University of Glasgow

n-body using Java fork/join

Jeremy.Singer@glasgow.ac.uk

Source code

- Adapted from ProgLangs shootout Java implementation
- Added input file reader to set up system
- Modified equations (softening, time interval) as specified on SICSA webpage
- 360 SLoC, 5 class files

Adding Parallelism

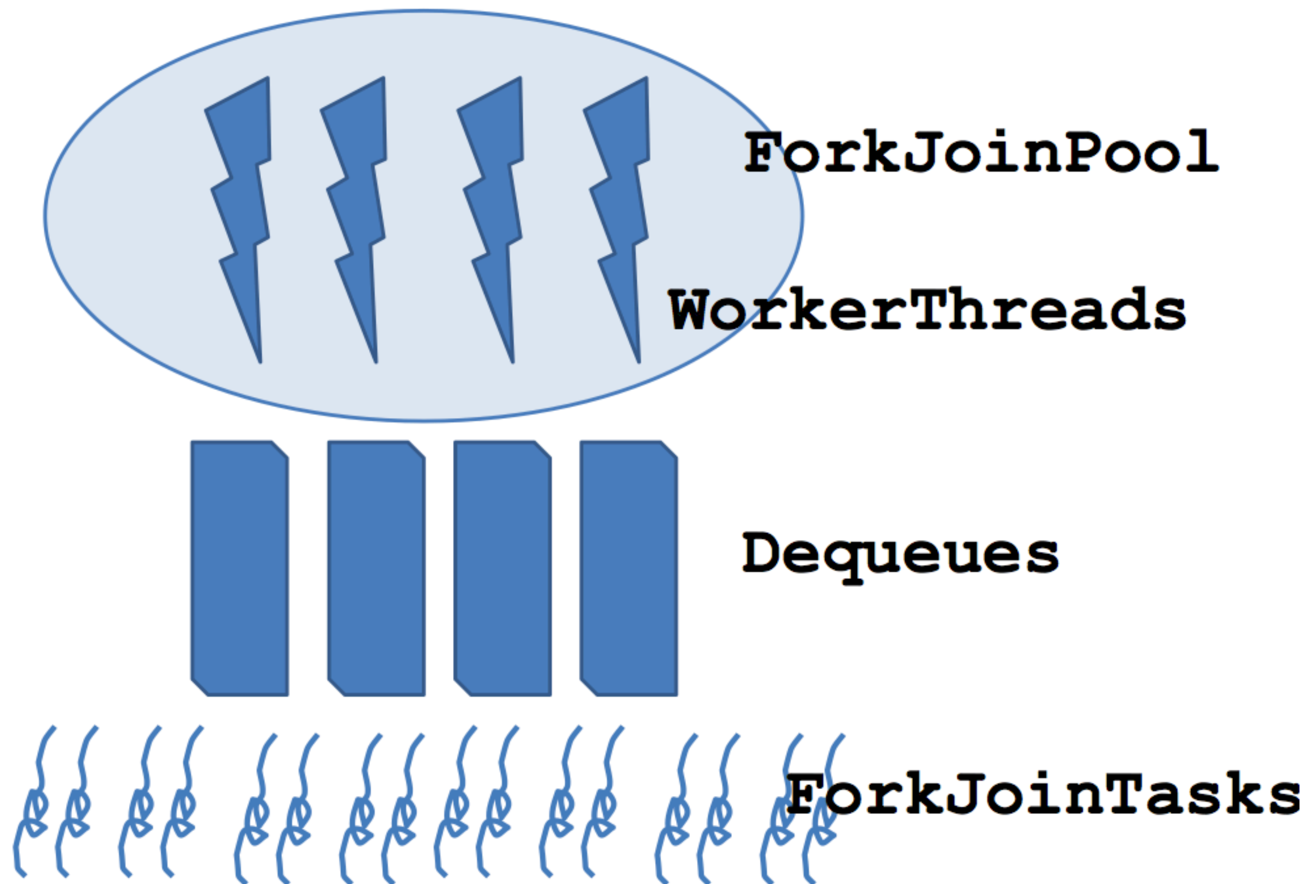
- Concentrated on the advance() code
- Double loop over bodies

```
public void advance(double dt) {
   double dx, dy, dz, distance, mag;

        for(int i=0; i < bodies.length; ++i) {
                for(int j=i+1; j < bodies.length; ++j) {
                        dx = bodies[i].x - bodies[j].x;
                        dy = bodies[i].y - bodies[j].y;
                        dz = bodies[i].z - bodies[j].z;

                        distance = Math.sqrt(dx*dx + dy*d
                        mag = dt / (distance * distance *

                        bodies[i].vx -= dx * bodies[j].ma
```

Parallelize Outer Loop

- Coarser granularity – reduces relative overhead of threading
- Unevenly sized work units (cf triangular matrix calculation)
- Ideal for the Java fork/join framework
  - lightweight threading
  - load-balancing through work-stealing

JSR 166 – Fork/Join Parallelism



Conflicting Writes

- When loop iterations are all in parallel, multiple bodies may have their *v* vectors updated in parallel
- Several strategies to avoid this
  - **`java.util.concurrent.atomic`**
  - **`atomic`** blocks (transactional memory)
  - **`synchronized`** blocks (standard Java)

