第34卷 第15期 计 算 机 工 程 2008年8月
Vol.34 No.15 Computer Engineering August 2008
·软件技术与数据库· 文章编号：1000—3428(2008)15—0044—03 文献标识码：A 中图分类号：TP311

资源约束的FPGA流水线调度

宋 健[1]，葛颖增[2]，窦 勇[1]
(1. 国防科技大学计算机学院，长沙 410073；2. 河南公安高等专科学校信息安全系，郑州 450002)

摘 要：循环是程序中十分耗时的部分，流水线能够加速循环执行但需要大量运算资源。由于FPGA资源有限，将循环代码在FPGA上加速时手动设计流水线不具有实际可行性。该文使用软件流水将循环自动映射到FPGA上，并实现资源约束下的流水线调度。通过探索整个或者局部资源组合空间，可以选择一个性能和面积比较平衡的设计。

关键词：流水线；模调度；资源约束；空间探索

FPGA Pipeline Scheduling Under Resource Constraints

SONG Jian[1], GE Ying-zeng[2], DOU Yong[1]
(1. School of Computer, National University of Defense Technology, Changsha 410073;
2. Department of Information Security, Henan Public Security Academy, Zhengzhou 450002)

【Abstract】Loop is the time-consuming part in program. The pipeline can accelerate its execution but needs too much computing resources. Due to limited resources, it's impractical to pipeline the loop by hand in FPGA. In this paper, the loops are mapped automatically onto FPGA using software pipelining, and the pipeline scheduling under resource constraints is implemented. By exploring the whole or part combination space, a design which balances performance and area can be chosen.

【Key words】pipeline; modulo scheduling; resource constraints; space exploration

1 概述

流水线是一种增加数字电路吞吐率的常用设计方法，流水化电路能够以更高的时钟频率运行。由于逻辑函数和互连会带来较长的组合路径延迟，流水线对基于FPGA的系统尤为重要，因此现在许多高性能FPGA电路都使用了流水线技术。

在高级综合研究中，针对循环的流水线调度得到了广泛的研究。Sehwa[1]系统对一个不含迭代间相关(Loop-Carried Dependence, LCD)循环进行流水化，在资源约束下最小化循环单次迭代执行时间。ALPS[2]采用整数线性编程(ILP)方法对循环流水线调度进行形式化研究，虽然ILP方法能够得到最优解，但算法复杂度太高而不适应规模很大的问题。

软件流水是体系结构研究中一种开发指令级并行的方法，它非常适用于超长指令字计算机。编译器将来自不同循环迭代的指令组织成一个新循环体，当这个循环体执行时表现出流水线的特性。

由于其固有的相似性，将软件流水的思想和算法应用于FPGA流水线调度是可行的。文献[3]将软件流水中的模调度算法和重定时技术相结合，进行FPGA循环流水线调度。文献[4]提出一种FPGA流水线设计空间探索方法，设计一个模块库在调度时进行模块选择和资源共享，但算法输入是吞吐率约束，在实现时不容易确定。

本文根据FPGA资源有限的特性，提出一个有效设计流程，采用软件流水中的迭代模调度技术实现包含循环的高级语言算法到FPGA平台寄存器传输级描述的流水线调度，并通过人工参与指导迭代过程，在目标系统的性能和面积之间进行平衡，一个流水线调度器原型已经构建并通过了基准程序测试。

