Dual-addressing Memory Architecture for Two-dimensional Memory Access Patterns

Yen-Hao Chen Yi-Yu Liu
Dept. of Computer Science and Engineering
Yuan Ze University, Chungli, Taiwan, R.O.C.
s971433@mail.yzu.edu.tw yyliu@saturn.yzu.edu.tw

Abstract

Cache performance is an important factor in modern computing systems due to large memory access latency. To exploit the principle of spatial locality, a requested data set and its adjacent data sets are often loaded from memory to a cache block simultaneously. However, the definition of adjacent data sets is strongly correlated with the memory organization. Commodity memory is a two-dimensional structure with two (row and column) access phases to locate the requested data set. Therefore, the adjacent data sets are neighbors of the requested data set in a linear order. In this paper, we propose a novel memory organization with dual-addressing modes as well as orthogonal memory access mechanisms. Our dual-addressing memory can be efficiently applied to two-dimensional memory access patterns. Furthermore, we propose a cache coherence protocol to tackle the cache coherence issue due to synonym data set of the dual-addressing memory. For benchmark kernels with two-dimensional memory access patterns, the dual-addressing memory achieves 60% performance improvement as compared to conventional memory. Both cache hit rate and cache utilization are improved after removing two-dimensional memory access patterns from conventional memory.

I. Introduction

With the increasing latency gap between dynamic random access memory (DRAM) and logic, DRAM has become one of the most critical performance bottlenecks in a computing system. Based on principle of temporal and spatial localities, caches are integrated between processor and main memory to provide an illusion of fast and large memory system [1]. Moreover, multiple cache levels are adopted to fulfill both bandwidth and capacity demands in modern processor designs [2]. Furthermore, high associativity is also used to prevent data eviction from cache block competition. As a result, the increased cache level and associativity complicate the controller design and at the same time slow down cache performance.

DRAM is conceptually considered as a one-dimensional array with one serial address for each memory cell. To maintain a feasible aspect ratio for memory chip fabrication, the DRAM organization is composed of a regular two-dimensional memory cell array and two orthogonal address decoding circuits. Consequently, the serial addresses obtained from address decoding circuits define neighborhood structure of a DRAM. Once the neighborhood structure of DRAM is fixed, the mapping of one requested data set and its adjacent data sets from DRAM to one cache block is fixed as well. However, there is no guarantee that the adjacent data sets comply with spatial locality of the requested data set unless the memory access patterns are in sequential address order.

To understand the effectiveness of spatial locality, cache utilization is proposed as an evaluation metric in this work. Furthermore, we propose a novel dual-addressing memory organization to support two-dimensional memory access patterns. Since there are two addresses for each dual-addressing memory cell, synonym data set incurs cache coherence issue. We propose a new cache coherence protocol - *WURF* to resolve cache inconsistency between row-major and column-major caches. We argue that using one conventional memory for versatile data structures and algorithms is not enough in modern computing systems. Specialized memory organizations for specific memory access patterns are imperative for high performance computing.

The rest of this paper is organized as follows. The preliminary of commodity memory organization and data placement layouts for spatial locality optimization is discussed in Section II. Section III presents a novel dual-addressing memory organization for efficient two-dimensional memory access patterns. In Section IV, the cache coherence protocol is proposed to tackle the synonym effect in our dual-addressing memory architecture. The experimental results are summarized in Section V. Section VI concludes this paper and points out important issues for future research.

II. Preliminary

In this section, we give preliminary background of commodity memory organization and review memory data placement layouts for cache spatial locality optimization.

A. Conventional Memory Organization

Commodity DRAM utilizes one transistor and one capacitor for one data bit storage. To maintain the DRAM chip form factor as well as decoding circuit efficiency, memory address is partitioned into row address and column address. Conventionally, the higher address bits are denoted as row address and the lower address bits are denoted as column address. Each memory access requires both row and column access phases.

This work is supported in part by the National Science Council of Taiwan under Grant NSC-100-2221-E-155-052 and NSC-101-2221-E-155-075.

