doi:10.3772/j.issn.1002-0470.2020.07.001

处理器访存子系统关键队列的性能建模①

李文青②[*,**] 吴 畏[*,**] 章隆兵[*,**,***] 肖俊华[*,**,***] 王 剑[*,**,***]

([*]计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)

([**]中国科学院大学 北京 100049)

([***]龙芯中科技术有限公司 北京 100190)

摘 要 处理器访存性能对其整体性能有着很大的影响,访存子系统的设计显得尤为重要。高性能超标量处理器的访存子系统中存在多个关键队列,如何快速进行设计折中,成为设计的关键。本文采用软件模拟器和回归模型结合的建模方法,提出了一个面向访存子系统关键队列的回归分析模型,并设计实现了相应的访存子系统模拟器。将软件模拟器与目标处理器设计的现场可编程门阵列(FPGA)原型验证平台进行准确性校准,并采用回归模型对软件模拟器的模拟数据进行分析,结果表明:实验验证结果稳定且对于所测试程序误差在10%以内。该建模方法可以量化分析访存子系统中关键队列大小与性能之间的关系,有效扩大硬件设计空间探索的范围,加快高性能处理器访存子系统的优化设计。

关键词 处理器设计空间探索;访存子系统;软件模拟器;回归模型

0 引 言

随着片上晶体管数量的增加,处理器计算能力迅猛提升,出现"存储墙"问题,访存成为处理器性能的一个重要瓶颈[1,2]。工业界和学术界使用许多技术来解决和缓解"存储墙"问题,如使用Cache、多发射乱序执行、向量指令等,从提高访存速度、挖掘指令间的并行性掩盖访存时间或减少访存次数等方面解决"存储墙"问题。

超标量处理器支持访存的推测执行,增加访存指令执行时间上的重叠,加快了程序的执行。处理器推测访存过程中,访存流水线中的几个关键队列有着重要的作用,这些队列保存指令状态、实现多条指令同时访存、确保存取的数据不因发出顺序改变而带来错误、并且维护访存指令的提交顺序。

中央处理器(central processing unit,CPU)计算能力的提高,加大了对访存速度的需求,访存子系统的设计越来越复杂,访存流水线中队列压力变大,CPU访存相关队列也越做越大。队列的增大对CPU直接的影响就是面积和功耗的增加,快速寻找队列大小的最佳组合和性能的折中点十分重要。

队列大小组合方面并没有很多研究可以参考,其中大多数是按照经验来做,同时也可以使用模拟、建模的方式找到一个性能较好的大小组合[3,4]。模拟处理器执行情况可以使用硬件仿真或者软件模拟。软件模拟是处理器设计空间探索常用的方式,但软件模拟硬件的执行通常要比真实处理器慢很多,时间上的开销使得巨大的设计空间无法得到足够的探索。硬件仿真虽然较软件模拟更快更准确,但灵活性差,资源需求大,也不适合做设计空间的探索。

本文采用了模拟器和回归模型相结合的建模方法,从模拟到预测,逐步扩大设计空间探索的范围,

① 国家自然科学基金(61432016,61521092)和中国科学院重点部署(ZDRW-XH-2017-1)资助项目。

② 女,1994年生,博士生;研究方向:计算机系统结构,处理器设计;联系人,E-mail:liwenqing@ict.ac.cn

(收稿日期:2019-07-19)

求解队列大小组合和性能的最佳平衡点。本文设计并实现了周期精确的访存子系统模拟器,对访存子系统的功能和行为进行了详细的模拟,并使用目标处理器的现场可编程门阵列(field programmable gate array,FPGA)原型验证平台进行了校准。通过实验和分析,提出了一种面向访存子系统关键队列大小的回归模型,该模型测试误差较低且结果稳定,模型可用于预测整个访存子系统设计空间中队列大小组合的性能。

本文剩余部分组织如下:第 1 节回顾了与本研究相关的前人的工作,第 2 节介绍模拟器和回归模型的设计,第 3 节描述了模拟器的实现,模型的构建以及误差分析,第 4 节使用模型预测不同条件下队列大小的最佳组合,第 5 节是整个文章的总结。

1 相关工作

在访存子系统的设计发展中,有很多工作对其中的队列进行了研究。文献[5-8]介绍了通过改变 load/store 队列的组织方式、相互关联性、数据依赖的解决方法等方面减少功耗、面积和延迟,提高处理器性能。本文侧重于在确定的队列组织方式和策略下,探索适合本架构的队列大小平衡,从而在特定架构下使硬件资源达到最高的利用率。

处理器设计空间探索通常基于模拟器展开,通过测试大量不同的处理器参数,得到对应响应,进而指导硬件设计。有许多模拟器在研究中得到广泛的应用,如 SimpleScalar、GEM5、SimICS、PTLsim、McSimA + 等,它们可在功能模拟、时序模拟、全系统模拟或应用级模拟等不同抽象层次对处理器进行仿真模拟。软件模拟硬件的速度较慢,这是影响对设计空间探索的一大因素。本实验中的模拟器对多种访存子系统中队列的进出时机、解决数据冲突策略进行了周期精确的详细模拟,忽略了其余部件不必要的细节,在提高模拟速度的同时,有针对性地模拟架构设计,得到了更准确的结果。

通过模拟器可以得到设定配置的对应响应(如性能、功耗、某事件发生次数等信息),理论上可以通过遍历整个设计空间找到全局最优解,然而因为资源、时间的限制和可配置参数的巨大组合数,使用软件模拟或者硬件仿真得到整个设计空间的结果是不可能的。对于搜索空间非常大的问题,一般采用启发式搜索[9](如 A*、模拟退火等)或预测模型[10-12](如线性回归、支持向量机等统计学或机器学习方法)方法。文献[13]通过少数模拟点得到 26 个处理器参数的模型,通过拟合出的模型系数得到对性能影响较大的处理器部件,指导处理器优化方向。自文献[13]之后,预测模型的方法被广泛运用到处理器设计空间探索中。本文使用预测模型的方法,通过分析和实验,得出访存流水线关键队列大小之间的回归模型。得到模型后无需再使用模拟器,便可以对整个设计空间中队列不同大小组合的性能进行预测评估,扩大了可行的设计空间探索范围,减少了设计空间探索的时间。

