CENG 3420
Computer Organization & Design

HW4 Review

Zhisheng ZHONG
CSE Department, CUHK
zszhong@link.cuhk.edu.hk

Spring 2022

Overview

Overview

Q1 Description

Parameter	Description
$S = 2^s$	Number of sets
E	Number of blocks
$B = 2^b$	Block size (bytes)
$m = \log_2(M)$	Number of main memory address bits
$M = 2^m$	Maximum number of unique memory addresses
$s = \log_2(S)$	Number of set index bits
$b = \log_2(B)$	Number of block offset bits
$t = m - (s + b)$	Number of tag bits
$C = B \times E \times S$	Cache size not including overhead (valid and tag bits)

Complete the form for different caches below:

Cache id	m	C	B	E	S	t	s	b
1.	32	1024	4	1				
2.	32	1024	8	4				
3.	32	1024	32	32				

Solution 1

$$\text{For} \quad S: \quad C = B \times E \times S \quad \rightarrow \quad S = C/B/E, \tag{1}$$

$$\text{For} \quad s: \quad s = \log_2(S), \tag{2}$$

$$\text{For} \quad b: \quad b = \log_2(B), \tag{3}$$

$$\text{For} \quad t: \quad t = m - (s + b). \tag{4}$$

According to Equations (1), (2), (3), and (4), we can fill the table:

Cache id	m	C	B	E	S	t	s	b
1.	32	1024	4	1	256	22	8	2
2.	32	1024	8	4	32	24	5	3
3.	32	1024	32	32	1	27	0	5

Overview

Q2.1, Q2.2, and Q2.3 Description

A processor has a 32-bit memory address space (i.e. 32-bit addresses). The memory is broken into blocks of 32 bytes each. The computer also has a cache capable of storing 16K bytes.

1. How many blocks can the cache store?
2. **direct-mapping**:

Tag	Block	Byte offset

3. **2-way set associative mapping**:

Tag	Set	Byte offset

Solution of Q2.1, Q2.2, and Q2.3

1. #(Block) = 16 K bytes/32 bytes = $(2^4 \times 2^{10})/2^5 = 2^9 = 512$ blocks.
2. **Direct mapping**

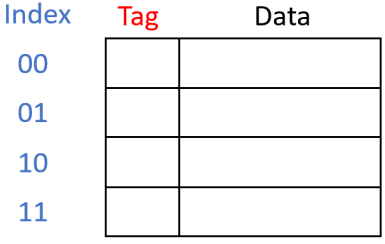
Tag	Block	Byte offset
A31-A14 (18 bits)	A13-A5 (9 bits)	A4-A0 (5 bits)
Remaining bits (32 - 5 - 9 = 18)	512 blocks = 2^9, 9 address bits	32 bytes per block

3. **2-way set associate mapping**

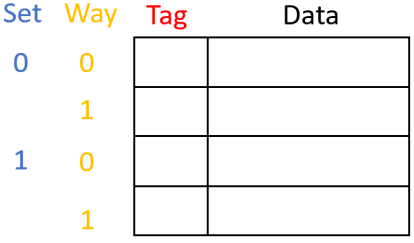
Tag	Set	Byte offset
A31-A13 (19 bits)	A12-A5 (8 bits)	A4-A0 (5 bits)
Remaining bits (32 - 5 - 8 = 19)	512 blocks/2-way= 2^8, 8 address bits	32 bytes per block

Q2.4 Description

Consider the following two empty caches, calculate cache hit rates for the reference word addresses: (a) “0, 4, 0, 4, 0, 4, 0, 4”; (b) “0, 3, 0, 3, 0, 3, 0, 3”.

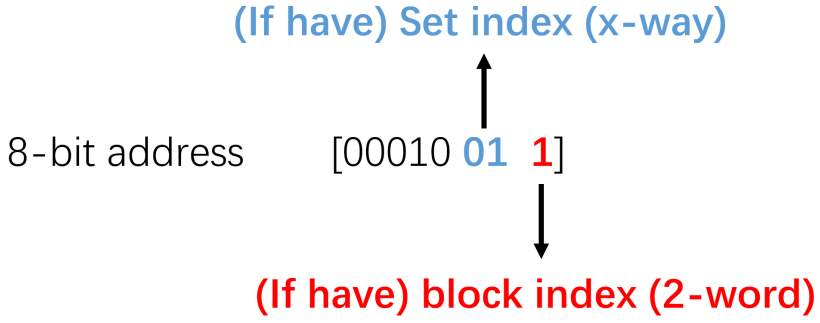


Direct Mapping



2-Way Set Associative

Note



There is no direct mathematical relationship between the number way and set.
In this question, there is no block index for the 2-way associative mapping cache.
$4 = (10\mathbf{0})_2, \quad 3 = (01\mathbf{1})_2, \quad 0 = (00\mathbf{0})_2.$
Thus, 4 and 0 belong to the same set.

