基于多个取指优先级的同时多线程处理器取指策略

孙彩霞, 张民选
(国防科学技术大学计算机学院, 湖南长沙 410073)

摘 要: 同时多线程(SMT, Simultaneous Multithreading)处理器中, 同时运行的线程在共享资源的同时也在竞争资源. 如果一个发生 L2 cache失效的线程长时间占用共享资源, 那么会导致其他线程运行速度减慢, 甚至会因为缺少资源而停顿下来, 从而降低了 SMT处理器的总体性能. 本文提出了一种基于多个取指优先级的同时多线程取指策略 MFP(Multiple Fetch Priorities), 用于减少 L2 cache失效给处理器性能带来的负面影响. 模拟结果表明, 无论使用 IPC作为度量标准还是使用 Hmean作为度量标准, 对于所有类型的工作负载, 尤其是存储器访问密集的工作负载, MFP都要优于现有的其他取指策略. 此外, 对于不同的取指策略, MFP表现出不同程度的提升. 相对于 PDG的提升最明显, 平均 IPC以及平均 Hmean分别提高了 19.2%和 27.7%.

关键词: 同时多线程; cache失效; 取指策略; 取指优先级; 资源分配
中图分类号: TP303 文献标识码: A 文章编号: 0372-2112 (2006) 05-0790-06

An Instruction Fetch Policy Based on Multiple Fetch Priorities for SMT Processors

SUN Cai-xia, ZHANG Min-xuan
(*School of Computer, National University of Defense Technology, Changsha, Hunan 410073, China*)

Abstract In Simultaneous Multithreading (SMT) processors, co-scheduled threads share the processor's resources, but at the same time compete for them. A thread missing in L2 cache may occupy most of available resources for a long time, causing other threads run slower than they could or even stall because of lack of resources. As a result, the overall performance of SMT processors is degraded. In this paper, we propose a novel fetch policy called MFP (Multiple Fetch Priorities) to prevent the negative effects caused by L2 cache misses. Results show that our policy outperforms previously proposed fetch policies for all types of workloads, especially for memory bounded workloads, whether using IPC as a metric or using the harmonic mean as a metric. Results also tell that our policy shows different degrees of improvement over other fetch policies. The increment over PDG is greatest, reaching 19.2% in IPC and 27.7% in Hmean on average.

Key words simultaneous multithreading; cache miss; fetch policy; fetch priority; resource allocation

1 引言

同时多线程处理器[1~3]通过同时运行来自不同线程的指令来提高性能. 同时运行的线程共享处理器中的一些资源, 主要包括发射队列, 物理寄存器, 执行单元和再定序缓冲(ROB, Reorder Buffer)等. 共享资源在线程之间的分配方式对 SMT处理器的总体性能具有决定性的影响. 目前, SMT处理器中的资源分配主要由取指策略决定.

在 SMT处理器中, 共享资源的数目是有限的, 如果一个线程长时间占用某种资源, 就可能导致其他线程运行速度减慢, 甚至会因为缺少资源而停顿下来, 从而降低了 SMT处理器的总体性能. 发生 L2 cache失效的 load指令通常会导致这种情况发生. 目前, 针对这个问题提出了很多取指策略, 这些策略大致分成两类: 一类是停止从发生 cache失效的线程取指, 有代表性的策略是 STALL[4]; 另一类是释放发生 L2 cache失效的线程占用的资源, 有代表性

收稿日期: 2005-04-26; 修回日期: 2005-12-18
基金项目: 国家自然科学基金 (No. 60376018); 国家 863高技术研究发展计划重大项目 (No. 2002AA110020)

的策略是 FLUSH[4]. 这些取指策略在一定程度上能够减少 L2 cache失效给性能带来的影响, 但是却都存在一定的不足.

本文提出了一种基于多个取指优先级的取指策略 MFP(Multiple Fetch Priorities), 用于解决 L2 cache失效带来的问题. 在 MFP中, 根据线程的 cache行为为线程分配不同的取指优先级. 没有 cache失效的线程具有较高的取指优先级, 如果一个线程发生了 cache失效, 那么它的取指优先级就要相对变低. 每个周期, MFP从具有最高取指优先级的线程取指, 使得没有 cache失效的线程可以优先利用取指带宽. 如果没有 cache失效的线程无法填满取指带宽, 那么就从拥有 cache失效、取指优先级较低的线程取指, 以减少资源浪费. 当只有两个线程同时运行时, MFP引入了资源分配机制, 通过控制分配给发生 cache失效的线程的资源数目来避免资源阻塞或资源独占.

