COMPUTER ARCHITECTURE

ASSIGNMENT-2 CACHES

DIPINTI MANANDHAR (M20CS020)

Problem-1

Write a RISC-V code for the following C code:

```
#define size 1024
int main ()
        {
                int reps, int step, option, I;
                reps =10;
                step=2;
                option=1;
                char a[size];
                for(i=0;reps;i++)
                {
                        for (j=0; j<size; j+=step)
                        {
                                if(option ==0)
                                a[0] =0;
                                else
                                a[j]+=1;
                        }
                }
                return 1;
        }
```

Implementation code with RISC-V is given below:

```
.data
a: .word 2048

.text
li a0,1024 #size=1024
li a1,2 #step=2
li a2,4 #reps=10
li a3,1 #option=1

jal a_array

li a0,10
ecall
```

```
a_array:

la s0,a
add s1,s0,a0
slli t1,a1,2

loop1:
beq a3,x0,stzero
lb t0,0(s0)
addi t0,t0,1
sb t0,0(s0)

j loop2
stzero:
sb x0,0(s0)
loop2:
add s0,s0,t1
blt s0,s1,loop1
addi a2,a2,-1
bgtz a2, a_array
jr ra
```

Answer the following common questions first:

- How big is your cache block?
  Ans: Cache block is 8 bytes.

- How many consecutive accesses (taking into account the step size) fit within a single block?
  Ans:
  Since here step=2, option=1, we have:
  a[j]+=1;
  When a[j] is fetched for 1st time, there will be cache miss. Here, our cache block is 8 bytes, char type size is 1 bytes. The entire block of 8 bytes is fetched from memory into cache. So in the 1st line of cache, 0 to 7th char data from memory gets stored. Then, when there is increment operation and storing it back, there will be a hit. Then for Access2, there will be a hit as data is in cache. Since step=2, there won't be consecutive accesses in memory fit within a single block. So the 2nd element will get accessed leaving the 1st element and so on.
  Therefore, there won't be consecutive accesses. There will be alternate accesses of step 2.

- How much data fits in the whole cache?
  Ans: Since we are considering cache block of size 8 bytes, number of lines 4, a total of 4*8=32 bytes of data fits in the whole cache.

- How far apart in memory are blocks that map to the same set (and could create conflicts)?
  Ans: Since a total of 32 bytes can fit in the cache, after the 32 bytes of data are fetched from memory to cache, for the next bytes, there will be conflicts. So block in memory are 3 blocks far apart that map to the same set and could create conflicts.

- What is your cache's associativity?
  Ans: Since we have considered Direct mapping, the cache's associativity is 1.

- Where in the cache does a particular block map to?
  Ans: A particular block maps to the fixed cache line.

Setting-1:

1. When considering why a specific access is a miss or hit: Have you accessed this piece of data before? If so, is it still in the cache or not?

Ans: A miss will occur when there is not availability of data in the cache. After the cache if full, the blocks get overwritten. When we access a block, after the task is completed, it will be evicted out of the cache. Therefore, the piece of data will not be in cache after accessing it.

2. What combination of parameters is producing the hit rate you observe? Write your answer in the form "[parameter A], [parameter B]" where the two parameters complete the following response: "Because [parameter A] in bytes is exactly equal to [parameter B] in bytes." Note: Don't forget that 'cache size' is a valid parameter that you implicitly set by choosing the block size and the # of blocks.

Ans: The [cache size] is 32 bytes which is equal to the [step size]. The hit rate of this combination of parameters is producing hit rate zero as each block's index which are accessed becomes identical and the blocks get overwritten.

3. What is our hit rate if we increase Rep Count arbitrarily? Write your answer as a decimal (e.g. "1.0" if the HR is 100%).

Ans: The hit rate will always be zero. The hit rate doesnot gets affected by the number of repeatation time, i.e. Rep Count. Since the step size is 8, the lower 3 bits will always be same.

4. How could we modify one program parameter get the highest possible hit rate? Write your answer in the form "[parameter], [value]" where [parameter] is the program parameter you want to change and [value] is the value you want to change it to. Note: We don't care if we access the same array elements. Just give us a program parameter modification that would increase the hit rate. However, do make sure that your proposed value is valid.

Ans: When we change the step size to 1, the hit rate will increase to 0.5. When the step size is 1, every load will get two blocks and we can access the next block in the cache.

