·体系结构与软件技术· 文章编号：1000-3428(2017)04-0046-06 文献标志码：A 中图分类号：TP332

浮点与整数资源区别分配的 SMT 处理器取指策略

蒋生健，胡向东，杨剑新

（国家高性能集成电路（上海）设计中心，上海 201204）

摘 要： 在同时多线程处理器中，各线程对于浮点和整数资源需求不同，合理分配线程的共享资源是提升处理器整体性能的重要因素。为此，提出一种浮点与整数资源区别分配的取指策略，合理分配各个线程对于浮点和整数资源的使用情况。实验结果表明，与 ICOUNT，STALL 等策略相比，该策略在算术平均 IPC 和调和平均 IPC 方面均取得一定的性能提升，同时其在处理浮点和整数混合型程序时也具有优势。

关键词： 同时多线程；取指策略；资源分配；线程级并行；浮点；整数

中文引用格式： 蒋生健，胡向东，杨剑新. 浮点与整数资源区别分配的 SMT 处理器取指策略[J]. 计算机工程，2017，43(4)：46-51.

英文引用格式： Jiang Shengjian，Hu Xiangdong，Yang Jianxin. Instruction Fetch Policy for SMT Processors with Different Allocations of Floating-point and Integer Resources[J]. Computer Engineering，2017，43(4)：46-51.

Instruction Fetch Policy for SMT Processors with Different Allocations of Floating-point and Integer Resources

JIANG Shengjian，HU Xiangdong，YANG Jianxin

（National High Performance Integrated Circuit Design Center，Shanghai 201204，China）

【Abstract】 In Simultaneous Multithreading（SMT） processors，different threads have different demands for floating-point and integer resources. How to allocate shared resources among threads is the key point to improve the whole performance for SMT processors. Aiming at this problem，this paper proposes an instruction fetch policy with different allocations of floating-point and integer resources，which reasonably allocates the usage of each thread for floating-point and integer resources. Experimental results show that，compared with ICOUNT，STALL policy，etc.，the proposed policy improves performance when using average IPC and harmonic average IPC as a metric. Meanwhile，it also has advantages when processing the programs mixing floating-point and integer.

【Key words】 Simultaneous Multithreading（SMT）；instruction fetch policy；resource allocation；thread-level parallelism；floating-point；integer

DOI： 10.3969/j.issn.1000-3428.2017.04.008

0 概述

同时多线程（Simultaneous Multithreading，SMT）技术[1]允许同时发射多个线程的指令到执行部件，可减少发射槽的水平浪费和垂直浪费，提升处理器的资源利用率和指令吞吐率。但 SMT 处理器各线程在共享处理器资源的同时也带来了资源的竞争，因此，合理分配共享资源对提升 SMT 处理器性能至关重要。

取指部件每周期从一个或多个线程中读取若干条指令进入流水线，从源头上决定了共享资源在线程间的分配。在 SMT 处理器中，如果某个线程发生长延时事件，如二级 Cache 缺失、数据 TLB（Translation Lookaside Buffer）缺失等，可能导致该线程长时间占用某种共享资源，影响其他线程的性能，甚至造成流水线的停顿。针对长延时事件，许多取指策略被提出，如 L2MP[2]，MLPAFP[3]，RUCOUNT[4] 等。有些取指策略直接对共享资源进行分配控制，如 MFP[5]，DCRA[6]，Hill-Climbing[7] 等，但是这些取指策略都没有考虑各线程对于浮点和整数资源需求的区别。

针对上述问题，本文提出一种浮点与整数资源区别分配的取指策略 DAFIR。在资源分配时，首先

基金项目： “核高基”重大专项“超级计算机处理器研发”（2013ZX01028-001-001-001）。

作者简介： 蒋生健（1989—），男，硕士研究生，主研方向为计算机体系结构、高性能微处理器设计；胡向东、杨剑新，高级工程师。

收稿日期： 2016-04-12 **修回日期：** 2016-05-12 **E-mail：** jiangsj09@163.com

根据各线程访问 Cache 的情况设定其资源优先级,资源优先级越高,所能分配的资源使用上限就越高。此外,在执行过程中,如果识别到线程为浮点型程序,则对使用的浮点和整数资源上限进行调整。DAFIR 通过合理分配线程对于浮点与整数资源的使用情况,提高资源的使用效率和处理器的整体性能。

