Adopting System Call Based Address Translation into User-Level Communication

Moon-Sang Lee, Sang-Kwon Lee, Joonwon Lee, and Seung-Ryoul Maeng
Department of Electrical Engineering and Computer Science
Korea Advanced Institute of Science and Technology
E-mail: {mslee, sklee, joon, maeng}@camars.kaist.ac.kr

Abstract—User-level communication alleviates the software overhead of the communication subsystem by allowing applications to access the network interface directly. For that purpose, efficient address translation of virtual address to physical address is critical. In this study, we propose a system call based address translation scheme where every translation is done by the kernel instead of a translation cache on a network interface controller as in the previous cache based address translation. According to our experiments, our scheme achieves up to 4.5\% reduction in application execution time compared to the previous cache based approach.

I. INTRODUCTION

Cluster systems are becoming prevalent in the area of high performance parallel computing. Aggregated computing power of multiple PCs seems to promise a low cost and high performance system. However, applications have not experienced the expected performance due to the overhead incurred by heavy-weight communication protocols like TCP/IP.

User-level communication (ULC) was proposed to reduce the software overhead of the communication subsystem. It allows an application process to transfer data directly from a user buffer to a network interface controller (NIC) bypassing the kernel. Accessing a NIC 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 [2].

In ULC, a user process accesses a NIC using its own virtual address which should be translated to a physical address. Since a NIC usually does not have sufficient memory to hold the whole address mappings, only a small portion of the mappings can be cached on its memory. When a cache miss occurs, the NIC acquires a physical address either by interrupting the kernel for miss handling or fetching the missed cache entry using DMA.

In this study, we propose an alternative translation scheme which relies on the kernel for feeding the needed physical address into the NIC’s memory. The main difference from the previous interrupt based scheme is that the kernel service is initiated by a system call rather than by a hardware interrupt which incurs longer delay than a system call. Our scheme defeats the purpose of ULC by using a system call, but it eliminates the delay incurred by cache misses on the NIC since the kernel provides the NIC with all physical addresses for data transfer. Though the system call adds some latency to every message, modern processors are getting faster and faster at a unprecedented rate alleviating the system call overhead.

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

II. ADDRESS TRANSLATION

Previous studies have shown that network protocols spend a significant amount of time on copying data between a user buffer and a dedicated kernel buffer [8], [10]. ULC schemes avoid such data copying with the use of zero-copy protocols which directly transfer data from an arbitrary user buffer to the memory in a NIC. Since user processes issue virtual addresses while the NIC needs physical addresses for data movement, it is necessary 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.

If the special translation hardware for DMA [9] 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 cannot be tolerated.

A small translation cache on a NIC is used to overcome the large access latency of the host memory. When an address translation is needed, the NIC first looks up the translation cache. User virtual address is used as an index to a cache entry, and the protection tag of the cache entry differentiates one user process from another. On a cache hit, the NIC simply uses the physical address in the hit cache entry. On a cache miss, the NIC needs to retrieve the physical address from the host memory.

Schoinas and Hill [7] have classified address translation mechanisms into four categories according to where the lookup and the miss handling are performed. If the lookup
is performed on the host processor, misses are handled there. If the lookup is performed on a NIC, there are two choices that can handle the misses. One is interrupting host processor for miss handling service, and the other is handling a cache miss with an intelligent DMA. Interrupting host processor launches kernel interrupt service routine (ISR) which directly installs the required physical address into an appropriate cache entry. Many previous ULC implementations adopt such an interrupting mechanism for miss handling [3], even though the cost of interrupt delivery is known expensive [2]. The user-managed TLB (UTLB) of VMMC [4] handles a cache miss with an intelligent DMA. The UTLB structure is similar to the typical two-level page table in modern operating systems. When a cache miss occurs, the DMA engine on a NIC fetches a physical address from the 2nd-level page table. By using DMA, the UTLB handles a cache miss in a couple of microseconds.

The cost for address translation can be expressed as below

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

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

It seems natural to use DMA, large cache size, and high degree of associativity for efficient address translation. However, the memory on a NIC is limited in size [1], and even more, if actual cost of a lookup is considered, a high degree of associativity is not always desirable. In the case of a software managed cache, the cost of a lookup is increasing linearly with the degree of associativity since it can search only one cache entry at a time. Furthermore, cache management for replacement becomes more complex and adds extra cost even for the case of a cache hit.

