一种高能效的面向单发射按序处理器的预执行机制

王箫音,佟 冬,党向磊,冯 毅,程 旭
(北京大学微处理器研究开发中心,北京 100871)

摘 要: 按序处理器凭借其在低成本、低功耗和高可扩展能力等方面的优势,越来越多地应用于多核处理器中.为进一步满足单线程程序的性能需求,有效提升按序处理器的访存性能至关重要.本文面向典型的单发射按序处理器提出一种高能效的预执行机制,充分利用预执行过程中的有效访存结果与计算结果加速程序的执行.为达到高能效的目标,一方面,本文提出基于收益预测的预执行动态调整策略,该策略采用三种收益预测方法来识别并避免无收益的预执行阶段.另一方面,本文采用基于信心估计的转移预测机制对预执行期间无法及时判定的转移指令进行优化.实验结果表明,在平均情况下,本文方法将基础处理器的性能提升 24.14%,而能耗仅增加 4.31%.与已有的两种预执行方法相比,本文方法在获取可比的性能优化效果的同时,能耗开销分别降低 7.72%和 10.72%,从而使能效性分别提高 10.3%和 11.39%.

关键词: 单发射按序处理器; 预执行; 访存延时包容
中图分类号: TP302.7 **文献标识码:** A **文章编号:** 0372-2112 (2011) 02-0458-06

An Energy-Efficient Executing Ahead Mechanism for Improving the Performance of Single-Issue In-Order Microprocessors

WANG Xiao-yin, TONG Dong, DANG Xiang-lei, FENG Yi, CHENG Xu
(*Microprocessor Research and Development Center of Peking University. Beijing* 100871, *China*)

Abstract: In-order microprocessors are increasingly adopted in a variety of multi-core chips due to their advantages in low power, low cost and high scalability. To further satisfy the performance requirement of single-thread applications, improving the load latency tolerance of in-order microprocessors is crucial. We propose an energy-efficient executing ahead mechanism which pre-executes the following instructions instead of stalling the processor when a long-latency cache miss occurs. This mechanism dynamically adjusts the executing ahead policy based on the prediction results of the performance benefit predictor to identify and eliminate the useless executing ahead periods. A confidence-based branch predictor is proposed for unresolvable branches during the useful executing ahead periods. Experimental results demonstrate that the performance is increased by 24.14% only with 4.31% energy overhead on average. Compared with two existing methods, the mechanism proposed in this paper decreases the energy consumption by 7.72% and 10.72% while achieving comparable performance enhancement, thus improves the energy-efficiency by 10.3% and 11.39%, respectively.

Key words: single-issue in-order microprocessors; executing ahead; load latency tolerance

1 引言

按序处理器凭借其在低成本、低功耗和高可扩展能力等方面的优势[1],越来越多地应用于面向吞吐率[2]甚至是高性能[3]的多核处理器中.随着处理器与存储器之间的速度差距日益扩大,片外存储器的访问延时成为制约单线程性能的重要瓶颈.由于单发射按序处理器不具备动态调度的指令窗口,当发生片上缓存失效这类长延时事件时,流水线将被迫停顿,从而造成性能损失.因此,有效提升单发射按序处理器的访存性能至关重要.

本文针对典型的单发射按序处理器提出一种高能效的预执行机制,充分利用预执行过程中的有效访存结果与计算结果加速程序的执行.为达到高能效的目标,一方面,本文方法采用基于收益预测的预执行动态调整策略,该策略根据处理器所处的运行阶段选择使用三种收益预测方法来识别并避免无收益的预执行阶段,从而减少对性能提升没有贡献的动态指令.另一方面,本文采用基于信心估计的转移预测机制对预执行期间无法及时判定的转移指令进行优化,进一步减少预执行过程中由于执行错误路径指令所造成的性能损失和能耗浪费.

收稿日期:2009-09-21;修回日期:2010-03-18
基金项目:国家 863 高技术研究发展计划(No.2006AA010202)

实验环境基于 UniCore-2 处理器的典型结构,使用 SPEC CPU2000 基准程序集进行评测.实验结果表明,在平均情况下,本文方法将基础处理器的性能提升 24.14%,而能耗仅增加 4.31%.与两种已有的预执行方法[4,5]相比,本文方法在获取可比的性能优化效果的同时,能耗开销分别降低 7.72% 和 10.72%,从而使能效性分别提高 10.3% 和 11.39%.

