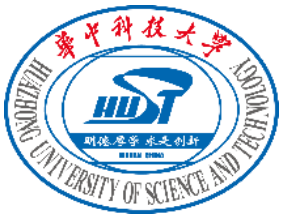


中国研究生创新实践系列大赛

中国光谷·“华为杯”第十九届中国研究生数学建模竞赛

学　　校	常州大学
参赛队号	22102920026
队员姓名	1.张铤波
	2.肖德豪
	3.陈佳璐

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　　PISA 架构芯片资源排布问题

摘　　要

PISA 架构芯片是当前主流的可编程交换芯片架构之一，应用前景广阔，对芯片资源排布问题的研究有利于提高芯片资源利用率，更好发挥芯片性能。本文围绕最优排布问题，在不同的约束条件下建立以排布流水线级数尽量短为优化目标的模型，结合启发式算法和优先级队列设计优化算法，最后通过编程计算得到排布结果。

针对问题一，要求在给定数据依赖、控制依赖和资源约束的条件下，给出最优化的排布算法和排布结果。建立模型时，以流水线总级数 n 为目标函数，以流水线总级数最少 $min\ n$ 为优化目标，**以依赖关系 Dw_i、Dr_i 和 $TCAM_m \leq 1$、$HASH_m \leq 2$、$ALU_m \leq 56$、$QUALIFY_m \leq 64$ 等资源限制为约束条件**，建立**单目标优化模型**。

求解模型时，首先利用 attachment2.csv 和 attachment3.csv 的信息，建立初始的有向无环图 g，并分别计算得到控制依赖和数据依赖；然后进行基本块重排，使得重排后的图 g_0 只与依赖关系有关；随后使用启发式算法决定排布顺序，计算从基本块 i 到运行结束所需的最少流水线级数 s_i 作为启发函数，优先排布启发函数最大的基本块，从而提高并行效率，缩短流水线级数；最后通过优先级队列实现排布过程，得到的排布结果保存在附件 result1.csv 中，部分结果如表 1 所示，占用的流水线级数为 44。

表 1　问题一的排布结果（部分）

流水线级数	分配的基本块编号				
0	365	15	16	17	…
1	172	170	11	12	…
…	…	…	…	…	…
44	97	88	-	-	-

分析结果时，计算了每级流水线的资源利用率，发现 HASH 资源、ALU 资源和 TCAM 资源的利用率都较高，说明本文的排布算法能充分利用芯片资源，而 QUALIFY 资源利用率较低，说明它的资源比较充足。分析了模型灵敏度，发现模型对 HASH 资源和 ALU 资源灵敏，增加资源能大幅减少占用级数，说明流水线中的这两种资源存在瓶颈。分析算法的空间复杂度为 $O(n^2)$，空间复杂度为 $O(n^2)$。

针对问题二，增加了共享资源条件，即不在一条执行流程上的基本块，可以共享 HASH

资源和 ALU 资源。建立模型时，将每级的 HASH 资源和 ALU 资源占用修改为 $HASH_m'$ 和 ALU_m'，以流水线总级数最少 $\min n$ 为优化目标，**以数据依赖、控制依赖和资源限制为约束条件**，建立**单目标优化模型**。模型求解时，修改 HASH 资源和 ALU 资源占用的计算方式。在依赖关系下，利用启发式算法选择流程长的基本块进行优先排布，使总的流水线级数尽量短。得到的排布结果保存在附件 result2.csv 中，部分结果如表 2 所示，占用的流水线级数为 32。

表 2 问题二的排布结果（部分）

流水线级数编号	分配的基本块编号				
0	365	15	16	17	…
1	172	170	11	12	…
…	…	…	…	…	…
32	97	88	-	-	-

分析问题二结果时，计算每级流水线的资源利用率，发现相对问题一的结果，问题二中的 HASH 资源和 ALU 资源利用率下降，说明资源共享减轻了这两种资源在流水线中的负担。分析了模型灵敏度，发现模型对 HASH 资源和 ALU 资源的灵敏度有所降低，说明资源瓶颈得到了缓解。分析算法的分析算法的空间复杂度为 $O(n^2)$，空间复杂度为 $O(n^2)$。

关键词：PISA；资源排布；基本块重排；优先级队列；启发式算法

目录

一、问题重述

1.1 问题背景

在当前多种高科技技术被垄断的形势下，我国的芯片技术亟待突破。芯片是指封装的集成电路，是电子行业的重要基础。当新的网络协议出现时，传统的交换芯片由于功能固定，因此必须重新设计，这样会造成大量的资源浪费，而可编程交换芯片的出现解决了这个问题，它不仅拥有和传统交换芯片相当的处理速度，而且兼备可编程性，有着广泛的应用前景。

PISA 是可编程交换芯片的一种，结构上包括报文解析、多级报文处理流水线以及报文重组三个部分。本问题只关注多级报文处理流水线部分，其中报文是指网络通信中的数据包；流水线由一些基本块串联而成，不同的芯片流水线的级数可能不同；基本块是源程序的一个程序片段。在 PISA 架构编程模型中，用户使用 P4 语言[1]得到 P4 程序，接着用编译器编译 P4 程序，最后得到可以在芯片上执行的机器码。当编译 P4 程序时，P4 程序被划分为一些基本块，接着将得到的基本块排布到流水线各级中。因为一系列基本块会占用一定的芯片资源，所以 PISA 架构芯片资源排布问题即简化成基本块的排布问题。因为流水线各级的资源和流水线各级之间的资源存在着多种约束条件，所以基本块排布问题变得复杂且困难。提高芯片的资源利用率，能够帮助我们更好的发挥芯片的性能，让芯片应用到更多的场景中。

1.2 问题描述

基本块需要执行指令来完成计算，指令执行时有两个操作：读取指令源操作数（简称读操作）和计算结果赋值给目的操作数（简称写操作）。在划分好的基本块中，每个指令都是并行执行的，执行均按照先读后写的顺序。值得注意的是，每个基本块只会给同一个变量赋值一次。基本块可以被抽象成数据结构中的图结点，并忽略结点中的具体指令，只保留读和写的信息。当两个结点有顺序时，用一条有向边从先执行的结点指向后执行的结点，P4 程序经过图抽象变成了一个有向无环图（注：P4 程序没有循环）。PISA 架构资源排布问题可以抽象成图中的各结点（即各基本块）在满足三种约束条件下排布到各级流水线中。三种约束分别是数据依赖、控制依赖和芯片资源约束。其中，数据依赖和控制依赖约束了基本块所在流水线的级数的大小关系。芯片的资源包括 TCAM、HASH、ALU、QUALIFY 四类。每一类资源都是严格的限制，资源排布时不能超出限制。在附件中给出了各基本块在程序中的邻接关系、每个基本块占用的四类资源的情况，以及每个基本块的读写情况。

1.3 问题要求

基于上述背景，题目给出了三个约束条件的数据文件，本文需要研究解决以下问题：

问题 1：以占用尽量短的流水线级数为优化目标

题目给出了资源约束条件，主要包括几种类型：（1）流水线每级的 TCAM、HASH、ALU 和 QUALIFY 资源最大值分别为 1、2、56、64；（2）任意流水级数相差 16 级为折叠级数，总流水线级数大于等于 32 不考虑折叠资源限制，小于 32 则存在折叠资源限制，具体限制为：折叠的两级 TCAM 之和最大为 1，HASH 之和最大为 3；（3）有 TCAM 资源的偶数级不超过 5；（4）每个基本块只能在流水线一级中。

在上述资源约束条件下进行资源排布，优化目标为占用尽量短的流水线级数。给出资源排布算法，并以特定的格式输出基本块排布结果。

问题 2：在资源共享下，占用尽量短的流水线级数

题目给出的资源约束条件，主要包括：（1）流水线每级的 TCAM 和 QUALIFY 资源最大值分别为 1 和 64；（2）流水线每级中在同一条执行流程上的基本块的 HASH 资源之和和 ALU 资源之和最大分别为 2 和 56；（3）折叠的两级 TCAM 之和最大为 1，分别计算每级中在同一条执行流程上基本块的 HASH 资源之和，两级的 HASH 资源之和再求和结果，其最大为 3；（4）有 TCAM 资源的偶数级个数不超过 5；（5）每个基本块只能在流水线一级中。在上述资源约束条件下进行资源排布，优化目标为占用尽量短的流水线级数。请给出资源排布算法，并以特定的格式输出基本块排布结果。

二、问题分析

问题一分析：

问题一本质上可以通过单目标优先模型进行求解，P4 程序每个基本块均会对一部分变量赋值和读取一部分变量，若不同的基本块在对同一变量进行操作时会产生一定数据依赖。数据依赖的存在会影响基本块之间执行的先后顺序，从而影响到两者之间排布的流水线级数关系。

基本块执行完后可能跳转到多个基本块执行，会产生控制依赖也同样会影响两者之间排布的流水线级数关系。

由于资源有限在进行资源排布时，需要满足芯片的资源约束，因此，在进行 PISA 架构资源排布问题可以转化为各基本块在满足三种约束条件下排布到各级流水线中。芯片的资源利用率越高，芯片的性能发挥得就越好，因此资源排布算法应该注重资源的利用率，优先考虑资源利用带来的影响，避免造成资源的浪费。

计划从资源的角度出发，构建一个优先级队列存放基本块。最后采用启发式算法在满足给定问题一中数据依赖、控制依赖以及各具体子问题的资源约束条件条件下对计算流水线级数推理，从而找到最优解，以求最大化芯片资源利用率。

问题二分析：

问题二引入不在一条执行流程上的基本块可以共享 HASH 资源和 ALU 资源的条件，并重新更改了部分资源约束条件。然而，问题二中 PISA 架构资源排布问题的数学模型可同问题一转化为各基本块在满足三种约束条件下排布到各级流水线中。

因为共享资源的存在，基本块在进行资源排布时，应优先考虑不可共享资源 TCAM、QUALIFY 对基本块选取优先度的影响，应当建立不同于问题一模型的优先级队列。最后同样采用启发式算法，利用在数据依赖和控制依赖下隐藏的流水线级数信息，在问题二给定新的资源约束条件下对计算流水线级数推理，从而找到最优解。

三、模型假设

（1） 假设在满足资源约束条件下，一级流水线上可以排布的基本块数量没有限制；
（2） 假设只要基本块的排布满足读写依赖、控制依赖，P4 程序就能正常运行；
（3） 资源排布只考虑所给约束条件的影响，与芯片的其它部分无关；
（4） 针对问题二中的条件，假设资源可共享时，共享同一资源的基本块数量不受限制。

四、符号说明

符号	含义
a_i	编号为 i 的基本块
$f(\cdot)$	排布方式映射
n	占用的最大流水线级数
Dw_i	与a_i存在写后读依赖或写后写依赖的基本块集合
Dr_i	与a_i存在读后写依赖的基本块集合
C_i	与a_i存在控制依赖的基本块集合
$TCAM_m$	第 m 级流水线占用的 TCAM 资源数
$HASH_m$	第 m 级流水线占用的 HASH 资源数
ALU_m	第 m 级流水线占用的 ALU 资源数
$QUALIFY_m$	第 m 级流水线占用的 QUALIFY 资源数
$HASH_m{}'$	资源共享条件下，第 m 级流水线占用的 HASH 资源数
$ALU_m{}'$	资源共享条件下，第 m 级流水线占用的 ALU 资源数

五、模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 问题一模型的分析

PISA 架构资源排布问题的数学模型可以转化为各基本块在满足三种约束条件下排布到各级流水线中，求解流水线级数最短的情况。首先根据流程图中的邻接基本块信息表可以得出各基本块是否存在控制依赖。其次结合各基本块读写的变量信息表可以得到各基本块的数据依赖，从而可以获得部分基本块之间排布的流水线级数信息。最后采用启发式算法利用得到的流水线级数信息在满足给定的资源约束条件下对计算流水线级数推理从而找到最优解。

5.1.2 问题一模型的建立

根据 PISA 架构芯片的资源排布问题分析，可以建立单目标优化模型[2,3]，约束条件为数据依赖、控制依赖、资源约束，优化目标为排布的流水线级数最小。

- **优化目标**

问题要求为基本块分配芯片的流水线级数，并且一个基本块只能排布到一个流水线级里，如图 1 所示，设一个基本块为a_i，可以建立抽象的排布方式$f(\cdot)$，如$f(a_i)=m$表示将a_i分配到编号为 m 的流水线中。

