Assignment 1

Computer Architecture CS3350B

Albirawi, Zaid
250626065

January 26th, 2015

1. Download MatTran1.c and MatTran2.c and compile the two programs using the commands in the comments at the beginning of each file.

    1.1. Tune the MatTran1 program by varying the value of the THRESHOLD parameter to get the best timing on your machine. What is the best value of the THRESHOLD?

Threshold	Time
2	6.612456841
	5.586508412
	5.601219679
4	4.707611488
	5.447765141
	4.689268359
8	4.772195409
	4.238593667
	4.22726377
16	5.115715428
	4.611902991
	4.616933712
32	4.911050513
	4.464731419
	4.462138417
64	7.337525750
	7.320147852
	7.310181031

Therefore, the best THRESHOLD value found is 8.

1.2. What do the two programs do? Which approach does each program take?

The programs perform a transpose operation on the matrices. MatTran1 takes advantage of cache properties to help accelerate the transpose operation. On the other hand, MatTran2 performs an iterative matrix copy operation.

1.3. Show the CPU info of the machine that your measurement is undertaken.

CPU: Intel(R) Xeon(R) CPU E5504 @ 2.00GHz
Cache Size: 4096 KB
Cores: 4
Threads: 8
Cache Alignment: 64
Address Sizes: 40 bits physical, 48 bits virtual

1.4. Choose some proper performance metrics and use perf to measure them for both programs. Which program is faster? Explain briefly why one is faster than the other.

Some of the proper performance metrics are execution time, total number of cycles, cache misses. MatTran1 is faster than MatTran2 because it takes advantage of the cache properties and tries to use that to their advantage.

2. In this exercise we look at memory locality properties of matrix computation. The following code is written in C. The A, B, and D are integer, matrices where elements within the same row are sorted contiguously. Assume each word is a 32-bit integer.

Algorithm 1: Matrix multiplication

```
for ($i = 0; i < 100; ++i$) do
    for ($j = 0; j < 100; ++j$) do
        for ($k = 0; k < 100; ++k$) do
            $D[i][j] = A[i][k] * B[k][j];$
```

2.1. How many 16-byte cache line are needed to store all 32-bit matrix elements of A being referenced? Elaborate the calculation steps.

32-bits = 4-bytes.
16-bytes 4-bytes = 4.

Therefore, the maximum amount of words allowed in each 16-byte cache line is 4 words per line.

A will hold $i * k = 100 * 100 = 10,000$ items. Hence, A will need $10000 \div 4 = 2500$ 16-byte cache lines to store all the matrix elements.

2.2. References to which variables exhibit temporal locality? Explain the reasons.

D exhibits temporal locality as a new value of D gets loaded into the cache once every 100 iterations.

2.3. References to which variables exhibit spatial locality? Explain the reasons.

Both D and A exhibit spatial locality as both arrays are iterated through row-wise.

3. Media applications that play audio or video files are part of a class of workloads called "streaming" workloads; i.e., they bring in large amounts of data but do not reuse much of

it. Consider a video streaming workload that accesses a 512KB working set sequentially with the following address stream:

0, 2, 4, 6, 8, 10, 12, 14, 16, …

In general, cache access time is proportional to capacity. Assume that main memory accesses take 60 ns and that memory accesses are 32% of all instructions. Consider a 64KB direct-mapped L1 cache with a 32-byte cache line. This L1 cache is attached to a processor and its hit time is 0.85 ns.

3.1. What is the miss rate for the address stream above? (Explain reasons.) How is this miss rate sensitive to the size of the cache or the working set?

Since the application is a streaming application, then there will not be a memory address that is requested more than once. Hence, misses will only happen when the program is fetching the next 32-byte line of data from the main memory, cold misses. Also, since the processor is only concerned with every other memory location, 0, 2, 4, 6, 8, 10, 12, 14, 16, …, the requests will be half the line size. Therefore,

$$\begin{aligned} Miss\ rate &= 1 \div (32 \div 2) * 100\% \\ Miss\ rate &= 6.25\% \end{aligned}$$

The miss rate is neither relative to the cache size nor working set sizes.

3.2. What is the Average Memory Access Time for this processor?

$$\begin{aligned} AMAT &= Hit\ time\ +\ Miss\ rate\ *\ Miss\ penalty \\ AMAT &= 0.85ns\ +\ 6.25\%\ *\ 60ns \\ AMAT &= 0.85ns\ +\ 3.75ns \\ AMAT &= 4.60ns \end{aligned}$$

Therefore, the Average Memory Access Time is 4.6ns.

