Coz: Finding Code that Counts with Casual Profiling

Charlie Curtsinger, Emery D. Berger

Grinell College
University of Massachusetts Amherst

Presented at SOSP 2015

Coz: Finding Code that Counts with ~~Casual~~ Causal Profiling

Charlie Curtsinger, Emery D. Berger

Grinell College
University of Massachusetts Amherst

Presented at SOSP 2015

Motivation

Conventional Profilers:

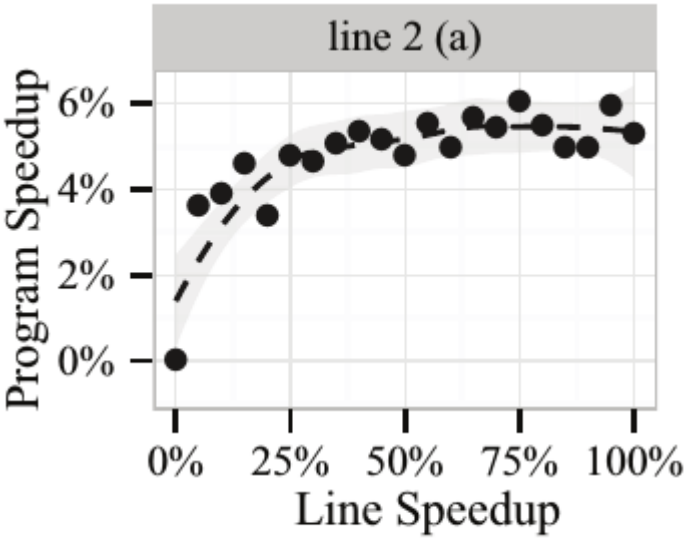
- don't tell you what to optimize
- don't tell you about gains
- are not really made for concurrency

example.cpp

```
1  void a() { // ~6.7 seconds
2    for(volatile size_t x=0; x<2000000000; x++) {}
3  }
4  void b() { // ~6.4 seconds
5    for(volatile size_t y=0; y<1900000000; y++) {}
6  }
7  int main() {
8    // Spawn both threads and wait for them.
9    thread a_thread(a), b_thread(b);
10   a_thread.join(); b_thread.join();
11 }
```

Motivation

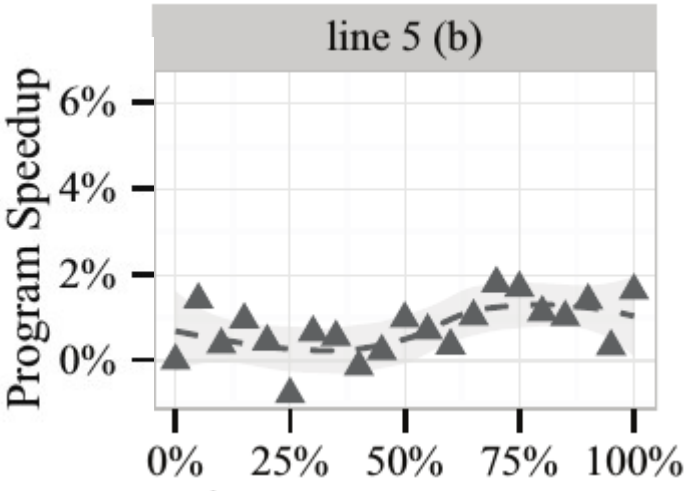
```
1  void a() { // ~6.7 seconds
2    for(volatile size_t x=0; x<2000000000; x++) {}
3  }
4  void b() { // ~6.4 seconds
5    for(volatile size_t y=0; y<1900000000; y++) {}
6  }
7  int main() {
8    // Spawn both threads and wait for them.
9    thread a_thread(a), b_thread(b);
10   a_thread.join(); b_thread.join();
11 }
```