1　相关研究

目前,学术界对 SMT 取指策略进行了广泛而深入的研究。本节主要介绍针对长延时事件和资源控制方面的典型取指策略。

ICOUNT 取指策略[8]在每个周期统计各个线程在译码、重命名阶段和发射队列中的指令数量,指令数量越少,取指优先级越高。ICOUNT 是最经典的取指策略之一,下述几种取指策略都基于 ICOUNT 实现。

STALL[9]和 FLUSH 取指策略直接对产生长延时事件的线程进行控制。这两种取指策略检测 Load 指令访问二级 Cache 的时间,如果超过阈值,则认为产生了长延时事件。在线程产生长延时事件时,STALL 策略取消该线程的取指权限,而 FLUSH 策略则在 STALL 的基础上清除该线程在流水线上的部分指令。

如果一条 Load 指令访问二级 Cache 缺失,那么必然有访问一级数据 Cache 缺失。文献[10]根据这一特点,利用 Load 指令访问一级数据 Cache 缺失来实现 DG(Data Miss Gating)和 PDG(Predictive Data Miss Gating)取指策略。DG 取指策略为每个线程设置一个缺失计数器,统计该线程当前有多少条指令访问一级数据 Cache 缺失。取指部件在取指之前查询每个线程的缺失计数器,当计数器的值大于某一阈值时则取消该线程的取指权限。另外,为了尽早知道一条 Load 指令是否会产生一级数据 Cache 缺失,PDG 取指策略采用了 Load 指令访问一级数据 Cache 缺失预测器。计数器的值在实际缺失或者预测缺失的情况都会增加,在该 Load 提交时减少。当计数器的值大于阈值时则停止该线程的取指。

DWarn[11]也是一种根据一级数据 Cache 缺失情况来实现的取指策略。在 ICOUNT 的基础上,首先从没有产生一级数据 Cache 缺失的线程中进行取指,当无法满足取指带宽时,再选择有一级数据 Cache 缺失的线程进行取指。DWarn +[12]取指策略在 DWarn 基础上区分产生访问二级 Cache 缺失的线程和只有一级数据 Cache 缺失的线程,同时控制线程使用资源的最大值。

文献[13]研究表明,对于整数和浮点混合的程序,可以通过更好的调度策略来减少功能单元的空闲,从而实现更好的性能。因此,FIRSFP 取指策略被提出。FIRSFP 检测取指、译码、重命名和发射队列中浮点和整数指令之间的比例关系,同时分析运行线程的整数和浮点指令的频率关系,确定下一个周期选择哪个线程能够使整数和浮点功能单元保持较高的使用效率。但 FIRSFP 没有考虑线程对于浮点和整数共享资源的使用情况,如浮点指令队列、整数指令队列以及浮点与整数物理寄存器等资源。

MFP 取指策略根据线程访问一级数据 Cache 和二级 Cache 缺失的情况把线程分为 3 个优先级,属于同一个优先级内的线程按照 ICOUNT 的顺序取指。同时 MFP 根据各个线程的优先级,对线程使用的共享资源设置阈值,如式(1)所示。

$$R_i = N \times \frac{RL_i}{\sum RL_j} \tag{1}$$

其中,R_i为线程 i 可以使用的某资源上限;N 为该资源的总量;RL_i表示线程 i 的资源优先级。如果线程使用的资源超过阈值,则停止取指。

DCRA 分配资源基于 2 个出发点:一是资源需求型线程必须限制资源的最大使用量,防止资源被完全独占;二是为了使资源需求型线程充分利用指令级并行性,应当在没有其他线程需要资源时,尽量提供多的资源。Hill-Climbing 通过若干周期的时间片方式监测线程性能,总是往性能更好的方向分配资源。ARPA[14]也利用了时间片的概念,计算时间片内性能与所分配资源数量的比值,比值越高表示资源的使用效率越高,则分配更多的资源给该线程。

在上述取指策略中,只有 FIRSFP 取指策略利用浮点与整数程序进行性能优化,但也只考虑了浮点与整数功能部件的使用率,没有对其他共享资源进行控制。另外,在直接控制各个线程资源使用上限的策略中,都是统一分配所有的资源,没有考虑不同线程对于浮点和整数资源需求的差异性。不同线程对于浮点和整数资源的需求不同,同一个线程在不同执行阶段对于浮点与整数资源的需求也不同。

