WildFire: A Scalable Path for SMPs

Erik Hagersten and Michael Koster
Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303

Abstract

Researchers have searched for scalable alternatives to the symmetric multiprocessor (SMP) architecture since it was first introduced in 1982. This paper introduces an alternative view of the relationship between scalable technologies and SMPs. Instead of replacing large SMPs with scalable technology, we propose new scalable techniques that allow large SMPs to be tied together efficiently, while maintaining the compatibility with, and performance characteristics of, an SMP. The trade-offs of such an architecture differ from those of traditional, scalable, Non-Uniform Memory Architecture (cc-NUMA) approaches.

WildFire is a distributed shared-memory (DSM) prototype implementation based on large SMPs. It relies on two techniques for creating application-transparent locality: Coherent Memory Replication (CMR), which is a variation of Simple COMA/Reactive NUMA, and Hierarchical Affinity Scheduling (HAS). These two optimizations create extra node locality, which blurs the node boundaries to an application such that SMP-like performance can be achieved with no NUMA-specific optimizations.

We present a performance study of a large OLTP benchmark running on DSMs built from various-sized nodes and with varying amounts of application-transparent locality. WildFire's measured performance is shown to be more than two times that of an unoptimized NUMA implementation built from small nodes and within 13% of the performance of the ideal implementation: a large SMP with the same access time to its entire shared memory as the local memory access time of WildFire.

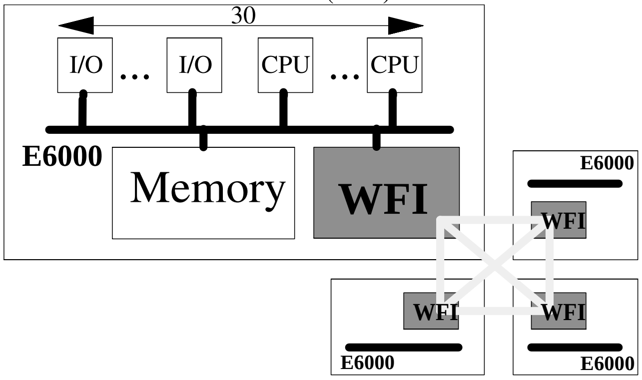
1. Introduction

There has been a concentrated effort in academia to find a scalable replacement for the SMP architecture. There still seems to be a wide-spread belief that SMPs will not scale over time. Many, including the authors of this paper, have claimed for a decade that the bandwidth limitations of the SMP "backplane" would eventually prevent the implementation of yet another generation of scalable SMPs. When we wrote this in 1988, the backplane of the state-of-the-art Sequent Symmetry had an effective bandwidth of about 40 MB/s. Today, some 10 years later, the passive backplane has been replaced by an active switch in Sun's E10000 "Starfire" capable of 12.5 GB/s [4]. SMP bandwidth has improved by roughly a factor of 300 over a time period of 10 years. This is faster than doubling every 18 months, as predicted by Moore's law for CPU development, which yields a factor of 128 improvement over the same time period. Thus, SMP bandwidth has scaled faster than Moore's law over the past 10 years.

Meanwhile, some companies have abandoned their SMP product lines and instead offer implementations of cc-NUMA architectures. While cc-NUMA architectures have the potential for greater scalability, they are less optimal for access patterns caused by "real" communication, such as producer-consumer and migratory data [1]. They also require substantial application and operating-system optimization in order to handle capacity and conflict misses well. Nor is the scheduling algorithm for cc-NUMA trivial. A process migrated to a different node may perform much worse than if it had stayed in the same node, because of a higher ratio of remote traffic.

SMPs provide a simpler model than NUMA for several architectural reasons. The SMP's uniform access time to shared memory provides a simple programming and performance model. An SMP does not require data and code to be placed in any special way for the application to run well. Popular code and data structures are easily shared by all the CPUs. This simplifies algorithms for managing resources, such as memory, processors and I/O devices. A suspended process may be re-scheduled on any other processor at a relatively small cost, even though running it on the CPU where it last ran has an advantage (affinity), leveraging its hot cache. Managing the memory is also easier; any free physical memory can be utilized when a page gets paged in. Non-uniform memory makes all these tasks more difficult. SMPs are also more efficient in handling communication misses (coherence misses), which are common in

Figure 1: WildFire connects up to four E6000 by inserting one WildFire Interface Board (WFI) in each node.



commercial applications [1, 3]. The SMP's current implementation style, often based on some kind of broadcast interconnect, is best served when fit into one cabinet. Physical constraints on the size of the cabinet, such as the size of an elevator or the cargo hold of an airplane, put an upper bound on the scalability of SMPs. Today, there are several SMP implementations scaling between 16 and 30 CPUs [2, 17] and one that scales all the way to 64 CPUs [4]. Our experience and several benchmark world records show these systems' bandwidths to be more than sufficient for the most important and fairly bandwidth-hungry commercial applications. Actually, published benchmarks for commercial workloads show that SMPs often scale better than cc-NUMAs [18].

SMPs are definitely alive and thriving. There is no apparent reason why they will suddenly disappear in a few years.

2. Scalability goal of MSMP

If we accept the fact that SMPs remain one of the primary alternatives for commercial server systems up to a certain scale for many years to come, the question of how to build "scalable" systems needs to be reformulated. The SMP's major limitation is not its viability, but rather that it is difficult to build an SMP with a huge number of CPUs spanning several physical boxes. The question is not how to replace SMPs with a new technology, such as cc-NUMA, but rather how to create a technology that allows high-end SMPs to be part of a scalable family of products providing growth beyond the box limit. We would simply like to ride the SMP curve for as long as it is technically and economically feasible and extend the SMP with a scalable technology such that SMP applications would also be able to run on configurations larger than a single SMP. We call such a scalable technology multiple SMP (MSMP).

The rules for designing an MSMP differ somewhat from other scalable systems. The MSMP should coexist with the SMP, which can be expected to account for far more revenue than the MSMP. The MSMP must, therefore, impose minimal additional complexity and cost on the SMP. Since running on an MSMP should have no impact on the application, it must run the same operating system as the SMP and cannot expect substantial OS modifications to accommodate its special needs. These constraints can often necessitate less optimal DSM solutions than traditional cc-NUMA implementations and add to the remote latency.

