

中国研究生创新实践系列大赛
中国光谷·“华为杯”第十九届中国研究生
数学建模竞赛

题 目 **基于整数非线性优化的芯片资源排布问题的研究**

摘要：

芯片资源分布问题对各种基于芯片的科技产品的工作效率提高和制作成本降低起重要作用。求解该问题，需要同时考虑硬件资源限制和软件程序编译优化问题，规则复杂难以简单刻画，跨越多个领域难度较高。本文针对 PISA 架构的芯片资源的排布问题展开研究，首先对基本块的控制依赖、数据依赖进行算法预处理，然后将问题建模成约束优化问题，采用整数规划方法和模拟退火启发式方法逐步进行求解，最终取得全局较优解：

针对预处理部分，本文首先从约束优化角度对问题建模，构思多种高效算法求解数据依赖和控制依赖，并将其简化为次序约束关系，为问题求解提供良好基础。

针对问题一，利用整数规划模型，根据具体问题建立了一系列的约束条件，将基本块相应分配流水线并最小化流水线的最大级数，使用 CPLEX 的 CP 优化器求解全局较优解。同时针对流水线级数的目标优化问题，引入资源利用率约束，提高流水线资源分配使用率。最终在满足资源限制条件下求得流水线最低级数为 51 级。

针对问题二，首先针对固定流水线上同一执行流程基本块资源求和限制问题，提出将其转换为最大路求解问题，并使用拓扑序的动态规划高效求解。其次使用模拟退火启发式算法，并利用问题一所求较优解作为初始点热启动，求解近似较优解，避开整数规划复杂度。

整个模型针对芯片程序基本块的流水级排布问题进行详细的数学建模和约束优化，实现了对给出程序指令依赖求解和流水分配重排，提高了芯片资源排布利用率。最后，本文提供了对使用算法和求解结果进行分析，对类似的资源配置问题有较强借鉴意义，具有广阔应用前景。

关键字： 资源排布 整数规划 非线性约束 启发式算法

目录

| | |
|------------------------|----|
| 1 问题重述 | 4 |
| 1.1 引言 | 4 |
| 1.2 问题的提出 | 4 |
| 2 模型的假设 | 6 |
| 3 符号说明 | 7 |
| 4 模型的建立 | 8 |
| 4.1 问题分析 | 8 |
| 4.1.1 整体分析 | 8 |
| 4.1.2 优化目标 | 8 |
| 4.1.3 约束方程 | 9 |
| 4.2 算法预处理 | 11 |
| 4.2.1 控制依赖算法 | 12 |
| 4.2.2 数据依赖算法 | 14 |
| 4.2.3 执行流程搜索算法 | 15 |
| 5 模型求解 | 17 |
| 5.1 问题一分析与求解 | 17 |
| 5.1.1 模型建立 | 17 |
| 5.1.2 模型求解 | 18 |
| 5.2 问题二分析与求解 | 18 |
| 5.2.1 模型建立 | 18 |
| 5.2.2 模型求解 | 19 |
| 6 模型评价 | 20 |
| 6.1 模型优点 | 20 |
| 6.2 模型缺点 | 20 |
| 6.3 模型的改进与推广 | 20 |
| 参考文献 | 20 |
| 附录 A Python 源程序 | 22 |

| | |
|---------------------|----|
| A.1 预处理程序 | 22 |
| A.2 问题一程序 | 25 |
| A.3 问题二程序 | 28 |

1 问题重述

1.1 引言

芯片是电子行业的基础工业项目，也是目前中国“卡脖子”领域的重要一环，是高科技领域的兵家必争之地。一组数据表示，2016年中国制造的芯片销售额为130亿元，占到了全球芯片销售额的3.8%。但中国消耗芯片占到了全球的60.5%，排名第二的美国，消耗芯片占比也仅仅只有11.4%。值得一提的是，美国的芯片制造能力是世界数一数二的，全球超过60%的芯片都来自于美韩。如何更有效的制造、开发芯片产品是一个重要的问题，也是加快我国芯片产业发展，抢占科技高地的关键性问题。

本课题针对的PISA（Protocol Independent Switch Architecture）是当前主流的可编程交换芯片架构，其有着和固定功能交换芯片相当的处理速率，同时兼具了可编程性，在未来网络中具有广阔的应用场景。在PISA的多级报文处理流水线部分中，编译器在编译P4程序时，会首先将P4程序划分为一系列的基本块，再将各基本块排布到流水线各级当中。由于各基本块均会占用一定的芯片资源，将基本块排布到流水线各级即是将各基本块的资源排布到流线各级当中（即需要确定每个基本块排布到流水线哪一级），因此我们将基本块的排布问题称作PISA架构芯片资源排布问题。^[1, 2]本论文致力于如何设计高资源利用率的资源排布算法以让芯片支持更多的业务，提高整个芯片的资源利用率。

1.2 问题的提出

本文围绕着芯片高资源利用率的资源排布算法展开了讨论。基本块是在源程序里截取的相关片段，其可以被抽象成一个节点，抽象后基本块中执行的具体指令被屏蔽，只保留读写的变量信息。当基本块A执行完可以跳转到基本块B执行时，在A和B之间增加一条有向边，这样P4程序即可表示为一个有向无环图（P4程序不存在循环），称作P4程序流程图。PISA架构资源排布即是将P4程序流程图中的各节点（即各基本块）在满足约束条件下排布到流水线各级当中。排布过程中有以下约束：

数据依赖：P4程序每个基本块均会写一部分变量（即对变量赋值）和读一部分变量（读取程序内变量的值），变量的读写使得基本块之间存在数据依赖，这将会限制基本块之间的排布顺序。数据依赖详细可以分为三类：

