DCC: A Dependable Cache Coherence Multicore Architecture

Omer Khan, Mieszko Lis, Yildiz Sinangil, *Member, IEEE,* and Srinivas Devadas, *Fellow, IEEE*
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract—Cache coherence lies at the core of functionally-correct operation of shared memory multicores. Traditional directory-based hardware coherence protocols scale to large core counts, but they incorporate complex logic and directories to track coherence states. Technology scaling has reached miniaturization levels where manufacturing imperfections, device unreliability and occurrence of hard errors pose a serious dependability challenge. Broken or degraded functionality of the coherence protocol can lead to a non-operational processor or user visible performance loss.
In this paper, we propose a dependable cache coherence architecture (DCC) that combines the traditional directory protocol with a novel execution-migration-based architecture to ensure dependability that is transparent to the programmer. Our *architecturally redundant* execution migration architecture only permits one copy of data to be cached anywhere in the processor: when a thread accesses an address not locally cached on the core it is executing on, it migrates to the appropriate core and continues execution there. Both coherence mechanisms can co-exist in the DCC architecture and we present architectural extensions to seamlessly transition between the directory and execution migration protocols.

Index Terms—Dependable architecture, cache coherence, multicores.

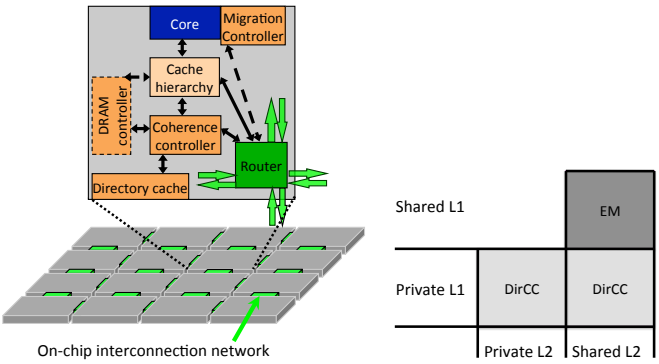
1 Introduction

The biggest challenge facing large-scale multicore processors is the convenience of programming. Today the shared memory abstraction is ubiquitous and multicores deploy a uniform hardware-coherent address space to efficiently guarantee a consistent view of data among cores. Snooping protocols do not scale to large core counts, while directory protocols require the overhead of directories, and, further, are complex to analyze, validate, and implement. The complexity of directory protocols is attributed to the directory indirections that require multiple controllers to interact via the on-chip interconnect. Network errors such as data corruption, misrouting, and erroneous allocation of virtual channels can cause the coherence protocol to deliver inconsistent data or, even worse, deadlock the system. Similarly, a loss of on-chip data and directory capacity due to memory bit-cell errors can severely affect system performance.

Semiconductor technology has reached miniaturization levels where shrinking device sizes and low voltage operation (for energy efficiency) are the root causes for an increase in hardware susceptibility to hard failures and device wear-out. Temperature and power supply variability due to aggressive power management is also expected to substantially contribute to device failure rates. Because of the inability to precisely control manufacturing, *process variation* can result in large variations of transistor threshold voltage, which in turn causes faults in hardware that is operating close to marginal conditions. Researchers predict that the threshold voltage for transistors on the same die could easily vary by 30% in future technology generations [1].

Today computer architects are investing heavily into means of detecting and correcting errors at the expense of area, power and performance. Traditionally, error detection and system reconfiguration is done during *manufacturing test*. Recently, researchers have also proposed several *online test* frameworks, where the system is concurrently or periodically tested at runtime, and subsequently reconfigured to bypass the faulty components [2], [3]. Common error correction techniques, including error correcting codes, sparing and replication, require large overhead. Instead of relying on these expensive error correction mechanisms, we propose an intelligent and architecturally redundant coherence architecture alternative that is not only fault-tolerant, but is also capable of improving performance by allowing the system to choose an appropriate coherence protocol at the fine granularity of OS-level pages.

• *Manuscript submitted: 05-Dec-2010. Manuscript accepted: 07-Jan-2011. Final manuscript received: 25-Jan-2011.*



(a) Fully distributed tiled multicore (b) On-chip cache organization

Fig. 1. Every tile has an in-order core, a cache hierarchy, a router, a coherence controller and a slice of the distributed directory cache. An execution migration controller allows each core to access another tile via the interconnect. Tiles connected to off-chip DRAM have a DRAM controller.

