计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2016.20150317
53(6):1221-1237,2016

一种缓存数据流信息的处理器前端设计

刘炳涛[1,2] 王 达[1] 叶笑春[1] 张 浩[1] 范东睿[1] 张志敏[1]
[1](中国科学院计算技术研究所 北京 100190)
[2](中国科学院大学 北京 100049)
(liubingtao@ict.ac.cn)

A Dataflow Cache Processor Frontend Design

Liu Bingtao[1,2], Wang Da[1], Ye Xiaochun[1], Zhang Hao[1], Fan Dongrui[1], and Zhang Zhimin[1]
[1](*Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190)
[2](*University of Chinese Academy of Sciences, Beijing* 100049)

Abstract In order to exploit both thread-level parallelism (TLP) and instruction-level parallelism (ILP) of programs, dynamic multi-core technique can reconfigure multiple small cores to a more powerful virtual core. Usually a virtual core is weaker than a native core with equivalent chip resource. One important reason is that the fetch, decode and rename frontend stages are hard to cooperate after reconfiguration because of their serialized processing nature. To solve this problem, we propose a new frontend design called the dataflow cache with a corresponding vector renaming (VR) mechanism. By caching and reusing the data dependencies and other information of the instruction basicblock, the dataflow cache exploits the dataflow locality of programs. Firstly, the processor core can exploit better instruction-level parallelism and lower branch misprediction penalty with dataflow cache; Secondly, the virtual core in dynamic multi-core can solve its frontend problem by using dataflow cache to bypass the traditional frontend stages. By experimenting on the SPEC CPU2006 programs, we prove that dataflow cache can cover 90% of the dynamic instructions with limited cost. Then, we analyze the performance effect of adding the dataflow cache to pipeline. At last, experiments show that with a frontend of 4-instruction wide and an instruction window of 512-entry, the performance of the virtual core with dataflow cache is improved up to 9.4% in average with a 28% maximum for some programs.

Key words processor microarchitecture; instruction cache (ICache); dataflow; instruction renaming; dataflow locality

摘 要 为了能够同时发掘程序的线程级并行性和指令级并行性,动态多核技术通过将数个小核重构为一个较强的虚拟核来适应程序多样的需求.通常这种虚拟核性能弱于占有等量芯片资源的原生核,一个重要的原因就是取指、译码和重命名等流水线的前端各阶段具有串行处理的特征较难经重构后协同

收稿日期:2015-04-23;**修回日期**:2015-07-14
基金项目:国家"九七三"重点基础研究发展计划基金项目(2011CB302501);国家"八六三"高技术研究发展计划基金项目(2015AA011204,2012AA010901);"核高基"国家科技重大专项基金项目(2013ZX0102-8001-001-001);国家自然科学基金重点项目(61332009,61173007)
This work was supported by the National Basic Research Program of China (973 Program) (2011CB302501), the National High Technology Research and Development Program of China (863 Program) (2015AA011204,2012AA010901), the National Science and Technology Major Projects of Hegaoji (2013ZX0102-8001-001-001), and the Key Program of the National Natural Science Foundation of China (61332009,61173007).

工作.为解决此问题,提出了新的前端结构——数据流缓存,并给出与之配合的向量重命名机制.数据流缓存利用程序的数据流局部性,存储并重用指令基本块内的数据依赖等信息.处理器核利用数据流缓存能更好地发掘程序的指令级并行性并降低分支预测错误的惩罚,而动态多核技术中的虚拟核通过使用数据流缓存旁路传统的流水线前端各阶段,其前端难协同工作的问题得以解决.对 SPEC CPU2006 中程序的实验证明了数据流缓存能够以有限代价覆盖大部分程序超过 90%的动态指令,然后分析了添加数据流缓存对流水线性能的影响.实验证明,在前端宽度为 4 条指令、指令窗口容量为 512 的配置下,采用数据流缓存的虚拟核性能平均提升 9.4%,某些程序性能提升高达 28%.

关键词 处理器微结构;指令缓存;数据流;指令重命名;数据流局部性

中图法分类号 TP303

因为结构复杂度增长和功耗墙[1]等问题,单核处理器频率和性能的增长在 21 世纪初遇到瓶颈.处理器结构设计师们转而通过增加片上集成的处理器核数来维持处理器符合摩尔定律的性能增长[2].如今多核处理器已经逐渐占据了桌面和移动处理市场.

片上多核处理器(chip multi-processor, CMP)通过发掘程序的线程级并行性(thread-level parallelism, TLP)来提供更高的处理能力.但利用程序的 TLP 有其局限性:1)历史遗留的大量串行程序并没有合理的并行化方案;2)通用处理领域大部分的计算并没有足够的 TLP 可供发掘;3)Hill 与 Marty[3]指出,根据阿姆达法则,随着程序中并行部分的加速,串行部分渐渐成为继续降低程序运行时间的瓶颈.这些都表明处理器的单线程处理能力仍然非常重要.

芯片资源总量固定的情况下,核数增加会提升处理器发掘 TLP 的能力,同时单核分得资源减少,降低处理器发掘指令级并行性(instruction-level parallelism, ILP)的能力.为了在多核设计中兼顾发掘程序的 TLP 与 ILP,学术界提出了动态多核(dynamic multi-core, DMC)技术[3].DMC 处理器可以动态地将数个较弱的原生核(native core, N-Core)合并为一个较强的虚拟核(virtual core, V-Core),增加处理器发掘 ILP 的能力.DMC 技术按应用的需求重构多核的组织形式,打破了处理器发掘 ILP 与发掘 TLP 之间不可调和的边界.采用 DMC 技术的处理器,其适应性和通用性强于固定搭配处理器核的其他类型的多核处理器.

用多个小的 N-Core 通过 DMC 技术动态重构出的 V-Core,其性能通常要弱于用等量芯片资源直接实现的一个大的 N-Core.一个重要的原因在于 V-Core 取得指令的能力弱于其执行指令的能力.首先,取指、译码和重命名等流水线前端各阶段具有串行处理的特点,较难协同工作来成倍增加指令的供给带宽.而流水线后端的发射、执行、写回等阶段因具有并行处理的特点,相对较易经重构后协同工作增加指令处理能力.其次,传统的流水线从存储器中取得的有效指令数目与执行单元处理的有效指令数目相等,前后端耦合工作.流水线的以上 2 个特点,决定了 DMC 处理器中不同粒度的 V-Core 与 N-Core 不能全部做到流水线前后端匹配,并导致了 DMC 处理器不能在各种动态组织形式下都达到极优性能.

为了解决 DMC 处理器中存在的上述问题,我们提出了一种新的处理器前端设计,称作数据流缓存(dataflow cache, DF-Cache),并给出了与之配合的向量重命名(vector renaming, VR)机制.DF-Cache 利用数据流局部性(dataflow locality),其操作的基本单元是经译码和重命名处理后的指令基本块,包含了数据依赖等信息.当指令基本块在 DF-Cache 中命中时,经向量重命名后,流水线可以立即将其指令交付发射队列,旁路了流水线前端各阶段.DF-Cache 的作用体现在如下 2 点:1)DF-Cache 提升了流水线的指令供给带宽并降低了指令供给延迟,进而提升了处理器发掘程序 ILP 的能力,同时降低分支预测错误带来的惩罚;2)DMC 处理器中的 V-Core 利用 DF-Cache 旁路流水线前端各阶段,并不需要取指、译码和重命名等流水线阶段经合并后协同工作.当 DF-Cache 对动态指令的覆盖率很高时,处理器执行的大部分指令并非直接取自前端,流水线前端的处理能力不再限制流水线后端对指令的执行,前后端不匹配的问题得以解决.

对 SPEC CPU2006 中程序的实验证明了 DF-Cache 能够以有限代价覆盖大部分程序超过 90%的动态指令;另外,我们分析了添加 DF-Cache 对流水线性能的影响.实验证明,在前端宽度为 4、指令窗

口大小为 512 的配置下,采用 DF-Cache 的 V-Core 程序执行性能平均提升 9.4%,某些程序执行性能提升高达 28%.

1 相关工作

DF-Cache 通过发掘数据流局部性来提升流水线前端指令供给能力,进而解决 DMC 处理器中 V-Core 前后端能力失配的问题.相关工作分为如下 2 节.

1.1 流水线前端的指令供给与数据流局部性

为了使处理器的指令窗口有足够负载,很多研究着眼于提高流水线前端的指令供给能力.这些研究通常采用的方法是通过使用各种缓存结构来旁路相应流水线阶段的操作,从而达到降低延迟、提高带宽的目的.

传统的指令缓存(instruction cache, ICache)利用计算的时间局部性,实现了容量大、延迟低的指令存储结构,从而降低取指延迟[4].DF-Cache 也是一种指令的缓存,降低指令进入发射队列之前的总的前端延迟.与 ICache 的不同有 2 点:1)DF-Cache 缓存的是动态指令的数据流图,可以直接供后端使用,而 ICache 缓存的是统一编址内存空间内的静态指令;2)DF-Cache 的操作单元是以分支等控制指令分割的指令基本块,而 ICache 操作单元是以地址索引的数条指令组成的缓存行.

Loop Cache[5]利用回跳分支指令发现计算中的循环,然后对循环指令进行缓存降低取指延迟.Loop Cache 在 Intel 的 Nehalem 架构处理器中已有应用[6],Nehalem 中的 loop stream detector 存储译码后的 28 条微指令,可以旁路译码等流水线阶段.相较 Loop Cache,DF-Cache 系统发掘了动态指令流中的数据流局部性,不局限于特殊指令或单个循环,可以加速多个或多重的循环代码的取指.