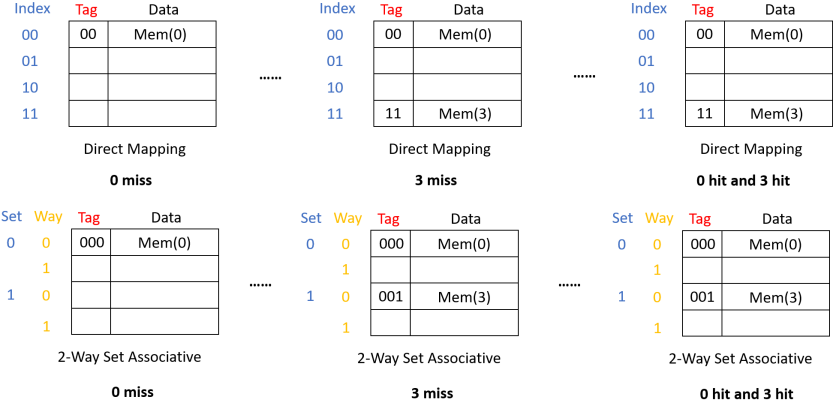
Solution of Q2.4

For "0, 4, 0, 4, 0, 4, 0, 4", the contents of the two caches are shown in the following:

The hit rate of direct mapping cache is $\frac{0}{8} = 0$.
The hit rate of 2-way associative mapping cache is $\frac{6}{8} = 75\%$.

Solution of Q2.4

For “0, 3, 0, 3, 0, 3, 0, 3”, the contents of the two caches are shown in the following.



The hit rate of direct mapping cache is $\frac{6}{8} = 75\%$.
The hit rate of 2-way associative mapping cache is $\frac{6}{8} = 75\%$.

Overview

Q3 Description

In the following questions, start from an empty cache and give the contents of the cache after the following sequence of memory references (addresses are hexadecimal numbers): A0, F1, FF, 35, C8, 89, FE, 88, A1, A2, A3, A9, 99, 80, 83.

1. A block=2 (2-word), 4-way cache with the LRU replacement.
2. A block=4 (4-word), 2-way cache with the FIFO replacement.

Note

Multi-word Direct Mapping: a more sophisticated version of a normal Direct Mapping. **A sequence of address** (data) will be input into the same block.

Tag	**data1**	**data2**
	A0	A1
	00	01

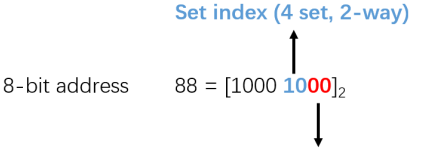
Least Recently Used (LRU): meaning it discards the least recently used items first.

First In, First Out (FIFO): similar to queue.

Solution of Q3

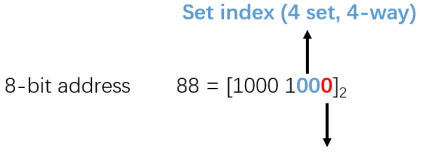
Similar to Q2.4, here we give an example for illustration:

A block=4, 2-way cache



	data1	data2	data3	data4	index
Set 10	mem(88)	mem(89)	mem(8A)	mem(8B)	-

A block=2, 4-way cache



	data1	data2	index
Set 00	mem(88)	mem(89)	-

Solution of Q3.1

Set 00

data1	data2	LRU/Total
mem(A0)	mem(A1)	0 / 0
mem(F0)	**mem(F1)**	1 / 1
mem(C8)	mem(C9)	2 / 4
mem(88)	**mem(89)**	3 / 5

