

Analysis of Memory Hierarchy Performance of 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 as a data transformation technique used in conjunction with tiling to improve cache performance. In this paper, we provide a theoretical analysis for the TLB and cache performance of block data layout. For standard matrix access patterns, we derive an asymptotic lower bound on the number of TLB misses for any data layout and show that block data layout achieves this bound. We show that block data layout improves TLB misses by a factor of $O(B)$ compared with conventional data layouts, where B is the block size of block data layout. This reduction contributes to the improvement in memory hierarchy performance. Using our TLB and cache analysis, we also discuss the impact of block size on the overall memory hierarchy performance. These results are validated through simulations and experiments on state-of-the-art platforms.

1. Introduction

The increasing gap between memory latency and processor speed is a critical bottleneck in achieving high performance. The gap is typically bridged through a multi-level memory hierarchy that can hide memory latency. The performance of this memory hierarchy system is severely impacted by the locality of data references. To improve memory hierarchy performance, compiler optimization techniques (e.g. loop permutation, fusion, and tiling) [13, 14, 21] have received considerable attention, which improve the locality of the data reference. These techniques, called *control transformations*, change the loop iteration order, thereby changing the data access pattern [4, 8, 19, 25]. Most

previous optimizations concentrate on single-level cache [8, 11, 15, 19, 23]. Multi-level caches in memory hierarchy were considered by a few researchers [20, 25]. However, most of these approaches target mainly the cache performance, paying less attention to the Translation Look-aside Buffer (TLB) performance. As the problem sizes become larger, the overall performance can drastically degrade because of TLB thrashing [22]. Hence, both TLB and cache must be considered in optimizing application performance. In [12], cache and TLB performance were considered *in concert*. In this analysis, TLB and cache were assumed to be fully-set associative. However, cache is direct mapped or small set-associative in most of state-of-the-art platforms.

Some recent work [11, 17, 18, 23] proposed *data transformations* that change the data layout in memory to match the data access pattern. It was proposed in [10] that both data and loop transformation can be applied to loop nests for optimizing cache locality. In [5, 6], a matrix is partitioned into small blocks of data. Data elements within one block are mapped onto contiguous memory. These blocks were laid out in memory by different space-filling curves. These data layouts have shown performance improvement over canonical row or column major layouts. Block data layout is one such layout where blocks are arranged in row-major order. ATLAS [2, 24] uses block data layout with tiling to exploit temporal and spatial locality. The combination of block data layout and tiling has shown high performance on various platforms. However, these results were confirmed through experiments; we are not aware of any formal analysis that addresses TLB performance.

In this paper, we study the impact of *block data layout*¹, with and without tiling, on the performance of both TLB and caches. First, we analyze the intrinsic TLB performance of block data layout. The TLB and cache performance for block data layout with tiling are analyzed. The block data

*Supported by the DARPA Data Intensive Systems Program under contract F33615-99-1-1483 monitored by Wright Patterson Air force Base, in part by NSF CCR-9900613, and in part by an equipment grant from Intel Corporation.

¹To avoid confusion, in this paper, ‘block’ is used in the context of a data transformation technique, e.g. block data layout. ‘tiling’ is used to represent a control transform technique.

layout with tiling shows better TLB performance compared with other state-of-the-art techniques like copying [11, 23] and padding [15, 19]. Simulations and experiments are conducted to verify this analysis.

Similar to the importance of tile size selection for tiling, appropriate block size selection for block data layout is critical to achieve high performance. In ATLAS, the selection of the optimal block size is done *empirically* at compile time by running several experiments with different block sizes [24]. The selection criteria does not have any supporting formal analysis. In [5, 6], it is observed that the block size should not be too small nor too large. However, no analytical bounds for block size were presented. In this paper, we propose an analytical bound for optimal block size in block data layout, on the basis of our TLB and cache analysis.