978-3-9815370-0-0/DATE13/©2013 EDAA

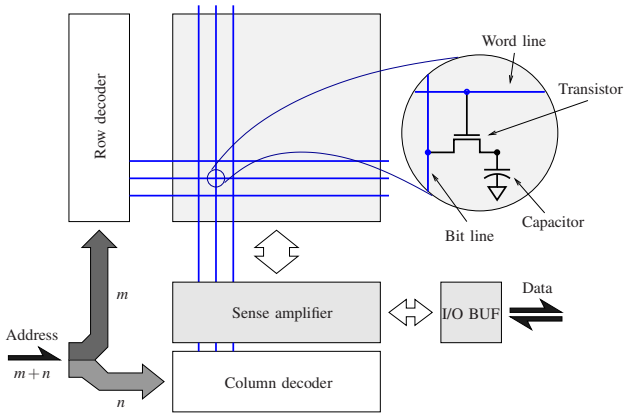

Fig. 1. Conventional memory organization.

Figure 1 draws the organization of a conventional memory. Once the memory address is ready, the row decoder decodes the row address and asserts a corresponding word line. The memory data sets on the word line, refer to a page, are then ready to be processed in the column access phase. According to the column address, sense amplifier and column decoder are activated to detect and select the corresponding memory data set of the page, respectively [3]. In the memory organization drawn in Figure 1, data sets within the same page can be directly accessed by column address decodings only with short access time while data sets from different pages require the assertions of different word lines followed by column address decodings with long access time. Therefore, the sequence of row access followed by column access explicitly defines the horizontal neighborhood structure of a DRAM. The horizontal neighborhood structure favors one-dimensional data array and multi-dimensional data array with row-major memory access patterns.

B. Spatial Locality and Cache Block Utilization

The spatial locality is a phenomenon that once a memory data address is accessed, the adjacent data addresses could be accessed in near future. Hence, a cache with larger block size could accommodate multiple adjacent data sets to exploit the spatial locality. Owing to the aforementioned horizontal neighborhood structure of a commodity DRAM, cache blocks often store consecutive data sets within one memory page. If the memory access patterns of a program is not in a sequential manner, the adjacent data sets stored in a cache block may not be accessed before cache eviction. Therefore, we formally define cache block utilization U_{cb} as follows. In Equation 1, A represents the number of data sets being accessed within a cache block and S represents the total number of data sets within a cache block.

$$U_{cb} = \frac{A}{S} \tag{1}$$

Assume that there are four data sets in each cache block. If each of the four data sets is accessed at least once before the cache block being evicted, the cache block utilization is $\frac{4}{4} = 100\%$. Higher cache block utilization indicates that the neighborhood structure corresponds with the memory access pattern. Hence, a cache system with larger cache block size achieves better hit rate owing to the principle of spatial locality. In contrast, if only one data set is accessed before the cache block being evicted, the cache block utilization is $\frac{1}{4} = 25\%$. It is unnecessary to load adjacent data sets into a cache block since the cache block utilization is low. The average cache block utilization $\bar{U_{cb}}$ is defined in Equation 2, where A_i is the number of data sets being accessed within the i-th cache block transfer from memory and N is the total number of cache block transfers due to cache misses. We will use this equation as the metric to evaluate spatial localities of different memory architectures.

$$\bar{U_{cb}} = \frac{\sum_{i=1}^{N} \frac{A_i}{S}}{N} = \frac{\sum_{i=1}^{N} A_i}{N \cdot S} \tag{2}$$

C. Previous Works

Many different data placement layouts are proposed in previous works for cache hit rate improvement. For applications with column-major memory access patterns, matrix transposition is an effective pre-processing to greatly make use of row-major neighborhood structure in DRAM. However, large matrix transposition results in poor cache performance due to poor temporal and spatial localities [4]. Square block data layout collects planar adjacent data sets for tile-based texture loadings and rendering operations in computer graphics applications [5]. Specialized texture memory combines push and pull memory architectures to balance both bandwidth and capacity requirements [6]. Space-filling curves can be used to specify the adjacent data sets on a two-dimensional plane [7]. Morton data layout is a z-order space-filling curve, which recursively defines adjacent data sets in both row and column directions [8]. The overall two-dimensional memory access performance of Morton data layout is in general consistent in various data sizes as compared to the row-major data layout [9]. However, compiler support is imperative to handle array indices of new Morton data layout without interfering programmers [10]. Park et al. evaluate two-dimensional tiled-based operations with several data placement layouts and conclude that the block data layout achieves best performance among others [11]. Stride pre-fetching is an aggressive approach to predict and pre-load data sets into cache for upcoming memory access [12]. Dahlgren and Stenstrom point out that both spatial locality and stride distance are important factors for pre-fetching [13]. Therefore, a high performance memory system should take both neighborhood structure (spatial locality) and regular memory access patterns (stride distance) into account. These observations motivate us to design a new memory organization for two-dimensional memory access patterns.

