Index No:							

UNIVERSITY OF MORATUWA

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

B.Sc. Engineering
2014 Intake Semester 6 Examination

CS4532 CONCURRENT PROGRAMMING

Time allowed: 2 Hours February 2018

ADDITIONAL MATERIAL: *None*

INSTRUCTIONS TO CANDIDATES:

1. This paper consists of **four (4)** questions in **eleven (11)** pages including Annex.
2. Answer **All** questions.
3. Answer the questions on the paper itself. **DO NOT** exceed the given space.
4. For MCQ and True/False questions, select the most appropriate answer. No penalty for wrong answers.
5. The maximum attainable mark for each question is given in brackets.
6. This examination accounts for 50% of the module assessment.
7. This is a closed book examination.

   NB: It is an offence to be in possession of unauthorized material during the examination.
8. Only calculators approved by the Faculty of Engineering are permitted.
9. Assume reasonable values for any data not given in or with the examination paper. Clearly state such assumptions made on the script.
10. 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.
11. This paper should be answered only in English.

Q1	Q2	Q3	Q4	Total

Index No:							

Question 1 (25 marks)

Fill in the blanks using one of the following terms. [1 × 5 marks]

Agglomeration	Banker's Algorithm	Barrier	Deadlock
Efficiency	Mapping	Multiplex	
Ostrich Algorithm	Race Condition	Utilization	

(i) When 2 or more processes read/write a shared resource and the final result depends on who runs correctly, it is referred to as ____________________ .

(ii) Parallel Programming let us get faster results at the cost of __________________ .

(iii) In __________________ phase tasks may be combined into larger tasks to improve the performance and reduce development costs.

(iv) ___________________________ is a dynamic strategy for dealing with deadlocks.

(v) _______________ is a generalized form of Rendezvous.

A forecasting program that predicts the path of a thunderstorm typically takes 6 hours to run on the current uniprocessor system. The Meteorology department plans to run this program in 1 hour (saving 5 hours), as it can provide advanced warning to the public who may get affected due to the thunderstorm. They are thinking of achieving this by benefiting from parallel programing on multi-core processors. Meteorology department noted that the fraction of the forecasting program that can be parallelized is 80%. Following 2 options are suggested to them:

(a) Purchase a 32-core server where the specification of a new CPU core is similar to the specification of the current uniprocessor. Price per core is \$80.
(b) Purchase a 32-core server where the specification of a new CPU core is 1.5× faster than the specification of the current uniprocessor. Price per core is \$125.

(vi) Compare pros and cons of options (a) and (b), and then recommend the most suitable option to achieve the above speed up target. [7 marks]

Hint: Amdahl's law in the context of concurrent programming can be given as $1/(1 - p + p/n)$, where n is the number of processors and p is the fraction that can be parallelized.

Consider the following 3 threads used to control 3 lights *L1*, *L2*, and *L3*:

Thread 1	Thread 2	Thread 3

`While(True){`
`    L1.on()`
`    Wait(5)`
`    L1.off()`
`    Wait(5)`
`}`
	`While(True){`
	`    L2.on()`
	`    Wait(5)`
	`    L2.off()`
	`    Wait(5)`
	`}`
		`While(True){`
		`    L3.on()`
		`    Wait(5)`
		`    L3.off()`
		`    Wait(5)`
		`}`

(vii) List 3 possible sequences that the lights may glow, except the case where all lights are on or off. [3 marks]

(viii) Propose a semaphore-based solution to make sure in each round *L2* will not switch on before *L1*, and *L3* will not switch on before *L1* and *L2*. [7 marks]

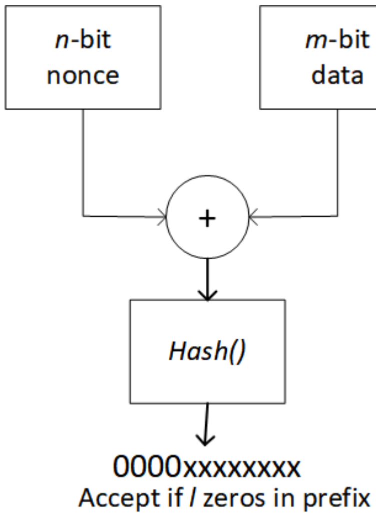
Index No:							

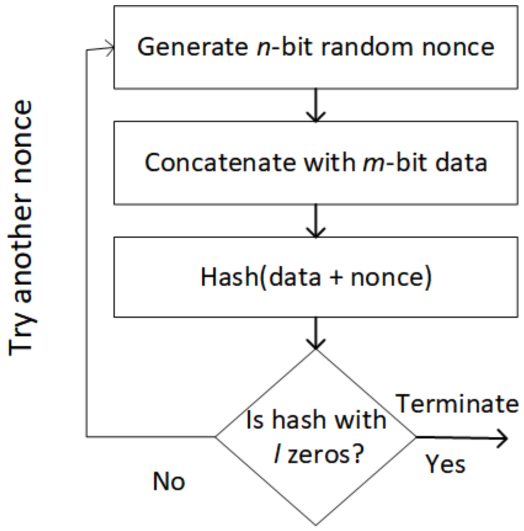
(ix) Could the same result be achieved, if we replace the semaphore(s) with condition variables? Briefly discuss. [3 marks]

Question 2 (25 marks)

A Proof of Work (PoW) is a piece of data which is difficult (costly and time-consuming) to produce but easy for others to verify whether it satisfies certain requirements. PoW is useful in several applications such as mitigating denial of service attacks, reduction of spam mails, and crypto currencies. You can attack/break these solutions if you can perform PoW very fast.

