Critical Factors in NUMA Memory Management

Jayashree Ramanathan and Lionel M. Ni

Department of Computer Science
Michigan State University
East Lansing, MI48824-1027

Abstract: The potential high latency of remote memory access seriously degrades performance in NUMA multiprocessors. Replication has been proposed as a solution, but it uses more memory and also raises consistency issues. In this paper, we identify and incorporate the critical factors that influence NUMA memory management into our performance metrics. Using trace-driven simulations, we show that under certain conditions no replication is better than replication. We also conclude that the effectiveness of replication depends on: (1) the ratio of access times to remote and local memory, (2) virtual address assignment to data, (3) data sharing characteristics, (4) overhead of enforcing consistency, and (5) amount of physical memory available relative to the data sharing characteristics.

1 Introduction

Medium to large-scale multiprocessors typically consist of a large number nodes connected together by means of a multi-stage interconnect (MIN) or a point-to-point interconnect. Each node has a processor and local memory. It is also possible for the nodes to share a common global memory. Such multiprocessors are characterized by a non-uniform memory access (NUMA) time, with local access being faster than remote or global access, and are also referred to as NUMA multiprocessors. We restrict ourselves to NUMA multiprocessors with no global memory such as BBN's TC2000. We are interested in mechanisms that reduce the performance degradation due to remote memory accesses in supporting a paged global virtual memory (PGVM). Though replication has been proposed as a solution [1] [2] [3], it requires more memory and also raises consistency issues. Since replication needs more memory, page replacement is important, particularly for applications with a high degree of sharing. Existing NUMA multiprocessors do not study page replacement in detail. Further, the critical factors that influence NUMA memory management, both with and without replication, and the conditions under which replication is effective have not been studied. We identify and incorporate those factors into our performance metrics. We use trace-driven simulations to determine the conditions under which replication is effective.

The rest of the paper is organized as follows. Section 2 summarizes related work and provides motivation for our work. Sections 3 and 4 outline memory management policies used in our study. Performance metrics are covered in Section 5, followed by a discussion of experiments and results in Section 6. The last section provides conclusions and future work.

2 Related Work

In this section, we overview existing literature in supporting PGVM in NUMA multiprocessors. The studies differ in whether replication or migration or both or neither is used, in managing read-only (RO) and read-write (RW) application pages. Further, studies that replicate RW pages differ in the consistency model supported and the policies used to enforce consistency.

Since the port of Mach is the same as supporting PGVM, studies that port Mach are of interest to us. Parallel virtual memory for the TC2000, derived from Mach by parallelizing kernel code [4], neither replicates nor migrates pages and uses software LRU approximation for page replacement. The port of Mach on RP3 replicates RO pages and private RW pages, but neither replicates nor migrates shared RW pages. Studies that migrate but do not replicate RW data include the port of Mach on ACE [5], trace-driven simulations by Black and others [6], support of GVM on an iPSC/2 by Li and Schaefer [7], migration by page pivoting for a mesh interconnect [8], and migration by extending...
the memory hierarchy into the interconnect [9].

We now look at studies that replicate RW pages. PLATINUM [1], an implementation of GVM on a Butterfly Plus, selectively replicates RW pages depending on the number of invalidations by the write-invalidate (WI) policy that enforces sequential consistency (SC). It uses Mach's computation model and a directory-based ownership scheme to manage application pages. Page tables are replicated and consistency is maintained by interrupting processors that actively use the updated page table entry. Page replacement policies are not studied. Another study is the implementation of various migration and replication policies along with the WI policy to enforce SC for a GP1000 [3]. Software LRU approximation is the only replacement policy studied. The write-update (WU) policy to enforce consistency is not studied. NUMA page placement is also reported in [10]. Another relevant implementation is PLUS [2] that uses replication and supports the weak coherence (WC) consistency model using the the WU policy, implemented by special hardware synchronization operations and remote reference counters. The software cache management scheme that adapts itself to the reference pattern proposed by Bennett and others [11] is also of interest.

