International Journal of Multimedia and Ubiquitous Engineering
Vol.10, No.8 (2015), pp.401-412
http://dx.doi.org/10.14257/ijmue.2015.10.8.39

Reconfigurable Parallel Multi-way Associative Cache with Miss-fetch Merge for Anisotropic Texture Filtering

Youngsik Kim

Department of Game and Multimedia Engineering, Korea Polytechnic University
kys@kpu.ac.kr

Abstract

Anisotropic texture filtering has been developed for a high quality three dimensional (3D) computer graphics and multimedia applications. This paper proposes an effective parallel multi-way set associative cache for anisotropic texture filtering, which can be adaptively reconfigured based on the number of probe samples. The proposed architecture also adopts a simple and efficient miss-fetch merge scheme to exploit the sequential access benefit. This paper constructs the trace-driven cache simulator in order to evaluate the proposed architecture. The benchmark suites for the simulation use OpenGL library. Experimental results show that the proposed 256KB cache architecture can reduce the memory access time of about 8.3% and 0.5% than the conventional 512KB and 1MB caches, respectively. Also, the hardware overhead is negligible.

Keywords: *anisotropic texture filtering, reconfigurable cache, miss-fetch merge*

1. Introduction

Various studies for a high quality 3D computer graphics and multimedia applications have been carried out in the field of the anisotropic texture filtering [1-5]. Anisotropic filtering is a method of enhancing the quality of texture images on 3D models that are at oblique camera viewing angles. In that case the texture image is projected to be non-orthogonal. Anisotropic filtering does not filter the same in every direction. Like bilinear and trilinear filtering, anisotropic filtering overcomes aliasing artifacts. It improves more on these other techniques by reducing blur and preserving detail at oblique camera viewing angles in Figure 1.

Anisotropic filtering can be the computing intensive job as well as the memory bandwidth job. However, the hardware implementation of the anisotropic texture filtering is often constrained by the limited memory bandwidth. Theoretical anisotropic filtering probes the texture in real-time on each pixel which is not the same in any orientation of anisotropy. In the hardware implementation of the anisotropic texture filtering, texture samples which are taken of the texture around the center point. The projected shape of the texture at that pixel depends on a sample pattern mapped. Each probe of anisotropic filtering can be considered as one trilinear MIP map filtering sample. A single trilinear MIP map filtering needs to take eight samples from two adjacent MIP levels. So sixteen probes in anisotropic filtering might require 128 texture samples. In that case the amount of computation and memory bandwidth might be sixteen times than those of a single trilinear filtering. However, the number of probes in anisotropic filtering does not need to be the maximum all the time. One common anisotropic filtering method on graphics hardware chooses the filtered pixel values from only one line of MIP map samples, which is referred to as "footprint assembly" [1].

ISSN: 1975-0080 IJMUE

Figure 1. A Trilinear Mipmapped Texture on the Left and the Same Texture Enhanced with Anisotropic Texture Filtering on the Right [17]

One anisotropic filtering method, called Footprint Assembly, is described in detail in [1]. In Figure 2, Footprint Assembly is explained. First, it approximates pixel footprint by parallelogram formed by the two side vectors, r1 and r2. And then, it roughly samples the area inside the parallelogram by assembling probes along the major axis, which is the larger of r1 and r2. The number of probes is calculated from the ratio of the major axis length to the minor axis which is the shortest among the side vectors, r1 and r2, and the diagonals, d1 and d2. Although Footprint Assembly can achieve somewhat high quality filtering images, but it needs a large amount of computation and memory bandwidth. In [1], they implement this directly into logic-embedded memories. Another method, called fast footprint MIP-mapping [2], adopts the prefiltered MIP-map data structure of currently available trilinear MIP-mapping implementations. The hardware realization of a filtering algorithm is described, which adapts the filter kernel to the shape of a pixel's texture space projection in [3]. A sub-texel precision anisotropic filtering, called shift-identification (SI) method [4], is implemented in hardware. In the SI method, the computation of the footprint coverage of each texel is proposed. The edge-function-based anisotropic texture filtering [5] is proposed, which approximates a footprint shape in the limited memory bandwidth. The prefetching texture cache architecture [6] and reconfigurable cache architectures [7, 8] are proposed in order to improve the cache memory performance. The selective z-test architecture [11] is proposed to reduce the memory bandwidth about texturing. However, the performance of hardware-based anisotropic texture filtering is still often constrained by the long miss-fetch latency of the texture cache.

