200

Q1 > 17

ual Processing

ated

Coláiste na hOllscoile
Corcaigh
University College Cork

Uimhir Scrúdaithe
Examination Number | 9 | 1 | 7 | 1 | 6 | 3 | |

Module Code CS4402

Paper No. _____

Mír
Section _____

Q1	10	1	6	X	17
Q2	10	20	10	10	50
					67

Calculator, Please state:	No. of Books submitted
Name	
Model	

Note: **If there are different sections on this paper, a separate Answer Book MUST be used for each section.**

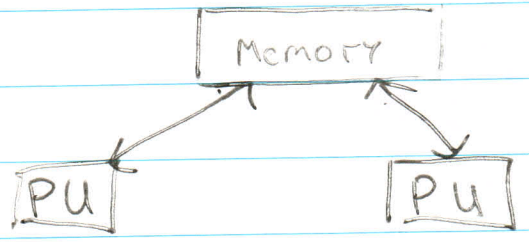
Do na Scrúdaitheoirí amháin **For Examiner's use only**	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
Iomlán **Total**	

Q4 > 17

Q1

A(i) Shared Memory Machine

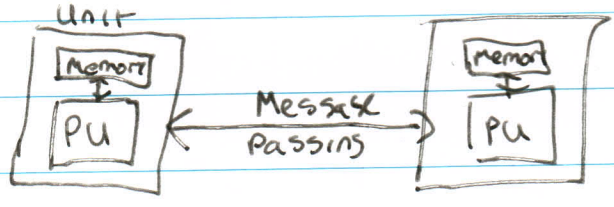
A machine in which each individual processing unit all access a common shared memory



The processing units do not have their own memory

(ii) Distributed Memory Machine

A machine in which each individual processing unit has its own local memory. Data must be shared through message passing



A hybrid machine combines both of these models

Q1 a (ii) SPMD. Single Program Multiple Data streams One OF the sub classifications OF MIMD in which a single program acts on Multiple data streams."

Flynns taxanomy

	Single Instruction Stream	Multiple Instructions
Single Data stream	SISO	MISD
Multiple Data Stream	SIMD	MIMD (SPMD | MPMD)

SPMD

Q1@(ii) Load balancing is the act OF attempting to balance the computational burden OF a multiprocessor system evenly across the individual processors in the system

10pts

Q1

b Gustafson's Law states that;

$$S(n) = n.\text{Serial Part} + (1 - \text{Serial Part})$$ ✓ 1pt

Q1

c. It provides no upperbound on speedup (as you increase the number of processors the speedup also increases

6pts The greater the serial part, the smaller the speedup

Q2 $Q_2 = 50$

A MPI_Bcast. Broadcast the value of of a buffer to all processors within a communication group

```
int MPI_Bcast ( void* buffer, // buffer to send
                int count, // the size of buffer
                MPI_Data type, // the type of the items in the buffer
                int root, // the rank of the proc sending buffer
                MPI_Comm comm_group // comm group of processors taking part in Bcast
);
```

Returns response code, MPI_Success if successfull (as do all the following methods)

MPI Gather. Gather a buffer of elements from many processors to one processor

```
int MPI_Gather ( void* buffer, // Send buffer
                 int count, // Send count
                 MPI_Data type, // type of items in buffer
                 void* buffer // receive buffer
                 int count, // receive count
                 MPI_Data type, // type of receive items
                 int root, // root of proc gathering to
                 MPI_Comm comm_group // group of processor taking part
);
```

MPI_Gatherv gather data from remote processors to a root where the length of each set of items a processor can send varies

```
int MPI_Gatherv ( void* buffer, // send buffer
                  int count, // count of send buffe
                  MPI_Data type, // type of send buffe
                  void * buffer, // receive buffer
                  int count, // count of receive buffe
                  int* displs, // Array instructing how
                  MPI_Data type, // type of receive bu
                  int root, // destination of gather
                  MPI_Comm commgroup // comm group
);                // # of procs performing gather
```

One of MPI_Double MPI_INT etc →

to stitch back the overall array

MPI_Comm_size get the size of a particular comm group

```
int MPI_Comm_size(MPI_Comm comm_group,
                  *int size. // where size will be
);                              placed
```

