Do not believe everything you read in the papers

Tim Harris
10 February 2016
The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.
Good intentions, bad clip art
Good intentions, bad clip art
An example from my recent work

Better: Algorithm running with 18/36/72 threads
What I want to compare

the performance using our C++ runtime system from Java (via an optimizing compiler with a lightweight native function interface)

with

the performance using standard Java fork-join.
What I am actually comparing

Differences in thread placement

Differences in memory placement

Differences in page sizes

Differences in GC activity

Changes in low-level code quality

Changes in work distribution granularity

...
This talk is about

• Making experimental work more methodical
• Some of the “usual suspects” when understanding performance
• Presenting results
This talk is about

• Making experimental work more methodical
• Some of the “usual suspects” when understanding performance
• Presenting results
• Caveats
  – I am mainly talking about work on shared-memory algorithms and data structures
  – Some of these observations may apply elsewhere, but I am sure the war stories differ
This talk is about

• Making experimental work more methodical
• Some of the “usual suspects” when understanding performance
• Presenting results
• Caveats
  – I am mainly talking about work on shared-memory algorithms and data structures
  – Some of these observations may apply elsewhere, but I am sure the war stories differ
• There are a lot of other elements to consider
  – Experimental design
  – Statistical analysis of results
Overview

1. Script everything, derive results from measurements
2. Plan how to present results before starting work
3. Understand simple cases first
Script everything, record everything

Building
• From checked-in code in repository
• Reduce dependencies on environment
• Record versions actually used

Running
• Record everything:
  • Machine used, system load, ...
  • Command lines invoked
  • UNIX environment

Generating results
• Take the output of a run (e.g., text logs)
• Clean up
• Generate finished clean graphs
  (e.g., PDF for papers and EMF for slides)
Script everything, record everything

- From checked-in code in repository
- Reduce dependencies on environment
- Record versions actually used

One “run” script.
One results file.
One “process” script.

- Record everything:
  - Machine used, system load, ...
  - Command lines invoked
  - UNIX environment

- Take the output of a run (e.g., text logs)
- Clean up
- Generate finished clean graphs (e.g., PDF for papers and EMF for slides)
• Take the output of a run (e.g., text logs)
• Clean up
• Generate finished clean graphs (e.g., PDF for papers and EMF for slides)

Script everything, record everything

Building
Running
Generating results

• From checked-in code in repository
• Reduce dependencies on environment
• Record versions actually used

• Record everything:
  • Machine, system load, …
  • Command lines invoked
  • UNIX environment

+ date
+ g++ --version
g++ (GCC) 4.9.1
Copyright (C) 2014 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

+ export CLIENTS_PER=10
+ CLIENTS_PER=10
+ export QUEUE=bunch-unreservedq
+ QUEUE=bunch-unreservedq
+ export TIME_MINUTES=120
+ TIME_MINUTES=120
+ FLAGS=
+ cp config-big-scale-both.hpp config.hpp
+ cat config.hpp
/*
 * config.hpp
 */
* Created on: 27.Jan.2015
* Author: erfanz
*/
Script everything, record everything

Building

Running

Generating results

- From checked-in code in repository
- Reduce dependencies on environment
- Record versions actually used
- Record everything:
  - Machine, system load, …
  - Command lines invoked
  - UNIX environment

```
+ date
Sun Jan 24 11:31:23 PST 2016
+ g++ --version
g++ (GCC) 4.9.1
```

Copyright (C) 2014 Free Software Foundation; see the
warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTicular PURPOSE.

```
+ export CLIENTS_PER=10
+ CLIENTS_PER=10
+ export QUEUE=bunch-unreservedq
+ QUEUE=bunch-unreservedq
+ export TIME_MINUTES=120
+ TIME_MINUTES=120
+ export SERVERS
+ export SERVERS
+ cp config-big-scale-both.hpp config.hpp
+ cat config.hpp /*
  * Created on: 27.Jan.2015
  * Author: erfanz */
+ srun: Job allocation 1955166 has been revoked.
```

```
srun: Job step aborted: Waiting up to 2 seconds for job step to finish.
```

```
srun: error: bunch003: task 2: Terminated
```

```
+ for SERVERS in 1 2 4 8 16 24 32 48
+ export CLIENT_MACHINES=4
+ CLIENT_MACHINES=4
+ MC=9
+ date
Sun Jan 24 11:38:45 PST 2016
+ sinfo
+ grep bunch-unreservedq
bunch-unreservedq up 4:00:00 100 idle bunch[001-100]
+ export SERVERS
```

```
+ salloc -pbunch-unreservedq -t120 -N9 -n9 --comment=brown-tx-scale-4-9
salloc: Granted job allocation 1955168
+ make -j
```

```
g++ -std=gnu++11 -g -O3 -Wall -Wconversion -Wextra -Wno-ignored-qualifiers
-Wno-write-strings -Isrc/util -Isrc/basic-types -Isrc/executor -Isrc/TSM-SI -Isrc/TSM-SI/client
-Isrc/TSM-SI/server -Isrc/TSM-SI/timestamp-oracle -c src/util/BaseContext.cpp
-o build/util/BaseContext.o
```

```
+ srun
```
Script everything, record everything