III. ADDRESS TRANSLATION WITH SYSTEM CALL

To send a message, a user process prepares a descriptor which contains a starting virtual address of a user buffer, the message size, and other related information like protection. When a user process initiates data transfer, a user command which is composed of the descriptor’s virtual address notifies a NIC that a descriptor is ready at the given virtual address. The NIC should translate the virtual address to a physical address to download the descriptor. In traditional cache based approaches, the NIC looks up the translation cache, and if a cache miss occurs it should be handled properly as mentioned in the previous section. After downloading the descriptor and checking errors on the downloaded descriptor, the NIC extracts a virtual address of a user buffer from the descriptor and translates it to a physical address using the translation cache. User data can be moved to the NIC after completing the above steps.

Address translation is on the critical path of data transfer since actual data movement can be done after its completion. Two cache lookups, one for the descriptor and the other for the user buffer, are sufficient for a small message. A large message comprising several pages needs several cache lookups and may induce multiple cache misses which not only increases the message latency but also delays other messages behind the message.

Addresses should always be translated by a qualified entity like the kernel or the firmware on a NIC to ensure the validity of translation. Previous cache based approach using interrupts relies on the kernel and the UTLB on the firmware for the address translation. In any case, the full correct mapping information is maintained by the kernel and the system call can facilitate the use of the information.

We propose an alternative translation scheme called SAT (System call based Address Translation) which relies on the kernel for the address translation. The SAT adopts a system call instead of an interrupt for a translation. At every data transfer the SAT module in the kernel space informs physical addresses of a user buffer to the NIC, and then the NIC immediately initiates data transfer without any cache lookup or miss handling.

Figure 1 shows a simplified sending process of the SAT. Upon a user request for a send transaction, a user process invokes a system call which makes the kernel generate a kernel-level descriptor, and physical addresses of a user buffer are appended at the tail of the descriptor. Since the kernel has the knowledge of physical addresses, a SAT command can be composed of a physical address of a kernel-level descriptor and notifies a NIC that a descriptor is ready at the given physical address. The NIC simply uses the physical address to download the descriptor using DMA. With the physical addresses in the downloaded descriptor the NIC downloads user data without any cache lookup.

Obviously, the SAT may add some latency to every message and waste processor cycles which would otherwise be used for useful computations. However, the SAT can reduce the message latency and increase the message throughput because of two reasons. First, the kernel has all address mapping information, thus cache misses can be eliminated on a NIC. Second, several lookups for a large message take long for a NIC even if all lookups hit on the cache. A host processor which is much faster than a NIC enables the lookups to be done more quickly.
IV. PERFORMANCE EVALUATION

A consortium of industry companies has specified the virtual interface architecture (VIA) [5] as an industry standard for system area network (SAN). We have implemented the VIA on Myrinet and applied the SAT to our implementation. Although our experiments are conducted with our VIA implementation on Myrinet, the SAT design is applicable to any user-level communication subsystem including the InfiniBand.

Experiments were performed on a system of four SMP nodes running Linux 2.4. Every node has four 1.5 GHz Xeon processors and is equipped with 66 MHz 64 bit PCI bus where a Myrinet NIC is plugged. Our Myrinet NIC, M2L-PCI64B, runs at 133 MHz and is interconnected with 1.28 Gbps full-duplex links.

We measured basic system features of our experimental environment. With repeated measurement we found that a dummy interrupt from the NIC to the host processor takes 19 μs, a 4 bytes DMA from the host memory to the NIC takes 2 μs, and a dummy system call takes 1.6 μs. According to our measurements, using a DMA instead of an interrupt could reduce cache miss penalty and using a system call is preferable to using a DMA when a cache miss occurs.

A. Experiment with Micro-benchmarks

Figure 2 shows the result of micro-benchmarks. The left and the right graph shows end-to-end latency and one-way bandwidth, respectively. DM stands for our base implementation of VIA using a direct mapped cache, SAT for the SAT applied implementation, and VIGM for Myricom’s VIA implementation. The user buffers in each micro-benchmark are small and the cache hit ratio is higher than 99.99 %, thus the effect of cache miss is not included in the results. To show the effect of cache miss, all cache lookups of direct-mapped cache are intentionally missed in ALL MISS.

VIGM shows lower latency than DM for small messages. For large messages beyond 1024 bytes, DM shows lower latency than VIGM. Some techniques like using programmed IO or embedding messages in a descriptor can reduce the latency of small messages, and VIGM may deal with small messages in such a way. Due to the lack of published materials for VIGM, we do not provide in-depth comparison between DM and VIGM.

The latency of the SAT is slightly higher than that of DM for small messages since it includes system call overhead, but the latency gap decreases as the message size increases. The SAT shows the best throughput across all message sizes since the burden on the NIC is lessened by removing the address translation from the NIC. In the case of all cache miss, the latency increases dramatically resulting 29.93 μs for a 4 bytes message.

Table I shows the result of end-to-end latency for three software managed caches and the SAT. SA2 and SA4 stands for the 2-way and the 4-way set-associative cache, respectively.

