Design and Performance of SMPs With Asynchronous Caches

Fong Pong, Michel Dubois*, Ken Lee
Computer Systems and Technology Laboratory
HP Laboratories Palo Alto
HPL-1999-149
November, 1999

E-mail: fpong@hpl.hp.com
kenlee@exch.hpl.hp.com
dubois@paris.usc.edu

Asynchronous, cache coherence, shared-memory multiprocessor, SMP

We propose and evaluate a cache-coherent symmetric multiprocessor system (SMP) based on asynchronous caches. In a system with asynchronous caches, processors and memory controllers may observe the same coherence request at different points in time. All protocol transactions are uni-directional and processors do not report snoop results. The need for an extensive interlocking protocol between processor nodes and memory controller which is characteristic of snooping buses is thus removed.

This design overcomes some of the scalability problem of a multi-drop shared-bus by employing high-speed point-to-point links, whose scalability prospects are much better than for shared buses. Memory and processors communicate through a set of queues. This highly pipelined memory system design is a better match to emerging ILP processors than bus-based snooping. Simulation results for ILP processors show that the shared-bus design is limited by its bandwidth. By contrast the parallel link design has ample bandwidth and yields large performance gain for the transaction processing and scientific benchmarks that we have considered.

Besides higher performance the asynchronous design we propose considerably simplifies the behavior expected from the hardware. This is important because snooping bus protocols are so complex today that their verification has become a major challenge.

* Department of Electrical Engineering-Systems. University of Southern California Los Angeles, California

1 Introduction

Because of their simplicity, shared-bus SMPs are a dominant architecture for small-scale systems. By taking advantage of the broadcast nature of a shared-bus, snooping protocols are the de facto schemes for achieving cache coherence. Figure 1 shows the basic configuration to support a four-states *MESI* protocol [22] which is widely used in cache-coherent shared-memory multiprocessor systems. In such a system, every processor is associated with a bus watcher (a "snooper") which monitors all bus activities. When a processor initiates a coherence transaction such as a load miss on the bus, all snoopers in all processors latch in the request. These snoop requests consult the local caches, take necessary actions and respond with appropriate snooping results. Each protocol transaction on the bus is deemed complete when all caches have reported their snoop result.

Although the snooping bus-based design is a classic, well-understood design and offers many good features, it is becoming harder to design a shared bus-based SMP that keeps pace with emerging ILP processor technology.

First and foremost, the multi-drop bus architecture is reaching its speed limit. When the clocking speed was low, the electrical length of the bus was short enough that distributed behavior of the bus could be ignored. However, as bus speeds increase, the processor boards connected to the bus behave as stubs resulting in reflections and ringing of bus signals. There exist several schemes for terminations and signaling to reduced reflections, but none solves the fundamental problem of stubs. Because of design constraints such as heat dissipation the space needed between stubs is longer at high speeds. This limits the operating speed of buses to 150MHz in current systems.

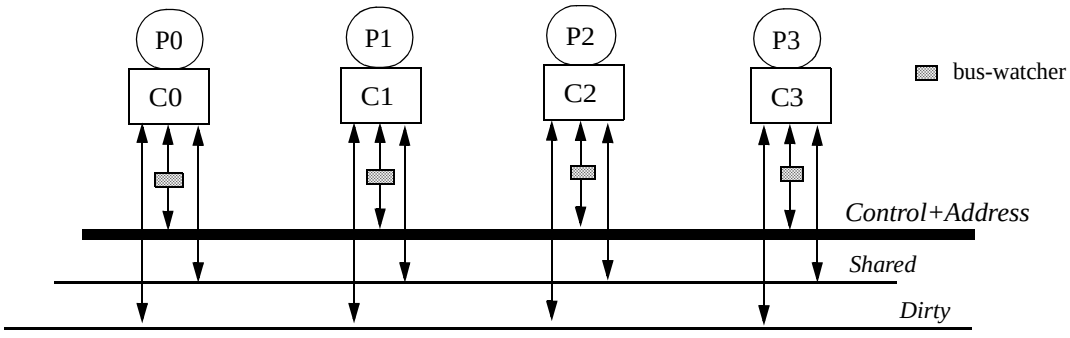

Figure 1 A Bus-based Cache System with Shared/Dirty Signals.

