Optimization Coaching

Vincent St-Amour
Sam Tobin-Hochstadt
Matthias Felleisen
PLT

Harvard - October 31st, 2012
It renders text, too!
2 hours later...

Neil
20 hours later...

(#s((phased-module-binding module-binding 0 zo 0))
  #<module-path-index>
  1
  parameter-name->arg-name
  #&
  parameter-name->arg-name)
(<lifted>)
  .
(#s((phased-module-binding module-binding 0 zo 0))
  #<module-path-index>
  1
  <lifted>
  #&
  <lifted>))
(argument-spec
  .
(#s((phased-module-binding module-binding 0 zo 0))
  #<module-path-index>
  1

define-syntax-rule
(define (flomap-lift-helper f) ...)

(define (flomap-lift-helper f) ...)

There must be a better way.
### Optimization Coach

20:0:

**flomap-lift-helper**

> Missed Inlining (0 success out of 46)

Consider turning this function into a macro to force inlining.
Dialog between compilers and programmers

404:34:
(/ ( - (* a2 z2) (* a1 z1))
 (flsqrt (+ (* u u) (* v v))))

✅ Float arithmetic specialization.

20:0:
flmap-lift-helper

❌ Missed Inlining (0 success out of 46)
Consider turning this function into a macro to force inlining.
Dialog between compilers and programmers

404:34:

(/ (- (* a2 z2) (* a1 z1))
 (flsqrt (+ (* u u) (* v v)))))

✔ Float arithmetic specialization.

20:0:

flmap-lift-helper

❌ Missed Inlining (0 success out of 46)
Consider turning this function into a macro to force inlining.

Successes
Dialog between compilers and programmers

Successes

Near misses

Recommendations
Programmers can do more than compilers

Recommendations can change semantics!

\[
\frac{1}{3} = 0.3333333333333333
\]
How does it work?

How well does it work?

How to extend it?
How does it work?
Overview

Compiler Instrumentation

Optimization Analysis

Recommendation Generation

Programmer Response
Type-Driven Specialization

#lang typed/racket/base

(define IM 139968)
(define IA 3877)
(define IC 29573)

(define last 42)
(define min 35.3)
(define max 156.8)
(define (gen-random)
    (set! last (modulo (+ (* last IA) IC) IM))
    (+ (/ (* (- max min) last) IM) min))
Type-Driven Specialization

```racket
#lang typed/racket/base

(define IM 139968)
(define IA 3877)
(define IC 29573)
(define last 42)
(define min 35.3)
(define max 156.8)
(define (gen-random)
  (set! last (modulo (+ (* last IA) IC) IM))
  (+ (/ (* (- max min) last) IM) min))
```
Type-Driven Specialization

#lang typed/racket/base

(define IM 139968)
(define IA 3877)
(define IC 29573)
(define last 42)
(define min 35.3)
(define max 156.8)
(define (gen-random)
  (set! last (modulo (+ (* last IA) IC) IM))
  (+ (/ (* (- max min) last) IM) min))
Compiler Instrumentation

Float Float

(- max min)

fl-

(- <Float> <Float>)

(fl- max min)

TR opt: prng-example.rkt 12:11
(- max min)
Float Float
binary float subtraction
Compiler Instrumentation

\[ (* (- \text{max} \text{min}) \text{last}) \]

\[ (* \text{<Number>} \text{<Number>}) \]; no change

\[ (* (- \text{max} \text{min}) \text{last}) \]

TR opt failure: prng-example.rkt 12:8
(* (- max min) last)
Float Integer
generic multiplication
Optimization Analysis

- Incomprehensible failure pruning
- Irrelevant failure pruning
- Harmless failure pruning
- Optimization proximity
- Causality merging
- Locality merging
Optimization Analysis

Incomprehensible failure pruning
Irrelevant failure pruning
Harmless failure pruning
Optimization proximity
Causality merging
Locality merging
Optimization Analysis

- Incomprehensible failure pruning
- Irrelevant failure pruning
- Harmless failure pruning
- Optimization proximity
- Causality merging
- Locality merging
Optimization Analysis

Incomprehensible failure pruning

Irrelevant failure pruning

Can't trace back to user program.

Optimization proximity

Causality merging

Locality merging
Optimization Analysis

Incomprehensible failure pruning

Irrelevant failure pruning

Non-optimized semantics is desirable.