2　DAFIR 取指策略

DAFIR 取指策略的目标主要有 3 个:1)相对于 ICOUNT 简化线程优先级的排序方法;2)减少发生 Cache 缺失的线程对于共享资源的使用权限;3)根据线程是否为浮点型程序,区别对待浮点与整数资源的分配,提高资源的利用率。

2.1　线程取指优先级排序

对于实际处理器的取指部件而言,从取指地址仲裁,到最后取到指令送入线程私有的取指缓冲中,这中间需要若干周期。本文实验中取指共需要 4 个时钟周期。在这 4 个时钟周期中,每周期都会发出某个线程的取指请求,而 ICOUNT 取指策略并没有考虑这部分指令的影响。当指令数据从指令 Cache 中返回以后,将指令数据缓存在可容纳 4 个指令组(每个指令组最多 4 条指令)的取指缓冲中,供指令

译码部件使用。

另一个方面,执行站台前各个流水段的指令数量对取指部件而言作用并不一样。一般情况下,离取指站台越远则作用越小。因为距离越远,指令到达相应站台所需要的时间通常越久,对于取指部件来说越难控制。而且前面站台的指令受到后续站台执行情况的影响,前面几个站台已经能大致反映后续站台的指令通过情况。如果后续某个站台发生某个线程阻塞的情况,则该线程前面站台的缓冲将会产生指令堆积。

与 ICOUNT 取指策略不同,DAFIR 只使用取指阶段发出取指请求但数据还未返回的指令数量和取指缓冲中的指令数量(属于译码阶段)进行线程取指优先级的排序。

2.2 线程资源优先级设定

共享资源分配方面,本文在 MFP 取指策略的基础上进行改进。根据线程访问一级数据 Cache 和二级 Cache 缺失情况,为每个线程设置资源优先级(Resource Priority,RP)。线程 RP 可取值为 1,2,3 中的一个,用来设置线程使用共享资源的阈值。初始化时每个线程的资源 RP 都设为 1;发生访问一级数据 Cache 时 RP 设为 3;线程进一步发生访问二级 Cache 缺失时设 RP 为 2。线程资源优先级的转换可以用一个有限状态机实现,如图 1 所示。

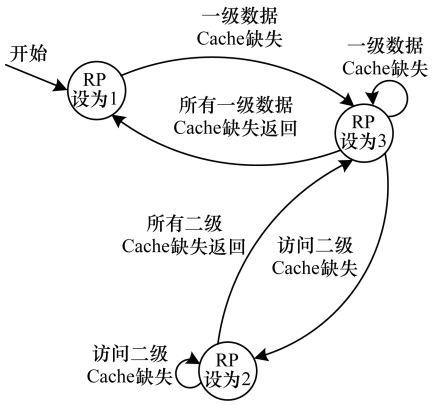

图 1 资源优先级状态机示意图

2.3 浮点与整数资源的区别分配

在 MFP 取指策略中,直接根据各个线程的取指优先级确定各个线程访问共享资源的上限。当线程没有发生 Cache 缺失时(RP 设为 1),使用资源不受限制;如果发生 Cache 缺失(RP 设为 2 或 3),则根据式(1)计算资源使用上限。DAFIR 取指策略对此做以下2 点改进:

1)识别一个线程是否属于浮点类型的程序。借鉴 Hill-Climbing 和 ARPA 使用时间片的方法,DAFIR 取指策略在执行过程中判断线程属于整数型程序还是浮点型程序。在每个时间片里,统计浮点指令分派队列 FQ 中指令数量之和,如果超过一定阈值,则认为该线程属于浮点型程序,否则属于整数型程序。在实验中,同时两线程(SMT2)模式下某个线程 FQ 平均使用条目数大于 1 表示为浮点型程序,同时四线程(SMT4)模式下大于 0.5 认为是浮点型程序。

2)区别对待整数型程序和浮点型程序的资源分配。整数型程序和浮点型程序对不同类型的共享资源的需求存在差别。DAFIR 取指策略根据前一个时间片的结果,判断每个线程属于整数型程序还是浮点型程序。如果所有线程都是整数型程序,则所有整数型资源按照 RP 来分配(见式(1),其中 *RL* 相当于 RP),浮点型资源的使用不设限制。如果所有线程都是浮点型程序,则所有整数型和浮点型资源都按照 RP 分配。如果整数型程序和浮点型程序都有,则整数型程序不进行浮点型资源的限制,整数资源在 RP 分配所得的基础上增加一部分使用额度(资源总量为 64 时增加 8,总量为 32 时增加 4),这部分额度可以认为是从浮点型程序处借用而来;而浮点型程序对于整数型资源在按照资源优先级 RP 分配的基础上减去一部分使用额度(资源总量为 64 时减 8,总量为 32 时减 4);对于浮点型资源,则在所有浮点型程序之间根据 RP 进行分配。

