CS 4240: Compilers

Lecture 18: Register Allocation Review

Instructor: Vivek Sarkar
(vsarkar@gatech.edu)

March 25, 2019

ANNOUNCEMENTS & REMINDERS

- **Project 2 due by 11:59pm on Wednesday, April 3rd**
  - 15% of course grade
  - Next lecture (March 27) will be a help session for Project 2

- FINAL EXAM: Wednesday, May 1, 2:40 pm - 5:30 pm
  - 30% of course grade

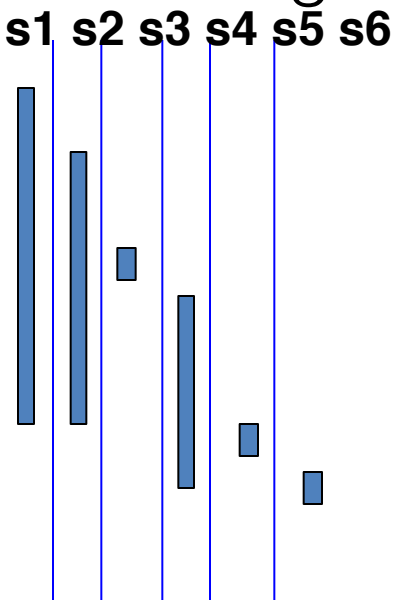
Register allocation (Lecture 12)

- Assume a 3-address code intermediate representation with an unbounded number of virtual/symbolic registers
- For each virtual register **r**, want to choose a physical machine register to store **r**'s values
- Intermediate step: determine the *live ranges* of each virtual register **r**

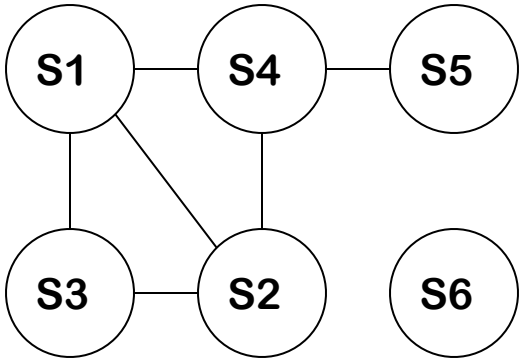
A Simple Example

```
s1 ← 2
s2 ←  4
s3 ← s1 + s2
s4 ← s3 + 1
s5 ← s1 * s2
s6 ← s4 * 2
```

Live Range



Interference



- **Live**: variable will be used before it is overwritten
- **Dead**: variable will be overwritten before it is used

A Simple Example (contd)

Intermediate Code w/ Symbolic Registers

```
s1 ← 2
s2 ← 4
s3 ← s1 + s2
s4 ← s3 + 1
s5 ← s1 * s2
s6 ← s4 * 2
```

Machine Code w/ Physical Registers

```
r1 ← 2
r2 ← 4
r3 ← r1 + r2
r3 ← r3 + 1
r1 ← r1 * r2
r2 ← r3 * 2
```

Building the Interference Graph

What is an "interference" ? (or conflict)

- Two values interfere if there exists an operation where both are simultaneously live
- If x and y interfere, they cannot occupy the same register

To compute interferences, we must know where values are "live"

The interference graph, $G_I = (N_I, E_I)$

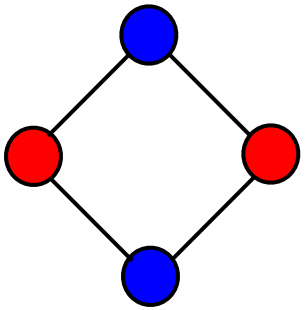
- Nodes in G_I represent values, or live ranges
- Edges in G_I represent individual interferences
  - For $x, y \in N_I$, $\langle x,y \rangle \in E_I$ iff x and y interfere
- A k-coloring of G_I can be mapped into an allocation to k registers