流水线级数从 0 开始编号，则需要的最大级数 n 为：

$$n=\max_i f(a_i) \tag{1}$$

优化目标为流水线级数尽量短，即：

$$\min n \tag{2}$$

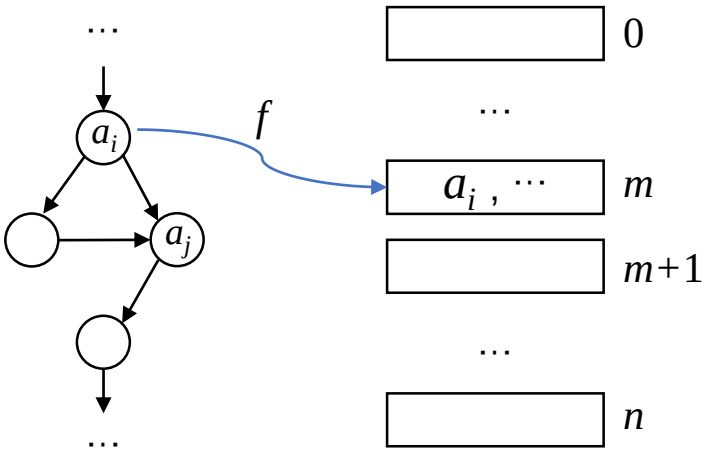


图 1 基本块排布示意图

- **约束条件一：数据依赖**

在数据依赖上，若两个程序基本块在同一条路径上，且它们对同一个存储单元进行了读写操作，则这两个基本块之间可能存在写后读、写后写、读后写这三种数据依赖关系。对于结点a_i，用集合Dw_i和Dr_i表示数据依赖关系：

$$Dw_i = \{a_j | a_j在a_i下游,且与a_i存在写后读或写后写依赖\} \tag{3}$$

$$Dr_i = \{a_j | a_j在a_i下游,且与a_i存在读后写依赖\} \tag{4}$$

对于写后写和写后读关系，严格要求先写的基本块级数小于后写或后读的基本块级数：

$$f(a_j) > f(a_i),\ a_j \in Dw_i \tag{5}$$

对于读后写关系，要求先读的基本块级数小于等于后写的基本块级数，两个基本块可以排布到同一流水线级：

$$f(a_j) \geq f(a_i),\ a_j \in Dr_i \tag{6}$$

- **约束条件二：控制依赖**

由于分支选择的存在，基本块之间可能存在控制依赖关系。根据定义，当基本块a_i的部分路径通过a_j时，两者存在控制依赖，用集合C_i表示控制关系：

$$C_i = \{a_j | a_j在a_i下游,且与a_i存在控制依赖\} \tag{7}$$

当a_i与a_j存在控制依赖关系，即 $a_j \in C_i$时，要求a_j排布的级数大于等于a_i排布的级数：

$$f(a_j) \geq f(a_i),\ a_j \in C_i \tag{8}$$

式(8)与式(6)可以合并为：

$$f(a_j) \geq f(a_i),\ a_j \in Dr_i \cup C_i \tag{9}$$

- **约束条件三：资源约束**

芯片流水线的资源是有限的，每级流水线存在资源限制。每级流水线的 TCAM 资源数不超过 1：

$$TCAM_m \le 1 \tag{10}$$

每级流水线 HASH 资源数不超过 2：

$$HASH_m \le 2 \tag{11}$$

每级流水线 ALU 资源数不超过 56：

$$ALU_m \le 56 \tag{12}$$

每级流水线 QUALIFY 资源数不超过 64：

$$QUALIFY_m \le 64 \tag{13}$$

式中 m 表示第 m 级流水线$(m = 0,1,\dots,n)$。

流水线的第 0 级到第 15 级与第 16 级到第 31 级分别为折叠级数，使得前 32 级流水线的 TCAM 资源和 HASH 资源存在约束：

$$\begin{cases} TCAM_m + TCAM_{m+16} \le 1 \\ HASH_m + HASH_{m+16} \le 3 \end{cases}, m = 0,1,\dots,15 \tag{14}$$

问题 1 还约定偶数级的 TCAM 资源较多，每级数量不超过 5。但是根据式(14)，由于折叠级数的存在，第 0 级到第 31 级流水线的TCAM ≤ 1，所以此项约束只需考虑第 32 级及之后的流水线，此项约束为：

$$TCAM_m \le 5, (m = 2k, k \ge 16) \tag{15}$$

综合约束条件和目标函数，问题一的目标优化模型如下：

$$\min n$$

$$s.t.\begin{cases} n = \max_i f(a_i) \\ f(a_j) > f(a_i), a_j \in Dw_i \\ f(a_j) \ge f(a_i), a_j \in Dr_i \cup C_i \\ \begin{cases} TCAM_m + TCAM_{m+16} \le 1, m = 0,1,\dots,15 \\ TCAM_m \le 5, (m = 2k, k \ge 16) \\ TCAM_m \le 1, \text{其它} \end{cases} \\ \begin{cases} HASH_m \le 2, m = 0,1,\dots \\ HASH_m + HASH_{m+16} \le 3, m = 0,1,\dots,15 \end{cases} \\ ALU_m \le 56 \\ QUALIFY_m \le 64 \end{cases} \tag{16}$$

5.1.3 问题一模型的求解

本问题是在满足控制依赖和数据依赖的前提下进行资源排布，所以我们必须先计算出各基本块之间的控制依赖和数据依赖。这两种依赖关系能决定各基本块之间排布的流水线级数大小关系，生成对应流水线级数大小关系图。然后结合问题一给定的资源约束条件，计算出每个子图所占最小的级数要求并以此为权重构建出进行资源排布时的优先级队列。存在优先级队列越靠前的基本块将优先进行资源排布，选择合适的级数。问题一模型流程如图 2 所示。

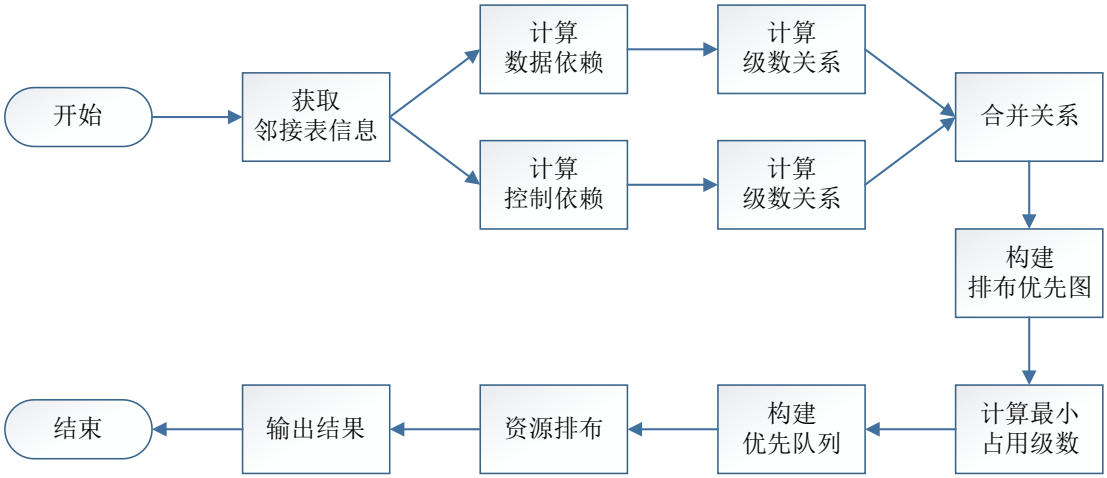


图 2　问题一模型求解整体流程图

（1）数据分析

如图 3 所示，通过统计基本块不同资源占用情况，可以观察出大部分基本块占用资源少，只存在少部分占用大量资源的基本块数据。从最大化芯片资源利用率的角度出发，可以优先排布占用资源较大的基本块。

图 3　基本块占用资源统计

（2）计算数据依赖关系

附件提供了流程图的邻接表形式数据，根据邻接表可以重建有向无环图。首先从 attachment3.csv 获取所有基本块在流程图中的邻接信息，其次通过邻接表把所有基本块分别存储为一个节点，最后通过邻接表把所有基本块之间的关系存储为边，从而构建基本块执行时的关系表。

当基本块 A 在基本块 B 之前执行时，基本块 A 与基本块 B 之间才可能存在某种数据依赖关系。数据依赖关系影响了基本块的排布顺序，因此要计算基本块之间的数据依赖关

