The Power of Priority: NoC based Distributed Cache Coherency

Evgeny Bolotin, Zvika Guz, Israel Cidon, Ran Ginosar and Avinoam Kolodny
Electrical Engineering Department,Technion - Israel Institute of Technology, Haifa 32000, Israel

ABSTRACT

The paper introduces Network-on-Chip (NoC) design methodology and low cost mechanisms for supporting efficient cache access and cache coherency in future high-performance Chip Multi Processors (CMPs). We address previously proposed CMP architectures based on Non Uniform Cache Architecture (NUCA) over NoC, analyze basic memory transactions and translate them into a set of network transactions. We first show how a simple, generic NoC which is equipped with needed module interface functionalities can provide infrastructure for the coherent access of both static and dynamic NUCA. Then we show how several low cost mechanisms incorporated into such a Vanilla NoC can facilitate CMP and boost performance of a cache coherent NUCA CMP. The basic mechanism is based on priority support embedded in the NoC, which differentiates between short control signals and long data messages to achieve a major reduction in cache access delay. The low cost Priority-based NoC is extremely useful for increasing performance of almost any other CMP transaction (i.e. uncached and cache-coherent R/W, search in DNUCA, isolating low priority traffic, synchronization and mutual exclusion support). Priority-based NoC along with the discussed NoC interfaces are evaluated in detail using CMP-NoC simulations across several SPLASH-2 benchmarks and static web content serving benchmarks showing substantial L2 cache access delay reduction and overall program speedup. For further system improvements, we introduce additional low cost NoC mechanisms that include: virtual invalidation rings, efficient store-and-forward multicast for short messages which is embedded within a wormhole NoC, and a cache-line search mechanism for the efficient operation of dynamic NUCA. These mechanisms can also expedite not only cache coherency but also other basic CMP transactions such as search and serialization primitives support.

1. Introduction

Microprocessor architecture is in transition towards multi-core architectures that exploit thread-level parallelism, and provide performance improvements as well as power-efficiency. Such chip multi-processors (CMPs)[1-7] need to employ large shared on-chip cache memory (typically a L2 cache). The cache must support parallel transactions with multiple cores. Hence, a distributed cache, comprised of multiple memory banks interconnected by a network on chip (NoC)[9,31-43], as illustrated in Figure 1, is an accepted and likely approach. In such a structure, the effective access time to the shared cache will become a major performance bottleneck, as both the number of cores and the number of clock cycles required for signal propagation across the die will increase with technology scaling.

This architecture raises many challenges because the system depicted in Figure 1 needs to efficiently discover the cached location of each physical memory address and maintain multiple data copies, while ensuring data coherency of shared data among all the cores. Traditional snooping protocols for cache coherency [25] are not suitable for implementation over a NoC, and are not scalable with the number of cores. Directory-based coherence protocols require multiple network traversals (e.g. to search the cached location, determine the sharing status, update or invalidate etc.). Consequently, a CMP equipped with a standard NoC and standard processor and cache network interfaces may incur large delays in cache transactions.

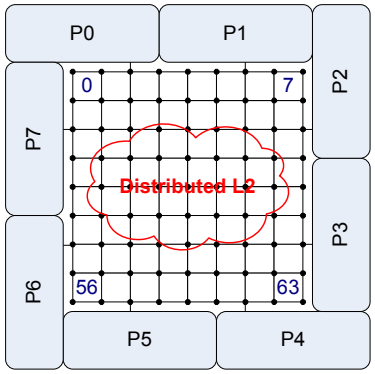

Figure 1. Modern CMP System interconnected by NoC :8 CPUs along with L2 Cache distributed in 64 Banks

Previous CMP research mainly addressed the principal architectural issues of distributed shared CMP cache over a NoC abstraction [1-4, 6-7]. In the evaluation of SNUCA and DNUCA [1,2] the authors make simplifying assumptions regarding network delays and behavior. They do not evaluate any detailed NoC design or optimize the NoC for supporting typical cache operations.

There has been substantial prior work in the area of cache coherency optimization in the context of multiprocessors [11-18]. The majority focused on in-protocol optimizations, releasing consistency model and speculation [10-13]. Some approaches combined snooping and directory-based protocols [12]. Several studies looked into broadcast and multicast snooping, and ring optimizations [14-17]. In [18] the authors tried to efficiently map a coherency protocol onto physical wires in several metal layers, with different widths and thicknesses. The token coherence method [19] suggests to exchange and count tokens to control coherence permissions.

Several recent papers focused on NoC–based CMP cache

coherency [21-23]. [23] explores the performance vs. energy tradeoffs for hardware and software snoop-based protocols in MPSoC. Their evaluation assumes a shared-medium interconnect among processors and shows that hardware solutions based on snooping are power inefficient. It does not address directory based coherence schemes which are the topic of our work.

Eisley et al. [21] addressed the cache coherency problem in CMP and proposed to alter the standard directory-based system by directories implemented inside NoC routers (in-network approach). In the proposed architecture, routers may steer requests towards nearby data copies. This approach enables to reduce memory access delay but requires additional storage and a more sophisticated router architecture to perform directory-related manipulations on every packet at every hop. An alternate solution, [22], proposes a software solution for memory coherency in MPSoCs. The approach relies on the programmer intervention for mapping local and shared variables and segments. Using uncached accesses for the shared segments, the cache coherency and memory consistency can be maintained at no hardware cost.

The main contribution of this paper is the introduction of hardware based NoC priority mechanism for efficient distributed directory-based cache-coherent access in both static and dynamic NUCA systems, termed Priority-based NoC. The main power of Priority-based NoC is its simplicity, low hardware cost and its generic nature that can extend and accelerate many other operations. The major cache-access delay reduction is achieved by differentiating short control messages that constitute the coherency protocol and allowing these short messages to bypass long data packets while preserving the coherence protocol correctness. The same idea can apply to other common CMP memory and synchronization transactions such as regular (uncached) R/W, DMA and common variable synchronizations [26]. Our solution has a very low hardware cost and it does not impact the router architecture as in [21]. Moreover, our approach is a complementary solution, and can be applied to provide additional speedup to the in-network directory of [21], or to the software messages of [22], as well as to other cache coherence protocols.