Graph Coloring

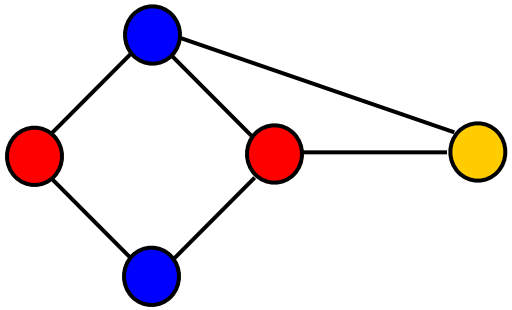
The problem

A graph G is said to be k-colorable iff the nodes can be labeled with integers 1… k so that no edge in G connects two nodes with the same label

Examples



2-colorable



3-colorable

Each color can be mapped to a distinct physical register

Computing LIVE Sets (live ranges)

A value v is live at program point p if ∃ a path from p to some use of v along which v is not re-defined

Data-flow problems are expressed as simultaneous equations

$$\text{LIVEOUT}(b) = \bigcup_{s \in succ(b)} \text{LIVEIN}(s)$$

$$\text{LIVEIN}(b) = \text{UEVAR}(b) \cup (\text{LIVEOUT}(b) - \text{VARKILL}(b))$$

where

UEVAR(b) is the set of names used in block b before being defined in b (Upwards Exposed Variables)

VARKILL(b) is the set of variables assigned in b

Solve the equations using a fixed-point iterative scheme

Computing LIVE Sets

The compiler can solve these equations with an iterative algorithm

```
WorkList ← { all blocks }
while ( WorkList ≠ Ø)
    remove a block b from WorkList
    Compute LIVEOUT(b)
    Compute LIVEIN(b)
    if LIVEIN(b) changed
        then add pred (b) to WorkList
```

The Worklist Iterative Algorithm

Why does this work?

- LIVEOUT, LIVEIN $\subseteq 2^{Names}$
- UEVAR, VARKILL are constants for b
- Equations are monotone
- Finite # of additions to sets

$\Rightarrow$ will reach a fixed point !

Speed of convergence depends on the order in which blocks are "removed" & their sets recomputed

The world's quickest introduction to data-flow analysis !

Register Allocation

1. Determine live ranges for each symbolic register
2. Determine overlapping ranges (**interference**)
3. Compute the benefit of keeping each love range in a register (**spill cost**)
4. Try to assign each live range to a machine register (**allocation**).
   If needed, **spill** or **split** live range
5. Generate code, including spills

Extended Linear Scan: an Alternate Foundation for Global Register Allocation

Compiler Construction (CC) conference, 2007

Vivek Sarkar (IBM T.J. Watson Research Center)
Rajkishore Barik (IBM India Research Laboratory)

