Reprinted from IEEE TRANSACTIONS
ON *COMPUTERS*
Volume C-23, Number 4, April 1974

pp. 346-351
PRINTED IN THE U.S.A.

An Investigation of Alternative Cache Organizations

JAMES BELL, DAVID CASASENT, MEMBER, IEEE, AND C. GORDON BELL, MEMBER, IEEE

Abstract—An investigation of the various cache schemes that are practical for a minicomputer has been found to provide considerable insight into cache organization. Simulations are used to obtain data on the performance and sensitivity of organizational parameters of various writeback and lookahead schemes. Hardware considerations in the construction of the actual cache-minicomputer are also noted and a simple cost/performance analysis is presented.

Index Terms—Cache, mapping, memory, minicomputer, simulation, storage.

Manuscript received May 31, 1972; revised April 10, 1973 and September 10, 1973.

J. Bell and C.G. Bell are with the Digital Equipment Corporation, Maynard, Mass.

D. Casasent is with Carnegie-Mellon University, Pittsburgh, Pa. 15213.

I. INTRODUCTION

USING computer simulations, a detailed analysis of various cache schemes in a PDP-8 minicomputer was undertaken to determine the feasibility of producing a minicomputer

about 5 times faster than existing machines with a minimal cost increase. As a first step, various cache schemes used in larger machines were restructured to fit within the framework of a minicomputer.

Because of the minicomputer's simplicity, perturbations in cache organization and parameters could easily be controlled. Such an investigation thus provides more insight into the direction and extent of performance changes, the reasons for these changes, and a more obvious hardware/cost tradeoff than is possible in other cache analyses.

II. Caches: What and Why?

Since the overall speed of a system is usually limited to its slowest element, the necessity of a fast memory for a fast processor is apparent. A primary memory using semiconductor storage of speed comparable to the processor is precluded by its relatively high unit cost which limits the system's cost/performance ratio.

The suitability of the cache memory concept previously used in large computers [1]-[4] was thus investigated. A cache memory is a fast buffer memory between the processor and the primary memory. The fundamental idea in such an organization is that by keeping the most frequently accessed data in the fast buffer memory (the cache), the average access time per datum is drastically reduced. Although the cache is usually only a small fraction of the size of the main memory, a large fraction of the address requests will be filled by the fast buffer memory (without resorting to accessing the slower main memory) due to the nonrandomness [3] of consecutive memory addresses.

This paper discusses various cache organizations which would capitalize on this phenomenon in the context of a specific minicomputer. Ideally a cache/primary memory pairing approaches the cache in average speed and the main memory in cost.

The following three sections outline several cache organizations, describe the simulation for various programs, and discuss the resultant data.

III. Basic Cache Organizations

The conceptually simplest form of cache organization is called pure associative or content addressable. It is based on an associative memory, i.e., an unordered memory which, when given a content value (in this case a label), returns data "associated with" the label. This is unlike the conventional explicitly addressed random-access memory and is significantly more expensive.

A cache can be thought of as a front end to the primary memory since every memory request is interpreted by the cache. Using association on each memory reference, the labels of all cells of the pure associative cache are tested simultaneously to see whether they match the address sought. If a match is found, the corresponding datum is read or written as needed. In this case, no reference need be made to the primary memory. If no match is found, the request must be passed on to the main memory. A cache with an associative memory organized in this way is discussed by Lee [5].

Under some sets of circumstances specified by the designer

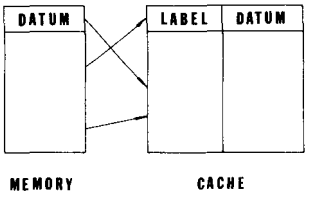

Fig. 1. Pure associative cache.

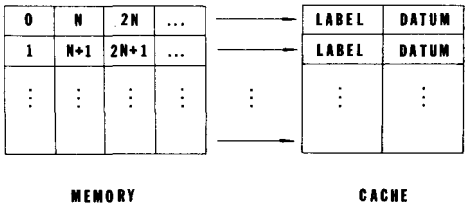

Fig. 2. Direct mapping.

of the cache, a label-datum pair is displaced from its cache cell to make room for a pair which is needed and not presently in the cache. The decisions as to when and in what manner to make these replacements are key organizational parameters of a cache. A sample strategy might be to replace cells of the cache in round-robin order whenever memory requests are passed to the primary memory.

