Tile/Line Dual Access Cache Memory based on Hierarchical Z-order Tiling Data Layout

by

BaoKang Wang

(415DB53)

The Division of Systems Engineering,
Graduate School of Engineering, Mie University.

Tsu, Mie

March, 2018

APPROVED BY:

Dr. Toshio Kondo (Chair of Advisory Committee)
Dr. Tetsushi Wakabayashi
Dr. Yoshikatsu Ohta

ABSTRACT

The increasing disparity between the data access speed of cache and processing speeds of processors has recently caused a major bottleneck in achieving high-performance 2-dimensional (2-D) data processing, such as that in image processing and scientific computing. However, multi-core and single instruction multiple data (SIMD) extensions are current technologies that can effectively improve the processing speeds of processors. Even though 2-D data are generally arranged in row-major layout (C language) or column-major layout (Fortran language) in memory, single-directional layouts have poor locality for 2-D data access because vertically adjacent data are stored far apart. As a result, the translation lookaside buffer (TLB) misses frequently occur and the excessive data transfer problem cannot be avoided by using conventional caches. Therefore, ineffective non-major-directional data access to cache memory has become a bottleneck for efficient 2-D data processing by utilizing extended SIMD instructions.

This thesis proposes a cache memory with both tile (column and row directions) and line (row direction) accessibility for efficient 2-D data processing to solve this problem. The proposed cache is based on a 4-level Z-order tiling data layout and a multi-bank memory array structure that supports skewed array storage schemes. 2-D data access to the proposed cache memory is enabled via a hardware-based multi-mode address translation unit that eliminates the overhead of software-based address calculation and transforms the conventional raster layout into the 4-level Z-order tiling data layout. The proposed layout maximizes utilization of the 2-D reference locality, minimizes the TLB misses, and reduces the amount of excessive data transfer by dividing the data into hierarchical 3-level tiles (the 1st-level is a 2048×2048 byte-sized large tile, the 2nd-level is a 64×64 byte-sized medium tile and the 3rd-level is an 8×8 byte-sized tile), each of which is arranged in raster scan order at the same level.

In addition, the author has added a dual 1-D/2-D data access mode to the proposed cache. The proposed cache can appropriately switch a 2-D access mode for the proposed tile and line access to a 1-D data access mode for conventional raster line access. Therefore, the proposed cache can be used for both 1-D and 2-D data processing. The author proposes a method of reducing tag memory to replace multiple tiles with an aligned tile set (RATS) in the cache to reduce the hardware overhead of the proposed cache. The RATS method greatly reduces the entire hardware scale and simplifies the cache architecture, even though it provides almost the same cache performance.

The author evaluated the proposed cache in two ways: First, to verify the feasibility of the proposed cache, a large-scale integration (LSI) layout of a SIMD-based general purpose-oriented datapath that embedded the proposed cache was designed in a 2.5×5 mm^2 area using 0.18 µm complementary metal oxide semiconductor (CMOS) technology. The read latency was limited to 3 clock cycles under a 3.9 ns clock period (250 MHz), which was the same as that for the conventional cache memory of an Intel or advanced reduced instruction set computer (RISC) machine (ARM) high-performance

processor. The entire hardware overhead of the proposed RATS-cache was reduced to only 7% of that required for a conventional cache by using the RATS method.

In addition, the author evaluated the performance of the proposed cache for matrix multiplication (MM) and LU decomposition (LUD) in terms of the number of L1, L2 cache and TLB misses and the execution time (cycles). The results from simulation for the proposed cache indicated a considerable reduction in L1 and L2 cache conflict misses compared with a conventional cache in power-of-two matrix size due to the column-directional address stride being sufficiently smaller than the page size. In addition, the proposed cache significantly improved the TLB performance compared to a conventional cache with all matrix sizes. Therefore, the proposed cache provided efficient column-directional parallel access that was the same as row directional parallel access so that it enabled efficient SIMD operation that required no transposition in MM. The proposed cache could provide almost the same performance as LUD for the column-major based LUD program as that for the row-major based LUD program. These results indicated that the proposed cache did not restrict our freedom in selecting either row- or column-major order coding. In addition, the results from simulation also revealed that the RATS cache and the Non-RATS cache (the proposed cache did not adopt the RATS method) provided almost the same performance, which was not inferior to that of the conventional cache.

Finally, the number of parallel load instructions required for parallel column- and row-directional access was reduced to about one fourth of that required for conventional raster line access for MM. Since the performance of the proposed cache was also affected by the performance of L1 and L2 caches, the execution time for parallel load instruction was about one third of that required for conventional load instruction. The proposed cache with tile/line accessibility further improved the performance of 2-D applications by using SIMD instructions.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and related work

Although computer performance has been considerably improving year by year, the demand for efficient 2-D data processing, such as that in numerical calculations, image processing, object recognition, and video coding, continues to increase. Improvements to processor and memory access speeds are important keys to improve the performance of 2-D data processing. Multi-core and single instruction multiple data (SIMD) extensions are current technologies that can effectively improve processing speeds [1] [2]. However, sufficiently efficient data access for parallel processing has not yet been obtained because conventional cache memory, which is based on either row- or column-major data layouts (raster or single-directional layouts), does not enable efficient operations to spatially reference 2-D data, such as those used in image processing or scientific computing. For example, conflict misses and translation lookaside buffer (TLB) misses have frequently occurred in large-scale matrix multiplication (MM) because the cache memory does not contain all the data needed for the program's execution or exploit 2-D data locality in the column direction. In addition, the raster layout is inefficient to access two-dimensionally localized block data in image processing, and the excessive data transfer problem cannot be avoided [3] [4].

A software-based, single-levels tiling technique (also known as blocking) has been used to effectively improve 2-D data access capability to solve these problems. This technique allows the cache to exploit temporal locality; therefore, this tiling technique can reduce the cache capacity miss rate. However, 2-D data processing by using SIMD operations is restricted because there is no availability of a column-directional parallel access function so that performance degradation cannot be avoided if it does not use manually or ingeniously optimized routines such as the OpenBLAS or the Intel math kernel library (MKL). Also, the tiling technique does not effectively reduce the number of TLB misses and the amount of excessive data transfer for two-dimensionally localized block data access. User-transparent tile-based addressing, on the other hand, which requires no address translation in the user software, relies on the compiler to restructure the loop code, which causes additional overhead in the execution time because the cost of tiling address computation cannot be avoided [5] [6] [7].

Hierarchical tile based addressing on the Z-Morton layout [8] is a more effective technique than conventional single-level tiling in terms of reducing column-directional conflict misses and TLB misses in the processing of power-of-two sized matrices. The Z-Morton layout, which is, a potential compromise between row- and column-major layouts [9] [10] can minimize conflict misses and TLB misses in both row and column directions and utilize the spatial reference locality, regardless of the data array size. However, address calculation for the Morton layout is considerably more complex than that for the row- or column- major layouts, which causes significant processing overhead. As such, Morton address calculation imposes a software-overhead cost to exploit the 2-D spatial reference locality benefits of the Z-Morton layout.

Wittenbrink and Somani [11] proposed a hardware approach with three important advantages over the software approach: 1) the image storing and transfer mechanisms, such as direct memory access (DMA) or I/O devices, are independent of the tiling scheme, 2) a small tile can be used to process images of an arbitrary size, and most importantly, 3) the address translation is transparent to the user software. Although this approach is highly effective in reducing the execution time, two problems remain to be solved: 1) each width of the 2-D processing area requires a different address bit-order interchange and 2) the approach cannot support simultaneous parallel small tile access.

Although other studies [12] [13] [14] [15] [16] have used the Morton layout to exploit 2-D data locality and reduce conflict misses and TLB misses, these approaches have increased the cycle time or access latency due to the versatility of Morton-index translations. In contrast, [17] proposed a hardware-based bit-permuting unit to translate the raster scan order address to a Z-Morton address that indicated that the hardware-based method reduced the overhead of Z-Morton address calculation (total number of instructions) by about 40% compared with that of the software-based method on a superscalar processor [18]. However, the main focus of this study was not to obtain column-directional parallel access capability but to facilitate the reduction of the software overhead for Z-Morton address calculation.

The author proposes a new cache memory architecture for one-dimensional and two-dimensional (1-D/2-D) data processing to eliminate these problems and combine it with a SIMD-based general purpose-oriented datapath [19] [20]. Our method provides more advanced features of a cache line-sized tile or line accessibility than the previous hardware and software cache tiling techniques and a user-transparent hybrid Z-order tile-based address implemented without additional latency.

1.2 Approach and main contributions of this thesis

The 13 most outstanding specific contributions of this thesis are highlighted below:

1. The author proposes a new 8-way set associative cache (32 Kbytes with a 64-byte cache line) with an 8×8 byte-sized tile and 64-byte-sized line accessibility. Its parallel aligned/unaligned tile and line access corresponding to parallel data access in the column and row directions can improve throughput with a low latency overhead.

2. The tile access corresponding to column-directional parallel data access can eliminate the transposition required in matrix calculation, orthogonal transform such as fourier transform or discrete cosine transform (DCT) and image feature detection, and it can provide efficient 2-D unit block access for image processing and video coding, even though its utilization may require significant modifications to the program code.

3. The author proposes a method of reducing tag memory that replaces multiple tiles with an aligned tile set (RATS) in the cache to reduce the hardware overhead of the proposed cache. The RATS method considerably reduces the entire hardware scale of the proposed cache and simplifies the cache architecture without affecting cache performance.

4. The proposed cache can minimize TLB misses in both row and column direction in 2-D processing. As a result, it can minimize the optimization efforts of an original tiling code in which it is difficult to utilize manually coded libraries such as OpenBLAS or Intel MKL. In other words, the proposed cache provides a column-directional parallel access function and allows either row-major based or column-major based 2-D program code so that it increases the degree of freedom of coding.

Therefore, the programmer's burden of coding to improve processing efficiency can be reduced by using our proposed cache.

5. The column-directional address stride becomes sufficiently smaller than the 4 Kbytes page size (virtual memory page size) in the 4-level Z-order tiling layout. This small stride minimizes the conflict misses and TLB misses in the column-directional adjacent or contiguous access. As a result, the proposed cache can provide efficient column-directional access as well as row directional access with a minimal hardware increase so that it enables efficient SIMD operations that require no transposition in matrix computation.

6. The proposed cache scheme can be freely configured from a direct-mapped cache to an n-way set associative cache. To the best of my knowledge, the author is the first to design a high-performance on-chip n-way set associative cache memory with tile/line accessibility for 2-D data processing and combine the proposed cache with a general purpose-oriented datapath that supports SIMD extensions. The author also evaluates three main specifications to demonstrate the feasibility of the proposed cache: read and write latency, clock period, and hardware scale.

7. The author adds a dual data access mode to the proposed cache memory to support the conventional raster line access for 1-D data processing. The proposed cache can appropriately switch a 2-D data access mode for the proposed tile and line access to a 1-D data access mode for the conventional raster line access.

8. The author proposes a new data layout called the 4-level Z-order tiling layout based on the conventional Z-Morton layout to maximize the exploitation of 2-D reference locality, reduce cache misses and TLB misses and reduce the amount of excessive data transfer. The proposed 4-level Z-order tiling layout divides data into hierarchical 3-level tiles (1st-level large tiles, 2nd-level medium tiles and 3rd-level tiles), each of which is arranged in raster scan order at the same level.

9. The author proposes a new data layout called the Cache-based hybrid Z-ordering layout based on the 4-level Z-order tiling layout and the conventional Z-Morton layout to efficient exploit hardware prefetching technique. The proposed layout

can exploit constant-stride prefetching in both major-directional and non-major directional access. In addition, combining the Strassen algorithm with the proposed layout can improve MM performance for any matrix size.

10. The address bit-order interchanger in the proposed cache is performed by hardware and is transparent to the processor. This address bit-order interchange that corresponds to a 64 Kbytes-wide area eliminates the address calculation overhead of Morton-index conversion for 2-D data access and allows the 4-level Z-order (based on the Z-Morton layout) curve tiling layout to be accessed as a conventional raster layout.

11. The author evaluates the performance of the proposed cache for MM and LUD in terms of the number of L1, L2 cache and TLB misses and the execution time (cycles). The results obtained from evaluation revealed that the RATS cache and the Non-RATS cache provide almost the same performance in matrix computation, which is not inferior to that of the conventional cache.

12. The author evaluates the hardware scale overhead of the RATS cache. Compared with the conventional cache, the entire hardware scale overhead of the RATS cache is reduced to only 5% and 7% for an 8-way set associative cache with a 32-byte cache line for the former and a 64-byte cache line for the latter. The RATS cache provides almost the same performance as that of the Non-RATS cache although it require only a minimum addition of hardware for both tile and line accessibility.

13. The author modifies the SimpleScalar simulator and evaluates the performance of load instruction reduction (SIMD processing) for parallel tile and line access. The results from evaluation indicated that the number of load instructions for SIMD processing is reduced to about one fourth of that required for non-SIMD processing. The execution cycles for SIMD processing is reduced to about one third of those required for non-SIMD processing due to the effect of L1 and L2 caches misses. The proposed cache with tile/line accessibility further improves the performance of 2-D applications by using SIMD instructions.

1.3 Thesis Organization

The remainder of this paper is organized as follows. Background information related to the thesis is described in Section 2. This section introduces the conventional raster layout and cache tiling method. Current SIMD instruction set extensions are described in Section 3. This section introduces the current SIMD architecture, the Intel MK-L/IPP library and non-contiguous memory access operation by using gather/scatter instructions. Skewed storage scheme is described in Section 4. This section introduces parallel data access in the row and column direction can be realized by using multi-bank memory structure that supports skewed storage scheme. Section 5 compares various hierarchical tiling data layouts such as the single-level tiling layout, multi-level Z-order tiling layout, Z-Morton layout and Morton hybrid layout. The proposed 4-level Z-order tiling data layout, the proposed cache architecture design, address translation unit and parallel tile/line access scheme are presented in Section 6. The RATS tag memory reduction method is presented in Section 7. The design result of the proposed cache and the performance evaluation results are presented in Section 8. Finally, the author provides a discussion and conclusion in Section 9.

2 Current insufficient utilization of 2-D spatial reference locality

2.1 Conventional raster layout

Matrices in most language implementations are usually stored in contiguous memory locations using either row-major or column-major raster layout. Figures 2.1 and 2.2 outline the configurations for row-major and column-major layouts. As shown in Figure 2.1, the row-major layout stores the first row of a 1-D matrix in contiguous memory, and then the second, etc. As shown in Figure 2.2, the column-major layout stores the first column of a 1-D matrix in contiguous memory, and then the second, etc. Therefore, consecutive elements of the rows of the matrix are stored in contiguous memory in the row-major layout and consecutive elements of the columns of the matrix are stored in contiguous memory in column-major layout. The raster layout is also called 1-D contiguous order layout. The row-major layout is by far the most commonly used today as it has been adopted in C language.

2.1.1 Layout features

There are three main advantages of the raster layout. (1) It is easy to generate the address of each element since the address linearly increases in the major, i.e, either in the row or column direction. (2) If the accessed data are stored in the same page, they do not occur TLB misses. If the accessed data are stored in the same cache line, they can be loaded onto the processor in one cycle. Therefore, all the contiguous elements linearly arranged in the raster layout can be fetched very quickly in the major direction. Parallel access by using the cache line in the major direction can be achieved. However, the raster layout provides poor access capability for 2-D arrays unless the data are accessed in the major direction (for a row-major layout, the row direction is the major direction and the column direction is a non-major direction). (3) This is efficient for 1-D data processing because the range of the same tag values maximizes in the major direction.

However, efficient access is only provided in the major direction because the elements for parallel access must be stored in almost the same cache line or the same

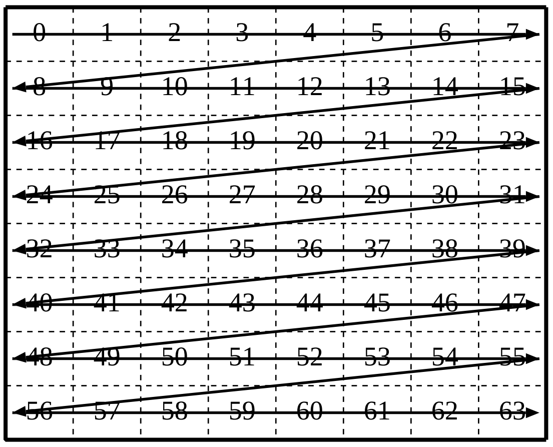


Figure 2.1: Raster layout: row-major order with 8×8 sized matrix.

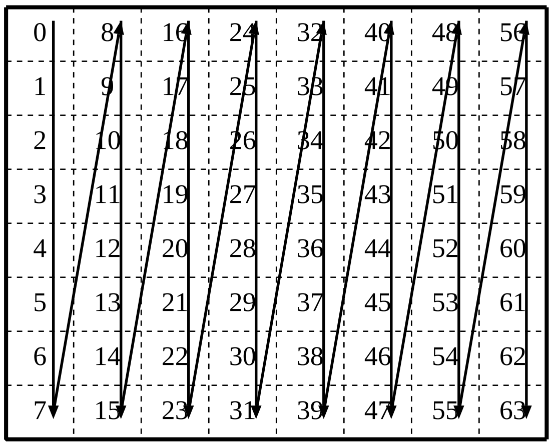


Figure 2.2: Raster layout: column-major order with 8×8 sized matrix.

page. As a result, the raster layout has poor accessibility in the non-major direction for large-sized 2-D data. Non-major-directional contiguous access causes frequent TLB misses if the stride length is greater than the page size of virtual memory so that the non-major directional contiguous access exceeds the TLB size. Consequently, the raster layout is inefficient for 2-D data access.

2.1.2 Difficulty of the 2-D locality utilization

This raster layout does not lead to good access efficiency for 2-D data without a skillful or tricky processing algorithm. This is because the raster layout imposes stride addressing on non-major directional contiguous elements so that it makes parallel data access difficult in the non-major direction and it can only provide efficient access capability in the major direction. In addition, the raster layout causes problem with frequent TLB misses in large-sized 2-D data access due to the small TLB sizes such as those with 64 entries.

2.1.2.1 Problems with TLB misses for large-sized 2-D data access

The processor generates logical addresses that just become virtual addresses. In addition, physical addresses are used to access memory. TLB is a cache of memory that stores recent translations of virtual to physical addresses for faster retrieval. Therefore, TLB only stores the most recently accessed page table entries. A TLB miss occurs when it does not contain the tag entry for translating a virtual address to a physical address. If there is a TLB miss, the translation proceeds by looking up the page table in a process. A page fault occurs if the processor accesses data that cannot be found either in the TLB or in the main memory.

The raster layout causes frequent TLB misses for large-sized 2-D data access due to the small TLB size. The major-directional small address stride for 2-D data access is mostly not over the page size whereas the non-major-directional large address stride (stride length is greater than page size) always exceeds the page size and it often causes TLB misses. Therefore, non-major directional access causes frequent TLB misses. Consider large scale matrices storing elements in a row-major layout for MM. Column-directional access causes frequent TLB misses because the address of elements in each column are mapped to a different page.

Listing 1: The basic MM version (ijk).

```
int main()
{
   for(i = 0; i < 32; i++)
    for(j = 0; j < 32; j++)
     for(k = 0; k < 32; k++)
         C[i][j]+= A[i][k] * B[k][j];
}
```

Suppose that there are three matrices, A, B, and C, each of which is a 32×32 matrix composed of double precision elements and laid out in memory in row-major order and that their beginning is aligned to a page boundary (e.g., A[0][0] is located at the beginning of a page and B[0][0] is located on another page.). Consider the basic MM algorithm in ijk order as shown in Listing 1.

Assume the page size is 8 Kbytes and 256 Kbytes of physical addresses are covered by a 32 entry TLB. Each matrix needs $2^5 \times 2^5 \times 2^3 = 8$ Kbytes. The column-directional elements of matrix B accessed from the different rows are in the same page in the k loop scans, so that a TLB miss is caused in the element access. Consider the matrix size in Listing 1 is N×N where N is larger than the page size. The memory pages are accessed consecutively in the row-directional matrix A access. As a result, TLB misses caused by row-directional access are equal to N^2/P (P is the page size). In contrast, the number of the TLB misses in the column-directional matrix B access is almost equal to N since accessing a different row causes a TLB miss since each row is assigned to N different pages. Consequently, the more matrix size sharply increases, and the TLB misses increase since the row-major layout cannot exploit the 2-D reference locality in the column direction.

2.1.2.2 Excessive data transfer problem

The raster layout also cannot provide efficient access to the 2-D localized array data because it only provides the data transmission function by cache line in the major direction. For example, in block matching for motion search or image recognition, the typical cache line size from 32 to 256 bytes is too long to load a block equal to or less than the conventional 16×16 pixel block so that more than half of the accessed cache line data is unused. As shown in Figure 2.3, a 32 byte cache line contains a number of data unnecessary for the 16×16 pixel macroblock access. If the required data are across the two adjacent cache lines, 64×16 pixel data transmission is required nevertheless the 48×16 pixel of the transmitted data may be unused. This inefficient raster data access frequently occur in the feature detection of the motion recognition.

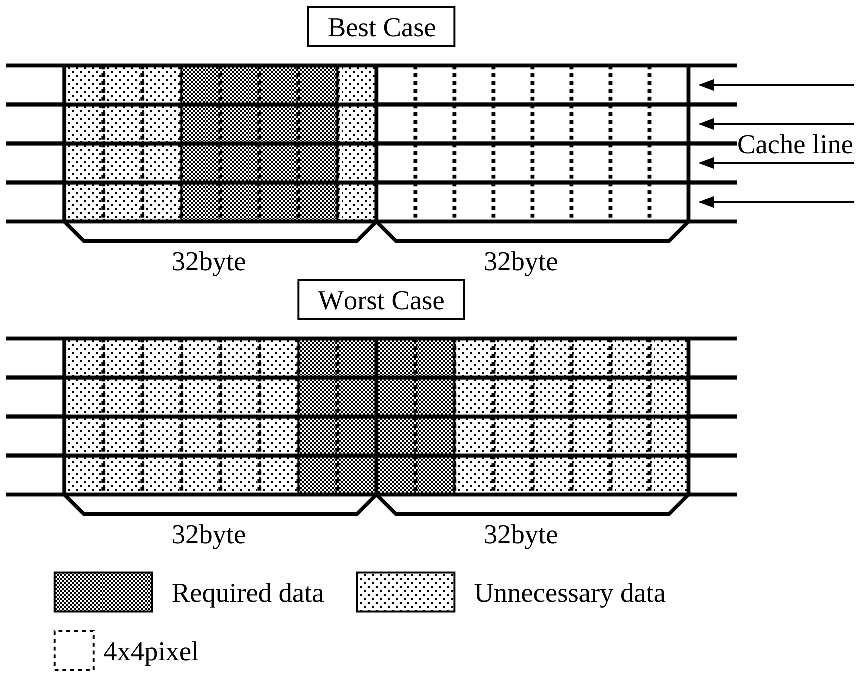


Figure 2.3: Excessive data transfer in raster data access.

2.1.2.3 Parallel data access capability in the major direction

The elements of each column in row-major layout are non-contiguous in memory even though the elements of each row are contiguous in memory. Therefore, the row-major layout only provides row-directional parallel access capability.

Figure 2.4 show that an 8×8 matrix is stored in row-major layout. The processor contiguously accesses the elements of the matrix in the row direction. Once the start address of a row is obtained, the contiguously accessed elements in the row direction can be fetched very quickly because they are almost in the same cache line or in the same page. However, the processor cannot parallel access elements of the matrix in the column direction since the address stride between adjacent elements in the column direction is determined by the 2-D data width. Therefore, the accessed elements may be stored in the same memory bank and cannot be read out in parallel even if the cache memory is composed of a multi-bank structure. Consequently, only part of the data can be read out in parallel.

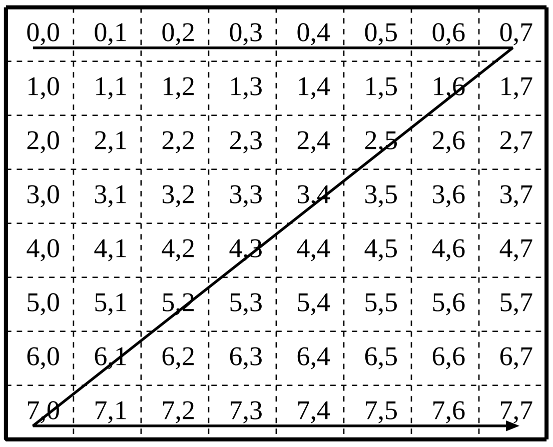


Figure 2.4: Mapping a 2-D matrix to row-major layout.

2.2 Conventional methods of improvement

2.2.1 Usage of transposition

Since elements in the non-major direction in raster layout are stored in non-contiguous locations in memory, non-major-directional contiguous access may cause significant performance degradation. Transposition of the turning row (column) into the column (row) has been used to solve this ineffective data access problem. For example, let us consider that the matrix in Figure 2.4 stores elements in row-major layout. Transposition can be used to enable the column data in Figure 2.5 to be contiguously accessed.

Transposition is required for 2-D data applications, such as matrix calculation, fast fourier transform (FFT), DCT or inverse discrete cosine transform (IDCT). Here, DCT is often used in image compression and video encoding. DCT in video encoding requires the second largest amount of operation and occupies about 10 to 20% of the total amount of encoding operations. Generally, N×N 2-D DCT consists of horizontal 1-D DCT and vertical 1-D DCT computing. The horizontal 1-D DCT processes data in the row direction while the vertical 1-D DCT processes data in the column direction. Horizontal 1-D DCT is simply performed with high processing efficiency since it is composed of 1-D operations in the row major layout. However, vertical 1-D DCT suffers from the high processing load of prior transposition that is required to match its column-directional operation to the row-major layout for efficient 1-D operation.

0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2	7,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3	7,3
0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4
0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5
0,6	1,6	2,6	3,6	4,6	5,6	6,6	7,6
0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7

Figure 2.5: Transpose a row-major order based 8×8 matrix to a column-major order based 8×8 matrix.

The author has analyzed VideoLANx264, which is the most popular open source encoder software for H.264 [21], to speed-up the 2-D DCT. This VideoLANx264 obtains a practical processing speed by utilizing SIMD instructions such as multimedia extensions (MMX), streaming SIMD extensions (SSE) and advanced vector extensions (AVX). The author estimates the throughput of the x264 DCT function with different block sizes by referring to x86 and x64 instruction throughput [22]. The estimated results are summarized in Table 2.1. The number of executed transpose operation cycles is as many as about 25-30% of the DCT processing cycles.

As a result, 2-D DCT processing can be executed without inefficient data access in the column direction by utilizing transpose operation. However, the transpose operation is so complicated for conventional SIMD instructions that it slows down the execution time together with frequent TLB miss occurrences due to the long address stride in the column direction. As can be seen from Figure 2.6, the basic transpose operation used in VideoLANx264 is performed by using SIMD instructions. There are 13 instructions for each 4×4 transpose matrix. Eight of the instructions are unpack instructions, and the rest are swap instructions. As a result, it is necessary to eliminate the overhead caused by the transpose operation to improve the performance of 2-D applications.

Table 2.1: Throughout count breakdown of DCT in videolan x264.

