To appear in the **Proceedings of the Eight International Conference on Parallel Architecture and Compilation Techniques (PACT)**, 2000.

# Address Partitioning in DSM Clusters with Parallel Coherence Controllers

Ilanthiraiyan Pragaspathy
Compaq Computer Corporation
334 South street
Shrewsbury, MA 01545
ilanthiraiyan.pragaspathy@compaq.com

Babak Falsafi
School of Electrical and Computer Engineering
Purdue University
1285 Electrical Engineering Building
West Lafayette, IN 47907-1285
*babak@ecn.purdue.edu, http://www.ece.purdue.edu/~babak*

## Abstract

*Recent research suggests that DSM clusters can benefit from parallel coherence controllers. Parallel controllers require address partitioning and synchronization to avoid handling multiple coherence events for the same memory address simultaneously. This paper evaluates a spectrum of address partitioning schemes that vary in performance, hardware complexity, and cost. Dynamic partitioning minimizes load imbalance in controllers by using hardware address synchronizers to distribute the load among multiple protocol engines at runtime. Static partitioning obviates the need for hardware synchronization and assigns memory addresses to protocol engines at design time, but may lead to load imbalance among engines.*

*We present simulation results indicating that: (i) dynamic partitioning performs best speeding up application execution on an 8 8-way cluster on average by 62% using four-engine as compared to single-engine controllers, (ii) block-interleaved static partitioning using low-order address bits is an attractive alternative and performs close to dynamic partitioning when protocol occupancies are low or there is little queueing, and (iii) previously proposed static schemes that partition memory pages either into home and remote engines or using low-order page address bits result in a high load imbalance in parallel controllers.*

## 1 Introduction

Clusters of symmetric multiprocessors (SMPs) are emerging as the architecture of choice for building large-scale parallel servers. Designers exploit SMPs as cost-effective building blocks, and connect a cluster of them together using low-latency high-bandwidth networks into large-scale multiprocessors. To preserve SMP software compatibility and portability, designers often connect the cluster using distributed shared memory (DSM). DSM extends SMPs' shared address space globally across the system's physically distributed memory [5,7].

DSM clusters typically use a directory-based cache coherence protocol to implement a global shared address space. Conventional DSM cluster designs incorporate hardwired coherence controllers with a single protocol event queue and engine per SMP. The queue serializes coherence event handling (e.g., servicing a shared miss) both from local SMP processors to remote memory and from remote SMP processors to memory local to the engine.

Recent research [11,4] indicates that conventional DSM cluster designs with serial controllers often suffer from a communication bottleneck. Sharing a single protocol engine among multiple SMP processors places high service demands on the engine. To address the controller bottleneck, recent proposals use parallel coherence controllers with either multiple and/or pipelined engines [11]. Parallel controllers handle multiple coherence events simultaneously, exploiting parallelism in coherence activity, improving communication performance, and mitigating the controller bottleneck.

To guarantee correct protocol functionality, parallel controllers require coordination in servicing coherence events to avoid handling multiple events on the same memory block simultaneously. Much like conventional pipelined processors, some proposals for parallel controllers use address interlocks [11] to prevent multiple coherence events on the same block to be serviced simultaneously. An address interlock mechanism compares the memory block address at the head of coherence event queue with addresses of the events currently in service, and prevents event dispatch upon a match. While simple to implement, address interlocks result in busy waiting and may reduce performance in the presence of a burst of events on the same memory block.

Alternatively, other proposals avoid handling multiple operations on a memory block simultaneously by partitioning the shared address space and exploiting event parallelism across partitions. Such an address partitioning can happen either statically at design time [11,8] or dynamically at runtime [4]. The static schemes simply use address demultiplexers to decide which engine will service a coherence event. The dynamic schemes require an address synchronization mechanism built into the coherence event queue to serialize servicing multiple events for the same memory block.

Dynamic schemes are expected to improve performance over static schemes because they use a single coherence event queue to dispatch and balance the load in the parallel controller. It follows from a simple queueing theory result that dynamic partitioning — i.e., multiple servers/single queue — is always superior in performance to static partitioning — i.e., multiple servers/multiple queues [6]. Static schemes, however, eliminate the need for address synchronization, reduce the hardware complexity, and may exhibit a balanced load in practice.

In this paper, we compare and contrast different address partitioning schemes to enhance parallelism in DSM controllers. We evaluate address partitioning in the context of parallel controllers using multiple coherence engines. Other papers have studied DSM controller design space in great detail and have compared and contrasted multiple coherence engines against pipelining [11]. While our results are equally applicable to pipelined engine implementations, evaluating address partitioning schemes for pipelined engines is beyond the scope of this paper.

We compare address partitioning by executing shared-memory applications on simulated DSM clusters with multi-engine controllers. Our results indicate that:

- Dynamic partitioning performs best eliminating the load imbalance among engines; on average a two-engine system improves performance by 37% and a four-engine system by 62% as compared to a single-engine system in a cluster of 8 8-way SMPs.
- Block-interleaved partitioning is an attractive alternative and performs close to dynamic partitioning when protocol occupancies are low (as in protocols with smaller block sizes) or there is little queueing (as in smaller SMP nodes); block-interleaved partitioning in a 8 8-way SMP achieves 97% of the performance of dynamic partitioning using 32-byte blocks.
- Home-based and page-interleaved partitioning result in the highest load imbalance among multiple engines and a lower performance achieving only about 85% of the performance of dynamic partitioning for the base case system.

The rest of this paper is organized as follows. Section 2 describes the base DSM cluster systems we study in this paper. Section 3 proposes our addressing partitioning schemes for multi-engine controllers. Section 4 presents the performance results, and finally Section 5 concludes the paper.

## 2 DSM Clusters

Figure 1 illustrates the anatomy of the distributed shared-memory machine we study in this paper. The DSM consists of a cluster of SMPs connected together over a low-latency high-bandwidth network. A coherent shared bus implements a fine-grain shared-memory abstraction within each SMP node. A DSM cluster device on each node extends the SMP shared-memory abstraction using a directory-based cache coherence protocol across the entire machine.