Summary

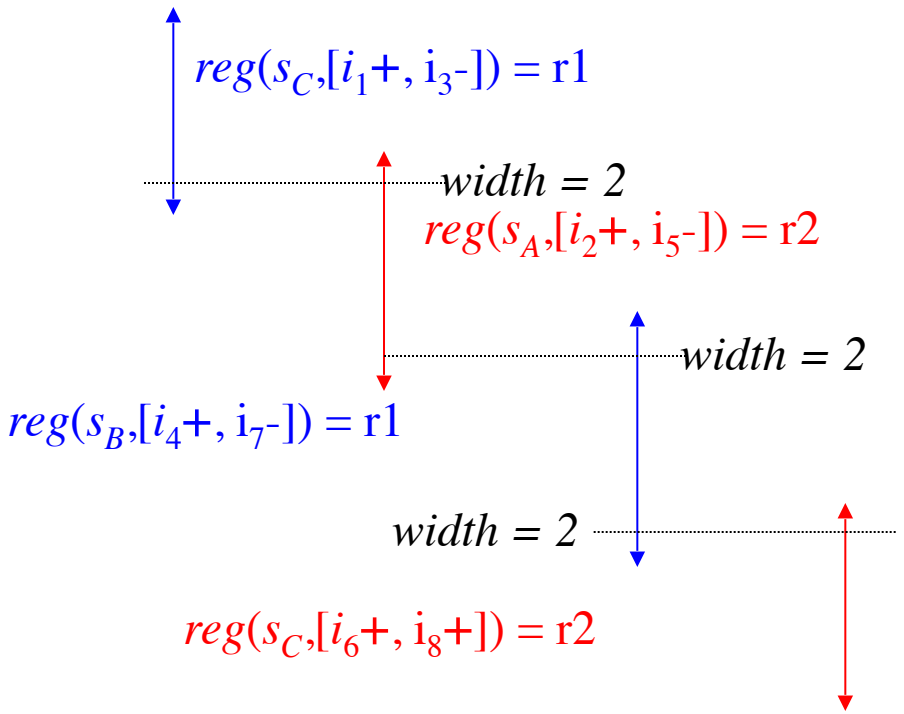
- Register allocation continues in importance since first FORTRAN compiler
- Motivation: theoretical and practical limitations of Graph Coloring
- Two variants of Register Allocation problem addressed in this paper:
  - Spill Free Register Allocation (SFRA) – decision problem
  - Register Allocation with Total Spills (RATS) – optimization problem
- Approach includes possibility of inserting register moves to change register assignments at different program points (regMoves=true/false)
- Two new Extended Linear Scan (ELS) register allocation algorithms
  - ELS_0 for SFRA problem (only for regMoves=true)
  - ELS_1 for RATS problem (two versions based on regMoves=true/false)
- Comparison of ELS_1 (regMoves=true) with Graph Coloring for RATS problem on eight SPECint2000 benchmarks
  - 15x – 68x speedups in *compile-time* relative to Graph Coloring (compile-time measured on largest functions in the eight benchmarks)
  - Up to 5.8% speedup (average = 2.3%) in *run-time* relative to GC (run-time measured for entire benchmark)
  - No coalescing or live range splitting heuristics implemented for this study

Spill Free Register Allocation

- Given a set of symbolic registers, $\boldsymbol{R}$, and k physical registers, determine if it is possible to assign each symbolic register s to a physical register, $reg(s,P)$ at each program point P where s is live.
  - If so, report the register assignments, including any register-to-register copy statements that need to be inserted.
  - If not, report that no feasible solution exists.
- Two program points are defined for each instruction i
  - i- = point at which input operands are read
  - i+ = point at which output operands are written

Example of a Spill Free Register Allocation with Register Copies (k=2)

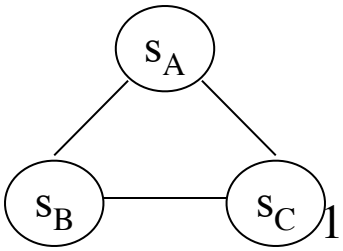
i_1: $s_C := \ldots$
/* Start of loop */
i_2: $s_A := \ldots$
i_3: $\ldots := s_C \; op \; \ldots$
i_4: $s_B := \ldots$
i_5: $\ldots := s_A \; op \; \ldots$
i_6: $s_C := \ldots$
i_7: $\ldots := s_B \; op \; \ldots$
i_8: **if** $s_C <= 0$ **goto** i_{10}
i_9: **goto** i_2
/* End of loop */



r1 := r2, register copy statement inserted between i_8 and i_9

Limitations of Graph Coloring Approach

- Graph Coloring is a more limited problem than Spill Free Register Allocation
  - Finding a *k*-coloring of an Interference Graph will lead to a feasible solution to the SFRA problem, but the converse is not true
- Non-linear space complexity for Interference Graph
  - $O(R^2)$ worst case, $O(R^{1.5})$ observed in practice
- Graph coloring is NP-hard (without a constant performance bound), but SFRA with regMoves=true can be solved optimally in linear time
- Interference Graph for previous example