2　相关工作

访存延时包容(Latency Tolerance)技术[6]通过将长延时的存储访问与其他访存操作或计算操作相重叠进行来隐藏访存延时,在优化访存性能方面具有较大潜力.非阻塞执行技术[7~9]与预执行技术[4,5,10]是两种典型的访存延时包容技术.当发生片上缓存失效时,这两种技术均能够利用处理器的空闲周期预先执行失效访存指令的后续指令,通过将多个存储访问的延时相重叠来提升访存性能.这两种技术的主要区别体现在预先执行后续指令时指令的执行与提交方式上.

非阻塞执行技术[7~9]能够继续执行并正常提交与失效访存指令数据无关的指令,因而需要支持指令的乱序提交,并借助存储指令的重执行机制[8]和寄存器相关的跟踪机制[9]确保程序执行的正确性.实现上述机制将消耗大量的硬件资源,并增加设计复杂度,从而导致非阻塞执行技术的性价比不高.

预执行技术[4,5,10]对上述处理机制进行了简化,从而避免造成过高的硬件开销和设计复杂度.在预执行过程中,数据无关的指令能够正常执行并产生有效的访存或计算结果,数据相关的指令则直接被移出流水线,所有指令均不提交结果.由于不需要改变原有处理器的执行与提交机制,预执行技术是一种简单有效的访存延时包容技术.

已有的预执行技术可主要分为 Runahead 方法[4,10]和 Multipass 方法[5]两类.其中,Dundas 等人在文献[10]中首次提出预执行(Runahead)的基本方法,并用于提升五级流水线按序处理器中 L1 Cache 的性能.但是,该方法对较长延时的包容能力有限.Mutlu 等人对其进行了方法改进[4]和效率优化[11],但均是针对乱序执行机制与部件所提出.在 Runahead 方法[4,10]中,所有预执行指令都需要在预执行结束后重新执行,这将导致不必要的执行代价和能耗开销.Multipass 方法[5]用于弥补静态调度超标量处理器在程序运行中发生访存失效这类长延时事件时的不足.该方法保存预执行期间的计算结果,并由系统适当插入回溯(Restart)指令使一次预执行可以重复多遍.Multipass 方法需要由软硬件协同完成,且由于未进行合理的预执行控制,预执行及其回溯过程会造成较大的能耗开销.

综上所述,已有的预执行技术均未能全面地考虑到单发射按序处理器对于低能耗与高能效的迫切需求,从而未能对预执行过程中有利于加速程序执行和降低能耗开销的因素加以充分利用.本文的目标是提出适合于单发射按序处理器的高能效的预执行机制,这既需要针对单发射按序处理器的预执行过程对加速程序执行的潜力进行有效的挖掘,又需要最大限度地减少对性能提升没有贡献的预执行动态指令.

3　高能效的预执行机制

本文针对典型的单发射按序处理器提出一种高能效的预执行机制.为减少对性能提升没有贡献的预执行动态指令,本文定义预执行收益(Benefit)为预执行期间所发起的存储访问的数目[12],并采用基于收益历史、基于无效值和基于预执行信息这三种收益预测方法(Benefit Predictor)来识别并避免无收益的预执行阶段.本文还采用基于信心估计的转移预测器(Confidence-based Branch Predictor,CBP)对无法及时判定的转移指令进行优化.采用预执行机制的典型单发射按序处理器的结构如图 1 所示.其中,灰色部分表示为支持预执行所增加的部件,在正常执行阶段,这些部件将处于能耗节省状态.