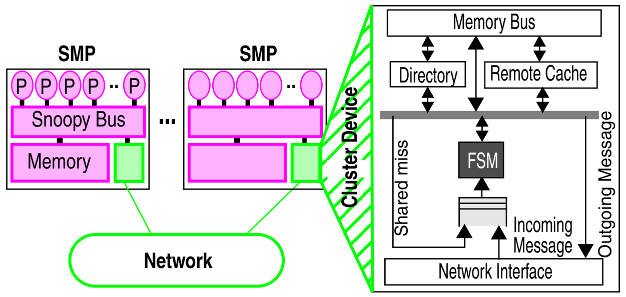


**FIGURE 1. A DSM cluster.**

The cluster device is a snoopy board that interfaces the SMP memory bus, on one side, and the network switches, on the other. The device includes a *directory* maintaining the identity of the remote sharers for each memory block on that node. A *remote cache* maintains a copy of the most recently-referenced remote data and serves as a backup for the SMP processor caches. The remote cache can be an SRAM/DRAM cache [9] on the device or it can be part of the SMP's memory [3,5]. A protocol *engine* — in the form of a finite-state-machine (FSM) — implements coherence across SMPs, services the coherence events out of a protocol *event queue*, and manages accesses to the directory and the remote cache. In the interest of brevity, in the rest of the paper we will also refer to a protocol engine as an FSM.

Memory pages are allocated and distributed round robin across the SMP nodes. Each node is assigned a set of pages for which the node will serve as the designated *home*. The directory on every node keeps track of the sharers for the home pages on that node. There are two types of coherence events on every node. A *shared miss* requiring the invocation of a coherence action on remote nodes, and an *incoming message* from other nodes requesting a local coherence action. The shared misses and incoming messages require access to the directory and/or the remote cache depending on the shared page they request access permission to. A shared miss or an incoming message (e.g., request a block copy) for a home address accesses the directory to find the current sharer/owner of a memory block. Conversely, shared misses and incoming messages (e.g., invalidation messages) on remote addresses only look up and access the remote cache.

## 3 Address Partitioning Schemes

In this section, we describe the address partitioning schemes we evaluate in this paper. The partitioning schemes vary in performance, and hardware complexity and cost. The schemes can generally be classified into *dynamic* and *static* address partitioning. A dynamic scheme relies on address synchronization hardware to exploit parallelism across different memory addresses while synchronizing and serializing all coherence events for a given

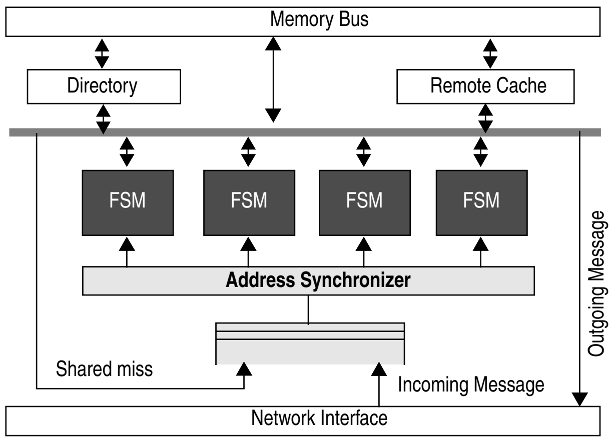


**FIGURE 2. Dynamic partitioning scheme.**
*The address synchronizer dynamically selects protocol events for different memory addresses to dispatch to the FSMs.*

address [4]. In a static scheme, the memory addresses are statically partitioned among the protocol engines so that each engine is responsible for a specific set of memory blocks. We study two different classes of static partitioning schemes in this paper — *interleave* partitioning, and *home-based* partitioning. These schemes trade off hardware complexity for a balanced load among the engines and improved performance.

## 3.1 Dynamic Partitioning

Dynamic partitioning has been previously studied in the context of DSMs with multiple software coherence controllers [4]. In this study, we evaluate dynamic partitioning in the context of hardwired multi-engine controllers and compare it as a partitioning scheme against other less hardware-intensive schemes.

In dynamic partitioning (Figure 2), protocol events are dispatched to any available FSM using in-queue synchronization. Protocol requests for memory addresses from the same cache block are handled serially by the same FSM, whereas protocol requests for memory addresses from different cache blocks are handled in parallel. An *address synchronizer* [4] performs a fully-associative search on the protocol event queue and finds independent events to dispatch, much as instruction issue logic searches the instruction window in superscalar processors to issue and execute independent instructions.

The synchronizer's search speed depends on the number of queue entries the logic must search. The number of messages in a DSM coherence queue for a single memory block is typically low and often does not exceed the number of processors in the system. As such, in practice the queue depth needed to find a small number (e.g., 2 to 4) of independent coherence events is quite small and well within the maximum depth that can be searched in a single cycle in hardware [2].

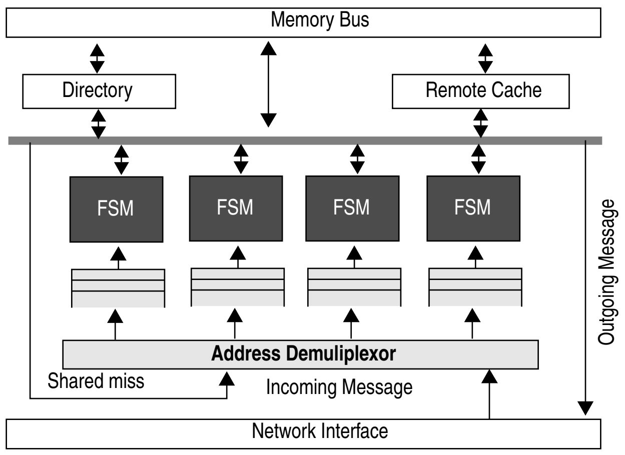


**FIGURE 3. Static interleave partitioning scheme.**
*The address bits are used to assign protocol events to the engines. For example, in the case of two FSMs using block interleaving, all odd block addresses are assigned to one FSM and all even addresses assigned to the other.*

While dynamic partitioning can best balance the load, it requires the directory and the remote cache to allow simultaneous access from all the FSMs. To reduce the likelihood of high protocol occupancy and contention among the FSMs, the directory and the remote cache can be multiported at the cost of higher hardware complexity. In this paper, we assume the shared resources in all the schemes to be multiported and contentionless and only focus on the load balancing among the FSMs.