2 软件模拟器与回归模型混合建模

2.1 访存流水线中关键队列

在超标量处理器中,将已经准备好的访存指令发射、执行,很大程度上加快了处理器的执行速度,但是访存指令的推测执行同时也会带来数据冲突的问题。访存子系统中,乱序执行需要依靠 LDQ(load queue)、STQ(store queue)和 MMQ(memory issue queue)这几个关键的队列来实现并维护正确的数据关系[5]。

LDQ 和 STQ 是访存指令的重定序队列,LDQ/STQ 按照程序顺序存放 load/store 指令。这 2 个队列在程序运行过程中记录已经在访存流水线中且尚未提交的访存指令的状态,并保证已经写回的访存指令按序提交。同时,2 个队列中一些进入退出机制(如回滚、例外、正常提交等情况)保证了写回寄存器和内存的数据的正确性。

MMQ 是访存流水线的第 1 级,按照一定的规则将其中未发射的访存指令发射到访存流水线中。MMQ 中指令的退出由 LDQ 和 STQ 中指令的状态进行决定。

从以上描述可以看出,这 3 个队列的大小影响着能同时在访存流水线中的访存指令的数量,不同

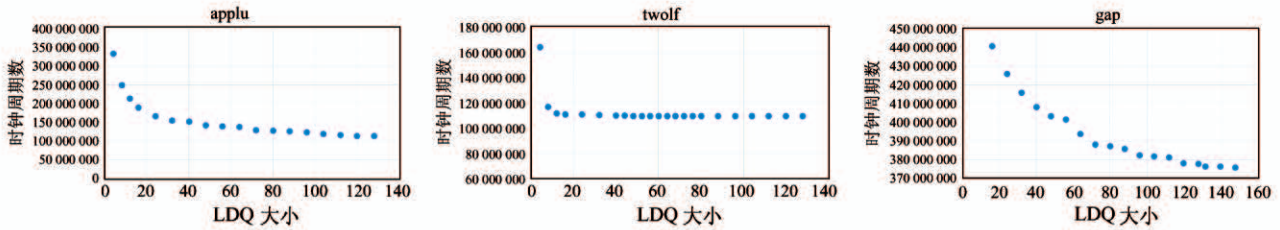
的查找和退出机制也会产生不同程度的回滚和例外的发生。

同时在流水线中的访存指令越多,指令之间的时间重叠就会增多,从而增加了整体的执行速度。但是回滚和例外也会因为流水线中指令数量的增加而增加,从而在一定程度上回滚导致更多发射出的指令无效。重新发射访存指令浪费了时间和资源,例外更是会打断指令的执行,造成更大的损失。

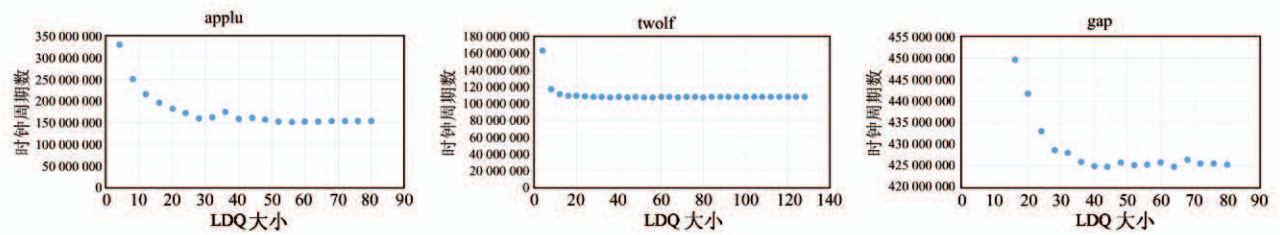
模拟器假定 ROQ(reorder queue,重排序队列)、MMQ、STQ 都无限大,其余配置都和目标机器相同,图 1(a)展示了不同 LDQ 的大小对运行时钟周期数的影响。假定模拟器 ROQ、MMQ、STQ 以及其余配置都和目标机器相同,图 1(b)展示了改变 LDQ 的大小对运行时钟周期数的影响。

从图 1 可以看出,随着 LDQ 的增大,总的时钟周期数呈下降趋势,在 LDQ 较大时,下降趋势逐渐变缓直至几乎不变。曲线所趋近的水平线是 LDQ 不出现满阻塞情况下的时钟周期数,也就是 CPU 不受访存流水线队列满阻塞情况下的性能。这时候,程序执行的周期数受到 MissQ(miss queue)大小、Cache、程序本身数据相关的影响,以及定浮点队列大小、流水线深度、流水线条数等 CPU 其他参数的影响。

在 STQ、MMQ 上做同样的实验,测得的结果和图 1 类似,其原因也大致相同。



(a) ROQ、MMQ、STQ 无限大



(b) ROQ、MMQ、STQ 和目标机器相同

图 1　ROQ、MMQ、STQ 不变,LDQ 和时钟周期的关系

总的来说,队列的增大在达到拐点之前对 CPU 性能有着正向的影响,且有着决定性的作用。

回滚和例外等因为乱序造成的因素虽然会带来负影响,但影响程度不如增大队列获得的收益大。对于一些程序,可以从图中看到,曲线下降趋势中会有一些波动。通过模拟器统计的数据也能看出,在那些点受例外回滚的影响较大。

对比观察图 1 中(a)和(b)可发现,将其余 3 个队列由无限大变为和目标机器相同,曲线的拐点都向左移动。这说明,访存流水线中队列不阻塞的点受到其他队列大小的影响。

