**Question 1**

A thread switch is more efficient than a process switch because:

* Thread switching involves switching from one thread to another in the same process. This operation is generally efficient because it doesn’t require hundreds of cycles.
* Process switching involves switching the memory address space. This is an expensive operation, because the processor must copy the state of each thread. Doing so is quite taxing and requires a lot of resources and cycles. Copying each thread involves, but is not limited to, saving and loading registers and memory maps, updating various tables/lists, copying data such as the program counter, etc. Furthermore, there are indirect costs such as cold cache and cache misses.

Therefore, a thread switch is more efficient than a process switch because the procedure of switching in and out of memory space (kernel + user) is costly.

We Have:

* 20 scalar variables 🡪 20 variables to add up
* 20 x 20 matrix 🡪 400 variables to add up

According to Amdahl’s Law:

Execution Execution time affected by improvement

time after = ------------------------------------------------------- + Execution time unaffected

improvement Amount of improvement

With 1 processor, the speed is:

(20 scalar variables) + (20 x 20 matrix) = 20t + 400t = 420t

With 100 processors, the speed is:

(400/100)t + 20t = 4t + 20t = 24t

Speed-up = 420t / 24t = 17.5

One processor’s load is twice than all the rest.

Previously, the work load was uniformly distributed and each processor got an equal share. Now, one processor does 2x as much work as the other processors (NOT combined).

Previous load: Each processor had a load of (100/100) = 1% (Uniformly distributed)

Previously, each processor did 1% of the workload

Now, the New Load Is:

1 processor = 2% 🡪 (B/C 1% x 2 = 2%)

99 processors = 98% 🡪 (100% - 2% = 98%)

If one processor has 2% of the load, then it must do: 2% x 400 additions.

2% x 400 = 8. So one processor is doing 8 matrix additions, while the rest 99 processors share 392 additions.

1 processor = 8 matrix additions 🡪 8/1 (matrix additions per processor)

99 processors = 392 additions 🡪 392/99 (matrix additions per processor)

So, according to Amdahl’s Law:

Execution time after improvement = Max (392t/99, 8t/1) + 20t = Max (3.959, 8) + 20t = 28t

(We take the max because whatever finishes slower is the “bottleneck”. While the 99 processors,

Speed-up = 420t / 28t = 15

**Question 2**

Temporal Locality

* size: It gets read on each iteration of two loops
* outer: Gets incremented by one at the end of the first for loop. It’s also used to reference elements in array.
* inner: Gets incremented by one at the end of the second for loop. It’s also used to reference the elements in an array
* lMat[outer][1]: In the second loop, it gets accessed over and over again, and is the same value. Since the same value is being called over and over again, this is temporal.

Spatial Locality

* rMat[outer][inner]
* rMat[inner][outer]
* lMat[outer][1]

^ These exhibit spatial locality because they are arrays, and each element is accessed with each increasing iteration. Since each element is accessed sooner or later, the compiler stores it in the cache. Therefore, this is spatial locality because the elements of the array are nearby each other in memory.

Note: You can say that all variables exhibit spatial locality because of how the compiler optimizes the memory and cache. Since this is a loop, the compiler will place the variables very close to each other in memory. However, you can ignore this b/c the question doesn’t ask for it. But Ninos said you can include it.

**Question 3**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Binary | Tag | Index | Result |
| 3 | 0000 0011 | 0000 | 001**1** | Miss |
| 180 | 1011 0100 | 1011 | 010**0** | Miss |
| 43 | 0010 1011 | 0010 | 101**1** | Miss |
| 2 | 0000 0010 | 0000 | 001**0** | Miss |
| 191 | 1011 1111 | 1011 | 111**1** | Miss |
| 88 | 0101 1000 | 0101 | 100**0** | Miss |
| 190 | 1011 1110 | 1011 | 111**0** | Miss |
| 14 | 0000 1110 | 0000 | 111**0** | Miss |
| 181 | 1011 0101 | 1011 | 010**1** | Miss |
| 44 | 0010 1100 | 0010 | 110**0** | Miss |
| 186 | 1011 1010 | 1011 | 101**0** | Miss |
| 253 | 1111 1101 | 1111 | 110**1** | Miss |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Binary | Tag | Index | Result |
| 3 | 0000 0011 | 0000 | 00**1** | Miss |
| 180 | 1011 0100 | 1011 | 01**0** | Miss |
| 43 | 0010 1011 | 0010 | 10**1** | Miss |
| 2 | 0000 0010 | 0000 | 00**1** | Hit |
| 191 | 1011 1111 | 1011 | 11**1** | Miss |
| 88 | 0101 1000 | 0101 | 10**0** | Miss |
| 190 | 1011 1110 | 1011 | 11**1** | Hit |
| 14 | 0000 1110 | 0000 | 11**1** | Miss |
| 181 | 1011 0101 | 1011 | 01**0** | Hit |
| 44 | 0010 1100 | 0010 | 11**0** | Miss |
| 186 | 1011 1010 | 1011 | 10**1** | Miss |
| 253 | 1111 1101 | 1111 | 11**0** | Miss |

