NetCache: A Network/Cache Hybrid for Multiprocessors *

Enrique V. Carrera and Ricardo Bianchini

COPPE Systems Engineering
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil 21945-970
{vinicio,ricardo}@cos.ufrj.br

Abstract. In this paper we propose the use of an optical network not only as the communication medium, but also as a system-wide cache for the shared data in a multiprocessor. More specifically, the basic idea of our novel network/cache hybrid (and associated coherence protocol), called NetCache, is to use an optical ring network on which some amount of recently-accessed shared data is continually sent around. These data are organized as a cache shared by all processors. We use detailed execution-driven simulations of a dozen applications to evaluate a multiprocessor based on our NetCache architecture. We compare a 16-node multiprocessor with a third-level NetCache against two highly-efficient systems based on the DMON and LambdaNet optical interconnects. Our results demonstrate that the NetCache multiprocessor outperforms the DMON and LambdaNet systems by as much as 99 and 79%, respectively. Based on these results, our main conclusion is that the NetCache is highly efficient for most applications.

1 Introduction

The gap between processor and memory speeds continues to widen; memory accesses usually take hundreds of processor cycles to complete, especially in scalable shared-memory multiprocessors. Caches are generally considered the best technique for tolerating the high latency of memory accesses. Microprocessor chips include caches so as to avoid processor stalls due to the unavailability of data or instructions. Modern computer systems use off-chip caches to reduce the average cost of memory accesses. Unfortunately, caches are often somewhat smaller than their associated processors' working sets, meaning that a potentially large percentage of memory references might still end up reaching the memory modules. This scenario is particularly damaging to performance when memory references are directed to remote memories through the scalable interconnection network. In this case, the performance of the network is critical.

The vast majority of multiprocessors use electronic interconnection networks. However, relatively recent advances in optical technology have prompted several studies on the use of optical networks in multiprocessors. Optical fibers exhibit extremely high bandwidth and can be multiplexed to provide a large number of independent communication channels. These characteristics can be exploited to improve performance

* This research was supported by Brazilian CNPq.

(significantly) by simply replacing a traditional multiprocessor network with an optical equivalent. However, we believe that optical networks can be exploited much more effectively. The extremely high bandwidth of optical media provides data storage in the network itself and, thus, these networks can be transformed into fast-access data storage devices.

Based on this observation and on the need to reduce the average latency of memory accesses in parallel computers, in this paper we propose the use of an optical network not only as the communication medium, but also as a system-wide cache for the shared data in a multiprocessor. More specifically, the basic idea of our novel network/cache hybrid (and associated coherence protocol), called NetCache, is to use an optical ring network on which some amount of recently-accessed shared data is continually sent around. These data are organized as a cache shared by all processors. Most aspects of traditional caches, such as hit/miss rates, capacity, storage units, and associativity, also apply to the NetCache.

The functionality provided by the NetCache suggests that a multiprocessor based on it should deliver better performance than previously-proposed optical network-based multiprocessors. To verify this hypothesis, we use detailed execution-driven simulations of a dozen applications running on a 16-node multiprocessor with a third-level shared cache, and compare it against highly-efficient systems based on the DMON [7] and LambdaNet [6] optical interconnects. Our results demonstrate that the NetCache multiprocessor performs at least as well as the DMON-based system for our applications; the performance differences in favor of the NetCache system can be as significant as 99%. The NetCache system also compares favorably against the LambdaNet multiprocessor. For nine of our applications, the running time advantage of the NetCache machine ranges from 7% for applications with little data reuse in the shared cache to 79% for applications with significant data reuse. For the other applications, the two systems perform similarly.

Given that optical technology will likely provide a better cost/performance ratio than electronics in the future, we conclude that designers should definitely consider the NetCache architecture as a practical approach to both low-latency communication and memory latency tolerance.

2 Background

2.1 Optical Communication Systems

Two main topics in optical communication systems bear a direct relationship to the ideas proposed in this paper: Wavelength Division Multiplexing (WDM) networks and delay line memories.

WDM networks. The maximum bandwidth achievable over an optic fiber is on the order of Tbits/s. However, due to the fact that the hardware associated with the end points of an optical communication system is usually of an electronic nature, transmission rates are currently limited to the Gbits/s level. In order to approach the full potential of optical communication systems, multiplexing techniques must be utilized. WDM is one such multiplexing technique. With WDM, several independent communication channels can be implemented on the same fiber. Optical networks that use this

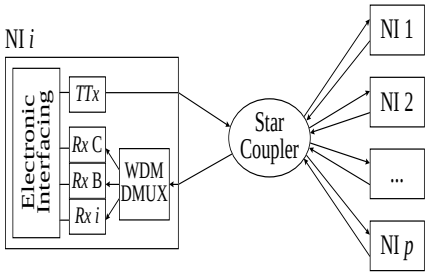

Fig. 1. Overview of DMON.