The contributions of this paper are as follows:

- We present a lower bound analysis for TLB performance. Further, we show that block data layout intrinsically has better TLB performance than canonical layouts (Section 2). Compared with row major layout, the number of TLB misses for block data layout is improved by $O(\sqrt{P_v})$ where P_v is the page size.
- We analyze the TLB and cache performance of tiling 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.
- On the basis of our cache and TLB analysis, we propose a block size selection algorithm that provides a tight analytical bound for block size (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 experiments on real platforms using matrix multiply, LU decomposition and Cholesky factorization (Section 4).

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 based as well as experimental results. Concluding remarks are presented in Section 5.

2. Block Data Layout and TLB Performance

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. With-

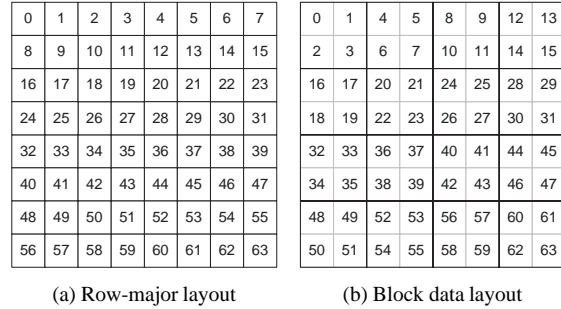


Figure 1. Various data layouts: block size 2×2 for (b)

out loss of generality, the canonical layout is assumed to be row major.

The following notations are used in this paper. P_v denotes virtual page size. S_{tlb} denotes the TLB entry capacity. In general, $S_{tlb} \ll P_v$. Block size is $B \times B$, where it is assumed $B^2 = kP_v$. Cache is assumed to be direct-mapped. S_{ci} is the size of the i^{th} level cache. Its line size is denoted as L_{ci} . We assume that TLB is fully set-associative and Least-Recently-Used(LRU) replacement policy is used.

2.1. Block Data Layout

To support multi-dimensional array representations in current programming languages, the default data layout is *row-major* or *column-major*, denoted as canonical layouts [7]. 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. 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. Figure 1 shows block data layout with block size 2×2 .

2.2. TLB Performance of Block Data Layout

In this subsection, we present a lower bound on the TLB misses for any data layout. We discuss the intrinsic TLB performance of block data layout. We present an analysis on the TLB performance of block data layout and show that its performance is improved when compared with conventional layouts. Throughout this paper, we consider an $N \times N$ array. Also it is assumed that N is large enough that $N \geq P_v \gg S_{tlb}$.

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, which are called *generic access pattern*² in this paper. 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 then all the columns is as follows.

Theorem 2.1 *For accessing an array along all the rows and then along all the columns, the asymptotic³ 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 $A_k = \{i \mid \text{at least one element of row } i \text{ is in page } k\}$. Similarly, let $B_k = \{j \mid \text{at least one element of column } j \text{ is in page } k\}$. 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 ($a_k + b_k \geq 2\sqrt{a_k \times b_k} \geq 2\sqrt{P_v}$), 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

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

As the problem size (N) increases, the number of pages accessed along a 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

²In the rest of this paper, we refer to the access pattern of all rows and all columns as generic access pattern

³This asymptotic [9] bound holds true when N is large. Also, the impact of S_{tlb} becomes negligible when N is large and hence does not appear in the bound.

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. 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.*

Detailed proof for this theorem can be found in [16]. 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.

