UTLB: A Mechanism for Address Translation on Network Interfaces

Yuqun Chen, Angelos Bilas, Stefanos N. Damianakis, Cezary Dubnicki, and Kai Li
Department of Computer Science, Princeton University, Princeton, NJ 08544
{yuqun, bilas, snd, dubnicki, li}@cs.princeton.edu

Abstract

An important aspect of a high-speed network system is the ability to transfer data directly between the network interface and application buffers. Such a *direct data path* requires the network interface to "know" the virtual-to-physical address translation of a user buffer, *i.e.*, the physical memory location of the buffer. This paper presents an efficient address translation architecture, User-managed TLB (UTLB), which eliminates system calls and device interrupts from the common communication path. UTLB also supports application-specific policies to pin and unpin application memory. We report micro-benchmark results for an implementation on Myrinet PC clusters. A trace-driven analysis is used to compare the UTLB approach with the interrupt-based approach. It is also used to study the effects of UTLB cache size, associativity, and prefetching. Our results show that the UTLB approach delivers robust performance with relatively small translation cache sizes.

1 Introduction

A computer cluster consists of multiple hosts connected with a high speed network. The goal of a cluster is to exploit the aggregate computing power of many microprocessors and the throughput of multiple I/O buses. The key to achieving high performance in a cluster is an efficient communication subsystem. The communication subsystem includes the network interface and the software that enables applications to access it. A major responsibility of the communication software is to facilitate efficient data transfer between the network interface and applications.

There are two kinds of data paths between the network interface and application buffers: indirect and direct. The *indirect* path involves copying data between an application buffer and a dedicated system buffer. This path is straightforward to implement but incurs significant overhead due to data copying. In contrast, the *direct* data path allows data transfers between the network interface and application buffers which are usually accomplished by programmed I/O or DMA. The direct path not only eliminates the copy overhead, but it also reduces involvement of the host processor. User-level protocols that take advantage of the direct path can increase the end-to-end communication bandwidth by as much as 100% [13].

User-level communication allows an application to issue communication requests directly to the network interface, bypassing the operating system (OS). It eliminates OS calls, and sometimes interrupts, from the common communication path. Fast user-level communication relies on the direct path to avoid copying data to and from a dedicated system buffer [46, 2, 35, 45, 7, 33].

Combining user-level communication and direct path communication introduces an address translation problem. Address translation is necessary because the application process uses virtual memory whereas the network interface accesses physical memory. The virtual-to-physical mappings are kept in the operating system and are inaccessible to applications running at the user level.

Furthermore, the communication subsystem must guarantee that the application buffer remains resident in physical memory until the data transfer is complete. As an I/O device, the network interface has no control over paging and swapping in the operating system. Therefore, the application buffer must be explicitly pinned in physical memory before data transfer can take place.

Existing approaches to communication-related address translation do not provide a good solution for user-level direct-path communication. Some approaches require system calls to initiate communication in kernel mode. Some restrict where in the application's virtual address space data communication takes place, thus making it difficult to achieve zero copy.

This paper presents an address-translation mechanism, called User-managed TLB (UTLB), for network interfaces that use the direct path to transfer data. By exploiting the spatial locality seen in most applications, UTLB eliminates system calls and device interrupts from the common case. The UTLB only invokes the operating system for *pinning* the user buffer when it is first used in communication. The unique translation table in the UTLB allows the application to specify a data buffer with virtual addresses, each of which the network interface can *safely* translate into a physical address with just a table lookup. The UTLB mechanism does not rely on OS modifications nor on esoteric OS features. Only a device driver that accesses the OS page-pinning and unpinning facility is required. Therefore, UTLB mechanism is portable across a wide variety of OS platforms and network interfaces.

We implemented the UTLB mechanism on Myrinet-based [5] PC clusters with both Linux and NT operating systems. Our implementations show that the translation lookup costs 0.5 μs in the best case. We conducted trace-driven simulations to compare the UTLB mechanism with an approach that handles page pinning and address translation using host interrupts. The simulations were also used to determine the effects of different UTLB cache parameters and benefit of prefetching translation entries.

2 Related Work

There is a large body of literature on communication subsystems. Most related work is on how and where address translation is performed in a communication subsystem. The four key issues are as follows. First, how to initiate data transfer requests from an application. Second, how to maintain consistency between the translations on the host and those on the network interface. Third, how to replace translation entries on the network interface. And fourth, how to deal with protection in a multiprogramming environment.

Early implementations of communication subsystems typically used dedicated, pinned, system-wide send and receive buffers. Each buffer is laid out in contiguous physical memory so that the network interface needs to know the starting physical address and the number of bytes for a data transfer. When sending a message, the application process traps into the OS to initiate the send. The OS copies the application data into the system send buffer. The network interface then transmits the message from the system buffer. Upon message arrival, the network interface DMAs the data into the system buffer. From there, the OS copies the data into an application process' buffer. Address translation is carried out in two places: one, in the kernel when copying the data to and from the application buffer, and two, on the network interface when accessing the kernel buffer.

A method to lift the constraint of using a large contiguous piece of physical memory for a system buffer is to use a table or a chain of descriptors. Each descriptor contains the address translation for a small contiguous piece of the kernel buffer. The descriptor table or list is stored in a linked list on the network interface. This approach is also called scatter/gather table. It allows the OS to initiate network data transfers at any place in physical memory. Furthermore, to avoid copying data to and from an application buffer, the OS can pin the buffer and store its address translations in the descriptor table or list. System calls are required to build the descriptors inside the OS. Autonet [41], for example, uses a chained list of descriptors and VAXClusters [28] uses a descriptor table.

Page remapping is a method to avoid copying [30, 10, 15, 27, 6]. When transfers are properly aligned and the "right" length (i.e. a multiple of the physical page size), the OS swaps the virtual-to-physical mappings between the pages of the kernel buffer with those of the application buffer. This technique can achieve *zero copy* with buffer restrictions. Furthermore, page remapping incurs significant overhead due to OS involvement, interrupt processing, and context switching.

Some systems use a communication processor that runs as a part of the operating system to either access or cache the OS page table in order to translate the application buffer's virtual addresses and initiate DMA transactions on the network interface. The Intel Paragon [36] dedicates an SMP microprocessor on a cache-coherent memory bus for this purpose. The Intel Paragon communication processor also has the ability to control page pinning and swapping, which eliminates the need for system calls and interrupts at the expense of taking a microprocessor away from doing useful computation.

Another approach is to use a protocol processor to deal with address translation and data transfer. Meiko CS-2 [24] and Typhoon [38] use a protocol processor. Stanford FLASH multiprocessor [29] uses a programmable processor to integrate a memory controller, an I/O controller, a network interface, and a programmable protocol processor.

Several communication subsystems transfer data directly between application buffers and network interface [12, 43, 35]. With this approach, the application is responsible for translating addresses or performing programmed I/O operations to access the network. But this approach is not designed for multiprogramming environments.

An improvement to this approach is to virtualize the network interface. The basic method is to provide a virtual communication port abstraction through which an application process can directly issue requests to the network interface, bypassing the OS. Examples of such systems include Application Device Channels (ADC) [14], Hamlyn [7], U-Net [45], and Virtual Interface Architecture [11]. They all require that applications explicitly pin buffers and install descriptors on the network interface. The descriptors contain address translations for both send and receive sides. Hamlyn calls such a unit a *slot*, U-Net calls it a *communication segment*, and VIA calls it a *memory region*. They typically deal with protection by using a permission key.

Memory-mapped communication takes a direct approach. PRAM [32], SHRIMP [2] and Memory Channel [20] implement memory-mapped communication models [44] that allow applications to send messages to remote memory. PRAM implements a physical memory-mapped model that allows an application to map network interface DRAM into its address space. Writes to this memory are propagated to remote network interface memory. It is the application's responsibility to move data from a send buffer to the sender's network interface memory and move data from the receiver's network interface memory into a receive buffer. The SHRIMP approach [2, 3, 4] implements protected user-level communication using the Virtual Memory-Mapped Communication (VMMC) model. This approach allows an application to send data directly from its virtual memory to a remote process' virtual memory in a multiprogramming environment. This approach requires receivers to pin and export receive buffers before the data is transfered. The OS translates the virtual addresses and stores the physical addresses on the network interface. The SHRIMP implementation uses a modified OS to automatically pin application buffers used for sending data. A User-level DMA (UDMA) mechanism [3] is used to allow the network interface hardware to obtain virtual-to-physical translations without OS intervention. SHRIMP also provides *automatic update* which automatically propagates application buffer updates to remote virtual memory buffers. Digital's Memory Channel [20] uses an approach that is similar to PRAM on the sending side and to SHRIMP's automatic update on the receiving side.

