

基于简化 Trace 的动态隐式断言执行

唐遇星 邓 鵬 窦 勇 周兴铭

(国防科技大学计算机学院分布与并行处理国家重点实验室 长沙 410073)

摘 要 分支指令与分支预测失败限制了处理器发掘指令级并行(ILP)的潜力. 通过 If-conversion 或 Predicated 执行将程序中的控制相关转化为数据相关,能较好地降低分支预测开销. 提出一种基于简化 Trace 结构的动态隐式断言执行机制(Dynamic Implicit Predication, DIP),而早期的相关研究主要集中于由编译器显式为宽发射处理器产生静态 Predicated 指令. 无需编译器或者其他二进制工具的帮助, DIP 可以在程序运行过程中识别可以进行断言变换的指令片断,完成指令转换与优化,并在以后的执行中使用优化后的指令 Trace. 基于 SPEC2000 模拟测试表明 DIP 可以有效避免错误的分支预测,提高并行度,单个程序的 IPC 平均提高 10.3%,基准程序的平均加速比可达 7.59%.

关键词 指令级并行;断言;动态隐式断言执行;踪迹缓冲;流水线

中图法分类号 TP302

Dynamic Implicit Predication Based on Lite Trace Cache

TANG Yu-Xing DENG Kun DOU Yong ZHOU Xing-Ming

(National Key Laboratory of Parallel and Distributed Processing, School of Computer,
National University of Defense Technology, Changsha 410073)

Abstract To exploit instruction level parallelism, modern microprocessor usually converts control dependences into data dependences. If-conversion and predicated execution are widely adopted to eliminate branch misprediction penalty. In this paper, a trace-based predicate mechanism named DIP (Dynamic Implicit Predication) is discussed. Previous predication execution depends on compiler to generate explicit predicated instructions. The candidates of if-conversion will be identified during dynamic execution. Classical trace cache has been modified to store DIP traces, which include instructions both from fall-through and target block behind the conditional branch. Hardware will add predication to DIP trace automatically. With the help of DIP, legacy applications can benefit from predication mechanism without recompiling source code. Simulation of DIP under various hardware configurations is presented in the paper. Results have shown promising performance improvement. For SPEC INT2000 benchmark, average IPC (Instruction Per Cycle) improvement achieves 10.3%, and average speedup of execution time is 7.59%.

Keywords ILP; predication; dynamic implicit predication; trace cache; pipelining

1 引 言

除了使用更复杂的分支预测机制,还可以使用

Predication 和 Speculation 两种方式来处理分支指令带来的控制相关. Predication 或者 If-conversion 通过将条件分支指令转换成 Predication 设置指令,在以后的执行中按照 Predicated 寄存器的值来提交

指令运行结果. 与 Speculation 相比, Predication 无需错误检查/恢复机制, 仅增加部分无效指令开销. 由于现代处理器的功能单元数目远远超过实际的取值能力, 增加一些额外的指令执行是可以承受的.

并非所有的分支指令都可以成为 If 分支, 例如间接转移指令由于可能存在不可知的多个分支目标而不能用 Predication 处理. 过于复杂的 Then 路径或者 Else 路径也会阻碍 Predication 的应用. Predication 虽然可以消除分支预测错误带来的额外开销, 但也使得原来无用路径的指令进入处理器流水线执行. 研究者常需借助于编译器、二进制工具或者程序运行 Profiling 技术来寻找适合 Predication 的场景.

现有的 Predication 方案, 大多数由于修改或者扩充了指令集体系结构 ISA 需要重新编译^[1-2], 或者需要特定的二进制工具辅助产生 Predication 信息^[3]. 本文提出了完全硬件实现、基于简化 Trace 结构的动态隐式 Predication 执行机制 (Dynamic Implicit Predication, DIP). DIP 无需编译器或者二进制工具对已有可执行代码作任何修改, 处理器中的 Predication 寄存器和 Predicated 指令对软件开发完全透明. Predication 处理后的指令存放在简化后的 Trace Cache 结构中. DIP 机制在处理器关键路径之外, 并可以在已有的 Trace Cache 设计^[4-5]上经过简化、修改来实现. 在模拟测试中, DIP 以较低的硬件开销, 将测试程序的平均 IPC 提高了 10.3%, 相对于未使用 DIP 机制的处理器而言, 平均的加速比达到 7.59%. 在对 DIP 扩展性的研究及其在深度流水线中的性能模拟研究过程中, DIP 展现出良好的扩展性能. 通过 DIP 避免预测失败对超流水结构设计很有价值.

本文第 2 节介绍 Predication 的相关研究工作; 第 3 节描述了 DIP 模型, 详细分析了 Trace 的形成过程以及 Trace 管理; 第 4 节详述了 DIP 的性能模拟, 除了对比不同 DIP 配置的性能差异之外, 还分析了 DIP 的可扩展性以及主要体系结构参数对 DIP 性能的影响.

2 相关研究工作

现代微处理器 (IA64) 和 DSP (TMS320C6x) 通过编译支持和特殊的 ISA 来实现 Predication^[1,6]. 依照处理器在 ISA 上对 Predication 的支持, 可以将

其分为以下 3 类:

(1) 完全 Predication. 在 IA64 结构中, 操作码的一个特殊域或者一组特殊的 Predication 寄存器用于表示当前指令是否需要 Predication. 在整个指令集上提供 Predication 可以方便编译器选择指令. 但是由于占用了操作码空间, 需要较长的指令格式, 同时带来新的 ISA 兼容问题.

