Context-Aware Address Translation for High Performance SMP Cluster System

Moon-Sang Lee *, Joonwon Lee #2, Seungryoul Maeng #3

* Digital Media R&D Center, Samsung Electronics Corporation
416, Maetan-3Dong, Yeongtong-Gu, Suwon-City, Gyeonggi-Do, Republic of Korea
\texttt{\textsuperscript{1}sang0627.lee@samsung.com}

# Division of Computer Science, Korea Advanced Institute of Science and Technology
373-1 Guseong-dong, Yuseong-gu, Daejeon, Republic of Korea
\texttt{\textsuperscript{2}joon@cs.kaist.ac.kr, \textsuperscript{3}maeng@cs.kaist.ac.kr}

Abstract—User-level communication allows an application process to access the network interface directly. Bypassing the kernel requires that a user process accesses the network interface using its own virtual address which should be translated to a physical address. A small caching structure which is similar to the hardware TLB on the host processor has been used to cache the mappings between virtual and physical addresses on the network interface memory.

In this study, we propose a new TLB architecture for the network interface. The proposed architecture splits an original caching structure into as many partitions as the number of processors on the SMP system and assigns a separate partition to each application process. In addition, the architecture becomes aware of user contexts and switches the content of caching structure in accordance with context switching. According to our experiments, our scheme achieves significant reduction in application execution time compared to the previous approach.

I. INTRODUCTION

SMP (Symmetric Multi-Processor) cluster systems are becoming popular in the areas of high performance computing. Aggregated computing power of multiple processors seems to promise a high performance computing system. However, applications have not experienced the expected performance due to the heavy-weight communication protocols. Traditional communication protocols like TCP/IP require that all network accesses go through the kernel, incurring system call, data copy, and context switching.

User-level communication (ULC) was proposed to reduce the software overhead of communication subsystem. It allows an application process to transfer data directly from a user buffer to a network interface bypassing the kernel by implementing the zero-copy protocol. Accessing a network interface directly from a user space avoids system call, data copy between the user and the kernel space, and context switching. Many previous researchers have shown that ULC guarantees low latency and high bandwidth between two communication endpoints [1].

In ULC, a user process accesses a network interface using its own virtual address which should be translated to a physical address. Since a network interface usually does not have sufficient memory to hold the whole address mappings, only a small portion of the mappings can be cached in a TLB-like structure on network interface memory (TLBN). The TLBN architecture is virtually indexed and virtually tagged like the hardware TLB in the host processor. Each entry in the TLBN maps a page from the virtual address space into the physical address space. The TLBN is small and shared by several application processes, thus one entry of the TLBN can be mapped to several pages which may be in the same or different virtual address spaces. To reduce the conflicts from the shared pages, existing techniques in processor cache design have been used [2].

The TLBN on the network interface is generally managed by firmware software, and the cost of a lookup is increasing linearly with the degree of associativity since it can search only one TLBN entry at a time. In addition, associative TLBN management for replacement becomes more complex and adds extra cost even for the case of a TLBN hit. Therefore, it is necessary to keep the translation mechanism simple while sustaining the miss ratio as low as possible to perform the address translation efficiently.

Previous studies for the TLBN do not consider the multi-programming SMP in depth, even though recent cluster systems are populated by such system. Furthermore, they only show the results of micro-benchmarks like end-to-end latency and one-way bandwidth, and do not provide the effect of the proposed schemes on application execution time. A complex scheme can effectively reduce the TLBN miss ratio, but the complexity may increase the application execution time since the TLBN is organized and managed in software. It is necessary to carefully evaluate an address translation scheme with real-workloads.

We propose a new TLBN architecture, called context-aware TLBN (CATLBN), for the network interface that adopts the zero-copy protocol in the multi-programming SMP system. CATLBN splits a conventional TLBN into as many partitions as the number of processors on the SMP system and distributes concurrent TLBN accesses to separate partitions to exploit the spatial locality of simultaneously running processes. In addition, several consecutive TLBN entries of a partition
constitutes a switching block, and the block is saved to and restored from the kernel memory according to the process context. By preserving a pile of address mappings across interleaved execution of application processes that share a same partition, CATLBN can mitigate conflict misses in the multi-programming system.

We use the virtual interface architecture (VIA) [3] as our base communication model to evaluate the proposed scheme. We implement the CATLBN design in our VIA implementation on the Myrinet network interface which provides flexible ways to explore various design options [4]. For system-wide perspective, we run real-workload benchmarks measuring the total execution time, and the traces from running benchmarks are used to analyze our experimental results.