The pure associative cache needs associative storage since any of M data words in the primary memory might be mapped onto any of N cells of the cache, as shown in Fig. 1. Let us assume instead that some fixed mapping $h(k)$ exists by which each address of primary memory maps onto one of N equivalence classes. One can then insist that word k of primary memory be in the specific cache cell $h(k)$, if it is in the cache at all. In this manner, the need for associative hardware can be eliminated.

A particularly natural mapping is to let $h(k)$ be K modulo N, where N is chosen to be a power of 2. In this case, we can use the low order $(\log_2 N)$ bits of k to determine which cache cell to examine. This "direct mapping" scheme is shown in Fig. 2. Consider the M words of primary memory to be arranged as an array of N rows and M/N columns. Direct mapping then means that row k of primary memory maps only onto cell k of the cache.

Another useful feature of direct mapping is its economy on the number of bits of label in the cache. In a pure associative cache any word can be in any cache location and the label must be the full primary memory address. However, in direct mapping, the low-order bits of the label are the same as the cache address and there is no need to store them explicitly. The label part of each cache cell in direct mapping need only be $(\log_2 M/N)$ instead of $(\log_2 M)$ bits wide.

Direct mapping and pure associative mapping are both special cases of a more general organization known as set associative, in which there exists a one-to-many mapping $H(k)$ of each memory address into some subset S of the N equivalence classes. When the cardinality of S is identically 1 or N, we have the cases of direct mapping or pure associativity, respectively. Intermediate values of S lead to hardware structures in which an intermediate amount of associative hardware is needed to do comparisons within the class S.

The addressing speed obtainable with the direct mapping scheme and its extremely low hardware overhead make it and its variants most attractive for use in a minicomputer. However, even within the class of direct mapping schemes, there are too many organizational parameters to vary experimentally in hardware. A software simulation was therefore undertaken.

IV. Simulation Programs

A software simulator for various cache organizations was the main tool used in the cache design. "Traces" or sequences of typical memory address requests to the primary memory by the processor during program execution were collected. For ease of data collection, the program execution was simulated on a larger computer rather than done directly.

To collect traces reflecting typical minicomputer usage, three programs were chosen to generate traces of about one million addresses each. One program was a numerical computation in FOCAL, a widely used interactive interpreter. The second was the fast Fourier transform (FFT), the most commonly requested program in the DECUS users' library. The final program used was the PDP-8 assembler (PAL) assembling a short program.

Once collected, these traces were used as input to the cache simulator. Rather than varying all parameters simultaneously, the effects of each parameter were isolated as much as possible. Each simulation run was designed to answer a specific question, e.g., how does performance vary with cache size? A constant assumption is that the cache was ten times as fast as the primary memory, with one hundred ns and 1-μs cycle times, respectively. Another constant assumption suited to minicomputers is that only one word of memory at a time can be accessed or transmitted over a bus.

V. Simulation Results

The first topic investigated via simulation was the effects of different policies for writing into primary memory. In the simplest policy, known as "write-through (WT)", if there is a write to a given address, that word in primary memory is updated even if there is also a copy of that word in the cache. In addition to conceptual simplicity and ease of implementation, WT has the advantage that obsolete information is never present in main memory. This can be particularly valuable in systems with independent input/output paths or multiple processors.

The simulation results for a cache with WT are displayed in Fig. 3. For uniformity, all results shown are for the FOCAL program; the results for the FFT and PAL programs are substantially identical. In the graphs, the various cache sizes are shown on a logarithmically scaled abscissa, while the ordinate is the effective cycle time that results. The log of the cache size is seen to be roughly linear with effective cycle time over the range shown; doubling the cache size increases the effective speed of the memory by about twenty percent.

