doi:10.11887/j.cn.201501003 http://journal.nudt.edu.cn

用于减少远程 Cache 访问延迟的最后一次写访问预测方法*

夏 军,徐炜遐,庞征斌,张 峻,常俊胜
(国防科技大学 计算机学院,湖南 长沙 410073)

摘 要:为减少远程 Cache 访问延迟,提高共享存储系统的性能,提出了一种新的基于程序内在写突发特性的最后一次写访问预测方法,并对一个具体的目录协议进行了改造,以支持该预测方法。通过预测 Cache 块的最后一次写访问并提前对其进行降级,处理器能直接从主存中读取数据,从而减少了远程 Cache 访问所需的一个网络跳步数。与当前基于指令的预测方法相比,该方法能极大减少存储开销。基准测试程序的评测结果表明,该方法能获得 83.1% 的预测准确率,并且能提高 8.57% 的程序执行性能,同时与基于指令的预测方法相比,该方法能分别减少历史踪迹表 69% 的存储开销和签名表 36% 的存储开销。

关键词:Cache 一致性协议;远程 Cache 失效;写突发;最后一次写访问;自降级
中图分类号:TP302.1 **文献标志码**:A **文章编号**:1001-2486(2015)01-014-07

A last-write-touch prediction scheme used to reduce remote Cache miss latency

XIA Jun, XU Weixia, PANG Zhengbin, ZHANG Jun, CHANG Junsheng
(College of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract: To reduce remote cache transfer latency and improve the performance of shared memory systems, a new last-write-touch prediction scheme that exploits the inherent write characteristics of a program is proposed and a directory protocol to support the scheme is adapted. By predicting a last-write-touch and self downgrading a cache block in advance, a processor can get the data from the memory directly and one network hop can be saved for a remote cache access. Compared with the existing instruction-based prediction technique, much storage overhead can be reduced. Experimental results show that it can achieve an average prediction accuracy of 83.1%, leading to improvements up to average 8.57% on the final application performance. Moreover, compared with the instruction-based prediction scheme, the scheme can reduce the storage overheads of the history table by 69% and the storage overheads of the signature table by 36%.

Key words: Cache coherence protocol; remote Cache miss; write burst; last-write-touch; self downgrade

在支持 Cache 一致性的共享存储系统中,所有处理器都能通过 Load/Store 指令访问所有的存储空间,这为用户提供了与单机一样熟悉的编程环境并简化了处理器之间的通信编程模型。共享存储系统需要使用 Cache 一致性协议来保证每个处理器中 Cache 数据的一致性。一般来说,Cache 一致性协议可以分为基于目录的 Cache 一致性协议和基于监听的 Cache 一致性协议两类。由于目录协议具有比监听协议更好的可扩展性,因此共享存储系统中目录协议的使用更为广泛。需要从远程 Cache 获得数据的 Cache 失效被称为远程 Cache 失效,此类 Cache 失效在某些情形下占据了整个 L2 Cache 失效的 60% 以上,从而会对系统性能产生重要影响[1-4]。然而在基于目录的共享存储系统中,目录的间接访问特性要求远程 Cache 失效需要 3 个网络跳步的一致性操作才能完成,从而增加了额外的访问延迟。

当前的研究表明,将 3 跳步的远程 Cache 失效转换为 2 跳步的远程 Cache 失效,能有效降低远程 Cache 访问延迟,提高共享存储系统的性能。一般来说,独占者/共享者预测和自作废/自降级是实现上述目标的两种有效方法。独占者/共享者预测是消除目录间接访问的一种有效手段。通过预测一致性请求的目标处理器,请求处理器可以将一致性请求直接发往目标处理器,从而避免对目录的间接访问。有许多研究人员对通过程序员[5]、编译器[6]、硬件[7]和软硬件联合设计[8]的方法实现目标处理器预测进行了研究。与独占

* 收稿日期:2014-06-11
基金项目:国家自然科学基金资助项目(61202119);国家 863 计划资助项目(2013AA014301)
作者简介:夏军(1976—),男,重庆人,副研究员,博士,硕士生导师,E-mail:ddk_kevin@163.com

者/共享者预测方法不同,自作废[9-10]或自降级[11-12]是通过预测一个 Cache 块的最后一次访问或最后一次写访问操作,并提前对 Cache 块进行作废或降级操作,使得请求处理器可以直接从主存中读取数据块,从而缩短一个网络跳步所需的延迟。上述两种方法是互补的,可以结合在一起使用,共同减少远程 Cache 访问延迟。