表 1、表 2 和表 3 分别是使用 SPEC CPU 2000[14] 中 mcf 程序 load-store 流,模拟器中的 ROQ 设为无限大,每次 MMQ、LDQ、STQ 中 1 个无限大、剩余 2 个队列不断改变大小得到这 2 个队列大小的各种组合。按照其中一个队列大小固定分类画图,观察在一个队列逐渐变化时,性能变为稳定的另一个队列大小的变化。如表 1 所示,ROQ 和 MMQ 无限大,当 STQ 为 4 时测得的 LDQ 大小变化对应的时钟周期逐渐稳定时 LDQ 大小是 36。

任何一个队列满阻塞都会造成指令发射的停止,从而减少访存流水线中的指令数。所以,队列最

表 1　ROQ、MMQ 无限大，STQ 大小不同时，对应的性能稳定时 LDQ 最小大小

STQ	4	12	20	28	36	40	44	46	48	50
LDQ	36	72	86	92	92	92	92	92	92	92

表 2　ROQ、STQ 无限大，MMQ 大小不同时，对应的性能稳定时 LDQ 最小大小

MMQ	4	12	20	28	32	36	40	44	52	60
LDQ	60	76	76	80	84	84	84	84	84	92

表 3　ROQ、LDQ 无限大，MMQ 大小不同时，对应的性能稳定时 STQ 最小大小

MMQ	4	12	20	28	32	36	40	44	52	60
STQ	24	44	54	64	64	68	72	80	80	80

佳大小会受到其他队列大小的影响。通过单纯改变一个队列大小找到的该队列最佳大小这一搜索方法是不妥当的。为了找到性能最佳折中点，需要将所有队列的大小变化一起考虑。

2.2　软件模拟器设计

不同的处理器会有不同的处理数据相关的策略。在访存流水线关键队列中，队列之间相互访问的方式、流水线中的指令回退的条件、检测到数据错误后的处理方式、指令退出各个队列的时机等策略都会有所不同[15,16]。这使得不同处理器即使队列大小相同，队列大小组合对性能的影响也是不一样的。也就是说，策略不同的处理器的最佳队列大小组合是不一样的。乱序访存的不同实现策略使得对每款处理器队列大小有不同的要求，使用模拟器有针对性地探索队列大小组合是一个很好的方法。

本文模拟器基于新一代的国产主流通用 CPU 访存子系统设计[17,18]，是一款基于 trace 的周期精确的模拟器，其结构如图 2 所示。

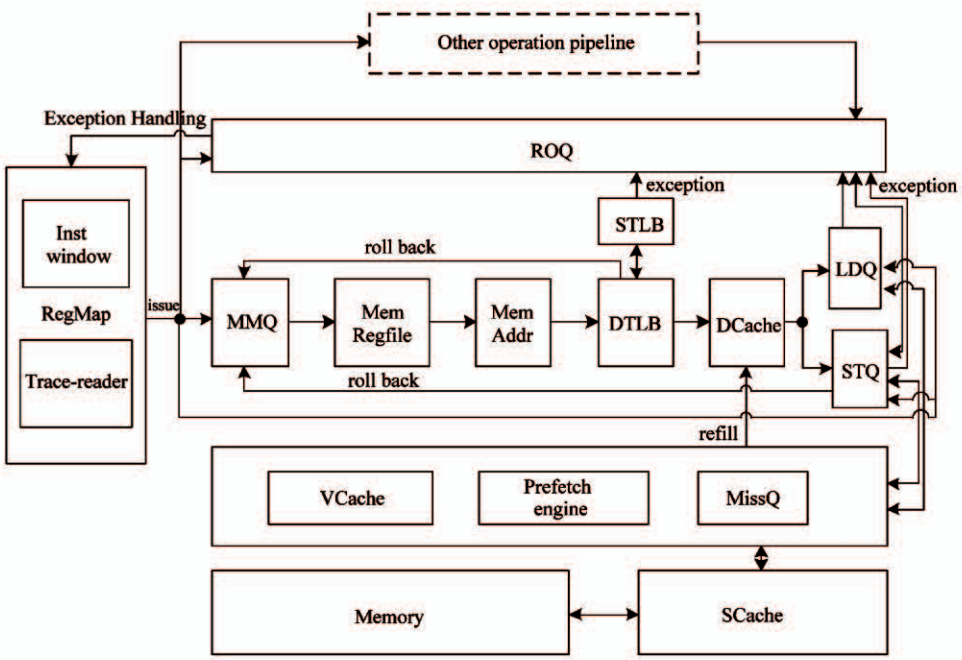


图 2　模拟器整体设计

模拟器的输入是程序访存流 trace。访存流是一个程序完整执行过程中访存指令的动态执行信息，包括 PC 值、访存地址、访存指令类型和数据位宽。模拟器在 RegMap 部件进行 trace 的读取并依

据 PC 的差值进行其他指令的插入,进而模拟出完整的程序执行流,发送往其余部件。程序可以根据要设计的处理器面向的领域和需求由设计人员自行选取,得到该程序的访存流 trace 即可使用。

模拟器对访存子系统相关部件等进行了详细的描述,优化掉处理器其他部分的无关细节避免影响模拟器速度。访存子系统主要由访存流水线和存储器层次访问部分组成。访存发射队列 MMQ、访存数据存取堆 MemRegfile、访存地址生成 MemAddr、数据快表 DTLB、数据 Cache DCache 和 LDQ/STQ 构成了访存流水线各流水级。模拟器详细模拟了这些部件的内部构成和它们之间的传递访问关系,并实现了多种可选择的处理冲突策略。模拟器中还实现了 STLB 和 3 级 Cache 层次以及预取、storefill 等存储器层次访问过程中的相关机制。重排序队列对访存子系统影响较大,所以也对此进行了详细的模拟。图 2中实线部分的方框均为模拟器重点模拟部件,箭头连线表示数据或控制传输,虚线框为简单模拟的处理器其他部件。

