DOI：10.7652/xjtuxb201412014

ARM GPU 的多任务调度设计与实现

丑文龙，梅魁志，高增辉，李博良
（西安交通大学电子与信息工程学院，710049，西安）

摘要：针对现有 GPU 任务调度系统在多任务环境下不能保证图形任务响应时间的问题，提出基于分类和多优先级队列(CPMQ)的调度方案，并在 ARM 的嵌入式 GPU 上实现验证。该方案中，将 GPU 的多任务划分为图形任务、通用计算任务和实时图形 3 类任务并分别建立队列排队，其中图形任务和通用计算任务按照优先级在各自队列中排队，实时图形按照任务截止时间排队。面向多队列的任务调度，优先从实时任务队列中选择任务，并按照加权公平算法分别在图形任务队列和通用计算队列中选择任务。实验结果表明：相比于 ARM GPU 的原有调度系统，CPMQ 在不显著增加通用计算任务的执行时间和调度开销的情况下，将实时图形任务的帧率提升了 5%～20%。

关键词：图形处理器；多任务；调度；排队
中图分类号：TP316 **文献标志码**：A **文章编号**：0253-987X(2014)12-0087-06

Design and Implementation of Multitask Scheduling for Embedded ARM GPU

CHOU Wenlong, MEI Kuizhi, GAO Zenghui, LI Boliang
(School of Electronics and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China)

Abstract: A scheduling solution of class priority multiple queue (CPMQ) is proposed to solve the problem that the response time to graphic tasks is not ensured by existing task scheduling systems of GPU under multitask conditions, and the schedule is implemented on an embedded system. Multiple tasks on GPU are firstly classified into three classes of tasks, that is, graphic tasks, real-time graphic tasks and general purpose computing tasks. These three classes of tasks then queued respectively with different queuing policy. Graphic tasks and general purpose computing tasks are queued by their priorities, while real-time graphic tasks are queued by their deadlines. When the multi-class tasks are scheduled, real-time graphic tasks are selected at first, and then graphic tasks and general purpose computing tasks are selected out using a weighted fair queuing algorithm. Experimental results and comparisons with the original scheduling system of ARM's GPU show that CPMQ increases the frame rate of reel time graphic tasks by 5%-20% without significant increase in execution time of general purpose computing tasks and scheduling expense.

Keywords: graphic processing unit(GPU); multitask; scheduling; queuing

对流畅的图形界面和强大的计算能力的需求，使得当前的嵌入式 GPU 应用日趋广泛。尽管嵌入

收稿日期：2014-06-23。 作者简介：丑文龙(1989—)，男，硕士生；梅魁志(通信作者)，男，副教授。 基金项目：国家高技术研究发展计划资助项目(2012AA010904)；国家自然科学基金资助项目(61375023)。
网络出版时间：2014-10-31 网络出版地址：http://www.cnki.net/kcms/detail/61.1069.T.20141031.1642.016.html

式 GPU 上同时运行着多个图形任务和通用计算任务，但是图形任务直接面向最终用户，决定了嵌入式系统的用户体验，是 GPU 上的关键任务。如何在多个任务竞争 GPU 资源的情况下保证图形任务的响应速度，成为 GPU 多任务调度的关键以及新的研究方向与热点，例如 Onur Mutlu 等的研究[1]。

面对多任务环境下图形处理，Linux 操作系统的直接渲染设施[2](direct rendering infrastructure，DRI)，采用了先到先服务(first come first served，FCFS)的简单调度策略。Kato 等提出了一个 GPU 调度框架 TimeGraph[3]，在 DRI 的基础上实现了资源预留和调度算法，提升了任务的实时性，其不足在于只支持使用 GLSL[4]实现的通用计算任务。Bautin 等提出了 GPU 资源管理框架 GERM，主要致力于公平地为各个应用分配 GPU 的计算、存储资源[5]，其采用优先级和 GPU 命令执行时间预测的方法，提升了调度的公平性，不足在于需要修改用户空间与 GPU 相关的函数库，例如 OpenGL 库。微软在 Vista 及其之后的操作系统中开发了 GPU 驱动模型 WDDM[6]，实现了 GPU 多任务调度，但是并没有开放其设计原理和实现方式。实验结果表明，在高负载情况下图形任务的响应时间会显著增加。ARM 在其 Mali T6 系列嵌入式 GPU 的 Linux 驱动程序中实现了完全公平调度算法[7](completely fair scheduling，CFS)，该算法公平地对待每一个任务以公平地共享 GPU，使用任务的优先级和任务运行时间作为调度的依据，在任务的优先级相同时，下一次优先调度使用 GPU 时间少的任务，不足在于没有考虑到不同类型的 GPU 任务对用户的重要性不同，无法实现区别对待。

