Understanding Cache behaviour using a simulator

CSL211

Shivam Mittal
2015csb1032
2015csb1032@iitrpr.ac.in

Part (a)

Objective

To understand cache hit/miss statistics of data cache in various configurations.

Introduction

Cache

Cache memory, also called CPU memory, is random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM. This memory is typically integrated directly with the CPU chip or placed on a separate chipthat has a separate bus interconnect with the CPU. [1]

Cache Miss

Cache miss is a state where the data requested for processing by a component or application is not found in the cache memory. It causes execution delays by requiring the program or application to fetch the data from other cache levels or the main memory. [2]

Cache Hit

A cache hit is a state in which data requested for processing by a component or application is found in the cache memory. It is a faster means of delivering data to the processor, as the cache already contains the requested data. [2]

Procedure

1. Use Dinero Cache Simulator IV [3] to get the cache statistics.
2. Write the code of matrix multiplication with variable matrix size, and instrumentation to get the memory trace.
3. Save the memory access trace for two matrix sizes - 200 and 500.
4. Run the Dinero Cache Simulator with the two different trace generated above.
5. Run the simulator for different configurations :
   a. Cache size : 16kb, 32kb
   b. Line/Block size : 32 byte, 64 byte
   c. Associativity : 1, 2, 4
   d. Replacement Policy - LRU
6. Compare the different statistics generated.

Observations

1. We compared the cache hit/miss statistics for different parameters. To compare one parameter, other parameters were kept constant and there values compared.
   a. For eg : To observe the impact of the matrix size, we compared the demand miss rate for matrix size 200*200 and 500*500. We varied the different parameters such as cache size, block size and associativity and for each constant configuration, we compared the values for the different matrix sizes.
2. We observed the following impacts :
   a. Matrix size impact
   b. Cache size impact
   c. Block size impact
   d. Associativity impact

Matrix Size Impact - Observation

Matrix Size	Cache-16kB Block-32b Assoc.-1	Cache-16kB Block-32b Assoc.-2	Cache-16kB Block-32b Assoc.-4	Cache-16kB Block-64b Assoc.-1	Cache-16kB Block-64b Assoc.-2	Cache-16kB Block-64b Assoc.-4
200*200	0.1099	0.0657	0.0630	0.4318	0.1097	0.0748
500*500	0.4032	0.2333	0.2958	0.5019	0.5313	0.5325

Matrix Size	Cache-32kB Block-32b Assoc.-1	Cache-32kB Block-32b Assoc.-2	Cache-32kB Block-32b Assoc.-4	Cache-32kB Block-64b Assoc.-1	Cache-32kB Block-64b Assoc.-2	Cache-32kB Block-64b Assoc.-4
200*200	0.0865	0.0634	0.0630	0.4209	0.0778	0.0316
500*500	0.3721	0.1234	0.0627	0.4438	0.2124	0.1770

Matrix Size Impact - Explanation

It is observed that when the matrix size is increased to 500*500 from 200*200, the demand miss rate increases in every case. This is obvious because when the size of the matrix increases, the data we need to read/write increases. Hence there would be more memory requests, resulting in higher chances of cache miss.

Cache Size Impact - Observation

Cache Size	Matrix - 200 Block-32b Assoc.-1	Matrix - 200 Block-32b Assoc.-2	Matrix - 200 Block-32b Assoc.-4	Matrix - 200 Block-64b Assoc.-1	Matrix - 200 Block-64b Assoc.-2	Matrix - 200 Block-64b Assoc.-4
16kB	0.1099	0.0657	0.0630	0.4318	0.1097	0.0748
32kB	0.0865	0.0634	0.0630	0.4209	0.0778	0.0316

Cache Size	Matrix - 500 Block-32b Assoc.-1	Matrix - 500 Block-32b Assoc.-2	Matrix - 500 Block-32b Assoc.-4	Matrix - 500 Block-64b Assoc.-1	Matrix - 500 Block-64b Assoc.-2	Matrix - 500 Block-64b Assoc.-4
16kB	0.4032	0.2333	0.2958	0.5019	0.5313	0.5325
32kB	0.3721	0.1234	0.0627	0.4438	0.2124	0.1770

Cache Size Impact - Explanation

As cache size increases, the demand miss rate decreases in every case. This is because when we increase the cache size, it can store more data, so the number of capacity misses will decrease as the capacity of the cache has increased. In general miss rate is inversely proportional to the square root of the cache size.

However, doubling the size of a cache requires twice the area, slows it down, and increases the power consumption. For larger cache size, hit time and miss penalty increases, so basically there is a tradeoff.

