HW 2

Rijish Ganguly

Performance and Parallelism

ECE 565

1. As we have 64B blocks, the offset requires 6 bits. Again, with 64B cache blocks with 512B total cache size, that means 8 blocks. As we have 8 blocks with 2-way set association, there will be 4 sets, thus requiring 2 set bits for the index. We have a 20 bit address minus and hence we would require 12 bits for the tag.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hex Address | Tag | Set | Offset | Hit/Miss |
| 0xABCDE. | 101010111100 | 11 | 011110 | M |
| 0x14327. | 000101000011 | 00 | 100111 | M |
| 0xDF148. | 110111110001 | 01 | 001000 | M |
| 0x8F220. | 100011110010 | 00 | 100000 | M |
| 0xCDE4A. | 110011011110 | 01 | 001010 | M |
| 0x1432F. | 000101000011 | 00 | 101111 | H |
| 0x52C22. | 010100101100 | 00 | 100010 | M |
| 0xABCF2 | 101010111100 | 11 | 110010 | H |
| 0x92DA3 | 100100101101 | 10 | 100011 | M |
| 0xF125C | 111100010010 | 01 | 011100 | M |

Way 1 Way 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | LRU | V | Tag | V | Tag |
| Set 0 | 0 | 1 | 010100101100 (0x52C22) | 1 | 000101000011 (0x14327) |
| Set 1 | 1 | 1 | 110011011110 (0xCDE4A) | 1 | 111100010010 (0xF125C) |
| Set 2 | 1 | 0 |  | 1 | 100100101101 (0x92DA3) |
| Set 3 | 1 | 0 |  | 1 | 101010111100 (0XABCF2) |

2. AMAT = Hit TimeL1 + Miss-RateL1 X Miss PenaltyL1

Miss-PenaltyL1 = Hit-TimeL2 + Miss-RateL2 X Miss-PenaltyL2

L2 cache hit time of 20 CPU cycle and a miss penalty of 250 CPU cycles

L1 hit time is 1 CPU cycle

1. L1 miss rate = 0.02 and L2 miss rate = 0.40

AAT = 1 + 0.02\*(20+250\*0.40) = 3.4 CPU cycles

1. L1 miss rate = 0.10 and L2 miss rate = 0.05

AAT = 1+0.10\*(20+250\*0.05) = 4.25 CPU cycles

3.

a. Code has been attached.

b.

Time taken for i-j-k: 10.8979 seconds

Time taken for i-k-j: 0.642497 seconds

Time taken for j-k-i: 24.1674 seconds

Hence, according to my code i-k-j had the best performance and the arrangement j-k-i had the worst performance which matches our expectation. Considering cache block size to be 64 bytes, for i-j-k we would have 1.125 misses per iteration and for the arrangement j-k-i we would have 2 misses per iteration. However, if j is the inner loop such as in case of i-k-j we would have 0.25 misses per iteration. Hence, for i-k-j we observe that code performance is the best because we get good temporal locality for matrix A and good spatial locality for matrices B and C

1. syscrtl -a | grep l2 revealed:

hw.l2cachesize: 262144 which is 262.144 KB

Let us assume the block size to be m x m

Then 3m2 \* 8 <= 262.144KB

m ~= 104.5

Code has been attached.

1. Execution time of the tiled version: 6.48707 seconds. When we compare it to the i-j-k arrangement without loop tiling, we observe that our code became significantly faster. The performance improvement is almost 40%. This is because loop tiling improves cache reuse and makes vectorization easier. However, i-k-j arrangement still gives me the best performance among all the versions. The tiled version was the second fastest. The reason I believe i-k-j has the best performance is because having j as the inner loop improves both kind of localities and the number of misses is drastically reduced compared to other cases. The block size I chose was 64X64 in order to accommodate the L2 cache.