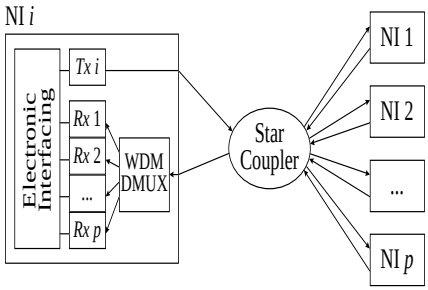

Fig. 2. Overview of LambdaNet.

multiplexing technique are called WDM networks. Due to the rapid development of the technology used in its implementation, WDM has become one of the most popular multiplexing techniques. The NetCache architecture focuses on WDM technology due to its immediate availability, but there is nothing in our proposal that is strictly dependent on this specific type of multiplexing.

Delay line memories. Given that light travels at a constant and finite propagation speed on the optical fiber (approximately 2.1E+8 meters/s), a fixed amount of time elapses between when a datum enters and leaves an optical fiber. In effect, the fiber acts as a delay line. Connecting the ends of the fiber to each other (and regenerating the signal periodically), the fiber becomes a delay line memory [11], since the data sent to the fiber will remain there until overwritten. An optical delay line memory exhibits characteristics that are hard to achieve by other types of delay lines. For instance, due to the high bandwidth of optical fibers, it is possible to store a reasonable amount of memory in just a few meters of fiber (e.g. at 10 Gbits/s, about 5 Kbits can be stored on one 100 meters-long WDM channel).

2.2 DMON

The Decoupled Multichannel Optical Network (DMON) is an interesting WDM network that has been proposed by Ha and Pinkston in [7]. The network divides its $p + 2$ (where p is the number of nodes in the system) channels into two groups: one group is used for broadcasting, while the other is used for point-to-point communication between nodes. The first group is formed by two channels shared by all nodes in the system, the control channel and the broadcast channel. The other p channels, called home channels, belong in the second group of channels.

The control channel is used for distributed arbitration of all other channels through a reservation scheme [3]. The control channel itself is multiplexed using the TDMA (Time Division Multiple Access) protocol [3]. The broadcast channel is used for broadcasting global events, such as coherence and synchronization operations, while home channels are used only for memory block request and reply operations. Each node can transmit on any home channel, but can only receive from a single home channel. Each node acts as "home" (the node responsible for providing up-to-date copies of the blocks)

for $1/p$ of the cache blocks. A node receives requests for its home blocks from its home channel. Block replies are sent on the requester's home channel.

Figure 1 overviews a network interface ("NI") in the DMON architecture, with its receivers (labeled "*Rx*") and tunable transmitter ("*TTx*"). As seen in the figure, in this architecture each node requires a tunable transmitter (for the control, broadcast, and home channels), and three fixed receivers (two for the control and broadcast channels, and one for the node's home channel).

We compare performance against an update-based protocol [1] we previously proposed for DMON. This protocol has been shown to outperform the invalidate-based scheme proposed for DMON in [7]. Our protocol is very simple since all writes to shared data are sent to their home nodes, through coalescing write buffers. Our update protocol also includes support for handling critical races that are caused by DMON's decoupling of the read and write network paths. The protocol treats these races by buffering the updates received during the pending read operation and applies them to the block right after the read is completed.

Given that a single broadcast channel would not be able to deal gracefully with the heavy update traffic involved in a large set of applications, we extended DMON with an extra broadcast channel for transferring updates. A node can transmit on only one of the coherence channels, which is determined as a function of the node's identification, but can receive from both of these channels. In addition, we extended DMON with two fixed transmitters (one for the control channel and the other for the coherence channel on which the node is allowed to transmit) to avoid incurring the overhead of re-tuning the tunable transmitter all the time. Besides the extra channel (and associated receivers) and the additional fixed transmitters, the hardware of the modified DMON network is the same as presented in figure 1. Thus, the overall hardware cost of this modified DMON architecture in terms of optical components is then $7 \times p$.

2.3 LambdaNet

The LambdaNet architecture has been proposed by Goodman *et al.* in [6]. The network allocates a WDM channel for each node; the node transmits on this channel and all other nodes have fixed receivers on it. In this organization each node uses one fixed transmitter and p fixed receivers, as shown in figure 2. The overall hardware cost of the LambdaNet is then $p^2 + p$.

