聊聊我最近读的编译器后端的论文

中科院软件所PLCT实验室 史宁宁
（知乎：小乖他爹）
2022年1月23日

目录

- 基本情况
- 寄存器分配
- 指令选择

目录

缘起-写作需要

目前，已经完成了两本编译器领域书籍的书稿，《华为方舟编译器之美》已经出版并且加印一次，《Android Runtime源码解析》将于2022年上半年上市。并且，计划在2022年带领团队完成一本有关OpenJDK for RISC-V的书稿。

在写作的过程中，发现在相关理论解释的时候，展开度不够，旁征博引的内容偏少，不能给读者一个更加宽广的知识面，不能举重若轻的谈笑间指点江山。

自己有希望在将来的书籍写作中能有所改善，能让读者更加轻松快乐的学到更多东西。同时，还希望将来能写一本理论结合实践（代码）的编译器书籍，既要包含基础的理论知识及其简要发展历史，还要有工业界普遍的实现，同时还要不局限于某个编译器/解释器。这就需要大量的理论知识做基础。

缘起-关注点转换

这些年陆陆续续接触了不少工业级别的编译器/解释器：LLVM/Clang、Tenon、方舟编译器、ART、OpenJDK，很多时候自己了解和熟悉的理论方面内容是跟着代码实现在走的，对于理论的发展历史、最初解决问题的方式以及该领域的整体情况，都掌握的不够多，缺乏了解。

随着知识网络的拓宽，补齐自己的知识网络，为之前没有关注的内容补齐基本的内容，也就成了一个急需完成的事情。

同时，关注点一直在工业化的代码层面，也容易让自己的思维被目前这几种固定模式所局限，逐渐复杂化。忘了理论问题的出发点和走过的历程。

阅读原则

- 不盲目追最新的论文内容
- 经典的综述论文
- 领域内最早的论文
- 领域发展过程中引发转折点的论文
- 介绍领域发展历程的论文

论文清单

- 1. A Survey on Register Allocation, Fernando Magno Quintao Pereira. 2008
- 2. A SURVEY OF REGISTER ALLOCATION TECHNIQUES, Jonathan Protzenko. 2009
- 3. Survey on Instruction Selection --An Extensive and Modern Literature Review, GABRIEL S. HJORT. 2013
- 4. Fifty years of peephole optimization, Pinaki Chakraborty. 2015
- 5. Peephole optimization, W. M. McKeeman. 1965

