engine for the memory blocks whose home is in this site.

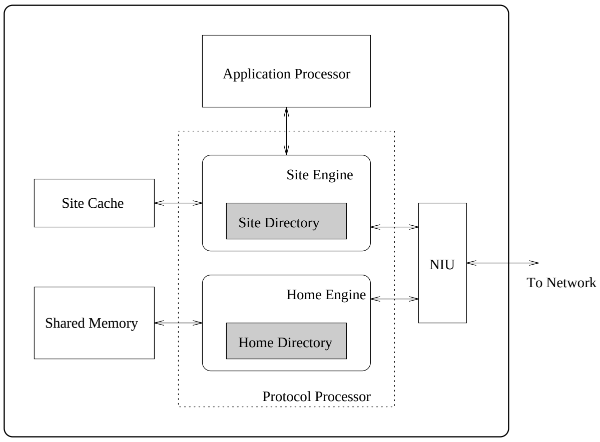

Figure 1: $*\mathcal{T}$-$\mathcal{NG}$ Site Configuration

2 Introduction Of I-structures

An I-structure is a data structure in which each element maintains a presence bit that has two possible states: *full* or *empty*, indicating whether a valid data has been stored in the element. There are two basic I-structure operations: *Iload* and *Istore*.

Iload An *Iload* operation performed on a full element returns the data of the element. An *Iload* performed on an empty element is blocked, and recorded in the element's deferred list for later processing.

Istore An *Istore* operation performed on an empty element writes the value to the element, and modifies the element's state to full. All blocked *Iloads* in the element's deferred list are resumed. An *Istore* operation performed on a full element results in an error.

Initially all the elements are in the empty state. An optional I-structure operation, *reclaim*, can be used to make an element reusable by reseting the element's state to empty, thereby allowing the element to be stored with a new value.

3 Motivation

Previous implementation of caching support for synchronization data-structures similar to I- and M-structures treats the synchronization protocol as a separate layer on top of the underlying cache coherence protocol [4]. However, combining them together can potentially produce more efficient execution.

The write-once characteristic of I-structures allows stale data to be safely tolerated, enabling the *Istore* operation to be carried out without the exclusive ownership of the accessed block. An *Iload* operation performed on a stale cache line will find the presence bit empty, and forward the request to the home site.

Implementing I-structures at cache coherence level also facilitates efficient maintenance of deferred lists. Under the layered approach, a deferred *Iload* normally requires the requesting site to obtain the exclusive ownership. With the integrated design, deferring an *Iload* usually does not need the exclusive ownership at the requesting site. This avoids the potential ping-pong phenomenon that would occur when an I-structure slot is used by multiple sites as a synchronization variable.

Due to space limitation, we will only discuss issues surrounding I-structures deferred list maintenance under a simple, three-state, invalidate-based coherence protocol. Other issues that need to be taken into account in a complete design include the choice of coherence states, the cache line organization and the memory management of deferred lists.

4 Deferred List Management

I-structure element size is typically a word. A cache line may, therefore, consist of multiple elements, including both I-structure and ordinary slots. This significantly complicates the design of coherence protocols that incorporate I-structures, especially the maintenance of deferred lists.

Our simple cache coherence protocol has three stable states for each memory block, namely H-uncached, H-shared

*This research is supervised by Professor Arvind and is supported in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-92-J-1310.

and H-modified, representing the situation that the block is not shared by any site, the block is shared by multiple sites with read-only permission, and the block is exclusively owned by a site with read-write privilege. For a memory block that is in the H-uncached or H-shared state, the deferred lists can be simply maintained at the block's home. For a memory block with the H-modified state, however, it is not clear whether the deferred lists should be maintained at the block's home, or at the modified site.

4.1 Maintaining Deferred Lists At Home

A straightforward method is to have the deferred lists always maintained at the accessed block's home, regardless of the block's HDir state. For a modified block, the memory keeps accurate knowledge about the presence bits of all the individual I-structure slots, although the block's ordinary slots may have been modified. With this presence bit knowledge, the home's HEng never sends protocol request to the modified site to request the data of an empty I-structure slot.

This approach has two serious drawbacks: (1) If an *Iload* operation performed at the modified site needs to be deferred, it has to be sent to the home where all the deferred lists are maintained, although the exclusive ownership is currently resident at the modified site. (2) Any *Istore* operation performed at the modified site requires that the modified cache line be written back to the memory, regardless of whether the home's deferred list for the I-structure slot is empty.

