

粗粒度可重构平台中循环自流水硬件实现

徐进辉^{1),2)} 杨梦梦²⁾ 窦 勇¹⁾ 周兴铭¹⁾

¹⁾(国防科学技术大学计算机学院 长沙 410073)

²⁾(解放军信息工程大学电子技术学院 郑州 450004)

摘 要 循环流水技术运用于粗粒度可重构体系结构可带来显著性能提升. 循环控制、流水线同步和存储器有效利用是其中的关键问题. 文中介绍了在粗粒度可重构体系结构 LEAP 上循环自主流水化的硬件实现. 该方法基于支持循环迭代自动调度的控制部件、数据驱动 ALU 和可配置静态交换路由. 利用动态调度循环中操作的优势, LEAP 可发掘更高的程序并行度; 分布式存储访问和高效数据重用则提高了带宽利用率. 实验结果表明, 相对于通用处理器, LEAP 有 13.08~535.65 倍的性能提升.

关键词 粗粒度可重构; 循环自主流水; 循环迭代控制; 数据驱动; 静态交换路由

中图法分类号 TP302 **DOI号**: 10.3724/SP.J.1016.2009.01080

The Implementation of Loop Self-Pipelining with Supports in Hardware for Coarse-Grained Reconfigurable Platform

XU Jin-Hui^{1),2)} YANG Meng-Meng²⁾ DOU Yong¹⁾ ZHOU Xing-Ming¹⁾

¹⁾(School of Computer, National University of Defence Technology, Changsha 410073)

²⁾(School of Electronics Technology, PLA Information Engineering University, Zhengzhou 450004)

Abstract Loop pipelining usually leads to significant performance improvements in coarse-grained reconfigurable architectures. Loop scheduling methods, synchronization of pipelines and measures efficiently utilizing the memory bandwidth are the key issues of loop pipelining techniques. This paper introduces the implementation of loop self-pipelining with supports in hardware on LEAP, which based on the hardware supporting automatically loop-iteration scheduling, data-driven ALU and configurable switch routers. Taking the advantages of dynamically scheduling iteration operations, LEAP exploits high degree of parallelisms. With the help of distributed memory access and efficiently data reusing of producer-consumer between computing elements, LEAP improves the bandwidth utilization. The experimental results show that the speedup over general processor can reach 13.08 to 535.22.

Keywords coarse-grained reconfigurable; loop self-pipelining; loop control; data-driven; static switch router

1 引 言

可重构处理器体系结构是一种理想的应用加速

平台. 由于硬件结构可以根据程序的数据流图重新组织, 可重构阵列已被证明其对于科学计算或多媒体应用具有良好的性能提升潜力. 与其它类型体系结构(通用处理器和定制硬件如 FPGA 或者 ASIC

等)一样,将程序转换为硬件或软件流水线,是一种提高应用性能的有效方法^[1].应用该项技术必须考虑在有限硬件资源和可能获得的性能提升之间进行折中,但是在通用计算平台和定制计算平台,流水技术尤其是循环流水技术均发挥了重要作用.

近来,有关可重构体系结构的很多研究都集中在循环流水线上^[1-7],这些工作可分为两类.一类为硬件流水线,包括在粗粒度可重构体系结构中将运算单元阵列组织成流水线、通过向量化技术将循环综合成硬件流水线以及将程序映射到数据驱动阵列上.另一类则类似于超长指令体系结构中的软流水.处理单元资源被视为一组功能单元,其中多个循环体通过编译器的准确调度并行执行.编译器的调度依赖于精确的资源使用模型,但是某些资源的使用难以确定,例如存储访问延迟与其所存储内容相关,无法精确预测.对于这两类方法,为了保证循环正确执行或者提高流水线吞吐量,循环调度的控制都是关键.

设计一个高效的支持循环自主流水化的可重构协处理器作为加速部件需要解决3个关键问题:

(1)循环控制问题.可重构协处理器针对循环进行加速,其本质在于充分发挥其计算资源丰富的优势,使得不同的循环迭代可并行执行.硬件实现的循环控制部件负责动态调度循环迭代在流水线上的执行,其实现方法决定了循环流水线的效率和并行度.循环控制的关键在于选择合适的时机来步进循环索引,产生信号驱动执行下一迭代.另一方面,循环迭代的输入数据通常依赖于循环索引值,循环控制部件必须解决输入数据地址产生和存储访问问题.同时,设计人员应避免循环控制部件成为进一步提高循环内并行度的瓶颈.

(2)流水线同步问题.如前所述,传统流水技术依赖于精确的资源调度,通过插入延迟单元(如FIFO或者寄存器)来达到流水线不同数据通路的同步,此即流水线平衡(balancing)技术.数据驱动可重构阵列通过数据流自然地达到同步^[5],获得更高的并行度.流水线的组织和连接方式、循环控制部件的实现方法均对同步问题产生影响,必须综合考虑.

(3)存储问题.在讨论可重构计算体系结构提供定制计算的优势的时候往往忽视该问题.现实体系结构框架下,对可重构计算加速器架构,有很大的约束限制.第一是存储带宽的约束限制,其次是内部局部存储器容量的限制.在存储带宽受限下有两种方法:①提高物理存储带宽;②通过发挥数据重用

性和局部性来充分利用带宽.而局部存储器容量小,但是相对速度比外部快,对计算能力很强的运算阵列来说能够充分利用局部存储器的数据是保证计算能力得到充分发挥的关键,比如通过运算单元之间直接互连,构成生产和消费者关系,消除了通用微处理器中通过寄存器文件转发数据的瓶颈.