## 3.2 Static Interleave Partitioning

In interleave partitioning (Figure 3), address interleaving is used to synchronize parallel event dispatch [11]. Here, address bits are used to determine which FSM should handle the protocol events for a memory address. The effectiveness of the scheme could depend on the actual bits used to partition addresses. We study two interleave partitioning schemes: *block-interleaved* partitioning and *page-interleaved* partitioning. In block-interleaved partitioning, the lower order bits of the block address are used to determine the partition. Page-interleaved partitioning uses the higher order bits of a memory address to determine the partition.

In interleaved partitioning, FSM utilization depends on both the memory access stride and sharing patterns in the application. Block-interleaved partitioning optimizes the protocol event dispatch to favor fine-grain sharing among processors; multiple FSMs can handle protocol events from processors actively sharing the same page. Block interleaving, however, may lead to a load imbalance for applications with regular memory access strides; e.g., array-style applications which access only even or odd memory blocks in alternating phases, or applications which pad data structures in multiples of cache blocks [13]. Conversely, page-

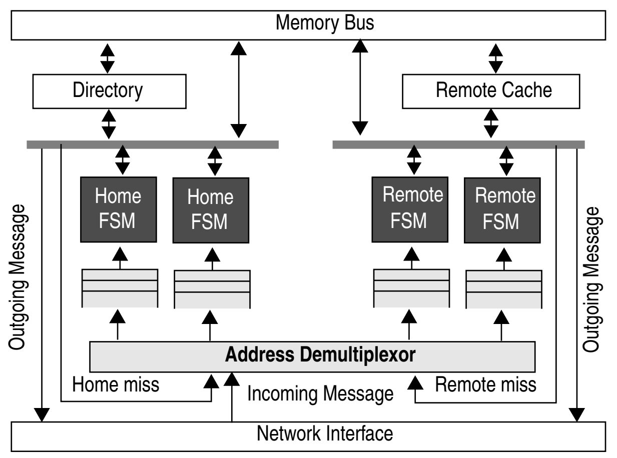


**FIGURE 4. Home-based partitioning scheme.** *FSMs are grouped into those responsible for home addresses (left) and remote addresses (right).*

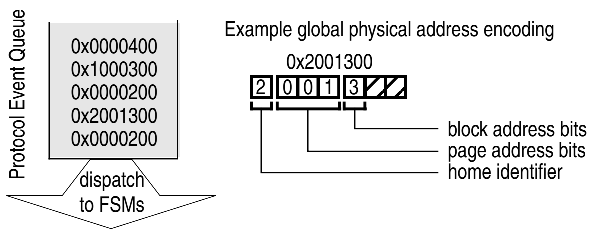
interleaved partitioning favors applications with coarse (i.e., page-based) sharing granularity while allowing any memory block (even or odd) across pages to be handled by multiple FSMs.

Interleave partitioning eliminates the sophisticated address synchronization hardware in dynamic partitioning and uses a simple demultiplexer which selects a protocol event based on the address bits. A flexible design may allow configuring the demultiplexor at boot time or the start of application execution using simple address masks to select the specific address bits identifying the partition. Much like dynamic partitioning, in interleave partitioning the FSMs share the protocol resources (e.g., the directory and remote cache) and may require multi-ported resources.

Interleave partitioning is also attractive from a design perspective because it allows implementing multi-engine DSMs using multiple device boards. Upon demand for higher communication bandwidth and performance, customers can plug in additional DSM boards into each SMP node using interleave partitioning — e.g., a single DSM board per node design can be upgraded to two boards per node where each board is in charge of different address partitions. A simple device configuration at boot time will be required to (statically) partition the address space among multiple boards. Such a plug-and-play style of multi-engine design is not so simple to implement with dynamic partitioning because of the requirement to search and synchronize addresses in a single event queue.

## 3.3 Static Home-Based Partitioning

In home-based partitioning (Figure 4), protocol events for (local) home memory addresses are handled by one set of FSMs and the protocol events for remote memory addresses are handled by another set [8]. Within a set, static



| Memory Address | Dispatch Status | | | |
|---|---|---|---|---|
| | Dynamic | Block | Page | Home |
| 0x0000200 | disp FSM 0 | disp FSM 2 | disp FSM 0 | disp FSM 0 |
| 0x2001300 | disp FSM 1 | disp FSM 3 | disp FSM 1 | disp FSM 3 |
| 0x0000200 | *wait* FSM 0 | *wait* FSM 2 | *wait* FSM 0 | *wait* FSM 0 |
| 0x1000300 | disp FSM 2 | *wait* FSM 3 | *wait* FSM 0 | *wait* FSM 3 |
| 0x0000400 | disp FSM 3 | disp FSM 0 | *wait* FSM 0 | *wait* FSM 0 |

**FIGURE 5. Address partitioning example in a system with four FSMs.**

block-interleaved partitioning is used to partition the protocol events among the FSMs. DSM clusters typically implement a global physical address space in which the upper address bits include a home identifier. An address demultiplexor uses the home identifier bits to dispatch a protocol event.

Because, only the home FSMs access the directory and only the remote FSMs access the remote cache, home-based partitioning reduces the sharing and contention in resources by a factor of two. As such, home-based partitioning reduces the hardware complexity of the resources (e.g., obviating the need for multiporting for two-engine designs) and the FSMs managing access to the resources, making this partitioning scheme the least expensive in hardware complexity and cost.

## 3.4 Address Partitioning Example

Figure 5 illustrates the dispatch of an example sequence of protocol events under the different address partitioning schemes for coherence controllers with four FSMs. The figure illustrates global physical address encoding for a system with 256-byte blocks, 4-Kbyte pages, and 1.6 Mbyte of addressable shared memory. The dynamic partitioning scheme dispatches events for memory addresses 0x0000200 and 0x2001300 in parallel. A second event for the memory address 0x0000200 cannot be dispatched due to a previous event for the same memory block and has to wait. But the events for 0x1000300 and 0x0000400 are dispatched to the remaining FSMs.