There are also several advantages to using large SMP nodes in an MSMP architecture. Large nodes reduce the number of nodes in a large-scale configuration which allows for simple, nonscalable approaches in the cache-coherence protocol. The DSM protocol only needs to keep a handful of nodes coherent and can avoid scalable and complicated solutions, such as SCI's linked lists. The complexity and latency of the interconnect is also reduced by a smaller number of nodes. Furthermore, each large node contains more memory banks, thereby allowing a higher degree of memory interleaving within a node. Large nodes also have a positive impact on node locality. Having fewer nodes implies that a larger fraction of random accesses will be local. Large popular data structures can also be more cheaply replicated in all the nodes of a system built from few nodes. Further, a node-aware load balancer is more likely to find an idle local processor if the nodes are large.

3. WildFire system overview

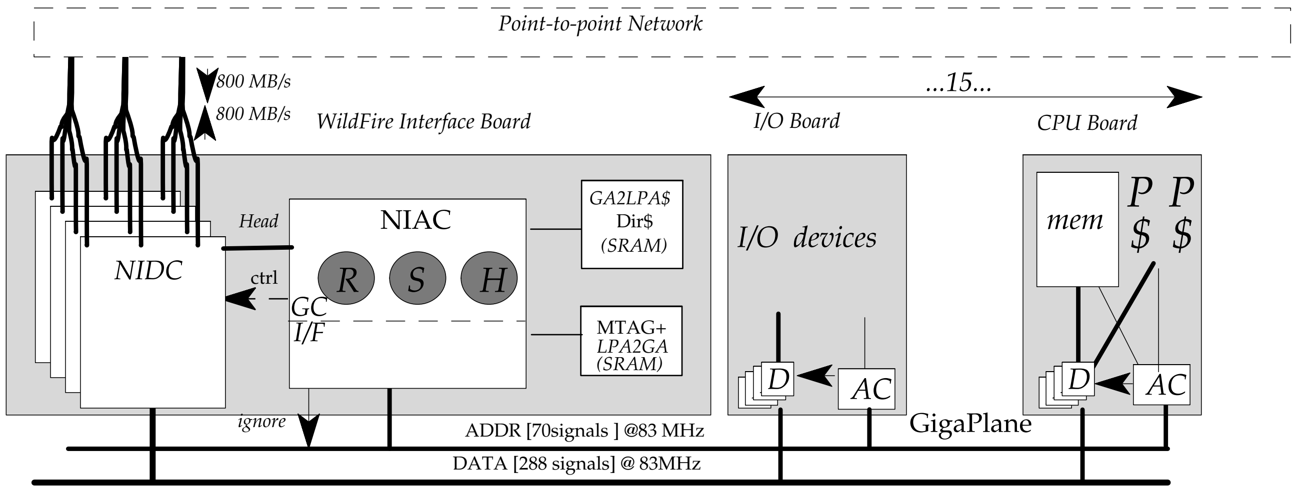
WildFire is an internal code name for a prototype shared-memory multiprocessor developed by Sun Microsystems. WildFire supports up to 112 UltraSPARC I or II processors, runs a slightly modified version of Solaris 2.6, and is 100% application-binary-interface (ABI) compatible with Sun's SMP multiprocessors. WildFire first booted in February 1997. This paper has been edited using a WildFire running Solaris 2.6. WildFire connects two to four unmodified Sun Enterprise E6500/E5500/E4500/E3500™ SMP servers. Supporting the entire SMP family can allow for flexibility in choosing the system's node size. The rest of this paper assumes nodes built from the largest SMP member, E6500, supporting up to 30 CPUs.

Each E6500 has a GigaPlane™ bus connecting up to 16 dual-processor or I/O boards. Boards are interchangeable so a system with minimal I/O (one I/O board) can have up to 30 processors (15 dual CPU boards). GigaPlane supports 50 M transactions/sec, 112 outstanding transactions, a peak data bandwidth of 3.2 GBytes/s, and an lmbench latency today of 252 ns to the entire shared memory [17, 15]. A WildFire Interface (WFI) board replaces a dual-processor or I/O board. Up to four E6500 nodes can be connected through their WFI board. WildFire supports full cache coherence and Total Store Order (TSO), like other Sun systems. By default, WildFire is a "Cache-Coherent Non-Uniform Memory Access" machine (cc-NUMA) built from unusually large nodes.

Wildfire appears as a single system to most layers of the operating system. Remote program-controlled I/O and DMA are transparently handled by the WFI; a process need not know whether an I/O device is connected to its local SMP node or a remote SMP node. Inter-node interrupts are handled uniquely on WildFire, but differences are invisible to processes and drivers. Only the low-level machine-specific layers of Solaris need know of Wildfire's hierarchical structure. Memory allocation is also segregated by node. Wherever possible, local memory is used to satisfy process memory allocation requests.

Shared memory across the system is supported both through multithreading of individual processes and explicit sharing of memory between processes. Wherever possible, threads of a multithreaded process are kept together on the same node. Only when a process has more threads than there are processors on a node does the process begin to span multiple nodes, causing process memory to be shared across the WildFire interconnect.

To reduce remote-memory traffic and improve average memory latency, WildFire also supports "Coherent



Memory Replication" (CMR). CMR is a version of a "Simple Cache-Only Memory Architecture" (S-COMA) [8], but cannot be called a true COMA since the coherence protocol assumes a fixed home location for each address. Specifically, CMR allows an SMP to allocate local "shadow" physical pages to correspond to remote physical pages. The operating system allocates and reclaims pages for replicated data. Coherence, however, is maintained by hardware at the 64-byte block level.

To avoid performance problems related to memory pressure [16], WildFire can switch between cc-NUMA and CMR at page-by-page and node-by-node granularity. All new pages are created as cc-NUMA pages. The Solaris operating system makes use of integrated hardware counters to determine which pages to switch from cc-NUMA to CMR. The selection is done using a variant of the Reactive NUMA (R-NUMA) algorithm [6]. This adaptive algorithm responds to memory-access patterns in order to dynamically decide when CMR can improve performance over cc-NUMA, and vice versa. Early evaluations showed this to be more useful for long-running commercial applications than starting all pages in CMR and converting some of them to cc-NUMA, as proposed for the PRISM architecture [5] based on SPLASH simulations.

WildFire's has a fairly conservative replication strategy in order to avoid situations where its associated overhead would increase the execution time rather than help. Excessive communication to a remote page will initially result in page migration. Page replication is only used for pages for which the migration does not help. The amount of local memory used for replication is dynamically adjusted and depends on the current memory pressure.

