1

Power Efficient Cache Coherence

Craig Saldanha and Mikko H. Lipasti

Department of Electrical and Computer Engineering
University of Wisconsin-Madison
saldanha@ece.wisc.edu, mikko@ece.wisc.edu

Summary. Snoopy coherence implementations employ various forms of speculation to reduce cache miss latency and improve performance. We study the effects of reduced speculation on both performance and power consumption in a scalable snooping design. We find that significant potential exists for reducing energy consumption by using serial snooping for load misses. We report only a minor 6.25% increase for average cache miss latency for a set of commercial workloads while finding substantial reductions in snoop-related activity. We also compare this implementation against a conventional directory protocol implementation, and find that while a directory protocol effectively reduces power consumption due to message traffic, its overall energy consumption is unlikely to be lower than the serial snooping protocol due to lower performance (longer average load latency) and increased memory and directory references.

1.1 Introduction

In the recent past, researchers in both academia and industry have paid a great deal of attention to power consumption in computing systems [8]. Much of this attention has focused on architectural and circuit techniques for reducing on-chip processor power and energy consumption via techniques such as clock-gating [1], memory subsystem storage structure optimizations, system bus optimizations, and main memory access [5]. Recently, a study by Moshovos et al. examined the potential for filtering remote snoop requests by checking them against a small *Jetty* table to avoid tag lookups and reduce on-chip power consumption induced by remote cache misses [7]. We believe that approaches such as these, as well as many others not mentioned here, will help alleviate power consumption problems in future processor chips.

At the same time, market pressure for improved performance is driving designers to build shared memory systems with a large numbers of processors in them. The complexity and frequency of the processor interconnect that provides cache coherence to the software running on these systems is increasing rapidly, as is the power consumed by the interconnect. Interchip busses

account for as much as 15-20% of total chip power [2]. There are several techniques that target coding and information compression as a means to reduce switching activity and thereby reduce power.

However, given that the energy to send a packet over a processor-to-processor interconnect is a function of the interconnect length, capacitance, and bus frequency, it is constant for a given system and circuit technology. Therefore the issue of power [1] consumption in the interconnect of a multiprocessor system must be dealt with at the architectural level by eliminating the transmission of unnecessary packets. This is the primary focus of our proposed serial snooping technique.

Various forms of speculation are routinely employed to reduce the latency of cache misses and overlap data fetch and transmission latency with checking for cache coherence. This paper presents a case study of a hypothetical shared-memory system that is similar to two recent high- end server systems: the IBM S80 [3] and the SunFire 6800 [10]. We find that opportunities exist for reducing speculation in the cache coherence implementation of such a system while sacrificing very little performance (as measured by effective cache miss latency). The mechanisms we propose reduce the number of address transactions (or snoop commands), data fetches, and data transmissions that occur in the system.

1.2 Snoopy coherence protocols

In this section we explain the principles of snoopy coherence protocols and the architectural trade-offs involved in the transmission of snoop packets and the subsequent tag-array accesses and data fetch and transmission. In a snoopy coherence protocol where the nodes are connected by a shared bus (a single set of wires connecting a number of devices or a network that is logically equivalent) every node can observe all transactions on the bus. Coherence is then maintained by having all the cache controllers *snoop* on the bus and monitor the transactions.

The three distinct stages that occur to satisfy a cache miss are *snooping:*, *data fetch* (from remote node or memory, and *data transmit* (also from remote node or memory). There is an opportunity for speculation at each of the three stages and the degree of speculation at each stage enables an architectural trade-off between performance and power consumption.

Snooping: Snooping protocols broadcast snoop packets to allow all nodes to see the snoop packet at the same time. This helps performance since remote tag lookups occur in parallel. This also means that the requesting node will see only a single tag array access latency while determining which nodes have

[1] Throughout this paper, we use the terms power and energy interchangeably, since we do not vary the time base (i.e. bus frequency) needed to convert from one to the other.

a copy of the requested data. Our simulations for a 4-way SMP with 4-way set associative 8MB L2-caches indicate that 32% of all load miss generated snoops, miss in all remote caches, and an average 57% hit in a single remote cache and only about 3.5% find data in all the other caches. These results differ from those reported by Moshovos et al. [7] due to larger caches and different workloads studied, but nevertheless indicate an opportunity for substantial power savings. Every time a snoop is sent to a node that does not contain the requested data, energy is wasted, both for the tag array access and to transmit the snoop packet across the bus. Thus, from a power saving perspective, a useful alternative would be to serialize the transmission of snoops. That is, begin with the node closest to the requestor, and then propagate the snoop to the next successive node in the path only if previous nodes in the path have failed to satisfy the request. Depending on which node (or memory if all nodes miss) satisfies the request there is the possibility for performance degradation since the requesting node now sees additional latency for each access that occurs serially. The total latency to satisfy the data request is no longer independent of which node will supply the data but is instead a function of how far the supplier of the data is from the requestor. The details of power savings and performance degradation associated with serial snooping are discussed in detail in Section 1.3.4.