- From checked-in code in repository
- Reduce dependencies on environment
- Record versions actually used

Building

- Record everything:
  - Machine used, system load, ...
  - Command lines invoked
  - UNIX environment

Running

- Take the output of a run (e.g., text logs)
- Clean up
- Generate finished clean graphs (e.g., PDF for papers and EMF for slides)

Generating results
Starting and stopping work

The test harness was parameterized on the algorithm to use, the number of concurrent threads to operate and the range of keys that might be inserted or deleted. In each case every thread performed 1000000 operations. Figure 6 shows the CPU accounted to the process as a whole for each of the algorithms tested on a variety of workloads.

"A pragmatic implementation of non-blocking linked lists", Tim Harris, DISC 2001
Starting and stopping work

• How much work to do?

Too little: results dominated by start-up effects. Normalized metrics vary as you vary the duration.

Short runs

Long runs
Starting and stopping work

• How much work to do?

- Too little: results dominated by start-up effects. Normalized metrics vary as you vary the duration.
- OK: results not sensitive to the exact choice of settings. Confirm this: double / halve duration with no change.
Starting and stopping work

• How much work to do?

Too little: results dominated by start-up effects. Normalized metrics vary as you vary the duration.

OK: results not sensitive to the exact choice of settings. Confirm this: double / halve duration with no change.

Too much??
Starting and stopping work

• How much work to do?

<table>
<thead>
<tr>
<th>Short runs</th>
<th>Long runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too little: results dominated by start-up effects. Normalized metrics vary as you vary the duration.</td>
<td>OK: results not sensitive to the exact choice of settings. Confirm this: double / halve duration with no change.</td>
</tr>
<tr>
<td>Deters experimentation if turnaround time is long (e.g. &gt;&gt; overnight)</td>
<td>Harder to separate resource re-use policy from the rest of the expt.</td>
</tr>
</tbody>
</table>
Starting and stopping work... what we imagine:

10,000,000 operations
10,000,000 operations
10,000,000 operations
10,000,000 operations

Measure duration = 2s

Throughput = 4M / 2s = 2M ops / s
Starting and stopping work... what we get:

1,000,000 operations
1,000,000 operations
1,000,000 operations
Constant load

• Fixed number of threads active
  – E.g., data structure micro-benchmarks
  – Look at how the structure under test behaves under varying loads

• Keep all threads active throughout experiment. Typically:
  – Create threads
  – Perform warm-up work in each thread
  – Barrier
  – Actual measurement interval
  – Main thread signals request to exit to others

• Investigate and report differences in actual work completed by threads
Constant work

• Fixed amount of work to perform
  – Share it among a set of threads – e.g., OpenMP parallel loop
  – Aim to use threads to complete the work more quickly
  – Measure from when the work is started until when it is all complete

• Show results for
  – Strong scaling: same amount of work as you vary the number of threads
  – Weak scaling: increase the work proportional to the threads

• Investigate and report differences in
  – Load imbalance (do threads finish early?)
  – Actual amount of work completed by threads (do some threads work faster?)
Unfairness: simple test-and-test-and-set lock

- Main thread runs a constant number of iterations, signals others to stop
- 2-socket Haswell, threads pinned sequentially to cores in **1 socket**
Unfairness: simple test-and-test-and-set lock

- Main thread runs a constant number of iterations, signals others to stop
- 2-socket Haswell, threads pinned sequentially to cores in both sockets

Operations per thread normalized to main

H/W thread number (0..36)

45x, not 45%!\[\]
Unfairness: Synchrobench, Fraser skip list, read only

Normalized throughput ops/s at 36 threads (1 per core)

