PERFORMANCE (REALLY) MATTERS

Emery Berger
UMASS AMHERST

(joint work with Charlie Curtsinger, Grinnell College)

emeryberger.com, @emeryberger
2020

- 2.65 GHz x 6
- 64-bit processor
- 4Gb RAM
1980s

4.77 MHz
16-bit processor
640Kb RAM
4.77 MHz
16-bit processor
640Kb RAM

2.65 GHz x 6
64-bit processor
4Gb RAM

4.77 MHz
16-bit processor
640Kb RAM
2.65 GHz x 6
64-bit processor
4Gb RAM

≈ 3000x

4.77 MHz
16-bit processor
640Kb RAM
-1980s
"Metal" Languages

1949  1957  1972  1985
40 Year Performance Ride!
40 Year Performance Ride!

Transistors (millions)

Clock Speed (MHz)

Year

Year
40 Year Performance Ride!

Transistors (millions)

Clock Speed (MHz)

Log scale

Year

Year
40 Year Performance Ride!

Transistors (millions)

Clock Speed (MHz)

Smaller = Faster
40 Year Performance Ride!

Smaller = Faster
Just buy new hardware and your code runs faster
"Metal" Languages
1980-1990s
"Irrational Exuberance" Languages

"Irrational Exuberance" Languages
HOW MANY BYTES IN AN...INT?
HOW MANY BYTES IN AN...INT?
HOW MANY BYTES IN AN...INT?

sizeof(1)
HOW MANY BYTES IN AN...INT?

`sizeof(1)` → 4
HOW MANY BYTES IN AN...INT?

C++

`sizeof(1)`

→ 4

Python
HOW MANY BYTES IN AN… INT?

C++: `sizeof(1)` → 4

Python: `>>> sys.getsizeof(1) → 4`
HOW MANY BYTES IN AN...INT?

C++

```c
sizeof(1) → 4
```

Python

```python
>>> sys.getsizeof(1)
24
```
HOW MANY BYTES IN AN… EMPTY LIST?
HOW MANY BYTES IN AN...EMPTY LIST?

`sizeof(list<int>)`
HOW MANY BYTES IN AN...EMPTY LIST?

```
#include <list>

std::size_t size = sizeof(list<int>);
```

→ 24
HOW MANY BYTES IN AN EMPTY LIST?

sizeof(list<int>) → 24
HOW MANY BYTES IN AN…EMPTY LIST?

C++

`sizeof(list<int>)` → 24

Python

`>>> sys.getsizeof([])`
HOW MANY BYTES IN AN...EMPTY LIST?

C++
```
sizeof(list<int>)
```
→ 24

Python
```
>>> sys.getsizeof([])
```
56
HOW MANY BYTES IN AN...EMPTY DICT?
HOW MANY BYTES IN AN... EMPTY DICT?

`sizeof(map<int, int>)`
HOW MANY BYTES IN AN… EMPTY DICT?

`sizeof(map<int, int>)` → 24
HOW MANY BYTES IN AN... EMPTY DICT?

```c++
sizeof(map<int, int>) → 24
```
HOW MANY BYTES IN AN…EMPTY DICT?

_in C++_
```
sizeof(map<int, int>)
```
→ 24

_in Python_
```python
>>> sys.getsizeof({})
```

How many bytes in an... empty dict?

C++

```
sizetof(map<int, int>) → 24
```
HOW MANY BYTES IN AN... EMPTY DICT?

C++: `sizeof(map<int, int>)` → 24

Python: `>> sys.getsizeof({})` → 64

Python: `240` (3.6)
HOW MANY BYTES IN AN ... EMPTY DICT?

```python
>>> sys.getsizeof({})
240
```
Something was said...not good.
What was it?

Slow!

They called you slow!
for i in range(n):
    for j in range(n):
        for k in range(n):
            C[i][j] += A[i][k] * B[k][j]
Matrix-Multiply Speedup vs. Pure Python

Python: 1
C: 47
parallel loops: 366
memory optimization: 6727
SIMD instructions: 62806
Matrix-Multiply Speedup vs. Pure Python

Python

C

1

47

100

1000

10000

100000

6727

62806
THE SIMPSONS
TAPPED OUT
≈2010: THE RIDE IS OVER
The Ride Is Over

Transistors (millions)

Clock Speed (MHz)
The Ride Is Over

Transistors (millions)

Clock Speed (MHz)
The Ride Is Over

Transistor counts still increased

Clock Speed (MHz)
The Ride Is Over

Transistors (millions)

Clock Speed (MHz)

Transistor counts still increased (Moore's Law)
The Ride Is Over

Transistors (millions)

Clock Speed (MHz)