软件模拟器除了实现便捷、灵活配置外,还可以在各个阶段加入计数变量,在程序模拟过程中统计到多种需要观察的值,从而掌握程序行为和处理器结构更全面的信息。

2.3 回归模型设计

回归模型是在定义域范围内自变量和因变量之间关系的数学表示。线性回归模型被广泛用于参数重要性估计和定义域范围内任意点的对应输出变量预测[13]。

多元线性模型基本形式可以表示如下:

$$y = \beta_0 + \sum_{i=1}^{n} \beta_i X_i + \epsilon \tag{1}$$

其中,y 是因变量,β_0 是常数项,该公式有 n 个自变量 X_i,每个自变量对应有系数 β_i,ϵ 是误差项。

对于复杂的情况,线性的关系往往是不够的,对于每个 X_i 可以构建其非线性的映射,将变量先做 X_i 的非线性函数,然后进行线性的回归拟合。

从前面关于访存队列大小对整体性能的影响分析来看,队列本身的大小在一定范围内对整体性能有着决定性的作用。这个范围的大小除了取决于处理器其他部件的参数外,还取决于访存子系统中其他队列的大小。

为了体现出这种关系,首先构造 X_i 关于队列大小的函数,再使用线性回归方程构建成为处理器性能关于队列大小的方程式。因为 ROQ 虽然不属于访存子系统中独有,但该队列对访存过程和访存队列的提交有着很大的影响,所以模型中考虑将其作为特征变量。

考虑到相互之间的大小影响,X_i 关于队列大小的子函数采用比值的形式更合适。第 1 类函数使用 2 个队列的比值体现队列间直接的影响;第 2 类函数使用 2 个队列的乘积与一个队列的比体现这 2 个队列和另一个队列的相互影响;第 3 类函数使用 3 个队列的乘积与一个队列的比体现这 4 个队列之间的关系。模型如式(2)所示。

$$cycle = \theta_0 + \sum_{1\leqslant a_1,\, b_1\leqslant 4,\, 1\leqslant i\leqslant 12}^{a_1\neq b_1} \theta_i \frac{S_{b_1}}{S_{a_1}} + \sum_{1\leqslant a_2,\, b_2,\, c_2\leqslant 4,\, 13\leqslant j\leqslant 24}^{a_2\neq b_2,\, a_2\neq c_2,\, b_2\neq c_2} \theta_j \frac{S_{b_2}S_{c_2}}{S_{a_2}} + \sum_{1\leqslant a_3,\, b_3,\, c_3,\, d_3\leqslant 4,\, 13\leqslant k\leqslant 24}^{a_3,\, b_3,\, c_3,\, d_3\text{互不相同}} \theta_k \frac{S_{b_3}S_{c_3}S_{d_3}}{S_{a_3}} \tag{2}$$

在使用软件模拟器的基础上,回归模型将进一步节省模拟时间,扩大设计空间探索范围,以找到符合条件较优的解。[19]

3 模拟器实现与模型构建

3.1 访存子系统模拟器实现

模拟器使用 C + + 语言实现,所有队列共有特性继承自父类,每个队列不同的退出机制实现在队列子类的退出函数中,相互访问、例外、回滚等也实现在各自的函数中,具有良好的模块性。可能存在的一些解决数据冲突问题的策略使用宏定义在对应函数中实现,方便配置不同硬件参数。

模拟器中的 RegMap 对应于硬件指令分配,开启一个线程读取访存指令 trace 并产生虚拟非访存指令,另一个线程按照处理器前端指令供应速率向 ROQ、MMQ、LDQ 和 STQ 分配指令。这部分可能会发生例外,导致流水线清空,所以使用 2 个指针记录已分配出去的位置和尚未提交的位置。

如图 2 所示,访存流水线的第 1 级访存发射队

列 MMQ 将源寄存器已就绪的指令按照一定顺序发射到访存流水线中。指令经过 MemRegfile、MemAddr、TLB 和 Cache，最后进入 LDQ 或 STQ 中，如果有数据冲突，则触发冲突的解决方法；否则，指令等待访存或者提交。模拟器实现时，将访问 DTLB 作为一级，下一级访问 DCache，模拟器时序和结果上等同于硬件中同时访问 DTLB 和 DCache，下一个时钟周期进行 Tag 比较。

模拟器实现了可扩展、多层次的 Cache 结构，其大小、延迟、替换策略等均可配置。DCache 失效请求由 missq 进行访问管理和 Cache 重填。同时，还实现了数据预取和 store fill 机制。

串行执行程序的软件模拟器为了模拟硬件的并行运行情况，每个部件的输入部分都必须是上一个时钟周期的值，模拟的同一个时钟周期内的结果不能被使用。软件对于流水级的逆序描述满足上述要求。模拟器整体功能部件运行伪代码如表 4 所示。

表 4　模拟器整体功能部件运行伪代码

```
while( ! trace _ finished) {
        STQ. retire( ) ;
        LDQ. retire( ) ;
        ROQ. retire( ) ;
        LDQ. arrive( ) ;
        STQ. arrive( ) ;
        DCache. access( ) ;
        MMU. access( ) ;
        Vaddr( ) ;
        mem _ register _ file( ) ;
        MMQ _ issue( ) ;
        other _ queue. issue( ) ;
        RegMap. dispatch( ) ;
        cycle + + ;
}
```

3.2　访存子系统模拟器性能数据

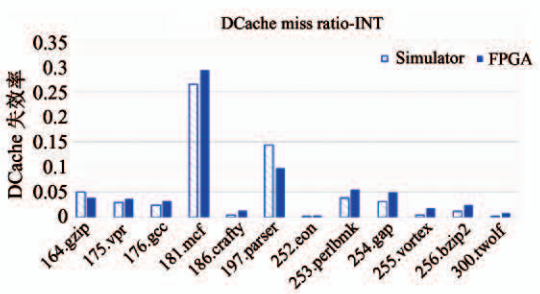
模拟器所模拟目标处理器为本课题组所设计的新一代国产处理器，模拟器校准使用该处理器 RTL 设计代码的 FPGA 原型验证平台产生的数据，该平台与目标处理器有着极高的相似度。