(2) 条件执行 (部分 Predication). 大多数的 ISA 提供条件指令, 最常见的是 CMOV 条件数据转移指令^[7]. 如果条件成立, 操作将执行, 否则该指令的语义等同于 NOP 空操作. 很少的条件指令简化了兼容性处理, 但 Predication 的使用仍然需要修改可执行程序, 并且编译器只有很少的选择空间, 难以产生高效的代码.

(3) 动态 Predication. 处理器保持原有 ISA 不变, 通过动态硬件选择并替代 If 分支, 将后续两个执行路径上的代码 Predication 化. 以往的动态 Predication 研究仍然需要编译器或者二进制工具提供识别可能的 If-branch^[8].

Mahlke 等^[9]提出在编译器过程使用 Hyperblock 结构来支持 Predication. Pneumatikatos 和 Sohi^[10]提出的 guarded execution 通过在指令中添加 Predication 信息, 并使用位掩码来实现 Predication. Tyson^[11]和 Mahlke^[12]研究了 Predication 可能带来的性能提升. 他们的研究表明 30% 的动态分支可以通过完全 Predication 消除, 而部分 Predication 可以提高近 5% 的性能.

在动态 Predication 方面, 早期 Heil 和 Smith^[13]提出用双路径硬件来同时执行 Then 路径和 Else 路径. Klauser^[8]提出了动态的 Hammock, 在指令流中动态插入 Cmov 指令将 Predication 与 ISA 独立开来. 动态的 Hammock 需要编译或者二进制工具来指导 If-branch 的选择, 每次遇到相同的 If-branch, 位于关键路径上指令插入模块都需要重复以前的操作.

与传统动态 Predication 模型相比, 本文提出的 DIP 机制无需任何静态工具的辅助, 完全使用动态 Predication 来进行优化. DIP 没有修改编译可见的 ISA, 但与 Trace 中指令对应的标记 Tag 实际上为整个 ISA 添加了 Predication 扩展. 测试表明 DIP 的性能超过了静态的部分 Predication 扩展. 除此以外, DIP 使用简化的 Trace 保留了 Predication 后的指令块, 增加了新的优化机会.

3 DIP 模型

如图 1 所示,类似于传统 Trace Cache 的结构与流程,DIP 位于处理器的关键路径之外^[5].除了添加相关的 Predication 寄存器资源,DIP 使用 If 分支过滤器(If-Branch Fliter)扫描从流水线提交段正常结束的指令.在找到合适的 If 分支指令之后,按照传统 Trace Cache 的方式,用封装器 Compactor 收集 Trace 指令集,使用优化器 Optimizer 优化 Trace 长度与指令序列,最终将经过 Predication 的指令组织成 DIP Trace ,存储在简化后的 Trace Cache 结构中(DIP Lite Trace Cache).重用 Predicated Trace 将避免在难以预测的 If 分支上产生预

测失败,提高处理器的 IPC,加速程序运行.

Trace 封装机制(Trace Compactor(图 1))将来自两条分支路径的指令封装在一条直线式的 Trace 中.可以增加分支历史记录,通过访问分支预测器纪录内容获得难以预测分支的两个最频繁分支信息;也可以等待下一次遇到相同分支指令的时候收集之后提交的指令来获得不同分支的指令;甚至可以直接扫描 If 分支后指令做目标地址分析来获得两个分支的指令构成 Trace.已有的 Trace 优化调度都可以用于 DIP Trace 上,并且由于真正消除了分支指令,还可以进一步简化指令调度,使用低开销的表调度方法. Trace Cache 的配置管理、IF 分支的识别和分支预测器的准确度是 DIP 性能的关键,相关内容在后面的章节以及模拟验证部分进一步讨论.

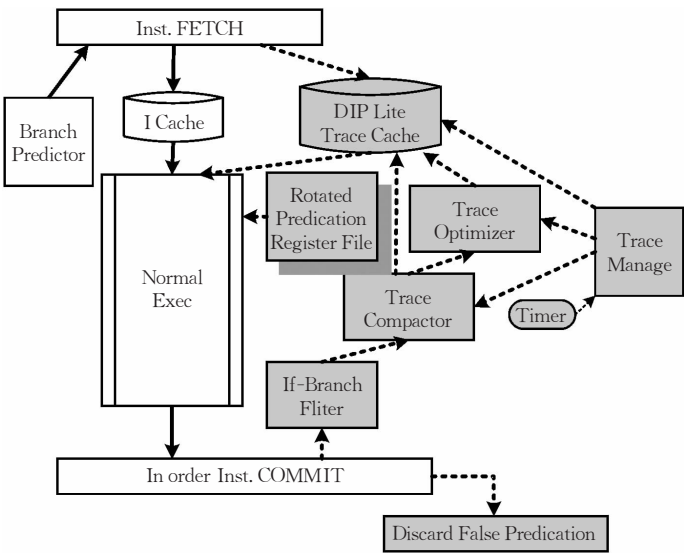


图 1 DIP 框架在流水线中的位置

3.1 DIP Lite Trace

在 DIP 模型中,指令块的 Predication 变化规则如下:以 If-分支所在基本块中的某条指令作为起点,将两条路径添加 Predication 的指令以及 Merge Point 之后部分没有变换的指令,作为一个连续的指令流,称为 DIP Lite Trace.