The Solaris 2.6 port that runs on Wildfire does include a number of changes to optimize performance. A hierarchical affinity scheduler tries to schedule a process first on the processor it last ran, then on some processor on the same node. Only when load imbalance exceeds a specified threshold is the process scheduled on a remote node. This scheduler increases the time each process spends in a node, increasing the benefit from the state built up in the large CMR memory. The first version of the WildFire operating system has some limitations in its CMR algorithm. Memory-resident pages and "large" physical pages cannot be replicated. These types of pages may still be "explicitly" replicated when created.

4. WildFire implementation

WildFire's interface is divided into two different ASICs: the Network Interface Address Controller (NIAC), which implements the coherence protocol, and the bit-sliced Network Interface Data Controller (NIDC), which provides a fat connection to/from the interconnect, as shown in the figure above. The four NIDC chips are controlled by the NIAC chip. Each WFI board exports three high-speed links of 800 MB/s in each direction, allowing construction of a four-node system without introducing the extra cost and latency of a switched network. Each link is bit-sliced and connected to all four NIDC chips, providing a fat and fast connection for the data transactions between the node's data bus and the interconnect. Interfacing a large SMP node puts a higher bandwidth demand on the coherence interface than a traditional DSM implementation; this is what prompted our bit-sliced solution. The smaller address transactions and the header part of a data transaction are detoured through the NIAC and its coherence protocol.

NIAC functionality is divided into two parts: the bus interface (I/F) and the global-coherence layer (GC). I/F acts as a proxy in the node's SMP protocol. It detects transactions which need attention from the global coherence protocol and asserts an "ignore" signal for those transactions. This signal, which effectively removes the transaction from the local snoop order, is one of the few hooks included in E6500's protocol in order to allow for MSMP implementations. Adding the WFI board slows down the access time to local memory by up to two cycles compared to the E6500, since the WFI board does not support the "fast arbitration mode" of the E6500 [17].

WildFire supports a global physical address space where higher-order address bits determine the node on which home memory resides. Remote cc-NUMA requests appear as GigaPlane transactions to memory physically residing on another SMP node; the I/F detects this by checking the higher-order bits. The memory tag (MTAG) data structure contains 2 bits of MOSI state per 64-byte block in the local physical memory. The I/F performs a lookup in the MTAG for each local access to the node's memory. If the state is inadequate (e.g., a "read exclusive" to a "shared" block), WFI asserts the "ignore" signal and invokes the global coherence layer (GC). Otherwise, the transaction proceeds as in a single SMP.

The global cache-coherence protocol is implemented in the GC layer of the NIAC, similarly to the Dash implementation [13]. Three protocol agents (Request, Home, and Slave) exchange messages over the global interconnect. The GC protocol is efficiently implemented directly in hardware and supports up to 40 simultaneous ongoing transactions in each WFI. A directory cache tracks the necessary coherence-directory state needed by the GC protocol, similarly to the FLASH prototype [9]. The SMP's memory is used to back the directory cache.

Part of the SMP's memory can be used to cache remote data by using the CMR technique. A remote page (backed by local CMR memory) is referenced on the local bus by using the CMR page's local physical address. Upon an access miss to the CMR page (insufficient MTAG state), the ignore signal is asserted and the transaction removed from the node's snoop order. The local physical address needs to be translated to the corresponding global address of the remote page before a remote request can be sent by the local WFI. To support CMR, WFI provides a data structure to translate between local physical CMR addresses and the global (remote) physical addresses in the home node. We call this translation Local Physical Address to Global Address, LPA2GA. The second address translation needed to support CMR, the reverse address translation (GA2LPA), is stored in an SRAM cache backed by memory. Unlike traditional S-COMA, the GA2LPA translation is only needed by the slave agents. The greater part of the extra hardware needed to implement CMR comes from these two address translation tables. All other hardware support needed to implement CMR is negligible.

The WFI has associative counters to monitor capacity and conflict behavior of accesses to remote pages and provide input to the operating-system policy deciding which pages should use CMR. This policy is similar to R-NUMA [6]. Software initializes a counter as "free." The first remote access which is identified as a capacity or conflict miss, a so-called excess misses (E-miss), initializes the address part of the counter with its page number. All subsequent E-misses to the same page will cause the counter to increment. Software periodically monitors and frees these counters. Pages showing an appropriate sharing pattern for a long duration will get replicated. This fairly conservative way of choosing CMR pages keeps the software overhead associated with setting up coherently shared pages low. WildFire's software also supports interfaces for placement and replication of pages under user control.

The "scalability" requirements for MSMP architectures are quite different from other DSM systems. Here, only a handful of nodes need be connected and simplicity is the guiding principle. Scalable directory approaches, such as linked lists, are not needed. WildFire uses a fairly traditional MOSI write-invalidate coherence protocol with a full-mapped directory representing each node in the system with one bit, and two bits identifying one node as the owner. This allows for very compact representation in the directory cache. WildFire implements a three-hop protocol to minimize latency to remote dirty data in nodes other than the home node or request node. The extra throughput requirement prompted by the large nodes precluded us from implementing a programmable protocol [9]. Instead, we designed a simple coherence protocol with no corner cases. This allowed for a fairly straightforward verification strategy and gave us the confidence to implement the protocol directly in hardware. The protocol was bug-free in first silicon. We call the approach a "deterministic directory." Unlike most other coherence-directory protocols, this protocol's directory state and the state of the caches are always in agreement. This is achieved by two separate features: the blocking directory and three-phase writebacks. The blocking directory only allows for one outstanding transaction per cache line; in other words, the protocol guarantees that all previous read requests to the same cacheline have been completed before new requests to the cache line are serviced. Nor are writebacks started until all previous requests to the cache line have been serviced. This guarantees that the cache line's state as represented by the directory always corresponds to the cache state in the different nodes and corner cases are avoided. Two simplified examples can be found in the Appendix. As can be seen from those examples, most global accesses will involve a total of three bus transactions; two in the requesting node and one in either the home node or the owning node. This may create a bandwidth problem for applications with poor memory locality, the effects of which are discussed further in Section 8. On the other hand, applications with poor locality will experience a latency problem in DSMs anyhow (as shown in Figure 6.)

5. Simple latency comparison

How does WildFire's MSMP approach compare to other system families? That question cannot be answered by looking at a single latency or bandwidth number since it involves a wide range of system sizes. We have elected to compare our DSM approach with the two commercially available systems: Origin 2000™ from SGI and NUMA-Q™ from Sequent. The three systems represent three very different approaches to building DSM systems, as shown in Figure 2. In order to compare technology from the same timeframe we use the data from Sun's previous generation SMPs, E6000, here.

