Scalable Subspace Snooping

Jaehyuk Huh Doug Burger
Computer Architecture and Technology Laboratory
Department of Computer Sciences
The University of Texas at Austin
cart@cs.utexas.edu - www.cs.utexas.edu/users/cart

Department of Computer Sciences
Technical Report TR-06-41
The University of Texas at Austin

August 2006

Abstract

Snooping tag bandwidth is one of the resources that limits the number of processors that can participate in a cache-coherent snooping system. In this paper, we evaluate a type of coherence protocol called subspace snooping, which decouples the snoop tag bandwidth from the address bus bandwidth. In subspace snooping, each processor snoops a set of logical channels, which are a subset of the total snoopable address busses in the system. Thus, each processor snoops a subset of the address space, reducing the number of tag matches required for a system of a given size. By dynamically assigning both processors and cache lines to channels, we support dynamic formation of subspaces, with the goal of having only sets of processors that share data snooping on each given channel.

Subspace snooping aligns best with systems for which the address bus bandwidth greatly exceeds the snooping tag bandwidth. Snooping optical interconnects exhibit such characteristics, providing enormous transmission bandwidth, but which quickly become limited by snooping tag energy and bandwidth as the number of processors increases. Optical busses can be subdivided into logical channels using either wave-division or time-division multiplexing, making them good candidates for a subspace snooping implementation. We evaluate a range of subspace snooping protocols on six parallel scientific benchmarks, running on an execution-driven simulator. We show an average 50% reduction in total snoops for 64-processor systems with 8-32 channels, and that for regular applications, subspace snooping shows large performance improvements the ratio of processor bandwidth to snoop bandwidth grows large.

[0]This material is based upon work supported by the Defense Advanced Research Project Agency (DARPA) under Contract NBCH30390004.

1 Introduction

The difficulty and expense of supporting cache coherence has limited scaling to very large shared-memory multiprocessors. The largest message-passing clusters now number in the tens of thousands of processors, but shared-memory machines have not kept pace. This divergence is in part due to the financial cost of developing custom hardware for large-scale shared-memory machines, and in part due to the complexity of the protocols.

The two broad classes of coherence protocols, snooping protocols and directory protocols, have traditionally targeted different scales of systems. Snooping systems offer low-cost, simple coherence at small system scales (two to a few tens of processors). Directory protocols have been built to scale much higher, but at great cost and complexity. Directory protocols also suffer from the latency of numerous point-to-point messages in a large-scale system, which can be reduced at the expense of additional protocol complexity.

Snooping protocols do not scale to large numbers of processors for three major reasons: bus bandwidth, bus speed, and snoop tag bandwidth. As more processors snoop a single address bus, the traffic on the bus grows linearly with the number of processors. Sun's Wildfire system mitigated this problem by providing multiple address-interleaved snoop buses [11] supported by a point-to-point data transfer network. This solution did not mitigate bus latency, since the bus backplane must still be routed to all snooping processors, resulting in long traces and slow bus speeds.

Perhaps the worst factor for scaling, however is snooping tag bandwidth. Snoop bandwidth has not traditionally been seen as a problem since the snoop tags bandwidth is matched to the address bus bandwidth. As the number of processors grows, however, the number of snoops that happen system-wide grow as $O(n^2)$. For each request that each processor puts on the snooping bus, n processors must snoop the request. For multiple interleaved address buses, providing the L2 (or L3) tag bandwidth for snooping–even with replicated tag banks–quickly becomes a performance and energy bottleneck. The ideal large-scale hybrid protocol would allow many processors to share a high-bandwidth, low-latency bus, but also to have scalable snoop tag bandwidth. Viewed another way, in the ideal system, processors would only snoop the bus transactions for operations on data that were often in their caches.

In the context of this paper, *subspaces* are regions of data that are consistently shared by a stable subset of processors. These subspaces can be dynamically evolving, so long as they are stable for sufficiently long to be useful. For example, the faces on a cubic data decomposition in a computational fluid dynamics (CFD) code are shared by stable pairs of processors for the duration of the application, assuming that it is a non-adaptive code.

This paper evaluates subspace snooping protocols, a class of protocols that attempts to exploit stable subspaces to improve snooping scalability and energy efficiency. In a subspace snooping system, the coherence interconnect is divided into some number of snoopable data channels (address buses being one example). The total address bandwidth, however, would be larger than an individual snooping processor could handle, so each individual processor could only snoop a subset of the channels. Ideally, stable sets of processors sharing a subspace would allocate that subspace to a single channel shared by that set of processors only. Processors not sharing that subspace would be snooping other channels, avoiding the energy and delay costs of unnecessary snoops. Figure 1 shows the difference among traditional snooping, directory coherence, and subspace snooping coherence. While subspace snooping does not achieve the sharing-list precision of directory protocols, it has the potential to offer many of the benefits of snooping protocols with a reduced number of total global snoops.

Subspace snooping protocols are likely to be best in systems where the aggregate available bus bandwidth exceeds the snooping tag bandwidth, and in which large numbers of processors can be partitioned into regular and fairly stable sharing sets, such as with many technical application. Optical buses may be an excellent match for the former constraint. Many processors may share them, their latency scales significantly

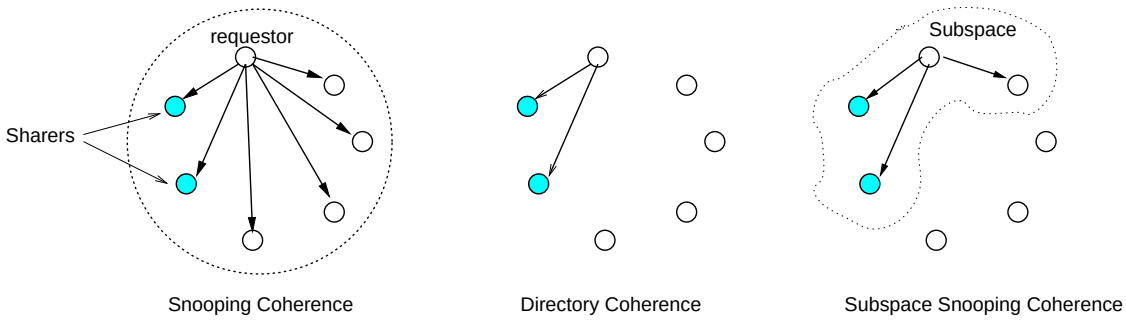

Figure 1: Snooping coherence, directory coherence and subspace snooping

better with added processors than do electrical buses, many channels may be implemented using wave- or time-division multiplexing, and, most important, the available bandwidth on the optical link greatly exceeds what a set of snoop tags can support.

In this paper, we evaluate a number of subspace snooping protocols on an optical bus framework. We simulate six scientific benchmarks–one from the NAS suite and five from Splash–on a simulated 64-processor system. We evaluate policies for assigning cache lines (or larger regions) are assigned to channels, and how the processors each select their set of channels to snoop. processors choose specific channels to snoop.

In the rest of this paper, we first review the limitation of snoop tag lookups for both energy and performance, and present optical buses and related work in Section 2. In Section 3, we define subspace snooping and discuss the mechanisms necessary for guaranteeing correct coherence. In Section 4, we present two subspace snooping protocols and a policy for subspace formation. In Section 5, we measure and evaluate the reduction in snoops that can be provided with such protocols, and we conclude in Section 6.

