一种多核间内存公平调度模型

刘虎球 赵 鹏
(清华大学计算机科学与技术学院 北京 100084)

摘 要 计算机的发展已进入多核时代,在共享内存的计算机系统中,内存需要为多核提供公平的服务.文中提出一种在多核环境下的内存公平调度模型,将多核调度问题转化为一个数学模型,极大地拓展了研究多核调度的思路,然后通过启发式算法求解,得到了一个性能较优的公平调度算法 FQ-SJF.基准 soplex 的实验结果表明,相比 FR-FCFS 调度算法,平均读取延迟比 FR-FCFS 减小了 10.6%,有效验证了提出的多核调度模型.

关键词 多核内存调度;多核调度模型;公平队列调度
中图法分类号 TP311 **DOI 号** 10.3724/SP.J.1016.2013.02191

A Multi-Core Fair Memory Scheduling Model

LIU Hu-Qiu ZHAO Peng
(*School of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Abstract The development of computers has entered the era of multi-core. In shared memory multi-core computer systems, we need to provide a fair service. This paper presents a memory fair scheduling model in a multi-core environment, which switchs the multi-core scheduling problem into a mathematical model, greatly expanding the idea of multi-core scheduling, and then we use the heuristic algorithm to get a better performance of the fair scheduling FQ-SJF. Through experiments of soplex benchmark, compared with FR-FCFS scheduling algorithm, its system throughput increased 10.6%, which effectively validated the model.

Keywords multi-core's memory scheduling; multi-core scheduling model; fair queuing scheduling

1 引 言

近年来,CPU 的多核技术得到了大力发展,在共享内存的多核体系结构中,多个处理器需要访问同一部分系统内存,因此内存调度的重要性日益明显,逐渐成为国内外的研究热点[1-2].人们也希望通过设计较好的内存体系结构来解决处理器和内存之间的发展速率差[3-4].

在内存调度中,主要完成的是接收内存请求队列中的请求,然后在算法控制下执行被选中的请求,调度算法包括 FCFS、RR 等,各种算法都存在相应的优劣势.但是总体上,内存调度算法均主要立足于提高系统的吞吐率,同时尽量减少进程的访问等待时间.合理调度能够有效地利用 Cache 中已经缓存的数据,提高 Cache 的利用率,从而缩短了 CPU 访问内存数据的等待时间[5-6].

本文首先提出了一个通用的内存公平调度模型.并通过理论上论证,目前的多种调度算法均为该模型的一个解,并且通过设置不同模型参数,使得该模型具有不同的代表意义.随后,通过使用启发式算法为模型构造了一个解,并通过模拟仿真实现该内

收稿日期:2011-07-01;最终修改稿收到日期:2013-05-27.本课题得到国家自然科学基金(61103020)资助.**刘虎球**,男,1989 年生,博士研究生,主要研究方向为多核间内存调度、存储管理等.E-mail:huqiuliu@gmail.com.**赵 鹏**,男,1985 年生,博士研究生,主要研究方向为多核体系结构、操作系统.

存调度算法,通过修改 M5 模拟器,并在 M5 模拟器上运行各种待测试的基准应用,与现行的几种多核调度算法进行性能上的综合比较并验证了模型的可用性.

2 相关工作

2.1 存储系统

为了有效定位和说明内存调度,首先简要地对计算机的基本组成和软硬件构造予以说明.现代计算机都以存储器为核心,这与古典的冯·诺依曼结构有所不同,从程序员的角度,计算机要开始工作,必须把程序装载到计算机的内存中.该过程可以归为存储管理过程.

在程序执行过程中,中央控制器所需要的指令从存储器中读取,运算器所需要的数据同样从内存中读取,执行完的结果同样需要写回到内存中,在当前的主流系统结构中,IO 设备也可以直接和存储器进行数据交换,可见存储器已经成为计算机的核心部件.对于内存调度,从操作系统的层次上看,主要由硬件完成,这是因为当前的体系结构中,存储器内部结构对于存储管理是透明的.当 CPU 将读取内存的请求交由内存控制器[7]后,此时完全由内存调度来完成,其中包括从内存请求队列中获取内存请求,然后按照内存调度算法处理该请求队列,待请求被执行完时,通知 CPU 可以继续执行,因而对于操作系统来说,是完全透明的.

在当前的多核系统中,各处理器通过共享内存资源完成一些混合应用,同时也通过内存共享完成部分核间数据共享和通信[8-9].因此内存调度更需要均衡地响应各个处理器的内存读写请求,从而有效提高各个处理器的利用率.但内存调度仍旧完全由存储器内部控制器完成,从操作系统角度看,属于硬件内部的逻辑.

2.2 时序约束

通过前面的描述,可以知道,内存调度通过有效的管理 Cache 和主存的数据,从而提高系统的 Cache 命中率,缩短存储器的访问周期.但是目前 Cache 和 DRAM 均受到物理材料的限制,受到综合的时序约束,下面将首先简要介绍一下在内存调度中主要考虑的一些时序约束.

