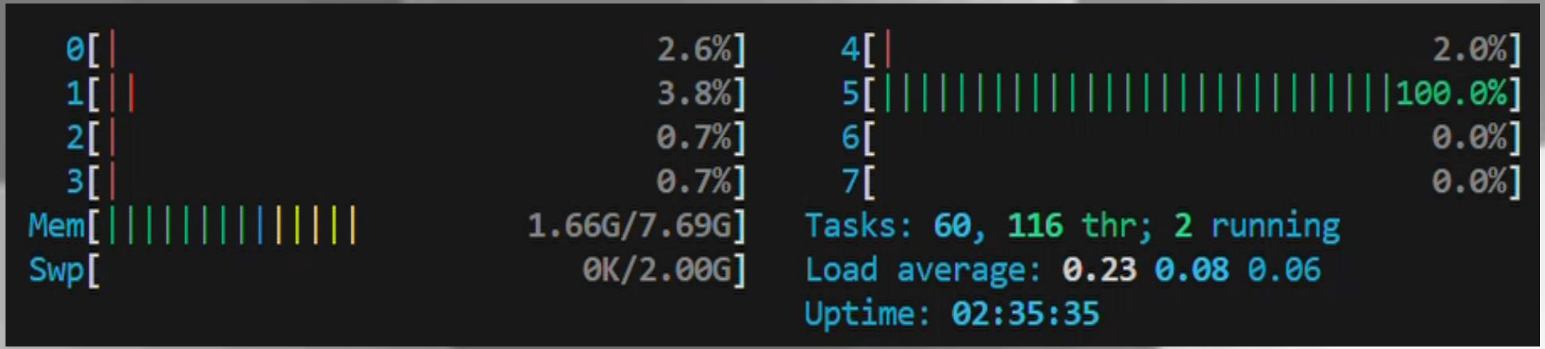
CPU OPITMIZATION

GPU OPTIMIZATION

PLUMMER SPHERE SIMULATION

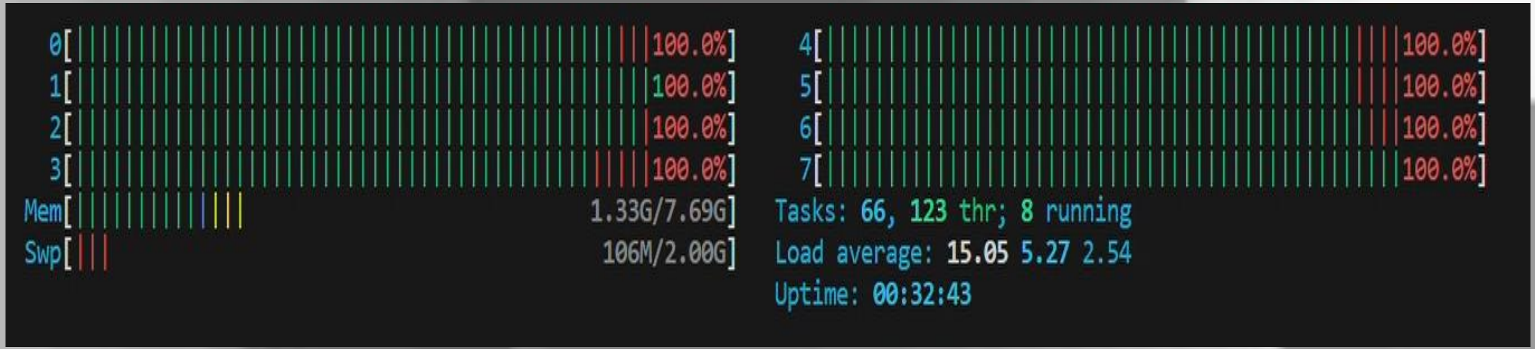
CPU OPTIMIZATION



```
    0[|                          2.6%]   4[|                          2.0%]
    1[||                         3.8%]   5[|||||||||||||||||||||||||||100.0%]
    2[|                          0.7%]   6[                           0.0%]
    3[|                          0.7%]   7[                           0.0%]
  Mem[||||||||||||||       1.66G/7.69G]  Tasks: 60, 116 thr; 2 running
  Swp[                       0K/2.00G]  Load average: 0.23 0.08 0.06
                                        Uptime: 02:35:35
```

BEFORE OPTIMIZATION

1 CORE ONLY

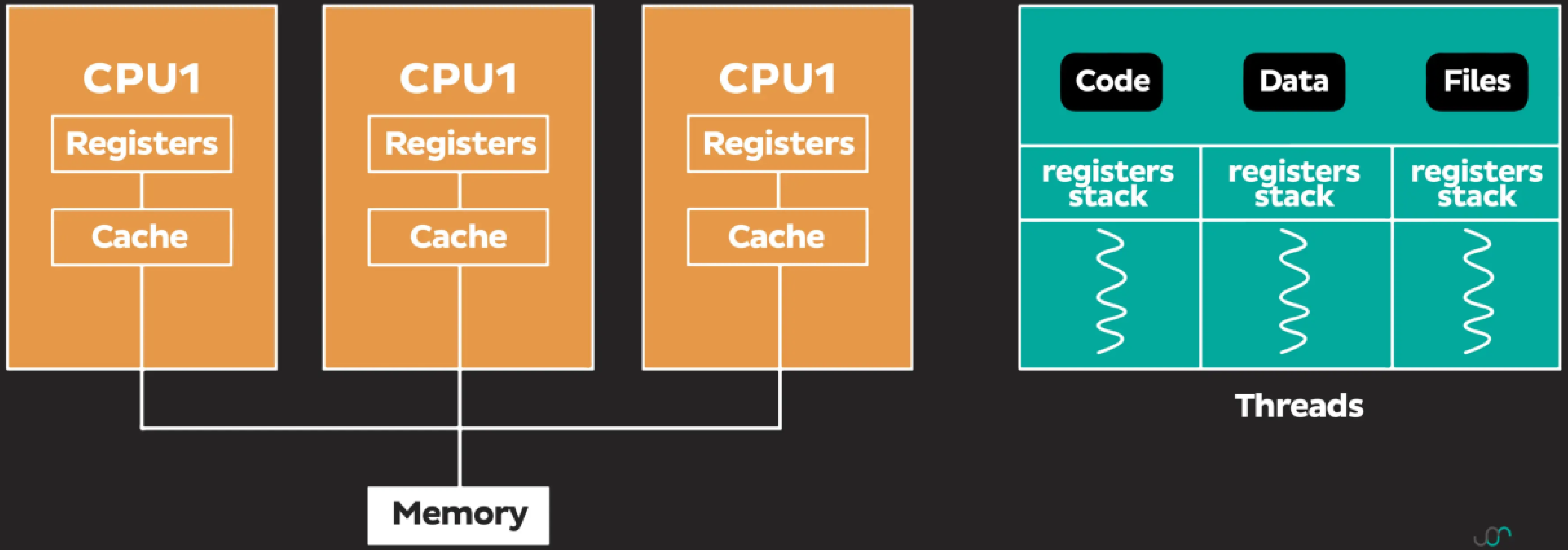


```
   0[|||||||||||||||||||||||||||||||||||||||||||||||100.0%]   4[||||||||||||||||||||||||||||||||||||||||||||||100.0%]
   1[|||||||||||||||||||||||||||||||||||||||||||||||100.0%]   5[||||||||||||||||||||||||||||||||||||||||||||||100.0%]
   2[|||||||||||||||||||||||||||||||||||||||||||||||100.0%]   6[||||||||||||||||||||||||||||||||||||||||||||||100.0%]
   3[|||||||||||||||||||||||||||||||||||||||||||||||100.0%]   7[||||||||||||||||||||||||||||||||||||||||||||||100.0%]
 Mem[|||||||||||||                         1.33G/7.69G]  Tasks: 66, 123 thr; 8 running
 Swp[|||                                   106M/2.00G]  Load average: 15.05 5.27 2.54
                                                        Uptime: 00:32:43
```