In addition, we describe other new NoC mechanisms and services for a further improvement of efficient coherent cache-access in NUCA, termed an Advanced-Services NoC. These include virtual wormhole invalidation rings for well-organized invalidation procedure, efficient store-and-forward multicast for short messages which is embedded within a wormhole NoC, and support for serialization and mutual exclusion primitives such as locks and Test&Set. For DNUCA based CMPs we introduce a cache-line search optimization mechanism that is required for efficient operation. Note that these additional mechanisms are also generic and can be used beyond the context of cache coherency.

The rest of the paper is structured as follows. In Section 2, we give a short background of CMP NUCA. Then, we analyze cache coherency memory transactions and translate them into a set of network transactions. Then, we show how the basic-functionality Vanilla NoC needs to be equipped with special NoC module interfaces in order to support cache-coherent communication of both static and dynamic NUCA. In Section 3, we turn to introduce Priority-based NoC and Advanced-Services NoC for boosting performance of coherent distributed cache access and reducing its latencies. In order to illustrate the generic nature of our simple hardware solutions we also explain their usage beyond the cache coherency paradigm, for normal (uncached) R/W and for serialization primitives support such as locks and Test&Set. In Section 4, our Priority-based NoC is evaluated in detail using cycle-accurate CMP-NoC simulations across several benchmarks. Section 5 summarizes the paper.

2. CMP NUCA Background

CMPs are shifting towards a NUCA [5], where the cache is divided into multiple banks, and accesses to closer banks result in shorter access times. In NUCA, performance depends on the average (rather than worst-case) latency. To further reduce the average access time, the authors of [5] have suggested the use of dynamic block migration, called Dynamic NUCA (DNUCA). In DNUCA, every access to a block moves the block one step closer to the processor, thus gradually reducing distances and access times to frequently-used data. This differs from the basic Static NUCA (SNUCA) design, where block placement is static, determined by address.

Several works have dealt with NUCA based CMP systems [1,2,4,7]. [1-2] have studied both SNUCA and DNUCA implementations for CMP. [3-4] suggested replicating shared blocks within the cache in order to improve their proximity to all sharing cores. [7] achieved vicinity for shared data by changing the common cache-in-the-middle CMP layout and by using a designated part of the cache capacity for shared blocks only. Using DNUCA, blocks may reside in different locations within the cache and hence a mechanism for locating blocks is needed. Search policies may vary from sequentially inquiring every bank to flooding of the interconnect in parallel, or use a hybrid intermediate policy such as phase-multicasting [1,5,6]. Since the search introduces significant delays to the cache access time and overload the interconnect, some search hints are required to direct the search to specific banks and accelerate detection of cache miss. [6] has leveraged bloom filters to devise such a complexity-effective search mechanism for CMPs.

3. NoC support for CMP NUCA

In this section we first describe a straightforward architecture of NUCA CMP over a Vanilla NoC. We briefly describe the CMP memory architecture, basic CMP communication infrastructure and details of directory-based

distributed coherency protocol over NoC. We also outline common L2 cache access transactions which are translated into multiple network transactions. We sketch the basics of the necessary NoC communication interfaces that support coherent cache accesses over NoC.

Then we show how by using a Priority-based NoC which is equipped with a simple priority mechanism we can drastically decrease cache access latency and speed up the total program running time. Finally, we outline further possible Advanced-Services NoC mechanism for efficient cache-access and overall CMP performance improvement such as: special broadcast and multicast mechanisms, ways for faster search in DNUCA, supporting synchronization primitives in CMP and others.

3.1 Cache-Access in NUCA over Vanilla NoC

In a CMP system such as in Figure 1 there are two levels of cache hierarchy: shared L2-cache and private L1-caches (within the processor cores) that maintain copies of L2 data and may need to modify it. Therefore, we need to provide mechanisms to keep this non-uniform memory system in a coherent state and support sequential consistency. In other words, we need to support total ordering among memory transactions in the parallel system as it would be executed on a sequential system [8].

Common snooping-based cache-coherency approaches[8] are not suitable for NoC and are not scalable with the growing number of nodes. Therefore, we focus on a directory based approach. This approach eliminates the need for using a slow and expensive shared bus or NoC broadcast. The directory serves as a serialization point for maintaining coherency.

We focus on a distributed directory scheme which is more scalable than a central directory approach. In practice, the distributed directory is implemented by extending each L2-cache-line (block) with the directory information, which tracks the state of this block. The directory information contains a *status vector* containing the identity of processors that store this cache-line in their L1 caches. It also contains a *modified* bit to indicate that this cache-line is in modified state in one of the L1 caches. In this work we target the four-state (MESI) write-back invalidation protocol [8].

L2-caches and directory deal with incoming transactions in-order for maintaining transaction consistency [8]. Since out of-order mechanism require a considerable additional hardware costs and protocol verification we do not address such systems in this paper.

We assume that the network maintains the ordering of messages for each source-destination pair. Therefore a Vanilla NoC would be equipped with a single service level (SL) [9] and virtual channel (VC) , and would perform static order-preserving packet routing.

When a processor performs a L2 cache transaction (upon L1 miss) it is translated into multiple transactions over the NoC. A basic read transaction by P0 is depicted in Figure 2. It is first translated into a read request packet and sent over the NoC towards a L2 node according to the address of the block for SNUCA, or after a search procedure for DNUCA. If the block is missing at the home node (L2-miss) then the block is fetched from the external memory, also via the NoC. Otherwise, if the block exists in the L2 cache, several scenarios are possible according to the state of the block which is stored in the directory. If the block is not in modified state, then L2 responds with read response packet which carries the desired cache-line (indicated by a bold arrow) back to the requestor P0 and sets the bit in the status vector indicating that the block is shared by P0.

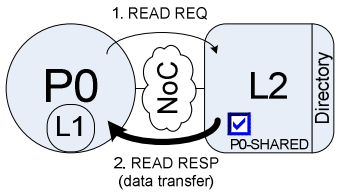

Figure 2. Read Transaction over NoC – the block was in shared state in directory and remained shared

Alternatively, if this block was previously read exclusively (with write permission) by some other processor P2 (see Figure 3), then a simple read transaction by P0 will lead to a write back procedure indicated by the red dotted arrows in Figure 3. The write back procedure consists of a request packet that is sent to the modifying node P2 and it consequently sends back the desired block to the home node L2 and it forwards it to the requestor P0.

