A Latency Tolerance Study

Overview of latency tolerance techniques for multiprocessors

Special case:
Tiling, Block Data Layout, and Memory Hierarchy Performance
--- N. Park et al TPDS 2003 ---

Nicolas A. Karakatsanis

Introduction to Latency Problem

- Latency of memory accesses increase by a factor of 5 per decade
  - Speed of microprocessors increase by a factor of 10 per decade
  - Speed of commodity memories increase by a factor of 2 per decade
- Multiprocessors greatly exacerbate the problem
  - Bus-based systems
    - latency increased by snooping
  - Distributed-memory systems
    - extra latency of network, network interface and endpoint processing
- Caches are not a panacea
  - help reduce the frequency of high-latency events
  - do not reduce inherent communication
  - cache miss rate is not negligible
- Latency grows with the size of a machine (number of nodes)
  - more communication relative to computation
  - more hops in the network gor general communication
  - likely more contention

Introduction to Latency

- Deal effectively with
  - bandwidth and latency related problem
- while
  - maintaining a convenient programming model
- Solution
  - Bandwidth can be improved by
    - throwing more hardware at the problem (wider links, rich topologies)
  - Latency is more fundamental limitation

Introduction to Latency

- Effective Latency reduction (multiprocessors)
  - reduction of access time to each level of the extended memory hierarchy (system responsibility)
    - tight interfaces (e.g. processor-cache, network interface)
    - quick act of cache controller when cache miss happens
    - appropriate network design
  - structure the system to reduce the frequency of high-latency accesses (system responsibility)
    - automatic replication in caches keeps most important data close to processor when it needs them (based on temporal and spatial locality)
  - structure the application to reduce the frequency of high-latency accesses
    - decomposition and assignment of computation to processors
    - structure access patterns to increase spatial and temporal locality (tiling)

Latency tolerance

- Previous latency reduction techniques often do not suffice
- Why not tolerate the remaining latency
  - hide the latency from the processor by
    - overlapping it with computation or other high-latency events
- Latency tolerance in multiprogramming for uniprocessors
  - deal with high latency of disk access by switching (through OS) to another process of different application
  - (latency of disk accesses) >> time to switch process through OS
    - => worthwhile technique
  - the execution time of its process is NOT reduced
  - better throughput and utilization of system
- Latency tolerance examined in this study different in two points
  - overlap latency with work from the same application
  - deal with memory or communication latencies, not disk accesses

Latency tolerance

- Latency
  - includes all time components from issue by processor until completion
- Memory latency
  - if cache miss is satisfied locally
- Communication latency
  - if cache miss is satisfied remotelly
  - includes the processor overhead, assist occupancy, transit delay, bandwidth-related costs, contention, protocol-based transfer (invalidations, acknowledgements)
  - one-way or round-trip
- synchronization latency
  - duration starting when processor issues a synchronization operation (e.g. lock or barrier) until it gets past that operation
  - therefore includes
    - time to access the synchronization variable
    - time spent waiting for the completion of an event it depends on
- instruction latency
  - duration starting from the issue of an instruction and ending with its completion at the pipeline (assuming no other latencies)
- Focus on tolerating communication latencies
  - some latency techniques derive from and can be applied to local memory latencies (uniprocessors)

Communication Latency

- Message
  - Communication triggered by one node to another by a single user operation
- Sender-initiated communication
  - initiated by the process that has produced or currently holds the data withour receiver solicitation (send operation at MPI)
- Receiver-initiated communication
  - initiated by the process that obtains the data (read miss of non-local data in a shared-address space)

Approaches to tolerate (communication) latency

- From the viewpoint of a processor
  - node-to-node communication architecture is a pipeline
- Latency tolerance offers better utilization of pipeline by overlapping resources
  - within communication pipeline (multiple words at a time)
  - communication-to-communication
  - computation-to-communication
- Four key approaches to tolerate latency
  - block data transfer (usage of long cache blocks for UP)
  - precommunication (prefetching for UP)
  - proceeding past communication in the same thread
  - multithreading

Block data transfer

- make individual messages larger (more than a word)
  - can be pipelined (processor sees only the latency of the first word)
  - communication-to-communication overlap is better exploited
  - amortizes the per-message endpoint overhead over large amount of sent data
  - amortizes the per-packet routing and header information
  - a large message may require one single acknowledgement rather than one per word
  - similar to using long cache block at uniprocessors
  - Necessary but not adequate
    - does not keep the processor or the communication assist busy while message is i progress
    - does not keep other network paths busy

Precommunication

- complementary to block data transfer
- generates communication before the point where the operation naturally appears
  - => generated communication (partially or entirely) completed before data is actually needed
- implemented either in
  - software (inserting precommunication operation earlier in code)
  - hardware (detecting the opportunity and pre-issuing the operation
- ideally suitable for receiver-initiated communication
  - move up the sender-initiated communication into the code
  - => data may reach the receiver before it is needed

Proceeding past Communication in the same thread

- complementary to block data transfer
- the communication operation may be generated at its natural position in the program
- processor is allowed to proceed past it and find other *independent* operation or communication in the same process or thread
- sender-initiated communication
  - generally used more effectively with sender-initiated communication
  - transactions immediately following communication operation are independent
- receiver-initiated data
  - difficult to use because after such operation the data transferred are being used, so the immediate operations are dependent
  - possible to use for receiver-initiated communication if
    - delay the actual use of the dependent data (push down)
    - find independent work beyond the dependent data

Multithreading

- complementary to block data transfer
- similar to proceed-past technique
- independent work is found on a different thread (mapped to run on the same processor)
- most general technique (compared to precommunication and past-proceed)
  - efficient regardless of the nature of communication operations (sender- or receiver-initiated)
- requires extra parallelism to be explicit in the form of additional threads at each processor

Further structuring of the application to tolerate latency

- Principal strategies for improving the performance of memory hierarchy (uniprocessors)
  - stride-one access
    - program loops should access contiguous data items
    - most cache systems are organized into blocks that contain multiple data items, which are at contiguous memory addresses
    - if loops iterates over successive items in memory (depending on data layout)
      - => can suffer at most one cache miss per cache block
  - tiling (blocking)
    - ensuring that data remains in cache between subsequent accesses to the same memory locations
      - i.e. when data are there, use them at the most before they are removed

Further structuring of the application to tolerate latency (...continue)

- Principal strategies for improving the performance of memory hierarchy (uniprocessors) (...continue)
  - Data reorganization
    - reorganize data structures so that data items used together are stored together in memory
  - Use of optimal data layout
    - default data layouts are either raw- or column-major
    - block data layout or morton data layout are promising
  - Combination of methods
    - example: tiling + padding

Further structuring of the application to tolerate latency (...continue)

- Principal strategies for improving the performance of memory hierarchy (multiprocessors)
  - synchronization
    - common problem
      - have several different processors updating the same shared data structure
    - locking mechanisms are essential
      - ensure read-exclusive mode (when one updates takes place, all other processors updates to the same structure are locked out
  - elimination of false sharing
    - common problem
      - have several different processors accessing distinct data items of the same cache block
      - cache block ping-pongs back and forth between those processor caches (false-sharing)
    - ensure that data used by different processors reside on different blocks
      - use of padding in data
        - inserting empty bytes in a data structure to ensure different elements are in different cache blocks

Introduction

Tiling, Block Data Layout, and Memory Hierarchy Performance

N. Park, B. Hong, V.R. Prasanna

Transactions on Parallel and Distributing Systems 2003

Introduction

- Targeted Problem
  - Increasing gap between
    - memory latency and processor speed
  - Critical bottleneck in performance
- Possible Solutions examined in this study
  - Latency reduction
  - Latency tolerance
- Howto
  - multilevel memory hierarchy (multilevel caches)
  - hardware solutions
  - software optimization methods

Hardware-based proposed methods

- Modern Processors (e.g. Intel Merced) provide
  - increased programmer control over data placement
  - movement in a cache-based memory hierarchy
  - memory streaming hardware support for media applications.
- Important to understand effectiveness of control and data transformations.

Software optimization methods

- Memory hierarchy gets deeper
  - => critical to efficiently manage the data
  - => improve data access performance
- Possible solution
  - Compiler optimization techniques
    - Tiling
      - Description: ***Tiling transforms loop nests so that temporal locality can be better exploited for a given cache size***
      - Limitations: Tiling focuses only on the reduction of capacity cache misses by decreasing the working set size
      - Current situation: most state-of-the-art machines is either direct-mapped or small set-associative
      - Consequence: Thus, it suffers from considerable conflict misses, thereby degrading the overall performance

Software optimization methods (continue...)

- Conflict/self-interaction misses
  - reused data are accessed between reuse by the same variables
- How to eliminate conflict misses
  - copying
    - copy non-contiguous reused data to consecutive locations
    - thus, each word within the block is mapped to a unique cache location
  - padding
    - analyzes array subscripts in loop nests to compute a memory access pattern for each variable
    - iteratively increments the base address of each variable until no conflicts result with other variables analyzed
  - data layout transformations - block data layout
    - a matrix is partitioned into submatrices called blocks
    - Data elements within one such block are mapped onto contiguous memory
    - Blocks are arranged in row-major order

What about TLB performance?

- Most of the previous methods target at <u>cache</u> performance
  - Tiling + copying
  - Tiling + padding
- As problem sizes become larger
  - TLB (Translation Look-Aside Buffer) performance becomes more significant
  - TLB thrashing is possible
    - =>significant degradation of high performance
- Therefore
  - <u>**Both TLB and Cache performance should be considered**</u>

TLB in concert with cache at multi-level memory hierarchies

- Few studies on multi-level caches
- Multi-level memory hierarchy studies ignore TLB performance
- TLB in concert with cache performance for multi-level caches
- In this analysis
  - TLB and cache are assumed to be
    - fully-set associative.
- However, in most of the state-of-the-art platforms
  - the cache is direct or small set-associative

Data layout transformations studies so far

- Recent studies propose transforming the data layout to match the data access pattern:
  - data and loop transformation be applied to loop nests for optimizing cache locality
  - conventional (row or column-major) layout to be changed to a recursive data layout
    - Morton layout (matches the access pattern of recursive algorithms)
    - *Only experimental data so far, no formal analysis*
- ATLAS project
  - tiling + block data layout
  - first input data are transformed first to block data layout,
  - then tiling is applied
  - *empirical estimation of optimal block size*

Purpose of this work

- Study <u>Block Data Layout</u> as
  - *a data transformation method to improve memory hierarchy performance*
- Steps
  - Analyze the intrinsic TLB performance of block data layout.
  - Analyze the TLB and cache performance using tiling and block data layout.
  - Based on the analysis, propose a block size selection algorithm
  - Comparative evaluation with the alternative Morton Data Layout

Contributions of the study

- TLB performance and Block Data Layout
  - Present a lower bound analysis of TLB performance.
  - Show that block data layout intrinsically has better TLB performance than row-major layout
  - Cost of accessing all rows and all columns is analyzed.
  - Number of TLB misses is improved by $O(\sqrt{P_v})$ compared with row-major layout (P_v)

Contributions of the study (continue...)

- Tiling + Block Data Layout
  - TLB and cache performance analysis
  - Tiled matrix multiplication,
  - Block data layout improves the number of TLB misses by a factor of B (B = block size)
  - Cache performance improvement
  - Validation through simulations (Superscalar)
- Block size selection algorithm
  - On the basis of TLB and cache analysis
  - validation using ATLAS proposed range of block sizes
- Validation
  - Comparison of analysis results with simulations and measurements
    - Tiled Matrix Multiplication (TMM)
    - LU decomposition (LU)
    - Cholesky Factorization (CF)

Contributions of the study (continue...)