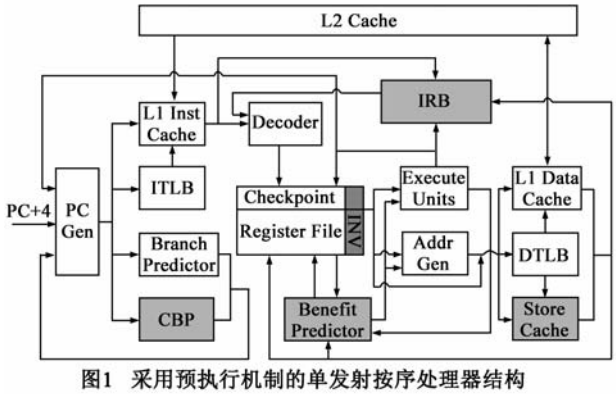


图1　采用预执行机制的单发射按序处理器结构

3.1　预执行基本过程

本文预执行机制的基本执行过程如图 2 所示.在初始时刻,处理器处于正常执行阶段,当检测到数据访问发生片上缓存失效时,将根据基于收益历史的收益预测(方法一)的结果决定是否进入预执行阶段.如果预测结果表明存在预期收益,处理器将对寄存器状态进行备份并建立检查点(Checkpoint),然后转换到预执行阶段;否则等待访问结果返回.

在预执行阶段,对于与失效 Load 指令存在数据相关的指令,其目标寄存器要标记为无效(INV).对于数据无关的指令,其计算结果将保存在指令与结果缓冲器(Instruction and Result Buffer,IRB)中,以避免预执行结束后对有效指令的重复执行.此外,Store 指令的存储数

据还将单独保存在小容量的 Store Cache[4] 中以便预执行的 Load 指令访问和使用.在预执行过程中,处理器将判断基于无效值的收益预测(方法二)的结果.如果预测结果表明存在预期收益,处理器将继续预执行过程;否则,将停止预执行并等待访问结果返回.对于预执行期间无法及时判定的转移指令,将使用其指令地址访问 CBP 以获取转移预测结果.当引发预执行的失效指令完成主存访问后,处理器将清空流水线,恢复备份的寄存器状态,并转换到合并结果阶段.

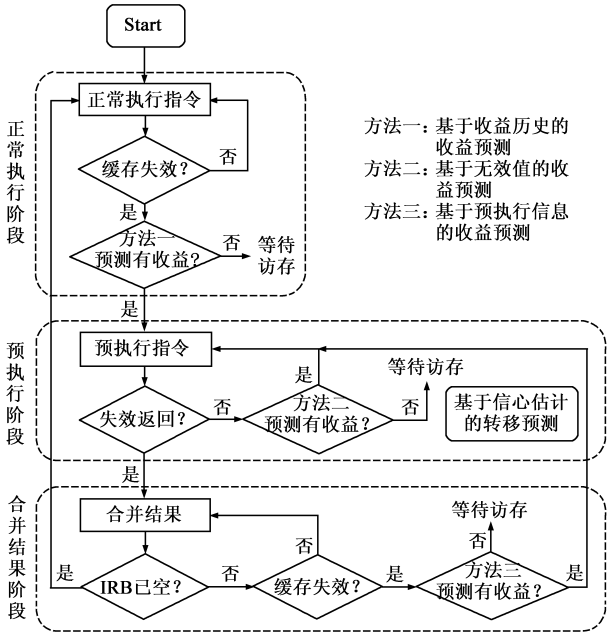


图2 本文预执行机制的基本执行流程

在合并结果阶段,处理器将从引发预执行的访存指令开始重新执行并提交结果.在此期间,当发现有缓存失效的主存访问尚未完成时,处理器将判断基于预执行信息的收益预测(方法三)的结果.如果预测结果表明存在预期收益,处理器将再次转换到预执行阶段;否则等待访问结果返回.对于在预执行阶段已经正确完成的指令,将直接提交结果,其他指令则需要重新执行.当所有预执行的指令都完成提交后,处理器将返回正常执行阶段.

3.2 基于收益预测的动态调整策略

为减少对性能提升没有贡献的预执行动态指令,本文采用基于收益预测的预执行动态调整策略来识别并避免无收益的预执行阶段.如第 3.1 节所述,该策略根据处理器所处的运行阶段选择使用基于收益历史、基于无效值和基于预执行信息这三种收益预测方法,能够有效减少动态执行的指令数目.本小节将对上述三种收益预测方法进行介绍.