Overview of ELS_0 Extended Linear Scan Algorithm for SFRA

- **Steps performed**
  - Compute count[P] = # intervals live at each program point P that is an interval end point
  - If any count[P] is > k then *return* false
  - Assign registers to intervals
  - Insert register moves as needed on each control flow edge
  - *Return* true
- **Complexity:** linear in size of IR and number of intervals
- **Correctness:** ELS_0 computes an exact solution to the SFRA decision problem

Space Savings for ELS_0 relative to Graph Coloring

Function	$\|S\|$	$\|IG\|$	$\|\mathscr{I}\|$	SCF
164.gzip.build_tree	161	2301	261	11.3%
175.vpr.try_route	254	2380	445	18.7%
181.mcf.sort_basket	138	949	226	22.7%
186.crafty.InputMove	122	1004	219	21.8%
197.parser.list_links	352	9090	414	4.5%
254.gap.SyFgets	547	7661	922	12.0%
256.bzip2.sendMTFValues	256	2426	430	17.7%
300.twolf.closepins	227	5105	503	9.8%

Table 1. Compile-time overheads for functions with the largest interference graphs in SPECint2000 benchmarks. $|S|$ = # symbolic registers, $|IG|$ = # nodes and edges in Interference Graph , $|\mathscr{I}|$ = # intervals in interval set, Space Compression Factor (SCF) = $|\mathscr{I}|/|IG|$.

Register Allocation with Total Spills

- Given a set of symbolic registers, $\boldsymbol{R}$, k physical registers, and estimated execution frequency *freq*[P] for each program point P, compute the following:
  1. *spilled*(s), which indicates if s is to be spilled, and
  2. for each symbolic register s with *spilled*(s) = *false*, a register assignment, *reg*(s, P) at each point P where s is live.
- Two versions of the RATS problem:
  - *regMoves* = *false*. Optimization goal is to minimize frequency-weighted *spill cost*.
  - *regMoves* = *true*. Optimization goal is to minimize combined frequency-weighted *spill cost* and *register moves* (using architecture-specific relative weightages)
- RATS optimization problem is NP-hard for both *regMoves* = *false* and *true* cases

Overview of ELS_1 Heuristic for RATS

- Steps performed for regMoves = true case:
  - Compute count[P] = # intervals live at each program point P that is an interval end point
  - Select point P with count[P] > k in decreasing order of freq[P]
    - Push symbolic register live at P with smallest spill cost on stack
    - Repeat until count[P] <= k for all program points
  - Resurrection: “unspill” symbolic registers on stack that no longer need to be spilled
  - Register assignment as in ELS0
    - Assign registers to remaining intervals (guaranteed to be spill free)
    - Insert register moves as needed on each control flow edge
- Modification for regMoves = false
  - “All or nothing” heuristic for register assignment avoids copies
  - Additional spills may need to be inserted
- Complexity: linear in size of IR and number of intervals

Summary

- Register allocation continues in importance since first FORTRAN compiler
- Motivation: theoretical and practical limitations of Graph Coloring
- Two variants of Register Allocation problem addressed in this paper:
  - Spill Free Register Allocation (SFRA) – decision problem
  - Register Allocation with Total Spills (RATS) – optimization problem
- Approach includes possibility of inserting register moves to change register assignments at different program points (regMoves=true/false)
- Two new Extended Linear Scan (ELS) register allocation algorithms
  - ELS_0 for SFRA problem (only for regMoves=true)
  - ELS_1 for RATS problem (two versions based on regMoves=true/false)
- Comparison of ELS_1 (regMoves=true) with Graph Coloring for RATS problem on eight SPECint2000 benchmarks
  - 15x – 68x speedups in *compile-time* relative to Graph Coloring (compile-time measured on largest functions in the eight benchmarks)
  - Up to 5.8% speedup (average = 2.3%) in *run-time* relative to GC (run-time measured for entire benchmark)
  - No coalescing or live range splitting heuristics implemented for this study