Coherence Domain Restriction on Large Scale Systems

Yaosheng Fu
Princeton University
Princeton, NJ
yfu@princeton.edu

Tri M. Nguyen
Princeton University
Princeton, NJ
trin@princeton.edu

David Wentzlaff
Princeton University
Princeton, NJ
wentzlaf@princeton.edu

ABSTRACT

Designing massive scale cache coherence systems has been an elusive goal. Whether it be on large-scale GPUs, future thousand-core chips, or across million-core warehouse scale computers, having shared memory, even to a limited extent, improves programmability. This work sidesteps the traditional challenges of creating massively scalable cache coherence by restricting coherence to flexible subsets (domains) of a system's total cores and home nodes. This paper proposes Coherence Domain Restriction (CDR), a novel coherence framework that enables the creation of thousand to million core systems that use shared memory while maintaining low storage and energy overhead. Inspired by the observation that the majority of cache lines are only shared by a subset of cores either due to limited application parallelism or limited page sharing, CDR restricts the coherence domain from global cache coherence to VM-level, application-level, or page-level. We explore two types of restriction, one which limits the total number of sharers that can access a coherence domain and one which limits the number and location of home nodes that partake in a coherence domain. Each independent coherence domain only tracks the cores in its domain instead of the whole system, thereby removing the need for a coherence scheme built on top of CDR to scale. Sharer Restriction achieves constant storage overhead as core count increases while Home Restriction provides localized communication enabling higher performance. Unlike previous systems, CDR is flexible and does not restrict the location of the home nodes or sharers within a domain. We evaluate CDR in the context of a 1024-core chip and in the novel application of shared memory to a 1,000,000-core warehouse scale computer. Sharer Restriction results in significant area savings, while Home Restriction in the 1024-core chip and 1,000,000-core system increases performance

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

MICRO-48, December 05 - 09, 2015, Waikiki, HI, USA

Copyright is held by the owner/author(s). Publication rights licensed to ACM.

ACM 978-1-4503-4034-2/15/12 ...\$15.00

DOI: <http://dx.doi.org/10.1145/2830772.2830832>

by 29% and 23.04x respectively when comparing with global home placement. We implemented the entire CDR framework in a 25-core processor taped out in IBM's 32nm SOI process and present a detailed area characterization.

Categories and Subject Descriptors

B.3.2 [Memory Structures]: Shared Memory

Keywords

Shared Memory, Cache Coherence, Home Placement

1. INTRODUCTION

The number of cores integrated within a single many-core chip, GPU, and even data center continues to increase. Programming systems with shared memory has been popular due to the ease of programming, similarity to single-threaded programming, easy mapping of certain classes of algorithm, and the widespread availability of applications based on POSIX Threads [1] and OpenMP [2]. Unfortunately, creating massively scalable cache coherence protocols has proven problematic. In this work, we address the challenge of massively (Thousands to Millions of caches) scalable cache coherence not by creating a new cache coherence scheme, but rather by creating a hardware framework that can be used by other coherence schemes to sidestep the challenges of scaling directory storage and communication cost. We study this framework in two primary use cases: future thousand-core chips and current day warehouse scale computers with shared memory. In addition, we see our solution being applicable to future GPU systems which are adopting shared memory.

As we advance to the future manycore era with hundreds or thousands of cores on a single chip [3, 4, 5, 6, 7, 8], the directory which tracks sharers in directory-based cache coherence systems presents a storage scalability challenge. The quadratic overhead of full-map directories [9] makes it unsuitable for manycore designs. For example, assuming a 64-byte cache line, the overhead for a full-map directory in a 64-core system is 12.5%, but this grows to 200% for a 1024-core system with the bookkeeping requiring twice the storage of the data. Much effort has gone into alleviating the directory storage overhead [10, 11, 12, 13, 14, 15]. Most of these efforts focused on trading sharing pattern storage for additional time needed to transition the coherence state

of a cache line. Unfortunately, a cache coherence protocol which can scale to a very large number of caches while having efficient cache state transition time and sharing meta-data storage has yet to be discovered.

The emergence of large data centers along with warehouse scale computers [16] has driven the development of applications such as whole web search, large-scale data mining, and hosted web applications. Contemporary data centers scale to hundreds of thousands of servers which collectively have millions of cores. The typical data center only has rigid pockets of cache coherence, usually within a single server of a few cores, requiring programmers to use a dual-programming environment where shared memory is used within a server while messaging is used between servers. In contrast, our work is the foundation for economically expanding shared memory across current day data centers. We introduce *warehouse scale shared memory* where limited shared memory can flexibly span the data center containing millions of cores. Such a design can be used to break down the boundaries that exist in current day data centers where virtual machines do not span servers, applications do not share memory across the data center, and programmers need to use a hybrid, shared memory along with messaging programming environment. In this paper we focus on addressing the scalability issue of warehouse scale shared memory. There are other important challenges for warehouse scale shared memory besides the scalability such as fault-tolerance, resource management and software infrastructure. Since implementing shared memory in the warehouse scale is a radical change and requires significant modification and re-design in both hardware and software, we do not expect to address all those issues in this paper and leave them to future work.

In this paper, we propose Coherence Domain Restriction (CDR), a novel coherence framework that is capable of enabling systems to scale to thousands or millions of cores, while keeping constant storage overhead and high performance. CDR is inspired by the observation that today’s shared memory applications are limited in parallelism by Amdahl’s law and serializing critical sections [17, 18, 19, 20], therefore causing them to use only a limited number of cores (but still larger than the core count of most current processors). While the total scalability of a shared memory application may be limited, this does not mean that the total number of cores in a system, total capacity of sharable memory, placement of sharers or home nodes, or the ability to quickly and flexibly rearrange the cores which are sharing memory should be restricted. We investigate two types of Coherence Domain Restriction: Sharer Restriction and Home Restriction. Sharer Restriction limits the number of sharers that can be tracked for a given coherence domain while Home Restriction limits the number and location of home nodes in a particular coherence domain. These two ideas are decoupled and can be used independently or together as we will show.

Sharer Restriction significantly reduces the storage overhead by selectively tracking cache line sharers on a

Figure 1: V-SR: Three applications run on (a) a many-core chip or (b) a warehouse scale shared memory computer of 4-core machines, each requiring 6, 4, and 16 cores respectively.

per-VM, per-application, or per-page basis rather than across all cores. Sharer Restriction is not a coherence scheme itself, but rather provides a mapping layer to decouple physical cores from coherence schemes. It is compatible with and beneficial to most existing coherence schemes. In Sharer Restriction, coherence protocols only operate on a small set of logical sharers which then get mapped into a much larger space of physical cores. Therefore, the selected coherence protocol itself does not need to scale to the entire system.

In contrast, Home Restriction places directory information (home nodes) near the cores that use it thereby reducing the network communication distance and energy. In addition, Home Restriction can alleviate hotspots at directory controllers. Home Restriction can act on either VM/application-level or page-level. Page-level Home Restriction leads to better communication locality than VM/application-level because of the restriction at a finer granularity.

CDR builds upon ideas from previous work which attempted to restrict coherence domains either statically such as in Scale-Out processors [21] and the Intel SCC [7] or dynamically like in Virtual Hierarchies [22]. CDR extends the static coherence domain work by breaking down the restrictions that rigid coherence boundaries have. While restricting homes has been investigated in Virtual Hierarchies, we extend this idea further by enabling dynamic resizing of home domains at runtime. We also evaluate how to utilize dynamic coherence domains in 1000+ core chips and warehouse scale shared memory systems which were not considered in previous work. Some major differences in our work are that CDR addresses the scalability of directory storage, the creation of page-level domains, and runtime modification of domain size, all of which Virtual Hierarchies did not address.

