一种支持同时多线程的 VLIW DSP 架构

沈 钰,孙义和
(清华大学微电子研究所,清华信息科学与技术国家实验室,北京 100084)

摘 要: 本文提出了一种支持同时多线程的动态分发超长指令字(VLIW)数字信号处理器(DSP)架构.该 DSP 架构上可以同时运行多个线程,功能单元可以执行来自多个线程的指令,有效地提高 DSP 的指令吞吐率.为了使多个线程的指令更有效地调度分发到功能单元,该 DSP 架构还支持指令动态分发,由硬件分发单元而不是编译器来完成多线程指令的动态分配.实验结果表明,相比于单线程而言,本文提出的 VLIW DSP 架构可以提高功能单元利用率,隐藏存储器访问时延,使处理器的指令吞吐率平均提高约 26.89%.

关键词: 同时多线程; 超长指令字; 数字信号处理器
中图分类号: TP302.1 **文献标识码:** A **文章编号:** 0372-2112 (2010) 02-0352-07

Architecture Design of Simultaneous Multithreading VLIW DSP

SHEN Zheng, SUN Yi-he
(*Institute of Microelectronics, Tsinghua University, Tsinghua National Laboratory of Information and Technology, Beijing* 100084)

Abstract: A novel simultaneous multithreading (SMT) VLIW DSP architecture with dynamic dispatch mechanism was presented. The SMT technology exploits the unused instruction slots by converting the thread – level parallelism to the instruction-level parallelism, improving the efficiency. With the dynamic dispatch mechanism, the processor issues instructions to functional unit at run-time rather than at compile-time, such that the issue conflicts among multiple threads are reduced significantly. The results show that the DSP architecture can effectively increase the functional unit utilization, hide the memory access latency, such that the processor throughput is improved by 26.89% with respect to single thread architecture.

Key words: simultaneous multithreading; VLIW architecture; digital signal processor

1 引言

超长指令字架构(very long instruction word,简称 VLIW)采用多个功能单元实现指令级并行,从而有效地提升数字信号处理性能[1],被广泛用于数字信号处理器(digital signal processor,简称 DSP)中.然而,由于缓存扑空导致的存储器访问时延以及单个程序中有限的指令并行度的限制,使得 VLIW 架构的计算资源无法被充分地利用.由于缓存扑空,在存储器访问时会产生数十周期的时延,处理器需要处于停顿状态来等待数据,这使得处理器中所有的功能单元都处于空闲等待而造成浪费.另一方面,单个应用程序中的指令并行度受到很多因素的限制.有研究表明,在硬件资源无限的情况下,单个程序的指令并行度仅能达到 7[2,3],而在资源受限的条件下,指令并行度只能达到 4 左右[4].单个应用程序中有限的指令并行度导致了 VLIW DSP 中的功能单元无法被充分地利用.

同时多线程(Simultaneous Multithreading,简称 SMT)技术可以克服上述因素导致的功能单元利用率低的问题.同时多线程技术在同一处理器上虚拟多个逻辑处理器,在每个逻辑处理器上运行一个程序线程,所有线程共享一套执行部件[5].同时多线程技术可以往那些空闲的功能单元发射其他线程的指令,从而将线程级并行度转换为指令级并行度,有效提高处理器的指令吞吐率,提高功能单元的利用率.同时多线程技术已经在采用超标量架构的通用处理器领域获得成功应用,并越来越受到研究人员的关注[6].

然而,VLIW 架构和同时多线程技术间的融合存在一些亟待解决的矛盾.顺序执行的 VLIW 架构本质上是一种独立架构[1].编译器认为在同一个执行包(execution-packet,简称 EP)中的指令之间不存在任何依赖关系,需要被同时发射并被同时执行.而在同时多线程技术中,为了充分地利用功能单元,处理器硬件需要有能力将一个 VLIW 指令执行包拆开,将同一执行包中的指令分多个时钟周期发射到相应的处于空闲状态的功能单元.而这和 VLIW 架构的设计初衷是相矛盾的,将同

收稿日期:2009-01-20;修回日期:2009-09-17

一执行包中的指令分多个周期发射会在指令之间引入额外的数据依赖关系,从而导致的执行结果错误.这就需要引入一些额外的机制来维持原有的数据依赖关系不变.

