计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2016.20160113
53(7):1467-1477,2016

# XOS:面向用户体验质量的高能效异构多核调度算法

宫晓利[1] 于海洋[2] 孙承君[1] 李 涛[1,3] 张 金[1] 马 捷[2]
[1](南开大学计算机与控制工程学院 天津 300071)
[2](南开大学软件学院 天津 300071)
[3](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)
(gongxiaoli@nankai.edu.cn)

# XOS: A QoE Oriented Energy Efficient Heterogeneous Multi-Processor Schedule Mechanism

Gong Xiaoli[1], Yu Haiyang[2], Sun Chengjun[1], Li Tao[1,3], Zhang Jin[1], and Ma Jie[2]
[1](*College of Computer and Control Engineering*, *Nankai University*, *Tianjin* 300071)
[2](*College of Software*, *Nankai University*, *Tianjin* 300071)
[3](*State Key Laboratory of Computer Architecture* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190)

**Abstract** Smart mobile devices are playing a more and more important part in people's daily life. However, the pursuit of increasing performance of mobile devices directly conflicts with the limited battery capacity. The inevitable contradiction between them begins to block the development of smart mobile devices. To overcome this limitation, the heterogeneous multi-processor architecture can balance the user experience and the energy consumption on smart mobile devices, which makes it become a new solution. Based on a compartmental QoE model, a schedule mechanism called XOS oriented heterogeneous multi-processor devices is presented to provide a high energy efficient solution. In XOS, the user interaction tasks are recognized by the operating system based on the cross-layer information, and more computing resources are allocated to these tasks to guarantee the quality of experience, while resources would be limited for other tasks to reduce energy consumption. A simulation system is built to verify the effectiveness of the XOS model and then make a reasonable optimization. Then the implementation and the experiment of the XOS are conducted on Odroid-XU3 board with Android operation system. The result shows that the tasks scheduled by XOS decelerate lessens 2.7%~7.3% QoE lost, whereas they reduce 8%~48% energy consumption at the same time.

**Key words** smart mobile devices; heterogeneous multi-processor (HMP); quality of experience (QoE); energy consumption optimization; schedule mechanism; cross-layer information

**摘 要** 智能移动设备的重要作用日益凸显,然而,对于性能的追求与有限电池容量的矛盾制约了产业的发展.异构多核处理器架构以其平衡性能与能耗的优势,成为一种新型的解决方案.用户体验优化是

**收稿日期**:2016-03-03;**修回日期**:2016-05-07
**基金项目**:教育部高等学校博士学科点专项科研基金项目(20130031120028);天津市应用基础与前沿技术研究计划青年项目(14JCQNJC00700);天津市自然科学基金项目(16JCYBJC15200);计算机体系结构国家重点实验室开放课题(CARCH201504)
This work was supported by the Specialized Research Fund for the Doctoral Program of Higher Education of China (20130031120028), the Research Plan in Application Foundation and Advanced Technologies in Tianjiin (14JCQNJC00700), the Natural Science Foundation of Tianjin (16JCYBJC15200), and the Open Project of the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences (CARCH201504).

智能移动设备的重要设计目标.借助一个分段式的用户体验模型,提出了面向异构多核设备的 XOS (experience oriented scheduler)调度算法.XOS 能够跨层获取任务信息,识别与用户直接交互的任务组,保证这些任务的计算资源分配以保障用户体验,同时限制非交互性任务的计算资源以降低能耗.通过建立一套仿真系统验证了算法的有效性并进行了调整优化,然后在 Odroid-XU3 开发板 Android 系统中进行了原型实现和验证.实验结果表明:XOS 算法对于不同类型的任务仅产生了 2.7%～7.3%的用户体验下降,但节省了 8%～48%的能量.

**关键词** 智能移动设备;异构多核处理器;体验质量;能耗优化;调度机制;跨层信息

**中图法分类号** TP316

智能移动设备的产业生态正在迅速发展,智能手机、平板电脑等在日常生活中的地位日益增长.各类在线应用商店中移动应用软件数量巨大,吸引着用户投入大量的时间与金钱,从而为设备制造商与软件开发人员提供了广阔的市场空间,进而推动了软硬件产品的更新换代,繁荣的市场吸引了更多的用户.这种相互促进的方式使得智能移动设备相关技术飞速进步.然而,电池容量与能量消耗问题已然成为制约其发展的瓶颈.为此,许多硬件解决方案与软件管理策略不断涌现.