Another direct approach allows applications to compose and retrieve messages using network interface registers [22, 42]. In addition to network interface registers, the Cray T3E [42] supports remote memory accesses with an approach similar to UDMA. It uses complete page tables to describe global communication segments and all communication pages are pinned in memory.

A network interface typically can hold only a limited number of translation entries, which poses questions about how to maintain consistency between the translations on the host and those on the network interface and how to deal with misses on the network interface. One approach is to let network interface interrupt the host processor on a translation entry miss, and the host processor installs the translation entry on the network interface. The VMMC [16] (the same communication API as that on SHRIMP [2]) for the Myrinet PC cluster employs this approach. It uses a per-process translation table on the network interface.

UNet-MM [1], an extension U-Net, stores address translations in a translation cache on the network interface. Misses in the translation cache are handled by the host OS which pins virtual pages and installs their translations on the network interface.

The UTLB approach described in this paper was presented in Hot Interconnect '97 [18] and in a project update note [19]. A recent paper by Schoinas and Hill [40] describes a similar approach. None of these papers deal with the issues of a shared translation cache in a multiprogramming environment. Further, they do not study translation replacement, nor the effects of prefetching translation entries.

3 Design of UTLB

The User-managed TLB (UTLB) is an address translation mechanism for user-level communication. The main ideas in UTLB are demand-driven page-pinning, protected translation table, and user-level lookup.

The first idea is *demand-driven page-pinning*: pin the local buffer when it is used in communication for the first time, at the same time supplying the address translations to the network interface. The buffer remains pinned in physical memory so that subsequent data transfers using this buffer can be initiated directly at the user level. For applications that display spatial locality in their communication patterns, the cost to pin the virtual pages is amortized over multiple communication requests.

The second idea is to establish a *protected* translation table for pinned virtual pages. UTLB allocates a translation table for each process on the network interface. A translation table contains physical addresses for a process' virtual pages that have been pinned in physical memory. The translation table is invisible to the user process. However, the user process can specify to the operating system where in the table to store the physical translations for a given virtual buffer, hence the name "User-managed" TLB. To transfer data on a virtual page, the user process specifies to the network interface the index in the translation table where the page's physical address stored. Using this index, the network interface reads the physical address directly from the translation table.

The third idea is to construct a fast *user-level lookup* data structure. The user process has to keep track of the mapping between the translation table indices and the pinned virtual pages. The lookup table uses a standard two-level page table architecture [21, 26]. It contains one entry for each virtual page. An entry can either be invalid or contain the index in the translation table where the physical address for this virtual page is stored. Only two memory references are required to obtain the UTLB index for a given virtual page address.

3.1 Per-process UTLB

Combining the above three ideas results in a *Per-process* UTLB shown in Figure 1. The communication subsystem allocates a fixed-sized translation table for each process. The translation tables are allocated directly in the network interface memory. They are protected from user processes. The user process asks the OS to pin certain virtual pages and "install" their physical translations at specified locations in

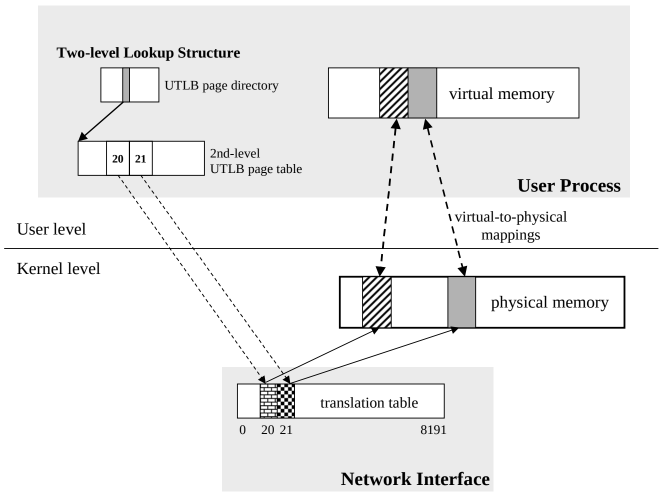


Figure 1: Structure of UTLB on VMMC

its translation table. This can be done with a device driver call. A user-level library maintains the mapping between the translation table indices and the virtual page addresses in a two-level lookup tree.

It is possible that the translation table is filled up while more virtual pages need to be pinned. The user-level library can detect such *capacity misses* and *evict* some translations already in the UTLB translation table. Eviction of an entry results in unpinning of the virtual pages. The UTLB library decides which translation table entries to evict and asks the OS to unpin corresponding virtual pages and invalidate the entries. To reduce the frequency of capacity misses, the user-level library monitors the virtual page usage and use a replacement policy such as *LRU* to select the victim entries for eviction.

To ensure correctness, the user-level library must only select virtual pages that will not be involved in any oustanding send requests. Otherwise, the network interface must be able to check for possible unpinned pages, and interrupt the host to pin pages before executing the requests.

3.2 Shared UTLB-Cache

A drawback of the per-process UTLB is that it statically allocates translation tables from the network interface memory. This results in a fairly small translation table for each process. On a workstation with large physical memory, a significant portion of a user process' virtual address space can be involved in data communication. The size of the UTLB translation table must be increased to reduce capacity misses.

To overcome the size limitation of the per-process UTLB translation table, we place a Shared UTLB-Cache on the network interface and move the entire translation tables to host physical memory (DRAM), as shown in Figure 3. In this scheme, the Shared UTLB-Cache caches the entries from the translation tables. Each Shared UTLB-Cache entry contains the process ID and part of the translation table index to uniquely identify an entry from a particular translation table. When a miss occurs in the Shared UTLB-Cache, the network interface simply reads the entry from the translation table in physical memory. The cost of reading an entry

```
User Program:
send_message(vaddr, nbytes)
    1) look up (vaddr, npages) in the user-level
       table
    2) if (some pages in the range are not pinned)
           find free UTLB translation entries
           invoke the ioctl call to
               a) lock the pages
               b) fill the translation entries
    3) submit the request to NI using the indices


Network Interface:
1) receive user request
2) obtain physical addresses by directly
   indexing the translation table
3) perform DMA
```

Figure 2: Pseudo-code that illustrate the steps taken by the user process to send data from its virtual buffer

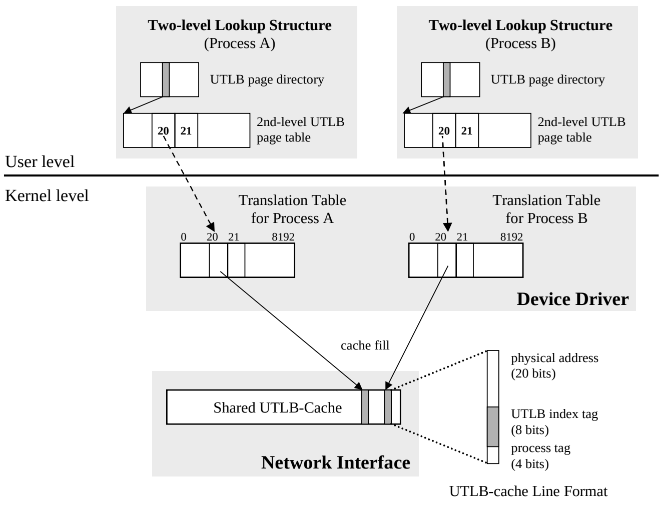


Figure 3: Structure of Shared UTLB-Cache

over the I/O bus is only a couple of microseconds. Therefore, in the worst case, when every translation lookup misses in the Shared UTLB-Cache, the lookup time is a few microseconds. This is typically better than interrupting the host OS and letting the host install the required translation on the network interface on every translation miss.