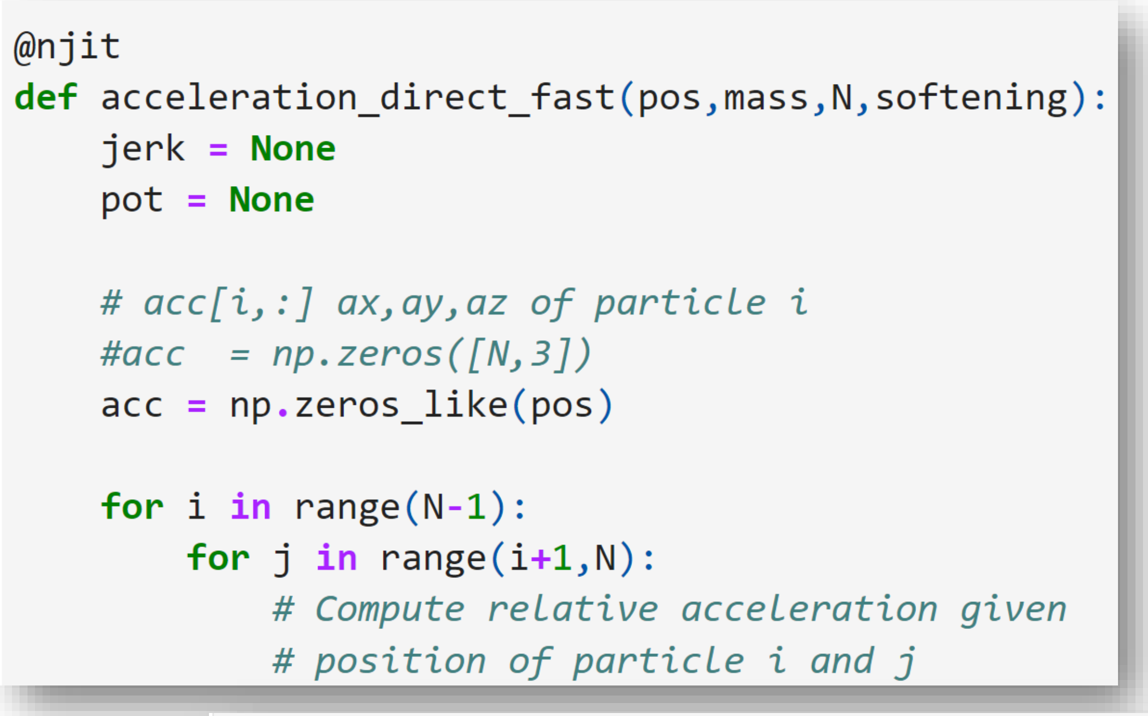
AFTER OPTIMIZATION

ALL CORES USED

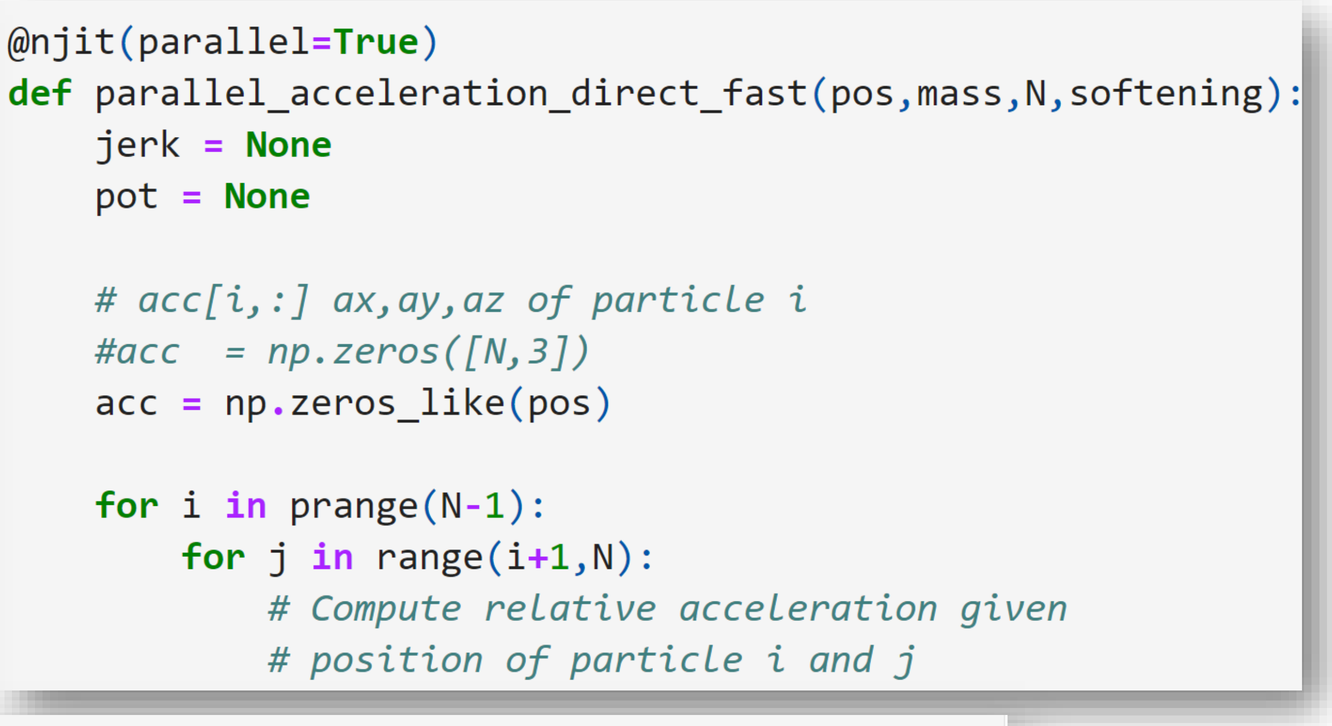
multiprocessing vs multithreading



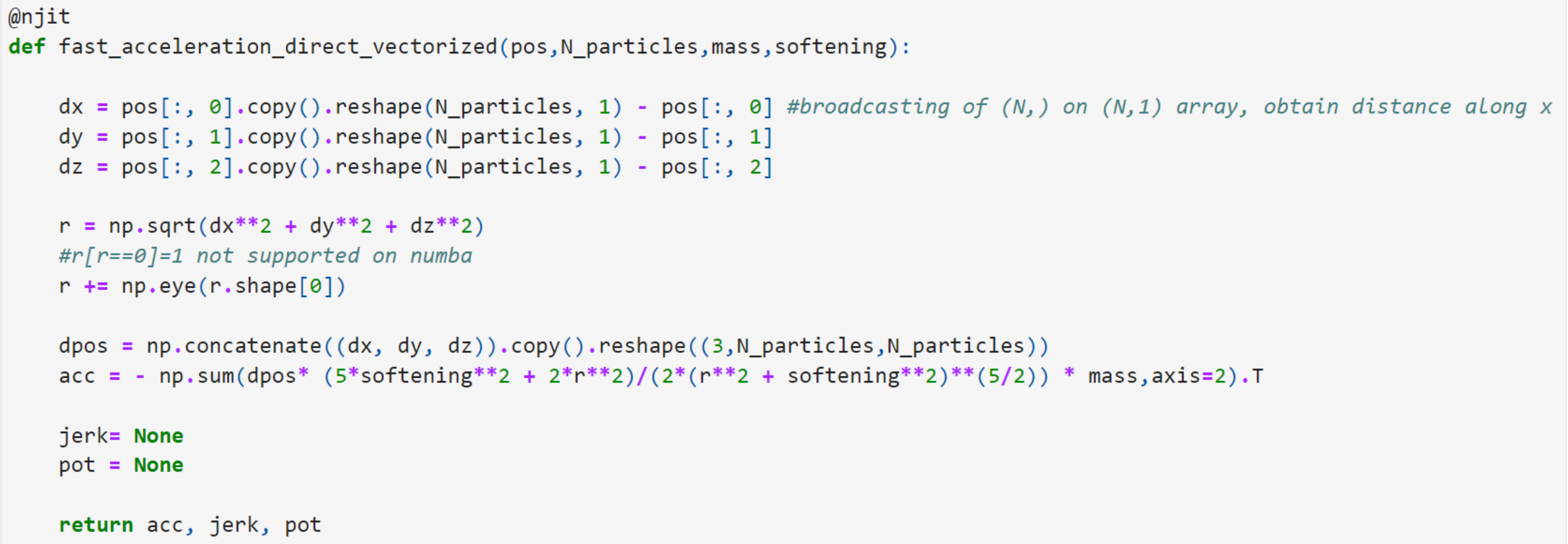
NUMBA



```
@njit
def acceleration_direct_fast(pos,mass,N,softening):
    jerk = None
    pot = None

    # acc[i,:] ax,ay,az of particle i
    #acc  = np.zeros([N,3])
    acc = np.zeros_like(pos)

    for i in range(N-1):
        for j in range(i+1,N):
            # Compute relative acceleration given
            # position of particle i and j
```