Data Fetch: DRAM access latency constitutes the significant portion of total latency to satisfy a load miss from memory. By allowing the memory controller to start its DRAM access before the snoop responses from the remote nodes arrive, some of this latency can be overlapped with the remote node tag-array accesses. Though this is advantageous from the point of view of maximizing performance, it contributes significantly to power consumption, since the power associated with DRAM access can be on the order of 300 mw [5]. This power is wasted every time a load miss is satisfied from one of the remote caches. Hence, accessing DRAM non-speculatively after all the snoop responses have been combined saves the most power.

The speculative fetching of data can also be applied to caches at the remote nodes. There is an opportunity to improve performance by allowing the data array look-up to occur in parallel with the tag array look-up. This allows the data fetch latency to be overlapped with the tag-array access latency allowing the data to be supplied more quickly if there is a hit. Speculative fetching of the data prior to determining a tag array hit or miss can also consume excess energy when a miss occurs. This is nevertheless a viable trade-off when performance is at a premium, as is evident from the fact that speculative data fetching techniques are employed in the IBM S-80 [2][3] and Sun Sunfire6800 citesunfire servers.

Data Transmit: Even with a speculative data fetch in parallel with the tag-array lookup, the requesting node must still tolerate the latency of the combining logic which combines the snoop responses to determine which node will supply the data as well as the latency of the actual transmission of the data from the source node to the requesting node. To hide this latency it is

possible to speculatively transmit the data before the snoop response combining has taken place. We are unaware of a snoopy coherence protocol that speculatively transmits fetched data, but the SGI Origin2000 which implements a directory protocol speculatively transmits data to the requestor if it finds that the directory state of the requested line is exclusive. Therefore, when minimizing the latency to satisfy a load miss is of primary importance, speculative transmission of data can be effective. The cost of doing so is the increased bus power and bandwidth consumption caused by the unnecessary transmission of data packets. For the purpose of our initial evaluation of performance and power we will assume a sufficiently large bus bandwidth so that contention between nodes to transmit data can be ignored.

1.3 Methodology

In this section we will describe the interconnect architecture that will form the basis of the power and performance discussions for our various schemes.

1.3.1 Address and Data Interconnect

For simplicity of discussion and simulation we have modeled a 4-way SMP with a single processor per node. The proposed schemes, however, are easily scalable and can be applied to architectures with multiple processors per node as well as additional nodes. The architecture we are modeling has separate data and address interconnects. We assume that each processor is mounted on a separate board (in practical systems there would be more than one processor per board). These boards are then attached via the address and data interconnects through the backplane.

The address interconnect is based on the SunFire 6800 [10]. The interconnect forms a tree of point-to-point connections and is logically equivalent to a broadcast bus. In order to broadcast a snoop, the snoop packet must travel to the root node before it is reflected down to all of the leaf nodes. Each bus transaction needs to pass through two levels of switches to get from the source node to the destination node. Our system models the memory controller at the root node, which is similar to the IBM S80 design [3] rather than connected to the leaf nodes like the SunFire 6800 [10]. Each link represents the delay to go from one block (either a node or a switch) to another. We assume link delay equal to a single bus cycle of 7ns. We also assume a single bus cycle to transmit a packet across a switch chip. These assumptions are mirror the design assumptions of the SunFire 6800 [10].

The data interconnect also forms a tree of point-to-point links. Each board has a board-level switch that links each processor on board to the backplane switch. The backplane switch connects the individual boards. In our model each board has only a single processor and so a board-level switch may seem unnecessary. However, in an attempt to model a large-scale system we include

a board-level switch in our latency and power calculations since in larger commercial systems there will be more than a single processor per board. Each link or switch adds one 7ns cycle of latency.

