Improving Data Cache Performance via Address Correlation: An Upper Bound Study

Peng-fei Chuang [1], Resit Sendag [2], and David J. Lilja [1]

[1] Department of Electrical and Computer Engineering
Minnesota Supercomputing Institute
University of Minnesota
200 Union St. S.E., Minneapolis, MN 55455, USA
{pengfei, lilja}@ece.umn.edu

[2] Department of Electrical and Computer Engineering
University of Rhode Island
4 E. Alumni Ave, Kingston, RI 02881, USA
sendag@ele.uri.edu

Abstract. Address correlation is a technique that links the addresses that reference the same data values. Using a detailed source-code level analysis, a recent study [1] revealed that different addresses containing the same data can often be correlated at run-time to eliminate on-chip data cache misses. In this paper, we study the upper-bound performance of an Address Correlation System (ACS), and discuss specific optimizations for a realistic hardware implementation. An ACS can effectively eliminate most of the L1 data cache misses by supplying the data from a correlated address already found in the cache to thereby improve the performance of the processor. For 10 of the SPEC CPU2000 benchmarks, 57 to 99% of all L1 data cache load misses can be eliminated, which produces an increase of 0 to 243% in the overall performance of a superscalar processor. We also show that an ACS with 1-2 correlations for a value can usually provide comparable performance results to that of the upper bound. Furthermore, a considerable number of correlations can be found within the same set in the L1 data cache, which suggests that a low-cost ACS implementation is possible.

1 Introduction

A recent study [1] has introduced address correlation, which is based on the dynamic linking of addresses that store the same value. This run-time correlation can be used to improve the performance of on-chip data caches by forwarding data to the processor on a miss or a partial hit[1] that is already resident in the cache at other correlated addresses. Source code-level analysis presented in [1] indicates that semantically equivalent information, duplicated references, and frequent values are the major causes of the address correlations, and that taking advantage of duplicated references has excellent potential to benefit object-oriented programs due to their extensive usage of aggregation classes.

In this study, we focus on the upper bound potential of address correlation, and specific optimizations that will lead to better understanding of how it works. An *address correlation system* (ACS) is designed that stores the addresses evicted from the cache and their correlated addresses, which remain in the L1 data cache to supply the data for the evicted addresses. We show that there is substantial potential for hiding memory latency by providing the data from a correlated address instead of incurring a full miss penalty. When we have limited resources, where to find a useful correlation becomes important. We show that a useful correlation can usually be found in cache lines

[1] A partial hit occurs when a request on an address is a hit in the cache, but the data at the address is not ready yet because it is in the process of being read by a previous miss in the same cache block. A partial hit can be as slow as a complete cache miss, depending on how close together the two accesses occur.

that are physically close to each other. Furthermore, the number of addresses that can be usefully correlated is usually bounded. Our results show that an ACS with 1-2 correlations for a value can usually provide comparable performance results to that of the upper bound results obtained in this study.

The remainder of this paper is organized as follows. Section 2 describes an address correlation system. Section 3 presents the experimental setup. The upper-bound performance results are given in Section 4. In Section 5, we present some results on how to reduce the cost of correlations. Section 6 discusses specific optimizations while related work is given in Section 7. Finally, we conclude in Section 8.

2 Address Correlation System

In this section, we start with a brief introduction of the concept of address correlation since it is new to the research community. We then describe the basic operation of an ACS. An exhaustive investigation of hardware design parameters for an address correlation mechanism and implementation details is beyond the scope of this paper.

2.1. What is address correlation?

Address correlation is a new approach for exploiting value locality. A conventional cache system, which relies heavily on the temporal and spatial locality that programs exhibit, may exceed the limits of locality via address correlation. In [1], we investigate this program phenomenon, which links the addresses that reference the same data, and show the causes of address correlations with a detailed source code-level analysis of the programs. Address correlation is based on the observation that there is redundancy in data that was stored to memory (store value locality) [3], and that a few values appear very frequently in memory locations (frequent value locality) [4]. Extending these two observations, in [1], we proposed run-time correlation of the addresses that store the same data to improve the performance of on-chip data caches by forwarding data to the processor on a miss that is already resident in the cache at other correlated addresses. In this study, we focus on the upper bound potential of the ACS and specific optimizations that will lead to designs for the run-time correlation of addresses.