A miss in the Shared UTLB-Cache requires that the network interface read translation entries over I/O bus. The miss penalty is therefore several times the hit cost which is simply a memory reference on the network interface. A miss in the Shared UTLB-Cache will have a perceivable impact on small-message latency.

We apply existing techniques in processor cache design [21, 37] to reduce the miss rates in the Shared UTLB-Cache. Misses fall into three categories: capacity misses, conflict misses, and compulsory misses [23]. When only one process is using the network interface, both capacity misses and conflict misses may occur in the Shared UTLB-Cache. Multi-programming may further increase conflict misses.

Capacity misses can be reduced by enlarging the size of Shared UTLB-Cache. Conflict misses can be reduced by making the cache set-associative. Set-associativity can also help reduce conflict misses that are caused by multi-programming. A simple scheme to reduce the conflict misses is to *offset* a translation table index by a process-dependent constant. The same index from different translation tables will be hashed into different locations in the Shared UTLB-Cache. Prefetching translation entries in the Shared UTLB-Cache reduces the miss rates when applications display spatial locality. But, it also incurs additional cost for fetching more entries. In Section 6, we evaluate the effect of cache size, associativity, and prefetching.

3.3 Hierarchical-UTLB

The Hierarchical-UTLB is a simplification of the UTLB. Instead of letting the user process search its lookup data structure for UTLB indicies, Hierarchical-UTLB directly uses the protected translation table as the lookup data structure. Under Hierarchical-UTLB, the user process submits virtual addresses to the network interface. The network interface directly searches Hierarchical-UTLB translation table for physical address translation (Figure 4).

The translation table in Hierarchical-UTLB resembles a typical two-level page table structure. The first-level directory points to second-level page tables in physical memory. Each page table entry stores the physical address of a pinned virtual page. The Hierarchical-UTLB translation table differs from a real page table in one important aspect: the entries in the second-level Hierarchical-UTLB translation table are the physical addresses of virtual pages that have been explicitly pinned by the user process.

The top-level directory of a Hierarchical-UTLB translation table is always stored in the network interface so that when there is a miss in the Shared UTLB-Cache it takes one memory reference in the SRAM to access the page directory and one DMA to access the second-level page table.

The user-level library only needs a bit array to maintain the memory-pinning status of virtual pages. In addition, a process can directly use the virtual address to represent its buffers. Instead of indices, the network interface simply uses the virtual address to look up the physical address in the Shared UTLB-Cache or query the UTLB page table on a miss.

The Hierarchical-UTLB eliminates the need to handle UTLB fragmentation: after complex data accesses, a user buffer's translations may be scattered in the translation table. Integrating Hierarchical-UTLB with the translation table used for *receive buffers* is also straightforward. Virtual addresses are uniformly used to represent both remote and local buffers.

In rare situations, the second-level translation tables in the Hierarchical-UTLB occupy too much physical memory A solution to this problem is to manage the second-level translation tables in the same manner as virtual memory paging. One bit of information is added to each entry in the top-level directory which indicates whether the second-level table is in physical memory or on the disk. If the second-level table is swapped out, the directory entry contains the disk block number instead of the physical address of the second-level table. When the network interface detects that a page of the second-level table has been swapped out, it can interrupt the host OS to bring in the page.

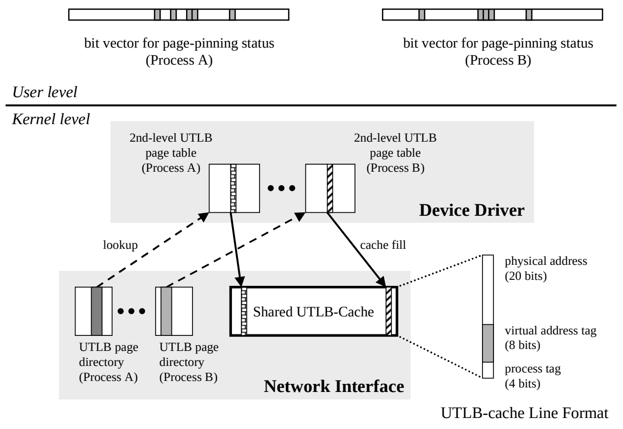

bit vector for page-pinning status (Process A)
bit vector for page-pinning status (Process B)
User level
Kernel level
2nd-level UTLB page table (Process A)
2nd-level UTLB page table (Process B)
Device Driver
lookup
cache fill
Shared UTLB-Cache
UTLB page directory (Process A)
UTLB page directory (Process B)
Network Interface
physical address (20 bits)
virtual address tag (8 bits)
process tag (4 bits)
UTLB-cache Line Format

Figure 4: Structure of Hierarchical-UTLB

To summarize, at the cost of one extra SRAM reference to process a UTLB cache miss, Hierarchical-UTLB simplifies the construction of UTLB and eliminates the fragmentation problem. In the rest of the paper, UTLB refers to Hierarchical-UTLB.

3.4 User-level replacement policies

An important feature of UTLB is that it allows an application to decide which virtual pages to unpin when the system runs out of available physical memory. Because the application process often has knowledge about its virtual memory access, it can use a custom replacement policy to minimize the number of page pinning and unpinning operations. UTLB predefines five replacement policies for applications to choose: LRU, MRU, LFU, MFU, and RANDOM.

An important issue related to the replacement policies is how to manage the amount of physical memory that a user process can pin. This is a complex issue especially in a multi-programming environment where physical memory pages can be shared. Enforcing a static limit on the number of pages a process can pin is straightforward, But, implementing a dynamic limit requires that the OS synchronize with the user-level UTLB data structures when reclaiming pinned physical pages.

The combination of the two issues is similar to the application-controlled file caching problem [8, 9], where multiple applications share a set of file cache blocks in the kernel and each application can choose its own replacement policy. So, related theoretical results of the application-controlled file caching problem [8] apply to the application-controlled memory pinning/unpinning problem. On the other hand, this requires experimental studies to understand the solutions to the combined problem.

4 An Implementation of UTLB

We developed an implementation of the Hierarchical-UTLB for our custom protected user-level communication model called Virtual Memory-Mapped Communication (VMMC).

4.1 VMMC

VMMC is a communication model that provides protected direct data transfer between the virtual address spaces of two applications. Two kinds of user buffers are used for data transfer: the *receive buffer* and the *send buffer*. Both buffers reside in the virtual address space of an application. The receive buffer is made visible to applications on remote hosts through an *export* system call. An application gains access rights to an exported receive buffer by *importing* it. The basic VMMC model supports *remote store*, which allows an application to send data from a local buffer directly into another application's receive buffer via the network (Figure 5).

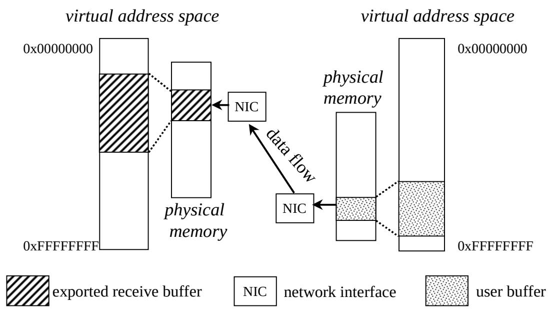


Figure 5: The VMMC Communication Model

VMMC was first implemented on the SHRIMP multicomputer [2] and then ported to the Myrinet network interface [17]. We later extended the VMMC model with three new features [18]:

- *Remote-fetch* allows an application to *fetch* data from a receive buffer into a local buffer.
- *Transfer-redirection* "redirects" incoming data from its default location to another user buffer specified by the application. This enables zero-copy implementations of high-level communication APIs [13].
- *Reliable communication* that implements a retransmission protocol at data link level (between network interfaces) and a dynamic node remapping procedure to deal with link and port failures.

The UTLB mechanism empowers the first two features.