1. S1在S2之前执行，当S1写某个变量而S2读该变量时，S1和S2存在写后读数据依赖。例如S1写了变量a，而S2读了变量a，则S1和S2存在写后读依赖。
2. S1在S2之前执行，当S1读某个变量而S2写该变量时，S1和S2存在读后写数据依赖。例如S1读了变量b，而S2写了变量b，则S1和S2存在读后写依赖。
3. S1在S2之前执行，当S1和S2均写某个变量时，S1和S2存在写后写数据依赖。

例如 S1 和 S2 均写了 c 变量，则 S1 和 S2 存在写后写数据依赖。

在 PISA 架构中，当基本块 A 和 B 存在写后读数据依赖或写后写数据依赖时，基本块 A 排布的流水线级数需要小于基本块 B 排布的级数；当基本块 A 和 B 存在读后写数据依赖时，基本块 A 排布的流水线级数需要小于或等于基本块 B 排布的级数。

控制依赖：基本块执行完后可能跳转到多个基本块执行，从而使得基本块之间也存在着控制依赖，这也将影响基本块控制依赖定义为当从某个基本块出发的路径，只有部分路径通过下游某个基本块时，两基本块构成控制依赖。在 PISA 架构中，如果基本块 A 与基本块 B 存在控制依赖，则 A 排布的流水线级数需要小于或等于 B 排布的流水线级数。如图1.1 所示的控制流图，B1 与 B2、B5、B6、B7、B8 之间存在控制依赖，B5 与 B6、B8 之间存在控制依赖，但 B5 和 B7 不构成控制依赖，因为从 B5 出发的两条路径下游都会经过 B7。

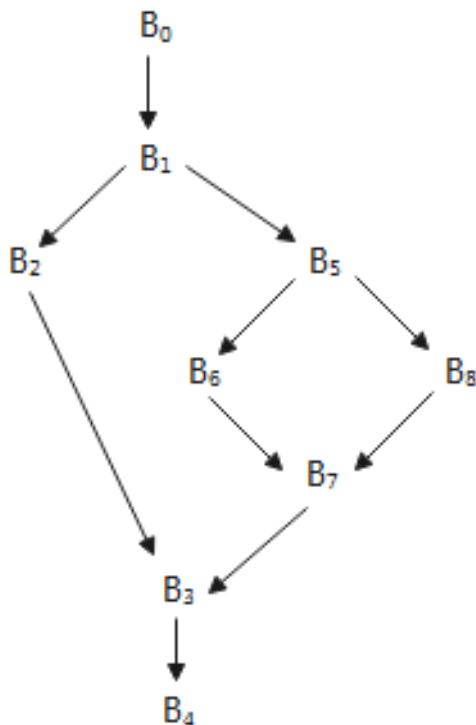


图 1.1 控制依赖示例

资源约束：芯片内部的资源约束，芯片中的资源包括 TCAM、HASH、ALU、QUALIFY 四类，资源排布时不能违反芯片的物理资源限制。

在满足以上前提的情况下，本文解决了以下两个问题：

问题一：给定资源约束条件如下：

- (1) 流水线每级的 TCAM 资源最大为 1;
- (2) 流水线每级的 HASH 资源最大为 2;
- (3) 流水线每级的 ALU 资源最大为 56;
- (4) 流水线每级的 QUALIFY 资源最大为 64;
- (5) 约定流水线第 0 级与第 16 级, 第 1 级与第 17 级, …, 第 15 级与第 31 级为折叠级数, 折叠的两级 TCAM 资源加起来最大为 1, HASH 资源加起来最大为 3。如果需要的流水线级数超过 32 级, 则从第 32 开始的级数不考虑折叠资源限制;
- (6) 有 TCAM 资源的偶数级数量不超过 5;
- (7) 每个基本块只能排布到一级。

在上述资源约束条件下进行资源排布, 并以占用的流水线级数尽量短为优化目标。

问题二: 在 P4 程序流程图中, 由一个基本块出发可以到达另一个基本块则两基本块在一条执行流程上, 反之不在一条执行流程上。对于这种不在一条执行流程上的基本块, 可以共享 HASH 资源和 ALU 资源, 基本块 2 和 3 中任意一个的 HASH 资源与 ALU 资源均不超过每级资源限制, 基本块 2 和 3 即可排布到同一年级。据此, 对问题 1 中的约束条件(2)、(3)、(5) 作如下更改:

- (2) 流水线每级中同一条执行流程上的基本块的 HASH 资源之和最大为 2;
- (3) 流水线每级中同一条执行流程上的基本块的 ALU 资源之和最大为 56;
- (5) 折叠的两级, 对于 TCAM 资源约束不变, 对于 HASH 资源, 每级分别计算同一条执行流程上的基本块占用的 HASH 资源, 再将两级的计算结果相加, 结果不超过 3。

其它约束条件同问题 1, 更改资源约束条件后重新考虑问题 1, 给出排布算法, 输出基本块排布结果。

2 模型的假设

1. P4 程序图由可以表示为有向无环图;
2. 每个基本块对数据的操作遵循先读后写的原则, 且仅会写同一数据一次;
3. 基本块被抽象成一个节点, 抽象后基本块中执行的具体指令被屏蔽, 只保留读写的变量信息, 每个基本块都能在流水线的同一个级上完成部署;
4. 资源排布过程仅由前文提到的三种约束限制, 其余条件忽略不计;
5. 没有数据依赖与控制依赖的情况下, 基本块在流水线上的位置没有先后限制。

