New Schemes of Cache Memory for Multimedia Applications

Shu-Lin Hwang
Dept. of Electrical Engineering
National Taiwan Universit
hwangsl@ccsun.mit.edu.t

Jing Liang Tsai
Dept. of Electrical Engineering
National Taiwan Universit
jltsai@bulls.csi .ntu.edu.tw

Feipei Lai
Dept. of Electrical Engineering &
Computer Science and Information
Engineering
National Taiwan Universit
flai@cc.ee.ntu.edu.t

Abstract

A recent confluence of hardware and software technologies has given computers the ability to process dynamic media data. Multimedia workloads such as MPEG-2 video player, 3D graphics and animations will occupy a large portion of the computer workload. While the processor speed increases rapidly, the required memory bandwidth for multimedia workloads is more critical than that for desktop workloads. This paper will focus on strengthening the memory system to suit multimedia workloads. Then, the Split Data Cache scheme that divides data cache into two parts (Stride and non-Stride) to makes better use of the spatial locality is proposed to enhance performance. The result shows that by adopting the Split Data Cache scheme, the miss rate can be reduced to 82%~87% of a conventional data cache. To further improve the performance, an Assisted Stream Buffer is proposed to hide the latency by prefetching data between the Stride data cache and the second level cache. By using the Stride History Table information, the Assisted Stream Buffer can yield an average hit ratio of 77.8%, with 12.5% extra bandwidth.

Keyword: Cache Memory, Stream Buffer, Multimedia, Workload

1. Introduction

The memory speed has not advanced as rapidly as the processor speed . The microprocessor performance has improved 55% per year since 1987 while the DRAM speed improved only 7% per yea r [1]. To mitigate the performance gap, cache memory has been introduced in the 1980s. Cache is a small amount of fast SRAM memory that keeps the recently referenced data. If the data is referenced again by the program before it is flushed out, the program can get the data from cache with a faster access time.

The cache performance is closely related to its configurations and its workloads. Studies showed that the commercial workloads are very different from the numeric workloads. A static locality analysis on numeric applications indicated that the self-temporary and self-spatial reuse of data is frequent in these applications [2]. It also showed that about 35% of the references exhibit only temporary locality. The Dual Data Cache [3] tries to make use of this characteristic by splitting data cache into two parts. Commercial workloads have been discussed in [4]. It suggests that the OLTP workload continues to benefit from larger on-board caches, up to 4-8MB range, but little benefit will be gained beyond thi s point. However, the multimedia workloads' characteristics are rarely discussed. In [5], an MPEG-2 software decoder is studied. This paper showed that a small set of tabular data contributes to more than 50% of the memory references, and the output data from decoder, which are only written once, are likely to pollute the cache.

Modern issues for improving memory systems can be divided into three major categories. The first one is the latency hiding technique. Contemporary microprocessors usually use two-level on-chip cache bridging between processor and memory. Other latency hiding techniques include a stream buffer [6], a prefetch buffer that brings the next block of data in advance to hide latency between level one cache and memory.

The second technique is to reduce cache misses. Cache misses have been categorized into three C's, that is, Compulsory, Capacity, and Conflict misses. There are several ways to reduce each category of misses, for example, set-associative caches, varying block sizes, victim cache [7], hardware and software prefetching [8]. Set-associative caches benefit from observed 2:1 cache rule of thumb, that a direct-mapped cache of size N has about the same miss rate as a 2 -way set-associative cache of size N/2 [1]. A larger block size can reduce compulsory misses, but it might increase conflict misses and capacity misses. The victim cache is a small fully associative cache block between a cache and its refill path. The victim cache contains only blocks that are discarded from a cache because of a miss. It is checked on a miss to see if it has the desired block before going to the next level of memory.

The third technique is to increase memory bandwidth as well as memory speed. Several companies have proposed new memory standards. For exampl e, SLDRAM (synchronous link DRAM) [9] can reach the speed of 400Mbit/s/pin with maximum I/O speed of 800MB/s, and DRDRAM (direct rambus DRAM) [10] reaches the speed of 800MBit/s/pin with maximum I/O speed of 1.6GB/s.

2. Related Work

We will review the related designs that have the potential to improve the performance under multimedia workloads while still fast enough to meet the modern processor speeds.

2.1 Stream Buffer

Jouppi's Stream Buffer [6] is a small prefetch buffer lies between the first level cache and second level cache. The buffer can hide the second level cache latency without affecting the hit time of the first level cache. Figure 1 shows the block diagram of a stream buffer.

When a cache miss occurs, the Stream Buffer starts to prefetch succes sive line at the miss target, set *Tag* field

to next address and *valid* bit to false. When the prefetched data return, they are stored and the *valid* bit is set to true. The prefetched line is only stored in the buffer, thus prevents the cache pollution problem. When accessing to the first-level cache we also compare their addresses with the first item of the Stream Buffer. If it is a miss in the cache but hit in the Stream Buffer, the cache can be reloaded in one cycle. One drawback is that only the head of the buffer has a tag comparator, thus non -sequential misses will flush the buffer. The results showed that the Stream Buffer can remove 72% of the instruction misses, but it can remove only 25% of the data cache misses. The reason is that data references tend to consist of interleaved streams of data from different sources, thus flushing between different streams degrades the Stream Buffer performance.