Trace Cache[7]比 Loop Cache 更进一步.它缓存译码后的指令,用首指令地址与后续数个分支的结果进行索引.如果与多个分支预测的结果匹配,Trace Cache 可以在 1 个周期内跨越多个分支提供指令,大大提升前端带宽.Trace Cache 已在商业处理器中得到应用,Intel 的 Sandy-Bridge 架构处理器中的 decoded uop cache 即是一种 Trace Cache,可以覆盖 80%的处理器动态指令[8].DF-Cache 相比较 Trace Cache 有 2 点改进:1)DF-Cache 相比较 Trace Cache 多缓存了基本块内的指令依赖信息,所以流水线利用 DF-Cache 能够旁路了重命名阶段前半部分消除名相关的工作;2)DF-Cache 的操作单元是指令基本块,相比较 Trace Cache,在存储空间使用效率[9]和与分支预测相关的操作上更具优势.实验证明,DF-Cache 以更小的硬件代价实现了更高的指令覆盖率.

Swanson 等人[10]提出了数据流局部性的概念,将其解释为程序中动态数据依赖的可预测性.为了发掘这种局部性,WaveScalar 提出了新的指令集和微结构.DF-Cache 则仅在微结构上做改进,不需要修改指令集,而对数据流局部性的发掘更有针对性.

Oberoi 与 Sohi[11]提出并行化流水线前端的方案.其中称为 Fragment Buffers 的结构可以缓存取得的指令并重用.根据测试程序的不同,可以使用 16 个这种缓冲区覆盖程序 20%~70%的指令.这种缓冲区的管理方式与 DF-Cache 类似,不过缓存的内容却不同,Fragment Buffers 并没有为了发掘数据流局部性做针对性的设计.

总而言之,提高指令供给能力的研究通常是通过有选择的缓存前端流水线各阶段的结果来加速后续的重复操作,指令覆盖率是衡量其效果的重要指标.首先,DF-Cache 的设计符合该逻辑;其次,DF-Cache 针对发掘数据流局部性而设计,与以往研究不同.结构上体现于 2 点:1)DF-Cache 缓存并重用了重命名阶段的数据依赖信息,据我们所知,这是之前研究所没有的;2)缓存的粒度为指令基本块,提高前端能力的同时可以高效地利用片上资源,与预测执行等技术更直接衔接.

1.2 动态多核技术

在多核处理器善于发掘 TLP 的基础上,DMC 技术合并 N-Core 构建较强的 V-Core 来适应程序对 ILP 发掘的需求.

Core Fusion[12]可以将宽度为 2 的 N-Core“融合(FUSE)”为宽度为 4 或 8 的 V-Core,它提出了详细的重构方案.前端的 Fetch 和 Decode 阶段通过同步措施协同工作,然而 Rename 阶段却没有融合.当处理器数个 N-Core 按照 V-Core 的模式工作时,N-Core 自身的重命名部件被旁路,而由单独的 SMU 负责 V-Core 的重命名和指令分发工作.当融合出的 V-Core 的宽度大于 8 时,SMU 便难以实现.Fetch 与 Decode 的同步开销增长亦不容忽略.Core Fusion 融合得出的 V-Core 前端不具可扩展性,导致 Core Fusion 的能融合的 N-Core 数目和获得的 V-Core 的宽度受到限制,V-Core 的性能也要弱于占用等量芯片资源的 N-Core.

CLP(composable lightweight processor)[13]通过特殊的广播与收集措施完成 V-Core 中指令的取指与

提交. 因为没有核间共享的结构,其可扩展性要强于 Core Fusion. 但是 CLP 依赖非传统 ISA 的 EDGE 指令集来实现分布式的指令处理结构,需要编译器的支持,其适应性不如 DF-Cache.

WiDGET[14]区别线程管理和计算资源分配,将计算资源从流水线中解耦合出来形成资源池,按应用需求分配计算资源给硬件线程形成 V-Core. 该 DMC 技术只重构计算资源,流水线的串行部分在形成 V-Core 时没有任何改变,其目的在于实现与功耗成比例的计算,其 V-Core 不具有可扩展性,流水线的宽度没有改变,所以无法动态调整芯片发掘 TLP 和发掘 ILP 的能力.

Dynamic Core Morphing[15]通过动态组合 2 个非对称核后端的计算资源,使其适应应用的需求,达到更好的功耗计算比. 该 DMC 技术重构的多种 V-Core 不能都达到前后端匹配,影响处理器总体能达到的性能.

从以上研究可以看出,目前已有的 DMC 技术并没有提出合理的流水线前端的重构方案. Core-Fusion 的前端融合缺乏可扩展性,CLP 为了构建 V-Core 借助于非传统 ISA,WiDGET 和 MorphCore 等则不考虑对流水线前端做动态改变. DF-Cache 加入流水线后可以部分替代传统的流水线前端,通过发掘数据流局部性为流水线后端提供指令,从而解决 DMC 中各种 V-Core 前端能力不足的问题.

2 数据流缓存的原理与扩展的流水线结构

以存储程序为特征的冯-诺依曼体系结构将程序指令存放在统一编址的存储空间内,我们称之为静态指令. 程序要被处理器执行,首先需要从内存中将静态指令取出,然后经过译码和重命名变为适合执行单元使用的微指令的形式,我们称之为动态指令. 提高静态指令到动态指令的转换效率和提高动态指令的执行效率是冯氏结构的 2 个基本问题. DF-Cache 尝试解决上述第 1 个问题. 我们分别从计算模型和处理器微结构 2 个角度阐述 DF-Cache 的原理及优势.

1) 从计算的角度看. DF-Cache 充分利用计算的数据流局部性,提高了处理器前端指令供给能力.

静态指令与动态指令的数量有量级上的差距,从侧面说明动态指令流有很好的时间局部性. 利用这种局部性,从内存到发射队列间的整个指令转换过程有可以优化的环节. 比如 ICache,它通过缓存并重用从内存中取得的指令,使得处理器执行的动态指令数量远大于从内存中取得的静态指令数量. DF-Cache 更进一步将缓存从取指阶段推进到重命名阶段之后,即指令转换的最后一个环节. 而 DF-Cache 利用的数据流局部性,则是计算的时间局部性在传统的 ISA 格局下一种特殊的表现形式.

比较程序存储的静态指令代码和执行的动态指令流,我们可以发现,以分支等控制指令为边界划分的指令基本块在内容上是一一对应的. 单个静态的指令基本块通常对应 1 个或数个动态的指令基本块实例,而这些实例内的指令依赖等信息完全相同. 所以动态指令流中地址连续的指令之间的重命名结果可以缓存并重用,这就是数据流局部性的具体体现. 着眼于此,我们的缓存结构将由控制指令分割的指令基本块作为操作单元. DF-Cache 缓存指令基本块内的各种信息,包括译码后的微指令和微指令间依赖等. 指令转换工作剩余部分则还有分支等控制指令的预测和验证,以及基本块间的操作数依赖表示. 这 2 点是指令转换过程中没有可重复性的部分. 当指令基本块在 DF-Cache 中命中时,通过复用存储的基本块内数据流信息,传统处理器前端的取指、译码和重命名的解决名相关部分等都被旁路,降低了前端的延迟,提高指令供给带宽.

2) 从微结构的角度看. DF-Cache 加入流水线后,解耦合了流水线的前端和后端. 这解决了 DMC 技术中前端难以重构和前后端能力失配的问题.

DMC 技术[12]中多个 N-Core 的前端很难通过合并达到更高的指令供给能力. 流水线的后端天然有并行处理的特点,发射队列与执行单元的动态组合问题可类比于传统单核处理器中分簇(clustered)后端的设计问题. 然而流水线的前端负责指令转换,串行处理的特点导致其很难协同工作. 尤其是重命名阶段,针对指令的重命名多采用级联比较器和多路器的设计[16],串联多个重命名部件导致一个超长的重命名延迟是无法接受的. 所以像 Core Fusion 中的 SMU[12]部件,重构的 V-Core 常另外设计结构替代单核内的重命名部件. 即便如此,因为指令重命名串行处理每条指令的特点,目前已知的前端设计方法不具有可扩展性,很难匹配 V-Core 的流水线后端增长的带宽需求.

DF-Cache 作为一种存储主体基于 RAM 的结构,通过整合多个核的 DF-Cache 可以实现更高的动态指令覆盖率,提供更高的读写带宽,具有可扩展性. 利用 DF-Cache 进行指令转换的带宽和延迟由向量化重命名决定,它与基本块的数量相关,而不与

指令数量直接相关.另一方面,DF-Cache 位于流水线前端与后端的结合处,DF-Cache 命中时取指、译码和重命名阶段被旁路,这些流水线阶段的能力并不直接决定流水线转化指令的能力.如果对动态指令流有很高的覆盖率,DF-Cache 可以满足后端对指令供给的需求.这样前后端的处理能力不匹配对 V-Core 的整体效率没有影响.另外,不同于执行单元数量和流水线宽度的设计考量,DF-Cache 不存在过度设计,它所要考虑的是对动态指令的覆盖率,与程序的 ILP 大小、流水线的执行能力强弱不直接相关.

添加 DF-Cache 后的流水线结构如图 1 所示.虚线上方为 DF-Cache 对一个传统流水线的扩展.DF-Cache 命中的指令基本块,不经过取指和译码阶段,而后端的执行和提交阶段为 2 条流水线所共用.

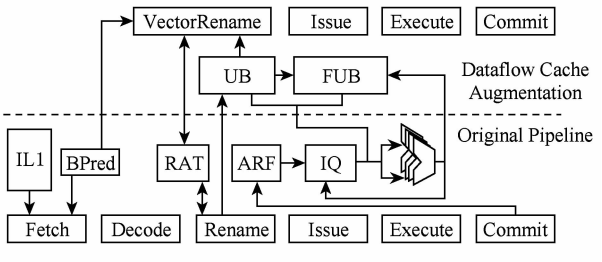


Fig. 1 Pipeline augmented with DF-Cache.

图 1 DF-Cache 扩展的流水线

从图 1 可以看出,如果 DF-Cache 覆盖了动态指令流,那么计算就在 DF-Cache 和执行单元之间构成了封闭循环,指令快速地填充到发射队列,后端执行单元被最大限度地使用.

