Critique of

"A Low-Overhead High-Performance Unified Buffer Management Scheme that Exploits Sequential and Looping References"

by Kim, Choi, Kim, Noh, Min, Cho and Kim

November 21, 2000

1 Introduction

This paper introduces a buffer management strategy which the authors name *Unified Buffer Management (UBM)*. It divides the buffer cache into three areas, used for blocks which are determined to be being accessed sequentially, in a 'looping' (recurrently sequential) fashion, and otherwise.

The third, miscellaneous, area is managed using traditional replacement algorithms. The second is better served by a period based scheme, which replaces blocks in decreasing order of loop periods. Blocks with the same loop period are simply managed via a Most Recently Used (MRU) algorithm. Blocks in the first area are not expected to be accessed again, so a simple MRU algorithm is used there as well.

Additionally, the authors propose a method for dynamically dividing buffer space between the areas, based on the notion of marginal gains. This assigns each buffer to the partition that will theoretically provide the greatest increase in hits if it were to have that buffer. This is obviously a very sensible scheme, which dynamically adjusts and does not need parameters to be chosen in advance.

I was impressed with this paper, certainly more than the other paper that we were asked to review (Chiueh, Venkitachalam and Pradhan). This paper is well-structured and easy to read, with relevant diagrams throughout.

Mathematical expressions for theoretical hit rates are derived, to allow the marginal gains to be calculated correctly.

A good amount of benchmarking data is provided, based on both trace-driven simulations and an actual implementation on the FreeBSD operating system. The related work is summarised well at the beginning, but also considered throughout the paper, direct comparisons being drawn with such schemes as 2Q and Application Controlled File Caching (ACFC). In the latter case, that author's own traces are used.

2 Worth a Thousand Words

The diagrams and graphs in this paper are certainly excellent, and provide considerable insight into the cache behaviour. The numerous Inter-Reference Gap (IRG) graphs serve to illustrate how looping blocks are important, and that UBM comes much closer to the theoretical best-case - the offline optimal (OPT) scheme - in handling them.

The space-time reference scattergrams also enable excellent visualisation of the different types of references. In particular Figure 7 shows that the classification strategy is effective, by separating the space-time scattergram into the components seen by UBM (although the boundary of the 'other' references is dense on the graph, so it may be that some of those references could be considered sequential references).

Figure 10 graphs marginal gain as a function of size for each area, and also the size of each area with time. It thus shows how the dynamic buffer allocation works - in essence it equalises the marginal gain in the 'looping' and 'other' areas, since if the marginal gain was greater in one area, it would make sense to give it blocks from the other. Interestingly, for the benchmark shown, the situation stabilises with the area for looping references being considerably larger than that for other references, supporting the idea that the distinction is worthwhile.

3 Well Chosen Benchmarks?

We always have to be sceptical, however, as to the benchmarks that are chosen for a paper. Most of the data relates to the Multi2 workload, consisting of cscope, cpp and postgres. As noted, both cscope and postgres produce considerable looping references, which is where the UBM scheme provides the greatest performance improvements. cpp is mostly sequential, and UB-M also indirectly helps here as it detects sequential references and prevents them from polluting the cache.

Thus the benchmarks could be considered to have been chosen to suit UBM. For instance we can ask why cpp was chosen instead of gcc, or ld (as used in the ULTRIX multiple application trace in the literature). Also, we notice that the authors ran the SPEC SDET benchmark (a simulated multiprogramming environment) to "warm up the caches", yet this was not used as one of the standard benchmarks elsewhere.

Of course, we have to consider the fact that most programs do indeed have looping, and certainly sequential, references, so while the workload may be slightly biased it is not unreasonable. In particular postgres, as a database server, is a sensible 'real-world' application for benchmarking.

Also, the authors deliberately do not compare UBM with "schemes based on recency and/or frequency factors such as FBR, LRU-K and LRFU", stating that any of these schemes can be used to manage 'other' references in the UBM scheme. While this is true, that statement applies to most of the other schemes as well (especially LRU), and it would have been interesting to see how managing the entire cache with these schemes compares to only managing the 'other' portion. We have to question the authors' motivations in omitting these algorithms from consideration.

4 Performance

Even considering the fact that the benchmark portfolio may have been tailored to UBM, the results are compelling - hit ratios increased by an average of 29.2% (in trace-driven simulations), and elapsed times reduced by an average of 28.7% (for a real implementation on FreeBSD). The system performance overhead - 5% for a CPU-bound task running in parallel with the benchmark - is tolerable in the light of such gains, especially given the claim that the reference implementation was not optimised. For instance, optimised arrangement of summary information and ghost buffers can make a real difference in terms of reducing the CPU's TLB and cache misses.

The authors admit that only having one block for caching sequential references (after the cache warms up) causes some extra misses, as loops are necessarily considered sequential references on the first iteration. It would have been interesting to consider the performance if the sequential cache was set to some larger minimum size. Of course, this introduces another parameter which needs to be tuned, and that is against the UBM philosophy to some extent. More ideally, a hypothetical marginal gain function for this effect would be introduced, and the authors mention future work in this area.

5 Mangoes and Grapes

Section 4.6, and Figure 11, compare UBM to the Application-Controlled File Caching (ACFS) scheme, a user-hint based approach to file caching designed by Cao, Felten and Lee. The ACFS data was obtained from the ACFS paper, while the data for other schemes was obtained from simultations.

It is prudent to worry about results taken from multiple sources in this way. However, hopefully this was done carefully enough for the results to be valid. The simulations were run on traces provided by Cao, and the data variable (hit rate) is well defined.

6 What's In A Name?

I did also feel that the name "Unified Buffer Management" is not really appropriate for the scheme that this paper details. Traditional cache management schemes treat the cache as a single entity, whereas UBM treats it as three partitions - hence it seems that the buffer cache is more divided than unified. A name such as "Dynamic Cache Partitioning" may have been more appropriate.

Also, naming the third category of references "other" references causes some ambiguity when the word "other" is used in various contexts. These references would be better labelled "miscellaneous".

7 Unanswered Questions

One fairly major shortcoming is that the detection algorithm, which is an integral part of this research, is not described in any detail. It is not clear how a sequential pattern is detected, how many sequential references are required before it is classified as such, and how the fields in the reference table are updated. Presumably, the time of last reference must also be kept in the table to facilitate the calculation of loop period.

The paper talks about files analogously to references, but of course each file could have multiple reference patterns. It is thus not clear whether multiple entries are kept for different ranges within the one file. This would be useful for applications like databases, which generally use a large database file and read certain ranges sequentially. Since postgres was the best performing benchmark, I suspect that this was indeed considered.

8 Conclusion

While simplicity is often the key to performance, the authors have demonstrated that a more complex buffer management scheme, using a dynamically divided buffer cache with different replacement algorithms applied to each partition, does indeed provide real gains. This research thus has fairly wide applicativity to cache design in general.

The paper is well-written, with excellent diagrams and graphs. While the benchmarks may have been selected to favour UBM to some extent, they are still relevant. The lack of detail in the description of the detection algorithm does limit its direct usefulness and the reproducability of the results. Nevertheless, this is a good paper which is worth considering when implementing cache algorithms.