Set 01

data1	data2	LRU/Total

Set 10

data1	data2	LRU/Total
mem(34)	**mem(35)**	0 / 3

Set 11

data1	data2	LRU/Total
mem(FE)	**mem(FF)**	0 / 2

Step 1: memory references A0, F1, FF, 35, C8, 89, all misses

Set 00

data1	data2	LRU/Total
mem(A0)	**mem(A1)**	**3 / 8**
mem(F0)	mem(F1)	**0 / 1**
mem(C8)	mem(C9)	**1 / 4**
mem(88)	mem(89)	**2 / 7**

Set 01

data1	data2	LRU/Total

Set 10

data1	data2	LRU/Total
mem(34)	mem(35)	0 / 3

Set 11

data1	data2	LRU/Total
mem(FE)	mem(FF)	0 / 6

Step 2: memory references FE, 88, A1: All hits, spatial locality, notice the change in index

Set 00

data1	data2	LRU/Total
mem(A0)	mem(A1)	3 / 8
mem(F0)	mem(F1)	0 / 1
mem(C8)	mem(C9)	1 / 4
mem(88)	mem(89)	2 / 7

Set 01

data1	data2	LRU/Total
mem(A2)	**mem(A3)**	**0 / 10**

Set 10

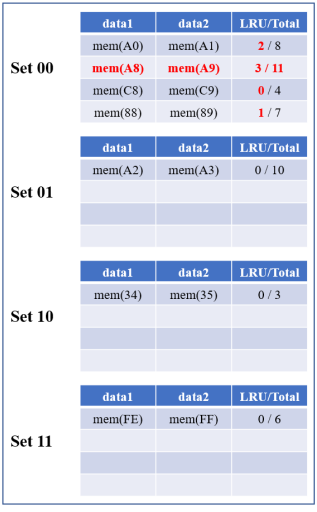
data1	data2	LRU/Total
mem(34)	mem(35)	0 / 3

Set 11

data1	data2	LRU/Total
mem(FE)	mem(FF)	0 / 6

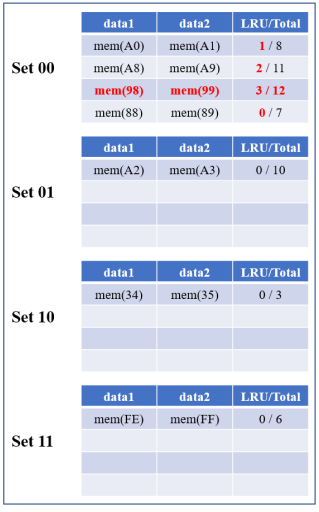
Step 3: memory references A2: miss, memory references A3: hit, spatial locality

Solution of Q3.1



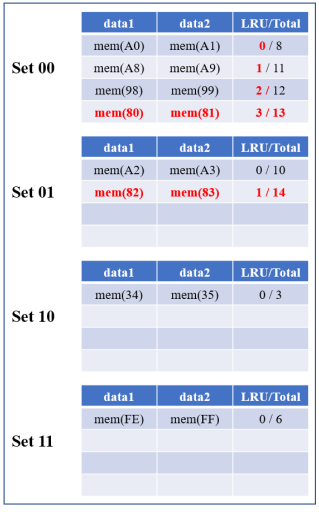
	data1	data2	LRU/Total
Set 00	mem(A0)	mem(A1)	2 / 8
	mem(A8)	mem(A9)	3 / 11
	mem(C8)	mem(C9)	0 / 4
	mem(88)	mem(89)	1 / 7
Set 01	mem(A2)	mem(A3)	0 / 10
Set 10	mem(34)	mem(35)	0 / 3
Set 11	mem(FE)	mem(FF)	0 / 6

Step 4: memory references A9: miss, replace (F0, F1)



	data1	data2	LRU/Total
Set 00	mem(A0)	mem(A1)	1 / 8
	mem(A8)	mem(A9)	2 / 11
	mem(98)	mem(99)	3 / 12
	mem(88)	mem(89)	0 / 7
Set 01	mem(A2)	mem(A3)	0 / 10
Set 10	mem(34)	mem(35)	0 / 3
Set 11	mem(FE)	mem(FF)	0 / 6

Step 5: memory references 99: miss, replace (C8, C9)



	data1	data2	LRU/Total
Set 00	mem(A0)	mem(A1)	0 / 8
	mem(A8)	mem(A9)	1 / 11
	mem(98)	mem(99)	2 / 12
	mem(80)	mem(81)	3 / 13
Set 01	mem(A2)	mem(A3)	0 / 10
	mem(82)	mem(83)	1 / 14
Set 10	mem(34)	mem(35)	0 / 3
Set 11	mem(FE)	mem(FF)	0 / 6