This paper proposes an effective parallel cache architecture for anisotropic texture filtering, which can adaptively reconfigure multi-way set associativity based on the number of filtering samples. The proposed architecture also adopts a simple and efficient miss-fetch merge scheme to reuse a previous miss-fetch. This paper constructs the trace-driven simulator which modifies the software OpenGL library, called Mesa 3D [12]. The proposed 256KB cache architecture can reduce the memory access time of about 8.3% and 0.5% than the conventional 512KB and 1MB caches, respectively.

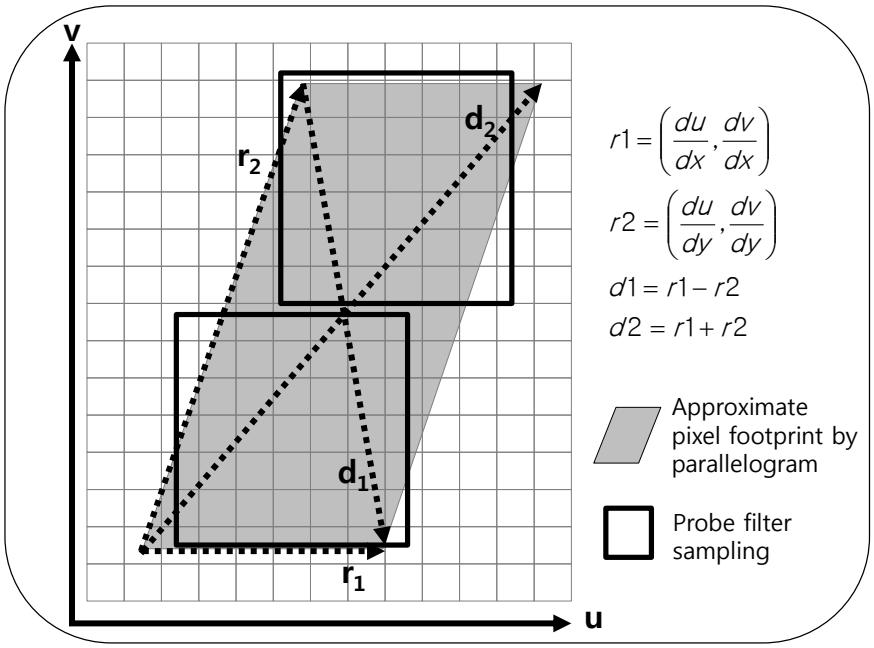

Figure 2. Footprint Assembly [1]

2. The Proposed Reconfigurable Parallel Cache Architecture

The proposed reconfigurable parallel multi-way associative cache architecture for anisotropic texture filtering is presented in Figure 3. The baseline architecture for anisotropic texture filtering has 16 texture units with its local cache. This architecture implements Footprint Assembly anisotropic filtering algorithm. The maximum number of filter probe samples is assumed to be 16, but there is no limit about the power of two unlikely the original algorithm [1]. According to the number of probes, the number of active texture units, which are the gray rectangles labeled as 'T' in Figure 3, can be changed. Also, the 16 local caches, which are the gray rounded rectangles labeled as '$', can be reconfigured as multi-way associative caches. The number of multi-way associative caches is equal to the number of probes, but the number of ways in each associative cache varies as in Algorithm 1 (Figure 4). For example, if the number of probes is 5, the proposed architecture consists of three 4-way associative caches and two 2-way associative caches. However, the conventional architectures have only five active texture units with its direct-mapped local cache under the same situation. The idea of reconfigurable multi-way set associative cache is already proposed in [7].

This paper proposes an effective way partitioning mechanism under the environment of parallel caches with the basic structure of [7] suitable for the anisotropic texture filtering. Also, the proposed architecture devises a simple and efficient scheme to merge miss-fetches, where the window size of cache TAG comparison is two. If the values of two TAGs are the same, the second miss-fetch can be omitted. If the values of two TAGs are different in a single cache block address, the second miss-fetch time can be much more reduced due to the sequential access.

