流体系结构指令存储器优化设计研究

管茂林[1],何 义[2],杨乾明[1],张春元[1],伍 楠[1]
(1.国防科学技术大学计算机学院,湖南长沙 410073; 2.北京油料研究所,北京 102300)

摘 要: 针对流体系结构中VLIW代码体积对指令存储器的容量和功耗带来的问题,本文通过分析流处理器的指令特征,提出了一种新的VLIW分域压缩技术.在此基础上,本文为流体系结构设计了分布式的片上指令存储器,并提出了SIMD流水的执行模式.实验结果证明,该技术减少了38%的片外指令访存,降低约65%的片上指令存储器空间需求;分布式指令存储器减少了约37%的片上指令存储器面积,使得MASA的系统面积降低了8.92%,并降低了61%的指令存储器功耗.

关键词: 流体系结构; 分布式指令存储器; VLIW压缩
中图分类号: TP393 **文献标识码:** A **文章编号:** 0372-2112 (2012) 07-1379-07
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2012.07.016

Optimized Design Research of Instruction Memory for Stream Architecture

GUAN Mao-lin[1], HE Yi[2], YANG Qian-ming[1], ZHANG Chun-yuan[1], WU Nan[1]
(1. *College of Computer, National University of Defense Technology, Changsha, Hunan* 410073, *China*;
2. *POL Research Institute of Beijing, Beijing* 102300, *China*)

Abstract: The huge VLIW code size has brought serious problems to the capacity and energy consumption of instruction memory in stream architecture. Through analyzing the characteristics of instruction, this paper proposes a novel domain-divided VLIW compression scheme, designs a distributed on-chip instruction memory for stream architecture and proposes a new SIMD pipeline execution model. The experiment results show that about 38% of the off-chip instruction accessing and 65% of the on-chip instruction memory space demand can be reduced by the compression; the distributed instruction memory depresses about 37% of the area of on-chip instruction memory, and about 8.92% of the area of MASA stream processor. At the same time, the energy consumption of the instruction memory is reduced by 61%.

Key words: stream architecture; distributed instruction memory; very long instruction word compression

1 引言

在当前诸多流处理器[1]中,指令存储器通常采用软件管理的便笺存储器(ScratchPad Memory, SPM)实现.尽管SPM相对于cache的资源消耗具有优势[2],但是,指令存储器在流处理器中的开销仍然不容忽视.Imagine中指令存储器占芯片面积的11%[13];因为运算簇(cluster)的数目多达16个,MERRIMAC中指令存储器的消耗在整个系统中的比例较小,但也与一个运算簇的开销相接近[4].这主要是因为流处理器的指令存储器中存储单元具有较高的位宽.为简化硬件,流处理器通常采用超长指令字(Very Long Instruction Word, VLIW)技术来开发程序的指令级并行性(Instruction Level Parallelism, ILP).Imagine、FT64、MASA中一条VLIW的宽度分别为576、668、512位.随着半导体和VLSI技术的发展,存储器从中获得的收益远小于计算资源获得的收益[5],计算资源越来越廉价,存储器占系统开销的比例将越来越大.

然而,VLIW技术在增加发射宽度来提高并行性的同时,也面临着代码稀疏的严重障碍.受限于程序固有的ILP以及编译器的能力,编译生成的VLIW常常并不能被打满,存在很多的空操作.空操作不仅增加了代码体积,而且严重浪费了片外带宽和片上存储空间,增加了VLIW存取、发射及执行的能量消耗.

国内外很多学者都已经认识到代码体积带来的问题,并在多个层面提出了相应的解决方案.基于硬件词典的指令压缩[8]是一种常见的压缩方法,面向特定领域

收稿日期:2011-09-02;修回日期:2011-11-30
基金项目:国家自然科学基金重点项目(No.61033008);国家自然科学基金(No.60903041,No.61103080);教育部博士点基金(No.20104307110002);国防科学技术大学杰出青年基金(No.CJ11-06-01)

的应用,词典压缩通常可取得很好的压缩效果,但是其缺点在于词典库不具备通用性,对于某些特殊程序,构建合适的词典库非常困难.另一种方法是引入数据压缩的方法,利用压缩编码算法对 VLIW 进行重新编码[10],该方法对应的解码逻辑比较复杂,可能带来较大的硬件开销.SPM 的优化是目前研究的热点,也是提高指令存储器利用率的一个重要而有效的手段.在流水可重构的指令存储器层次中[11],部分 cache bank 可以在运行时进行配置,使其在低于正常 cache 的电压和频率下工作来降低功耗.针对实时嵌入式系统,Stefan Metzlaff 等人在函数粒度上设计了一个硬件控制的指令便签存储器,每条指令都可以从其时间可预测的本地快速便签存储器中取得以提高系统性能[12].指令寄存器文件技术也从单流出处理器应用到 VLIW 体系结构[14],频繁执行的指令被挑选出放到片上指令寄存器文件中以便在程序执行时能够快速取出.通过分析程序特征,我们之前为流体系结构构建了基于 kernel 热代码优化的软件管理和软硬件混合管理的指令存储器[15].