本文着重介绍在粗粒度可重构体系结构LEAP中循环自主流水化技术的实现.LEAP支持循环迭代的硬件自动执行,该功能通过将循环索引步进功能转化为存储访问单元(mPE)实现.mPE产生一系列的存储地址,读取的数据流驱动数据通路(由cPE构成)的执行,计算结果最终写回存储器.cPE是流水化的数据驱动ALU,流水线的平衡由数据流动态地达成同步,其中还设置FIFO以获取更高流水线吞吐量.考虑到流水线中信号传输延迟,本文讨论了保证FIFO不溢出的最小深度需求.本文同时介绍了静态路由矩阵的实现,相对于传统的静态虫洞路由,它需要的硬件资源更少.通过分布式的mPE和多重配置,充分使输入数据相关,可以减少50%的局部存储器和30%局部存储器的带宽需求.在FPGA上的实验证明,循环自动流水化技术可以达到极高的流水线吞吐量,有效地利用了存储带宽,相对于通用处理器LEAP可达到13.08~535.65倍加速比.

2 LEAP 体系结构概述

LEAP体系结构旨在将多重循环程序直接通过粗粒度PE阵列形成硬件流水线执行.将循环程序分解为几个部分,图1给出了概念性的实现方法:循环index变量顺序变化,根据循环下标变量不断地取数据,数据被计算,计算的结果存储到指定的地址.图2是硬件实现的概念性结构,数据流图转化为一个包含4个部分的抽象硬件结构.其一为控制循环步进的有限状态机,其二为循环体的数据通路.另外两个部分为用于读/写操作的地址生成单元和数据FIFO.当循环控制有限状态机步进产生循环索引变量时,两个FIFO均可在将满的情况下向状态机发送信号暂停它,以保证不产生溢出.在循环索引变量使能后,读操作地址发送到存储体,返回值经FIFO流入数据通道,结果最终写回存储体.在硬件完成后,抽象硬件结构中的有限状态机的条件表达式、地址产生单元的参数以及数据通路连接均被固定,所有单元由一集中控制器控制.

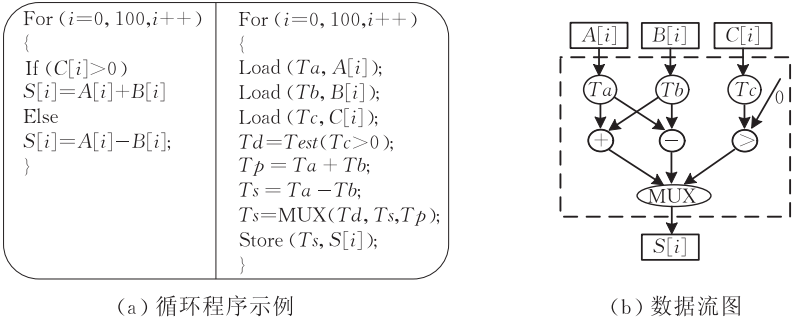


图 1

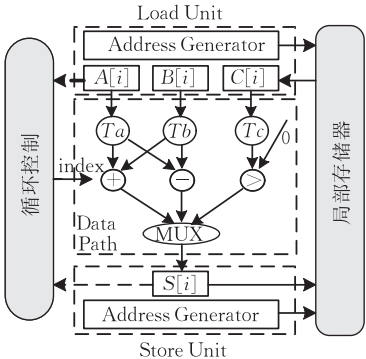


图 2 循环程序硬件实现概念性结构

在 LEAP 中,主要的设计思想是将上述概念性的结构模块化,消除集中控制瓶颈,并且通过循环控制器、地址产生单元和数据通路连接参数化,支持在不同配置下执行各种循环,如图 3 所示.其中 mPE 用于循环控制和数据访问,它结合这两种功能驱动循环的执行. mPE 内部有独立的循环状态机,可以连续产生局部的循环下标变量值,通过地址运算部件,就可以向存储器模块取数和存储结果. cPE 中没有循环状态机控制,由数据驱动,根据当前配置的操作

作功能,仅当所需要的数据就绪,从操作数缓存中取数据,进入运算流水线执行. 所有处理单元通过一个路由结点构成的网络连接在一起. 给定循环中,所有 mPE 和 cPE 间互连的拓扑结构不变,因此网络配置可存储于一个配置存储器中. 处理单元中同样拥有配置存储器, mPE 的用于存储循环控制表达式的参数和地址表达式的参数, cPE 的用于配置单元功能.

作为协处理器,接口控制器与主处理器连接完成配置信息的加载、数据的加载和结果返回以及对协处理器发送运行指令、对循环运行状态的监控. 每个 mPE、cPE 和网络矩阵都有多层配置,局部存储器被分成多个独立的模块,可以通过链表式 DMA 机制实现多个循环之间数据加载、功能配置和运算的重叠执行.

