

一种流体系结构中软硬结合的异构核协同方法

任 巨 何 义 荀长庆 杨乾明 文 梅 伍 楠 张春元

(国防科学技术大学计算机学院 长沙 410073)

摘 要 在流体系结构中,标量核同流处理核是异构核,它们之间的协同是流处理器能够正确、高效运转的基础.文中针对异构核间所采用的软件协同方法性能低的问题,提出了一种软件和硬件相结合的异构核协同方法,并基于 MASA-I 流处理 SOPC 系统进行了实现.使用媒体和数字信号处理领域核心算法进行测试的结果表明,与软件协同方法相比,使用文中方法的协同性能有 2 个量级的提升,程序整体性能提高一倍.

关键词 异构核;核间协同;软硬结合;流处理器;体系结构

中图法分类号 TP302 **DOI 号**: 10.3724/SP.J.1016.2008.02038

A Hardware/Software Method for Heterogeneous Cores Cooperating on Stream Architecture

REN Ju HE Yi XUN Chang-Qing YANG Qian-Ming WEN Mei WU Nan ZHANG Chun-Yuan

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

Abstract In stream architecture, scalar core and stream core are heterogeneous cores, and achieving the cooperation between them is a significant basis on which the stream architecture can run correctly and efficiently. Aiming at solving the problem of low performance caused by software method, this paper presents a cooperative method in which software modules work with hardware logic, and implements it on the MASA-I stream processing SOPC system. Kernel algorithms of media processing and signal processing are used to evaluate this method. The results show that it is more advanced than the software method in that hardware/software method can achieve promotion of cooperation performance by two orders of magnitude and double the kernel performance.

Keywords heterogeneous cores; cores cooperation; hardware/software; stream processor; computer architecture

1 引 言

VLSI 技术的高速发展使得在处理器芯片上集成多个处理核成为可能,而为了适应不同工作负载

对处理器结构的不同需求,异构多核体系结构成为研究的热点.流体系结构^[1]是一种异构多核体系结构,如 Imagine^[2]、Merrimac^[3]、RAW^[4]、Trips^[5]、FT64^[6]和 MASA^[7]等,本文所研究的 Imagine、FT64 和 MASA 采用的流体系结构,其结构如图 1 所示.

收稿日期:2008-05-31;最终修改稿收到日期:2008-09-11.本课题得到国家自然科学基金(60673148,60703073)、国家教育部博士点基金(20069998025)与国家“八六三”高技术研究发展计划项目基金(2007AA01Z286)资助.任 巨,男,1980 年生,博士研究生,主要研究方向为高性能微处理器体系结构、并行处理. E-mail: renju@nudt.edu.cn; se7en.rj@gmail.com.何 义,男,1982 年生,博士研究生,主要研究方向为高性能微处理器体系结构.荀长庆,男,1984 年生,硕士研究生,主要研究方向为高性能微处理器体系结构、并行处理.杨乾明,男,1984 年生,硕士研究生,主要研究方向为高性能微处理器体系结构.文 梅,女,1975 年生,博士,副教授,主要研究方向为高性能微处理器体系结构、并行计算.伍 楠,男,1981 年生,博士研究生,主要研究方向为高性能微处理器体系结构、流计算、并行编译.张春元,男,1964 年生,博士,教授,博士生导师,主要研究领域为高性能微处理器体系结构、并行处理技术、嵌入式系统.

它具有两种主要的处理核心:由许多简单快速的计算单元阵列和大量的本地寄存器文件构成的流处理核;由通用处理器核构成的标量核. 标量核和流处理核的结构迥异,功能不同,是异构核^[8]. 这种结构特别适合处理符合流计算模型^[1]的问题,包括 H. 264 编码、流体力学计算、快速傅立叶变换、卷积等媒体处理、科学计算、信号处理领域的应用及核心算法. 流计算模型将应用分解成流级和核心级两部分,由标量核执行流级程序,流处理核执行核心程序,二者协同完成整个应用,协同任务包括核心程序(kernel)的加载、输入流数据的加载、核心程序的执行、输出流数据的存储以及标量数据的传递等. 这些任务的执行机制和性能要求各不相同,导致异构核间的协同比较复杂,而协同的效率直接影响整个处理器的性能,因此异构核协同方法是确保流处理器正确高效运行的关键.