III. Dual-addressing Memory

In this section, we propose a novel memory organization to support two-dimensional memory access patterns. Figure 2

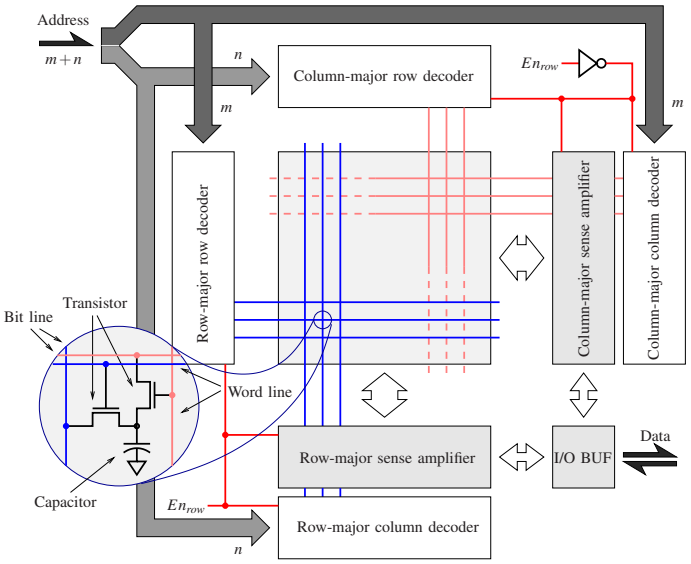

Fig. 2. Dual-addressing memory organization.

depicts the organization of our dual-addressing memory. There are two transistors and one capacitor for one data bit storage. Unlike the dual-ported memory which supports two consecutive memory accesses from the same address decoding architecture [14], the dual-addressing memory utilizes two address decoding architectures. In Figure 2, two word lines and two bit lines are orthogonal to each other, respectively. Hence, there are two pairs of decoding and sensing circuits in dual-addressing memory. The address decoding scheme, which is the same to conventional memory drawn in Figure 1, is denoted as row-major decoding. The address decoding scheme, which is orthogonal to conventional row-major decoding, is denoted as column-major decoding. As drawn in Figure 2, there are $m+n$ address bits with enable signal En_{row} to select one decoding scheme. The row-major decoding decodes the upper m bits and asserts a corresponding horizontal word line before multiplexing vertical bit lines by the lower n bits. The column-major decoding decodes the lower n bits and asserts a corresponding vertical word line before multiplexing horizontal bit lines by the upper m bits. Notice that the upper and lower address bits can be the same ($m=n$) in practice such that the decoder and sense amplifier can be shared by both addressing schemes. Without loss of generality, we assume $m \neq n$ in this paper.

To utilize the dual-addressing memory, programmers are required to specifically declare a dual-addressable (two-dimensional and multi-dimensional) data structure. The data structure will then be bound to the dual-addressing memory after compilation. There are four types of new instructions required to support dual-addressing memory access. These instruction types are row-major read, row-major write, column-major read, and column-major write. The row-major read and write instructions behave as read and write instructions for conventional memory, respectively. The column-major read and write instructions swap the upper and lower address bits

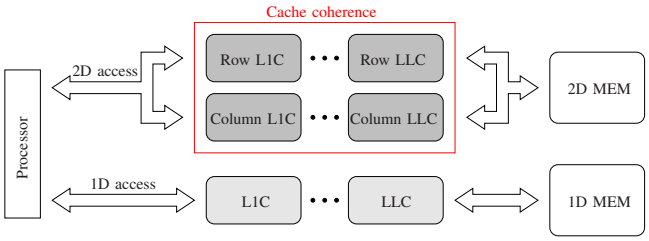

Fig. 3. Proposed memory system architecture.

before performing column-major word line decoding and bit line multiplexing.