The example in the figure also indicates the increase in load imbalance in static schemes as compared to the dynamic scheme. Block-interleaved partitioning dispatches events based on the block address bits. The events for memory addresses 0x0000200 and 0x2001300 are dispatched in parallel to FSMs 2 and 3 respectively. A second event for 0x0000200 has to wait because of the previous event for that address. The event for 0x1000300 also has to wait due to the previous event for 0x2001300, because they have the same block address bits. The event for memory address 0x0000400 is dispatched to FSM 0.

Page-interleaved partitioning dispatches events based on the page address bits. The events for addresses 0x0000200 and 0x2001300 are dispatched in parallel. All other events will have to wait because they have the same page address bits as 0x0000200 which is currently being handled.

In home-based partitioning, home events are handled by FSMs 0 and 1 while remote events are handled by FSMs 2 and 3. Between FSMs of the same set, block bits are used to partition the memory addresses. For instance, a home event for an even block address is handled by FSM 0 and a home event from an odd block address is handled by FSM 1. The current node identifier is assumed to be 0. In this example, events for addresses 0x0000200 and 0x2001300 are dispatched in parallel to FSMs 0 and 3 respectively. Other home events for even block addresses, 0x0000200 and 0x0000400, have to wait for FSM 0 to finish handling the current event while the event for 0x1000300, a remote event for an odd memory address, has to wait for FSM 3.

## 4 Performance Results

### 4.1 Methodology

We use Wisconsin Wind Tunnel II [10] to simulate DSM clusters interconnected using multi-engine coherence controllers. Our base system is a 64-processor machine consisting of eight nodes. Each node is an 8-way SMP with 600 MHz dual-issue processors with 1-Mbyte data caches interconnected by a 100-Mhz split-transaction bus. We model a highly-interleaved memory system, characteristic of high-performance SMP servers. A snoopy MOESI coherence protocol keeps the caches within each node consistent. The DSM hardware extends the shared-memory abstraction across the nodes. WWT-II assumes perfect instruction caches but models data caches and their contention at the memory bus accurately. WWT-II further assumes a point-to-point network with a constant latency of 80 cycles but models contention at the network interfaces.

In this paper, we are interested in evaluating DSM clusters with aggressive remote caching and assume a remote cache large enough to fit the entire remote working set of an application [9,3]. In these experiment, communication only consists of true memory sharing among the processors, and therefore our results indicating performance improvement using multi-engine protocols are conservative. Evaluating address partitioning in the context of DSMs with remote capacity/conflict traffic is beyond the scope of this paper.

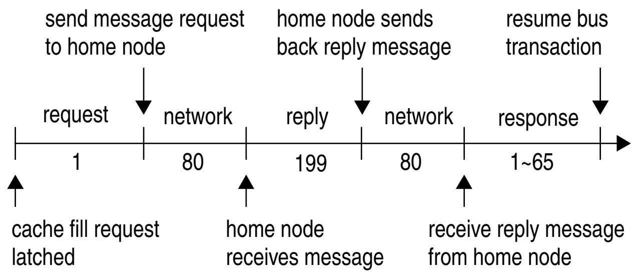

send message request to home node
home node sends back reply message
resume bus transaction
request
network
reply
network
response
1
80
199
80
1~65
cache fill request latched
home node receives message
receive reply message from home node

**FIGURE 6. Event latency on a remote read miss.** *The latencies are given in terms of 600-MHz processor cycles.*

### 4.2 Microbenchmark Experiments

Communication in parallel applications running on a DSM cluster can be either latency-bound or bandwidth-bound. Latency-bound applications do not experience any queueing delays at the FSMs and would only benefit from reduced protocol occupancy and not from multiple coherence engines. Bandwidth-bound applications have bursty protocol events, and as such experience significant queueing delays and would benefit from multi-engine coherence controllers.

To find the latency and bandwidth characteristics of the different address partitioning schemes, we use a set of simple remote read microbenchmarks. Each microbenchmark, consists of a tight loop, in which processors iterate requesting memory blocks (by taking remote read misses) from a physically contiguous set of shared pages. In the first microbenchmark, we evaluate the round-trip latency and the breakdown of protocol event occupancies as measured by a single processor requesting remote data from another DSM node. Figure 6 illustrates the breakdown of protocol events and their associated latencies. There are three distinct phases in the events following a read miss: a request phase on the caching node (requesting node), a reply phase on the home node (replying node), followed by a response phase on the caching node.

The request latency is a single cycle from the time the request is latched in the protocol queue (Figure 1) until the request message is sent out. The reply latency is 199 cycles for the replying FSM to dispatch the event, access the directory, read a 64-byte block from memory, and inject it into the network. The response latency varies depending on whether the remote cache is empty, incurring a single cycle to place the block, or if a dirty block must be replaced from the remote cache, incurring extra latency of up to 64 cycles. The total minimum round-trip latency for a remote read varies between 365 to 425 cycles. Given the occupancies, the maximum achievable request/response and reply bandwidth will be 581 Mbytes/sec and 193 Mbytes/sec respectively with a single FSM per node. Therefore, the reply

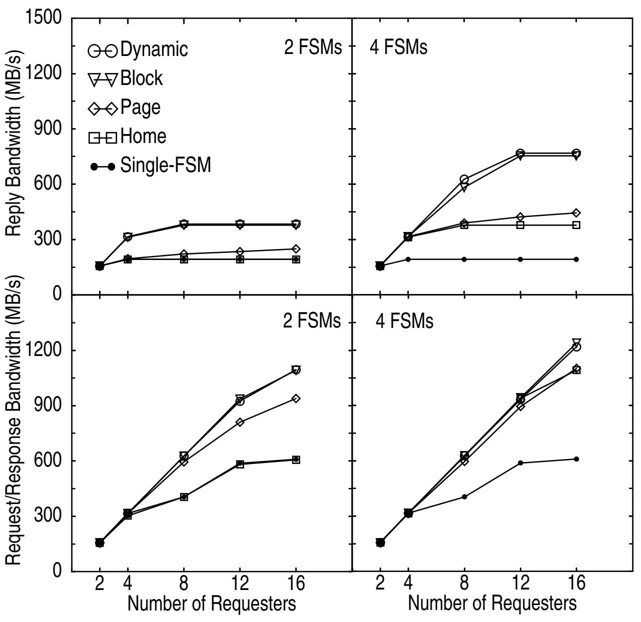


**FIGURE 7. Reply (top) and request/response (bottom) bandwidth of two-FSM (left) and four-FSM (right) systems.**

bandwidth can saturate much faster than the request/response bandwidth.

