UNIVERSITY OF OSLO

Faculty of mathematics and natural sciences

Examination in	INF3380 — Parallel programming for scientific problems
Day of examination:	June 9th, 2015
Examination hours:	9.00 – 13.00
This problem set consists of 4 pages.	
Appendices:	None
Permitted aids:	Calculator One double-sided A4-sheet with handwritten notes

Please make sure that your copy of the problem set is complete before you attempt to answer anything.

Weighting of the problems

Problem 1: 20%
Problem 2: 20%
Problem 3: 10%
Problem 4: 25%
Problem 5: 25%

Problem 1 (weight 20%)

1a (weight 10%)

Please define *speedup* in connection with parallel computing.

1a { Speedup = $\frac{T_s}{T(p)}$ ← serial time (Ts), ← parallel time (T(p))
* It's important to mention that Ts is the best possible serial algorithm.

1b (weight 10%)

There are many reasons why perfect speedup is seldom achievable in practice, please describe two of them.

1b { Two typical reasons:
① load imbalance
② Communication overhead

Problem 2 (weight 20%)

2a (weight 5%)

What is the intended result of doing an *all-to-all* broadcast involving p processors?

start: M_0 on P_0, M_1 on P_1, M_2 on P_2 . . . M_{p-1} on P_{-1} → all-to-all → result: $M_0, M_1, M_2 \cdots M_{p-1}$ on all P processors

(Continued on page 2.)

2b (weight 10%)

If the p processors are arranged as a ring, please explain (step by step) how the all-to-all broadcast can be carried out by making use of one-to-one communications.

see Fig 4.9 in the textbok

2c (weight 5%)

Assuming that the cost model for sending a message of m words from one processor to another is

$$t_s + t_w m,$$

where t_s and t_w are two constants. Please derive the cost model of carrying out an all-to-all broadcast on a ring of p processors, each initially has m words as its own data.

Total p-1 steps.
Each step costs ts+twm
Total cost: (p-1)·(ts+twm)

Problem 3 (weight 10%)

What will be written to the screen by the following OpenMP program using 4 threads, and why?

```
int main (int argc, char *argv[])
{
  int   i, n;
  float a[100], sum;

  n = 100;
  for (i=0; i < n; i++)
    a[i] = i * 1.0;
  sum = 0.0;

#pragma omp parallel for default(shared) reduction(+:sum)
  for (i=0; i < n; i++)
    sum = sum + a[i];

  printf("   Sum = %f\n",sum);
  return 0;
}
```

result. $\sum_{i=0}^{99} i = 4950$
(no matter how many threads are used, actually)

Problem 4 (weight 25%)

The following definition is about a matrix-vector multiplication $y = A * x$, where A is a $n \times n$ matrix, x and y are both vectors of dimension $n \times 1$.

$$y_i = a_{i,1}x_1 + a_{i,2}x_2 + \ldots + a_{i,n}x_n, \quad i = 1, 2, \ldots, n.$$

(Continued on page 3.)

4a (weight 5%)

Please write a serial C function

```
void mat_vec(int n, double** A, double* x, double* y)
```

that implements $y = A * x$.

4a
```
int i,j;
for (i=0; i<n; i++) {
  double tmp = 0.0;
  for (j=0; j<n; j++) {
    tmp += A[i][j] * x[j];
  }
  y[i] = tmp;
}
```

4b (weight 5%)

Parallelize the above serial C function with help of OpenMP.

4b Add # omp pragma parallel for private(j) before the i-loop

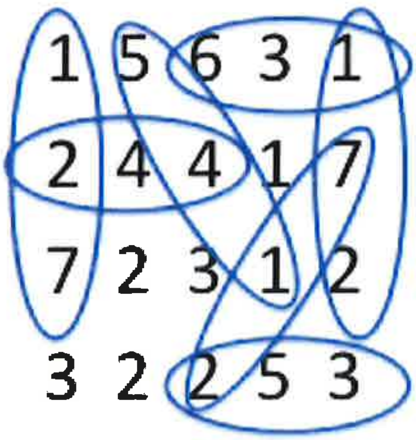
4c (weight 15%)

Sketch an MPI implementation (show only the important details), where initially only process 0 has values of matrix A and vector x. In the end, the entire y vector should be available on process 0.

4c MPI_Scatterv, individual comp; MPI_Gatherv

Problem 5 (weight 25%)

Given a 2D table v that contains $M \times N$ positive integers, we want to count the total number of "triple-friends of 10", that is, three consecutive numbers (in the horizontal, vertical, or diagonal directions) that sum up to 10. The following picture shows some examples of "triple-friends of 10", which are marked by circles.



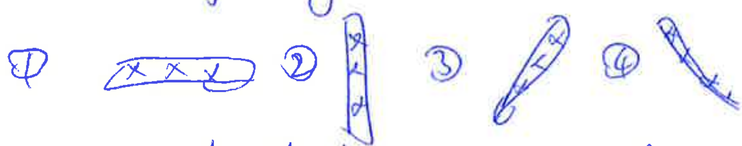
5a (weight 10%)

Write a serial C function

```
int count_friends_of_ten (int M, int N, int** v)
```

that returns the total number of "triple-friends of 10" inside the 2D $M \times N$ table v.

Important: four possible ways of forming "triple-friends"



Also: special attention on the boundary.

(Continued on page 4.)

5b (weight 10%)

Sketch an MPI program (show only the important details), assuming that only process 0 has values of v in the beginning.

For simplicity: 1D partitioning
* MPI_Scatterv (with overlap); * local counting; * MPI_Allreduce

5c (weight 5%)

Derive a formula for the time usage of the parallel implementation, and then derive the associated isoefficiency function. (For simplicity, you can now assume $M = N$.)

* Assume initially process 0 has the entire array v.
* Assume serial time usage $N^2 \cdot c$, where c is a constant representing the cost of check the 4 possible "triple-friends" pr. point.
* Cost of MPI_Scatter (each processor receives roughly $\frac{N^2}{p}$)

$$t_s \log p + t_w \frac{N^2}{p}(p-1)$$

* Cost of local computation

$$\frac{N^2}{p} \cdot c$$

* Cost of MPI_Reduce

$$(t_s + t_w) \cdot \log p$$

$$\rightarrow T(p) \simeq 2t_s \log p + t_w(N^2 + \log p) + \frac{N^2}{p} \cdot c$$

$$T_0 = p \cdot T(p) - T_s = 2t_s p \cdot \log p + t_w p N^2 + t_w p \log p$$

The overhead grows too fast; therefore not possible to maintain efficieny