CSE 583 Literature Presentation [***Group 20***]: Combinatorial register allocation and instruction scheduling

Daiwen Zhang, Zhongwei Xu

December 1, 2025

University of Michigan, Ann Arbor

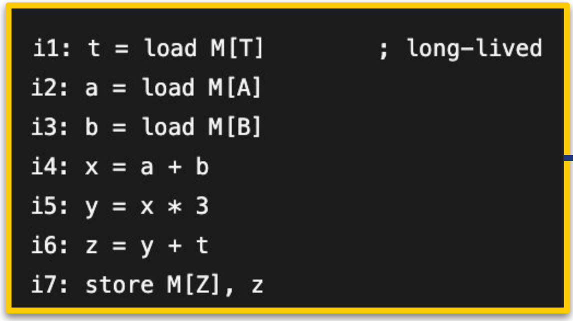
Overview

- Lozano, Roberto Castañeda, et al. "**Combinatorial register allocation and instruction scheduling.**" *ACM Transactions on Programming Languages and Systems (TOPLAS)* 41.3 (2019): 1-53.
    - Published in 2019, based on previous work since 2012.
- A **Combinatorial Optimization** approach to **Register Allocation** and **Instruction Scheduling**.
- Unison: An implementation in Haskell & C++, integrated with the LLVM toolchain.
    - https://github.com/unison-code/unison

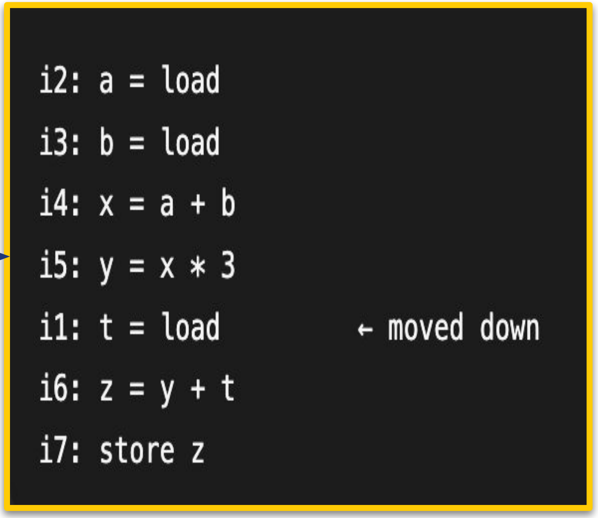
Backgrounds

- **Register Allocation**: Mapping temporaries to registers or memory.
- **Instruction Scheduling**: Reorder Instructions to improve total latency or throughput.
- Both problems are difficult, and are **mutually interdependent**.
- Today’s compilers solve each problem in isolation with heuristics algorithms.
- This reduces compilation time but precludes optimal solutions.

Previous Study



```
i1: t = load M[T]          ; long-lived
i2: a = load M[A]
i3: b = load M[B]
i4: x = a + b
i5: y = x * 3
i6: z = y + t
i7: store M[Z], z
```



```
i2: a = load
i3: b = load
i4: x = a + b
i5: y = x * 3
i1: t = load          ← moved down
i6: z = y + t
i7: store z
```

Hardware constraints:

- Only **2 physical registers** (R1, R2)
- 1 load unit, 1 ALU slot

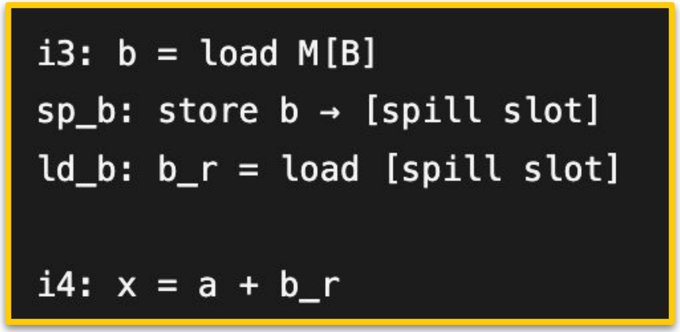
Previous Study (IS first && RA first)

RA first: Pass uses **the original program order** to compute liveness:

- Before i4 the live set is {t, a, b}.
- Peak live = **3**, but the machine provides only **2** registers.

