Graph-Coloring and Treescan Register Allocation Using Repairing

Q. Colombet*, B. Boissinot*, P. Brisk‡, S. Hack† and F. Rastello*

*INRIA, ENS-Lyon
‡University of California Riverside
†Saarland University

CASES, Taipei, Taiwan, October 9-14 2011
Outline

1. Decoupled Register Allocation
2. Register Constraints: Antipathy and Repairing
3. Graph Coloring based Coalescing
4. (Tree)Scan based Coalescing
5. Experiments
6. Conclusion
Outline

1. Decoupled Register Allocation
2. Register Constraints: Antipathy and Repairing
3. Graph Coloring based Coalescing
4. (Tree)Scan based Coalescing
5. Experiments
6. Conclusion
Register allocation
- Decoupled register allocation
  - Register pressure lowering (Spill)
  - Splitting/Coloring/Coalescing
- Based on Static Single Assignment (SSA)
- Graph and scan based approaches
**Context**

**Register allocation**
- Decoupled register allocation
  - Register pressure lowering (Spill)
  - Splitting/Coloring/Coalescing
- Based on Static Single Assignment (SSA)
- Graph and scan based approaches

**Register constraints handling**
- SSA splitting not enough
- Split before constrained instructions
## Context

### Register allocation
- Decoupled register allocation
  - Register pressure lowering (Spill)
  - Splitting/Coloring/Coalescing
- Based on Static Single Assignment (SSA)
- Graph and scan based approaches

### Register constraints handling
- SSA splitting not enough
- Split before constrained instructions

\[
a = \ldots
\]
\[
R_1 = \ldots
\]
\[
b = \ldots
\]
\[
\ldots = a, b
\]
\[
c = \ldots
\]
\[
\ldots = b
\]
\[
R_2 = \ldots
\]

(a) W/o split

\[
a \quad R_1
\]
\[
|\quad |
\]
\[
b \quad R_2
\]
\[
|\quad |
\]
\[
c
\]
**Register allocation**

- Decoupled register allocation
  - Register pressure lowering (Spill)
  - Splitting/Coloring/Coalescing
- Based on Static Single Assignment (SSA)
- Graph and scan based approaches

**Register constraints handling**

- SSA splitting not enough
- Split before constrained instructions

\[
\begin{align*}
  a &= \ldots \\
  R_1 &= \ldots \\
  b &= \ldots \\
  \cdots &= a, b \\
  c &= \ldots \\
  \cdots &= b \\
  R_2 &= \ldots \\
  \end{align*}
\]

(a) W/o split

\[
\begin{align*}
  a' &= \ldots \\
  (a') &= (a) \\
  R_1 &= \ldots \\
  b &= \ldots \\
  \cdots &= a', b \\
  c &= \ldots \\
  \cdots &= b \\
  (c') &= (c) \\
  R_2 &= \ldots \\
  \end{align*}
\]

(b) With split
Motivations

Impact of live range splitting

- Intermediate representation (IR) reconstruction
- Increase the size of the IR and of the interference graph (IG)
- More difficult coalescing problems
Outline

1. Decoupled Register Allocation
2. Register Constraints: Antipathy and Repairing
3. Graph Coloring based Coalescing
4. (Tree)Scan based Coalescing
5. Experiments
6. Conclusion
Register Constraints and Repairing

\[ a, b = \ldots \]
\[ \ldots = b \uparrow \{ R_1, R_3 \} \]
\[ \ldots = a, b \]

(a) Initial code

\[ R_1, R_2 = \ldots \]
\[ \ldots = R_2 \uparrow \{ R_1, R_3 \} \]
\[ \ldots = R_1, R_2 \]

(b) Allocated code

\[ R_1, R_2 = \ldots \]
\[ (R_3) = (R_2) \]
\[ \ldots = R_3 \uparrow \{ R_1, R_3 \} \]
\[ (R_2) = (R_3) \]
\[ \ldots = R_1, R_2 \]

(c) Repairing
Basis

Concept
- Tolerate violation of register constraints during assignment
- Repair afterward

Restriction
Repairing does NOT address:
- Aliasing (e.g. Register pairing)
- Live range constraints (e.g. stack, frame pointers)
Outline