Table I lists row-major and column-major decoding results with $m=2$ and $n=3$. The first line in each entry represents the decoded address of row-major decoding (colored in red). The second line in each entry represents the decoded address of column-major decoding (colored in blue). From Table I, it is clear that each memory data set has row-major and column-major addresses for row-major and column-major memory accesses, respectively. Consequently, row-major memory access favors horizontal neighborhood structure while column-major memory access favors vertical neighborhood structure. Therefore, our dual-addressing memory is capable of maintaining spatial locality in two-dimensional memory access patterns.

IV. Cache Coherence for Synonym Addressings

Conventionally, cache coherence issue occurs in a system with multiple processors. However, cache coherence is an important dual-addressing memory system design issue even in a single-core processor system. Since two decoded (row-major and column-major) addresses are associated with one dual-addressing memory data set, this is so-called synonym addressings. The synonym addressing results in two copies of cache data from the same memory data set. Hence, cache coherence is required between the two caches. Figure 3 depicts a memory system hierarchy with both conventional memory and dual-addressing memory denoted as "1D MEM" and "2D MEM", respectively. Cache coherence must be maintained between row-major and column-major caches in Figure 3. Notice that there is no coherence issue between conventional cache and dual-addressing cache since the memory data is disjoined after compilation.

We use a dual-addressing memory with $m=2$ and $n=3$ (Table I) as an example to illustrate the cache coherence issue. Figure 4 lists the statuses of row-major and column-major caches. Assume that each cache block accommodates four data sets. The row-major cache contains two blocks with memory addresses {12, 13, 14, 15} and {20, 21, 22, 23}. The column-major cache contains one block with memory address {24, 25, 26, 27}. According to Table I, there are two synonym data sets {14, 25} and {22, 26}. The two synonym data sets bring out the cache coherence issue. For example, if there is a row-major write on address 14, a cache hit occurs in row-major cache. Nevertheless, the cache block with column-major address 25 must be simultaneously updated to maintain

TABLE I

ROW-MAJOR AND COLUMN-MAJOR DECODINGS

	000	001	010	011	100	101	110	111
00	0 (00000) 0 (00000)	1 (00001) 4 (00100)	2 (00010) 8 (01000)	3 (00011) 12 (01100)	4 (00100) 16 (10000)	5 (00101) 20 (10100)	6 (00110) 24 (11000)	7 (00111) 28 (11100)
01	8 (01000) 1 (00001)	9 (01001) 5 (00101)	10 (01010) 9 (01001)	11 (01011) 13 (01101)	12 (01100) 17 (10001)	13 (01101) 21 (10101)	14 (01110) 25 (11001)	15 (01111) 29 (11101)
10	16 (10000) 2 (00010)	17 (10001) 6 (00110)	18 (10010) 10 (01010)	19 (10011) 14 (01110)	20 (10100) 18 (10010)	21 (10101) 22 (10110)	22 (10110) 26 (11010)	23 (10111) 30 (11110)
11	24 (11000) 3 (00011)	25 (11001) 7 (00111)	26 (11010) 11 (01011)	27 (11011) 15 (01111)	28 (11100) 19 (10011)	29 (11101) 23 (10111)	30 (11110) 27 (11011)	31 (11111) 31 (11111)

Valid	Dirty	Cache data address			
1	0	12	13	14	15
1	0	20	21	22	23

(a) Row-major cache.

Valid	Dirty	Cache data address			
1	0	24	25	26	27
0	-	-	-	-	-

(b) Column-major cache.

Fig. 4. Cache coherence example.