基本指令块被缓存的信息集合称为 config.指令转换时如果命中这个 config,后端分配物理寄存器和发射队列空间给基本块的实例,我们称其为属于该 config 的 1 个帧(frame).config 对应静态的指令基本块,而 frame 对应动态的指令基本块.config 存放微指令的编码以及基本块内指令间的相对数据依赖等信息,frame 存放指令的操作数等信息.

DF-Cache 含有 2 个组件:微指令缓冲区(uop buffer, UB)和帧更新缓冲区(frame update buffer, FUB),分别存放 config 和 frame,如图 1 所示.

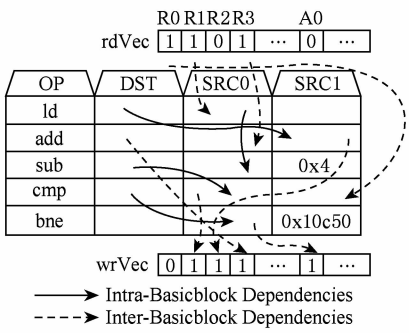
DF-Cache 的功能主要有 3 个:1)UB 命中时,它替代取指和译码阶段以及重命名阶段的部分功能,完成指令的获取;2)与发射队列(issue queue, IQ)并列的 FUB,与指针追踪部件配合,完成指令的唤醒和选择;3)指令从 UB 到 FUB 的填充,由 VR 部件负责.本节分别介绍这 3 个功能的实现.

2.1 微指令缓冲区 UB

UB 存储译码后的指令基本块,逻辑功能如图 2 所示,物理实现示意如图 3 所示.

PC	Opcode	Operands		
0x10c50	ld	R2	R1	
0x10c54	add	R3	R3	R2
0x10c58	sub	R1	R1	0x4
0x10c5c	cmp	A0	R1	R0
0x10c60	bne	A0	0x10c50	

(a) Code



(b) Dataflow

uopid	op	DST(BK0)		SRC0(BK1)		SRC1(BK2)	
		bid: *uopid*	arch	*bid*: *uopid*	arch	*bid*: *uopid*	arch
0	ld	2: 1		1: 2	rd R1		
1	add		wr R3		rd R3		wr R2
2	sub	1: 3				I-0x4	
3	cmp	1: 4			wr R1		rd R0
4	bne				wr A0	I-0x10c50	

(c) Data in main-UB

idxPC	0x10c50	brPC	0x10d60
LRUPtrs		mainUBPtr	
uopCNT	5	refCNT	0
rdVec	1101…0…	wrVec	0111…1…

(d) Data in mega-UB

Fig. 2 The logic function and organization of UB.

图 2 UB 的逻辑功能与结构

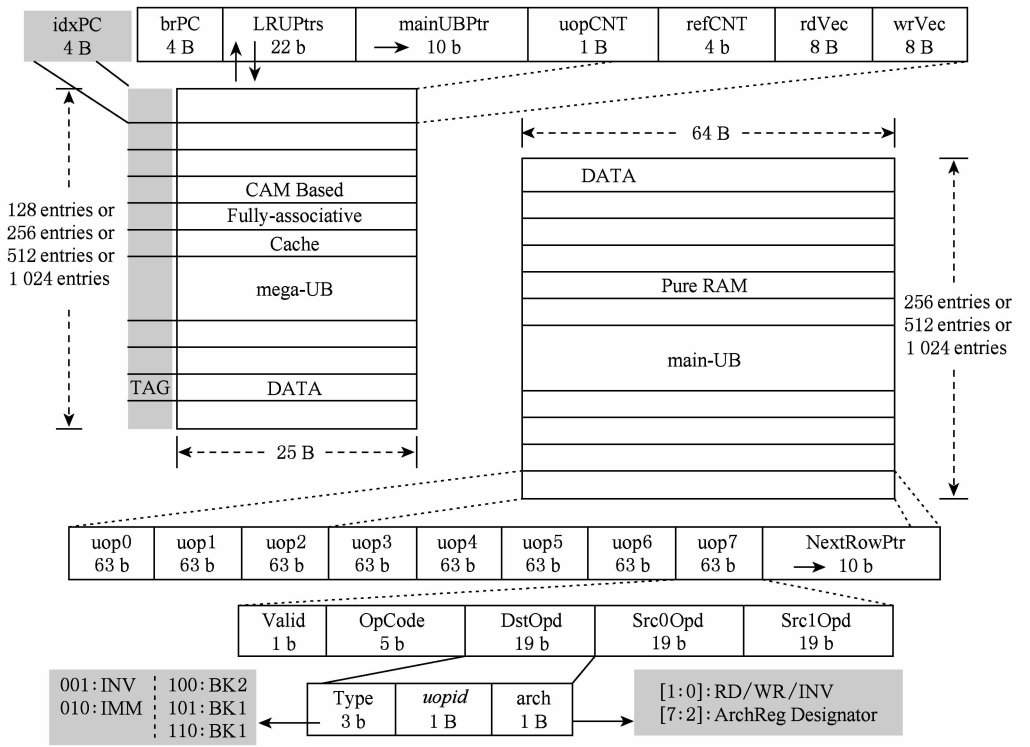


Fig. 3 Implementation of UB.

图 3 UB 的逻辑物理实现

UB 由主微指令缓冲区(main-UB)和宏微指令缓冲区(mega-UB)组成. main-UB 存储微指令相关的信息,比如微指令编码、微指令间寄存器依赖等,如图 2(c)所示;main-UB 由 RAM 实现,mega-UB 存储基本块的宏信息,比如首指令的 PC 值(idxPC)、最后的控制指令的 PC 值(brPC)等,如图 2(d)所示;mega-UB 是基于 CAM 结构的全相连 Cache,用 idxPC 做索引标签(tag),UB 的各字段组成及大小评估如图 3 所示.

读取缓存的微指令是 UB 在流水线中的关键路径操作,主要有 2 步:1)通过 idxPC 检索 mega-UB,获得微指令在 main-UB 中的存储地址(mainUBPtr);2)在 main-UB 中取出微指令. main-UB 对指令基本块分行存储,各行之间通过指针组成链表. 只有首行的微指令读取需要检索 mega-UB,后续的微指令通过追踪指针(NextRowPtr)访问 main-UB 即可获得.

图 2(a)展示了地址从 0x10c50 到 0x10c60 的指令基本块. 首指令地址 0x10c50 作为该指令基本块的索引,通过检索 mega-UB 的 idxPC 字段确认 DF-Cache 是否命中,并获得基本块在 main-UB 中的首行存储地址,进而在 main-UB 中读取微指令. 同时,0x10c60 作为 brPC 从 mega-UB 中读出,分支预测器(BPred)使用 brPC 做分支预测,获得下一个基本指令块的 idxPC,继续检索 mega-UB,循环往复.

DF-Cache 的取指方式与 ICache 相比,有 3 个特点值得注意:

1) 基于传统的取指逻辑,BPred 必须在 ICache 取到控制指令后才可以利用其 PC 值作出分支预测. 而 mega-UB 与 BPred 的互动不需要访问 main-UB. 这缩短了分支预测之间的间隔,理论上每个周期都可以从新的指令基本块开始取指,多个指令基本块的微指令读取周期可以重叠(overlap).

2) 获得同样多的指令,mega-UB 的访问次数不会多于 ICache. 因为 mega-UB 只有遇到控制指令时才进行检索,而 ICache 在这种情况下也必须进行 tag 比较. 同时 ICache 在遇到 Cache 行分界时都要进行检索,其访问次数应大于 mega-UB.

3) mega-UB 与 BPred 以及后续提到的向量重命名逻辑三者配合,可以比较自然地实现基于检查点(checkpoint)的预测执行. 每次 mega-UB 的检索,都是进行了一次分支预测,产生了一个新的 frame 到 FUB 中,同时进行一次向量重命名. 此时建立 checkpoint,存储当前节点的结构寄存器映射以及一个最新的 frame 相关信息. 将来控制指令执行后,如果发现分支预测错误可以进行回卷. 首先丢掉(flush)后续预测执行的 frame;错误的分支预测指令所在的 frame 因为没有提交(commit),其所在的 config 引用计数(refCNT)就不为 0,从而 config 也

不会被替换掉,利用 checkpoint 中的信息做现场恢复,利用正确执行方向的 idxPC 检索 mega-UB 恢复执行.

图 2(b)展示了我们如何利用依赖指针(dependence pointer)[17-18]表示指令的操作数依赖.依赖指针是前向的(forward)数据流依赖表示,例如指令 B 依赖指令 A,那么在指令 A 的操作数寄存器旁并列存放着指向指令 B 的源操作数物理寄存器的指针,当指令 A 的操作数更新后,指令追踪部件负责将操作数传递给指令 B.指令依赖指针与 RAM 的实现相结合用来实现可扩展的发射队列.指令基本块作为整体,对结构寄存器文件的读写集合存储在结构寄存器读写向量内,用来做向量重命名.

图 2(c)展示了基本块内的微指令和依赖信息在 main-UB 中的存储情况.FUB 分为 3 个块(bank),依次存储微指令的 2 个源操作数和 1 个目的操作数,bank 由 *bid* 索引.指令间的相对依赖由 *uopid* 和 *bid* 组成的依赖指针链表示.arch 项记录了这个操作数是来自或写入哪个结构寄存器.

图 2(d)列举了 mega-UB 的字段.idxPC 用来做缓存的检索,判断缓存是否命中;brPC 提供给分支预测器使用;LRUPtrs 为记录基本块访问的时间先后顺序的双向链表指针,用来维护 LRU(least recently used)信息;mainUBPtr 是基本块在 main-UB 的首行地址;uopCNT 记录了基本块内的微指令数目,这个字段用来在 FUB 中分配连续的空间;refCNT 记录属于此 config 的 frame 实例数量;rdVec 和 wrVec 这 2 个字段用来做向量重命名.

