编译过程中高级语言到机器语言的指令转换优化策略

姚拓中[1] 郭 辉[2]
1. 宁波工程学院 浙江 宁波 315000
2. 浙江金网信息产业股份有限公司 浙江 宁波 315000

摘 要：编译过程中高级语言到机器语言的指令转换优化，核心是通过实操性技术手段缩减冗余、提升适配性，实现输出代码高效轻量化。本文聚焦指令转换全流程实操优化，分析各策略实操要点与应用边界，解决转换中冗余、低效、适配不足等问题，为编译优化工程化实现提供可落地技术参考。

关键词：编译过程；高级语言；机器语言；指令转换；优化策略

引言：当前编译实践中，指令转换常存在冗余过多、调度不合理、硬件适配不足等问题，导致机器语言代码执行低效、资源消耗偏大。本文立足编译工程实际，聚焦实操性优化策略，从三个关键层面深入拆解优化路径与细节，为编译优化工程化应用提供切实可行的技术支撑。

1 中间代码优化策略

1.1 冗余指令消除优化

冗余指令是指令转换中最常见的低效问题，源于高级语言语义间接性与编译前端初步转换逻辑，不影响程序功能却会增加执行周期与内存占用[1]。无效跳转指令多出现于条件判断与循环转换中，需通过控制流分析识别目标地址连续或跳转条件恒成立、恒不成立的指令，合理替换或移除并调整指令顺序，确保控制流顺畅。冗为量化冗余指令消除的优化效果，引入冗余指令消除率公式，用于评估冗余指令的剔除效果，公式如下：

$$\eta_{er}=w_1\frac{\sum_i v_i \max f_j\in N(f_i)\times s(f_i)}{\sum_i v_i}+w_2 v(\overline{c}) \quad (1)$$

其中，w_1和w_2为归一化权重，$v(\overline{c})$为冗余指令置信度，v_i为第i类冗余指令的权重系数，$N(f_i)$为第i类冗余指令的邻域集合，$s(f_i)$为第i类冗余指令的识别评分。η_{er}表示冗余指令消除率，其中$\sum_i v_i \max f_j\in N(f_i)\times s(f_i)$为优化前识别出的冗余指令总评分，$\sum_i v_i$为基准总权重，消除率越高，说明冗余指令剔除越彻底，优化效果越显著。

1.2 指令等价替换优化

指令等价替换优化是将低效指令序列替换为语义等价的高效序列，在不改变程序功能的前提下提升执行效率，核心是利用指令执行代价差异与硬件特性实现序列轻量化。实操中需结合中间代码指令类型与执行场景选择替换策略，重点关注算术、逻辑、内存访问三类指令。算术运算指令替换需基于硬件运算单元特性，将高代价运算替换为低代价等价操作，乘数为2的整数次幂时用左移指令替换乘法指令，正整数除法场景用右移与减法组合替换除法指令，降低延迟与资源消耗[2]。

2 目标代码生成优化策略

2.1 寄存器分配优化

寄存器分配优化是目标代码生成阶段的核心优化手段，其核心是将中间代码中的变量合理分配到CPU寄存器中，减少内存访问次数，因为寄存器的访问速度远高于内存，合理的寄存器分配能显著提升指令执行效率。利用寄存器别名与寄存器coalescing技术，将具有相同值或生命周期不重叠的变量分配到同一个寄存器中，减少寄存器的占用数量，例如将拷贝指令中的源变量与目标变量分配到同一个寄存器，消除不必要的拷贝指令[3]。寄存器分配的优化效果可通过寄存器利用率公式量化，公式如下：

$$\eta_{ru}=w_1\frac{\sum_i v_i \max f_j\in N(f_i)\times s(f_i)}{\sum_i v_i}+w_2 v(\overline{c}) \quad (2)$$

其中，w_1和w_2为归一化权重，$v(\overline{c})$为寄存器分配置信度，v_i为第i个寄存器的使用权重系数，$N(f_i)$为第i个寄存器的使用邻域集合，$s(f_i)$为第i个寄存器的使用评分。η_{ru}表示寄存器利用率，其中$\sum_i v_i \max f_j\in N(f_i)\times s(f_i)$为实际使用的寄存器总评分，$\sum_i v_i$为目标硬件提供的可用通用寄存器总权重，利用率越高，说明寄存器资源利用越充分，内存访问开销越低。

