Performance Improvement of Processors Using Superpages

Abhishek Patel¹, Prabhat Dansena²

¹M. Tech., CSE, India
²M. Tech., School of Computer and Information Sciences, UoH, India

Abstract—In order to improve system performance, many modern processors support superpage technology, which is also called variable page. It improves TLB coverage greatly without increasing TLB size. But superpaging is a great challenge for operating systems. In this paper we implement dynamic variable page technology. The dynamic paging choose proper size page according to the application program address space size. The number of TLB misses can reduced for large programs like large matrix multiplication. The performance of system can be improved by using dynamic variable paging by some number of percentage. The overall result will depend on the simulation process on a system using dynamic variable page sizes as 4KB, 16KB, 64KB etc. Page size depend on the program code ex. small page size for small program code and large page size for a huge computation or instructions code.

Keywords—TLB, superpage, page replacement, thrashing, dynamic paging, page fault.

I. INTRODUCTION

Today’s all modern processors TLB technology for virtual memory, which contains recently used virtual to physical address translations. TLB search the demand address in it’s buffer by parallel manner so the accessing can fast, so it give high accessing speed for CPU. TLB’s are usually fully-associative. As the technology is developed every day, the memory continuously increases as the applications sets are increased. This tendency poses a severe challenge to TLB coverage, which is defined as the amount of memory mapped by the TLB without increasing a TLB miss.

The TLB size has remained small because of the hardware limit. In order to increase TLB coverage without increasing TLB size, superpages are supported by most modern processors. Superpage is composed of the power of 2 base pages (base page is the page whose size is configured when the system starts up) and the virtual address and physical address of the base page should be continuous and aligned. This is the hardware feature modern processors for superpage support.

There are many limitations on superpage as, their size must be a power of 2 multiple of the base page size and they must be continuous and aligned in virtual and physical address space. These limitations pose several challenges to operating systems to use superpages.

II. PROBLEM DEFINITION

(1) Superpage technology is proposed to meet the requirements of the large scale applications, all of which need larger linear space.

(2) For applications with small working sets such as web service, it will not benefit from superpage, and the performance may be reduced because of the extra cost for superpaging.

(3) In this paper they use Static Page Replacement Algorithms for superpaging, but Dynamic paging algorithms are much better than the Static Paging.

III. IMPLEMENTATION

Virtual memory is good in theory, but for its operation to be practical it must be properly implemented. Algorithms responsible for replacing pages in physical memory from the secondary source are primarily responsible for the speed and efficiency of the final system. To operate effectively, the loading of extraneous information must be minimized or completely eliminated. More importantly, information that is swapped out of the physical memory must be chosen carefully. If a page has been removed from memory to make way for another requested Page, but then is immediately requested once again, we say the replacement is thrashing. Thrashing page replacement has the potential to bring virtual memory to an immediate slowdown, since it causes the manager to make redundant memory reads and writes, while relying heavily on the speed of the secondary storage device. Thankfully most thrashing can be avoided naturally, as a program’s scope of operation tends to remain relatively small throughout its lifetime. This idea, the principal of locality, states that program code and data references will most likely not be contiguous, but will reliably cluster in predictable areas. Without the clustering behavior of pages, “predictive non-thrashing algorithms could not function”, aware of these principals, we can begin evaluating the variety of page replacement algorithms.

All paging algorithms function on three basic policies: a fetch policy, a replacement policy, and a placement policy. In the case of static paging, describes the process with a shortcut: the page that has been removed is always replaced by the incoming page; this means that the placement policy is always fixed.
Since we are also assuming demand paging, the fetch policy is also a constant; the page fetched is that which has been requested by a page fault. This leaves only the examination of replacement methods.

IV. Static Page Replacement Algorithms

A. First-In, First-Out (FIFO)

First-in, first-out is as easy to implement as Random Replacement, and although its performance is equally unreliable or worse, claims, its behavior does follow a predictable pattern. Rather than choosing a victim page at random, the oldest page (or first-in) is the first to be removed. Conceptually compares FIFO to a limited size queue, with items being added to the queue at the tail. When the queue fills (all of the physical memory has been allocated), the first page to enter is pushed out of head of the queue. Similar to Random Replacement, FIFO blatantly ignores trends, and although it produces less page faults, still does not take advantage of locality trends unless by coincidence as pages move along the queue.

B. Least Recently Used (LRU)

We have seen that an algorithm must use some kind of behavior prediction if it is to be efficient. One of the most basic page replacement approaches uses the usage of a page as an indication of its “worth” when searching for a victim page: the Least Recently Used (LRU) Algorithm. LRU was designed to take advantage of “normal” program operation, which generally consists of a series of loops with calls to rarely executed code. In terms of the virtual addressing and pages, this means that the majority of code executed will be held in a small number of pages; essentially the algorithm takes advantage of the locality principal.