The following are our key contributions:

- The Sharer Restriction framework and how it achieves constant storage overhead with core scaling.
- The Home Restriction framework and how it increases performance and decreases communication energy by localizing communication between cores.
- Investigation of coarse-grain VM/application-level and fine-grain page-level as granularities for both Sharer Restriction and Home Restriction.

- Discussion of the software-hardware interface and system software (OS/Hypervisor) techniques to efficiently support CDR.
- We evaluate CDR in the context of a future, tiled, 1024-core chip and a 100,000-machine (10 cores per machine) warehouse scale system. We find that Sharer Restriction achieves significant area savings with 0.3% performance loss on average and that Home Restriction speeds up the performance by 29% in the former system and 23.04x in the latter system versus global home placement.
- We implement the entire CDR framework in a 25-core Processor taped out in IBM's 32nm SOI process and provide a detailed area analysis.

2. COHERENCE DOMAIN RESTRICTION

Coherence Domain Restriction (CDR) is a novel cache coherence framework that enables shared memory in systems that scale to thousands or even millions of cores. The key insight of CDR is that instead of keeping a single global cache coherence domain, we provide flexible fine-grain coherence domains for the subset of cores that are used by a VM/application or physical page.

CDR leverages two independent techniques: Sharer Restriction (SR) and Home Restriction (HR). The basic idea of Sharer Restriction is to restrict the sharer domain of coherence protocols to a subset of cores accessed by a VM, application, or page. By doing this, we can greatly reduce the hardware overhead introduced by tracking all cores across the system. Home Restriction applies a similar idea by restricting the home domain of a VM, application, or page to be within a subset of home nodes to provide localized communication between request caches and home nodes.

2.1 Sharer Restriction

Figure 1 introduces the basic idea of VM/application-level Sharer Restriction (V-SR) applied to a manycore chip (1a) and a warehouse scale computer with shared memory (1b). Each application runs only on a small subset of cores; thus it is not necessary to track sharers beyond those cores. V-SR provides isolation between physical cores and the coherence scheme by mapping logical sharers used in the coherence scheme into physical cores in the system. Therefore, the coherence scheme only needs to keep track of a small number of logical sharers that get mapped into the subset of physical cores used by the application. The mapping mechanism can be re-configured at runtime to add new cores into or remove old cores from the sharer domain. The cores in a sharer domain **do not need to be contiguous**, but rather can be any set of cores in the system. Moreover, sharer domains can overlap on the same physical core without affecting each other.

We further explore Sharer Restriction at a finer granularity at the page-level which we call page-level Sharer Restriction (P-SR). In P-SR, each physical page shared by a subset of cores can form its own sharer domain. In

Schemes	Full-map	Coarse Vector (2:1)	Limited pointers (4 ptrs)
No SR	$O(n)$	$O(n)$	$O(\log(n))$
SR	$O(s) = O(1)$	$O(s) = O(1)$	$O(\log(s)) = O(1)$
No SR	1024 bits	512 bits	40 bits
SR	64 bits	32 bits	24 bits
No SR	100,000 bits	50,000 bits	68 bits
SR	8 bits	4 bits	12 bits

Table 1: Asymptotic storage cost (top) for coherence schemes with and without Sharer Restriction. Storage comparison of sharer list size for a 1024-core chip (middle) with max sharer domain of 64 cores and a 100,000-machine warehouse scale shared memory cluster (bottom) with max sharer domain of 8 machines.

effect, this brings the benefit of even smaller sharer domains especially for private pages which only have one sharer. Page-level granularity provides a convenient location to manage the mapping of information due to widespread use of page tables and TLBs.

In order to take advantage of fine-grain sharer domains, we purposefully limit the maximum sharer domain size (s) which dictates the storage overhead of cache coherence in the system. This number is independent of the total number of cores (n). Rather it is set to track the total expected number of cores that will need to share data, i.e. the maximum application or page scalability. Depending on the coherence scheme used, the number of bits needed to store the sharer list changes. For instance, in a full-map directory [9], s bits are needed per cache line in the directory in contrast to n bits being needed if the full map directory can address the entire system. Because the maximum sharer domain (s) is set independently of the total number of cores (n) and can stay constant as system size grows, we treat it as a constant in asymptotic analysis. Other coherence schemes also see storage savings when combined with Sharer Restriction. For instance, the storage analysis for coarse vector and limited pointers schemes with and without Sharer Restriction is shown in Table 1.

The maximum sharer domain size should be judiciously chosen such that the number of sharers stays below it. As evidenced from numerous PARSEC studies [23, 18], typical applications today have limited scalability, on the order of tens of cores. Many shared memory programs only explore simple communication patterns such as producer-consumer or nearest-neighbor communication [24], which results in a very small number of sharers for each page. In the results section, we choose 64 cores as a conservative maximum sharer domain size for a manycore chip and 8 machines for a warehouse scale computer (each machine contains 10 cores with on-chip coherence, but SR is applied only on the inter-machine coherence scheme). In the rare case that the domain does expand beyond the maximum limit, the directory scheme is switched to inaccurate tracking or broadcast on a per-line basis. The details are explained later in Section 3.1.3.

2.2 Home Restriction

On shared memory systems with directory protocols, each directory access results in a round-trip message

Figure 2: NoC communication shows a cache miss which requires a remote invalidation. On the left, the home node is distant even though the core that needs to be invalidated is local. On the right, Home Restriction is used to localize home nodes and invalidations.

communication between the request node and the home node (or directory) if they are not the same. In addition, if cache line invalidations are needed, additional round trip communication is required. In order to map an address to a home node (or directory) in a directory protocol, it is typical to interleave the home location of cache lines across the entire system, evenly distributing the load. The downside to such a solution is that when cache misses occur, they may need to communicate with the home node (or directory) on the other side of the system, incurring large communication cost.

In order to reduce the average communication distance, Home Restriction (HR) restricts the home domain of a VM/application or physical page to be within a subset of home nodes (or directories) for shorter communication distance. HR uses a mapping mechanism that maps logical home locations to physical home locations. The simplest mapping scheme maps the home locations across the same nodes that the VM/application or page is running on, thereby localizing request cache to home node communication versus being spread across the system. This implies that the home domain and sharer domain will be the same if both SR and HR are applied. Similarly, the home domain is adjusted at runtime to match the subset of cores used by a VM/application or page. Nevertheless, **home nodes and request nodes do not have to be the same** and home nodes can be mapped to arbitrary locations. More advanced mapping strategies can be adopted to locate home nodes at better locations to minimize communication cost. For example, if the request nodes are distributed, the best home node location may be in the center-of-mass of the communicating cores which may not be a request node. Similar to SR, HR limits the maximum home domain size (h) to be constant.

In general, the asymptotic worst-case communication cost without HR on a manycore chip with 2D mesh on-chip network is $O(\sqrt{n})$, while with HR, assuming the cores in an application or utilizing a page are co-located, it is $O(\sqrt{h}) = O(1)$ with a maximum home domain size (h). An example of this can be seen in Figure 2, in which a network request results in a round trip distance

of 50 hops without HR, while only 6 hops are required with HR since both request nodes and home nodes are located in the same subset of cores. In warehouse scale computers, HR can achieve even better performance benefits for two reasons: 1) inter-machine communication latency is much longer than on-chip communication and 2) warehouse scale computers usually contain a much larger number of homes.

