Advanced Computer Architecture

Chapter 4: Caches and Memory Systems
Part 2: miss rate reduction using software

October 2023

Paul H J Kelly

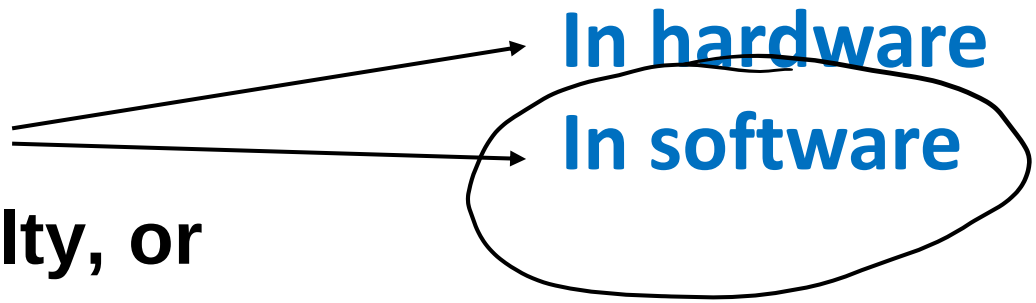
These lecture notes are partly based on the course text, Hennessy and Patterson's *Computer Architecture, a quantitative approach (3rd, 4th, 5th and 6th eds)*, and on the lecture slides of David Patterson and John Kubiatowicz's Berkeley course

Average memory access time:

AMAT = HitTime + MissRate × MissPenalty

There are three ways to improve AMAT:

1. **Reduce the miss rate,**
2. **Reduce the miss penalty, or**
3. **Reduce the time to hit in the cache**



We now look at each of these in turn…

Reducing misses by software prefetching

- Some processors have instructions to trigger prefetching explicitly, in software
- *Almost never worth using on sophisticated processors with good hardware prefetching*
- May be useful on simpler processors
- Some care is needed to ensure prefetch accesses don't have unwanted side effects
  - Eg memory-mapped i/o registers
    - (this is the function of the R10KCBARRIER macro)
- Prefetch instructions may target addresses that would cause a page fault or protection violation
  - Prefetches of addresses that would result in a page fault or exception are silently squashed

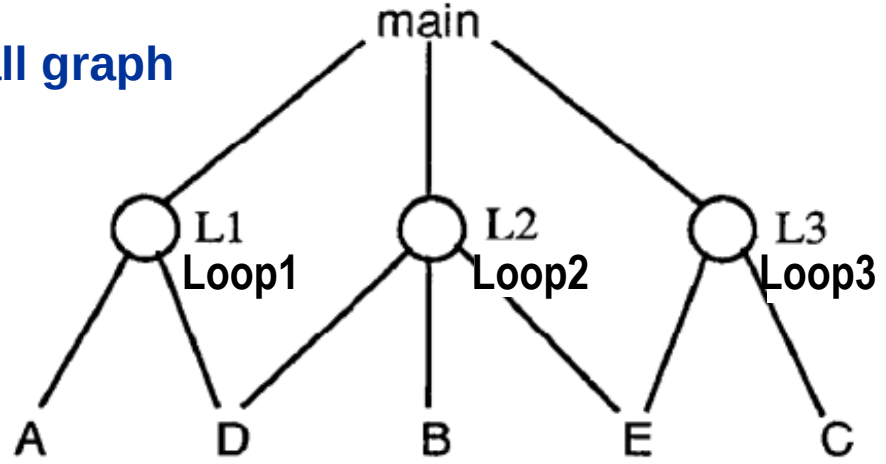
```
1:
        R10KCBARRIER(0(ra))
        LOAD(t0, UNIT(0)(src), .Ll_exc\@)
        LOAD(t1, UNIT(1)(src), .Ll_exc_copy\@)
        LOAD(t2, UNIT(2)(src), .Ll_exc_copy\@)
        LOAD(t3, UNIT(3)(src), .Ll_exc_copy\@)
        SUB     len, len, 8*NBYTES
        LOAD(t4, UNIT(4)(src), .Ll_exc_copy\@)
        LOAD(t7, UNIT(5)(src), .Ll_exc_copy\@)
        STORE(t0, UNIT(0)(dst), .Ls_exc_p8u\@)
        STORE(t1, UNIT(1)(dst), .Ls_exc_p7u\@)
        LOAD(t0, UNIT(6)(src), .Ll_exc_copy\@)
        LOAD(t1, UNIT(7)(src), .Ll_exc_copy\@)
        ADD     src, src, 8*NBYTES
        ADD     dst, dst, 8*NBYTES
        STORE(t2, UNIT(-6)(dst), .Ls_exc_p6u\@)
        STORE(t3, UNIT(-5)(dst), .Ls_exc_p5u\@)
        STORE(t4, UNIT(-4)(dst), .Ls_exc_p4u\@)
        STORE(t7, UNIT(-3)(dst), .Ls_exc_p3u\@)
        STORE(t0, UNIT(-2)(dst), .Ls_exc_p2u\@)
        STORE(t1, UNIT(-1)(dst), .Ls_exc_p1u\@)
        PREFS(  0, 8*32(src) )
        PREFD(  1, 8*32(dst) )
        bne     len, rem, 1b
         nop
```