本文提出一种能保证图形任务性能的基于分类和多优先级队列(class priority multiple queue，CPMQ)的 GPU 多任务调度系统。在多任务环境下，CPMQ 调度系统在保证其他任务正确执行的前提下，有效降低了图形任务的响应时间，提升了应用系统的图形用户体验。

1 GPU 多任务调度框架与调度目标

1.1 GPU 多任务调度框架

GPU 任务调度属于操作系统 GPU 驱动程序的一部分，随操作系统平台和 GPU 硬件而变化，但是其核心调度框架是一致的，如图 1 所示。

多个任务通过驱动程序中的多任务调度模块的调度而使用 GPU，该模块有两个主要子模块：排队模块按照一定的排队规则将多个任务排序；分发模块从排队模块中选择下一个要运行的任务，将其发送给 GPU 硬件。GPU 硬件由管理部件和计算核心等组成，前者负责为各个计算核心分配任务，后者负责具体的计算。管理部件接收到驱动程序的多任务调度模块发送来的任务后，会根据各个计算核心的运行情况，将任务分配给具体的计算核心，任务执行完后，会通过中断通知驱动程序，多任务调度模块会进行下一次任务调度。

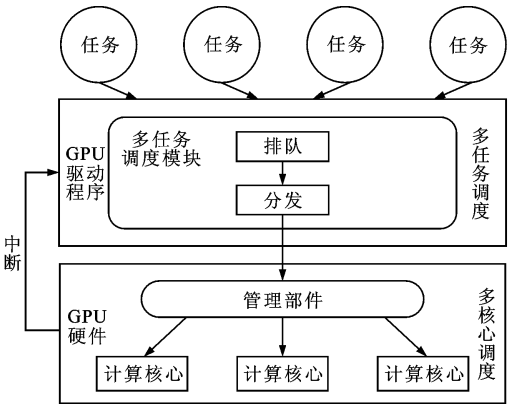

图 1 GPU 多任务调度框架

由此可见，GPU 的调度分为多任务调度和多核心调度两个阶段，分别由 GPU 驱动程序和 GPU 硬件完成。

1.2 GPU 多任务调度目标

面向多任务环境的调度方案，总是在快速响应与公平分配资源之间权衡。在智能手机、智能电视等平台，图形任务是关键应用，能否被快速响应与处理是评价用户体验的重要因素。因此，本文的 GPU 多任务调度的目标为在不显著增加其他任务执行时间的情况下，降低图形任务的响应时间，即提升图形任务的帧率。

记 T 为任务的执行时间，即响应时间

$$T = T_{\text{exec}} + T_{\text{overhead}} \tag{1}$$

式中：T_{exec} 为任务的真正执行时间；T_{overhead} 为调度开销，包括排队时间、任务切换时间等。

对于 GPU 而言，任务在计算核心之间的分配由 GPU 内部的硬件完成，负责充分利用计算核心、核心间的负载均衡等工作，处于软件层的 GPU 多任务调度不能直接控制计算核心的利用率，所以对于同一个任务，T_{exec} 是不变的。要降低图形任务响应时间，就要降低其调度开销 T_{overhead}，对于同一个调度系统而言，一次调度的调度开销 T_{sched} 为系统中所有任务调度开销之和