Home Restriction can be applied either at the page-level (P-HR) or VM/application-level (V-HR). Page-level Home Restriction (P-HR) achieves better communication locality than VM/application-level Home Restriction (V-HR) at the cost of a larger number of concurrent home domains. P-HR is ideal for private pages as no network traffic is generated because the request node and home node (or directory) are co-located.

3. IMPLEMENTATION

The key design of the CDR framework is the mapping between logical cores and physical cores. By adding a flexible mapping layer that abstracts logical cores from physical cores, we expose a much smaller logical domain space to coherence protocols and homing policies. One primary challenge with CDR is how to translate logical cores efficiently to physical cores and vice versa.

3.1 Implementation of Sharer Restriction

In our implementation of Sharer Restriction, each sharer domain is assigned a unique sharer domain ID (SDID) and each physical core in the domain is assigned a logical sharer ID (LSID). Within each sharer domain, logical sharer IDs are exposed to the coherence protocol instead of actual physical core IDs, and they are then translated back to physical core IDs for network routing when coherence messages are sent to those sharers. Since the number of logical sharers in a domain is limited by the maximum domain size, each logical sharer can be represented with fewer bits in the directory compared with directly storing the physical core ID. The coherence protocol is effectively dealing with a smaller system with a limited number of sharers. A physical core can participate in multiple sharer domains with multiple pairs of sharer domain ID and logical sharer ID, so different sharer domains can overlap on the same physical cores. A physical core is uniquely determined given a pair of sharer domain ID and logical sharer ID. We utilize an OS-maintained sharer map table (SMT) to translate logical sharer IDs to physical core IDs. The table is cached for fast look-up using sharer map caches (SMC), similar to a TLB caching a page table.

3.1.1 Hardware Modification for Sharer Restriction

Figure 3a shows the implementation details of V-SR at the request node. We add an extra register to store the current pair of sharer domain ID and logical sharer ID which is updated upon context switches. When the core needs to send any coherence message to the home, the ID pair is appended. Figure 3b shows the implementation for P-SR. Instead of having a special register that is swapped out on every context switch, the ID

Figure 3: Architectural implementation of the request node for (a) VM/Application-level Sharer Restriction (V-SR) and (b) Page-level Sharer Restriction (P-SR).

Figure 4: Architectural implementation of the home node with Sharer Restriction.

pairs are stored in TLB entries associated with physical pages. Each time a TLB access occurs, the ID pair is read out along with the physical page number. The hardware needed for P-SR is a superset of V-SR: we can emulate V-SR by setting all physical pages of a VM/application to have the same sharer domain ID.

Figure 4 shows the modification at the home node (or directory) in order to support SR. As mentioned above, coherence messages destined for the home node (or directory) contain the sharer domain ID and local sharer ID which are exposed to the coherence protocol instead of the physical core ID. For example, in a 1024-core chip with full-map directory scheme and maximum share domain of 64 cores, each directory entry only needs to maintain a sharer vector of 64 bits representing local sharer IDs from 0 to 63. If invalidation or downgrade of sharers is needed, logical sharer IDs for those sharers along with the sharer domain ID are fed into the sharer map cache to look-up the corresponding physical core IDs. Otherwise, no access to the sharer map cache is needed. Accesses to the sharer map cache can be pipelined to reduce the performance impact when sending out multiple invalidation messages for a widely shared cache line.

A special case for Sharer Restriction is self-eviction of dirty lines from a write-back cache, which seems to be problematic because the cache line being evicted does not contain the information about its sharer domain ID and local sharer ID. However, this is easily solved because at the home node (or directory), processing an evicted dirty line does not need the sharer domain ID or local sharer ID because the line is exclusively owned and the owner is known.

3.1.2 VM/Application-level and Page-level Sharer Restriction Interaction

Figure 5: An inter-domain communication example.

Considering that V-SR already results in constant directory storage, it is usually not worthwhile to further restrict sharer domains to the page level at the cost of more complex domain management and larger sharer map cache size. However, P-SR does provide more flexibility than V-SR. For instance, V-SR does not allow shared data between two VMs/applications because there is no global cache coherence. But there is no such constraint for P-SR. P-SR operates on physical pages rather than virtual pages; therefore a page shared by two VMs/applications naturally forms a unique domain. As already mentioned, a P-SR implementation can fully emulate V-SR, so V-SR and P-SR can co-exist in one system with the hardware support for P-SR.

An example implementation of combining V-SR and P-SR is shown in Figure 5. In the beginning, only V-SR is used (emulated by P-SR hardware) and all pages in the same VM/application are assigned with the same sharer domain ID. When inter-domain communication is needed, a shared physical page is allocated mapping to two virtual pages in two VMs/applications similar to inter-process communication. The shared physical page is switched from V-SR to P-SR by assigning a new domain ID to it, but all other non-shared physical pages maintain the previous domain ID.

3.1.3 Runtime Modification of Sharer Domains

Over the lifetime of a program, the number of sharers in a domain might change. For example, a VM/application might have only one core during a startup or I/O phase, but when the program transitions to a computationally intensive phase, the number of cores may grow to the maximum available. In order to track all sharers, the sharer domain needs to be adjusted at runtime. When a VM/application finishes, all relevant private caches lines as well as sharer map cache lines are flushed, and the sharer domain IDs are recycled. By flushing all relevant cache lines, we avoid losing sharer information when sharer domains are recycled.

Adding a new sharer into a sharer domain is very simple: just assign a new pair of sharer domain ID and logical sharer ID to the new sharer and update the sharer map table. Removing an existing sharer requires additional invalidation of all relevant cache lines on this sharer core, but it is unnecessary to remove sharers unless the maximum domain size is reached. We expect most sharer domains to stay within the limit of maximum domain size, but if a domain does need to scale beyond the limit, a simple solution is to stop accurately tracking additional sharers and switch to broad-

Figure 6: Architectural implementation of the request node for (a) VM/Application-level Home Restriction (V-HR) and (b) Page-level Home Restriction (P-HR).

cast or alternative inaccurate directory schemes such as coarse vectors on a per-line basis during an invalidation. Broadcast or inaccurate tracking are only required for cache lines that contain new sharers beyond the tracking scope because there is no extra bit in the directory entry to keep track of them, other lines still have accurate sharer information thereby do not need to switch to broadcast or inaccurate directory schemes. We adopt the broadcast scheme in the 25-core processor to simplify the hardware design.

3.2 Implementation of Home Restriction

Home Restriction (HR) decouples logical home nodes (or directories) from physical homes (or directories) by providing a mapping layer between them. Similar to Sharer Restriction, we maintain a home domain ID (HDID) for each home domain and a logical home ID (LHID) for each home node inside a home domain. Homing policies only generate logical home IDs and the actual mapping from logical home IDs to physical home IDs is managed by HR. This enables the OS to cooperate with home policies and optimize home locations for better performance. Home map caches (HMC) that are backed by a home map table (HMT) are adopted to translate logical home IDs to physical ones.

