Performance Matters

Emery Berger & Charlie Curtsinger*

College of Information
and Computer Sciences

UMass Amherst

*now at Grinnell College
A few months ago,
in a valley far, far away...
This is Bob.
This is Bob.

Bob has an idea for a startup.
found 8,000,000 similar images
found 8,000,000 similar images
It’s going to totally disrupt image search.

found 8,000,000 similar images
The Prototype Google
The Prototype Google

Take a picture
Take a picture

Send it to Ogle

The Prototype 📷
Take a picture

Send it to Ogle

Add it to the database
Take a picture → Send it to Ogle → Add it to the database → Find similar pictures

The Prototype Ogle
The Prototype Ogle

Take a picture

found 8,000,000 similar images

Send it to Ogle

Send results

Find similar pictures

Add it to the database

The Prototype Ogle
found 8,000,000 similar images

Send results

Find similar pictures
found 8,000,000 similar images
Send results
Find similar pictures
I HAD FUN ONCE
IT WAS AWFUL
found 8,000,000 similar images

Send results

Find similar pictures

compress

save
Send results

Find similar pictures

compress

save

feature extraction
found 8,000,000 similar images

Send results

Send results

Find similar pictures

compress

save

feature extraction

search for similar pix
Send results

Find similar pictures

compress

save

feature extraction

search for similar pix

send results
found 8,000,000 similar images

Send results

Find similar pictures

compress

save

feature extraction

search for similar pix

send results
Ogle is totally disrupting image search.
Ogle is too slow!
found 8,000,000 similar images
This is also Bob.
This is also Bob. in the ‘80s
This is also Bob. 
in the ‘80s
This is also Bob. in the ‘80s
Searching database...
Searching database...

found 4 results:
Searching database...

found 4 results:
catisgumpy.bmp
Searching database...

found 4 results:
catitisgumpy.bmp
funisawful.bmp
Searching database...

found 4 results:
catisgumpy.bmp
funisawful.bmp
catdoesnotlikefun.bmp
Searching database...

found 4 results:
catisgumpy.bmp
funisawful.bmp
catdoesnotlikefun.bmp
from_dad.bmp
Searching database...

found 4 results:
catisgumpy.bmp
funisawful.bmp
catdoesnotlikefun.bmp
from_dad.bmp
Searching database...

found 4 results:
catisgumpy.bmp
funisawful.bmp
catdoesnotlikefun.bmp
from_dad.bmp

loading…

Ogle is too slow!
Graphosearch is too slow!

Ogle is too slow!
Performance used to be easy

**Transistors (millions)**

<table>
<thead>
<tr>
<th>Year</th>
<th>Transistors (millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1970</td>
<td>0.001</td>
</tr>
<tr>
<td>1975</td>
<td>0.01</td>
</tr>
<tr>
<td>1980</td>
<td>0.1</td>
</tr>
<tr>
<td>1985</td>
<td>1</td>
</tr>
<tr>
<td>1990</td>
<td>10</td>
</tr>
<tr>
<td>1995</td>
<td>100</td>
</tr>
<tr>
<td>2000</td>
<td>1,000</td>
</tr>
<tr>
<td>2005</td>
<td>10,000</td>
</tr>
<tr>
<td>2010</td>
<td>100,000</td>
</tr>
<tr>
<td>2015</td>
<td>1,000,000</td>
</tr>
</tbody>
</table>

**Clock Speed (MHz)**

<table>
<thead>
<tr>
<th>Year</th>
<th>Clock Speed (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1970</td>
<td></td>
</tr>
<tr>
<td>1975</td>
<td></td>
</tr>
<tr>
<td>1980</td>
<td></td>
</tr>
<tr>
<td>1985</td>
<td></td>
</tr>
<tr>
<td>1990</td>
<td></td>
</tr>
<tr>
<td>1995</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td></td>
</tr>
<tr>
<td>2015</td>
<td></td>
</tr>
</tbody>
</table>
Performance used to be easy

Transistors (millions)

Clock Speed (MHz)

Log scale

Year

Year
Performance used to be easy

Transistors (millions)

Clock Speed (MHz)

Smaller = Faster
Performance used to be easy

Smaller = Faster

Just buy new hardware and your code runs faster
Performance analysis in the ‘80s
Back to the present...

Ogle is too slow!
Ogle is too slow!

Back to the present...
Performance not easy anymore

Transistors (millions)

Clock Speed (MHz)
Performance not easy anymore

Transistors (millions)

Clock Speed (MHz)
Performance not easy anymore

What happened?
Performance not easy anymore

Transistors (millions)

Clock Speed (MHz)

What happened?
Too Hot!

Year
Year
Performance not easy anymore

Transistors (millions)

Clock Speed (MHz)

Transistor counts still increased

What happened?

Too Hot!
Where do all those transistors go?
Where do all those transistors go?
Instead of faster processors, we just get more.
Where do all those transistors go?

Instead of faster processors, we just get more.

Four cores
Where do all those transistors go?

Instead of faster processors, we just get more.

Four cores

Really just four separate processors
Where do all those transistors go?

Instead of faster processors, we just get more.

Four cores

Really just four separate processors
Like having four Pentium Pros from 1995
Pending Updates

Inbox by Gmail - the...  
34.1 MB  
Apr 22, 2015  
• Bug fixes and performance improvements

TripAdvisor Hotels Fl...  
55.9 MB

Yelp  
46.1 MB

Google Drive - free...  
43.0 MB

Gogobot - City Guid...  
56.3 MB
Pending Updates

- Bug fixes and performance enhancements.

Updated April 22, 2015

- Apple Watch: just get a shiny new Apple Watch? You're in luck! Add the new Gogobot app to your Apple watch to discover the best places to eat, play and stay near you!

- Bug fixes and performance enhancements!
What’s New in Fly Delta 3.3
- Introducing Fly Delta for Apple Watch – Keep track of time to departure and arrival along with gate and baggage carousel information
- Any eBoarding pass(es) you save to Passbook will automatically be available on Apple Watch
- Fixed an issue which caused the app to quit unexpectedly before boarding if a seat had not yet been assigned
- Fixed an issue which caused some customers to see an error message when attempting to view/change seats
- Several other bug fixes and performance improvements
AP Mobile
23.3 MB
Apr 20, 2015
* Support for Apple Watch! - Stay up-to-date with breaking news notifications and force touch to save a story to your phone.
* Improved story sharing experience through AP’s Bigstory.ap.org site.
* Bug fixes and the usual performance enhancements.

Thanks for your support!

Fly Delta
51.5 MB
Apr 21, 2015
What’s New in Fly Delta 3.3
* Introducing Fly Delta for Apple Watch – Keep track of time to departure and arrival along with gate and baggage carousel information
* Any eBoarding pass(es) you save to Passbook will automatically be available on Apple Watch
* Fixed an issue which caused the app to quit unexpectedly before boarding if a seat had not yet been assigned
* Fixed an issue which caused some customers to see an error message when attempting to view/change seats
* Several other bug fixes and performance improvements

Updated April 22, 2015

TripIt - Travel Organize...
15.6 MB
What’s New ▼

Movies by Flixster, with...
Updated April 22, 2015

AP Mobile
23.3 MB
Apr 20, 2015

* Support for Apple Watch
* Improved story sharing
* Bug fixes and user enhancements.

Thanks for your support.

Updated April 20, 2015

Google Play Music
13.1 MB
Apr 20, 2015

Crash fixes and general performance improvements.