Setting-2

1. How many memory accesses are there per iteration of the inner loop (not the one involving Rep Count)?

   Ans: There are 2 memory accesses per iteration of the inner loop. We can easily conclude from the code: a[j]+=1

2. What is the repeating hit/miss pattern? Write your answer in the form "MMHHMH" and so on, where your response is the shortest pattern that gets repeated.
   Ans: The hit will be: Access 2 of block 0 and both access of block 2. The miss will occur in the Access 1 of block 0. This is due to the fact that there are a total of 4 blocks in each line of cache and step size is 2. The pattern is "MHHH".

3. Keeping everything else the same, what does our hit rate approach as Rep Count goes to infinity? Try it out by changing the appropriate program parameter and letting the code run! Write your answer as a decimal.
   Ans: When the repeat time increases, there is increase in hit rate. When putting the value of reps=1000, we get hit rate of 0.9998. Therefore, from this hit rate, we can conclude that data will not get deleted as the cache size is very huge.

Setting-3:

1. What is the hit rate of our L1 cache? Our L2 cache? Overall? Write your answer in the form "[L1 HR], [L2 HR], [Overall HR]" where each hit rate is a decimal rounded to two places.
   Ans: [L1 HR] =50%
   [L2 HR] = 0
   [Overall HR] = 25%

2. How many accesses do we have to the L1 cache total? How many of them are misses? Write your answer in the form "[# of L1 accesses], [# of L1 misses]".
   Ans: [# of L1 accesses] =32
   [# of L1 misses] = 16

3. How many accesses do we have to the L2 cache total? HINT: Think about how this relates to the L1 cache (think about what the L1 cache has to do in order to make us access the L2 cache)?
   Ans: A total of 16 accesses is there as there are 16 miss in L1 cache which leads to accessing in cache L2.

4. What program parameter would allow us to increase our L2 hit rate, but keep our L1 hit rate the same?
   Ans: L2 hit rate can be increased when we increase the number of repeat time. When rep time is 2, the L2 hit rate obtained was 0.5. Similarly when rep time is 4, the L2 hit rate obtained was 0.75. Therefore, we can conclude that L2 cache is huge.

5. Do our L1 and L2 hit rates decrease (-), stay the same (=), or increase (+) as we (1) increase the number of blocks in L1, or (2) increase the L1 block size? Write your answer in the form "[1_L1], [1_L2], [2_L1], [2_L2]" (e.g. if I thought L1 will stay the same for both modifications while L2 will decrease for the first and increase for the second, I would answer "=, -, =, +").
   Ans: When there is increase in number of L1 blocks, L1 hit rate will increase and L2 hit rate will decrease.
   [1_L1] -> '+'
   [1_L2] -> '-'
   [2_L1] -> '+'
   [2_L2] ->'-'

Problem-2

Solutions:

The code discussed in class is given below:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<sys/time.h>
#include<cstdlib>
#define drand48 1.0/RAND_MAX * rand
#define N 1000
#define index(i,j) (i+(N*j))

int matMult(float *c, float *a, float *b)
{
	int i,j,k;

	for(k=0;k<N;k++)
	{
		for(j=0;j<N;j++)
		{
			for(i=0;i<N;i++)
			{
				c[index(i,j)] += a[index(i,k)]*b[index(k,j)];
			}
		}
	}
}

int main()
{
	float *a,*b,*c; //WHERE WILL THIS ALLOCATE? STACK
	a=(float*)malloc(N*N*sizeof(float)); //WHERE WILL THIS ALLOCATE? HEAP
	b=(float*)malloc(N*N*sizeof(float)); //WHERE WILL THIS ALLOCATE? HEAP
	c=(float*)malloc(N*N*sizeof(float)); //WHERE WILL THIS ALLOCATE? HEAP
	int i;
```