编译器后端

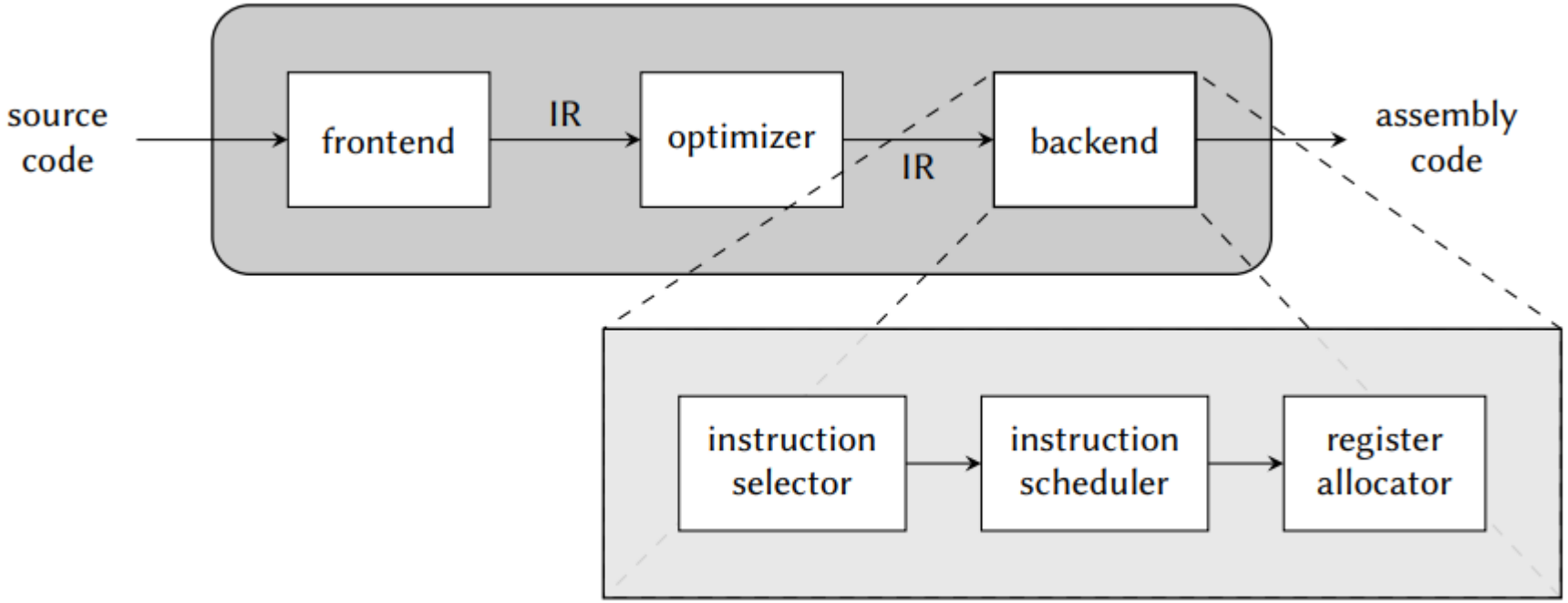


Figure 1.1: Overview of a common compiler infrastructure.

From:< Survey on Instruction Selection >P2

目录

背景

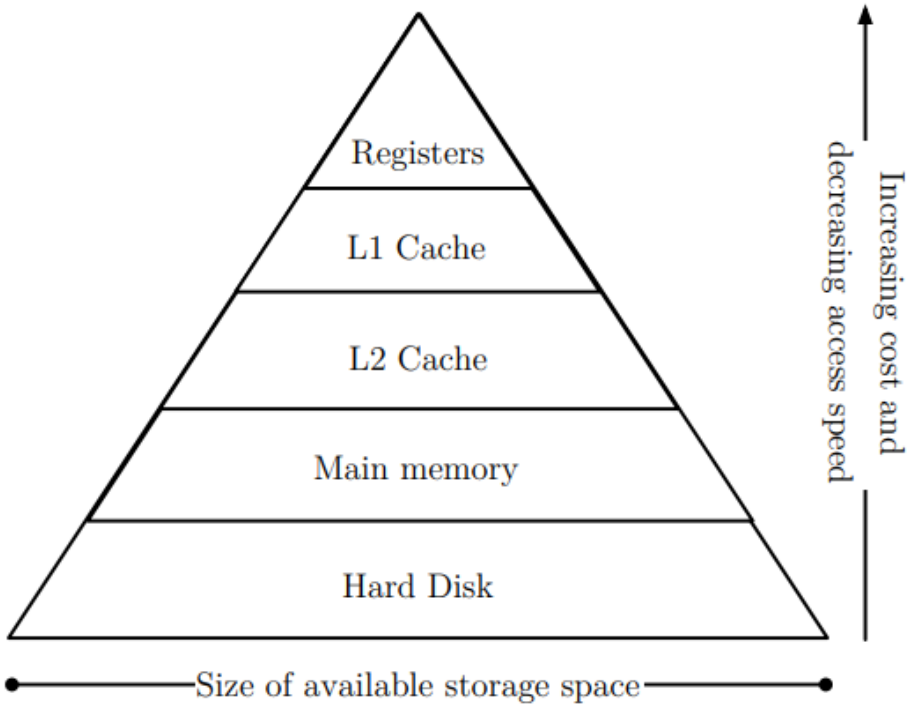


Figure 1: Memory hierarchy in a typical computer architecture.

