Multiple Particle Simulation

Parallel Functional Programming

Pavan Ravindra (UNI: phr2114)

Molecular Dynamics Simulations

Update rules (velocity Verlet):

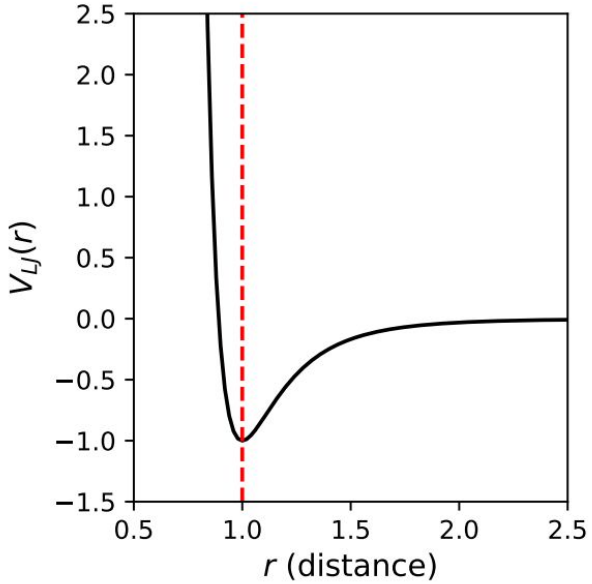
$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i(t) + \Delta t \mathbf{v}_i(t) + \frac{\Delta t^2}{2m_i}\mathbf{F}_i(t)$$

$$\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t) + \frac{\Delta t}{2m_i}[\mathbf{F}_i(t) + \mathbf{F}_i(t+\Delta t)]$$

Force Calculation:

$$\mathbf{F}_i(t) = \sum_{i \neq j} \mathbf{F}_{ij}(t)$$

Lennard-Jones Potential:



$$V_{LJ}(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

Molecular Dynamics Simulations

Update rules (velocity Verlet):

$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i(t) + \Delta t \mathbf{v}_i(t) + \frac{\Delta t^2}{2m_i}\mathbf{F}_i(t)$$

$$\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t) + \frac{\Delta t}{2m_i}[\mathbf{F}_i(t) + \mathbf{F}_i(t+\Delta t)]$$

Force Calculation:

$$\mathbf{F}_i(t) = \sum_{i \neq j} \mathbf{F}_{ij}(t)$$

Algorithm for *N* interacting point particles:

1. <u>Generate initial conditions:</u>
   - Specify $\mathbf{r}_i(0)$ and $\mathbf{v}_i(0)$ for all particles
   - For our purposes: *O(N)*

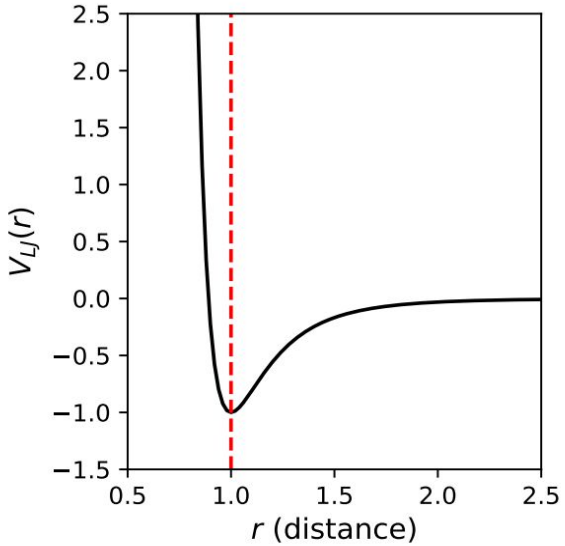
2. <u>Iteratively update positions and velocities:</u>
   - Compute forces between every pair of particles: ~N^2
   - Need to do this for all *T* timesteps: $O(N^2 T)$