4.2 Implementation details

Our UTLB implementation is a part of the VMMC communication mechanism for Myrinet PC clusters. Myrinet [5] is a switched point-to-point network capable of transfering data at 160 MB/sec on each link. The Myrinet PCI network interface has a 33 MHz RISC microprocessor (LANai 4.2) and 1 MB Static RAM (SRAM). A cluster of 300 MHz Pentium-II PC workstations are connected to the Myrinet network. The host operating system is Windows NT 4.0.

The communication subsystem of VMMC consists of three components: the VMMC Myrinet firmware, the device driver, and the user-level library [16] (see Figure 6). The VMMC device driver initializes the network interface and downloads the firmware, called the Myrinet Control Program (MCP), into the network interface SRAM. The driver also allocates a special command post buffer from the Myrinet SRAM and maps it into the application's address space. The user-level VMMC library posts communication requests to the command buffer. The address of a command buffer is used to identify the user process. The MCP polls user requests from each command buffer and process them in the order that they are received.

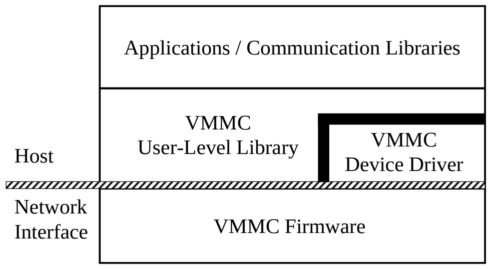


Figure 6: The VMMC System Architecture

Our implementation of Hierarchical-UTLB follows directly from the layout in Section 3.3; we implemented a Shared UTLB-Cache. The size of Shared UTLB-Cache that we chose is 32 KB (or 8 K entries). The device driver allocates the two-level translation table dynamically for each application that uses the VMMC system. An `ioctl()` call is added to the VMMC device driver for pinning virtual pages and storing physical addresses in the translation table. The implementation did not require any modifications to the Windows NT operating system. An earlier implementation of UTLB on Linux was also done without OS modifications. The device driver allocates and pins a "garbage" page. All UTLB translation table entries are initialized with the physical address of the garbage page. This scheme saves the network interface from checking the validity of user-submitted indices. At worst, the network interface transfers data to and from an unused garbage page; no harm is done to the system or other applications.

5 Performance of UTLB

The overall cost for translating a virtual page in the UTLB includes the overhead on the host processor and the overhead on the network interface. Each overhead varies depending on whether the virtual page is pinned and whether the physical address is in the UTLB network interface cache. The fastest path to translate a virtual address on the network interface is taken when the virtual page is pinned and its physical address is present in the UTLB network interface cache (a hit). The total overhead for this path is only 0.9 μs (0.4 μs on the host and 0.5 μs on the network interface).

The time on the network interface is measured with LANai's real-time clock register, with an accuracy of 0.5 μs. Because the LANai 4.x processor has *no* instruction or data caches, averaging the total time of repeated operations gives the exact timing for an individual operation. On the host, the time is measured with the Pentium processor's cycle counter with an accuracy of a CPU clock cycle. Reading the cycle counter has an overhead of 39 cycles.

num_pages	1	2	4	8	16	32
check min	0.2	0.2	0.2	0.2	0.2	0.2
(μs) max	0.4	0.6	0.6	0.6	0.6	0.7
pin (μs)	27	30	36	47	70	115
unpin (μs)	25	30	36	50	80	139

Table 1: UTLB overhead on the host processor.

5.1 Host-side performance

Table 1 lists the overhead of UTLB host-side operations: user-level lookup (as *check*), page-pinning (as *pin*), and page-unpinning (as *unpin*). The lookup procedure checks a bit map to see if the virtual pages of a buffer are already pinned in the physical memory. The cost of checking the bit map varies with the first bit's position in the bit map. The table reports both the minimum and maximum costs from all posible bit positions.

If some virtual pages are not pinned, the lookup procedure invokes a device driver `ioctl()` call to pin these pages and store their physical addresses in the UTLB translation table. As the numbers suggest, page-pinning is an expensive operation. They validate the UTLB design strategy of on-demand pinning and translation caching.

5.2 Network interface performance

Table 2 shows the cost of UTLB operations on the network interface: hit cost, DMA cost and miss handling cost (as *miss cost*). The number given here is for a direct-mapped cache with 8K entries. The hit cost is the time it takes the network interface to look up a virtual page's physical address in the UTLB network interface cache [1].

num_entries	1	2	4	8	16	32
DMA cost (μs)	1.5	1.6	1.6	1.9	2.1	2.5
total miss cost (μs)	1.8	1.9	1.9	2.3	2.8	3.2

Table 2: UTLB overhead on the network interface (The hit cost is a constant 0.8 μs.)

The miss cost includes the time to obtain the physical address for the second-level page table (see Section 3.3) and the time to DMA the entries into the UTLB network interface cache. Multiple entries are prefetched at once during a miss to exploit the spatial locality of a program's data access. The prefetching cost remains relatively constant with respect to the number of entries fetched because DMA setup dominates the total fetch time for a small number of words (see Table 2).

6 Application-driven Analysis of UTLB

We would like to answer three questions:

- How does the UTLB compare with the approach where the network interface interrupts the host to handle translation misses?
- What are the appropriate values for the size and the associativity of the Shared UTLB-Cache?
- What are effects of prefetching translation entries?

We used a trace-driven simulation approach to evaluate these issues. A trace-driven approach allows us to determine the best values for parameters and to compare the UTLB with other address translation mechanisms. We chose communication traces from a Myrinet cluster of SMP workstations because they allow us to study the behavior of UTLB

[1] The Myrinet VMMC firmware breaks down data transfer at 4KB page boundaries. Translation lookups are performed one page at a time. Therefore, we list the hit cost for only one entry here.

under a high degree of multi-processing. The cluster consists of four 4-way Intel SMP workstations connected with a Myrinet network. Each SMP has four 200 MHz PentiumPro microprocessors and 256 MB of DRAM.

We ran a number of applications from the SPLASH2 [47] Application Suite with the Home-based Release Consistency SVM Protocol [48, 39]. On each SMP, there are four application processes and a protocol process, all of which use Myrinet for sending and receiving messages. We instrument the VMMC software to trace each send and remote read request along with a globally-synchronized clock [31]. Time stamps are used to serialize the traces from the five processes on each SMP. The traces are then fed to a UTLB simulator.

The simulator mimics the behavior of a network interface translation cache, the host-side UTLB driver, and user-level library. The simulator reads traces, serializes the communication requests using the time stamps in the trace, and derives detailed statistics on translation misses, and the number of page pinnings and unpinnings. The network interface translation cache is simulated with direct-mapping, 2-way, and 4-way associativity. The simulator implements an LRU replacement policy to manage pinned virtual pages given a fixed physical memory constraint. We also developed a simulator for the interrupt-based approach where the network interface interface interrupts its host CPU on a translation miss, and the CPU handles page pinning, unpinning, and installing new translation entries.

6.1 Applications

Seven applications from the SPLASH-2 suite are used:

- **Barnes** implements the original Barnes-Hut algorithm for NBody simulation. Each process gets a partition of the particles and calculates their new positions during one time step. Communication in this application is moderate as the particle partition exhibits spatial locality.

- **FFT** implements a parallel 2D Fast Fourier Transform algorithm. This program exhibits high degree of data communication.

- **LU** is a parallel LU matrix decomposition program.

- **Raytrace** uses a task-farm model to raytrace a scene. Communication in Raytrace revolves around the task queues.

- **Radix** sorts an array of integer keys in parallel. The algorithm consists of a number of radix-sort phases. During a phase, each process sorts a contiguous sequence of the keys according to part of the keys. At the end of the phase, the results from each processes are combined to form a new array for subsequent radix-sort phases.

- **Volrend** uses a task-farm model to render a 3-D volume. Communication in this application also centers on the task queues.

- **Water** calculates movements of molecules using a spatialized algorithm to exploit data locality.

Table 3 shows the application problem size, communication memory footprint, and translation lookup frequency. The communication memory footprint indicates the average number of distinct virtual pages used for communication on each node. The number of translation lookups is the average number of communication operations performed on each node.