Transistor counts still increased

(Moore's Law)
The Ride Is Over

Transistor counts still increased

(Moore's Law)

(Dennard Scaling)
Performance in the ‘80s

just chill and wait for faster hardware!
Modern-day performance

why is this so hard??
Modern-day performance

why is this so hard??
Typical 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);
}

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

Typical performance evaluation
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 setITimer(int msec)
{
    struct ITimer timer;
    timer.it_value.tv_usec = (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);
    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");
}

Typical performance evaluation
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 timeval timer;
    timer.tv_sec = (msec / 1000) / 1000;
    timer.tv_usec = (msec % 1000) / 1000;
    settimeofday(&timer, 0);
    setTimer(TIMER_REAL, &timer, 0);
}
Which is faster?

A  A′
Which is faster?

A
90s

A′
Which is faster?

A
90s

A'
87.5s
Which is faster?

A
90s

A'
87.5s

A'
2.8% faster!
Why is $A'$ faster than $A$?
Why is $A'$ faster than $A$?
Why is $A'$ faster than $A$?

The code change!
Why is $A'$ faster than $A$?

The code change!
Why is \( A' \) faster than \( A \)?

Or was it an accident?
Why is $A'$ faster than $A$?

Or was it an accident?
Why is $A'$ faster than $A$?

Layout biases measurement

Mytkowicz et al. (ASPLOS’09)
Mytkowicz et al. (ASPLOS’09)
Producing Wrong Data Without Doing Anything Obviously Wrong!

Todd Mytkowicz  Amer Diwan
Department of Computer Science
University of Colorado
Boulder, CO, USA
{mytkowit,diwan}@colorado.edu

Matthias Hauswirth
Faculty of Informatics
University of Lugano
Lugano, CH
Matthias.Hauswirth@unisi.ch

Peter F. Sweeney
IBM Research
Hawthorne, NY, USA
pfs@us.ibm.com
Layout biases measurement

Mytkowicz et al. (ASPLOS’09)
Producing Wrong Data Without Doing Anything Obviously Wrong!

Todd Mytkowicz  Amer Diwan
Department of Computer Science
University of Colorado
Boulder, CO, USA
{mytkowit,diwan}@colorado.edu

Matthias Hauswirth
Faculty of Informatics
University of Lugano
Lugano, CH
Matthias.Hauswirth@unisi.ch

Peter F. Sweeney
IBM Research
Hawthorne, NY, USA
pfs@us.ibm.com

Link Order
Changes function addresses
Layout biases measurement

Mytkowicz et al. (ASPLOS’09)
Producing Wrong Data Without Doing Anything Obviously Wrong!

Todd Mytkowicz  Amer Diwan  Matthias Hauswirth  Peter F. Sweeney
Department of Computer Science  Faculty of Informatics  IBM Research
University of Colorado  University of Lugano  Hawthorne, NY, USA
Boulder, CO, USA  Lugano, CH  pfs@us.ibm.com
{mytkowit,diwan}@colorado.edu  Matthias.Hauswirth@unisi.ch

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! (±40%)

Environment
Variable Size
Moves the program stack
Why is $A'$ faster than $A$?
Why is $A'$ faster than $A$?

map to same cache set
Why is \textbf{A}' faster than \textbf{A}?
Why is $A'$ faster than $A$?
Why is $A'$ faster than $A$?
Why is \( A' \) faster than \( A \) ?

### Code Comparison

#### Version A

```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();
        f->setTimer();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

#### Version A'

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

### Diagram

A map to same set