2 设计方法学

本文采用图1所示流程将高级语言描述的算法映射到FPGA上，在提供待处理的算法和资源约束条件下，实现部分过程的自动化。

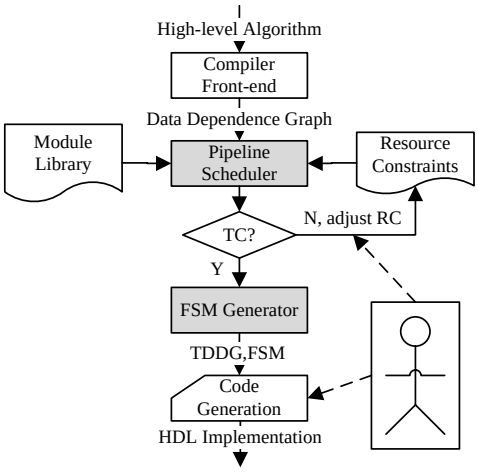

图1 FPGA流水线调度流程

首先将高级语言描述的算法输入编译器前端工具，获取程序中循环执行频率和迭代次数等信息，分析程序数据相关性，得到每个循环的数据相关图(Data Dependence Graph, DDG)。接着将DDG送入流水线调度器中，同时从模块库中

基金项目：国家自然科学基金资助项目(60633050)
作者简介：宋 健(1982－)，男，硕士研究生，主研方向：高性能体系结构技术；葛颖增，讲师；窦 勇，教授、博士生导师
收稿日期：2007-08-27 **E-mail：**songjian6cc@sina.com

选择一个功能部件集合作为约束也送入调度器。当所有循环完成软件流水调度后，得到它们执行时间的估计，根据运算量大小对处于不同执行路径上的循环进行平衡，重新分配资源和调度。当得到一个比较合理的调度后，将结果信息送入状态机生成器，生成各个循环的控制状态机。最后综合所有信息生成寄存器传输级实现的 HDL 代码，其中资源约束调整和代码生成需要设计者参与，现阶段并不能自动实现。

3 FPGA 流水线调度

3.1 问题形式化定义

FPGA 流水线调度的输入包括数据相关图。通过编译器前端工具，对高级语言程序进行数据流分析，为每个循环体构建一个数据流图。通过“if 转换”消除循环体中的条件分支，通过静态单赋值消除反相关和输出相关，并在数据流图上标注不同循环迭代之间的数据相关，构成数据相关图。

数据相关图 DDG 由<*V*, *E*>二元组构成。其中，*V* 表示结点集合，每个结点表示数据路径中的操作，操作 x_i 具有属性 $latency(x_i)$，表示执行该操作的模块输出延迟；*E* 表示边集合，它连接 2 个具有真相关的操作。边的一个属性是相关距离(distance)，表示 2 个相关操作之间间隔的迭代次数，另一个属性是延迟(delay)，表示前驱和后继操作启动的最小时间间隔，它与边的相关类型、2 个操作的输出延迟有关。

本文考虑的资源主要是功能部件，设 *O* 表示循环体中的操作类型集合，x_i 表示某种操作，$Num(x_i)$表示操作 x_i 的个数，$R(x_i)$表示分配给该 DDG 用来执行操作 x_i 的可用资源数目，资源约束可以表示为 $1 \le R(x_i) < Num(x_i),\ x_i \in O$。

流水线调度定义为

$$S: \mathrm{DDG} \xrightarrow{R} \mathrm{TDDG}$$

其中，*R* 表示资源约束；TDDG 表示调度后的数据相关图，它包含调度信息，每个操作都被分配到相应控制步上。此外，TDDG 还包含流水线启动间隔(Initiation Interval, II)和循环的总执行时间估计 *T*。*II* 表示循环下一次迭代启动时刻相对于当前迭代必须停顿的时间(以周期数计算)。给定初始资源约束为 R_0，得到调度 S_0，判断是否满足时间约束，如果不满足增加资源，重新执行调度直到给定一个资源约束 R_i 得到合理的调度 S_i。

3.2 迭代模调度

模调度是实现软件流水线调度的一种方法，但计算复杂度高，难于实现，而且调度结果是次优的。文献[5]提出的迭代模调度算法能够有效降低复杂度，能在较小的流水线启动间隔下调度成功，比传统的循环展开加代码复制调度算法更加高效。迭代模调度适应实际的机器模型，已经在许多编译器和体系结构研究项目中实现。