系。流程如图 4 所示，步骤如下：

Step1：查询 attachment3.csv 获得基本块 A 和 B 之间的执行顺序；

Step2：当基础块 B 处于 A 下游时，查询 attachment2.csv 获得基本块 A 和 B 读写的变量信息；

Step3：根据数据依赖判定条件获到基本块 A 和 B 的数据依赖关系，如：写后读、写后写、读后写以及无数据依赖。

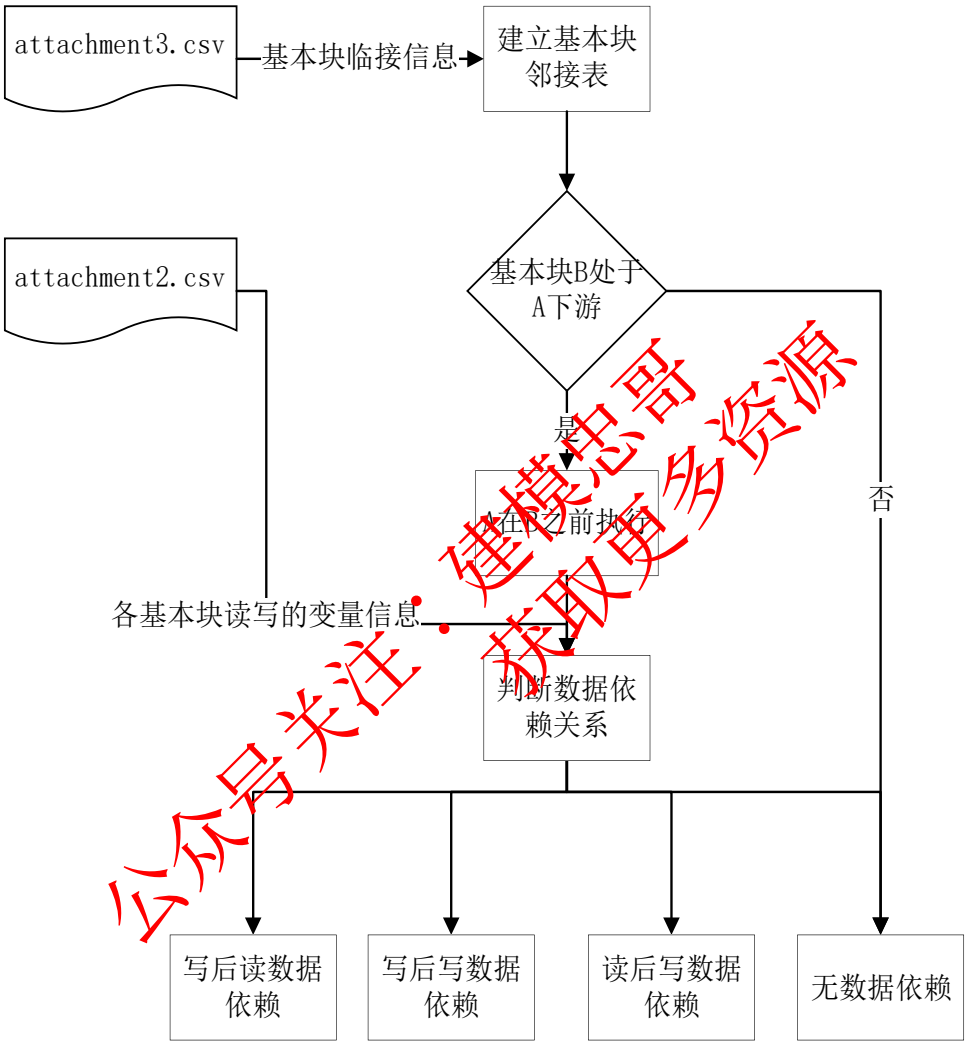


图 4 计算数据依赖关系流程图

（3）计算控制依赖关系

同理，计算基本块之间的控制依赖关系，基本步骤为：

Step1：查询基本块 A 和基本块 B 的邻接关系，基本块 B 处于 A 的下游则执行 Step2，否则得出二者不存在控制依赖关系；

Step2：判断从基本块 A 出发的路径是否只有部分路径通过下游基本块 B，若是，则基本块 A 和基本块 B 之间存在控制依赖关系，否则不存在控制依赖关系。

流程图如图 5 所示。

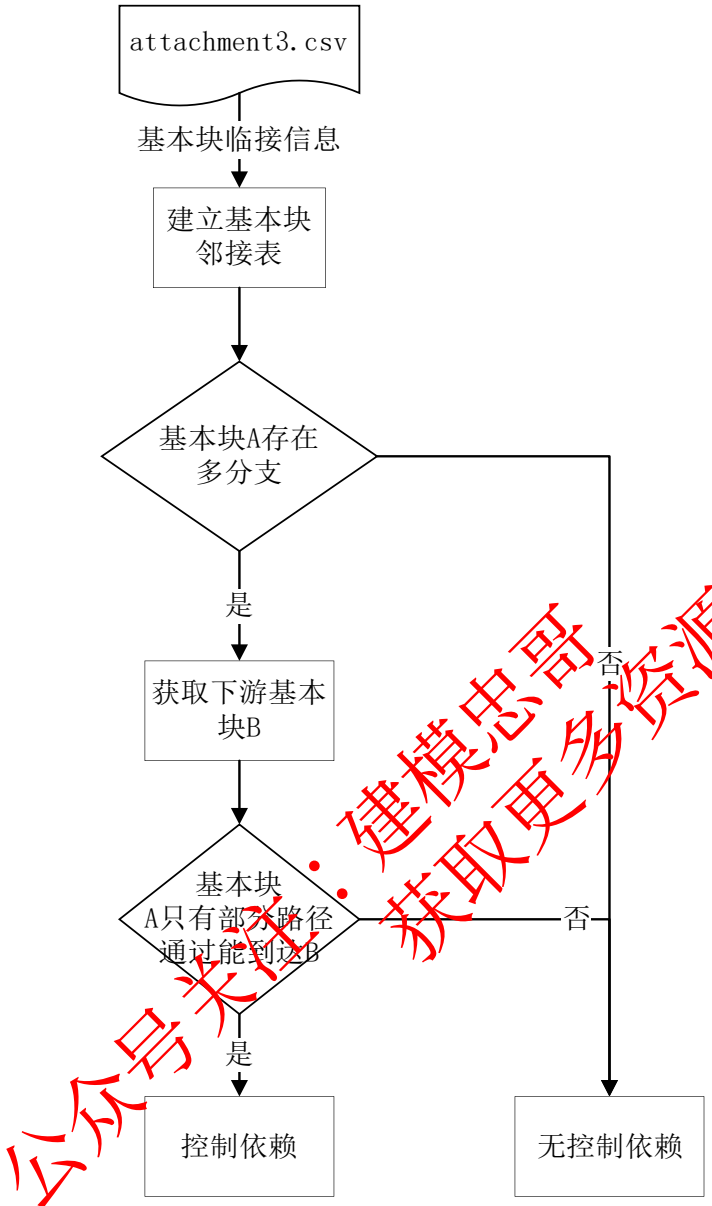


图 5 计算控制依赖关系流程图

（4）计算排布的流水线级数大小关系

根据基本块 A 和 B 的不同依赖关系（包括数据依赖和控制依赖）可以得出基本块 A 和 B 之间排布的流水线级数大小关系。

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<基本块 B 排布的级数；

当基本块 A 与基本块 B 存在控制依赖，则 A 排布的流水线级数≤B 排布的流水线级数；其他情况下则无排布的流水线级数大小关系。

最后通过遍历所有基本块可以得到两张流水线级数大小关系表。

过程如图 6 所示。

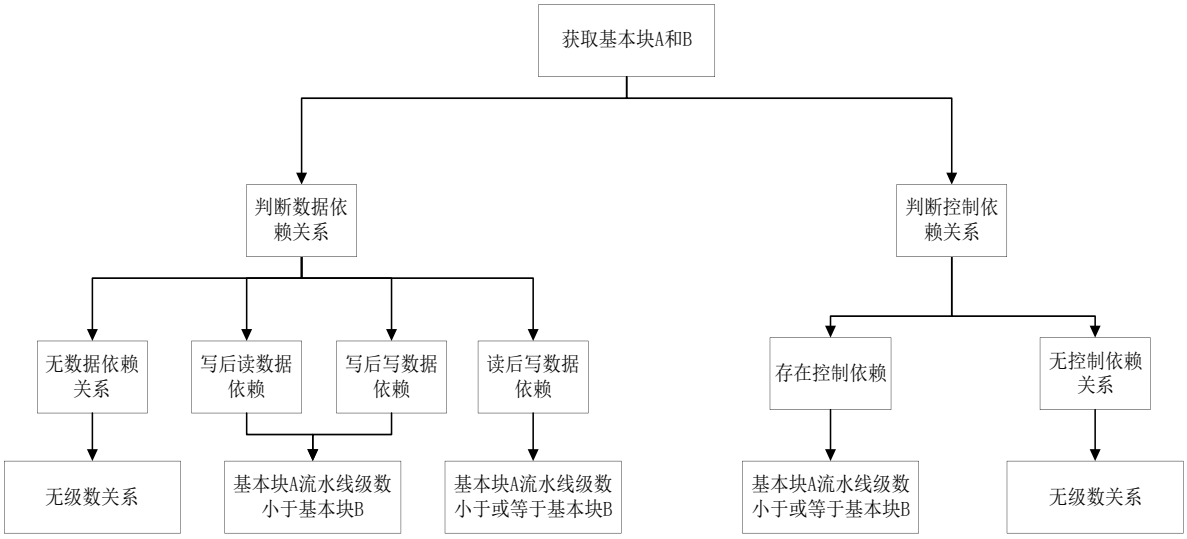


图 6 计算流水级数大小关系

（5）构建排布优先队列

通过两张流水线级数大小关系表获得一张总的关系表，合并操作如下处理：

① 基本块 A 流水线级数≤基本块 B 与基本块 A 流水线级数＜基本块 B 合并为基本块 A＜基本块 B；

② 基本块 A 流水线级数＜基本块 B 与无级数关系的 A、B 合并为基本块 A 流水线级数＜基本块 B；

③ 基本块 A 流水线级数≤基本块 B 与无级数关系的 A、B 合并为基本块 A 流水线级数≤基本块 B。

合并之后建立优先级队列用于存放基本块进行资源排布的优先度关系，越靠前的基本块将优先进行资源排布。

a. 构建排布优先图

依据得到级数关系表，可以构建基本块之间的排布优先图[4,5]，图中每一子图的根节点排布的流水线级数都小于或等于该子图所有其它节点。

具体步骤如下：

Step1：把各基本块存储为一个节点，

Step2：当基本块之间存在级数大小关系时，存储其关系存储为边，流水线级数大小关系存储到边权。最后得到邻接表存储的排布优先图。

b. 计算排布优先图子图所占用最小流水线级数

具体步骤如下：

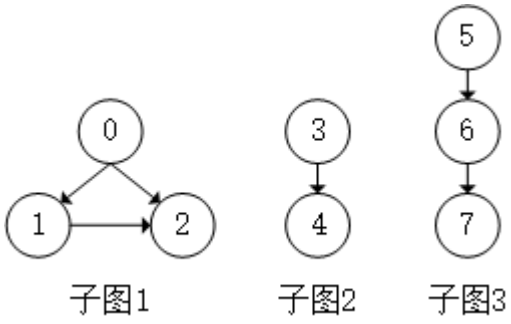
Step1：依据得到的排布优先图存储的邻接表，划分出排布优先图中所有子图；

Step2：查找数据文件 attachment1.csv 中各基本块需要的资源信息计算出每个子图中所有节点每种资源共占总数。

Step3：计算出每个子图在满足问题一中每级流水线资源约束条件下最小的流水线级数要求。

给出一个例子的步骤示例：

① 构建划分排布优先图子图



② 计算出每个子图中所有节点每种资源总数

BLOCK	TCAM	HASH	ALU	QUALIFY
0	0	2	11	0
1	0	1	9	0
2	0	1	6	0
3	1	0	7	0
4	1	0	7	1
5	0	1	11	0
6	0	0	4	0
7	0	0	1	1

③ 计算每一子图最小流水线级数要求

子图	TCAM	HASH	ALU	QUALIFY	最小占用级数
1	0	4	26	0	3
2	2	0	14	1	2
3	0	1	16	0	1

c. 构建排布优先队列

构建优先级队列 queue 用于存储子图 ALG_i 的根节点 ALG_i-root，排列权值为根节点 ALG_i-root 所在子图 ALG_i 所占最小的流水线级数，操作如图 7 所示。

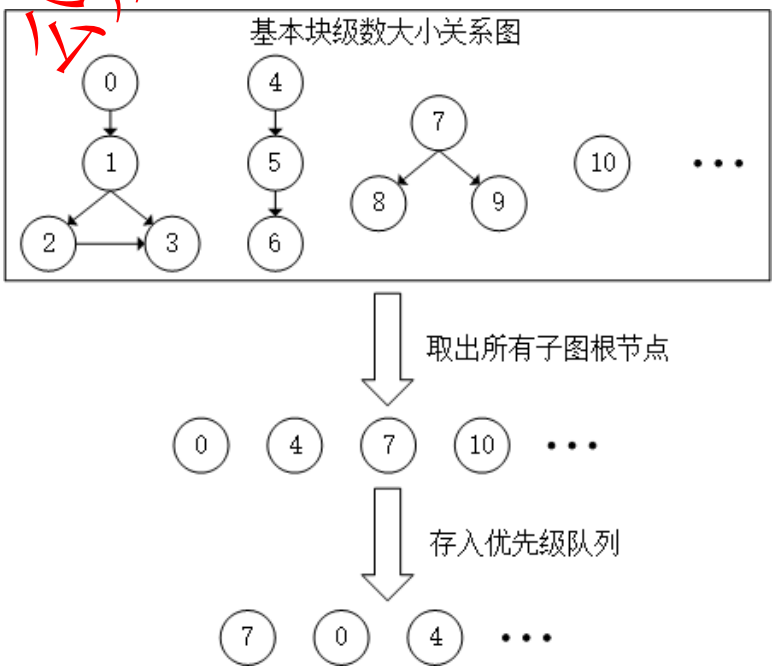


图 7 构建排布优先队列示例图

（6）资源排布

根据构建的优先级选择基本块优先进行资源排布，具体流程图如图 8 所示。进行资源排布前已分配一级流水线，当前排布优先队列首节点 queue-N，构建临时队列 queue2，用于存放取出的节点，以节点 queue-N 为根节点的图为 ALG_queue-N。

具体步骤如下：

Step1：当排布优先队列不为空时，取出当前首节点 queue-N，为空时结束；

Step2：将节点填入符合条件的一级流水线。若填入成功则执行 Step3，若填入失败即不存在一级流水线符合要求执行 Step4；

Step3：记录填入流水线级数，将图 ALG_queue-N 去掉节点 queue-N 后所有子图 ALG_NO_queue-Ni 根节点依次放入排布优先队列（如图 9 所示），排列权值为满足对应子图 ALG_NO_queue-Ni 所有节点资源约束下的最小流水线级数要求；

Step4：若此时排布优先队列 queue 为空，则增加一级流水线并存入节点 queue-N，临时队列 queue2 所有节点放入排布优先队列 queue 执行 step2；若排布优先队列 queue 不为空，将 queue-N 存入临时队列 queue2，取出队首节点 ALG_queue-N2 并执行 Step2。

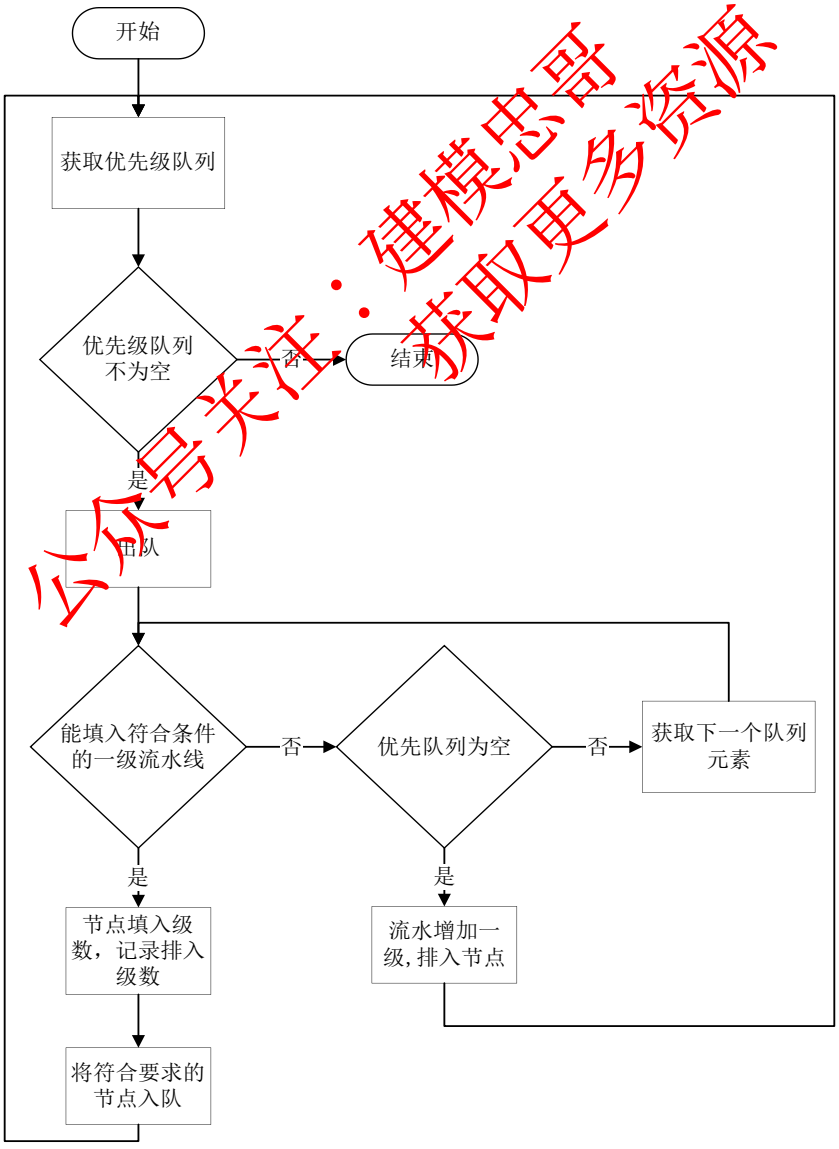


图 8　资源排布流程图

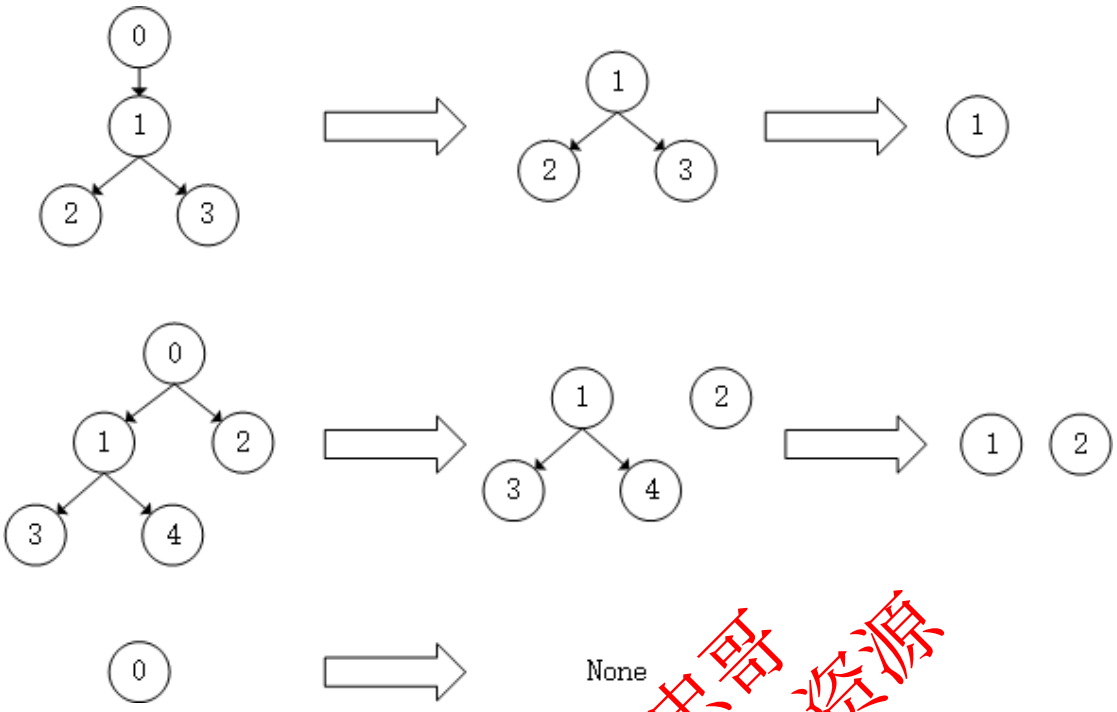


图 9 去除根结点生成图所有子图根节点依次放入排布优先队列示例图

5.1.4 问题一结果的分析

（1）结果分析

在问题一所给资源约束条件下进行资源排布，输出的结果如表 3 所示：

表 3 部分流线资源排布

流水线级数编号	分配的基本块编号				
0	365	15	16	17	…
1	172	170	11	12	…
2	162	390	373	391	…
3	164	153	382	383	…
4	156	530	531	532	…
5	563	564	554	541	…
6	555	583	-	-	-
7	534	585	569	570	…
…	…	…	…	…	…
44	97	88	-	-	-

