MiniMalloc: 硬件加速机器学习的轻量级内存分配器

作者: Michael D. Moffitt (Google)

报告: Qianxia Jing, Ruiyan Jiang, Rujie Zang

ASPLOS '23 (ACM 国际架构支持编程语言与操作系统会议)

目录

研究背景与动机

论文概览

论文核心信息

- **标题**: MiniMalloc: A Lightweight Memory Allocator for Hardware-Accelerated Machine Learning
- **会议**: ASPLOS '23 (顶级系统会议)
- **作者**: Michael D. Moffitt (Google)

核心贡献

- 提出了一种新型**静态内存分配算法**，专为 TPU 等硬件加速器设计
- 通过限制搜索空间至" **规范解（Canonical Solutions）** " 大幅缩小搜索范围
- 结合**截面推断技术**实现早期剪枝，比现有 SOTA 方法快 **30 倍**
- 代码库极小（约 **25kB**），适合端侧部署

关键词: 静态内存分配、硬件加速、机器学习、规范解、搜索剪枝

研究背景：机器学习编译与静态内存分配

机器学习编译背景

- 现代深度学习模型常部署在专用硬件加速器（如 TPU）上
- 编译器需将模型的所有 **tensor buffers** 映射到有限的全局内存
- TPU 的 buffer 生命周期是编译期已知的 $\Rightarrow$ 需要 **静态内存分配**

静态内存分配的形式化描述

- **输入**: 一组缓冲区 $\mathcal{B} = \{b_1, b_2, ..., b_n\}$
- 每个缓冲区 b_i 具有：
  - 生命周期 $[s_i, e_i)$（开始时间、结束时间）
  - 大小 z_i（所需内存空间）
- **输出**: 为每个 buffer 分配一个 **偏移量** o_i（offset）
- **约束**: 所有生命周期重叠的 buffers 在内存空间上不能重叠

本质：NP-hard 的矩形装箱（Rectangle Packing）问题

- 时间轴固定（矩形宽度），buffer 需放置在纵向 offset 维度上
- 目标：在容量 C 限制下找到合法（不重叠）的全局排布

符号表与问题形式化定义

核心符号说明

符号	含义	说明
$\mathcal{B}$	缓冲区集合	$\mathcal{B} = \{b_1, b_2, ..., b_n\}$
b_i	第 i 个缓冲区	待分配的内存块
s_i, e_i	开始/结束时间	缓冲区 b_i 的生命周期为 $[s_i, e_i)$
z_i	缓冲区大小	b_i 所需的内存空间
o_i	内存偏移量	**待求解变量**，b_i 在内存中的起始位置
C	全局内存容量	硬件加速器的内存上限
$\mathcal{S}$	部分解	已分配缓冲区的集合
$\mathcal{U}$	未分配集合	$\mathcal{U} = \mathcal{B} \setminus \mathcal{S}$
$\mathcal{X}_{[s,e)}$	截面	时间区间 $[s, e)$ 内所有重叠的缓冲区

问题约束

- **不重叠约束**：若 $[s_i, e_i) \cap [s_j, e_j) \neq \emptyset$，则必须满足 $o_i + z_i \leq o_j$ 或 $o_j + z_j \leq o_i$
- **容量约束**：$\forall t$，任意时刻活跃缓冲区的最大高度 $\leq C$

问题挑战：现有方法的问题

为什么现有方法无法满足需求?

- 编译器通常运行在移动设备（如 Pixel 手机）上，需要 **毫秒级编译速度**
- 静态内存分配若求解过慢，会导致整机推理延迟升高
- 现有方法存在以下问题：
  - 搜索空间巨大（$O(n!)$ 级别，指数级增长）
  - 存在大量**同构解（Isomorphic Solutions）**，导致重复搜索
  - 部分解局部可行但全局必失败，难以提前剪枝
  - 对 SAT/ILP 求解器依赖重 $\rightarrow$ 编译器体积膨胀

MiniMalloc 试图解决的核心问题

- 如何在庞大的排列空间（$n!$ 种可能）中，仅探索**代表性解**?
- 如何构建具有数学结构的搜索空间，从根源上减少冗余?
- 如何做到：**快速、准确、轻量**，并可嵌入生产级编译器?