针对上述问题,本文通过分析流体系结构中的指令特征,提出了一种新的 VLIW 分域压缩技术;在此基础上,本文为流体系结构设计了分布式的片上指令存储器.实验结果证明,该技术能有效的减少流处理器的片外指令访存和片上存储空间需求,减少流处理器中片上指令存储器的资源消耗和面积需求;同时,该技术还能有效地降低指令系统的功耗.

2 VLIW 压缩

2.1 指令特征分析与子域划分

如图 1 所示,流处理器的运算簇采用了分布式寄存器文件结构,功能单元的每个输入端口都拥有对应的本地寄存器文件(Local Register File,LRF),输出端口均连接至簇内互联总线.指令执行完成后,不负责将结果写入对应的寄存器文件,而是将其放到簇内互联总线上,由需要该指令结果的功能单元所对应的 LRF 从总线上读取.这就使得所有 LRF 都与功能单元一样,需要在 VLIW 中设置相应的操作位来完成写操作.因此,各功能单元在 VLIW 中对应的字段可分为两个部分:功能单元执行部分和 LRF 写操作部分.而在同一条 VLIW 中,这两部分的指令码显然没有任何相关性,可以划分为互相独立的子域.这种划分方法将 VLIW 中不相关的字段尽可能的分开,使得空操作与有效代码尽量的分离,提高了代码压缩的空间.

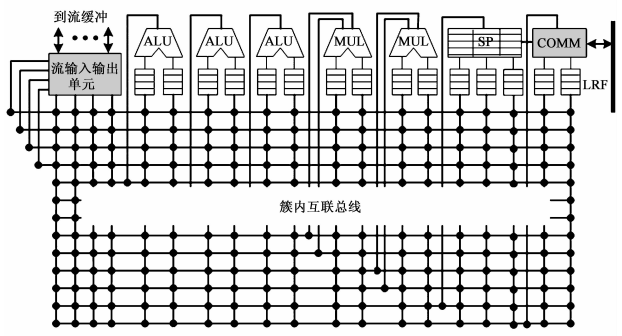
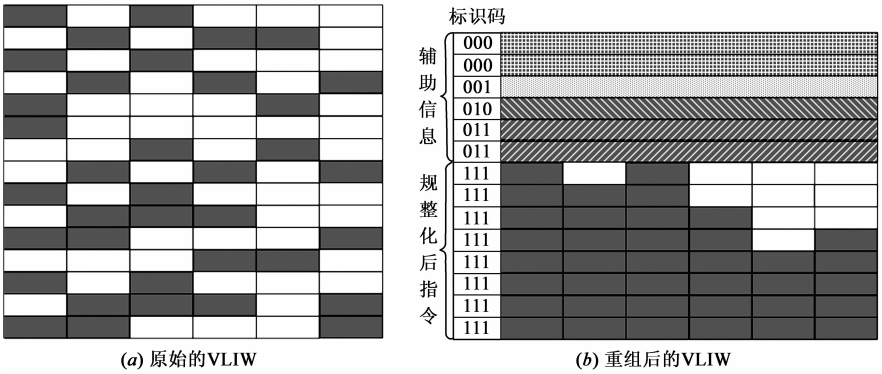


图1 流处理器运算簇结构

2.2 代码压缩

通过分析流体系结构中的指令特征,本文提出了一种新的 VLIW 分域压缩技术,其主要思想是将 VLIW 各个子域中的空操作剔除,减少无效指令对存储带宽和存储空间的浪费.如图 2 所示,图 2(*a*)是程序原始的 VLIW 示意图,其中白色部分表示无效的空操作;图 2(*b*)中下半部分灰色部分是分域压缩之后的 VLIW,它将原始指令中的空操作删除,并将所有相同子域的有效指令按先后顺序无缝的连续组合.形象的说,就是将图 2(*a*)中的白色部分删除,灰色部分"自由落体"堆积生成压缩后的 VLIW.编译器对每条 VLIW 的各个子域分别进行检测,判断其是否为空操作,并记录该信息.根据已记录的子域空操作信息,对原始的 VLIW 代码自底向上依次记录各子域代码的有效操作,并计算各个子域的有效代码的数目以及所有子域有效代码数目的最大值,这也是压缩后的 VLIW 序列的长度.然后检测出每条分支指令,获取其分支目标指令的偏移地址,在这两条指令之间,根据前面记录的子域空操作信息,计算分支指令在压缩后的指令中每个子域实际的偏移地址,记录该信息.