Block Size Impact - Observation

Block Size	Matrix - 200 Cache-16kB Assoc.-1	Matrix - 200 Cache-16kB Assoc.-2	Matrix - 200 Cache-16kB Assoc.-4	Matrix - 200 Cache-32kB Assoc.-1	Matrix - 200 Cache-32kB Assoc.-2	Matrix - 200 Cache-32kB Assoc.-4
32 bytes	0.1099	0.0657	0.0630	0.0865	0.0634	0.0630
64 bytes	0.4318	0.1079	0.0748	0.4209	0.0778	0.0316

Block Size	Matrix - 500 Cache-16kB Assoc.-1	Matrix - 500 Cache-16kB Assoc.-2	Matrix - 500 Cache-16kB Assoc.-4	Matrix - 500 Cache-32kB Assoc.-1	Matrix - 500 Cache-32kB Assoc.-2	Matrix - 500 Cache-32kB Assoc.-4
32 bytes	0.4032	0.2333	0.2958	0.3721	0.1234	0.0627
64 bytes	0.5019	0.5313	0.5325	0.4438	0.2124	0.1770

Block Size Impact - Explanation

As seen above, increasing the block size, generally results in increasing the miss rate. This can be explained from the fact that, when the block size increases:

1. If we need to evict one block, then a larger chunk of data is evicted. It is obvious that if larger data is evicted, there is a higher chance of getting a miss because for more number of addresses there would be a miss.
2. The number of blocks that can be saved on the cache decreases, so if data which are spread at large distances needs to be accessed (non continuous), the number of misses will increase.

But, when block size increases, the number of compulsory misses decrease, hence in one case, the miss rate had decreased. Because this factor outweighed the negative factors given above.

Associativity Impact - Observation

Associativity	Matrix - 200 Cache - 16kB Block - 32b	Matrix - 200 Cache - 16kB Block - 64b	Matrix - 200 Cache - 32kB Block - 32b	Matrix - 200 Cache - 32kB Block - 64b	Matrix - 500 Cache - 16kB Block - 32b	Matrix - 500 Cache - 16kB Block - 64b	Matrix - 500 Cache - 32kB Block - 32b	Matrix - 500 Cache - 32kB Block - 64b
1	0.1099	0.4318	0.0865	0.4209	0.4032	0.5019	0.3721	0.4438
2	0.0657	0.1097	0.0634	0.0778	0.2333	0.5313	0.1234	0.2124
3	0.0630	0.0748	0.0630	0.0316	0.2958	0.5325	0.0627	0.1770

Associativity Impact - Explanation

We can see that on increasing the associativity, cache miss ratio decreases in general. This is because when we increase the associativity, the number of conflict misses decreases because each set has more blocks and hence there will be lesser chances of conflict between two addresses.

But, increasing the associativity of a cache increases the latency and power consumption. Hence the designer needs to carefully examine the tradeoff.

Conclusion

The general trend which is observed by studying all the different configurations is:

1. The cache miss ratio increases when the matrix size increases.
2. The cache miss ratio decreases when the cache size increases.
3. The cache miss ratio increases when the block size increases.
4. The cache miss ratio decreases when the associativity increases.

Part (b)

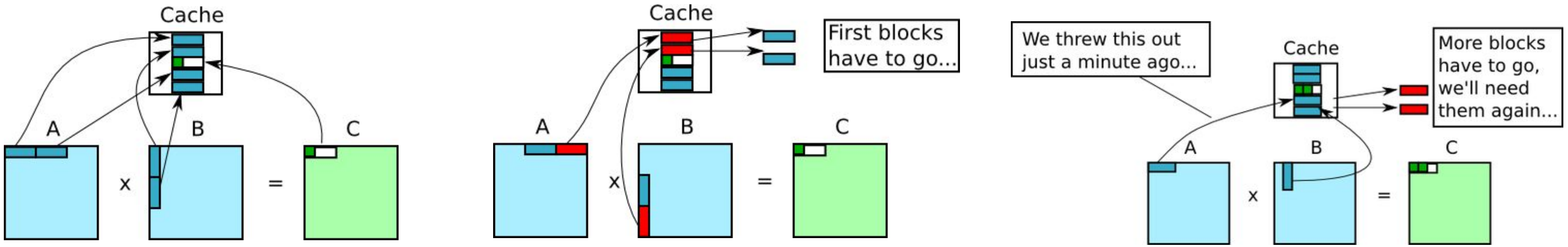
Objective

To experiment with cache specific code optimization.

Introduction

We can optimize our code to increase the utilization of cache by using tiling.