图 4 说明了循环例子程序按照数据流图 1(a)映射到 LEAP 结构上的执行过程. 存储数据的节点被映射到 mPE 上,算术运算和条件运算被映射到 cPE 上. mPE 从存储器中连续读取数据,例如, $A[0]A[1]\cdots A[100]$,形成取数据 A、B 和 C 流水

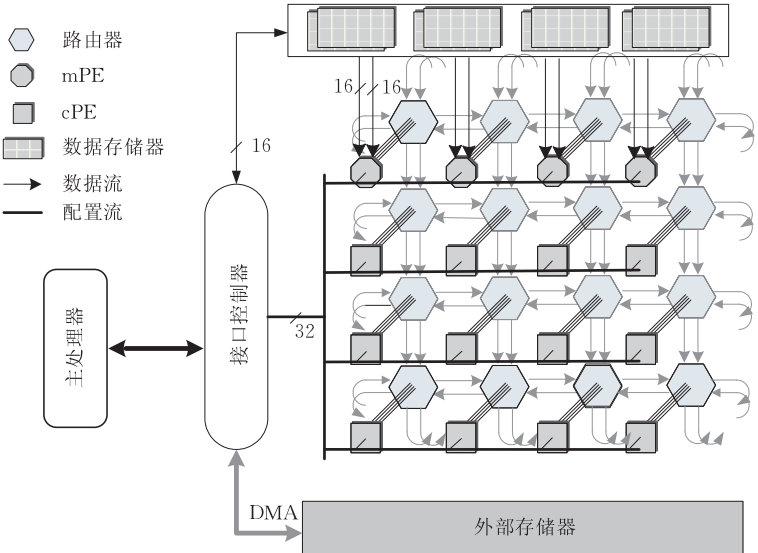


图 3 LEAP 体系结构(包含主控处理器、外部存储器、分布式存储模块、由 4×4 路由结点连接的 3×4 cPE 和 1×4 mPE)

线,该流水线在硬件上连接到 cPE 的 +、- 和比较操作运算流水线,这些流水线又将数据注入到选择操作 cPE,根据 Test 操作得到的结果,从 + 或 - 操作中选择计算结果,最后运算结果仍然按照流水线的方式,存入存储器中. 在这个执行过程中,首先取数据之间有重叠,之后取数据与运算形成重叠,运算

又与存储重叠,理想情况下,当流水线启动后,经过一段延迟,计算能力达到每拍一个循环迭代. 这要求极高的存储带宽或者数据重用的优化调度. 另一优点是在循环运行过程中,流水线的连接模式可配置,降低了互连网络的硬件复杂性.

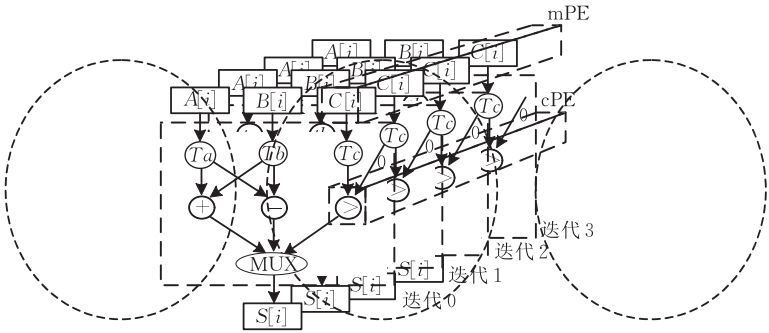


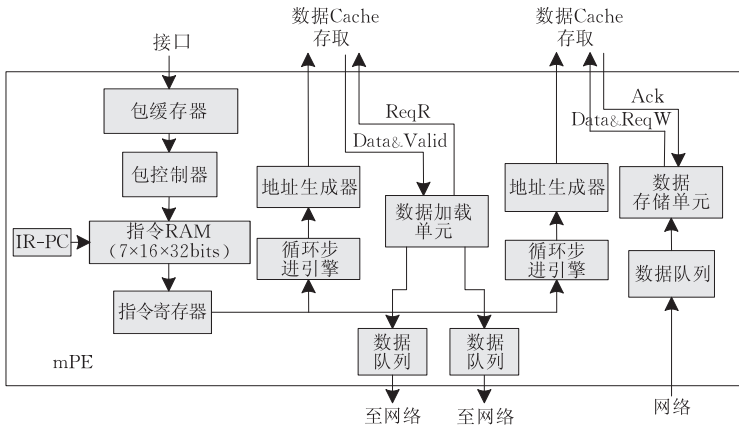
图 4 循环程序示例映射到 LEAP 上的执行过程

3 循环自主流水线的硬件实现

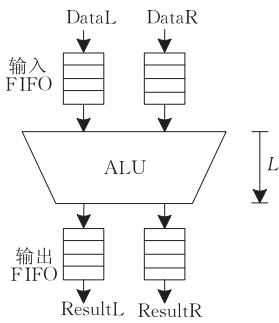
3.1 循环控制与存储访问

循环控制与存储访问是实现循环程序自动流水线执行的关键. 具体实现结构如图 5,该结构主要包括 3 个部分,第 1 个部分是 mPE 的配置部件,它主要提供循环步进的参数、地址产生的参数和读写数

据的参数. 第 2 部分是读数据部件,它包括产生循环下标变量值的循环步进状态机、地址生成器和数据缓存,两个数据输出队列用于将一个数据分发到两个目的 PE 处使用,这有利于充分节省存储带宽. 第 3 部分是写结果部件,它包括产生循环下标变量值的循环步进状态机、地址生成器、地址缓存和数据缓存. 其中读数据部件和写结果部件可以独立工作,分别对应于高级语言的 Load 和 Store 操作.



(a) mPE 结构(包含独立循环步进引擎、地址产生单元、数据队列和配置存储器)



(b) cPE 结构(包含数据驱动型 ALU, 输入输出 FIFO)

图 5

在这个过程中,循环状态机是控制循环流水线的关键.