TABLE II

SYSTEM SPECIFICATIONS

Simics target machine	
CPU	Ultrasparc IV+
CPU clock rate	2GHz
L1 cache	2-way, write back, write allocate, LRU
L1 cache hit latency	3 cycles
L2 cache	2-way, write back, write allocate, LRU
L2 cache hit latency	23 cycles
Memory clock rate	667MHz
Operating system	SuSE7.3

DRAMsim2	
DRAM type	DDR3
Banks / rows / columns	8 / 8192 / 1024
Clock period	1.5 *ns*
Page policy	Open page

cache coherence. Similarly, if there is a column-major read on address 18, a cache miss occurs in column-major cache. However, the synonym data can be forwarded from row-major cache with address 20.

To remedy the cache coherence issue, a cache protocol is required for dual-addressing memory. MESI (modified, exclusive, shared, and invalid) is one of the most widely used cache coherence protocol implemented on many processor systems [15]. Applying the MESI protocol ensures cache invalidation before a write operation on shared data in both row-major and column-major caches. However, the cache invalidation overhead is pretty high owing to the orthogonal neighborhood structures in two different caches. For example in Figure 4, if there is a write operation on column-major cache {24, 25, 26, 27}, the row-major cache blocks {12, 13, 14, 15} and {20, 21, 22, 23} are inevitably invalidated. Therefore, the MESI protocol on a dual-addressing cache system is relatively costly as compared to that on a conventional cache system with multiple processors.

In this paper, in stead of using classical MESI protocol, we propose a new cache protocol for dual-addressing memory. Besides of common read and write operations, our cache coherence protocol has two distinct operations - update and forward. Accordingly, there are four major operations and states in our cache protocol: *write (W)*, *update (U)*, *read (R)*, and *forward (F)*, denoted as *WURF*. Figure 5 draws the state transitions of our *WURF* protocol. The superscript and subscript indicate the cache level and row-major/column-major access, respectively. For *write* operation, the data is written into present cache on a cache hit while *write* operation is performed in next level cache on a cache miss. For *update* operation, there is no need to update next level cache on a cache hit since our cache write policy is write-back while *update* operation proceeds with next level cache to ensure cache coherence on a cache miss. For *read* operation, present cache returns requested data on a cache hit while *forward* and *read* operations are invoked in its orthogonal cache and next level cache, respectively, on a cache miss. For *forward* operation, the synonym data can be forwarded to shorten cache miss latency on a cache hit while the present cache returns to ready state on a cache miss. Therefore, a *write* request from processor on one addressing is simultaneously accompanied with an *update* request on its orthogonal addressing throughout all cache levels to maintain the cache coherence.

V. EXPERIMENTAL RESULTS

We evaluate our proposed memory system on a full system simulator Simics [16]. The dual-addressing memory is constructed based on DRAMsim2 [17]. The overall system specifications are listed in Table II. The detailed cache parameters of three evaluated memory data placement layouts are summarized in Table III. Notice that the total cache size are set to the same size for a fair comparison. Nine benchmark kernels with two-dimensional memory access patterns are outlined in Table IV. All simulations are based on a 100M-instruction warm up phase followed by a 10M-instruction sampling phase.

The overall system performance are summarized in Table V. Columns *Baseline*, *Morton*, and *DA* represent commodity memory as the baseline, Morton data layout, and dual-addressing memory, respectively. Columns *IPC* and *Imp* represent instructions per cycle and performance improvement ratio, respectively. In average, our dual-addressing memory

