UNIVERSITY OF MORATUWA

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

MSc in Computer Science
2014 Intake Semester 2 Examination

CS5225 PARALLEL AND CONCURRENT PROGRAMMING

Time allowed: 2 Hours August 2014

ADDITIONAL MATERIAL: *None*

INSTRUCTIONS TO CANDIDATES:

1. This paper consists of **7** questions in **7** pages.
2. Answer **All** questions from Section A and any **THREE** questions from Section B.
3. Start answering each of the main questions on a new page.
4. The maximum attainable mark for each question is given in brackets.
5. This examination accounts for 50% of the module assessment.
6. This is a closed book examination.

   NB: It is an offence to be in possession of unauthorised material during the examination.
7. Only calculators approved by the Faculty of Engineering are permitted.
8. Assume reasonable values for any data not given in or with the examination paper. Clearly state such assumptions made on the script.
9. In case of any doubt as to the interpretation of the wording of a question, make suitable assumptions and clearly state them on the script.
10. This paper should be answered only in English.

Section A

Answer All Questions

Question 1 (10 marks)

Select the most appropriate answer, and write the corresponding sub-question number and the answer number in your answer book. [10 × 1]

(i) Which of the following factors contributed to the migration from uniprocessor systems to shared memory multiprocessor systems?

A) Flattening in clock speed of CPUs

B) Flattening in Instruction-Level Parallelism (ILP)

C) Increased power consumption

D) All of the above

(ii) Liveness property says

A) All processes get a fair share of CPU
B) No process will starve
C) Nothing bad happens ever
D) Something good happens eventually

(iii) What makes GPUs an attractive choice for parallel computing?

A) High cost
B) High throughput
C) Large inbuilt memory
D) Low latency

(iv) Which of the following is **NOT** an advantage of Monitors?

A) Implementation doesn't rely on the OS or programming language

B) Enforces mutual exclusion

C) Synchronization code doesn't depend on the number of threads competing

D) Synchronization code is centralized

(v) Which of the following statements are True about Parallel Algorithms?

(p) Work ≥ Span
(q) Throughput is always a good measure of performance
(r) By allocating more resources to a scalable parallel algorithm for a given problem, we can solve a larger version of the problem

A) (*p*) and (*q*) only
B) (*p*) and (*r*) only
C) (*q*) and (*r*) only
D) All three

(vi) In which of the following phase does "each computational task is assigned to a processor while attempting to satisfy competing goals of maximizing processor utilization and minimizing communication costs"?

A) Agglomeration
B) Communication
C) Mapping
D) Partitioning

(vii) Which of the following Solution Patterns for Parallelism is most suitable to solve the following problem?

```
int[] A = .. int[] B = .. int[] C = ..
for (int i; i < N; i++){
    C[i] = F(A[i], B[i - 1], C[i])
}
```

A) Divide and Conquer
B) Fork/Join
C) Loop Parallel
D) None of the above

(viii) Which of the following MPI function can be used by all the processes to find the maximum value among a large array of numbers?

A) MPI_Allreduce
B) MPI_Gather
C) MPI_Reduce
D) MPI_Scatter

(ix) Which of the following is **NOT** a step involved while writing a Map-Reduce job?

A) Write the Combine function
B) Write the Kernel
C) Write the Mapper
D) Write the Reducer

(x) Which of the following statements are True about Concurrent Data Structures?

(p) Locks, semaphores, and monitors serialize the execution of mutually exclusive code
(q) A wait-free operation is guaranteed to complete after a finite no of its own steps
(r) Wait-freedom is weaker than Lock-freedom

A) (*p*) and (*q*) only
B) (*p*) and (*r*) only
C) (*q*) and (*r*) only
D) All three

Question 2 (10 marks)

State whether the following statements are TRUE or FALSE. Give one sentence justification for your answer.

Write the corresponding sub-question number and the justification in your answer book. [5 × 2]

(i) Starvation may automatically go away after some time.

(ii) $T_p \geq T_1/P$

where P is the number of processors, and T_p and T_1 are execution times on P and single processor systems, respectively.

(iii) Work stealing is a static load balancing technique.

(iv) MPI can be applied to any large data-intensive application.

(v) Map-Reduce programming model provides a powerful abstraction to handle embarrassingly parallel problems.

Question 3 (5 marks)

Write the most appropriate short answer (word/phrase) for the following questions.
Write the corresponding sub-question number and the answer in your answer book. [5 × 1]

(i) A critical section that lets only N (≥ 1) threads to enter at a given time is called a ________.

(ii) When a __________ lock occurs, each thread repeats the same state again and again, but doesn't progress any further.

(iii) While using data-level parallelism, large data units lead to _________________ .

(iv) Temporal and __________ locality in data access can be used to design efficient/fast parallel algorithms.

(v) _______________ breaks a computational task into small steps (which may have dependencies) and assign execution of steps to different threads.

Section B

Answer Any THREE Questions

Question 4 (25 marks)

(i) Consider the following program with 2 threads.

```
Thread 1
      while(true){
            print "Red" + math.rand(10);
      }

Thread 2
      while(true){
            print "Blue" + math.rand(10);

      }
```

Math.rand(10) generates a random value between 1 and 10.