IS First: A traditional scheduler often hoists loads early to “hide latency”:

- At i4, the live set is still {t, a, b}.
- Peak live = **3** > registers available (2) → RA must spill again.



```
i3: b = load M[B]
sp_b: store b → [spill slot]
ld_b: b_r = load [spill slot]

i4: x = a + b_r
```

Integrated Approach

The first combinatorial approach to register allocation and instruction scheduling that is **complete** (as state-of-the-art compilers), **scales up** to medium-sized problems, and generate **executable codes**.

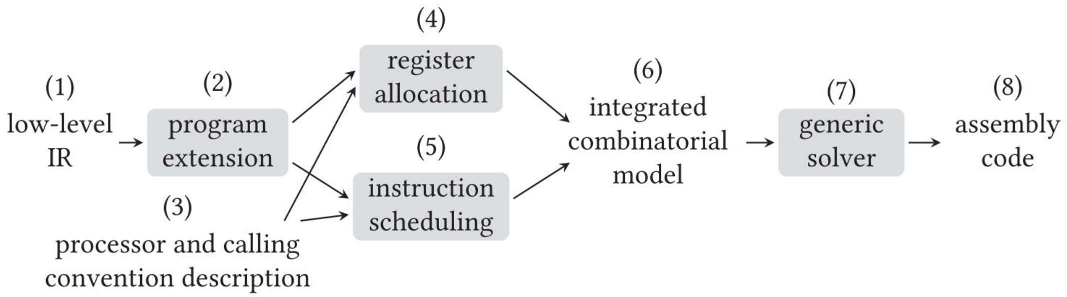

(1) low-level IR
(2) program extension
(3) processor and calling convention description
(4) register allocation
(5) instruction scheduling
(6) integrated combinatorial model
(7) generic solver
(8) assembly code

Fig. 2. Our approach to combinatorial register allocation and instruction scheduling.

Source: Fig. 2 from the paper presented

What is Optimization Modeling?

01	Decisions	• Variables
02	Collection of rules	• Constraints/Restrictions on the decision variables
03	A goal/metric to evaluate decisions	• An objective function over the decision variables

Problem Description

Mathematical Formulation

Optimization Solver

What is Combinatorial Optimization?

Boolean SAT

$$(a \vee b \vee \neg c) \wedge (\neg b) \wedge \cdots$$

- Binary Variables
- Conjunctive Normal Form (CNF)

- NP-complete
- Conflict-driven clause learning (CDCL)

Mixed-Integer Linear Programming (MILP)

$$\begin{aligned} \min_{x} \quad & c^T x \\ s.t. \quad & Ax \geq b; \\ & x \in \mathbb{R}^n,\ x_j \in \mathbb{Z}, \forall j \in J. \end{aligned}$$

- Continuous/discrete variables
- Linear inequalities
- Linear objective functions