We use two microbenchmarks to evaluate the bandwidth characteristics of the systems. In one benchmark, we measure the peak reply bandwidth of the various systems by forcing all nodes to request shared data from a single node. In another benchmark, we evaluate the peak request/response bandwidth by forcing each SMP processor from a single node to request shared data from a distinct remote node. In both cases, all memory blocks in a page are accessed in order. While not similar to many real-world applications, these microbenchmarks will help us determine the maximum number of requesters that can be handled efficiently under the different partitioning schemes.

Figure 7 compares the reply (top) and request/response (bottom) bandwidth characteristics of the different address partitioning schemes in systems with two (left) and four (right) FSMs. The reply bandwidth saturates as the number of requesters increases. The saturation bandwidth for a single-FSM system is 192 Mbytes/sec as expected. The request/response bandwidth for a single-FSM system reaches 561 Mbytes/sec, very close to the expected saturation bandwidth.

Figure 7 (top left) illustrates the reply bandwidth results for two-FSM systems. Home-based partitioning only allows a single FSM to handle events for home addresses, resulting in a two-FSM system that behaves exactly like a single-FSM system. Because, all processors march down physically-contiguous pages starting at an even-numbered page in both microbenchmarks, page-interleaved partitioning is also unable to exploit parallelism resulting in bandwidths close to the single-FSM system. With an increase in the number of requestors, however, queueing delays at the replier introduce a time drift in the requests, resulting in an interleaving of requests for odd-/even-numbered pages. The required time drift in requests for odd-/even-numbered blocks is much smaller resulting in a uniform interleaving. As such, block-interleaved partitioning performs on par with dynamic partitioning achieving a saturation bandwidth that is twice that of a single-FSM system.

Figure 7(top right) shows the reply-bandwidth in systems with four FSMs per node. Home-based and page-interleaved partitioning utilize only half of the FSMs and achieve about twice the bandwidth of a single-FSM system. Block-interleaved and dynamic partitioning perform the best achieving nearly four times the bandwidth of a single-FSM system.

Figure 7 (bottom) shows the request/response bandwidth in systems with two and four FSMs. Much as in the case for reply bandwidth, the request/response bandwidth for home-based partitioning with two FSMs behaves identical to a single-FSM system. The request/response bandwidth does not entirely saturate in any of the other schemes with either two or four FSMs. This result indicates that request/response bandwidth is not likely to saturate even with as many as 16 processors per node.

## 4.3 Macrobenchmark Experiments

The microbenchmark results helped identify the key latency and bandwidth characteristics of the different address partitioning schemes for a simple remote read miss. In real applications, however, more complex interactions between the memory system and the protocol result in increased protocol occupancies. As such, our simple request/reply/response model breaks down. Real applications also have irregular memory access stride and sharing patterns resulting in more complex variations in performance gaps among the different address partitioning schemes. In this section, we evaluate performance using shared-memory applications.

We expect the dynamic partitioning scheme to perform best by distributing the protocol handling load evenly among the FSMs. The static partitioning schemes primarily rely on a few address bits in the memory address to do the partition. The performance of each static scheme depends on how evenly the values of the chosen bits are distributed among the protocol events.

In the case of interleave partitioning, the performance primarily depends on the data layout in memory, the memory access strides, the sharing granularity of the algorithm, and the selected interleaving bits. In case of the home-based partitioning, the performance depends on both the application’s sharing patterns and the system page alloca-

| Benchmark | Description | Input Set |
|---|---|---|
| *barnes* | N-body simulation | 16K particles |
| *cholesky* | Sparse factorization | tk29.O |
| *em3d* | 3-D wave propagation | 76K nodes, 15% remote |
| *fft* | Complex radix-$\sqrt{n}$ FFT | 1M points |
| *ocean* | Ocean simulation | 514x514 ocean |
| *radix* | Integer radix sort | 4M integers |

**TABLE 1. Applications and input sets**

tion and placement policy. In this study, we use a round-robin page allocation policy. However, we measured the distribution of remote misses for our applications on the base system configuration and found that with a remote cache large enough to hold the entire remote working set of every node, our allocation policy results in an even distribution of coherence misses.

Table 1 presents the applications we use in this study and the input parameters. *Barnes*, *cholesky*, *fft*, *ocean* and *radix* are from the SPLASH-2 [13] benchmark suite. *Em3d* is a shared-memory implementation of the Split-C benchmark [1]. *Barnes* is primarily latency-bound and does not gain from multi-engine controllers. *Cholesky*, *fft* and *radix* are communication-bound and exhibit poor speedups over a uniprocessor. *Em3d* has a moderate communication-to-computation ratio and achieves 50% efficiency with 64 processors. In the rest of the section, we present performance results normalized to a system with a single FSM per node**.**

### 4.3.1 Base Results

Figure 8 compares the performance of the address partitioning schemes for the base case system. Our base system is a cluster of 8 8-way SMPs. The graph shows that dynamic partitioning achieves a performance improvement of 37% for a two-FSM system and 62% for a four-FSM system as compared to a single-FSM system. Not surprisingly, dynamic partitioning performs better than any of the static partitioning schemes and achieves balanced utilization of the FSMs.

The graph shows that there are three classes of applications. The first class is *barnes*, which is primarily latency-bound and does not benefit from the presence of multiple FSMs. The partitioning scheme has no effect on this application. Latency-bound applications rather would benefit from a lower protocol occupancy which would reduce the round-trip time of a coherence message between nodes.