Cache 是当前存储体系中访问速率较高的存储芯片,当 CPU 发出读请求时,存储系统首先会根据 CAM(Content Addressable Memory)检查 Cache 是否命中,若命中则直接读取,否则将访问请求提交至内存访问队列中.主存 DRAM 作为存储系统的重要核心部件,内存调度在执行内存请求时将内存中的数据及时推送到存储器中,为了完成有效的内存调度,内存控制器需要对存储器的时序进行规划,当处理完毕时,及时返回数据给 CPU,防止等待时间过长.

2.3 多核内存调度算法

从前文的介绍可以看出,尽管内存调度不属于操作系统范畴,但是对存储器系统性能有着非常重要的作用.和存储系统一样,内存调度同样存在调度目标,其中包括吞吐率、平均等待时间、公平性、Cache 命中率等[10-11],下面将简单地进行介绍.

(1)吞吐率.在存储系统中,吞吐率是指在一段时间内,系统的单位时间的吞吐量,是一种访问速率的量度,该指标主要从 CPU 的角度进行设计,实际中常常使用每个总线周期指令条数(Instructions Per Clock)IPC 指标代替.

(2)平均等待时间.内存请求平均访问的等待时间,该参数较小则表示系统的访问速度较快,同样存储器的访问周期也越小.

(3)公平性.表示各个内存请求的延迟是否公平,在目前的多核调度中,是考虑的另一个重要因素,主要是防止个别请求过长等待,保证公平性和服务质量.

(4)Cache 命中率.通过有效的调度可以提高 Cache 的命中率,提高访问速度,该部分目前逐渐被引入到调度系统中.

目前的调度算法主要考虑了上述因素,而上述指标同样常常用于测试基准的评价指标[12],下面将分别从上述角度对已有的一些多核调度算法做一些简单的介绍.

2.3.1 通用多核调度算法

在计算机的调度算法中,经典的调度算法占据着重要地位,其中包括先来先服务(FCFS)、优先级调度算法、最长时间未被访问(LRU)、最小频率访问(LFU)等,其中 FCFS 算法后来衍生了一个系列的算法,具体在第 2 小节中予以详细介绍.而优先级调度算法使用并不多,针对内存请求队列,内存控制器通过识别请求的优先级标签,从而有效地调度内存,为有实时性要求的请求提供服务保障.

2.3.2 基于 FCFS 衍生的多核调度算法

FCFS 调度算法实现最为简单,人们在研究多核调度时,同样构造了一些基于 FCFS 的衍生算法,其中包括 FR-FCFS(First Ready, First Come First

Service)、FR-VFTF(First Ready, Virtual Finish Time First)等，FR-FCFS，即从请求的就绪队列的队头中获取内存请求，该算法有效地减少了内存控制器在获得内存请求后的就绪等待时间，但是该算法可能引起部分请求等待过长[13]. FR-VFTF 算法和 FR-FCFS 存在细微区别，主要是因为 FR-VFTF 需要估算内存访问请求的虚拟结束时间，优先执行就绪队列中最早结束的访问请求.

2.3.3　基于替换策略系列调度算法

在前面的内存存储体系结构中，Cache 容量尽管受限，但是对存储系统的性能却有着不可忽略的作用. 因此很多调度算法重点考虑了私有 Cache 和共享 Cache 的存储情况[13]，提高 Cache 的命中率，即对一些进入 Cache 的数据行优先考虑，具体的现行基于替换策略的算法主要有按顺序、优先级、行打开、行关闭、最多等待、最少等待等[14-15].

2.3.4　基于 FQ 系列调度算法

在前面的多核调度算法中，都较为关注系统的吞吐率，但是对系统的线程级别的延迟并未量化，调度算法中还出现了一部分以公平策略为核心的调度算法[16-17]，其中主要体现在各个请求的延迟均等或各个线程访问延迟一致，当然该类算法对系统的性能也有一定影响，一般体现在系统的吞吐率有所下降，不同算法在性能和公平性上侧重点略微不同. FQ(Fair Queue)算法以 FQ-VFTF[13](Fair Queue, Virtual Finish Time First)最为典型，即从就绪队列的头部选择最早完成的请求执行调度. 需要注意的是，该算法的最早完成时间使用虚拟完成时间进行估算，各个 CPU 等比例使用内存. 在调度的过程中，尽量使各个 CPU 发出的各个请求延迟在统计意义上均等，同时始终保证在任何时刻，各个 CPU 得到的服务仅和当前的请求现状有关，而和历史上的请求没有关系.

3　多核调度模型

实际中，在很多情况下，调度算法需要为应用程序提供较高的吞吐率，但是还需要保证提供 QoS (Quality of Service). 在网络中，曾经有 Fair Queue 的概念，在多核内存调度领域中，尽管引入了该思想，但是算法大都基于请求来考虑. 另外，目前的多核调度算法的研究大都从传统算法进行构造，然后使用模拟器反复验证. 实际上，作为一个多核调度问题，完全可以将其抽象成一个具体的数学模型，从而通过使用数学模型的求解方法直接求解出高效的多核调度算法.

3.1　模型假设

由于实际中内存时序相对较为复杂，不利于进行实际中的抽象建模并分析，因此在本次研究中，以简化的 DDR2 时序为背景做出一定的模型假设来进行理论建模分析，鉴于研究需要，参考文献[13]对模型先做出如下假设：

(1)内存的物理结构和上文中描述的一致，即主要包括 Channel、Rank、Bank、Row、Col，并且允许在 Bank 上进行并行调度，其中 Bank 和 Channel 调度器的调度算法可以不一致.

