NWCache: Optimizing Disk Accesses via an Optical Network/Write Cache Hybrid [⋆]

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 a simple extension to the I/O architecture of scalable multiprocessors that optimizes page swap-outs significantly. More specifically, we propose the use of an optical ring network for I/O operations that not only transfers swapped-out pages between the local memories and the disks, but also acts as a system-wide write cache. In order to evaluate our proposal, we use detailed execution-driven simulations of several out-of-core parallel applications running on an 8-node scalable multiprocessor. Our results demonstrate that the NWCache provides consistent performance improvements, coming mostly from faster page swap-outs, victim caching, and reduced contention. Based on these results, our main conclusion is that the NWCache is highly efficient for most out-of-core parallel applications.

1 Introduction

Applications frequently access far more data than can fit in main memory. Reducing disk access overhead is the most serious performance concern for these out-of-core applications. For this reason, programmers of these applications typically code them with explicit I/O calls to the operating system. However, writing applications with explicit I/O has several disadvantages [14]: programming often becomes a very difficult task [21]; I/O system calls involve data copying overheads from user to system-level buffers and vice-versa; and the resulting code is not always portable (performance-wise) between machine configurations with different memory and/or I/O resources, such as different amounts of memory or I/O latency. In contrast with the explicit I/O style of programming, we advocate that out-of-core applications should be based solely on the virtual memory mechanism and that disk access overheads should be alleviated by the underlying system. Essentially, our preference for virtual memory-based I/O is analogous to favoring shared memory instead of message passing as a more appropriate parallel programming model.

Basically, virtual memory-based I/O involves reading pages to memory and writing (or swapping) pages out to disk. Page reads can usually be dealt with efficiently by dynamically prefetching data to main memory (or to the disk controller cache) ahead of its use, e.g. [13, 10]. In cases where dynamic prefetching is not effective by itself,

[⋆] This research was supported by Brazilian CNPq.

prefetching can be improved with compiler involvement [14] or with future-access hints [9]. Page swap-outs are more difficult to optimize however, even though these writes happen away from the critical path of the computation. The performance problem with page swap-outs is that they are often very bursty and, as a result, the operating system must at all times reserve a relatively large number of free pages frames to avoid having to stall the processor waiting for swap-out operations to complete. In fact, the more effective the page prefetching technique is, the greater is the number of free page frames the operating system must reserve. This situation is especially problematic for scalable multiprocessors where not all nodes are I/O-enabled (e.g. [1, 2]), since both disk latency and bandwidth limitations delay page swap-outs.

In this paper, we propose a simple extension to the I/O architecture of scalable multiprocessors that uses an optical network to optimize page swap-outs significantly. Our proposal is based on the observation that the extremely high bandwidth of optical media provides data storage on the network itself and, thus, these networks can be transformed into fast-access (temporary) data storage devices. Specifically, we propose NWCache, an optical ring network/write cache hybrid that not only transfers swapped-out pages between the local memories and the disks of a scalable multiprocessor, but also acts as a system-wide write cache for these pages. When there is room in the disk cache, pages are copied from the NWCache to the cache such that the pages swapped out by a node are copied together. The NWCache does not require changes to the multiprocessor hardware and only requires two simple modifications to the operating system's virtual memory management code.

The NWCache has several performance benefits: it provides a staging area where swapped-out pages can reside until the disk is free; it increases the possibility of combining several writes to disk; it acts as a victim cache for pages that are swapped out and subsequently accessed by the same or a different processor; and it reduces the data traffic across the multiprocessor's interconnection network and memory buses.

In order to assess how the performance benefits of NWCache affect overall performance, we use detailed execution-driven simulations of several out-of-core parallel applications running on an 8-node scalable cache-coherent multiprocessor with 4 I/O-enabled nodes. We consider the two extremes in terms of the page prefetching techniques: optimal and naive prefetching. Under the optimal prefetching strategy, our results demonstrate that the NWCache improves swap-out times by 1 to 3 orders of magnitude. The NWCache benefits are not as significant under the naive prefetching strategy, but are still considerable. Overall, the NWCache provides execution time improvements of up to 64% under optimal prefetching and up to 42% under naive prefetching. Our results also show that a standard multiprocessor often requires a huge amount of disk controller cache capacity to approach the performance of our system.