The second class is *em3d*, which is characterized by an irregular access pattern and a balanced sharing pattern. In

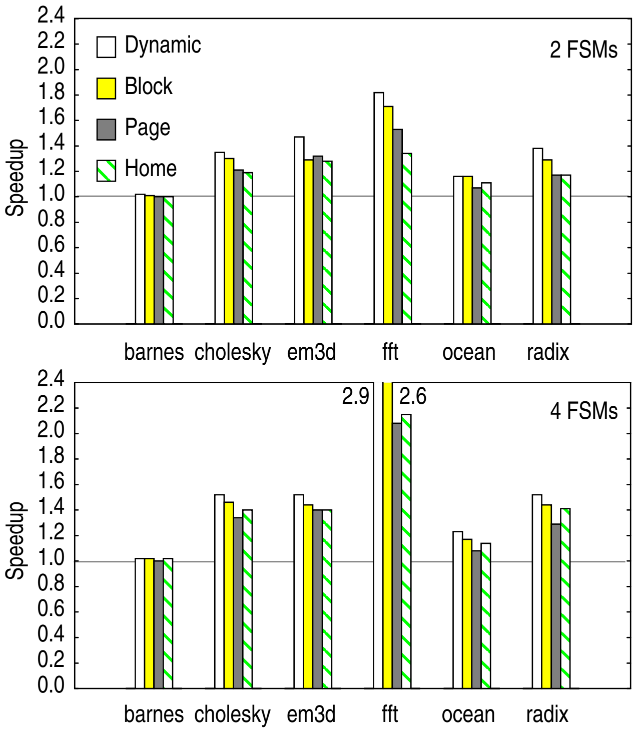


**FIGURE 8. Base performance results.**

*The graphs compare the relative performance of the address partitioning schemes on a cluster of 8 8-way SMPs with two (top) and four (bottom) FSMs per SMP. The graphs plot speedups normalized to a single-FSM system.*

*em3d*, computation iterates over a bipartite graph [1] and exhibits a repetitive but irregular sequence of memory addresses. The figure shows that all static schemes perform almost identically for *em3d*. The performance of the static schemes can never be on par with the dynamic scheme for irregular access strides. This is because, aliasing of addresses in bit-based static schemes results in false dependencies which reduce the available number of independent protocol events. This result suggests that for irregular but repetitive access patterns, a static scheme based on bit hashing may perform closer to the ideal dynamic scheme.

The third class consists of *cholesky*, *fft*, *ocean* and *radix*, which are primarily bandwidth bound. In these applications block-interleaved partitioning performs better than page-interleaved partitioning. This is expected because these are fine-grain shared memory applications and actively share data within a page, reducing the chances of finding independent protocol events in the case of page-interleaved partitioning.

Home-based partitioning generally does not perform as well as interleave partitioning in a system with two FSMs because of the inherent load imbalance problem. In a system with four FSMs, home-based partitioning is really a composite scheme as it uses block-based interleaving within the home or remote FSM set and performs better

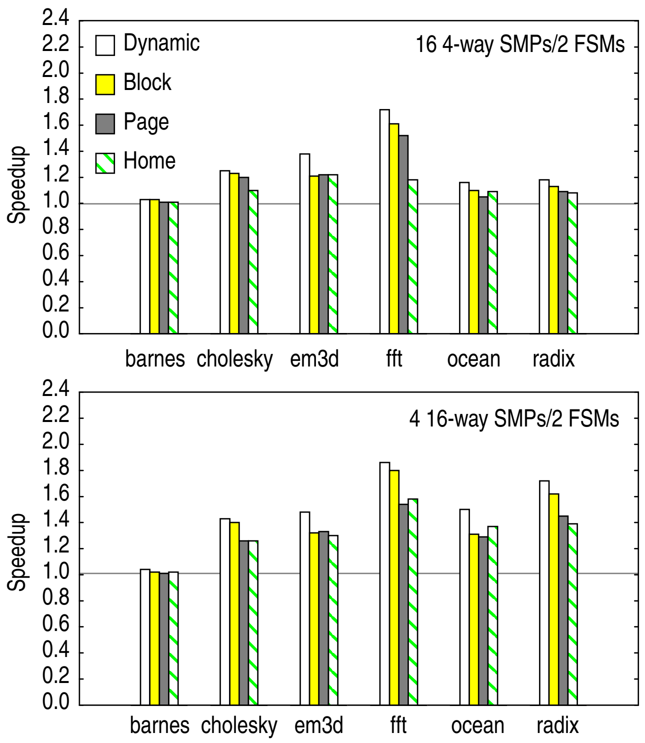


**FIGURE 9. Impact of clustering degree on the performance of systems with two FSMs.**

*The graphs compare performance of the address partitioning schemes for a clustering degree of 4 (top) and 16 (bottom). The speedups are normalized a single-FSM system.*

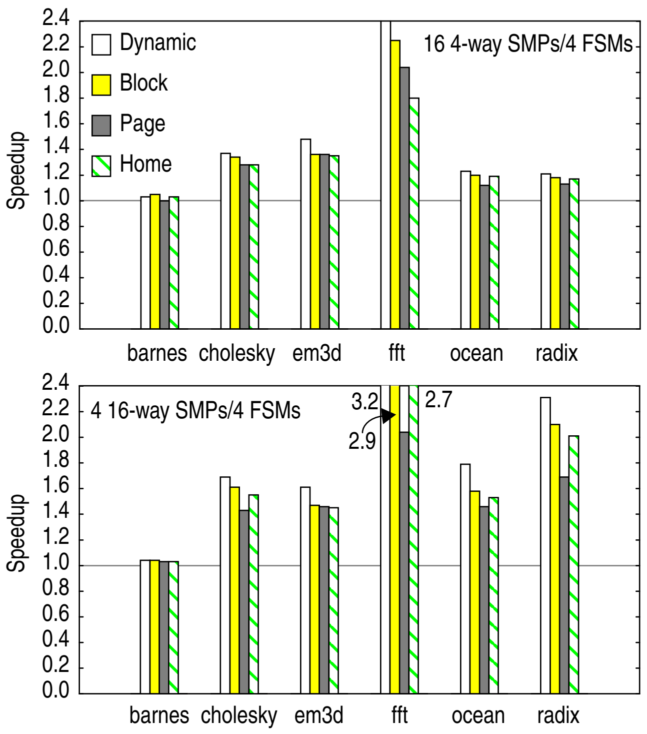


**FIGURE 10. Impact of clustering degree on the performance of systems with four FSMs.**

*The graphs compare performance of the address partitioning schemes for a clustering degree of 4 (top) and 16 (bottom). The speedups are normalized to a single-FSM system.*

than page-interleaved but falls short of block-interleaved partitioning.

