分配问题及其数学模型*

苏 明 薛宏熙 洪先龙

(清华大学计算机科学与技术系,北京 100084)

摘要 随着集成电路的发展、集成度的提高,特别是80年代中期专用集成电路的出现,数字系统的高层次综合已成为数字系统设计的一种实用的、有效的设计方法. 本文讨论高层次综合中的分配问题. 首先将分配问题化为整数规划问题,然后提出了一个启发式的结群分配算法. 从实验结果看,该算法以较少的CPU时间得到了与已发表的几种算法相似甚至更好的结果.

关键词 分配,结群,数据通路,整数规划.

高层次综合是指从算法级的行为描述到实现它的寄存器传输级结构描述的综合. 该结构包括一个数据通路,即一个由寄存器、功能单元、多路器和总线构成的互连网络;以及一个控制数据通路中数据传输的控制器. 通常实现给定行为的硬件结构有许多种,综合的一个重要任务就是找出一个满足约束条件和目标集合的、花费最少的硬件结构.

高层次综合的第一步通常是将语言描述翻译到一种中间表示格式并进行初始优化. 接下来的两步,即从行为到结构转换的核心部分是调度(Scheduling)和分配(Allocation),它们是紧密相关的两步[1],分别用于安排操作的执行顺序和实现操作的部件. 此外,要设计数据通路的抽象结构,还必须决定数据通路中各部件是如何实现的,这一步也称作模块确定(Module Binding). 一旦调度和分配完成,数据通路设计好之后,还需要综合一个按调度要求驱动数据通路的控制器.

在上述各任务中,为了找到满足约束条件的最优解,必须检查庞大的设计空间. 需要搜索的设计空间又是多维的、非连续空间,很难找到一个规范的搜索集来系统地搜索整个空间. 而且,设计空间的形状通常是由问题说明的,也无法保证搜索整个空间. 即使对于高层次综合中的任何一个任务(例如,约束条件下的分配问题)都是NP难题[2].

1 分配问题

分配又称作数据通路分配(Data Path Allocation),它的任务是将操作和变量(或值)赋给相应的硬件进行运算和存放. 这些硬件通常是功能块级的模块,如功能单元、寄存器和多

* 本文1993-10-22收到,1994-03-21定稿

作者苏明,1965年生,讲师,主要研究领域为数字系统的综合、模拟、验证及设计系统的支撑环境. 薛宏熙,1938年生,副教授,主要研究领域为数字系统的计算机辅助设计,包括综合、模拟、测试等. 洪先龙,1940年生,教授,主要研究领域为VLSI CAD,包括布图算法、软件技术及系统.

本文通讯联系人:苏明,北京 100084,清华大学计算机科学与技术系

路器等.分配问题包括3个部分:功能单元的分配、寄存器的分配和互连线路的分配.一般来讲,在控制数据流图(一种中间表示格式)[3]中,操作赋给功能单元、变量(或值)赋给寄存器、多路器和互连线路则用来建立各单元之间的数据传输通路.分配的目标在于:建立一个功能块级模块组成的数据通路,使得所需硬件资源的费用最少.

分配由它所要实现的调度结果约束.在分配时,调度结果中的所有操作,都必须赋给相应的功能单元进行运算,调度在同一控制步(对应于时钟周期)[4]中的两个操作(非互斥操作),不能共享同一个功能单元.在调度结果中,所有跨越控制步边界的变量(或值),都应该存放到寄存器中.不跨越控制步边界的变量(或值),则无需存放到寄存器中,而直接连接到其目标模块上.同样,在同一控制步中并存的两个变量(非互斥操作产生的),也不能共享同一个寄存器.

2　分配问题的数学模型

在进行分配前,首先假设数据通路中各单元之间是点到点的连接方式.当分配完成后,再考虑将相关连线依据某些规则(或称经验)合并成总线.

2.1　术语与定义

调度完成后,调度的结果将作为分配的输入.所以,在进行分配时,已经知道操作所调度到的控制步、操作的数据相关性、操作的各输出是否需要存放到寄存器中和各操作的等价集(由调度在同一控制步中共享同一功能单元的互斥操作组成的集合)[3].根据这些输入条件,就可建立分配问题的数学模型.在建立数学模型的过程中,将用到下面几个术语与定义.

定义1. 等价集作为一个特殊的操作,称为等价集操作.该操作的操作类型集是等价集中各元素操作类型的并集;该操作的输入个数是等价集中各元素输入个数的最大值;该操作的输出个数是等价集中各元素输出个数的最大值;该操作的数据相关性是等价集中各元素数据相关性的并集.

