Unified Dual Data Caches

Ben Juurlink
Computer Engineering Laboratory
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology
P.O. Box 5031, 2600 GA Delft, The Netherlands
benj@ce.et.tudelft.nl

Abstract

The dual data cache is a cache organization with a split temporal/spatial cache. The temporal sub-cache stores data exhibiting temporal locality and the spatial sub-cache saves data exhibiting spatial locality. A locality prediction table is used to predict the type of locality load/store instructions exhibit. In this way, both types of locality can be exploited more effectively. Unfortunately, the dual data cache does not make effective use of the entire cache capacity. If most memory references exhibit the same type of locality, only one sub-cache will be used. In this paper we, therefore, propose a cache organization called the Unified Dual Data Cache *that employs only one (unified) cache unit. If a cache miss occurs and the locality prediction is temporal, only the missing block is fetched from the next memory level. If on the other hand spatial locality is predicted, adjacent blocks are also brought to the cache. In fact, we present two versions of the UDDC called the UDDC Type A (UDDC-A) and the UDDC Type B (UDDC-B), respectively. The difference between the two types is that in the UDDC-B each smaller block is tagged, while in the UDDC-A the smaller blocks within a larger block share the tag.*

1. Introduction

Caches are very effective in reducing the average memory access time because programs exhibit two types of locality: temporal and spatial. Temporal locality refers to the property that recently accessed data items tend to be referenced again in the near future, and spatial locality describes the tendency that adjacent memory locations are referenced close together in time. Unfortunately, many applications, in particular numerical applications, have characteristics that may degrade the cache performance. For example, sometimes numerical applications access vectors with a stride unequal to one. Because caches store blocks of consecutive data, non-unit strides limit the ability to take advantage of spatial locality. In the worst case, the stride is larger than the block size. A second factor that may limit the effectiveness of conventional cache organizations for numerical applications is that when the vector length is larger than the cache size, the vector sweeps itself out, so temporal locality is not exploited. This situation is even worse if the stride and the cache size are not co-prime, since then not all cache sets are used to store the vector elements.

In order to remedy these problems, Gonzalez et al. [6] proposed a cache organization called the *Dual Data Cache* (DDC). It consists of two sub-caches. The *temporal cache* stores data exhibiting temporal locality and employs small lines. The *spatial cache* stores data with spatial locality and uses larger lines. The DDC may also decide to bypass the cache if a vector interferes with itself. A history table called the *locality prediction table* is employed to predict the type of locality.

Unfortunately, the dual data cache does not make effective use of the entire cache capacity. If most memory references exhibit the same type of locality, only one of the sub-caches will be used. Moreover, the locality behavior may vary during program execution. While some parts of a program may exhibit mainly temporal locality, other parts may exhibit primarily spatial locality. To reduce this drawback this paper proposes a cache organization called the *Unified Dual Data Cache* (UDDC). In fact, two versions of the UDDC are presented, called the UDDC Type A (UDDC-A) and the UDDC Type B (UDDC-B), respectively. Both cache organizations support two block sizes. The difference between the two types is that in the UDDC-B each smaller block is tagged, while in the UDDC-A the smaller blocks within a larger block share the tag.

This paper is organized as follows. In Section 2 related work is described. Section 3 describes the DDC and presents both versions of the UDDC. Performance results are given in Section 4, and concluding remarks are given in Section 5.

Proceedings of the Euromicro Symposium on Digital System Design (DSD'03)
0-7695-2003-0/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

2. Related Work

The (unified) dual data cache is a way of supporting multiple line sizes. Large lines are employed if a load/store instruction exhibits spatial locality and small lines are employed if it exhibits temporal locality. In this section some related proposals are described.

Most contemporary processors employ a split L1 instruction/data cache. Besides the advantage of being able to fetch an instruction and a data word in the same cycle without needing a multi-ported cache, a split cache also offers the opportunity to select a different block size for the instruction cache and the data cache. This is useful because, in general, there is more spatial locality in the code segment.

Lee and Tyson [12] describe a related cache partitioning technique called *region-based caching*. They show that stack, global, and heap data exhibit different memory reference characteristics and, therefore, propose to partition the cache by these memory regions. The effects of employing different line sizes for the stack-, global-, and heap-cache were not investigated, however.

The L1 cache of the HP PA7200 processor consists of a large, external, direct-mapped main cache and a smaller, on-chip, fully-associative *assist cache* [3]. The assist cache not only serves to reduce conflicts in the main cache, but also as a spatial sub-cache. At compile time, some load/store instructions are marked as "spatial locality only". The data accessed by these instructions are not cached in the main cache but only in the assist cache.