Starbucks
33.7 MB
What’s New ▼

Updated April 20, 2015

Shazam
41.4 MB
What’s New ▼

Updated April 18, 2015

Instagram
20.5 MB
What’s New ▼

Movies by Fandango
49.5 MB
What’s New ▼
<table>
<thead>
<tr>
<th>Updated</th>
<th>Name</th>
<th>Size</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apr 22, 2015</td>
<td>AP Mobile</td>
<td>23.3 MB</td>
<td>Apr 20, 2015</td>
<td>* Support for Apple Watch with breaking news notifications and improvements.</td>
</tr>
<tr>
<td>Apr 16, 2015</td>
<td>Kindle</td>
<td>77.3 MB</td>
<td>Apr 15, 2015</td>
<td>Performance and Stability Improvements</td>
</tr>
<tr>
<td>Apr 13, 2015</td>
<td>Facebook</td>
<td>51.2 MB</td>
<td>Apr 12, 2015</td>
<td>What’s New ¶</td>
</tr>
<tr>
<td>Apr 11, 2015</td>
<td>Evernote</td>
<td>63.7 MB</td>
<td>Apr 11, 2015</td>
<td>What’s New ¶</td>
</tr>
<tr>
<td>Apr 10, 2015</td>
<td>Movies by Fandango</td>
<td>49.5 MB</td>
<td>Apr 10, 2015</td>
<td>What’s New ¶</td>
</tr>
<tr>
<td>Apr 9, 2015</td>
<td>TripIt</td>
<td>15.6 MB</td>
<td>Apr 9, 2015</td>
<td>What’s New ¶</td>
</tr>
<tr>
<td>Apr 8, 2015</td>
<td>Instagram</td>
<td>20.5 MB</td>
<td>Apr 7, 2015</td>
<td>What’s New ¶</td>
</tr>
<tr>
<td>Apr 7, 2015</td>
<td>Amazon Local</td>
<td>10.6 MB</td>
<td>Apr 7, 2015</td>
<td>What’s New ¶</td>
</tr>
</tbody>
</table>

*Note: ¶ indicates new features or changes in the app.*

Thanks for your support!
Thanks for using Facebook! To make our app better for you, we bring updates to the App Store every 2 weeks. You can update the app automatically (without checking back here) by going to Settings > iTunes & App Store > Automatic Downloads and turning on Updates.

Every update of our Facebook app includes improvements for speed and reliability. As other new features become available, we’ll highlight those for you in the app.
Where do all those transistors go?
Instead of faster processors, we just get more.

Four cores
Where do all those transistors go?

Instead of faster processors, we just get more.

Four cores

and a lot of extra “stuff”
Why is this so hard?
Why is this so hard?

Performance Optimization

current approaches out of steam
Why is this so hard?

Performance Analysis
  current approaches unsound

Performance Profiling
  current approaches ineffective
Why is this so hard?

Performance Analysis

*current approaches unsound*
STABILIZER

We’ve been doing it wrong
We’ve been doing it wrong
Memory layout affects performance
We’ve been doing it wrong

Memory layout affects performance.
changing a program will change its layout
We’ve been doing it wrong

Memory layout affects performance

changing a program will change its layout

no way to measure effect of change in isolation
We’ve been doing it wrong

Memory layout affects performance
changing a program will change its layout
no way to measure effect of change in isolation

STABILIZER eliminates the effect of layout
We’ve been doing it wrong

Memory layout affects performance
changing a program will change its layout
no way to measure effect of change in isolation

STABILIZER eliminates the effect of layout
enables sound performance evaluation
We’ve been doing it wrong

Memory layout affects performance
changing a program will change its layout
no way to measure effect of change in isolation

STABILIZER eliminates the effect of layout
enables sound performance evaluation

Case Studies
STABILIZER

We’ve been doing it wrong

Memory layout affects performance
changing a program will change its layout
no way to measure effect of change in isolation

STABILIZER eliminates the effect of layout
enables sound performance evaluation

Case Studies
evaluation of LLVM’s optimizations with STABILIZER
We’ve been doing it wrong

Memory layout affects performance
changing a program will change its layout
no way to measure effect of change in isolation

STABILIZER eliminates the effect of layout
enables sound performance evaluation

Case Studies
evaluation of LLVM’s optimizations with STABILIZER
Unsound performance evaluation
Unsound performance evaluation
Unsound performance evaluation
Unsound performance evaluation

```c
int main(int argc, char **argv) {
  topframe = (void**)__builtin_frame_address(0);
  setHandler(Trap::TrapSignal, onTrap);
  setHandler(SIGALRM, onTimer);
  setHandler(SIGSEGV, onFault);
  for(Function* f: functions) {
    f->setTrap();
  }
  setTimer(interval);
  int r = stabilizer_main(argc, argv);
  return r;
}

void setTimer(int msec) {
  struct itimerval timer;
  timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
  timer.it_value.tv_usec = 1000 * (msec % 1000);
  timer.it_interval.tv_sec = 0;
  timer.it_interval.tv_usec = 0;
  setitimer(ITIMER_REAL, &timer, 0);
}

static void flush_icache(void* begin, size_t size) {
  uintptr_t p = (uintptr_t)begin & ~15UL;
  size_t meaning_of_life = 42;
  for (size_t i = 0; i < size; i += 32) {
    asm("icbi 0,%0" : : "r"(p));
    p += 32;
  }
  for (size_t i = 16; i < size; i += 32) {
    asm("icbi 0,%0" : : "r"(p));
    p += 32;
  }
  asm("isync");
}

DataHeapType* getDataHeap() {
  static char buf[sizeOf(DataHeapType)];
  static DataHeapType* _theDataHeap = new (buf) DataHeapType;
  return _theDataHeap;
}
```
Unsound performance evaluation

```cpp
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}

void setTimer(int msec) {
    struct itimerval timer;
    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &timer, 0);
}

DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}

void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```
Unsound performance evaluation

```c
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}

DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}

void setTimer(int msec) {
    struct itimerval timer;
    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &timer, 0);
}

static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}

static void flush_ilcache(void* begin, size_t size) {
    int i = 0;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}

void settimer(int msec) {
    struct itimerval timer;
    timer.it_value.tv_sec = (msec / 1000) / 10; // msec / 1000
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &timer, 0);
}
```
A

Unsound performance evaluation

```c
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}

DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}

void setTimer(int msec) {
    struct itimerval timer;
    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &timer, 0);
}