取指部件根据各个线程取指和译码阶段的指令数量进行线程选择,指令数量越少的优先级越高。如果高优先级的线程某项资源使用超过上限,则取消该线程本周期的取指权限,选择优先级更低的线程进行取指。

2.4 策略实现

对每个线程设置一个 5 位的计数器,记录每个线程已经发出取指但数据还未返回的指令数量。每个线程设置一个 5 位的计数器,记录每个线程取指缓冲中的指令数量。分别用一个加法器把 2 个数据相加保存到一个 6 位的计数器中,用于线程的选择。计数器在分配相应缓冲条目时增加,释放条目时减少。

实验结果表明,时间片的长度设置为 16k 个时钟周期较为合适,因此,设置一个 14 位的计数器用来实现时间片的功能。为每个线程设置一个 15 位的计数器,记录每个线程在时间片内使用的 FQ 条目数之和。实验中,在 SMT2 模式下,FQ 平均使用率大于 1 则表示为浮点型应用,所以,SMT2 时该计数器最高位为 1 时表示为浮点型应用,计数器可以停止增加。对于 SMT4 时平均使用率大于 0.5 个条目即可确认类型,所以,次高位为 1 即可停止增加。计数器在时间片计数器全 1 时清零重新计数。

为每个线程设置 2 位标志位表示该线程是否产生了访问一级数据 Cache 缺失和访问二级 Cache 缺失。Cache 缺失信息由 Cache 控制部件给出。

为每个线程设置 IQ,FQ,iReg,fReg 使用计数

器,根据各个资源的最大值设置计数器的位宽。另外,需要再设置一个相同的计数器用于记录每个线程的资源使用阈值。在模拟器中资源使用阈值可以通过实时计算得到,由于存在乘除运算,在实际的处理器芯片中不可能在一个时钟周期就得到结果,因此可以事先计算线程在不同状况下的资源使用上限,将其保存在一个表中,执行时根据线程状态直接查询即可。对于 SMT4 模式,表格的长度将会达到1 296 项(每个线程有 3 种 RP,属于整数或者浮点程序,共 6 种可能性,$6^4 = 1\ 296$)。可以对资源分配进行适当简化,例如不考虑每个线程分别是否为浮点型应用,只考虑是否有浮点型应用和本线程是否为浮点型应用,即可将表格长度缩小为 1/4(324 项)。

取指部件在每个周期根据指令计数器和该线程是否已经超出某项共享资源的使用上限来选择线程进行取指。

3　实验设置

3.1　模拟器

本文采用 XSim 性能模拟器进行实验模拟。XSim 基于 X 处理器(某国产处理器)核心开发的事件驱动型性能模拟器,支持同时多线程技术。XSim 为 X 处理器研发同时多线程技术提供了研究平台,并且在单线程模式下基于实际处理器核心进行性能校准,可以大致反映真实芯片的性能,实验结果具有较大的参考价值。表 1 列出了模拟器的基本配置。

表 1　模拟器基本配置

参数	配置
取指/译码/重命名/提交带宽	4 条指令/周期
发射带宽	7 条指令/周期
指令分派队列	整数:32,浮点:26
指令发射队列	9 个发射队列,每个队列 4 个条目
Load 指令队列	32 条目
Store 指令队列	16 条目
ROB 长度	128 条目
物理寄存器数量	整数:64,浮点:64
指令 Cache	32 KB,2 路组相连,128 Byte/行
一级数据 Cache	32 KB,4 路组相连,128 Byte/行
二级 Cache	512 KB,8 路组相连,128 Byte/行
三级 Cache	4 MB,16 路组相连,128 Byte/行
Cache 访问延迟	指令 Cache:4 周期 一级数据 Cache:4 周期 二级 Cache:11 周期 三级 Cache:45 周期
主存访问延时	120 周期