(*a*) 原始的VLIW (*b*) 重组后的VLIW

图2 VLIW压缩

3 分布式指令存储器

3.1 总体结构

本文以有 8 个 cluster 的 MASA 流处理器为例,设计了分布式指令

存储器.如图 3 所示,分布式指令存储器不仅将流处理器中多个 cluster 共享的集中式指令存储器分布到各个 cluster 中,而且将指令存储器根据划分好的 VLIW 子域,进一步切割成多个更小的分布式指令存储器(Distributed Instruction Memory,DIM),分别与各子域对应的功能单元直接紧密连接,为其提供所需的指令.DIM 中只存放压缩后的有效指令.所有的 cluster 中相同子域对应的 DIM 采用统一编址模式.编译器根据经典的图着色算法,将压缩后的每个子域的非空指令,分别在各自对应的所有 DIM 中进行分配.此外,在所有 cluster 相同子域对应的 DIM 以及微控制器(MicroController,MC)之间增加了指令传输通路.当执行某条 VLIW 时,首先从对应的 DIM 中获取这条 VLIW 各个子域的有效指令,在微控制器内组装成原有的 VLIW,然后再依次发送到各个 cluster 启动执行.

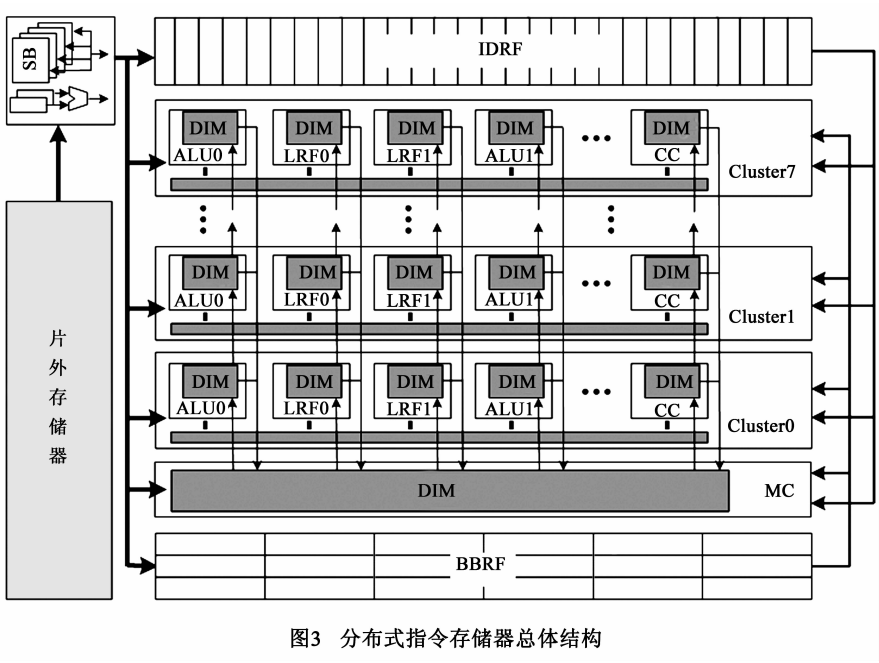


图3　分布式指令存储器总体结构

3.2　BBRF

为了保证压缩后的 VLIW 能正确执行,需要在两方面还原程序的执行信息.首先,需要在合适的时机插入空操作以还原原始的 VLIW.其次,执行分支指令时,PC 能指向各子域正确的目标地址.为此,本文设计了索引寄存器文件(Indexed Register File,IDRF)和 kernel 起始地址与分支偏移量寄存器文件(Begin & Branch Register File,BBRF),如图 3 所示.IDRF 和 BBRF 与所有的 cluster 连接.IDRF 中存放编译器为程序的每条原始 VLIW 建立的若干位的空操作标识指令,分别与 VLIW 的每个子域对应,每一位分别表示对应子域中的指令是否为空操作.BBRF用来存储 kernel 的起始地址和分支信息,如图 4 所示.编译器统计并记录各个子域分支指令和目的地址间的非空指令数作为各个子域的偏移地址,当执行分支指令时,PC 根据每个子域的偏移地址寻找压缩后 VLIW 在各个子域的目标指令.对于 kernel 的起始地址信息,需要记录 VLIW 所有子域的第一条有效指令所在 DIM 对应的 cluster 编号,以及指令在该 DIM 中的地址.

