Machine Block Placement

code layout optimizations.

龙英池

2022 年 8 月 2 日

目录

代码布局 (Code Layout)

什么是代码布局

指令连续地、有顺序地储存在内存中.LLVM 中，这些优化过程抽象成控制流图中的基本块 (Basic Block, BB) 的排列方式.

```
if (a < b)
    c = 1;
```

(a) C

```
; a is in a0
; b is in a1
; c is in a2
bge a0 a1
    .LBB0_2
j .LBB0_1
li a2 1
.LBB0_2:
```

(b) asm

图: 源代码翻译为汇编

机器代码布局优化

基于机器相关的，代码布局的优化主要包含，基本块放置 (Basic block placement)、基本块对齐 (Basic block alignment)、冷热代码分离 (Hot-Cold Splitting)[1].

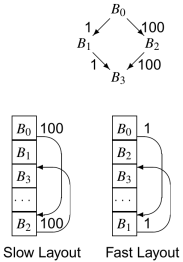
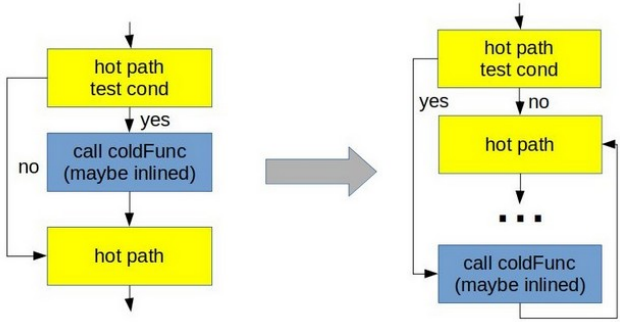


图: 两种不同布局的优劣 [2]

基本块放置 - Basic Block Placement

```
// hot path
if (cond)
coldFunc();
// hot path again
```

(a) C Codes



(b) 布局优化示例

- 不进行分支跳转往往比跳转代价更低 (maintain fall through)
- 更好地利用 Cache (μop-cache) (局部性)

基本块对齐 - Basic Block Alignment

A brief explanation of this manner:

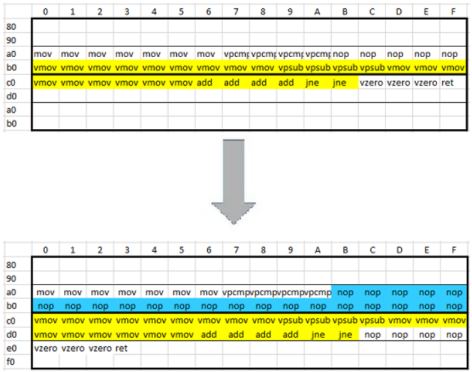


图: 基本块对齐

冷热代码分离 - 分离前

冷热代码分离的核心思想是，将冷代码 (如错误处理) 分成一个独立的函数调用，从原过程 (procedure) 中分离，然后用一个函数调用语句代替冷代码，从而提高热代码的连续性、局部性.

```
void foo(bool cond1, bool cond2) {
    // hot path
    if (cond1)
        // cold code 1
    //hot code
    if (cond2)
        // cold code 2
}
```

冷热代码分离 - 分离后的代码

```
void foo(bool cond1, bool cond2) {
// hot path
if (cond1)
    cold1();
//hot code
if (cond2)
    cold2();
}

void cold1() __attribute__((noinline)) { // cold code 1 }
void cold2() __attribute__((noinline)) { // cold code 2 }
```