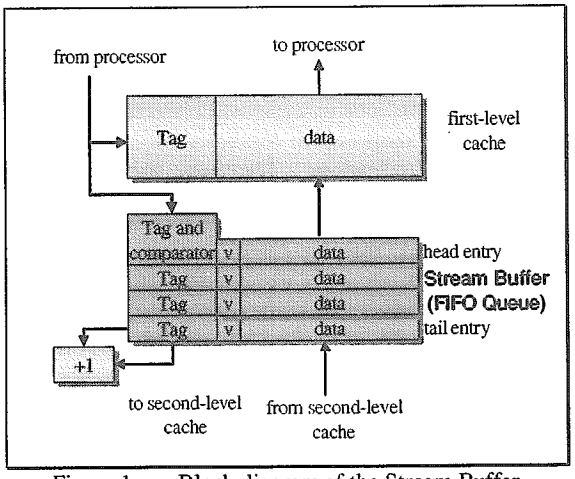

Figure 1. Block diagram of the Stream Buffer

To avoid the flushing problem, Jouppi also proposed a multi-way Stream Buffer that can handle several streams at the same time. It uses a multiplexor to switch between different buffers that contain different streams.

2.2 Stride Directed Prefetching

Because numerical programs often have poor cache performance, a Stride Directed Prefetching [12] method was proposed. A table called Stride Prediction Table (SPT) is used to direct the prefetching. The block diagram of the Stride Prediction Table is showed in Figure 2.

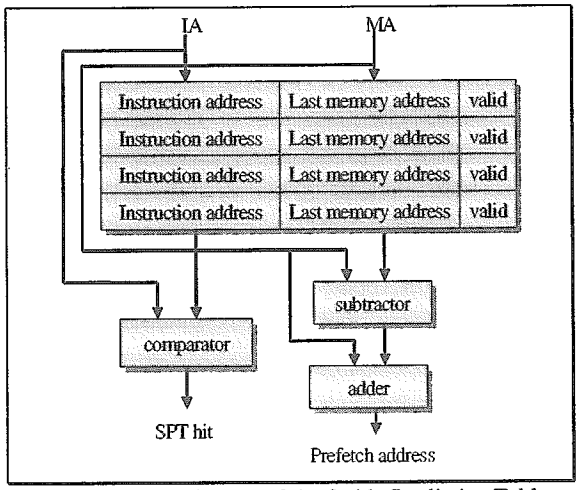

Figure 2. Block diagram of the Stride Prediction Table

The lower bits of instruction address (IA) are used to index the SPT. When a cache reference takes place, the current IA and IA field in the SPT entry are compared. The distance between memory address (MA) and last memory address in the entry is also calculated and added to the current MA. If it is a SPT hit, the new address is used as the prefetch address. If it is a miss, an entry in the SPT is selected and updated, and no prefetching will occur.

2.3 Selective Cache Line Replacement

A different caching method, the Selective Cache Line Replacement [17], is proposed. It, unlike the aggressive method such as prefetching that tries to bring data into the cache in advance, tries to bypass some types of data from the cache. This idea comes from the observation that a large percentage of data misses are caused by a very small number of instructions. In this method, instructions that are marked C/NA (Cacheable/Non-Allocatable) will not invoke the allocation policy of the hardware management algorithm. This is achieved by using a Miss Prediction Table. The paper evaluated several different cache schemes, but we will only describe the Improved Dynamic 2-bit Counter Scheme.

In the Improved Dynamic 2-bit Counter Scheme, each line of the cache has associated with it the address of the load instruction that brought that line into the cache. Each entry of the Miss Prediction Table contains Instruction Address and a 2-bit counter. On a cache hit, the 2-bit counter associated with the instruction that caused the hit is decremented and in addition, the 2-bit counter associated with the instruction that brought the cache line into the cache is also decremented. Thus, those instructions that do useful prefetching of data for other instructions are not marked C/NA. On a cache miss, if the 2-bit counter associates with the instruction in the Miss Prediction Table is in the highest state (11_2 state), the data will be sent directly from the second level cache to the processor without changing the data cache content.

The results show that although total memory references reduce only 1.36%, the required bandwidth reduces substantially for SPEC92 benchmarks. SPEC INT92 reduces the average required bandwidth from 21% to 23% for different cache configurations, and SPEC FP92 reduces the average required bandwidth from 5% to 10%. This is because if a reference is a non -cacheable one, it does not need to transfer the whole cache block from the lower level memory hierarchy to the processor, thus reduces the required bandwidth.