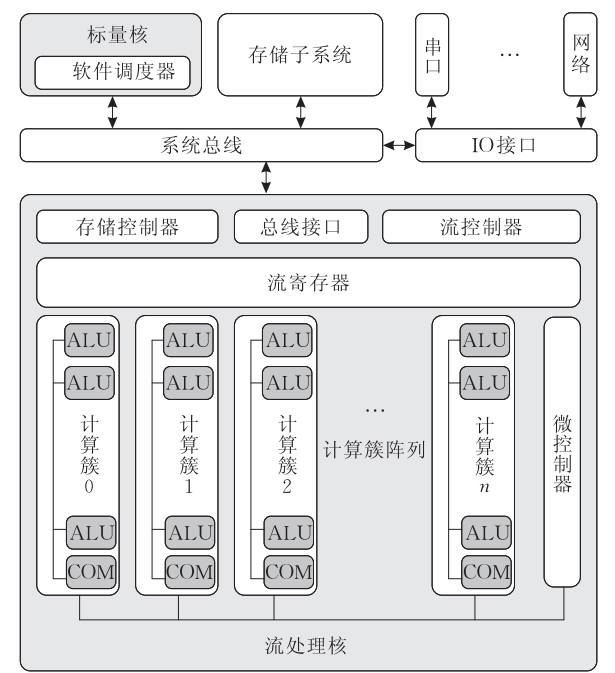


图 1 流体系结构内部结构图

Imagine、FT64 和 MASA 流体系结构采用软件方法实现异构核之间的协同:当执行流应用时,通过调用在标量核上运行的软件调度器完成标量核同流处理核之间的协同. 该方法的缺点是:首先,因为软件调度器在标量核上运行,所以在实际运行时协同软件与流处理核要通过总线不断交互,占用了标量核和总线资源. 其次,软件运行的速度无法匹配高速的硬件运行,低速的异构核协同导致整个处理器性能的下降. 例如,在标量核向流处理核发送流指令时,软件调度器通过总线查询流处理核的当前状态,根据此状态判断是否能够发送. 如果可以则发送流

指令给流处理核,否则不断查询流处理核状态,直到能够发送. 软件不断查询的过程耗费大量时间而且影响执行性能,又长期占用标量核和总线资源.

针对上述问题,本文提出了一种新的软硬结合的异构核协同方法:不再单纯使用标量核上的协同软件,而是在硬件结构、软件模块、指令集和编译等多个层次上进行设计和扩展,共同完成异构核间的协同. 本文基于 MASA-I 流处理 SOPC 系统^[9]实现了该方法,并使用实际的应用核心程序进行性能评测和对比分析,结果显示本文提出的异构核协同方法使协同性能有 2 个量级的提升,程序整体性能提高一倍.

本文第 2 节介绍异构核协同的相关研究;第 3 节阐述异构核的协同任务;第 4 节详述软硬结合的异构核协同方法的设计和实现;第 5 节对软硬结合的异构核协同方法进行性能评测和对比分析;最后总结全文.

2 相关研究

与本文研究相关的异构核间的协同工作方式主要有下面两种^[10]:

(1) 功能卸载模式. 这种模式在主处理核上执行主程序,加速处理核作为关键性能函数的加速器. 主程序逻辑不改变,需要加速执行的关键函数针对加速部件进行优化和重新编译. Imagine 和 FT64 的异构核协同都采用这种模式,这是因为流处理核的计算功能强大而控制功能很弱,因此主程序需要在具有强控制功能的标量核运行,并且功能卸载模式中,主程序执行逻辑不变,只需要针对流处理核加速,这也易于程序员的开发^[11]. 此外,Imagine 设计了运行时调度程序^[12],并且通过编译产生了大量的辅助信息支持该程序的动态调度.

(2) 计算加速模式. 这种模式以加速核为中心处理计算密集的应用,经过并行切分的主程序在加速核上并行运行,而主处理核主要为加速核提供控制和系统服务工具. 这种模式一般用于数学库函数加速. Score^[13]采用计算加速模式处理媒体应用,它的标量核用于计算页调度和支持操作系统.

CELL 在一个芯片上集成了 PowerPC 主处理器核心(PPE)和协处理器核心(SPE),既能以 PPE 为中心支持功能卸载模式,也能以 SPE 为中心支持计算加速模式.

本文提出的异构核协同方法采用了功能卸载模式. 与 Imagine 和 FT64 不同的是,我们不只使用软

件方法,而是通过软件模块和硬件逻辑单元结合的方法完成协同任务.

3 协同方法和任务

异构核协同方法的研究要以异构核间协同任务的要求为依据,根据任务的不同特征和需求设计相应的协同方法.流体系结构中异构核协同工作包括核心程序的加载、流数据的加载和保存、核心程序的启动以及标量数据的传递等复杂任务,它们可以通过下面 4 个基本任务的组合来完成.

(1)标量核发送流指令.获取流指令执行的状态信息,并根据状态信息选择当前流指令的发送时机;

(2)动态调度并发射流指令.根据标量核发送过来的指令相关性信息和流处理核各个功能部件的当前状态信息来动态调度流指令,并将流指令发射到流处理核;

(3)流数据在标量核、流处理核及存储外设间的传输;

(4)标量数据在标量核同流处理核间的传输.

需要指出的是,流数据的传输过程中可能发生双缓冲情况:流处理核使用流寄存器文件(SRF)来存放流数据,因此硬件级的流长不能超过编译器为该条流分配的 SRF 空间的容量,而应用级的流长是没有限制的.对于流长超过 SRF 容量的情况,采用双缓冲机制:将一条长流分解成多个长度相同的短流,交替使用 SRF 中的两部分空间来传输这些短流,双缓冲的详细情况参见文献[8].

4 软硬结合的异构核协同方法

根据异构核协同的 4 个基本任务,本文提出了一种软硬结合的异构核协同方法:将最复杂的和对灵活性要求高的任务交给软件模块,而对性能需求高的任务交给硬件逻辑;通过硬件、软件和编译等多个层次的共同协作完成异构核间的协同任务.图 2 是软硬结合的异构核协同框架图.