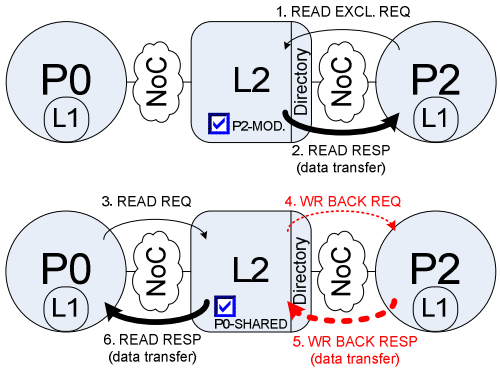

Figure 3. Read Transaction over NoC with write back procedure: the block was in Exclusive state due to an exclusive read by P2 and had to be written back before responding to P0

Exclusive read request to a block in L2 will behave slightly differently. Similarly to a regular read request it will cause a main memory read in case of L2-miss, regular read response in case that the block is not used by any other processor, and write back procedure in case that some other processor is modifying this block. However, if the block is shared by several other processors (Figure 4), the directory has to stall the transaction, send invalidation messages to all sharing processors and only after receiving invalidation acknowledge from all the sharing processors, it is allowed to respond with the requested cache line (Figure 4) and to

mark the directory status of this block as modified.

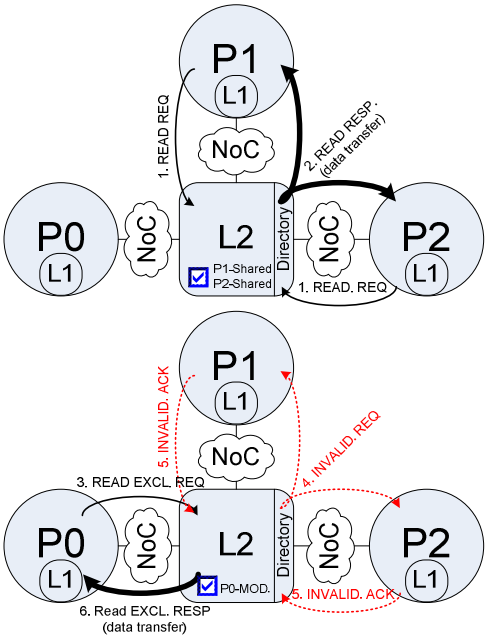

Figure 4. Read Exclusive Transaction over NoC: the block was in Shared state in directory and it causes invalidation procedure

Another possible transaction towards L2 is a write transaction which happens as a result of L1 eviction. It is not acknowledged to the processor and the only possible outcome of this transaction can be a successive eviction of L2 to main-memory.

Vanilla NoC for Coherent CMP NUCA:

The previously discussed distributed CMP cache transactions can be carried out by regular unicast packets transmitted over any wormhole based multi-hop Vanilla NoC [9,31-43] without implementing any special mechanisms besides network interfaces for processors and L2-caches. The Vanilla NoC should also preserve packet ordering by the means of static packet routing and single SL and VC for each physical link.

Vanilla NoC Interfaces for CMP:

Assuming simple coherence-preserving and in-order processor interface, each read request stalls the requesting interface from sending new requests until it receives a response message from L2. In addition, each write back and invalidation procedure stall the L2 home node and its directory from dealing with other pending requests, since it has to preserve total command order. On the other hand there is a need to prevent system deadlocks, and therefore the processors and L2 banks must be able to consume request messages and respond to invalidation and write back requests.

Processor Interface Architecture:

The processor interface manages two packet queues. The first is a source queue, which contains transactions that are originated from the local CPU and waiting to be transmitted towards L2 via the network. The second queue is a response queue which stores the response packets for received requests (invalidation and write back requests) that are ready to be transmitted over the network. The processor interface is responsible for packetizing the local CPU L2 memory transactions, enqueing them in the source queue and scheduling the queue for transmission over the network not before receiving a response to a previously transmitted request packet and upon available buffers in the adjacent network router as in regular wormhole NoC scheduling [9]. Therefore, the scheduling of the source queue is stalled upon sending read or read exclusive requests, and is awakened after receiving read data from L2. Upon receiving a write back request or an invalidation request it immediately forwards it to the cache controller and enqueues a response packet in the respond queue. In addition, the scheduler must give priority to the respond queue over the source queue in order to eliminate protocol deadlocks. In other words, the processor interface must first respond to all received requests and only then it turns to send its own transactions.

L2-cache Interface Architecture:

The L2 network interface also manages two queues. The first queue contains incoming requests from the network, such as read and read exclusive. The second queue, stores the response packets for already processed requests. The mission of L2 is to serve as a serialization point and treat incoming requests in-order. Therefore, when L2 is in the middle of write back or invalidation procedures (waiting for write back of cache-line or for multiple invalidation acknowledges) it is not processing any arriving requests. In this state the L2 interface is stalled, meaning that L2 is not processing requests from its request queue and is not enqueueing any new responses. L2 interface enters this state upon processing read or read exclusive request for a block that the directory shows that it is modified state and needs to be written back, or upon processing a read exclusive request for a block that the directory shows that it is shared and needs to be invalidated. Otherwise, if the requested block is not shared and not modified, an immediate response is enqueued into the respond queue and L2 is not stalled.

The Vanilla NoC which preserves ordering, combined with the necessary processor and L2 interfaces, provides the needed mechanisms for memory coherent L2 access. However, as described in the previous section, each memory transaction results in a series of network transactions which dominate the delay of cache access in distributed CMP systems. Therefore in the following sections we describe the possible NoC-related mechanisms that will shorten the cache access time and provide total program speedup within insignificant hardware cost.

3.2 Priority-based NoC

We start with the following observations:

Observation A:

L2-cache total access delay is a summation of the following

delay components: the queuing time at the processor interface, the round-trip delay of request and response messages between processor and L2-cache bank over the NoC, the queuing time in L2 incoming requests and outgoing response queue and the round trip delay of write back and invalidation procedures.