Synchronized Solution

```
synchronized (bodies[j]) {
  bodies[j].vx += dx * iBody.mass * mag;
  bodies[j].vy += dy * iBody.mass * mag;
  bodies[j].vz += dz * iBody.mass * mag;
}
```

Avoid Over-Allocation

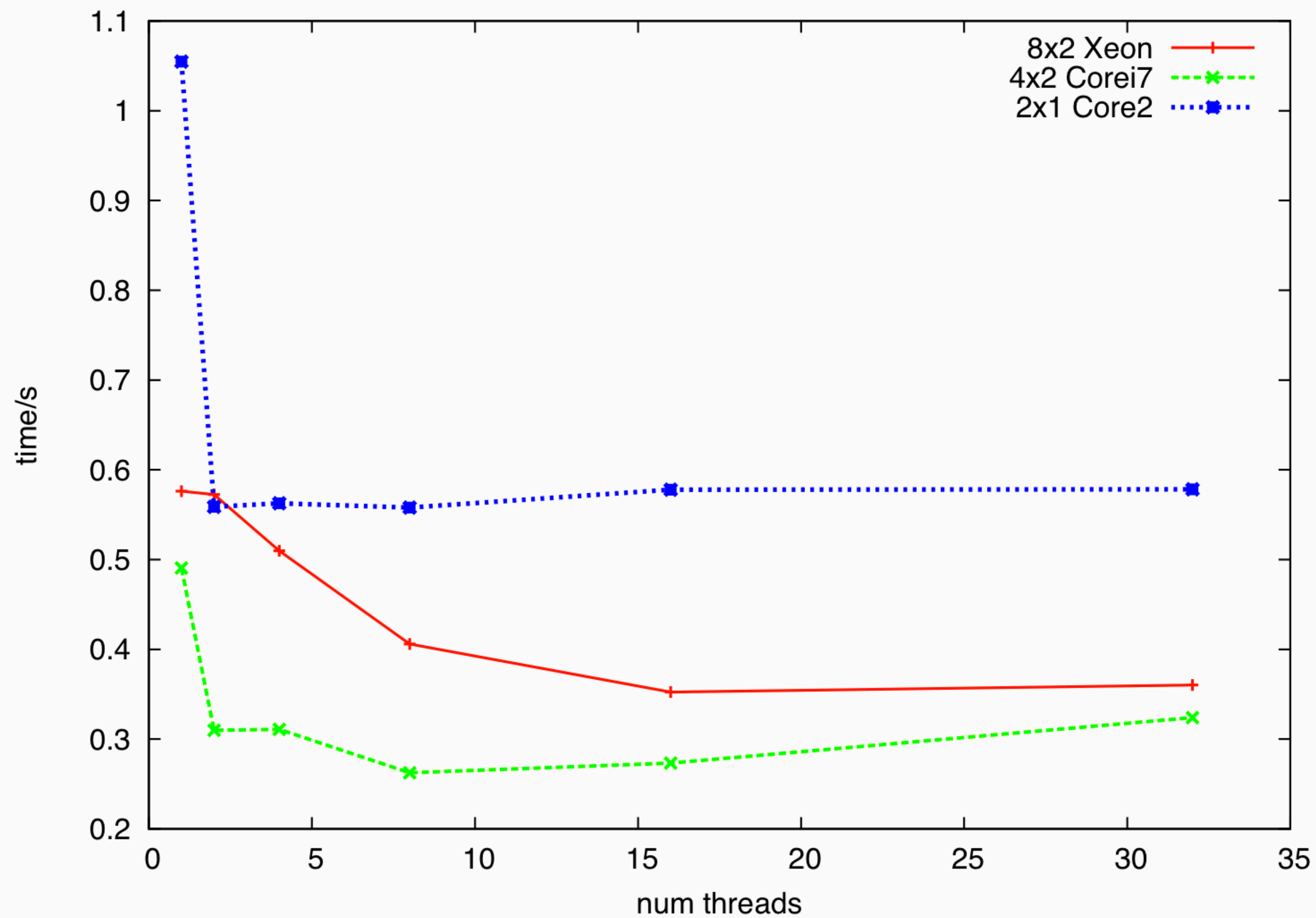
- Re-use ForkJoin tasks at each iteration
- (simply require reinitialization)

- More parallelism could be extracted
  - single for loops, not double. Maybe not enough computation in each iteration?

Results

- for 20 iterations of 1024-body system
- Time reported in seconds, measured using **`System.nanoTime()`**, for the 20 iterations of the loop only.
- Means of 10 runs, low variance in general

Actual Results



Possible Explanations

- for lack of scaling?
  - GC effects? (unlikely)
  - thread affinity problems (all threads staying on same core?)
  - compute intensive benchmarks – not good for hyperthreading
  - spinlocking for synchronized access?
  - memory issues with non-local allocation

Conclusions

- nBody is easy to implement in Java fork/join
  - 1 night's hacking
- require performance / profiling tools to assist in diagnosing scalability problems
  - MSc project