$$T_{\text{sched}} = \sum_{p=0}^{n} T_{p\_\text{overhead}} \quad (2)$$

式中：T_{p_overhead} 为任务 p 的调度开销。要降低图形任务的调度开销就需要增加其他任务的开销，使得其他任务的执行时间增加。

2　CPMQ 多任务调度设计与实现

本文提出的 CPMQ 结构如图 2 所示，由分类器、排队器、分发器及统计反馈器 4 部分组成。

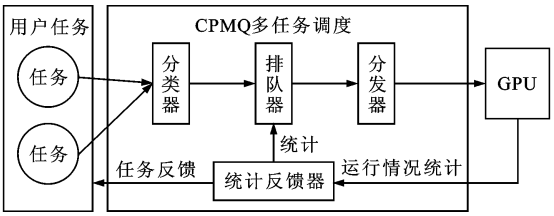

图 2　CPMQ 多任务调度系统整体结构图

2.1　分类器

分类器用于将任务分类，是 CPMQ 排队系统与用户的接口，分类信息是 CPMQ 调度系统运行的基础。为了保证图形任务的响应时间，CPMQ 多任务调度系统将任务分类，以实现更加细粒度的任务调度。虽然 GPU 上运行的任务数量多、类型广，但作为主要面向图形计算的处理器，其运行的任务可分为以下几类。

(1)图形任务，以下简称 G 任务。G 任务是用户能直接感觉到的纯图形计算类型的任务，需要较快的响应速度以保证用户界面的流畅性，包括一般的应用程序等。

(2)通用计算任务，以下简称 C 任务。C 任务是使用 GPU 的计算能力进行非图形计算的任务，由于计算结果不用于显示，所以相对于图形计算任务而言，通用计算任务对响应时间的要求不严格，只要保证计算结果正确和计算时间合理，例如矩阵运算、大数据处理等。

(3)实时图形任务，以下简称 RT 任务。这类任务是特殊的图形任务，与一般图形任务相比，RT 任务需要更加快速的响应速度和更确定的响应时间，包括 3D 游戏等。

分类器由配置接口和分类配置库两个模块组成，如图 3 所示。在用户提交任务之前，需要向 CPMQ 调度系统提交任务的类型信息，该信息将通过配置接口，保存在分类配置库中。之后，当提交任务的时候，分类器将从分类配置库查找任务的分类信息并输出，作为排队器的排队依据之一。

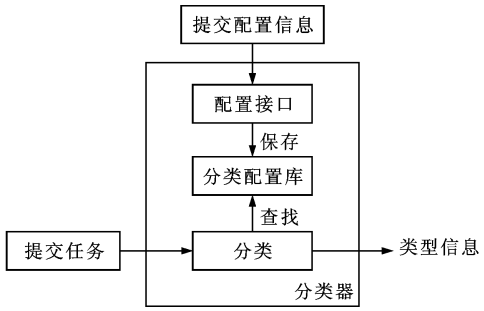

图 3　分类器

2.2　排队器

排队器按照一定的排队规则维护所有任务的有序性，是 CPMQ 排队系统的重点。

GPU 任务以 GPU 命令的形式使用 GPU，一个任务对应多个 GPU 命令。按照 GPU 命令的有无，将任务分为就绪态和可运行态两个状态。当任务中的所有 GPU 命令执行完后，该任务处于就绪状态；当任务中有 GPU 命令时，任务就处于可运行状态。状态之间转换如图 4 所示。

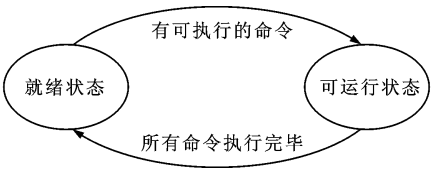

图 4　任务状态转换图

CPMQ 中对应就绪态队列和可运行态队列两组队列。队列分为实时组和非实时组两组。实时组中的队列为实时图形队列，简称 RT 队列；非实时组中有两个队列，分别为图形任务队列(简称 G 队列)和通用计算任务队列(简称 C 队列)，如图 5 所示。

