Register allocation: What does the NP-completeness of Chaitin et al. really prove?

or

Revisiting register allocation: Why and how?

Alain Darte

Joint work with F. Bouchez (LIP), C. Guillon (STmicro), and F. Rastello (LIP)

CNRS
Laboratoire de l’Informatique du Parallélisme
École normale supérieure de Lyon

Montbonnot, Sept. 21, 2006, ID Laboratory
Outline

1. NP-completeness of register allocation and ambiguities
   - Classical register allocation views
   - Example: iterated register coalescing
   - Confusions and questions

2. Determining if $k$ registers are enough
   - NP-completeness proof of Chaitin et al.
   - Easy case: no critical edge, strict program, swaps
   - Where did the NP-completeness “disappear”? 

3. Coalescing problems
   - Main results
   - Example: proof for optimistic coalescing

4. Conclusion
Outline

1. NP-completeness of register allocation and ambiguities
   - Classical register allocation views
   - Example: iterated register coalescing
   - Confusions and questions

2. Determining if $k$ registers are enough
   - NP-completeness proof of Chaitin et al.
   - Easy case: no critical edge, strict program, swaps
   - Where did the NP-completeness “disappear”?

3. Coalescing problems
   - Main results
   - Example: proof for optimistic coalescing

4. Conclusion
What is register allocation?

Assign variables of a program to physical registers

- for a fixed instruction schedule;
- unlimited number of variables to place in:
  - a pool of limited resources (registers).
  - a pool of unlimited resources (memory).
- some architectural subtleties:
  - specific registers (e.g., sp, fp, r0);
  - variable affinities (e.g., auto-inc), register pairing (e.g., 64 bits operations);
  - distributed register banks, variable sizes, etc.
  - ...
What is register allocation?

Assign variables of a program to physical registers
- for a fixed instruction schedule;
- unlimited number of variables to place in:
  - a pool of limited resources (registers).
  - a pool of unlimited resources (memory).
- some architectural subtleties:
  - specific registers (e.g., sp, fp, r0);
  - variable affinities (e.g., auto-inc), register pairing (e.g., 64 bits operations);
  - distributed register banks, variable sizes, etc.
- ...

Play with colors (registers) and color changes (transfers) with register-to-register moves → coalescing, live-range splitting; insertions of stores and loads → spilling.
What do we learn at school?

“Register allocation is NP-complete because graph coloring is NP-complete.”

- variable ⇔ vertex;
- interferences between variables ⇔ edge;
- variable assignment ⇔ graph coloring.
⇒ use heuristics for register assignment (coloring), spilling (load/store insertion), and coalescing (move removal);
What do we learn at school?

“Register allocation is NP-complete because graph coloring is NP-complete.”

- variable ⇔ vertex;
- interferences between variables ⇔ edge;
- variable assignment ⇔ graph coloring.

⇒ use heuristics for register assignment (coloring), spilling (load/store insertion), and coalescing (move removal);

“But it is polynomial for a basic block.”

- live range of a variable (with unique def.) = interval;
- interference graph = interval graph

⇒ easy to color with a minimal number of colors.

Register allocation
What do we learn at school?

“Register allocation is NP-complete because graph coloring is NP-complete.”

- variable $\Leftrightarrow$ vertex;
- interferences between variables $\Leftrightarrow$ edge;
- variable assignment $\Leftrightarrow$ graph coloring.

$\Rightarrow$ use heuristics for register assignment (coloring), spilling (load/store insertion), and coalescing (move removal);

“But it is polynomial for a basic block.”

- live range of a variable (with unique def.) $=$ interval;
- interference graph $=$ interval graph

$\Rightarrow$ easy to color with a minimal number of colors.

Hum. . . All this is confusing and misleading. We’ll see why later.
Global register allocation with graph coloring:


Given: $k$ registers, interference graph, affinities (for coalescing).

- **Simplify** remove a non-move-related vertex with degree $< k$;
- **Coalesce** merge 2 move-related vertices (e.g., conservatively);
- **Freeze** give up about some moves;
- **Potential spill** remove a vertex and push it on a stack;
- **Select** pop a vertex and assign a color;
- **Actual spill** if no color is found, really insert load/store.

See power-point slides.
So, what is confusing?

Local register allocation is polynomial?

- Yes for deciding if $k$ registers are enough, by renaming variables to get unique definitions.
- But what if more registers are needed, i.e., if some spilling is necessary?
  