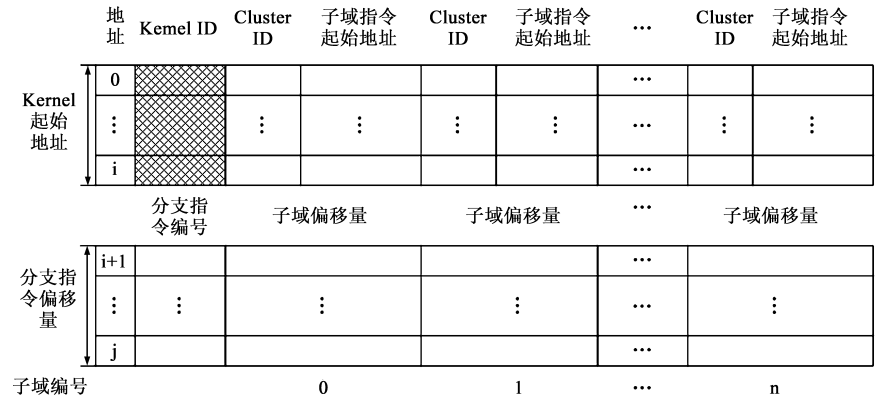
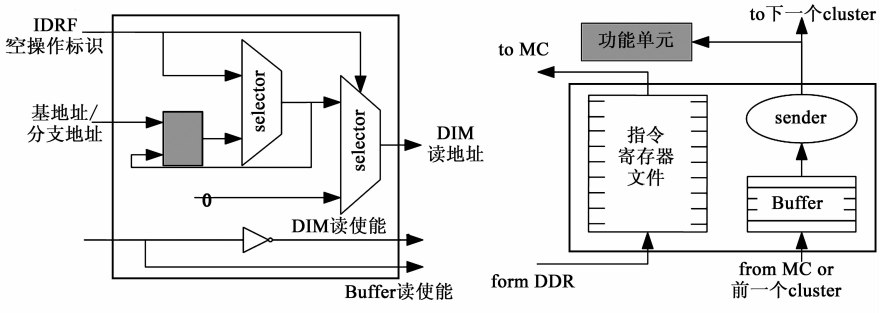


图4　BBRF结构



(*a*) 本地控制器　　(*b*) 指令存储传输模块

图5　DIM结构

3.3 DIM

DIM 的结构如图 5 所示,包括本地控制器和指令存储传输模块.另外,微控制器的 DIM 还包括一个很小的指令重组寄存器文件,负责从各个 cluster 接收要执行的有效指令,然后再将其发送给 cluster 执行.本地控制器负责产生 DIM 的读地址、DIM 和 Buffer 的读使能,分支指令依然在全局的微控制器中执行.指令存储传输模块包括指令寄存器文件、缓冲单元(Buffer)和传输器(sender)三个部分.指令寄存器文件存储由片外存储器加载的有效指令;Buffer 存储由互连通路上传输至当前 cluster 的指令;sender 将当前接收到的指令通过互连通路传输至下一个 cluster,同时负责将需要执行的指令送入功能单元执行.

4 系统执行机制

4.1 指令加载

分域压缩改变了原始 VLIW 各子域的操作之间的相关关系,如何在片外合理的组织 VLIW 写入片上指令存储器,成为 VLIW 压缩后的一个关键问题.首先将压缩后的 VLIW 进行规整化,即所有的子域都按照压缩后最长子域进行规整,长度不足的子域均在最后添补适量的空操作,如图 2(*b*)中白色部分所示.VLIW 压缩后,程序执行所需的辅助信息需要与 VLIW 一并打包送入运算簇.本文将 VLIW 规整化、添加辅助信息等过程称为 VLIW 重组.为了区分压缩后的指令与辅助信息,本文在重组后的指令之前添加了 3 位标识码,用以说明其后的每条指令的实际意义,如图 2(*b*)所示.表 1 对重组后的 VLIW 进行了详细说明.