表 3 中第一列为流水线级数编号，每行对应所在流水线级分配的基本块编号。从表中流水线各级排布信息结合各基本块使用的资源信息，可以计算出每级流水线四种资源的利用率，如图 10。

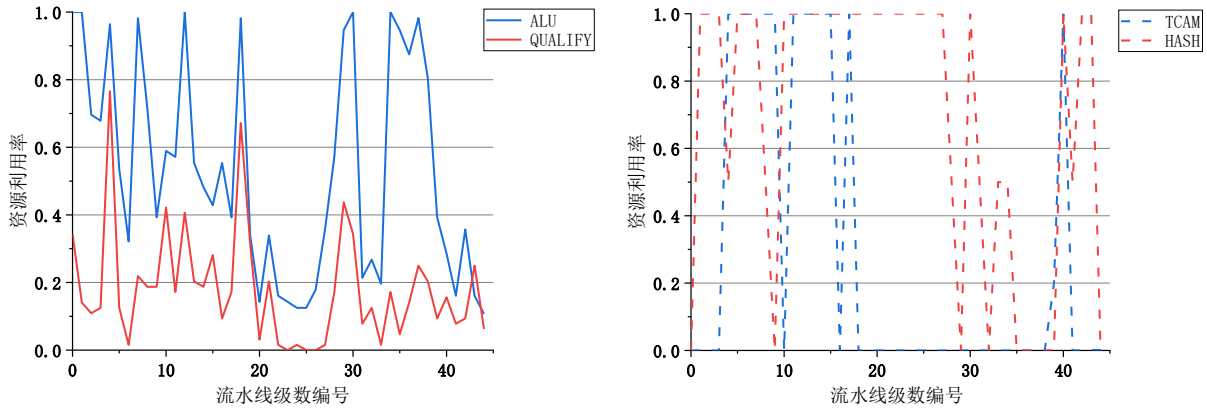


图 10 排布流水线每级 ALU、QUALIFY、TCAM、HASH 资源利用率

通过图 10 发现，在各级流水上四种资源的资源利用率都比较高，符合预期在最大化芯片资源利用率的同时去优化占用的流水线级数尽量短的目标。但流水线部分级也存在四种资源利用率都较小的情况，表明未能将资源利用率达到最优，模型算法改进的地方。

（2）结果仿真

为了检验资源排布结果的正确性，即是否依然满足三个约束，我们抽出部分基本块进行验证。图 11（a）是根据基本块邻接表信息建立的有向图，图 11（b）是所取基本块存在的控制依赖，图 11（c）是所取基本块存在的数据依赖，接着分析流水线级里的基本块是否满足流水线等级大小关系，以及所在的流水线级是否满足资源约束条件，最终判断问题一排布结果的正确性。

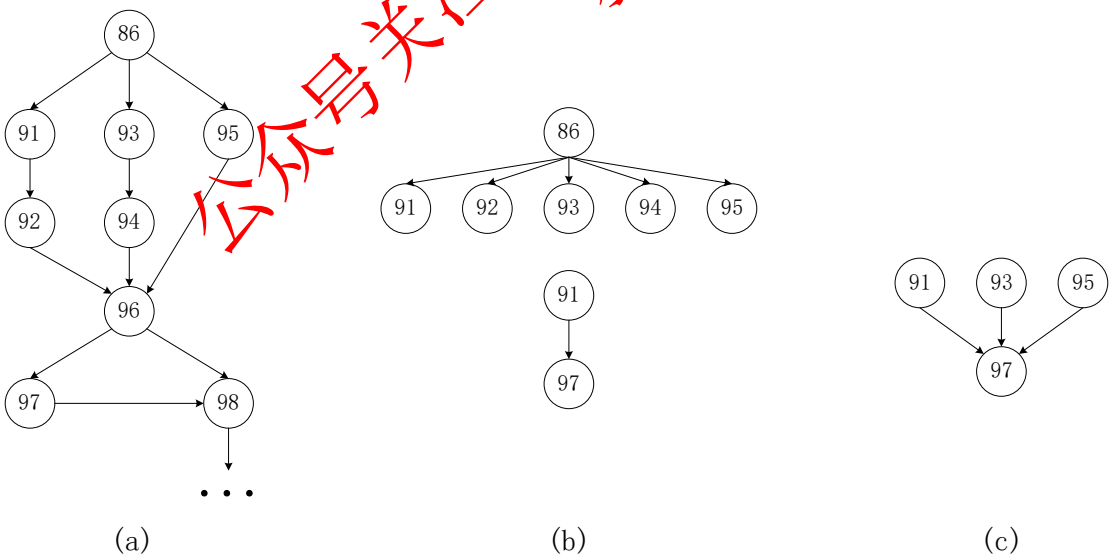


图 11 仿真实验基本块不同关系图

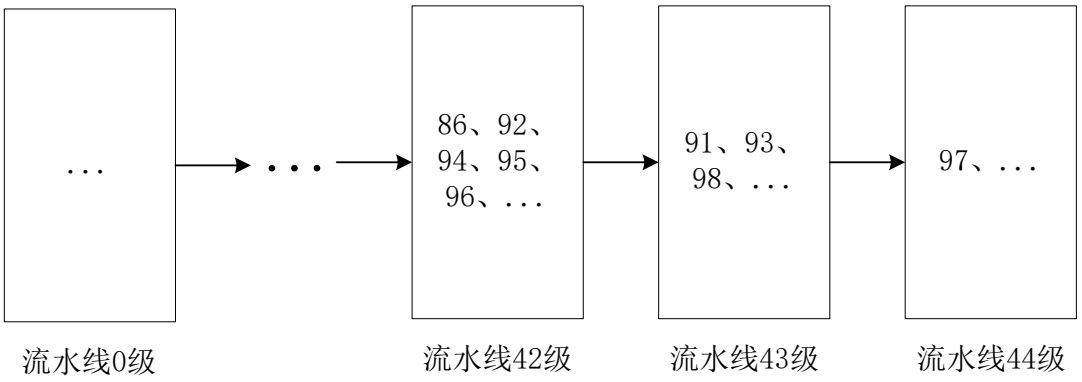


图 12 仿真实验流水线排布结果展示

从排布结果图（图 12）中可知，流水线级里的基本块满足控制依赖和数据依赖带来的流水线等级大小关系。我们还需计算 38 级、39 级和 40 级内的所有基本块资源之和是否满足资源约束条件。这 3 级流水线资源占用情况如表 4 所示。

表 4 流水线资源占用情况

流水线级数	TCAM	HASH	ALU	QUALIFY
42	0	2	20	6
43	0	2	9	16
44	0	0	5	4

从表 4 可知，随机抽取基本块所在的流水线满足资源约束。靠后的流水级资源仅仅 HASH 达到最大值，其他的三个资源仍然有填入空间，算法还存在别的改进点。

（3）模型灵敏度分析

针对问题一的资源约束条件，对 TCAM、HASH、ALU 和 QUALIFY 四个资源的最大值做灵敏度分析，分析图如下所示。

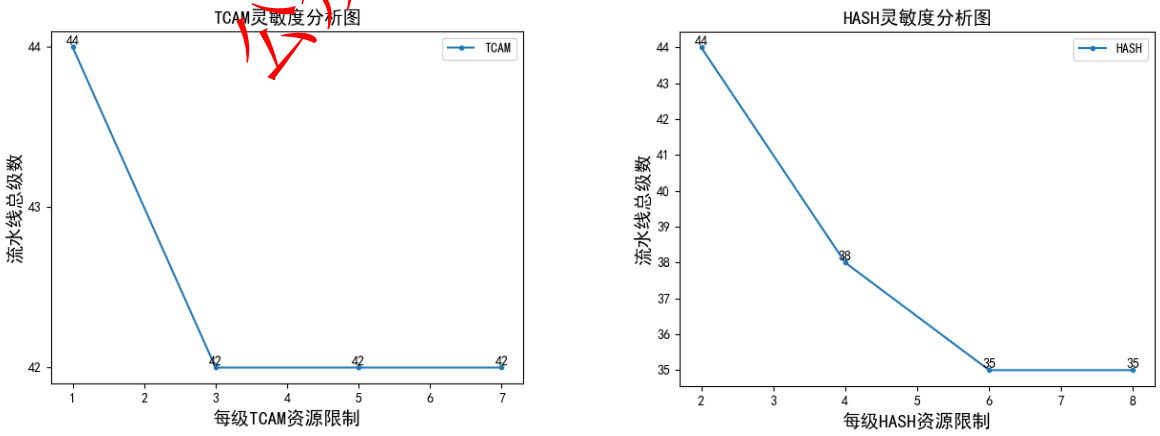


图 13 TCAM、HASH 资源灵敏度分析图

图 13（左）展示了 TCAM 资源约束最大值和流水线总级数的关系，当 TCAM 资源约束最大值从 1 递增时，流水线总级数缓慢降低，当 TCAM 资源约束增加到 3 后，流水线总级数就受 TCAM 资源约束的影响非常小，表明流水线总计数对 TCAM 资源敏感度低。

图 13（右）展示了 HASH 资源约束最大值和流水线总级数的关系，因为前 32 级存在着折叠限制，为了控制变量所以对折叠两级的最大 HASH 也做了相应调整。当 HASH 资源

约束最大值从 2 开始递增时，流水线总级数快速降低，当 HASH 增大到 6 后，流水线总级数保持不变。当 HASH 资源约束最大值较小时，流水线总级数对其更加敏感。

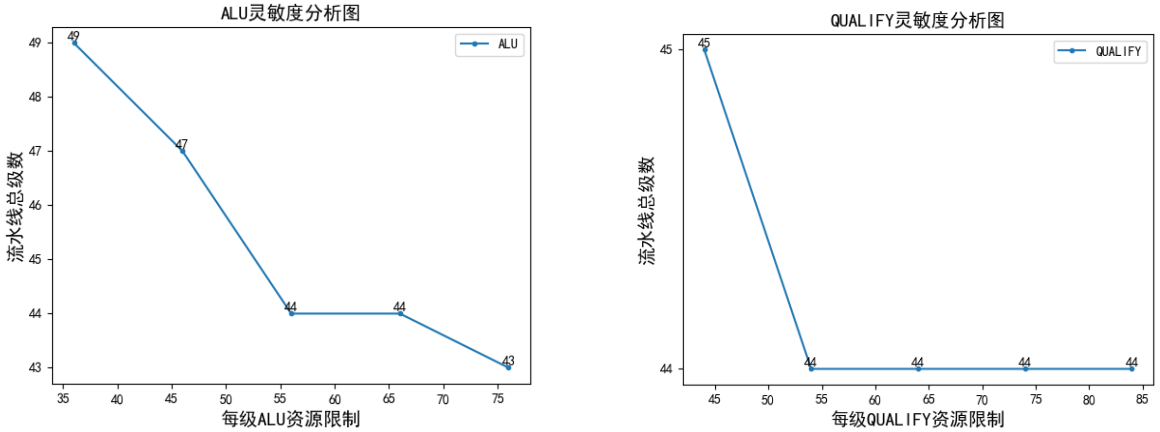


图 14 ALU、QUALIFY 资源灵敏度分析图

图 14（左）展示了 ALU 资源约束最大值和流水线总级数的关系，当 ALU 资源约束最大值从 36 开始递增时，流水线总级数快速降低，当 ALU 在区间[56,66]时，流水线总级数保持为 40 级，当继续增大时，流水线总级数还会降低。表明流水线总计数对 ALU 资源敏感度高。

图 14（右）展示了 QUALIFY 资源约束最大值和流水线总级数的关系，当 QUALIFY 资源约束最大值从 44 开始变大时，流水线总级数缓慢降低，当 QUALIFY 资源约束最大值增大到 54 时，流水线总级数不变。表明流水线总计数对 QUALIFY 资源敏感度低。