Block size	Memory Access	1-D DCT	Transpose	1-D DCT	Others	Total	SIMD Extension
DCT4×4DC(dct4x4dc function)							
4×4	6	9	13	9	4.5	41.5	AVX
DCT4×4residual(sub<block_size>_dctfunction)							
4×4	10.5	9	13	9	8	49.5	AVX
8×8	8	9	18	9	28.75	72.75	AVX2
16×16	48	36	72	36	53.5	245.5	AVX2

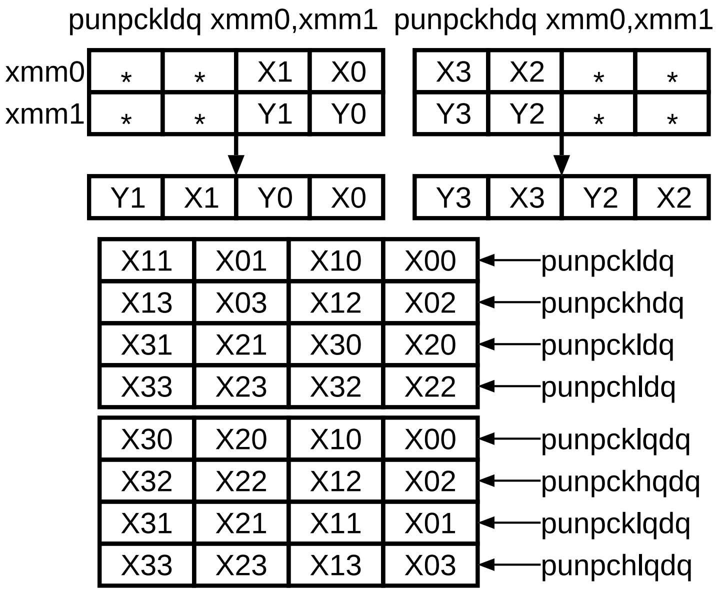
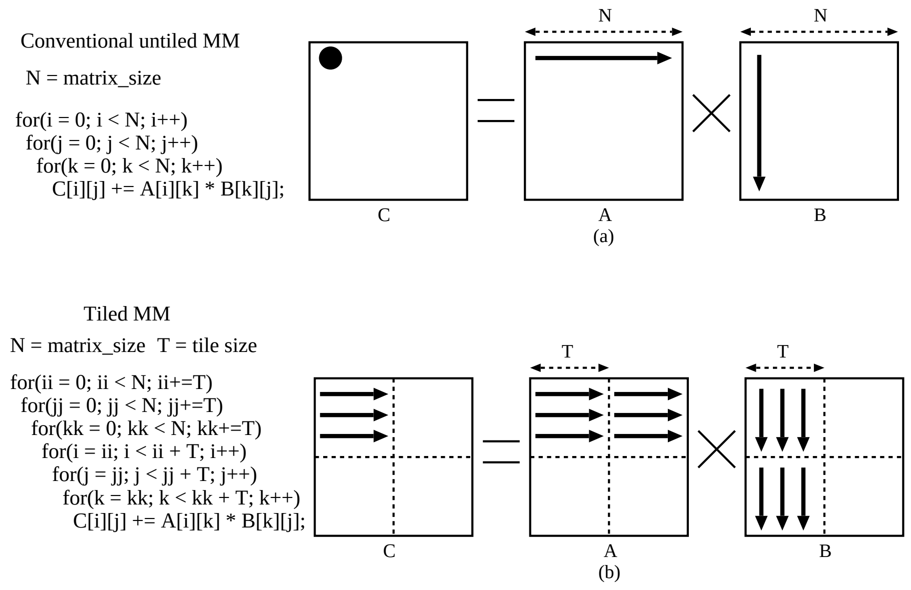


Figure 2.6: Excessive data transfer in raster data access.

2.2.2 Cache tiling

Single-level tiling technique (known as blocking) in the raster layout and hierarchical tiling data layout have been used to effectively improve 2-D data access capability on both hardware and software levels. Tiling is a widely used loop iteration reordering technique to improve 2-D reference locality by enhancing the reuse of data in the cache without changing the processing order of the raster layout [23]. On the other hand, hierarchical tiling data layout divides the raster layout into hierarchical n-level tile, each of which is arranged in raster scan order. It improves 2-D reference locality and TLB performance.



```
N = matrix_size

for(i = 0; i < N; i++)
  for(j = 0; j < N; j++)
    for(k = 0; k < N; k++)
      C[i][j] += A[i][k] * B[k][j];
```

```
N = matrix_size  T = tile size

for(ii = 0; ii < N; ii+=T)
  for(jj = 0; jj < N; jj+=T)
    for(kk = 0; kk < N; kk+=T)
      for(i = ii; i < ii + T; i++)
        for(j = jj; j < jj + T; j++)
          for(k = kk; k < kk + T; k++)
            C[i][j] += A[i][k] * B[k][j];
```

Figure 2.7: Tiled and untiled MM.

Figure 2.7 shows an example of a conventional untiled MM code and a tiled MM code (tiling method). For conventional untiled MM code, the capacity misses will occur frequently if matrix size N is larger than cache size. Tiling technique improves cache performance by dividing the N×N matrix into sufficiently small size of T×T. T×T sized sub matrix are small and can be fit in the cache. Therefore, tiling reduces the capacity misses efficiently. There are two kinds of tiling method: software-based tiling and hardware-based tiling.

2.2.2.1 Software-based cache tiling method

Software-based tiling is a compiler optimization technique attempting to divide a loop's iteration space into multiple tiles, so as to ensure that data used in a loop stays in the cache until it is reused. The division of loop iteration space leads to division of a large-sized matrix into smaller tiles, thus fitting accessed matrix elements into cache size, reducing cache misses. As shown in Figure 2.8, 2-D data in memory are divided into equal sized multiple tiles. All tiles are arranged in row-major order in memory. The tile size is equal to cache size to minimize the capacity misses.

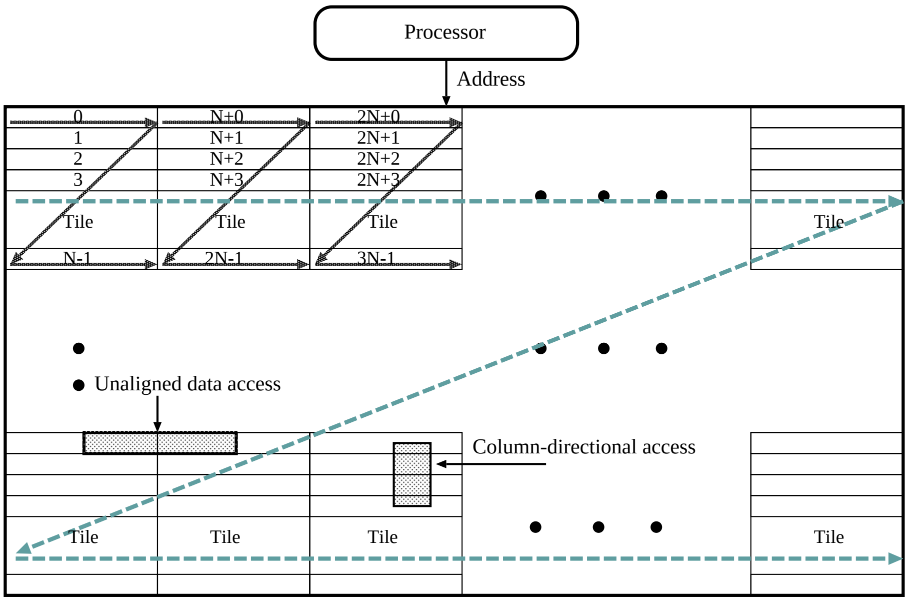


Figure 2.8: Software-based tiling.

However, software-based tiling containing no address translation steps in the user software, depend on the compiler to insert additional translation steps so that it suffers from additional overhead executing the translation steps. There are several software-based techniques that can reduce the tiling address calculation overhead. However, the address calculation overhead still cannot be sufficiently decreased because the address translation requires judgment of each tile boundary. In addition, unaligned row-directional access is difficult realized due to non-contiguous addresses between the row-directionally adjacent tiles are non-contiguous.

2.2.2.2 Hardware-based cache tiling method

To reduce the address calculation overhead of the software-based tiling, a hardware-based cache tiling structure for morphological image processing was proposed by Wittenbrink and Somani. They showed that the hardware-based tiling approach has three important advantages over the software approach: 1) The image storing and transfer mechanisms, such as direct memory access (DMA) or I/O devices, are independent of the tiling scheme. 2) A small tile can be used to process images of arbitrary size, and, most importantly, 3) The address translation is transparent to the user software. These advantages are based on the address translation taking place in hardware, and not in software. As shown in Figure 2.9, by address bit-order interchange, the hardware-based address translation unit translates a raster address (logic address) from an external processor into a tiling address (virtual address) to access data in memory. At a result, the address bit-order interchange eliminates the tiling address calculation overhead and allow the tiling data of virtual address space to be accessed as conventional raster data of logic address space.

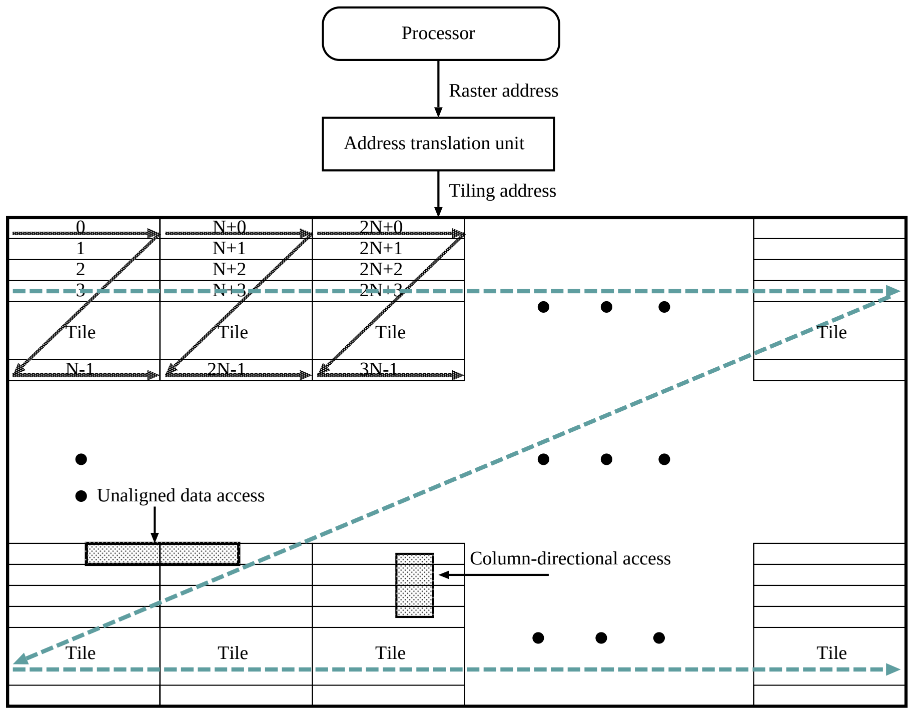


Figure 2.9: Hardware-based tiling.

Compared with the software-based tiling, unaligned row-directional access can be easily realized because the address translation is transparent to the user software. However, this conventional address translation unit has a drawback, that is each width of the 2-D processing area requires a different address bit-order interchange. For applications such as MM, if the width of the 2-D processing area is not fixed, a different address bit-order interchange is required for the width when the matrix size changes. For example, if the matrix size is 128, the width of the 2-D processing area is set at 128. This complex address translation can avoid mapping the waste space of the 2-D data processing area. However, it increases hardware scale of the address translation and may cause a clock-cycle constraint problem.

2.2.2.3 Problem of the cache tiling method

Software-based tiling and hardware-based tiling method improve the 2-D reference locality and reduce the cache capacity misses efficiently. However, there are several problems remain to be solved:

- It cannot support parallel aligned/unaligned data access in both row and column direction.
- Conventional tiling only provides a major-directional access capability. 2-D data processing by SIMD operations is restricted by no availability of non-major-directional parallel access function so that it cannot avoid performance degradation caused by the ineffective non-major-directional access. In addition, the transposition processing required in such as DCT or FFT cannot be eliminated. Transpose operation causes a significant processing overhead.
- Conventional tiling improves the 2-D reference locality and reduce capacity misses efficiently. However, it is inefficient for reducing TLB misses because the non-major directional access causes frequent TLB misses for large-sized 2-D data access.
- For conventional hardware-based tiling, each width of the 2-D processing area requires a different address bit-order interchange. This complex address translation not only increases the hardware scale of address translation and but also may cause a clock-cycle constraint problem.

- Conventional tiling cannot support simultaneous parallel small tile access. As a result, it cannot provide efficient 2-D unit block access so that excessive data transfer problem remains to be solved for image processing and video coding.

2.2.3 Hardware prefetching

In addition to the previously discussed method of cache tiling, many modern processors now uses hardware prefetching techniques to improve the performance of execution of applications [24]. Hardware prefetching is a technique used by processors that predicts soon-to-be used instructions or data and loads them from their original low-speed main memory into the high-speed cache memory before they are accessed. Prefetching has been used as an effective technique to improve the cache behavior of conventional raster layout by reducing memory access latency.

Hardware prefetching keeps track of memory access patterns, and fetches cache lines that may be accessed based on the memory access pattern in the future. There are several kinds of hardware prefetching methods. A commonly used method is sequential unit-stride prefetching, where cache lines are accessed with stride of one. For example, if the processor contiguously accesses the cache lines of A, A+1 and A+2, the prefetcher can predict that A+3 will be accessed next, and starts fetching the cache line of A+3.

Listing 2: The basic MM version (ikj).

```
int main()
{
  for(i = 0; i < N; i++)
   for(k = 0; k < N; k++)
    for(j = 0; j < N; j++)
        C[i][j]+= A[i][k] * B[k][j];
}
```

Another commonly used method of hardware prefetching is called constant-stride prefetching, which is similar to sequential unit-stride prefetching, but with variable access stride (not always + 1). For example, if the processor contiguously accesses the cache lines of A, A+100 and A+200, the constant-stride prefetcher can detect the memory access pattern and start fetching the cache line of A+300. Consider the MM given in Listing 2. All arrays in Listing 2 can be prefetched by using sequential unit-stride prefetching, because all of them are accessed in the major direction of the row-major layout.

There are significant cases where sequential unit-stride prefetching cannot be used. Consider the MM given in Listing 3. Array A[i][k] can be prefetched in the innermost loop by using the sequential unit-stride prefetch because the sequential accesses in the row direction fits into the sequential unit-stride pattern. However, array B[k][j] cannot be prefetched by using the sequential unit-stride prefetch because the sequential access of array B[k][i] is oriented in the non-major column direction. The access pattern would be B, B+N, B+2N, and so on, which cannot be detected by the sequential unit-stride prefetcher.

Listing 3: The basic MM version (ijk).

```
int main()
{
  for(i = 0; i < N; i++)
   for(j = 0; j < N; j++)
    for(k = 0; k < N; k++)
        C[i][j]+= A[i][k] * B[k][j];
}
```

There have been some recent architectures that can detect constant-stride accesses, which would enable all the arrays in the MM prefetcher to operate well. However, the recent Intel processor series, having constant-stride hardware prefetchers, cannot perform prefetching if the stride crosses 4 KB page boundaries [25]. Because of this constraint, constant-stride prefetchers in Intel processors cannot operate any better than unit-stride prefetchers if problem instances are too large, where each row is more than 4KB (512 doubles) page size.

3 Current SIMD instruction set extensions

Although computer performance has been considerably improving year by year, the demand for efficient 2-D data processing, such as that for numerical calculations, image processing, object recognition, and video coding, continues to increase. One technique to solve this problem is the implementation of single instruction stream multiple data stream (SIMD) instruction set extensions. However, the non-major-directional memory access has not kept with the parallel operation of SIMD processor extensions so that it has become a bottleneck for efficient 2-D data processing that utilizes extended SIMD instructions. Transposition has been used to improve the performance of the non-major-directional memory access, however, it causes a significant overhead. Therefore, to improve the performance of 2-D data processing, it is necessary to enable efficient parallel memory access in the non-major direction as well as eliminate the overhead caused by the transposition processing.

3.1 SIMD parallel processing

SIMD is a type of parallel processing in which multiple sets of operands may be fetched to multiple operation units and may be operated upon simultaneously within a single instruction cycle. The SIMD instruction set extensions have become increasingly more popular, and have been included in most current computer processors. The SIMD extensions, such as Intel's AVX and SSE, can exploit the parallelism operating on multiple contiguous data at a time in image processing and MM algorithms.

The AVX and SSE instructions allow the hardware to divide a single wide ALU into multiple parallel smaller ALUs that operate simultaneously. For example, a single 256-bit ALU can be divided into four 64-bit ALUs, eight 32-bit ALUs, sixteen 16-bit ALUs or thirty two 8-bit ALUs. Therefore, the programmer can use the data transfer instructions to transfer either four 64-bit data or eight 32-bit data or sixteen 16-bit data or thirty two 8-bit data.

Figure 3.10 shows an example of a SIMD addition. A single 32-bit ALU is divided into four 8-bit ALUs. Each individual 8-bit register A is added to each individual 8-bit register B to form the result in register C. The conventional addition requires four

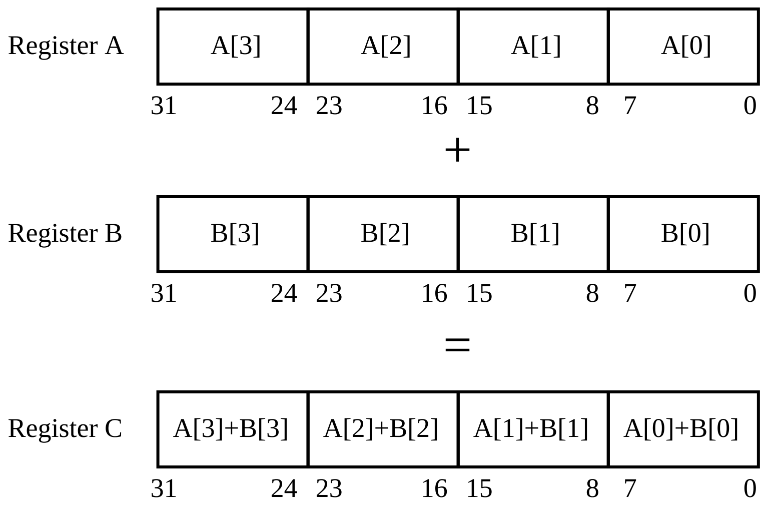


Figure 3.10: SIMD addition: C = A + B.

instructions and four cycles to complete this addition. Compared with the conventional addition, the SIMD addition only requires one instruction and can be completed in only one cycle. This is a speedup of four times. In addition, SIMD provides significantly performance improvement when dealing with matrix in a large loop.

The SIMD extensions are useful technologies that can effectively improve the processing speed of 2-D applications. These SIMD instructions are designed to exploit significant data-level parallelism of raster layout for scientific computing or image processing. Contiguous major-directional access of the raster layout makes it possible to use SIMD instructions that operate on vectors of data. However, the raster layout does not provide non-major-directional parallel accessibility. Ineffective non-contiguous memory access has becomes a bottleneck for efficient 2-D data processing. To solve this problem, current SIMD architecture has added gather/scatter instruction to the AVX to provide parallel non-major-directional accessibility. By gather/scatter operations, the processor can load/store multiple non-contiguous elements from/to memory. However, the TLB misses occur frequently if we use the gather/scatter instructions, this is because the raster layout cannot exploit the 2-D reference locality [26]. After all, transposition, which becomes a significant processing load, is commonly used to minimize the inefficient non-major-directional accesses.

3.2 Gather/scatter operations

3.2.1 Non-contiguous memory access by gather and scatter operations

Scientific computing such as matrix calculation, image processing or video encoding requires non-major-directional memory access. However, the raster layout does not efficiently correspond to the non-major-directional access leading to non-contiguous large stride data access. Therefore, some of the latest hardware architecture (such as Intel Haswell architecture) provide gather and scatter operations for the non-contiguous data access. The gather and scatter operations are kinds of data parallel operations, where a number of data are loaded (gathered) from or stored (scattered) to given locations.

The gather operation can be thought of as an operation of multiple element loads, where the elements (non-contiguous in memory) are merged into a single vector register. Another vector register is used to specify the addresses of the multiple elements. The scatter operation can be thought of as an operation of multiple element stores, where a single vector register holds the elements (non-contiguous in memory) to be stored, and another vector register specifies the addresses of the multiple elements for the stores. The latest Intel AVX/AVX2 supports a gather instruction for a 256-bit (eight 32-bit) YMM register.

Figure 3.11 shows the VGATHERDPS instruction that loads eight single-precision floats. Eight non-contiguous elements are loaded from memory and stored in the YMM1 register. The YMM0 register stores the addresses of the eight elements. Scatter operation, on the other hand, uses eight elements in the YMM1 register and stores them in memory by using the addresses in the YMM0 register. The processor can load/store multiple non-contiguous elements from/to memory by these gather/scatter operations. For example, the Intel Skylake hardware implements a hardware gather instruction called VGATHER. The processor can parallel load/store multiple non-contiguous elements from a multi-bank memory architecture by using the VGATHER instruction. However, this non-major-directional access in the raster layout becomes long stride non-contiguous access so that it frequently causes TLB misses. After all, transposition is the most efficient way to correspond to the non-major-directional access.

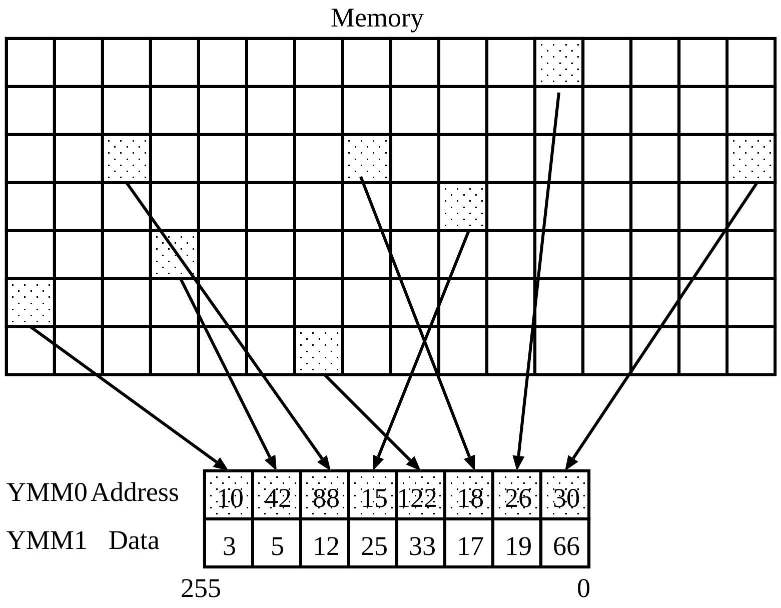


Figure 3.11: Gather operation: VGATHERDPS instruction.

3.3 Efficient transposition

3.3.1 Transposition routine with Intel AVX2

Although transposition minimizes the inefficient non-major-directional access. However, the overhead caused by transposition is significant. Intel AVX2 adopts a vector permute instruction (VPERM) to accelerate the speed of conventional transposition processing, which can take arbitrary bytes from a source register and places them in any position in a destination register controlled by an index register, as shown in Figure 3.12. YMM3, YMM2 and YMM1 correspond to the index register, source register and destination register. The YMM2 register stores the index value. For example, the zeroth index value in the YMM2 register is zero. Therefore, it stores the zeroth element in the YMM3 register to the zeroth storage in the YMM1 register. The fifth index value in the YMM2 register is six. Therefore, it stores the sixth element in the YMM3 register to the fifth storage in the YMM1 register.

In addition, it is possible to use only four instructions to transpose a 4×4 matrix by combining the VPERM instruction with the conventional VBLEND and VSHUF instructions. That significantly reduces the number of instructions required for conven-

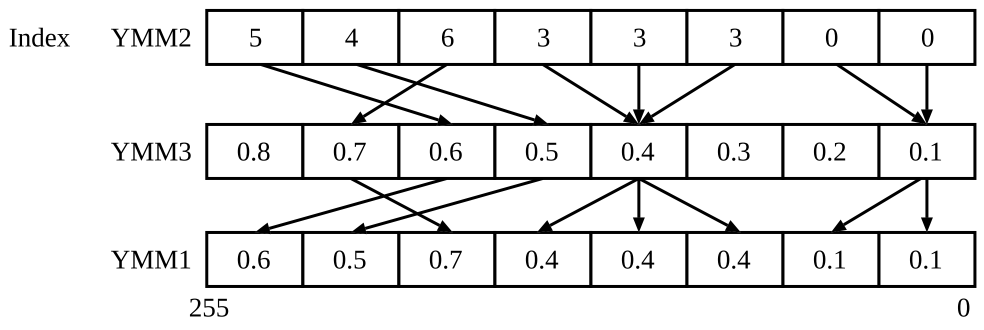


Figure 3.12: Elements permutation in YMM register: VPERM instruction.

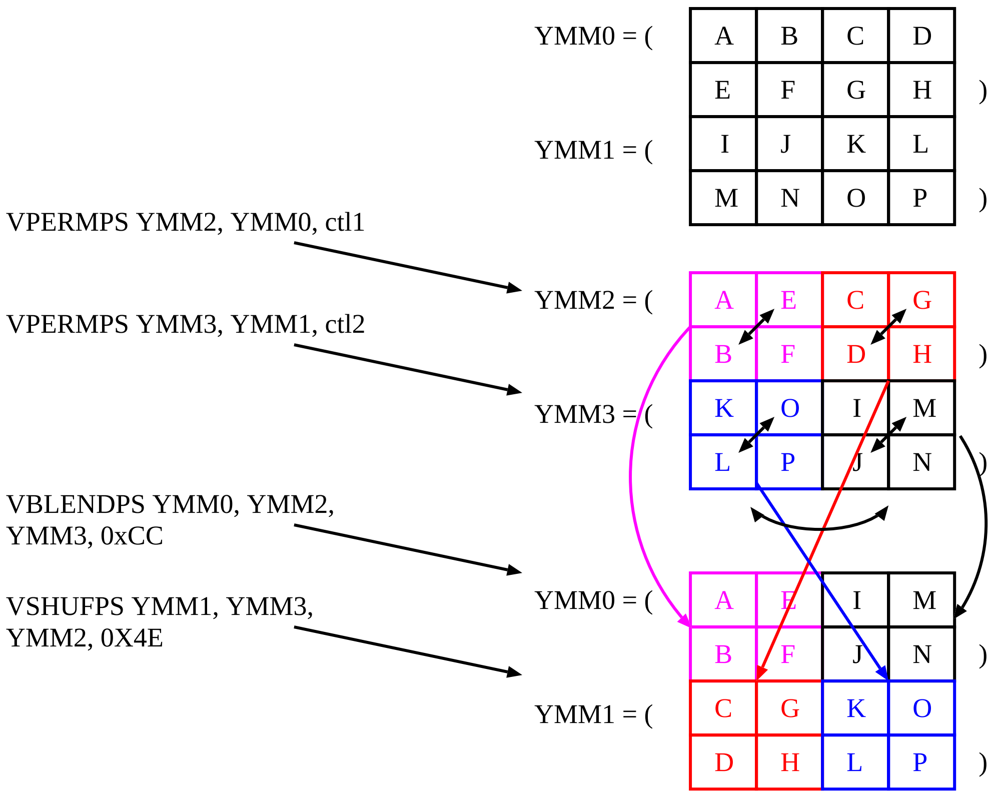


Figure 3.13: 4×4 transposition processing.

tional transposition as was explained in Section 2.2.1. Figure 3.13 shows transposition processing by using a combination of VPERM, VBLEND and VSHUF instructions. The 4×4 transposition can be completed by only using four instructions. Although

Intel AVX2 improves the speed of conventional transposition processing, the overhead caused by the transposition is still not eliminated. The TLB misses problem remains to be solved for large-sized 2-D data access.

