Evaluating the Performance Impact of Dynamic Handle Lookup in Modern Network Interfaces

Reza Azimi
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario M5S 3G4, Canada
Email: azimi@eecg.toronto.edu

Angelos Bilas
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario M5S 3G4, Canada
Email: bilas@eecg.toronto.edu

***Abstract*— Recent work in low-latency, high-bandwidth communication systems has resulted in building user–level Network Interface Controllers (NICs) and communication abstractions that support direct access from the NIC to applications virtual memory to avoid both data copies and operating system intervention. Such mechanisms require the ability to directly manipulate user–level communication buffers for delivering data and achieving protection. To provide such abilities, NICs must maintain appropriate translation data structures. Most user–level NICs manage these data structures statically which results both in high memory requirements for the NIC and limitations on the total size and number of communication buffers that a NIC can handle.**

In this paper, we categorize the types of data structures used by NICs and propose *dynamic handle lookup* as a mechanism to manage such data structures dynamically. We implement our approach in a modern, user–level communication system, we evaluate our design with both micro-benchmarks and real applications, and we study the impact of various cache parameters on system performance. In this work we focus mostly on the results of our work. We find that, with appropriate cache tuning, our approach reduces the amount of NIC memory required in our system by a factor of two for the total NIC memory and by more than 80% for the lookup data structures. For larger system configurations the gains can be even more significant. Moreover, our approach eliminates the limitations imposed by current NICs on the amount of host memory that can be used for communication buffers. Our approach increases execution time by at most 3% for all but one applications we examine.

I. Introduction

Recent work in improving the performance of interconnection networks in scalable servers and storage systems has resulted in new network interface controllers (NICs) that support user–level communication [1], [2], [3]. The main capabilities of these NICs are: (i) the ability for user programs to directly access the network in a protected manner for sending and receiving data and (ii) data transfer directly to- and from program virtual memory without OS intervention. These capabilities are now part of industry standards that are in use today or are being proposed for future interconnects [4], [5], [6]. However, modern NICs require certain extensions to provide these capabilities.

First, they require efficient support for translating between virtual and physical memory addresses. NICs usually perform DMA transfers only to and from physical host memory. However, user programs usually employ virtual addresses to specify communication buffers. Therefore, most user–level communication standards [4], [5] and NICs [3], [2] require efficient address translation by the NIC.

Second, to allow user programs to directly access the network without OS intervention, NICs must be able to verify user requests. For this purpose, the virtual memory regions used as communication buffers must be registered with the NIC prior to the actual communication operations. Registration implies that the NIC is aware of virtual memory regions that can be used for direct data transfer. Therefore, NICs must maintain information about registered communication buffers [7]. Also, such systems require a mechanism to authenticate remote programs. This requires NICs to maintain authentication information for point–to–point communication connections. Finally, given that most servers today are required to support multi–programmed workloads, NICs must handle requests of multiple user processes simultaneously. For protection purposes, NICs must directly identify local user processes by some handle that is used to retrieve user-specific communication contexts.

Therefore, NICs must lookup at least three different types of resources during their operation: (i) address translation information (ii) registered communication buffers and authentication information, and (iii) processes communication context information. The solution employed in most NICs is to use static lookup tables in NIC memory. The user applications must provide all lookup entries required for the NIC operation prior to any communication operation. This static method for management of NIC memory has two major implications.

First, it requires modern NICs to support large amounts of on-board memory at significant cost. For instance, Myrinet NICs are priced as shown in Table I [8]. We see that increasing the amount of memory by 2 MBytes increases the NIC cost by more than 30% whereas increasing memory by 6 MBytes increases the NIC cost by more than 60%. Other products, such as the VI/IP and iSCSI NICs by Emulex [9] employ higher amounts of on-board memory at higher costs. Generally, in many modern clusters the cost of the system area network (including switch costs) that interconnects the nodes of the system is about the same as the cost of the system nodes.

Second, and more importantly, all modern NICs impose static limits on the amount of host memory that can be used

NIC Memory Size (MBytes)	2	4	8
NIC Price (USD)	995	1,295	1,595

TABLE I
MEMORY CONFIGURATION AND PRICE INFORMATION FOR THE MYRINET NICS.

for communication buffers. In all cases the user is able to only use a subset of the host memory for communication. This is a severe limitation for application servers in various domains, such as compute servers [7] or database storage servers [10], especially as application needs change over time and among applications in multiprogrammed workloads. The solution that has been employed in these cases is some type of application level management of communication buffers, which is cumbersome, complex, and sometimes not even possible due to OS restrictions. The easier approach is to further increase the amount of NIC memory. However, this only postpones the problem until application working sets increase further and unnecessarily increases the cost of the base system.

In this paper, we evaluate the performance implications of a generic scheme, called *dynamic handle lookup* [11] that maintains all important lookup tables in the larger host memory and uses parts of NIC memory as a cache. Dynamic handle lookup eliminates the limitations of most user–level communication systems due to the limited size of NIC memory. Such limitations are in particular imposed on the total size of the communication virtual address space, the total number of communication buffers and connections, and the number of processes that use the system concurrently. In our evaluation we use both synthetic micro–benchmark and real applications from the SPLASH-2 [12] suite.

Overall, we find that by using dynamic handle lookup NIC memory size can be reduced substantially by as much as 50% for the total amount of memory and up to 75% for lookup data structures. The overhead of using more complex dynamic lookup structures is at most 3% across all but one applications we examine.

The rest of the paper is organized as follows. Section II provides the necessary background on user–level communication systems. Section III discusses related work. Section IV summarizes the design of dynamic lookup mechanisms. Section V presents the results of our performance evaluation and analysis, both with synthetic micro-benchmarks and real applications. Finally, Section VI draws our conclusions.