No arbitration is necessary for accessing transmission channels. Each node simultaneously receives all the traffic of the entire network, with a subsequent selection, by electronic circuits, of the traffic destined for the node. This scheme thus allows channels to be used for either point-to-point or broadcast communication.

Differently from DMON, the LambdaNet was not proposed with an associated coherence protocol. The LambdaNet-based multiprocessor we study in this paper uses a write-update cache coherence protocol, where write and synchronization transactions are broadcast to nodes, while the read traffic uses point-to-point communication between requesters and home nodes. Just as the update-based protocol we proposed for DMON, the memory modules are kept up-to-date at all time. Again, in order to reduce the write traffic to home nodes, we assume coalescing write buffers.

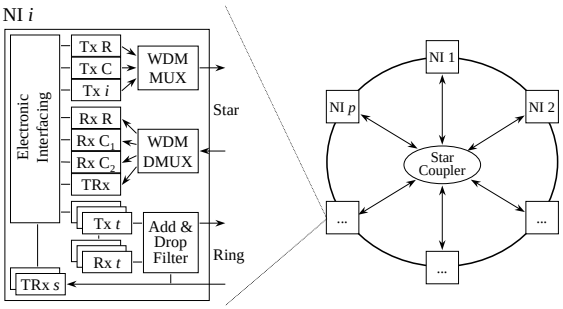

Fig. 3. Overview of NetCache Architecture.

The combination of the LambdaNet and the coherence protocol we suggest for it represents a performance upper bound for multiprocessors that do not cache data on the network, since the update-based protocol avoids coherence-related misses, the LambdaNet channels do not require any medium access protocol, and the LambdaNet hardware does not require the tunning of transmitters or receivers.

3 The Network/Cache Hybrid

3.1 Overview

Each node in a NetCache-based multiprocessor is extremely simple. In fact, all of the node's hardware components are pretty conventional, except for the network (NetCache) interface. More specifically, the node includes one processor, a coalescing write buffer, first and second-level caches, local memory, and the NetCache interface connected to the memory bus. The interface connects the node to two subnetworks: a star coupler WDM subnetwork and a WDM ring subnetwork. Figure 3 overviews the NetCache architecture, including its interfaces and subnetworks. The star coupler subnetwork is the OPTNET network [1], while the ring subnetwork is unlike any other previously-proposed optical network. The following sections describe each of these components in detail.

3.2 Star Coupler (Sub)Network

Just as in DMON, the star coupler WDM subnetwork in the NetCache divides the channels into two groups: one group for broadcast-type traffic and another one for direct point-to-point communication. Three channels, a request channel and two coherence channels, are assigned to the first group, while the other p channels, called home channels, are assigned to the second group.

The request channel is used for requesting memory blocks. The response to such a request is sent by the block's home node (the node responsible for providing up-to-date copies of the block) on its corresponding home channel. The coherence channels are

used for broadcasting coherence and synchronization transactions. Just like the control channel in DMON, the request channel uses TDMA for medium access control. The access to the coherence channels, on the other hand, is controlled with TDMA with variable time slots [3]. Differently from DMON, home channels do not require arbitration, since only the home node can transmit on the node's home channel.

As seen in figure 3, each node in the star coupler subnetwork requires three fixed transmitters (one for the request channel, one for the home channel, and the last for one of the coherence channels), three fixed receivers (for the broadcast channels), and one tunnable receiver labeled "*TRx*" (for the home channels). The hardware cost for this subnetwork is then $7 \times p$ optical components.

3.3 Ring (Sub)Network

The ring subnetwork is the most interesting aspect of the NetCache architecture. The ring is used to store some amount of recently-accessed shared data, which are continually sent around the ring's WDM channels, referred to as cache channels, in one direction. These data are organized as cache blocks shared by all nodes of the machine. In essence, the ring becomes an extra level of caching that can store data from any node. However, the ring does not respect the inclusion property of cache hierarchies, i.e. the data stored by the ring is not necessarily a superset of the data stored in the upper cache levels. More specifically, the ring's storage capacity in bits is given by: $capacity\_in\_bits = (num\_channels \times fiber\_length \times transmission\_rate) \div speed\_of\_light\_on\_fiber$, where $speed\_of\_light\_on\_fiber = 2 \times 10^8 m/s$.

