LLVM Greedy Register Allocator – Improving Region Split Decisions

Marina Yatsina
Intel Corporation, Israel
April 16-17, 2018 European Developers Meeting
Bristol, United Kingdom
Motivation

```assembly
movl %ecx, %ebp
movl %ebx, %ecx
movl %edi, %ebx
movl %edx, %edi
cltd
movl 4(%esp), %esi
idivl %esi
movl %edi, %edx
movl %ebx, %edi
movl %ecx, %ebx
movl %ebp, %ecx
```
Motivation

```assembly
movl  %ecx, %ebp
movl  %ebx, %ecx
movl  %edi, %ebx
movl  %edx, %edi
cld
movl  4(%esp), %esi
idivl  %esi
movl  %edi, %edx
movl  %ebx, %edi
movl  %ecx, %ebx
movl  %ebp, %ecx
```
Motivation

```
movl %ecx, %ebp
movl %ebx, %ecx
movl %edi, %ebx
movl %edx, %edi
cltd
movl 4(%esp), %esi
idivl %esi
movl %edi, %edx
movl %ebx, %edi
movl %ecx, %ebx
movl %ebp, %ecx
```

* idiv implicitly clobbers %edx
Motivation

```
movl  %ecx, %ebp
movl  %ebx, %ecx
movl  %edi, %ebx
movl  %edx, %edi
cltd
movl  4(%esp), %esi
idivl %esi
```

* idiv implicitly clobbers %edx
Motivation

* idiv implicitly clobbers %edx
Motivation

```Assembly
movl %ecx, %ebp
movl %ebx, %ecx
movl %edi, %ebx
movl %edi, %edi
cltd
movl 4(%esp), %esi
idivl %esi
movl %edi, %edx
movl %ebx, %edi
movl %ecx, %ebx
movl %ebp, %ecx
```

* idiv implicitly clobbers %edx
Motivation

```
movl %edx, %ebp
cltd
movl 4(%ebp), %esi
idivl %esi
movl %ebp, %edx
```

* idiv implicitly clobbers %edx
Greedy Register Allocator

• Greedy Register Allocator Overview
• Region Split
• Encountered Issues
• Performance Impact
Greedy Register Allocator

• Greedy Register Allocator Overview
• Region Split
• Encountered Issues
• Performance Impact
High Level Design of Code Generator

Instruction Selection → Scheduling and Formation → SSA-based Machine Code Optimizations

Register Allocation ← Prolog/Epilog Code Insertion ← Late Machine Code Optimizations

Code Emission ← Code Emission
Greedy Register Allocator Overview

• General flow

Live Interval Analysis -> Spill Weight Calculation -> Priority Queue Construction

Register Assignment -> Eviction -> Split

Spill
Greedy Register Allocator Overview

- General flow

1. Live Interval Analysis
2. Spill Weight Calculation
3. Priority Queue Construction
4. Register Assignment
5. Eviction
6. Split
7. Spill
Greedy Register Allocator Overview

- General flow

  Stand alone pass
  Live Interval Analysis
  Register Assignment
  Spill
  Spill Weight Calculation
  Priority Queue Construction
  Eviction
  Split
Live Interval Analysis

BB#0:
\[ x = \ldots \]
\[ \ldots \]

BB#1:
\[ \ldots = \ldots x \ldots \]
\[ \ldots \]

BB#2:
\[ \ldots \]
\[ \ldots = \ldots x \ldots \]
\[ \ldots \]
Live Interval Analysis

• Earlier pass named SlotIndexes added numbering to the instructions

```
BB#0:
000  BB#0:
001  x = ...
002  ...

BB#1:
003  BB#1:
004  ... = ...x...
005  ...

BB#2:
006  BB#2:
007  ...
008  ... = ...x...
009  ...
```
Live Interval Analysis

- Analyze x’s live interval

- BB#0:
  - 001 x = ...
  - 002 ...

- BB#1:
  - 004 ... = ...x...
  - 005 ...

- BB#2:
  - 008 ... = ...x...
  - 009 ...

- x
Live Interval Analysis

- Look for uses and defs

```
000  BB#0:
001  x = ...
002  ...

003  BB#1:
004  ... = ...x...
005  ...

006  BB#2:
007  ...
008  ... = ...x...
009  ...
```
Live Interval Analysis

• Connect them into intervals

000  BB#0:
001  x = …
002  …

003  BB#1:
004  … = …x…
005  …

006  BB#2:
007  …
008  … = …x…
009  …

x
Live Interval Analysis

- Represented $x$'s liveness as a collection of segments

000  BB#0: 
001  $x = \ldots$
002  $\ldots$

003  BB#1: 
004  $\ldots = \ldots x \ldots$
005  $\ldots$

006  BB#2: 
007  $\ldots$
008  $\ldots = \ldots x \ldots$
009  $\ldots$

$x$ is live in intervals [001, 004), [006, 008)
Greedy Register Allocator Overview

- General flow

Live Interval Analysis → Spill Weight Calculation → Priority Queue Construction

Register Assignment → Eviction → Split

Spill
Greedy Register Allocator Overview

• General flow

RegAllocGreedy Pass

Live Interval Analysis → Spill Weight Calculation → Priority Queue Construction

Register Assignment → Eviction → Split

Spill
Greedy Register Allocator Overview

• General flow

Live Interval Analysis → Spill Weight Calculation → Priority Queue Construction

Register Assignment → Eviction → Split

Spill

Spill Weight Calculation

Eviction

Split
Greedy Register Allocator Overview

- General flow

For every interval:
- Live Interval Analysis
- Spill Weight Calculation
- Priority Queue Construction

