Memory Hierarchy Performance of Tiling and Block Data Layout

Neungsoo Park, Bo Hong and Viktor K. Prasanna
Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2562
neungsoo, bohong, prasanna@halcyon.usc.edu

Technical Report No. **CENG 02-15**

Department of Electrical Engineering – Systems
University of Southern California
Los Angeles, California 90089-2562
213-740-4465

Memory Hierarchy Performance of Tiling and Block Data Layout *

Neungsoo Park, Bo Hong and Viktor K. Prasanna
Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2562
{neungsoo, bohong,prasanna}@halcyon.usc.edu
http://advisor.usc.edu

Abstract

Recently, several experimental studies have been conducted on block data layout in conjunction with tiling as a data transformation technique to improve cache performance. In this paper, we analyze cache and TLB performance of such alternate layouts (including block data layout and Morton layout) when used in conjunction with tiling. We derive a tight lower bound on TLB performance for standard matrix access patterns, and show that block data layout and Morton layout achieve this bound. To improve cache performance, block data layout is used in concert with tiling. Based on the cache and TLB performance analysis, we propose a data block size selection algorithm that finds a tight range for optimal block size. To validate our analysis, we conducted simulations and experiments using tiled matrix multiplication, LU decomposition and Cholesky factorization. For matrix multiplication, simulation results using UltraSparc II parameters show that tiling and block data layout, with a block size given by our block size selection algorithm, reduces up to 93% of TLB misses compared with other techniques (copying, padding, etc.). The total miss cost is reduced considerably. Experiments on several platforms (UltraSparc II and III, Alpha, and Pentium III) show that tiling with block data layout achieves up to 50% performance improvement over other techniques that use conventional layouts. Morton layout is also analyzed and compared with block data layout. Experimental results show that matrix multiplication using block data layout is up to 15% faster than that using Morton data layout.

Index Terms: block data layout, tiling, TLB misses, cache misses, memory hierarchy

*Supported by the DARPA Data Intensive Systems Program under contract F33615-99-1-1483 monitored by Wright Patterson Air force Base and the National Science Foundation under grant No. 99000613 and in part by an equipment grant from Intel Corporation.

1 Introduction

The increasing gap between memory latency and processor speed is a critical bottleneck in achieving high performance. The gap is typically bridged by a multi-level memory hierarchy that can hide memory latency. This memory hierarchy consists of multi-level caches, which are typically on- and off- chip caches. To improve the effective memory hierarchy performance, various hardware solutions have been proposed [3, 7, 9, 10, 19]. Recent processors such as Intel Merced [24] provide increased programmer control over data placement and movement in a cache-based memory hierarchy, in addition to providing some memory streaming hardware support for media applications. To exploit these features, it is important to understand the effectiveness of control and data transformations.

Along with hardware solutions, compiler optimization techniques have received considerable attention [14, 15, 22]. As the memory hierarchy gets deeper, it is critical to efficiently manage the data. To improve data access performance, one of the well-known optimization techniques is tiling. Tiling transforms loop nests so that temporal locality can be better exploited for a given cache size. However, tiling focuses only on the reduction of capacity cache misses by decreasing the working set size. Cache in most state-of-the-art machines is either direct-mapped or small set-associative. Thus, it suffers from considerable conflict misses, thereby degrading the overall performance [6, 12]. To reduce conflict misses, copying [12, 25] and padding [16, 20] techniques with tiling have been proposed. However, most of these approaches target mainly the cache performance, paying less attention to the Translation Look-aside Buffer (TLB) performance. As problem sizes become larger, TLB performance becomes more significant. If TLB thrashing occurs, the overall performance will be drastically degraded [23]. Hence, both TLB and cache must be considered in optimizing application performance.

Most previous optimizations, including tiling, concentrate on single-level cache [6, 12,

16, 20, 25]. Multi-level memory hierarchy has been considered by a few researchers. For improving multi-level memory hierarchy performance, a new compiler technique is proposed in [27] that transforms loop nests into recursive form. However, only multi-level caches were considered [21, 27] with no emphasis on TLB. It was proposed in [13] that cache and TLB performance be considered *in concert* to select the tile size. In this analysis, TLB and cache were assumed to be fully-set associative. However, the cache is direct or small set-associative in most of the state-of-the-art platforms.

Some recent work [4, 11, 12, 17, 18, 25] proposed changing the data layout to match the data access pattern, to reduce cache misses. It was proposed in [11] that both data and loop transformation be applied to loop nests for optimizing cache locality. In [4], conventional (row or column-major) layout is changed to a recursive data layout, referred to as Morton layout, which matches the access pattern of recursive algorithms. This data layout was shown to improve the memory hierarchy performance. This was confirmed through experiments; we are not aware of any formal analysis.

The ATLAS project [26] automatically tunes several linear algebra implementations. It uses block data layout with tiling to exploit temporal and spacial locality. Input data, originally in column major layout, is re-mapped into block data layout before the computation begins. The combination of block data layout and tiling has shown high performance on various platforms. However, the selection of the optimal block size is done *empirically* at compile time by running several tests with different block sizes.

In this paper, we study *block data layout* as a data transformation to improve memory hierarchy performance. In block data layout, a matrix is partitioned into sub-matrices called blocks. Data elements within one such block are mapped onto contiguous memory. These blocks are arranged in row-major order. First, we analyze the intrinsic TLB performance of block data layout. We then analyze the TLB and cache performance using tiling and block data layout. Based on the analysis, we propose a block size selection algorithm. *Morton data*

layout is also discussed as a variant of block data layout. The contributions of this paper are as follows:

- We present a lower bound analysis of TLB performance. Further, we show that block data layout intrinsically has better TLB performance than row-major layout (Section 2). As an abstraction of matrix operations, the cost of accessing all rows and all columns is analyzed. Compared with row major layout, we show that the number of TLB misses is improved by $O(\sqrt{P_v})$ where P_v is the page size.

- We present TLB and cache performance analysis when tiling is used with block data (Section 3.1 and 3.2). In tiled matrix multiplication, block data layout improves the number of TLB misses by a factor of B, where B is the block size. Cache performance analysis is also presented. We validate our analysis through simulations using SimpleScalar [2].

- On the basis of our cache and TLB analysis, we propose a block size selection algorithm (Section 3.3). The best block sizes found by ATLAS fall in the range given by our algorithm.

- We validate our analysis through simulations and measurements using matrix multiply, LU decomposition and Cholesky factorization (Section 4).

- We compare the performance of block data layout and Morton data layout. Block size selection for Morton data layout is limited. This limitation causes the performance of Morton data layout to be worse than that of block data layout. Experimental results on UltraSparc II and Pentium III show that matrix multiplication and LU decomposition executions using block data layout were up to 15.8% faster than that obtained using Morton data layout.

The rest of this paper is organized as follows. Section 2 describes block data layout and gives analysis of its TLB performance. Section 3 discusses the TLB and cache performance when tiling and block data layout are used in concert. A block size selection algorithm is described based on this analysis. Section 4 shows simulation and experimental results. Concluding remarks are presented in Section 5.

2 Block Data Layout and TLB Performance

In this paper, we assume the architecture parameters to be fixed (e.g. cache size, cache line size, page size, TLB entry capacity, etc.). The following notations are used in this paper. S_{tlb} denotes the number of TLB entries. P_v denotes virtual page size. It is assumed that the TLB is fully set-associative with Least-Recently-Used(LRU) replacement policy. Block size is $B \times B$, where it is assumed $B^2 = kP_v$. S_{ci} is the size of the ith level cache. Its line size is denoted as L_{ci}. Cache is assumed to be direct-mapped.

In Section 2, we analyze the TLB performance of block data layout. We show that block data layout has better intrinsic TLB performance than conventional data layouts.

2.1 Block Data Layout

To support multi-dimensional array representations, most programming languages provide a mapping function, which converts an array index to a linear memory address. In current programming languages, the default data layout is *row-major* or *column-major*, denoted as canonical layouts [5]. Both row-major and column-major layouts have similar drawbacks. For example, consider a large matrix stored in row-major layout. Due to large stride, column accesses can cause cache conflicts. Further, if every row in a matrix is larger than the size of a page, column accesses can cause TLB trashing, resulting in drastic performance degradation.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

(a) Row-major layout

0	1	4	5	8	9	12	13
2	3	6	7	10	11	14	15
16	17	20	21	24	25	28	29
18	19	22	23	26	27	30	31
32	33	36	37	40	41	44	45
34	35	38	39	42	43	46	47
48	49	52	53	56	57	60	61
50	51	54	55	58	59	62	63

(b) Block data layout

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	47	58	59	62	63

(c) Morton data layout

Figure 1: Various data layouts: block size 2×2 for (b) and (c)

In block data layout, a large matrix is partitioned into sub-matrices. Each sub-matrix is a $B \times B$ matrix and all elements in the sub-matrix are mapped onto contiguous memory locations. The blocks are arranged in row-major order. Another data layout of recent interest is Morton data layout [4]. Morton data layout divides the original matrix into four quadrants and lays out these sub-matrices contiguously in the memory. Each of these sub-matrices is further recursively divided and laid out in the same way. At the end of recursion, elements of the sub-matrix are stored contiguously. This is similar to the arrangement of elements of a block in block data layout. Morton data layout can thus be considered as a variant of the block data layout. They only differ in the order of blocks. Figure 1 shows block data layout and Morton data layout with block size 2×2. Due to the similarity, the following TLB analysis holds true for Morton data layout also.

2.2 TLB Performance of Block Data Layout

In this section, we present a lower bound on the TLB misses for any data layout. We discuss the intrinsic TLB performance of block data layout using a generic access pattern. We give an analysis on the TLB performance of block data layout and show improved performance compared with conventional layouts. Throughout this section, we consider an $N \times N$ array.

2.2.1 A Lower Bound on TLB Misses

In general, most matrix operations consist of row and column accesses, or permutations of row and column accesses. In this section, we consider an access pattern where an array is accessed first along *all* rows *and* then along *all* columns. The lower bound analysis of TLB misses incurred in accessing the data array along all the rows and all the columns is as follows.

Theorem 2.1 *For accessing an array along all the rows and then along all the columns, the asymptotic minimum number of TLB misses is given by* $2\frac{N^2}{\sqrt{P_v}}$.

