

自动映射多循环程序到有限 FPGA 资源的 参数化流水线模板

董亚卓¹⁾ 窦 勇²⁾ 宋 健³⁾ 刘明政⁴⁾

¹⁾(中国人民解放军 91655 部队 北京 100036)

²⁾(国防科学技术大学计算机学院 长沙 410073)

³⁾(中国人民解放军 61785 部队 北京 100075)

⁴⁾(总后勤部科学研究所应用室 北京 100071)

摘 要 FPGA 为加速计算密集型应用提供了一个灵活高效的平台,然而,由于片上资源有限,在一些情况下,需要将大规模应用中包括的多个循环程序分别映射到 FPGA 上执行,当一个循环程序执行完毕后,需要重新配置 FPGA 以执行下一个循环程序,FPGA 重构过程在整个程序执行过程中占用了较多时间.文中设计了一个参数化流水线模板,并提出了相应的指令分配调度策略,实现了自动将多循环程序顺序映射到目标 FPGA 片上系统,同时在程序切换时,不需要进行 FPGA 重构.实验结果表明,对每个循环程序,文中设计的流水线模板能达到与专用硬件结构相当的执行节拍,同时节约了程序切换时的重构时间.

关键词 循环;FPGA;流水线模板;指令调度

中图法分类号 TP302

DOI号: 10.3724/SP.J.1016.2009.00152

Pipelined Template for Mapping Multiple Loop Nests on FPGA with Restricted Resources

DONG Ya-Zhuo¹⁾ DOU Yong²⁾ SONG Jian³⁾ Liu Ming-Zheng⁴⁾

¹⁾(Unit 91655, People's Liberation Army, Beijing 100036)

²⁾(School of Computer, National University of Defense Technology, Changsha 410073)

³⁾(Unit 61785, People's Liberation Army, Beijing 100075)

⁴⁾(Application Laboratory, Logistic Science and Technology Research Institute, Beijing 100071)

Abstract FPGA provides a convenient and flexible solution to speed up loop-intensive algorithms. However, these loop nests in a large scale application have to be mapped onto the target FPGA orderly because of the limited resources on-chip. FPGA reconfiguration which needs a long time is inevitable when switching between the loop nests. This paper presents a pipelined template and instruction schedule method corresponding to execute all the loop nests in sequence without FPGA reconfiguration. Experiments show that the pipelined template can achieve a comparative execution cycles for a loop comparing with the special hardware and without the need of FPGA reconfiguration.

Keywords loop; FPGA; pipelined template; instruction schedule

收稿日期:2007-12-22;最终修改稿收到日期:2008-10-14. 本课题得到国家自然科学基金重点项目(60633050)和 91655 部队青年科研基金资助. 董亚卓,女,1979 年生,博士,主要研究方向为可重构计算和高级综合技术. E-mail: dongyazhuo@nudt.edu.cn. 窦 勇,男,1966 年生,研究员,博士生导师,主要研究领域为可重构计算和高性能计算机体系结构. 宋 健,男,1982 年生,硕士,助理工程师,主要研究方向为网络安全和信息检索技术. 刘明政,男,1981 年生,硕士,助理工程师,主要研究方向为图像和信号处理技术.

1 引言

现场可编程门阵列(Field Programmable Gate Array, FPGA)作为可编程逻辑器件的典型代表,它的出现及日益完善适应了当今时代的数字化发展浪潮,它正广泛应用在现代数字系统设计中. FPGA 不仅具有设计灵活、性能高、速度快等优势,而且上市周期短、成本低廉. 同时,随着设计技术和制造工艺的完善,器件性能、集成度、工作频率等指标不断提升, FPGA 已越来越多地成为系统级芯片设计的首选. 然而,现有 FPGA 仍然存在两点不足:(1)片上资源有限,不能满足规模较大的应用;(2)FPGA 重构过程需要较长时间(ms 量级).

FPGA 设计的首要任务是考虑如何利用系统级设计中提供的各种资源,同时满足性能与代价约束,找到一个最好的平衡点来实现从功能到结构再到实现的转换,也就是设计空间探索技术. 根据片上资源数量和目标程序规模,目前基于循环程序的设计空间探索技术分为两个研究方向:(1)当片上系统资源足够时,循环展开以进一步增加并行性,文献[1-2]分别研究了在资源足够的前提下,将单个循环程序和多个循环程序执行展开的设计空间探索方法.(2)系统资源不足,在这种情况下又有两种解决方案:①循环分片^[3-4],将一个较大的循环程序分成多个满足目标片上系统资源约束的小的循环片,分别映射执行;②软件流水^[5],即在有限的硬件资源上重叠执行不同迭代层的不同语句,增加并行. 相对于循环分片,软件流水是一种更灵活高效的解决办法,其指令调度方法在很大程度上决定着软件流水并行性的优劣,是一个关键问题.

