软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(6):2631−2647 [doi: 10.13328/j.cnki.jos.007097]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

位宽感知的寄存器绑定算法*

高 猛[1,2], 赵家程[1,2], 崔慧敏[1,2], 冯晓兵[1,2]

[1](中国科学院 计算技术研究所, 北京 100190)

[2](中国科学院大学, 北京 100049)

通信作者: 赵家程, E-mail: zhaojiacheng@ict.ac.cn

摘 要: 寄存器绑定是高层次综合中的一个基础优化问题, 主要目标是在保证电路功能的同时最小化寄存器资源的使用. 传统的方法尝试将编译器的寄存器分配算法应用于寄存器绑定中, 但却忽略了分配问题与绑定问题的差异性, 因此在绑定过程中引入了额外的资源约束, 或采用了不适合电路设计的编译优化技巧, 从而导致资源浪费. 为解决这些问题, 将寄存器绑定问题转化为连续多重着色问题, 并提出一种基于位宽与顶点度结合的启发式求解方法. 所提方法通过对变量的位宽和活跃区间等信息的细粒度刻画和建模, 能够进一步优化寄存器资源的开销, 同时无需插入额外的指令. 将该算法与两种典型算法进行比较, 实验结果表明, 所提算法在 MiBench 测试集的 96.72% 的测试用例中达到理论最优解, 比其他两种方法分别提高 31.5% 和 25.1%; 在 Rosetta 测试集的所有测试用例中均表现为最优解, 比其他两种方法分别提高 7.41% 和 7.39%.

关键词: 高层次综合; 寄存器绑定; 资源共享

中图法分类号: TP314

中文引用格式: 高猛, 赵家程, 崔慧敏, 冯晓兵. 位宽感知的寄存器绑定算法. 软件学报, 2024, 35(6): 2631–2647. http://www.jos.org.cn/1000-9825/7097.htm

英文引用格式: Gao M, Zhao JC, Cui HM, Feng XB. Bitwidth-aware Register Binding Algorithm. Ruan Jian Xue Bao/Journal of Software, 2024, 35(6): 2631–2647 (in Chinese). http://www.jos.org.cn/1000-9825/7097.htm

Bitwidth-aware Register Binding Algorithm

GAO Meng[1,2], ZHAO Jia-Cheng[1,2], CUI Hui-Min[1,2], FENG Xiao-Bing[1,2]

[1](Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Register binding is a fundamental optimization problem in high-level synthesis, primarily aiming to minimize the number of employed registers while ensuring circuit functionality. Conventional approaches attempt to apply register allocation algorithms from compilers to register binding, but they often overlook the inherent disparities between the allocation and binding problems. Finally, this introduces additional resource constraints during the binding and utilizes compilation optimization techniques that are unsuitable for digital circuit design, causing resource waste. To this end, this study transforms the register binding problem into a continuous multicoloring one and proposes a heuristic solving method that combines the bit width and vertex degrees. This method provides a more fine-grained characterization and modeling of variables in information such as the bit width and live intervals, which further reduces register resource overhead compared to existing methods, without the insertion of additional instructions. Meanwhile, a comparison is conducted between the proposed algorithm and two typical algorithms. Experimental results demonstrate that the proposed algorithm yields a theoretically optimal solution in 96.72% of the test cases in the MiBench benchmark, improving 31.5% and 25.1% over other methods respectively. In the Rosetta benchmark, this algorithm consistently exhibits the optimal solution across all test cases, outperforming the other two methods

* 本文由“编译技术与编译器设计”专题特约编辑冯晓兵研究员、郝丹教授、高耀清博士、左志强副教授推荐.
收稿时间: 2023-09-10; 修改时间: 2023-10-30; 采用时间: 2023-12-14; jos 在线出版时间: 2024-01-05
CNKI 网络首发时间: 2024-03-23

by 7.41% and 7.39% respectively.

Key words: high-level synthesis; register binding; resource sharing

近年来, 由于摩尔定律[1]的逐渐失效, 晶体管规模扩展的成本正以指数级的方式变得更加昂贵[2], 从前主要依赖晶体管的规模扩大来提升硬件性能的方法也逐渐变得难以实现. 为了继续提高程序性能, 新的体系结构设计开始向特定领域架构 (DSA) 方向发展[3]. 这种转变也导致了当前超大规模集成电路设计 (VLSI) 的流程发生了重要的变化, 特别是对于专用硬件加速器的设计. 为了实现高效快速的硬件加速器设计, 采用高层次综合 (HLS) 作为基于低级硬件描述语言 (例如 Verilog 或 VHDL) 的传统 VLSI 设计的替代方案[2,4]开始逐渐获得关注.

高层次综合 (high-level synthesis, HLS) 是一种将高级语言描述的逻辑结构自动转换成硬件描述语言表达的电路模型的技术. 传统的 HLS 过程主要分为 3 个基本步骤: 调度, 分配与绑定[5]. 其中调度过程将程序中的每个操作分配给特定的时钟周期, 并生成有限状态机, 以确保操作的顺序和时间满足计算需求. 分配过程确定可供使用的硬件资源数量, 包括处理器单元, 存储器等, 并为每个操作分配适当的资源. 绑定过程则通过在操作之间共享功能单元以及在变量之间共享寄存器或存储器来节省面积, 从而实现资源的高效利用. 在给定的计算任务和硬件资源约束下, HLS 通过将计算任务映射到可用的硬件资源上, 以实现最优的性能和资源利用率[6,7].

本文主要研究寄存器资源绑定问题, 即如何将程序的输入, 输出和临时变量映射到可用的寄存器单元上, 以最小化所需寄存器资源的问题. Brisk 等人[8]证明了静态单赋值 (SSA) 形式的程序的寄存器绑定问题可以在多项式时间内计算出最优解. 学术界的进一步研究工作可以分为两个不同的方向: (1) 通过与电路设计流程中的其他问题协同研究, 寻找电路设计的全局最优解. 典型的如 Rim 等人[9]将布图规划问题纳入绑定问题中进行协同考虑, 从而探索全局最优的布局与绑定方案. (2) 在寄存器绑定问题中引入位宽考量, 由于 Brisk 等人[8]的研究限制了寄存器的大小必须完全一致, 而在电子设计领域, 寄存器可以具有不同的长度. 通过消除电路中的冗余位宽, 可以降低电路的资源消耗, 这为进一步优化寄存器绑定问题提供了潜在的机会和可能性. 如 Kum 等人[10], Cong 等人[11]在传统图着色算法与团划分算法[12]中加入位宽选择策略, 实现位宽感知的寄存器绑定算法研究, Canesche 等人[13]通过插入额外的复制和交换指令进行数据交换, 以减少寄存器资源的开销.

寄存器绑定问题和寄存器分配问题存在着诸多的相似性, 两个问题的核心思路都是依据变量的活跃信息实现寄存器共享, 这也启发现有的方法[11]将寄存器分配的图着色抽象引入到解决寄存器绑定当中. 但是两个问题存在着一个明显的差异: 寄存器分配的目标是在寄存器数量固定的前提下将变量映射到寄存器中, 而寄存器绑定则需要生成所需的寄存器和连线. 这种差异性使得直接将寄存器分配的图着色抽象应用于寄存器绑定问题时可能会引发多种问题. 首先, 寄存器分配问题的抽象会给寄存器绑定问题引入不必要的约束限制, 如 Cong 等人[11]将寄存器绑定问题映射为加权区间图着色问题 (weighted-interval-graph coloring, WIGC) 进行求解, 在此过程中需要对变量进行对齐限制, 这增加了变量的位宽和活跃区间, 从而导致了寄存器空间的浪费; 其次, 寄存器分配中的一些优化技巧并不适用于电路设计, 如 Canesche 等人[13]的算法在寄存器绑定过程中引入了寄存器分配中常见的活跃区间分割技巧, 然而, 在硬件中频繁地移动数据会增加电路控制逻辑的复杂度, 反而可能会增加综合后电路的面积.

针对上述问题, 本文提出一种针对寄存器绑定问题的多重图着色抽象. 由于 HLS 可以根据绑定生成所需的寄存器与对应连线, 无需划分为寄存器组, 因此不需要遵循对齐等寄存器分配问题中常见的约束. 针对这些特性, 我们将寄存器绑定问题映射为连续多重着色 (consecutive multi-coloring, CMC) 问题, 对变量的位宽和活跃区间进行细粒度建模. 在此基础上, 我们权衡了权重与顶点的度的双重影响, 提出一种基于优先度的启发式算法, 该算法无需插入额外的复制与交换指令来进行数据交换, 并能够进一步减少寄存器数量. 本文的主要贡献包含以下 3 个方面.

• 我们将寄存器绑定问题映射为连续多重着色问题, 并提出一种在多项式时间内计算理论下界的方法. 与传统方法中将该问题映射为加权区间图着色问题不同[11], 我们的方法无需引入额外的对齐约束, 更准确地建模了寄存器绑定问题中的变量位宽和活跃区间信息, 从而可以实现更节省寄存器资源的分配方案.

• 为了求解连续多重着色问题, 我们提出一种基于顶点的度与权重感知的图着色算法 (CMC-h 算法). 具体而

言, 算法首先使用 CMC 问题理论下界对应的分配方案作为初始方案. 然后, 通过考虑顶点的邻接关系和变量的位宽, 计算每个顶点的优先度. 最后, 按照优先度的顺序进行求解, 以实现高效的寄存器绑定.