Proof: Consider an arbitrary mapping of array elements to pages. Let $A_k = \{i|$ at least one element of row i is in page k $\}$. Similarly, let $B_k = \{j|$ at least one element of column j is in page k $\}$. Let $a_k = |A_k|$ and $b_k = |B_k|$. Note that $a_k \times b_k \geq P_v$. Using the mathematical identity that the arithmetic mean is greater than or equal to the geometric mean ($a_k + b_k \geq 2\sqrt{P_v}$), we have:

$$\sum_{k=1}^{\frac{N^2}{P_v}} (a_k + b_k) \geq 2\frac{N^2}{P_v}\sqrt{P_v}.$$

Let x_i (y_j) denote the number of pages where elements in row i (column j) are scattered. The number of TLB misses in accessing all rows consecutively and then all columns consecutively is given by $T_{miss} \geq \sum_{i=1}^{N}(x_i - O(S_{tlb})) + \sum_{j=1}^{N}(y_j - O(S_{tlb}))$. $O(S_{tlb})$ is the number of page entries required for accessing row i (column j) that are already present in the TLB. Page k is accessed a_k times by row accesses, thus, $\sum_{i=1}^{N} x_i = \sum_{k=1}^{\frac{N^2}{P_v}} a_k$. Similarly, $\sum_{j=1}^{N} y_j = \sum_{k=1}^{\frac{N^2}{P_v}} b_k$. Therefore, the total number of TLB misses is given by

$$T_{miss} \geq \sum_{k=1}^{\frac{N^2}{P_v}} (a_k + b_k) - 2N \cdot O(S_{tlb}) \geq 2 \times \frac{N^2}{\sqrt{P_v}} - 2N \cdot O(S_{tlb}).$$

As the problem size (N) increases, the number of pages accessed along one row (column) becomes larger than the size of TLB (S_{tlb}). Thus the number of TLB entries that are reused is

reduced between two consecutive row (column) accesses. Therefore the asymptotic minimum number of TLB misses is given by $2\frac{N^2}{\sqrt{P_v}}$. $\odot$

We obtained a lower bound on TLB misses for any layout when data are accessed along all rows and then along all columns. This lower bound of TLB misses also holds when data is accessed along an arbitrary permutation of all rows and columns.

Corollary 2.1 *For accessing an array along an arbitrary permutation of row and column accesses, the asymptotic minimum number of TLB misses is given by* $2\frac{N^2}{\sqrt{P_v}}$.

2.2.2 TLB Performance

In this section, we consider the same access pattern as discussed in Section 2.2.1. Consider a given $N \times N$ array stored in a canonical layout. Without loss of generality, canonical layout is assumed to be row-major layout. During the first pass (row accesses), the memory pages are accessed consecutively. Therefore, TLB misses caused by row accesses is equal to $\frac{N^2}{P_v}$. During the second pass (column accesses), elements along the column are assigned to N different pages. Hence, a column access causes N TLB misses. Since $N \gg S_{tlb}$, all N column accesses result in N^2 TLB misses. The total number of TLB misses caused by all row accesses and all column accesses is thus $\frac{N^2}{P_v} + N^2$. Therefore, in canonical layout, TLB misses drastically increase due to column accesses.

Compared with canonical layout, block data layout has better TLB performance. The following theorem shows that block data layout minimizes the number of TLB misses.

Theorem 2.2 *For accessing an array along all the rows and then along all the columns, block data layout with block size* $\sqrt{P_v} \times \sqrt{P_v}$ *minimizes the number of TLB misses.*

To analyze the number of TLB misses for this data access pattern, we consider two cases: $B^2/P_v \geq 1$ and $B^2/P_v \leq 1$. For each case, we estimate the number of TLB misses by

comparing the TLB size, the number of page entries in a row, and the number of pages in a block. The optimal block size is then derived from these estimations. Detailed proof of the above theorem is presented in Appendix A. In general, the number of TLB misses for a $B \times B$ block data layout is $k\frac{N^2}{B} + \frac{N^2}{B}$. It is reduced by a factor of $\frac{(P_v+1)B}{P_v(k+1)}(\approx \frac{B}{k+1})$ when compared with canonical layout. When $B = \sqrt{P_v}$ $(k = 1)$, this number approaches the lower bound shown in Theorem 2.1.

The above theorem holds true even when data in block data layout is accessed along an arbitrary permutation of all rows and columns.

Corollary 2.2 *For accessing an array along an arbitrary permutation of rows and columns, block data layout with block size* $\sqrt{P_v} \times \sqrt{P_v}$ *minimizes the number of TLB misses.*

Similar to Theorem 2.2 and Corollary 2.2, the number of TLB misses is minimized when blocks are stored in Morton data layout and elements are accessed along rows and columns.

Corollary 2.3 *For accessing an* $N \times N$ *array along along all the rows and then along all the columns (or along an arbitrary permutation of rows and columns), Morton data layout with block size* $\sqrt{P_v} \times \sqrt{P_v}$ *minimizes the number of TLB misses.*

To verify our analysis, simulations were performed using the SimpleScalar simulator [2]. It is assumed that the page size is $8KByte$ and the data TLB is fully set-associative with 64 entries (similar to the data TLB in UltraSparc 2.) Double precision data points are assumed. The block size is set to 32. Table 1 shows the comparison of TLB misses using block data layout with using canonical layout. Table 1 (a) shows the TLB misses for the "first all rows and then all columns" access. For small problem sizes, TLB misses with block data layout are considerably less than those with canonical layout. This is due to the fact that TLB entries used in a column(row) access are almost fully reused in the next column(row)access. For a problem size of 1024×1024, a 504.37 times improvement in the number of TLB misses

Table 1: Comparison of TLB misses

(a) Along all rows and then all columns

Layout	1024	2048	4096
Block Layout	2081	81794	1196033
Morton Layout	2072	274473	1081466
Canonical Layout	1049601	4198401	16793601

(b) Arbitrary permutation of row and column accesses

Layout	1024	2048	4096
Block Layout	64140	273482	1080986
Morton Layout	64257	273477	1080955
Canonical Layout	1053606	4208690	16822675

(c) Arbitrary permutation of all rows followed by arbitrary permutation of all columns accesses

Layout	1024	2048	4096
Block Layout	64501	274473	1080465
Morton Layout	64813	274472	1081469
Canonical Layout	1053713	4208681	16822395

is obtained with block data layout. This number is much less than the lower bound obtained from Theorem 2.1. This is because the TLB entries are reused for this problem size. For larger problem sizes the TLB entries cannot be reused. The total number of TLB misses approaches the lower bound. For these large problem sizes, TLB misses with block data layout are upto 16 times less compared with canonical layout.

To verify Corollary 2.1 and 2.2, two sets of access patterns were simulated: an arbitrary permutation of all rows and columns, and an arbitrary permutation of all rows followed by an arbitrary permutation of all columns. With these access patterns, TLB entries referenced during one row(column) access are not reused when accessing the next row(column). The number of TLB misses with block data layout approaches the lower bound on TLB misses. The results are shown in Table 1 (b) and (c). Morton data layout shows a performance similar to block data layout.

Even though block data layout has better TLB performance compared with canonical layouts with generic access pattern, it alone does not reduce cache misses. The data access pattern of tiling matches well with block data layout. In the following section, we discuss the

```
for kk=0 to N by B
    for jj=0 to N by B
        for i=0 to N
            for k=kk to min(kk+B-1,N)
                r = X(i,k)
                for j=jj to min(jj+B-1,N)
                    Z(i,j) += r*Y(k,j)
```

(a) 5-loop tiled matrix multiplication

```
for jj=0 to N by B
    for kk=0 to N by B
        for ii=0 to N by B
            for i=ii to min(ii+B-1,N)
                for k=kk to min(kk+B-1,N)
                    r = X(i,k)
                    for j=jj to min(jj+B-1,N)
                        Z(i,j) += r*Y(k,j)
```

(b) 6-loop tiled matrix multiplication

Figure 2: Tiled matrix multiplication

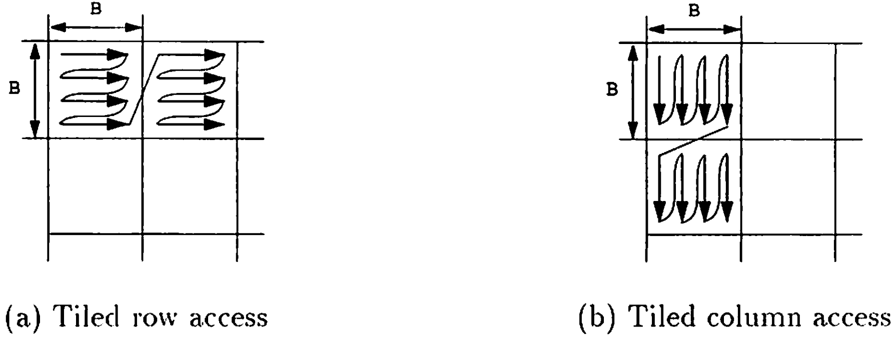
performance improvement of TLB and caches when block data layout is used in conjunction with tiling.

3 Tiling and Block Data Layout

Tiling is a well-known optimization technique that improves cache performance. Tiling transforms the loop nest so that temporal locality can be better exploited for a given cache size. Consider an $N \times N$ matrix multiplication represented as $\mathbf{Z} = \mathbf{XY}$. The working set size for the usual 3-loop computation is $N^2 + 2N$. For large problems, the working set size is larger than the cache size, resulting in severe cache thrashing. To reduce cache capacity misses, tiling transforms the matrix multiplication to a 5-loop nest tiled matrix multiplication (TMM) as shown in Figure 2(a). The working set size for this tiled computation is $B^2 + 2B$. To efficiently utilize block data layout, we consider a 6-loop TMM as shown in Figure 2(b) instead of a 5-loop TMM.

3.1 TLB Performance

In this section, we show the TLB performance improvement of block data layout with tiling. To illustrate the effect of block data layout on tiling, we consider a generic access pattern abstracted from tiled matrix operations. The access pattern is shown in Figure 3 where the



(a) Tiled row access

(b) Tiled column access

Figure 3: Tiled accesses

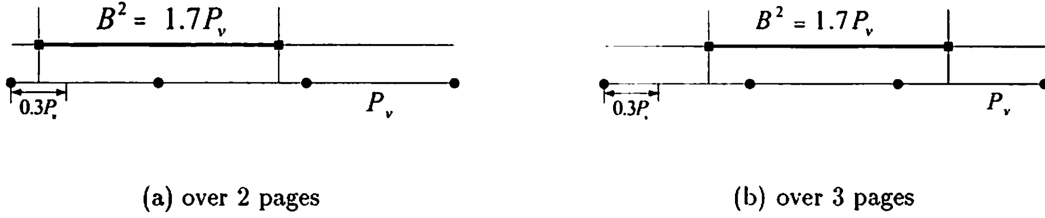
tile size is equal to B.

With canonical layout, TLB misses will not occur when accessing consecutive tiles in the same row, if $B \leq S_{tlb}$. Hence, the tiled accesses along the rows generate $\frac{N^2}{P_v}$ TLB misses. This is the minimum number of TLB misses incurred in accessing all the elements in a matrix. However, tiled accesses along columns cause considerable TLB misses. B page table entries are necessary for accessing each tile. For all tiled column accesses, the total number of TLB misses is $T_{col} = B \times \frac{N}{B} \times \frac{N}{B} = \frac{N^2}{B}$. It is reduced by a factor of B compared with the number of TLB misses for all column accesses without tiling (see Section 2.2).

The total number of TLB misses are further reduced when block data layout is used in concert with tiling, as shown in Theorem 3.1. Throughout this paper, the block size of block data layout is assumed to be the same as the tile size so that the tiled access pattern matches block data layout. In block data layout, a 2-dimensional block is mapped onto 1-dimensional contiguous memory locations. A block extends over several pages, as shown in Figure 4 for an example of block size $B^2 = 1.7P_v$. To analyze TLB misses for column accesses using block data layout, the average number of pages in a block is required.