Following is a possible implementation of a PoW solution using hashing. Given a *m*-bit data, a node needs to find a *n*-bit nonce ($m >> n$) that produces a hash where all *l* Most Significant Bits (MSBs) are zeros. This is typically achieved by trying all possible combinations of nonce (from 0 to 2^n-1) until a hash with the desired number of zero MSBs is found. Suppose we need to figure out a way to speedup up this process.





Tick **TRUE** or **FALSE**. Give **one sentence justification**. [2 × 3 marks]

		True	False
(i)	This is an embarrassingly parallel problem.		

		True	False
(ii)	Given k processors to solve the problem, maximum speedup is 2^n.		
(iii)	Dynamic load balancing is desirable in this solution.		

(iv) Outline a solution to speed up the calculation of PoW using one of the Solution Patterns for Parallelism discussed in the class. Your target is to find a suitable nonce as fast as possible. [6 marks]

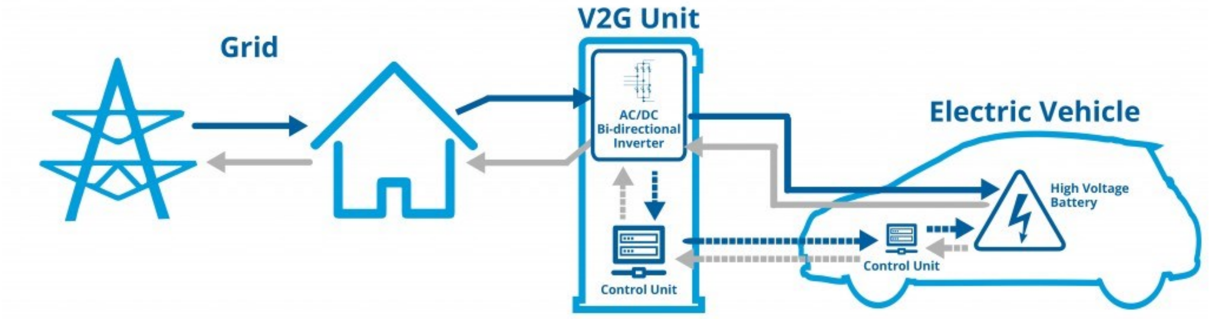
(v) Outline a GPU-based solution to speed up the calculation of PoW to find a suitable nonce. You solution should include a kernel and how to invoke it. Assume you have access to inbuilt *CUDA_Hash*(*data*) function. [13 marks]

Kernel launch code:

Kernel code:

Question 3 (25 marks)

With the popularity of electric cars one of the biggest challenges the electricity providers face is the large increasing in power consumption throughout the day. For example, some cars charge at normal rate consuming ~16 A while rapid charging draw ~32 A from the power grid. Aggregation of total current drawn by multiple cars on a neighborhood introduces a significant load to the power grid. Moreover, controlling the current aggregation is difficult as cars are charged from the respective houses and the car owners do not have any visibility on what other cars are charged at the same time.



Source: http://www.cenex.co.uk/vehicle-to-grid/

Index No:							

As a way of solving this problem CEB is proposing to allocate a guaranteed 100 A and 400 A for a given neighborhood to charge cars during peak and off-peak hours, respectively. To utilize this capacity an agent-based module is to be attached to the charging (V2G) unit at a house. Agent-based modules at each V2G unit are expected to negotiate with each other to decide who will charge and when. The proposed unit will switch on the V2G units only if there is sufficient current capacity in the power grid. If not, agent-based module will retry with a random delay. You are required to develop a solution for the concurrent allocation of charging current.

Tick **TRUE** or **FALSE**. Give **one sentence justification**. [2 × 6 marks]

		True	**False**
(i)	In the context of this problem safety property should ensure that current drawn from the electricity grid is within the allocation.		
(ii)	In the context of this problem liveness property should ensure that each car eventually gets an opportunity to charge.		
(iii)	It is sufficient to provide 3-boundard waiting in the proposed negotiation algorithm among agents.		
(iv)	A timetable specifying the charging time of each car satisfy the safety, liveness, and efficiency properties.		
(v)	There is a possibility that an agent-based solution may lead to a deadlock.		
(vi)	This is a shared-memory mutual exclusion problem.		

(vii) Give pseudo code of a negotiation solution designed to ensure concurrent charging of multiple cars (both normal and rapid charging) while satisfying peak and off-peak maximum current consumption limits. [13 marks]

Index No:							

Question 4 (25 marks)

Faulhaber's formula, expresses the sum of the p-th powers of the first n positive integers. We can write the sequence as:

$$f(p, n) = 1^p + 2^p + 3^p + 4^p + \ldots + n^p$$

(i) "*Regardless of whether we use PThreads, OpenMP, CUDA, or MPI balancing the load among multiple threads/nodes is a difficult problem*." Do you agree or disagree with this statement? Justify. [4 marks]

(ii) Outline PThread or OpenMP based solution to calculate Faulhaber's formula $f(p, n)$ for a given p ($1 \leq p \leq 1{,}000$) and n ($1 \leq n \leq 100{,}000$). Each thread should take a number n, calculate n^p, and then add it to the total to calculate $f(p, n)$. [9 marks]

(iii) Outline an MPI program (using pseudo code) capable of calculating Faulhaber's formula $f(p, n)$ where $1 \leq p \leq 1{,}000$ and $1 \leq n \leq 100{,}000$. Indicate MPI functions and key parameters (it is not essential to follow exact function signature). Result need to be in node 0. [12 marks]

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 -------------------------