(2)同一个进程或线程的请求需要按序完成，即一个线程的第 k 个内存访问请求必须建立在第 $k-1$ 个请求已经完成的基础上，该假设主要是为了保证 CPU 执行指令的线程级有序性.

(3)内存控制器可以通过请求直接获得该请求对应的 CPU、线程、请求发起时间、优先级等，在后文中也称为贴标签.

(4)内存访问请求有序进入内存请求的队列，内存控制器可以自由地执行队列中的任意请求，同时执行完毕时将数据返回到 Cache，然后通知 CPU 访问该数据.

(5)内存调度关注的是目前通用的调度场景下的情形，忽略了各种特殊应用场景下的其它时序和延迟要求. 例如在一些嵌入式系统中，对实时性要求相对较高，而对吞吐率却并不严格，此种情况未予以考虑.

3.2　符号约定

为了后面描述模型的方便，需要对 Bank、Channel 等时间进行符号化约定，因此对访问的队列也进行符号化约定，此外还对描述模型所需的符号进行了初步约定，具体如表 1 所示.

表 1　队列符号约定

符号	含义
N_{cpu}	CPU 个数
N_{thread}	线程个数
N_{request}^{i}	线程 i 的请求个数
N_{i}^{bank}	第 i 个 Channel 对应的 Bank 数目
N_{channel}	内存系统的 Channel 数目
m_{i}^{k}	第 i 个线程的第 k 个内存请求
a_{i}^{k}	第 m_{i}^{k} 个请求的到达时间
L_{i}^{k}	第 m_{i}^{k} 个请求的总服务时间
t_{RCD}	内存的行寻址到列寻址延迟时间
t_{RP}	内存行地址控制器预充电时间
t_{RFC}	内存的行刷新周期时间
t_{RAS}	一个行地址从激活到复位的时间

（续 表）

符号	含义
$B_l^j.L_i^k$	第 m_i^k 个请求在 Channel l 上的 Bank j 上的服务时间
$B_l^j.S_i^k$	第 m_i^k 个请求在 Channel l 上的 Bank j 上的服务开始时间
$B_l^j.F_i^k$	第 m_i^k 个请求在 Channel l 上的 Bank j 上的服务结束时间
$C_l.L_i^k$	第 m_i^k 个请求在 Channel l 上的服务所需时间
$C_l.S_i^k$	第 m_i^k 个请求在 Channel l 上的服务开始时间
$C_l.F_i^k$	第 m_i^k 个请求在 Channel l 上的服务结束时间
t_{CL}	内存的内存读操作前列地址控制器的潜伏时间，即读取的列地址写入后，经过该时间可以读取总线上的数据
t_{WL}	内存的内存写操作前列地址控制器的潜伏时间，即对地址控制器赋值后，应在该时间内在总线上准备好待写入数据
t_{WR}	对一个激活的 Bank 中完成有效的写操作及预充电前，必须等待的时钟周期数

3.3 多核调度优化模型

根据前面的介绍，目前的主流算法大都关注了吞吐率和公平性，实际中这两个评价指标也足以概括当前的主流调度算法. 表 2 对模型使用的符号做了进一步的补充说明.

表 2 模型符号约定

符号	含义
λ_1	吞吐率的加权系数
λ_2	公平性的加权系数
F	公平性，其中该值越大越好
T	吞吐率，其中该值越大越好
$W_{total}.m_i^k$	第 m_i^k 个请求的总读取等待时间
$W_{ready}.m_i^k$	第 m_i^k 个请求的就绪等待时间
$W_{queue}.m_i^k$	第 m_i^k 个请求的在就绪队列中等待时间
$W_{bank}.m_i^k$	第 m_i^k 个请求的在 Bank 队列中等待时间
$W_{channel}.m_i^k$	第 m_i^k 个请求的在 Channel 队列中等待时间
$S_l^j.m_i^k$	第 m_i^k 个请求的 Channel l 上的 Bank j 上的执行次序

使用层次分析法的建模方法，得到最优化模型，模型以吞吐率和公平性为目标函数，满足内存访问和调度的一些时序约束. 综合上文描述，具体模型如下所示：

$$\max \lambda_1 \times F + \lambda_2 \times T \tag{1}$$

s. t.