A simple improvement can be achieved if the SEng at the modified site maintains a *deferred flag* (Dflag) for each I-structure slot whose presence bit is 0, recording whether the slot's deferred list at the home site is empty. When an *Istore* operation is performed in the modified site, the SEng checks the corresponding Dflag, only writing back the modified cache line when the slot's Dflag has been set to 1.

4.2 Maintaining Deferred Lists With Ownership

Maintaining the deferred lists with the accessed block's ownership gives rise to an obvious advantage: when an *Iload* operation performed in the modified site needs to be deferred, the SEng locally appends the *Iload* in the slot's deferred list. Moreover, when an *Istore* takes place, the SEng knows whether the modified cache line needs to be written back to the memory, since all the deferred *Iloads* information is available at the modified site.

This approach requires that, for a memory block, the deferred lists be transferred whenever the block's ownership is changed. In other words, when the home or any site obtains a block's ownership, it also receives the block's deferred lists. An optimization that avoids moving the entire deferred lists over the network only requires the transfer of the deferred lists' head pointers. In this case, the site that currently owns the memory block always maintains the head pointers of the corresponding deferred lists, from which all the deferred *Iloads* stored in the network can be retrieved.

4.3 Approach Adopted In $*\mathcal{T}$-$\mathcal{NG}$

For a memory block with the H-uncached or H-shared state, a deferred *Iload* is maintained in the block's home. For a memory block with the H-modified state, a deferred *Iload* is allowed to be recorded either at the slot's home deferred list which is maintained at the block's home, or at the slot's site deferred list which is maintained at the modified site, depending on the time and place at which the *Iload* operation is performed.

For a modified memory block, the modified site's SEng maintains a Dflag for each individual I-structure slot whose presence bit is 0, indicating whether the slot's deferred list at the home is empty. When an *Iload* needs to be deferred at the modified site, the SEng handles building the deferred list entry locally. In this case, no protocol message is generated.

When an *Istore* is performed at the modified site, the SEng resumes all the deferred *Iloads* queued locally, and then checks the slot's Dflag. If the Dflag has been set to 1, the SEng changes the SDir state to C-shared, and sends the most up-to-date data, along with the remaining deferred *Iloads* that are maintained at the site for other I-structure words in the same cache-line, to the home. If the Dflag is not set, the SEng does not change the SDir state.

When an *Iload-req* is received at the home for a modified memory block, the HEng checks the slot's presence bit in the memory. If the presence bit is 1, the HEng sends a protocol message to the modified site to request the block's most up-to-date data. When the presence bit is 0, the HEng either records the *Iload-req* in the home's corresponding deferred list if the deferred list is currently not empty, or sends a *fwd*-request containing the *Iload-req* to the modified site if the home's deferred list of the slot is empty.

5 Current Status

The work described here is still under progress. There are many options that need to be evaluated with benchmarks and we are working on a simulator to support this experimentation. Integrating synchronization and cache coherence protocols introduces additional design choices and complications, but may potentially improve the overall performance. Actual performance results from simulations will provide a clearer picture of which features are important for performance, and which ones should be avoided in order to simplify the protocol.

References

[1] Arvind, Rishiyur Nikhil, and Keshav Pingali. I-structures: Data Structures for Parallel Computing. CSG Memo 269, Laboratory For Computer Science, MIT, March 1989.

[2] Paul S. Barth, Rishiyur S. NiKhil, and Arvind. M-structures: Extending a Parallel Non-strict Functional Language with State. CSG Memo 327, Laboratory For Computer Science, MIT, March 1991.

[3] Derek Chiou, Boon S. Ang, Arvind, Michael J. Beckerle, Andy Boughton, Robert Greiner, James E. Hicks, and James C. Hoe. StarT-NG: Delivering Seamless Parallel Computing. CSG Memo 371, Laboratory For Computer Science, MIT, February 1995.

[4] David Kranz, Beng-Hong Lim, Anant Agarwal, and Donald Yeung. *Low-cost Support for Fine-grain Synchronization in Multiprocessors.* Kluwer Academic Publishers, 1994.

[5] Xiaowei Shen. Implementing Global Cache Coherence In *T-NG. Master thesis, Department of Electrical Engineering and Computer Science, MIT, April 1995.