• 我们在 LLVM 编译器[14]中实现了上述算法, 并在 MiBench[15]与 Rosetta[16]测试集上对算法进行了实验评估. 评估结果表明, 在 MiBench 测试集中 CMC-h 算法的绑定结果接近理论最优解, 在 96.72% 的样例中 CMC-h 的绑定方案是最优的, 相比于 Cong 算法[11]和 Swap 算法[13]分别将这一比例提高了 31.5% 和 25.1%; 在 Rosetta 测试集中, CMC-h 的绑定方案在所有测试样例中均表现为最优解, 而 Swap 算法和 Cong 算法仅在 93.10% 和 93.12% 的样例中达到最优解.

1 相关工作

本节旨在详细介绍与寄存器绑定问题相关的研究工作. 寄存器绑定是高层次综合过程中一项基础优化问题, 其也可视为寄存器分配问题的一种变体. 首先, 我们将介绍高层次综合过程中相关的优化方法和常用工具. 随后, 我们将探讨寄存器绑定问题与寄存器分配问题以及寄存器打包问题之间的相似性和差异性, 并评估各种算法在解决这些问题时的适用性. 最后, 我们还将讨论位宽分析在寄存器绑定问题中的重要影响.

1.1 高层次综合

高层次综合 (HLS) 是一种将高级语言描述的逻辑结构自动转换成硬件描述语言表达的电路模型的过程. 该技术已经成为一种关键的使能技术, 尤其对于 FPGA 设计而言. 通过 HLS, 设计者可以在软件层面上描述组件的功能, 并自动生成相应的硬件代码, 从而实现软硬件解决方案的快速部署. 相较于传统的电路设计方法, HLS 使得设计者不必了解电路设计的细节, 从而可以将注意力更多地专注于算法逻辑结构的设计与设计空间的探索上. 已有的研究表明, HLS 可以提高 FPGA 设计的开发效率和可维护性[2,4,17].

传统的 HLS 过程主要分为 3 个基本步骤: 调度, 分配与绑定[5]. 其中分配过程确定可供使用的硬件资源量, 调度过程将程序中的每个操作分配给特定的时钟周期, 并生成有限状态机. 绑定过程通过在操作之间共享功能单元以及在变量之间共享寄存器或存储器来节省面积. HLS 在给定的计算任务和硬件资源约束下, 将计算任务映射到可用的硬件资源上, 以实现最优的性能和资源利用率. 这 3 个过程相互影响, 根据解决它们的顺序, 每个问题都有不同的解决方案. 其中绑定问题通常是在分配和调度之后制定和解决的[6,7].

目前, HLS 工具已经在某些情况下能够生成与手工优化电路相媲美的电路质量[2]. 经过多年的发展, 已经涌现出超过 30 种不同的 HLS 工具[18], 其中多个工具基于开源编译器框架 LLVM/GCC 等进行开发. 例如, 多伦多大学研发的 LegUp[19]利用 LLVM 编译器作为前端编译器, 能够将输入的 C 语言自动转换为适用于 FPGA 的 Verilog 语言. 另一个开源工具 Bambu[6]也采用 LLVM/GCC 的标准 IR 作为输入, 实现了广泛的转换和优化范围, 为 EDA 领域的研究人员提供了探索和集成新方法的机会.

1.2 寄存器绑定问题关联问题

寄存器绑定问题通常被拿来与寄存器分配问题和寄存器打包问题相互比较. 寄存器绑定对电路中的寄存器资源进行共享, 提高资源的利用率. 寄存器分配确定如何将程序中的变量分配给有限数量的寄存器, 以最小化内存访问次数并提高执行效率. 而寄存器打包则涉及如何将变量组合在一起, 并分配给不同的寄存器集合, 以最大程度地利用寄存器容量. 虽然这些问题在具体细节上存在差异, 但它们都涉及优化寄存器的使用, 因此在解决某个特定问题时, 可以借鉴其他关联问题的方法进行求解.

寄存器分配问题通常被认为是一个 NP 完全问题, 因此在实际应用中常采用图着色算法进行求解[20]. 然而, 在 2005–2006 年间, 基于静态单赋值 (SSA) 形式的程序, 寄存器分配问题被证明可以在多项式时间内获得最优解[21−23]. 寄存器绑定问题与寄存器分配的主要区别在于: 寄存器绑定问题中的寄存器大小是任意长度, 相较于通常的架构, 对寄存器的访问更加灵活[24]. 因此, 若采用传统的寄存器分配算法来解决寄存器绑定问题, 则会引入过多传统架构的限制, 导致绑定的结果较差.

寄存器打包问题试图通过变量合并的方式[25]将多个变量分配到同一寄存器的不同位中, 从而解决可变位宽

的寄存器分配问题[26−28]. 该问题限制了寄存器组的大小为固定的大小. 尽管与寄存器绑定问题存在一定的相似性, 但在传统硬件设计领域中, 将多个变量分配到同一寄存器的不同位中会导致物理寄存器布局变得极为复杂, 同时也会增加物理寄存器的寻址和存储延迟. 因此, 这种技术始终受到一定程度的限制[28]. 当将寄存器打包算法应用于寄存器绑定问题时, 实际的分配效果可能稍逊于 Cong 等人提出的方法[11,27].

1.3 位宽分析

位宽分析作为位宽感知的寄存器绑定算法的基本组成部分, 主要利用静态分析技术推断变量的位宽[29−31]. 传统高级语言如 C/C++等描述电路的逻辑结构存在很大局限性, 因为这些语言只提供具有受限长度的计算类型, 例如固定位宽的 8/16/32/64 等. 研究显示, 在一组用 C 编写的基准程序中, 平均有 40% 的位宽冗余[29], 识别出这些多余的比特可以降低潜在的硬件资源成本. 经验公式[32−34]表明, 乘法器的面积与两个输入的位宽乘积成正比, 加法器的面积则与输入的位宽成正比. 对于寄存器和多路复用器等其他资源类型, 可以采用合理的线性假设来估计它们的面积和位宽之间的关系. 由于操作位的减少也可以带来布线成本的节省, 因此通过优化电路的位宽分配, 还可以有效地减少电路的面积和功耗. 然而, 从传统物理意义范畴出发为硬件设计明确指定比特宽度费时费力. 因此更理想的设计流程是自动分析变量和操作的位宽, 以减轻设计者的负担并减少出错的机会. 本文的寄存器绑定算法未限制位宽分析的具体实现方式, 因此, 任何实现方式都可以满足我们的需求.

当前编译器主要从范围分析与位掩码分析两种层面对变量位宽进行分析. Stephenson 等人[29]提出了基于变量范围分析的位宽分析技术, 该方法试图通过变量范围的正向与反向传播来估计变量的范围, 从而实现对变量的高位截断技术. Gort 等人[35]提出了掩码分析与范围分析相结合的位宽分析技术, 使得位宽分析可以针对变量的每个比特进行精确的分析. 这两种分析形式提供了不同层面的互补信息, 因此需要综合考虑[35].

2 背景知识与研究动机

本节主要对相关背景知识与研究动机进行梳理, 首先我们对后文将要涉及的图论相关知识进行简单介绍. 然后我们给出本文所关心的寄存器绑定问题的定义与典型样例. 我们首先介绍一种寄存器绑定问题的经典抽象方法加权区间图着色 (WIGC) 抽象[11]. 接着我们介绍一种通过插入额外的拷贝指令在多项式时间内给出最优解的优化算法[13]. 最后我们通过样例分析指出了两种方法中存在的寄存器浪费现象, 并由此引出我们新的设计.

2.1 背景知识

本节总结了弦图理论中的相关概念. 对于无向图 $G(V,E)$, 图上的一个环中不相邻的两个顶点相连的边称为弦. 给定无向图 $G(V,E)$, 如果图中任何一个长度大于 3 的环至少有一条弦, 则称 G 是弦图.

我们使用 $N(v)$ 表示与顶点 v 相邻的顶点集合, 若一个图的子图顶点两两相邻, 则称该子图为团, 我们记顶点数最多的团为最大团, 其顶点数记为团数 $\omega(G)$. 如果顶点 $v\in V$ 的邻域 $N(V)\cup\{v\}$ 是一个团, 则称该点为单纯点. 我们称一个点的序列 $v_1,v_2,\ldots,v_n$ 为一个完美消除序列 (PEO), 当且仅当该序列满足顶点 v_i 在 $v_i,v_{i+1},\ldots,v_n$ 的诱导子图中为一个单纯点.

给定一些区间, 用每个顶点表示一个区间, 若两个区间的交集非空, 则代表两区间的顶点间存在一条边, 由此构成的图称为区间图. 关于弦图与区间图, 有如下几个重要的结论[23].

定理 1. 无向图 $G(V,E)$ 是弦图当且仅当 G 存在完美消除序列.

定理 2. 弦图 G 的色数 $\chi(G)$ 等于团数 $\omega(G)$.

引理 1. 区间图必定为弦图.

引理 2. 弦图 G 按照 PEO 逆序贪心着色, 该方案的着色数等于 $\chi(G)$.

引理 3. 区间图按顶点对应区间的右端点从小到大排序, 该序列为完美消除序列.

2.2 寄存器绑定问题定义

由于本文关注的问题是调度后的寄存器绑定问题, 因此如无额外说明, 假定后文提及的每个输入程序都是满足 SSA 形式的线性结构程序, 且本文中的寄存器绑定问题不涉及指令间的调度. 以下给出寄存器绑定问题定义.

