Exam Parallel Programming 12 April 2007
Department of Computer Science, Faculty of Sciences

1. How is the *efficiency* of a parallel program defined?

2. Given below is the pseudo-code for a parallel algorithm that tries to solve the All-pairs Shortest Paths (ASP) problem, i.e. it computes the shortest route between any two cities, where the lengths of the direct routes betweens the cities are given by a NxN matrix C. The send/receive primitives used are FIFO-ordered, so messages between two nodes always arrive in the order they were sent.

   Explain why this algorithm is incorrect and may sometimes compute incorrect results, depending on the timing behavior of the program.

```
int lb, ub;        /* lower/upper bound for this CPU */
int rowK[N], C[lb:ub, N];     /* pivot row ; matrix */

for (k = 1; k <= N; k++) {
   if (k >= lb && k <= ub) {          /* do I have it? */
      rowK = C[k,*];
      for (p = 1; p <= nproc; p++) /* broadcast row */
         if (p != myprocid) SEND(p, rowK);
   } else
      RECEIVE_FROM_ANY(&rowK);        /* receive row */
   for (i = lb; i <= ub; i++)      /* update my rows */
      for (j = 1; j <= N; j++)
         C[i,j] = MIN(C[i,j], C[i,k] + rowK[j]);
}
```

3. Explain why the usage of asynchronous message passing may lead to the problem of buffer overflows, while synchronous message passing does not have this problem.

4. Explain how a select statement can be used to implement a server process with the following property. The server contains an integer variable X (initialized to zero) and accepts two different types of messages, to increase resp. decrease the value of X. For both messages, the server returns a message to the sender with the new value of X. The value of X, however, should always be between zero and ten. Messages that try to decrease X below zero or increase it above ten should therefore not be serviced immediately but should be delayed until X has an appropriate value. Implement this server process with a select statement, using syntax from SR, Ada, or (clearly explained) pseudo-code.

5. Many iterative parallel programs have to decide at the end of each iteration whether the program should terminate or continue. Each process therefore has to check whether its part of the computation satisfies the given convergence criterion. Next, the processes

have to communicate to check if *all* processes satisfy the criterion; if so, the program terminates, else it continues.

This behavior can be expressed using two procedures:

```
Vote(int YesOrNo);    /* vote whether to terminate (1=yes, 0=no) */
int AwaitDecision(); /* outcome of the voting (1=stop, 0=continue) */
```

Each process first calls Vote, indicating whether or not it wants to terminate and then calls AwaitDecision. The latter procedure gives the outcome of the voting process: it returns 1 if all processes want to terminate, and 0 otherwise. The procedure blocks until the outcome of the votes is known.

Show how these procedures can be implemented using Linda's Tuple Space. For the Tuple Space operations, indicate clearly which parameters are actuals and which are formals. Try to come up with a simple solution, with a small number of lines of code. Make sure that the procedure AwaitDecision returns as soon as the outcome of the votes is clear (i.e., one process votes 'no', or all processes have voted 'yes'). The procedures need to work correctly for only 1 iteration; the number of processes (P) is known and fixed.

6. Consider the following (synthetic) HPF program

```
program hpfprogram
  real s, X(100), Y(100) ! s is scalar, X and Y are arrays
  !HPF$ PROCESSORS P(4)
  !HPF$ ALIGN X(:) WITH Y(:)
  !HPF$ ALIGN  s WITH Y(*)
  !HPF$ DISTRIBUTE Y(BLOCK) ONTO P

  X = X * 3.0        ! Multiply each X(i) by 3.0
  do i = 2,99
    Y(i) = X(i-1) + X(i+1)
  enddo
  s = SUM(X)  ! Add all X(i) values
end
```

   (a) Explain how an HPF compiler will parallelize this program

   (b) Explain which communication statements the HPF compiler will generate; assume that the compiler generates MPI calls and tries to make optimal use of the MPI primitives.

7. Consider the following four different parallel search algorihms:
   1 IDA* (a search algorithm based on work-stealing) without a shared transposition table
   2 IDA* with a shared replicated transposition table
   3 IDA* with a shared partitioned transposition table
   4 Transposition-Driven Search (TDS)

(a) The four algorithms differ in the number of search-nodes they analyse (expand and evaluate). Rank the four algorithms in order of increasing number of nodes searched and explain your ordering.

(b) The four algorithms also have different communication overheads for handling transposition table lookups and stores. Discuss for each algorithm from what type of communication overhead it suffers.

(c) Why does TDS by far obtain the best speedup of the four algorithms?

8. Explain why divide-and-conquer parallelism is a good model for programming computational grids. Why do divide-and-conquer systems like Satin obtain nearly the same speedup on a 64-node local cluster as on a wide-area grid with 4 clusters of 16 nodes (64 nodes in total), despite the fact that the wide-area network between the clusters is about 1000 times slower than the local network (e.g., Myrinet) used within a cluster?

Points

1	2	3	4	5	6a	6b	7a	7b	7c	8
10	10	10	10	15	5	5	5	5	5	10

Total: 90 (+ 10 = 100)