More closely related to our work is the work of Johnson et al. [9]. They propose a hardware mechanism that detects spatial locality by counting how many of the smaller blocks in a larger block are valid in the cache. When a line is displaced its spatial locality information is stored in a table called the *Memory Address Table* (MAT) whose entries correspond to memory regions called *macroblocks*. The idea is that the access characteristics of the cache lines contained within each macroblock are relatively uniform. On a memory access the MAT is accessed in parallel with the data cache. If an entry is found that indicates spatial reuse, a larger block is fetched, otherwise the smaller size is fetched. If no entry is found the larger fetch size is chosen. One of the most important differences between the work of Johnson et al. and the DDC and our proposals is that in the former the effective address is used to guide the choice of fetch size whereas the DDC and UDDCs use the PC.

The *Adaptive Line Size* (ALS) cache [17] is similar to the work of Johnson et al. [9] since it also keeps track of adjacent blocks while they are cached. The main difference is that the ALS cache can support more than two line sizes but to do so it adds the current line size and a 2-bit saturating counter *to each memory block*. In other words, the ALS cache keeps track of the access characteristics of all lines while they are in memory and does not partition memory into macroblocks.

Van Vleet et al. [16] classify every read operation as a load (just the missing line is fetched) or a superload (adjacent lines are also fetched). They present an optimal off-line algorithm as well as an online algorithm using unlimited space, and investigated the performance improvements that can be obtained if the fetch size decision is based on profiling.

A simplified version of the DDC called the *selective cache* is also described in [6]. Like the UDDCs, the selective cache employs only one cache unit. It does not, however, support multiple line sizes. It is, essentially, a conventional cache with a selective caching policy.

We finally remark that a survey of cache organizations with a split temporal/spatial cache can be found in [13].

3. The Dual Data Cache and the Unified Dual Data Caches

In this section we briefly review the Dual Data Cache and present the Unified Dual Data Caches.

3.1. The Dual Data Cache

The DDC (Figure 1) consists of a split temporal/spatial cache. The temporal cache stores data exhibiting temporal locality and the spatial cache stores data displaying spatial locality. Both sub-caches are organized as a conventional cache on the understanding that the temporal cache has a smaller line size than the spatial cache. Each time the processor issues a memory reference, both caches are accessed in parallel. A hit occurs if the requested data is found in either sub-cache. Otherwise, the requested data is fetched from the next memory level and, depending on the predicted type of locality, cached in one of the two sub-caches or not cached at all. The prediction is made with the help of a history table called the *locality prediction table* that contains information about recently executed load/store instructions, such as the instruction address, the data address referenced last, and the stride between the last and previous to last address. The prediction can be temporal, spatial, or bypass. If the prediction is temporal (spatial), the required data is brought to the temporal (spatial) sub-cache. If the prediction is bypass, it is placed in neither sub-cache.

We briefly describe the locality prediction algorithm since it is also part of the UDDCs. More details can be found in [6].

If the stride varies frequently, the default prediction is used. As was done in [6], we assume the default prediction is temporal. If, on the other hand, the stride does not change for two successive executions of a load/store instruction, the prediction depends on the length of the stride. If the stride is

IEEE COMPUTER SOCIETY

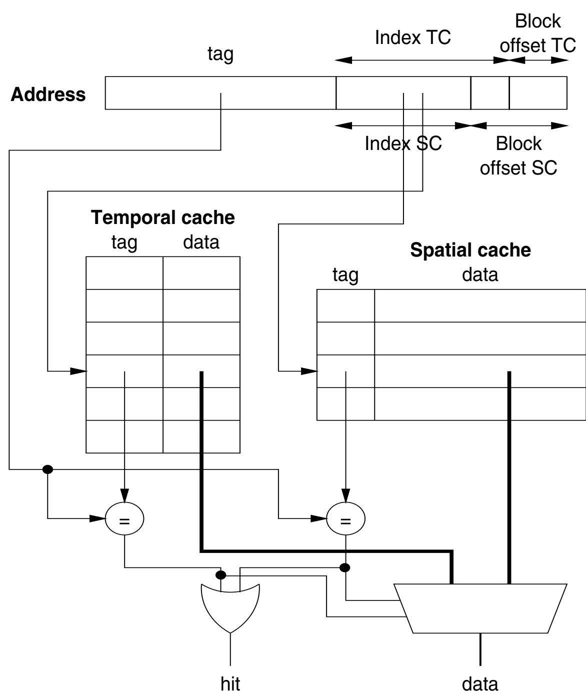

Figure 1. Simplified block diagram of the DDC. In this example, both sub-caches have the same capacity.

smaller than the line size of the spatial cache, the prediction is spatial. If it is larger the prediction also depends on the number of times the same stride has occurred (the *length*) and the *stride family*. The stride family x is defined as the set of strides $\sigma \cdot 2^x$ where σ is odd. For example, $14 = 7 \cdot 2^1$ and $10 = 5 \cdot 2^1$ both belong to stride family 1. All strides in the same family have the same self-interference behavior [7, 15]. Note that the stride family is equal to the number of trailing zeroes in the binary representation of the stride. If the *length* multiplied by 2^x is not larger than the size of the temporal cache, the prediction is temporal. Otherwise, the prediction is bypass.