On average, block-interleaved partitioning achieves a performance that is 93% of the dynamic scheme. Page-interleaved and home-based partitioning achieve about 85% of dynamic partitioning's performance.

### 4.3.2 Clustering Degree

Clustering degree refers to the number of processors in every SMP node. While increasing the clustering degree does not affect the memory access strides and sharing patterns in applications, it often increases the total amount of traffic per node and results in higher queueing at the DSM boards [12]. Previous research has shown that higher clustering increases the performance improvement of multi-engine controllers over single-engine controllers for software protocol handlers [4]. In this section, we study the impact of clustering degree on the performance of the address partitioning schemes in systems with multiple FSMs while maintaining the number of processors and the total amount of memory in the system constant.

Figure 9 compares the performance of our different address partitioning schemes for a cluster of 16 4-way SMPs (top) and a cluster of 4 16-way SMPs (bottom). The increased coherence traffic increases the benefits of using multi-engine controllers. For a system using dynamic partitioning, the performance improvement from a single-FSM system rises from 29% to 51% as the clustering degree is increased from 4 to 16. With an increase in clustering degree, the pressure on the FSMs is increased due to higher protocol activity. An increase in protocol event arrival rate intensifies queueing at a higher rate in multi-server/multi-queue systems — such as our static partitioning systems — as compared to multi-server/single-queue systems — such as our dynamic partitioning system [6]. As such, the gap between both interleaving systems and dynamic partitioning grows with an increase in clustering degree.

Home-based partitioning's performance, however, improves with an increase in clustering degree. Home-based partitioning performs best when there is a balanced distribution of protocol events for home and remote addresses. A higher clustering degree reduces the number of nodes in the system, resulting in a more balanced distribution of protocol events per node. Due to this balancing effect, home-based partitioning performs relatively better with higher clustering degree. For two-FSM systems, the performance of home-based partitioning improves from 86% to 87% of dynamic, while the average performance of

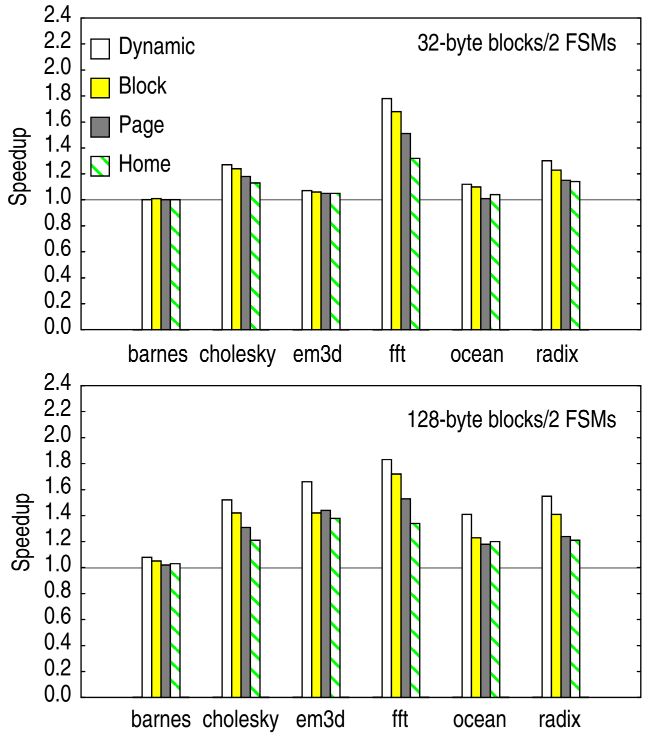


**FIGURE 11. Impact of block size on the performance of systems with two FSMs.**

*The graphs compare performance of the address partitioning schemes for a block size of 32 bytes (top) and 128 bytes (bottom). The speedups are normalized to a single-FSM system.*

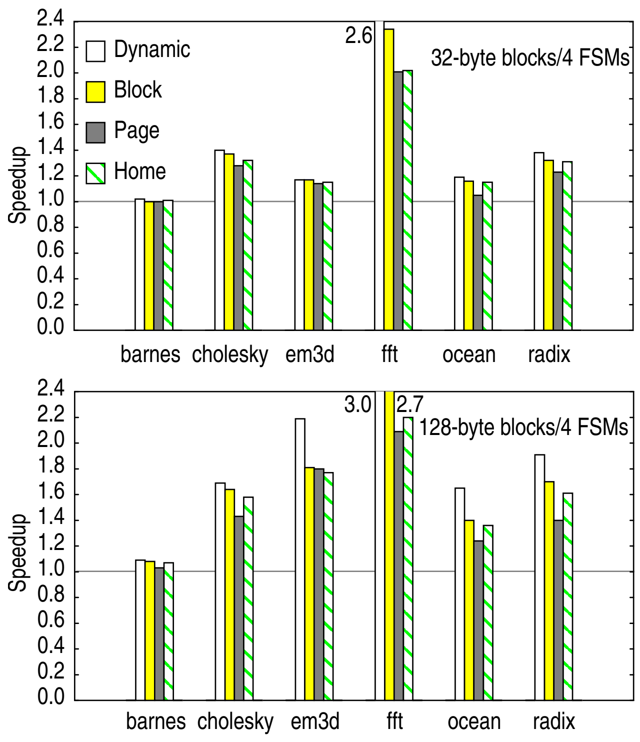


**FIGURE 12. Impact of block size on the performance of systems with four FSMs.**

*The graphs compare performance of the address partitioning schemes for a block size of 32 bytes (top) and 128 bytes (bottom). The speedups are normalized to a single-FSM system.*

the other static partitioning schemes drops from 93% to 90% of dynamic.