Step 6: memory references 80: miss, replace (88, 89); memory references 83: miss. finish

Solution of Q3.2

Set 00

data1	data2	data3	data4	FIFO/Total
mem(A0)	mem(A1)	mem(A2)	mem(A3)	0 / 0
mem(F0)	**mem(F1)**	mem(F2)	mem(F3)	1 / 1

Set 01

data1	data2	data3	data4	FIFO/Total
mem(34)	**mem(35)**	mem(36)	mem(37)	0 / 3

Set 10

data1	data2	data3	data4	FIFO/Total
mem(C8)	mem(C9)	mem(CA)	mem(CB)	0 / 4
mem(88)	**mem(89)**	mem(8A)	mem(8B)	1 / 5

Set 11

data1	data2	data3	data4	FIFO/Total
mem(FC)	mem(FD)	mem(FE)	**mem(FF)**	0 / 2

Step 1: memory references A0, F1, FF, 35, C8, 89, all misses

Set 00

data1	data2	data3	data4	FIFO/Total
mem(A0)	**mem(A1)**	**mem(A2)**	**mem(A3)**	**1 / 10**
mem(F0)	mem(F1)	mem(F2)	mem(F3)	0 / 1

Set 01

data1	data2	data3	data4	FIFO/Total
mem(34)	mem(35)	mem(36)	mem(37)	0 / 3

Set 10

data1	data2	data3	data4	FIFO/Total
mem(C8)	mem(C9)	mem(CA)	mem(CB)	0 / 4
mem(88)	mem(89)	mem(8A)	mem(8B)	1 / **7**

Set 11

data1	data2	data3	data4	FIFO/Total
mem(FC)	mem(FD)	**mem(FE)**	mem(FF)	0 / **6**

Step 2: memory references FE, 88, A1, A2, A3: All hits, spatial locality, notice the change in FIFO index

Solution of Q3.2

Set 00

data1	data2	data3	data4	FIFO/Total
mem(A0)	mem(A1)	mem(A2)	mem(A3)	1 / 10
mem(F0)	mem(F1)	mem(F2)	mem(F3)	0 / 1

Set 01

data1	data2	data3	data4	FIFO/Total
mem(34)	mem(35)	mem(36)	mem(37)	0 / 3

Set 10

data1	data2	data3	data4	FIFO/Total
mem(A8)	**mem(A9)**	mem(AA)	mem(AB)	**1** / 11
mem(88)	mem(89)	mem(8A)	mem(8B)	**0** / 7

Set 11

data1	data2	data3	data4	FIFO/Total
mem(FC)	mem(FD)	mem(FE)	mem(FF)	0 / 6

Step 3: memory reference A9: misses, replace (C8, C9, CA, CB). notice the change in FIFO index

Set 00

data1	data2	data3	data4	FIFO/Total
mem(A0)	mem(A1)	mem(A2)	mem(A3)	**0** / 10
mem(80)	mem(81)	mem(82)	**mem(83)**	**1 / 14**

Set 01

data1	data2	data3	data4	FIFO/Total
mem(34)	mem(35)	mem(36)	mem(37)	0 / 3

Set 10

data1	data2	data3	data4	FIFO/Total
mem(A8)	mem(A9)	mem(AA)	mem(AB)	**0** / 11
mem(98)	**mem(99)**	mem(9A)	mem(9B)	**1 / 12**

Set 11

data1	data2	data3	data4	FIFO/Total
mem(FC)	mem(FD)	mem(FE)	mem(FF)	0 / 6

Step 4: memory reference 99, 80: misses, replace (88, 89, 8A, 8B) and (F0, F1, F2, F3), notice the change in FIFO index. Memory reference 83: hits, spatial. finish

Overview

Q4 Description

Suppose the access times to the cache and the main memory are 50 ns and 200 ns respectively. When the CPU runs a program, it accesses the cache 2000 times and main memory 50 times. Calculate the hit rate and access efficiency of this cache-main memory system.