定义2. 变量从产生到它最后一次被引用之间的控制步,称为变量的生存期.

定义3. 等价集操作的任一输入变量或输出变量,称为等价集变量.等价集变量的生存期是对应于等价集变量的各变量生存期的并.

定义4. 由操作和变量组成的集合称作运算集合,用A表示.

定义5. 由功能单元和存储单元(寄存器)组成的集合称作模块集合,用B表示.

在模块集合中,两个相同类型的模块是有区别的(即同一类型的模块可以有多个用于分配且足够分配时使用).若模块集合中的模块b可以用来实现运算a,则表示为:

$$a \rightarrow b.$$

根据调度结果,可以知道各运算之间的数据相关关系,从而可以建立起运算之间的一种连接关系L:

$L=\{(a1,i1,a2,i2)|$ 运算$a1$的第$i1$个输出被运算$a2$的第$i2$个输入引用$\}$

在建立数学模型的过程中,还要用到下面一些变量(或常量).

$x_{a,b,j}$:运算a是否在控制步j送往模块b中执行.

$$x_{a,b,j}=\begin{cases}1, & \text{运算 } a \text{ 在控制步 } j \text{ 送往模块中执行}\\ 0, & \text{否则}\end{cases}$$

$W_{b1,i1,b2,i2}$：$b1$ 的第 $i1$ 个输出与 $b2$ 的第 $i2$ 个输入之间是否有连线.

$$W_{b1,i1,b2,i2}=\begin{cases}1, & \text{有连线}\\ 0, & \text{否则}\end{cases}$$

$y_{a,b,j}$：在控制步 j 中，运算 a 是否在模块 b 中执行. 对于变量则表示运算 a 是否存放在模块 b 中.

$$y_{a,b,j}=\begin{cases}1, & \text{运算 } a \text{ 在模块 } b \text{ 中执行}\\ 0, & \text{否则}\end{cases}$$

T_{step}：所需的控制步数目(由调度结果得到). in_b：模块 b 的输入端口数. out_b：模块 b 的输出端口数. c_b：模块 b 的费用. E_a：运算 a 的执行时间(或生存期). D_b：模块 b 的延迟时间(寄存器的延迟时间为 1). L_b：流水线模块 b 的等待时间(*Latency*). 为了便于表示，对于非流水线模块，令 $L_b=D_b$. c_w：一根连线的费用，假设各连线具有相同的费用. c_m：折合到一个输入端所对应的多路器费用.

$$c_m=\begin{cases}0, & \text{输入端个数小于 2}\\ \text{大于 1 的常量}, & \text{否则}\end{cases}$$

2.2 分配问题的数学模型

对于分配问题，其目标是找到费用最省的硬件实现. 因此，分配的目标就是求模块费用、连线费用和多路器费用之和的最小值，即求下式的最小化问题：

$$\sum_{b\in B}c_b+c_w\times\Big(\sum_{b1,b2\in B}\sum_{i1=1}^{out_{b1}}\sum_{i2=1}^{in_{b2}}W_{b1,i1,b2,i2}\Big)+c_m\times\Big(\sum_{b1,b2\in B}\sum_{i1=1}^{out_{b1}}\sum_{i2=1}^{in_{b2}}W_{b1,i1,b2,i2}\Big) \tag{1}$$

对于各运算，必须由一个模块来执行. 即：

$$\sum_{j=1}^{T_{step}}\sum_{b\in B}x_{a,b,j}=1,\quad \text{对于 } a\in A \tag{2}$$

运算 a 的执行时间(或生存时间)应该等于它的执行周期(或生存期). 即：

$$\sum_{j=1}^{T_{step}}\cdot\sum_{b\in B\wedge a\to b}y_{a,b,j}=E_a,\quad \text{对于 } a\in A \tag{3}$$

而

$$y_{a,b,j}=\sum_{i=j-E_a+1}^{j}x_{a,b,i},\quad \text{对于 } a\in A, b\in B \text{ 且 } a\to b \tag{4}$$

式(4)说明，若运算 a 在控制步 j 的前 E_a-1 步内开始执行，则在控制步 j 中应仍在执行.