2 Background

Wavelength Division Multiplexing 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 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 NWCache 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 [18], 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).

3 The Optical Network/Write Cache Hybrid

3.1 Traditional Multiprocessor Architecture and Virtual Memory Management

We consider a traditional scalable cache-coherent multiprocessor, where processors are connected via a wormhole-routed mesh network. As seen in figure 1, each node is the system includes one processor (labeled "μP"), a translation lookaside buffer ("TLB"), a coalescing write buffer ("WB"), first ("L1") and second-level ("L2") caches, local memory ("LM"), and a network interface ("NI"). Each I/O-enabled node also includes one disk and its controller connected to the I/O bus. The multiprocessor can be extended with a NWCache simply by plugging the NWCache interface ("NWC") into the I/O bus of its nodes. The disk controller of the I/O-enabled nodes can then be plugged to the NWCache interface. This interface connects the node to the optical ring and filters some of the accesses to the disk, as detailed below.

The only part of the multiprocessor operating system we must consider is its virtual memory management code. Again, we assume a standard strategy here. More specifically, our base system implements a single machine-wide page table, each entry of which is accessed by the different processors with mutual exclusion. Every time the access rights for a page are downgraded, a TLB-shootdown operation takes place, where all other processors are interrupted and must delete their TLB entries for the page.

The operating system maintains a minimum set of free page frames per node of the multiprocessor. On a page fault, the operating system sends a request for the page[2] to the corresponding disk across the interconnection network. (We assume that pages are stored in groups of 32 consecutive pages. The parallel file system assigns each of these

[2] For simplicity of presentation, from now on we will not differentiate a virtual memory page from a disk block.

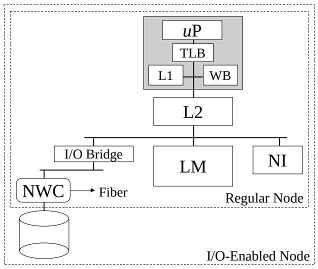

Fig. 1. Overview of Node.

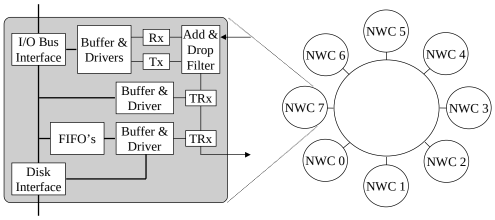

Fig. 2. Overview of NWCache System.

groups to a different disk in round-robin fashion.) For each request, the disk controller reads the page from its cache (cache hit) or disk (cache miss).

When the page faulted on arrives from its disk, the global page table is updated, allowing all other processors to access (and cache) the page data remotely. If the arrival of this page reduces the number of free page frames on the node below the minimum, the operating system uses LRU to pick a page to be replaced. If the page is dirty, a page swap-out operation is started. Otherwise, the page frame is simply turned free.

Page prefetching is not within the scope of this work. Thus, we consider the extreme prefetching scenarios: optimal prefetching and naive prefetching. Optimal prefetching attempts to approximate the performance achieved by highly-sophisticated compilers [14] or application hints [9] that can prefetch data from disks to disk caches or to memory. Our idealized technique then assumes that all page requests can be satisfied directly from the disk cache; i.e. all disk read accesses are performed in the background of page read requests.

Under the naive-prefetching scenario, the only prefetching that takes places occurs on a disk cache miss, when the controller fills its cache with pages sequentially following the missing page. We believe this technique is very naive for two reasons: a) files in our system are stripped across several disks; and b) several of our applications do not fetch pages sequentially.

A page that is swapped out of memory is sent to the corresponding disk cache across the network. The disk controller responds to this message with an ACK, if it was able to place the page in its cache. Writes are given preference over prefetches in the cache. The ACK allows the swapping node to re-use the space occupied by the page in memory. The disk controller responds with a NACK, if there was no space left in its cache (i.e. the disk controller's cache is full of swap-outs). The controller records the NACK into a FIFO queue. When room becomes available in the controller's cache, the controller sends a OK message to the requesting node, which prompts it to re-send the page.

3.2 The Optical Ring

Overview. Figure 2 overviews the optical ring that implements the NWCache. The ring is only used to transfer pages that were recently swapped out of memory by the different multiprocessor nodes to the disks, while "storing" these pages on the network itself. All other traffic flows through the regular interconnection network.