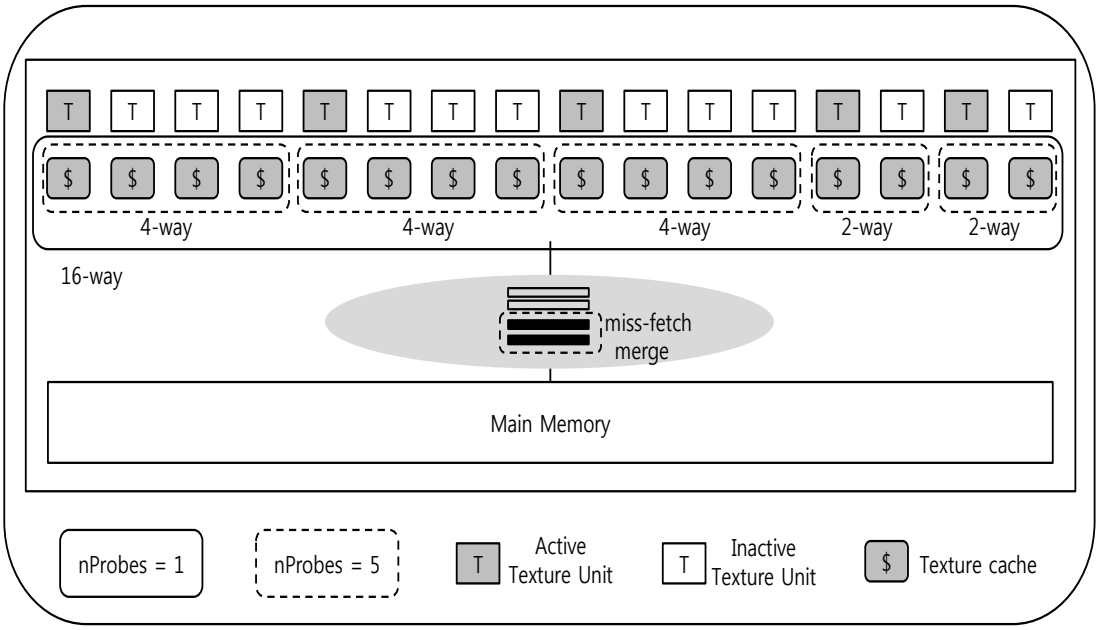

Figure 3. The Proposed Architecture of a Reconfigurable Parallel Multi-way Associative Cache

Algorithm 1: Proposed operational mechanism

N = the number of parallel texture units with its local cache;
while (rendering with anisotropic filtering)
{
  S = the number of Footprint Assembly probes for the current anisotropic filtering;
  for (k=0; k<=$\log_2$N; k++) // find smallest k for S>= N/2^k
  {
    if (S>= N/2^k)
      break;
  }
  if (k==0)
    Configure all N local texture caches as the direct mapped cache;
  else
  {
    p = (N/2^{k-1}) – S;
    q = S – p;
    for (i=0; i<p; i++) // configure p 2^k-way associative caches
      Sequentially configure 2^k local texture caches as the 2^k-way associative cache;
    for (i=0; i<q; i++) // configure q 2^{k-1}-way associative caches
      Sequentially configure 2^{k-1} local texture caches as the 2^{k-1}-way associative cache;
  }
}