static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}

void setTimer(int msec) {
    struct timerreal timer;
    timer.tv_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.tv_value.tv_usec = 1000 * (msec % 1000);
    timer.tv_interval.tv_sec = 0;
    timer.tv_interval.tv_usec = 0;
    settimer(TIMER_REAL, &timer, 0);}
```
Unsound performance evaluation

dataMapType getDataHeap() {
  static char *buf = (void*)__builtin_frame_address(0);
  static DataHeapType _theDataHeap = new (buf) DataHeapType;
  return _theDataHeap;
}

int main(int argc, char **argv) {
  setHandler(Trap::TrapSignal, onTrap);
  setHandler(SIGALRM, onTimer);
  setHandler(SIGSEGV, onFault);
  for(Function* f: functions) {
    f->setTrap();
  }
  setTimer(interval);
  int r = stabilizer_main(argc, argv);
  return r;
}

DataHeapType* getDataHeap() {
  static char buf[sizeof(DataHeapType)];
  static DataHeapType* _theDataHeap = new (buf) DataHeapType;
  return _theDataHeap;
}

void setTimer(int msec) {
  struct itimerval timer;
  timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
  timer.it_value.tv_usec = 1000 * (msec % 1000);
  timer.it_interval.tv_sec = 0;
  timer.it_interval.tv_usec = 0;
  setitimer(ITIMER_REAL, &timer, 0);
}

static void flush_icache(void* begin, size_t size) {
  uintptr_t p = (uintptr_t)begin & ~15UL;
  size_t meaning_of_life = 42;
  for (size_t i = 0; i < size; i += 32) {
    asm("icbi 0,%0" : : "r"(p));
    p += 32;
  }
  for (size_t i = 16; i < size; i += 32) {
    asm("icbi 0,%0" : : "r"(p));
    p += 32;
  }
  asm("isync");
}

void setTimer(int msec) {
  struct itimerval timer;
  timer.it_value.tv_sec = (msec / 1000) / 1000;
  timer.it_value.tv_usec = 1000 * (msec % 1000);
  timer.it_interval.tv_sec = 0;
  timer.it_interval.tv_usec = 0;
  setitimer(ITIMER_REAL, &timer, 0);
}
Which is faster?

![Graph comparing time for versions A and A']
Is $A'$ faster than $A$?
Is $A'$ faster than $A$?
Is $A'$ faster than $A$?
Is $A'$ faster than $A$?
Is $A'$ faster than $A$?

$A'$ is 2.8% faster.
Is $A'$ faster than $A$?

what about variance?

2.8% faster
Which is faster?

A ×30

A′ ×30

[Bar chart showing number of runs over time for versions A and A′]
Is $A'$ faster than $A$?
Is A' faster than A?

still 2.8% faster
Why is $A'$ faster than $A$?
Why is A’ faster than A?
Why is $A'$ faster than $A$?

Was it the code change?
Why is $A'$ faster than $A$?

Was it the code change?
Why is $A'$ faster than $A$?

Or was it the new layout?
Why is \( A' \) faster than \( A \)?

Or was it the new layout?
Why is $A'$ faster than $A$?

Layout biases measurement

Mytkowicz et al. (ASPLOS’09)
Layout biases measurement
Mytkowicz et al. (ASPLOS’09)

Link Order
Changes function addresses
Layout biases measurement

Mytkowicz et al. (ASPLOS’09)

Link Order
Changes function addresses

Environment

Variable Size
Moves the program stack
Layout biases measurement

Mytkowicz et al. (ASPLOS’09)

**Link Order**
Changes function addresses

**Larger than the impact of -O3**

**Environment**

**Variable Size**
Moves the program stack
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}

void setTimer(int msec) {
    struct itimerval timer;
    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &timer, 0);
}

static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}

DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapTyper _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}

void setTimer(int msec) {
    struct itimerval timer;
    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &timer, 0);
}

static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}

DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
int main(int argc, char **argv) {
  topFrame = (void**)__builtin_frame_address(0);
  setHandler(Trap::TrapSignal, onTrap);
  setHandler(SIGALRM, onTimer);
  setHandler(SIGSEGV, onFault);
  for(Function* f: functions) {
    f->setTrap();
  }
  setTimer(interval);
  int r = stabilizer_main(argc, argv);
  return r;
}

void setTimer(int msec) {
  struct itimerval timer;
  timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
  timer.it_value.tv_usec = 1000 * (msec % 1000);
  timer.it_interval.tv_sec = 0;
  timer.it_interval.tv_usec = 0;
  setitimer(ITIMER_REAL, &timer, 0);
}

static void flush_icache(void* begin, size_t size) {
  uintptr_t p = (uintptr_t)begin & ~15UL;
  for (size_t i = 0; i < size; i += 16) {
    asm("icbi 0,%0" : : "r"(p));
    p += 16;
  }
  asm("isync");
}

DataHeapType* getDataHeap() {
  static char buf[sizeof(DataHeapType)];
  static DataHeapType* _theDataHeap = new (buf) DataHeapType;
  return _theDataHeap;
}
Blame the cache
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}

DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}

void setTimer(int msec) {
    struct itimerval timer;
    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &timer, 0);
}

void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}

static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}

DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}

static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}

DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}

void setTimer(int msec) {
    struct itimerval timer;
    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &timer, 0);
}

DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}

void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}

map to same set
map to same set

Nothing here
Blame the cache

map to
same set

Nothing here
no conflict
Blame the cache

```
int main(int argc, char **argv) {
  topFrame = (void**)__builtin_frame_address(0);
  setHandler(Trap::TrapSignal, onTrap);
  setHandler(SIGALRM, onTimer);
  setHandler(SIGSEGV, onFault);
  for(Function* f: functions) {
    f->setTrap();
  }
  setTimer(interval);
  int r = stabilizer_main(argc, argv);
  return r;
}

static void flush_icache(void* begin, size_t size) {
  uintptr_t p = (uintptr_t)begin & ~15UL;
  for (size_t i = 0; i < size; i += 16) {
    asm("icbi 0,%0" : : "r"(p));
    p += 16;
  }
  asm("isync");
}

DataHeapType* getDataHeap() {
  static char buf[sizeof(DataHeapType)];
  static DataHeapType* _theDataHeap = new (buf) DataHeapType;
  return _theDataHeap;
}

void setTimer(int msec) {
  struct itimerval timer;
  timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
  timer.it_value.tv_usec = 1000 * (msec % 1000);
  timer.it_interval.tv_sec = 0;
  timer.it_interval.tv_usec = 0;
  setitimer(ITIMER_REAL, &timer, 0);
}
```

DataHeapType* getDataHeap() {  
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
  }

int main(int argc, char **argv) {
  topFrame = (void*)__builtin_frame_address(0);
  setHandler(Trap::TrapSignal, onTrap);
  setHandler(SIGALRM, onTimer);
  setHandler(SIGSEGV, onFault);
  for(Function* f: functions) {
    f->setTrap();
  }
  setTimer(interval);
  int r = stabilizer_main(argc, argv);
  return r;
}

void setTimer(int msec) {
    struct itimerval timer;
    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &timer, 0);
}
int main(int argc, char **argv) {
topFrame = (void**)__builtin_frame_address(0);
setHandler(Trap::TrapSignal, onTrap);
setHandler(SIGALRM, onTimer);
setHandler(SIGSEGV, onFault);
for(Function* f: functions) {
f->setTrap();
}
setTimer(interval);
t = stabilizer_main(argc, argv);
return t;
}

void setTimer(int msec) {
struct itimerval timer;
timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
timer.it_value.tv_usec = 1000 * (msec % 1000);
timer.it_interval.tv_sec = 0;
timer.it_interval.tv_usec = 0;
setitimer(ITIMER_REAL, &timer, 0);
}

static void flush_icache(void* begin, size_t size) {
uintptr_t p = (uintptr_t)begin & ~15UL;
for(size_t i = 0; i < size; i += 16) {
asm("icbi 0,%0" : : "r"(p));
p += 16;
}
asm("isync");
}

DataHeapType* getDataHeap() {
static char buf[sizeof(DataHeapType)];
static DataHeapType* _theDataHeap = new (buf) DataHeapType;
return _theDataHeap;
}

DataHeapType* getDataHeap() {
static char buf[sizeof(DataHeapType)];
static DataHeapType* _theDataHeap = new (buf) DataHeapType;
return _theDataHeap;
}
Blame the cache or branch predictor

```c
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}

static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}

DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}

void setTimer(int msec) {
    struct itimerval timer;
    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &timer, 0);
}
```

or TLB

```c
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```
Blame the cache or branch predictor or branch target predictor.
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}

void setTimer(int msec) {
    struct itimerval timer;
    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &timer, 0);
}

DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}

static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}

void setTimer(int msec) {
    struct itimerval timer;
    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &timer, 0);
}

DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
A

int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}

void setTimer(int msec) {
    struct itimerval timer;
    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &timer, 0);
}

DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}

void setTimer(int msec) {
    struct itimerval timer;
    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &timer, 0);
}

DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
Is $A'$ faster than $A$?
Is $A'$ faster than $A$?

Let's do a poll

it's faster
Is $A'$ faster than $A$?

Let's do a poll

- It's faster
- It's faster
- It's faster
- It's faster
- It's faster
- It's faster
Do we trust this?

it's faster

it's faster

it's faster

it's faster
Is $A'$ faster than $A$?
Is $A'$ faster than $A$?

- it's faster
- it's slower
- they're the same
But it ran faster!
What if we only talk to Bob?

- it’s faster
- it’s slower
- they’re the same
But it ran faster!
What if we only use this layout?

it's faster
it's slower
they're the same
But it ran faster!