3.4 Non-major-directional operation in the Intel library

It is important to exploit SIMD instructions and decrease the ineffective non-major-directional access to improve the performance of 2-D applications. Intel developed a technique of compiler optimization, called Intel MKL library to accelerate math processing routines to improve the performance of 2-D applications when running on system equipped with an Intel high-performance processor. The Intel MKL library contains numerous functions, such as basic linear algebra subprograms (BLAS), a linear algebra package (LAPACK), FFT and vector math. This library performs efficient matrix transposition and greatly improves the performance of the non-major-directional access. For instance, Intel MKL library includes out-of-place and in-place transposition routines. Moreover, sgemm and dgemm are used to perform MM. Sgemm and dgemm include a parameter to specify that the matrices are transposed before an operation is executed. These transposition routines improve the performance of the conventional transposition. In addition, BLAS is widely used to improve the performance of scientific computing, such as MM [27] [28] [29]. The BLAS provides standard building blocks for performing basic vector and matrix operations.

The Intel MKL library efficiently improves the performance of BLAS since it takes advantage of special features in the new generation of Intel's high-performance processors such as vector registers or SIMD instructions that greatly speed up MM. We do not need to modify the source code to take advantage of the new features of Intel processors with Intel MKL library. All we have to do is to link the code to Intel MKL library to maximize the use of instruction- and register-level SIMD parallelism and make use of the cache tiling technique. The BLAS significantly improves the performance of MM. However, it cannot eliminate the overhead caused by matrix transposition or provide parallel non-major-directional accessibility.

In addition, Intel integrated performance primitives (Intel IPP) offer developer's high-quality, low-level building blocks for image processing, signal processing, and other computations that involve large vector and matrices. Intel IPP libraries are designed to take advantage of Intel SSE, Intel AVX/AVX512 instructions. These instructions process 128, 256, and 512 bits of data in a single instruction, and accelerate matrix processing. The performance of 2-D applications can be improved by five and ten times by using the Intel IPP library. Similarly to Intel MKL library, Intel IPP significantly improves the performance of image processing or signal processing by maximizing use of SIMD instructions. However, it cannot provide parallel non-major-directional accessibility and the overhead caused by transposition cannot be eliminated.

4 Skewed storage scheme

4.1 Parallel row- and column-directional access by using skewed storage scheme

Parallel processing is particularly important in current computer systems. The SIMD architecture is designed to perform vector operations in parallel by employing a number of operation units to undertake work on each element of the vector. It can easily provide significantly higher memory access throughput by using memory with a wide I/O port. However, this simple wide I/O memory is not effective for acceleration of non-major directional access, which is the column-directional access required to eliminate the significant computing loads of transpositions in 2-D processing.

Figure 4.14 shows a 4×4 sized matrix that is stored in a 4-bank structured memory where each row or diagonal of matrix A can be accessed in parallel without conflict so that the conventional storage scheme only provides parallel access to a row or a diagonal. However, each column of matrix A cannot be accessed in parallel since all its elements are in the same bank so that parallel access cannot be performed due to memory conflict. As a result, simple I/O expansion requires n time's load/store operations for column access. Here, n equals the number of elements that compose the column. In contrast, the following skewed storage scheme allows parallel single time access to both the row and column.

4.1.1 Skewed storage scheme

The 4-bank structured memory with a skewed storage scheme in Figure 4.15 provides parallel row- and column-directional access to a 4×4 sized matrix. As shown in the Figure, the first, second, third and fourth row of the matrix are stored with additional skews corresponding to 0, 1, 2 and 3. Since the matrix is skewed, additional operations are needed to align the elements after the elements have been loaded to the processor register.

For example, if the processor accesses elements of a row, as shown in Figure 4.15, note that the elements in the first row are properly aligned. However, the elements in the second, third and fourth row need to be rotated left once for the second row, twice

A00	A01	A02	A03
A10	A11	A12	A13
A20	A21	A22	A23
A30	A31	A32	A33
Bank0	Bank1	Bank2	Bank3

Figure 4.14: 4-bank structured memory.

A00	A01	A02	A03
A13	A10	A11	A12
A22	A23	A20	A21
A31	A32	A33	A30
Bank0	Bank1	Bank2	Bank3

Figure 4.15: 4-bank structured memory that supports skewed storage scheme.

for the third and three times for the forth. The elements in the Nth row generally need to be rotated (N-1) times to the left to be aligned. However, we can no longer access diagonals without conflict, as shown in Figure 4.15. This is because there is no way to store an N×N matrix in N-bank memory when N is even, so that all the rows, columns and diagonals can be accessed without conflict. An N-bank structured memory with a skewed storage scheme causes an increase in the entire scale of hardware. This is because each memory bank needs an address generation circuit to calculate the address. In addition, a skewed storage scheme should be adopted in L1 cache memory because if it is adopted in a lower level cache or main memory, the L1 cache must be individually stored in both row and column data.

4.1.2 Parallel non-major-directional tile access by using skewed storage scheme

Parallel access in the row and column directions can be achieved by using a skewed storage scheme. However, the conventional cache cannot provide efficient access to 2-D localized array data because it only has conventional cache line accessibility. As a result, it suffers from the excessive data transfer problem.

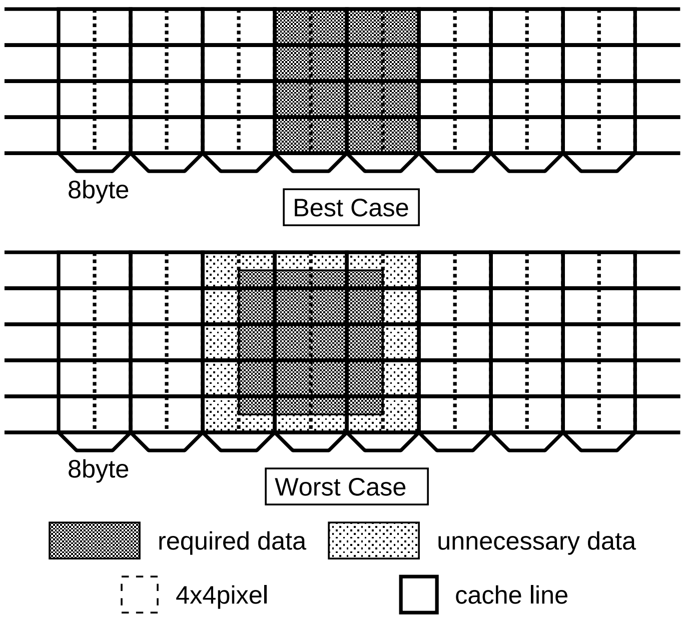


Figure 4.16: Block-offset mapping.

Block-offset mapping is a method of data allocation that is suitable for 2-D localized data access [4], where the conventional cache line is divided into rectangular tiles and each tile is allocated in raster scan order. Figure 4.17 shows an example of block-offset mapping that consists of 8×4 size raster tile shapes. Excessive data transmission does not occur in the best case when reading the 16×16 macroblock. Excessive data transmission is suppressed to 224 byte in the worst case. The number of accesses for 8×4-byte tiles and 32 byte raster lines are summarized in Table 4.2. Block-offset mapping makes it possible to reduce about 44% of accesses compared with conventional raster mapping. This block-offset mapping is also called a single-level tiling data layout or a block data layout.

Table 4.2: Access count reduction rate.

Access count							
Block size	16×16	16×8	8×16	8×8	8×4	4×8	4×4
Raster line access (32×1)	23.5	11.75	19.5	9.75	4.88	8.75	4.38
Block-offset mapping (8×4)	13.66	7.91	8.91	5.16	3.28	3.78	2.41
Access count reduction rate	41.8%	32.7%	54.3%	47.1%	32.7%	56.8%	44.9%

If the conventional cache line is divided into sublines in the main memory, parallel row- and column-directional access can be easily implemented by using a multi-bank memory array structure that supports a skewed array storage scheme. As shown in Figure 4.17, line data can be accessed by feeding a common address signal to all banks. On the other hand, tile data can be accessed by incremental addressing between neighboring banks. Conversion between skewed and non-skewed data arrays must naturally be performed by the upper or lower barrel shifter in each access.

Figure 4.17 shows that the 32 byte cache line is divided into four 8 byte-sized sublines that compose an 8×4 byte-sized tile. This means the data alignment is 8 byte and the CPU fetches 32 bytes starting at the requested address of an 8 byte stride. Data alignment can also be set to 4 byte if the 32 byte cache line is stored by using 4×8 byte-sized tiles in the main memory, as shown in Figure 4.18. The number of memory banks increases to eight to meet the requirements for accessing data in parallel in both row and column directions.

Using a single-level tiling data layout and a skewed storage scheme can provide parallel row- and column-directional accessibility and reduce the amount of excessive data transfer. However, the single-level tiling data layout only divides the data into single-level cache line-sized tiles. It does not effectively reduce the number of TLB

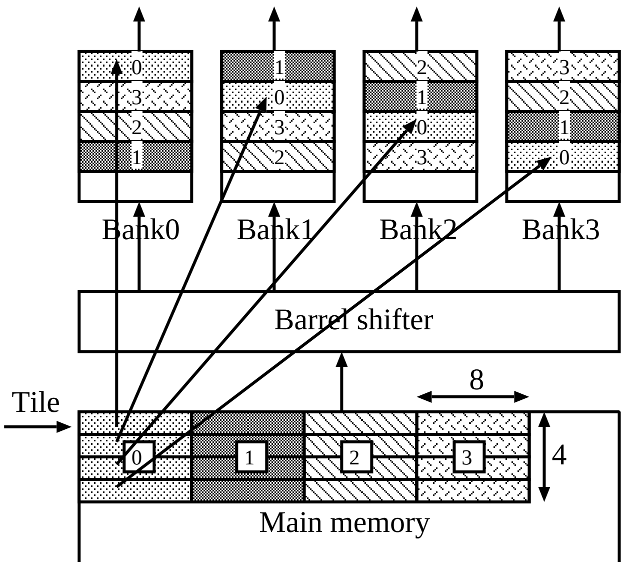


Figure 4.17: 8×4 byte-sized tile access by using skewed storage scheme.

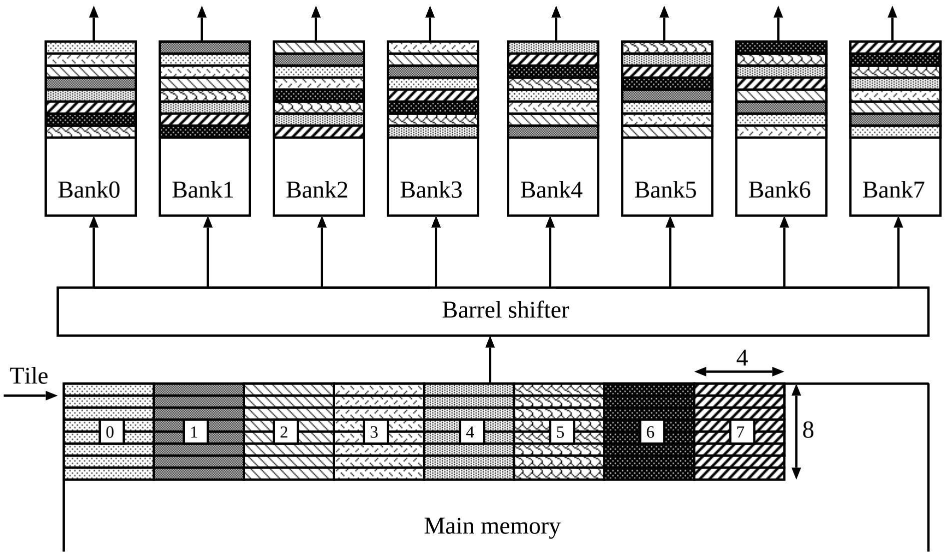


Figure 4.18: 4×8 byte-sized tile access by using skewed storage scheme.

misses since each tile in the column direction is mapped to a different page.

For example, Figure 4.19 shows that a 32 byte cache line is stored as an 8×4 byte-sized tile in the memory. As was explained in Section 2.1.2.1, the conventional computer

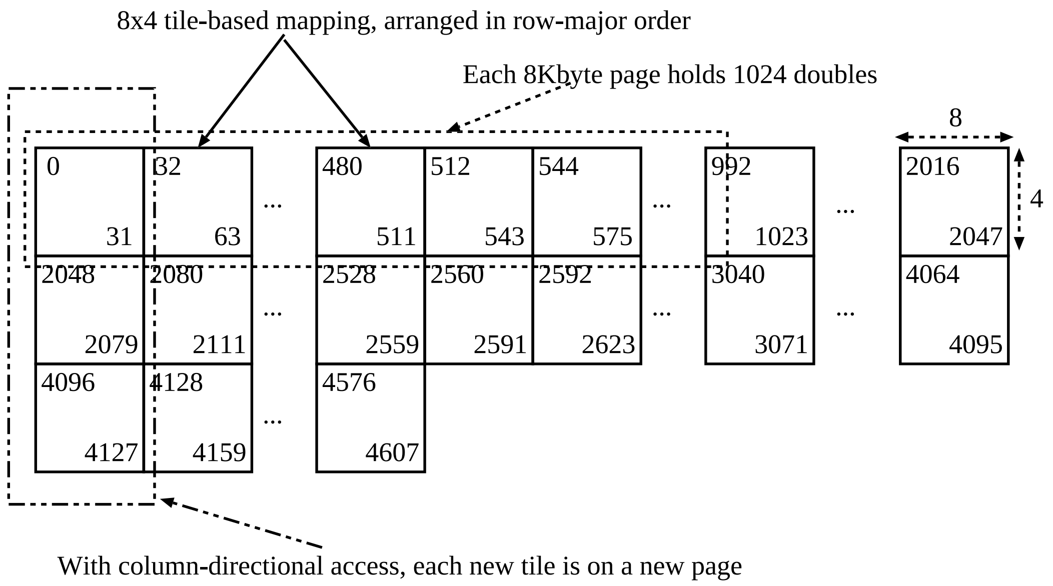

8x4 tile-based mapping, arranged in row-major order
Each 8Kbyte page holds 1024 doubles
8
4
0
31
32
63
480
511
512
543
544
575
992
1023
2016
2047
2048
2079
2080
2111
2528
2559
2560
2591
2592
2623
3040
3071
4064
4095
4096
4127
4128
4159
4576
4607
...
With column-directional access, each new tile is on a new page

Figure 4.19: Block-offset mapping for a large-sized 2-D data access.

system adopts a TLB to speedup address translation: e.g., a typical 64-entry data TLB with 8 Kbytes pages has an effective span of 64×8 KB = 512 KB. Unfortunately, if the elements in the column direction are contiguously accessed, TLB misses frequently occur. This figure shows a matrix having rows that are composed of 2048 doubles and stored in the single-level tiling data layout. Each 8 Kbytes page holds 1024 doubles; in other words 256 tiles each of which is composed of four 8 byte sized sublines. When the processor accesses data in the row direction, one page is accessed every 256/8 = 32 accesses (therefore, the hit rate is 1 - 1/32 = 96.9%); however, when the processor accesses data in the column direction, one page is accessed every 4 accesses (therefore, the hit rate is 1 - 1/4 = 75%). The author found that the single-level tiling data layout was still not effective to reduce TLB misses. It is possible to solve this problem by recursively applying the tile again. The author will describe this technique in the next section.

5 Hierarchical tiling data layout

5.1 Effectiveness of hierarchical tiling data layout

The raster layout is not suitable for multi-dimensional applications, since it is effective in only major-directional data access and not effective in utilizing 2-D reference locality. Conventional tiling method can improve capability utilizing 2-D reference locality. In addition, column-directional access can be performed by implementing skewed storage scheme or gather/scatter mechanism. By combining a skewed storage scheme with a single-level tiling data layout, as shown in Section 4.1.2, parallel data access in both row and column directions can be performed. However, the TLB misses problem remains to be solved for efficient column-directional parallel access. Hierarchical tiling data layout (multi-level tiling data layout) is based on tile's recursive partitioning into small size of sub-tiles, each of which is arranged in raster scan order. It can map a 2-D data array to a 1-D data array while preserving data reference locality in both row and column directions. It is more effective than conventional single-level tiling in terms of reducing TLB misses in the data access of power-of-two sized address strides. It improves the utilization capability of 2-D reference locality.

5.1.1 Block data layout

As shown in Section 2.1.1, the raster layout has some drawbacks. Consider a large matrix stored in row-major layout, column-directional data accesses cause frequent TLB misses due to the large address stride. Actually, if the row size of a matrix is greater than the number of TLB entries, column-directional data access causes frequent TLB misses and leads to significant performance degradation.

Block data layout is more efficient than a raster layout due to good utilization of 2-D reference locality and TLB performance. Data in the block data layout are partitioned into equally sized tiles, each of which is ordered in raster scan order. The block data layout is also called a single-level tiling data layout, as shown in Section 4.1.2. Figure 5.20 shows that block data layout divides an 8×8 sized matrix into 4×4 sized of sub-matrix (4×4 sized tile). The tiles are arranged in the row-major order so that all elements of each tile are stored in contiguous locations in memory. For

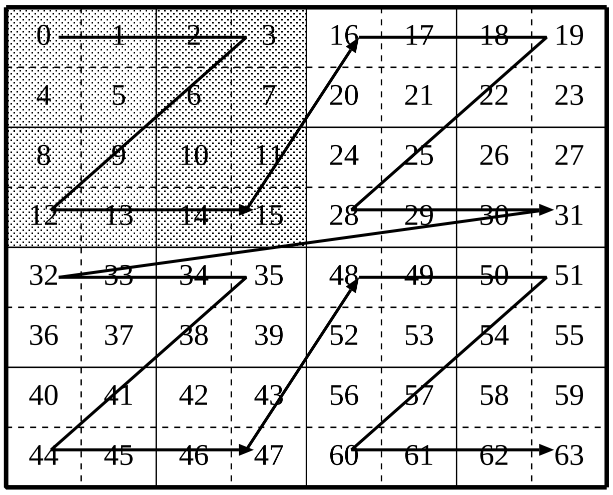


Figure 5.20: Block data layout, using 4×4 tile.

example, consider cache line is 128 byte (16 quad words for x86 architecture), each tile holds 16 quad words sub-matrix. In the row-directional access, the four iterations 0, 1, 2, 3 access locations on the same cache line. The remaining 12 locations on this tile are not accessed. Therefore, for large-sized 2-D data access, the expected cache hit rate is 75% because each tile has to be loaded four times to obtain 16 quad words.

Compared with raster layout, block data layout significantly decrease the number of TLB misses in the column-directional access. In addition, all elements of each tile has the same tag if the tile size is equal to page size so that any data access in the same tile does not cause conflict misses. However, the block data layout only matching tile size to page size cannot provide parallel tile access capability. We can overcome this problem by dividing the block data layout into multi-level tiling data layout, such as Z-Morton layout or Morton hybrid layout.

5.1.2 Z-Morton layout

Compared with the conventional raster layout and the block data layout, the multi-level tiling data layout such as the Z-Morton layout, have been proposed as a compromise between the row-major and column-major layouts for 2-D arrays. The Z-Morton layout is a mapping of multi-dimensional data to one dimensional space that preserves their locality. As shown in Figure 5.21, the Z-Morton layout divides an original 8×8 data array into four quadrants, which are ordered in raster scan order. Each quadrant is further divided into child quadrants that are laid out in a similar manner. At the end of recursion, elements of the quadrant are mapped onto contiguous memory location. This is similar to the arrangement of elements of a tile in block data layout. However, Z-Morton layout maps all quadrants onto contiguous memory location. Therefore, Z-Morton layout can be considered as a multi-level block data layout.

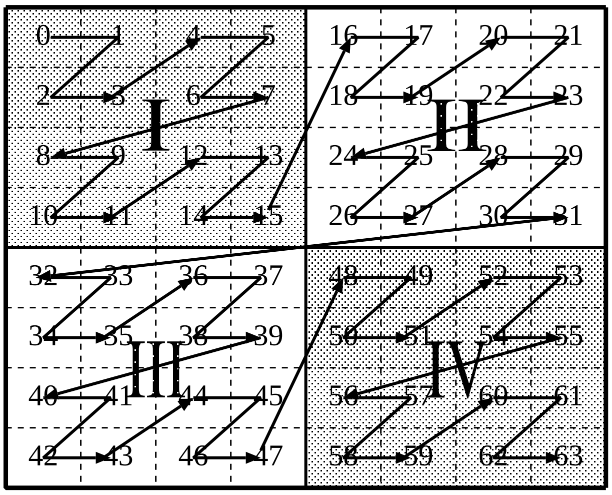


Figure 5.21: Z-Morton layout.

The benefits of this data layout are as follows:

- If the quadrant size at a certain level is matched with the page or cache line size to meet alignment restrictions, cache line conflicts, TLB misses, and page faults between the main memory and the hard disk can be reduced. In addition, compatibility with burst transfer can be improved by the Z-Morton layout, because each divided data or element data of the lowest level divided data can be contiguously accessed with a high probability.
- If the lowest level divided unit data can be assigned to a cache line, i.e., both row and column-directional several contiguous data are accessed as the smallest unit data in parallel, which we call a line and a tile, unnecessary data transfer to and from the external processor or the lower level cache is reduced.

- The Z-Morton layout provides substantial 2-D spatial reference locality when traversed in the row and column directions. Given a cache with any power-of-two cache line size, the cache hit rate of row-directional access is the same as the cache hit rate of column-directional access. For row- and column-directional data access, the theoretical cache hit rate for a cache with cache line size 2^{2k} is 1 - ($1/2^k$).

- Within each quadrant, sub-quadrants are recursively placed in the same Z-order. If the quadrant size at a certain level is matched with the page size, unused regions are not allocated in virtual memory.

Table 5.3: Theoretical cache and TLB hit rate for row-directional access of a large-sized 2-D array of double-precision floating-point operations. We do not consider the additional conflict misses due to 2-D data locality.

	Row-major layout	Z-Morton layout	Column-major layout
32 byte cache line	75%	50%	0%
64 byte cache line	87.5%	64.6%	0%
128 byte cache line	93.8%	75%	0%
4 KB page	99.8%	95.6%	0%
8 KB page	99.9%	96.9%	0%

Table 5.4: Theoretical cache and TLB hit rate for column-directional access of a large-sized 2-D array of double-precision floating-point operations. We do not consider the additional conflict misses due to 2-D data locality.

	Row-major layout	Z-Morton layout	Column-major layout
32 byte cache line	0%	50%	75%
64 byte cache line	0%	64.6%	87.5%
128 byte cache line	0%	75%	93.8%
4 KB page	0%	95.6%	99.8%
8 KB page	0%	96.9%	99.9%

Table 5.3 and 5.4 show the theoretical hit rate for row-directional access and column-directional access, respectively. As shown in Table 5.3, for 32 byte cache line (4 quad words, k = 1) this gives a hit rate of 50%. For 64 byte cache line (8 quad words, k = 1.5) the hit rate is 64.6%. For 128 byte cache line (16 quad words, k = 2) the hit rate is 75%. For 4 KB page (512 quad words, k = 4.5) the hit rate is 95.6%. For 8 KB page (1024 quad words, k = 5) the hit rate is 96.9%. Row-directional access in the row-major layout gives high hit rate, but the hit rate of column-major layout is 0%. Compared with the raster layout, the Z-Morton layout provides the same cache hit rate for both row- and column-directional access, as shown in Table 5.3 and 5.4. The hit rate of the Z-Morton layout is low because the Z-Morton layout does not provide cache line-sized tile access in the column direction.

In addition, the overall 2-D memory access performance of the Z-Morton layout is in general consistent for various data sizes compared with the raster layout. However, in order to handle the array indices of the Z-Morton layout without address conversion burden [18] [30], it is imperative that the raster layout is automatically transformed to the Z-Morton layout. In addition, since the Z-Morton layout divides data into multiple-levels, the complexity of the hierarchical Z-Morton imposes a certain amount of address calculation overhead for address translation on each access, which takes logarithmic time with respect to the address length [31] [32]. Some previous software-based techniques employ logical manipulations or look-up tables [13] to reduce the address calculation cost, but the overhead is still not eliminated. As such, the Morton address calculation imposes a software-overhead cost to exploit the 2-D spatial reference locality benefits of the Z-Morton layout.

5.1.3 Morton hybrid layout

Z-Morton layout can also be used only on higher-level quadrants, while the lowest-level quadrants still use conventional row- or column-major layout such as Morton hybrid layout [33] [34]. Figure 5.22 and 5.23 show a 16×16 sized Z-Morton layout and Morton hybrid layout, respectively. Figure 5.23 shows that Morton hybrid layout divides original 16×16 matrix into equally sized quadrants. Elements are stored within in column-major the lowest-level quadrants. Outside the lowest-level quadrants, high-level quadrants are stored in Z-Morton ordering. This provides 2-D data locality inside each quadrant, with fast address calculation among different quadrants.

Morton hybrid layout is designed to take advantage of hardware architecture and compiler optimization while still exploiting the benefits of hierarchical tiling data layout. To reduce cache misses and TLB misses, the lowest-level quadrant size is typically chosen to be equal to cache size or page size. To take advantage of the Intel library and compiler optimizations on current processor, such as BLAS technique, the lowest-level quadrant of a hierarchical tiling data layout must be row or column-major order. Therefore, it is possible to combine the BLAS optimization with the Morton hybrid layout to improve cache performance.

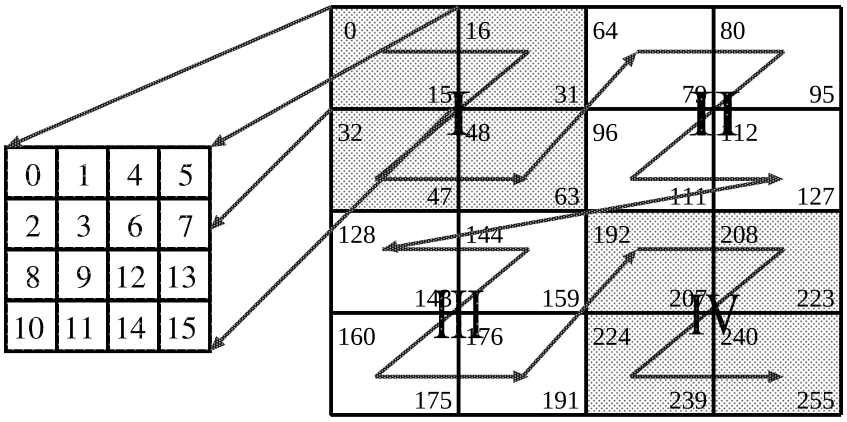


Figure 5.22: Z-Morton layout: 16×16 matrix.

A related work [33] proposed a method that apply Morton hybrid layout to dense BLAS techniques to improve the performance of dense matrix multiplication significantly. In addition, the evaluation results also showed that the lowest-quadrant size is largely not only dependent on the memory architecture but also the BLAS implementation. This is one disadvantage of Morton hybrid layout. While Z-Morton layout has the advantage that its performance is only dependent on memory architecture, Morton hybrid layout requires significant cache testing to achieve best performance on both memory architecture and the BLAS implementation.