We remark that for a read operation, the DDC operates very much like a two-way set-associative cache. The tag check can, therefore, not be overlapped with the transmission of data, which is one of the advantages of direct-mapped caches [8]. The Unified Dual Data Caches presented in the next section do not have this drawback.

3.2. The Unified Dual Data Caches

The DDC attempts to reduce cache pollution caused by non-unit strides and does not cache vectors that interfere with themselves. A limitation of the DDC, however, is that when most memory references exhibit the same type of locality the effective cache capacity is significantly reduced. In this section we introduce the two versions of the UDDC that overcome this problem.

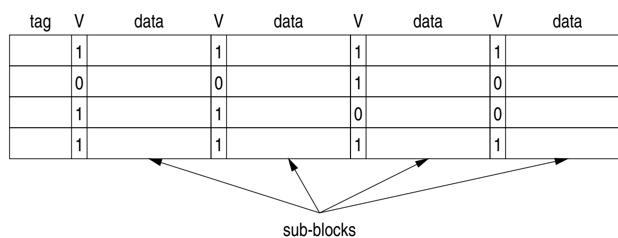

Figure 2. UDDC-A, sub-block placement.

The UDDC-A uses the same locality prediction mechanism as the DDC but there is only one (unified) cache unit. It employs *sub-block placement* [8] which is a technique introduced originally to reduce the amount of tag storage. Each cache block is divided into several sub-blocks and a valid bit is added to every sub-block. A cache hit occurs if the tag of the requested word matches the tag of the corresponding cache entry *and* the valid bit of the requested sub-block is set. Figure 2 depicts an example with four sub-blocks per block. We remark that such a cache organization is sometimes also called a sectored cache.

On a cache miss, the locality prediction determines if only the miss sub-block is fetched from the next memory level or the adjacent sub-blocks as well. If the prediction is temporal, only the requested sub-block is fetched. Furthermore, if the tag of the miss block does not match the tag of the block to be replaced, the other sub-blocks in the block are invalidated. If the prediction is spatial, the adjacent sub-blocks in the block are also fetched. In the example depicted in Figure 2, the first and last block were loaded on a spatial miss (a miss generated by a load whose prediction is spatial). The third sub-block of the second block was loaded on a temporal miss. Because the tag of the requested sub-block did not match the tag of the replacement block, all other sub-blocks have been invalidated. The second sub-block of the third block was also loaded on a temporal miss, but in this case the tag of the requested sub-block matched the tag of the replacement block.

The advantage of the UDDC-A is the increase in effective cache capacity. Additionally, it requires less tag storage than the DDC. A potential disadvantage, however, is that when the tag of the requested sub-block does not match the tag of the block to be replaced, the other sub-blocks have to be invalidated. We, therefore, present another version of the UDDC in which every smaller block is tagged.

The UDDC-B is like a conventional cache with a small block size. However, if the locality prediction is spatial, several adjacent blocks are fetched from the next memory level. We follow the terminology of [8] and define a cache

IEEE COMPUTER SOCIETY

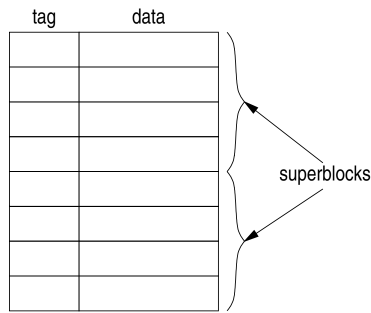

Figure 3. The UDDC-B. The accolades indicate the correspondence of four blocks with one super-block.

block as the unit of information associated with a tag. The smaller blocks of the UDDC-B will, therefore, be referred to as *blocks* and the larger blocks as *super-blocks*. In this cache organization, the other blocks in the super-block are still valid after a block is replaced. Figure 3 gives an example with four blocks in a super-block.

We remark that it is also possible to associate two (or more) address tags to each larger block by adding a bit to each smaller block that indicates the address tag with which it is associated. Such a cache organization is called a decoupled sectored cache [14]. The UDDC-A and UDDC-B are on opposite sites of the spectrum.

4. Performance Analysis

4.1. Tools and Benchmarks

We modified the `sim-safe` simulator of the SimpleScalar toolset [1] to generate address traces. These traces were subsequently fed into a cache simulator that can simulate conventional caches as well as the dual data caches. Amongst others, the simulator keeps track of the delay caused by cache misses which is used to calculate the average memory access time.