另一方面,即使同时多线程 VLIW 架构可以将多个线程的指令分多个周期分发到相应的功能单元,现有 VLIW 架构的静态分发机制还是会限制指令分发的灵活性而导致线程之间产生大量的分发冲突.在 VLIW 架构中,编译器在编译阶段已经将指令分配给相应的功能单元[14].假设这么一种情况,两个线程在同一周期将加法指令分发给功能单元 *A*,即使此刻另一个可以执行加法指令的功能单元 *B* 正处于空闲状态,但由于指令分配在编译阶段已经完成,两个线程的加法指令都被分发给 *A* 单元,并产生一次分发冲突.其中一个线程需要停止发射加法指令直到 *A* 单元处于空闲状态.由于这种分发冲突是在多线程程序运行的时候产生的,所以无法通过编译器在编译阶段解决.同时多线程架构执行过程中的分发冲突是不可避免的.分发冲突限制了吞吐率的提高,需要一些机制尽可能减少其产生.

为了解决上述提到的问题,本文提出了一种支持同时多线程的 VLIW 数字信号处理器架构.该 VLIW DSP 架构有以下特点

(1)该 VLIW DSP 架构引入了一组专门的寄存器用于维持同时多线程架构中原有的数据依赖关系.同一执行包内部的指令可以分多个时钟周期发射到相应的功能单元而不影响指令的执行结果.

(2)该 VLIW DSP 架构采用动态分发机制来降低多线程指令分发冲突.由硬件分发单元在执行阶段动态地完成指令分配,可以将指令分发到任何可以执行该指令且处于空闲状态的功能单元.

为了验证并评估该架构的效率,基于该架构,论文设计并实现了一个采用动态分发机制的双线程 VLIW DSP.实验结果证明了本文提出的架构可以提高 VLIW DSP 的功能单元利用率,隐藏存储器访问时延,提高指令吞吐率,从而有效提升数字信号处理性能.

2　相关工作

Kaxiras 等人[7]在 StarCore VLIW DSP 平台上研究实现同时多线程技术,而 Stephan 等人[8]则是将多线程技术应用于 Trimedia DSP 架构.在这两个 VLIW DSP 架构中,在同一时钟周期,处理器的只能分发来自一个线程的指令.当一个线程由于缓存扑空或中断而停顿时,另一个线程的指令才能获得执行.这种多线程技术通常被称为块多线程(block multithreading,简称 BMT).

Bharath Iyer 等人[9]提出了一种扩展拆分发射(extended split issue,简称 XSI)机制,在 VLIW DSP 上实现同时多线程技术.XSI 机制将执行过程拆分成两级:*E*1、*E*2.在 *E*1 级中,执行结果暂时写入一个延时寄存器堆,而在整个执行包分发结束时由 *E*2 级将延时寄存器堆中的结果写入架构寄存器堆中.XSI 机制维持了原有的数据依赖关系,使得执行包中的指令可以分多个周期进行分发.但是由于改变了执行过程,而改变了原有处理器的流水线结构,同时也改变了功能单元的执行时延.另一方面,由于仍然采用静态分发机制,硬件分发单元只能根据编译器分配的结果将指令分发到功能单元,而无法减少分发冲突.

3　同时多线程 VLIW DSP 架构

3.1　同时多线程技术

在同时多线程架构中,在同一处理器上虚拟多个逻辑处理器,每个逻辑处理器上运行一个程序线程,但它们同时共享一套执行部件.编译器可以将同时多线程处理器作为多个虚拟的处理器进行处理,而并不需要专门对应用程序进行调整.针对单线程处理器开发的应用代码也不需要进行任何改动就可以直接移植到同时多线程处理器中.

```
procedure foreground thread
begin
  for each execution-packet do
    for each instuctions in the execution packet do
    begin
    dispatch to any functional unit can execute this
    instrution;
    turn this fuctional unit state to busy;
    end
end

procedure background thread
begin
  for each exection-packet do
    for each functional unit can execute this instruction do
    begin
      if this functional unit state is a vailable then
      begin
        dispatch instruction to this functional unit turn this
        functional unit state to busy
      end
      else
        check next functional unit
    end
end
```