2 Scaling Snooping Cache Coherence

Traditionally, the bandwidth of address and data buses has limited the scalability of snooping cache coherence systems. Bus bandwidth has been increasing with faster system clocks, wider buses, split transactions, separate address/data buses and switched networks. However, the cost of wide electrical buses still limits the expansion of snooping coherence systems. Recently, there has been a significant improvement of the cost-performance of optical interconnects for multiprocessor systems. Optical interconnects can provide large bandwidth with relatively low cost and energy consumption, thus improving the scalability of snooping coherence systems. However, even though optical interconnects can provide significant increases in address bus bandwidth, snoop tag bandwidth will still limit the scalability of snooping systems, since all processor' snoop tags must respond to every bus transaction.

In this section, we first show that snooping systems will be limited by snoop tag bandwidth in terms of both energy and performance scalability. We also describe current-generation optical interconnection technologies for multiprocessors and other snooping-related work.

2.1 Power Limitation of Snoop Tag Lookups

As multiprocessor sizes increase to solve larger problems, the sizes of data sets also typically scale with the number of processors. Consequently, to support the increased coherence traffic, address bus bandwidth must increase linearly with the number of processors. Since all snooping tags must perform a lookup for

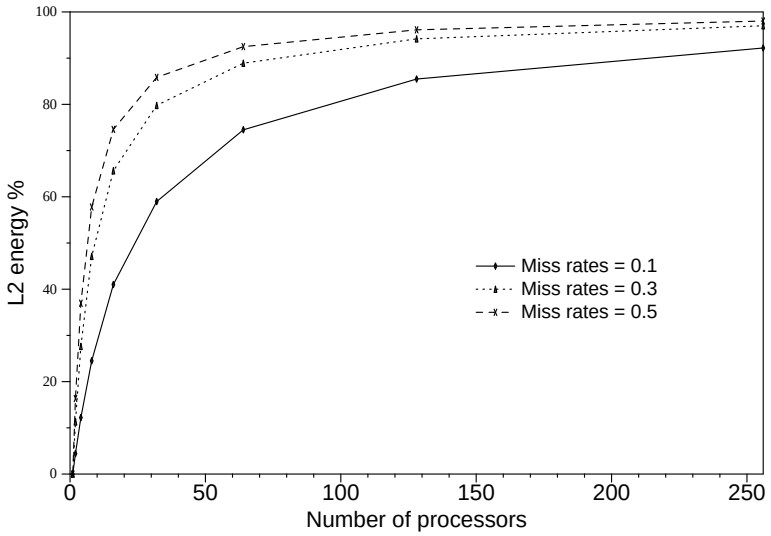

Figure 2: Energy consumed by snoop tag accesses (% of total dynamic cache energy)

every bus requests, the total number of snoop tag accesses will increase quadratically with the number of processors. Assuming the data set size scales linearly with the number of processors (N_p), and that the number of L2 misses per processor ($Misses_{L2}$) is a constant and independent of N_p, the number of bus transactions is $N_p \times Misses_{L2}$. Since every tag should be snooped for each L2 miss, the number of total snoop lookups is $Np \times Np \times Misses_{L2}$

The quadratic increase of the number of tag snoops will consume significant and growing power to access the tag arrays and drive the I/O pins. A recent study showed that a significant fraction of L2 cache energy is consumed for snoop tag lookups [20]. We use a simple model to demonstrate the extent to which the energy consumption of snoop tags will increase as the number of processors in SMPs increases. We assume that each processor incurs the same number of L2 misses and the number of L2 misses per processor is constant across different numbers of processors, and that the data set scales with the number of processors. $Energy_{Data}$ and $Energy_{Tag}$ are the energy used for each access to the data array and tag array in the local cache. $RemoteHits$ is the rate for missed blocks to be found in one of the remote caches. The model for the ratio of snoop energy to the total energy ($Ratio_{Snoop}$) is as follows:

$Energy_{DataAll} = Energy_{Data} \times (1 + Misses_{L2} \times RemoteHits)$

$Energy_{Snoop} = Energy_{Tag} \times (Np - 1) \times Misses_{L2}$

$Energy_{TagAll} = Energy_{Snoop} + Energy_{Tag} \times (1 + Misses_{L2})$

$Ratio_{Snoop} = Energy_{Snoop}/(Energy_{Data} + Energy_{TagAll})$

Figure 2 shows the fraction of L2 energy consumed for tag snooping, as the number of processors increases. At 128 processors, with 10% of L2 miss rate for each processor, tag lookups consume more than 80% of the total L2 dynamic energy. With the miss rates of 20% and 30%, snoop tag lookups consume more than 95% of the dynamic L2 energy at 128 processors.

2.2 Snoop Tag Bandwidth Analysis

In conventional snooping protocols, snoop tag bandwidth is tightly coupled with bus bandwidth. Since the bus bandwidth has traditionally been lower than the than the potential bandwidth of snoop tag arrays, bus bandwidth has been the focus of research to scale snooping coherence [2, 17, 15]. However, even though optical links with wave-division multiplexing can greatly improve the address network bandwidth, snoop

Bus Systems	SGI Challenge	Sun Gigaplane-XB	Sun Fireplane
Bus Bandwidth	47.6M/sec	167M/sec	150M/sec
Maximum Processors	36	64	24
Processor Clocks	150MHz	300MHz	750MHz
Transactions/1K cycle/processor	8.8 transactions	8.6 transactions	8.3 transactions

Table 1: Bandwidth requirements for past SMP systems

tag bandwidth remains constrained by tag array speed and pin bandwidth. Although multi-ported snoop tags can increase tag bandwidth, multi-porting is not a scalable solution for large tag arrays due to large area and energy overheads.

Table 1 shows a simple extrapolation of snoop bandwidth for three SMP systems. Estimated from the snoop bandwidth and processor clock speed data, the last row shows how much snooping bandwidth was provided to each design, normalized to the processor clock speeds at the years when the buses were first introduced. For all three systems, approximately 9 bus transactions/1K cycles should be sustained for each processor. If we make the unrealistic assumption that a large L2 or L3 snooping tag array can service a snoop request every processor clock cycle, 1000 transactions/1K cycle are reached at about 110 processors, for this set of systems and snooping bandwidths.

2.3 Optical Interconnection Technologies for Snooping Cache Coherence

Optical interconnection technologies for multiprocessors have been improving rapidly. Arrays of Vertical Cavity Surface Emitting Lasers (VCSELs) and arrays of photodectors (PDs) provide inexpensive and fast electro-optic and opto-electric conversion [18, 26]. In current technologies, a VCSEL can transmit 3-5 Gb/s and an array of the VCSELs can achieve 200-300 Gb/s transmission rates [14]. In addition to traditional optical fibers, polymer waveguides enable dense board-level optical interconnections [26]. These optical device advances have made optical interconnects a high-bandwidth alternative for traditional electrical buses in multiprocessors.

The optical interconnects have two advantages over traditional electrical wires for snooping multiprocessor systems. First, the bandwidth/cost and bandwidth/power of optical interconnects are superior to those of electrical wires. In optical links, furthermore, multiple wavelengths can co-exist in a single optical fiber [3, 21]. Such wave-length division multiplexing (WDM) can multiply the interconnection bandwidth without adding more physical links. The number of wavelengths are typically limited by the cost and latency of electro-optic/opto-electric conversion devices. Optical links for wide area networks, in which the bandwidth per distance is more important than the conversion latency of optical and electrical signals, use a dense WDM with tens or even hundreds of wavelengths. However, in current technologies, the optical interconnects for multiprocessors are constrained by the conversion latency and can support coarse-grained WDMs with 4-12 wavelengths, but the number of wavelengths are likely to increase as the optical technologies mature.

