Cache Models and Program Transformations

Andrew Lenharth
Adapted from slides by Keshav Pingali

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 Storage

- A matrix is a 2-D array of elements, but memory is a 1-D array
- Mapping from (r,c) -> address
  - Column-major: addr = base + r + c*n
  - Row-major: addr = base + r*n + c

Matrix-vector product

- Code:
  ```c
  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
  - Compiler will register-allocate y[i]
  - If provably doesn’t alias with A or x
  - You can get this effect manually
    ```c
    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 \( r_2 \)
- \( \text{stackDistance}(r_1, r_2) \): number of distinct cache lines referenced between \( r_1 \) and \( r_2 \)
- Stack distance was defined by Mattson et al. (IBM Systems Journal paper)

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

- 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.)

- Small cache model:
  - A: \( N^2 \) misses
  - x: \( N + (N(N-1)) \) misses (reuse distance=O(1))
  - 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) 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 (|x|) numbers
    • Transition: (|x|) > C
  – LRU replacement:
    • by the time we get to end of a row of A, first few elements of x
      will have been used and will have to be replaced
    • Transition: (|x|) = C

• Note:
  – optimal replacement requires perfect knowledge about future
  – most real caches use LRU or something close to it
  – some architectures support "streaming"
  – in software: hints to tell processor not to cache certain references

Scenario II

• Code:
  for j = 1,N
    for i = 1,N
      y(i) = y(i) + A(i,j)*x(j)
• Inner loop is known as AXPY in NA literature
  y = α · 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)
• Inner loop is known as AXPY in NA literature
  y = α · x + y
• Miss ratio picture exactly the same as Scenario I
  roles of x and y are interchanged

Scenario III (contd.)

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

• Small cache model:
  – A: N²/b misses
  – x: N/b misses (reuse distance=O(N))
  – y: N/b misses (reuse distance=O(1))
  – Total = (N²+N)/b
  – Miss ratio = (N²+N)/4bN
    ~ 0.5/b + 0.25/bN
• Large cache model:
  – Misses:
    • A: N²/b misses
    • x: N/b misses
    • y: N/b misses
    • Total = (N²+2N)/b
    • Miss ratio = (N²+2N)/4bN
    ~ 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 ~ C
  – Optimal: roughly when (N+2b) = C

• Jump from large cache model to small cache model will be more gradual in reality because of conflict misses

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:
  \[
  \begin{align*}
  \text{for } j = 1, N \\
  \text{for } i = 1, N \\
  y(i) &= y(i) + A(i,j) * x(j)
  \end{align*}
  \]

- Large cache model:
  - Same as Scenario III

- Small cache model:
  - Misses:
    - \( A: N^2 \)
    - \( x: N/b \)
    - \( y: N/b + N(N^2 - 1)/b = N^2/b \)
    - Total: \( N^2/(1+b) + N/b \)
  - Miss ratio: \( 0.25(1+1/b) + 0.25/bN \)
  - Transition from large cache to small cache model
    - LRU: \( Nb + N + b = C \) \( \Rightarrow N \sim C/(b+1) \)
    - Optimal: \( N + 2b = C \) \( \Rightarrow N \sim C \)
  - Transition happens much sooner than in Scenario III (with LRU replacement)

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
    - \( 2B^2 \approx N^2 \) \( \Rightarrow B \approx \sqrt{N} \)
    - Actually we can do better than this
    - Code: blocked code
      \[
      \begin{align*}
      \text{for } bi = 1, N/B \\
      \text{for } bj = 1, N/B \\
      \text{for } i = bi, \min(bi+B-1, N) \\
      \text{for } j = bj, \min(bj+B-1, N) \\
      y(i) &= y(i) + A(i,j) * x(j)
      \end{align*}
      \]
    - 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 \( B \sim C/2 \) for row-major order of storage and LRU replacement.
    - Since entire MVM computation is a sequence of block MVMs, this map's miss ratio will be \( 0.25/b \) independent of \( N \)

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.
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

K loop innermost
- A: good spatial locality
- C: good temporal locality

I loop innermost
- B: good temporal locality

J loop innermost
- B,C: good spatial locality
- A: good temporal locality

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

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 = $3 N^2 b^4 N^3 = 0.75bN = 0.019$ ($b = 4, N=10$)

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: $N^2/b$ misses (good spatial locality)
  - B: $N^3$ misses (poor temporal and spatial locality)
  - Miss ratio $\rightarrow 0.25 (b+1)/b = 0.3125$ (for $b = 4$)

- 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?)
Miss ratios for other versions

DO I = 1, N
DO J = 1, N
DO K = 1, N
C(I,J) = C(I,J) + A(I,K)*B(K,J)

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
- A: good temporal locality
  \(N^3/b + N^3/b + N^2/b)/4N^3 \rightarrow 0.5/b\)

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

Transpose B

Construct B Transpose

DO I = 1, N
DO J = 1, N
DO K = 1, N
C(I,J) = C(I,J) + A(I,K)*B^T(J,K)

Costs?
Tradeoffs?

MMM experiments

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

Fused Multiply-Add

d = a + b * x is sufficiently common that it appears as an instruction in most ISAs
Marketing likes to use this instruction to compute "peak" FLOPS: \(2 * \text{number of FMA per cycle} \times \text{vector width} \times \text{number of threads}\)

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.