Virtual Hierarchies to Support Server Consolidation

Michael R. Marty and Mark D. Hill
Computer Sciences Department
University of Wisconsin—Madison
{mikem, markhill}@cs.wisc.edu

ABSTRACT

Server consolidation is becoming an increasingly popular technique to manage and utilize systems. This paper develops CMP memory systems for server consolidation where most sharing occurs within Virtual Machines (VMs). Our memory systems maximize shared memory accesses serviced within a VM, minimize interference among separate VMs, facilitate dynamic reassignment of VMs to processors and memory, and support content-based page sharing among VMs. We begin with a tiled architecture where each of 64 tiles contains a processor, private L1 caches, and an L2 bank. First, we reveal why single-level directory designs fail to meet workload consolidation goals. Second, we develop the paper's central idea of *imposing a two-level virtual (or logical) coherence hierarchy on a physically flat CMP that harmonizes with VM assignment.* Third, we show that the best of our two virtual hierarchy (VH) variants performs 12-58% better than the best alternative flat directory protocol when consolidating Apache, OLTP, and Zeus commercial workloads on our simulated 64-core CMP.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles—cache memories, shared memory

General Terms

Management, Performance, Design, Experimentation

Keywords

Server consolidation, virtual machines, cache coherence, memory hierarchies, chip multiprocessors (CMPs), multicore, partitioning

1. INTRODUCTION

Server consolidation is becoming an increasingly popular technique to manage and utilize systems. In server consolidation (also called *workload consolidation*), multiple server applications are deployed onto *Virtual Machines (VMs)*, which then run on a single, more-powerful server. Manufacturers of high-end commercial servers have long provided hardware support for server consolidation such as logical partitions [22] and dynamic domains [9]. VMware [43] brought server consolidation to commodity x86-based systems without hardware support, while AMD and Intel have recently introduced hardware virtualization features [3, 20].

Virtualization technology's goals include the following: First and most important, VMs must isolate the function of applications and operating systems (OSs) running under virtualization. This is virtualization's *raison d'être*. Second, VMs also should isolate the performance of consolidated servers (e.g., to mitigate a misbehaving VM from affecting others). Third, the system should facilitate *dynamic reassignment (or partitioning* [37]) of a VM's resources (e.g., reassigning processor and memory resources to a VM). Flexible server resource management reduces wasteful over-provisioning and can even manage heat [19]. Fourth, the system should support inter-VM sharing of memory to enable features like *content-based page sharing.* This scheme, pioneered by Cellular Disco [7] and currently used by VMware's ESX server [44], eliminates redundant copies of pages with identical contents *across* VMs by mapping them to the same physical frame. Inter-VM page sharing, especially for code pages, can reduce physical memory demands by up to 60% [44].

Chip Multiprocessors (CMPs, multicores) provide excellent opportunities to expand server and workload consolidation. Sun's "Niagara" T1 processor already offers 32 threads on a single die with nominal power usage [25]. Intel's Tera-scale project explores using over a hundred identical processor/cache tiles on a single chip [19]. Figure 1 illustrates our baseline 64-tile CMP architecture with each tile consisting of a processor core, private L1 caches, and an L2 bank.

Our thesis is that the memory systems for large CMPs should be optimized for workload consolidation, as well as traditional single-workload use. In our view, these memory systems should:

- minimize average memory access time by servicing misses within a VM when possible;
- minimize interference among separate VMs to better isolate consolidated server performance;
- facilitate dynamic reassignment of cores, caches, and memory to VMs; and
- support inter-VM page sharing to reduce physical memory demand.

Figure 2 shows how future CMPs may run many consolidated workloads with *space sharing*. That is, different regions of cores are assigned to different VMs. But memory systems proposed for future CMPs appear not to target this kind of workload consolidation. Use of global broadcast for all coherence across such a large number of tiles is not viable. Use of a global directory in DRAM or SRAM forces many memory accesses to unnecessarily cross the chip, failing to minimize memory access time or isolate VM performance. Statically distributing the directory among tiles can do much better, provided that *VM monitors* (*hypervisors*) carefully map virtual pages to physical frames within the VM's

ISCA '07, June 9-13, 2007, San Diego, California, USA.

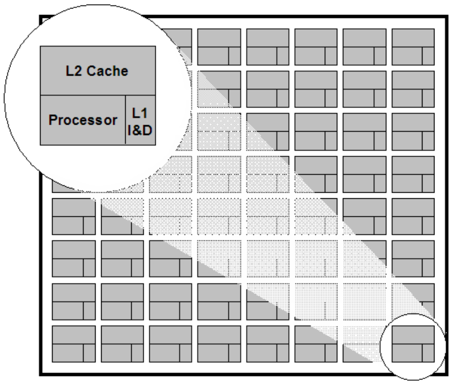


Figure 1: Tiled CMP architecture

tiles. Requiring the hypervisor to manage cache layout complicates memory allocation, VM reassignment and scheduling, and may limit sharing opportunities.

This paper proposes supporting workload consolidation on many-core CMPs by beginning with a straightforward idea and then developing some of its subtle implications. We propose ***to impose a two-level virtual (or logical) coherence hierarchy on a physically flat CMP that harmonizes with VM assignment.*** We seek to handle most misses within a VM (intra-VM) with a *level-one coherence protocol* that minimizes both miss access time and performance interference with other VMs. This first-level intra-VM protocol is then augmented by a *global level-two inter-VM coherence protocol* that obtains requested data in all cases. The two protocols will operate like a two-level protocol on a physical hierarchy, but with two key differences. First, the *virtual hierarchy (VH)* is not tied to a physical hierarchy that may or may not match VM assignment. Second, the VH can dynamically change when VM assignment changes.

Our VH protocols use level-one directory protocols where each block has a *dynamic home tile* within its VM. Home tile assignment can change on VM reassignment. Memory requests that are not completely satisfied at the home tile invoke the global level-two coherence. This occurs when VM assignment dynamically changes (without requiring the flushing of all caches) and for inter-VM sharing.

We specify two VH protocol classes that differ in how they perform level-two coherence. Protocol VH$_{\text{A}}$ uses a directory logically at memory. It is a virtualized version of a two-level directory protocol. Due to the dynamic nature of VM assignment, however, global directory entries must either point to all subsets of tiles (> 64 bits per block) or face additional complexity. Protocol VH$_{\text{B}}$ reduces the in-memory directory to a single bit per block, but sometimes uses broadcast. It reduces resource overheads, facilitates optimizations, and only uses broadcast for rare events like VM reassignment and for some inter-VM sharing.

Our VH protocols can also support workload consolidation running on a single OS without virtualization. While we will refer to hypervisors and virtual machines throughout the paper, the reader can replace these terms with OS and process, respectively.

Our contributions regarding CMP memory systems for workload consolidation are as follows:

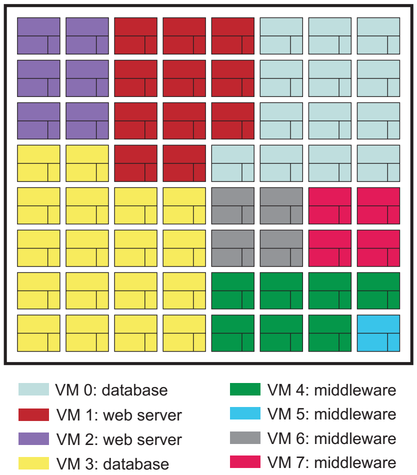


Figure 2: CMP running consolidated servers

- We show the value of imposing a virtual hierarchy (VH) on a CMP.
- We specify two protocol classes, VH$_{\text{A}}$ and VH$_{\text{B}}$, that address the goals of minimizing memory access time, minimizing inter-VM interference, facilitating VM reassignment, and supporting inter-VM sharing. VH$_{\text{A}}$ uses pure directory-based techniques, while VH$_{\text{B}}$ reduces global state and complexity.
- We find VH$_{\text{B}}$ performs 12-58% faster than the best alternative flat directory protocol when consolidating Apache, OLTP, and Zeus workloads on a 64-tile CMP.