Second, optical interconnection can broadcast signals efficiently with passive components. In optical interconnects, high fan-outs at high frequencies are feasible. A passive star coupler can provide all-to-all connectivity over hundreds of nodes. This broadcast capability makes the optical links desirable for snooping networks. In current technologies, hundreds of fan-outs are possible without a significant loss of signal strength. The optical broadcast can be more power-efficient than hierarchical packet switched buses that are widely used for snooping address buses. Furthermore, the topology of optical interconnections can be simple and easily extended for more processors. Current electrical interconnections in multiprocessors use switched networks, the topology of which is fixed and not modularized. Optical interconnects can connect multiple processors with passive star couplers, which simply divide light signals into multiple links [9].

Recent studies have shown that off-the-shelf parallel optical fibers can be used for multiprocessor interconnections. The Lambda-connect project at Lawrence Livermore National Laboratory and Multi-wavelength Assemblies for Ubiquitous Interconnects (MAUI) demonstrated that multi-wavelength parallel optical interconnect (MPOI) can greatly improve the bandwidth of multiprocessor systems [13, 23]. In both projects, parallel multi-mode fiber ribbon cables with 10-12 wires have been used for short distance optical interconnects with 4 channel WDM. In MAUI, an array of 48 VCSELs and 48 photodetectors were used to construct 12 wires and 4 wavelengths/wire. For each fiber, signals from 4 VCSELs with different wavelengths are multiplexed. Four photodectors in the receiving port pick up four different wavelengths.

Due to these technologies, high-bandwidth optical interconnects are becoming more viable. However, the availability of this high bandwidth will necessitate innovations in snooping, which researchers have been exploring in both optical and electrical contexts.

2.4 Related Snooping Work

In addition to the high bandwidth, the efficient broadcast capability of optical links has led to several projects exploring optical snooping bus designs [22, 1, 7]. In SPEED, optical buses have multiple WDM channels and the channels are divided into a shared channel and multiple private channels. Bus requests for writes (ownership request) are sent over a shared channel, and read requests are sent to memory through private channels [10]. SYMNET used passive Y-splitters/couplers to connect processing nodes in a tree [14]. SYMNET modified a coherence protocol to eliminate combined snoop results from all processors.

Other work has looked at improving effective snoop bandwidth in a conventional, electrical context. In *Jetty*, coarse-grained filters between snoop tags and a bus are used to discard snoops on the blocks which are not in caches [20]. The coarse-grained filters are much smaller than the cache tags, consuming less energy. *RegionScout* exploits the observation that there are large continuous private regions, which are not shared by other processors [19]. The RegionScout filters detect private regions and use the detected regions to reduce unnecessary snoop tag lookups.

Subspace snooping contains elements of both traditional snooping and directory protocols. Other recent work on improving coherence bandwidth has also used elements of snooping and directories adaptively in one system [17], extended by multicasting snoop requests based on the prediction of potential sharers [2, 15], and using a token-based coherence mechanism on fast but un-ordered switched networks [16].

3 Subspace Snooping Coherence Architecture

Subspaces are regions of data that are consistently shared by a stable subset of processors. A subspace is represented by a partition of address space (data) and a subset of processors (sharers), which share the address partition. To maintain coherence for a cache block, bus requests should be delivered to the processors in the subspace the block is mapped to. Subspaces are not static, but may dynamically evolve during program execution.

Figure 3 depicts conventional snooping and subspace snooping coherence. In conventional snooping protocols, all processors snoop every bus request, incurring a large number of snoop tag lookups. Even if the address bus consists of multiple address-interleaved buses, processors must snoop all the buses. In subspace snooping, we divide the available bus bandwidth into multiple channels. Coherence messages for a subspace are delivered through the channel designated to the subspace. The processors in a subspace must snoop the channel for the subspace. In Figure 3 (b), there are three channels and each processor snoops only two channels. For example, P1 snoops only channel 1 and 3. The channel 1 is snooped by P1, P3, and P4, so a bus transaction on the channel 1 will cause three tag lookups. The last channel is reserved for a fully

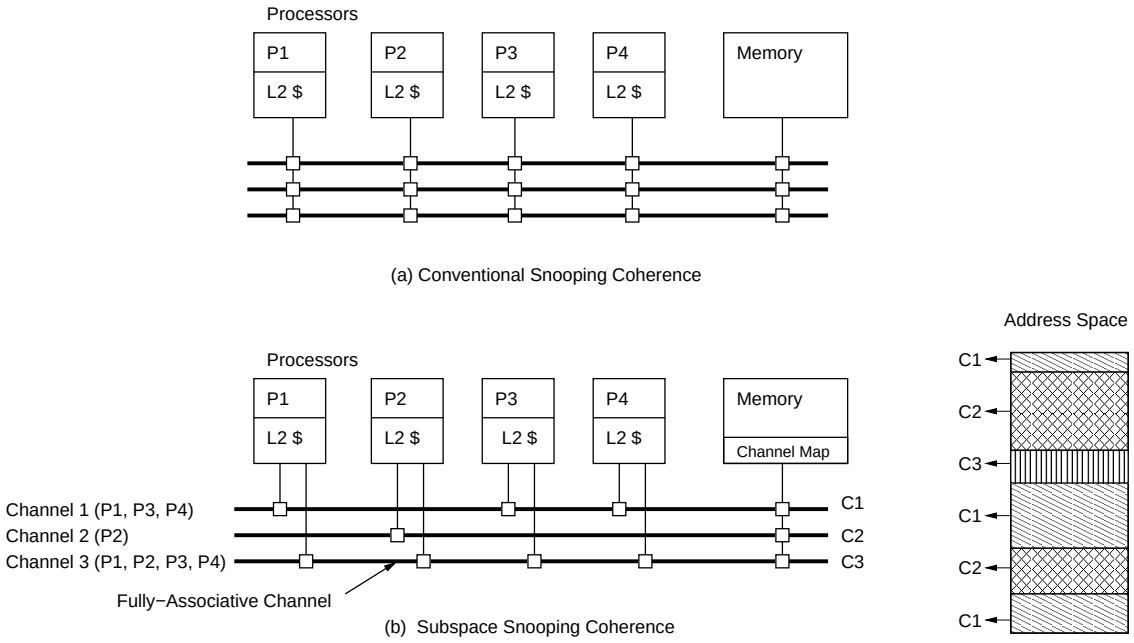

Figure 3: Conventional snooping and subspace snooping coherence

associative channel, which is snooped by all processors. The fully associative channel covers the subspace shared by all processors or the subspaces which can not be mapped to any other channels.

There are three key issues to design subspace snooping coherence:

- *Mapping data and processors to subspaces:* The physical address space is partitioned and mapped to subspaces. Subspace snooping should have a mechanism to maintain the address mapping and to route bus requests to correct channels.
- *Guaranteeing correctness:* Subspace snooping should guarantee the correctness of coherence by preventing processors from caching addresses which the processors do not snoop. Since subspaces are not static, processors may attempt to access addresses mapped on un-snooped channels.
- *Forming optimized subspaces:* Subspaces should be formed to minimize snoop tag lookups. Subspace snooping should identify frequently communicating processors and group them to share a channel.