3.2.1 Hardware Modification for Home Restriction

Home Restriction only requires hardware modification to the request node/CPU/core. Figure 6a and 6b show the implementation of the V-HR and P-HR frameworks, respectively. Similar to Sharer Restriction, the home domain ID for V-HR is stored in a special register while in P-HR it is stored in the TLB. The domain core count (current size of the domain) is also kept to determine home mapping. The home allocation unit is a combinational logic block that does home allocation for home domains. It takes the lower-order bits of the physical address (above the cache line offset) along with the domain core count to compute the logical home ID. This logical home ID along with the home domain ID

Figure 7: Home Allocation

indexes the home map cache to determine the physical core ID of the home. The home allocation unit and home map cache only need to be accessed upon cache misses on the request node. Besides, the extra access latency can be overlapped with the data cache pipeline.

The purpose of the home allocation unit is to distribute physical addresses as evenly as possible across all home nodes for arbitrary domain core count. For a given domain core count, the mapping rule from addresses to homes is fixed. Figure 7a shows an example of home allocation for a home domain that grows from 1 to 5 cores (or directories). We can see that each time a new home node is added, half of the address space managed by a single previous home is reallocated to it, while all other homes remain unchanged. The detailed re-homing process is explained later in Section 3.2.2. By reducing the number of existing home nodes that need to be re-homed, the growth of a home domain is faster. Removing an existing home node which has the largest logical home ID in the domain is exactly the reverse process of adding a new one. Otherwise this home node swaps its logical home ID with the one with the largest logical home ID first before the reverse process. Besides, the address spaces assigned to both home nodes need to be re-distributed to match the home allocation rule. Therefore, two home nodes need to be modified instead of one in this case. Removing a home has higher cost than adding a home in general, but it is not necessary in most cases. The address space is evenly distributed when domain core count is a power of two. The detailed home allocation algorithm is described in Figure 7b and the logic is easy to implement in hardware.

Eviction of dirty data is interesting for Home Restriction. The cache needs to determine the appropriate home for the dirty, evicted cache line. Unfortunately, in a standard cache, neither the home location nor home domain ID is known. We propose several solutions. First, we can store the home domain ID in the tag or data array of the request node's private cache. This is at the expense of every cache line having additional information stored in it. A second solution which works for V-HR is to write-back dirty data when there is a context switch to an application with a different home domain ID. Conveniently this does not affect thread switches. A third solution is to simply use write-through caches.

We have implemented the first solution in hardware and evaluated it in the results section in order to support both V-HR and P-HR with write-back caches.

3.2.2 Runtime Modification of Home Domains

Home domains can be modified at runtime to minimize communication distance or balance directory request load across different home nodes. As mentioned before in Section 2.2, we choose a simple home mapping scheme that maps home nodes to the same nodes that the VM/application or the page is running on. This scheme sets the home domain to be the same as the sharer domain if SR is also adopted. Therefore, home domains also need to be adjusted at runtime as VM/applications/pages expand or shrink. For V-HR, domains do not frequently change size as the size is linked to the number of cores used in a VM/program. In contrast, P-HR requires potentially frequent domain size change as the number of cores accessing a shared page changes. Modifying home domains can bring a performance benefit but also causes the sharing information in the directories to be out of date and incorrectly distributed due to re-allocation of home nodes. In order to add or remove a home node to a pre-existing home domain, the system software has to guarantee that the directories contain the updated sharing information. Our home allocation algorithm in Figure 7b guarantees that at most two of the existing homes are affected for each modification; others remain unchanged.

The runtime modification for V-HR requires freezing all relevant cores and updating information, but this is not necessary for P-HR modification as shown below:

1. Invalidate the TLB entry for the page in all relevant cores and set the page as poisoned, which means TLB misses for this page will be delayed until the poisoned state is cleared.
2. Flush relevant directory entries and corresponding cache lines in the to-be-modified homes. Given that the subset of addresses that need to be re-homed are known, we can selectively flush them from the directory without affecting other entries. For example, if a home domain grows from 16 cores to 17 cores, we only need to flush 1/32 of a page from logical home 0.
3. Update the home map table and invalidate the corresponding entries in relevant home map caches.
4. Clear the poisoned state of the page.

3.2.3 Home Domain Merging

Modern applications may access a large number of pages, which creates a large amount of home domains if P-HR is used. This introduces pressure on both software and hardware management, and could potentially lead to significant overhead. In order to reduce the total number of home domains, we adopt domain merging at runtime to merge domains with the same mapping. When a new page is allocated or a P-HR page expands to an additional core, we merge it into an existing domain if they are accessed by the same set of cores and

have the same mapping from logical homes to physical homes. A previous study [24] has shown that for modern workloads most of the shared data is either accessed by all cores or a small subset of cores. Based on this observation, we further merge all pages whose domain size grows beyond a threshold into a single home domain containing all cores used by the application. We choose 3 as the merging threshold during our evaluation. Simulation results in Section 4.4.2 show that after home domain merging, the total number of home domains is greatly reduced while still achieving good performance.

3.3 OS Management of the CDR Framework

Based on the above implementation details, we see that SR and HR have a similar software and hardware flow. Therefore, sharer domains and home domains can be managed similarly. For V-SR or V-HR, the OS maintains a table for mapping VMs/applications to domains. This table is read out upon each context switch between VMs/applications. At the page-level (P-SR, P-HR), the mapping information is stored along with page table entries and is filled into the TLB on a TLB miss.

The OS also needs to maintain the sharer/home map tables which provide the mapping from logical IDs to physical IDs. These tables can be statically allocated to a piece of memory with fixed size. For a 1024-core chip with maximum domain size of 64, each domain requires 64 entries, each of which is 2 bytes (10 bits for the physical core ID and 6 bits for meta-data). Therefore, each domain only needs 128 bytes of memory. If the OS limits the maximum number of sharer/home domains to be 16384, the total memory requirement for the sharer map table and home map table is 4MB. A warehouse scale computer may require a larger number of domains, but the sharer/home map table can be distributed across machines to amortize the storage overhead. Since the sharer/home map tables are pre-allocated with fixed size, it is very convenient to support hardware refill of sharer/home map caches without triggering interrupt handlers from the core. Upon a sharer/home map cache miss, the refill address can be calculated based on the missed domain ID and logical ID, so a single memory look-up is required for the refill.

The OS itself can also be fit into the CDR framework by assigning one or more pre-allocated domains to it. Managing the memory for the OS is an interesting use case for CDR. Since we do not maintain global cache coherence for Sharer Restriction, it is likely not efficient to run an OS that utilizes a large amount of global shared memory as SR switches to inaccurate tracking or broadcast when there are more sharers than that can be accurately tracked. However, there are many examples of distributed OSes that do not require global shared memory such as Barrelfish [25], fos [26, 27], Exokernel [28], Corey [29], and Disco [30, 31]. Home Restriction can be easily turned off for a traditional OS that wants to use all directories without affecting the correctness.