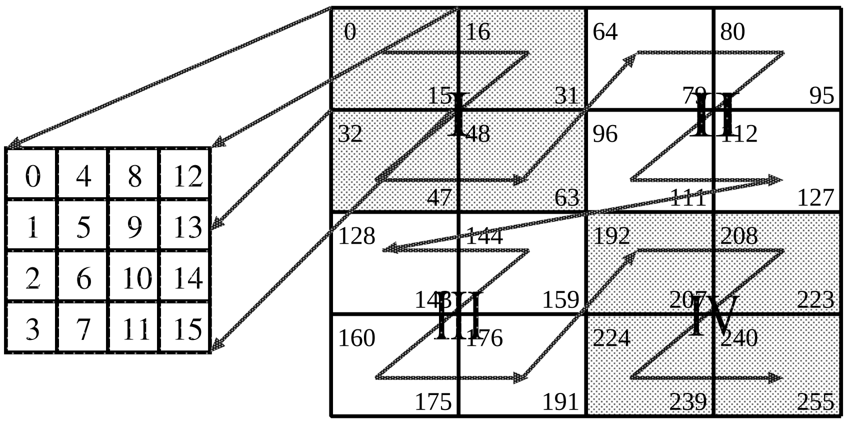


Figure 5.23: Morton hybrid layout: 16×16 matrix.

5.2 Problem of the hierarchical tiling data layout

As shown in Section 4, parallel accessibility for both tile and line as well as the reduction of cache conflict misses is highly effective for achieving high-speed 2-D data transfer. The conventional Z-Morton layout has been used to improve 2-D spatial reference locality and reduce conflict misses efficiently by a lot of previous works. However, the following two problems prevent the conventional Z-Morton layout to meet these requirement:

- To support parallel cache line-sized tile access, the memory must be divided into several 2-bytes-wide banks corresponding to the 2-byte-size subline. This division causes an area increase of the cache memory.
- The conventional Z-Morton layout is not suitable for row or column data parallel access, because the cascaded subline addresses are non-contiguous in both row and the column directions.

Similarly to the Z-Morton layout, the block data layout and Morton hybrid layout are also used to reduce TLB misses. They also cannot provide parallel data access in both row and the column directions since the subline address in the row or column direction are non-contiguous. In addition, an inevitable significant overhead is caused by address calculation when these hierarchical tiling data layouts are usually implemented on software. To solve these problems, this thesis proposes a 4-level Z-order tiling data layout and a Cache-based hybrid Z-ordering layout based on a hardware-based address conversion. The hardware conversion can eliminate the address calculation overhead but also maximize utilization of 2-D reference locality. Parallel tile and line access can be performed by using our proposed layout and the multi-bank cache organization that supports skewed storage scheme.

6 Proposed cache memory with tile and line accessibility

6.1 Proposed Multi-level Z-order tiling data layout

Although SIMD extensions are nowadays commonly found on most architectures, such as AVX or AVX2, their effects are limited by a lack of column-directional parallel access capability. Therefore, the parallel access of both tiles and lines is required for achieving high-speed 2-D data transfer as well as the reduction of cache conflict misses and TLB misses. The conventional hierarchical tiling data layout can improve 2-D spatial reference locality and reduce conflict misses and TLB misses efficiently. However, several problems (as shown in Section 5.2) prevent the hierarchical tiling data layout to meet these requirement. To eliminate these problems and provide substantial 2-D spatial reference locality, the author proposes two new data layouts based on the conventional Z-Morton layout and Morton hybrid layout called the 4-level Z-order tiling layout and the Cache-based hybrid Z-ordering layout.

6.1.1 4-level Z-order tiling data layout

The proposed 4-level Z-order tiling data layout hierarchical divides data into 3-level tiles, each of which is arranged in raster scan order at the same level, as shown in Figure 6.24. The majority of modern processors support memory pages of large sizes (4 Mbytes or 2 Mbytes), which are called “large pages” or “super pages”. The large page covers a larger address range than the small page so that using large pages can reduce TLB misses. Therefore, the 1st-level large tile size is equal to a large page size of 4 Mbytes. The 2nd-level medium tile is chosen equal to a page of 4 Kbytes, which is the size commonly used in conventional computer systems. The 3rd-level tile is simultaneously accessed to and from the cache memory, and its size is equal to the cache line size. Therefore, parallel data access in the column direction can be realized as this tile access. The tile access can eliminate the transposition required in matrix calculation, FFT or DCT. The tile access can also provide efficient 2-D unit block access for image processing and video coding, although its utilization might require significant modification of the program code.

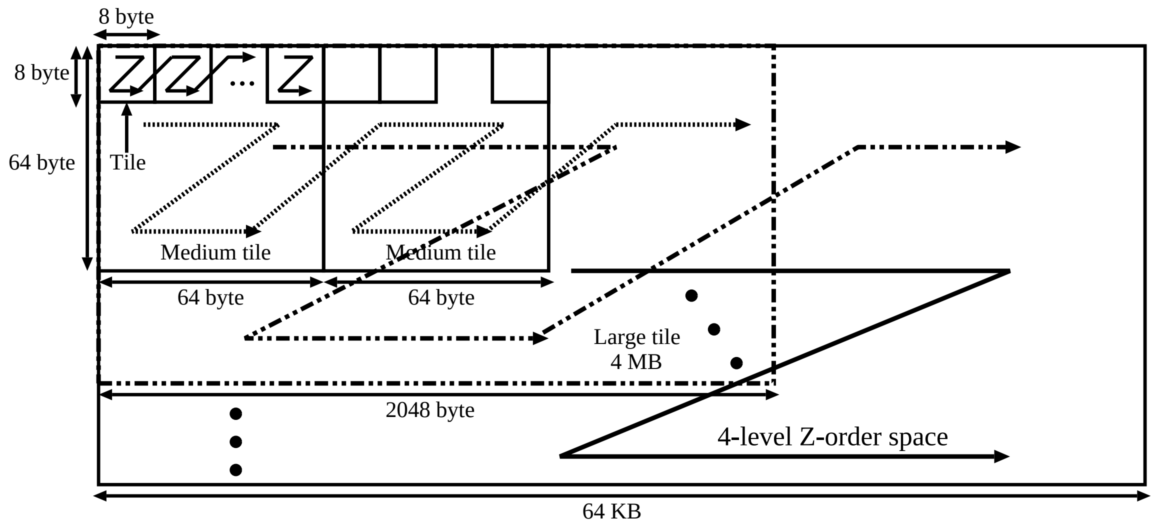


Figure 6.24: 4-level Z-order tiling layout.

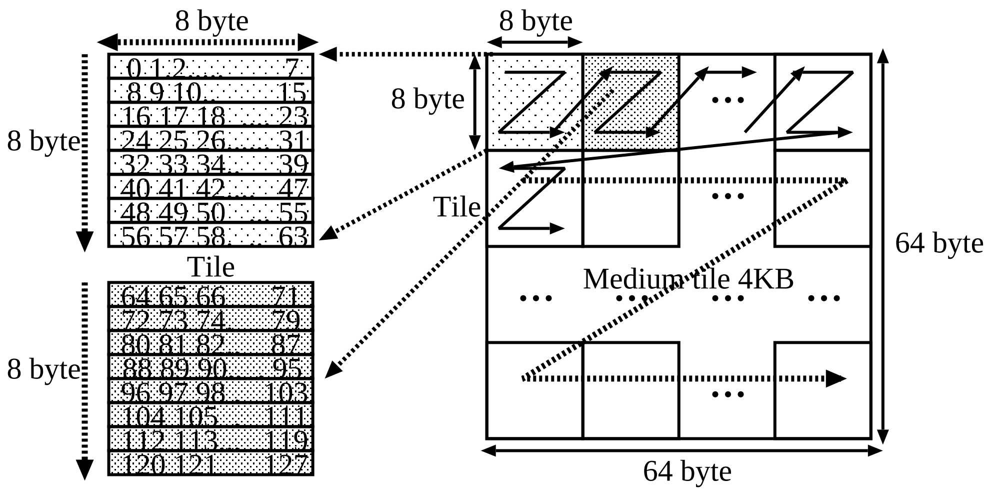


Figure 6.25: Medium tile and tile store data in raster scan order.

Figure 6.25 shows that a medium tile composed of 8×8 tiles store data in raster scan order, and the conventional 64-byte cache line stores an 8×8 byte-sized tile. The numbers in the tile represent 8-byte data that are contiguously accessed in the column direction. Finally, the width of the 4-level Z-order tiling space is 64 Kbytes, as shown in Figure 6.24. The large tile is 4 Mbytes and medium tiles (4 Kbytes) are arranged in raster scan order in each large tile. Therefore, the division of the address space into medium or large tile can minimizes the TLB updates, TLB misses, and excessive data transfers between the main memory and the auxiliary storage.

6.1.2 Cache-based hybrid Z-ordering layout

The 4-level Z-order tiling data layout divides data into 3-level tiles, each of which is arranged in row-major order at the same level, as shown in Figure 6.24. If a 2-D program contiguously accesses data in the column direction, it may causes allocation of unused regions in the virtual memory because the medium tiles in the column direction are not mapped onto the location of contiguous memory. The author proposes a Cache-based hybrid Z-ordering layout to avoid this problem.

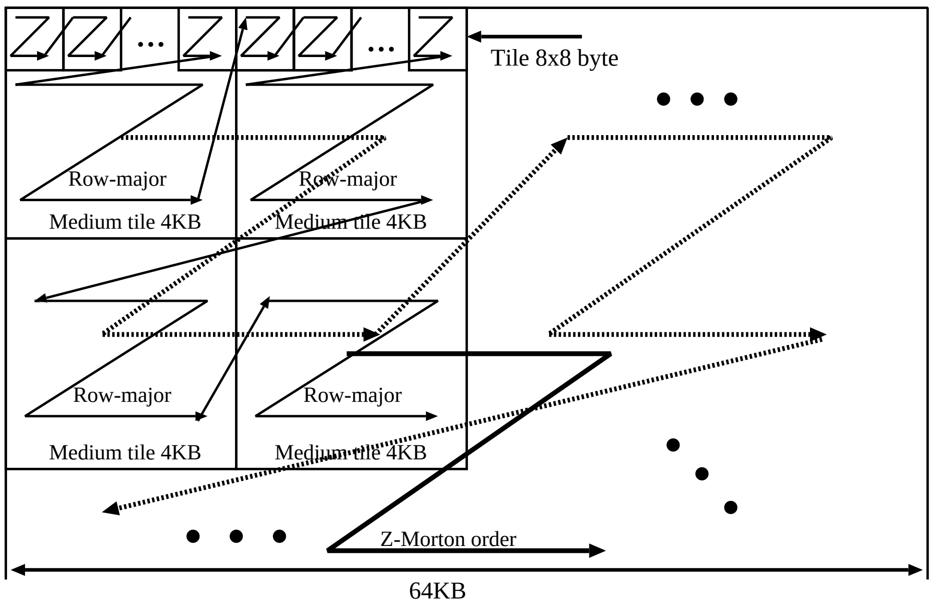


Figure 6.26: Cache-based hybrid Z-ordering layout (Medium tile and tile store data in raster scan order).

Data in the proposed layout is recursively subdivided into equally sized tiles stops at medium tiles with a size of T (4-byte-word×32). As shown in Figure 6.26, the size of a medium tile is equal to 4 KB because modern computer systems normally support a 4 KB page size. The division of the address space, which is similar to the 4-level Z-order tiling data layout, into small or medium tiles, whose internal data tags are equal to one another, matches the 2-D spatial reference locality to not only minimize TLB updates or misses, but also excessive data transfers between the main memory and auxiliary storage. A conventional 64 byte-sized cache line in each medium tile is divided into 8×8 byte-sized unit tiles. Unit tile access corresponds to parallel access of contiguous double words in the column direction. As shown in Figure 6.26, row-major order is adopted inside the unit tiles and medium tiles. Data outside the medium tile are stored in Z-Morton ordering among medium tiles. Based on the proposed layout, allocation of unused regions in virtual memory can be significantly reduced.

6.1.2.1 Utilization of hardware prefetching

The Cache-based hybrid Z-ordering layout can also exploit constant-stride hardware prefetching and cache tiling as well as the locality of the Z-Morton layout. Athanasaki [35] proved that prefetching in combination with cache tiling and block data layout, set optimal tiling size being equal to L1 cache size can decrease the L2 and TLB misses. However, this approach could only exploit prefetching in the major-directional access. Compared with this previous research, the proposed Cache-based hybrid Z-ordering layout has three main features:

- It can exploit constant-stride prefetching in both row- and column-directional access.
- It can be used for 2-D data processing because all levels of tiles are mapped into contiguous memory locations. Data in each tile are accessed in raster scan order and prefetching can be satisfactorily exploited.
- The medium tile size is equal to the cache way size and page size so that prefetch can be effective for the column-directional access in the same medium tile. As a result, the probability of crossing 4 KB page boundaries is less than 1/32 in the column-directional contiguous access and TLB misses can be minimized in processing using cache blocking techniques.

For example, consider the tiled MM that is shown in Listing 4. Arrays A[i][k] and B[k][j] can be loaded by using constant-stride prefetching in corresponding row direction and column direction. The matrix size is 32. The author has defined the size of a medium tile to be 256×16 byte because the MM accesses data in the row direction by 32×8 byte (256 byte).

Listing 4: The basic tiled MM version (6-loop tiled code).

```
int main()
{
double A[N][N], B[N][N], C[N][N];
 for(ii = 0; ii < N; ii+=32)
  for(jj = 0; jj < N; jj+=32)
   for(kk = 0; kk < N; kk+=32)
    for(i = ii; (i < N && i < ii + 32); i++)
     for(j = jj; (j < N && j < jj + 32); j++)
      for(k = kk; (k < N && k < kk + 32); k++)
        C[i][j]+= A[i][k] * B[k][j];
}
```

By using constant-stride prefetching, there is only one miss occurs in the row-directional access (32 accesses) for array A[i][k]. In addition, two misses occur in the column-directional access (32 accesses) for array B[k][j] because there are only 16 rows in each medium tile. Column-directional access crosses the boundary of the medium tile, which causes additional cache misses and TLB misses. Therefore, the total cache hit rate is 1 - 3/64 = 95.3%. Column-directional access causes a TLB miss per 16 access and row-directional access causes a TLB miss per 32 accesses in terms of TLB performance. Therefore, the TLB hit rate for column-directional access is 1 - 1/16 = 93.8%. The TLB hit rate for row-directional access is 1 - 1/32 = 96.9%. Table 6.5 summarizes the theoretical cache and TLB hit rates for the three data layouts.

For the raster layout, the row-directional access only causes 1 miss per 32 accesses if sequential unit-stride prefetching is used. Column-directional access causes 32 misses per 32 accesses because each stride access crosses 4 KB page boundaries. As a result, hardware prefetching cannot be used, but the cache hit rate is only 1 - 33/64 = 48.4%. Table 6.5 shows the theoretical hit rate of different level of memory hierarchy for a MM by using double-precision floating-point operations.

The author defines the size of a medium tile to be 128×32 byte for another MM by single-precision floating-point operations because MM accesses the data in the row direction by 32×4 byte (128 byte). By using constant-stride prefetching, there is

Table 6.5: Theoretical cache and TLB hit rate for MM with double-precision floating-point operations. The author does not consider the additional conflict misses due to 2-D data locality.

	Row-major layout	Proposed layout	Column-major layout
64 byte cache line	48.4%	95.3%	48.4%
4 KB page $*$	99.8%	96.9%	0%
4 KB page $**$	0%	93.8%	99.8%

$*$(row-directional access), $**$(column-directional access)

only one miss occurred in the row-directional access (32 accesses) for array A[i][k]. In addition, there is only one misses occurred in the column-directional access (32 accesses) for B[k][j] because there are 32 rows in each medium tile. Therefore, the total cache hit rate is 1 - 2/64 = 96.9%. Column-directional access causes a TLB miss per 32 accesses and row-directional access causes a TLB miss per 16 accesses in terms of TLB performance. Therefore, the TLB hit rate for column-directional access is 1 - 1/32 = 96.9%. The TLB hit rate for row-directional access is 1 - 1/16 = 93.8%. Table 6.6 summarizes the theoretical cache and TLB hit rates for the three data layouts.

For the raster layout, the row-directional access causes only 1 miss per 32 accesses if the sequential unit-stride prefetching is used. Column-directional access causes 32 misses per 32 accesses because the stride crosses 4 KB page boundaries. As a result, hardware prefetching cannot be used, but the cache hit rate is only 1 - 33/64 = 48.4%. Table 6.6 shows the theoretical cache and TLB hit rate.

Table 6.6: Theoretical cache and TLB hit rate for MM with single-precision floating-point operations. The author does not consider the additional conflict misses due to 2-D data locality.

	Row-major layout	Proposed layout	Column-major layout
64 byte cache line	48.4%	96.9%	48.4%
4 KB page $*$	99.8%	93.8%	0%
4 KB page $**$	0%	96.9%	99.8%

$*$(row-directional access), $**$(column-directional access)

6.1.2.2 Strassen algorithm for MM

Strassen came up with a recursive MM algorithm to multiply N×N matrices in 1969, by using fewer arithmetic operations, which ran faster than the conventional MM algorithm [36]. The Strassen method of MM is a typical divide and conquer algorithm. In contrast to the conventional MM algorithm that involves 8 multiplications and 4 additions, the Strassen algorithm is based on a scheme involving 7 multiplications and 18 additions for the product of two 2×2 matrices. This basic scheme that performs a single level recursion on 2×2 tile is:

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \text{x} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Figure 6.27: Strassen algorithm performs a single level recursion 2×2 tile.

Table 6.7: Strassen algorithm.

Level 1	T1 = A11 + A22	T6 = B11 + B22
	T2 = A21 + A22	T7 = B12 - B22
	T3 = A11 + A12	T8 = B21 - B11
	T4 = A21 - A11	T9 = B11 + B12
	T5 = A12 - A22	T10 = B21 + B22
Level 2	Q1 = T1 × T6	Q5 = T3 × B22
	Q2 = T2 × B11	Q6 = T4 × T9
	Q3 = A11 × T7	Q7 = T5 × T10
	Q4 = A22 × T8	
Level 3	T1 = Q1 + Q4	T3 = Q3 + Q1
	T2 = Q5 - Q7	T4 = Q2 - Q6
Level 4	C11 = T1 - T2	C12 = Q3 + Q5
	C21 = Q2 + Q4	C22 = T3 - T4

The divide and conquer technique utilized in the Strassen algorithm has a drawback. If the division proceeds to the level of small sized matrix elements, uncontiguous different data loads causing cache and TLB misses increase so that the recursion overhead becomes significant. Therefore, the Strassen algorithm provides poor performance compared with the conventional MM algorithm for small sized matrices because the L2 and L3 cache misses occur frequently [37] [38]. As shown in Table 6.7, 10 temporary variables are required in level 1, seven in level 2, and four in level 3. In order to reduce the overhead, we have to stop recursion at a sufficiently large-sized tile whose submatrices perform the conventional MM algorithm.

The Strassen algorithm internally uses a hierarchical tiling data layout that is known as the Z-Morton layout [39]. However, if the Z-Morton layout recursively divides 2-D data until the single level of a 2×2-sized tile, combining the Strassen algorithm with the Z-Morton layout provides poor performance for small sized matrices due to the address conversion overhead and excessive data transfer.

The proposed layout recursively divides the data into equally sized tiles until a given truncation size of 128 bytes is reached in the row direction, which results in a base medium tile size 128×32. Combining the Strassen algorithm with the proposed layout can improve MM performance for small sized matrices of 64×64. Outside the medium tile, data is stored in Z-Morton ordering. Therefore, all addresses are stored in the contiguous memory location and the hardware prefetching technique can be exploited. As a result, the number of L2 and L3 cache misses, as well as the TLB misses can be reduced and the Strassen algorithm can be effective for any matrix size of more than 64×64 by using the proposed layout.

6.2 The proposed cache architecture

In Section 6.1, the author presented the 4-level Z-order tiling layout based on the Z-Morton layout and the Cache-based hybrid Z-ordering layout based on the Morton hybrid layout, which can effectively exploit 2-D data locality and reduce TLB misses in the column direction as well as those in the row direction. In this section, the author proposes a new cache memory with tile and line accessibility to eliminate the overhead of the Z-order tiling address calculation and support both parallel tile access in the column direction and parallel line access in the row direction. The proposed two layouts provide 8×8 byte tile access that enables parallel access of eight double-precision floating-point data in the column direction which is useful in scientific computation utilizing SIMD operations.

As shown in Figure 6.28, the proposed cache is the area enclosed by the dotted line. The cache consists of an address conversion unit and multiple tag and data memory banks. It is an 8-way set associative cache of 32 Kbytes with a 64-byte cache line. Furthermore, the proposed cache divides tag and data memory into eight banks and provides access function of both 8×8 byte-sized tile or 64-byte-sized line to the 4-level Z-order tiling layout. On the other hand, the proposed cache with a 32-byte cache line divides the tag and data memory into four banks to provide access function of the 8×4 byte-sized tile and the 32-byte-sized line [15] [16]. The second and third level caches and the main memory adopt the 4-level Z-order tiling layout. In this paper, the author primarily shows the specific design of the proposed cache with a 64-byte cache line.

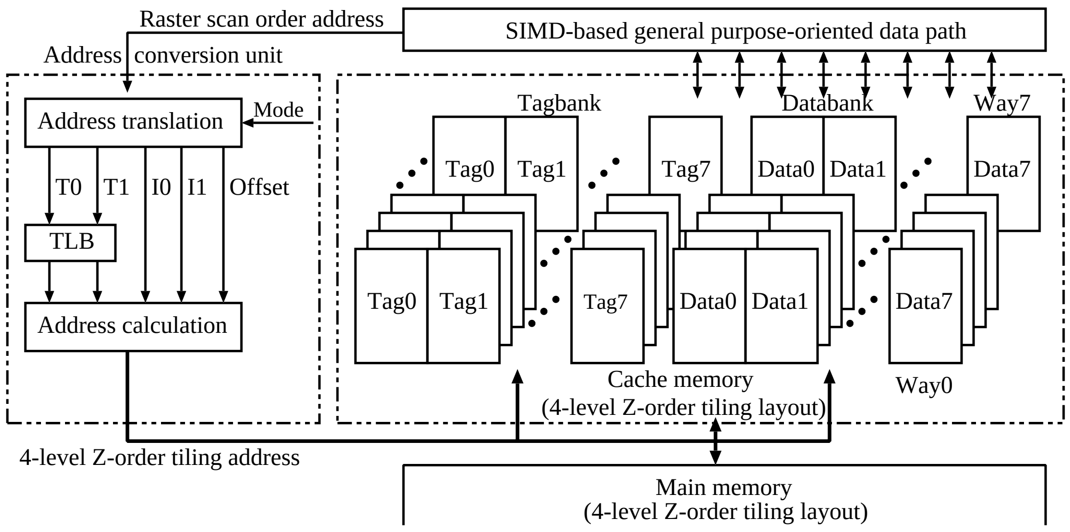

Raster scan order address
SIMD-based general purpose-oriented data path
Address
conversion unit
Address translation
Mode
T0
T1
I0
I1
Offset
TLB
Address calculation
Tagbank
Databank
Way7
Tag0
Tag1
Tag7
Data0
Data1
Data7
Cache memory
(4-level Z-order tiling layout)
Way0
4-level Z-order tiling address
Main memory
(4-level Z-order tiling layout)

Figure 6.28: Block diagram of the proposed cache memory.

6.2.1 Parallel tile and line accessibility

The proposed cache provides 8×8 byte-sized tile and 64 byte-sized line accessibility. Tile access corresponding to parallel column-directional data access can eliminate the transposition in matrix computation. Therefore, the proposed cache provides parallel data accessibility in both row and column directions. In addition, since both row and column-directional several contiguous data are accessed as a line or a tile, excessive data transfer to and from the external processor or the lower level cache is reduced. Figure 6.29 shows an example of the 8×8 byte-sized tile access for motion estimation. In the best case, excessive data transmission is not occurred when reading the 16×16 macroblock. In the worst case, excessive data transmission is suppressed from 128×16 - 16×16 = 1792 byte to 8×3×8×3 - 16×16 = 320 byte.

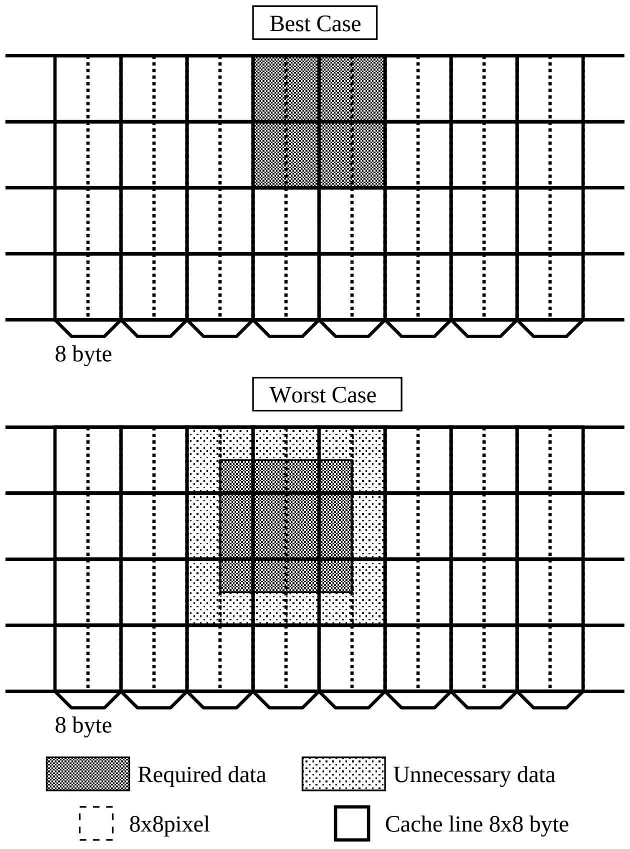


Figure 6.29: Tile data access.

6.2.2 Improvement of data transfer speed

The proposed cache adopts the 4-level Z-order tiling data layout. Cache line-sized tile access corresponding to parallel data access in the column direction can exploit 2-D locality. The theoretical hit rate of the proposed layout for row-directional and column-directional access is almost the same as that of for major-directional access of the row- and column-major layouts, respectively. Therefore, data transfer efficiency between processor and main memory can be improved significantly.

6.2.3 Improvement of degree of coding flexibility for 2-D program

The 4-level Z-order tiling data layout can exploit 2-D reference locality in both row and column direction. The proposed cache can minimize TLB misses in 2-D processing by only using the conventional simple tiling code. Therefore, it can also minimizes the optimization efforts of an original tiling code in which it is difficult to utilize the manually coded libraries such as OpenBLAS or Intel MKL. In other words, the proposed cache that supports column-directional parallel access allows either row-major based or column-major based 2-D program coding so that it increases the degree of coding flexibility. Therefore, the programmer's coding burden to improve processing efficiency can be reduced by using our proposed cache.

6.2.4 Load instructions reduction by using SIMD extensions

The proposed cache provides parallel column-directional access and row-directional access so that it can improve the instruction throughput with low latency overhead of the cache memory. By using SIMD extensions, the number of load instructions required for column-directional access (8×8 byte-sized tile access) or row-directional access (64 byte-sized line access) can be reduced to only one eighth of that required for conventional raster line access.