3.2.1 基于收益历史的收益预测

基于收益历史的收益预测方法根据以往预执行阶段的收益对本次预执行的收益进行预测.本文采用一个 N 位的移位寄存器——收益历史寄存器来记录最近 N 次预执行阶段的收益情况.对于收益历史寄存器中的任意一位,数值为 0 表示相应的预执行阶段的收益为零,数值为 1 表示相应的预执行阶段的收益大于零.

如果收益历史寄存器的所有位均为零,表明近一时期的预执行阶段均未获得收益,那么,预测本次预执行的收益为零,并将收益历史寄存器的最低位更新为 1;否则,预测本次预执行存在收益,并在预执行结束后根据实际收益情况更新收益历史寄存器.为保证初始预执行过程的顺利进行,收益历史寄存器的初始值被设置为全 1.

3.2.2 基于无效值的收益预测

基于无效值的收益预测方法根据具有无效值的寄存器在所有寄存器中所占比例对继续预执行的潜在收益进行预测.当具有无效值的寄存器在所有寄存器中所占比例大于阈值 $T_1(T_1<1)$时,表明后续指令是数据无关指令的几率较低,因而继续预执行难以产生有效的存储访问,预测继续预执行的收益为零;否则,预测继续预执行存在收益.

3.2.3 基于预执行信息的收益预测

在合并结果阶段,如果处理器检测到有数据失效已经发起主存访问但尚未返回结果,表明该存储访问是在之前的预执行阶段中所发起.此时,需要结合以往预执行过程的信息对本次预执行的收益作出分析.有两种情况将导致本次预执行的潜在收益有限.首先,如果该数据失效的访问结果很快就能返回(情况一),则一旦进入预执行阶段,执行周期和所执行的指令数目都十分有限.其次,即使情况一不发生,如果大部分后续指令与最近一次预执行阶段已经预执行过的指令是重复的(情况二),也将导致无法获取收益.

基于预执行信息的收益预测方法根据最近预执行阶段的信息对上述两种情况进行识别,以预测本次预执行的收益.针对情况一,该方法将最近 M 次主存访问的平均延时与此次数据访问已经访存的周期数的差值和阈值 T_2 相比较,如果该差值小于阈值 T_2,表明访存即将结束,预测本次预执行的收益为零;否则,预测本次预执行存在收益.针对情况二,该方法将后续指令中与最近一次预执行重复的动态指令数目和最近 M 次预执行阶段的动态指令数目平均值相比较,如果前者大于后者乘以阈值 $T_3(T_3<1)$,表明重复的指令较多,预测本次预执行的收益为零;否则,预测本次预执行存在收益.

3.3 基于信心估计的转移预测

如果预执行的转移指令与失效 Load 指令存在数据相关,使得转移指令的源操作数始终无法就绪,处理器

只能一直沿着预测方向的路径取指并执行.这类转移指令被称为无法及时判定的转移指令[4].一旦无法及时判定的转移指令发生误预测,预执行的错误路径上的几十条甚至上百条指令将被丢弃,不仅会导致性能损失,还会造成能耗浪费[12].因此,针对无法及时判定的转移指令,本文采用基于信心估计[13]的转移预测器(CBP)对其进行优化.

CBP 采用指令地址索引的直接映射或组相联结构,每个表项记录无法及时判定的转移指令的执行信息.如图 3 所示,Tag 域记录转移指令地址的高位,Valid 域表示该表项是否有效,BTB 域记录转移指令的目标地址,Taken 域是一个两位的饱和预测器,预测该转移指令是否发生跳转,Confidence 域是一个两位的饱和预测器,表示对 CBP 预测正确性的信心.

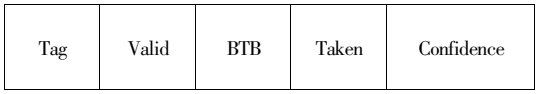


图 3　CBP 表项信息

在预执行阶段,对于无法及时判定的转移指令,将使用其指令地址访问 CBP.当发生表项命中时,如果 Confidence 域的数值大于或等于 2,则根据 Taken 域产生跳转方向预测结果;否则仍然使用传统转移预测器的预测结果.通过对无法及时判定的转移指令提供单独的预测器,并对其预测过程引入信心估计的机制[13],可以有效地减少发生误预测的次数,从而减少指令流一直处于错误路径的预执行阶段的数目.