2.2. The Components of ACS

Address correlation requires tracking the contents of the cache, the relationships between different locations in the cache, and the memory access history. This section describes the basic operation of an ACS to study the upper bound potential of address correlation.

As shown in Fig. 1, ACS contains two main components: the Address Linking Table (ALT) and the Address Correlation Table (ACT). The ALT stores the correlations between addresses currently residing in the L1 data cache. The ACT keeps all the addresses evicted from the cache and their correlated addresses in the L1 data cache that can supply data for them. In other words, the ACT is the table providing the information about alternative data source for a cache miss.

When a cache line is brought into the L1 data cache, all the addresses in that cache line are linked with the addresses of other cache lines according to their values. An address will never be linked with the addresses in the same cache line since they all will be evicted from the cache at the same time. If a new value is stored to an address, all the links to the updated address are removed and new links are generated for the address based on its new value.

When a cache line is evicted from the cache, the linking information of each address in the line is moved from the ALT to the ACT. Lists of links for those addresses in the ACT that correlate with the evicted addresses are updated to properly reflect the availability of data (alternative data source). Removal of an entry from the ACT occurs when all the linked addresses of an ACT entry are evicted

from the L1 data cache or written with new values, or when the address of an ACT entry is put back to the L1 data cache again.

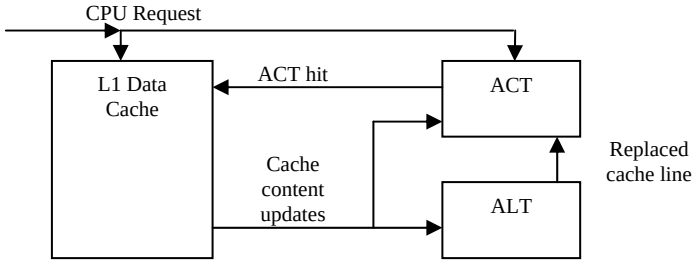

Fig. 1. *Address Correlation system.* The L1 data cache and the ACT are accessed at the same time. When there is a hit in the ACT (which is a miss in the L1 data cache), a correlated address is supplied from the ACT. The L1 data cache then uses this information to supply data in the next cycle

2.3. Servicing Cache Misses Partial Hits Faster

The ACT is accessed in parallel with the L1 data cache on a memory access. When the requested address misses in the cache but hits in the ACT, an address is then sent to the L1 data cache. The L1 data cache uses this information to find the value for a previous cache miss.

When a cache miss is supplied by an ACT hit, it is possible to cancel the data request to lower level memory such as L2 cache. In this study, we have explored two design alternatives on a cache miss: canceling the data request to lower level memory on an ACT hit (*cancel_L2*), and allowing the missed cache line to be brought into L1 from lower level memory (*with_L2*). The *cancel_L2* may change the cache behavior and its contents. However, it substantially reduces the traffic between the L1 and L2 data caches and the activities in the ACS system. The details of these designs are left as future work since it is not our intent in this study to design a specific mechanism for address correlation. The results given in the later sections are all for the *cancel_L2* configuration unless otherwise specified.

The ACS can also supply the data for partial hits. A partial hit occurs when a request for an address is a cache hit, but the data at the address is not yet available in the L1 data cache because it is still in transit from the lower level memory. Fast data delivery of partial hits is made possible by delaying the time for an entry to be removed from the ACT. An entry of addresses of the same cache line in the ACT is removed after the data in its cache line is in the cache instead of when the tag of its cache line is set in the cache. Since an address whose data exists in the cache will never be in the ACT, a hit in the ACT indicates that there was either a cache miss or a partial hit.

3 Experimental Setup

To determine the performance potential of an ACS, we used an execution driven simulation study based on the SimpleScalar simulator [5]. We modified the memory module to study the potential of address correlation and implemented an ACS for an upper-bound study.