2 相关工作

为了减少 L2 cache失效给性能带来的负面影响, 提出了许多取指策略. 本节介绍几种典型的、使用比较广泛的取指策略.

ICOUNT是由 Tullsen等在文献[2]中提出的, 这种策略优先从处于译码段、重命名段和发射队列中指令数目最少的线程取指. 然而, ICOUNT不能解决 L2 cache失效带来的问题, 因为只要一个线程在流水线前端的指令数最少, 就会从该线程取指, 而不管这个线程是否发生了 L2 cache失效, 这样共享资源很有可能被阻塞 (clogging)或独占 (monopolization). 我们首先介绍 ICOUNT的原因在于下面的几种取指策略都是基于 ICOUNT的.

STALL试图阻止发生 L2 cache失效的线程占用大量共享资源, 以免造成资源阻塞或资源独占. STALL会检测一个线程是否拥有未决 (pending)的 L2 cache失效, 如果有, 就停止从该线程取指. 然而, 存在的问题是检测到 L2 cache失效后再停止取指可能太晚了, 以致于资源阻塞或资源独占已经发生了. 另一个问题是如果其他线程都不需要发生 L2 cache失效的线程所占用的资源, 那么停止从该线程取指会造成资源浪费.

FLUSH是对 STALL的扩展, 它通过释放阻塞线程 (即发生 L2 cache失效的线程)占用的所有资源解决 STALL中可能存在的资源阻塞或资源独占问题. 然而, 如果其他线程都不需要阻塞线程占用的资源, 那么将会造成极大的资源浪费. 而且, 释放资源时需要清除阻塞线程已经取出的指令, 这样就需要重新取指, 从而造成功耗的增加.

DG(Data Gating)[5]在每次发生 L1 data cache失效时都会停止从相应的线程取指, 以此减少 L1 data cache失效带来的影响. 然而, 如果 L1 data cache失效并没有导致 L2 cache失效, 这时停止从线程取指会极大的降低线程的运行速度. 此外, 就像 STALL一样, 在 DG中也存在资源浪费, 而且更加严重.

PDG(Predictive Data Gating)[5]只要预测到某个线程会发生 cache失效, 就会停止从该线程取指. 使用 cache失效预测器, 可以避免 STALL中存在的 L2 cache失效发现过晚带来的问题. 但是由于 cache失效很难预测[6], 所以 PDG所带来的好处是有限的. 而且, PDG仍然采用停止取指的方式, 因此 STALL中存在的资源浪费问题在 PDG中同样存在.

DWarn[7]采用了一种特别的方式处理 L2 cache失效问题, 我们在设计 MFP策略时借鉴了 DWarn的一些设计思想. 当检测到一个线程发生 cache失效时, DWarn会降低该线程的取指优先级, 而不是马上停止从该线程取指, 从而可以在 L1 data cache失效没有导致 L2 cache失效时避免不必要的损失. 但是发生 L2 cache失效的线程的指令仍然有机会进入流水线占用资源, 尤其是当同时运行的线程数目很少时, 极有可能造成资源阻塞或资源独占.

从上面的分析可以看出, 当前的取指策略主要存在两个方面的问题: 第一, 不能有效地阻止共享资源被发生 L2 cache失效的线程阻塞或独占; 第二, 在阻止一个线程占用其他线程不需要的资源时会造成资源浪费.

3 基于多个取指优先级的同时多线程取指策略

MFP的目标有两个: 一个是减少 L2 cache失效给性能带来的负面影响, 另一个是试图解决当前取指策略中存在的问题, 即资源阻塞或资源独占及资源浪费.