在以前的自作废或自降级研究中,有的需要软件的参与来确定作废 Cache 块的时机,对软件不透明;有的采用简单的等待固定时间间隔对 Cache 块进行降级操作的方法,导致最后一次写访问的预测不精确;有的使用基于指令踪迹的方法来预测最后一次写访问操作的发生时机,但其需要的额外存储开销过大。由于上述方法都存在一定的不足,因此需要研究新的对软件透明、预测精度高、存储开销小的自作废或自降级方法。

现提出了一种新的基于程序内在写突发特性的最后一次写访问预测方法,即基于写突发次数的自降级预测机制(Number-based DownGrade Predictor,NDGP)。实验分析发现处理器最后一级 Cache 的写突发访问次数一般分布在一个较小的数值范围内并且呈现出可重复模式,通过追踪写突发次数的历史出现模式,就能预测 Cache 块何时不会再被本地处理器写访问,从而可以对其进行降级。与当前基于指令的最后一次写访问预测方法,即基于指令踪迹的自降级预测机制(Trace-based DownGrade Predictor,TDGP)[11]相比,NDGP 使用写突发次数而非程序计数器(Program Counter,PC)来计算历史踪迹和生成签名。由于写突发次数的位宽远小于 PC 的位宽,所以 NDGP 能大大减少历史踪迹表和签名表的存储开销。还对一个基于目录的五态(Modified,Owned,Exclusive,Shard,Invalid,MOESI)一致性协议进行了改造,以支持 NDGP 进行最后一次写访问预测操作。

1　NDGP 设计

1.1　写突发特性

最后一次写访问预测的关键是能预测出 Cache 块何时不会再被本地处理器写访问,从而可以通过降级将脏数据写回主存中。一个数据块被以拥有写访问权限的方式载入 Cache,本地处理器就可以对该 Cache 块反复进行写操作,直到别的处理器请求该数据块或该数据块被其他数据块替换掉。如果能找到 Cache 块的写突发次数与请求该数据块的一致性事物之间的关系,那么就能预测何时可以对 Cache 块进行降级操作。

为了对写突发模式的特性进行研究,采用 Simics[13]全系统模拟器和可以仿真 Cache 和存储系统的通用执行驱动的多处理器模拟器(General Execution-driven Multiprocessor Simulator,GEMS)[14]模拟器构建了一个 16 核的 CMP 系统。基于该实验平台,对 NPB3.3 和 Splash2 中的基准测试程序进行了写突发特性分析,表 1 列出了所使用的基准测试程序及其对应的输入规模。

表 1　基准测试程序及其输入规模

Tab.1　Benchmarks and input size

程序	输入规模	程序	输入规模
bt	Class S	lu-c	512 matrix
cg	Class S	lu-nc	512 matrix
ft	Class S	ocean-c	258 grid
is	Class S	ocean-nc	258 grid
lu	Class S	radiosity	room
sp	Class S	radix	4M keys
barnes	64K particles	water-ns	512 mol.
fft	256K points	water-sp	512 mol.
fmm	16K particles	cholesky	tk15.O

数据块的写突发次数定义为:在数据块以拥有写访问权限的方式载入 Cache 到其他处理器请求该数据块期间,该数据块在 Cache 中被写访问的次数。实验追踪了所有处理器核发出的最后一级 Cache 访问请求。

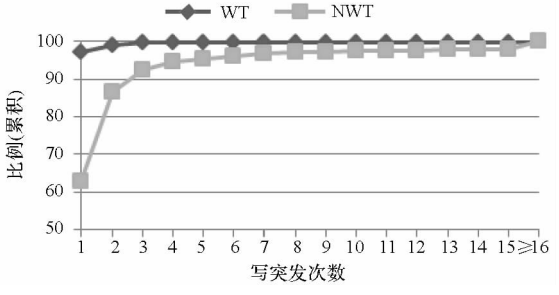


图 1　表 1 中基准测试程序写突发次数的平均累积分布

Fig.1　Average cummulative distributions of the write burst numbers of the benchmarks in Tab.1

图 1 给出了表 1 中基准测试程序的写突发次数的平均累积分布示意图。标记为 WT 的曲线为写突发次数增加了权重,而标记为 NWT 的曲线未考虑权重。采用的权重是同一地址出现相同写突发次数的数量。为写突发次数增加对应的权重