Origin is a DSM-optimized architecture focused on reducing the remote latency to clean remote data. Each DSM

Table 1 The latency of some scalable architectures measured by the lm-bench benchmark [15]

	Origin[1]	NUMA-Q[1]	WildFire
CPU	R10000	P6	UltraSPARC I/II
CPU cache	1-4 MB	.5 MB	.5 - 4 MB[2]
#CPUs/node	2	4	28
Node cache	None	32 MB	0-6 GB (CMR)
Page replication	Read	Read	Read/Write
Local memory Latency	472 ns	250 ns	330 ns (252 ns)[3]
Local cache2cache Latency	1036 ns	300 ns	470 ns (400ns)[3]
Remote memory latency (nearest node)	704 ns	2000 ns	1762 ns
Remote cache2cache (3hop nearest nodes)	1272 ns	2500 ns	2150 ns
Latency for extra router hop	50 ns	20 ns	No router
Memory overhead replicating 10% of the data in all the nodes (~100 CPUs)	490%	240%	30% (3 extra copies)
# router hops grows	Log	Linearly	No router

node consists of two R10000 CPUs connected to a memory controller (Hub in Figure 2). The directory state is co-located with the data in the DRAM banks, allowing the large directory state to be accessed cheaply if the data are clean in the home node. However, the directory lookup in DRAM also will be on the critical path for accesses to dirty data in the cache of another node. Each CPU has a fairly large cache, but there is no node cache (a.k.a. remote access cache [13]) to help create extra node locality. Instead, a high-bandwidth interconnect of hypercube type is implemented by distributed routers (R) to handle the increased global traffic. The bisectional bandwidth of a system with 32 CPUs is 6.4 GB/s [10, 12].

NUMA-Q is built from small proprietary SMP building blocks with four P6 processors in each node. Each node has a node cache of 32 MB (N$), shared by all the CPUs. Since the nodes are built from SMP nodes, each with its own memory controller, the directory could not be co-located with the data. However, the directory is still implemented using SDRAM. The coherence protocol is a variation of the SCI, with a four-hop dirty-data protocol. It is implemented in two chips (here called: I/F and GC) in a programmable fashion to lower the risk of protocol bugs. Nodes are connected using an SCI ring of 1 GB/s, implemented with Datapump™ chips by Vitesse (DP) [14].

Table 1 lists some properties of the compared systems. WildFire's local latency is shorter than Origin's and longer than NUMA-Q's. For remote traffic, the order is reversed with Origin as the fastest, followed by WildFire and NUMA-Q. Origin's and NUMA-Q's remote latency are more dependant on system size than WildFire. WildFire's larger nodes also incurs a lower cost to replicate data.

The remote-latency numbers in Table 1 are fairly hard to compare out of context. A system built from large nodes will not experience any remote latency until the system size is larger than its node size, which is at 28 CPUs for WildFire. Even above 28 CPUs, a comparison is not straightforward. At this size, traditional DSMs will experience, on average, a much longer latency than the quoted latency in the table, which is to "nearest node."

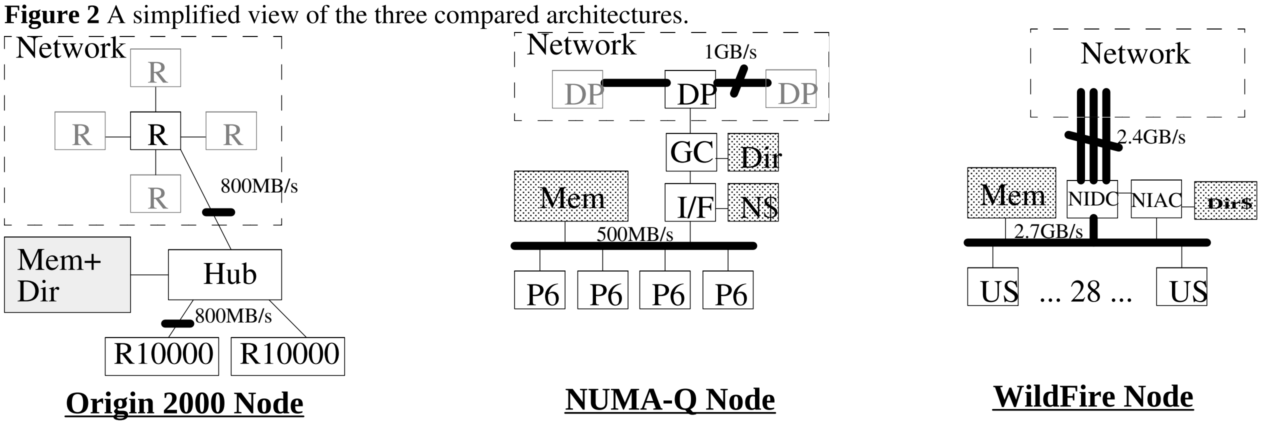
A normal SMP application is not optimized for an architecture with non-uniform memory-access time. If the application and the operating system are not rewritten, we can assume that the access pattern is randomly distributed over the entire shared address space.

*avg_latency=locality*local_latency+(1-locality)*remote_latency* **[F1]**

For random accesses with no extra optimizations, the locality is approximately 1/N, where N is the number of nodes in the system. As N grows, this locality-for-free effect diminishes. Figure 3 shows the average access time to clean data from memory, assuming random distribution of accesses. In the 4 to 28 CPU range, traditional DSMs have an access time 2 to 8 times that of WildFire's. This is not surprising since WildFire simply behaves as an E6000 for this system size. It should be noted that this is really the sweet spot of the market for servers. Origin shows a slightly better latency than WildFire for systems above 28 CPUs.

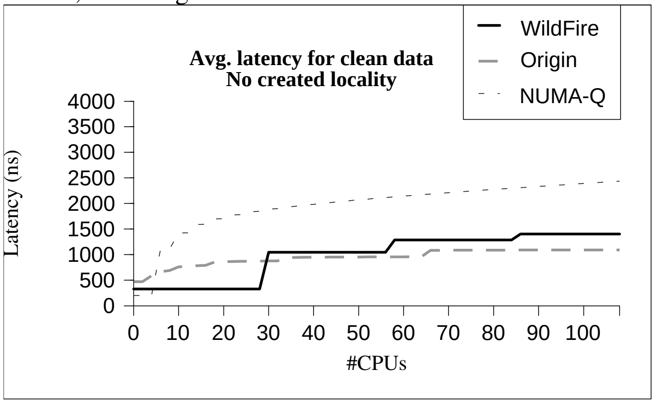
In systems with large caches, a large fraction of accesses is to migratory-shared data structures. Typically, such data structures will not be found clean in the home memory and

Figure 2 A simplified view of the three compared architectures.