Flow:
- Spill
- Register Assignment
- Eviction
- Split
Spill Weight Calculation

- Intervals calculated by Live Interval Analysis

\[ w \times x \times y \times z \]
Spill Weight Calculation

- Estimate spill weight of each interval based on interval characteristics
  - w has uses in a hot loop
    - Higher spill weight
  - x is cheaply rematerializable
    - Lower spill weight

<table>
<thead>
<tr>
<th></th>
<th>30 KG</th>
<th>5 KG</th>
<th>10 KG</th>
<th>20 KG</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Greedy Register Allocator Overview

- General flow

  - Live Interval Analysis
  - Spill Weight Calculation
  - Priority Queue Construction
  - Register Assignment
  - Eviction
  - Split
  - Spill
Greedy Register Allocator Overview

• General flow

Live Interval Analysis → Spill Weight Calculation → Priority Queue Construction

For every interval

Register Assignment → Eviction → Split

Spill
Priority Queue Construction

w
x
y
z
Priority Queue Construction

\[ w \] \[ x \] \[ y \] \[ z \]
Priority Queue Construction
Priority Queue Construction

- Calculate interval allocation priority and insert into the queue

\[ w \quad x \quad y \quad z \]
Priority Queue Construction

- Calculate interval allocation priority and insert into the queue
  - $w$ is local in one basic block
    - Lower allocation priority
Priority Queue Construction

• Calculate interval allocation priority and insert into the queue
  • x is global and spans across a lot of instructions
    • Higher allocation priority
Priority Queue Construction

• Calculate interval allocation priority and insert into the queue
Priority Queue Construction

- Calculate interval allocation priority and insert into the queue
Priority Queue Construction

• The Priority Queue will always dequeue the interval with the highest priority
Greedy Register Allocator Overview

• General flow

Live Interval Analysis → Spill Weight Calculation → Priority Queue Construction

Register Assignment → Eviction → Split

Spill
Greedy Register Allocator Overview

• General flow

For every interval in queue

Live Interval Analysis → Spill Weight Calculation → Priority Queue Construction

For every interval in queue

Register Assignment → Eviction → Split

Spill
Greedy Register Allocator Overview

• General flow

Live Interval Analysis → Spill Weight Calculation → Priority Queue Construction

Register Assignment → Eviction → Split

Spill
Register Assignment

- R0, R1 are the physical registers in the system
Register Assignment

Priority Queue

y w z x

R0 R1
Register Assignment

- Dequeue interval with highest priority
Register Assignment

Priority Queue

y  w  z

x

R0  R1
Register Assignment

• Assign to available register if possible
Register Assignment

- Dequeue interval with highest priority
Register Assignment

Priority Queue: w, y

Registers: R0, R1

Values:
- R0: x, x, x
- R1: 

Symbols:
- z
Register Assignment

- Assign to available register if possible
Register Assignment

• Dequeue interval with highest priority
Register Assignment

Priority Queue

w

R0

x

x

z

R1

x

x

z
Register Assignment

- Interference with x in R0
Register Assignment

- Interference with z in R1
Greedy Register Allocator Overview

- General flow

1. Live Interval Analysis
2. Spill Weight Calculation
3. Priority Queue Construction
4. Register Assignment
5. Eviction
6. Split
7. Spill
Eviction

- Compare spill weights of interfering intervals
Eviction

• x cheaper than w
Eviction

• Evict x
Eviction

• Assign w
Eviction
Eviction

- Enqueue x back to the queue
  - Usually receives the same allocation priority
Register Assignment
Register Assignment

• Dequeue interval with highest priority
Register Assignment

Priority Queue

x

R0

R1

y

w

z

z
Register Assignment

• Interference with w in R0
Register Assignment

• Interference with z in R1
Eviction

• Compare spill weights of interfering intervals
  • Can we Evict?

Priority Queue

x

5 KG

R0

R1

w

z

z

30 KG

20 KG
Eviction

• Compare spill weights of interfering intervals
  • Can we Evict? No!
  • x is the cheapest one
Register Assignment

Priority Queue

x

R0

w

R1

z

z
Register Assignment

• Mark x to be split
Register Assignment

Priority Queue

R0
- W
R1
- Z
- Z

x*
Register Assignment

- Enqueue $x^*$ back to the queue
- Intervals marked to be split receive lower allocation priority
Register Assignment

Priority Queue

R0
- w
R1
- z
- z
Register Assignment

- Dequeue interval with highest priority
Register Assignment

Priority Queue

y

R0
w

R1
z
Register Assignment

• Assign to available register if possible
Register Assignment

- Dequeue interval with highest priority
Register Assignment

• x is marked to be split
Greedy Register Allocator Overview

- General flow

- Live Interval Analysis → Spill Weight Calculation → Priority Queue Construction

- Register Assignment → Eviction → Split

- Spill
Split

- Is split beneficial?

Priority Queue
Split

• Is split beneficial?
  • (Assume) Yes!