The swapped-out pages are continually sent around the ring's WDM channels, referred to as cache channels, in one direction. In essence, the ring acts as a write cache, where a page that is swapped out of main memory resides until enough room for it exists in its disk controller's cache. If the page is requested again while still stored in the NWCache, it can be re-mapped to main memory and taken off the ring.

The storage capacity of the ring is completely independent of the individual or combined local memory sizes. The storage capacity of the ring is simply proportional to the number of available channels, and the channels' bandwidth and length. More specifically, the capacity of the ring is given by: $capacity\_in\_bits = (num\_channels \times fiber\_length \times transmission\_rate) \div speed\_of\_light$, where $speed\_of\_light = 2 \times 10^8 m/s$.

Ring management. Each cache channel transfers and stores the pages swapped out by a particular node. A page can be swapped out to the NWCache if there is room available on the node's cache channel. An actual swap out to the NWCache allows the swapping node to re-use the space occupied by the page in memory right away. At the time of swapping, the node must set a bit (the Ring bit) associated with the page table entry for the page, indicating that the page is stored on the NWCache. The swapping node must also send a message to the NWCache interface of the corresponding I/O-enabled node. This message includes the number of the swapping node and the number of the page swapped out. The remote NWCache interface then saves this information in a FIFO queue associated with the corresponding cache channel.

Every time the disk controller attached to the interface has room for an extra page in its cache, the interface starts snooping the most heavily loaded cache channel and copies as many pages as possible from it to the disk cache. After the page is sent to the disk controller cache, the ACK is sent to the node that originally swapped the page out. The ACK is taken by the node to mean that it can now re-use the space occupied by the page on the ring and prompts it to reset the Ring bit associated with the page.

Two important characteristics of how pages are copied from the NWCache to the disk cache increase the spatial locality of the different writes in the cache: a) pages are copied in the same order as they were originally swapped out; and b) the interface only starts snooping another channel after the swap-outs on the current channel have been exhausted. When a node swaps consecutive pages out, these two characteristics allow several writes to be batched to disk in a single operation.

Pages can be re-mapped into memory straight from the NWCache itself. On a page fault, the faulting node checks if the Ring bit for the page is set. If not, the page fault proceeds just as described in the previous section. Otherwise, the faulting node uses the last virtual-to-physical translation for the page to determine the node that last swapped the page out. Then the faulting node can simply snoop the page off the correct cache channel. Moreover, the faulting node must send a message to the NWCache interface of the I/O-enabled node responsible for the page informing the interface of the page and cache channel numbers. This message tells the remote NWCache interface that the page does not have to be written to disk, since there is again a copy of it in main memory. The remote interface then takes the page number off of the cache channel's FIFO queue and sends the ACK to the node that originally swapped the page out to the NWCache.

Note that the NWCache does not suffer from coherence problems, since we do not allow more than one copy of a page beyond the disk controller's boundary. The single copy can be in main memory or on the NWCache.

Software cost. The software cost of the NWCache approach is negligible (provided that the kernel code is available, of course). The operating system code must be changed to include the Ring bits and to drive the NWCache interface.

Hardware cost. The electronic hardware cost of the NWCache is restricted to the I/O bus and disk interfaces, the FIFOs, and the buffers and drivers that interface the electronic and optical parts of the NWCache interface. The optical hardware requirements of the NWCache are also minimal. The NWCache interface at each node can read any of the cache channels, but can only write to the cache channel associated with the node and, thus, does not require arbitration. The NWCache interface regenerates, reshapes, and reclocks this writable cache channel. To accomplish this functionality, the interface requires two tunable receivers, one fixed transmitter, and one fixed receiver, as shown in figure 2. One of the tunable receivers is responsible for reading the pages to be written to disk from the NWCache, while the other tunable receiver is used to search the NWCache for a page that has been faulted on locally. The fixed transmitter is used to insert new data on the writable cache channel. In conjunction with this transmitter, the fixed receiver is used to re-circulate the data on the writable cache channel. Thus, the optical hardware cost for the ring is then only $4 \times n$ optical components, where n is the number of nodes in the multiprocessor. The number of cache channels is n. This cost is pretty low for small to medium-scale multiprocessors, even for today's optical technology costs. As aforementioned, commercial WDM products can be found with more than one hundred channels. Mass production of optical components and further advances in optical technology will soon make these costs acceptable even for large-scale multiprocessors.

