Minimizing Associativity Conflicts in Morton Layout

Jeyarajan Thiyagalingam[1,2], Olav Beckmann[1], and Paul H. J. Kelly[1]

[1] Department of Computing, Imperial College London,
180 Queen's Gate, London SW7 2AZ, United Kingdom
www.doc.ic.ac.uk/{~jeyan,~ob3,~phjk}
[2] Harrow School of Computer Science, University of Westminster,
Watford Road, Northwick Park, Harrow HA1 3TP, United Kingdom
t.jeyan@wmin.ac.uk

Abstract. Hierarchically-blocked non-linear storage layouts, such as the Morton ordering, have been shown to be a potentially attractive compromise between row-major and column-major for two-dimensional arrays. When combined with appropriate optimizations, Morton layout offers some spatial locality whether traversed row- or column-wise. However, for linear algebra routines with larger problem sizes, the layout shows diminishing returns. It is our hypothesis that associativity conflicts between Morton blocks cause this behavior and we show that carefully arranging the Morton blocks can minimize this effect. We explore one such arrangement and report our preliminary results.

1 Introduction

For dense multidimensional arrays, programming languages mandate one of the two canonical layouts — row-major and column-major. Traversing an array in its major order results in excellent spatial locality; however, traversing an array in opposite order to its major order can lead to an order-of-magnitude worse performance.

In our earlier work [7] we considered whether Morton layout can be an alternative storage layout to canonical layouts. Although Morton layout offers equal spatial locality both in row- and column-major order traversals, our early work in the area suggested that the performance of standard Morton layout may be disappointing. Following this observation, we proposed two optimization schemes for Morton arrays — unrolling and alignment. By exhaustively evaluating these optimizations, we demonstrated that unrolling combined with strength-reduction of the Morton index calculation and correct alignment of the base address of Morton arrays can lead to a significant improvement in performance [7]. A key remaining weakness which we address in this paper is that the performance of Morton layout tends to deteriorate with larger problem sizes.

Contributions of this Paper. In this paper we discuss how minimizing associativity conflicts in the two-dimensional Morton layout may lead to further improvements in performance. The main contributions of this paper are:

- We perform an in-depth analysis of associativity conflicts in Morton layout.
- We propose a hybrid layout scheme to minimize associativity conflicts in Morton arrays, and we discuss combining this scheme with a modest amount of padding.

– We demonstrate the effectiveness of our proposed scheme through an experimental evaluation, using a suite of non-tiled micro-benchmarks.

This work differs from other work in this area, which mainly focused on optimizing for temporal locality through hierarchical tiling, by targeting spatial locality of non-tiled applications.

Structure of the Remainder of this Paper. In Section 2 we discuss related and previous work relevant to this paper. We illustrate the impact of associativity conflicts in Morton arrays and we propose a variant of Morton layout to address this problem in Section 3. Following this, we evaluate and report performance results for our proposed scheme in Section 4. Section 5 concludes the paper and discusses future work.

2 Previous Work

Many authors have studied recursive layouts in the context of performance optimization. Notably, Wise *et al.* [8], Chatterjee *et al.* [1, 2], and Gustavson [4] pioneered these layouts, focusing on optimizing for temporal locality. Their implementations are either tiled or recursively formulated. In [4] Gustavson adopts a similar approach to Chatterjee [1, 2]; however, Gustavson's work is focused on deriving optimal layouts for particular problems rather than a generic solution.

Recursive layouts, by definition, require a complete decomposition up to the element level. Many authors [1–3] have identified that such a complete decomposition may lead to increased conflict misses between different blocks, and have proposed different variants to fully recursive layouts. Chatterjee *et al.* [1] propose a family of non-linear alternative layouts, called 4D layouts, which intermix recursive and linear layouts. In one such variant, Chatterjee *et al.* divide two-dimensional arrays into linearly arranged tiles, which are themselves blocked.