$$F = \min\left\{\frac{L_i^k}{W_{total}.m_i^k + L_i^k},\ i=1\cdots N_{thread},\ k=1\cdots N_{request}^i\right\} \tag{2}$$

$$T = \frac{\sum_{i=1,\ k=1}^{N_{thread},\ N_{request}^i} \text{if } m_i^k \text{ completed, } L_i^k \text{ else } 0}{\sum_{i=1,\ k=1}^{N_{thread},\ N_{request}^i} L_i^k} \tag{3}$$

$$W_{total}.m_i^k = W_{ready}.m_i^k + W_{queue}.m_i^k + W_{bank}.m_i^k + W_{channel}.m_i^k \tag{4}$$

$$L_i^k = B_l^j.L_i^k + C_l.L_i^k,\ l=1\cdots N_{channel},\ j=1\cdots N_l^{bank} \tag{5}$$

$$B_l^j.S_i^k \geq \max\{B_l^j.F_i^{k-1}, a_i^k\} \tag{6}$$

$$B_l^j.F_i^k \geq B_l^j.S_i^k + B_l^j.L_i^k \tag{7}$$

$$C_l.S_i^k \geq \max\{C_l.S_i^{k-1}, B_l^j.F_i^k\} \tag{8}$$

$$C_l.F_i^k \geq C_l.S_i^k + C_l.L_i^k \tag{9}$$

$$\text{Other memory timing constrains} \tag{10}$$

3.4 模型验证及其说明

首先对上述模型的合理性进行解释说明，F 和 T 分别概括了多核内存调度算法的公平性指标和吞吐率指标. 在此处，吞吐率使用了在一个周期内完成的有效的访问时间数和总的请求访问时间总数之比. 为了使上述目标加权平均值最大化，调度算法一方面需要尽量提高内存请求的访问速率，保证吞吐率；另一方面控制来自不同处理器的访问请求的等待时间应当尽量均匀，从而保证公平性.

模型对内存访问的细微时序没有直接给出约束的表达式，这是因为内部实现对于不同的物理结构可能不一致，实际中主要包括访问的 Bank 的行是否命中时需要的等待时间、行刷新时间、行充电、激活时间等，不同硬件存在不同约束，因而没有直接给出明确的约束条件，另外在等待时间上被细分成了就绪前等待时间、队列中等待进入 Bank 队列的时间、Bank 中等待时间、Channel 中等待时间等，因此方程(6)、(9)分别给出了 Bank、Channel 对请求的服务开始和结束时间约束.

3.4.1 参数设置

3.4.1.1 模型加权系数

从前面的模型中可以看出，模型的目标函数中包括公平性和吞吐率两个指标，在公平性和吞吐率两个指标的计算上可以看出两者模式基本一致，并且已经包括了归一化过程. 现对模型的加权系数 λ_1 和 λ_2 增加一致性约束条件，即

$$\lambda_1 + \lambda_2 = 1 \tag{11}$$

对于不同的应用场景，用户可以通过配置不同的 λ_1、λ_2 来表明对不同策略的侧重性. 当 $\lambda_1=1$ 时，表示系统唯独关心各个请求的公平性，即模型的结果会尽量使各个请求的等待时间相对访问时间均等；同理，当 $\lambda_2=1$ 时，系统仅关心系统的吞吐率，此时系统会尽量减少将被执行的请求的等待时间，如优先发长度较长或访问频率较高的请求等，而实际中，如果对此没有过多的侧重，可以将 λ_1、λ_2 设置相当即可，否则按照具体需要实际配置即可.

3.4.1.2　模型度量周期

在模型中，细化思考即可知道，调度可能发生在任意时刻，而上述模型的计算需要一定的参考周期，即考虑一个多大范围内的内存访问请求的公平性和系统吞吐率. 总体上说，调度周期主要为：调度的实际队列、一个测试周期 T、从开机至调度时刻，其中最后一种方式充分利用了调度的历史消息，同时被评价的请求可以限制是当前尚未运行结束的线程发出的访问请求，从而防止因为过多不相关的历史流量带来的持续影响，本文使用该策略.

3.4.2　模型验证

调研发现，在目前的 FCFS、FQ 系列算法中，均以吞吐率和公平性为指标[18]，其中吞吐率在优化中更为常见，本文中的模型通过最小化评测周期内各请求中最大的等待时间所占比例，来保证各个请求的等待时间尽量均等. 同时，通过计算各个线程完成的传送量和需求量之比来表示评测周期内系统的吞吐率，通过将其最大化，有效地提高系统的吞吐率. 因此，模型的优化目标概括了调度算法的主要性能指标.

该模型有效地将计算机内存调度问题抽象为一个数学建模问题，从而将对调度算法的研究转化为求解模型，此时用于求解模型的一些方法均可以借鉴到该数学模型中，其中包括求解复杂模型的启发式算法. 也可以通过计算机模拟的方法求解本模型，当然具体的构造方法依赖于具体问题. 下一节中，将对模型进行求解.

3.5　模型改进

在此模型中，对预取没有详细的说明和约束. 实际中，合理地使用预取甚至分段技术[19-20]能够极大提高多核性能，同时也可以有效地减少访问的延迟. 另外，硬件的 Cache 架构、容量也会对内存系统性能产生较大影响[21]. 在本文的研究中，我们在同等条件下进行比较，忽略了硬件带来的影响.

在本模型中，主要是在请求和线程的角度进行优化. 实际上，在必要的场合，我们更期待 CPU 的利用率更高，因此从 CPU 的角度进行量化和评价将产生更好的结果. 调度的复杂性也在一定程度上得到减弱. 另外，当前的模型没有考虑优先级和实时性等问题，因此改进空间仍旧非常大.

4　模型求解与验证

4.1　算法提出

从第 3 节中介绍的模型可以知道，为了获得模型的较优解，在减少每个请求的访问时间的同时需要关注各个请求、CPU、线程的平均延迟情况，这是因为请求的延迟相差较大将影响调度的公平性.

