



DUBLIN CITY UNIVERSITY

SEMESTER ONE EXAMINATIONS 2010

MODULE TITLE: Concurrent Programming
MODULE CODE: CA463/CA463D
COURSE: BSc. in Computer Applications (Software Engineering Stream),
Study Abroad
YEAR: 4/X
EXAMINERS: Dr. P. Gibson
Dr. F. Bannister
Dr. Martin Crane Ext: 8974
TIME ALLOWED: 2 Hours
INSTRUCTIONS: Please answer **any 3** questions:

*Requirements for this paper
Please tick (X) as appropriate*

- | | |
|--------------------------|------------------------------|
| <input type="checkbox"/> | <i>Log Table</i> |
| <input type="checkbox"/> | <i>Graph Paper</i> |
| <input type="checkbox"/> | <i>Attached Answer Sheet</i> |
| <input type="checkbox"/> | <i>Statistical Tables</i> |
| <input type="checkbox"/> | <i>Floppy Disk</i> |
| <input type="checkbox"/> | <i>Actuarial Tables</i> |

THE USE OF PROGRAMMABLE OR TEXT STORING CALCULATORS IS EXPRESSLY FORBIDDEN

Please note that where a candidate answers more than the required number of questions, the examiner will mark all questions attempted and then select the highest scoring ones.

PLEASE DO NOT TURN OVER THIS PAGE UNTIL YOU ARE INSTRUCTED TO DO SO

Question 1**[Total marks: 33]**

1(a)

[6 marks]

Define in words and as a formula what is meant by Amdahl's law.

Answer:

Amdahl's Law

Gene Amdahl divided a program into 2 sections, one that is inherently serial and the other which can be parallel. Let s be the fraction of the program which is inherently serial. Then,

$$\text{Speed-up, } S_p = \frac{T(1 + (1-s))}{T\left(s + \frac{(1-s)}{P}\right)} = \frac{P}{1 + (P-1)s} \leq \frac{1}{s}, \forall P > 1$$

If $s = 5\%$, then $S \leq 20$.

1(b)

[27 marks]

Amdahl's Law was originally formulated for a fixed problem size, but, as was shown from the Sandia experiments, Amdahl's result can be used to characterize the behaviour of the serial fraction s as a function of the number of processors p as the problem size grows with p . According to Amdahl's Law, how would s have to vary in order to maintain each of the following conditions? Your answers should express s as a function of p .

1(b) (i)

[5 marks]

(i) Speedup of $p/2$.**Answer:**

According to Amdahl's Law, if s is the serial fraction, then the parallel time T_p is given by

$$T_p = s T_1 + (1-s) T_1/p \text{ (where } T_1 \text{ is time on one proc),}$$

and hence the speedup S_p is given by

$$S_p = T_1/T_p = p/(sp + (1-s)).$$

If we set Amdahl's formula for S_p equal to $p/2$ and solve for s , we obtain $s = 1/(p-1)$.

1(b) (ii)

[5 marks]

(ii) Speedup of $p-1$.**Answer:****Module Code: CA463****Page 2 of 9****Semester 1 Exam****MODERATION STAGE**

If we set Amdahl's formula for S_p equal to $p - 1$ and solve for s , we obtain $s = 1/(p - 1)2$.

1(b) (iii)

[5 marks]

- (iii) If the efficiency of a parallel code is the serial cost: parallel cost ratio and is given by $E_p = T_1/(p T_p)$ (where T_1 is time on one proc). How would s have to vary to maintain Constant efficiency?

Answer:

According to Amdahl's Law, $S_p = T_1/T_p = 1/(sp + (1 - s))$ and the efficiency, $E_p = T_1/(p T_p)$.

For a constant Efficiency, say E , and solving for s gives

$$s = ((1/E) - 1)/(p - 1).$$

Thus, as p grows, s must diminish proportionally to $1/(p - 1)$.

1(b) (iv)

[12 marks]

- (iv) Given a program to be parallelised is split up into four parts in sequence: P1;P2;P3;P4 such that $P_1 = 11\%$, $P_2 = 18\%$, $P_3 = 23\%$, $P_4 = 48\%$, which add up to 100% (where the % represents the percentage of the overall code). Say P_1 after parallelisation is not sped up, P_2 is sped up by a factor of five, P_3 is sped up by a factor of twenty and P_4 is sped up by a factor of 1.6. Using Amdahl's law, determine the overall speedup of the program after parallelisation. Comment on your result.