Priority Queue
Split

• Is split beneficial?
  • (Assume) Yes!
  • Find the best way to split

R0
  - w
  - y
R1
  - z
  - z

Priority Queue
Split

- Do the split
- Add COPY instruction between the intervals

Priority Queue

<table>
<thead>
<tr>
<th>x*</th>
<th>x1</th>
<th>x2</th>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>y</td>
<td>z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>y</td>
<td>z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x2 = COPY x1
x1 = COPY x2
Split

Priority Queue

x1  x2  R0  R1

w  y  z  z
Split

• Calculate spill weights
  • Split artifacts may receive higher weight than the original interval
Split

Priority Queue

R0
- w

R1
- z

x1
- y

x2
Split

- Enqueue x1, x2 into the queue
  - This will also calculate their allocation priority
Register Assignment

Priority Queue: x2, x1

R0: w, y
R1: z, z
Register Assignment

• Dequeue interval with highest priority
Register Assignment

Priority Queue

x1

R0

R1

w

y

z

z
Register Assignment

- Assign to available register if possible
Register Assignment

• Dequeue interval with highest priority
Register Assignment
Register Assignment

- Interference with y in R0
Register Assignment

• Interference with z in R1
Register Assignment

Priority Queue

x2

R0
w
y
z

R1
x1
x1
x1
Eviction

• Compare spill weights of interfering intervals
Eviction

• y cheaper than x2
Eviction

• Evict y
Eviction

- A split artifact can evict original interval
- Part of the split-eviction gradual refinement

Priority Queue

\[ x_2 \quad y \]

\[ \text{R0} \quad \text{R1} \]

\[ w \quad z \quad x_1 \quad x_1 \quad z \quad x_1 \quad x_1 \]
Eviction

• Assign x2
Eviction
Eviction

• Enqueue y back to the queue
Register Assignment
Register Assignment

- Dequeue interval with highest priority
Register Assignment
Register Assignment

- Interference with x2 in R0
Register Assignment

- Interference with z in R1
Eviction

- Compare spill weights of interfering intervals
  - Can we Evict?

![Priority Queue Diagram]

- Priority Queue
- R0
- R1

10 KG
15 KG
20 KG
Eviction

• Compare spill weights of interfering intervals
  • Can we Evict? No!
  • $y$ is the cheapest one

Priority Queue

<table>
<thead>
<tr>
<th></th>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$x_2$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$x_1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$w$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>10 KG</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 KG</td>
<td></td>
</tr>
<tr>
<td>20 KG</td>
<td></td>
</tr>
</tbody>
</table>
Eviction

Priority Queue

y

R0
- w
- x2
- x2

R1
- x1
- z
- x1
- z
- x1
Register Assignment

• Mark y to be split
Register Assignment

Priority Queue

$y^*$

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>x1</td>
</tr>
<tr>
<td>x2</td>
<td>z</td>
</tr>
<tr>
<td>x2</td>
<td>x1</td>
</tr>
<tr>
<td>x1</td>
<td>z</td>
</tr>
<tr>
<td>x1</td>
<td>x2</td>
</tr>
</tbody>
</table>
Register Assignment

- Enqueue \( y^* \) back to the queue
Register Assignment

Priority Queue

R0
- w
- x2
- x2

R1
- x1
- z
- x1
- z
- x1
Register Assignment

• Dequeue interval with highest priority
Register Assignment

- Register Assignment

Priority Queue

\[
y^* \quad \begin{array}{c}
\text{R0} \\
\text{R1}
\end{array}
\]

\[
\begin{array}{c}
w \\
x_2 \\
x_2 \\
z \\
x_1 \\
x_1
\end{array}
\]
Split

- Is split beneficial?
Split

• Is split beneficial?
  • (Assume) No!
Greedy Register Allocator Overview

- General flow

  - Live Interval Analysis
  - Spill Weight Calculation
  - Priority Queue Construction
  - Register Assignment
  - Eviction
  - Split
  - Spill
Spill
Spill

- Spill around uses
  - Spill after def
  - Reload before use
Spill

• Create new intervals for spills and reloads
Spill

• Calculate spill weights

![Priority Queue diagram]

- y1
- y2
- R0
- R1

- w
- z
- x1
- x2

<table>
<thead>
<tr>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 KG</td>
</tr>
<tr>
<td>2 KG</td>
</tr>
</tbody>
</table>
Spill

- Spill

Priority Queue
Spill

• Enqueue y1, y2 into the queue
  • This will also calculate their allocation priority

Priority Queue

\[ \begin{align*}
  &y1 & y2 \\
  &R0 & R1 \\
  &w & x1 \\
  &z & x1 \\
  &x2 & z \\
  &x2 & x1 \\
\end{align*} \]
Register Assignment

- Dequeue interval with highest priority
Register Assignment

Priority Queue

y1

y2

R0

w

x2

x2

R1

x1

z

x1

z

x1
Register Assignment

- Assign to available register if possible
Register Assignment

• Dequeue interval with highest priority
Register Assignment

Priority Queue

R0
w
x2
y2
x2

R1
x1
z
x1
z
x1
Register Assignment

• Assign to available register if possible

Priority Queue

R0

w

x2

y2

x2

R1

x1

z

x1

y1

z

x1
Greedy Register Allocator

- Greedy Register Allocator Overview
- Region Split
- Encountered Issues
- Performance Impact
Motivation

```
movl $ecx, %ebp
movl $ebx, $ecx
movl $edi, $ebx
movl $edx, $edi
```