异构多核处理器架构以其平衡性能与能耗的优势,成为该领域的一个研究热点[1-5].在智能移动设备系统中,多种软件任务并存,例如计算型任务(如游戏、图片处理等)和 IO 型任务(如 GPS 导航等)等.不同类型的任务对性能和能耗的需求存在巨大的差异,传统处理器架构难以同时满足.在异构多核架构中,多种不同类型的核心集成在一个芯片中以应对多样化的需求.在本文后续描述中,将高性能高能耗的核心称之为大核,而低性能低能耗的核心则称之为小核.

智能移动设备领域使用的异构多核架构主要有 2 种:1)大核与小核在芯片的资源分配上存在较大差异,小核主要作为辅助控制器使用,从低速外设中收集信息并与大核进行数据交换[6].这种方案虽然具有高能效的优点,但对开发人员的编程技能要求较高.2)ARM 上实现的 big.LITTLE[7] 架构,big.LITTLE 使用相似的频率和相同的指令集以及同步的缓存机制,使得多核心编程开发更为简化,成为产业与科研的热点.IKS(in kernel switcher)与 GTS (global task scheduling)是 big.LITTLE 架构上典型的多核调度算法.IKS 与 GTS 均以任务负载为调度基准,使低负载任务运行于小核上,高负载任务运行于大核上.这种方法使得系统能够充分发挥异构多核架构的优势,达到降低能耗的效果.

智能移动设备作为一种用户交互驱动的设备,用户体验质量(quality of experience, QoE)是一个重要的设计指标,因此需要在能耗约束和用户体验之间寻求平衡.根据对于三星 Exynos 5422 异构多核处理器的测试,大核拥有大约相当于小核 3 倍的处理速度,却需产生接近后者十倍的能耗.小核的性能已经能够满足部分任务的需求,而部分任务在大核上也无法产生相应的用户体验提升却消耗了大量能量.因此,需要针对不同任务的特性设计调度机制以获取最佳性能能耗比.

在传统的异构多核调度算法中,基于任务周期性的历史运行负载信息[8-9]对任务进行大小核之间的迁移.然而在现代智能移动设备中,大多数任务能够在很短的时间内完成,这使得难以进行精确的任务负载追踪.同时,线程池机制的大量使用使得同一线程执行的多段代码之间缺少必然的逻辑关系,这增加了负载追踪的复杂度.

为了解决以上问题,本文提出了 XOS(experience oriented scheduler)异构多核调度算法,基于跨层信息,在保障用户体验质量的同时提高能效.该调度算法实现了 3 个目标:

1) 确保用户体验质量;

2) 改进能量利用效率;

3) 避免给程序开发人员带来巨大的编程负担.

## 1 相关技术介绍

### 1.1 智能移动设备中的任务

智能移动设备多采用现代化的多任务操作系统,可以支持多进程、多线程的运行环境.设备中并发运行着多种不同类型的任务.依据与用户直接交互形式的差异,本文中将任务分为 3 类:

1) 后台任务.后台任务是指处于等待外部设备数据状态的服务任务,包括检测信号强度、电量、

统计存储用量等,或是由用户人工置于后台的任务.用户通常不等待这些任务运行的直接结果,因而用户体验很少受到此类任务的影响.

2) 强交互任务.强交互任务与用户的主观体验直接相关,如 3D 游戏、网页渲染等.用户通常直接等待此类任务产生的反馈结果,任务的响应时间会对用户体验质量产生明显影响.因此,需向此类任务分配更多的计算资源以加速其执行.

3) 轻交互任务.此类任务交互频率较低、间隔长,因而有足够的时间完成计算任务;或者是任务本身资源占用比较少,能够在很短时间内完成,例如文本输入、图标选择等.对于轻交互任务,存在某个特定的任务执行时间阈值,只要任务能够在这个时间内完成,用户体验将不会受到执行时间的影响.例如,如果文字输入软件的处理速度与用户的反应速度相当,软件响应快于用户输入速度并不会带来更好的用户体验.