What if we only use this layout?

it’s faster
But it ran faster!

What if we only use this layout?

it’s faster

Upgrade libc
But it ran faster!
What if we only use this layout?

it's faster

Upgrade libc
Changes layout
But it ran faster!
What if we only use this layout?

it’s faster

Change Username
But it ran faster!
What if we only use this layout?

it’s faster

Change Username
Changes layout
But it ran faster!
What if we only use this layout?

it’s faster

Run in a new directory
But it ran faster!
What if we only use this layout?

it’s faster

Run in a new directory
Changes layout
But it ran faster!
What if we only use this layout?

Layout is Brittle
But it ran faster!
What if we only use this layout?

Layout is Brittle

Layout biases measurement
Mytkowicz et al. (ASPLOS’09)
But it ran faster!
What if we only use this layout?

Layout is Brittle

Layout biases measurement
Mytkowicz et al. (ASPLOS’09)

Can we eliminate the effect of layout?
But it ran faster!
What if we only use this layout?

Layout biases measurement
Can we eliminate the effect of layout?

YES
Memory layout affects performance

makes performance evaluation difficult

STABILIZER eliminates the effect of layout

enables sound performance evaluation

Case Studies

evaluation of LLVM's optimizations with STABILIZER
STABILIZER

Memory layout affects performance makes performance evaluation difficult.

STABILIZER eliminates the effect of layout enables sound performance evaluation.

Case Studies

evaluation of LLVM’s optimizations with STABILIZER.
STABILIZER randomizes layout

Layout biases measurement
STABILIZER

randomizes layout

function addresses

Layout biases measurement
STABILIZER

randomizes layout

function addresses    stack frame sizes

Layout biases measurement
STABILIZER

randomizes layout

function addresses  stack frame sizes
heap allocations

Layout biases measurement
STABILIZER

repeatedly randomizes layout

function addresses  stack frame sizes
heap allocations

Layout biases measurement
STABILIZER repeatedly randomizes layout
function addresses stack frame sizes
heap allocations

Layout biases measurement
during execution
STABILIZER

repeatedly randomizes layout

function addresses    stack frame sizes
heap allocations

Layout biases measurement
STABILIZER

repeatedly randomizes layout

function addresses  stack frame sizes
heap allocations

Layout biases measurement

a completely random layout cannot bias results
Sound Performance Evaluation

A $\times 30$  A' $\times 30$
Sound Performance Evaluation

A ×30

A’ ×30

Percent of Observed Runtimes

Time (s)
Is $A'$ faster than $A$?

![Graph showing distribution of runtimes]

- Time (s)
- Percent of Observed Runtimes

The graph compares the distribution of runtimes for $A$ and $A'$. The distribution for $A'$ is shifted to the left, indicating faster runtimes compared to $A$. The exact runtime comparison is not specified in the graph.
Is \( A' \) faster than \( A \)?
Is $A'$ faster than $A$?
Is $A'$ faster than $A$?
Is $A'$ faster than $A$?

The graph shows a histogram with the percentile distribution of observed runtimes. The distribution for $A'$ is shifted to the left compared to $A$, indicating that $A'$ is faster. The sign in the middle states "WRONG WAY".
The Statistical Approach

If $A' = A$
The Statistical Approach

*Hypothesis testing*

If \( A' = A \)
The Statistical Approach

If \( A' = A \)

what is the probability of measuring a speedup this large by chance?
If \( A' = A \)

easy to compute for the normal distribution

what is the **probability** of measuring a speedup this large by chance?
If \[ A' = A \]

easy to compute for the normal distribution
STABILIZER

repeatedly randomizes layout

what is the probability of measuring a speedup this large by chance?
**STABILIZER**

_repeatedly randomizes layout_

what is the probability of measuring a speedup this large by chance?

if there is a low probability
STABILIZER repeatedly randomizes layout

this speedup is real
STABILIZER repeatedly randomizes layout

this speedup is real

not due to the effect on memory layout
STABILIZER

repeatedly randomizes layout

this speedup is real

not due to the effect on memory layout

what does re-randomization do?
STABILIZER

repeatedly randomizes layout
STABILIZER

repeatedly randomizes layout

one random layout per-run
**STABILIZER**

_repeatedly randomizes layout_

---

![Graph with probability density and time axis]

- **Probability Density**
- **Time (s)**

---

*one random layout per-run*
Stabilizer

repeatedly randomizes layout

many random layouts in each run

one random layout per-run
STABILIZER
repeatedly randomizes layout
**STABILIZER**

repeatedly randomizes layout

**STABILIZER generates a new random layout every \( \frac{1}{2} \) second**
STABILIZER generates a new random layout every \( \frac{1}{2} \) second.

Total execution time is the sum of all periods.
STABILIZER generates a new random layout every $\frac{1}{2}$ second. Total execution time is the sum of all periods. The sum.
STABILIZER

repeatedly randomizes layout

STABILIZER generates a new random layout every $\frac{1}{2}$ second

Total execution time is the sum of all periods

The sum of a sufficient number
**STABILIZER** repeatedly randomizes layout

**STABILIZER** generates a new random layout every $\frac{1}{2}$ second.

Total execution time is the sum of all periods.

The sum of a sufficient number of independent, identically distributed random variables.
Central Limit Theorem

The sum of a sufficient number of independent, identically distributed random variables is approximately normally distributed.
Central Limit Theorem

execution times are normally distributed.

The sum of a sufficient number of independent, identically distributed random variables is approximately normally distributed.
Memory layout affects performance makes performance evaluation difficult

**STABILIZER** eliminates the effect of layout enables sound performance evaluation

Case Studies
evaluation of LLVM’s optimizations with **STABILIZER**
Memory layout affects performance makes performance evaluation difficult

STABILIZER eliminates the effect of layout enables sound performance evaluation

Case Studies
evaluation of LLVM’s optimizations with STABILIZER
Case Studies

evaluation of LLVM's optimizations with STABILIZER
Case Studies

evaluation of LLVM’s optimizations with STABILIZER on each benchmark
Case Studies

evaluation of LLVM’s optimizations with STABILIZER

on each benchmark
across the whole benchmark suite
Case Studies

evaluation of LLVM’s optimizations with STABILIZER

on each benchmark

across the whole benchmark suite

first, build benchmarks with STABILIZER
Build programs with **STABILIZER**
Build programs with STABILIZER

> szc main.c
Build programs with STABILIZER

> szc main.c
Build programs with STABILIZER

> szc -Rcode main.c
Build programs with **STABILIZER**

```sh
> szc -Rcode -Rheap -Rstack main.c
```
Build programs with **STABILIZER**

```
> szc -Rcode -Rheap -Rstack main.c
```

now run the benchmarks
Run benchmarks as usual

\[ A \times 30 \quad A' \times 30 \]
Run benchmarks as usual

\[ A \times 30 \quad A' \times 30 \]

drop the results into R
Run benchmarks as usual

\[ A \times 30 \quad A' \times 30 \]

drop the results into R
Is $A'$ faster than $A$?
Is $A'$ faster than $A$?