1.3.2 Types of Speculation

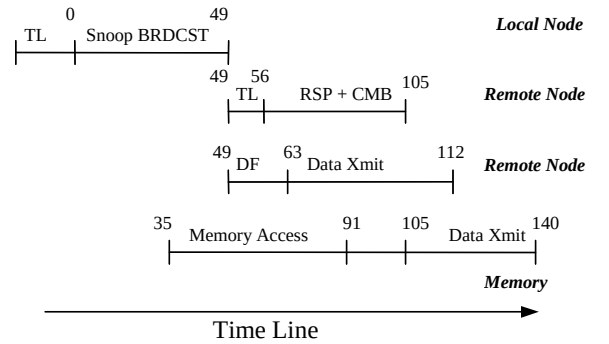
Our discussion on the architectural trade-offs involved in snoopy coherence protocols implies three degrees of freedom in their design: Snooping, data fetch and data transmission. Snooping can be done either serially or in parallel. Parallel snooping is straightforward and simply implies that the snoop packets are broadcast thereby arriving at every node in the system at the same time. In serial snooping, the snoop packet is sent to a single node at a time serially, starting with the node nearest to the requestor and proceeding until the request is satisfied or until all the nodes have been snooped. This is advantageous because the node closest to the requestor supplies the data when available but more importantly power is never wasted, from either speculative tag and data array look-ups or to transmit unnecessary snoop response packets and data. Non-speculative data fetch is done by a node only after the supplier of the data is determined by combining the snoop responses while speculative data fetch involves performing the data array lookup in parallel with the tag-array lookup. Lastly, speculative transmission of data allows the transmission of data to the requestor even before the results of the snoop responses have been determined by the combining network while serial data transmission disallows this. Note that we consider serial snooping only for read operations. Serial snooping of write-related commands has consistency model implications that are beyond the scope of this paper. Serial snooping of reads does not violate the PowerPC consistency model [6].

1.3.3 Parallel Snoop Protocols

We will now present a detailed analysis of several interesting cases; more detailed derivation of our results can be found in a technical report [9].

Parallel Snoop, Speculative Data Fetch, Speculative Data Transmit (PSS-FST). This is the most aggressive implementation of the snoopy coherence protocol. Snoop packets are broadcast to all nodes so that the tag-array lookups for every node occur in parallel. Nodes access their tag and data arrays simultaneously so that in the event of a hit the data is ready for transmission to the requestor. The latencies involved to satisfy a data request that misses in the local cache can be explained with the help of Figure 1.1. The diagram uses a timeline to indicate the latencies involved in completing various operations and also shows the operations that occur serially and in parallel.

To explain the parallel snoop, speculative data fetch and speculative transmit configuration, consider a read by P1 that missed in its local cache and is found in M state in P3. We assume the start of the snoop transaction as time 0, since we are interested in knowing the latency between the time the snoop



0: Data request misses in the local cache and a snoop is initiated
35: Snoop packet reaches memory controller
49: Snoop packet arrives simultaneously at all remote nodes.
Each node begins tag-array look-up and data access in parallel
56:Tag check completed.Snoop response available at remote node.
63:Data fetch complete. Data available in case of a hit
91:Results of combining snoop responses available to memory controller.
105:Results of combining snoop responses available to remote nodes.Memory access also completes
112:Data from remote nodes available to requestor.
140:Data from memory available to requestor.

Fig. 1.1. PSSFST Coherence protocol

is sent out by the requestor and the time when it is satisfied either by a remote node or memory. At time 0 P1 sends its snoop packet out on the address interconnect. Since the interconnect is logically equivalent to a broadcast bus, the snoop request must travel to the root node before being reflected down to all of the branches. The packet passes through 2 switches and 3 links to get to the memory controller, while it must pass through 3 switches and 4 links to get to each remote node. Since each link as well as each switch has a single bus cycle latency, the snoop request is available at the memory controller at 35ns (5 cycles) and at the remote node after 49ns (7cycles).

As soon as the snoop request is available, the memory controller begins the DRAM access, which has a 70 ns latency (we assume a slightly more conservative access latency than [5]). Similarly when the snoop reaches the remote nodes, the tag-array look-up and the data-array access are started simultaneously. We assume a single bus cycle for a tag-look up and a 2-cycle latency for a data- fetch operation to complete. At 56 ns the snoop responses are available at each remote node and they must be sent to the combining logic. The combining of the snoop responses is done at the root node and the process of combining incurs one bus cycle. The combining logic decides which node will supply the requested data or whether it will come from the memory. Since it takes 3 bus cycles to send responses from the remote nodes to the root node and a cycle to perform the combining, the result of the snoops is available at 84ns. They take an additional cycle to be transmitted back to the memory controller and 3 additional cycles to be sent back to the remote

nodes. Therefore after 105 ns the results of the snoops are available at all the nodes. This is similar to the snoop response latency of 100ns reported for the SunFire 6800 [10] which is consistent with the fact that the address interconnect structures in both systems are very similar.