目前,有两种方法实现软件流水指令调度“move-then-schedule”^[6]方法和“schedule-then-move”方法.“move-then-schedule”技术是先移动指令增加并行,再执行调度,这种方法在实现时面临的困难是,很难在程序执行之前明确判断指令最优的调度方向和距离.“schedule-then-move”方法又有两种方案实现. 一种是“unroll-while-scheduling”^[7],在执行循环展开的同时,执行指令调度;另外一种为模调度(modulo scheduling)^[8],设计思想是调度一个循环层的指令,之后每隔一个启动间距,不断重复这一调度过程,保证不同的循环层次间在同一时刻执行的指令满足数据相关和控制相关. 模调度面临的一个关键问题是中间结果的暂存和运算部件的互

连. ShiftQ^[9]作为面向特定应用的嵌入式程序加速和自动优化系统 PICO^[10]的一部分,为模调度提供了一种代价较小的中间结果存储和互连结构. 除此之外, Hadda^[11]等人用不等式方程表示资源限制,在避免冲突的前提下尽可能开发细粒度并行. 文献[12]基于整数线性规划和分支搜索方法,提出了一种满足目标代价约束的模调度方法. 上述这些工作都是针对单个循环程序.

在图像、多媒体处理、模式识别等应用中,存在大量规模较大的应用,其内部包括多个循环,而片上系统的资源是有限的,在一些情况下并不能将一个应用中包括的所有循环全部映射到目标可重构芯片上,这时,通常情况下的做法是将各个循环程序分别映射到可重构硬件上执行,每更换一个循环,则重新执行硬件配置,而 FPGA 的配置过程是很慢的(ms 量级),因而,这种方法在很大程度上制约了程序的执行速度.

本文基于软件流水的模调度思想和 ShiftQ 体系结构模型,提出了一种将一个应用中的多个循环程序映射到有限 FPGA 资源的流水线模板,模板中设计的运算单元和互连结构,对该应用中的所有循环程序通用. 这些循环程序可以顺序映射到这个模板上执行,在程序切换时,无需重新配置 FPGA. 本文的设计目标是在满足资源约束的前提下,保证目标应用中的所有循环程序总体执行时间最短.

本文第 2 节介绍本文的相关研究背景;第 3 节介绍参数化的流水线模板体系结构;第 4 节介绍参数提取算法;第 5 节给出实验结果和性能分析;最后做总结.

2 研究背景

本文的研究基于软件流水指令调度方法和 ShiftQ 体系结构模型. 软件流水^[13]的主要思想是在资源限制、相关性(循环体内和体间)和周期性的限制下,重新组织循环体,形成执行时间更短的新循环. 它通过重叠不同循环体的执行来提高速度. 即在一个循环体尚未执行完毕之前,启动下一个循环体. 二者之间的时间差称为启动间距(Initiation Interval,简称 II). 调度结果由装入、核心和排空 3 部分组成. 在装入部分,前 p 个循环体以 II 为间隔依次启动,随之出现一个重复模式,即核心,其长度恰好等于 II. 它被反复执行直到所有循环体都被启动为止. 核心部分结束后进入排空部分,完成最后 p

个已启动但未执行完的循环体. 核心包括循环体中出现的所有操作,但这些操作可以来自不同的循环体. 只要循环体的循环次数足够大,执行时间将主要消耗在核心的执行上. 软件流水的主要目标就是找到尽可能小的 II.

II 受两个因素影响^[8]:资源限制和相关限制. 资源限制包括功能单元 FU(Function Unit)、总线等的限制,理论上存在一个 ResMII(最小的资源限制 II),可以保证指令间的并行性而不引起资源冲突. 相关限制包括数据、控制等相关,理论上存在一个 RecMII(最小的相关限制 II),可以保证指令间的并行性而不引起相关冲突. 启动间距 $II = \text{Max}(\text{ResMII}, \text{RecMII})$,即最佳启动间距取决于 RecMII 和 ResMII 的最大值. 设循环体内使用某种资源的次数为 Num_OP,系统提供的该种资源总数为 Num_FU,则 $\text{ResMII} = \text{Max}(\text{Num_OP} / \text{Num_FU}) (0 \leq i \leq \text{循环体内使用的不同资源类型总数})$.