- 1. 循环状态机是可配置的,通过给定循环的边界条件,可以从初始条件,顺序产生下标变量的值,直到达到循环结束条件.
- 2. 循环状态机产生的下标变量是循环中其它操作必须的信号,是全局信号.
- 3. 循环状态机是受反馈条件控制的,根据取数据、处理

数据和存储结果的不同情况,可能出现循环状态机停顿等待. 循环递进时,状态机要读取操作数 FIFO 的状态为不满且写结果地址 FIFO 也不满时,才能发出一个有效的下标变量值.

- 4. 循环下标值到达边界条件值,不代表循环的结束,循环结束的条件是所有循环下标值都发出,且所有的写结果都已经完成. 需要两个计算器,一个记录循环执行的次数,一个记录循环写出结果的个数,当二者相等且循环下标值发出完

成后,表示循环结束.

循环控制本质上是对数据流的控制,LSP 通过分布式 mPE 实现循环控制的非集中化. 循环体被拆分成独立的算术操作或者读写操作,分别映射到循环控制条件相同的 mPE 或 cPE 上. 每个 mPE 中循环控制引擎独立工作,只关注本地状态. 单元间的协同是通过 mPE 或 cPE 中的 FIFO 状态表示. 循环片间异步执行. 当循环完成,由存储操作映射其上的 mPE 向主控处理器报告结束状态. 在循环执行过程中,循环索引变量用于将存储访问转化为数据流,由数据流推动数据通路中运算过程执行.

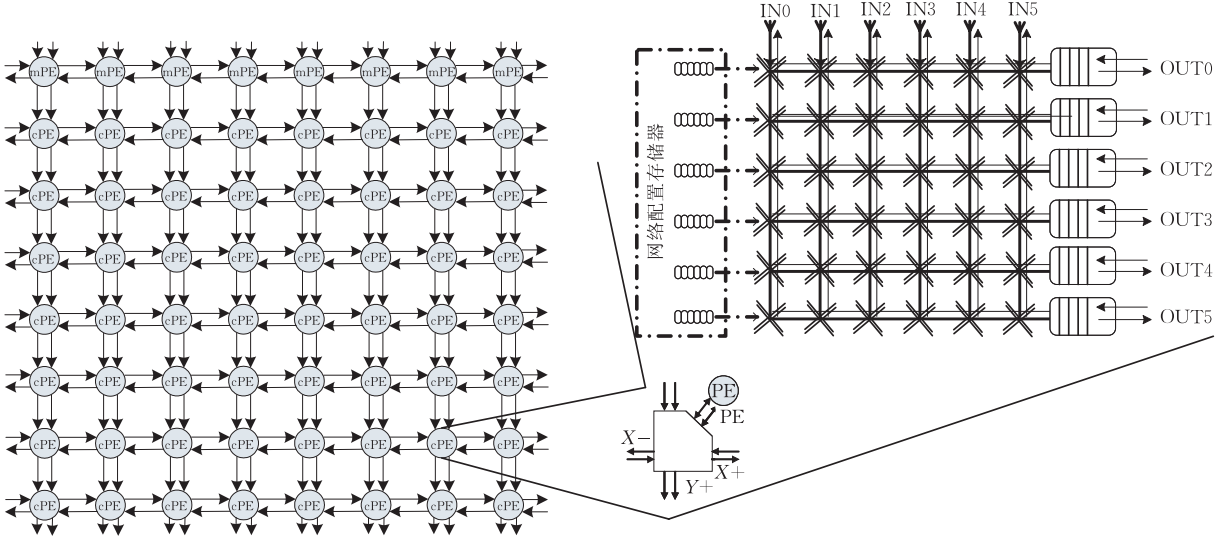


图 6 8×8 网络拓扑结构及路由结点结构(6 个输入输出端口,位宽 6bits 的配置存储器)

数据溢出会导致数据的丢失,因此插入 FIFO 需要考虑 FIFO 深度问题. 因为 ALU 是不具有存储能力的流水线,当它的下级通知它停止时,ALU 流水线上仍然有数据向下流动,比如,6 级浮点乘法器,当它的输出端接收到停止信号时,输入端控制逻辑停止向运算流水线送数据,但是此时在浮点乘法器中仍然有数据在运算,无法停止,如果此时输出端的 FIFO 已经满载,剩余的数据就会丢失.

在设计数据驱动型 ALU 时需要考虑的两个重要因素是输入 FIFO 和输出 FIFO 的深度问题. 输入 FIFO 的深度与阵列形成的数据延迟有关,而输出 FIFO 的深度与 ALU 的流水线有关. 先考虑输出 FIFO 的情况,输出 FIFO 的物理深度为 D_{out} ,输出 FIFO 达到一定数据量时需禁止从输入 FIFO 取数放进 ALU 流水线,若这时输出 FIFO 中的数据量为 L_{out} ,另外 ALU 流水线深度为 L ,则需满足

$$D_{out} \geq L_{out} + (L + 1),$$

其中, $(L + 1)$ 表示数据量到达 L_{out} 时输出 FIFO 还

3.2 数据驱动型 ALU 的实现

cPE 中主要组成部分是数据驱动型 ALU, ALU 前端是输入 FIFO,后端是输出 FIFO(图 6). 我们使用“栅栏”来实现数据在进入 ALU 之前的对齐和顺序化. 数据在进入 ALU 之前需进入输入 FIFO 排队等待,当 ALU 需要的所有数据都到来时,数据从 FIFO 弹出进入 ALU 进行操作. 一旦产生结果,便会把结果存入输出 FIFO 中,等待着送进数据网络. 这种插入 FIFO 的方式与插入寄存器的方式相比更具有灵活性.