Observation B:

All NoC transactions which constitute the L2-cache access procedure are of an equally very high importance since they directly contributed to the delay period which separates between the processor and its desired data (L1 cache-miss).

Observation C:

L2-cache accesses consist of two types of messages: first a short control messages (either request or acknowledge), and second, long messages that carry the cache line (64 bytes and additional overhead – in our example).

From analyzing observations A-C, we propose to differentiate between short control messages and long data messages by equipping the NoC with multiple priorities similar to QNOC[9], and by giving a higher priority to the (short) control messages over the (longer) data messages. Although according to observation B all messages are of the same importance, giving priority to short messages significantly decreases their delay without a large impact on the delay of long data packets. This is especially true in the case of a wormhole NoCs where short messages can be blocked behind long worms that are not even destined to the same destination nodes.

There are three types of short messages in our system: read or read exclusive requests, write back request and invalidation request or acknowledges. By giving priority to each type of messages we can speed up different phases in the coherent cache-access protocol. For example, a read request message with higher priority will reach L2 earlier and cause an earlier response, either a data response, a write back or an invalidation procedure. Thus, leading to a total shorter transaction round trip delay in the NoC, that is translated into a shorter stall time at the processor interfaces, leading to shorter queueing delays in the processor transmitting queues.

Similarly, by giving a higher priority to write back and invalidation request messages and acknowledge messages, we can not only reduce the round trip delays of write back and invalidation procedures which are components of total transaction delay, but also reduce the stalling time of the L2-cache. This allows the L2-bank to start serving the next pending request earlier, leading to a minimal overall L2 cache-access delays and an overall system speedup.

Interface Support for Coherence Protocol Correctness:

A Priority-based NoC with multiple priorities can no longer provide in-order delivery among packets that are transmitted with different priorities. As a result, a system which uses Vanilla NoC interfaces becomes vulnerable to coherency protocol failures. Figure 5 shows an example of a possible failure. A distant processor P0 requests a cache-line from directory (1) using high priority request, as a result the directory responds with a long and low priority message (2). Meanwhile, a nearby processor P1 performs an exclusive read (3) of the same cache-line, resulting in a high priority invalidation message (4) which may reach processor P0 before the low priority read response. A Vanilla processor interface would invalidate the cash-line, and reply with invalidation acknowledge towards L2-bank. Later, when read response (2) finally reaches P0, its processor stores this cache-line in the local L1 and P0 consumes it despite the fact that P1 already modified it.

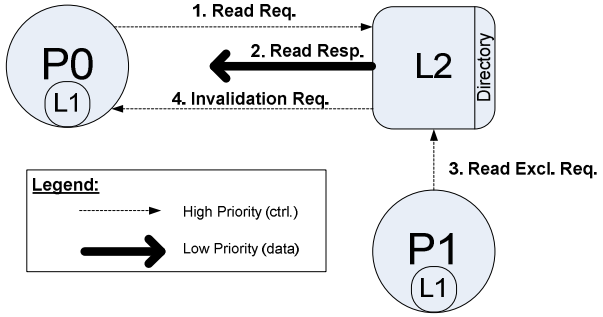

Figure 5. An example illustrating the need for serialization of transactions at processor interface for Priority-based NoC

This is clearly a coherency violation. The coherency protocol fails because the invalidation arrived before the read response. A similar problem may occur when a processor receives a write back request before a read response. A simple solution to this problem is a state-preserving serialization of transactions in the processor interface. The processor interface should not immediately invalidate or write back upon every request without checking first whether this invalidation or write back is for a block that the processor just requested and did not get a response yet.

In summary, we propose to minimize the overall L2-cache transaction delay by implementing a low cost priority mechanism in NoC and applying it for short control messages. This approach does not only minimize the average NoC traversal delays but also minimizes the queuing delays in the processor and the L2-cache interfaces. The Priority-based NoC approach is a generic method for efficient CMP communication. It can support other variations of cache-coherency protocol and can be literally useful for any other CMP related communication tasks, such as search in DNUCA, isolating low priority traffic (such as prefetch and DMA) out of high priority traffic, synchronization and mutual exclusion support primitives support (see next Sections).

3.3 Advanced–Services NoC

Although Priority-based NoC is a powerful tool for CMP communication, we explore further advanced services and mechanisms for distributed NUCA CMP that can be added on-top of the Priority-based NoC. Advanced-Services NoC is a portfolio of solutions that a system architect can decide

to use for boosting performance and saving power in CMP.

3.3.1 Special Broadcast for Short Messages

Observation D:

Invalidation procedure is unique since it might require sending invalidation messages to multiple processors that share the specific cache-line and gathering invalidation acknowledge messages from them. It becomes an important L2 cache-access delay component in programs having a large amount of write-sharing among the processors.

Therefore, in addition to the proposed priority mechanism which leads to a substantial speedup, one might also want to use a more efficient multicast or broadcast schemes instead of multiple unicast messages (which are the only type supported by the Vanilla NoC). However, broadcast-based invalidation is an undesired solution for a wormhole NoC. Wormhole broadcast is deadlock-sensitive and extremely slow especially when several broadcast trees coexist in the network. The wormhole broadcast tree traversal time in the network is dominated by the speed of the slowest leaf. In our CMP system each memory bank is a potential source of invalidation broadcast. Therefore, there is a need to provide deadlock-freedom (by adding additional VCs). In addition, multiple broadcast trees may also slow each other.

Store & Forward Broadcast Embedded in Wormhole NoC:

Because of the complications described above, we propose to enhance the wormhole NoC router with a message replication mechanism for short control messages only. In this way, we achieve a performance of store and forward (S&F) broadcast by a small hardware investment, as we do not increase router buffering.

The sketch of the enhanced router architecture is as following. A generic input-queued wormhole router[9][27] consists of several flit buffers for better performance of the wormhole pipeline[27]. The length of control messages is few flits only, which would fit into such a queue. In a wormhole router the flow control is performed on a flit by flit basis, i.e. flit-based flow control. For implementing S&F broadcast the output port of the router schedules the packet only when there are enough buffers to contain the whole packet, i.e. packet-based flow control. The input buffer management logic does not remove the packet from the input buffer before it is transmitted over all scheduled output ports.