表 1 重组后的 VLIW 描述

标识码	描述	存储区域
000	原始 VLIW 的各个子域是否有效的标识指令	IDRF
001	原始 VLIW 及其各个子域的有效代码长度	MC
010	程序执行时各个子域第一条有效指令的地址	BBRF
011	分支指令在各个子域实际的分支偏移量	BBRF
111	各子域实际的有效代码	DIM

重组后的 VLIW 包含了多种信息,因此,指令加载时需要区分各二进制代码的类型,引导不同类型的二进制代码集合写入各自对应的存储区域.从片外存储器读取一条 VLIW 后,根据其标识码进行判断,区分该条指令后续二进制代码记录的信息类型,并产生各存储器的"写使能"信号.IDRF、微控制器和 BBRF 的写使能均可根据表 1 中的标识码信息进行判定生成.而对于 DIM 的写使能,不仅需要判定读入 VLIW 的标识码,还要判断该子域的非空操作是否已经写入完成.若写入某子域对应的 DIM 的指令数已经达到该子域的非空操作数,即将对应的 DIM 写使能信号置为无效,阻止后续的空操作指令写入 DIM 中.

4.2 硬件解压缩模型

程序的执行过程如下:首先,微控制器从 BBRF 中读取 kernel 各个子域第一条有效指令在各 DIM 中的地址,并将各 DIM 的 PC 值指向对应的起始地址;然后,从 IDRF 中读取第一条 VLIW 对应的空操作标识指令,并将各子域的标识位送入对应的 DIM 中.若某 DIM 对应的子域标识位为 1,则将其 PC 指向的指令送入微控制器 DIM 中的指令重组寄存器文件,PC 加 1;否则,将 1 位的空操作标识当作一条指令送入指令重组寄存器文件,PC 值不变,这样可以避免大量空操作的传输.然后将指令重组寄存器文件的所有指令通过指令传输通路依次发送至所有 cluster 并启动 cluster 执行.在指令执行时,若译码为一条分支指令,微控制器读取 BBRF 中该分支指令各子域对应的偏移量,若分支成功,各 DIM 通过当前 PC 值与偏移量计算出分支目标指令的 PC 值,若分支不成功,则执行下一条指令.按照这种执行方式,依次对 kernel 中所有的 VLIW 进行处理,直至 kernel 结束.图 6 展示了硬件解压缩模型,根据上述的执行模式可以看出,该解压缩模型并不是将压缩后的代码还原成原有的 VLIW,而是在硬件执行上保持了原 VLIW 所完成的功能.可以看出,所有功能单元从各自的 DIM 中获取的指令,在逻辑上可以组装还原成未压缩的原始 VLIW.

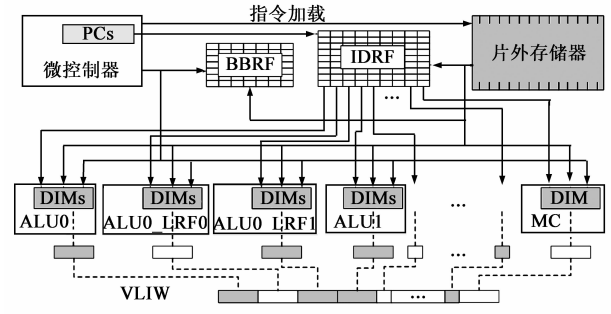


图6 硬件解压缩模型

4.3 SIMD 流水执行

采用分布式指令存储器,指令不能同时到达所有的 cluster,所以流体系结构不能保持原有的 SIMD 模式.本文提出了新的执行模式,如图 7 所示.要执行的 VLIW 在微控制器重组之后发送到 cluster0 的各个 DIM 的 Buffer 中,Buffer 将所有的指令发送给对应的功能单元执行,同时将其送往 cluster1 的 Buffer 中;下一周期 cluster1 执行该些指令并将其送往 cluster2 的 Buffer 中,cluster0 执行下一条 VLIW,并将其送往 cluster1 中;依此类推.可以看出,传统流体系结构中 8 个 cluster 同时执行的一条 VLIW,被转变为 8 个 cluster 分 8 个周期流水完

成.虽然单条指令的执行时间延长了 8 倍,但由于 cluster 向后一个 cluster 传送当前指令的同时,也在接收前一个 cluster 传来的下一条指令,这种流水的方式保持了流体系结构的 SIMD 执行模式,在本文中称为 SIMD 流水模式.