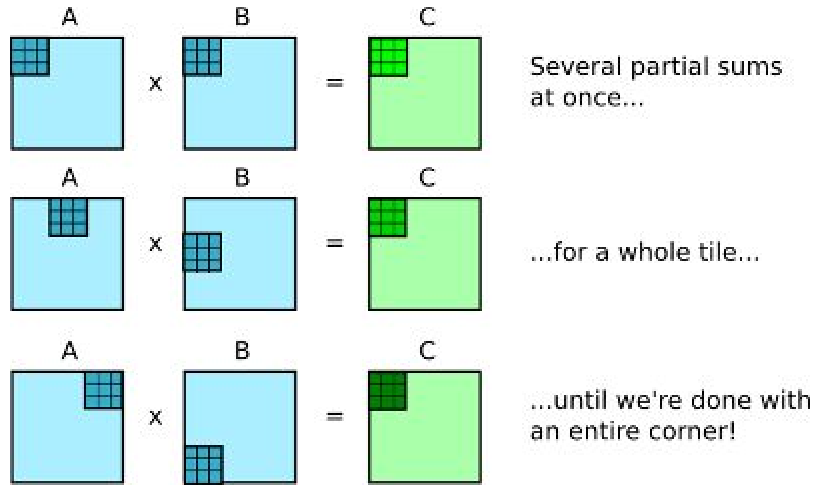
In traditional matrix multiplication method, when the innermost loop of your matrix multiplier reads entire rows/columns in sequence, the cache gradually fills up with data. Cache size is limited, so if your rows are really long, the cache must throw away what it loaded initially, to make room for new stuff. When you reach the end of a row/column pair and start the next one, you'll need some data that were recently in the cache. Since they've been thrown out, there would be a cache miss.



Cache
A
B
C
x
=
Cache
First blocks have to go...
A
B
C
x
=
We threw this out just a minute ago...
Cache
More blocks have to go, we'll need them again...
A
B
C
x
=

Introduction

This is a tragic waste of effort, because matrix multiplication is just a massive addition of a pile of products - it doesn't matter much which order you add them up in. If we work out partial sums for a *tile* of results at a time, it'll be ok as long as all the right products contribute to the right sums in the end. It will still cause some re-loading, but there's less, because if tiles fit into caches where whole matrices can not, you'll get more of the overall work done per cacheful: values in a tile go into more than one result value before they are evicted.


A
B
C
x
=
Several partial sums at once...
A
B
C
x
=
...for a whole tile...
A
B
C
x
=
...until we're done with an entire corner!

Introduction

Trivial matrix multiplication

```
        for(i =0;i<SIZE;i++){
         for(j=0;j<SIZE;j++){
                  int sum=0;
        for(k=0; k < SIZE; k++) {
                sum += M[i][k] * N[k][j];
                fprintf(a,"0 %p\n", &M[i][k]);
                fprintf(a, "0 %p\n", &N[k][j]);
        }
        C[i][j] = sum;
        fprintf(a,"1 %p\n", &C[i][j]);
         }
    }
```

Optimized matrix multiplication

```
        for (i=0; i<SIZE; i+=TILE ){
        for (j=0; j<SIZE; j+=TILE ){
        for (k=0; k<SIZE; k+=TILE ){
                for (y=i; y<i+TILE; y++ ){
                for (x=j; x<j+TILE; x++ ){
                        for (z=k; z<k+TILE; z++ ){
                        sum = sum+M[y][z]*N[z][x];
                        fprintf(a,"0 %p\n", &M[y][z]);
                         fprintf(a, "0 %p\n", &N[z][x]);
                        }
                        C[y][x] = sum;
                        fprintf(a,"1 %p\n", &C[y][x]);
              }}}}}
```

Procedure

1. Use Dinero Cache Simulator IV [3] to get the cache statistics.
2. Write the **optimized code** of matrix multiplication with variable matrix size, and instrumentation to get the memory trace.
3. Save the memory access trace for two matrix sizes - 200 and 500.
4. Run the Dinero Cache Simulator with the two different trace generated above.
5. Run the simulator for different configurations :
   a. Cache size : 16kb, 32kb
   b. Line/Block size : 32 byte, 64 byte
   c. Associativity : 1, 2, 4
   d. Replacement Policy - LRU
6. Compare the different statistics generated for optimized and non-optimized version of code.