Solution of Q4

Denote t_c (the cache access times), t_m (the main memory access times).
The following three solutions can get full marks:

1. 1 miss = 1 main memory access, miss penalty = t_m
2. 1 miss = 1 main memory access + 1 cache access, miss penalty = $t_m + t_c$
3. (**optimal**, refer to Lecture 17, Page 22)
   1 miss = 1 main memory access + 2 cache access, miss penalty = $t_m + 2t_c$

Case 1: 1 miss = 1 main memory access, miss penalty = t_m

The hit rate:

$$h = \frac{N_h}{N_h + N_m} = \frac{2000}{2000 + 50} \approx 0.976 = 97.6\%.$$

The average memory access time:

$$t_a = ht_c + (1 - h)M = ht_c + (1 - h)t_m,$$

The access efficiency:

$$e = \frac{t_c}{t_a} = \frac{t_c}{ht_c + (1 - h)t_m} = \frac{50}{0.976 \times 50 + (1 - 0.976) \times 200} \approx 0.93 = 93\%.$$

Case 2: 1 main memory access + 1 cache access, miss penalty = $t_m + t_c$

The hit rate:

$$h = \frac{N_h}{N_h + N_m} = \frac{2000 - 50}{2000} = 0.975 = 97.5\%.$$

The average memory access time:

$$t_a = ht_c + (1 - h)M = ht_c + (1 - h)(t_m + t_c),$$

The access efficiency:

$$e = \frac{t_c}{t_a} = \frac{t_c}{ht_c + (1 - h)(t_m + t_c)} = \frac{50}{0.975 \times 50 + (1 - 0.975) \times 250} \approx 0.91 = 91\%.$$

Case 3: 1 miss = 1 main memory access + 2 cache access, miss penalty = $t_m + 2t_c$

The hit rate:

$$h = \frac{N_h}{N_h + N_m} = \frac{2000 - 50 \times 2}{2000 - 50 \times 2 + 50} \approx 0.974 = 97.4\%.$$

The average memory access time:

$$t_a = ht_c + (1 - h)M = ht_c + (1 - h)(t_m + 2t_c),$$

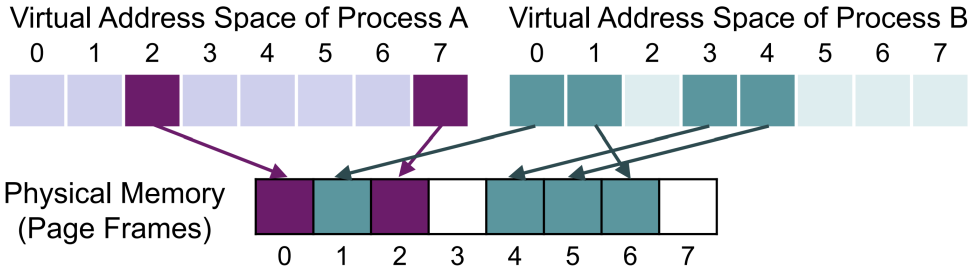
The access efficiency:

$$e = \frac{t_c}{t_a} = \frac{t_c}{ht_c + (1 - h)(t_m + 2t_c)} = \frac{50}{0.974 \times 50 + (1 - 0.974) \times 300} \approx 0.89 = 89\%.$$

Overview

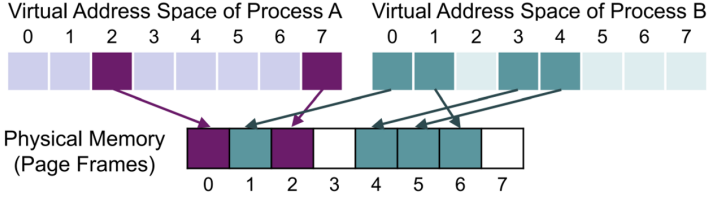
Q5 Description

Please draw the page tables for processes A and B:



Solution of Q5

A page table is the data structure used by a virtual memory system in a computer operating system to store the mapping between virtual addresses and physical addresses. (Refer Lecture 18, Page 13).



Page Table of Process A

Process A ID	Valid Bit	Frame Num.
0	0	-
1	0	-
2	1	0
3	0	-
4	0	-
5	0	-
6	0	-
7	1	2

Page Table of Process B

Process B ID	Valid Bit	Frame Num.
0	1	1
1	1	6
2	0	-
3	1	4
4	1	5
5	0	-
6	0	-
7	0	-

Overview

Q6 Description

Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a 2-issue RISC-V:

```
LOOP: lw   x31, 0(x20)     # x31 = some value
add  x31, x31, x21         # add scalar in x21
sw   x31, 0(x20)           # store result
addi x20, x20, -8          # decrement pointer
blt  x22, x20, LOOP        # branch if x22 < x20
```

1. Using the scheduled instruction to calculate IPC (instructions per clock cycle).
2. Suppose we have four registers (x28, x29, x30, x31), please design a solution to unroll the loop for better IPC.