ShiftQ 为软件流水的模调度提供了一个代价较低的中间结果暂存和运算部件互连结构,其核心思想为:每个运算单元配置一个由扩展寄存器组组成的 buffer,保存中间结果. buffer 中的寄存器采用轮转策略,保证一个中间结果在传递完毕后才会被替换. 图 1(a) 中所示为基本的 ShiftQ 体系结构,其

中 V_1, V_2, \dots, V_5 为扩展寄存器,组成两个 buffer,每个 buffer 与一个运算单元相连,保存该运算单元产生的中间结果. 当一个新的中间结果产生后,buffer 中的数据沿着寄存器逐个向下移动,最下面一个寄存器中的数据被淘汰,如图 1(b) 所示, $C_0 \dots C_{II-1}$ 为一系列的控制信号,控制寄存器中的数据流动. 扩展寄存器通过交叉开关与运算单元的输入端相连,实现中间结果的传递.

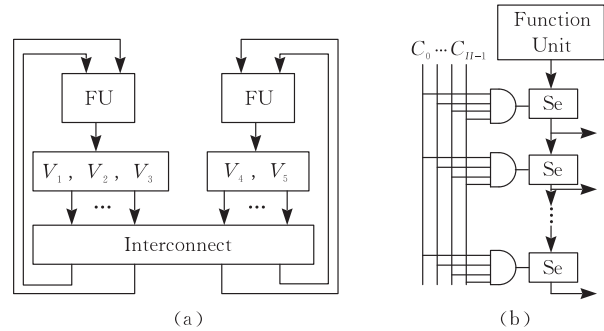


图 1 ShiftQ 基本结构及其 buffer 控制

3 参数化流水线模板

给定目标应用和资源有限的 FPGA,生成一个统一的流水线模板,能够将该应用中的所有循环程序顺序映射到这个模板上执行. 图 2 所示为本文提出的流水线模板体系结构模型.

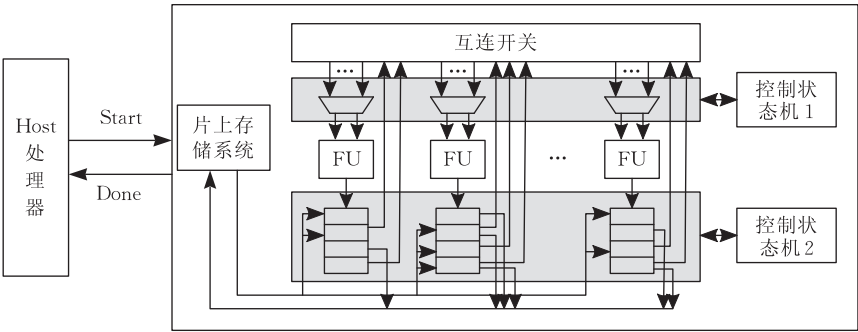


图 2 流水线模板体系结构

流水线模板结构中包括多个运算单元 FU,每个 FU 连接一个扩展寄存器组,保存该 FU 产生的中间结果,控制状态机 2 用于实现扩展寄存器中数据的流入、查询和替换操作. 扩展寄存器与 FU 的输入端通过互连开关连接. 可能有多个寄存器连接到某个 FU 的同一输入端口,控制调度状态机 1 用于控制 MUX 从多个输入中进行选择. 源操作数通过 LOAD 操作读入扩展寄存器,计算结果通过 STORE 操作保存到片内存储系统. 在接收到来自 Host 处理器的 Start 信号后,程序启动执行,执行完

毕产生 Done 信号,反馈给 Host 处理器. 本文提出的流水线模板是一个参数化的体系结构,参数值由具体目标应用的特点决定. 定义如下参数:

(1) 运算单元 FU 的类型和个数用 $FU_i, \text{NUM_FU}[i]$ 表示.

(2) 每个 FU 对应的扩展寄存器个数用 $\text{num_reg}_1, \text{num_reg}_2, \dots$ 表示,第 i 个 FU 对应的扩展寄存器用 $\text{reg}_i^1, \text{reg}_i^2, \dots$ 表示. 扩展寄存器的个数是由该 FU 要保存的中间结果个数和中间结果替换时机决定的.

(3) 每个 FU 的输入端口个数用 $num_Inport_1, num_Inport_2, \dots$ 表示, 第 i 个 FU 的每个输入端口用 $In_port_i^1, In_port_i^2, \dots$ 表示. 输入端口个数是由该 FU 要执行的指令个数和操作数来源决定的.

(4) 扩展寄存器与 FU 输入端口的连接关系集合表示为 $connect$, 如其中 $\{(reg_2^1, In_port_1^3)\} \in connect$ 表示第 2 个 FU 的第 1 个寄存器与第 1 个 FU 的第 3 个输入端口相连. 连接关系集合由中间结果的传递关系决定.



图 3 参数提取算法流程

算法输入为描述了数据流和控制流信息的程序中间表示 IR (Intermediate Representation) 文件, 输出为确定目标流水线模板的参数值. 以图 5 中的应用为例说明这一过程.