图 1　同时多线程算法伪码

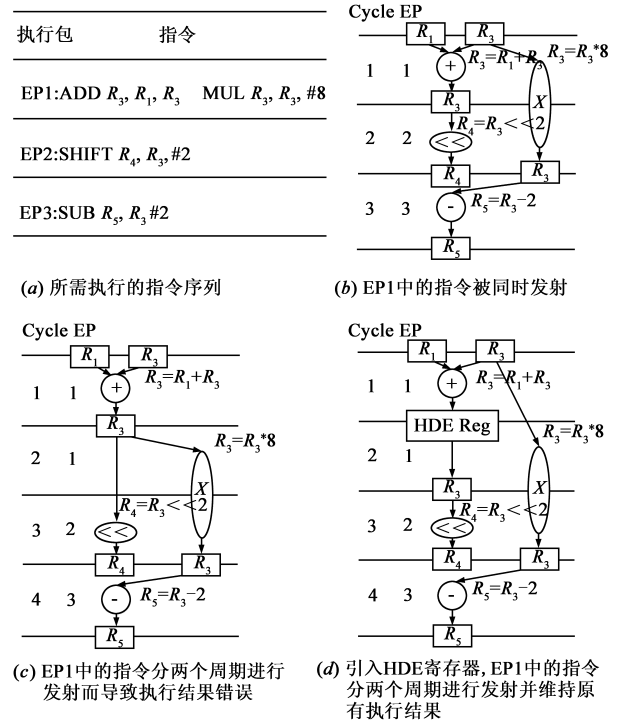
除了共享执行单元之外,同时多线程处理器需要保持所有线程的执行状态,而需要复制通用寄存器堆,

控制寄存器堆以及其他的控制部件.其他主要的执行部件,如功能单元,指令缓存以及数据缓存都可以被多个线程共享.每个周期,指令获取单元都可以从一个线程获取一个指令包,每个线程的指令被存放在各自的指令池中,取指单元会根据各个线程指令池的填充情况动态地选择获取各个线程的指令.分发单元将各个指令池中的指令分发到相应的功能单元,分发出去的指令都会被打上一个线程标签.功能单元根据线程标签将执行结果存放到相应线程的寄存器堆.

在实时数字信号处理应用中,虽然整个系统的吞吐率是衡量系统整体性能的一个指标,但是实时系统更加关注单任务执行性能.在本文提出的架构中,为所有的线程分配了不同级别的优先级,具有最高优先级的线程被称为前台线程.图 1 描述了在该架构中实现的同时多线程算法伪码,该算法中,前台线程可以访问所有功能单元,使用所有计算资源.和在单线程处理器中运行一样,前台线程可以达到最高性能.具有较低优先级的线程被称为后台线程.后台线程可以使用那些没有被更高优先级线程占用的功能单元,只有当执行该指令的功能单元处于空闲状态的时候,后台线程的指令才会被发射.为了有效地填充空闲的功能单元,需要将后台线程的执行包拆分成多个周期发射.

然而,在 VLIW 架构中,同一执行包中的指令是相互独立、互不依赖的,需要被同时发射并执行[1].而在同时多线程技术中,为了充分利用空闲的功能单元,处理器硬件需要根据功能单元的占用情况,将一个 VLIW 执行包中的指令分多个时钟周期发射到空闲的功能单元.这可能会引入额外的数据依赖关系,而导致错误的执行结果.