Drakenberg *et al.* [3] propose a semi-hierarchical layout (called HAT) and a linear-algebra framework to determine conflict misses at compile time. Their layout is very similar to one of the 4D layouts mentioned in [1, 2]. Their work, similar to ours, also considers non-tiled or non-recursive algorithms but does not discuss padding.

3 Conflict Misses in Morton Layout

Associativity Conflicts in the Standard Morton Layout. One of the advantages of using the standard Morton scheme, where blocking is applied recursively up to the element level, is its simplicity — there is no need to choose blocking factors. This is potentially useful for developing programs independently of underlying architectural features. However, different sub-blocks within a Morton array may conflict with each other. We demonstrate this effect in Figure 1(a): For a page-sized direct-mapped cache, every page-sized sub-block of a Morton array conflicts with every other sub-block. This appears to suggest that complete decomposition is sub-optimal.

In general, for a w-way associative cache with capacity C, addresses aligned at $(\frac{C}{w})$ bytes are mapped to the same set. If each word is l bytes, each way holds $\frac{C}{wl}$ words.

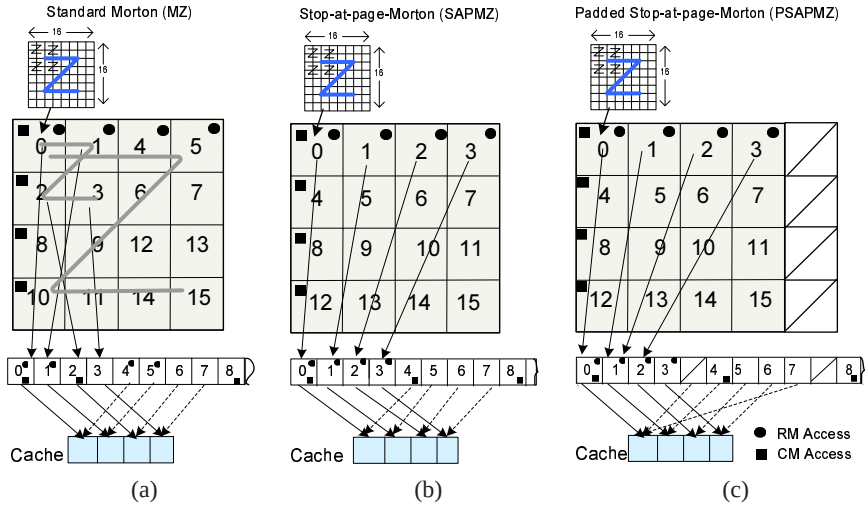


Fig. 1. Conflict Misses in Morton Layout. We show three different variants of the Morton scheme, where each location in the array represents a page-sized Morton block. In each case, we also show the mapping of pages into linear memory and into a direct-mapped 4-page cache. When accessing the standard Morton array (MZ) in row-major order, Morton pages 0 and 1 conflict with pages 4 and 5. Similarly, when accessing the MZ array in column-major order, pages 0 and 2 conflict with pages 8 and 10. This generates 2 misses per row or column for each traversal order. However, when accessing the Morton variant (SAPMZ) in row-major order, traversal of a single row is free from conflicts, but a column-major traversal suffers worse conflicts than the MZ layout. The diagram on the far right (PSAPMZ) shows how padding can be used to eliminate systematically recurring conflicts for column-major traversal. The same analysis is valid for caches with higher levels of associativity.

Note that C, w and l are typically powers-of-two, so $\frac{C}{wl}$ will also be a power-of-two. Two arbitrary word-addresses s and t collide if $|s-t| = \frac{C}{wl}$. With row-major layout, this happens with `A[i,s]` and `A[i,t]`, or for elements separated by a multiple of $|s-t|$. With Z-Morton layout this happens with `A[i,u]` and `A[i,v]` when $|u-v| = \sqrt{\frac{C}{lw}}$, if $\frac{C}{wl}$ is an even power-of-two (*i.e.*, if $\frac{C}{wl}$ words form a "square"). If $\frac{C}{wl}$ is an odd power-of-two, addresses `A[i,u]` and `A[i,v]` collide if $|u-v| = 2\sqrt{\frac{C}{2lw}}$.