![Diagram showing comparison of runtimes between $A'$ and $A$]

- Time (s)
- Percent of Observed Runtimes

**Wrong Way**
If A' = A
If $A' = A$
If $A' = A$

what is the **probability** of measuring a difference at least this large?
The Student’s t-test

If \( A' = A \)

what is the probability of measuring a difference at least this large?
The Student’s t-test

\[ t.\text{test}(\text{times.A}', \text{times.A}) \]

If \( A' = A \)

what is the \textit{probability} of measuring a difference at least this large?
The Student’s t-test

If \( p\)-value

what is the \textit{probability} of measuring a difference at least this large?
The Student's t-test

If p-value ≤ 5%

If \( A' = A \)

what is the probability of measuring a difference at least this large?
The Student’s t-test

If p-value \leq 5\%

95\% Confidence

If \( A' = A \)

what is the probability of measuring a difference at least this large?
The Student's t-test

If p-value $\leq 5\%$
we reject the null hypothesis

If $A' = A$
what is the probability of measuring a difference at least this large?
The Student's t-test
If p-value $\leq 5\%$
we reject the null hypothesis

$A' \neq A$

Random chance not responsible for the measured difference
The difference is real

$A' \neq A$
Speedup of -O2 over -O1

Significant
Yes
No

Speedup

libquantum  milc  bzip2  sphinx3  nand  lbm  perbench  hmmer  h264ref  cactusADM  wrf  sjeng  gobmk  gromacs  zeusmp  mcf  gcc  astar
Speedup of -O2 over -O1

Significant
Yes
No
Speedup of -O2 over -O1

Significant

Yes
No

Speedup of -O2 over -O1

libquantum  milc  bzip2  sphinx3  nand  lbm  perlbench  hminer  h264ref  cactusADM  wrf  sjeng  gobmk  gromacs  zeusmp  mcf  gcc  astar

-10%  0%  10%  20%
Speedup of -O2 over -O1

- Significant
  - Yes
  - No

- libquantum, milc, bzip2, sphinx3, nand, lbm, perlbench, hmmner, h264ref, cactusADM, wrf, sjeng, gobmk, gromacs, zeusmp, mcf, gcc, astar

- Speedup of -O2 over -O1
Speedup of \(-O3\) over \(-O2\)

- Significant
  - Yes
  - No
Speedup of -O3 over -O2

Significant

Yes
No
Speedup of -O3 over -O2
Speedup of -O3 over -O2
Speedup of -O3 over -O2

Significant

Yes

No

Speedup
What do the results mean?
Comparing optimizations

-02 ×30

-03 ×30
Comparing optimizations

-\texttt{O2} \times 30 \quad \text{\texttt{lbm}} \times 30

-\texttt{O3} \times 30 \quad \text{\texttt{lbm}} \times 30
Comparing optimizations

-\( -O2 \times 30 \)  
  \( \text{lbm} \times 30 \)  
  \( \text{astar} \times 30 \)

-\( -O3 \times 30 \)  
  \( \text{lbm} \times 30 \)  
  \( \text{astar} \times 30 \)
Comparing optimizations

-\texttt{O2} \times 30 \quad -\texttt{O3} \times 30

\texttt{lbm} \times 30 \quad \texttt{lbm} \times 30

\texttt{astar} \times 30 \quad \texttt{astar} \times 30

\ldots
Comparing optimizations

-02 ×30  -O3 ×30
Is -O3 faster than -O2?

The graph shows the distribution of runtime percentages for different versions. The x-axis represents time (s) ranging from 0 to 100, and the y-axis represents the percent of observed runtimes ranging from 0% to 4%. The two versions, -O2 and -O3, are compared across different times. The graph indicates that -O3 generally has a lower percent of observed runtimes, suggesting it is faster than -O2.
Is -O3 faster than -O2?
Is -O3 faster than -O2?
Is -O3 faster than -O2?

WRONG WAY
If $-O3 = -O2$
If \(-O3\) = \(-O2\)
If $-\text{O3} = -\text{O2}$

what is the probability of measuring these differences?
If \(-O3 = -O2\)

what is the probability of measuring these differences?
Analysis of Variance

\[ \text{aov}(\text{time} \sim \text{opt} + \text{Error(benchmark/opt)}, \text{times}) \]

If \(-O3 = -O2\)

what is the probability of measuring these differences?
Analysis of Variance

\[ \text{If } p\text{-value} \]

If \(-O3 = -O2\)

what is the probability of measuring these differences?
Analysis of Variance

If p-value ≤ 5%

If -O3 = -O2

what is the probability of measuring these differences?
Analysis of Variance

If p-value \leq 5%
we reject the null hypothesis

If \[ -O3 = -O2 \]

what is the probability of measuring these differences?
Analysis of Variance

If p-value \leq 5% we reject the null hypothesis

-O3 vs -O2
Analysis of Variance

If \( p\)-value \( \leq 5\% \)
we reject the null hypothesis

\[-O3 \text{ vs } -O2\]

\( p\)-value = 26.4\%
Analysis of Variance

If p-value \( \leq 5\% \) we reject the null hypothesis

\[-O3 \quad \text{vs} \quad -O2\]

p-value = 26.4\%

one in four experiments will show an effect that does not exist!
Analysis of Variance

If $p$-value $\leq 5\%$
we reject the null hypothesis

-03 vs -02

$p$-value $= 26.4\%$

fail to reject the null hypothesis
Analysis of Variance

If p-value $\leq 5\%$
we reject the null hypothesis

The effect of \text{-O3} over \text{-O2} is indistinguishable from noise
Analysis of Variance

If p-value $\leq 5\%$
we reject the null hypothesis

The effect of $-O_3$ over $-O_2$ is indistinguishable from noise

Did STABILIZER hide the effect?
Memory layout affects performance makes performance evaluation difficult

STABILIZER eliminates the effect of layout random layout enables sound performance evaluation

Case Studies showed that -O3 does not have a statistically significant effect across our benchmarks
Memory layout affects performance makes performance evaluation difficult

STABILIZER eliminates the effect of layout random layout enables sound performance evaluation

Case Studies showed that -O3 does not have a statistically significant effect across our benchmarks
STABILIZER

Memory layout affects performance
makes performance evaluation difficult

STABILIZER eliminates the effect of layout
random layout enables sound performance evaluation

Case Studies
showed that -O3 does not have a statistically
significant effect across our benchmarks
STABILIZER

Memory layout affects performance
makes performance evaluation difficult

STABILIZER eliminates the effect of layout
random layout enables sound performance evaluation

Case Studies
showed that -O3 does not have a statistically significant effect across our benchmarks
found 8,000,000 similar images
found 8,000,000 similar images
Software Profilers
Software Profilers

<table>
<thead>
<tr>
<th>%</th>
<th>cumulative</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>seconds</td>
</tr>
<tr>
<td>20.05</td>
<td>8.02</td>
</tr>
<tr>
<td>9.56</td>
<td>3.82</td>
</tr>
<tr>
<td>19.95</td>
<td>7.98</td>
</tr>
<tr>
<td>45.19</td>
<td>11.31</td>
</tr>
<tr>
<td>5.25</td>
<td>2.10</td>
</tr>
</tbody>
</table>
Software Profilers

Number of calls to each function

<table>
<thead>
<tr>
<th>%</th>
<th>cumulative time (seconds)</th>
<th>calls</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.05</td>
<td>8.02</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9.56</td>
<td>3.82</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>19.95</td>
<td>7.98</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>45.19</td>
<td>11.31</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5.25</td>
<td>2.10</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
## Software Profilers

### Runtime for each function

<table>
<thead>
<tr>
<th>%</th>
<th>cumulative</th>
<th>time</th>
<th>seconds</th>
<th>calls</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.05</td>
<td></td>
<td>8.02</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9.56</td>
<td></td>
<td>3.82</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>19.95</td>
<td></td>
<td>7.98</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>45.19</td>
<td></td>
<td>11.31</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5.25</td>
<td></td>
<td>2.10</td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

### Number of calls to each function
Software Profilers

<table>
<thead>
<tr>
<th>%</th>
<th>cumulative</th>
<th>time</th>
<th>seconds</th>
<th>calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.05</td>
<td>8.02</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.56</td>
<td>3.82</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19.95</td>
<td>7.98</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45.19</td>
<td>11.31</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.25</td>
<td>2.10</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Software Profilers

Frequently executed code

<table>
<thead>
<tr>
<th>%</th>
<th>cumulative time</th>
<th>seconds</th>
<th>calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.05</td>
<td>8.02</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9.56</td>
<td>3.82</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>19.95</td>
<td>7.98</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>45.19</td>
<td>11.31</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5.25</td>
<td>2.10</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Software Profilers

Frequently executed code

Code that runs for a long time

<table>
<thead>
<tr>
<th>%</th>
<th>cumulative time</th>
<th>cumulative seconds</th>
<th>calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.05</td>
<td>8.02</td>
<td>8.02</td>
<td>1</td>
</tr>
<tr>
<td>9.56</td>
<td>3.82</td>
<td>3.82</td>
<td>1</td>
</tr>
<tr>
<td>19.95</td>
<td>7.98</td>
<td>7.98</td>
<td>1</td>
</tr>
<tr>
<td>45.19</td>
<td>11.31</td>
<td>11.31</td>
<td>1</td>
</tr>
<tr>
<td>5.25</td>
<td>2.10</td>
<td>2.10</td>
<td>1</td>
</tr>
</tbody>
</table>
Are these places where Bob should focus on performance?

