**1. (36 points)** Consider the following three processors (X, Y, and Z) that are all of varying areas. Assume that the single-thread performance of a core increases with the square root of its area.

Processor X: Core Area = A
Processor Y: Core Area = 4A
Processor Z: Core Area = 16A

**(a)** You are given a workload where S fraction of the work is serial and (1 – S) fraction of the work is infinitely parallelizable. If executed on a die composed of 16 Processor X's, what value of S would give a speedup of 4 over the performance of the workload on just Processor X?

$$S + \frac{1-S}{16} = \frac{1}{4}$$

$$1 + 15S = 4$$

$$15S = 3$$

$$S = 20\%$$

**(b)** Given a homogeneous die of area 16A, which of the three processors (X, Y, or Z) would you use on your die to achieve maximal speedup? What is that speedup over just a single Processor X? Assume the same workload as in part (a).

X: $S + \frac{1-S}{16} = \frac{1}{16} + \frac{15}{16}S$ if $S = 20\%$ speedup = 4

Y: $\frac{S}{2} + \frac{1-S}{8} = \frac{1}{8} + \frac{3}{8}S$ if $S = 20\%$ speedup = 5 ✓

Z: $\frac{S}{4} + \frac{1-S}{4} = \frac{1}{4}$ Speedup = 4

**(c)** Now you are given a heterogeneous processor of area 16A to run the above workload. The die consists of 1 Processor Y and 12 Processor X's. When running the workload, all sequential parts of the program will be run on the larger core while all parallel parts of the program run exclusively on the smaller cores. What is the overall speedup achieved over a single Processor X?

$$\frac{S}{2} + \frac{1-S}{12} = \frac{1}{12} + \frac{5S}{12}$$

if $S = 20\%$

$$\text{Speedup} = \frac{1}{\frac{1}{12} + \frac{5 \times 0.2}{12}} = \frac{6}{1} = 6.$$

**(d)** One programmer optimizes the given workload so that it has 4% of its work in serial sections and 96% of its work in parallel sections. Which configuration would you use to run the workload if given the choices between the processors from part (a), part (b), and part (c)?

(a) $\frac{4\%}{1} + \frac{96\%}{16} = 4\% + 6\% = 10\%$ speedup = 10. ✓

(b) Y: $\frac{4\%}{2} + \frac{96\%}{8} = 2\% + 12\% = 14\%$ speedup = 7.14

Z: $\frac{4\%}{4} + \frac{96\%}{4} = 25\%$ speedup = 4.

(c) $\frac{4\%}{2} + \frac{96\%}{12} = 2\% + 8\% = 10\%$ speedup = 10. ✓

**(e)** What value of S would warrant the use of Processor Z over the configuration in part (c)?

Z: $S + \frac{1-S}{4} = \frac{1}{4}$ speedup = 4

C: $\frac{1}{12} + \frac{5}{12}S > \frac{1}{4}$

$\frac{5}{12}S > \frac{1}{6}$

$5S > 2$

$S > 40\%$

**(f)** Typically, for a realistic workload, the parallel fraction is not infinitely parallelizable. What are the three fundamental reasons why?

i) Sychronization,

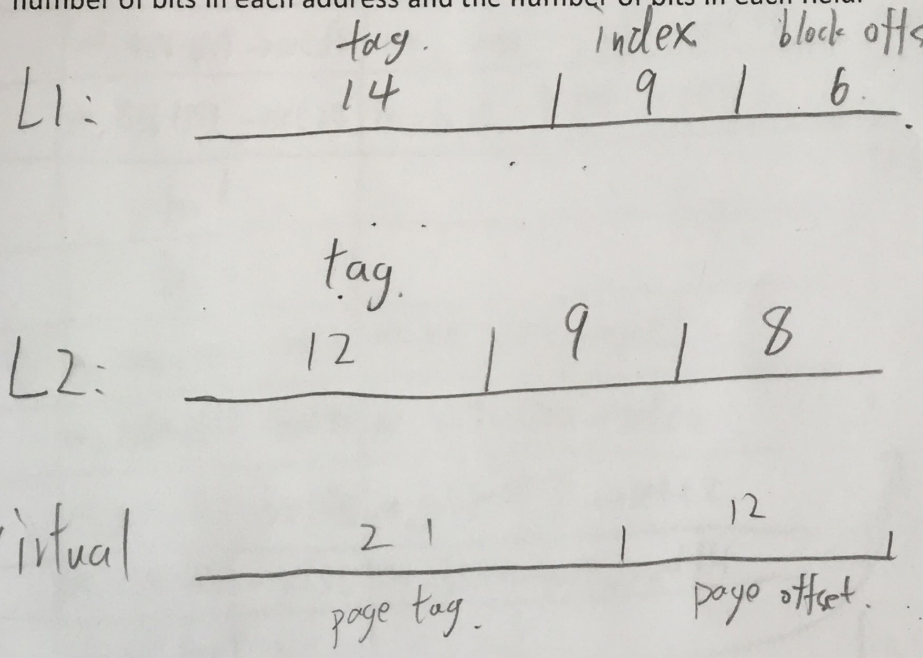
ii) number of codes are not infinite.

iii) limit of hardware.

**2. (36 points)**

Consider the complete memory hierarchy. You have a paged memory system. The processor has 512 Mbytes of memory and an 8 GB virtual address space with 4 Kbyte pages. The L1 cache is a 64 KB, 2-way set associative with 64 byte lines. The L2 cache is a 1 MB, 8-way set associative with 256 byte lines. Address translation is supported by a 4 entry TLB. Each page table entry is 32 bits.

