Eshita Arza (COE15B013)
Akshay Kumar (CED15I031)
Aditya Prakash (CED15I025)
12 February 2018 - CA LAB 6

Code Optimisation for Matrix Multiplication

Problem Statement : Take two square matrices- A, B, and analyse cache misses for each of the following inner loop iterations:

1. i-j-k
2. j-i-k
3. k-i-j
4. i-k-j
5. j-k-i
6. k-j-i

Below are few terms that will be used majorly in our analysis:

1. Spatial Locality: It is the use of data elements within close storage locations.
2. Temporal Locality: Reuse of specific data/resources within a relatively small time duration.
3. Access Stride: It's the number of locations in the memory between beginnings of successive array elements.
4. Row Major Access: Accessing columns in a row.
5. Column Major Access: Accessing rows in a column.
6. Compulsory Miss: The first reference to a block of memory.

Assumptions for the analysis:

- Line Size = 32 Bytes
- Cache can only hold one row of the matrix at a time.
- Using Stride-1 (elements of array are stored consecutively without gaps)
- “i” is row index for matrices A and C.
- “j” is column index for matrices B and C.
- “k” is the row index for B, and column index for A.
- C array is allocated in row-major order.

Analysis :

When we step through columns in a row, we are accessing successive elements in memory (spatial locality helps). So, the compulsory miss is only for the first element accessed from a cache block.

When we step through rows in a column, we are accessing distant elements in memory (No spatial locality). So, the compulsory miss happens for every element being accessed (100% miss rate).

Case 1 : i-j-k :

```
for (i=0; i<n; i++)  {
  for (j=0; j<n; j++) {
    sum =0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
} }
```

- Every iteration will have 2 load operations, but 0 store operations.
- The misses per iteration is :
  - For matrix A = 0.25 (Due to row major access)
  - For matrix B = 1.00 (Due to column major access)
  - For matrix C = 0.00 (Due to no store operations)
  - **Total = 1.25**

Case 2 : j-i-k :

```
for (j=0; j<n; j++) {
  for (i=0; i<n; i++)  {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
} }
```

- Every iteration will have 2 load operations, but 0 store operations.
- The misses per iteration is :
    - For matrix A = 0.25 (Due to row major access)
    - For matrix B = 1.00 (Due to column major access)
    - For matrix C = 0.00 (Due to no store operations)
    - **Total = 1.25**

Case 3 : k-i-j :

```
for (k=0; k<n; k++) {
  for (i=0; i<n; i++) {
    r = a[i][k];
    for (j=0; j<n; j++)
      c[i][j] += r * b[k][j];
  }
}
```

- Every iteration will have 2 load operations, and 1 store operation.
- The misses per iteration is :
    - For matrix A = 0.00 (Due to fixed element)
    - For matrix B = 0.25 (Due to row major access)
    - For matrix C = 0.25 (Due to row major access)
    - **Total = 0.50**

Case 4 : i-k-j :

```
for (i=0; i<n; i++) {
  for (k=0; k<n; k++) {
    r = a[i][k];
    for (j=0; j<n; j++)
      c[i][j] += r * b[k][j];
  }
}
```

- Every iteration will have 2 load operations, and 1 store operation.
- The misses per iteration is :
    - For matrix A = 0.00 (Due to fixed element)
    - For matrix B = 0.25 (Due to row major access)
    - For matrix C = 0.25 (Due to row major access)
    - **Total = 0.50**

Case 5 : j-k-i :

```
for (j=0; j<n; j++) {
  for (k=0; k<n; k++) {
    r = b[k][j];
    for (i=0; i<n; i++)
      c[i][j] += a[i][k] * r;
  }
}
```

- Every iteration will have 2 load operations, and 1 store operation.
- The misses per iteration is :
    - For matrix A = 1.00 (Due to column major access)
    - For matrix B = 0.00 (Due to fixed element)
    - For matrix C = 1.00 (Due to column major access)
    - **Total = 2.00**

Case 6 : k-j-i :

```
for (k=0; k<n; k++) {
  for (j=0; j<n; j++) {
    r = b[k][j];
    for (i=0; i<n; i++)
      c[i][j] += a[i][k] * r;
  }
}
```

- Every iteration will have 2 load operations, and 1 store operation.
- The misses per iteration is :
    - For matrix A = 1.00 (Due to column major access)
    - For matrix B = 0.00 (Due to fixed element)
    - For matrix C = 1.00 (Due to column major access)
    - **Total = 2.00**

*** **End of Analysis** ***

Inferences:

- For the iterations “i-j-k” and “j-i-k”, the TOTAL misses per iteration is **1.25**.
- For the iterations “k-i-j” and “i-k-j”, the TOTAL misses per iteration is **0.50**.
- For the iterations “j-k-i” and “k-j-i”, the TOTAL misses per iteration is **2.00**.

Therefore, the combination of “k-i-j” and “i-k-j”, which exploits spatial locality to the maximum extent gives better results in terms of a lower cache miss rate.

***Note**: In case the matrices are stored in column major order, the combination “j-k-i” and “k-j-i” will give lower cache miss rates.*