2.4 Dual Data Cache

The observation that different types of data exhibit different locality properties results in the design of Dual Data Cache. Scalar variables usually have high temporal locality and vectors with a small stride exhibit a very high spatial locality, while random accesses (i.e., sparse matrix computation) do not exhibit any types of locality. The Dual Data Cache consists of two independent blocks of cache, one is Spatial Cache, which exploits spatial locality, and the other one is Temporal Cache, which exploits temporal locality. Figure 3 shows the block diagram of the Dual Data Cache.

The Dual Data Cache uses a Locality Prediction Table to predict the data type of current memory reference. Different types of data will go to different caches by using a multiplexor and the Locality Prediction Table. The Spatial Cache and Temporal Cache need not have the same cache size. They can also have different block sizes, associativities, and replacement strategies. However, the

hardware will be more complex to ensure data consistency if different configurations are used. If a memory reference is predicted as having spatial locality, the required data might still be found in theTemporal Cache. This is because other temporal type of instructions might have brought the data into the Temporal Cache. Thus, two caches must be looked up at the same time when a memory reference occurs. The hardware of the Locality Prediction Table is showed in Figure 4.

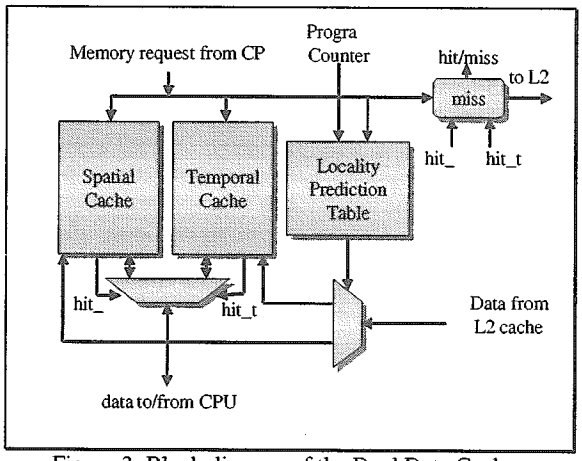

Figure 3. Block diagram of the Dual Data Cache

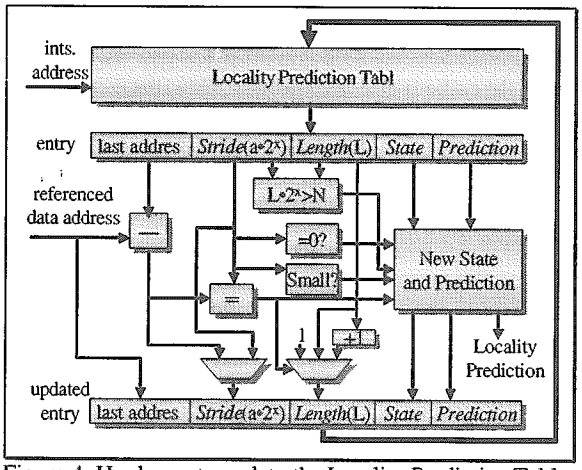

Figure 4. Hardware to update the Locality Prediction Table

Every time a memory reference occurs, the Locality Prediction Table is looked up. If the instruction address is not in the table, an entry is allocated to this instruction. Then it is initialized as follows: a) the address of the instruction, b) the address of the referenced data, c) *Stride* equal to zero, d) *Length* equal to one, e) *State* equal to Initial, and e) *Prediction* equal to Bypass. There are three states, Initial, Transient, and Steady. Initial state is reached after the first reference to a vector. Transient state is reached as long as two consecutive references have different strides. In these two states, the Locality Prediction is the default value (Bypass in the origin al paper). The third state, Steady state, indicates that successive strides are equal. In this state the Locality Prediction is given by the updated value of the *Prediction* field.

3. Simulation Environment and Benchmarks

The experiments were performed on SUN UltraSPARC family workstations with Solaris 2.5.1 operating system. UltraSPAC processor uses 64 bit SPARC V9 instruction set. The processor has 16K L1 instruction cache alone with 16K L1 data cache. The instruction cache is 2-way associative, and the block size is 32 bytes. The data cache is direct-mapped with 16 byte subblocks.

This paper focuses on multimedia workloads, and the non-commercial multimedia benchmark suite readily available is MediaBench [13]. This benchmark suite, proposed by UCLA, is a collection of public domain multimedia applications that is commonly used.

MediaBench is composed of complete applications coded in high-level languages. MediaBench 1.0 contains 19 applications culled from available image processing, communications and DSP applications. The components include:

JPEG: is a standardized compression method for full-color and gray-scale images.

MPEG: is the current dominant standard for high -qualit digital video transmission.

GSM: European GSM 06.10 provisional standard for full-rate speech transcoding, pr -ETS 300 036, which uses residual pulse excitation/long term prediction coding at 13kbit/s.

G.721 Voice Compression: Reference implementations of the CCITT (International Telegraph and Telephone Consultative Committee) G.711, G.721 and G.723 voic compressions.

PGP: PGP uses "message digests" to form signatures. message digest is a 128-bit cryptographically strong one-way hash function of the messa ge (MD5).

