

# Constraint-based Register Allocation and Instruction Scheduling

**Roberto Castañeda Lozano** – SICS

Mats Carlsson – SICS

Frej Drejhammar – SICS

Christian Schulte – KTH, SICS



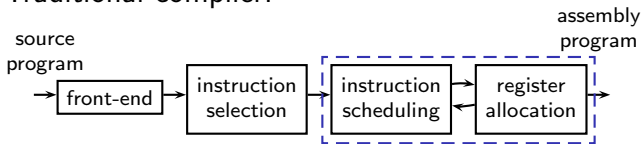
CP 2012

# Outline

- 1 Introduction
- 2 Program Representation
- 3 Constraint Model
- 4 Decomposition
- 5 Evaluation
- 6 Conclusion

# Problems in Traditional Register Allocation

- Traditional compiler:



- Problems:

- Interdependencies: staging is suboptimal
- NP-hardness: sub-optimal, complex heuristic algorithms

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

# Can We Do Better?

- 1 Potentially optimal code: integration, optimization
  - 2 Simplicity, flexibility: separation of modeling and solving  
... this 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)

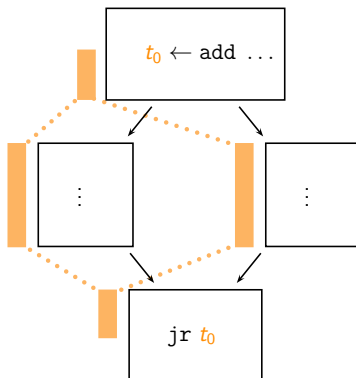
# Our Approach

- Constraint model that unifies
  - global register allocation with all essential aspects
    - register assignment, spilling, coalescing, ...
  - 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)

- 1 Introduction
- 2 Program Representation**
- 3 Constraint Model
- 4 Decomposition
- 5 Evaluation
- 6 Conclusion

# Liveness and Interference

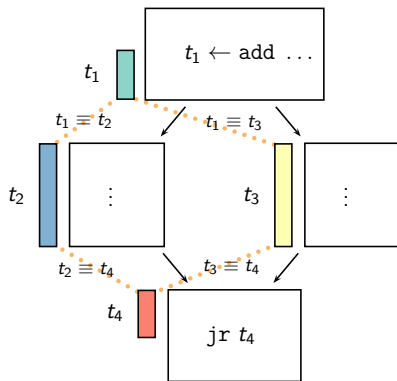
- A temp is live while it might still be used:



- Two temps interfere if they are live simultaneously
  - non-interfering temps can share registers

# Linear Static Single Assignment Form (LSSA)

- $t_0$  is *global*: live in multiple blocks
- How to model interference of global temps?
- LSSA: decompose global temps into multiple local temps



- Invariant: all temps are local  $\rightarrow$  simple interference model



- 1 Introduction
- 2 Program Representation
- 3 Constraint Model**
- 4 Decomposition
- 5 Evaluation
- 6 Conclusion

# Register Assignment

**to which register do we assign each temporary?**

# Register Assignment as Rectangle Packing

## Register Assignment

temp live ranges

temp size

interfering temps cannot share registers

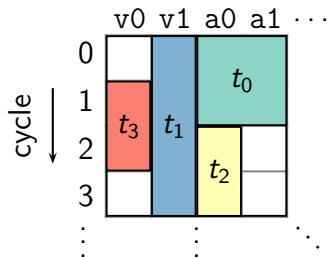
## Rectangle Packing

rectangles

rectangle width

rectangles cannot overlap

→ based on (Pereira *et al.*, 2008)



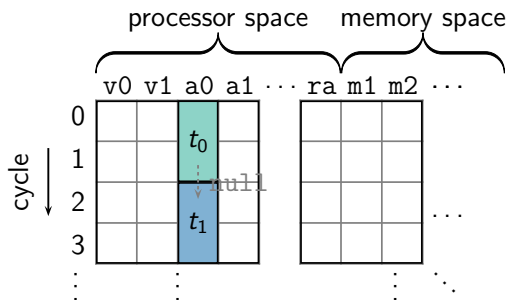
# Spilling and Coalescing

- Spilling: saving a temp in memory
- Requires copying temps from/to memory
- Introduce optional copy instructions:

$$t_1 \leftarrow \{\text{null}, \text{sw}, \text{move}\} t_0$$

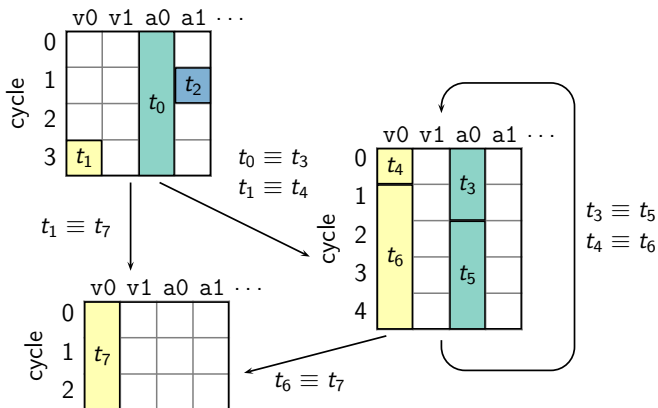
**which operation implements each copy?**

- if a copy is inactive (`null`), its temps are coalesced



# Global Register Allocation

- Link between blocks: congruences
- Congruent temps must be assigned the same register:



# Instruction Scheduling

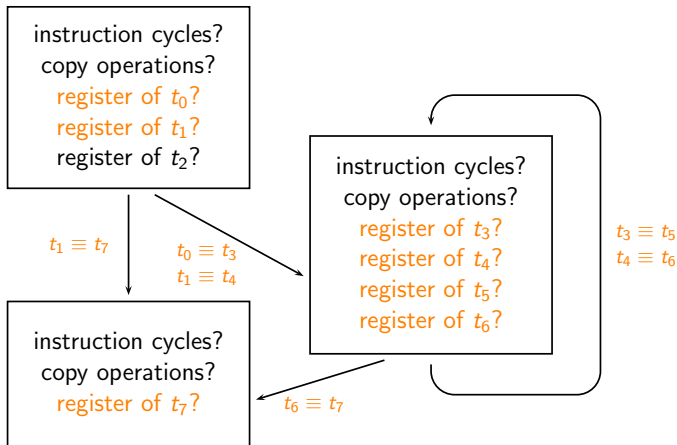
**in which cycle is each instruction issued?**

- Connection to register allocation: live ranges
- Classic scheduling model with:
  - precedences
  - resource constraints

- 1 Introduction
- 2 Program Representation
- 3 Constraint Model
- 4 Decomposition**
- 5 Evaluation
- 6 Conclusion

# LSSA Decomposition

- Only link between blocks: congruences



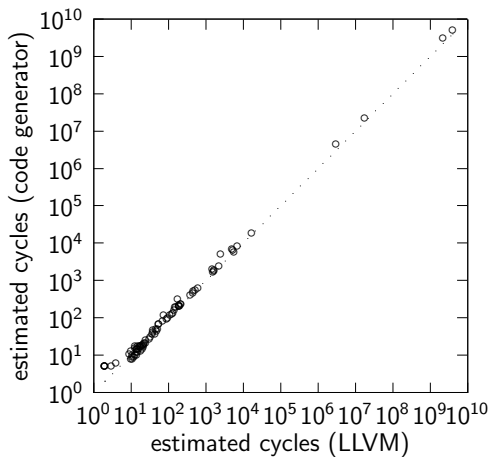


- 1 Introduction
- 2 Program Representation
- 3 Constraint Model
- 4 Decomposition
- 5 Evaluation**
- 6 Conclusion

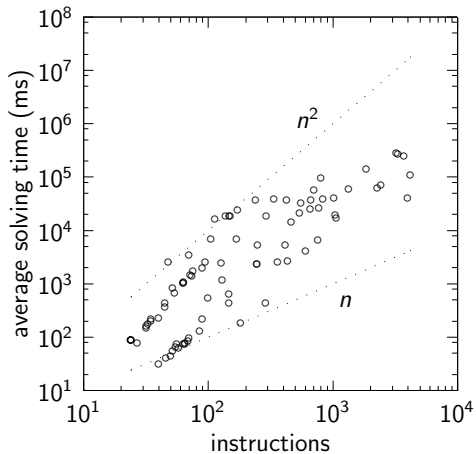
# Experiment Setup

- 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



- 1 Introduction
- 2 Program Representation
- 3 Constraint Model
- 4 Decomposition
- 5 Evaluation
- 6 Conclusion**

# Conclusion

- 1 Model that unifies:
  - all 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, implied constraints ...
  - integration with instruction selection
  - evaluate for other processors and benchmarks