4. EVALUATION

In this section we first present the area overhead of

Processor	1024 cores, x86-64 ISA, in-order, IPC=1 except for memory accesses, one thread per core, 2GHz
L1 Caches	8KB, private, 4-way set associative, 1-cycle latency
L2 Caches	64 KB, private, 4-way set associative, 5-cycle latency
Directory Caches	Distributed shared by all cores, no backup storage, 4-way set associative, 5-cycle latency
SMC/HMC	256 entries, direct mapped, 1-cycle hit, 100-cycle miss
NoC	Tiled 32×32 2D mesh network, X-Y routing, 64-bit flits and links, 1 cycle per hop
Coherence Protocol	Directory-based MESI protocol, homing by cache line interleaving
Memory	100-cycle latency

(a) The manycore chip

System	100,000 machines with a 10-core processor each
Processor	10 cores, x86-64 ISA, in-order, IPC=1 except for memory accesses, one thread per core, 2.5GHz
L1 Caches	32KB, private, 8-way set associative, 1-cycle latency
L2 Caches	256 KB, private, 8-way set associative, 5-cycle latency
L3 Caches	25 MB, shared by all cores, 24-way set associative, 10-cycle latency
Directory Caches	Distributed shared by all machines, no backup storage, 24-way set associative, 10-cycle latency
SMC/HMC	256 entries, direct mapped, 1-cycle hit, 120-cycle miss
Inter-node Network	3D mesh network, X-Y-Z routing, 16GB/s each direction, 200 cycles per hop
Inter-node Protocol	Directory-based MESI protocol, homing by page interleaving for the baseline and cache line interleaving for other schemes
Memory	120-cycle latency

(b) The warehouse scale computer with shared memory

Table 2: Main parameters of modeled systems

implementing Coherence Domain Restriction (CDR), and then evaluate Sharer Restriction and Home Restriction separately. For SR, we present an analysis of storage savings and the performance impact of SR. For HR, we simulate multi-threaded workloads on both a manycore chip and a warehouse scale computer with shared memory. We then measure the performance benefits of HR over other frameworks.

4.1 Methodology

We use the PriME simulator [32] to simulate modeled systems, as detailed in Table 2. For the manycore system described in Table 2a, each core adopts a simple in-order architecture with x86-64 ISA, running at 2GHz. L1 and L2 are private caches, with 8KB and 64KB storage space respectively. A directory-based MESI protocol is maintained through distributed shared directory caches. Each directory cache contains twice as many directory entries as L2 cache lines. We assume 1-cycle extra latency for accessing the sharer/home map cache although the latency can be fully overlapped in many pipelined designs. Every cache miss in the sharer/home map cache takes one memory look-up which is 100 cycles in the modeled system. Of course, a technology with a smaller feature size is required in order to actually fit 1024 cores on a chip. The warehouse scale computer contains 1,000,000 cores in total across 100,000 machines. It is described in Table 2b in which CDR is applied in the inter-machine coherence protocol operating on a per-machine basis. The processors are modeled after the Intel Xeon E5-2670v2 [33].

The three frameworks that we evaluate CDR against are global home placement (Global-home), Scale-Out processors [21] (Scale-out) and Virtual Hierarchies [22]

(a) The area breakdown of a core. (b) The area overhead of CDR in a post-layout core.

Figure 8: Area Analysis of CDR

(VH). Global-home is the baseline framework where home nodes are interleaved across the entire system on a per cache line basis and memory is kept coherent among all cores in the system. Scale-Out processors statically partition all of the cores in the system into small groups to ensure low storage overhead and communication cost. These groups, called pods, are similar to sharer/home domains, but are rigid in size and shape and do not enable inter-pod communication. Virtual Hierarchies provides VM-level dynamic home domains but does not explore runtime modification of home domains, so the domain size for one VM is fixed once assigned.

In our studies on the manycore system, Scale-out uses a fixed pod size of 64 cores; Virtual Hierarchies uses the same VM size as the maximum number of threads of each benchmark, and CDR uses a maximum home/sharer domain size of 64 in which the domain size can change at runtime. For the warehouse scale computer, the maximum pod/domain size is set to 8 machines which correspond to 80 cores. We do not vary the coherence protocol, and all frameworks use the same directory scheme (Full-map) with directory caches [12].

We utilize the 13 multi-threaded benchmarks from the PARSEC benchmark suite [23]. Applications are simulated with small input sets, and we simulate the entire benchmark. For simulations on the manycore system, PARSEC benchmarks are executed with different thread count parameters (1, 2, 4, 8, and 16), but this parameter only specifies the minimum number of threads rather than the actual number. Especially for *ferret*, the actual number of threads is more than 64 when a thread count parameter of 16 is used. Since our simulator assumes one thread per core and both Scale-out and CDR have a maximum domain size of 64 in our configuration, we either omit this benchmark configuration or simulate with a parameter of 8 instead for compatibility. For the boundary evaluation of SR, we configure each benchmark to run with closest to 72 threads (and more than 64 threads). Unfortunately, *facesim*, *swaptions* and *x264* cannot meet this constraint so we omit them. Simulations on the warehouse scale computer set the actual number of threads to be 10, 20 and 40 in order to execute on 1, 2 and 4 machines. We omit *swaptions* and *x264* for the same reason.

4.2 Area Overhead of CDR

Figure 9: Performance degradation of Full-map with V-SR running PARSEC on the 1024-core system. Results are normalized to Full-map without V-SR. Benchmarks are simulated with a thread count parameter of 16 except *ferret* uses 8.

Figure 10: Cache miss rates (left) and absolute number of cache misses (right) of sharer map caches for Full-map with V-SR running PARSEC on 1024-core system.

We have implemented the entire CDR framework in Verilog in a 25-core processor that was taped out in IBM’s 32nm SOI process. We have received chips back from the foundry (Oct. 2015) and they are under test. All the architectural modifications for page-level Sharer Restriction and Home Restriction discussed in Section 3 are seamlessly integrated with the memory system of the processor (The VM/application-level design can be fully emulated by the page-level design so there is no need to implement them separately). We have leveraged the OpenSPARC T1 core in the chip we built, but in this paper, we evaluate a more standard x86-64 design. The logic to implement CDR is the same between different ISAs as it only affects the cache system, therefore we present logic area based on our OpenSPARC T1 design. Each core of the 25-core processor contains a 8KB private write-through L1 data cache, a 8KB private write-back L1.5 data cache and a 64KB distributed shared L2 cache with integrated directories. We present the area breakdown of a core in the 25-core processor from the post-layout design in IBM’s 32nm SOI process in Figure 8. Sharer Restriction costs 1.9% of the post-layout area, which is mainly due to the sharer map cache. Home Restriction costs 3.0% of the area because we store the Home Domain ID per cache line in the private L1.5 as discussed in Section 3.2.1. This is not necessary if write-through caches are used in the L1.5. The total area overhead of CDR is 4.9% within a core. Since we leverage smaller OpenSPARC T1 cores and caches in our processor compared to a standard x86-64 design, the area overhead of CDR is expected to be lower in the x86-64 case.

4.3 Sharer Restriction Evaluation

4.3.1 Storage Analysis

Both Scale-out and Full-map with Sharer Restriction

Figure 11: Performance degradation of Full-map with V-SR running PARSEC on the 1024-core system. Results are normalized to Full-map without V-SR. Benchmarks are simulated with closest to 72 threads.

tion achieve constant overhead scaling as the number of cores increases due to constant pod size and maximum sharer domain size. In contrast, full-map is not scalable with more cores/nodes, incurring around 14x and 1440x storage overhead respectively when scaling to 1024 and 100,000 cores/nodes.

4.3.2 Performance

Figure 9 presents the performance degradation of Full-map with V-SR on a 1024 core system. Since there is no violation of the maximum domain size, there is no need to use P-SR for finer-grain domains in this case. As shown in Figure 9, all benchmarks are within 3% performance degradation, and the average performance loss is 0.3%. Figure 10 shows both the overall sharer map cache (SMC) miss rates and total number of cache misses for all benchmarks calculated by dividing the total number of misses by the total number of accesses across all sharer map caches in the system. Since each application only maintains one sharer domain with at most 64 cores, they can all fit in one sharer map cache so all misses are cold misses. Considering that the number of accesses on sharer map caches is already a very small portion of memory accesses, an average SMC miss rate of $\sim 1\%$ explains the reason for negligible performance impact of V-SR. The *blackscholes* application has higher SMC miss rate than other applications because it has far fewer SMC accesses resulting in a higher cold miss rate. Since the total number of SMC accesses varies for different benchmarks, the absolute number of cache misses can have a more straightforward reflection of the performance degradation than the SMC miss rate. Applying V-SR on the warehouse scale computer leads to similar results because the sharer domain size is independent of scale, so we do not present this scenario.