- NP-complete (decision problem)
- Branch-and-Cut based on LP relaxation

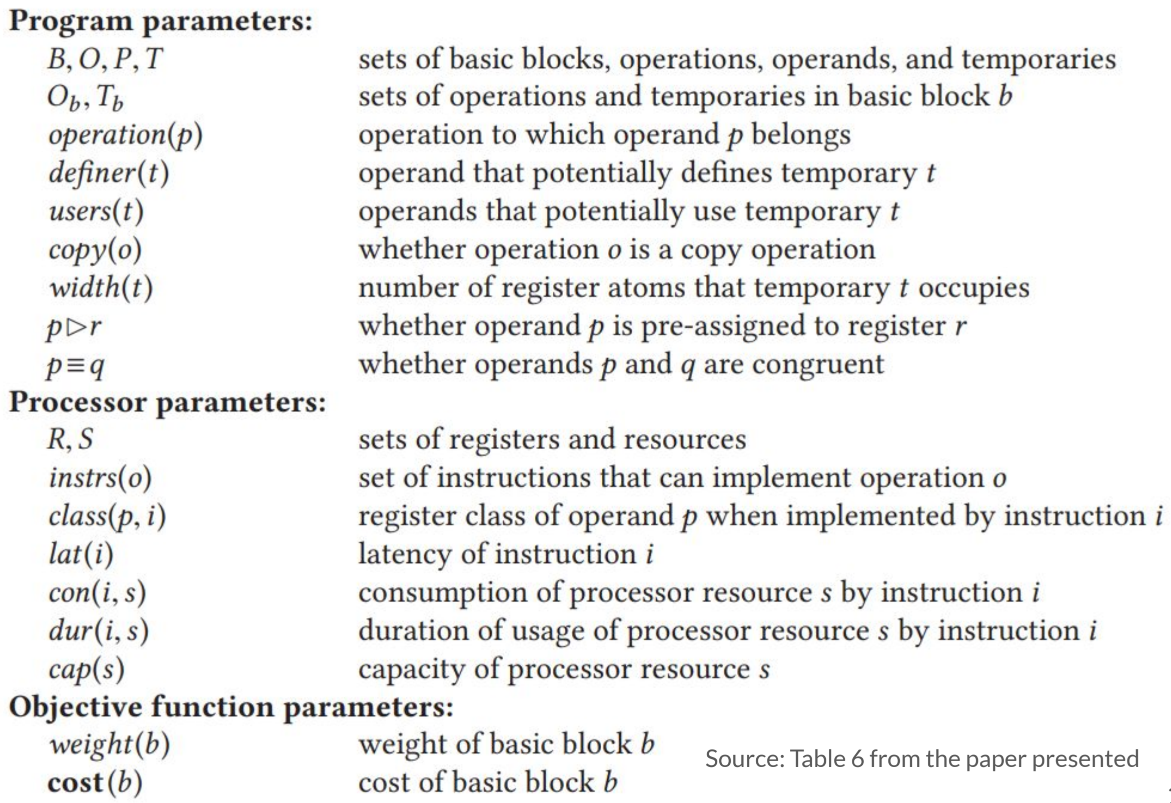
Constraint Programming

$$\texttt{all-diff}([x_1, x_2, \ldots, x_n])$$

- Discrete/binary variables
- Logical/set constraints

- Many are NP-complete
- Various searching and propagation methods

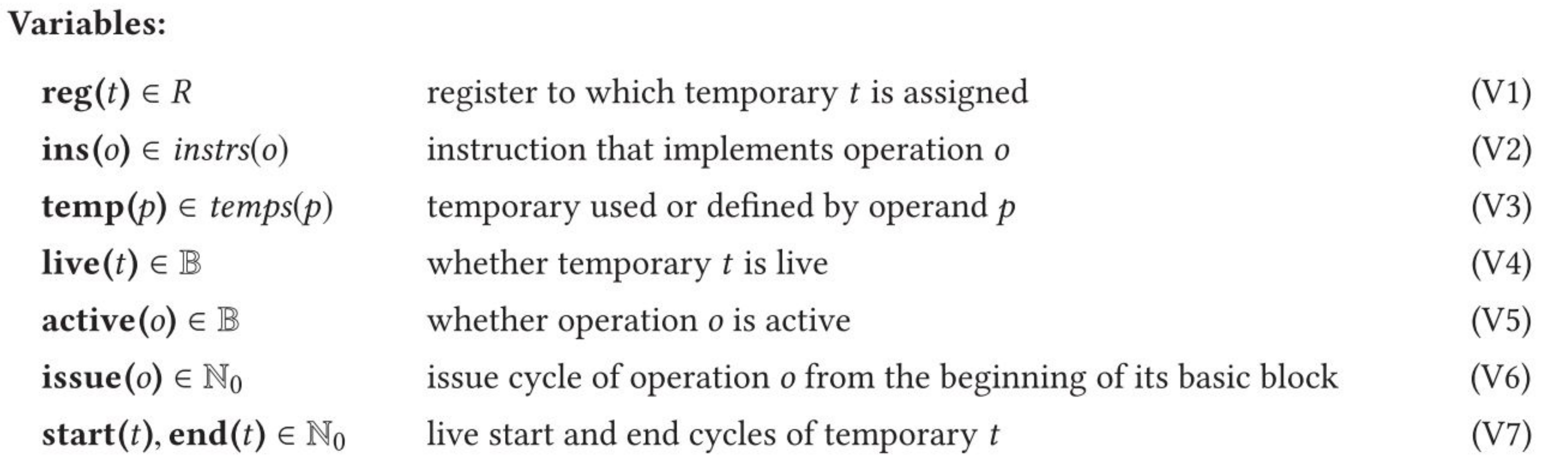
Formulation



Program parameters:	
B, O, P, T	sets of basic blocks, operations, operands, and temporaries
O_b, T_b	sets of operations and temporaries in basic block b
$operation(p)$	operation to which operand p belongs
$definer(t)$	operand that potentially defines temporary t
$users(t)$	operands that potentially use temporary t
$copy(o)$	whether operation o is a copy operation
$width(t)$	number of register atoms that temporary t occupies