图 2 描述了在 VLIW 架构中一个指令序列的执行情况,其中,ADD,SUB 以及 SHIFT 指令的时延是 1 个时钟周期,MUL 指令的时延是 2 个时钟周期.执行包 1(图中标注为 EP1)包含 2 条指令:ADD 和 MUL.图 2(*b*)描述了在单线程 VLIW 架构中指令的执行情况,在时钟周期 1,EP1 中的 ADD 和 MUL 指令被同时发射,ADD 和 MUL之间不存在依赖关系.然而,如图 2(*c*)所示,在多线程架构中,假如 EP1 进行分发的时候,乘法单元已经被更高优先级的线程所占用而此刻加法单元处于空闲状态,那么 EP1 就会被拆分成两个周期发射:周期 1 发射 ADD,周期 2 发射 MUL.在周期 2,当 MUL 从 R_3 获取操作数时,就会得到周期 1 ADD 产生的结果数据,这并不是程序所期望的.拆分 EP1 之后,EP1 内部指令之间的依赖关系就被引入到执行过程中,从而导致错误的执行结果.这种同一执行包中内部指令之间的依赖关系被称为水平依赖,而不同执行包之间的依赖关系则被称为是垂直依赖.为了可以拆分执行包并同时维持原有的依赖关系,本文提出的架构引入一组水平依赖消除(horizontal dependency elimination 简称 HDE)寄存器.图 2(*d*)描述了引入 HDE 寄存器的多线程 VLIW 架构中指令的执行情况.在周期 1,由于 ADD 所在执行包 EP1 并没有被发射完全,ADD 的执行结果写入 HDE 寄存器中,而并不更新 R_3 寄存器中的数据.那么,在周期 2,同一执行包中的 MUL 操作就能获取正确的操作数.周期 2,EP1 中的所有指令都已经被分发时,HDE 寄存器中暂存的执行结果就可以写入 R_3 寄存器.由于引入了 HDE 寄存器,同一个执行包中的指令可以分多个周期进行发射而不会影响程序的依赖关系.



(*a*) 所需执行的指令序列 (*b*) EP1中的指令被同时发射

(*c*) EP1中的指令分两个周期进行发射而导致执行结果错误 (*d*) 引入HDE寄存器,EP1中的指令分两个周期进行发射并维持原有执行结果

图2 VLIW架构中指令的执行情况

图 3 描述了 HDE 寄存器在电路中的实现.如果执行包中还有指令未被分发,功能单元先将指令执行结果暂存在 HDE 寄存器.而当执行包中的所有指令都被分发完成,HDE 寄存器中的执行结果被写入到架构寄存器堆中.如果功能单元所执行的指令为其所在执行包中最后一条分发的指令,那么当这条指令执行完成时,功能单元通过旁路路径将其执行结果直接写到寄存器堆中.这样在不改变流水线的结构同时也不会改变指令执行时延.由于 HDE 寄存器并不位于处理器的关键路径,而只需要在处理器数据通道中增加了一个二选一选择器,HDE 寄存器的实现并不影响处理器的工作频率.在 VLIW 架构中,功能单元的时延对于编译器是可见的,如果功能单元的执行级采用了多个流水级实现,那么每个功能单元的每个执行流水级都需要一个 HDE 寄存器用于暂存执行结果.

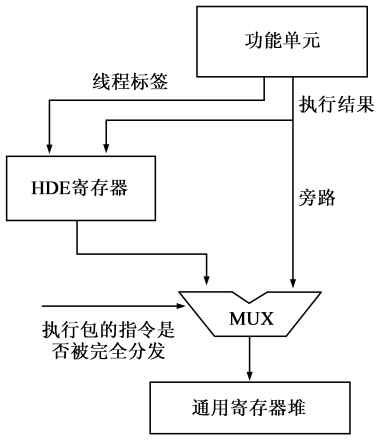
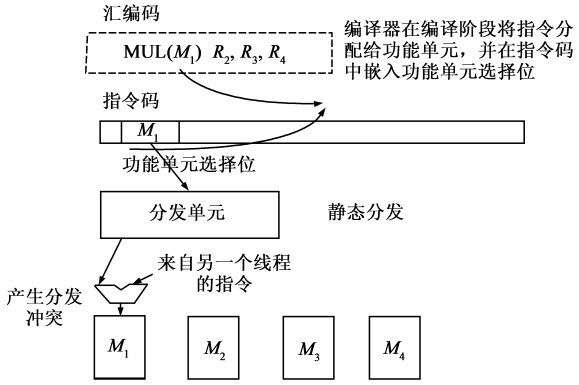