Reviewing our data interconnect, the data needs to traverse 4 links and 3 switches to travel from the source to the requestor. The data transmit is also done speculatively. Hence, 7 cycles later, at 112ns, data from the remote nodes reaches P1. Note that if multiple nodes attempt to transmit data in parallel there will be contention on the bus. For the purpose of this study we are assuming a sufficiently large bus bandwidth so that contention issues can be ignored. Also note that when cache line sizes are larger than the width of the data bus interconnects then multiple data packets must be sent in response to a single snoop request. To simplify our analysis, all our discussions on latency and power account only for the critical packet from a remote node or memory to be transferred to the requestor in order to satisfy the load miss. The remaining data packets will be transferred non-speculatively and though they will contribute to the overall power consumption, their contribution will the same for all of the schemes. Since we are performing a comparative study between different versions of the snoopy protocol rather than trying to estimate absolute values of power, these non-critical words can be excluded without affecting our relative comparisons.

Since the results of the snoop reach P1 at 105ns it knows in advance that it will accept data from P3 and discard data from other nodes. Figure 1.1 shows that if the requested data is present in any of the remote nodes then the snoop request can be satisfied in 112ns. If no remote node has a copy of the data then it takes an additional 28ns to satisfy the request from memory. It is important to note that memory speculatively fetches its data but it is never required to speculatively transmit its data. This is because the results of the snoop are available to the memory controller at 91ns, before the DRAM access completes at 105ns. This scheme offers the best performance but also consumes the most power because of the high degree of speculation involved.

To look at the overall power consumption of this configuration we examine scenarios that will yield the worst case power consumption. The power consumption of the various operations that are performed during a snoop transaction are represented by the following symbols:

- P_{link}*:* Power consumed to send a packet across a link in the address or data interconnect.
- P_{sw}: Power consumed to route packets across a switch
- P_{tag}: Power consumed to do a tag-array lookup
- P_{cache}: Power consumed to fetch a block from cache
- P_{mem}: Power consumed to access DRAM

The power consumption of this configuration is as follows:
If a remote processor node supplies the data,
P_{total}: $32P_{link} + 21P_{sw} + 3P_{tag} + 3P_{cache} + P_{mem}$

If a memory supplies the data,
P_{total}: $23P_{link} + 14P_{sw} + 3P_{tag} + 3P_{cache} + P_{mem}$

Parallel Snoop, Speculative Data Fetch, Non-Speculative Data Transmit (PSSFNT). This configuration differs from the first (PSSFST) in that remote nodes speculatively fetch data in parallel with the tag-lookup but they do not transmit data until the snoop responses have been combined and it is known which node will supply the data. By transmitting data non-speculatively the latency to satisfy a request from a remote node is increased by 42ns but if the request is satisfied from memory there is no performance loss. This is intuitive since the memory controller receives the results of the snoop combining before it completes its DRAM access and therefore it does not have to speculatively transmit data even in the most aggressive configuration (PSSFST). Further details of this configuration can be found in [9].

The power consumption when a remote processor node supplies the data:
P_{total}: $24P_{link} + 15P_{sw} + 3P_{tag} + 3P_{cache} + P_{mem}$
If memory supplies the data:
P_{total}: $23P_{link} + 14P_{sw} + 3P_{tag} + 3P_{cache} + P_{mem}$

Parallel Snoop, Non-Speculative Data Fetch, Non-Speculative Data Transmit. (PSNFNT). This scheme is less aggressive than the previous two schemes since it disables speculative access from memory and data cache. Data fetch occurs only after snoop responses have been combined and the node that will satisfy the request has been identified. The result of the reduced parallelism is an increased latency for both requests satisfied by remote node cache-cache transfers (168ns) as well as those satisfied from memory (196ns). The reduced speculation leads to significant power savings. This is because there is no power wasted by nodes that will not supply data to perform data cache accesses. Further details of this configuration can be found in [9].

The power consumption when a remote processor node supplies the data:
P_{total}: $24P_{link} + 15P_{sw} + 3P_{tag} + P_{cache}$
If a memory supplies the data:
P_{total}: $23P_{link} + 14P_{sw} + 3P_{tag} + P_{mem}$

1.3.4 Serial Snoop Protocols

In all the configurations we have presented so far we have assumed that snoops are broadcast on the address interconnect. With this broadcast technique snoop packets are transmitted on every link since all nodes must see the snoop packet simultaneously. A more *power-aware* methodology for snoopy protocols is serial snooping. The basic idea is to prevent wasting power unnecessarily by transmitting snoop packets to nodes that either do not have a copy of the data or nodes that have a copy but are not responsible for sourcing the data as the result of a snoop.