$p \triangleright r$	whether operand p is pre-assigned to register r
$p \equiv q$	whether operands p and q are congruent
Processor parameters:	
R, S	sets of registers and resources
$instrs(o)$	set of instructions that can implement operation o
$class(p, i)$	register class of operand p when implemented by instruction i
$lat(i)$	latency of instruction i
$con(i, s)$	consumption of processor resource s by instruction i
$dur(i, s)$	duration of usage of processor resource s by instruction i
$cap(s)$	capacity of processor resource s
Objective function parameters:	
$weight(b)$	weight of basic block b
$\mathbf{cost}(b)$	cost of basic block b

Source: Table 6 from the paper presented

Decision Variables

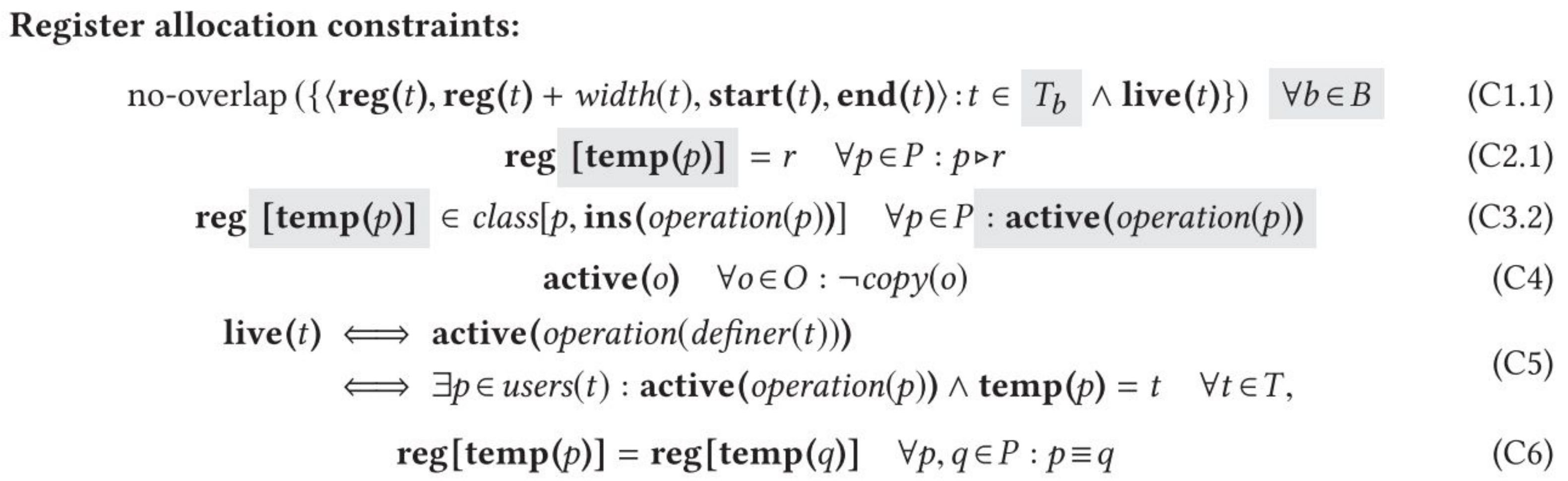


Variables:

$\mathbf{reg}(t) \in R$	register to which temporary t is assigned	(V1)
$\mathbf{ins}(o) \in \mathit{instrs}(o)$	instruction that implements operation o	(V2)
$\mathbf{temp}(p) \in \mathit{temps}(p)$	temporary used or defined by operand p	(V3)
$\mathbf{live}(t) \in \mathbb{B}$	whether temporary t is live	(V4)
$\mathbf{active}(o) \in \mathbb{B}$	whether operation o is active	(V5)
$\mathbf{issue}(o) \in \mathbb{N}_0$	issue cycle of operation o from the beginning of its basic block	(V6)
$\mathbf{start}(t), \mathbf{end}(t) \in \mathbb{N}_0$	live start and end cycles of temporary t	(V7)

Source: Table 7 from the paper presented

Constraints: Register Allocation



Register allocation constraints:

$$\text{no-overlap}\left(\{\langle \mathbf{reg}(t), \mathbf{reg}(t) + width(t), \mathbf{start}(t), \mathbf{end}(t)\rangle : t \in T_b \wedge \mathbf{live}(t)\}\right) \quad \forall b \in B \tag{C1.1}$$

$$\mathbf{reg}\,[\mathbf{temp}(p)] = r \quad \forall p \in P : p \triangleright r \tag{C2.1}$$

$$\mathbf{reg}\,[\mathbf{temp}(p)] \in class[p, \mathbf{ins}(operation(p))] \quad \forall p \in P : \mathbf{active}(operation(p)) \tag{C3.2}$$

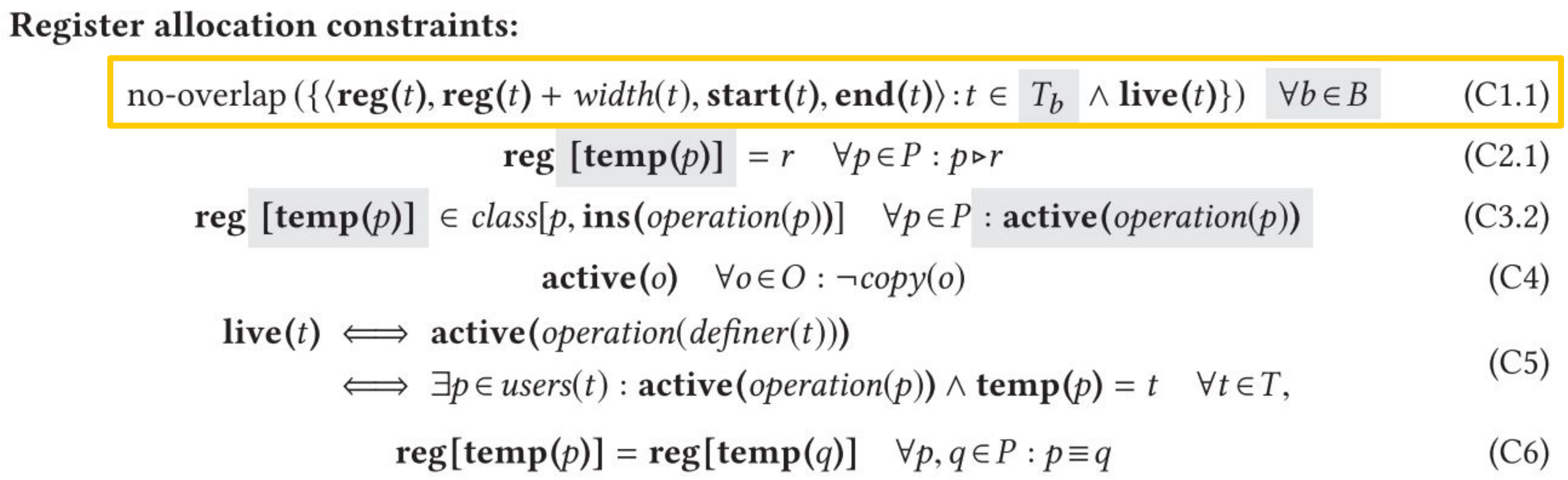
$$\mathbf{active}(o) \quad \forall o \in O : \neg copy(o) \tag{C4}$$