We propose a novel dependable cache coherence architecture (DCC) that combines traditional directory coherence (DirCC) with a novel execution-migration-based coherence architecture (EM) [4]. The DCC architecture ensures error resilience and increased dependability in a manner that is transparent to the programmer. EM follows a protocol that ensures that no writable data is ever shared among caches, and therefore does not require directories. When a thread needs to access an address cached on another core, the hardware efficiently migrates the thread's execution context to the core where the memory is allowed to be cached and continues execution there. Our DCC architecture allows these two independent coherence protocols to co-exist in the hardware. We present architectural extensions to seamlessly transition between the DirCC and EM protocols that ensure dependable operation of cache coherence.

2 Cache Coherence Architectures

Our baseline architecture (shown in Figure 1a) is a multicore chip that is fully distributed across tiles with a uniform address space shared by all tiles. Each tile in the chip communicates with others via an on-chip network. Such physical distribution allows the system layers to manage the hardware resources efficiently. Each tile has an Intel Atom-like core with a 2-level L1/L2 instruction and data cache hierarchy. Because multicores are expected to be critically constrained by limited package pin density, we expect a small number of on-chip memory controllers orchestrating data movement in and out of the chip, therefore limiting off-chip memory bandwidth [5].

Cache coherence lies at the core of correct functionality in the shared memory computational model and the cache hierarchy organization dictates the terms for keeping the caches coherent.

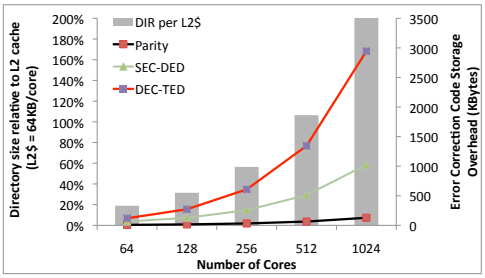


Fig. 2. Directory cache overhead relative to L2 cache (shown with bars). Although low overhead encoding schemes (shown as lines) e.g., parity, may be enough for error detection, the cost of error correcting codes increases sharply as the number of on-chip cores increase.

Figure 1b shows all possible cache configurations for our 2-level cache hierarchy. Under private-L1, a directory protocol is responsible for coherence. Although a shared-L2 caters well to applications with large shared working sets, many traditional high-performance parallel applications require larger private working sets, therefore making private-L2 an attractive option [6]. Hence, our baseline architecture utilizes a private-L2 for the DirCC configuration. On the other hand, a shared-L1/L2 configuration unifies the physically distributed per-core caches into a logically shared cache. Because only one copy of each cache line can be present on chip, coherence is trivially ensured and a directory protocol is not needed. The shared-L1/L2 organization is used in the EM architecture.

2.1 Directory Based Cache Coherence (DirCC)

The directory protocol brings *data* to the locus of the computation that is to be performed on it: when a memory instruction refers to an address that is not locally cached, the instruction stalls while the coherence protocol brings the data to the local cache and ensures that the address can be safely shared or exclusively owned.

A coherence controller utilizes a MOESI protocol with all possible cache-to-cache transfers to manage coherence for the associated address space regions. At a high level, the coherence controller implements protocol management state machines and contacts the directory to record coherence-related information for associated cache lines. The DirCC architecture implements a full-map physically distributed directory [7]. Although a full-map directory provides maximal performance, its area grows significantly as the core count increases (Figure 2). There are directory schemes that are more area-efficient than full-map [7], [8], but they incur the cost of additional protocol messages such as full-chip broadcast, and therefore performance loss. To keep track of data sharers and minimize expensive off-chip accesses, an on-chip directory cache is maintained. The directory cache is sized appropriately as a function of the number of L2 cache entries tracked by the coherence protocol [8]. On a directory cache eviction, the entry with the lowest number of sharers is chosen and all sharers for that entry are invalidated.