3.2　多线程程序组合

本文使用 SPEC CPU2000[15] 测试程序集进行实验评估。采用 ref 测试集,利用 X 处理器编译器进行编译。实验共随机选择 21 组 2 线程组合,8 组 4 线程组合,具体如下:

SMT2 测试程序组合(21 组):

gzip_equake,gcc_mcf,gcc_gzip,mesa_gzip
mesa_crafty,eon_equake,mcf_mcf,lucas_mcf
gcc_mgrid,mesa_art,twolf_crafty,ammp_vpr
perlbmk_galgel,mesa_twolf,bzip2_art,lucas_bzip2
mgrid_art,vpr_vpr,lucas_art,applu_twolf,galgel_art

SMT4 测试程序组合(8 组):

eon_mesa_ammp_gzip,gcc_eon_mesa_equake
gcc_mgrid_gzip_mcf,mesa_crafty_art_bzip2
mesa_galgel_twolf_vpr,eon_lucas_twolf_bzip2
applu_vpr_bzip2_art,lucas_twolf_vpr_art

执行程序时,首先为每个线程跳过开始的10^8 条指令,然后每个线程都至少模拟 10^8 条指令后停止执行,输出模拟结果。如果有的线程运行比较快,先达到 10^8 条指令,则继续执行,直到最慢的线程模拟完 10^8 条指令。

3.3　性能评价方法

IPC 表示每周期指令数(Instruction Per Cycle),本文采用算术平均 IPC 和调和平均 IPC[16] 2 种标准进行性能评价。

算术平均 IPC 是同时多线程处理器中所有线程 IPC 的平均值,如式(2)所示。

$$AverageIPC = \frac{\sum_{i}^{T} IPC_i}{T} \tag{2}$$

其中,IPC_i 为处理器在 SMT 模式下线程 i 的 IPC 值;T 表示线程数。

调和平均 IPC 不仅考虑了运行 SMT 时某个线程的 IPC 值,还考虑了该处理器在单线程模式下运行该程序的 IPC 值,如式(3)所示。

$$HarmonicIPC = \frac{T}{\sum_{i}^{T} \frac{SIPC_i}{IPC_i}} \tag{3}$$

其中,$SIPC_i$ 为该程序在单线程模式下的 IPC 值,其他参数与式(2)相同。

算术平均 IPC 反映了处理器的平均指令吞吐率,算术平均 IPC 越高,表示处理器一个周期内能处理的指令数量越多。调和平均 IPC 能反映程序在 SMT 处理器上的执行效率,可体现处理器的整体性能和线程间的公平性。

4 实验结果及分析

4.1 总体情况

DAFIR 取指策略相对于其他取指策略在算术平均 IPC 和调和平均 IPC 方面的性能对比如图 2 和图 3所示。

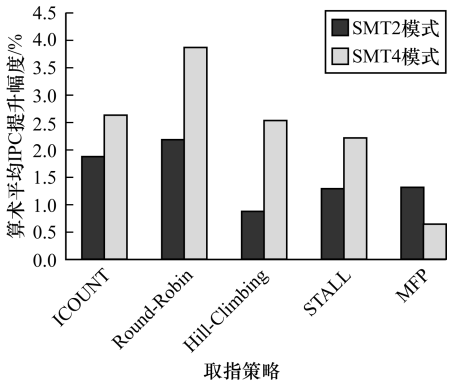

图 2 算术平均 IPC 提升幅度

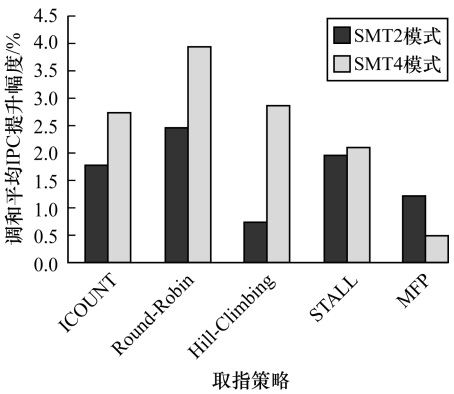

图 3 调和平均 IPC 提升幅度

实验结果表明,相对于所测试的 5 种取指策略,DAFIR 在提升处理器指令吞吐率和线程公平性都有一定的优势。在算术平均 IPC 方面,SMT2 模式下相对于 ICOUNT 平均提升 1.87%,相对于 STALL 提升1.27%,相对于 MFP 提升 1.32%;SMT4 模式下相对于 ICOUNT 提升 2.64%,相对于 STALL 提升2.22%,相对于 MFP 提升 0.64%。在调和平均 IPC 方面,SMT2 模式下相对于 ICOUNT 平均提升1.77%,相对于 STALL 提升 1.95%,相对于 MFP 提升1.21%;SMT4 模式下提升幅度则分别为 2.73%,2.10% 和 0.50%。

