Parallel matrix multiplier

```
--  This program applies an affine transformation
--
--    v = a . u + k
--    ~   ~   ~   ~
--  to an infinite sequence of vectors, u, overlapping
--  the multiplications of the components.

--
--  the multiplier cell
--

PROC multiplier(VAL REAL32 aij,
                CHAN OF REAL32 north, south, west, east )
  REAL32 uj,  aij.times.uj,  vi :
  SEQ
    north ? uj
    WHILE TRUE
      SEQ
        PAR
          south ! uj
          aij.times.uj := aij * uj
          west ? vi
        PAR
          east ! vi + aij.times.uj
          north ? uj
:
```

```
--
--  the other component processes of the array
--

PROC offset(VAL REAL32 ki, CHAN  OF REAL32 east)
  WHILE TRUE
    east ! ki
:

PROC sink(CHAN OF REAL32 north)
  WHILE TRUE
    REAL32 discard :
    north ? discard
:

--
--  configuration constants and interface procedures
--

VAL INT n IS 3 :        -- size of the matrix

VAL [n][n]REAL32 a IS   -- example value for the matrix
       [[ 0.980085(REAL32), -0.14112(REAL32),  -0.139708(REAL32) ],
        [-0.139708(REAL32), -0.989992(REAL32),  0.0199149(REAL32)],
        [-0.14112(REAL32) ,  0.0(REAL32),      -0.989992(REAL32) ]] :

VAL [n]REAL32 k IS      -- example value of the offset
        [ 0.1(REAL32),       2.0(REAL32),      30.0(REAL32)] :

PROC produce.u([n]CHAN OF REAL32 out)
  --
  --  produce an infinite sequence of vectors
  --
  ... omitted code
:

PROC consume.v([n]CHAN OF REAL32 in)
  --
  --  consume an infinite sequence of result vectors
  --
  ... omitted code
:
```

```
--
--  structure of the program
--

[n+1][n]CHAN OF REAL32 north.south, west.east :
PAR
  --
  --  producer of co-ordinates u[j]
  --
  produce.u(north.south[0])
  --
  -- the matrix multiplier array
  --
  PAR
    PAR i = 0 FOR n
      offset(k[i], west.east[0][i])
    PAR i = 0 FOR n
      PAR j = 0 FOR n
        multiplier(a[i][j],
                   north.south[i][j], north.south[i+1][j],
                   west.east  [j][i], west.east  [j+1][i] )
    PAR j = 0 FOR n
      sink(north.south[n][j])
  --
  --  consumer of co-ordinates v[i]
  --
  consume.v(west.east[n])
```