DF-Cache 未命中时,由 ICache 负责取指,同时 DF-Cache 采集指令基本块的信息,并逐步完成基本块内的指令依赖表示.整个基本块取指译码完成后,尝试在 DF-Cache 中存储该指令基本块.

DF-Cache 满时,包括 2 种情况:mega-UB 溢出和 main-UB 溢出.DF-Cache 使用 LRU 策略进行替换,在 mega-UB 中 LRUPtrs 组成的双向链表尾逐个向前清理,直到 mega-UB 中有空闲条目,main-UB 中有足够空行.

DF-Cache 中条目的构建与替换不是流水线的关键路径操作,限于篇幅限制不再继续展开.

对动态指令的覆盖率决定了 DF-Cache 的实际应用效果.着眼于应用的数据流局部性特征,我们对不同设计参数配置的 UB 进行实验,考察 DF-Cache 的指令覆盖率.同时,DF-Cache 的面积、功耗与延迟作为约束,也决定了其是否有实现的价值.详细分析见 3.1 节的实验部分.

2.2 向量重命名 VR

传统的重命名部件的操作对象是单条指令.1)为每条指令分配存储操作数的物理寄存器和发射队列空间;2)解决名相关依赖,比对指令的操作数编码和寄存器重命名表(register aliasing table, RAT),获得每个操作数依赖的物理寄存器编号或者从寄存器文件中读出操作数的值,并更新 RAT;3)将指令和操作数等写入发射队列.

向量重命名与传统重命名结果相同,但过程不同.我们将指令的寄存器依赖分为 2 类:位于基本块内的和位于基本块间的依赖.1)DF-Cache 存储了基本块内的依赖,只要将 config 中存储的依赖关系加上 frame 在 FUB 中的偏移,即可得到绝对依赖关系;2)向量重命名部件对数个基本块的读写向量进行处理,分析基本块间的依赖得出操作数传递路径,并将其补充到发射队列里,完成基本块间的指令依赖表示.指令进入发射队列后,不再有块内和块间的区分,被独立地调度执行.

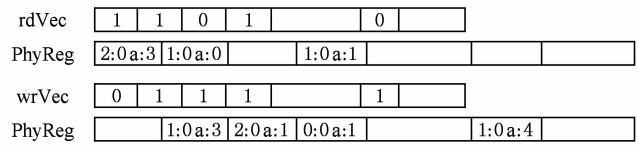
DF-Cache 连续命中时,向量重命名部件可在 2 个流水的时钟周期内完成数个基本块的重命名和发射队列的填充操作.

周期 1.VR 完成基本块内的重命名,并为基本块间的重命名做准备.VR 的操作对象是被分支预测选中且在 DF-Cache 中命中的数个指令基本块.首先,每个 frame 获得唯一的 *frameID* 并从 main-UB 向 FUB 中复制微指令,即在 FUB 中分配一块连续的物理寄存器和发射队列空间并对应 *uopid* 复制基本块内指令的相对依赖指针和立即数操作数.其次,main-UB 中的结构寄存器读写字段也被读取,VR 将表示块间依赖的读写向量和对应的物理寄存器的地址存储到向量重命名部件内,为第 2 周期的工作做准备.图 4 展示了第 1 周期操作的结果.

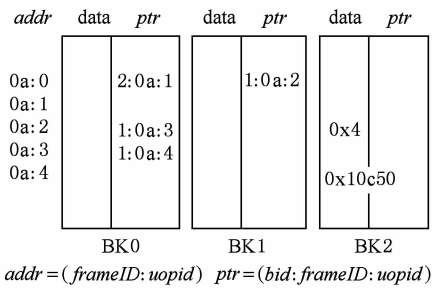
周期 2.VR 完成基本块间的重命名.利用读写向量和对应的物理寄存器地址,VR 将块间寄存器依赖补充到指令窗口中,同时更新 RAT.更新依赖时,结构寄存器的写向量的物理寄存器依赖指针地址作为更新地址,对应的读向量的物理寄存器地址作为更新值.我们以图 4(a)的基本块为例子,假设分支预测器给出连续 3 个 0x10c60 的分支都是预测命中,那么我们要将该 config 的 3 个 frame 更新到指令窗口中.假设 3 个 frame 在 VR 的第 1 周期内分得的 *frameID* 分别是 0x0a,0x0b,0x0c.第 2 周期的向量重命名过程和指令窗口更新结果如图 5 所示.

uopid	op	DST(BK0)		SRC0(BK1)		SRC1(BK2)	
		bid: *uopid*	arch	*bid*: *uopid*	arch	*bid*: *uopid*	arch
0	ld	2:1		1:2	rd R1		
1	add		wr R3		rd R3		wr R2
2	sub	1:3				I-0x4	
3	cmp	1:4			wr R1		rd R0
4	bne				wr A0	I-0x10c50	

(a) Data in main-UB



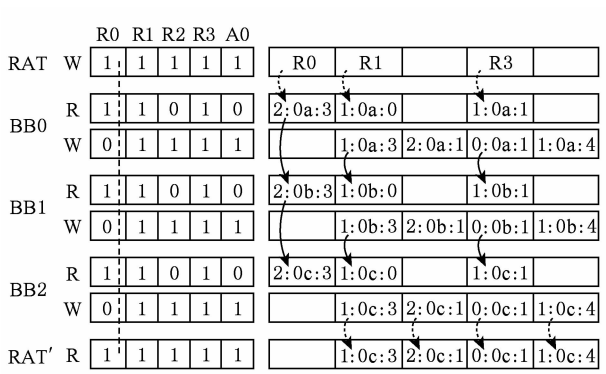
(b) Boundary physical register address



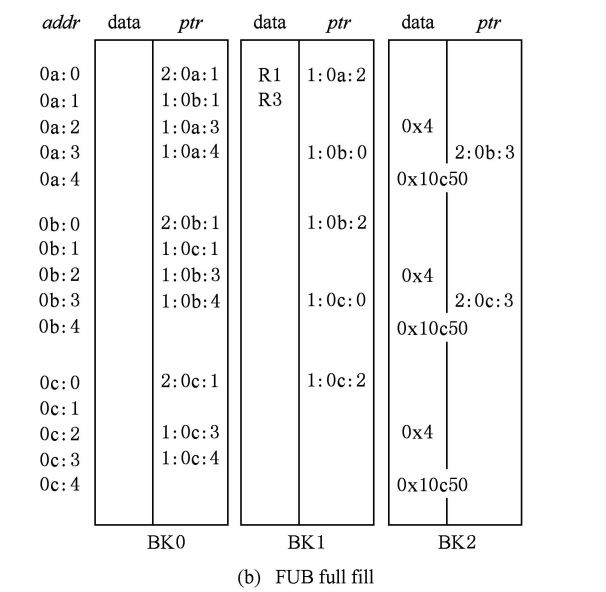
(c) FUB partial fill

Fig. 4 First cycle of VR.

图 4 向量重命名第 1 周期



(a) Vector renaming



(b) FUB full fill

Fig. 5 Second cycle of VR.

图 5 向量重命名第 2 周期

向量重命名要求获得每个 frame 的结构寄存器读写向量,并且有对应物理寄存器的地址.依赖的发现和表示过程举例说明.首先是 R0 结构寄存器,如图 5(a)所示的虚竖线,从上到下依次为 1 10 10 10 1.VR 从读向量为 1 的位置入手,其获得的操作数来自离它最近的前面一个标记为 1 的位置①.比如 BB0 的读向量的 R0 位置的 1 和对应的物理寄存器地址 2:0a:3 说明,BB0 的第 4 条微指令的第 2 个源操作数来自 RAT 记录的 R0.如果 RAT 中 R0 位置是有效的,那么将结构寄存器的值拷贝到 FUB 的 BK2 的 0a:3 地址的 data 字段;如果 R0 的位置是无效的,而是记录 1 个物理寄存器的地址,物理寄存器地址(2:0a:3)拷贝到 R0 所记录的物理寄存器的依赖指针里,延续依赖链.再比如 BB1 的读向量的 R1 位置的 1 和对应的物理寄存器地址 1:0b:0 说明,BB1 的第 1 条微指令的第 1 个源操作数来自 BB0 的第 4 条微指令的第 1 个源操作数(1:0a:3),将(1:0b:0)写入到(1:0a:3)位置的物理寄存器的依赖指针内,延续依赖链.最后,见图 5(a)中的 RAT′,将每个最后写结构寄存器的物理寄存器的地址更新到 RAT 表中.更新后的 FUB 见图 5(b).

VR 大部分的逻辑操作可以向量化地并行进行.以图 5(a)的向量重命名的位矩阵为例.VR 可以分别操作各个结构寄存器对应的纵向位向量而互不影响,而针对单个横向的读位向量分别查询距离最近的 1 并负责更新 FUB 中的依赖,也可以并行进行.假设 1 个周期可以预测并写入到 FUB 中 n 个基

① FUB 采用单链的指针追踪实现唤醒和选择,使得发射队列可扩展.如果采用广播的形式完成唤醒,要考察标记为 1 的最近的写,通过减少参数传递的跳数,缩短指令唤醒的延迟.

本指令块,而结构寄存器有 m 个,那么需要$(n+1)m$个判断位是否为 1 并查询长度为$(2n+1)$的向量中最后的 1 的逻辑部件.硬件开销基本固定,与各基本块内的指令数和单周期内重命名的总指令数不直接相关.

2.3 帧更新缓冲区 FUB

FUB 的设计类似于 RUU[19],流水线并没有单独的物理寄存器文件,物理寄存器的地址与 frame 在 FUB 中的地址信息 *frameID* 相关联,FUB 除了存储操作数的依赖信息并调度微指令外,还存储操作数.

与 RUU 不同的是,FUB 由 RAM 而不是 CAM 组成,数块 RAM 加上指令追踪部件实现可扩展的发射队列.我们采用了类似 ForwardFlow[18] 中 Dataflow Queue 的设计.