4.1 FU 配置算法

FU 配置算法用于确定流水线模板中 FU 的个数和类型, 它取决于应用程序的规模和片上系统资源数量. FU 配置算法的设计目标是在满足芯片资源约束的前提下, 尽可能让当前应用中的所有循环程序的总体执行时间最短.

假设有 n 个循环程序, m 种类型的操作. 假设 n 个循环程序在资源足够时的最小启动间距分别为 II_1, II_2, \dots, II_n . 第 j 个循环程序, 定义 m 种操作的个数分别为 $num(op[j][1]), \dots, num(op[j][m])$. 假设这个循环程序需要的资源集合为 Res_j , 定义 $Res_j = \{num_FU_j^1, num_FU_j^2, \dots, num_FU_j^m\}$, 其中 $num_FU_j^i$ 表示第 i 类 FU 的个数. 给每个 FU 定义一个权重: w_1, w_2, \dots, w_m , 权重值是由将该运算单元映射到片上系统时需要的门数决定的.

为了将 MUX 的复杂度控制在一定范围内, 为 FU 定义一个最大输入端口个数 mux_max , 保证分配给每个 FU 的负载个数不能过多. 由此可以计算出每种类型的 FU 的最小需求个数为 $min_FU_j^i = num(op[j][i]) / mux_max$. 则循环 j 需要的最小资源集合为 $Res_{min}^j = \{min_FU_j^1, \dots, min_FU_j^m\}$, 定义目标 FPGA 资源总量为 S_{FPGA} , 如果 $S_{FPGA} < Res_{min}^j$, 则该循环程序不能采用软件流水指令调度技术在目标 FPGA 芯片上映射. 定义目标应用中所有循环程序最小资源需求集合为 $Res_{min} = \bigcup_{j=1}^n Res_{min}^j$, Res_{min} 是保证该目标应用采用软件流水指令调度映射到 FPGA 需要的最小资源数量.

不同的循环程序, 参数值不同, 第 4 节的参数提取算法就是要为目标应用中的所有循环程序定义一套统一的参数, 进而生成一个通用的流水线模板.

4 参数提取算法

参数提取算法分为 5 个步骤: FU 配置算法、调度、优化、分配和组合, 如图 3 所示.

一个循环需要的最大运算部件个数也就是保证循环体中的每条语句对应一个运算部件, 即 $max_FU_j^i = num(op[j][i])$, 定义 $Res_{max}^j = \{max_FU_j^1, \dots, max_FU_j^m\}$, 则当前应用需要的最大资源数量为 $Res_{max} = \bigcup_{j=1}^n Res_{max}^j$, 这时, 运算单元和循环指令一一对应, 启动间距 $II = 1$. 在此基础上再增大片上系统资源, 也不会加快程序执行速度. FU 配置算法如图 4 所示.

S1. 如果 S_{FPGA} 大于 Res_{max} , 片上资源足够, 取 $Res = Res_{max}$, 为每个循环程序都生成操作和运算部件一对一的硬件结构, 启动间距 $II = 1$.

S2. 如果 Res_{min} 已经超出了片上系统资源约束, 则当前应用无法完全映射到目标 FPGA 上, 应用中的循环程序要被分块执行, 不在本文的讨论范围之内.

S3~S5. 如果 $Res_{min} \leq S_{FPGA} < Res_{max}$, 设计启发式搜索算法配置 FU, 其设计思想是从最小的资源集合 Res_{min} 开始, 适当增加某种类型的 FU 个数, 最大程度上减少程序执行节拍. 因为 II 的值与程序执行节拍成反比, 所以我们只要考虑 II , II 的值越小越好. 一旦 Res 超出片上系统资源约束, 算法终止.

S4. 在当前资源 Res 下, 计算每个循环程序对应的 II' . 定义 $delay_i = II'_i / II_i$, $delay_i$ 标识了执行节拍被最长延迟的循环程序, 这个算法的调整目标就是减少这些循环程序的执行延迟.

S5. 假设有 p 个循环程序的执行延迟都等于最大执行延迟 $delay$, 尝试调整这 p 个程序, 最后选择调整代价最小的一个作为最终的调整目标. II 是由循环程序中的操作个数 Num_OP 和相应运算单元的个数 Num_FU 决定的 $II = Num_OP / Num_FU$. 计算当前循环程序中由每个运算单元所决定的 II

值,找到决定 II 最大值的 FU,从 1 开始增加相应的运算单元,直到 II 降低,如果某个 FU 增加了 4 个,II 值仍没有改善,则放弃对该循环程序的调整,在这种情况下,很可能是在这个循环程序中存在控制相关,II 值由控制相关距离决定,而增加运算单元的个数并不能减少 II 的值.最后,分别计算这 p 个循环程序的调整代价,找到代价最小的一个作为最终的调整目标.