实验中使用程序测试集为 SPEC CPU 2000，是由标准性能评价机构（standard performance ealuation corporation，SPEC）开发的用于评测 CPU 性能的基准程序测试组，是常用的通用处理器平台测试集。

FPGA 平台运行的处理器虽然主频与真实机器有着很大的差异，总体绝对时间上会不同，但是处理器的时钟周期在 FPGA 上运行和 ASIC 芯片上是一样的，所以校准和预测的数据统计都使用硬件计数器的时钟周期数来表示处理器性能。在访存周期方面，将模拟器访存周期数设置成和 FPGA 访存周期相同，图 3 是这种情况下 SPEC CPU 2000 访存流在模拟器和在 FPGA 平台上运行硬件性能计数器得出的 DCache 失效率对比。图 4 是二者的 MMQ 和 STQ 阻塞率对比，LDQ 阻塞在 FPGA 和模拟器中都几乎为 0。由于乱序执行的原因，模拟器运行的程序流和 SPEC CPU 2000 程序流不完全一样，且模拟器仅有访存子系统的详细模拟，添加的指令的执行时间情况也不相同，因此图 4 中数据有些差异，但是差异并不算很大且趋势相同。

3.3　模型拟合及误差分析

实验中模拟器分别改变 MMQ、ROQ、LDQ、STQ 大小组成不同的大小组合运行程序，得到相应的总时钟周期。队列大小的选取应在设计空间大小范围内，并且分布较为均匀且组合随机，以便对整个设计空间不同组合附近都有模拟，从而得到更符合模拟情况的模型参数。



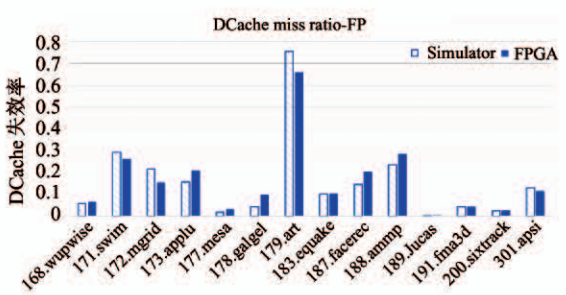


图 3　SPEC CPU 2000 load-store 流在模拟器和 FPGA 原型验证平台上的 DCache 失效率

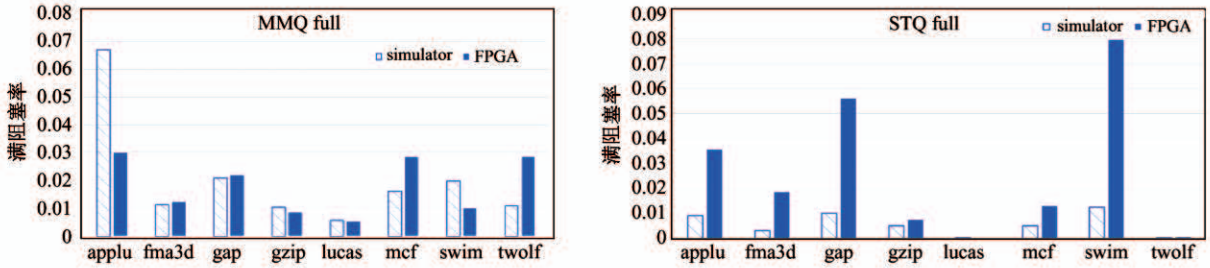


图 4　SPEC CPU 2000 load-store 流在模拟器和 FPGA 原型验证平台上的队列满阻塞率

将得到的数据顺序随机,然后分成 5 份,使用其中的 4 份数据依据 2.3 节提出的模型式(2)进行线性回归拟合,得到各个参数,此时便得到了该处理器设计在测试程序下的性能模型,通过输入队列大小可以预测处理器在该程序集下的运行时钟周期数。然后将未参与拟合的一份数据的队列大小和拟合出来的参数带入式(2),计算得到预测时钟周期数,与对应模拟器得到数据进行对比,得到测试集结果。将 4 份参与拟合的训练数据的队列大小输入公式,得到的时钟周期数和模拟器得到的周期数进行对比,得到训练集结果。实验使用交叉验证,将这 5 份数据都作为一次测试数据,分别进行 5 次上述实验,得到 5 组实验结果。

表 5 为 fma3d 程序 load-store 流的数据 5 次交叉验证的线性回归拟合参数、平均误差和标准差。表 6 是 mcf、applu、twolf、gap 和 swim 程序 load-store 流的数据 5 次交叉验证平均误差和标准差,限于篇幅没有列出 5 次拟合的参数,每个程序 5 次的拟合参数结果相似。