From: A Survey on Register Allocation P2

常见问题

- Spilling（溢出）
- Coalescing（合并）

常见的方法

- Graph Coloring
- Linear Scan
- Integer Linear Programming
- Partitioned Quadratic Programming(Partitioned Boolean Quadratic Problem,　PBQP)
- Multi-Flow of Commodities(MFC)
- SSA based register allocation

Graph Coloring-Chaitin

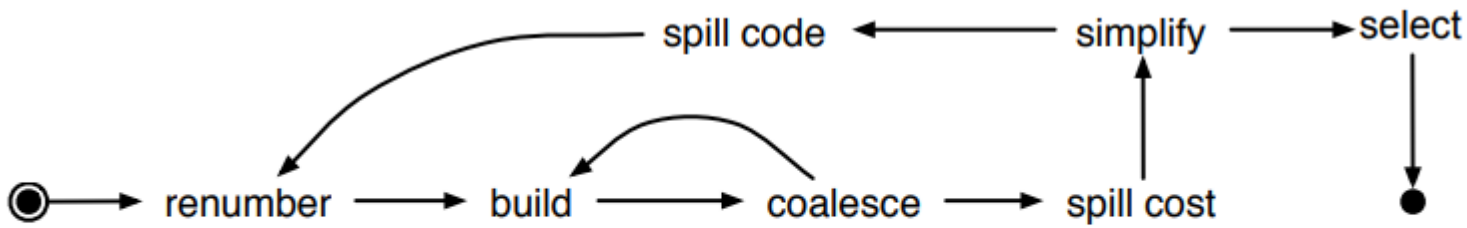


Figure 6: Chaitin *et al.*'s iterative graph coloring based register allocator. This Figure was taken from [11].

From:< A Survey on Register Allocation >P6

Graph Coloring-Briggs

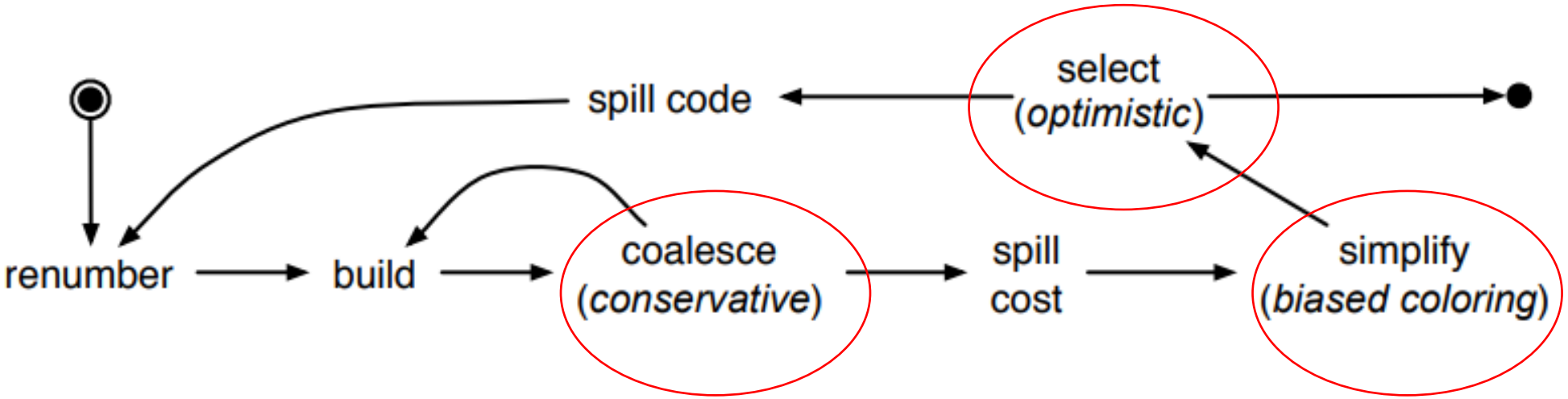


Figure 8: Briggs *et al.* graph coloring based register allocator.