II. USER–LEVEL COMMUNICATION

User–level comunication systems support direct user access to network resources and data transfer operations to local and remote virtual memory without host processor and OS intervention [13], [14], [2]. They use the capabilities of modern NICs to reduce communication overheads. Modern NICs usually employ a communication assist, which can be a special–purpose network processor or a general–purpose processor and usually few MBytes of static or dynamic memory. NICs also have one or more DMA engines for transferring data between the host memory and the NIC buffers and also between the NIC buffers and the network link. The firmware runs on the communication assist and implements the communication protocol by managing NIC resources, mainly controlling the DMA engines and responding to system events.

User–level communication systems usually consist of three major components: (i) A kernel–level device driver that is responsible for initializing the NIC and performing trusted functions that cannot be performed directly by the user. (ii) A network control program (NCP) that runs on the NIC and performs all protocol processing tasks. The NCP usually implements a set of state machines that handle all system events. (iii) A lightweight user–level library to provide the communication API for user programs.

The user–level library in co-operation with the NCP manages per-process, memory-mapped, send, receive, and completion queues on NIC memory. The path in which data is transferred from host memory to NIC memory and then to the network link is called *send path*. The path in which data is transferred from the network link to NIC memory and then to host memory is called *receive path*.

User–level communication systems provide two major types of communication operations, connection establishment and data transfer operations. In general, NIC memory is divided into three regions:

NCP code: This region contains the firmware code that runs on the communication assist and handles all system events. It usually varies between 50-150 KBytes.

Send and receive data buffers: Data that is being received or sent is first transfered to buffers on the NIC. Such buffers are short-lived and usually a small number is adequate for good performance [15]. Furthermore, they are always managed dynamically and their number can be adjusted based on the available NIC memory.

Lookup data structures: This region contains all data structures needed for lookup operations. The most important types of such lookup data structures are used to maintain information for address translation, registered communication buffers, connections, and process communication contexts.

Unlike the NCP code and the data buffers, the size of the lookup data structures grow with the application communication working set size. Given the current trend for increasing application working set sizes, applications that use user-level communication to improve the cost of communication operations require large numbers of communication buffers with a corresponding increase in the size of lookup data structures. For instance, application servers with 64 GBytes of main memory would require approximately 64 MBytes of memory only for virtual address translation, which exceeds by far the capabilities of modern NICs used in system area networks. Furthermore, building NICs with similar capabilities increases the NIC cost significantly. In summary, in cluster configurations where communication among nodes is intense, the lookup data structures currently constitue the largest com-

ponent of NIC memory.

III. Related Work

The design and implementation of dynamic handle lookup is presented in more detail in [11]. In this work we focus on the performance implications of these mechanisms.

The authors in [16] propose a User-managed Translation Look-aside Buffer (UTLB) to dynamically manage translations for virtual memory used as send communication buffers. UTLB is implemented as a per-process two-level custom page table in the host memory. A shared TLB cache resides in NIC memory and caches the most popular entries of the driver-level tables. On a miss the NIC fetches the appropriate table entry using DMA. However, UTLB's dyanmic approach is only applicable to the send path of the NIC. Address translation on the receive path is static, which means that all translation entries for registered communication buffers reside in the NIC memory throughout program execution. This results in high memory requirements and limitations on the total size as well as the number of the registered communication buffers. The limitations of this approach have motivated to a large extent our work.

The authors in [17] propose the concept of *virtual networks* to address the issue of supporting large numbers of users over fast, user-level communication systems. The main goal of the work is to efficiently multiplex the system resources among multiple applications with different demands. The main abstraction is the network *endpoint* that consists of the process send and receive queue and the address translation information of the static buffers involved in the communication. Endpoints reside in the host memory and are cached on NIC memory. Unlike many user–level communication systems, the virtual networks abstraction does not support RDMA operations to arbitrary memory locations, which eliminates the need for permanent lookup entries in the NIC memory.

U-Net/MM [18] is an extension of the U-Net [19] system that also uses endpoints as the main communication facility. Each endpoint is associated with a buffer area that is pinned to contiguous physical memory and holds all buffers used with that endpoint. The U-Net/MM incorporates a TLB structure, in order to handle arbitrary user-space virtual addresses. Similar to our work, U-Net/MM uses an interrupt-based mechanism for handling TLB misses. However, U-Net/MM does not support RDMA operations. Thus, applications can provide the NIC with address translation information upon posting send and receive requests, whereas with RDMA operations, data transfer occurs asynchronously. Moreover, in U-Net/MM, whenever there is an eviction from the TLB, the OS has to be notified to unpin the corresponding pages. In contrast, in *miNI* there is no need to notify the host system in the case of an eviction. Furthmore, U-Net/MM does not deal with the protection issues of communication buffers. Finally, UNet/MM is only evaluated with micro-benchmarks, and therefore, there is little evidence how different communication working set size of the real applications affects its performance.

The Virtual Interface (VI) architecture [4] is an industry standard for user-level communication. The VI supports both send and receive operations as well as RDMA operations. Similar to *miNI*, communication buffers must be registered prior to any data transfer. The registration includes both pinning the virtual region and setting up the protection information for it. In current VI implementations [20], [21], [22], [3] all registered regions are statically pinned. This limits the total size of the registered buffer to a fraction of the total physical memory available, unless the user process dynamically manages registration and deregistration of communication buffers. For instance, the cLAN NICs [3] impose 1GByte limit on registered memory, which is very little for modern applications [10]. Another industry standard in system area networks is Infiniband [5]. The core operations and protection model of Infiniband is significantly influenced by VI. More specifically, it follows the same rules as VI in registering the communication buffers with small refinements in the protection model.

Finally authors in [23] present a survey of different mechanisms used for address translation in NICs. They define four requirements for address translation: (i) Flexibility of use for higher system layers. (ii) The ability to cover all of the user address space. (iii) The ability to take advantage of locality. (iv) Graceful degradation when system limits are exceeded. They evaluate various alternative methods of implementing address translation in NICs by using simulation. They find that hardware lookup structures in the NIC are not required since the software schemes are fast enough. They suggest that the NIC should handle all address translation misses which introduces the limitations discussed in this work.