表 5　fma3d 程序 load-store 流的数据 5 次交叉验证结果

	test_1	test_2	test_3	test_4	test_5
	4.3553E+06	4.3562E+06	4.3470E+06	4.3609E+06	4.3515E+06
	3.4389E+04	3.4985E+04	3.3407E+04	3.8410E+04	3.5116E+04
	3.7973E+04	3.8181E+04	3.7208E+04	3.9176E+04	3.9429E+04
	3.8088E+04	3.7869E+04	3.2425E+04	3.8124E+04	3.7988E+04
	3.0895E+05	3.1768E+05	3.1468E+05	3.2129E+05	3.5830E+05
	3.9205E+05	3.9858E+05	4.0406E+05	3.9239E+05	3.9732E+05
	3.9575E+05	3.9778E+05	4.0971E+05	4.0104E+05	4.1132E+05
	1.2172E+05	1.2125E+05	1.1660E+05	1.2589E+05	1.3384E+05
	1.0759E+05	1.0952E+05	1.1110E+05	1.0714E+05	1.1070E+05
	1.3757E+05	1.3497E+05	1.3769E+05	1.3553E+05	1.3484E+05
	5.4238E+04	5.2222E+04	5.0706E+04	5.3567E+04	5.6643E+04
	5.1354E+04	5.0651E+04	4.9196E+04	4.9570E+04	5.1954E+04
	5.6515E+04	5.3961E+04	5.5506E+04	5.5887E+04	5.6906E+04
	−3.0572E+02	−3.1985E+02	−3.2122E+02	−3.4952E+02	−3.1829E+02
权重 θ 值	−4.0268E+02	−4.1221E+02	−3.8249E+02	−4.3628E+02	−3.9947E+02
	−5.3110E+02	−5.4495E+02	−5.4819E+02	−5.1815E+02	−5.3134E+02
	−3.4257E+03	−3.6295E+03	−3.5941E+03	−3.5814E+03	−4.0916E+03
	−4.1890E+03	−4.2992E+03	−4.3157E+03	−4.3734E+03	−5.0770E+03
	−4.2592E+03	−4.3506E+03	−4.4476E+03	−4.4095E+03	−4.4722E+03
	−1.0899E+03	−1.1215E+03	−1.0962E+03	−1.1332E+03	−1.2749E+03
	−1.6830E+03	−1.6510E+03	−1.6237E+03	−1.7163E+03	−1.8144E+03
	−1.2650E+03	−1.2704E+03	−1.2901E+03	−1.2681E+03	−1.2951E+03

表 5 续

	-5.8206E+02	-5.5436E+02	-5.1739E+02	-5.5177E+02	-6.1636E+02
	-7.3330E+02	-6.6825E+02	-6.7193E+02	-7.1634E+02	-7.6623E+02
	-5.4649E+02	-5.1667E+02	-5.0737E+02	-5.1767E+02	-5.4792E+02
	7.0814E+00	7.4773E+00	7.8464E+00	7.3393E+00	7.0864E+00
	5.4973E+01	5.7413E+01	5.6865E+01	5.8194E+01	6.6617E+01
	1.6773E+01	1.6953E+01	1.6663E+01	1.7414E+01	1.9315E+01
	8.9641E+00	8.2918E+00	7.9911E+00	8.4751E+00	9.4392E+00
测试集平均误差	0.030195	0.030703	0.031099	0.031361	0.033227
测试集误差的标准差	0.040216	0.041219	0.039365	0.043261	0.044086
训练集平均误差	0.030939	0.030776	0.030980	0.030554	0.030474
训练集误差的标准差	0.041621	0.041315	0.041688	0.040763	0.040999

表 6 mcf,applu,twolf,gap 和 swim 程序 load-store 流的数据 5 次交叉验证结果

			test_1	test_2	test_3	test_4	test_5
mcf	测试集	平均误差	0.046697	0.047055	0.047701	0.047679	0.047806
		误差的标准差	0.073517	0.073342	0.074825	0.074878	0.074811
	训练集	平均误差	0.046853	0.047280	0.047231	0.047505	0.047189
		误差的标准差	0.073369	0.073223	0.073516	0.074132	0.073063
applu	测试集	平均误差	0.037130	0.037210	0.036831	0.038770	0.038416
		误差的标准差	0.052747	0.052930	0.052453	0.056373	0.056472
	训练集	平均误差	0.038315	0.038801	0.038105	0.038546	0.037645
		误差的标准差	0.055049	0.055634	0.054498	0.055286	0.053956
twolf	测试集	平均误差	0.033865	0.034079	0.035047	0.034950	0.035237
		误差的标准差	0.046935	0.047076	0.047935	0.047670	0.048114
	训练集	平均误差	0.034990	0.035196	0.035254	0.034756	0.035171
		误差的标准差	0.048332	0.048323	0.048442	0.047540	0.048379
gap	测试集	平均误差	0.028641	0.028487	0.030006	0.028988	0.029814
		误差的标准差	0.042517	0.044647	0.047296	0.045083	0.047597
	训练集	平均误差	0.029269	0.029314	0.028629	0.028967	0.028605
		误差的标准差	0.045614	0.045221	0.044505	0.044975	0.044376
swim	测试集	平均误差	0.076222	0.075953	0.075796	0.078448	0.080413
		误差的标准差	0.099814	0.101870	0.099501	0.104120	0.106490
	训练集	平均误差	0.078051	0.076132	0.077363	0.076501	0.076995
		误差的标准差	0.103200	0.100790	0.102230	0.100880	0.101370

实验中,测试数据没有参与训练,其结果可以反映模型的准确程度。将训练数据使用模型计算得到的结果和测试数据得到的结果相比较,保证参数的训练没有过拟合。使用交叉验证,检查结果是否稳定,排除数据的偶然性。对 SPECCPU 测试集中不同类型的应用程序进行测试,模型结果表现都稳定且误差较小,表明该模型对于不同类型应用程序都有效。表 5 和表 6 实验结果证明式(2)在这些程序所测得数据集上表现良好,误差在 10% 以内,无过拟合现象,结果稳定。

4 通过模型预测队列大小

对于面向不同领域的处理器将会有着不同的设计需求,处理器在性能要求方面有着不同的评价标准。在使用本文所采用的方法时,可以根据设计修

改模拟器的构成,使用面向该领域通用的程序测试集(如 SPECCPU、EEMBC、COREMARK、LINPACK 等)或者设计者根据需求构建的程序集,得到队列大小配置相应的性能数据。采用本文提出的回归模型,得到目标类应用相应的模型系数,即可使用该模型预测队列大小和性能的关系,从而根据设计的需求得到合理的队列大小组合。

已经拟合好的回归模型可以预测整个设计空间范围内不同队列大小搭配的性能结果,使得对这部分的设计空间探索更准确和容易。

本实验中,为了可视化数据,图 5 展示了在 ROQ128、MMQ32 情况下 LDQ 和 STQ 不同大小的性能情况,ROQ128、LDQ64 情况下 MMQ 和 STQ 不同大小的性能情况以及在 ROQ128、STQ48 情况下 MMQ 和 LDQ 不同大小的性能情况。