与传统 Trace 概念相比,DIP Lite Trace 不再包含有控制流指令,无需标记 Trace 中控制类指令的位置,在 Trace 内部做优化调度的时候也无需考虑 Trace 失效和失效恢复的问题,简化了传统 Trace Cache 中复杂的 Tag 结构与相关控制机制. Trace Cache 的 Tag 结构可以提供在整个 ISA 空间上扩展 Predication 的能力,同时无需修改 ISA 和二进制文件.图 2 是 SPEC CPU2000 164. gzip 在 DIP Lite Trace 的简单示例.

比较图 2 两侧 Predication 前后的指令可以看到,If 分支被替换成 Predication 设置指令 SetEq,它以相同的条件设置 Predication 寄存器 P1 和 P2.在 SetEq 指令(No.Pf)和路径合并点 No.Ps 之间的指令添加了 Predication,合并点及其之后的指令则保持不变.在实际硬件设计中 SetEq 这样的指令和添加了 Predication 的指令是编译不可见的,只保存在 Lite Trace Cache 内部,因此可以使用包含足够信息的任意内部编码格式.为简化设计,也可以在对应 NO.If 到 NO.It 指令的编码上添加额外字段.

在处理器开始取值周期之前,已经确定的 PC 地址被同时递交给传统指令 Cache (I-Cache)和 DIP Lite Trace Cache (DIP-Cache).如果 DIP-Cache 命中,将由 DIP-Cache 向处理器流水线提供指令.否则处理器从 I-Cache 获取新指令.

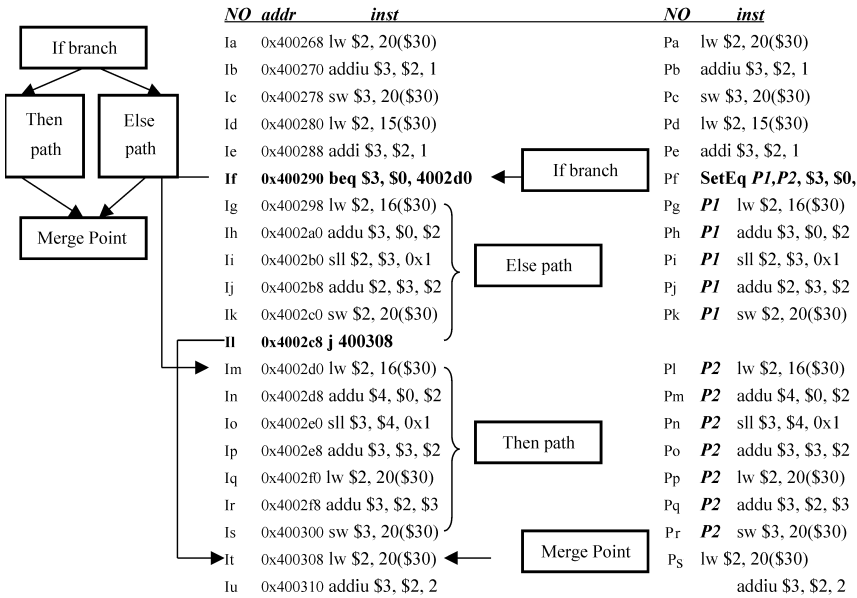


图 2 DIP Lite Trace 形成示例

3.2 DIP 对相关性带来的影响

除了将 If 分支的控制相关转换为针对 Predication 寄存器的数据相关之外, DIP 将两条路径上的指令同时调度进入流水线, 引入了以往不曾存在的数据相关性。

如图 3 右部所示, DIP 在寄存器相关表中为不同的 Predication 建立表目, Predicated Tag 域标识

相应的 Predication 变量如 P1 或者 P2, Seq 域记录在路径中对这个寄存器进行写操作的第一条指令。使用这一扩展可以在指令发射段将两条路径上的指令区别对待, 以提高 Predication 执行的并行度。在 P1 指令准备发射时将不再会考虑相反 Predication 上对 2 寄存器的读写操作。

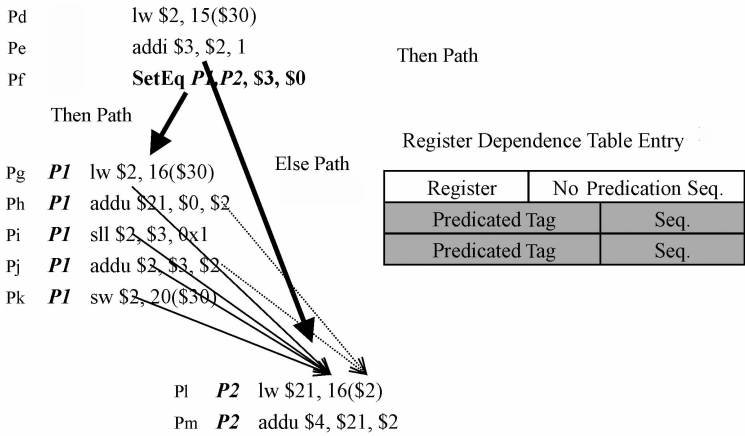


图 3 DIP 中的数据相关

3.3 DIP Trace 管理

从图 1 中看到: DIP 机制不在处理器的关键路径上, 只是在取值、发射和指令提交段增加的 Predication 相关的扩充。在指令取值段, PC 取值地址同时发到传统的指令 Cache 和 DIP Lite Trace Cache, 且 Lite Trace Cache 有更高的优先级。增加数个(32/64)一位的 Predication 寄存器是必需的, 其编码可以保存在 Trace 内部结构中, 无需出现在二进制可执行文件内部。在处理器的提交段, 按照 Predication 的

值来决定指令执行结果是否写入可见的体系结构空间中。同时对于从指令 Cache 中获取的指令, DIP 将在提交段检测其中的 If-分支指令以寻找可能的 If-conversion 机会。

选择 DIP Trace 的起点和终点是 Trace 构建的核心问题。现有的 DIP 遵循传统 Trace Cache 的做法, 选择基本块第一条指令作为起点。从起点到 If-分支的这部分代码无需进行 Predication 变换, 减少这部分代码有助于在 Trace 中容纳更多经过 Predi-

cation 变换的指令,增加 DIP Trace 的有效长度.