The broadcast routing is very simple in mesh topology. It can be performed in XY-like manner, (see Figure 6) while the routers along X-axis would have to replicate packets up and down and forward it further in the X-direction. The routers along the Y-direction of packet propagation only forward the packet without replication. On the other hand, the long packets carrying data are transmitted using the regular wormhole mechanism.

This broadcast service for short control messages can be enhanced with a priority mechanism and used for invalidations as well as for other kinds of system traffic, i.e. search in DNUCA, synchronization messages and more.

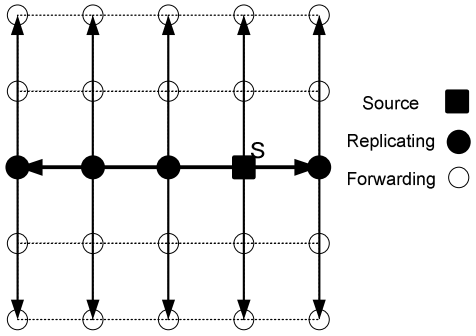

Figure 6. XY Based S&F Broadcast in wormhole NoC

Another mechanism that is useful in CMP is message consolidation. Consider an example of gathering invalidation acknowledge messages from all invalidated processors in a more efficient way than unicast. A router that supports message consolidation (a.k.a. gather function) maintains a state for each active broadcast at each output port. The acknowledge messages return from the broadcast tree leaves via the same route as the original broadcast tree. Once all acknowledge messages from all ports that have transmitted the original broadcast are received, the router invokes a single acknowledge message towards the root. This approach minimizes the amount of messages during invalidation cycle and reduces invalidation delay and total power dissipation.

Broadcast on a Virtual Ring:

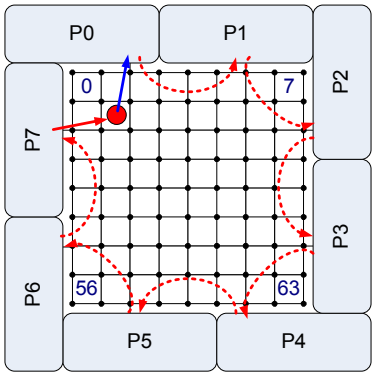

Figure 7. Virtual Invalidation ring implemented on top of existing NoC

Another approach for efficient invalidation can be implemented using a special multicast solution, termed virtual-ring. It is formed among the former broadcast tree leaves (the processors in our case). A virtual ring can be built on top of the existing network at almost no additional hardware cost. The basic idea is shown in Figure 7. Upon invalidation, L2 bank sends a single invalidation request to the nearest processor marked as ring invalidation message. When a processor network interface receives ring invalidation message it invalidates its local L1-cache and immediately forwards the message towards the next processor in the ring. The invalidation message can carry a counter of the already visited processors and then to visit all

processors in the ring. Otherwise, it can carry the sharing status vector which will indicate which processors really need invalidation, and the invalidation ring would be dynamic according to the vector value. When the ring invalidation message reaches the last processor in the ring it is forwarded to the L2-bank that originated the invalidation procedure. This last message also serves as invalidation acknowledge consolidation for all processors in the ring.

Again, this technique can be combined with our priority mechanism and the ring invalidation messages can traverse the network at higher priority than the long data messages. The virtual ring approach minimizes the number of messages and consequently reduces power consumption. However, the virtual ring can become a longer path than sending small number of invalidation messages and its acknowledges using unicast. Therefore the efficiency of this approach is a function of the number of sharing processors that need to be invalidated. In such cases, one can use a hybrid approach. For instance, a L2-interface can decide to send invalidation using broadcast or unicast as a function of the number of processors that are about to be invalidated. If the number of processors is below some threshold number it will send several unicast messages. Otherwise it will use the virtual ring invalidation.

3.3.2 Optimized Search over NoC in D-NUCA

Observation E:

Using DNUCA, blocks may reside in different locations within the cache and hence a mechanism for locating blocks is needed. Therefore the delay of search procedure in D-NUCA can become an important factor which constitutes the cache- transaction delay

In the Vanilla NoC, broadcast search can be implementing only by sending multiple unicast search request messages to all memory banks. We propose two approaches for improving upon the Vanilla NoC.

Priority Search:

Since without knowing the placement of the desired block in DNUCA the processor is stalled, it is of very high importance that the search would be executed as fast as possible. Therefore we can allocate a special priority for short search messages to allow them to bypass long data messages.

S&F Broadcast Search

The S&F broadcast which was described above, can be useful for flooding the short search messages along the grid of L2 banks. It would drastically decrease the number of search messages in the system which would lead to a faster search procedure along with less interference and delays for other messages in the system. In addition it will reduce the power consumption of the search procedure.

3.3.3 Synchronization and Mutual Exclusion support

In NoC-based CMPs, similarly to regular multiprocessor machines, the issues of point-to-point, global (barrier) synchronization and mutual exclusion among distributed processes is very important. Mutual exclusion operations are usually supported by hardware lock mechanisms and atomic primitives such as Test&Set and its sophisticated derivatives. Synchronization algorithms are also implemented using locks.

When a processor is spinning on Lock (testing a lock), the processor gets L1 cache miss, brings the current value of the lock to its local L1 cache from L2-home node and it keeps spinning on its value locally until the value of the lock is changed and the processor gets a notification about this via the standard invalidation of the lock in the local L1 cache. Therefore, at first glance it seems that there is not much space for optimization for locks in CMP, since the spinning is performed locally. However, whenever a lock which is tested by many processors is released, the waiting processors compete to take hold of it. In the case of barriers, many processors must be synchronized and then released. This is a typical case for barrier synchronization where each processor busy-waits on a locked counter to see if it reached a certain value. Such activity often causes hot spots in the memory system interconnect.

It can be seen that synchronization algorithms will perform well under low-contention periods over NoC, but under high-contention periods there is a need to incorporate additional mechanisms into NoC and NOC interfaces for more efficient synchronization. There has been already a substantial work performed in this field of software and hardware support for Locks in the context of multiprocessors and CMPs [24-26].