IV. Design of Dynamic Handle Lookup

In this section we summarize the dynamic handle lookup mechanisms we evaluate. A more detailed description of these mechanisms can be found in [11]. However, we first define some important terms that are used frequently in the description of our design.

Virtual Memory Handle: It specifies the virtual address in host memory of the source or destination location for data transfer operations. VMHs are used for virtual to physical address translation in the send and receive path of the NIC. In our current implementation the VMH is 48 bits and is composed from a virtual address and the process Id.

Communication Buffer Handle: It specifies the communication buffer used in data transfer operations. CBHs are used by the receiving NIC to obtain protection information about a buffer and by the sending NIC to specify a communication buffer in the cluster. In our current implementation the CBH is 48 bits and is composed from the 32-bit buffer Id and the 16-bit process Id.

Process Communication Handle (PCH): In multiprogrammed workloads, it specifies which process is involved In a communication operation. In our implementation the PCH is a 16-bit integer.

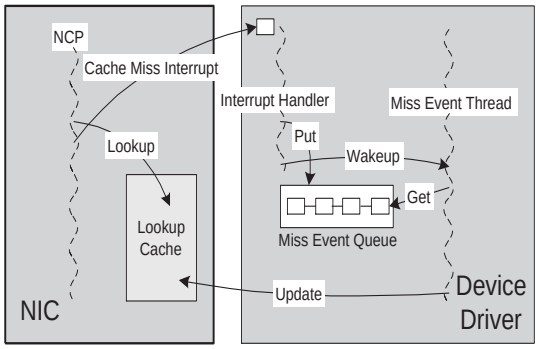

Fig. 1. Cache miss handling path.

A. Basic mechanisms

The main idea in dynamic handle lookup is to move all lookup tables to the much larger host memory and use the NIC memory as a cache. On a cache miss, the NCP issues an interrupt to the host CPU that is handled by the NIC device driver. The handler fetches the missed entries from lookup tables in host memory and places them in the cache in NIC memory. The NIC cache is shared among all processes that use the communication system. This allows for better management of the NIC memory. However, the performance of the communication system might substantially degrade if many processes compete for the same cache entries.

The two major problems in such a design are: (i) How to fetch the missed entries from the host memory and (ii) how should the NCP handle requests that miss.

(i) There are two ways for the NCP to request entries that miss: To interrupt the host CPU or to DMA the entries directly from the host memory itself. The latter is faster and does not impose any overhead on the host CPU. However, it requires that the host CPU prepares all request entries before the execution of the communication operation that needs the entries. This requires a priori knowledge of what operations are going to take place, which is difficult to obdain for long RDMA operations. Moreover, with this method the NCP must be aware of the physical structure of the lookup tables in host memory, which limits the design of the lookup tables.

Furthermore, handling cache miss interrupts can be done in two ways: Using an interrupt handler or using a separate thread running in the kernel. The former is faster since it incurs no context switch overhead and unpredictable scheduling delay. However, this approach requires implementing custom internal kernel functions to provide the necessary functionality. Moreover, interrupt handlers cannot be blocked in many OSs, limiting the kernel functions that can be called in the interrupt handler context.

In this work we choose to handle lookup cache miss interrupts in a separate thread running in the kernel. Figure 1 shows the cache miss handling path. The NIC issues a miss interrupt when a lookup in one of the cache data structures misses. The interrupt handler in the kernel device driver just prepares a *miss event descriptor* with the information given by the NIC and places it in the miss event queue. It then wakes up the in-kernel *miss event thread*, which does all processing and updates the on-NIC cache.

(ii) When a cache miss happens during processing a request, the NCP blocks the request and processes other active connections. Meanwhile, the NCP has two ways to handle the blocked request: To buffer the request in the NIC memory until the missed entries are fetched from the lookup tables in the cache or to drop the request and rely on the underlying transport protocol to retransmit the unacknowledged request. Buffering requests requires a potentially high amount of NIC memory, since the miss handling delay from host memory may be long. Moreover, it complicates the design of the NCP. On the other hand, dropping a request is simple and requires no NIC memory. However, it incurs the retransmission overhead. In *miNI*, we use the second approach, of dropping requests that miss, to both minimize NIC memory requirements and simplify NCP implementation.

In summary, in our design we favor design choices that lead to a simpler and more portable implementation, despite the fact that this may increase miss penalties. Our intention is to not only understand and build systems that perform well, but systems that are robust and realistic. Performance-wise, as explained in Section V, our evaluation confirms our intuition that we can limit the overall performance overhead by reducing the miss rate with proper cache configuration and tuning.

Although we have implemented dynamic handle lookup for all important data structures, we only present the detailed evaluation of dynamic VMH lookup here.

B. VMH Lookup

VMH lookup is required both in the send and the receive paths. Also, RDMA read and write operations are handled differently. For RDMA reads, in the send path data is sent as the result of a remote read request. In this case the RDMA read request contains the CBH of the communication buffer and the offset of the data to be read within the buffer. The CBH is used to lookup the virtual address of the start of the buffer. The start address is added to the offset to form the VMH of the area from which the data must be sent. In the receive path, NIC processing is simpler since the VMH of the destination of the data on the host memory has already been posted at the time of issuing the RDMA operation. Conversely, for RDMA writes, in the receive path data has to be written into a communication as a result of an RDMA write operation, issued from a remote node. In this case, the RDMA request contains the CBH of the communication buffer and the offset within the buffer. The CBH is used to lookup the starting virtual address of the communication buffer. This address is added to the offset to form the VMH of the area to which the data will be written. For RDMA writes the send path is simpler since data is sent as a result of an RDMA write request issued at the local node and the VMH of the source data is in the request descriptor posted by the user process.