Answer:

Assume that the running time of the old computation was 1, for some unit of time. The running time of the new computation will be the length of time the unimproved fraction takes, (which is $(1-Par)$, for a parallel fraction Par), plus the length of time the improved fraction takes. The length of time for the improved part of the computation is the length of the improved part's former running time divided by the speedup, making the length of time of the improved part $Par/Speedup$. The final speedup is computed by dividing the old running time by the new running time: $0.11/1 + 0.18/5 + 0.23/20 + 0.48/1.6 = 0.4575$

or a little less than $\frac{1}{2}$ the original running time which we know is 1. Therefore the overall speed boost is $1 / 0.4575 = 2.186$ or a little more than double the original speed using the formula $(P1/S1 + P2/S2 + P3/S3 + P4/S4)-1$. Notice how the $20\times$ and $5\times$ speedup don't have much effect on the overall speed boost and running time when 11% is not sped up, and 48% is sped up by $1.6\times$.

--[End of Question 1]--

Question 2

[Total marks: 33]

2(a)

[8 marks]

Explain the differences between semaphores and monitors in SR.

2(b)

[10 marks]

Module Code: CA463

Page 3 of 9

Semester 1 Exam

MODERATION STAGE

Show, using SR code, how semaphores may be used to implement monitors and how monitors may be used to implement semaphores.

2(c)

[15 marks]

The Hungry Birds Problem: Given that there is a nest with n baby birds and a parent bird. The baby birds eat out of a common dish that initially contains F portions of food. Each baby repeatedly eats one portion of food at a time, sleeps for a while, and then comes back to eat. When the dish becomes empty, the baby bird which empties the dish awakens the parent bird. The parent refills the dish with F portions, then waits for the dish to become empty again. The pattern repeats forever. Represent the birds as processes and develop code that simulates their actions. Use semaphores for synchronization. Assume the existence of produce() and consume() procedures. (Hint: adapt the Producer-Consumer Solution).

--[End of Question 2]—

Question 3

[Total marks: 33]

3(a)

[9 marks]

Differentiate clearly between Rendezvous and Remote Procedure Call (RPC) in SR, using diagrams where necessary.

Answer:

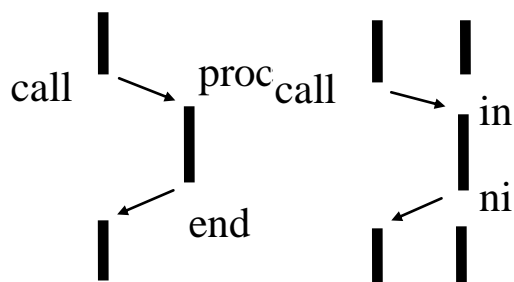
Both RPC & Rendezvous are suited to Client-Server architectures.

The RPC mechanism is simply an intermodule communication mechanism. SR uses the module for RPC. Inside the module we still need to provide synchronisation – unlike rendezvous whereby a synchronisation expression is provided for with the in command and the oldest invocation which makes the synchronisation expression succeed is automatically serviced (see below for the answer to 3(b)). RPC combines aspects of monitors and synchronous message passing:

- * Module exports operations, invoked with call (see diagram).
- * call blocks (delays caller) until serviced.

The rendezvous mechanism combines the actions of servicing a call with the processing of the information conveyed by the call. With rendezvous, a process exports operations that can be called by other processes.

As with RPC, a process can invoke an operation by calling the operation. The key difference between RPC and rendezvous is that the client call to the operation is serviced by an existing server process. The server process rendezvous with the client calling the operation by means of executing an in statement.



3(b)

[9 marks]

What is the syntax of the `in` statement that implements Rendezvous in SR? How are synchronisation and scheduling catered for in Rendezvous?

Answer

In SR rendezvous is accomplished by means of the `in` statement. The general form of the `in` statement is:

```
in operation (formals)
    st sync_expr by sched_expr -> block
[] operation (formals)
    st sync_expr by sched_expr -> block
...
[] else -> block
ni
```

For the `in` statement, the synchronisation expression has been seen already with guarded synchronised message passing. Since the scope of the formals of the operation is the entire guarded operation, the synchronisation and scheduling expression can depend on the value of the formals, and hence on the values of the arguments of the call.

If there is no scheduling expression and several guards are satisfied (the synchronisation expression is true and there is a call to the operation) one of them is chosen nondeterministically. Of the nondeterministically chosen guard, if there are several invocations, and no scheduling expression, the `in` statement services the oldest invocation that makes the guard succeed. If there is a scheduling expression, then the `in` statement services the invocation of the guard which minimises the scheduling expression.

3(c)

[15 marks]

Implement the Readers/Writers problem in SR using rendezvous.

Answer

```
Resource rw()
    Op read_request(); op write_request();
    Op read_release(); op write_release();

Body rw

Process server
    Var nr:=0, nw:=0;

    do true ->
        In read_request() & nw=0 -> nr++
        [] read_release() -> nr-
        [] write_request & (nr=0 and nw=0) -> nw++
        [] write_release() -> nw-
        ni
    od
end server
```

```
process reader(n:=1 to noreaders)
    read_request()
    read_release()
end reader

process writer(n:=1 to noreaders)
    write_request()
    write_release()
end writer

end rw
```

--[End of Question 3]—

Question 4

[Total marks: 33]

4(a)

[10 marks]

Explain the difference between Low Level Concurrency Objects and High Level Concurrency Objects in Java. Why would you use the latter rather than the former?