6.3 Parallel tile and line access mechanism

Tile access corresponding to column-directional parallel data access can eliminate the transposition required in matrix calculations, FFT, DCT, 2-D FIR filter and so on. Also, it can effectively reduce the amount of excessive data transfer and satisfactorily exploit the 2-D data locality as compared with the conventional raster line access because image processing applications operate on small blocks in 2-D data [40]. However, line access as cache line access is still essential for current conventional applications. Therefore, in this section, the author proposes a new cache memory with both tile and line accessibility that is based on a multiple eight-bank memory array structure. The memory banks are defined as tag-bank0-7 and data-bank0-7, as shown in Figure 6.28. Multiple sublines composing each tile are loaded from the L2 cache to the memory banks. However, if some of the sublines are stored in the same bank, the whole tile may not be accessed in a single read cycle. Figure 6.30 shows a conventional storage scheme. Sublines of each tile are stored in different memory bank and can be read out in parallel. However, sublines of each line are stored in the same memory bank cannot be read out in parallel.

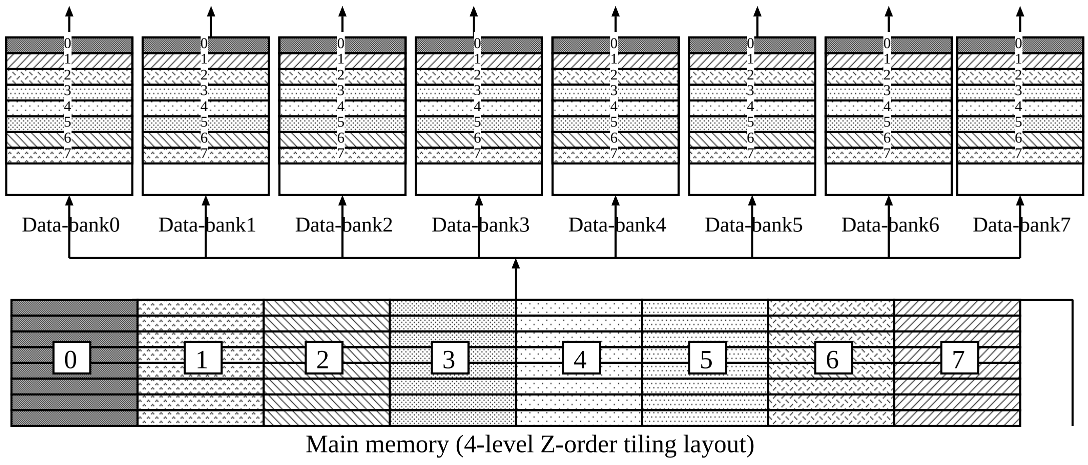


Figure 6.30: Conventional storage scheme.

To eliminate this problem, the author adopts a skewed array scheme to store the tile or line to the cache memory. Figure 6.31 and 6.32 show that the parallel tile and line are loaded to the processor, respectively. The line can be accessed by feeding a common address signal to all banks. On the other hand, the tile can be accessed by incremental address feeding between neighboring banks. The skew and un-skew operations are left to the external processor because their circuit implementation causes an overhead in the hardware scale and transfer delays.

To provide the parallel 8×8 byte-sized tile and 64 byte-sized line access function, the data memory of the proposed cache must be divided into eight banks (data-bank0-7), as shown in Figure 6.28. Each tile and line can be accessed in parallel via the skewed array scheme. Figure 6.31 shows that the fourth tile are transferred to the processor. After the sublines of the tile are loaded to the processor register, additional shift operations are need to align the sublines. Figure 6.32 shows a line access that the fourth row

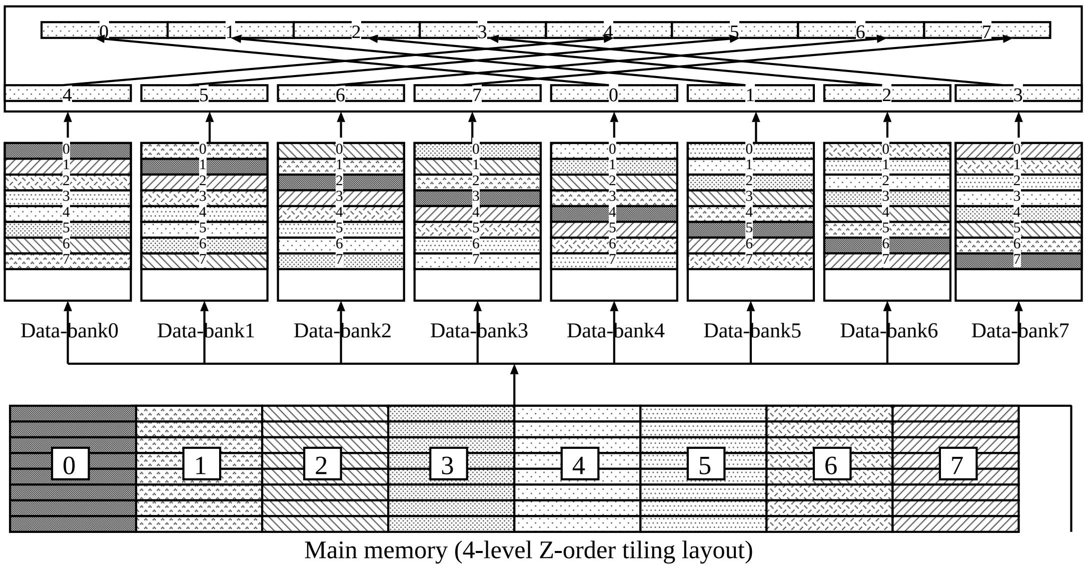


Figure 6.31: Parallel tile and line access scheme (line access).

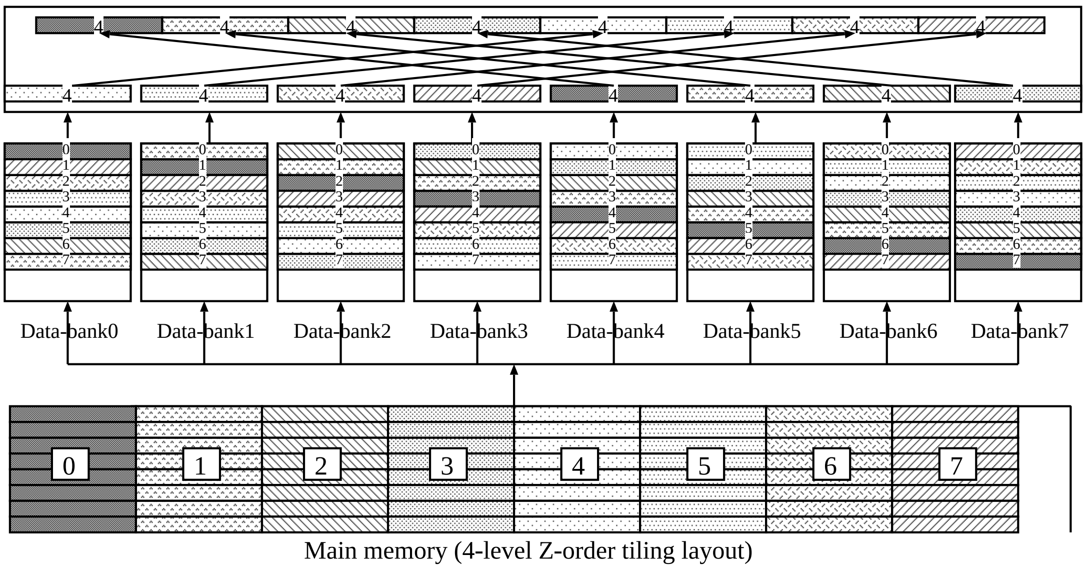


Figure 6.32: Parallel tile and line access scheme (tile access).

elements are loaded to the processor. Additional shift operations also need to align the elements. This parallel aligned/unaligned data access in the row/column direction can improve the instruction throughput with low latency overhead of the cache memory. The number of load instruction required for column-directional contiguous access is only one eighth of that required for conventional raster line access.

6.4 Hardware-based address bit-order interchange unit for raster scan order to the 4-level Z-order tiling space

In this section, the author proposes a hardware-based address translation unit that transparently translates the conventional raster scan order address to a 4-level Z-order tiling address by adding an additional pipeline stage. Our hardware-based method has the following differences from conventional software-based address translation:

- Address translations using the hardware unit not only improve data locality but also eliminate the processing overhead for Z-order address calculation and reduce the programmer's workload. To improve 2-D data locality, the 1st, 2nd and 3rd level tiles of the 4-level Z-order tiling layout are aligned to 4 Mbytes, 4 Kbytes, and 64 byte in Z-order, respectively.

- The author adopts a pure hardware-based address translation unit. A logical address of raster layout can be transformed to a virtual address of 4-level Z-order via simple bit-order interchange operations in the hardware transparent to the OS or software. As a result, the 4-level Z-order tiling layout is accessed as a conventional raster layout.

- For applications such as MM, if the width of the 2-D processing area is not fixed, a different address bit-order interchange is required for the width when the matrix size changes. This complex address translation increases the address translation overhead and may cause a clock-cycle constraint problem [17]. To avoid this different address bit-order interchange and simplify the address calculation, the individual width of the 4-level Z-order tiling area is fixed to the sufficiently large size of 64 Kbytes-wide, which meets almost all space requirement for 2-D applications.

As shown in Figure 6.28, via the address translation unit, it is possible to convert a logical address in raster scan order from an external processor into a virtual address to allow an external processor to access the 4-level Z-order tiling data as conventional raster scan order data. For almost all 2-D data applications, the width of the 2-D data to be processed is smaller than 64 Kbytes. Therefore, to locate the entire 2-D data in the 4-level Z-order tiling space, the author replaces the specific single-level Z-order tiling address translations, such as 256 byte, 512 byte and 1Kbyte widths [11] to only a 4-level Z-order address translation of a sufficient width of 64 Kbytes. This proposed hardware-based address translation unit can further simplify the address calculation, compared with previous work [17].

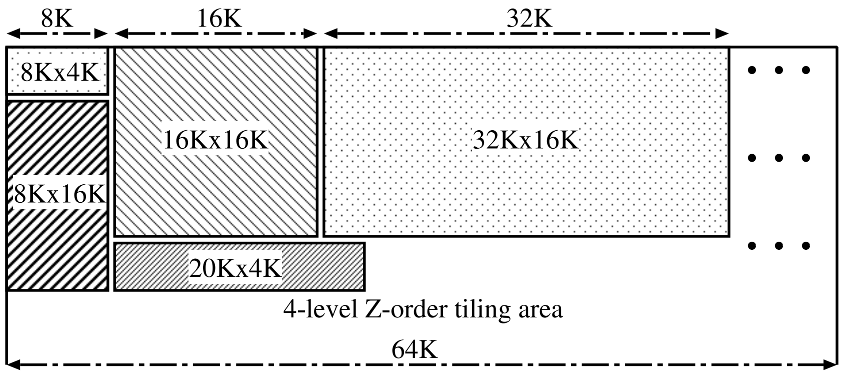


Figure 6.33: Various sizes of 2-D data allocation in the 4-level Z-order tiling layout.

Figure 6.33 shows how 2-D data of various sizes are allocated to the 64 Kbytes-wide 4-level Z-order tiling area, in which the main memory and the L2/L3 cache store all data in the same way as the proposed L1 cache memory. Figure 6.34 shows a memory allocation assigning the tiling area between the stack area and static area in the dynamic area. The higher and lower address pointers determine the address range of the tiling area. Inside the tiling area, all data are stored by tile. Outside the tiling area, data are stored by line in the conventional raster scan order. In addition, line accessibility to the tiling area provides a transfer function between the tiling area and the raster scan order area. In the transfer operation, the compiler assigns load/store instructions for the line in the tiling area and normal load/store instructions in the outside area.

The translation from the conventional raster scan order address to the 4-level Z-order tiling address is performed by the address bit-order interchange, as shown in Figure 6.35. In addition, the proposed tiles and lines are stored in the tiling area. The conventional raster lines are accessed in raster scan order outside the tiling area. Consider a 2-D application in which any data are accessed in- or outside the tiling area. If a cache miss occurs on a tile/line read, the corresponding tile/line is loaded

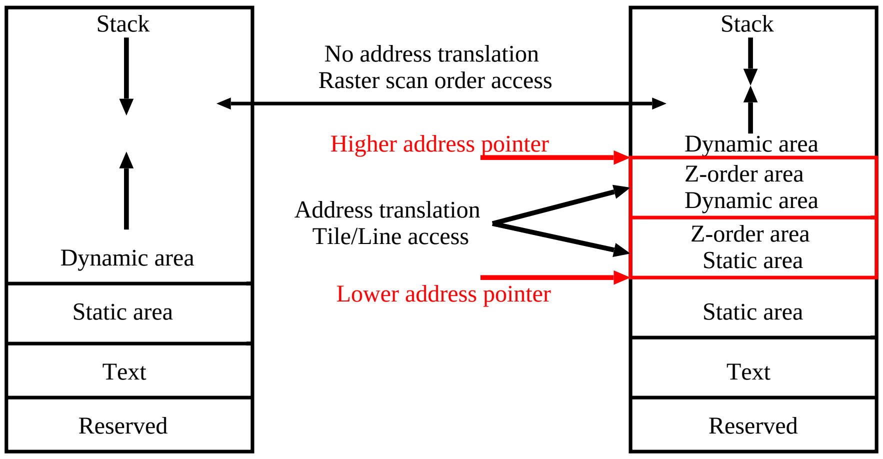


Figure 6.34: Memory allocation for 4-level Z-order tiling space.

from the tiling area in the virtual memory space. If another cache miss occurs on a conventional raster line read, the corresponding raster line is loaded from outside of the tiling area. Therefore, the tile/line from the tiling area and the conventional raster line from outside of the tiling area should be stored in the same cache. The inside and the outside of the Z-order area should be used for 2-D and 1-D data processing, respectively.

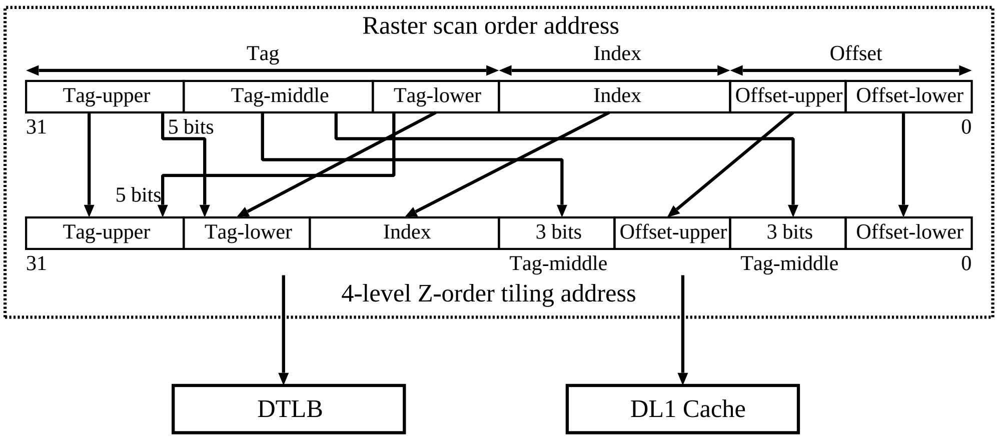


Figure 6.35: The address bit interchange.

The cache memory address consists of six fields. To convert a raster scan order address to a 4-level Z-order tiling address, the tag-middle and the index bit fields are interchanged, and then, the lower-order 3 bits of the tag-middle field are inserted between the offset-upper and offset-lower fields. Finally, the middle-order 5 bits of

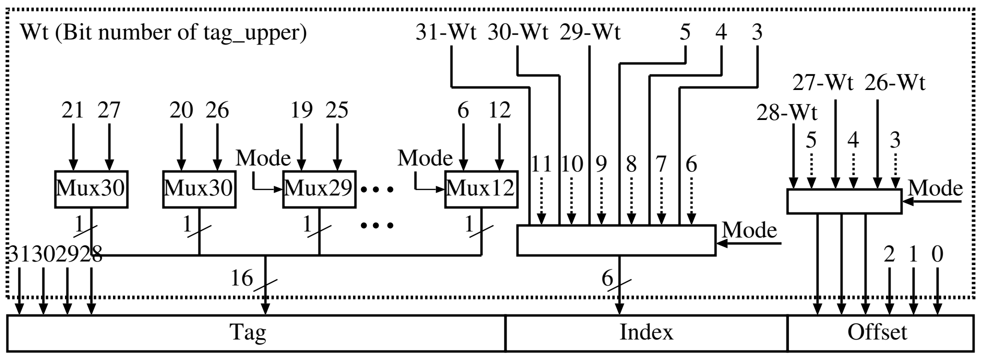


Figure 6.36: Address translation circuit.

the tag-lower field and lower-order 5 bits of the tag-upper field are interchanged, as shown in Figure 6.35. In this case, the tag-lower and index fields consist of 10 bits for the 64 Kbytes width (2^{10} (tag-lower + index) $\times$ 2^6 (offset) = 64 Kbytes). The higher-order 5 bits of the tag-lower field specify where a medium tile is placed in the column-directional position of a large tile. In addition, the higher-order 5 bits of the tag-middle field specify where a tile is placed in a medium tile, and the lower-order 3 bits of the tag-middle field specify where a constituent subline of a tile is placed in the column direction in each tile. Field offset-upper represents the tile number in the row direction in each medium tile. A translation circuit is shown in Figure 6.36. Since the translator consists of single-stage 2 to 1 multiplexers selecting the 4-level Z-order tiling address or the raster scan order address, it does not increase the access cycle time or the latency.

6.5 Tag memory reduction method

A conventional cache mainly consists of two arrays: tag and actual data. The tag array stores tag bits and valid bit for the cache line that are currently stored in the cache. The index bits of the effective address are used to look up a tag in the tag array while the combined index and offset bits are used to retrieve the data from the data array. If the valid bit retrieved from the tag array is 1 then the data retrieved from the data array is valid. The retrieved data might be valid but for a different address. Therefore, the tag of the stored data (retrieved from the tag array) is compared to the tag bits of the address. There is a cache hit if they are the same as each other and the valid bit is 1. The data memories of the proposed cache illustrated in Figure 6.37 and 6.38 shows that the sublines of each tile are stored in each data bank, respectively. Multiple sublines with the same color compose a tile.

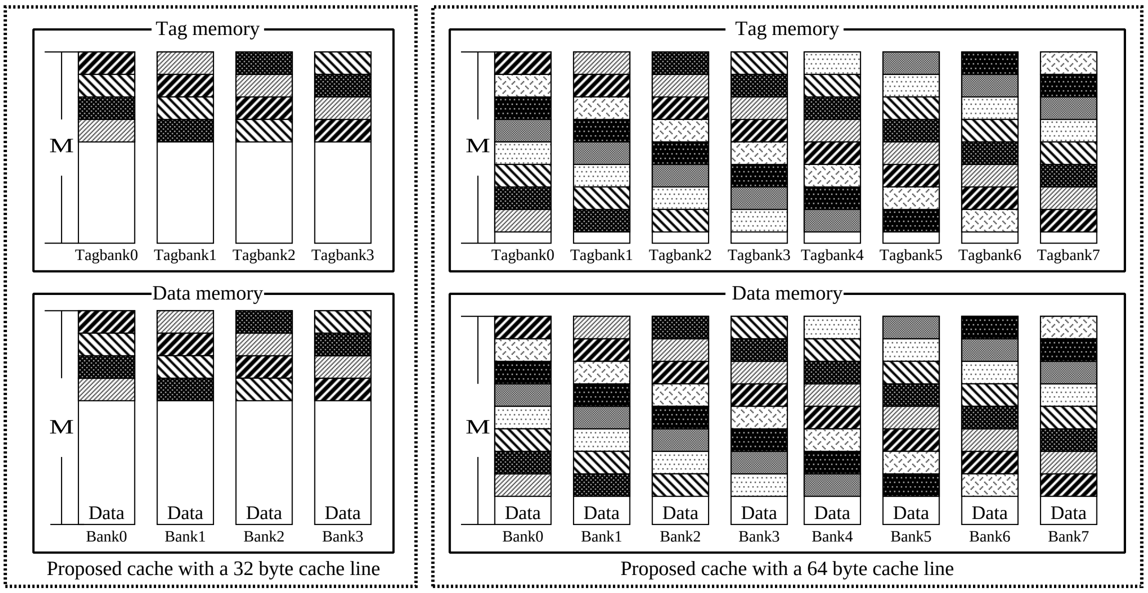

Tag memory
M
Tagbank0 Tagbank1 Tagbank2 Tagbank3
Data memory
Data Data Data Data
Bank0 Bank1 Bank2 Bank3
Proposed cache with a 32 byte cache line
Tag memory
Tagbank0 Tagbank1 Tagbank2 Tagbank3 Tagbank4 Tagbank5 Tagbank6 Tagbank7
Data memory
Bank0 Bank1 Bank2 Bank3 Bank4 Bank5 Bank6 Bank7
Proposed cache with a 64 byte cache line

Figure 6.37: Each tag is added to each constituent subline of a skewed array composing an aligned tile.

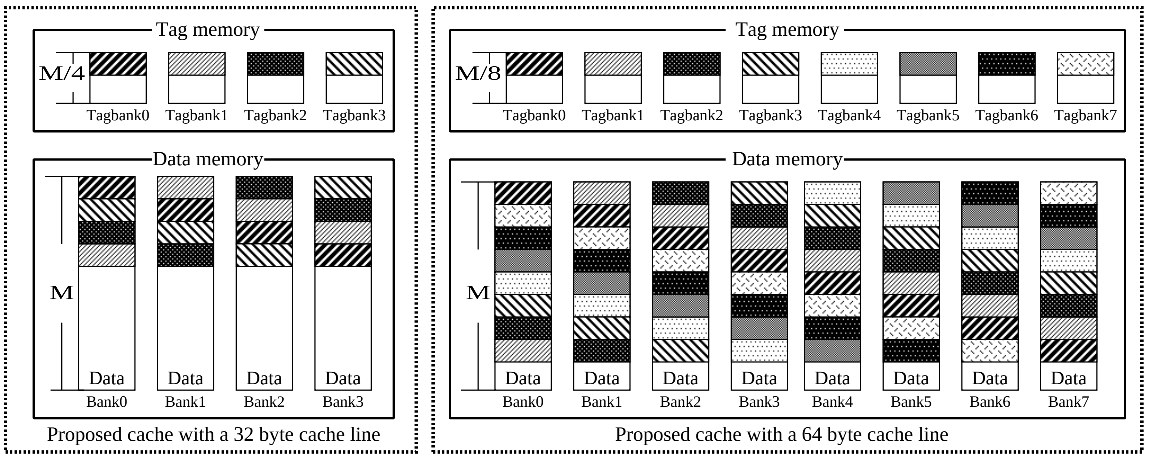

Tag memory
M/4
Tagbank0 Tagbank1 Tagbank2 Tagbank3
Data memory
M
Data Data Data Data
Bank0 Bank1 Bank2 Bank3
Proposed cache with a 32 byte cache line
Tag memory
M/8
Tagbank0 Tagbank1 Tagbank2 Tagbank3 Tagbank4 Tagbank5 Tagbank6 Tagbank7
Data memory
M
Bank0 Bank1 Bank2 Bank3 Bank4 Bank5 Bank6 Bank7
Proposed cache with a 64 byte cache line

Figure 6.38: Each tag is added to each aligned tile.

As shown in Figure 6.37, if a tag is allocated to each subline in the tile in the same way as that in a conventional cache (each subline of the tile has a “tag” and a “data” field), the capacity of the tag memory becomes 4 times (the proposed cache with a

32-byte cache line [19] [20]) or 8 times (the proposed cache with a 64-byte cache line in this paper) that of the conventional cache because the tag memory has to be divided into four- or eight-banks to store each subline tag. For suppressing this tag memory capacity increase, the author proposes a tag memory reduction method adding a tag to each aligned tile (each tile has a "tag" and a "data" field) and the capacity of the tag memory is ideally reduced to the same capacity as that of the conventional cache, as shown in Figure 6.38.

6.6 Unaligned tile access in the column direction

In the previous section, the author proposes a tag memory reduction method to suppress the hardware scale increase of the proposed cache. However, this method disables non-aligned tile access because the two tag data of the column-directionally adjacent tiles stored in the same tag bank are not accessed simultaneously.

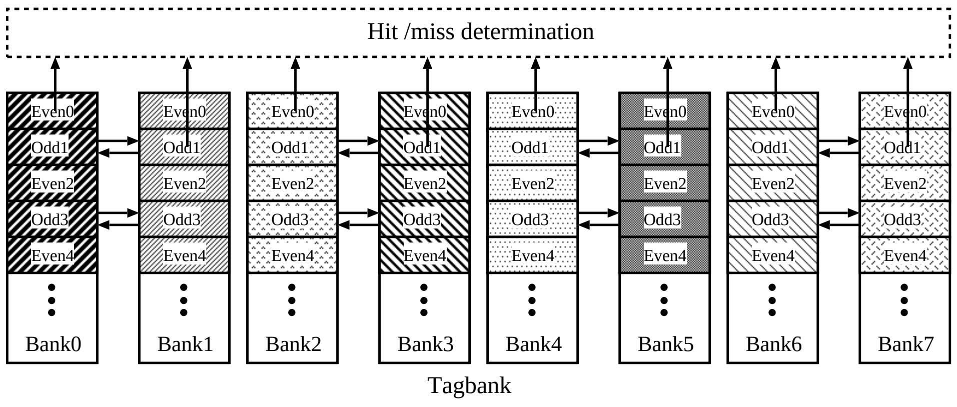


Figure 6.39: Unaligned tile access scheme.

To solve this problem, the storage locations of the tags in the odd-entries are interchanged between tag-bank0 and tag-bank1 or between tag-bank2 and tag-bank3 for the cache with a 32-byte cache line. If the cache line size is greater than 32 byte, the storage locations of the tags in the odd-entries are interchanged between tag-bankN and tag-bankN+1 (N is an even number and N = 0 $\sim$ Cache line size/8-1), as shown in Figure 6.39. As a result, the two tag data of column-directionally adjacent tiles can be read out in parallel. Therefore, the proposed cache can access column-directionally aligned/unaligned tiles and 8-byte boundary aligned lines in parallel.

6.7 Dual data access mode for 1-D/2-D data access

In the previous section, the author proposed a new cache memory with both tile and line accessibility for efficient 2-D data processing. To support the conventional raster line access for 1-D data processing, the author also adds a dual data access mode to the proposed cache memory.

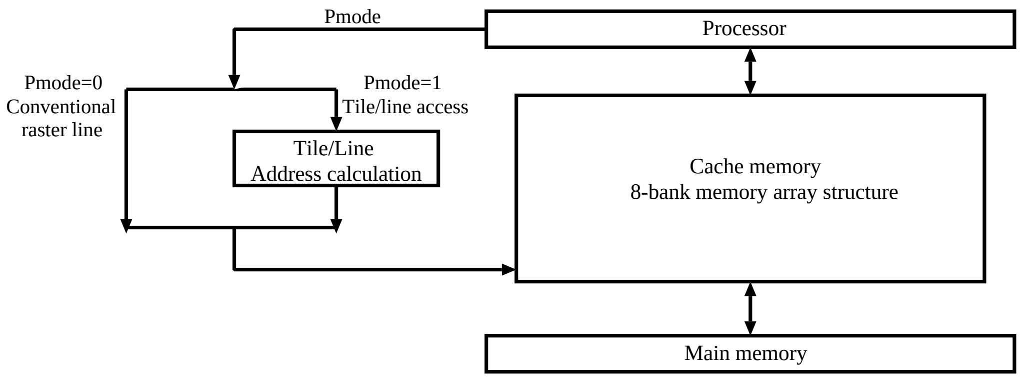


Figure 6.40: Proposed cache for both 1-D and 2-D data processing.

