Cache Models and Program Transformations
machines with caches.

Let us look at how to make some of the kernels run well on
understand what causes cache misses (cold, capacity, conflict).

We have studied caches and virtual memory, and we
isolated key kernels (MVM, MMM, linear system solvers, ...).

We have looked at computational science applications, and

Goal of this lecture
What is, how do we improve the miss rate?

What transformations can we do to improve performance?

For different cache models?

Can we predict the miss ratio of different variations of this program?

We want to study two questions.

Total number of references $N^2 = N \sum (x \times \tau t) \forall \tau \forall \tau t = (t \forall \tau t$ $\forall \tau t = j \forall \tau t$

For $j = i \forall \tau t$

For $i = i \forall \tau t$

Code:

Matrix-vector product
Problem size (size of arrays)

- small cache model: miss if reuse distance is some function of
  - large cache model: no capacity misses

We will look at two extremes

- LRU Replacement Strategy

(Fully associative cache (so no conflict misses)

Cache model:

- of distinct cache lines referenced between \( t_1 \) and \( t_2 \),
- reuse distance \( distance(t_1, t_2) \) is the number
- in some memory stream, reuse \( distance(t_1, t_2) \) is the number
- line in some memory stream, reuse

Reuse Distance: If \( t_1 \) and \( t_2 \) are two references to the same cache
Miss ratio \( \frac{N^2/(N + 2N^2)}{4} \leftarrow 0.5 \)

- Vector \( y \): \( N \) cold misses
- Vector \( x \): \( N \) cold misses, (1) capacity misses
- Matrix \( A \): \( N^2 \) cold misses

Misses:

Small cache: assume cache can hold fewer than \((2N+2)\) numbers

- Cache line size = 1 floating-point number
- LRU replacement strategy
- Fully associative cache (no conflict misses)

Cache model:

Scenario 1
Large cache model: cache can hold \((2N+2)\) numbers or more.
It is easy to show that miss ratios are identical to Scenario I.

\[(\tau)^x = (\tau)^y\]
\[
\lambda_p\text{ for } i, j \in N
\]

\[\text{for } i = 1, n\]

Code: walk matrix A by columns

Same cache model as Scenario I but different code.

Scenario II
Let us assume $A$ is stored in row-major order.

$$(\tau x, y) + (\tau y, z) = (\tau y, z)$$

for $i' = 1, N$

for $i' = 1, N$

**Code** (original) !? Loop order

can exploit spatial locality

cache line size = blocksize of floating point numbers

**LRU Replacement Strategy**

fully associative cache (no conflict misses)

**Cache model**

Scenario III
Transition from small cache to large cache when \( c = 2N + 2p \).

\[
\frac{p}{I_{2N}} \left( \frac{1}{1 + \frac{q}{N}} \right) \frac{q}{N} = \frac{1}{I_{2N}} \frac{q}{N}
\]

- Miss ratio
- Misses:
- Large cache: \( \frac{p}{I_{2N}} \left( \frac{1}{1 + \frac{q}{N}} \right) \frac{q}{N} \)
- Vector \( x \): Misses
- Vector \( x \): Misses
- Matrix \( A \): Misses

Small cache: \( \frac{p}{I_{2N}} \left( \frac{1}{1 + \frac{q}{N}} \right) \frac{q}{N} \)

- Miss ratio
- Misses:
- Large cache: \( \frac{p}{I_{2N}} \left( \frac{1}{1 + \frac{q}{N}} \right) \frac{q}{N} \)
- Vector \( x \): Misses
- Vector \( x \): Misses
- Matrix \( A \): Misses
Small/Large transition size $= 2000$

- Small cache miss ratio $= 0.12$
- Large cache miss ratio $= \frac{1}{16} = 0.06$
- Cache size $= 32$ KB $\iff c = 4$ K
- Line size $= 32$ bytes $\iff b = 4$

Let us plug in some numbers for SCI Octane:

**Miss ratios for Scenario III**

$q$ number of CPs in one cache line
$c =$ size of cache in # of CPs

Roughly, this is when $N > \frac{c}{2}$. 
Note: we are not walking over \( A \) in memory layout order

\[
(\mathbf{x} \star f(x,t)) \star f(t) + (\mathbf{A} \star f(x,t)) = (\mathbf{A} \star f(x,t))
\]

for \( t = 1, N \)

for \( t = 1, N \)

Code: if loop order

(can exploit spatial locality)

- cache line size = \( b \) floating-point numbers
- LRU replacement strategy
- fully associative cache (no conflict misses)

Cache model:

Scenario 1

\[
\begin{array}{c}
\mathbf{A} \\
\mathbf{x} \\
\end{array} \\
\begin{array}{c}
\mathbf{\Lambda} \\
\mathbf{\Lambda} \\
\end{array}
\]
Transition from small cache to large cache when \( c < bN + N + p \)

\[
\frac{q}{I + 1} \left( \frac{q}{I + 1} \right) = \frac{q}{N} \quad \text{vector } \vec{x} \\
\frac{q}{N} \quad \text{cold misses} \\
\frac{q}{N} \quad \text{vector } \vec{x} \\
\frac{q}{N} \quad \text{cold misses} \\
\frac{q}{N} \quad \text{matrix } A \\
\frac{q}{N} \quad \text{cold misses}
\]

Large cache:

\[
(0.25)^* \frac{q}{I + 1} + \frac{q}{I + 1} + (q/4N + q/8N + q/25N) \quad \text{cold misses} \\
\frac{q}{I - N} N + \text{capacity misses} \\
\frac{q}{I} N \quad \text{vector } \vec{x} \\
\frac{q}{N} \quad \text{cold misses} \\
\frac{q}{N} \quad \text{vector } \vec{x} \\
\frac{q}{N} \quad \text{cold misses} \\
\frac{q}{N} \quad \text{matrix } A \\
\frac{q}{N} \quad \text{cold misses}
\]

Small cache:
Let us plug some numbers in for SGI Octane:

* Small/large transition size = 800
* Small cache miss ratio = 0.31
* Large cache miss ratio = 1 − 0.16 = 0.84

\[
\text{Cache size} = 32 \text{ KB} \quad c = 4K
\]
\[
\text{Line size} = 32 \text{ bytes} \quad b = 4
\]

**Miss ratios for Scenario IY**

\[
\text{Number of FP's in one cache line} = b
\]
\[
\text{Number of cache in # of FP's} = c
\]

Roughly, this is when \( c < \left( \frac{q}{b+1} \right) N \).
(\tau^*)^x (\tau^* \tau) \mathbb{A} + (\tau^* \tau) \mathbb{A} = (\tau^* \tau) \mathbb{A} \\
\text{for } \mathbb{A} \mathbb{A} = \mathbb{A} \mathbb{A} \\
\text{for } \mathbb{A} \mathbb{A} = \mathbb{A} \mathbb{A} \\
\text{for } \mathbb{A} \mathbb{A} = \mathbb{A} \mathbb{A} \\
\text{for } \mathbb{A} \mathbb{A} = \mathbb{A} \mathbb{A} \\
The Code:

\text{Scenario V: Blocked Code}
problem size.

For octane, we have miss ratio is roughly 0.06 independent of
\[
q/B: \text{vector}
\]
\[
q/B: \text{vector}
\]
\[
q/B: \text{vector}
\]
\[
q/B: \text{matrix} \ A: \text{vector}
\]

Misses within a block:

A significant under-estimate of the right value for block size.
\[
\hat{B} \geq B \text{ (which is determined block size of block computation) (2B) to}
\]
while executing code within block (2B = c).

Pick block size B so that you effectively have large cache model.
Miss ratio predictions for MVM point and blocked codes

Putting it all together for SGI Octane
than predicted.
so transition from large to small cache model should happen sooner.
Conflict misses will have the effect of reducing effective cache size.
We have assumed a fully associative cache.
Predictions agree reasonably well with experiments.

Experimental Results on SCI Octane

![Graph showing MVM L1-cache miss rates with different code blocks and sizes]
The text on the image appears to be a mathematical or technical document, but it is not legible due to the quality of the image. The text seems to be discussing equations or formulas, possibly related to algebra or mathematical transformations.