This 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.*

Even though block data layout has better TLB performance compared with canonical layouts for generic access patterns, 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 performance improvement of TLB and caches when block data layout is used in conjunction with tiling.

```

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

3. Performance Analysis of Block Data Layout with Tiling

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 $Z = XY$. For large problems, its performance can suffer from severe cache and TLB thrashing. To reduce cache and TLB misses, tiling transforms the matrix multiplication to a 5-loop nest tiled matrix multiplication (TMM) as shown in Figure 2(a). To efficiently utilize block data layout, we consider a 6-loop TMM as shown in Figure 2(b).

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 whole matrix is accessed first along the rows then along the columns, in a tiled pattern. The 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. 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).

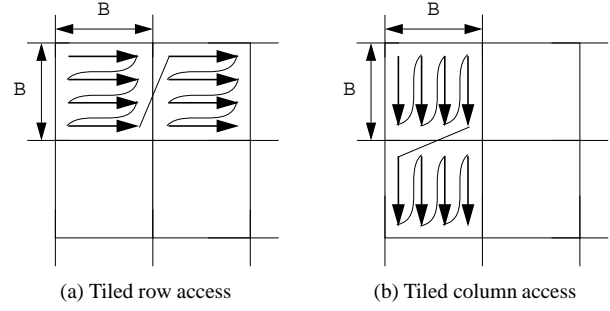


Figure 3. Tiled accesses

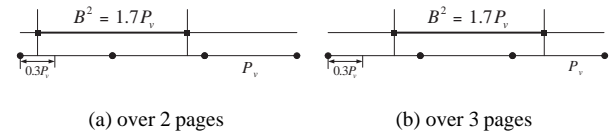


Figure 4. Blocks extending over page boundaries

The total number of TLB misses are further reduced when block data layout is used in concert with tiling⁴. This is formally stated in Theorem 3.1. To analyze TLB misses for tiled accesses using block data layout, we need to know the number of pages that a block of data is mapped onto. This is stated in Lemma 3.1.

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 that one block of data is mapped onto is $k + 1$.

Proof: For block size kP_v , assume that $k = n + f$, where n is a non-negative integer and $0 \leq f < 1$. An illustrative example of a block extending over page boundaries is shown in Figure 4. 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: $(1 - f) \times (n + 1) + f \times (n + 2) = k + 1$. \odot

Theorem 3.1 Assume that an $N \times N$ array is stored using block data layout. For tiled access along the rows and then the columns, 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

⁴Throughout 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 the block data layout.

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 $T_{col} = (k + 1) \times \frac{N}{B} \times \frac{N}{B} = (k + 1) \frac{N^2}{kB}$. Therefore, the total TLB misses caused by all row and all column accesses is $T_{miss} = (2 + \frac{1}{k}) \frac{N^2}{P_v}$. \odot

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 $\frac{\sqrt{kP_v} + \sqrt{k}}{2k+1} = \frac{B + \sqrt{k}}{2k+1}$.

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 misses (TM) is

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

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

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 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 as shown in Figure 5 (a). This degrades the overall performance.