- Comparative evaluation between Block Data Layout (BDL) and Morton Data Layout (MDL)
  - Block size for MDL is limited
    - => If MDL Block size falls out of block optimal range
      - =>Then Performance Degradation
    - Example
      - Ultrasparc II, Pentium III
      - Cholesky Factorization and LU decomposition
        - =>up to 15.8% slower with MDL (*depending on the problem size*)

Assumptions and Notations

- Fixed architecture parameters
  - Cache size, Cache Line Size, page size, TLB entry capacity
- Notations
  - S_{tlb} denotes number of TLB entries
  - P_v denotes virtual page size
  - S_{ci} denotes the size of the *ith* cache
  - L_{ci} denotes the size of line cache of the *ith* cache
- Assumptions
  - TLB is fully set-associative with Least-Recently-Used (LRU) replacement policy
  - Block size is *B* x *B*, where $B^2 = kP_v$
  - Cache is direct-mapped

Block Data Layout

- Multidimensional array representations
  - Mapping functions
    - array index $\Rightarrow$ linear memory address
  - Current default data layouts
    - row-major or column-major, denoted as canonical layouts
  - Drawbacks of canonical layout
    - Example: large matrix stored in row-major layout
      - Due to large stride
        - =>column accesses can cause cache conflicts.
      - If every row in a matrix is larger than the size of a page
        - => column accesses can cause TLB trashing
        - => drastic performance degradation.

Block Data Layout

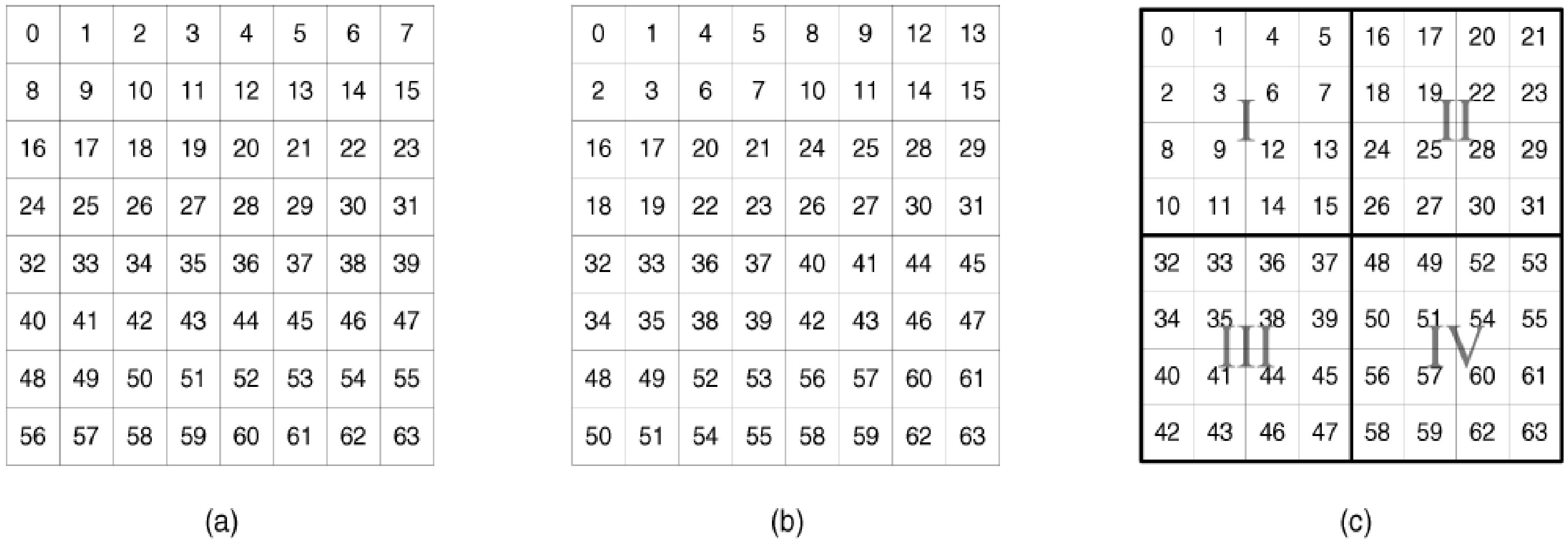
- Block data layout (BDL)
  - A large matrix is partitioned into submatrices.
  - Each submatrix is a B x B matrix
  - All elements in the submatrix are mapped onto contiguous memory locations
  - blocks are arranged in row-major order
- Morton Data Layout (MDL)
  - Variant of Block Data Layout
  - Divides the original matrix into four quadrants
  - Lays out these submatrices contiguously in the memory
    - Each of these submatrices is further recursively divided and laid out in the same way
      - End of recursion: Elements of the submatrix are stored contiguously
- Difference between BDL and MDL
  - Order of blocks

Data Layouts Comparison

Row-major layout (canonical)

Block data layout

Morton data layout



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)

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)

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)

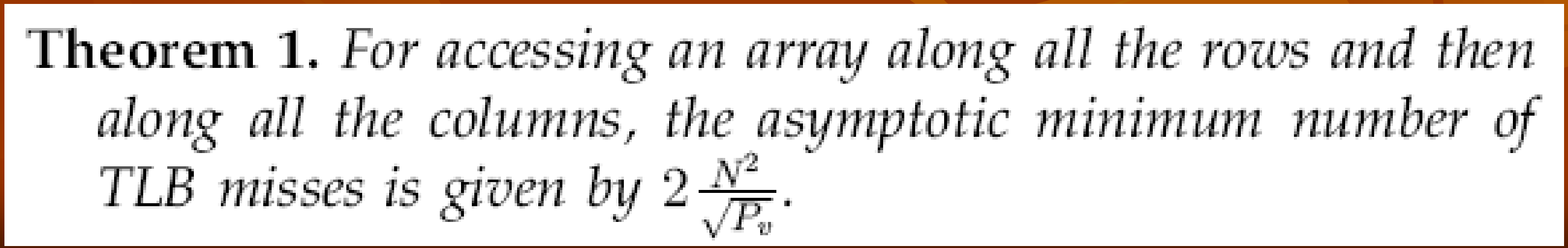
Fig. 1. Various data layouts: block size 2×2 for (b) and (c). (a) Row-major layout, (b) block data layout, and (c) Morton data layout.

TLB performance of BDL

- Presentation of a lower bound on TLB performance
- Discussion of intrinsic TLB performance of BDL
- Analysis of TLB performance of DBL
- Show improved performance of BDL compared with conventional layouts
- From now assume N x N arrays

Lower bound on TLB misses

- Usual matrix operations
  - row and column accesses, or
  - permutations of row and column accesses.
- Access pattern in this study
  - an array is accessed first along all rows and then along all columns.
- Lower bound analysis on TLB misses for this memory access pattern



Theorem 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 of theorem 1 – Lower bound analysis

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 $2\frac{N^2}{\sqrt{P_v}}$. □

Bottom line

The asymptotic minimum number of TLB misses is given by $2\frac{N^2}{\sqrt{P_v}}$

Corollary 1. *For accessing an array along an arbitrary permutation of row and column accesses, the asymptotic minimum number of TLB misses is given by* $2\frac{N^2}{\sqrt{P_v}}$.

TLB performance analysis (canonical layout)

- Memory access pattern
  - first all rows, then all columns
- N x N array, canonical (row-major layout)
- First pass (row accesses)
  - memory pages are accessed consecutively.
    - =>TLB misses (row accesses) = $\frac{N^2}{P_v}$
- Second pass (column accesses)
  - Elements along the column are assigned to N different pages
    - =>A single column access causes N TLB misses
    - Since $N \gg S_{tlb}$ all column accesses cause TLB misses = N^2
- Total number of TLB misses = $\frac{N^2}{P_v} + N^2$

Therefore in canonical layout, column access drastically increase total TLB misses

TLB performance analysis (Block Data Layout)

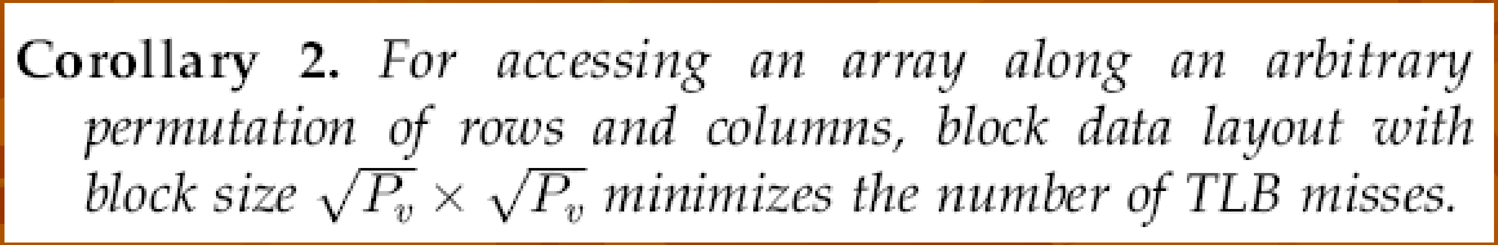
- Block Data Layout has better performance compared with canonical layout

Theorem 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 prove this theorem we consider two cases
  - $k \leq 1$
  - $k \geq 1$
- For each case, we estimate the number of TLB misses by comparing
  - TLB size
  - number of page entries in a row, and
  - number of pages in a block.
- The optimal block size is then derived from these estimations.
- Generally the number of TLB misses for a B x B block data layout is given by $k\frac{N^2}{B}+\frac{N^2}{B}$ It is reduces by a factor of $\frac{(P_v+1)B}{P_v(k+1)}(\approx \frac{B}{k+1})$ when compared with canonical layout
- When $B=\sqrt{P_v}$ the number of TLB misses approaches the lower bound

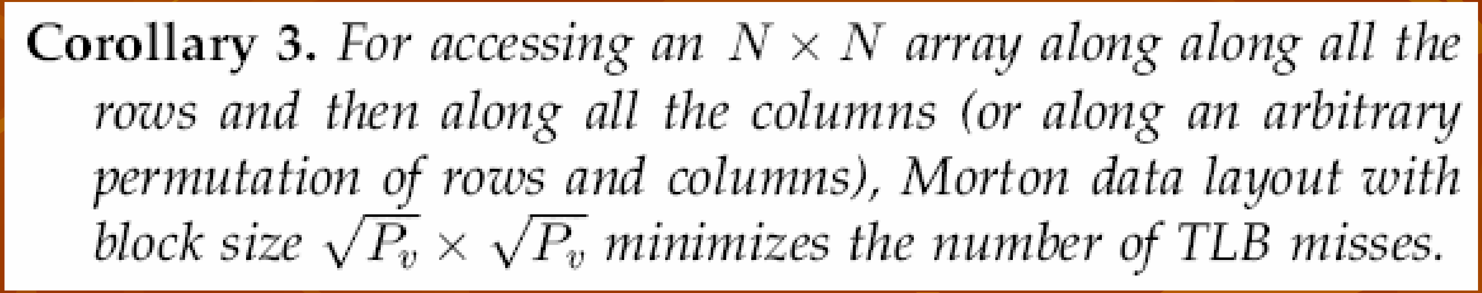
TLB performance analysis (Block Data Layout)

- Generalization for arbitrary memory access patterns



Corollary 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.*

- Number of TLB misses is minimized even when Morton Data Layout is used (providing the block size is well chosen