Example. As a practical example, consider the ADI algorithm. In a slightly simplified form, this contains the loop shown in Figure 2. For good performance, this loop requires row-to-row re-use: the reference `A[i-1][j]` should come from cache, having been loaded on the previous iteration of the for-`i` loop. At what datasize will associativity conflicts prevent this in a standard Morton array? According to our analysis in the preceding paragraph, on a Pentium 4 processor with an 8-way set-associative 512KB L2 cache, and an array of 8-byte double words, addresses `A[i,u]` and `A[i,v]` will conflict ($\frac{C}{wl}$ is odd in this case) when $|u-v| = 2\sqrt{\frac{512\times 2^{10}}{2\times 8\times 8}} = 128$. With an 8-way set-associative cache, this means that we begin to lose row-to-row re-use at a row length of

```
1  for( int i = 1; i < sz; ++i )
2    for( int j = 0; j < sz; ++j )
3      A[i][j] += A[i-1][j];
```

Fig. 2. Simplified loop from the ADI algorithm, requiring row-to-row re-use for good performance.

$128 \times 8 = 1024$. This observation is confirmed by our experimental results for the Adi algorithm on a Pentium 4 processor [6, 7] (see also Figure 3).

Stop-at-Page Morton Layout: A Hybrid Scheme. An alternative to standard Morton, as suggested by Chatterjee *et al.* [1] and Drakenberg *et al.* [3], is to divide the array into row-major or column-major ordered blocks and arranging the elements within each block in Z-Morton order. Choosing page-sized Morton blocks guarantees unbiased TLB behavior for both row-major and column-major traversals. We refer to this scheme as Stop-at-Page-Morton (since we stop the blocking at page-level). This enables us to utilize the compromise property of Morton layout and to minimize the effects of associativity conflicts among Morton blocks at the same time. The resulting effect on associativity conflicts is shown in Figures 1(b).

Notice, however, the increased dilation effect and the introduction of systematically repeating conflicts for column-major traversal of SAPMZ arrays. A simple method to avoid these systematically recurring conflicts is to pad each row of the array by a Morton block, whenever the number of Morton blocks in a row is even. We refer to this technique as Padded Stop-at-Page-Morton scheme (PSAPMZ). Although PSAPMZ does not improve spatial locality for column-major traversal, it does minimize associativity conflicts for column-major traversal. This is illustrated in Figure 1(c).

Implementation Issues. For standard Morton arrays, it is necessary to round up the array sizes of each dimension to the next power-of-two. For Stop-at-Page-Morton, padding is only necessary up to the next page size and will never be worse than the storage requirements for standard Morton, except for array sizes smaller than a page. The Padded-Stop-at-Page-Morton pads the row-length by an additional page-sized Morton block when the number of such blocks in a row would otherwise be even.

Page size may vary from architecture to architecture, and may not always correspond to an even power-of-two number of words. For most x86 variants (including the systems we used to test our hypothesis) the page size is 4KB, and we chose a 16×16 array of 8-byte doubles (half a page) as the largest Morton block.

4 Experimental Evaluation

Benchmark kernels and architectures. To test our proposed Stop-at-Page-Morton and Padded Stop-at-Page-Morton layouts, we have collected a suite of simple implementations of standard, non-tiled numerical kernels operating on two-dimensional arrays and carried out experiments on the Pentium 4 architecture. Table 1 summarizes the details

Processor	Operating System	L1/L2/Memory Parameters	Compiler and Flags Used
Pentium 4 2.0 GHz	Linux 2.4.26	L1 D-cache: 4-way, 8KB, 64B cache line L2 cache: 8-way, 512KB, 128B cache line Page size: 4KB Main Memory: 512MB DDR-RAM	Intel C/C++ Compiler v8.1 -xW -ipo -O3 -static

Table 1. Cache and CPU configuration used in the experiments. Compiler and compiler flags match those used by the vendor in their SPEC CFP2000 (base) benchmark reports [5].