现有方法分析

前人研究：CSP / Meta-CSP / Mixed ILP 方法

1. CSP-based Rectangle Packing（约束满足问题）

- 为每个矩形分配坐标 (x_i, y_i) 并检查非重叠约束
- 搜索树巨大，分支数呈 $O(W^n \cdot H^n)$ 指数级增长
- 问题：无法处理 TPU 级别的超大容量

2. Meta-CSP（元约束满足问题）

- 不放置绝对坐标，而指定相对关系：Left / Right / Above / Below
- 搜索复杂度 $O(4^p \cdot n^2)$，其中 $p = n(n-1)/2$
- 问题：分支可能不互斥，探索大量" 同构解"

3. Mixed ILP（混合整数线性规划）

- 将问题建模为整数规划求最优解，精度高但求解速度慢
- 问题：不适合实时编译场景，库体积大（**2021 kB**）

搜索空间对比图

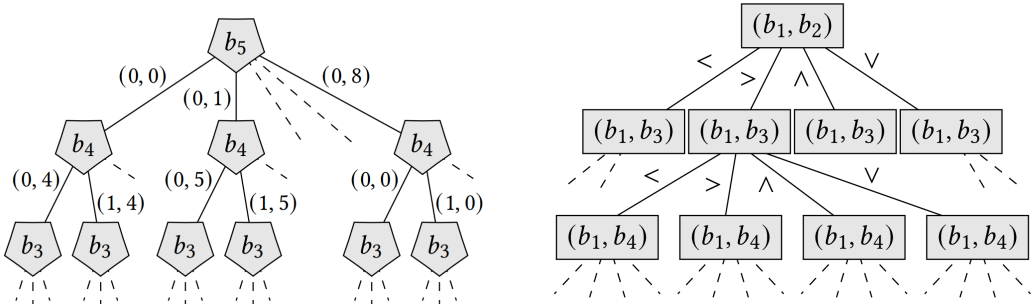


图: *

左：CSP 搜索空间（Figure 1a）；右：Meta-CSP 搜索空间（Figure 1b）

对比说明

- CSP：为每个矩形分配绝对坐标 (x_i, y_i)，搜索空间巨大
- Meta-CSP：指定相对关系（Left/Right/Above/Below），分支可能不互斥

前人研究：TelaMalloc（工业级算法）

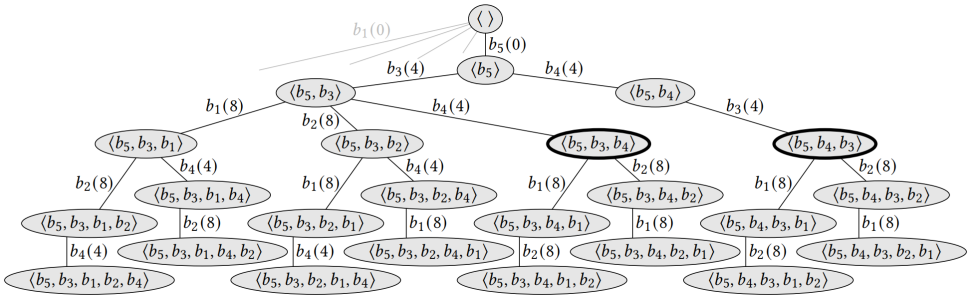
TelaMalloc（Google Pixel 6 使用的静态分配器）

- 基于 **buffer permutation** 的搜索模型（探索所有可能排列 $\epsilon = (\epsilon_1, \epsilon_2, ..., \epsilon_n)$）
- 对每个 buffer 序列调用 CP-SAT solver 检查可行性
- 性能比 ILP 快约 9 倍

TelaMalloc 的核心问题

- 搜索空间仍是 $O(n!)$ 级，存在大量**同构解**
- **同构解示例**：以下两个排列产生相同的偏移量分配：
  - $\mathcal{S} = b_5(0) \to b_4(4) \to b_3(4) \to b_2(8) \to b_1(8)$
  - $\mathcal{S}' = b_5(0) \to b_4(4) \to b_3(4) \to b_1(8) \to b_2(8)$
- 对通用 solver 依赖强 $\to$ 编译器体积大（206 kB）