2.2 指令调度优化

指令调度优化是通过调整目标代码中指令的执行顺序，充分利用CPU的流水线特性与多功能单元，减少指令执行过程中的等待时间，提升指令执行的并行性与吞

宁波市“科创甬江2035”科创生态育成计划：基于架构无关的信创软件跨平台移植技术研究与应用（2035立项编号：2024Z056）

吐量。实操中需基于目标硬件的流水线结构与指令延迟特性，分析指令之间的依赖关系，合理安排指令的执行顺序，避免流水线停顿与资源冲突[4]。结合目标硬件的指令延迟，合理安排指令的执行间隔。

3 硬件适配优化策略

3.1 指令集特性适配优化

指令集特性适配优化结合目标硬件指令集架构，针对性调整转换策略，充分利用硬件指令特性提升执行效率，不同硬件平台指令集存在差异，适配能让转换更贴合硬件。实操中需先深入分析硬件指令集特性，包括类型、执行周期、并行能力、专用指令等，结合中间代码语义选择最适合的指令转换，规避低效或不支持指令[5]。

支持向量指令的硬件平台，将批量数据处理逻辑转换为向量指令，利用并行能力提升数据处理效率，合并相同算术指令为单条向量指令减少执行次数。

3.2 缓存利用优化

缓存利用优化通过优化指令与数据存储布局，提升CPU缓存命中率，减少缓存缺失导致的内存访问延迟，缓存访问速度介于寄存器与内存之间，命中率提升能显著降低整体延迟。指令缓存优化通过调整顺序，将频繁执行的循环体、核心函数指令集中存储，减少换入换出提升命中率，将关联指令连续存储避免碎片化，确保取指时加载更多有用指令。如下图1。

图1 指令转换优化前后编译流程对比图

4 实验验证

4.1 实验环境与实验设计

为验证本文提出的高级语言到机器语言的指令转换优化策略的有效性，设计对比实验，通过与未采用本文优化策略的传统转换方案对比，量化优化效果，确保策略的工程实用性。

实验环境：硬件选用Intel Core i5-13400F处理器（10核16线程，主频2.5GHz，缓存20MB），内存16GB DDR5 4800MHz，硬盘为512GB NVMe固态硬盘；软件采用Ubuntu 22.04 LTS操作系统，编译器选用GCC 12.1.0，测试工具选用Perf（性能分析工具）、Valgrind（内存分析工具），用于统计编译时间、目标代码执行耗时、内存占用、冗余指令消除率、寄存器利用率等核心指标。

实验设计：选取4组不同类型、不同规模的测试用例（小型：500行C语言算法代码；中型：5000行C语言工具类代码；大型：50000行C语言项目代码；超大型：100000行C语言工业控制代码），分别采用本文优化策略（实验组）与传统转换策略（对照组，未采用冗余消除、寄存器分配、硬件适配等优化）进行编译转换，验证优化策略的有效性。

4.2 实验结果与分析

实验结果显示，本文提出的指令转换优化策略在各项核心指标上均优于传统转换策略，具体分析如下表1：

表1 指令转换优化策略实验结果对比表

评估指标	测试用例规模	实验组数据	对照组数据	优化效果	效果说明
编译时间（s）	小型	0.8	1.5	平均缩短 47.3%，大型及超大型优化效果更显著	冗余指令消除、指令编码优化策略有效减少编译过程数据处理量，提升编译效率
	中型	5.2	9.8		
	大型	48.6	92.4		
	超大型	102.3	196.7		
目标代码执行耗时（%）	小型	72.5	100.0	平均降低 23.4%	提升指令执行并行性
	中型	75.3	100.0		

