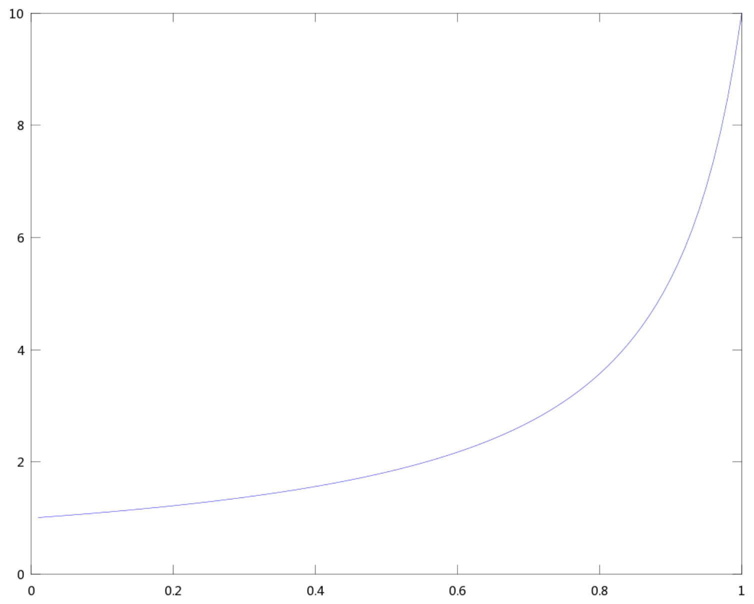
1.14

a



y=10 / (10-9*x)

b

2= 10/(10 - 9*x)

x= 5/9 = 0.56

So, the percentage needed is about 0.56

c

(0.56/10)/(1-0.56+0.56/10) = 1/9 = 0.11111

d

Maximun speedup = 10 / (10 -9*1) = 10

5 = 10 /(10 - 9*x)

x= 8/9 = 0.89

e

When the percentage of vectorization is 70% Speeup = 10 / (10 - 9*0.7) = 2.7

plus the addtional speedup of 2

4.7 = 10 / (10 - 9*x)

x = 0.87

1.17

a

1 / (0.4 /2 + 0.6) = 1.25

b

1 / (0.01 + 0.99/2) = 1.98

c

1 / (0.2 + 0.8 * (0.6+ 0.4/2)) = 1.19

d

1/ (0.8 + 0.2 * (0.01 + 0.99/2)) = 1.11

2.1

a

double precision element is 8B and the block size is 64B, it can hold 8 element. so we can divide the matrix to 4 8x8 matrix, then the minimum size of the cache for 2 8x8 matrix is:

2 x 8 x 8 x 8B = 1KB

b

The block version: When visiting the 8x8 matrix, each row will evoke one miss, the first column will evoke 8 misses, but they will be cached up, so ,the totall misses for one 8x8 matrix is:

8 + 8

The unblock version 64B block can hold 8 element , when visiting a row of 256 elements, the missrate is 1/8 but when visiting columns, one visit will evoke a miss and after 256 visits of a column, the 16KB cache is full and next column will evoke misses too. so the totall misses for one 8x8 matrix is:

8x8+8

so 9 misses for the unblock version compared to 2 misses for the block version.

c

```
for(int ii=0;i<N;i+=B){
    for(int jj=0;j<N;j+=B){
        for(int i=ii;i<min(N,ii+B);i++){
            for(int j=jj;j<min(N,jj+B);j++){
                output[i][j]=input[j][i];
            }
        }
    }
}
```

d

2 way set associative.

e

the result is related to the band width of the l2 cache and the size to l1 cache of the computer.

2.8

a

direct-mapped:0.86ns two-way : 1.12ns four-way : 1.37ns so two-way requires 1.12/0.86 = 1.30 times of the direct-mapped.

four-way requires 1.37/0.86 = 1.56 times of the direct-mapped.

b

16KB : 1.27ns 32KB : 1.35ns 64KB : 1.37ns

So:

The access time of 32KB is 1.35/1.27 = 1.06 of that of 16KB.

The access time of 32KB is 1.37/1.27 = 1.08 of that of 16KB.

c

Miss per referenced instruction :

```
DM :   0.006664 / 0.3 = 0.022

2-Way: 0.00366  / 0.3 = 0.012

2-Way: 0.009887 / 0.3 = 0.033

2-Way: 0.000266 / 0.3 = 0.00088
```

Hit Cycles The cacti access time is 0.5ns

```
DM :    0.87 / 0.5  = 2 cycles

2-Way : 1.12 / 0.5  = 3 cycles

4-Way : 1.37 / 0.83 = 2 cycles

8-Way : 2.03 / 0.79 = 3 cycles
```

Miss Penalty:

```
DM :    10 / 0.5  = 20 cycles

2-Way : 10 / 0.5  = 20 cycles

4-Way : 10 / 0.83 = 13 cycles

8-Way : 10 / 0.79 = 13 cycles
```