cld
movl $esi, %esp
idivl %esi
movl %edi, %edx
movl %ebx, %edi
movl %ecx, %ebx
movl %ebp, %ecx
```
Exploration

- Why did the register allocator create these redundant mov instructions?
Exploration

• Why did the register allocator create these redundant mov instructions?

• Artifacts of split

\[ x^* \rightarrow x_1 \rightarrow x_2 \]

\[ \begin{align*}
  x_2 &= \text{COPY} \, x_1 \\
  x_1 &= \text{COPY} \, x_2
\end{align*} \]
Exploration

• Why did the register allocator create these redundant mov instructions?
  • Artifacts of split
  • If we would have chosen to do the split differently we could have avoided the redundant mov instructions
  • Why was this way to split was chosen?
Greedy Register Allocator Overview

• General flow

1. Live Interval Analysis
2. Spill Weight Calculation
3. Priority Queue Construction
4. Register Assignment
5. Eviction
6. Split

Spill
Region Split

• How to find the best way to split?
• How to know if split is beneficial?

Priority Queue
Find Best Split
Find Best Split

• The registers already have assigned intervals
Find Best Split

• The registers already have assigned intervals
Find Best Split

• The registers already have assigned intervals

• These intervals impose allocation constraints
Find Best Split

- Do the split of x for each one of the registers
Find Best Split

• Do the split of \( x \) for each one of the registers
Find Best Split

• Do the split of $x$ for each one of the registers

$x$  \hspace{2cm} R0  \hspace{2cm} R1  \hspace{2cm} R2  \hspace{2cm} \ldots  \hspace{2cm} Rn
Find Best Split

- Do the split of $x$ for each one of the registers
Find Best Split

- Do the split of x for each one of the registers
- Estimate split cost, e.g. the amount of spill code this split may cause
Find Best Split

• Do the split of $x$ for each one of the registers
  • Choose the cheapest one

$x$

R0  R1  R2  ...  Rn

20$  10$  15$  ...  20$
Find Best Split

- Do the split of $x$ for each one of the registers
Find Best Split for Given Register

• How do we do the best split for a given register R1?
Find Best Split for Given Register

- The region split is usually divided into 2 intervals

\[
x = \begin{cases} 
  x_1 & \text{COPY } x_1 \\
  x_2 & \text{COPY } x_2 
\end{cases}
\]
Find Best Split for Given Register

• The region split is usually divided into 2 intervals
  • ByReg
    • Parts of x that pass on R1 register
    • Should comply with current allocation constraints provided by intervals already assigned to R1
  • ByStack
    • Parts of x that pass on or on the stack
    • Usually where the already allocated R1 intervals interfere with x
Find Best Split for Given Register

• A good split
Find Best Split for Given Register

• A good split
  • Reduces the transitions between ByReg and ByStack
    • Each such transitions is potentially a spill/reload
      • In case ByStack is not allocated to another register
  • Places the transitions in blocks less frequently executed
Find Best Split for Given Register

- A good split
  - Reduces the transitions between ByReg and ByStack
    - Each such transition is potentially a spill/reload
      - In case ByStack is not allocated to another register
  - Places the transitions in blocks less frequently executed
  - Use Hopfield neural network
    - Converges to a result that satisfies the above characteristics

\[ x = \text{COPY} \ x_1 \to \ x_2 \]
\[ x_1 = \text{COPY} \ x_2 \]
Determine if Split is Beneficial

- Split reduces the amount of spills compared to spilling around uses
Determine if Split is Beneficial

- Split reduces the amount of spills compared to spilling around uses
- Use/def blocks must have x in a register at some point
Determine if Split is Beneficial

- Split reduces the amount of spills compared to spilling around uses
  - Use/def blocks must have x in a register at some point
  - If the split can create “regions” of several basic blocks where x is passed by register this will reduce the amount of spills
    - Only if constraints allow it
Determine if Split is Beneficial

- Split reduces the amount of spills compared to spilling around uses
  - Use/def blocks must have x in a register at some point
  - If the split can create “regions” of several basic blocks where x is passed by register this will reduce the amount of spills
    - Only if constraints allow it
Greedy Register Allocator

- Greedy Register Allocator Overview
- Region Split
- Encountered Issues
- Performance Impact
Region Split Cost Issues

- Inaccurate split cost calculation
  - Root cause of the following encountered issues
- Does not model the affect of local interference caused by the split
  - Makes the split cost inaccurate
  - The “cheapest” split may actually be more expensive than other splits
  - Can choose suboptimal split
Local Interference

- Find best split of interval x for R1
  - Using Hopfield neural network
Local Interference

• Find best split of interval $x$ for $R1$
  • Using Hopfield neural network
  • The network determines how $x$ will be passed on the CFG edges

![Diagram showing local interference with BB#1 and R1]
Local Interference

- Find best split of interval x for R1
  - Using Hopfield neural network
  - The network determines how x will be passed on the CFG edges
    - “ByReg” interval or “By stack” interval
Local Interference

• Find best split of interval x for R1
  • Using Hopfield neural network
  • The network determines how x will be passed on the CFG edges
    • “ByReg” interval or “By stack” interval
    • Determined which basic block will have a copy between these two intervals
Local Interference

- Find best split of interval x for R1
  - Using Hopfield neural network
  - The network determines how x will be passed on the CFG edges
    - “ByReg” interval or “By stack” interval
    - Determined which basic block will have a copy between these two intervals
  - The Hopfield neural network does not model what happens to x inside the basic block
Local Interference

- The Hopfield neural network does not model what happens to x inside the basic block.
Local Interference

- The Hopfield neural network does not model what happens to x inside the basic block
- x split for R1 determined x’s ByReg interval should enter and leave BB#1
Local Interference

- The Hopfield neural network does not model what happens to x inside the basic block
  - x split for R1 determined x’s ByReg interval should enter and leave BB#1
  - y in BB#1 is already assigned to R1
Local Interference

- The Hopfield neural network does not model what happens to x inside the basic block
- x split for R1 determined x’s ByReg interval should enter and leave BB#1
- y in BB#1 is already assigned to R1
- x is used in BB#1
Local Interference

- The Hopfield neural network does not model what happens to x inside the basic block
  - x split for R1 determined x’s ByReg interval should enter and leave BB#1
  - y in BB#1 is already assigned to R1
  - x is used in BB#1
  - y interferes with assigning x to R1 locally in BB#1
Local Interference

- The Hopfield neural network does not model what happens to $x$ inside the basic block
  - $x$ split for R1 determined $x$’s ByReg interval should enter and leave BB#1
  - $y$ in BB#1 is already assigned to R1
  - $x$ is used in BB#1
  - $y$ interferes with assigning $x$ to R1 locally in BB#1
    - The part of $x$ that contains this local interference will be added to $x$’s “ByStack” split artifact
Local Interference

- Local interferences may have very negative affects on assignment of the “ByStack” split artifact
- Can cause bad eviction chains
  - Encountered issues #1, #2
- Can cause a lot of reloads
  - Encountered issue #3
- This affect is not considered during split cost calculation
Encountered Issue

• Bad eviction chain
  • Cyclic eviction/split chain – Issue #1
  • Domino effect eviction – Issue #2

• Multiple reloads from the same location
  • Issue #3
Encountered Issue #1

- Bad eviction chain – scenario 1

- llvm/test/CodeGen/X86/greedy_regalloc_bad_eviction_sequence.ll