An appropriate figure of merit α for a cache computer is defined as the ratio of the execution time of a program run on a conventional machine to the execution time on a cache machine. The effective or average cycle time for the cache machine (the ordinate in Figs. 3-6) is the cycle time of its slow primary memory divided by α. From Fig. 3, a 512 word cache with WT is seen to provide an index of performance α of 2.2, where a machine with no cache is defined to have an α of unity.

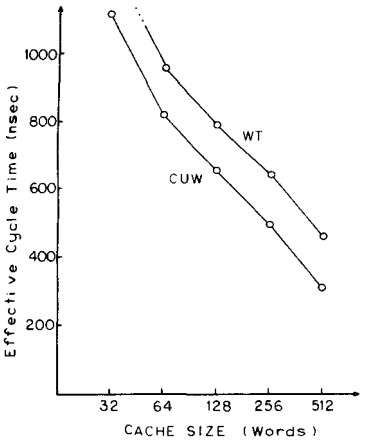

Fig. 3. Speed versus cache size, write through (WT) and conflicting-use writeback (CUW).

The average cycle time of a cache/primary memory pair can be thought of as a weighted average of the speed of the cache and the speed of the primary memory, the weighting factors being functions of the size and organization of the cache. To gain maximum advantage from the cache, the fraction of references to primary memory must be minimized. This points up the weakness of the WT approach, where the fraction of references to primary memory asymptotically approaches the fraction of writes (rather than zero) as the cache size is increased. More sophisticated writeback schemes must thus be examined.

The circumstances responsible for most of the "unnecessary" writes that occur using WT can easily be characterized. During the time a word resides in the cache, it may be updated several times; however, as long as the word remains in the cache, it does not matter whether the copy in primary memory is out of date, since requests for the word are filled by the cache. Thus, it is only when the word is ultimately displaced from the cache by a different word that an accurate copy need be rewritten into primary memory.

A write-back method which does just this was thus devised. A primary memory word is now accessed if and only if that word has no counterpart in the cache. Whenever a word of primary memory is read or written, that word is also copied into the cache, thereby displacing some other word whose cache position conflicts with its own. The word displaced from the cache is then written back into primary memory. This scheme will be called "conflicting-use writeback (CUW)".

CUW offers a significant performance increase over WT, as shown in Fig. 3. For example, with a 512 word cache, WT takes over forty percent longer on the average. The index of performance CUW is 3.2 compared to 2.2 for WT. (All comparisons of α will assume a constant cache size of 512 words.)

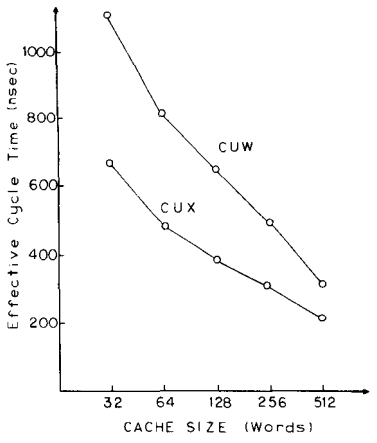

Fig. 4. Speed versus cache size, conflicting-use writeback without X bit (CUW) and with X bit (CUX).

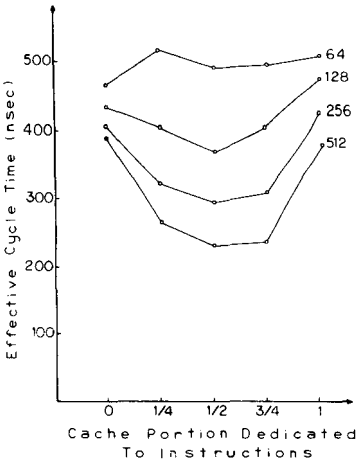

Fig. 5. Speed versus cache dedication for 64, 128, 256, and 512 word caches using CUX writeback.