Note that, even though our optical ring acts as a cache for disk data, the ring does not guarantee non-volatility (as some disk controller caches do), much in the same way as using idle node memory for swap-outs (e.g. [3]). This is not a serious problem however, since these approaches optimize the virtual memory management for applications that do not involve stringent reliability constraints, such as scientific applications.

4 Methodology and Workload

We use a detailed execution-driven simulator (based on the MINT front-end [20]) of a DASH-like [12] cache-coherent multiprocessor based on Release Consistency [4]. Memory, I/O, and network contention are fully modeled. The simulation of the multiprocessor operating system is limited to the part that really matters for our purposes: virtual memory management.

Our most important simulation parameters and their default values are listed in table 1. The values of the other parameters are comparable to those of modern systems, but have an insignificant effect on our results. The disk cache and main memory sizes we simulate were purposedly kept small, because simulation time limitations prevent us from using real life input sizes. In fact, we reduced optical ring and disk cache storage

Parameter	Value
Number of Nodes	8
Number of I/O-Enabled Nodes	4
Page Size	4 KBytes
TLB Miss Latency	100 pcycles
TLB Shootdown Latency	500 pcycles
Interrupt Latency	400 pcycles
Memory Size per Node	256 KBytes
Memory Bus Transfer Rate	800 MBytes/sec
I/O Bus Transfer Rate	300 MBytes/sec
Network Link Transfer Rate	200 MBytes/sec
WDM Channels on Optical Ring	8
Optical Ring Round-Trip Latency	52 usecs
Optical Ring Transfer Rate	1.25 GBytes/sec
Storage Capacity on Optical Ring	512 KBytes
Optical Storage per WDM Channel	64 KBytes
Disk Controller Cache Size	16 KBytes
Min Seek Latency	2 msec
Max Seek Latency	22 msecs
Rotational Latency	4 msec
Disk Transfer Rate	20 MBytes/sec

Table 1. Simulator Parameters and Their Default Values – 1 pcycle = 5 nsecs.

capacities by a factor of 32 and main memory sizes by a factor of 256 with respect to their sizes in real systems. Our goal with these reductions was to produce roughly the same swap-out traffic in our simulations as in real systems.

Our choice for amount of optical storage deserves further remarks. Increasing the size of the NWCache can be accomplished by lengthening the optical fiber (and thus increasing ring round-trip latency) and/or using more cache channels. With current optical technology the capacity of the NWCache could be increased a few fold, but certainly not by a factor of 32. Nevertheless, we believe that in the near future this magnitude increase in size will be possible. In fact, our capacity assumptions can be considered 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 [15].

Our application workload consists of seven parallel programs: Em3d, FFT, Gauss, LU, Mg, Radix, and SOR. Table 2 lists the applications, their input parameters, and the total data sizes (in MBytes) the inputs lead to. All applications *mmap* their files for both reading and writing[3] and access them via the standard virtual memory mechanism.

[3] The UNIX *mmap* call forces the user to specify the largest possible size of a writable file. This was not a problem for us, since it was always possible to determine the exact final size of each writable file for our applications. Nevertheless, we feel that the UNIX *mmap* call is much too restrictive for a pure virtual memory-based style of programming.

Program	Description	Input Size	Data (MB)
Em3d	Electromagnetic wave propagation	32 K nodes, 5% remote, 10 iters	2.5
FFT	1D Fast Fourier Transform	64 K points	3.1
Gauss	Unblocked Gaussian Elimination	570 × 512 doubles	2.3
LU	Blocked LU factorization	576 × 576 doubles	2.7
Mg	3D Poisson solver using multigrid techs	32 × 32 × 64 floats, 10 iters	2.4
Radix	Integer Radix sort	320 K keys, radix 1024	2.6
SOR	Successive Over-Relaxation	640 × 512 floats, 10 iters	2.6

Table 2. Application Description and Main Input Parameters.

5 Experimental Results

In this section we evaluate the performance of the NWCache comparing it against the performance of a standard multiprocessor. Before doing this, however, it is important to determine the best minimum number of page frames for each combination of prefetching technique and multiprocessor. We performed experiments where we varied this minimum number for each of the applications in our suite. In the presence of the NWCache, the vast majority of our applications achieved their best performance with a minimum of only 2 free page frames, regardless of the prefetching strategy.