3 符号说明

| 符号 | 意义 |
|----------|--|
| S_i | 代表程序的第 i 个节点 |
| t_i | 代表第 i 个节点所需的 TCAM 资源数量 |
| h_i | 代表第 i 个节点所需的 HASH 资源数量 |
| a_i | 代表第 i 个节点所需的 ALU 资源数量 |
| q_i | 代表第 i 个节点所需的 QUALIFY 资源数量 |
| r_i | 代表第 i 个节点被放置的流水线级数 |
| R_i | 代表与第 i 个节点存在依赖（控制依赖或者数据依赖）的其余节点的流水线级数的集合 |
| N | 代表所有基本块的数量 |
| P_j | 代表芯片上的第 j 个流水线 |
| x_{ij} | 代表第 i 个基本块是否处于第 j 级流水线，1 代表是，0 代表否 |
| y_j | 代表芯片上的第 j 个流水线是否被基本块占用，1 代表占用，0 代表未占用 |
| T_c | 代表第 j 个流水线上提供的最大 TCAM 资源数量 |
| H_c | 代表第 j 个流水线上提供的最大 HASH 资源数量 |
| A_c | 代表第 j 个流水线上提供的最大 ALU 资源数量 |
| Q_c | 代表第 j 个流水线上提供的最大 QUALIFY 资源数量 |
| T_j | 代表第 j 个流水线上已经被利用的 TCAM 资源数量 |
| H_j | 代表第 j 个流水线上已经被利用的 HASH 资源数量 |
| A_j | 代表第 j 个流水线上已经被利用的 ALU 资源数量 |
| Q_j | 代表第 j 个流水线上已经被利用的 QUALIFY 资源数量 |
| Son_i | 代表第 i 个节点所有子节点 |

4 模型的建立

4.1 问题分析

4.1.1 整体分析

本问题已经将 P4 程序抽象成基本块，从而 PISA 架构芯片资源排布问题可简化为基本块在流水线各层级中的排布问题。对于划分好的基本块，每个基本块中的程序指令为并行执行，执行时按照先读后写的顺序。

问题的本质是约束优化问题。约束主要来源于两个方面：每级流水线上硬件资源的限制、P4 程序基本块之间的程序依赖（包括数据依赖和控制依赖）。对于资源限制的建模，利用所提供附件内的各个基本块占用资源数量计算；对于依赖的建模，首先需要根据所给程序流图，利用编译原理相关知识，处理出具体基本块之间的控制依赖和数据依赖关系，将其表达成数学约束形式，具体如 4.1.3 叙述。优化的目标是尽量提高流水线资源的利用率，也即最小化使用流水线的最大级数。

从约束优化问题的思路分析，欲求解模型首先需要将各种约束表示成数学形式，随后可以使用包括混合整数规划、包括模拟退火在内的智能优化算法等等方法 [3] 求解。整体的求解思路如下 4.2 所示：

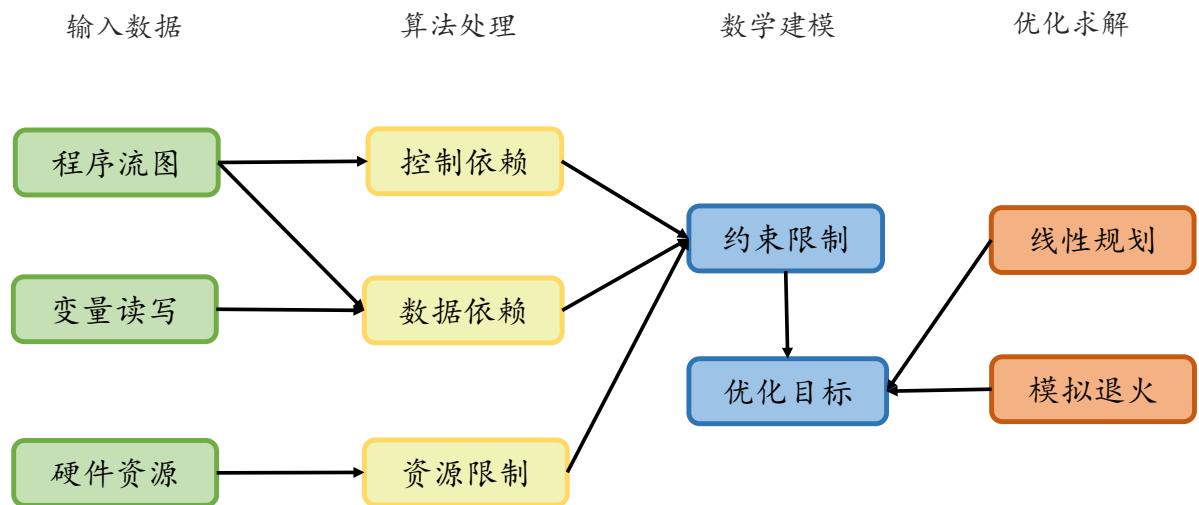


图 4.2 整体求解思路

4.1.2 优化目标

P4 程序图可以表示为一个有向无环图，设 P4 程序图共有 N 个节点，第 i 个节点被记为 $S_i (i = 0, 1, 2, 3, \dots, N)$ ，第 i 个节点需要的 TCAM、HASH、ALU、QUALIFY 的资源数

分别记为 (t_i, h_i, a_i, q_i) 。

由于每个流水线级都能部署一个完整的基本块，所以在不考虑任何依赖的情况下，每个基本块都可以独立地放置在某一个单独的流水线上。将每个流水线级所能提供的最大的 TCAM、HASH、ALU、QUALIFY 的数量记为 (T_c, H_c, A_c, Q_c) ，则对于 $\forall x \in (1, N)$ 而言，都有 $t_i \leq T_c, h_i \leq H_c, a_i \leq A_c, q_i \leq Q_c$ 成立。设所需要的流水线级数为 M ，若不考虑任何的约束条件，则可以将每个基本块放入一个单独的流水线级中，即 $M \leq N$ 。

x_{ij} 表示第 i 个节点是否部署在第 j 级流水线上， $x_{ij} \in \{0, 1\}$ ， x_{ij} 取 1 表示部署，取 0 表示未部署。用 r_i 表示第 i 个节点部署的流水线的级数，即 $r_i = \arg \max_j x_{ij}$ 。要使流水线所使用的级数最少，则优化的目标是使流水线级数的最大值在满足约束的情况下尽可能的小，即：

