Some Aspects of the Dynamic Behavior of Hierarchical Memories

PETER M. FENWICK

Abstract — **In a computer system with a cache memory, the cache is effectively empty following a job switch, leading to a low hit rate and consequently lowered performance until the cache becomes reasonably full. The analysis shows that a job must run for several milliseconds before the average performance approaches that expected from a steady-state analysis. Care may therefore be needed in designing memory systems which have very high page-fault rates from main memory, or which include many levels in the memory hierarchy.**

Index Terms — **Bubble memory, cache memory, dynamic behavior, memory hierarchy, performance degradation.**

I. Hierarchical Memories

An extensive literature exists on the expected advantages of hierarchical memory systems, and many analyses have been published outlining methods of optimizing such a hierarchy. The work described here was prompted by the advent of charge-coupled and bubble memories, which, with an access time of a millisecond or so, showed promise for filling the "access gap" between semiconductor RAM (access about 500 ns) and rotating disks (access 10–30 ms). With such buffer stores included, a typical computer would be expected to have a four-level memory hierarchy as perhaps

Manuscript received September 12, 1983; revised December 12, 1984. This work was supported in part by the University of Auckland, Auckland, New Zealand, and in part by Burroughs Corporation, Mission Viejo, CA.

The author is with the Department of Computer Science, University of Auckland, Auckland, New Zealand.

100 ns cache	1000 words
500 ns RAM	1 million words
1 ms buffer	10 million words
20 ms disk	100 million words.

The sizes of the different levels may be subject to adjustment if the hierarchy cost/performance is optimized according to the extant optimization procedures [1]–[3], but this does not affect the present argument.

II. The Dynamic Behavior of Cache Memories

Many studies have been done on the behavior of cache memories, but almost all of these studies are concerned only with the steady-state performance. Very few authors have paid any attention to the dynamic behavior of a cache, particularly as occurs on a major change of program environment such as a job switch by the operating system. The cache is then initially empty as far as the new environment is concerned and accumulates relevant new data only as a result of "misses." The miss rate is initially high, but as the cache fills, the miss rate falls, and the rate of filling also drops. A considerable time may then be needed for the cache to fill and reach its expected performance. The behavior is in many ways analogous to the charging of a capacitor through a resistor—the cache is "charged" with data at a rate which decreases as the cache fills. Although the analogy is not exact, a cache of 2000 words receiving 2 million requests per second may be expected to fill in accordance with a "time constant" of about 1 ms, reaching 99 percent of its steady-state filling in about 5 ms. While the cache is filling and operating at a low hit rate, the processor performance will be below the value expected from the steady-state cache behavior. Reasonable performance can be expected only if the cache is allowed to fill, i.e., if the processor is able to run for several milliseconds on each job. It is essential to distinguish between the instantaneous processor performance, which is a function of the hit rate at that instant, and the average performance since the job switch. The average performance is dominated by the initially low performance with the cache nearly empty, and approaches the equilibrium value far more slowly than does the instantaneous performance.

A question which must be raised here concerns the relevance of address traces to studies of cache behavior. In a multiprogramming environment the address trace applies only for a time less than the page-fault interval, time slice, or other interruption to a running program. A trace of tens of thousands of memory references may be of doubtful relevance in some contexts.

III. The Optimization of Memory Hierarchies

Procedures for the optimization of memory hierarchies depend on two basic facts. The first is that faster memories are more expensive than slower memories, and the second is that a larger memory will have a lower page-fault rate, or its equivalent, than a smaller memory in an otherwise similar hierarchy. At the main memory level a large memory will have a low page-fault rate, but will be expensive. Reducing its size will reduce the cost but increase the fault rate. If the fault rate is within the capacity of the next level (buffer or disk) the effect will be a gradual reduction in performance as the fault rate increases with smaller memory and an improvement in the ratio of cost to performance. When the fault rate approaches or exceeds the capacity of the disk or buffer, the performance falls off substantially as the system "thrashes." The optimum memory size may then be expected to be that which gives a fault rate approaching the capacity of the next level; a larger memory costs more, and a smaller one thrashes.

If a bubble memory buffer has an access time of 1 ms, the page-fault interval should not be more than about 2 ms for a cost-effective configuration. Now this page-fault interval is comparable to the cache filling time, and the processor might never be able to achieve its optimal steady-state performance before a page fault occurs.