Figure 4. Proposed Operational Mechanism

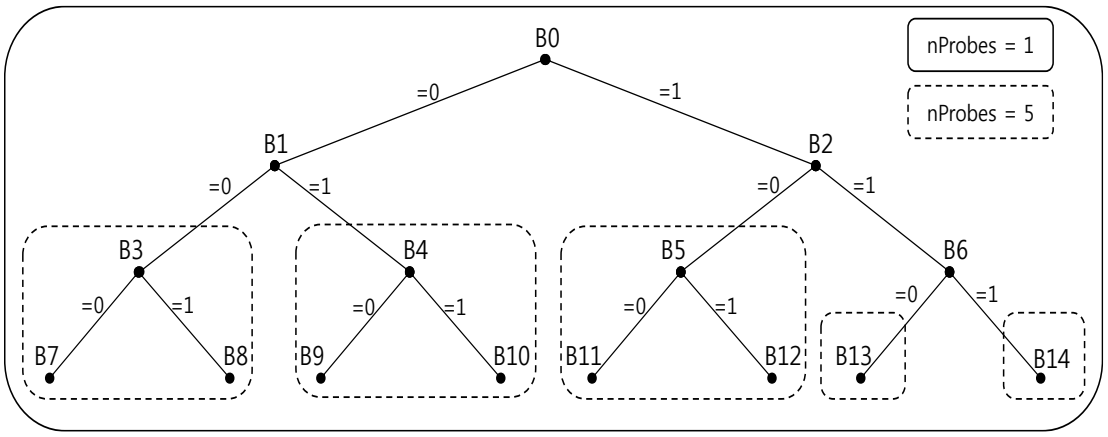

Figure 5. Pseudo LRU Tree Partitioning

The operational algorithm of way partitioning for the proposed cache architecture is shown in Algorithm 1. First, it finds the smallest number, k, where S is greater than or equal to $N/2^k$, S is the number of probes, and N is the number of texture units. If k is 0, then S is equal to N and the proposed architecture consists of N active texture units with its directed mapped cache. Otherwise, the proposed cache architecture consists of p 2^k-way associative caches and q 2^{k-1}-way associative caches, where S = p + q and p = $(N/2^{k-1}) - S$. For example, if the S = 5 and N = 16, then k = 2, p = 3, and q = 2 as shown in Figure 3. The small lookup table which stores pre-calculated way partitioning results can be used for the hardware implementation. So, both the hardware overhead and the computation time for Algorithm 1 are negligible.

In order to apply the original pseudo LRU algorithm to this reconfigurable parallel cache, this paper proposes the pseudo LRU tree partitioning as shown in Figure 5. Pseudo-LRU is a cache replacement algorithm which nearly simulates the Least Recently Used (LRU) algorithm. The tree based pseudo LRU algorithm is an algorithm to select an item that most nearly has been accessed least recently, given a set of items and a sequence of access events to the items. The CPU cache of the Intel 486 and many processors in the Power Architecture family uses this pseudo LRU algorithm. The pseudo LRU algorithm in two-way set associative requires one bit to indicate which line of the two has been referenced more recently. As such, three bits are used in four-way set associative cache. Each bit represents one branch point in a binary search tree for the items in question. The bit 1 represents that the left side has been referenced more recently than the right side, and the bit 0 vice-versa. To update the tree with an access to the specific node, traverse the tree to find the node and, during the traversal, set the node flags to denote the direction that is opposite to the direction taken.

This paper applies this technique to the proposed architecture. According to the number of probes, S, the original pseudo LRU tree is partitioned into several sub-trees, where the number of sub-trees is S. Each pseudo LRU sub-tree is sequentially assigned to a corresponding multi-way associative cache. In Figure 5, five sub-trees are shown where S = 5 in dotted line. Also, a whole single tree is shown where S = 1 in full line.

3. Simulation Environment

The architectural models for the simulation are shown in Table 1. A, B, C, and D modes are the 1-way or 16-way conventional caches with 512KB or 1MB. E and F modes are the conventional reconfigurable caches with 512KB or 1MB in [7]. This paper assumes that the result from E and F models is chosen from the best of all set-associativity results. The P model has the proposed reconfigurable cache mechanism with 256KB, which adopts the miss-fetch merge scheme with 1 cycle comparison overhead.

The cache block size is 32B or 64B and the cache access time is 2 cycles. The main memory latency is same in [10].

This paper constructs the trace-driven cache simulator in order to evaluate the proposed architecture. The benchmark suites for the simulation use OpenGL library as 3D graphics API in Table 2, Table 3, and Figure 6. OpenArena [13], Nexuiz [14], and Xonotic [15] are well-known first person shooting 3D games. OpenBve [16] is the train simulator with multiple maps and trains. In order to generate traces, the simulator modifies the software OpenGL library, called Mesa 3D [12], where the Footprint Assembly anisotropic filtering is called whenever a linear filtering call occurs in benchmark suites. In Table 3, there are various distributions of the number of probes for four benchmarks. Especially for the benchmark OpenArena, most of the texture units are active because the proportion of the number of probes 16 is much greater than others.

4. Performance Evaluation

Figure 7 shows the memory access time for various architectural models, where performance results are classified into two cache block sizes, 32B and 64B. For seven architectural models, the memory access time can be broken down into the miss-fetch time and the fixed cache access time of two cycles.