$$\min \max(r_i) \quad (i = 0, 1, 2, 3, \dots, N) \quad (4.1)$$

4.1.3 约束方程

(1) 数据依赖

由于不同的基本块可能会对同一变量产生读写操作，造成读写数据冲突，总共会产生三种不同的冲突：“写后读”、“写后写”和“读后写”。

假设对某一变量 \mathbf{X} 而言，在一条程序运行的流程线上，存在两个基本块 S_1 和 S_2 ， S_1 和 S_2 的执行顺序为： $S_1 \rightarrow S_2$ ，即 S_1 会比 S_2 先执行。如果 S_1 先写了变量 \mathbf{X} ，而 S_2 读取了变量 \mathbf{X} ，则节点 S_1 和 S_2 之间存在“写后读”依赖，需要 S_1 所在的流水线级数小于 S_2 的流水线级数，即 $r_{S_1} < r_{S_2}$ ；如果 S_1 先写了变量 \mathbf{X} ，而后 S_2 再次对变量 \mathbf{X} 进行了写操作，则 S_1 和 S_2 之间存在“写后写”依赖，需要 S_1 所在的流水线级数小于 S_2 所在的流水线级数，即： $S_1 < S_2$ ；如果 S_1 先读取了变量 \mathbf{X} ，而 S_2 对变量 \mathbf{X} 进行了写操作，则 S_1 和 S_2 之间存在“读后写”依赖，需要 S_1 所在的流水线级数小于等于 S_2 的流水线级数，即 $r_{S_1} \leq r_{S_2}$ 。

(2) 控制依赖

控制依赖定义为：当从某个基本块出发的路径，只有部分路径通过下游某个基本块时，两基本块构成控制依赖。

假设节点 S 存在 M 条出边，从第 m 条出边能够抵达的所有的顶点构成的集合记为 $L_m (m = 1, 2, 3, \dots, M)$ ，记节点 S 所控制的点构成的集合为 R_S 。当 $M = 1$ 时，从节点 S 出发的所有出边能够抵达 L_1 中所有的点，则节点 S 不与集合 L_1 中的所有节点构成控制依赖，即 $R_S = \emptyset$ 。当 $M \geq 2$ 时，节点 S 拥有多条出边，设能够从节点的第 m 条出边抵达节点 S_x ，则 $S_x \in L_m$ 。当对于 $\forall i \in \{1, 2, 3, \dots, M\}$ ，都有 $S_x \in L_i$ 时，则从节点 S 出发的路径，全部都经过了结点 S_x ，即 S_x 不被 S 所控制，即 $S_x \notin R_S$ 。反之，对于任意的节点

S_y 而言，如果 $\exists i, j \in \{0, 1, 2, 3, \dots, M\}$ ，使得 $S_y \in L_i$ 且 $S_y \notin L_j$ 时，则节点 S_y 被节点 S 所控制，即 $S_y \in R_S$ 。因为节点 S 有 M 条出边，所以节点 S 必然拥有 M 个子节点。由于 P4 程序属于有向无环图，则节点 S 的子节点无法相互抵达，则集合 R_S 至少存在 M 个元素。在集合 $L_i (i = 1, 2, 3, \dots, M)$ 中出现过，但不是在所有集合中存在的节点必然被节点 S 所控制，即：

$$R_S = (L_1 \cup L_2 \cup L_3 \dots \cup L_M) - (L_1 \cap L_2 \cap L_3 \dots \cap L_M) \quad (4.2)$$

R_S 中的所有节点在控制依赖的限制下，流水线级数必须小于或等于 S 所在的流水线级数，即： $r_S \leq r_i (i \in R_S)$ 。

将为了满足数据依赖和控制依赖，节点 S_i 所在的级数必须小于的节点构成的集合记为： $R_1^<$ ，将为了满足数据依赖和控制依赖，节点 S_i 所在的级数必须小于或等于的节点构成的集合记为： $R_1^≤$ 。

(3) 资源约束

资源约束定义为：每个流水级上的所有基本块占用的资源总和不能超出流水级上的物理限制。

事实上，资源约束在日常生产生活中是普遍存在的线性规划问题：处理问题的能力或者物理资源总是有限的，如何合理配置资源使其能够发挥最大作用常常是线性规划中较为关注的。在本题目的问题中，资源约束是芯片每个流水级内可利用的 TCAM、HASH、ALU、QUALIFY 的资源总数限制的，可以用如下的不等式表示：

$$\left\{ \begin{array}{l} \sum_{i=0}^N x_{ij} t_i < T_c (j = 0, 1, 2, 3 \dots) \\ \sum_{i=0}^N x_{ij} h_i < H_c (j = 0, 1, 2, 3 \dots) \\ \sum_{i=0}^N x_{ij} a_i < A_c (j = 0, 1, 2, 3 \dots) \\ \sum_{i=0}^N x_{ij} q_i < Q_c (j = 0, 1, 2, 3 \dots) \end{array} \right. \quad (4.3)$$

为了对每级流水级的资源数的利用效率有明确的数学定义，定义变量“资源利用率” d_j 表示第 j 级流水级上所有资源的整体利用效率：假设第 j 级流水级上已经使用的 TCAM、HASH、ALU、QUALIFY 资源为 T_j, H_j, A_j, Q_j ，则有：

$$d_j = \frac{\frac{T_j}{T_c} + \frac{H_j}{H_c} + \frac{A_j}{A_c} + \frac{Q_j}{Q_c} + 1}{5} \quad (4.4)$$

显然这是一个归一化的结果， $d_j \in [0, 1]$ 。对整个程序流程中，若要达到流水级数最短的效果，每级流水级上的资源利用率会被期望更高，即有优化目标：

$$\max(\min(d_j)) \quad \& \quad \max(\max(d_j)) \quad (j = 0, 1, 2, 3 \dots) \quad (4.5)$$