Corollary 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.*

Validation of TLB analysis

- Simulations were performed (SimpleScalar)
- Assumptions
  - page size is 8KBytes
  - TLB data is fully set-associative with 64 entries (Ultrasparc II)
  - Block factor B=32
  - Double precision data points
- Table 1 shows the number of TLB misses when using
  - canonical layout, block data layout and Morton data layout

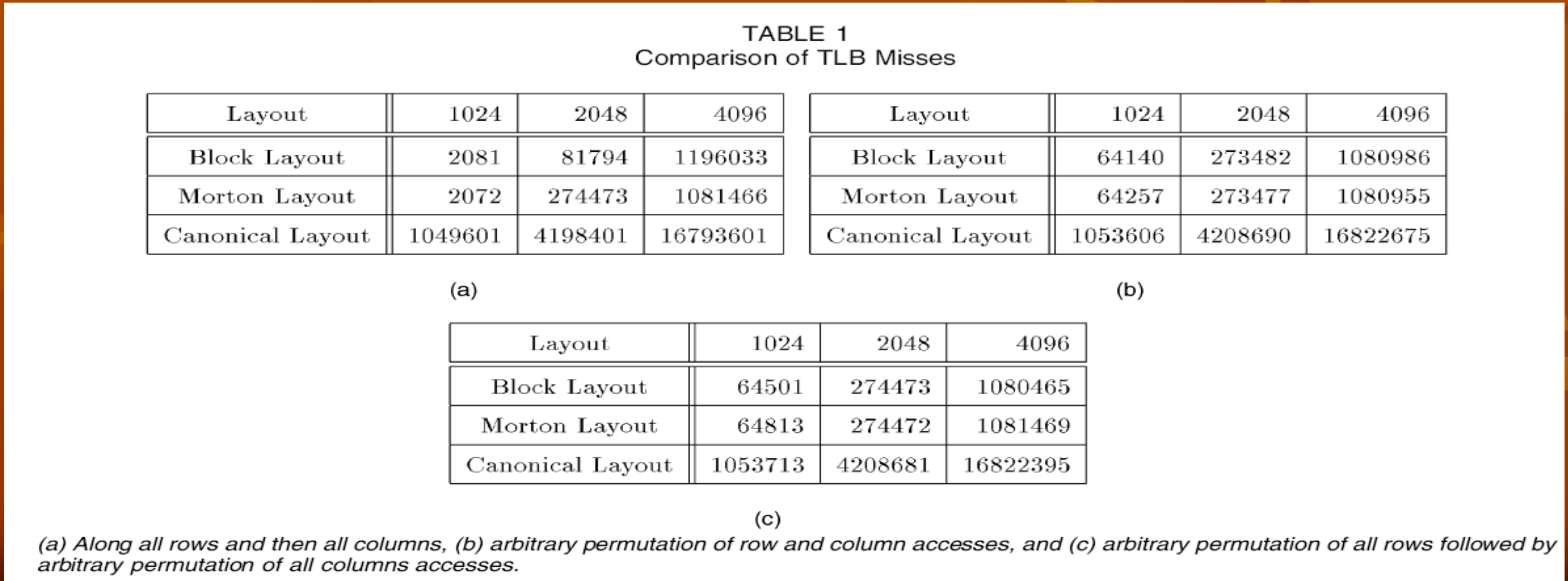


TABLE 1
Comparison of TLB Misses

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

(a)

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

(b)

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

(c)

(a) Along all rows and then all columns, (b) arbitrary permutation of row and column accesses, and (c) arbitrary permutation of all rows followed by arbitrary permutation of all columns accesses.

Tiling Optimization Method

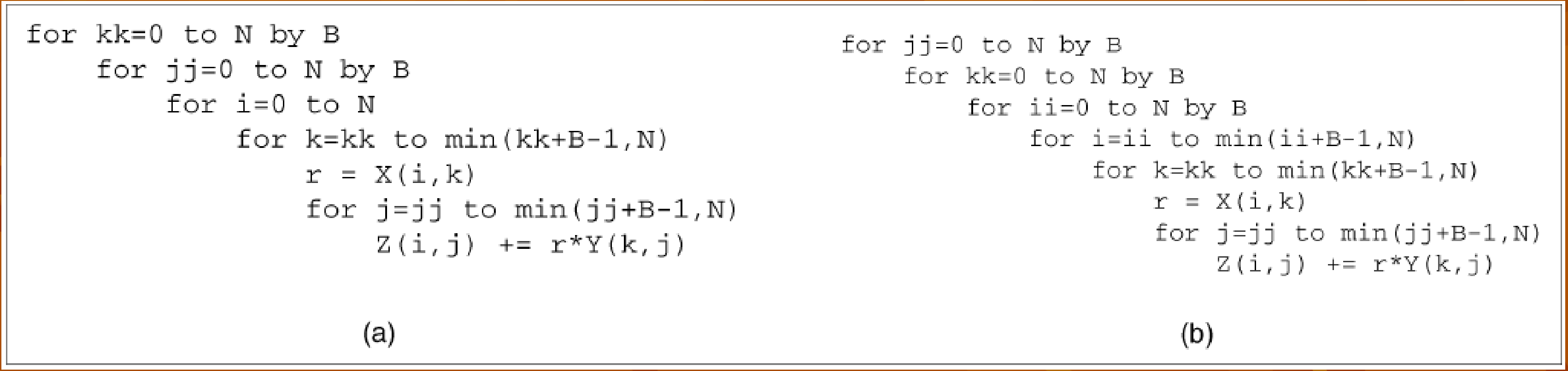
- Block data layout
  - has better TLB performance compared with canonical layouts with generic access pattern,
- However
  - it alone does not reduce cache misses.
- Proposed solution $\Longrightarrow$ Tiling
  - Well known optimization technique
  - improves cache performance
  - tiling + block data layout
    - improves TLB performance as well
    - The data access pattern of tiling matches well with block data layout.

Tiling Optimization Method

- Tiling
  - *compiler optimization method*
  - *effectively maps application structure to machine structure for full use of hardware benefits*
  - *transforms the loop nest so that temporal locality can be better exploited for a given cache size*
  - Consider an N x N matrix multiplication represented as Z = XY
    - usual 3-loop computation
      - working set size = $N^2 + 2N$
      - if problem size > cache size => cache thrashing (cache capacity misses), degradation of performance
    - 5-loop tiled computation
      - Transformation of the matrix multiplication to a 5-loop nest tiled matrix multiplication (TMM)
      - working set size = $B^2 + 2B$,where B=tile size
    - 6-loop tiled computation using block data layout
      - effective utilization of block data layout reduces working set size more
      - choice of optimal block size
      - set tile size to be equal with the block size

TLB performance (Tiling + canonical layout)

- Tiled Matrix Multiplication Example



```
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)
```

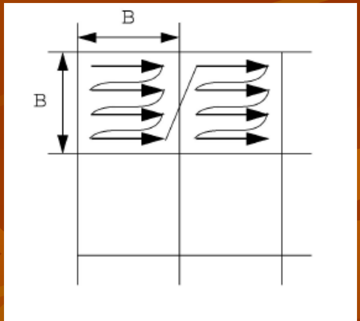
(a)

```
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)
```

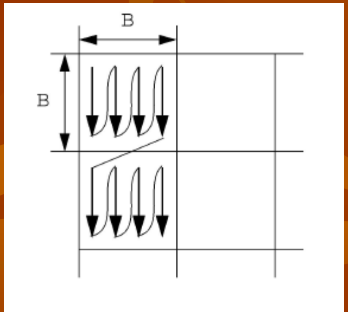
(b)

- Illustration of the effect of block data layout on tiling
  - consider a generic access pattern abstracted from tiled matrix operations. (Tile size=B)

- Tiling with Canonical layout,
  - TLB misses will not occur when accessing consecutive tiles in the same row, if $B \leq S_{tlb}$
    - =>TLB misses (row accesses) = $\frac{N^2}{P_v}$
  - At column accesses B page table entries are necessary to access each tile.
    - =>TLB misses (column accesses) = $B \times \frac{N}{B} \times \frac{N}{B} = \frac{N^2}{B}$
  - Total TLB misses $\frac{N^2}{P_v} + \frac{N^2}{B}$
    - reduced by a factor of B, compared without tiling



(a)



(b)

TLB performance (Tiling + BDL)

- The total number of TLB misses are further reduced when
  - block data layout is used in concert with tiling (shown in Theorem 3)
- Block size of block data layout is assumed to be the same as the tile size so that
  - tiled access pattern matches block data layout.
- In block data layout
  - a two-dimensional block is mapped onto one-dimensional contiguous memory locations
  - A block extends over several pages (e.g. $B^2 = 1.7P_v$)

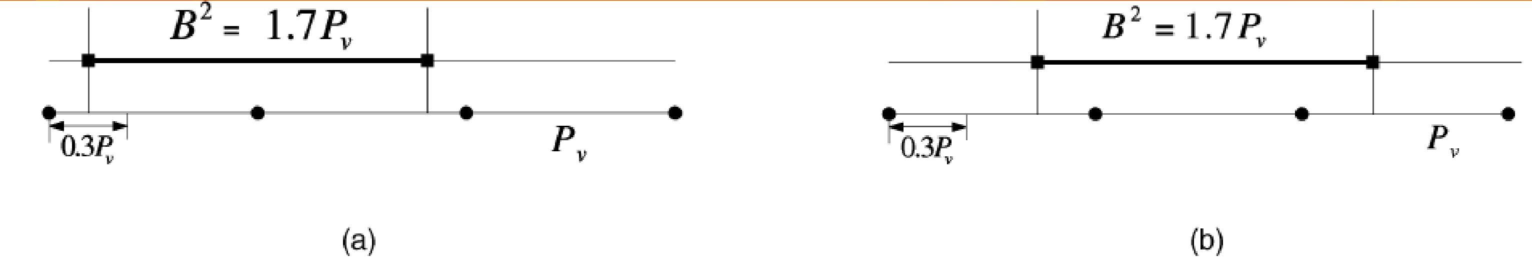
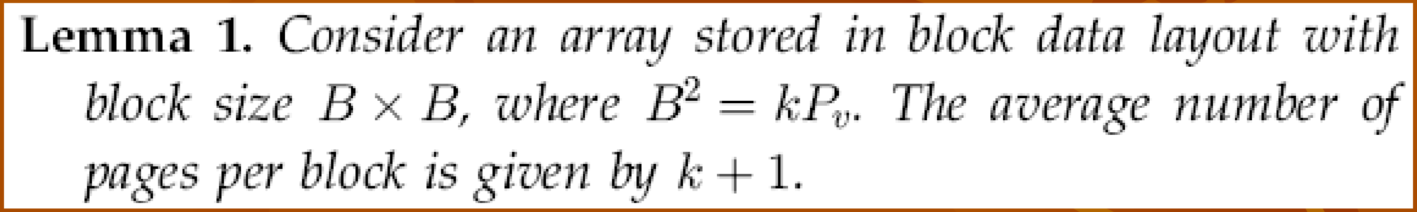


Fig. 4. Block extending over page boundaries. (a) Over two pages and (b) over three pages.