DIP Trace 中使用合并点之后的指令作为终点,保证在 Trace 执行完之后有确切的取值目标地址.不足之处在于限制了 If-conversion 使用的范围.事实上如果 Then 路径和 Else 路径上的指令数目足够多,使得在 Trace 中的指令取完之前,Predication 就可以确定,从而可以沿着 Else 或者 Then 路径选择下一步取值地址.该模式可以在分支条件不成立时减少不必要的 Then 路径执行,增加 Predication 应用的机会.在流水线提交段发现 If-分支指令,Compactor 收集来自两个分支目标的指令流之后,DIP 使用额外的 Trace 起点和终点的启发式判定方法来确定是否可以建立 DIP Trace.

DIP 模型是基于对传统 Trace Cache 机制的简化,简化 Trace 的构造和管理更新都位于处理器关键路径之外.由于消除了 Trace 中的分支指令,DIP Trace 无需复杂的状态检查和恢复机制以保证 Trace 执行的原子性,增加了 Trace 内部动态优化的机会,并可以简化指令调度.基于 Trace 的设计也使得 DIP 无需修改二进制代码就可以应用 Predication 方法.

4 DIP 模拟与验证

研究组基于 simplescalar v3.0d 框架^[14]实现了整个 DIP 机制.主要的修改包括在 SimpleScalar 的乱序流水线外侧添加 DIP Lite Trace Cache, If 分支检测等硬件部件.SimpleScalar 流水线的各段也做了相应扩展,以获得更好的 Predication 性能.表 1 给出了 DIP 的基本体系结构(Baseline Architecture)参数.

表 1 基本体系结构参数

参数	说明
指令格式	64 位 Alpha 指令,64 位寻址能力
一级指令缓冲	32KB 容量, 32 Byte 行宽, 2 路组相连, 命中延时为 1 周期
二级数据缓冲	32KB 容量, 32 Byte 行宽, 4 路组相连, 命中延时为 1 周期
二级 Cache	256KB 容量, 64 Byte 行宽, 4 路组相连, 命中延时为 6 周期
DIP Trace Cache	1024 条内部指令存储
分支预测器	默认 bimod 预测器,2048 个单元,90% rate in SPEK2kCint
取值带宽	每周期 4 条指令
指令发射带宽	每周期 4 条指令
分支失效开销	清空流水线需 3 个周期
Reorder buffer	16 个记录并另有 8 个记录用于 load/store 操作
功能单元与操作 延时周期 (total/issue)	4 Int ALU(1/1), 1 Int Mult(2/1)/Div(20/19), 4 memory(1/1), 4 FP Add(2/1), 1 FP Mult(4/1)/Div(12/12) /Sqrt(24/24)

分支预测器是测试的性能关键因素之一.模拟器默认的 Bimod 预测器是性能与开销适中的选择.目前的 DIP 模型没有使用特殊的过滤机制来筛选难预测的分支,为简化设计直接将连续两次预测失败的分支认定为难以预测.DIP Trace 定时清空,以适应分支 Predictability 特性的动态变化.

完整 SPECint2000 测试程序集被用于整个测试过程,为了减少模拟开销和延时,使用了 SIMPOINT 输入集.所有测试程序都在 Alpha EV6/Unix 平台上使用 SPEC Peak 配置编译并链接.以往的 Predication 研究由于 ISA 或者二进制文件内容上的改动,在测试 Predication 性能时使用与原有(未加入 Predication 扩展)结构不同的新可执行文件.而 DIP 中运行的测试程序与在没有加入 DIP 机制的处理器中运行的程序完全相同

4.1 性能模拟

基本的 DIP 模拟中限制 DIP Trace Cache 结构中只能存储 1024 条内部指令,每条 Trace 的长度至多为 16 条指令.性能模拟主要收集以下数据:

(1)整个程序运行过程中 Predication 变换后指令所占的比例.这反映了在经过 GCC 高级优化之后动态 Predication 优化可用的机会;

(2)整个程序的平均 IPC. DIP 消除了 If 分支的控制相关,并且避免了不必要的数​​据相关,这一参数可以评价 DIP 对 ILP 并行性的影响;

(3)整个程序的运行时间(Cycle).消除控制相关的 DIP 可以提高 ILP,但 DIP Trace 中有近一半的指令是被废弃的,实际是无需执行的,最终的性能提升还需要考察运行时间.

如图 4 所示,虽然每个测试程序都经过 SPEC Peak 的静态优化,在动态运行中仍然有使用动态 Predication 的机会.Gzip 中包含太多的间接分支,并且 Bimod 分支预测器在 gzip 上有 97% 的准确度,这使得 DIP 使用的机会不多,但 Gzip 中仍有几条较长的 Trace 可以被 DIP 优化.虽然每个程序用于 Predication 的指令数目各不相同,平均仍有 11.6% 的指令是经过 DIP 优化的.