Figure 6.40 shows that the proposed cache can appropriately switch a 2-D data access mode for the proposed tile and line access to a 1-D data access mode for the conventional raster line access. In the 1-D data access mode (Pmode = 0), the proposed cache does not adopt the skewed array storage scheme. It works in the same way a conventional cache. Data in the cache and main memory are stored in raster scan order. The miss penalty of conventional 1-D data access to the proposed cache is not increased as compared with that of the conventional cache. Therefore, the conventional raster line access can also be achieved by switching the dual data access mode to 1-D data access mode.

7 Hardware scale optimization of the proposed cache

In Section 6.5, the author proposed a tag memory reduction method that can reduce the tag memory capacity of the proposed cache to the same as a conventional cache. However, via this method, the capacity of each tag memory bank is reduced to only 32 words (128 words / 4 -banks) or to 8 words (64 words / 8 -banks) for a cache memory with a 32-byte or 64-byte cache line, respectively. For VLSI hardware design, each tag memory bank is composed of one SRAM macro. The size of conventional SRAM macro is commonly 32 words. Therefore, the entire hardware scale of the proposed cache will be increased if the size of SRAM macro we use is smaller than 32 words. In addition, the entire hardware overhead of the proposed cache will also be increased because the tag memory is divided into 4 -banks or 8 -banks for the proposed cache with a 32-byte or a 64-byte cache line, respectively.

7.1 RATS-tag memory reduction method

To reduce the hardware scale overhead of the proposed cache while preserving high performance of the proposed cache, the author proposes a RATS-tag memory reduction method. The RATS method reduces the entire hardware scale and simplifies the cache architecture. The main features of the proposed RATS method are as follows:

- The proposed n-way set associative cache adopts the least recently used (LRU) replacement policy. The LRU replacement policy replaces the aligned tile set that has not been accessed for the longest time.
- If a line miss occurs, the processor loads an aligned tile set from the lower level cache or the main memory. An aligned tile set contains eight aligned tiles. Each tile of an aligned tile set can be stored in the same cache way set.
- If a line hit occurs, the cache updates the LRU bit of the aligned tile set containing the hit line. If a tile hit occurs, the cache updates the LRU bit of the accessed aligned tile.
- An enable flag is added to each aligned tile set for parallel access by line and is set when any tile of an aligned tile set are stored in the same cache way set, the

enable flag is set. The line of which enable flag is set can be read out from the cache if its enable flag is set.

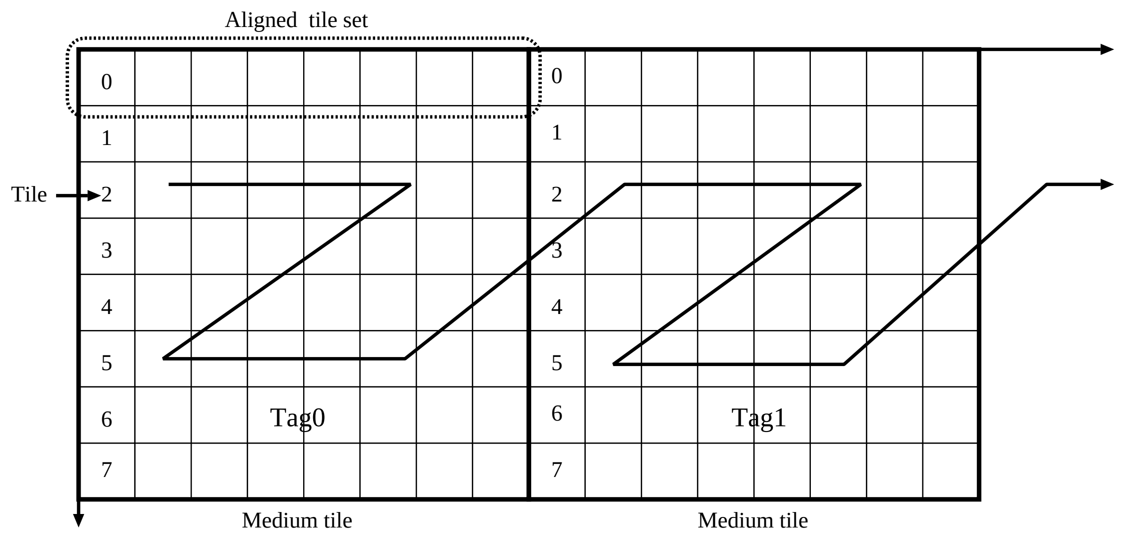


Figure 7.41: Aligned tile set in each medium tile.

Figure 7.41 shows the relationship between the aligned tile set and each tile. Each medium tile contains eight aligned tile sets and each aligned tile set contains eight aligned tiles. All the tiles in an aligned tile set have the same tag value. The number in the beginning of each line represents the index value of each aligned tile set. If a cache miss occurs due to the line access, an aligned tile set will be loaded from the lower level cache or main memory and all the tiles of the aligned tile set are stored to the same cache way set. Via this method, all sublines of the line are stored in the same cache way set and it is not necessary to read the tag values from the eight tag memory banks. Therefore, the number of tag memory banks of the proposed cache can be reduced to only two. As shown in Figure 7.42, to support unaligned tile access in the column direction, the tag values in each odd row of the medium tile are stored in tagbank0 and the tag values in each even row of the medium tile are stored in tagbank1. The RATS method can simplify the proposed cache architecture.

The miss penalty for line access is 8 times that of the conventional raster line access since the corresponding aligned tile set has to be loaded to the cache when a cache miss occurs. Furthermore, the author defines two LRU update modes of RATS-T for

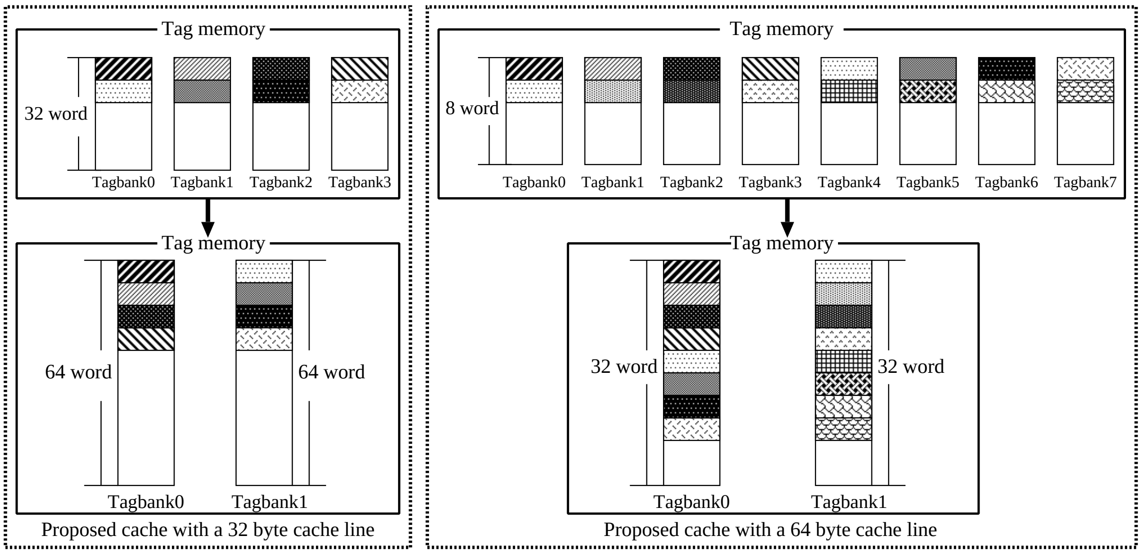


Figure 7.42: Two-bank tag memory structure.

tile access and RATS-S for line access. The RATS-T and RATS-S accesses update the LRU bit of aligned tile set at its constituent tile hit and at its constituent line hit, respectively. As shown in Figure 7.42, the number of tag memory banks for the proposed cache is reduced from four/eight to only two. The capacity of each tag bank is increased to 64 or 32 words for the proposed cache with a 32- or 64-byte cache line, respectively. These tag memory banks can be easily implemented by SRAM macro.

7.1.1 RATS-T access mode

The author defines the unit tile access as RATS-T access where the cache updates the LRU bit of the accessed aligned unit tile when a unit tile hit occurs. When a unit tile miss occurs, the cache loads the corresponding align tile set from main memory and stores each tile of the align tile set to the cache by using LRU replacement policy.

7.1.2 RATS-S access mode

The author defines the unit line access as RATS-S access where the cache updates LRU information by the aligned unit tile set when a unit line hit occurs. When a unit line miss occurs, the cache loads the corresponding align tile set from main memory and stores each tile of the align tile set to the same cache way set by using LRU replacement policy.

7.2 The proposed RATS-cache architecture

Figure 7.43 shows the proposed RATS-cache architecture. The proposed RATS-cache adopts two-bank tag memory structure. To support parallel line access, an 8-bit wide flag bank is added to the proposed RATS-cache. Each bit of the flag bank is added to each aligned tile set. An enable flag is set when any tile of an aligned tile set are stored in the same cache way set. Furthermore, the proposed RATS-cache divides data memory into eight banks to support parallel 8×8 byte-sized tile and 64-byte-sized line access.

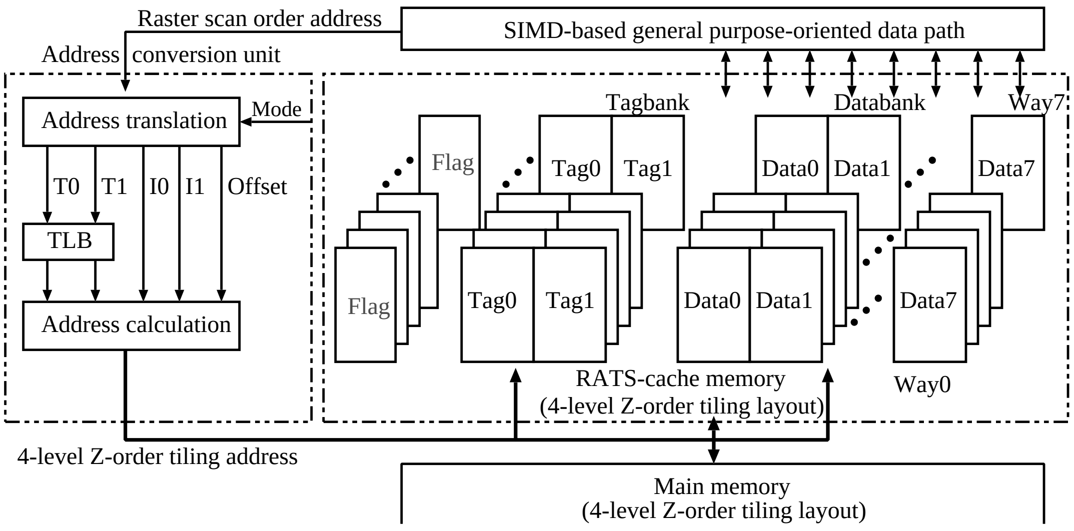

Raster scan order address
SIMD-based general purpose-oriented data path
Address conversion unit
Address translation
Mode
T0
T1
I0
I1
Offset
TLB
Address calculation
Tagbank
Databank
Way7
Flag
Tag0
Tag1
Data0
Data1
Data7
RATS-cache memory
(4-level Z-order tiling layout)
Way0
4-level Z-order tiling address
Main memory
(4-level Z-order tiling layout)

Figure 7.43: Block diagram of the proposed RATS-cache memory.

8 Evaluation

8.1 Hardware scale evaluation

8.1.1 Implementation

Here, the author presents the design of the proposed cache combined with a SIMD-based general purpose-oriented datapath [41] large-scale integration (LSI) to evaluate three main specifications: read and write latency, clock period, and hardware scale. The author compares our proposed cache with a conventional cache because the author is the first to design an on-chip cache with both tile and line accessibility and combine the cache with the SIMD-based general purpose-oriented datapath. In addition, the logic design is performed using the Systemverilog and Design Compiler under educational 0.18 µm complementary metal oxide semiconductor (CMOS) technology in the chip fabrication program of the VLSI Design and Education Center (VDEC).

The constitutions of the trial design are direct-mapped type with 4 Kbytes capacity, 2-way set associative type with 8 Kbytes capacity or 8-way set associative type with 32 Kbytes capacity. The direct-mapped cache adopts a 2-stage structure composed of the first address converting stage and the second memory access stage. The data access latency of the cache becomes 2 cycles due to this 2-stage structure. The proposed 2-way/8-way set associative cache adopts the LRU replacement algorithm. The LRU algorithm needs to use counters causing hardware cost and delay time increase. In addition, the 2-way/8-way set associative cache requires an additional stage for way selection; thus, its latency becomes 3 cycles being equal to that of the L1 data cache in the recent Intel or ARM processors, as shown in Table 8.9. The clock period of the Memory Macro the author uses is 3.43 ns and the clock period of the proposed cache memory is limited to 3.9 ns. Therefore, the evaluation result shows that the clock period of the proposed cache with the dual data access mode does not increase compared with that of the conventional cache.

8.1.2 Hardware scale overhead and speed performance evaluation

To show the feasibility of the proposed cache, the author combines the proposed 2-way set associative cache memory (32-byte cache line) with the SIMD-based datapath [41] and design its LSI chip [19] [20]. The reason using 2-way set associative cache (32-byte cache line) is that its hardware scale requirement is not huge and the SIMD-based datapath only provides the 32 byte data transfer function. The LSI chip layout is designed in a 2.5×5 mm^2 area by using 0.18 µm CMOS technology. The details of the specifications are shown in Table 8.8. Figure 8.44 shows the chip layout.

Table 8.8: Chip specification.

Technology	0.18 µm	Clock frequency	250 MHz
Chip size	2.5×5 mm^2	Memory Macro net area	2.86 mm^2
Aspect ratio	1.0	Datapath logic scale*	124184
Voltage supply	1.8 V	Peripheral circuit scale*	23073

*NUMBER OF NAND GATE EQUIVALENTS

The SIMD-based datapath occupies an area of 1.62 mm^2, which is 34% of the total layout area. The proposed 2-way set associative cache memory occupies the remaining area. The cache memory is composed of 48 SRAM macros for the data and tag memories, accounting for 46% and 14% of the total layout area, respectively. The minimum clock period becomes 4 ns due to the 3.9 ns cycle period of the proposed cache memory shown in Table 8.9 at the logic synthesis level. This minor change in the cycle period (from 3.9 ns to 4 ns) is caused by the 3.43 ns cycle period of each Memory Macro based on the educational design grade. In addition, the conventional n-way set associative cache and the proposed 2-way/8-way set associative cache have the same clock period.

Table 8.9: Cache speed.

Cache structure	Clock period	Latency
Conventional n-way set associative cache	3.9 ns	3-cycle
Proposed 2-way/8-way set associative cache	3.9 ns	3-cycle
Proposed Direct-mapped cache	3.9 ns	2-cycle

The hardware area overhead of the proposed cache compared with that of the conventional cache is shown in Table 8.10. The tag memory of the conventional cache

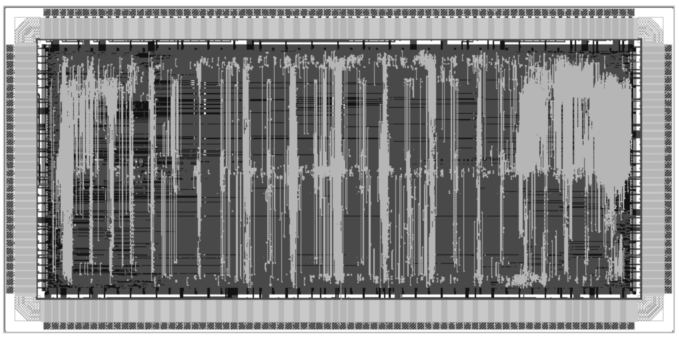


Figure 8.44: Chip layout.

is divided into two banks for unaligned cache line access in the row direction. For parallel unaligned tile/line access in both column and row directions, the proposed Non-RATS cache (cache that does not use the RATS method) has an eight memory banks for parallel tag access, as shown in Figure 6.28. In contrast, the proposed RATS cache has only two memory banks due to the RATS-T and RATS-S. Table 8.10 and Table 8.11 show the entire hardware overhead of the proposed cache, respectively. For example, by assigning only a tag to an aligned tile, the tag array area of the proposed Non-RATS cache with a 32-byte cache line is reduced from 4 times to 2 times in comparison to that of the conventional cache memory area. In addition, via the RATS tag memory reduction method, the proposed RATS cache and the conventional cache have the same tag memory area. Furthermore, the peripheral circuit scale of the proposed RATS cache increases to approximately two or three times that of the conventional cache. As a result, the entire scale of the proposed 8-way set associative RATS cache with a 32-byte cache line is only 1.05 times that of the conventional cache scale. The entire scale of the proposed 8-way set associative RATS cache with a 64-byte cache line is only 1.07 times that of the conventional cache scale. Therefore, our proposed cache can achieve high-performance with a negligible area cost and no latency increases to the conventional cache.

8.1.3 Critical path for loading

Figure 8.45 shows the critical path for loading of the proposed 2-way/8-way set associative cache. The critical path delays are obtained by summation of the following

Table 8.10: Hardware scale for the 2-way set associative cache (Number of NAND gate equivalents).

		32-byte cache line, 2-way		64-byte cache line, 2-way	
	Cache memory	Non-RATS	RATS	Non-RATS	RATS
Peripheral circuit scale	Conventional	10997	10997	17860	17860
	Proposed	23073	23885	35836	35701
	P/C	2.10	2.17	2.00	2.00
Tag memory scale	Conventional	26360	26360	26360	26360
	Proposed	26360×2	26360	26360×4	26360
	P/C	2.00	1.00	4.00	1.00
Data memory scale	Conventional	169230	169230	338460	338460
	Proposed	169230	169230	338460	338460
	P/C	1.00	1.00	1.00	1.00
Whole cache scale	Conventional	206587	206587	382680	382680
	Proposed	245023	219475	479736	400521
	P/C	1.18	1.06	1.25	1.04

Table 8.11: Hardware scale for the 8-way set associative cache (Number of NAND gate equivalents).

		32-byte cache line, 8-way		64-byte cache line, 8-way	
	Cache memory	Non-RATS	RATS	Non-RATS	RATS
Peripheral circuit scale	Conventional	84089	84089	54430	54430
	Proposed	182582	131203	260990	160848
	P/C	2.17	1.56	4.79	2.96
Tag memory scale	Conventional	105440	105440	105440	105440
	Proposed	105440×2	105440	105440×4	105440
	P/C	2.00	1.00	4.00	1.00
Data memory scale	Conventional	676920	676920	1353840	1353840
	Proposed	676920	676920	1353840	1353840
	P/C	1.00	1.00	1.00	1.00
Whole cache scale	Conventional	866449	866449	1513710	1513710
	Proposed	1070382	913563	2036590	1620128
	P/C	1.23	1.05	1.34	1.07

three times: (1)The time converting the raster scan order address into a 4-level Z-order tiling address. (2)The access time of the SRAM macro (tag memory and data memory). (3)The time for selecting data from 2-way/8-way set associative cache and for providing the result to the processor.

On the first cycle, the address conversion performs tiling address translation and generates multiple addresses by using bit-order interchange and addition operation. Each address is send to corresponding cache memory bank for loading tile and line. Since the address translator consists of multiple adder circuits and single-stage 2 to 1 multiplexers selecting the 4-level Z-order tiling address or the raster scan order address, it does not increase the access cycle time. In addition, the tile/line way selection for line access consists of eight single-stage 8 to 1 multiplexers selecting the subline from the cache, it does not cause a long delay time. The longest delay time is the access time for tag and data memory access. The minimum clock period of the proposed cache is suppressed to 3.9 ns.

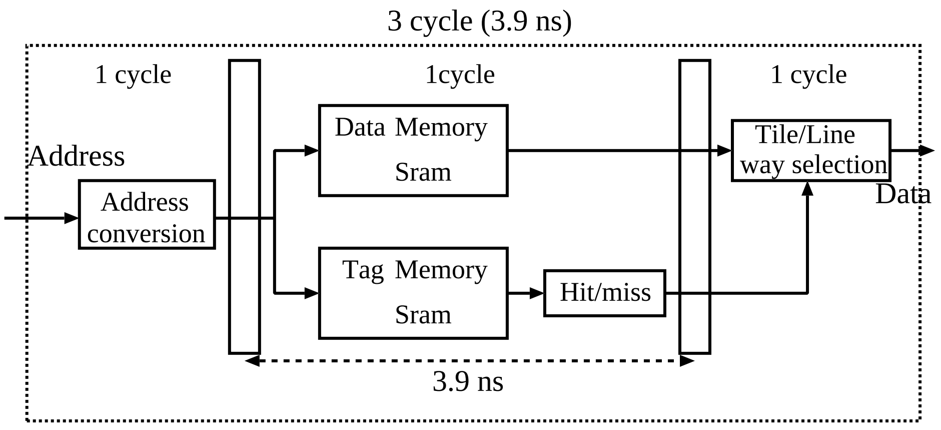


Figure 8.45: Critical path for loading of the proposed 2-way/8-way set associative cache.

8.2 Performance evaluation

To verify the effectiveness of our proposed cache memory, the author evaluates the performance of our proposed RATS method for MM and LUD. MM is the most important and useful operation in image processing and scientific computing. LUD is an important operation in numerical linear algebra because it is widely used for solving linear systems of equations, computing the determinant of a matrix, or as a building block of other operations. Listing 5 shows a 5-loop tiled MM program, where arrays A and C store data in row-major and array B stores data in column-major. In addition, the author considers that all arrays in the LUD store data in column-major in order to evaluate the column access performance of our proposed cache. The author uses MM and LUD as representative examples for 2-D data applications and evaluate the data access performance of our proposed cache in both row and column directions. For 1-D data processing, we only need to switch the dual data access mode to 1-D data access mode. The proposed cache is worked in the same way as a conventional cache and provides the conventional raster line accessibility. Therefore, we only need to evaluate the performance of the RATS-T and RATS-S accesses for 2-D data processing.

8.2.1 Execution environment

The author uses a simulator based on the SimpleScalar-3.0 toolkit [42] [43] [44] [45] for our experiments. For the RATS-T and RATS-S evaluation, the author does not use SIMD instructions because the SimpleScalar simulator does not support SIMD instructions. Instead, the author selects double precision corresponding to tile or line for SIMD parallel processing. In this simulation, the author measures the execution time (cycles), the number of TLB misses and the number and rate of L1 cache (DL1) and unified L2 (UL2) cache misses, for various matrix sizes for N, ranging from 128 to 2048. The author compares the proposed cache memory with the 4-level Z-order tiling layout against two cache memories with the conventional raster layout (row-major order) and the Z-Morton layout. Furthermore, the author compares the proposed RATS cache with the proposed Non-RATS cache to verify the effectiveness of our proposed RATS method. For the RATS cache, if a tile or line miss occurs, the processor loads an aligned tile set from the lower level cache or the main memory. For the Non-RATS cache, only the corresponding tile is loaded from the lower level cache or the main memory on a cache miss to a tile or line read.

The author simulates a common 4-way issue superscalar processor configuration with a memory hierarchy as shown in Table 8.12. The author sets the size of the DL1 and UL2 caches in accordance to the corresponding sizes of the conventional cache used in a high-performance Intel processor. Note that, the latency of the TLB and DL1 cache is 1-cycle. The author uses the 5-loop tiled MM kernels, as shown in Listing 5. The code is a 5-loop tiled MM code with ijk order. The tile size is 32×32. Since MM uses 3 arrays (A, B and C) so that the sum is 3×32×32×8 bytes = 24,576 bytes < 32 Kbytes, the DL1 cache locality is exploited. The tiled LUD program is one of the modified versions used by Wolf and Lam [46]. The author revises the processing order of the LUD program from the row-major order [40] to the column-major order [47] in order to evaluate the column-directional access performance of our proposed cache. The author uses a 64×64 tile size because the LUD uses only 1 array, the sum is 64×64×8 bytes = 32,768 bytes = 32 Kbytes. Thus, the tile size is equal to the size of the DL1 cache. The DL1 cache locality is exploited.

Table 8.12: Processor and memory hierarchy configuration.

CPU	Instr.Fetch Queue	4
	Instr.Issue	4-way out-of-order
	Branch Predictor	Bimod, 2 K entry BTB
Cache	Data L1 Cache	32 Kbytes, 8-way set associative, 64-byte cache line
	Unified L2 cache	256 Kbytes, 8-way set associative, 64-byte cache line
	Latency (cycles)	L1 hit: 1, L2 hit: 6, Memory: 30
TLB	DTLB	64 entry, 4 Kbytes page size, Hit: 1, Miss: 8

Listing 5: The basic tiled MM version (5-loop tiled code).

```
int main()
{
 for(jj = 0; jj < N; jj+=Tile)
  for(kk = 0; kk < N; kk+=Tile)
   for(i = 0; i < N; i++)
    for(j = jj; (j < N && j < jj + Tile); j++)
     for(k = kk; (k < N && k < kk + Tile); k++)
         C[i][j]+= A[i][k] * B[k][j];
}
```

8.2.2 RATS-T access evaluation

8.2.2.1 Performance of multi-level cache and TLB

Figures 8.46-8.53 show the evaluation results for RATS-T access. The X-axis has clusters of bars for each matrix size and the Y-axis plots the number of DL1 and UL2 cache misses. The Y-axis uses logarithmic scale. The label "4-level Z-order" means that the proposed cache does not adopt the RATS method (Non-RATS cache). For LUD, the raster scan order (row-major LUD) uses the row-major based LUD program and the raster scan order (column-major LUD) uses the column-major based LUD program. All three data layouts use the conventional simple tiling code and the author creates the 2-D matrix in C using malloc() to guarantee that all elements of the matrix are contiguously allocated in memory.

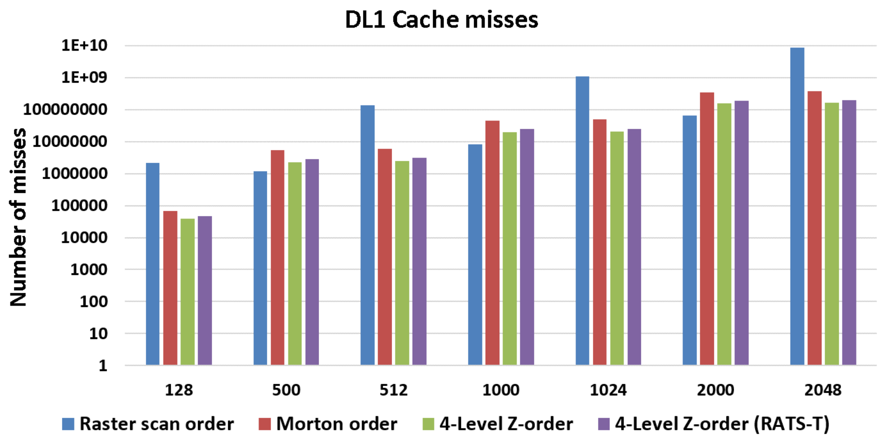


Figure 8.46: RATS-T access, N×N MM: Number of DL1 cache misses.

Table 8.13: DL1 Cache miss rate for MM.