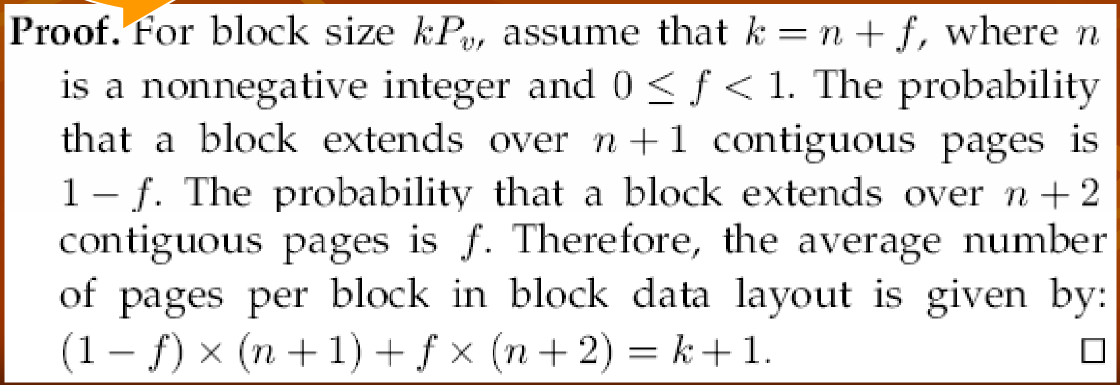
TLB performance (Tiling + BDL)

- Analysis of TLB misses for column accesses using block data layout, requires
  - the average number of pages in a block

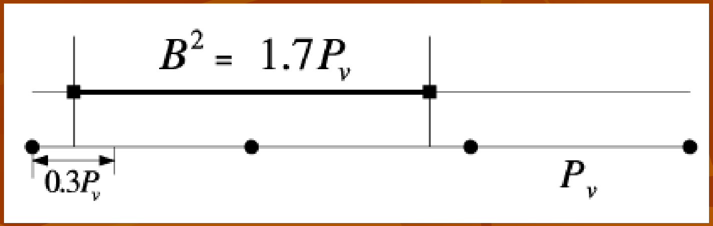


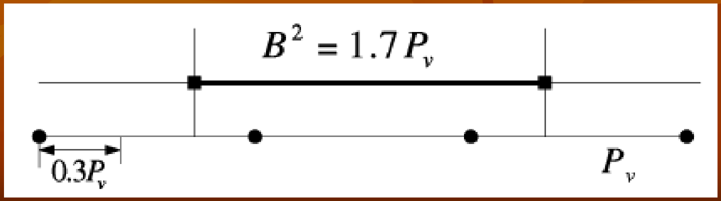
Lemma 1. *Consider an array stored in block data layout with block size $B \times B$, where $B^2 = kP_v$. The average number of pages per block is given by $k+1$.*

  - The proof of lemma 1 follows, based on the fig. of the previous slide



Proof. For block size kP_v, assume that $k = n + f$, where n is a nonnegative integer and $0 \leq f < 1$. 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: $(1-f) \times (n+1) + f \times (n+2) = k+1$. □





TLB performance (Tiling + BDL)

Theorem 3. *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 of theorem
  - Blocks in block data layout are arranged in row-major order
    - => a page overlaps between two consecutive blocks that are in the same row, continuously accessed
    - => TLB misses (tiled row accesses) = $\frac{N^2}{P_v}$ (min. TLB misses)
  - However, no page overlaps between two consecutive blocks in the same column.
    - => each block along the same column goes through *(k+1)* different pages (Lemma 1)
    - => TLB misses (tiled column accesses) = $T_{col} = (k+1) \times \frac{N}{B} \times \frac{N}{B} = (k+1)\frac{N^2}{kP_v}$
  - Total TLB misses(row and column accesses)
    - $T_{miss} = (2+\frac{1}{k})\frac{N^2}{P_v}$

TLB performance (Tiling + BDL)

- Number of TLB misses
  - using canonical layout $\Rightarrow$ $\frac{N^2}{P_v} + \frac{N^2}{B}$ where $B = \sqrt{kP_v}$
  - using block data layout $\Rightarrow$ $(2 + \frac{1}{k})\frac{N^2}{P_v}$
- Block data layout reduces the number of TLB misses by $\frac{\sqrt{kP_v}+\sqrt{k}}{2k+1} = \frac{B+\sqrt{k}}{2k+1}$
- Validation of analysis
  - simulations (SimpleScalar) for tiled row and column accesses (BDL)
    - block size = 32 x 32, block size = tile size

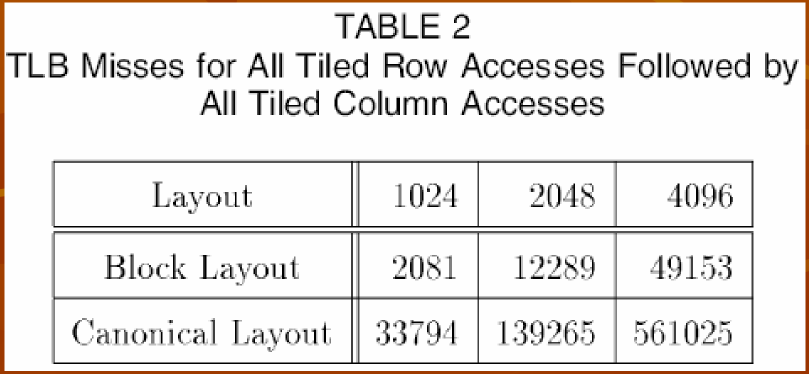


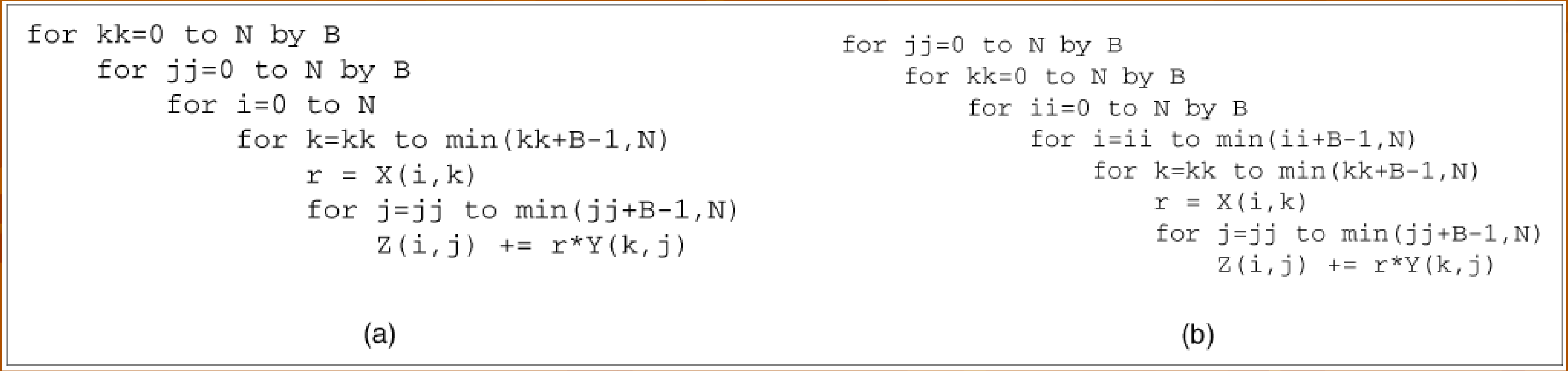
TABLE 2
TLB Misses for All Tiled Row Accesses Followed by All Tiled Column Accesses

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

    - number of TLB misses with block data layout is ***91 percent less*** than that with canonical layout
    - When problem size =1024 x 1024, TLB misses with block data layout is 2,081, close to the min. TLB misses (2,048)
      - special case: each block starts on a new page

TLB performance (real applications)

- Tiled Matrix Multiplication (TMM) $\Rightarrow$ Z=XY



```
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)

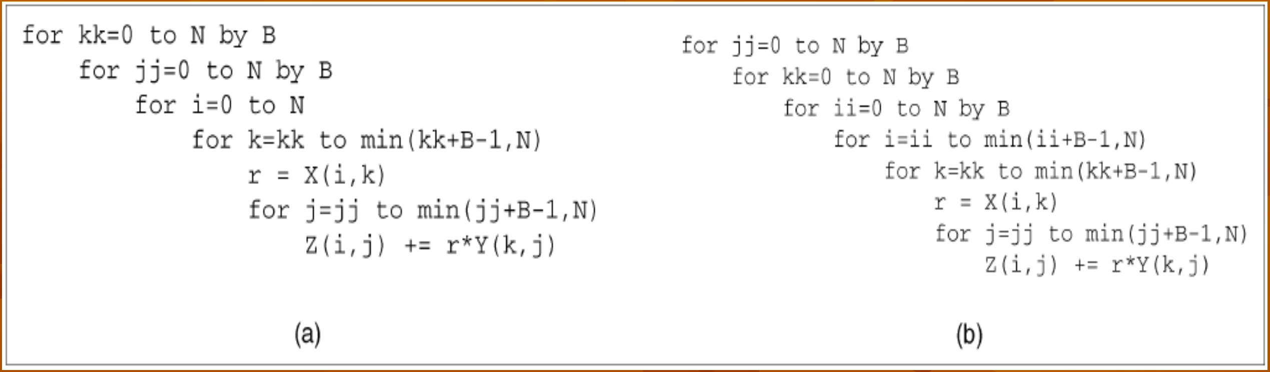
```
for jj=0 to N by B
     for kk=0 to N by B
          for ii=0 to N by B
               for i=ii to min(ii+B-1,N)
                    for k=kk to min(kk+B-1,N)
                         r = X(i,k)
                         for j=jj to min(jj+B-1,N)
                              Z(i,j) += r*Y(k,j)
```

(b)

- 5-loop TMM (tiling + canonical layout)
  - Array Y accessed in a tiled row pattern
  - Arrays X and Z accessed in a tiled column pattern
  - A tile of each array is used in the inner loops (*i,j,k*)
  - outer loops (*kk,jj*)
  - Number of TLB misses for each array
    - (average number of pages per tile) x (number of tiles accessed in the outer loops)
      - => avg number of pages/tile = $B + \frac{B^2}{P_v}$
      - => number of TLB misses = $2N^3\left(\frac{1}{B^2} + \frac{1}{BP_v}\right) + N^2\left(\frac{1}{B} + \frac{1}{P_v}\right)$

TLB performance (real applications)

- Tiled Matrix Multiplication (TMM) ⇒ Z=XY