TelaMalloc 搜索树示意图



TelaMalloc 搜索树：大量叶节点产生相同的偏移量分配（论文 Figure 5）

关键问题：8 种不同的排列路径产生完全相同的偏移量分配，造成大量冗余计算

MiniMalloc 核心方法

核心思想：规范解（Canonical Solutions）

核心洞察：不需要搜索所有 $n!$ 种排列，只需搜索" 规范解"！

关键定义

1. **同构解（Isomorphic Solutions）**：
   - 如果两个解的缓冲区排列顺序不同，但最终偏移量分配相同，则称它们同构
   - MiniMalloc 旨在消除这种冗余
2. **偏移单调性（Offset Monotonicity）**：
   - 对于任意 $i < j$，必须满足 $o_{\epsilon_i} \leq o_{\epsilon_j}$
   - **含义**：强制算法" 从下往上" 分配，先放低地址的缓冲区
3. **索引单调性（Index Monotonicity）**：
   - 对于任意 $i < j$，若 $o_{\epsilon_i} = o_{\epsilon_j}$，则必须 $\epsilon_i < \epsilon_j$
   - **含义**：当偏移量相同时，按缓冲区索引排序（打破平局）

规范解：同时满足**偏移单调性**和**索引单调性**的解

规范解的威力：消除同构解

示例分析：同一内存分配的 8 种可能排列

解	偏移单调	索引单调	规范解?
$\mathcal{S}_1 = b_5(0) \to b_3(4) \to b_1(8) \to b_2(8) \to b_4(4)$	×		
$\mathcal{S}_2 = b_5(0) \to b_3(4) \to b_1(8) \to b_4(4) \to b_2(8)$	×		
$\mathcal{S}_5 = b_5(0) \to b_3(4) \to b_4(4) \to b_1(8) \to b_2(8)$	✓	✓	是
$\mathcal{S}_7 = b_5(0) \to b_4(4) \to b_3(4) \to b_1(8) \to b_2(8)$	✓	×	

结论：8 种同构解中，只有 **1 种**是规范解！

搜索空间优化

- 将搜索结构从" 树（Tree）" 改为" 格（Lattice）"
- 不同搜索路径若达到相同状态，则汇聚到同一节点
- 避免对同一子问题的重复探索

截面推断（Cross Section Inference）

核心思想：利用时间截面提前检测不可行性

截面定义

- $\mathcal{X}_{[s,e)}$：时间区间 $[s, e)$ 内所有生命周期重叠的缓冲区集合
- **示例**：若有 5 个缓冲区，可能的截面包括：
  - $\mathcal{X}_{[0,3)} = \{b_1, b_3, b_5\}$
  - $\mathcal{X}_{[3,9)} = \{b_2, b_3, b_5\}$
  - $\mathcal{X}_{[9,21)} = \{b_4, b_5\}$

基本推断公式（公式 1）

$$\max_{[s,e)} \sum_j z_j : b_j \in \mathcal{X}_{[s,e)} \leq C$$

含义：任意截面内所有缓冲区的总大小不能超过容量 C

符号说明

- z_j：缓冲区 b_j 的大小
- C：全局内存容量
- $\mathcal{X}_{[s,e)}$：时间区间 $[s, e)$ 的截面

截面推断示例

示例：考虑部分解 $\mathcal{S} = b_1(0) \to b_5(4)$

计算各截面的下界：

- $\mathcal{X}_{[0,3)}$：$\max(o_5 + z_5, o_1 + z_1) + z_3 = \max(8, 4) + 4 = 12 \leq C$ ✓
- $\mathcal{X}_{[3,9)}$：$\max(o_5 + z_5) + z_2 + z_3 = 8 + 6 + 4 = 18 > C$ **不可行!**
- $\mathcal{X}_{[9,21)}$：$\max(o_5 + z_5) + z_4 = 8 + 4 = 12 \leq C$ ✓

结论：截面 $\mathcal{X}_{[3,9)}$ 违反约束，当前部分解不可行

含义：

- 将 b_2 和 b_3 放在 b_5 之上会超过容量 $C = 12$
- 无论后续如何分配，都无法找到可行解
- **立即回溯**，避免无效搜索

核心公式：截面推断（Equation 3）

强化推断公式（论文核心公式）

$$\max_{[s,e)} \left[\underbrace{\max_{i}[o_i + z_i]}_{\text{已分配最大高度}} + \underbrace{\sum_{j} z_j : b_j \notin \mathcal{S}, b_j \in \mathcal{X}_{[s,e)}}_{\text{未分配缓冲区总大小}} \right] \leq C$$