To gain an understanding of performance tradeoffs for the DirCC protocol, we consider the average memory latency (AML), a metric that dominates program execution times in data-centric multicores.

$$AML_{DirCC} = cost_{\$access,DirCC} + rate_{\$miss,DirCC} \times cost_{\$miss,DirCC} \quad (1)$$

While $cost_{\$access,DirCC}$ mostly depends on the cache technology itself, the performance of the DirCC protocol primarily depends on $rate_{\$miss,DirCC}$ and $cost_{\$miss,DirCC}$. Many multithreaded parallel applications exhibit a variety of producer-consumer communication patterns for data sharing. When an application shows strong sharing patterns, DirCC performance is impacted in the following ways: (i) the directory causes an indirection leading to an increase in the cache miss access latency for both producer and the consumer, (ii) the automatic read data replication results in one address being stored in many core-local caches, reducing the amount of cache left for other data, therefore adversely impacting miss rates, and (iii) a write to shared data or a directory cache eviction requires all shared copies of the data to be invalidated, which results in higher miss rates and additional protocol latencies.

2.2 Execution Migration Based Coherence (EM)

The execution migration protocol always brings the *computation* to the data: when a memory instruction requests an address not cached by the current core, the execution context (architecture state in registers and the TLB) moves to the core that is *home* for that data. The physical address space in the system is partitioned among the cores, and each core is responsible for caching its region of the address space. When a core C executes a memory access for address A, it first computes the *home* core H for A. If $H = C$, it is a *core hit*, and the request for A is forwarded to the cache hierarchy. If $H \neq C$, it is a *core miss*; the core C halts execution and migrates the architectural state to H via the on-chip interconnect. The thread context is loaded in the remote core H and the memory access is now performed locally.

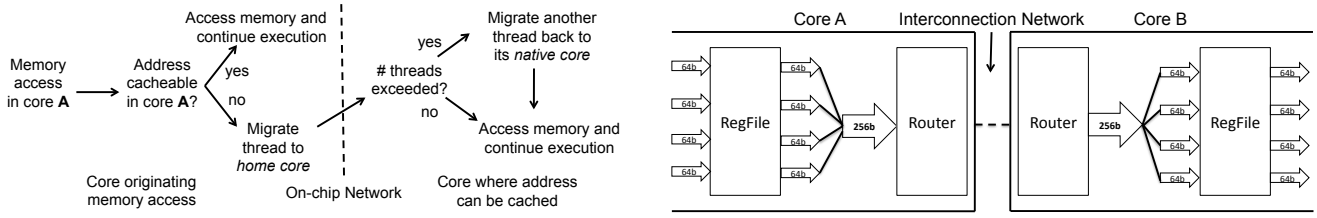
It is important to note that the EM protocol does not require a directory and its associated overheads. Although the EM protocol efficiently exploits spatial locality, the opportunities for exploiting temporal locality are limited to register values. As soon as the execution context migrates, the data that was local before the migration is now remote. Subsequently, any memory accesses to the same core become local. To gain an understanding of performance tradeoffs for the EM protocol, we consider the average memory latency (AML).

$$AML_{EM} = cost_{\$access,EM} + rate_{\$miss,EM} \times cost_{\$miss,EM} + rate_{core\_miss} \times cost_{migration} \quad (2)$$

While $cost_{\$access,EM}$ mostly depends on the cache technology itself, EM performance primarily depends on the other variables. Because data replication is prohibited, the cache miss rates are lowered significantly compared to DirCC. Additionally, the EM protocol does not suffer from directory indirections, and therefore reduces the latency of cache misses. However, the performance of the EM protocol is critically constrained by the frequency and cost of migrations.

The core miss rate is primarily dependent on the application's data sharing patterns. But the placement of data also plays a key role, as it determines the frequency and distance of migrations. Data placement has been studied extensively in the context of NUMA architectures (e.g., [9]) as well as more recently in the NUCA context (e.g., [6]). The operating system controls memory-to-core mapping via the existing virtual memory mechanism: when a virtual address is first mapped to a physical page, the OS chooses where the relevant page should be cached by mapping the virtual page to a physical address range assigned to a specific core. Since the OS knows which thread causes a page fault, the following heuristic works well: similar to a first-touch-style scheme [10], the OS maps the page to the first core to access it, taking advantage of data access locality to reduce the core miss rate while keeping the threads spread among cores. This allows the OS to assign a *home* core to each page; 4 KB pages can be used as in traditional operating systems.