如图 2 所示,在标量核中设计有一个软件协同模块,在标量核和流处理核之间添加一个异构核协同单元,二者构成了异构核协同的核心.在指令的发送和发射中,软件协同模块负责接收标量核上经编译生成的流操作,生成流指令二进制码,发送流指令给异构核协同单元;异构核协同单元负责根据流处理核各功能部件的状态信息动态调度并发射流指令

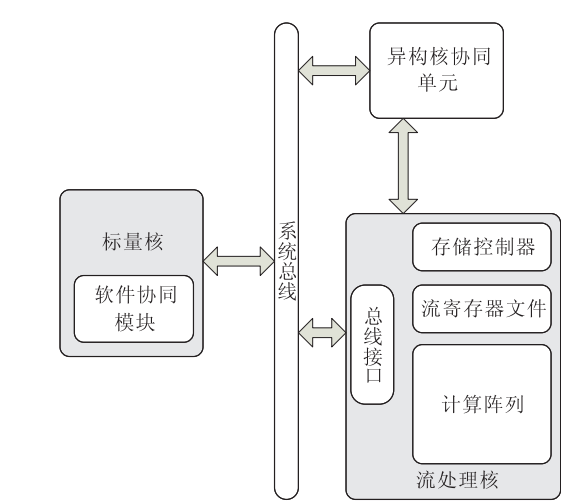


图 2 软硬结合的异构核协同框架

到流处理核.在数据的传输过程中,异构核协同单元负责发射流传输指令并控制流处理核的存储控制器,软件模块负责初始化外设接口寄存器.这种结构的优势在于:(1)使用硬件模块完成状态查询和流指令的动态调度,加快了流指令的发送和发射速度;(2)使用硬件模块直接控制数据的传输,提高了数据传输速度;(3)使用软件模块完成与各种不同外设的交互,具有灵活性.下面详细介绍本方法的软硬件组成和协同任务的实现.

4.1 异构核协同单元

软硬结合的协同方法中,用于异构核协同的硬件模块称为异构核协同单元,其内部结构如图 3 所示,主要分成 3 个部分:指令发送单元、协同接口、指令调度单元.异构核协同单元主要负责标量核同流处理核之间的协同,包括流指令的发送、流指令的动态调度和发射、标量数据和流数据的传输等.

4.1.1 指令发送单元

包含指令 FIFO 队列、指令缓冲和发射控制逻辑.指令发送单元将从总线传输过来的流指令缓存在指令 FIFO 队列中;发送控制逻辑根据来自协同接口的流处理核状态信号,控制指令 FIFO 队列向协同接口发送流指令;当出现双缓冲情况时,发送控制逻辑将双缓冲相关的流指令送入指令缓冲,并循环发送这些流指令,直到双缓冲结束,双缓冲的完成还需要编译和指令集的支持,见 4.2 节;发送控制逻辑还通过总线向标量核发送流处理核状态.

4.1.2 协同接口

其内部是一组接口寄存器,是指令发送单元和指令调度单元的中间模块,负责保存标量核和流处理核之间传输的指令、状态和数据.标量核和流处理核的交互是通过对该寄存器组的读写完成的,包括

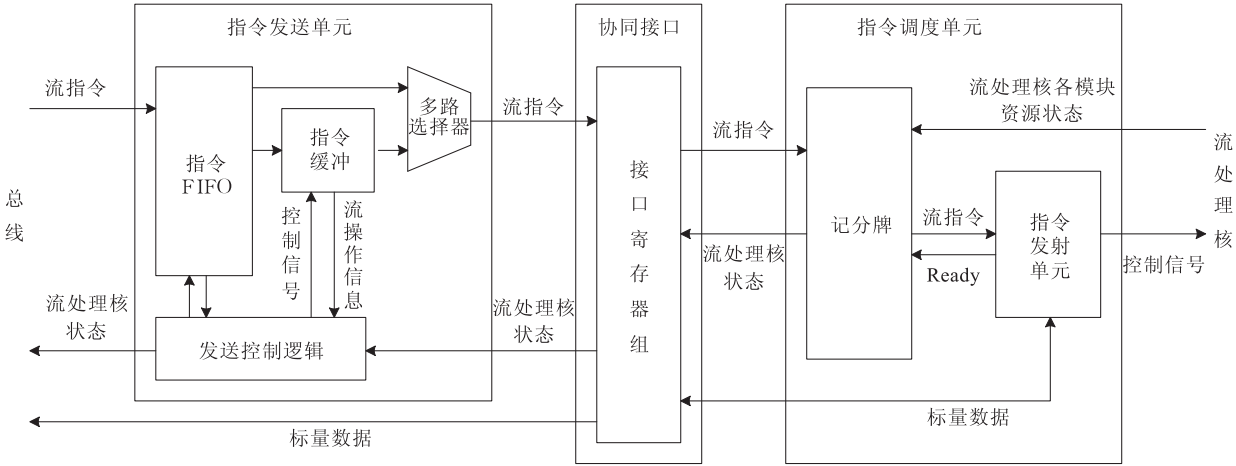


图 3 异构核协同单元内部结构图

指令发送单元加载流指令到指令调度单元,标量核获取流处理核状态,标量数据在标量核同流处理核之间的传递.对这些寄存器的读写方式有两种:(1)通过地址直接访问,这是因为一部分寄存器(如标量寄存器)被映射成标量核地址空间的一段,因此标量核可以通过对这部分地址空间的读写来直接访问寄存器;(2)通过发射流指令对寄存器进行间接访问,如核心程序产生的标量结果向标量核的传输,其具体实现见 4.3 节.协同接口寄存器组内的主要寄存器及访问方式如表 1 所示.其中 StartMaskRF 和 FreeMaskRF 分别存放指令调度单元中指令队列的状态码 StartMask 和 FreeMask,状态码有 32 位,对应记分牌中指令队列的 32 个项.