在 2.2 节中已经提到了 FUB 的部分实现,FUB 由 main-FUB 和 mega-FUB 组成.main-FUB 作为指令窗口的组成部分①,与依赖追踪部件协同完成指令的唤醒(wakeup)和发射(issue).mega-FUB 用 *frameID* 索引,记录属于 frame 整体的信息,比如 frame 属于哪个 config 以及该 frame 还有多少条 uop 没有发射等,这些信息用来协助管理 UB 和 FUB.

依赖追踪部件沿依赖链逐个传递操作数,同时它负责唤醒指令并提供可以发射的指令给选择逻辑(select logic).在更新操作数后,追踪部件会将指令的 2 个源操作数读出,如果操作数都准备好,负责将这条指令交给选择逻辑等待发射,发射如果不能立刻完成,依赖追踪部件会阻塞在这条指令上,直到发射完成.在更新操作数后如果另外 1 个源操作数没准备好,依赖追踪部件会继续沿依赖链传递操作数.这种指令依赖追踪与 RAM 结合的设计保证发射窗口有可扩展性,实现较大的指令窗口同时对性能没有太大影响[17-18,20].

每个指令追踪部件皆负责一段 RAM 的更新,不属于其管理范围的操作数会传递给负责该操作数的指令追踪部件处理.每个指令追踪部件有辅助的 FIFO 缓存属于该部件但无法立刻追踪的操作数.RAM 的分段越多,FUB 的读写带宽越大.总的指令追踪部件数量与 RAM 的分段数相同,并且决定了 FUB 单周期内能提供给选择逻辑进行发射的微指令数.

3 实验结果与分析

3.1 数据流缓存的动态指令覆盖率与实现代价评估

DF-Cache 能以有限的代价覆盖足够的动态指令才有应用价值.我们首先考察 DF-Cache 在不同设计参数下对指令的覆盖率,然后分析 DF-Cache 的访问延迟、面积和功耗,并与 32KB 的 ICache 进行比较.

我们使用 SPEC CPU2006[21] 中的部分程序作为测试程序集,使用 ref 等级的输入数据,如表 1 所示.我们用 ESESC 模拟器[22] 运行测试程序集,每个程序我们运行 20 亿条动态指令,分析 DF-Cache 对动态指令流的覆盖率.

Table 1 Selected SPEC CPU2006 Benchmarks And Input

表 1 选定的 SPEC CPU2006 测试程序集与输入

Program	Input
401. bzip2	input. source 280
403. gcc	166. i -o 166. s
429. mcf	inp. in
433. milc	< su3imp. in
444. namd	--input namd. input --iterations 38 --output namd. out
445. gobmk	--quiet --mode gtp< 13x13. tst
450. soplex	-s1 -e -m45000 pds-50. mps
453. povray	SPEC-benchmark-ref. ini
456. hmmer	nph3. hmm swiss41
458. sjeng	ref. txt
462. libquantum	1397 8
464. h264ref	-d foreman_ref_encoder_baseline. cfg
470. lbm	3000 reference. dat 0 0 100_100_130_ldc. of
471. omnetpp	omnetpp. ini
473. astar	BigLakes2048. cfg

在使用 CACTI 对不同参数的 DF-Cache 设计的面积、功耗和延迟进行分析后,我们认为 mega-UB 容量为 256 个条目、main-UB 容量为 512 个数据行的配置较为合理,并将其作为 3.2 节实验的 DF-Cache 配置.该配置下 DF-Cache 对测试程序的平均动态指令覆盖率大于 90%,其功耗、面积和访问延迟都要优于 32 KB 的 ICache.

① 另一部分是标量重命名所使用的发射队列,如图 1 中 IQ 所示,同样采用类似 Dataflow Queue 的设计.除了填充指令方式不同,IQ 与 FUB 在指令唤醒和选择上是统一的.

3.1.1 数据流缓存的动态指令覆盖率

DF-Cache 采用 LRU 替换策略,指令基本块能否在 DF-Cache 中持续命中,取决于其数据流局部性.数据流局部性的衡量标准即 1 个静态指令基本块的相邻 2 个动态实例之间间隔的指令基本块种类数.举例说明,假设 mega-UB 的条目数为 4,遇到某个 config 的第 1 个 frame 时,该 config 被放到队列头,如果该 config 的第 2 个 frame 到来前遇到的基本块不超过 3 种,那么该 config 没有被替换,第 2 个 frame 仍能够命中.程序中属于相同 config 的 frame 之间间隔的 config 种类少,表明该程序有好的数据流局部性,用较小的 mega-UB 容量即可覆盖较多的动态指令.mega-UB 中条目越多则动态指令覆盖率越高.另外,mega-UB 是基于 CAM 的全相连 Cache,条目增多会增加功耗并增大访问延迟.

mega-UB 中每个条目对应单个指令基本块,而 main-UB 中每个数据行存储 8 个译码后的微指令,main-UB 中每个数据行只能属于单个指令基本块,所以 main-UB 的行数应该大于等于 mega-UB 的条目数.

参照图 3,mega-UB 与 main-UB 的宽度是确定的,分别为 25 B 与 64 B.需要考量的是部件的高度,即 mega-UB 中的条目数与 main-UB 中的数据行数.mega-UB 的实现分为 4 种:CAM128,CAM256,CAM512,CAM1024.CAM 代表了其硬件结构,数字代表了其含有的条目数.main-UB 的实现分为 3 种:RAM256,RAM512,RAM1024.两者叉积构成 DF-Cache 的各种配置,例如 CAM128 与 RAM256 的组合记为 mega-UB128 main-UB256.图 6 展示了有代表性的 DF-Cache 配置对测试程序的覆盖率结果.

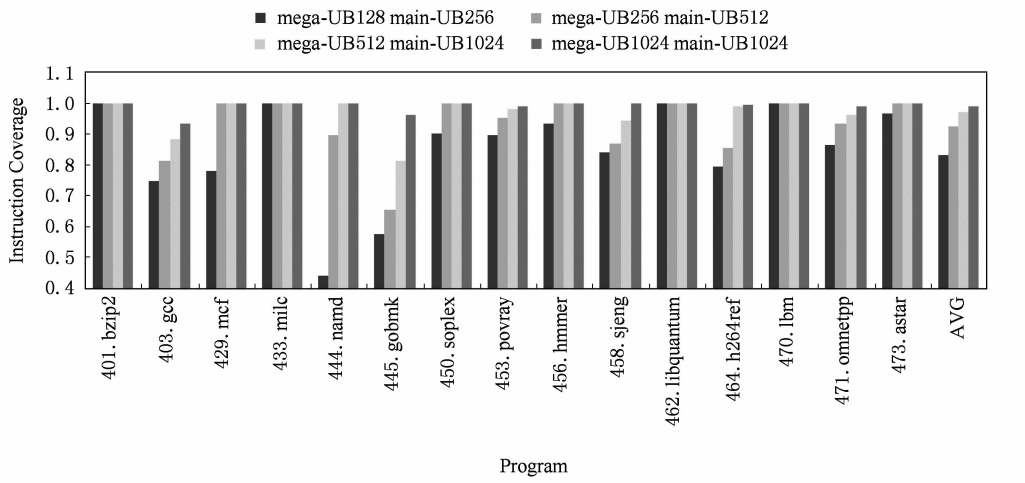


Fig. 6 Instruction coverage by DF-Cache of different configurations.

图 6 不同配置 DF-Cache 的指令覆盖率

从图 6 可以看出,不同的程序表现出不同的特征.第 1 类程序包括 gcc,namd,gobmk,h264ref 等,这些应用的数据流局部性较差,对 mega-UB 的容量敏感.体现出这些应用的计算比较复杂,其控制流很可能与输入数据相关,通常其分支预测准确性也较低.第 2 类程序包括 milc,libquantum,lbm,astar 等,这些程序数据流局部性好、控制流简单,较小的 mega-UB 即可覆盖接近 100%的动态指令.第 3 类程序包括其他的测试程序,当 mega-UB 的容量超过某值后,即可有较高动态指令覆盖率.从设计空间探索的角度看,我们所取的 4 个配置样本点无法完全满足第 1 类程序的数据流局部性要求,能满足第 2 类程序的数据流局部性要求,能覆盖到第 3 类程序的数据流局部性临界点.

观察经过几何平均的覆盖率,可以看到 mega-UB256 main-UB512 的 DF-Cache 配置对动态指令的覆盖率超过了 90%.该数据已经超过了 SandyBridge 中 decoded uop cache 的 80%的指令覆盖率.

3.1.2 数据流缓存的实现代价分析

CACTI 通过对存储器件进行建模可以分析其面积、功耗和访问延迟.我们首先分析 DF-Cache 的组件 mega-UB 和 main-UB,然后计算得出不同配置的 DF-Cache 估计的实现代价.

mega-UB 和 main-UB 的物理实现如图 3 所示,我们对 RAM,CAM,ICache 用 CACTI5.3 版本进行评估,配置间有差异的输入参数如表 2 所示,另外所有配置使用 65nm 工艺,有 1 个读写端口,bank 数为 1.

Table 2 CACTI Inputs

表 2 CACTI 的输入

Config	Capacity/B	Block Width/B	Associativity	Output Width/b	Custom Tag	Tag Width/b
CAM128	3 200	25	0	200	1	32
CAM256	6 400	25	0	200	1	32
CAM512	12 800	25	0	200	1	32
CAM1024	25 600	25	0	200	1	32
RAM256	16 384	64		512	0	
RAM512	32 768	64		512	0	
RAM1024	65 536	64		512	0	
ICache	32 768	64	4	512	0	

访问延迟、功耗与面积的评估结果分别如图 7～9 所示:

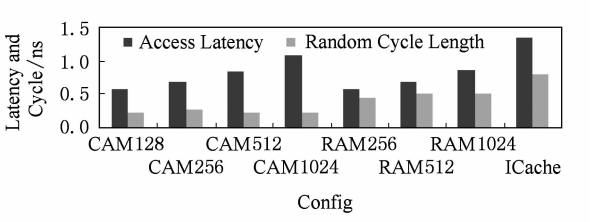


Fig. 7 Latency and cycle of CAM, RAM and ICache.