Most of the hardware methods mainly focus on maintaining a list of nodes waiting on a lock. It is maintained entirely in hardware and the releaser grants the lock to one of the waiting nodes on a list without affecting others. Implementing such subscription mechanism for lock at NoC interface over Priority-based NoC will eliminate busy wait over network. It may reduce the lock handover time as well as the interference of lock traffic with data access and coherence traffic.

4. Numerical Evaluation

We evaluated the schemes proposed in Section 3 through simulation. In Subsection 4.1 we present the simulation environment used, which includes CMP and NoC simulators and a description of the evaluated CMP benchmarks. In Subsection 4.2 we present the simulation results that demonstrate the advantages of the priority-based NoC approach over the Vanilla NoC in terms of cache-access delay and overall program speedup.

4.1 Simulation Environment

4.1.1 CMP and NoC Simulators

The NoC-based CMP system illustrated in Figure 1 is fully modeled by combining cycle-accurate NoC and cycle-approximate multi-processor system simulator. Simics [28] is used for simulating parallel programs execution in CMP

and producing L2 access traces which are fed into an OPNET-based NoC simulator [9]. NoC simulator is used for accurate measurement of cache-access transactions delay including coherency protocol, network communication delays and delays due to contentions on shared network and CMP resources.

The system that is modeled in Simics comprises a 8-processor CMP design, using the x86 in-order processor model as a building block. Each processor has a private, 2-way set associative, 64KB L1 data cache, and all processors share a 16MB, 16-way set associative L2 data cache. (Instruction caching is not implemented, but instructions are assumed to be available with no delay). Cache block size is 64 bytes for both L1 and L2.

The L2-access traces from Simics include L2-transactions and the inter-transaction delays, which represent the times at which the processor uses its L1-cache without accessing L2. These traces are fed into the OPNET-based tool which simulates the full directory-based coherent cache-access mechanism over NoC which was described in Section 3.

The developed OPNET models include the Vanilla and the Priority-based NoCs and the NoC interfaces for coherent CMP communication (see Section 3). Both the Vanilla and Priority-based NoCs are grid-based wormhole NoCs, similar to [9] with static XY routing mechanism and equal capacity for all links. Flit size is 16 bit and the buffer size at routers is 4 flits. The clock frequency of the processors is assumed to be 10 GHz for simplicity and according to ITRS projections.

4.1.2 Evaluated Benchmarks

We run Mandrake 10.1 Linux with SMP kernel version 2.6, custom-compiled to interface with Simics, on top of which we run Linux programs as benchmarks. Our benchmarks include three SPLASH-2 benchmarks[29]: (Fft, Ocean and Radix), and two static web content serving (Apache HTTP server and Zeus web server). For both web benchmarks, we use the SURGE[30] toolkit to generate a workload of web requests from a 30,000-file, 700MB repository with a zero back-off time. This toolkit generates a Zipf distribution of file accesses, which was shown to match real-world web workloads (used by SpecWeb2005).Typical parallel benchmarks begin with some setup code, which is run serially, and only then embark on parallel processing. Since we are interested in the parallel part of the applications, we fast forward through the serial part and performed the measurements only in the parallel part of the code.

4.2 Simulation Results

We simulated about a million instructions per each benchmark. The measured delay of transaction is calculated from the time that the processor writes the transaction into its NoC interface queue until the processor receives the requested cache-line from L2. In other words, the overall delay consists of the queueing time at the processor interface, the end-to-end (ETE) NoC delay towards the L2-bank, the queuing delay at the L2 interface, the delay of possible invalidation or write back procedures, and finally the ETE NoC delay from L2 to the requesting processor.

We measure the total L2-access delay of both read and read exclusive commands. These commands are extremely performance critical since they stall the processor. We also measure the total program throughput which is the number of executed commands. Although we fully simulate writes resulting from eviction from L1-cache, we do not include them in the average delay calculation since they do not affect the performance of the processors (these writes do not cause processor stalls).

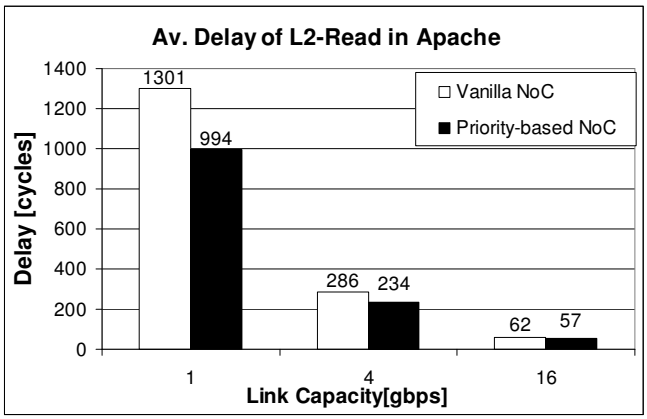

Figure 8. Average delay of L2 Read Transaction using Vanilla and Priority-based NoCs in Apache for different link capacities

Figure 8 demonstrates the distribution of L2 read transactions average delays in the Apache server benchmark for Vanilla and Priority-based NoCs as a function of link capacity in the network. The graph shows that the delay of L2 transactions is affected by the NoC link capacity or alternatively by the network load. When the NoC is lightly loaded then the read delays are relatively small. During high network loads the L2 access delays can become extremely large. A similar behavior is observed in read exclusive delays.