Most of the previous work is actual implementation. Performance is usually measured in terms of speedup or execution time. Such a performance measure applies to the specific multiprocessor in question and does not provide a comparison across different multiprocessors. Simulation studies reported above are tailored to specific architectures. To the best of our knowledge, the influence of the variation in design parameters, such as the ratio of access times to remote and local memory and the number of pages, on performance has not been reported. Further, no study has been done as to when replication is effective. Also, page replacement has not been studied in detail. In contrast, we measure performance by the effective memory access time. We identify factors, both hardware and software, that are critical to NUMA memory management and incorporate them in our performance measure. We use trace-driven simulations that allow great flexibility in varying the various factors and also have a shorter turnaround time than actual implementation. We determine the conditions under which replication is effective and also study page replacement.

3 NUMA Policies

In this section, we outline policies used in our study for all memory management issues except page replacement, which we discuss in the next section.

Page Placement

Fault processor placement, that places the virtual page in the faulting processor's local memory, and Modulo placement, that places the virtual page in the local memory of the processor given by: (virtual page number) mod (number of processors), are used.

Page Replication

Full replication (FR), that always replicates pages in the faulting processor's local memory, and no replication (NR), that maintains a single physical copy of any virtual page, are used.

Consistency Model

Consistency is not an issue in NR, as there is a single physical copy of any virtual page. However, in the case of FR, some consistency model needs to be enforced among the various physical copies to ensure correct program execution. We use the WI and WU policies to enforce the SC consistency model. WU is also referred to as multicast, as it is similar to multicast communication.

Page Migration

Page migration occurs under certain conditions when the page replacement policy is invoked. We do not study explicit page migration.

TLB and Page Table Management

Each processor has its own TLB. A single-hand clock policy is used for TLB replacement. The two-level indexed page tables are replicated on all processors and are never replaced.

TLB and Page Table Consistency

In both the FR and NR cases, each processor has its own TLB and page table. In the FR case, since the TLB and page table entries in each processor pertain to the processor's local copy, consistency is not an issue. However, in the NR case, the TLB and page table entries in each processor pertain to the single physical page allocated to the virtual page. Status flags such as dirty, which indicates whether the page has been updated, and reference, which indicates whether the page has been referenced, are modified and therefore consistency is an issue. Our page replacement policy does not make use of the reference flag. We adopt the following approach to update the dirty flag. Whenever a processor updates a page, it updates the dirty flag only in its own TLB and page tables, not those in the other processors. When a page is replaced, it is determined if it has to be written to the disk by checking the dirty flags in all processors. Therefore, it is not necessary to keep the TLB and page table entries consistent at all times.

Kernel Page Management

Information that is needed for managing virtual
pages, such as the processors using the virtual page and the number and type of operations (read, write, etc.) on the virtual page, are maintained in a shared virtual page data structure that resides in kernel area. Kernel pages are not replicated and are interleaved across all memory modules.

We conclude this section by outlining the possible conditions during virtual-to-physical address translation. Address translation can result in a TLB hit, or a page table hit, or it can result in a miss in both the TLB and the page table, referred to as a fault. A fault on a virtual page allocated remotely is called a remote page fault (RPF). Any page returned to the free list is marked as a disk cache page (to be reused later, thereby avoiding a transfer from disk) and a fault on such a page is referred to as a disk cache fault (DCF). A fault on a virtual page not in memory and present only on the disk is called a disk page fault (DPF).

4 Page Replacement Policies

Since replication uses more memory, efficient page replacement policies are important, particularly for applications with a high degree of sharing. Existing NUMA multiprocessors do not address page replacement in detail. We adapt the BSD CLOCK, software LRU approximation, and LRU policies to the NUMA environment. We refer to the adapted LRU policy as LERN, an acronym for LRU Extension for Replicated NUMA memory management. Though, for performance reasons, LRU requires an expensive hardware implementation, since it gives a best-case performance estimate and also LERN takes less simulation time than software LRU approximation, we use the LERN page replacement policy. We will not discuss BSD CLOCK and software LRU approximation further; details can be found in [12].