比之前的Chaitin的算法，采用了更为保守的合并（coalesce）策略，只有当合并不会伤害到干涉图的着色能力时候，才会进行合并。同时，还进行了biased coloring。

图来自:< A Survey on Register Allocation >P7

Graph Coloring-Iterated Register Coalescing

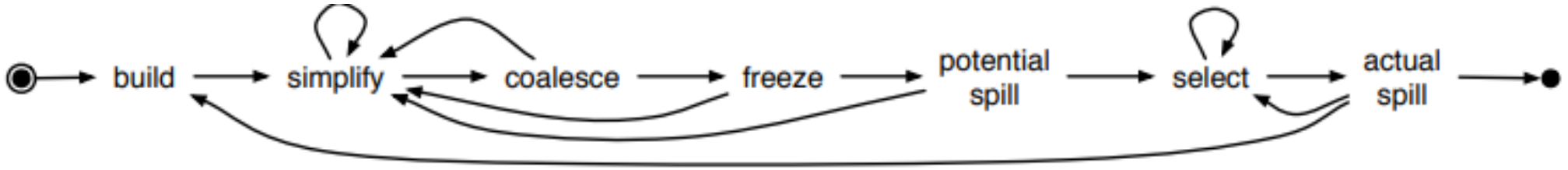


Figure 9: Iterated Register Coalescing. This Figure was taken from [3].

目前应用最广的图着色算法。
增量保守策略，只有保持图的着色能力不变时候才会去移除一个特别的移动指令。

图来自:< A Survey on Register Allocation >P8

Linear Scan

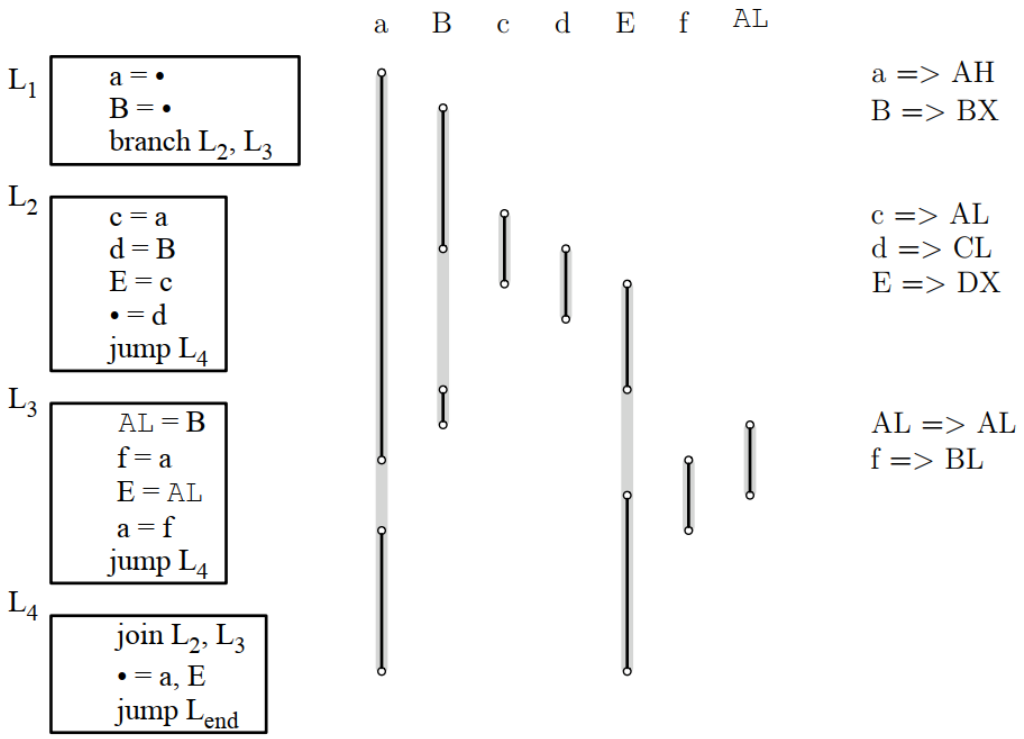


Figure 10: Linear Scan register allocation.