将能更好地显现写突发模式具有的重复出现的特性。

从 NWT 曲线可以看出,小于 16 的写突发次数所占的比例超过了 95%,这说明最后一级 Cache 的写突发访问次数一般分布在一个较小的数值范围内。因此,如果将写突发次数用作踪迹,那么只需 4 位存储单元就可以覆盖超过 95% 的踪迹,与将 PC 当作踪迹相比较,踪迹存储开销将大大减少。

从图 1 可以看出,WT 曲线达到饱和的速度快于 NWT 曲线。在 WT 曲线中,小于 5 的写突发次数所占比例接近 100%,这说明写突发次数呈现出大量重复的模式。如果重复出现的写突发次数具有较好的时间局部性,那么写突发次数就可以被用来准确地预测 Cache 块何时能被降级。

1.2 NDGP 实现

该 NDGP 实现方法与最后一次访问预测(Last Touch Prediction,LTP)[10] 和 TDGP[11] 类似,差别在于 NDGP 计算踪迹和生成签名的方法与之不同。NDGP 使用写突发次数而非 PC 来计算踪迹和产生签名,从而大幅减少历史踪迹表和签名表的存储开销。

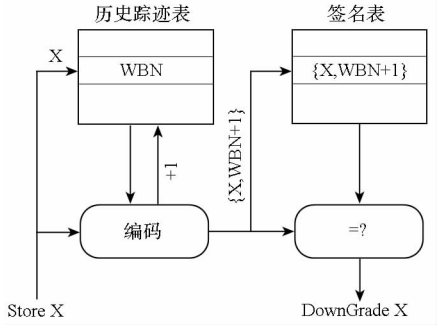


图 2 两级 NDGP 实现结构

Fig. 2 Two-level number-based downgrade predictor

两级 NDGP 的实现结构如图 2 所示。历史踪迹表用于存储当前踪迹,而签名表用于记录可以用来预测自降级的踪迹。处理器最后一级 Cache 的每个 Cache 块在历史踪迹表中都有对应项。当一个新的数据块因写失效被载入 Cache 时,历史踪迹表中的对应项被设置为 1,表明当前的写突发次数为 1。后续对该 Cache 块的每次写操作都会使得历史踪迹表中对应项所存储的写突发次数加 1。从中可看出,踪迹计算只使用了简单的递增计数逻辑,而非 LTP 和 TDGP 所采用的更为复杂的截断二进制加法。另外,NDGP 采用将地址和写突发次数进行并联操作来产生签名,而不是像 TDGP 那样采用异或操作来产生签名。

当一个显式降级请求到达时,对应 Cache 块的当前签名将被登记在签名表中。每当对一个 Cache 块进行写访问操作时,该 Cache 块当前新生成的签名将在签名表中进行查找。如果能找到匹配项,该 Cache 块将被降级。NDGP 也使用 2 位的置信计数器为预测的训练过程增加滞后效应。如果一个降级操作后面紧跟随一个来自同一处理器核的写操作,那么触发该降级操作的预测是失败,因为该处理器核还未完成写突发过程。目录可以发现这种情形并通知 NDGP 预测失败,此时 NDGP 将会降低签名表中对应项的置信度。

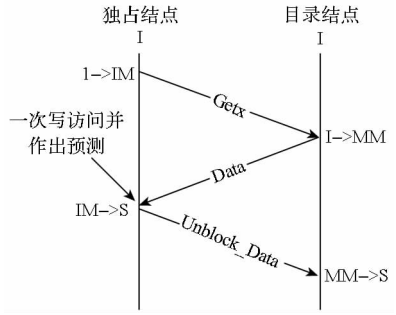
2 一致性协议改造

为了在共享存储系统中支持 NDGP,除了每个处理器核需要增加最后一次写访问预测机制之外,基础的 Cache 一致性协议也需要进行改造。本文所使用的基础协议为 GEMS 所提供的 MOESI 目录协议[14],并为其新增了预测协议流程和预测失败流程,同时也新增了 Cache 瞬时状态和目录瞬时状态,以解决因预测而引入的协议冲突问题。