Melting a Lennard-Jones Crystal

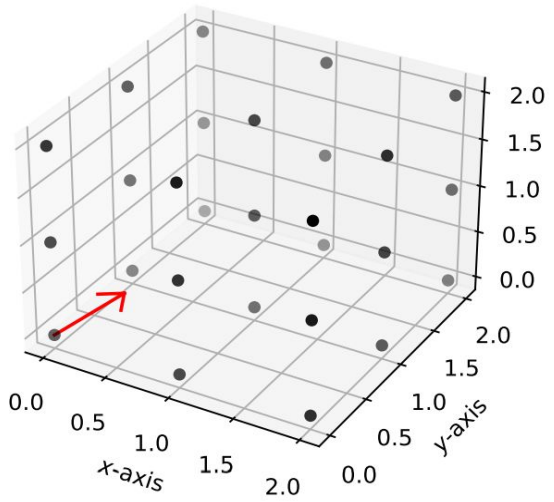
$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i(t) + \Delta t \mathbf{v}_i(t) + \frac{\Delta t^2}{2m_i}\mathbf{F}_i(t)$$

$$\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t) + \frac{\Delta t}{2m_i}[\mathbf{F}_i(t) + \mathbf{F}_i(t+\Delta t)]$$

Lennard-Jones Potential:

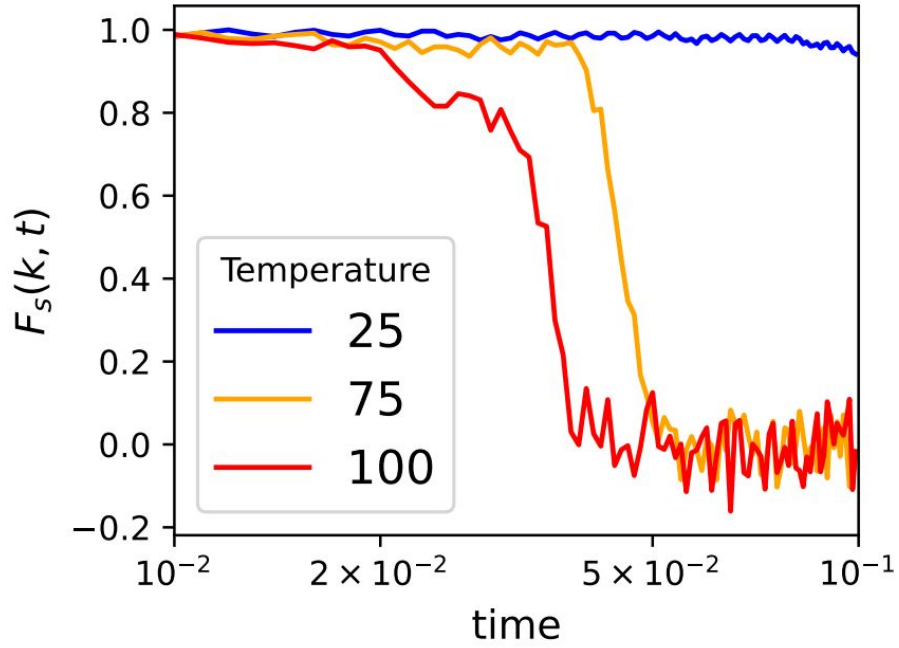


Lennard-Jones Cubic Crystal:



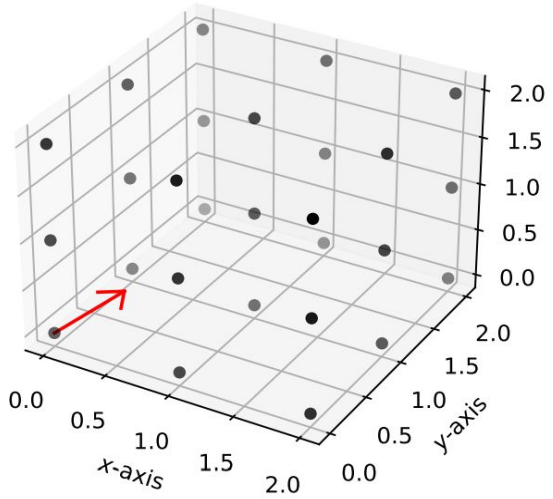
Periodic Boundary Conditions!