PEGWIT: A program for public key encryption and authentication.

Ghostscript: An interpreter for the PostScript language.

Mesa: is a 3-D graphics library clone of OpenGL. All display output functions were removed from the library and demo programs included in the package.

RASTA: A program for speech recognition that supports the following techniques: PLP,RASTA, and Jah -RASTA.

EPIC: Experimental image compression utility. The compression algorithms are based on a bi -orthogonal critically sampled dyadic wavelet decomposition and a combined run-length/Huffman entropy coder.

ADPCM: Adaptive differential pulse code modulation is one of the simplest and oldest forms of audio coding.

In the experiments, we select larger input files for some of the appli cations to enlarge the problem size for a more accurate result. The benchmarks were analyzed to ensure that they are as resistant as possible to compiler optimizations that might not translate into real world performance gains.

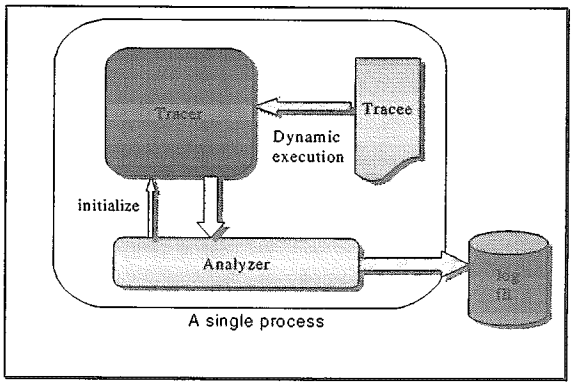

Figure 5. Program-driven simulation

We choose program-driven simulation (Figure 5) to save storage space. The simulation tools we use are Shade [15] and SpixTools [16] released by SUN. Shade is a program-driven simulation tool. We only need to initiali ze

Shade and supply the information of what instructions we interested in the beginning. Then, we write the analyzer procedure that handles these instructions. SpixTools is a collection of programs that provide instruction level profiling of user programs.

4. Multimedia Workloads

The impression of a multimedia application is that it has a small core that dominates the execution time. Figure 6 shows the relationship of static instructions and their cumulative contributions to dynamic execution. In Figure 6, we found that 13 out of 17 applications spend 70% of dynamic execution on less than 200 static instructions, which are less than 1% of the average static instruction count (average static instruction count of these 13 applications is 30000). However, the instructions that affect memory system design are those generate memory accesses. Figure 7 shows the relationship of static instructions and their cumulative contributions to total dynamic memory access.

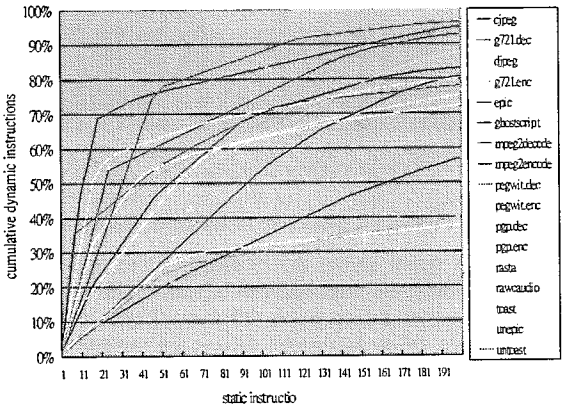

Figure 6. Core size analysis

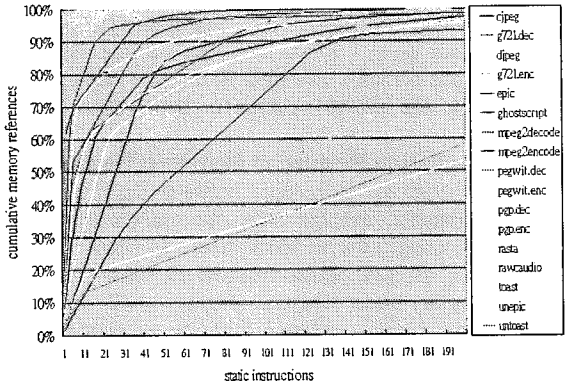

Figure 7. Memory reference analysis

In recent studies, researchers start to put emphasis on stream reference activity that occurs in execution time. Many new hardware schemes are proposed to make use of this phenomenon. The Spatial Locality Detection Table [11 is used to decide how many blocks to be fetched on each cache miss. The Stride Prediction Table [12] is used to decide whether prefetching or not on a cache miss. The Locality Prediction Table [3] is used to decide which part of the block should reside in a Dual Data Cache when data arrives from lower-level cache. The stream buffer also tries to benefit from stream references.

As previously described, we know that few static instructions generate most of the memory references. To capture the stream reference pattern, we use a Stride History Table (SHT) similar to the Stride Prediction Table, but with a two-bit saturation counter attached to each entry. The SHT scheme is presented in Figure 8.