显而易见,为了降低能耗,后台任务应该被分配到小核执行;同时,强交互性任务应该分配到大核执行以提高用户体验质量.本文聚焦于轻交互任务的调度,这些任务如果调度合理,就能够在不影响用户体验质量的前提下降低能耗.轻交互任务又可以细分为长任务和短任务.短任务是指能够在几百毫秒内完成的任务,例如文字输入、按钮点击等;长任务则是需要几秒甚至更长的时间才能完成的任务.在后文中会分别对这 2 类任务进行处理.

### 1.2 IKS 与 GTS 调度算法

IKS 是 big.LITTLE 上的第 1 代调度算法,其本质为利用内核中 DVFS(动态电压频率调整)来实现计算资源的管理.在该算法中,一个大核与一个小核被分为一组.同一组中,只能有一个核心处于开启状态,调度器依据 DVFS 中对频率的设定控制核心的工作状态和工作频率.IKS 可以保持对 Linux 内核中原有同构多核调度算法的兼容,但是由于同一组中 2 个核心不能同时开启,所以 IKS 并不能充分利用异构多核架构的计算资源.

GTS 是针对 big.LITTLE 架构而专门设计的多核调度算法[10].GTS 算法根据任务负载进行任务调度.如图 1 所示,CPU 中所有的大核为一簇,所有的小核为一簇.任务首先被分配至大核簇,如果此后任务的负载低于大核簇的迁移临界值,则任务会被迁移至小核簇;同理,如果任务运行于小核簇,而其负载高于小核簇的迁移临界值,则任务会迁移至大核簇.如果大核簇上没有任务运行,那么大核簇将完全关闭以降低能耗.但是由于小核簇至少包含一个 CPU 用于运行操作系统,所以小核簇会一直处于开启状态.

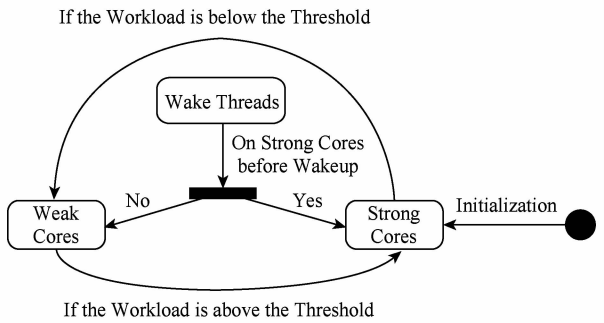


Fig. 1 The GTS model.

图 1 GTS 模型

相比于 IKS,GTS 能够同时开启所有的核心,充分利用异构多核 CPU 的计算资源.然而,GTS 仅以任务负载作为任务迁移的依据,并没有针对移动任务的特征和用户体验的需求进行优化.

### 1.3 QoE 用户体验质量

用户体验质量 QoE 是指用户对于服务体验的主观感受,用于描述业务应用的舒适程度.QoE 是一个复杂的主观指标,对其进行量化描述十分困难.有的研究者试图利用一些客观因素来量化 QoE,例如执行时间、资源消耗等.但是这些因素并不能直接刻画用户体验质量.另一种常用的量化方法是基于用户主观感知的打分法,例如国际电信联盟(International Telecommunication Union, ITU)提出的一种在广播、游戏与电视换台过程中用户体验质量与响应时间的关系模型[11].该模型使用平均主观意见分(mean opinion score, *MOS*)进行度量.*MOS* 通常用五分制来表示,数值从 1(最差)到 5(最好)表示不同的用户体验.其研究结果表明,用户体验质量的 *MOS* 是一个复杂的函数,电视换台体验的 *MOS* 与响应时间的关系[11]:

$$MOS=\max(\min(-1.02\ln(Time),5),1). \quad (1)$$

为了简单起见,本文将用户体验质量分为 3 段:高用户体验部分(good QoE)、低用户体验部分(bad QoE)以及介于两者之间的连续变化部分,如图 2 所示.根据 ITU 研究结果,如果响应时间过短或过长,那么响应时间的变化对用户体验质量不会产生显著影响.换言之,处于高用户体验区或低用户体验区的部分任务,可以在不影响其用户体验质量的前提下降低其性能,以达到降低系统能耗的效果.这是 XOS 调度算法建立的基础.

