Code Generation is a Constraint Problem

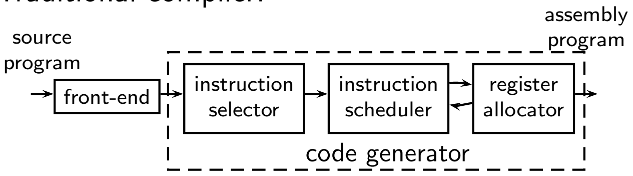
Roberto Castañeda Lozano – SICS
Mats Carlsson – SICS
Frej Drejhammar – SICS
Christian Schulte – KTH, SICS

SweConsNet'12

Outline

Compilers, Code Generation, Register Allocation

- Traditional compiler:



- Key code generation task: register allocation
  - assignment
  - spilling
  - coalescing
  - packing
  - local vs. global

Problems in Traditional Code Generation

- All tasks are interdependent
  - staging is sub-optimal
- Each task is NP-hard: solved by heuristic algorithms
  - fast but sub-optimal and complex

    "Lord knows how GCC does register allocation right now". (Anonymous, GCC Wiki)

Can We Do Better?

1. Potentially optimal code:
   - task integration
   - combinatorial optimization
2. Simpler design: separation of modeling and solving

. . . sounds like something for CP

- previous CP approaches:
  - scheduling only (Malik *et al.*, 2008)
  - integrated code generation
    - scheduling, *assignment* (Kuchcinski, 2003)
    - selection, scheduling, allocation (Leupers *et al.*, 1997)

  → limitation: local (cannot handle control-flow)

Our Approach

- Constraint model unifying
  - global register allocation with all its essential aspects
  - instruction scheduling
- Based on a novel program representation
- Robust code generator based on a problem decomposition
- Current code quality: on par with LLVM (state of the art)

Running Example: Factorial

```
int factorial(int n) {
  int f = 1;
  while (n > 0) {
    f = f * n;
    n--;
  }
  return f;
}
```

Low-Level Program Representation

- After instruction selection
- Control-flow graph:
  - vertices: blocks of instructions without control-flow
  - arcs: jumps and branches
- Instruction: defined temps, operation, used temps

$$t_7 \leftarrow \texttt{mul}\ t_6, t_5$$

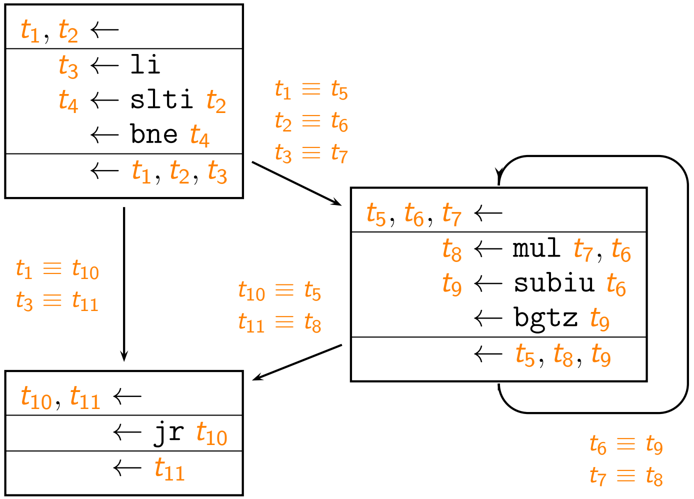
- A temp is live while it might still be used
- Two temps interfere if they are live simultaneously
  - non-interfering temps can share registers

Linear Static Single Assignment (LSSA)

- How to model interference of *global* temps?
    - start from Static Single Assignment (SSA)
    - decompose global temps into multiple local temps
- New invariant: in LSSA all temps are local

  $\rightarrow$ simple interference model
- Input form to the register allocation model