4.2　内存控制器

在对内存控制器结构进行阐述之前，首先简要描述一下内存控制器的作用，从前文描述可以知道，内存控制器主要完成的是维护内存请求队列，同时从队列中选择合适的请求交由下层硬件执行，因此控制器的一个重要功能即为调度和决策. 为了完成该项任务，内存控制器需要一定的存储空间来缓存上层的内存访问请求.

另外，在前文介绍的内存体系结构中，已经说明了内存的 Row、Bank、Rank、Channel 内部层次关系，因此内存控制器同样需要完成 Row、Bank、Rank、Channel 级别的调度控制，为了设计简单，在各个调度器中使用的算法均一致.

从图 1 可以看出，相比文献[13]的内核控制器结构，该内存控制器中存在“内存访问请求缓冲＋综合调度”区，进入该区域的请求已经被贴上了时序、CPU、线程的标签. 在实现时，该区域被循环使用，为了控制方便，使用了链表进行组织. 该区域的大小对系统的性能存在一定的影响，这是因为如果该区域填充满后，CPU 将出现空闲等待. 调度算法的主要工作在于综合调度，然后将要执行的内存访问请求交由 Bank 调度器对应执行，Bank 执行完毕后由 Channel 完成最终的数据传输，在图中除了“Cache 行缓冲区”上的数据总线外，其余总线均为控制总线. 此外，在图 1 中没有明确区分读写请求.

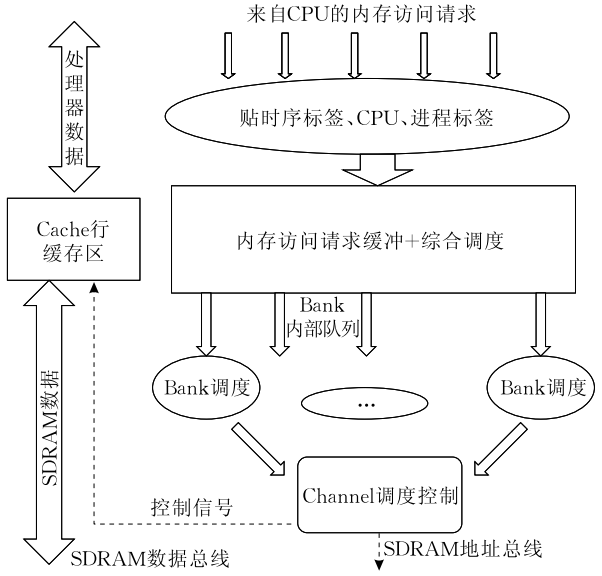

图 1　内存控制器结构

另外,需要注意的是,在内存控制器中也没有详细介绍 SDRAM 数据总线和地址总线,实际上由 Channel 调度控制器管理的地址总线还需注意时序问题.在图 1 中,可以看出,和处理器的数据交互均通过 Cache 的行缓冲区来完成.

4.3 算法描述

根据前面的模型介绍,利用启发式算法,同时参考现有的一些算法,通过综合考虑优先级等因素,利用模型可以构造出如下算法 1.

算法 1. FQ-SJF.

1. 若存在优先级不等的请求,则选择优先级最高的请求;否则转步 2;

2. 在定义公平性指标下,找到最不公平的访问请求,若其指标在容忍区间内则转步 3;否则执行该次访问;

3. 从队列头部找到数据已经在 Cache 的行缓冲区的请求,执行该请求;否则转入步 4;

4. 从队列中选取一个所需执行时间最短请求.

对于调度算法,若行缓冲区较大,则还存在较为关键的一项决策,即替换问题,需要选择一行用于新的内存请求换入.从以前的研究情况来看,LRU 算法在换出方面具有较好的性能,因此在本算法中,选取替换最长时间未被使用的行.

4.4 仿真测试

使用 M5 模拟器对该内存调度系统进行测试,其中开发测试的环境架构主要为:在 Windows XP 上运行 VMware 7.0 虚拟机,然后在 VM 中运行 ubuntu,进而在 ubuntu 中运行 M5 模拟器并采集 Trace.模拟器的参数和真机环境具体介绍如表 3 所示.

表 3 测试环境

属性	CPU 主频	内存/外存	缓存	系统
真机参数	单核 2GHz	2GB/250GB	4MB	XP 5.1.2600
虚拟机 VM 参数	2GHz	512MB/20GB	缺省	VM 7.0
M5 参数	4 核 2GHz	128MB/无限制	缺省	M5.opt 2.0

整体上使用 M5 模拟器,但是实际中,为了保证测试的数据 Trace 始终一致,在初次测试的同时,将请求队列中的 Trace 保存下来,测试其它算法时直接使用 Trace 作为调度对象,然后计算出性能评价所需的关键结果等.其中上层运行的基准应用为 SPEC2000,通过该方案,有效提高了测试的速率,同时保证了各个算法测试的数据的一致性.

下面对单一应用的 Trace 进行简单的分析,并且针对各算法进行一个初步的比较和运算,在本次测试中,采集和测试的 Trace 主要包括如下几个:gcc、mcf、sphinx、astar、bzip2、crafty、gap、gzip、h264ref、soplex、vortex、vpr 等,首先选择 gcc 应用作为研究对象进行简要的 Trace 分析.