Policy LERN

In the following discussion, a master page refers to the first physical page allocated to a virtual page. All physical pages allocated thereafter to the same virtual page are referred to as replicas. Any physical page can be in one of the following states: (1) Free: It is not allocated to any virtual page, (2) Replica: It is a replica page used locally, (3) Master.L: It is the master copy and is used locally, (4) Master.R: It is the master copy and is used remotely, and (5) Master.L.R: It is the master copy and is used both locally and remotely. Each processor has a free list (FL), a remote list (RL), and a LRU list (LRUL). FL contains free pages. RL contains master.R pages. LRUL contains master.L, master.L.R and replica pages. Note that for the NR case, LRUL has no replica pages.

LERN for FR works as follows. Each processor reorders its LRUL on each reference. When LERN is invoked, a page can be replaced either from LRUL or RL. If the number of pages in LRUL is greater than a minimum value, the page is replaced from LRUL. Otherwise, the page is replaced from RL. In the former case, master.L.R pages are skipped as long as the number of replica and master.L pages is above a minimum limit. If a master.L.R page is chosen to be replaced, it is transferred to RL after changing its state to master.R, and the replacement policy on RL is invoked. Otherwise, a replica or master.L page from LRUL is replaced. The RL replacement policy replaces the page that is replicated on the least number of processors. FIFO is used to resolve a tie. The first replica of the replaced page becomes the new master and the virtual page is effectively migrated. LERN for NR differs from that for FR in the following respects. There are no replica pages in LRUL. On each reference, each processor reordered the list (LRUL or RL) in which the page currently referenced is located. The reordering is necessary to implement LRU, but is costly. The other option is to use a FIFO ordering for remote references, in which case true LRU is not implemented. The RL replacement policy replaces the page that is used in the least number of remote processors. The page is deallocated and the TLB and page table entries on all processors where it is active are invalidated. The page state transition diagram for FR and NR that explain LERN's interaction with other memory management policies can be found in [12].

5 Performance Metrics

We identify the various types of memory accesses possible and derive expressions for the corresponding memory access times [12]. The time to enforce the consistency model is not included in the above expressions, but recorded by the simulator cumulatively for all accesses. Our key performance metric is the effective memory access time (EMAT), defined as the mean memory access time for the application averaged over all its accesses (first entry of Table 1). The second entry gives an alternate expression for EMAT obtained by substituting the above expressions. The overhead of each factor is given by the third entry.

6 Experimental Results

6.1 Method of Study

We use a trace driven simulator written in Gnu C++ to study the various aspects of NUMA memory management. Parameters to the simulator include various memory management policies and hardware parame-
Table 1. Performance metrics

<table>
<thead>
<tr>
<th>Performance Metric</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMAT</td>
<td>[ \sum_{\text{all access types}} \left{ \frac{\text{Number of accesses of a given access type}}{\text{EMAT}} \times \left{ \frac{\text{Access Time for that access type}}{\text{Total number of accesses}} \right} \right} + \left{ \frac{\text{Consistency time}}{\text{Total number of accesses}} \right} ]</td>
</tr>
<tr>
<td>Factor Overhead</td>
<td>[ \sum_{\text{all factors}} \left{ \frac{\text{Number of times the factor appears in total access time for all accesses}}{\text{EMAT} \times \text{Total number of accesses}} \times \left{ \frac{\text{Time for the factor}}{\text{Total number of accesses}} \right} \right} ]</td>
</tr>
</tbody>
</table>

6.2 Workload

Simulations are conducted with synthetic traces, compiler output-based traces, and actual application traces. Synthetic traces capture different page access patterns, help validate the simulator, and are not discussed further. The compiler output-based trace is for near neighbor computation on a grid of points. The inputs to the trace-generator are: grid size, number of processors, virtual page size, number of grid points per virtual page, number of iterations, and virtual address assignment policy. The trace generated corresponds to a row-wise processing of the grid points by the individual processors. Three virtual address assignment policies are used: column major, row major, and block major. Column major and row major policies assign virtual addresses to the grid points column-wise and row-wise, respectively. Block major assigns virtual addresses one block at a time. A block denotes the portion of the grid assigned to each processor for computation. Within a given block, virtual addresses are assigned row-wise. The behavior of column major and row major are similar, except that the former incurs more page faults because the grid points are processed row-wise by each processor, whereas column major assigns virtual addresses column-wise. Column major is not studied further. We use a 64-point grid processed by 16 processors in our study. The actual application address trace is generated by an Alliant emulator.
for the Particle-In-Cell (PIC) benchmark for an eight-processor configuration. The trace length is approximately 16 million. There are several phases of locality in the references by the eight processors. We assume a Harvard-type architecture with separate instruction and data caches and ignore instruction traces.