Lemma 3.1 *Consider an array stored in block data layout with block size $B \times B$, where $B^2 = kP_v$. The average number of pages per block is given by $k + 1$.*

Proof: For block size kP_v, assume that $k = n + f$, where n is a non-negative integer and



(a) over 2 pages

(b) over 3 pages

Figure 4: Blocks extending over page boundaries

$0 \le f < 1$. The probability that a block extends over $n+1$ contiguous pages is $1-f$. The probability that a block extends over $n+2$ contiguous pages is f. Therefore, the average number of pages per block in block data layout is given by: $(1-f)\times(n+1)+f\times(n+2) = k+1$. ⊙

Theorem 3.1 *Assume that an $N \times N$ array is stored using block data layout. For tiled row and column accesses, the total number of TLB misses is* $(2+\frac{1}{k})\frac{N^2}{P_v}$.

Proof: Blocks in block data layout are arranged in row-major order. So, a page overlaps between two consecutive blocks that are in the same row. The page is continuously accessed. The number of TLB misses caused by all tiled row accesses is thus $\frac{N^2}{P_v}$, which is the minimum number of TLB misses. However, no page overlaps between two consecutive blocks in the same column. Therefore, each block along the same column goes through $(k+1)$ different pages according to Lemma 3.1. The number of TLB misses caused by all tiled column accesses is thus $T_{col} = (k+1) \times \frac{N}{B} \times \frac{N}{B} = (k+1)\frac{N^2}{kP_v}$. Therefore, the total TLB misses caused by all row and all column accesses is $T_{miss} = (2+\frac{1}{k})\frac{N^2}{P_v}$. ⊙

For tiled access, the number of TLB misses using canonical layout is $\frac{N^2}{P_v} + \frac{N^2}{B}$, where $B = \sqrt{kP_v}$. Using Theorem 3.1, compared with canonical layout, block data layout reduces the number of TLB misses by $\frac{\sqrt{kP_v}+\sqrt{k}}{2k+1} = \frac{B+\sqrt{k}}{2k+1}$.

To verify our analysis, simulations for tiled row and column accesses were performed using the SimpleScalar simulator. The simulation parameters are equal to those in Section 2.

Table 2: TLB misses for all tiled row accesses followed by all tiled column accesses

Layout	1024	2048	4096
Block Layout	2081	12289	49153
Canonical Layout	33794	139265	561025

A 32×32 block size was considered. The block size is the same as the page size. Table 2 shows TLB misses for 3 different cases. For problem sizes of 2048×2048 and 4096×4096, the number of TLB misses conform our analysis in Theorem 3.1. The number of TLB misses with block data layout is 91% less than that with canonical layout. For a problem size of 1024×1024, TLB misses with block data layout is 2081, which is very close to the minimum number of TLB misses (2048). This is a special case in which each block starts on a new page.

A similar analytical result can be derived for real applications. Consider the 5-loop TMM with canonical layout in Figure 2 (a). Array **Y** is accessed in a tiled row pattern. On the other hand, arrays **X** and **Z** are accessed in a tiled column pattern. A tile of each array is used in the inner loops (i, k, j). The number of TLB misses for each array is equal to the average number of pages per tile, multiplied by the number of tiles accessed in the outer loops (kk, jj). The average number of pages per tile is $B + \frac{B^2}{P_v}$. Therefore, the total number of TLB misses is given by: $2N^3(\frac{1}{B^2} + \frac{1}{BP_v}) + N^2(\frac{1}{B} + \frac{1}{P_v})$.

Consider the 6-loop TMM on block data layout as shown in Figure 2 (b). A $B \times B$ tile of each array is accessed in the inner loops (i, k, j) with block layout. The number of TLB misses for each array is equal to the average number of pages per block multiplied by the number of blocks accessed in the outer loops (ii, kk, jj). According to Lemma 3.1, the average number of pages per block is $\frac{B^2}{P_v} + 1 (= k + 1)$. Therefore, the total number of TLB

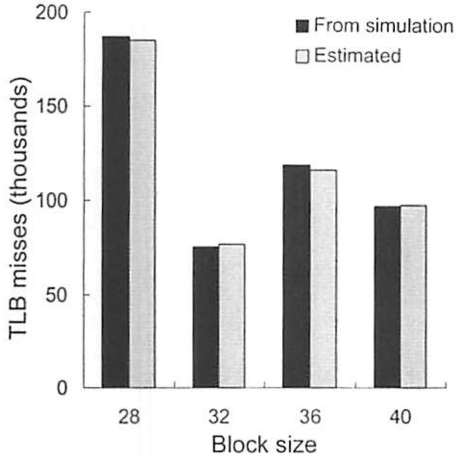


Figure 5: Comparison of TLB misses from simulation and estimation

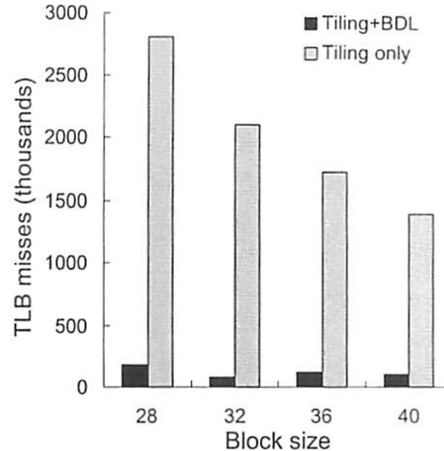


Figure 6: Comparison of TLB misses using tiling+BDL and tiling only

misses (TM) is

$$TM = \left(\frac{B^2}{P_v}+1\right)\left\{2\left(\frac{N}{B}\right)^3+\left(\frac{N}{B}\right)^2\right\} = 2N^3\left(\frac{1}{BP_v}+\frac{1}{B^3}\right)+N^2\left(\frac{1}{P_v}+\frac{1}{B^2}\right). \quad (1)$$

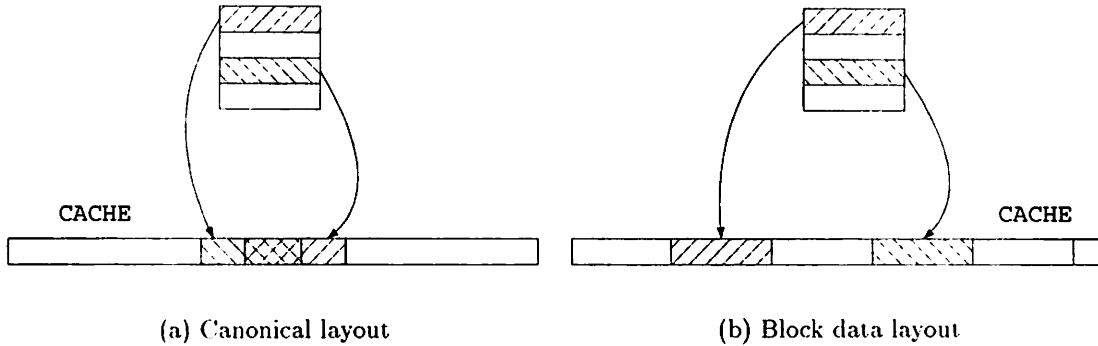
Compared with the 5-loop TMM with canonical layout, TLB misses decrease by a factor of $O(B)$ using the 6-loop TMM. Note that the 6-loop TMM uses block data layout.

To verify our TLB miss estimation, simulations on the 6-loop TMM were performed. The problem size was fixed at 1024×1024. Simulation parameters were the same as those in Section 2. Figure 5 compares our estimations (given by Eq. (1)) with the simulation results. Figure 6 shows that block data layout reduced TLB misses considerably compared with tiling.

3.2 Cache Performance

For a given cache size, tiling transforms the loop nest so that the temporal locality can be better exploited. This reduces the capacity misses. However, since most of the state-of-the-art architectures have direct-mapped or small set-associative caches, tiling can suffer from

considerable conflict misses that degrade the overall performance. Figure 7 (a) shows cache conflict misses. These conflict misses are determined by cache parameters such as cache size, cache line size and set-associativity, and runtime parameters such as array size and block size. Performance of tiled computations is thus sensitive to these runtime parameters.



(a) Canonical layout

(b) Block data layout

Figure 7: Example of conflict misses

If the data layout is reorganized from a canonical layout to a block layout (assuming tile size is same as block size) before tiled computations start, the entire data that is accessed during a tiled computation will be localized in a block. As shown in Figure 7 (b), a self interference miss does not occur if the block is smaller than the cache since all elements in a block can be stored in contiguous memory locations.

In general, cache miss analysis for direct mapped cache with canonical layout is complicated because the self interference misses cannot be quantified easily. Cache performance analysis of tiled algorithm was discussed in [12]. The cache performance of tiling with copying optimization was also presented. We observe that the behavior of cache misses for tiled access patterns on block layout is similar to that of tiling with copying optimization on canonical layout. We have derived the total number of cache misses for 6-loop TMM (which uses block data layout). Detailed proof can be found in Appendix B. For i^{th} level cache with

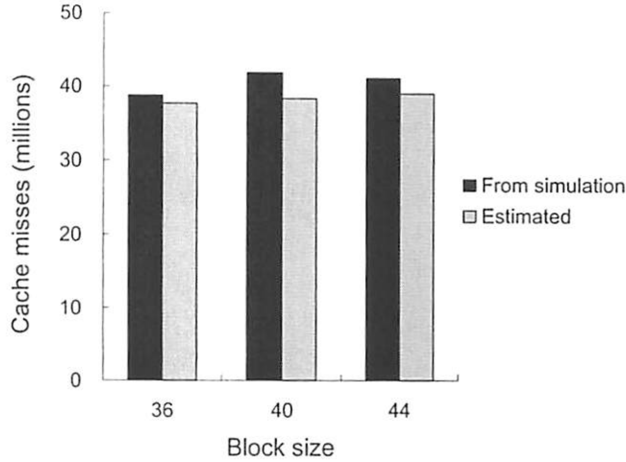


Figure 8: Comparison of cache misses from simulation and estimation for 6-loop TMM

line size L_{ci} and cache size S_{ci}, the total number of cache misses (CM_i) is:

$$CM_i \approx \begin{cases} \frac{N^3}{L_{ci}} \left\{ \frac{1}{B} \left(2 + \frac{(3L_{ci}+2L_{ci}^2)}{S_{ci}} \right) + \frac{1}{N} + \frac{4B+6L_{ci}}{S_{ci}} \right\} & \text{for } B < \sqrt{S_{ci}} \\ \frac{N^3}{L_{ci}} \left\{ \frac{4B}{S_{ci}} + \frac{2}{B} - \frac{2S_{ci}}{B^2} + 2 - \frac{1}{N} + \frac{6L_{ci}}{S_{ci}} \right\} & \text{for } \sqrt{S_{ci}} \leq B < \sqrt{2S_{ci}} \\ \frac{N^3}{L_{ci}} \left\{ 1 + \frac{2}{B} + \left(1 + \frac{L_c}{B}\right) \left(\frac{B+2L_c}{S_{ci}} \right) \right\} & \text{for } \sqrt{2S_{ci}} \leq B \end{cases} \quad (2)$$

To verify the cache miss estimations, we conducted simulations using SimpleScalar for 6-loop TMM with block data layout. The problem size was fixed at 1024×1024. A $16KByte$ direct mapped cache was assumed (similar to L1 data cache in UltraSparc II). Figure 8 compares our estimated values (given by Eq. (2)) with the simulation results.

3.3 Block Size Selection

To test the effect of block size, experiments were performed on several platforms. Figure 9 shows the execution time of a 6-loop TMM with size 1024×1024 on UltraSparc II (400 MHz) as a function of block size. It can be observed that block size selection is significant for achieving high performance.