调度算法首先计算资源约束和回路(recurrence)约束下的循环启动间隔，取两者最大值作为调度时尝试的初值。设资源约束下的最小启动间隔为 *ResMII*，由相关回路引起的最小启动间隔是 *RecMII*，则启动间隔的下界 *MII*=max{*ResMII*, *RecMII*}，计算公式如下：

$$ResMII = \max_{x \in O}\{\frac{Num(x)}{R(x)}\},\quad RecMII = \max_{c \in C}\{\frac{Delay(c)}{Distance(c)}\}$$

其中，*C* 表示 DDG 中的基本回路集合；*c* 是一个基本回路(elementary circuit)；$Delay(c)$表示回路上所有边的延迟之和；$Distance(c)$表示回路上所有边的相关距离之和。

计算出 *MII* 之后，迭代算法从 *II*=*MII* 开始尝试调度。根据优先级函数选择当前未调度队列中优先级最大的操作，计算它可以调度的最早时间步 *Estart*，计算 *Estart* 时遵守与前驱操作的约束关系，然后在 *Estart* 到 *Estart*+*II*-1 区间内寻找时间槽，根据模调度思想，如果在时间步 *t* 上产生资源冲突，在 $t \pm k \times II$ 时间步上也会产生冲突，没必要尝试更长的时间段。找到时间槽 *slot* 后，取消所有和该操作产生资源冲突和相关约束操作的调度，然后将该操作调度到 *slot* 时间步上。每调度一个操作将预算值 *budget* 减去 1，当预算值耗尽并且还有操作没有被调度，将 *II* 加 1，继续上述过程直到所有操作调度成功。

在迭代模调度中，使用模保留表(modulo reservation table)记录已调度操作和它所使用的功能部件。MRT 的一维是启动间隔 *II*，另一维是功能部件序列，如果操作调度到时间步 *t* 并且使用资源 *r*，将((*t* mod *II*), *r*)加入到 MRT 中。由于使用流水功能部件，在 MRT 的每一个入口都可以启动新操作。

3.3 模块库

现有许多算术运算已经在 FPGA 上实现，这为复杂算法的加速提供便利。将单个算术运算实现称为模块(module)，或者称为功能部件(FU)。预先构建一个模块库，将应用算法映射到 FPGA 上时，可根据实际情况选择合适的模块集合来实现，如果现有模块不满足条件，开发新的模块加入到库中，增加了设计灵活性。

多周期流水模块能提高吞吐率和并行性，本文模块库中的模块都是流水实现的，单周期模块可以看作是一站流水线，不包含多周期非流水模块。本文使用流水模块开发循环体内的并行性，使用软件流水开发循环不同迭代之间的并行性。

3.4 资源空间探索

根据上文定义，对于集合 *O* 中的某种操作 x_i，当可用资源数目 $R(x_i)$小于这种操作的总数目 $Num(x_i)$时是资源约束的。为了考察资源数目大小对于调度结果的影响，穷举所有可能的资源数目组合，共有 $\prod_{x_i \in O} Num(x_i)$ 种组合方式。当集合 *O* 很大而且 $Num(x_i)$也很大时，可能的组合方式就会非常多，在这个空间内进行调度的时间就会急剧增加。为了减小探索空间规模，可以给定一个初始资源约束 R_0，对于某种类型操作 x_i，只在[1, $R_0(x_i)$]范围内进行组合，从而有效减少探索空间。统一的资源约束表示如下：当 α=1 时进行完全空间探索，当 0< α<1 时进行部分空间探索。

$$R_0(x_i) = \lceil \alpha \times Num(x_i) \rceil$$

由于存在回路相关，DDG 中可能存在环，在这种情况下，增加资源数目并不一定能够减小启动间隔 *II* 和循环体一次迭代执行时间。定义资源约束上的偏序关系：

$$R_1 < R_2 \Leftrightarrow \exists O_s \subseteq O \text{且} O_s \neq \Phi$$
$$\forall x_i \in O_s, R_1(x_i) < R_2(x_i)$$
$$\text{且} \forall x_j \in O - O_s, R_1(x_j) = R_2(x_j)$$