定义 1. 寄存器绑定问题. 给定程序 P 的变量集合 V, 及其权重映射 $w: V \to \mathbb{N}$, 假设每个变量 $v \in V$ 都存在对应活跃区间 l_v, 映射 I 为每个变量 $v \in V$ 分配一个区间 $I(v)$, 且满足如下约束:

a) (区间约束) $\forall v \in V, |I(v)| = w(v)$.

b) (冲突关系) $\forall a,b \in V$, 若 $l_a \cap l_b \neq \varnothing$, 则 $I(a) \cap I(b) = \varnothing$,

其中, $\mathbb{N}$ 代表自然数集, 权重映射 w 代表每个变量的位宽. 该定义与 Cong 等人提出的绑定问题定义[11]是一致的, 但这里强调了绑定方案 I 会单独考虑变量的每个比特分配情况. 与传统的寄存器分配问题将每个寄存器视为固定的位宽不同, 寄存器绑定问题允许根据变量的位宽灵活地调整寄存器的位宽. 这种灵活性有助于减少所需的寄存器数量, 从而最大程度地利用寄存器资源.

例 1: 图 1 左侧是一段简单的程序源码示例, 在该样例中我们忽略调度等其他 HLS 过程, 图 1 右侧对该示例的寄存器绑定问题进行了更加详细的展示[13]. 图的横向每个方框表示 1 比特位, 例如变量 a 需要至少 5 位的寄存器; 图的纵向的每条横线标记了不同的程序点, 用于表示变量的活跃区间. 每个变量的位宽与活跃区间构成了一个矩形, 因此, 可以简化地说寄存器绑定问题的解决方案应尽可能紧密地打包所有的矩形, 而不允许重叠.

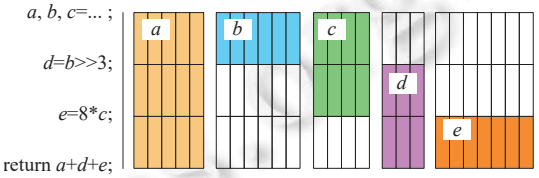


图 1 样例例源码与变量位宽-活跃区间示意图

2.3 加权区间图着色抽象

Cong 等人[11]将寄存器绑定问题转化为加权区间图着色问题进行求解, 从而允许寄存器以任意位宽进行绑定, 我们首先给出其基本定义.

对于给定程序 P, 可以导出区间图 $G(V,E)$. 其中每个顶点 $v \in V$ 对应于程序中的变量, 边 $(a,b) \in E$, 当且仅当两个顶点 $a,b \in V$ 对应变量的生命周期相互重叠, 即它们不能绑定到同一个寄存器中. 给图 G 的每个顶点 v 赋予一个权重 $w(v)$ 表示对应变量的位宽, 我们称该图为加权图, 并记为 $G(V,E,W)$. 独立集是指图中互不相邻的顶点构成的集合. 假设 $C = \{c_1, c_2, \ldots, c_k\}$ 是图 $G(V,E)$ 的某一着色方案, 该方案将顶点集合 V 划分为 k 个独立集 $V_1, V_2, \ldots, V_k$, 其中 V_i 表示第 i 个独立集, c_i 表示 V_i 中的变量所着的颜色. 若定义独立集 S 的权重为 $w(s) = \max\{w(v) | v \in S\}$, 颜色 c_i 的权重定义为 $w(V_i)$, 则着色方案 C 的权重可以定义为:

$$\Phi(G,C) = \sum_{i=1}^{k} w(V_i).$$

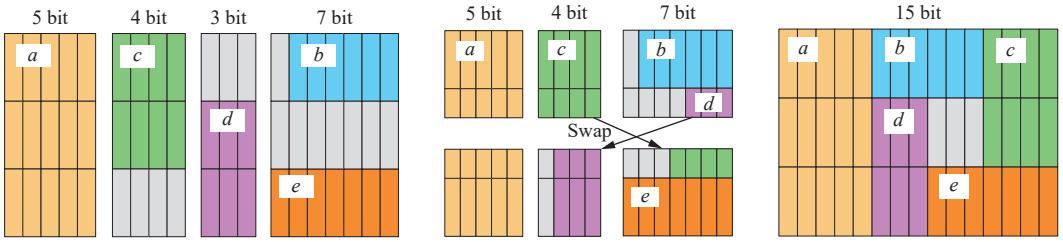
基于以上定义, 位宽感知的寄存器分配问题可以映射为加权区间图着色问题, 定义如下.

定义 2. 加权区间图着色问题. 给定加权区间图 $G(V,E,W)$, 求解使得权重函数 $\Phi(G,C)$ 最小化的着色方案 C.

WIGC 问题在某些研究工作中也被称为区间图的最大着色问题 (max-coloring problem)[36]. 当顶点的权重一致时, 这个问题可以在多项式时间内解决, 然而对一般问题而言, 该问题已被证明是 NPC 问题[36,37]. WIGC 问题的理论下界可以通过子图的色数进行估计. 我们将图 G 按照顶点的权重划分子图, 并按权重从大到小排序为 $G_1, G_2, \ldots, G_m$, 子图 G_i 的顶点对应的权重记为 w_i, 记 $U_i = \bigcup_{j=1}^{i} G_j$, $k_i = \chi(U_i) - \chi(U_{i-1})$, $k_1 = \chi(G_1)$. 则 WIGC 下界可以通过如下公式进行估计:

$$L(G) = \sum_{i=1}^{m} w_i \cdot k_i.$$

Cong 等人[11]提出了一种基于 WIGC 抽象的启发式算法, 我们称之为 Cong 算法. 该算法首先将变量按位宽降序排列, 通过启发式方法将位宽相近的变量合并装入同一个寄存器中进行分配. 图 2(b) 显示了 Cong 算法对例 1 的解决方案. 在该方案中, 变量 b 和变量 e 共享一个 7 位寄存器, 而其余变量则分别使用与它们的位宽相同的寄存器, 总共需要分配 19 位寄存器.



(a) Cong 算法 (共 19 bit) (b) Swap 算法 (共 16 bit) (c) 最优方案 (共 15 bit)

图 2 3 种寄存器绑定方案对例 1 产生的分配结果示意图

2.4 Swap 算法

Canesche 等人[13]在 WIGC 抽象的基础上, 进一步提出了通过插入复制和交换指令来降低寄存器数量的方法, 我们称之为 Swap 算法.

Swap 算法首先在每个基本块中的连续指令之间插入并行拷贝, 从而将每个变量的活跃区间限制在了相邻的两条指令之间, 不同程序点间的活跃变量相互独立. 接着独立地解决每个程序点上的加权区间图着色问题. 最后, Swap 算法尝试合并多个程序点不同着色方案, 并给出的最优拼接方案. Swap 算法对例 1 的解决方案需要在第 2 和第 3 条指令之间插入交换指令, 以便将变量 c 和 d 的活跃区间分割开来. 这样, 变量 c 和 d 的部分活跃区间可以与变量 b 和 e 共享一个 7 位寄存器, 而其余活跃区间可以共享一个 4 位寄存器. 该方案总共需要分配 16 个寄存器. 与 Cong 算法方法相比在比特位宽上共减少了 3 位, 但同时需要增加一条交换指令.

Swap 算法以插入额外指令为代价减少了绑定的寄存器数量, 它是 WIGC 抽象的一种改进形式, 由于过程中的每一步算法均是最优解, 因此 Swap 算法是一种多项式时间的最优解算法. 但 Swap 算法的最优性限制在该算法设置的对齐约束之中, 对于整个寄存器绑定问题的求解并非最优, 我们将在第 2.5 节中详细阐述.

2.5 研究动机

本节通过样例分析仔细研究了 Cong 算法与 Swap 算法可能造成的寄存器浪费现象, 并由此指出已有算法的局限性. 我们将在第 3 节中通过引入新的抽象来解决这些问题.

为了使接下来的分析更加清晰明确, 我们将绑定算法中寄存器的空闲状态定义为寄存器空泡, 并通过对寄存器空泡的分析来评估已有算法的分配方案. 一般而言主要寄存器空泡包括两种不同的状态, 一是变量位宽与寄存器大小不匹配产生的位宽冗余, 二是由于冲突关系造成的寄存器空置. 图 2 中的灰色区域展示了不同绑定方案产生的寄存器空泡, 通常我们认为寄存器空泡的范围越大, 造成寄存器浪费的可能性也就越高.

Cong 算法将可能同时产生两种类型的寄存器空泡. 如图 2(a) 所示, 变量 b 与变量 e 合并时, 6 bit 变量 b 存储在 7 bit 寄存器中, 产生了 1 bit 的位宽冗余; 变量 b 的活跃区间结束, e 的活跃区间开始之前, 寄存器处于空闲状态且无法被其他变量复用, 从而产生了寄存器空置. 因此 Cong 算法在 3 种绑定方案中产生了最多的寄存器空泡.

Swap 算法通过插入交换与并行拷贝指令, 解决了寄存器空置产生的寄存器空泡问题, 但却仍然存在两点缺陷. 其一, Swap 算法需要合并多个子着色方案, 在合并过程中会扩大合并的寄存器大小, 因此增加了位宽冗余产生的寄存器空泡. 如图 2(b) 所示的 4 bit 寄存器, 是由子着色方案的 4 bit 寄存器与 3 bit 寄存器合并而成, 因此造成了额外的位宽冗余. 其二, 拷贝指令产生额外代价. Swap 算法频繁地移动数据会使得电路的控制逻辑变得更为复杂. 如图 2(b) 所示, 算法插入了额外的交换指令, 这可能会增加多路复用器的使用, 反而导致综合后的电路面积更大, 我们将在第 4.5 节详细阐述这一矛盾.