Since the core miss cost is application-dependent and sometimes unavoidable, it is critical that the migrations be as efficient as possible. Therefore, we utilize the efficiency of in-hardware migrations and an intelligent one-way migration protocol to achieve the shortest possible migration latencies. What happens if the target home core is already executing another thread? We *evict* the executing thread and migrate it elsewhere before allowing the new thread to enter the core. Our migration framework features two execution contexts at each core: one for the core's *native* thread (i.e., the thread



(a) The migration protocol invoked on each *core miss*

(b) Microarchitecture support for a single execution context transfer

Fig. 3. A fast, hardware-level migration framework for the EM protocol.

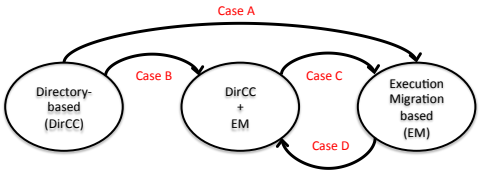


Fig. 4. The DCC architecture allows three modes of operation for cache coherence. Default is either DirCC or EM, and the third mode allows a hybrid scheme that selectively uses one of the two protocols. DCC operates at the granularity of OS pages.

originally assigned there and holding its private data), and one for a *guest* thread. When an incoming guest migration encounters a thread running in the guest slot, this thread is evicted to its native core. Figure 3a shows the protocol support for ensuring a successful migration, and Figure 3b shows the hardware support required to enable migrating a thread from one core to another. The greater the available network bandwidth, the faster the migrations. Therefore, our migration framework can be scaled by widening the router datapaths.

3 Dependable Cache Coherence Architecture

In this section we describe the details of the proposed dependable cache coherence architecture (DCC) that combines the directory protocol (DirCC) with a novel execution migration protocol (EM) to enable *architecturally redundant* cache coherence for the large-scale multicore era, where not only manufacturing fault rates are expected to be high, but many errors are likely to evolve during the lifetime of the processor. Because these schemes can co-exist and seamlessly transition at granularity of OS pages, DCC is capable of offering error-resilient hardware cache coherence.

3.1 Fault Detection and Reconfiguration

Fault tolerance techniques are generally categorized into detection/isolation followed by correction/recovery due to errors. This paper focuses on the fault correction/recovery aspect. However, fault detection and system reconfiguration are a necessary component of any fault tolerance scheme and may require additional complexity.

Traditionally, error detection and system reconfiguration is done during *manufacturing test*. Recently, researchers have also proposed several *online test* frameworks. For example, Constantinides et al. have proposed a software-based error detection and diagnosis technique, which is based on using firmware to periodically insert special instructions and test sequences for diagnosis [2]. Their architecture extends the existing scan chains to provide access to microarchitectural components for detecting errors and subsequently de-configuring the faulty components.

3.2 Architecturally Redundant Coherence

The DCC architecture enables runtime transitions between the two coherence protocols at the level of memory accesses, thereby enabling fine-grain redundancy with the goal of keeping user-level performance impact to a minimum. For each memory instruction, the translation look-aside buffer (TLB) provides the information about the home core for that address. Note that the EM protocol operates on an OS-page granularity (generally 4 KBytes) of the OS, whereas, the DirCC protocol operates at cache line (generally 64 Bytes) granularity.

Figure 4 provides an overview of the modes of operation for the DCC architecture. The various usage models for enabling dependability are discussed below:

3.2.1 *Case A*

If a directory controller or link is faulty, the address segments that this directory node is responsible for are transitioned to EM. Alternatively, if the system opts to transition between DirCC and the EM protocol at the application level, observing high cache miss rates can trigger this transition for all address regions.

3.2.2 *Case B*

This scenario allows a hybrid scheme where parts of the address segments are handled via DirCC, and parts via EM. If portions of the data cache are disabled because of memory errors, the pages with high cache miss rates can be transitioned to the EM protocol. In DirCC, a write to shared data requires an expensive invalidation of all shared copies, whereas EM involves no such invalidations. Detecting pages with high invalidation rates can also guide transition of such pages from DirCC to EM.

