Amdahl's law, virtualization and cache exercises

EXAM2014 Given a single core computer used for running a financial application. This application uses 90% of the execution time for computational intensive operations. The remaining 10% is spent waiting for Input/Output hard disk accesses. The computational intensive part is divided in 75% for floating point operations and 25% for other instructions. The average CPI of a floating point operation is 12. The rest of the instructions have an average CPI of 4.
We have two different alternative platforms for running this application:

- **Alternative A**: A single core processor with a clock frequency 50% higher than the original computer. For this new system the floating point instructions take 10% more clock cycles per instruction and the rest of the instructions take 25% more cycles than the original ones. The Input/Output time is the same as the original platform.
- **Alternative B**: A four core processor with a frequency 50% smaller than the original computer. The floating point instructions require 20% less clock cycles than the original ones. The rest of the instructions take the same amount of clock cycles as the original ones. The Input/Output time is the same as the original platform.

Answer the following questions:

1. What would be the overall speedup of Alternative A?
2. What would be the overall speedup of Alternative B? Assume that the computational intensive part is completely parallelizable and that the Input/Output part is sequential.

2.20 Virtual Machines (VMs) have the potential for adding many beneficial capabilities to computer systems, such as improved total cost of ownership (TCO) or availability. Could VMs be used to provide the following capabilities? If so, how could they facilitate this?

a) Test applications in production environments using development machines?
b) Quick redeployment of applications in case of disaster or failure?
c) Higher performance in I/O-intensive applications?
d) Fault isolation between different applications, resulting in higher availability for services?

2.21 Virtual machines can lose performance from a number of events, such as the execution of privileged instructions, TLB misses, traps, and I/O. These events are usually handled in system code. Thus, one way of estimating the slowdown when running under a VM is the percentage of application execution time in system versus user mode. For example, an application spending 10% of its execution in system mode might slow down by 60% when running on a VM. Figure below lists the early performance of various system calls under native execution, pure virtualization, and paravirtualization for LMbench using Xen on an Itanium system with times measured in microseconds (courtesy of Matthew Chapman of the University of New South Wales).

a) What types of programs would be expected to have smaller slowdowns when running under VMs?
b) What is the median slowdown of the system calls in the table above under pure virtualization and paravirtualization?
c) Which functions in the table above have the largest slowdowns? What do you think the cause of this could be?

Benchmark	Native	Pure virtualization	Paravirtualization
Null call	0.04	0.96	0.50
Null I/O	0.27	6.32	2.91
STAT	1.10	10.69	4.14
OPEN/CLOSE	1.99	20.43	7.71
Install sighandler	0.33	7.34	2.89
Handle signal	1.69	19.26	2.36
FORK	56.00	513.00	164.00
EXEC	316	2084	578
FORK+EXEC sh	1451.00	7790.00	2360.00

2.1 The transpose of a matrix interchanges its rows and columns; this is illustrated below:

Here is a simple C loop to show the transpose:

```
for (i = 1; i <=4; it+) {
  for (j = 1; j <=4; j++) {
    output [j] [i] = input [i] [j] ;
  }
}
```

A ll A21 A31 A41
A12 A22 A32 A42
A13 A23 A33 A43
A14 A24 A34 A44

A ll A12 A13 A14
A21 A22 A23 A24
A31 A32 A33 A34
A41 A42 A43 A44

Assume that both the input and output matrices are stored in the row major order *(row major order* means that the row index changes fastest). Assume that you are executing a **256 x 256 double-precision** transpose on a processor with a **16 KB fully associative** (don't worry about cache conflicts) least recently used (LRU) replacement Ll data cache with **64 byte blocks**. Assume that the L1 cache misses or prefetches require 16 cycles and always hit in the L2 cache, and that the L2 cache can process a request every two processor cycles. Assume that each iteration of the inner loop above requires four cycles if the data are present in the Ll cache.

For the simple implementation given above, this execution order would be nonideal for the input matrix; however, applying a loop interchange optimization would create a nonideal order for the output matrix. Because loop interchange is not sufficient to improve its performance, it must be blocked instead.

a) What should be the minimum size of the cache to take advantage of blocked execution?
b) How do the relative number of misses in the blocked and unblocked versions compare in the minimum sized cache above?
c) Write code to perform a transpose with a block size parameter B which uses B x B blocks.
d) What is the minimum associativity required of the L1 cache for consistent performance independent of both arrays' position in memory?

2.2 Assume you are designing a hardware prefetcher for the unblocked matrix transposition code above. The simplest type of hardware prefetcher only prefetches sequential cache blocks after a miss. More complicated "non-unit stride" hardware prefetchers can analyze a miss reference stream and detect and prefetch non-unit strides. In contrast, software prefetching can determine nonunit strides as easily as it can determine unit strides. Assume prefetches write directly into the cache and that there is no "pollution" (overwriting data that must be used before the data that are prefetched). For best performance given a nonunit stride prefetcher, in the steady state of the inner loop how many prefetches must be outstanding at a given time?