With canonical data layout, tiling technique is sensitive to problem and tile sizes. Several

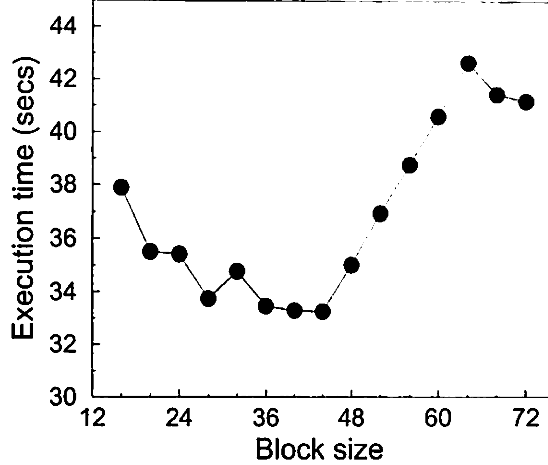


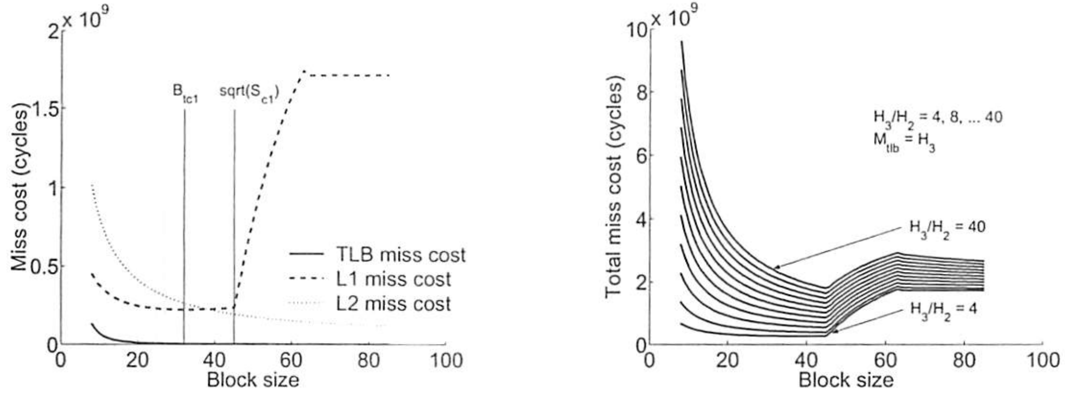
Figure 9: Execution time of TMM of size 1024 × 1024

GCD based tile size selection algorithms [6, 8, 12] were proposed to optimize tiled computation. However, their performance is still sensitive to the problem size. In [13], TLB and cache performance were considered in concert. This approach showed better performance than algorithms that separately consider cache or TLB. However, all these approaches are based on canonical data layout. On the other hand, ATLAS [26] utilizes block data layout. However, the best block size is determined *empirically* at compile time by evaluating the actual performance of the code with a wide range of block sizes.

In a multi-level memory hierarchy system, it is difficult to predict the execution time (T_{exe}) of a program. But, T_{exe} is proportional to the total miss cost of TLB and cache. In order to minimize T_{exe}, we will evaluate and minimize the total miss cost for both TLB and l-level caches. We have:

$$MC = TM \cdot M_{tlb} + \sum_{i=1}^{l} CM_i H_{i+1} \quad (3)$$

where MC denotes the total miss cost, CM_i is the number of misses in the ith level cache, TM is the number of TLB misses, H_i is the cost of a hit in the ith level cache, and M_{tlb} is the penalty of a TLB miss. The $(l+1)^{th}$ level cache is the main memory. It is assumed that all data reside in the main memory ($CM_{l+1} = 0$). Using the derivative of MC with respect



(a) Miss cost of TLB, L1, and L2 cache (B_{tc1} is obtained using Eq.(4))

(b) Total miss cost with various L2 miss penalty

Figure 10: Miss cost estimation for 6-loop TMM (UltraSparc II parameters)

to the block size, we can find the optimal block size that minimizes the overall miss cost.

For a simple 2-level memory hierarchy that consists of only one level cache and TLB, the total miss cost (denoted as MC_{tc1}) in Eq. (3) reduces to:

$$MC_{tc1} = TM \cdot M_{tlb} + CM \cdot H_2,$$

where H_2 is the access cost of main memory. In the above estimation, M_{tlb} and CM are substituted with Eq.(1) and Eq.(2), respectively. Using the derivative of MC, the optimal block size (B_{tc1}) that minimizes the total miss cost caused by L1 cache and TLB misses is given as

$$B_{tc1} \approx \sqrt{\frac{\left(\frac{2L_{1c}M_{tlb}}{P_v} + \left[2 + \frac{3L_{c1}+2L_{c1}^2}{S_{c1}}\right] H_2\right) S_{c1}}{4H_2}}. \tag{4}$$

We now extend this analysis to determine a range for optimal block size in a multi-level memory hierarchy that consists of TLB and two levels of cache. The miss cost is classified into two groups: miss cost caused by TLB and L1 cache misses and miss cost caused by L2 misses. Figure 10 (a) and (b) show the miss cost estimated through Eqs.(1) and (2). Fig. 10(a) is the separated TLB, L1, and L2 miss cost, using UltraSparc II parameters.

Fig. 10(b) shows the variance of the estimated total miss costs as the ratio between L1 cache miss penalty (H_2) and L2 cache miss penalty (H_3) varies. Using Eq.(4), we discuss the total miss cost for 3 ranges of block size:

Lemma 3.2 *For $B < B_{tc1}$, $MC(B) > MC(B_{tc1})$.*

Proof: Using the derivatives of TLB and cache miss equations (Eq.(1) and (2)), it can be easily verified that $\frac{dMC_{tc1}}{dB} < 0$ and $\frac{dCM_2}{dB} < 0$ for $B < B_{tc1}$. This is shown in Figure 10(a). For $B < B_{tc1}$, TLB, L1, and L2 miss cost increase as block size decreases, thereby increasing the total miss cost. Therefore, the optimal block size cannot be in the range $B < B_{tc1}$. ⊙

Lemma 3.3 *For $B > \sqrt{S_{c1}}$, $MC(B) > MC(\sqrt{S_{c1}})$.*

Proof: In the range $B > \sqrt{S_{c1}}$, TLB miss cost is optimized by tiling and block data layout. However, the change in TLB miss cost is negligible as the block size increases. Since block size is larger than L1 cache size, self-interferences occur in this range. The number of L1 cache misses drastically increases as shown in Figure 10(a). For $\sqrt{S_{c1}} \le B < \sqrt{2S_{c1}}$, the ratio of derivatives of Eq.(2) for L1 and L2 misses is as follows:

$$\left|\frac{H_2\frac{dCM_1}{dB}}{H_3\frac{dCM_2}{dB}}\right| = \frac{H_2}{H_3}\left|\frac{\frac{N^3}{L_{c1}}\left[\frac{4}{S_{c1}} + \frac{4Sc1}{B^3} - \frac{2}{B^2}\right]}{\frac{N^3}{L_{c2}}\left[\frac{4}{S_{c2}} - \left(2 + \frac{3L_{c2}+2L_{c2}^2}{S_{c2}}\right)\frac{1}{B^2}\right]}\right|.$$

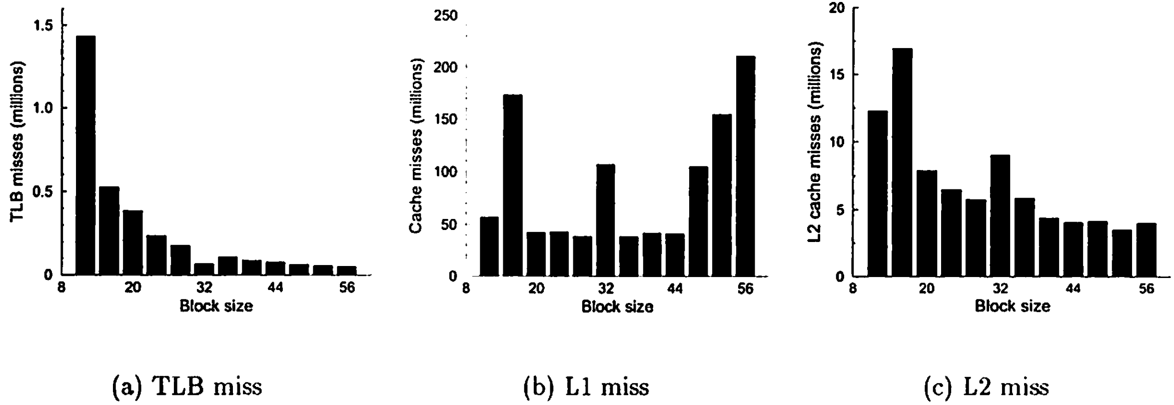
Let $B^2 = \alpha S_{c1}$ $(1 \le \alpha < 2)$. Note that $L_{c2} \ll S_{c2}$.

$$\left|\frac{H_2\frac{dCM_1}{dB}}{H_3\frac{dCM_2}{dB}}\right| \approx \frac{H_2}{H_3}\cdot\frac{L_{c2}}{L_{c1}}\cdot\frac{S_{c2}}{S_{c2} - 2\alpha S_{c1}}\cdot\left(2\alpha - 1 + \frac{2}{\sqrt{\alpha}}\sqrt{S_{c1}}\right) > \frac{H_2}{H_3}\cdot\frac{L_{c2}}{L_{c1}}\cdot\frac{S_{c2}}{S_{c2} - 4S_{c1}}\cdot\left(3 + \sqrt{2}\cdot\sqrt{S_{c1}}\right)$$

In a general memory hierarchy system, $\frac{S_{c2}}{S_{c2}-4S_{c1}} \approx 1$ since $S_{c1} \ll S_{c2}$. Also, $\frac{L_{c2}}{L_{c1}} \ge 1$ and $\sqrt{2S_{c1}} > \frac{H3}{H2}$. Therefore,

$$\left|\frac{H_2\frac{dCM_1}{dB}}{H_3\frac{dCM_2}{dB}}\right| > 1$$

Thus, although the number of L2 cache misses decreases ($\frac{dCM_2}{dB} < 0$), the total miss cost increases for $\sqrt{S_{c1}} \le B < \sqrt{2S_{c1}}$ because the increase in L1 cache miss cost is larger than



(a) TLB miss (b) L1 miss (c) L2 miss

Figure 11: Simulation results of 6-loop TMM

the decrease in L2 cache miss cost. For $B \geq \sqrt{2S_{c1}}$, there is no reuse in L1 cache. Thus, the L1 cache miss cost saturates. Figure 10(b) shows the change of the total miss cost as the ratio of $\frac{H_3}{H_2}$ varies. Even though L2 miss penalty is 40 times that of L1 miss penalty, $TM(B) > TM(\sqrt{S_{c1}})$ for $B \geq \sqrt{2S_{c1}}$, because L1 self-interference miss cost is dominantly large for $B \geq \sqrt{2S_{c1}}$. Therefore, the optimal block size cannot be in the range $B > \sqrt{S_{c1}}$. ⊙

Theorem 3.2 *The optimal block size B_{opt} satisfies $B_{tc1} \leq B_{opt} < \sqrt{S_{c1}}$.*