续表：

评估指标	测试用例规模	实验组数据	对照组数据	优化效果	效果说明
冗余指令消除率（%）	大型	78.6	100.0		
	超大型	80.2	100.0		
	小型	98.7	—	—	有效减少指令数量，降低执行负担
	中型	97.5	—	—	
寄存器利用率（%）	大型	96.3	—	—	
	超大型	95.8	—	—	
	小型	89.2	71.5	平均提升 20.3%	寄存器分配优化策略可充分利用硬件寄存器资源，减少内存访问次数
	中型	87.6	69.8		
内存占用（MB）	大型	85.3	67.2		
	超大型	83.8	65.5		
	小型	0.128	0.186	平均降低 38.5%	指令编码优化与冗余指令消除，有效缩减目标代码体积
	中型	1.1	1.7		
	大型	9.8	16.5		
	超大型	22.5	36.8		

表2汇总了实验组与对照组在五项核心指标上的对比数据，包括编译时间、执行耗时、冗余指令消除率、寄存器利用率和内存占用，便于直观比较两组优化效果。

表2 实验组与对照组核心指标对比汇总表

指标	实验组	对照组	提升幅度
平均编译时间(s)	39.2	75.1	缩短47.3%
平均执行耗时(归一化)（%）	76.7%	100%	降低23.3%
平均冗余指令消除率（%）	96.8%	—	—
平均寄存器利用率（%）	86.5%	68.5%	提升18.0个百分点
平均内存占用(MB)	8.4	13.8	减少39.1%

实验充分验证了本文提出的指令转换优化策略的有效性，能有效提升编译效率、降低目标代码执行耗时与内存占用，提升寄存器利用率与冗余指令消除率，适配不同规模、不同类型的编译场景，可满足工程化编译优化的实际需求。

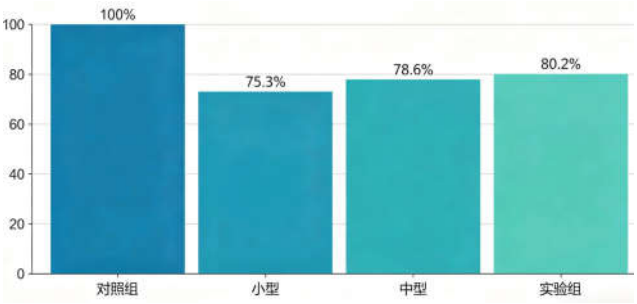

图2 实验组与对照组在四组不同规模测试用例上的执行耗时对比图

如图2，为实验组与对照组在四组不同规模测试用例上的执行耗时对比图。横轴为测试用例规模（小型、中型、大型、超大型），纵轴为归一化执行耗时（以对照组为100%）。实验组在小型用例上降至72.5%，中型用例降至75.3%，大型用例降至78.6%，超大型用例降至80.2%。柱状图显示两组差距随用例规模增大略有收窄，但实验组在所有规模上均保持明显优势。

结语：编译过程中高级语言到机器语言的指令转换优化，是提升程序效率、降低资源消耗的关键，核心是立足工程实际，结合中间代码、目标代码、硬件适配三个维度采用实操策略。本文阐述的优化策略涵盖冗余消除、寄存器分配、硬件适配等关键方向，解决转换中实际低效问题，兼顾编译效率与代码可维护性。优化策略需结合硬件特性与执行场景灵活调整，平衡效果与实现成本。

参考文献

[1]于涛,王珊珊,徐芊卉,等.基于下推自动机的同步数据流语言可信编译[J].软件学报,2025,36(8):3554-3569.

[2]张明明,张富林,刘建戈,等.基于深度学习的编译型语言代码转换技术研究[J].计算机技术与发展,2026,36(1):24-30.

[3]陆炜,李伟津,施彬彬,等.LLMCfuzz:基于大语言模型的航空发动机编译器模糊测试方法[J].计算机学报,2025,48(12):2875-2892.

[4]张忠坤,林泓宇,谭智元,等.图形化PDDL语言编译系统的设计与应用[J].计算机工程与设计,2024,45(2):626-632.

[5]张磊,李响,陈宏君,等.面向国产处理器的ST语言保密编译方案[J].工业控制计算机,2025,38(6):10-11,14.