Higher degree of associativity leads to larger latency regardless of the message size due to the management overhead of complex cache structure, and a large message magnifies the latency gap between the direct-mapped and the set-associative cache since the message needs several cache lookups. The cache based address translation may reduce cache misses with a high degree of associativity, but it increases the per-message latency. On the other hand, the SAT shows reasonable latency for small messages and low latency for large messages while guaranteeing no cache miss.

B. Experiment with Real-workloads

We use software distributed shared memory (DSM) system to measure the cache miss ratio and its effect on real-workload execution time. KAIST distributed shared memory (KDSM) system is an HLRC-based DSM implementation over TCP/IP [6]. We have ported the KDSM system on our VIA implementation. The SPLASH-2 benchmark suite [11] is tested with four processes running on each node. Table II shows problem size, the number of packets, and the number of cache lookups per node for each application. Three applications, fft, ocean and water-spatial, are known to generate heavy communication traffic. The message size distribution is shown in Figure 3 where some applications show two humps while others (raytrace and water-spatial) do not. This traffic pattern has been seen in many request/reply communications and implies that the communication subsystem should take care of not only small messages but also large messages. Considering Table II and Figure 3, we can forecast that the performance of fft and ocean would be most sensitive to the efficiency of the communication subsystem.

1) Cache Miss Ratio: Figure 4 shows the miss ratios for various cache sizes and associativities. For each associativity, cache sizes are varied with 1k, 2k, 4k, 8k, and 16k entries. Increasing cache size is not helpful for the direct-mapped cache since simultaneous accesses from four concurrent processes

1 It takes 0.6 μs for 850 MHz P-III. It is well-known that the micro-architecture of Xeon is not good for a system call.

2 For the sake of simplicity we exclude the result of interrupting mechanism in the rest of this paper.
TABLE II

<table>
<thead>
<tr>
<th>Workload</th>
<th>Problem Size</th>
<th># of packet</th>
<th># of lookup</th>
</tr>
</thead>
<tbody>
<tr>
<td>fft</td>
<td>4M points</td>
<td>402,750</td>
<td>3,058,976</td>
</tr>
<tr>
<td>lu</td>
<td>2k x 2k matrix</td>
<td>184,532</td>
<td>827,730</td>
</tr>
<tr>
<td>ocean</td>
<td>514 x 514 ocean</td>
<td>492,585</td>
<td>2,007,335</td>
</tr>
<tr>
<td>radix</td>
<td>16M keys</td>
<td>235,994</td>
<td>1,459,432</td>
</tr>
<tr>
<td>raytrace</td>
<td>256 x 256 car</td>
<td>180,164</td>
<td>1,495,832</td>
</tr>
<tr>
<td>water-sp.</td>
<td>2 steps 4096 molecules</td>
<td>651,898</td>
<td>978,753</td>
</tr>
</tbody>
</table>

Fig. 3. Message Size Distribution of SPLASH-2 Applications

Fig. 4. Cache Miss Ratio of SPLASH-2 Applications

interfere with each other and conflict misses dominate over capacity misses. For set-associative caches, larger cache sizes reduce cache misses resulting that the 4-way set-associative cache with 16k entries has the miss ratios below 5 %.

2) Application Execution Time: Cache misses for address translation enlarge the per-message latency which affects execution time of parallel applications. Figure 5 shows the total execution time of SPLASH-2 applications for various cache configurations with 16k cache entries.

Our base implementation (DM) and other variations from it outperform VIGM for all six applications. Particularly, the variation (ALL MISS) which has 100 % cache miss also performs better than VIGM since VIGM is inferior for message throughput and SPLASH-2 applications have heavy communication traffic. Although the miss ratio of SA4 is much lower than that of DM as shown in Figure 4, its execution time is slightly less than that of DM because the high degree of set-associativity prolongs the cache lookup time.

Notice that in spite of the use of system call which adds some latency to every message, the SAT shows shorter execution time than any other cache based approaches for all applications except the raytrace case. Raytrace has the least communication traffic and most of them are small messages; raytrace is more dependent on the host processor than the communication subsystem. Other than the raytrace case, the SAT shows up to 4.5 % improvement over the best cache based approach.

V. CONCLUSION

This study has demonstrated an approach for address translation with real-workloads. Because of the characteristic of software managed cache the high cache hit ratio with associativity does not help an application reduce its execution time. Based on the high speed micro-processor and the fully informative kernel, our scheme has achieved up to 4.5 % reduction in application execution time compared to the best cache based approach. The future work of this study is to integrate the SAT into the InfiniBand and investigate the feasibility. We expect that the SAT could fully utilize the ultimate speed of the InfiniBand.

REFERENCES