Applications	Problem Size	Footprint (4 KB pages)	# translation lookups
FFT	4M elements	10,803	43,132
LU	4K x 4K matrix	12,507	25,198
Barnes	32K particles	2,235	35,904
Radix	4M keys	6,393	11,775
Raytrace	256 x 256 car	6,319	14,594
Volrend	256^3 CST head	2,371	9,438
Water-spatial	15,625 molecules	1,890	8,488

Table 3: Application problem size, communication memory footprint, communication translation lookup frequency.

6.2 Comparing UTLB with an interrupt-based mechanism

To compare the two approaches, we assume that the cache structures are the same for both cases. We varied a set of parameters to study the behavior of the UTLB mechanism and the interrupt-based approach. The parameters include the amount of physical memory available to each process, the size and the associativity of the network interface translation cache. For each application and a particular set of parameters, the UTLB simulator reports the number of user-level check misses, the number of network interface translation misses (caused by capacity or conflict), and the number of pages unpinned. The simulator for the interrupt-based approach reports only the number of network interface translation misses and the number of unpin operations.

Table 4 shows the number of check misses per lookup, the number of network interface translation misses, and the number of unpinned pages per lookup. All the numbers are averaged over the total number of lookups. The numbers for the interrupt-based approach are under the *Intr* label. Our trace-driven simulations show that UTLB requires fewer page pinning and unpinning operations than the interrupt-driven approach for all cache sizes. The cache simulated here is a direct-mapped cache with index offsetting (see Section 6.3). The host physical memory is unlimited, therefore, UTLB does not unpin application pages. However, the interrupt-based approach always unpins a page that is evicted from the network interface translation cache [1]. This is a major difference between UTLB and the interrupt-based approach.

Note that network interface misses are handled differently by the two approaches. With UTLB, a network interface interrupt moves the missed entry from host memory into the network interface translation table directly, whereas the interrupt-based approach has to interrupt the CPU for every translation miss in the network interface. On most computer systems, interrupts are an order of magnitude more expensive than memory references over the I/O bus. The UTLB mechanism can offer significantly faster miss handling than an interrupt-based approach. On the other hand, once in the interrupt handler, pin or unpin requires no protection domain crossing, whereas the UTLB approach requires paying the overhead of a system call.

The real cost of a translation lookup depends on the components shown in the table. In particular, the cost function

Cache Entries	Characteristic (per lookup)	Application													
		Barnes		FFT		LU		Radix		Raytrace		Volrend		Water	
		UTLB	Intr	UTLB	Intr	UTLB	Intr	UTLB	Intr	UTLB	Intr	UTLB	Intr	UTLB	Intr
1 K	check misses	0.04	-	0.25	-	0.49	-	0.54	-	0.43	-	0.25	-	0.10	-
	NI misses	0.10	0.10	0.50	0.50	0.50	0.50	0.62	0.62	0.48	0.48	0.31	0.31	0.35	0.35
	unpins	0.00	0.09	0.00	0.49	0.00	0.46	0.00	0.54	0.00	0.41	0.00	0.22	0.00	0.31
2 K	check misses	0.04	-	0.25	-	0.49	-	0.54	-	0.43	-	0.25	-	0.10	-
	NI misses	0.07	0.07	0.50	0.50	0.49	0.49	0.60	0.60	0.46	0.46	0.29	0.29	0.27	0.27
	unpins	0.00	0.04	0.00	0.48	0.00	0.43	0.00	0.44	0.00	0.33	0.00	0.13	0.00	0.21
4 K	check misses	0.04	-	0.25	-	0.49	-	0.54	-	0.43	-	0.25	-	0.10	-
	NI misses	0.05	0.05	0.49	0.49	0.49	0.49	0.57	0.57	0.45	0.45	0.27	0.27	0.12	0.12
	unpins	0.00	0.02	0.00	0.46	0.00	0.37	0.00	0.30	0.00	0.24	0.00	0.07	0.00	0.03
8 K	check misses	0.04	-	0.25	-	0.49	-	0.54	-	0.43	-	0.25	-	0.10	-
	NI misses	0.04	0.04	0.46	0.46	0.49	0.49	0.55	0.55	0.44	0.44	0.25	0.25	0.11	0.11
	unpins	0.00	0.01	0.00	0.40	0.00	0.33	0.00	0.16	0.00	0.14	0.00	0.03	0.00	0.02
16 K	check misses	0.04	-	0.25	-	0.49	-	0.54	-	0.43	-	0.25	-	0.10	-
	NI misses	0.04	0.04	0.38	0.38	0.49	0.49	0.54	0.54	0.43	0.43	0.25	0.25	0.10	0.10
	unpins	0.00	0.00	0.00	0.25	0.00	0.17	0.00	0.09	0.00	0.07	0.00	0.01	0.00	0.00

Table 4: Average translation overhead breakdown: UTLB vs. Intr. (infinite host memory, direct-mapped translation cache with cache index offsetting, and no prefetch)

Cache Entries	Characteristic (per lookup)	Application													
		Barnes		FFT		LU		Radix		Raytrace		Volrend		Water	
		UTLB	Intr	UTLB	Intr	UTLB	Intr	UTLB	Intr	UTLB	Intr	UTLB	Intr	UTLB	Intr
1 K	check misses	0.04	-	0.49	-	0.49	-	0.55	-	0.43	-	0.25	-	0.10	-
	NI misses	0.10	0.10	0.50	0.50	0.50	0.50	0.62	0.62	0.48	0.48	0.31	0.31	0.35	0.35
	unpins	0.00	0.09	0.40	0.49	0.33	0.46	0.21	0.54	0.13	0.41	0.00	0.22	0.00	0.31
2 K	check misses	0.04	-	0.49	-	0.49	-	0.55	-	0.43	-	0.25	-	0.10	-
	NI misses	0.07	0.07	0.50	0.50	0.49	0.49	0.60	0.60	0.46	0.46	0.29	0.29	0.27	0.27
	unpins	0.00	0.04	0.40	0.48	0.33	0.43	0.21	0.44	0.13	0.35	0.00	0.13	0.00	0.21
4 K	check misses	0.04	-	0.49	-	0.49	-	0.55	-	0.43	-	0.25	-	0.10	-
	NI misses	0.05	0.05	0.50	0.49	0.49	0.49	0.58	0.57	0.45	0.45	0.27	0.27	0.12	0.12
	unpins	0.00	0.02	0.40	0.46	0.33	0.37	0.21	0.31	0.13	0.28	0.00	0.07	0.00	0.03
8 K	check misses	0.04	-	0.49	-	0.49	-	0.55	-	0.43	-	0.25	-	0.10	-
	NI misses	0.04	0.04	0.50	0.48	0.49	0.49	0.56	0.56	0.44	0.44	0.25	0.25	0.11	0.11
	unpins	0.00	0.01	0.40	0.42	0.33	0.34	0.21	0.23	0.13	0.21	0.00	0.03	0.00	0.02
16 K	check misses	0.04	-	0.49	-	0.49	-	0.55	-	0.43	-	0.25	-	0.10	-
	NI misses	0.04	0.04	0.49	0.47	0.49	0.49	0.55	0.55	0.44	0.44	0.25	0.25	0.10	0.10
	unpins	0.00	0.00	0.40	0.39	0.33	0.33	0.21	0.21	0.13	0.17	0.00	0.01	0.00	0.00

Table 5: Average translation overhead breakdown: UTLB vs. Intr. (4 MB host memory, direct-mapped translation cache with cache index offsetting, and no prefetch)

for each mechanism is:

$$\begin{aligned} lookup_{utlb} &= user\_check\_hit \\ &+ user\_pin\_cost \cdot check\_miss\_rate \\ &+ ni\_check\_hit \\ &+ ni\_miss\_cost \cdot ni\_miss\_rate \\ &+ user\_unpin\_cost \cdot unpin\_rate \end{aligned}$$

$$\begin{aligned} lookup_{intr} &= ni\_check \\ &+ (intr\_cost + kernel\_pin\_cost) \cdot ni\_miss\_rate \\ &+ unpin\_kernel\_cost \cdot unpin\_rate \end{aligned}$$

In the above equations,

- *ni_miss_cost* is the average cost for the network interface to fetch entries from the UTLB translation table in host memory.
- *user_check_hit* and *ni_check_hit* are the costs that every UTLB translation lookup incurs. The interrupt-based approach incurs the *ni_check_hit* cost every time.