There are two data structures that hold the address translation information used during VMH lookup: (i) the *VMH*

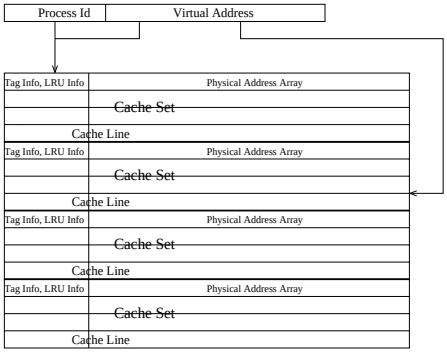

Fig. 2. The structure of the VMH lookup cache.

Lookup Table that resides in host memory and (ii) the *VMH Lookup Cache* that is located in NIC memory and acts as a cache for the VMH table.

1) VMH Lookup Table: For each process, there is one VMH table in the device driver that keeps the virtual–to–physical address mappings for all pinned pages of the process. When there is a miss in the VMH cache, the device driver looks for the missed entries in the VMH table. If they are found, the driver just updates the VMH cache. Otherwise, it pins all the pages in the missed address range and updates both the VMH table and the VMH cache with the physical addresses.

2) VMH Lookup Cache: The send and receive paths of the NIC use the same cache for uniformity and simplicity. We use a set-associative cache structure with configurable associativity, cache line size, and cache size. Figure 2 shows the structure of the VMH cache. We use the lower bits of the virtual address in the VMH as the set index and the rest of the virtual address bits and the process Id as the tag information. This prevents static partitioning of the cache among processes. Also, it reduces the chance of conflicts among the addresses close to each other for which we expect spatial locality of access. The VMH cache is shared among all processes using the system.

The VMH cache lines can have multiple entries. The cache line size can be configured at compile time. Multiple–entry cache lines allow for a smaller tag-array. For each entry in a cache line, the full 32-bit word is allocated for the physical page address, allowing up to 16 TBytes of physical memory to be used for communication buffers. The cache line is also the unit of data transfer between the VMH table and cache. Therefore, when there is a miss for a virtual address in the cache, the missed virtual address will be aligned to the cache line boundaries. This may result in effective prefetching for applications with sufficient spatial locality in their communication buffer access pattern. However, this also increases the cache miss penalty, mostly due to larger pinning costs.

The VMH cache can be configured with different associativity levels. Although with higher levels of associativity the number of conflicts may be reduced, the cost of lookup in the cache tag array increases, which is specially an issue in caches implemented in software.

The VMH cache uses an LRU replacement policy for the entries of each set. The cache replacement is implemented in the NIC. Since the NCP access to the VMH cache is read–only, the replacement does not include any write–back operation and the entries to be replaced are simply thrown away. When an entry is replaced in the VMH cache, it remains in the host VMH table. Entries in the VMH table are pinned as well. However, as mentioned above, once a line is evicted from the VMH cache, it also becomes a candidate for eviction from the VMH table, in case it reaches the high water mark. However, VMH table replacement happens periodically with long intervals. If a page is accessed frequently, it will be brought back to the VMH cache and thus, will not be evicted from the VMH table.

V. Results

The goal of our evaluation is twofold: (i) we would like to tune the lookup cache design parameters in our extensions so that we both reduce the memory requirements as well as impose little additional overhead to real applications, and (ii) we are interested in gaining insight on how the system behaves as the parameters vary within a wide range of values and applications.

In this work, we use Virtual Memory Mapped Communication (VMMC) [14] as the base for implementing our approach. VMMC provides protected, user–level communication between the sender's and the receiver's virtual address spaces. VMMC guarantees FIFO message delivery between any two processes in the system and tolerates transient network errors by using packet retransmission. More specificlly, VMMC includes a reliability mechanism [15] that retransmits packets until they are acknowledged by the peer node.

The experimental system we use for evaluation is a cluster of four 2-way PentiumIII nodes. The exact configuration of each node is shown in Table II. The nodes in the cluster are interconnected with a Myrinet network [2] through a 16-port, full crossbar, Myrinet switch. The PCI–based NIC is composed of a 32–bit, general-purpose processor (LANai9) with 2 MBytes of SRAM.

Processors	2 x Intel Pentium III, 800 MHz
Cache	32K (L1), 512K (L2)
Memory	512MB SDRAM
OS	RedHat Linux Kernel 2.2.16-3smp
PCI buses	32 bits, 33 MHz, 133MBytes/s
NIC	Myricom M3M-PCI64B
NIC CPU	LANai9, 133 MHz
Network Link	Bi-directional, 160MBytes/s/direction

TABLE II
Cluster node configuration.

To evaluate the impact of our approach on the performance of the system, we use both synthetic micro-benchmarks as well as real applications. We use micro-benchmarks mainly to provide basic measurements in uncontended conditions and to independently stress specific components of the system.

The applications we use are a subset of the SPLASH-2 suite [12] on top of a shared virtual memory (SVM) system that provides the illusion of a single system image.

The specific applications we use are FFT, WaterNsquared, WaterSpatial, LUContiguous, Ocean, Volrend, Radix, and Raytrace. This set of applications covers a wide range of communication patterns. The SVM protocol we use is GeNIMA [24], a home-based, page-level protocol. GeNIMA has been optimized to be used with system area networks that support remote RDMA operations.

For the real applications, we use the largest problem set size permitted by the 2-GByte address space of the 32-bit Linux OS. Using larger cluster configurations would result in less stress for the NIC memory subsystem, since the application working set would be divided among more nodes and therefore, the communication working set of each node would be smaller.

In this work we present detailed results for VMH lookup, since it is more critical to system performance compared to CBH and PCH lookup. We present statistics on both miss rates and application speedups. We divide cache misses based on two factors: (i) the path in which they occur (send or receive) and (ii) miss type, i.e. cold misses versus capacity or conflict misses (non-cold) [25]. The first factor tells us about the communication working sets in either paths for each application whereas the second helps us understand better the impact of each cache parameter.

