CSC/ECE 506: Architecture of Parallel Computers
Sample Test 1

This was a 120-minute open-book test. You were allowed use the textbook and any course notes that you had. You were not allowed to use electronic devices. You were to answer five of the six questions. Each question was worth 20 points. If you answered all six questions, your five highest scores counted.

Question 1. *[Average score 16.8]* *(1 point each)* For each of the following terms or concepts, choose the programming model with which it is most closely associated:

- Data parallel (write "D")
- Message-passing ("M")
- Shared address space ("S")

(a) Array processor	*Ans.:* D	(k) Local array indexes	*Ans.:* M
(b) Buffer-management overhead	*Ans.:* M	(l) Lock operations	*Ans.:* S
(c) Coherence problems	*Ans.:* S	(m) Logically single thread of control	*Ans.:* D
(d) Peterson's algorithm	*Ans.:* S	(n) Loosely coupled multiprocessor	*Ans.:* M
(e) Distributed-memory multiprocessor	*Ans.:* M	(o) NUMA machine	*Ans.:* S
(f) GPU	*Ans.:* D	(p) **receive** operation	*Ans.:* M
(g) Barrier synchronization	*Ans.:* S	(q) CUDA Kernel Method	*Ans.:* D
(h) **for_all** loop	*Ans.:* D	(r) SPMD	*Ans.:* D
(i) cluster architecture	*Ans.:* M	(s) Shared-memory multiprocessor	*Ans.:* S
(j) Thread blocks	*Ans.:* D	(t) Tightly coupled multiprocessor	*Ans.:* S

The most frequently missed part was (k). More people thought local array indexes are associated with data-parallel algorithms. In message-passing, when an array is divided up among the different processors, each processor uses a local index to iterate through its portion of the array. On data-parallel machines, registers aren't used in this fashion because the different processing elements go from one element to another of their array portion as dictated by the control processor. However, data-parallel machines do have local index registers, so I gave credit for the "D" answer.

The next most frequently missed part was (o). A NUMA machine *can* address all its memory, so it can run shared-memory programs. Thus, the shared-memory model is most closely associated with it.

Question 2. <GPU L3-video & Program 1> (George)
(a) *(2 points per blank, max. 10 points)* Modern GPUs can support a maximum of 1024 blocks and maximum of 65536 threads per block. Fill in the blanks in the host code given below to check for these two conditions before invoking the kernel. *Note:* This is not the complete code. Memory allocation and error-checking have been removed so the code can fit on one page.

```
#include <stdio.h>
#include <cuda_runtime.h>

#define MAX_THREADS _65536__

#define MAX_BLOCKS ___1024__
```

```
__global__ void
vectorAdd(const float *A, const float *B, float *C, int numElements) {
    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < numElements)
         C[i] = A[i] + B[i];
}

int main(int argc, char * argv[]){
  ....
 // here numElements = argv[1]

 int threadsPerBlock = 256;
  int blocksPerGrid =(numElements + threadsPerBlock - 1) / threadsPerBlock;

 // Check the boundary conditions for max number of elements that GPU can handle

  if(numElements> __MAX_THREADS__ * _MAX_BLOCKS_ ) {
    printf("The given number of elements cannot be executed in one go; tiling   is
required\n");
    exit(-1);
  }

  .......

 // The idea here is to meet the physical limitations of the GPUs before calling the kernel

  while( _blocksPerGrid_ > MAX_BLOCKS ) {

    int _threadsPerBlock_  *= 2;
    int blocksPerGrid = (int)((numElements+threadsPerBlock-1)/threadsPerBlock);
  }

 // Launch the Vector Add CUDA Kernel

  printf("CUDA kernel launch with %d blocks and %d threads per block\n",
       blocksPerGrid, threadsPerBlock);

  vectorAdd <<< blocksPerGrid, threadsPerBlock >>>(d_A, d_B, d_C, numElements);
  return 0;
}
```

(b) *(2 points)* What will be the output of the program if the executable file is executed like this: "`./vectorAdd 68400000`"?

Answer: The program will hit the return condition and print, "The given number of elements cannot be executed in one go; tiling is required."

(c) *(5 points)* Fill in the blanks in the output of the printf before the kernel invocation when the file is executed like this : "`./vectorAdd 307969`"

CUDA kernel launch with _602_ blocks and _512_ threads per block

Initially the block number will be calculated as:
blocksPerGrid = (307969+ 256 -1)/256 = 1204
Now 1204> 1024. Hence the while loop below will execute.
While loop:
threadsPerBlock = 2 × 256 = 512
blocksPerGrid = (307969 + 512 – 1)/512 = 602.5 → 602

Since 602 is less than 1024, the loop will execute only once.