Proof: This follows from Lemma 3.2 and 3.3. Therefore, an optimal block size that minimizes the total miss cost is located in

$$B_{tc1} \leq B_{opt} < \sqrt{S_{c1}}. \tag{5}$$

We select a block size that is a multiple of L_{c1} (L1 cache line size) in this range. ⊙

To verify our approach, we conducted simulations using UltraSparc II parameters (Table 3). Figure 11 shows the simulation results of 6-loop TMM using block data layout. As discussed, the number of TLB and L2 misses decreased as block size increases. Also, the minimum number of L1 misses was obtained for $B = 36$ and then drastically increased for $B > 45$. Figure 12 shows the total miss cost. For UltraSparc II, $B_{tc1} = 32.2$, $\sqrt{S_{c1}} = 45.3$,

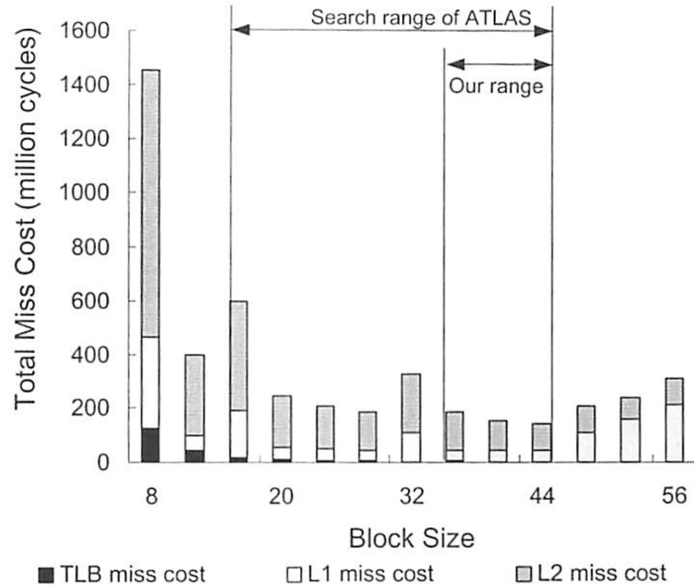


Figure 12: Total miss cost for 6-loop TMM

and $L_{c1} = 4$. Theorem 3.2 suggests the range for optimal block size is to be 36–44. Simulation results show that the optimal block size for this architecture was 44.

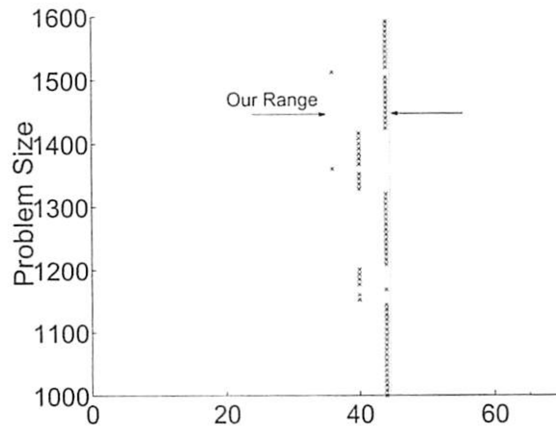


Figure 13: Optimal block sizes for 6-loop TMM

We also tested ATLAS on UltraSparc II. Through a wide search ranging from 16 to 44, ATLAS found 36 and 40 as the optimal block sizes. These blocks lie in the range given by Eq. (5). We further tested 6-loop TMM with respect to different problem and block

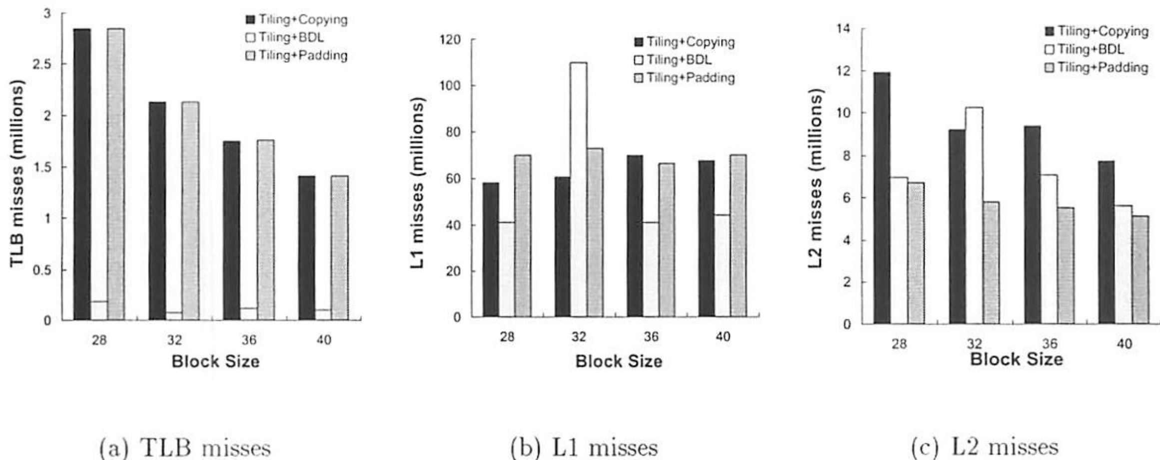
sizes. For each problem size, we performed experiments by testing block sizes ranging from 8–80. In these tests, we found that the optimal block size for each problem size was in the range given by Eq. (5) as shown in Figure 13. These experiments confirm that our approach proposes a reasonably good range for block size selection.

4 Experimental Results

To verify our analysis, we performed simulations and experiments on the following applications: tiled matrix multiplication(TMM), LU decomposition, and Cholesky factorization(CF). The performance of tiling with block data layout (tiling+BDL) is compared with other optimization techniques: tiling with copying(tiling+copying), and tiling with padding(tiling+padding). For tiling+BDL, the tile size (of the tiling technique) is chosen to be the same as the block size of the block data layout. Input and output is in canonical layout. All the cost in performing data layout transformations (from canonical layout to block data layout and vice versa) is included in the reported results. As stated in [12], we observed that the copying technique cannot be applied efficiently to LU and CF applications, since copying overhead offsets the performance improvement. Hence we do not consider tiling+copying for these applications. In all our simulations and experiments, the data elements are double-precision.

4.1 Simulations

To show the performance improvement of TLB and caches using tiling+BDL, simulations were performed using the SimpleScalar simulator [2]. The problem size was 1024×1024. Two sets of architecture parameters were used: UltraSparc II and Pentium III. The parameters are listed in Table 3.



(a) TLB misses (b) L1 misses (c) L2 misses

Figure 14: Simulation results for TMM using UltraSparc II parameters

Figure 14 compares the TMM simulations of different techniques, based on UltraSparc II parameters. Tiling+BDL reduces L1 and L2 cache misses as well as TLB misses. Block size 32 leads to increased L1 and L2 cache misses for block data layout because of the cache conflicts between different blocks. Tiling+BDL reduced 91–96% of TLB misses as shown in Figure 14(a). This confirms our analysis presented in Section 3.1. Figure 15 shows the total miss cost (calculated from Eq. (3)) for TMM using block size 40×40. L1, L2, and TLB miss penalties were assumed to be 6, 24, and 30 cycles, respectively. This figure shows that tiling+BDL results in the smallest total miss cost and that the TLB miss cost with tiling+BDL is negligible compared with L1 and L2 miss costs. Though tiling+BDL has more L2 cache misses than tiling+padding, its total miss cost is smaller. Figure 12 shows the effect of block size on the total miss cost for TMM using tiling+BDL. As discussed in Section 3.3, the best block size (44) is in the range 36–44 suggested by our approach.

Figure 16 presents simulation results for LU using Intel Pentium III parameters. Similar to TMM, the number of TLB misses for tiling+BDL was almost negligible compared with that for tiling+padding as shown in Figure 16(a). For both techniques, L1 and L2 cache misses were reduced considerably because of 4-way set-associativity. For tiling+padding, when the block size was larger than L1 cache size, the padding algorithm in [16] suggested a

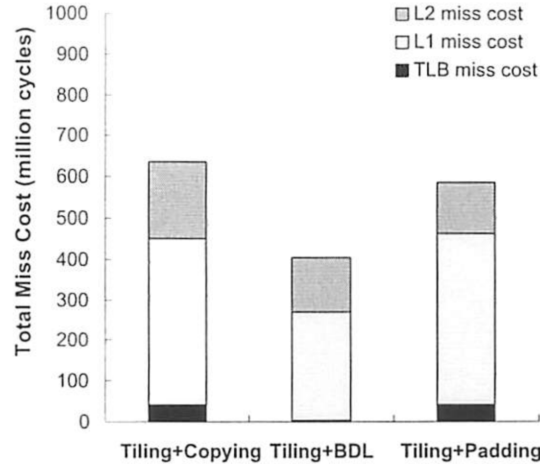
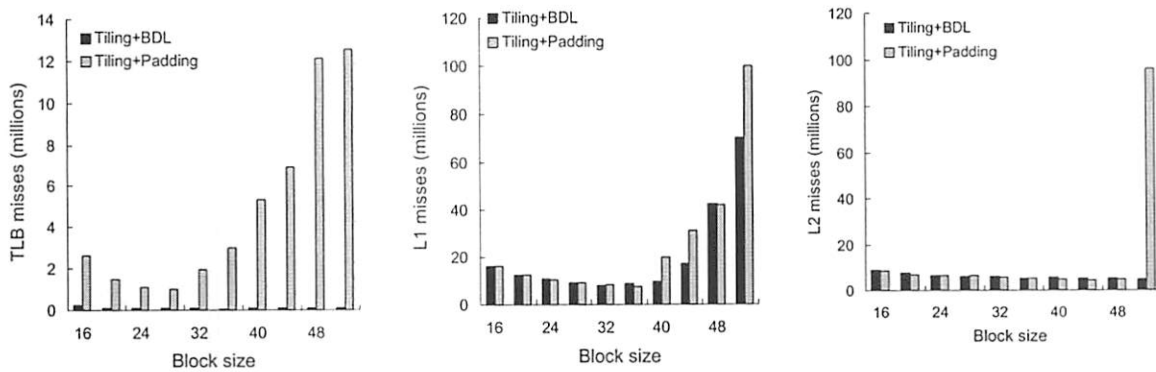


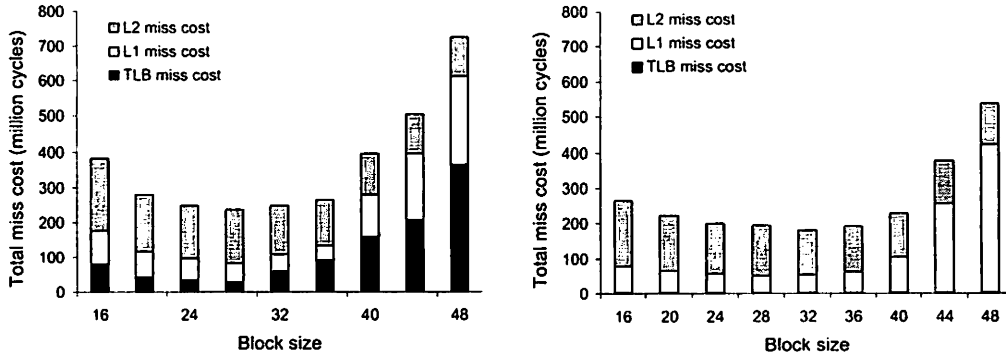
Figure 15: Total miss cost for TMM using UltraSparc II parameters



(a) TLB misses (b) L1 cache misses (c) L2 cache misses

Figure 16: Simulation results for LU using Pentium III parameters

pad size of 0. There is essentially no padding effect, thereby drastically increasing L1 and L2 cache misses. Figure 17 shows the block size effect on total miss cost using tiling+padding and tiling+BDL. Tiling+padding reduced L1 and L2 cache miss costs considerably. However, TLB miss costs were still significantly high, affecting the overall performance. As discussed in Section 3.3, the suggested range for optimal block size is 32–44. Simulations validate that the optimal block size achieving the smallest miss cost locates in the range selected using our approach.