In order to evaluate the performance of the UDDCs, we used kernels and two benchmarks from the SPEC CFP 2000 suite. The kernels are used in many numerical applications:

- *mm64* and *mm100*: Multiplication of two $n \times n$ matrices using the familiar inner-product form. Because the cache behavior is influenced by the matrix dimension n, two values of n were used, $n = 64$ (*mm64*) and $n = 100$ (*mm100*).

- *lu80*: LU factorization with partial pivoting. The matrix dimension $n = 80$.

- *fft-ct*: An implementation of the Cooley-Tukey Fast Fourier Transform [4] of a vector of 2^{18} elements. It has been obtained from [10]. This benchmark accesses vectors with strides that are powers of two.

- *fft*: An alternative implementation of the FFT that uses unit strides. The vector length is again 2^{18}. In the original code, obtained from [5], the real and imaginary parts of the complex numbers were stored in two separate arrays allocated consecutively in memory. Since the array size is a multiple of the cache size, this resulted in a lot of conflict misses. In order to improve the cache performance of this kernel, we merged these arrays.

We also employed the four SPEC CFP 2000 benchmarks written in C (*177.mesa*, *179.art*, *183.equake* and *188.ammp*). Floating-point benchmarks were used because the DDC as well as the UDDCs are targeted at numerical/vectorizable applications. In order to reduce the simulation time, we used the large reduced input datasets from [11]. The execution characteristics of these reduced input datasets are similar to the execution profiles of the SPEC 2000 reference datasets. As argued in [11], using these reduced input datasets should be more representative than reducing the runtime by simulating, say, 100 million references after skipping the first 50 million, since this may cause the execution profile to be completely different from the execution profile obtained with the full input dataset.

4.2. Simulation Parameters

As was done in [6], we assume direct-mapped caches with write-back and write-allocate policies. The bus connecting the cache memory to the next memory level is eight bytes wide. The first 8-byte word of a requested block arrives after 10 cycles, and after that, successive words arrive at a rate of one per cycle. In other words, the miss penalty of a cache with a block size of B bytes is $9 + \lceil B/8 \rceil$ cycles, and the cost of a bypass is 10 cycles.

For the kernels, because they have relatively small working sets, we consider cache sizes of 8KB, 16KB, and 32KB. For the SPEC benchmarks, cache capacities of 32KB, 64KB, and 128KB are considered. As in [6], the line size of the temporal sub-cache of the DDC is 8 bytes and the block size of the spatial sub-cache is 32 bytes. For the UDDC-A (UDDC-B) this corresponds to a sub-block (block) size of 8 bytes and a block (super-block) size of 32 bytes. For the kernels, a locality prediction table of 16 entries was sufficient to have no misses except cold start misses. For the SPEC benchmarks, a direct-mapped locality prediction table with 256 entries was used.

The performance of a UDDC will be compared to a conventional, direct-mapped cache with a 32-byte block size

0-7695-2003-0/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

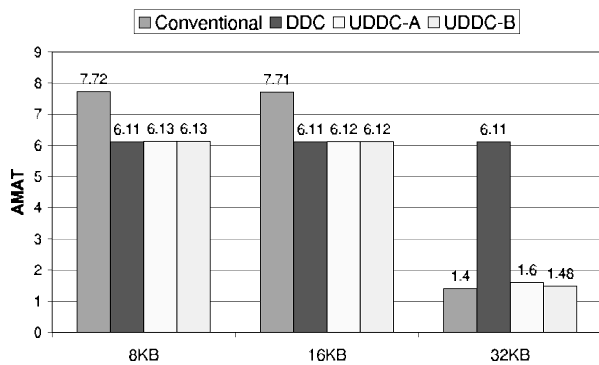

Figure 4. Average memory access times of the *mm64* kernel

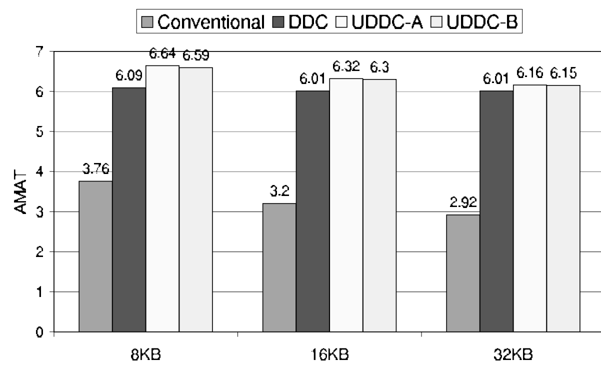

Figure 5. Average memory access times of the *mm100* kernel

and the same capacity, as well as a DDC with the same capacity. The size of the temporal sub-cache of the DDC is equal to the size of its spatial sub-cache. In the experiments described in [6], the additional memory required by the locality prediction table was taken into account by comparing a conventional cache with a capacity of S bits to a DDC with a spatial sub-cache of $S/2$ bits and a temporal sub-cache of $S/4$ bits. We cannot perform something similar because the UDDCs do not consist of split sub-caches. We remark, however, that the total size of a locality prediction table with 256 entries is less than 10% of the total size (tag and data bits) of a 32KB direct-mapped cache with a 32-byte block size.

