1 Timings of the parallel calculation

Note: The Timing notebook is for reference as related to the paper only - it will not be directly reproducible on re-runs of the above notebooks as it relies on the semi-manual creation of the tasks.log file. The tasks.log file used to generate the original timing data is available for download here.

```
import time
```

Generate labels in the proper order, in case we want to match labels to times (we don't at this point):

```
percentages = range(76,99,3)
region_boundaries = [
     ('v2', 136, 1868), #27f-338r
     ('v2.v3', 136, 2232),
     ('v2.v4', 136, 4051),
     ('v2.v6', 136, 4932),
     ('v2.v8', 136, 6426),
     ('v2.v9', 136, 6791),
     ('v3', 1916, 2232),
     ('v3.v4', 1916, 4051),
     ('v3.v6', 1916, 4932),
     ('v3.v8', 1916, 6426),
     ('v3.v9', 1916, 6791),
     ('v4', 2263, 4051), #515f-806r
     ('v4.v6', 2263, 4932),
     ('v4.v8', 2263, 6426),
     ('v4.v9', 2263, 6791),
     ('v6', 4653, 4932), #967f-1046r
     ('v6.v8', 4653, 6426),
     ('v6.v9', 4653, 6791),
     ('v9', 6450, 6791),
     ('full.length', 0, 7682), # Start 150, 250, 400 base pair reads
     ('v2.150', 136, 702),
     ('v2.250', 136, 1752),
     ('v2.v3.400', 136, 2036), # Skips reads that are larger than amplicon size
     ('v3.v4.150', 1916, 2235),
     ('v3.v4.250', 1916, 2493),
     ('v3.v4.400', 1916, 4014),
     ('v4.150', 2263, 3794),
     ('v4.250', 2263, 4046),
     ('v4.v6.400', 2263, 4574),
     ('v6.v8.150', 4653, 5085),
     ('v6.v8.250', 4653, 5903),
     ('v6.v8.400', 4653, 6419)
]
```

```
labels = []
for p in percentages:
```

```
    for rb in region_boundaries:
        labels.append("%i.%s" % (p,rb[0]))
```

```
ntasks = len(labels)
```

2 Extract times from log

Due to a bug in the Hub, we can't use the standard approach to extracting metadata about execution time.

Fortunately, we can extract most of what we want from the controller's log.

I made a filtered version of the log that only has the relevant lines for the real computation, which we will use here.

We have the arrival and completion time of each task, and since the Scheduler was run with HWM=1, and the IPython overhead of these tasks is so very small, this maps very closely to the actual runtime of each task.

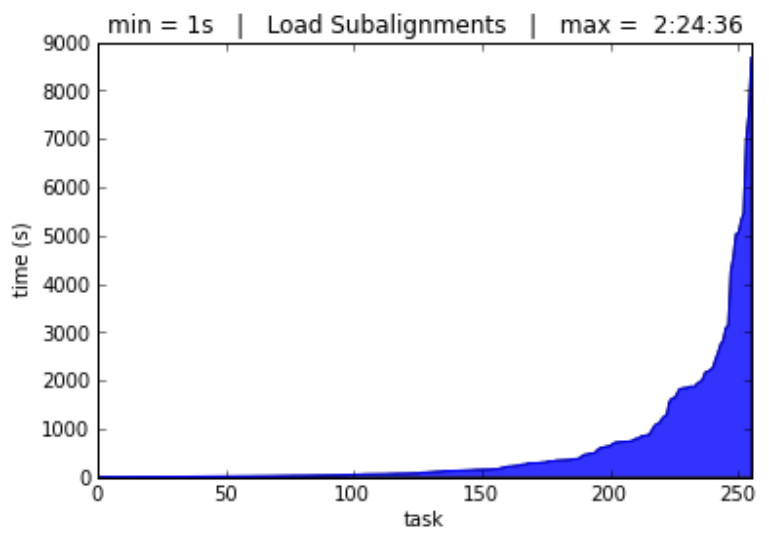
```
stages = ['Load Subalignments', 'Filter', 'Build Trees', 'Compute Distances']

with open('tasks.log') as f:
    lines = f.readlines()

log_segments = [ lines[i*ntasks*2:(i+1)*ntasks*2] for i in range(len(stages)) ]
```

```
from datetime import datetime

tfmt = "%Y-%m-%d %H:%M:%S.%f"

def parse_logline(line):
    parts = line.split()
    timestamp = ' '.join(parts[:2])
    dt = datetime.strptime(timestamp, tfmt)
    msg_id = parts[4][1:-1]
    which = parts[5]
    engine_id = int(parts[7])

    return msg_id, dt, which, engine_id

def parse_log(lines):
    history = [] # ordered, to match labels, if we care
    started = {}
    finished = {}
    for line in lines:
        msg_id, dt, which, engine_id = parse_logline(line)
        if which == 'arrived':
            started[msg_id] = dt
            history.append(msg_id)
        elif which == 'finished':
            finished[msg_id] = dt
```