IV. Performance Analysis

In one of the few papers describing the dynamic behavior of caches, Pohm, Agrawal, and Monroe [4] give a relation for the miss rate of a partly full cache and use this to develop a model for the dynamic behavior of the cache. They give results for the variation of hit rate with time, but do not extend this to a prediction of system performance. The present analysis uses their model, which assumes initially that the cache miss ratio (MR) is approximated by the expression

$$\mathrm{MR} = \frac{a0}{fw + a1}$$

where w is the cache size, f is the "filling factor," and $a0$ and $a1$ are constants. The two constants are readily calculated from the steady-state MR (or hit ratio), noting that an empty cache has an MR of 1. (This last shows that $a0 = a1$.) Many caches use multiword blocks to provide some measure of "prefetching" and a higher final hit rate. The model [4, eq. 9] allows for blocks by simply multiplying the filling rate by the block size. This assumes that all prefetched words are actually used and gives a rather optimistic filling rate, but in the absence of any better information, it is probably a reasonable approximation.

Pohm *et al.* give both the differential equation of the filling factor and its analytical solution as a time function; for a numerical solution it has been found preferable to use an iterative solution to the second equation to avoid initial-value problems with the original differential equation. The calculation then proceeds by calculating in succession the filling factor, the miss rate, the effective memory cycle time, and then the time between successive memory requests, assuming a constant processor "think" time. Integrating this last time gives the total elapsed time, which, divided into the corresponding number of memory requests, gives the average memory request rate (i.e., system performance) as a function of time. Calculations were done for a system with the following parameters:

1024	word cache
50 ns	cache access time
500 ns	memory access time
200 ns	average processor think time
85 percent	final cache hit rate
1 or 4	words per cache block.

Results for the calculation are shown in Fig. 1 (cache hit rate) and Fig. 2 (average processor performance), both plotted as a function of time since the job switch. The steady-state memory request rate is 3.15 requests per microsecond, and the system is arbitrarily assumed to have reached the steady-state behavior when it reaches 90 percent of this value, or 2.83 requests per microsecond. With one-word blocks, equilibrium occurs after about 4 ms, while with four-word blocks it occurs in about 1 ms. While the improvement with multiword blocks is quite dramatic, the earlier reservations on the model should be remembered: the improvement might not be this great, and 2.0–2.5 ms might be a more realistic value. A comparison between the figures shows that the instantaneous hit rate approaches the final value much more quickly than does the average performance: this is in accordance with the earlier observation on instantaneous versus average performance.

From these results it is obvious that a page-fault interval of 1 or 2 ms, such as might be thought optimum with a bubble memory buffer, can lead to a performance degradation of 15–25 percent with single-word blocks in the cache. The main benefit from using multiple-word cache blocks might in fact be in the faster approach which they give to final cache equilibrium, rather than in improvements in the cache hit rate and steady-state performance.

V. Buffer Memories and the Operating System

In most cases, a page fault will result in action by the operating system to process the fault and to resume or initiate some other job. An entry to the operating system is a major context change and

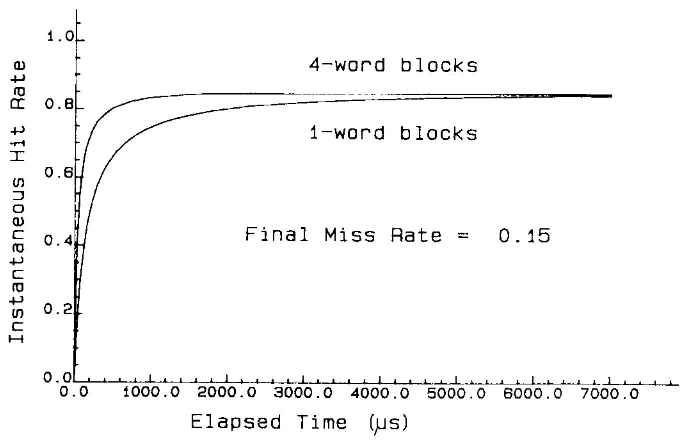

Fig. 1. Hit rate after context change.

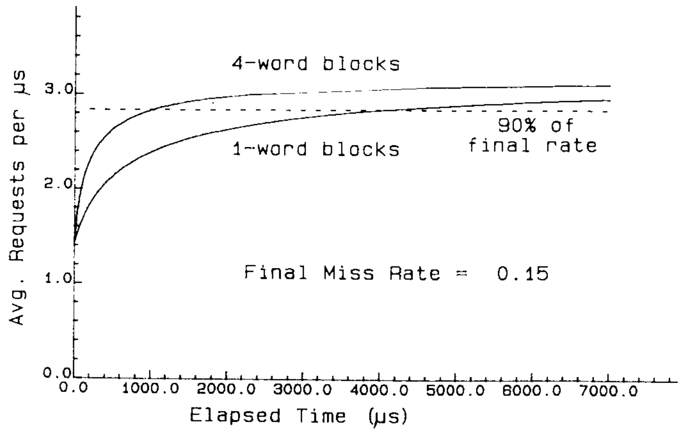

Fig. 2. Cumulative average performance after context change.