Conventional Profile for `example.cpp`

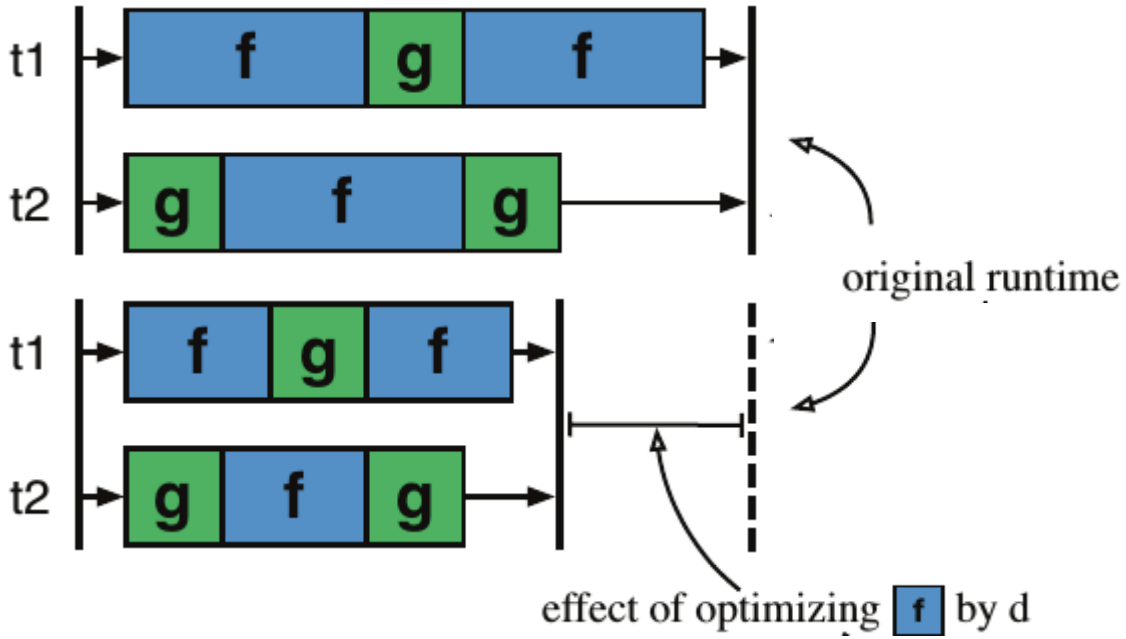
```
    %   cumulative     self              self     total
 time      seconds   seconds   calls  Ts/call  Ts/call  name
55.20         7.20      7.20       1                    a()
45.19        13.09      5.89       1                    b()

% time    self    children    called     name
                                        <spontaneous>
  55.0    7.20       0.00                a()
-----------------------------------------------
                                        <spontaneous>
  45.0    5.89       0.00                b()
```