EM performance is dominated by the overheads of migrations; therefore, we expect the core miss rate to determine a transition of some pages back to the DirCC protocol. Also, because DirCC performs well under high read data sharing and low write frequency, detecting such pages can trigger EM to DirCC transitions.

3.2.3 *Case C*

If the directory cache is unable to maintain the coherence requirements (for example, large portions of the directory cache are faulty), or the directory controller (or link) becomes faulty at runtime, the address segments that this directory node is responsible for are transitioned to the EM protocol.

3.2.4 *Case D*

If a migration controller or migration link is faulty for a tile and the directory controller is alive, the pages assigned to the corresponding core are transitioned to the directory protocol.

In the worst-case scenario where both coherence mechanisms are broken, the processor is interrupted to invoke an OS-level reassignment of address segments to working nodes in the system.

3.3 Coherence Protocol Transitions

In this section we discuss the DCC architectural mechanisms to enable transitions between DirCC and the EM protocols. Each transition is performed at the minimum granularity of the OS page size.

3.3.1 *DirCC → EM*

When a page is selected for transition to the EM protocol, the directory sends a special broadcast message to all cores. This message at each core takes the following actions: (a) flush the cache lines (if modified state), (b) update the TLB for the associated page and mark it stalled for EM (note that on initial page allocation, the OS is expected to assign a home core and set EM to disabled by default), and (c) send a reply message to the directory. The directory waits for all replies from each core and in parallel invalidates all the related cache lines (associated with the page) in the directory cache. Any transient requests to access this page results in the directory initiating a reply message to the requesting core to retry. When all replies

are accounted for at the directory (implying all transient requests have been observed and cleaned up at the directory), another special broadcast message is sent to all cores. This updates the TLB to enable the EM protocol. During the time between stalled and enable mode, the TLB blocks new memory accesses. When the page is transitioned to EM, subsequent accesses to the page now result in the TLB identifying the home core for the access and EM takes over for that page.

3.3.2 EM → DirCC

Because EM only allows a single core to cache any data associated with a page, the migration controller at the home core (for the page being transitioned) initiates a cache flush for the addresses in the page. A special broadcast message is sent to all cores to update the TLB for this page to EM disable. The home core waits for all replies for this broadcast. This allows all inflight migrations to be observed at the home core, which results in an update to the page's TLB for each thread and eviction of the thread back to its native core. For any subsequent memory access initiated for this page, the request is forwarded to the associated directory controller.

4 Evaluation

We used the Graphite simulator [11] to model the DirCC and EM protocols. We implemented a tile-based multicore with 256 cores; various processor parameters are summarized in Table 1. On-chip directory caches (not needed for EM) were set to sizes recommended by Graphite on basis of the total L2 cache capacity.

4.1 LU

We first evaluate the average memory latency (AML) of the SPLASH-2 LU_NON_CONTIGUOUS benchmark. This benchmark exhibits strong read/write data sharing, which causes mass evictions of the cache lines actively used by the application. At the same time, replication of the same data in many core caches adds to the cache capacity requirements in DirCC. A combination of capacity and coherence misses results in a 9% cache hierarchy miss rate for the DirCC, while the EM protocol eliminates all such misses and only incurs 0.1% compulsory misses. Together, the caches and the off-chip components contribute 17.8 cycles/memory-access for DirCC and 2.4 for the EM protocol.

The DirCC cache misses also incur a latency of several round-trip messages due to directory indirections: reads that no core holds exclusively take at least two, while accesses requiring broadcast invalidations incur many messages. The measured cost of the directory protocol comes out to 17.4 cycles/memory-access. On the other hand, migrations add to the cost of accessing memory for the EM protocol. We measured a core miss rate of 65% for the LU_NON_CONTIGUOUS benchmark. With an average of 12 network hops per migration, we measured an overhead of 51 cycles per migration, resulting in an amortized cost of 26 cycles/memory-access. Overall, the AMLs of DirCC and the EM protocol come out to 35.2 and 28.4 cycles/memory-access, respectively. This corresponds to a 1.25× advantage for EM over the DirCC protocol.