Solution of Q6

The above code in C language style:

```
while(index_i < index_j)
{
    temp = a[index_j];
    temp = temp + some_scalar;
    a[index_j] = temp;
    index_j = index_j - 1;
}
```

Function: Add a scalar to a vector.

Solution of Q6

The scheduled instruction:

	ALU or branch	Data transfer	CC
LOOP:	nop	lw x31, 0(x20)	1
	add x20, x20, -8	nop	2
	add x31, x31, x21	nop	3
	blt x22, x20, LOOP	sw x31, 8(x20)	4

IPC $= 5/4 = 1.25$.

Solution of Q6

A possible solution:

	ALU or branch	Data transfer	CC
LOOP:	addi x20, x20, -32	lw x28, 0(x20)	1
	nop	lw x29, 8(x20)	2
	add x28, x28, x21	lw x30, 16(x20)	3
	add x29, x29, x21	lw x31, 24(x20)	4
	add x30, x30, x21	sw x28, 32(x20)	5
	add x31, x31, x21	sw x29, 8(x20)	6
	nop	sw x30, 16(x20)	7
	blt x22, x20, LOOP	sw x31, 24(x20)	8

IPC $= 14/8 = 1.75 > 1.25$.
Deal with four items at once.
You can change the order of the register x28, x29, x30, and x31.

Overview

Q7 Description

Assume we have a program where 10% of the execution time is purely sequential and that the rest of the execution time can be improved by parallelization. For the part of the code that can be parallelized, each core gives only 80% improvement. For instance, 5 cores give $5 \times 80\% = 4$ times improvement.

1. Create a speedup chart, showing speedup on the Y-axis and the number of cores on the X-axis. Show the graph for 25 to 200 cores, for instance by plotting with 25 cores interval.
2. What is the maximal speedup that can be achieved regardless how many cores we add?

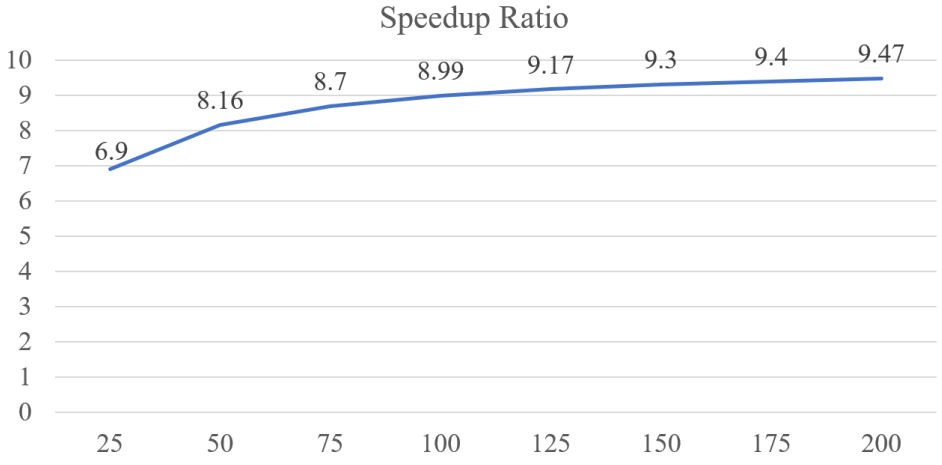
Solution of Q7

Apply Amdahl's law, the function of speedup S in terms of the number of core N can be:

$$S(N) = \frac{1}{\frac{1-10\%}{N \times 80\%} + 10\%}.$$

Core Num	25	50	75	100	125	150	175	200
Speedup	6.90	8.16	8.70	8.99	9.17	9.30	9.40	9.47

Solution of Q7



The maximal speedup that can be achieved is:

$$\lim_{N\to+\infty} S(N) = \lim_{N\to+\infty} \frac{1}{\frac{1-10\%}{N\times 80\%} + 10\%} = 10.$$

THANK YOU!