Assumption: L1 is connected directly to the main memory.

3.3. Assuming a base CPI of 1.0 without any memory stalls and the L1 hit time determines the cycle time, what is the total CPI for this processor? What is the time per instruction for this processor?

$CPI_{stall} = CPI_{ideal} + Memory\ Stall\ Cycles$
$CPI_{stall} = CPI_{ideal} + memory\ accesses\ per\ instruction * data\ miss\ rate * cycle\ miss\ penalty$

$memory\ accesses\ per\ instruction = 0.32$
$data\ miss\ rate\ = miss\ rate = 0.0625$
$cycle\ miss\ penalty = miss\ penalty \div cycle\ time$
$cycle\ miss\ penalty = 60ns \div 0.85ns$
$cycle\ miss\ penalty = 1200 \div 17$

Hence,

$CPI_{stall} = CPI_{ideal} + memory\ accesses\ per\ instruction * miss\ rate * cycle\ miss\ penalty$
$CPI_{stall} = 1 + 0.32 * 0.0625 * 1200 \div 17$
$CPI_{stall} = 2.41$

$Time\ per\ instruction = CPI_{stall} * cycle\ time$
$Time\ per\ instruction = (1 + 0.32 * 0.0625 * 1200 \div 17) * 0.85$
$Time\ per\ instruction = 2.05$

Therefore, the total CPI for this processor is 2.41 and the time per instruction is 2.05.

Assumptions: miss rate = data miss rate, there is no instruction cache.

3.4. Consider a 16MB direct-mapped L2 cache with 90% miss rate and 14.5ns hit time. Is it better or worse to attach this L2 to the processor? (Assume a base CPI of 1.0 without any memory stalls.) Explain the reasons.

Since, the L1 cache is attached to the the processor, it is safe to assume that L2 will be sit on top of L1. Therefore,

$$CPI_{stall} = CPI_{ideal} + memory\ accesses\ per\ instruction * L1\ miss\ rate * cycle\ miss\ penalty$$

$memory\ accesses\ per\ instruction = 0.32$
$data\ miss\ rate\ = miss\ rate = 0.0625$
$cycle\ miss\ penalty = L1\ miss\ penalty \div cycle\ time$

$L1\ miss\ penalty = L2\ hit\ time + L2\ miss\ rate\ * L2\ miss\ penalty$
$L1\ miss\ penalty = 14.5ns + 0.9 * 60ns$
$L1\ miss\ penalty = 68.5ns$

Hence,
$cycle\ miss\ penalty = 68.5ns \div 0.85ns$
$cycle\ miss\ penalty = 1370 \div 17$

$CPI_{stall} = 1 + 0.32 * 0.0625 * 1370 \div 17$
$CPI_{stall} = 2.61$

$Time\ per\ instruction = CPI_{stall} * cycle\ time$
$Time\ per\ instruction = (1 + 0.32 * 0.0625 * 1370 \div 17) * 0.85$
$Time\ per\ instruction = 1.37$

To conclude, it is worst to attach the L2 cache to the processor, because its total CPI and time per instruction rates are higher. Therefore, it takes longer to processes instruction.

Assumptions: miss rate = data miss rate, there is no instruction cache, L1 hit time = cycle time.

4. For a direct-mapped cache design with a 16-bit address, the following bits of the address are used to access the cache.

Tag	Index	Offset

15 - 10	9 - 4	3 - 0

4.1. What is the cache line size (in words)?

The cache line size is $2^{offset-bits} = 2^4 = 16\ bytes$ long

The size in words depends on the word size.
The line size for a 4-byte word is 4 words.
The line size for an 8-byte word is 2 words.

4.2. How many entries (i.e. cache lines, or cache blocks) does the cache have?

The number of cache lines/blocks is $2^{index-bits} = 2^6 = 64$ lines/blocks.

4.3. What is the ratio between total bits required for such a cache implementation over the data store bits?

$$Total\ bits = valid\ bit + tag\ bits + Data\ bits$$
$$Total\ bits = 1\ bit + 6\ bits + 16\ bytes$$
$$Total\ bits = 1\ bit + 6\ bits + 128\ bits$$
$$Total\ bits = 135\ bits$$

$$Data\ bits = 128\ bits$$

$$ratio = Total\ bits \div Data\ bits = 135 \div 128\ bits \approx 1.06$$

Therefore, the ratio for the total bits over the store bits is approximately 1.06.

Starting from power on, the following byte-addressed cache references are recorded:
4, 182, 46, 6, 196, 94, 197, 22, 190, 54, 197, 265.