已有绑定算法产生大量寄存器空泡的根本原因在于对寄存器的利用存在不合理的限制. 与寄存器分配问题不同, 寄存器绑定问题中, 寄存器的连线可以根据绑定结果生成, 无需遵循通用可编程寄存器常见的对齐约束. Cong 算法采用的 WIGC 抽象, 仅允许同一独立集内的变量共享同一个寄存器, 为寄存器绑定问题引入了额外的对齐约束. Swap 算法虽然对 WIGC 抽象有所改进, 但仍未能够完全消除对齐约束带来的影响. 如图 2(c) 所示显示了一种

最优的绑定方案, 该方案将寄存器空间视为一个连续的线性空间, 任何访问都通过一段区间来表示, 不同的变量共享同一个寄存器的不同比特位. 在该分配方案中, 变量 b, c, d, e 错位地共享了一部分寄存器, 共需 15 位寄存器, 相比于 Swap 算法进一步减少了 1 位寄存器, 且无需插入额外的指令. 这是 WIGC 抽象方法所无法实现的寄存器绑定方案. 因此我们需要提出一种新的抽象, 该抽象能够更加自由地描述变量之间的共享关系, 且不会产生额外的寄存器约束.

3 图的连续多重着色抽象

在本节中, 为了解决已有算法中存在的问题, 我们引入了图的连续多重着色模型. 首先, 我们给出了连续多重着色问题的明确定义. 接下来, 我们对连续多重着色问题的理论下界进行了详细分析, 以作为评估和设计我们算法的依据. 最后, 为了实际解决寄存器绑定问题, 我们提出了一种基于优先度的启发式算法, 该算法不需要插入额外的拷贝指令, 且相比于已有算法能够进一步减少寄存器的数量.

3.1 连续多重着色抽象

根据第 2.5 节的分析结果, 我们提出一种图的连续多重着色抽象来对寄存器绑定问题进行求解. 为了更清晰地阐述我们的思路, 我们首先引入图的多重着色问题 (multi-coloring, MC) 的定义.

定义 3. 图的多重着色问题. 给定加权图 $G(V,E,W)$, 对每个顶点 $v \in V$ 染 $w(v)$ 种不同颜色, 记颜色集合为 $C(v)$, 求解符合条件的着色方案, 使得对于任意边 $(a,b) \in E$, 都有 $C(a) \cap C(b) = \varnothing$ 成立.

我们记 MC 问题的最小色数为 $\gamma(G)$. MC 问题强调了我们对于寄存器绑定问题中变量之间更为灵活的共享方式的建模, 它可以很好地描述比特粒度的寄存器共享问题. 在 MC 问题中, 每个顶点 v 的颜色集合 $C(v)$ 可以视为按比特粒度分配的寄存器, 而每个边 (a,b) 则可以视为两个变量 a,b 的访问冲突. 但是, MC 问题并不能实际地解决寄存器绑定问题, 因为该问题中的颜色集合 $C(v)$ 是离散的, 而寄存器的访问则是连续的, 无法满足寄存器绑定问题中的区间约束.

为了实际地解决寄存器绑定问题, 我们进一步定义颜色集的连续性. 我们称颜色集合 $C(v)$ 是连续的, 当且仅当存在某个 $k \in \mathbb{N}$, 使得 $|C(v) \cap [k, k+w(v))| = w(v)$. 在此基础上, 我们提出了图的连续多重着色问题, 其定义如下.

定义 4. 图的连续多重着色问题. 给定加权图 $G(V,E,W)$, 求解满足条件的多重着色方案, 使得任意顶点 $v \in V$ 的颜色集合 $C(v)$ 是连续的.

连续多重着色问题在某些研究工作中也被称为区间着色 (interval coloring) 问题[37]. 该问题与寄存器绑定问题是等价的, 因为寄存器绑定问题中的程序 P 可以一一映射为 CMC 问题中的加权图 $G(V,E,W)$, 其中变量集合映射为图的顶点集合, 变量的位宽映射为顶点权重, 变量的冲突关系映射为图的边, 寄存器集合映射为颜色集合, 区间约束则映射为颜色集合的连续约束. 我们记 CMC 问题的最小色数为 $\gamma^*(G)$.

CMC 抽象精确描述了每个变量分配在寄存器中的位宽与活跃区间, 无需遵循额外的对齐约束. 对于任意一个 WIGC 问题的解 C, 总是可以通过将 C 中的每个寄存器拼接成为一个大寄存器得到一个 CMC 问题的解, 反之 CMC 问题的解却不一定是 WIGC 问题的解. 因此 CMC 抽象相比于 WIGC 抽象更加契合寄存器绑定问题, 能够更加灵活地描述变量之间的复用关系.

图 2(c) 显示了 CMC 抽象的一种多重着色方案. 在该着色方案中, 变量 a 被分配到了寄存器 $R[14{:}10]$, 变量 b 被分配到了寄存器 $R[9{:}4]$, 变量 c 被分配到了寄存器 $R[3{:}0]$, 变量 d 被分配到了寄存器 $R[9{:}7]$, 变量 e 被分配到了寄存器 $R[6{:}0]$. 该着色方案的色数为 15, 而 WIGC 抽象对该样例限定的理论下界范围为 $L(G) = 16$, 因此 CMC 抽象突破了 WIGC 抽象限定的理论下界.

3.2 理论下界分析

即使对于区间图, CMC 问题仍然已被证明是 NPC 问题[36,37], 因此我们无法在多项式时间内给出该问题的解. 但作为参考, 在我们可以在多项式时间内给出该问题的理论下界.

由于 CMC 问题是 MC 问题的特例, $\gamma(G) \leqslant \gamma^*(G)$, 因此, MC 问题的最优解即为 CMC 问题的理论下界. 对于

SSA 形式的线性结构程序 P, 其对应的干涉图为区间图, 对于区间图我们可以有更多成熟的结论.

引理 4. 区间图的 MC 问题多项式时间可解.

证明: MC 问题可以转化为图着色问题进行求解, 只需将每个顶点转化为对应色数的完全子图, 从而转换为传统图着色问题进行求解. 根据引理 1, 区间图必定是弦图, 转换后映射到对应图着色问题中也必定为弦图. 因此区间图的 MC 问题可以转化为弦图的图着色问题. 根据定理 2, 该问题必定在多项式时间内可解. 因此, 区间图的 MC 问题也可以在多项式时间内求解.

我们给出无需构建干涉图的理论下界算法, 如算法 1 所示, 算法在第 2 行按照变量活跃区间的末端从大到小排序, 按照引理 3, 该顺序必定为对应区间图的完美消除序列的逆序. 我们在第 5 行对变量的每个比特进行编号, 从而将对应加权区间图转化为区间图, 对区间图的每个顶点分别进行着色, 其中每个顶点对应变量的每个比特. 算法第 8–21 行按照完美消除序列逆序对区间图的每个顶点进行着色. 其中第 13–15 行利用了完美消除序列的性质, 在无需构建干涉图的情况下仍然能够维护每个顶点 v 的邻居节点 $N(v)$ 的已着色情况. 该算法的时间复杂度为 $O(|V|+|E|)$, 其中排序算法的时间复杂度通常为 $O(|V|\log|V|)$, 但在该问题中, 可以利用桶排序算法加速到 $O(|V|)$ [23].

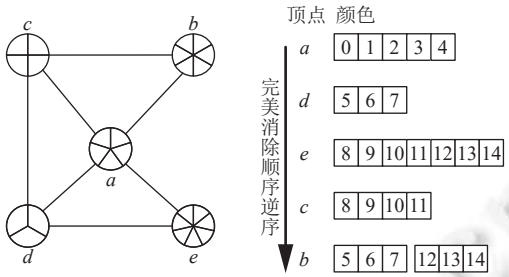
算法 1. 理论下界分析.

Input: 变量列表 V, 每个变量 $v \in V$ 的位宽 W_v 与活跃区间 I_v;
Output: 理论下界 L.

1. **function** $LowerBound(V, W, I)$
2. $Sort(V, key \leftarrow lambda\ \ v : I_v.end, reversed \leftarrow \text{True})$
3. $M \leftarrow 0$
4. **for** $v \in V$ **do**
5. $index_v \leftarrow [M, M + W_v) \cap \mathbb{N}$
6. $M \leftarrow M + W_v$
7. **end for**
8. $freeColor_i \leftarrow \text{True}, \forall i \in [0, M) \cap \mathbb{N}$
9. $Color_v \leftarrow \varnothing, \forall v \in V$
10. $lastEnd \leftarrow I_{V_0}.end$
11. $kill_i \leftarrow \varnothing, \forall i \in [0, lastEnd) \cap \mathbb{N}$
12. **for** $v \in V$ **do**
13. $kill_{I_v.start} \leftarrow kill_{I_v.start} \cup \{v\}$
14. $freeColor_{color_i} \leftarrow \text{True}, \forall i \in index_u, u \in kill_k, k \in [I_v.end, lastEnd] \cap \mathbb{N}$
15. $lastEnd \leftarrow I_v.end$
16. **for** $i \in index_v$ **do**
17. $j \leftarrow \min\{k | freeColor_k = \text{True}\}$
18. $Color_v \leftarrow Color_v \cup \{v\}$
19. $freeColor_j \leftarrow \text{False}$
20. **end for**
21. **end for**
22. $L \leftarrow \max(Color)$
23. **return** $L, Color, index$
24. **end function**

图 3(a) 显示了例 1 程序对应的干涉图, 图中的顶点 a, b, c, d, e 分别被等分为 5, 6, 4, 3, 7 份, 代表每个顶点可以

着多种不同的颜色. 图 3(b) 显示了按照完美消除顺序 $\{b,c,e,d,a\}$ 的逆序着色的一种多重着色的最优分配方案. 在该方案中, 变量 b 被分配给了离散的寄存器 $\{5,6,7,12,13,14\}$ 中, 不满足 0 的第 2 个约束, 即区间约束. 但若按照另一种完美消除顺序 b,c,a,d,e 的逆序进行着色, 其着色方案就如图 2(c) 所示, 为一种连续的多重着色方案.