The paper is organized as follows: Section 2 presents flat coherence protocols for a tiled CMP architecture. Section 3 presents our protocols for imposing a virtual hierarchy on a flat CMP. Section 4 discusses our evaluation methodology and Section 5 shows our evaluation results. Section 6 touches upon related work, and Section 7 concludes.

2. FLAT DIRECTORY COHERENCE

This section discusses some existing flat coherence protocol options for many-core, tiled CMPs. We assume directory-based coherence because CMPs with 64 or more tiles make frequent broadcasts slow and power-hungry. When a memory access is not satisfied within a tile, it seeks a directory, which provides information regarding whether and where a block is cached.

Directory protocols differ in the location and contents of directory entries. We discuss and will later evaluate three alternative directory implementations that have been advocated in the literature: (1) DRAM directory with on-chip directory cache; (2) duplicate tag directory; and (3) static cache bank directory. When applied to server consolidation, however, we will find that all these directory implementations fail to minimize average memory access time and do not isolate VM performance, but do facilitate VM resource reassignment and inter-VM sharing.

2.1 DRAM Directory w/ Directory Cache (DRAM-DIR)

Like many previous directory-based systems [26, 27], a CMP can use a protocol that stores directory information in DRAM. A straightforward approach stores a bit-vector for every memory block to indicate the sharers. The bit-vector, in the simplest form, uses one bit for every possible sharer. If the coherence protocol implements the Owner state (O), then the directory must also

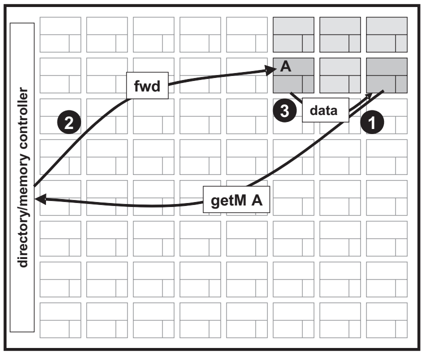


Figure 3: DRAM-DIR directory protocol with its global indirection for local intra-VM sharing (Section 2.1)

contain a pointer to the current owner. The state size can be reduced at a cost of precision and complexity (e.g., a coarse-grained bit vector [14] where each bit corresponds to a cluster of possible sharers).

A DRAM-based directory implementation for the CMP shown in Figure 3 treats each tile as a potential sharer of the data. Any miss in the tile's caches then issues a request to the appropriate directory controller. Directory state is logically stored in DRAM, but performance requirements may dictate that it be cached in on-chip RAM at the memory controller(s).

Figure 3 illustrates a sharing miss between two tiles in the same VM. The request fails to exploit *distance locality.* That is, the request may incur significant latency to reach the directory even though the data is located nearby. This process does not minimize memory access time and allows the performance of one VM to affect others (due to additional interconnect and directory contention). Since memory is globally shared, however, this design does facilitate VM resource reassignment and inter-VM sharing.

2.2 Duplicate Tag Directory (TAG-DIR)

An alternative approach for implementing a CMP directory protocol uses an exact duplicate tag store instead of storing directory state in DRAM [4, 8, 18]. Similar to shadow tags, directory state for a block can be determined by examining a copy of the tags of every possible cache that can hold the block. The protocol keeps the copied tags up-to-date with explicit or piggybacked messages. A primary advantage of a complete duplicate tag store is that it eliminates the need to store and access directory state in DRAM. A block not found in a duplicate tag is known to be idle (uncached).

As illustrated in Figure 4, a sharing miss still fails to exploit distance locality because of the indirection at the directory, which is now in SRAM and may be centrally located. Therefore this approach also fails to minimize average memory access time and to isolate VM performance from each other. Contention at the centralized directory can especially become problematic because of the cost in increasing the lookup bandwidth.

Furthermore, implementing a duplicate tag store can become challenging as the number of cores increases. For example, with a tiled 64-core CMP, the duplicate tag store will contain the tags for all 64 possible caching locations. If each tile implements 16-way associative L2 caches, then the *aggregate associativity* of all tiles is 1024 ways. Therefore, to check the tags to locate and invalidate sharers, a large power-hungry 1024-way content addressable memory may be required.

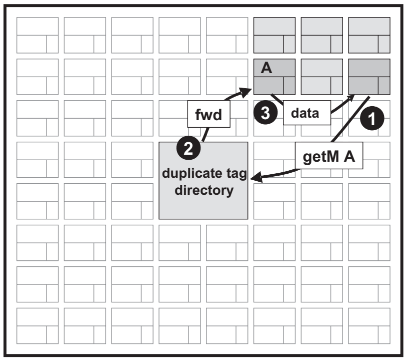


Figure 4: TAG-DIR with its centralized duplicate tag directory (Section 2.2)

2.3 Static Cache Bank Directory (STATIC-BANK-DIR)

A directory can be distributed among all the tiles by mapping a block address to a tile called the *home tile (node)* [23, 45]. Tags in the cache bank of the tile can then be augmented with directory entry state (e.g., a sharing bit vector). If a request reaches the home tile and fails to find a matching tag, it allocates a new tag and obtains the data from memory. The retrieved data is placed in the home tile's cache and a copy returned to the requesting core. Before victimizing a cache block with active directory state, the protocol must first invalidate sharers and write back dirty copies to memory.

This scheme integrates directory state with the cache tags thereby avoiding a separate directory either in DRAM or on-chip SRAM. However the tag overhead can become substantial in a many-core CMP with a bit for each sharer, and the invalidations and writebacks required to victimize a block with active directory state can hurt performance.

Home tiles are usually selected by a simple interleaving on low-order block or page frame numbers. As illustrated in Figure 5, the home tile locations are often sub-optimal because the block used by a processor may map to a tile located across the chip. If home tile mappings interleave by page frame number, hypervisor or operating system software can attempt to remap pages to page frames with better static homes [12] at a cost of exposing more information and complexity to the hypervisor or OS. Dynamic VM reassignment would then further complicate the hypervisor's responsibility for using optimal mappings. Thus, a distributed static cache bank directory also fails to meet our goals of minimizing memory access time and VM isolation. In fact, VM isolation especially suffers because a large working set of one VM may evict many cache lines of other VMs.

3. VIRTUAL HIERARCHIES

This section develops the paper's key idea for supporting server consolidation, namely, *imposing a two-level virtual coherence hierarchy on a physically flat CMP that harmonizes with VM assignment.* The next three sub-sections develop our virtual hierarchy (VH) ideas. We first specify a level-one directory protocol for intra-VM coherence. By locating directories within a VM, the level-one protocol minimizes memory access time and

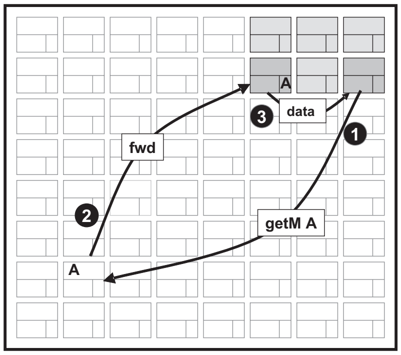


Figure 5: Static-Bank-Dir protocol with interleaved home tiles (Section 2.3)

isolates performance. We then develop two alternative global level-two protocols for inter-VM coherence (VH_A and VH_B).

3.1 Level-One Intra-VM Directory Protocol

We first develop an intra-VM directory protocol to minimize average memory access time and interference between VMs. When a memory reference misses in a tile, it is directed to a home tile which either contains a directory entry (in an L2 tag) pertaining to the block or has no information. The latter case implies the block is not present in this VM. When a block is not present or the directory entry contains insufficient coherence permissions (e.g., only a shared copy when the new request seeks to modify the block), the request is issued to the directory at the second level.