图3　HDE寄存器的电路实现

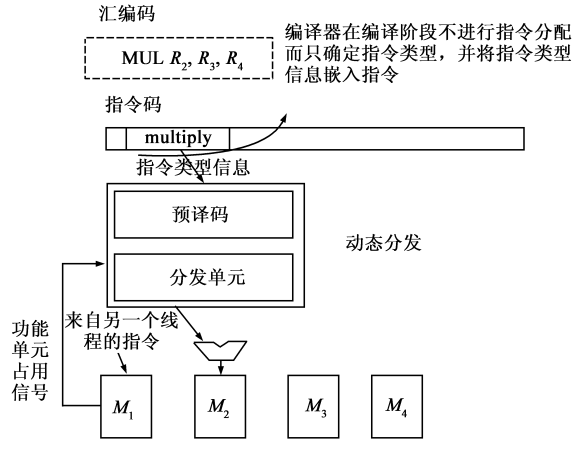
3.2　动态分发机制

当多个线程的指令同时分发给同一个功能单元时,在这一周期就会产生一个分发冲突.论文引入了分发冲突率(issue conflict rate,简称 ICR)对分发冲突进行量化,计算方式如表达式(1)所示,式中 T_{conflict}是指在运行过程中有分发冲突产生的周期数,T_{total}指总的运行周期数.

$$\text{ICR} = T_{\text{conflict}} / T_{\text{total}} \tag{1}$$



(*a*) 静态分发机制



(*b*) 动态分发机制

图4　静态分发机制和动态分发机制

VLIW 编译器通常在编译阶段就将指令分配给功能单元,并在指令码中通过功能单元选择位嵌入功能单元分配信息.图 4 描述了两种分发机制之间的区别.在图 4(*a*)描述的静态分发机制中,根据指令码中的功能单元选择位,分发单元将该 MUL 指令分发给 *M*1 单元.即使在架构中存在多个可以执行该指令且处于空闲状态的 *M* 单元,由于编译器在编译过程中已经完成指令分配,该 MUL 指令只能分配给 *M*1 执行.如果此刻 *M*1 单元被具有更高优先级的线程占据,那么就会产生一次分发冲突,分发单元只能停止分发 MUL 指令直到 *M*1 单元处于空闲状态.图 4(*b*)描述了本文架构所支持的动态分发机制.在动态分发机制中,编译器不再负责指令和功能单元之间的分配而是将指令根据指令类型进行编码分类.分发单元根据从指令码中提取的指令类型来分发指令.在图 4(*b*)示例中,分发单元通过对指令操作码进行预解码,判断该指令是 MUL 指令,可以由 M 单元执行.那么,即使 *M*1 单元被其他线程占用,分发单元也可以将该指令分发到其他 *M* 单元.只有当所有 *M* 单元都被更高优先级线程占用时,才会产生一次分发冲突.动态分发机制可以显著的降低分发冲突率,从而更有效地提高功能单元利用率,提高指令吞吐率.

4　设计实现

基于本文提出 DSP 架构,论文设计并实现了一款支持同时多线程的 VLIW DSP.该 DSP 基于一个 6 发射,32 位定点,10 级流水线的 VLIW 架构,支持两个线程同时运行.图 5 描述了该 DSP 架构的框图,该处理器具有 6 个功能单元,包括 2 个 *A* 单元,2 个 *M* 单元和 2 个 *D* 单元.其中,*A* 单元执行所有算术逻辑指令以及所有移位/旋转指令;*M* 单元执行所有乘法指令以及部分算术指令;*D* 单元负责执行所有存储器访问操作,流程控制指令,比较指令以及部分算术指令.

采用了动态分发机制,该 DSP 的分发单元可以根据指令类型将指令动态分发到相应的功能单元.图 6 列举了该 DSP 所支持的指令类型以及指令的执行时延.根据执行指令的功能单元类型可以将所有指令分成 4 类:(1)可以被所有功能单元执行;(2)只可以被 *A* 单元执行;(3)只可以被 *M* 单元执行;(4)只可以被 *D* 单元执行.

架构中主要的执行部件,如功能单元,指令缓存以及数据缓存被两个线程共享.而为了保持两个线程的执行状态,需要复制该 DSP 的某些架构部件.为了储存来自两个线程的指令,架构中包含了两个指令池,取指单元将不同线程的指令存储在各个指令池中.为了维护每个线程的控制流程,架构中包含了两套程序计数器(Program Counter,简称 PC)控制单元.而为了保持线程的计算结果,该架构中包含了两份通用寄存器堆以及控制寄存器堆.