Table 1. Architectural Models for the Simulation

architectural model	A	B	C	D	E [7]	F[7]	P
reconfigurable scheme	1-way fixed	16-way fixed	1-way fixed	16-way fixed	conventional reconfigurable		proposed
cache size	512KB		1MB		512KB	1MB	256KB
texture unit	16 texture units with local texture cache						
cache	16 local caches with 2 cycles access latency						
cache block size	32, 64 bytes						
miss-fetch merge overhead	no						1 cycle
main memory latency	first latency – next chunk latency (54 – 2 cycles)						

Table 2. Benchmark Suites

Benchmark	Description
OpenArena[13]	A first person shooter based on Quake III Arena death-match style
Nexuiz [14]	An arena first person shooter with great graphics and intense gameplay
Xonotic [15]	A fast-paced first person shooter. It combines addictive, arena-style gameplay with rapid movement and a wide array of weapons
OpenBve [16]	An highly customizable train simulator for support with multiple maps and trains

(a) OpenArena[13] (b) Nexuiz[14]

(c) Xonotic[15] (d) OpenBve[16]

Figure 6. Screen Shot Images of Benchmark Suites

Table 3. Distribution of the Number of Probes for Benchmark Suites

Benchmark	Distribution of the number of probes (%)															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
OpenArena[13]	4.7	5.7	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	89.3
Nexuiz[14]	83.7	4.3	0.6	0.6	0.7	0.4	0.2	0.3	0.1	0.1	0.1	0.4	0.0	0.1	0.4	8.1
Xonotic[15]	54.9	5.1	3.0	1.7	1.2	1.2	1.1	0.8	0.8	0.7	0.5	0.5	0.5	0.4	0.5	27.2
OpenBve[16]	70.6	14.0	1.0	13.5	0.4	0.3	0.0	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

The proposed architecture, model P, shows the lowest memory access time among seven models for all benchmarks as shown in Figure 7. Even if the texture units are highly active, the proposed architecture can exploit the local caches of the small portion of inactive texture units. The proposed 256KB cache architecture can reduce the memory access time of about 8.3% and 0.5% than all other caches with 512KB and 1MB, respectively. Also, the proposed model P can reduce the memory access time of about 6.1% than the model E with 512KB cache in [7]. But the model P is slightly inferior to the model F with 1MB cache in [7] by -1.5% for the memory access time. In all benchmarks except the benchmark Xonotic [15], the proposed model P provides the best performance.

Figure 8 presents the cache hit ratio for seven architectural models about four benchmarks in two cache block sizes. The proposed 256KB cache architecture can provide the similar cache hit ratios of about 0.2% and -0.3% to those of the conventional 512KB and 1MB caches, respectively. Even when having the similar cache hit ratios, the reduction of the memory access time in the proposed architecture is due to the miss-fetch merge scheme.

In order to evaluate the effect of miss-fetch merge scheme for other architectures, in Table 4, the architectural models are configured. A, B, C, and D modes are the conventional caches with 512KB or 1MB. E and P models have the proposed reconfigurable cache mechanism with 512KB. Also, B, D, and P models have the miss-fetch merge scheme. Models with the miss-fetch merge scheme also have 1 cycle comparison overhead.