4.3. Experimental Results

4.3.1 Kernels

Figure 4 depicts the average memory access times (AMATs) in cycles for the *mm64* kernel that multiplies two 64×64 matrices $C = A \times B$. Several observations can be made from this figure. First, when the cache size is 8KB or 16KB, the DDC and both types of the UDDC perform better than the conventional cache. Since the matrix dimension n is a power of two and the matrix B is accessed along the columns, not all cache sets are used to store the elements of a column of B (the 8KB cache uses 16 sets and the 16KB cache uses 32 sets). The conventional cache, therefore, incurs a cache miss on each access to an element of B. The DDC and the UDDCs, on the other hand, decide not to cache the columns of B, since $64 \times 8 = 512$ belongs to stride family 9 and $64 \times 2^9 = 32768$ is in these cases larger than the cache size. These cache organizations, therefore, benefit from the fact that the cost of a cache bypass is smaller than the time to fetch a 32-byte block. Furthermore, the row of A which is accessed in the inner loop of matrix multiplication and reused in the next iteration of the middle loop can stay in the cache, since it is not replaced by elements of B.

The second observation that can be made from Figure 4 is that when the cache size is 8KB or 16KB, the DDC, the UDDC-A, and the UDDC-B perform almost equally as well. The AMATs for the UDDC-A and UDDC-B are identical, and they both are slightly larger than the AMAT for the DDC. This marginal difference is caused by cross-interference of elements of C and elements of the first row of B with elements of the row of A that is reused across iterations of the middle loop, since the UDDCs do not have a separate temporal cache as the DDC. The loads of elements of the first row of B are predicted temporal because the stride varies when they are loaded.

However, when the cache capacity is 32KB, the DDC performs significantly worse than the other cache organizations. In this case the (aggregate) cache capacity is sufficient to hold the matrix B in cache, which means that most elements of B are now reused across iterations of the outer loop, though some of them are replaced by elements of A and C. The DDC, however, decides not to cache the matrix B because the smaller size of its temporal sub-cache would cause self-interference. The reason that both types of the UDDC perform slightly worse than the conventional cache is that they fetch an 8-byte block each time they miss an element of B since the loads of elements of B are predicted temporal, whereas the conventional cache fetches 32-byte blocks.

For *mm100* (Figure 5), however, the conventional cache outperforms the other cache organizations. The reason is that the conventional cache exploits the spatial locality exhibited by the references to the matrix B. When $B[i][j]$ is loaded, $B[i][j+1]$ is implicitly prefetched (unless $B[i][j]$ is the last element in a cache line) which in this case is not replaced before the next iteration of the middle loop. The dual data caches, however, are unable to detect and exploit this spatial locality because $B[i][j]$ and $B[i][j+1]$ are loaded by non-consecutive executions of the same instruction. They

IEEE COMPUTER SOCIETY

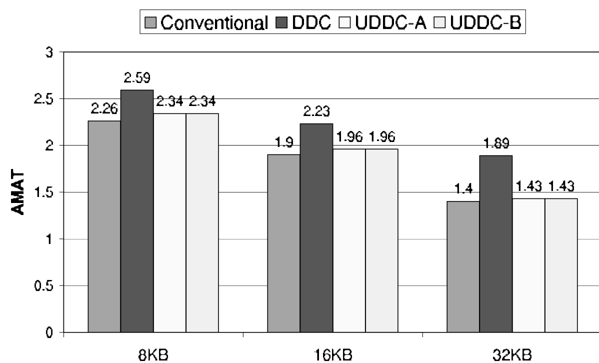

Figure 6. Average memory access times of the *lu80* kernel

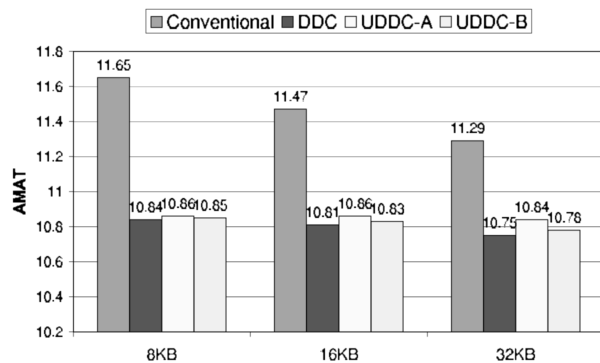

Figure 7. Average memory access times of the *fft-ct* kernel

predict that the loads of elements of B exhibit temporal locality (since $100 \times 8 = 800$ belongs to stride family 5 and $100 \times 2^5 = 3200$ is in all cases smaller than the (temporal) cache size). The matrix B is larger than the cache, however, so there is no chance to exploit temporal locality across iterations of the outer loop. We also observe that both types of the UDDC perform slightly worse than the DDC. This is due to conflicts between elements of B and elements of A.