The processor/memory model used in this study is capable of issuing 8 instructions per cycle with out-of-order execution. It has a 64-entry RUU and a 64-entry load/store queue. The processor has a 32 KB, 2-way set associative first-level (L1) instruction cache. Various sizes of the L1 data cache (8KB, 16KB, 32KB) with 4-way set associativity are examined in the following simulations. The L1 data cache is non-blocking with 4 ports. Both caches have block sizes of 32 bytes and 1-cycle hit latency. The L2 cache size is 1MB. The L2 cache has 64-byte blocks and a hit latency of 12 cycles. The round-trip main memory access latency is 200 cycles for all of the experiments. There is a 64-entry 4-

way set associative instruction TLB and 128-entry 4-way set associative data TLB, each with a 30-cycle miss penalty.

The test suite used in this study consists of selected MinneSPEC [6] CPU2000 benchmark programs. All binaries are SimpleScalar PISA and compiled with SimpleScalar gcc at -O3 and each benchmark ran to completion.

4 Performance of an ACS

In this section, we examine the performance results of an ACS, which was described in Section 2. This analysis will give us the real potential that an address correlation mechanism can offer for exceeding the limits of locality. The results are given for a 32 KB L1 data cache with 4-way associativity, unless specified otherwise.

In Fig. 2, we show the reduction in L1 data cache misses with different cache sizes and the effect of address correlation on reducing the number of misses. The original superscalar processor with an 8KB, 4-way associative L1 data cache is used as the basis for these comparisons. We see that address correlation can eliminate most of the misses in the L1 data cache. That is, the data requested is usually already in the cache at other addresses.

Fig. 3 shows how memory accesses are serviced. We can see that the ACS services 57 to 99% of the misses in the L1 cache. We categorized the memory accesses that are serviced by the ACS as the percentage of the misses that would have been serviced by the L2 cache and memory, in the absence of the ACS. For example, for *181.mcf*, 69% of the misses in the L1 data cache that would have been serviced by the L2 cache, and 37% of the misses that would have been serviced by the memory, in a system without address correlation are serviced by the ACS. Since memory access has a substantially higher latency than the L2 cache, the more misses that are serviced by the ACS instead of by memory, the more latency that can be hidden from the processor. These results show that the ACS is very effective in hiding the latency for memory references.

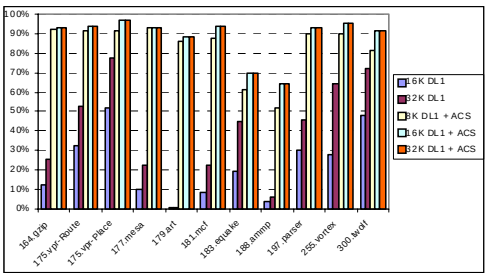

Fig. 2. The percentage cache miss reduction for varying L1 data cache sizes with and without the ACS. The original superscalar processor with an 8KB, 4-way associative L1 data cache is used as the basis for these comparisons

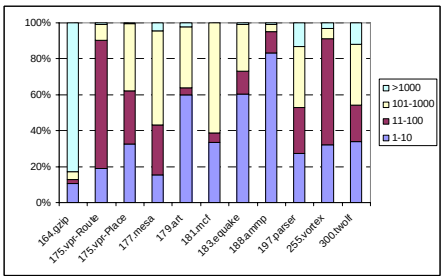

Fig. 3. The fraction of memory references that are serviced by the L1 cache, the ACS, the L2 cache, and memory

Finally, we give the potential overall processor speedup results that could be obtained in a system using address correlation. Here we assume 1 cycle is required for an ACT lookup and another 1 cycle is needed to supply data from the L1 data cache. While it is optimistic to choose only 2 cycles for the ACS to supply an alternative address for a miss, our aim is to show the potential rather than the actual speedups for this new mechanism.