4.2 浮点与整数型程序的区别

DAFIR 区别对待浮点与整数资源的分配,图 4 显示了在 SMT2 模式下,不同浮点与整数类型组合程序时,DAFIR 相对于 ICOUNT 的性能提升。其中,FI 表示 2 个线程一个为浮点型程序一个为整数型程序;II 表示 2 个线程均为整数型程序,FF 表示 2 个线程均为浮点型程序。线程的类型在执行过程中根据线程对浮点资源使用情况决定,且在每个时间片结束时进行更新,因此,在整个执行过程中一个线程的类型不是固定不变的。而在结果呈现中,浮点型程序指的是在整个执行过程中对浮点资源平均使用率达到一定阈值的测试程序。

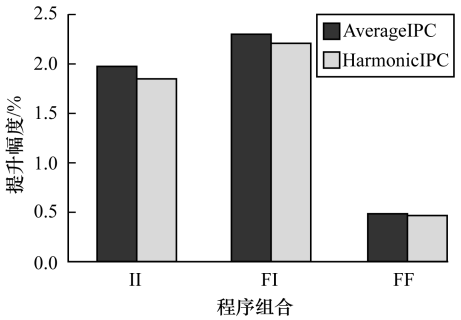

图 4 不同整数与浮点型程序组合的性能提升幅度

实验结果表明,属于 SPEC CPU2000 的浮点程序最终均表现为浮点型。因此,采用时间片的方式能有效识别出线程属于浮点类型还是整数类型程序。可以看出,DAFIR 取指策略在面对浮点型和整数型混合程序时能够得到明显的性能提升,算术平均 IPC 提升 2.29%。对于程序组合 II 也能提升性能,但是比 FI 类型提升更小。而对于 FF 型程序组合,性能提升最小,约为 0.5%。

综上可知,DAFIR 取指策略通过及时调整整数型线程和浮点型线程对于整数和浮点共享资源的使用上限,能更合理地分配共享资源,明显提升处理器在运行浮点和整数混合型程序时的性能。

4.3 资源分配方式的影响

DAFIR 借鉴了 MFP 取指策略关于资源分配的思想,并对资源分配方式进行了优化。图 5 展示了不使用资源分配策略、使用与 MFP 相同的资源分配策略和使用 DAFIR 资源分配策略对算术平均 IPC 的影响。其中,ToNo 表示与 DAFIR 有相同的线程选择方式(根据取指阶段发出但未取回的指令数量和取指缓冲中的指令数量决定线程取指优先级),但是不对共享资源进行分配控制的取指策略;ToMFP 表示与 DAFIR 有相同的线程选择方式,但是采用与 MFP 相同的共享资源分配方式(所有共享资源相同分配)的取指策略。

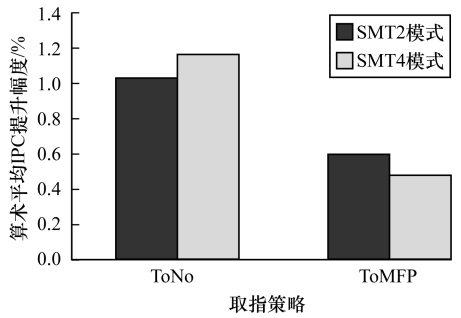

图 5 不同资源分配方式下的算术平均 IPC 提升幅度

实验结果表明,相对于不使用资源分配,DAFIR 在 SMT2 模式下提升 1.03%,在 SMT4 模式下提升 1.16%。相对于所有共享资源同等对待的分配方式,DAFIR 在 SMT2 模式下提升 0.60%,在 SMT4 模式下提升 0.48%。可以看出,根据线程执行时使用浮点和整数共享资源的情况进一步细化对共享资源的分配控制,能够提升处理器的整体性能。另外,对比 DAFIR 相对于 ICOUNT 取指策略的性能提升情况(SMT2 提升 1.87%,SMT4 提升 2.64%)可以看出,取指策略仅使用取指和译码阶段的指令数量能够比使用执行站台前指令数量取得更好的性能,而且还能降低实现的复杂度,减小硬件开销。