So, what is confusing?

Local register allocation is polynomial?
- Yes for deciding if \( k \) registers are enough, by renaming variables to get unique definitions.
- But what if more registers are needed, i.e., if some spilling is necessary?

Global register allocation is NP-complete?
- But the proof only addresses the coloring without considering register-to-register moves! So is it really hard to decide if \( k \) registers are enough? ▶ This talk
So, what is confusing?

Local register allocation is polynomial?
- Yes for deciding if \( k \) registers are enough, by renaming variables to get unique definitions.
- But what if more registers are needed, i.e., if some spilling is necessary?

Global register allocation is NP-complete?
- But the proof only addresses the coloring without considering register-to-register moves! So is it really hard to decide if \( k \) registers are enough? ☛ This talk
- What about spilling? Coalescing?
Outline

1. NP-completeness of register allocation and ambiguities
   - Classical register allocation views
   - Example: iterated register coalescing
   - Confusions and questions

2. Determining if $k$ registers are enough
   - NP-completeness proof of Chaitin et al.
   - Easy case: no critical edge, strict program, swaps
   - Where did the NP-completeness “disappear”?

3. Coalescing problems
   - Main results
   - Example: proof for optimistic coalescing

4. Conclusion
Interpretation of original proof

Chaitin et al. ⇒ NP-complete if each variable is assigned to a unique register.

Extension ⇒ if live-range splitting is allowed, remains NP-complete because of critical edges.
But proves nothing if blocks & moves can be inserted!

Switch

\[ B_{a,b} \]
\[ a = 1 \]
\[ b = 2 \]
\[ x = a + b \]

\[ B_{a,c} \]
\[ a = 3 \]
\[ c = 4 \]
\[ x = a + c \]

\[ B_{b,d} \]
\[ b = 5 \]
\[ d = 6 \]
\[ x = b + d \]

\[ B_{c,d} \]
\[ c = 7 \]
\[ d = 8 \]
\[ x = c + d \]

return \( a + x \)\n
return \( b + x \)\n
return \( c + x \)\n
return \( d + x \)
Maxlive: max. number of variables simultaneously live

Assume swaps, a strict program, edge splitting allowed

1. One needs Maxlive ≤ k, so spill to get Maxlive ≤ k.
2. Split critical edges (= add basic blocks).
3. Color each program point independently with at most Maxlive colors.
4. Use permutations to match colors (thanks to swaps).

This gives a correct assignment. . .
Maxlive: max. number of variables simultaneously live

Assume swaps, a strict program, edge splitting allowed

1. One needs Maxlive \( \leq k \), so spill to get Maxlive \( \leq k \).
2. Split critical edges (= add basic blocks).
3. Color each program point independently with at most Maxlive colors.
4. Use permutations to match colors (thanks to swaps).

This gives a correct assignment. . . but expensive in moves, even after conservative register coalescing (Appel-George).
Maxlive: max. number of variables simultaneously live

Assume swaps, a strict program, edge splitting allowed

1. One needs Maxlive $\le k$, so spill to get Maxlive $\le k$.
2. Split critical edges (= add basic blocks).
3. Color each program point independently with at most Maxlive colors.
4. Use permutations to match colors (thanks to swaps).

This gives a correct assignment. . . but expensive in moves, even after conservative register coalescing (Appel-George).

More promising approaches

- Basic block coloring (interval graph);
- SSA-like coloring = subtrees of a tree (chordal graph);
- Guided live-range/edge splitting + permutation motion.
After results by Brisk et al., Hack et al., Bouchez et al. on SSA and register allocation, Pereira and Palsberg wondered:

“Can we do polynomial-time register allocation by first transforming the program to SSA form, then doing linear-time register allocation for the SSA form, and finally doing SSA elimination while maintaining the mapping from temporaries to registers?”

They show it is NP-complete if swaps are not available.

Reduce from $k$-coloring circular-arc graph. Make sure Live = $k$ on the back edge (where SSA will split) so that a non-trivial permutation is impossible.

Note: polynomial for a fixed $k$. (See Garey, Johnson, Miller, Papadimitriou.)
After results by Brisk et al., Hack et al., Bouchez et al. on SSA and register allocation, Pereira and Palsberg wondered:

“Can we do polynomial-time register allocation by first transforming the program to SSA form, then doing linear-time register allocation for the SSA form, and finally doing SSA elimination while maintaining the mapping from temporaries to registers?”