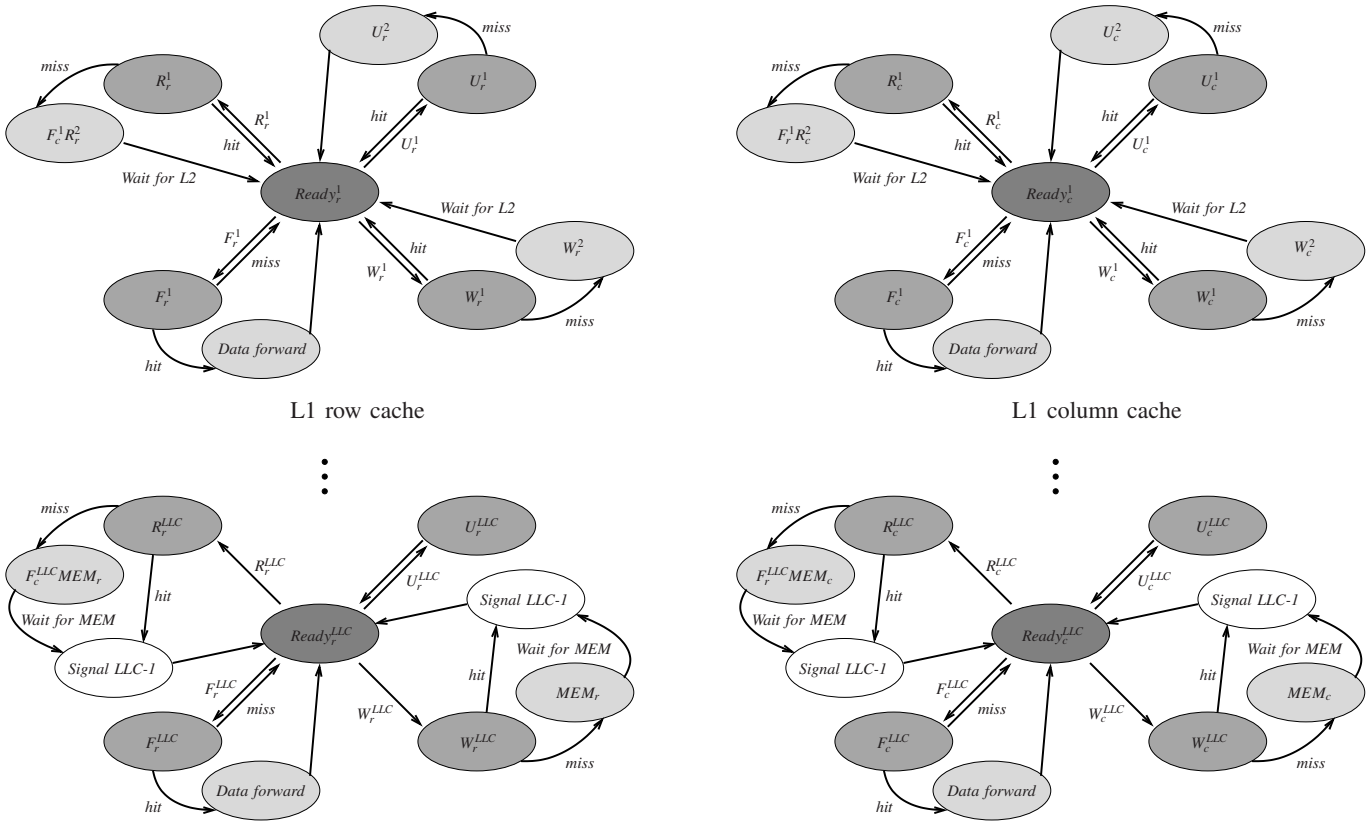

Fig. 5. State transitions of *WURF* protocol.

TABLE III

CACHE PARAMETERS

		Baseline	*Morton*	*Dual-addressing*
L1 cache	Conventional	16KB, 64B cache block	16KB, 64B cache block	8KB, 64B cache block
	Row-major	-	-	4KB, 64B cache block
	Column-major	-	-	4KB, 64B cache block
L2 cache	Conventional	64KB, 256B cache block	64KB, 256B cache block	32KB, 256B cache block
	Row-major	-	-	16KB, 256B cache block
	Column-major	-	-	16KB, 256B cache block

TABLE IV

BENCHMARK KERNELS

M-Trans	Matrix transposition
M-Mult	Matrix multiplication
LU-Decomp	LU decomposition
MCO	Matrix multiplication chain order
O-BST	Optimal binary search tree
RL-CHKY	Right-looking cholesky factorization
LL-CHKY	Left-looking cholesky factorization
HWI-Comp	Haar wavelet image compression
HWI-Decomp	Haar wavelet image decompression

TABLE V

COMPARISONS ON INSTRUCTION PER CYCLE (IPC)