3.1 基本思想

在 MFP中, 根据线程的 cache行为为每个线程赋予不同的取指优先级. MFP支持三个取指优先等级: Level 1, Level 2和 Level 3. Level 1是最高优先级, Level 3是最低优先级. 最初, 所有的线程都处在最高取指优先级, 即 Level 1. 如果某个线程发生了 L1 data cache失效, 那么该线程的取指优先级降低到 Level 2 之后, 如果 L1 data cache失效导致 L2 cache失效, 那么继续降低该线程的取指优先级, 也就是变成了 Level 3 处于同一取指优先级的线程由 ICOUNT进行排序. 图 1详细描述了取指优先级的转换过程.

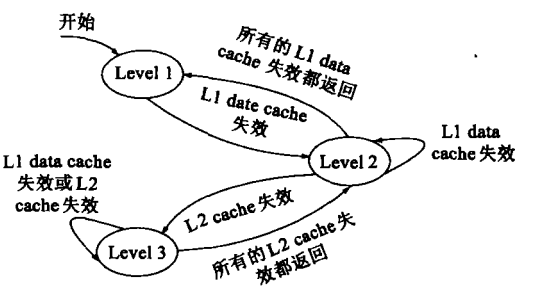

图 1　取指优先级的转换

MFP的优势体现在以下三个方面:

(1)通过降低发生 cache失效的线程的取指优先级, MFP优先从没有 cache失效的线程取指, 使得共享资源被拥有 cache失效的线程阻塞或独占的可能性大大降低;

(2)线程发生 cache失效后没有马上停顿下来, 而是被降级. 这样, 当没有 cache失效的线程无法填满取指带宽时, 可以从发生 cache失效、取指优先级较低的线程取指, 从而将少了资源浪费;

(3)当线程中的 L1 data cache失效变成 L2 cache失效后, 线程的取指优先级进一步降低, 使得只拥有 L1 data cache失效的线程可以优先于发生 L2 cache失效的线程占用共享资源, 提高资源的利用效率.

然而, 当只有两个线程同时运行时, MFP仍然无法阻止共享资源的阻塞或独占. 这是因为, 在我们的策略中, 把 ICOUNT2.8[2]作为基本取指策略, 每个周期从两个线程取出 8条指令. 由于存在分支指令, 使得从一个线程同时取出 8条指令的可能性并不大, 这样就必须使用另一个线程的指令填满取指带宽, 即便是这个线程发生了 L2 cache失效. 逐渐的, 共享资源就可能被发生 L2 cache失效的线程阻塞或独占. 为了解决这个问题, 我们引入资源分配(Resource Allocation)机制. 当只有两个线程运行时, 除了降低拥有 cache失效的线程的取指优先级外, 还要通过资源分配机制限制这些线程所能占用的共享资源数目. 当发生 cache失效的线程试图占用更多的资源时, 就会停止从该线程取指, 以避免该线程阻塞或独占共享资源.

3.2 资源分配机制

当只有两个线程同时运行时, MFP需要控制共享资源的分配. 比较常用的资源分配机制有两种: 静态资源分配(SRA, Static Resource Allocation)和动态资源分配(DRA, Dynamic Resource Allocation).

在静态资源分配机制中, 分配给 Level 2和 Level 3的线程的资源数目是固定的, 都等于 T/N, 其中 T 是某种资源的总数, N 是同时运行线程的总数, 这里 N 等于 2.

在动态资源分配机制中, 根据每个线程的取指优先级动态的在线程之间分配资源. 分配给线程 $i(i=0, 1)$的某种资源的数目 M_i 由等式(1)定义, 其中 PL_i 是线程 i的取指优先级, T 是某种资源的总数.

表 1 某 32项资源的分配

		SRA		DRA	
PL_0	PL_1	M_0	M_1	M_0	M_1
1	1	-	-	-	-
1	2	-	16	-	21
1	3	-	16	-	24
2	1	16	-	21	-
2	2	16	16	16	16
2	3	16	16	13	19
3	1	16	-	24	-
3	2	16	16	19	13
3	3	16	16	16	16

"-"表示线程可以使用的资源数目不受限制

$$M_i = \frac{PL_i}{PL_i + PL_{1-i}} * T \quad (1)$$

表 1给出了所有可能情况下某种资源在两个线程之间的分配情况. 由于每个线程的取指优先级只有 3种, 所以共有 9种情况. 从表 1可以看出, 在静态资源分配机制中, 发生 cache失效的线程最多可以占用一半的共享资源. 而在动态资源分配机制中, 低取指优先级的线程可以从高取指优先级的线程借用共享资源, 这样, 前者就可以利用后者不需要的资源, 从而提高资源利用率.