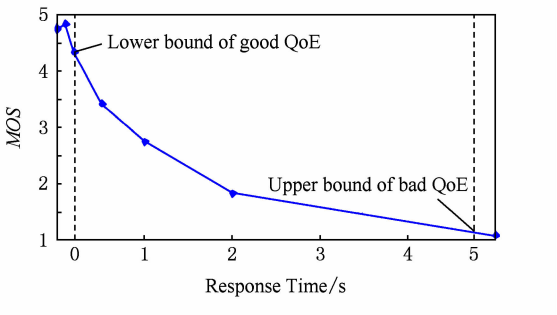


Fig. 2 *MOS* for response time.

图 2 响应时间与 *MOS* 折线图

## 2 XOS 算法设计

### 2.1 调度原则

XOS 调度算法将兼顾能耗与用户体验质量. 为此,算法的设计满足 6 个原则:

① 出于降低能耗的考虑,任务初始化于小核;

② 算法能够识别轻交互任务,当任务在小核上执行时间过长时,为避免 QoE 出现损失,提前将任务迁移至大核;

③ 如果任务运行于大核而用户体验质量仍未能得到改进,那么此时应该以降低能耗为首要目标,将其迁至小核;

④ 交互事件完成之后,任务应该立即回到小核上以减少能耗;

⑤ 传统的同构多核调度算法中对于同类核心的管理代码应该加以利用;

⑥ 不应该给应用开发人员带来额外的编程负担,避免开发复杂化.

### 2.2 调度模型

根据调度算法的设计原则,本文提出了面向 QoE 的 XOS 调度算法模型. XOS 通过获取交互事件开始及结束的信息识别交互性任务及其运行状态的变化,并根据大小核的差异决定任务是否需要在大小核之间迁移. 在本文中,一次交互指的是一次用户的操作,能够触发一次有意义的应用功能,例如从图库中选择一张图片或发送一个信息等.

图 3 展示了一个交互任务的调度模型:

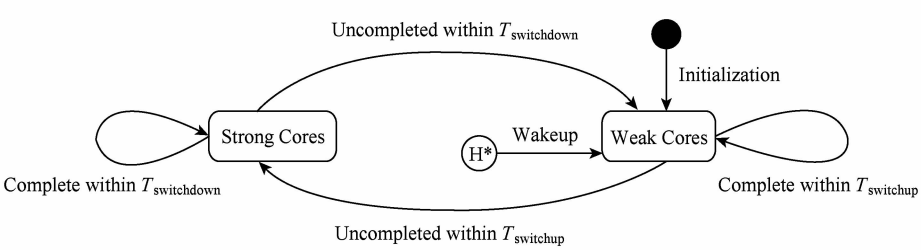


Fig. 3 The XOS scheduling model.

图 3 XOS 算法模型

一个任务接受了用户的交互操作,或与交互任务建立依赖关系后开始进入交互状态. 根据调度模型,一个交互任务的完整调度流程分为 4 个步骤:

Step1. 任务在小核上创建并初始化. 考虑到小核性能已经足够满足许多任务的需求,因而出于降低能耗的目的,每个任务均在小核上完成初始化工作.

Step2. 任务运行于小核. 如果交互任务能够在短时间内完成($t<T_{switchup}$),这意味着小核的性能足够,那么任务将一直运行于小核. 虽然将这些任务迁移到大核上能够加速处理,但是由于任务在小核上的完成速度已经能够满足需求,此时再加速这些任务对提高用户体验并无益处.

Step3. 任务迁移至大核运行. $T_{switchup}$ 被设置为任务从小核迁移至大核的时间临界点. 即如果任务不能够在 $T_{switchup}$ 时间内完成,这表明若任务继续停留在小核,则 QoE 将开始显著下降,那么此时任务需要被迁移至大核以保证用户体验质量.

Step4. 任务从大核迁移回小核运行. $T_{switchdown}$ 被设置为任务从大核迁回至小核的时间临界点. 如果任务在 $T_{switchdown}$ 的时间内没有完成,这表示此任务给用户带来的体验已经很低,此时即使加速此任务,用户体验质量也无法改进. 因此,该任务将会被迁回至小核执行,以降低能耗.

另外,如果交互任务执行结束或者被转移至后台使得其不再处于交互状态,则无论任务此时运行于何种核心,都会迁移到小核上运行以节省能耗.

以上是任务在大小核之间迁移的基本流程. 在大核簇与小核簇的内部,仍旧按照传统同构多核调度算法的规则进行内部调度,充分发挥处理器性能.