Frequently executed code

Code that runs for a long time
Would this speed up Ogle?
Would this speed up Ogle?
Would this speed up Ogle?
Would this speed up Ogle?
Would this speed up Ogle?

Frequently executed code
Would this speed up Ogle?

Frequently executed code

Code that runs for a long time
Would this speed up Ogle?

Frequently executed code

Code that runs for a long time

Profilers do a bad job finding important code in parallel programs.
Would this speed up Ogle?

Frequently executed code

Code that runs for a long time

Profilers do a bad job finding important code in parallel programs.

We need better tools.
What would speed up Ogle?
What would speed up Ogle?
What *would* speed up Ogle?

Hey, look over here
What would speed up Ogle?

Hey, look over here

What would this information look like?
Causal Profile
Causal Profile

Tells you where optimizations will make a difference
Causal Profile
Tells you where optimizations will make a difference
Causal Profile
Tells you where optimizations will make a difference
Causal Profile
Tells you where optimizations will make a difference

Program Speedup

Speedup
Causal Profile
Tells you where optimizations will make a difference

Program Speedup vs. Speedup

?-? Speedup
Causal Profile

Tells you where optimizations will make a difference
Causal Profile

Tells you where optimizations will make a difference
Causal Profile

Program Speedup

Speedup
If you speed up this much
If you speed up this much
Causal Profile

If you speed up this much

The program will run this much faster
Causal Profile

How do we know
Causal Profile

How do we know this change?
Causal Profile

How do we know this change causes this effect?
How do we know this change causes this effect?

Run an experiment
Performance Experiments
Performance Experiments

If we could magically speed up ...
Performance Experiments

If we could magically speed up ...
Performance Experiments

If we could magically speed up... 

Speeding up by this much...
If we could magically speed up …

Performance Experiments

Speeding up …

by this much…

speeds up the program by this much.
Performance Experiments

If we could magically speed up ...
Performance Experiments

If we could magically speed up ...
Performance Experiments

If we could magically speed up ...

More speedup in ... leads to a larger program speedup.
Performance Experiments

If we could magically speed up ...
Performance Experiments

If we could magically speed up...
Performance Experiments

If we could magically speed up... 

No program speedup
Performance Experiments

We’re going to have to do this without magic.
Performance Experiments

We’re going to have to do this without magic.
Performance Experiments

We’re going to have to do this without magic.

Otherwise we’d just do this.
Virtual Speedup

“Speed up” by slowing everything else down.
Virtual Speedup

“Speed up” by slowing everything else down.

Speeding up by this much...
Virtual Speedup

“Speed up” by slowing everything else down.

Speeding up by this much... speeds up the program by this much.
Speedup Results

Program Speedup

Speedup

?
Speedup Results

Program Speedup

[Graph showing speedup results with a single data point]
Virtual Speedup

“Speed up” by slowing everything else down.
Virtual Speedup

“Speed up” by slowing everything else down.
Virtual Speedup

“Speed up” by slowing everything else down.

A larger speedup has no additional effect
Speedup Results

Program Speedup

\(?\) Speedup
Virtual Speedup

“Speed up” by slowing everything else down.

Each time runs, pause all other threads.
Virtual Speedup

“Speed up” by slowing everything else down.

Each time runs, pause all other threads.
Virtual Speedup

“Speed up” by slowing everything else down.

Each time runs, pause all other threads.
Speedup Results

Program Speedup

Speedup
Speedup Results

Program Speedup

Speedup
Virtual Speedup

“Speed up” by slowing everything else down.

Each time runs, pause all other threads.
Virtual Speedup

“Speed up” by slowing everything else down.

Each time runs, pause all other threads.
Virtual Speedup

“Speed up” by slowing everything else down.

Each time runs, pause all other threads.
Speedup Results

Program Speedup

[Graph showing two lines representing speedup results]
Speedup Results

Program Speedup

Speedup
Virtual Speedup

“Speed up” by slowing everything else down.

Each time runs, pause all other threads.
Virtual Speedup

“Speed up” by slowing everything else down.

Each time runs, pause all other threads.
Virtual Speedup

“Speed up” by slowing everything else down.

Each time runs, pause all other threads.
Virtual Speedup

“Speed up” by slowing everything else down.

Each time runs, pause all other threads.
Speedup Results

Program Speedup

Speedup
Speedup Results

Speeding up \( \rightarrow \) slows the program down!
Is runtime meaningful?

Take a picture

Send it to Ogle

Find similar pictures

Send results

Add it to the database

found 8,000,000 similar images
Is runtime meaningful?
Is runtime meaningful?

How fast do results come back?
Is runtime meaningful?

How long between request and response?

How fast do results come back?
Progress Points
Bob wants to send responses faster.
Bob wants to send responses faster.

He marks ⬤ as a progress point.
Bob wants to send responses faster.

He marks ✈️ as a *progress point*.

Each time this code runs...
Bob wants to send responses faster.
He marks as a progress point.
Each time this code runs...
Progress Points

Many requests running for many users.
Progress Points

Many requests running for many users.
Progress Points

One progress point measures throughput.
Progress Points

One progress point measures throughput.

If I speed up [?], how much faster do I run [?]

[Diagram with various shapes and arrows]
Progress Points
Bob wants to minimize response time.
Bob wants to minimize response time. He adds *latency progress points*.
Bob wants to minimize response time. He adds *latency progress points*.
Bob wants to minimize response time. He adds *latency progress points*.
Bob wants to minimize response time.
Bob wants to minimize response time.

Little’s Law: $W = \frac{L}{\lambda}$
Bob wants to minimize response time. Little’s Law: $W = \frac{L}{\lambda}$
Bob wants to minimize response time.

Little’s Law: \( W = \frac{L}{\lambda} \)

Latency = Transactions
Bob wants to minimize response time.

Little's Law: \( W = \frac{L}{\lambda} \)

\[ \text{latency} = \frac{\text{transactions}}{\text{throughput}} \]
Coz: a Causal Profiler for Linux

(coming to Debian, in Unstable now)
Coz: a Causal Profiler for Linux

> coz run --- ./some_program args

(coming to Debian, in Unstable now)
Coz Produces Causal Profiles

Program Speedup

? Speedup
Coz Produces Causal Profiles

Let’s use it to improve Ogle
Using Causal Profiling on Ogle

found 8,000,000 similar images
Using Causal Profiling on Ogle
Using Causal Profiling on Ogle
Using Causal Profiling on Ogle

dedup
compression
ferret
image comparison
Using Causal Profiling on Ogle

SQLlite

dedup
compression

ferret
image comparison
Using Causal Profiling on Ogle
Ferret
image comparison
Ferret
image comparison

segmentation → feature extraction → indexing → ranking
Ferret
image comparison

input

segmentation

feature extraction

indexing

output

ranking
Ferret
image comparison

input → segmentation → feature extraction → indexing → ranking → output
Ferret

Line Speedup vs Program Speedup for different lines:
- Line 320
- Line 358
- Line 255

Line Speedup:
- 0%
- 25%
- 50%
- 75%
- 100%
Ferret

input → segmentation → feature extraction → indexing → ranking → output
Ferret

- **Input**
- **Segmentation**
- **Feature Extraction**
- **Indexing**
- **Ranking**
- **Output**
Ferret

- Input
- Segmentation
- Feature Extraction
- Indexing
- Ranking

Probably doesn’t need as many threads
Ferret

input

segmentation

feature extraction

indexing

ranking

output
Ferret

input

segmentation

feature extraction

indexing

ranking

21% Speedup
What did Causal Profiling predict?
What did Causal Profiling predict?