For future processor designs with deep speculation, multiple cores, and/or multithreading [13, 14,

23], the shared-bus will no doubt become a major bottleneck even in small multiprocessor configurations. An alternative is to have multiple short bus segments bridged by the memory controller [15, 18]. This approach has its own limitations. It is difficult to connect more than two full busses to the memory controller without an inordinate number of I/O pins. Furthermore, all bus transactions that occurs in one bus segment must be propagated to other bus segments unless the memory controller is equipped with *coherence filters*. A coherence filter essentially keeps track of memory blocks that are cached in the bus segment. Regardless of these possible extensions, the fundamental problem of a shared-bus design remains. For instance, every request must start with an arbitration cycle and spend one cycle for bus turnaround at the end. The protocol sets an upper bound on the maximum attainable throughput.

Secondly, the bus snooping scheme requires all processors (and all snooping agents such as I/O bridge chips with I/O caches) to synchronize their responses. Generally speaking, the snoop results serve three purposes: 1) they indicate the *safe* completion of the snoop request in the cache hierarchy of the local processor, 2) they provide *sharing* information, and 3) they identify which entity should respond with the missing data block, i.e, either another processor or memory. For the purpose of illustration (see Figure 1), assume that the snooping results are propagated to all processors via two bus lines *Shared* and *Dirty*. For a load miss, the processor may load the data block into the exclusive (*E*) or shared (*S*) state depending on whether the *Shared* or *Dirty* signal is asserted. In the case where a processor has the most recently modified copy of the requested data, it asserts the *Dirty* signal preventing the memory from responding with the data.

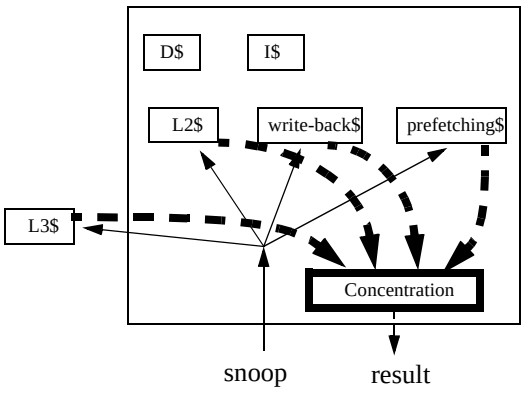

Figure 2 Illustration of Snooping Paths in Modern Multiprocessors.

The common approach is to require that all caches connected to the bus generate their snoop

results in exactly the same bus cycle. This requirement imposes a fixed latency time constraint between receiving each bus request and producing the snoop result and this fixed snooping delay must accommodate to the worst possible case. This constraint presents a number of challenges for the designers of highly-integrated processors with cache hierarchies. As illustrated in Figure 2, many modules must be snooped and the final results must be concentrated. In order to meet the fixed latency constraint, the processors may require ultra-fast snooping logic paths. The processor may have to adopt a priority scheme assigning a higher priority to snoop requests than to requests from the processor's execution unit.

More relaxed designs such as Intel's P6 bus [16] allow processors and memory controllers to insert wait states when they are slow to respond. This scheme complicates the logic design because every processor must closely monitor the activities of other processors on the bus in order to re-generate snooping results when wait states are observed. In the case of Intel's bus, for instance, the processor must repeat its snooping result two cycles after observing a wait state cycle.

Yet another approach to synchronizing snoop results is the use of a shared wired-or *Inhibit* signal on the bus as was implemented in the SGI Challenge [12]. Processors may snoop at different speeds but must report the end of their snoop cycle on this new bus line. The transaction remains "open" for as long as any processor has not pulled down its *Inhibit* line. Again this interlocking of processor and memory signals on the bus results in complex bus protocols and bookkeeping of pending transactions in the interface of each processor, which in fact limits the number of concurrent transactions on the bus. The design still relies on a shared line and on bus watching and is complex to verify. This complexity increases with the number of concurrent transaction tolerated on the bus. Current designs are so complex that verification has become a major development cost.