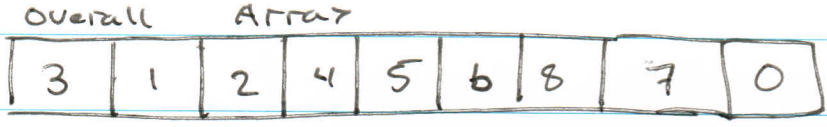
10pt

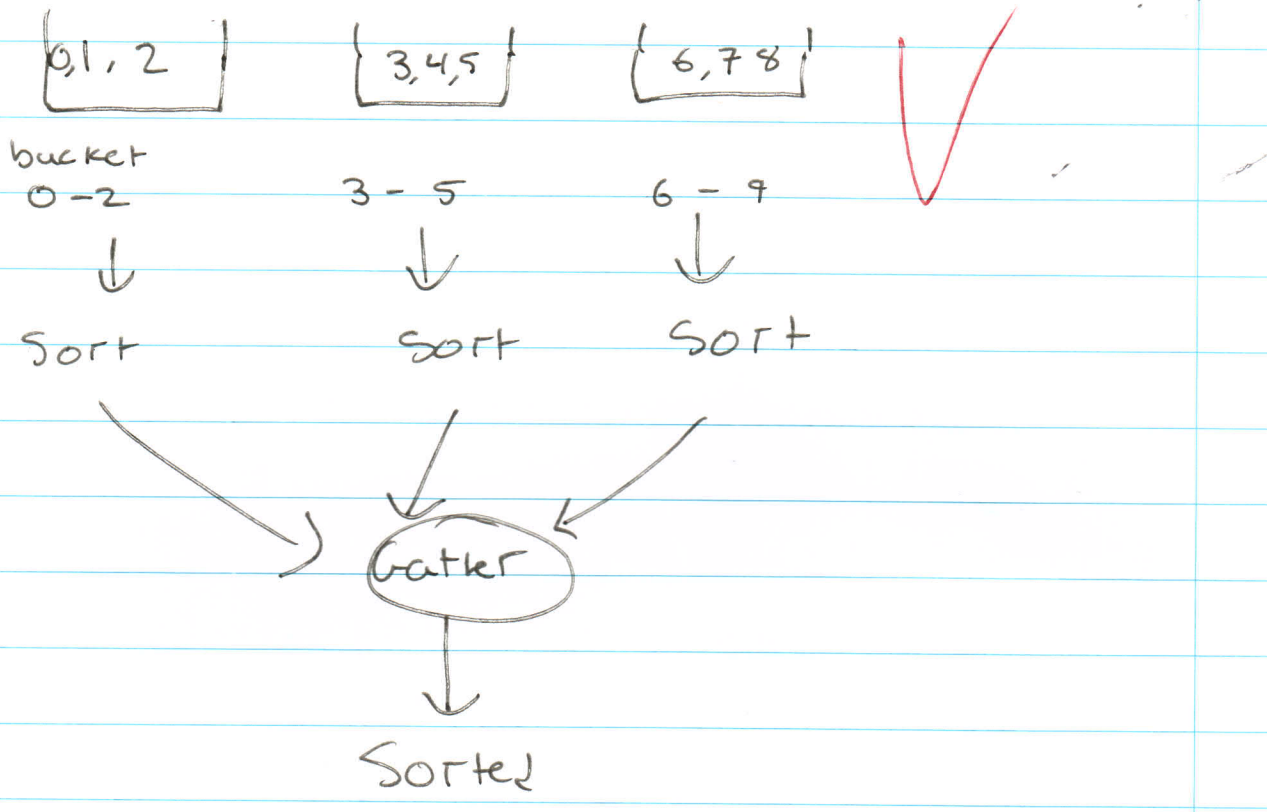
Q2 B

The entire array is broadcast to all Processors in the Sorting group each Processor gets the items for their particular bucket:

Once a Processor gets their bucket it sorts the bucket internally.