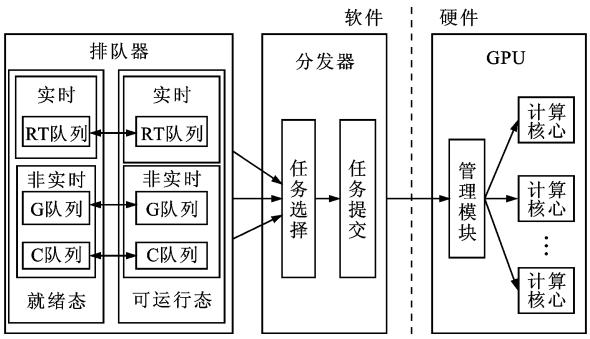

图 5　排队器与分发器结构

常用的排队算法有公平排队算法(fair queuing，FQ)、加权公平排队算法(weighted fair queuing，WFQ)[8]、优先级排队算法(priority queuing，PQ)[9]和最早截止时间优先算法(earliest deadline first，EDF)[10]。FQ 算法侧重于任务调度的公平

性,忽略了任务实时性需求;WFQ 在 FQ 的基础上加入了优先级的考虑,但不能保证任务的实时性;PQ 根据任务的优先级排队,能优先调度重要的任务;EDF 加入了对于任务截止时间的考虑,适用于实时任务的调度。

因此,排队器中的实时图形任务按照 EDF 算法排队,即按照任务的截止时间从小到大排列,截止时间最小的任务在队首。当一个任务加入实时图形队列时,需要根据其截止时间信息将该任务插入到正确的位置。

对于图形任务和通用计算任务,优先级决定了其在队列中的位置,进而决定了任务使用 GPU 的时机。优先级越高,其在各自队列中就能越早地得到调度,获得 GPU 资源。图形队列和通用计算队列内部按照 PQ 算法在各自队列中排队,即按照任务的优先级从高到低排列。图形任务和通用计算任务的 GPU 优先级 G_i 的计算方法相同

$$G_i = \left(P_i / \sum_{j=1}^{n} P_j\right) / \left(T_i / \sum_{j=1}^{n} T_j\right) \tag{3}$$

GPU 优先级是在任务的 CPU 优先级的基础上,结合运行时间计算得到的。假设队列中有 n 个任务,对于任务 i,CPMQ 会记录其使用 GPU 的累计时间 T_i 及其 CPU 优先级 P_i。Linux 系统中,任务的 CPU 优先级通过 nice 值表示,取值范围为 $-20\sim19$,值越小表示任务的优先级越高;为了计算方便,令 $P_i=20-\text{nice}$,其取值范围为 $1\sim40$,值越大任务优先级越高。相同 CPU 优先级下,任务累计执行时间占所有任务总执行时间的比例越小,GPU 优先级就越高,表示该任务没有得到与其 CPU 优先级相称的 GPU 时间,下次调度优先考虑该任务;相同累计执行时间占比的情况下,CPU 优先级越高,GPU 优先级就越高,下次调度优先考虑该任务。

2.3 分发器

分发器用于从排队器的可运行态队列中获取下一个要运行的任务,并将该任务发给 GPU 硬件执行,是 CPMQ 调度系统与硬件的接口。

分发器分为任务选择和任务提交两个模块。任务选择模块用于从排队器的可运行态队列中选出一个可运行任务;任务提交模块用于将任务选择模块选出的任务提交给 GPU 硬件执行,任务将被提交给 GPU 的管理模块,之后管理模块将该任务分配给计算核心执行。

任务选择模块按照 PQ 排队算法在实时组和非实时组之间选择任务,实时组的优先级总是高于非实时组的优先级,直到 RT 队列为空,分发器才在非实时组中选择任务。非实时组中,分发器采用 WFQ 算法从 G 队列和 C 队列中选择优先级高的队列并从中选择任务,两个队列的优先级均使用该队列中任务的平均优先级

$$Q = \left(\sum_{i=1}^{n} P_i\right)/n \tag{4}$$

式中:P_i 与式(3)中的相同。Q 越大,该队列的优先级越高。Q 的取值范围为 $1\sim40$。

分发器优先调度 G 队列中的任务,其中的任务被调度一次,Q 值就减 1,直到 Q 为 0,或者小于通用队列的 Q 值,或者队列为空。然后再调度 C 队列中的任务,对 C 队列也采用同样的调度方法。