首先查看 Trace 的抵达时刻和次序关系,前 300000 个访问请求的抵达时刻分析图如图 2 所示,从图中可以看出,该应用的 Trace 到达时刻较为均匀,并且和次序呈线性关系,由于数据较多,放大前 30000 个 Trace 分析,其次序如图 3 所示,从图中可以看出 Trace 仍旧具有较好的线性时序关系.需要指出的是,该 Trace 没有区分指令和数据 Trace,而是混合在一起进行测试.其中读请求共 268840 个,写请求 31160 个.

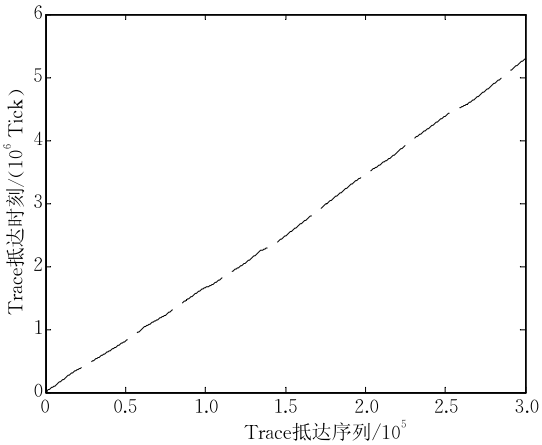

图 2 gcc 的 Trace 抵达时刻分析图

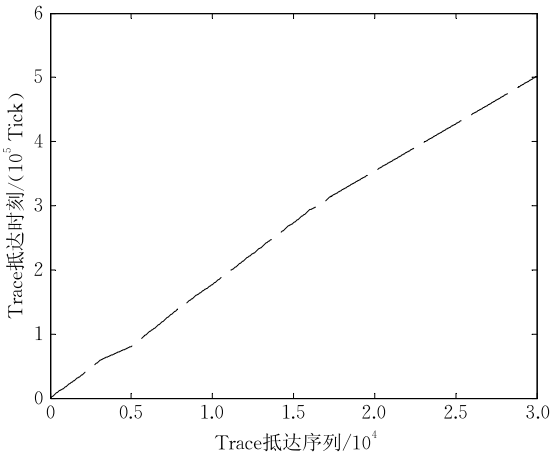

图 3 前 30000 个 Trace 抵达时刻分析图

下面将分别介绍 FR-FCFS 和 FQ-SJF 算法的运行结果,由前面的 Trace 分析结果可以看出,整个过程中内存请求间隔较为均匀,因此下文主要针对前 30000 个访问请求的 Trace 进行调度模拟分析.评测指标选用平均访问延迟,并选取 FR-FCFS 的前 30000 个访问请求的平均访问延迟作为归一化的参考值,针对每相邻的 100 个请求取平均访问延迟,归一化后绘制平均访问延迟图例.

在图 4 中描述了负载较小时，两种调度算法对 gcc 的内存访问请求的处理延迟大体一致，表明算法在轻负载时表现较优. 而随着负载增加，在图 5、图 6 中可以看出较重负载时，FQ-SJF 算法的读取延迟均要比 FR-FCFS 小，性能上提高了接近 10%，并且随着负载增加，提升空间越大. 当然，随着负载的增加，会受到总线等限制，访问延迟也会极大增加.