The ring works as follows. Each cache channel stores the cached blocks of a particular home node. Assuming a total of c cache channels, each home node has $t = c/p$ cache channels for its blocks. Since cache channels and blocks are associated with home nodes in interleaved, round-robin fashion, the cache channel assigned to a certain block can be determined by the following expression: $block\_address \ mod \ c$. A particular block can be placed anywhere within a cache channel.

Just as in a regular cache, our shared cache has an address tag on each block frame (shared cache line). The tags contain part of the shared cache blocks' addresses and consume a (usually extremely small) fraction of the storage capacity of the ring. The access to each block frame by different processors is strictly sequential as determined by the flow of data on the cache channel. The network interface accesses a block via a shift register that gets replicas of the bits flowing around the ring. Each shift register is as wide as a whole block frame. When a shift register is completely filled with a block, the hardware moves the block to another register, the access register, which can then be manipulated. In our base simulations, it takes 10 cycles to fill a shift register. This is the amount of time the network interface has to check the block's tag and possibly to move the block's data before the access register is overwritten.

The NetCache interface at each node can read any of the cache channels, but can only write to the t cache channels associated to the node. In addition, the NetCache interface regenerates, reshapes, and reclocks these t cache channels. To accomplish this functionality, the interface requires two tunable receivers, and t fixed transmitter and receiver sets, as shown in figure 3. One of the tunable receivers is for the cache channel used last and the other is for pre-tuning to the next channel. The t fixed transmitters

are used to insert new data on any of the cache channels associated with the node. In conjunction with these transmitters, the t fixed receivers are used to re-circulate the data on the cache channels associated with the node. Thus, the hardware cost for this subnetwork is then $2 \times p + 2 \times c$ optical components.

Taking both subnetworks into account, the overall hardware cost of the NetCache architecture is $9 \times p + 2 \times c$. In our experiments we set $c = 8 \times p$, leading to a total of $25 \times p$ optical components. This cost is a factor of 3.6 greater than that of DMON, but is linear on p, as opposed to quadratic on p as the LambdaNet's.

3.4 Coherence Protocol

The coherence protocol is an integral part of the NetCache architecture. The NetCache protocol is based on update coherence, supported by both broadcasting and point-to-point communication. The update traffic flows through the coherence channels, while the data blocks are sent across the home and cache channels. The request channel carries all the memory read requests and update acknowledgements. The description that follows details the protocol in terms of the actions taken on read and write accesses.

Reads. On a read access, the memory hierarchy is traversed from top to bottom, so that the required word can be found as quickly as possible. A miss in the second-level cache blocks the processor and is handled differently depending on the type of data read. In case the requested block is private or maps to the local memory, the read is treated by the memory, which returns the block to the processor.

If the block is shared and maps to another home node, the miss causes an access to the NetCache. The request is sent to the corresponding node through the request channel, and the tunable receiver in the star coupler subnetwork is tuned to the correct home channel. In parallel with this, the requesting node tunes one of its tunable receivers in the ring subnetwork to the corresponding cache channel. The requesting node then waits for the block to be received from either the home channel or the cache channel, reads the block, and returns it to the second-level cache.

When a request arrives at the home node, the home checks if the block is already in any of its cache channels. In order to manage this information, the NetCache interface keeps a hash table storing the address of its blocks that are currently cached on the ring. If the block is already cached, the home node will simply disregard the request; the block will eventually be received by the requesting node. If the block is not currently cached, the home node will read it from memory and place it on the correct cache channel, replacing one of the blocks already there if necessary. (Replacements are random and do not require write backs, since the memory and the cache are always kept up-to-date.) In addition to returning the requested block through a cache channel, the NetCache sends the block through the home node's home channel.

It is important to note that our protocol starts read transactions on both the star coupler and ring subnetworks so that a read miss in the shared cache takes no longer than a direct access to remote memory. If reads were only started on the ring subnetwork, shared cache misses would take half a roundtrip longer (on average) to satisfy than a direct remote memory access.

Writes. Our multiprocessor architecture implements the release consistency memory model [4]. Consecutive writes to the same cache block are coalesced in the write

buffer. Coalesced writes to a private block are sent directly to the local memory through the first and second-level caches. Coalesced writes to a shared block are always sent to one of the coherence channels in the form of an update, again through the local caches. An update only carries the words that were actually modified in each block.

Each update must be acknowledged by the corresponding block's home node before another update by the same node can be issued, just so the memory modules do not require excessively long input queues (i.e. update acks are used simply as a flow control measure). The other nodes that cache the block simply update their local caches accordingly upon receiving the update. When the home node sees the update, the home inserts it into the memory's FIFO queue, and sends an ack through the request channel. The update acks usually do not overload the request channel, since an ack is a short message (much shorter than multi-word updates) that fits into a single request slot.