图 5 给出的例子中,假设 FU 资源有限,配置如下:2 个 LOAD,1 个 STORE,3 个 ADD 和两个 MUL.

4.2 调 度

在 FU 的个数和类型确定后,调度过程用于对这些 FU 进行指令分配.本文沿用文献[8]中的模调度方法,其基本思想为根据指令的启动时间、执行延迟和相关关系,为每条指令定义执行时间槽,选择在该时间槽内空闲的 FU,将该指令分配给该 FU 在该时刻执行,如果不成功,则增大 II,重新执行调度过程.图 5(c)和(d)中所示为示例应用中的两个循环程序的部分调度结果,这两个循环程序的启动间距分别为 2 和 3,执行延迟分别为 8 和 9.

4.3 优 化

调度算法得到的结果可能是不平衡的,也就是同一类型的若干 FU 中,某些 FU 的负载个数可能比其它一些 FU 的负载个数大很多,负载个数多的 FU 对应的输入多路选择器 MUX 就会比负载少的 FU 对应的多路选择器 MUX 复杂,延迟相差就会比较大,而整体延迟是由延迟最大的 MUX 决定的,所以,我们希望同一类型的 FU 之间的负载个数尽

可能差不多,以保证总体延迟尽可能小.优化步骤的设计目标就是平衡同一类 FU 的负载个数.

假设对第 i 类 FU,第 j 个运算部件执行的操作个数为 a_i^j ,则平衡算法要尽可能地减小如下 S_i 的值,定义 $Mum_op[i]$ 为第 i 类操作的个数, $Mun_FU[i]$ 为第 i 类运算部件的个数.

$$S_i = \sum_{j=1}^{Num_FU(i)} (a_i^j - Num_OP[i] / Num_FU[i]);$$

在此基础上,对当前应用用到的所有类型的 FU 定义一个平衡度 S (即某种类型指令在所有指令中的数量比例与 S_i 的乘积):

$$S = \sum_{i=1}^N (Num_OP[i] / Num_OP \times S_i),$$

其中 $Num_OP = \sum_{i=1}^N Num_OP[i]$.

可见, S 的值越小, FU 的负载就越平衡.即每个 FU 执行的操作个数 a_i^j 越接近越好.对第 i 类 FU,定义最小和最大负载个数分别为

$$Num_{small}^i = \lfloor Num_OP[i] / Num_FU[i] \rfloor;$$

$$Num_{small}^i = Num_{small}^i + 1.$$

图 6 所示为负载优化算法.经过平衡算法之后,同一类型的 FU 之间的负载相差不会大于 1.定义 $Load1$ 和 $Load2$ 分别为 $FU1$ 和 $FU2$ 的负载个数,当两者相差不大于 1 时,算法终止.其中 $(MRT[FU_i][j] = 1)$ 表示运算部件 FU_i 在时刻 j 被调度,否则表示该运算部件在该时刻空闲.优化算法的设计思想是在不改变启动间距和功能正确性的前提下调整个别操作到相同类型的功能部件的同一个模时刻上执行.

```

S1  for (int i=0; i<II; i++)
    if (|Load1-Load2|<2) break;
S2  if ((MRT[FU1][i]==1)&&(MRT[FU2][i]==0)&&(|Load1-Load2|≥2))
    将 FU1 在 i 时刻调度的指令 OP 调度到 FU2 在 i 时刻执行;
    Load1--;
    Load2++;
S3  else if ((MRT[FU1][i]==0)&&(MRT[FU2][i]==1)&&|Load2-Load1|≥2)
    将 FU2 在 i 时刻调度的指令 OP 调度到 FU1 在 i 时刻执行;
    Load1++;
    Load2--;
    endif
endfor

```

图 6 负载优化算法

4.4 分 配

每个运算单元要执行的操作和调度时刻确定后,分配过程为每个 FU 产生扩展寄存器和交叉互连结构,本文采用 ShiftQ 体系结构^[9],在指令执行过程中,动态分配寄存器单元,一旦有新的结果产

生,则判断寄存器队列中最后一个寄存器中保存的中间结果是否使用完毕,如果使用完毕则将其淘汰,寄存器队列中的数据平移,将新中间结果保存在第一个寄存器中,如果最后一个寄存器中的中间结果还要使用,则在队列头再增加一个寄存器.因此,队

列长度由该 FU 执行的操作集合上所有结果变量的生存期共同决定.