1. Decoupled Register Allocation
2. Register Constraints: Antipathy and Repairing
3. Graph Coloring based Coalescing
4. (Tree)Scan based Coalescing
5. Experiments
6. Conclusion
Bias the coloring with negative affinities

\[ a, b = \ldots \]
\[ \ldots = b^{\uparrow \{R_1, R_3\}} \]
\[ \ldots = a, b \]

(a) Initial code  (b) Interference graph
Bias the coloring with negative affinities

\[ a, b = \ldots \]
\[ \ldots = b^{\uparrow\{R_1, R_3\}} \]
\[ \ldots = a, b \]

(a) Initial code

\[ R_1, R_2 = \ldots \]
\[ (R_3) = (R_2) \]
\[ (R_2) = (R_3) \]
\[ \ldots = R_3^{\uparrow\{R_1, R_3\}} \]

(b) Interference graph

\[ \ldots = R_1, R_2 \]

(c) Allocated code
Dealing with Negative Affinities

- Iterated register coalescer:

  - build graph
  - simplify
  - coalesce
  - freeze
  - potential spill
  - select
  - actual spill

- Strategies:

  a  b
  \[ \overrightarrow{-w} \]
Dealing with Negative Affinities

- **Iterated register coalescer:**

  - **Strategies:**
    - Add a node
    - Dummy node
Dealing with Negative Affinities

- Iterated register coalescer:

- Strategies:
Dealing with Negative Affinities

- Iterated register coalescer:

- Strategies:

```
  a  b
  -w

Safe to add interference?

Conservative

  a  y
  -w

Add a node

  a  ab  b
  -w

Freeze

  a  b
  -w

Dummy node

Freeze
```
The Conservative Rule

Coalescing rule for negative affinities

Let \( K \) be the number of available registers.
Let \((u, R)\) be a negative affinity between a regular vertex \( u \) and a pre-colored vertex \( R \).
\((u, R)\) can be replaced with an interference edge if, afterwards, \( u \) has at most \( K - 1 \) neighbors of degree at least \( K \).

1. \( R \) is a pre-colored node, thus it is colorable
2. Neighbors of \( u \) which degrees are less than \( K \) are simplifiable
3. \( u \) has at most \( K - 1 \) neighbors of degree at least \( K \)

Because of 2 and 3, \( u \) will be simplifiable.
\( \Rightarrow \) The new graph is also colorable.
Outline

1. Decoupled Register Allocation
2. Register Constraints: Antipathy and Repairing
3. Graph Coloring based Coalescing
4. (Tree)Scan based Coalescing
5. Experiments
6. Conclusion
**Treescan**

\[ a = \]

\[ c = \]
\[ b_1 = a \]
\[ = c \]

\[ b_2 = \]
\[ = b_2, a \]

\[ b_3 = \phi(b_1, b_2) \]
\[ \Downarrow b_3 \]

---

**Principles**
- Use Dominance-preserving order
- Allocate available color at def
- Release color at last-use

**Advantages**
- One color per variable globally
- Shuffle code by already existing $\phi$
- Fast liveness (Boissinot et al. CGO’08)
Treescan

Principles
- Use Dominance-preserving order
- Allocate available color at def
- Release color at last-use

Advantages
- One color per variable globally
- Shuffle code by already existing $\phi$
- Fast liveness (Boissinot et al. CGO’08)
Treescan

\[
a = \quad c = \quad b_1 = a \\
\quad = c \quad b_2 = \quad = b_2, a
\]

\[
b_3 = \phi(b_1, b_2) \\
\quad \downarrow \quad \equiv b_3
\]

**Principles**
- Use Dominance-preserving order
- Allocate available color at def
- Release color at last-use

**Advantages**
- One color per variable globally
- Shuffle code by already existing $\phi$
- Fast liveness (Boissinot et al. CGO’08)
Treescan

Principles
- Use Dominance-preserving order
- Allocate available color at def
- Release color at last-use

Advantages
- One color per variable globally
- Shuffle code by already existing $\phi$
- Fast liveness (Boissinot et al. CGO’08)
Treescan

Principles
- Use Dominance-preserving order
- Allocate available color at def
- Release color at last-use

Advantages
- One color per variable globally
- Shuffle code by already existing ϕ
- Fast liveness (Boissinot et al. CGO’08)
**Treescan**

\[
a =
\]

\[
c = \\
\quad b_1 = a \\
\quad = c
\]

\[
b_2 = \\
\quad = b_2, a
\]

\[
b_3 = \phi(b_1, b_2) \\
\quad = b_3
\]