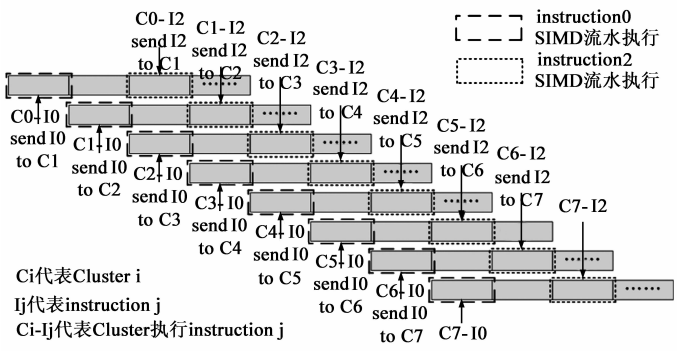


图7　SIMD流水模式

5　实验与结果分析

5.1　压缩效率分析

为了评估压缩效果,本文选择 RS(Reed Solomon 解码算法[16])、Susan(Mibench 中的图像处理边角检测算法[17])、LUCAS(利用 Lucas-Lehmer 方法检测梅生素数[18])、MPEG2(MPEG2 标准视频图像压缩算法[19])、Ygx2(IAPCM 中的利用拉格朗日与欧拉算法计算二维爆炸流体力学问题[3])、H.264(JVT 提出的一种新的视频编码算法[7])等六个应用在 MASA 流处理器上进行了测试分析.实验结果如图 8 所示.其中,片外压缩比是重组后的 VLIW 代码体积与原始 VLIW 代码体积的比值,片上压缩比是存放在分布式指令存储器内的所有有效指令包括原始 VLIW 中所有的非空指令、空操作标识指令、kernel 起始地址和分支偏移信息等的代码体积与原始 VLIW 代码体积的比值.average 是六个应用的平均压缩效率.

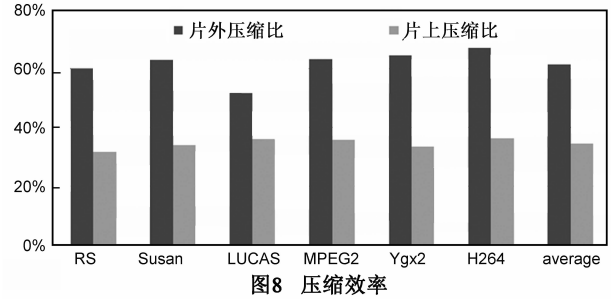


图8　压缩效率

从图 8 中可以看出,在流处理器中采用分域压缩技术,片外指令码平均可获得 61.44% 的压缩比,片上指令码平均可获得 34.58% 的压缩比,能够有效的减少代码体积,节省宝贵的指令存储器空间.

J Liu 等人通过在 Intel Itanium 处理器上分析测试 SPEC CPU2000、MediaBench 和 PacketBench 等得出,空操作数平均达到 35% 左右[6].按照这种方式,即便是将其中的空操作完全删除,其压缩效率也远低于本文提出的分域压缩技术的压缩效率.这主要有两个原因.一是流处理器运算簇中功能单元众多,虽然在核心循环体中,计算单元的利用率较高,但是在非核心循环部分,其利用率较低;其他特殊功能单元在面对不同的应用时,利用率也差异很大.二是分布式寄存器文件结构使得功能单元与其对应的 LRF 的写操作解耦合,使得我们能够对流处理器的 VLIW 进行更深层次的子域划分,解耦合了其中绝大部分弱关联的子域字段,增加了代码压缩的空间.

5.2　资源消耗

本文对采用分布式指令存储器的 MASA 流处理器进行了资源消耗分析,并与采用集中式指令存储器进行了对比.集中式指令存储器采用 RAM 实现,在分布式指令存储器中,IDRF 采用 RAM 实现,而 DIM 和 BBRF 均采用寄存器文件实现.MASA 中集中式指令存储器能够存储 2K 条 VLIW.根据 5.1 节对应用压缩比的分析,本文将每个 DIM 定义为 128 项,即总容量为集中式指令存储器的 50%,大于所有应用的片上压缩比.这样一方面是为了保证程序执行的性能,另一方面也简化了实验,避免分别为每个应用定制分布式指令存储器,同时也方便了对比分析.

表 2　资源消耗评估(单位:逻辑门)

	集中式指令存储器	分布式指令存储器		面积节省
整个系统	7068824	6438285		8.92%
片上指令存储器	1901514	总数	1193376	37.24%
		IDRF	88170	
		BBRF	3173	
		每个 cluster 的所有 DIM	135638	
		MC 的 DIM	16932	

本文采用了 0.18μm CMOS 工艺,利用 Synopsis Design Complier 工具基于标准单元库逻辑综合获取代价信息,寄存器文件、RAM 均由 Memory Compiler 定制生成.系统各部分的资源消耗如表 2 所示.可以看出,虽然 IDRF、BBRF 在解压缩时不可或缺,但是它们所产生的资源消耗在整个分布式指令存储器中仍然只占很小的比例,最大的开销仍然是存放实际有效代码的 DIM.这主要是因为压缩时产生的辅助信息的代码相对于实际有效代码只是很小的一部分,不需要太大的存储空间,这一点在前面的压缩效率评估中的片上压缩比中也有一定的体现.从表 2 中可以看出,相对于集中式指令存储器,分布式指令存储器减少了 37.24% 的片上指令存储器面积,使得 MASA 的系统面积减少了 8.92%.