各模块在其等待时间(或延迟时间)内最多只能执行一个运算. 即：

$$\sum_{a\in A\wedge a\to b}\sum_{j=i}^{i+L_b-1}x_{a,b,j}\leqslant 1,\quad \text{对于 } b\in B, 1\leqslant i\leqslant T_{step} \tag{5}$$

运算 a 在模块 b 中执行时，其它运算不能在 b 中执行. 即：

$$\sum_{a\in A\wedge a\to b}y_{a,b,j}\leqslant 1,\quad \text{对于 } b\in B, 1\leqslant j\leqslant T_{step} \tag{6}$$

若 $(a1,i1,a2,i2)\in L$(即运算 $a1$ 的第 $i1$ 个输出被运算 $a2$ 的第 $i2$ 个输入引用)，假设运算 $a1$ 由模块 $b1$ 执行，$a2$ 由模块 $b2$ 执行，则必然存在一条从模块 $b1$ 的第 $i1$ 个输出端口到模块 $b2$ 的第 $i2$ 个输入端口的连线. 这可以表示为：

$$W_{b1,i1,b2,i2} \geqslant \sum_{j=1}^{T_{step}} x_{a1,b1,j} + \sum_{j=1}^{T_{step}} x_{a2,b2,j} - 1 \quad \text{对于}(a1,i1,a2,i2) \in L, \text{且 } a1 \rightarrow b1, a2 \rightarrow b2 \quad (7)$$

这样,我们将分配问题形式化为在式(2)—(7)的约束下,求式(1)最小值的整数规划问题.其中约束条件有$O(T_{step} \times |A| \times |B|)$个,变量有$O(T_{step} \times |A| \times |B|)$个($T_{step}$为控制步数目,$|A|$为运算的个数,$|B|$为模块的个数).

3　结群分配算法

前面,我们介绍了分配问题的建模过程.在这个数学模型中,将分配问题形式化为整数规划问题.但对于较大规模的整数规划问题(几十个约束或变量),现有的算法并不是特别有效[5].为此,我们提出了一个启发式的结群分配算法(CAA),该算法在功能单元和寄存器个数最少的情况下,考虑整体的费用最省.引入等价集操作和等价集变量,来考虑互斥操作和互斥变量的分配;全局地考虑功能单元、寄存器和互连线路的分配,使得整个数据通路的硬件资源最省.

结群分配算法是一个重复迭代的过程.开始时,根据所需功能单元(或寄存器)的数目,对应于每一个功能单元(或寄存器)设定一个群.根据启发函数(称作结群费用函数)对操作(或变量)进行结群,当所有的操作(或变量)都结入某个群中时,将功能单元(或寄存器)依次分配给各个群,即完成了功能单元、寄存器和互连线路的分配.结群费用的计算需要考虑功能单元、寄存器和互连线路等因素之间的相互影响、相互作用,操作op_k(或变量v_k)结入群p_i中的结群费用(用$CV_{i,k}$表示)的计算方法如下:

(1) 若操作op_k(或变量v_k)不能由群p_i所对应的功能单元(或寄存器)实现,则$CV_{i,k}=\infty$.

(2) 否则,$CV_{i,k}$的值等于将操作op_k(或变量v_k)结入群p_i中所引起的数据通路费用的增值(数据通路的费用用功能单元、寄存器和互连线路折合到一位所需的标准门或CMOS管对数来计算).

结群分配算法的主要步骤如下:

(1) 为每个功能单元(或寄存器)设定一个相应的群;

(2) 计算各操作(或变量)的结群费用;

(3) 选择最小的结群费用$CV_{i,k}$,将操作op_k(或变量v_k)结入群p_i中;

(4) 若所有的操作(或变量)均已结入群中,则结束.否则,

(5) 计算群p_i与剩余操作(或变量)的结群费用,转(3).

可以证明结群算法可以在有限步内结束.

4　实验与结果

结群分配算法已在Sun4/110与华胜4075(UNIX系统)工作站上,用C语言实现.结群分配算法的输入由基于浓度扩散调度算法[4]的输出结果得到.

表1给出了一个基例Fifth-order Elliptical Wave Filter(EWF)[6]的分配结果与比较.其中加法需要1个控制步来完成;乘法需要2个控制步来完成;流水线乘法器的等待时间为

1 个控制步. 我们考虑在 19 个控制步中,对于流水线和非流水线乘法器的调度. 对于常量和其它系统一样,用 ROM 单元来存放. 图 1 给出了在 19 个控制步中采用流水线乘法器的分配结果.