Temperature Analysis (Sanity Check)



Self-intermediate scattering function:

$$F_s(k,t) = \sum_{i=1}^{N} e^{i\mathbf{k}\cdot(\mathbf{r}_i(t)-\mathbf{r}_i(0))}$$

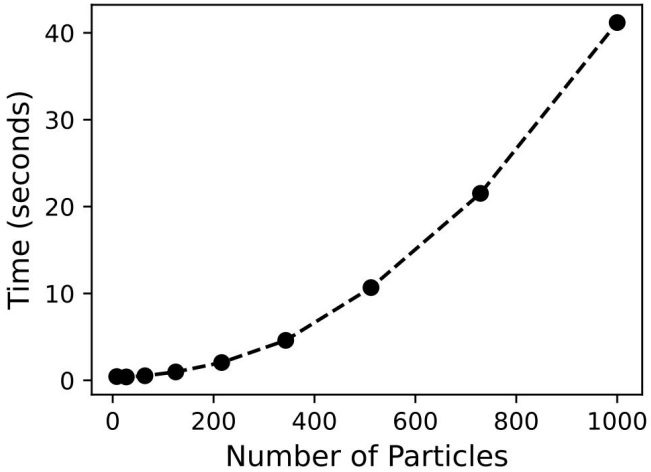


Parallelizing the Calculation

- Recall: the force calculation is the computational bottleneck: ~$O(N^2)$
- Involves calculating the force between every pair of particles
- Overall plan: implement the force calculation as a single `map` call:
  - Then we can trivially parallelize the calculation of the force vector $\mathbf{F}_i(t)$ for each particle i

$$\mathbf{F}_i(t) = \sum_{i \neq j} \mathbf{F}_{ij}(t)$$

Empirical Runtime Scaling:

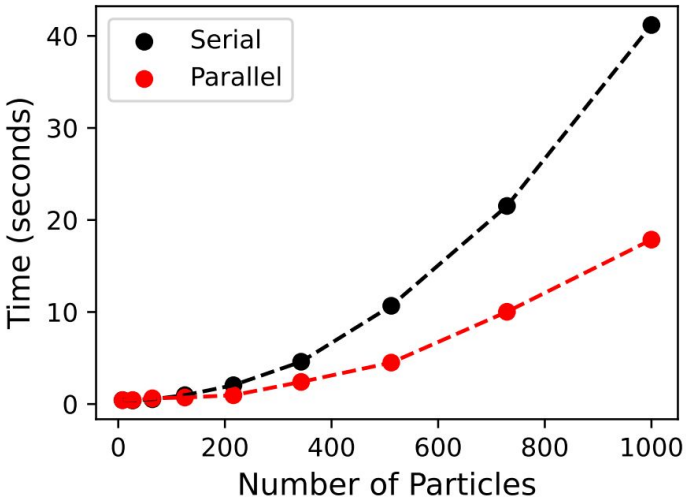


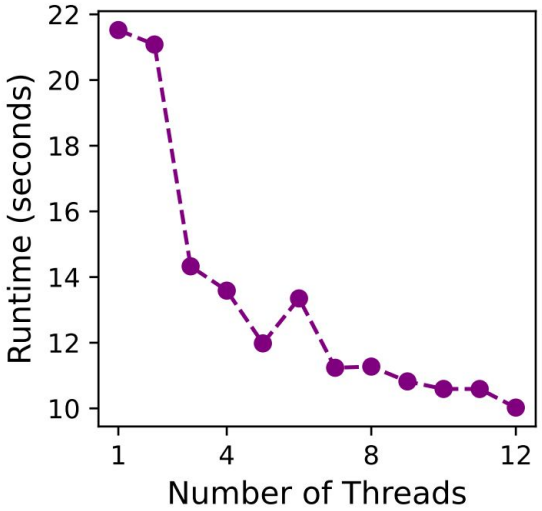
Force Calculation

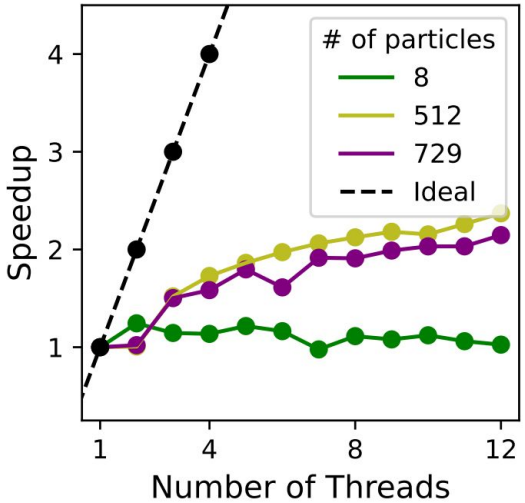
```haskell
data MDVector = MDVector !Double !Double !Double
```

```haskell
-- Computes list of forces on all particles given a configuration
forceMatrix :: [MDVector] -> Double -> [MDVector]
forceMatrix rs boxLength =
  map totalForce rs `using` parList rseq
  where
    -- Gets force acting on particle at r1 due to particle at r2
    forceVector r1 r2
      | r1 == r2 = zeroVector
      | otherwise = vectorMultiply flj (unitVector r12)
        where r12 = displacement r2 r1 boxLength
              d12 = vectorNorm r12
              sor = sigma / d12
              flj = 24.0 * epsilon * (2 * (sor ** 12.0) - (sor ** 6.0)) / d12
    -- Computes total force on particle at r due to all other particles
    totalForce r = foldr vectorAdd zeroVector $ map (forceVector r) rs
```

Speedup Overview



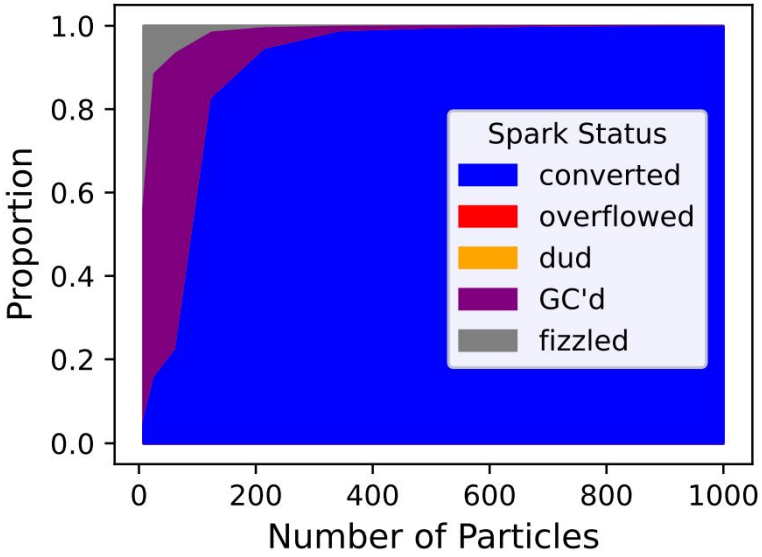




- Parallel implementation helps the most for large system sizes
- Speedup isn’t quite ideal but still improves consistently!

Spark outcomes

- For large system sizes: almost all sparks are converted :)
- Never have any issues with spark pool overflowing :)
- Parallelism is less efficient for small system sizes (as expected)
- GC’d vs. fizzled seems strange… runtime system to blame?