Figure 10 shows the impact of clustering degree on a system with four FSMs. For a system using dynamic partitioning, the performance improvement from a single-FSM system more than doubles from 46% to 94% as the clustering degree is increased from 4 to 16. For four FSMs, home-based partitioning drops in performance from 89% to 88% of dynamic while the performance of page-interleaving and block-interleaving drop significantly, from 90% to 78% and 96% to 92% of dynamic respectively.

### 4.3.3 Cache Block Size

An increase in the protocol block size increases the protocol occupancy due to increased data transfer time between memory and the network. It also increases the overall protocol bandwidth out of a node. Depending on the application's sharing patterns, large blocks may either result in false sharing, thereby increasing the number of protocol events, or enable exploiting spatial locality in memory accesses, thereby reducing protocol activity. An increase in protocol activity benefits more from multiple FSMs because the latter reduce queueing.

Figure 11 and Figure 12 illustrate the impact of block size on the performance of address partitioning schemes for a system with two and four FSMs respectively. The speedup using dynamic partitioning of a two-FSM system increases from 26% to 51% while that of a four-FSM system increases from 45% to 92% when the block size is increased from 32 bytes to 128 bytes.

Block size affects both the access stride and the sharing pattern, thereby affecting the interleaving schemes. A lower block size reduces spacial locality and splits up accesses from a processor to memory addresses within a single block, to span over successive blocks. A lower block size removes false sharing and splits up accesses from two different processors to distinct memory locations within a block into accesses to successive blocks. The resulting split enables simultaneous dispatch of multiple events using block-interleaved partitioning. Therefore, block-interleaving performs better for a lower block size. From the graph, the performance of block-interleaved partitioning is 97% of dynamic for 32-byte blocks and falls to 90% of dynamic for 128-byte blocks.

An increase in protocol occupancy due to a larger block size also implies a reduced service rate. From queueing theory, a reduction in service rate, increases the queueing delays exponentially. So even though the speedups compared to a single-FSM system increase, the relative performance of the static schemes compared to the dynamic scheme decreases. The performance of page-interleaved partitioning falls from 91% to 85% of dynamic for two-FSM systems and from 89% to 78% of dynamic for four-FSM systems as the block size is increased from 32 bytes to 128 bytes. The performance of home-based partitioning also falls from 88% to 81% of dynamic for two-FSM systems and from 92% to 83% of dynamic for four-FSM systems.

## 5 Conclusions

In this paper, we evaluated four address partitioning schemes for multi-engine coherence controllers in DSM clusters. Dynamic partitioning best balances the load among multiple coherence engines by synchronizing protocol events directly in hardware at runtime. Block-interleaved partitioning uses low-order bits in the memory block number to select a coherence engine. Similarly, page-interleaved partitioning uses the low-order bits in the memory page number to select an engine. Home-based partitioning reduces hardware complexity most and partitions the addresses between addresses for (local) home pages and those for remote pages. Home-based partitioning also reduces contention on the directory and the remote cache by grouping engines into those accessing the directory and those accessing the remote cache, obviating the need for multi-porting the resources.

We studied the address partitioning schemes by executing shared-memory application on simulated DSM clusters with multi-engine controllers. Our results indicated that: (i) dynamic partitioning performs best eliminating the load imbalance among engines; on average a two-FSM system improved performance by 37% and a four-FSM system by 62% as compared to a single-FSM system in an 8 8-way cluster, (ii) block-interleaved partitioning is an attractive alternative and performs close to dynamic when protocol occupancies are low or there is little queueing; block-interleaved partitioning in a 8 8-way cluster achieves 97% of the performance of dynamic partitioning using 32-byte blocks, and (iii) home-based and page-interleaved partitioning result in the highest load imbalance among multiple engines and a lower performance achieving only about 85% of the performance of dynamic partitioning for the base case system.

## References

[1] D. E. Culler, A. Dusseau, S. C. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. Yelick. Parallel programming in Split-C. In *Proceedings of Supercomputing '93*, pages 262–273, Nov. 1993.

[2] B. Falsafi. *Fine-Grain Protocol Execution Mechanisms & Scheduling Policies on SMP Clusters*. PhD thesis, Computer Sciences Department, University of Wisconsin–Madison, 1998.

[3] B. Falsafi and D. A. Wood. Reactive NUMA: A design for unifying S-COMA and CC-NUMA. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 229–240, June 1997.

[4] B. Falsafi and D. A. Wood. Parallel dispatch queue: A queue-based programming abstraction to parallelize fine-grain communication protocols. In *Proceedings of the Fifth IEEE Symposium on High-Performance Computer Architecture*, Feb. 1999.

[5] E. Hagersten and M. Koster. WildFire: A scalable path for SMPs. In *Proceedings of the Fifth IEEE Symposium on High-Performance Computer Architecture*, pages 172–181, Feb. 1999.

[6] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice Hall, 1984.

[7] T. Lovett and R. Clapp. STiNG: A CC-NUMA compute system for the commercial marketplace. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, May 1996.

[8] M. Michael, A. K. Nanda, B.-H. Lim, and M. L. Scott. Coherence controller architectures for SMP-based CC-NUMA mulitprocessors. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, May 1997.

[9] A. Moga and M. Dubois. The effectiveness of SRAM network caches in clustered DSMs. In *Proceedings of the Fourth IEEE Symposium on High-Performance Computer Architecture*, pages 103–112, February 1998.

[10] S. S. Mukherjee, S. K. Reinhardt, B. Falsafi, M. Litzkow, S. Huss-Lederman, M. D. Hill, J. R. Larus, and D. A. Wood. Fast and portable parallel architecture simulators: Wisconsin Wind Tunnel II. *IEEE Concurrency*, 2000. To appear.

[11] A. K. Nanda, A.-T. Nguyen, M. M. Michael, and D. J. Joseph. High-throughput coherence controllers. In *Proceedings of the Sixth IEEE Symposium on High-Performance Computer Architecture*, Jan. 2000.

[12] I. Schoinas, B. Falsafi, M. D. Hill, J. Larus, and D. A. Wood. Sirocco: Cost-effective fine-grain distributed shared memory. In *Proceedings of the Sixth International Conference on Parallel Architectures and Compilation Techniques*, October 1998.

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