(a) Tiling+Padding (b) Tiling+BDL

Figure 17: Effect of block size on LU decomposition using Pentium III parameters

Table 3: Features of various experimental platforms

Platforms	Speed (MHz)	L1 cache			L2 cache			TLB		
		Size (KB)	Line ($Byte$)	Ass.	Size (KB)	Line ($Byte$)	Ass.	Entry	page (KB)	Ass.
Alpha 21264	500	64	64	2	4096	64	1	128	8	128
UltraSparc II	400	16	32	1	2048	64	1	64	8	64
UltraSparc III	750	64	32	4	4096	64	4	512	8	2
Pentium III	800	16	32	4	512	32	4	64	4	4

4.2 Execution on real platforms

To verify our block size selection and the performance improvements using block data layout, we performed experiments on several platforms as tabulated in Table 3. `gcc` compiler was used in these experiments. The compiler optimization flags were set to "`-fomit-frame-pointer -O3 -funroll-loops`". Execution time was the user processor time measured by sys-call `clock()`. All the data reported here is the average of 10 executions. The problem sizes ranged from 1000×1000 to 1600×1600.

Figure 18 shows the comparison of execution time of tiling+BDL with other techniques.

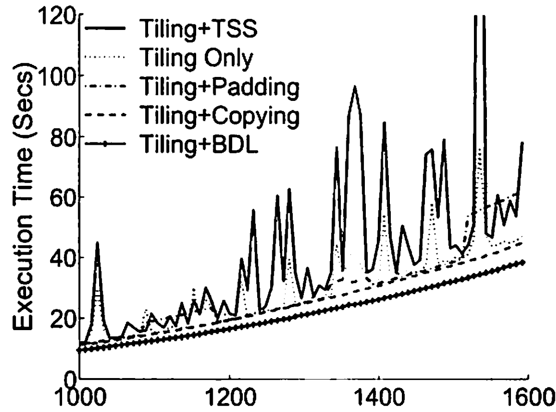
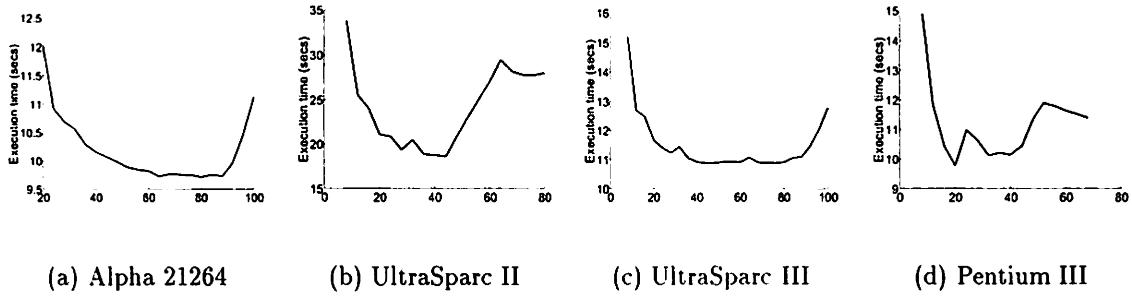


Figure 18: Execution time comparison of various techniques for TMM on Pentium III

The performance of tiling+TSS (tile size selection algorithm [6]) shown in this figure selects block size based on GCD computation. Tiling solves the cache capacity miss problem but it cannot avoid conflict misses. Conflict misses are strongly related to the problem size and block size. This makes tiling sensitive to problem size. As discussed on Section 3.2, block data layout greatly reduces conflict misses, resulting in smoother performance compared with others.

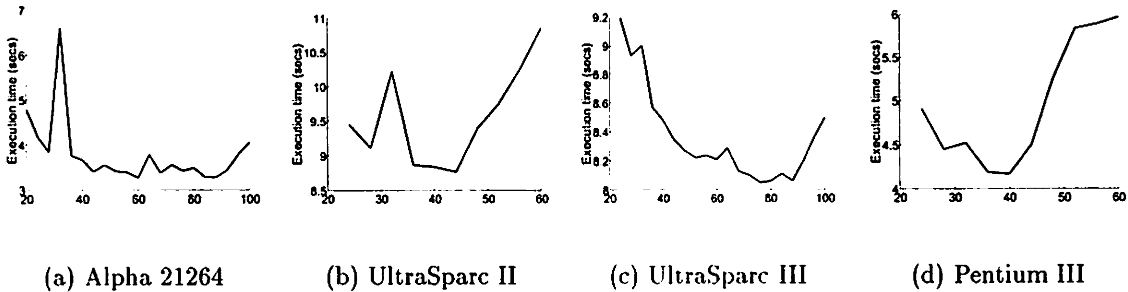
The effect of block size on tiling+BDL is shown in Figures 19–21. Various problem sizes were tested and results on all these problems showed similar trends as in Figures 19–21. As an illustration, the results for problem size of 1024×1024 are shown. As shown in Figures 19–21, the optimal block sizes for Pentium III, UltraSparc II, Sun UltraSparc III and Alpha 21264 are 40, 44, 76, and 76 respectively. All these numbers are in the range given by our block size selection algorithm. For example, the range for best block size on Alpha 21264 is 64–78. This confirmed that our block size selection algorithm proposes a reasonable range. As discussed earlier, block sizes 32 and 64 should be avoided (for use with block data layout) because the performance degrades due to conflict misses between blocks.

Figures 22–24 show the execution time comparison of tiling+BDL with tiling+copying and tiling+padding. In these figures, block size for tiling+BDL was given by our algorithm



(a) Alpha 21264 (b) UltraSparc II (c) UltraSparc III (d) Pentium III

Figure 19: Effect of block size on TMM



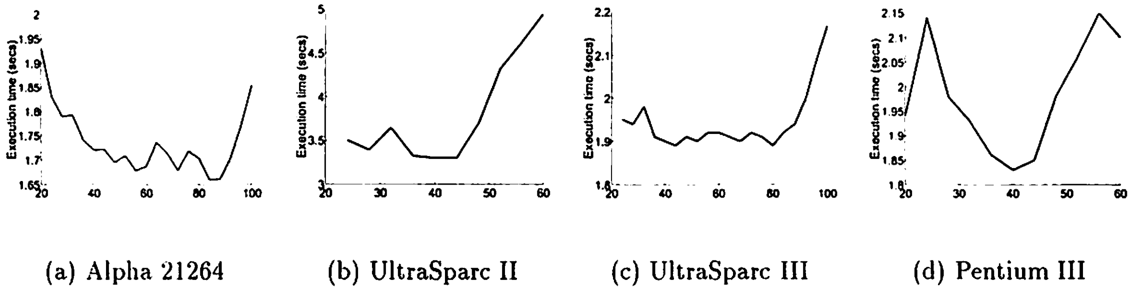
(a) Alpha 21264 (b) UltraSparc II (c) UltraSparc III (d) Pentium III

Figure 20: Effect of block size on LU decomposition

discussed in Section 3.3. The tile size for the copying technique was given by the approach in [12]. The pad size was selected by the algorithm discussed in [16]. Tiling+BDL technique is faster than using other optimization techniques, for almost all problem sizes and on all the platforms.

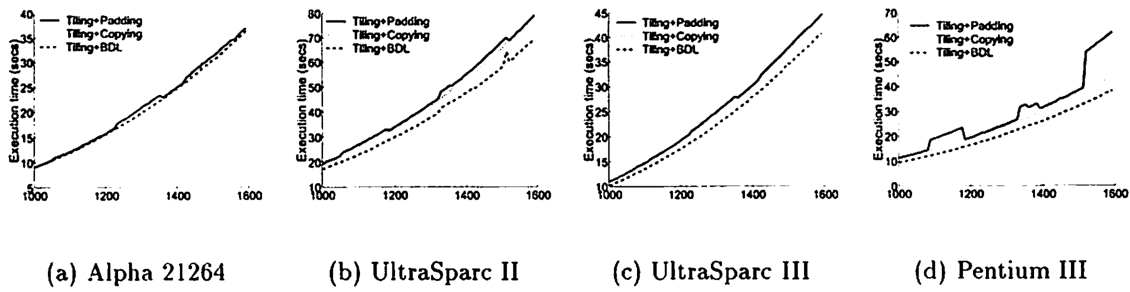
4.3 Block data layout and Morton data layout

Recently nonlinear data layouts have been considered to improve memory hierarchy performance. One such layout is the Morton data layout(MDL) as defined in Section 2.1. Similar to block data layout, elements within each block are mapped onto contiguous memory locations. However, Morton data layout uses a different order to map blocks as shown in Figure 1. This order matches the access pattern of recursive algorithms. In this section, we compare the performance of recursive algorithms using MDL (recursive+MDL) with iterative tiled



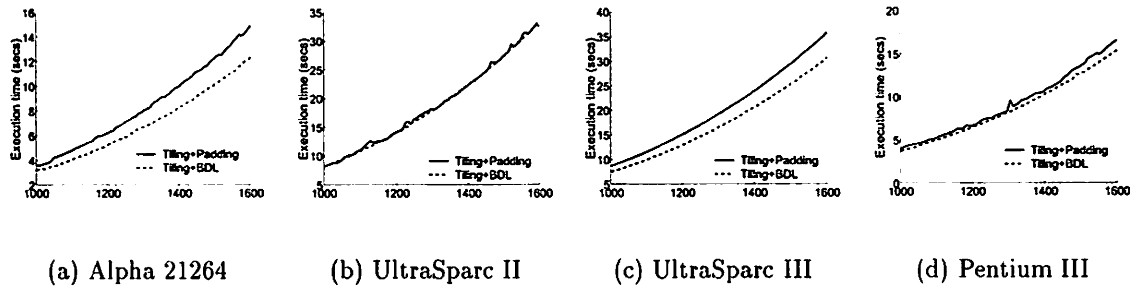
(a) Alpha 21264 (b) UltraSparc II (c) UltraSparc III (d) Pentium III

Figure 21: Effect of block size on Cholesky factorization



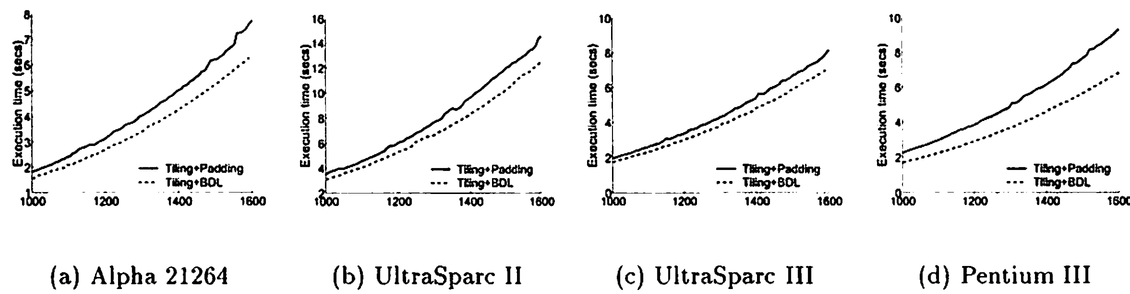
(a) Alpha 21264 (b) UltraSparc II (c) UltraSparc III (d) Pentium III

Figure 22: Execution time of TMM



(a) Alpha 21264 (b) UltraSparc II (c) UltraSparc III (d) Pentium III

Figure 23: Execution time of LU decomposition



(a) Alpha 21264 (b) UltraSparc II (c) UltraSparc III (d) Pentium III

Figure 24: Execution time of Cholesky factorization

Table 4: Comparison of execution time of TMM on various platforms: All times are in seconds.