4　实验评估

实验环境基于 UniCore-2 处理器的典型结构,分别采用 SimpleScalar[14]与 DRAMsim[15]模拟器对处理器及主存系统进行建模,并集成 Wattch 1.02[16]功耗模型支持功耗评估.本文从 SPEC CPU2000 基准程序集中选择 13 个典型程序进行模拟,其中包括 8 个整数类型和 5 个浮点类型测试程序,使用 SimPoint[17]工具选取由 100M 条指令构成的代表程序片段来实际运行.基础处理器配置如表 1 所示.

本文选取 Runahead 方法[4]和 Multipass 方法[5]在基础处理器模型中加以实现.在实验中,对于 Runahead 方法,选取 Runahead Cache 为 512B 2 路组相联结构;对于 Multipass 方法,选取 Advance Store Cache 为 512B 2 路组相联结构,Result Store 为 256 表项,使用硬件回溯机制(当具有无效值的寄存器所占比例超过 3/4 时进行回溯)来代替回溯指令.本文预执行方法的相关参数如表 2 所示.在正常执行阶段,为支持预执行所增加的部件都将处于能耗节省状态.其中,Store Cache 的容量仅为 L1 DCache 的 1/32,其面积开销可忽略不计.

表 1　基础处理器配置参数

参数	配置值
流水线	8 级,单发射按序执行
频率	1GHz
转移方向预测器	1024 表项
转移历史寄存器	8 位
目标地址缓冲器	128 表项,直接映射
L1 ICache	16KB,4 路组相联,每行 32 字节 2 周期访问延时
L1 DCache	16KB,4 路组相联,每行 32 字节 2 周期访问延时
L2 Unified Cache	512KB,8 路组相联,每行 64 字节 12 周期访问延时
指令 TLB	8 表项全相联一级 TLB 64 表项 4 路组相联二级 TLB
数据 TLB	8 表项全相联一级 TLB 64 表项 4 路组相联二级 TLB
主存	150 周期平均访问延时

表 2　本文预执行方法相关参数

参数	配置值
IRB	256 表项
Store Cache	512B,2 路组相联
CBP	32 表项,直接映射
其他参数	$N=2; M=5; T_1=3/4; T_2=10; T_3=0.8$

4.1　性能

本文方法与 Runahead[4]方法和 Multipass[5]方法对基础处理器的性能加速结果如图 4 所示.在平均情况下,Runahead 方法、Multipass 方法和本文方法分别将基础处理器的性能提升 20.67%、23.21%和 24.14%.

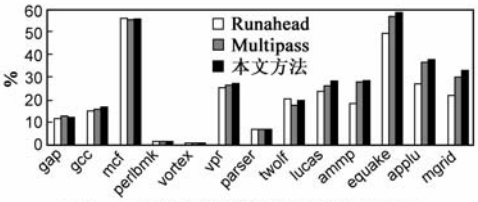


图4　几种预执行方法的性能加速结果

以基础处理器的执行时间为基准值将本文方法的执行时间作规格化,并将两者按照占用程序执行时间的各种事件进行分解,得出的数据如图 5 所示.其中,左边的柱状数据代表基准处理器的执行时间,右边的柱状数据代表本文方法的执行时间.以 mcf 为例,本文方

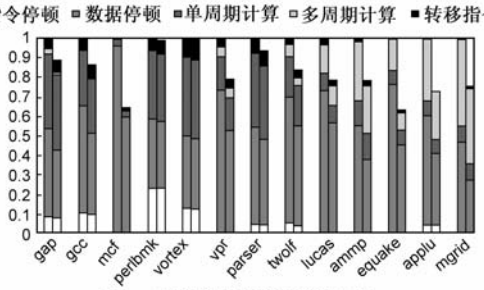


图5　规格化的执行时间分解

法通过充分发掘存储级并行,能够将多个访问的延时相重叠,从而减少了程序因数据访问而停顿的时间.以equake为例,本文方法通过在预执行期间对有效的计算结果进行保存,能够隐藏多周期计算指令的延时,从而加速浮点指令的执行.

4.2 能耗

本文方法与Runahead[4]方法和Multipass[5]方法的能耗开销如图6所示.在平均情况下,Runahead方法、Multipass方法和本文方法分别使基础处理器的能耗增加13.04%、16.84%和4.31%.

