Cache Models and Program Transformations

Keshav Pingali
University of Texas at Austin

Memory wall problem

- Optimization focus so far:
  - reducing the amount of computation
  - (eg) constant folding, common sub-expression elimination, …
- On modern machines, most programs that access a lot of data are memory bound
  - latency of DRAM access is roughly 100-1000 cycles
- Caches can reduce effective latency of memory accesses
  - but programs may need to be rewritten to take full advantage of caches

Do cache optimizations matter?

Vendor BLAS (multiple levels of blocking)

3 nested loops MMM (no blocking)

MM for square matrices of various sizes
UltraSPARC III: peak 2 GFlops

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
- Extend this to MMM
  - aka Level-3 Basic Linear Algebra Subroutines (BLAS)
  - most important kernel in dense linear algebra

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”
  - (e.g. 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 \( r_1 \) will be a capacity miss if 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:

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

- 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 = \( N^2 + 2N \) misses
  - Miss ratio = \( (N^2 + 2N)/4N^2 \approx 0.25 + 0.5/N \)
- 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 = \( (2N^2+N)/4N^2 \approx 0.5 + 0.25/N \)

Scenario I (contd.)

- 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:
\[ \forall i, j \in [1..N]: (y(i), A(i,j), x(j), y(i), A(i,j), x(j), \ldots, y(i), A(i,j), x(j)) \]

- Question: as problem size increases, when do capacity misses begin to occur?
- Depends on replacement policy:
  - Optimal replacement:
    - 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:
    - Not all \( i \) elements of \( A \) and first few elements of \( x \) can 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

Miss ratio graph

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

Scenario II

- Code:
  for \( j = 1..N \)
  for \( i = 1..N \)
  \[ y(i) = y(i) + A(i,j) \times x(j) \]
- Inner loop is known as AXPY in NA literature
- Miss ratio picture exactly the same as Scenario I
  - roles of \( x \) and \( y \) are interchanged

Scenario III

- Code:
  for \( i = 1..N \)
  for \( j = 1..N \)
  \[ y(i) = y(i) + A(i,j) \times x(j) \]
- 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 = \( (N^2+2N)/4bN^2 \)
  - \( \sim 0.25/b + 0.5/bN \)
- 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 \( (N^2+2N)/b \Rightarrow N \sim C \)
    - Optimal: roughly when \( (N^2+2N)/b \Rightarrow N \sim C \)
- 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 \( b \approx 4 \) (SGI Octane)

Scenario III (contd.)

Address stream:
\[ \forall i, j \in [1..N]: (y(i), A(i,j), x(j), y(i), A(i,j), x(j), \ldots, y(i), A(i,j), x(j)) \]

- 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 \( (N^2+2N)/b \Rightarrow N \sim C \)
    - Optimal: roughly when \( (N^2+2N)/b \Rightarrow N \sim C \)
    - 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 \( b \approx 4 \) (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:**
  
  \[
  \text{for } j = 1, N \\
  \text{for } i = 1, N \\
  \quad y(i) = y(i) + A(i,j) \times x(j)
  \]

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

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

- **Transition from large cache to small cache model:**
  - LRU: \(N b + N + b = C \Rightarrow N \sim \frac{C}{b+1}\)
  - 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
    - \(A \sim 2B \Rightarrow A \sim C\)
    - Actually we can do better than this
  - **Code:** blocked code
    
    \[
    \text{for } bj = 1, N, B \\
    \text{for } bi = 1, N, B \\
    \text{for } j = bi, \min(bi+B-1, N) \\
    \text{for } i = bj, \min(bj+B-1, N) \\
    \quad y(i) = y(i) + A(i,j) \times x(j)
    \]

- **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 \(B=\frac{C}{2}\) for row-major order of storage and LRU replacement

- **Since entire MVM computation is a sequence of blocked MVMs, the miss ratio will be:** \(0.25/b\) independent of \(N\)

**Miss ratios**

- **DAXPY**
  - Miss ratio: \(0.25(1 + \frac{1}{b})\)

- **DDOT**
  - Miss ratio: \(0.50/b\)

**Key transformations**

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

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

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

**Scenario V (contd.)**

- **Better code:** interchange the two outermost loops and fuse \(bi\) and \(i\) loops
  
  \[
  \text{for } i = 1, N \\
  \text{for } i = bi, \min(bi+B-1, N) \\
  \]

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

MMM

\[
\text{DO } I = 1, N/\text{row-major storage} \\
\text{DO } J = 1, N \\
\text{DO } K = 1, N \\
C(I,J) = C(I,J) + A(I,K) \cdot B(K,J)
\]

- 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

MMM miss ratios (simulated)

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

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

- 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 = \( 3 N^2 / b \times 4N^3 = 0.75/bN \) (eg \( b = 4, N=10 \))

Miss ratios for other versions

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

- Small cache scenario:
  - Cold and capacity misses
  - Miss ratio:
    - A: \( N^2/b \) misses (good spatial locality)
    - B: \( N^3 \) misses (poor temporal and spatial locality)
    - Miss ratio \( A \leq 0.25(1/b) \) (good temporal locality)
    - Miss ratio \( B \leq 0.5 \) (good spatial locality)
    - Miss ratio \( \leq 0.5/b \) (good spatial locality)
    - Miss ratio \( \leq 0.5 \) (good spatial locality)

Transition out of large cache

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

- 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 \( N \approx \sqrt{C} \)
- For our cache, we get \( N \approx 45 \) which agrees quite well with data
As in blocked MVM, we actually need to stripmine only two loops:

\[
\begin{align*}
    &\text{for } b_i = 1, 2, 3, \\
    &\text{for } b_j = 1, 2, 3, \\
    &\text{for } k = 1, 2, \ldots, n, \\
    &\text{for } i = 1, 2, \ldots, m, \\
    &\text{for } j = 1, 2, \ldots, p, \\
    &y(i, j) = y(i, j) + a(i, k)w(k, j)
\end{align*}
\]

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.
- We have also ignored hardware prefetching.