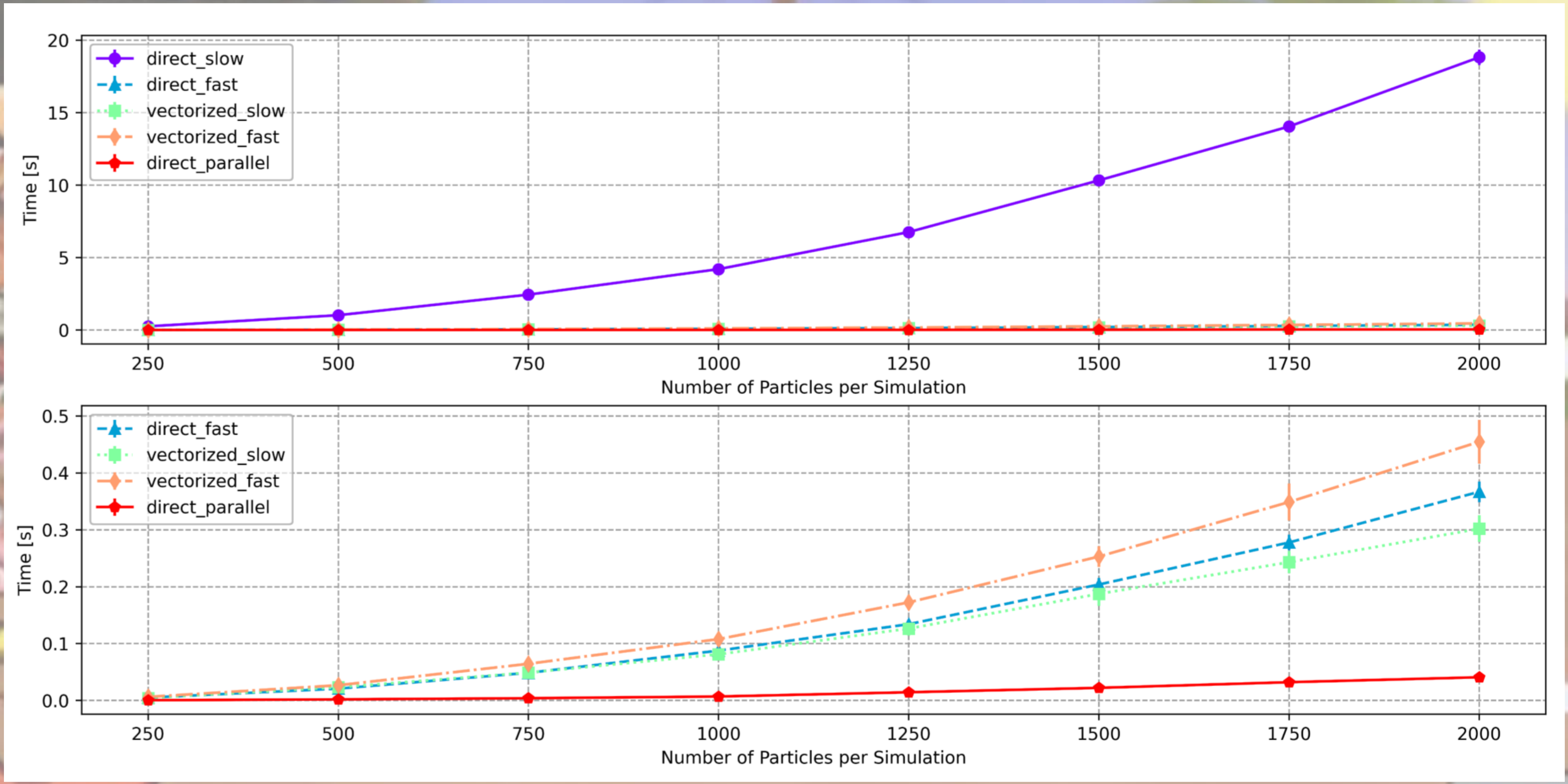


```
@njit(parallel=True)
def parallel_acceleration_direct_fast(pos,mass,N,softening):
    jerk = None
    pot = None

    # acc[i,:] ax,ay,az of particle i
    #acc  = np.zeros([N,3])
    acc = np.zeros_like(pos)

    for i in prange(N-1):
        for j in range(i+1,N):
            # Compute relative acceleration given
            # position of particle i and j
```

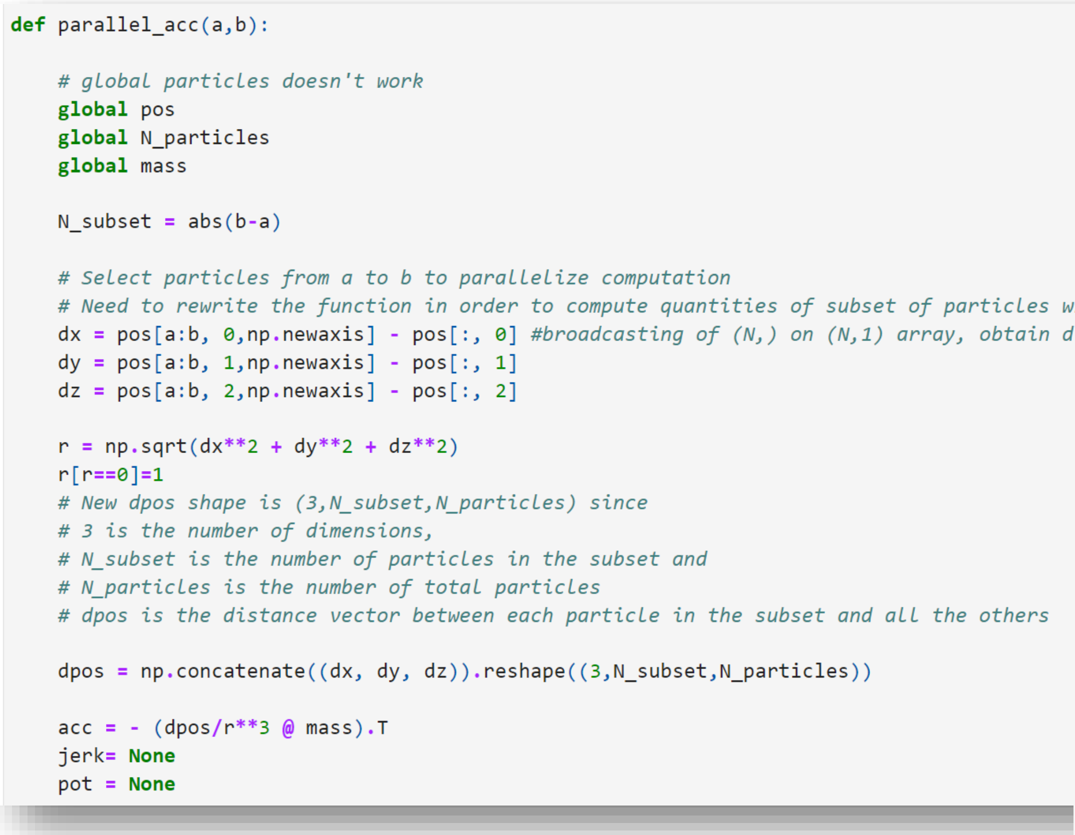


```
@njit
def fast_acceleration_direct_vectorized(pos,N_particles,mass,softening):

    dx = pos[:, 0].copy().reshape(N_particles, 1) - pos[:, 0] #broadcasting of (N,) on (N,1) array, obtain distance along x
    dy = pos[:, 1].copy().reshape(N_particles, 1) - pos[:, 1]
    dz = pos[:, 2].copy().reshape(N_particles, 1) - pos[:, 2]

    r = np.sqrt(dx**2 + dy**2 + dz**2)
    #r[r==0]=1 not supported on numba
    r += np.eye(r.shape[0])

    dpos = np.concatenate((dx, dy, dz)).copy().reshape((3,N_particles,N_particles))
    acc = - np.sum(dpos* (5*softening**2 + 2*r**2)/(2*(r**2 + softening**2)**(5/2)) * mass,axis=2).T

    jerk= None
    pot = None

    return acc, jerk, pot
```