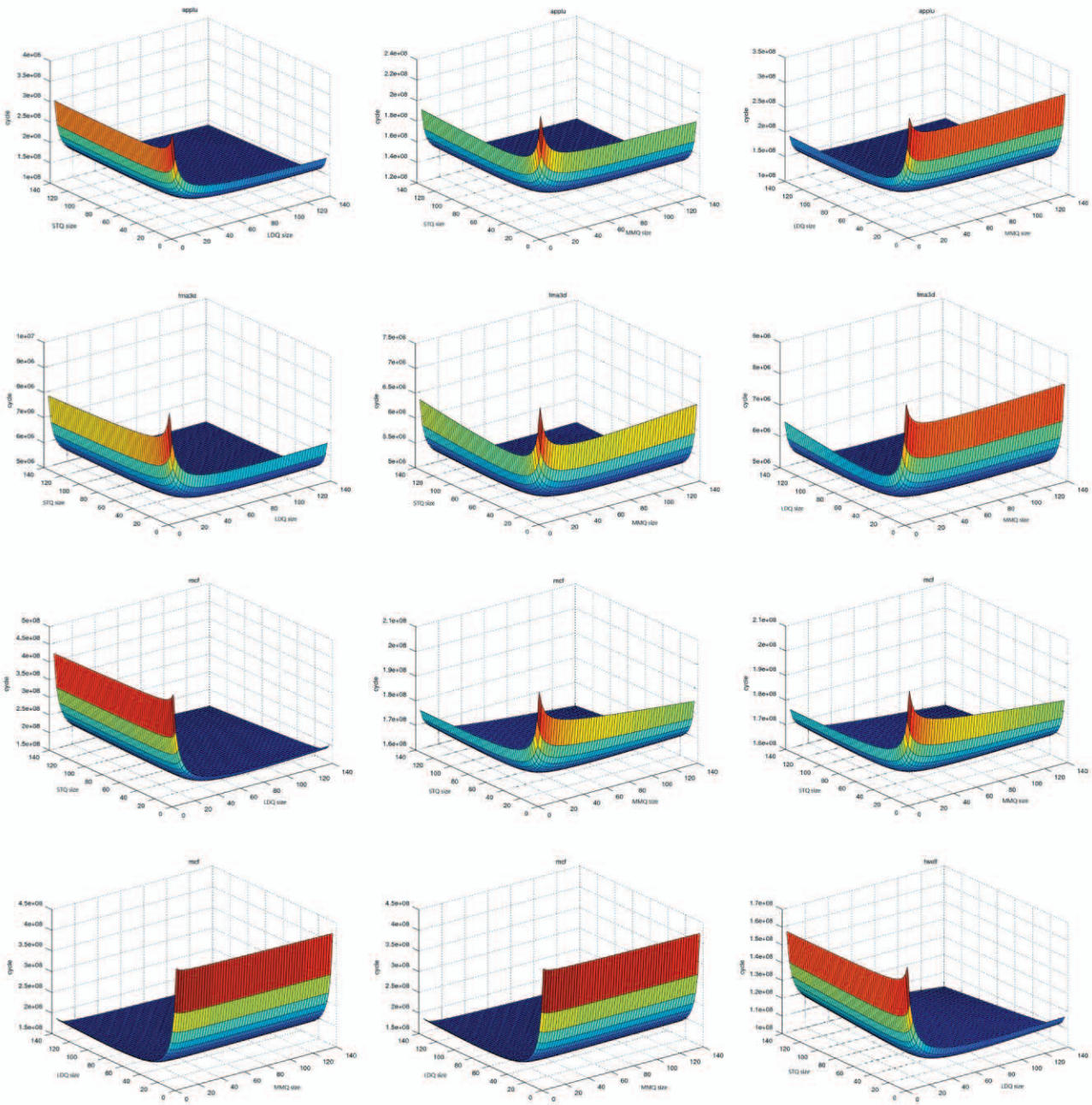


图 5　不同程序 MMQ,ROQ,LDQ,STQ 固定其中 2 个队列大小,另外 2 个队列大小对性能的影响

对于回归模型,可以使用梯度下降等求最优解的方法找到合适的队列大小组合,从而达到资源面积利用和性能折中的最佳点。本实验的设计空间中队列大小参数都是整数,设计空间虽然大,但是有限个数的点以及回归模型 $O(1)$ 的计算复杂度,使得即使遍历整个设计空间也十分快。快速得到整个设计空间中不同队列大小的性能预测结果后,便可以很快找到符合面积功耗和性能要求的队列大小配置。例如,设置条件为队列大小每增 10、总时间减少 1% 以上则选择改变队列大小,根据这个条件找到最佳队列大小组合如表 7 所示。表 7 中,有给定 3 个队列,预测某程序达到满足阈值条件的最小队列值,或者给定 1 个、2 个固定队列大小,预测某程序达到满足阈值条件的最佳队列组合值。阈值的设定和队列大小固定的数目可以根据实际需求改变,通过得到的模型快速评估性能,进而得到不同条件下最佳队列大小的组合方案。

表 7 使用线性回归模型得到满足阈值条件的最佳队列大小组合

给定队列大小	预测队列大小						
		fma3d	mcf	applu	twolf	gap	swim
MMQ = 32, ROQ = 128, LDQ = 64	STQ	31	25	35	18	31	41
MMQ = 32, ROQ = 128, STQ = 48	LDQ	43	59	58	41	42	54
MMQ = 32, LDQ = 64, STQ = 48	ROQ	88	170	144	92	85	128
ROQ = 128, LDQ = 64, STQ = 48	MMQ	29	18	35	38	32	20
MMQ = 32	LDQ	43	57	57	39	41	53
ROQ = 128	STQ	31	25	36	18	30	40
ROQ = 128	MMQ	28	18	35	33	30	20
	LDQ	43	55	57	38	41	52
	STQ	31	25	35	18	30	40
ROQ = 144	MMQ	30	20	36	36	32	20
	LDQ	45	61	62	41	44	58
	STQ	33	27	38	20	33	44
ROQ = 160	MMQ	32	22	37	38	33	18
	LDQ	48	67	67	43	46	64
	STQ	35	30	40	22	35	48