Linear Static Single Assignment: Factorial

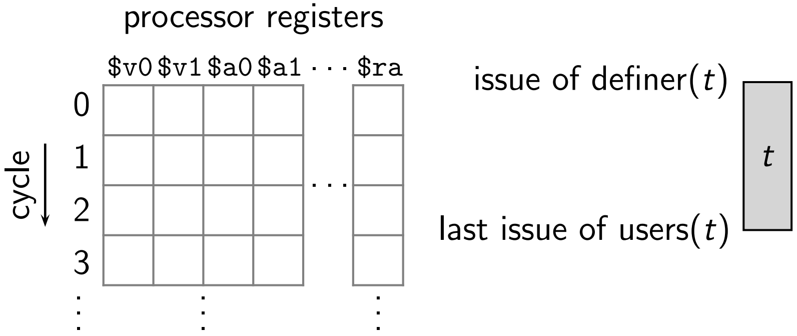


Register Assignment

to which register do we assign each temporary t?

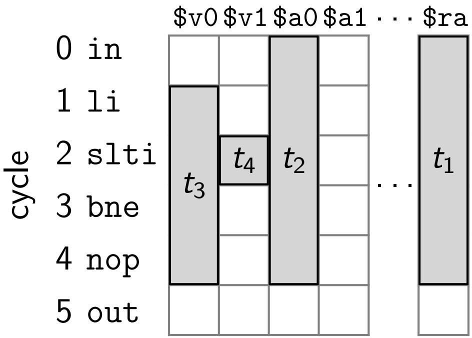
r_t?

Register Assignment: Geometric View



- Interfering temps cannot share registers: disjoint2
- Global: congruent temps share the same register

Register Assignment: Example



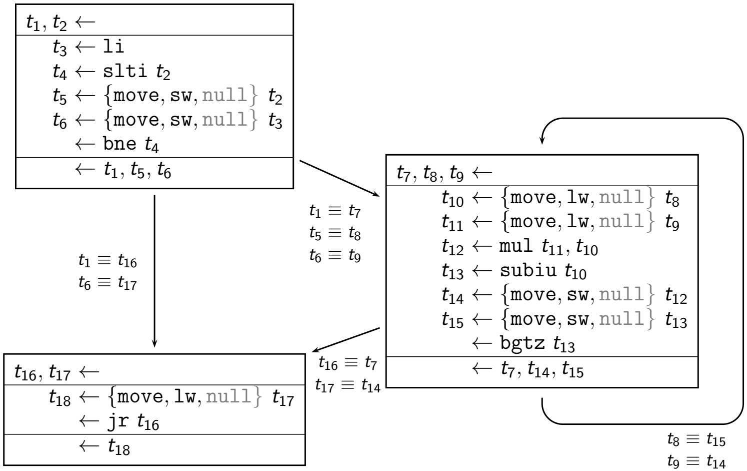
$$r_{t_1} \mapsto \texttt{\$ra},\ r_{t_2} \mapsto \texttt{\$a0},\ r_{t_3} \mapsto \texttt{\$v0},\ r_{t_4} \mapsto \texttt{\$v1}$$

Spilling and Coalescing: Copies

- Spilling requires copying temps from/to memory
  - introduce copy instructions
- Copy operations:
  - register to memory (`sw`)
  - memory to register (`lw`)
  - register to register (`move`)
  - nothing (`null`)
- Copies can be implemented by different operations:

  $t_6 \leftarrow \{\texttt{sw}, \texttt{move}, \texttt{null}\}\ t_3$

Spilling and Coalescing: Factorial with Copies


$t_1, t_2 \leftarrow$

$t_3 \leftarrow \texttt{li}$

$t_4 \leftarrow \texttt{slti}\ t_2$

$t_5 \leftarrow \{\texttt{move}, \texttt{sw}, \texttt{null}\}\ t_2$

$t_6 \leftarrow \{\texttt{move}, \texttt{sw}, \texttt{null}\}\ t_3$

$\leftarrow \texttt{bne}\ t_4$

$\leftarrow t_1, t_5, t_6$

$t_7, t_8, t_9 \leftarrow$

$t_{10} \leftarrow \{\texttt{move}, \texttt{lw}, \texttt{null}\}\ t_8$

$t_{11} \leftarrow \{\texttt{move}, \texttt{lw}, \texttt{null}\}\ t_9$

$t_{12} \leftarrow \texttt{mul}\ t_{11}, t_{10}$

$t_{13} \leftarrow \texttt{subiu}\ t_{10}$

$t_{14} \leftarrow \{\texttt{move}, \texttt{sw}, \texttt{null}\}\ t_{12}$

$t_{15} \leftarrow \{\texttt{move}, \texttt{sw}, \texttt{null}\}\ t_{13}$

$\leftarrow \texttt{bgtz}\ t_{13}$

$\leftarrow t_7, t_{14}, t_{15}$

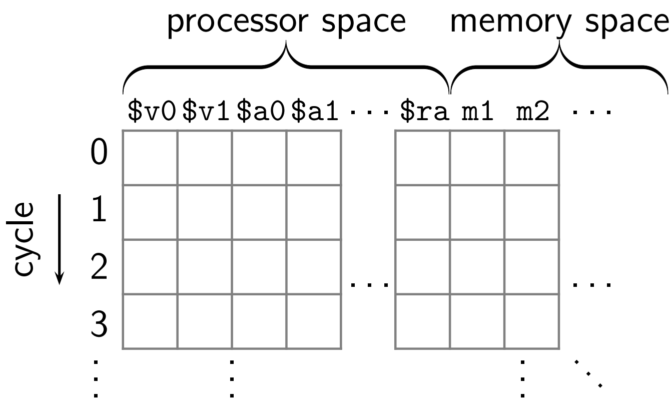
$t_{16}, t_{17} \leftarrow$

$t_{18} \leftarrow \{\texttt{move}, \texttt{lw}, \texttt{null}\}\ t_{17}$

$\leftarrow \texttt{jr}\ t_{16}$

$\leftarrow t_{18}$

Spilling and Coalescing: Unified Register File



Spilling and Coalescing: Operation Variables

which operation implements each copy instruction *i*?

Spilling and Coalescing: Constraints

- Operation selection:

  The register spaces to which copy temps are allocated are determined by the selected operation.

  if $t_6 \leftarrow t_3$ is implemented by `sw`:

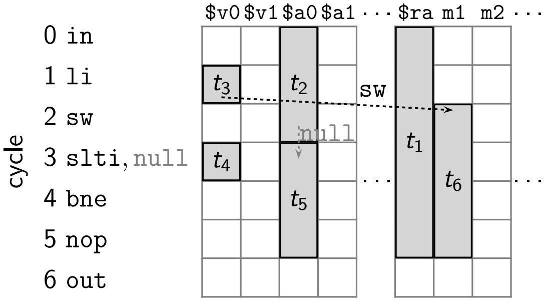
  - t_3 must be assigned to processor registers
  - t_6 must be assigned to memory registers

- Coalescing:

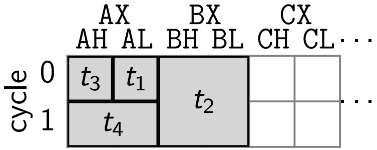
  A copy is implemented by `null` *iff its temps are assigned to the same register.*

Spilling and Coalescing: Example

- Block with two copies:
  - $t_5 \leftarrow$ `null` t_2
  - $t_6 \leftarrow$ `sw` t_3



Register Packing



- No model changes required

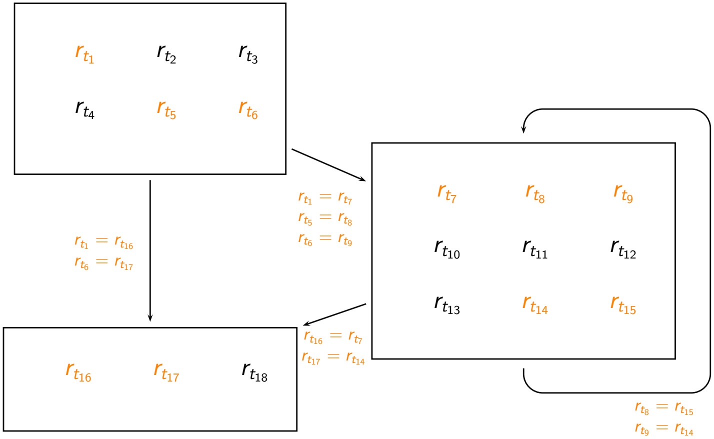
Instruction Scheduling

in which cycle is each instruction *i* issued?