冷热代码分离 - 图

这样做的好处是，热指令都会存在同一个 cache line 里，可以提高 CPU 前端数据结构的利用效率，例如 I-cache 和 DSB-Cache

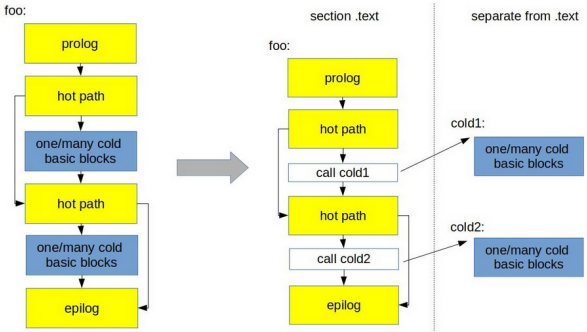


图: 将冷代码过程转换为函数调用

代码块放置 (Block Placement)

从 CFG 到代码布局 (Code Layout) - Extended-TSP

控制流图 (CFG) 由若干基本块 (Basic Blocks) 组成，代码块放置的目标是让经常连在一起执行的基本块，在最终的代码布局 (Code Layout) 中放在一起，这样最直接的好处就是更好地利用 I-cache.

事实上这个问题直到 2021 年才被形式化，背后的数学问题是 Ext-TSP [3]. 给定无向图 $G = (V, E)$，带有正边权 $w : E \to R^+$，一个不递增的单调代价函数 $f(\cdot)$，满足 $f(1) = 1$，$f(i) = 0, i > k$，k 是一个问题相关的常量.

构建 V 的一个序列 d，最优化

$$\sum_{(u,v)\in E} f(|d_u - d_v|) \cdot w(u, v) \tag{1}$$

$d_u \in \{1, \ldots, |V|\}$，是 u 在序列中的位置.

控制流图链剖分 - Pettis-Hansen

首先对 CFG 建立链剖分，建立的原则是，经常被执行的边被放在一起，形成一条链 (Chain).
一条链包含一个或多个基本块 (BB)，并且每个路径都有一个优先级 (priority) 决定了他们在最终的代码布局中的情况.

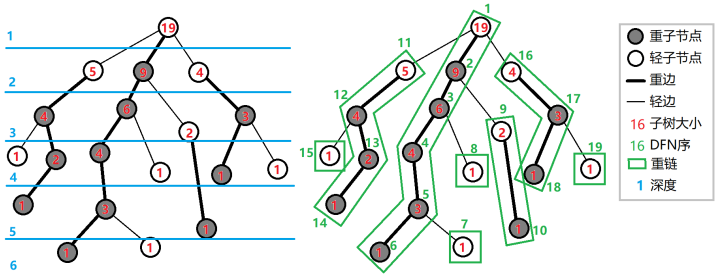


图: [4] 轻重链剖分，树上链剖分的一种

Kruskal ? Prim ? - Pettis-Hansen

CFG 没有树这么好的性质，CFG 是一个有向图. 构建链的方法是每次从边集合中选一条没有被选中的边（按照边的权重顺序），然后形成链状结构.

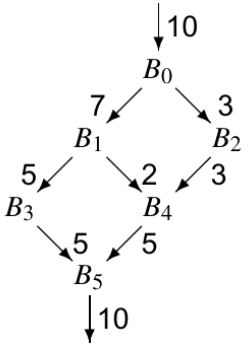
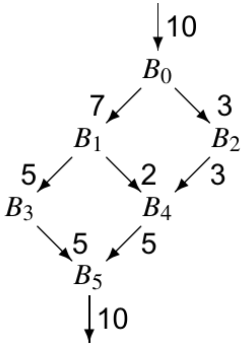


图: CFG 的一个例子 [2]

Greedy heuristic chain decomposition on CFGs

$E \leftarrow |$ edges $|$
for each block b **do**
    make a degenerate chain, d, for b
    priority(d) $\leftarrow E$
end for
$P \leftarrow 0$
for each CFG edge $< x, y >, x \neq y$, in decreasing freq order **do**
    $t \leftarrow$ priority(a)
    append b onto a
    priority(a) $\leftarrow \min$ (t, priority(b), $P{+}{+}$)
end for

Greedy heuristic chain decomposition on CFGs



(a) CFG 的一个例子 [2]