---

**Principles**
- Use Dominance-preserving order
- Allocate available color at def
- Release color at last-use

**Advantages**
- One color per variable globally
- Shuffle code by already existing \(\phi\)
- Fast liveness (Boissinot et al. CGO’08)
Scan using Repairing

\[
\begin{array}{cccc}
  a & b & c & R_1 & R_2 \\
\end{array}
\]

\[
\rightarrow
\]

\[R_1 = \]

\[b = \]

\[= a, b\]

\[c = \]

\[= b\]

\[R_2 = \]

\[\]
Scan using Repairing

\[ R_2 = (c) = (R_1) \]

\[ R_1 = \]
\[ b = \]
\[ = a, b \]
\[ c = \]
\[ = b \]

\[ \rightarrow \]

\[ a \quad b \quad c \quad R_1 \quad R_2 \]
Scan using Repairing

\[ R_1 = \]

\[ b = \quad = a, b \]

\[ c = \quad = b \]

\[ R_2 = \]
Scan using Repairing

\[
\begin{align*}
R_1 \equiv (R_2) &= (a) \\
R_1 &= \begin{array}{c}
\quad b \\
\quad = a, b \\
\quad c \\
\quad = b \\
R_2 &= \end{array}
\end{align*}
\]
Scan using Repairing

\[
\begin{align*}
R_1 \equiv & \quad (a) \\
\rightarrow b = & \quad a, b \\
c = & \quad b \\
R_2 = & \\
\end{align*}
\]
Scan using Repairing

\[
\begin{align*}
R_1 & \equiv (a) \\
(R_2) & = (a) \\
\Rightarrow b & = a, b \\
c & = b \\
R_2 & = \ldots
\end{align*}
\]
Scan using Repairing

\[
\begin{align*}
R_1 &\equiv (a) \\
R_2 &\equiv (a) \\
b &\equiv a, b \\
c &\equiv b \\
R_2 &\equiv (a)
\end{align*}
\]
Scan using Repairing

\[ R_2 = (c) \]

\[ (R_2) = (a) \]

\[ R_1 \equiv (a) \]

\[ b = \]

\[ \rightarrow = a, b \]

\[ c = \]

\[ = b \]

\[ R_2 = \]

Global color is preserved

Repair on the fly
Scan using Repairing

(a) $R_2 = (a)$

$R_1 = (a)$

$b = a, b$

$c = b$

$R_2$ =
Scan using Repairing

\[ (R_2) = (a) \]

\[ R_1 \equiv (a) \]

\[ b = \]
\[ = a, b \]

\[ \rightarrow c = \]
\[ = b \]

\[ R_2 = \]
Scan using Repairing

\[(R_2) = (a)\]

\[R_1 \equiv (a)\]

\[b = \]
\[= a, b\]

\[c = \]
\[\rightarrow = b\]

\[R_2 = \]
Scan using Repairing

\[(R_2) = (a)\]

\[R_1 \equiv (a)\]

\[b = \]
\[= a, b\]

\[c = \]
\[= b\]

\[\rightarrow R_2 = \]
Scan using Repairing

\[
\begin{align*}
(R_2) &= (a) \\
R_1 &= (a) \\
b &= a, b \\
c &= b \\
(R_1) &= (c) \\
\rightarrow R_2 &= (c)
\end{align*}
\]
Scan using Repairing

\[
\begin{align*}
R_1 \equiv (R_2) &= (a) \\
b &= a, b \\
c &= b \\
R_1 \equiv (R_2) &= (c)
\end{align*}
\]
Scan using Repairing

\[ (R_2) = (a) \]
\[ R_1 \equiv (a) \]
\[ b = \]
\[ = a, b \]
\[ c = \]
\[ = b \]
\[ (R_1) = (c) \]
\[ R_2 \equiv (c) \]
\[ \rightarrow (c) = (R_1) \]
Scan using Repairing

\[
\begin{align*}
(a) & \quad R_1 \quad R_2 \\
(R_2) & = (a) \\
R_1 & \equiv (a) \\
b & = b \\
= a, b \\
c & = b \\
(R_1) & = (c) \\
R_2 & \equiv (c) \\
(c) & = (R_1)
\end{align*}
\]