4.5 组 合

为不同的循环程序生成的 ShiftQ 结构, 每个 FU 对应的虚拟寄存器个数和互连结构是不同的, 组合过程将这些 ShiftQ 结构组合, 生成一个对所有循环程序通用的流水线结构. 本文中, 我们取每个 FU 对应的扩展寄存器的最大值作为最终的扩展寄存器个数, 假设 n 个循环程序中, 第 i 个 FU 需要的寄存器个数分别为 $reg_i^1, reg_i^2, \dots, reg_i^n$, 则取最终寄存器个数 $num_reg_i = \max\{reg_i^1, reg_i^2, \dots, reg_i^n\}$. 假设 n 个循环程序中, 第 i 个 FU 需要的输入端口个数分别为 $inport_i^1, inport_i^2, \dots, inport_i^n$, 则取最终输入端口个数 $num_inport_i = \max\{inport_i^1, inport_i^2, \dots, inport_i^n\}$. 假设对各个循环程序, 寄存器与 FU 输入端口之间的连接集合分别为 $connect_1, connect_2, \dots, connect_n$, 则取互连关系的并集, 作为最终的互连关系集合 $connect = \bigcup_{i=1}^n connect_i$. 图 5(c) 所示为这两个循环程序最终产生的流水线模板结构.

5 实验与性能比较

实验分为两个部分: (1) 验证 FU 配置算法的有

效性; (2) 验证优化算法的有效性. 表 1 所示为各种运算部件的权重和执行节拍延迟, 其权重值是由将该运算部件映射到 FPGA 上所需门个数决定的, 这一权重值用于确定流水线模板中运算部件占用的资源总量.

表 1 运算部件的权重和执行节拍

FU 名称	占用门的数量	延迟
load	120	1
store	304	1
adder	317	1
multiplier	5503	2
fp_adder	6620	4
fp_multiplier	14953	3
fp_divider	17107	8
abs	323	1
min	247	1

5.1 FU 配置算法验证

本文选择 5 个测试程序, Livermore Kernel 包括 3 个循环基本块. HLSynth92 是一个高级综合系统的测试工具包, 从中选择两个循环基本块. 我们在 Medi-aBench 中选择了 3 个循环程序, Kalman filter 是一个包括 3 个循环程序的滤波器. 最后, 选择 3 个图像处理程序组成一个应用, 分别是 Sobel 边缘检测, DT_forward 前向扫描和 DT_backward 后向扫描程序.

表 2 给出在有限的 FPGA 资源约束下 FU 配置算法的执行结果. 第 2 列所示为该应用所需要的运

表 2 FU 配置算法的执行结果

测试程序	FU 配置	Kernel	最小资源			最大资源			资源配置		专用	
			Res_{min}	II	门数量	Res_{max}	II	门数量	Res	资源约束	II	II
Livermore Kernel	[ADD, MUL, LOAD, STORE, FADD, FMUL]	K1	[1,0,1, 1,1,1]	3	221314	[2,0,3, 1,2,3]	1	59397			1	1
		K2	[1,0,2, 1,1,1]	4	22434	[4,0,5, 1,2,2]	1	174756	[4,0, 5,1, 3,3]	66891	1	1
		K7	[2,0,3, 1,2,2]	4	44444	[6,0,9, 1,8,8]	1	175870			3	3
HLSynth92	[ADD, MUL]	DiffEq	[2,2]	6	11640	[5,6]	6	34603	[2,2]	11640	6	6
		Elliptic	[7,2]	16	13225	[26,8]	16	52266	[7,2]	13225	16	16
Mediabench	[ADD, MUL, LOAD, STORE, FADD, FMUL]	De3	[2,1,1, 1,1,2]	4	43087	[6,2,2, 4,4,6]	1	130562			2	2
		De4	[1,0,1, 1,1,1]	4	22314	[2,0,4, 2,2,4]	1	74774	[3,2, 4,2, 2,4]	86097	1	1
		De6	[2,1,1, 1,1,1]	4	28134	[7,4,2, 2,2,4]	1	98131			3	2
Kalman filter	[FADD, FMUL]	Pos	[2,3]	4	58099	[7,9]	4	180917	[2,3]	58099	4	4
		Align	[5,7]	4	137771	[20,28]	4	551084	[5,7]	137771	4	4
		Shape	[2,4]	4	73052	[7,15]	4	270635	[2,4]	73052	4	4
Image processing	[ADD, MUL, LOAD, STORE, ABS, MIN]	Sobel	[11,4,3, 1,1,0]	4	26486	[44,15,12, 1,2,0]	1	98883			2	2
		DT_forward	[9,3,5, 1,0,2]	4	20760	[35,10,17, 1,0,8]	1	70445	[35,10, 17,1, 1,8]	70768	1	1
		DT_backward	[8,3,5, 1,0,2]	4	20443	[30,10,17, 1,0,8]	1	68860			1	1

算部件种类,第 3~5 列分别给出最小资源需求数量、启动间距 II 和占用的门数量.第 6~8 列给出最大资源需求数量、启动间距 II 和占用的门数量.第 10 列给出对当前应用的资源约束(片上系统门数量),在这些约束下,第 9 列给出经过 FU 配置算法,生成的统一流水线模板的资源配置结果,向量中的各个数值与第 2 列的运算部件一一对应.第 11 列所示为各个循环程序在流水线模板结构下所达到的启动间距.第 12 列所示为在当前资源约束下(第 11 列所示),为这个循环程序生成专用硬件结构,所能达到的启动间距.

