# Benchmarking Concurrent Programs

Scott West

ETH Zürich

November 23, 2012

# Measure, then cut

One result from the parallel language study: SCOOP has poor performance on shared-memory benchmarks.

A successful application of benchmarking: we found a problem!

# Measure, then cut

One result from the parallel language study: SCOOP has poor performance on shared-memory benchmarks.

A successful application of benchmarking: we found a problem!

# Concurrency v. Parallelism

**Definition (Parallelism)**

A program doing more than one thing at a time to compute a result faster. The result is deterministic.

**Definition (Concurrency)**

A program with multiple threads of control that which interleave in a non-deterministic way.

They are not exclusive: concurrency can be used to implement parallelism.

# Our task

A performance benchmark should be a way to measure a relevant aspect of execution properties.

Parallelism benchmarks define a task that should have the same result every time, and the result must arrive quickly!

We aim to define a way to benchmark concurrent programs.

# Our task

A performance benchmark should be a way to measure a relevant aspect of execution properties.

Parallelism benchmarks define a task that should have the same result every time, and the result must arrive quickly!

We aim to define a way to benchmark concurrent programs.

# Our task

A performance benchmark should be a way to measure a relevant aspect of execution properties.

Parallelism benchmarks define a task that should have the same result every time, and the result must arrive quickly!

We aim to define a way to benchmark concurrent programs.

# What is a concurrent benchmark?

Since a concurrent program executes nondeterministically, benchmarking will concern profiling the performance characteristics of various *synchronization*, *communication*, and *coordination* tasks.

# Benchmark Overview

We wish to examine *scalability*: how does a benchmark react to increased contention/threads/problem size.
We will examine concurrent programs with multiple patterns:

- **no share** threads do not share memory or coordinate.
- **1-N comm.** a single master thread dispatches work to multiple workers.
- **M-N comm.** multiple master threads dispatch work
- **mutex** multiple threads contend for a critical section with high contention.
- **benign share** threads share memory, but do not have guarded access to it.

# Noninterference

**Purpose**

*To test the cost of running multiple threads by the runtime system.*

Some concurrency abstractions have non-trivial departures from POSIX-like threads (such as green threads) and this such a test would determine the effect of any such abstractions.

This would take the shape of running some compute heavy task in each thread. I,e.

```
fib (30)
```

# 1-N communication

**Purpose**

*To examine the cost of having multiple threads contending for a single event.*

This test replicates a common design in server applications where a main thread dispatches work to multiple reader threads.

The test will have a single channel of communication where the master dispatches integers to the workers. The workers continually receive and discard the integers.

# M-N communication

**Purpose**

*To test the effect of multiple producers accessing a single consumption channel.*

Similar to the previous test, but with multiple masters and workers at either end of the work distribution.

# Mutex

**Purpose**

*To quantify the cost of gaining exclusive access to a critical section.*

The critical section should be heavily contended as this is the case where performance issues tend to arise.

The mutex operations may be noops (if a CS is not explicitly required), but *must* update some shared state in a simple way (ie, adding 1).

# Interfering

**Purpose**

*To examine the overhead of continually accessing a shared resource without any extra protection not provided by the implementation.*

Very often there are benign dataraces, this test quantifies the performance cost of modifying shared memory without an explicit mutex protecting it.

# Preliminary results

| Benchmark | STM (Haskell) | SCOOP |
|---|---|---|
| No share | – | – |
| 1-N comm. | – | – |
| M-N comm. | 0.478 s | 63.747 s |
| Mutex | 0.067 s | 24.235 s |
| Benign share | – | – |

# Going further

- Add more microbenchmarks, gathered from real situations.
- Implement the benchmarks in various concurrency approaches, with a focus on languages that focus on concurrency rather than data parallelism.
    - Candidates for other approaches: Erlang, Go, Ada, Clojure, F#, D, Java, Scala, Orc...
- Validation step: use them to predict performance of larger programs.

# Publication

Rough plan: ISSTA submission early next year.