The remainder of this paper is organized as follows. In the next section, we analyze the address translation cost and describe several design alternatives to efficient address translation. Section 3 describes the CATLBN architecture. In Section 4, we evaluate the proposed scheme using real-workload benchmarks. Finally, in the conclusion we summarize our experimental results and propose future directions for this work.

II. ADDRESS TRANSLATION

In ULC, a user process issues a virtual address while a network interface needs a physical address for DMA operation. If we consider several design issues shown in the next subsection, a network interface is generally required to translate a user virtual address to a physical address. The mapping information needed for the address translation is huge since a large portion of the user virtual address space is involved in communication and there are many user processes participating in communication [5]. If the special translation hardware for DMA [6] does not exist and the mapping information resides only in the host kernel memory, every translation requires an access to the host memory over I/O bus, whose latency can not be tolerated.

A small TLB-like structure on a network interface (TLBN) has been used to overcome the large access latency of the host memory. When a translation is needed, the network interface first looks up the TLBN. The user virtual address is used as an index to the TLBN, and the protection tag of each TLBN entry differentiates one user process from another. On a TLBN hit, the network interface simply uses the physical address in the hit entry. On a TLBN miss, the network interface needs to fetch the missed entry from the host memory.

A. Design Alternative

Schonias and Hill [7] have classified address translation mechanisms into four categories according to where the lookup and the miss handling are performed. If the host processor can execute both of the lookup and the miss handling in the user address space, it is the most desirable method for the address translation since the fast host processor can do it very quickly. However, a malicious or a buggy process may corrupt the address mappings in its own virtual address space and make the system crash. Using a system call could be an effective way for quickly resolving the miss using the fast host processor as shown in [8], but it adds some latency to every message and wastes processor cycles. One should carefully trade off the miss ratio and the system call overhead. If the lookup is performed on the network interface, there are two choices that can handle the miss. One is interrupting the host processor for miss handling service which installs the required physical address into the appropriate TLBN entry. Many previous implementations adopt such interrupting mechanism for miss handling [9], [5], even though the cost of interrupt delivery is known expensive [3]. The user-managed TLB (UTLB) of VMMC [2] handles the miss with intelligent DMA. When the miss occurs, the network interface calculates the correct address of the UTLB structure which is similar to the typical two-level page table in modern operating systems, and the DMA engine fetches the required physical address from the structure. By using DMA, the UTLB handles a TLBN miss in an order of less time than the interrupting mechanism.

The cost for address translation can be expressed as below where $n$ is the number of pages of a message, $C_{\text{lookup}}$ and $C_{\text{miss}}$ is the cost for a TLBN lookup and the TLBN miss penalty respectively, and $P_{\text{miss}}$ is the TLBN miss ratio.

$$n \times (C_{\text{lookup}} + C_{\text{miss}} \times P_{\text{miss}})$$

According to the formula, there are two effective ways for reducing the translation cost. The first is increasing the hit ratio. Like processor cache, a larger size or a higher associativity of the TLBN may increase the hit ratio. The second is reducing the miss penalty. As being dealt with the UTLB, a DMA is preferable to an interrupt for handling a miss.

It seems natural to use DMA, larger TLBN, and high degree of associativity for efficient address translation. However, the memory size of the network interface is limited, and even more, if the actual cost of a lookup is considered, the high degree of associativity is not always desirable. In the case of software-managed TLB, the cost of a lookup is increasing linearly with the degree of associativity since it can search only one entry at a time. Furthermore, the TLBN management for replacement becomes more complex with associativity and adds extra cost even for the simple LRU policy.

A simple scheme which offsets the TLBN index with a process-dependent constant can mitigate TLBN misses as shown in the UTLB study [2]. By offsetting the same index of different processes into different TLBN entries, the conflict misses among different address spaces can be reduced just as there is the virtual associativity. However, the exact value of the process-dependent constant is not specified in the study, and if we consider various parallel applications which use communication buffers in various patterns, it is not an easy problem to determine one single constant which satisfies all parallel applications. Even worse, an inappropriate constant may induce extra misses as in the case of offsetting with set-associativity in the UTLB study.
A large message comprising several pages needs several lookups and may induce multiple misses. If a lookup which belongs to a large message misses, prefetching consecutive TLBN entries may satisfy subsequent lookups for the rest pages of the message. Since the DMA time for moving data increases at a much slower rate than the rate of the increment in data size, fetching additional TLBN entries can reduce the overall miss ratio without enlarging the miss penalty. However, prefetching is highly dependent on the frequency of large messages, and is not useful when the consecutive TLBN entries already have correct mappings.