在图 4~图 6 的测试中,显示了固定大小的 DIP Trace Cache 在 8 路、4 路、2 路相连和直接相连下的性能提升(s16a4 表示 16 sets with 4-way association).测试表明高相连度确实可以减少 Trace Cache 冲突,但在许多程序上差异并不明显.考虑到性能和实现上的复杂度,4 路组相连是比较好的选择,其性能值标记在柱状图顶部.

从图 5 可以看到,在将控制相关转变为 Predication 的数据相关之后,平均的 IPC 提高了 10.3%. 虽然 Predication 中部分指令运行结果最后丢弃,在图 6 中显示相对于没有使用动态隐式 Predication 的处理器结构,DIP 在运行时间上平均有 7.56% 的加速比.

同时比较图 5 和图 6, gap 是动态 Predication

运用最多的测试程序,但是获得最大加速比的却是程序 parser. 这是由于 Predication 中的废弃指令造成的, gap 的 DIP Trace 通常较长,虽然 DIP 提高了 IPC,但是由于 Trace 中近一半指令最终运行结构并没有被使用,使得最多的 Predication 其性能并不是最高. 实际上 gap 是长 Trace 使用次数最多的程序之一.

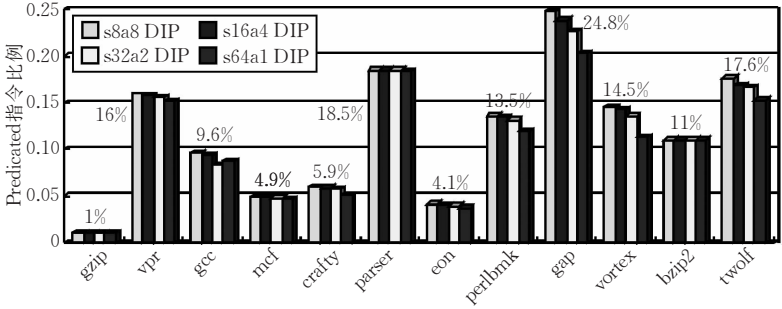


图 4 动态 Predication 后指令在运行中所占比例

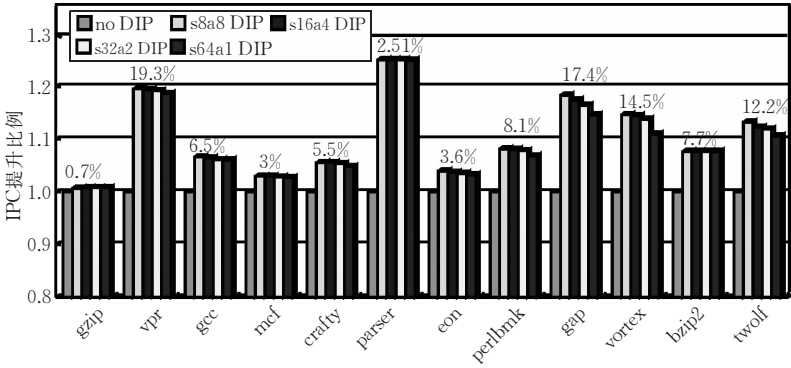


图 5 不同 DIP Trace Cache 相连度下的 IPC 增长

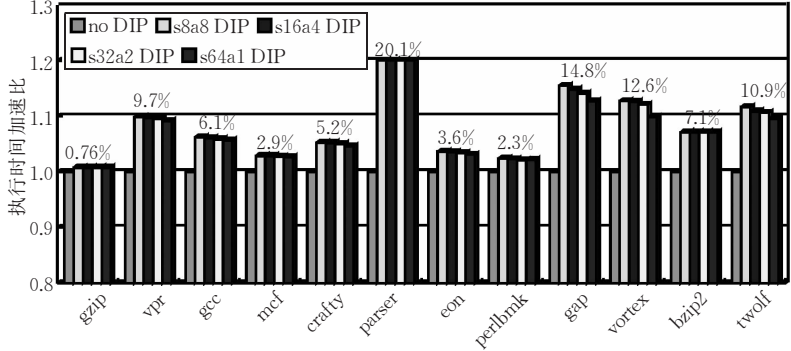


图 6 整个程序运行时间的加速比

4.2 DIP 可扩展行分析

由于 DIP Lite Trace 的替换率远远高于传统的指令 Cache,进一步提高性能的直接方法就是增大 Trace Cache 的容量. 增大的容量有两种使用方法: 一种是增加每条 Trace 的长度,使得更长的 If-Then-Else 语句能够被 DIP 利用;另一种是保持 Trace 长度不变,增加可以容纳的 Trace 数目.

在图 7 和图 8 中 tx 表示一条 DIP Trace 中至多可以保存 x 条内部指令,例如在 $t16$ 配置下,如果

一条经过 Predication 的 If-Then-Else 指令流需要保存的指令数目为 $18(>16)$,这条 Trace 是会被 DIP 产生并保留在 Trace Cache 中的. 我们考察了 x 从 8 增长到 32 的过程,均使用 4 路相连 16 个 Sets,容量从 512 个增加到 2048 个内部指令操作,但均至多保存 64 条 DIP Trace. 图 9 和图 10 考察了在 4 路相连下,Cache Sets 的数目从 16 组增加到 128 组时并行度和执行时间加速比的情况.