This reduction in the average number of writes to primary memory per change in cache occupancy is one effective way of improving performance. The shift from WT to CUW lowered this number to exactly 1. But a further lowering is still possible in many cases. Consider a certain address in memory which is sometimes read but never written (most instructions fall into this category). Such a word will eventually move to the cache, and will eventually be displaced from the cache. But even when displaced, it need not be written back into primary memory, since it has not been changed. To implement this improvement, an extra X bit can be added per cache cell for bookkeeping purposes to record any change to the occupant. The X bit is reset to zero for each new occupant, and is set to one by any updating of that cell. The results of simulating UW with (CUX) and without (CUW) an X bit are shown in Fig. 4, from which the effective cycle time with a 512 word cache memory is seen to decrease from 317 to 207 ns when the X bit is included. Stated differently, the performance is increased by 50 percent, from an α of 3.2 to an α of 4.9. This 50 percent performance increase more than compensates for the hardware costs of including the single extra bit.

Some overlap of the CUX operations can occur. For instance, the cache word can be written immediately following the access time of the primary memory, not at the end of its cycle. In large computers many overlapped operations occur and an in-depth cache analysis is difficult.

Thus far, the cache memory has been assumed to be composed of homogeneous cells. But conceivably a functionally specialized partitioning of the cache could give higher performance. For example, perhaps a cache devoted exactly half to instructions and half to data would be more effective than a homogeneous one; alternatively, one that holds just instructions could be better than one holding just data. To test these hypotheses, the effects of dividing the cache into sections dedicated to specific uses were investigated.

Simulated caches of varying sizes were divided into quarters and each quarter was devoted exclusively to instructions or to data. Fig. 5 shows how performance using the CUX writeback scheme varies with the proportion of the cache devoted to data. Typically, the best performance occurs with half of the cache devoted to instructions and half to data. However, the index of performance of 4.4 for this case is somewhat worse than the 4.9 value found for the corresponding 512 word homogeneous cache. As shown in Fig. 6, the performance of the best dedicated cache CUXD (half allotted to instructions and half to data) in general is quite similar to that of a homogeneous cache (CUX); the extra complexity of dedicated cache control is thus not justifiable.

The caches found on large computers differ from those discussed here in that each of their cache cells contain multiple words which are moved as a block to and from primary memory. The performance increase derived from such caches can be attributed to two conceptually different causes called lookback and lookahead. Within loops, cache benefits are due to reencountering previously used words in the cache (i.e., lookback) while in straight line code, cache benefits are due to lookahead. Since only single word blocks have been discussed thus far, lookahead has played no role.

This restriction was now removed and the cache simulator was modified to include a type of lookahead while retaining single word cache cells. Every time a word was brought into the cache from the primary memory, the next consecutive word was brought into the next consecutive cache cell at the same time. Fig. 6 depicts the value of this lookahead (CUXL) relative to a cache without lookahead (CUX). This lookahead does result in an improved α of 6.0 (an indication that, in general, consecutive memory locations are more likely to be referenced) but once again at the cost of added complexity in the cache control. Furthermore, it requires a double width, slow main memory. Such a scheme was not considered for the machine constructed as it would have required a new slow memory configuration, new bus and pin assignments, and eliminated compatability with existing machines. The use of lookahead only on instructions and not data, was also examined: this proved to be about 4 percent less effective than the use of lookahead on both.

In summary, various organizations designed to enhance the

performance of a cache/primary memory pair as far as possible along the continuum from pure primary memory performance ($\alpha = 1$) toward pure cache performance ($\alpha = 10$) have been simulated. For the organizations simulated, a cache of 512 words can reasonably be expected to yield an α in the range of 3 to 6, depending on the complexity of the cache control which can be tolerated.

VI. Hardware Considerations

Various hardware considerations have already been alluded to. The general interfacing problems and the hardware tradeoffs involved in the various schemes will now be considered. Fig. 7 shows the basic timing diagram for a fast cache memory. Three time points are noted for reference. At T_1 the fast memory timing begins. At T_2 the label-datum pair is obtained from the memory. By T_3 (which is actually T_1 of the next cycle) the next memory address (MA) location and input data (DI), if a write cycle, must be loaded. If the word is an instruction, this requires that the decoding as well as the computational operations be performed in a critical $T_3 - T_2$ time. Any computation, shifting, incrementing, etc., requiring the accumulator (AC) register can be overlapped with the next cycle by loading the AC at T_2 of the next cycle, thereby overlapping instruction execution as much as possible. Fast bipolar memories have $t_{cycle} \simeq 2t_{access}$. For the cache memory actually implemented, access time is 60 ns and cycle time is 110 ns. *S* series TTL devices were used in all critical paths to achieve the required speed.