In order to evaluate the performance of SR with violations of maximum domain size, we simulate the performance degradation of Full-map with V-SR and P-SR on a 1024 core system in Figure 11. Benchmarks are configured with closest to 72 threads. Although sharer domains larger than 64 cores can cause broadcasts, both V-SR and P-SR still result in low performance penalty (3.3% and 1.2% on average) because broadcasts are triggered on a per-line basis as mentioned in Section 3.1.3. Overall, P-SR has better performance than V-SR due to fewer broadcasts with finer-grain sharer domains, but V-SR achieves better performance than P-SR for a few benchmarks because P-SR introduces more domains and leads to more SMC misses.

Figure 12: Performance gain of Scale-out, Virtual Hierarchies, V-HR and P-HR normalized to Global-home running PARSEC on 1024-core system. Each benchmark is simulated with multiple thread count parameters: 1, 2, 4, 8, 16. *Ferret* uses more than 64-threads when running with a parameter of 16, therefore we omit this result.

Figure 13: Performance gain of Scale-out, Virtual Hierarchies, V-HR and P-HR normalized to Global-home running PARSEC benchmarks on the 100,000-machine system. Each benchmark is simulated on 1, 2 and 4 machines.

4.4 Home Restriction Evaluation

4.4.1 Performance

In the first experiment, we run individual PARSEC benchmarks with varying thread counts on a simulated system with 1024 cores (The *ferret* benchmark with thread parameter of 16 is omitted), and the performance results of different schemes are shown in Figure 12. Because cache missing request nodes and home nodes are in the same home domain, V-HR shortens the average communication distance and achieves a performance gain. P-HR localizes communication even further at page level, resulting in the best performance among all schemes. Compared to Global-home, a 24% average performance gain is achieved by V-HR and 29% by P-HR. Scale-out achieves a similar goal by statically dividing the system into separate pods, but because pod size is fixed, a mismatch between pod size and application size results in less gain, especially for applications with fewer threads. On average, Scale-out has 18% performance gain over Global-home. Virtual Hierarchies performs better than Scale-out because the VM size of each application is adjusted to match the maximum application size (the number of cores used by the application). However, since it cannot adjust VM size at runtime to match application needs like V-HR nor can it have finer-grain domains at page-level like P-HR, Virtual Hierarchies performs worse than HR.

Similarly, we run individual PARSEC benchmarks with varying thread counts executing on a varying number of machines on a warehouse scale shared memory computer with 100,000 machines. Because of the longer inter-machine communication latency and larger number of homes, localized communication results in a much larger performance impact as shown in Figure 13. Global-home results in much worse performance than all other schemes because the home nodes are distributed across all 100,000 machines. P-HR and V-HR

Figure 14: Performance gain of P-HR due to home domain merging.

Figure 15: Overall cache miss rates of home map caches for P-HR without and with home domain merging.

lead to more performance gains than other schemes with increasing number of execution machines. On average, P-HR, V-HR, Virtual Hierarchies and Scale-out lead to 23.04x, 16.35x, 13.43x and 6.07x performance gain respectively compared to Global-home.

4.4.2 Home Domain Merging of P-HR

As mentioned in Section 3.2.3, we adopt, by default, home domain merging to reduce the total number of home domains for P-HR. In order to investigate its performance impact, we simulate P-HR both with and without home domain merging in the 1024-core system. Figure 14 presents the performance gain of P-HR due to home domain merging, which can be up to ~30% for *canneal* and is 7% on average across all benchmarks. The performance benefit is mainly caused by two reasons: 1) Fewer home domains result in fewer runtime modification operations 2) Fewer home domains lead to lower home map cache miss rate as shown in Figure 15.

After home domain merging, the average HMC miss rate drops significantly from 25% to less than 1%.

5. RELATED WORK

Directory-based protocols are motivated by bus-based protocols' inability to scale beyond a few tens of cores. Early proposals include organizations like duplicate-tag[34, 35] or sparse directory [12], with sharer storage schemes like full-map share vector [9], coarse-grain vector [12, 36], or limited pointers [10, 11]. While these techniques help solve the bandwidth scalability challenge, they are inadequate for manycore systems and warehouse scale shared memory computers because of the exponentially increasing storage. To address the storage challenge, researchers have recently proposed the use of bloom filters [13, 37], highly-associative caches [15, 38], and deduplication of directory entries and sharer tags [14, 37, 15]. These generalized solutions are orthogonal to our approach and can be applied together to increase the scalability of the system. Another solution is to use hierarchical coherence systems on a manycore chip [39]. In contrast, we have demonstrated in our paper that CDR works well in a 1024-core chip with constant storage overhead and negligible performance loss. Compared to a flat system, the hierarchical design introduces extra traffic and directory lookup when two sharers from different subtrees communicate with each other. In addition, multi-level coherence protocols are also harder to design and verify.

Another direction researchers have explored is to not keep directories at all. Non-uniform cache architecture [40] (NUCA) and its variants (D-NUCA, R-NUCA [24]) maintain only one copy per cache line on-chip, therefore obviating the need to track sharers. Software-managed coherence [41, 42, 43, 44, 45, 46, 47, 48] off-loads the coherence tracking from hardware to software. Message-passing systems, like Intel's SCC chip [7, 49], also do not need directories. The above schemes all have inherent draw-backs compared to traditional directories: the NUCA approach, while simpler, trades complexity for spatial-temporal locality performance; software-managed coherence can have poor performance; message-passing and software-managed coherence can necessitate changes in programming model.

The Scale-Out architecture [21] restricts the coherent space by physically partitioning the cores into fixed-size pods. Communication is only possible among cores within the same pod. Our motivation is similar, to reduce the scope of coherence, however our work completely decouples sharer restriction from home restriction, hence achieving more flexibility in creating and maintaining the coherence domains at runtime, resulting in better performance. CDR can flexibly locate cores and home nodes and hence does not suffer from unused cores that can hinder Scale-Out designs. CDR's inter-domain communication is also realizable through combining V-SR with P-SR, a feature not available in the Scale-Out architecture.

In Virtual Hierarchies (VH) [22], Marty et al. pro-

posed a malleable two-level coherence hierarchy to facilitate spatial locality for communicating cores which shares some similarity with V-HR in our work. However, V-HR supports runtime modification of home domains for performance optimization which is not implemented in VH. VH also does not consider the storage overhead problem which we address with SR. In VH, each directory entry contains a global full-map bit-vector to track sharers, whereas in CDR we restrict the sharer set to reduce storage overhead.

The problem of cache line placement in NUCA is also similar to home-node allocation, as only one copy of a cache line is allowed to exist. In static NUCA, the location of the cache line is statically mapped. In dynamic NUCA, a line can exist in multiple possible locations and moves around as dictated by the performance policy. As the location of the cache line is not exact, the requester needs to do multiple searches. Finally, in reactive NUCA, entries are classified as either private or shared. Classified as private, the line is statically mapped as a member of a fixed, small-size, cluster. Classified as shared, however, the mapping reverts back to being statically mapped. Recent proposals have also explored replication [50, 51], coherence protocol based optimization [52, 53, 54] and software configurable policies [55, 56], trading implementation complexity for performance. With regard to cache line placement, this paper explores runtime modification of the home node and how to support it at the software-hardware interface.