(d) *(3 points)* The **if** statement is included to turn off threads that are not operating on grid elements. In part (c), how many threads does the if statement "turn off"?

Answer: total number of threads created = 602 × 512 = 308,224

The number of threads which are not "turned off" = 307,969

Hence the number of threads which will be turned off = 308,224– 307,969 = **255**

Question 3. <Amdahl's law. L3> (George)
Suppose you're tasked with working with a partially parallelizable program (there is a portion of the program that *must* be executed serially). When executed with 1 processor, the execution time (T_0) is 10 seconds. When executed with N = 3 processors, the execution time (T_1) is 8 seconds.

(a) *(3 points)* What is the speedup achieved by the second execution over the first?

Answer: $S = \frac{T_0}{T_1} = \frac{10}{8} = 1.25$

(b) *(5 points)* What fraction of execution time ***s*** consists of serial code?

Answer:

$$\frac{T_1}{T_0} = s + \frac{(1-s)}{p}$$

$$s = \frac{p\,T_1/T_0 - 1}{p-1} = \frac{3\times 8/10-1}{3-2} = 0.7$$

(c) *(5 points)* How long will it take the program to execute when run with 4 processors?

Answer:

$$T_2 = s \times T_0 + \frac{1-s}{p} \times T_0 = 0.7 \times 10 + \frac{0.3}{4} \times 10 = 7.75$$

(d) *(3 points)* What is the limit on achievable speedup for this algorithm?

Answer:

$$max\ speedup = \frac{1}{s} = 10/7 \approx 1.4286$$

(e) *(5 points)* According to Gustafson's law, when more processors are available, workloads will expand to take advantage of them. An expanded version of the program in this question can be executed in 8 seconds, but it requires 100 processors. What is the amount of execution time if only 1 processor is used?

You can assume the fraction of execution time ***s*** consisting of serial code stays the same.
Answer:

$$8 = s \times T'_0 + \frac{1-s}{p} \times T'_0 = 0.7 \times T'_0 + \frac{0.3}{100} \times T'_0$$

$$T'_0 = 8/(0.7 + 0.003) = \frac{800}{703} \approx 11.3798$$

Question 4. <Cache & cache organization L4 & L5> (Sharan)

This question concerns inclusion policies. Suppose we have a cache that has an equal number of L1 and L2 sets. The L1 cache is 2-way, and the L2 is 4-way associative. Let's focus first on Set 0 in both the L1 and the L2.

(a) *(8 points)* Assuming that this is an exclusive cache, fill in the missing block numbers (A–G) in the table below. If a line is empty, write a dash ("—").

Ref. #	Block refer-enced	L1 cache (Set 4)		Exclusive L2 cache (Set 4)			
		MRU line	LRU line	MRU line	…		LRU line
1	A	A	—	—	—	—	—
2	B	B	A	—	—	—	—
3	D	D	B	A	—	—	—
4	C	C	D	B	A	—	—
5	A	A	C	D	B	—	—
6	D	D	A	C	B	—	—
7	B	B	D	A	C	—	—
8	G	G	B	D	A	C	—
9	F	F	G	B	D	A	C
10	C	C	F	G	B	D	A
11	H	H	E	F	G	B	D

(b) *(2 points each)* What is the number of the first reference (Ref. #) where—

- the exclusive cache would eject a block from the L2? *Answer:* Exclusive would eject when the 7th block from set 4 Is referenced. That is H, at reference 11.
- an *inclusive* cache would eject a block from the L2? *Answer:* Inclusive would eject when the 5th block from set 4 Is referenced. That is G, at reference 8.
- a *NINE* cache would eject a block from the L2? *Answer:* At reference 8 i.e G, NINE would eject a block.
- the contents of the L2 differ between an exclusive cache and a NINE cache? *Answer:* At reference 5, when A is re-referenced, it stays in the L2 with NINE, but not with exclusive.
- one of the caches (inclusive, exclusive, or NINE) would have an L2 miss while another has an L2 hit? *Answer:* Reference 8, where B is referenced. This will be an L2 hit unless the policy is inclusive.

(c) *(2 points)* Suppose that the line size is 1024 bytes and the L1 has 2^6 sets. What is the *minimum* possible distance between the first address in A and the first address in B? *Answer:* Since A and B both map to Set 4, their 6-bit index field must be the 0. The first address in each block has a 10-bit offset field of 0. So the least significant 16 bits of both addresses are all 0. Thus, A and B must be at least 216 bytes apart.