```
        struct timeval start, end;

        for(i=0; i<N*N; i++){
        a[i]=drand48()*2-1;
        b[i]=drand48()*2-1;
        c[i]=0;
}

gettimeofday(&start, NULL);

matMult(c,a,b);

gettimeofday(&end, NULL);

double elapsed_time=(end.tv_sec - start.tv_sec)+ 1.0e-6*(end.tv_usec - start.tv_usec);
printf("Elapsed Time is %lf\n",elapsed_time);

double gflops = 2e-9*N*N*N/elapsed_time;

printf("GFLOPS is %lf\n",gflops);

return 0;
}
```

Question 1:

Which 2 ordering performs best for these 1000-by-1000 matrices? Write your answer in the form "[Ordering1],[Ordering2]" (e.g. "ijk, ikj").

Answer:

By implementing the above code and noting down all the possible combinations of the variables "i", "j", "k", the results are shown below:

ORDER	ELAPSED TIME	GFLOPS
i j k	3.7532	0.532866
i k j	7.462322	0.268013
j i k	3.912155	0.511227
k i j	7.097611	0.2817855
j k i	**3.339576**	**0.598878**
k j i	**3.420029**	**0.584790**

From the above table, we can clearly say that the orders "j k i" and "k j i" has least elapsed time and highest GFLOPS than other orderings. Therefore, the 2 orderings that performs best for these 1000-by-1000 matrices are:
[j k i], [k j i]

Question 2:

Explain why this ordering works best.

Answer:

For order [j k i], the loop will be as follows:

```
for(j=0;j<N;j++)
{
        for(k=0;k<N;k++)
        {
                for(i=0;i<N;i++)
                {
                        c[index(i,j)] += a[index(i,k)]*b[index(k,j)];  #line 1
                }
        }
}
```

Explanation:

The outermost loop depends on the 'j' value. Since here in *line 1,* we can see that j is being accessed. So when 'j' is kept in outer loop, the 'j' in the innermost loop will get affected minimum and 'j' will be changed minimum number of times. Therefore, there is great utilization of cache as two loops i.e. 'k' and 'i' loop will be completed and only value of 'j' will be incremented. The order has a spatial locality. This ordering works best than [k j i] as in the 'line1', there is involvement of 'k' in both the matrix 'a' and 'b'. Therefore, the elapsed time of the order [j k i] is the least and GFLOPS is highest. There will be maximum amount of utilization because 'j' index will happen inside the inner most loop.

For order[k j i], the loop will be as follows:

```
for(k=0;k<N;k++)
{
        for(j=0;j<N;j++)
        {
                for(i=0;i<N;i++)
                {
                        c[index(i,j)] += a[index(i,k)]*b[index(k,j)];  #line 1
                }
        }
}
```

Explanation:

It is similar to the previous order of [j k i]. 'k' needs to be change minimum and when it is kept in the outermost position then the 'k' in 'line 1' will get affected the minimum. The two loops 'j' and 'i' will be completed and only the value of 'k' will change. Therefore, there is maximum utilization as 'k' index will happen inside the inner most loop. It's elapsed time is slower than [j k i] and GFLOPS is 2nd highest than [j k i] because there is involvement of 'k' in both the matrices 'a' and 'b'.

<u>Question 3</u>

Which two ordering performs the worst?

Answer

From the table, we can say that the orders [i k j] and [k i j] performs the worst as it has the highest elapsed time and lowest GFLOPS.

Implementation of cache blocking in the transpose_blocking() function inside the code provided in transpose.c :
We have to write the following code:

```
void transpose_blocking(int n, int blocksize, int *d, int *s) {
  for (int i = 0; i < n; i += blocksize)
   for (int j = 0; j < n; j += blocksize)
      for (int k = i; k < i + blocksize && k < n; ++k)
         for (int l = j; l < j + blocksize && l < n; ++l){
            d[k + l*n] = s[l + k*n];
}}
```

Setting-1: Changing array sizes

- Fix the blocksize to be 20, and run your code with n equal to 100, 1000, 2000, 5000, and 10000. Record your output in exercise3.txt.
  Answer:

Block size	n	Testing naïve transpose (milliseconds)	Testing transpose with blocking (milliseconds)
20	100	0.028	0.034
20	1000	6.438	3.79
20	2000	41.333	14.5
20	5000	359.256	90.366
20	10000	1720.698	390.256

Setting-1 : Changing array sizes

Question 1

At what point does cache blocked version of transpose become faster than the non-cache blocked version?

Ans: At n=1000, the cache blocked version of transpose become faster than the non-cache blocked version.

Question 2

Why does cache blocking require the matrix to be certain size before It outputs the non-cache blocked code?

Ans: The advantage obtained by locality is minimum than the program complexity. Due to this reason, cache blocking require the matrix to be certain size before it outputs the non-cache blocked code.

Setting-2 : Changing blocksize

How does performance change as blocksize increases? Why is this the case?

Ans: There is no difference in naïve transpose and blocking transpose. All the advantages from the locality will vanish if the block size increases.