```
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);
}
```
Why is \( A' \) faster than \( A \)?

map to

same set

Nothing here
Why is \( A' \) faster than \( A \)?

Map to same set

Nothing here

No conflict
It's the cache
It's the cache or branch predictor.
It's the cache or branch predictor.
It's the cache or branch predictor or branch target predictor.
It’s the cache
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);
}

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;
}

or TLB

or branch predictor
A′
or branch
target
predictor
or
prefetcher
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;
}

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;
}
}

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);
}


Speedup of -O3 over -O2

- Significant
- No
help?
found 8,000,000 similar images
It’s going to totally disrupt image search.
The Prototype
The Prototype Ogle

Take a picture
The Prototype

Take a picture

Send it to Ogle
The Prototype

Take a picture

Send it to Ogle

Add it to the database
The Prototype Ogle

Take a picture

Send it to Ogle

Add it to the database

Find similar pictures
The Prototype Ogle

Take a picture

Send it to Ogle

Add it to the database

Send results

Find similar pictures

found 8,000,000 similar images
Send results

Find similar pictures

found 8,000,000 similar images

Send results
found 8,000,000 similar images

Send results

Find similar pictures
found 8,000,000 similar images

Send results

Find similar pictures
found 8,000,000 similar images

Send results

Find similar pictures
found 8,000,000 similar images

Send results

Find similar pictures
found 8,000,000 similar images

Send results

Find similar pictures
Ogle is too slow!
Software Profilers
Software Profilers
<table>
<thead>
<tr>
<th>%</th>
<th>cumulative</th>
</tr>
</thead>
<tbody>
<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>% time</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

<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>
Software Profilers

<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>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>19.95</td>
<td>11.31</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>
<tr>
<td>%</td>
<td>cumulative time</td>
<td>seconds</td>
<td>calls</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------</td>
<td>---------</td>
<td>-------</td>
</tr>
<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>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

Are these places where Homer should focus on performance?

Frequently executed code

Code that runs for a long time

<table>
<thead>
<tr>
<th>%</th>
<th>time</th>
<th>cumulative 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>
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?

DOH!

loading...
Would this speed up Ogle?

Frequently executed code

loading...
Would this speed up Ogle?

Frequently executed code

Code that runs for a long time

DOH!
Would this speed up Ogle?

Frequently executed code

Code that runs for a long time

Profilers do a bad job finding important code in modern applications!

parallel / asynchronous / concurrent

loading...
Would this speed up Ogle?

Frequently executed code

Code that runs for a long time

Profilers do a bad job finding important code in modern applications! 
*parallel / asynchronous / concurrent*

We need to do better!
What would speed up Ogle?
What *would* speed up Ogle?
What would speed up Ogle?

Hey, look over here

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
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
If you speed up this much
Causal Profile

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
How do we know this change?
Causal Profile

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

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...
Performance Experiments

If we could magically speed up ...

Speeding up \( \cdot \) 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 ...

More speedup in ...
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...
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.
“Speed up” by slowing everything else down.

Speeding up by this much...
“Speed up” by slowing everything else down.

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

Program Speedup

[Blank Graph]
Speedup Results

Program Speedup

? Speedup
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.
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.
Program Speedup

Speedup Results

Program Speedup

? Speedup

? Speedup
Speedup Results

Program Speedup

Speedup
Take a picture

Send it to Ogle

Find similar pictures

Add it to the database

Send results

found 8,000,000 similar images
found 8,000,000 similar images
How long between request and response?
How long between request and response?
(latency)
How long between request and response? (latency)

How fast do results come back?
How long between request and response?

(latency)

How fast do results come back?

(thruput)

found 8,000,000 similar images
Progress Points
Homer wants to send responses faster.
Homer wants to send responses faster. He marks as a progress point.
Homer wants to send responses faster. He marks as a progress point. Each time this code runs...
Homer wants to send responses faster.

He marks 🚀 as a progress point.

Each time this code runs...
Progress Points
Many requests running for many users.
Many requests running for many users.
Progress Points
Progress Points

One progress point measures throughput.
Progress Points

One progress point measures throughput.

If I speed up ???, how much faster do I run ???
Progress Points
Homer wants to minimize response time.
Homer wants to minimize response time.

He adds *latency progress points*.
Homer wants to minimize response time.

He adds latency progress points.
Homer wants to minimize response time. He adds *latency progress points*.
Homer wants to minimize response time.
Homer wants to minimize response time.

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