- Example: MIPS "memcpy" library code – handwritten assembler – unrolled 12 times, manually scheduled, with prefetching to initiate loading the source and destination cache lines into cache (heavy use of macros)
- From https://elixir.bootlin.com/linux/v5.9.2/source/arch/mips/lib/memcpy.S

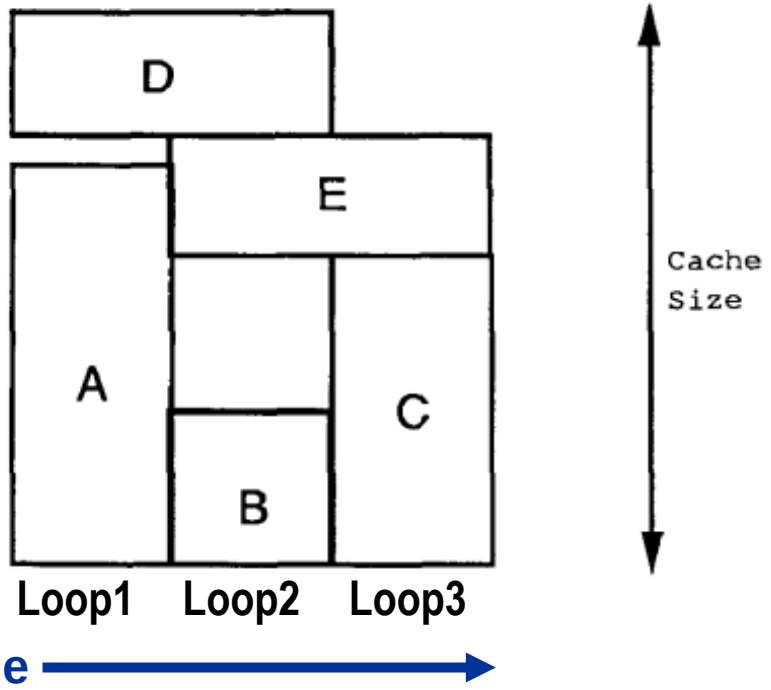
Reducing instruction-cache misses

- McFarling [1989]* reduced instruction cache misses by 75% on 8KB direct mapped cache, 4 byte blocks in software
- Instructions
  - By choosing instruction memory layout based on callgraph, branch structure and profile data
  - Reorder procedures in memory so as to reduce conflict misses
  - (actually this really needs the *whole* program – a link-time optimisation)

Call graph



Packing code for each function into the I-cache



Function E is placed to avoid conflicts with B and C, but can be placed in addresses that conflict with A

* "Program optimization for instruction caches", ASPLOS89, http://doi.acm.org/10.1145/70082.68200

- **Storage layout transformations**
  - ***Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays**
  - ***Permuting a multidimensional array*: improve spatial locality by matching array layout to traversal order**
  - **Improve *spatial* locality**
- **Iteration space transformations**
  - ***Loop Interchange*: change nesting of loops to access data in order stored in memory**
  - ***Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap**
  - ***Blocking*: Improve temporal locality by accessing "blocks" of data repeatedly vs. going down whole columns or rows (wait for Chapter 4)**
  - **Can also improve *temporal locality***

Array Merging - example

```
/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];

/* After: 1 array of stuctures */
struct merge {
    int val;
    int key;
};
struct merge merged_array[SIZE];
```

"Array of Structs" vs
"Struct of Arrays"

(AoS vs SoA)

Reducing conflicts between val & key (example?)

Improve spatial locality (counter-example?)