A surprising challenge for an intra-VM protocol is finding the home tile (that is local to the VM). For a system with a physical hierarchy, the home tile is usually determined by a simple interleaving of fixed power-of-two tiles in the local part of the hierarchy. For an intra-VM protocol, the home tile is a function of two properties: which tiles belong to a VM, and how many tiles belong to a VM. Moreover, dynamic VM reassignment can change both. It is also not desirable to require all VM sizes to be a power-of-two.

To this end, we support *dynamic home tiles* in VMs of arbitrary sizes using a simple table lookup that must be performed before a miss leaves a tile (but may be overlapped with L2 cache access). As illustrated in Figure 6, each of the 64 tiles includes a table with 64 six-bit entries indexed by the six least-significant bits of the block number. The figure further shows table values set to distribute requests approximately evenly among the three home tiles in this VM (p12, p13, and p14).[1] Tables would be set by a hypervisor (or OS) at VM (or process) reassignment. We also consider other approaches for setting the tables using hardware predictors, even on a per-page basis, but we do not explore them further in this paper.

The complement of a tile naming the dynamic home is allowing an intra-VM (level-one) directory entry to name the tiles in its current VM (e.g., which tiles could share a block). In general, the current VM could be as large as all 64 tiles, but may contain any subset of the tiles. We use the simple solution of having each intra-VM directory entry include a 64-bit vector for naming any of the tiles as sharers. This solution can be wasteful if VMs are small, since only bits for tiles in the current VM will ever be set. Of course, more compact representations are possible at a cost of additional bandwidth or complexity.

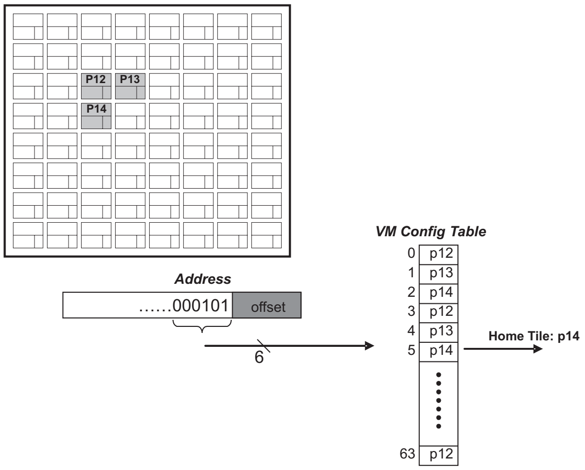


Figure 6: Tiles in a virtual machine use a configuration table to select dynamic homes within the VM

3.2 Virtual-Hierarchy-A (VH_A) Protocol

Virtual-Hierarchy-A (VH_A) implements global second-level coherence with a directory in DRAM and an optional directory cache at the memory controller(s). This second-level protocol operates somewhat like the single-level DRAM directory protocol described in Section 2.1, but with two key changes.

First, a level-two directory entry must name subsets of level-one directories. This is straightforward with a fixed physical hierarchy where the names and numbers of level-one directories are hard-wired into the hardware. An entry for 16 hard-wired level-one directories, for example, can name all subsets in 16 bits. In creating a virtual hierarchy, the home tile for a block may change with VM reassignment and the number of processors assigned to the VM may not be a power-of-two. Thus any tile can act as the level-one directory for any block. To name all the possible sub-directories at the level-two directory, we adapt a solution from the previous sub-section that allows an intra-VM directory entry to name the tiles in its current VM. Specifically, each level-two (inter-VM) directory contains a full 64-bit vector that can name any tile as the home for an intra-VM directory entry.

Second, a key challenge in two-level directory protocols is resolving races, a matter not well discussed in the literature. All directory protocols handle a coherence request to block B with a sequence of messages and state changes. A directory protocol is called *blocking* if, while it is handling a current request for B, it delays other requests for B (until a completion message for the current request) [15]. For single-level directories, blocking protocols are simpler (fewer transient states) than non-blocking ones, but forgo some concurrency. Naively implementing blocking in a two-level directory protocol, however, leads to deadlock because of possible cyclic dependencies at level-one directories.

Our VH_A two-level directory protocol resolves races as follows:

[1] Variants of the idea exist, e.g., the Cray T3E [36] translated node identifiers and the IBM Power4 [40] used a hard-wired interleaving for its three L2 cache banks.

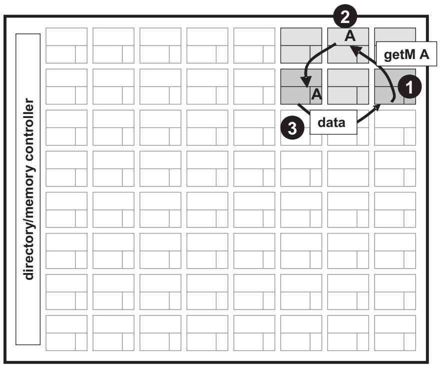


Figure 7: VH$_A$'s first level of coherence enables fast and isolated intra-VM coherence.

1) A processor sends a coherence request to a level-one directory, where it possibly waits before it either completes or reaches a *safe* point from which it makes a level-two directory request.
2) Each level-two request is eventually handled by the blocking level-two directory.
3) The level-two directory may forward requests to other level-one directories which handle these requests when their current request completes or reaches a safe point.
4) The coherence request completes at its initiating processor, which sends completion messages to unblock appropriate directories.

Any transient state can be considered a safe point if the protocol can handle any second-level action at any time. To reduce complexity and state-space explosion, we minimize the number of safe points. For example, requests that require actions at both protocol levels take no level-one actions until second-level actions complete (e.g., we do not start invalidating local sharers in parallel with issuing a second-level request).

Figure 7 illustrates how VH$_A$'s intra-VM coherence enables localized sharing within the VM to meet our goals of minimizing average memory access time and mitigating the performance impact of one VM on another. Specifically, (1) a processor issues a request to the dynamic home tile that is local to the VM; (2) a directory tag is found and the request redirects to the owning tile; (3) the owner responds to the requestor. We omit the completion message for brevity. The shared resources of cache capacity, MSHR registers, directory entries, and interconnect links are mostly utilized only by tiles within the VM.

Figure 8 shows how the second level of coherence allows for inter-VM sharing due to VM migration, reconfiguration, or page sharing between VMs. Specifically, (1) the request issues to the home tile serving as the level-one directory and finds insufficient permission within the VM; (2) a second-level request issues to the global level-two directory; (3) coherence messages forward to other level-one directories which then, in turn, (4) handle intra-VM actions and (5) send an ack or data on behalf of the entire level-one directory; (6) finally the level-one directory finishes the request on behalf of the requestor (completion messages not shown).

We expect the intra-VM protocol to handle most of the coherence actions. Since the level-two protocol will only be used for inter-VM sharing and dynamic reconfiguration of VMs, we now consider an alternative to VH$_A$ that aims to reduce the complexity and state requirements of both protocol levels.

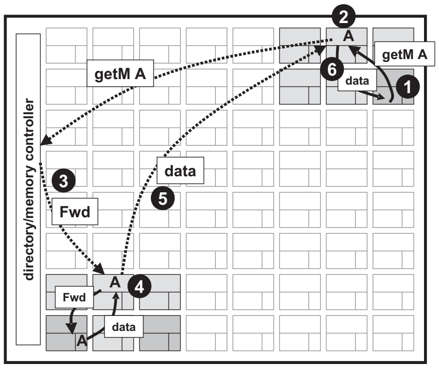


Figure 8: VH$_A$'s second-level coherence (dashed lines) facilitates VM reassignment and content-based page sharing.

3.3 VIRTUAL-HIERARCHY-B (VH$_B$) Protocol

VIRTUAL-HIERARCHY-B (VH$_B$) implements global second level coherence with a directory at DRAM with very small entries: a *single bit* tracks whether a block has *any* cached copies. With these small entries, VH$_B$ will have to fall back on broadcast, but broadcasts only occur after dynamic reconfiguration and on misses for inter-VM sharing.