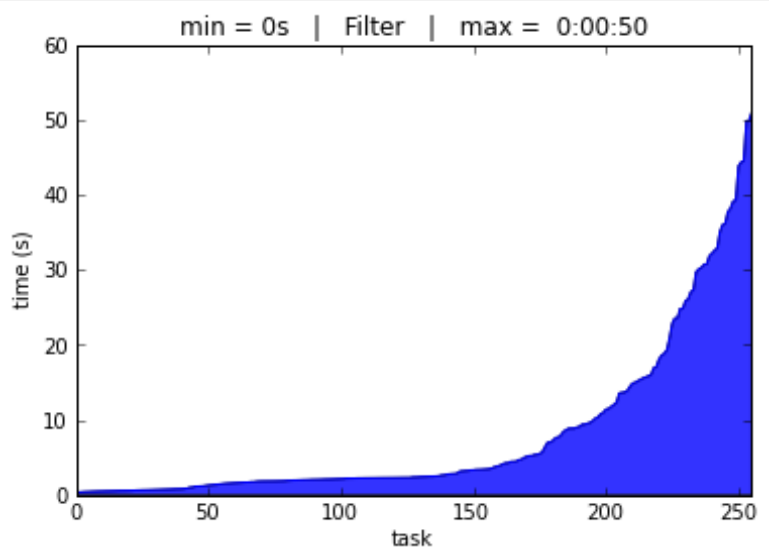
```
        else:
            raise ValueError("expected arrived or finished, got %s" % which)

    # list ordered by start time:
    runtimes = [ (finished[msg_id] - started[msg_id]).total_seconds() for msg_id in
        history ]
    wall = (max(finished.values()) - min(started.values())).total_seconds()

    return runtimes, wall
```

```
def hms(t):
    """render time as HH:MM:SS"""
    it = int(t)
    h = it / 3600
    m = (it % 3600) / 60
    s = (t % 60)
    return "%3i:%02i:%02i" % (h,m,s)
```

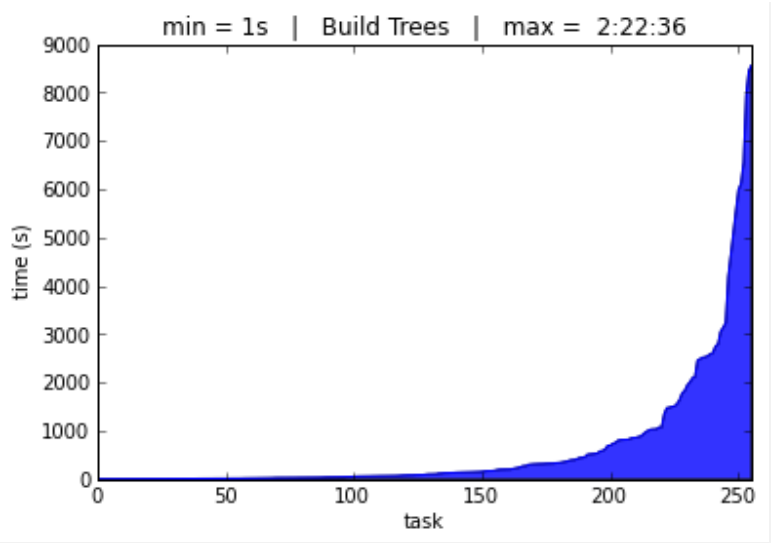
```
serial_times = {}
parallel_times = {}

for stage, segment in zip(stages, log_segments):
    fig = plt.figure()
    times, wall = parse_log(segment)
    serial = sum(times)
    serial_times[stage] = serial
    parallel_times[stage] = wall

    plt.title("min = %is | %s | max =%s" % (min(times),stage,hms(max(times))))
    n = len(times)
    stimes = sorted(times)
    plt.plot(sorted(times))
    plt.fill(range(n)+[n-1,0], sorted(times)+[0,0], alpha=0.8)
    # plt.hist(times, 25)
    plt.ylabel("time (s)")
    plt.xlabel("task")
    plt.xlim(0,n-1)
    display(fig)
    print "%s:" % stage
    print " serial calculation: %s" % hms(serial)
    print " parallel calculation: %s" % hms(wall)
    print " serial / parallel : %8.1fx" % (serial / wall)
    sys.stdout.flush()

# avoid double-plot, because we used display()
plt.close('all')
```




```
<matplotlib.figure.Figure at 0x3d6e810>

Load Subalignments:
    serial calculation:  40:57:31
  parallel calculation:   2:46:24
    serial / parallel :     14.8x
```