For example, on our Myrinet implementation of UTLB, the *ni_check* is measured at 0.8 μs per lookup, the *user_check* at 0.5 μs, and 10 μs for invoking the system interrupt handler by the network interface. Per-page cost for pinning and unpinning the application buffer depends on how many pages are involved in one call. The SVM applications, whose traces we use to drive our simulation, typically transfer one page of data at a time. On a 300 MHz Pentium-II PC running Windows NT 4.0 pinning one page takes 27 μs and unpinning take 25 μs. On Linux, the pinning and unpinning costs are similar to those on NT. When computing the lookup cost for the interrupt-based approach, the pinning and unpinning costs must be adjusted to factor out context switches. Using these numbers, we can calculate the average translation lookup cost for given applications, as shown in Table 6. The reason why the lookup cost for FFT is higher that for Barnes is that FFT accesses a large amount of virtual memory and hence incurs high page pinning overhead. In both applications, UTLB offers faster translation lookup than the interrupt-based approach.

We also ran the simulation with 4 MB memory restriction on each process to study how each address translation

Cache Entries	Barnes		FFT	
	UTLB	Intr	UTLB	Intr
1 K	2.6 μs	4.9 μs	9.0 μs	21.7 μs
4 K	2.5 μs	2.5 μs	8.9 μs	20.9 μs
16 K	2.5 μs	1.9 μs	8.7 μs	14.8 μs

Table 6: Average lookup cost comparison: UTLB vs. Intr. (infinite host memory, no prefetch, with cache index offsetting)

mechanism behaves under limited memory constraints. The results are shown in Table 5. The number of page pinnings and unpinnings increases for most applications. UTLB stills achieves lower overhead than the interrupt-based approach with the physical memory contraint.

Our trace-driven simulation results show that UTLB has fewer page pinnings and unpinnings than the interrupt-based approach. UTLB does not suffer from a large number of interrupts as does the interrupt-based approach. In addition, the unique design of the host-side translation table permits UTLB to keep translations "alive" even after they are evicted from the network interface translation cache. This results in fewer unpinned pages than the interrupt-based approach. Our results also show that the cost for pinning and unpinning pages can sometimes dominate the cost of address translation (e.g. FFT). It is therefore important to reduce the cost to pin and unpin application pages.

6.3 The effect of cache size and associativity

A miss in the UTLB network interface cache costs several times more than a hit. The average translation lookup cost in the network interface depends on the hit/miss ratio in the UTLB cache. Misses in the network interface cache include capacity, conflict, and compulsory misses. Cache size and associativity directly affect the capacity and conflict miss rates.

In Table 8, we show the overall miss rates in the network interface cache for various cache sizes and associativities. In the table, the rows marked with "direct-nohash" represent a simple direct-mapped cache. The rows marked with "direct" represent a direct-mapped cache that offsets each virtual address with a process-dependent constant in the network interface. This address offsetting technique is used to reduce conflict misses resulting from simultaneous accesses to the network interface by multiple processes. Our simulation results show that this technique works very well. The overall miss rates in a direct-mapped cache are close to, and frequently lower than, those of two-way and four-way set-associative caches. The same offsetting technique is also used for set-associative caches. Cache miss rates would be higher without offsetting.

However, offsetting the cache index may interfere with set-associativity. This may explain the fact that miss rates in the set-associative cache (with offsetting) are higher than those in the direct-mapped cache (with offsetting).

When the actual cost of lookup is considered, the set-associative caches lose to the direct-map cache. The reason is that a set-associative lookup needs to check more entries per cache line than a direct-mapped cache. In a hardware cache, checking of multiple entries on a line can be done in parallel. Since the Shared UTLB-Cache is implemented in Myrinet firmware, the network interface processor can only check one cache entry at a time. Therefore, the cost per translation lookup is higher in a set-associative UTLB cache than a direct-mapped cache. For this particular implementation of UTLB, the trace-driven analysis justifies our choice to use direct-mapping (with offset) for the cache.

6.4 The effect of prefetching

Enlarging the translation cache and offsetting the translation indices can reduce the capacity and conflict misses in the UTLB network interface translation cache. Figure 7 shows the breakdown of translation misses on the network interface for all seven applications, using infinite host memory and a direct-mapped network interface cache without any prefetching. As expected, the number of conflict misses and capacity misses decrease as the cache size increases. But more importantly, this breakdown shows that compulsory misses still constitute the majority of translation misses.

To reduce compulsory misses, we let the Shared UTLB-Cache prefetch multiple entries when handling a translation miss in the network interface cache. We plot the miss rates and lookup cost from RADIX as a function of prefetching size in the two graphs shown in Figure 8. As expected, the overall miss rates decrease as prefetching becomes more aggressive. Prefetching can effectively reduce overall miss rate. This happens for two reasons. First, an application usually displays spatial locality. Prefetching consecutive translation entries takes advantage of such locality. Second, the cost of fetching multiple translation entries increases at a much slower rate than the rate at which overall miss rate drops. As a result, average lookup cost decreases as fetching becomes more aggressive. However, in order for prefetching to work well, translations for contiguous application pages must be available during a miss.

6.5 User-level page-pinning

One way to ensure the availability of translations for contiguous pages is to sequentially *pre-pin* application pages on a check miss in the UTLB user-level library. If the communication's data access pattern displays spatial locality, prepinning reduces the page-pinning overhead for each page pinned, because on most computer systems, pinning a user buffer one page at a time is significantly more expensive than pinning the entire buffer all at once.

Currently the UTLB uses a sequential pre-pinning policy, where if a virtual page needs to be pinned, the user library tries to pin a number of contiguous pages starting with that page. On the other hand, unpinning is still done one page at a time.

Cost	pages	barnes	radix	raytrace	water	FFT	LU
pin	1	1.0	13.0	10.5	2.5	6.1	12.0
	16	0.8	7.3	5.0	1.5	15.8	2.3
unpin	1	0.1	0.1	0.8	0.1	0.1	0.1
	16	0.1	10.8	3.5	0.1	93.0	0.1

Table 7: Amortized pinning and unpinning for different page-pinnng strategy.

In Table 7, we compare the translation lookup performance of UTLB and that of UTLB with 16-page prepinning. The physical memory limit in both cases is 16 MB. In

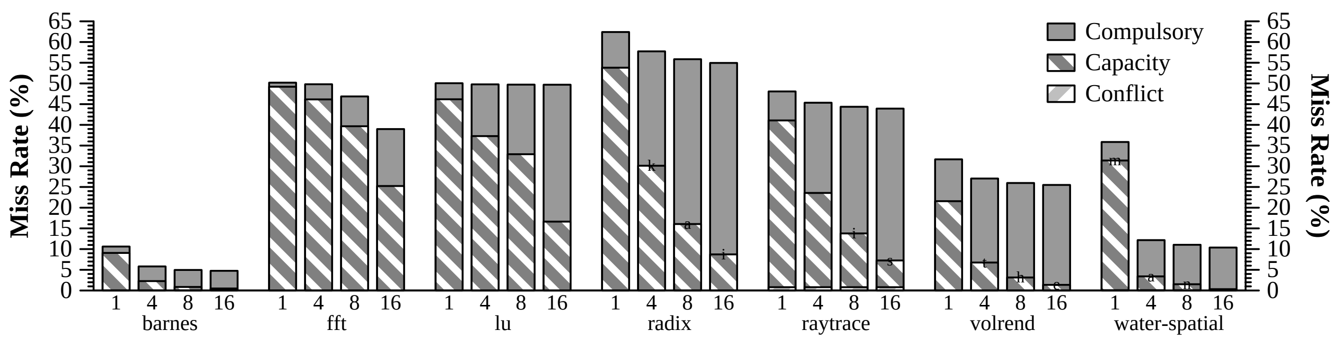
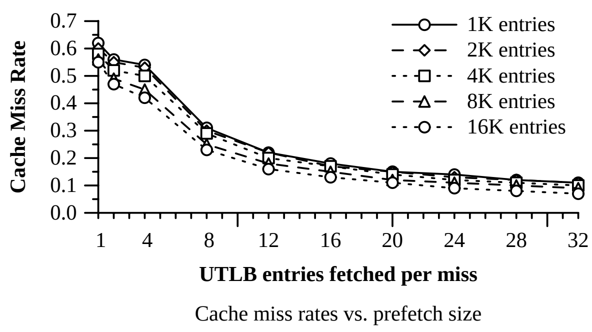