[1]The NUMA-Q number are not explicitly published and have been extracted from [14], the Origin 2000 numbers are from [10]
[2]UltraSPARC I of 167 HMz and 0.5 Mbyte caches are used in this study.
[3]This is the current access time for E6500, the number used for the comparison and in Section 8 are based on the older E6000 with 167 MHz CPUs.

Figure 3. The average latency to shared memory for cache misses, assuming random distribution.



will be satisfied from some other cache. In a NUMA system, that cache will typically be in a node other than the requesting node or home node. Figure 4 shows the average latency for this type of accesses. The figure shows an even larger latency difference between WildFire and other DSMs for systems with less than 28 CPUs. The reason for this is the rather efficient cache-to-cache implementation within a single WildFire SMP node supported by a broadcast snooping protocol. In a DSM system, a cache-to-cache transfer most often involves a so-called three-hop or four-hop transaction: a remote access to the home node, a lookup in the directory, a remote access to a third node where the dirty data resides and, finally, a remote data packet sent back to the home node. Both NUMA-Q and Origin implement their directory structure in SDRAM. WildFire has a large directory cache implemented in SRAM. The data set size for migratory data tends to be fairly small and fits in the directory cache. This reduces the directory-lookup overhead for systems larger than 28 CPUs.

Another factor that reduces the random cache-to-cache latency is the 1/N effect of WildFire's rather large nodes. This makes up for WildFire's long latency to remote nodes and makes it the overall fastest solution for migratory data.

A real application would experience an access mix of the two access types discussed above. The ratio between the two categories varies widely with applications. We have

Figure 4. The average access time to satisfy a migratory cache miss, assuming random distribution.

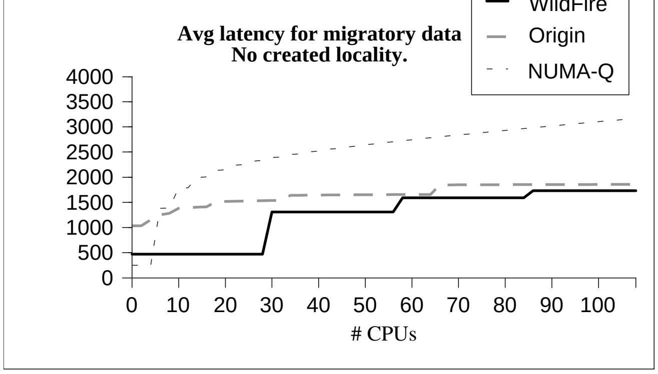
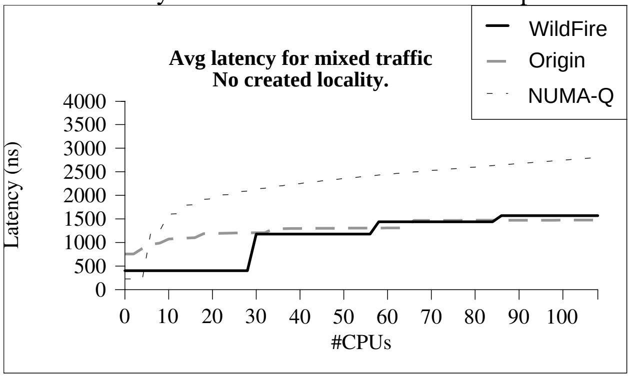


Figure 5. The average latency to satisfy a cache miss assuming 50% cache-to-cache misses and average distribution of memory accesses to the shared address space.



seen many commercial applications with 40--60% cache-to-cache accesses, which have also been reported by others [1]. Figure 5 shows the average access time for the three systems assuming 50% of each kind. The figure shows that the WildFire approach is very advantageous in systems up to 28 CPUs. Above 28 CPUs, WildFire is roughly on par with Origin, but still far ahead of NUMA-Q.

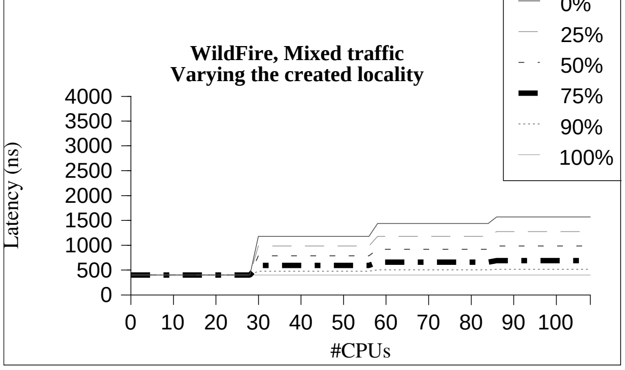
If the application and the operating system are altered, or large node cache added, more locality can be created. The total locality will have a component caused by this optimization. We call it "created locality" and define it as the fraction of accesses that are made local to a node due to some extra measure. The remaining transactions will still be randomly distributed and experience the 1/N locality:

locality = created_locality + (1-created_locality) / N **[F2]**

This way we isolate the locality effect caused by the large nodes from the forced locality effect, which is less dependant on the number of nodes in the system.

One way of creating locality is by adding a third-level node cache, such as in the NUMA-Q, or by adding support for migration and read-only replication with a page granularity, as done by Origin. WildFire's CMR can be viewed as a large node cache, but will also effectively support page migration and read-only or read/write replication of pages with a coherence unit of one cache line. The effect of WildFire's average latency as a function of its created

Figure 6. The effect of created locality on the average access time assuming 50% cache-to-cache misses



locality can be studied in Figure 6. This figure assumes that the created locality is independent of the number of nodes in the system.

It is apparent that DSMs, regardless of the efficiencies of their implementations, require a substantial amount of created locality in order to run applications well. The OLTP application studied in this paper resulted in 75% created locality. A 75% created locality (highlighted in the picture) keeps the average latency in the range of 1.5 times the SMP's latency.