2.4 统计反馈器设计

统计反馈器负责监测任务的执行状态,统计任务的执行结果,其由两个模块组成:统计模块和反馈模块,如图 6 所示。

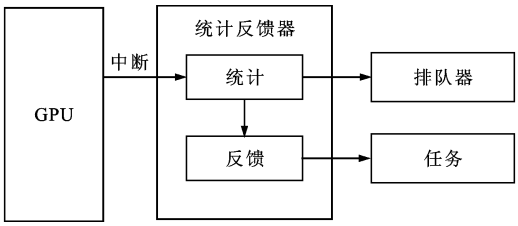

图 6 任务的执行统计与反馈交互

统计模块统计任务的执行时间等参数,将更新后的执行时间代入式(3)并重新为任务排序;反馈模块根据任务的行为(例如命令数量和命令执行时间)识别并处理问题程序。对于在排队的 RT 型任务,更新其 deadline 值,即减去自上次调度以来的毫秒数值,并重新排队;对于非 RT 任务,根据优先级计算公式重新计算优先级,然后重新排队。

反馈模块会检测一个任务是否是问题程序,判断标准为:如果一个任务不断产生大量的 GPU 命令,同时,这些命令执行时间很短,那么就认为该任务是问题程序。CPMQ 系统会给该类型任务两次机会:当该任务是第 1 次被认定是问题程序时,降低该任务所在的进程的优先级,来降低任务发送 GPU 命令的速率,从而降低该任务对 GPU 资源的占用,同时,记录警告次数;若是第 2 次被认定是问题程序,就将该任务的 GPU 优先级置为最低,让其他任务先执行;若第 3 次被认定是问题程序,就直接结束该任务。

3　实验及结果分析

3.1　实验环境与过程

3.1.1　硬件平台　实验使用基于三星 Exynos 5420 芯片的 Arndale Octa 开发板作为平台，操作系统为 Android 4.2。GPU 为 ARM 的 Mali T628，其中有 6 个计算核心。使用 CPMQ 替代 Mali T628 内核驱动中的调度模块。

3.1.2　实验测试程序

(1)实时图形任务。NenaMark2 是用于 GPU 评测的软件，其提供了 3D 渲染测试场景，包含了图形计算中的常用操作，例如光照、粒子系统、纹理、动态阴影等，能比较全面地评价 GPU。NenaMark2 在测试完后会报告渲染过程中的平均帧率(frame per second，FPS)，同时在运行过程中，也会显示实时帧率。

(2)图形任务 GLCube。虽然 Android 系统是多任务的，但是 Android 图形应用程序在运行时会独占屏幕，同时只有一个图形应用程序能在屏幕上显示。为了测试调度系统对多个图形任务的调度情况，本文实现了一个 3D 应用，其在屏幕上绘制 4 个旋转着的立方体，显示每个立方体的 FPS，并且能和其他图形应用程序同时运行。4 个立方体属于 4 个不同的进程，所以 GLCube 代表了 4 个图形类测试应用。同时为了便于获取 FPS 数据，4 个进程会将每秒的 FPS 值保存在 4 个不同的文件中。

(3)通用计算任务。matrix_mul 是使用 OpenCL 实现的矩阵相乘程序，其在 kernel 中计算 1 000×1 000 的两个浮点矩阵的乘法运算。matrix_mul 接收一个数字参数，表示矩阵相乘的次数，次数越大，则计算量越大，例如 matrix_mul 2 表示运行两次高维矩阵相乘。单独运行 matrix_mul 时，每一次矩阵相乘大概需要 5 s。