当预测条件满足时,预测协议流程将被触发,完成对数据块的降级操作。如果此时历史踪迹表中对应的写突发次数为 1,则说明独占结点刚将该数据块载入 Cache 且仅对 Cache 中的该数据块进行了一次写访问操作。在原来的协议流程中,独占结点将向目录结点发送 Exclusive_Unblock 消息,并将自身的 Cache 状态修改为 M 态(Modified)。当 Exclusive_Unblock 消息到达目录结点后,目录状态也被修改为 M 态。但是在新的协议流程中,Cache 状态将被降级为 S 态(Shared),并且新增消息 Unblock_Data 将替换原来的 Exclusive_Unblock 消息。Unblock_Data 消息携带独占结点写入 Cache 中的数据,当它到达目录结点后,数据会被写入主存且目录状态被修改为 S 态。图 3(a)给出了写突发次数为 1 时的预测协议流程。

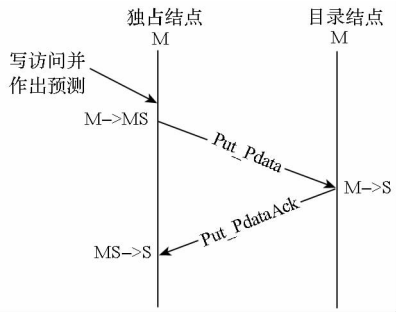
如果预测条件满足时写突发次数大于 1,这说明独占结点的当前写访问操作命中了 Cache。在原来的协议流程中,数据将直接写入 Cache。但是在新的协议流程中,数据除了写入 Cache 之外,独占结点还将向目录结点发送新增消息 Put_Pdata,并且 Put_Pdata 消息携带写入 Cache 的新数据。独占结点发出 Put_Pdata 消息后,其 Cache 状态修改为新增的 Cache 瞬时状态 MS 态(Modified - to - Shared),以防止任何一致性转发请求访问对应的 Cache 块。当 Put_Pdata 消息到

达目录结点且无冲突发生时,Put_Pdata 携带的数据将被写入主存且目录状态被修改为 S 态。最后,目录结点向独占结点返回新增消息 Put_PdataAck,独占结点在收到 Put_PdataAck 消息之后将 Cache 状态从 MS 态修改为 S 态。图 3(b)给出了写突发次数大于 1 时的预测协议流程。



(a)写突发次数等于 1

(a)Write burst number equal to 1



(b)写突发次数大于 1

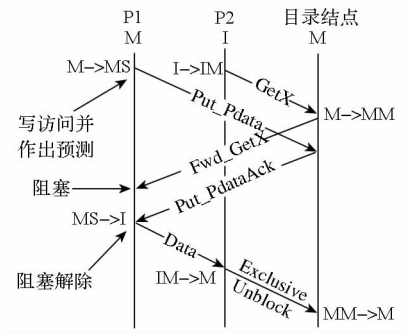
(b)Write burst number greater than 1

图 3　预测协议流程

Fig. 3　Base prediction flows

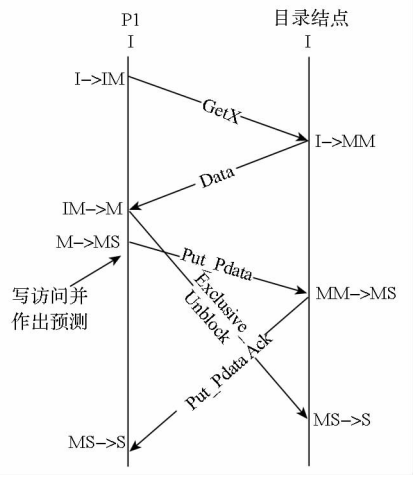
当目录结点检测到预测失败时,预测失败协议流程将被触发。目录结点中每个目录项新增 PredictFlag 和 PredictID 位域,用于预测失败检测。当预测发生时,Cache 块会被降级,并且在目录结点收到 Unblock_Data 或 Put_Pdata 消息时,对应目录项的 PredictFlag 会被置位且独占结点的结点号会被保存在 PredictID 中。如果目录结点收到了一个 GetX 消息,对应目录项的 PredictFlag 有效且 GetX 的源结点号与 PredictID 相同,则说明出现了预测失败情形。此时,目录结点将会向独占结点返回一个携带了预测失败标志位的 Data 消息。

新增的 Cache 瞬时状态 MS 用于解决预测操作与一致性转发请求之间的冲突问题。当独占结点作出预测并发出 Put_Pdata 消息后,Cache 状态被修改为 MS 态。为了简化预测操作与来自其他结点的一致性转发请求之间的协议冲突处理,在 Put_PdataAck 消息返回独占结点将 Cache 状态转换为稳定的 S 态之前,任何一致性转发请求都被禁止访问对应的 Cache 块。图 4(a)给出了预测操作与一致性转发请求 Fwd_GetX 之间的协议冲突处理示例。