We can reorganize a canonical layout to a block layout for tiled computations. Then as shown in Figure 5 (b), a self interference miss does not occur since all elements in a block can be mapped into contiguous locations in cache without any conflict.

In general, cache miss analysis for direct mapped cache with canonical layout is complicated because the self-in-

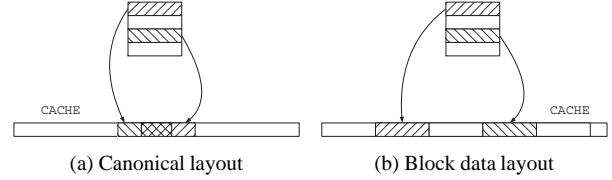


Figure 5. Example of conflict misses

terference misses cannot be quantified easily. Cache performance analysis of tiled algorithm was discussed in [11]. 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. Also, self-interference misses can be easily quantified when block data layout is used. According to these, we have derived the total number of cache misses for 6-loop TMM with block data layout. Detailed proof can be found in [16]. For i^{th} level cache with line size L_{ci} and cache size S_{ci} , the total number of cache misses (CM_i) is:

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

3.3. Block Size Selection

To achieve high performance, it is significant to select the block size of block data layout. In this section, we describe an approach for selecting the block size. 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:

$$MC = TM \cdot M_{tlb} + \sum_{i=1}^l CM_i H_{i+1} \quad (4)$$

where MC denotes the total miss cost, CM_i is the number of misses in the i^{th} level cache, TM is the number of TLB misses, H_i is the cost of a hit in the i^{th} level cache, and M_{tlb} is the cost 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 ($CM_{l+1} = 0$).

For a simple 2-level memory hierarchy that consists of only one level cache and TLB, the total miss cost (denoted

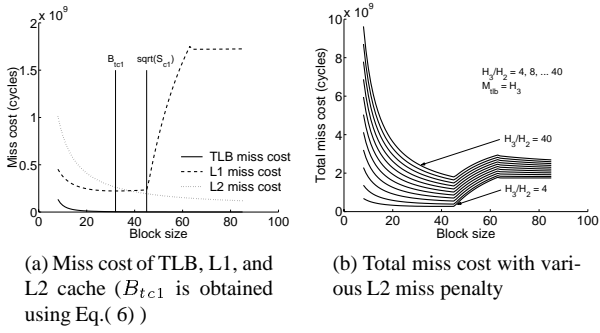


Figure 6. Miss cost estimation for 6-loop TMM (UltraSparc II parameters)

as MC_{tc1} in Eq. (4) reduces to:

$$MC_{tc1} = TM \cdot M_{tlb} + CM \cdot H_2, \quad (5)$$

where H_2 is the access cost of main memory. In the above estimation, M_{tlb} and CM are substituted with Eq.(2) and Eq.(3), respectively. Using the derivative of MC_{tc1} , the optimal block size, B_{tc1} , which minimizes the total miss cost caused by L1 cache and TLB misses is given as

$$B_{tc1} \approx \sqrt{\frac{\left(\frac{2L_{c1}M_{tlb}}{P_v} + \left[2 + \frac{3L_{c1} + 2L_{c1}^2}{S_{c1}}\right] H_2\right) S_{c1}}{4H_2}}. \quad (6)$$

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. Figures 6 (a) and (b) show the miss cost estimated through Eqs.(2) and (3). Figure 6(a) represents the individual cost of TLB, L1, and L2 miss, using UltraSparc II parameters. Figure 6(b) shows the change of estimated total miss costs based on different ratios of L1 cache miss penalty (H_2) and L2 cache miss penalty (H_3). Using Eq.(6), 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: According to the derivatives, $\frac{dMC_{tc1}}{dB} < 0$ and $\frac{dCM_2}{dB} < 0$ for $B < B_{tc1}$, TLB, L1, and L2 miss costs increase as block size decreases. This is shown in Figure 6(a), thereby increasing the total miss cost. Therefore, the optimal block size cannot be in the range $B < B_{tc1}$. \odot