3.1.3　实验过程　实验中，同时运行 NenaMark2、4 个 GLCube 和 matrix_mul，然后通过调整 matrix_mul 的参数，逐渐增加系统的负载，观测并记录 NenaMark2 和 GLCube 的 FPS 变化情况，以及 matrix_mul 的执行时间变化情况，当 matrix_mul 执行完毕时，结束本次实验并记录数据。在 Mali T628 自带的多任务调度系统和 CPMQ 多任务调度系统上分别运行。

实验之前，通过 CPMQ 系统的分类器配置接口为任务分类，NenaMark2 设置为实时图形任务，超时时间为 5 ms，各个 GLCube 设置为图形任务，matrix_mul 设置为通用计算任务。

3.2　实验数据分析

3.2.1　实时图形任务性能　实时图形任务如图 7 所示。相同条件下，二者的帧率都会下降，CPMQ 对于实时图形任务的调度比原驱动要好，整体帧率都高于原驱动。原驱动中，当通用计算的矩阵相乘次数大于 3 时，图形任务的帧率已经降低到 20 帧/s 以下，能明显感觉到画面的延迟和卡顿；而 CPMQ 调度系统中，当矩阵相乘次数达到 5 时，也基本能保证 24 帧/s 的帧率。相比原驱动，CPMQ 使实时图形任务的整体性能提升 5%～20%。

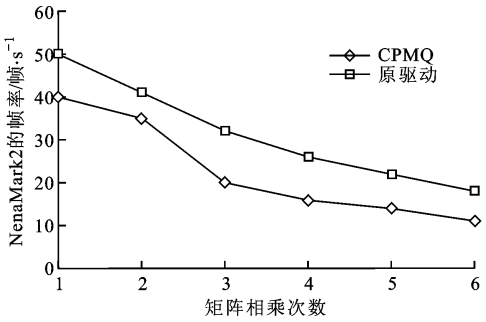

图 7　NenaMark2 的帧率随 matrix_mul 矩阵相乘次数的变化

3.2.2　图形任务性能　图形任务如图 8 所示。由于 GLCube 比 NenaMark2 简单，计算量较小，其帧率均比相同条件下的 NenaMark2 的帧率高。随着矩阵相乘次数的增长，二者的帧率都会下降，CPMQ 对于图形任务的调度比原驱动要好，但是由于有实时图形任务的争抢，图形任务的整体性能提升没有实时图形任务明显，在 1%～15%之间。

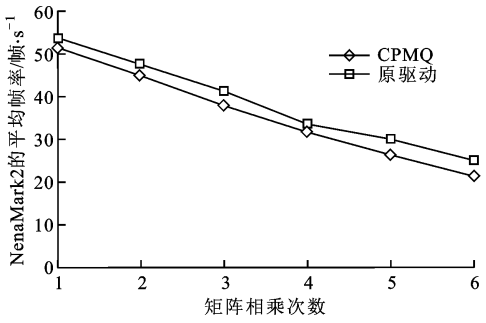

图 8　GLCube 程序的帧率随 matrix_mul 矩阵相乘次数的变化

3.2.3　通用计算任务性能　通用计算任务如图 9 所示。实验结果表明，在相同条件下，kernel 单次循环平均执行时间都处于上升趋势，由于 CPMQ 降低了通用计算任务对 GPU 的占用，导致计算时间加长，性能下降 3%～25%。

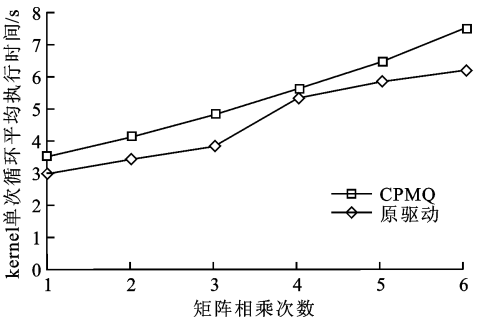

图9 matrix_mul随其自身矩阵相乘次数的变化

4 结 论

本文针对现有GPU多任务调度不能保证图形任务响应时间的问题,在嵌入式GPU平台上设计并实现了基于CPMQ排队算法的GPU多任务调度系统。CPMQ算法是本文在现有排队算法的基础上,结合GPU任务分类而提出的新算法。实验表明,在多任务环境下,该调度方案相比于ARM GPU的原有调度系统,CPMQ在不显著增加通用计算任务的执行时间和调度开销的情况下,将实时图形任务的帧率提升了5%~20%。