Most sharing misses are satisfied within a VM via intra-VM coherence. Those not satisfied within a VM are directed to the memory (directory) controller. If the block is idle, it is returned to the requestor and the bit set to non-idle. If the block was non-idle, the pending request is remembered at the directory and the request is broadcast to all tiles. Once the request succeeds, the requestor sends a completion message to the memory controller.

VH$_B$ augments cached copies of blocks with a token count [30] to achieve three advantages. First, tiles without copies of a block do not need to acknowledge requests (one cache responds in the common case [13]). Second, the single per-block bit in memory can be thought of as representing either all or none tokens [32]. Third, level-one directory protocols can eliminate many transient states because it is no longer necessary to remember precise information (e.g., collecting invalidate acknowledgements for both local and global requests). This also makes it much easier to reduce the level-one directory state in tags, for example, by using limited or coarse-grained sharing vectors (the granularity could even adapt to VM size) or no state at all (relying on a broadcast within the VM).

To resolve races, VH$_B$ borrows some of VH$_A$'s strategies outlined in the previous section. Like VH$_A$, requests block at both protocol levels until receipt of completion messages. Thus an inter-VM request that reaches a blocked inter-VM directory buffers and waits until unblocked. If the block at memory is non-idle, the inter-VM directory broadcasts the request to all tiles. Request messages include a bit to distinguish between first-level and second-level requests. Unlike VH$_A$, second-level requests will reach both level-one directories and level-one sharers. Level-one sharers could ignore second-level requests and wait for the level-one directory to take appropriate action. Doing so would essentially make the race-handling algorithm identical to VH$_A$.

Instead, we exploit the broadcast for two reasons. First, exploiting the second-level broadcast reduces the complexity at the level-one directory by obviating the need to generate and track additional forward and invalidate messages. Second, the broadcast

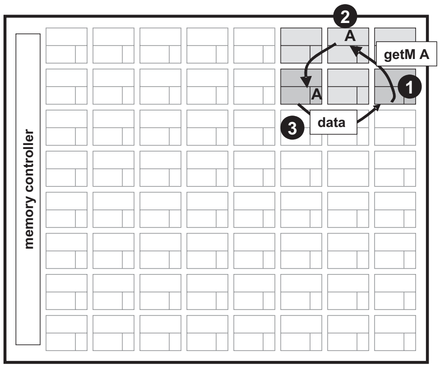


Figure 9: VH_B's first level of coherence, like VH_A, enables fast and isolated intra-VM coherence.

actually allows the level-one directory to be inexact. An inexact directory will enable optimizations discussed at the end of this section. Thus in VH_B, all tiles respond to both first-level and second-level requests. To handle races, requestors always defer to a second-level request. Furthermore, level-one directory tiles immediately re-forward the request to any tiles that have pending requests outstanding for the block. This forwarding may be redundant, but it ensures that winning requests always succeed.

Figure 9 illustrates how VH_B's intra-VM sharing operates similar to VH_A. Figure 10 shows a request requiring second-level coherence: (1) a processor's request issues to the dynamic home, (2) the request escalates to the memory controller because of insufficient permissions, (3) the memory controller initiates a broadcast because the bit indicates sharers exist, (4) the sharer responds directly to the requestor. We omit completion messages for brevity.

With a global broadcast as the second-level protocol, first-level directories in VH_B can be inexact, making subtle optimizations more amenable. One optimization that we implement recognizes private data and only allocates a cache tag in the requestor's local cache instead of also allocating a tag at the home tile to store unneeded directory state. If data transitions from private to shared, then the second level of coherence will locate the block with the broadcast. Our results in Section 5 will show this optimization improves performance. Another optimization we implement allows the victimization of a tag with active sharers in its directory without invalidating those sharers. Other possible optimizations that we do not implement include directly accessing memory (with prediction) without first checking the home tile, and selecting home tiles by using (per-page) predictors instead of a VM configuration table.

3.4 Virtual Hierarchy Discussion

Both VH_A and VH_B address our four goals for supporting server consolidation on physically flat CMPs, namely, minimizing memory access time, minimizing inter-VM performance interference, facilitating VM reassignment, and supporting inter-VM sharing.

VH_A and VH_B gracefully handle VM (or process) reassignment and rescheduling without flushing caches. The hypervisor (or OS) only needs to update the VM Config Tables on each affected core to reflect the new assignments. As processors request cache lines, the second coherence level will automatically locate the lines, on demand, at their old dynamic homes.

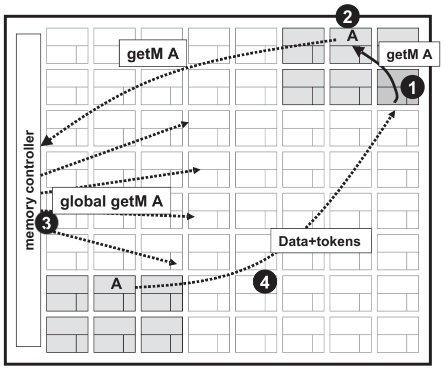


Figure 10: VH_B's second level of coherence (dashed lines) uses a broadcast to reduce memory state memory to 1-bit per block.

Protocol VH_A uses pure directory-based techniques. This approach, however, must allow the level-one and level-two directories to name any possible set of tiles, thus relying on exact directory state. Attempts to reduce directory state may either limit the flexibility of the virtual hierarchies or require additional complexity. Moreover, correctly managing all transient coherence states in a two-level directory approach is non-trivial [31].

Protocol VH_B reduces level-two directory state at DRAM from at least 64 bits down to a single bit per block. This is significant, for example, saving 4GB of DRAM in a 32GB system (assuming 64-byte blocks). It can also allow level-one directory entries to be approximate, or even eliminate level-one directory state by falling back on a broadcast to tiles within the VM (thereby approximating a virtual bus per VM).

Protocol VH_B reduces transient states that must be tracked thereby easing complexity. These simplifications make it easier to add optimizations that are theoretically orthogonal (e.g, approaches to managing replication). As an example of how VH_B reduces transient states, level-one directories in VH_B do not need to handle inter-VM and intra-VM invalidations.

Designers might consider using VH's first-level of coherence without a backing second-level coherence protocol. This option—VH_{null}—could still accomplish many of our goals with sufficient hypervisor support. Nonetheless, we see many reasons why VH_A/VH_B's two coherence levels may be preferred.

First, VH_{null} impacts dynamic VM reassignment as each reconfiguration or rescheduling of VM resources requires the hypervisor to explicitly flush all the caches of the VMs affected. VH_A/VH_B, on the other hand, avoids this complexity by implicitly migrating blocks to their new homes on demand. Second, VH_{null} supports read-only content-based page sharing among VMs, but, on a miss, obtains the data from off-chip DRAM. VH_A/VH_B improves the latency and reduces off-chip bandwidth demands of these misses by often finding the data on-chip. Third, VH_{null} precludes optimized workload consolidation at the OS/process level unless the OS is rewritten to operate without global cache coherence. VH_A/VH_B provides the cache coherence to support virtual hierarchies for individual OS processes. Fourth, VH_{null} adds complexity to the hypervisor, because some of its memory accesses (e.g., for implementing copy-on-write) must either bypass or explicitly flush caches. VH_A/VH_B, on the other hand, allows the hypervisor to use caching transparently. Fifth, VH_A/VH_B's second level of coherence allows subtle optimizations at the first-level that

Table 1: Simulation Parameters

Processors	64 in-order 2-way SPARC, 3 GHz
L1 Caches	Split I&D, 64 KB 4-way set associative, 2-cycle access time, 64-byte line
L2 Caches	1 MB per core. 10-cycle data array access, 64-byte line
Memory	16-64GB, 8 memory controllers, 275-cycle DRAM access + on-chip delay
Interconnect	8x8 2D Grid. 16-byte links. 5-cycle total delay per link

Table 2: Server Consolidation Configurations

Configuration	Description
oltp16x4p	Sixteen 4-processor OLTP VMs
apache16x4p	Sixteen 4-processor Apache VMs
zeus16x4p	Sixteen 4-processor Zeus VMs
jbb16x4p	Sixteen 4-processor SpecJBB VMs
oltp8x8p	Eight 8-processor OLTP VMs
apache8x8p	Eight 8-processor Apache VMs
zeus8x8p	Eight 8-processor Zeus VMs
jbb8x8p	Eight 8-processor SpecJBB VMs
oltp4x16p	Four 16-processor OLTP VMs
apache4x16p	Four 16-processor Apache VMs
zeus4x16p	Four 16-processor Zeus VMs
jbb4x16p	Four 16-processor SpecJBB VMs
mixed1	Two 16-processor OLTP VMs, two 8-processor Zeus VMs, four 4-processor SpecJBB VMs
mixed2	Four 8-processor OLTP VMs, four 8-processor Apache VMs

are not easily done with $VH_{null,}$ such as not allocating first-level directory entries for unshared data.