需装载的数据量.

若输入 FIFO 的物理深度为 D_{in} ,输入 FIFO 达到一定数据量时需产生 ack 信号禁止从上级 PE 的输出 FIFO 取数放入数据网络,若这时输入 FIFO 中的数据量为 L_{in} ,由于输入 FIFO 的停止信号到达前一路由结点输出端口延迟为 1,故需满足

$$D_{in} \geq L_{in} + 1.$$

3.3 静态 Switch Matrix 互连

有两大类连接逻辑部件的方法,一类是静态连接,比如在 FPGA 芯片中,逻辑块 LB 之间的连接通过交叉矩阵连接,两条连接线的交叉点是否连接上受 Pass Transistor 的值控制,这些值存储在静态存储器中,因此可以对这种连接关系进行配置,它是一种物理的连接关系. 另一类,是动态连接,用于并行计算机之间或芯片内部多处理器之间,通过路由器连接处理部件的数据输入输出通道,路由器根据在数据链路上的报文目的地,通过路由和报文仲裁,动态选择报文在路由器中通过,虽然没有物理信号直

接相连,但是通过路由器间接相连,构成一种逻辑上的连接关系.静态可配置的连接方法逻辑设计最简单,固定连接关系,没有报文冲突分解的问题,但是在路由连通性、硬件速度和灵活性方面受到资源的制约.动态连接的路由器在灵活性和硬件速度方面有优势,但是由于需要报文缓存、路由仲裁等复杂的逻辑设计,硬件复杂度高.

循环程序执行具有流水线固定连接关系的特点为优化网络设计提供了新的机会.我们以静态互连方法为基础,提出了带缓存的可配置电路交换路由器.一个 4 路由器结构如图 3 所示,配置存储器的每位单元控制输入和输出通道的连接点,每条通道包括数据信号和数据有效信号以及一条前方状态反馈信号 ACK.在每条输出通道出口处增加了一个数据缓存,用寄存器将长线分段隔离,提高硬件流水线的频率,同时需要增加控制逻辑进行流量控制.

输入端口可以同任意一个输出端口相连,在循环执行之前配置存储器的值被加载到交叉点,确定了所有的连接路线,相当于电路交换方式(circuit switching),配置完成之后链路之间没有冲突.在一条链路内部有一系列缓冲寄存器或 FIFO 队列连接构成流水线,流量控制仅局限在链路本身的停顿与再启动的控制,不需要全局报文请求的仲裁逻辑.流水线的频率仅受限于路由器内部连线长度和路由器之间的连线长度.

在 LEAP 体系结构中,实现了 6×6 的上述路由器,以此为基础,构成 8×8 的处理器阵列.每个路由器有 2 条双向通道连接 PE;在垂直的 Y 方向,一个路由器的两条输出通道顺序连接下一个路由器的两条输入通道,最后一个路由器再环回第一个路由器,构成单向双通道环路是为了顶端的 mPE 能够经过最短的路径将存储器数据输出到其它 cPE 上,经过 cPE 的计算,数据从上向下流动,最后结果经过尽可能短的路径返回 mPE;在 X 方向,构成正负两个方向的环路.

3.4 多重配置

多重配置通过分布于每个 PE 中的配置存储器实现.每个 PE 有一套配置存储器和配置控制部件,用于配置信息的接受和加载.mPE 配置存储器的位宽在逻辑上有 224 位,物理上由 7 个 32 位宽存储体组成,主要用于存储 3 重循环的控制参数、1 个读数据和—个写结果地址表达式参数.cPE 的配置存储器位宽逻辑上有 32 位,用于指定 cPE 的操作类型和运算结果的传递目标.配置存储器的深度有 16 层,

有一个配置存储器指针标识当前在用的配置层号.

配置信息从协处理器接口输入,采用广播方式加载到每个 PE 的配置存储器中.配置流通过 LEAP 内部专用的配置总线进入 PE,PE 的配置控制部件根据配置信息的目标地址有选择地读取配置信息.这样可以支持多个循环同时在 PE 阵列上执行,也可以支持多个循环以不同层配置的方式顺序在阵列上执行.同时可以实现当一些 PE 接收配置信息时,另一些 PE 可以同时计算.

多层配置之间的切换有两种方式,一种是被动配置,由主机通过接口控制器发出执行某层配置的命令,此时,当前配置指针寄存器指向对应的层号.另一种是自动配置,通过指定下一待执行配置的层号,在当前循环执行结束后,配置控制逻辑自动更新配置指针寄存器,新的配置立即生效,开始新的循环执行,无需等待外部触发,减小多重配置之间的切换延迟.

4 实验结果及分析

4.1 实验环境及综合结果

可重构计算系统通常对其面向的应用领域进行优化,系统结构上存在很大差异,迄今为止尚未形成公认的系统性能评测标准,因此常用性能评价方法是比较可重构系统与通用系统上目标应用的执行效率(时钟周期、指令并行度和存储访问效率等).我们在 Gidel 公司的 ProcSuperStar 80-3 开发平台上进行原型设计,该开发平台提供 3 块 Altera 的 Stratix FPGA 芯片和 64MB 片外存储器,存储器控制器 IP 核 PROCMultiPort 完成 FPGA 芯片对片外存储器的操作.主机采用单路 P4 2.80GHz 处理器,1.50GB 内存,主机与开发板之间通过 PCI 总线进行通信.该主机也用作典型的 CISC 通用处理平台与 LEAP 进行性能对比,另采用 SimpleScalar 3.0d 模拟 RISC 通用处理平台.开发平台提供软件工具 ProcWizard 自动整合软件和硬件,产生 HDL 代码和 C++ 应用驱动代码,HDL 综合工具采用了 Quartus II 3.0,软件开发及编译采用 Microsoft Visual C++ 6.0.