其中，$R_1(x_i)$<$R_2(x_i)$和 $R_1(x_j)$=$R_2(x_j)$表示整数小于和相等关系。因此，给定初始约束 R_0，可能存在一个资源配置 R_k 满足 R_k<R_0，且能够获得相同小的启动间隔 *II*。满足条件的 R_k 不是唯一的，可以根据模块实现代价和关键性来选择一个资源配置。如果设计者不显式给出初始约束，流水线调度器设置 α=0.5，从这个缺省值开始执行调度并逐步减少模块的数目。

通过命令参数，可以让调度器只在初始资源约束 $R_0(x_i)$上调度，而不是在[1, $R_0(x_i)$] 资源组合空间内进行探索。

3.5 目标电路执行模型

每个循环可以表示成一个 DDG 并映射到一个处理单元

(PE)上执行。每个 PE 由一个数据分派器(data dispatcher)和若干个宏功能部件(mFU)组成。每个 mFU 由 1 个功能部件(FU)、1 个寄存器队列(RegQueue)和 2 个多路选择器组成，选择器为 FU 提供输入，RegQueue 寄存结果供 mFU 使用。数据分派器根据调度器输出信息，在不同控制步将操作数、多路选择信号和 FU 使能信号送入 mFU，并将结果加载到队列中。硬件执行模型和 mFU 结构如图 2 所示。

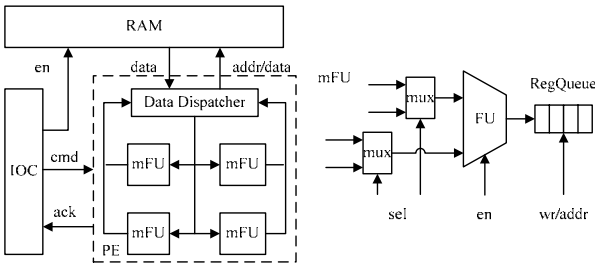

图 2　硬件执行模型和 mFU 结构

数据分派器使用一个具有 *II* 个状态(不含初始和结束状态)的 FSM 实现，将每个 mFU 的控制信号组成一个向量，*II* 个状态的控制信号组成一个控制矩阵。在每个状态下，从矩阵中选择一行控制信号来驱动 mFU 工作。

循环所需要的外部数组数据保存在 RAM 中，由一个 I/O 控制器(IOC)负责为 PE 提供数据和将结果写回 RAM。读状态机负责数据输入，必须在 *II* 个周期内将循环下一次迭代需要的数据送到 PE 中；写状态机负责结果写回，每隔 T_0 个周期写回一次结果，T_0 为流水线调度后循环体执行时间。

对于循环中赋值语句右侧的数组读操作，如果它是前面迭代或当前迭代产生的，则不需要访问 RAM，直接从缓冲器读取；如果是独立于迭代的，通过 IOC 从 RAM 中预先读取。对于赋值语句左侧的数组写操作，如果与后续操作不存在输出相关，可通过 IOC 将结果写回到 RAM 中。通过消除冗余的数组读写操作，能够有效降低访存时间，特别在数字图像的卷积类处理中，存在大量邻域像素复用，可将下次迭代用到的数据缓存在寄存器中。

4　实验与分析

本文选择 MCNC 高级综合基准程序集 HLSynth91 中 2 个典型程序——差分方程 DiffEq 和五阶 Elliptic 滤波器进行测试，并在整个资源空间上进行探索。和已有的表调度算法相比，在减小流水线启动间隔方面能够取得更好效果。

在实验中，使用 2 站流水线乘法器和单周期加法器。

设操作总数为 *N*，根据平均数不等式，完全空间探索时调度次数表示为

$$\prod Num(x_i) \le \left(\frac{\sum Num(x_i)}{|O|}\right)^{|O|} = \left(\frac{N}{|O|}\right)^{|O|}$$