4.4. List the final state of the cache, with each valid entry represented as a record of <index, tag, data> in binary.

Byte	Line	bit-address	Data	Hit
4	4 ÷ 16 = 00	0000 0000 0000 0100	MEM[000-015]	0
182	182 ÷ 16 = 11	0000 0000 1011 0110	MEM[176-191]	0
46	46 ÷ 16 = 02	0000 0000 0010 1110	MEM[032-047]	0
6	6 ÷ 16 = 00	0000 0000 0000 0110	MEM[000-015]	1
196	196 ÷ 16 = 12	0000 0000 1100 0100	MEM[192-207]	0
94	94 ÷ 16 = 05	0000 0000 0101 1110	MEM[080-095]	0
197	197 ÷ 16 = 12	0000 0000 1100 0101	MEM[192-207]	1
22	22 ÷ 16 = 01	0000 0000 0001 0110	MEM[016-031]	0
190	190 ÷ 16 = 11	0000 0000 1011 1110	MEM[176-191]	1
54	54 ÷ 16 = 03	0000 0000 0011 0110	MEM[048-063]	0
197	197 ÷ 16 = 12	0000 0000 1100 0101	MEM[192-207]	1
265	265 ÷ 16 = 16	0000 0001 0000 1001	MEM[256-271]	0

##, <index, tag, data>
00, <000000, 000000, MEM[000-015]>
01, <000001, 000000, MEM[016-031]>
02, <000010, 000000, MEM[032-047]>
03, <000011, 000000, MEM[048-063]>
05, <000101, 000000, MEM[080-095]>
11, <001011, 000000, MEM[176-191]>
12, <001100, 000000, MEM[192-207]>
16, <010000, 000000, MEM[256-271]>

4.5. What is the hit ratio? Elaborate the calculation steps.

$Miss\ rate = 4 \div 12 \approx 0.33$

Therefore, the hit ratio is approximately 0.33, refer to the table in question 4.4 for more information.

4.6. If this cache is 2-way-set-associative, will it improve the hit ratio regarding to those recorded cache references? Explain the reasons.

Byte	Line	bit-address	Data-way 0	Data-way 1	Hit
4	4 ÷ 16 = 00	0000 0000 0000 0100	MEM[000-007]		0
182	182 ÷ 16 = 11	0000 0000 1011 0110	MEM[176-183]		0
46	46 ÷ 16 = 02	0000 0000 0010 1110		MEM[040-047]	0
6	6 ÷ 16 = 00	0000 0000 0000 0110	MEM[000-007]		1
196	196 ÷ 16 = 12	0000 0000 1100 0100	MEM[192-199]		0
94	94 ÷ 16 = 05	0000 0000 0101 1110		MEM[080-087]	0
197	197 ÷ 16 = 12	0000 0000 1100 0101	MEM[192-199]		1
22	22 ÷ 16 = 01	0000 0000 0001 0110	MEM[016-023]		0
190	190 ÷ 16 = 11	0000 0000 1011 1110		MEM[184-191]	0
54	54 ÷ 16 = 03	0000 0000 0011 0110	MEM[048-055]		0
197	197 ÷ 16 = 12	0000 0000 1100 0101	MEM[192-199]		1
265	265 ÷ 16 = 16	0000 0001 0000 1001		MEM[263-271]	0

A 2-way-set-associative cache will not improve the hit ratio regarding to those recorded cache references because none of the recorded cache references evict/overwrite any of the other cache references. 2-way-set-associative cache tries to minimize cache evictions to improve the hit ratio but in this example it proves detrimental to the ratio by decreasing it to $3 \div 12 = 0.25$ fro, .

5. Recall that we have two write policies and write allocation policies, and their combinations can be implemented either in L1 or L2 cache. Assume the following choices for L1 and L2 caches:

L1	L2
Write through, non-write allocate	Write back, write allocate

5.1. Buffers are employed between different levels of memory hierarchy to reduce access latency. For this given configuration, list the possible buffers needed between L1 and L2 caches, as well as L2 cache and memory.

Buffers needed between L1 and L2 caches: write buffer, because L1 has a non-write allocate.

Buffers needed between L2 cache and memory: none.

5.2. Describe the procedure of handling an L2 write-miss, considering the component involved and the possibility of replacing a dirty block.

The procedure of handling an L2 write-miss is to read from the L2 cache, and check the dirty bit. If the dirty bit is set, then write the data to the main memory, because the main memory has not been updated yet. Then load the new data into the L2 cache and mark it as dirty.