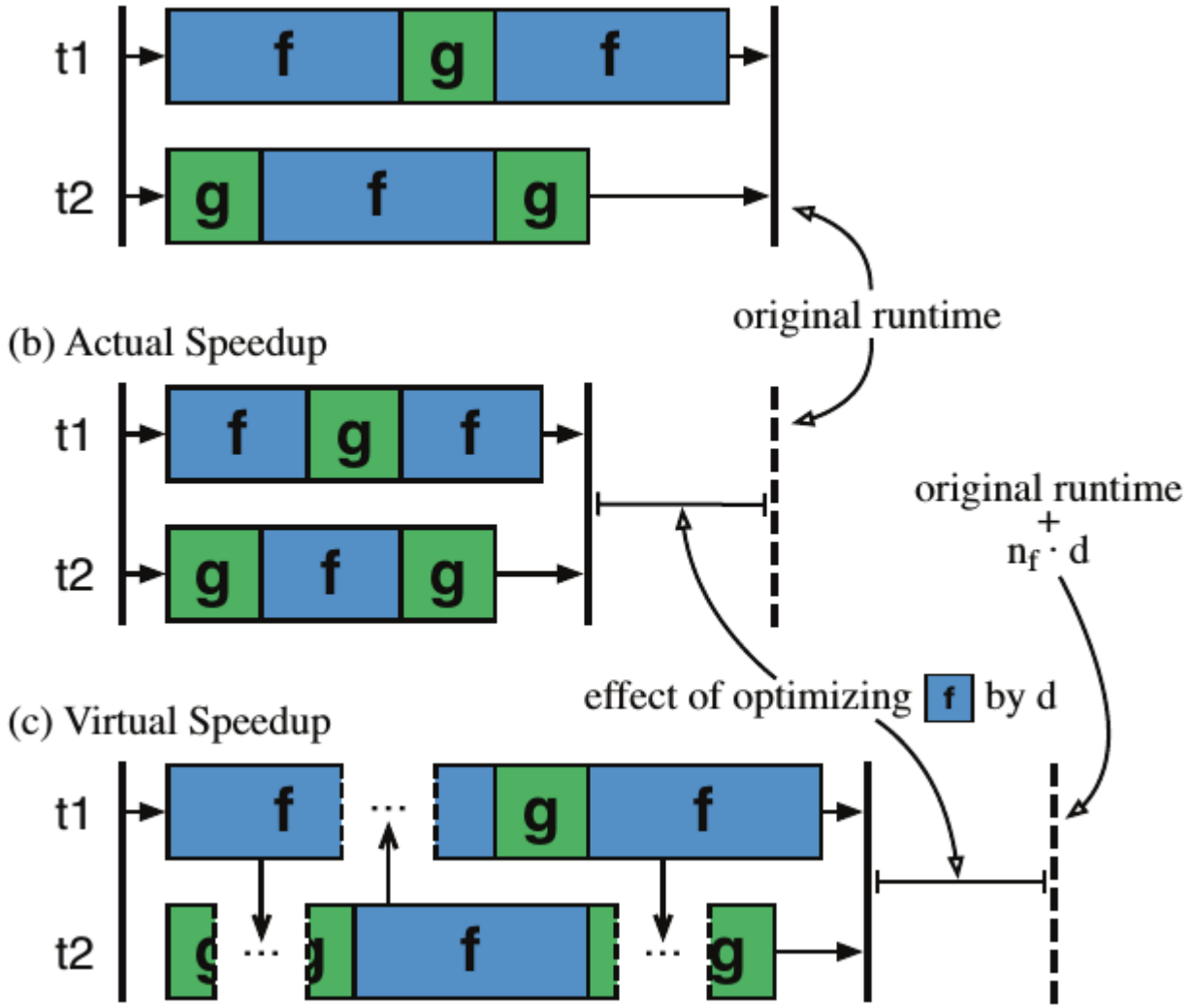


Idea

Perform *performance experiments* during program execution that measure how fast a program gets if you slow down a specific line of code



Idea

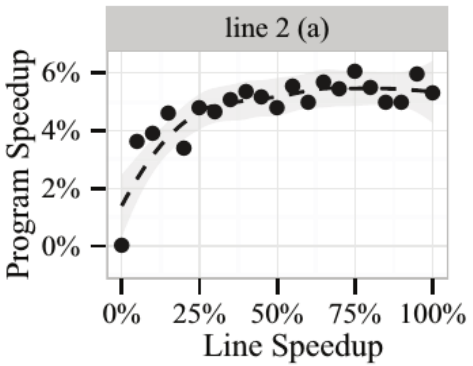

t1
f
g
f
t2
g
f
g
original runtime
(b) Actual Speedup
t1
f
g
f
t2
g
f
g
original runtime
+
$n_f \cdot d$
effect of optimizing f by d
(c) Virtual Speedup
t1
f
...
g
f
t2
g
...
g
f
...
g

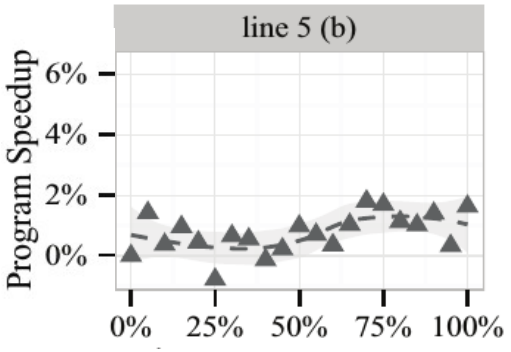
Implementation

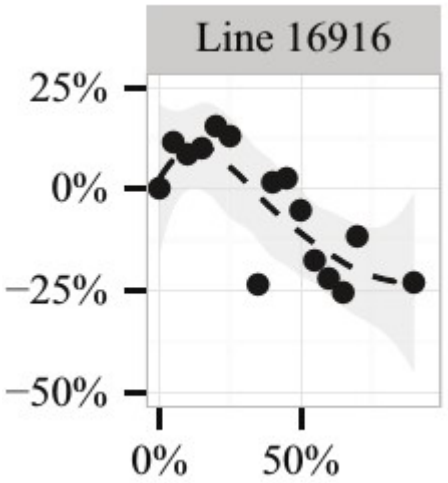
1) **Startup**

```
coz run --- <program> <args>
```

2) **Experiment Initialization**
   randomly choose source line and speedup (5% steps)
3) **Apply virtual speedup**
   sample each thread every ms, apply virtual speedup when selected source line runs
4) **Ending virtual experiment**
   after predetermined time, at least five visits to selected line, else double time
5) **Produce profile**
   analyze all logged results, calculate speedups
6) **Interpret casual profiles**







Challanges

Blocking: Who has to wait when?

