```
type point = rec(double x, y); double G = 6.67e-11;
point p[1:n], v[1:n], f[1:PR,1:n];  # position, velocity,
double m[1:n];              # force and mass for each body
```
initialize the positions, velocities, forces, and masses;

```
procedure barrier(int w) { # efficient barrier from Section 3.4 }

# calculate forces for bodies assigned to worker w
procedure calculateForces(int w) {
  double distance, magnitude; point direction;
  for [i = w to n by PR, j = i+1 to n] {
    distance = sqrt( (p[i].x - p[j].x)**2 +
                     (p[i].y - p[j].y)**2 );
    magnitude = (G*m[i]*m[j]) / distance**2;
    direction = point(p[j].x-p[i].x, p[j].y-p[i].y);
    f[w,i].x = f[w,i].x + magnitude*direction.x/distance;
    f[w,j].x = f[w,j].x - magnitude*direction.x/distance;
    f[w,i].y = f[w,i].y + magnitude*direction.y/distance;
    f[w,j].y = f[w,j].y - magnitude*direction.y/distance;
} }

# move the bodies assigned to worker w
procedure moveBodies(int w) {
  point deltav;   # dv = f/m * DT
  point deltap;   # dp = (v + dv/2) * DT
  point force = (0.0, 0.0);
  for [i = w to n by PR] {
    # sum the forces on body i and reset f[*,i]
    for [k = 1 to PR] {
      force.x += f[k,i].x; f[k,i].x = 0.0;
      force.y += f[k,i].y; f[k,i].y = 0.0;
    }
    deltav = point(force.x/m[i] * DT, force.y/m[i] * DT);
    deltap = point( (v[i].x + deltav.x/2) * DT,
                    (v[i].y + deltav.y/2) * DT);
    v[i].x = v[i].x + deltav.x;
    v[i].y = v[i].y + deltav.y;
    p[i].x = p[i].x + deltap.x;
    p[i].y = p[i].y + deltap.y;
    force.x = force.y = 0.0;
} }

process Worker[w = 1 to PR] {
  # run the simulation with time steps of DT
  for [time = start to finish by DT] {
    calculateForces(w);
    barrier(w);
    moveBodies(w);
    barrier(w);
} }
```

Figure 11.11 Shared variable program for the *n*-body problem.