The main appeal of linear scan is the allocation speed. This fact makes linear scan an attractive option to Just in Time (JIT) compilers, like Java HotSpot and LLVM.

From:< A Survey on Register Allocation >P8-9

Graph Coloring VS Linear Scan

- Linear Scan is undoubtedly much faster at compile-time than graph-coloring based approaches.
- At runtime, linear scan gives worse results than graph coloring.
- Linear scan suffers from many defaults: always slower than iterated register coalescing, it doesn't even try to coalesce variables.
- Moreover, most programs are only compiled very infrequently: the goal of a program is to be run, after all.

From: <A SURVEY OF REGISTER ALLOCATION TECHNIQUES> P5

SSA based register allocation

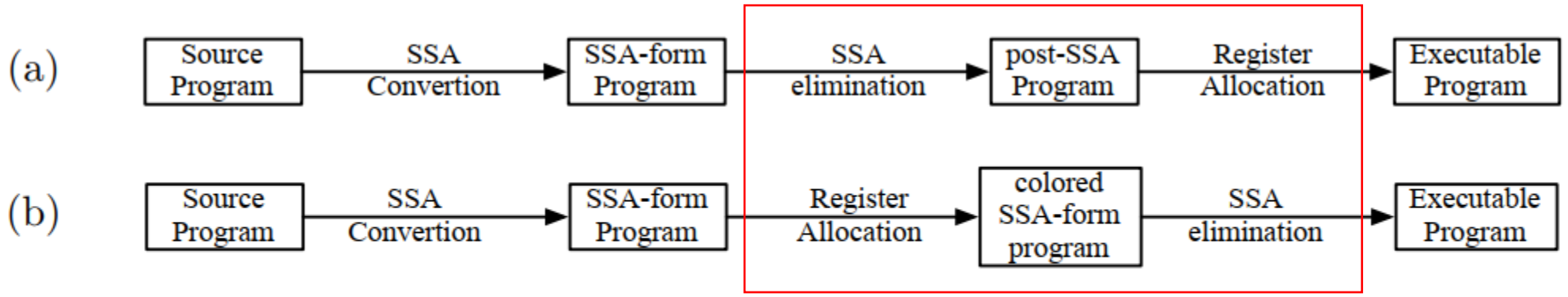


Figure 12: (a) Traditional register allocation, (b) SSA-based register allocation.

From:< A Survey on Register Allocation >P12

总结

- if you have no time, just use linear scan;
- if you have more time, then:
  - if you are in a JIT context, use graph-coloring with the lossy allocator;
  - otherwise:
    * if performance is critical (especially on x86), use optimal spilling from Appel-George with optimal coalescing from Grund and Hack
    * if you like new things, use register allocation via coloring of chordal graphs, or even by puzzle-solving
    * if you just want a widely tested, robust-performanced, well-documented algorithm, use the classical iterated register coalescing from Appel and George.