**(a)** Show the breakdown of the fields of virtual address, physical address, the physical address as interpreted by the L1 cache, and the physical address as interpreted by the L2 cache. Clearly mark the number of bits in each address and the number of bits in each field.




**(b)** What is the number of L2 cache lines per page? What is the number of entries in the page table? What is the page table size in pages?

① 4 kB/256 byte. = 16.

② 512 MB/4kB = 128 × 1024 entry.

③ (1024 × 128.) × 32 bits. = 1024 × 128 × 4 Bytes

= 512 KB.

512 KB / 4KB = 128 pages.

**(c)** If a program has a block access pattern (in decimal) :
0, 512, 4096, 1024, 1536, 0, 512, 2048, 2560, 3072, 3584, 4096, 1024, 3072, 4096, 3584, 2048,
indicate, for each access, whether it is an L1 hit/miss or L2 hit/miss (if it is an L1 miss).

| | L1 | L2 |
|---|---|---|
| 0 | m, 0-63 → set 0. | m, 0-255 → set 0. |
| 512 | m, 512-575 → set 8 | m, 512-767 → set 2 |
| 4096 | m, 4096-4159 → set 64 | m, 4096-4351 → set 16. |
| 1536 | m, 1536-1599 → set 24 | m, 1536 - 1791 → set 6. |
| 0 | hit. | |
| 512. | hit. | |
| 2048. | m, 2048-2111 → set 32 | m, 2048-2303 → set 8. |
| 2560 | m, 2560-2623 → set 40 | m, 2560-2815 → set 10. |
| 3072 | m, 3072-3135 → set 48 | m, 3072-3327 → set 12. |
| 3584 | m, 3584-3647 → set 56 | m, 3584-3639 → set 14 |
| 4096. | hit. | |

| | L1 | 2 2 |
|---|---|---|
| 1024 | m, 1024-1087 → set 16 | m, 1029-1279 → |
| 3072 | hit | |
| 4096 | hit | |
| 3584 | hit. | |
| 2048 | hit. | |

**(d)** Is it possible for this system to address the L1 cache concurrently with the address translation provided by the TLB? Justify your answer using the address breakdowns shown in part (a). What would be the benefit of doing so?

64 bytes block

No, because block offset need 6 bits, index needs 9 bits (512 sets). totally 15 bits.

So, We need last 15 bits for addressing L1 cache. but page size is 4KB(12 bits), we can only get 12 bits of ical address (PA).

If we can do so, we can access L1 cache without com the PA. (if it is a L1 hit).

**3. (28 points)**

**(a)** Define/describe the principle of locality. Include a description of the different types of locality that occur in a computer system and different ways in which a designer can take advantage of locality in order to make a computer faster.

Temporal: a program reference same memory location many times. with in a small time.

Spatial: a program tend to reference a cluster of memory locations at a time.

**(b)** Explain the difference between data reuse and data locality. Given an example if necessary.

ta reuse: some data will be used for many times.

cality: one program will access memory in the nearby area in a sho time peorid.

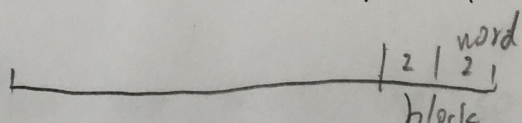
**(c)** Consider a cache with 256 bytes in a machine that employs 32-bit addresses. The word size is 4 bytes and the block size is 16 bytes. Show the values in the cache and tag bits after each of the following memory access operations (1 through 9), for the following two cache organizations:

(i) direct mapped — 256/16 = 16 sets.
(ii) two-way associative — 256/16/2 = 8 sets.

Also indicate whether the access was a hit or a miss, justifying your answer.

All addresses below are hexadecimal. Use the Least Recently Used (LRU) replacement algorithm as needed.

1. Read 0010
2. Read 001C
3. Read 0018
4. Write 0010
5. Read 0484
6. Read 051C
7. Read 001C
8. Read 0210
9. Read 051C




4bit set
DM.

0010 X set 1
001C V set 1
0018 V set 1
0010 V set 1
0484 X set 8
051C X set 1
001C X set 1
0210 X set 1
051C X set 1

3bit set
2-way.

0010 X set 1.
001C V set 1
0018 V set 1
0010 V set 1
0484 X Set 0.
051C X set 1
001C V set 1
0210 X set 1
051C X set 1

**d)** Many x-way set-associative caches implement true least-recently-used (LRU) replacement within each cache set. To do this, many caches store the data blocks of a set in fixed locations and store replacement-state bits, R<r-1> ... R<0> to provide replacement information within each set (e.g., which block is LRU). On each access, R<r-1>...R<0> are read, and updated (unless the access was to the most-recently-used block). On a miss, R must be interpreted to determine which block should be replaced. What is the minimum number of replacement-state bits needed in each set of an x-way set-associative cache? Why? Explain, in detail, the logic needed to update R<r-1>...R<0> on a cache hit.

For x blocks in one set of the x-way set-associative cache, each block we need $\log_2(x)$ bits for storying the order of the block.

Since there are x blocks in the set, totally $x \cdot \log_2(x$

Each time when we access the set, update the $\log_2(x)$ bi

correspond to each block and maintain the order.