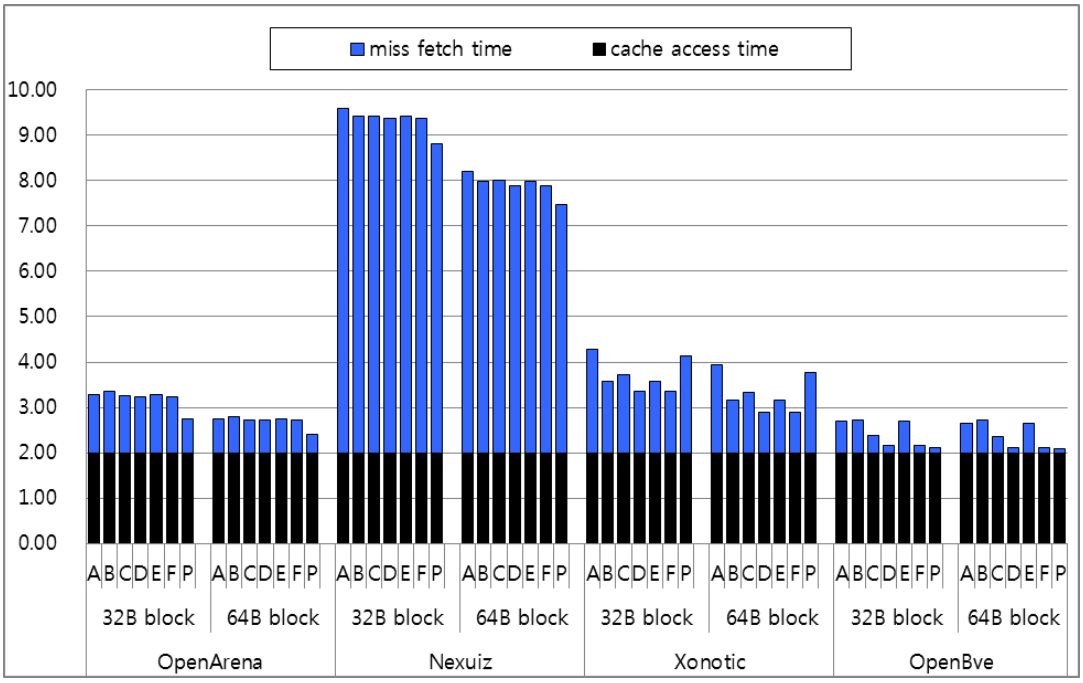

Figure 7. The memory Access Time (cycles) of Architectural Models in Table 1

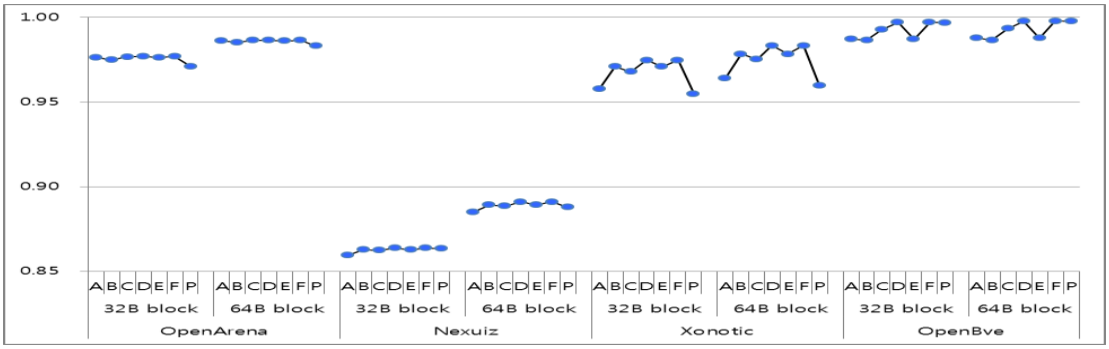

Table 4. Architectural Models for Evaluating the Effect of Miss-fetch Merge Scheme

architectural model	A	B	C	D	E	P
reconfigurable scheme	No, 1-way fixed				yes	
cache size(total-local)	512KB-32KB		1MB-64KB		256KB-16KB	
miss-fetch merge	no	yes	no	yes	no	yes

texture unit	16 texture units with local texture cache
cache	16 local caches with 2 cycles access latency
cache block size	32, 64 bytes
miss-fetch merge check	1 cycle overhead
main memory latency	first latency – next chunk latency (54 – 2 cycles)