(a) 预测操作与一致性转发请求之间的冲突处理

(a) Race between a prediction and a forwarded request



(b) 预测操作与目录阻塞解除消息之间的冲突处理

(b) Race between a prediction and an unblock operation

图 4　协议冲突处理流程

Fig. 4　Race resolving flows

新增的目录瞬时状态 MS 用于解决预测操作与目录阻塞解除消息(Unblock)之间的冲突问题,图 4(b)给出了预测操作与目录阻塞解除消息之间的协议冲突处理示例。P1 发出的 Exclusive_Unblock 消息和 Put_Pdata 消息到达目录结点的先后顺序是不确定的。如果 Exclusive_Unblock 消息先到达目录结点,那么就不存在协议冲突,后到达的 Put_Pdata 消息将如图 3(b)所示那样被处理。但是,如果 Put_Pdata 消息先到达目录结点并且将目录状态变为与发出 Put_Pdata 消息的结点相同的 MM 态的结点,那么就存在协议冲突,此时目录状态将被修改为 MS 态并且 Put_PdataAck 消息被返回给 P1。最后,后到达的 Exclusive_Unblock 消息将目录状态转换为稳定的 S 态。

3 实验评测

3.1 实验平台

该实验使用 Simics 模拟器[13]结合 GEMS 模拟器[14]的方法来进行全系统模拟。模拟环境为一个由 16 个瓦片构成的片上多处理器(Chip Multi-Processors,CMP)系统,其中每个瓦片包含一个 AMD 处理器核和两级私有 Cache。该 CMP 系统采用 GEMS 提供的 MOESI 目录协议来维护 Cache 一致性,并且运行未经修改的 CentOS4.7 操作系统。表 2 给出了该 CMP 系统的系统参数,而表 1 给出了评测所使用的基准测试程序。

表 2 模拟系统参数

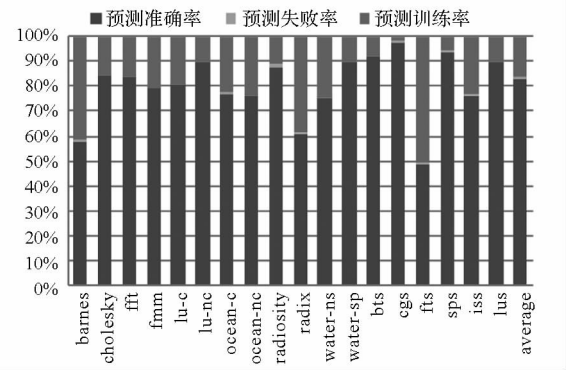
Tab. 2 Simulated system parameters

参数	值
处理器模型	AMD – hammer,顺序发射,单发射
L1 指令/数据 Cache	64KB,4 路组相联,64B line size,2 cycles
L2 Cache (私有)	2MB,8 路组相联,64B line size,6cycles,LRU
互连网络	全交叉互连拓扑结构,1 cycle 链路延迟
主存延迟	158 cycles

3.2 预测有效性

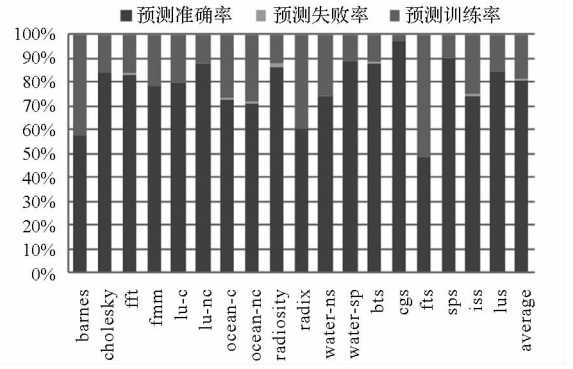
图 5(a)和图 5(b)分别给出了 NDGP 和 TDGP 的预测成功率、预测失败率和预测训练率。预测成功率是所有预测成功的写访问所占的比例;预测失败率是所有预测失败的写访问所占的比例;预测训练率是所有未做出预测的写访问所占的比例。