AVG Access Time :

```
DM :    0.022 * 20 + (1-0.022) * 2 = 2.3960 * 0.5 = 1.198

2-Way : 0.012 * 20 + (1-0.012) * 3 = 3.2040 * 0.5 = 1.602

4-Way : 0.033 * 13 + (1-0.033) * 2 = 2.3630 * 0.83= 1.961

8-Way : 0.0008* 13 + (1-0.0008)* 3 = 3.0088 * 0.79= 2.377
```

The lowest is direct-mapped

2.10

a

Cycle time : 0.51ns

Access time: 1.12ns

1.12 / 0.51 = 2.19 pipestages

b

```
             Area            Energy per access
Pipelined    1.19mm^2        0.16nJ
Banked       1.36mm^2        0.13nJ
```

The banked design requires more area because it need more circuit to decide wether hit or not, it consumes less engergy because its active memory arrays are less then the pipelined design.

2.12

a

16B

b

None merging write buffer takes 2 cycles to fill one L2 entry, it ends up taking 8 cycles.

Merging write buffer takes only 4 cycles.

The speedup is 2.

c

With non-blocking caches, write request can be process from the write buffer, it result in needing less entries.

Practices

2.1

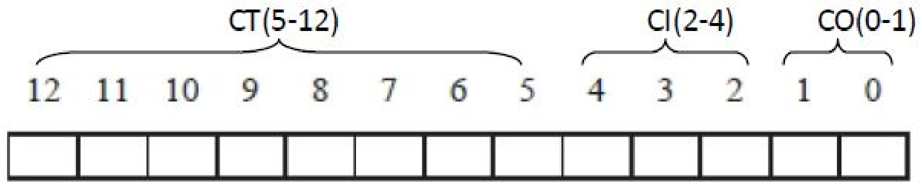


Figure 1: Figure2.1

2.2

a

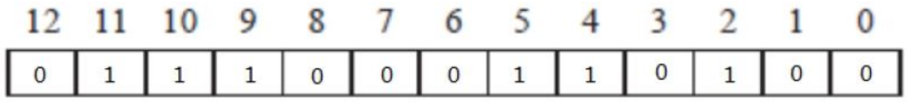


Figure 2: Figure2.2.a

b

2.3