So, for matrix multiplication the UDDCs only perform significantly better than the DDC when one of the matrices fits in the UDDCs but not in the temporal sub-cache of the DDC. In other cases, their performance is similar. The reason is that matrix multiplication does not significantly profit from a larger cache size, unless the cache can hold a whole matrix. During *LU* factorization, however, the size of the active sub-matrix $A[i : n-1][i : n-1]$ shrinks. Figure 6 depicts the AMATs obtained for the *lu80* kernel. It can be seen that the conventional cache as well as both UDDC types significantly outperform the DDC. Since the DDC uses (mainly) its spatial sub-cache to store the active sub-matrix whose size is half the size of the conventional cache and the UDDCs, the conventional cache and the UDDCs can keep a larger active sub-matrix in cache. The reason that the UDDCs perform marginally worse than the conventional cache is that in each iteration of the outer loop the matrix A is first accessed along a column and the conventional cache exploits the spatial locality exhibited by these references, while the UDDCs predict temporal locality for these loads.

The *fft-ct* kernel performs an FFT of a vector of $n = 2^{18}$ elements. It performs $m = \log_2 n$ iterations and during iteration i, it accesses the vector with a stride of 2^{m-i}. Figure 7 shows that for this kernel the dual data caches perform significantly better than the conventional cache. The reason is that they do not cache the vector until the stride is smaller than the larger block size. *fft-ct* also does not significantly profit from a larger cache size. Since the vector is significantly larger than the cache, there is not much reuse.

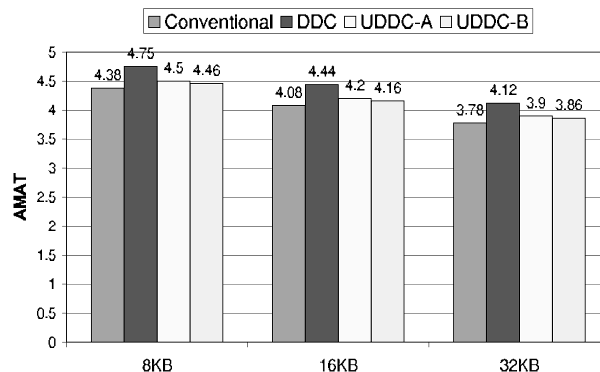

Figure 8. Average memory access times of the *fft18* kernel

However, there are also variations of the FFT that access the vector with unit stride. Figure 8 depicts the AMATs for such an implementation. Two conclusions can be drawn from this figure. First, when comparing it to Figure 7, this implementation is much more efficient than a direct implementation of the Cooley-Tukey algorithm. Second, in this case the conventional cache and the UDDCs outperform the DDC.

In general we observe that except for one kernel (*mm100*), the UDDCs never perform significantly worse than the most efficient cache organization, while the DDC sometimes does. Furthermore, the performance of the UDDC-A is somewhat remarkable. Even though the other sub-blocks in a block have to be invalidated when the tag of the requested block does not match the tag of the replacement block, its performance is always very close to that of the UDDC-B. Finally, we remark that some kernels exhibit spatial locality that the dual data caches are unable to detect, because adjacent words are accessed by non-consecutive executions of the same instruction.