表 2 中的 HLSynth92 和 Kalman filter 两个测试中的循环程序随着 FU 个数的增加,启动间距没有改变,这是因为,在这两个测试程序中存在控制相关,启动间距是由控制相关距离决定的,而不是由资源数量决定的.从表 2 中可以看出,Mediabench 中的 De6 循环程序,流水线模板相对专用硬件结构,启动间距增大,延长了程序执行节拍.一般情况下,当一个应用中的各个循环程序使用的资源数量很不均衡的时候,如一个循环程序需要大量的加法器和

少量的乘法器,而另外一个循环程序需要大量的乘法器和少量的加法器,会出现这样的延迟.而对绝大多数应用,流水线模板和专用硬件结构可以达到相同的启动间距,即程序执行节拍相同.而本文提出的流水线模板,在执行该应用的循环程序时,无需重新配置硬件资源,节约了 FPGA 重构时间,所以,本文的流水线模板在加快程序执行速度上有一定优势.

5.2 优化算法验证

表 3 所示为优化算法的执行结果,其中标记为 \times 表示没有使用该功能部件,资源向量表示为每个程序配置的功能部件个数,粗黑体为取得优化效果的功能部件.例如在 K2 程序中,优化前某个 FMUL 比另一个多执行 2 个操作,优化后 FMUL 执行操作个数之差不超过 1.第 9 到 11 列给出程序优化前和优化后的平衡度的值以及平衡度降低的值.其中 HLSynth92 和 Kalman 滤波器的数据相关图中含有环路,流水线启动间距较大,调度是很不平衡的,因此优化效果非常明显.从表 2 中可以看出,优化后的结果平衡度降低,同一类型的 FU 之间负载之差小于 1,负载平衡.

表 3 优化算法执行结果

Kernel set	Kernel	FU 执行操作个数之差(优化前/优化后)						S 优化前	S 优化后	降低
		ADD	MUL	LOAD	STORE	FADD	FMUL			
Livermore kernel	K1	0/0	\times	0/0	0/0	1/1	0/0	0.33	0.33	0
	K2	0/0	\times	1/1	0/0	1/1	2/1	1.33	1.0	0.33
	K7	0/0	\times	0/0	0/0	1/1	1/1	0.67	0.67	0
资源		ADD:2, MUL:0, LOAD:3, STORE:1, FADD:3, FMUL:3								
HLSynth92	DiffEq	5/1	2/0	\times	\times	\times	\times	4.8	0.8	4
	Elliptic	7/1	2/0	\times	\times	\times	\times	5.2	0.8	4.4
资源		ADD:3, MUL:2								
Mediabench	De3	2/0	0/0	0/0	0/0	0/0	0/0	0.55	0.0	0.55
	De4	1/1	0/0	0/0	0/0	0/0	0/0	0.36	0.36	0
	De6	2/1	0/0	2/0	0/0	0/0	2/0	1.45	0.36	1.09
资源		ADD:3, MUL:2, LOAD:2, STORE:1, FADD:1, FMUL:2								
Kalman filter	Pos	\times	\times	\times	\times	2/1	1/1	3.08	2.31	0.77
	Align	\times	\times	\times	\times	2/1	0/0	1.85	1.23	0.62
	Shape	\times	\times	\times	\times	3/1	3/1	7.38	1.69	5.69
资源		FADD:6, FMUL:7								

6 总 结

本文基于软件流水的模调度和 ShiftQ 体系结构模型,提出了一种将多循环程序自动映射到有限 FPGA 资源的流水线模板.实验证明,本文提出的流水线模板可以达到与专用硬件结构相当的执行速度,同时在执行程序切换时,无需硬件重构,节省了重构时间,因而在提高程序执行速度上有一定优势.

随着应用程序规模的不断增大,互连网络和输入端多路选择器将变得越来越复杂,有关其复杂性的变化规律和降低策略是本课题下一步的研究目标.

