Parallel matrix multiplier

```
PROC produce.xj(VALUE j, CHAN south)
  -- north row: source of x values
PROC consume.yi(VALUE i, CHAN east)
  -- west column: sink for y values
PROC offset(VALUE ki, CHAN west)
  -- east column: source of k offsets
PROC multiplier(VALUE aij, CHAN north, south, west, east)
  -- middle: responsible for a values
PROC sink(CHAN north)
  -- south row: sink for unused outputs
SEQ
  -- initialize a and k
  PAR
    PAR j = [0 FOR n]
      produce.xj(j, ...)
    PAR i = [0 FOR n]
      offset(k[i], ...)
    PAR i = [0 FOR n]
      PAR j = [0 FOR n]
        multiplier( ... )
    PAR j = [0 FOR n]
      sink(...)
    PAR i = [0 FOR n]
      consume.yi(i, ...)
```

```
PROC produce.xj(VALUE j, CHAN south) = -- north row: source x values
  WHILE TRUE
    south ! ANY                        :



PROC consume.yi(VALUE i, CHAN east) =  -- west column: sink y values
  WHILE TRUE
    east ? ANY                       :



PROC offset(VALUE ki, CHAN west) =     -- east column: source k offsets
  WHILE TRUE
    west ! ki                        :



PROC multiplier(VALUE aij, CHAN north, south, west, east) =
  VAR xj, aij.times.xj, yi :           -- responsible for a values
  SEQ
    north ? xj
    WHILE TRUE
      SEQ
        PAR
          south ! xj
          aij.times.xj := aij * xj
          east ? yi
        PAR
          west ! yi + aij.times.xj
          north ? xj                                     :



PROC sink(CHAN north) =                -- south row
  WHILE TRUE                           --   sink for unused outputs
    north ? ANY       :
```

```
DEF n = 3 :
VAR a[n * n],  k[n] :
SEQ
  -- initialize a and k

  CHAN north.south[n * (n + 1)],  east.west[n * (n + 1)] :
  PAR
    PAR j = [0 FOR n]
      produce.xj(j, north.south[j])

    PAR i = [0 FOR n]
      offset(k[i], east.west[(n * n) + i])
    PAR i = [0 FOR n]
      PAR j = [0 FOR n]
        multiplier( a[(n * i) + j],
                    north.south [(n * i) + j],
                    north.south [(n * (i + 1)) + j],
                    east.west   [i + (n * j)],
                    east.west   [i + (n * (j + 1))] )
    PAR j = [0 FOR n]
      sink(north.south[(n * n) + j])

    PAR i = [0 FOR n]
      consume.yi(i, east.west[i])
```