3.3 实现

为了根据 cache行为给线程赋予取指优先级, 每个线程需要 1个 L1 data cache失效计数器(L1M-Counter)和 1个 L2 cache失效计数器(L2M-Counter). 线程每次发生 L1 data cache失效时, L1M-Counter都要加 1, 每次装载 L1 data cache时把相应线程的 L1M-Counter减 1. 每次 L1 data cache失效导致 L2 cache失效时, L2M-Counter加 1, 每次装载 L2 cache时把相应线程的 L2M-Counter减 1. 如果一个线程的 L2M-Counter非零, 那么这个线程的取指优先级为 Level 3, 否则如果 L1M-Counter非零, 则该线程的取指优先级为 Level 2. 只有 L1M-Counter和 L2M-Counter都是零时, 线程才处于 Level 1.

两个线程同时运行时, 为了监视资源使用情况, 每个线程需要 5个资源使用计数器. 3个用于监视指令队列的使用, 即整型队列, 浮点队列和 load/store队列. 2个用于监视物理寄存器的使用, 即整数寄存器和浮点寄存器. 再定序缓冲和功能单元也是关键的资源, 但是我们在实验中假定同时运行的多个线程是完全独立的, 因此可以为每个线程提供单独的再定序缓冲, 并把功能单元设计成流水的, 这样, 就不必监视这两种资源的使用. 指令译码阶段增加资源使用计数器的值; 提交阶段如果释放了物理寄存器, 则减少相应线程的物理寄存器使用计数器的值; 如果有指令从队列发射出去执行, 则减少相应线程的指令队列使用计数器的值. 当只有两个线程运行时, 每个周期, MFP会检查处于 Level 2或 Level 3的线程的资源使用计数器, 如果超过了分配给它的数目, 则停止从该线程取指.

现在考虑如何实现资源分配机制. 如果使用 SRA, 并不需要额外的电路, 因为分配给线程的资源数目是固定的. 如果使用 DRA, 则需要简单的控制逻辑. DRA的实现可以有两种方法[8]. 一种方法是使用组合电路实现等式(1), 该电路的输入包括两个线程的取指优先级以及资源的总数, 输出分配给每个线程的资源数目. 另一种方法是使用直接映射表, 由两个线程的取指优先级索引, 通过查找该表获得每个线程可以占用的资源数目.

4 实验设置

我们采用 SMTSIM[9]进行实验模拟. SMTSIM 模拟了所有类型的延时, 包括 cache访问延时, 分支误预测延时, TLB

失效处理延时等. 同时, SMTSIM可以收集和统计实验中有关的信息, 如 cache失效率, 分支预测精度以及 IPC 等, 在本文的实验中, 我们只关心每个线程的 IPC 值. 表 2给出了模拟器的基本配置.

表 2　模拟器的基本配置

参数	值
取指带宽	8条指令 /周期
基本取指策略	ICOUNT2. 8
指令对列	32项整型队列, 32项浮点队列, 32项 load/store队列
功能单元	6个整形, 3个浮点, 4个 load /store
物理寄存器	384个整数, 384个浮点
再定序缓冲	256项 /线程
分支预测器	2k项的 gshare
分支目标缓冲	256项, 4路组相联
返回地址栈	256项
L1I cache	64kB, 2路组相联, 每行 64字节, 1个周期访问延时
L1D cache	64kB, 2路组相联, 每行 64字节, 1个周期访问延时
L2 cache	512kB, 2路组相联, 每行 64字节, 10个周期访问延时
主存	100个周期访问延时

表 3　测试程序

类型	测试程序
存储器访问密集	mcf, twolf, vpr, parser, ammp, applu, art, swim
高 ILP	aspi, fma, eon, gcc, gzip, vortex, crafty, bzip2