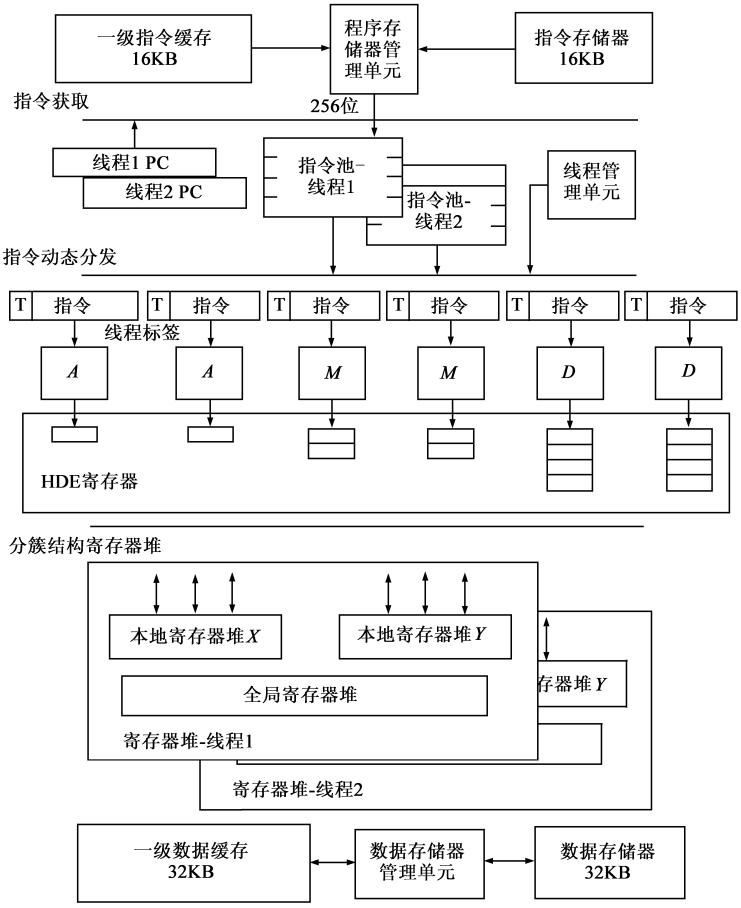


图5 DSP架构框图

指令类型	执行功能单元	指令时延（周期）	分发类型
Integer/Logic ALU	*A*, *M*, *D*	1	*X*
Bit Permutation	*A*, *M*, *D*	1	*X*
NOP	*A*, *M*, *D*	1	*X*
Shift/Rotate	*A*	1	*A*
Vector SUM	*A*	1	*A*
Multiply/Mac	*M*	2	*M*
Dot Product	*M*	2	*M*
Compare	*D*	1	*D*
Load/Store	*D*	4	*D*
Push/Pop	*D*	4	*D*
Branch/Jump	*D*	5	*D*
Call/Return	*D*	5	*D*D

图 6 指令类型及指令时延

图 7 给出了该 DSP 中各部分的硬件开销所占比重. 采用 0.13μm 工艺库进行综合的结果表明,在最坏情况下,该 DSP 频率可以达到 200MHz,而工作于峰值频率时的功耗约为 70mW. 该 DSP 的规模约为 35 万等效二输入与非门,在其所有硬件开销中,功能单元逻辑占了约 22.9%,缓存逻辑占了约 48.6%. 而约有 12.8%的硬件资源被用于支持同时双线程,其中包括寄存器堆,HDE 寄存器,以及同时多线程控制逻辑.

5 实验结果及分析

5.1 评估实验平台

为了评估本文提出的同时多线程技术对 VLIW DSP 指令吞吐率的影响,设计并采用了数字信号处理基准程序对该 DSP 架构进行评估. 论文将仿真器配置成不同的架构模型,其中包括(1)基本的单线程架构模型;(2)上文第二节中所描述的块多线程(BMT)架构[7,8];c)扩展拆分发射(XSI)[9]架构. 其中 BMT 模型和 XSI 模型都配置成支持两个同时多线程

论文所采用的评估程序包括数字信号处理基准算法以及一些媒体应用程序. 数字信号处理算法选自 BDTI DSP 基准程序包[10],而媒体应用程序则选择部分常用音视频编解码程序[11,12]. 为了保证程序执行性能,所有程序都采用手写汇编优化,尽可能提高程序的指令级并行度,提高程序的执行效率.