C1 = 1 Word Block

C2 = 2 Word Blocks

C3 = 4 Word Blocks

From (3A), the miss rate for C1 is:

Miss(C1) = 12/12

From (3B), the miss rate for C2 is:

Miss(C2) = 9/12

C3 (is the same as C2):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | Binary | Tag | Index | Result |
| 3 | 0000 0011 | 0000 | 00**1** | Miss |
| 180 | 1011 010 0 | 1011 | 01**0** | Miss |
| 43 | 0010 1011 | 0010 | 10**1** | Miss |
| 2 | 0000 0010 | 0000 | 00**1** | Hit |
| 191 | 1011 1111 | 1011 | 11**1** | Miss |
| 88 | 0101 1000 | 0101 | 10**0** | Miss |
| 190 | 1011 1110 | 1011 | 11**1** | Hit |
| 14 | 0000 1110 | 0000 | 11**1** | Miss |
| 181 | 1011 0101 | 1011 | 01**0** | Hit |
| 44 | 0010 1100 | 0010 | 11**0** | Miss |
| 186 | 1011 1010 | 1011 | 10**1** | Miss |
| 253 | 1111 1101 | 1111 | 11**0** | Miss |

|  |  |  |  |
| --- | --- | --- | --- |
|  | C1 | C2 | C3 |
| # Of Hits | 12 | 12 | 12 |
| Access Time | 2 | 3 | 5 |
| # Of Misses | 12 | 9 | 9 |
| Stall Time | 25 | 25 | 25 |
| Total Cycles | 324 | 261 | 285 |

Total Cycles = ([# Of Hits] \* [Access Time]) + ([# Of Misses] \* [Stall Time])

C1 [Total Cycles] = (12 \* 2) + (12\*25) = 324

C2 [Total Cycles] = (12 \* 3) + (9\*25) = 261

C3 [Total Cycles] = (12 \* 5) + (9\*25) = 285

The best cache design is C2 because it has the lowest number of total cycles. See detailed calculation above for more information.

**Question 4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Word | Block | Set # | H or M | Tag | Index |
| 3 | 1 | 1 | M | 000000 | 0**1** |
| 180 | 90 | 2 | M | 010110 | 1**0** |
| 43 | 21 | 1 | M | 000101 | 0**1** |
| 2 | 1 | 1 | H | 000000 | 0**1** |
| 191 | 95 | 3 | M | 010111 | 1**1** |
| 88 | 44 | 0 | M | 001011 | 0**0** |
| 190 | 95 | 3 | H | 010111 | 1**1** |
| 14 | 7 | 3 | M | 000001 | 1**1** |
| 181 | 90 | 2 | H | 010110 | 1**0** |
| 44 | 22 | 2 | M | 000101 | 1**0** |
| 186 | 93 | 1 | M | 010111 | 0**1** |
| 253 | 126 | 2 | M | 011111 | 1**0** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Set | LRU | Block 1 [T0] | Block 2 [T1] | Block 3 [T2] |
| 0 | None |  |  |  |
| 1 | Tag 0 | 0000 | 0000 |  |
| 2 | Tag 0 | 1011 | 1011 |  |
| 3 | None |  |  |  |
| 4 | Tag 0 | 0101 |  |  |
| 5 | Tag 1 | 0010 | 1011 |  |
| 6 | Tag 1 | 0010 | 1111 |  |
| 7 | Tag 1 | 1011 | 0000 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Address | Tag | Line | H / M |
| 3 | 00000011 | 0 | M |
| 180 | 10110100 | 1 | M |
| 43 | 00101011 | 2 | M |
| 2 | 00000010 | 3 | M |
| 191 | 10111111 | 4 | M |
| 88 | 01011000 | 5 | M |
| 190 | 10111110 | 6 | M |
| 14 | 00001110 | 7 | M |
| 181 | 10110101 | 0 | M |
| 44 | 00101100 | 1 | M |
| 186 | 10111010 | 2 | M |
| 253 | 11111101 | 3 | M |

|  |  |  |
| --- | --- | --- |
| Set | LRU | Block |
| 0 | None |  |
| 1 | Tag 0 | 0000 |
| 2 | Tag 0 | 1011 |
| 3 | None |  |
| 4 | Tag 0 | 0101 |
| 5 | Tag 1 | 1011 |
| 6 | Tag 1 | 1111 |
| 7 | Tag 1 | 0000 |

For MRU the hit rate is 2/12, So the miss rate is 10/12.

For LRU the hit rate is 2/12, so the miss rate is 10/12.

Therefore, the efficacy of both replacement methods is the same. LRU replacement is similar to MRU in terms of miss rate for cache.