Section 3.1 presents a formal definition of subspace snooping and the correctness invariant. In section 3.2, we will discuss issues to implement subspace snooping and compare it to directory protocols.

3.1 Formal Definition and Correctness

Subspace snooping partitions the physical address space and maps the subspaces to logical channels. A processor snoops a subset of the total channels. Formally, for a set of processor $P = \{p_1, p_2, ..., p_n\}$ and address space $A = \{a_1, a_2, ..., a_m\}$, subspace snooping can be defined with the following components:

- A set of channels, $C = \{c_1, c_2, c_3, ..., c_x\}$
- Channel mapping, $ChannelMap = \{a \Rightarrow c \mid a \in A\ ,\ c \in C\}$

- Processor mapping, $ProcMap = \{p \Rightarrow sc \mid p \in P\ ,\ sc \subset C\}$

Each processor snoops a subset of channels ($ProcMap$), and each block address is mapped to one channel ($ChannelMap$). A *channel directory* maintains the channel mapping between addresses and channels. For each block of memory, the channel directory has the corresponding channel id. For a bus request, the channel directory should route the request to a correct channel. In our baseline protocol, a node must first send a bus request to the channel directory, which will put the request on the correct channel. Due to the two-step bus transactions to route bus requests on correct channels, the baseline subspace snooping requires three-hop cache-to-cache transfers, increasing communication latencies.

To guarantee the correctness of subspace snooping, one condition must be satisfied. A processor must snoop a channel, if at least one address in its cache is mapped to the channel:

- **Correctness invariant:** *For each address A_i, cached in processor P_j, P_j must snoop channel C_k to which the address A_i is mapped.*

If a processor attempts to read a block into its cache and does not snoop the channel mapped to the block, the correctness invariant can be violated. We define such violation as *channel conflict*. To resolve the channel conflicts, subspace snooping can take one of two actions: 1) move the conflicting address to another channel, which the requesting processor already snoops, or 2) make the processor snoop the new channel. In both cases, there is some cost to resolve such channel conflicts.

To move the mapping of an address to a new channel, the address should be invalidated from all the caches that snoop the old channel, if the caches do not snoop the new channel. The block can not reside in the caches, if the caches no longer snoop the channel to which the block is newly mapped. If a request is an upgrade or read for ownership, such invalidation does not cause any performance degradation, since the request will invalidate the block anyway. However, if a request is a read for sharing, the invalidation may cause subsequent misses from other caches.

3.2 Implementation Issues

In optical interconnects, channels can be created in various ways. First, when an optical bus consists of multiple optical fibers, each channel can use a separate set of fibers. Second, for the same set of fibers, wavelength division multiplexing (WDM) can make separate channels, assigning a wavelength to each channel. Third, for the same set of fibers and a wavelength, time division multiplexing (TDM) can use different time slots to distinguish different channels. Any combination of these techniques can be used to create logical channels. With these techniques, snoop interfaces can discard requests on un-snooped channels without consuming any pin bandwidth.

The initial bus requests from processors to the channel directory do not need to use the broadcasting address bus. Instead, subspace snooping uses un-ordered data networks to send the initial requests to the channel directory, to avoid using the broadcast bandwidth of address bus. Subspace snooping reduces the latency increase from indirect bus accesses by embedding channel ids in cache tags. Cache tags store the current channel ids of cached blocks. For upgrade transactions, which change a block state from a shared to a modified state, issuing processors can put bus requests on correct channels directly by using the channel ids. However, for read or read-exclusive requests, issuing processors conservatively send requests to the channel directory, which will queue the requests on correct channels.

Subspace snooping restricts the number of channels snooped by each processor. If a processor needs to snoop a new channel, it should stop snooping one of the current channels. Disconnecting a processor from a channel is a costly operation. The processor should flush any block in its local cache, if the block is mapped

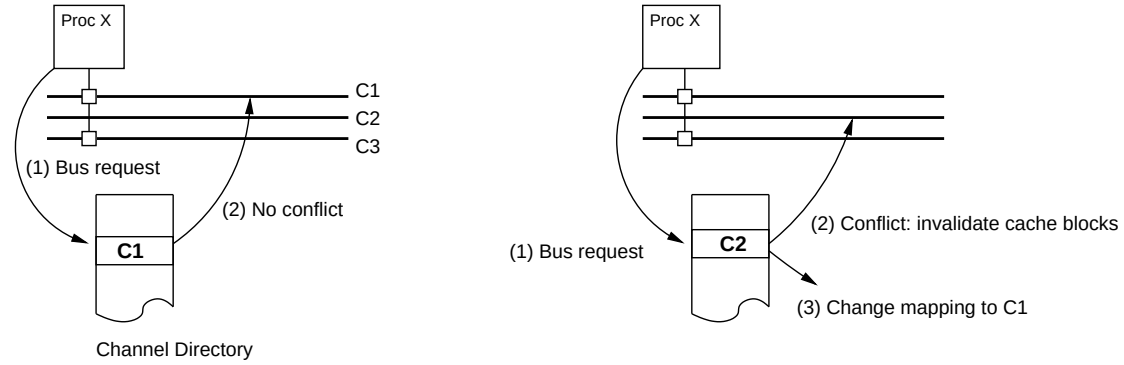

Figure 4: Baseline subspace snooping protocol

to the channel to be disconnected. The channel ids in the cache tags are checked and blocks are flushed if the blocks are mapped to the disconnected channel.

If a block is shared by the processors that do not have a common channel, the block will keep causing mapping conflicts and block invalidations. Such frequently conflicting addresses are moved to the fully-associative channel (FA channel). The FA channel is a safety net to map the conflicting blocks, which otherwise can not be mapped to other channels. However, the fully-associative channel consumes the same snoop tag bandwidth as conventional snooping protocols, and thus its use should be minimized.

3.3 Comparing Subspace Snooping to Directory Protocols

Directory protocols can provide higher scalability than traditional snooping protocols. However, due to the complexity of protocols and the overheads for the directory, the directory protocols have not replaced snooping protocols. Subspace snooping mitigates the snoop tag limitation of snooping protocols, with the simplicity of snooping protocols. In this section, we compare subspace snooping to directory protocols:

- Subspace snooping uses broadcast buses and caches are snooped atomically through the buses. Subspace snooping may avoid the protocol complexity of directory protocols.
- Each cache maintains coherence states in tags. Unlike the directory in directory protocols, the channel directory does not have sharing states, and thus the size of channel directory is much smaller than that of the directory in directory protocols.
- Increasing the granularity of coherence in directory protocols can degrade the performance significantly due to false sharing. Subspace snooping decouples sharing states from channel mapping states. Subspace snooping may increase the channel mapping granularity without incurring false sharing.
- Since subspace mapping changes less frequently than sharing states in directory protocols, predicting the channel id may be more accurate than predicting sharers for directory protocols. Such channel prediction allows two-hop cache-to-cache transfers.