Optimization proximity

Causality merging

Locality merging
Optimization Analysis

Optimization proximity

\[ (\star (- \max \min) \text{ last}) \]

\[ \Delta = 1 \]
Optimization Analysis

Optimization proximity

Near miss, report

Float

Float

Integer

Irritant

Δ = 1
Optimization Analysis

Optimization proximity

\[ \Delta = 2 \]
Optimization Analysis

Optimization proximity

Too far, don't report

\[ \Delta = 2 \]
Optimization Analysis

Causality merging

(\(\frac{\ast (- \text{max} \text{ min} \text{ last}) \text{ IM}}{}\))

Irritant

(\(\ast (- \text{max} \text{ min} \text{ last})\))

Irritant

Real

Integer

Float

Integer
Optimization Analysis

Causality merging

$$\left\lceil \left( \frac{\left(\ast \left(\left( - \text{max} \text{ min} \text{ last} \right) \text{ IM}\right) \right)}{\text{Irritant Irritant}} \right) \right\rceil$$

Badness = 2
Optimization Analysis

Causality merging
Optimization Analysis

Causality merging

Near Miss = Root

Badness = # Nodes

Irritants = Leaves
Recommendation Generation

12:5:

\((/ (* (- max min) last) IM)\)

This expression has a Real type. The optimizer could optimize it if it had type Float. To fix, change the highlighted expression(s) to have Float type(s).
Greg, I think your code is perfectly fine! Let me explain why.

In your code:

```
(->fl last) (->fl IM)
(+ (/ (* (- max min) last) IM) min)
```

You're converting an integer to a floating point number and then performing arithmetic operations. This is correct and expected. The code snippet you've provided compiles and runs as expected, and I don't see any issues with it.

The expression

```
(/ (* (- max min) last) IM)
```

is a valid floating point division. The type checking and conversion are handled correctly by the compiler. Therefore, there's no need to change the type of any variable or expression. The code is performing as intended.

I hope this clears up any confusion! If you have any questions, feel free to ask.
The Racket inliner in one slide

Based on a design by Serrano

Fuel

• Inlining cost $\propto$ Function size
• Inlining stops when out of fuel

Loop unrolling for free!
(define (negate-pixel pixel) ; small function
  (+ #x010101 (bitwise-xor pixel #x3f3f3f)))

(define (clamp-pixel pixel) ; large function
  ; Clamp required because JPG occasionally sends a delta too high or too low, leaving us with out-of-range pixels.
  ; Clamp each channel to [40, 7F].
  ...
)

(define (kernel-decode base-pixel delta-pixel)
  ... (negate-pixel ...) ...
  ... (clamp-pixel clampable) ...)

... (kernel-decode ...) ...
(define (negate-pixel pixel) ; small function
 (+ #x010101 (bitwise-xor pixel #x3f3f3f)))

(define (clamp-pixel clampable)
 ; Clamp required because JPG occasionally sends a delta too
 ; high or too low, leaving us with out-of-range pixels.
 ; Clamp each channel to [40, 7F].

(define (kernel-decode base-pixel delta-pixel)
 ... (negate-pixel ...) ... 
 ... (clamp-pixel clampable) ...)

... (kernel-decode ...) ...
(define (negate-pixel pixel) ; small function
  (+ #x010101 (bitwise-xor pixel #x3f3f3f)))

(define (clamp-pixel pixel) ; large function
  ; Clamp required because JPG occasionally sends a delta too high or too low, leaving us with out-of-range pixels.
  ; Clamp each channel to [40, 7F].
  ...
)

(define (kernel-decode base-pixel delta-pixel)
  ...
    (negate-pixel ...
    ...
    (clamp-pixel clampable ...
    ...
    (kernel-decode ...) ...

  Optimization Coach

  39:0:
  clamp-pixel

  Missed Inlining (0 success out of 10)
  Consider turning this function into a macro to force inlining.
(define (negate-pixel pixel) ; small function
  (+ #x010101 (bitwise-xor pixel #x3f3f3f)))

(define (clamp-pixel pixel)
  ; Clamp required because JPG occasionally sends a delta too high or too low, leaving us with out-of-range pixels.
  ; Clamp each channel to [40, 7F].
  ...
)

(define (kernel-decode base-pixel delta-pixel)
  ... (negate-pixel ...) ...
  ... (clamp-pixel clampable) ...)

... (kernel-decode ...) ...
inlining:
#(negate-pixel #<path:video.rkt> 28 0 793 73 #f)
in: video.rkt:18:0: kernel-decode
size: 1 fuel: 96

no inlining, out of fuel:
#(clamp-pixel #<path:video.rkt> 39 0 1228 534 #f)
in: video.rkt:18:0: kernel-decode
size: 99 fuel: 96
inlining:
#(negate-pixel #<path:video.rkt> 28 0 793 73 #f)
in: video.rkt:18:0: kernel-decode
size: 1 fuel: 96

no inlining, out of fuel:
#(clamp-pixel #<path:video.rkt> 39 0 1228 534 #f)
in: video.rkt:18:0: kernel-decode
size: 99 fuel: 96
Compiler Instrumentation

Inlining:
```
#(negate-pixel #<path:video.rkt> 28 0 793 73 #f)
in: video.rkt:18:0: kernel-decode
size: 1 fuel: 96
```