Serial snooping works by transmitting a snoop packet only to the nearest node. If the nearest node has a valid copy, it sources the data to the requestor and snoop transaction ends without either the memory or any of the other

remote nodes seeing the transaction. On the other hand, if the *nearest neighbor* is unable to satisfy the request, it forwards the request to the next level in the tree hierarchy.

This snooping methodology makes the assumption that the switches in the data interconnect are slightly more intelligent and are able to forward snoops to the appropriate nodes. Note again, that we consider serial snooping only for read operations, which does not violate the rules of the PowerPC consistency model.

There are three serial snooping configurations that are more conservative in terms of speculation but offer significant opportunities for power saving. The configurations are serial snoop/speculative data fetch/speculative transmit (SSSFST), speculative fetch/nonspeculative transmit (SSSFNT, and nonspeculative fetch and transmit (SSNFNT). The following sections discuss the latency and power issues for a snoop initiated by a local miss in P1 and satisfied by P2, P3, P4, and Memory.

Requested data is sourced by P2. The snoop initiated by P1 takes three cycles to traverse two links and a switch to get to P2. The tag access completes and the results are available in the same cycle so that the data access can begin. Hence, in spite of the fact that the tag-check and data access occur serially, they appear to be taking place in parallel. The results of the snoop reach P1 in 49 ns and the data which is non-speculatively fetched and transmitted reaches P1 in 56ns. The snoop never reaches the root node and therefore memory is never accessed. Thus, if a snoop request is satisfied within the same subtree by the nearest neighbor, there is a performance gain as well as power savings.

The power consumption for this configuration is:

P_{total}: $6P_{link} + 3P_{sw} + P_{tag} + P_{cache}$

Requested data is sourced by P3. This example describes the scenario of what happens when P2 is unable to satisfy the request from P1. P2 forwards the request to switch1 which routes it to the root node and from there to memory and back down the tree to P3. P3 receives the snoop 8 bus cycles after it reached P2 which is the latency for P2 to do a look up and re-transmit the snoop. P3 determines that it has a copy of the requested data and transmits the snoop response and the data back to P1 at 140ns. The snoop reaches the memory controller 2 cycles after it reaches the root node (63ns after the transaction began). In this example we have assumed that the memory does a speculative access to avoid the significant latency penalty if the data is not found in any of the caches.

This case obviously expends more power than when P2 sources the data because the snoop request travels to more nodes but it is still significantly more power-efficient than the parallel snoop configurations. The power consumption of this configuration is as follows:

If Memory does not speculatively fetch the data:

P_{total}: $16P_{link} + 10P_{sw} + 2P_{tag} + P_{cache}$

If Memory fetches data speculatively:

P_{total}: $16P_{link} + 10P_{sw} + 2P_{tag} + P_{cache} + P_{mem}$

Requested data is sourced by P4. In this case, the load miss by P1 is satisfied by P4 or memory. Only after P2 and P3 have determined that they do not have a copy of the requested data does the snoop request reach P4. Therefore, 15 cycles (105 ns) after the snoop request originated from P1, P4 performs a tag look-up to determine if it has a copy of the requested data. P4 then transmits the data to P1. Data from P4 arrives at the requestor 168ns after the transaction started. This is the maximum latency to satisfy a load miss from a remote cache. If the snoop request misses in all of the remote nodes then it must be satisfied from memory.

Requested data is sourced from Memory. The latency to satisfy a load miss from memory depends on the degree of speculation used by the memory controller. If the memory controller fetches data speculatively it begins its DRAM access at 63ns even before P3 has determined whether it experienced a hit or a miss. If the memory controller also transmits its data speculatively then the latency to satisfy the load miss is 168ns, which is the same as the latency for data obtained from P4.

The drawback of this scheme is that the power to perform the DRAM access as well as to transmit the data packet on the bus is wasted if either P3 or P4 experiences a hit. If the memory controller only performs a speculative data fetch but does not transmit the data speculatively, no power or bus bandwidth is wasted to transmit unnecessary packets but the load miss is satisfied in 182ns. If the focus of the design were on conserving power then the memory controller would not perform its DRAM access until it has determined that the snoop missed in all 4 remote nodes. Here, the load miss latency is 250ns.