### 2.3 交互事件信息

应用程序内部的上下文语义信息在到达操作系

统内核前就已丢失,因而交互行为很难被操作系统自动检测.因此,本文将引入一种跨层通信机制收集应用层信息并将其直接传递给调度器.

交互的开始与结束是基于程序的功能设计的,是具有上下文语义的,不能简单地以其是否占据前台或者是否有内容变化为条件进行判断.交互的形式可以是一个简单的信息提示,也可以是多种处理与复杂功能回调的综合反馈.因此,本文提出了一个UI检测辅助框架,应用开发人员可以很容易地使用此框架在代码中标注交互事件的开始与结束.框架中为每个交互事件赋予独立的ID,并在运行时将事件的开始和结束标志传递到调度器中,同时,程序前后台切换与屏幕关闭等信息也会被传递到调度器.

当一个处于交互事件中的任务被调度时,所有被此任务调用而产生依赖关系的任务都应被当作交互任务对待.因此,XOS算法中包含一个任务依赖关系检测机制.当应用程序中任务之间发生依赖关系时,依赖关系信息将被发送至调度器.XOS会根据应用层的运行信息,建立一个任务依赖关系树.在任务迁移的过程中,依赖关系树中所有的任务会绑定为一个整体,依据根结点的属性进行迁移判断.

由于跨层获取信息需要程序开发人员在程序源代码中添加支持代码,为了减少开发人员的工作量,本文将跨层通信机制所需的代码编写为库文件供应用开发人员直接调用.

## 3 仿真实验

### 3.1 仿真程序设计

为了验证XOS调度算法的有效性,并确定2.2节提到的2个时间临界点 $T_{\text{switchup}}$ 与 $T_{\text{switchdown}}$,首先采用程序模拟调度的方式进行仿真实验.在仿真实验中,同时实现了GTS算法和XOS算法的调度逻辑,并通过模拟对2种算法的性能进行比较.

仿真实验程序用JAVA语言编程,图4为其程序类图.程序中使用一个线程(SimulationThread)的一次循环执行代表内核时间片的一次运转.使用任务实体类(TaskEntity)来代表任务,并细分为短任务类(ShortTask)与长任务类(LongTask),所有的任务会存放于任务对象池(TaskObjectPool)数据结构中.任务以及任务对象池在仿真初始化时产生.仿真线程由任务实体类对象与调度算法类对象(GTS,XOS)组成,并在一次循环过程中分别按照GTS与XOS算法对这些任务对象进行模拟调度,并分别记录2种算法对于任务的执行结果.

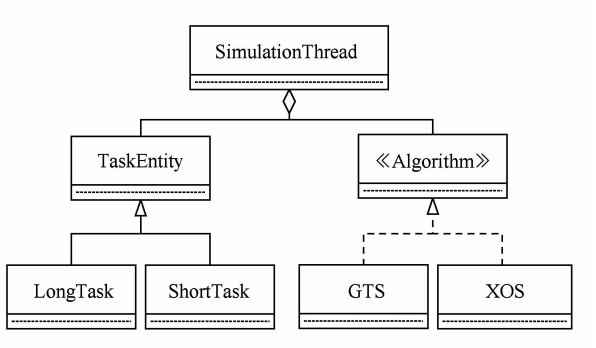


Fig. 4 The class diagram of the simulation problem.

图4 仿真实验程序类图

每个任务都有启动时间、持续时间、任务负载等属性,这些属性都是在仿真实验开始时随机生成,并且,短任务与长任务的持续时间有明显的差异.

### 3.2 仿真参数设置

在仿真实验中需要配置一些实验参数,例如真实环境中的硬件性能参数、能耗参数等.这些参数对于仿真实验的有效性至关重要.

1) 能耗参数设置

本文实验中使用了Odroid-XU3开发板,其CPU为三星Exynos 5422,包含4个Cortex-A15核心与4个Cortex-A7核心,组成big.LITTLE架构.开发板上附带了4个能耗传感器,能够实时监测大核簇、小核簇、GPU和内存的能耗信息.本文使用简单的循环计算程序对CPU进行能耗测试,同时,使用一个循环线程来读取能耗传感器的值.通过测量,小核单个核心在满负载状态下的能耗为180 mW,大核为1 555 mW.另外,当小核簇或者大核簇在开启时会产生基础能耗,小核簇的基础能耗为282 mW,大核簇的基础能耗为1 013 mW.最后,通过测试不同核心对SysBench[12]指标程序执行时间长短的不同,得到大核与小核的计算性能比为2.5∶1.以上信息将作为基本参数用于模拟实验.