B. Preliminary Observation

We have found several important observations while evaluating the feasibility of our VIA implementation. There are four concurrent processes on our 4-way SMP system, and they issue network accesses simultaneously which induce conflict misses on a shared TLBN. In general, conflict misses can not be resolved with a larger TLBN, rather a set-associative TLBN can efficiently resolve such conflict misses. Increasing TLBN size is not helpful for the direct-mapped and the 2-way set-associative TLBN. However, for the 4-way set-associative TLBN, larger sizes help reduce TLBN misses resulting that the miss ratio decreases to less than 11% across all benchmarked applications.

The TLBN miss prolongs the per-message latency which affects the application execution time. Although enlarging the TLBN size does not reduce the application execution time of the direct-mapped and the 2-way set-associative TLBN, it slightly reduces the application execution time in case of the 4-way set-associative TLBN. It is notable that the application execution time is more affected by the associativity rather than the TLBN size, and it implies that resolving conflict misses is critical to improve application performance. Furthermore, simplifying the TLBN architecture while providing the miss ratio as low as possible can significantly reduce the application execution time.

III. CONTEXT-AWARE ADDRESS TRANSLATION

In this section, we propose a new TLBN architecture for zero-copy protocols considering both of simultaneous and interleaved execution of multiple processes. By assigning a separate TLBN partition to each concurrent process, we reduce conflict misses from simultaneous accesses to a shared TLBN. In addition, we utilize the TLBN affinity to reduce conflict misses from interleaved TLBN accesses in the multi-programming system.

A. Splitting TLBN

A parallel application generally launches many symmetric processes whose communication buffers have the same virtual addresses. A user virtual address may have the same TLBN index of another user virtual address since the index is calculated with a portion of virtual address. The index interference from concurrent processes massively induces conflict misses, and they dominate the overall TLBN miss [2]. Therefore, it is crucial to separate the indices of different user virtual addresses.

The index interference from concurrent processes on the SMP system can be simply addressed by assigning a separate TLBN to each concurrent process instead of using a single shared TLBN. We split a single large TLBN into as many small partitions as the number of processors on the SMP system and distribute concurrent TLBN accesses to separate partitions. Figure 1 shows an example of the split TLBN for the 4-way SMP system. Every lookup, first, is examined with the pid which is a unique number assigned by the communication subsystem to determine which partition to use. The partition selection function, $F(pid)$, can be expressed as $(pid \mod n)$ if the number of processor is two to the $n$. Then, the TLBN index is calculated with a portion of the user virtual address. If the indexed entry has the matching protection tag (PTAG) with the current lookup, the physical address in the entry is used for DMA. Otherwise, the network interface handles the miss appropriately.

![Split TLBN for the 4-way SMP system](image)

Obviously, splitting the TLBN may decrease the available number of entries for an application process to use, resulting in the increased capacity miss. However, the split TLBN aggressively exploits the spatial locality of an application process with separate partitions and reduces the conflict miss which otherwise requires a high degree of associativity to be resolved. As a result, the split TLBN with the simple selecting function can reduce the overall miss ratio while preserving fast lookups.

B. Preserving TLBN Affinity

In the multi-programming system, there may be lots of processes on a system competing for the host processor. If they are many more than the number of host processor, a partition in the split TLBN is accessed by more than one process. In addition to the interference from concurrently executing processes, the interleaved execution of user process also interferes with each other and collapses the warmed-up TLBN of the previously executed process. If a TLBN can be allocated to a user process separately, the interference can be resolved, but such per-process TLBN requires huge memory in

---

1Implementation details and evaluation environments are described in Section IV.
the network interface. On the multi-programming SMP system, it is important to preserve the TLBN affinity of each user process across context switches.

Addresses should always be translated by a qualified entity like the kernel or the firmware on the network interface to ensure the validity of translation. Previous approach using interrupts relies on the kernel and the UTLB on the firmware for the translation. In any case, the full correct mappings are maintained by the kernel and reside in the kernel space. We invest some amount of kernel memory to cache the TLBN according to user context. While the cache on the network interface, TLBN, is mainly for reducing the access latency of the host memory, the cache on the kernel memory, named TLBN cache, is for keeping the TLBN affinity across context switches.

Figure 2 shows the context-aware TLB architecture on the network interface (CATLBN). In the kernel space, every user process has its own TLBN cache which is as large as a TLBN partition and is allocated when a user process opens the network interface. The address translation table (ATT) is a kind of page table for communication buffers, where the correct virtual-to-physical mappings of all communication buffers are maintained. In the network interface, a partition is divided into blocks, and each block has a flag specifying which process currently owns the block. A portion of the TLBN which is warmed-up during the process execution is saved to the TLBN cache when the process switches out and is restored back to the same place of the same partition when the process switches in.