→ I/O simple (delay after io finishes)

→ Synchronization primitives: hard, don't punish twice

Potentially *unblocking* calls

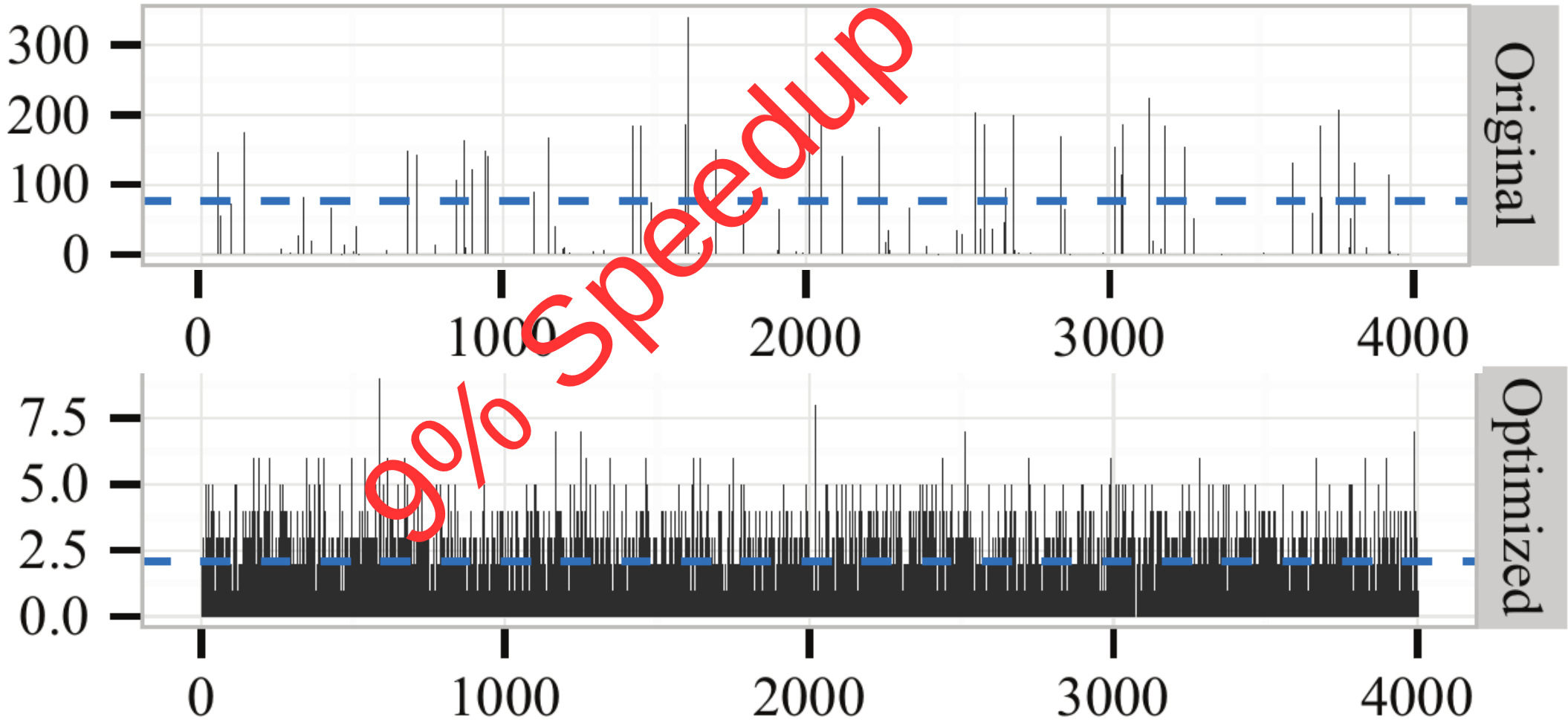
`pthread_mutex_unlock`	unlock a mutex
`pthread_cond_signal`	wake one waiter on a c.v.
`pthread_cond_broadcast`	wake all waiters on c.v.
`pthread_barrier_wait`	wait at a barrier
`pthread_kill`	send signal to a thread
`pthread_exit`	terminate this thread