2) 迁移临界设置

根据ITU的研究报告[11],任务的*MOS*值超过3.5即可计为用户体验优良状态,按式(1)即为任务的响应时间控制在430 ms之内.所以在仿真实验中选择430 ms作为任务从小核迁移到大核的临界点.

根据Agawi[13]的测试结果,智能移动设备的屏幕响应时间大约在100~130 ms之间.程序可用的运行时间应减去设备的响应时间,因此设置 $T_{\text{switchup}}=300$ ms.同时,根据式(1),设置 $T_{\text{switchdown}}=5$ s.

3) 任务集合设置

仿真实验设计了长任务集合与短任务集合. 每个集合内各自包含 500 个随机任务. 短任务的持续时间在 300 ms 之内随机选择,长任务的持续时间随机分布在 5～40 s 之间. 任务的 CPU 负载设置为 1%～100%之间的随机值. 这里的时间值、负载值都是任务在小核上运行时的数值,如果任务切换到大核上,那么这些值将按照大小核的计算性能比例,依据 Amdahl 定律进行转换. 通过随机设置启动时间,确保同一时间只有 1～3 个任务在进行仿真调度.

### 3.3 仿真结果

1) XOS 调度模型改进

通过对仿真实验数据的分析发现,在简单短任务调度的能效方面 XOS 比 GTS 具有明显优势,但随着任务在大核上执行时间的增长,XOS 算法能耗快速增加. 由于交互任务一直运行于大核,超过特定时间点后,XOS 算法的能效开始低于 GTS. 此后,使用 GTS 算法能够获得更好的能量利用率.

基于该方面的考虑,本文选择一定的时间之后使用 GTS 算法的调度逻辑来调度任务,以获取更优的能效. 即在原有的 2 个时间点 $T_{\text{switchup}}$ 与 $T_{\text{switchdown}}$ 之间,加入一个时间点 $T_{\text{switchtoGTS}}$. 图 5 展示的是加入 $T_{\text{switchtoGTS}}$ 时间点之后的调度模型. 如果一个任务在迁移到大核并且执行时间达到 $T_{\text{switchtoGTS}}$ 之后,仍旧没有完成,那么它将会遵循 GTS 的调度原则进行调度. 如果它在达到 $T_{\text{switchdown}}$ 时间点之后仍然没有完成,那么它将会按照 XOS 算法的调度原则强制迁移至小核.

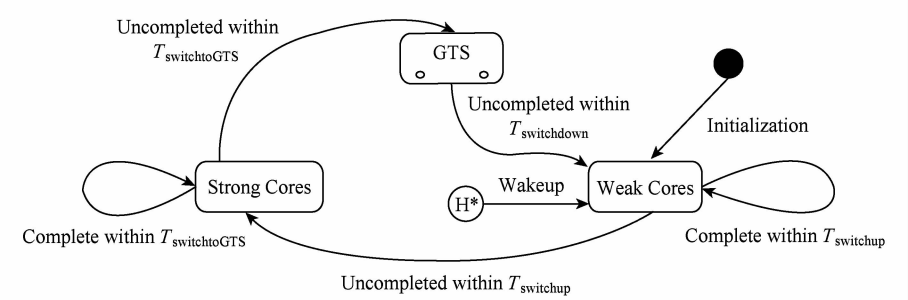


Fig. 5 The improved XOS schedule model.

图 5 改进的 XOS 模型

本文设计了一个实验以寻找 GTS 与 XOS 两种算法的时间临界点 $T_{\text{switchtoGTS}}$. 实验设置如下:首先取消 $T_{\text{switchdown}}$,即如果任务从小核迁移至大核,那么它将一直运行于大核直到完成. 按照任务持续时间的不同设置了 9 类任务集合,任务的持续时间从 1 100 ms 到 1 500 ms 递增,每类任务的持续时间增加 50 ms,第 1 类任务的任务持续时间为 1 100 ms,第 9 类任务集合的任务持续时间为 1 500 ms. 分别使用 2 种算法的调度逻辑调度这些任务,得到的实验结果如图 6 所示. 由图 6 可知,如果任务的响应时间超过 1 400 ms,那么 XOS 算法的能耗将会赶超 GTS 算法. 因此,在后续仿真实验中将 $T_{\text{switchtoGTS}}$ 设定为 1 400 ms.