4. EVALUATION METHODOLOGY

We evaluate our protocols with full-system simulation using Virtutech Simics [42] extended with the GEMS toolset [29]. GEMS provides a detailed memory system timing model which accounts for all protocol messages and state transitions.

4.1 Target System

We simulate a 64-core CMP similar to Figure 1 with parameters given in Table 1. A tiled architecture was chosen because it maximizes the opportunity to overlay a virtual hierarchy on an otherwise flat system. Each core consists of a 2-issue in-order SPARC processor with 64 KB L1 I&D caches. Each tile also includes a 1 MB L2 bank used for both private and shared data depending on the policy implemented by the protocol. The 2D 8x8 grid interconnect consists of 16-byte links. We model the total latency per link as 5 cycles, which includes both the wire and routing delay. Messages adaptively route in a virtual cut-through packet switched interconnect. DRAM, with a modeled access latency of 275 cycles, attaches directly to the CMP via eight memory controllers along the edges of the CMP. The physical memory size depends on the configuration simulated, ranging from 16 to 64 GB. We set the memory bandwidth artificially high to isolate interference between VMs (actual systems can use memory controllers with fair queueing [33]).

4.2 Approximating Virtualization

Beyond setting up and simulating commercial workloads, a full-system simulation of virtual machines presents additional difficulties. Our strategy approximates a virtualized environment by concurrently simulating multiple functionally-independent machine instances. We realistically map and interleave the processors and memory accesses onto our CMP memory system timing model. Therefore we capture the memory system effects of server consolidation, though we do not model the execution of a hypervisor. Furthermore, although we target a system that supports inter-VM content-based page sharing and dynamic reassignment, we do not simulate page sharing between VMs and resources are never reassigned during our evaluation runs. Each VM maps onto adjacent processors to maximize the space sharing opportunities of our protocols.

4.3 Workloads

We consolidate commercial server workloads where each workload is considered a virtual machine and runs its own instance of Solaris 9. The workloads used are the following: an online transaction processing workload (OLTP), static web-serving workloads (Apache and Zeus), and a Java middleware workload (SpecJBB). Alameldeen et al. [2] further describe all workload configurations. To account for non-determinism in multithreaded workloads, we pseudo-randomly perturb simulations and calculate runtime error bars to 95% confidence [1].

Table 2 shows the different configurations of server consolidation we simulate. We first consider configurations that consolidate multiple instances of the same type of workload into virtual machines of the same size. These homogenous configurations allow us to report overall runtime after completing some number of transactions because all units of work are equivalent (i.e., all VMs complete the same type of transaction).

We then simulate server consolidation configurations of different workloads and of different virtual machine sizes. For these mixed configurations, the simulator runs for a chosen number of cycles and we count the number of transactions completed for each virtual machine. We then report the *cycles-per-transaction (CPT)* for each VM.

4.4 Protocols

We now discuss the implementation details of the protocols simulated. All implementations use write-back, write-allocate L1 caching with the local L2 bank non-inclusive. An important consideration for CMP memory systems is the number of replicas of a cache block allowed to coexist [45, 11, 8, 5]. This paper focuses on creating a virtual hierarchy to support server consolidation, and we consider the replication policy an independent issue that can be layered on top of our proposed mechanisms. Our protocols attempt to limit replication in order to maximize on-chip cache capacity. Nonetheless, in Section 5, we also simulate protocols which maximize replication by always allocating L1 replacements into the local L2 bank.

DRAM-DIR implements the protocol described in Section 2.1. The directory considers the cache hierarchy in each tile as private. To reduce replication, we implement a policy that uses a simple heuristic based on the coherence state of the block. In our MOESI implementation, an L1 victim in state M, O, or E will always allocate in the local L2 bank. However a block in the S-state will not allocate in the local bank because it is likely that another tile holds the corresponding O-block. If the O-block replaces to the directory, then the subsequent requestor will become the new owner. The E-state in this protocol sends a non-data control message to the directory upon replacement to eliminate a potential race. DRAM-DIR implements a one megabyte directory cache at each memory controller totaling a generous 8 MB of on-chip capacity. To further give DRAM-DIR the benefit of doubt, we fairly partition the 8 MB directory cache between VMs for simulations of homogenous workloads.

TAG-DIR implements the protocol described in Section 2.2 with full MOESI states. A centralized tag store, consisting of copied tags from every tile's caches, checks all tags on any request. We charge a total of three cycles to access the 1024-way CAM (for copied L2 tags) and to generate forward or invalidate messages. Like DRAM-DIR, each tile is nominally private and replication is controlled with the same heuristic based on the coherence state. To ensure that the duplicate tags are kept up-to-date, tiles send explicit control messages to the tag store when cleanly replacing data.

STATIC-BANK-DIR implements the protocol described in Section 2.3 with MESI states (the home tile is the implicit Owner). Home tiles interleave by the lowest six bits of the page frame address. Each L2 tag contains a 64-bit vector to name tiles sharing the block. The L2 controllers handle both indirections and fetches from memory. L1 caches always replace to the home tile and clean copies silently replace. In Section 5, we also describe the impact of interleaving home tiles by block address rather than page frame address.

VH$_A$ implements the two-level directory protocol as described in Section 3.2 with MOESI states at both levels. To help manage complexity, the L2 controller treats incoming L1 requests the same regardless if the request is from the local L1 or another L1 within the VM domain. Therefore L1 victims always replace to the dynamic home tile. The L2 cache ensures exact level-one directory state by enforcing strict inclusion amongst L1 sharers.

VH$_B$ implements the hybrid directory/broadcast protocol as described in Section 3.3 with states equivalent to MOESI. We also implement an optimization for private data. Private data does not allocate a directory tag at the dynamic home and always replaces to the core's local L2 bank. To distinguish private from shared data, we add a bit to the L1 cache tag that tracks if the block was filled by memory (with all the tokens). Since VH$_B$ allows inexact first-level directories, an L2 victim with active directory state does not invalidate sharers.

5. EVALUATION RESULTS

Uncontended Sharing Latencies. Before we consider the simulation of consolidated server workloads, we first verify expected protocol behavior with a microbenchmark. The microbenchmark repeatedly chooses random processor pairs in a VM to modify (and therefore exchange) a random set of blocks. Figure 11 shows the average sharing miss latencies as the number of processors in the VM increases from 2 to 64. Generally the flat directory protocols are not affected by VM size (as expected), while the VH$_A$ and VH$_B$ protocols dramatically reduce sharing latency for VM sizes much less than 64 (as expected by design). As the number of processors per VM grows, the virtual hierarchy flattens to eventually match the sharing performance of flat protocols. Lines are not smooth due to implementation effects (e.g., directory locations) and interactions among random blocks.

Runtime for Homogeneous Consolidation. Figure 12 shows the normalized runtime for the consolidated server workloads running multiple virtual machines of the same workload type (homogenous consolidation). We point out the following highlights:

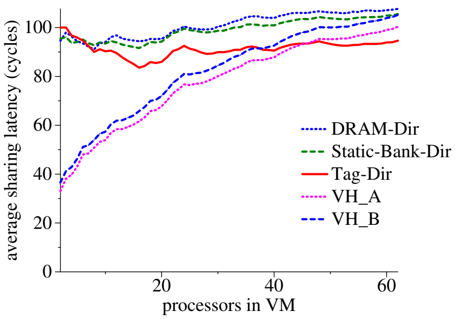


Figure 11: Uncontended L1-to-L1 sharing latency as the number of processors per virtual machines varies.