	Baseline	*Morton*		*DA*	
Program	*IPC*	*IPC*	*Imp* (%)	*IPC*	*Imp* (%)
M-Trans	0.248	0.221	89.1	0.398	160.3
M-Mult	0.330	0.316	95.9	0.487	147.5
LU-Decomp	0.312	0.312	99.9	0.503	160.9
MCO	0.428	0.432	100.9	0.547	127.7
O-BST	0.307	0.309	100.5	0.527	171.6
RL-CHKY	0.304	0.298	98.0	0.489	161.0
LL-CHKY	0.432	0.482	111.5	0.505	116.9
HWI-Comp	0.209	0.233	111.4	0.455	217.2
HWI-Decomp	0.260	0.279	107.7	0.461	177.6
average			101.7		160.1

achieves 60% and 57% performance improvement over the baseline and Morton, respectively.

Table VI compares the cache hit rates. Subscripts $1D$ and $2D$ represent the result of conventional cache and average result of row-major and column-major cache, respectively. By removing the two-dimensional data from commodity memory, the $1D$ cache hit rate is improved even the cache size is reduced by half. The $2D$ cache hit rate is high since the dual-addressing memory supports both row-major and column-major neighborhood structures.

Finally, we compare the cache utilizations defined in Section II. From Table VII, it is clear that a memory system with high hit rate is not guaranteed to have high cache utilization,

TABLE VI

COMPARISONS ON CACHE HIT RATE

	L1 cache				*L2 cache*			
Program	*Baseline*$_{1D}$ (%)	*Morton* (%)	*DA*$_{1D}$ (%)	*DA*$_{2D}$ (%)	*Baseline*$_{1D}$ (%)	*Morton* (%)	*DA*$_{1D}$ (%)	*DA*$_{2D}$ (%)
M-Trans	61.9	50.9	78.5	93.8	27.9	60.5	76.3	83.3
M-Mult	83.5	74.6	95.7	93.7	19.7	47.7	73.9	87.2
LU-Decomp	73.5	64.6	74.9	96.9	24.1	50.2	95.6	87.2
MCO	85.9	82.0	95.4	95.8	25.8	50.5	81.7	73.9
O-BST	34.1	53.8	87.1	95.8	68.8	47.6	92.1	74.5
RL-CHKY	73.5	69.5	75.3	96.9	26.2	56.2	97.4	91.5
LL-CHKY	93.5	95.4	77.4	97.9	71.2	61.8	73.6	56.4
HWI-Comp	82.0	81.7	95.5	95.1	56.4	59.9	79.9	88.7
HWI-Decomp	92.2	84.9	95.6	93.7	57.4	59.7	77.6	85.9
average	75.6	73.0	86.1	95.5	41.9	54.9	83.1	80.9

TABLE VII

COMPARISONS ON CACHE UTILIZATION

	L1 cache				*L2 cache*			
Program	*Baseline*$_{1D}$ (%)	*Morton* (%)	*DA*$_{1D}$ (%)	*DA*$_{2D}$ (%)	*Baseline*$_{1D}$ (%)	*Morton* (%)	*DA*$_{1D}$ (%)	*DA*$_{2D}$ (%)
M-Trans	15.5	12.4	28.3	100.0	30.1	49.2	85.1	100.0
M-Mult	14.3	9.3	24.5	99.8	28.9	47.6	93.7	100.0
LU-Decomp	14.2	10.6	24.5	99.3	26.5	49.4	69.6	97.4
MCO	17.8	13.9	38.3	98.6	28.0	49.2	87.7	94.2
O-BST	9.4	8.8	24.4	99.3	26.9	44.9	71.8	97.7
RL-CHKY	13.2	12.3	24.6	99.2	26.2	49.0	60.3	95.2
LL-CHKY	30.2	30.8	22.9	90.5	33.0	56.1	83.8	61.9
HWI-Comp	12.9	12.6	24.9	79.8	28.6	47.1	86.6	90.8
HWI-Decomp	21.9	11.9	31.2	59.3	37.2	48.0	76.1	53.7
average	16.6	13.6	27.1	91.7	29.5	49.0	79.4	87.9

and vice versa. A cache system with high cache utilization indicates the system stores less unused data in its cache. Our overall 2*D* cache utilizations are relatively high since the dual-addressing memory complies with spatial locality of two-dimensional memory access patterns.