- **whether this is a good idea depends on access pattern**

(actually this is a transpose: 2*SIZE -> SIZE*2)

Consider matrix-matrix multiply (tutorial ex)

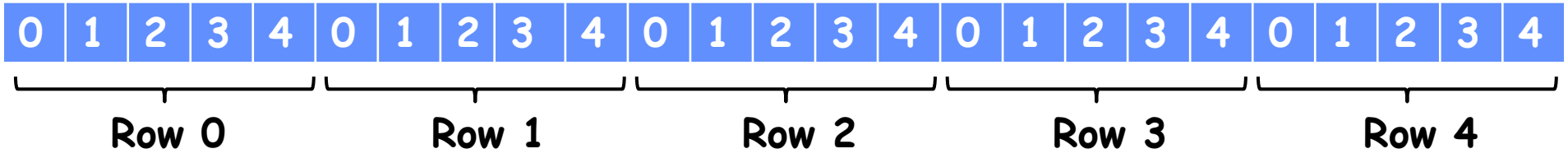
- **MM1:**

```
for (i=0;i<N;i++)
 for (j=0;j<N;j++)
   for (k=0;k<N;k++)
     C[i][j] += A[i][k] * B[k][j];
```

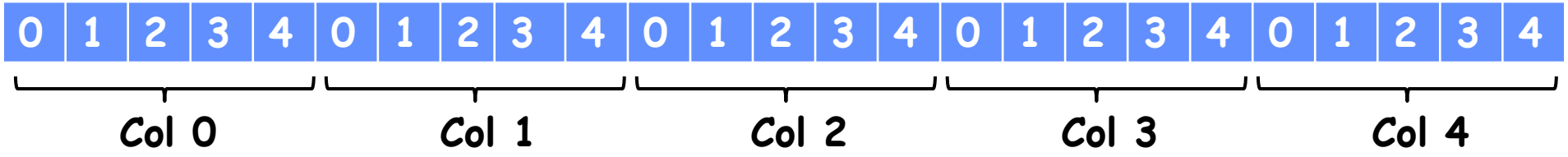
- **MM2:**

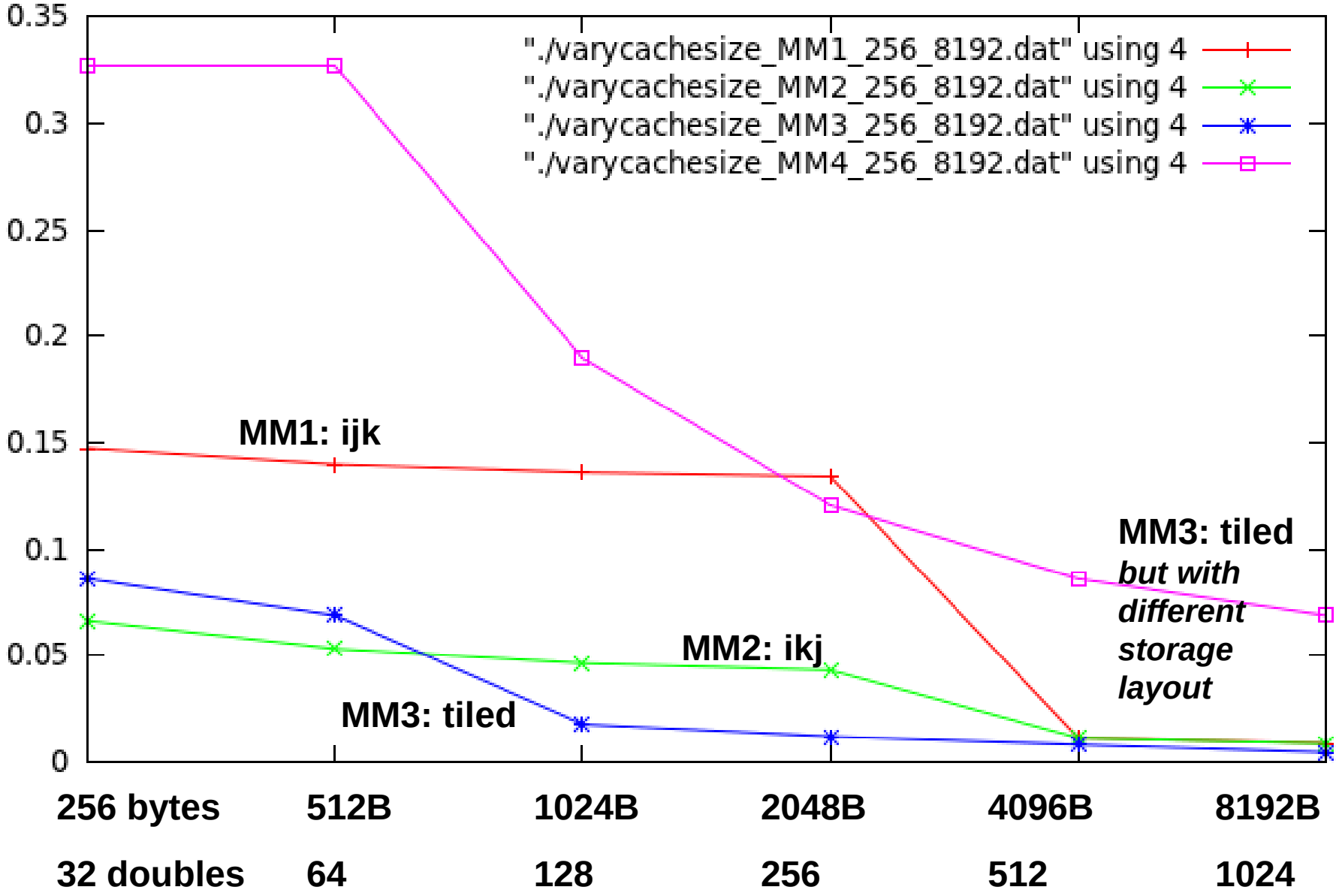
```
for (i=0;i<N;i++)
 for (k=0;k<N;k++)
   for (j=0;j<N;j++)
     C[i][j] += A[i][k] * B[k][j];
```