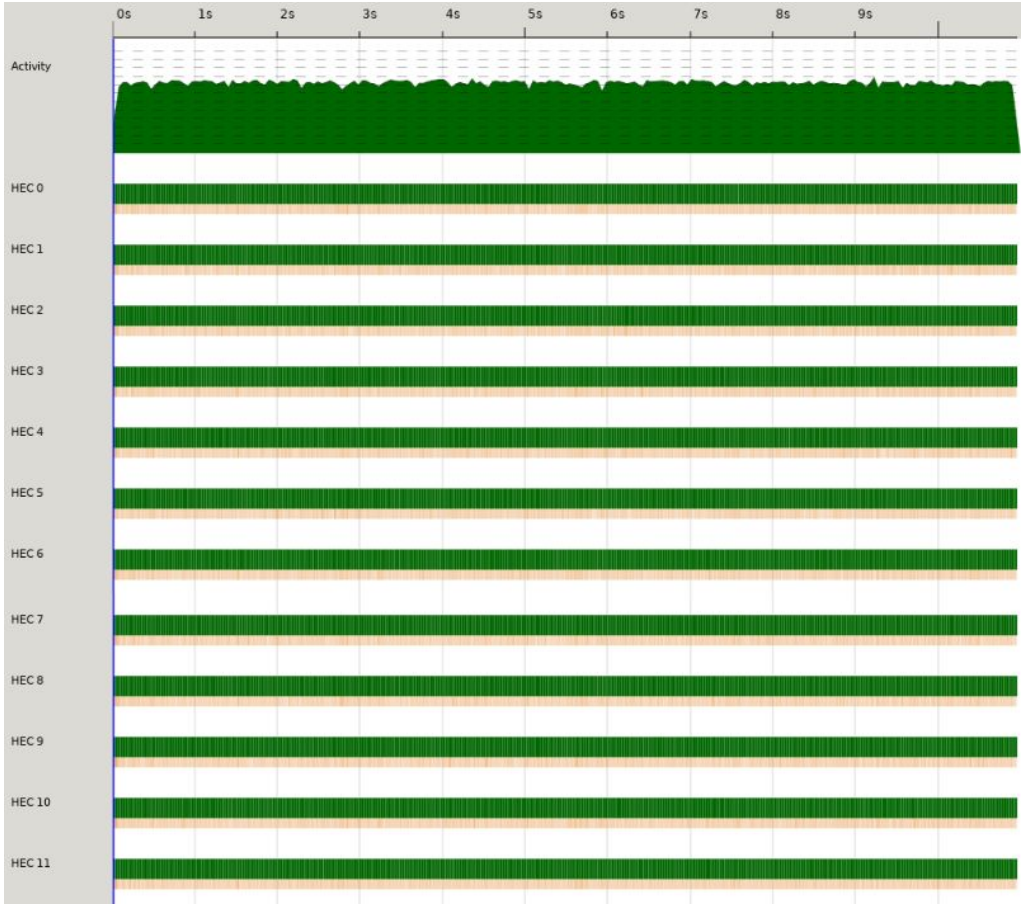



```
2 SPARKS: 792 (31 converted, 0 overflowed, 0 dud, 407 GC'd, 354 fizzled)
3 SPARKS: 2673 (474 converted, 0 overflowed, 0 dud, 2098 GC'd, 101 fizzled)
4 SPARKS: 6336 (1817 converted, 0 overflowed, 0 dud, 4060 GC'd, 459 fizzled)
5 SPARKS: 12375 (10131 converted, 0 overflowed, 0 dud, 2015 GC'd, 229 fizzled)
6 SPARKS: 21384 (20143 converted, 0 overflowed, 0 dud, 1105 GC'd, 136 fizzled)
7 SPARKS: 33957 (33335 converted, 0 overflowed, 0 dud, 512 GC'd, 110 fizzled)
8 SPARKS: 50688 (50181 converted, 0 overflowed, 0 dud, 395 GC'd, 112 fizzled)
9 SPARKS: 72171 (71831 converted, 0 overflowed, 0 dud, 236 GC'd, 104 fizzled)
10 SPARKS: 99000 (98741 converted, 0 overflowed, 0 dud, 155 GC'd, 104 fizzled)
```

