Cache Models and Program Transformations
Goal of lecture

• Develop abstractions of real caches for understanding program performance
• Study the cache performance of matrix-vector multiplication (MVM)
  – simple but important computational science kernel
• Understand MVM program transformations for improving performance
Matrix-vector product

- **Code:**
  
  ```
  for i = 1,N
    for j = 1,N
      y(i) = y(i) + A(i,j)*x(j)
  ```

- **Total number of references = 4N^2**
  - This assumes that all elements of A, x, y are stored in memory
  - Smart compilers nowadays can register-allocate y(i) in the inner loop
  - You can get this effect manually
    ```
    for i = 1,N
      temp = y(i)
      for j = 1,N
        temp = temp + A(i,j)*x(j)
      y(i) = temp
    ```
  - To keep things simple, we will not do this but our approach applies to this optimized code as well
Cache abstractions

• Real caches are very complex
• Science is all about tractable and useful abstractions (models) of complex phenomena
  – models are usually approximations
• Can we come up with cache abstractions that are both tractable and useful?
• Focus:
  – two-level memory model: cache + memory
Stack distance

- $r_1, r_2$: two memory references
  - $r_1$ occurs earlier than $r_2$
- stackDistance$(r_1, r_2)$: number of distinct cache lines referenced between $r_1$ and $r_2$
- Stack distance was defined by Mattson et al (IBM Systems Journal paper)
  - arguably the most important paper in locality
Modeling approach

- First approximation:
  - ignore conflict misses
  - only cold and capacity misses

- Most problems have some notion of “problem size”
  - (eg) in MVM, the size of the matrix (N) is a natural measure of problem size

- Question: how does the miss ratio change as we increase the problem size?

- Even this is hard, but we can often estimate miss ratios at two extremes
  - large cache model: problem size is small compared to cache capacity
  - small cache model: problem size is large compared to cache capacity
  - we will define these more precisely in the next slide.
Large and small cache models

• Large cache model
  – no capacity misses
  – only cold misses

• Small cache model
  – cold misses: first reference to a line
  – capacity misses: possible for succeeding references to a line
    • let \( r_1 \) and \( r_2 \) be two successive references to a line
    • assume \( r_2 \) will be a capacity miss if \( \text{stackDistance}(r_1,r_2) \) is some function of problem size
      • argument: as we increase problem size, the second reference will become a miss sooner or later

• For many problems, we can compute
  – miss ratios for small and large cache models
  – problem size transition point from large cache model to small cache model
MVM study

• We will study five scenarios
  – Scenario I
    • i,j loop order, line size = 1 number
  – Scenario II
    • j,i loop order, line size = 1 number
  – Scenario III
    • i,j loop order, line size = b numbers
  – Scenario IV
    • j,i loop order, line size = b numbers
  – Scenario V
    • blocked code, line size = b numbers
Scenario I

- **Code:**
  ```plaintext```
  for i = 1,N
      for j = 1,N
          y(i) = y(i) + A(i,j)*x(j)
  ```plaintext```
- **Inner loop is known as DDOT in NA literature if working on doubles:**
  - Double-precision DOT product
- **Cache line size**
  - 1 number
- **Large cache model:**
  - **Misses:**
    - A: \(N^2\) misses
    - x: N misses
    - y: N misses
    - Total = \(N^2+2N\)
    - Miss ratio = \((N^2+2N)/4N^2\)
    ~ 0.25 + 0.5/N


**Scenario I (contd.)**

Address stream: \[ y(1) \quad A(1,1) \quad x(1) \quad y(1) \quad y(1) \quad A(1,2) \quad x(2) \quad y(1) \ldots \quad y(1) \quad A(1,N) \quad x(N) \quad y(1) \quad y(2) \quad A(2,1) \quad x(1) \quad y(2) \]

- **Small cache model:**
  - A: \(N^2\) misses
  - x: \(N + N(N-1)\) misses (reuse distance=\(O(N)\))
  - y: \(N\) misses (reuse distance=\(O(1)\))
  - Total = \(2N^2+N\)
  - Miss ratio = \((2N^2+N)/4N^2\)
    \(~0.5 + 0.25/N\)

- **Transition from large cache model to small cache model**
  - As problem size increases, when do capacity misses begin to occur?
  - Subtle issue: depends on replacement policy (see next slide)
Scenario I (contd.)