- **Row-major storage layout (default for C):**



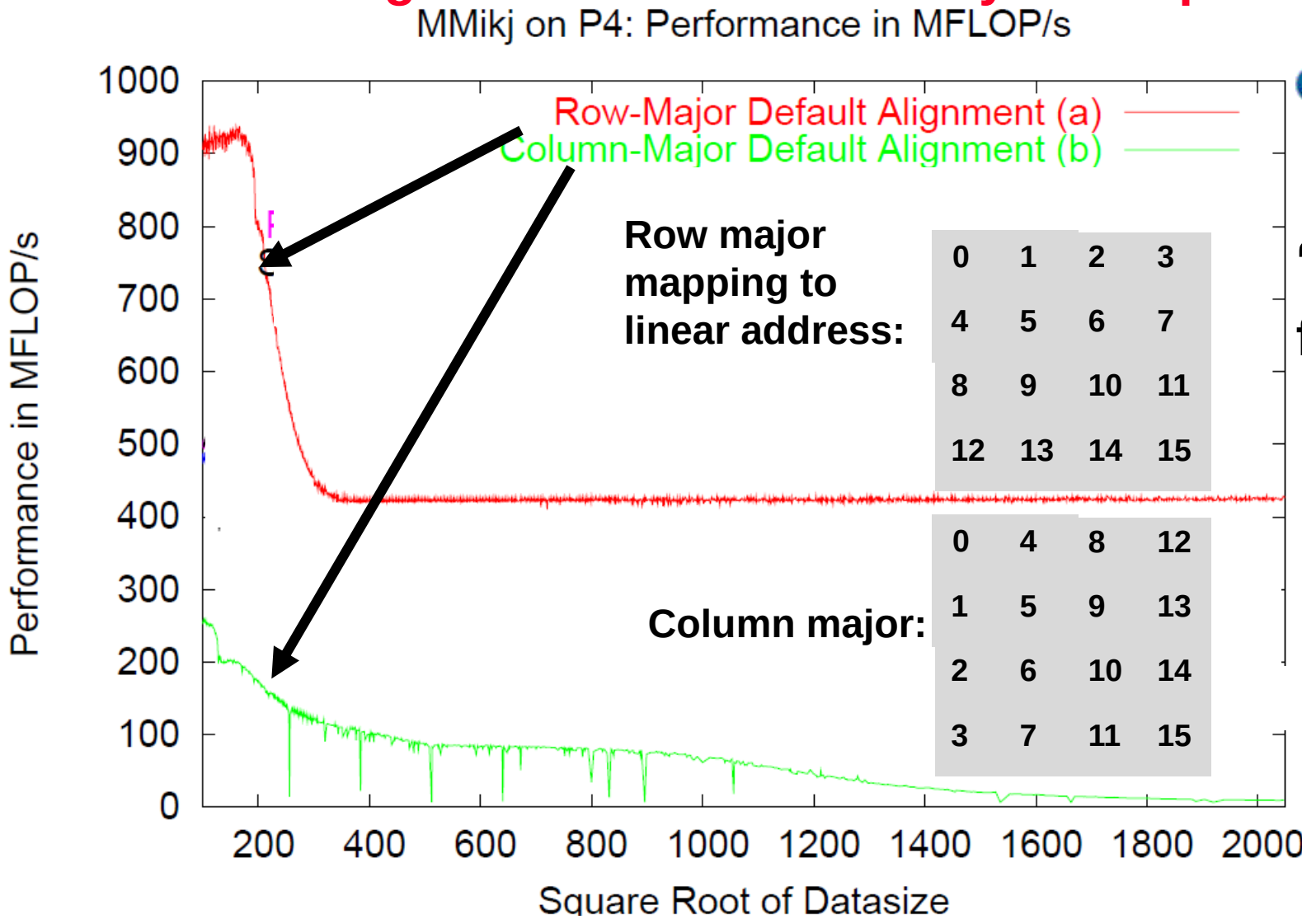
- **Column-major storage layout (default for Fortran):**