6. Application-transparent optimizations

To successfully exploit WildFire as a large SMP, it should not be required that user processes be aware of their locations, or the node locations of their memory regions, to get good performance and scalability. WildFire uses additional kernel modules for Coherent Memory Replication and Hierarchical Affinity Scheduling control. CMR and HAS policies are implemented by daemon processes which periodically sample CPU load and the excess-remote-cache-miss counters. Set-aware load balancing moves processes from one node to another to balance CPU load, and the CMR daemon will migrate or replicate memory pages to minimize the frequency of remote memory cycles. In this way, WildFire transparently and continuously optimizes the location of both processes and their favorite memory regions for best application performance. Additionally, the system will attempt to optimize the initial placement of processes and memory for improved locality. The kernel itself is replicated transparently on all nodes at boot time; but, the prototype OS used in the tests described below does not replicate many of its data structures.

To analyze the application-transparent optimizations, the following questions were asked:

1. Is MSMP as implemented in WildFire viable for commercial workloads running unmodified SMP applications?
2. What operating system features can be used to effectively hide the latency of remote memory from the application?
3. How close to an SMP's performance characteristics is an MSMP?

7. Application example

We have chosen to study two of the key features and their impact on performance in more detail: the coherent memory replication and the scheduling policies. A large commercial OLTP benchmark workload is used. It exhibits intense shared-data update activity, stressing the ability of the system to deal with migratory data sharing. Cache-miss ratios and memory traffic are high, even for commercial workloads, and this workload imposes a unique demand for a very large, shared-memory region.

The system was scaled to 900 warehouses on 240 disks connected to one of the nodes through 8 fiber- channel interfaces. WildFire's I/O architecture does not add significant performance penalties for inter-node programmed I/O or DMA operations. The shared memory was configured to be approximately 2 GB in size, and 4 GB of physical memory were configured on each node so as to have enough memory for both the Shared Global Area (SGA) and process private memory.

Database testing was done on a 16 CPU E6000 and on a two-node E6000 WildFire with 8 CPUs in each node. For the two-node tests, the second node was connected to the existing system and half the CPU boards (including their memory) were moved over. As mentioned before, the I/O was left connected to only one of the nodes. This small-scale evaluation using only 16 CPUs allowed us to compare our results with an equivalent "ideal" SMP system as a standard for comparison and to avoid software contention which would impose an artificial limit on the performance of both configurations. We wanted to focus primarily on the effect of remote-memory latency in these experiments.

8. Experimental results

The effects of the different optimizations were studied by first turning them on one by one, and later turning them all on, while measuring the execution time and the ratio of memory accesses staying local in one node. The following configurations are compared:

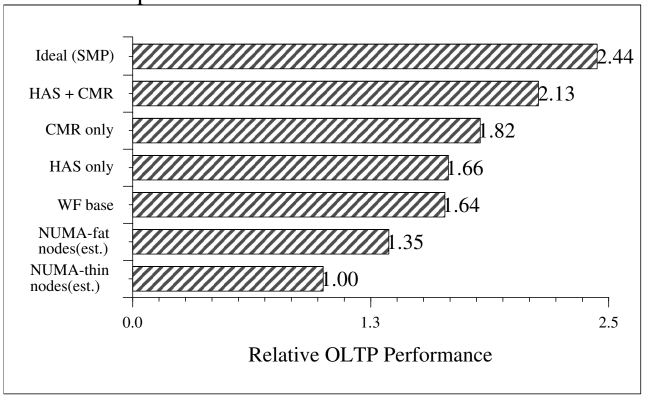
NUMA fat nodes represents the expected performance of a NUMA system built from hardware identical to WildFire's, but running a completely unmodified operating system resulting in random distribution of accesses. Still, 50% of the memory accesses are local since the system is built from only two nodes. However, no completely unoptimized WildFire operating system exists, so this performance number had to be modelled based on the number of L2 cache misses measured on a real WildFire system, and the average access time assuming 50% local accesses [F1].

NUMA thin nodes is modelled similarly to the NUMA fat nodes, but assumes a system built from eight nodes with two CPUs in each node, i.e., 12.5% local accesses, and a local and remote latency equal to WildFire's. The performance of this system is set to 1.0 in Figure 7. The increased memory locality of NUMA with fat nodes gives it a 35% performance edge over the NUMA built from thin nodes.

WildFire base is the real WildFire system with as many optimizations as possible turned off. The allocation of memory is locality aware and some kernel text and data structures are statically replicated by this kernel. This version of the kernel has a flat affinity scheduler. WildFire base runs 21% faster than NUMA fat. It can be expected that a NUMA thin would experience a similar speedup if a similarly optimized kernel was used.

HAS only is running WildFire base with the hierarchical affinity scheduling turned on. This will maximize the time a process stays within a node; it also introduces some overhead in the scheduler and adds some extra imbalance to the system. On the positive side there is an increased likelihood of finding requested data in a cache local to the node.

Figure 7. Relative OLTP performance from different DSM and SMP implementations.



However, the state stored in the local caches is not large enough to make this a huge advantage and the measured performance gain is very small compared to WildFire base.

CMR only is running WildFire base with the coherent memory replication turned on. Some pages will be replicated in both nodes. Processes frequently migrate between nodes; but, since much of the important read-mostly data is replicated in both nodes, the memory locality is increased enough to give this system an 82% performance advantage over a NUMA built of thin nodes.

HAS and CMR has both the hierarchical affinity scheduler and the coherent memory replication turned on. The combination of rare process migration between nodes and the huge read/write caches supported by CMR result in a multiplication effect compared to using only one of the two techniques at a time. The net result is a 113% performance increase compared to the plain NUMA built from thin nodes.

Ideal SMP is built from a single SMP with 16 processors, as discussed earlier, and thus experiences "100% memory locality." This represents the upper bound for these kinds of locality optimizations. The performance of the HAS and CMR system is only 13% slower than this ideal system when running this unaltered commercial SMP application on the WildFire port of the Solaris operating system.

The most performance-critical property of the different systems is the amount of memory locality. Memory locality was measured as the ratio of local memory accesses to total memory traffic (L2 cache misses and DMA traffic) using hardware counters on the WFIs, as shown in Figure 8. Expected locality for the modeled fat/thin NUMA systems is also shown as a reference. Kernel replication and initial placement policy together accounted for 71% of the local memory cycles, up from an expected 50% for random placement. Coherent Memory Replication increased the measured locality to 87%.

A measured locality of 87% in a two-node system implies a "created locality" of 75%. This is a measure of the effectiveness of the kernel replication and CMR. With a created locality of 75%, the remaining 25% will be distributed equally among the two nodes, resulting in a 75% + (100% - 75%)/2 equals the 87.5% measured locality on a two node system.