- Default (first touch?)
- `numactl --membind=0,1`
Unfairness: Synchrobench, Fraser skip list, read only

Normalized throughput (ops/s at 36 threads (1 per core))

- Default (first touch?)
- `numactl --membind=0,1`
- `numactl --membind=0`
Unfairness: Synchrobench, Fraser skip list, read only

Normalized throughput (ops/s at 36 threads) (1 per core)

- Default (first touch?)
- `numactl --membind=0,1`
- `numactl --membind=0`
Unfairness: Synchrobench, Fraser skip list, read only

Normalized throughput ops/s at 36 threads (1 per core)

cnumactl --interleave=0,1
Script everything, record everything

**Building**
- From checked-in code in repository
- Reduce dependencies on environment
- Record versions actually used

**Running**
- Record everything:
  - Machine used, system load, ...
  - Command lines invoked
  - UNIX environment

**Generating results**
- Take the output of a run (e.g., text logs)
- Clean up
- Generate finished clean graphs (e.g., PDF for papers and EMF for slides)
Generating results

General principle: derive results from numbers you measure, not from numbers you configure
Generating results

General principle: derive results from numbers you measure, not from numbers you configure

- Configuration setting written in incorrect file
- Code that reads the setting is buggy
- Environment variable set incorrectly ("GOMP_PROC_BIND")
- System overrides the settings (e.g., thread pinning)
- Setting is invalid and ignored at runtime
Generating results

“Bind threads 1 per socket”

“Run for 10s”

“Use 50% reads”

“Distribute memory across the machine”

Have each thread report where it is running

Record time at start & end

Measured #reads/#ops

Actual locations and page sizes used
Overview

1. Script everything, derive results from measurements
2. Plan how to present results before starting work
3. Understand simple cases first
Plan how to present results before starting work

• Why?
  – Make sure you can illustrate the problem you are solving and you know the questions you want to see answered
    • How bad are things now?
    • How much scope exists for improvement?
  – Time to practice explaining the format of the results to other people
  – Time to notice and resolve difficulties running experiments
  – Coding/tweaking/experimenting will expand to fill the time available
    • Let them!
Running pairs of workloads together on a 2-socket machine

Run “triangle counting” and “equake” together on the 2-socket machine. Time how long triangle counting takes compared with running alone on 1 socket.
Running pairs of workloads together on a 2-socket machine
Running pairs of workloads together on a 2-socket machine
Why does this format work?

• Easy to explain what a good result is like and what a bad result is like

• A neutral result is “quiet”
  – All the squares are white
  – No need to understand what the workloads actually do

• Captures trade-offs
  – Results here often come in pairs
  – Green with red
  – We will see both of them together

• “Dashboard” while doing the work
Another example – scalability microbenchmark

SPARC T5-8, 1024 threads

Normalized speedup

Perf relative to 1 thread and no work distribution overheads

~400x speed-up about max we would expect given the IPC here

Different work scheduling mechanisms, vary the batch size used for distribution

(See USENIX ATC 2015 for the different techniques.)
Microbenchmark results

SPARC T5-8, 1024 threads

Y-intercept shows the best-case overhead at very large batch sizes.

Expect a straight horizontal line for perfect scaling to smaller batch sizes.
Why does this work?

• Easy to explain what a good result is like and what a bad result is like

• A neutral result is “quiet”
  – All the squares are white
  – No need to understand the different workloads

• Captures trade-offs
  – Results here often come in pairs
  – Green with red
  – We will see both of them together
Trade-offs

• Parallel stop-the-world garbage collector
• Suppose it takes 5% of execution time on average
• Do you care?
Trade-offs

Submit request

Get response

All I care about is the ratio of red to grey
Trade-offs

Now I do care that unlucky requests are delayed

- Fan-outs / nesting
- Real time systems
- Low-latency trading
Minimum mutator utilization

- Minimum mutator utilization
- Whole-run throughput
- Max pause time

Time window

0% to 100%
Minimum mutator utilization

- Minimum mutator utilization
- Time window
- Max pause time
- Whole-run throughput
- Shorter pauses, worse throughput
Minimum mutator utilization

Minimum mutator utilization

Max pause time

Whole-run throughput

Strictly better

Time window

0% to 100%
Bandwidth vs latency

Increasing offered load (e.g. number of active clients)
Bandwidth vs latency

Frequently: both metrics become worse as the load increases

Increasing offered load (e.g. number of active clients)
Summary

• Make formats easy to explain, e.g.:
  – Ideal behaviour is a horizontal line
  – Ideal behaviour is a blank heat map

• Make numbers easy to read off
  – What does a y-intercept mean?
  – What does a x-intercept mean?
  – Is anything hidden where lines are clumped together?