After having inserted the update in its queue, the home node checks whether the updated block is present in a cache channel. If so, besides updating its memory and local caches, the home node updates the cache channel. If the block is not present in a cache channel, the home node will not include it.

There are two types of critical races in our coherence protocol that must be taken care of. The first occurs when a coherence operation is seen for a block that has a pending read. Similarly to the update protocol we proposed for the DMON network, our coherence protocol treats this type of race by buffering updates and later combining them with the block received from memory. The second type of race occurs because there is a window of time between when an update is broadcast and when the copy of the block in the shared cache is actually updated. During this timing window, a node might miss on the block, read it from the shared cache and, thus, never see the update. To avoid this problem, each network interface includes a small FIFO queue that buffers the block addresses of some previous updates. Any shared cache access to a block represented in the queue must be delayed until the address of the block leaves the queue, which guarantees that when the read is issued, the copy of the block in the shared cache will be up-to-date. The management of the queue is pretty simple. The entry in front of the queue can be dropped when it has resided in the queue for the same time as two roundtrips around the ring subnetwork, i.e. the maximum amount of time it could take a home node to update the copy of the block in the shared cache. Hence, the maximum size of the queue is the maximum number of updates that can be issued during this period; in our simulations, this number is 54.

4 Methodology and Workload

We simulate multiprocessors based on the NetCache, DMON and LambdaNet interconnects. We use a detailed execution-driven simulator (based on the MINT front-end [12]) of 16-node multiprocessors. Each node of the simulated machines contains a single 200-MHz processor, a 16-entry write buffer, a 4-KByte direct-mapped first-level data cache with 32-byte cache blocks, a 16-KByte direct-mapped second-level data cache with 64-byte cache blocks, local memory, and a network interface. Note that these cache sizes are purposedly kept small, because simulation time limitations prevent us from using real life input sizes.

Operation	Latency
Shared cache read hit	
1. 1st-level tag check	1
2. 2nd-level tag check	4
3. Avg. shared cache delay	25
4. NI to 2nd-level cache	16
Total shared cache hit	46
Shared cache read miss	
1. 1st-level tag check	1
2. 2nd-level tag check	4
3. Avg. TDMA delay	8
4. Memory request	1^*
5. Flight	1
6. Memory read	76^+
7. Block transfer	11
8. Flight	1
9. NI to 2nd-level cache	16
Total shared cache miss	119

Table 1. Read Latency Breakdowns (in pcycles) for NetCache – 1 pcycle = 5 nsecs.

Operation	Latency	
	LambdaNet	DMON
2nd-level read miss		
1. 1st-level tag check	1	1
2. 2nd-level tag check	4	4
3. Avg. TDMA delay	–	8
4. Reservation	–	1^*
5. Tuning delay	–	4
6. Memory request	1^*	2
7. Flight	1	1
8. Memory read	76^+	76^+
9. Avg. TDMA delay	–	8
10. Reservation	–	1^*
11. Block transfer	11^*	12
12. Flight	1	1
13. NI to 2nd-level	16	16
Total 2nd-level miss	111	135

Table 2. Read Latency Breakdowns (in pcycles) for DMON and LambdaNet – 1 pcycle = 5 nsecs.

Shared data are interleaved across the memories at the block level. All instructions and first-level cache read hits are assumed to take 1 processor cycle (pcycle). First-level read misses stall the processor until the read request is satisfied. A second-level read hit takes 12 pcycles to complete. Writes go into the write buffer and take 1 pcycle, unless the write buffer is full, in which case the processor stalls until an entry becomes free. Reads are allowed to bypass writes that are queued in the write buffers. A memory module can provide the first two words requested 12 pcycles after the request is issued. Other words are delivered at a rate of 2 words per 8 pcycles. Memory and network contention are fully modeled.

In the coherence protocols we simulate only the secondary cache is updated when an update arrives at a node; the copy of the block in the first-level cache is invalidated. In addition, in order to reduce the write traffic, our multiprocessors use coalescing write buffers for all protocol implementations. A coalesced update only carries the words that were actually modified in each block. Our protocol implementations assume a release-consistent memory model [4].

The optical transmission rate we simulate is 10 Gbits/s. In the NetCache simulations we assume 128 cache channels. The length of the fiber is approximately 45 meters. These parameters lead to a ring roundtrip latency of 40 cycles and a storage capacity of 32 KBytes of data. The shared cache line size is 64 bytes. The NetCache storage capacity we simulate is compatible with current technology, but is highly conservative with respect to the future potential of optics, especially if we consider multiplexing techniques such as OTDM (Optical Time Division Multiplexing) which will potentially support 5000 channels [10].