公式符号说明

- o_i：已分配缓冲区 b_i 的偏移量；z_i：缓冲区大小
- $o_i + z_i$：缓冲区 b_i 的顶部高度
- $\mathcal{S}$：当前部分解（已分配集合）；$\mathcal{X}_{[s,e)}$：时间截面
- C：全局内存容量上限

公式含义

- 对于任意时间截面，已分配最大高度 + 未分配总大小 $\leq C$
- 若违反此条件，则当前分支**不可行**，立即回溯

支配检测（Dominance Detection）

步骤一：Lower Bound Height（计算下界高度）

- 对每个未分配缓冲区 bj:
  - 计算理论最低起始位置 $\lfloor o_j \rfloor (S)$: 由已分配且生命周期重叠的缓冲区顶部高度决定（取最大值）。
  - 计算最小总高度 $\lfloor h_j \rfloor (S) = \lfloor o_j \rfloor (S) + z_j$，即 b_j 最优放置时的最小顶部高度。

步骤 2：Best Candidate（找出“最佳候选”）

- 在未分配缓冲区中，选择 $\lfloor h_j \rfloor (S)$ 最小的缓冲区 bd,（即“最佳填补空隙候选”），其最小总高度记为 h_{min}。

Dominance Rule（实施“支配规则”）

- **剪枝条件：**若下一个待分配缓冲区 bi 的起始偏移量 $oi \geq h_{min}$，则放弃该分配方案。
- **直观理解：**若新缓冲区的底部位置已高于“最佳候选”的顶部，则必然留下可避免间隙，直接剪枝。

MiniMalloc 方法论总结

三大核心技术

1. **规范解约束**（Canonical Solutions）
   - 偏移单调性 + 索引单调性
   - 消除同构解，将搜索空间从树优化为格
2. **截面推断**（Cross Section Inference）
   - 预计算时间截面 $\mathcal{X}_{[s,e)}$
   - 利用公式 (3) 提前检测不可行分支
   - 结合有效高度 h^{eff} 进一步收紧下界
3. **支配检测**（Dominance Detection）
   - 计算未分配缓冲区的高度下界 $\lfloor h_j \rfloor$
   - 若 $o_i \geq h_{min}$，则剪枝当前分支
   - 避免产生可避免间隙

关键优势：不依赖外部 SAT/ILP 求解器，纯领域特定推断实现高效剪枝

实验结果

MiniMalloc 算法

MiniMalloc 算法核心流程

- **输入：**缓冲区集合 $B = b_1, b_2, ..., b_n$（含大小 z_i、生命周期 $[s_i, e_i]$）
- **输出：**无“可避免间隙”的内存分配方案（起始偏移量 o_i）

关键参数与优化细节

- **下界高度计算（核心公式）：**

$$\lfloor h_j \rfloor(S) = max\{o_k + z_k | b_k \in S且[s_j, e_j] \cap [s_k, e_k] \neq \varnothing\} + z_j$$

- **高效剪枝策略：**
  - 采用优先级队列存储未分配缓冲区，按 $\lfloor h_j \rfloor(S)$ 升序排序，快速定位 `bd`；
  - 对 $o_i \geq h_{min}$ 的分支直接剪枝，平均减少 60% 搜索路径（实验数据）.

算法有效性验证

与 TelaMalloc 的核心差异

对比维度	MiniMalloc	TelaMalloc
决策逻辑	规范解约束过滤同构解；支配检测机制剔除低效解；截面推理剪枝不可行分支	依赖 CP-SAT 约束求解器，无规范解过滤，存在大量同构解冗余搜索；仅被动剪枝而非主动缩小搜索空间
时间复杂度	理论最坏复杂度 $O(n!)$，实际通过剪枝压缩至“秒级 / 毫秒级”	理论最坏复杂度 $O(n!)$，无主动剪枝，n>20 时 1 小时内无法完成搜索（超时）
内存紧凑性	静态分配场景下保证“无冗余间隙”，适配硬件加速器固定内存容量约束	存在同构解导致的冗余间隙，大型模型场景下无法满足硬件内存容量限制