A. NIC Memory Requirements

By using dynamic handle lookup in our system we are able to reduce the total NIC memory requirements from 800 KBytes to 400 KBytes. In particular, the memory used for all lookup structures is reduced by about 75% from 520 KBytes to 130 KBytes. Figure 3 summarizes the breakdown of the memory consumption for the code and various data structures on the NIC memory in the base system and after our modifications. More importantly, dynamic handle lookup does not have any of the constraints on the total size and number of communication buffers and the total number of connections of the base system.

Memory saving on larger system configurations can be even more significant. A system that uses 64 GBytes of host memory for application communication buffers, as is common in data-base servers [10], would require about 32 MBytes of NIC memory for VMH translation, which exceeds the capabilities of most NICs, even excluding CBH translation. Extrapolating from our results, our approach would require at most 8 MBytes of memory (including all NIC data structures), which is within the capabilities of existing NICs. Moreover, our approach is able to use all 64 GBytes of host memory for communication, even with smaller NIC memories, although, potentially at a higher performance cost.

B. Micro-benchmarks

Figure 4 shows how latency and bandwidth vary as we change the VMH cache miss ratio. The micro-benchmark we use is essentially a ping-pong test, where requests miss on one of the two nodes only. The micro-benchmark is able to control the miss rate by using specific source and destination buffers for communication. We see that the bandwidth degrades rapidly and the latency increases linearly as the miss ratio increases. In both cases the negative effect of a miss is higher for larger message sizes. This is due to the fact that the cost of dropping and retransmitting on a miss increases with message size, due to the lack of negative acknowledgments and receive side buffering in our system.

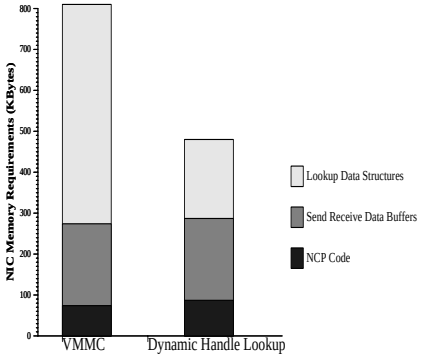

Fig. 3. The breakdown of NIC memory requirements in VMMC and Dynamic Handle Lookup.

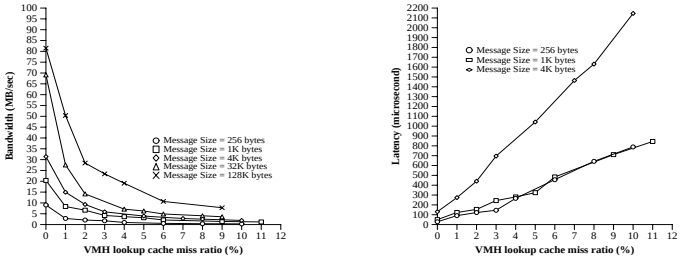

Fig. 4. The effect of VMH miss on ping-pong latency and bandwidth.

C. Real Applications

For each application we choose a fairly large problem size to stress the VMH lookup cache. Due to the importance of VMH translation for the common communication path we mostly focus our evaluation with real applications on the VMH cache.

Parameter	Values
Cache Size **(C)**	8K, 16K, 32K entries
Cache Line **(L)**	8, 16, 64, 128 entries
Associativity **(A)**	2, 4, 8 lines

TABLE III
THE RANGE OF VALUES FOR EACH OF THE VMH LOOKUP CACHE PARAMETERS.

We vary each of the three VMH lookup cache design parameters, cache size (**C**), line size (**L**), and associativity (**A**) with a wide range, as shown in Table III. We first

examine the impact of each cache parameter on the cache miss ratio. Figure 5 shows the VMH lookup cache miss ratio breakdown for each set of parameters. The first observation is that the number of misses varies greatly, between about 0-40%, among different configurations. Overall, the miss ratio of WaterNsquared, WaterSpatial, Ocean, Raytrace, and Volrend is within the range 0-2% range, whereas for FFT, LUContiguous, and Radix the miss ratio is much larger. Second, we observe that all parameters have a significant effect on system miss rates, as explained next.

Increasing the cache line size results in very effective prefetching and reduces the number of misses for most applications even when the cache line size is increased up to 128 entries. However, when increasing the cache line size from 16 to 64 and then to 128 entries, Ocean exhibits a large number of conflicts with non-cold misses more than doubling when the cache size is less than 32K entries.

Cache size has a small effect on the number of misses at small line sizes, but the importance of the cache size increases for larger line sizes. Doubling the cache size from 16 to 32K entries more than halves the total number of misses for FFT and LUContiguous in the L128 configurations (Figure 5).

Finally, cache associativity is important mostly at smaller cache and line sizes. Increasing associativity from 2 to 4 has a positive effect on most applications. Increasing the associativity from 4 to 8 seems to have a smaller effect. Furthermore, since the VMH lookup cache is implemented in software, higher associativities result in higher lookup times, e.g. for Radix C8.L128, Ocean C32.L128, and WaterSpatial C8.L64, and C8.L128 (Figure 5). For these reasons, an associativity of 4 seems to be the best compromise between reducing the number of misses and performance impact.

We also look at the impact of cache configuration on parallel speedups (Figure 6). There is a close correlation between the VMH lookup cache miss rates and parallel speedup, even for the configurations with small miss ratio. Generally we can divide applications in two categories. In the first category belong WaterNsquared, WaterSpatial, Volrend, Raytrace, and Ocean that, except for few configurations, are not very sensitive to the cache configuration due to the small miss rates they exhibit. Overall, for these applications, a small VMH lookup cache of 8K entries results in similar performance to the static system configuration, where the full VMH table is maintained on NIC memory. The second category includes FFT, Radix, and LUContiguous that show significant variations in speedups with cache configurations. For these applications the cache line size is the most important parameter, resulting in more than 100% in speedup variation.

Figure 7 shows the normalized execution time of each application in the original VMMC system versus their best performance with dynamic handle lookup. We observe that except for Radix, the overhead of dynamic handle lookup is negligible. For some applications the new system performs slightly better. We have not investigated this effect completely yet, but we believe that the changes in the common path due to the imported CBH table elimination result in somewhat lower system overheads for cases that do not exhibit many VMH and CBH misses.