4 Subspace Snooping Protocols

In this section, we present two subspace snooping protocols. Our baseline protocol always sends requests, except upgrade requests, to the channel directory first. After receiving requests, the channel directory for-

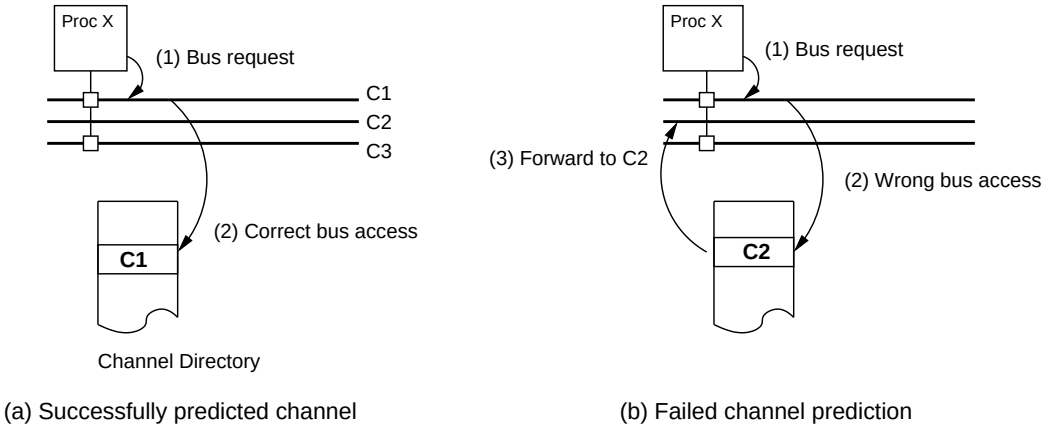

Figure 5: Performance subspace snooping protocol

wards them to correct channels. Our performance protocol predicts channels for missed blocks and broadcasts requests through the predicted channels. The channel directory will correct the speculation of channel ids. In 4.4, we present our policy to form subspaces to reduce conflicts as well as snoop tag lookups.

4.1 Baseline Subspace Snooping Protocol

For read misses, requesting processors do not know the correct channels for the missed blocks. Therefore, bus requests for reads are always forwarded first to the channel directory. For such forwarding, our implementation does not use the address broadcasting bus. Instead, the bus request is sent through a point-to-point data network to the channel directory. The forwarding does not need to use the address bus, since request serialization occurs when the channel directory puts an actual bus transaction on the bus.

To eliminate unnecessary accesses to the channel directory, the current channel ids of blocks are appended to cache tags. For upgrade transactions (changing a block state from shared to modified), requesting processors can put requests on correct channels by using the embedded channel ids.

If a requesting processor is not snooping on the channel the requested block is mapped to, the channel directory starts the conflict resolution procedure. For a mapping conflict, we change the mapping of the conflicting block to one of the channels the requesting processor is snooping. The channel directory adds the new channel id to the bus request when it forwards the request through the old channel. The processors on the old channel should invalidate the block, if they are not snooping the new channel. The mapping conflicts never occur for upgrade requests. For a upgrade transaction, the requesting processor already has the shared copy, so the processor must be snooping the current channel of the block the processor attempts to upgrade. Figure 4 shows the flows of protocol execution for a read request without and with channel conflicts.

If a block causes too many conflicts, we map the block to the fully associative channel. In the channel directory, we record the history of conflicts for each block. If the number of conflicts for a block become larger than the threshold, the block is moved to the fully-associative channel. Note that moving the mapping of an address to the fully-associative channel, does not cause any block invalidation, since all processors snoop the FA channel. However, if too many blocks are mapped to the FA channel, conflicts may decrease, but snoop tag lookups will increase.

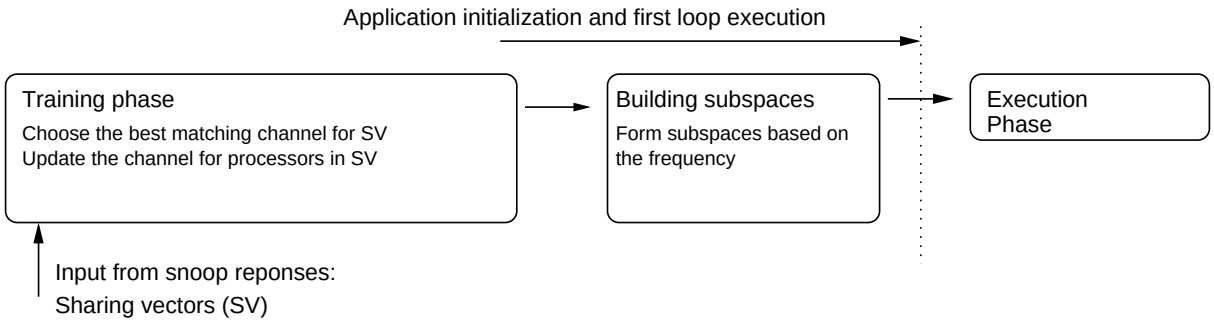

Figure 6: Training and building subspaces

4.2 Performance Subspace Snooping Protocol

The performance protocol improves the baseline protocol by broadcasting requests on predicted channels. The channel directory, which snoops all channels, verifies the predicted channels. If the predicted channel is incorrect, the channel directory initiates the second broadcast on correct channels. If the prediction is correct, the performance protocol reduces the latencies from the baseline protocol. If the prediction is incorrect, unlike the baseline protocol, the performance protocol may cause extra snoop tag lookups at the initially failed broadcasts of requests.

To predict the channel ids for read misses, we use a simple mechanism using invalid blocks. In multiprocessors, many invalidated blocks remain invalid till processors access them again. If a missed block is in the invalid state, we use the channel id in the local cache, and place the request on the predicted channel. However, if the requesting processor does not snoop the predicted channel currently, the request is sent to the channel directory conservatively. Figure 6 shows the flows of protocol execution with speculative bus accesses.

4.3 Subspace Formation

The goals of subspace formation are to map the smallest set of processors on each channel to reduce snoop tag lookups and to minimize conflicts and block invalidations at the same time. However, achieving these two goals is difficult since pursuing one goal may conflict with the other goal. If there are too few processors on channels, common sharing patterns can not be mapped to any channel, and thus are mapped to the fully-associative channel.

In this paper, we use a decoupled approach to build subspaces from training inputs. At the initialization and the first iteration of applications, the system uses a normal address-interleaved snooping protocol. During the training phase, sharing patterns are collected from snoop responses, and fed to the subspace training controller. At the end of the first iteration, subspaces are formed from the trained sharing patterns.

To train common sharing patterns, we use a $N \times M$ matrix T (N = the number of channels, and M = the number of processors). Each row of the matrix T corresponds to a channel, and for each channel (row), we record the number of accesses to processors. For each snoop response, the training controller obtains sharing vector SV (SV[i] = 1, if processor i has a cached block for the snoop, and SV[i] = 0, otherwise). The controller looks for a channel (row) from matrix T, which matches best with the input sharing vector SV. We select a row with the highest score, with a simple matching function, F_{match}:

- $F_{match}(c) \; = \; Sum_{overlapped}(c) \; - \; Sum_{excluded}(c)$
- $Sum_{overlapped}(c) \; = \; \Sigma \, T[c][i], \; if \; SV[i] = 1$

Application	Dataset/parameters
AppBT	36x36x36 grid
Barnes	16K particles
FMM	16K particles
Ocean	258x258 grid
Water nsquared	512 molecules
Water spatial	512 molecules

Table 2: Application parameters for workloads

- $Sum_{excluded}(c) \ = \ \Sigma \, T[c][i], \ if \ SV[i] = 0$

For the best matching channel c_{best}, $T[N][M]$ is updated as follows:

- For each i, $T[c_{best}][i] \ = \ T[c_{best}][i] + 1, \ if \ SV[i] = 1$

Each processor can snoop maximum k channels. Therefore, when subspaces are formed, for each column (processor), k channels are selected with the highest T values. Subspaces are formed by choosing those channels for each processor.