Potentially *blocking* calls

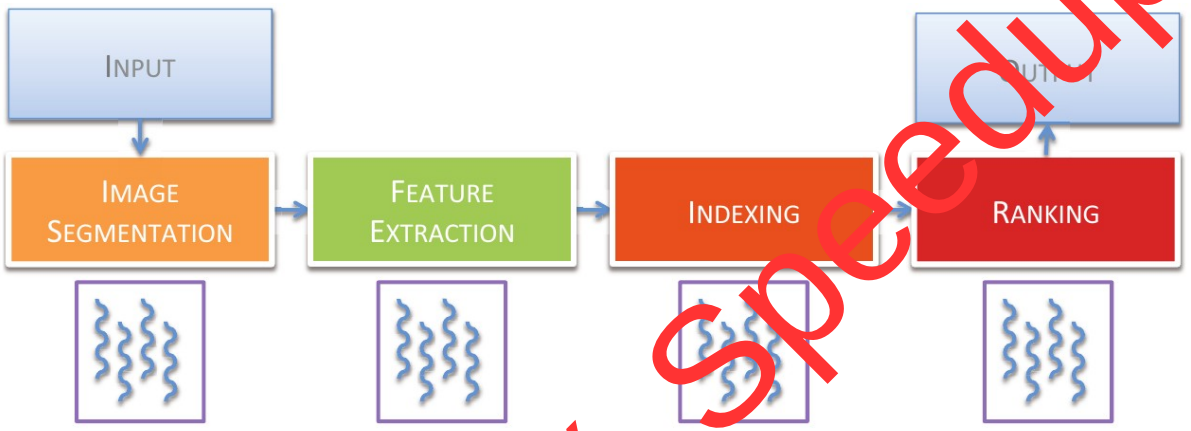
`pthread_mutex_lock`	lock a mutex
`pthread_cond_wait`	wait on a condition variable
`pthread_barrier_wait`	wait at a barrier
`pthread_join`	wait for a thread to complete
`sigwait`	wait for a signal
`sigwaitinfo`	wait for a signal
`sigtimedwait`	wait for a signal (with timeout)
`sigsuspend`	wait for a signal

Results (dedup)

While loop in `hashtable_search` function

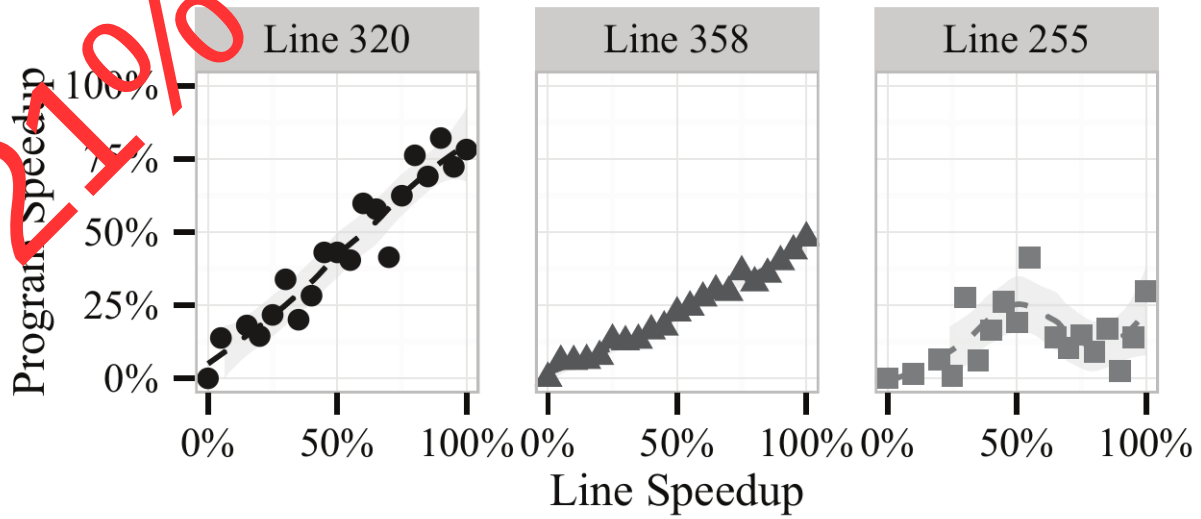


Results (ferret)