```assembly
movl  $ecx, $ebp
movl  $ebx, $ecx
movl  $edi, $ebx
movl  $edx, $edi
cld
movl  4($esp), $esi
idivl  $esi
movl  $edi, $edx
movl  $ebx, $edi
movl  $ecx, $ebx
movl  $ebp, $ecx
```
Encountered Issue #1

• Bad eviction chain – scenario 1

• Ilvm/test/CodeGen/X86/
  greedy_regalloc_bad_eviction_sequence.ll

```
movl %ecx, %ebp
movl %ebx, %ecx
movl %edi, %ebx
movl %edi, %edi
cld
movl 4(%esp), %esi
idivl %esi
movl %edi, %edi
movl %ebx, %edi
movl %ecx, %ebx
movl %ecx, %ebx
```
Encountered Issue #1

• Bad eviction chain – scenario 1
  • Cyclic eviction/split chain
Encountered Issue #1

- Bad eviction chain – scenario 1
  - Cyclic eviction/split chain

```
movl   %ecx, %ebp
movl   %ebx, %ecx
movl   %edi, %ebx
movl   %edx, %edi
cltd
movl  4(%esp), %esi
idivl %esi
movl   %edi, %edx
movl   %ebx, %edi
movl   %ecx, %ebx
movl   %ebp, %ecx
```
Encountered Issue #1

- Bad eviction chain – scenario 1
  - Cyclic eviction/split chain
    - \textit{y evicts x} from edi
Encountered Issue #1

• Bad eviction chain – scenario 1
  • Cyclic eviction/split chain
  • \texttt{y evicts x} from \texttt{edi}
Encountered Issue #1

- Bad eviction chain – scenario 1
  - Cyclic eviction/split chain
    - \textit{y evicts x} from \textit{edi}
Encountered Issue #1

- Bad eviction chain – scenario 1
  - Cyclic eviction/split chain
    - y evicts x from edi
Encountered Issue #1

• Bad eviction chain – scenario 1
  • Cyclic eviction/split chain
    • y evicts x from edi
    • x is split into x1 and x2
Encountered Issue #1

- Bad eviction chain – scenario 1
  - Cyclic eviction/split chain
    - \texttt{y evicts x} from \texttt{edi}
    - \texttt{x} is split into \texttt{x1} and \texttt{x2}
Encountered Issue #1

- **Bad eviction chain – scenario 1**
  - **Cyclic eviction/split chain**
    - y *evicts* x from edi
    - x is split into x1 and x2
Encountered Issue #1

- Bad eviction chain – scenario 1
  - Cyclic eviction/split chain
    - \texttt{y evicts x} from edi
    - x is split into x1 and x2
      - x1 represent the part of the split that has local interference with y
Encountered Issue #1

• Bad eviction chain – scenario 1
  • Cyclic eviction/split chain
    • y evicts x from edi
    • x is split into x1 and x2
      • x1 represents the part of the split that has local interference with y
Encountered Issue #1

• Bad eviction chain – scenario 1
  • Cyclic eviction/split chain
    • \textbf{y evicts x} from edi
    • x is split into x1 and x2
      • x1 represent the part of the split that has local interference with y
    • \textbf{x1 evicts y} from edi

\begin{align*}
\text{movl} & \quad \text{ecx, ebx} \\
\text{movl} & \quad \text{ebx, ecx} \\
\text{movl} & \quad \text{edi, ebx} \\
\text{movl} & \quad \text{edx, edi} \\
\text{cltd} & \\
\text{movl} & \quad 4(\text{esp}), esi \\
\text{idivl} & \quad esi \\
\text{movl} & \quad \text{edi, edx} \\
\text{movl} & \quad \text{ebx, edi} \\
\text{movl} & \quad \text{ecx, ebx} \\
\text{movl} & \quad \text{ebp, ecx}
\end{align*}
Encountered Issue #1

• Bad eviction chain – scenario 1
  • Cyclic eviction/split chain
    • y evicts x from edi
    • x is split into x1 and x2
      • x1 represent the part of the split that has local interference with y
    • x1 evicts y from edi
Encountered Issue #1

• Bad eviction chain – scenario 1
  • Cyclic eviction/split chain
    • y evicts x from edi
    • x is split into x1 and x2
      • x1 represent the part of the split that has local interference with y
    • x1 evicts y from edi
Encountered Issue #1

• Bad eviction chain – scenario 1
  • Cyclic eviction/split chain
    • Every such “movl” duo was created by cyclic eviction/split chain
Encountered Issue #1

- Bad eviction chain – scenario 1
  - Cyclic eviction/split chain
    - Every such “movl” duo was created by cyclic eviction/split chain
Encountered Issue #1

• Bad eviction chain – scenario 1
  • Cyclic eviction/split chain
    • Every such “movl” duo was created by cyclic eviction/split chain
Encountered Issue #1

• Bad eviction chain – scenario 1
  • The problem
    • $x$ is split in such a way that creates local interference split artifact
    • That artifact causes cyclic eviction
  • The solution
    • Tailored for this case
      • Identify if a split will create a local interference artifact
      • Identify if that split artifact will cause a cyclic eviction
      • Increase split cost
        • Make this split less attractive compared to other splits
    • Commit: https://reviews.llvm.org/rL316295
Encountered Issue #2

- Bad eviction chain – scenario 2
  - [https://bugs.llvm.org/show_bug.cgi?id=26810](https://bugs.llvm.org/show_bug.cgi?id=26810)
Encountered Issue #2

- Bad eviction chain – scenario 2
  - https://bugs.llvm.org/show_bug.cgi?id=26810
  - This time parts of the chain are spread around

![Eviction Chain Diagram]