In this paper, we propose an efficient cache-coherent shared-memory multiprocessor system based on an *asynchronous-snooping* scheme. In this paper *asynchronous* refers to a model of cache-coherent systems first theoretically introduced and proven correct in [2]. It does not refer to variable-delay snooping using an inhibit line as described above. In fact it does not require reporting snoop results or synchronizing the snoops in any way. Because of this simplification, fast, high-bandwidth point-to-point links can be used to communicate requests and data between processors and memory. Snooping requests to different processors are propagated independently through queues. The number of pending and concurrent protocol transactions is only limited by the size of

these FIFO queues. However by emulating a shared-bus broadcast protocol, the topology of the point-to-point interconnection is made transparent to the processors and the memory.

The various design aspects of the new system are given in Sections 2 and 3. Section 4 is dedicated to the evaluation methodology and system simulation models. We compare the effectiveness of various bus-based configurations with our parallel link design. These results are presented and discussed in Section 5 and our final comments conclude the paper in Section 6.

2 A New Asynchronous Cache System Design

2.1 The Architectural Organization

We advocate an asynchronous cache system such as the one shown in Figure 3. In this design, processors and memory controller communicate via unidirectional high-speed links [10, 25]. A set of queues buffer requests and data blocks in case of access conflicts and also serve as adaptor between data paths of different width.

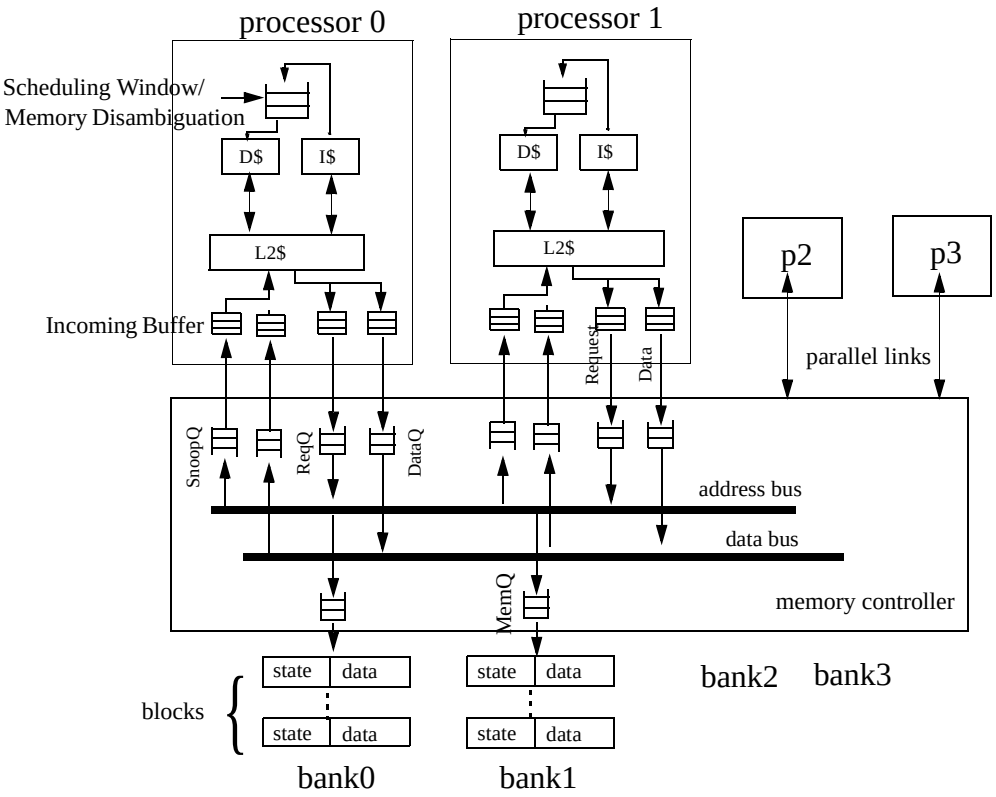

Figure 3 The Proposed Asynchronous Cache System.

On the memory controller side, received requests are stored in a request queue. Through a high-

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.