Address stream: \( y(1) \ A(1,1) \ x(1) \ y(1) \ A(1,2) \ x(2) \ y(1) \ldots \ y(1) \ A(1,N) \ x(N) \ y(1) \ y(2) \ A(2,1) \ x(1) \ y(2) \)

- **Question:** as problem size increases, when do capacity misses begin to occur?
- **Depends on replacement policy:**
  - **Optimal replacement:**
    - do the best job you can, knowing everything about the computation
    - only \( x \) needs to be cache-resident
    - elements of \( A \) can be “streamed in” and tossed out of cache after use
    - So we need room for \( (N+2) \) numbers
    - Transition: \( N+2 > C \) \( \Rightarrow \) \( N \sim C \)
  - **LRU replacement**
    - by the time we get to end of a row of \( A \), first few elements of \( x \) are “cold” but we do not want them to be replaced
    - Transition: \( (2N+2) > C \) \( \Rightarrow \) \( N \sim C/2 \)
- **Note:**
  - optimal replacement requires perfect knowledge about future
  - most real caches use LRU or something close to it
  - some architectures support “streaming”
    - in hardware
    - in software: hints to tell processor not to cache certain references
• Jump from large cache model to small cache model will be more gradual in reality because of conflict misses
Scenario II

- Code:
  
  $$\text{for } j = 1, N$$
  $$\text{for } i = 1, N$$
  $$y(i) = y(i) + A(i,j) \times x(j)$$

- Inner loop is known as AXPY in NA literature
  
  $$y = \alpha \cdot x + y$$

- Miss ratio picture exactly the same as Scenario I
  - roles of $x$ and $y$ are interchanged
Scenario III

- **Code:**
  
  ```
  for i = 1,N
    for j = 1,N
      y(i) = y(i) + A(i,j)*x(j)
  ```

- **Cache line size**
  - \(b\) numbers

- **Large cache model:**
  - Misses:
    - \(A\): \(N^2/b\) misses
    - \(x\): \(N/b\) misses
    - \(y\): \(N/b\) misses
    - Total = \((N^2+2N)/b\)
    - Miss ratio = \((N^2+2N)/4bN^2\)
    - \(~ 0.25/b + 0.5/bN\)
Scenario III (contd.)

Address stream: \( y(1) A(1,1) x(1) y(1) y(1) A(1,2) x(2) y(1) \ldots y(1) A(1,N) x(N) y(1) \)

\[ y(2) A(2,1) x(1) y(2) \]

• Small cache model:
  \( A: N^2/b \) misses
  \( x: N/b + N(N-1)/b \) misses (reuse distance=\( O(N) \))
  \( y: N/b \) misses (reuse distance=\( O(1) \))
  Total = \( (2N^2+N)/b \)
  Miss ratio = \( (2N^2+N)/4bN^2 \)
  \( \sim 0.5/b + 0.25/bN \)

• Transition from large cache model to small cache model
  As problem size increases, when do capacity misses begin to occur?
  LRU: roughly when \( (2N+2b) = C \)
  \( N \sim C/2 \)
  Optimal: roughly when \( (N+2b) \sim C \Rightarrow N \sim C \)

• So miss ratio picture for Scenario III is similar to that of Scenario I but the y-axis is scaled down by \( b \)

• Typical value of \( b = 4 \) (SGI Octane)
Miss ratio graph

- Jump from large cache model to small cache model will be more gradual in reality because of conflict misses.
Scenario IV

- **Code:**
  
  ```
  for j = 1,N
  for i = 1,N
      y(i) = y(i) + A(i,j)*x(j)
  ```

- **Large cache model:**
  - Same as Scenario III

- **Small cache model:**
  - Misses:
    - A: \(N^2\)
    - x: \(N/b\)
    - y: \(N/b + N(N-1)/b = N^2/b\)
    - Total: \(N^2(1+1/b) + N/b\)
    - Miss ratio = \(0.25(1+1/b) + 0.25/N\)

- **Transition from large cache to small cache model**
  - LRU: \(Nb + N + b = C \Rightarrow N \sim C/(b+1)\)
  - optimal: \(N + 2b \sim C \Rightarrow N \sim C\)

- **Transition happens much sooner than in Scenario III (with LRU replacement)**
Miss ratios

\[ \text{Miss ratio} = \frac{0.25(1+1/b)}{C/(b+1)} \]

\[ \text{Miss ratio} = \frac{0.50/b}{C/2} \]

\[ \text{Miss ratio} = \frac{0.75/b}{N} \]

\[ \text{Miss ratio} = \frac{0.25/b}{C/(b+1)} \]

\[ \text{Miss ratio} = \frac{DAXPY}{DDOT} \]
Scenario V

• Intuition: perform blocked MVM so that data for each blocked MVM fits in cache
  – One estimate for B: all data for block MVM must fit in cache
    \[ B^2 + 2B \approx C \]
    \[ B \approx \sqrt{C} \]
  – Actually we can do better than this