$$\begin{aligned} \mathbf{live}(t) &\iff \mathbf{active}(operation(definer(t))) \\ &\iff \exists p \in users(t) : \mathbf{active}(operation(p)) \wedge \mathbf{temp}(p) = t \quad \forall t \in T, \end{aligned} \tag{C5}$$

$$\mathbf{reg}[\mathbf{temp}(p)] = \mathbf{reg}[\mathbf{temp}(q)] \quad \forall p, q \in P : p \equiv q \tag{C6}$$

Source: Table 7 from the paper presented

Constraints: Register Allocation



Register allocation constraints:

$$\text{no-overlap}\left(\{\langle \mathbf{reg}(t), \mathbf{reg}(t) + \mathit{width}(t), \mathbf{start}(t), \mathbf{end}(t)\rangle : t \in T_b \land \mathbf{live}(t)\}\right) \quad \forall b \in B \tag{C1.1}$$

$$\mathbf{reg}\,[\mathbf{temp}(p)] = r \quad \forall p \in P : p \triangleright r \tag{C2.1}$$

$$\mathbf{reg}\,[\mathbf{temp}(p)] \in \mathit{class}[p, \mathbf{ins}(\mathit{operation}(p))] \quad \forall p \in P : \mathbf{active}(\mathit{operation}(p)) \tag{C3.2}$$

$$\mathbf{active}(o) \quad \forall o \in O : \neg \mathit{copy}(o) \tag{C4}$$

$$\begin{aligned} \mathbf{live}(t) &\iff \mathbf{active}(\mathit{operation}(\mathit{definer}(t))) \\ &\iff \exists p \in \mathit{users}(t) : \mathbf{active}(\mathit{operation}(p)) \land \mathbf{temp}(p) = t \quad \forall t \in T, \end{aligned} \tag{C5}$$

$$\mathbf{reg}[\mathbf{temp}(p)] = \mathbf{reg}[\mathbf{temp}(q)] \quad \forall p, q \in P : p \equiv q \tag{C6}$$

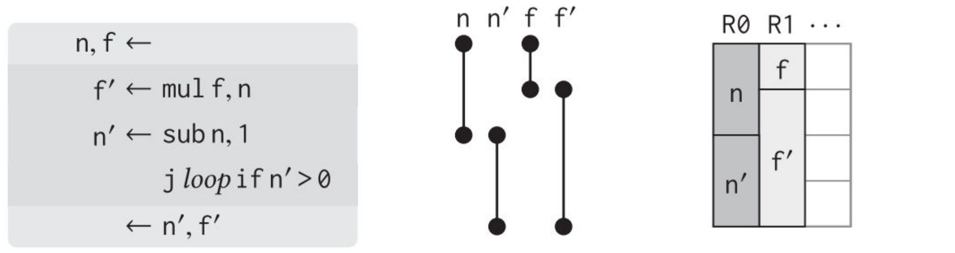
Source: Table 7 from the paper presented

Constraints: Register Allocation

$$\text{no-overlap}\left(\{\langle \mathbf{reg}(t), \mathbf{reg}(t) + \mathit{width}(t), \mathbf{start}(t), \mathbf{end}(t)\rangle : t \in T_b \land \mathbf{live}(t)\}\right) \quad \forall b \in B \qquad \text{(C1.1)}$$

- no-overlap $(\{\langle \mathbf{l}(i), \mathbf{r}(i), \mathbf{t}(i), \mathbf{b}(i)\rangle : i \in (1, n)\})$ (also known as *diffn*)

$$\mathbf{r}(i) \leq \mathbf{l}(j) \lor \mathbf{l}(i) \geq \mathbf{r}(j) \lor \mathbf{b}(i) \leq \mathbf{t}(j) \lor \mathbf{t}(i) \geq \mathbf{b}(j) \quad \forall i, j \in (1, n) : i \neq j.$$



```
    n, f ←
      f′ ← mul f, n
      n′ ← sub n, 1
              j loop if n′ > 0
          ← n′, f′
```

(a) Program representation. (b) Live ranges. (c) Register assignment.