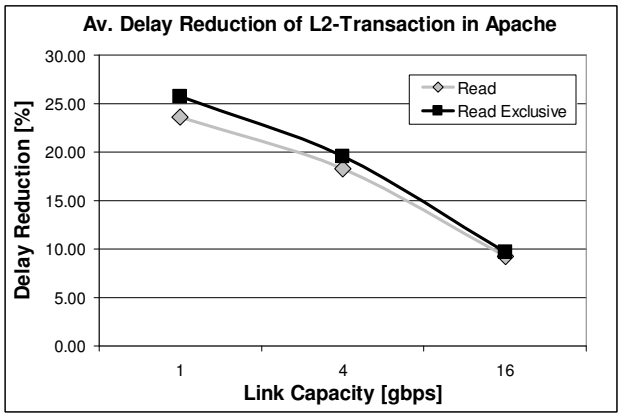

Figure 9. Read and Read Exclusive average delay reduction in Apache as a function of different link capacity

One can also observe from the graph that the delay improvement achieved using a Priority-based NoC is growing with the network load. Figure 9 quantifies the

delay reduction that can be achieved by using a Priority-based NoC in both read and read exclusive transactions in the Apache benchmark. As expected, the major delay reduction is achieved in a relatively highly loaded network with a large contention over the network resources. In such cases, the priority based mechanism allows for short control messages to bypass long worms that are blocked in the network. It achieves a delay reduction of 26% and 24% for regular and exclusive read respectively in a heavily loaded NoC, and a 10% delay reduction for both read and read exclusive transactions in a lightly loaded NoC.

We also explore the potential delay reduction of using a Priority-based NoC compared to the Vanilla-approach across several benchmarks over a medium loaded NoC. The results summarized in Figure 10 clearly show the advantage of Priority-based NoC over Vanilla approaches.

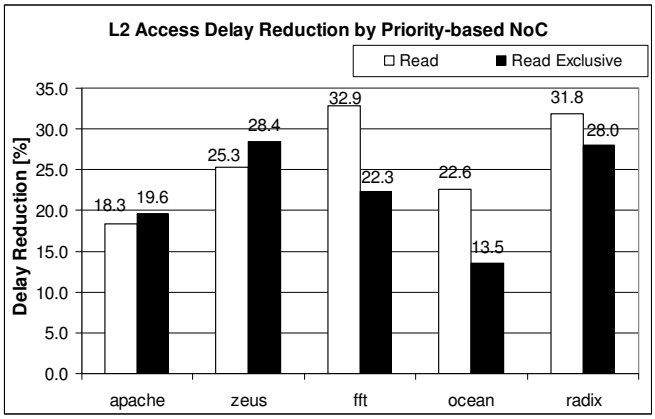

Figure 10. Average L2 access delay reduction in read and read exclusive transactions by using Priority-based NoC over several benchmarks

The actual L2-transaction delay as well as the delay reduction of the Priority based approach is a function of many factors that depend on the system and the benchmarks. These include the frequency of L2-access (or inter-transaction delay), the load of the specific routing path in the network, the amount of read-write sharing in the benchmark and as a result the amount of invalidations and write backs, the number of sharers that need to be invalidated during each invalidation, the distance from the desired L2 bank and finally the contention on that bank. Despite this complication, using simple and low-cost Priority-based NoC approach we succeed to achieve a substantial delay reduction across a variety of benchmarks. The maximal delay reduction reached 33% in FFT read and 28% in Zeus read exclusive transactions.

Finally, we examine the total application speedup using a Priority-based NoC. The results are depicted in Figure 11. We observe that the priority-based NoC improves the overall system performance in all benchmarks. The best improvement is achieved in Zeus, which boosts overall system performance by 9.4%. The system performance improvement strongly depends on the amount of L2-delay reduction and the frequency of L2-access in the benchmark.

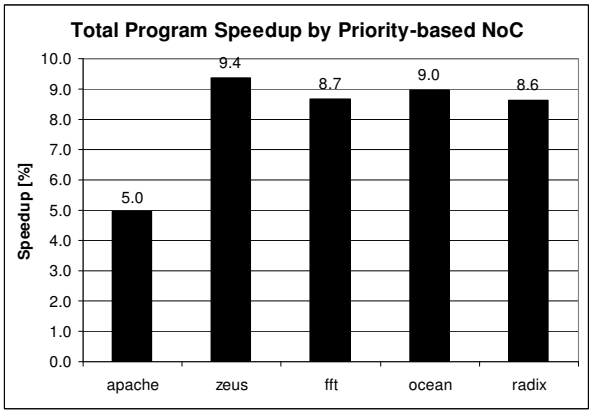

Figure 11. Overall Program Speedup by using priorities over several benchmarks

5. SUMMARY

We explored the NoC communication paradigm associated with cache coherency of static and dynamic NUCA CMP. We described its basic mapping over a Vanilla-NoC system, including the details of processor and L2 network interfaces. The main contribution of the paper is the extension of the vanilla NoC and processor interface with a simple and low cost priority mechanism. Such Priority-based NoC is capable of differentiating and prioritizing short control messages from long data packets. This generic approach fits and expedites almost any CMP communication task (i.e uncached and cache-coherent R/W, search in DNUCA, isolating low priority traffic, synchronization and mutual exclusion support) for any proposed coherency-protocol modifications. In particular, we showed how to boost performance of directory-based coherency protocol using priority-based NoC, while maintaining coherency correctness. We show how to further enrich the unicast-based communication services of such a Vanilla NoC by Advanced-Services NoC mechanisms such as: virtual invalidation rings, efficient store-and-forward multicast for short messages which is embedded within a wormhole NoC, and a cache-line search mechanism for the efficient operation of dynamic NUCA. In addition to cache coherency operations, these mechanisms can also improve other basic CMP transactions such as search and synchronization support. Detailed CMP-NoC simulations show an impressive (up to 33%) reduction in L2-access delay by using priority-based NoC instead of the simplistic Vanilla-NoC across a variety of benchmarks. The simulations also demonstrate a substantial total system speedup in all simulated benchmarks (up to 9.4% speedup).

6. REFERENCES

[1] B. M. Beckmann and D. A. Wood. Managing wire delay in large chip multiprocessor caches. In MICRO 37, Dec. 2004

[2] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger and S.W. Keckler. A NUCA substrate for Flexible CMP Cache Sharing. ICS 05 , June, 2005

[3] Bradford M. Beckmann, Michael R. Marty, and David A. Wood, ASR: Adaptive Selective Replication for CMP Caches, MICRO 2006

[4] M. Zhang and K. Asanovic, Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors, ISCA-32, June 2005.