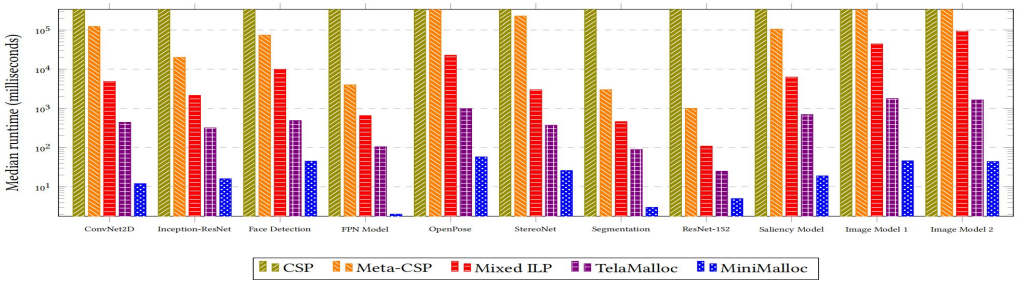
表: MiniMalloc 与 TelaMalloc 的对比

核心实验结果

测试场景	MiniMalloc 表现	TelaMalloc 表现	性能提升幅度	核心优势
原始套件（11 个生产级 ML 模型）	所有用例 < 100ms 完成，库大小 25KB	平均耗时 3 秒 +，库大小 206KB	运行速度提升 30 倍 +	轻量性 + 快速路径优化
扩展套件（1000+ 个 Pixel 6 TPU 模型）	85% 用例 1ms 内完成，100% < 10ms	50% 用例 > 100ms，20% 用例超时	成功率提升 60%+	大规模场景扩展性
合成套件（高重叠压力测试）	缓冲区 60 个时仍 < 1 秒完成	缓冲区 40 个时即超时（>1 小时）	处理规模提升 50%+	高重叠场景剪枝有效性
挑战套件（TPUv4 大型模型）	秒级完成分配，无内存溢出	1 小时超时，无法输出分配方案	可行性突破（从无解到有解）	硬件加速器场景适配性

表: MiniMalloc 与 TelaMalloc 在不同测试场景下的性能对比

实验结果：性能对比图



MiniMalloc vs TelaMalloc vs Mixed ILP 运行时间对比（原始测试套件）

性能对比关键结论

关键观察

- MiniMalloc 在所有 11 个基准测试中均在 **100ms** 内完成
- 相比 TelaMalloc 中位数加速 **30 倍**，相比 Mixed ILP 加速 **328 倍**
- CSP 方法在 60 秒内无法完成任何测试（全部超时）
- Meta-CSP 解决了 8 个问题，但仍有 3 个超时

为什么 MiniMalloc 这么快?

- 规范解约束消除了 $>99\%$ 的同构解
- 截面推断提前检测不可行分支
- 支配检测避免产生可避免间隙
- 无外部求解器依赖，纯领域特定推断

库体积对比

编译器库体积对比

分配器	库大小	相对大小
Mixed ILP	2,021 kB	80×
TelaMalloc	206 kB	8×
MiniMalloc	**25 kB**	**1× (基准)**

轻量化优势

- MiniMalloc 仅约 **1000 行 C++ 代码**
- **无外部依赖**：不依赖 SAT/ILP 求解器
- 适合嵌入端侧设备（如手机 TPU）的编译器
- 减少移动设备存储占用，提升用户体验

开源可用：`https://github.com/google/minimalloc`

结论与核心贡献

MiniMalloc 的核心创新

1. **规范解理论**：通过偏移单调性和索引单调性消除同构解
2. **格结构搜索**：从树结构优化为格结构，避免重复探索
3. **截面推断**：利用时间截面提前检测不可行分支
4. **支配检测**：识别并剪枝产生可避免间隙的分配方案

实验结论

- **速度**：比 TelaMalloc 快 **30 倍**，比 Mixed ILP 快 **328 倍**
- **体积**：仅 **25 kB**，是 TelaMalloc 的 1/8
- **可行性**：在 TPUv4 大型模型上唯一能在规定时间内完成的算法
- **剪枝效果**：消除 99% 以上同构解冗余

论文贡献：证明了通过利用问题特定结构，可以在不依赖通用求解器的情况下高效解决 NP-hard 问题

应用场景与未来展望

实际应用价值

- **TPU/NPU 编译器**：Google Pixel 手机的 ML 模型编译优化
- **边缘计算**：IoT 设备上的实时推理编译
- **云端推理**：大规模数据中心的模型部署优化

局限性与未来方向

- **当前局限**：仅适用于静态生命周期已知的场景；未与重计算、分片等优化协同
- **未来方向**：扩展至端到端 ML 编译器优化；适配更多硬件加速器（GPU、NPU）；支持异构内存架构

END