- Repair on the fly
- Global color is preserved
Bias Coloring

\[ a = \phi(d, \ldots) \quad R_1 \quad R_2 \quad R_3 \]

\[ c = \ldots \]
\[ b = f(a) \]
\[ \ldots = b \]
\[ d = c \]
\[ \text{call} \]

(a) None

\[ a = \phi(d, \ldots) \quad R_1 \quad R_2 \quad R_3 \]

\[ c = \ldots \]
\[ b = f(a) \]
\[ \ldots = b \]
\[ d = c \]
\[ \text{call} \]

(b) Register hints

\[ a = \phi(d, \ldots) \quad R_1 \quad R_2 \quad R_3 \]

\[ c = \ldots \]
\[ b = f(a) \]
\[ \ldots = b \]
\[ d = c \]
\[ \text{call} \]

(c) Pre-partitioning, here \{a,d\}

\[ a = \phi(d, \ldots) \quad R_1 \quad R_2 \quad R_3 \]

\[ c = \ldots \]
\[ b = f(a) \]
\[ \ldots = b \]
\[ d = c \]
\[ \text{call} \]

(d) Caller saved flag

\[ a = \phi(d, \ldots) \quad R_1 \quad R_2 \quad R_3 \]

\[ c = \ldots \]
\[ b = f(a) \]
\[ \ldots = b \]
\[ d = c \]
\[ \text{call} \]

(e) Round robin

\[ a = \phi(d, \ldots) \quad R_1 \quad R_2 \quad R_3 \]

\[ c = \ldots \]
\[ b = f(a) \]
\[ \ldots = b \]
\[ d = c \]
\[ \text{call} \]

(f) Move related

Note: Assume \( R_1 \) and \( R_2 \) are caller saved
Outline

1. Decoupled Register Allocation
2. Register Constraints: Antipathy and Repairing
3. Graph Coloring based Coalescing
4. (Tree)Scan based Coalescing
5. Experiments
6. Conclusion
Experimental Setup

- Production compiler for STMicroelectronics ST2xx VLIW
- 32 registers, 8 of which are callee-save.
- Decoupled register allocator:
  - Spill using Hack’s heuristic
  - Coloring using iterated register coalescer (IRC) based or treescan allocator
- SPEC CINT 2000
Interference Graphs Size

- Conservative has smaller graphs than Split
- Dummy node approach not reported because it is not a realistic approach
Freeze achieves good result and is simple
Other techniques compete with Split
Treescan is 6.97x faster than IRC
i.e. allocation time drops from 38% to 5% of the compile time
Caller heuristic is the most expensive (needs liveness information)
Split is almost as expensive
H: Hints; R: Round-robin; C: Caller; M: Move related; A: Aggressive; W: Web; S: Split.

(Lower is better)

- Architecture agnostic numbers
- Pre-coalescers give good improvement (13% and 14%)
- Split improvements do not justify its overhead
Treescan - Execution Time of Generated Code

H: Hints; R: Round-robin; C: Caller; M: Move related; A: Aggressive; W: Web; S: Split. (Lower is better)

- Treescan generated code is at least as fast as IRC one
- Post-scheduling benefits from round-robin
- Pre-coalescers achieve very good improvements
Outline

1. Decoupled Register Allocation
2. Register Constraints: Antipathy and Repairing
3. Graph Coloring based Coalescing
4. (Tree)Scan based Coalescing
5. Experiments
6. Conclusion
Conclusion

- Repairing works with both graph and scan based approaches
- Code quality is comparable with and without repairing
- It does not imply the overhead of live range splitting
- Treescan is as simple and efficient as linear scan
  AND it generates code as good as IRC
- With more budget, treescan can perform better than IRC
## Runtime Details