5.3 能耗分析

根据文献[13],本文构建了如下的指令存储器功耗模型.从集中式指令存储器中读取一条 VLIW 送入所有 cluster 所需的能耗可根据式(1)计算,其中 S_{CIM} 表示集中式指令存储器的容量,Wd_{VLIW} 为 VLIW 的宽度,E_{sram} 为访问 1 位 SRAM 的能耗,E_w 表示单位长度线传输消耗的能量,$Wd_{cluster}$ 表示送入 cluster 的 VLIW 宽度,C 表示 cluster 的数目,A_{clst}、A_{srf}、A_{comm} 分别表示 cluster、SRF 和簇间通信总线的面积.与之对应,VLIW 所有有效指令从分布式指令存储器中读出,且传递至各个 cluster 对应的功能单元所需的能耗可由式(2)计算.其中 S_{DIM_i} 表示第 i 个子域对应的 DIM 容量,Wd_i 表示第 i 个子域的指令宽度,E_{RF} 表示读取 1 位寄存器文件数据所需的能量,λ 表示平均非空操作比例,S_{Buffer} 表示 Buffer 的容量,E_{Buffer} 表示访问 1 位 Buffer 中数据需要的能量,Wd_{IDRF} 表示 IDRF 的宽度.

$$E_{CIM} = S_{CIM} * Wd_{VLIW} * E_{sram} + Wd_{cluster} * E_w * \left(\sqrt{C} * \sqrt{C * A_{clst} + C * A_{srf} + A_{comm}}\right) \quad (1)$$

$$E_{DIM} = \sum\left(S_{DIM_i} * Wd_i * E_{RF} * \lambda\right) + Wd_{cluster} * \sqrt{A_{clst}} * \lambda * 8 + S_{Buffer} * Wd_{cluster} * E_{Buffer} * \lambda + S_{IDRF} * Wd_{IDRF} * E_{sram} + Wd_{IDRF} * E_w * \left(\sqrt{C} * \sqrt{C * A_{clst} + C * A_{srf} + A_{comm}}\right) \quad (2)$$

分布式指令存储器从三个方面减少了指令存储器功耗,第一,将高位宽的长线全局 VLIW 传输转换为低位宽的短距离指令传输;第二,分域压缩技术在很大程度上减少了读取和传输指令的总位数;第三,分布式指令存储器使得单个 DIM 的容量只有原来的集中式指令存储器容量的 1/16,这极大的降低了读写指令存储器的能耗.根据上述公式,通过在 Msim[9] 模拟器中建模测试,分析应用在集中式和分布式指令存储器下的功耗,我们发现分布式指令存储器降低了 61% 的指令存储器功耗,降低了 9.45% 的系统总功耗.另外,5.1 节已经指出,分域压缩技术减少了约 38% 的片外指令访存,因此,该技术还能有效的减少访问片外指令存储器的能量开销.

6 结论

VLIW 处理器的代码体积一直是困扰研究人员的重要问题.在流体系结构中,大量的功能单元和分布式寄存器文件结构使得 VLIW 结构异常复杂,代码体积问题更加严重.本文通过分析流处理器中的指令特征,提出了一种新的 VLIW 分域压缩技术,并设计了分布式指令存储器对压缩后的代码进行解压缩执行.实验证明,分域压缩技术能够有效减少流处理器中的指令访存、降低片上指令存储器空间需求;分布式指令存储器能够有效减少片上指令存储器面积,降低系统资源消耗;二者结合还能降低系统功耗.

参考文献

[1] 张春元等.流处理器研究与设计[M].电子工业出版社,2009.

[2] R Banakar, et al. Scratchpad memory: design alternative for cache on-chip memory in embedded systems[A]. Proceedings of the 10th International Symposium on Hardware/Software Codesign[C]. Estes Park, USA: IEEE Computer Society, 2002. 73 - 78.