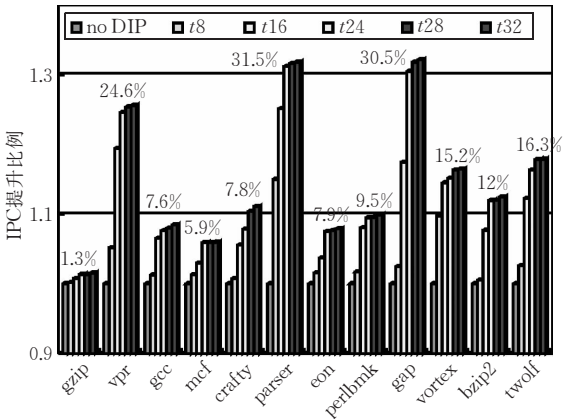


图 7 增大单个 Trace 最大尺寸时并行度(IPC)的增长

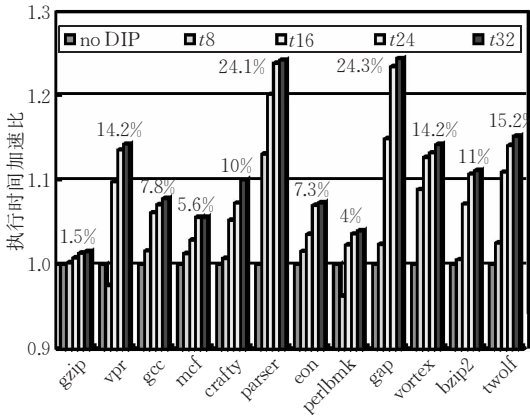


图 8 增大单个 Trace 最大尺寸时的加速比

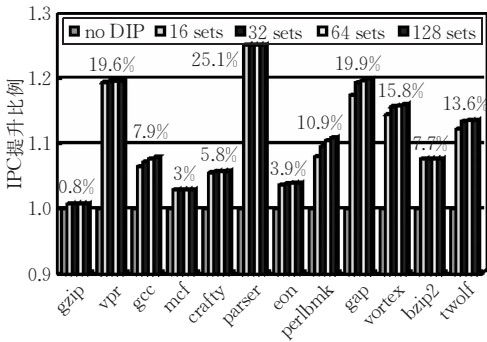


图 9 增加 Trace 数目对并行度的影响

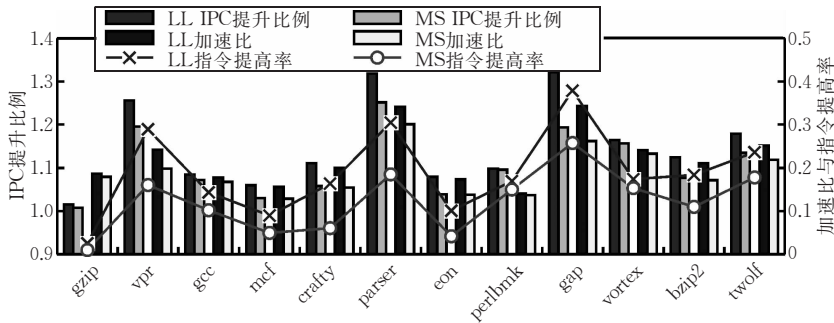


图 11 两种 Trace 增长方式的比较

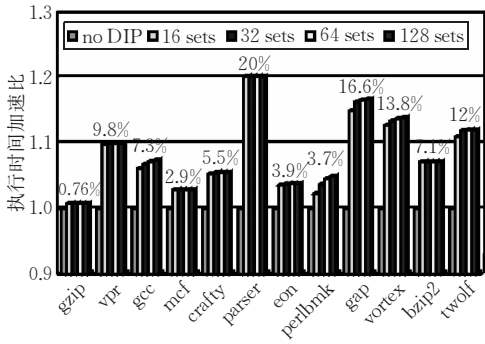


图 10 不同 DIP Trace 数目下的加速比

Trace 长度限制增大后,让一些较长的 If-Then-Else 指令流也能得到 DIP 的优化.但是大多数程序在 t_{24} 之后执行时间上的增长很不明显.这一方面是由于超长的 DIP Trace 比较罕见,另一方面是由于长 Trace 中错误路径上的指令执行阻碍了性能提高.更多的 Trace 数目可以提高程序性能,但是对近一半的测试程序而言,这样的提升低于 1%.这表明程序中一定长度下的 DIP Trace 数目是有限的.要从更多 DIP Trace 中获取性能,加大 Trace 的长度限制比增加 Trace 的数目更有效.图 11 比较了 16KB DIP Lite Trace Cache 在两种增长方式下的性能.

表 2 中的两种配置可以保存相同数量的内部指令,长 Trace Line 相比更多 Trace Sets 更容易产生指令存储空间上的浪费,但却可以保存那些较长的 DIP Trace.并且由于 Trace Cache 要为每一条 Trace 保存相应的 Tag 标记结构,在相同的内部指令存储空间下,长 Trace Line 要更为节省 Tag 空间.图 11 的测试显示,在并行度、执行之间加速和 Predicaion 使用率三个性能指标上,长 Trace Line 都超过了更多 Trace Sets 的扩充方法.因此当有更多的硬件资源来扩充 DIP 存储时,延长 Trace Line 要优于增加 Trace 数量.

表 2 扩大 DIP Lite Trace Cache 的两种配置方式

Name	相连度	组数	行宽(指令数)/条	指令长度	RIMP 容量/KB
LL 增加 Trace 长度	4	16	32	64 位(8 字节)	16
MS 增加 Trace 数	4	32	16	64 位(8 字节)	16

4.3 DIP 深度流水模拟

现代处理器使用超流水技术来增加主频. 更多的流水段数目意味着在分支预测失败时需要更多的周期来清空流水线中的指令. DIP 消除了部分控制流指令, 同时消除了这些指令上分支失败的开销, 保证流水线操作不被中断, 同时减轻了对分支预测器的压力, 这些优势对深流水线结构显得尤其重要. 在

图 12 和图 13 中 px 表示处理流水线在分支预测预测失败时有 x 个指令周期额外开销, 图中显示了不同失效开销下 DIP 模型的性能提升. 柱装图顶部标识的是 $p32$ 下 DIP 的性能提升结果.