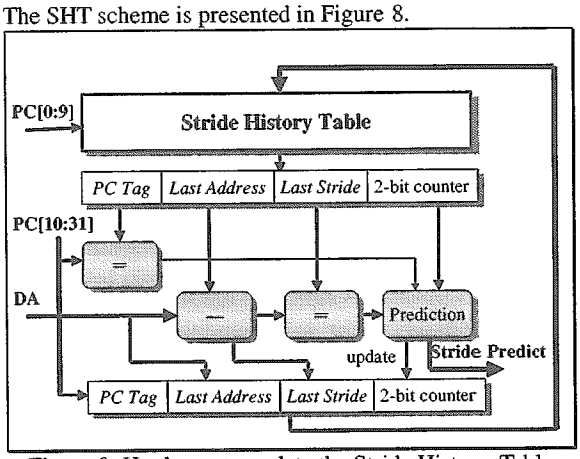

Figure 8. Hardware to update the Stride History Table

5. Proposed Architectures

5.1 Split Data Cache

The stream references take about 50% of total data references, which are easier to predict. We proposed a Split Data Cache scheme using Stride data cache for stride references and non-Stride data cache for no -stride references. The architecture is showed in Figure 9.

To meet the CPU speed, the SHT might need to be pipelined. When a hit occurs in data cache, new values for SHT entry are calculated, and these values will be updated at the next cycle. To prevent from getting the wrong value during computation, the data forwarding technique can be used. When a data miss occurs, the memory request is first issued to the second level cache. The SHT will have two or more cycles to make the prediction before dat a arrives from the lower level cache, there will be an abundant of time.

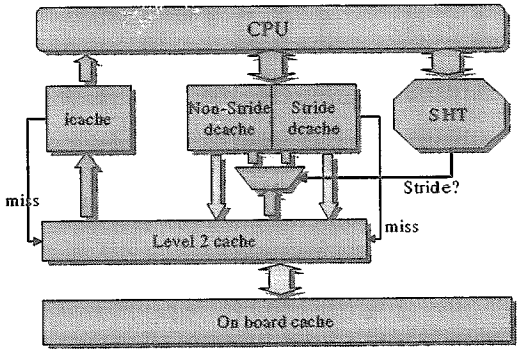

Figure 9. Block diagram of the Split Data Cache

In the Split Data Cache, data consistency property must be preserved. When a Split Data Cache miss occurs, different size of blocks might be brought in from the second level cache depending on the SHT information. If a block is brought into the Stride data cache, and some blocks in the non-Stride data cache lie in the range of this block, a consistency problem will occur. There are several ways to deal with this problem. If the blocks in the non-Stride data cache are not dirty, we can invalid the block by simply turning the valid bit off. If the blocks are dirty, we can proceed the write back operation to the second level cache first, but this might introduce extra delay in the memory hierarchy. One alternative is to add a bus between the Stride data cache and non -Stride data cache. If some blocks in the non-Stride data cache lie in the

range of a required block of the Stride data cache, the dirty blocks are copied directly to the non-Stride data cache and the valid bits are turned off. Some control signal can also be sent to the second level cache, and the returned data can skip these parts. Although it need s more hardware and control to achieve the goal, the bandwidth requirement between the first level cache and the second level cache can be relieved.

In our experiments, a data cache access will be treated as a miss if the data is not in the Split Data Cache, or the data can not be found in the Stride data cache when the access is predicted as a stream reference. In the second case, if the data is already in the non-Stride data cache, it is copied from the non-Stride data cache to the Stride data cache. Because the block size of the Stride data cache is larger than that of a non-Stride data cache, the rest of the block will still be retrieved from the second level cache. We call this a fast miss, since it can still provide a single cycle hit.

5.2 Assisted Stream Buffer

In a traditional cache, stream buffer might make too much useless prefetch, and waste the precious memory bandwidth. In a Split Data Cache system, we can prevent this situation by using the SHT information. We propose an Assisted Stream Buffer (ASB) to back up the Stride data cache by using the SHT information. The Assisted Stream Buffer can further improve the memory system performance by hiding the second level cache latency. The architecture is showed in Figure 10.

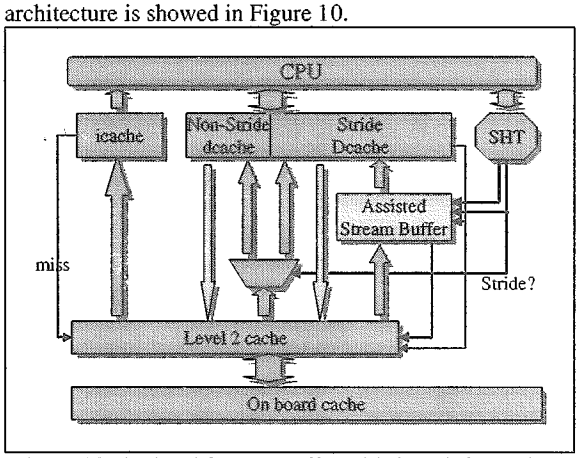

Figure 10. Assisted Stream Buffer with SHT information

The Assist Stream Buffer is an 8 -entry fully associative cache using the LRU replacement. It does not do anything until a Stride data cache miss occurs. When a Stride data cache miss occurs, the ssisted Stream Buffer gets the *Last Stride* value from the SHT and data address from the address bus. It then checks if the data is in the buffer, if so, returns the data to the Stride data cache. It also checks if the next data block or previous data block is in the buffer depending on positive or negative *Last Stride* value. If the block cannot be found in the buffer, and if the *Last Stride* value information from the SHT indicates that the instruction causing this data miss has a small stride (no larger than two block-size, e.g. 128 bytes in our experiments), the memory request is issued. If the instruction produces large stride references, no prefetch will take place.