(a) Pentium III

Size	iterative+BDL	recursive+MDL
1024	10.37	10.98
1280	20.43	20.64
1408	27.06	28.21
1600	39.77	43.78
2048	83.27	87.64

(b) UltraSparc II

Size	iterative+BDL	recursive+MDL
1024	18.87	21.80
1280	36.17	40.63
1408	48.76	53.70
1600	70.44	81.61
2048	149.65	170.86

algorithms using BDL (iterative+BDL), for matrix multiplication and LU decomposition. We show that the performance of recursive+MDL is comparable with that of iterative+BDL if the block size of MDL lies in the optimal block size range for BDL as given by our algorithm (Eq. (5) in Section 3.3). However, if the block size of MDL is outside this range, recursive+MDL is slower than iterative+BDL.

Similar to block data layout, block size for Morton layout also plays an important role in the performance. However, due to recursion, the choice of block sizes is limited. For an $N \times N$ matrix, if the depth of recursion is d, the block size of MDL is given by $B_{MDL} = \frac{N}{2^d}$. Such a block size can lie outside the optimal range given by our approach. Our experiment results show that this degrades the overall performance.

Experiments using TMM and LU were performed on UltraSparc II and Pentium III. Table 4 shows the execution time comparison of MM using iterative+BDL with recursive+MDL. For iterative+BDL, we selected the block size according to the algorithm discussed in Section 3.3. For recursive+MDL, we tested various recursion depths (resulting in various basic block sizes) and used the best for comparison. For problem size 1280×1280 and 1408×1408, optimal block sizes for recursive+MDL were 40 and 44 respectively, which were in the range given by our algorithm, 36–44. Both the layouts showed competitive performance for these

Table 5: Comparison of execution time of LU decomposition on various platforms: All times are in seconds.

(a) Pentium III

Size	iterative+BDL	recursive+MDL
1024	4.15	4.43
1280	8.10	8.10
1408	10.85	11.57
1600	15.85	18.44
2048	33.58	35.90

(b) UltraSparc II

Size	iterative+BDL	recursive+MDL
1024	8.77	9.94
1280	18.97	18.54
1408	22.76	22.45
1600	33.51	35.58
2048	75.30	81.66

cases. For problem size 1600×1600, recursive+MDL was up to 15.8% slower than iterative+BDL. Among possible choices of 25, 50, and 100, the performance of recursive+MDL was optimized at block size 25, where $25 = \frac{1600}{2^5}$. This is because it is outside the optimal range specified by our algorithm. Table 5 shows the execution time comparison of tiled LU decomposition using BDL and recursive LU decomposition [27] using MDL. These results confirm our analysis.

5 Concluding Remarks

This paper studied a critical problem in understanding the performance of algorithms on state-of-the-art machines that employ multi-level memory hierarchy. We showed that using block data layout, TLB misses as well as cache misses are reduced considerably. Further, we proposed a tight range for block size using our performance analysis. Our analysis matches closely with simulation based as well as experimental results.

This work is part of the Algorithms for Data IntensiVe Applications on Intelligent and Smart MemORies (ADVISOR) Project at USC [1]. In this project we focus on developing algorithmic design techniques for mapping applications to architectures. Through this we

understand and create a framework for application developers to exploit features of advanced architectures to achieve high performance.

Acknowledgment

We would like to thank Shriram Bhargava Gundala for careful reading of drafts of this work. We also would like to thank Sriram Vajapeyam and Cauligi S. Raghavendra for their inputs on a preliminary version of this work.

Appendix A TLB performance of block data layout

This section gives a detailed proof of Theorem 2.2 in Section 2.2. The theorem is repeated for convenience:

Theorem *For accessing an array along all the rows and then along all the columns, block data layout with block size $\sqrt{P_v} \times \sqrt{P_v}$ minimizes the number of TLB misses.*

Proof: Suppose the block size $B^2 = kP_v$. Two cases (for $k \geq 1$ and $k \leq 1$) are discussed separately.

Case I: $k \geq 1$. We consider three scenarios for this case.

1. $\frac{N}{B} > S_{tlb}$

   Accesses to the first row cause $\frac{N}{B}$ TLB misses. However, these entries cannot be reused since S_{tlb} is small. Therefore, TLB misses caused by row accesses is $T_{row} = \frac{N}{B} \times N$. Similarly, TLB misses caused by column accesses are $T_{col} = \frac{N}{B} \cdot k \cdot N$. Therefore, the total number of TLB misses is

$$T_{miss} = \frac{N^2}{B} + k\frac{N^2}{B} = \frac{N^2}{\sqrt{P_v}}(\frac{1}{\sqrt{k}} + \sqrt{k}).$$

To minimize the total TLB misses,

$$\frac{dT_{miss}}{dk} = \frac{N^2}{\sqrt{2P_v}} \times \frac{1}{\sqrt{k}} \times (1 - \frac{1}{k}).$$

Therefore, as k decreases, the total number of TLB misses decreases. The total number of TLB misses is minimized when $k = 1$. Note that when $B = \sqrt{P_v}$ ($k = 1$), the number of TLB misses is $2\frac{N^2}{\sqrt{P_v}}$, which is the lower bound given by Theorem 2.1.

2. $\frac{N}{B} \leq \frac{S_{tlb}}{k}$

   In this scenario, both column and row access can reuse TLB entries. Therefore, the total number of TLB misses is

$$T_{miss} = 2k\frac{N^2}{B^2} = 2\frac{N^2}{P_v}.$$

   This is equal to twice the number of TLB misses caused by all row accesses in canonical layout. Therefore, this will be the minimum number of TLB misses for such an access pattern.

3. $\frac{S_{tlb}}{k} < \frac{N}{B} \leq S_{tlb}$

   In this scenario, only row accesses can reuse TLB entries accessed in the previous row accesses. TLB misses for row accesses are $T_{row} = k\frac{N^2}{B^2}$. Therefore, the total number of TLB misses is

$$T_{miss} = k\frac{N^2}{B^2} + k\frac{N^2}{B} = \frac{N^2}{P_v} + \frac{N^2}{\sqrt{P_v}}\sqrt{k}.$$

   As k decreases, TLB misses also decrease. The number of TLB misses for block data layout is minimized when k approaches 1. Note that this scenario will reduce to scenario 2 when $k = 1$. Therefore, the minimum number of TLB misses in this scenario is the same as that in scenario 2.

Case II: $k \leq 1$. Three scenarios are discussed as follows:

1. $\frac{N}{B} > \frac{S_{tlb}}{k}$

   The first row access causes $k\frac{N}{B}$ TLB misses. These entries cannot be reused in the next row access. TLB misses caused by row accesses are $T_{row} = k\frac{N}{B} \cdot N$. On the other hand, the first column access causes $\frac{N}{B}$ TLB misses, since all the elements in each block are stored in one page. The TLB misses caused by column accesses is $T_{col} = \frac{N}{B} \cdot N$. Therefore, the total number of TLB misses is

$$T_{miss} = k\frac{N^2}{B} + \frac{N^2}{B} = \frac{N^2}{\sqrt{P_v}}(\frac{1}{\sqrt{k}} + \sqrt{k}).$$

   To minimize the total TLB misses,

$$\frac{dT_{miss}}{dk} = \frac{N^2}{\sqrt{2P_v}} \times \frac{1}{\sqrt{k}} \times (1 - \frac{1}{k}).$$

   Therefore, as k increases, the total number of TLB misses decreases. The total number of TLB misses is minimized when $k = 1$, $B = \sqrt{P_v}$. Again, the minimum number is $2\frac{N^2}{\sqrt{P_v}}$, equal to the lower bound given by Theorem 2.1.

2. $\frac{N}{B} \leq S_{tlb}$

   In this scenario, both row and column access can reuse TLB entries. Therefore, the total number of TLB misses is

$$T_{miss} = 2k\frac{N^2}{B^2} = 2\frac{N^2}{P_v}.$$

   This is equal to twice the number of TLB misses caused by all row accesses in the canonical layout. Therefore, it is the minimal number of TLB misses caused by all row (1st pass) and then all column (2nd pass) accesses.

3. $S_{tlb} < \frac{N}{B} \leq \frac{S_{tlb}}{k}$

   In this scenario, only row accesses can reuse TLB entries accessed in the previous row accesses. TLB misses of row accesses is denoted as: $T_{row} = k\frac{N^2}{B^2}$. Therefore, the total number of TLB misses is

$$T_{miss} = k\frac{N^2}{B^2} + \frac{N^2}{B} = \frac{N^2}{P_v} + \frac{N^2}{\sqrt{kP_v}}.$$

As k increases, TLB misses decrease. Like scenario 3 in Case I, the minimum number of TLB misses in this scenario is obtained when $k = 1$, and this number is the same as that in the previous scenario.

According to the above analysis, block data layout with block size $\sqrt{P_v} \times \sqrt{P_v}$ minimizes the total number of TLB misses. As the problem size (N) increases, this minimum number asymptotically approaches the lower bound given by Theorem 2.1.

$\odot$

Appendix B Cache Miss Analysis

In this section, we provide detailed cache miss analysis for a tiled access pattern with block data layout. Individual levels of cache are not considered explicitly, as this analysis is applicable to all cache levels. We consider a tiled program that consists of nested loops. Each loop level is denoted by the loop index i, j, l, etc. Arrays referenced by the program are denoted as u, v, etc. Within an iteration of a loop l, a portion of an array v (called the footprint $F_p(v)$) is referenced. The body of the loop l will be executed $R(v)$ times, where $R(v)$ is the reuse factor.

Let (i) $ICM_l(v)$ denote the number of *intrinsic* cache misses [12] caused by accessing array v during the first iteration of loop l; (ii) $SCM_l(v)$ denote the number of self-interference misses when array v is accessed in one iteration of loop l; (iii) $CIM(v)$ denote the number of cross-interference misses between array v and other arrays for an iteration of loop l. The number of cache misses caused by array v for one iteration of loop l is thus:

$$CM_l(v) = ICM_l(v) - SCM_l(v) + R(v) \times \{SCM_l(v) + CIM(v)\} \quad (6)$$

$CIM(v)$ in the above equation can be calculated as:

$$CIM(v) = ICM_l(v) \times PrCF(v), \quad (7)$$

where $PrCF(v)$ denotes the probability of conflict between one element of array v and elements of other arrays for loop l. It is given by

$$PrCF(v) = \sum_{u \neq v} Prov_{cf}(v, u),$$

where $Prov_{cf}(v, u)$ is the probability that an element of array v falls into the footprint of the array u, accessed with a stride (s_u) in the cache. For simplicity, it is assumed that an element of array v does not conflict with elements in two or more arrays at the same time.