Figure 7: Breakdown of translation cache miss rates for 1K-16K cache entries (with infinite host memory and no prefetch)



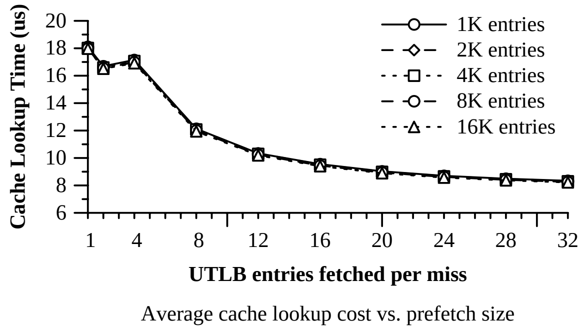


Figure 8: Prefetching effect in the translation cache (RADIX with infinite host memory and a direct-mapped cache)

both cases, the misses in the network interface are the same. The performance difference between the two approaches are from the amortized cost of page-pinnings and unpinnings. We show the amortized cost (averaged over total translation lookups) because page-pinning cost is not a linear function with respect to the number of pages pinned in a system call.

The applications fall in two categories based on their communication patterns: *regular*, which include FFT and LU, and *irregular*, which include the rest [25, 34]. Even the simple sequential policy is very effective for most applications. The only exception is FFT which performs a lot of unnecessary pinning/unpinning with 16-page prepinning. FFT is a regular application with a strided access pattern such that it does not access most of the pages that are prepinned. UTLB is forced to unpin these unused pages when physical memory limit is reached.

7 Conclusions

This paper describes the design and implementation of UTLB, a user-managed address translation mechanism for network interfaces. By maintaining a user-level lookup data structure and a protected translation table, UTLB avoids system calls in the common path of communication and completely eliminates device interrupts. The UTLB approach supports very large communication memory footprints, and requires no special operating system support.

We also presented two enhancements to the basic UTLB design: Shared UTLB-Cache and Hierarchical-UTLB. The Shared UTLB-Cache structure allows an arbitrarily large per-process translation table to be constructed in host memory. We have shown, with micro-measurements and trace-driven analysis, that the overhead of accessing a UTLB translation entry over the system I/O bus is low. This overhead can be further reduced by prefetching multiple translation entries upon a miss in the Shared UTLB-Cache. The Hierarchical-UTLB design further simplifies the construction of UTLB. It also opens up an opportunity to combine the address translation for local buffers with those for receive buffers. In particular, virtual addresses can be used uniformly to represent all buffers.

We conducted trace-driven simulation studies and obtain the following results:

- Our results show that the UTLB approach has fewer misses including both user-level check misses and network interface translation misses than the interrupt-based approach. The UTLB approach can detect most translation misses at user level to avoid interrupts, whereas the interrupt-based approach requires an interrupt on every translation miss.
- The UTLB approach is less sensitive to the translation table sizes than the interrupt-based approach. Even with 1,024 entries, the UTLB approach works quite well.
- Our simulation results also show that direct-mapped approach is adequate for implementing the translation table. This simplifies the design of UTLB data structures on the network interface.
- Prefetching can reduce the amortized overhead of pinning for applications that have regular access patterns and it does not benefit applications that have irregular access patterns.

Our study has several limitations. First, our traces are from shared memory parallel programs though they ran in a

Cache		Applications						
Entries	Associativity	Barnes	FFT	LU	Raytrace	Radix	Volrend	Water
1 K	direct	0.10	0.31	0.35	0.48	0.50	0.50	0.62
	2-way	0.12	0.30	0.32	0.48	0.49	0.50	0.63
	4-way	0.13	0.30	0.30	0.49	0.50	0.51	0.63
	direct-nohash	0.36	0.50	0.51	0.57	0.60	0.78	0.90
2 K	direct	0.07	0.27	0.29	0.46	0.49	0.50	0.60
	2-way	0.06	0.26	0.27	0.46	0.48	0.50	0.60
	4-way	0.07	0.22	0.26	0.47	0.48	0.50	0.60
	direct-nohash	0.35	0.42	0.48	0.57	0.60	0.74	0.90
4 K	direct	0.05	0.12	0.27	0.45	0.49	0.49	0.57
	2-way	0.05	0.11	0.25	0.45	0.47	0.49	0.57
	4-way	0.04	0.10	0.25	0.44	0.46	0.49	0.57
	direct-nohash	0.27	0.35	0.47	0.56	0.60	0.71	0.90
8 K	direct	0.04	0.11	0.25	0.44	0.46	0.49	0.55
	2-way	0.04	0.10	0.25	0.44	0.44	0.49	0.55
	4-way	0.04	0.10	0.25	0.41	0.43	0.49	0.55
	direct-nohash	0.27	0.35	0.46	0.56	0.57	0.71	0.90
16 K	direct	0.04	0.10	0.25	0.38	0.43	0.49	0.54
	2-way	0.04	0.10	0.25	0.37	0.43	0.49	0.54
	4-way	0.04	0.10	0.25	0.34	0.43	0.49	0.54
	direct-nohash	0.27	0.35	0.46	0.50	0.55	0.71	0.90

Table 8: Overall miss rates in Shared UTLB-Cache vs. cache size (infinite host memory, no prefetch, with cache index offsetting for direct, 2 and 4 way)

multiprogramming environment. The memory accesses are quite balanced on all processors. Thus, they may not reveal certain behaviors that multiple independent programs have. Second, we have not compared the per-process UTLB with Shared UTLB-Cache approach because we lack multiple program traces. Third, although the UTLB allows applications to use user-specific strategies for page pinning and unpinning, we only used LRU policy in this study; we have not explored other choices.

Fast address translation alone is not sufficient for good end-to-end communication performance. The design of the translation caching mechanism and its integration with the communication subsystem must allow applications to send and receive data, without copying, using arbitrary buffers. This goal drove the design of the UTLB.

Acknowledgements

This project is sponsored in part by DARPA under grant N00014-95-1-1144, by NSF under grant MIP 9420653 and CDA96-24099, and by Intel Corporation.

We would like to thank Paul Pierce for helping us port VMMC to Windows NT, Ed Felten for useful discussions, and Douglas Clark and Larry Peterson for their useful comments on the draft of the paper. We would also like to thank the program committee and the anonymous referees for their helpful comments.

References

[1] Anindya Basu, Matt Welsh, and Thorsten von Eicken. Incorporating memory management into user-level network interfaces. In *Presentation at IEEE Hot Interconnects V*, August 1997. Also available as Tech Report TR97-1620, Computer Science Department, Cornell University.

[2] M. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. Felten, and J. Sandberg. A virtual memory mapped network interface for the SHRIMP multicomputer. In *Proceedings of the 21st Annual Symposium on Computer Architecture*, pages 142–153, April 1994.

[3] M. A. Blumrich, C. Dubnicki, E.W. Felten, and K. Li. Protected, user-level dma for the shrimp network interface. In *Proceedings of the Second International Symposium on High Performance Computer Architecture*, February 1996.

[4] Matthias A. Blumrich, Richard D. Alpert, Yuqun Chen, Douglas W. Clark, Stefanos N. Damianakis, Cezary Dubnicki, Edward W. Felten, Liviu Iftode, Kai Li, Margaret Martonosi, and Robert A. Shillner. Design choices in the shrimp system: An empirical study. In *Proceedings of the 25th Annual Symposium on Computer Architecture*, June 1998.

[5] 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*, 15(1):29–36, February 1995.

[6] J.C. Brustoloni and P. Steenkiste. Effects of buffering semantics on i/o performance. In *Proceedings of the Second USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, October 1996.

