CENG3420 Homework 3

Due: Apr. 23, 2017

Solutions

Q1 (**10%**) Explain how page offset, page number, virtual address and physical address are associated to each other.

A1 Example Ans:

- Virtual Address = OS address length
- Physical Address = $\log_2$(RAM size) bits
- Offset = $\log_2$(page size) bits
- Virtual Page Number bits = Virtual Address - Offset
- Physical Page Number bits = Physical Address - Offset

Q2 (**15%**) Elaborate advantages and disadvantages of **LARGE** page size.

A2 Sample Answer:
Advantages:

1. Fewer page faults
2. Smaller page table
3. Fewer TLB misses

Disadvantages:

1. Page faults are expensive
2. Wasted space if pages are under-utilized

Q3 (**10%**) Here are two different I/O systems intended for use in transaction proceeding:

- System A can support 15,000 I/O operations per second and use the processor with MIPS rate of 50.
- System B can support 1,000 I/O operations per second and use the processor with MIPS rate of 500.

Assume that each transaction requires 5 I/O operations and each I/O operation requires 10,000 instructions. Ignoring response time, what is the maximum transactions per second for each system.

A3 Each transaction requires 10,000 $\times$ 5 = 50,000 instructions.

- For System A:
  CPU limit: 50M / 50K =1000 trans/second;
  I/O limit: 15,000 / 5=3000 trans/second;
  Therefore, max 1000 trans/second.
- For System B:
  CPU limit: 500M / 50K =10000 trans/second;
  I/O limit: 1,000 / 5=200 trans/second;
  Therefore, max 200 trans/second.

Q4 (**10%**) For the following code:

```
for (int i = 0; i < N; ++i) {
        sum[i] = 0;
        for (int j = 0; j < i; ++j) {
                sum[i] = (sum[i] + array[j]) % N;
        }
}
```

Clearly the code takes $\mathcal{O}(N^2)$ time. We would like to improve the actual running time. Which of these strategies would you recommend. Why? (2% for choice and 8% for reason)

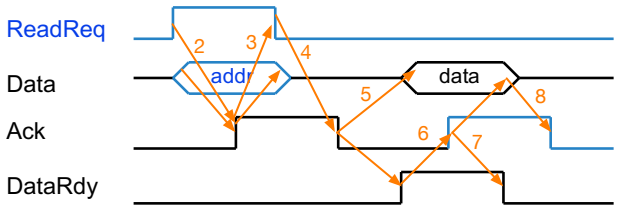
```
// Option1
for(int i = 0; i < N; ++i) {
        sum[i] = 0;
        parallel_for(int j = 0; j < i; ++j) {
                sum[i] = (sum[i] + array[j]) % N;
        }
}

// Option2
parallel_for(int i = 0; i < N; ++i) {
        sum[i] = 0;
        for (int j = 0; j < i; ++j) {
                sum[i] = (sum[i] + array[j]) % N;
        }
}
```

A4 Option 2, because Option 1 results in a *race condition*. If the *race condition* was not an issue, Option 2 would still be better because we would pay the overhead of forking and joining multiple threads only once, instead of each time within the outer loop (as in Option 1).

Q5 (**18%**) Considering a scenario that data is transferred from memory to I/O devices. Complete the following Asynchronous Bus Handshaking Protocol.

1. I/O device requests by raising `ReadReq` & putting addr on the data lines
2. Memory sees `ReadReq`, reads addr from data lines, and raises `Ack`



3. I/O device sees `Ack` and releases the `ReadReq` and data lines
4. Memory sees `ReadReq` go low and drops `Ack`
5. When memory ready, putting data on data lines & raises `DataRdy`
6. I/O device sees `DataRdy`, reads data from data lines & raises `Ack`
7. Memory sees `Ack`, releases data lines, and drops `DataRdy`
8. I/O device sees `DataRdy` go low and drops `Ack`

Q6 (**10%**) In the design of a multi-core processor, there are fixed on chip cache resources. We assume maximum of n cores can be designed with those resources. Let k be the real designed core number ($r = \frac{n}{k}$ is integer.) Define a speed up factor $s(r)$ as sequential performance gain by using the resources equivalent to r cores to form a single core, and obviously $s(1) = 1$. Given f the fraction of software that is parallelizable across multiple cores, prove the speed up of the multi-core processor in terms of f, r, n, and $s(r)$ is

$$S(f, r, n) = \frac{1}{\frac{1-f}{s(r)} + \frac{f \times r}{n \times s(r)}} \tag{1}$$

A6

$$\begin{aligned} S(f, r, n) &= s(r) \times \frac{1}{(1-f) + \frac{f}{k}} \\ &= s(r) \times \frac{1}{(1-f) + \frac{f \times r}{n}} \\ &= \frac{1}{\frac{1-f}{s(r)} + \frac{f \times r}{n \times s(r)}}. \end{aligned} \tag{2}$$

Q7 (**20%**) For the following loop code,

```
lp: lw      $t0, 0($s1)
    lw      $t1, 0($s2)
    addu    $t0, $t0, $t1
    sw      $t0, 0($s1)
    addi    $s1, $s1, -4
    addi    $s2, $s2, -4
    bne     $s1, $0, lp
```

1. (**4%**) Write down the 4 times unrolled code.

	ALU or Branch	Data Transfer	cc
			1
			2
			3
			4
			5
			6
			7
			8

2. (**16%**) Schedule the unrolled code and fill the table (**you are free to add more rows**).

A7 1. Trivial

2. A sample solution. As shown in the Table 1. Be careful of the data dependencies.

Table 1: A7

	ALU or Branch	Data Transfer	CC
lp	`addi $s1, $s1, -16`		1
	`addi $s2, $s2, -16`	`lw $t0, 16($s1)`	2
		`lw $t4, 16($s2)`	3
	`addu $t0, $t0, $t4`	`lw $t1, 12($s1)`	4
		`lw $t5, 12($s2)`	5
	`addu $t1, $t1, $t5`	`lw $t2, 8($s1)`	6
		`lw $t6, 8($s2)`	7
	`addu $t2, $t2, $t6`	`lw $t3, 4($s1)`	8
		`lw $t7, 4($s2)`	9
	`addu $t3, $t3, $t7`	`sw $t0, 16($s1)`	10
		`sw $t1, 12($s1)`	11
		`sw $t2, 8($s1)`	12
	`bne $s1, $0, lp`	`sw $t3, 4($s1)`	13

Q8 (**7%**) Name 3 cache enhancement techniques and elaborate them.

A8 Details can be found at last page of slides L11-VM

1. Write buffer
2. Prefetch
3. Load-through approach