表 4　多线程负载

线程数	类型	包含的测试程序
2	ILP	{gzip, bzip2}, {gcc, aspi}, {vortex, fma}, {eon, crafty}
	MIX	{gzip, vpr}, {gcc, ammp}, {art, vortex}, {fma, parser}, {aspi, twolf}, {crafty, art}, {bzip2, swim}, {eon, applu}
	MEM	{mcf, vpr}, {ammp, parser}, {twolf, art}, {mcf, swim}
4	ILP	{aspi, fma, eon, gcc}, {gzip, vortex, crafty, bzip2}, {fma, eon, gcc, crafty}
	MIX	{fma, eon, parser, ammp}, {aspi, gzip, mcf, art}, {crafty, bzip2, vpr, parser}, {eon, gcc, twolf, art}, {vortex, aspi, mcf, ammp}, {gcc, fma, parser, applu}
	MEM	{vpr, parser, ammp, applu}, {mcf, art, vpr, twolf}, {twolf, vpr, art, swim}
6	ILP	{aspi, fma, eon, gcc, gzip, vortex}, {fma, eon, gcc, gzip, vortex, crafty}
	MIX	{fma, eon, gcc, vpr, parser, ammp}, {aspi, fma, eon, twolf, vpr, parser}, {eon, gcc, gzip, mcf, art, vpr}, {aspi, gcc, eon, vpr, swim, parser}
	MEM	{mcf, twolf, vpr, parser, ammp, applu}, {twolf, vpr, parser, ammp, applu, art}
8	ILP	{aspi, fma, eon, gcc, gzip, vortex, crafty, bzip2}
	MIX	{eon, gcc, gzip, aspi, mcf, twolf, ammp, applu}, {vortex, crafty, bzip2, fma, vpr, parser, art, swim}
	MEM	{mcf, twolf, vpr, parser, ammp, applu, art, swim}

表 3给出了实验中所使用的所有测试程序, 这些程序都来自 SPEC2000测试集[10], 并且都使用 ref输入数据集. 在实验中, 完整的模拟测试程序要花费大量的时间, 有时甚至是不可能完成的. 因此我们采用文献[11]中所提到的方法, 只模拟每个程序中最有代表性的三亿条指令片段. 根据 cache行为可以把我们使用的测试程序分成两组: 一组是存储器访问密集的程序, 这些程序的特点是 cache失效很多, 平均每条指令遇到的 L2 cache失效在 0. 02到 0. 12之间; 另一组是指令级并行性 (ILP, Instruction Level Parallelism)较高的程序, 这组程序的特点是 cache失效率较低. 表 4给出了实验中使用的多线程负载. MEM 型负载中的测试程序全部来自于表 3中的第一组, ILP型负载中的测试程序全部来自于表 3中的第二组, MIX型的负载则由两组中的程序按相同比例组合而成. 为了避免实验结果倾向于某个测试程序组合, 我们为每种类型的负载随机抽取了多组, 如 4线程 MIX 型负载中共包含 6组测试程序, 最终结果取多组测试程序运行结果的平均值.

我们在对结果进行比较时使用两个度量标准: IPC和 Hmean[12]. IPC 是每个周期运行的指令数目, 就像 Tullsen 在[4]中所描述的那样, 如果取指策略优先运行指令级并行性高的线程, 那么使用 IPC 进行度量有时是不准确的. Hmean是负载中每个测试程序相对 IPC的调和平均值 (相对 IPC是指多线程运行环境下的 IPC与单线程运行时 IPC 的比值), 它可以有效地识别优先运行高 IPC线程所带来的吞吐量提升这一假象. 因此, 同时使用 IPC和 Hmean, 可以公平地进行比较.

5　实验结果

在 MFP中, 我们分别实现了第 3节中描述的两种资源分配机制, 因此我们首先比较实现不同资源分配机制的 MFP取指策略的性能差异, 然后再把 MFP策略和其他的取指策略进行比较. 我们把采用 DRA 机制的 MFP叫做 MFP-D, 采用 SRA 机制的 MFP叫做 MFP-S. 由于只有两个线程运行时才使用资源分配机制, 因此在比较 MFP-D 和 MFP-S 只需要运行两线程工作负载.

