## UCSB CS140 Exercise #2

Mark your best choice for each problem in the answer sheet and submit it to gradescope.com.

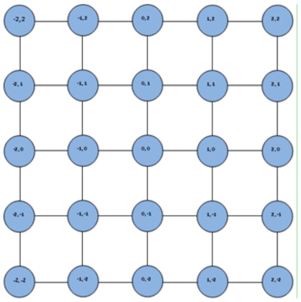
**Question 1.** Memory allocated for the executable of a C program is divided into 4 areas: text, stack, heap, and data/BSS (for initialized/un-initialized global or static variables).

```
int  *r;   /* Line 1*/
void f(int *r){   /* Line 2*/
   int z=5;      /*  Line 3*/
   r[z]=4;       /*  Line 4*/
}
void main(void){
   r=malloc(10*sizeof(int));
   f(r);
   free(r);
}
```

Mark the space location for variable r in Line 2, and array element r[z] in Line 4.

(a) stack, heap (b) data/BSS, heap (c) stack, stack (d) data/BSS, stack (e) none of them is correct.

**Question 2.** The bisection width is the minimum number of links that must be removed to partition the network into two equal halves (or two sets that differ by at most one node in size). The bisection bandwidth is the minimum sum of bandwidths of these links among all possible partitionings. What is the bisection bandwidth of the following 5x5 mesh, assuming the bandwidth of every link is 1Gb per second? A planar 2D mesh is just like a 2D toroidal mesh, except that it doesn't have the “wraparound” links.

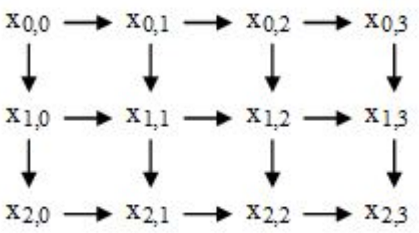


(a) 5Gb/s (b) 6Gb/s (c) 7Gb/s (d) 8Gb/s (e) None of them is correct

**Question 3.** Assume the cache line (cache block) in a multiprocessor shared memory system is of size 64 bytes. Choose a statement from (a) to (d) below that is false or choose (e)

(a) When a processor executes a store instruction to write a word of 4 bytes to memory, this triggers the writing of a data block of 64 bytes from a local processor cache to memory.
(b) When a processor executes a load instruction of 4 bytes to read such a word from memory, this triggers the reading of a data block from memory to a local processor cache when there is a cache miss.
(c) Writing a data block to a processor cache has to invalidate copies of this block in other processor caches even these processors do not use this data block any more.
(d) False sharing can occur when multiple processors write data to different but consecutive data addresses in memory.
(e) All of them are correct.

**Questions 4-5**. A task graph with 12 tasks is shown below as follows, each task named $x_{i,j}$ costs 1 time unit to execute.



**Q4**: What is the degree of parallelism in this graph?

(a) 1 (b) 2 (c) 3 (d) 4 (e) None of them is correct

**Q5:** What is the maximum speedup achievable in executing the above task graph in a parallel architecture?

(a) 1 (b) 2 (c) 3 (d) 4 (e) None of them is correct

**Question 6**.

(1) Assume a sequential program takes 1,000 seconds, what is the maximum speedup possible according to Amdahl's Law for a program that is 10% inherently serial and 90% fully parallelizable if you can only use 3 processors?

(2) Following the assumption of (1), if the parallelized program needs to spend an additional 100 seconds to exchange data and synchronize processors, what is the maximum speedup possible? Notice the additional 100 seconds are charged to every processor for data exchange and synchronization in this case.

Select one of the following 4 choices to answer (1) and (2):

(a) 3, 2 (b) 3, 2.5 (c) 2.5, 2 (d) 2.5, 2.5 (e) None of them is correct

**Question 7:** Choose a statement from (a) to (d) below that is false or choose (e)

(a) MPMD (Multiple program multiple data) is flexible as it allows different processing units run different code.
(b) In running SPMD (Single Program Multiple Data) code, each processing unit executes the same binary, but operates on different data based on processing unit ID.
(c) SPMD simplifies parallel programming because we write the same program for a different number of processing units.
(d) At each time step of SPMD code execution, all processing units execute the exactly same machine instruction.
(e) All of them are true.