As mentioned before, a global transaction will generate a total of three bus transactions while local transaction only generates one transaction. This increase in bus traffic is the bandwidth bottleneck in WildFire. However, good locality will limit the negative effect. The SMP equivalent bandwidth, SysBW, can be derived as:

$$SysBW * (locality+3*(1-locality))= BusBW*\#Nodes \quad ==>$$
$$SysBW = BusBW *\#Nodes/(3-2*Locality) = \qquad (using\ [F2])$$
$$= BusBW*\#Nodes/(3-2*(created\_locality+(1-created\_locality)/N)) \qquad \textbf{[F3]}$$

$$SysBW(2\ Nodes,\ 75\%\ created\ locality) = 4.3\ GB/s.$$

E6000's CPUs share 2.7 GB/s. In WildFire, each CPU runs at 87% the speed in E6000, so each CPU requires only 87% of the bandwidth. The number of CPUs we can we put in each of the two node while maintaining the equivalent per-CPU bandwidth can be calculated as:

$$CPUsPerNode = 4.3/(2*0.87*2.7) = 0.91$$

Thus, for this kind of application, each WildFire node should be able to have about 91% as many CPUs as a single E6000 while maintaining the same bus utilization.

To evaluate the effect of replicating only the frequently accessed region of the SGA, we conducted a crude experiment, relying on the known characteristics of the layout of the SGA in memory. In particular, many of the "hot" data structures reside in the lower address range of the SGA. Thus, limit the replication of the SGA to the lower end of its address range is an approximation of the performance achieved if only 25% of SGA could be replicated. Results are shown in Figure 9.

Figure 8. The locality of traffic, i.e., what ratio of cache misses is satisfied locally in a node.

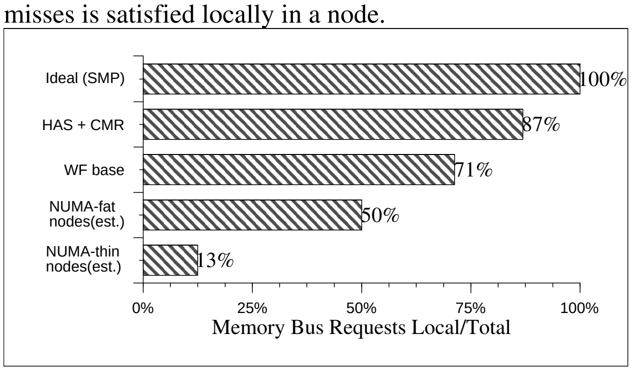
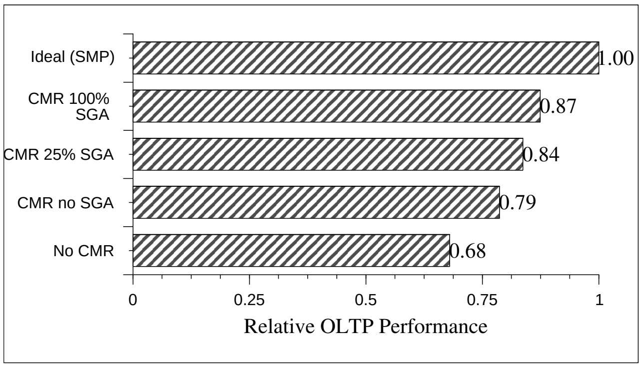


Figure 9. Graph showing the importance of large data replication.



Limiting replication to 25% of the SGA, or approximately 450 MBytes out of 1.8 GBytes, resulted in some 3% performance degradation. These results imply that dynamic replication for special memory regions such as database SGAs could result in a savings of physical memory (less physical replication across nodes). Fat nodes are a definite advantage here; where the two-node system uses 450 MBytes extra physical memory (450 MB * 1 nodes), an eight thin-node system would need to use 3.1 GB of extra physical memory (450 MB * 7 nodes) to replicate 25% of the SGA region.

9. Related work

COMA [7] and S-COMA [8, 16] are similar to, and have inspired, the creation of WildFire. However, WildFire hardware has been simplified and does not allow the home to migrate between nodes and is not a true COMA. Home migration is instead supported through the software for page migration. WildFire also has a sophisticated algorithm for deciding which data structures should be replicated, while COMA and S-COMA have no support for such a selection strategy.

KSR-1 was architecturally very similar to WildFire in that groups of processors, connected by a ring, formed large nodes. Several nodes were connected together using yet another ring structure. However, rather than using a high-volume tight nonscalable technology to form the large nodes, the rather ineffective ring structure was used also at this level. KSR claimed to be a COMA; but, since each page required space to be allocated all the time in a "home node," it actually only implemented memory replication. Similarly to COMA and S-COMA, KSR only supported replication and cannot dynamically switch to/from the cc-NUMA strategy.

The R-NUMA [6] and the ASCOMA [11] algorithm for page caching strategy is very similar to the one used in WildFire; but, here WildFire chose a simplified implementation strategy based on associative counters. We have also added a scheduling strategy and shown it to be vital to good performance when using an algorithm such as R-NUMA. PRISM [5] presents yet another alternative approach to switching between cc-NUMA and S-COMA. These three simulation studies are based on fairly short-running technical applications. The performance study presented here is based on a real hardware implementation and a long-running commercial application.

10. Conclusion

In conclusion, MSMP, as implemented in Sun's WildFire prototype, appears to be a viable architecture for OLTP workloads. Application-transparent locality and load balancing are able to relieve the burden of memory-locality awareness from the database application. Coherent Memory Replication and Hierarchical Affinity Scheduling are effective kernel features and are able to manage processes and memory pages for good locality.

These results on the WildFire prototype demonstrate that MSMP with Coherent Memory Replication can effectively hide the locality issue from user processes for the studied application, extending the SMP model beyond the SMP box. The performance results demonstrate 2.13 times that of a NUMA implementation with no optimizations. While the ideal implementation, an even larger SMP with the same memory access time as the local memory access time of WildFire shows 2.44 times. WildFire's applications-transparent optimizations bring it within 13% of the ideal performance.

WildFire is a prototype installed at number of external beta sites. It is not a product offered by Sun Microsystems, Inc.

11. Acknowledgments

The WildFire architecture was developed by Sun's High-End Server Engineering group, based in Massachusetts. We would like to thank the entire team for tireless and enthusiastic work during the WildFire hardware and software implementation. Cathy Melior-Benoit, Anders Landin, Ken Won, Alan Mandel, Jon Wade, Brad Carlile, Andy Phelps, Alan Charlesworth and Ashok Singhal provided helpful comments on early drafts of this paper. Mark Hill and David Wood added valuable help during the development of the WildFire architecture as well as this paper.