Change the above program to make sure the sum of all *Red* values it had printed so far is always less than the sum of all *Blue* values it has printed. For example, if *Blue* had printed 1, 5, and 7 then it is OK for *Red* to print 2 and 9 because $2 + 9 < 1 + 5 + 7$. [12]

(ii) You are given a matrix $\mathbf{A}_{m\times n}$ and a vector $\mathbf{V}_{n\times 1}$. Suppose both m and n are expected to be in the thousands (i.e., $m, n > 1000$). Outline a CUDA program to calculate the matrix-vector multiplication **AV**. Your solution should include the code for the Kernel function and the code required to invocate the Kernel function. [9]

(iii) Using a suitable diagram briefly explain how a Combining Tree can increase the parallelism while updating a shared counter. [4]

Question 5 (25 marks)

(i) Consider the following program with 2 threads.

```
Thread 1
      while(true){
            print "Red";
            print "Red";
      }

Thread 2
      while(true){
            print "Blue";
      }
```

(a) Provide 4 possible outcomes of the above program. [2]

(b) Rewrite the above program using a semaphore(s) and 1 print statement per thread such that we get the following sequence of outputs. [9]

Red, *Red*, *Blue*, *Red*, *Red*, *Blue*, *Red*, ….

(ii) Consider an $m \times n$ matrix A. 1-norm (magnitude) of matrix A is obtained as follows:

$$\| A_1 \| = \max_j \sum_{i=0}^{m-1} |a_{i,j}|$$

where i and j are row and column numbers, respectively.

Following is an illustration of steps involved in calculating 1-norm of a matrix A.

$$A = \begin{bmatrix} 2 & -5 \\ 3 & 4 \end{bmatrix} \rightarrow \max[5 \quad 9] \rightarrow \| A_1 \| = 9$$

(a) Write the pseudocode of a parallel algorithm to calculate 1-norm of a $m \times n$ matrix A. Use a Parallel For loop(s). [5]

(b) How much work to be performed by the algorithm? [3]

(c) What is the span of the algorithm? [2]

(d) How much parallelism is available in the algorithm? [2]

(e) Is your algorithm still useful, if m is small and n is large? Briefly Explain. [2]

Question 6 (25 marks)

(i) Zipf's Law states that in the English language, the frequency of any word is inversely proportional to its rank in the frequency table. Thus, the most frequent word will occur approximately twice as often as the 2nd most frequent word, three times as often as the 3rd most frequent word, and so on. This can be formally written as the probability of encountering the r-th most common word is roughly $P(r) = 0.1/r$.

Suppose we want to check whether the Zipf's law proposed in 1935 is still true by counting the frequency distribution of word occurrences of the top 100 books ranked by the New York Times during the last 25 years.

Write a pseudocode that shows how you can use Map-Reduce to check whether the Zipf's law is still true. The answer should provide *map* and *reduce* functions. [17]

(ii) What type of load balancing would you recommend while solving following problems? Briefly explain.

(a) Frequency distribution of word occurrences of the top 100 books ranked by the New York Times during the last 25 years. [4]

(b) Matrix-Matrix multiplication. [4]

Question 7 (25 marks)

(i) What are the four conditions required for deadlock to occur? Explain all four using a suitable example(s). [5]

(ii) Following is an MPI style message passing system. *mq* refers to a message queue.

```
Thread 1
      mq1.receive();
      mq2.send(x);

Thread 2
      mq2.receive();
      mq1.send(y);
```

a) Will this code lead to a deadlock? Briefly explain. [2]

b) Does this program, violate the four conditions required for a deadlock to occur? Explain. [3]

(iii) The variance of a set of numbers **X** can be calculated as follows, where E[ ] is the expected value, and μ is mean.

$$\mathrm{Var}(X) = E[(X - \mu)^2] = E[X^2] - (E[X])^2$$

Outline an MPI program (using pseudo code) that can be used to calculate the variance of one million numbers. Once the calculation is complete all process involved in the computation need to know the value. Use relevant MPI functions that are given in the Appendix. Note that it is impractical to create one million concurrent processes.

Hint: $E[X] = \mu$ [15]

Appendix – MPI Functions

```
. . .
#include <mpi.h>
. . .
int main(int argc, char* argv[]) {
   . . .
   /* No MPI calls before this */
   MPI_Init(&argc, &argv);
   . . .
   MPI_Finalize();
   /* No MPI calls after this */
   . . .
   return 0;
}
```

```
int MPI_Init(int *argc, char **argv)
int MPI_Comm_size(MPI_Comm comm, int *size)
int MPI_Comm_rank(MPI_Comm comm, int *rank)
int MPI_Finalize()
int MPI_Send (void *buf,int count, MPI_Datatype datatype, int dest, int
      tag, MPI_Comm comm)
int MPI_Recv (void *buf,int count, MPI_Datatype datatype, int source, int
      tag, MPI_Comm comm, MPI_Status *status)
int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype
      datatype, MPI_Op op, int root, MPI_Comm comm)
int MPI_Allgather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void
      *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)
int MPI_Allreduce (void *sendbuf, void *recvbuf, int count, MPI_Datatype
      datatype, MPI_Op op, MPI_Comm comm)
int MPI_Bcast( void *buffer, int count, MPI_Datatype datatype, int root,
      MPI_Comm comm)
int MPI_Gather(void *sendbuf, int sendcnt, MPI_Datatype sendtype, void
      *recvbuf, int recvcnt, MPI_Datatype recvtype, int root,
      MPI_Comm comm)
int MPI_Scatter(void *sendbuf, int sendcnt, MPI_Datatype sendtype, void
      *recvbuf, int recvcnt, MPI_Datatype recvtype, int root,
      MPI_Comm comm)
```

Operation Value	Meaning
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical and
MPI_BAND	Bitwise and
MPI_LOR	Logical or
MPI_BOR	Bitwise or
MPI_LXOR	Logical exclusive or
MPI_BXOR	Bitwise exclusive or
MPI_MAXLOC	Maximum and location of maximum
MPI_MINLOC	Minimum and location of minimum

-------------------------- END OF THE PAPER --------------------------