• Show and expect to see trade-offs
Overview

1. Script everything, derive results from measurements
2. Plan how to present results before starting work
3. Understand simple cases first
Understand simple cases first

• Why? Almost without exception:
  – There are bugs in the test harness
  – There are bugs in the data processing scripts (grep, cut-n-paste, …)
  – There are unexpected factors influencing the results
Understand simple cases first

• Why? Almost without exception:
  – There are bugs in the test harness
  – There are bugs in the data processing scripts (grep, cut-n-paste, …)
  – There are unexpected factors influencing the results

• Before paying any attention to actual results, try to identify simple test cases that should have known behavior
  – (Even if you do not care about them, or they are contrived)
  – Do they behave as expected?
  – Can you completely explain them? (“Memory system effects” is not an answer)
  – Add them to regression tests, and watch for them breaking
Basic checks to make

• Should the workload be 100% user mode?
  – Confirm this with “top”
  – Check that “strace” is quiet (no system call activity)

• Where are the threads running?

• Where is the memory they access located?

• What do profiling tools show?
  – Can you use with optimized builds? If not, check impact of disabling optimization
  – If you have long-running use cases, does the profile actually match them?
  – Look at 1-thread workloads – as expected?
  – Increase thread count and look for trends
Synchrobench, Fraser skip-list, 100 % read only, X5-2
Synchrobench, Fraser skip-list, 100% read only, X5-2

Is this a good set of results? It’s certainly not a good graph.
Synchrobench, Fraser skip-list, 100 % read only, X5-2

Which of these lines (if either) would be perfect scaling?

Ugly numbers. Is this good performance or poor?

Is this a good set of results? It’s certainly not a good graph

Most of the data is buried down here.
Synchrobench, Fraser skip-list, 100 % read only, X5-2

Normalize to optimized sequential code (and report absolute baseline). Self-relative scaling is almost never a good metric to use.
Synchrobench, Fraser skip-list, 100 % read only, X5-2

Synergy: “horizontal is good” formats are unaffected by switching to/from log-scale axes
Synchrobench, Fraser skip-list, 100 % read only, X5-2

Disable Turbo Boost, becomes flatter
Synchrobench, Fraser skip-list, 100 % read only, X5-2

Improvements to tuning of GC and use of memory fences.
Synchrobench, Fraser skip-list, 100 % read only, X5-2

Initially horizontal (as expected) at low thread counts.
Synchobench, Fraser skip-list, 100 % read only, X5-2

What is happening here? The simplest case that is not yet understood.
(It was a stray process still running on the machine)
Overview

1.Script everything, derive results from measurements
2. Plan how to present results before starting work
3. Understand simple cases first
An example from my recent work

Better

Algorithm running with 18/36/72 threads

Previous system

New system
An example from my recent work

Better

1. Work distribution chunk size 1024 vs 4096

Algorithm running with 18/36/72 threads
An example from my recent work

1. Work distribution chunk size 1024 vs 4096
2. Some additional GC activity with fork-join

Algorithm running with 18/36/72 threads
An example from my recent work

Better

- Work distribution chunk size 1024 vs 4096
- Some additional GC activity with fork-join
- False sharing on VM “-UseMembar” page
Future work

• Three aspects to this talk:
  – Working practices to try to make sure there is time to understand results
  – Formats for presenting results to help understand them
  – Recurring problems from this particular area of research
Future work

• Three aspects to this talk:
  – Working practices to try to make sure there is time to understand results
  – Formats for presenting results to help understand them
  – Recurring problems from this particular area of research

• I would like to have more common infrastructure for running experiments
  – Help run experiments consistently
  – Same allocator, same thread placement, ...
  – Use raw output logs as part of artefact evaluation processes
  – By using it, help convince others that experiments are run well
Further reading

• Books
  – Huff & Geis – “How to Lie with Statistics”

• Papers and articles
  – Bailey – “Twelve Ways to Fool the Masses”
  – Fleming & Wallace – “How not to lie with statistics: the correct way to summarize benchmark results”
  – Heiser – “Systems Benchmarking Crimes”
Integrated Cloud
Applications & Platform Services