Edge	Set of Chains	P
—	$(B_0)_E$, $(B_1)_E$, $(B_2)_E$, $(B_3)_E$, $(B_4)_E$, $(B_5)_E$	0
(B_0, B_1)	$(B_0, B_1)_0$, $(B_2)_E$, $(B_3)_E$, $(B_4)_E$, $(B_5)_E$	1
(B_3, B_5)	$(B_0, B_1)_0$, $(B_2)_E$, $(B_3, B_5)_1$, $(B_4)_E$	2
(B_4, B_5)	$(B_0, B_1)_0$, $(B_2)_E$, $(B_3, B_5)_1$, $(B_4)_E$	2
(B_1, B_3)	$(B_0, B_1, B_3, B_5)_0$, $(B_2)_E$, $(B_4)_E$	3
(B_0, B_2)	$(B_0, B, B_3, B_5)_0$, $(B_2)_E$, $(B_4)_E$	3
(B_2, B_4)	$(B_0, B_1, B_3, B_5)_0$, $(B_2, B_4)_3$	4
(B_1, B_4)	$(B_0, B_1, B_3, B_5)_0$, $(B_2, B_4)_3$	4

(b) 算法每个过程选择的边，和对应的边集

图: 链剖分实例

基于 WorkList 组合最终的代码布局 (Code Layout)

$t \leftarrow$ chain headed by the CFG entry node
$WorkList \leftarrow \{(t, priority(t))\}$ $\Leftarrow$ Heap[5]
while $WorkList \neq \emptyset$ **do**
    remove a chian c of lowest priority from $WorkList$
    for each block x in c in chain **do**
        place x and the end of assembly codes
    end for
    for each block x in c **do**
        for each edge $< x, y >$ where y is unplaced **do**
            $t \leftarrow$ chain containing $< x, y >$
            if $(t, priority(t)) \notin WorkList$ **then** $WorkList \leftarrow WorkList \bigcup \{(t, priority(t))\}$
            end if
        end for
    end for
end while

Example

Edge	Set of Chains	P
—	$(B_0)_E$, $(B_1)_E$, $(B_2)_E$, $(B_3)_E$, $(B_4)_E$, $(B_5)_E$	0
(B_0, B_1)	$(B_0, B_1)_0$, $(B_2)_E$, $(B_3)_E$, $(B_4)_E$, $(B_5)_E$	1
(B_3, B_5)	$(B_0, B_1)_0$, $(B_2)_E$, $(B_3, B_5)_1$, $(B_4)_E$	2
(B_4, B_5)	$(B_0, B_1)_0$, $(B_2)_E$, $(B_3, B_5)_1$, $(B_4)_E$	2
(B_1, B_3)	$(B_0, B_1, B_3, B_5)_0$, $(B_2)_E$, $(B_4)_E$	3
(B_0, B_2)	$(B_0, B, B_3, B_5)_0$, $(B_2)_E$, $(B_4)_E$	3
(B_2, B_4)	$(B_0, B_1, B_3, B_5)_0$, $(B_2, B_4)_3$	4
(B_1, B_4)	$(B_0, B_1, B_3, B_5)_0$, $(B_2, B_4)_3$	4

(a) 上一个过程形成的链

Step	WorkList	Code Layout
—	$(B_0, B_1, B_3, B_5)_0$	
1	$(B_2, B_4)_3$	B_0, B_1, B_3, B_5
2	$\emptyset$	$B_0, B_1, B_3, B_5, B_2, B_4$

(b) 生成最终的代码布局

图: WorkList 生成最终布局示例

LLVM 中如何实现这个算法

TL; DR

算法主体主要实现在 lib/CodeGen/MachineBlockPlacement.cpp

其中分支概率等信息来源于 Block{Frequency, ProbabilityInfo}

各个 Target 需要实现 { analyze, insert, remove }Branch 等虚函数，RISC-V 在 5 年前，D40808[6] 实现；两年前，D84833 包含了一个间接分支实现 [7]

