Process farming

```
--  This chapter uses about 'number.of.workers' processes to compute
--  the effect of
--
--    SEQ i = 0 FOR N
--      result[i] := f(task[i]))
--
--  in a way that could be executed about 'number.of.workers' times
--  more quickly on that many processors.

--
--  sizing constants
--

VAL INT N IS ... :            -- the number of independent tasks

VAL INT number.of.workers IS ... :
                              -- the number of tasks executed at once

REAL64 FUNCTION f(VAL REAL64 x) IS ... :
                              -- the task to be performed N times

VAL [N]REAL64 task IS ... :   -- the N items of data to be f(.)'d

[N]REAL64 result :            -- and the output variable
```

```
--
--  protocols
--

PROTOCOL TASK.STREAM
  CASE
    new.task; INT; REAL64
    no.more.tasks
:

PROTOCOL RESULT.STREAM
  CASE
    new.result; INT; REAL64
    no.more.results
:

PROTOCOL ADDR.TASK.STREAM
  CASE
    new.task.packet; INT; INT; REAL64
    no.more.task.packets
:

PROTOCOL ADDR.RESULT.STREAM
  CASE
    new.result.packet; INT; INT; REAL64
    no.more.result.packets
:

--
--  worker: processes a sequence of tasks
--

PROC worker(CHAN OF TASK.STREAM task.in,
            CHAN OF RESULT.STREAM result.out )
  BOOL more.tasks :
  SEQ
    more.tasks := TRUE
    WHILE more.tasks
      task.in ? CASE
        INT index :
        REAL64 task :
        new.task; index; task
          REAL64 result :
          SEQ
            result := f(task)
            result.out ! new.result; index; result
        no.more.tasks
          more.tasks := FALSE
    result.out ! no.more.results
:
```

```
--
-- foremen
--
--   task.foreman divides a stream of addressed tasks into a stream
--     of tasks for its worker, and a stream of addressed tasks for
--     the rest of the farm
--   result.foreman combines a stream of results from its worker
--     with a stream of addressed results from the rest of the farm
--

PROC task.foreman(CHAN OF ADDR.TASK.STREAM from.farmer,
                  CHAN OF TASK.STREAM to.worker,
                  CHAN OF ADDR.TASK.STREAM to.farm     )
  BOOL more.tasks :
  SEQ
    more.tasks := TRUE
    WHILE more.tasks
      from.farmer ? CASE
        INT addr, index :
        REAL64 task :
        new.task.packet; addr; index; task
          IF
            addr = 0
              to.worker ! new.task; index; task
            addr > 0
              to.farm ! new.task.packet; addr - 1; index; task
        no.more.task.packets
          more.tasks := FALSE
    to.worker ! no.more.tasks
    to.farm ! no.more.task.packets
:
```

```
PROC result.foreman(CHAN OF ADDR.RESULT.STREAM from.farm,
                    CHAN OF RESULT.STREAM from.worker,
                    CHAN OF ADDR.RESULT.STREAM to.farmer)
  BOOL more.from.worker, more.from.farm :
  SEQ
    more.from.worker, more.from.farm := TRUE, TRUE
    WHILE more.from.worker OR more.from.farm
      ALT
        more.from.worker & from.worker ? CASE
          INT index :
          REAL64 task :
          new.result; index; task
            to.farmer ! new.result.packet; 0; index; task
          no.more.results
            more.from.worker := FALSE
        more.from.farm & from.farm ? CASE
          INT addr, index :
          REAL64 result :
          new.result.packet; addr; index; result
            to.farmer ! new.result.packet; addr + 1; index; result
          no.more.result.packets
            more.from.farm := FALSE
    to.farmer ! no.more.result.packets
:

--
--  work.detail runs a worker and a smaller farm
--

PROC work.detail(CHAN OF ADDR.TASK.STREAM from.farmer, to.farm,
                 CHAN OF ADDR.RESULT.STREAM from.farm, to.farmer )
  CHAN OF TASK.STREAM to.worker :
  CHAN OF RESULT.STREAM from.worker :
  PAR
    task.foreman(from.farmer, to.worker, to.farm)
    worker(to.worker, from.worker)
    result.foreman(from.farm, from.worker, to.farmer)
:
--
--  farmer's boy implements a farm consisting of no workers
--

PROC farmers.boy(CHAN OF ADDR.TASK.STREAM from.farm,
                 CHAN OF ADDR.RESULT.STREAM to.farm )
  PAR
    from.farm ? CASE no.more.task.packets
    to.farm ! no.more.result.packets
:
```

```
--
--  farmer distributes addressed tasks to a fixed-sized farm
--

PROC farmer(VAL []REAL64 task,
                         CHAN OF ADDR.TASK.STREAM out,
                         CHAN OF ADDR.RESULT.STREAM in,
                                                []REAL64 result )
  [number.of.workers]BOOL idle :
  INT begun, completed :
  SEQ
    SEQ i = 0 FOR number.of.workers
      idle[i] := TRUE
    begun, completed := 0, 0
    WHILE completed < (SIZE task)
      ALT
        INT addr, index :
        in ? CASE new.result.packet; addr; index; result[index]
          completed, idle[addr] := completed + 1, TRUE
        (begun < (completed + number.of.workers)) AND
                                    (begun < (SIZE task)) & SKIP
          IF addr = 0 FOR number.of.workers
            idle[addr]
              SEQ
                out ! new.task.packet; addr; begun; task[begun]
                begun, idle[addr] := begun + 1, FALSE
    out ! no.more.task.packets
    in ? CASE no.more.result.packets
:


--
--  the structure of the farm
--

[number.of.workers + 1]CHAN OF ADDR.TASK.STREAM outgoing :
[number.of.workers + 1]CHAN OF ADDR.RESULT.STREAM incoming :
PAR
  farmer(task, outgoing[0], incoming[0], result)
  PAR i = 0 FOR number.of.workers
    work.detail(outgoing[i], outgoing[i+1],
                incoming[i+1], incoming[i] )
  farmers.boy(outgoing[number.of.workers],
              incoming[number.of.workers] )
```