MMijk	Matrix multiply, ijk loop nest order (usually poor due to large stride)
MMikj	Matrix multiply, ikj loop nest order (usually best due to unit stride)
Jacobi2D	Two-dimensional four-point stencil smoother
ADI	Alternating-direction implicit kernel, ij,ij order
Cholesky	K-variant (usually poor due to large stride)

Table 2. Numerical kernels used in our experimental evaluation [6].

of the architecture. The kernels used in our experiments are shown in Table 2. We carried out measurements over a full range of problem sizes and followed the experimental approach detailed in [6] to minimize the effects of external interference.

Performance Results. Figure 3 shows our results in detail, and we make some comments directly in the figure. For each experiment, we give a broad characterization of the performance of the different Morton schemes we tested.

Impact of the Padded and Non-Padded Stop-at-Page-Morton Scheme. For Adi, MMijk and Cholesky-k kernels our theoretical conclusions from Section 3 are supported by our experimental data. For MMijk and Cholesky, the SAPMZ scheme does not offer an improvement over standard Morton; however, padding the Stop-at-Page-Morton scheme does help in these benchmarks. For the remaining benchmarks, Jacobi2D and MMikj, neither SAPMZ nor Padded SAPMZ offer an improvement over standard Morton.

5 Conclusions and Future Work

Our hypothesis of how conflict misses in Morton layout can be minimized by the SAPMZ and Padded SAPMZ layouts is supported by some but not all experimental results. The padded SAPMZ scheme, in contrast to the SAPMZ scheme, has consistently improved performance. Further, the reduction in associativity conflicts offered by padded SAPMZ results in both row-major and column-major traversals of hierarchical arrays being yet closer in performance to row-major traversal of row-major arrays. There are number of interesting issues that remain to be addressed:

– Exploring large arrays. Our experimental results are limited to a range of problem sizes where the performance gain by SAPMZ/Padded SAPMZ occurs close to the

upper limit of the problem sizes we consider. We would like to extend our work such that it covers a larger range of problem sizes.
- Hardware performance counters: Some features in the graphs may be explained with the help of hardware performance counters. We plan to explore this further by using one of the available hardware performance counter measurement tools.
- Mixed Morton layouts: So far, we have only considered Z-Morton as the underlying layout. Mixing this with other members of the Morton layout family, which may have complementary effects, may lead to lower associativity conflicts.

References

[1] Siddhartha Chatterjee, Vibhor V. Jain, Alvin R. Lebeck, Shyam Mundhra, and Mithuna Thottethodi. Nonlinear array layouts for hierarchical memory systems. In *ICS '99: Proceedings of the 13th International Conference on Supercomputing*, pages 444–453, 1999.

[2] Siddhartha Chatterjee, Alvin R. Lebeck, Praveen K. Patnala, and Mithuna Thottethodi. Recursive array layouts and fast parallel matrix multiplication. In *SPAA '99: Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 222–231, 1999.

[3] Peter Drakenberg, Fredrik Lundevall, and Björn Lisper. An efficient semi-hierarchical array layout. In *Proceedings of the Workshop on Interaction between Compilers and Computer Architectures*. Kluwer, January 2001. Available via `www.mrtc.mdh.se`.

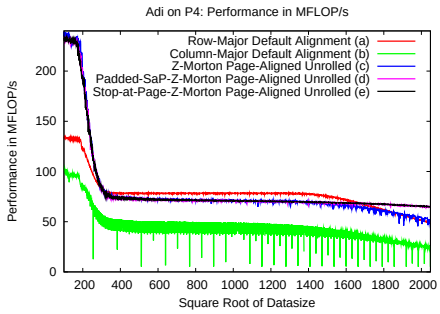
[4] Fred G. Gustavson. New generalized data structures for matrices lead to a variety of high performance algorithms. In R. Wyrzykowski, J. Dongarra, M. Paprzycki, and J. Waniewski, editors, *PPAM 2001: Proceedings of the 4th International Conference on Parallel Processing and Applied Mathematics*, volume 2328 of *LNCS*, pages 418–436. Springer-Verlag, September 2002.