```
movaps %xmm3, %xmm4
mulps %xmm0, %xmm1
addps %xmm1, %xmm2
movaps 48(%esp), %xmm1
movaps %xmm2, %xmm3
movaps (%esp), %xmm2
movaps %xmm2, (%esp)
movaps %xmm3, %xmm4
movaps %xmm4, %xmm5
movaps %xmm5, %xmm6

%ymm7 %ymm5 %ymm4 %ymm3 %ymm2
%ymm7 %ymm5 %ymm4 %ymm3
%ymm5 %ymm4 %ymm3 %ymm2
```
Encountered Issue #2

- Bad eviction chain – scenario 2
- Domino effect eviction
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - x evicts y from xmm2
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - x evicts y from xmm2
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - $x$ evicts $y$ from $\text{xmm2}$
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - x evicts y from xmm2
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - x evicts y from xmm2
    - y is split into y1 and y2 for xmm2
Encountered Issue #2

• Bad eviction chain – scenario 2
  • Domino effect eviction
    • x evicts y from xmm2
    • y is split into y1 and y2 for xmm2
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - x evicts y from xmm2
    - y is split into y1 and y2 for xmm2
Encountered Issue #2

• Bad eviction chain – scenario 2
  • Domino effect eviction
    • x evicts y from xmm2
    • y is split into y1 and y2 for xmm2
      • y1 represent the part of the split that has local interference with x
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - x evicts y from xmm2
    - y is split into y1 and y2 for xmm2
      - y1 represents the part of the split that has local interference with x
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - x evicts y from xmm2
    - y is split into y1 and y2 for xmm2
      - y1 represent the part of the split that has local interference with x
    - y1 cannot evict x from xmm2

```plaintext
movapd %xmm3, %xmm4
movapd %xmm2, %xmm3
movapd %xmm0, %xmm1
movapd %xmm1, %xmm2
mulpd %xmm0, %xmm1
addpd %xmm1, %xmm2
mulpd %esp, %xmm2
movapd 48(%esp), %xmm1
movapd %xmm2, %xmm3
movapd %esp, %xmm2
mulpd %xmm0, %xmm1
mulpd 32(%esp), %xmm0
subpd %xmm1, %xmm2
movapd %xmm1, %xmm3
movapd 16(%esp), %xmm1
movapd %xmm2, (%esp)
movapd %xmm3, %xmm1
movapd %xmm4, %xmm5
movapd %xmm5, %xmm4
movapd %xmm7, %xmm5
```
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - x evicts y from xmm2
    - y is split into y1 and y2 for xmm2
      - y1 represent the part of the split that has local interference with x
    - y1 cannot evict x from xmm2

```assembly
movapd %xmm3, %xmm1
mulpd %xmm0, %xmm1
addpd %xmm1, %xmm2
movapd 48(%esp), %xmm1
movapd %xmm2, %xmm3
movapd (%esp), %xmm2
mulpd %xmm0, %xmm1
mulpd 32(%esp), %xmm0
subpd %xmm1, %xmm2
movapd 16(%esp), %xmm1
movapd %xmm2, (%esp)
```

```
movapd %xmm3, %xmm2
movapd %xmm4, %xmm5
movapd %xmm5, %xmm4
movapd %xmm7, %xmm5
```
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - x evicts y from xmm2
    - y is split into y1 and y2 for xmm2
      - y1 represent the part of the split that has local interference with x
    - y1 cannot evict x from xmm2