Fig. 4. Local register assignment for the *loop* basic block from Example 2.

Source: Fig. 4, Sec. 3 from the paper presented

Constraints: Instruction Scheduling

Instruction scheduling constraints:

$$\begin{aligned}&\mathbf{issue}(\mathit{operation}(q)) \geq \mathbf{issue}(\mathit{operation}(p)) + \mathit{lat}\,[\mathbf{ins}(\mathit{operation}(p))]\\&\quad\forall t \in T, \forall p \in \{\mathit{definer}(t)\}, \forall q \in \mathit{users}(t) : \mathbf{active}(\mathit{operation}(q)) \land \mathbf{temp}(q) = t\ .\end{aligned} \tag{C7.1}$$

$$\begin{aligned}&\text{cumulative}\,(\{\langle \mathbf{issue}(o), \mathit{dur}\,[\mathbf{ins}(o), s], \mathit{con}\,[\mathbf{ins}(o), s]\rangle : o \in O_b \land \mathbf{active}(o)\}, \mathit{cap}(s))\\&\quad\forall b \in B, \forall s \in S.\end{aligned} \tag{C8.1}$$

Source: Table 7 from the paper presented

Constraints: Integration

Integration constraints:

$$\mathbf{start}(t) = \mathbf{issue}(operation(definer(t))) \quad \forall t \in T : \mathbf{live}(t) \tag{C9}$$

$$\mathbf{end}(t) = \max_{p \in users(t) : \mathbf{temp}(p) = t} \mathbf{issue}(operation(p)) \quad \forall t \in T : \mathbf{live}(t) \tag{C10}$$

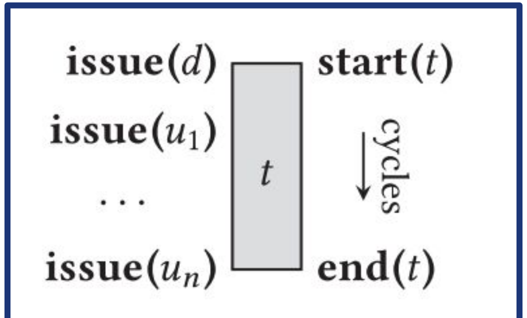


Fig. 14. Live range of t.

Source: Table 7, Fig. 14 from the paper presented

Objective Functions

Objective function:

$$minimize \sum_{b \in B} weight(b) \times \mathbf{cost}(b) \qquad ((5))$$

Optimizing Speed:

$$weight(b) = freq(b); \qquad \mathbf{cost}(b) = \mathbf{issue}(exit(b)) - 1 \quad \forall b \in B. \qquad (6)$$

Optimizing Code Size:

$$weight(b) = 1; \qquad \mathbf{cost}(b) = \sum_{o \in O_b : \mathbf{active}(o)} size[\mathbf{ins}(o)] \quad \forall b \in B. \qquad (7)$$

Source: Table 7, Sec. 8 from the paper presented

Experiments

- Open-source implementation in Haskell: Unison
- Run two algorithms in parallel:
    - Decomposition-based algorithm with strategies to exploit problem structures, based on Gecode with C++;
    - Off-the-shelf solver based on Chuffed with MiniZinc.
- Evaluation Setup:
    - Processors: Hexagon V4, ARM1156T2F-S, MIPS32
    - Benchmark: MediaBench, SPEC CPU2006
    - Baseline: LLVM 3.8