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

Proof: In the range $B > \sqrt{S_{c1}}$, 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 6(a). For $\sqrt{S_{c1}} \leq B < \sqrt{2S_{c1}}$, although the number of L2 cache misses decreases ($\frac{dCM_2}{dB} < 0$), the ratio of derivatives of Eq.(3) for L1 and L2 misses is as follows:

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

Therefore, the total miss cost increases for $\sqrt{S_{c1}} \leq B < \sqrt{2S_{c1}}$. For $B \geq \sqrt{2S_{c1}}$, there is no reuse in L1 cache. Thus, the L1 cache miss cost saturates. As shown in Figure 6(b), $TM(B) > TM(\sqrt{S_{c1}})$ for $B \geq \sqrt{2S_{c1}}$, because L1 miss cost is dominantly larger than L2 miss cost and TLB miss cost for $B \geq \sqrt{2S_{c1}}$. Therefore, the optimal block size cannot be in the range $B > \sqrt{S_{c1}}$. \odot

Detailed proof of Lemma 3.3 can be found in [16].

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 $B_{tc1} \leq B_{opt} < \sqrt{S_{c1}}$. We select a block size that is a multiple of L_{c1} (L1 cache line size) in this range. \odot

4 Experimental Results

To verify our TLB performance analysis, simulations for the generic access pattern (accessing along all rows and then all columns) were performed. Furthermore, three applications (matrix multiplication, LU decomposition, and Cholesky factorization) are tested through simulations and executions on real platforms to confirm our analysis.

4.1 Simulations of generic access pattern

To verify our TLB performance analysis, simulations were performed using the SimpleScalar simulator [3]. 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. A 32×32 block size is considered for block data layout.

Table 1 compares the TLB misses of block data layout with canonical layout when the matrix is accessed with a generic access pattern. Table 1 (a) shows the TLB misses for accesses along all rows and then all columns. For small problem sizes, TLB misses with block data layout are considerably less than those with canonical layout. For problem size 1024×1024 , TLB entries used in a column(row) access are almost fully reused in the next column(row) access, thereby $O(S_{tlb})$ in Eq.(1) becoming relatively large.

Table 1. Comparison of TLB misses

Layout	1024	2048	4096
Block Layout	2081	81794	1196033
Canonical Layout	1049601	4198401	16793601

(a) Along all rows and then all columns

Layout	1024	2048	4096
Block Layout	64140	273482	1080986
Canonical Layout	1053606	4208690	16822675

(b) Arbitrary permutation of row and column accesses

Layout	1024	2048	4096
Block Layout	64501	274473	1080465
Canonical Layout	1053713	4208681	16822395

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