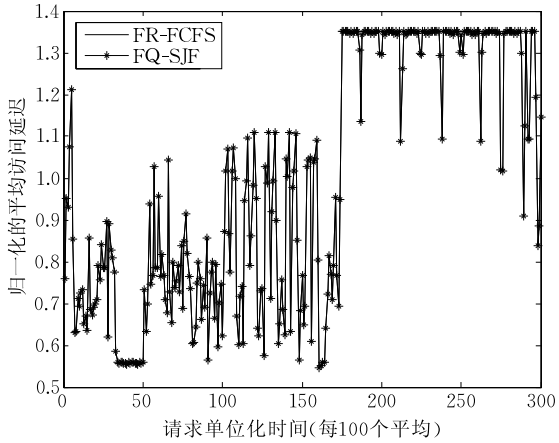

图 4　gcc 基准 FR-FCFS 和 FQ-SJF 平均读取延迟

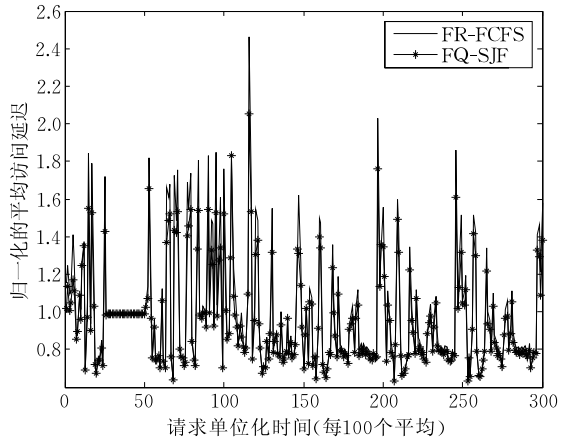

图 5　vortex 基准 FR-FCFS 和 FQ-SJF 平均读取延迟

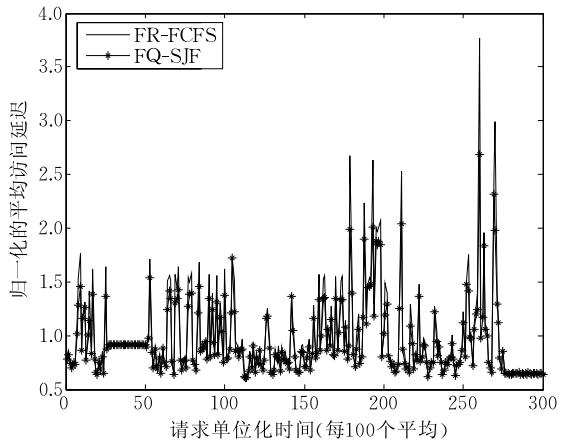

图 6　gap 基准 FR-FCFS 和 FQ-SJF 平均读取延迟

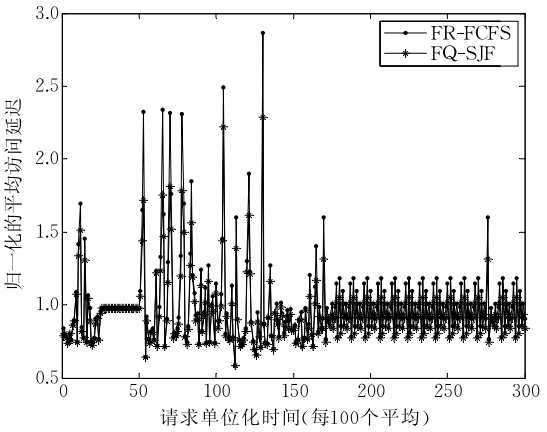

图 7　gzip 和 vpr 混合内存访问平均读取延迟

图 7 展示了在多个测试基准混合运行时的平均读取延迟分析图. 从图中可以看出，测试结果表现较优，性能得到提升.

一些其它基准的测试结果如表 4 所示，FQ-SJF 相对 FR-FCFS 的平均读取延迟同样减小了许多，单个 soplex 基准平均访问延迟减小了 10.6%，在多个基准的同等混合性能测试中，延迟减小了 6.1%.

表 4　基准测试结果

测试基准	加速比	减少延迟/%	测试基准	加速比	减少延迟/%
vortex	1.052	4.9	astar	1.101	9.2
soplex	1.118	10.6	gcc	1.000	0
h264ref	1.102	9.3	bzip2	1.176	14.9
gap	1.064	6.0	gzip,vpr	1.065	6.1

4.5　算法改进

算法上，FQ-SJF 仅为模型的一个普通解，通过进一步求解，将可以获得性能更佳的调度算法，从 Trace 分析来看，到达时刻和次序基本呈现线性关系，另外，在负载较高时，FQ-SJF 性能会降低，同时调度开销相对较大，因此此处可以通过改进 FQ-SJF，使得在必要时将 FQ-SJF 退化为 FR-FCFS 算法，此时内存控制器的调度开销将大为减小.

在内存控制器一节已经提到，对于 Bank 和 Channel 的调度直接和核心调度算法一致. 实际上，在 Bank 级别和 Channel 级别可以做出一些改进，也可以引入对 Cache 行的刷新和替换控制，从而加速特定的基准应用. 另外，部分系统可能对实时性存在要求，此时算法需要综合权衡各大指标，并将优先级高的请求优先调度. 需要特别指出的是，复杂的调度算法常常伴随着增加了内核控制器内部实现该算法的复杂性.

5 结 论

本文提出了一个多核间内存调度模型,该模型将多核调度的实际问题直接转化为一个数学模型中的优化问题,从而在构造算法时可以参照数学模型,求解的方法快速衍生出高效合理的内存调度算法,并对模型进行了初步的求解验证,得到了一个模型解.从测试结果上看,相比传统的 FR-FCFS 算法,系统吞吐率和延迟均有较大改善.

致 谢 本文作者得到了清华大学计算机科学与技术系操作系统实验室的老师和同学们的许多帮助和建议,在此表示感谢!

参 考 文 献

[1] Rixner S, Dally W J, Kapasi U J, et al. Memory access scheduling//Proceedings of the 27th International Symposium on Computer Architecture. Vancouver, Canada, 2000: 128-138

[2] Cai Qiong, González J, Rakvic R, et al. Meeting points: Using thread criticality to adapt multicore hardware to parallel regions//Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT 08). Toronto, Canada, 2008: 240-249

[3] Bai Zhong-Ying. The Principle of Computer Organization. 4th Edition. Beijing: Science Press, 2008(in Chinese)
(白中英. 计算机组成原理. 第 4 版. 北京: 科学出版社, 2008)

[4] Nesbit K J, Dhodapkar A S, Smith J E. AC/DC: An adaptive data cache prefetcher//Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques. Antibes, Juan-les-Pins, France, 2004: 135-145

[5] Iyer R, Zhao Li, Guo Fei, et al. QoS policies and architecture for cache/memory in CMP platforms//Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. San Diego, Canada, 2007: 25-36