4.2 Raytrace

Next, we evaluate the AML of the SPLASH-2 RAYTRACE benchmark. This benchmark is dominated by read-only data sharing and a working set that fits our data caches. Therefore, the DirCC protocol observes a relatively low cache hierarchy miss rate of 1.5%, while the EM protocol incurs 0.3% cache misses. Together, the caches and the off-chip components contribute 3.1 cycles/memory-access for DirCC and 3.0 for the EM protocol.

Since DirCC observes very few and mostly inexpensive directory indirections, the measured cost of the directory protocol comes out to 2.7 cycles/memory-access. The core miss rate in RAYTRACE is 29%. With an average of 11 network hops per migration, we measured an overhead of 47 cycles per migration, resulting in an amortized cost of 12 cycles/memory-access. Overall, the AMLs of DirCC and the EM protocol come out to 5.8 and 15 cycles/memory-access, respectively. This corresponds to a 2.6× advantage for DirCC over EM.

Parameter	Settings
Cores	256 in-order, single issue cores
L1 I/D cache/ core	32/16 KB, 4/2-way set associativity
L2 cache/ core	64 KB, 4-way set associativity
Electrical network	2D Mesh, XY routing, 256b flits 2 cycles per hop (+ contention delays) 1.5 Kbits execution context size [13]
Data Placement	FIRST-TOUCH, 4 KB page size
Directory Coherence	MOESI protocol, Full-map physically distributed 10 MB directory, 16-way set assoc.
Memory	30 GB/s bandwidth, 75 ns latency

TABLE 1
DEFAULT PROCESSOR CONFIGURATION

Our initial evaluation results indicate that depending on the data sharing patterns of an application, either cache coherence protocol can perform better than the other. We note that protocol-specific application transformations are possible for either protocol, and optimizations for the EM protocol have been developed [12]. The proposed DCC architecture is not only poised to enable a dependable, architecturally redundant cache coherence mechanism, but it can be designed to enhance system performance by intelligently choosing either the DirCC or the EM protocol at the granularity of applications or OS pages.

5 Conclusion

Today microprocessor designers are investing heavily into means of detecting and correcting errors at the expense of area, power and performance. In this paper we have proposed a novel dependable cache coherence architecture (DCC) that provides architectural redundancy for maintaining coherence between on-chip caches.

References

[1] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," in *IEEE Micro*, 2005.

[2] K. Constantinides, O. Mutlu, T. Austin, and V. Bertacco, "Software-based online detection of hardware defects: Mechanisms, architectural support and evaluation," in *MICRO*, 2007.

[3] M. D. Powell, A. Biswas, S. Gupta, and S. S. Mukherjee, "Architectural core salvaging in a multi-core processor for hard-error tolerance," in *ISCA*, 2009.

[4] O. Khan, M. Lis, and S. Devadas, "EM^2: A Scalable Shared-Memory Multicore Architecture," *MIT-CSAIL-TR-2010-030*, 2010.

[5] S. Borkar, "Thousand core chips: a technology perspective," in *DAC*, 2007.

[6] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive NUCA: near-optimal block placement and replication in distributed caches," in *ISCA*, 2009.

[7] D. Chaiken, C. Fields, K. Kurihara, and A. Agarwal, "Directory-based cache coherence in large-scale multiprocessors," in *COMPUTER*, 1990.

[8] A. Gupta, W. Weber, and T. Mowry, "Reducing memory and traffic requirements for scalable directory-based cache coherence schemes," in *ICPP*, 1990.

[9] B. Verghese, S. Devine, A. Gupta, and M. Rosenblum, "Operating system support for improving data locality on cc-numa compute servers," *SIGPLAN Not.*, 1996.

[10] M. Marchetti, L. Kontothanassis, R. Bianchini, and M. Scott, "Using simple page placement policies to reduce the cost of cache fills in coherent shared-memory systems," in *IPPS*, 1995.

[11] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald *et al.*, "Graphite: A distributed parallel simulator for multicores," in *HPCA*, 2010.

[12] K. S. Shim, M. Lis, M. H. Cho, O. Khan *et al.*, "System-level optimizations for memory access in the execution migration machine (EM^2)," in *Workshop on Computer Arch. and Operating System co-design*, 2011.

[13] K. K. Rangan, G. Wei, and D. Brooks, "Thread motion: fine-grained power management for multi-core systems," in *ISCA*, 2009.