2) 仿真结果分析

图 7 与图 8 分别为短任务集合与长任务集合的仿真实验结果. 表 1 为任务集合的 QoE 统计结果,包含用户体验优良($MOS \geqslant 3.5$)的任务数量,以及

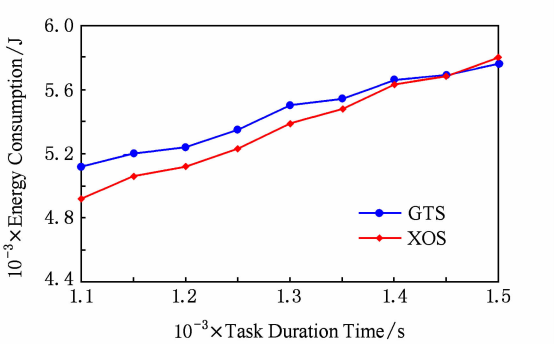


Fig. 6 The result of the experiment without $T_{\text{switchdown}}$.

图 6 去掉 $T_{\text{switchdown}}$ 之后的实验仿真结果

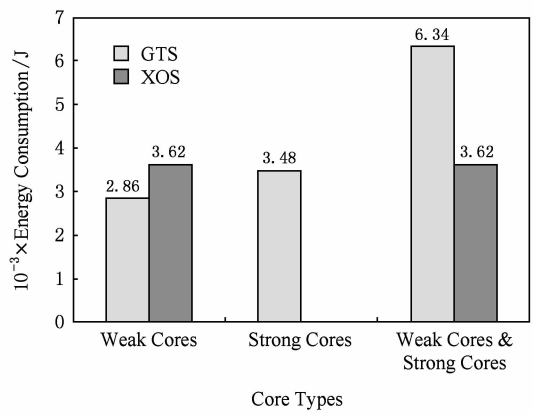


Fig. 7 The result of short tasks simulation.

图 7 仿真实验短任务能耗结果

所有任务根据式(1)计算出的用户体验 *MOS* 值的总和.如图 7 所示,XOS 算法应用于短任务集合的能耗低于 GTS 算法,而且 2 种算法得到的用户体验优良的任务数量均为 500.因此,XOS 使得短任务的能耗降低了 42.9%,虽然总用户体验 *MOS* 值略微降低,但是每个任务的用户体验仍处于优良.图 8 表明长任务的总体能耗降低了 14.7%.而从表 1 可得,由于长任务的用户体验 *MOS* 值不高,XOS 没有产生明显的用户体验质量损失.总体而言,XOS 算法能够在整体用户体验质量变化不大的情况下获得系统能耗的明显降低.

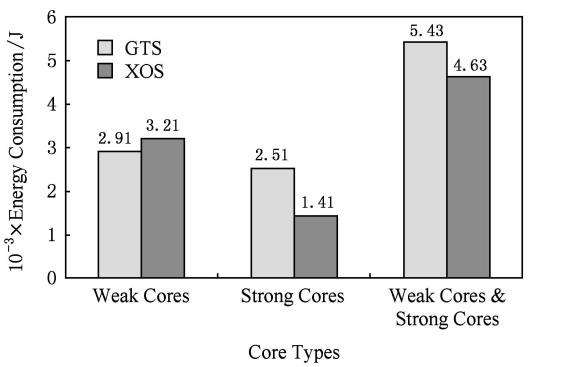


Fig. 8 The result of long tasks simulation.

图 8 仿真实验长任务能耗结果

**Table 1 The *MOS* Result of QoE in the Simulation**

**表 1 仿真实验中 QoE 的 *MOS* 统计结果**

| Algorithm | *MOS*≥3.5 | | *MOS* Sum | |
|---|---|---|---|---|
| | Short Tasks | Long Tasks | Short Tasks | Long Tasks |
| GTS | 500 | 0 | 2 467 | 315 |
| XOS | 500 | 0 | 2 272 | 313 |

## 4 XOS 算法实现

本文在 hardkernel 发布的 Linux-odroidxu3-3.10.y-android 的基础上实现了 XOS 调度算法,并将其部署运行在相应的 Odroid-XU3 开发板上进行验证.按照模拟仿真实验的结果,XOS 中还保持了 GTS 算法的调度逻辑并在特定的条件下被激活,因此,该调度算法在 GTS 的数据结构和代码逻辑基础上扩展实现.