5.1.5 算法复杂度分析

（1）存储复杂度分析

模型建立的时候使用 attachment3.csv 文件读取各基本块在流程图中的邻接基本块信息建立邻接表，其中结点数量为 n 边数为 e，其存储复杂度为 O(n+e)。模型采用二维数组存储各基本块之间排布的流水线级数大小关系，其存储复杂度为 $O(n^2)$。模型采用优先级队列存储基本块，其存储复杂度为 O(n)。经比较可知，模型最大空间复杂度为 $O(n^2)$。

（2）运算复杂度分析

算法在计算依赖关系时，对 n 个结点进行了遍历，并通过深度优先遍历下游结点，判断依赖关系，时间复杂度为 $O(n^2)$。计算启发函数时，从每个结点依次进行深度优先遍历，计算子图占用资源，时间复杂度为 $O(n^2)$。进行排布时，每个结点排布一次，时间复杂度为 O(n)。模型的时间复杂度为 $O(n^2)$。

5.2 问题二模型的建立与求解

5.2.1 问题二模型的分析

问题二下 PISA 架构资源排布问题的数学模型可同问题一转化为各基本块在满足三种约束条件下排布到各级流水线中。由于共享资源的存在，基本块在进行资源排布时应优先考虑不可共享资源 TCAM、QUALIFY 的影响，缔造不同问题一模型的优先级队列。最后同样采用启发式算法利用得到的流水线级数信息在新的资源约束条件下对计算流水线级数推理从而找到最优解。

5.2.2 问题二模型的建立

问题二的模型与问题一类似，可以建立单目标优化模型。

- **优化目标**

问题二的优化目标与问题一相同，为占用流水线级数 n 最短：

$$\min n \tag{17}$$

级数 n 的结果通过对基本块a_i的排布方式$f(\cdot)$来确定：

$$n = \max_i f(a_i) \tag{18}$$

- **约束条件一：资源限制**

问题二增加了 HASH 资源和 ALU 资源可共享这一条件。计算每级资源占用时，共享的资源只计算一次，调整后的每级 HASH 资源和 ALU 资源占用为$HASH_m{}'$ 和$ALU_m{}'$，对应的资源约束为：

$$\begin{cases} HASH_m{}' \le 2, m = 0,1,... \\ HASH_m{}' + HASH_{m+16}{}' \le 3, m = 0,1...,15 \end{cases} \tag{19}$$

$$ALU_m{}' \le 56 \tag{20}$$

TCAM 资源和 QUALIFY 资源的占用计算方法和约束条件不变：

$$\begin{cases} TCAM_m + TCAM_{m+16} \le 1, m = 0,1...,15 \\ TCAM_m \le 5, (m = 2k, k \ge 16) \\ TCAM_m \le 1, 其它 \end{cases} \tag{21}$$

$$QUALIFY_m \le 64 \tag{22}$$

- **约束条件二：数据依赖**

问题二的数据依赖约束条件与问题一相同，对于写后写和写后读关系，严格要求先写的基本块级数小于后写或后读的基本块级数：

$$f(a_j) > f(a_i),\ a_j \in Dw_i \tag{23}$$

式中Dw_i表示与a_i存在写后写和写后读依赖关系的基本块的集合。

对于读后写关系，要求先读的基本块级数小于等于后写的基本块级数，两个基本块可以排布到同一流水线级：

$$f(a_j) \ge f(a_i),\ a_j \in Dr_i \tag{24}$$

式中Dr_i表示与a_i存在读后写依赖关系的基本块的集合。

- **约束条件三：控制依赖**

问题二的控制依赖约束与问题一相同：

$$f(a_j) \ge f(a_i),\ a_j \in C_i \tag{25}$$

式中C_i表示与a_i存在控制依赖关系的基本块的集合。

根据约束条件和目标函数，建立问题二优化模型为：

$$\min n$$

$$
\begin{cases}
n = \max\limits_{i} f(a_i) \\
f(a_j) > f(a_i), a_j \in Dw_i \\
f(a_j) \geq f(a_i), a_j \in Dr_i \bigcup C_i \\
s.t. \begin{cases} TCAM_m + TCAM_{m+16} \leq 1, m = 0,1,...,15 \\ TCAM_m \leq 5, (m = 2k, k \geq 16) \\ TCAM_m \leq 1, \text{其它} \end{cases} \\
\begin{cases} HASH_m{}' \leq 2, m = 0,1,... \\ HASH_m{}' + HASH_{m+16}{}' \leq 3, m = 0,1,...,15 \end{cases} \\
ALU_m{}' \leq 56 \\
QUALIFY_m \leq 64
\end{cases}
\tag{26}
$$

5.2.3 问题二模型的求解

问题二模型流程如图所示，与问题一模型的求解主要区别在于不在一条执行流程上的基本块，可以共享 HASH 资源和 ALU 资源，因此资源约束条件进行了相应更改。

问题二下由各基本块之间的控制依赖和数据依赖所产生的排布流水线级数大小关系未改变，但由于共享资源的存在在进行基本块资源排布的优先度应该有所改变。为此我们设计了新的构建优先级队列的方法，具体体现在计算优先级队列权重。同时在进行资源排布，判断是否满足资源约束条件的方法做了相关更改以满足共享资源的存在。求解模型具体实现步骤如图 15 所示。

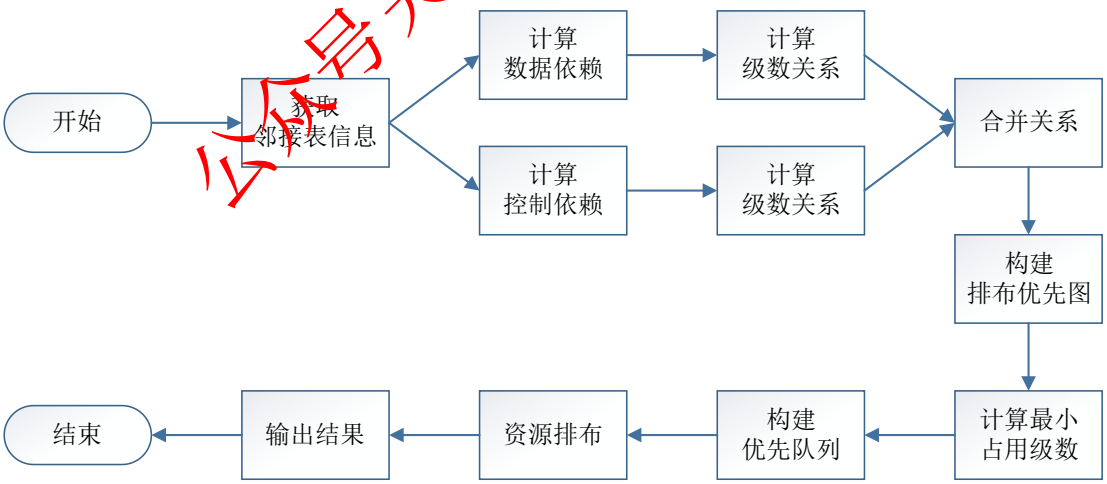


图 15 问题二模型求解整体流程图

（1）构建排布优先图

如图 16 所示，依据 attachment3.csv，获取各基本块在流程图中的邻接基本块信息，使用邻接表存储基本块流程图。依据基本块流程图遍历所有基本块分别计算出所有基本块之间的数据依赖关系和控制依赖关系，通过依赖关系可以得到两张所有基本块之间排布的流水线级数大小关系表。然后合并两张流水线级数大小关系表，即可计算出所有基本块之间的排布流水线关系。最后依据得到级数大小关系表，构建基本块之间的排布优先图[4]。

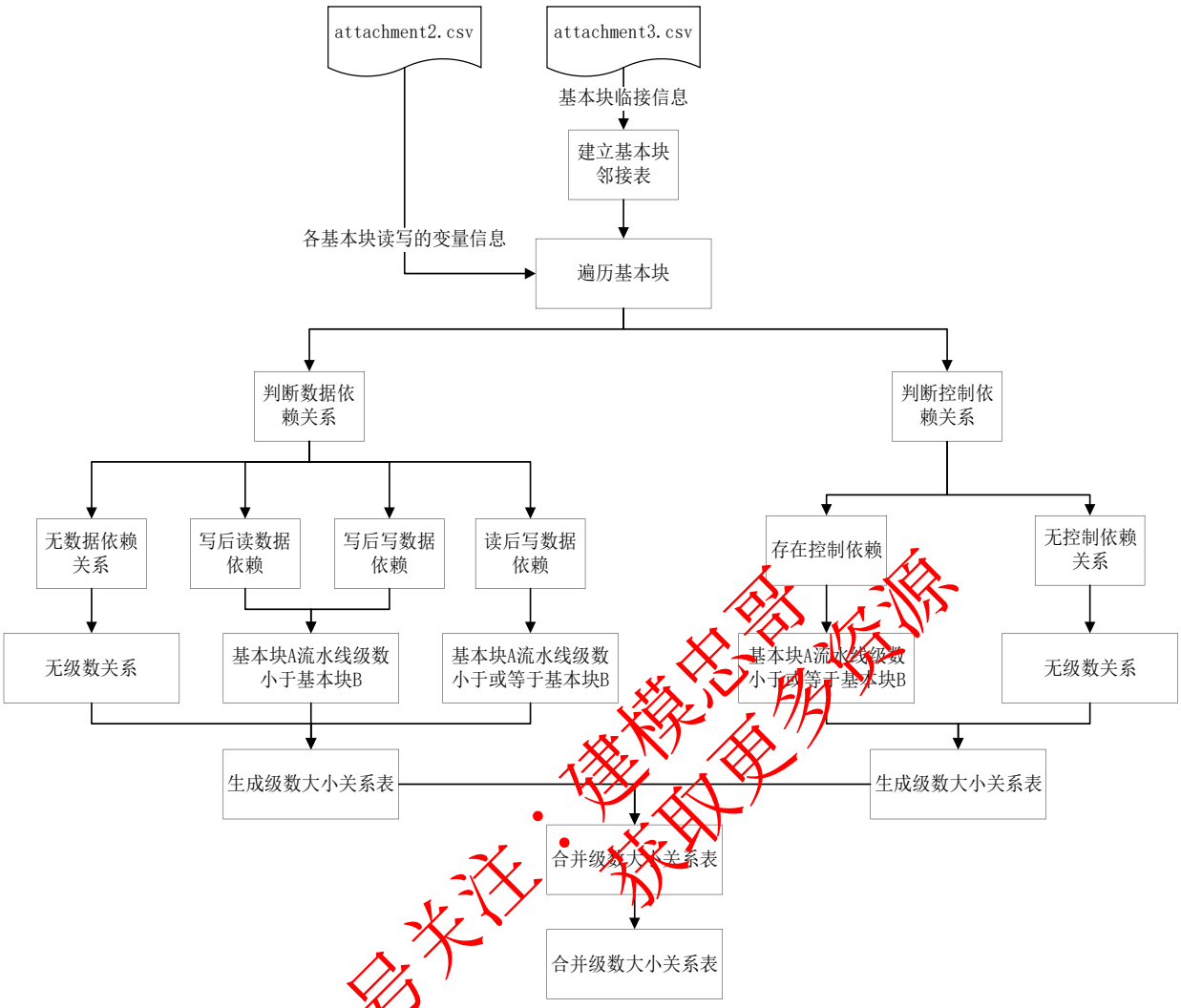


图 16 构建排布优先图流程图

（2）计算排布优先级队列所需权重

依据得到的排布优先图存储的邻接表，先划分出排布优先图中所有子图并查找出该子图的根节点。通过查找数据文件 attachment1.csv 中各基本块需要的资源信息，计算每个子图根节点在满足问题二中流水线每级资源约束条件下所需要最小流水线级数要求 X_i，计算每个子图在去除根节点后所有结点中最大的最小流水线级数要求 Y_i。最将根节点最小级数要求更新为 X_i+Y_i 。