may cause the processor to operate inefficiently with a low cache hit rate during much of the page-fault servicing. Furthermore, the fault servicing may well take several hundred microseconds, which is a very large portion of the time between page faults if the page fault is "matched" to the buffer access capacity. This leads to the separate and obvious conclusion, not related to cache dynamic behavior, that a computer which expects to use a high fault rate, as can be done with bubble memory, should have as much of the page-fault and job-resumption logic as possible implemented in hardware/firmware to minimize operating system overheads. As a possible alternative there may be some benefit in using a very large cache, in the expectation that a reasonable amount of operating system information will remain between page faults, thereby reducing the time spent in servicing the fault. Note though that a very large cache might not have time to fill between context changes unless it has a correspondingly large block size. Therefore, a large cache might not be cost effective, as much of its capacity may remain unused in practice.

VI. Use of Buffer Memories

Most emphatically, the preceding discussion does not say that there is no place for a fast disk buffer memory within a hierarchical memory. Rather, it is intended to point out possible interactions between the buffer and cache if the design does not allow for the dynamic behavior of the system. There does seem to be a definite place for a fast disk or paging buffer within a memory hierarchy. Although it seems unwise to design a system with a very high deliberate page-fault rate, there will often be a high fault rate when a program is being initiated or resumed, or when it enters a major new region (perhaps a subroutine). A fast buffer connected as a "disk cache," and loaded with a working set of hopefully related pages at the first page fault of the context change, should be able to satisfy some of the following faults at buffer speed rather than disk speed, with a consequent improvement in system performance. The improvement should be especially significant for brief executions of large programs (such as short compilations) for which the time spent in loading pages can be 10 or 100 times the actual processing time.

VII. Conclusions

It is apparent from the above results that under appropriate conditions there may be considerable interference between a cache and a disk-buffer memory, particularly if a cache of relatively low performance is combined with a high-performance buffer. While a buffer memory does in principle allow a very high page-fault rate, this is a potentially inefficient mode of operation because of both the cache effects and the extra overheads from frequent calls on the operating system. Buffer memories should be used rather to decrease the average access times to disks during program startup or other times of inherently high page-fault rate.

Although fast disk buffers have not in fact been introduced to the extent that was often predicted when this work was done in 1979, the results presented here are still relevant. The fact that a cache takes a considerable time to fill, and that it has a low performance while it is filling, must be remembered by any designer of a multilevel memory system. As processor speeds continue to increase, some designers are considering the use of a small very fast cache in an attempt to match processor memory-request intervals of 10–20 ns to conventional cache speeds. The points raised here may be applicable to such a design.

Addendum

Two recent papers by Clark [5] and Strecker [6] relate to the subject of this paper. Clark studies cache behavior as measured on a real system under a real workload and includes an extensive discussion of most aspects of cache performance in the light of his observations. One conclusion notes the difficulty of relating address traces to the memory access patterns of real processors.

Strecker deals with the transient behavior of a cache, again supported by actual measurements. He does not deal with the real-time performance, however, measuring time only in terms of processor requests, but does develop a model for cache behavior which should be superior to that of Pohm *et al.*

The modeling program used in this paper has been revised to use both the Pohm model and the newer one of Strecker. The results for the two models are in excellent agreement. Using Strecker's values for a static miss rate of 0.05, the two models differ by less than 2 percent in the value for the average processor performance (Fig. 2 above).

Acknowledgment

The work for this correspondence was done while the author was on study leave from the University of Auckland and employed by Burroughs Corporation (Mission Viejo Plant). The author thanks both of these organizations for financial support and for the opportunity to undertake the work reported in this paper.

References

[1] C. K. Chow, "Determination of cache's capacity and its matching storage hierarchy," *IEEE Trans. Comput.*, vol. C-25, pp. 157–164, Feb. 1976.
[2] J. S. MacDonald and K. L. Sigworth, "Storage hierarchy optimization procedure," *IBM J. Res. Develop.*, vol. 19, no. 2, pp. 133–140, Mar. 1975.
[3] T. A. Welch, "Memory hierarchy configuration analysis," *IEEE Trans. Comput.*, vol. C-27, pp. 408–413, May 1978.

[4] A. V. Pohm, O. P. Agrawal, and R. N. Monroe, "The cost and performance tradeoffs of buffered memories," *Proc. IEEE*, vol. 63, pp. 1129–1135, Aug. 1975.
[5] D. W. Clark, "Cache performance in the VAX-11/780," *ACM Trans. Comput. Syst.*, vol. 1, no. 1, pp. 24–37, Feb. 1983.
[6] W. D. Strecker, "Transient behavior of cache memories," *ACM Trans. Comput. Syst.*, vol. 1, no. 4, pp. 281–293, Nov. 1983.