From: <A SURVEY OF REGISTER ALLOCATION TECHNIQUES> P11

目录

- 基本情况
- 寄存器分配
- 指令选择

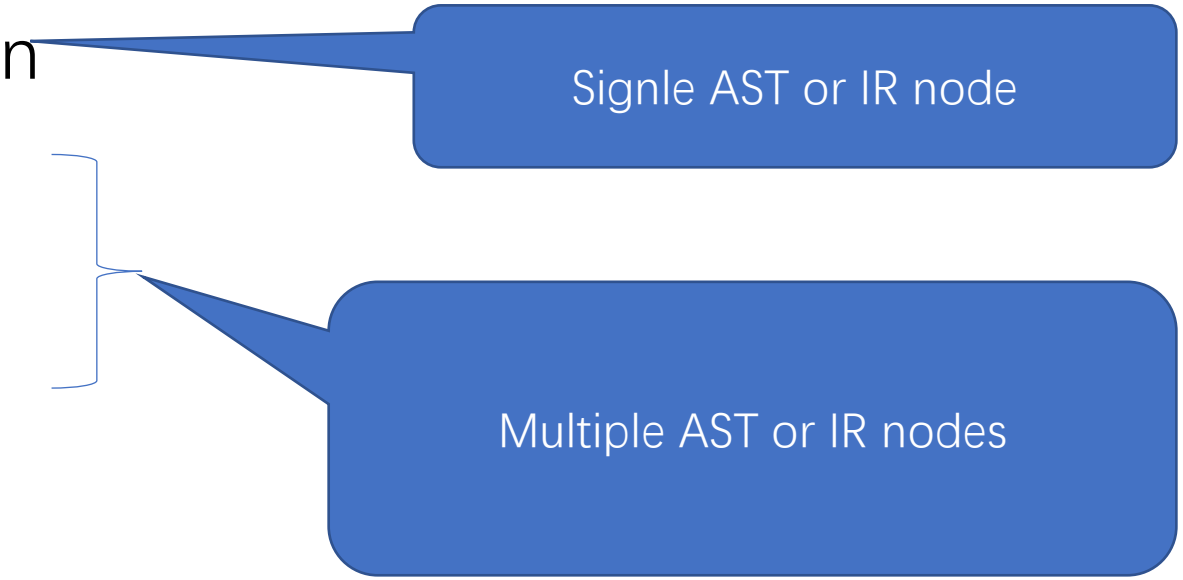
指令选择基础

- To do this we need to decide which machine instructions of the target machine to use for implementing the IR code; this is the responsibility of the instruction selector.
- 指令选择可以分为两个部分：
  - pattern matching – detecting when and where it is possible to use a certain machine instruction;
  - pattern selection – deciding which instruction to choose in situations when multiple options exist.