- Classic scheduling model:
  - dependencies among instructions
  - resource constraints
- Connection to register allocation: live ranges

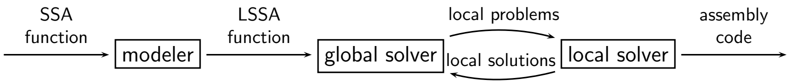
LSSA Decomposition

- Key: congruences are the only link between blocks



r_{t_1} r_{t_2} r_{t_3}
r_{t_4} r_{t_5} r_{t_6}
r_{t_7} r_{t_8} r_{t_9}
$r_{t_{10}}$ $r_{t_{11}}$ $r_{t_{12}}$
$r_{t_{13}}$ $r_{t_{14}}$ $r_{t_{15}}$
$r_{t_{16}}$ $r_{t_{17}}$ $r_{t_{18}}$
$r_{t_1} = r_{t_7}$
$r_{t_5} = r_{t_8}$
$r_{t_6} = r_{t_9}$
$r_{t_1} = r_{t_{16}}$
$r_{t_6} = r_{t_{17}}$
$r_{t_{16}} = r_{t_7}$
$r_{t_{17}} = r_{t_{14}}$
$r_{t_8} = r_{t_{15}}$
$r_{t_9} = r_{t_{14}}$

Solving Strategy

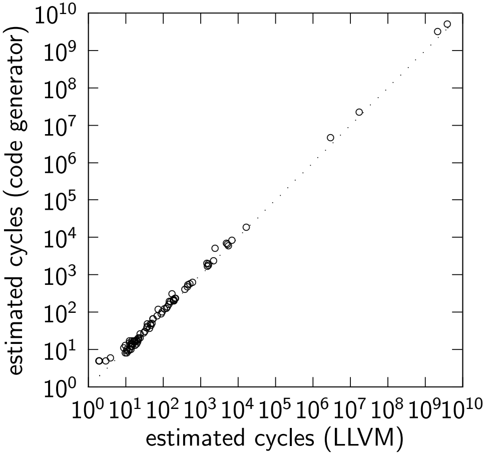


1. Convert to LSSA form
2. Solve satisfaction problem for global register assignment
3. Solve local optimization problems for each block
   - minimize makespan
4. Combine local solutions to form a global one
5. Iterate until optimality (or time-out)

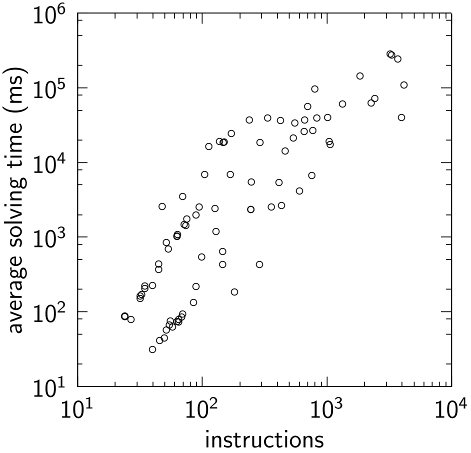
Experiment Setup

- 86 functions from `bzip2` (SPECint 2006 suite)
- Selected MIPS32 instructions with LLVM 3.0
- Implementation with Gecode 3.7.3
- Sequential search on standard desktop machine

Quality of Generated Code vs. LLVM



Solving Time



Conclusion

1. Model unifying:
   - essential aspects of register allocation
   - instruction scheduling

   → state-of-the-art code quality
2. Problem decomposition

   → robust generator for thousands of instructions

- Key: tailored problem representation (LSSA form)
- Lots of future work:
  - search heuristics, symmetry breaking . . .
  - integration with instruction selection
  - evaluate for other processors and benchmarks