Program	Description	Input Size
CG	Conjugate Gradient kernel	1400 × 1400 doubles, 78148 non-0s
Em3d	Electromagnetic wave propagation	8 K nodes, 5% remote, 10 iterations
FFT	1D Fast Fourier Transform	16 K points
Gauss	Unblocked Gaussian Elimination	256 × 256 floats
LU	Blocked LU factorization	512 × 512 floats
Mg	3D Poisson solver using multigrid techniques	24 × 24 × 64 floats, 6 iterations
Ocean	Large-scale ocean movement simulation	66 × 66 grid
Radix	Integer Radix sort	512 K keys, radix 1024
Raytrace	Parallel ray tracer	teapot
SOR	Successive Over-Relaxation	256 × 256 floats, 100 iterations
Water	Simulation of water molecules, spatial alloc.	512 molecules, 4 timesteps
WF	Warshall-Floyd shortest paths algorithm	384 vertices, (i,j) w/ 50% chance

Table 3. Application Description and Main Input Parameters.

The latency of shared cache reads in the NetCache architecture is broken down into its base components in table 1. The latency of 2nd-level read misses in the LambdaNet and DMON-based systems is broken down in table 2. In the interest of saving space and given that most or all coherence transaction overhead is usually hidden from the processor by the write buffer, we do not present the latency breakdown for these transactions. Please refer to [1] for further details. All numbers in tables 1 and 2 are in pcycles and assume channel and memory contention-free scenarios. The values marked with "*" and "+" are the ones that may be increased by network and memory contention/serialization, respectively.

Our application workload consists of twelve parallel programs: CG, Em3d, FFT, Gauss, LU, Mg, Ocean, Radix, Raytrace, SOR, Water, and WF. Table 3 lists the applications and their inputs.

5 Experimental Results

5.1 Overall Performance

Except for CG, LU, and WF, all our applications exhibit good speedups (> 10) on a 16-node NetCache-based multiprocessor with a 32-KByte shared cache. Figure 4 shows the running times of our applications. For each application we show, from left to right, the NetCache, LambdaNet, and DMON-U results, normalized to those of NetCache.

A comparison between the running time of the LambdaNet and DMON-U systems shows that the former multiprocessor performs at least as well as the latter for all applications, but performance differences are always relatively small; 6% on average. This relatively small difference might seem surprising at first, given that the read latency is by far the largest overhead in all our applications (except WF) and the contention-free 2nd-level read miss latency in the DMON-U system is 22% higher than in the LambdaNet system. However, the LambdaNet multiprocessor is usually more prone to

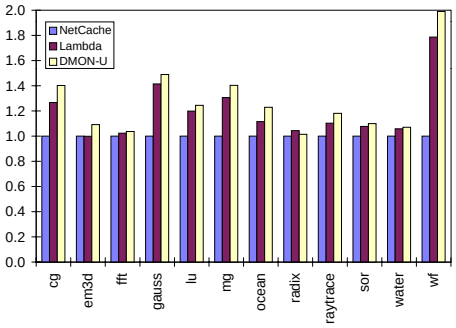

Fig. 4. Running Times.

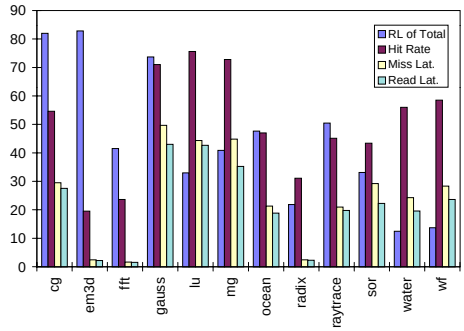

Fig. 5. Read Latencies and Hit Rates.

contention effects than the DMON-U and NetCache systems, due to two characteristics of the former system: a) its read and write transactions are not decoupled; and b) its absence of serialization points for updates from different nodes leads to an enormous bandwidth demand for updates. Contention effects are then the reason why performance differences in favor of the LambdaNet system are not more significant.

The NetCache system compares favorably against the DMON-U multiprocessor. The performance advantage of the NetCache system averages 27% for our applications. Except for FFT and Radix, where the performance difference between these two systems is negligible, the advantage of the NetCache ranges from 7% for Water to 99% for WF, averaging 32%. This significant performance difference can be credited to a large extent to the ability of the shared cache to reduce the average read latency, since their contention-free 2nd-level read miss latencies only differ by 13% and contention affects the two systems in similar ways.