• Code: blocked code
  
  for \( bi = 1, N, B \)
  for \( bj = 1, N, B \)
  
  for \( i = bi, \min(bi+B-1, N) \)
  for \( j = bj, \min(bj+B-1, N) \)
  
  \[ y(i) = y(i) + A(i,j) \times x(j) \]

• Choose block size B so
  – you have large cache model while executing block
  – B is as large as possible (to reduce loop overhead)
  – for our example, this means \( B \approx \frac{c}{2} \) for row-major order of storage and LRU replacement

• Since entire MVM computation is a sequence of block MVMs, this means miss ratio will be \( 0.25/b \) independent of N!
Scenario V (contd.)

- **Code: blocked code**
  
  ```
  for bi = 1,N,B
    for bj = 1,N,B
      for i = bi,min(bi+B-1,N)
        for j = bj,min(bj+B-1,N)
          y(i)=y(i)+A(i,j)*x(j)
  
  Better code: interchange the two outermost loops and fuse bi and i loops
  for bj = 1,N,B
    for i = 1,N
      for j = bi,min(bi+B-1,N)
        y(i)=y(i)+A(i,j)*x(j)
  ```

This has the same memory behavior as doubly-blocked loop but less loop overhead.
Miss ratios

\[
\text{Miss ratio} = \frac{0.25(1+1/b)}{C/(b+1)}
\]

- 0.25(1+1/b)
- 0.75/b
- 0.50/b
- 0.25/b

- Blocked
- DDOT
- DAXPY

- C/(b+1)
- C/2
Key transformations

- **Loop permutation**
  \[
  \text{for } i = 1,N \\
  \text{for } j = 1,N \\
  \text{for } j = 1,N \\
  S \\
  \Rightarrow \\
  \text{for } i = 1,N \\
  \text{for } i = 1,N \\
  S
  \]

- **Strip-mining**
  \[
  \text{for } i = 1,N \\
  \text{for } bi = 1,N,B \\
  S \\
  \Rightarrow \\
  \text{for } i = bi, \min(bi+B-1,N) \\
  \text{for } i = bi, \min(bi+B-1,N) \\
  S
  \]

- **Loop tiling = strip-mine and interchange**
  \[
  \text{for } i = 1,N \\
  \text{for } j = 1,N \\
  S \\
  \Rightarrow \\
  \text{for } bi = 1,N,B \\
  \text{for } j = 1,N \\
  \text{for } i = bj, \min(bj+B-1,N) \\
  S
  \]
Notes

• Strip-mining does not change the order in which loop body instances are executed
  – so it is always legal

• Loop permutation and tiling do change the order in which loop body instances are executed
  – so they are not always legal

• For MVM and MMM, they are legal, so there are many variations of these kernels that can be generated by using these transformations
  – different versions have different memory behavior as we have seen
Matrix multiplication

• We have studied MVM in detail.
• In dense linear algebra, matrix-matrix multiplication is more important.
• Everything we have learnt about MVM carries over to MMM fortunately, but there are more variations to consider since there are three matrices and three loops.
DO I = 1, N // row-major storage
  DO J = 1, N
    DO K = 1, N
      C(I,J) = C(I,J) + A(I,K)*B(K,J)

• Three loops: I,J,K
• You can show that all six permutations of these three loops compute the same values.
• As in MVM, the cache behavior of the six versions is different

IJK version of matrix multiplication
DO I = 1, N  //row-major storage  
   DO J = 1, N  
      DO K = 1, N  
         C(I,J) = C(I,J) + A(I,K)*B(K,J)  
      \end{verbatim}

\textbf{IJK version of matrix multiplication}

- \textbf{K loop innermost}
  - A: good spatial locality
  - C: good temporal locality
- \textbf{I loop innermost}
  - B: good temporal locality
- \textbf{J loop innermost}
  - B,C: good spatial locality
  - A: good temporal locality
- \textbf{So we would expect IKJ/KIJ versions to perform best, followed by IJK/JIK, followed by JKI/KJI}
MMM miss ratios (simulated)

L1 Cache Miss Ratio for Intel Pentium III
- MMM with N = 1…1300
- 16KB 32B/Block 4-way 8-byte elements
Observations