- VH$_B$ performs 12-58% faster than the best alternative flat directory protocol, TAG-DIR, for the Apache, OLTP, and Zeus configurations.
- VH$_A$ also outperforms or performs the same as the flat alternatives for most configurations. However for some, notably the 4x16p configurations, VH$_A$ is slower than VH$_B$ (and even TAG-DIR for JBB 4x16p). This is due to VH$_A$'s implementation that currently allows no migration of private data in a tile's local L2 bank. For VH$_B$, private data only allocates in the tile's local L2 bank rather than consuming a tag at the dynamic home for unneeded directory state.

Memory Stall Cycles. To gain further insights into the performance differences, Figure 13 shows the breakdown of memory system stall cycles. The bars show the normalized amount of cycles spent on servicing off-chip misses, hits in the local L2 bank, hits in remote L2 banks, and hits in remote L1 caches. The figure indicates that off-chip misses contribute to the majority of time spent in the memory system. Consequently, using a larger backing L3 cache for future many-core CMPs, as suggested by Intel [19], may be especially beneficial for supporting consolidated workloads. Nonetheless, we also see a significant number of cycles spent on misses serviced by remote L1 and L2 caches.

Our results show server consolidation increases the pressure on the CMP cache hierarchy. When running 16 instances of Apache with DRAM-DIR, 76% of cycles are spent on off-chip misses. But when running 4 instances of Apache, this number drops to 59% of cycles, due to increased hits in the local L2 banks.

Figure 13 illustrates how VH$_A$ and VH$_B$ reduce runtime by greatly decreasing the number of cycles spent servicing misses to remote L1 and L2 cache banks. For OLTP 8x8p, VH$_B$ spent 49% less memory cycles on remote L1 and L2 misses compared to DRAM-DIR and 39% less compared to TAG-DIR. This is due to VH$_B$'s reduced sharing latency within virtual machines, averaging 49 cycles instead of the 95 and 130 cycles for TAG-DIR and DRAM-DIR respectively. DRAM-DIR devotes the most cycles to these remote misses because of the global indirection and misses to the directory caches (85% overall directory cache hit rate for OLTP 8x8p when fairly partitioned).

TAG-DIR incurs slightly more cycles to off-chip misses because the request must first check the centralized tag directory before accessing the memory controller. The penalty averages 20 cycles for each DRAM access. Surprisingly, for some configurations, VH$_A$ and VH$_B$ have fewer off-chip misses than DRAM-DIR and TAG-DIR. We suspect that shared data with high

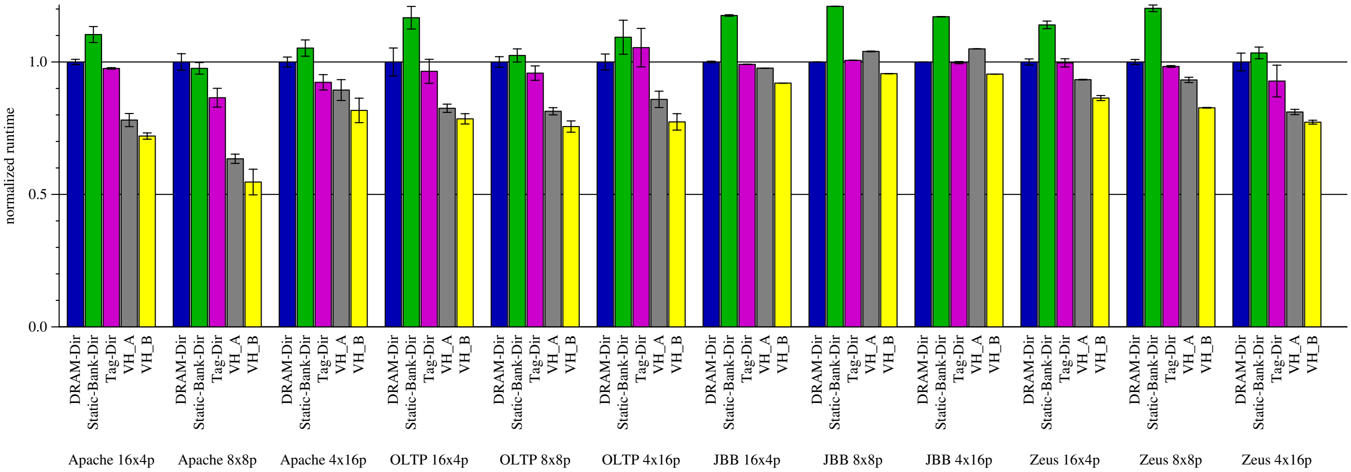


Figure 12: Normalized runtime for homogenous consolidation

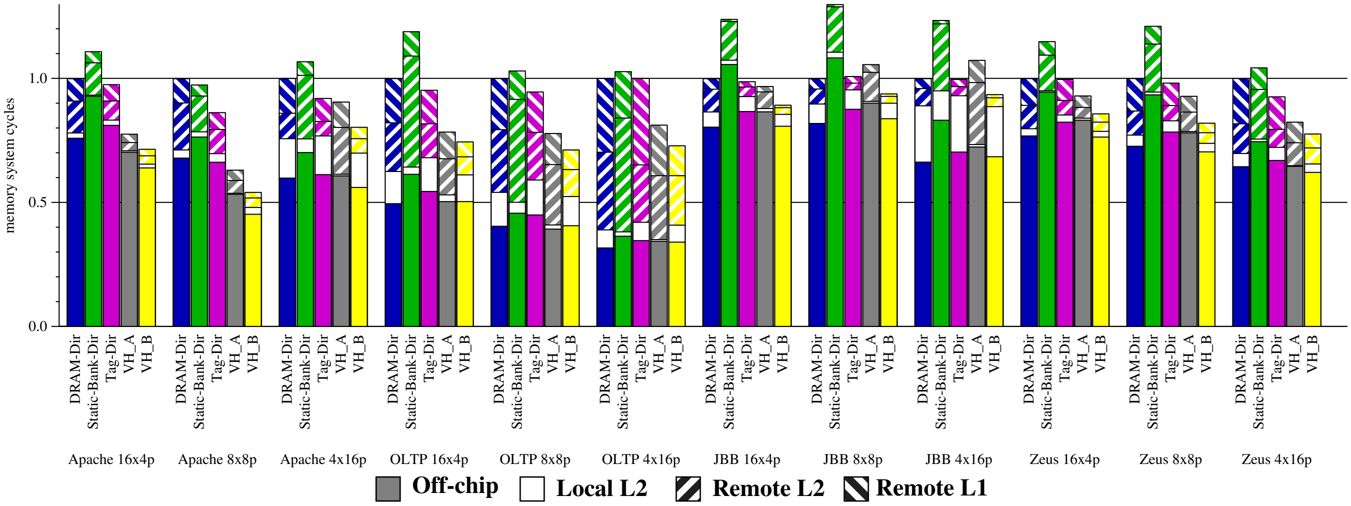


Figure 13: Memory stall cycles, by type, for homogenous consolidation

locality [5] and L1 miss rates is more likely to stay in the cache hierarchy because of more opportunities to update the LRU at the dynamic home (e.g., multiple cores access the same home tile).