表 1 协同接口主要寄存器及访问方式		
寄存器	通过地址访问	通过流指令
标量寄存器(STRs)	R	R/W
流描述寄存器(SDRs)		R/W
流描述寄存器长度字段(SDRLength)		R/W
存储器地址寄存器(MARs)		R/W
微控制器寄存器(UCRs)		R/W
微程序计数器(MPC)		W
指令队列忙碌寄存器(StartMaskRF)		R
指令队列空闲寄存器(FreeMaskRF)		R

4.1.3 指令调度单元

指令调度单元的主要模块有记分牌和指令发射单元,通过动态调度流指令的发射来控制流处理核内部各功能部件的工作:指令调度单元接受来自协同接口寄存器组的流指令,并根据流处理核的当前状态将流指令发射到流处理核内部的功能部件上,以执行流指令的方式启动流应用的核心程序、流数据和标量数据的传输.流指令的动态调度和发射主要依靠记分牌结构:记分牌内部具有一个指令队列,是一个 32 项的动态优先级 FIFO 队列,每项对应一

个指令发射槽,每项的结构和功能如表 2 所示;指令队列按 FIFO 顺序缓存未执行的指令、正在执行的指令以及指令之间的相关性信息;记分牌内部逻辑(包括资源译码单元、指令入队出队控制单元、资源分析单元、指令发射判断单元)根据指令队列状态码 StartMask(表示指令正在执行)、FreeMask(表示指令队列空闲项)、资源需求码 ResMask 和资源使用情况 BusyMask 来控制流指令入队出队、指令发射和指令相关信息的更新.指令动态调度和发射的具体实现过程见 4.3 节.

表 2 指令队列项的结构		
名字	位数	描述
StreamInstr	80	流指令
LIS	5	发射槽号
CompDep	32	指令完成相关性掩码
IssueDep	32	指令流出相关性掩码
ResMask	52	资源相关性掩码
Prog	1	指令发射标志

4.2 异构核协同的软件

4.2.1 软件协同模块

软件协同模块是运行在标量核上的一个动态调度器,向编译过后的流级程序提供调用接口,完成标量核和异构核协同单元之间的链接,其内部结构如图 4 所示.软件协同模块主要完成两个功能:

(1)生成流指令二进制码并提交给异构核协同单元.流应用程序中的流函数经编译后生成的是流操作,流操作中包含流指令和很多编译辅助信息(如指令间相关信息),软件协同模块通过调用接口接收编译器发送过来的流操作,保存在流操作数组中,软件逻辑负责根据流操作的内容生成流指令的二进制码,并通过同总线相连的交互接口发送给异构核协同单元.

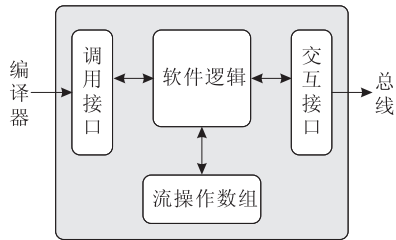


图 4 软件协同模块内部结构图

(2)流数据进行传输时写外设接口控制器,使外设准备好传输.在流处理核同外设接口进行流数据传输时,软件协同模块负责初始化外设接口控制器.由于外设不同,软件协同模块通过加载驱动程序的方式使它们进入就绪状态,准备好同流处理核间的数据传输.这种只需要在软件协同模块中加载驱动程序而不更改流处理器硬件,就能够支持系统中的各种外设,体现了软硬结合协同方法的灵活性.

4.2.2 指令集扩展和编译支持

双缓冲的执行需要指令集的扩展和编译的支持,本文采用斯坦福大学的 StreamC 编译器 ISTREAM^[14],并针对双缓冲的要求进行了修改.

对流指令集的扩展包括修改指令格式和增加新的指令.增加的流指令是 StreamLen send,用来在双缓冲情况下,将流的总长度通过指令发送给异构核协同单元.新的指令格式如图 5 所示,其中 RAW Mask 和 WAR Mask 是指令的相关信息,LIS 是流指令的发射槽号,DB 是双缓冲标志,StreamInstr 是具体的指令内容,包括寄存器读写指令、流数据传输指令、启动核心程序执行指令等,用来控制流处理核的执行.

