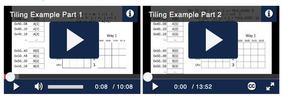
PRE21.1. 21 Text and Videos

Videos

Tiling Examples



Video credits: Geoffrey Herman, Text credits: Geoffrey Herman

The Big Picture

Each of the coding techniques we introduce aim to improve the spatial locality or the temporal locality of the code we write so that the code performs better on the cache.

- Loop Fusion: In loop fusion, we are increasing the temporal locality of our code by making repeated accesses of the same element of an array happen consecutively rather than after full traversals of an array
- Loop Fission: In loop fission, we are also increasing the spatial locality of our code by separating loop traversals so that we can take advantage of the spatial locality of accesses of multiple different arrays when the accesses to one array collide with the accesses of the other array.
- Loop inversion: In loop inversion, we are trying to improve the spatial locality of our code by using row traversals or other high spatial locality code structures rather than column traversals or other low spatial locality code structures.
- Tiling: In tiling, we are trying to improve the temporal locality of our code by breaking up large data structures into a series of smaller data structures so that we can perform repeated traversals over the smaller data structure before they get replaced in the cache.

Loop Fusion

Original has 2 x LARGE/(Number of elements of A in a cache block) misses

```
for (int j = 0; j < LARGE; j++) {
  sum += A[j];
}
for (int j = 0; j < LARGE; j++) {
  product *= A[j];
}
```

After loop fusion has LARGE/(Number of elements of A in a cache block) misses

```
for (int j = 0; j < LARGE; j++) {
  sum += A[j];
  product *= A[j];
}
```

Loop Fission

Original has LARGE misses for accesses to A

```
for (int j = 0; j < LARGE; j++) {
  sum += A[j];
  for (int k = 0; k < LARGE; k++) {
    other_sum += B[j][k];
  }
}
```

After loop fission has LARGE/(Number of elements of A in a cache block) misses

```
for (int j = 0; j < LARGE; j++) {
  sum += A[j];
}
for (j = 0; j < LARGE; j++) {
  for (int k = 0; k < LARGE; k++) {
    other_sum += B[j][k]
  }
}
```

Loop Inversion

Original has m x n/(Number of elements of A in a cache block) misses

```
for (i = 0; i < m; i ++)  {
  for (j = 0; j < n; j ++) {
    sum += a[j];
  }
}
```

After loop inversion has n/(Number of elements of A in a cache block) misses

```
for (j = 0; j < n; j ++)  {
  for (i = 0; i < m; i ++) {
    sum += a[j];
  }
}
```

Considerations for when to use Loop Inversion.

- You have nested loops.
- Incrementing the inner loop has a large stride length (in Bytes), but incrementing the outer loop would have a small stride length if it was the inner loop.
- Common cases include
  - Column traversals for 2D arrays
  - `struct` arrays where the inner loop traverses the array rather than individual structs

Tiling

Considerations for when to use tiling.

- You are re-using the same piece of data multiple times. If your code does not have data re-use, tiling will NOT help.
- The code's data re-use does NOT have temporal locality (i.e., you are not re-using the data before it gets kicked out of the cache).
- Other techniques such as loop fission, loop fusion, and loop inversion are insufficient to create temporal locality in your data re-use.
- Be on the look-out for column and row traversals in the same innermost loop (or the equivalent of column and row traversals for structs).

The number of misses for tiling depends on the tile size. Considerations for choosing your tile size.

- All tiles should fit inside the cache until you are done using those tiles
- Make the tile as large as possible given the previous consideration
- Remember to consider the size of your data type when doing your calculation

1D Tiling

Original

```
for (int i = 0; i < M; i ++)  {
  for (int j = 0; j < N; j ++) {
    sum += a[j] / b[i];
  }
}
```

After Tiling

```
for (int j = 0; j < N; j += TILE_SIZE) {
  for (int i = 0; i < M; i ++)  {
    for (int jj = j; jj < min(N,j + TILE_SIZE); jj ++) {
      sum += a[jj] / b[i];
    }
  }
}
```

Multi-dimensional Tiling

Original

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      c[i*N+j] += a[i*N + k]*b[k*N + j];
```

After Tiling

```
  for (int i = 0; i < N; i += TILE_SIZE)
    for (int j = 0; j < N; j += TILE_SIZE)
      for (int k = 0; k < N; k += TILE_SIZE)
      /* B x B mini matrix multiplications */
        for (int ii = i; ii < min(N,i+TILE_SIZE); ii++)
          for (int jj = j; jj < min(N,j+TILE_SIZE); jj++)
            for (int kk = k; kk < (min(N,k+TILE_SIZE); kk++)
              c[ii*N+jj] += a[ii*N + kk]*b[kk*N + jj];
```

Prefetching

- Use prefetching when other techniques cannot further improve your miss rate and your access pattern is predictable and your number of future predictable accesses is large.
- You may assume that all misses will take 100-200 clock cycles in this class (yes real miss rates are somewhat variable).
- Prefetch far enough ahead in your access pattern such that a cache miss by the pre-fetcher will be resolved before your code catches up. You will need to think about how your code will be implemented at the assembly to pre-fetch effectively.
- Do not pre-fetch so far ahead that you are pre-fetching more than the entire cache size ahead.

Pre-fetching syntax is

```
void builtin_prefetch(const void *addr)
```

Make sure that you pre-fetch the address of the elements you want to pre-fetch and not just the elements. For example,

```
builtin_prefetch(&A[i+8]);
```

Mark as read

- [ ] (a) I've read this!

Select all possible options that apply.

Save & Grade *Unlimited attempts* Save only

PRE 21

Assessment overview

Total points: 0/75

Score: 0%

Question PRE21.1

Value: 1

Total points: — /1

Auto-graded question

Previous question Next question

Personal Notes

No attached notes

Attach a file

Add text note