Problem size: 192 doubles, 1536 bytes per row

Permuting multidimensional arrays to improve spatial locality



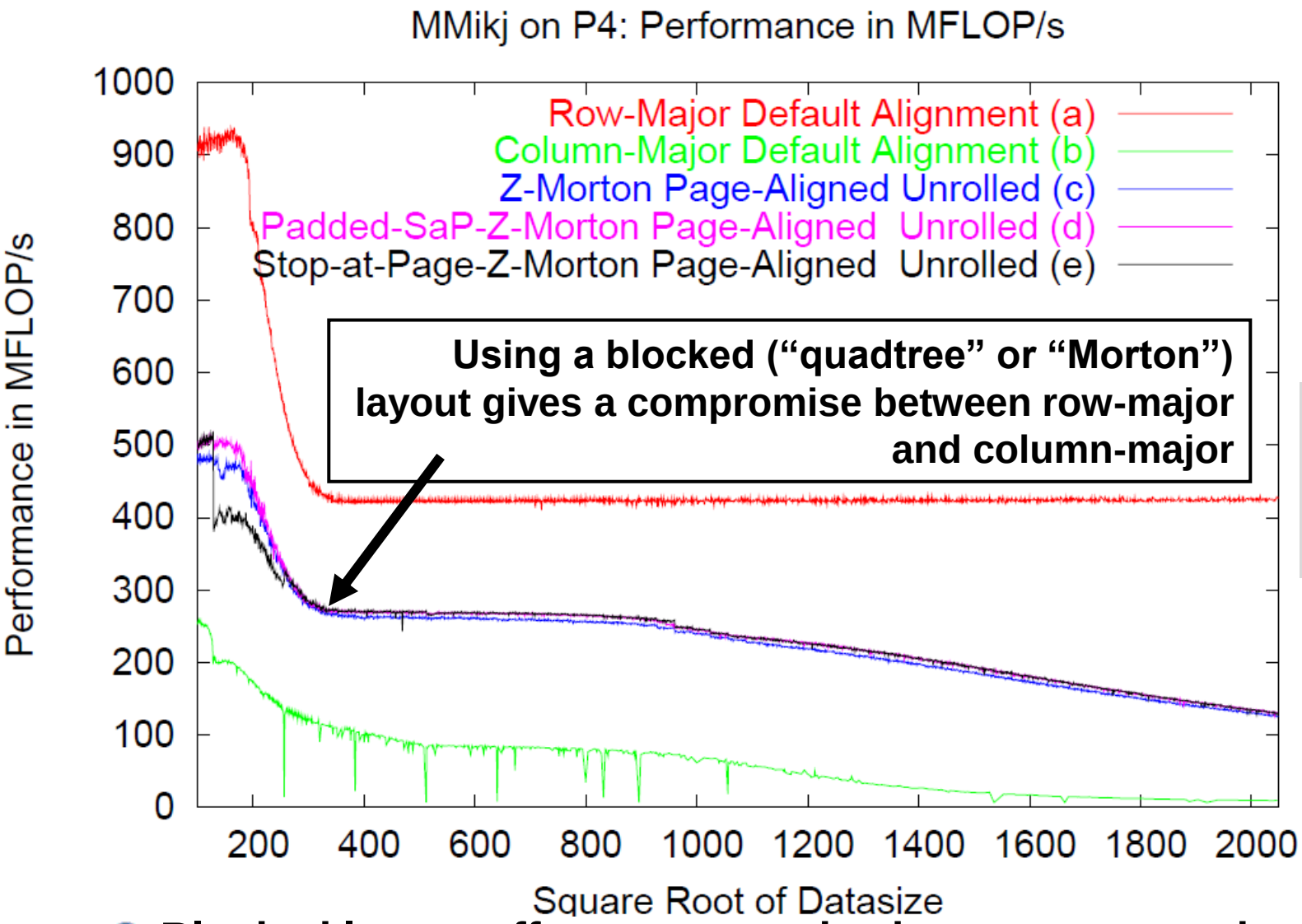
- Matrix-matrix multiply on Pentium 4

"ikj" variant:

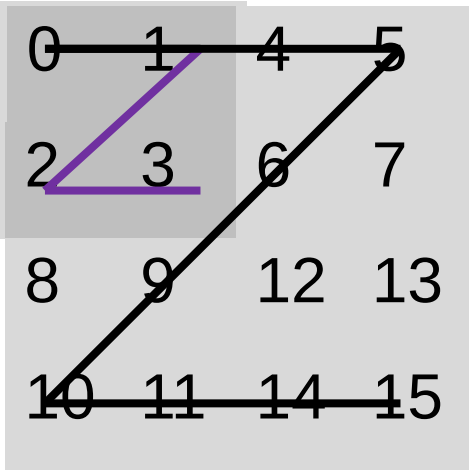
```
for i
 for k
  for j
   C[ij]+=A[ik]
        *B[kj]
```

- Traverses B and C in row-major order
- Which is great if the data is stored in row-major order
- If data is actually in column-major order…

Permuting multidimensional arrays to improve spatial locality



A variant of Morton-order layout is used for texture caching in some GPUs



Morton-order mapping to linear address

- **Blocked layout offers compromise between row-major and column-major**
- **Some care is needed in optimising address calculation to make this work (Jeyan Thiyagalingam's Imperial PhD thesis)**

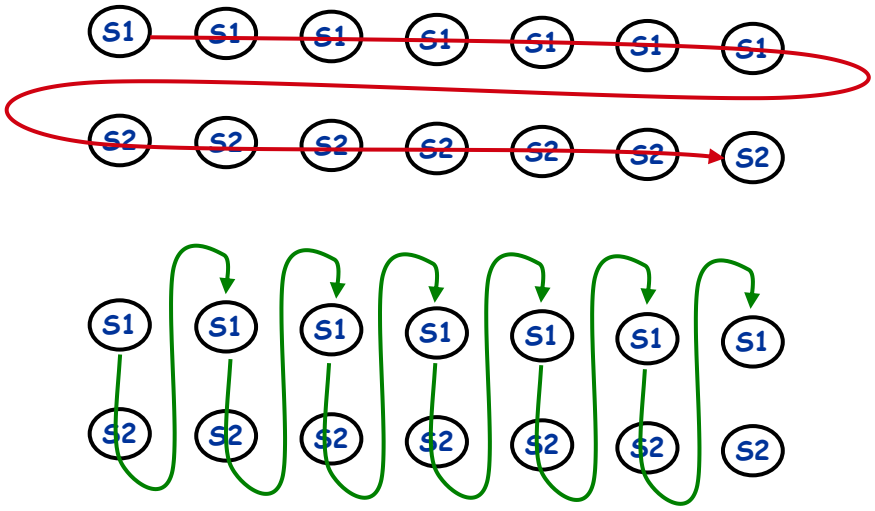
Loop Interchange: example

```
/* Before */
for (k = 0; k < 100; k = k+1)
    for (j = 0; j < 100; j = j+1)
        for (i = 0; i < 5000; i = i+1)
            x[i][j] = 2 * x[i][j];
/* After */
for (k = 0; k < 100; k = k+1)
    for (i = 0; i < 5000; i = i+1)
        for (j = 0; j < 100; j = j+1)
            x[i][j] = 2 * x[i][j];
```

Sequential accesses: instead of striding through memory every 100 words; improved spatial locality

Loop Fusion: example

```
/* Before */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
   S1:  a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
   S2:  d[i][j] = a[i][j] + c[i][j];
```




```
/* After fusion */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
   {S1: a[i][j] = 1/b[i][j] * c[i][j];
    S2: d[i][j] = a[i][j] + c[i][j];}
```