6. Experiment Results

6.1 Split Data Cache

6.1.1 Simulation Results on Level One Cache

We choose a 32K 2-way instruction cache with 16K 2-way non-Stride data cache and 16K -way Stride data cache. To compare with the conventional cache design, a conventional cache with 32K 2-way instruction cache and 32K 2-way data cache is used as the baseline . Both architectures use 128K 2-way unified caches with 64 byte blocks and 32 byte subblocks in second level cache. Write-back, write-allocate, and back-invalidation are all simulated to approach the real performance. The baseline architecture uses 32 byte blocks in the first level cache, while the Stride data cache uses 64 byte blocks. Three block sizes (8 bytes, 16 bytes, and 32 bytes) for non -Stride data cache are used. To evaluate the potential of this architecture, Stride Prediction Tables with 2048 entries are used. The result is showed in Figure 11.

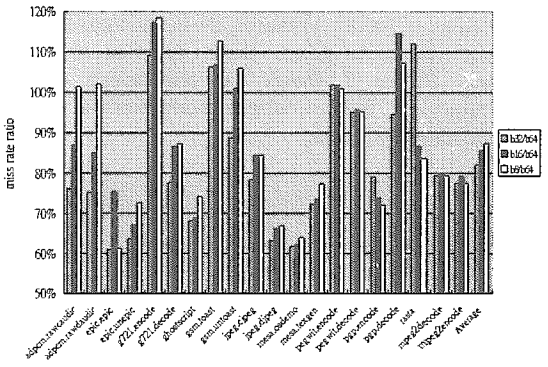

Figure 11. Miss rate of the Split Data Cache

The result shows that by adopting the Split Data Cache scheme, the miss rate can be reduced to 82%~87%, of a conventional data cache. Other result also shows that 1%~17% of Stride data cache misses are fast misses. The block size of the non -Stride data cache can affect the performance dramatically. The miss rate decreases obviously while the block size of the non-Stride data cache increase from 8bytes to 32 by es. In *rawcaudio*, using a 32 byte block non -Stride data cache can reduce the miss rate by 25%, but using a 8 byte block non-Stride data cache will increase the miss rate by 2 %. For the most of the benchmark, the b32/b64 scheme is better than b8/b64 scheme.

6.1.2 Simulation Results on Required Bandwidth

Although the differences between the applications are considerable, average miss rate can be reduced to 82%~87% using different block sizes. To choose a better block size, we will further consider the extra bandwit required by the three schemes. Figure 12 shows the required bandwidth compared with the conventional data cache.

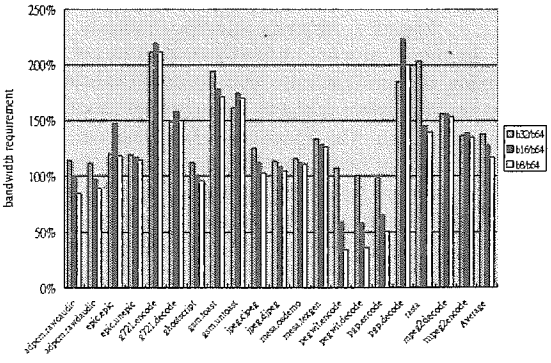

Figure 12. Required bandwidth for the Split Data Cache

In Figure 12, assume all Stride data cache misses need 64 bytes of bandwidth, although some fast misses

actually need less bandwidth. The result shows that the extra bandwidth required are 20% for 8 byte blocks, 30% for 16 byte blocks, and 38% for 32 byte blocks. We find that larger block can reduce the miss rate and needs some extra bandwidth, which means that these non -stream references in the program exhibit some types of spatial locality and the block size we choose should not be too small.

6.1.3 Simulation Results on Level Two Cache

To evaluate the overall performance gain, the impact on the second level cache must also be evaluated. The total second level cache misses are showed in Figure 13. The result shows a remarkable reduction on the miss count. Some applications even reduce their second level cache misses by 50% or more. One reason is that the Stride data cache fetches 64 byte blocks at one time, generates some of the misses.

