

Question 1: Arrays

Sean Marshallsay
Exam Number: Y0071170

- a Each array takes up 4 kB of memory and the first element of B directly follows the last element of A in main memory. In a direct-mapped cache scheme byte i in main memory is mapped to byte $i \% n$ in the cache where n is the number of bytes in the cache. This means that (in this particular case) the first byte of the cache will be mapped to the first byte of A and also the first byte of B. Therefore the part of the arrays being accessed can not both be in the cache at the same time and the cache must be refreshed each time A or B are indexed on the right hand side of the equation.

One way to overcome the problem using hardware is to increase the size of the cache. This would mean that A and B no longer map to the same addresses in the cache. Alternatively a different caching scheme could be used such as fully-associative, where any cache line can be mapped to any main memory address, or set-associative, where two direct-mapped caches are used allowing both A and B to be in the cache at the same time.

One way to overcome this problem in the software would be to split the arrays in half, storing the first half of A then the first half of B then the second half of A then the second half of B. This would mean that all of A maps to the first half of the cache while all of B maps to the second half, since the algorithm only accesses one part of each array at a time this would not be a major problem. If the two arrays do not need to be stored contiguously then inserting some memory (which is not a multiple of 4 kB in size) between the arrays will mean they no longer map to the same addresses. If the cache refreshes on a line by line basis (rather than all at once) then you could reverse the order of B in memory and change the algorithm to the following:

```
for (i = 0; i < 1024; i++) {  
    A[i] = A[i] + B[1023 - i];  
}
```

though this would introduce an extra calculation and is a more complex

for the programmer.

- b
 - The L1 cache must have been queried for every memory reference: 1000 cycles.
 - 40 of those times there was an L1 cache miss so the L2 cache had to be queried: 400 cycles.
 - 20 of those times there was an L2 cache miss and the main memory had to be queried: 2000 cycles.
 - This is a total of 3400 cycles for 1000 memory references and therefore the average access time for the program was 3.4 cycles per reference.