LEAP 阵列在 FPGA 上的综合结果见表 1.表中分别给出了 mPE、cPE 和 6 输入 6 输出路由结点单独综合结果以及组合成 7×7 和 3×3 阵列的综合结果.后 4 列中数据分别为综合后各部件占用的逻辑单元、存储器、DSP 数目以及频率.逻辑部件的连

接对于流水线的组织和阵列频率的提高有很大影响,表 1 中对两种互连结构(见 3.2 节)的路由结点分别在 FPGA 上实现,由结果可以看出静态互连逻

辑简单,不占用存储器,且静态互连延迟短,有利于阵列结构的频率提升. LEAP 原型阵列中均采用静态互连结构.

表 1 FPGA 综合结果

| 部件 | | 逻辑单元 | 存储/bits | DSP | 频率/MHz |
|--------|----|----------------------|------------------------|------------------|--------|
| mPE | | 778 | 5536 | 12 | 133.46 |
| cPE | | 758 | 3104 | 2 | 106.85 |
| 路由结点 | 静态 | 454 | 0 | 0 | 214.87 |
| | 动态 | 1205 | 2592 | 0 | 195.24 |
| 7×7 阵列 | | 73877/79040 (93%) | 365680/7427520 (4%) | 168/176 (95%) | 42.50 |
| 3×3 阵列 | | 47155/79040 (59%) | 592576/7427520 (7%) | 160/176 (90%) | 47.21 |

4.2 性能测试及分析

中值滤波等 7 个典型应用(见表 2)用于 LEAP 原型和通用处理平台的对比测试. 为了达到最高性能,前 4 个测试程序采用 7×7 原型阵列,后 3 个测试程序采用 3×3 原型阵列. LEAP 原型支持 16 位定点运算、整数运算、逻辑运算、整数测试、分支预测和特殊运算等 6 类 23 种操作. 原型的存储器分为外

部存储器和局部存储器,外部存储器由主机和 LEAP 阵列共享,用于缓存大量数据;局部存储器通过 DMA 与外部存储器交换数据,组织形式为分布式多模块结构,数据一致性由软件控制. 鉴于在 mPE 和多个局部存储块之间实现交叉互连过于复杂,LEAP 原型中每个局部存储块由两个 mPE 共享,而每个 mPE 只能访问其中两个特定存储块.

表 2 测试程序

| 应用 | 原型 | 处理规模 | 说明 |
|--------|---------|---------------------------------|----------------------------|
| Median | 7×7 阵列 | 320×240,480×360 图像 | 中值滤波 |
| Sobel | | 320×240,480×360 图像 | Sobel 边缘检测 |
| FFT | | 512 点,1024 点 | 快速傅立叶变换 |
| Quant | | 一帧(15840 个宏块,每个宏块包含 6 个 8×8 的块) | 量化算法,取自 Mediabench 中 MPEG2 |
| MM | 3×10 阵列 | 64×64,128×128 矩阵 | 两种不同规模的矩阵乘 |
| Fdct | | 一帧 | 正向余弦变换,其余同 Quant |
| Idct | | 一帧 | 反向余弦变换,其余同 Quant |

典型应用程序在 3 个平台上的性能测试结果见表 3. 表格中后 3 列中数据为应用程序在 3 种不同平台上执行所需时钟周期数. 表 3 结果显示,对于所有测试的典型应用,LEAP 原型均有大幅度的性能加速,相对于 PC,最低可加速 13.08,最高可达到 513.08. 可观察到其中中值滤波和 FFT 的加速比尤

其突出,前者由于核心循环包含指令少,且嵌套层次过深,后者则由于数据访问模式复杂,屏蔽了编译优化效果,不能有效开发发掘循环中的数据并行性. SimpleScalar 上模拟结果大体与 PC 中结果相符,由于 SimpleScalar 采用的是类 MIPS 的精简指令集结构,因此执行的指令数目和周期数要高于 PC 平台.

表 3 典型应用在 3 种平台上的性能测试结果

单位:时钟周期

| 测试程序 | 数据规模 | LEAP /K | SS /M | PC /M | 加速比 LEAP/SS | 加速比 LEAP/PC |
|--------|---------|---------|---------|--------|-------------|-------------|
| Median | 320×240 | 220.0 | 41.86 | 112.88 | 190.27 | 513.08 |
| | 480×360 | 478.8 | 94.67 | 256.46 | 197.72 | 535.65 |
| Sobel | 320×240 | 217.0 | 9.94 | 2.84 | 20.76 | 13.08 |
| | 480×360 | 474.2 | 22.52 | 6.28 | 47.49 | 13.24 |
| FFT | 512 | 6.7 | 3.69 | 2.31 | 550.75 | 343.49 |
| | 1024 | 12.8 | 7.35 | 5.17 | 574.22 | 403.69 |
| MM | 64×64 | 79.1 | 4.77 | 1.38 | 60.30 | 17.42 |
| | 128×128 | 580.3 | 42.82 | 10.53 | 73.79 | 18.15 |
| Fdct | 一帧 | 2839.0 | 2369.38 | 173.81 | 834.58 | 61.23 |
| Idct | | 2839.0 | 2615.09 | 174.10 | 921.13 | 61.32 |
| Quant | | 2131.6 | 373.40 | 52.79 | 175.17 | 24.77 |