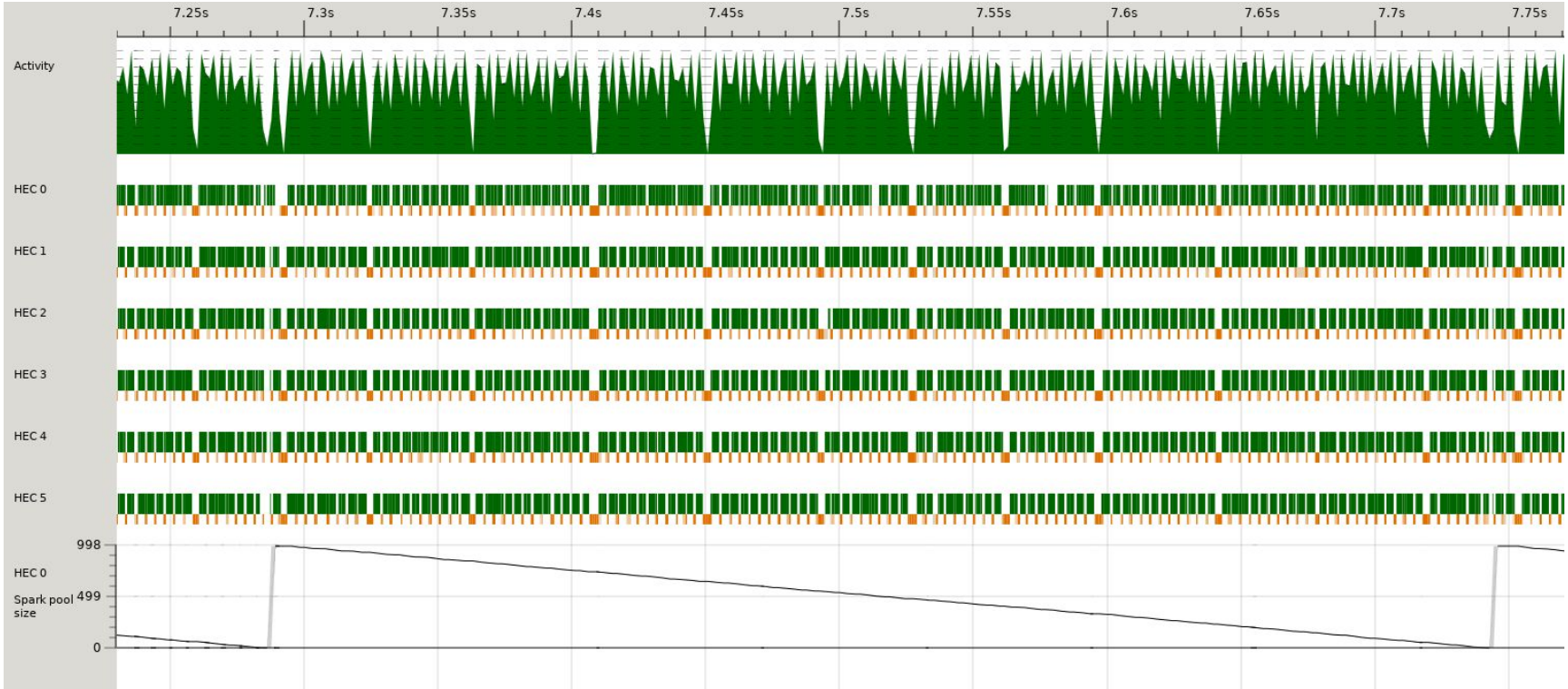
Load Balancing

- Workload seems pretty balanced across all threads!