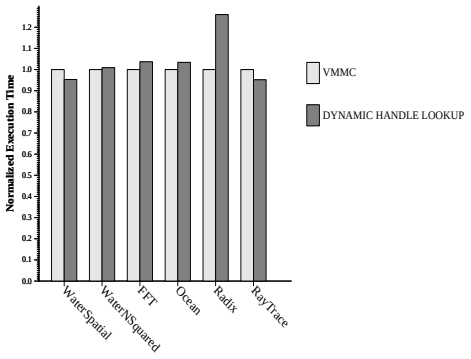

Fig. 7. Normalized execution time of each application in the original VMMC system versus dynamic handle lookup for the best VMH and CBH cache configurations.

We observe that a configuration of (C16, L64, A4) results in the best tradeoff between performance and NIC memory requirements. This configuration uses about 75 KBytes of NIC memory that is a substantial reduction from the original configuration, which uses more than 256 KBytes for the full VMH lookup table. Moreover, the current configuration does not have a limit on the amount of host memory it can support, although performance tradeoffs may change as application working set sizes increase.

Another important metric in the system is the miss penalty, defined as the time between the occurrence of the miss in the NIC and the time when the missed entries are brought to the cache on NIC memory. We divide this time into two parts: (i) *miss delay time* is the time between the occurrence of a miss interrupt up to the point when the miss handler in the driver is scheduled by the OS, and (ii) *miss processing time* is the time to service a miss in the device driver and to update the lookup cache. While both of them depend on various system parameters, the miss delay time is more unpredictable since it depends on the OS scheduling delay.

Our results show that the distribution of the miss delay time and the miss processing time are independent from the lookup cache configuration. However, they both vary across different applications.

Overall, the miss processing time incurs high variations, within the 5-500μs range. This is justified by the fact that servicing a miss includes expensive OS memory management functions and the fact that the miss handler has to compete with the application threads for CPU cycles. The delay miss time incurs lower variations and is usually within the 0-50μs range. Misses outside this range are due to the fact that we use a kernel thread to service the miss requests, that needs to be scheduled by the OS with no guarantee on the scheduling delay.

VI. Conclusions

To improve communication performance modern interconnects used in scalable servers provide user applications with

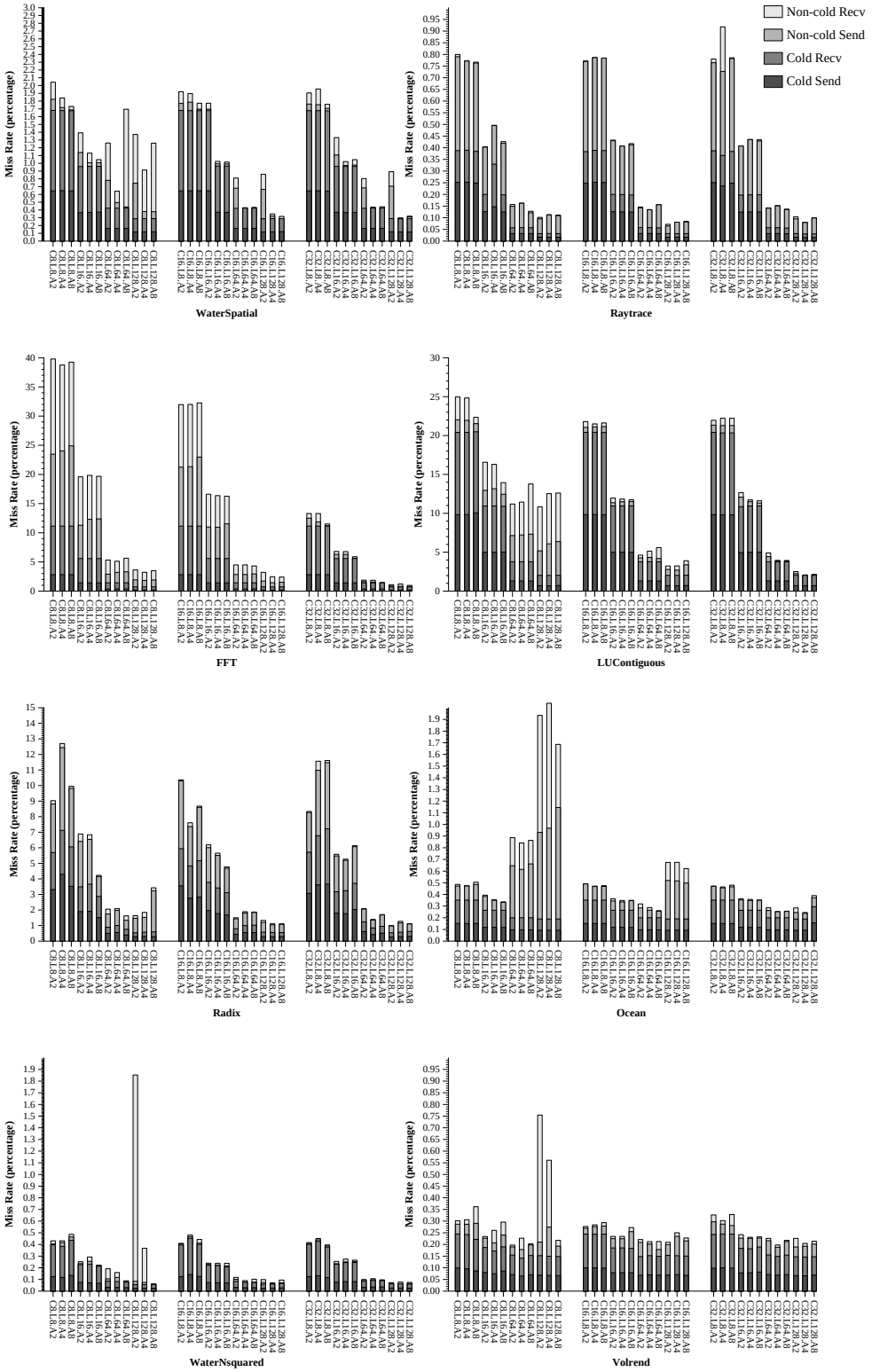

Fig. 5. VMH lookup cache miss breakdown of the SPLASH-2 applications.