In Fig. 4, the speedup results show a wide range from 0% to 243%. Our preliminary observation shows that supplying data from a correlated address may work especially well for memory intensive benchmarks. 175.vpr, 177.mesa and 300.twolf are the benchmarks that cannot benefit from address correlation even if there is a large reduction in the data misses and partial hits. These benchmarks do not exhibit memory-intensive behavior as is the case for 179.art and 181.mcf. In Fig. 3, we see that in the case of *179.art* and *181.mcf*, a large portion of the misses in the L1 data cache, which are

eliminated by the ACS, would have been serviced by the memory in a system without an ACS. Therefore, it is easy to understand the performance results in Fig. 4 for the above benchmarks. Fig. 4 also shows that from the two variations of the ACS design, which is mentioned in section 2, *cancel_L2* performs better for *179.art*, *181.mcf* and *188.ammp*, while for the other benchmarks *with_L2* performs better.

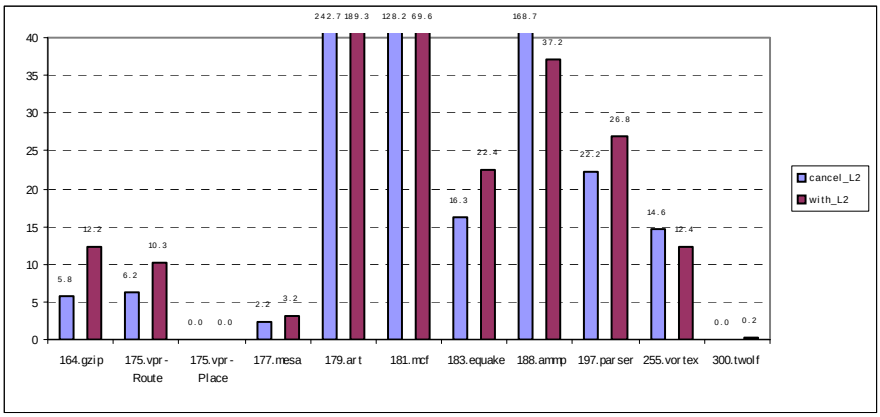

Fig. 4. The percentage speedup obtained by supplying data from a potential address residing in the L1 cache. In the case of *cancel_L2*, a hit in the ACT cancels the request for data from the lower level memory, while in *with_L2*, the request to lower level memory is allowed to continue and bring the data from lower level memory into the cache

5 Reducing the cost of correlations

The performance results in Section 4 are only for an infinite table size ACS. They are nevertheless interesting since they gave us a sense of how much additional performance we can expect from an aggressive and accurate design of address correlation.

In Figs. 5-10, we show the results of some experiments that we have conducted to gain more insights into the performance potential of a realistic hardware implementation of an ACS system. Figs. 5 and 6 show the number of different addresses that can potentially supply the data on a miss and on a partial hit, respectively. We can see that for *181.mcf*, the same data can be found in 1 to 10 different addresses for 48% of the data misses, while for 12% of the misses it can be found in 11 to 100 different addresses, and so on. The interval ">1000" is more likely due to trivial values (0, 1, etc.), which can be a source of address correlation. Very similar results are obtained for the partial hits, as shown in Fig. 6.

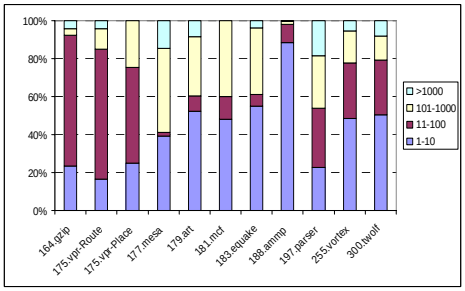

Fig. 5. The number of different addresses in the cache in which the missed data is found

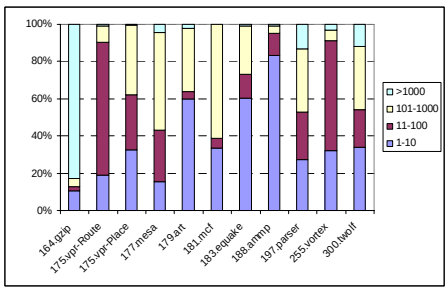

Fig. 6. The number of different addresses in the cache in which the partial hit data is found