Little’s Law: $W = \frac{L}{\lambda}$

Homer wants to minimize response time.

transactions++

transactions--
Homer wants to minimize response time.

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

\( \text{latency} = \text{transactions} \)
Homer 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

*(ships with Debian / Ubuntu)*
Coz: a Causal Profiler for Linux

*(ships with Debian / Ubuntu)*

> sudo apt install coz-profiler
Coz: a Causal Profiler for Linux

*(ships with Debian / Ubuntu)*

> sudo apt install coz-profiler
> coz run --- ./some_program args
Coz: a Causal Profiler for Linux

*(ships with Debian / Ubuntu)*

> sudo apt install coz-profiler
> coz run --- ./some_program args

Random *performance experiments*
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
Using Causal Profiling on Ogle
Using Causal Profiling on Ogle

dedup compression
ferret
image comparison

SQLlite

PARSEC
['pär-"sek] A unit of measure
Using Causal Profiling on Ogle

dedup compression ferret image comparison

PARSEC
['pär-"sek] A unit of measure
Ferret
Ferret

Line Speedup

Program Speedup

Line 320

Line 358

Line 255

0% 25% 50% 75% 100%

Line

- - line 320
- - line 358
- - line 255

ranking

indexing

segmentation
Ferret

input

segmentation

feature extraction

indexing

ranking

output
Ferret

input

segmentation

feature extraction

indexing

output

ranking
Ferret

input

segmentation

feature extraction

indexing

output

Probably doesn’t need as many threads
Ferret

input

segmentation

feature extraction

indexing

ranking

output
Ferret

input

segmentation

feature extraction

indexing

ranking

output

21% Speedup
What did Coz predict?
What did Coz predict?

Increased from 16 to 22 threads
What did Coz predict?

Increased from 16 to 22 threads

27% increase in ranking throughput
What did Coz predict?

27% increase in ranking throughput
What did Coz predict?

27% increase in ranking throughput
What did Coz predict?

27% increase in ranking throughput
What did Coz predict?

27% increase in ranking throughput

Coz predicted a 21% improvement
What did Coz predict?

27% increase in ranking throughput

Coz predicted a 21% improvement

Exactly what we observed
Using Causal Profiling on Ogle

SQLite

dedup compression ferret image comparison

PARSEC ['pär-"sek] A unit of measure
Using Causal Profiling on Ogle

PARSEC
['pär-"sek'] A unit of measure

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

deduplication process diagram with images and text
Dedup
Compression via deduplication
Dedup
Compression via deduplication

hash_function( )
Dedup
Compression via deduplication

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

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

hash_function( "I HAD FUN ONCE" )
Dedup
Compression via deduplication

\[ i = \text{hash}\_\text{function}(\text{I HAD FUN ONCE}) \]
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

\[ i = \text{hash	extunderscore function}(\quad) \]
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

Coz says the loop that accesses this list is important
Dedup
Compression via deduplication
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?

\[ i = \text{hash\_function}(\_\_\_\_\_) \]
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

2.3%
Dedup
Compression via deduplication

Bin Utilization
2.3%

9% Speedup

82%
Dedup
Compression via deduplication

What did Coz predict?
Dedup
Compression via deduplication

What did Coz predict?

Blocks per-bucket
Dedup
Compression via deduplication

What did Coz predict?

Blocks per-bucket

Before: 76.7
Dedup
Compression via deduplication

What did Coz predict?

Blocks per-bucket

Before: 76.7
After: 2.09
What did Coz predict?

Blocks per-bucket

- Before: 76.7
- After: 2.09

96% traversal speedup

Dedup
Compression via deduplication
Dedup
Compression via deduplication

What did Coz 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

SQLite

dedup compression

ferret image comparison
Using Causal Profiling on Ogle

SQLite 25%

dedup compression

ferret image comparison
Using Causal Profiling on Ogle

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

- **dedup compression**: 9%
- **ferret image comparison**: 21%
- **SQLite**: 25%
## 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>swap0tions</td>
<td>15.80%</td>
<td>-10, +16</td>
<td>reordered loop nests</td>
</tr>
</tbody>
</table>
Effective Performance Profiling

% sudo apt install coz-profiler
the "C"

python
SCALENE
SCRIPTING-LANGUAGE AWARE PROFILING

github.com/emeryberger/scalene
% scalene yourprogram.py
% scalene --cpu-only yourprogram.py
% scalene --cpu-only --html yourprogram.py
% scalene --help
#!/usr/bin/env python3
import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.array(np.random.uniform(0, 100, size=(10**8)))
main()
<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>Mem % Python</th>
<th>Net (MB)</th>
<th>Memory usage over time / %</th>
<th>Copy (MB/s)</th>
<th>test2-2.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4%</td>
<td>1%</td>
<td>26%</td>
<td>97%</td>
<td>9</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>60%</td>
<td>34%</td>
<td>93%</td>
<td>84</td>
<td>763</td>
<td>87%</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```python
#!/usr/bin/env python3
import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.array(np.random.uniform(0, 100, size=(10**8)))
    main()
```
generated by the scalene profiler
<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>Mem % Python</th>
<th>Net (MB)</th>
<th>Memory usage over time / %</th>
<th>Copy (MB/s)</th>
<th>test2-2.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4%</td>
<td>1%</td>
<td>26%</td>
<td>97%</td>
<td>9</td>
<td>87%</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1%</td>
<td>60%</td>
<td>34%</td>
<td>93%</td>
<td>84</td>
<td></td>
<td>120</td>
<td></td>
</tr>
</tbody>
</table>

```python
#!/usr/bin/env python3
import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.array(np.random.uniform(0, 100, size=(10**8)))
    main()
```