最终完成的基本块整体排布必须针对每级流水级都满足以上条件，即每级流水线上所有的基本块所需要的 TCAM、HASH、ALU、QUALIFY 资源总数不能超过规定的流水级提供的物理资源总数，且希望每级整体的资源利用率维持在较高的水平。

结合以上三种约束，可以构建所有基本块之间的顺序约束。根据顺序约束，将所有基本块在流水线之中进行排布，并使得流水线级数最小，并获得最终的排布结果。

4.2 算法预处理

在求解问题 1 和问题 2 之前，需要先根据输入的程序流图和基本块读写情况处理出基本块之间的控制依赖和数据依赖，进而能够在问题 1 和问题 2 中列出约束方程。同时，由于问题 2 涉及同一执行路径上的节点约束，还需要设计计算图上子节点执行路径的算法。

首先可以由题目给出的基本条件预处理得知程序流图只有一个根节点为第 365 号，程序流图可视化如图4.3所示，其中红色节点为根节点。

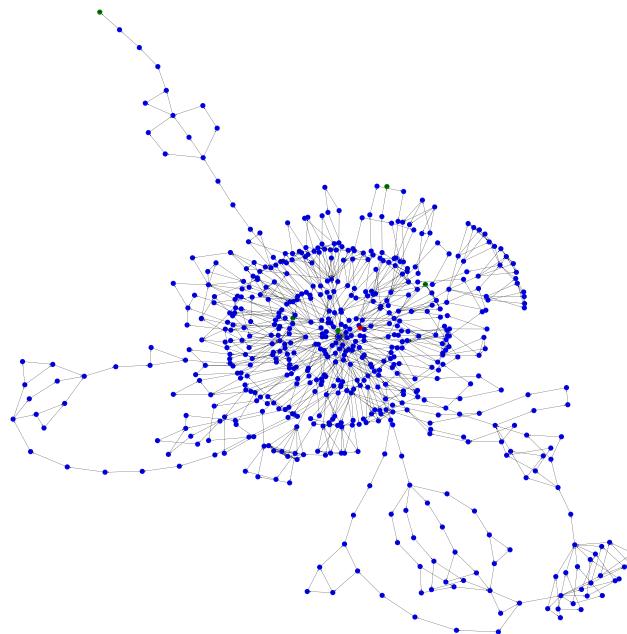


图 4.3 程序流图

4.2.1 控制依赖算法

控制依赖的定义是：当从某个基本块出发的路径，只有部分路径通过下游某个基本块时，两基本块构成控制依赖；相应的在程序流图中，从节点 S_x 可以到达某节点 S_y ，但是并不是 S_x 的所有路径都通过 S_y ，那么 S_y 控制依赖于 S_x 。

下图4.4即为控制依赖的一个例子，节点 2 和节点 4 为 if 判断后的两个分支，具体执行哪个节点需要根据节点 1 的 ‘if’ 值判断，因而不能在流水调度的时候将节点 2 或 4 置于节点 1 之前，此即为控制依赖。

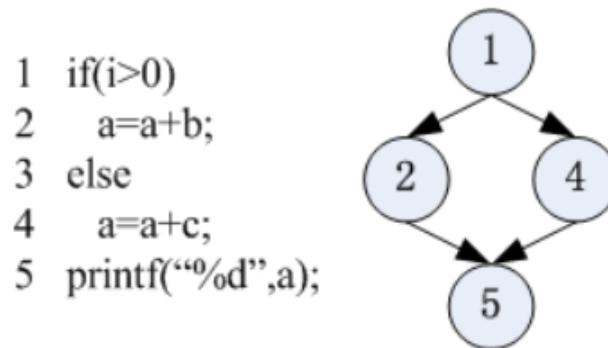


图 4.4 控制依赖图示

在现代编译器实现中，对于程序流图中基本块的求解方式往往利用先计算支配节点、确定支配边界后求解程序控制依赖图；或者使用利用了必经节点性质的 Lengauer-Tarjan 算法 [4]。在本问题中，由于已经假定是单个程序的执行且程序中不存在循环，也即程序流图是只有一个顶点的有向无环图，因此提出一种简化版本的求解控制依赖的算法，可以仅使用单次深度优先搜索完成控制依赖的求解。为提高后续约束求解变量效率，将控制约束简化成表格表示。

算法的思想如4.5所示，假设图中 1 节点下共有 2、3、4 三个子节点，这些子节点分别能够到达的节点集合记为 A、B、C，那么 1 节点经过的所有节点一定是 2、3、4 节点经过节点和 2、3、4 节点自身的并集；而控制依赖于 1 节点的点，也就是 1 节点能够到达但不是全部路径都到达的节点，一定属于 A、B、C 不共同相交的部分。

利用该思想，从程序流图的根节点开始深度优先搜索所有的子节点，每个节点 S_{cur} 在搜索的时候需要做如下操作：如果是叶节点，则 Son_{cur} 直接置为空；否则在其每个子节点 S_{next_i} ($i = 0, 1, 2, \dots, k$) 搜索返回时，获取每个子节点能够到达的所有节点 Son_{next_i} ，则该节点经由子节点 S_{next_i} 能够到达的所有子节点为 $path_i = Son_{next_i} \cup S_{next_i}$ ；那么经由该节点能够到达，但不是所有路径都能到达的节点即为 $Control_{S_{cur}} = (\bigcup_{i,j,i \neq j} (path_i - path_j)) \cap path_0$ ，能够到达的所有子节点即为 $\bigcup_i (path_i)$ ，此即为控制依赖于 S_{cur} 的节点。