♩ They show it is NP-complete if swaps are not available.

- Reduce from $k$-coloring circular-arc graph.
- Make sure Live = $k$ on the back edge (where SSA will split) so that a non-trivial permutation is impossible.

Note: polynomial for a fixed $k$. (See Garey, Johnson, Miller, Papadimitriou.)
Chaitin et al’s variant if swaps are not available

\[ y_d = b + x_{b,d} \]
\[ y_c = d + x_{c,d} \]
\[ y_b = d + x_{b,d} \]
\[ y_a = c + x_{a,c} \]
\[ x_c = c + d \]
\[ x_b = b + d \]
\[ x_a = a + c \]
\[ x_a, b = a + b \]
\[ x_a, c = a + c \]
\[ x_b, d = b + d \]
\[ x_c, d = c + d \]

Register pressure = 3 on all edges
Chaitin et al’s variant if swaps are not available

NP-complete if moves on entry/exit of basic blocks only, even for $k = 3$. 

$\begin{align*}
  a &= 1 \\
  b &= 2 \\
  x_{a,b} &= a + b \\
  y_a &= b + x_{a,b} \\
  x_a &= 1 \\
  y_b &= a + x_{a,b} \\
  x_b &= 5 \\
  \text{return } x_a + y_a + a \\

  a &= 3 \\
  c &= 4 \\
  x_{a,c} &= a + c \\
  y_a &= c + x_{a,c} \\
  x_a &= 2 \\
  y_c &= a + x_{a,c} \\
  x_c &= 6 \\
  \text{return } x_b + y_b + b \\

  b &= 5 \\
  d &= 6 \\
  x_{b,d} &= b + d \\
  y_b &= d + x_{b,d} \\
  x_b &= 3 \\
  y_d &= b + x_{b,d} \\
  x_d &= 7 \\
  \text{return } x_c + y_c + c \\

  c &= 7 \\
  d &= 8 \\
  x_{c,d} &= c + d \\
  y_c &= d + x_{c,d} \\
  x_c &= 4 \\
  y_d &= c + x_{c,d} \\
  x_d &= 8 \\
  \text{return } x_d + y_d + d
\end{align*}$

register pressure = 3 on all edges
If swaps are not available, what can we conclude?

NP-complete if moves on entry/exit of basic blocks only.
If swaps are not available, what can we conclude?

NP-complete if moves on entry/exit of basic blocks only.

But why not inserting moves in the middle of a block?
If swaps are not available, what can we conclude?

NP-complete if moves on entry/exit of basic blocks only.

But why not inserting moves in the middle of a block?

NP-complete if instructions can define two variables simultaneously.

Replace each pair of definitions such as $y_a = b + x_{a,b}$ and $x_a = 1$ by one instruction $(x_a, y_a) = f(b, x_{a,b})$. 
If swaps are not available, what can we conclude?

NP-complete if moves on entry/exit of basic blocks only.

But why not inserting moves in the middle of a block?

NP-complete if instructions can define two variables simultaneously.

Replace each pair of definitions such as \( y_a = b + x_{a,b} \) and \( x_a = 1 \) by one instruction \( (x_a, y_a) = f(b, x_{a,b}) \).

But, often, either swaps are available or such instructions have low register pressure (ex: function call, 64 bits load).
If swaps are not available, what can we conclude?

NP-complete if moves on entry/exit of basic blocks only.

- But why not inserting moves in the middle of a block?

NP-complete if instructions can define two variables simultaneously.
Replace each pair of definitions such as \( y_a = b + x_{a,b} \) and \( x_a = 1 \) by one instruction \((x_a, y_a) = f(b, x_{a,b})\).

- But, often, either swaps are available or such instructions have low register pressure (ex: function call, 64 bits load).

Polynomial if instructions have only one result! Greedy traversal along the control-flow graph where \( \text{Live} = k \).
If swaps are not available, what can we conclude?

NP-complete if moves on entry/exit of basic blocks only.
   ☛ But why not inserting moves in the middle of a block?

NP-complete if instructions can define two variables simultaneously.
   Replace each pair of definitions such as \( y_a = b + x_{a,b} \) and \( x_a = 1 \) by one instruction \( (x_a, y_a) = f(b, x_{a,b}) \).
   ☛ But, often, either swaps are available or such instructions have low register pressure (ex: function call, 64 bits load).