The best configuration for the standard multiprocessor is not obvious. Under optimal prefetching, four of our applications (Gauss, LU, Radix, and SOR) favor large numbers (> 12) of free page frames, while two of them (Em3d and MG) achieve their best performance with only 2 free frames. The other application, Radix, requires 8 free frames for best performance. On the other hand, under naive prefetching, all applications except SOR favor small numbers (2 or 4) of free page frames. Thus, we picked 12 and 4 frames as the best minimum numbers of free frames under optimal and naive prefetching, respectively. All the results in this subsection will be presented for these configurations.

We are interested in assessing the extent to which the benefits provided by the NWCache actually produce performance improvements. As we have mentioned in the introduction, the NWCache has several performance benefits: it provides a staging area where swapped-out pages can reside until the disk is free; it increases the possibility of combining several writes to disk; it acts as a victim cache for pages that are swapped out and subsequently accessed by the same or a different processor; and it reduces the data traffic across the multiprocessor's interconnection network and memory buses. We now look at statistics related to each of these benefits in turn.

Write staging. Tables 3 and 4 show the average time (in pcycles) it takes to swap a page out of memory under the optimal and naive prefetching strategies, respectively. The tables show that swap-out times are 1 to 3 orders of magnitude lower when the NWCache is used. The main reason for this result is that the NWCache effectively increases the amount of disk cache from the viewpoint of the memory. A swap out is only delayed in the presence of NWCache when the swapping node's cache channel fills up. In contrast, when the NWCache is not assumed, swap outs are much more frequently delayed due to lack of space in the disk cache. As expected, the tables also show that

Application	Standard	NWCache
em3d	49.2	1.8
fft	86.6	3.1
gauss	30.9	1.0
lu	39.6	2.0
mg	33.1	0.6
radix	48.4	2.7
sor	31.8	1.3

Table 3. Average Swap-Out Times (in Mpcycles) under Optimal Prefetching.

Application	Standard	NWCache
em3d	180.4	2.8
fft	318.1	31.8
gauss	789.8	86.3
lu	455.0	24.3
mg	150.8	19.2
radix	1776.9	2.8
sor	819.4	12.5

Table 4. Average Swap-Out Times (in Kpcycles) under Naive Prefetching.

Application	Standard	NWCache	Increase
em3d	1.11	1.12	1%
fft	1.20	1.39	16%
gauss	1.06	1.07	1%
lu	1.13	1.24	10%
mg	1.11	1.16	5%
radix	1.08	1.12	4%
sor	1.46	2.30	58%

Table 5. Average Write Combining Under Optimal Prefetching.

Application	Standard	NWCache	Increase
em3d	1.10	1.10	0%
fft	1.35	1.38	2%
gauss	1.03	1.04	1%
lu	1.05	1.05	0%
mg	1.05	1.11	6%
radix	1.05	1.07	2%
sor	1.18	1.37	16%

Table 6. Average Write Combining Under Naive Prefetching.

swap-out times are much higher under the optimal prefetching technique than under its naive counterpart. This is a consequence of the fact that under optimal prefetching the reduced page read times effectively cluster the swap-outs in time, increasing contention.

Write combining. Given the way swap-outs are copied from the NWCache to the disk cache, the locality of the writes in the disk cache is often increased. When consecutive pages can be found in consecutive slots of the disk controller's cache, the writes of these pages can be combined in a single disk write access. The data in tables 5 and 6 confirm this claim. The tables present the average number of swap-outs that are combined in each disk write operation; maximum possible combining factor is 4, the maximum number of pages that can fit in a disk controller cache. Results show that increases in write combining are only moderate under the naive prefetching strategy, but can be significant under the optimal prefetching strategy. Again, the temporal clustering of swap-outs under optimal prefetching is responsible for this result. It becomes more common for the disk controller to find consecutive writes in its cache at the same time.