The number of TLB misses using block data layout is 504.37 times less than that using canonical layout. It is also less than the lower bound obtained from Theorem 2.1. For larger problem sizes, $O(S_{tlb})$ in Eq.(1) becomes negligible, since the TLB entries cannot be reused. Hence the total number of TLB misses approaches the lower bound. As shown in Table 1 (a), TLB misses with block data layout are upto 16 times less compared with canonical layout. Table 1 (b) and (c) confirm Corollary 2.1 and 2.2. 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.

Table 2 shows simulation results for tiled row and column accesses. Block size is set to be the same as the tile size. As shown in Table 2, the number of TLB misses conform our analysis from Theorem 3.1. The number of TLB misses with block data layout is 91% less than that with canonical layout.

4.2 Experimental results for various applications

To show the effect of block data layout, we performed simulations and experiments on the following applications: tiled matrix multiplication(TMM), LU decomposition, and Cholesky factorization(CF). The performance of tiling with

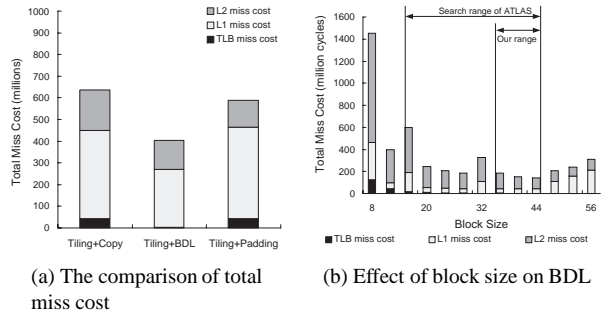
Table 2. TLB misses for all tiled row accesses followed by all tiled column accesses

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

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 (in tiling) is chosen to be the same as the block size in block data layout. Initial and final data layouts are canonical layouts. All the costs in performing data layout transformations (from canonical layout to block data layout and vice versa) are included in the reported results. As stated in [11], 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.

4.2.1 Simulation results

To show the performance of TLB and caches using tiling+BDL, simulations were performed using the SimpleScalar simulator [3]. The problem size was 1024×1024 .

**Figure 8. Total miss cost for TMM using UltraSparc II parameters**

Figures 7 and 8 show the TMM simulation results, based on UltraSparc II parameters. As shown in Figure 7(a), Tiling+BDL reduced 91–96% of TLB misses. This confirms our analysis presented in Section 3.1. Figure 8 shows the total miss cost (calculated from Eq. (4)) for TMM. L1, L2, and TLB miss penalties were assumed to be 6, 24, and 30 cycles, respectively. Figure 8(a) shows the comparison of the total miss cost of tiling+BDL with that of tiling+copying and tiling+padding. The comparison shows that tiling+BDL results in the smallest total miss cost. Specifically, the TLB miss cost of tiling+BDL is negligible compared with L1 and L2 miss costs. Figure 8(b) shows the effect of block size on the total miss cost for TMM using tiling+BDL. As discussed in Section 3.3, $B_{tc1} = 32.2$, $\sqrt{S_{c1}} = 45.3$, and $L_{c1} = 4$ using this architecture parameters. Theorem 3.2 suggests the range for optimal block size to be 36–44. Simulation results show that the optimal block size for this architecture was 44.

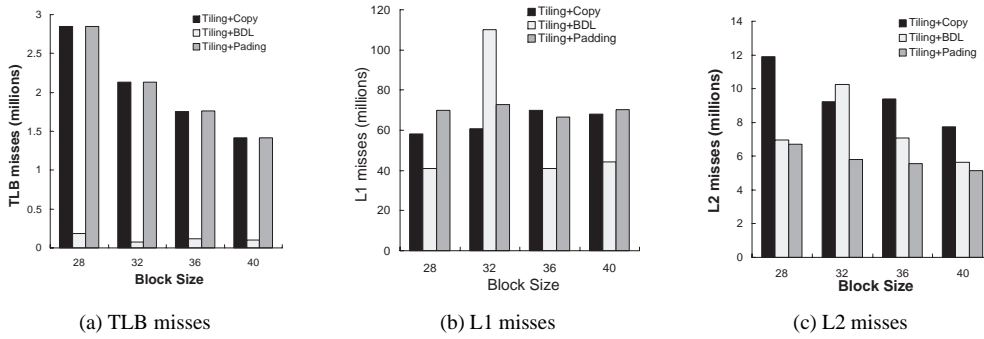


Figure 7. Simulation results for TMM using UltraSparc II parameters

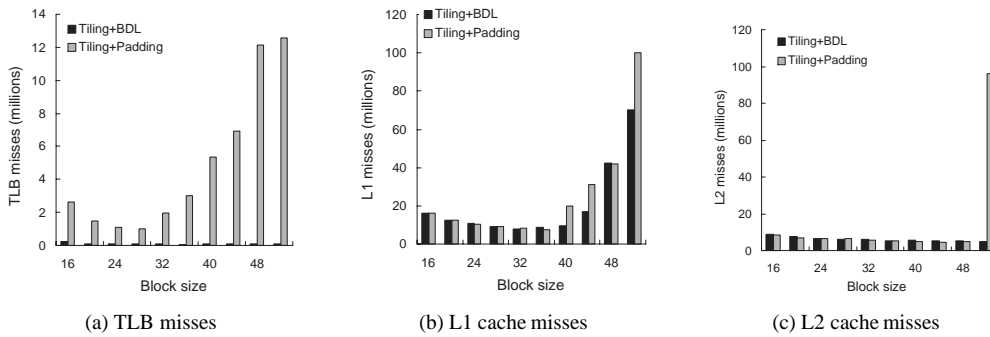


Figure 9. Simulation results for LU using Pentium III parameters

As shown in Figure 8(b), our proposed range is much tighter than the search range of ATLAS.