The choice of a direct mapping cache scheme added very little in hardware, essentially one 5-bit comparator between the label and the higher order MA bits, (for a 512 word cache-16*K* word primary core memory system). Any cache scheme requires timing loops within the CPU control, one controlling the fast cache memory and the other the slow primary memory. Their starting and the switching between them are controlled by a cache control unit (CCU) which is essentially the above 5-bit comparator. A two bus system is another practical requirement in any cache scheme: a fast bus on which the fast memory is mounted, and a slow bus to which the slow primary memory and all peripherals interface. The CCU controls the interfacing of these two buses and the central processing unit (CPU).

The 50 percent performance increase obtainable with the CUX writeback scheme greatly encouraged its implementation. This was rather easily incorporated by feeding the *X* bit of the accessed cache word, the status of the CPU's read/write flip-flop, and the result of the comparison to a logic network within the cache control. This logic then controls the sequence of fast memory and slow memory reads and writes to be generated, thus controlling the two timing loops and their interconnection.

The majority of solid-state memories evaluated had latched outputs, so that after a three cycle read-write-write sequence the original datum read out three cycles before is still present at the memory's data-out lines. This enabled some simplification in the flow chart. The nondestructive readout feature of a solid-state memory necessitated a preclear of the fast memory when power was turned on; this was rather easily implemented.

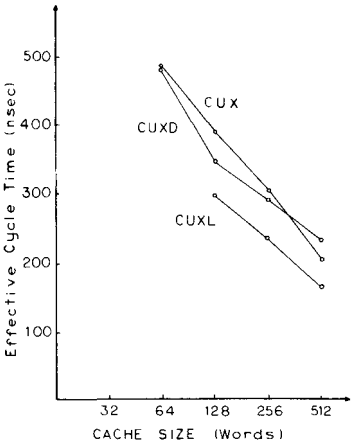

Fig. 6. Speed versus cache size, CUX writeback with lookahead (CUXL), CUX writeback scheme with best dedication (CUXD).

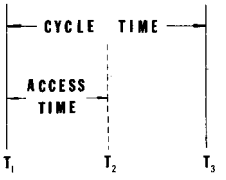

Fig. 7. Simplified cache timing.

An added feature of the direct mapping cache scheme chosen is the ease with which the number of words of fast cache memory can be increased for users requiring a larger slow primary memory and/or a yet faster average system speed. The user need only purchase as much fast cache memory as his speed needs require and may add on more later in much the same way that core is added presently. The increase in cache size merely changes the label length and thus the number of bits that need be compaired. By fixing the label at 5 bit, a wide variety of cache/slow memory sizes can be handled.

From a cost/performance standpoint, the increased parallelism and speed of the CPU increases its cost by 50 percent due to the added boards, wiring and increased IC count. This increase plus the cost of a 512 word X 18 bit solid state memory imply a price differential of certainly no more than $5000. Even with such conservative assumptions, an interesting cost/performance comparison results between the minimum PDP-8/E system, the average PDP-8/E system, and a time-shared PDP-8/E system with and without a cache scheme and faster processor. As shown in Table I, the performance/cost ratio of the designated system can be as much as 4.3 times greater than that of a conventional PDP-8/E.

Conclusion

The origin and explanation of the speed improvements produced by variants of a basic cache scheme have been discussed. In the context of minicomputers, somewhere

TABLE I
PERFORMANCE COST COMPARISON

Configuration Costs (K$)				Performance	Performance/Cost Ratio Compared to 8/E		
	Minimum	Average	TSS	α	Minimum	Average	TSS
8/E	5	10	35	1.0	1	1	1
8/E with cache	10	15	40	5.0	2.5	3.3	4.3

between a performance increase factor of 5 and 6, a breakpoint occurs. The implementation of a cache using conflicting use write adds little to the hardware costs of a fast processor and improves performance by a factor of 3.2, while the addition of an X bit raises performance by an additional 50 percent to an α of 4.9 with only a small hardware increase. Any further variants provide only minor performance improvements at more substantial hardware costs. Therefore a reasonable compromise between performance and cost in a minicomputer is to use a simple cache scheme and aim for about 90 percent hits ($\alpha = 5$) in the cache, rather than the higher hit ratios sought in larger machines.