Figure 4 demonstrates that a comparison between the NetCache and LambdaNet multiprocessors also clearly favors our system. Our running times are 20% lower on average for our application suite. The performance of the two systems is equivalent for 3 applications: Em3d, FFT, and Radix. For the other 9 applications, the performance advantage of the NetCache multiprocessor averages around 26%, ranging from about 7% for SOR and Water to 41 and 79% for Gauss and WF, respectively. Disregarding the 5 applications for which the performance difference between the two systems is smaller than 10%, the average NetCache advantage over the LambdaNet goes up to 31%. Again, the main reason for these favorable results is the ability of the NetCache architecture to reduce the average read latency, when a non-negligible percentage of the 2nd-level read misses hit in the shared cache.

5.2 Effectiveness of Data Caching

Figure 5 concentrates our data on the effectiveness of data caching in the NetCache architecture, again assuming 16 processor nodes. For each application we show a group of 4 bars. The leftmost bar represents the read latency as a percentage of the total execution time for a NetCache multiprocessor without a shared cache. The other bars represent our system with a 32-KByte shared cache: from left to right, the shared cache hit rate,

the percentage reduction of the average 2nd-level read miss latency, and the percentage reduction in the total read latency.

The figure demonstrates that for only 3 applications in our suite (Radix, Water, and WF) the read latency is a small fraction of the overall running time of the machine without a shared cache; the read latency fraction is significant for the other 9 applications. It is exactly for the applications with large read latency fractions that a shared cache is most likely to improve performance. However, as seen in the figure, not all applications exhibit high hit rates when a 32-KByte shared cache is assumed.

In essence, the hit rate results divide our applications into 3 groups as follows. The first group (from now on referred to as *Low-reuse*) includes the applications with insignificant data reuse in the shared cache. In this group are Em3d, FFT, and Radix, for which less than 32% of the 2nd-level read misses hit in the shared cache. As a result, the reductions in 2nd-level read miss latency and total read latency are almost negligible for these applications. The second group of applications (*High-reuse*) exhibit significant data reuse in the shared cache. Three applications belong in this group: Gauss, LU, and Mg. Their hit rates are very high, around 70%, leading to reductions in 2nd-level read miss latency and total read latency of at least 35%. The other 6 applications (CG, Ocean, Raytrace, SOR, Water, and WF) form the third group (*Moderate-reuse*), with intermediate hit rates and latency reductions in the 20 to 30% range.

Overall, our caching results suggest that the NetCache multiprocessor could not significantly outperform the LambdaNet system for Em3d, FFT, Radix, and Water; these applications simply do not benefit significantly from our shared cache, either because their hit rates are low or because the latency of reads is just not a performance issue. On the other hand, the results suggest that WF should not benefit from the NetCache architecture either, which is not confirmed by our simulations. The reason for this (apparently) surprising result is that the read latency reduction promoted by the NetCache has an important side-effect: it improves the synchronization behavior of 7 applications in our suite (CG, LU, Mg, Ocean, Radix, Water, and WF). For WF in particular, a 32-KByte shared cache improves the synchronization overhead by 56% by alleviating the load imbalance exposed during this application's barrier synchronizations.

5.3 Comparison to Other Systems

To end this section we compare the NetCache architecture qualitatively to yet another set of systems.

One might claim that it should be more profitable to use a simpler optical network (such as DMON or our star coupler subnetwork) and increase the size of 2nd-level caches than to use a full-blown NetCache. Even though this arrangement does not enjoy the *shared* cache properties of the NetCache architecture, it is certainly true that larger 2nd-level caches could improve the performance of certain applications enough to outperform a NetCache system with small 2nd-level caches. However, the amount of additional cache space necessary for this to occur can be quite significant. Furthermore, even on a system with extremely large 2nd-level caches, most applications should still profit from a shared cache at a lower level, especially when the remote memory accesses of a processor can be prevented by accesses issued by other processors (as in Gauss). These effects should become ever more pronounced, as optical technology

costs decrease and problem sizes, memory latencies (in terms of pcycles) and optical transmission rates increase.

In the same vein, it is possible to conceive an arrangement where the additional electronic cache memory is put on the memory side of a simpler optical network. These memory caches would be shared by all nodes of the machine, but would add a new level to the hierarchy, a level which would effectively slow down the memory accesses that could not be satisfied by the memory caches. The NetCache also adds a level to the memory hierarchy, but this level does not slow down the accesses that miss in it. Thus, even in this scenario, the system could profit from a NetCache, especially as optical costs decrease and latencies and transmission rates increase.