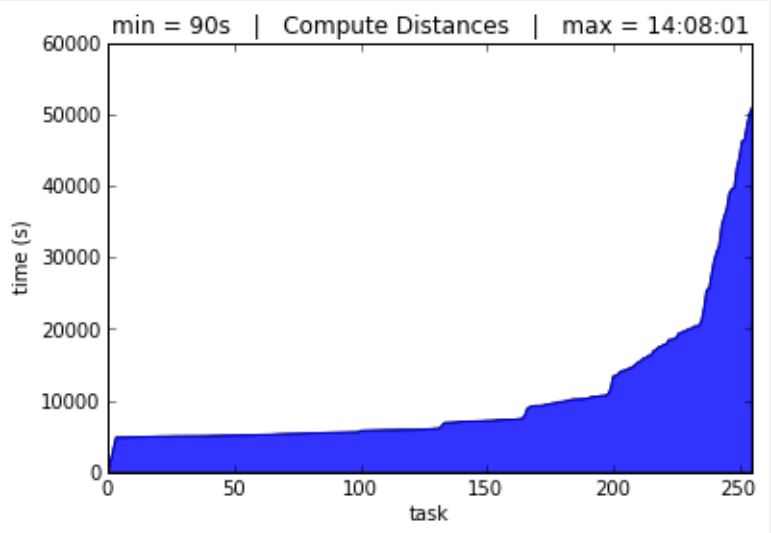



```
<matplotlib.figure.Figure at 0x3d6ead0>

Filter:
    serial calculation:   0:33:21
  parallel calculation:   0:01:15
    serial / parallel :     26.4x
```




```
<matplotlib.figure.Figure at 0x3d97b90>
```

```
Build Trees:
    serial calculation:  44:36:14
  parallel calculation:   2:44:40
    serial / parallel :     16.3x
```




```
<matplotlib.figure.Figure at 0x3dc1650>
```

```
Compute Distances:
    serial calculation: 733:25:48
  parallel calculation:  19:51:29
    serial / parallel :     36.9x
```

3 Stats on total calculation time, and fraction in each major step.

This time, computing the distance matrices totally dominates the runtime (90% serial, 80% real)

```
total_serial = sum(serial_times.values())
total_wall = sum(parallel_times.values())

print "total calculation time: %s (%.1f days)" % (hms(total_serial), total_serial /
    (3600*24))
print "total parallel runtime: %s" % hms(total_wall)
print " serial / parallel: %8.1fx" % (total_serial / total_wall)
print

for stage in stages:
    serial = serial_times[stage]
    wall = parallel_times[stage]
    print "%s:" % stage
    print "\t serial runtime: %s" % (hms(serial))
    print "\t serial percent: %8.1f%%" % (100 * serial / total_serial)
    print "\tparallel runtime: %s" % (hms(wall))
    print "\tparallel percent: %8.1f%%" % (100 * wall / total_wall)
    print
```

```
total calculation time:   819:32:55 (34.1 days)
total parallel runtime:    25:23:50
     serial / parallel:       32.3x

Load Subalignments:
  serial runtime:  40:57:31
  serial percent:      5.0%
parallel runtime:   2:46:24
parallel percent:     10.9%

Filter:
  serial runtime:   0:33:21
  serial percent:      0.1%
parallel runtime:   0:01:15
parallel percent:      0.1%

Build Trees:
  serial runtime:  44:36:14
  serial percent:      5.4%
parallel runtime:   2:44:40
parallel percent:     10.8%

Compute Distances:
  serial runtime: 733:25:48
  serial percent:     89.5%
parallel runtime:  19:51:29
parallel percent:     78.2%
```