图 7 CAM, RAM, ICache 的延迟与周期

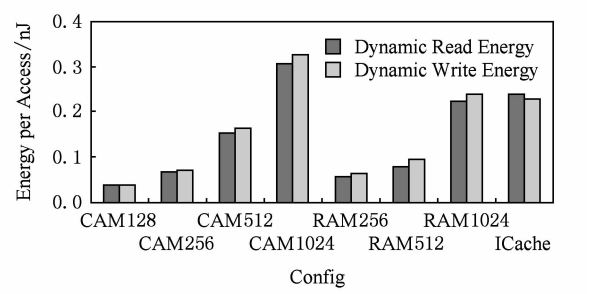


Fig. 8 Energy per access of CAM, RAM and ICache.

图 8 CAM, RAM, ICache 的单次访问功耗

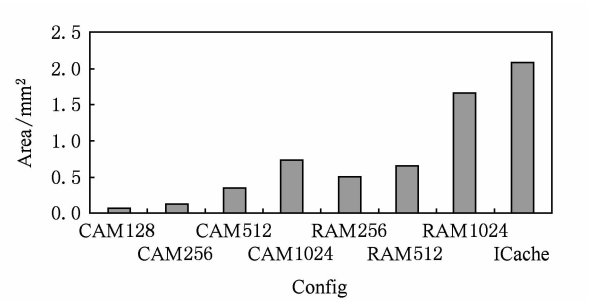


Fig. 9 Area of CAM, RAM and ICache.

图 9 CAM, RAM, ICache 的面积

mega-UB 使用 CAM 实现,其访问延迟、单次读写功耗和面积开销的变化曲线都比较陡峭,说明其可扩展性较差.但 CAM256 的各项数据仍在可接受的范围内.

main-UB 使用 RAM 实现,它在功耗和面积方面都要大大优于同样容量的 ICache,这是因为它使用从 mega-UB 中获得的 RAM 地址直接被访问,没有普通 Cache 的标签阵列(tag array).

从 UB 中取微指令分 2 种情况:如果是基本块的首行,首先访问 mega-UB,然后从 main-UB 中取得微指令;否则,利用基本块每行末尾的指针取得后续行的微指令.前者的代价较大,我们称之为最差情况(worst case),而后者为最好情况(best case).

UB 的各项指标计算如下:访问延迟为 2 个组件访问延迟的和(worst case)或 main-UB 的访问延迟(best case).周期等于 mega-UB 和 main-UB 周期较大者(访问 mega-UB 与访问 main-UB 流水实现).读写事件动态功耗为 2 个组件功耗的和(worst case)或 main-UB 的功耗(best case).面积开销为 2 个组件面积开销的和.UB 各种配置的评估结果如图 10～12 所示.

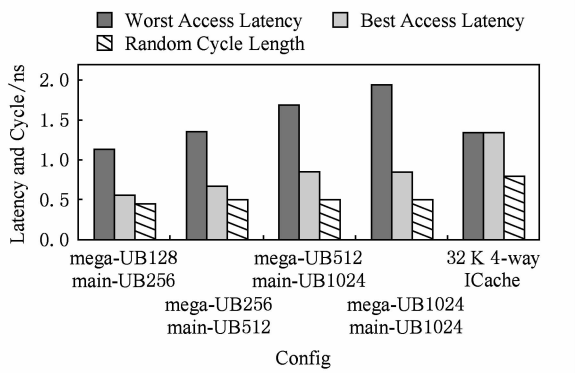


Fig. 10 Latency and cycle of UB.

图 10 UB 的延迟与周期

可以看出,mega-UB256 main-UB512 的配置在访问延迟和单次读写功耗以及面积开销方面都要优

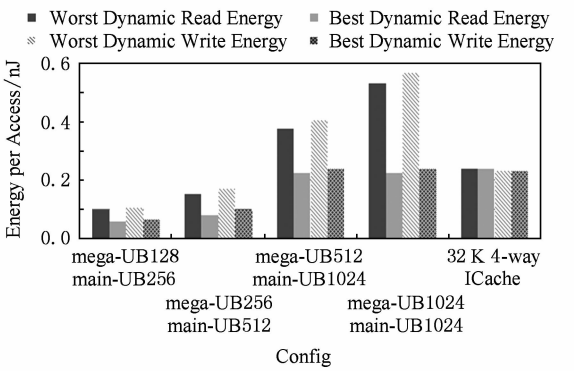


Fig. 11 Energy per access of UB.

图 11 UB 的单次访问功耗

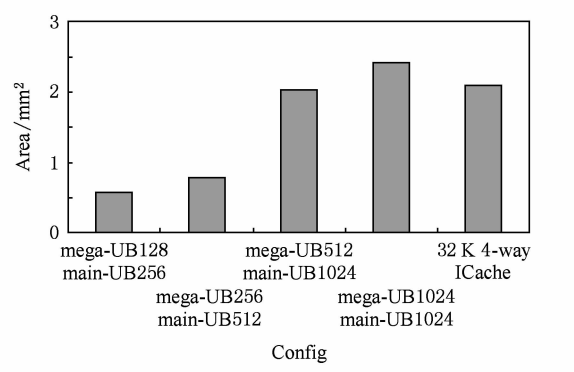


Fig. 12 Area of UB.

图 12 UB 的面积

于同样存储 32KB 指令数据的 4-Way ICache. 其功耗开销在最差情况也不大于 ICache, 而最优情况下只有 ICache 的一半. 面积开销只有 ICache 的 40%.

前面 2.1 节简单提到, mega-UB 与 main-UB 是松散耦合的, 与传统的 ICache 不同. 从该点出发, DF-Cache 可以在结构上进行优化进而提高性能. 首先, mega-UB 和 main-UB 可以通过地址缓存队列连接, mega-UB 负责不断将预测获得的指令基本块首数据行地址放到队列中, 而 main-UB 可以从队列中取得基本块首数据行地址, 自由地完成后续的微指令取出操作. 其次, main-UB 基于 RAM 实现, 其访问延迟和功耗低, 可以倍频工作, 或者多个读写端口同时访问多个基本块, 提高指令供给的宽度. 最后, 对在存储空间内连续的数个基本块, 可以在 main-UB 中链接起来, 使得 main-UB 有连续跨基本块预取指的能力, 而不完全依赖 mega-UB.

通过以上分析可以看出, DF-Cache 的结构和工作方式与传统的 ICache 相比, 在性能、面积和功耗上都有优势. 另外考虑到 DF-Cache 旁路了译码和重命名阶段, DF-Cache 可以降低处理器的动态功耗. 在 ESESC 中集成的 mcpat 工具里添加 DF-Cache 和向量重命名部件的延迟、面积和功耗模型, 并在 ESESC 的时序模型中添加必要的性能计数器, 即可以实时分析得到添加了 DF-Cache 的处理器运行特定测试程序时的动态功耗. 本文重点在于 DF-Cache 加入流水线对功能和性能的影响, 其具体的功耗优势不再继续展开. 本节通过分析说明了, DF-Cache 能以可接受的代价实现较高的动态指令覆盖率.

3.2 加入数据流缓存对流水线的影响

DF-Cache 加入流水线的作用可概括如下: 首先它提升了指令供给带宽并降低了指令供给延迟; 其次, DF-Cache 解耦合了流水线的前后端, DF-Cache 负责向后端供应指令, 可以解决 DMC 处理器中 V-Core 流水线前端能力较弱的问题. 本节, 我们从上述 2 个方面分析 DF-Cache 对流水线的影响.

表 3 展示了 ESESC 模拟处理器核的基准配置的全部参数设置, 我们记该配置为 F4-B128. F4 表示流水线宽度为每周期 4 条指令, B128 表示指令窗口容量为 128.

Table 3 Base Configuration F4-B128

表 3 基准配置 F4-B128

Parameter	Setup
FE Depth	3-cycle latency decode, 2-cycle latency rename
BTB	4096 entries, 4-way set associative
BPred	Hybrid local/global predictor, 11b history register, 2b saturate counter, 16K-entry PHT for global/local/choice each
Pipeline Width	4 instructions per cycle
Instruction Window Size	128 entries
FU	2INT/2FP/2MEM
FU Latency	INT: mul 3 cycles, div 8 cycles, others 1 cycle, FP: mul 3 cycles, div 8 cycles, others 3 cycles
LSQ	64 entries
L1 ICache	32 KB, 4-way set associative, 64B line, 2-cycle hit latency
L1 DCache	32 KB, 4-way set associative, 64B line, 3-cycle hit latency
L2 Cache	1 MB, 8-way set associative, 64B line, 10-cycle hit latency
L3 Cache	8 MB, 16-way set associative, 64B line, 12-cycle hit latency
Memory	8 B/cycle bandwidth, 256banks, 8192rows, 1024columns, column size: 256 B, 156-cycle worst latency

表 4 列举了我们选择进行实验的 10 种处理器核配置. 仅与基准配置不同的参数设置在表 4 中进行了列举, 包括流水线宽度、指令窗口大小、功能部件配备和是否搭配有 DF-Cache.