Key transformations

Loop permutation

Strip-mining

Loop tripling = strip-mining + interchanging
in some codes.

Warning: therefore loop permutation and tiling may be illegal
 instances are executed; permutation (and therefore tiling) do.

Stripping does not change the order in which loop body
 interchange. It is sometimes called stripping-and-interchange.

Tiling/blocking can be viewed as stripping followed by
Cache model: assume cache line size is \( b \) P's

\[
C(i', j') = C(i', j') + A(i', j') * B(k, j')
\]

for \( k = 1 \) to \( N \)

for \( j = 1 \) to \( N \)

for \( i = 1 \) to \( N \)

Code:

Matrix-matrix Product
Miss ratio: 0.25 ≤ \( q/(I + q) \) ≤ 0.75

Total number of references: \( \forall N \equiv q/(I + q) N \)

Total number of misses: \( q/(I + q) N \leftarrow (I + (I + q) N) * q/N \)

\( I + (I + q) N = C \)

Total number of misses per cache line of \( C \):

- Matrix \( \mathcal{C} \):
  - \( I \)
- Matrix \( \mathcal{B} \):
  - \( N * q \)
- Matrix \( \mathcal{A} \):
  - \( (q/N) * q \)

Misses for each cache line of \( C \):

Small cache:
Intuition: Lot of data reuse, so once matrices all fit into cache, code goes full blast.

For large cache model, miss ratio decreases as the size of the cache increases:

$$\text{Miss ratio} = \frac{3}{N} \cdot \frac{4bN}{pN} = 0.75$$

$$\text{Cold misses} = \frac{q}{N} \cdot 3$$

Large cache:
So \( N^2 + 2N + q \geq c \)

3. a whole row of \( A \): \( N \) hosts
2. a whole row of \( C \): \( N \) hosts
1. all of \( B \): \( N^2 \) hosts

Between successive accesses to same cache line of \( B \), we touch

Reuse distance is largest for elements of \( B \).

Answer depends on the loop order, let us look at if:

there are capacity misses?

Transition out of large cache model: How large can \( N \) get before
For the Octane, this gives transition size = 36, which is quite a few ways off.

\[
\frac{c}{\sqrt{3}} > N
\]

Had we used full data set, we obtain 3N^2 > c which gives

\[
\text{For Octane, } c = 4k, \text{ so transition size } = 64
\]

\[
(\text{subject}) \approx 1 - (I + q - c)^N > N
\]

Roughly, this gives
For some of the versions, there is a medium cache model - see the figure.

For large values of $N$, there are three asymptotic miss ratios. Similarly, you can figure out the performance for all $6$ versions of WIM.
Question: What should the order of the outer loops be?

Choose B so we have large cache model when executing block code.

\[
\lambda(\tau) (\tau) \lambda = (\tau) (\tau) \lambda
\]

for \( k = p_k + B - 1 \), \( N \)

for \( j = p_j + B - 1 \), \( N \)

for \( i = p_i + B - 1 \), \( N \)

for \( B = 1 \), \( N \)

for \( B = 1 \), \( N \)

for \( B = 1 \), \( N \)

Code:

\[
\begin{array}{c}
\text{b} \\
\text{c} \\
\text{A} \\
\text{B} \\
\end{array}
\]

Blocked code:
can obtain from blocking alone will be more.

As before, we have ignored conflict misses, so actual miss ratio we

Since $\beta = 64$, miss ratio is roughly $0.003$.

Miss ratio of blocked code $= 0.75/\beta$. 
How does a compiler which transformation to apply?

How can a compiler determine legality of a transformation?

Neither transformation is necessarily legal or beneficial.

Permutation and tilting:

Key compiler transformations for perfectly-nested loops:

Assignment statements are contained in the innermost loop.

A perfectly-nested loop nest is a loop nest in which all

perfectly-nested loop nests:

Distinguishing characteristic of MVM and WM:

Blocking can improve cache performance dramatically.

As usually written, these kernels have poor cache performance.

We have looked at two kernels: MVM and WM.

Summary