（3）构建排布优先队列

构建优先级队列用于存储子图的根节点，排列权值为（2）中所求根节点所占最小的流水线级数，如图 17 所示。

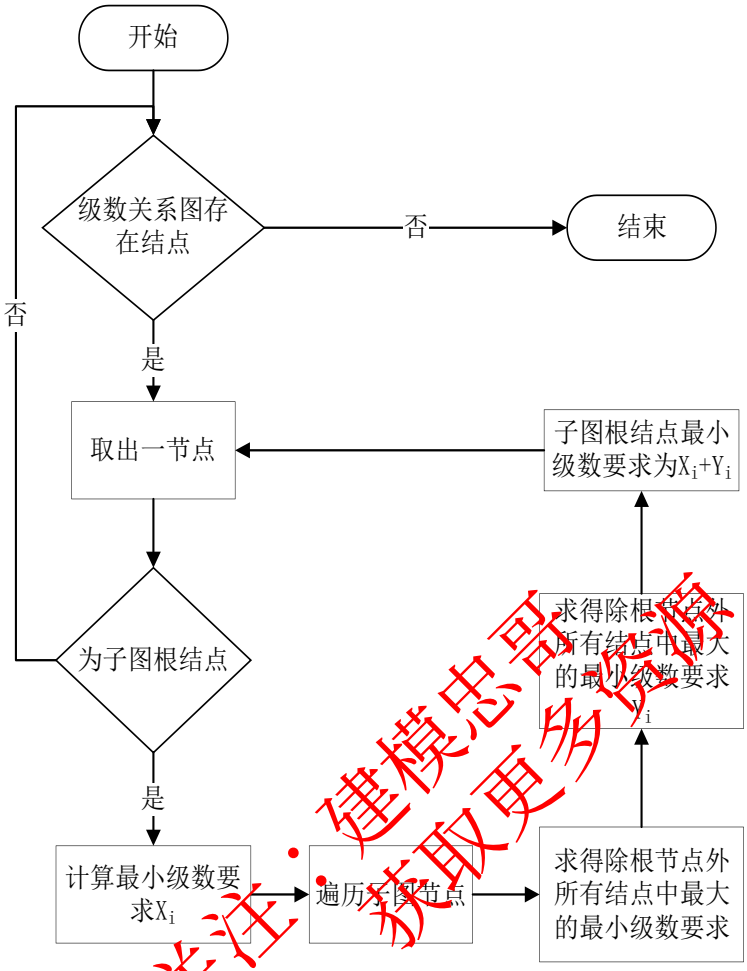


图 17 计算排布优先级队列所需权重流程图

（4）资源排布

根据构建的优先级选择基本块优先进行资源排布，具体流程同问题一模型求解，区别在于判断基本块是否满足填入当前流水线级别的判断。

Step1: 获取当前需要排布的基本块 *i*，获取当前流水线级内的已排布基本块列表 *l*；

Step2: 读取 *i* 的邻接关系，计算流水线级内的共享资源；

Step3: 判断共享资源是否满足约束条件，若是，执行下一步，否则结束；

Step4: 计算非共享资源，判断是否足够 *i* 排入，若是，则排入，否则结束；

算法步骤的具体流程如图 18 所示。

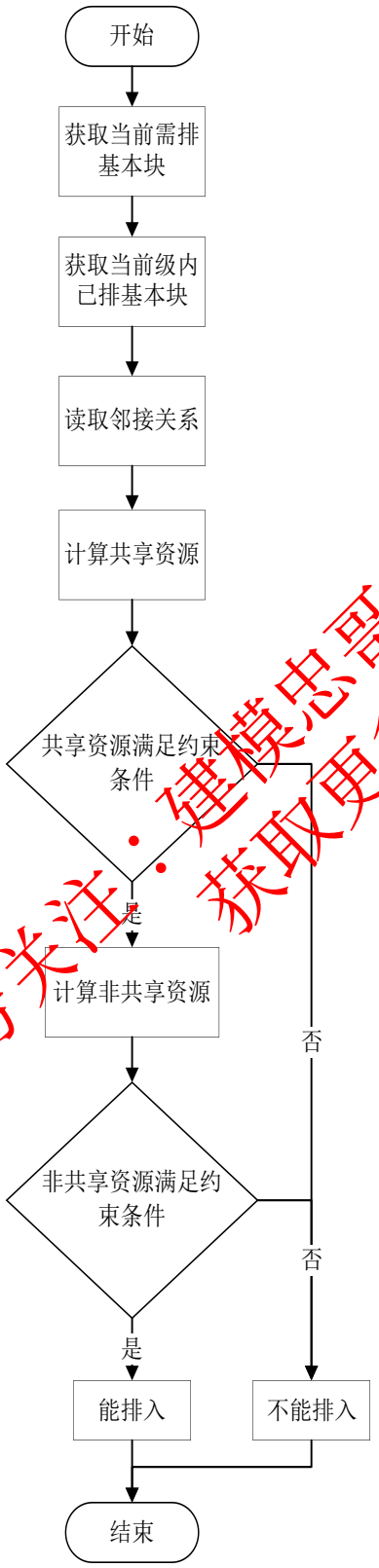


图 18 判断基本块是否能排入当前级数流程图

5.2.4 问题二结果的分析

（1）结果分析

在问题二所给约束条件下进行资源排布，输出的结果如表 5 所示：

表 5 部分流线资源排布

流水线级数编号	分配的基本块编号				
0	365	15	16	17	...
1	172	170	11	12	...
2	530	531	532	505	...
3	563	564	541	581	...
4	555	585	569	570	...
5	534	602	603	-	-
6	546	-	-	-	-
7	537	549	-	-	...
...	...	...	...	...	...
32	97	88	-	-	-

表中第一列为流水线级数编号，每行对应所在流水线级分配的基本块编号。同表 1 比较放发现，在问题二下排布的流水线级数要远小于问题一下排布的流水线级数，主要是因为共享资源的存在，相对下会减少部分资源限制所带来的影响。

从表 5 中流水线各级排布信息结合各基本块使用的资源信息，可以计算出每级流水线四种资源的利用率，如图 19。

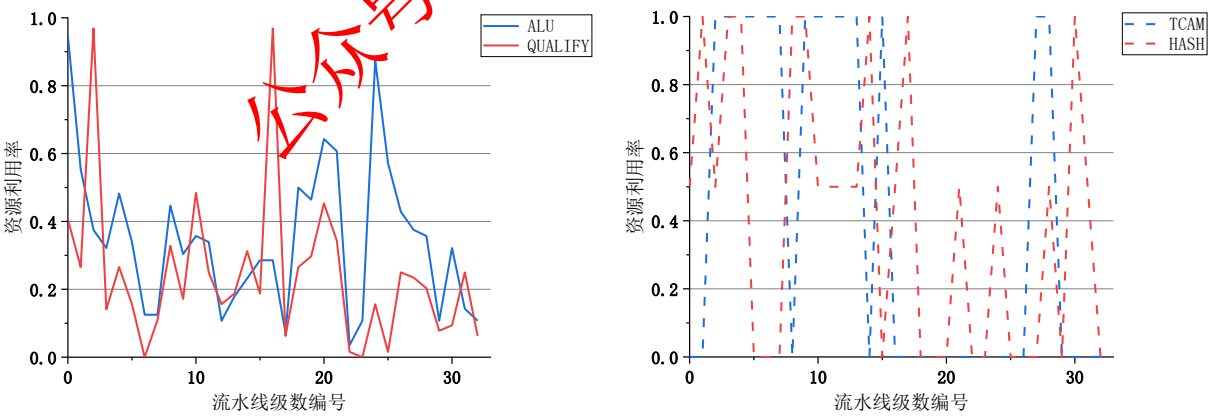


图 19 排布流水线每级 ALU、QUALIFY、TCAM、HASH 资源利用率

对比图 19 以及问题一的图 10，在可以共享 HASH 资源和 ALU 资源的问题二上，HASH 资源和 ALU 资源在资源利用率上大幅度降低，不可共享的 QUALIFY 资源和 TCAM 资源在资源利用率存在小幅度提升。与问题一相同，也存在四种资源利用率都较小的情况。

（2）模型灵敏度分析

针对问题二的资源约束条件，继续分析 TCAM、HASH、ALU 和 QUALIFY 四个资源对模型灵敏度的影响，具体分析如下。

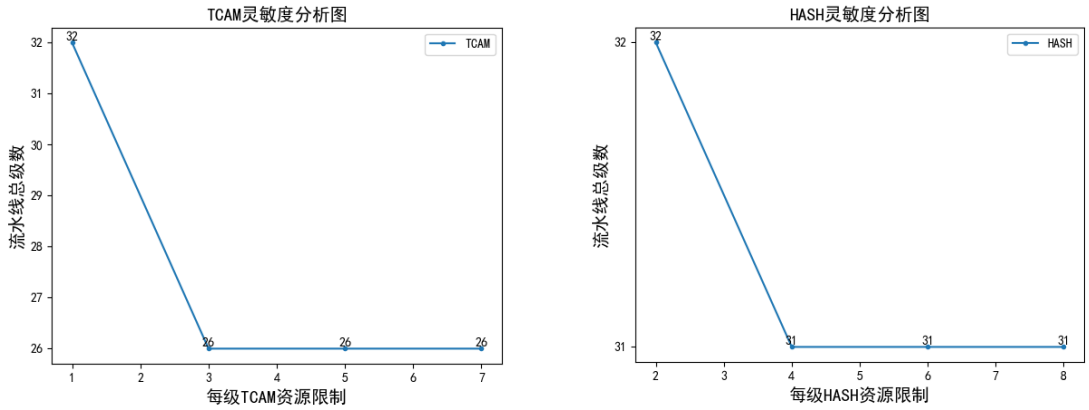


图 20 TCAM、HASH 资源灵敏度分析图

图 20（左）展示了每级 TCAM 资源限制和流水线总级数的关系，TCAM 从 1 到 3 增加时，流水线总级数陡降 6 级，后续增加 TCAM 资源对流水线总级数没有影响，总的来说，流水线总级数对每级 TCAM 资源限制敏感度高。

图 20（右）展示了每级 HASH 资源限制和流水线总级数的关系，因为前 32 级存在着折叠限制，为了控制变量所以对折叠两级的最大 HASH 也做了相应调整。当每级 HASH 资源限制从 2 开始递增时，流水线总级数降低 1 级，当 HASH 增大到 4 时，流水线总级数不变。当每级 HASH 资源限制较小时，流水线总级数相对更敏感。

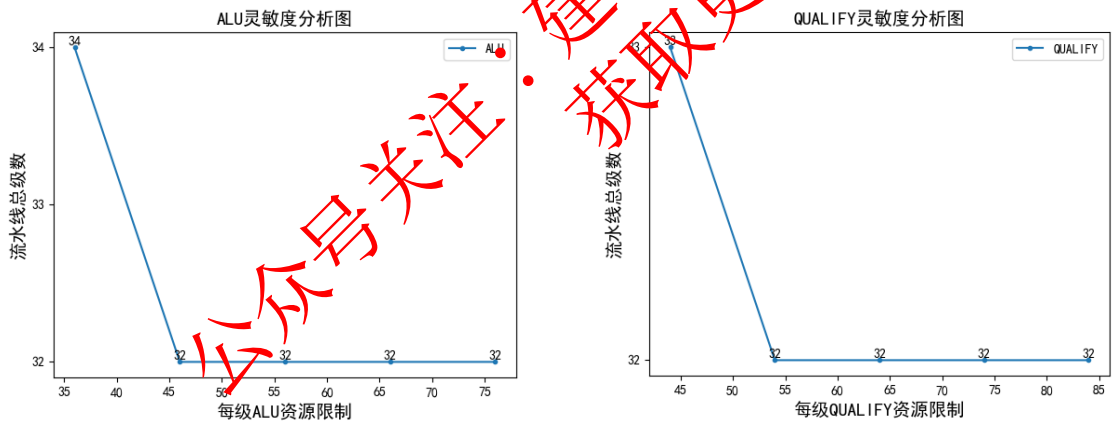


图 21 ALU、QUALIFY 资源灵敏度分析图

图 21（左）展示了每级 ALU 资源限制和流水线总级数的关系，当每级 ALU 资源限制从 36 开始递增时，流水线总级数降低 2 级，当增大到 46 后，流水线总级数不发生变化。可以发现，ALU 在[36,46]区间时，流水线总计数对 ALU 资源敏感度高。

图 21（右）展示了每级 QUALIFY 资源限制和流水线总级数的关系，当每级 QUALIFY 资源限制在[44,54]区间上时，流水线总级数仅下降一级。表明流水线总计数对 QUALIFY 资源敏感度低。

5.2.5 算法复杂度分析

A.存储复杂度分析

模型建立的时候存储有向图，其中图结点数量为 n 边数为 e，其存储复杂度为 O(n+e)。模型采用二维数组存储各基本块之间的各种关系，其存储复杂度为 $O(n^2)$。模型采用优先级队列存储基本块，其存储复杂度为 O(n)。综上所述，模型的空间复杂度为 $O(n^2)$。

B.运算复杂度分析

算法中的大部分方法基于深度优先遍历，复杂度为 O(n)，对 n 个结点进行遍历，时间

复杂度为 $O(n^2)$。进行排布时，每个结点排布一次，排布时计算共享资源占用的算法的复杂度为 $O(n)$，时间复杂度为 $O(n^2)$。模型总的时间复杂度为 $O(n^2)$。

六、 模型的评价与推广

6.1 模型的优点

（1）本文将资源排布问题转化为建立单目标优化模型，避免了复杂的理论计算过程，也简化了计算过程；

（2）本文根据优先级队列和启发式算法优先排布占用资源相对比较多的基本块，避免了低级流水线中资源占用过低导致的资源碎片问题；

（3）针对问题二充分考虑共享资源带来的影响，并针对性的设计了启发式算法满足真实生产工作需求。

6.2 模型的缺点

本文模型的时间复杂度存在改进空间，算法可以进一步优化。

6.3 模型的推广

在实际 PISA 架构芯片资源排布中基本块和变量信息会更多，基本块邻接关系会更为复杂，资源约束条件会更详细。所以在保证本文模型结果正确性的同时还需要对算法进行进一步地简化，降低计算时的复杂程度，提高效率，方便应对更复杂的情况。

七、参考文献

[1] Bosshart P, Daly D, Gibb G, et al. P4: Programming protocol-independent packet processors[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(3): 87-95.
[2] 姜启源, 谢金星, 叶俊. 数学模型 (第五版)[M]. 高等教育出版社 , 2018.
[3] 司守奎. 数学建模算法与应用 [M]. 北京：国防工业出版社， 2011.
[4] 阳西述. 用优先图辅助并发程序设计[J]. 计算机应用与软件, 2008(08):283-285.
[5] 周凯. 基于零件优先图和遗传烟花算法的退役电动汽车电池组拆解序列优化[D]. 辽宁: 大连理工大学, 2021.