Victim caching. Table 7 presents the page read hit rates in the NWCache under the optimal and naive prefetching techniques. The table shows that hit rates are slightly higher under optimal prefetching than under naive prefetching, again due to the temporal characteristics of the page swap-outs under the two techniques. In addition, these results show that hit rates can be as high as 50+% (Gauss and MG) or as low as 10-% (Em3d). These results are a combination of two factors: the size of the memory working set and the degree of data sharing in the applications. Gauss, MG, and Em3d exhibit a significant amount of sharing, but only Gauss and MG have working sets that can (almost) fit in the combined memory/NWCache size. The other applications achieve hit

Application	Naive	Optimal
em3d	8.5	10.0
fft	9.8	13.0
gauss	49.9	58.3
lu	13.5	19.5
mg	41.1	59.1
radix	17.2	22.6
sor	25.8	24.1

Table 7. NWCache Hit Rates Under Different Prefetching Techniques.

Application	Standard	NWCache	Reduction
em3d	13.4	9.7	28%
fft	25.9	19.6	24%
gauss	16.7	10.4	38%
lu	21.5	20.3	6%
mg	19.1	6.7	63%
radix	12.6	9.2	27%
sor	14.3	10.2	29%

Table 8. Average Page Fault Latency (in Kpcycles) for Disk Cache Hits Under Naive Prefetching.

rates in the 10 to 25% range.

The beneficial effect of victim caching is more pronounced under naive prefetching, since read fault overheads represent a significant fraction of the applications' execution times. Furthermore, pages can be read only slightly faster from the NWCache than the disk controller's cache, which reduces the potential gains of victim caching under optimal prefetching.

Contention. The NWCache alleviates contention for two reasons: a) page swap-outs are not transferred across the interconnection network; and b) page reads that hit in the NWCache are transferred neither across the network nor across the I/O node's memory bus (whenever the request for the corresponding I/O node can be aborted in time). In order to evaluate the benefit of the NWCache in terms of contention alleviation, we collected statistics on the average latency of a page read from the disk controller's cache under the naive prefetching technique. A comparison of the statistics from the standard multiprocessor against those of the NWCache-capable multiprocessor provides a rough estimate of the amount of contention that is eliminated.

Table 8 displays these data in thousands of processor cycles. The table shows that the NWCache reduced disk cache hit latencies by as much as 63%. For most applications, reductions range from 24 to 38%. Given that it takes about 6K pcycles to read a page from a disk cache in the total absence of contention, we can see that the contention reductions provided by the NWCache are always significant. For instance, the average disk cache hit takes about 21K pcycles for LU running on the standard multiprocessor. Out of these cycles, about 15K are due to contention of several forms. In the presence of the NWCache, the number of cycles due to contention in LU is reduced to about 14K, which means a 7% reduction. At the other extreme, consider the reduction in disk cache hit latency achieved by MG, 63%. Out of the 19K cycles that MG takes to read pages from disk caches on the standard multiprocessor, about 13K are due to contention. In the presence of the NWCache, the number of cycles due to contention in MG is reduced to about 700, which means a 95% reduction in contention.

Under optimal prefetching the NWCache is not as successful at alleviating contention, since there is usually not enough time to prevent a page transfer through the network and the I/O node bus when the page read request hits in the NWCache.

The results we presented thus far confirm that the NWCache indeed benefits performance significantly in several ways. The greatest performance gains come from very fast swap-outs, victim caching, and contention reduction.

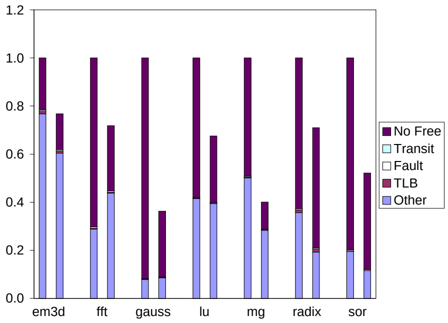

Fig. 3. Performance of Standard MP (left) and NWCache MP (right) Under Optimal Prefetching.

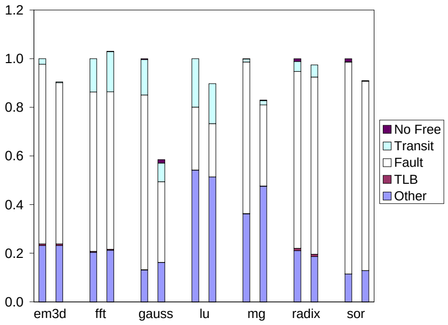

Fig. 4. Performance of Standard MP (left) and NWCache MP (right) Under Naive Prefetching.