Increased from 16 to 22 threads
What did Causal Profiling predict?

Increased from 16 to 22 threads

27% increase in ranking throughput
What did Causal Profiling predict?

27% increase in ranking throughput
What did Causal Profiling predict?

27% increase in ranking throughput
What did Causal Profiling predict?

27% increase in ranking throughput
What did Causal Profiling predict?

27% increase in ranking throughput

Causal Profiling predicted a 21% improvement
What did Causal Profiling predict?

27% increase in ranking throughput

Causal Profiling predicted a 21% improvement

Exactly what we observed
Using Causal Profiling on Ogle

dedup compression
ferret image comparison
Using Causal Profiling on Ogle

- SQLite
- dedup
- compression
- ferret
- image comparison
Dedup
Compression via deduplication
Dedup
Compression via deduplication
Dedup
Compression via deduplication

I HAD FUN ONCE
IT WAS AWFUL
Dedup
Compression via deduplication
Dedup
Compression via deduplication
Dedup
Compression via deduplication
Dedup
Compression via deduplication

grumpycat1.jpg
Dedup
Compression via deduplication

grumpycat1.jpg

funisawful.jpg
Dedup
Compression via deduplication
Dedup
Compression via deduplication

hash_function()
Dedup
Compression via deduplication

\[ i = \text{hash\_function}(\cdot) \]
Dedup
Compression via deduplication

\[ i = \text{hash\_function}(\quad) \]
Dedup
Compression via deduplication

hash_function(I HAD FUN ONCE)
Dedup
Compression via deduplication

\[ i = \text{hash\_function}(\text{I had fun once}) \]
Dedup
Compression via deduplication

\[ i = \text{hash\_function}( ) \]
Dedup
Compression via deduplication

hash_function()
Dedup

Compression via deduplication

\[ i = \text{hash\_function}(\quad) \]
Dedup
Compression via deduplication

\[ i = \text{hash} \_\text{function}(\quad) \]
Dedup
Compression via deduplication

Hash table is accessed concurrently by many threads
Dedup
Compression via deduplication

Hash table is accessed concurrently by many threads
Dedup
Compression via deduplication

Hash table is accessed concurrently by many threads

Causal Profiler says the loop that accesses this list is important
Dedup
Compression via deduplication

More hash buckets should lead to fewer collisions
Dedup
Compression via deduplication

More hash buckets should lead to fewer collisions
Dedup
Compression via deduplication

More hash buckets should lead to fewer collisions

No performance improvement
Dedup
Compression via deduplication

What else could be causing collisions?
Dedup
Compression via deduplication

What else could be causing collisions?

\[ i = \text{hash}\_\text{function}(\quad) \]
Dedup
Compression via deduplication

Horrible hash function!
Dedup
Compression via deduplication

Horrible hash function!

Bin Utilization
2.3%
Dedup

Compression via deduplication

Bin Utilization 2.3%
Dedup
Compression via deduplication

Bin Utilization

<table>
<thead>
<tr>
<th>Original Version</th>
<th>After Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3%</td>
<td>82%</td>
</tr>
</tbody>
</table>
Dedup
Compression via deduplication

Bin Utilization

2.3%

9% Speedup

82%
What did Causal Profiling predict?
What did Causal Profiling predict?

Blocks per-bucket
What did Causal Profiling predict?

Blocks per-bucket

Before: 76.7

Dedup
Compression via deduplication
Dedup
Compression via deduplication

What did Causal Profiling predict?

Blocks per-bucket

Before: 76.7
After: 2.09
What did Causal Profiling predict?

**Blocks per-bucket**

Before: 76.7

After: 2.09

96% traversal speedup

Dedup
Compression via deduplication
Dedup
Compression via deduplication

What did Causal Profiling predict?

Blocks per-bucket

Before: 76.7
After: 2.09

96% traversal speedup

9% predicted speedup, exactly what we observed
Using Causal Profiling on Ogle
Using Causal Profiling on Ogle

dedup
compression
ferret
image comparison

SQLite
#if THREAD_SAFE
#if THREAD_SAFE
config_t global_config = { 

#if THREADSAFE
config_t global_config = {
    ...
}
#if THREAD_SAFE
config_t global_config = {
  ...
  .unlock = pthread_mutex_unlock,
}
#if THREAD_SAFE
config_t global_config = {
    ...
    .unlock = pthread_mutex_unlock,
    .getsize = sqlite_usable_size,
#if THREAD_SAFE
config_t global_config = {

...;

.unlock = pthread_mutex_unlock,
.getsize = sqlite_usable_size,
.nextitem = sqlite_pagecache_next,
#if THREAD_SAFE
config_t global_config = {
    ...
    .unlock = pthread_mutex_unlock,
    .getsize = sqlite_usable_size,
    .nextitem = sqlite_pagecache_next,
    ...
};
#endif
void sqlite_unlock(lock* l) {
    global_config.unlock(l);
}

#if THREAD_SAFE
config_t global_config = {
    ...
    .unlock = pthread_mutex_unlock,
    .getsize = sqlite_usable_size,
    .nextitem = sqlite_pagecache_next,
    ...
};
#endif
void sqlite_unlock(lock* l) {
    global_config.unlock(l);
}

Indirect Call
void sqlite_unlock(lock* l) {
    global_config.unlock(l);
}

**Indirect Call**
Cheap, but almost the same cost as pthread_mutex_unlock
void sqlite_unlock(lock* l) {
    global_config.unlock(l);
}

void sqlite_unlock(lock* l) {
    global_config.unlock(l);
}

void sqlite_getsize(void* p) {
    global_config.getsize(p);
}
void sqlite_unlock(lock* l) {
    global_config.unlock(l);
}

void sqlite_getsize(void* p) {
    global_config.getsize(p);
}

void sqlite_nextitem(item* i) {
    global_config.nextitem(i);
}
void sqlite_unlock(lock* l) {
    global_config.unlock(l);
}

void sqlite_getsize(void* p) {
    global_config.getsize(p);
}

void sqlite_nextitem(item* i) {
    global_config.nextitem(i);
}
Simple SQL Database

Line Speedup

Program Speedup

Line 16916

Line 18974

Line 40345
Using Causal Profiling on Ogle

SQLite

dedup compression

ferret
image comparison
Using Causal Profiling on Ogle

- dedup compression
- ferret image comparison

SQLite 25%
Using Causal Profiling on Ogle

- SQLite: 25%
- dedup compression: 9%
- ferret image comparison
Using Causal Profiling on Ogle

- SQLite: 25%
- dedup compression: 9%
- ferret image comparison: 21%
# Summary of Optimizations

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Speedup</th>
<th>Diff Size</th>
<th>Change Summary</th>
</tr>
</thead>
</table>


# Summary of Optimizations

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Speedup</th>
<th>Diff Size</th>
<th>Change Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>memcached</td>
<td>9.39%</td>
<td>-6, +2</td>
<td>removed unnecessary locks</td>
</tr>
</tbody>
</table>
## Summary of Optimizations