5.1　MFP-D与 MFP-S的比较

图 2给出了 MFP-D 相对于 MFP-S 的 IPC 提升和 Hmean提升. 可以看出无论使用 IPC作为度量标准还是使用 Hmean作为度量标准, MFP-D 都要优于 MFP-S, 分别提高了 3. 3%和 4. 2%. 这是因为 DRA 在分配资源时, 同时检查两个线程的 cache行为, 并根据这些行为动态的调整分配给发生 cache失效的线程的资源数目; 而 SRA 在给一个发生 cache失效的线程分配资源时, 只考虑该线程的 cache 行为, 忽略另一个线程. 因此 DRA 能够更有效的利用共享资源, 提高处理器的性能.

此外, 从图 2我们还可以看出对于 MIX 型的负载, MFP-D 获得的提升更加明显. 关键的原因在于只有当两个线程的取指优先级不同时, SRA 和 DRA 获得的资源分配情况才有所不同 (参见表 1). 而从表 5可以看出, MIX 型负

载中的两个线程处于不同取指优先级的机会更多.

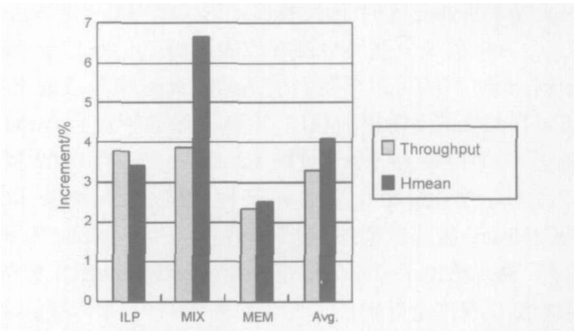

图 2 MFP-D 相对于 MFP-S 的 IPC/Hmean 提升

表 5 两线程负载中线程取指优先级的分布

负载类型	处于相同取指优先级	处于不同取指优先级
ILP	53.8	46.2
MIX	24.8	75.2
MEM	64.4	35.6

5.2 MFP与其他取指策略的比较

我们只给出 MFP-S与其他策略的比较结果, 把这些结果和图 2中的数据结合, 我们不难得到 MFP-D 相对于其他取指策略的性能提升.

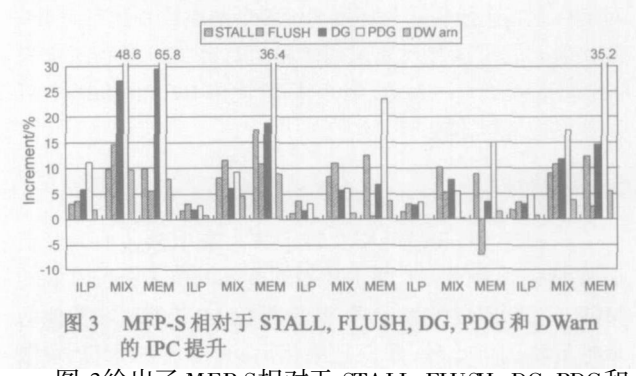

图 3 MFP-S 相对于 STALL, FLUSH, DG, PDG 和 DWarn 的 IPC 提升

图 3给出了 MFP-S相对于 STALL, FLUSH, DG, PDG和 DWarn的 IPC提升. 结果表明对于所有类型的工作负载, MFP-S都要优于其他的取指策略.

MFP-S相对于 PDG的提升最明显, ILP负载、MIX 负载和 MEM 负载的平均 IPC提升分别达到了 4.9%, 17.4%和 35.2%. 其次是 DG, 对于三种类型负载, MFP-S获得的平均 IPC提升分别为 3.0%、11.6%和 14.5%. 这是因为在 DG中, 线程每次发生 L1 data cache失效时都要停止取指. 而事实上, 当同时运行的线程数目不多时, 对共享资源的压力并不大, 停止取指会造成严重的资源浪费. PDG存在和 DG相同的问题, 再加上 cache失效预测的准确率又很低, 使得 PDG的效果更加不理想. 而 MFP采取的策略是降低发生 cache失效的线程的取指优先级而不是停止从这些线程取指, 这样可以极大的减少资源浪费. 因此, 相对于 PDG和 DG, MFP获得了很大的性能提升. 从图 3中, 我们还可以看出, 随着同时运行的线程数目的增多, MFP相对于 PDG和 DG的性能提升在变低. 这是因为随着线程数目的增多, 线程对共享资源的竞争变得愈发激烈, 造成资源浪费的可能性也越来越小, 从而使得 MFP带来的性能提升很有限.