![Fig. 2. CATLBN architecture](image)

Figure 3 describes how the address translation works in the CATLBN architecture. If a lookup hits in the TLBN, the translation finishes successfully with the correct physical address. If it misses, a block index is calculated from the TLBN index, and the block owner is compared with the current pid which is looking up the TLBN. If the block owner is equal to the pid, fetching the missed entry from the ATT and updating the PTAG make the lookup hit. Otherwise, the TLBN entries in the block are saved to the TLBN cache which belongs to the current block owner, and the address mappings in the pid’s TLBN cache are restored back to the block. Updating the owner of the block to pid finishes switching address mappings for the block.

Fig. 3. Basic algorithm for TLBN cache management

A process context may be prolonged with the zero-copy protocol since the process does not use system call for communication and the kernel takes less chance of context switching. Thus, it is highly expected that the process will continue to lookup adjacent entries of the TLBN. Restoring the block in bulk potentially helps subsequent lookups hit in the TLBN.

The CATLBN architecture can be effective just as they have the per-process TLBN. TLBN accesses from concurrent processes are distributed to separate partitions, and TLBN accesses from interleaved processes are taken care by preserving the TLBN affinity. Therefore, the conflict miss which dominates the TLBN miss can be reduced remarkably.

According to the basic algorithm, a missed lookup may miss again after restoring the matching block since the TLBN cache could hold inappropriate mapping. It, then, is necessary to fetch the correct physical address from the ATT. However, two separate DMAs, one for switching the block and the other for fetching the missed TLBN entry, are not tolerable due to the large start-up cost of DMA.

Normally, network devices provide the DMA chaining facility which enables multiple DMAs with a single start-up command. We can take advantage of such chaining facility to combine two separate DMAs. The basic algorithm can be refined with three sequentially chained DMAs for save, restore, and fetch (SRF) when a lookup misses and the pid is not equal to the block owner. The fetching DMA is always chained to the block switching regardless of the equality between the pid and the block owner, and overwrites the TLBN entry for which the lookup has just missed. As a result, every TLBN miss can be satisfied with a single chained DMA.

Chaining separate DMAs can reduce the average DMA latency by overlapping the start-up delay, but the lookup may still wait too long since the fetching DMA is at the end of the chain. The DMA speed of I/O device is asymmetric. A write DMA which transfers data from the device to the host memory is always faster than a read DMA that is actually composed of a request to and a reply from the host memory. Restoring is a read DMA, and it would delay the fetching DMA too far. If the lookup is for a synchronization message like a lock acquiring message, the delay may degrade the application performance severely.
The best effort approach changes the DMA order by pushing the restoring DMA to the end of DMA chain. The order: save, fetch, and restore (SFR), returns the missed TLBN entry immediately and initiates the restoring DMA at the tail of the chain. Note that we do not wait the restoring DMA to complete, but try to restore the block and proceed to process other operations while expecting that it would restore effective mappings for subsequent lookups in time. If the next lookup misses in the same block, fetching a single entry from the ATT satisfies it since the block owner and the current process are same.

When there is no entry fetched from the ATT in a block, it is unnecessary to save the block which may be in the initial or the restored state. We define the block affinity as the number of fetched entries in a block and use it as an indicator whether to save the block or not. The block affinity increases by one whenever a fetching DMA is done for the block and is set to zero when the block is restored. With the block affinity, it is possible to omit the unnecessary saving DMA from the DMA chain, and the average DMA latency can be reduced. Figure 4 summarizes the refined algorithm for TLBN cache management.

In summary, our design of CATLBN architecture is based on the expectation for spatial and temporal locality of TLBN lookup. Once a process starts to lookup a TLBN, it is highly expected that the process will continue to lookup the adjacent entries of the TLBN which may compose a user buffer. In addition, the zero-copy protocol may prolong the process context since a process keeps the occupation of host processor making the kernel have less chance of context switching. Thus, recently referenced TLBN entries are likely to be referenced again in the near future. We try to increase the hit ratio of TLBN lookup by preserving the warmed-up TLBN entries and reduce the average lookup time in spite of fetching additional TLBN entries. We expect the CATLBN architecture can reduce the average lookup time overall since the time for moving data in DMA increases at a slow rate.