RAW Mask	WAR Mask	LIS	DB	StreamInstr
----------	----------	-----	----	-------------

图 5 指令格式

与指令集的扩展相对应,编译器要为每条流指令添加一个 db 参数来表示它是双缓冲操作,并且在双缓冲情况下,编译结果中需要添加一条 StreamLen send 指令.图 6 显示了双缓冲发生时流加载函数的编译结果,SDR 0 和 SDR 1 表示 SRF 中的两片大小相同的空间,len 参数表示该空间的大小,参数

```
SDR write(SDR 0, adr 2000, len 100, s 17, db 1)  SRF 分配的两块
SDR write(SDR 1, adr 2100, len 100, s 18, db 1)  -- 半空间长度
StreamLen send(slo len 630, s 19, db 1)
MAR write(MAR 0, adr 0, s 20, db 1)
Stream Receive(SDR 0, s 21, db 1)
Memory Save(SDR 0, MAR 0, s 22, db 1)
----- 流的总长度
```

图 6 发生双缓冲时长流加载的编译结果

s 是编译分配的逻辑发射槽号,db 为 1 时表示该操作是双缓冲操作.双缓冲发生时,编译生成的流指令送入异构核协同单元进行指令动态调度和发送,其具体的实现见 4.3 节.

4.3 异构核协同任务的实现

异构核协同的 4 个基本任务在软硬结合协同方法下的实现过程详解如下.

4.3.1 生成流指令及流指令的动态发送

应用程序执行时,需要生成对应的流指令,并经过标量核—总线—指令发送单元—协同接口的路径把流指令送入记分牌,这一过程成为流指令的发送,具体实现如下:

编译器将应用程序中与流处理核相关的流函数通过静态编译生成流操作序列,软件协同模块根据流操作生成对应的流指令序列.异构核协同单元通过总线接收软件协同模块发送过来的流指令,按顺序存入指令 FIFO,准备发送.这时发送控制逻辑从接口寄存器组读取记分牌内部指令队列的当前状态 FreeMask,如果 FIFO 中第一条流指令的发射槽对应的 FreeMask 位为 1(即记分牌内指令队列的对应项空闲),则将该流指令通过接口寄存器组发送到记分牌,同时该指令离开 FIFO 队列,后面的指令向前移动.相反地,若 FreeMask 对应位为 0,则暂存在 FIFO 中的流指令不发送,然后由发送控制逻辑负责继续查询 FreeMask,直到 FreeMask 对应位变为 1,再立即发送.

4.3.2 流指令动态调度及发射

当流指令进入记分牌后,就开始指令的动态调度及发射过程.这里的指令发射同前面的指令发送有所不同,是指记分牌根据指令相关信息动态调度流指令并向流处理核发射的过程,流指令在指令发送单元内是按序发送的,而在记分牌内是乱序发射的.

记分牌中,资源译码单元根据接收到的流指令相关信息生成 ResMask.指令入队出队控制单元负责初始化队列项的各字段,初值如下:

StreamInstr, LIS, ResMask 字段保持不变.

完成相关字段 $CompDep = RAWMask \& \sim FreeMask$.

流出相关字段 $IssueDep = WARMask \& (\sim (StartMask \& \sim FreeMask))$.

$Prog = 0$.

根据指令队列项字段和当前指令队列的状态码判断指令是否可以发射,发射条件如下:

$CompDep \& \sim free\ mask = 0$;

IssueDep&~start mask=0;
资源相关性得到满足;ResMask&BusyMask=0;
同步操作相关性得到满足,这点已经隐藏在资源相关性中;

按 FIFO 顺序该指令前面没有 Barrier 指令;
按 FIFO 顺序是满足所有以上条件的排在最前的指令.

指令发射判断逻辑将满足发射条件的流指令送入指令发射单元进行发射. 指令发射时,需要更改记分牌内指令队列项的相关性信息:所有指令项的 IssueDep 相应位清零,表示发射相关性已经满足;Prog 置 1,StartMask 相应位置 1;该指令占用的资源码通过指令发射判断单元发送给资源分析单元,对流处理核当前资源使用情况进行更新并发送给指令队列来更新 ResMask.

当指令使用的全部资源变成空闲时,表示该指令执行完毕,指令入队出队控制单元将该指令移出指令队列,该指令后的所有指令向前挪一项. 同时,队列中所有指令的 CompDep 相应位清零,表示指令完成相关性已经满足;StartMask 相应位清零,FreeMask 相应位置 1.

相比之下,流指令的动态调度及发送过程在原有的纯软件协同方法中完全由软件调度器完成,其中指令执行状态和流处理核状态信息的软件查询过程严重影响了指令发送的性能,根据第 5 节的数据可知,软件方法的异构核协同时间占了核心程序总时间的一半以上,这主要是由于指令发送前软件查询状态引起的.

4. 3. 3 标量数据传输

应用程序执行时,需要在标量核同流处理核之间传输标量数据,包括变长流的长度和核心程序产生的标量结果,它们都存放在相应的寄存器中. 标量数据传输是利用协同接口寄存器组内的标量寄存器来完成的(以标量核读取流处理核的标量数据为例):首先,异构核协同单元将寄存器读操作流指令(如读流长度所在寄存器的内容)发送给流处理核;然后,寄存器读操作会将读出的标量数据先写入标量寄存器;这里,对标量寄存器的写操作是通过一个 MOVE 或 WRITE_IMM 指令来实现,因此指令发送单元内的发送控制逻辑测试 FreeMask 的值直到 MOVE 指令或 WRITE_IMM 指令已经完成后,再由标量核通过总线读标量寄存器地址获得其中的标量数据.

而在原有的纯软件方法中,标量数据的传输也

是通过发送寄存器读写操作流指令来完成的,因此由于指令发送前对状态的循环查询过程引起的性能开销依然存在.