对于所有类型的负载, MFP-S相对于 FLUSH 都有一定的提升. 对于 ILP负载、MIX负载和 MEM 负载, 平均 IPC提升分别为 3.2%, 10.7% 和 2.4%. 但是, 对于 8线程的 MEM 负载, MFP-S却下降了 7.0%. 这是因为 8个存储器行为密集的程序同时运行时, 对共享资源造成的压力非常大, 这时释放发生 L2 cache失效的线程占用的资源对其他线程的运行非常有益. 不过, 获得这种好处的代价是要重新取出被清除的指令, 从而带来了额外的功耗开销.

对于 ILP负载、MIX负载和 MEM 负载, MFP-S相对于 STALL的平均 IPC 提升分别为 1.8%, 9.1% 和 12.2%. MFP-S相对于 DWarn的平均 IPC提升并不明显, 对于 ILP负载, 只提高了 0.7%, 而对于另外两种负载的提升也不超过 6.0%. 此外, 当同时运行的线程数目超过 4时, MFP-S和 DWarn所获得的 IPC几乎相同. 这是因为当运行线程数目大于 2时, MFP优于 DWarn的地方在于 MFP区分拥有 L2 cache失效的线程和只拥有 L1 data cache失效的线程, 让只拥有 L1 data cache失效的线程优先取指并竞争资源. 而随着同时运行的线程数目的增多, 从发生 cache失效的线程取指的可能性越来越小, 从而使得 MFP的优越性无法体现出来.

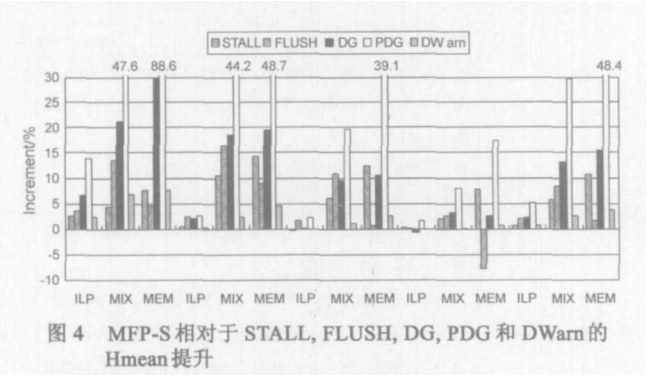

图 4 MFP-S 相对于 STALL, FLUSH, DG, PDG 和 DWarn 的 Hmean 提升

图 4给出了 MFP-S相对于 STALL, FLUSH, DG, PDG和 DWarn的 Hmean提升. 结果表明对于所有类型的工作负载, MFP-S都要优于其他的取指策略. 主要原因在于 MFP从来不直接停止或清除发生 cache失效的线程, 而是给这些线程赋予较低的取指优先级, 使得这些线程仍然有机会竞争资源. 这样, 发生 cache失效的线程可以充分利用其他线程不需要的资源. 因此, MFP获得的效果是, 在不影响某些线程执行的同时, 尽可能提高其他线程的性能, 从而可以更好的平衡各个线程的 IPC, 进而提高 Hmean.

从图 3和图 4我们还可以得出一个结论: 运行 MIX负载和 MEM负载时 MFP-S获得的提升要比运行 ILP负载时获得的提升明显. 回想我们在第 4节所提到的, MFP和其他取指策略的区别在于, MFP在减少 L2 cache失效给性能带来的负面影响的同时, 试图解决其他取指策略中存在的问题, 即资源阻塞或资源独占, 以及资源浪费. 而我们知道

只有发生 cache失效, 才有可能随之带来这些问题. 因此, 当运行包含存储器行为密集型测试程序的工作负载时, MFP策略更加突出.

6 结论

本文提出了一种新颖的同时多线程处理器取指策略 MFP, 用于减小 L2 cache失效给性能带来的负面影响, 同时试图解决当前取指策略中存在的一些问题, 包括资源阻塞或资源独占, 以及资源浪费. 模拟结果表明:

(1)无论使用 IPC作为度量标准还是使用 Hmean作为度量标准, 对于所有类型的工作负载, MFP都要优于现有的其他取指策略. 尤其是对于存储器访问密集的工作负载, MFP获得的提升更加明显.