从图 12 和图 13 中可以看到对于采用高主频、深流水的现代处理器设计而言, DIP 有更大的使用价值.

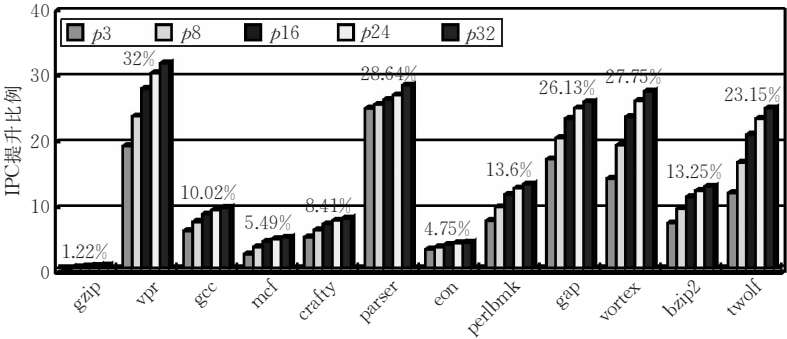


图 12 不同失效开销下 DIP 的并行度增加

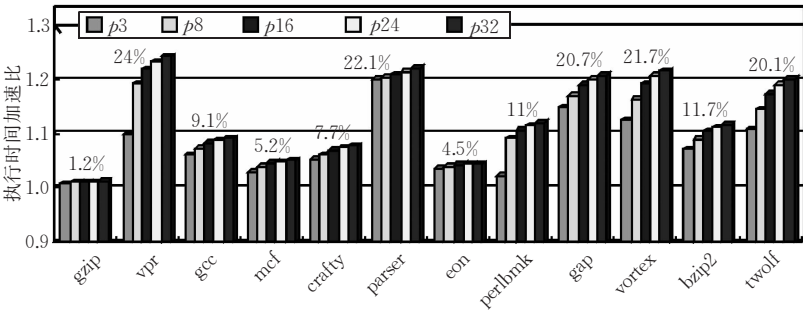


图 13 不同失效开销下 DIP 的执行时间加速比

4.4 其它体系结构参数分析

DIP 依赖于重用保存在 Lite Trace Cache 中经过 Predication 变换的指令来提高性能. 经过 Predication 变换的指令被取入一个指令缓冲队列, 进入乱序执行流水线中. 在前面所有的实验中我们均假设乱序流水线为 4 发射结构, 与之对应取值带宽为每周至多 4 条指令, 保留站(或者 reorder-buffer)中最多有 12 条正在执行的指令.

虽然 Predication 需要执行更多在错误路径上的指令, 进一步的模拟实验表明, 由于单个测试程序的 ILP 有限, 取值带宽和保留站的数目并不构成性能的瓶颈. 当取值带宽从 4 增加到每周取值 32 条指令, 并将保留站数目增加一倍, 同时修改相应的流水线缓冲和同步机制之后, 对整体性能的影响

低于 1%.

DIP 需要专门的 Trace Cache 机制, 其硬件实现开销是设计中不可忽略的因素. 在进一步的研究工作中, 实验结果表明, 相关 Trace 指令重组的存储和控制机制性能要优于同等硬件代价下传统指令 Cache 的性能^[15].

5 结 论

DIP 机制消除了难以预测的 If 分支上可能的失败开销. 在动态运行期间, DIP 透明地识别 If 分支, 构成、保存并重用经过 Predication 优化后的指令流. 在无需改变编译可见的 ISA 结构, 不修改程序二进制文件的前提下, 整体性能提高(>7%)超

过了利用条件执行的部分 Predication 优化的收益(5%). 模拟表明 DIP 结构在高主频、深流水的现代处理器中 DIP 有更大的应用价值.

下一步的工作将集中在更加精确的 Trace 长度和起点、终点控制以及在 Trace 中使用更多指令调度策略. 同时在传统 Trace Cache 中添加 DIP, 而不仅仅将动态 Predication 作为单独 Trace 机制使用, 并考察不同类型 Trace 在执行中的相互影响也是未来工作的重点.

致 谢 同组的曹宏嘉博士建立了动态优化测试平台, 王晓东副教授对实验的选择和论文的撰写提出了许多宝贵的建议, 在此表示感谢!

参 考 文 献

- [1] Sharangpani H, Aurora K. Itanium processor microarchitecture. *IEEE Micro*, 2000, 20(5): 24-43
- [2] Sias J, Hunter H, Hwu W. Enhancing loop buffering of media and telecommunication applications using low-overhead predication//*Proceedings of the 34th Annual International Symposium on Microarchitecture*. Austin USA, 2001: 262-273
- [3] Chuang W, Calder B, Ferrante J. Phi-predication for lightweight if-conversion//*Proceedings of the International Symposium on Code Generation and Optimization*. San Francisco USA, 2003: 179-190
- [4] Jacobson Q, Smith J E. Trace preconstruction//*Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA-00)*. BC, Canada, 2000: 37-46
- [5] Rotenberg Eric, Bennett Steve, Smith J E. Trace cache: A low latency approach to high bandwidth instruction fetching//*Proceedings of the 29th Annual International Symposium on Microarchitecture*. Paris, France, 1996: 24-35
- [6] Sohm Oliver. Variable-length decoding on the TMS320C6000 DSP platform. Texas Instruments Inc. USA: Application Report; SPAR805, 2002
- [7] Kerbyson D J, Hoisie A, Pakin S, Petrini F, Wasserman H J.