6.3 Fixed and Variable Parameters

Detailed experiments on the workload show that fault processor placement is better than modulo placement and also that WU is better than WI to enforce SC. Therefore, experiments relevant to this paper's objective use the fault processor and WU policies. Since we would like to determine the conditions under which replication is effective, we study both FR and NR. Since FR uses more memory, we vary the number of pages allocated to the application. The variation is done so as to cover both the low page range region, where there are a lot of disk page faults and the high page range region, where further increase in the number of pages did not improve the EMAT. Since replication is proposed as a solution to alleviate the performance degradation due to high remote memory access time, we vary the ratio of remote to local memory access times (R) in steps of 2 from 2 to 10. Table 2 list the various parameters, what they represent, their pattern in the overhead graphs to be discussed later, and their values in our study. The kernel lookup time and LERN time are based on the consideration that the kernel pages might be allocated in remote memory. The value of R for the multicast operation that enforces the WU policy is optimistic because it assumes that all remote updates can take place simultaneously, which might not be true always. All values are normalized to the local memory access time $t_{local}$, which is assumed to be 0.5 μs.

6.4 Discussion of Results

6.4.1 Near Neighbor/Row Major

Figures 1 through 4 show the overhead percentage of various factors for R=2 and R=10, as the number of pages is varied in the Near Neighbor/Row Major

Table 2. Factors

<table>
<thead>
<tr>
<th>Factor</th>
<th>What it represents</th>
<th>Overhead graph pattern</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>Local memory access</td>
<td>Black</td>
<td>1</td>
</tr>
<tr>
<td>remote</td>
<td>Remote memory access</td>
<td>White backslash in black</td>
<td>R</td>
</tr>
<tr>
<td>rpfos</td>
<td>Remote page fault kernel lookup</td>
<td>Black crosshatch in white</td>
<td>100</td>
</tr>
<tr>
<td>rlc</td>
<td>Remote to local copy</td>
<td>Closely spaced black backslash in white</td>
<td>560</td>
</tr>
<tr>
<td>dfos</td>
<td>Disk fault kernel lookup</td>
<td>White</td>
<td>100</td>
</tr>
<tr>
<td>dmc</td>
<td>Disk to memory copy</td>
<td>White dots in black</td>
<td>40000</td>
</tr>
<tr>
<td>lern</td>
<td>LERN policy</td>
<td>Horizontal black lines in white</td>
<td>500</td>
</tr>
<tr>
<td>multicast</td>
<td>multicast (write update)</td>
<td>Black dots in white</td>
<td>R</td>
</tr>
<tr>
<td>remwr</td>
<td>Write to a master copy allocated remotely</td>
<td>Widely spaced black backslash in white</td>
<td>R</td>
</tr>
</tbody>
</table>

Figure 3. Percentage Overhead vs. No. of Pages (Near Neighbor/Row Major/FR/R=10)

Figure 4. Percentage Overhead vs. No. of Pages (Near Neighbor/Row Major/NR/R=10)

1These values are typical of the node processor in TC2000
application, for the FR and NR cases. The overhead graphs for FR can be divided into the low page range region, the high page range region, and the transition range region. The transition range region is caused by the LERN overhead that is absent in the high page range region. The insignificance of LERN overhead results in the absence of the transition region for NR.