Table 4 Selected Configurations in Sample Space

表 4 样本空间内选定的配置

Configuration	Pipeline Width /instructions per cycle	Instruction Window Size /entries	FU	DF-Cache
F4-B128	4	128	2INT/2FP/2MEM	N
F4-B256	4	256	4INT/4FP/2MEM	N
F4-B512	4	512	8INT/8FP/2MEM	N
F6-B256	6	256	4INT/4FP/2MEM	N
F8-B512	8	512	8INT/8FP/2MEM	N
D4-B128	4	128	2INT/2FP/2MEM	Y
D4-B256	4	256	4INT/4FP/2MEM	Y
D4-B512	4	512	8INT/8FP/2MEM	Y
D6-B256	6	256	4INT/4FP/2MEM	Y
D8-B512	8	512	8INT/8FP/2MEM	Y

F4-B128,F6-B256,F8-B512 模拟的是有不同 ILP 发掘能力的 N-Core,其前端指令供给能力与后端的执行能力匹配较好. F4-B256,F4-B512 模拟的是通过 DMC 技术重构得来的 V-Core,因为流水线前端各阶段的融合效果差,它们的指令供给能力要弱于指令执行能力. D4-B128,D4-B256,D4-B512,D6-B256,D8-B512 这 5 个 D 系列配置,与 F 系列的配置一一对应,仅额外添加了 DF-Cache. 要考察前端结构对流水线的影响,我们需要控制非相关变量. 因而忽略流水线后端 V-Core 融合导致的额外开销,操作数传递延迟设置为 0. 举例说明,F4-B256,D4-B256,F6-B256,D6-B256 虽属不同结构,却拥有相同性能的后端. 这使得模拟获得的实验数据能够体现出不同前端结构带来的影响,有可比较性.

与 Core Fusion 等 DMC 设计的实验类似,我们关注 V-Core 执行单线程程序发掘 ILP 的能力,暂时不考虑多道程序和多线程程序在 DMC 中的执行. 我们使用 ESESC 的 TASS 采样模式运行了每个测试程序 10 亿条动态指令,TASS 采样设置 RWDT 各阶段的比例为 250:245:2:13. 模拟的结果如图 13 所示. IPC 数值已经归一化到 F4-B128,方便比较.

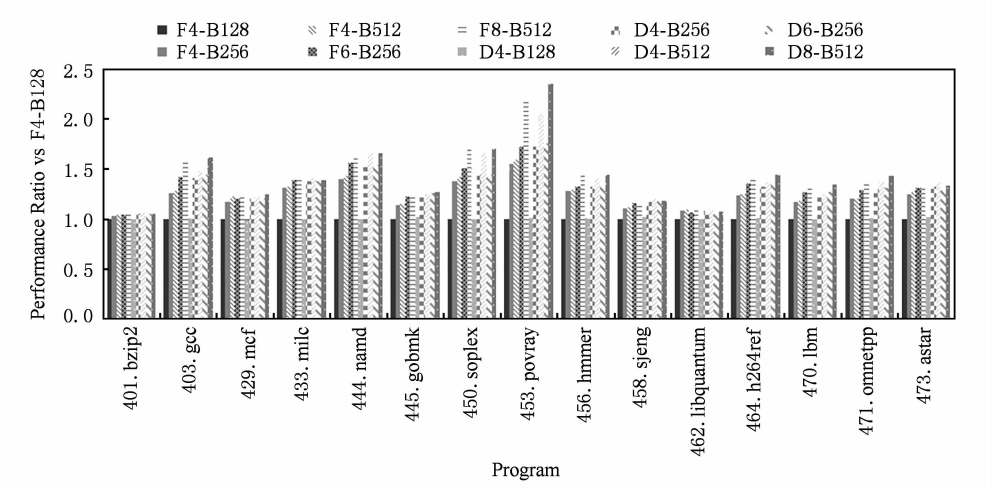


Fig. 13 One billion uops simulation result.

图 13 10 亿条动态指令模拟结果

简要分析没有 DF-Cache 的 F 系列配置的测试结果反映出的程序与处理器结构的特点.

首先,比较 F4-B128,F6-B256,F8-B512 这 3 个 N-Core 的数据,可以看到大部分的测试程序随着处理器核发掘 ILP 的能力上升而执行性能有所提升. bzip 和 libquantum 提升较小,说明程序本身蕴含的 ILP 较有限或者长延时指令较多,对指令窗口大小不敏感. gcc,namd,soplex,povray,h264ref 则含有较高的 ILP. 再对比图 6 我们可以看到,程序的数据流局部性与指令级并行性之间没有明显的相关关系,计算的这 2 个特性分别对应了微结构上流水线前端转化指令的能力和流水线后端执行指令的能力. 流水线前后端相匹配,且满足程序的 2 个特性才可以有比较高的执行效率.

其次,分别比较 F4-B256 和 F6-B256,以及 F4-B512 和 F8-B512 这 2 组 V-Core 对比 N-Core 的数据. 例如 gcc,milc,namd,povray 等测试程序,可以明显看到 DMC 重构得到的 V-Core 因为其前端相对较弱,影响到了流水线对这些程序的执行效果. 占用相同资源的 N-Core 性能强于重构得来的 V-Core.

3.2.1 数据流缓存对流水线性能影响

DF-Cache 将前端的译码和重命名信息进行缓存,当指令基本块在其中命中时,通过向量重命名后,可以快速填充到后端发射队列中. 所以 DF-Cache 可以提升前端带宽、降低前端延迟. 但是,传统的处理器取指、译码和重命名等功能已经流水化实现,并且分支预测部件的正确率都很高,DF-Cache 能发挥多少作用值得商榷,本节通过对比 D 系列 N-Core 相对 F 系列 N-Core 的测试数据来研究 DF-Cache 的加入对流水线性能的影响.

拥有不同 ILP 发掘能力的 N-Core 在有或无 DF-Cache 的情况下的执行性能对比如图 14 所示:

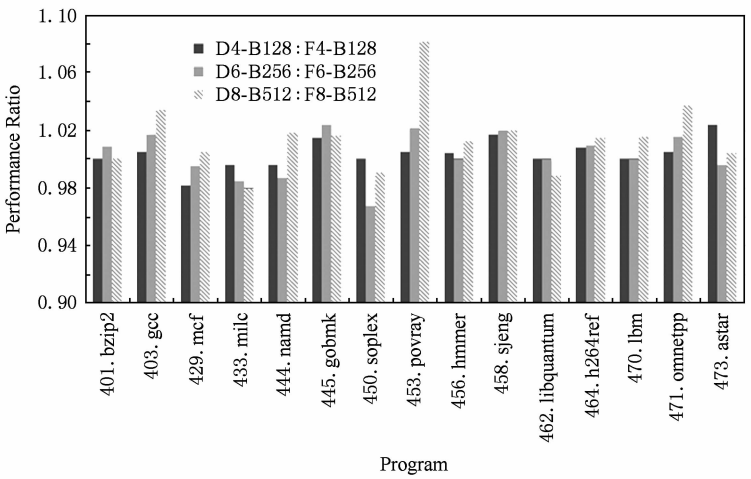


Fig. 14 The influence of DF-Cache on the pipeline of N-Core.

图 14 DF-Cache 对 N-Core 流水线性能影响

1) DF-Cache 提升了部分程序的性能,例如 gcc, gobmk, povray, sjeng, h264ref, omnetpp. 同时参照图 6 和图 13 可以得出,这些程序能够被 DF-Cache 加速存在 3 个因素:①程序本身要含有足够的 ILP. DF-Cache 提高指令供给,从而使得后端的计算能力得到更好的利用. 而 bzip2 与 libquantum 因为程序自身或处理器后端能力原因导致 ILP 较低,所以 DF-Cache 对其提升有限. ②这些程序的数据流局部性较差. 与直觉相反,DF-Cache 覆盖率较低的程序反而受到的 DF-Cache 加速效果更好,这是因为这些程序局部性较差,从而指令供给能力提升的空间也大. ③程序的分支预测准确率也影响 DF-Cache 的效果.

2) mcf, milc, soplex, libquantum, astar 的部分测试结果要比没有 DF-Cache 的情况更差. 这主要有 2 个原因:①如上段所述,数据流局部性好的程序前端带宽提升的空间非常有限,前后端匹配的流水线已经能非常好地利用后端的计算资源,DF-Cache

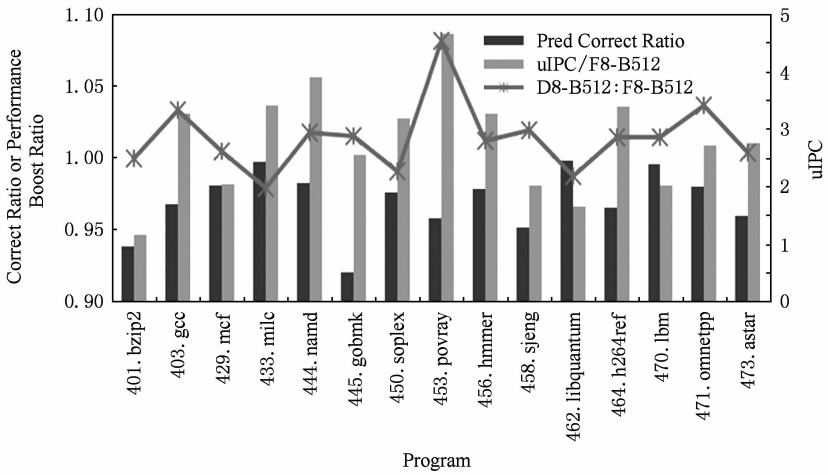


Fig. 15 Performance data of D8-B512 and F8-B512.

图 15 D8-B512 与 F8-B512 的性能数据

的加入能提供的正面效果较弱.②通过分析程序的执行过程,我们发现发射队列的快速填充导致了不在关键路径上的指令抢占了属于关键路径上的指令的执行时机,我们称这种降低程序执行效率的现象为关键路径遮挡.如果指令调度器实现了优先关键路径指令的调度算法,该问题即可解决.

图15重点展示了D8-B512和F8-B512的性能对比数据和F8-B512执行的ILP绝对数值以及分支预测准确率.可以看到DF-Cache对流水线的加速效果与程序的ILP呈较明显的正相关关系,而与程序的分支预测正确率呈较明显的反相关关系.举例说明,povray程序有高的ILP而分支预测准确率低,其受到DF-Cache加速效果最明显,达到8.1%;bzip2虽然有低的分支预测准确率,但其ILP有限,DF-Cache基本没有加速其执行;libquantum的分支预测准确率高但ILP低,DF-Cache的加入反而影响其执行,原因见上段的2点.所有程序中,soplex与astar属于偏离上述相关关系的测试程序,经过对它们执行过程的详细分析其性能表现与关键路径遮挡现象有关,指令调度策略属于流水线的后端执行能力研究的范围,与前端指令供给不相关,在此不再做展开.