功能	实现	源代码位置
边权	BlockFrequency	[8]
块放置	MachineBlockPlacement	buildCFGChains()[9]
块对齐	MachineBlockPlacement	alignBlocks()[9]
分支分析	RISCVInstrInfo	analyzeBranch()[10, 6]

表: 各个功能的实现情况和所在的位置

最开始的布局优化 Pass - CodePlacementOptPass

2009 年 10 月，此时 LLVM 大仓库还只有 clang、compiler-rt、llvm. 这时 LLVM 的布局优化 Pass 还没有基于 Macine-IR，CodePlacementOptPass ($O(n^2)$) [11] 将一些循环尾部的，无条件跳转到循环首部 (back-edges) 的块，移动到循环的开头.

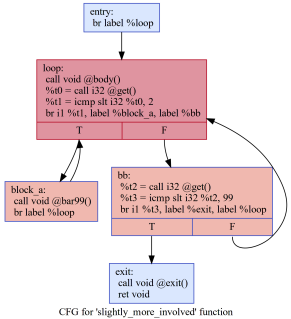


图: block_a 可以被提到循环的开头

Branch Probability Basic Block Placement - Pettis-Hansen

2011 年，基于分支概率的基本块被引入，Branch Probability Basic Block Placement 第一次全名出现在了 LLVM 中 [12]. 这个算法先把所有的 BB 形成若干链，然后基于强连通分量合并这些链，形成最终的 layout.
这个算法很快被废弃，在 11 月完成了现在所看的版本的雏形，同时也是上面介绍给大家的版本 [13, 14]. 这个版本的算法递归地从循环中构建链，同时尽量保证 CFG 的拓扑序正确.

版本	如何处理 Loop	如何从 Chain 到最终布局
[12]	无特殊处理	强连通分量排序
[13]	递归建立 Loop-Based Chains	逆后序遍历
[14]	基于版本 [13] 多建立一个 WorkList	函数整体看成一个 Chain

表: 三个实现的区别

LLVM 对 Pettis-Hansen 算法的改进

2021 年 12 月，MachineBlockPlacement 的基本功能有了新的改进.D113424[15] 引入一个方案，来优化已经完成链剖分之后的，生成最终的代码布局的过程.

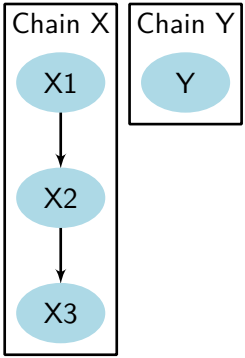


图: 需要合并的两个 Chain 示意

BOLT - Binary optimization and layout tool

Facebook 在 2018 年开源了他们的二进制优化工具 – BOLT[16, 17]. 这个仓库现在已经合并到 LLVM，可以在对编译后的二进制进行基本的重排，冷热代码分离等工作.

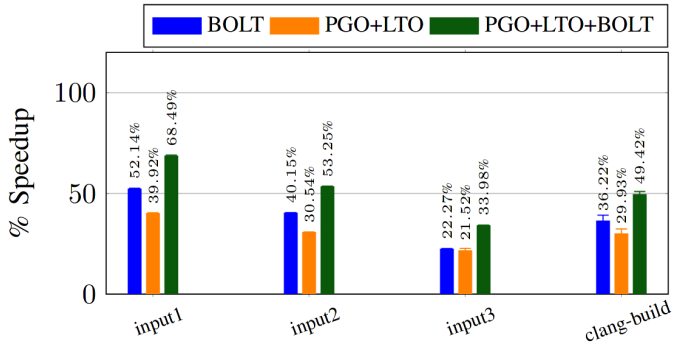


图: 优化后的 Clang

RISC-V 的支持情况

RISC-V 支持 Machine Block Placment

各个 Target 需要实现 { analyze, insert, remove }Branch 等虚函数，RISC-V 在 5 年前，D40808[6] 实现；两年前，D84833 包含了一个间接分支实现 [7].
D40808[6] 中的测试用例非常基础：