Matrix size	128	500	512	1000	1024	2000	2048
Raster scan order	4.34%	0.04%	4.36%	0.04%	4.36%	0.04%	4.36%
Morton order	0.14%	0.14%	0.14%	0.14%	0.14%	0.14%	0.14%
4-level Z-order	0.08%	0.08%	0.08%	0.08%	0.08%	0.08%	0.08%
4-level Z-order (RATS-T)	0.09%	0.09%	0.09%	0.09%	0.09%	0.09%	0.09%

Figure 8.46 and 8.47 show the number of the DL1 cache misses for MM and LUD.

In Listing 5, the array B[k][j] accesses data in the column direction, that is the loop k scans different rows of array B[k][j]. The conventional raster layout stores data in row-major order, spatial reuse cannot be exploited for B[k][j] along the innermost loop k and conflict misses increase sharply at the power-of-two sized matrix size composed of row over 4 Kbytes page size. Figure 8.46 and 8.47 show the conflict misses increase sharply at power-of-two-sized matrix. For MM, the DL1 cache performance of the Z-Morton order and the 4-level Z-order are little bit inferior to the raster scan order for matrices whose row directional size, that is, column-directional address stride is non-power-of-two. In contrast to the Z-Morton order recursively dividing the address space into multiple 2^n-sized tiles, the 4-level Z-order divides the address space into square tiles of 8×8 byte-sized tile, 64×64 byte-sized medium tile and 2048×2048 byte-sized large tile, respectively. Therefore, the 4-level Z-order reduces the TLB misses in the column-directional contiguous access as well as the Z-Morton order although row directional data access performance may little bit decrease, as compared with the raster scan order. For LUD, the 4-level Z-order provides the less number of DL1 cache misses in all cases, as shown in Figure 8.47. The proposed RATS cache only slightly increases the number of DL1 cache misses because the 4-level Z-order tiling layout can maximize the utilization of 2-D data locality and minimize the conflict misses. Figure 8.47 shows that the number of DL1 cache misses for the RATS-T access is similar to the raster scan order (row-major LUD). However, the number of DL1 cache misses for RATS cache little bit increases because the LUD uses only 1 array and the miss penalty of the RATS-T access is 8 times of the Non-RATS cache.

Table 8.14: DL1 Cache miss rate for LUD.

Matrix size	128	500	512	1000	1024	2000	2048
Raster scan order∗	5.60%	0.25%	6.18%	0.31%	6.20%	0.26%	6.21%
Raster scan order∗∗	0.42%	0.27%	0.45%	0.33%	0.45%	0.30%	0.45%
Morton order	0.20%	0.21%	0.22%	0.22%	0.23%	0.23%	0.23%
4-level Z-order	0.20%	0.21%	0.22%	0.22%	0.23%	0.23%	0.23%
4-level Z-order (RATS-T)	0.36%	0.41%	0.46%	0.46%	0.48%	0.48%	0.48%

∗(column-major LUD), ∗∗(row-major LUD)

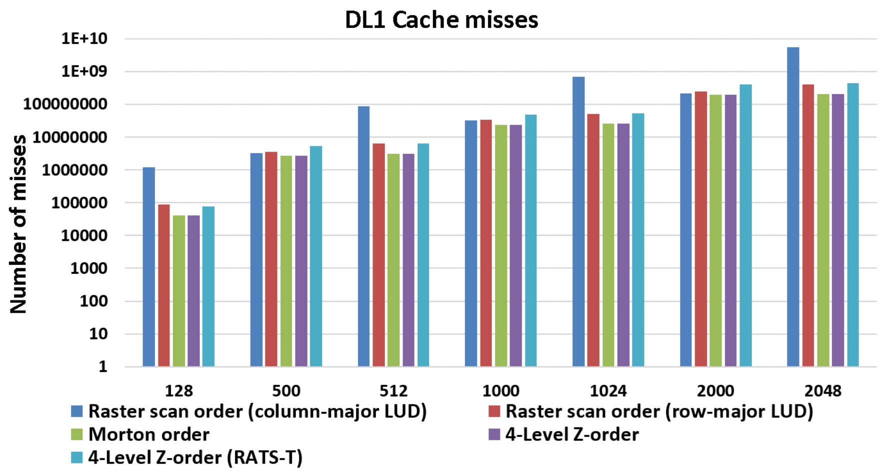


Figure 8.47: RATS-T access, N×N LUD: Number of DL1 cache misses.

Table 8.13 and 8.14 show the DL1 cache miss rate. For MM, the miss rate increase of the RATS cache is at most 0.05% (0.05% = 0.09%-0.04%) to the raster scan order, as shown in Table 8.13. The DL1 cache performance degradation of the 4-level Z-order is superior to the Z-Morton order at non-power-of-two sized matrix. For LUD, the maximum increase of the DL1 cache miss rate is only 0.18% (0.18% = 0.48%-0.30% at matrix size of 2000), as shown in Table 8.14. The overall performance of the MM and LUD can be improved if the SIMD instructions are used together with the proposed cache. Because the proposed cache provides parallel tile accessibility as well as parallel line accessibility and the number of load instructions required for column-directional contiguous access can be reduced to 1/8 at the maximum, as shown in Section 6.2.4. In addition, the matrix transposition required in matrix calculation can be eliminated.

Figure 8.48 and 8.49 show the number of UL2 cache misses of the raster scan order where the 4-level Z-order are almost equal to each other at the non-power-of-two sized matrix for both MM and LUD. In addition, the proposed RATS cache and Non-RATS cache achieve almost the same UL2 cache performance as the Z-Morton layout for both MM and LUD. Figure 8.50 and 8.51 show the number of TLB misses. As expected, the 4-level Z-order and the Z-Morton order achieve the best performance. Since the medium tile size of the 4-level Z-order matches the page size, the TLB misses can be minimized for any matrix size, as shown in Section 3.

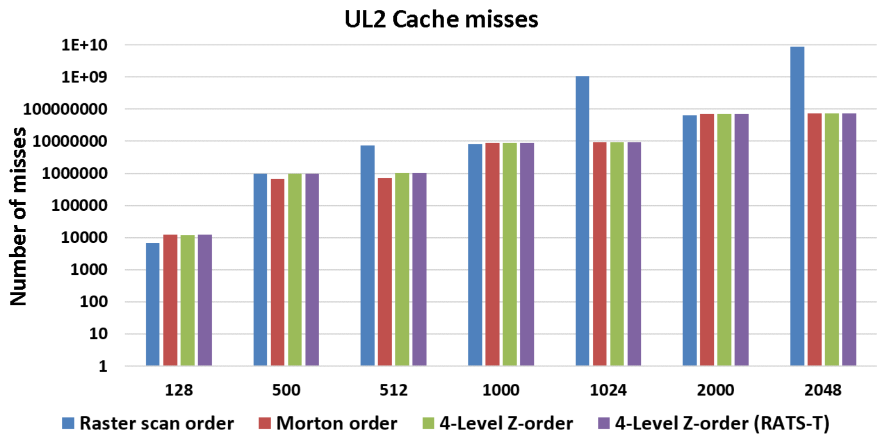


Figure 8.48: RATS-T access, N×N MM: Number of UL2 cache misses.

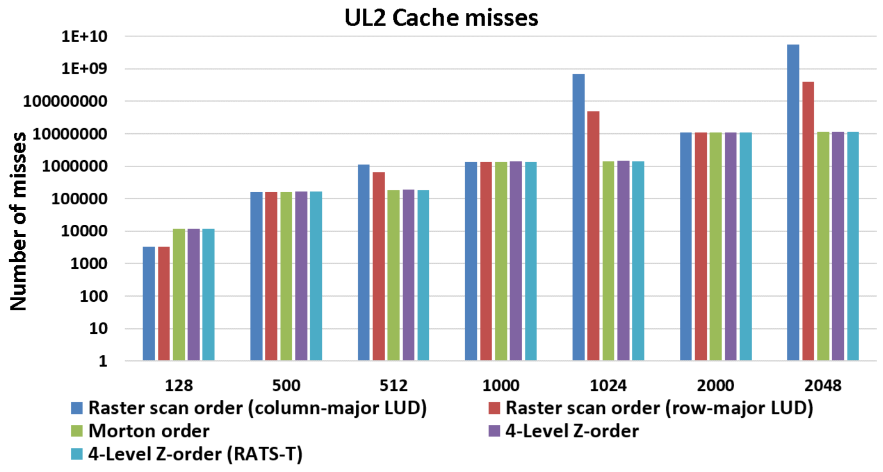


Figure 8.49: RATS-T access, N×N LUD: Number of UL2 cache misses.

8.2.2.2 Execution time evaluation

Figures 8.52-8.53 show the overall speedup of each configuration normalized to the raster scan order configuration for MM and LUD. In all cases, the proposed Non-RATS cache achieves similar performance to that of the Z-Morton layout. For MM and LUD, the evaluation results also show that the RATS cache and the Non-RATS cache provide almost the same matrix computation performance which is not inferior to that of the conventional cache. For LUD, the proposed RATS cache can provide almost the same performance to the column-major based LUD program as that to the

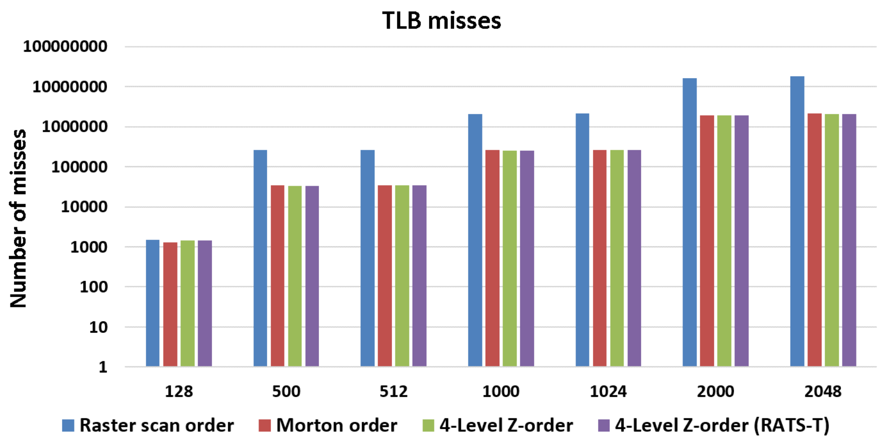


Figure 8.50: RATS-T access, N×N MM: Number of TLB misses.

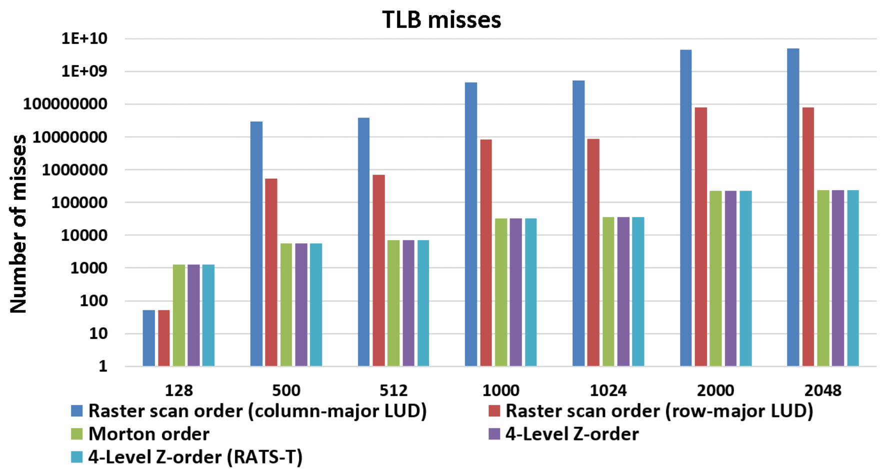


Figure 8.51: RATS-T access, N×N LUD: Number of TLB misses.

row-major based LUD program. This means the 4-level Z-order can correspond to 2-D spatial reference locality with little conflict misses regardless of matrix size. Our proposed cache provides superior access capability in the column direction as well as in the row direction so that it enables us to select either row- or column-major order coding without restriction and it is effective to reduce coding effort.

In addition, at the non-power-of-two sized matrix, the Z-Morton order and the 4-level Z-order achieve the similar execution time performance as compared with the raster scan order although they suffer from little bit higher number of DL1 cache misses, as shown in Figures 8.46-8.47 and Figures 8.52-8.53. This is because their TLB performance are better than the raster scan order at any matrix size, as shown in Figure 8.50 and 8.51. Finally, via the RATS method, the overall performance of the proposed RATS cache is degraded by only 0-1% for MM and by 1-3% for LUD. Therefore, the proposed RATS cache can provide almost the same access performance as the non-RATS cache in spite of the minimal entire hardware increase.

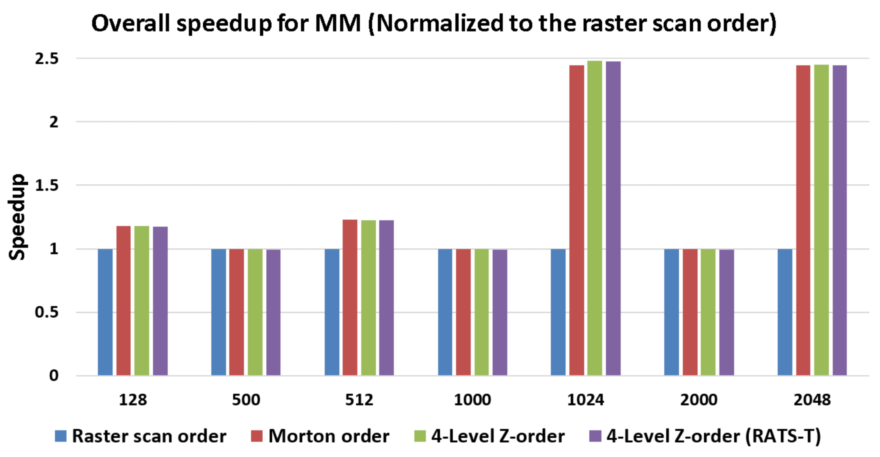


Figure 8.52: RATS-T access, overall speedup for MM.

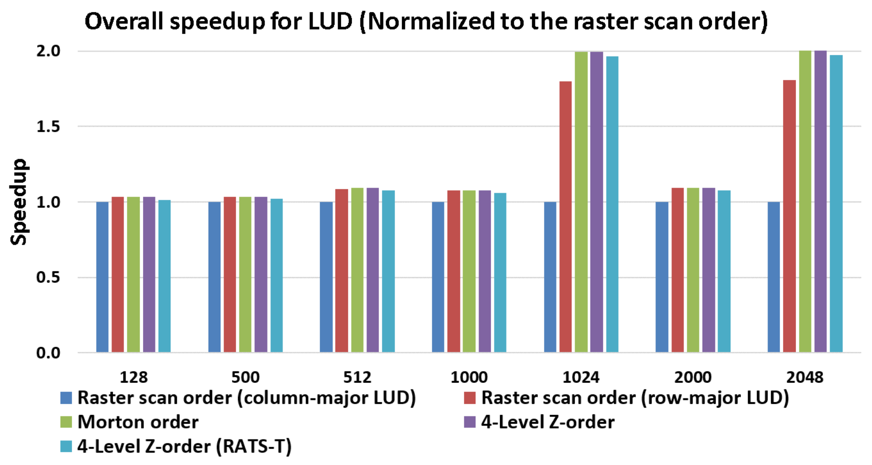


Figure 8.53: RATS-T access, overall speedup for LUD.

8.2.3 RATS-S access evaluation

8.2.3.1 Performance of multi-level cache and TLB

In this section, the author presents the evaluation results for RATS-S access. As shown in Section 5, if the cache line size is 64-byte, the miss penalty of the RATS-S access is eight times that of the conventional raster line access. Here, the author modifies the SimpleScalar simulator and evaluate the execution time, the number of misses for DL1 and UL2 cache memory and TLB.

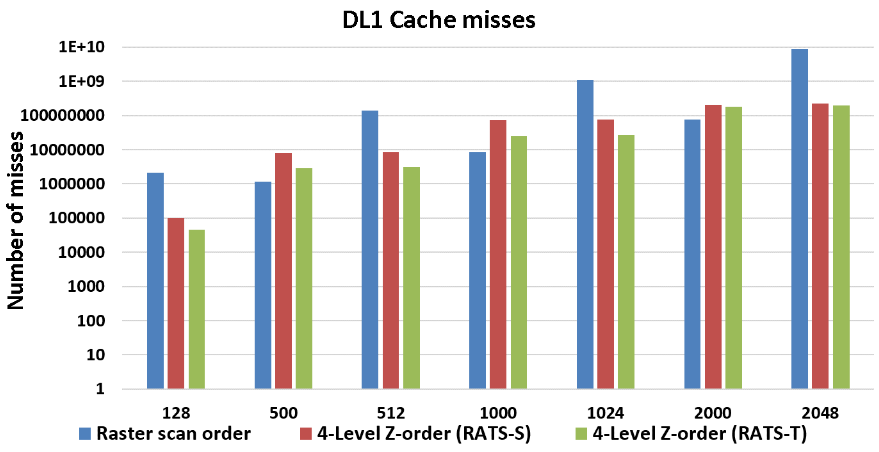


Figure 8.54: RATS-S access, N×N MM: Number of DL1 cache misses.

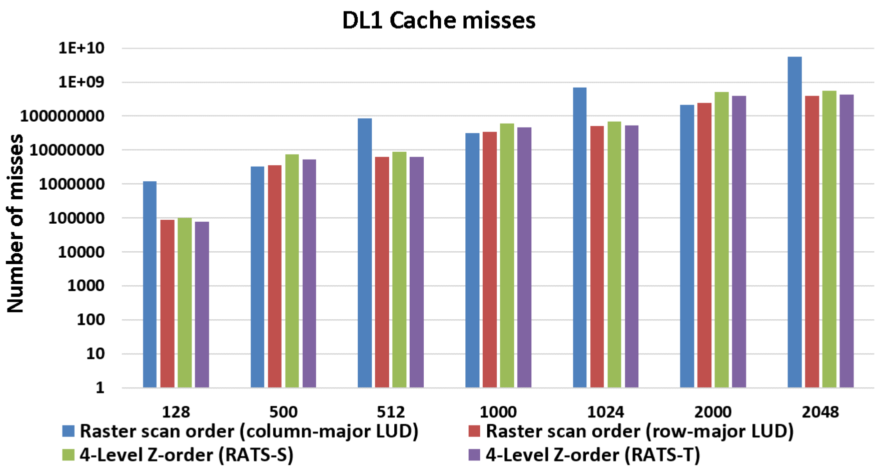


Figure 8.55: RATS-S access, N×N LUD: Number of DL1 cache misses.

Figures 8.54-8.57 show the number of DL1 and UL2 cache misses for MM and LUD. The author compares the proposed RATS-S access with the proposed RATS-T access and the conventional raster line access (tiled). The proposed RATS-S and RATS-T access provides fewer DL1 and UL2 cache misses at the power-of-two matrix size than the conventional raster line access. Compared with RATS-S access, the RATS-T access provides the fewer DL1 cache misses because the sublines of the aligned tile set can be stored in different cache way set, as shown in Figure 8.54 and 8.55. In Figure 8.55, the raster scan order achieves better DL1 cache performance using the row-major based LUD compared with the RATS-S access suffering from eight times miss penalty.

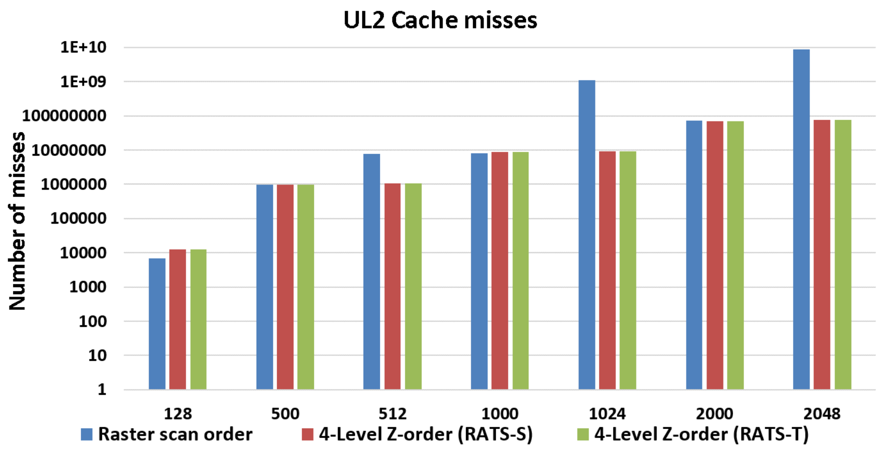


Figure 8.56: RATS-S access, N×N MM: Number of UL2 cache misses.

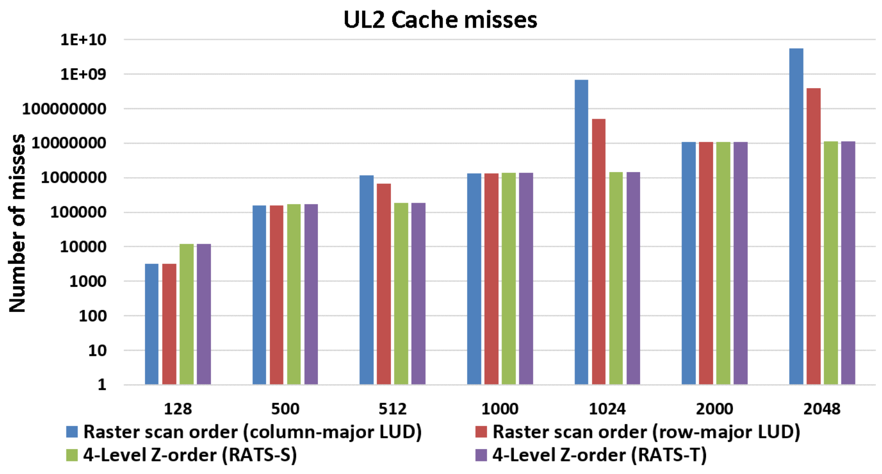


Figure 8.57: RATS-S access, N×N LUD: Number of UL2 cache misses.

As shown in Figure 8.56 and 8.57, the 4-level Z-order and the raster scan order have the almost the same UL2 cache performance at the non-power-of-two matrix size as expected. Figure 8.58 and 8.59 show the number of TLB misses. As expected, the RATS-T and RATS-S access achieve the best performance. Since the medium tile size of the 4-level Z-order matches the page size, the TLB misses can be minimized for any matrix size, as shown in Section 3.

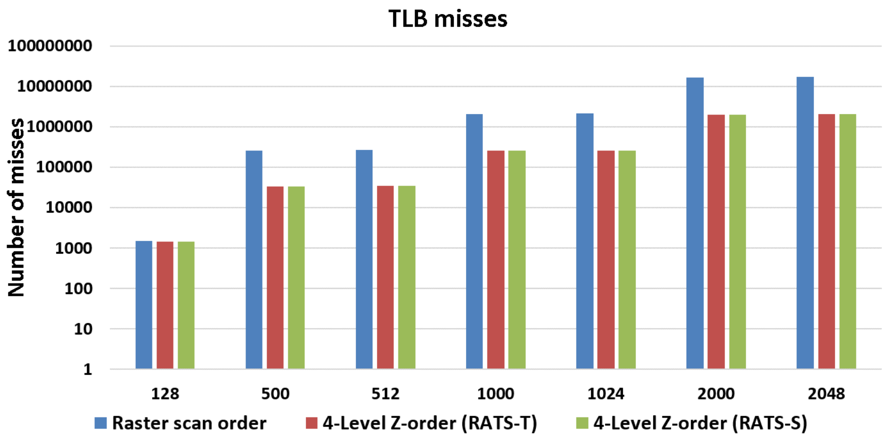


Figure 8.58: RATS-S access, N×N MM: Number of TLB misses.

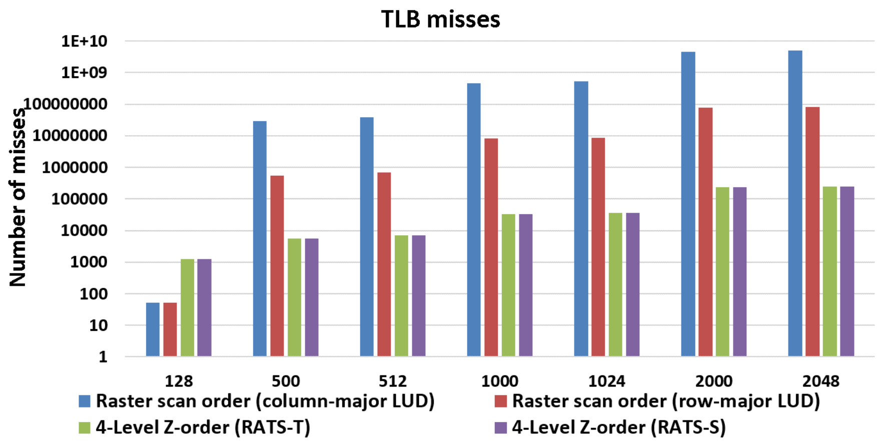


Figure 8.59: RATS-S access, N×N LUD: Number of TLB misses.

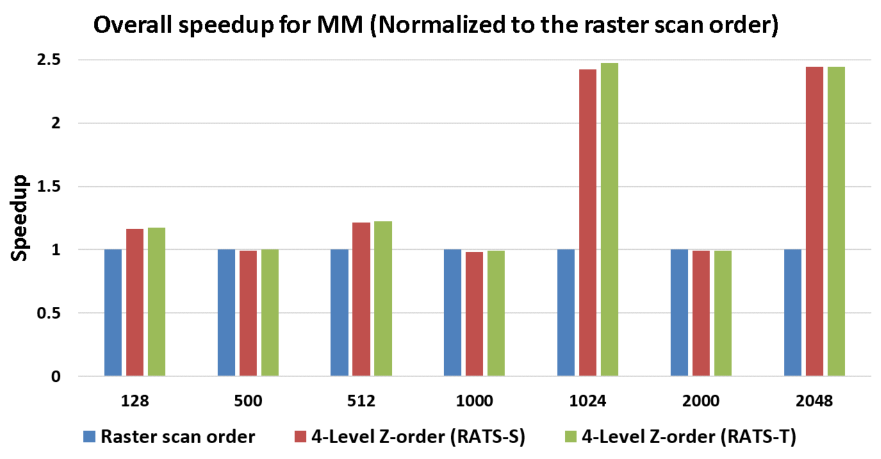


Figure 8.60: RATS-S access, overall speedup for MM.

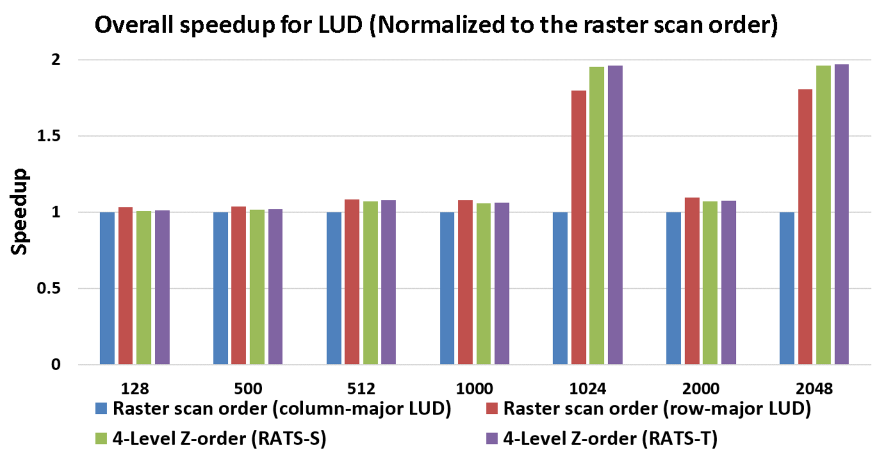


Figure 8.61: RATS-S access, overall speedup for LUD.