总而言之,带有DF-Cache的处理器前端,能够更有效地发掘计算的数据流局部性、增强流水线的指令转化能力.1)它提升指令供给带宽,增强处理器发掘ILP的能力;2)它缩短指令供给延迟,降低了分支预测错误带来的惩罚.

3.2.2 数据流缓存解耦合流水线前后端

由3.2节开始对F系列的V-Core和N-Core数据的对比可知,V-Core前端能力较弱影响其性能.这是目前大部分DMC技术需要解决的问题.

首先,我们对比D系列的V-Core与F系列的V-Core的性能数据,分析研究DF-Cache加入流水线后解耦合前后端的作用与效果.如图16所示,除了bzip2,libquantum等ILP较差的程序外,V-Core运行的所有程序在DF-Cache的帮助下,IPC都有很大提升.D4-B256相对于F4-B256性能平均提升了5.5%,D4-B512相对于F4-B512性能平均提升了9.4%,尤其是povray性能高达28%,这说明DF-Cache为后端提供了更多的动态指令,能够较大提升流水线性能.

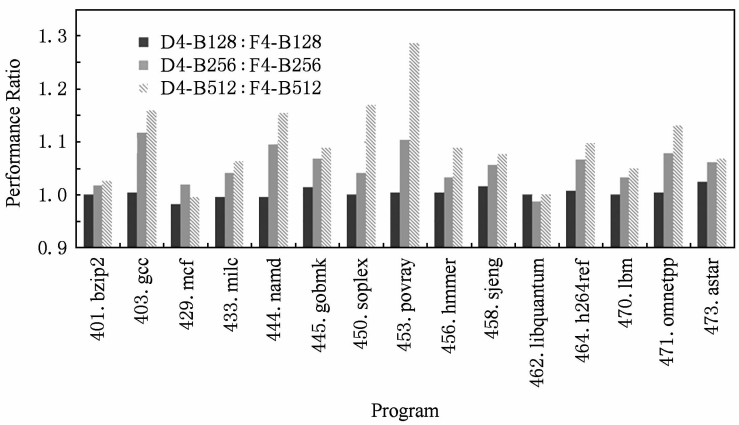


Fig. 16 The influence of DF-Cache on the pipeline of V-Core.

图16 DF-Cache对V-Core流水线性能影响

然后,我们对比具有基本相同硬件开销的集成了DF-Cache的V-Core与没有集成DF-Cache的N-Core的数据.参照图13,可以看出D4-B256和F6-B256,以及D4-B512和F8-B512性能差距较小,平均性能差距小于5%.

而Core Fusion等DMC设计其合并后的V-Core性能要大大弱于同宽度的N-Core①.这说明V-Core流水线后端的指令执行能力在DF-Cache的帮助下得到了充分利用.

通过上面2段对DF-Cache加入流水线后性能下限和上限的分析,我们可以看到V-Core不再受限于能力失配的前端,性能更接近其后端的指令执行能力.DMC技术在DF-Cache的帮助下有更高的实用价值.

① 例如Core Fusion的8发射V-Core在运行SPEC整型程序时性能只达到6发射N-Core的76%,当然Core Fusion全面建模了后端,我们对V-Core后端的估计相对乐观.

4 结束语

数据流缓存尝试发掘程序的数据流局部性.通过缓存并重用指令基本块内的数据流信息,数据流缓存可以旁路取指、译码和重命名等流水线前端阶段,解耦合流水线的指令供给和指令执行功能.

通过分析程序的数据流局部性,我们证明了数据流缓存能够以有限的代价覆盖90%以上的动态指令.在流水线宽度为4条指令、指令窗口大小为512条指令的虚拟核的配置下,采用数据流缓存提升了SPEC CPU2006平均9.4%的执行性能.

数据流缓存能够缩短指令供给延迟,提高指令供给带宽,降低分支预测错误带来的惩罚,通过解放后端来增强处理器核发掘ILP的能力.另外,数据流缓存替代流水线前端供给指令,解决动态多核技术中虚拟核的前端难以协作工作的问题,提升了动态多核技术中虚拟核的性能,增强动态多核技术的实用性.

参 考 文 献

[1] Agarwal V, Hrishikesh M S, Keckler S W, et al. Clock rate versus IPC: The end of the road for conventional microarchitectures [C] //Proc of the 27th Int Symp on Computer Architecture. New York: ACM, 2000: 248-259

[2] Borkar S, Dubey P, Kahn K, et al. Platform 2015: Intel processor and platform evolution for the next decade [J/OL]. [2015-03-04]. http://www.researchgate.net/publication/247190040

[3] Hill M D, Marty M R. Amdahl's law in the multicore era [J]. Computer, 2008, 41(7): 33-38

[4] Hennessy J, Patterson D. Computer Architecture: A Quantitative Approach [M]. San Francisco, CA: Morgan Kaufmann, 2011

[5] Lee L H, Moyer B, Arends J. Instruction fetch energy reduction using loop caches for embedded applications with small tight loops [C] //Proc of the 1999 Int Symp on Low Power Electronics and Design. New York: ACM, 1999: 267-269

[6] Singhal R. Inside Intel next generation Nehalem microarchitecture [C/OL] //Proc of the 20th Symp of Hot Chips. [2015-03-04]. http://www.cs.uml.edu/~bill/cs515/Intel_Nehalem_Processor.pdf

[7] Rotenberg E, Bennett S, Smith J E. Trace cache: A low latency approach to high bandwidth instruction fetching [C] //Proc of the 29th Int Symp on Microarchitecture. Los Alamitos, CA: IEEE Computer Society, 1996: 24-35

[8] Lempel O. 2nd generation Intel core processor family: Intel core i7, i5 and i3 [C/OL] //Proc of the 23rd Symp of Hot Chips. [2015-03-04]. http://www.hotchips.org/wp-content/uploads/hc_archives/hc23/HC23.19.9-Desktop-CPUs/HC23.19.911-Sandy-Bridge-Lempel-Intel-Rev 07.pdf

[9] Black B, Rychlik B, Shen J P. The block-based trace cache [C] //Proc of the 26th Int Symp on Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 1999: 196-207

[10] Swanson S, Schwerin A, Mercaldi M, et al. The WaveScalar architecture [J]. ACM Trans on Computer Systems, 2007, 25(2): article 4

[11] Oberoi P S, Sohi G S. Parallelism in the front-end [C] //Proc of the 30th Int Symp on Computer Architecture. New York: ACM, 2003: 230-240

[12] Ipek E, Kirman M, Kirman N, et al. Core fusion: Accommodating software diversity in chip multiprocessors [C] //Proc of the 34th Int Symp on Computer Architecture. New York: ACM, 2007: 186-197

[13] Kim C, Sethumadhavan S, Govindan M S, et al. Composable lightweight processors [C] //Proc of the 40th Int Symp on Microarchitecture. Los Alamitos, CA: IEEE Computer Society, 2007: 381-394

[14] Watanabe Y, Davis J D, Wood D A. WiDGET: Wisconsin decoupled grid execution tiles [C] //Proc of the 37th Int Symp on Computer Architecture. New York: ACM, 2010: 2-13

[15] Rodrigues R, Annamalai A, Koren I, et al. Performance per Watt benefits of dynamic core morphing in asymmetric multicores [C] //Proc of Int Conf on Parallel Architectures and Compilation Techniques. Los Alamitos, CA: IEEE Computer Society, 2011: 121-130

[16] Yeager K C. The MIPS R10000 superscalar microprocessor [J]. IEEE Micro, 1996, 16(2): 28-40

[17] Raasch S E, Binkert N L, Reinhardt S K. A scalable instruction queue design using dependence chains [C] //Proc of the 29th Int Symp on Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2002: 318-329

[18] Gibson D, Wood D. ForwardFlow: A scalable core for power-constrained CMPs [C] //Proc of the 37th Int Symp on Computer Architecture. New York: ACM, 2010: 14-25

[19] Sohi G S, Vajapeyam S. Instruction issue logic for high-performance, interruptible pipelined computers [C] //Proc of the 14th Int Symp on Computer Architecture. New York: ACM, 1987: 27-34

[20] Kim I, Lipasti M H. Half-price architecture [C] //Proc of the 30th Int Symp on Computer Architecture. New York: ACM, 2003: 28-38

[21] Henning J L. SPEC CPU2006 benchmark descriptions [J]. ACM SIGARCH Computer Architecture News, 2006, 34(4): 1-17

[22] Ardestani E K, Renau J. ESESC: A fast multicore simulator using time-based sampling [C] //Proc of the 19th IEEE Int Symp on High Performance Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2013: 448-459

Liu Bingtao, born in 1983. PhD candidate. Student member of China Computer Federation. His main research interests include processor microarchitecture, reconfigurable computing and heterogeneous computing.

Zhang Hao, born in 1981. PhD, associate professor. Member of China Computer Federation. Associate chief architect of the Godson-T many core processor. His research interests include high throughput CPU microarchitectures and application analysis.

Wang Da, born in 1981. PhD, associate professor. Member of China Computer Federation. Her main research interests include IC testing and analysis, micro architecture design, many-core processor, and VLSI design and testing.

Fan Dongrui, born in 1979. PhD, professor and PhD supervisor. Member of China Computer Federation. His main research interests include many-core processor design, high throughput processor design and low power micro-architecture.

Ye Xiaochun, born in 1981. PhD, associate professor. Member of China Computer Federation. His main research interests include high-performance computer architecture, software simulation, algorithm paralleling and optimizing.

Zhang Zhimin, born in 1963. Professor and PhD supervisor. His main research interests include SoC design and computer architecture.