4. 3. 4 流数据传输

流数据传输是指流数据在流处理核和 DRAM 或外设接口之间的传输. 按照前面介绍的流指令的发送过程,异构核协同单元向流处理核发送流传输指令,如 memop,使流处理核准备好传输. 如果流传输的另外一端是外设,则由标量核上的软件协同模块加载驱动程序,初始化外设接口控制器,建立流处理核中的 SRF 同外设接口的连接,然后启动流传输. 流指令发射以后,标量核就将控制权交给异构核协同单元,由协同单元向地址产生器发送读写信号,控制流数据的传输. 如果另一端是 DRAM,则异构核协同单元的指令单元发送控制信号控制地址产生器通过总线读写 DRAM.

前文提到的双缓冲是流数据的传输中的特殊情况,需要编译和硬件等不同层次的支持,下面以 4. 2 节图 6 所示的情况为例,介绍双缓冲的实现机制. 从图 6 中的编译结果可知,要传输的长流总长度为 630,而编译在 SRF 中分配的空间只有 200,所以需要将长流切分成 7 段短流,采用双缓冲:将 SRF 可用空间分成大小为 100 的两块空间,循环交替使用,即一段短流加载至其中一块空间(对应图 6 中的 Stream Receive 指令),而另一块空间将其中的短流加载至流处理核的片外 DRAM(对应图 6 中的 Memory Save 指令),总共经过 7 次发送后完成长流加载,实际发送的流指令序列如图 7 所示. 然而编译结果只有 6 条流指令,因此指令发送单元的指令 FIFO 也只接收到 7 条流指令,所以需要指令发送单元动态产生新的流指令来完成双缓冲.

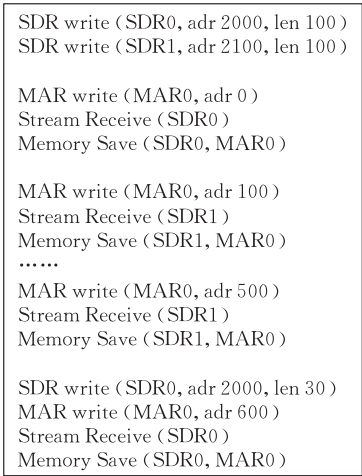


图 7 双缓冲情况时实际发送的流指令

首先,将编译获得的 6 条流指令送入指令发送单元的指令 FIFO 队列缓存,发送控制单元检测到指令 db 字段为 1,则将这 6 条流指令从 FIFO 发送到指令缓冲,同时终止 FIFO 向接口寄存器发送流指令;其次,发送控制单元获取 StreamLen send 指令中的流总长 630,和 SDR write 指令中的半缓冲长度 100,计算出总共发送 6 次长为 100 的短流和一次长为 30 的短流;然后,发送控制单元控制指令缓冲向接口寄存器组先发送 SDR write 指令,再循环 6 次发送 MAR write、Stream Receive 和 Memory Save 指令,注意,发送控制逻辑要更改指令的参数字段,在第奇数次发送时,指令参数为 SDR 0,第偶数次发送时,指令参数为 SDR 1;最后一次循环发送前,需要发送一次 SDR write 指令,长度参数改为 30.在完成双缓冲后,发送控制逻辑继续指令 FIFO 的流指令发送,在硬件上由一个多路选择器来选择指令 FIFO 或指令缓冲.

使用原有纯软件方法传输流数据时,每发送一个数据都需要软件通过总线向地址产生器发送读写信号来控制数据的传输,软件的低速影响了传输性能.并且双缓冲情况在纯软件方法的实现中,由软件调度器动态生成实际发送的流指令序列并发送给流处理核,当流长度大大超过 SRF 分配的空间容量时,需要发送的流指令条数也根据发送次数成线性增长,软件状态查询带来的开销也随之增大.

5 异构核协同性能评测和分析

为了评价本文提出的软硬结合的异构核协同方法,我们在 MASA-I 流处理 SOPC 系统上实现了该方法.整个系统是在 Altera 公司的 StratixII EP2S180 上实现的,设计中采用符合 Altera 公司的 AVLAON 总线标准的片内高速总线,采用 NIOS II 作为标量核;为了进行各类时间的统计,在硬件中加入性能统计模块,统计诸如 kernelBusy、kernelBlock、agBusy、agBlock 等时间项.本文采用了不同的测试手段来测试协同单元的主要功能和性能,包括数据的传输、核心程序的执行.为了进行性能和硬件资源开销的对比,同样的测试也在采用软件协同方法的 MASA-I 中运行(运行平台的其他硬件设置相同).

5.1 数据传输

在流体系结构中,数据传输的性能主要取决于流数据传输的性能.本文测试了软硬结合协同

方法和软件协同方法在流数据传输中的性能差别,测试手段是加载并保存由 8192 个整形数据所组成的流.测试结果如表 3 所示,表中 Load time 和 Save time 分别是加载和保存数据所消耗的时间,agTotalBusyTimeh 和 agAvalonBlockTime 分别是存储控制器中地址产生器的繁忙时间和地址产生器的 Avalon 总线阻塞时间.