图 5 前面各列针对表 1 列出的每个基准测试程序分别给出了对应的预测成功率、预测失败率和预测训练率,而最后一列给出了所有程序的平均预测成功率、平均预测失败率和平均预测训练率。从图 5 中可看出,NDGP 平均能获得 83.1% 的预测准确率、0.61% 的预测失败率和 16.3% 的预测训练率,而 TDGP 平均能获得 80.6% 的预测准确率、0.46% 的预测失败率和 18.9% 的预测训练率。NDGP 获得的高预测准确率说明写突发次数具有较好的时间局部性。与 TDGP 不同,NDGP 使用写突发次数来计算历史踪迹和生成签名,这使得 NDGP 可以不依赖于 PC 进行预测。因此,相比 TDGP,NDGP 有更多的机会进行预测,从而能获得更高的预测准确率和更低的预测训练率。



(a) NDGP 的预测结果

(a) Prediction results of NDGP



(b) TDGP 的预测结果

(b) Prediction results of TDGP

图 5 NDGP 和 TDGP 的预测有效性比较

Fig. 5 Comparison of prediction effectiveness between NDGP and TDGP

虽然 NDGP 获得了更高的预测准确率和更低的预测训练率,但其预测失败率却比 TDGP 高,这是因为 NDGP 会遇到更多的子踪迹别名(subtrace aliasing)[10]场景,而减少子踪迹别名出现的概率是降低 NDGP 预测失败率的重要途径。目前 NDGP 仅依靠历史上单个写突发次数进行预测,如果采用写突发次数序列进行预测,则有可能降低子踪迹别名出现的概率,但其代价为历史踪迹表和签名表的存储开销会相应增加。后续工作将对采用写突发次数序列减少预测失败率的可行性和效果进行评估,确定合适的写突发次数序列长度参数,在预测失败率和存储开销之间找到合适的折中点。

3.3 程序执行性能

一个正确的预测可以使得远程共享失效能直接从主存中读取数据,从而减少了一个网络跳步数,而一个错误的预测会使得刚降级的处理器核需要重新获取写权限,从而增加了写访问延迟。图 6 给出了因预测正确而从主存中直接读取数据的远程共享失效的比例。在所有的远程共享失效

中,NDGP 和 TDGP 平均分别能使 87.4% 和 86.4% 的远程共享失效从主存中直接读取数据。由于 NDGP 具有较高的预测准确率,所以能从主存中直接读取数据的远程共享失效的比例也高一些。

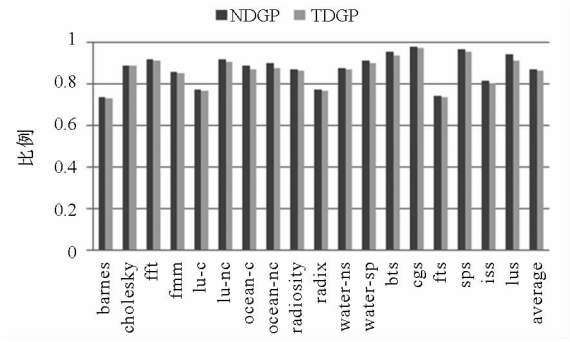


图 6　因预测正确而能从主存中直接读取数据的远程共享失效的比例

Fig. 6　Percentage of shared misses satisfied by memory data for correct predictions

图 7 给出了因远程共享失效延迟减少所带来的程序性能加速,其中,Base 为未采用预测方法的程序原始执行时间。NDGP 和 TDGP 的执行时间相对于 Base 进行了归一化处理。NDGP 和 TDGP 分别能平均提高程序 8.57% 和 8.65% 的执行性能。虽然 NDGP 能使更多的远程共享失效缩短访问延迟,但是它的预测失败率也更高,最终 NDGP 能获得与 TDGP 几乎一样的程序执行性能。

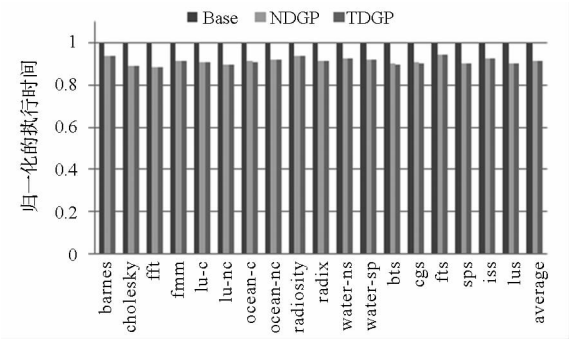


图 7　程序执行时间

Fig. 7　Program execution time

3.4　存储开销

采用文献[11]中给出的方法来实现历史踪迹表和签名表。历史踪迹表实现为 L2 Cache Tag 阵列的一个副本,而签名表实现为一个 16 路组相联、包含 64K 项的 Cache。由于模拟系统中物理地址为 40 位,因此使用 40 位地址来生成签名。