REFERENCES

[1] L. Barroso, K. Gharachorloo, and E. Bugnion. *Memory System Characterization of Commercial Workloads*. ACM/IEEE *International Symposium on Computer Architecture (ISCA)*, June 1998.

[2] T. Brewer and G. Astfalk. *The Evolution of the HP/Convex Exemplar.* In Proceedings of *COMPCON* Spring 1997.

[3] B. Carlile. *Seeking the Balance: Large SMP Warehouses. Database Programming & Design*, August 1996.

[4] A. Charlesworth. *STARFIRE: Extending the SMP Envelope. IEEE Micro* Jan/Feb 1998. (http://www.sun.com/servers/enterprise/10000/wp/)

[5] K. Ekanadham, B-H. Lim, P. Pattnaik, and M. Snir. *PRISM: An Integrated Architecture for Scalable Shared Memory*. In *Proc.* HPCA 1998.

[6] B. Falsafi, D. Wood. *Reactive NUMA: A Design for Unifying S-COMA with CC-NUMA*. ACM/IEEE *International Symposium on Computer Architecture (ISCA)*, June 1997.

[7] E. Hagersten, A. Landin, and S. Haridi. *DDM - A Cache-Only Memory Architecture*. *IEEE Computer,* 25(9):44-54, Sept. 1992.

[8] E. Hagersten, A. Saulsbury, and A. Landin. *Simple COMA Node Implementations.* In Proceedings of *Hawaii International Conference on System Science*, January 1994.

[9] M. Heinrich, et al. *The Performance impact of flexibility in the Stanford FLASH multiprocessor*. Proceedings of the 6th *International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS) 1994.

[10] C. Hristea, D. Lenoski, and J. Keen. *Measuring Memory Hierarchy Performances of Cache-Coherent Multiprocessors Using Micro Benchmarks.* In Proceedings of *Supercomputing* 1997.

[11] C-C. Kou, J. Carter, R. Kuramkote, and M. Swanson. *AS-COMA: An Adaptive Hybrid Shared Memory Architecture*. ICPP Aug 1998.

[12] J. Laudon and D. Lenoski. *The SGI Origin: A ccNUMA Highly Scalable Server.* ACM/IEEE *International Symposium on Computer Architecture (ISCA)*, June 1997

[13] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy. *The directory-based protocol for the DASH multiprocessor. International Symposium on Computer Architecture (ISCA)*, 1990.

[14] T. Lovett and R. Clapp. *STiNG: A cc-NUMA computer system for the commercial marketplace.* ACM/IEEE *International Symposium on Computer Architecture (ISCA)*, June 1996.

[15] L. McVoy and C. Staelin. *lmbench: Portable tools for performance analysis.* USENIX January 1996.

[16] A. Saulsbury, T. Wilkinson, J. Carter, A. Landin, and S. Haridi. *An Argument for Simple COMA.* In *Proceedings of* HPCA 1995.

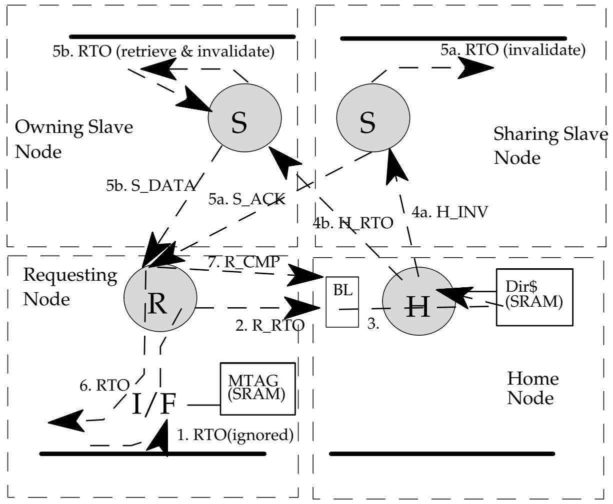
[17] A. Singhal, D. Broniarczyk, F. Cerauskis, J. Price, L. Yaun, C. Cheng, D. Doblar, S. Fosth, N. Agarwal, K. Harvery, E. Hagersten, and B. Liencres. *A High Performance Bus of Large SMPs.* In *Proceedings of IEEE Hot Interconnects*, P 41-52, Aug 1996.

[18] *TPC-C and TPC-D Results.* http://www.tpc.com/

APPENDIX

Example: Read-to-own (RTO) access, data shared in two remote non-home nodes

1. The I/F detects an RTO which cannot be satisfied locally. It asserts the ignore signal and queues the transaction for the GC. If the accessed page is CMR, do an LPA2GA translation.
2. Send an R_RTO request to the home node.
3. The cache line address is marked as "blocked" in the blocking logic in the home node; a home agent is allocated and performs a directory-cache lookup. The directory entry identifies the two nodes with a shared copy, one of them is identified as the "owner."
4a. The home agent sends H_INV demands to the shared node's slave agent. If the accessed page is CMR in the slave node, do GA2LPA.
4b. The home agent sends an H_RTO demand to the owner's slave agent. If the accessed page is CMR in the slave node, do GA2LPA.
5a. The shared slave agent initiates an invalidate SMP transaction (RTO) and sends a S_ACK reply to the request agent once the SMP transaction has been queued.
5b. The owned slave agent initiates invalidate-retrieve SMP transactions and sends a S_DATA reply to the request agent.
6. The replies carries the "number-of-replies" which tells the request agent to expect two replies before it reissues the RTO transaction on its SMP bus and provides the data.
7. After receiving both replies, the request agent sends an R_CMP to the home node's blocking logic, which now will allow new transactions to the cache line.



Example: Writeback (WB) NUMA, not canceled

1. The I/F detects an WB to remote memory. It asserts the ignore signal and queues the transaction for the GC. The effect is that the issuing device has still not seen its WB.
2. A request-agent instance is allocated in the GC and sends an R_WB request to the home node.
3. The cache line address is marked as "blocked" in the blocking logic; a home agent is allocated and performs a directory-cache lookup. If the requesting node is not the "owner," the writeback is canceled.
4. The home agent sends H_ACK demands to the shared node's slave agent.
5. The request agent reissues the WB transaction on its SMP bus.
6. The issuing device now sees its own WB and will provide the data to the request agent if it is still the owner. If it is no longer the owner, it will cancel the writeback operation.
7. The request agent sends a data packet to the home node, which unblocks the cache line
8. The home agent reissues the WB transaction on its SMP bus and updates the memory.