ACKNOWLEDGMENT

The authors also wish to thank the assistance of R. Hamel and C. Kaman of Digital Equipment Corporation during this project. The simulation was performed by W. Corbin at Carnegie-Mellon University.

REFERENCES

[1] C.J. Conti, D.H. Gibson, and S.H. Pitkowsky, "Structural aspects of the system/360 model 85, I.–General organization," *IBM Syst. J.*, vol. 7, pp. 2-14, 1968.
[2] G.G. Scarott, "The efficient use of multilevel storage," in *1965 Proc. IFIP Cong.* Washington, D.C.: Spartan, p. 137.
[3] J.S. Liptay, "Structural aspects of the IBM system/360 model 85, II.–The cache," *IBM Syst. J.*, vol. 7, no. 1, pp. 15-21, 1968.
[4] D.H. Gibson, "Consideration in block-oriented systems design," in *1967 Spring Joint Comput. Conf., AFIPS Conf. Proc.*, vol. 30. Washington, D.C.: Spartan, pp. 69-80.
[5] F. Lee, "Study of 'look-aside' memory," *IEEE Trans. Comput.*, vol. C-18, pp. 1062-1064, Nov. 1969.

James Bell received the B.A. degree in mathematics from Dartmouth College, Hanover, N.H., in 1964 and the M.S. and Ph.D. degrees in computer science from Stanford University, Stanford, Calif., in 1966 and 1968, respectively. His doctoral thesis there involved the design of *Proteus*, a minimal extensible computer language.

He has previously held positions with Bell Labs, IBM, Stanford Research Institute, and Control Data Corporation, and has taught at Stanford and Northeastern Universities. His specialty is programming languages and compilers, an area in which he has published several papers. He is currently Manager of the Research and Development Group, Digital Equipment Corporation, Maynard, Mass.

David Casasent (S'58-M'69) received the Ph.D. degree in electrical engineering from the University of Illinois, Urbana, in 1969.

From 1964 to 1969 he was a Research Assistant in the Digital Computer Laboratory, engaged in research in hybrid analog and digital systems and optical techniques and systems for information processing. He is currently an Assistant Professor at Carnegie-Mellon University, Pittsburgh, Pa., and a Consultant to Digital Equipment Corporation and Battelle Research Laboratories. His research work involves computer hardware, radar processing, electron optics, electro optics and optical data processing. He is President Elect of the Optical Society of America, Pittsburgh Section, and past Chairman of the IEEE Electron Devices Group, Pittsburgh Section. He is the author of two texts on analog and digital electronics and over 30 technical papers.

Dr. Casasent is a member of Optical Society of America, SID, HKN and ΣT.

C. Gordon Bell (M'66) was born in Kirksville, Mo., on August 19, 1934. He received the B.S. degree in electrical engineering in 1956 and the M.S. degree in 1957, both from Massachusetts Institute of Technology, Cambridge.

In 1959 he was with the Speech Communications Laboratory at the M.I.T. Division of Sponsored Research. From 1959 to 1960 he was a Research Engineer with the Electronic Systems Laboratory at M.I.T. From 1960 to 1966 he was Manager in charge of computer design at the Digital Equipment Corporation, Maynard, Mass. He is on leave as Professor of Electrical Engineering and Computer Science at Carnegie-Mellon University, Pittsburgh, Pa. He is presently Vice President of Engineering, Digital Equipment Corporation, Maynard, Mass. His research interests include general systems design and design management, design of multiple processor computer systems, for either parallel or multiprocessing, design automation design, documentation structure including storage retrieval, applied artificial intelligence, and computer applications of most types.

Mr. Bell is a member of the Association for Computing Machinery and Eta Kappa Nu.