In this paper, we evaluate this decoupled subspace formation, which requires a training phase to gather information of sharing patterns. However, continously updating subpaces is also a feasible policy. We leave such policy as future work.

5 Experimental Results

We ran the subspace snooping evaluation using SimpleMP [24], an execution-driven multiprocessor based on SimpleScalar's out-of-order simulator. The cache hierarchy we modeled consists of separate level-one data and instruction caches, and snooping level-two caches using a 5-state MOESI protocol. The L1 instruction and data caches are 32KB, 4-way associative, with 64B blocks. The unified L2 caches are 8-way associative with 512KB capacity, and also use 64B blocks. Each L1 and L2 cache can have eight pending misses. We simulated a 64-processor system running six scientific benchmarks, five from the SPLASH-2 benchmark suite [25] and AppBT, a parallel shared-memory version of BT in the NAS parallel benchmark suite [4]. Table 2 shows the data set sizes used. We modeled traditional test and test-set locks, and implemented atomic swaps using load-locked and store-conditional instruction pairs.

We simulate subspace snooping protocols with 8, 16, and 32 logical channels (wavelengths) in the optical address bus, reserving one channel to be the fully-associative channel (FA channel). To determine the best number of per-processor snooping channels, that minimizes total snoops, we explored the design space and found that for the 8, 16, and 32 channel configurations, the best number of per-processor channels were 3, 4, and 5, respectively. We assume those numbers for the rest of this section.

We also assume a separate optical data bus. As an optimization, we send memory accesses to private pages–such as stacks and local heaps–directly to memory on the data bus, bypassing the snooping buses. To model the optical latencies, we used estimated latencies from the literatures [14, 12], assuming that an address bus transaction takes 50 cycles from the start of bus access to the completion of the snoop responses. We assume that a transfer of a 64B data block takes 64 cycles, and that sending a bus request to the channel directory takes 22 cycles. We form subspaces after the first iteration of each application's main loop.

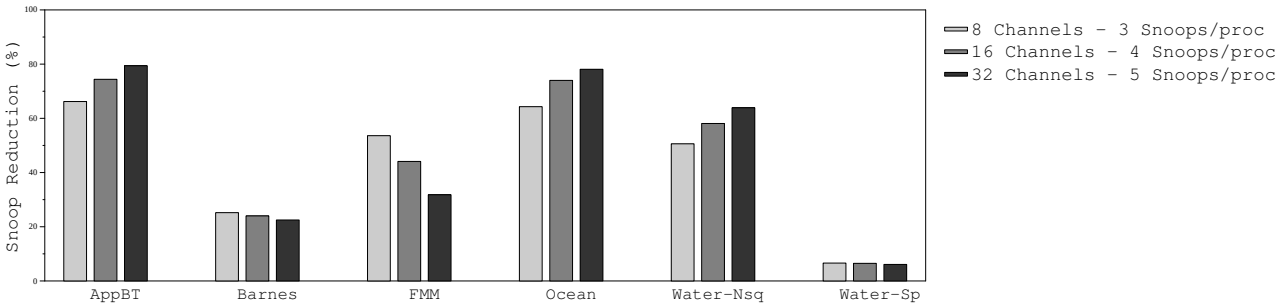

Figure 7: Snoop reduction rates with 8, 16, and 32 channels

Application	sharers < 8	$8 \leq$ sharers < 32	sharers ≥ 32	FA usage
AppBT	91.8%	3.1 %	5.1%	10.1%
Barnes	20.6%	13.6%	65.8%	58.8%
FMM	59.4%	26.7%	13.9%	24.0%
Ocean	95.5%	0.0%	4.5%	10.9%
Water nsquared	68.8%	0.2%	31.0%	29.0%
Water spatial	0.2%	87.7%	12.0%	90.4%

Table 3: Distribution of bus accesses and FA channel usages

5.1 Reducing Snoop Tag Lookups

Figure 7 shows the reduction of snoop tag lookups with 8, 16, and 32 logical channels. The six benchmarks show snoop reductions of 5-80% across different numbers of channels. The most regular applications, AppBT and Ocean, show the largest reductions (60-80%) in snoops. For Barnes and Water-Sp, the snoop tag lookup reductions are quite small (5% and 20%). For these two irregular applications, our channel mapping algorithm could not form stable subspaces effectively for the majority of memory accesses, forcing many or most bus accesses to occur on the fully associative channel.

As the number of logical channels increases, the reductions increase for the three most regular applications (AppBT, Ocean, and Water-Nsq). With more logical channels, the subspace formation algorithm has greater flexibility to form subspaces with smaller numbers of processors. For the three regular applications, the reductions increase by 10-20% from 8 channels to 32 channels. As the bandwidth and the number of wavelengths in optical interconnects increase, subspace snooping will allow a more fine-grained partitioning of processors, reducing more snoop accesses.