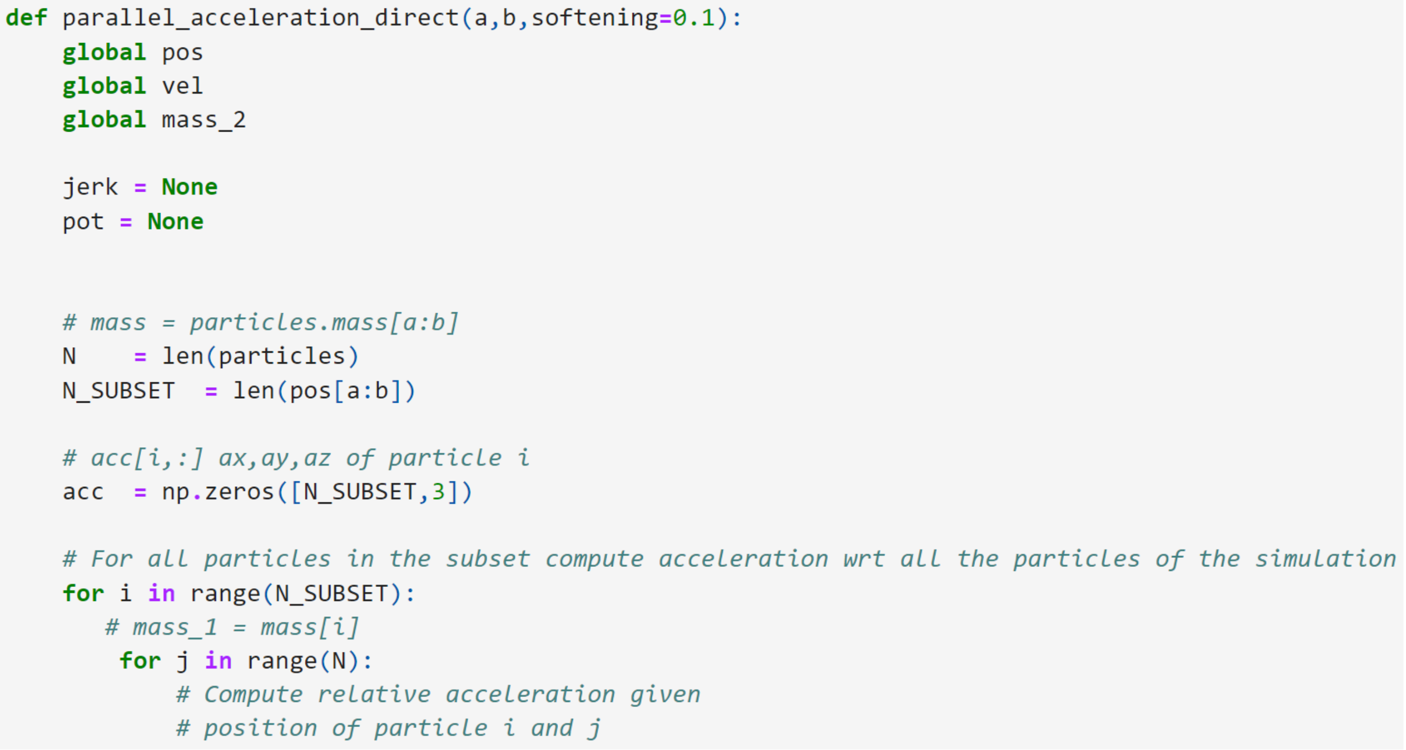
NUMBA ACCELERATION



MULTIPROCESSING

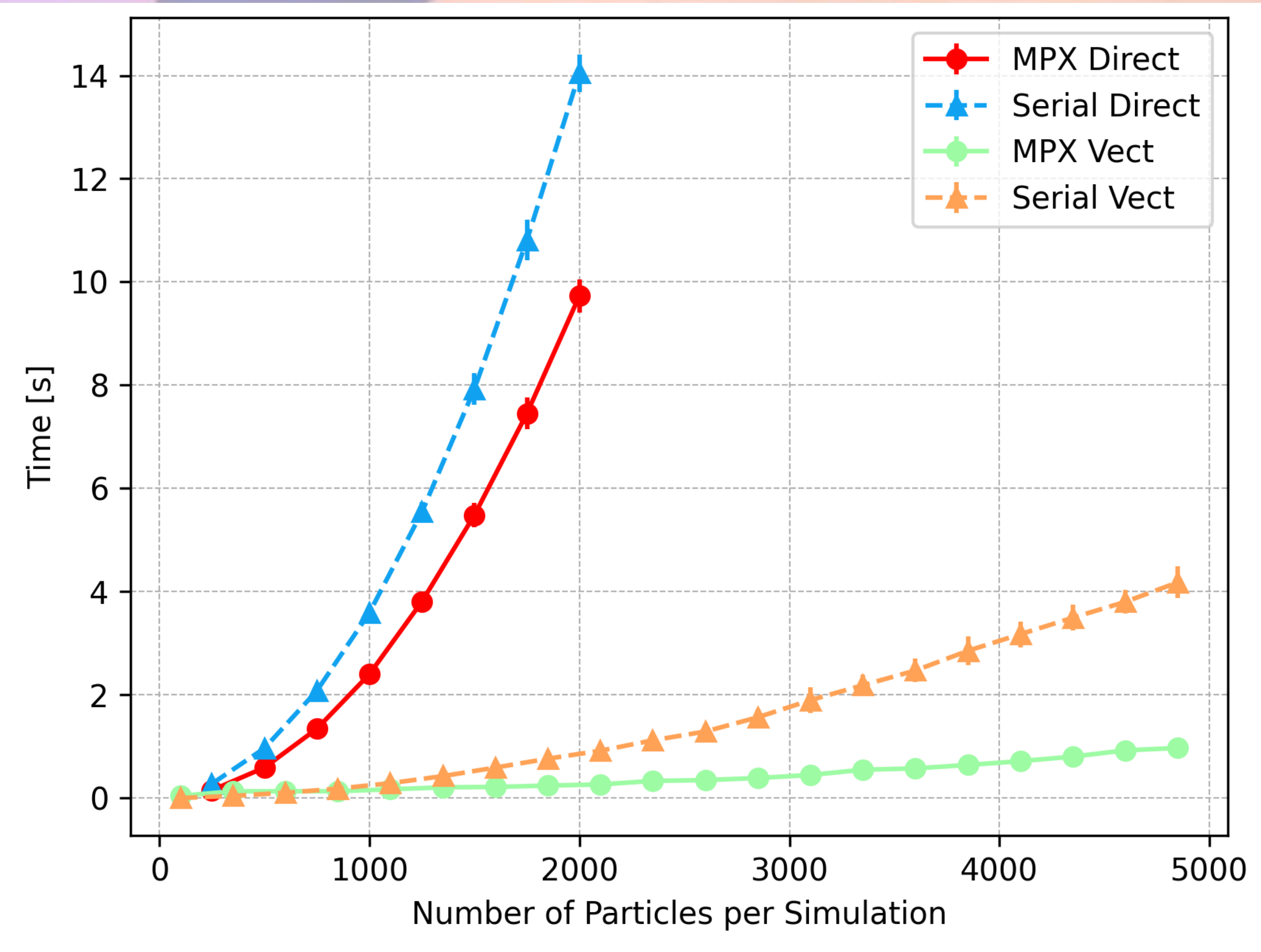


```python
def parallel_acc(a,b):

    # global particles doesn't work
    global pos
    global N_particles
    global mass

    N_subset = abs(b-a)

    # Select particles from a to b to parallelize computation
    # Need to rewrite the function in order to compute quantities of subset of particles wrt all the others
    dx = pos[a:b, 0,np.newaxis] - pos[:, 0] #broadcasting of (N,) on (N,1) array, obtain distance along x in an (N,N) matrix
    dy = pos[a:b, 1,np.newaxis] - pos[:, 1]
    dz = pos[a:b, 2,np.newaxis] - pos[:, 2]

    r = np.sqrt(dx**2 + dy**2 + dz**2)
    r[r==0]=1
    # New dpos shape is (3,N_subset,N_particles) since
    # 3 is the number of dimensions,
    # N_subset is the number of particles in the subset and
    # N_particles is the number of total particles
    # dpos is the distance vector between each particle in the subset and all the others

    dpos = np.concatenate((dx, dy, dz)).reshape((3,N_subset,N_particles))

    acc = - (dpos/r**3 @ mass).T
    jerk= None
    pot = None
```