Question 5. <cache coherence, consistency, L6 & L7> (Sharan)

For each of the following systems, tell whether they would need a cache-coherence protocol, a memory-consistency model, neither, or both. For partial credit, you may explain your ans

(a) A uniprocessor system that has a 2-level cache *Answer:* Neither

(b) A single-chip multicore processor with caches where memory is connected to the processors via a shared bus *Answer:* Cache coherence, but not consistency.

(c) A multiprocessor that has no caches *Answer:* Memory consistency, but not both

(d) A NUMA multiprocessor with single-level caches *Answer:* Both

(e) A data-parallel processor without caches where all communication between processors occurs through registers. *Answer:* Neither

Question 6. <shared memory parallel programming, Dependencies, L8 & L9> (George)
(5 points each) In this question, you are asked to compute the speedup and efficiency of different kinds of parallelization. In each case,

(i) Identify the kind of parallelism for the parallelized version on the right.

(ii) compare the speedup of the parallelized version of the code with the serial version;

(iii) compute the efficiency, assuming that N = 100 processors are available, and

(iv) compute the efficiency, assuming that no more processors are available than can effectively be used by the parallelization. That is, if the parallelization can use only two processors, then base your efficiency calculation on two processors being available.

In all parts, you may assume that N = 100, and $T_{S1} = T_{S2} = \ldots = T_{S4} = 1$. Remember to include your steps for partial credit.

(a)
```
for (i = 1; i <= N; i++) {
   S1: x[i] = z - B[i]*A[i];
   S2: y[i] = a[i] + c[i];

}
```
⇒
```
for_all (i = 1; i <= N; i++) {
   S1: x[i] = z - B[i]*A[i];
   S2: y[i] = a[i] + c[i];
}
```

Answer: $T_{serial} = 100 \times (1+1) = 200$ $T_{parallel} = 1+1 = 2$

(i) DOALL (ii) Speedup = 100 (iii) Efficiency = 100/100 = 1 (iv) Efficiency = 100/100 = 1

(b)
```
for (i=1; i<=N; i++) {
   S1: a[i] = c[i] - d[i];
   S2: b[i] = b[i-1]+a[i]*d[i];
}
```
⇒
```
post(0);
for_all (i=1; i<=N; i++) {
   S1: a[i] = c[i] - d[i];
   S2: temp = a[i] * d[i];
   wait(i-1);
   S3: b[i] = b[i-1] + temp;
   post(i);
}
```

Answer: T_{serial} = 100 × (1+1) = 200 $T_{parallel}$ = 1 +1+ 100 × 1 = 102

(i) DOACROSS (ii) Speedup = 200 / 102 ≈1.96 (iii) Efficiency ≈1.96/100 ≈0.0196 (iv) Efficiency≈1.96/100

(c)

```
for (i = 1; i <= N; i++) {
  S1: a[i+1] = a[i]+d[i];
  S2: c[i+2] = x - b[i]/c[i];
}
```

⇒

```
for (i = 1; i <= N; i+=1) {
  S1: a[i+2] = a[i]+d[i]; }
for (i = 1; i <= N; i+=2) {
  S2: c[i+2] = x - b[i]/c[i]; }
for (i = 2; i <= N; i+=2) {
  S2: c[i+2] = x - b[i]/c[i]; }
```

Answer: T_{serial} = 100 × (1+1) = 200 $T_{parallel}$ = max(100,50,50) = 100

(i) Function (?) (ii) Speedup = 200/100 ≈ 2 (ii) Efficiency ≈ 2/100 = .02 (iii) Efficiency ≈ 2/3

(d)

```
for (i=1; i<=N; i++) {
  S1: d[i+1] = a[i] / d[i];
  S2: c[i+1] = c[i] * d[i] * x;
}
```

⇒

```
for (i=1; i<=N; i++) {
  d[i+1] = a[i] / d[i];
  post(i);
}
for (i=1; i<=N; i++) {
  wait(i);
  c[i+1] = c[i] * d[i] * x;
}
```

Answer: T_{serial} = 100 × (1+1) = 200 $T_{parallel}$ = 1 + 100 × 1 = 100

(i) DOPIPE (ii) Speedup = 200/100 = 2 (iii) Efficiency ≈ 2/100 = 0.02 (iv) Efficiency ≈ 2/2 = 1