0-7695-2003-0/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

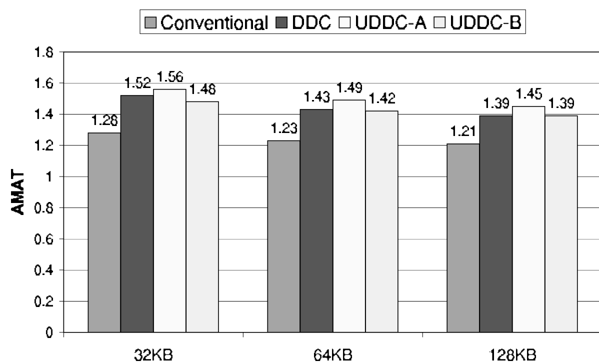

Figure 9. Average memory access times of the SPEC CFP 2000 benchmark *equake*

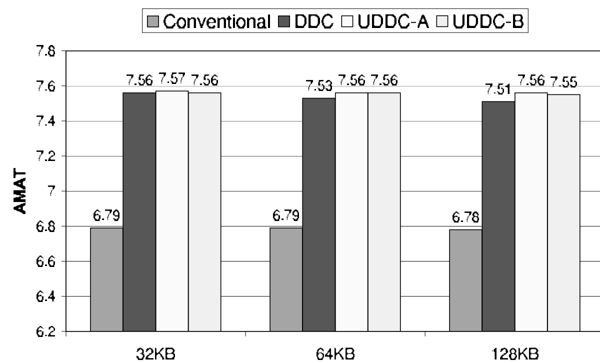

Figure 11. Average memory access times of the SPEC CFP 2000 benchmark *art*

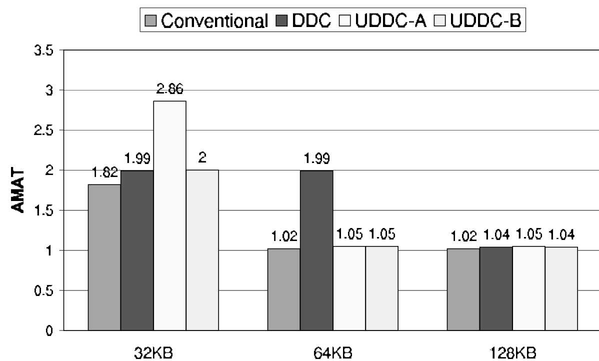

Figure 10. Average memory access times of the SPEC CFP 2000 benchmark *mesa*

4.3.2 SPEC Benchmarks

Figure 9 depicts the AMATs for the SPEC CFP 2000 benchmark *equake*. It can be seen that the conventional cache performs better than the dual data caches, and that the UDDC-B performs slightly better than the DDC. The performance of the conventional cache is already quite good for this benchmark, with a miss rate of less than 2.2% for the smallest cache size. Furthermore, the complete set of performance results shows that the dual data caches predict temporal locality for most load/store instructions. This indicates that there is spatial locality in *equake* that the dual data caches are unable to discover and exploit.

Figure 10 depicts the results obtained for *mesa*. When the cache size is 32KB, the DDC and the UDDC-B perform slightly worse than the conventional cache, but the AMAT of the UDDC-A is considerably longer. However, when the cache size is 64KB, the DDC performs significantly worse than the other organizations, and the conventional cache and the UDDCs perform almost identically. In this case the working set fits in the conventional cache and the UDDCs, but not in the temporal sub-cache of the DDC.

Figure 11 depicts the results for *art*. We first remark that the AMATs are rather high for this benchmark and hardly reduce when the cache size is increased. The AMAT of each direct-mapped cache, for example, corresponds to a miss rate of 44.5%. However, this more or less agrees with results obtained with the reference workload [2], which show that on average *art* exhibits miss rates of 35.6%, 35.2%, and 35.1% for direct-mapped caches of 32KB, 64KB, and 128KB, respectively. Second, the conventional, direct-mapped cache performs significantly better than the dual data caches, indicating once again that there is spatial locality that the dual data caches are unable to detect.

Finally, Figure 12 depicts the results obtained for *ammp*. For this benchmark the AMATs are also rather high as they correspond to miss rates of 30-32%. In this case this does not agree with results obtained using the full reference input [2]: direct-mapped caches of 32KB, 64KB, and 128KB exhibit miss rates of 6.9%, 5.9% and 5.0%, respectively. We conjecture that the reduced input decreases the number of memory references but its memory footprint is similar to the footprint of the reference input. In other words, the total size of all data structures used in this benchmark does not change significantly when the reduced input is employed but the number of memory references does. This happens, for example, when the number of loop iterations is altered to reduce the runtime. Because of this, a large number of misses incurred when the reduced input is used are most likely compulsory misses. We also observe that for this benchmark the dual data caches perform considerably better than the conventional cache and that the UDDCs perform slightly better than the DDC.

5. Concluding Remarks

We have presented two versions of the Unified Dual Data Cache. Both versions fetch smaller blocks from the next memory level on a miss if the locality prediction is tempo-

IEEE COMPUTER SOCIETY

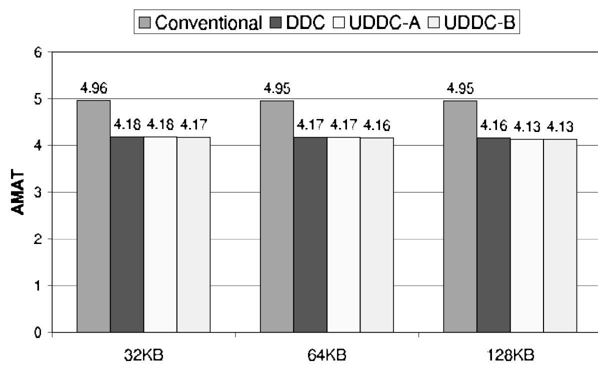

Figure 12. Average memory access times of the SPEC CFP 2000 benchmark *ammp*

ral and larger blocks if the prediction is spatial. The difference between the two versions is that each smaller block of the UDDC-B is tagged while in the UDDC-A the smaller blocks within a larger block share the tag.