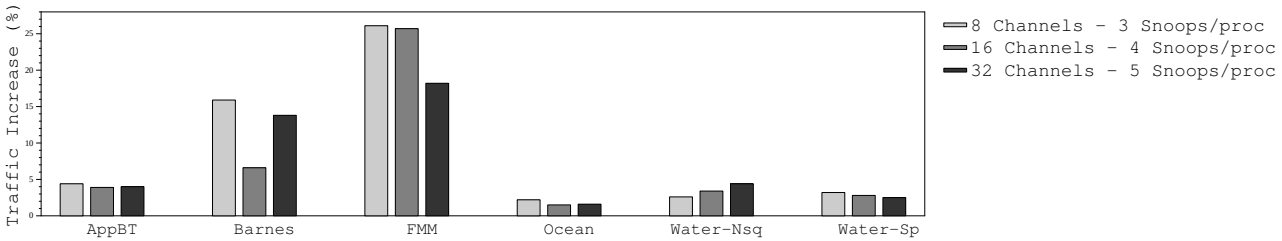

Figure 8: Bus traffic increases for shared data with 8, 16, and 32 channels

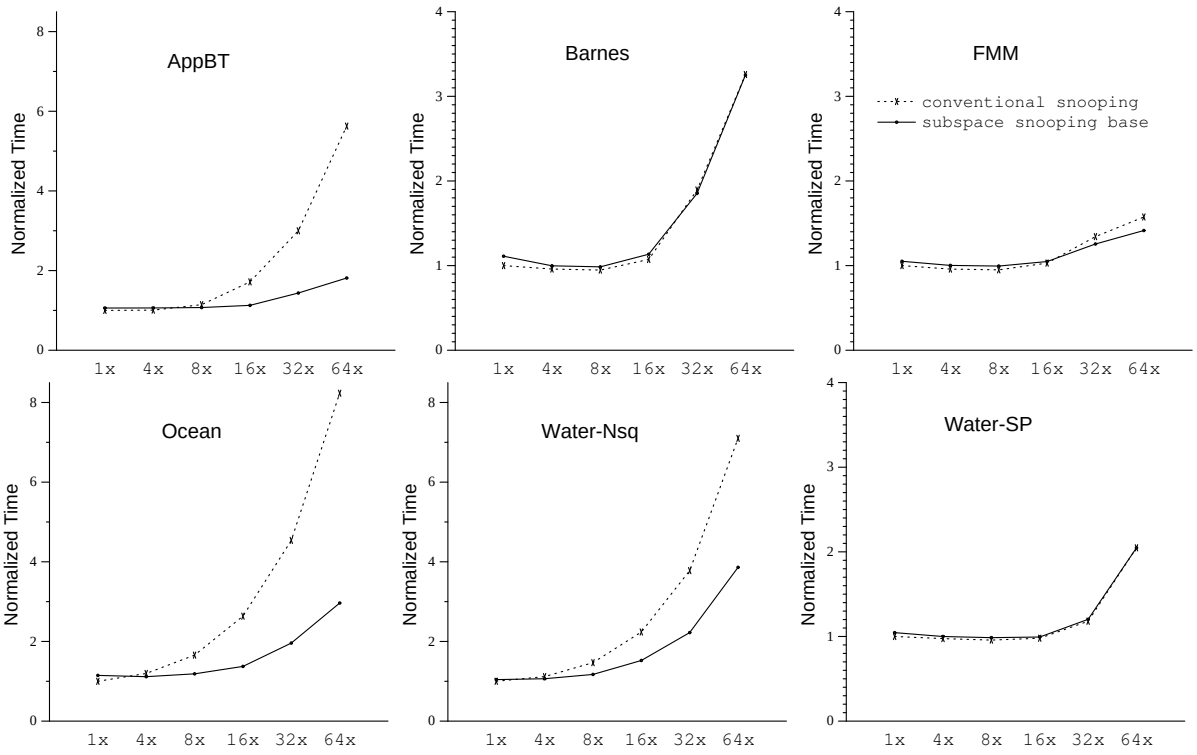

Figure 9: Performance scalability of subspace snooping

Table 3 shows the distribution of bus accesses for different numbers of sharers. For each block address, we record processors which access the block at least once during the execution of applications. Table 3 shows the frequencies of three classes of bus accesses: fewer than 8 sharers, 8 to 32 sharers, and more than 32 sharers. For AppBT and Ocean which showed the best snoop reduction, more than 90% bus accesses are for the addresses with less than eight sharers. Subspace snooping can effectively find subspaces with small numbers of processors for such applications. However, for Barnes and Water-Sp, the majority of accesses are for the addresses with more than 8 sharers. Water-Sp shows distinct patterns with 87% of bus accesses with 8 to 32 sharers. The last column of Table 3 shows the usage of the FA channel in the 16 channel configuration. The FA usage is low for AppBT and Ocean, but in Water-Sp, 90% accesses are sent through the FA channel, since the common sharing patterns with 8-32 processors can not be mapped to any other channels.

The performance disadvantage of subspace snooping is the increase of bus accesses due to channel conflicts. Resolving such conflicts can incur cache misses by invalidating cache blocks to maintain correctness. Such extra misses caused by invalidations will increase if subspace snooping is unable to form a good set of subspaces. Figure 8 shows the increase of total bus accesses due to mapping conflicts. Except Barnes and FMM, four other benchmarks show the low increases of bus accesses, less than 5%. For those applications, the effect of extra bus accesses is small, but FMM has a relatively large increase–up to 25% of bus accesses.

5.2 Performance Scalability

In this section, we reduce effective snoop bandwidth to project how well subspace snooping will scale to larger systems with more processors. Since our simulation infrastructure is unable to model systems with hundreds of processors, due to limited simulation time, memory constraints and application scalability, we

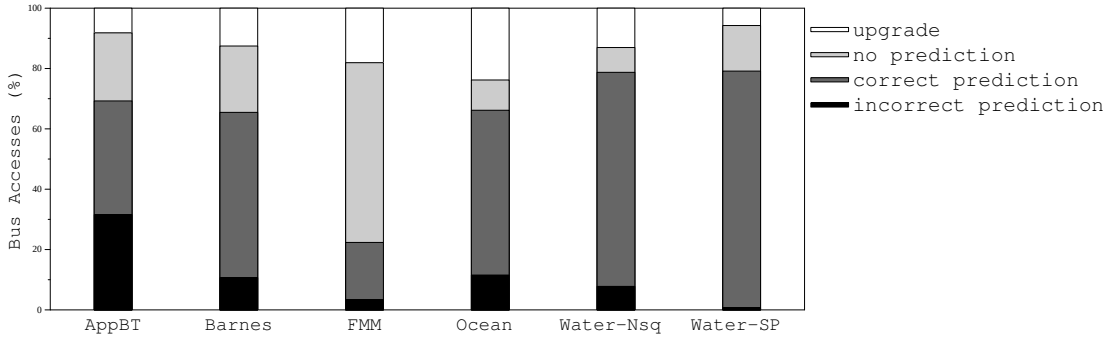

Figure 10: Channel prediction accuracy (16 channels)

used an alternate approximation. Instead of scaling the number of processors, we decreased the available snooping tag bandwidth for a fixed number of processors (64). We assume the number of bus accesses will linearly scale with the number of processors and the consumption of tag bandwidth will increase linearly. We simulate the effect of scaling the system by a factor of x, by reducing snooping tag bandwidth by $1/x$, but leaving address bus bandwidth constant, since we assume that snoop tag bandwidth is decoupled from address bus bandwidth.

Figure 9 presents the scalability results of six applications. The x-axis represents system scaling factors of 1, 4, 8, 16, 32, and 64 in a 64-processor system. In 1x system, each snoop tag bank can process one request per processor cycle, which is highly optimistic; the StarFire bus can process a snoop every three processor cycles, and the snoop bus in the Sun Fireplane interconnect can process a snoop only every 6 processor cycles [6, 5]. Those numbers do not necessarily imply a bound on potential snooping tag speed, since for those systems, snoop tag bandwidth is coupled to bus bandwidth.

For the six applications, the execution times increase by more than twice at 64x scaling with the conventional snooping. For the regular applications (AppBT, Ocean, and Water-Nsq), it appears that subspace snooping can potentially support larger systems effectively, given a sufficiently high-bandwidth optical fabric. At that ratio, the Ocean benchmark incurs a slowdown of eight times with conventional snooping but subspace snooping slows the increase to three times.

5.3 Accuracy of Performance Subspace Snooping Protocols

The performance subspace snooping protocol enhances the baseline protocol by sending bus requests directly through predicted channels. However, incorrect predictions can waste snoop tag bandwidth, since a request must be broadcast twice on two channels in that case. Figure 10 shows the prediction accuracy for bus accesses with 16 channels.

We break down bus accesses to four classes: upgrade, no prediction, correct prediction, and incorrect prediction. For upgrades, requests are always placed on correct channels directly without checking the channel directory. Therefore, upgrades do not use prediction. For read misses, the performance subspace snooping uses channel prediction only when there are invalid cache blocks for missed addresses and processors are snooping predicted channels. Otherwise, requesting processors send requests to the channel directory conservatively, using no prediction.

For these benchmark applications, the ratios of upgrades are 10-25% except for Water-Sp. The performance protocol can predict channels for more than 50% of bus accesses correctly except for the FMM benchmark, which has a large fraction of no channel predictions being made. Across all benchmarks, the incorrect prediction ratios are relatively low (less than 10% except for AppBT).

6 Conclusions

In this paper, we have explored a new class of coherence protocols intended to expand the scalability of snooping-like protocols. If snooping optical interconnects become common, snooping protocols will face a severe snoop tag bandwidth limitation as the system is scaled, as well as an energy consumption limit.