[7] G. Buzzard, D. Jacobson, M. Mackey, S. Marovich, and J. Wilkes. An implementation of the hamlyn sender managed interface architecture. In *Proceedings of the Operating Systems Design and Implementation Symposium*, October 1996.

[8] Pei Cao, Edward W. Felten, Anna Karlin, and Kai Li. A study of integrated prefetching and caching strategies. In *Proceedings of the ACM SIGMETRICS*, May 1995.

[9] Pei Cao, Edward W. Felten, Anna R. Karlin, and Kai Li. Implementation and performance of integrated application-controlled file caching, prefetching and disk scheduling. *ACM Transactions on Computer Systems*, 14(4):311–343, November 1996.

[10] David R. Cheriton. The unified management of memory in the v distributed system. Draft, 1988.

[11] Compaq/Intel/Microsoft. *Virtual Interface Architecture Specification, Version 1.0*, December 1997.

[12] C. Dalton, G.Watson, D. Banks, C. Calamvokis, A. Edwards, and J. Lumley. Afterburner. *IEEE Network*, 7(4):36–43, 1995.

[13] Stefanos N. Damianakis. *Efficient Connection-Oriented Communication on High-Performance Networks.* PhD thesis, Dept. of Computer Science, Princeton University, May 1998. Available as technical report TR-582-98.

[14] Peter Druschel, Bruce S. Davie, and Larry L. Peterson. Experiences with a high-speed network adapter: A software perspective. In *Proceedings of SIGCOMM '94*, pages 2–13, September 1994.

[15] Peter Druschel and Larry L. Peterson. Fbufs: A high-bandwidth cross-domain transfer facility. In *Proceedings of the 14th Symposium on Operating Systems Principles*, pages 189–202, December 1993.

[16] C. Dubnicki, A. Bilas, K. Li, and J. Philbin. Design and implementation of virtual memory-mapped communication on myrinet. In *Proceedings of the 1997 International Parallel Processing Symposium*, 1997.

[17] C. Dubnicki, L. Iftode, E.W. Felten, and K. Li. Software support for virtual memory-mapped communication. In *Proceedings of the 1996 International Parallel Processing Symposium*, pages 372–381, April 1996.

[18] Cezary Dubnicki, Angelos Bilas, Yuqun Chen, Stefanos N. Damianakis, and Kai Li. Vmmc-2: Efficient support for reliable, connnection-oriented communication. In *IEEE Hot Interconnects V*, August 1997.

[19] Cezary Dubnicki, Angelos Bilas, Yuqun Chen, Stefanos N. Damianakis, and Kai Li. Shrimp project update: Myrinet communication. *IEEE MICRO*, 18(1):50–52, January 1998.

[20] R. Gillet, M. Collins, and D. Pimm. Overview of network memory channel for pci. In *Proceedings of the IEEE COMPCON '96*, pages 244–249, 1996.

[21] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach, 2nd Ed.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1996.

[22] Dana S. Henry and Christopher F. Joerg. A tightly-coupled processor-network interface. In *Proceedings of 5th International Conference on Architectur al Support for Programming Languages and Operating Systems*, pages 111–122, October 1992.

[23] Mark D. Hill. *Aspects of Cache Memory and Instruction Buffer Performance*. PhD thesis, Unversity of Berkeley, 1987.

[24] Mark Homewood and Moray McLaren. Meiko CS-2 interconnect elan – elite design. In *Proceedings of Hot Interconnects '93 Symposium*, August 1993.

[25] L. Iftode, J. P. Singh, and Kai Li. Understanding application performance on shared virtual memory. In *Proceedings of the 23rd Annual Symposium on Computer Architecture*, May 1996.

[26] Intel Corporation. *Pentium Processor Data Book*, 1993.

[27] D.B. Johnson and W. Zaenepoel. The peregrine high-performance rpc system. *Software: Practice and Experience*, 23(2):201–221, February 1993.

[28] N.P. Kronenberg, H.M. Levy, and W.D. Strecker. Vaxclusters: A closely-coupled distributed system. *ACM Transactions on Computer Systems*, 4(2):130–146, May 1986.

[29] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy. The stanford flash multiprocessor. In *Proceedings of the 21st Annual Symposium on Computer Architecture*, pages 302–313, April 1994.

[30] P.J. Leach, P.H. Levine, B.P. Douros, J.A. Hamilton, D.L. Nelson, and B.L. Stumpf. The architecture of an integrated local network. *IEEE Journal on Selected Areas in Communications*, SAC-1(5), 1983.

[31] Cheng Liao, Margaret Martonosi, and Douglas W. Clark. Performance monitoring in a Myrinet-connected SHRIMP cluster. In *Proc. of 2nd SIGMETRICS Symposium on Parallel and Distributed Tools*, August 1998.

[32] Richard Lipton and Jonathan Sandberg. Pram: A scalable shared memory. Technical Report CS-TR-180-88, Princeton University, September 1988.

[33] Evangelos P. Markatos and Manolis G.H. Katevenis. Telegraphos: High-performance networking for parallel processing on workstation clusters. In *Proceedings of the 2nd International Symposium on High-Performance Computer Architecture*, pages 144–153, February 1996.

[34] L. R. Monnerat and R. Bianchini. Efficiently adapting to sharing patterns in software DSMs. In *Proceedings of 4th International Symposium on High-Performance Computer Architecture*, February 1998.

[35] Scott Pakin, Mario Lauria, and Andrew Chien. High performance messaging on workstations: Illinois fast messages (fm) for myrinet. In *Proceedings of Supercomputing '95*, 1995.

[36] Paul Pierce. The Paragon implementation of the NX message passing interface. In *Proceedings of Scalable High-Performance Computing Conference (SHPCC) 94*, 1994.

[37] Steven A. Przybylski. *Cache and memory Hierarchy Design: A Performance-Directed Approach*. Morgan Kaufmann Publishers, 1990.

[38] S.K. Reinhardt, J.R. Larus, and D.A. Wood. Tempest and typhoon: User-level shared memory. In *Proceedings of the 21st Annual Symposium on Computer Architecture*, pages 325–336, April 1994.

[39] Rudrajit Samanta, Angelos Bilas, Liviu Iftode, and Jaswinder Pal Singh. Home-based svm protocols for smp clusters: Design and performance. In *Proceedings of 4th International Symposium on High-Performance Computer Architecture*, 1998.

[40] Ioannis Schoinas and Mark D. Hill. Address translation mechanisms in network interfaces. In *Proceedings of 4th International Symposium on High-Performance Computer Architecture*, 1998.

[41] Michael D. Schroeder, Andrew D. Birrell, Michael Burrows, Hal Murray, Roger M. Needham, Thomas L. Rodeheffer, Edwin H. Satterthwaite, and Charles P. Thacker. Autonet: a high-speed, self-configuring local area network using point-to-point links. *IEEE Journal on Selected Areas in Communications*, 9(8):1318–1335, October 1991.

[42] Steven L. Scott. Synchronization and communication in the t3e multiprocessor. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 26–36, October 1996.

[43] J.M. Smith and C.B.S. Traw. Giving applications access to gb/s networking. *IEEE Network*, 7(4):44–52, July 1993.

[44] A. Z. Spector. Performing remote operations efficiently on a local computer network. *Communications of the ACM*, 25(4):260–273, April 1982.

[45] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-net: A user-level network interface for parallel and distributed computing. In *Proceedings of the 15th Annual Symposium on Operating System Principles*, pages 40–53, December 1995.

[46] T. von Eicken, D.E. Culler, S.C. Goldstein, and K.E. Schauser. Active messages: a mechanism for integrated communication and computation. In *Proceedings of 19th ISCA*, pages 256–266, May 1992.

[47] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta. Methodological considerations and characterization of the splash-2 parallel application suite. In *Proceedings of the 22nd Annual Symposium on Computer Architecture*, May 1995.

[48] Y. Zhou, L. Iftode, and K. Li. Performance evaluation of two home-based lazy release consistency protocols for shared virtual memory systems. In *Proceedings of the Operating Systems Design and Implementat ion Symposium*, October 1996.