下表4.2.3即为根据所给程序流图 G 求解控制依赖次序表格 T 的算法，其中 G 中顶点

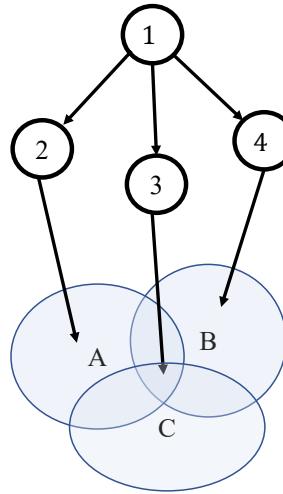


图 4.5 算法思想图示

$V_i (i = 0, 1, 2, \dots, N)$ 对应基本块 $S_i (i = 0, 1, 2, \dots, N)$; 第 i 个顶点到第 j 个顶点存在有向边, 表示第 i 个基本块执行完毕后能够跳转到第 j 个基本块, $T[i, j] = 1$ 表示第 j 个基本块控制依赖于第 i 个基本块。

算法 1 控制依赖求解

输入: 程序流图 G

输出: 控制依赖次序表格 T

```

1: T = [], Son = []
2: for 所有当前要搜索的节点 cur do
3:   for 当前节点的所有子节点 next do
4:     Path = []
5:     if 子节点没有被搜索过 then
6:       进入子节点搜索
7:     else
8:       Son[cur] |= Son[next]
9:       Path += (Son[next] + next)
10:      T[cur] = ( $\bigcup_{i,j,i \neq j} (path_i - path_j)$ )  $\cap path_0$ 
11: return T

```

4.2.2 数据依赖算法

利用程序流图和基本块读写变量的数据，可以处理出数据的依赖。数据依赖是指程序语句引用前一个语句的数据的一种情况，显然只需要考虑在同一执行流程上的节点；进而该依赖在程序流图中体现为相连通的两个节点，读写了同一个变量，具体可分为三类：写后读，写后写，读后写。由于问题已经假设每个基本块先读后写，所以读后写冲突的两个基本块可以置于同一流水线，对于写后读和写后写而言，则必须保证后执行的基本块被分配大于前者的级数。

具体处理算法可利用4.2.1处求解处的 Son 数组，记录了所有节点在程序流图上可达的节点， $Son[i, j] = 1(i < j)$ 即表示 S_i 和 S_j 在同一执行流程上。对于每对在同一执行流程上的节点，检查其读写的变量是否有交集即可判断是否冲突。如4.6所示，节点 8 和节点 14 在同一执行流程上，且节点 8 写了节点 14 要读的变量 x_1 ，因而节点 8 和节点 14 存在写后读冲突，同理节点 8 和节点 14 还在变量 x_2 上存在读后写冲突。

和4.2.1相同，将数据依赖处理成基本块之间的流水线级数次序关系：小于等于关系和小于关系，显然小于关系优先级高于小于等于，因此两个基本块如果同时存在小于关系和小于等于关系，则设为小于关系。在4.6中， x_1 上的写后读冲突要求节点 8 对应的流水线次序小于节点 14，而 x_{14} 上的读后写冲突要求节点 8 次序小于等于节点 14，因此取小于关系。

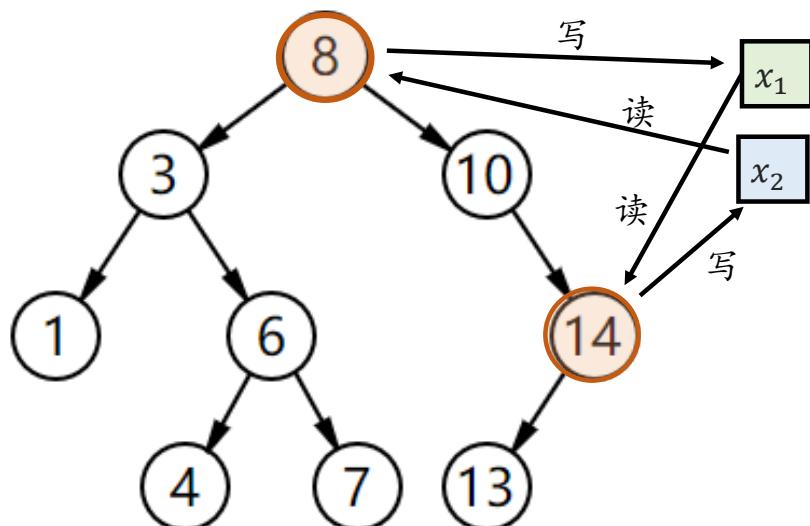


图 4.6 数据依赖图示

算法 2 数据依赖求解

输入: 程序流图 G, 子节点数组 Son

输出: 数据依赖小于关系数组 LT, 小于等于关系数组 LE

```
1: LT = [], LE = []
2: for G 上所有节点 do
3:   for 当前节点 cur 的所有子节点 next do
4:     Path = []
5:     if cur 写的变量和子节点 next 读的变量有重合或 cur 写的变量
       和子节点 next 写的变量有重合 then
6:       LE[cur].append(next)
7:       LT[cur].pop(next)
8:     if cur 读的变量和子节点 next 写的变量有重合且 next 不在
       LT[cur] 中 then
9:       LE[cur].append(next)
10: return T
```

4.2.3 执行流程搜索算法

在问题 2 中, 资源约束需要确定某级流水线内, 在同一执行流程的所有节点(即程序流图中相互间可达), 进而求解同一执行流程节点的资源限制之和是否满足限制。假设已经确定了流水线上基本块的分布, 由于原图上路径数量级很大(比如第 365 号节点到第 0 号节点的路径数为 $1e20$ 的数量级), 直接通过在原图上遍历在给定流水线上的节点的执行路径并不现实。因此将其转换为最长路问题, 利用其 DAG 的性质利用动态规划解决。

给定第 j 个流水线, 欲求解所有同一执行路径的基本块资源之和是否满足限制, 即所有执行路径基本块资源之和的最大值是否满足限制。如果将在给定流水线上占用的资源定义为“路径”, 具体表示如 4.6; 其中 $C[i]$ 代表同一执行路径上受限制的资源 $H[i]$ 和 $A[i]$, d_j 则为在第 j 个流水线上所定义的路径。那么问题可以转换为求解原图上的最长路问题。

$$c_{ij} = \begin{cases} C[i], & \text{if } x_{ij} = 1 \\ 0, & \text{if } x_{ij} = 0 \end{cases} \quad (4.6)$$