The cache misses of array v is computed as follows:

$$CM(v) = NIO(l) \times CM_l(v), \tag{8}$$

where $NIO(l)$ denotes the total number of iterations of outer loops. The total number of misses incurred by accessing all arrays is the sum of misses incurred in accessing individual arrays ($\sum_i CM(i)$). The above cache miss equation (Eq.(8)) is applicable to any data layout with nested loops. But the factors ($SCM_l(v)$, $Prov_{cf}(v, u)$, $R(v)$, etc.) cannot be quantified unless the data layout and loop structure are known.

For block data layout, we can easily quantify $SCM_l(v)$ and $Prov_{cf}(v, u)$ in the above equations. The number of self-interferences can be derived by considering three ranges of block sizes. (i) When the block size is less than the cache size, there is no self-interference. (ii) When the block size is larger than twice the cache size, there is no reused element in cache, resulting in $\frac{B^2}{L_c}$ self-interferences misses. (iii) When the block size is in between the above ranges, $\frac{2(B^2 - S_c)}{L_c}$ self-interference misses occur. Hence,

$$SCM_l(v) = \begin{cases} 0 & \text{for } B < \sqrt{S_c} \\ \frac{2(B^2 - S_c)}{L_c} & \text{for } \sqrt{S_c} \leq B < \sqrt{2S_c} \\ \frac{B^2}{L_c} & \text{for } \sqrt{2S_c} \leq B \end{cases}$$

For loop l, $F_p(v)$ elements of array v are accessed with a stride (s_v). The average number of

Table 6: Parameters of TMM

Array	Reuse Factor			Footprint		
	i	k	j	i	k	j
$\mathbf{X}(i,k)$			B	B	1	
$\mathbf{Y}(k,j)$	N			B^2	B	
$\mathbf{Z}(i,j)$		B		B	B	

cache lines occupied by $F_p(v)$ elements is

$$NCL(v) = \begin{cases} \frac{F_p(v)s_v}{L_c} + 1 & \text{if } 1 < s_v < L_c \\ F_p(v) & \text{otherwise} \end{cases}$$

During a tiled computation, a block of array v is accessed in loop l. Hence, $ICM_l(v)$ is equal to the number of cache lines, $NCL(v)$. For loop l, array u is accessed with stride(s_u) whose footprint size is $F_p(u)$. It occupies $NCL(u)$ cache lines in the cache. The probability of conflicting with array u is

$$Prov_{cf}(v,u) = \frac{NCL(u)}{S_c/L_c}.$$

Therefore, the cache misses of array v on block data layout is

$$CM(v) = NIO(l) \times \left\{ NCL(v) - SCM_l(v) + R(v) \times \left(SCM_l(v) + NCL(v) \times \sum_{u \neq v} \frac{NCL(u)L_c}{S_c} \right) \right\}. \quad (9)$$

Consider the 6-loop TMM shown in Figure 2(b). The reuse factors and footprint sizes of arrays X, Y and Z can be determined. The values are shown in Table 6. For example, consider an array Y in loop i. B^2 elements of Y are referenced in each iteration of loop i. These B^2 elements are reused N times. $NIO(l)$ can be obtained directly from the code (Figure 2(b)). For example, $NIO(i) = N^3/B^3$. According to Eq.(9), the number of cache

misses for **Y** and **Z** are as follows:

$$CM(\mathbf{Y}) \approx \begin{cases} \frac{N^3}{L_c}\left\{\frac{1}{N} + \left(1+\frac{L_c}{B^2}\right)\frac{3(B+L_c)}{S_c}\right\} & \text{for } B < \sqrt{S_c} \\ \frac{N^3}{L_c}\left\{\frac{2S_c}{B^2}\left(\frac{1}{N}-1\right) + 2 - \frac{1}{N} + \left(1+\frac{L_c}{B^2}\right)\frac{3(B+L_c)}{S_c}\right\} & \text{for } \sqrt{S_c} \le B < \sqrt{2S_c} \\ \frac{N^3}{L_c} & \text{for } \sqrt{2S_c} \le B \end{cases}$$

$$CM(\mathbf{Z}) \approx \frac{N^3}{L_c}\left\{\frac{1}{B} + \left(1+\frac{L}{B}\right)\frac{(B+2L)}{S_c}\right\}$$

In the 6-loop TMM, each element of array **X** is immediately allocated to a register. So, its probability of conflicts with other arrays is 0. Thus, the number of cache misses for array **X** is

$$CM(\mathbf{X}) = \frac{N^3}{BL_c}.$$

The total number of cache misses for the 6-loop TMM with block data layout is thus:

$$CM = \sum_v CM(v) = CM(\mathbf{X}) + CM(\mathbf{Y}) + CM(\mathbf{Z}) \tag{10}$$

$$\approx \begin{cases} \frac{N^3}{L_c}\left\{\frac{1}{B}\left(2+\frac{(3L_c+2L_c^2)}{S_c}\right) + \frac{1}{N} + \frac{4B+6L_c}{S_c}\right\} & \text{for } B < \sqrt{S_c} \\ \frac{N^3}{L_c}\left\{\frac{4B}{S_c} + \frac{2}{B} - \frac{2S_c}{B^2} + 2 - \frac{1}{N} + \frac{6L_c}{S_c}\right\} & \text{for } \sqrt{S_c} \le B < \sqrt{2S_c} \\ \frac{N^3}{L_c}\left\{1 + \frac{2}{B} + \left(1+\frac{L_c}{B}\right)\left(\frac{B+2L_c}{S_c}\right)\right\} & \text{for } \sqrt{2S_c} \le B \end{cases} \tag{11}$$

The above analysis focuses on the access pattern of 6-loop TMM. Because matrix multiplication is the kernel of many linear algebra computations, the analysis can be generalized or directly applied to other linear algebra applications.

References

[1] ADVISOR Project. http://advisor.usc.edu.

[2] D. Burger and T. M. Austin. The SimpleScalar Tool Set, Version 2.0. Technical Report 1342, University of Wisconsin-Madison Computer Science Department, June 1997.

[3] J. B. Carter, W. C. Hsieh, L. B. Stoller, M. R. Swanson, L. Zhang, E. L. Brunvand, A. Davis, C.-C. Kuo, R. Kuramkote, M. A. Parker, L. Schaelicke, and T. Tateyama.

Impulse: Building a Smarter Memory Controller. *Proceedings of the Fifth International Symposium on High Performance Computer Architecture (HPCA-5)*, pages 70–79, January 1999.

[4] S. Chatterjee, V. V. Jain, A. R. Lebeck, S. Mundhra, and M. Thottethodi. Nonlinear Array Layouts for Hierarchical Memory Systems. *Proceedings of the 13th ACM International Conference on Supercomputing (ICS '99)*, June 1999.

[5] M. Cierniak and W. Li. Unifying Data and Control Transformations for Distributed Shared-Memory Machines. *Proceedings of the SCM SIGPLAN'95 Conference on Programming Language Design and Implementsion*, pages 205–217, June 1995.

[6] S. Coleman and K. S. McKinley. Tile Size Selection Using Cache Organization and Data Layout. *Proceedings of the SIGPLAN '95 Conference on Programming Language Design and Implementation*, June 1995.

[7] R. Espasa, J. Corbal, and M. Valero. Command Vector Memory Systems: High Performance at Low Cost. Technical Report UPC-DAC-1998-8, Universitat Polit'ecnica de Catalunya, 1998.

[8] K. Esseghir. Improving data locality for caches. Master's thesis, Dept. of Computer Scienece, Rice University, September 1993.

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

[10] T. L. Johnson, M. C. Merten, and W. W. Hwu. Run-time Spatial Locality Detection and Optimization. *Proceedings of the 30th International Symposium on Microarchitecture*, December 1997.

[11] M. Kandemir, A. Choudhary, J. Ramanujam, and P. Banerjee. Improving Locality Using Loop and Data Transformations in an Integrated Framework. *Proceedings of the 31st IEEE/ACM International Symposium on Microarchitecture*, November 1998.

[12] M. Lam, E. Rothberg, and M. E. Wolf. The Cache Performance and Optimizations of Blocked Algorithms. *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IV)*, April 1991.

[13] N. Mitchell, K. Högstedt, L. Carter, and J. Ferrante. Quantifying the Multi-Level Nature of Tiling Interactions. *International Journal of Parallel Programming*, 1998.

[14] D. Padua. The Fortran I Compiler. *IEEE Computing in Science & Engineering*, January/Febrary 2000.

[15] D. A. Padua. Outline of a Roadmap for Compiler Technology. *IEEE Computing in Science & Engineering*, Fall 1996.

[16] P. R. Panda, H. Nakamura, N. Dutt, and A. Nicolau. Augmenting Loop Tiling with Data Alignment for Improved Cache Performance. *IEEE Transactions on Computers*, 48(2), Feburary 1999.

[17] N. Park, D. Kang, K. Bondalapati, and V. K. Prasanna. Dynamic Data Layouts for Cache-conscious Factorization of DFT. *Proceedings of International Parallel and Distributed Processing Symposium 2000 (IPDPS 2000)*, April 2000.

[18] N. Park and V. K. Prasanna. Cache Conscious Walsh-Hadamard Transform. *International Conference on Acoustics, Speech, and Signal Processing 2001 (ICASSP 2001)*, May 2001.

[19] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. A Case for Intelligent DRAM: IRAM. *IEEE Micro*, April 1997.

[20] G. Rivera and C.-W. Tseng. Data Transformations for Eliminating Conflict Misses. *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'98)*, June 1998.

[21] G. Rivera and C.-W. Tseng. Locality Optimizations for Multi-Level Caches. *Proceedings of IEEE Supercomputing'99(SC'99)*, November 1999.

[22] V. Sarkar and G. R. Gao. Optimization of Array Accesses by Collective Loop Transformations. *the Proceedings of the 1991 International Conference of Supercomputing*, June 1991.

[23] A. Saulsbury, F. Dahgren, and P. Stenström. Receny-based TLB Preloading. *The 27th Annual International Symposium on Computer Architecture(ISCA)*, June 2000.

[24] H. Sharangpani. Intel Itanium Processor Microarchitecture Overview. *Microprocessor Forum*, October 1999.

[25] O. Temam, E. D. Granston, and W. Jalby. To Copy or Not to Copy: A Comile-Time Technique for Assessing When Data Copying Should be Used to Eliminate Cache Conflicts. *Proceedings of IEEE Supercomputing'93(SC'93)*, November 1993.

[26] R. C. Whaley and J. Dongarra. Automatically Tuned Linear Algebra Software (ATLAS). *Proceedings of SC'98*, November 1998.

[27] Q. Yi, V. Adve, and K. Kennedy. Transforming Loops to Recursion for Multi-Level Memory Hierarchies. *ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation (PLDI 2000)*, June 2000.