• Miss ratios depend on which loop is in innermost position
  – so there are three distinct miss ratio graphs
• Large cache behavior can be seen very clearly and all six version perform similarly in that region
• Big spikes are due to conflict misses for particular matrix sizes
  – notice that versions with J loop innermost have few conflict misses (why?)
IJK version

DO I = 1, N  // row-major storage
  DO J = 1, N
    DO K = 1, N
      C(I,J) = C(I,J) + A(I,K)*B(K,J)
  END DO
END DO

• Large cache scenario:
  – Matrices are small enough to fit into cache
  – Only cold misses, no capacity misses
  – Miss ratio:
    • Data size = 3 N^2
    • Each miss brings in b floating-point numbers
    • Miss ratio = 3 N^2 / b*4N^3 = 0.75/bN  (eg) 0.019 (b = 4, N=10)
DO I = 1, N//row-major storage
    DO J = 1, N
        DO K = 1, N
            C(I,J) = C(I,J) + A(I,K)*B(K,J)

• Large cache scenario:
  – Matrices are small enough to fit into cache
  – Only cold misses, no capacity misses
  – Miss ratio:
    • Data size = 3 N^2
    • Each miss brings in b floating-point numbers
    • Miss ratio = 3 N^2 /b*4N^3 = 0.75/bN = 0.019 (b = 4,N=10)
**IJK version (small cache)**

```plaintext
DO I = 1, N
  DO J = 1, N
    DO K = 1, N
      C(I,J) = C(I,J) + A(I,K)*B(K,J)
  
  • Small cache scenario:
    – Matrices are large compared to cache
      • stack distance is not O(1) => miss
    – Cold and capacity misses
    – Miss ratio:
      • C: N^2/b misses (good temporal locality)
      • A: N^3 /b misses (good spatial locality)
      • B: N^3 misses (poor temporal and spatial locality)
      • Miss ratio \( \rightarrow 0.25 \frac{(b+1)}{b} = 0.3125 \) (for \( b = 4 \))
```
Miss ratios for other versions

\[
\begin{align*}
\text{DO } I & = 1, N \text{//row-major storage} \\
\text{DO } J & = 1, N \\
\text{DO } K & = 1, N \\
C(I,J) &= C(I,J) + A(I,K) \times B(K,J)
\end{align*}
\]

- **K loop innermost**
  - A: good spatial locality
  - C: good temporal locality  \(0.25(b+1)/b\)
- **I loop innermost**
  - B: good temporal locality  \((N^2/b + N^3 +N^3)/4N^3 \rightarrow 0.5\)
- **J loop innermost**
  - B,C: good spatial locality  \((N^3/b + N^3/b + N^2/b)/4N^3 \rightarrow 0.5/b\)
  - A: good temporal locality

- **So we would expect IKJ/KIJ versions to perform best, followed by IJK/JIK, followed by JKI/KJI**
MMM experiments

L1 Cache Miss Ratio for Intel Pentium III

- MMM with $N = 1 \ldots 1300$
- 16KB 32B/Block 4-way 8-byte elements

Can we predict this?
Transition out of large cache

DO I = 1, N//row-major storage
DO J = 1, N
DO K = 1, N
C(I,J) = C(I,J) + A(I,K)*B(K,J)

• Find the data element(s) that are reused with the largest stack distance
• Determine the condition on N for that to be less than C
• For our problem:
  – $N^2 + N + b < C$ (with optimal replacement)
  – $N^2 + 2N < C$ (with LRU replacement)
  – In either case, we get $N \sim \sqrt{C}$
  – For our cache, we get $N \sim 45$ which agrees quite well with data
Block code

As in blocked MVM, we actually need to stripmine only two loops

for bi = 1,N,B
for bj = 1,N,B
   for bk = 1,N,B
      for i = bi, min(bi+B-1,N)
         for j = bj, min(bj+B-1,N)
            for k = bk, min(bk+B-1,N)
               y(i) = y(i) + A(i,j)*x(j)
Notes

• So far, we have considered a two-level memory hierarchy
• Real machines have multiple level memory hierarchies
• In principle, we need to block for all levels of the memory hierarchy
• In practice, matrix multiplication with really large matrices is very rare
  – MMM shows up mainly in blocked matrix factorizations
  – therefore, it is enough to block for registers, and L1/L2 cache levels
• How do we organize such a code?
  – We will study the code produced by ATLAS.
  – ATLAS also introduces us to self-optimizing programs.