Consider a small instruction core that accesses a large array with one-byte interval. In a conventional cache hierarchy, the first level cache misses will occur on location *32k, k: 0, 1, 2,...*, and it will be the same on the second level cache because we use 32 byte subblocks. But if the Split Data Cache is adopted, the misses of the level one cache will only occur on location *0, 3, 64k, k: 1, 2,...*, (cache miss on location 3 is a fast miss) in our example. The misses of the second level cache will occur on location *0, 32, 64k, k: 1, 2,....* The first miss will bring a 32 -byte subblock into the level two cache and the second miss will bring the other subblock. All the following misses will bring 64 byte blocks into the second level cache. This cuts the total miss count to about a half. Although the total bandwidth between the level two caches and main memory is still the same, it reduces the setup time for ac cessing main memory and makes the use of the memory bus more effective.

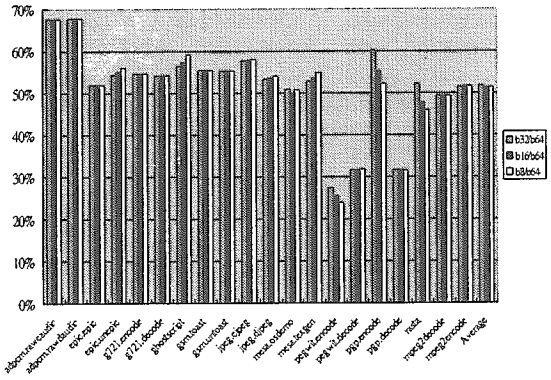

Figure 13. The second level cache misses using the Split Data Cache

6.1.4 Impact of Fixed-Size Stride History Table

To verify the effect on the size of the SHT, we change the table size from 64 entries to 2048 entries. In this experiment we fix the block size of non -Stride data caches to 16 bytes. The result is showed in Figure 14. In this figure we find that using different sizes of SHT will change the average miss rate ratio from 84 .89% to 92.37%. The average miss rate ratio in a 64 entry SHT is 90.81%, which means that if the hardware budget is limited, a 64 entry SHT can be used.

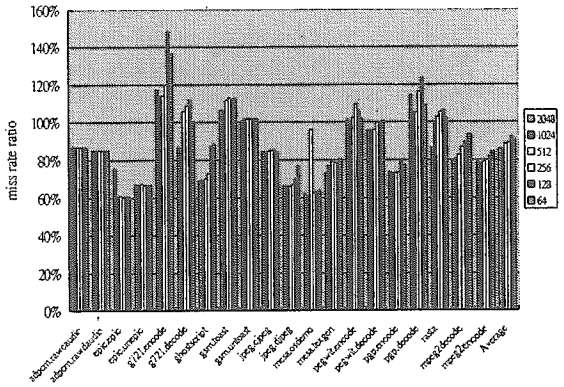

Figure 14. Impact of fixed-size SHT

6.2 Assisted Stream Buffer

The Stream Buffer will not reduce an miss rate, but it can hide the memory latency if the prefetched data is used later. Two parameters, buffer hit rate and bandwidth requirement, need to be evaluated. If the buffer has a high hit rate, but the buffer generates too much useless prefetch, total performance might be degraded because of the limited bandwidth. First, the Split Data Cache with the Assisted Stream Buffer (16 byte blocks for non-Stride data cache) is evaluated and the results are showed in Figure 15 and Figure 16. Figure 15 shows that Assisted Stream Buffer can benefit from the separated first level data cache and SHT information, and have about 77.8% hit rate which can significantly hide the latency between the Stride data cache and the second level cache.

Figure 16 shows that the extra bandwidth wasted on the useless prefetching are 12.5%. An observation shows that the useful prefetched data are never used within 16 cycles. Moreover, the average first-hit time on these data after the prefetching ranges from 10^2 to 10^5, which means that the Assisted Stream Buffer has abundant of time to wait for the second level cache to complete the prefetching request. This result also shows that hardware prefetching technique without using extra prefetch buffer will have a poor performance, because the prefetch will always replace blocks in the first level cache too early, and probably will increase the miss rate.

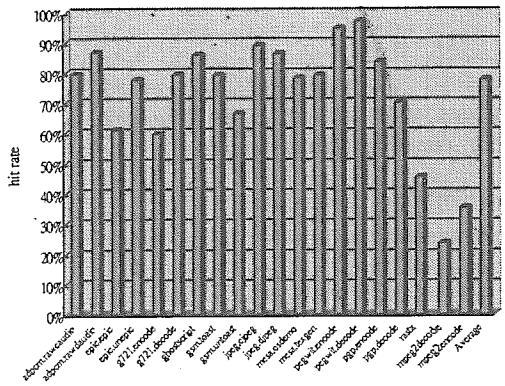

Figure 15. ASB hit rate of the Split Data Cache

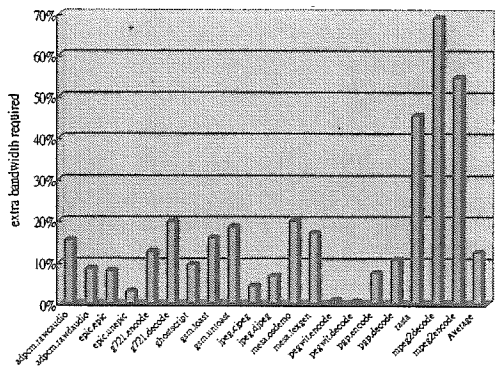

Figure 16. Increased bandwidth

7. Conclusion

The result shows that multimedia applic ations have small computation cores that generate most of the memory references. Based on this fact, the Stride History Table is used to record the program history and make useful prediction. The result also shows that a 64 entry SHT can capture 90% of the stream references by using 2048 entry SHT, which means that the hardware for SHT can be made small.