```
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)

```
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)

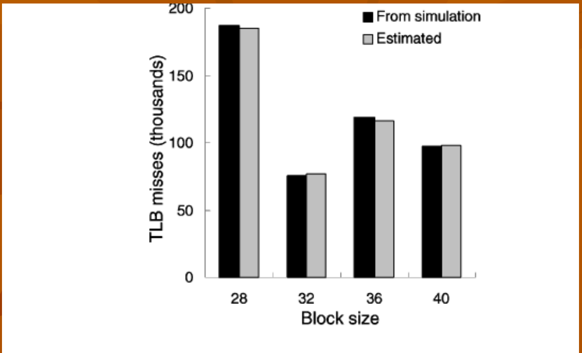


Fig. 5. Comparison of TLB misses from simulation and estimation.

- 6-loop TMM (tiling + blocked data layout)
    - Array Y accessed in a tiled row pattern
    - Arrays X and Z accessed in a tiled column pattern
    - A tile of each array is used in the inner loops (*i,j,k*)
    - Outer loops (*jj,kk,ii*)
    - Number of TLB misses for each array
        - (average number of pages per tile) x (number of tiles accessed in the outer loops)
            - => avg number of pages/tile = $\frac{B^2}{P_v} + 1 (= k+1)$ *(Lemma 1)*
            - => number of TLB misses

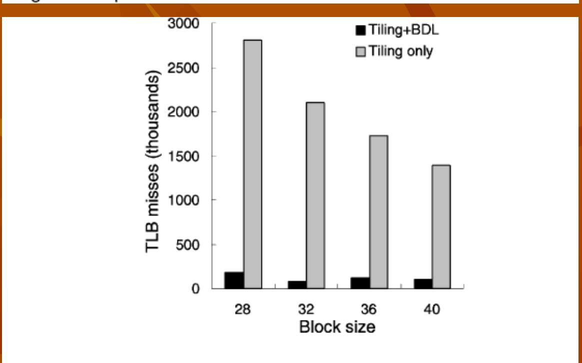


Fig. 6. Comparison of TLB misses using tiling + BDL and tiling only.

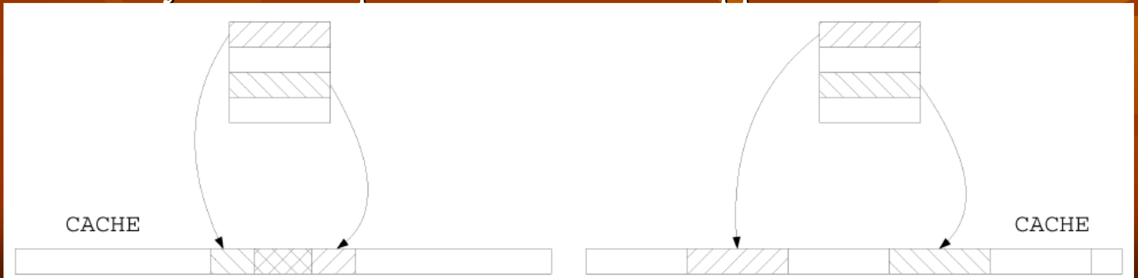
$$TM = \left(\frac{B^2}{P_v}+1\right)\left\{2\left(\frac{N}{B}\right)^3+\left(\frac{N}{B}\right)^2\right\}$$
$$= 2N^3\left(\frac{1}{BP_v}+\frac{1}{B^3}\right)+N^2\left(\frac{1}{P_v}+\frac{1}{B^2}\right)$$

(1)

6-loop TMM with BLD reduces TLB misses by a factor of *O(B)* compared with the 5-loop TMM with canonical layout

Cache performance

- Tiling
  - transforms the loop nest for a given cache size
    - =>temporal locality can be better exploited.
    - =>reduction of the capacity misses.
- However,
  - most of state-of-the art architectures have direct-mapped or small set-associative caches
    - =>tiling can suffer from considerable conflict misses
    - =>degradation of the overall performance
- Conflict misses are determined by
  - cache parameters
    - cache size, cache line size and set-associativity
  - runtime parameters
    - array size and block size.
- Therefore
  - Performance of tiled computations sensitive to lot of parameters
  - Positive effect of the parameter of block size if
    - block data layout with optimal block size is applied



Cache performance

- If block data layout is applied before tiled computations start
  - entire data that is accessed during a tiled computation will be localized in a block
  - And If block size < cache size
    - all elements can be stored in contiguous locations
      - self-interference misses are eliminated
- We observe similar behaviours of cache misses
  - for tiled access patterns on block layout and
  - for copying optimization on canonical layout
- Total number of cache misses for 6-loop TTM (tiling +BDL)
  - for *ith* level cache with S_{ci} cache size and L_{ci} cache line size

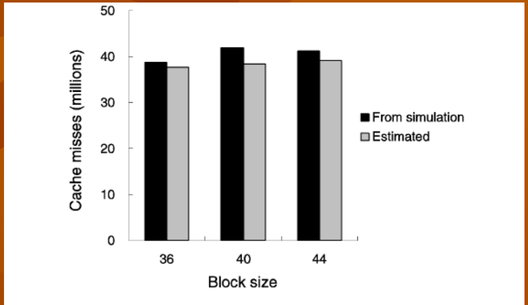


Fig. 8. Comparison of cache misses from simulation and estimation for 6-loop TMM.

Validation using SimpleScalar simulator
16KByte direct-mapped cache (L1 cache at Ultrasparc II)

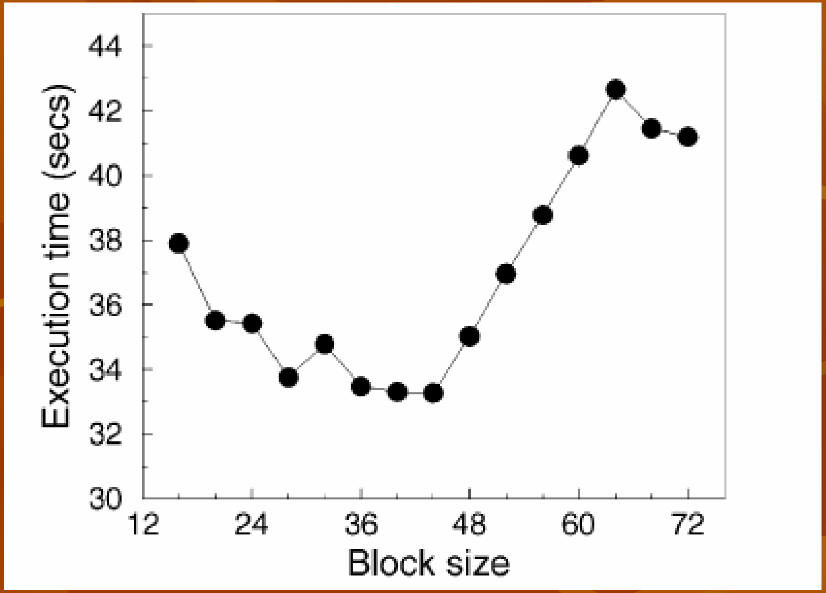
$$CM_i \approx \begin{cases} \frac{N^3}{L_{ci}}\left\{\frac{1}{B}\left(2+\frac{(3L_{ci}+2L_{ci}^2)}{S_{ci}}\right)+\frac{1}{N}+\frac{4B+6L_{ci}}{S_{ci}}\right\} & \text{for } B<\sqrt{S_{ci}} \\ \frac{N^3}{L_{ci}}\left\{\frac{4B}{S_{ci}}+\frac{2}{B}-\frac{2S_{ci}}{B^2}+2-\frac{1}{N}+\frac{6L_{ci}}{S_{ci}}\right\}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

(2)

N. Park, B. Hong, and V.K. Prasanna, "Memory Hierarchy Performance of Tiling and Block Data Layout," Technical Report USC-CENG 02-15, Dept. of Electrical Eng., Univ. of Southern California, Jan. 2003.

Block Size Selection

- Execution time of 6-loop TMM as a function of block size
  - size 1024 x 1024
  - UltraSparc II (400 MHz)



Block size selection is significant for achieving high performance

Block Size Selection

- Tiling + canonical data layout sensitive to
  - tile sizes
    - Several GCD-based tile size selection algorithms
  - problem sizes
    - remains a problem when problem size >> tile size
- Proposed solution approach