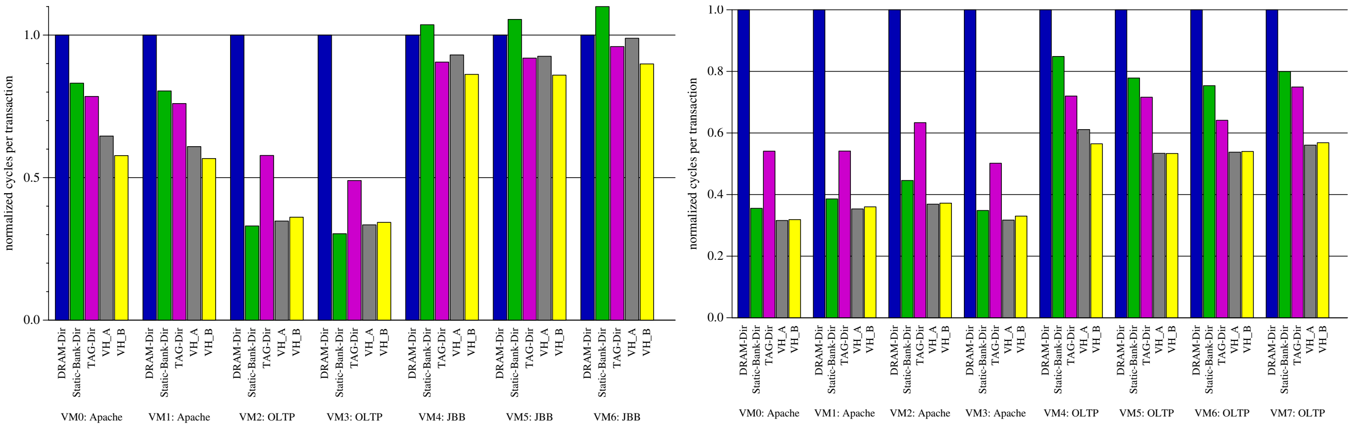
Static-Bank-Dir incurs the most cycles for off-chip misses. This is due to bank conflict because all VMs utilize all cache banks. On the other hand, Dram-Dir and Tag-Dir do not use home tiles, and the dynamic homes for VH_A and VH_B are isolated by virtual machine.

Effect of Interleaving. We also ran simulations where Static-Bank-Dir chooses home tiles by interleaving on block address instead of the page frame address (with no overlap). Interleaving on block address substantially hurt performance, especially for the massively consolidated configurations, because of increased set conflict. For example, with no overlap between the block and page frame address, hot OS blocks map to the same set in the same tile. Static-Bank-Dir's relative slowdown (not shown in figures) for the 16x4p configurations is 26%, 37%, 40%, and 21% respectively for Apache, OLTP, JBB, and Zeus. This slowdown is likely exaggerated by the nature of our homogeneous simulations, but the effects are still modestly present for mixed1 and mixed2 heterogeneous experiments.

Effect of Replication Policy. Table 3 shows the effect of increasing replication by always replacing L1 data into the core's local L2 bank. As shown, this replication policy had a minor effect on overall runtime for the 8x8p configurations (and is consistent with data for other configurations not shown). This suggests that compensating for non-local coherence through increased replication is not effective for these workloads.

Cycles-per-Transaction for Mixed Consolidation. Figure 14 shows the cycles-per-transaction (CPT) of each virtual machine running the mixed configurations. Comparisons are made within each virtual machine because the units of work differ between workload type and VM size.

VH_B offered the best overall performance by showing the lowest CPT for the majority of virtual machines. Dram-Dir performs poorly, because we did not fairly partition the directory cache for the mixed configurations. This severely impacted the effectiveness of the 8MB directory cache, resulting in a hit rate of only 45-55%. Static-Bank-Dir slightly outperformed VH_B for the OLTP virtual machines in the mixed1 configuration. This is because the working set of OLTP is very large and the Static-Bank-Dir protocol allows one VM to utilize the cache resources of other VMs. However where Static-Bank-Dir slightly improved the performance of the OLTP VMs in mixed1, it made



(mixed1) (mixed2)

Figure 14: Cycles-per-transaction (CPT) for each virtual machine running the mixed1 and mixed2 configurations

Table 3: Relative Performance improvement from low vs. high replication

	Apache 8x8p	OLTP 8x8p	Zeus 8x8p	JBB 8x8p
DRAM-DIR	-2.8%	-2.1%	1.0%	-0.2%
STATIC-BANK-DIR	-1.3%	1.4%	5.7%	-0.3%
TAG-DIR	2.2%	0.1%	1.2	-0.2%
VH_A[1]	n/a	n/a	n/a	n/a
VH_B	-1.7%	-2.6%	-4.2%	-0.1%

the JBB virtual machines perform more poorly because of the interference. On the other hand, VH_B isolates the cache resources between virtual machines thereby offering good overall performance. Further investigation of fairness is future work.

6. RELATED WORK

Our work creates a virtual hierarchy to help manage the vast cache resources of a many-core CMP when running consolidated workloads. An alternative approach minimizes hardware change by relying on the OS or hypervisor to manage the cache hierarchy through page allocation. The Cellular Disco [7] hypervisor exploited the structure of the Origin2000 to increase performance and performance isolation by using NUMA-aware memory allocation to virtual machines. Cho et al. [12] explore even lower level OS page mapping and allocation strategies in a system that interleaves cache banks by frame number, like STATIC-BANK-DIR. In contrast, our proposal frees the OS and hypervisor from managing cache banks and offers greater opportunities for VM scheduling and reassignment.

Many proposals partition or change the default allocation policy of monolithic shared L2 caches. Some schemes partition the cache based on the replacement policy [24, 38, 39], or partition at the granularity of individual cache sets or ways [10, 35]. Other approaches by Lie et al. [28] and Varadarajan et al. [41] partition a cache into regions, but neither schemes address coherence between regions. Rafique et al. [34] also propose support to allow the OS to manage the shared cache. Our VH scheme works at a higher level than managing a single shared cache.

Our work assigns the resources of a tile, including cache banks and processor cores, to virtual machines (or OS processes). Hsu et al. [17] studied optimality for cache sharing based on various metrics. Other approaches explicitly manage Quality of Service (QoS) in shared caches. Iyer examined QoS [21] with mechanisms for thread priorities such as set partitioning, per-thread cache line counts, and heterogeneous regions. Applying additional policies to better balance resources *within* a virtual machine domain is a topic of future work.

There has been much previous work in organizing CMP caches to exploit distance locality through replication or migration. D-NUCA was proposed to improve performance by dynamically migrating data closer to the cores based on frequency of access. While shown to be effective in uniprocessor caches [23], the benefits in a CMP are less clear [6]. To our knowledge, there is no complete, scalable solution for implementing D-NUCA cache coherence in a multiprocessor. Huh et al. [18] also studied trade-offs in L2 sharing by using a configurable NUCA substrate with unspecified mapping functions. All of their results relied on a centralized directory like our TAG-DIR. CMP-NuRapid [11] exploited distance locality by decoupling the data and tag arrays thereby allowing the flexible placement of data. However their CMP implementation requires a non-scalable atomic bus. Other recently proposed replication schemes include Victim Replication [45], Cooperative Caching [8], and ASR [5]. In Section 5, we showed how replication is an orthogonal issue to our work.

Finally, previous commercial systems have offered support for multiprocessor virtualization and partitioning [9, 16, 22]. Smith and Nair characterize these systems as either supporting physical or logical partitioning [37]. Physical partitioning enforces the assignment of resources to partitions based on rigid physical boundaries. Logical partitioning relaxes the rigidity, even offering the time-sharing of processors. VH offers a new way to space-share the vast cache resources of future many-core CMPs and applies to either physical or logical partitioning (e.g., time-sharing can be supported by saving and restoring the VM Config Tables).

7. CONCLUSION

CMPs provide excellent new opportunities to expand server and workload consolidation. Memory systems for future many-core CMPs should be optimized for workload consolidation as well as traditional single-workload use. These memory systems should maximize shared memory accesses serviced within a VM, minimize interference among separate VMs, facilitate dynamic

[1] We did not implement an alternative replication policy for VH_A because of the additional complexity involved with an already complex protocol.

reassignment of cores, caches, and memory to VMs, and support sharing among VMs.