The power consumption for these cases is:

If Memory does not speculatively fetch the data:

P_{total}: $18P_{link} + 11P_{sw} + 3P_{tag} + P_{cache}$

If Memory fetches data speculatively:

P_{total}: $18P_{link} + 11P_{sw} + 3P_{tag} + P_{cache} + P_{mem}$

If Memory fetches and transmits data speculatively:

P_{total}: $21P_{link} + 12P_{sw} + 3P_{tag} + P_{cache} + P_{mem}$

If the snoop misses in all remote nodes and memory supplies the data:

P_{total}: $17P_{link} + 9P_{sw} + 3P_{tag} + P_{mem}$

1.4 Directory Protocols

It is straightforward to see the potential for power-saving with the serial snooping protocol as compared to a more speculative parallel snooping protocol. However, serial snooping can provide a power efficient alternative even to vastly different protocols like a directory protocol. In this section we will present an analysis of a directory protocol, while a comparative study of all

three protocols i.e parallel snooping, serial snooping and directory protocols follows in Section 1.5.

Our analysis of directory protocols is based on the SGI Origin 2000 described in with some additional assumptions to facilitate a comparison with the serial and parallel snooping schemes. We model an interconnect structure identical to the snooping cases. The directory and all of the system memory is located at the root node. This differs from the approach of the SGI Origin 2000, which assumes system memory, and the directory divided amongst the processor nodes. However, in order to maintain consistency across all the schemes discussed in this paper and thereby facilitate comparison between the various protocols we model the directory and memory at the root node. Comparison to systems with distributed memory and alternative interconnects is left to future work. As with the other protocols we only consider read misses; issues and opportunities for handling stores are left to future work.

We will now provide a detailed analysis of the various conditions involved in satisfying a read request with the directory protocol outlined above. Our previous assumptions of a 7ns bus cycle, a 1 bus cycle link traversal and tag lookup latency, a 2 cycle data cache access and 10 cycle memory access latency remain unchanged. Our discussion on performance and power of the directory schemes will follow the same example of a data cache miss by node P1 being satisfied by P2, P3, P4, and Memory.

Request satisfied by Memory (Shared/Unowned). As with other schemes we measure latency from the time P1 misses in its local cache and initiates a network transaction to satisfy its request for data. In this case we will assume that the requested data is found in Unowned or Shared state. In either case, the request will be satisfied by memory at the root node with the same latency. It takes 35ns (5 cycles) for P1s request to reach the root node as it traverses 3 links and 2 switches. At the root node, a directory lookup and a memory access are initiated simultaneously. Each of these completes in 10 cycles and therefore the state of the line and the data from memory are available at 105ns. Additional power savings would be possible by serializing the directory lookup and memory fetch, and avoiding the latter when it is not necessary. However, this would cause a dramatic increase in average load latency and hence, we do not consider it further.

Since the line is in Shared or Unknown state, the home node knows that it has the most recent copy of the data and is therefore responsible for sourcing this data to the requestor. It takes a further 5 cycles for this data to be transmitted back to the requestor. Hence the data request from P1 is satisfied in 140 ns when the line is in Shared or Unknown state.

The power consumed by this case is:

P_{total}: $6P_{link}$ + $4P_{sw}$ + PDir+ P_{mem}

Request Satisfied by P2 (Exclusive State). Now, we assume that the data that missed in P1's local cache resides in the local cache of P2. P1 initiates a network transaction to the home node. The request arrives at the home node after a 35ns (5-cycle) latency. The home node initiates a directory look

up to determine the state of the line and its owner, and also speculatively accesses memory. Both these events complete after 10 cycles at 105 ns. The home node determines that the requested line is in Exclusive state and resides at node P2. At this time it is unknown whether P2 has the line in Clean or Dirty Exclusive state, so the home node speculatively transmits the data to the requestor. It also forwards the request to node P2. The data reaches P1 at the same time that the request reaches P2 after traversing 3 links and 2 switches in 35 ns. P2 completes its Tag lookup after 1 cycle or at time 147ns after the transaction began at P1. If the line is Clean Exclusive, then P1 has the most recent copy of the data from Memory and P2 need not wait for the data cache fetch to complete. It transmits a response to P1, which reaches P1 after 21 ns traversing 2 links and a switch. If the line is Dirty Exclusive then P2 has the most recent copy of the data. It must wait until the Data Fetch completes at 154 ns and then forward this data to P1 and to Memory. In this case P1's miss is satisfied a cycle later at 175 ns.

The power consumed by this configuration is:

If requested Data is Clean Exclusive:

P_{total}: $11P_{link} + 7P_{sw} + PDir + P_{mem} + P_{tag} + P_{cache}$

If requested Data is Dirty Exclusive:

P_{total}: $13P_{link} + 8P_{sw} + PDir + P_{mem} + P_{tag} + P_{cache}$

Request Satisfied by P3. In this analysis we assume that P1's miss can be satisfied by P3 (or P4, since the cases are equivalent), which has the line in Clean or Dirty Exclusive state. As before, P1's request for data reaches the home node at 35ns. The home node completes its directory lookup and speculatively transmits data back to P1 at 140ns. It also forwards the request to the current owner, which we assume is P3 in this analysis. P3 does a local tag array lookup and a speculative data fetch.