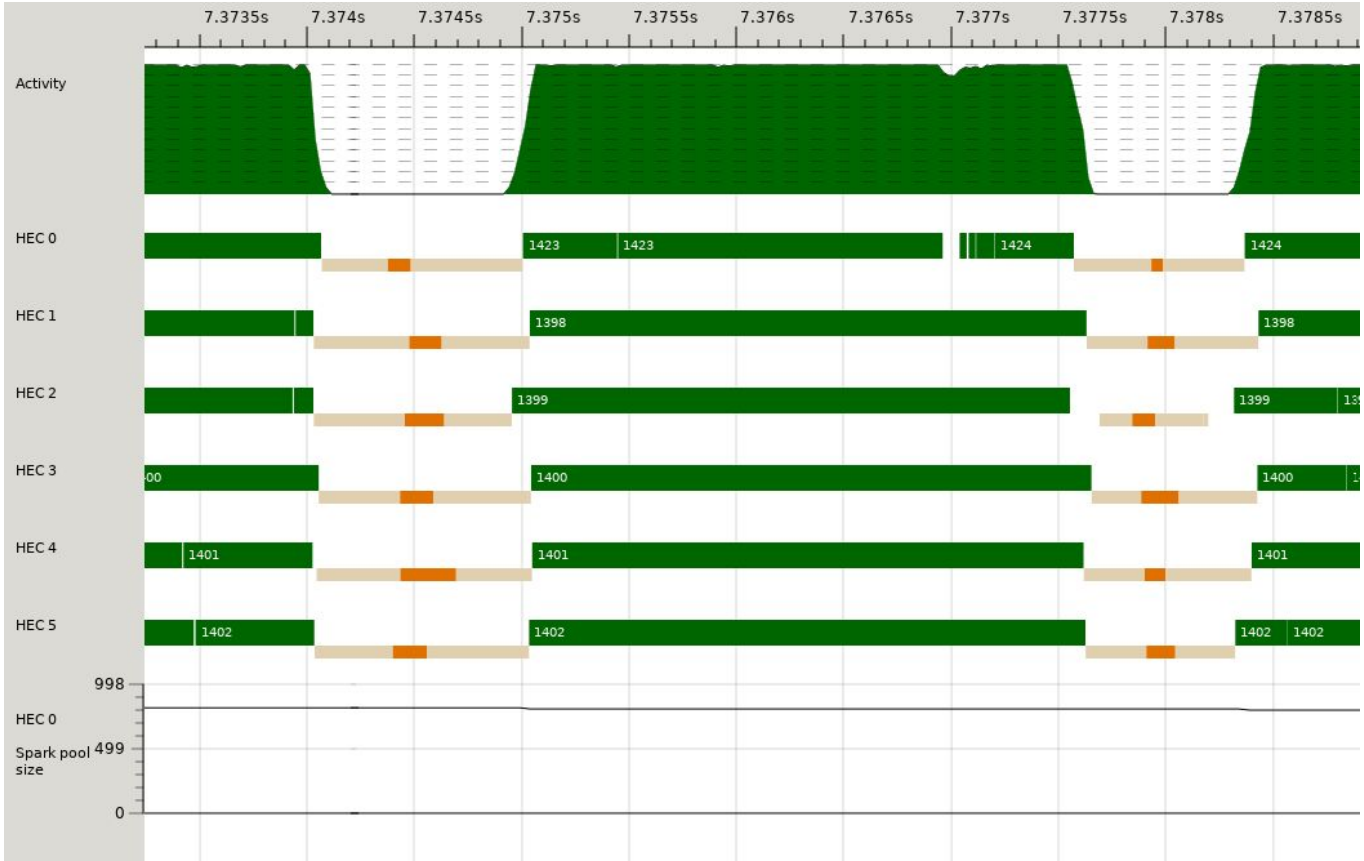
- Let’s take a closer look…



Load Balancing (cont.)



Load Balancing (cont.)



How about parListChunks instead of parList?

```haskell
-- Computes list of forces on all particles given a configuration
forceMatrix :: [MDVector] -> Double -> [MDVector]
forceMatrix rs boxLength =
  map totalForce rs `using` parList rseq
  where
    -- Gets force acting on particle at r1 due to particle at r2
    forceVector r1 r2
      | r1 == r2 = zeroVector
      | otherwise = vectorMultiply flj (unitVector r12)
        where r12 = displacement r2 r1 boxLength
              d12 = vectorNorm r12
              sor = sigma / d12
              flj = 24.0 * epsilon * (2 * (sor ** 12.0) - (sor ** 6.0)) / d12
    -- Computes total force on particle at r due to all other particles
    totalForce r = foldr vectorAdd zeroVector $ map (forceVector r) rs
```

How about parListChunks instead of parList?

Total Chunk Size	-N1	-N2	-N3	-N4	-N5	-N6	-N7	-N8	-N9	-N10	-N11	-N12
1	37.5	25.3	21.8	19.5	**19.0**	19.6	**18.2**	**17.9**	**17.4**	**16.7**	**16.3**	**15.3**
2	41.0	29.6	25.6	24.6	23.7	23.1	23.3	20.9	20.5	21.0	19.9	18.7
4	39.8	28.8	25.4	23.6	22.3	22.1	22.2	21.9	20.7	19.6	20.4	17.8
5	43.4	30.8	26.3	23.8	22.8	22.4	21.6	21.1	21.5	20.8	21.2	19.7
10	**37.4**	**22.6**	**20.8**	**21.0**	20.0	19.2	18.4	18.2	17.9	17.7	18.2	18.2
20	37.7	25.3	23.1	21.3	20.2	**19.0**	19.7	19.5	20.0	19.3	19.2	21.5
50	37.9	25.8	23.3	21.5	20.4	20.2	19.7	19.9	20.7	20.9	20.1	20.1
100	40.2	27.3	26.6	24.0	21.6	22.3	22.9	23.1	23.1	20.5	22.0	21.6

- For few threads: all chunk sizes are basically the same
- For more threads: better off with just going one-by-one (same as parList)