(a) 多重着色干涉图 (b) 理论下界对应分配方案

图 3 图的多重着色干涉图与分配示意图

3.3 连续多重着色算法

为了得到实际有意义的寄存器绑定方案, 我们考虑直接对 CMC 问题进行求解. 由于我们目前难以在多项式时间内求解 CMC 问题的最优解, 因此我们提出了一种基于优先度的启发式方法, 称之为 CMC-h 算法 (如图 4). 它的基本思想是按照优先度顺序对顶点进行贪心着色, 其中优先度的设计对图着色问题中重要的分配依据顶点的度与 CMC 问题中顶点特有的特征顶点权重进行了综合考虑. 我们在第 4 节中实验证明了, 该算法的均衡考虑在实际求解中取得了优异的分配效果.

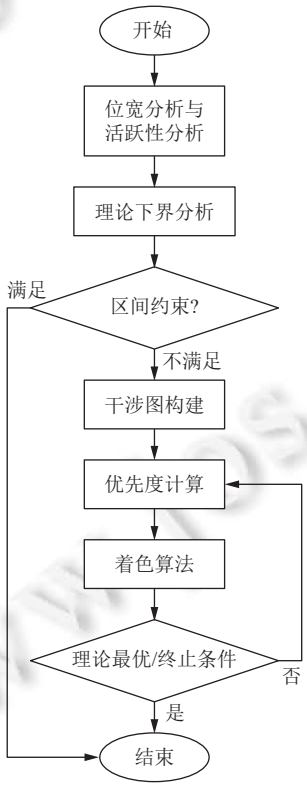


图 4 CMC-h 算法的一般流程

我们首先定义加权图 $G(V,E,W)$ 的度, 对任意顶点 $v \in V$, 我们定义顶点的度为其邻居节点的权重和, 即 $d(v) = \sum_{u \in N(v)} w(u)$, 其中 $N(v)$ 代表了顶点 v 的邻居节点集合, 由此我们定义顶点的优先度为:

$$r(v) = \alpha d^*(v) + (1-\alpha) w^*(v),$$

其中, $d^*(v), w^*(v)$ 为归一化后顶点的度与权重, $\alpha \in [0,1]$ 为参数.

我们按照变量的优先度顺序进行贪心着色, 算法 2 展示了 CMC-h 算法的具体过程. 算法的第 2 行, 我们首先对问题进行理论下界分析, 并得到初始的着色方案, 只有在该方案不连续的情况下, 我们才会采用启发式方法进行着色. 通常, 我们选择 $\alpha \in \{0, 0.5, 1\}$ 进行 3 次着色并选择最优结果, 我们将在第 4.3 节中进行详细论述. 算法的第 6 行, 我们计算了变量的优先度, 该优先度决定了顶点着色的顺序. 在这里我们对顶点的度与权重进行了归一化处理. 第 8–16 行代表了按照优先度顺序进行连续着色的过程. 与计算理论下界时有所不同, 由于无法通过完美消除序列快速计算当前顶点的已着色情况, 因此我们在第 4 行构建了完整的干涉图. 其中, 构建的干涉图为加权图, 需要为每个顶点赋予等于该顶点对应变量位宽的权重. 在第 11, 12 行需要每次重新计算顶点 的邻居节点 的已着色情况. 第 13, 14 行要求对同一顶点染多种连续的颜色, 而不是每个比特分别着色, 从而保证每个顶点的颜色集合均是连续的.

着色算法的时间复杂度为 $O(|V|+|E|)$, 排序算法的时间复杂度为 $O(|V|\log|V|)$, 构建干涉图的时间复杂度为 $O(|V|^2)$, 因此算法的整体复杂度仍然为 $O(|V|^2)$.

算法 2. CMC-h 算法.

Input: 变量列表 V, 每个变量 $v \in V$ 的位宽 W_v 与活跃区间 I_v;
Output: 着色方案 $Color$.

1. **function** $CMC-h(V,W,I)$
2. $L, Color, index \leftarrow LowerBound(V,W,I)$
3. **if** $\neg isContinue(Color)$ **then**
4. $G(V,E,W) \leftarrow BuildInterferenceGraph(V,W,I)$
5. **for** $\alpha \in \{0, 0.5, 1\}$ **do**
6. $r(v) = \alpha d^*(v) + (1-\alpha) w^*(v), \forall v \in V$
7. $Sort(V, key \leftarrow lambda\ v : r(v), reversed \leftarrow \text{True})$
8. $Color_v \leftarrow \varnothing, \forall v \in V$
9. $done \leftarrow \varnothing$
10. **for** $v \in V$ **do**
11. $freeColor_i \leftarrow \text{True}, \forall i \in [0, M) \cap \mathbb{N}$
12. $freeColor_i \leftarrow \text{False}, \forall i \in index_u, u \in N(v) \cap done$
13. $j \leftarrow \min\{k | freeColor_s = \text{True}, \forall s \in [k, k+W_v) \cap \mathbb{N}\}$
14. $Color_v \leftarrow Color_v \cup ([j, j+W_v) \cap \mathbb{N})$
15. $done \leftarrow done \cup \{v\}$
16. **end for**
17. **if** $\max(Color) = L$ **then**
18. **return** $Color$
19. **end if**
20. **end for**
21. **end if**
22. **return** $Color$
23. **end function**

4 实验评估

4.1 实验环境

本节将从如下几个角度分别对本文提出的工作进行评估.

1) 本文提出的连续多重着色抽象 (CMC 抽象) 相比于第 2 节介绍的 Cong 算法和 Swap 算法的区别是什么, 它们分别划定了怎样的问题求解范围.

2) 本文提出的启发式算法 (CMC-h 算法) 的实际分配效果如何, 与 Cong 算法和 Swap 算法相比较效果如何.

3) CMC-h 算法进行寄存器绑定需要多长时间.

4) CMC-h 算法计算的绑定方案是否可综合, 对逻辑综合会造成怎样的影响.

本文寄存器绑定算法应用的单元是一个函数, HLS 中调度与绑定通常被划分为独立的优化过程, 且寄存器绑定通常是在调度之后的资源绑定阶段进行, 因此调度算法的选择并不影响本文寄存器绑定算法的有效性. 为了模拟寄存器绑定的输入, 我们按照尽早调度原则 (ASAP) 进行调度[1], 然后再进行寄存器绑定算法的评估.

我们在 LLVM 11.0.0 版本中实现了本文提出的技术, 使用 LLVM 自带的位宽分析功能处理前置的分析工作. 同时使用了 MiBench[15]与 Rosetta[16]套件作为测试集, 如表 1 所示, MiBench 是一组共 35 个具有商业代表性的用于基准测试的嵌入式应用程序, 涉及工业自动化, 消费类设备, 办公, 网络, 安全与通信这 6 个不同的类别; Rosetta 是一组针对真实场景的 FPGA 的基准测试套件, 包括机器学习与视频处理等热门场景下的多个完全开发的应用程序. 实验在 Intel Xeon Gold 6248 CPU 20 核处理器上进行, 时钟为 2.50 GHz. 使用的操作系统是 Ubuntu Linux LTS 20.04 版本. 我们的实现将代码生成为 LLVM 机器代码中间表示 (MIR), 并使用 LLVM 中已经存在的寄存器分配基础设施进行分配. 我们并没有将这些结果映射到逻辑门中, 因为我们缺少必要的工具来做这件事. 然而作为一个概念的验证, 我们将一些分配结果手动翻译成了 Verilog 并进行综合分析, 我们将在第 4.5 节进行具体的解释.

表 1 MiBench 测试集[15]与 Rosetta 测试集[16]

测试集	测试场景	测试样例
MiBench	工业自动化	basicmath, bitcount, qsort, susan
	消费类	jpeg, lame, mad, tiff2bw, tiff2rgba, tiffdither, tiffmedian, typeset
	办公	ghostscript, ispell, rsynth, sphinx, stringsearch
	网络	dijkstra, patricia, CRC32, sha, blowfish
	安全	blowfish, pgp sign, pgp verify, rijndael, sha
	通信	CRC32, FFT, IFFT, ADPCM, GSM
Rosetta	3D渲染	integer arithmetics
	数字识别	Hamming distance, KNN voting
	垃圾邮件过滤	dot product, scalar multiplication, vector addition, Sigmoid function
	光流 (物体运动检测)	1D convolution, outer product
	二值神经网络 (BNN)	binarized 2D convolution, binarized dot product
	人脸检测	image scaling, cascaded classifiers

我们对 MiBench[15]中的 609 个函数与 Rosetta[16]中的 22 253 个函数进行了分析与测试. 表 2 总结了寄存器绑定的汇总结果, 其中运行时间以毫秒为单位, 寄存器大小以位为单位, “Geo”是几何平均值, “Avg”是算术平均值. 结果显示在 MiBench 测试集中 CMC-h 算法相比于 Swap 算法和 Cong 算法分别减少 1.8% 和 1.4% 的寄存器, 同时相比于最优解仅多使用 0.13% 的寄存器资源; 在 Rosetta 测试集中 CMC-h 算法相比于 Swap 算法和 Cong 算法分别减少 1.97% 和 1.98% 的寄存器, 同时在所有的测试用例中 CMC-h 算法均达到了最优解.

4.2 CMC 抽象的最优解评估

本节评估了 CMC 抽象的最优解相比已有方法的理论最优解的差别, 进而表明我们引入 CMC 抽象的必要性. 其中 CMC 问题的最优解通过对 CMC-h 算法未达到理论下界的样例分析, 采用整数线性规划方法 (ILP) 精确求解

得到, 在给出的测试集中我们暂未发现实际的最优解不等于我们给出的理论下界的样例, 但我们仍无法在理论上证明 $\gamma(G)=\gamma^*(G)$. Cong 算法的最优解我们目前无法准确求解, 但我们可以通过第 2.4 节给出的理论下界 $L(G)$ 代替, 实际的最优解结果将不小于该值. 而 Swap 算法的分配结果本身就是在其抽象问题下的一种理论最优解.