Polynomial if instructions have only one result! Greedy traversal along the control-flow graph where \( \text{Live} = k \).

So, NP-completeness did not disappear, it was simply not there! The proof of Chaitin et al. does not say anything about register allocation with live-range splitting and critical edge splitting.
Register allocation remains difficult

- When critical edges cannot be split. But...
- Because optimal spilling is (almost) always hard.
- Because optimal coalescing is, in most cases, NP-complete.

But, if moves are more suitable than loads and stores, it is in general easy to decide if some spilling is necessary or not.

**Spill test**  Chaitin (degree \( \geq k \)) \( \rightarrow \) Briggs (potential spill) \( \rightarrow \)

Appel-George (iterated) \( \rightarrow \) Biased coloring \( \rightarrow \) Optimal test
Outline

1. NP-completeness of register allocation and ambiguities
   - Classical register allocation views
   - Example: iterated register coalescing
   - Confusions and questions

2. Determining if $k$ registers are enough
   - NP-completeness proof of Chaitin et al.
   - Easy case: no critical edge, strict program, swaps
   - Where did the NP-completeness “disappear”? 

3. Coalescing problems
   - Main results
   - Example: proof for optimistic coalescing

4. Conclusion
Links between different approaches

- **Aggressive coalescing**
- **Conservative coalescing**
- **Optimistic heuristic**
- **Incremental conservative coalescing**
- **Optimistic coalescing**

**G**: Initial graph, $k$-colorable or greedy-$k$-colorable
**$G_1$**: obtained by incremental conservative coalescing, greedy-$k$-colorable
**$G_2$**: obtained by optimistic de-coalescing, greedy-$k$-colorable
**$G_3$**: optimally coalesced greedy-$k$-colorable
**$G_4$**: optimally coalesced $k$-colorable
**$G_5$**: obtained by aggressive coalescing
Main complexity results

$G$ interference graph, $G_f$ graph after coalescing.

**Aggressive coalescing** NP-complete, even if $G$ is chordal or greedy-3-colorable.

**Conservative coalescing** NP-complete even if $G$ is greedy-2-colorable, one requires $G_f$ to be chordal or greedy-3-colorable, and only affinities can be merged.

**Incremental conservative coalescing (Briggs, George)**
NP-complete if $G$ is arbitrary. Polynomial if $G$ is chordal.

**Open** if $G$ is greedy-$k$-colorable.

**Optimistic coalescing (Park & Moon) = conservative de-coalescing**
NP-complete even if $G$ is chordal and $k = 4$. 
Optimistic coalescing: from vertex cover, degree $\leq 3$
Optimistic coalescing: from vertex cover, degree $\leq 3$

![Graph example for optimistic coalescing](image)
Optimistic coalescing: from vertex cover, degree $\leq 3$
Optimistic coalescing: from vertex cover, degree $\leq 3$

Remove central node $\rightarrow$
Reduction for optimistic coalescing (Cont’d)
Outline

1. NP-completeness of register allocation and ambiguities
   - Classical register allocation views
   - Example: iterated register coalescing
   - Confusions and questions

2. Determining if $k$ registers are enough
   - NP-completeness proof of Chaitin et al.
   - Easy case: no critical edge, strict program, swaps
   - Where did the NP-completeness “disappear”?

3. Coalescing problems
   - Main results
   - Example: proof for optimistic coalescing

4. Conclusion
Conclusions and future works

Be careful  Chaitin et al. reduction from graph $k$-coloring does not really mean that “coloring” variables is hard.

Maxlive $\leq k$  is in general a good test for deciding if spilling is necessary. Even iterated register coalescing overspills.

Spilling is hard  what to spill and where is challenging.

Splitting (some) critical edges  does not seem to be a problem.

Spilling under SSA  does not seem to be a good strategy.

Coalescing is hard  in theory, even with “nice” graph structures. But good optimistic heuristics should be possible.

More experiments  need to be done for exploring this new view and tradeoffs between spilling & coalescing.
That’s all!
Any questions?
If moves can be anywhere, the proof is broken.
Failure of incremental conservative coalescing

\[ \text{greedy-3-colorable} \]

\[ \text{greedy-4-colorable} \]

\[ \text{greedy-3-colorable} \]