When we have limited resources, where to find a useful correlation becomes an important issue. Memory allocation patterns of data, if there are any, may be one of the possible sources for finding data that can be correlated. In Fig. 7, we have plotted the distance of the addresses, in bytes, where the data can be found on a miss. Due to readability, only 3 out of the 10 benchmarks studied are shown in the figure. We can see that, for 255.vortex, the data values for 35% of all misses can be found within a distance of 128 bytes. For 183.equake, the majority of the addresses that can be correlated fall within the range of 1000 bytes, and for 164.gzip, we can find useful data distributed in a period of 1500 to 1800 bytes. This behavior implies there may be some certain storage allocation pattern that can be exploited for address correlation. Although this information may be used for an individual benchmark, a common solution for all benchmarks is not straight-forward. While we show a distance of up to 7KB in Fig. 7, we see the addresses scattered all over the address range when we plot the whole range. We conclude that this figure does not show useful information for a hardware implementation since there is a random distribution of addresses that are possible candidates for supplying a miss.

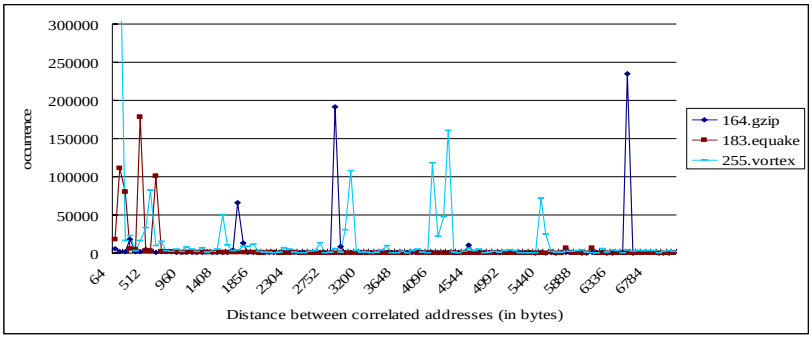

Fig. 7. The number of misses found within an address distance of 7KB, measured in bytes

Figs. 8 and 9 measure the distance of the potential correlated addresses, in sets, in which the data miss and partial hit are found, respectively. We can see that, usually 35% to 50% of the potential addresses for a data miss, and 30% to 45% of the potential addresses for a partial hit, are in the same set in the cache. The distance of the data in sets gives us insights into efficient designs for an address correlation mechanism. Having found the data within the same set, the address correlation mechanism would require fewer searches for potential addresses with which to correlate.

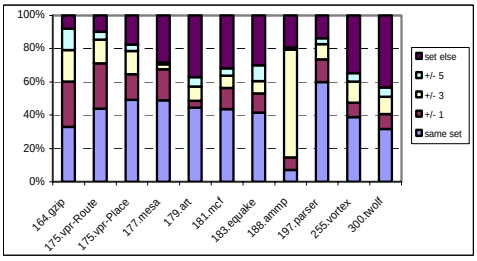

Fig. 8. The distance to the address in which data is found in the cache on a miss, measured by sets

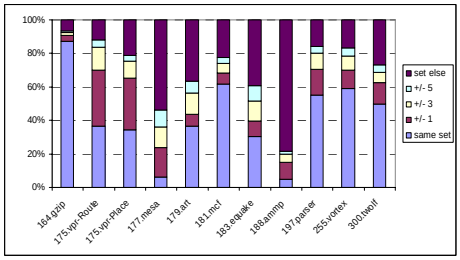

Fig. 9. The distance to the address in which data is found in the cache on a partial hit, measured by sets

In Fig. 10, we show the effect of limiting the number of correlations. Figs. 5 and 6 have shown that, at a given time, thousands of addresses may contain the same data within a 32KB L1 data cache. It is impractical trying to correlate all these addresses. We have used a simple FIFO replacement policy to limit the number of correlations and compare the performance results with the upper bound where all addresses that can be correlated are taken into account. This figure shows that two addresses

correlated to each other would be enough for comparable results to that of the upper bound. On average, correlating between two addresses can eliminate 75% of the misses eliminated in the upper bound. Four and eight correlations can eliminate 85 and 88% of the misses eliminated in the upper bound, respectively. These results suggest that an ACS with 1-2 correlations for a value can usually provide comparable performance results to that obtained in the upper bound results study.