The buckets are then gathered up in order into the overall sorted array





Assume exists an M such that
M is max item in list

```
int MPI_Sort (int n, int * a, int root,
              MPI_Comm comm)
{
  int size, rank, i, rc;
  MPI_Comm_size (comm, & size);
  MPI_Comm_rank (comm, & rank);
  // Broadcast entire array to all processors
  rc = MPI_Bcast (a, n, MPI_INT, root, comm);
  IF (rc != MPI_Success) {
      MPI_Abort (0, comm);
  }
  int * bucket = (int *) calloc (n, sizeof(int));
  int bucketCount = 0;
  int bucketRangeLength = M / size;
  for (i = 0; i < n; i++) {
      IF (a[i] > rank * bucketRangeLength &&
          a[i] < (rank + 1) * bucketRangeLength) {
          bucket [bucketCount++] = a [i]; }
  }
  Merge_sort (bucket, bucket_count);
  int * overallBucketCount = (int *) calloc (size, sizeof(int));
rc= MPI_Gather (bucketCount, 1, MPI_INT, overallBucketCount,
                1, MPI_Int, root, comm);
IF (rc!= MPI_Success) {
    MPI_Abort (0, comm);
}
```

Continued on next booklet

200

booklet

Coláiste na hOllscoile
Corcaigh
University College Cork

Uimhir Scrúdaithe
Examination Number 9 1 7 1 6 3

Module Code CS4402

Paper No. _____

Mír
Section _____

Do na Scrúdaitheóirí amháin **For Examiner's use only**	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
Iomlán **Total**	

Calculator, Please state:	No. of Books submitted
Name	2
Model	

Note: If there are different sections on this paper, a separate Answer Book MUST be used for each section.

Continued From Previous booklet

```
int * displs ;
IF (rank = root){
    displs = (int *) calloc ( size, sizeof(int));
    displs[0] = 0;
    For (i = 1; i ≤ size; i ++) {
        displs[i] = displs[i-1] + overallBucketCount[i-1];
    }
}

rc= MPI_Gatherv (&bucket, bucketCount, MPI_INT,
                 ra, n, &displs, MPI_INT, root, comm);

return rc;
}
```

20pts

Q2

c $\text{Speedup} = \frac{T(1)}{T(n)}$

$T(1) = n^2$

$T(n) = T_{comm}(n) + T_{compute}(n)$

$T_{comm}(n) =$

T_{comm} For Bcast $= n.T_{comm}$

T_{comm} For bucketCount gather $= size.T_{comm}$

T_{comm} For gatherv $= n.T_{comm}$

$\Rightarrow T_{comm}(n) = 2n + size.T_{comm}$

$T_{compute}(n) =$

$T_{compute}$ Fill buckets $= n.T_{compute}$

$T_{compute}$ Sort interal buckets $= n/size \log n/size$

From assumption of similarly sized buckets sorting using normal merge sort

$T_{compute}$ displacements $= size.T_{compute}$

$\Rightarrow T_{compute}(n) = n + \frac{n}{size} \log \frac{n}{size} + size.T_{compute}$

$\Rightarrow T(n) = 2n + size.T_{comm} + n + \frac{n}{size} \log \frac{n}{size} + size.T_{compute}$

$\Rightarrow \text{Speedup} = \frac{n^2}{2n + size.T_{comm} + n + \frac{n}{size} \log \frac{n}{size} + size.T_{compute}}$

10 pts

Q2

D Negative Facts

1 entire array must be broadcast to all processors taking part in sort

⇒ This increases communication complexity substantially

2 Each processor must iterate the entire array to find the contents of its buckets.

⇒ increases linearly with length of array

3 A processor does not know how many items are going to be in its bucket. this means it must allocate a bucket of size n

⇒ increases space complexity

10pts

4 An upper and lower bound must be known on the contents of the array, in this case it was assumed m and 0.

5 = Not all arrays are good candidates for bucket sort. The must be evenly distributed

eg [1, 200042, 200043, 200000, ...]

2d Positive Facts

1 It has a low computation complexity and this reduces well as the size of the sorting group is increased

2 It has minimal sequential parts making it a good candidate for speedup