附录

附录 1 问题一所用代码

```
'''图的邻接表方法表示'''
import csv
from collections import Counter
from queue import PriorityQueue
import random

# 结点存放基本块信息
class VNode(object):
    def __init__(self,data):
        self.data = data  # 编号
        self.W=None  # 写变量编号
        self.R=None  # 读变量编号
        self.firstarc = None  # 第一条边
        self.firstfather = None  # 第一条父结点


# 边信息
class ArcNode(object):
    def __init__(self,adjVex):
        self.adjVex = adjVex  # 边上结点编号
        self.w = None  #依赖关系权重
        self.nextarc = None  # 下一条边


class AGraph(object):
    # 实现输入图的顶点和边，能够得到邻接表
    def __init__(self, vers, edges):
        # 图的顶点
        self.vers = vers
        # 图的边
        self.edges = edges
        # vexLen 图的顶点个数
        self.vexLen = len(self.vers)
        # 存放顶点的列表
        self.listVex = [VNode for i in range(self.vexLen)]
        for i in range(self.vexLen):
            self.listVex[i] = VNode(self.vers[i])
        # 表示图的表
        for edge in self.edges:
            c1 = edge[0]
```

```
            c2 = edge[1]
            self.__addEdge(c1, c2)

    def addW(self, key, Wlist):
        V = self.__getPosition(key)
        self.listVex[V].W = Wlist

    def addR(self, key, Rlist):
        V = self.__getPosition(key)
        self.listVex[V].R=Rlist

    def __addEdge(self, c1, c2):
        # 找到 顶点 c1,c2 的下标值
        p1 = self.__getPosition(c1)
        p2 = self.__getPosition(c2)
        # 插入孩子结点
        edge2 = ArcNode(p2)
        # 尾插法
        if self.listVex[p1].firstarc is None:
            self.listVex[p1].firstarc = edge2
        else:
            self.__LinkLast(self.listVex[p1], edge2)
        # 插入父节点
        father_edge = ArcNode(p1)
        # 尾插法
        if self.listVex[p2].firstfather is None:
            self.listVex[p2].firstfather = father_edge
        else:
            self.__LinkLast_father(self.listVex[p2], father_edge)

    def __getPosition(self, key):
        for i in range(self.vexLen):
            if self.vers[i] == key:
                return i

    def __LinkLast(self, list, edge):
        p = list.firstarc
        while p.nextarc:
            p = p.nextarc
        p.nextarc = edge


    def __LinkLast_father(self, list, edge):
        p = list.firstfather
```

```
            while p.nextarc:
                p = p.nextarc
            p.nextarc = edge


        # 打印信息
        def print(self):
            for i in range(self.vexLen):
                print(self.listVex[i].data, end="->")
                edge = self.listVex[i].firstarc
                while edge:
                    print(self.listVex[edge.adjVex].data, end=" ")
                    edge = edge.nextarc
                print()

        # 遍历结点连接的所有结点
        # 计算 i,j 读写依赖，i 在 j 上游，当前遍历到 j
        def dfs_node_Trave(self, i, j):
            # print(i,j)
            global flag_v
            global data_dependence
            if flag_v[j] ==1:
                return 0
            else:
                flag_v[j] = 1
                #计算读写依赖
                if i != j:
                    if set(g.listVex[i].W) & set(g.listVex[j].R):
                        data_dependence[i][j] = 2   # 写后读数据依赖   基本块 A 流水线
级数   小于   基本块 B 排布的级数
                    elif set(g.listVex[i].W) & set(g.listVex[j].W):
                        data_dependence[i][j] = 2   # 写后写数据依赖   基本块 A 流水线
级数   小于   基本块 B 排布的级数
                    elif set(g.listVex[i].R) & set(g.listVex[j].W):
                        data_dependence[i][j] = 1   # 读后写数据依赖   基本块 A 流水线
级数   小于或等于    基本块 B 排布的级数
                    else:
                        data_dependence[i][j] = 0
                #遍历子结点
                p = g.listVex[j].firstarc
                while p:
                    k = p.adjVex
                    self.dfs_node_Trave(i, k)
                    p = p.nextarc
```

```
        return 0

# 计算结点直连结点数
def num_Vex(self, i):
        p = g.listVex[i].firstarc    # 第一条边
        Vexs=[]
        while p: #    当前结点 存在 多分支
            Vexs.append(p.adjVex)
            p=p.nextarc
        return Vexs

#    遍历结点连接的所有结点
#    计算控制依赖
def dfs_control_dependence(self, i, num_Vex ):
        global flag_v2
        global temp
        global son_vs_nums
        if flag_v2[i] == 1:
            return 0
        else:
            flag_v2[i] = 1

            for k in range(len(num_Vex)):
                if num_Vex[k] == i:
                    son_vs_nums[k] = son_vs_nums[k] + 1
            # print(j, end="")
            temp.append(i)

            p = g.listVex[i].firstarc
            while p:
                j = p.adjVex
                self.dfs_control_dependence(j, num_Vex)
                p = p.nextarc
        return 0


# 获取头结点
def find_root(self):
        list=[]
        for i in range(self.vexLen):
            if self.listVex[i].firstfather is None:
                list.append(self.listVex[i].data)
        return list
```

```
# 计算可达性
# i 到 j 是否存在路径
def dfs_link(self, i, j):
    global link
    global flag_v3
    if flag_v3[j] == 1:
        return 0
    else:
        flag_v3[j] = 1
        link[i][j] = 1
        #遍历子结点
        p = g.listVex[j].firstarc
        while p:
            k = p.adjVex
            self.dfs_link(i, k)
            p = p.nextarc
    return 0

# 计算深度
def dfs_deep(self, i):
    global flag_v5
    global deep
    if flag_v5[i] == 1: ###
        return 0
    else:
        #操作
        flag_v5[i] = 1
        # 遍历子结点
        p = g.listVex[i].firstarc
        while p:
            k = p.adjVex
            self.dfs_deep(k)
            deep[i]=max(deep[i],deep[k]+1)
            p = p.nextarc
    return 0


# 第一题计算预期资源
def dfs_pre_1(self, i, j):
    global flag_v4
    if flag_v4[j] == 1:
        return 0
    else:
        flag_v4[j] = 1
```

```
            # 计算下游资源和
            pred_TCAM[i] = pred_TCAM[i] + TCAM[j]
            pred_HASH[i] = pred_HASH[i] + HASH[j]
            pred_ALU[i] = pred_ALU[i] + ALU[j]
            pred_QUA[i] = pred_QUA[i] + QUA[j]
            # 遍历子结点
            p = g.listVex[j].firstarc
            while p:
                k = p.adjVex
                self.dfs_pre_1(i, k)
                p = p.nextarc
        return 0


    # 第二题计算预计资源
    def dfs_pred_2(self, i):
        global flag_v5
        if flag_v5[i] == 1:
            return 0
        else:
            # 遍历子结点
            p = g.listVex[i].firstarc
            if p is None:
                pred_HASH[i] = HASH[i]
                pred_ALU[i] = ALU[i]
            else:
                while p:
                    k = p.adjVex
                    self.dfs_pred_2(k)
                    pred_HASH[i] = max(pred_HASH[i], pred_HASH[k]+HASH[i])
                    pred_ALU[i] = max(pred_ALU[i], pred_ALU[k]+ALU[i])
                    p = p.nextarc
        flag_v5[i] = 1
        return 0


# 递归求解共享资源占用
def OC(t_list):
    global total_HASH
    global total_ALU

    if len(t_list)==0:
        return 0,0
    else:
```

```
            temp = t_list.pop()
            #只有一个，返回
            if len(t_list)==0:
                return HASH[temp],ALU[temp]
            else:
                list=[]
                #记录与 temp 不共享块
                for i in range(len(t_list)):
                    if link[t_list[i]][temp] == 1 or link[temp][t_list[i]] == 1:
                        list.append(t_list[i])
                #不共享块的占用资源
                H, A=OC(list)
                return H+HASH[temp], A+ALU[temp]


if __name__ == '__main__':
    #  读取文件，获取节点和边
    f3 = open("attachment3.csv", "r")
    nodelist = []
    edgelist = []

    #  读取表三建立邻接表
    while True:
        line3 = f3.readline()
        if line3:
            line3 = line3.strip()    #  去掉尾随空格

            if (line3.__contains__(',')):    #  存在边
                node = line3.split(',')[0]    #  获取图节点
                nodelist.append(node)    #  存储图节点
                out_nodes = line3.split(',')[1:]    #  获取图边，该节点是起点

                # print(node, out_nodes)

                for ins in range(len(out_nodes)):    #  遍历边
                    if out_nodes[ins].strip() != '':
                        out_edge = (node, out_nodes[ins])
                        if out_edge not in edgelist:
                            edgelist.append(out_edge)
            else:
                nodelist.append(line3)    #  存储图节点
        else:
            break
    f3.close()
```

```
g = AGraph(nodelist, edgelist)    #    邻接表

f2 = open("attachment2.csv", "r")
#  读取表二获取读取信息，得出基本块的数据依赖关系
while True:
    line2 = f2.readline()
    if line2:
        line2 = line2.strip()    #  去掉尾随空格
        if (line2.__contains__(',')):    #  存在变量读取
            node = line2.split(',')[0]    #  结点编号
            if line2.split(',')[1] == 'W':
                g.addW(node, line2.split(',')[2:])
            else:
                g.addR(node, line2.split(',')[2:])
    else:
        break
f2.close()

# data_dependence  存储  数据依赖  关系
print("计算数据依赖")
data_dependence = [[0 for i in range(g.vexLen)] for j in range(g.vexLen)]
#  根据邻接表  和  attachment2.csv  得出  数据依赖  关系
for i in range(g.vexLen):
    flag_v = [0 for i in range(g.vexLen)]
    g.dfs_node_Trave(i,i)

#    控制依赖  关系
print("计算控制依赖")
control_dependence = [[0 for i in range(g.vexLen)] for j in range(g.vexLen)]
for i in range(g.vexLen):
    # flag_v2 = [0 for i in range(g.vexLen)]
    # print("结点",end='')
    # print(i,end='')
    # print("直连数和结点", len(g.num_Vex(i)), g.num_Vex(i))

    if len(g.num_Vex(i))>1:    #  存在多分支
        # print("分支数", len(g.num_Vex(i)))
        temp = []
        # flag_v2 = [0 for i in range(g.vexLen)]
        son_vs_nums = [0 for i in range(g.vexLen)]


        for j in range(len(g.num_Vex(i))):
```

```
                flag_v2 = [0 for i in range(g.vexLen)]
                # print("分支结点", g.num_Vex(i)[j])
                g.dfs_control_dependence(g.num_Vex(i)[j], g.num_Vex(i) )

            for j in range(len(g.num_Vex(i))):   #  结点到一代孩子  分支数
                # print("结点到一代孩子  分支数", j, son_vs_nums[j])
                if son_vs_nums[j] < len(g.num_Vex(i)):
                    # print(i, g.num_Vex(i)[j], "满足条件")
                    control_dependence[i][g.num_Vex(i)[j]] = 1

            # print("temp", temp, type(temp))
            # print("g.num_Vex(i)", g.num_Vex(i), type(g.num_Vex(i)))

            # temp  去掉  第一代  孩子
            temp_no_vex = []
            for m in temp:
                if m not in g.num_Vex(i):
                    temp_no_vex.append(m)

            # print("temp_no_vex", temp_no_vex)
            result = Counter(temp_no_vex)
            for e, f in result.items():   #    e 元素   f 次数
                # print("元素  次数", e,f)
                if f < len(g.num_Vex(i)):
                    control_dependence[i][e]=1
                    # print(i, e, "满足条件")

# 合并  数据依赖  控制依赖
dependence = [[0 for i in range(g.vexLen)] for j in range(g.vexLen)]
for i in range(g.vexLen):
    for j in range(g.vexLen):
        if data_dependence[i][j]>=control_dependence[i][j]:
            dependence[i][j]=data_dependence[i][j]
        else:
            dependence[i][j]=control_dependence[i][j]


# 构建优先图
print("重构有向图")
nodelist_1 = []
edgelist_1 = []
nodelist_2 = []
edgelist_2 = []
nodelist_total = []
```

```
edgelist_total = []
#  重组邻接表
for i in range(g.vexLen):
        nodelist_1.append(i)    #  存储图节点
        nodelist_2.append(i)    #  存储图节点
        nodelist_total.append(i)    #  存储图节点
        #  根据下游结点的依赖关系重建有向图
        for j in range(g.vexLen):
                out_edge = (i, j)
                if dependence[i][j] == 1:         #  有软优先
                        if out_edge not in edgelist_1:
                                edgelist_1.append(out_edge)
                                edgelist_total.append(out_edge)
                elif dependence[i][j] == 2:      #  有硬优先
                        if out_edge not in edgelist_2:
                                edgelist_2.append(out_edge)
                                edgelist_total.append(out_edge)
g_1 = AGraph(nodelist_1, edgelist_1)    #  邻接表
g_2 = AGraph(nodelist_2, edgelist_2)    #  邻接表
g_t = AGraph(nodelist_total, edgelist_total)    #  邻接表
# g_1.print()
# g_2.print()
# print(g_1.find_root())
print("根结点为", g_t.find_root())

#  计算可达性
link = [[0 for i in range(g.vexLen)] for j in range(g.vexLen)]
for i in range(g.vexLen):
        flag_v3 = [0 for j in range(g.vexLen)]
        g.dfs_link(i, i)


########################
#  第一问
#  读取数据
TCAM = [0 for i in range(g.vexLen)]
HASH = [0 for i in range(g.vexLen)]
ALU = [0 for i in range(g.vexLen)]
QUA = [0 for i in range(g.vexLen)]
f1 = open("attachment1.csv", "r")
#  读取表 1 资源信息
_ = f1.readline()
while True:
        line = f1.readline()
```

```
            if line:
                line = line.strip()    #  去掉尾随空格
                if (line.__contains__(',')):    #  存在变量读取
                    node = int(line.split(',')[0]) #  结点编号
                    TCAM[node] = int(line.split(',')[1])
                    HASH[node] = int(line.split(',')[2])
                    ALU[node] = int(line.split(',')[3])
                    QUA[node] = int(line.split(',')[4])
            else:
                break
        f1.close()
        # print(TCAM[0],HASH[0],ALU[0],QUA[0])
        # print(TCAM[606], HASH[606], ALU[606], QUA[606])

        #  计算下游预期资源
        print("计算预期资源")
        pred_TCAM = [0 for i in range(g.vexLen)]
        pred_HASH = [0 for i in range(g.vexLen)]
        pred_ALU = [0 for i in range(g.vexLen)]
        pred_QUA = [0 for i in range(g.vexLen)]
        pred_n = [0 for i in range(g.vexLen)]
        for i in range(g.vexLen):
            flag_v4 = [0 for j in range(g.vexLen)]
            g_t.dfs_pre_1(i,i)
        #  根据数据依赖计算
        deep=[1 for i in range(g.vexLen)]
        for i in range(g.vexLen):
            flag_v5 = [0 for j in range(g.vexLen)]
            g_2.dfs_deep(i)
        #  计算下游预期占用级数
        for i in range(g.vexLen):
            pred_n[i]=max(pred_TCAM[i], pred_HASH[i]//2,
pred_ALU[i]//56,pred_QUA[i]//64)
            pred_n[i] = max(pred_n[i], deep[i])
        # for i in range(g.vexLen):
        #         print(pred_n[i])

        #  设计优先级
        for i in range(g.vexLen):
            pred_n[i] = -pred_n[i]    #-pred_n[i]

        #  排布
        print("排布")
        Done = [0 for i in range(g.vexLen)]    #基本块是否已排布
```

```
next = [0 for i in range(g.vexLen)]   #基本块是否必须是下一级
his_T = [0 for i in range(16)]   #前 16 个 TCAM
his_H = [0 for i in range(16)]   #前 16 个 HASH
jishu = 0   #当前级数
total = 0
result = []   #最终结果
p = PriorityQueue()   #维护优先队列
p.put((pred_n[365], 365))   #入根结点 365
res_TCAM = 1   #1
res_HASH = 2   #2
res_ALU = 56   #56
res_QUA = 64   #64   #资源初始化
_TCAM = res_TCAM
_HASH = res_HASH
_ALU = res_ALU
_QUA = res_QUA   #保存最大资源，计算利用率
list = []   #当前级数的基本块
q_save=set()   #暂存
#开始排布
while not p.empty():
    #取一个结点
    _item = p.get()
    temp_node = _item[1]
    # print('get',temp_node)
    #判断能否填入,是否必须在下一块
    if res_TCAM>=TCAM[temp_node] and res_HASH>=HASH[temp_node]\
            and res_ALU>=ALU[temp_node] and res_QUA>=QUA[temp_node]\
            and next[temp_node]!=1:
        #能填入
        #记录
        list.append(temp_node)
        # print('append',temp_node)
        total=total+1
        Done[temp_node] = 1
        #资源减少
        res_TCAM=res_TCAM-TCAM[temp_node]
        res_HASH=res_HASH-HASH[temp_node]
        res_ALU=res_ALU-ALU[temp_node]
        res_QUA=res_QUA-QUA[temp_node]
        #子结点入队
        i= g_t.listVex[temp_node].firstarc   #第一条子边 i
        while i:
            k = i.adjVex   #子结点编号 k
            kf = g_t.listVex[k].firstfather   #父边 kf
```

```
                flag = 1
                while kf:     #遍历父结点
                    kf_d = kf.adjVex    #父结点编号 kf_d
                    if Done[kf_d]==0:    #父结点编号 kf_d 未排布
                        flag=0            #不是根结点
                    kf = kf.nextarc
                if flag == 1:    #k 是根结点
                    p.put((pred_n[k],k))      #k 入队
                    # print('put',k)
                    if dependence[temp_node][k]==2:
                        #数据依赖严格小于
                        next[k]=1    #排布到下一块
                i = i.nextarc    #遍历子结点
            #队列取空了
            if p.empty():
                #  重新入队
                for i in range(len(q_save)):
                    _t = q_save.pop()
                    p.put((pred_n[_t], _t))
                    # print('put', _t)
        else:
            #不能填入
            #暂存在集合中
            q_save.add(temp_node)
            # print('save',temp_node)
            if p.empty():
                #队列为空
                #  换新流水线
                # copylist=list
                # result.append(list)    #保存当前级
                print(list)
                #print(jishu, 1-res_TCAM/_TCAM, 1-res_HASH/_HASH, 1-res_ALU/_ALU,
1-res_QUA/_QUA)
                # print('排完一级')
                list.clear()     #清空当前级列表
                #  输出
                #更新资源
                res_ALU = 56 #56
                res_QUA = 64 #64
                #更新 TCAM
                if jishu>=16 and jishu <=31:
                    res_TCAM = 1- his_T[jishu-16]    #1- his_T[jishu-16]
                elif jishu >= 32 and jishu%2==0:
                    res_TCAM = 5
```

```
                else:
                    res_TCAM = 1    #1
                #更新 HASH
                if jishu>=16 and jishu <=31:
                    res_HASH = min(2, (3 - his_H[jishu - 16])) #min(2, (3 - his_H[jishu -
16])
                else:
                    res_HASH = 2 #2
                #保存最大资源，计算利用率
                _TCAM=res_TCAM
                _HASH=res_HASH
                _ALU=res_ALU
                _QUA=res_QUA
                #重新入队
                for i in range(len(q_save)):
                    _t=q_save.pop()
                    p.put((pred_n[_t],_t))
                    # print('put', _t)
                #级数+1
                jishu = jishu + 1
                #重置 next
                for i in range(len(next)):
                    next[i]=0
            #不为空，继续取下一个

    #排布完输出
    print(list)
    #print(jishu, 1-res_TCAM/_TCAM, 1-res_HASH/_HASH, 1-res_ALU/_ALU, 1-
res_QUA/_QUA)
    print(jishu)
    #print(total)
```