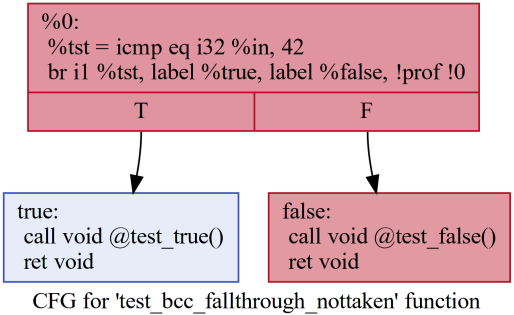


图: 理想布局：%0 → false → true

参考文献 (References)

References I

[1] D. Bakhvalov, "Machine code layout optimizations. | Easyperf." https://easyperf.net/blog/2019/03/27/Machine-code-layout-optimizatoins, 03 2019.

[2] K. D. Cooper and L. Torczon, *Engineering a compiler*. Elsevier, 2011.

[3] J. Mestre, S. Pupyrev, and S. W. Umboh, "On the extended tsp problem," *arXiv preprint arXiv:2107.07815*, 2021.

[4] I, "树链剖分 - OI Wiki." https://oi-wiki.org/graph/hld/, 03 2022.

[5] G. E. Forsythe, "Algorithms," *Communications of the ACM*, vol. 7, no. 6, pp. 347–349, 1964.

[6] LLVM Project, "[RISCV] Implement branch analysis - Phabricator." https://reviews.llvm.org/D40808, 07 2017.

References II

[7] LLVM Project, "[RISCV] Implement indirect branch generation in position independent code for the RISC-V target - Phabricator." https://reviews.llvm.org/D84833, 07 2020.

[8] LLVM Project, "lib/Analysis/BlockFrequencyInfoImpl.cpp - Doxygen." https://llvm.org/doxygen/BlockFrequencyInfoImpl_8cpp_source.html, 07 2022.

[9] LLVM Project, "lib/CodeGen/MachineBlockPlacement.cpp - Doxygen." https://llvm.org/doxygen/MachineBlockPlacement_8cpp_source.html, 07 2022.

[10] LLVM Project, "lib/Target/RISCV/RISCVInstrInfo.cpp - Doxygen." https://llvm.org/doxygen/RISCVInstrInfo_8cpp_source.html, 07 2022.

References III

[11] LLVM Project, "CodePlacementOpt.cpp - Code Placement pass.." https://github.com/llvm/llvm-project/blob/a48f44d9ee15b2d9f696c5de6f3e60dc84639c0f/llvm/lib/CodeGen/CodePlacementOpt.cpp, 10 2009.

[12] LLVM Project, "Implement a block placement pass based on the branch probability." https://github.com/llvm/llvm-project/commit/10281425643667a7385f5bc6047c980fde136a13, 10 2011.

[13] LLVM Project, " Completely re-write the algorithm behind MachineBlockPlacement based on discussions with Andy.." https://github.com/llvm/llvm-project/blob/bd1be4d01c47b96d86681e01001ce373237e4141/llvm/lib/CodeGen/MachineBlockPlacement.cpp, 10 2011.

[14] LLVM Project, "Rewrite #3 of machine block placement. This is based somewhat on the ." https://github.com/llvm/llvm-project/commit/8d150789271d85f227236e78db14ed8857d05fc1, 11 2011.

References IV

[15] LLVM Project, "ext-tsp basic block layout - Phabricator." https://reviews.llvm.org/D113424, 12 2021.

[16] Facebook, "BOLT - Binary Optimization and Layout Tool." https://github.com/llvm/llvm-project/tree/main/bolt, 04 2022.

[17] M. Panchenko, R. Auler, B. Nell, and G. Ottoni, "Bolt: a practical binary optimizer for data centers and beyond," in *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 2–14, IEEE, 2019.