Warehouse scale shared memory has much commonality to distributed shared memory systems in high performance computing [57, 58, 59, 60]. We see CDR as a potential mechanism to expand the scale of cache coherent systems for most use cases beyond current designs to a warehouse scale, while keeping hardware costs in check. Accessing large amounts of shared DRAM across the data center has also recently been popularized by the RAMCloud project [61] and key-value stores [62, 63]. CDR addresses some of these use cases, such as enabling access to memory on other nodes, but does not enable fully scalable, globally coherent data access by massive numbers of cores. While focusing on solving the scalability issue, CDR did not address other important challenges in warehouse scale shared memory such as fault-tolerance, resource management and software.

6. CONCLUSION

CDR explores the insight that many shared memory applications only use a subset of cores in a large shared memory system. By dynamically creating multiple, arbitrary coherence domains to restrict sharers or homes on a per-VM/application or per-page basis storage and communication distance can be reduced. As a result, CDR achieves constant storage overhead with the increase of core counts with SR, and a 29% and 23.04x performance gain respectively with HR on a 1024-core system and a 1,000,000-core warehouse scale computer compared to global home placement using a full-map directory. The entire CDR framework has been implemented in Verilog in a 25-core processor taped out in

IBM's 32nm process with reasonable area overhead.

7. ACKNOWLEDGEMENTS

This work was supported by the NSF under Grant CCF-1217553, AFOSR under Grant FA9550-14-1-0148, and DARPA under Grants N66001-14-1-4040 and HR0011-13-2-0005. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

8. REFERENCES

- [1] F. Mueller, "A library implementation of POSIX threads under UNIX," in *In Proceedings of the USENIX Conference*, pp. 29–41, 1993.
- [2] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *Computational Science Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, 1998.
- [3] D. Johnson, M. Johnson, J. Kelm, W. Tuohy, S. S. Lumetta, and S. Patel, "Rigel: A 1,024-core single-chip accelerator architecture," *Micro, IEEE*, vol. 31, pp. 30–41, July 2011.
- [4] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *Micro, IEEE*, vol. 27, no. 5, pp. 15–31, 2007.
- [5] D. Wentzlaff, C. J. Jackson, P. Griffin, and A. Agarwal, "Configurable fine-grain protection for multicore processor virtualization," in *Proceedings of the 39th Annual International Symposium on Computer Architecture, ISCA '12*, pp. 464–475, 2012.
- [6] C. Ramey, "TILE-Gx ManyCore Processor: Acceleration Interfaces and Architecture," in *Hot Chips 23*, 2011.
- [7] J. Howard *et al.*, "A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), IEEE International*, pp. 108–109, 2010.
- [8] G. Chrysos, "Knights Corner, Intel's first Many Integrated Core (MIC) Architecture Product," in *Hot Chips 24*, 2012.
- [9] L. M. Censier and P. Feautrier, "A new solution to coherence problems in multicache systems," *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 1112–1118, 1978.
- [10] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, "An evaluation of directory schemes for cache coherence," in *ACM SIGARCH Computer Architecture News*, vol. 16, pp. 280–298, IEEE Computer Society Press, 1988.
- [11] D. Chaiken, J. Kubiawicz, and A. Agarwal, "LimitLESS Directories: A Scalable Cache Coherence Scheme," in *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS IV*, pp. 224–234, ACM, 1991.
- [12] A. Gupta, W. Dietrich Weber, and T. Mowry, "Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes," in *In International Conference on Parallel Processing*, pp. 312–321, 1990.
- [13] J. Zebchuk, V. Srinivasan, M. K. Qureshi, and A. Moshovos, "A tagless coherence directory," in *Microarchitecture (MICRO), 42nd Annual IEEE/ACM International Symposium on*, pp. 423–434, IEEE, 2009.
- [14] H. Zhao, A. Shriraman, and S. Dwarkadas, "SPACE: sharing pattern-based directory coherence for multicore scalability," in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques, PACT '10*, pp. 135–146, ACM, 2010.
- [15] D. Sanchez and C. Kozyrakis, "SCD: A Scalable Coherence Directory with Flexible Sharer Set Encoding," in *Proceedings of the 18th international symposium on High Performance Computer Architecture (HPCA-18)*, 2012.
- [16] X. Fan, W.-D. Weber, and L. A. Barroso, "Power Provisioning for a Warehouse-Sized Computer," in *Proceedings of the 34th Annual International Symposium on Computer Architecture, ISCA '07*, pp. 13–23, 2007.
- [17] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, 2008.
- [18] M. Bhaduria, V. M. Weaver, and S. A. McKee, "Understanding parsec performance on contemporary cmps," in *Workload Characterization (IISWC), IEEE International Symposium on*, pp. 98–107, IEEE, 2009.
- [19] P. D. Bryan, J. G. Beu, T. M. Conte, P. Faraboschi, and D. Ortega, "Our many-core benchmarks do not use that many cores," in *Workshop on Duplicating, Deconstructing, and Debunking (WDDD)*, vol. 6, p. 8, 2009.
- [20] N. Ioannou and M. Cintra, "Complementing user-level coarse-grain parallelism with implicit speculative parallelism," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 284–295, ACM, 2011.
- [21] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi, "Scale-out processors," in *Computer Architecture (ISCA), 39th Annual International Symposium on*, pp. 500–511, June 2012.
- [22] M. R. Marty and M. D. Hill, "Virtual hierarchies to support server consolidation," in *Proceedings of the 34th annual international symposium on Computer architecture, ISCA '07*, pp. 46–56, 2007.
- [23] C. Bienia, *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.
- [24] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive NUCA: Near-optimal Block Placement and Replication in Distributed Caches," in *Proceedings of the 36th Annual International Symposium on Computer Architecture, ISCA '09*, pp. 184–195, 2009.
- [25] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhanian, "The Multikernel: A New OS Architecture for Scalable Multicore Systems," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, SOSP '09*, pp. 29–44, ACM, 2009.
- [26] D. Wentzlaff and A. Agarwal, "Factored operating systems (fos): the case for a scalable operating system for multicores," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 2, pp. 76–85, 2009.
- [27] D. Wentzlaff, C. Gruenwald, III, N. Beckmann, K. Modzelewski, A. Belay, L. Youseff, J. Miller, and A. Agarwal, "An operating system for multicore and clouds: Mechanisms and implementation," in *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10*, pp. 3–14, 2010.
- [28] D. R. Engler, M. F. Kaashoek, and J. O'Toole, Jr., "Exokernel: An Operating System Architecture for Application-level Resource Management," in *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles, SOSP '95*, pp. 251–266, ACM, 1995.
- [29] S. Boyd-Wickizer, H. Chen, R. Chen, Y. Mao, F. Kaashoek, R. Morris, A. Pesterev, L. Stein, M. Wu, Y. Dai, Y. Zhang, and Z. Zhang, "Corey: An operating system for many cores," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI'08*, pp. 43–57, USENIX Association, 2008.
- [30] E. Bugnion, S. Devine, and M. Rosenblum, "Disco: Running commodity operating systems on scalable multiprocessors," in *Proceedings of the ACM Symposium on Operating System Principles*, pp. 143–156, 1997.
- [31] K. Govil, D. Teodosiu, Y. Huang, and M. Rosenblum, "Cellular Disco: Resource management using virtual clusters on shared-memory multiprocessors," in *Proceedings of the ACM Symposium on Operating System Principles*, pp. 154–169, 1999.
- [32] Y. Fu and D. Wentzlaff, "PriME: A parallel and distributed simulator for thousand-core chips," in *Performance*