5.2 提高处理器指令吞吐率

本文采用每周期执行指令数(Instruction per cycle,简称 IPC)对提出的同时多线程 DSP 架构以及其他架构模型对指令吞吐率的影响进行比较(如图 8). 与基本的单线程 DSP 架构相比,所有双线程模型都有利于 IPC 百分比的提升. 在只访问片内存储器的情况下,如果程序的指令级并行度越低,同时多线程技术可以获得的 IPC 提升百分比越高. 而在访问片内缓存的情况下,由于多线程可以有效的隐藏存储访问时延,同时多线程技术可以更显著地提升 IPC. 如图 8 所示,在只访问片内缓存的情况下,相比于单线程架构,可使数字信号处理程序的 IPC 平均提高约 26.89%.

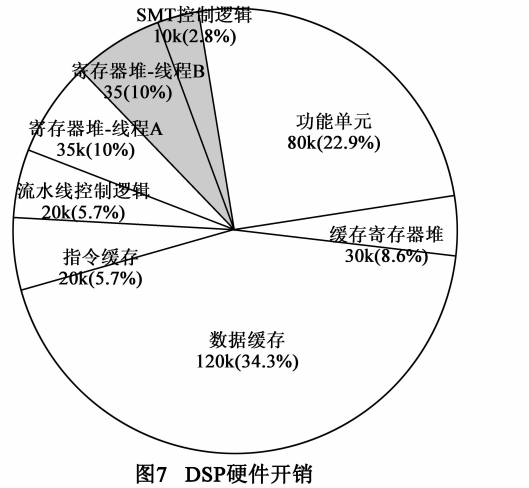


图7 DSP硬件开销

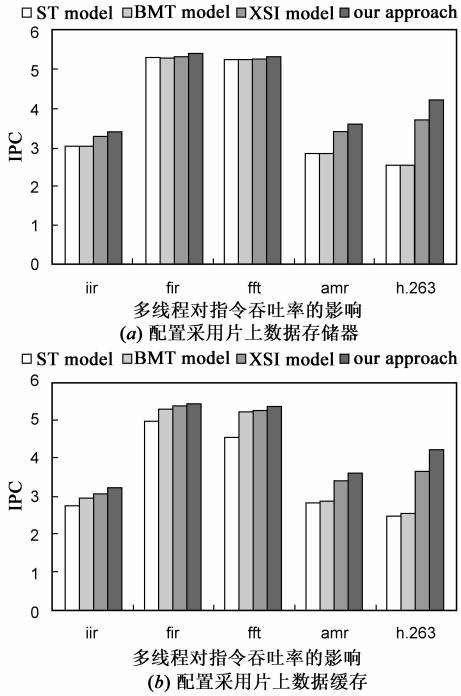


图8　执行数字信号处理程序的吞吐率比较

5.3　减少分发冲突

采用动态分发机制,分发单元可以在指令执行期间动态地将指令分发到功能单元.指令可以由任意一个可以执行该指令且处于空闲状态的功能单元执行.图 9 描述了本文提出的采用动态分发机制的同时多线程 DSP 架构与采用静态分发机制的 XSI 架构之间分发冲突率的比较.从图中可以发现,分发冲突率随着程序指令并行度的提高而增加.采用动态分发机制可以将将数字信号处理程序执行过程的分发冲突减少约 34.60%.动态分发机制给分发单元提供了更多自由度,分发单元可以根据指令类型进行指令分发,降低了分发冲突率,从而可以进一步提高处理器的吞吐率.