Figure 9 and 10 present 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 9(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 [15] suggested a pad size of 0. There is essentially no padding effect, thereby drastically increasing L1 and L2 cache misses. Figure 10 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.

4.2.2 Application execution results on real platforms

To verify our block size selection and the performance improvements using block data layout, we

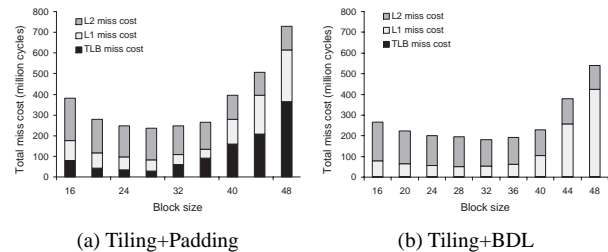


Figure 10. Effect of block size on LU decomposition using Pentium III parameters

performed experiments on several platforms. The parameters are 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()`. The problem sizes ranged from 1000×1000 to 1600×1600 .

The experimental results of TMM using tiling+BDL on UltraSparc II is shown in Fig. 11. Fig. 11(a) shows the best block size for TMM with respect to different problem sizes. For each problem size, we performed experiments by test-

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

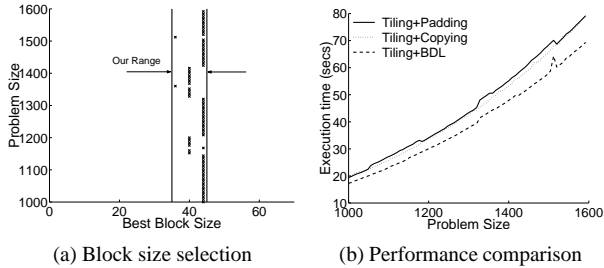


Figure 11. Experimental results for TMM on UltraSPARC II

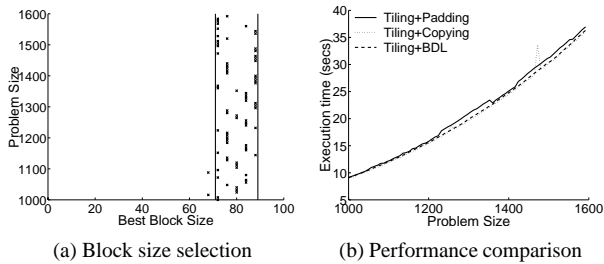


Figure 12. Experimental results for TMM on Alpha 21264

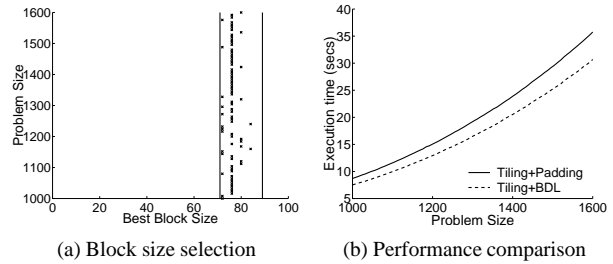


Figure 13. Experimental results for LU on UltraSPARC III

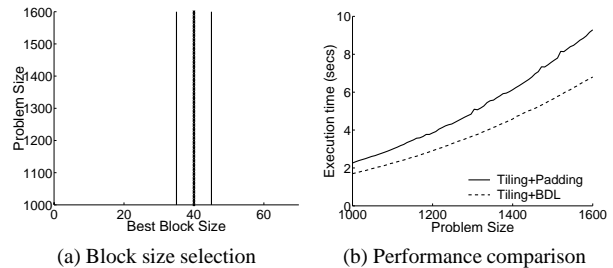


Figure 14. Experimental results for Cholesky factorization on Pentium III

ing block sizes ranging from 8–80. In all these tests, we found that the optimal block size for each problem size was in the range given by Theorem 3.2. This is shown in Figure 11(a). We also tested ATLAS. 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 Theorem 3.2. These experiments confirm that our approach proposes a reasonably good range for block size selection. Figures 11(b) show the execution time comparison of tiling+BDL with tiling+copying and tiling+padding. Figure 12–14 show experimental results for 3 different applications on 3 different platforms. Tiling+BDL technique is faster than using other optimization techniques, for almost all problem sizes and on all the platforms. These results confirm our analysis. More experimental results are available in [16].

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 presented a lower bound on the number of TLB misses for any data layout and showed that block data layout achieves this bound. The number of TLB misses using tiling and block data layout were considerably reduced compared with copying or padding techniques. We showed that block data layout with tiling leads to improved overall memory hierarchy performance compared with other techniques. Further, we proposed a tight range for block size in ATLAS using our performance analysis. Our analysis was verified using simulations as well as actual execution 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.

References

- [1] ADVISOR Project. <http://advisor.usc.edu>.