5 结 论

处理器访存在程序运行时占有很大的时间消耗,超标量处理器设计过程中,为了实现访存的推测执行,通常使用一些队列来保存各个访存行为的信息。本文讨论了访存流水线中各队列大小对性能的影响原因和方式,采用了软件模拟器和分析模型结合的建模方法,对队列大小组合进行探索,找到队列大小组合平衡点使得硬件资源利用率达到最优。本文提出了一种面向访存子系统的回归模型,并设计了相应的软件模拟器,经实验验证,软件模拟器能较准确地模拟访存行为,回归模型适合于访存子系统的建模使用,对数据的拟合误差较小且结果稳定,可用于下一代处理器访存子系统设计空间探索和对性能的预测。

参考文献

[1] Wulf W A, McKee S A. Hitting the memory wall: implications of the obvious[J]. *ACM SIGARCH Computer Architecture News*, 1995, 23(1): 20-24

[2] Wilkes M V. The memory wall and the CMOS end-point [J]. *ACM SIGARCH Computer Architecture News*, 1995, 23(4): 4-6

[3] Karkhanis T S, Smith J E. A first-order superscalar processor model[C]// Proceedings of the 31st Annual International Symposium on Computer Architecture, Munich Germany, 2004: 338-349

[4] Chen X E, Aamodt T M. Hybrid analytical modeling of pending cache hits, data prefetching, and MSHRs[J]. *ACM Transactions on Architecture and Code Optimization*, 2011, 8(3):1-28

[5] Park I, Ooi C L, Vijaykumar T N, et al. Reducing design complexity of the load/store queue[C]//Proceedings of the 36th Annual International Symposium on Microarchitecture, San Diego, USA, 2003: 411-422

[6] Ros A, Kaxiras S. The superfluous load queue[C]//Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Fukuoka City, Japan, 2018: 95-107

[7] Josipovic L, Brisk P, Ienne P. An out-of-order load-store queue for spatial computing[J]. *ACM Transactions on Embedded Computing Systems* (*TECS*), 2017, 16(5s): 125

[8] Sethumadhavan S, Roesner F, Emer J S, et al. Late-binding: enabling unordered load-store queues[J]. *ACM SIGARCH Computer Architecture News*, 2007, 35(2): 347-357

[9] Lodi A, Martello S, Vigo D. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems[J]. *Informs Journal on Computing*, 2017, 11(4):345-357

[10] Bhunia A K, Das A, Bhunia A K, et al. Handwriting recognition in low-resource scripts using adversarial learning[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Long Beach, USA, 2019: 4767-4776

[11] Zhang C, Liang B, Huang Z, et al. Look more than once: an accurate detector for text of arbitrary shapes[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Long Beach, USA, 2019: 10552-10561

[12] Wang T, Liu C. Fully convolutional network based skeletonization for handwritten Chinese characters[C]//Proceedings of the 30th Innovative Applications of Artificial Intelligence, New Orleans, USA, 2018: 2540-2547

[13] Joseph P J, Vaswani K, Thazhuthaveetil M J. Construction and use of linear regression models for processor performance analysis[C]//The 12th International Symposium on High-Performance Computer Architecture, Austin, USA, 2006: 99-108

[14] Henning J L. SPEC CPU2000: measuring CPU performance in the new millennium[J]. *Computer*, 2000, 33(7): 28-35

[15] Abella J, González A. SAMIE-LSQ: set-associative multiple-instruction entry load/store queue[C]//Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium, Rhodes Island, Greece, 2006: 10

[16] Baugh L, Zilles C. Decomposing the load-store queue by function for power reduction and scalability[J]. *IBM Journal of Research and Development*, 2006, 50(2.3): 287-297

[17] 吴瑞阳, 汪文祥, 王焕东, 等. 龙芯 GS464E 处理器核架构设计[J]. 中国科学:信息科学, 2015, 45(4): 480-500

[18] 胡伟武. 自主 CPU 发展道路及在航天领域应用[J]. 上海航天, 2019, 36(1):5-13

[19] Clapp R M, Dimitrov M, Kumar K, et al. Quantifying the performance impact of memory latency and bandwidth for big data workloads[C]//IEEE International Symposium on Workload Characterization, Atlanta, USA, 2015: 213-224

Performance modeling of key queues in processor memory subsystem

Li Wenqing[*,**], Wu Wei[*,**], Zhang Longbing[*,**,***], Xiao Junhua[*,*****], Wang Jian[*,**,***]

([*] State Key Laboratory of Computer Architecture, Institute of Computer Technology, Chinese Academy of Sciences, Beijing 100190)

([**] University of Chinese Academy of Sciences, Beijing 100049)

([***] Loongson Technology Corporation Limited, Beijing 100190)

Abstract

Memory access performance of processors has a great impact on the overall performance, so the design of memory subsystem is particularly important. There are many key queues in the memory subsystem of high performance superscalar processors. How to quickly get the trade-off of design becomes the key for design. This paper uses a way that combines software simulator with regression model. It proposes a regression analysis model for key queues of memory subsystem whose corresponding simulator is designed and implemented. By calibrating the accuracy of the software simulator with the target processor's FPGA prototype verification platform and using the regression model to analyze the simulation data of the simulator, the results show that the experimental verification results are stable and the error of the tested program is less than 10%. This modeling method can quantitatively analyze the relationship between key queue size and performance in memory subsystem, effectively expand the scope of hardware design space exploration, and accelerate the optimization design of memory subsystem for high performance processors.

Key words: processor design space exploration, memory subsystem, software simulator, regression model