the ability to directly transfer data between application virtual address spaces in different nodes. To support such data transfer operations, NICs need to translate between virtual and physical addresses as well as between other types of user and system handles. In most current user–level communication systems the mechanisms used for address translation are static. With

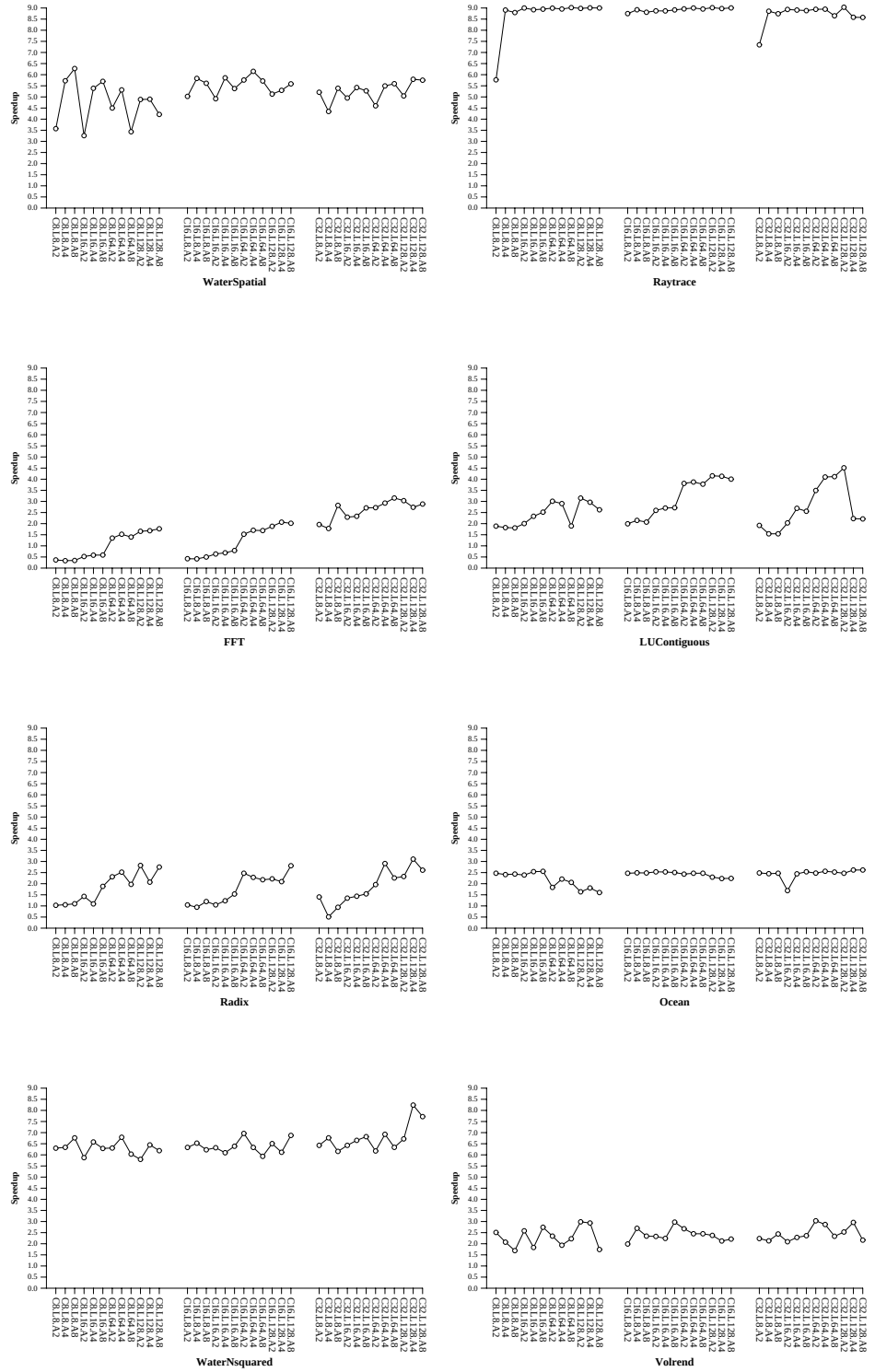

Fig. 6. Parallel speedup for the SPLASH-2 applications in different VMH lookup cache configurations (8 processors used).

increasing host memory sizes, this approach requires large on-NIC memory to maintain various mappings and, more importantly, it imposes limitations on the amount of host memory that can be used for communication buffers.

Dynamic handle lookup [11] deals with these shortcomings by extending a user–level communication system to use the NIC memory only as a cache for the larger host memory. In this work we evaluate the impact of dynamic handle lookup on system performance with both micro-benchmarks and real applications.

Dynamic handle lookup on the NIC results in significant reductions in memory requirements with a small impact on

system performance for the configurations we examine. Total NIC memory requirements are reduced by about 50%, whereas memory requirements for lookup operations are reduced by as much as 75% in our prototype, with negligible performance penalty. The overhead of using more complex dynamic lookup structures relative to the static data structures is at most 3% across all but one applications we examine. In larger system configurations, memory savings can be even more significant. Furthermore, our extensions eliminate any NIC-imposed restrictions on how much host memory can be used for communication buffers, an important problem in scalable storage, database, and application servers.