对于 NDGP,使用 4 位宽的写突发次数来计算历史踪迹,因为图 1 表明 4 位存储单元就可以覆盖超过 95% 的踪迹。在模拟系统中,PC 的宽度为 64 位,对于 TDGP,本文与文献[11]一样采用完整的 PC 来计算历史踪迹。

NDGP 和 TDGP 的历史踪迹表的项数都与 L2 Cache 的表项数相同。历史踪迹表的每个表项由 1 位的有效标志、Tag 位域和历史踪迹构成。在模拟系统中,L2 Cache Tag 位域的宽度为 22 位,而历史踪迹表 Tag 位域的宽度与 L2 Cache Tag 位域宽度相同。NDGP 的历史踪迹位域宽度为 4 位,而 TDGP 的历史踪迹位域宽度为 64 位。因此,NDGP 的历史踪迹表的表项宽度为 27 位,而 TDGP 的历史踪迹表的表项宽度为 87 位。与 TDGP 相比,NDGP 的历史踪迹表的存储开销降低了 69%。

NDGP 和 TDGP 的签名表被组织成 Cache 的形式且 Cache 的索引位为 12 位。每个 Cache 项由 1 位的有效标志、Tag 位域和 2 位的置信计数器构成。由于 NDGP 的签名位宽为 44 位,TDGP 的签名位宽为 64 位,所以 NDGP 签名 Cache 的 Tag 位宽为 32 位,TDGP 签名 Cache 的 Tag 位宽为 52 位。因此,NDGP 签名 Cache 的表项宽度为 35 位,而 TDGP 签名 Cache 的表项宽度为 55 位。与 TDGP 相比,NDGP 的签名表的存储开销降低了 36%。

4　结论

本文提出了一种新的用于减少远程 Cache 失效延迟的最后一次写访问预测方法 NDGP。NDGP 基于程序内在的写突发特性,采用写突发次数来计算历史踪迹和生成签名,实现对最后一级 Cache 的最后一次写访问的预测功能。与当前基于指令的最后一次写访问预测方法 TDGP 相比,NDGP 的实现更简单且消耗的存储资源更少。实验结果表明 NDGP 能获得与 TDGP 几乎一样的预测准确率和程序执行性能,但能减少历史踪迹表 69% 的存储开销和签名表 36% 的存储开销。

参考文献(References)

[1] Acacio M E, Gonzalez J, Garcia J M, et al. A novel approach to reduce L2 miss latency in shared multiprocessors [C]// Proceedings of the 16th International Parallel and Distributed Processing Symposium, Fort Lauderdale, USA, 2002:62 - 70.

[2] Gharachorloo K, Sharma M, Steely S, et al. Architecture and design of Alphaserver GS320 [C]//Proceedings of the International Conference on Architectural Support for Programming Language and Operating Systems (ASPLOS IX), Cambridge, USA, 2000: 13 - 24.

[3] Iyer R, Bhuyan L N, Nanda A. Using switch directories to

speed up cache-to-cache transfers in CC-NUMA multiprocessors[C]// Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS'00), Cancun, Mexico, 2000: 721 – 728.

[4] Zhang Z. Architectural sensitive application characterization: the approach of high-performance index-set (HP-Set)[R]. Technical Report HPL – 2001 – 75, HP Laboratories Palo Alto, USA, 2001.

[5] Abdel-Shafi H, Hall J, Adve S V, et al. An evaluation of the fine-grain producer-initiated communication in cache-coherent multiprocessors[C]//Proceedings of the 3rd IEEE Symposium On High-Performance Computer Architecture, San Antonio, USA, 1997:204 – 215.

[6] Trancoso P, Torrellas J. The impact of speeding up critical sections with data prefetching and forwarding[C]//Proceedings of the International Conference on Parallel Processing, Minneapolis, USA, 1996:79 – 86.

[7] Martin M M K, Harper P J, Sorin D J, et al. Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors[C]//Proceedings of the International Symposium on Computer Architecture, San Diego, USA, 2003:206 – 217.