a

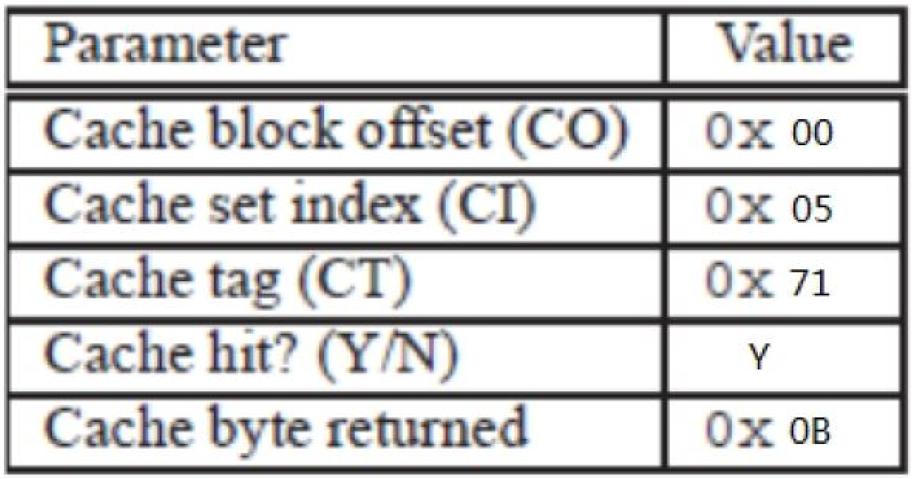
b

2.4

a

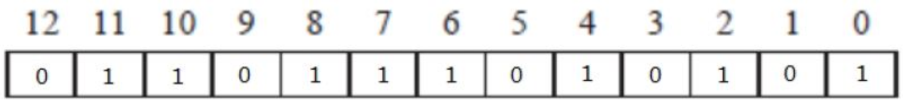
b

2.5



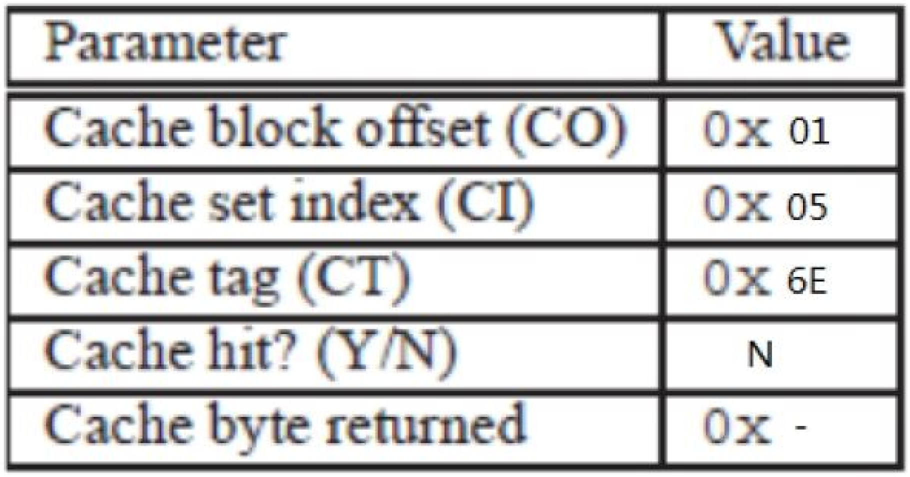
Parameter	Value
Cache block offset (CO)	0x 00
Cache set index (CI)	0x 05
Cache tag (CT)	0x 71
Cache hit? (Y/N)	Y
Cache byte returned	0x 0B

Figure 3: Figure2.2.b



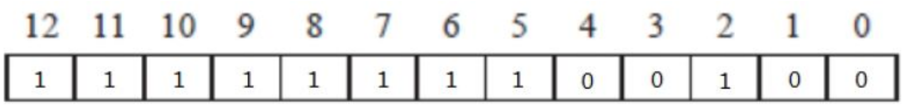
12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	0	1	0	1	0	1

Figure 4: Figure2.3.a



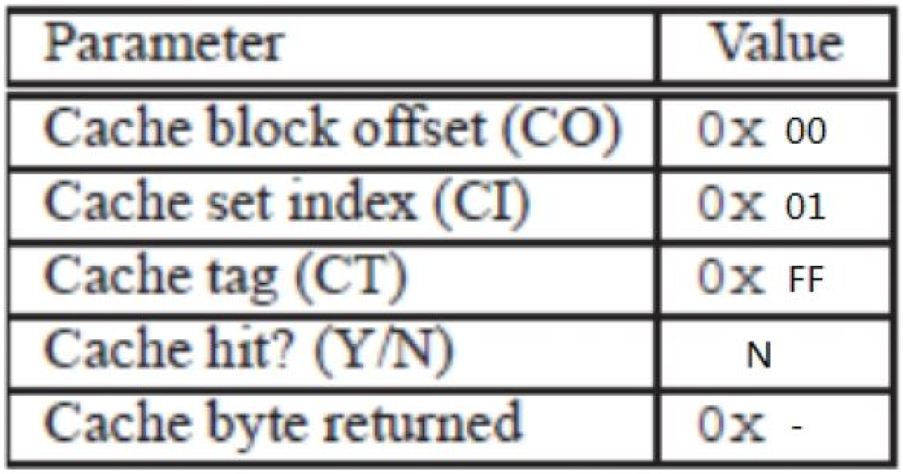
Parameter	Value
Cache block offset (CO)	0x 01
Cache set index (CI)	0x 05
Cache tag (CT)	0x 6E
Cache hit? (Y/N)	N
Cache byte returned	0x -

Figure 5: Figure2.3.b



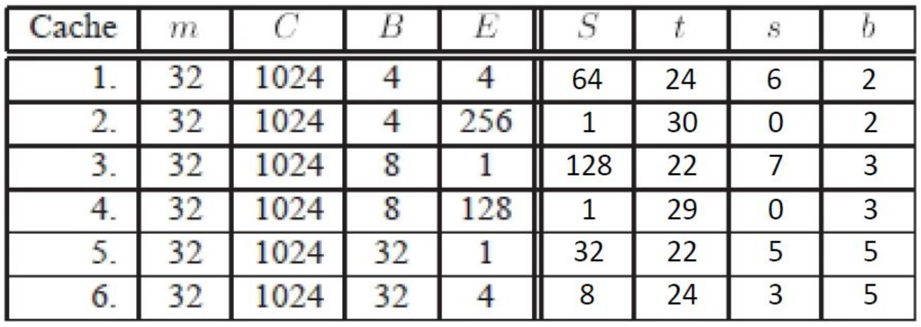
12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	0	0	1	0	0

Figure 6: Figure2.4.a



Parameter	Value
Cache block offset (CO)	0x 00
Cache set index (CI)	0x 01
Cache tag (CT)	0x FF
Cache hit? (Y/N)	N
Cache byte returned	0x -

Figure 7: Figure2.4.b



Cache	m	C	B	E	S	t	s	b
1.	32	1024	4	4	64	24	6	2
2.	32	1024	4	256	1	30	0	2
3.	32	1024	8	1	128	22	7	3
4.	32	1024	8	128	1	29	0	3
5.	32	1024	32	1	32	22	5	5
6.	32	1024	32	4	8	24	3	5

Figure 8: Figure2.5