附录 2 问题二所用代码

```
    ########################
    # 第二问
    # 计算下游预期资源
    print("计算第二问预期资源")
    pred_TCAM = [0 for i in range(g.vexLen)]
    pred_HASH = [0 for i in range(g.vexLen)]
    pred_ALU = [0 for i in range(g.vexLen)]
    pred_QUA = [0 for i in range(g.vexLen)]
    pred_n = [0 for i in range(g.vexLen)]
    for i in range(g.vexLen):
```

```
            flag_v4 = [0 for j in range(g.vexLen)]
            g_t.dfs_pre_1(i,i)
        # 修改 HASH 和 ALU
        # 重置
        for i in range(g.vexLen):
            pred_HASH[i]=0
            pred_ALU[i]=0
        # 计算
        for i in range(g.vexLen):
            flag_v5 = [0 for j in range(g.vexLen)]
            g_t.dfs_pred_2(i)

        # 根据数据依赖计算
        deep = [1 for i in range(g.vexLen)]
        for i in range(g.vexLen):
            flag_v5 = [0 for j in range(g.vexLen)]
            g_2.dfs_deep(i)
        # 计算下游预期占用级数
        for i in range(g.vexLen):
            pred_n[i] = max(pred_TCAM[i], pred_HASH[i] // 2, pred_ALU[i] // 56, pred_QUA[i]
// 64)
            pred_n[i] = max(pred_n[i], deep[i])
        # for i in range(g.vexLen):
        #         print(pred_n[i])

        for i in range(g.vexLen):
            # print(pred_n[i])
            pred_n[i] = -(pred_n[i]*10+TCAM[i])

        # 排布
        print("排布")
        Done = [0 for i in range(g.vexLen)]    # 基本块是否已排布
        next = [0 for i in range(g.vexLen)]    # 基本块是否必须是下一级
        his_T = [0 for i in range(16)]    # 前 16 个 TCAM
        his_H = [0 for i in range(16)]    # 前 16 个 HASH
        jishu = 0    # 当前级数
        total = 0
        result = []    # 最终结果
        p = PriorityQueue()    # 维护优先队列
        p.put((pred_n[365], 365))    # 入根结点 365
        res_TCAM = 1    #1
        res_HASH = 2 #2
        res_ALU = 56    #56
        res_QUA = 64 #64    # 资源初始化
```

```
_TCAM = res_TCAM
_HASH = res_HASH
_ALU = res_ALU
_QUA = res_QUA   # 保存最大资源，计算利用率
total_HASH = res_HASH
total_ALU = res_ALU   # 记录总资源
list = []   # 当前级数的基本块
q_save = set()   # 暂存
# 开始排布
while not p.empty():
    # 取一个结点
    _item = p.get()
    temp_node = _item[1]
    # print('get',temp_node)
    # 判断能否填入,是否必须在下一块
    # 计算实际可占用资源
    real_TCAM = res_TCAM
    real_HASH = total_HASH
    real_ALU = total_ALU
    real_QUA = res_QUA
    if len(list)>0:
        #有排布时
        not_list=[]
        for li in list:
        #找不可共用的基本块
            if link[li][temp_node]==1 or link[temp_node][li]==1:
                not_list.append(li)
                # # 计算可用资源
                # real_HASH=real_HASH-HASH[li]   #减去不可共用的 HASH
                # real_ALU=real_ALU-ALU[li]   #减去不可共用的 HASH
        #计算可用资源
        HASH_t,ALU_t=OC(not_list)
        real_HASH=total_HASH-HASH_t
        real_ALU=total_ALU-ALU_t
    # 没有排布时，可用资源等于总资源

    # 判断能否填入,是否必须在下一块
    if real_TCAM >= TCAM[temp_node] and real_HASH >= HASH[temp_node] \
            and real_ALU >= ALU[temp_node] and real_QUA >= QUA[temp_node] \
            and next[temp_node] != 1:
        # 能填入
        # 记录
        list.append(temp_node)
        # print('append',temp_node)
```

```
                total = total + 1
                Done[temp_node] = 1
                #  计算剩余资源
                res_TCAM = res_TCAM - TCAM[temp_node]
                res_HASH = min(res_HASH, real_HASH - HASH[temp_node])
                res_ALU = min(res_ALU, real_ALU - ALU[temp_node])
                res_QUA = res_QUA - QUA[temp_node]
                #  子结点入队
                i = g_t.listVex[temp_node].firstarc    #  第一条子边 i
                while i:
                    k = i.adjVex    #  子结点编号 k
                    kf = g_t.listVex[k].firstfather    #  父边 kf
                    flag = 1
                    while kf:    #  遍历父结点
                        kf_d = kf.adjVex    #  父结点编号 kf_d
                        if Done[kf_d] == 0:    #  父结点编号 kf_d 未排布
                            flag = 0    #  不是根结点
                        kf = kf.nextarc
                    if flag == 1:    # k 是根结点
                        p.put((pred_n[k], k))    # k 入队
                        if dependence[temp_node][k] == 2:
                            #  数据依赖严格小于
                            next[k] = 1    #  排布到下一块
                    i = i.nextarc    #  遍历子结点
                #  队列取空了
                if p.empty():
                    #  重新入队
                    for i in range(len(q_save)):
                        _t = q_save.pop()
                        p.put((pred_n[_t], _t))
            else:
                #  不能填入
                #  暂存在集合中
                q_save.add(temp_node)
                # print('save',temp_node)
                if p.empty():
                    #  队列为空
                    #  换新流水线
                    # result.append(list)    #保存当前级
                    print(list)    #输出结果
                    #  计算资源利用率
                    #print(jishu, 1-res_TCAM/_TCAM, 1-res_HASH/_HASH, 1-res_ALU/_ALU,
1-res_QUA/_QUA)
                    # print('排完一级')
```

```
                list.clear()  # 清空当前级列表
                # 更新资源
                res_ALU = 56 #56
                res_QUA = 64   #64
                # 更新 TCAM
                if jishu >= 16 and jishu <= 31:
                    res_TCAM = 1 - his_T[jishu - 16]   #1 - his_T[jishu - 16]
                elif jishu >= 32 and jishu % 2 == 0:
                    res_TCAM = 5
                else:
                    res_TCAM = 1   #1
                # 更新 HASH
                if jishu >= 16 and jishu <= 31:
                    res_HASH = min(2, (3 - his_H[jishu - 16]))   #min(2, (3 - his_H[jishu
- 16]))
                else:
                    res_HASH = 2 #2
                # 保存最大资源，计算利用率
                _TCAM = res_TCAM
                _HASH = res_HASH
                _ALU = res_ALU
                _QUA = res_QUA
                # 记录总资源
                total_TCAM = res_TCAM
                total_HASH = res_HASH
                total_ALU = res_ALU
                total_QUA = res_QUA
                # 重新入队
                for i in range(len(q_save)):
                    _t = q_save.pop()
                    p.put((pred_n[_t], _t))
                    # print('put', _t)
                # 级数+1
                jishu = jishu + 1
                # 重置 next
                for i in range(len(next)):
                    next[i] = 0
            # 不为空，继续取下一个

    # 排布完输出
    print(list)
    #print(jishu, 1-res_TCAM/_TCAM, 1-res_HASH/_HASH, 1-res_ALU/_ALU, 1-
res_QUA/_QUA)
    print(jishu)
```

```
#print(total)
```