本文方法的能耗优势主要体现在如下几个方面.首先,预执行阶段的计算结果将被保存,从而避免已经正确完成的指令在预执行结束后重复执行;其次,基于收益预测的预执行动态调整策略能够识别并避免无收益的预执行阶段,有效地减少了对性能提升没有贡献的预执行动态指令;最后,基于信心估计的转移预测器能够减少处理器执行错误路径指令的能耗浪费.因此,与Runahead和Multipass两种方法相比,本文方法分别使处理器的能耗开销降低了7.72%和10.72%.

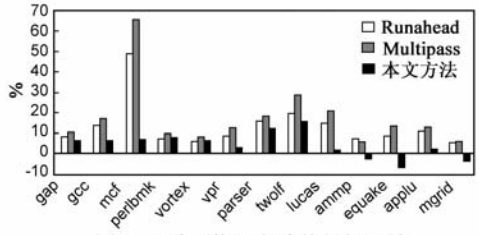


图6 几种预执行方法的能耗开销

4.3 能效性

本文采用能耗延迟积(Energy Delay Product, EDP)[18]作为处理器能效性(Energy Efficiency)的度量指标,用于衡量处理器设计在性能与能耗两个方面的综合结果.能耗延迟积可使用程序执行所消耗的总能量与执行时间的乘积来计算,其数值越小,表明处理器的能效性越高;而能效性越高,表明处理器设计在性能与能耗两个方面达到了更优的整体结果.以基础处理器的能耗延迟积为基准值,将Runahead[4]方法、Multipass[5]方法和本文方法的能耗延迟积作规格化,得出的数据如图7所示.在平均情况下,Runahead方法和Multipass方法分别使基础处理器的能耗延迟积减小6.32%和5.17%,而本文方法则将基础处理器的能耗延迟积减小了15.97%.实验数据表明,与Runahead和Multipass两种方法相比,本文方法在获取可比的性能优化效果的同时,还具有更小的能耗开销,从而使能效性分别提高10.3%和11.39%.

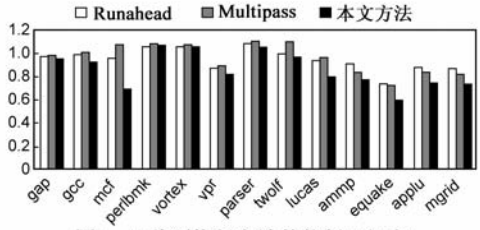


图7 几种预执行方法的能耗延迟积

5 结论

本文针对典型的单发射按序处理器提出一种高能效的预执行机制,充分利用预执行过程中的有效访存结果与计算结果加速程序的执行.为提高处理器的能效性,本文采用基于收益预测的预执行动态调整策略,该策略根据处理器所处的运行阶段选择使用三种收益预测方法来识别并避免无收益的预执行阶段,从而减少对性能提升没有贡献的预执行动态指令.为进一步优化预执行过程中指令流的正确性,本文采用基于信心估计的转移预测机制对无法及时判定的转移指令进行优化,从而减少预执行过程中由于执行错误路径指令造成的性能损失和能耗浪费.

通过面向单发射按序处理器的预执行过程对加速程序执行的潜力进行有效的挖掘,并减少对性能提升没有贡献的预执行动态指令,本文方法能够有效提升单发射按序处理器的访存性能,并且较好地保持其低功耗和低成本的优势.与已有的预执行方法相比,本文方法在获取可比的性能优化效果的同时,具有更小的能耗开销,从而在提高处理器的能效性方面也具有优势.

参考文献:

[1] K Asanovic, R Bodik, et al. The Landscape of Parallel Computing Research: A View from Berkeley[R]. California, USA: Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, 2006.

[2] P Kongetira, et al. Niagara: A 32-way multithreaded Sparc processor[J]. IEEE Micro, 2005, 25(2): 21 – 29.

[3] H Q Le, W J Starke, et al. IBM POWER6 microarchitecture [J]. IBM Journal of Research and Development, 2007, 51(6): 639 – 662.