- [33] “Intel®Xeon®Processor E5-2670 v2 (25M Cache, 2.50 GHz).” http://ark.intel.com/products/75275/Intel-Xeon-Processor-E5-2670-v2-25M-Cache-2_50-GHz.
- [34] P. Kongetira, K. Aingaran, and K. Olukotun, “Niagara: A 32-way multithreaded sparc processor,” *Micro, IEEE*, vol. 25, no. 2, pp. 21–29, 2005.
- [35] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzky, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, “Piranha: a scalable architecture based on single-chip multiprocessing,” *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2, pp. 282–293, 2000.
- [36] R. T. Simoni, *Cache coherence directories for scalable multiprocessors*. PhD thesis, to the Department of Electrical Engineering, Stanford University, 1992.
- [37] H. Zhao, A. Shriraman, S. Dwarkadas, and V. Srinivasan, “SPATL: Honey, I Shrunk the Coherence Directory,” in *Parallel Architectures and Compilation Techniques (PACT), International Conference on*, pp. 33–44, 2011.
- [38] M. Ferdman, P. Lotfi-Kamran, K. Balet, and B. Falsafi, “Cuckoo directory: A scalable directory for many-core systems,” in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pp. 169–180, IEEE, 2011.
- [39] M. M. K. Martin, M. D. Hill, and D. J. Sorin, “Why on-chip cache coherence is here to stay,” *Communications of the ACM*, vol. 55, pp. 78–89, July 2012.
- [40] C. Kim, D. Burger, and S. W. Keckler, “An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches,” in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS X*, pp. 211–222, ACM, 2002.
- [41] P. Keleher, A. L. Cox, and W. Zwaenepoel, “Lazy release consistency for software distributed shared memory,” in *Proceedings of the 19th Annual International Symposium on Computer Architecture, ISCA ’92*, pp. 13–21, 1992.
- [42] P. Keleher, A. L. Cox, S. Dwarkadas, and W. Zwaenepoel, “TreadMarks: distributed shared memory on standard workstations and operating systems,” in *Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference, WTEC’94*, 1994.
- [43] X. Zhou, H. Chen, S. Luo, Y. Gao, S. Yan, W. Liu, B. Lewis, and B. Saha, “A case for software managed coherence in manycore processors,” in *Poster session presented in the 2nd USENIX Workshop on Hot Topics in Parallelism*, 2010.
- [44] C. Fensch and M. Cintra, “An OS-based alternative to full hardware coherence on tiled CMPs,” in *High Performance Computer Architecture (HPCA), IEEE 14th International Symposium on*, pp. 355–366, IEEE, 2008.
- [45] J. H. Kelm, D. R. Johnson, M. R. Johnson, N. C. Crago, W. Tuohy, A. Mahesri, S. S. Lumetta, M. I. Frank, and S. J. Patel, “Rigel: An Architecture and Scalable Programming Interface for a 1000-core Accelerator,” in *Proceedings of the 36th Annual International Symposium on Computer Architecture, ISCA ’09*, pp. 140–151, 2009.
- [46] B. Choi, R. Komuravelli, H. Sung, R. Smolinski, N. Honarmand, S. Adve, V. Adve, N. Carter, and C.-T. Chou, “DeNovo: Rethinking the Memory Hierarchy for Disciplined Parallelism,” in *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pp. 155–166, Oct 2011.
- [47] H. Sung, R. Komuravelli, and S. V. Adve, “DeNovoND: Efficient Hardware Support for Disciplined Non-determinism,” in *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS ’13*, pp. 13–26, ACM, 2013.
- [48] H. Sung and S. V. Adve, “DeNovoSync: Efficient Support for Arbitrary Synchronization Without Writer-Initiated Invalidations,” in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS ’15*, pp. 545–559, ACM, 2015.
- [49] T. G. Mattson, M. Riepen, T. Lehnig, P. Brett, W. Haas, P. Kennedy, J. Howard, S. Vangal, N. Borkar, G. Ruhl, et al., “The 48-core SCC Processor: the programmer’s view,” in *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–11, 2010.
- [50] G. Kurian, O. Khan, and S. Devadas, “The locality-aware adaptive cache coherence protocol,” in *Proceedings of the 40th Annual International Symposium on Computer Architecture, ISCA ’13*, pp. 523–534, ACM, 2013.
- [51] P. Foglia, C. A. Prete, M. Solinas, and G. Monni, “Re-NUCA: Boosting CMP performance through block replication,” in *Digital System Design: Architectures, Methods and Tools (DSD), 13th Euromicro Conference on*, pp. 199–206, IEEE, 2010.
- [52] Ros, Alberto and Acacio, Manuel E. and García, José M., “Direct coherence: Bringing together performance and scalability in shared-memory multiprocessors,” in *Proceedings of International Conference on High Performance Computing*, pp. 147–160, 2007.
- [53] Ros, Alberto and Acacio, Manuel E. and García, José M., “DiCo-CMP: Efficient cache coherency in tiled CMP architectures,” in *Parallel and Distributed Processing, IEEE International Symposium on*, pp. 147–160, April 2008.
- [54] H. Hossain, S. Dwarkadas, and M. C. Huang, “POPS: Coherence Protocol Optimization for Both Private and Shared Data,” in *Parallel Architectures and Compilation Techniques (PACT), International Conference on*, pp. 45–55, Oct 2011.
- [55] J. Zhou and B. Demsky, “Memory management for many-core processors with software configurable locality policies,” in *Proceedings of the 2012 International Symposium on Memory Management, ISMM ’12*, pp. 3–14, ACM, 2012.
- [56] M. Schuchhardt, A. Das, N. Hardavellas, G. Memik, and A. Choudhary, “The impact of dynamic directories on multicore interconnects,” *Computer*, vol. 46, pp. 32–39, October 2013.
- [57] J. Laudon and D. Lenoski, “The SGI Origin: A ccNUMA Highly Scalable Server,” in *Computer Architecture, 1997. Conference Proceedings. The 24th Annual International Symposium on*, pp. 241–251, June 1997.
- [58] S. Corporation, “White Paper: Technical Advances in the SGI®UV Architecture,” June 2012.
- [59] C. Dubnicki, A. Bilas, K. Li, and J. Philbin, “Design and Implementation of Virtual Memory-mapped Communication on Myrinet,” in *Proceedings of the International Parallel Processing Symposium*, pp. 388–396, 1997.
- [60] R. Kessler and J. Schwarzmeier, “Cray T3D: a new dimension for Cray Research,” in *Compcon Spring ’93, Digest of Papers.*, pp. 176–182, Feb 1993.
- [61] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, D. Ongaro, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman, “The Case for RAMcloud,” *Communications of the ACM*, vol. 54, pp. 121–130, July 2011.
- [62] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Voshall, and W. Vogels, “Dynamo: Amazon’s highly available key-value store,” in *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles, SOSP ’07*, pp. 205–220, 2007.
- [63] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani, “Scaling memcache at facebook,” in *The 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pp. 385–398, USENIX, 2013.