5　结束语

本文提出一种浮点与整数资源区别分配的新型取指策略 DAFIR。实验结果表明,相对于所测试的其他几种典型取指策略,DAFIR 取指策略在算术平均 IPC 和调和平均 IPC 方面均有一定的提高。相对于 ICOUNT,两线程时算术平均 IPC 提升 1.87%,调和平均 IPC 提升 1.77%,四线程时分别提升 2.64% 和 2.73%。在处理浮点与整数混合型程序时,DAFIR 能明显提升处理器的性能。两线程时算术平均 IPC 提升 2.29%,调和平均 IPC 提升 2.21%。下一步将研究更高效、复杂度更低的资源分配策略。

参考文献

[1] Tullsen D M, Eggers S J, Levy H M. Simultaneous Multithreading: Maximizing On-chip Parallelism [J]. ACM SIGARCH Computer Architecture News, 1995, 23(2):392-403.

[2] Cazorla F J, Ramirez A, Valero M, et al. Optimising Long-latency-load-aware Fetch Policies for SMT Processors [J]. International Journal of High Performance Computing and Networking, 2004, 2(1):45-54.

[3] Eyerman S, Eeckhout L. A Memory-level Parallelism Aware Fetch Policy for SMT Processors[C]//Proceedings of the 13th IEEE International Symposium on High Performance Computer Architecture. Washington D. C., USA: IEEE Press, 2007: 240-249.

[4] Weng Lichen, Liu Chen. A Resource Utilization Based Instruction Fetch Policy for SMT Processors [J]. Microprocessors and Microsystems, 2015, 39(1):1-10.

[5] 孙彩霞,张民选. 基于多个取指优先级的同时多线程处理器取指策略[J]. 电子学报, 2006, 34(5):790-795.

[6] Cazorla F J, Ramirez A, Valero M, et al. Dynamically Controlled Resource Allocation in SMT Processors[C]// Proceedings of the 37th International Symposium on Microarchitecture. Washington D. C., USA: IEEE Press, 2004: 171-182.

[7] Choi S, Yeung D. Learning-based SMT Processor Resource Distribution via Hill-climbing [J]. ACM SIGARCH Computer Architecture News, 2006, 34(2):239-251.

[8] Tullsen D M, Eggers S J, Emer J S, et al. Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor [J]. ACM SIGARCH Computer Architecture News, 1996, 24(2):191.

[9] Tullsen D M, Brown J A. Handling Long-latency Loads in a Simultaneous Multithreading Processor [C]//Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture. Washington D. C., USA: IEEE Computer Society, 2001:318-327.

[10] El-Moursy A, Albonesi D H. Front-end Policies for Improved Issue Efficiency in SMT Processors [C]// Proceedings of the 9th International Symposium on High-performance Computer Architecture. Washington D. C., USA: IEEE Press, 2003:31-40.

[11] Cazorla F J, Ramirez A, Valero M, et al. DCache Warn: An I-Fetch Policy to Increase SMT Efficiency [C]// Proceedings of the 18th International Parallel and Distributed Processing Symposium. Washington D. C., USA: IEEE Press, 2004:74.

[12] 孙彩霞,张民选. DWarn+:一种改进的同时多线程处理器取指策略[J]. 小型微型计算机系统, 2007, 28(9):1720-1723.

[13] Liu Yannan, Chen Tianzhou, Zhang Tiefei, et al. Dealing with the Functional Units Starvation in SMT [C]// Proceedings of the 14th IEEE International Conference on High Performance Computing and Communication & the 9th IEEE International Conference on Embedded Software and Systems. Washington D. C., USA: IEEE Press, 2012:208-215.

[14] Wang Huaping, Koren I, Krishna C M. An Adaptive Resource Partitioning Algorithm in SMT Processors[C]// Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques. New York, USA: ACM Press, 2008:230-239.

[15] Henning J L. SPEC CPU2000:Measuring CPU Performance in the New Millennium [J]. Computer, 2000, 33(7): 28-35.

[16] Luo Kun, Gummaraju J, Franklin M. Balancing Throughput and Fairness in SMT Processor[C]//Proceedings of 2001 IEEE International Symposium on Performance Analysis of Systems and Software. Washington D. C., USA: IEEE Press, 2001:164-171.

编辑　金胡考