IV. PERFORMANCE EVALUATION

Experiments were performed on a system of two SMP nodes running Linux 2.6. Each node has four 1.5 GHz Xeon processors with 4 GB main memory and is equipped with 66 MHz 64 bit PCI bus where a Myrinet network interface is plugged.

We use the software distributed shared memory (DSM) system to measure the TLBN miss ratio and its effect on application execution time. The KAIST distributed shared memory (KDSM) system is a HLRC-based software DSM implementation over TCP/IP [10]. We have ported the KDSM system to use the VIA APIs, and used the KDSM over VIA as our evaluation framework for application performance. Four applications from the SPLASH-2 benchmark suites [11] were run on top of the KDSM system.

1) Experiment for SMP system: We run SPLASH-2 applications with 8 processes across two 4-way SMP nodes. We compare the proposed architecture, SPLIT, with the best of previous TLBN architectures, SA4. Figure 5 shows the result of miss ratio and lookup time for various TLBN sizes. The split TLBN investigates the same amount of network interface memory as the 4-way set-associative TLBN, but has four separate partitions each of which is one fourth of the investigated memory. The split TLBN distributes TLBN accesses from concurrent processes to separate partitions and shows as same miss ratios as the 4-way set-associative TLBN. The simple partition selection function enables the split TLBN to resolve conflict misses efficiently, thus the lookup time of SPLIT is much lower than that of SA4.

Since a parallel program may show the execution time variation between runs, we repeated every run of an application a hundred times and calculated the mean and variance of application execution time. According to our measurement, the standard deviation is within 1 % of the average execution time. Figure 6 shows the normalized execution time of SPLASH-2 application on the split and SA4 TLBN for various TLBN sizes. The base time for normalization is the 4-way set-associative TLBN with 4k entries. As shown in the graph, the proposed SPLIT shows less execution time than SA4 in every TLBN size across all applications.

2) Experiment for Multi-programming SMP System: To evaluate the CATLBN architecture on the multi-programming...
SMP system, we run twice as many processes as the number of host processor on a node. The left graph in Figure 7 shows the miss ratio variation of CATLBN with 32k entries in accordance with the block size. The first point in each line graph represents the miss ratio of the split TLBN. The numbers on the x-axis denote the number of TLBN entries per block. Since one TLBN entry is 8 bytes in size, the block size ranges from 8 bytes to 512 bytes. As the block size gets larger, the miss ratio gets lower until it is saturated around the block size of 64 entries.

The CATLBN with a single entry block operates as same as the split TLBN, thus they show almost the same miss ratios. The additional latency which comes from switching a single entry block is added to every missed lookup and the increased lookup time is shown at the second point in the right graph of Figure 7. The average lookup time decreases as the block size gets larger. Although switching a block of 64 entries takes longer than fetching a single entry, our algorithm for TLBN management minimizes the enlarged DMA latency. i.e., chaining several DMAs, reordering DMAs in a chain, and omitting saving DMA based on the block affinity. Therefore, the CATLBN with the block size of 64 entries outperforms other TLBN architectures across all applications. Note that SA4 reduces the miss ratio but shows little reduction of lookup ratios. The additional latency which comes from switching a block of 64 entries takes longer than fetching a single entry block is added to every missed lookup and the miss ratio gets lower until it is saturated around the block size of 64 entries.

V. CONCLUSION

This study has demonstrated two approaches for address translation in the multi-programming SMP system. We have evaluated the proposed schemes with real-workload benchmarks in the multi-programming SMP system. Our experiments provide consistent results in the unloaded SMP system. However, there are significant variations of application execution time in the loaded multi-programming SMP system resulting that the standard deviation of execution time exceeds the average reduction of application execution time. To get sufficient confidence in the loaded multi-programming SMP system, it is worth experimenting on the coscheduler adopted cluster system.

The CATLBN outperforms previous schemes showing 7.3 % ~ 22.5 % reductions in application execution time. However, our result presents that application execution time is dispersed across a hundred runs, and the standard deviation varies from 5.4 % up to 61.7 % of the average execution time. Such variance is mainly due to the competition for the host processor. In current Linux, processes of a parallel application are independently scheduled by the kernel scheduler on each node without any effort to coschedule them. If a message has arrived and the destined process has been scheduled out, the message can not be consumed immediately, and other dependent processes should wait until the process consumes the message, proceeds its execution, and outputs intermediate data. Recent researches for coscheduling [12] rely primarily on the arrival of a message to determine when and which process to schedule. We expect that communication-driven coscheduling could settle the large variance of execution time in the loaded multi-programming SMP system.

REFERENCES