The tag lookup completes at 147 ns and if the requested line is found in clean exclusive state then P3 may transmit this data immediately to P1 without waiting for the data fetch operation to complete. Data from P3 must traverse 4 links and 3 switches en route to P1 and hence arrives at the requestor after a 7-cycle (49 ns) delay. If the requested line is found in Dirty Exclusive state then P3 must wait until 154ns for the data fetch to complete and then forward this data to P1 and memory. In this case P1s request is finally satisfied 203ns after it missed locally.

The power consumed by this configuration is:

If requested Data is Clean Exclusive:

P_{total}: $13P_{link} + 9P_{sw} + PDir + P_{mem} + P_{tag} + P_{cache}$

If requested Data is Dirty Exclusive:

P_{total}: $14P_{link} + 9P_{sw} + PDir + P_{mem} + P_{tag} + P_{cache}$

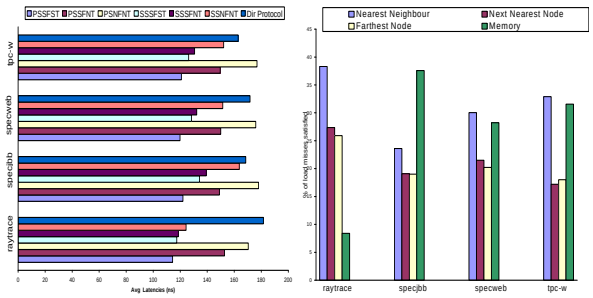


Fig. 1.2. Average Load Miss Latency and Load Miss Distribution

1.5 Simulation Results

We use an augmented version of the SimOS-PPC [4] full system simulator to collect statistics on load misses. We studied the behavior of load misses in four benchmarks: raytrace from the Splash-2 Benchmark suite, specweb99, specjbb2000 and tpc-w on a 4-way SMP with a 4-way set associative 8MB L2 cache with 128 byte lines.

Figure 1.2 shows a plot of average latencies to satisfy a load miss for the 7 configurations described in the paper starting with the most aggressive parallel snooping technique (PSSFST) and progressing through to the most conservative serial snooping technique (SSNFNT) along with the directory protocol for comparison. Figure 1.2 shows that directory protocol performs poorly as compared to the snooping techniques for read misses because of the directory lookup latency. Within the serial snooping techniques, the most aggressive configuration (SSSFST) has the lowest latency to satisfy a load miss but the most conservative configuration (SSNFNT) does not have the worst performance. This is because the effectiveness of the serial snoop depends upon how many times a load miss can be satisfied by its nearest neighbor. When this is the case the latency to satisfy the load miss is 56ns as compared to 112ns in the most aggressive case (PSSFST) and 168ns in the most conservative case with parallel snoop (PSNFNT). Even when the snoop request is satisfied by the next best node using the serial snooping technique, the latency to satisfy a node miss is 140ns which is still less than the latencies for both parallel snoop cases with less than maximum speculation (i.e. PSSFNT and PSNFNT).

The latencies of serial snoop configurations depend on the location where the load miss is satisfied. Figure 1.2 shows that on average 31% of load misses are satisfied by the node nearest the requestor, 21% are satisfied by the next nearest node, 20% are satisfied by the farthest node, and 26% of all load misses are satisfied by memory. Figure 1.2 gives a clear indication that serial snoop performs worse than only the most speculative configuration and the latency penalty is on average 6.25% with the best case being only a 2.6% latency increase in raytrace. The performance penalty for the most conserva-

tive configuration (SSNFNT), which would yield maximum power savings, is on average 23% and in the best case 8.7% (also in raytrace). This indicates that serial snooping configurations provide opportunities for power savings and still perform better than some parallel snoop configurations.