Actual Speedup

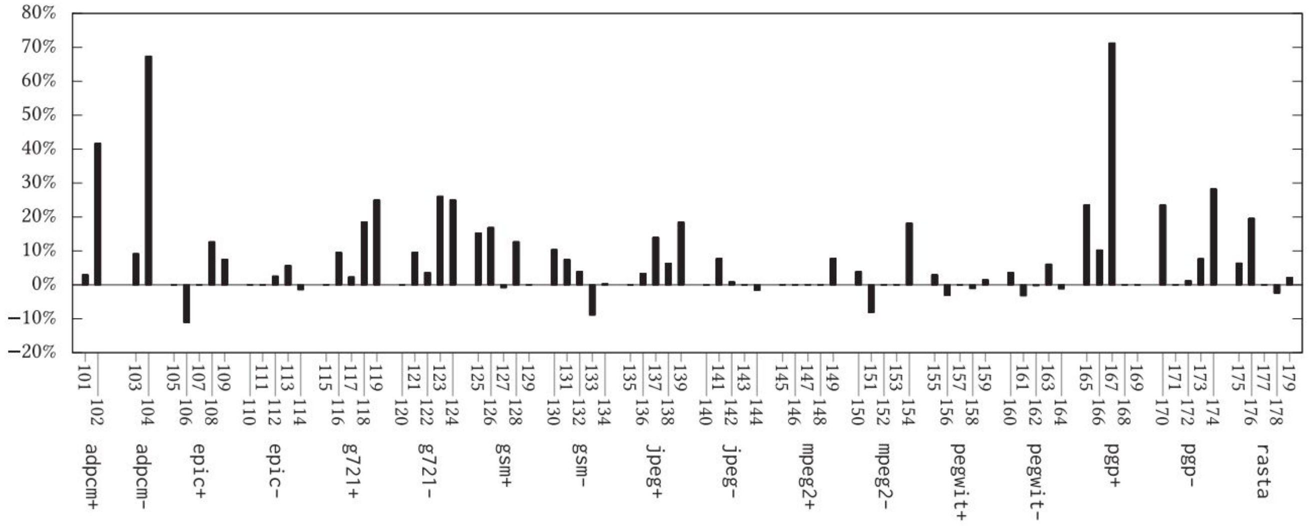


Fig. 23. Actual function speedup over LLVM grouped by application and mode.

Source: Fig. 23 from the paper presented

Actual Speedup

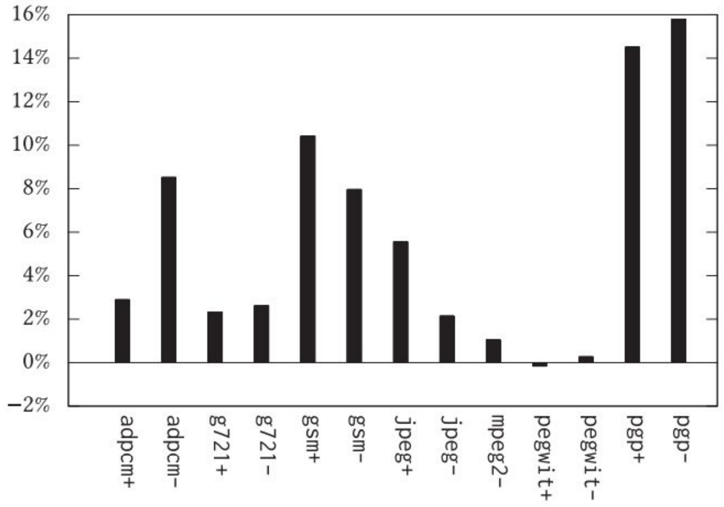


Fig. 24. Actual application speedup over LLVM.

Source: Fig. 24 from the paper presented

Discussions

1. An integrated approach that covers many aspects of Register Allocation and Instruction Scheduling;
2. A solid open-source implementation that integrates with the LLVM toolchain;
3. No Instruction Selection and Global Instruction Scheduling;

4. Rely on CP solving approaches, did not fully utilize modern MIP solvers;
5. Potential improvements on the formulation.

Discussions

1. An integrated approach that covers many aspects of Register Allocation and Instruction Scheduling;
2. A solid open-source implementation that integrates with the LLVM toolchain;
3. No Instruction Selection and Global Instruction Scheduling;

4. Rely on CP solving approaches, did not fully utilize modern MIP solvers;
5. Potential improvements on the formulation.

Thank you!