由于原图为只有一个根节点(入度为 0)的有向无环图, 因此求解最长路可以直接使用拓扑排序结合动态规划算法。第 i 个节点为终点的最长路径长度 Dis_i , 则 i 父节点 k 的最长距离满足 4.7。

$$Dis_i = \max_k(Dis_i, Dis_k) + c_{ij} \quad (4.7)$$

具体算法如下:

算法 3 同一执行路径最大资源之和求解

输入: 程序流图 G(根节点 root), 入度数组 IN, 资源 C, 流水线级数 j

输出: 最大路径 Cmax, 也即同一执行路径下最大资源之和

```
1: 最大距离数组 Dis = [], 节点队列 Queue = [], 路径数组 d = []
2: for 所有的基本块编号 i do
3:   c[i,j] = C[i] * x[i,j]
4:   Dis[root] = c[root,j]
5:   Queue.put(root)
6: while Queue 不为空 do
7:   cur = Queue.pop()
8:   for cur 的所有子节点 next do
9:     IN[next] -= 1
10:    Dis[next] = max(Dis[next],Dis[cur]+c[next])
11:    if IN[next] == 0 then Queue.put(next)
12: return max(Dis)
```

5 模型求解

5.1 问题一分析与求解

5.1.1 模型建立

在满足已有的约束条件下，对问题一中的参数进行赋值，确定了每个流水级的资源总数，明确了资源约束，即：

$$\begin{cases} T_c = 1(i = 0, 1, 2, 3 \dots) \\ H_c = 2(i = 0, 1, 2, 3 \dots) \\ A_c = 56(i = 0, 1, 2, 3 \dots) \\ Q_c = 64(i = 0, 1, 2, 3 \dots) \end{cases} \quad (5.8)$$

并且约定流水线第 0 级与第 16 级，第 1 级与第 17 级，…，第 15 级与第 31 级为折叠级数，折叠的两级 TCAM 资源加起来最大为 1，HASH 资源加起来最大为 3。注：如果需要的流水线级数超过 32 级，则从第 32 开始的级数不考虑折叠资源限制，这一限制约束可以表述为如下的不等式：

$$\begin{cases} x_{ij}t_i + x_{i+16,j} \cdot t_{i+16} \leq 1(j = 0, 1, 2, 3 \dots, 15) \\ x_{ij}h_i + x_{i+16,j} \cdot h_{i+16} \leq 3(j = 0, 1, 2, 3 \dots, 15) \end{cases} \quad (5.9)$$

第六个约束条件限制了有 TCAM 资源的偶数级不超过 5 个，可以表示为如下不等式：

$$\sum_{j=2k}^M T_j \leq 5(j = 0, 1, 2, 3 \dots) \quad (5.10)$$

其中 $M = \max(r_i)(i = 1, 2, 3, \dots, N)$ ，表示一共有 R 级流水线。

用 $d_{max} = \max(d_i)(i = 0, 1, 2, 3, \dots, M)$ 表示 M 级流水线中最大的资源利用率，用 $d_{min} = \min(d_i)(i = 0, 1, 2, 3, \dots, M)$ 表示 M 级流水线中最低的资源利用率。优化的目标是使资源利用率大的流水线的资源利用率接近 1，资源利用率小的流水线的利用率接近 0，这样就可以尽可能的使每级流水线布满基本块，减少流水线的级数。则优化目标为：

$$\min Mdd = M - d_{max} + d_{min} \quad (5.11)$$

结合上述的约束不等式和优化目标，可得问题一的求解模型如下所示：

$$\begin{aligned}
 & \min Mdd = M - d_{max} + d_{min} \\
 & \text{s.t.} \\
 & \sum_{i=0}^N x_{ij}t_i < T_c \quad (j = 0, 1, 2, 3 \dots) \\
 & \sum_{i=0}^N x_{ij}h_i < H_c \quad (j = 0, 1, 2, 3 \dots) \\
 & \sum_{i=0}^N x_{ij}a_i < A_c \quad (j = 0, 1, 2, 3 \dots) \\
 & \sum_{i=0}^N x_{ij}q_i < Q_c \quad (j = 0, 1, 2, 3 \dots) \\
 & x_{ij}t_i + x_{i+16,j} \cdot t_{i+16} \leq 1 \quad (j = 0, 1, 2, 3 \dots, 15) \\
 & x_{ij}h_i + x_{i+16,j} \cdot h_{i+16} \leq 3 \quad (j = 0, 1, 2, 3 \dots, 15) \\
 & \sum_{j=2k}^M T_j \leq 5 \quad (j = 0, 1, 2, 3 \dots) \\
 & r_i \leq r_j \quad (i = 0, 1, 2, 3, \dots, N) \quad (r_j \in R_i^{\leq}) \\
 & r_i < r_j \quad (i = 0, 1, 2, 3, \dots, N) \quad (r_j \in R_i^{<})
 \end{aligned} \tag{5.12}$$

5.1.2 模型求解

观察建立的数学模型的不等式可以发现，这是一个整数非线性规划问题。为了能够在有限的时间内得出了一个较优的可行解，我们采用了 CPLEX 中的 CP (Constraint Programming) 优化器进行求解。CP 优化器的约束条件可以是非线性的，符合本次模型的约束不等式，CP 优化器采用的是带空间缩小的分支限界算法进行求解，可以在一个有限的时间内返回一个次优的可行解。