[3] 袁国兴,邵京云.评几种高档微处理器在运算科学计算问题时的性能[EB/OL].北京应用物理与计算数学研究所,http://www.ccw.com.cn,2003,4.
Yuan Guoxing, Shao jingyun. Evaluating performance of scientific applications on several high performance processors[EB/OL]. Beijing Institute of Application Physics and Computing Mathematics, http://www.ccw.com.cn, 2003, 4. (in Chinese)

[4] Mattan Erez. MERRIMAC high-performance and highly-efficient scientific computing with streams[D]. Palo Alto, CA: Stanford University, 2007.

[5] James Balfour, et al. An energy-efficient processor architecture for embedded systems[J]. IEEE Computer Architecture Letters, 2008, 7(1): 29 - 32.

[6] J Liu, et al. Analysis and characterization of Intel Itanium instruction bundles for improving VLIW processor performance [A]. Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences[C]. NY, USA: IEEE Computer Society, 2006. 389 - 396.

[7] YuriV Ivanov, C J Bleakley. Dynamic complexity scaling for real-time H.264/AVC video encoding[A]. Proceedings of the 2007 ACM International Conference on MultiMedia[C]. Augsburg, Germany: ACM Press, 2007. 962 - 970.

[8] 赖明澈等.基于代码特征分析的 TTA 指令压缩技术与解压部件实现[J].电子学报,2008,36(11):2234 - 2238.
Lai Ming-che, et al. Characteristics analysis--based code compression and one-cycle, decompression engine for transfer triggered architecture[J]. Acta Electronica Sinica, 2008, 36(11): 2234 - 2238. (in Chinese)

[9] 杨乾明.多核流体系结构模拟器研究与实现[D].湖南长沙:国防科学技术大学,2008,11.
Yang Qianming. Research and implement of simulator based on multi-core stream architecture[D]. Changsha, Hunan: National University of Defense Technology, 2008, 11. (in Chinese)

[10] Talal Bonny, Jorg Henkel. LICT: Left-uncompressed instructions compression technique to improve the decoding performance of VLIW processors[A]. Proceedings of the 46th

ACM/IEEE Design Automation Conference[C]. San Francisco, USA: ACM Press, 2009. 903 – 906.

[11] Zhiguo Ge, et al. A DVS-based pipelined reconfigurable instruction memory[A]. Proceedings of the 46th ACM/IEEE Design Automation Conference[C]. San Francisco, USA: ACM Press, 2009. 897 – 902.

[12] Stefan Metzlaff, et al. A dynamic instruction scratchpad memory for embedded processors managed by hardware[A]. Proceedings of the 24th International Conference on Architecture of Computing Systems[C]. Lecture Notes of Computer Science 6566, 2011. 122 – 134.

[13] Brucek Khailany. The VLSI implementation and evaluation of area and energy efficient streaming media processors[D]. Palo Alto, CA: Stanford University, 2003.

[14] David Black-Schaffer, et al. Hierarchical instruction register organization[J]. IEEE Computer Architecture Letters, 2008, 7(2): 41 – 44.

[15] He Yi, et al. Software managed instruction scratchpad memory optimization in stream architecture on hot code analysis of kernels[A]. Proceedings of the 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools[C]. Lille, France: IEEE Computer Society, 2010. 823 – 830.

[16] Wen Mei, et al. A parallel reed-solomon decoder on the imagine stream processor[A]. Proceedings of the Second International Symposium on Parallel and Distributed Processing and Applications[C]. Hongkong, China: Lecture Notes of Computer Science 3358, 2004, (12): 28 – 33.

[17] Matthew R Guthaus, et al. MiBench: a free, commercially representative embedded benchmark suite[A]. IEEE 4th Anmual Workshop on Workload Characterization[C]. Austin, USA: IEEE Computer Society, 2001. 3 – 14.

[18] GIMPS. The Math[EB/OL]. http://www.mersenne.org/math.htm, 2007.

[19] AHN, et al. Evaluating the imagine stream architecture[A]. Proceedings of the 31st Annual International Symposium on Computer Architecture[C]. Munich, Germany: IEEE Computer Society, 2004. 14 – 25.

作者简介

管茂林　男, 1983 年 6 月生, 江苏射阳人. 2007 年于国防科学技术大学获工学硕士学位, 现为国防科学技术大学计算机学院博士研究生. 主要研究方向: 高性能计算机体系结构与编译技术.

E-mail: gfkdgmlgml@163.com

何　义　男, 1982 年 6 月生, 湖北仙桃人. 2010 年于国防科学技术大学获工学博士学位, 现为北京油料研究所工程师. 主要研究方向: 高性能计算机体系结构、系统模拟与仿真等.

E-mail: grant.hey@gmail.com