表 2 绑定测试结果对比汇总 (以函数为单位)

算法	对比项	MiBench		Rosetta	
		Geo	Avg	Geo	Avg
CMC-h	时间 (ms)	0.98	5.87	0.18	0.60
	大小 (bit)	320.33	499.19	161.01	219.46
	最优解 (bit)	320.00	498.54	161.01	219.46
Swap	时间 (ms)	0.53	3.80	0.090	0.33
	大小 (bit)	324.75	506.38	162.26	223.88
Cong	时间 (ms)	0.15	0.96	0.025	0.073
	大小 (bit)	325.54	508.13	162.27	223.91

图 5 显示了 MiBench 测试集中 CMC 抽象的最优解, Cong 算法的理论下界与 Swap 算法的最优解之间的对比, Cong 算法的理论下界与 Swap 算法的最优解分配结果是一致的, 因此我们在图 5 中合并展示. 其中, 横轴上的每个刻度代表一个函数, 按照 CMC 问题最优解需要的比特数从小到大排序; 纵轴代表所需的寄存器比特数量. 结果显示, 在 22.66% 的样例中, CMC 问题最优解突破了原有 Swap 算法已经达到理论最优解的限制范围, 平均可以减少 6.96% 寄存器数目, 在其余样例中 CMC 问题最优解等于 Swap 算法的最优解.

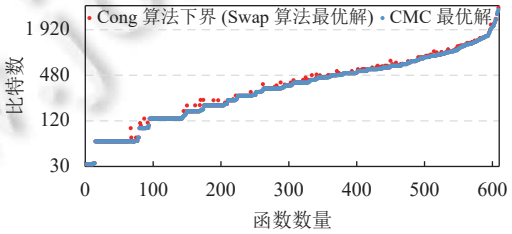


图 5 MiBench 测试集中 CMC 抽象的最优解, Cong 算法下界和 Swap 算法最优解的比较

4.3 绑定方案对比

图 6 对不同的分配器实现在寄存器绑定问题上所需的比特数量进行了比较, 以 Cong 算法为基线进行相对对比. 图中的每个横向刻度代表一个函数, 按照 CMC-h 算法提升的相对百分比进行排序, 而图的纵轴表示其他算法相对于 Cong 算法分配结果的提升百分比. 可以观察到, 在 MiBench 测试集中 CMC-h 算法在仅有 1.1% 的样例中的分配结果略逊于 Swap 算法. 这是由于 CMC-h 算法在变量的分配顺序上存在一定的随机性, 导致在少数样例中错过最优的分配结果, 产生了寄存器冗余现象. CMC-h 算法与 Swap 算法在 76.4% 的样例中同时取得最优解, 在 0.3% 的样例中二者取得相同的分配结果, 但都不是理论最优. 在 22.2% 的样例中, CMC-h 算法优于 Swap 算法; 在 Rosetta 测试集中, CMC-h 算法在 6.9% 的测试样例中分配了比其他两种方法更少的寄存器, 同时在所有的样例中均取得了最优的分配结果. 这些实验结果表明, CMC-h 算法相比于 Cong 算法与 Swap 算法, 能够取得更加优秀的绑定结果, 进一步减少寄存器资源的使用.

图 7 从另一个角度强调了这一结果, 在 MiBench 测试集中, Cong 与 Swap 算法分别只有 73.56% 和 77.34% 的样例达到了 CMC 问题的最优解, 而 CMC-h 算法在 96.72% 的样例中达到了最优解, 相比于前两种方法分别将这一比例提升了 31.5% 和 25.1%. 实验结果表明, CMC-h 算法的绑定结果已接近理论最优, 在所有的测试样例中, 相比于最优解平均仅多使用 0.13% 的额外资源. 在 Rosetta 测试集中, 相比于 Cong 和 Swap 算法均提高了 7.4%, CMC-h 在所有测试用例中均达到了理论最优的分配结果.

此外, 我们简单评估了参数 $\alpha\in[0,1]$ 对于实验结果的影响. 如图 8 所示, 我们等间距地选择 0–1 之间的 11 个

参数在 MiBench 测试集中进行测试, 实验结果表明, 单次连续多种着色达到 CMC 理论最优解的样例占比始终大于 84.4%, 不同参数的选择占比略有不同. 同时, 实验表明通过重复多次搜索可以显著提高这一比例. 在图 7 中, 1–3 次搜索最优解可达占比分别为 84.4%, 94.7% (+10.3%) 和 96.7% (1.3%), 继续提高搜索次数将出现显著的时间-效果边际效益递减, 可见搜索以 1–3 次最为合适.

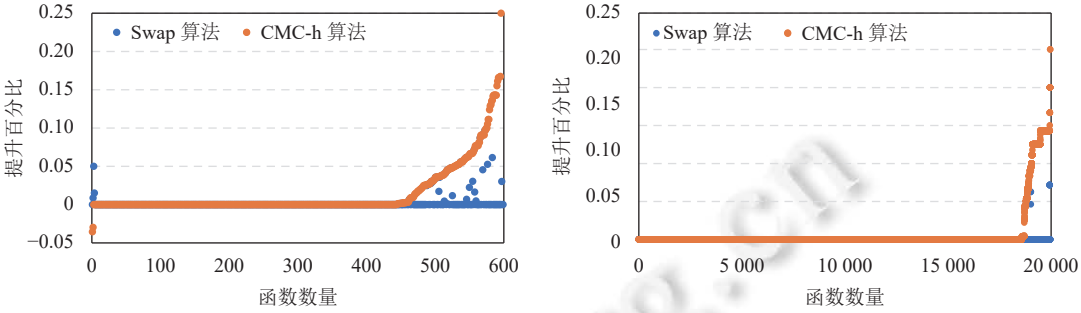


图 6 以 Cong 算法为基线的绑定方案相对对比

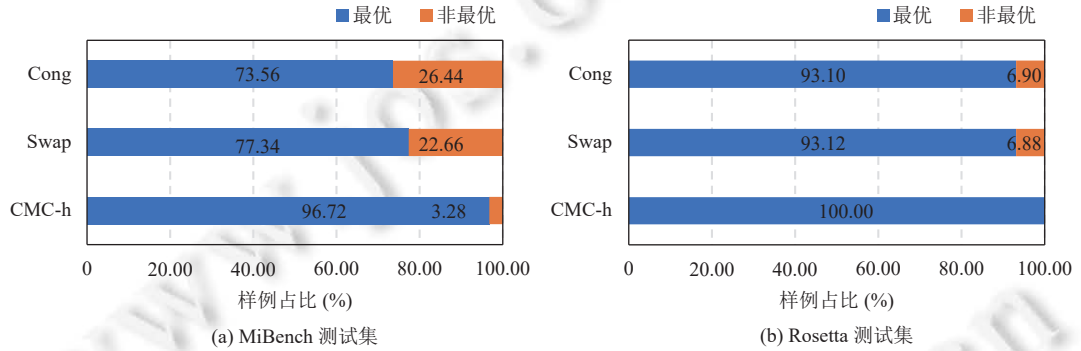


图 7 绑定方案达到 CMC 理论最优解样例占比

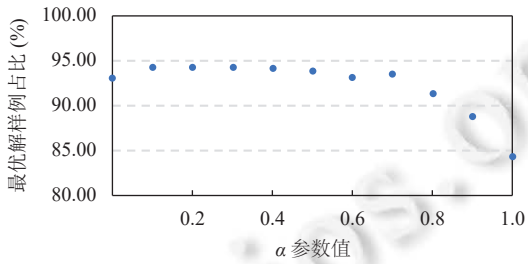
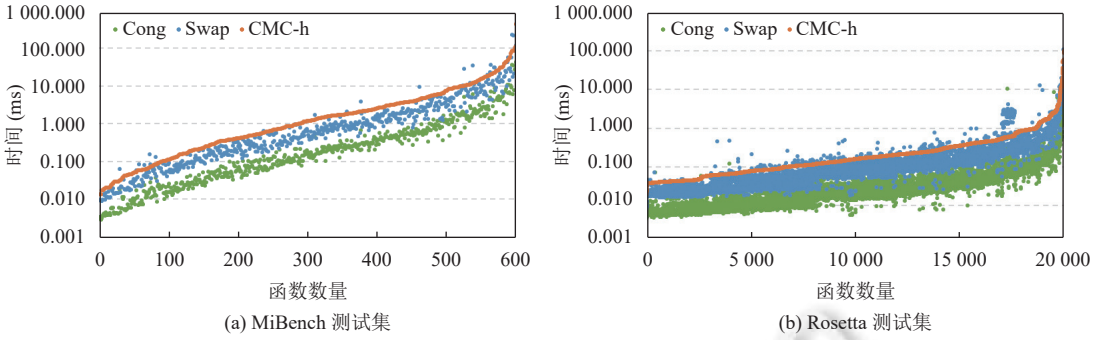


图 8 单次连续多重着色达到 CMC 理论最优解的样例占比

4.4 编译时间对比

图 9 比较了本节评估的 3 种算法的运行时间. 图 9 中的横轴每个点表示一个函数, 按照 CMC 方法的分配时间进行排序; 纵轴代表每种算法的运行时间. 在 MiBench 测试集中 CMC-h 算法平均耗时分别是 Cong 算法和 Swap 算法的 6.1 倍和 1.5 倍, 在 Rosetta 测试集中分别为 8.2 倍和 1.8 倍. 然而, 异常值对这些结果有很大影响. 主要原因是, 在最坏情况下 CMC-h 算法在 2 次分配中选取最优结果. 据统计, 在 MiBench 测试集的 93.1% 的样例中, CMC-h 算法可以在至多一次分配过程中取得最优解, 耗费的时间比 Swap 算法仅多 27.1%, 这意味着 CMC-h 算法倾向于在存在优化空间的样例中花费更多的时间进行搜索, 以达到更优的绑定效果.