CP 优化器经过 200 万次的搜索，在有限的时间内求解出满足条件的最少流水线级数为 51。详细的代码见附录A.2，具体的结果见附件一。

5.2 问题二分析与求解

5.2.1 模型建立

与问题一不同，问题二将资源约束替换为每级流水线中同一执行路径上的基本块资源之和约束，如果继续使用问题一中采用 x_{ij} 零一变量表示的方法，由于图上路径数量级过大，遍历执行路径生成的约束数量爆炸难以求解。即使是按5.2.1提到的简化为最大值满足约束的方法，也由于需要首先得知流水线级数和遍历原图，难以实现。因此，在问题二中，使用对于同一执行路径上资源之和的约束，直接将其建模成一个非线性优化问题，采用启发式算法求解。

将所有的约束表达成一个惩罚函数 $F_{constraint}$ ，与原本的优化对象 $M = max(ri)(i = 1, 2, \dots, N)$ 相结合，使用如5.13的优化目标，其中 α 为惩罚项权重，

$$\min Mdd = (\alpha * F_{constraint} + M) \tag{5.13}$$

其中惩罚函数 $F_{constraint}$ 由依赖惩罚函数 $F_{dependency}$ 和资源限制惩罚函数 $F_{resource}$ 组成：

$$F_{constraint} = F_{dependency} + F_{resource} \quad (5.14)$$

其中依赖惩罚函数 $F_{dependency}$ 按5.15求解：

$$\begin{aligned} F_{dependency} &= Num(r_i > r_j \ (i = 0, 1, 2, 3, \dots, N) \ (r_j \in R_i^{\leq})) \\ &\quad + Num(r_i \geq r_j \ (i = 0, 1, 2, 3, \dots, N) \ (r_j \in R_i^{<})) \end{aligned} \quad (5.15)$$

对于 $F_{resource}$ 则分为相比问题一没有变化的资源约束 F_{old} 和执行路径相关的约束限制 F_{new} 。

$$\begin{aligned} F_{resource} &= F_{old} + F_{new} \\ F_{old} &= Num\left(\sum_{j=0}^M (t_i * x_{ij} - T_c) > 0\right) \\ &\quad + Num\left(\sum_{j=0}^M (x_{ij} * q_i - Q_c) > 0\right) \\ &\quad + Num\left(\sum_{j=0}^M (x_{ij} t_i + x_{i+16,j} \cdot t_{i+16}) > 1\right) \end{aligned} \quad (5.16)$$

对于 F_{new} , 即同一流水线下同一执行路径下的资源之和约束惩罚项之和。利用的方法，可以求出第 j 个流水线上不同执行路径 hash 和 alu 资源之和最大值 H_{maxj} 和 A_{maxj} :

$$\begin{aligned} F_{new} &= Num\left(\sum_j H_{maxj} > 2\right) \\ &\quad + Num\left(\sum_j A_{maxj} > 56\right) \\ &\quad + Num\left(\sum_j x_{ij} H_{maxi} + x_{i+16,j} \cdot H_{maxi+16} \leq 3\right) \end{aligned} \quad (5.17)$$

5.2.2 模型求解

该约束可使用包括模拟退火、差分进化、粒子群算法等启发式算法。其中模拟退火算法（Simulated Annealing, SA）是一种通用的优化算法，理论上算法具有概率的全局优化性能。

模拟退火来自冶金学的专有名词退火。模拟退火的原理也和金属退火的原理近似：我们将热力学的理论套用到统计学上，将搜寻空间内每一点想像成空气内的分子；分子的能量，就是它本身的动能；而搜寻空间内的每一点，也像空气分子一样带有“能量”，以表示该点对命题的合适程度。演算法先以搜寻空间内一个任意点作起始：每一步先选择一个“邻居”，然后再计算从现有位置到达“邻居”的概率。可以证明，模拟退火算法所得解依概率收敛到全局最优解 [5]。

模拟退火优化器经过 2.5h 的搜索，在有限的时间内求解出满足条件的最少流水线级数为 **51**。详细的代码见附录A.3，具体的结果见附件二。

6 模型评价

6.1 模型优点

本文通过对 PISA 架构的芯片资源排布问题的数学建模，将题目给出的 607 个基本块在满足约束条件前提下，在不同流水级进行排布，并实现流水线分配级数最小化，实现对芯片资源的合理调度和资源有效利用，主要优点有：

1. 数学限制完备：针对每个题目给出的限制条件均有对应的公式表示，在数学逻辑上对其有较强限制。
2. 求解算法精确：预处理算法功能完备，复杂度低。代码对求解结果限制较好，结果完全满足约束要求。
3. 数学形式优美：针对基本块之间的相互依赖关系建立了相关模型，通俗易懂，将依赖关系合理进行了数学表达。
4. 求解方便：利用 Cplex 的 CP 优化器完成了求解器的搭建，便于整个求解过程的代码完成。使用模拟退火启发求解非线性问题。

6.2 模型缺点

1. 由于约束规则复杂，整体程序时间较不稳定，运行效率不高。
2. 模拟退火启发式程序可控性不强，无解的情况比较容易出现，可能寻找到的不是全局最优解。

6.3 模型的改进与推广

1. 本文采取的策略可以广泛利用在资源配置以及时间顺序的排布问题上，并能够取得较好的效果。
2. 可以尝试采用其他的启发式算法以提高程序效率以及寻找全局最优解的能力。

参考文献

- [1] Bosshart P, Gibb G, Kim H S, et al., Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN, 43(4):99-110, 2013.
- [2] Bosshart P, Daly D, Gibb G, et al., P4: programming protocol-independent packet processors, 44(3):87-95, 2014.
- [3] 黄文奇, 何琨, 四维时空高效利用的装箱调度问题及其可计算性证明, 计算机学报, 36(9):9, 2013.
- [4] Appel A W, Modern Compiler Implementation in C: Basic Techniques, USA: Cambridge University Press, 1997.

[5] Wikipedia contributors, Simulated annealing — Wikipedia, The Free Encyclopedia, 2022.