For the kernels we have observed that in general the UDDCs never perform considerably worse than the best cache organization while the DDC sometimes does. The exception is *mm100*, for which the conventional cache performs better than the UDDCs as well as the DDC. The reason is that this kernel contains spatial locality that the dual data caches are unable to discover and exploit, because consecutive words in the same cache line are loaded by non-consecutive executions of the same instruction.

For the SPEC benchmarks we have observed that in general the conventional cache yields the best results. The DDC and the UDDCs only outperforms the direct-mapped cache for *ammp*. This might indicate that the other three benchmarks considered are not vectorizable (recall that the dual data caches are targeted at numerical/vectorizable applications), but a more plausible explanation is that these codes also contain spatial locality that the dual data caches are unable to detect. For example, the dual data caches are unable to detect the spatial reuse occurring when different fields of a structure (record) are accessed, because they are loaded by different instructions. To detect such cases requires a more substantial modification of the DDC, which is beyond the scope of this paper. For the *mesa* benchmark using 32KB caches, both types of the UDDC as well as the conventional cache perform much better than the DDC. In this case the working set fits in the conventional cache and the UDDCs but not in one of the sub-caches of the DDC, which is exactly the reason why the UDDCs have been proposed.

References

[1] D. Burger and T. Austin. The SimpleScalar Tool Set, Version 2.0. Technical Report 1342, Univ. of Wisconsin-Madison, Comp. Sci. Dept., 1997.

[2] J. F. Cantin and M. D. Hill. Cache Performance for SPEC CPU2000 Benchmarks. Available via `http://www.cs.wisc.edu/multifacet/misc/spec2000cache-data/`, May 2003.

[3] K. Chan, C. Hay, J. Keller, G. Kurpanek, F. Schumacher, and J. Sheng. Design of the HP PA7200 CPU. *Hewlett-Packard Journal*, 1996.

[4] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Computation of the Complex Fourier Series. *Mathematics of Computation*, 19:297–301, 1965.

[5] D. Cross. Fast Fourier Transforms. Webpage, 2002. Code can be downloaded from `http://www.intersrv.com/~dcross/fft.zip`.

[6] A. González, C. Aliagas, and M. Valero. A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality. In *Proc. Int. Conf. on Supercomputing*, pages 338–347, 1995.

[7] D. Harper III and D. Linebarger. A Dynamic Storage Scheme for Conflict Free Vector Access. In *Proc. Int. Symp. on Computer Architecture*, 1987.

[8] J. Hennessy and D. Patterson. *Computer Architecture - A Quantitative Approach*. Morgan Kaufmann, 2nd edition, 1996.

[9] T. L. Johnson, M. C. Merten, and W. W. Hwu. Run-Time Spatial Locality Detection and Optimization. In *Proc. Int. Symp. on Microarchitecture*, pages 57–64, 1997.

[10] S. Kifowit. Fast Fourier Transforms. Webpage, 2002. Code can be downloaded from `http://ourworld.compuserve.com/homepages/steve_kifowit/cfft.cpp`.

[11] A. KleinOsowski, J. Flynn, N. Meares, and D. J. Lilja. Adapting the SPEC 2000 Benchmark Suite for Simulation-Based Computer Architecture Research. In *Proc. Workshop on Workload Characterization, Int. Conf. on Computer Design (ICCD)*, 2000.

[12] H.-H. S. Lee and G. S. Tyson. Region-Based Caching: an Energy Efficient Memory Architecture for Embedded Processors. In *Proc. of the Int. Conf. on Compilers, Architecture, and Synthesis for Embedded Systems (CASES-00)*, pages 120–127, 2000.

[13] M. Prvulović, D. Marinov, Z. Dimitrijević, and V. Milutinović. Split Temporal/Spatial Cache: A Survey and Reevaluation of Performance. IEEE TCCA Newsletter, July 1999.

[14] A. Seznec. Decoupled Sectored Caches: Reconciliating Low Tag Volume and Low Miss Ratio. In *Proc. Int. Symp. on Computer Architecture*, 1994.

[15] M. Valero, T. Lang, J. Llaberia, M. Peiron, E. Ayguade, and J. Navarro. Increasing the Number of Strides for Conflict-Free Vector Access. In *Proc. Int. Symp. on Computer Architecture*, 1992.

[16] P. van Vleet, E. Anderson, L. Brown, J.-L. Baer, and A. Karlin. Pursuing the Performance Potential of Dynamic Cache Line Sizes. In *Proc. Int. Conf. on Computer Design*, 1999.

[17] A. V. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, and X. Ji. Adapting Cache Line Size to Application Behavior. In *Proc. Int. Conf. on Supercomputing*, pages 145–154, 1999.

IEEE COMPUTER SOCIETY