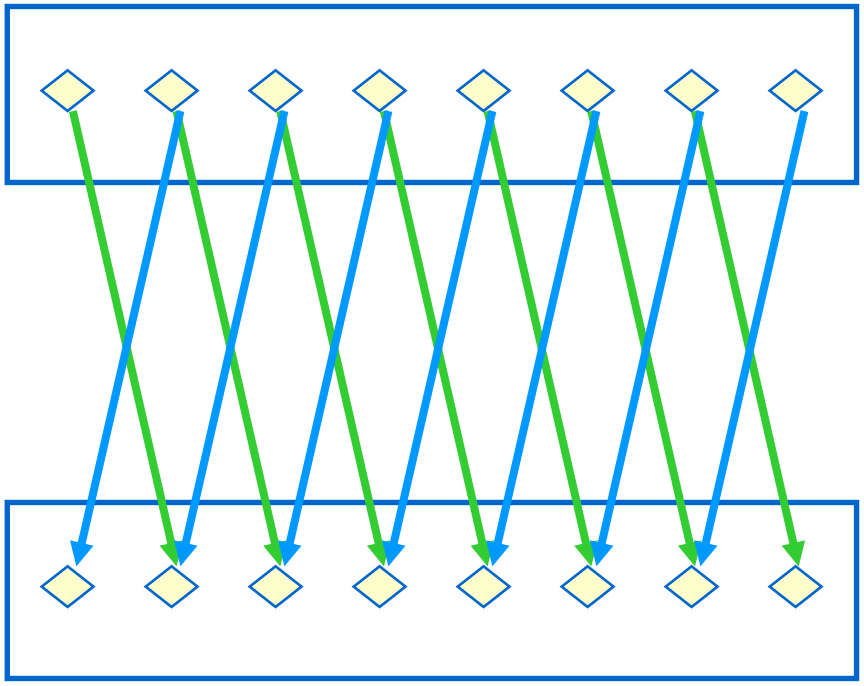
```
/* After array contraction */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
   {    cv = c[i][j];
     S1: a = 1/b[i][j] * cv;
     S2: d[i][j] = a + cv;}
```

2 misses per access to a & c vs. one miss per access; improve spatial locality

The real payoff comes if fusion enables Array Contraction: values transferred in scalar instead of via array

Fusion is not always so simple

- Dependences might not align nicely
- Example: one-dimensional convolution filters



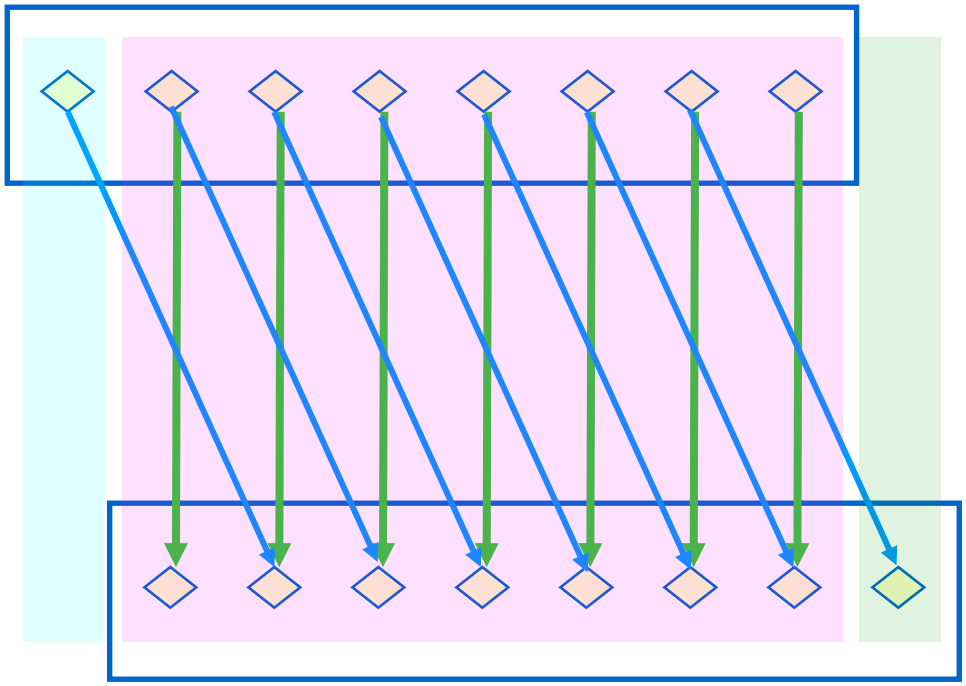
```
for (i=1; i<N; i++)
  V[i] = (U[i-1] + U[i+1])/2
```

```
for (i=1; i<N; i++)
  W[i] = (V[i-1] + V[i+1])/2
```

- “Stencil” loops are not directly fusable

Loop fusion – code expansion

- We make them fusable by shifting:



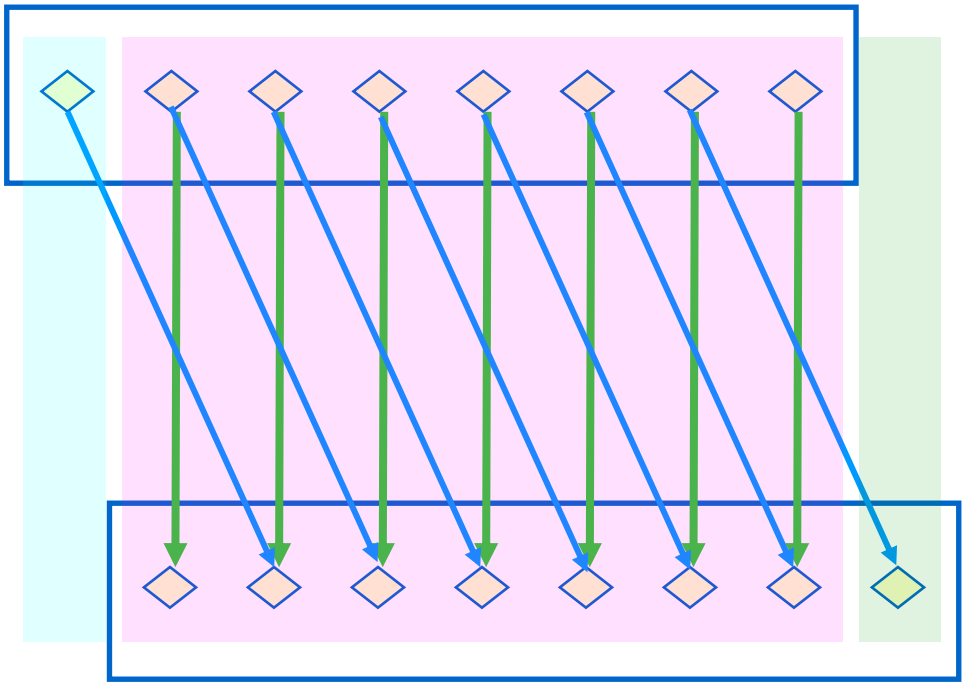
```
V[1] = (U[0] + U[2])/2
for (i=2; i<N; i++) {
  V[i] = (U[i-1] + U[i+1])/2
  W[i-1] = (V[i-2] + V[i])/2
}
W[N-1] = (V[N-2] + V[N])/2
```

- The middle loop is fusable
- We get lots of little edge bits

Loop fusion – code expansion

This transformation is important in image-processing filters, finite difference solvers, and convolutional neural networks

- We make them fusable by shifting:



```
V[1] = (U[0] + U[2])/2
for (i=2; i<N; i++) {
  V[i%4] = (U[i-1] + U[i+1])/2
  W[i-1] = (V[(i-2)%4] + V[i%4])/2
}
W[N-1] = (V[(N-2)%4] + V[N%4])/2
```

- The middle loop is fusable
- We get lots of little edge bits
  - Contraction is trickier
  - We need the last *two* Vs
  - We need 3 V locations
  - Quicker to round up to four

Summary

We can reduce the miss rate at the software level …..

- **By using prefetch instructions**
  - **If they work better than predictive prefetch hardware**
- **By transforming storage layout**
  - **Might help with *spatial* locality**
  - **Might help with associativity conflicts**
  - **Can't help with *temporal* locality**
- **Storage layout optimisations are disruptive – they affect all the code that might use that data**
- **Loop interchange, fusion, tiling**
  - **Can get *really* messy to implement by hand**
  - **Can lead to a large space of possible schedules – it can be hard to know what will work best**
  - **Loop fusion can be very powerful but often breaks abstraction boundaries**

Further reading

Algorithms and locality: cache-oblivious algorithms:

- https://en.wikipedia.org/wiki/Cache-oblivious_algorithm

Compilers that optimise for locality:

- Michael E. Wolf and Monica S. Lam. 1991. **A data locality optimizing algorithm**. PLDI91.
- Kathryn S. McKinley, Steve Carr, and Chau-Wen Tseng. 1996. **Improving data locality with loop transformations**. ACM Trans. Program. Lang. Syst. 18, 4 (July 1996)
- Uday Bondhugula, Albert Hartono, J. Ramanujam, and P. Sadayappan. 2008. **A practical automatic polyhedral parallelizer and locality optimizer**. PLDI08

Programming Abstractions for Data Locality

- https://sites.google.com/a/lbl.gov/padal-workshop/

Optimisations for convolutional neural networks

- Yizhi Liu, Yao Wang, Ruofei Yu, Mu Li, Vin Sharma, Yida Wang. **Optimizing CNN model inference on CPUs**. USENIX ATC'19.