[5] C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In ASPLOS X, pages 211–222, Oct. 2002

[6] R. Ricci, S. Barrus, D. Gebhardt, and R. Balasubramonian, Leveraging Bloom Filters for Smart Search Within NUCA Caches, 7th Workshop on Complexity-Effective Design (WCED), June 2006

[7] Z. Guz, I. Keidar, A. Kolodny, U. C. Weiser, Nahalal: Memory Organization for Chip Multiprocessors, Technical Report CCIT 600, Technion Department of Electrical Engineering, September 2006

[8] David E. Culler et al., "Parallel Computer Architecture: A Hardware/Software Approach", Morgan Kaufmann Publishers Inc, 1997.

[9] E. Bolotin, et al., "QNoC: QoS Architecture and Design Process for Networks on Chip", JSA, Feb 2004

[10] S. V. Adve and K. Gharachorloo, "Shared memory consistency models: A tutorial," IEEE Computer, vol. 29, no. 12, pp. 66–76, 1996.

[11] D. Lenoski, et al., "The DASH prototype: implementation and performance," SIGARCH Comp. Arch. News, vol. 20, no. 2, pp. 92–103, 1992.

[12] X. Shen, Arvind, and L. Rudolph, "CACHET: an adaptive cache coherence protocol for distributed shared-memory systems," in Proc. 13th Int. Conf. Supercomputing, Jun. 1999, pp. 135–144.

[13] J. Huh, et al., "Speculative incoherent cache protocols," IEEE Micro, vol. 24, no. 6, Nov./Dec. 2004.

[14] D. Dai and D. Panda, "Reducing cache invalidation overheads in wormhole routed DSMs using multidestination message passing," in Proc. 1996 Int. Conf. Par. Processing, Aug. 1996, pp. 138–145.

[15] E. E. Bilir, et al., "Multicast snooping: a new coherence method using a multicast address network," in Proc. 26th Int. Symp. Comp. Arch., Jun. 1999, pp. 294–304.

[16] L. Barroso et al., "Piranha: A scalable architecture based on singlechip multiprocessing," in Proc. 27th Int. Symp. Comp. Arch., Jun. 2000, pp. 282–293.

[17] L. Barroso and M. Dubois, "Performance evaluation of the slotted ring multiprocessor," in IEEE Trans. Comp., July 1995, pp. 878– 890.

[18] L. Cheng, et al., "Interconnect-aware coherence protocols," in Proc. 33rd Int. Symp. Comp. Arch., Jun. 2006.

[19] H. E. Mizrahi, et al., "Introducing memory into the switch elements of multiprocessor interconnection networks," in Proc. 16th Int. Symp. Comp. Arch., Jun. 1989, pp. 158–166.

[20] M. M. K. Martin, M. D. Hill, and D. A. Wood, "Token coherence: Decoupling performance and correctness," in Proc. 30th Int. Symp. Comp. Arch., Jun. 2003, pp. 182–193

[21] Noel Eisley, Li-Shiuan Peh and Li Shang, "In-Network Cache Coherence", In Proceedings of the 39th International Symposium on Microarchitecture (MICRO), Orlando, Florida, December 2006.

[22] Frédéric Pétrot, "On Cache Coherency and Memory Consistency Issues in NoC Based Shared Memory Multiprocessor SoC Architectures"

[23] Mirko Loghi et al. "Exploring the energy efficiency of cache coherence protocols in single-chip multi-processor". GLSVLSI 2005.

[24] Matteo Monchiero et al, "Efficient Synchronization for Embedded On-Chip Multiprocessors", IEEE TVLSI October 2006

[25] Radovic, Z. Hagersten, E. , "Efficient Synchronization for Nonuniform Communication Architectures", Supercomputing, ACM/IEEE 2002 Conference

[26] D. Lenoski, J. Laudon, K. Gharachorloo,W-D.Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. S. Lam. The Stanford Dash Multiprocessor. IEEE Computer, 25(3):63–79, March 1992.

[27] E. Bolotin, I. Cidon, R. Ginosar and A. Kolodny, "Cost Considerations in Network on Chip", Integration-The VLSI Journal, special issue on Network on Chip, Volume 38, Issue 1, October 2004, pp. 19-42.

[28] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, and G. Hallberg. Simics: A full system simulation platform. IEEE Computer, 35(2):50–58, Feb. 2002.

[29] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In Proceedings of the 22nd Annual International Symposium on Computer Architecture, pages 24–37, June 1995.

[30] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In Measurement and Modeling of Computer Systems, pages 151–160, June 1998.

[31] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal Network on Chip: Concepts, Architectures, and Implementations", IEEE Design and Test of Computers, 2005.

[32] F.Moraes et al,"HERMES: an Infrastructure for Low Area Overhead Packet-switching NoC," VLSI Journal, 2004.

[33] M. Dall'Osso et al., "XPIPES: a Latency Insensitive Parameterized Network-on-Chip Architecture" ICCD, 2003.

[34] M. Millberg et al., "The Nostrum Backbone-A Communication Protocol Stack for Networks on Chip," VLSI Design Conf., Jan 2004.

[35] D.S. Tortosa and J. Nurmi, "Proteo: A New Approach to Network-on-Chip," IASTED CSN'02, Spain, 2002.

[36] S. Kumar et al., "A Network on Chip Architecture and Design Methodology," ISVLSI 2002.

[37] J. Hu, R. Marculescu, "DyAD - Smart Routing for Networks-on-Chip," DAC 2004.

[38] J.Henkel, W.Wolf, and S.Chakradhar, "On Chip Networks: A scalable communication-centric embedded system design paradigm", in Procedings, VLSI Design 2004

[39] D. Bertozzi et al., " NoC synthesis flow for customized domain specific multiprocessor systems-on-chip". IEEE Trans. on Parallel and Dist.Systems, 16(2):113–129, 2005.

[40] A. Hemani et al., Network on a chip: An architecture for billion transistor era. In IEEE NorChip, 2000.

[41] L Benini, G.D. Micheli, Networks on chips: a new SoC paradigm, IEEE Computer 35 (1) (2002) 70–78.

[42] P. Guerrier, A. Greiner, A generic architecture for on-chip packet-switched interconnections, DATE 2000

[43] E. Rijpkema, K. Goosens, P. Wielage, A router architecture for networks on silicon, in: Proceedings of Progress 2001