Overall performance. Figures 3 and 4 show the normalized execution times of each of our applications under optimal and naive prefetching, respectively. From top to bottom, each bar in the graphs is divided into: the stall time resulting from the lack of free pages frames ("NoFree"); the overhead of waiting for another node to finish bringing a page into memory ("Transit"); the overhead of page faults ("Fault"); the overhead of TLB misses and TLB shootdown ("TLB"); and the components of execution time that are not related to virtual memory management ("Others"), including processor busy, cache miss, and synchronization times.

Figure 3 shows that under the optimal prefetching strategy, NoFree overheads are always very significant for the standard multiprocessor, especially for Gauss and SOR. The operating system frequently runs out of free frames on the standard multiprocessor due to the fact that page reads are completed relatively quickly, while page swap-outs are very time-consuming. When the multiprocessor is upgraded with the NWCache, NoFree times are reduced significantly as a result of its much faster page swap-outs.

The figure also demonstrates that for several applications the time taken on non-virtual memory operations is significantly reduced in the presence of the NWCache. These reductions come from improvements in the cost of remote data accesses and better synchronization behavior, both of them a result of the significant reduction of the traffic through the memory system (network and memories). Overall, we see that the NWCache provides performance improvements that average 41% and range from 23% (Em3d) to 60 and 64% (MG and Gauss), when optimal prefetching is assumed. In fact, improvements are greater than 28% in all cases, except Em3d.

The performance results when naive prefetching is assumed are quite different. Under this technique, execution times are dominated by page fault latencies, since disk cache hit rates are never greater than 15%. The page fault latencies then provide the much needed time for swap-outs to complete. As a result, NoFree times almost vanish, diminishing the importance of the fast swap-outs in the NWCache architecture.

Under naive prefetching, the addition of the NWCache to the multiprocessor improves performance from 3 (Radix) to 42% (Gauss) for all applications except FFT, which degrades by 3%. NWCache-related improvements come from reasonable reductions in page fault latencies, which result from reading pages off the optical cache and from alleviating contention.

Discussion. In summary, we have shown that the NWCache is extremely useful when prefetching is effective, mostly as a result of fast swap-outs. The NWCache is not as efficient when prefetching is either ineffective or absent, even though victim caching and contention alleviation improve the performance of many applications significantly. We expect results for realistic and sophisticated prefetching techniques [14, 9] to lie between these two extremes. In addition, as prefetching techniques improve and optical technology develops, we will see greater gains coming from the NWCache architecture.

6 Related Work

As far as we are aware, the NWCache is the only one of its kind; an optical network/write cache hybrid has never before been proposed and/or evaluated. A few research areas are related to our proposal however, such as the use of optic fiber as delay line memory and optimizations to disk write operations.

Delay line memories. Delay line memories have been implemented in optical communication systems [11] and in all-optical computers [8]. Optical delay line memories have not been used as write 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, 6]. The only aspects of optical communication that these designs exploit are its high bandwidth and the ability to broadcast to a large number of nodes. Besides these aspects, the NWCache architecture also exploits the data storage potential of optics.

Disk write operations. Several researchers have previously set out to improve the performance of write operations for various types of disk subsystems. These efforts include work on improving the small write performance of RAIDs (e.g. [19]), using non-volatile RAM as a write cache (e.g. [17]), logging writes and then writing them to disk sequentially (e.g. [16]), using idle or underloaded node memory for storing page swap-outs [3], and using a log disk to cache writes directed to the actual data disk [7]. The two latter types of work are the closest to ours.

Storing page swap-outs in another node's memory is only appropriate for workstation networks where one or more nodes may be idle or underloaded at each point in time. The same technique could not be applied in our computing environment, a multiprocessor running an out-of-core application, since processors are always part of the computation and usually do not have spare memory they could use to help each other.

The storage architecture proposed in [7], the Disk Caching Disk (DCD), places the log disk between the RAM-based disk cache and the data disk, effectively creating an extra level of buffering of writes. New data to be written to disk is then stored in the RAM cache and later written sequentially on the log disk. Overwriting or reading a block requires moving around the log disk to find the corresponding block. When the data disk is idle, the data is copied from the log disk to the data disk. This scheme improves performance as it reduces seek and rotational latencies significantly when writing new data to the log disk and, as a result, frees space in the RAM cache more rapidly. Overwriting or reading involves seek and rotational latencies that are comparable to those of accesses to the data disk.