generated by the `scalene` profiler

**CPU**

**PYTHON**

**vs. NATIVE**

**+ SYS%**
<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>Mem % Python</th>
<th>Net (MB)</th>
<th>Memory usage over time / %</th>
<th>Copy (MB/s)</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4%</td>
<td>1%</td>
<td>26%</td>
<td>97%</td>
<td>9</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1%</td>
<td>34%</td>
<td></td>
<td>93%</td>
<td>84</td>
<td></td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory usage: [ ] (max: 1618.94MB)  
est2-2.py: % of time = 100.00% out of 6.35s.

```python
#!/usr/bin/env python3
import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.array(np.random.uniform(0, 100, size=(10**8)))
    main()
```

generated by the [scalene](https://scalene.run) profiler
### Memory Usage Over Time

Memory usage: 1618.94MB (max: 1618.94MB)

test2-2.py: % of time = 100.00% out of 6.35s.

<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>Mem % Python</th>
<th>Net (MB)</th>
<th>Memory usage over time / %</th>
<th>Copy (MB/s)</th>
<th>test2-2.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4%</td>
<td>1%</td>
<td>26%</td>
<td>97%</td>
<td>9</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>60%</td>
<td></td>
<td>93%</td>
<td>84</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>34%</td>
<td></td>
<td></td>
<td>763</td>
<td>87%</td>
<td></td>
<td>120</td>
<td></td>
</tr>
</tbody>
</table>

```python
#!/usr/bin/env python3
import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.array(np.random.uniform(0, 100, size=(10**8)))
    main()
```

generated by the [scalene](https://scalene.com) profiler
## MEMORY USAGE OVER TIME

Memory usage: [graph] (max: 1618.94MB)

```
test2-2.py: % of time = 100.00% out of 6.35s.
```

<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>Mem % Python</th>
<th>Net (MB)</th>
<th>Memory usage over time / %</th>
<th>Copy (MB/s)</th>
<th>test2-2.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4%</td>
<td>1%</td>
<td>26%</td>
<td>97%</td>
<td>9</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>60%</td>
<td>93%</td>
<td>84</td>
<td>87%</td>
<td>763</td>
<td></td>
<td>120</td>
<td></td>
</tr>
</tbody>
</table>

```
#!/usr/bin/env python3
import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.array(np.random.uniform(0, 100, size=(10**8)))

    main()
```

generated by the scalene profiler

### CPU
**Python vs. Native**  
+ **Sys%**

### MEMORY
**Python vs. Native**  
**MEMORY USAGE OVER TIME**  
% OF MEM ALLOCATED

### COPY
**VOLUME**  
(MB/s)

---

**SCALENE**

---
<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>Mem % Python</th>
<th>Net (MB)</th>
<th>Memory usage over time / %</th>
<th>Copy (MB/s)</th>
<th>Python Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4%</td>
<td>1%</td>
<td>26%</td>
<td>97%</td>
<td>9</td>
<td>__________</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1%</td>
<td>34%</td>
<td></td>
<td>763</td>
<td>87%</td>
<td>__________</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```python
#!/usr/bin/env python3
import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.array(np.random.uniform(0, 100, size=(10**8)))
    main()
```

generated by the [scalene](https://scalene.com) profiler
34% of runtime in native code

```
#!/usr/bin/env python3
import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.array(np.random.uniform(0, 100, size=(10**8)))
    main()
```

generated by the `scalene` profiler
34% of runtime in native code

763MB allocated in native code

<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>Mem % Python</th>
<th>Net (MB)</th>
<th>Memory usage over time / %</th>
<th>Copy (MB/s)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4%</td>
<td>1%</td>
<td>26%</td>
<td>97%</td>
<td>9</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1%</td>
<td>34%</td>
<td></td>
<td></td>
<td>763</td>
<td>87%</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory usage: (max: 1618.94MB)

Generated by the `scalene` profiler

```python
#!/usr/bin/env python3
import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.array(np.random.uniform(0, 100, size=(10**8))

main()
```
### Memory Usage Analysis

<table>
<thead>
<tr>
<th>Line</th>
<th>CPU %</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>Mem % Python</th>
<th>Net (MB)</th>
<th>Memory usage over time / %</th>
<th>Copy (MB/s)</th>
<th>test2-2.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4%</td>
<td>1%</td>
<td>26%</td>
<td>97%</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>60%</td>
<td></td>
<td></td>
<td>93%</td>
<td>84</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1%</td>
<td>34%</td>
<td></td>
<td>87%</td>
<td>763</td>
<td></td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **34% of runtime in native code**
- **763MB allocated in native code**
- **87% of memory activity**

Generated by the **scalene** Profiler
34% of runtime in native code

763MB allocated in native code

87% of memory activity "sawtooth" pattern
34% of runtime in native code

763MB allocated in native code

87% of memory activity in a "sawtooth" pattern

120MB/s copying