Observations

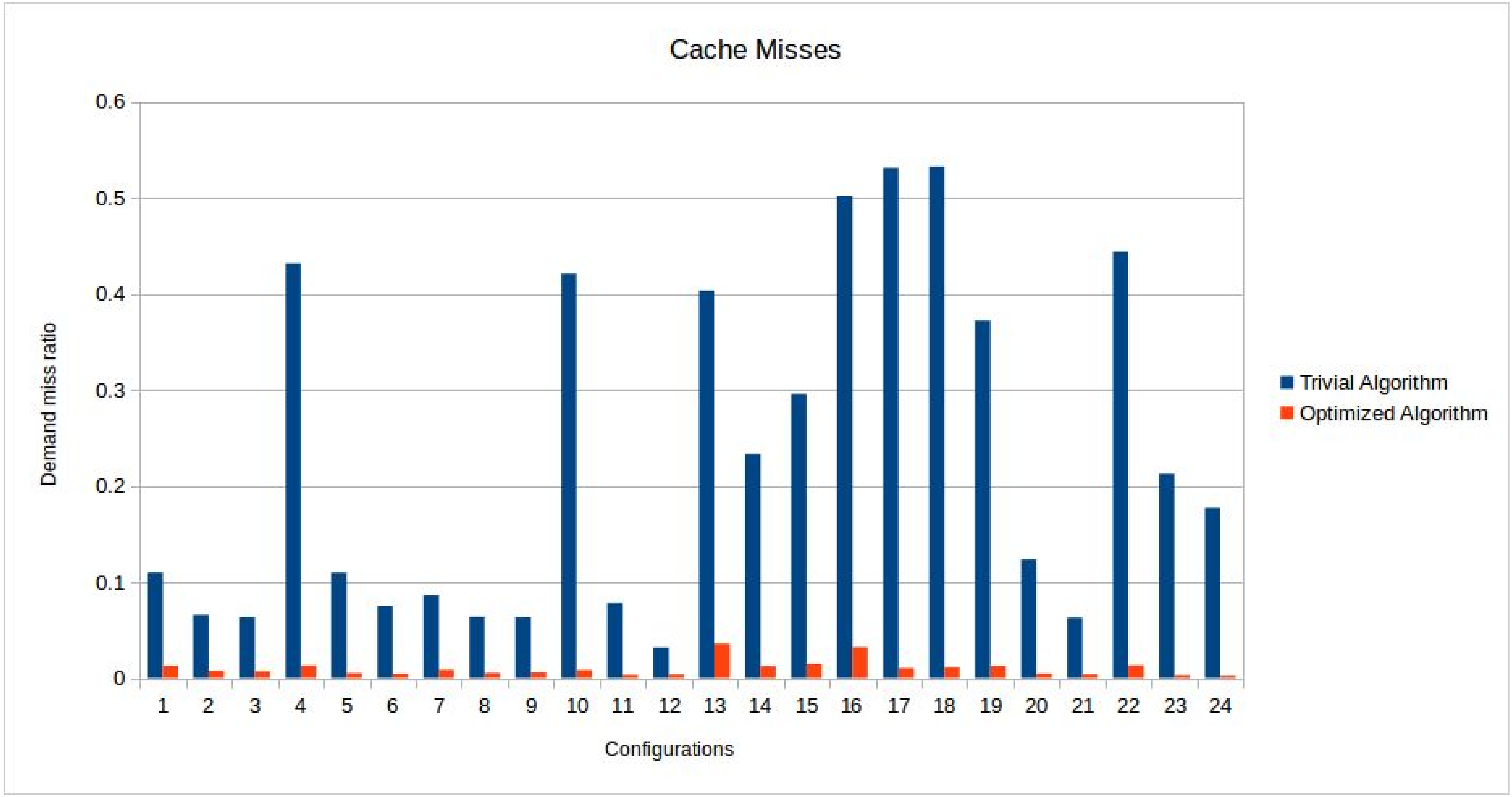
Trivial Algorithm	0.1099	0.0657	0.0630	0.4318	0.1097	0.0748
Optimized Algorithm	0.0128	0.0075	0.0070	0.0130	0.0051	0.0044

Trivial Algorithm	0.0865	0.0634	0.0630	0.4209	0.0778	0.0316
Optimized Algorithm	0.0087	0.0053	0.0058	0.0083	0.0034	0.0038

Observations

Trivial Algorithm	0.4032	0.2333	0.2958	0.5019	0.5313	0.5325
Optimized Algorithm	0.0358	0.0124	0.0145	0.0320	0.0104	0.0113

Trivial Algorithm	0.3721	0.1234	0.0627	0.4438	0.2124	0.1770
Optimized Algorithm	0.0126	0.0046	0.0040	0.0131	0.0030	0.0023


Cache Misses
Demand miss ratio
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
Configurations
Trivial Algorithm
Optimized Algorithm

Result

$$\frac{CacheMissRatio(Optimized)}{CacheMissRatio(Trivial)} = \frac{\sum_{all} Cachemissratio(Optimized)}{\sum_{all} Cachemissratio(Trivial)}$$

$$\frac{CacheMissRatio(Optimized)}{CacheMissRatio(Trivial)} = \frac{0.2411}{5.4875} = 0.04393$$

Ratio of cache miss of the optimized algorithm to the trivial algorithm is equal to 4.393 %

The cache misses improves by 95.607% by using the optimised algorithm.

References

[1] - http://searchstorage.techtarget.com/definition/cache-memory

[2] - https://www.techopedia.com/

[3] - http://pages.cs.wisc.edu/~markhill/DineroIV/)

[4] - https://www.quora.com/What-is-the-best-way-to-multiply-two-matrices-in-C++

[5] - Computer Organisation and Architecture - Smruti Ranjan Sarangi