表 1 EWF 的分配结果与比较

系统	功能单元数	寄存器数	多路器输入端数	标准门数	管对数
CAA	2+,2×	10	32	144	168
ELF[7]	2+,2×	11	30	148	177
HAL[8]	2+,2×	12	29	154	188
SAW[9]	2+,2×	12	34	164	195
CAA	2+,1×^{P}	10	30	140	165
ELF	2+,1×^{P}	11	30	148	177
HAL	2+,1×^{P}	12	26	148	183

+:加法器,×:乘法器,×^{P}:流水线乘法器

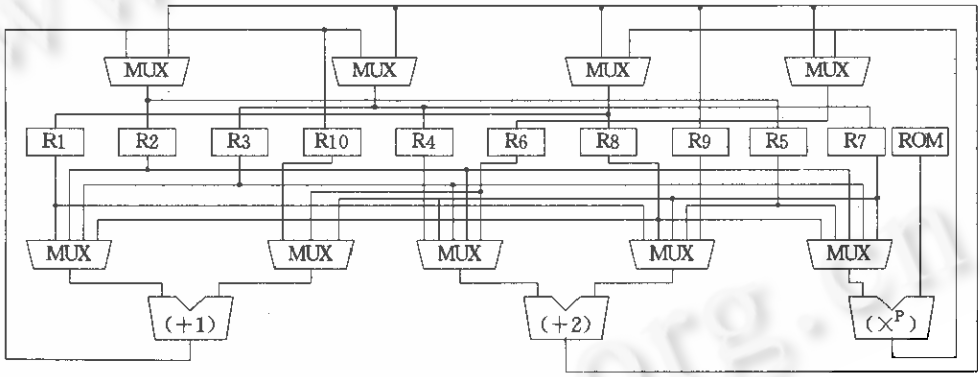

图1 EWF的分配结果

从表中可以看出,各算法所占用的功能单元是相同的,结群分配算法(CAA)所占用的寄存器最少,而所需的多路器输入端数较多. "标准门数"栏表示选用 TTL 电路工艺来实现数据通路时,各电路中寄存器折合到一位时所需的标准门数和一位多路器折合到每个输入端时所需的标准门数;"管对数"栏表示选用 CMOS 电路工艺来实现数据通路时,各电路中寄存器折合到一位时所需的管对数和一位多路器折合到每个输入端时所需的管对数. 我们选用 TTL 电路工艺的主从触发器组成寄存器和二选一多路器,它们折合后所用的标准门数分别是 8 个和 2 个;选用 CMOS 电路工艺的主从触发器组成寄存器和二选一多路器,它们折合后所用的管对数分别是 12 个和 1.5 个.

从实验结果来看,结群分配算法在较少的 CPU 时间内(≤0.2s),得到了与已发表的几种算法相似甚至更好的结果. 因此,这种形式化描述方法为解决分配问题建立了一个较好的数学模型.

参考文献

1 苏明,薛宏熙,洪先龙.数字系统的高层次综合.计算机辅助设计与图形学学报,1993,5(2):81—87.

2 McFaland M C, Parker A C, Camposano R. Tutorial in high—level synthesis. Proc. of the 25th DAC, Anaheim, 1988. 330—336.

3 Su M, Xue HX, Hong XL. A global scheduling algorithm for CDFG with nested conditional branches. Proc. of the 3rd CAD/CG, Hangzhou, 1993. 526—530.

4 苏明,元彦宏,薛宏熙等.基于浓度扩散的调度算法.计算机学报,1993,16(4):257—264.

5 Papadimitriou C H, Steiglitz K. Combinative optimization: algorithms and complexity. Englewood Cliffs: Printice Hall, 1982.

6 Kung S Y, Whitehouse H J, Kailath T. VLSI and modern signal processing. Englewood Cliffs: Prentice Hall, 1985. 256—264.

7 Ly T A, Elwood W L, Girczyc E F. A generalized interconnect model for data path synthesis. Proc. of the 27th DAC, Orlando, 1990. 168—173.

8 Paulin P G, Knight J P. Force—directed scheduling for the behavioral synthesis of ASIC's. IEEE Trans. on CAD, 1989,8(6):661—679.

9 Thomas D E, Hitchcok C Y, Kowalski T J *et al*. The system architect's workbench. Proc. of the 25th DAC, Anaheim, 1988. 337—343.

ALLOCATION AND ITS MODEL

Su Ming　Xue Hongxi　Hong Xianlong

(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Abstract　With the developing of IC and the increasing of integration level, high—level synthesis becomes a practical and efficient method of digital system design. In this paper, a task of high—level synthesis, allocation, is studied. First of all, allocation is modeled as an integer programming problem. Then a heuristic algorithm, called clustering allocation algorithm, is introduced. The experimental results show that the algorithm gets the same results or even better results than other published algorithms in less CPU time.

Key words　Allocation, clustering, data path, integer programming.