Disk page faults constitute the major overhead in both the FR and NR low page range regions for R=2. For R=10, remote access and disk page faults share the overhead for NR, whereas disk page faults still constitute the major overhead for FR. In the FR high page range region, disk page faults are the minimum possible. Since our objective is to obtain an EMAT as close to \( t_{\text{local}} \) as possible, we would like the local access overhead to be as high as possible. Therefore, multicast and remote writes constitute the major overhead. They account for a higher major overhead for the higher value of R=10. In the NR high page range region, remote access is the major overhead for both R=2 and R=10. EMAT for FR and NR are shown in Figures 5 and 6, respectively. The steep slope corresponds to the transition from the low page range region to the high page range region. Since FR uses more pages than NR, LERN invocations and RPFs account for EMAT for FR being better than that for FR, in the low page range region, for low values of R. At high values of R, these factors are overshadowed by the remote access overhead in NR. In the high page range region, EMAT flattens after a certain number of pages, the actual number being smaller for NR than for FR. For R=2, EMAT for FR is only slightly less than that for NR. For R > 2, EMAT for FR is better than that for NR. The best values of EMAT occur in the high page range region for both FR and NR and the dominant factors are the remote access for NR and remote master writes and multicast for FR. It is seen that the effectiveness of replication depends on the value of R and the consistency overhead.
6.4.2 Near Neighbor/Block Major

Figures 7 through 10 show the overhead percentage of various factors for \( R=2 \) and \( R=10 \), as the number of pages is varied in the Near Neighbor/Block Major application, for the FR and NR cases. Again, the overhead graphs for FR can be divided into the low page range region, the high page range region, and the transition region. The transition region is caused by the LERN overhead that is absent in the high page range region. The insignificance of LERN overhead results in the absence of the transition region in NR. In the FR and NR low page range region, disk page faults constitute the major overhead for both \( R=2 \) and \( R=10 \). In the FR and NR high page range region, disk page faults are at their minimum and it is desirable for the local access overhead to be high. Therefore, multicast and remote writes are the dominant factors for FR, while the frequency of remote access is the dominant factor for NR, for both \( R=2 \) and \( R=10 \). EMAT for FR and NR are shown in Figures 11 and 12, respectively. The steep slope corresponds to the transition from the low page range region to the high page range region. EMAT flattens in the high page range region in all cases after a certain number of pages; the actual number of pages for NR is less than that for FR. Since block major leads to fewer shared pages, remote access overhead in the NR high page range region is less than the multicast and remote write overhead in the FR high page range region, resulting in a better EMAT for NR than that for FR for both \( R=2 \) and \( R=10 \).

**Row Major versus Block Major**

Row major results in the leftmost processor accessing a given row's virtual page first. Since fault processor placement allocates the page local to the first faulting processor, more pages are allocated local to a given processor. In block major, processors access virtual pages within their block except when processing border grid points and, consequently, have fewer pages allocated locally. Therefore, the transition from...
the low page range region to the high page range region occurs at fewer pages for NR with block major than NR with row major. Remote access overhead for NR is considerably lower for block major than that for row major, resulting in a better EMAT in the entire page range. In essence, block major results in fewer shared pages than row major and therefore, NR is as good or better than FR and uses fewer pages. Therefore, proper virtual address assignment by the compiler or the programmer results in good performance even without replication.

6.4.3 Application PIC

Due to lack of space, we do not include the overhead graphs and EMAT curves for the PIC application; they can be found in [12]. PIC exhibits a high degree of sharing and for high values of R, FR performs better than NR. FR for the PIC application uses more memory than FR for the Near Neighbor/Row Major application, emphasizing the additional memory needs of FR.

7 Conclusions

The experiments discussed in the previous section showed that under certain conditions, NR is better than FR. Further, the effectiveness of replication depends on: (1) the ratio of access times to remote and local memories, (2) virtual address assignment policy, (3) data sharing characteristics of the application, (4) overhead in enforcing consistency, and (5) physical memory available relative to the sharing characteristics of the application.

Our experiments emphasize the need for memory management policies to be application-dependent. Therefore, we feel that characterization of parallel applications will aid in developing memory management policies tailored to typical applications. We also plan to study the effectiveness of dynamic page migration in reducing the remote accesses for NR and remote master writes for FR. The influence of efficient multicast algorithms, hardware coherent caches, and consistency models such as weak ordering, on the consistency overhead needs to be studied. Algorithms that automatically estimate whether FR will be effective, based on application characteristics and the available physical memory need to be developed. We also would like to extend our work to local/remote/global architectures such as ACE and wormhole-routed multicomputers such as Symult 2010 and nCUBE-2.

Acknowledgements

We thank P.C. Yew and Hock-Beng Lim at the CSRD, U. of Illinois at Urbana-Champaign, for providing us with the PIC application address trace.

References