参 考 文 献

[1] So B, Hall M, Diniz P. A compiler approach to fast design space exploration in FPGA-based systems//Proceedings of the ACM Conference Programming Languages Design and Implementation. Berlin, Germany, 2002: 165-176

- [2] Ziegler Heidi E, Hall Mary W. Evaluating heuristics in automatically mapping multi-Loop applications to FPGAs//Proceedings of the 13th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Monterey, California, 2005: 184-195
- [3] Abu-Sufah W A, Kuck D J, Lawrie D H. On the performance enhancement of paging systems through program analysis and transformations. IEEE Transactions on Computers, 1981, 30(5): 341-356
- [4] Allen J, Kennedy K. Automatic loop interchange//Proceedings of the 1984 SIGPLAN Symposium on Compiler Construction. Montreal, Canada, 1984: 233-246
- [5] Lam M. Software pipelining: An effective scheduling technique for VLIW machines//Proceedings of the the SIGPLAN 88 Conference on Programming Language Design and Implementation (PLDI). Atlanta, Georgia, ACM SIGPLAN Notices, 1988: 318-328
- [6] Moon S M, Ebciogu K. An efficient resource-constrained global scheduling technique for superscalar and VLIW processors//Proceedings of the 25th Annual International Symposium on Microarchitecture. Portland, Oregon, 1992: 55-71
- [7] Aiken A, Nicolau A. A realistic resource-constrained software pipelining algorithm//Advances in Languages and Compilers for Parallel Processing. London: Pitman/The MIT Press, 1991: 274-290
- [8] Ramakrishna Rau B. Iterative modulo scheduling: An algorithm for software pipelining loops//Proceedings of the ACM SIGMICRO Newsletter. San Jose, CA, USA, 1994: 63-74
- [9] Aditya Shail, Schlansker Michael S. ShiftQ: A buffered interconnect for custom loop accelerators//Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems. Atlanta, Georgia, USA, 2001: 158-167
- [10] Kathail Vinod, Aditya Shail, Schreiber Robert, Ramakrishna Rau B, Cronquist Darren C, Sivaraman Mukund. PICO: Automatically designing custom computers. IEEE Computer, 2002, 35(9): 39-47
- [11] Hadda Cherroun, Alain Darte, Paul Feautrier. Scheduling under resource constraints using dis-equations//Proceedings of the Conference on Design, Automation and Test in Europe. Munich, Germany, 2006: 1067-1072
- [12] Fan Kevin, Kudlur Manjunath, Park Hyunchul, Mahlke Scott A. Cost sensitive modulo scheduling in a loop accelerator synthesis system//Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture. Barcelona, Spain, 2005: 219-233
- [13] Liu Li, Li Wen-Long, Chen Yu, Li Sheng-Mei, Tang Zhi-Zhong. Hiding memory access latency in software pipelining. Journal of Software, 2005, 16(10): 1833-1841(in Chinese) (刘利, 李文龙, 陈彧, 李胜梅, 汤志忠. 软件流水中隐藏存储延迟的方法. 软件学报, 2005, 16(10): 1833-1841)



DONG Ya-Zhuo, born in 1979, Ph. D.. Her research interests include reconfigurable computing and high level synthesis.

DOU Yong, born in 1966, Ph. D., researcher, Ph. D.

Background

This paper is mainly supported by the National Natural Science Foundation of China (60633050). This project aims to deal with the urgent science computation requirements of our country, research the pivotal technologies of high-efficiency parallel computer architecture. Based on the analysis of some typical applications, a series of studies are carried through, including memory architecture, hardware acceleration and high level synthesis etc. More than 20 papers of the team was published in international conference and journals including FPGA2005, ASP-DAC 2007 and ASAP2007 etc.

The authors' work was used in high level synthesis. High Level Synthesis (HLS) tools provide a bridge between the algorithm written in a high level language (Matlab, C, C++, etc.) and a lower level Hardware Description Lan-

guage (HDL). They concentrates on one class of applications called window operations. This kind of applications is widely used in signal, image and video processing and requires much computation and data manipulation. Reconfigurable hardware boards provide a convenient and flexible solution to speed up these algorithms.

supervisor. His research interests include reconfigurable computing and high-efficiency computer architecture.

SONG Jian, born in 1982, M. S., associate engineer. His research interests include network security and information retrieval.

LIU Ming-Zheng, born in 1981, M. S., associate engineer. His research interests include image and signal processing.

This paper presents a pipelined template and instruction schedule method corresponding to execute all the loop nests in sequence without FPGA reconfiguration. Experiments show that the pipelined template can achieve a comparative execution cycles for a loop comparing with the special hardware and without the need of FPGA reconfiguration. Thus the pipelined template is better.

This paper presents a pipelined template and instruction schedule method corresponding to execute all the loop nests in sequence without FPGA reconfiguration. Experiments show that the pipelined template can achieve a comparative execution cycles for a loop comparing with the special hardware and without the need of FPGA reconfiguration. Thus the pipelined template is better.