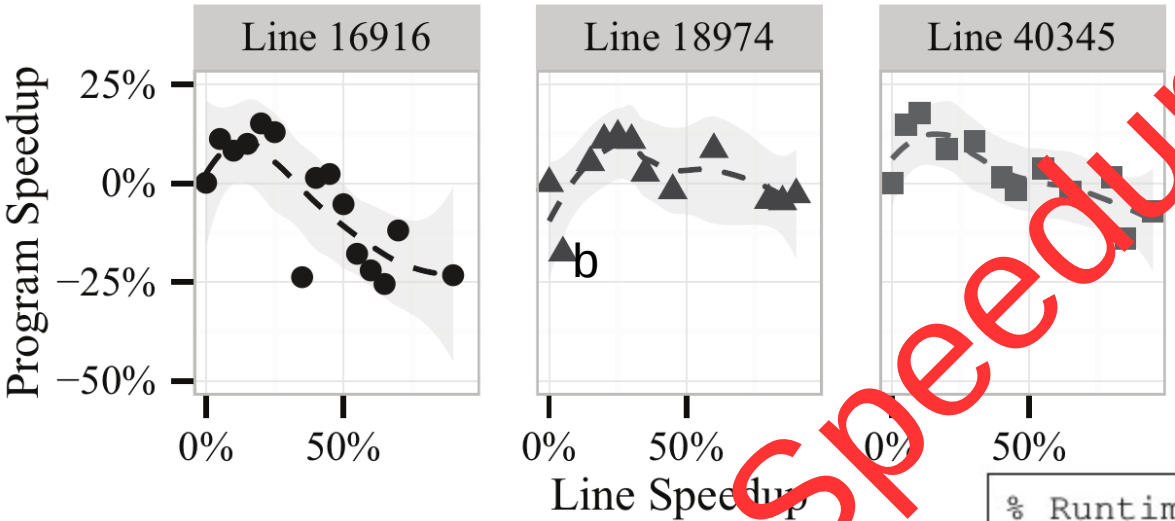


21% Speedup

Improvement opportunities in three of the four threads



Results (SQLite)



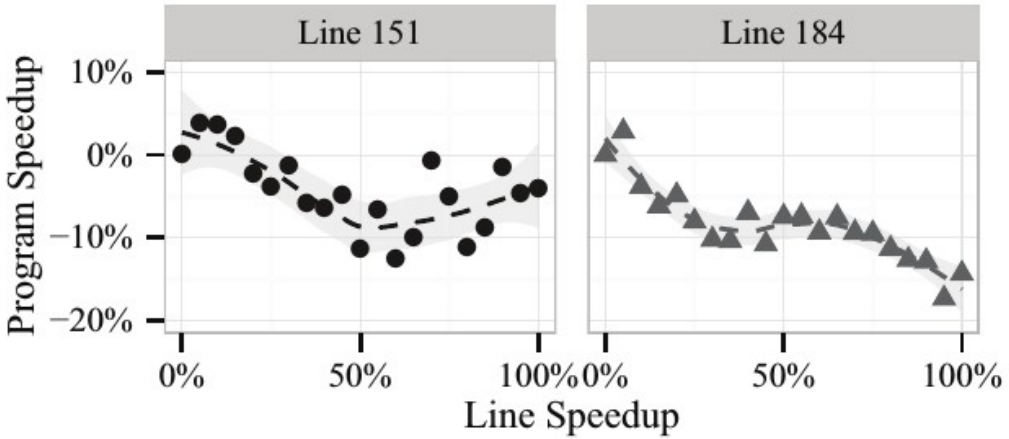
26% Speedup

% Runtime	Symbol
85.55%	_raw_spin_lock
1.76%	x86_pmu_enable_all
... 30 lines ...	
0.10%	rcu_irq_enter
0.09%	**sqlite3MemSize**
0.09%	source_load
... 26 lines ...	
0.03%	__queue_work
0.03%	**pcache1Fetch**
0.03%	kmem_cache_free
0.03%	update_cfs_rq_blocked_load
0.03%	**pthreadMutexLeave**
0.03%	sqlite3MemMalloc