  - Consider both cache and TLB performance
  - Tiling eliminates cache capacity misses only
    - Only cache performance is improved
  - Need a method to deal with conflict cache misses
    - Block Data Layout looks promising
      - Improves TLB performance drastically
      - Block size is very important
        - So far only empirical methods (ATLAS)
      - Block size selection algorithm is needed

Block Size Selection

- Execution time in a multilevel memory hierarchy system
  - difficult to predict
  - proportional to the total miss cost of TLB and cache
    - => mimimize execution time = minimize the total sum of TLB and cache miss cost
  - Total miss cost of multi-level memory hierarchy

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

- *MC* = total miss cost
- CM_i = number of cache misses of *i*th level cache
- H_i = cost of a hit on the *i*th level cache (cost of a cache miss on the (*i-1*)th level cache
- *TM* = number of TLB misses
- M_{tlb} = cost of a TLB miss (TLB miss penalty)

Block Size Selection

- Problem
  - Estimation of the optimal block size for a multi-level memory hierarchy
    - Optimal block size = Block size that minimizes the total miss cost (MC)
- Method
  - Examine a simple case of 2-level hierarchy (TLB + L1 cache only)
    - Based on (1) (2) and (3) solve the equation
      - (Derivative of the total miss cost (*MC*)) = 0
    - Solution
      - optimal block size that minimizes the MC = B_{tc1}
  - Examine a usual case of 3-level hierarchy (TLB+L1+L2 caches)
    - Use eq. (1) , (2) and (3) and the previous optimal block size to define an optimal range of block sizes

Block Size Selection

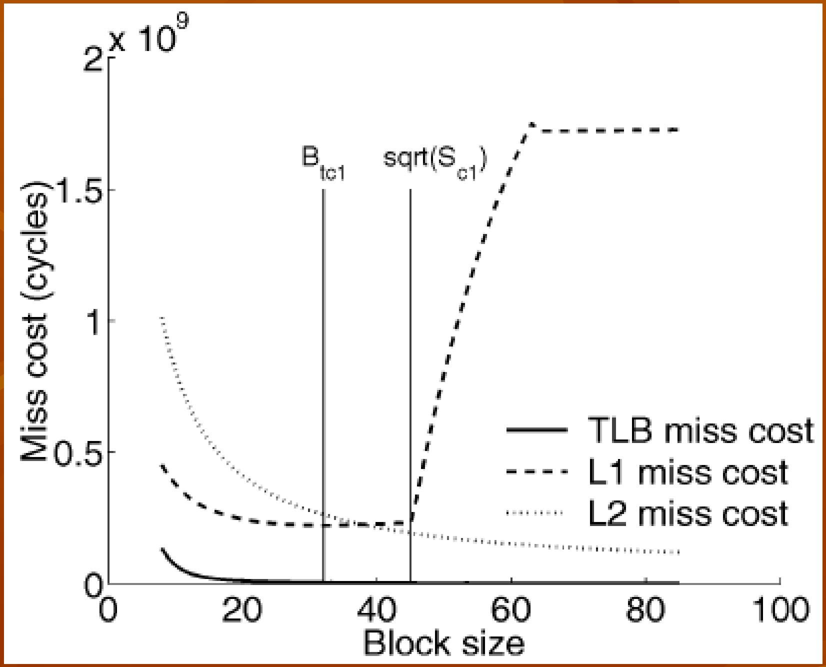
- 2-level memory hierarchy (TLB + L1 cache only)
  - Based on eq (3) => $MC_{tc1} = TM \cdot M_{tlb} + CM \cdot H_2$
    - MC_{tc1} = total miss cost of the 2-level hierarchy system
    - H_2 = the cost of a hit (access cost) at the main memory
  - Substitute *TM* and *CM* from (1) and (2)
  - Solve the equation: (derivative of *MC*) = 0
  - Solution

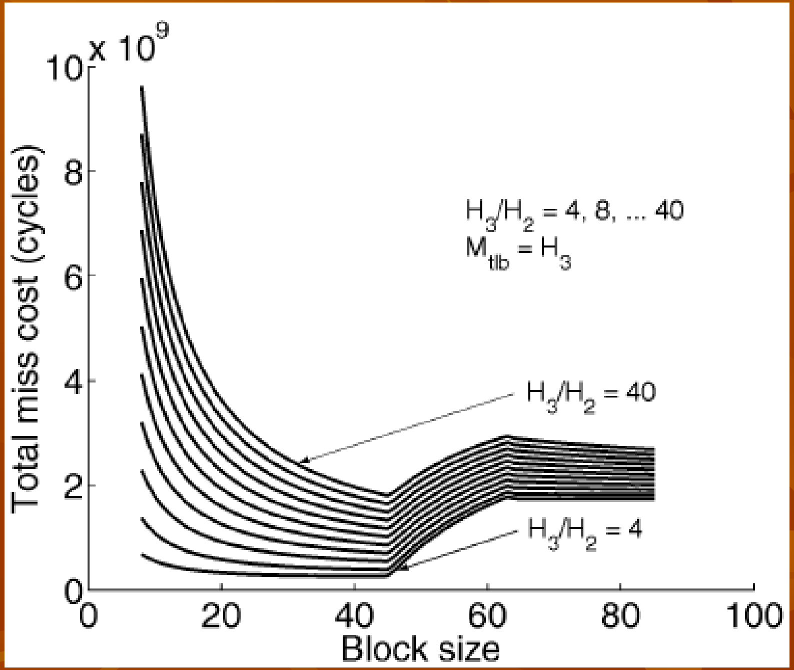
$$B_{tc1} \approx \sqrt{\frac{\left(\frac{2L_{c1}M_{tlb}}{P_v} + \left[2 + \frac{3L_{c1}+2L_{c1}^2}{S_{c1}}\right]H_2\right)S_{c1}}{4H_2}} \qquad \textbf{(4)}$$

    - B_{tc1} = optimal block size that minimizes the total miss cost of the TLB and L1 cache

Block Size Selection

- 3-level memory hierarchy system (TLB+L1+L2 caches)
  - Extension of the analysis of the 2-level hierarchy system





Block Size Selection

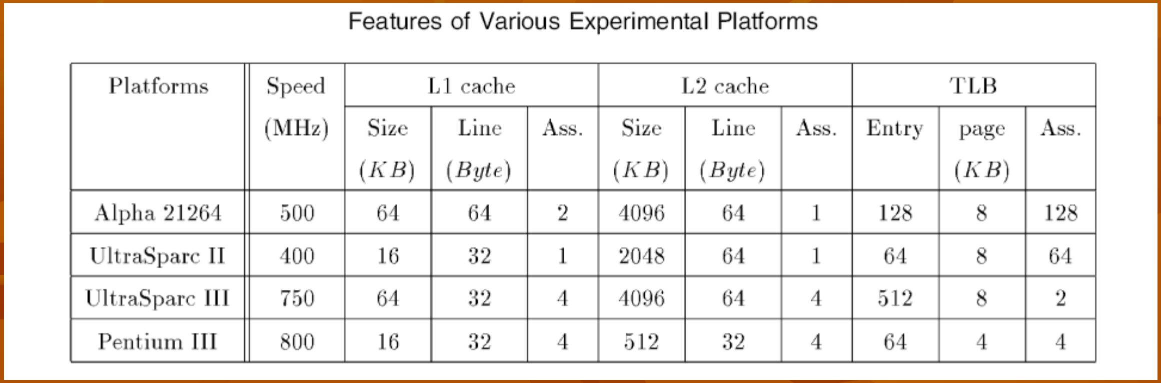
- Lemma 2 For $B < B_{tc1}$, $MC(B) > MC(B_{tc1})$
  - Proof
    - For $B < B_{tc1}$ we observe that $\frac{dMC_{tc1}}{dB} < 0$ and $\frac{dCM_2}{dB} < 0$

- Lemma 3 For $B > \sqrt{S_{c1}}$, $MC(B) > MC(\sqrt{S_{c1}})$
  - Proof
    - TLB miss cost is negligible as the block size increases
    - Block size is larger than L1 cache size
      - self-interferences occur in this range.
      - The number of L1 cache misses drastically increases
    - At the region $\sqrt{S_{c1}} \leq B < \sqrt{2S_{c1}}$ $\left|\frac{H_2 \frac{dCM_1}{dB}}{H_3 \frac{dCM_2}{dB}}\right| > 1$
      - the ratio between the L1 and L2 derivatives of *CM*
      - the increase in L1 cache miss cost is larger than the decrease in the L2 cache miss cost
    - At the region $B \geq \sqrt{2S_{c1}}$
      - no reuse takes place in L1 cache => L1 cache miss cost saturates
      - (L1 self-interference cost) >> L2 cache miss cost

- Theorem 4 The optimal block size B_{opt} satisfies $B_{tc1} \leq B_{opt} < \sqrt{S_{c1}}$
  - Select a block size that is a multiple of L1 cache line size in this range

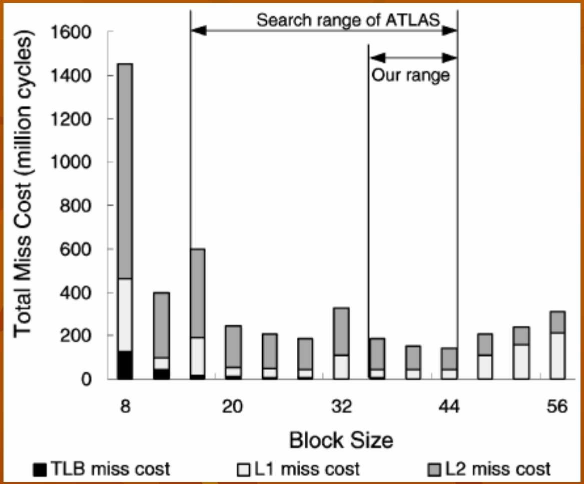
Block Size Selection

- Validation using Ultrasparc II parameters

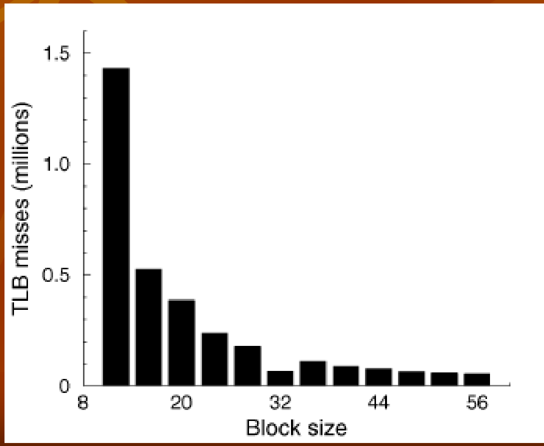


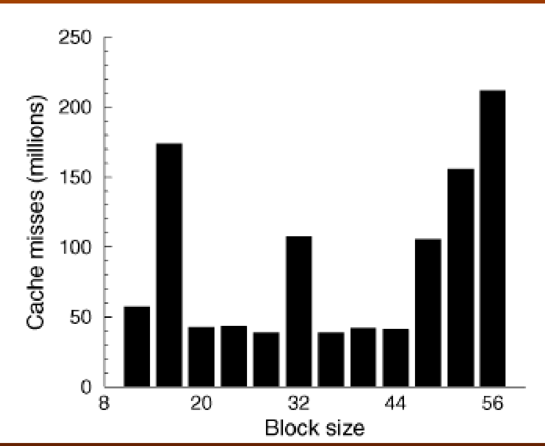