表 3 数据传输性能对比

各时间项	软件方法消耗的时间/Cycle 数	软硬结合法消耗的时间/Cycle 数
Load time	48923	34203
Save time	38385	25823
agTotalBusyTime	32361	17386
agAvalonBlockTime	15960	985

从表 3 中可以看出,使用软硬结合的协同方法,流数据加载和保存时间都减少 30% 以上;地址产生器的繁忙时间也有减少近 50% 的,地址产生器的 Avalon 总线阻塞时间则大幅度降低,仅为软件方法的 6%.这是因为软件方法中,每传输一个数据都要由软件调度器通过总线向地址产生器发送一次读写控制信号,速度十分缓慢;而软硬结合的协同方法使用异构核协同单元直接向流处理核中的地址产生器发送读写控制信号,速度快,并且不需要占用总线资源.

5.2 核心算法执行

本文选择媒体和数字信号处理领域的常用核心算法程序的执行来测试两种协同方法的性能,表 4 是使用的测试程序描述和参数说明.

表 4 测试采用的核心算法描述及参数

核心程序	简要描述及数据规模参数
FFT	1024 点的复数 FFT,采用基 2 的 FFT 蝶形算法
FIR	一维宽度为 8 的滤波算法,输入长度为 1024 点
DCT	块大小为 4×4 的离散余弦变换,输入 120 个 16×16 的宏块
CONV	3×3 卷积运算,输入数据为 1024×1024 个点
BlockSearch(BS)	Mpeg 中 8×8 的块搜索算法,输入数据为 1024×768

图 8 是在使用软硬结合方法(HS)后,相对于软件方法(SW)的各 kernel 的协同速度和执行速度的加速比,图中左纵轴是核心执行速度加速比坐标,右纵轴是协同速度加速比对数坐标.注意,这里为了单独评测核心的执行速度,所以在统计 kernel 执行时间时不包括 kernel 运行前后的流数据加载和保存时间.从直方图显示的加速比可见,使用软硬结合的方法,FIR 获得高达 38 倍的加速(由于它大部分执行时间都是协同开销),其他各个核心程序的执行也获得 2 倍以上的加速;从折线图显示的加速比来看,

协同速度可以提高 2 个量级. 这是因为软件方法的大部分时间耗费在流指令发送时, 标量核调用软件循环查询流处理核状态, 而软硬结合的异构核协同方法是由硬件层的异构核协同单元查询状态.

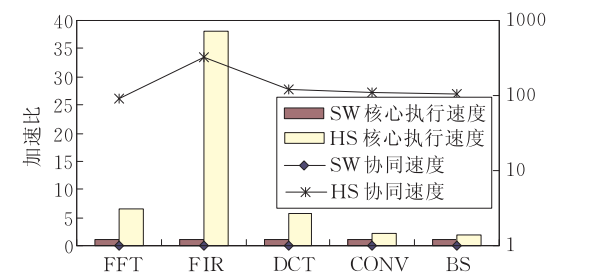


图 8 软硬结合方法相对于软件方法的核心执行速度和协同速度的加速比

进一步分析软件方法和软硬结合协同方法中, 核心程序总执行时间内的各部分时间所占比例, 结果如图 9 所示. 核心程序执行总时间包含 5 部分: 流数据加载和保存时间、kernel busy 时间 (kernel 在流处理核中的运行时间)、kernel block 时间 (kernel 在流处理核中的阻塞时间)、标量核执行时间、异构核协同时间开销. 对于每种核心程序而言, 使用这两种协同方法, 除异构核协同时间外的 4 部分消耗的绝对时间基本相同 (数据存取时间也相差不到 50%), 但是对比图中左侧 5 个直方和右侧 5 个直方, 这 4 部分时间占总时间的百分比相差却很大, 这其实是异构核协同时间所占比例的巨大变化造成的. 使用软件方法的异构核协同时间在 5 个测试中都占总执行时间的 30% 以上, 特别是 FFT、FIR 和 DCT 异构核协同时间占 70%, 而使用软硬协同方法时这部分时间的百分比仅占 5% 以下, 这导致核心程序总执行速度提高一倍.

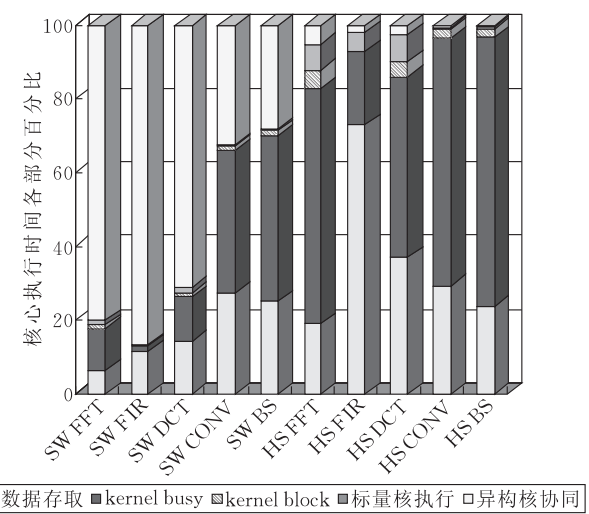


图 9 核心程序执行时间各部分百分比

从上面的测试可知, 使用软硬结合的异构核协同方法能够在实际的算法执行中获得良好的性能提升, 实验中选取的应用程序是媒体和数字信号处理领域的常用核心算法, 具有一定的代表性. 对于更广泛的应用而言, 只要符合流计算模型, 就需要异构核协同完成, 因此提高异构核的协同性能就一定会提高应用的性能. 性能提高的大小与应用本身的特点有关, 协同开销大的应用, 性能提高明显.