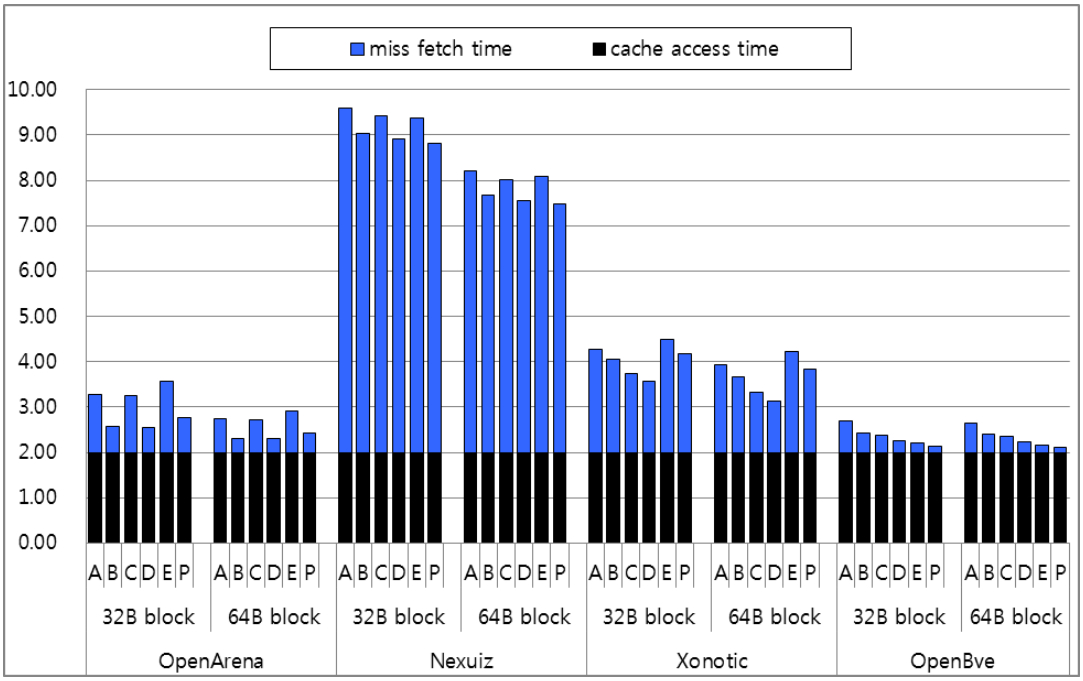

Figure 9. The Memory Access Time (cycles) of Architectural Models in Table 4

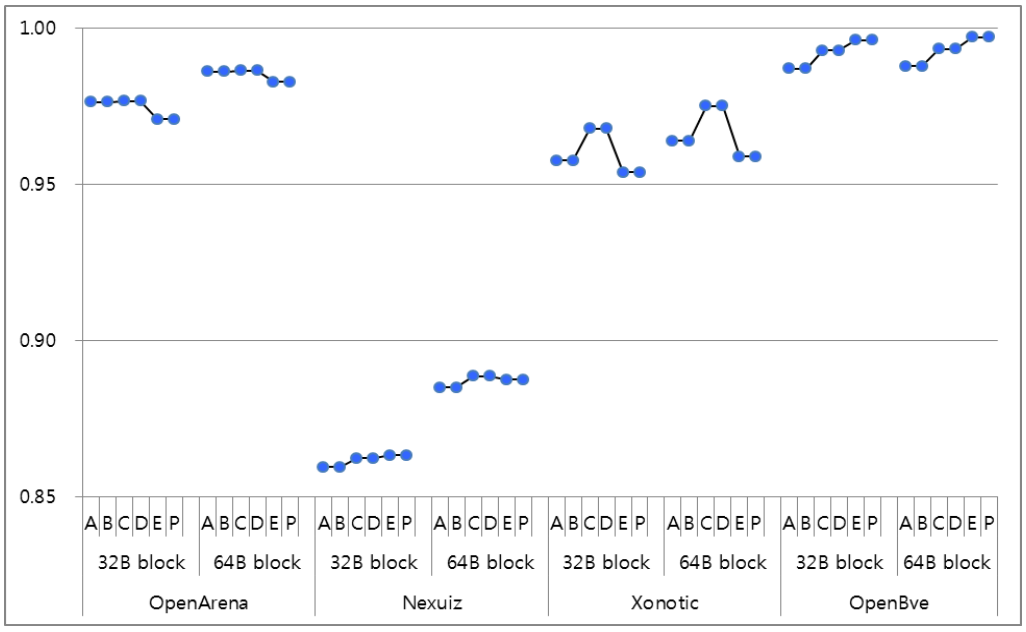

Figure 10. The Cache Hit Ratios of Architectural Models in Table 4

Figure 9 shows the memory access time for various architectural models defined in Table 4 in order to evaluate the effect of miss-fetch merge scheme for other architectures. Similar to Figure 7 and figure 8, for six architectural models, the memory access time can be broken down into the miss-fetch time and the fixed cache access time of two cycles. The proposed architecture, model P, shows the lowest memory access time among six models for all benchmarks as shown in Figure 9. The proposed 256KB cache architecture can reduce the memory access time of about 6.2% and -0.5% than the conventional 512KB and 1MB caches, respectively. The proposed 256KB cache architecture can provide the slightly better or similar performance than the conventional 512KB and 1MB caches, respectively. Also, models B, D, and P with the miss-fetch merge scheme can achieve a performance improvement of about 9.4% than models A, C, and E without the miss-fetch merge scheme.