```
movapd %xmm3, %xmm1
mulpd %xmm0, %xmm1
addpd %xmm1, %xmm2
movapd 48(%esp), %xmm1
movapd %xmm2, %xmm3
movapd (%esp), %xmm2
mulpd %xmm0, %xmm1
mulpd 32(%esp), %xmm0
subpd %xmm1, %xmm2
movapd 16(%esp), %xmm1
movapd %xmm2, (%esp)
movapd %xmm3, %xmm2
movapd %xmm4, %xmm5
movapd %xmm5, %xmm4
movapd %xmm7, %xmm5
```
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - x evicts y from xmm2
    - y is split into y1 and y2 for xmm2
      - y1 represents the part of the split that has local interference with x
    - y1 cannot evict x from xmm2
    - y1 evicts z from xmm3
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - x evicts y from xmm2
    - y is split into y1 and y2 for xmm2
      - y1 represent the part of the split that has local interference with x
    - y1 cannot evict x from xmm2
    - y1 evicts z from xmm3
Encountered Issue #2

• Bad eviction chain – scenario 2
  • Domino effect eviction
    • x evicts y from xmm2
    • y is split into y1 and y2 for xmm2
      • y1 represent the part of the split that has local interference with x
      • y1 cannot evict x from xmm2
    • y1 evicts z from xmm3
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - x evicts y from xmm2
    - y is split into y1 and y2 for xmm2
      - y1 represent the part of the split that has local interference with x
    - y1 cannot evict x from xmm2
    - y1 evicts z from xmm3
Encountered Issue #2

• Bad eviction chain – scenario 2
  • Domino effect eviction
    • y1 evicts z from xmm3
Encountered Issue #2

• Bad eviction chain – scenario 2
  • Domino effect eviction
    • y1 evicts z from xmm3
    • z is split into z1 and z2 for xmm3
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - $y_1$ evicts $z$ from $x_{mm3}$
    - $z$ is split into $z_1$ and $z_2$ for $x_{mm3}$
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - $y_1$ evicts $z$ from $xmm3$
    - $z$ is split into $z_1$ and $z_2$ for $xmm3$
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - y1 evicts z from xmm3
    - z is split into z1 and z2 for xmm3
      - z1 represent the part of the split that has local interference with y1
Encountered Issue #2

• Bad eviction chain – scenario 2
  • Domino effect eviction
    • y1 evicts z from xmm3
    • z is split into z1 and z2 for xmm3
      • z1 represent the part of the split that has local interference with y1
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - y1 evicts z from xmm3
    - z is split into z1 and z2 for xmm3
      - z1 represent the part of the split that has local interference with y1
    - z1 cannot evict y1 from xmm3

```
movapd  %xmm3, %xmm1
mulpd  %xmm0, %xmm1
addpd  %xmm1, %xmm2
movapd  48(%esp), %xmm1
movapd  %xmm2, %xmm3
```

```
movapd  %xmm0, %xmm1
mulpd  %xmm0, %xmm1
mulpd  32(%esp), %xmm0
subpd  %xmm1, %xmm2
movapd  16(%esp), %xmm1
movapd  %xmm2, (%esp)
movapd  %xmm3, %xmm2
movapd  %xmm4, %xmm3
movapd  %xmm5, %xmm4
movapd  %xmm7, %xmm5
```
Encountered Issue #2

• Bad eviction chain – scenario 2
  • Domino effect eviction
    • y₁ evicts z from xmm3
    • z is split into z₁ and z₂ for xmm3
      • z₁ represent the part of the split that has local interference with y₁
    • z₁ cannot evict y₁ from xmm3

\[
\begin{align*}
x & \quad \text{Evicts} \\
y & \quad \text{Split} \\
z & \quad \text{Evicts} \\
\text{Interfering part of y} & \\
z_1 & \quad \text{Evicts} \\
\text{Split} \\
\text{Interfering part of z} \\
xmm3 & \\
y_1 & \\
z_1
\end{align*}
\]
Encountered Issue #2

- **Bad eviction chain – scenario 2**
  - Domino effect eviction
    - y1 evicts z from xmm3
    - z is split into z1 and z2 for xmm3
      - z1 represent the part of the split that has local interference with y1
    - z1 cannot evict y1 from xmm3

```assembly
movapd %xmm3, %xmm1
mulpd %xmm0, %xmm1
addpd %xmm1, %xmm2
movapd 48(%esp), %xmm1
movapd %xmm2, %xmm3
movapd (%esp), %xmm2
mulpd %xmm0, %xmm1
mulpd 32(%esp), %xmm0
subpd %xmm1, %xmm2
movapd 16(%esp), %xmm1
movapd %xmm2, (%esp)
movapd %xmm3, %xmm5
movapd %xmm4, %xmm5
movapd %xmm5, %xmm4
movapd %xmm7, %xmm5
```
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - $y_1$ evicts $z$ from $x_{mm3}$
    - $z$ is split into $z_1$ and $z_2$ for $x_{mm3}$
      - $z_1$ represents the part of the split that has local interference with $y_1$
    - $z_1$ cannot evict $y_1$ from $x_{mm3}$
    - $z_1$ evicts $w$ from $x_{mm4}$
Encountered Issue #2

• Bad eviction chain – scenario 2
  • Domino effect eviction
    • $y_1$ evicts $z$ from xmm3
    • $z$ is split into $z_1$ and $z_2$ for xmm3
      • $z_1$ represent the part of the split that has local interference with $y_1$
    • $z_1$ cannot evict $y_1$ from xmm3
    • $z_1$ evicts $w$ from xmm4
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - y1 evicts z from xmm3
    - z is split into z1 and z2 for xmm3
      - z1 represent the part of the split that has local interference with y1
    - z1 cannot evict y1 from xmm3
    - z1 evicts w from xmm4

Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - y1 evicts z from xmm3
    - z is split into z1 and z2 for xmm3
      - z1 represent the part of the split that has local interference with y1
    - z1 cannot evict y1 from xmm3
    - z1 evicts w from xmm4
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - Every such “movl” duo was created by the domino effect
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - Every such “movl” duo was created by the domino effect
Encountered Issue #2

- Bad eviction chain – scenario 2
  - Domino effect eviction
    - Every such “movl” duo was created by the domino effect
Encountered Issue #2

• Bad eviction chain – scenario 2
  • The problem
    • y is split to fit the register it was evicted from
    • This split creates local interference split artifact that causes domino effect eviction
  • The solution
    • Tailored for this case
      • Identify if a split for evicted register creates local interference artifact
      • Identify if that split artifact will cause domino effect eviction
      • Increase split cost
        • Make this split less attractive compared to other splits
    • Commit: https://reviews.llvm.org/rL316295
Encountered Issue #3

• Multiple reloads from the same location
Encountered Issue #3

- Multiple reloads from the same location
  - All the reloads are from the same location