(2)对于不同的取指策略, MFP表现出不同程度的提升. 相对于 PDG的提升最明显, 平均 IPC以及平均 Hmean分别提高了 19.2%和 27.7%. 其次是 DG, 平均 IPC提升和平均 Hmean提升分别为 9.7%和 10.3%.

(3)当只用两个线程同时运行时, 相对于其他取指策略, MFP获得了极大的性能提升. 尤其是使用动态资源分配机制在线程之间分配资源时, MFP获得的提升更加明显. 与静态资源分配机制相比, 使用动态资源分配机制获得的额外 IPC提升如下: 对 ILP负载, 提高了 3.8%, 对 MIX负载, 提高了 3.9%, 对 MEM 负载, 提高了 2.3%; 对三种负载的 Hmean提升分别为 3.4%、6.6%和 2.5%.

参考文献:

[1] D Tullsen, S Eggers, et al. Simultaneous multithreading Maximizing on-chip parallelism[A]. Proceedings of the 22nd Annual International Symposium on Computer Architecture[C]. Santa Margherita Ligure, Italy: ACM Press, 1995. 23(2): 392-403.

[2] D Tullsen, S Eggers, J Emer, et al. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor[A]. Proceedings of the 23rd Annual International Symposium on Computer Architecture[C]. PA, USA: ACM Press 1996. 24(2): 191-202.

[3] S J Eggers, J Emer, H M Levy, et al. Simultaneous multithreading a platform for next-generation processors[J]. IEEE Micro, IEEE Computer Society Press, 1997, 17(5): 12-19.

[4] D Tullsen, J Brown. Handling long-latency loads in a simultaneous multithreaded processor[A]. Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture[C]. Texas, USA: IEEE Computer Society Press, 2001. 318-327.

[5] A E EMoursy, D Albonesi. Front-end policies for improved issue efficiency in SMT processors[A]. Proceedings of the 9th International Conference on High Performance Compute Architecture[C]. California, USA: IEEE Computer Society Press, 2003. 31-42.

[6] A Yoaz, M Erez, et al. Speculation techniques for improving load related instruction scheduling[A]. Proceedings of the 26th Annual International Symposium on Computer Architecture[C]. Georgia, USA: ACM Press, 1999. 27(2): 42-53.

[7] F J Cazorla, A Ramirez, et al. DCacheWarn: an I-Fetch policy to increase SMT efficiency[A]. Proceedings of the 18th International Parallel and Distributed Processing Symposium[C]. Santa Fe, New Mexico: IEEE Computer Society Press, 2004. 74-83.

[8] F J Cazorla, A Ramirez, et al. Dynamically controlled resource allocation in SMT processors[A]. Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture[C]. Portland, Oregon: IEEE Computer Society Press, 2004. 171-182.

[9] D Tullsen. Simulation and modeling of a simultaneous multithreading processor[A]. Proceedings of the 22nd Annual Computer Measurement Group Conference[C]. San Diego, CA, USA: ComputerMeasurementGroup, 1996. 819-828.

[10] The standard performance evaluation corporation[Z]. http://www.specbench.org.

[11] T Sherwood, E Perelman, et al. Basic block distribution analysis to find periodic behavior and simulation points in applications[A]. Proceedings of the 10th Intl. Conference on Parallel Architectures and Compilation Techniques[C]. Barcelona, Spain: IEEE Computer Society Press, 2001. 3-14.

[12] K Luo, J Gummaraju, et al. Balancing throughput and fairness in SMT processors[A]. Proceedings of the Intl. Symposium on Performance Analysis of Systems and Software[C]. Arizona, USA: IEEE Computer Society Press, 2001. 164-171.

作者简介:

孙彩霞　女, 1979年生, 博士研究生. 2001年获得国防科学技术大学计算机科学与技术专业学士学位; 2003年开始提前攻读博士学位. 主要研究方向为高性能计算机系统结构与 VLSI设计. E-mail: cxsun1979@163.com

张民选　男, 1954年生, 教授, 博士生导师. 主要研究方向为高性能计算机系统结构、微处理器设计及 ASIC技术等.