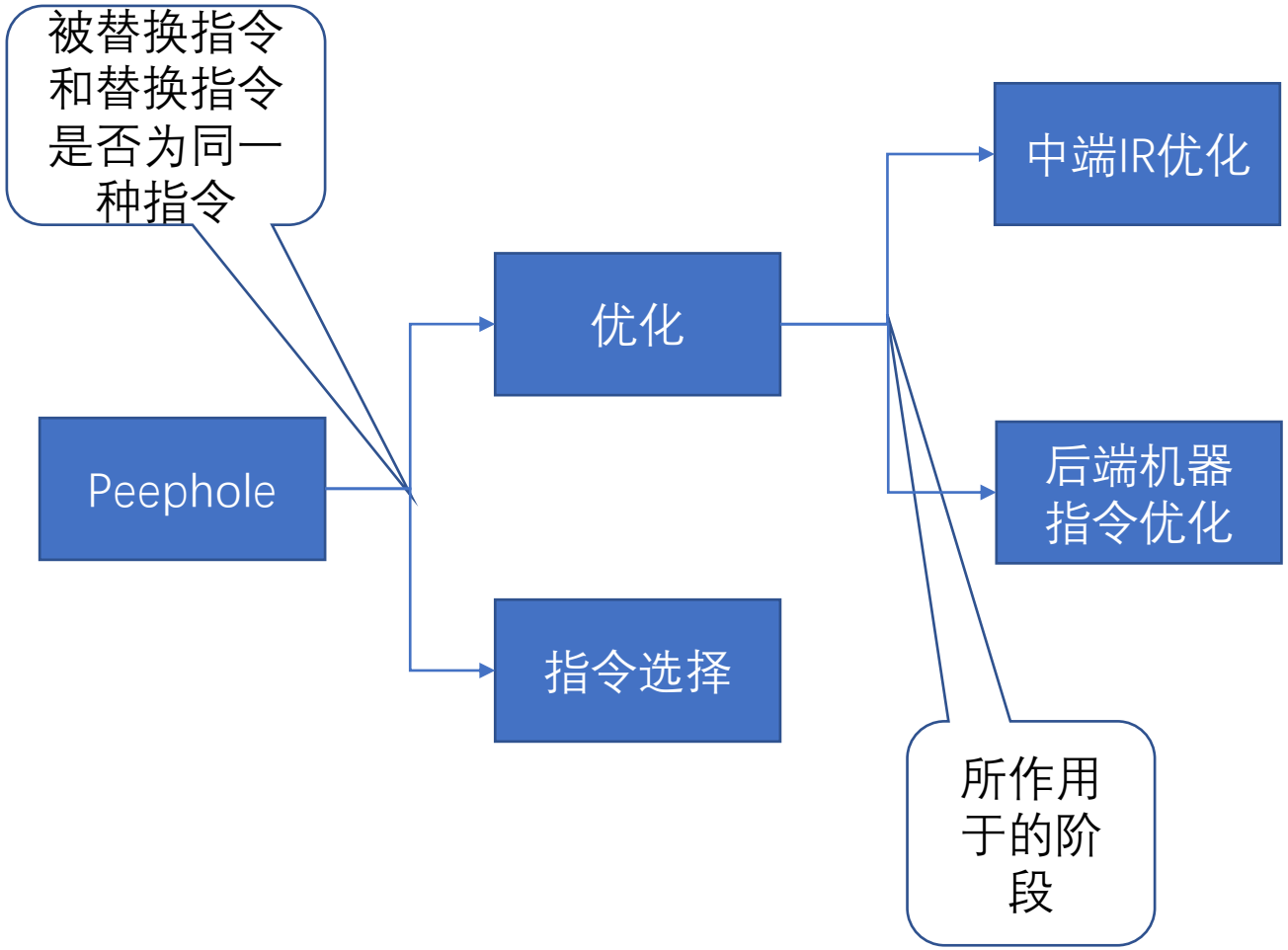
From:< Survey on Instruction Selection >P3

指令选择分类

- Macro expansion
- Tree covering
- DAG covering
- Graph covering



Peephole optimization



Common peephole optimization techniques

Table 1. Common peephole optimization techniques

Optimization	McKeeman[1]	Bagwell Jr[5]	Wulf *et al.*[6]	Tanenbaum *et al.*[7]
Elimination of redundant instructions				
Removal of redundant load and store instructions	✓		✓	
Removal of unreachable instructions			✓	
Removal of useless test and compare instructions			✓	
Removal of inconsequential instruction sequences (null sequences)	✓	✓	✓	✓
Improvements in flow of control				
Elimination and coalescing of jump instructions			✓	
Modification of comparisons				✓
Algebraic simplifications				
Evaluation of constant expressions (constant folding)	✓	✓		✓
Modifying instructions to simpler forms (operator strength reduction)		✓	✓	✓
Simplification of logic expressions		✓		
Reordering arithmetic instructions	✓			✓
Use of machine idioms				
Addressing optimizations			✓	✓
Using special instructions	✓			✓

From:< . Fifty years of peephole optimization > P2187

New topics related to peephole optimization

- The concept of peephole optimization can be extended from general purpose programing languages to domain-specific languages.
- It will be interesting to see how object oriented programing can be used to realize peephole optimization.
- Peephole optimization can be applied in just-in-time compilers too.
- Peephole optimization can also be used in compilers for parallel computers.
- Peephole optimization may be used to replace more energy consuming instruction sequences by equivalent but less energy-consuming instruction sequences.

From:< . Fifty years of peephole optimization > P2189-2190

Thanks!