表 4 给出了在 LEAP 原型和 SimpleScalar 模拟器中的一些重要统计结果. 其中在 LEAP 原型上 $\#OP$ 为并行操作数目, $\#AP$ 为计算单元活跃比, 即实际计算时间在程序总执行时间中所占的比例, $\#VBW$ 为实际利用的局部存储器有效带宽, $\#MAN$

为访存操作数目;而在 SimpleScalar 中, $\#IPC$ 为每拍执行指令数, $\#AP$ 为 IPC 与指令发射带宽之比 (设置为 4 发射), $\#INSN$ 为总执行指令数目, 而 $\#MAN$ 为访存指令数目.

| 表 4 程序执行统计结果 | | | | | | 单位:时钟周期 | | | |
|--------------|---------|--------|-----------|-----------|----------|---------|-----------|------------|-----------|
| 测试程序 | 数据规模 | LEAP | | | | SS | | | |
| | | $\#OP$ | $\#AP/\%$ | $\#VBW/b$ | $\#MAN$ | $\#IPC$ | $\#AP/\%$ | $\#INSN/M$ | $\#MAN/M$ |
| Median | 320×240 | 30 | 42.30 | 7×16 | 597856 | 1.88 | 47.25 | 90.01 | 3.86 |
| | 480×360 | | 41.20 | | 1200016 | 1.88 | 47.25 | 203.42 | 87.32 |
| Sobel | 320×240 | 16 | 41.47 | 7×16 | 597856 | 1.85 | 46.25 | 18.44 | 7.55 |
| | 480×360 | | 40.62 | | 1200016 | 1.84 | 46.00 | 41.63 | 17.05 |
| FFT | 512 | 10 | 40.26 | 10×16 | 25600 | 1.36 | 34.00 | 0.62 | 0.21 |
| | 1024 | | 43.49 | | 51200 | 1.38 | 34.50 | 1.53 | 0.42 |
| MM | 64×64 | 16 | 41.62 | 17×16 | 278528 | 1.88 | 47.25 | 8.95 | 3.51 |
| | 128×128 | | 82.60 | 17×16 | 4456448 | 1.50 | 37.50 | 70.43 | 27.66 |
| Fdct | 一帧 | 18 | 39.17 | 19×16 | 27394048 | 1.68 | 43.50 | 4148.61 | 1292.50 |
| Idct | | 18 | 39.17 | 19×16 | 27394048 | 1.56 | 43.75 | 4230.05 | 1289.46 |
| Quant | | 17 | 32.58 | 4×16 | 2395008 | 1.05 | 42.00 | 393.12 | 157.34 |

分析表 4 结果可知, LEAP 上的应用程序性能提升主要来自于两个方面. 首先, 表 4 结果表明 LEAP 可以高效地发掘程序中的并行度. 中值滤波峰值并行操作数目高达 30, FFT 最低为 10 个并行操作. 峰值并行操作数目与 $\#AP$ 相乘可获得应用程序平均并行操作数目, 其中以 128×128 矩阵乘最高为 13.22, 512 点 FFT 最低, 仅为 4.03. 前者主要得益于其计算访存比极高, 导致计算密集, 从而 $\#AP$ 高达 82.60%. 与之相反, SimpleScalar 受限于指令发射宽度以及访存延迟等, IPC 最高为 1.88, Quant 的 IPC 最低仅为 1.05, 其中条件分支指令过多导致指令并行度下降是主要原因, 而且注意 SimpleScalar 统计 IPC 时包含了访存指令. LEAP 中分布式的独立循环控制和存储访问, 通过数据流驱动由数据驱动计算单元构成的流水线执行计算, 在消除循环的控制相关带来的额外开销的同时, 也变相地提高了指令发射宽度和循环内指令并行度. 同时循环自主流水技术利用数据流驱动阵列的动态调度功能, 更有效地挖掘出循环迭代间存在的并行度.

其次, LEAP 较好地解决了存储墙的问题. 从程序结构角度分析可发现这些典型应用均具有较好的数据并行性, SimpleScalar 中访存指令占总的执行指令数目比例很大, 其中以 Sobel 边缘检测程序为最高, 达 41%, 存储延迟是指令级并行度提高的主要瓶颈, 此即存储墙问题. LEAP 中局部存储器采用分布式多模块结构, 每个模块均提供 2 个读端口, 1 个写端口, 因此在 3×10 LEAP 原型中, 局部存储器物理带宽为 480 位. 表 3 第 5 列中数据 (有效带宽)

表明 LEAP 中物理带宽利用率较高, 这得益于 mPE 采用分布式的独立存储访问控制. 另一方面, 对比 LEAP 和 SimpleScalar 中存储访问数目可发现, 前者仅为后者的 10%~30% 左右, 一是得益于程序映射到 LEAP 上是我们利用输入数据相关进行的存储优化, 二是流水线连接构成的生产者消费者之间直接数据传递, 消除了中间数据的存储带宽需求.