No inlining, out of fuel:
```
#(clamp-pixel #<path:video.rkt> 39 0 1228 534 #f)
in: video.rkt:18:0: kernel-decode
size: 99 fuel: 96
```
## Compiler Instrumentation

inlining:
```
#(negate-pixel #<path:video.rkt> 28 0 793 73 #f)
in: video.rkt:18:0: kernel-decode
size: 1 fuel: 96
```

no inlining, out of fuel:
```
#(clamp-pixel #<path:video.rkt> 39 0 1228 534 #f)
in: video.rkt:18:0: kernel-decode
size: 99 fuel: 96
```
Harmless Failure Pruning

no inlining, out of fuel:
#(for-loop #<path:video.rkt> 63 2 2353 281 #f)
in: video.rkt:63:2: for-loop
size: 52 fuel: 8

Unrolling has to stop at some point
Optimization Analysis

Locality Merging

4×

inlining:
#(kernel-decode #<path:video.rkt> 18 0 427 276 #f)

20×

no inlining, out of fuel:
#(kernel-decode #<path:video.rkt> 18 0 427 276 #f)

4 < 20 → Optimization failure

\[ \Delta = (size - fuel) \]
Recommendation Generation

No unrollings: function → macro

Otherwise: make smaller / break into pieces

Just over limit: **begin-encourage-inline**

Fast/slow path: break off slow path

Opt./kw. args: multiple specialized functions
define-syntax-rule
(define (clamp-pixel pixel) ; large function
  ; Clamp required because JPG occasionally sends a delta too
  ; high or too low, leaving us with out-of-range pixels.
  ; Clamp each channel to [40, 7F].
  ...)

define-syntax-rule
(define (kernel-decode base-pixel delta-pixel)
  ... (negate-pixel ...) ...
  ... (clamp-pixel clampable) ...)

How well does it work?
Baseline: Non-optimized
Coached: Followed recommendations (Minutes of work)

Lower is better
Baseline: Non-optimized
Coached: Followed recommendations  (Minutes of work)

Lower is better
Baseline: Non-optimized
Coached: Followed recommendations (Minutes of work)
Gold standard: Hand-optimized by experts (Days of work)

Lower is better
How to extend it?
Our prototype

{ Type-driven rewriting
  Inlining
  Common subexpression elimination
  Test reordering
  Scalar replacement
  Loop-invariant code motion
  Devirtualization
  Specialization of polymorphic containers
  Case-of-case transformation }
Our prototype

\{ 
Type-driven rewriting

Inlining

Common subexpression elimination

Test reordering

Scalar replacement

Loop-invariant code motion

Devirtualization

Specialization of polymorphic containers

Case-of-case transformation
\}

Coaching Recipe

- Log successes and failures
- Define optimization analysis metrics
- Add recommendation generation logic
Scalar replacement

(define twiddle-c
  (* c (exp (/ (* pi 0.0+1.0i n/2) ->fl k)))
  (+ b twiddle-c))

(define real-c (real-part c))
(define imag-c (imag-part c))
... 
(define real-twiddle-c ...)
(define imag-twiddle-c ...)

(define real-b (real-part b))
(define imag-b (imag-part b))
(make-rectangular
  (+ real-b real-twiddle-c)
  (+ imag-b imag-twiddle-c))
Scalar replacement

(define twiddle-c
  (* c (exp (/ (* pi 0.0+1.0i (->fl k))
           n/2))))

(printf "~a\n" twiddle-c)
(+ b twiddle-c)

(define real-c (real-part c))
(define imag-c (imag-part c))
...
(define real-twiddle-c ...)
(define imag-twiddle-c ...)

(define real-b (real-part b))
(define imag-b (imag-part b))
(make-rectangular
  (+ real-b real-twiddle-c)
  (+ imag-b imag-twiddle-c))
 Scalar replacement

\[
\Delta = \frac{\text{(# boxed uses)}}{\text{(# unboxed uses)}}
\]

Irritants = \{boxed uses\}
Specialization of polymorphic containers

(define-type 3D-path
  (Vectorof (List Float Float Float))))

(: p : 3D-path)
(define p (vector '(1.2 3.4 5.6)
                   '(7.8 9.1 0.1)
                   '(1.1 2.1 3.1)))

(define p (vector 1.2 3.4 5.6
                   7.8 9.1 0.1
                   1.1 2.1 3.1))
Specialization of polymorphic containers

(define-type 4D-path
  (Vectorof (List Float Float Float Float)))

(: p : 4D-path)
(define p (vector '(1.2 3.4 5.6 12.3)
                   '(7.8 9.1 0.1 45.6)
                   '(1.1 2.1 3.1 78.9)))
Specialization of polymorphic containers

(define-type 4D-path
  (Vectorof (List Float Float Float Float)))

(: p : 4D-path)
(define p (vector '(1.2 3.4 5.6 12.3)
                   '(7.8 9.1 0.1 45.6)
                   '(1.1 2.1 3.1 78.9)))

$n = $ (element size) - (max optimized element size)
Conclusion
The take-away

**Key idea:** The compiler *talks back*

*General* optimization analysis techniques

+ *Optimization-specific* heuristics

*Targeted* recommendations
The take-away

**Key idea:** The compiler *talks back*

**General** optimization analysis techniques

+ **Optimization-specific** heuristics

**Targeted** recommendations

racket-lang.org
Demo