参考文献:

[1] JOG A, KAYIRAN O, NACHIAPPAN N C, et al. OWL: cooperative thread array aware scheduling techniques for improving GPGPU performance [C]// Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, NY, USA: ACM, 2013: 395-406.

[2] PAUL B. Introduction to the direct rendering infrastructure [EB/OL]. (2000-08-10) [2014-03-23]. http://dri.sourceforge.net/doc/DRIintro.html.

[3] KATO S, LAKSHMANAN K, RAJKUMAR R, et al. TimeGraph: GPU scheduling for real-time multi-tasking environments [C]// Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference. Berkeley, CA, USA: USENIX Association, 2011: 17-30.

[4] MARROQUIM R, MAXIMO A. Introduction to GPU programming with GLSL [C]// Proceedings of the 2009 Tutorials of the 22nd Brazilian Symposium on Computer Graphics and Image Processing. Washington, DC, USA: IEEE Computer Society, 2009: 3-16.

[5] BAUTIN M, DWARAKINATH A, CHIUEH T. Graphic engine resource management [C]// Proceedings of the International Society for Optics and Photonics. Bellingham, WA, USA: SPIE, 2008: 68180O.

[6] PRONOVOST S. Windows display driver model (WDDM) v2 and beyond [C/OL]// Proceedings of the Windows Hardware Engineering Conference. [2014-03-23]. http://ci.nii.ac.jp/naid/10018383501/.

[7] WONG C S, TAN I, KUMARI R D, et al. Towards achieving fairness in the Linux scheduler [J]. Operating Systems Review, 2008, 42(5): 34-43.

[8] BENNETT J C, ZHANG Hui. WF^2Q: worst-case fair weighted fair queuing [C]// Proceedings of the 15th Annual Joint Conference of the IEEE Computer Societies on Networking the next Generation. Piscataway, NJ, USA: IEEE, 1996: 120-128.

[9] WONG H T. Packet scheduling using dual weight single priority queue: USA, 6570883 [P]. 2003-05-27.

[10] DOYTCHINOV B, LEHOCZKY J, SHREVE S. Real-time queues in heavy traffic with earliest-deadline-first queue discipline [J]. The Annals of Applied Probability, 2001, 11(2): 332-378.

[本刊相关文献链接]

张虹,郑霄,赵丹.GPU加速窦房结计算机仿真的实现及优化.2014,48(7):60-64.[doi:10.7652/xjtuxb201407011]

周秦武,隋芳芳,白平,等.嵌入式无接触视频心率检测方法.2013,47(12):55-60.[doi:10.7652/xjtuxb201312010]

李亮,王恩东,朱正东,等.应用动态生成树的GPU显存数据复用优化.2013,47(10):44-50.[doi:10.7652/xjtuxb201310008]

张保,董小社,白秀秀,等.CPU-GPU系统中基于剖分的全局性能优化方法.2012,46(2):17-23.[doi:10.7652/xjtuxb201202004]

向坤,陈娟,张安学,等.提高喇叭天线增益的超介质构建方法.2011,45(2):92-96.[doi:10.7652/xjtuxb201102019]

邹华,高新波,吕新荣.一种八叉树编码加速的3D纹理体绘制算法.2008,42(12):1490-1494.[doi:10.7652/xjtuxb200812012]

刘晓东,寻亮,马栋,等.基于球体追踪的动态视差遮挡映射算法.2007,41(12):1401-1405.[doi:10.7652/xjtuxb200712004]

赵保华,张炜,林华辉,等.一种通信有限状态机的被动测试及其错误诊断.2007,41(6):640-644.[doi:10.7652/xjtuxb200706003]

(编辑 武红江)