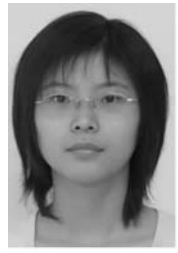
[4] O Mutlu, J Stark, C Wilkerson, Y N Patt. Runahead execution: An alternative to very large instruction windows for out-of-order processors[A]. Int'l Symposium on High-Performance Computer Architecture[C]. Anaheim, California, USA: IEEE Computer Society, 2003. 129 – 140.

[5] R D Barnes, S Ryoo, W W Hwu. "Flea-flicker" multipass pipelining: An alternative to the high-power out-of-order offense[A]. Int'l Symposium on Microarchitecture[C]. Barcelona, Spain: IEEE Computer Society, 2005. 319 – 330.

[6] D E Culler, J P Singh, A Gupta. Parallel Computer Architec-

ture: A Hardware/Software Approach, second edition [M]. USA: Morgan Kaufmann Publishers, Inc, 1996.
[7] S T Srinivasan, et al. Continual flow pipelines[A]. Int'l Conference on Architectural Support for Programming Languages and Operating Systems[C]. Boston, Massachusetts, USA: ACM Press, 2004. 107 – 119.
[8] S Nekkalapu, H Akkary, K Jothi, R Retnamma, X Song. A simple latency tolerant processor[A]. Int'l Conference on Computer Design[C]. Lake Tahoe, California, USA: IEEE Computer Society, 2008. 384 – 389.
[9] A Hilton, S Nagarakatte, A Roth. iCFP: Tolerating all-level cache misses in in-order processors[A]. Int'l Symposium on High-Performance Computer Architecture[C]. Raleigh, North Carolina, USA: IEEE Computer Society, 2009. 431 – 442.
[10] J Dundas, T Mudge. Improving data cache performance by pre-executing instructions under a cache miss[A]. Int'l Conference on Supercomputing[C]. Vienna, Austria: IEEE Computer Society, 1997. 68 – 75.
[11] O Mutlu, H Kim, Y N Patt. Techniques for efficient processing in runahead execution engines[A]. Int'l Symposium on Computer Architecture[C]. Madison, Wisconsin, USA: IEEE Computer Society, 2005. 370 – 381.
[12] Y Chou, B Fahs, S G Abraham. Microarchitecture optimizations for exploiting memory-level parallelism[A]. Int'l Symposium on Computer Architecture[C]. München, Germany: IEEE Computer Society, 2004. 76 – 87.
[13] E Jacobsen, E Rotenberg, J E Smith. Assigning confidence to conditional branch predictions[A]. Int'l Symposium on Microarchitecture[C]. Paris, France: IEEE Computer Society, 1996. 142 – 152.
[14] D Burger, T M Austin. The SimpleScalar tool set, version 2.0 [J]. ACM Computer Architecture News, 1997, 25(3): 13 – 25.
[15] D Wang, B Ganesh, N Tuaycharoen, K Baynes, A Jaleel, B Jacob. DRAMsim: A memory system simulator[J]. ACM Computer Architecture News, 2005, 33(4): 100 – 107.
[16] D Brooks, et al. Wattch: A framework for architectural-level power analysis and optimizations[A]. Int'l Symposium on Computer Architecture[C]. Vancouver, British Columbia, Canada: IEEE Computer Society, 2000. 83 – 94.
[17] E Perelman, et al. Picking statistically valid and early simulation points[A]. Int'l Conference on Parallel Architectures and Compilation Techniques[C]. New Orleans, Louisiana, USA: IEEE Computer Society, 2003. 244 – 255.
[18] R Gonzalez, M Horowitz. Energy dissipation in general purpose microprocessors[J]. IEEE Journal of Solid-State Circuits, 1996, 31(9): 1277 – 1284.

作者简介:



王箫音　女,1983 年生于河北保定,北京大学信息科学技术学院博士研究生.研究方向包括微处理器结构、低功耗高速缓存设计和存储系统性能优化.
E-mail: wangxiaoyin@mprc.pku.edu.cn

佟　冬　男,1971 年生于吉林长春,北京大学信息科学技术学院副教授.研究方向包括计算机体系结构、超大规模集成电路设计、系统芯片及软硬件协同设计.