Just like the DCD, the NWCache also attempts to improve the write performance by creating an extra level of buffering. However, the NWCache places this buffer between main memory and the disk caches and thus does not require any modifications to standard disk controllers. In addition, overwriting or reading data from the NWCache is as efficient as writing new data to the NWCache. Another advantage of the NWCache is that it creates an exclusive path for disk writes to reach the disk controllers. Nevertheless, the technology used to implement the additional buffering in the DCD allows it much more buffer space than our optics-based buffer.

7 Conclusions

In this paper we proposed a novel architectural concept: an optical network/write buffer hybrid for page swap-outs called NWCache. The most important advantages of the NWCache are its fast swap-outs, its victim caching aspect, and its ability to alleviate contention for memory system resources. Through a large set of detailed simulations we showed that an NWCache-equipped multiprocessor can easily outperform a traditional multiprocessor for out-of-core applications; performance differences in favor of our system can be as large as 64% and depend on the type of disk data prefetching used. Based on these results and on the continually-decreasing cost of optical components, our main conclusion is that the NWCache 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.

References

1. A. Agarwal, R. Bianchini, D. Chaiken, K. Johnson, D. Kranz, J. Kubiatowicz, B.-H. Lim, K. Mackenzie, and D. Yeung. The MIT Alewife Machine: Architecture and Performance. In *Proceedings of the 22nd International Symposium on Computer Architecture*, June 1995.
2. R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. The Tera Computer System. In *Proceedings of the 1990 International Conference on Supercomputing*, July 1990.
3. E. Felten and J. Zahorjan. Issues in the Implementation of a Remote Memory Paging System. Technical Report 91-03-09, Department of Computer Science and Engineering, University of Washington, March 1991.
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. 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.
7. Y. Hu and Q. Yang. DCD – Disk Caching Disk: A New Approach for Boosting I/O Performance. In *Proceedings of the 23rd International Symposium on Computer Architecture*, pages 169–177, May 1996.
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. T. Kimbrel *et al.* A Trace-Driven Comparison of Algorithms for Parallel Prefetching and Caching. In *Proceedings of the 2nd USENIX Symposium on Operating Systems Design and Implementation*, October 1996.
10. D. Kotz and C. Ellis. Practical Prefetching Techniques for Multiprocessor File Systems. *Journal of Distributed and Parallel Databases*, 1(1):33–51, January 1993.
11. R. Langenhorst *et al.* Fiber Loop Optical Buffer. *Journal of Lightwave Technology*, 14(3):324–335, March 1996.
12. D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta, and J. Hennessy. The DASH Prototype: Logic Overhead and Performance. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):41–61, January 1993.
13. K. McKusick, W. Joy, S. Leffler, and R. Fabry. A Fast File System for UNIX. *ACM Transactions on Computer Systems*, 2(3):181–197, August 1984.
14. T. Mowry, A. Demke, and O. Krieger. Automatic Compiler-Inserted I/O Prefetching for Out-Of-Core Applications. In *Proceedings of the 2nd USENIX Symposium on Operating Systems Design and Implementation*, October 1996.
15. 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.
16. M. Rosenblum and J. Ousterhout. The Design and Implementation of a Log-Structured File System. *ACM Transactions on Computer Systems*, 10(2):26–52, February 1992.
17. C. Ruemmler and J. Wilkes. UNIX Disk Access Patterns. In *Proceedings of the Winter 1993 USENIX Conference*, January 1993.
18. D. B. Sarrazin, H. F. Jordan, and V. P. Heuring. Fiber Optic Delay Line Memory. *Applied Optics*, 29(5):627–637, February 1990.
19. D. Stodolsky, M. Holland, W. Courtright III, and G. Gibson. Parity Logging Disk Arrays. *ACM Transactions on Computer Systems*, 12(3):206–235, August 1994.
20. 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*, January 1994.
21. D. Womble, D. Greenberg, R. Riesen, and D. Lewis. Out of Core, Out of Mind: Practical Parallel I/O. In *Proceedings of the Scalable Parallel Libraries Conference*, October 1993.

This article was processed using the LaTeX macro package with LLNCS style