This paper develops CMP memory systems for workload consolidation by imposing a two-level virtual (or logical) coherence hierarchy on a physically flat CMP that harmonizes with VM assignment. In particular, we show that the best of our two virtual hierarchy variants performs 12-58% better than the best alternative flat directory protocol when consolidating Apache, OLTP, and Zeus commercial workloads on a 64-tile CMP.

8. ACKNOWLEDGEMENTS

We thank Philip Wells for providing assistance with simulating consolidated workloads. We also thank Yasuko Watanabe, Dan Gibson, Andy Phelps, Min Xu, Milo Martin, Benjamin Serebrin, Karin Strauss, Kathleen Marty, Kevin Moore, Kathryn Minnick, the Wisconsin Multifacet group, and the Wisconsin Computer Architecture Affiliates for their comments and/or proofreading. Finally we thank the Wisconsin Condor project, the UW CSL, and Virtutech for their assistance.

This work is supported in part by the National Science Foundation (NSF), with grants EIA/CNS-0205286, CCR-0324878, and CNS-0551401, as well as donations from Intel and Sun Microsystems. Hill has significant financial interest in Sun Microsystems. The views expressed herein are not necessarily those of the NSF, Intel, or Sun Microsystems.

9. REFERENCES

[1] A. R. Alameldeen and D. A. Wood. Variability in Architectural Simulations of Multi-threaded Workloads. In *Proceedings of the Ninth IEEE Symposium on High-Performance Computer Architecture*, pages 7–18, Feb. 2003.

[2] A. R. Alameldeen and D. A. Wood. IPC Considered Harmful for Multiprocessor Workloads. *IEEE Micro*, 26(4):8–17, Jul/Aug 2006.

[3] AMD. *AMD64 Virtualization Codenamed Pacifica Technology: Secure Virtual Machine Architecture Reference Manual*, May 2005.

[4] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 282–293, June 2000.

[5] B. M. Beckmann, M. R. Marty, and D. A. Wood. ASR: Adaptive Selective Replication for CMP Caches. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2006.

[6] B. M. Beckmann and D. A. Wood. Managing Wire Delay in Large Chip-Multiprocessor Caches. In *Proceedings of the 37th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2004.

[7] E. Bugnion, S. Devine, K. Govil, and M. Rosenblum. Disco: Running Commodity Operating Systems on Scalable Multiprocessors. *ACM Transactions on Computer Systems*, 15(4):319–349, 1997.

[8] J. Chang and G. S. Sohi. Cooperative Caching for Chip Multiprocessors. In *Proceedings of the 33nd Annual International Symposium on Computer Architecture*, June 2006.

[9] A. Charlesworth. Starfire: Extending the SMP Envelope. *IEEE Micro*, 18(1):39–49, Jan/Feb 1998.

[10] D. Chiou, P. Jain, S. Devadas, and L. Rudolph. Dynamic Cache Partitioning via Columnization. In *Proceedings of Design Automation Conference*, 2000.

[11] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Optimizing Replication, Communication, and Capacity Allocation in CMPs. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, June 2005.

[12] S. Cho and L. Jin. Managing Distributed, Shared L2 Caches through OS-Level Page Allocation. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2006.

[13] A. Gupta and W.-D. Weber. Cache Invalidation Patterns in Shared-Memory Multiprocessors. *IEEE Transactions on Computers*, 41(7):794–810, July 1992.

[14] A. Gupta, W.-D. Weber, and T. Mowry. Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes. In *International Conference on Parallel Processing (ICPP)*, volume I, pages 312–321, 1990.

[15] 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, Jan. 1999.

[16] HP Partioning Continuum. http://h30081.www3.hp.com/products/wlm/docs/HPPartitioningContinuum.pdf, June 2000.

[17] L. R. Hsu, S. K. Reinhardt, R. Iyer, and S. Makineni. Communist, Utilitarian, and Capitalist Cache Policies on CMPs: Caches as a Shared Resource. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2006.

[18] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler. A NUCA Substrate for Flexible CMP Cache Sharing. In *Proceedings of the 19th International Conference on Supercomputing*, June 2005.

[19] From a Few Cores to Many: A Tera-scale Computing Research Overview. ftp://download.intel.com/research/platform/terascale/terascale_overview_paper.pdf, 2006.

[20] Intel Corporation. *Intel Virtualization Technology Specifications for the IA-32 Intel Architecture*, 2005.

[21] R. Iyer. CQoS: A Framework for Enabling QoS in Shared Caches of CMP Platforms. In *Proceedings of the 18th International Conference on Supercomputing*, pages 257–266, 2004.

[22] J. Jann, L. M. Browning, and R. S. Burugula. Dynamic reconfiguration: Basic building blocks for autonomic computing on IBM pSeries servers. *IBM Systems Journal*, 42(1), 2003.

[23] C. Kim, D. Burger, and S. W. Keckler. An Adaptive, Non-Uniform Cache Structure for Wire-Dominated On-Chip Caches. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct. 2002.

[24] S. Kim, D. Chandra, and Y. Solihin. Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2004.

[25] P. Kongetira. A 32-way Multithreaded SPARC Processor. In *Proceedings of the 16th HotChips Symposium*, Aug. 2004.

[26] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 241–251, June 1997.

[27] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy. The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 148–159, May 1990.

[28] C. Liu, A. Savasubramaniam, and M. Kandemir. Organizing the Last Line of Defense before Hitting the Memory Wall for CMPs. In *Proceedings of the Tenth IEEE Symposium on High-Performance Computer Architecture*, Feb. 2004.

[29] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. *Computer Architecture News*, pages 92–99, Sept. 2005.

[30] M. M. K. Martin, M. D. Hill, and D. A. Wood. Token Coherence: Decoupling Performance and Correctness. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 182–193, June 2003.

[31] M. R. Marty, J. D. Bingham, M. D. Hill, A. J. Hu, M. M. K. Martin, and D. A. Wood. Improving Multiple-CMP Systems Using Token Coherence. In *Proceedings of the Eleventh IEEE Symposium on High-Performance Computer Architecture*, Feb. 2005.

[32] M. R. Marty and M. D. Hill. Coherence Ordering for Ring-based Chip Multiprocessors. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2006.

[33] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith. Fair Queuing CMP Memory Systems. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2006.

[34] N. Rafique, W.-T. Lim, and M. Thottethodi. Architectural Support for Operating System-Driven CMP Cache Management. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2006.

[35] P. Ranganathan, S. Adve, and N. P. Jouppi. Reconfigurable Caches and their Application to Media Processing. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, June 2000.

[36] S. L. Scott. Synchronization and Communication in the Cray T3E Multiprocessor. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 26–36, Oct. 1996.

[37] J. E. Smith and R. Nair. *Virtual Machines*. Morgan Kaufmann, 2005.

[38] G. E. Suh, S. Devadas, and L. Rudolph. A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning. In *Proceedings of the Eighth IEEE Symposium on High-Performance Computer Architecture*, Feb. 2002.

[39] G. E. Suh, L. Rudolph, and S. Devadas. Dynamic Cache Partitioning for CMP/SMT Systems. *Journal of Supercomputing*, pages 7–26, 2004.

[40] J. M. Tendler, S. Dodson, S. Fields, H. Le, and B. Sinharoy. POWER4 System Microarchitecture. *IBM Journal of Research and Development*, 46(1), 2002.

[41] K. Varadarajan, S. K. Nandy, V. Sharda, A. Bharadwaj, R. Iyer, S. Makineni, and D. Newell. Molecular Caches: A Caching Structure for Dynamic Creation of Application-Specific Heterogeneous Cache Regions. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2006.

[42] Virtutech AB. Simics Full System Simulator. http://www.simics.com/.

[43] VMware. http://www.vmware.com/.

[44] C. A. Waldspurger. Memory Resource Management in VMware ESX Server. In *Proceedings of the 2002 Symposium on Operating Systems Design and Implementation*, Dec. 2002.

[45] M. Zhang and K. Asanovic. Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, June 2005.