[6] Chang Ji-Chuan, Sohi G S. Cooperative caching for chip multiprocessors//Proceedings of the 33rd International Symposium on Computer Architecture (ISCA'06). Boston, USA, 2006: 264-276

[7] Hur I, Lin C. Adaptive history-based memory schedulers//Proceedings of the 37th International Symposium on Microarchitecture. Portland, USA, 2004: 343-354

[8] Suleman M, Mutlu O, Joao J A, Patt Y N. Data marshaling for multi-core architectures//Proceedings of the Annual International Symposium on Computer Architecture (ISCA 10). Saint-Malo, France, 2010: 441-450

[9] Lee C J, Mutlu O, Narasiman V, et al. Prefetch-aware DRAM controllers//Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture. Como, Italy, 2008: 200-209

[10] Cuppu V, Jacob B, Davis B, Mudge T. A performance comparison of contemporary DRAM architectures//Proceedings of the International Symposium on Computer Architecture. Atlanta, USA, 1999: 222-233

[11] Acacio M E, Gonzalez J, Garcia J M, Duato J. An architecture for high-performance scalable shared-memory multiprocessors exploiting on-chip integration. IEEE Transactions on Parallel and Distributed Systems, 2004, 15(8): 755-768

[12] Liu Meng-Xiao, Ji Wei-Xing, Wang Zuo, et al. High performance memory management for a multi-core architecture//Proceedings of the 9th IEEE International Conference on Computer and Information Technology (CIT 2009). Xiamen, China, 2009: 63-68

[13] Nesbit K J, Aggarwal N, Laudon J, Smith J E. Fair queuing memory systems//Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 06). Orlando, USA, 2006: 208-222

[14] Burger D, Goodman J R, Kägi A. Memory bandwidth limitations of future microprocessors//Proceedings of the 23rd Annual International Symposium on Computer Architecture. Philadephia, USA, 1996: 78-89

[15] Qureshi M K, Lynch D N, Mutlu O, et al. A case for MLP-aware cache replacement//Proceedings of the 33rd International Symposium on Computer Architecture. Boston, USA, 2006: 167-178

[16] Ebrahimi E, Lee C J, Mutlu O, et al. Fairness via source throttling: A configurable and high-performance fairness substrate for multi-core memory systems//Proceedings of the ACM SIGARCH Computer Architecture News. Pittsburgh, USA, 2010: 335-346

[17] Mutlu O, Moscibroda T. Parallelism-aware batch scheduling enhancing both performance and fairness of shared DRAM systems//Proceedings of the 35th International Symposium on Computer Architecture (ISCA 08). Beijing, China, 2008: 63-74

[18] Rafique N, Lim Won-Taek, Thottethodi M. Effective management of DRAM bandwidth in multicore processors//Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007). Brasov, Romania, 2007: 245-258

[19] Bhadauria M, McKee S A. Optimizing thread throughput for multithreaded workloads on memory constrained CMPs//Proceedings of the 5th Conference on Computing Frontiers (CF 08). Ischia, Italy, 2008: 119-128

[20] Jouppi N P. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers//Proceedings of the 17th Annual International Symposium on Computer Architecture. Seattle, USA, 1990: 364-373

[21] Bhattacharjee A, Martonosi M. Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors//Proceedings of the Annual International Symposium on Computer Architecture (ISCA 09). Austin, USA, 2009: 290-301

LIU Hu-Qiu, born in 1989, Ph. D. candidate. His main research interests include multi-core memory schedule algorithm and architectures.

ZHAO Peng, born in 1985, Ph. D. candidate. His current research interests include multi-core architecture, memory schedule and operation system.

Background

Computer memory is an important device to store information, which is the key resource of the computer system, effective and reasonable managing of the resources could effectively improve the computer's performance. In this paper, switching the multi-core memory scheduling problem into a mathematical model, by the model, we could get a better performance algorithm.

In recent years, multi-core architecture has been the mainstream of computer, multiple processors still share system memory. The importance of the memory scheduling has become increasingly evident, and it has become a research hotspot. Researchers also hope to design better processor and memory architecture to improve the performance.

At present, a wide range of scheduling algorithms have been proposed in multi-core memory, but most of the starting point is consistent. As summarized in the paper, the current research focuses on four-class multi-core memory scheduling algorithms, which includes the general-purpose multi-core scheduling algorithm, the multicore scheduling algorithm based the FCFS, the scheduling algorithm which based on replacement strategy, and the FQ scheduling algorithm.

The generic algorithms, including the LRU classical algorithm, are also appeared in cache replacement management strategy. Due to the application of environmental change, in the multi-core environment, FCFS algorithm cannot be directly used, so there are a large number of improved algorithms. With better performance, and simultaneously it solves the problem when directly transplanted it.

Most algorithms take account of the throughput of the system and part of them have also consider fairness which is similar to network QoS(quality of service). So this paper tries to present a general scheduling model. And through theoretical argument, we want to verify that a variety of algorithms are the model's solution and try to get some solution of the model to solve the problem of multi-core memory scheduling.

Then, by using the heuristic algorithm construction method, the model constructs a solution of the multi-core scheduling algorithm. We have evaluated the proposed algorithm's system performance and validated model's integrity.