It is intuitive that the power savings will increase as the degree of speculation is reduced. We quantify the power savings in terms of the reduction in activity; activity is represented by symbolic terms that correspond to the different types of activities that are included in the equations presented in Section 1.3. The power consumed for each of the seven configurations is based on statistics from our execution-driven simulation and is shown as a weighted sum of each of the different types of activities. The weights are determined according to the load miss distributions presented in Figure 1.2. To quantify power savings we compare each of the proposed configurations with the most speculative configuration, which consumes the most power. Table 1.1 summarizes the total power for each of the six cases as well as the savings relative to the baseline case (PSSFST):

Table 1.1. Weighted Average Power Consumption $P_{total}(P_{save})$

Configuration	P_{link}	P_{sw}	P_{tag}	P_{cache}	P_{mem}
PSSFST	46.8	19.2	3	3	1
PSSFNT	23.75 (23.05)	14.75 (4.5)	3	3	1
PSNFNT	23.75 (23.05)	14.75 (4.5)	3	0.736 (2.24)	0.264 (0.736)
SSSFST	14.2 (32.6)	7.9 (11.3)	2.16 (0.84)	0.74 (2.26)	0.69 (0.31)
SSSFNT	13.43 (33.37)	7.76 (11.44)	2.16 (0.83)	0.74 (2.26)	0.69 (0.31)
SSNFNT	13.43 (33.37)	7.76 (11.44)	2.16 (0.83)	0.74 (2.26)	0.26 (0.74)
Directory	10.47 (36.33)	6.73 (12.47)	1 (2)	1 (2)	$1+1P_{dir}$ ($-1P_{dir}$)

The relative power consumption due to P_{link}, P_{sw}, P_{tag} and P_{cache} decrease significantly as the degree of speculation decreases from parallel snooping configurations to serial snooping configurations. Table reftable1 shows low weights on P_{link}, P_{sw}, P_{tag} and P_{cache} activities in the directory protocol. These weights are even less than our most efficient serial snooping technique. However in the directory protocol these low activities are negated by the constant power consumption associated with the directory and memory look ups, which have a higher power cost and thereby outweigh the potential savings. In the serial snooping technique it is worthwhile to note the opportunity for power savings achieved by checking the nearest neighbor before forwarding a request to memory as is evident by the drop in P_{mem}. It is clear that maximum power savings are achieved with no speculation in snooping, data fetch and data transmit. However, it is more interesting to note that these savings are only slightly more than the savings obtained by using serial snooping with full speculation for memory. This technique is a clear winner with substantial power savings and minimal performance degradation.

1.6 Conclusion

The use of speculation to reduce latency is an important architectural consideration while designing coherency protocols for modern SMP systems. We have conducted a preliminary performance and power analysis for varying degrees of speculation in a scalable snooping protocol modeled after the IBM S80 and SunFire 6800 systems. We conclude that there is significant potential for power savings without severe performance degradation by reducing the degree of speculation in certain operations. Specifically, we find that employing serial snooping for read commands with speculative data fetch and transmit from memory provides substantial reduction in power consumption without significant performance overhead (only 6.25% latency increase) over both speculative snooping and directory protocol implementations.

We wish to thank IBM and Intel for their generous equipment and financial donations that have enabled much of this work. Finally, this work was supported in part by funding from the National Science Foundation under grants CCR-0073440, CCR-0083126, CCR-0133437, and EIA-0103670.

References

1. D. Albonesi. Dynamic IPC/clock rate optimization. In *Proceedings of ISCA-25*, pages 282–292, June 1998.
2. Jeff Brown. Personal communication, March 2001.
3. The RS/6000 Enterprise Server Model S80 Technology and Architecture. Technical White Paper. www.austin.ibm.com/resource/technology/s80techarch.html, 1999.
4. Tom Keller, Ann Marie Maynard, Rick Simpson, and Pat Bohrer. Simos-ppc full system simulator. http://www.research.ibm.com/arl/projects/simosPPC.html.
5. A. Lebeck, X. Fan, H. Jeng, and C. Ellis. Power aware page allocation. In *Proc. Ninth Intl. Conference on Architecture Support for Programming Languages and Operating Systems*, November 2000.
6. C. May, E. Silha, R. Simpson, and H. Warren. *The PowerPC Architecture: A Specification for a new family of RISC processors. Second Edition.* Morgan Kaufmann Publishers, Inc.
7. A. Moshovos, B. Falsafi, and A. Choudhary. JETTY: Filtering snoops for reduced energy consumption in smp servers. In *Proceedings of the 7th International Symposium on High- Performance Computer Architecture*, January 2001.
8. T. Mudge. Power: A first class design constraint for future architectures. *Computer*, 34(4):52–57, April 2001.
9. Craig Saldanha and Mikko H. Lipasti. Power efficient cache coherence. Technical report, Department of Electrical and Computer Engineering, University of Wisconsin-Madison, December 2002.
10. Sunfire 3800-6800 servers-computing the net effect. Technical Whitepaper. Available from www.sun.com/servers/white-papers, 2001.