```
#!/usr/bin/env python3
import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.array(np.random.uniform(0, 100, size=(10**8)))
    main()
```
34% of runtime in native code
763MB allocated in native code
87% of memory activity "sawtooth" pattern
120MB/s copying
converts to numpy array

<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>Mem % Python</th>
<th>Net (MB)</th>
<th>Memory usage over time / %</th>
<th>Copy (MB/s)</th>
<th>test2-2.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4%</td>
<td>1%</td>
<td>26%</td>
<td>97%</td>
<td>9</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>60%</td>
<td></td>
<td>93%</td>
<td>84</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1%</td>
<td>34%</td>
<td></td>
<td></td>
<td>763</td>
<td>87%</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

generated by the scalene profiler
34% of runtime in native code

763MB allocated in native code

87% of memory activity "sawtooth" pattern

120MB/s copying

converts to numpy array

already a numpy array!

generated by the `scalene` profiler
Memory usage: (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of 6.35s.

<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>Mem % Python</th>
<th>Net (MB)</th>
<th>Memory usage over time / %</th>
<th>Copy (MB/s)</th>
<th>test2-2.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4%</td>
<td>1%</td>
<td>26%</td>
<td>97%</td>
<td>9</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1%</td>
<td>60%</td>
<td>93%</td>
<td>84</td>
<td>763</td>
<td>87%</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1%</td>
<td>34%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#!/usr/bin/env python3

import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.array(np.random.uniform(0, 100, size=(10**8)))

    main()

generated by the scalene profiler
Memory usage: [red] (max: 1618.94MB)  
**test2-2.py**: % of time = 100.00% out of 6.35s.

<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>Mem % Python</th>
<th>Net (MB)</th>
<th>Memory usage over time / %</th>
<th>Copy (MB/s)</th>
<th>test2-2.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4%</td>
<td>1%</td>
<td>26%</td>
<td>97%</td>
<td>9</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>import numpy as np</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>def main():</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x = np.array(range(10**7))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>60%</td>
<td></td>
<td></td>
<td>93%</td>
<td>84</td>
<td>y = np.array(np.random.uniform(0, 100, size=(10**8)))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1%</td>
<td>34%</td>
<td></td>
<td>763</td>
<td>87%</td>
<td></td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>main()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

generated by the **scalene** profiler

Memory usage: [red] (max: 865.97MB)  
**test2-3.py**: % of time = 100.00% out of 5.40s.

<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>Mem % Python</th>
<th>Net (MB)</th>
<th>Memory usage over time / %</th>
<th>Copy (MB/s)</th>
<th>test2-3.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6%</td>
<td>1%</td>
<td>34%</td>
<td>93%</td>
<td>21</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>import numpy as np</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#@profile</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>def main():</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>67%</td>
<td></td>
<td></td>
<td>94%</td>
<td>82</td>
<td>x = np.array(range(10**7))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>25%</td>
<td>5%</td>
<td></td>
<td>763</td>
<td>85%</td>
<td>y = np.random.uniform(0, 100, size=(10**8))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>main()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

generated by the **scalene** profiler
### test2-2.py

<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>Mem % Python</th>
<th>Net (MB)</th>
<th>Memory usage over time / %</th>
<th>Copy (MB/s)</th>
<th>test2-2.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4%</td>
<td>1%</td>
<td>26%</td>
<td>97%</td>
<td>9</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1%</td>
<td>34%</td>
<td></td>
<td>93%</td>
<td>84</td>
<td>87%</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```python
#!/usr/bin/env python3
import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.random.uniform(0, 100, size=(10**8))

main()
```

generated by the **scalene** profiler

---

### test2-3.py

<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>Mem % Python</th>
<th>Net (MB)</th>
<th>Memory usage over time / %</th>
<th>Copy (MB/s)</th>
<th>test2-3.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6%</td>
<td>1%</td>
<td>34%</td>
<td>93%</td>
<td>21</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>67%</td>
<td>5%</td>
<td></td>
<td>94%</td>
<td>82</td>
<td>85%</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```python
#!/usr/bin/env python3
import numpy as np

# @profile
def main():
    x = np.array(range(10**7))
    y = np.random.uniform(0, 100, size=(10**8))

main()
```

generated by the **scalene** profiler
### test2-2.py

```
#!/usr/bin/env python3
import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.random.uniform(0, 100, size=(10**8))
    main()
```

generated by the `scalene` profiler

### test2-3.py