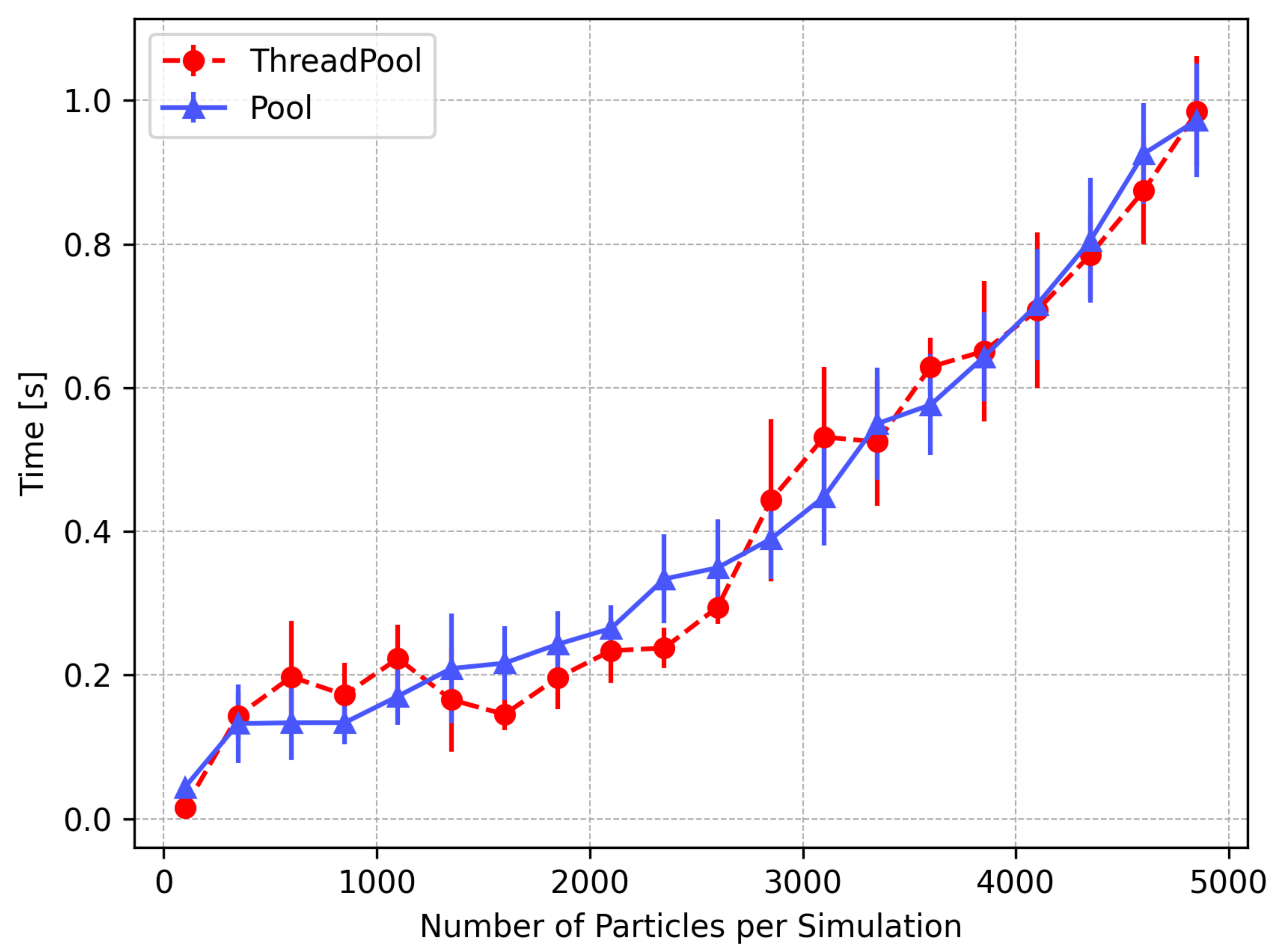


```python
def parallel_acceleration_direct(a,b,softening=0.1):
    global pos
    global vel
    global mass_2

    jerk = None
    pot = None


    # mass = particles.mass[a:b]
    N     = len(particles)
    N_SUBSET  = len(pos[a:b])

    # acc[i,:] ax,ay,az of particle i
    acc  = np.zeros([N_SUBSET,3])

    # For all particles in the subset compute acceleration wrt all the particles of the simulation
    for i in range(N_SUBSET):
       # mass_1 = mass[i]
        for j in range(N):
            # Compute relative acceleration given
            # position of particle i and j
```