(a) MiBench 测试集 (b) Rosetta 测试集

图 9 绑定算法运行时间

CMC-h 算法需要重新构建干涉图进行着色分配, 这一过程时间复杂度为 $O(|V|^2)$, 相较于其他分配方案 $O(|V|+|E|)$ 的时间复杂度略高. 但这种时间付出是完全值得的, 首先, CMC-h 算法本质上仍然是多项式时间复杂度的算法, 除去构建干涉图部分, 算法在时间复杂度上并未高于其他算法. 其次, 在高层次综合领域, 由于编译开销是一次性的, 通过一定程度的编译时间换取更优的功耗和面积设计是常见的处理方法, CMC-h 算法在给定的最大样例中仍然处于秒级时间, 而逻辑综合所需的时间往往长达数分钟甚至更久[38], 寄存器绑定占用编译时间通常不足 1%. 因此我们能够通过有限的搜索显著提升绑定算法中达到最优解的样例比例.

4.5 样例综合分析

本文所讨论的技术旨在用于高层次综合领域. HLS 工具将用 C/C++或 SystemC 等高级语言编写的程序转换为寄存器传输级 (RTL) 结构. 我们熟悉的常用工具有 LegUp, Intel HLS 和 Vivado 等. 它们可以将变量分组并对齐到特定比特位宽 (如 1/2/4/8/16 等) 的寄存器中, 但不支持一般的位宽分配. 因此, 为了演示我们的绑定算法运用于高层次综合的可行性, 我们将 LLVM 实现产生的解决方案手动映射到 Verilog 上, 并由此导出具体的 RTL 级电路代码. 我们使用 Vivado 工具针对 Zynq-7000 后端架构, 对 3 种算法映射出的电路代码进行逻辑综合.

表 3 展示了 3 种算法对例 1 分配结果所产生的电路实际消耗的硬件资源数量. 其中表资源 (LUT) 本质是随机存取存储器 (RAM), 是 FPGA 上实现的组合逻辑电路的主要方式. 触发器资源 (FF) 可存储 1 比特数据, 是寄存器的基本存储单元, 在本例中状态机 (FSM) 需要固定存储 2 比特的状态. 输入/输出单元 (IO) 是芯片与外界电路的接口. Swap 算法插入的额外指令采用了非阻塞赋值的方式实现[13], 在本样例中由于指令的并行性, 该插入指令在实际生成电路中并未产生额外的代价. 由此我们得出了如下结论.

表 3 各算法为例 1 生成 Verilog 代码综合后的资源使用量 (LUT 合并前)

资源	Cong	Swap	CMC-h
LUT	22	24	20
FF	21	18	17
IO	25	26	25

• 比特粒度的寄存器绑定, 生产的逻辑电路是可综合的, 且分配的寄存器数量能够准确反映出实际生产电路的寄存器数目.

• Swap 算法产生了最多的 LUT, 这是由于该算法频繁的数据移动使得数据的分布更为零散, 从而使物理寄存器的复用逻辑变得极为复杂.

• CMC-h 算法相比于其他算法使用的 3 种资源数目均为最少.

值得注意的是, 在本样例中 CMC-h 算法并未产生比 Cong 算法更复杂的寄存器绑定关系. 如图 2 所示, Cong 算法仅在变量 b 和 e 之间共 7 比特寄存器复用, 但电路需要控制变量 d 在第 2 拍写入, 因此控制逻辑共需要插入 10 个二选一多路复用器, 而 CMC-h 算法将会在变量 b, c, d, e 之间共 10 比特寄存器复用, 同样需要插入 10 个二选一多路复用器. 在本样例中, 变量 e 的低 3 位为 0, 可以通过触发器的复位端口实现置 0, 因此 CMC-h 算法产生

的电路比 Cong 算法少用两个多路复用器. 图 10 展示了本文 CMC-h 算法所生成代码的寄存器级 (RTL) 电路图. 根据上述分析, 触发器的 D 端口与 R 端口 (复位) 共需要 8 个 LUT, 由于控制逻辑较为简单, CE 端口只产生 1 个 LUT. 计算单元需要 10 个 LUT, 状态机需要 1 个 LUT, 因此共需 20 个 LUT. 相比之下, Swap 算法插入复制与拷贝指令使得数据共享的状况变得更为复杂, 因此寄存器的 D 端口与 CE 端口将需要更多的 LUT, 从而让电路变得更为复杂.

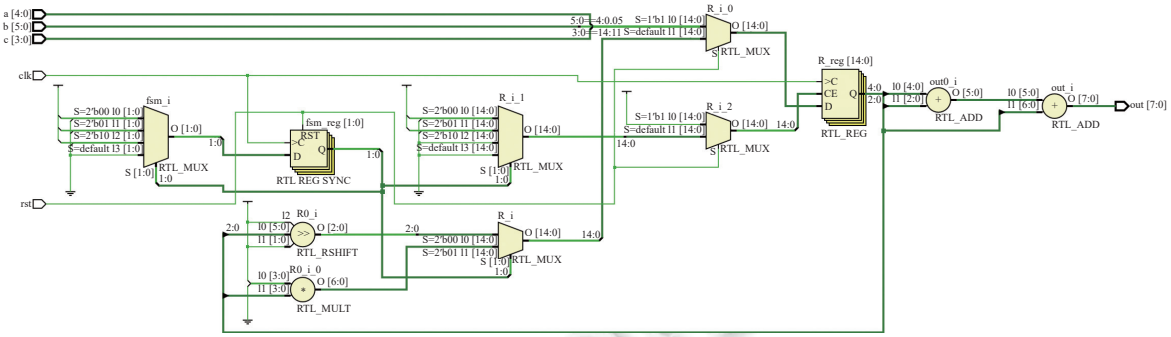


图 10 CMC-h 算法分配例 1 代码对应 RTL 电路

5 总 结

本文提出了一种基于图的连续多重着色的寄存器绑定抽象, 新的抽象可以突破原有算法的理论下界, 最大限度地减少寄存器空泡的产生, 且无需插入额外的指令. 我们在不考虑区间约束的条件下给出了一个新的理论下界, 该下界可以作为我们的算法的理论边界提供重要的参考依据. 我们提出了一种基于优先度的启发式方法, 尽管我们的方法目前还不是最优解, 但实验表明, 在所有的测试集中该方法相比于最优解平均仅多用了 0.13% 的寄存器资源, 在 96.72% 的样例中我们的绑定方案是最优的, 相比于其他的方法我们将这一比例提升了 25.1%.

本文的工作在高层次综合领域表现出了潜力, 本文的连续多重着色算法不仅可以应用于寄存器绑定问题, 同样也可以应用于加法器, 进位保留加法器等其他无需对齐约束的硬件资源绑定中. 本文给出的理论下界对应的离散寄存器分配方案, 也可以作为具有参考意义的解生成有效的电路在 FPGA 等场景下应用. 然而在实际的综合过程中, 当前已有的寄存器绑定算法均存在一些问题. 首先, 当前算法仅考虑寄存器的位宽, 而没有考虑互连单元与控制路径的复杂性, 因此无法直接在全局的电路设计中取得面积功耗的最优解. 其次, 位宽分析并没有考虑同一变量在不同程序点的位宽变化. 然而, 在绑定问题中增加更多的考虑因素可能使得算法的设计变得更为复杂, 反而拖累实际的优化效果. 因此我们下一步的工作准备在寄存器绑定中加入更多的分析与考虑, 以便于在实际的综合过程中得到更好的效果.

References:

[1] Moore GE. Cramming more components onto integrated circuits. Proc. of the IEEE, 1998, 86(1): 82–85. [doi: 10.1109/JPROC.1998.658762]

[2] Schafer BC, Wang Z. High-level synthesis design space exploration: Past, present, and future. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2020, 39(10): 2628–2639. [doi: 10.1109/TCAD.2019.2943570]

[3] Hennessy JL, Patterson DA. A new golden age for computer architecture. Communications of the ACM, 2019, 62(2): 48–60. [doi: 10.1145/3282307]

[4] Cong J, Lau J, Liu G, Neuendorffer S, Pan PC, Vissers K, Zhang ZR. FPGA HLS today: Successes, challenges, and opportunities. ACM Trans. on Reconfigurable Technology and Systems, 2022, 15(4): 51. [doi: 10.1145/3530775]

[5] Coussy P, Gajski DD, Meredith M, Takach A. An introduction to high-level synthesis. IEEE Design & Test of Computers, 2009, 26(4): 8–17. [doi: 10.1109/MDT.2009.69]

[6] Ferrandi F, Castellana VG, Curzel S, Fezzardi P, Fiorito M, Lattuada M, Minutoli M, Pilato C, Tumeo A. Invited: Bambu: An open-

source research framework for the high-level synthesis of complex applications. In: Proc. of the 58th ACM/IEEE Design Automation Conf. (DAC). San Francisco: IEEE, 2021. 1327–1330. [doi: 10.1109/DAC18074.2021.9586110]

[7] Canis A, Choi J, Aldham M, Zhang V, Kammoona A, Czajkowski T, Brown SD, Anderson JH. LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems. ACM Trans. on Embedded Computing Systems, 2013, 13(2): 24. [doi: 10.1145/2514740]