Figure 10 presents the cache hit ratio for six architectural models of Table 4 about four benchmarks in two cache block sizes. There is no change about the cache hit ratio between models with or without the miss-fetch merge scheme. The proposed 256KB cache architecture can provide the similar cache hit ratios of about 0.1% and -0.4% to those of the conventional 512KB and 1MB caches, respectively.

5. Conclusion

This paper proposes a reconfigurable parallel multi-way set associative cache architecture for anisotropic texture filtering, which adopts a simple and efficient miss-fetch merge scheme. This paper constructs the trace-driven cache simulator in order to evaluate the proposed architecture. The benchmark suites for the simulation use OpenGL library. Experimental results show that the performance of the proposed 256KB architecture is superior to the conventional 512KB and 1MB cache architectures, respectively, about the memory performance. Also, the hardware overhead is negligible.

References

[1] A. Schilling, G, Knittel, and W. Strasser, "Texram: A smart memory for texturing", IEEE Computer Graphics and Applications, vol. 16, no. 3, (**1996**), pp. 32-41.
[2] T. Hüttner and W. Straßer. "Fast footprint mipmapping", Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, (**1999**).
[3] J.P. Ewins, M.D. Waller, M. White, and P.F. Lister, "Implementing an anisotropic texture filter." Computers & Graphics, vol. 24, no. 2, (**2000**), pp. 253-267.
[4] M. Bóo and M. Amor. "High-performance architecture for anisotropic filtering", Journal of Systems Architecture, vol. 51, no. 5, (**2005**), pp. 297-314.
[5] H. C. Shin, J.A. Lee and L.S. Kim, "A cost-effective VLSI architecture for anisotropic texture filtering in limited memory bandwidth", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 14, no. 3, (**2006**), pp. 254-267.
[6] H. Igehy, M. Eldridge, and K. Proudfoot, "Prefetching in a texture cache architecture", Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, (**1998**).
[7] William E. Coyle, David W. Nuechterlein, Kim E. O'Donnell, Thomas A. Sartorius, Kenneth D. Schultz, and Emmy M. Wolters, "Reconfigurable multi-way associative cache memory", U.S. Patent No. vol. 5,367,653, (**1994**) November 22.
[8] J. T. Battle, "Reconfigurable texture cache." U.S. Patent No. 6,002,410, (**1999**) December 14.
[9] M. B. Smith and M. J. Tresidder, "Pseudo-LRU cache memory replacement method and apparatus utilizing nodes", U.S. Patent No. 5,594,886. 14 January (**1997**).
[10] S.Thoziyoor, "CACTI 5.3," HP Laboratories, Palo Alto, CA, (**2008**).
[11] J. Park, "A pixel pipeline architecture with selective z-test scheme for 3D graphics processors." Microprocessors and Microsystems, vol. 37, no.3, (**2013**), pp. 373-380.
[12] Mesa 3D, http://www.mesa3d.org
[13] OpenArena, http://www.openarena.ws/smfnews.php
[14] Nexuiz, http://www.alientrap.org/games/nexuiz
[15] Xonotic, http://www.xonotic.org
[16] OpenBve, https://sites.google.com/site/openbvesim/home
[17] Anisotropic filtering in Wikipedia, http://en.wikipedia.org/wiki/Anisotropic_filtering

Author

Youngsik Kim, he received the B.S., M.S., and Ph.D. degree in Dept. Computer Science from the Yonsei University, Korea, in 1993, 1995, and 1999 respectively. He had worked for System LSI, Samsung Electronics Co. Ltd from Aug. 1999 to Feb. 2005 as a senior engineer. Since March 2005 he has been working for Dept. of Game & Multimedia Engineering in Korea Polytechnic University. His research interests are in 3D Graphics and Multimedia Architectures, Game Programming, and SOC designs.