需要说明的是软硬结合的方法需要消耗比软件方法更多的硬件资源, 表 5 给出了在 MASA-I 流处理 SOPC 系统上实现的两种方法的硬件资源开销对比. 其中逻辑资源 ALUT 和寄存器资源 Register 的单位都是个数, Memory 资源的单位是 bit 数.

表 5 软硬结合方法与软件方法的硬件资源开销对比

	ALUT	Register	Memory
软件协同方法	83276	24838	3432688
软硬结合协同方法	89214	26998	3443920

从表 5 中数据可知, 软硬结合的协同方法与软件协同方法相比, ALUT 增加了 7.1%, Register 资源增加了 8.6%, Memory 资源仅增加了 0.3%, 相对于获得的性能提升而言这是值得的.

6 结束语

本文提出了一种流体系结构中软硬结合的异构核协同方法, 具体的实现方法包括在硬件层设计一个异构核协同单元, 在软件层设计一个软件协同模块, 以及对编译和指令集进行相应扩展. 这种方法具有如下优点: 在流指令发送中, 由硬件完成流处理核的状态查询及指令的动态调度, 还能在双缓冲情况下动态生成新指令并发送; 在数据传输中, 使用硬件发送控制信号控制地址产生器通过总线读写数据, 这避免了采用纯软件方法进行状态查询、指令发送和数据传输的低效问题. 实验数据表明, 采用本方法能够提高协同速度, 从而提高流处理器的整体性能. 本方法对其他的异构核协同研究具有借鉴意义.

参 考 文 献

[1] Rixner M. Stream Processor Architecture. Boston, MA, USA: Kluwer Academic Publishers, 2002

[2] Kapasi U J, Dally W J et al. The imagine stream processor// Proceedings of the IEEE International Conference on Computer Design. Freiburg, Germany, 2002: 282-288

- [3] Dally W J, Hanrahan P, Erez M et al. Merrimac: Supercomputing with streams//Proceedings of the SuperComputing'03. Phoenix, Arizona, US, 2003: 35-42
- [4] Talor M B et al. The RAW microprocessor: A computational fabric for software circuits and general purpose programs. IEEE Micro, 2002, (3/4): 25-35
- [5] Sankaralingam K et al. Distributed microarchitectural protocols in the TRIPS prototype processor//Proceedings of the 39th Annual International Symposium on Microarchitecture. Orlando, Florida, USA, 2006: 480-491
- [6] Yang Xue-Jun, Yan Xiao-Bo et al. A 64-bit stream processor architecture for scientific applications//Proceedings of the 34th Annual International Symposium on Computer Architecture. San Diego, CA, USA, 2007: 210-219
- [7] Wen Mei, Wu Nan, Li Hai-Yan, Zhang Chun-Yuan. Multiple-morphs adaptive stream architecture. Journal of Computer Science and Technology, 2005, 20(5): 635-646
- [8] Wen Mei. Key techniques research of stream architecture [Ph. D. dissertation]. Department of Computer Science, National University of Defense Technology, Changsha, 2006(in Chinese)
- (文梅. 流体系结构关键技术研究[博士学位论文]. 国防科学技术大学, 长沙, 2006)
- [9] Yang Qian-Ming, Wu Nan, Xun Chang-Qin, He Yi. The implement of MASA-I stream processor on FPGA. Computer Engineering & Science, 2008, 30(3): 114-118(in Chinese) (杨乾明, 伍楠, 荀长庆, 何义. MASA-I 流处理器在 FPGA 上的实现. 计算机工程与科学, 2008, 30(3): 114-118)
- [10] Kahle J A et al. Introduction to the Cell multiprocessor. IBM Journal of Research and Development, 2005, 49(4/5): 589-604
- [11] Ohara M et al. MPI microtask for programming the Cell broadband engine processor. IBM Systems Journal, 2006, 45(1): 85-102
- [12] Mattson P. A programming system for the Imagine media processor[Ph. D. dissertation]. Department of Electrical Engineering, Stanford University, California, USA, 2001
- [13] Markovskiy Y. Quasi-static scheduling for SCORE [Ph. D. dissertation]. CS, U. C. Berkeley, California, USA, 2004
- [14] Mattson P et al. Imagine programming system developer's guide. <http://cva.stanford.edu>, 2002



REN Ju, Ph. D. candidate. His research interests include computer architecture, parallel processing.

HE Yi, Ph. D. candidate. His research interests include computer architecture.

XUN Chang-Qing, master candidate. His research interests include computer architecture, parallel processing.

YANG Qian-Ming, master candidate. His research interests include computer architecture, parallel processing.

WEN Mei, Ph. D. , associate professor. Her research interests include computer architecture, parallel computing.

WU Nan, Ph. D. candidate. His research interests include computer architecture, stream computing, and compiler design.

ZHANG Chun-Yuan, professor, Ph. D. , Ph. D. supervisor. His research interests include multi-computer architecture, scientific computing.