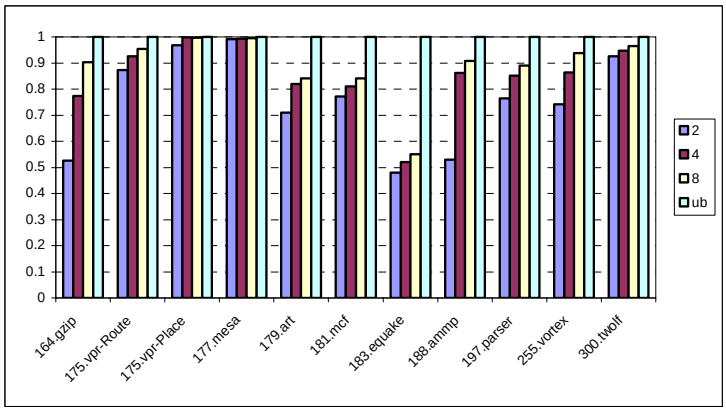

Fig. 10. These results show that an ACS with 2, 4, and 8 correlations for a value can perform almost as well as the upper bound (ub)

6 Discussion

Set distance provides a more meaningful hint on where to look for correlatable addresses compared to byte distance. Figs. 8-9 show that at least 30% of the misses and 35% of the partial hits can be served by addresses of the same set. If we limit the search of address correlation within the same cache set, the linking information can be stored using a compressed format. For a 4-way associative cache with 8 words cache line size, a 2-bit cache line number and a 3-bit word position within a cache line can identify a correlated address. Furthermore, Fig. 10 suggests that 1-2 correlations for a value can usually provide good performance results. These two observations give insights for possible design alternatives for implementing address correlation with low search overhead and low hardware cost. One possible way to exploit address correlation is storing the linked addresses within the L1 data cache. Such a design is currently under investigation.

In addition to searching for correlation information in the cache, there are other ways to accurately detect addresses that have the same value at run-time. The load-store queue is one of the potential sources for revealing the relation between two memory addresses. When a load or a store is followed by a store using the same register containing the same value, we can safely link the two addresses as shown in Fig. 11. This method requires checking the dependencies to ensure that the value in the register, which is the destination or the source of a load or store and the source of a later store, is unchanged. One advantage of this method is that the correlation of addresses comes for free, and there is no additional work for searching for addresses in cache to correlate. We expect that semantically equivalent information and duplicated references, which are two of the sources of address correlations, will be good candidates to be correlated in the load-store queue. In addition, wider issue machines would benefit more from this design since addresses storing that same value are more likely to be found in the load-store queue at the same time. Moreover, additional instruction history can be used to assist finding related addresses.

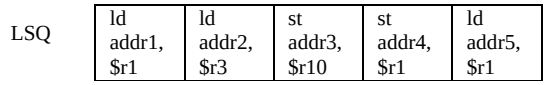

Fig. 11. Content of addr1 is loaded into $r1 and is later stored into addr4. Since addr1 and addr4 hold the same value, they can be correlated to each other

7 Related Work

Value locality [2] describes the recurrence of a previously seen value within a storage location. It allows the classical dataflow limit to be exceeded by executing instructions before their operand values are available.

There have been several studies on exploiting different kinds of value locality. Store value locality [3, 10-13], a recently discovered program attribute that characterizes both memory-centric (based on message passing) and producer-centric (based on program structure) prediction mechanisms for stored data values, introduces the concept of redundancy in data words stored to memory by computer programs. In the study, many store instructions are shown to be silent; that is, they do not change the state of the system because they write a value that already exists at the write address. Consequently, they can safely be eliminated from the dynamic instruction stream.