8.2.3.2 Execution time evaluation

Figures 8.60-8.61 show the evaluation results of the execution time performance for RATS-S access. The overall speedup of each configuration is normalized to the raster scan order configuration for both MM and LUD. The author also compares the RATS-S access and the RATS-T access performance. The overall performance of the RATS-S access is degraded by only 0-1% for MM and by 0-1% for LUD, as compared with that of the RATS-T access. For the RATS-S access, if a cache miss occurs in the line access, an aligned tile set is loaded from the lower level cache or the main memory and each loaded tile of the aligned tile set is stored to the same cache way set. This cause a little performance degradation. Putting Figures 8.52-8.53 and Figures 8.60-8.61 together shows that the proposed RATS cache is not inferior to the proposed Non-RATS cache about the execution time. Consequently, these results show that the proposed RATS cache can provide almost the same performance as compared with that of the Non-RATS cache although it requires only the minimal additional hardware for both tile and line accessibility.

8.2.4 Parallel tile/line access evaluation

This section describes how the author implements the tile and line access function in the SimpleScalar simulator and evaluate its performance in parallel tile and line access. In addition, because the SimpleScalar does not support SIMD instructions, the author revises the method of calculating execution cycles for load instructions and evaluate the effectiveness of the column-directional parallel access.

The MM algorithm is in Listing 5, where the row- and column-directional access are equal to each other. The author creates a 2-D matrix in C using malloc() to guarantee that all elements of the matrix are contiguously allocated in the memory space. If the read address is in the address range of matrix A, line access mode is selected as the row-directional access to the matrix A access. If the read address is in the address range of matrix B, the tile access mode is selected as the column-directional access to the matrix B access data.

8.2.4.1 Evaluation of reduction to load instructions

Section 6.2.4 explains that the number of load instructions required for parallel tile and line access can be reduced by using SIMD extensions to only one eighth of that required for conventional raster line access in which SIMD extensions are not used. This is because only one load instruction can read an 8×8 byte-sized tile or a 64 byte-sized line from the proposed cache. The author carried out simulations to verify that the proposed cache can efficiently reduce the load instructions for parallel tile and line access.

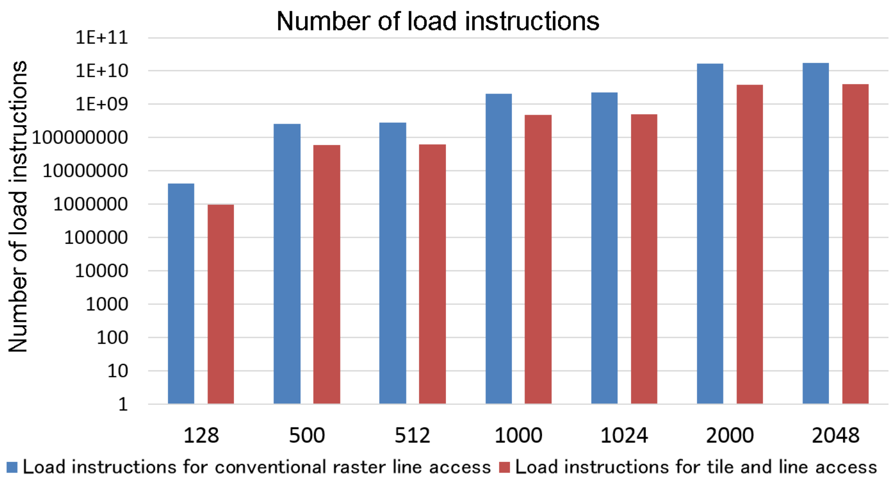


Figure 8.62: Number of load instruction reduction.

Only the frequency of reduced load instructions for tile and line access is counted in the simulation. Parallel load instruction can be used for parallel tile and line access. For conventional raster line access, parallel load instructions can only be used for major-directional data access. Therefore, the number of load instructions for column-directional access is reduced to one eighth of that required for conventional raster line access (non-major-directional data access). The number of load instruction for line access is equal to that required for conventional raster line access (major-directional data access). The results obtained from evaluations are shown in Figure 8.62. In all cases, the number of load instruction required for parallel tile/line access are reduced to about one fourth of that required for conventional raster line access. As a result, the proposed cache with tile and line accessibility can considerably improve the effective transfer rate of the load instruction.

8.2.4.2 Execution time evaluation

This section describes how the author evaluates the performance improvement in execution time for parallel tile and line access by using parallel load instructions. The author calculates the execution time by using the following equation: Execution time = the number of load instructions × load throughput + the number of DL1 cache misses × DL1 cache miss penalty + the number of UL2 cache misses × UL2 cache miss penalty. As shown in Table 8.12, the DL1 cache miss penalty is 6 cycles and the UL2 cache miss penalty is 30 cycles. The author assumes that the load throughput is 1 cycle. The equation to calculate the execution time for conventional load instruction is in Eq.(1) and that to calculate the execution time for parallel load instruction is shown in Eq.(2).

$$\begin{aligned}\text{The execution time for conventional access } &= \text{the number of all load instructions} \times 1\\ &+ \text{the number of DL1 cache misses} \times 6\\ &+ \text{the number of UL2 cache misses} \times 30\end{aligned} \tag{1}$$

$$\begin{aligned}\text{The execution time for parallel access } &= \text{the number of parallel load instructions} \times 1\\ &+ \text{the number of DL1 cache misses} \times 6\\ &+ \text{the number of UL2 cache misses} \times 30\end{aligned} \tag{2}$$

Table 8.15: The number of DL1 and UL2 cache misses for parallel tile and line access.

Matrix size	DL1 cache misses	UL2 cache misses
128	30863	4234
500	2100107	942261
512	2714173	1018387
1000	20657849	7965729
1024	22252463	8359164
2000	153340142	66805002
2048	164824433	71158862

The number of DL1 and UL2 cache misses are shown in Table 8.15. The author evaluates the performance improvement by using the Eq.(3) and parallel load instructions. The results obtained from evaluations are shown in Table 8.16. Although the parallel load instructions for parallel tile/line access are only one fourth of the conventional load instructions, the performance of the proposed cache is also affected by the performance of DL1 and UL2 caches. As a result, the execution time of the parallel load instructions are about one third of that required for conventional load instructions when the matrix size is ≥ 128.

$$\text{Performance improvement} \; = \frac{\text{execution time of parallel access}}{\text{execution time for conventional access}} \tag{3}$$

Table 8.16: Execution time evaluation.

Matrix size	Conventional access	Parallel access	C/P
128	12×10^5	43×10^5	3.4
500	10×10^7	27×10^7	2.7
512	11×10^7	29×10^7	2.6
1000	83×10^7	21×10^8	2.6
1024	89×10^8	23×10^9	2.6
2000	66×10^9	17×10^{10}	2.6
2048	72×10^9	19×10^{10}	2.6

9 Conclusion

Ineffective non-major-directional access to the cache memory has become a bottleneck for efficient 2-D data processing that utilizes extended SIMD instructions. The main achievement and contribution of my work has been to propose a new cache memory with tile/line dual accessibility in improving the performance of non-major-directional cache memory access. The proposed cache provides little TLB miss rate, parallel data access in both row and column directions and low excessive data transfer for efficient 2-D data processing.

The author has shown that parallel tile access that corresponds to the column-directional data access can eliminate the transposition required in matrix calculation, orthogonal transform such as DCT and FFT, 2-D FIR filter and image feature detection. Parallel tile access can also provide efficient 2-D unit block access for image processing and video coding even though its utilization may require significant modifications to the program code.

Furthermore, the author has proposed a 4-level Z-order tiling data layout and a Cache-based hybrid Z-ordering layout to improve 2-D reference locality. The author has shown that the Cache-based hybrid Z-ordering layout can exploit hardware prefetching well and further improve the performance of Strassen algorithm as compared with the conventional raster layout and Z-Morton layout. The author also has proposed a hardware-based address bit-order interchanger to perform address translation. This address bit-order interchange that corresponds to a 64 Kbytes-wide area eliminates the address calculation overhead of Morton-index conversion for 2-D data access and allow the 4-level Z-order tiling layout and the Cache-based hybrid Z-ordering layout to be accessed as a conventional raster layout.

The author has proposed a method of reducing RATS tag memory that considerably reduces the entire hardware scale of the proposed cache and simplifies the proposed cache architecture. After analysis and consideration of the experimental results, the author has proved that the proposed cache achieves both parallel tile and line accessibility by the minimal overhead hardware increase although its access efficiency can be outperformed as compared to those previous studies.

Then, the author has combined the proposed Non-RATS cache with a SIMD-based general purpose-oriented datapath and fully implemented it in a 2.5×5 mm^2 chip area to show the feasibility of the proposed cache. Furthermore, by the proposed RATS tag memory reduction method, the author has shown that the increase rate of the entire hardware scale of the proposed RATS cache is greatly suppressed to only 5% and 7% for an 8-way set associative cache with a 32-byte cache line for the former and a 64-byte cache line for the latter. Under the 3.9 ns clock period, the read latency of the proposed cache is limited to 3 clock cycles, which is the same as that for the conventional cache memory of an Intel or ARM high-performance processor.

Finally, the author has modified the SimpleScalar simulator to evaluate the performance of the proposed cache and provided the following four important conclusions:

- While providing column-directional parallel access capability, the proposed cache succeeds to suppress the conflicts miss increase to the power-of-two sized matrix computation. As a result, it suppresses conflict miss rate increase to only 0.05-0.18% of that of a conventional cache regardless of matrix size and it enables almost one cycle load of 8 double precision data in both row and column directions so that it not only makes matrix transposition unnecessary but also allows effective utilization of 2-D reference locality by SIMD operations.

- The proposed cache reduces the TLB misses in 2-D processing by only using conventional simple tiling code. For LUD, the proposed cache can provide almost the same performance to the column-major based LUD program as that to the row-major based LUD program. In other words, the proposed cache provides column-directional parallel access function and does not restrict our freedom in selecting row-major based or column-major based 2-D program code so that it increases the degree of coding flexibility.

- The performance of RATS-T and RATS-S accesses is only reduced by 0-1% for MM and 1-3% for LUD compared with that for the proposed Non-RATS cache. Therefore, the proposed RATS cache provides a high-performance of the RATS-T and RATS-S access with the minimal hardware increase to the normal DL1 cache structure.

- In MM, the number of parallel load instruction required for parallel tile and line access is reduced to about one fourth of that required for conventional raster line access. Since the performance of the proposed cache is also affected by the performance of DL1 and UL2 caches, the execution time for the parallel load instruction is about one third of that required for conventional load instruction. The proposed cache with tile/line accessibility further improves the performance of 2-D applications by using SIMD instructions.

From the evaluation results, the author finds that the proposed cache provides high-performance with high feasibility for 1-D/2-D data processing. Therefore, it is desirable to incorporate the standard function of the conventional cache into the proposed cache. However, the RATS cache needs a little improvement. If a line miss of the RATS cache occurs, the processor must load an aligned tile set from the lower level cache or the main memory and each tile of an aligned tile set must be stored in the same cache way set. The cache miss rate increases if there is not an optimal tile/line replacement algorithm because the tile data and line data may be stored in the same cache way set so that the line access causes frequent cache misses. Therefore, to further reduce the cache miss rate, it is necessary to develop a new tile/line replacement algorithm that reduce the number of replacement for line access.

In addition, current processors can use gather/scatter operations to perform non-contiguous memory access by providing index vector and its base address. However, conventional gather/scatter operations cause frequent TLB misses due to the raster layout and data in the multi-bank memory may not be read out in parallel by bank conflicts. Therefore, to improve the effectiveness of the proposed cache, it is desirable to also support the gather/scatter function suppressing TLB misses and bank conflict problems.

A Appendix

A.1 Number of cycles required for data loadings

To show that our proposed cache with tile/line accessibility is suitable for efficient video encoding, the author evaluates the performance of the proposed cache by simulating the number of cycles of data loadings in our early work [48]. In the experiment, the proposed cache uses the block-offset mapping method. The measurements are performed with the four different cache organizations. In the JM Reference Software (JM11.0), the author has embedded functional simulator of the proposed cache memory and evaluated its performance in the motion estimation using EPZS method.

Table 1.17: Condition of the simulation.

Item	Contents
Image sequence (ITE Hi-Vision Test Sequence*)	Horse Race (390-420) Bronze_with_credits (260-290)
Image size of input	1920×1056, 30_{fps}
Number of frame	30
Encoder	JM Reference Software
Structure of sequence	M=2 (IBBP...)

Memory hierarchy	Capacity	Latency
L1 cache	1 KB-32 KB	1 cycle
L2 cache	256 KB	10 cycle
Main memory	No limit	200 cycle

* Popular interlace test sequence in Japan.

Table 1.17 shows that the assumed loading latency, data capacity in each memory and an encode condition in JM Reference Software. For simple analysis, direct-mapped cache structure is adopted. Figure. 1.63 shows the evaluation results of the HorseRace sequence. The evaluation results of the Bronze with credits sequence is shown in Figure 1.64. The author compares the proposed cache with the conventional cache, the two-bank interleaved cache [2] and the block-offset mapping cache [4]. The two-bank interleaved cache provides aligned/unaligned cache line accessibility. The block-

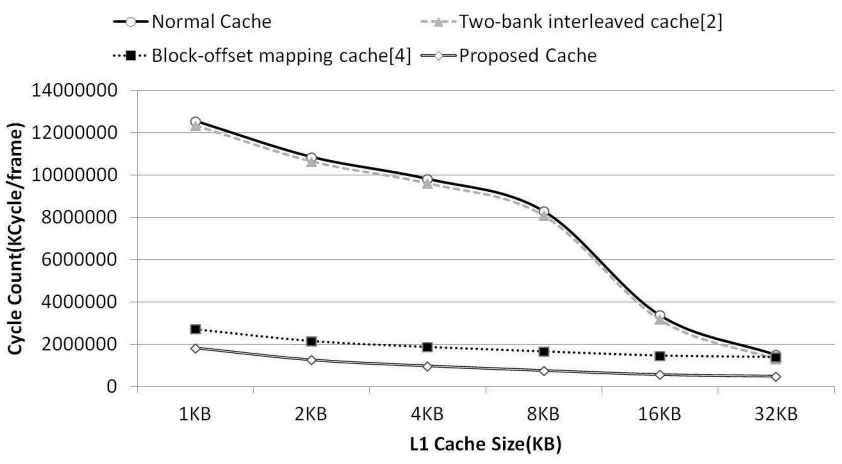


Figure 1.63: Evaluation results in the HorseRace sequence.

offset mapping cache provides tile accessibility. For any cache capacity, the number of loading cycles is reduced as compared with another cache memory. In particular, the performance is significantly improved in the low-capacity cache, but the improvement degree becomes no small as the cache capacity increase. If the cache capacity is 32 KB, the number of loading cycles can reduce about 60 to 80%. In addition, the number of access cycle in the motion estimation is reduced to less than 40 percent of the current required cycles for a conventional cache memory. This means the memory access overhead in the motion estimation is significantly reduced.

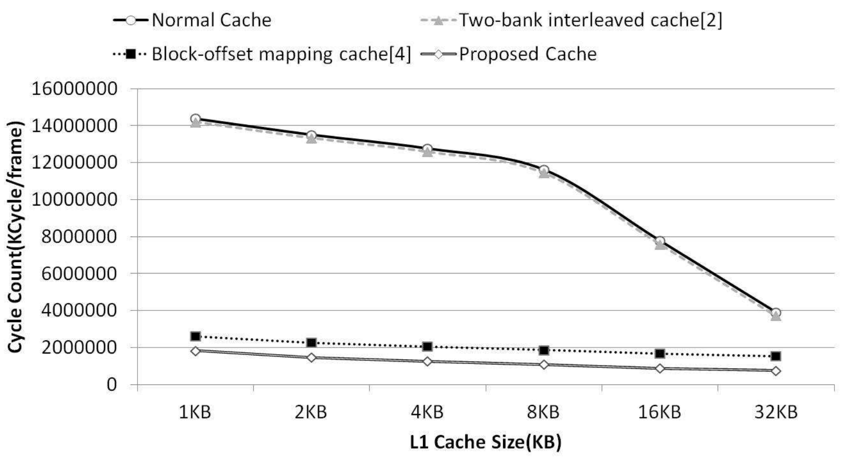


Figure 1.64: Evaluation results in the Bronze with credits sequence.

Acknowledgments

I would like to extend thanks to the many people, who so generously contributed to the work presented in this thesis. Firstly, I would like to express my special appreciation and thanks to my advisor professor Dr.Toshio Kondo, you have been a tremendous mentor for me. It has been an honor to be your last Ph.D student. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. Your advice on both research as well as on my career have been invaluable. I would also like to thank Dr. Takahiro Sasaki and Dr. Yuki Fukazawa, their constructive advice has improved my researches, in particular developing the SimpleScalar simulator and studying the VLSI design flow.

I would also like to thank my committee members, professor Dr.Tetsishi Wakabayashi and professor Dr.Yoshikatsu Ohta for serving as my committee members even at hardship. I also want to thank you for letting my defense be an enjoyable moment, and for your brilliant comments and suggestions, thanks to you.

I especially thank my parents. Words cannot express how grateful I am to my mother and my father. My hard-working parents have sacrificed their lives for myself and provided unconditional love and care. I love them so much, and I would not have made it this far without them. I would also like to thank all of my friends who supported me in writing, and incented me to strive towards my goal. At the end I would like express appreciation to my beloved girlfriend who spent sleepless nights with and was always my support in the moments when there was no one to answer my queries. Without her encouragement and understanding it would have been impossible for me to finish this work.

References

[1] H. Chang and W. Sung. Efficient vectorization of SIMD programs with non-aligned and irregular data access hardware, Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems. New York, USA, 2008, 167-176.

[2] M. Alvarez, E. Salami, A. Ramirez, and M. Valero. Performance Impact of Unaligned Memory Operations in SIMD Extensions for Video Codec Applications, Proceedings of the International Symposium on Performance Analysis of Systems & Software, 2007.

[3] J. H Kim, G. H Hyun, and H. J Lee. Cache organization for H.264/AVC motion compensation, Proceedings of the 13th IEEE International on Embedded Real-Time Computing Systems and Applications, 2007, 534-541.

[4] Yoon. S, Chae. S. I. Cache optimization for H.264/AVC motion compensation, IEICE Transactions on Information and Systems 2008, E91-D(12): 2902-2905, (IEICE).

[5] N. Park, B. Hong and V. K. Prasanna. Tiling, block data layout and memory hierarchy performance, IEEE Transactions on Parallel and Distributed Systems 2003; **14**(7): 640-654.

[6] M. S. Lam, E. E. Rothberg and M. E. Wolf. The cache performance and optimizations of blocked algorithms, ACM SIGARCH Computer Architecture News 1991; **19**(2): 63-74.

[7] N. Park, B. Hong and V. K. Prasanna. Analysis of memory hierarchy performance of block data layout, Proceedings of the International Conference on Parallel Processing, Vancouver, Canada, 2002; 35-44.

[8] S. Chatterjee, A. R. Lebeck, P. K. Patnala and M. Thottethodi. Recursive array layouts and fast parallel matrix multiplication, Proceedings of the 11th Annual ACM symposium on Parallel Algorithms and Architectures, Saint Malo, France, 1999; 222-231.

[9] J. Thiyagalingam, O. Beckmann and P. Kelly. Minimizing associativity conflicts in Morton layout, Proceedings of Parallel Processing and Applied Mathematics, Poznan, Poland, 2006; 1082-1088.

[10] J. Thiyagalingam, O. Beckmann and P. Kelly. Improving the performance of morton layout by array alignment and loop unrolling, Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing, College Station, 2003; 241-257.

[11] C. M. Wittenbrink and A. K. Somani. Cache tiling for high performance morphological image processing, Journal Machine Vision and Applications 1993; **7**(1): 12-22.

[12] E. Athanasaki and N. Koziris. Fast indexing for blocked array layouts to reduce cache misses, International Journal of High Performance Computing and Networking 2005; **3**(5): 417-433.

[13] J. Thiyagalingam, O. Beckmann and P. Kelly. Is morton layout competitive for large two-dimensional arrays yet?, Concurrency and Computation: Practice & Experience 2006; **18**(11): 1509-1539.

[14] E. Athanasaki, N. Koziris. Improving cache locality with blocked array layouts, Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing, Spain, 2004; 308-317.

[15] J. Mellor-Crummey, D. Whalley and K. Kennedy. Improving Memory Hierarchy Performance for Irregular Applications Using Data and Computation Reorderings, International Journal of Parallel Programming 2001; **29**(3): 217-247.

[16] A. Ghane. The effect of reordering multi-dimensional array data on CPU cache utilization, M.S.thesis, Simon Fraser University, 2013, Canada.

[17] W. Lim and M. Thottethodi. Evaluating ISA support and hardware support for recursive data layouts, Proceedings of the 14th International Conference of High Performance Computing, Germany, 2007; 95-106.

[18] S. T. Gabriel and D. S. Wise. The opie compiler from row-major source to morton-ordered matrices, Proceedings of the 3rd Workshop on Memory Performance Issues: in conjunction with the 31st International Symposium on Computer Architecture, New York, USA, 2004; 136-144.

[19] B. K. Wang, Y. Fukazawa, T. Kondo and T. Sasaki. A Cache Memory with Unit Tile and Line Accessibility, Proceedings of the 2016 International Conference on High Performance Computing & Simulation, Innsbruck, Austria, 2016; 866-874.

[20] B. K. Wang, Y. Fukazawa, T. Kondo and T. Sasaki. A Cache Memory with Unit Tile and Line Accessibility, Proceedings of the 2016 IEEE/ACIS 15th International Conference on Computer and Information Science, Okayama, Japan, 2016; 121-126.

[21] X264 FreeH.264 /AVC Encoder [Online], Available: http://www.videolan.org/developers/x264.html.

[22] X86, x64 Instruction Latency, Memory Latency and CPUID dumps. http://instlatx64.atw.hu/, accessed July 04, 2013.

[23] M. Kowarschik and C. Wei. An overview of cache optimizaiton techniques and cache-aware numerical algorithms, In Algorithms for Memory Hierarchies, volume 2625 of LNCS, Springer, 2003; 213-232.

[24] Yuki. T, Renganarayanan. L, Rajopadhye. S, Anderson. C, Eichenberger. A.E, O'Brien. K. Automatic creation of tile size selection models, Proceedings of the 2010 International Symposium on Code Generation and Optimization (CGO), 2010; 190-199.

[25] Intel 64 and IA-32 Architectures Optimization Reference Manual.

[26] M. Ma, J. Hou, J. Ye, M. Arunachalam, R. Gutierrez. Optimizing non-contiguous memory access on intel xeon phi coprocessors, Proceedings of the High Performance Computing and Communications (HPCC), 2015; 1615-1620.

[27] J. J. Dongarra, J. D. Croz, S. Hammarling, I. Duff. A set of level 3 Basic Linear Algebra Subprograms, ACM Transactions on Mathematical Software, 1990; 1-17.

[28] J. J. Dongarra, J. D. Croz, S. Hammarling, R.J. Hanson. An extended set of FORTRAN Basic Linear Algebra Subprograms, ACM Transactions on Mathematical Software, 1988; 1-17.

[29] C. L. Lawson, R. J. Hanson, D. R. Kincaid, F. T. Krogh. Basic Linear Algebra Subprograms for FORTRAN usage, ACM Transactions on Mathematical Software, 1979; 308-323.

[30] D. S. Wise, J. D. Frens, Y. Gu, G. A. Alexander. Language support for Morton-order matrices, Proceedings of the ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming, 2001; Snowbird, UT; 24-33.

[31] N. Reissmann, J, C. Meyer and M. Jahre. A Study of Energy and Locality Effects using Space-filling Curves, Proceedings of the 2014 IEEE International Parallel & Distributed Processing Symposium Workshops, Orlando, Florida USA, 2014; 815-822.

[32] I. Jonsson, Recursive Blocked Algorithms. Data Structures, and High-Performance Software for Solving Linear Systems and Matrix Equations, Ph.D. thesis. UMEA University, 2003, Sweden.

[33] K. P Lorton, D. S. Wise. Analyzing block locality in Morton-order and Morton-hybrid matrices, Proceedings of the 2006 Workshop on Memory Performance: Dealing with Applications, Systems and Architectures (MEDEA 06), New York, USA, 2006, ACM, 512.

[34] P. Gottschling, D. S Wise, A. Joshi. Generic support of algorithmic and structural recursion for scientific computing, International Journal of Parallel, Emergent and Distributed Systems, 24(6), 2009; 479-503.

[35] E. Athanasaki. Non-linear memory layout transformations and data prefetching techniques to exploit locality of references for modern microprocessor architectures

with multilayered memory hierarchies, Ph.D. thesis. National Technical University of Athens, 2006, Greece.

[36] Gaussian Elimination Is Not Optimal, V. Strassen. Numer. Math., 13:354-356, 1969.

[37] F. Desprez and F. Suter. Mixed Parallel Implementation of the Top Level Step of Strassen and Winograd Matrix Multiplication Algorithms, Proceedings of the 15th Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS 2001), San Francisco, 2001.

[38] J. Huang, TM. Smith, GM. Henry and RA. van de Geijn. Strassen's Algorithm Reloaded, Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2016), Salt Lake City, Utah, USA, 2016; 690-701.

[39] D. S. Wise and J. D. Frens. Morton-order matrices deserve compilers' support. Technical Report 533, Computer Science Dept, Indiana University, Nov.1999.

[40] S. Y. Jou, S. J. Chang and T. S. Chang. Fast Motion Estimation Algorithm and Design for Real Time QFHD High Efficiency Video Coding, IEEE Transactions on Circuits and Systems for Video Technology 2015; **25**(9): 1533-1544.

[41] Y. Fukazawa, K. Watanabe, Y. Minoura, T. Kondo and T. Sasaki. SIMD-based Datapath with Efficient Operation Structure, Proceedings of the 2016 IEEE International Conference on Acoustics, Speech and Signal Processing, Shanghai, China 2016; 1031-1035.

[42] Simple scalar homepage: http://www.simplescalar.com.

[43] E. Athanasaki and N. Koziris. Blocked Array Layouts for Multilevel Memory Hierarchies, Proceedings of the 9th Panhellenic Conference in Informatics, Thessaloniki, Greece, 2003; 193-207.

[44] E. Athanasaki and N. Koziris. Fast indexing for blocked array layouts to improve multi-level cache locality, In Interaction between Compilers and Computer Architectures, 2004; 109-119.

[45] J. P. Michael, M. Penner, and V. K. Prasanna. Optimizing graph algorithms for improved cache performance. Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2002), Fort Lauderdale, USA, 2002; 769-782.

[46] M. S. Lam, and M. E. Wolf. A data locality optimizing algorithm, Proceedings of the ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation, Toronto, Canada, 1991; 30-44.

[47] S. Coleman and K. S. McKinley. Tile size selection using cache organization and data layout, Proceedings of the ACM SIGPLAN 1995 Conference on Programming Language Design and Implementation. California, USA, 1995; 279-290.

[48] B. K.Wang, T. Inomata, T. Kondo and T. Sasaki. A Cache Memory with both Line and Tile Unit Accessibility, Proceedings of the 2013 International Workshop on Smart Info-Media Systems in Asia, Nagoya, Japan, 2013; 231-234.