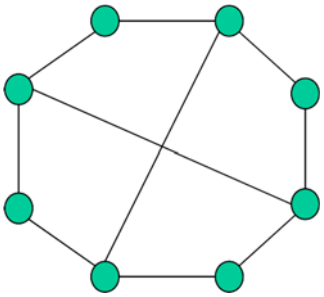
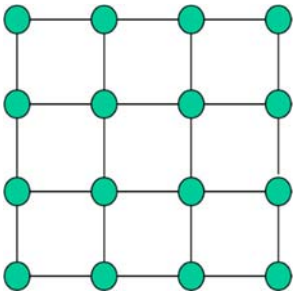
CS/CoE 1541 – Example of questions for Exam 3

Question 1:
Let the distance between two nodes be defined as the number of links between the nodes. What is the diameter and what is the bisection width of each of the networks shown below?



Diameter = 3

Bisection width = 4



Diameter = 6

Bisection width = 4

Question 2:

(a) Consider the following skeleton for a CUDA program

```
_global_void my_kernel (int *a, int * b)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x ;   /*note that blockDim.x = 2 */
  a[idx+1] = threadIdx.x  ;
  b[idx] = blockIdx.x  ;
}
void main(){
  …
  my_kernel<<< 4,2 >>> (a , b)                /* four blocks, each containing 2 threads */
  …
}
```

Assuming that arrays a[] and b[] are allocated in the global memory and are initialized to -1, what will be the values of their elements after my_kernel finishes execution?

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
-1	0	1	0	1	0	1	0	1

b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]
0	0	1	1	2	2	3	3	-1

(b) For each of the following statements, circle the correct answer

- The _syncthreads() CUDA barrier synchronizes all the treads of a:

  (i) kernel (ii) thread block (iii) warp

- In a CUDA kernel, it is more efficient to use a block size of

  (i) 64 threads (ii) 16 threads (iii) 8 threads

- In a GPU, the shared memory is shared among all the threads of a

  (i) kernel (ii) thread block (iii) warp

- cudaMalloc() is used to dynamically allocate space in

  (i) shared memory (ii) global memory (iii) CPU memory

- All the threads in a thread block execute on the same

  (i) SM - streaming multiprocessor (ii) SP - streaming processor (iii) device

- The number of registers in an SM determines the maximum number of ____ allowed on the SM

  (i) warps (ii) thread blocks (iii) threads

Question 3: Consider the following Pthread program which computes the sum of N numbers using P threads. The sum will be computed in "sum[0]":

```
#define P  8        /* P is a power of 2 */
#define N 1024   /* N is a power of 2 */
void *compute_sum ( void *);

struct arg_to_thread {int id} ;
float A[N];                                  /* assume that A[] is initialized to some values */
float sum[P] ;

main (int argc, char *argv[] )    {
        int i  ;
        pthread_t p_threads[P];
        pthread_attr_t  attr;
        struct arg_to_thread my_arg[P] ;
        pthread_attr_init (&attr);
        for (i=0; i < P ; i++     ){
                my_arg[i].id = i ;
                pthread_create (&p_threads[i], &attr, compute_sum, (void*) &my_arg[i]);
        }
        for (i=0; i< num_threads; i++)
                pthread_join (p_threads[i], NULL);
}

void  *compute_sum (void *arg) {
        struct arg_to_thread *local_arg ;
        int i, half, idx ;
        local_arg = arg;
        idx =  (*local_arg).id;                                  /* idx is the id given to the thread */

        for (i =  N/P * idx     ; i <  N/P * (idx +1) ; i++)       /* compute a partial sum */     /*line 1*/

                sum[    idx     ] = sum[     idx     ] + A[i] ;                                    /*line 2*/

        half = P/2 ;                                                                                /*line 3*/

        for (i =      0     ;   half >= 1         ; i++) {       /* compute the global sum */        /*line 4*/

                if(          idx  < half         ) {                                               /*line 5*/

                        sum[    idx      ] = sum[    idx    ] + sum[      idx + half       ]  ;     /*line 6*/

                }                                                                                   /*line 7*/

                half =      half / 2        ;                                                       /*line 8*/
        }
}
```

(a) Ignoring the need for synchronization, **complete the lines labeled /*line 1*/ to /*line 7*/ in the function "compute_sum"** such that the sum of the 1024 numbers is computed in sum[0]. The sum is computed by forking 8 threads, each of which computes the sum of 128 numbers (in parallel). The 8 partial sums are then added together using a tree reduction algorithm.

(b) Indicate after which line(s) you should add barrier synchronizations and explain why are these barriers needed?

Between lines 4 and 5

To make all processors wait until other processors finish the current step

(c) What is the speedup and efficiency obtained from the parallel execution when N = 1024 and P = 8?

Serial execution time = 1023 steps

Execution time using 8 processors = 128+3 = 131 steps

Speedup = 1023/131 = 7.8

Efficiency = 7.8 / 8 = 0.976

(d) Assuming that you can use as many processors as you want, what is the maximum speedup that can be obtained to solve the problem for N = 1024?

Can finish in 10 steps is we use 512 processors

Maximum speedup = 1023/10 = 102.3

(e) Amdahl law indicates that if α is the fraction of the task that has to execute serially, then the maximum speedup that can be obtained is $1/\alpha$. Why can't we apply Amdahl's law to obtain the answer to part (d)?.

The serial part is not purely serial – it uses a binary tree (not serial and not completely parallel).

Question 4: Consider a superscalar architecture with two pipelined units, one for load/store and one for ALU instructions. The following two tables indicate the order of execution of two threads, A and B, and the latencies mandated by dependences between instructions. For example, A2 and A3 should execute after A1 and there should be at least four cycles between the execution of A3 and A5 and at least three cycles between A4 and A5. In other words, the tables indicate the schedule if the instructions of each of the threads are executed with no multithreading.

Note that an instruction that executes on one pipeline (for example A1 on the load/store pipeline) cannot execute on the other pipeline.

time	Load/store pipeline	ALU pipeline
t	A1	
t+1	A3	A2
t+2		A4
t+3		
t+4		
t+5		
t+6	A5	
t+7	A6	A7

time	Load/store pipeline	ALU pipeline
t	B1	B2
t+1		B3
t+2	B4	B5
t+3	B6	
t+4		B7
t+5	B8	
t+6		
t+7		

Show the execution schedule for the two threads on the two pipelines assuming:

(a) Coarse grain multithreading

time	Load/store pipeline	ALU pipeline
t	A1	
t+1	A3	A2
t+2		A4
t+3		
t+4	B1	B2
t+5		B3
t+6	B4	B5
t+7	B6	
t+8		B7
t+9	B8	
t+10		
t+11	A5	
t+12	A6	A7
t+13		
t+14		

(b) fine grain multithreading

time	Load/store pipeline	ALU pipeline
t	A1	
t+1	B1	B2
t+2	A3	A2
t+3		B3
t+4		A4
t+5	B4	B5
t+6		
t+7	B6	
t+8	A5	
t+9		B7
t+10	A6	A7
t+11	B8	
t+12		
t+13		
t+14		

(c) Simultaneous multithreading (with priority given to thread A)

time	Load/store pipeline	ALU pipeline
t	A1	
t+1	A3	A2
t+2	B1	A4
t+3		B2
t+4		B3
t+5	B4	B5
t+6	A5	
t+7	A6	A7
t+8	B6	
t+9		B7
t+10	B8	
t+11		
t+12		
t+13		
t+14		