[8] Brisk P, Dabiri F, Jafari R, Sarrafzadeh M. Optimal register sharing for high-level synthesis of SSA form programs. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2006, 25(5): 772–779. [doi: 10.1109/TCAD.2006.870409]

[9] Rim M, Mujumdar A, Jain R, De Leone R. Optimal and heuristic algorithms for solving the binding problem. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 1994, 2(2): 211–225. [doi: 10.1109/92.285747]

[10] Kum KI, Sung W. Combined word-length optimization and high-level synthesis of digital signal processing systems. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2001, 20(8): 921–930. [doi: 10.1109/43.936374]

[11] Cong J, Fan YP, Han GL, Lin YZ, Xu JJ, Zhang ZR, Cheng X. Bitwidth-aware scheduling and binding in high-level synthesis. In: Proc. of the 2005 Asia and South Pacific Design Automation Conf. Shanghai: IEEE, 2005. 856–861. [doi: 10.1109/ASPDAC.2005.1466476]

[12] Wu Q, Bian JN, Xue HX. Scheduling with resource allocation for system-level synthesis. Ruan Jian Xue Bao/Journal of Software, 2007, 18(2): 220–228 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/220.htm [doi: 10.1360/jos180220]

[13] Canesche M, Ferreira R, Nacif JA, Quintão Pereira FM. A polynomial time exact solution to the bit-aware register binding problem. In: Proc. of the 31st ACM SIGPLAN Int'l Conf. on Compiler Construction. Seoul: ACM, 2022. 29–40. [doi: 10.1145/3497776.3517773]

[14] Lattner C, Adve V. LLVM: A compilation framework for lifelong program analysis & transformation. In: Proc. of the 2004 Int'l Symp. on Code Generation and Optimization. San Jose: IEEE, 2004. 75–86. [doi: 10.1109/CGO.2004.1281665]

[15] Guthaus MR, Ringenberg JS, Ernst D, Austin TM, Mudge T, Brown RB. MiBench: A free, commercially representative embedded benchmark suite. In: Proc. of the 4th Annual IEEE Int'l Workshop on Workload Characterization. Austin: IEEE, 2001. 3–14. [doi: 10.1109/WWC.2001.990739]

[16] Zhou Y, Gupta U, Dai S, Zhao R, Srivastava N, Jin HC, Featherston J, Lai YH, Liu G, Velasquez GA, Wang WP, Zhang ZR. Rosetta: A realistic high-level synthesis benchmark suite for software programmable FPGAs. In: Proc. of the 2018 ACM/SIGDA Int'l Symp. on Field-programmable Gate Arrays. Monterey: ACM, 2018. 269–278. [doi: 10.1145/3174243.3174255]

[17] Cong J, Liu B, Neuendorffer S, Noguera J, Vissers K, Zhang ZR. High-level synthesis for FPGAs: From prototyping to deployment. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2011, 30(4): 473–491. [doi: 10.1109/TCAD.2011.2110592]

[18] Nane R, Sima VM, Pilato C, Choi J, Fort B, Canis A, Chen YT, Hsiao H, Brown S, Ferrandi F. A survey and evaluation of FPGA high-level synthesis tools. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2016, 35(10): 1591–1604. [doi: 10.1109/TCAD.2015.2513673]

[19] Canis A, Choi J, Aldham M, Zhang V, Kammoona A, Anderson JH, Brown S, Czajkowski T. LegUp: High-level synthesis for FPGA-based processor/accelerator systems. In: Proc. of the 19th ACM/SIGDA Int'l Symp. on Field Programmable Gate Arrays. Monterey: ACM, 2011. 33–36. [doi: 10.1145/1950413.1950423]

[20] Chaitin GJ, Auslander MA, Chandra AK, Cocke J, Hopkins ME, Markstein PW. Register allocation via coloring. Computer Languages, 1981, 6(1): 47–57. [doi: 10.1016/0096-0551(81)90048-5]

[21] Bouchez F, Darte A, Guillon C, Rastello F. Register allocation: What does the NP-completeness proof of Chaitin *et al*. really prove? Or revisiting register allocation: Why and how. In: Proc. of the 19th Int'l Workshop on Languages and Compilers for Parallel Computing. New Orleans: Springer, 2006. 283–298. [doi: 10.1007/978-3-540-72521-3_21]

[22] Hack S, Grund D, Goos G. Register allocation for programs in SSA-form. In: Proc. of the 15th Int'l Conf., CC 2006, Held as Part of the Joint European Conf. on Theory and Practice of Software Compiler Construction. Vienna: Springer, 2006. 247–262. [doi: 10.1007/11688839_20]

[23] Pereira FMQ, Palsberg J. Register allocation via coloring of chordal graphs. In: Proc. of the 3rd Asian Symp. on Programming Languages and Systems. Tsukuba: Springer, 2005. 315–329. [doi: 10.1007/11575467_21]

[24] Yuan XL, Shen XB. A new register allocation algorithm. Chinese Journal of Computers, 1998, 21(S1): 68–72 (in Chinese with English abstract). [doi: 10.3321/j.issn:0254-4164.1998.z1.013]

[25] Briggs P, Cooper KD, Torczon L. Improvements to graph coloring register allocation. ACM Trans. on Programming Languages and Systems, 1994, 16(3): 428–455. [doi: 10.1145/177492.177575]

[26] Nandivada VK, Barik R. Improved bitwidth-aware variable packing. ACM Trans. on Architecture and Code Optimization, 2013, 10(3): 16. [doi: 10.1145/2509420.2509427]

[27] Tallam S, Gupta R. Bitwidth aware global register allocation. In: Proc. of the 30th ACM SIGPLAN-SIGACT Symp. on Principles of

Programming Languages. New Orleans: ACM, 2003. 85–96. [doi: 10.1145/604131.604139]

[28] Ergin O, Balkan D, Ghose K, Ponomarev D. Register packing: Exploiting narrow-width operands for reducing register file pressure. In: Proc. of the 37th Int'l Symp. on Microarchitecture (MICRO-37'04). Portland: IEEE, 2004. 304–315. [doi: 10.1109/MICRO.2004.29]

[29] Stephenson M, Babb J, Amarasinghe S. Bidwidth analysis with application to silicon compilation. ACM SIGPLAN Notices, 2000, 35(5): 108–120. [doi: 10.1145/358438.349317]

[30] Patterson JR. Accurate static branch prediction by value range propagation. In: Proc. of the 1995 ACM SIGPLAN Conf. on Programming Language Design and Implementation. La Jolla: ACM, 1995. 67–78. [doi: 10.1145/207110.207117]

[31] Budiu M, Sakr M, Walker K, Goldstein SC. BitValue inference: Detecting and exploiting narrow bitwidth computations. In: Proc. of the 6th Int'l Euro-Par Conf. Germany: Springer, 2000. 969–979. [doi: 10.1007/3-540-44520-X_137]

[32] Constantinides GA, Cheung PYK, Luk W. Optimal datapath allocation for multiple-wordlength systems. Electronics Letters, 2000, 36(17): 1508–1509. [doi: 10.1049/EL:20001044]

[33] Li P, Zhang P, Pouchet LN, Cong J. Resource-aware throughput optimization for high-level synthesis. In: Proc. of the 2015 ACM/SIGDA Int'l Symp. on Field-programmable Gate Arrays. Monterey: ACM, 2015. 200–209. [doi: 10.1145/2684746.2689065]

[34] Chen DM, Cong J, Fan YP, Zhang ZR. High-level power estimation and low-power design space exploration for FPGAs. In: Proc. of the 2007 Asia and South Pacific Design Automation Conf. Yokohama: IEEE, 2007. 529–534. [doi: 10.1109/ASPDAC.2007.358040]

[35] Gort M, Anderson JH. Range and bitmask analysis for hardware optimization in high-level synthesis. In: Proc. of the 18th Asia and South Pacific Design Automation Conf. (ASP-DAC). Yokohama: IEEE, 2013. 773–779. [doi: 10.1109/ASPDAC.2013.6509694]

[36] Pemmaraju SV, Penumatcha S, Raman R. Approximating interval coloring and max-coloring in chordal graphs. ACM Journal of Experimental Algorithmics, 2005, 10: 1–19. [doi: 10.1145/1064546.1180619]

[37] Bouchard M, Čangalović M, Hertz A. On a reduction of the interval coloring problem to a series of bandwidth coloring problems. Journal of Scheduling, 2010, 13(6): 583–595. [doi: 10.1007/s10951-009-0149-1]

[38] Xiao YL, Park D, Butt A, Giesen H, Han ZY, Ding R, Magnezi N, Rubin R, DeHon A. Reducing FPGA compile time with separate compilation for FPGA building blocks. In: Proc. of the 2019 Int'l Conf. on Field-programmable Technology (ICFPT). Tianjin: IEEE, 2019. 153–161. [doi: 10.1109/ICFPT47387.2019.00026]

附中文参考文献:

[12] 吴强, 边计年, 薛宏熙. 系统级综合中结合资源分配的调度算法. 软件学报, 2007, 18(2): 220–228. http://www.jos.org.cn/1000-9825/18/220.htm [doi: 10.1360/jos180220]

[24] 袁小龙, 沈绪榜. 一种新的寄存器分配算. 计算机学报, 1998, 21(S1): 68–72. [doi: 10.3321/j.issn:0254-4164.1998.z1.013]

高猛(1995－), 男, 博士生, CCF 学生会员, 主要研究领域为异构编译, 高层次综合.

崔慧敏(1979－), 女, 博士, 研究员, 博士生导师, CCF 专业会员, 主要研究领域为并行计算, 并行编译, 并行编程.

赵家程(1989－), 男, 博士, 副研究员, 主要研究领域为异构编译.

冯晓兵(1969－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为编程与编译, 程序分析.