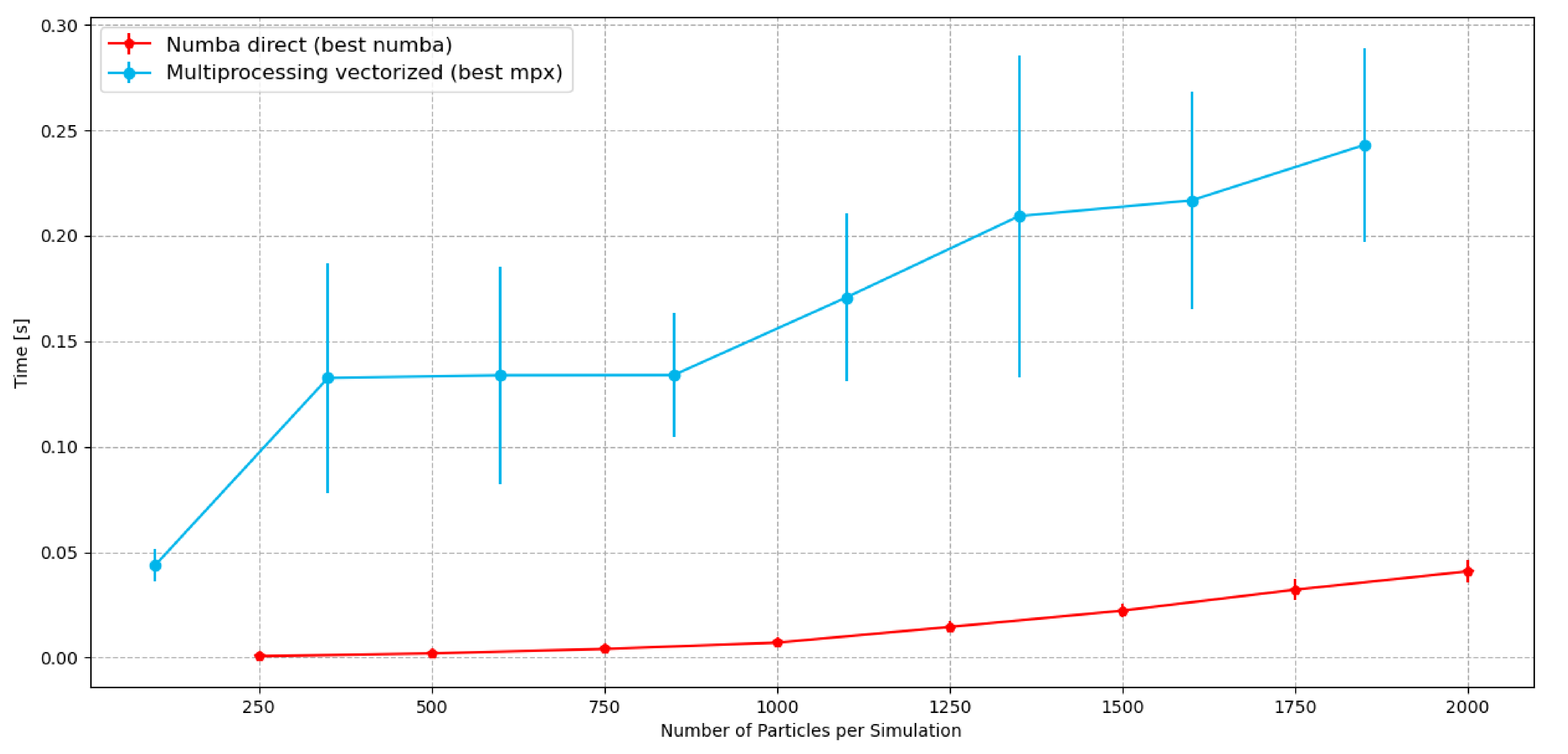
MULITPROCESSING SINGLE EVOLUTION, PARALLEL COMPUTATION



POOL VS. THREADPOOL



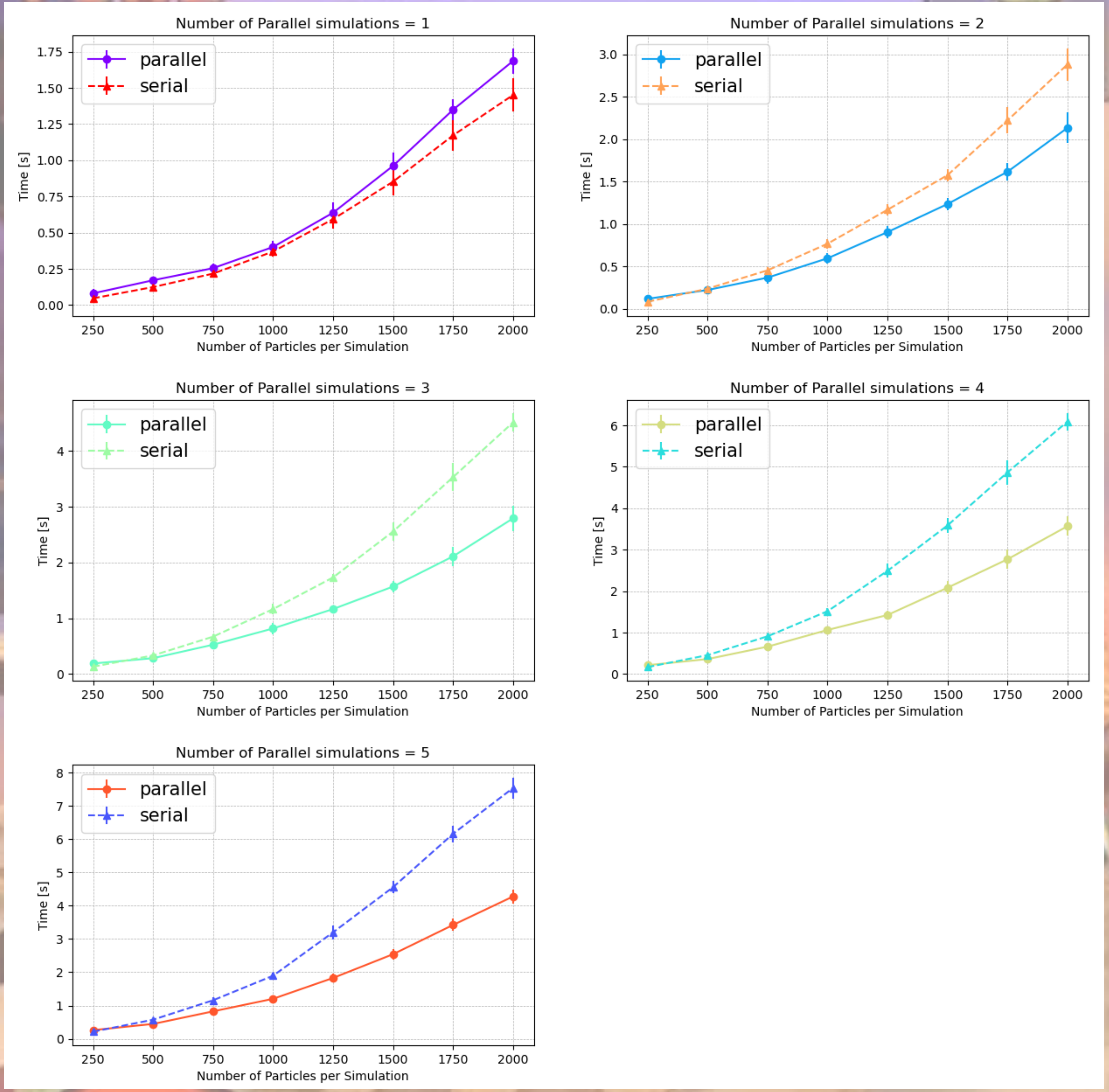
NUMBA VS MULTIPROCESSING - SINGLE EVOLUTION