4(b)

[23 marks]

Implement the Bounded Buffer Class in Java using the Lock and Condition objects, giving code for the `void put(Object x)` and `Object take()` methods.

--[End of Question 4]—

Question 5**[Total marks: 33]**

5(a)

[5 marks]

Compare and contrast MPI & Linda as libraries for concurrent programming.

Answer:

Linda (like MPI) is a machine independent concurrent programming concept. It, in itself, is not a language but an extension to existing sequential languages which provides for concurrent processing eg C is standard C with Linda extensions.

Linda is based on a distributed data concept called the tuple space. The tuple space is a logically shared, associatively addressed memory space into which tuples are placed and removed from. There are two types of tuples:

- i) passive tuples which contain resolved data, and
- ii) active tuples which contain function calls which have yet been evaluated.

When all the elements of an active tuple have been evaluated, the active tuple becomes a passive tuple and its signature changes.

The concept of a tuplespace allows for a radical simplification of distributed computing software. This model is used in the Java platform through the JavaSpaces technology. Applications can run on any Java-enabled machine on the network, and processing is automatically shared by all of the VMs. The mechanism is a simple one: the space acts as a "Job Jar", a shared resource of tasks unfinished and completed.

MPI is a totally different model for concurrent programming, providing a richly-endowed library of commands for message passing between nodes on a concurrent or distributed machine.

5(b)

[10 marks]

Write a short code fragment in C-LINDA for Matrix-Matrix multiplication for two (3 X 3) matrices A and B. Give in your answer a high level description of the algorithm.

Answer

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

int main ( )
{
    int i, x, y, z;
    int a [3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    int bt [3][3] = {1, 0, 2, 1, 2, 1, 1, 1, 1};
    int c_row [3];

    out ("A", 1, a [1]);          /* rows of matrix A */
    out ("A", 2, a [2]);
    out ("A", 3, a [3]);

    out ("B", 1, bt [1]);         /* columns of matrix B */
    out ("B", 2, bt [2]);
    out ("B", 3, bt [3]);

    for (i=1; i <= 3; i++)
        eval ("worker", worker (i));

    for (i =1; i <= 3; i++)
        in ("worker", i);
}
```

```

printf ("Answer is:\n");
for (i = 1; i <= 3; i++)
{
    in ("C", i, ?c_row);
    printf ("%3d, %3d, %3d\n",
            c_row [0], c_row [1], c_row [2]);
}

return 0;
}

int worker (int index)
{
    int i, j;
    int row [3], column [3], result [3];

    rd ("A", index, ?row);

    for (i = 1; i <= 3; i++)
    {
        rd ("B", i, ?column);

        result [i] = 0;
        for (j = 1; j <= 3; j++)
            result [i] += row [j] * column [j];
    }

    out ("C", index, result);
    return (index);
}

```

5(c)

[18 marks]

Implement code for a matrix-vector product routine in MPI for a (4 X 4) Matrix and a (4 X 1) vector. Give in your answer a high level description of the algorithm.

Answer:

```

#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[]) {
int A[4][4], b[4], c[4], line[4], temp[4], local_value, myid;
MPI_Init(&argc, &argv); MPI_Comm_rank(MPI_COMM_WORLD, &myid);
if(myid==0) {
    for(int i=0;i<4;++i) {
        b[i]=4-i;
        for(int j=0; j<4; ++j) {
            A[i][j]=i+j;
        }
    }
    line[0]=A[0][0]; line[1]=A[0][1];
    line[2]=A[0][2]; line[3]=A[0][3];
}
MPI_Bcast(b,4,MPI_INT,0,MPI_COMM_WORLD);
if(myid==0) {
    for(int i=1;i<4;++i) {

```

```

        temp[0]=A[i][0]; temp[1]=A[i][1]; temp[2]=A[i][2];
        temp[3]=A[i][3];
        MPI_Send(temp,4,MPI_INT,i,i,MPI_COMM_WORLD);
        /* no need to send b here */
    }
    else {
        MPI_Recv(line,4,MPI_INT,0,myid, MPI_COMM_WORLD,
        MPI_STATUS_IGNORE);
        /* no need to receive b here */
    }
    c[myid]=line[0]*b[0]+line[1]*b[1]+line[2]*b[2]+line[3]*b[3];
    if(myid!=0) MPI_Send(&c[myid],1,MPI_INT,0,myid,MPI_COMM_WORLD);
    else {
        for(int i=1;i<4;++i)
            MPI_Recv(&c[i],1,MPI_INT,i,i,MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
        } /* Calculating the trace in one command */
    int local_value=line[myid], trace=0;
    MPI_Reduce(&local_value,&trace,1,MPI_INT,MPI_SUM,0,
    MPI_COMM_WORLD);
    MPI_Finalize(); return 0;
}

```

--[End of Question 5]