5 相关工作

当前多个研究项目致力于在可重构体系结构应用流水技术以获得更高性能. 软流水技术对于提高循环的吞吐量有重要作用. Garp^[6] 中采用 IMS (Raus Iterative Modulo Scheduling) 调度算法^[8] 流水化最内层循环. XPP^[2] 中则采用流水向量化方法^[9], 但该方法限制循环体不能包含迭代间真相关, 且循环体内不能存在条件分支. Mei^[1] 等提出了一种无需显示的 epilogue 和 prologue 的软流水方法, 且可在需要硬件虚拟化时产生多重配置.

前文所述几种流水技术静态构造流水线, 无法利用数据流驱动阵列中的动态调度优势. 并且, 为了最大化流水线吞吐量, 必须采用流水平衡技术, 这样循环的最终延迟则必定由其中最长一条数据通路决定.

与本文关系更密切的是动态循环流水^[5]. 与本文工作相同, 动态循环流水技术也应用于数据流驱动粗粒度可重构阵列, 通过数据流来自然达到不同数据通路的同步, 并且目的都是实现循环的自动执行. 不同点在于本文介绍的方法采用专用循环控制

部件(mPE)实现分布式独立循环控制和存储访问,而动态循环流水技术则是复制循环控制硬件结构(主要提供计数功能),该方法无法支持循环直接映射,且增加了编译复杂度.本文的方法进一步考虑了存储问题、分布式的局部存储体和存储访问方式,结合数据驱动计算单元构成的流水线,提高了物理存储带宽及其利用率,而且减少了带宽需求.

6 总 结

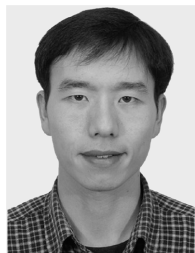
本文介绍了在粗粒度可重构体系结构 LEAP 上的循环自主流水化的硬件实现方法,该方法基于专用循环控制引擎控制循环迭代的执行.循环流水线由数据驱动 ALU 连接而成,不同数据通路可由数据流自然地达成同步.利用数据流驱动阵列动态调度操作的优势,LEAP 可发掘出更高的并行度.其中采用的静态路由结点逻辑简单,延迟更短.实验结果表明,支持循环自主流水化的 LEAP 体系结构执行效率高,性能潜力大,且具有更高的物理存储带宽及利用率.

参 考 文 献

- [1] Mei B, Vernalde S, Verkest D et al. Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling//Proceedings of the Design, Automation and Test in Europe Conference and Exhibition. Munich, Germany, 2003: 10296-10301
- [2] Baumgarte V, Ehlers G, May F et al. PACT XPP—A self-

reconfigurable data processing architecture. The Journal of Supercomputing, 2003, 26(2): 167-184

- [3] Lee J, Choi K, Dutt N. An algorithm for mapping loops onto coarse-grained reconfigurable architectures//Proceedings of the Languages, Compilers, and Tools for Embedded Systems (LCTES'03). San Diego, CA, 2003: 183-188
- [4] Barat F, Jayapala M, Beeck P. Software pipelining for coarse-grained reconfigurable instruction set processors//Proceedings of the 15th International Conference on VLSI Design (VLSID'02). Bangalore, India, 2002: 338-344
- [5] Cardoso J M P. Dynamic loop pipelining in data-driven architectures//Proceedings of the 2nd Conference on Computing Frontiers(CF'05). Ischia, Italy, 2005: 106-115
- [6] Callahan T, Wawrzynek J. Adapting software pipelining for reconfigurable computing//Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems(CASES00). San Jose, CA, USA, 2000: 57-64
- [7] Hannig F, Dutta H, Teich J. Regular mapping for coarse-grained reconfigurable architectures//Proceedings of the 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2004). Montreal, Quebec, Canada, 2004, V: 57-60
- [8] Rau B. Iterative module scheduling: An algorithm for software pipelining loops//Proceedings of the ACM 27th Annual International Symposium on Microarchitecture (MICRO-27). New York, 1994: 63-74
- [9] Weinhardt M, Luk W. Pipeline vectorization. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2001, 20(2): 234-248
- [10] Budiu M. Spatial computation [Ph. D. dissertation]. CMU CS Technical Report CMU-CS-03-217, December 2003



XU Jin-Hui, born in 1978, Ph. D. candidate. His research interests include reconfigurable computing, high performance computing.

YANG Meng-Meng, born in 1980, Ph. D. candidate. Her research interests include trusted computing, reconfigurable computing.

Background

This paper focuses on the loop pipelining in reconfigurable computing platform. Most recent researches on the reconfigurable architecture uses loop pipelining techniques to increase the performance of computing intense application. These works can be classified to two types. One uses the reconfigurable units as the stages of pipeline. And another one relates to the software pipeline in VLIW. But all of them need the support of compiler to accurately schedule the instructions to execute concurrently. Scheduling depends on the accurate model of system resource, i. e. the latency of memory access related to the its content can't be exactly predicate.

rable computing.

DOU Yong, born in 1966, professor, Ph. D. supervisor. His research interests include high performance computing, parallel computer architecture.

ZHOU Xing-Ming, born in 1938, professor, Ph. D. supervisor, member of Chinese Academy of Sciences. His research interests include parallel computer architecture, mobile computing.

The work of this paper belongs to the project "Loop Engine Array Processor", which attached a coarse-grained reconfigurable array to general purpose processor to accelerate the execution of nested loop. This paper designs and implements the loop control engine to directly support nested loop, also imposes the loop self pipelining technique in the array. So the mapping of nested loop program to array is straightforward, and the scheduling and synchronization of loop iterations is automatically reached. The mechanism in this paper can expose more fine- and middle-grained parallelism, namely instruction level parallelism and iteration parallelism.