<table>
<thead>
<tr>
<th></th>
<th>gzip</th>
<th>vpr</th>
<th>mcf</th>
<th>crafty</th>
<th>parser</th>
<th>perlbmk</th>
<th>gap</th>
<th>vortex</th>
<th>bzip2</th>
<th>twolf</th>
<th>G.Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>0.89</td>
<td>1.05</td>
<td>1.02</td>
<td>1</td>
<td>1.04</td>
<td>0.96</td>
<td>0.99</td>
<td>1.01</td>
<td>0.96</td>
<td>1</td>
<td>0.99</td>
</tr>
<tr>
<td>HC</td>
<td>0.93</td>
<td>1.06</td>
<td>0.99</td>
<td>1.01</td>
<td>1.06</td>
<td>0.97</td>
<td>0.98</td>
<td>1</td>
<td>0.97</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>HR</td>
<td>0.91</td>
<td>1.07</td>
<td>0.99</td>
<td>1.03</td>
<td>1.03</td>
<td>0.96</td>
<td>1</td>
<td>0.98</td>
<td>0.87</td>
<td>1.01</td>
<td>0.98</td>
</tr>
<tr>
<td>HCR</td>
<td>0.93</td>
<td>1.07</td>
<td>0.97</td>
<td>1.01</td>
<td>1.03</td>
<td>0.97</td>
<td>0.98</td>
<td>0.99</td>
<td>0.87</td>
<td>1.01</td>
<td>0.98</td>
</tr>
<tr>
<td>HW</td>
<td>0.87</td>
<td>1.01</td>
<td>0.99</td>
<td>0.98</td>
<td>0.97</td>
<td>0.94</td>
<td>0.97</td>
<td>0.95</td>
<td>0.95</td>
<td>0.99</td>
<td>0.96</td>
</tr>
<tr>
<td>HA</td>
<td>0.87</td>
<td>1.01</td>
<td>0.99</td>
<td>0.98</td>
<td>0.97</td>
<td>0.94</td>
<td>0.97</td>
<td>0.95</td>
<td>0.95</td>
<td>0.99</td>
<td>0.96</td>
</tr>
<tr>
<td>HAC</td>
<td>0.87</td>
<td>1</td>
<td>0.96</td>
<td>0.97</td>
<td>0.96</td>
<td>0.94</td>
<td>0.96</td>
<td>0.96</td>
<td>0.95</td>
<td>0.99</td>
<td>0.96</td>
</tr>
<tr>
<td>HAR</td>
<td>0.86</td>
<td>1.03</td>
<td>0.97</td>
<td>0.99</td>
<td>0.98</td>
<td>0.93</td>
<td>0.96</td>
<td>0.93</td>
<td>0.87</td>
<td>0.99</td>
<td>0.95</td>
</tr>
<tr>
<td>HAM</td>
<td>0.87</td>
<td>1.02</td>
<td>0.96</td>
<td>1</td>
<td>0.97</td>
<td>0.94</td>
<td>0.97</td>
<td>0.95</td>
<td>0.88</td>
<td>0.99</td>
<td>0.95</td>
</tr>
<tr>
<td>HARC</td>
<td>0.85</td>
<td>0.99</td>
<td>0.96</td>
<td>0.97</td>
<td>0.97</td>
<td>0.93</td>
<td>0.95</td>
<td>0.94</td>
<td>0.87</td>
<td>0.99</td>
<td>0.94</td>
</tr>
<tr>
<td>HARCS</td>
<td>0.86</td>
<td>0.99</td>
<td>1</td>
<td>0.97</td>
<td>0.97</td>
<td>0.95</td>
<td>0.96</td>
<td>0.95</td>
<td>0.89</td>
<td>0.99</td>
<td>0.95</td>
</tr>
<tr>
<td>HACM</td>
<td>0.87</td>
<td>0.99</td>
<td>0.93</td>
<td>0.98</td>
<td>0.96</td>
<td>0.94</td>
<td>0.96</td>
<td>0.96</td>
<td>0.87</td>
<td>0.99</td>
<td>0.94</td>
</tr>
<tr>
<td>HARC M</td>
<td>0.85</td>
<td>0.99</td>
<td>0.95</td>
<td>0.98</td>
<td>0.97</td>
<td>0.93</td>
<td>0.96</td>
<td>0.96</td>
<td>0.85</td>
<td>0.99</td>
<td>0.94</td>
</tr>
<tr>
<td>HARM</td>
<td>0.86</td>
<td>1.04</td>
<td>0.99</td>
<td>1</td>
<td>0.98</td>
<td>0.93</td>
<td>0.97</td>
<td>0.95</td>
<td>0.85</td>
<td>0.99</td>
<td>0.96</td>
</tr>
</tbody>
</table>
Freeze Bad Results

\begin{equation}
\begin{array}{c}
 a \\
 w \\
 R_1 \\
 b \\
 -2w \\
 R_2 \\
 (\text{freeze})
\end{array}
\end{equation}