For regular applications, we have shown that subspace snooping increases the effective snoop tag bandwidth by forming subspaces, using a hardware mechanism that recognizes common communication patterns and forms subspaces. We used pair-wise communication statistics between two processors to find subspaces.

The subspace snooping results showed a 5-80% reduction of snoop tag lookups by using 8, 16, and 32 logical channels. Two applications with regular sharing patterns showed 60-80% reduction of snoop tag lookups. Such snoop reduction allowed the systems to scale better than a conventional snooping protocol, so long as the application has sufficient regularity.

While these results show that subspace snooping has potential for large-scale systems running regular, scientific applications, we do not believe that it can be made practical on electrical buses. With optics, however, the tradeoff space is quite different, and new types of coherence protocols may well arise if optical interconnects, particularly snooping ones, become widespread.

We believe that many large-scale applications have stable sharing patterns that can be exploited transparently and much more efficiently, but it may require significantly larger systems (a higher number of processors), with more logical channels and bigger datasets than we can safely simulate. Nevertheless, we have shown that significant reductions in energy and tag contention are possible using subspace snooping, even for these relatively small-scale benchmarks.

References

[1] A. F. Benner, M. Ignatowski, J. A. Kash, D. M. Kuchta, and M. B. Ritter. Exploitation of optical interconnects in future server architectures. *IBM Journal of Research and Development*, (4/5), 2005.

[2] E. E. Bilir, R. M. Dickson, Y. Hu, M. Plakal, D. J. Sorin, M. D. Hill, and D. A. Wood. Multicast snooping: a new coherence method using a multicast address network. In *ISCA '99: Proceedings of the 26th annual international symposium on Computer architecture*, pages 294–304, Washington, DC, USA, 1999. IEEE Computer Society.

[3] C. Brackett. Dense wavelength division multiplexing networks: Principles and applications. *IEEE Journal of Selected Areas in Communications*, (6), Aug. 1990.

[4] D. Burger and S. Mehta. Parallelizing appbt for shared-memory multiprocessors. Technical Report 1308, Computer Sciences Department, University of Wisconsin, September 1995.

[5] A. Charlesworth. Starfire: Extending the smp envelope. *IEEE Micro*, 18(1):39–49, 1998.

[6] A. Charlesworth. The sun fireplane system interconnect. In *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, pages 7–7, New York, NY, USA, 2001. ACM Press.

[7] P. W. Dowd, J. A. Perreault, J. Chu, J. C. Chu, D. C. Hoffmeister, and D. Crouse. LIGHTNING: a scalable dynamically reconfigurable hierarchical WDM network for high-performance clustering. In *Proc. of the Fourth IEEE Int'l Symp. on High Performance Distributed Computing (HPDC-4)*, pages 220–229, 1995.

[8] B. G. Fitch, R. S. Germain, M. Mendell, J. Pitera, M. Pitman, A. Rayshubskiy, Y. Sham, F. Suits, W. Swope, T. J. C. Ward, Y. Zhestkov, and R. Zhou. Blue matter, an application framework for molecular simulation on blue gene. *J. Parallel Distrib. Comput.*, 63(7-8):759–773, 2003.

[9] A. Groot, R. Deri, R. Haigh, F. Patterson, and S. DiJaili. High-performance parallel processors based on star-coupled wdm optical interconnects. In *Proceedings of MPPOI*, 1996.

[10] J.-H. Ha and T. Pinkston. The speed cache coherence protocol for an optical multi-access interconnect architecture. In *Proceedings of the Second Workshop on Massively Parallel Processing Using Optical Interconnections*, 1995.

[11] E. Hagersten and M. Koster. Wildfire: A scalable path for SMPs. In *Proceedings of the 5th International Symposium on High-Performance Computer Architecture*, pages 172–181, Jan. 1999.

[12] A. K. Kodi and A. Louri. Design of a high-speed optical interconnect for scalable shared-memory multiprocessors. *IEEE Micro*, 25(1):41–49, 2005.

[13] B. E. Lemoff, M. E. Ali, G. Panotopoulos, G. M. Flower, B. Madhavan, A. Levi, and D. W. Dolfi. MAUI: Enabling fiber-to-the-process with parallel multiwavelength optical interconnects. *Journal of Lightwave Technology 22*, 2004.

[14] A. Louri and A. K. Kodi. An optical interconnection network and a modified snooping protocol for the design of large-scale symmetric multiprocessors (smps). *IEEE Transactions on Parallel and Distributed Systems*, (12):1093–1104, Dec. 2004.

[15] M. M. K. Martin, P. J. Harper, D. J. Sorin, M. D. Hill, and D. A. Wood. Using destination-set prediction to improve the latency/bandwidth tradeoff in shared memory multiprocessors. In *Proceedings of the 30th Int. Symp. on Computer Architecture*, pages 206–217, June 2003.

[16] M. M. K. Martin, M. D. Hill, and D. A. Wood. Token coherence: decoupling performance and correctness. In *Proceedings of the 30th Int. Symp. on Computer Architecture*, pages 182–193, June 2003.

[17] M. M. K. Martin, D. J. Sorin, M. D. Hill, and D. A. Wood. Bandwidth adaptive snooping. In *Proceedings of the 8th International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2002.

[18] F. Mederer, I. Ecker, J. Joos, M. Kicherer, H. J. Unold, K. J. Ebeling, M. Grabherr, R. Jager, R. King, , and D. Wiedenmann. High performance selectively oxidized VCSELs and arrays for parallel high-speed optical interconnects. *IEEE Transactions on Advanced Packaging*, 24(4):442–429, Nov. 2001.

[19] A. Moshovos. Regionscout: Exploiting coarse grain sharing in snoop-based coherence. In *Proceedings of the 32nd International Symposium on Computer Architecture*, June 2005.

[20] A. Moshovos, G. Memik, B. Falsafi, and A. N. Choudhary. JETTY: Filtering snoops for reduced energy consumption in SMP servers. In *HPCA*, pages 85–96, 2001.

[21] B. Mukherjee. WDM-based local lightwave networks–part i: Singlehop systems. *IEEE Net. Mag.*, May 1992.

[22] O. O. Ogunsola, A. Benner, and J. D. Meindl. A practical symmetric multi-processor architecture design study using optical multi-drop networks. In *Proceedings of the 5th Annual Austin CAS Conference*, Feb. 2004.

[23] R. Patel, S. Bond, M. Pocha, M. Larson, H. Garrett, R. Drayton, H. Petersen, D. Krol, R. Deri, and M. Lowry. Multiwavelength parallel optical interconnects for massively parallel processing. *IEEE Journal of Selected Topics in Quantum Electronics*, (2), 2003.

[24] R. Rajwar and J. R. Goodman. Speculative lock elision: Enabling highly concurrent multithreaded execution. In *Proceedings of the 34th Int. Symp. on Microarchitecture*, pages 294–305, Dec. 2001.

[25] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22th International Symposium on Computer Architecture*, pages 24–36, Santa Margherita Ligure, Italy, 1995.

[26] R. J. W. Y.S. Liu, W. Hennessy, P. Piacente, J. J. Rowlette, M. Kadar-Kallen, J. Stack, Y. Liu, A. Peczalski, A. Nahata, and J. Yardley. Plastic vcsel array packaging and high density polymer waveguides for board and backplane optical interconnect. In *Proceedings of Electronic Components and Technology Conference*, pages 999–1005, 1998.