```
#!/usr/bin/env python3
import numpy as np

@profile
def main():
    x = np.array(range(10**7))
    y = np.random.uniform(0, 100, size=(10**8))
main()
```

generated by the `scalene` profiler
```python
#!/usr/bin/env python3
import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.random.uniform(0, 100, size=(10**8))
    main()
```

generated by the `scalene` profiler
<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>Mem % Python</th>
<th>Net (MB)</th>
<th>Memory usage over time / %</th>
<th>Copy (MB/s)</th>
<th>test2-2.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4%</td>
<td>1%</td>
<td>26%</td>
<td>97%</td>
<td>9</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td><strong>1%</strong></td>
<td><strong>34%</strong></td>
<td></td>
<td></td>
<td>763</td>
<td><strong>87%</strong></td>
<td>120</td>
<td>test2-2.py</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```python
#!/usr/bin/env python3
import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.array(np.random.uniform(0, 100, size=(10**8)))
main()
```

generated by the *scalene* profiler

<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>Mem % Python</th>
<th>Net (MB)</th>
<th>Memory usage over time / %</th>
<th>Copy (MB/s)</th>
<th>test2-3.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6%</td>
<td>1%</td>
<td>34%</td>
<td>93%</td>
<td>21</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td><strong>67%</strong></td>
<td></td>
<td></td>
<td></td>
<td>763</td>
<td><strong>85%</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td><strong>25%</strong></td>
<td><strong>5%</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```python
#!/usr/bin/env python3
import numpy as np

@profile
def main():
    x = np.array(range(10**7))
    y = np.random.uniform(0, 100, size=(10**8))
main()
```

generated by the *scalene* profiler
### test2-2.py

```python
#!/usr/bin/env python3

import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.array(np.random.uniform(0, 100, size=(10**8)))

main()
```

generated by the *[scalene]* profiler

---

### test2-3.py

```python
#!/usr/bin/env python3

import numpy as np

@profile
def main():
    x = np.array(range(10**7))
    y = np.random.uniform(0, 100, size=(10**8))

main()
```

generated by the *[scalene]* profiler
SCALENE
SCRIPTING-LANGUAGE AWARE PROFILING

precise

CPU
PYTHON
vs. NATIVE
+ SYS%

MEMORY
PYTHON
vs. NATIVE

MEMORY USAGE
OVER TIME

COPY VOLUME
(MB/s)

% OF MEM ALLOCATED

MEMORY

SCALENE

python™
SCALENE
SCRIPTING-LANGUAGE AWARE PROFILING

precise
CPU PYTHON vs. NATIVE
MEMORY PYTHON vs. NATIVE
MEMORY USAGE OVER TIME
COPY VOLUME (MB/s)
% OF MEM ALLOCATED

fast
MEMORY
CPU
SCALENE
python
Normalized profiler execution time (mdp)
Normalized profiler execution time (mdp)

memory_profiler: 152x
(12m38s vs. 5s)
Normalized profiler execution time (mdp)

```
memory_profiler: 152x
(12m38s vs. 5s)
```
DEFERRED SIGNAL DELIVERY
DEFERRED SIGNAL DELIVERY
DEFERRED SIGNAL DELIVERY

python™  python™  python™
DEFERRED SIGNAL DELIVERY
DEFERRED SIGNAL DELIVERY
DEFERRED SIGNAL DELIVERY

Python

C++

Python

Python

Python
DEFERRED SIGNAL DELIVERY

python™ python™ python™ python™

python™

C++ C++

python™
DEFERRED SIGNAL DELIVERY
DEFERRED SIGNAL DELIVERY

python™  python™ python™  python™

python™  C++  C++  python™

python™
INFERRING EXECUTION TIME

Python

C++

C++

Python

(wall clock)
INFERRING EXECUTION TIME

(wall clock)
INFERRING EXECUTION TIME

\[
\text{python_time} += \text{interval} \\
\text{c_time} += \text{delay} - \text{interval}
\]
DEFERRED SIGNAL DELIVERY

(wall clock) python_time += interval
c_time += delay - interval

(virtual time) sys_time = 1 - elapsed_virtual / elapsed_wallclock
DEFERRED SIGNAL DELIVERY

delay

(wall clock) python_time += interval
c_time += delay - interval

takesometime.py: % of time = 100.00% out of 1.03s.
### DEFERRED SIGNAL DELIVERY

\[
\text{(wall clock)} \quad \text{python\_time} + = \text{interval} \\
\text{c\_time} + = \text{delay} - \text{interval}
\]

\[\approx 30\% \quad \approx 70\%\]

---

```python
python_time += interval
c_time += delay - interval
```

**takesometime.py:** % of time = 100.00% out of 1.03s.

<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>takesometime.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>import time</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>import taketime</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>c_its_per_sec = 250000</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>python_its_per_sec = 1080000</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>import timeit</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>def py_taketime(n):</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td>\quad d = 1.01 |</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td>\quad for i in range(n):</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td>\quad \quad d = d * d + d * d</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>\quad return n</td>
</tr>
<tr>
<td>11</td>
<td>7%</td>
<td>1%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>18%</td>
<td>3%</td>
<td></td>
<td>c_secs = .7</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td>python_secs = .3</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td>start = time.process_time()</td>
</tr>
<tr>
<td>16</td>
<td>1%</td>
<td>70%</td>
<td></td>
<td>p = taketime.run(int(c_secs * c_its_per_sec))</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td>q = py_taketime(int(python_secs * python_its_per_sec))</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td>end = time.process_time()</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td>print(end-start)</td>
</tr>
</tbody>
</table>
```
DEFERRED SIGNAL DELIVERY

delay

(wall clock) python_time += interval
c_time += delay - interval

takesometime.py: % of time = 100.00% out of 1.03s.