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Speedup</th>
<th>Diff Size</th>
<th>Change Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>memcached</td>
<td>9.39%</td>
<td>-6, +2</td>
<td>removed unnecessary locks</td>
</tr>
<tr>
<td>sqlite</td>
<td>25.60%</td>
<td>-3, +3</td>
<td>removed DIY vtable implementation</td>
</tr>
</tbody>
</table>
## Summary of Optimizations

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Speedup</th>
<th>Diff Size</th>
<th>Change Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>memcached</td>
<td>9.39%</td>
<td>-6, +2</td>
<td>removed unnecessary locks</td>
</tr>
<tr>
<td>sqlite</td>
<td>25.60%</td>
<td>-3, +3</td>
<td>removed DIY vtable implementation</td>
</tr>
<tr>
<td>blackscholes</td>
<td>2.56%</td>
<td>-61, +4</td>
<td>manual common subexpression elimination</td>
</tr>
</tbody>
</table>
## Summary of Optimizations

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Speedup</th>
<th>Diff Size</th>
<th>Change Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>memcached</td>
<td>9.39%</td>
<td>-6, +2</td>
<td>removed unnecessary locks</td>
</tr>
<tr>
<td>sqlite</td>
<td>25.60%</td>
<td>-3, +3</td>
<td>removed DIY vtable implementation</td>
</tr>
<tr>
<td>blackscholes</td>
<td>2.56%</td>
<td>-61, +4</td>
<td>manual common subexpression elimination</td>
</tr>
<tr>
<td>dedup</td>
<td>8.95%</td>
<td>-3, +3</td>
<td>fixed degenerate hash function</td>
</tr>
</tbody>
</table>
# Summary of Optimizations

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Speedup</th>
<th>Diff Size</th>
<th>Change Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>memcached</td>
<td>9.39%</td>
<td>-6, +2</td>
<td>removed unnecessary locks</td>
</tr>
<tr>
<td>sqlite</td>
<td>25.60%</td>
<td>-3, +3</td>
<td>removed DIY vtable implementation</td>
</tr>
<tr>
<td>blackscholes</td>
<td>2.56%</td>
<td>-61, +4</td>
<td>manual common subexpression elimination</td>
</tr>
<tr>
<td>dedup</td>
<td>8.95%</td>
<td>-3, +3</td>
<td>fixed degenerate hash function</td>
</tr>
<tr>
<td>ferret</td>
<td>21.27%</td>
<td>-4, +4</td>
<td>rebalanced pipeline thread allocation</td>
</tr>
</tbody>
</table>
## Summary of Optimizations

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Speedup</th>
<th>Diff Size</th>
<th>Change Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>memcached</td>
<td>9.39%</td>
<td>-6, +2</td>
<td>removed unnecessary locks</td>
</tr>
<tr>
<td>sqlite</td>
<td>25.60%</td>
<td>-3, +3</td>
<td>removed DIY vtable implementation</td>
</tr>
<tr>
<td>blackscholes</td>
<td>2.56%</td>
<td>-61, +4</td>
<td>manual common subexpression elimination</td>
</tr>
<tr>
<td>dedup</td>
<td>8.95%</td>
<td>-3, +3</td>
<td>fixed degenerate hash function</td>
</tr>
<tr>
<td>ferret</td>
<td>21.27%</td>
<td>-4, +4</td>
<td>rebalanced pipeline thread allocation</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>37.50%</td>
<td>-1, +0</td>
<td>removed custom barrier with high contention</td>
</tr>
</tbody>
</table>
## Summary of Optimizations

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Speedup</th>
<th>Diff Size</th>
<th>Change Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>memcached</td>
<td>9.39%</td>
<td>-6, +2</td>
<td>removed unnecessary locks</td>
</tr>
<tr>
<td>sqlite</td>
<td>25.60%</td>
<td>-3, +3</td>
<td>removed DIY vtable implementation</td>
</tr>
<tr>
<td>blackscholes</td>
<td>2.56%</td>
<td>-61, +4</td>
<td>manual common subexpression elimination</td>
</tr>
<tr>
<td>dedup</td>
<td>8.95%</td>
<td>-3, +3</td>
<td>fixed degenerate hash function</td>
</tr>
<tr>
<td>ferret</td>
<td>21.27%</td>
<td>-4, +4</td>
<td>rebalanced pipeline thread allocation</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>37.50%</td>
<td>-1, +0</td>
<td>removed custom barrier with high contention</td>
</tr>
<tr>
<td>streamcluster</td>
<td>68.40%</td>
<td>-1, +0</td>
<td>removed custom barrier with high contention</td>
</tr>
</tbody>
</table>
## Summary of Optimizations

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Speedup</th>
<th>Diff Size</th>
<th>Change Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>memcached</td>
<td>9.39%</td>
<td>-6, +2</td>
<td>removed unnecessary locks</td>
</tr>
<tr>
<td>sqlite</td>
<td>25.60%</td>
<td>-3, +3</td>
<td>removed DIY vtable implementation</td>
</tr>
<tr>
<td>blackscholes</td>
<td>2.56%</td>
<td>-61, +4</td>
<td>manual common subexpression elimination</td>
</tr>
<tr>
<td>dedup</td>
<td>8.95%</td>
<td>-3, +3</td>
<td>fixed degenerate hash function</td>
</tr>
<tr>
<td>ferret</td>
<td>21.27%</td>
<td>-4, +4</td>
<td>rebalanced pipeline thread allocation</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>37.50%</td>
<td>-1, +0</td>
<td>removed custom barrier with high contention</td>
</tr>
<tr>
<td>streamcluster</td>
<td>68.40%</td>
<td>-1, +0</td>
<td>removed custom barrier with high contention</td>
</tr>
<tr>
<td>swapoptions</td>
<td>15.80%</td>
<td>-10, +16</td>
<td>reordered loop nests</td>
</tr>
</tbody>
</table>
Backup Slides
Testing Virtual Speedups
Testing Virtual Speedups

Two versions of A
Testing Virtual Speedups

Two versions of A

One with a delay added
Testing Virtual Speedups

Two versions of A

One with a delay added, and one unmodified.
Testing Virtual Speedups

Two versions of A

One with a delay added, and one unmodified.

Two versions of our program
Two versions of our program
Two versions of our program

- One with a faster A
Two versions of our program

C \rightarrow A \rightarrow A

C \rightarrow B

“fast”

one with a faster A
Two versions of our program

“fast”

one with a faster A

one with a slower A
Two versions of our program

“fast”

“slow”

one with a faster

one with a slower
Two versions of our program

“fast”

“slow”
Two versions of our program

“fast”

“slow”

Speeding up by
Two versions of our program

"fast"

"slow"

Speeding up by shortens execution by this much
Two versions of our program

"fast"

Speeding up A by
shortens execution by this much

Use virtual speedup to emulate the same speedup on this version.

"slow"
Two versions of our program

Speeding up $A$ by $\frac{1}{2}$ shortens execution by this much

"fast"

"slow"
Two versions of our program

```
C -> A -> A
```

```
C -> B
```

“fast”

```
C -> A -> A
```

```
C -> B
```

“slow”

Speeding up A by this much shortens execution by this much.
Two versions of our program

"fast"

"slow"

Speeding up by shortens execution by this much

"fast"
Two versions of our program

Speeding up by this much

“fast”

“slow”
Two versions of our program

“fast”

“slow”

Speeding up by

shortens execution by this much
Two versions of our program

"fast"

Speeding up \[A\] by \[\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\] virtually speeding up \[A\] should have the same effect

"slow"
Redis Results
Waiting for accept response
Cleaning up after a query

networking.c:875
Custom Number to String Conversion
Frequently Executed Hash Function

![Graph showing the execution of hash functions across different locations and their impact on speedup.](image)
Allocating Memory for Custom String Representation
Setting up Custom String Representation

![Graph showing custom string representation with object.c:70 on the y-axis and several data points plotted against the x-axis.](Image)
Changing Length of Custom String

sds.c:203
“Pinning” Custom Strings