A performance evaluation of an Alpha EV7 processing node. *International Journal of High Performance Computing Application*, 2004, 18(2): 199-209

- [8] Klauser A, Austin T, Grunwald D, Calder B. Dynamic hammock predication for non-predicated Instruction set architectures//*Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*. Paris, France, 1998: 278-285
- [9] Mahlke S A, Lin D C, Chen W Y, Hank R E, Bringmann R A. Effective compiler support for predicated execution using the hyperblock//*Proceedings of the 25th International Conference on Microarchitecture*. Portland, USA, 1992: 45-54
- [10] Pnevmatikatos D N, Sohi G S. Guarded execution and branch prediction in dynamic ILP processors//*Proceedings of the 21st International Symposium on Computer Architecture*. IL, USA, 2000: 120-129
- [11] Tyson G S. The effects of predicated execution on branch prediction//*Proceedings of the 27th Annual International Symposium on Microarchitecture*. San Jose, USA, 1994: 196-206
- [12] Mahlke S A, Hank R E, Bringmann R A, Gyllenhaal J C, Gallagher D M, Hwu W. Characterizing the impact of predicated execution on branch prediction//*Proceedings of the 27th Annual International Symposium on Microarchitecture*. San Jose, USA, 1994: 217-227
- [13] Heil T H, Smith J E. Selective dual path execution. University of Wisconsin-Madison, USA: Technical Report; ECE-96-8, 1996
- [14] Austin T, Larson E, Ernst D. SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer*, 2002, 35(2): 59-67
- [15] Tang Y X, Deng K, Cao H. J, Zhou X M. Trace-based runtime instruction rescheduling for architecture extension//*Proceedings of the Embedded Software and Systems*. Hong Kong, China, 2005: 4-15
- [16] Tremblay M, Chan J, Chaudhry S, Conigliaro A W, Tse S S. The MAJC architecture: A synthesis of parallelism and scalability. *IEEE Micro*, 2000, 20(6): 12-25
- [17] Aramon J L, Gonzalez J, Gonzalez A, Smith J E. Dual path instruction processing//*Proceedings of the 16th International Conference on Supercomputing*. New York, USA, 2002: 220-229



TANG Yu-Xing, born in 1977, Ph.D., assistant researcher. His research interests include high performance processor architecture, dynamic binary translation, dynamic compiling and optimization.

DENG Kun, born in 1976, Ph.D., associate professor. His research interests include high performance processor architecture, binary translation and dynamic optimization.

DOU Yong, born in 1965, Ph.D., professor, Ph.D. supervisor. His research interests include high performance computer architecture and microprocessor design.

ZHOU Xing-Ming, born in 1938, professor, Ph.D. su-

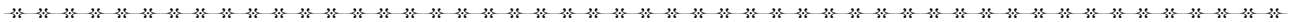
pervisor, member of Chinese Academic of Science. His research interests include high performance computer architec-

ture, parallel and distributed database and CSCW.

Background

This paper’s work belongs to the research of microarchitecture and dynamic optimization. Recently, several international research groups present their own framework and new optimization algorithm. And this research direction, combined with binary translation and virtualization, is just take-off. Researches use different new predictor hardware to solve the branch problem. But always there are some branches which are hard to predict. This paper uses the predication mechanism to solve these hard-to-predict branches. DIP (Dynamic Implicit Predication) combines the simple hardware extension and trace optimization, which can be treat as a novel dynamic optimization method.

This paper’s work is a part of the team works of ARCH group in parallel and distributed processing laboratory (PDL of School of Computer, National University of Defense Technology). After 10 more years research in architecture of high performance computer and microprocessor, ARCH group had many breakthroughs in MT, CMP, Trace scheduling, Dynamic Translation and Optimization. Now the group focuses on design microprocessor based on dynamic translation and optimization. DIP has been used in the architecture design of the proposed processor. And it solves the performance problem of hard-to-predict branch, but still keep the original ISA unchanged.



构建智能平台 打造精品期刊

《智能系统学报》 双月刊

《智能系统学报》由中国人工智能学会与哈尔滨工程大学联合主办,是中国人工智能学会会刊. 中国人工智能学会是我国智能科学技术领域唯一的国家级民间学术团体,隶属于中国科学技术协会. 目前,中国人工智能学会有 22 个专业委员会和工作委员会,100 多位全国著名学者和知名专家担任专业委员会的领导职务,20 多位院士活跃在智能科学技术的各个领域.

《智能系统学报》主要刊登智能科学领域最新的科研成果和高水平学术论文,内容包括人工智能与计算智能、智能控制与决策、智能信息处理、模式识别、专家系统与知识工程、机器学习与知识发现以及人工心理与机器情感等.

中国标准连续出版物号: ISSN 1673-4785 CN 23-1538/TP

邮发代号: 14-190 订阅价: 15 元/期 90 元/年

通信地址: 哈尔滨市南岗区南通大街 145 号 1 号楼 邮编: 150001

电 话: 0451-82534001 82518134 网址: <http://www.tis.net.cn>

电子信箱: tis@vip.sina.com