10000 to 1

(tasks per second)

Ben Clifford
benc@hawaga.org.uk
Baseline A: my laptop, local threads without logging

10188 tasks/sec
Baseline B: NERSC Perlmutter with Work Queue

0.63 tasks/sec
Variant L: my laptop, local threads, default logging on

```bash
$ parsl-perf --config parsl/tests/configs/local_threads.py
```

### Iteration 1
Will run 10 tasks to target 120 seconds runtime
Submitting tasks / invoking apps
All 10 tasks submitted ... waiting for completion
Submission took 0.002 seconds = 4216.228 tasks/second
Runtime: actual 0.003s vs target 120s
Tasks per second: 3837.775

### Iteration 2
Will run 460532 tasks to target 120 seconds runtime
Submitting tasks / invoking apps
All 460532 tasks submitted ... waiting for completion
Submission took 99.553 seconds = 4625.993 tasks/second
Runtime: actual 101.031s vs target 120s
Tasks per second: 4558.311
Cleaning up DFK
The end
Variant R: perlmutter Work Queue, 1 core per task

$ parsl-perf --resources '{"cores":1, "memory":0, "disk":0}' ...

(2.8x speedup on a host with 256 cores...)

0.62

1.68

tasks/sec
Variant C: Work Queue coprocesses

Persistent Python workers

== Task Vine "serverless" mode

1.68
400

tasks/sec
Variant M: Monitoring

- basic monitoring
  - serialization
  - process forking
  - network message delivery

- in-task resource monitoring
  - 400 tasks/sec
  - 300 tasks/sec
  - 11 tasks/sec
Variant S: my serialization mess up

![Graph showing tasks/second against cumulative tasks]

11 tasks/sec
Variant Q: Work Queue queue length slow down

![Work Queue batch size slowdown graph](graph.png)
Cost of monitoring

Cost of task-level resource monitoring

WQ queue length (Q)

Monitoring serialization cache bug (S)

One core per task (R)

Naive Perlmutter WQ (B)

Work Queue coprocesses (C)

Cost of logging on laptop (L)

Laptop (A)

10000 tasks/sec

1000

100

10

1