**Questions 8-10.** The following sequential code segment in C calls function compute() *n* times,

```
for(int i=0;  i<n; i++){
  compute(i);
}
```

To parallelize the above code, the following incomplete SPMD code with block mapping divides *n* iterations to *p* processors evenly and each processor handles about *n/p* iterations started from 0. Select a proper formula for variables first and last, controlling what is executed at each node. The numbering of processor nodes starts from 0. Function floor(x) returns the largest integer smaller than or equal to x and ceil(x) returns the smallest integer bigger than or equal to x.

```
double n_over_p= (double) n/ ((double) noproc());
int me=mynode();
int first =
int last =
for( i=first;  i<=last && i<n; i++){
  compute(i);
}
```

**Q8.** Select a proper formula for variables first and last, controlling what is executed at each node.

(a) first = me*floor(n_over_p); last=(me+1)*floor(n_over_p)-1;
(b) first = me*floor(n_over_p)+1; last=(me+1)*floor(n_over_p);

(c) first = me*ceil(n_over_p); last=(me+1)*ceil(n_over_p)-1;
(d) first = me*ceil(n_over_p)+1; last=(me+1)*ceil(n_over_p);

**Q9**. Assume computation cost of the above code is dominated by function compute(i) with cost i+1 time units. Other cost such as loop overhead is considered as 0. When p=3, and n=6, what is the workload assigned to the first processor, second processor, and third processor? The workload of a processor is the sum of computation cost of all tasks assigned to this processor.

(a) 2, 2, 2
(b) 3, 7, 11
(c) 7, 7, 7
(d) 5, 7, 9
(e) All of them are incorrect.

**Q10:** The following SPMD code uses a different mapping method (called cyclic mapping) to parallelize the sequential code.

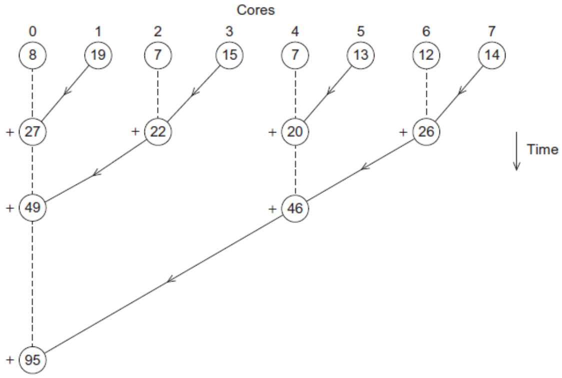
```
int me=mynode();
p=noproc();
for(int  i=me;  i<n; i=i+p){
  compute(i);
}
```

Again assume computation cost of the above code is dominated by function compute(i) with cost i+1 time units. Other cost such as loop overhead is considered as 0. When p=3, and n=6, what is the workload assigned to the first processor, second processor, and third processor?

(a) 2, 2, 2
(b) 3, 7, 11
(c) 7, 7, 7
(d) 5, 7, 9
(e) All of them are incorrect

**Questions 11-12**. Given p processors and each processor i owns a value x[i] where 0≤i<p. The following pseudo SPMD code adds these p numbers from p processors in a tree summation style as illustrated below when p=8 and process 0 has the final summation result. Send(data, dest) means to send data to destination processor dest. Receive(data, src) means to receive data from source processor src. Notation % is the integer mod function that computes the remainder. We call the code is not working if Proc 0 does not have the correct final result, the code does not terminate, or the code has pending

communication (send a message but there is no process to receive this message or receive a message while there is no message sent to this process).



```
me=mynode();
p= noproc();
sum=x[me];
if( (me %2 !=0  &&me>0)
        Send (sum, me -1);
For( gap=1; gap<p; gap =gap*2){
        if(me%(2*gap) ==0)){
                Receive(other_sum, me+gap);
                sum= other_sum+sum;
                if( (me % (4*gap)) !=0  &&  me>=2*gap)
                        Send (sum, me -2*gap);
        }
}
```

**Q11**: Select a true statement from the following four choices.

(a) This code works correctly only when p is a power of 2
(b) This code works correctly only when p is not a power of 2
(c) This code works correctly for any integer p.
(d) None of above answers is correct.

**Q12**. What is the total number of receive functions called in the above SPMD code for processor 0 and processor 2 respectively? Assume p is a power of 2 and function log() uses base 2.

(a) 1, 1 (b) log(p), 1 (c) 1, log (p) (d) 3, 1 (e) log (p), log(p) -1 (f) None of them is correct