Recent studies have also introduced Frequent Value Locality [2, 7-9], another type of locality in which a few values appear very frequently in memory locations and are, therefore, involved in a large fraction of memory accesses. Tracking the values involved in memory accesses has shown that, at any given point in the program's execution, a small number of distinct values occupy a very large fraction of these referenced locations. It has been shown [4] that, in the tested programs, ten distinct values occupy over 50% of all memory locations and on an average account for nearly 50% of all memory accesses during program execution.

Extending these two complementary studies, in our previous work, we propose a new technique, *Address Correlation* [1], which is based on the dynamic linking of addresses that store the same value. This previous study investigated the causes of address correlations using a detailed source code-level analysis.

In this paper, we study the upper bound potential of address correlation and investigate specific optimizations to exploit it.

8 Conclusion

We have demonstrated the upper bound potential of an address correlation system (ACS). The ACS stores the addresses evicted from the cache and their correlated addresses, which remain in the L1 data cache to supply the data for the evicted addresses. We show that there is substantial potential for hiding memory latency by providing the data from a correlated address instead of incurring a full miss penalty. When we have limited resources, where to find a useful correlation becomes important. We show that a useful correlation can usually be found in cache lines that are physically close to each other. Our results also show that, usually, 40% of the misses and 42% of the partial hits in the L1 data cache can be found within the same set. Furthermore, the number of addresses that can be usefully correlated is usually bounded. Simulation results illustrate that an ACS with 1-2 correlations for a value can usually provide comparable performance to that of the upper bound results obtained in this study. Based on the attributes that we have studied in this paper, we are currently investigating an L1 data cache design that supports address correlation.

References

1. R. Sendag, P. Chuang, and D. J. Lilja, “Address Correlation: Exceeding the Limits of Locality”, IEEE Computer Architecture Letters, Volume 2, May 2003.
2. M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, “Value Locality and Load Value Prediction,” in Proceedings of the second international conference on architectural support for programming languages and operating systems, pp. 138-147, 1996.
3. K. M. Lepak and M. H. Lipasti, “On the Value Locality of Store Instructions,” in Proceedings of the 27th International Symposium on Computer Architecture, 2000.
4. Y. Zhang, J. Yang, and R. Gupta, “Frequent value locality and value-centric data cache design,” ACM 9th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 150-159, Cambridge, MA, November 2000.
5. D.C. Burger, T.M. Austin, and S. Bennett, “Evaluating future Microprocessors: The SimpleScalar Tool Set,” Technical Report CS-TR-96-1308, University of Wisconsin-Madison, July 1996.
6. AJ KleinOsowski and D. J. Lilja, “MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research,” Computer Architecture Letters, Volume 1, May, 2002
7. J. Yang and R. Gupta, “Energy Efficient Frequent Value Data Cache Design,” ACM/IEEE 35th International Symposium on Microarchitecture, November 2002.
8. J. Yang and R. Gupta, “Frequent Value Locality and its Applications,” Special Issue on Memory Systems, ACM Transactions on Embedded Computing Systems, 2002.
9. J. Yang, Y. Zhang, and R. Gupta, “Frequent Value Compression in Data Caches,” ACM/IEEE 33rd International Symposium on Microarchitecture, pp. 258-265, December 2000.
10. K. M. Lepak and M. H. Lipasti, “Temporally Silent Stores,” in Proceedings of the Tenth International Conference. on Architectural Support for Programming Languages and Operating Systems, October 2002.
11. K. M. Lepak and M. H. Lipasti, “Silent Stores for Free,” in Proceedings of the 33rd Annual ACM/IEEE International. Symposium on Microarchitecture, December 2000.
12. G. B. Bell, K. M. Lepak, and M. H. Lipasti, “Characterization of Silent Stores,” in Proceedings of the International Conference on Parallel Architectuers and Compilation Techniques, 2000.
13. K. M. Lepak, G. B. Bell, and M. H. Lipasti, “Silent Stores and Store Value Locality,” IEEE Transactions on Computers, Vol. 50, No. 11, November 2001.