We find that the communication working sets of many realistic applications are in many cases small and that small on-NIC lookup caches have almost no impact on performance. This indicates that for many classes of applications large-memory NICs are not necessary. However, there are applications where larger caches are required. Moreover, in most cases the cache configuration parameters have a large impact both on the number of cache misses and the overall system performance. We find that there is considerable spatial locality in the communication access pattern of most applications and prefetching with large cache lines is effective and overshadows the increased miss penalty costs, due to the larger cache line size. Finally, in our system design we avoid unnecessary complexity. This makes our approach appropriate for incorporating in NICs that use general purpose processors with firmware as well as more aggressive, hardware implementations.

Overall, our results show that techniques similar to dynamic handle lookup can be very useful in building the next-generation NICs for modern application and storage servers, where application requirements on communication buffers exceed by far the static capabilities of today's NICs as well as smaller system configurations where NIC cost is an important factor.

VII. Acknowledgments

We would like to thank the members of the ATHLOS project at the University of Toronto for the useful discussions during the course of this work. We thankfully acknowledge the support of Natural Sciences and Engineering Research Council of Canada, Canada Foundation for Innovation, Ontario Innovation Trust, the Nortel Institute of Technology, Communications and Information Technology Ontario, and Nortel Networks.

References

[1] M. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. Felten, and J. Sandberg, "A virtual memory mapped network interface for the shrimp multicomputer," in *Proc. of the 21st International Symposium on Computer Architecture (ISCA21)*, Apr. 1994, pp. 142–153.

[2] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. Su, "Myrinet: A gigabit-per-second local area network," *IEEE Micro*, vol. 15, no. 1, pp. 29–36, Feb. 1995.

[3] Giganet, "Giganet cLAN family of products," http://www.emulex.com/products.html, 2001.

[4] D. Dunning and G. Regnier, "The Virtual Interface Architecture," in *Proc. of The 1997 IEEE Symposium on High Performance Interconnects (HOT Interconnects V)*, Aug. 1997.

[5] InfiniBand Trade Association, "Infiniband architecture specification, ver. 1.0," http://www.infinibandta.org, Oct. 2000.

[6] Internet Engineering Task Force (IETF), "iSCSI, version 08," in *IP Storage (IPS), Internet Draft, Document: draft-ietf-ips-iscsi-08.txt*, Sept. 2001.

[7] P. Jamieson and A. Bilas, "Cables: Thread control and memory management extensions for shared virtual memory clusters," in *Proc. of The 8th IEEE Symposium on High-Performance Computer Architecture (HPCA8)*, Feb. 2002.

[8] M. Inc., "Myrinet product list and prices," http://www.myri.com/myrinet/product_list.html, 2002.

[9] Emulex, "Gn9000/vi: 1gb/s vi/ip pci host bus adapter," http://www.emulex.com/products/viip/index.html.

[10] Y. Zhou, A. Bilas, S. Jagannathan, C. Dubnicki, J. Philbin, and K. Li, "Experiences with vi communication for database storage," in *Proc. of the 29th International Symposium on Computer Architecture (ISCA29)*, May 2002.

[11] R. Azimi and A. Bilas, "Design and implementation of dynamic handle lookup in modern network interfaces," in *To appear in the Proc. of The Workshop on Communication Architecture for Clusters (CAC'03)*, Apr. 2003.

[12] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. of the 22nd International Symposium on Computer Architecture (ISCA22)*, Santa Margherita Ligure, Italy, June 1995, pp. 24–36.

[13] R. Gillett, M. Collins, and D. Pimm, "Overview of network memory channel for PCI," in *Proc. of the IEEE Spring COMPCON '96*, Feb. 1996.

[14] C. Dubnicki, A. Bilas, Y. Chen, S. Damianakis, and K. Li, "VMMC-2: efficient support for reliable, connection-oriented communication," in *Proc. of The 1997 IEEE Symposium on High Performance Interconnects (HOT Interconnects V)*, Aug. 1997, a short version of this appears in IEEE Micro, Jan/Feb, 1998.

[15] J. Tang and A. Bilas, "Tolerating network failures in system area networks," in *Proc. of the 2002 International Conference on Parallel Processing (ICPP02)*, Aug. 2002.

[16] Y. Chen, A. Bilas, S. N. Damianakis, C. Dubnicki, and K. Li, "UTLB: A mechanism for address translation on network interfaces," in *Proc. of The 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS8)*, San Jose, CA, Oct. 1998, pp. 193–203.

[17] A. M. Mainwaring and D. E. Culler, "Design challenges of virtual networks: Fast, general-purpose communication," in *Proc. of The 1999 ACM Symposium on Principles and Practice of Parallel Programming (PPoPP99)*, May 1999, pp. 119–130.

[18] A. Basu, M. Welsh, and T. von Eicken, "Incorporating memory management into user-level network interfaces," http://www2.cs.cornell.edu/U-Net/papers/unetmm.pdf, 1996.

[19] A. Basu, V. Buch, W. Vogels, and T. von Eicken, "U-net: A user-level network interface for parallel and distributed computing," *Proc. of the Fifteenth Symposium on Operating Systems Principles (SOSP15)*, December 1995.

[20] P. Buonadonnaa, A. Geweke, and D. Culler, "An implementation and alaysis of the virtual interface architeture," in *Supercomputing (SC)*, 1998, pp. 7–13.

[21] National Energy Research Scientific Computing Center, "M–via: A high performance modular via for linux," http://www.nersc.gov/research/FTG/via, 1998.

[22] M. Banikazemi, B. Abali, and D. Panda, "Comparison and evaluation of design choices for implementing the virtual interface architecture (via)," in *Workshop on Communication and Architectural Support for Netwok-based Parallel Computing CAPCN-HPCA*, 2000.

[23] I. Schoinas and M. D. Hill, "Address translation mechanisms in network interfaces," in *Proc. of The 4th IEEE Symposium on High-Performance Computer Architecture (HPCA4)*, 1998.

[24] A. Bilas, C. Liao, and J. P. Singh, "Using network interface support to avoid asynchronous protocol processing in shared virtual memory systems," in *Proc. of the 26th International Symposium on Computer Architecture (ISCA26)*, May 1999.

[25] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1990.