```
 movl 12(esp), %ecx       # 4-byte Reload
 movzb1 (%ecx,%ebp), %ecx
 addl %edx, %edi
 addl %edi, %eax
 movl 12(esp), %edi       # 4-byte Reload
 movl %ecx, %edx
 shll $8, %edx
 movzb1 1(%edi,%ebp), %edi
 shll %ecx, %edx
 movl 12(esp), %ecx      # 4-byte Reload
 movzb1 -1(%ecx,%ebp), %ecx
```
Encountered Issue #3

- Multiple reloads from the same location
  - All the reloads are from the same location
  - Appeared in a hot loop after a higher level change

```
movl  12(esp), %ecx         # 4-byte Reload
movzl  (ecx,ebp), %ecx
addl  %edx, %edi
addl  %edi, %eax

movl  12(esp), %edi         # 4-byte Reload
movl  %ecx, %edx
shll  $8, %edx
movzl  1(%edi,ebp), %edi
shll  %ecx, %edx

movl  12(esp), %ecx         # 4-byte Reload
movzl  -1(%ecx,ebp), %ecx
```
Encountered Issue #3

- Multiple reloads from the same location

<table>
<thead>
<tr>
<th>Before Change</th>
<th>After Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop MBB is the</td>
<td>Loop MBB is the</td>
</tr>
<tr>
<td>same until Greedy</td>
<td>same until Greedy</td>
</tr>
</tbody>
</table>

```
movl 12(%esp), %ecx           # 4-byte Reload
movzl (ecx, %ebp), %ecx
addl %edx, %edi
addl %edi, %eax

movl 12(%esp), %edi           # 4-byte Reload
movl %ecx, %edx
shll $8, %edx
movzl (edi, %ebp), %edi
shll %ecx, %edx
movl 12(%esp), %ecx          # 4-byte Reload
movzl -1(%%ecx, %ebp), %ecx
```
Encountered Issue #3

- Multiple reloads from the same location

<table>
<thead>
<tr>
<th>Before Change</th>
<th>After Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop MBB is the same until Greedy</td>
<td>Loop MBB is the same until Greedy</td>
</tr>
<tr>
<td>x is split for R0</td>
<td>x is split for R1</td>
</tr>
</tbody>
</table>

Code snippet:

```assembly
movl 12(%esp), %ecx # 4-byte Reload
movzbl (%ecx,%ebp), %ecx
addl %edx, %edi
addl %edi, %eax
movl 12(%esp), %edi # 4-byte Reload
movl %ecx, %edx
shll $8, %edx
movzbl 1(%edi,%ebp), %edi
subl %ecx, %edx
movl 12(%esp), %ecx # 4-byte Reload
movzbl -1(%ecx,%ebp), %ecx
```
Encountered Issue #3

- Multiple reloads from the same location

<table>
<thead>
<tr>
<th>Before Change</th>
<th>After Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop MBB is the same until Greedy</td>
<td>Loop MBB is the same until Greedy</td>
</tr>
<tr>
<td>x is split for R0</td>
<td>x is split for R1</td>
</tr>
<tr>
<td>Split doesn’t have local interferences</td>
<td>Split has local interference in Loop’s MBB</td>
</tr>
</tbody>
</table>
Encountered Issue #3

- Multiple reloads from the same location

<table>
<thead>
<tr>
<th>Before Change</th>
<th>After Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop MBB is the same until Greedy</td>
<td>Loop MBB is the same until Greedy</td>
</tr>
<tr>
<td>x is split for R0</td>
<td>x is split for R1</td>
</tr>
<tr>
<td>Split doesn’t have local interferences</td>
<td>Split has local interference in Loop’s MBB</td>
</tr>
<tr>
<td></td>
<td>Local interference spilled &amp; reloaded around uses</td>
</tr>
</tbody>
</table>

```assembly
movl 12(esp), %ecx     # 4-byte Reload
movzbl (%ecx,%ebp), %ecx
addl %edx, %edi
addl %edi, %eax
movl 12(esp), %edi     # 4-byte Reload
movl %ecx, %edx
shll $8, %edx
movzbl 1(%edi,%ebp), %edi
subl %ecx, %edx
movl 12(esp), %ecx     # 4-byte Reload
movzbl -1(%ecx,%ebp), %ecx
```
Encountered Issue #3

- Multiple reloads from the same location
  - The problem
    - Local interference interval has a lot of uses
    - This interval is spilled and reloaded
  - Solution
    - Identify if the created local interference interval will spill
    - Increase split cost
      - Make this split less attractive compared to other splits
    - Commit: https://reviews.llvm.org/rL323870
Greedy Register Allocator

• Greedy Register Allocator Overview
• Region Split
• Encountered Issues
• Performance Impact
Fix for Bad Eviction Chains - Issues #1, #2

- Fix affected mostly EEMBC workloads
- Regressions unrelated to this change
- No actual compile time impact on CTMark
Fix for Multiple Reloads - Issue #3

• Fix affected mostly EEMBC workloads
• No actual compile time impact on CTMark
Conclusions

• Local interference caused by split may have a negative affect
• Current split cost does not take this affect into account
• Committed solutions tailored to catch 3 specific scenarios
• Need a more holistic approach for quantifying the cost of local interferences caused by split