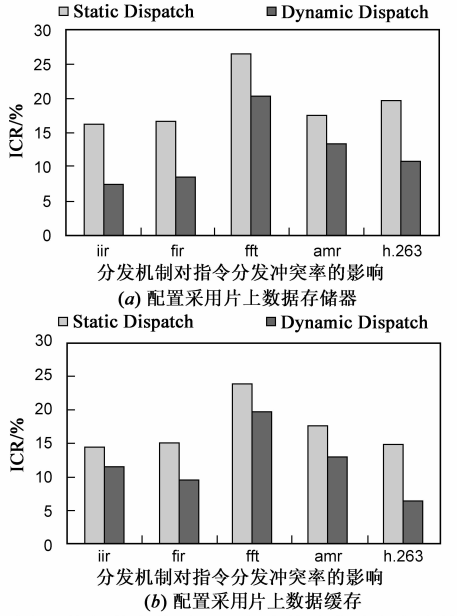


图9　执行数字信号处理程序的分发冲突率比较

6　结论

本论文提出了一种支持同时多线程的动态分发 VLIW DSP 架构.该 DSP 架构引入了一组 HDE 寄存器用于消除由于执行包拆分引入的执行包内部指令之间的依赖关系,而使 DSP 硬件可以将同一执行包内部的指令分多个时钟周期发射到相应的功能单元,并维持指令执行结果.为了降低多线程之间的分发冲突,该 DSP 架构还采用动态分发机制,由硬件分发单元而不是编译器来完成指令的分配.基于该 DSP 架构,论文设计并实现了一款支持 2 个同时多线程的 VLIW DSP.采用 0.13μm 工艺库进行综合的结果表明,在最坏情况下,该 DSP 可以工作于 200MHz,而工作于峰值频率时的功耗约为 70mW.该 DSP 的规模约为 35 万等效二输入与非门,在所有硬件开销中,仅需要 12.8% 的硬件资源用于支持同时双线程.实验结果证明,针对数字信号处理,本文提出的双线程 DSP 架构可以将 IPC 提高约 26.89%,将分发冲突减少约 34.60%,从而有效的提高处理器性能.

参考文献:

[1] Rau B, Fisher J. Instruction-Level Parallel Processing: History, Overview, and Perspective [J]. Journal of Supercomputing. 1993,7(21):9 – 50.

[2] Wall D W. Limits of instruction-level parallelism[A]. In Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems [C]. Santa Clara, United States, 1991, 26(4):176 – 189.

[3] Lam M S. Limits of control flow on parallelism[A]. In Proceedings of the 18th International Symposium on Computer Architecture[C]. Queensland, Australia, 1992, 20(2):46 – 57.

[4] Butler M. Single instruction stream parallelism is greater than two[A]. In Proceedings of the 18th International Symposium on Computer Architecture[C]. Toronto, Canada, 1991. 276 – 286.

[5] Tullsen D M. Simultaneous multithreading: maximizing onchip parallelism[A]. In Proceedings of the 22nd Annual International Symposium on Computer Architecture[C]. Barcelona, Spain, 1995. 392 – 403 .

[6] Bayoumi M A. Parallel Algorithms and Architectures for DSP Applications[M]. Kluwer Academic Publishers, Norwell, USA, 1991. 352 – 369.

[7] Kaxiras S. Comparing Power Consumption of an SMT and a CMP DSP for Mobile Phone Workloads[A]. In Proceedings of

the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems[C]. Atlanta, USA, 2001. 211 – 220.

[8] Suijkerbuijk S, Juurlink. Implementing Hardware Multithreading in a VLIW Processor[A], In Proceedings of the 17th International Conference on Parallel and Distributed Computing and Systems[C]. Las Vegas, USA, 2005. 674 – 679.

[9] Bharath Iyer. Extended Split – Issue: Enabling Flexibility in the Hardware Implementation of NUAL VLIW DSPs[A]. In Proceedings of the 31nd Annual International Symposium on Computer Architecture[C]. Munchen, Germany, 2004, 32(2): 364 – 369.

[10] BDTI Benchmark Results [OL]. http://www. bdti. com, 2005.

[11] AMR Speech Codec [OL]. http://www. 3gpp. org/ftp/Specs/html-info/26-series. htm, 2009.

[12] The ITU-T specification for H. 263[OL]. http://www. itu. int/rec/T – REC – H. 263, 2009.

作者简介:

沈 钲 男, 1982 年生, 浙江省绍兴市人. 2000 年 9 月考入清华大学电子工程系电子科学与技术专业. 2004 年 7 月本科毕业并获得工学学士学位. 2009 年 7 月获得电子科学与技术专业工学博士学位. 主要研究方向: 可配置数字信号处理器结构设计, 并行处理器结构.

E-mail: shenzheng00@gmail. com

孙义和 男, 1945 年生, 安徽省合肥市人, 教授, 博士生导师, 中国电子学会高级会员. 1970 年清华大学自动控制系毕业, 后留校任教. 长期从事微电子学和超大规 模集成电路的教学、科学研究工作. 主要研究方向: LSI/VLSI 测试方法学和可测性设计, 多媒体 VLSI 设计技术, 网络和数据安全 VLSI 结构.

E-mail: sunyh@tsinghua. edu. cn