C. Least Frequently Used (LFU)

Often confused with LRU, Least Frequently Used (LFU) selects a page for replacement if it has not been used often in the past. Instead of using a single age as in the case of LRU, LFU defines a frequency of use associated with each page. This frequency is calculated throughout the reference stream, and its value can be calculated in a variety of ways. The most common frequency implementation begins at the beginning of the page reference stream, and continues to calculate the frequency over an ever-increasing interval. Although this is the most accurate representation of the actual frequency of use, it does have some serious drawbacks. Primarily, reactions to locality changes will be extremely slow.

Assuming that a program either changes its set of active pages, or terminates and is replaced by a completely different program, the frequency count will cause pages in the new locality to be immediately replaced since their frequency is much less than the pages associated with the previous program. Since the context has changed, and the pages swapped out will most likely be needed again soon (due to the new program’s principal of locality), a period of thrashing will likely occur.

V. The Dynamic Variable Page Technology

A variety of methods must be employed to develop replacement algorithms that work hand-in-hand with the locality changes present in complex programs. Dynamic paging algorithms accomplish this by attempting to predict program memory requirements, while adjusting available pages based on reoccurring trends. This policy of controlling available pages is also referred to as “prefetch” paging, and is contrary to the idea of demand paging. Dynamic variable page technology is the superpage technology in general. The superpage allocation strategy, that is how/when/what size superpages to allocate, is the basic strategy for this technology. Dynamic variable page technology can help the program use superpages with different sizes adaptively at run time. This technology is more flexible than the static one and is suitable for most applications. In addition, the switch is added to the codes to control which parts are mapped with superpage. This needs to add controller to mmap( ), brk( ), and expand_stack( ) when the address space is created.

VI. Dynamic Page Replacement Algorithms

A. Working Set Page Replacement (WSR)

Working Set Replacement (WSR) algorithms can either be very simple or extremely complex. Essentially, the most basic algorithms assume that each program will use only a limited number of its pages during a certain interval of time. During this interval, the program is allowed to freely page fault and add pages, growing until the time has expired. When the interval has expired, the virtual memory manager removes all page references unused during the previous interval. We refer to the set of pages used by a program during its previous interval as its working set. For this to work reliably with minimal thrashing, the time elapsed may be dynamically adjusted to provide maximal correspondence with locality changes. These adjustments can be made a variety of ways, but are usually determined as a function of the rate of page faults occurring within the program.
B. Page-Fault Frequency (PFF)

Working set algorithms do not always use a specific time interval to determine the active set. Various page fault frequency (PFF) algorithms can also be used to monitor the rate at which a program incurs faults. This is very similar to modifying the time interval but is not subject to a minimal time for change to occur; page allocation or release may occur rapidly during periods of locality transition, rather than attempting to suddenly minimize the time interval for evaluation to accomplish the same goal. It is these types of dynamic changes that can add complexity to the working set implementation. PFF does have its limitations depending on the application, however. An example program may require unrelated references to a database, causing a large fault frequency. In this scenario, the program would not benefit from keeping the old references in memory. Rapid changes in the fault frequency due to this type of access would result in either wasted page allocation or rapid thrashing with this algorithm, both detracting from its usefulness.

VII. EXPERIMENTAL RESULT

SPECCPU benchmarks are extracted from real applications, which consist of fourteen floating point and twelve integer applications. As a compute-intensive benchmark, SPECCPU is mainly used to measure the performance of processors, memory hierarchy and compilers. SPECCPU can be measure the performance of system with dynamic variable page technology. The systems’ performance with 16KB, 64KB, and 16 MB page size are also measured separately. We assumet the experiment for a matrix multiplication using dynamic superpaging concept on a 16 MB page support Processor, and the base page size is 16 KB can be simulated on a SPEC CPU suit Benchmark and the result (TLB miss rate) after the simulation using variable page size e.g. 16 KB, 64 KB, and 16 MB is shown in the fig. 1.

![Fig. 1: TLB misses using page sizes 16 KB, 64 KB and 16 MB.](image)

As seen in the fig. 1 that the no. of TLB misses is 16 MB page size <64 KB < 16 KB. So the TLB misses ire depend on the page size and the application type.

VIII. CONCLUSION

The performance optimization of TLB is an important part of current processors’ performance optimization. The variable page technology is becoming a mainstream method of TLB performance optimization, which improves system performance by reducing the TLB miss rate. This paper implements and analyses the two different technologies, static variable page and dynamic variable page, which bring a significant performance improvement for a variety of processors. Although the static variable page technology is not flexible, it improves the performance of specific computing applications. Dynamic variable page technology is an inevitable trend as the working sets of applications become larger and larger. We can evaluate the technology on SPECCPU benchmarks and conclude that an average of 15% performance improvement is gained compared with static page technology.
REFERENCES