一般来说，常用算术运算包括加法、乘法和除法，因此 $|O| \le 3$，完全空间探索的复杂度为 $O(N^3)$，通过设置约束参数 α 进行部分空间探索，能够有效降低调度执行时间。

4.1　差分方程 DiffEq

差分方程算法包含 6 个乘法和 5 个加法操作。把 DiffEq 看作不含 LCD 时的调度(实际上含有 LCD)如图 3 所示，不同乘法器个数下的曲线标注在一个图中。当加法器增加到某个值后，继续增加并不能减小 *II*，这种情况在考虑 LCD 的调度中更加明显。如果考虑 LCD，在数据相关图中形成环的可能性增大，使得必须串行执行的操作增多，配置更多资源是多余的。在表 1 中，由于 DDG 中存在一个长度为 6 的环，因此启动间隔最小值不会低于 6。区间[2,5]表示在加法器数目从 2 变化到 5，乘法器配置 1 个情况下，流水线调度的启动间隔都为 6。

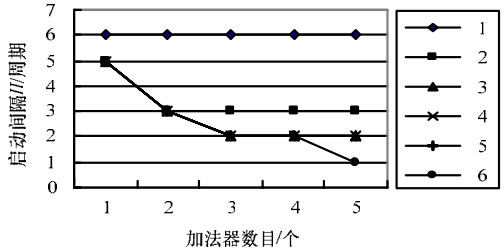

图 3　DiffEq 完全空间探索(不含 LCD)

表 1　DiffEq 完全空间探索(含有 LCD)

加法器数目	乘法器数目	启动间隔 *II*
1	1	7
[2,5]	1	6
1	2	6
[2,5]	2	6
1	[3,6]	6

4.2　五阶 Elliptic 滤波器

五阶 Elliptic 滤波器含有 26 个加法和 8 个乘法操作，具有较多的循环内数据相关和循环迭代之间相关。由于操作数目较大，只探索部分资源组合空间，设置 α=0.5，即加法器不超过 13 个，乘法器不超过 4 个，也分 2 种情况进行空间探索。在图 4 中，增加功能部件能够减小 *II*，当增加到一定值时变化比较缓慢。在表 2 中，由于存在数据相关环，因此 *II* 最小值只能取到 16，当功能部件增加到某个值时 *II* 保持不变。4 个加法器和 1 个乘法器或者 3 个加法器和 2 个乘法器能够获得 *II*=16，从功能部件实现代价考虑，选择前者更为合理。

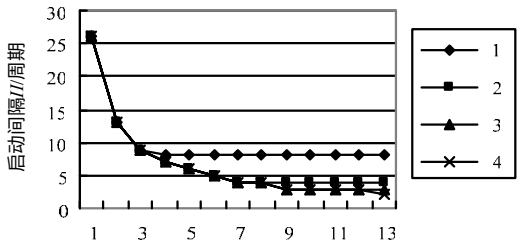

图 4　Elliptic 部分空间探索(不含 LCD)

表 2　Elliptic 部分空间探索(含有 LCD)

加法器数目	乘法器数目	启动间隔 *II*
1	[1,4]	28
2	[1,4]	18
3	1	17
[4,13]	1	16
[3,13]	[2,4]	16

5　结束语

本文提出了一种将高级语言描述的循环代码映射到 FPGA 平台上的设计流程，将软件流水的迭代模调度算法应用到 FPGA 流水线设计，通过开发的原型调度器自动生成满足资源约束且最大化并行性的调度方案。实验结果表明，生成的调度功能正确并且能够取得较好并行性。

由于迭代模调度算法本身的特性，需要大量寄存器来实现硬件电路，因此今后将着重研究如何减小寄存器需求。此外，寄存器分配策略和 HDL 代码自动生成也需要深入探讨。

（下转第 50 页）