[8] Demetriades S, Cho S. Predicting coherence communication by tracking synchronization points at run time[C]//Proceedings of the International Symposium on Microarchitecture, Vancouver, Canada, 2012:351 – 362.

[9] Lebeck A R, Wood D A. Dynamic self-invalidation: reducing coherence overhead in shared-memory multiprocessors[C]// Proceedings of the International Symposium on Computer Architecture, Santa Margherita Ligure, Italy, 1995:48 – 59.

[10] Lai A, Falsafi B. Selective, accurate, and timely self-invalidation using last-touch prediction[C]//Proceedings of the International Symposium on Computer Architecture, Vancouver, Canada, 2000: 139 – 148.

[11] Somogyi S, Wenisch T F, Hardavellas N, et al. Memory coherence activity prediction in commercial workloads[C]// Proceedings of the 2004 Workshop on Memory Performance Issues, Munich, Germany, 2004:37 – 45.

[12] Cheng L, Carter J B, Dai D. An adaptive cache coherence protocol optimized for producer-consumer sharing[C]// Proceedings of the International Symposium on High Performance Computer Architecture, Scottsdale, USA, 2007: 328 – 339.

[13] Magnusson P S, Christensson M, Eskilson J, et al. Simics: A full system simulation platform[J]. IEEE Transactions on Computer, 2002, 35(2):50 – 58.

[14] Martin M M K, Sorin D J, Beckmann B M, et al. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset[J]. ACM SIGARCH Computer Architecture News, 2005 33(4):92 – 99.

(上接第 13 页)

参考文献(References)

[1] Baek S H, Park K H. Matrix-stripe-cache-based contiguity transform for fragmented writes in RAID-5 [J]. IEEE Transaction Computer, 2007,56(8): 1040 – 1054.

[2] Du Y M, Xiao N, Liu F, et al. CSWL: cross-SSD wear-leveling method in SSD-based RAID systems for system endurance and performance[J]. Journal of Computer Science and Technology, 2013, 28(1): 28 – 41.

[3] Chung T S, Park D J, Park S, et al. System software for flash memory: a survey[C]//Proceedings of International Conference Embedded and Ubiquitous Computing, 2006: 394 – 404.

[4] Baek S H, Park H H. Prefecting with adaptive cache culling for striped disk array[C]//Proceedings of USENIX, Boston, 2008: 363 – 376.

[5] 刘秀菊. RAID5 小写更新的冗余管理机制[J]. 微电子学与计算机,2012, 29(7): 112 – 115.
LIU Xiuju. A Redundancy management policy for RAID5[J]. Microelectronics & Computer, 2012, 29(7): 112 – 115. (in Chinese)

[6] Li Z, Jin P Q, Su X, et al. CCF-LRU: a new buffer replacement algorithm for flash memory[J]. IEEE Transaction on Consumer Electronics, 2009, 55(3): 1351 – 1359.

[7] Woodhouse D. JFFs: the journaling flash file system[EB/OL]. 2001[2008 – 12 – 05]. http://sources. redhat. com/jffs2/jffs2. pdf.

[8] Wu C T, He X B. GSR: a global stripe-based redistribution approach to accelerate RAID5 scaling[C]//Proceedings of 41st ICPP, 2012: 460 – 469.

[9] Zhang G Y, Zheng W M, Shu J W. ALV: a new data redistribution approach to RAID5 scaling [J]. IEEE Transactions on Computer, 2010, 59(3): 345 – 357.

[10] Kim H, Ahn S. BPLRU: a buffer management scheme for improving random writes in flash storage[C]//Proceedings of 6th USENIX Conference on File and Storage Technologies, 2008.

[11] Bucy J S, Schindler J, Schlosser S W, et al. The disksim simulation environment version 4.0 reference manual[R]. Carnegie Mellon University Parallel Data Lab Technical Report CMU – PDL – 08 – 101, 2008.

[12] Prabhakaran V, Wobber T. SSD extension for DiskSim simulation environment[CP/OL]. 2009. http://research. microsoft. com/en-us/downloads/b41019e2-1d2b-44d8-b512-ba35ab814cd4/

[13] 方健,宋振龙,李琼,等. 基于闪存的存储阵列研究[C]// 第十七届计算机工程与工艺年会暨第三届微处理器技术论坛, 2013:170 – 176.
FANG Jian, SONG Zhenlong, LI Qiong, et al. The research of raid based on NAND flash[C]//Proceedings of NCCET, 2013:170 – 176. (in Chinese)