- [2] Automatically Tuned Linear Algebra Software (ATLAS). <http://math-atlas.sourceforge.net/>.
- [3] 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.
- [4] J. Chame, M. Hall, and J. Shin. Compiler Transformations for Exploiting Bandwidth in PIM-Based Systems. *Proceedings of Solving the Memory Wall Workshop, held in conjunction with the ISCA 2000*, June 2000.
- [5] 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 ICS '99*, June 1999.
- [6] S. Chatterjee, A. R. Lebeck, P. K. Patnala, and M. Thottethodi. Recursive Array Layouts and Fast Parallel Matrix Multiplication. *Proceedings of the 11th ACM SPAA*, pages 222–371, June 1999.
- [7] M. Cierniak and W. Li. Unifying Data and Control Transformations for Distributed Shared-Memory Machines. *Proceedings of the SCM SIGPLAN PLDI 1995*, pages 205–217, June 1995.
- [8] S. Coleman and K. S. McKinley. Tile Size Selection Using Cache Organization and Data Layout. *Proceedings of the SIGPLAN PLDI 1995*, June 1995.
- [9] E. Horowitz, S. Sahni, and S. Rajasekaran. *Computer Algorithms in Pseudocode: The Human Dimension*. W. H. Freeman Press, 1998.
- [10] 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.
- [11] M. Lam, E. Rothberg, and M. E. Wolf. The Cache Performance and Optimizations of Blocked Algorithms. *Proceedings of ASPLOS-IV*, April 1991.
- [12] N. Mitchell, K. Högstedt, L. Carter, and J. Ferrante. Quantifying the Multi-Level Nature of Tiling Interactions. *International Journal of Parallel Programming*, 1998.
- [13] D. Padua. Outline of a Roadmap for Compiler Technology. *IEEE Computing in Science & Engineering*, Fall 1996.
- [14] D. Padua. The Fortran I Compiler. *IEEE Computing in Science & Engineering*, January/February 2000.
- [15] 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), February 1999.
- [16] N. Park, B. Hong, and V. K. Prasanna. Tiling, Block Data Layout, and Memory Hierarchy Performance. Technical Report USC-CENG 01-05, Department of Electrical Engineering, USC, September 2001.
- [17] N. Park, D. Kang, K. Bondalapati, and V. K. Prasanna. Dynamic Data Layouts for Cache-conscious Factorization of DFT. *Proceedings of IPDPS 2000*, April 2000.
- [18] N. Park and V. K. Prasanna. Cache Conscious Walsh-Hadamard Transform. *ICASSP 2001*, May 2001.
- [19] G. Rivera and C.-W. Tseng. Data Transformations for Eliminating Conflict Misses. *ACM SIGPLAN PLDI 1998*, June 1998.
- [20] G. Rivera and C.-W. Tseng. Locality Optimizations for Multi-Level Caches. *Proceedings of IEEE SC'99*, November 1999.
- [21] J. Sanchez, A. Gonzalez, and M. Valero. Static Locality Analysis for Cache Management. *PACT 1997*, November 1997.
- [22] A. Saulsbury, F. Dahgren, and P. Stenström. Recency-based TLB Preloading. *ISCA 2000*, June 2000.
- [23] 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 SC'93*, November 1993.
- [24] R. C. Whaley and J. Dongarra. Automatically Tuned Linear Algebra Software (ATLAS). *Proceedings of SC'98*, November 1998.
- [25] Q. Yi, V. Adve, and K. Kennedy. Transforming Loops to Recursion for Multi-Level Memory Hierarchies. *ACM SIGPLAN PLDI 2000*, June 2000.