6 Related Work

As far as we are aware, the NetCache architecture is the only one of its kind; a network/cache hybrid has never before been proposed and/or evaluated for multiprocessors. A few research areas are related to our proposal however, such as the use of optic fiber as delay line memory and the use of optical networks in parallel machines. Delay line memories have been implemented in optical communication systems [9] and in all-optical computers [8]. Optical delay line memories have not been used as caches and have not been applied to multiprocessors before, even though optical networks have been a part of several parallel computer designs, e.g. [5, 7].

Ha and Pinkston [7] have proposed the DMON network. Our performance analysis has shown that the NetCache multiprocessor consistently outperforms the DMON-U system. However, DMON-based systems have an advantage over the NetCache architecture as presented here: they allow multiple outstanding read requests, while the NetCache architecture does not. This limitation results from the star coupler subnetwork (OPTNET) having a single tunable receiver that must be tuned to a single home channel on a read access. We have eliminated this limitation by transforming a read request/reply sequence into a pair of request/reply sequences [2]. This extension only affects a read request that is issued while other requests are outstanding and, thus, its additional overhead should not affect most applications.

7 Conclusions

In this paper we proposed a novel architectural concept: an optical network/cache hybrid (and associated coherence protocol) called NetCache. Through a large set of detailed simulations we showed that NetCache-based multiprocessors easily outperform other optical network-based systems, especially for the applications with a large amount of data reuse in the NetCache. In addition, we evaluated the effectiveness of our shared cache in detail, showing that several applications can benefit from it. Based on these results and on the continually-decreasing cost of optical components, our main conclusion is that the NetCache architecture is highly efficient and should definitely be considered by multiprocessor designers.

Acknowledgements

The authors would like to thank Luiz Monnerat, Cristiana Seidel, Rodrigo dos Santos, and Lauro Whately for comments that helped improve the presentation of the paper. We would also like to thank Timothy Pinkston, Joon-Ho Ha, and Fredrik Dahlgren for their careful evaluation of our work and for discussions that helped improve this paper.

References

1. E. V. Carrera and R. Bianchini. OPTNET: A Cost-Effective Optical Network for Multiprocessors. In *Proceedings of the International Conference on Supercomputing '98*, July 1998.
2. E. V. Carrera and R. Bianchini. Designing and Evaluating a Cost-Effective Optical Network for Multiprocessors. November 1998. Submitted for publication.
3. P. W. Dowd and J. Chu. Photonic Architectures for Distributed Shared Memory Multiprocessors. In *Proceedings of the 1st International Workshop on Massively Parallel Processing using Optical Interconnections*, pages 151–161, April 1994.
4. K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. L. Hennessy. Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 15–26, May 1990.
5. K. Ghose, R. K. Horsell, and N. Singhvi. Hybrid Multiprocessing in OPTIMUL: A Multiprocessor for Distributed and Shared Memory Multiprocessing with WDM Optical Fiber Interconnections. In *Proceedings of the 1994 International Conference on Parallel Processing*, August 1994.
6. M. S. Goodman *et al.* The LAMBDANET Multiwavelength Network: Architecture, Applications, and Demonstrations. *IEEE Journal on Selected Areas in Communications*, 8(6):995–1004, August 1990.
7. J.-H. Ha and T. M. Pinkston. SPEED DMON: Cache Coherence on an Optical Multichannel Interconnect Architecture. *Journal of Parallel and Distributed Computing*, 41(1):78–91, 1997.
8. H. F. Jordan, V. P. Heuring, and R. J. Feuerstein. Optoelectronic Time-of-Flight Design and the Demonstration of an All-Optical, Stored Program. *Proceedings of IEEE. Special issue on Optical Computing*, 82(11), November 1994.
9. R. Langenhorst *et al.* Fiber Loop Optical Buffer. *Journal of Lightwave Technology*, 14(3):324–335, March 1996.
10. A. G. Nowatzyk and P. R. Prucnal. Are Crossbars Really Dead? The Case for Optical Multiprocessor Interconnect Systems. In *Proceedings of the 22nd International Symposium on Computer Architecture*, pages 106–115, June 1995.
11. D. B. Sarrazin, H. F. Jordan, and V. P. Heuring. Fiber Optic Delay Line Memory. *Applied Optics*, 29(5):627–637, February 1990.
12. J. E. Veenstra and R. J. Fowler. MINT: A Front End for Efficient Simulation of Shared-Memory Multiprocessors. In *Proceedings of the 2nd International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '94)*, January 1994.