To benefit from the stream references, a Split Data Cache is proposed. The main idea of separating data cache into two parts is to make better use of the s patial locality. The Split Data Cache outperforms the conventional 2 -way data cache; it also outperforms the conventional 4-way data cache. The result shows that by adopting the Split Data Cache scheme, the miss rate can be reduced to 82%~87% of a conventional data cache. We found that the size of the non-Stride data cache can be reduced with only minor performance change.

To further improve the memory system performance, we proposed an Assisted Stream Buffer to hide the latency between the Stride data cach e and the second level cache. By using the SHT information, the Assisted Stream Buffer can yield an average hit ratio of 77.8%, with 12.5% extra bandwidth.

Although the new DRAM standards can provide bandwidth up to 1.6 GB/s, the setup time for accessing the main memory is still large. A SLDRAM has to wait for 32.5ns~77.5ns to get the first data depend ing on the accessing mode. In a 500Mhz CPU, it equals 17~39 computation cycles. By adopting the Split Data Cache scheme, we can reduce the miss count on the second level cache to 53%. Thus the total setup time spent on accessing memory can be reduced remarkably.

References

[1] John L Hennessy and David A Patterson, "Computer Architecture A Quantitative Approach, Second Edition," Morgan Kaufmann Publishers, inc.

[2] F. Jesus Sanchez, Antonio Gonzalez and Mateo Valero, "Static Locality Analysis for Cache Management," in Proceedings of the 3rd international conference on the Practical Application of Constraint Technology, Nov. 1997.

[3] Antonio Gonzalez, Carlos A iagas and Mateo Valero, "A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality," in Proceedings of the 2 nd international symposium on Computer Architecture, pp.338-347, 1995.

[4] Luiz Andre Barroso, Kourosh Gharachorloo and Edouard Bugnion, "Memory System Characterization of CommercialWorkloads," in Proceedings of the 25 th international symposium on Computer Architecture, June 1998.

[5] Peter Soderquist and Miriam Leeser, "Memory Traffic and Data Cache Behavior of an MPEG -2 Software Decoder," in Proceedings of the IEEE/ACM international conference on Computer-aided Design, pp.417-422, Oct. 1997.

[6] Jouppi, N.P., "Improving direct -mapped cache performance by the addition of a small full -associative cache and prefetch buffers," in Proceeding of the 17th international symposium on Computer Architecture, pp.364-373, May 1990.

[7] Subbarao Palacharla and R.E. Kessler, "Evaluating Stream Buffer as a Secondary Cache Relpacement," in Proceedings of the 21st international symposium on Computer Architecture, pp.24-33, April 1994.

[8] O. Temam, C. Fricker and W. Jalby, "Cache Interference Phenomena," in Proceedings of the 1994 conference on Measurement and modeling of computer systems, pp. 261-271, 1994.

[9] Peter Gillingham, MOSAID Techn ologies Inc., "SLDRAM Architectural and Functional Overview," SLDRAM Consortium, 1997.

[10] "RAMBUS ® TECHNOLOGY OVERVIEW," Rambus Inc., Feb. 1999.

[11] Teresa L. Johnson, Matthew C. Merten, and Wen-mei W. Hwu, "Run-time Spatial Locality Detection and Optimization," in Proceedings of the thirtieth annual IEEE/ACM international symposium on Microarchitecture, pp. 57-64, 1997.

[12] John W. C. Fu, Janak H. Patel, and Bob L. Janssens, "Stride Directed Prefetching in Scalar Processors," in Proceedings of the 25th annual international symposium on Microarchitecture, pp. 102-110, 1992.

[13] Chunbo Lee, Miodrag Potkonjak and William H. Mangione-Smith, "MediaBench: A Tool fo Evaluating and Synthesizing Multimedia and Communications Systems," in Proceedings of the thirtieth annual IEEE/ACM international symposium on Microarchitecture, pp.330-335, 1997.

[14] "SPEC CPU95 Press Release," Standard Performance Evaluation Corporation, Aug. 1995.

[15] "Introduction to Shade," Sun Microsystems, Inc., V5.33A, June 1997.

[16] "Introduction to SpixTools," Sun Microsystems, Inc., V5.33A, Feb. 1993.

[17] Gary Tyson, Matthew Farrens, John Mathhews, and Andrew R. Pleszkun, "Managing Data Caches Using Selective Cache Line Replacement," International Journal of Parallel Programming, Vol. 25, No. 3, 1997.