Features of Various Experimental Platforms

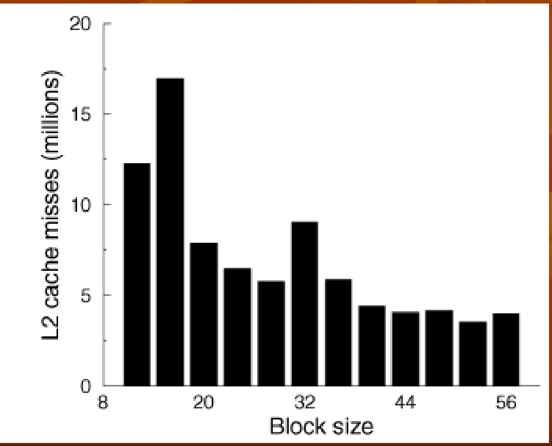
Platforms	Speed	L1 cache			L2 cache			TLB		
	(MHz)	Size (KB)	Line $(Byte)$	Ass.	Size (KB)	Line $(Byte)$	Ass.	Entry	page (KB)	Ass.
Alpha 21264	500	64	64	2	4096	64	1	128	8	128
UltraSparc II	400	16	32	1	2048	64	1	64	8	64
UltraSparc III	750	64	32	4	4096	64	4	512	8	2
Pentium III	800	16	32	4	512	32	4	64	4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



  - 6-loop TMM using BDL
  - TLB misses and L2 cache misses increase as block size B increases
  - Optimal block size = 36, L1 misses dramatically increase when B>45
  - B_{tc1} = 32.2, $\sqrt{S_{c1}}$ = 45.3, L_{c1} = 4

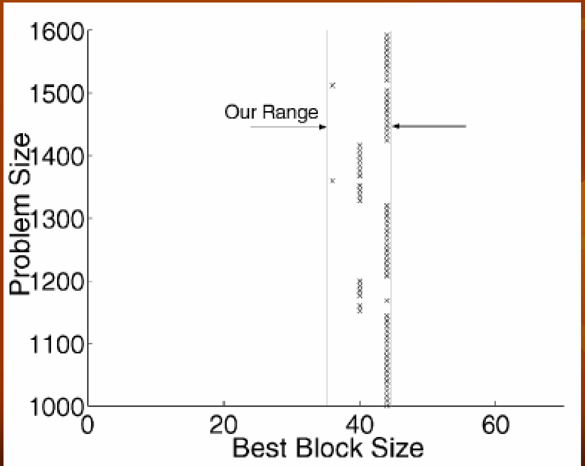






Block Size Selection

- Test 6-loop TMM with respect to
  - different block sizes
  - different problem sizes
- For each problem size [1000x1000 ... 1600x1600]
  - for various block sizes ranging: [8 ... 80]
    - find the optimal block size
- Return of test series
  - optimal block size always belongs to the range $B_{tc1} \le B_{opt} < \sqrt{S_{c1}}$
  - block size analysis is validated with experimental data from 6-loop TMM

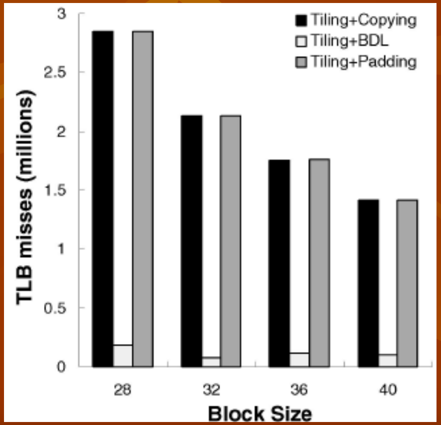


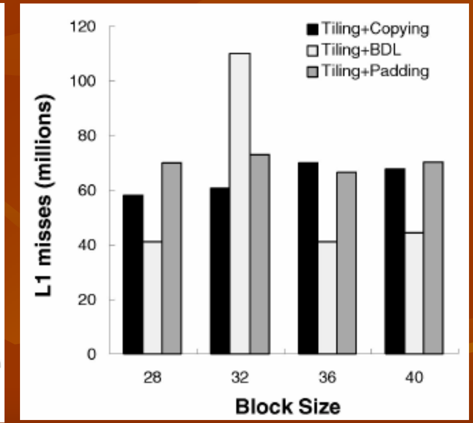
Validation using simulated and experimental data

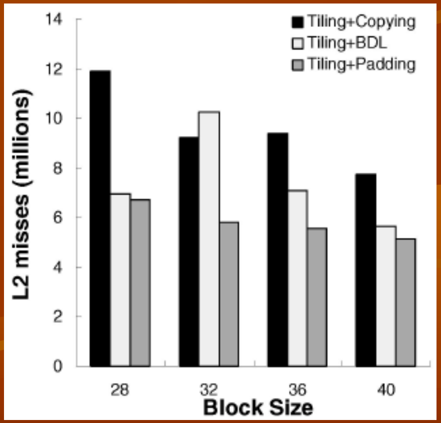
- Simulations and experiments were performed on
  - Tiled Matrix Multiplication
  - LU decomposition
  - Cholesky factorization
- Optimization methods examined
  - Tiling + BDL
  - Tiling + copying (only for TMM due to big offset)
  - Tiling + padding
- Input/Output at canonical layout
  - cost of conversion to other data layout included in reported results
- Data elements are double precision

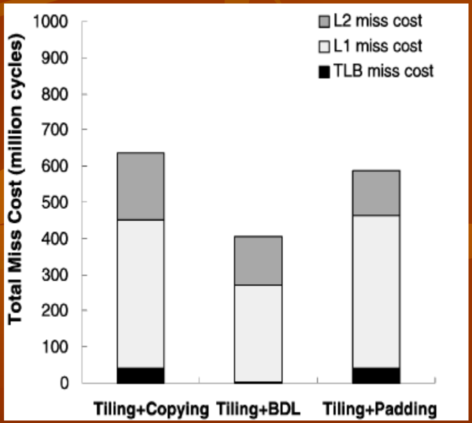
Validation through simulation - TMM

- SimpleScalar simulator
  - Ultrasparc II
  - Problem size: 1024 x 1024
- Tiling + BDL reduced L1, L2 *and* TLB misses
  - 91-96% reduction of TLB misses compared to other methods
  - block size 32 (and 64) leads to increased L1 and L2 misses <= cache conflicts between blocks
- Total miss cost
  - calculated using block size 40 x 40
  - penalties for L1=6cycles , L2=24cycles ,TLB=30cycles



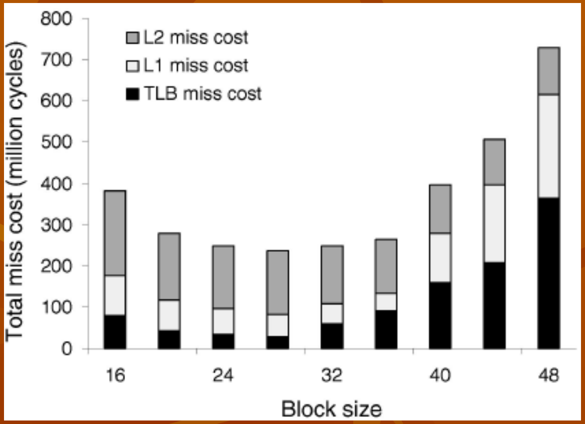


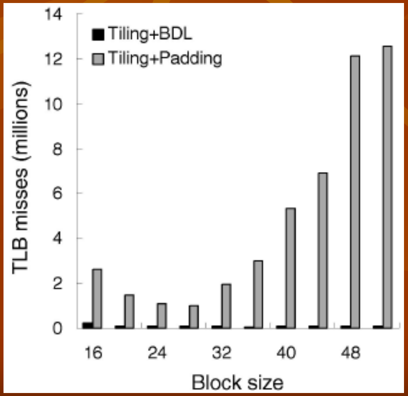


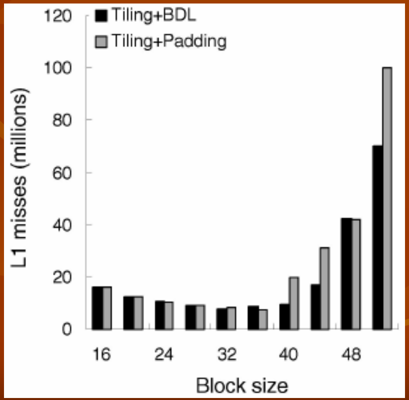


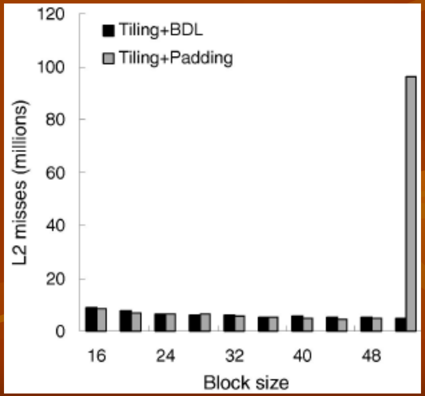
Validation through simulation - LU

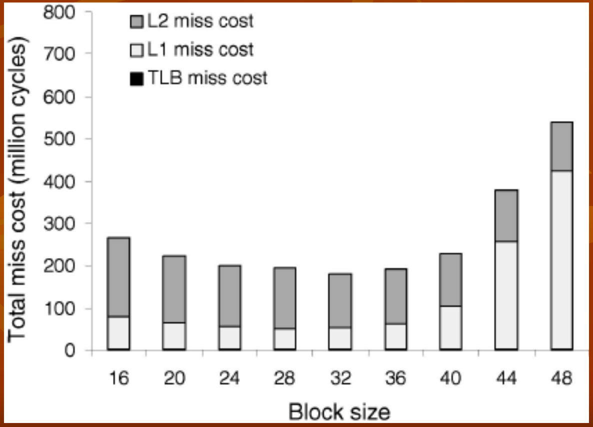
- SimpleScalar simulator
  - Pentium III
  - Problem size: 1024 x 1024
- Tiling + BDL
  - TLB misses almost negligible compared to other method
- 4-way set-associativity
  - significant reduction of L1 and L2 misses for both methods
- Tiling + padding
  - when block size > L1 cache size
    - pad size = 0, effectively no padding => L1 n L2 misses increase drastically
- Tiling + BDL wins because of better TLB performance





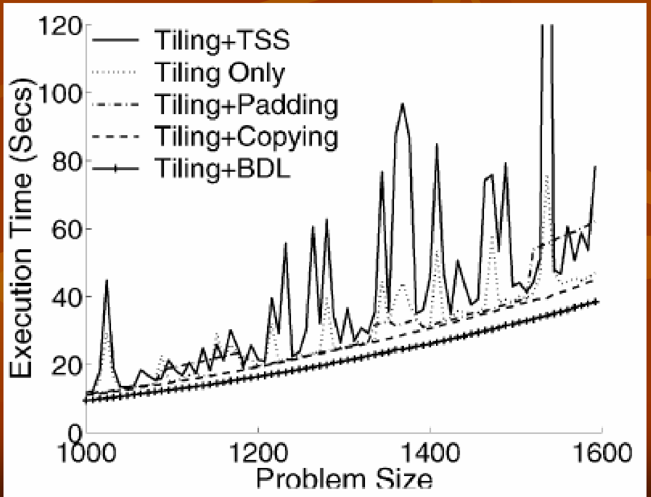






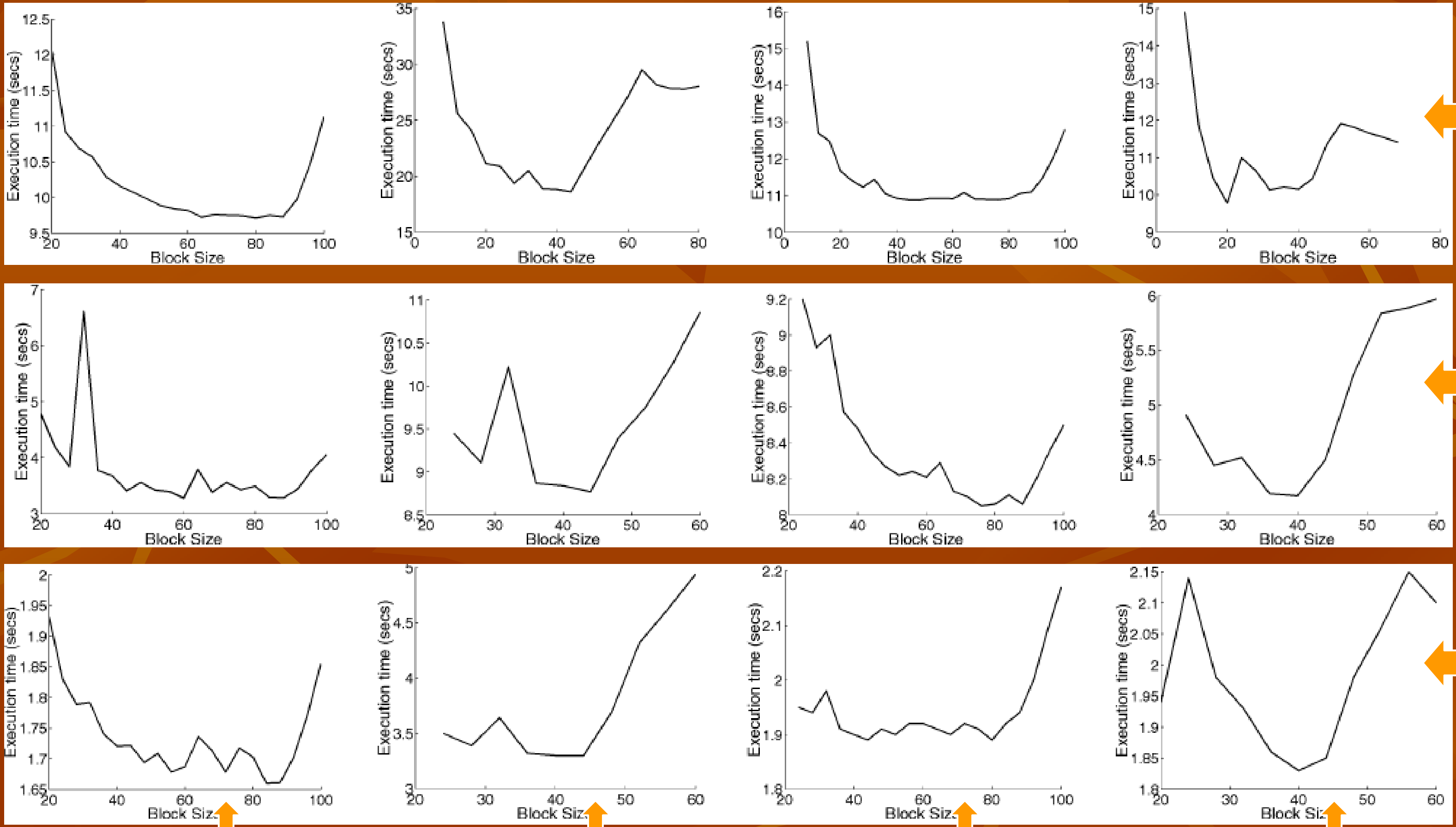
Validation through experiments on real platforms

- Experiments performed on several platforms
  - Pentium III, Ultrasparc II, Ultrasparc III, Alpha 21264
- gcc compiler optimization flags
  - `-fomit-frame-pointer -O3 -funroll-loops`
- execution time
  - user processor time measured by `clock()` (sys-call)
- all data averaged over 10 executions
- problem size range [1000x1000 ... 1600x1600]
- Tiling suffers from conflict misses
  - => sensitive to tile size and problem size
  - When tiling+BDL (optimal block size)
    - => reduce conflict misses,
    - eliminate problem size dependence



Validation through experiments on real platforms

- Various problem sizes examined on different platforms
  - similar performance between different problem sizes=> only results for problem size 1024x1024 are presented



Tiled Matrix Multiplication

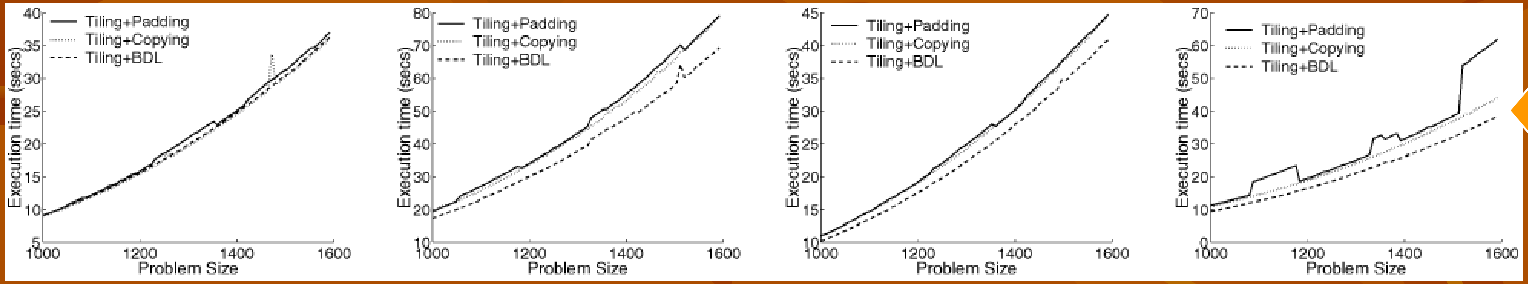