<table>
<thead>
<tr>
<th>Line</th>
<th>CPU % Python</th>
<th>CPU % native</th>
<th>Sys %</th>
<th>takesometime.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>import time</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>import taketime</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>c_its_per_sec = 250000</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>python_its_per_sec = 1080000</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>import timeit</td>
</tr>
</tbody>
</table>
| 6    |              |              |       | def py_taketime(n):
| 7    |              |              |       |     d = 1.01 |
| 8    |              |              |       |     for i in range(n): |
| 9    |              |              |       |         for j in range(10): |
| 10   |              |              |       |         d = d * d + d * d |
| 11   | 7%           | 1%           | 3%    |     return n |
| 12   | 18%          |              |       | c_secs = .7   |
| 13   |              |              |       | python_secs = .3 |
| 14   | 1%           | 70%          |       | start = time.process_time() |
| 15   |              |              |       | p = taketime.run(int(c_secs * c_its_per_sec)) |
| 16   |              |              |       | q = py_taketime(int(python_secs * python_its_per_sec)) |
| 17   |              |              |       | end = time.process_time() |
| 18   |              |              |       | print(end-start) |

≈30%

≈70%
### Other "Irrational exuberance" Languages

<table>
<thead>
<tr>
<th>Scripting Language</th>
<th>malloc interposition</th>
<th>Monkey patching</th>
<th>Thread enum.</th>
<th>Stack inspection</th>
<th>Opcode disassembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perl</td>
<td>✓ (1)</td>
<td>✓</td>
<td>threads-&gt;list()</td>
<td>Devel::StackTrace</td>
<td>B::Concise</td>
</tr>
<tr>
<td>Tcl/Tk</td>
<td>✓ (2)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Python</td>
<td>✓ (3)</td>
<td>✓</td>
<td>threading.enumerate()</td>
<td>sys._current_frames()</td>
<td>dis</td>
</tr>
<tr>
<td>Lua</td>
<td>✓ (4)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHP</td>
<td>✓ (5)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>sys.call</td>
<td>disassemble</td>
</tr>
<tr>
<td>Ruby</td>
<td>✓ (6)</td>
<td>✓</td>
<td>Thread.list</td>
<td>caller</td>
<td>RubyVM::InstructionSequence</td>
</tr>
</tbody>
</table>
SCALENE
SCRIPTING-LANGUAGE AWARE PROFILING

github.com/emeryberger/SCALENE
SCALENE
SCRIPTING-LANGUAGE AWARE PROFILING

precise

CPU PYTHON vs. NATIVE
+ SYS%

MEMORY PYTHON vs. NATIVE

MEMORY USAGE OVER TIME

COPY VOLUME (MB/s)

% OF MEM ALLOCATED

github.com/emeryberger/SCALENE
SCALENE
SCRIPTING-LANGUAGE AWARE PROFILING

github.com/emeryberger/SCALENE

precise
- CPU PYTHON vs. NATIVE
- MEMORY PYTHON vs. NATIVE
- MEMORY USAGE OVER TIME
- COPY VOLUME (MB/s)
- % OF MEM ALLOCATED

fast

Normalized profiler execution time (mdp)
SCALENE
SCRIPTING-LANGUAGE AWARE PROFILING

github.com/emeryberger/SCALENE

precise fast

CPU PYTHON vs. NATIVE + SYS%
MEMORY PYTHON vs. NATIVE MEMORY USAGE OVER TIME COPY VOLUME (MB/s) % OF MEM ALLOCATED

% pip install -U scalene
SCALENE
SCRIPTING-LANGUAGE
AWARE
PROFILING

github.com/emeryberger/SCALENE

precise

CPU
PYTHON
vs. NATIVE
+ SYS%

MEMORY
PYTHON
vs. NATIVE

MEMORY
USAGE
OVER
TIME

COPY
VOLUME
(MB/s)

% OF MEM
ALLOCATED

fast

% pip install -U scalene
Effective Performance Profiling

COZ

% sudo apt install coz-profiler

github.com/plasma-umass/coz

github.com/emeryberger/SCALENE