[5] SPEC 2000 CPU Benchmarks. `http://www.specbench.org/`, 2000.

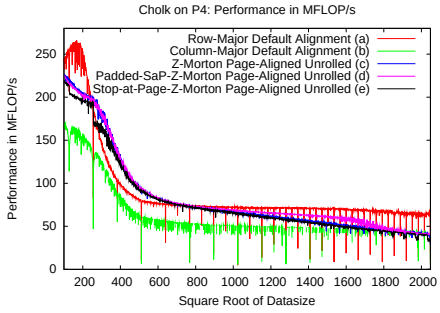
[6] Jeyarajan Thiyagalingam. *Alternative Array Storage Layouts for Regular Scientific Programs*. PhD thesis, Department of Computing, Imperial College, London, U.K., January 2005.

[7] Jeyarajan Thiyagalingam, Olav Beckmann, and Paul H. J. Kelly. Improving the performance of Morton layout by array alignment and loop unrolling reducing the price of naivety. In Lawrence Rauchwerger, editor, *Proceedings of 16th International Workshop on Languages and Compilers for Parallel Computing*, volume 2958 of *LNCS*, pages 241–257. Springer-Verlag, October 2003.

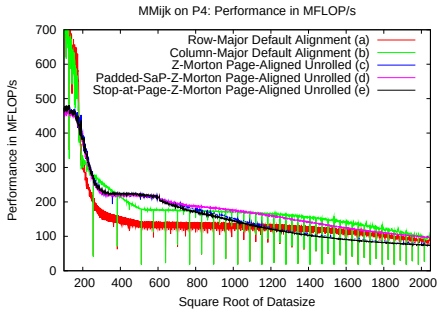
[8] David S. Wise, Jeremy D. Frens, Yuhong Gu, and Gregory A. Alexander. Language support for morton-order matrices. In *PPoPP '01: Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*, pages 24–33, 2001.



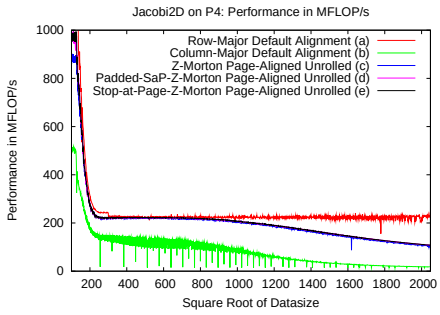
- Standard Morton performs almost as well as row-major.
- The performance of standard Morton begins to deteriorate around 1600×1600.
- Both SAPMZ and Padded SAPMZ sustain the performance for problem sizes larger than 1600×1600.

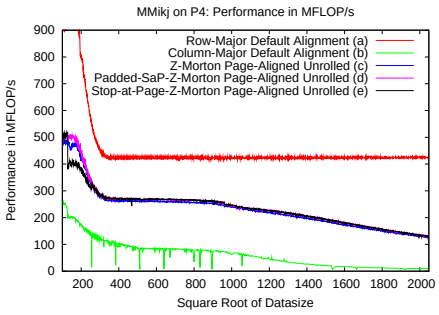


- Both SAPMZ and Padded SAPMZ never perform worse than standard Morton.
- SAPMZ does not result in a noticeable improvement over standard Morton.
- For a sub-range of problem sizes between 900×900 and 1900×1900, the padded SAPMZ scheme out-performs standard Morton and SAPMZ.



- Both SAPMZ and Padded SAPMZ never perform worse than standard Morton.
- SAPMZ does not result in a noticeable improvement over standard Morton.
- For problem sizes larger than about 512×512, Padded SAPMZ out-performs standard Morton and SAPMZ.





- For the Jacobi2D and MMikj benchmarks, SAPMZ and Padded SAPMZ scheme do not offer a performance improvement over standard Morton.

Fig. 3. Performance of different layouts using a suite of micro-benchmarks running on the Pentium 4 system described in Table 1.