MULTIPROCESSING

PARALLEL EVOLUTIONS

SERIAL ACCELERATION ESTIMATE

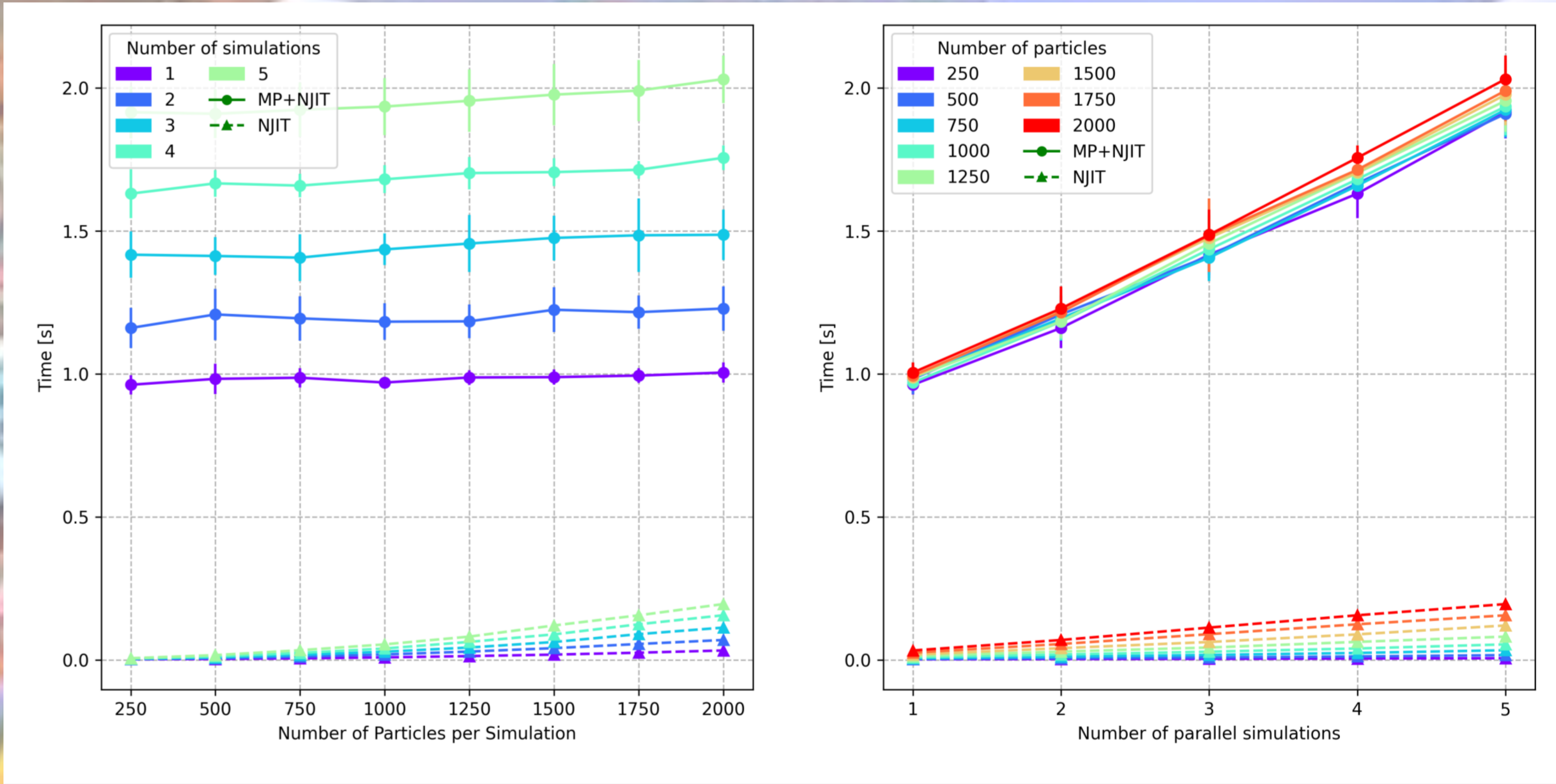


MULTIPROCESSING

PARALLEL EVOLUTIONS

SERIAL ACCELERATION ESTIMATE

USING NUMBA NJIT ON PARALLEL SIMULATIONS


Number of simulations
MP+NJIT
NJIT
Time [s]
Number of Particles per Simulation
Number of particles
MP+NJIT
NJIT
Time [s]
Number of parallel simulations