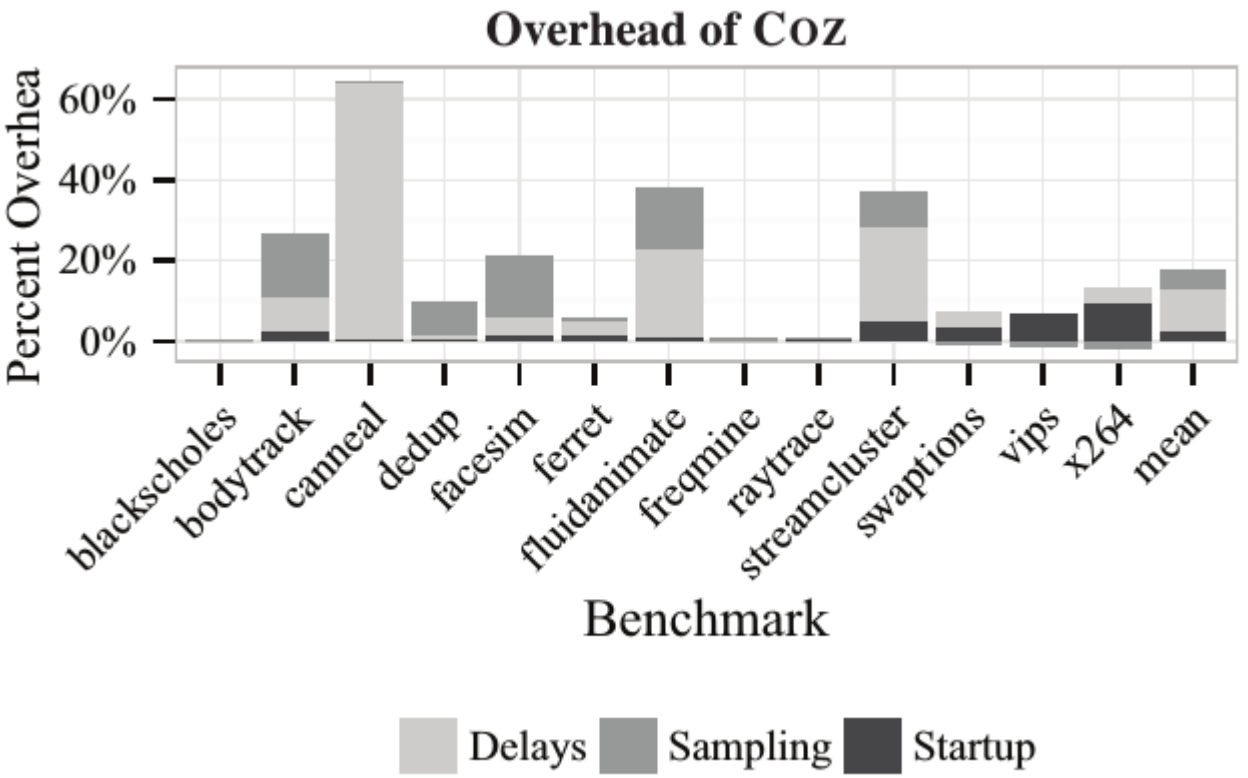
Remove unnecessary 'vtable' calls

Results (more)

Benchmark	Cause	Improvement
fluidanimate	Custom barrier	37.5%
streamcluster	Custom barrier	68%
memcached	'false' Lock sharing	9%
blackscholes	Common subexpressions	2.6%
swaptions	Inefficient array handling	15%



Efficiency


Overhead of Coz
Percent Overhea
60%
40%
20%
0%
blackscholes
bodytrack
canneal
dedup
facesim
ferret
fluidanimate
freqmine
raytrace
streamcluster
swaptions
vips
x264
mean
Benchmark
Delays
Sampling
Startup

Discussion

- Cool Idea
- How long do you have to sample for reasonable coverage of 148k lines of sqlite? (1ms, each 5 times, so 500s? How is the coverage of this?)