LU decomposition

Cholesky Factorization

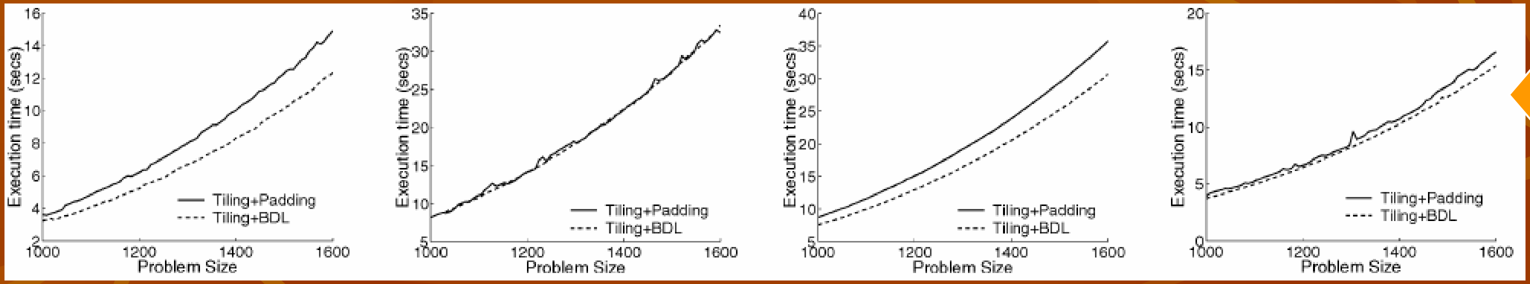
Alpha 21264 Ultrasparc II Ultrasparc III Pentium III

Validation through experiments on real platforms

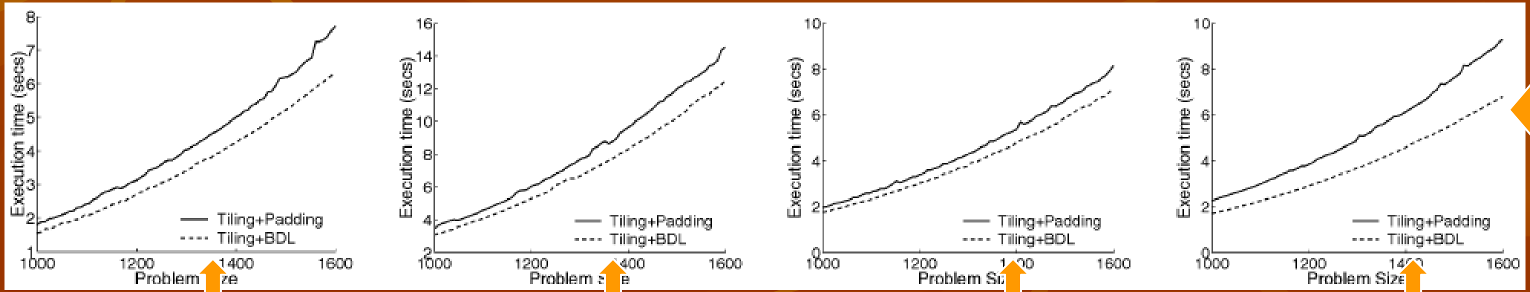
- Various problem sizes examined on different platforms
  - similar performance between different problem sizes=> only results for problem size 1024x1024 are presented



Tiled Matrix Multiplication



LU decomposition



Cholesky Factorization

Alpha 21264

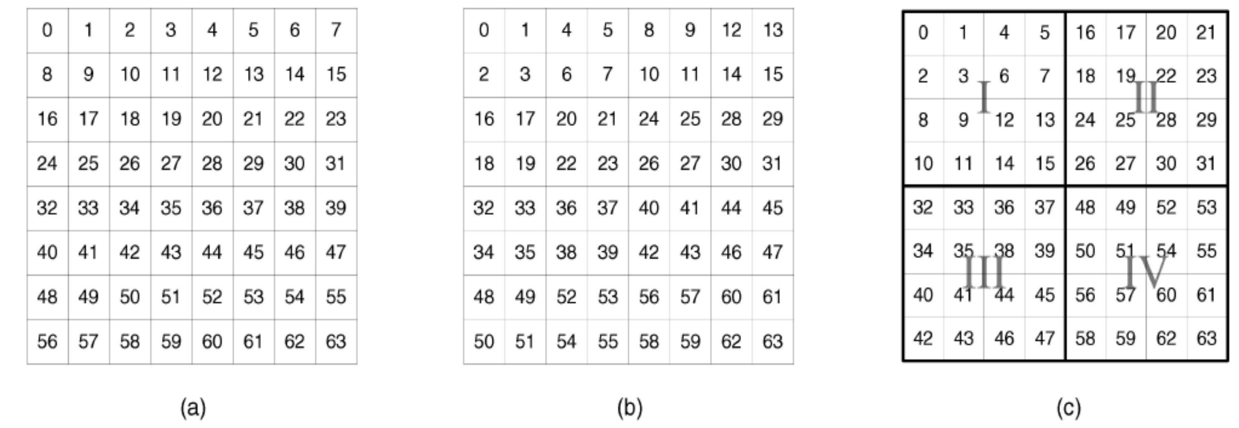
Ultrasparc II

Ultrasparc III

Pentium III

Comparative evaluation of data layouts

- Morton Data Layout (MDL) vs Block Data Layout (BDL)
  - Similarities
    - Elements within each block are mapped onto contiguous memory locations
  - Discrepancies
    - a different order to map blocks
    - order is not raw- or column-major
    - order matches the access pattern of recursive algorithms



(a)

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

(b)

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

(c)

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

Fig. 1. Various data layouts: block size 2×2 for (b) and (c). (a) Row-major layout, (b) block data layout, and (c) Morton data layout.

Row-major layout (canonical) **Block data layout** **Morton data layout**

MDL vs BDL

- Comparison of performance of
  - recursive algorithms using MDL (recursive + MDL)
  - iterative tiled algorithms using BDL (iterative + BDL)
- Applications used
  - Matrix Multiplication
  - LU decomposition
- MDL block size is limited because of recursion

  $B_{MDL} = \frac{N}{2^d}$
  - where *d* is the depth of recursion and assuming a *N x N* matrix
- If MDL block size lies at range $B_{tc1} \leq B_{opt} < \sqrt{S_{c1}}$
  - optimal performance for both MDL and BDL
- else
  - performance of MDL is degraded significantly compared to BDL

MDL vs BDL

- Iterative + BDL
  - Block size selected according to the block size selection algorithm proposed
- Recursive + MDL
  - Tested various recursive depths
  - Choose the depth with the best performance
  - Chosen depth determined the optimal block size
- Platforms used for experiments
  - Ultrasparc II
  - Pentium III

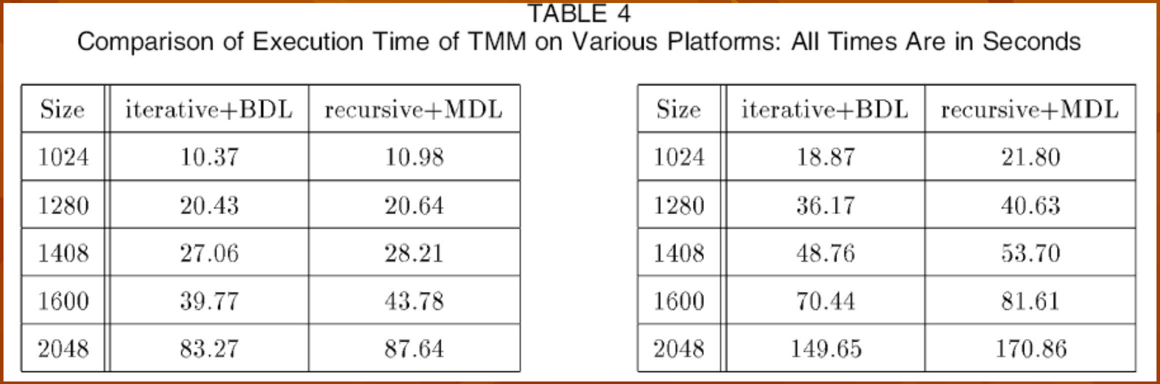


TABLE 4
Comparison of Execution Time of TMM on Various Platforms: All Times Are in Seconds

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

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

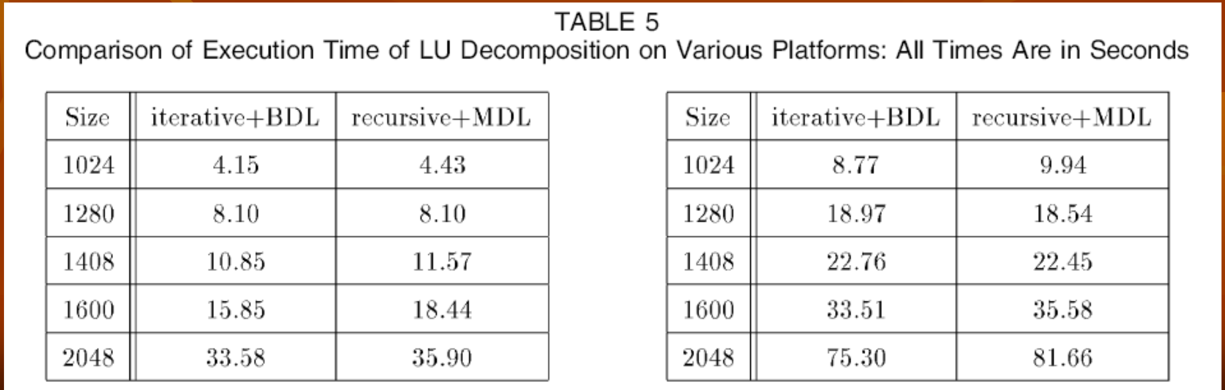


TABLE 5
Comparison of Execution Time of LU Decomposition on Various Platforms: All Times Are in Seconds

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

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

Conclusion

- Algorithms are strongly related to underlying multi-level memory hierarchies
- Block data layout improves performance of both TLB and caches
- Block size plays an important role to the efficiency of the BDL method
- A block size selection algorithm was proposed
  - validated both with simulated and experimental data
- Tiling+BDL exhibits better performance than tiling+copying and tiling+padding
  - validated both with simulated and experimental data

References

- "Tiling, Block Data Layout and Memory Hierarchy Performance", N. Park et al, Transactions on Parallel and Distributed Systems 2003
- "The Cache Performance and Optimizations of Blocked Algorithms", M. Lam et al, ACM 1991
- "Comparative Evaluation of Latency Reducing and Tolerating Techniques", A. Gupta et al, ACM 1991
- "Comparing Latency Tolerance Techniques for software DSM systems", R. Pinto et al, TDPS 2003
- "Latency Tolerance", Book chapter 11, Parallel Computer Architecture, D. Culler et al
- "Parallel Programming Considerations", "Application Issues", Chapter 3 & 4, Sourcebook on Parallel Computing, J. Dongarra et al