VI. CONCLUSION AND FUTURE WORK

We have proposed a novel dual-addressing memory to support two-dimensional memory access patterns. With two orthogonal neighborhood structures, dual-addressing memory provides row-major and column-major memory accesses. Since there are two addressing modes in dual-addressing memory, synonym data sets in row-major and column-major caches result in cache inconsistency issue. To maintain the cache coherence among dual-addressing caches, we propose a new cache protocol - *WURF*. The experiments on two-dimensional access patterns demonstrate that the overall performance of dual-addressing memory is 60% and 57% higher than those of commodity memory and Morton data layout, respectively. By removing the two-dimensional data from commodity memory, both cache hit rate and cache utilization are greatly improved. We argue that using specialized memory organizations for specific memory access patterns are imperative for high performance computing. Currently, we are designing a virtual memory system for two-dimensional address translation from user space to physical space with variable data sizes.

REFERENCES

[1] D. A. Patterson, J. L. Hennessy, "Computer organization and design: the hardware/software interface", Morgan Kaufmann, 2011.

[2] G. Sun, C. J. Hughes, C. Kim, J. Zhao, C. Xu, Y. Xie, Y. K. Chen, "Moguls: a model to explore the memory hierarchy for bandwidth improvements", *in Proceedings of International Symposium on Computer Architecture*, pp.377-388, 2011.

[3] B. Jacob, S. Ng, D. Wang, "Memory systems: cache, dram, disk", Morgan Kaufmann, 2007.

[4] S. Chatterjee, S. Sen, "Cache-efficient matrix transposition", *in Proceedings of High Performance Computer Architecture*, pp.195-205, 2000.

[5] M. Pharr, R. Fernando, T. Sweeney, "GPU Gems 2: Programming techniques for high-performance graphics and general-purpose computation", Addison-Wesley, 2005.

[6] M. Cox, N. Bhandari, M. Shantz, "Multi-level texture caching for 3D graphics hardware", *in Proceedings of International Symposium on Computer Architecture*, pp.86-97, 1998.

[7] H. Sagan, "Space-filling curves", Springer-Verlag, 1994.

[8] S. Chatterjee, V. V. Jain, A. R. Lebeck, S. Mundhra, M. Thottethodi, "Nonlinear array layouts for hierarchical memory systems", *in Proceedings of International Conference on Supercomputing*, pp.444-453, 1999.

[9] J. Thiyagalingam, O. Beckmann, P. H. J. Kelly, "An exhaustive evaluation of row-major, column-major and Morton layouts for large two-dimensional arrays", *in Proceedings of UK Performance Engineering Workshop*, pp.1-12, 2003.

[10] D. S. Wise, J. D. Frens, Y. Gu, G. A. Alexander, "Language support for Morton-order matrices", *in Proceedings of ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*, pp.24-33, 2001.

[11] N. Park, B. Hong, V. K. Prasanna, "Tiling, block data layout, and memory hierarchy performance", *in IEEE Transaction on Parallel and Distributed Systems*, pp.640-654, vol.14, i.7, July 2003.

[12] J. W. C. Wu, J. H. Patel, "Data prefetching strategies for vector cache memories", *in Proceedings of International Symposium on Computer Architecture*, pp.54-63, 1991.

[13] F. Dahlgren, P. Stenstrom, "Evaluation of hardware-based stride and sequential prefetching in shared-memory multiprocessors", *in IEEE Transaction on Parallel and Distributed Systems*, pp.385-398, vol.7, i.4, April 1996.

[14] N. Weste, D. Harris, "CMOS VLSI design: a circuits and systems perspective", Addison Wesley, 2010.

[15] M. S. Papamarcos, J. H. Patel, "A low-overhead coherence solution for multiprocessors with private cache memories", *in Proceedings of International Symposium on Computer Architecture*, pp.348-354, 1984.

[16] Wind River Simics, *http://www.windriver.com/products/simics/*.

[17] P. Rosenfeld, E. Cooper-Balis, B. Jacob, "DRAMSim2: A cycle accurate memory system simulator", *in IEEE Computer Architecture Letters*, pp.16-19, vol.10, i.1, November 2011.