1) 算法的内核实现

在数据存储上,扩展 *task_struct* 中的数据结构 *sched_avg*,加入对进程的交互状态(*qoe_in_interaction_process*)、交互开始时间(*qoe_interaction_start_time_jiffies*)、大小核切换时间(*qoe_switch_to_highprocess_for_performance*, *qoe_switch_to_lowprocess_for_saving*)等信息的存储,并在相应的时机进行更新.

在执行逻辑上,借助 GTS 对迁移条件检查的时间点对 QoE 和交互状态进行检查,并完成任务的核心迁移.在 GTS 算法中一共有 5 个迁移点,在内核源代码中分别实现为 select_task_rq_fair, run_rebalance_domains, hmp_force_up_migration, hmp_idle_pull, hmp_offload_down.每个任务的负载在 __update_task_entity_contrib 中定期更新,并检查是否需要触发任务迁移事件.在迁移时,任务会被 hmp_up_migration 与 hmp_down_migration 加以评估,以决定一个任务是否继续在原有的核上运行.

借用 GTS 的接口,在每次任务负载更新函数 __update_task_entity_contrib 被调用时,XOS 检查任务的交互状态和在当前核心上的运行时间,以决定是否需要触发迁移以及迁移的方向.类似地,在其他 4 个迁移点,XOS 也需要判断迁移条件和方向.此部分逻辑在每次确定迁移方向前都被调用,因此在内核中被实现为一个内联函数 should_stay_low_cpu.同时,在迁移条件的检查点也需要确认任务在大核上的运行时间和交互状态,以确定是否应该使用 GTS 调度算法,该内联函数实现为 should_follow_gts.为了保证 GTS 算法在接管调度器时能够正确地计量任务的负载,所有 GTS 算法相关的数据结构和属性更新均保持原有执行逻辑,不受 XOS 算法的影响.

任务的交互状态信息由应用程序借助 sysfs 接口传递给内核.调度器通过在 sysfs 中注册一些属性文件实现与用户空间的交互,其中 move_foreground 与 move_background 用于向内核传递任务的状态变化,由交互任务向文件中写入对应的任务号即可通知内核改变其交互状态.对于任务依赖关系,则通过 attached_to_foreground, detached_from_foreground 两个接口获得.被前台任务调用的任务,将其调用者的任务号写入相应接口即可通知调度器.

任务间的依赖关系在内核中以树的形式记录.每个任务的 task_struct 数据结构中加入变量 qoe_attached_to_process.如果任务没有依赖关系,那么这个任务的依赖树就是一棵只有根节点的树,该变量中保存的是该任务自己的任务号.当任务产生依赖时,被调用者记录了调用者的任务号,形成一个以父节点形式存储的树状结构.在进行迁移条件检查

时,如果发现任务存在对其他任务的依赖,则沿着该组任务的 qoe_attached_to_process 寻找到这个依赖树的树根,并以根任务的属性进行迁移条件判断.

2) 跨层类库实现

为了减少应用开发人员在跨层通信时的编程负担,本文将 Android 层的信息传递代码编写为 Java 类库.开发人员可以通过直接调用类的静态方法进行交互信息传递.静态方法有 2 个:标识交互事件开始的方法 foreground(int *pid*)和标识交互任务结束的方法 background(int *pid*).对于任务依赖关系,可以通过另外 2 个静态方法 attachForeground(int pid)与 attachBackground(int *pid*)进行传递.第 5 节中均使用这个类库来进行跨层信息的传递.

上述方法中的参数为交互线程的任务 ID 编号.由于 Android 层的线程编号与对应的 Linux 内核中的线程编号不同,因此在 Android 层需要使用 android.os.Process.myTid()方法来获取对应于 Linux 内核层的线程的任务 ID.

## 5 实验与评估

### 5.1 实验设计

为了验证 XOS 算法的实际运行效率,本文设计一系列实验,将任务分别使用 GTS 与 XOS 执行,并分别统计响应时间与能耗信息,以比较 2 种算法的性能.如表 2 所示,本文抽取了在智能移动设备上用户的几类典型交互行为,包括图片上传、网页浏览、音乐播放、视频播放以及电子书阅读,并为每种行为都分别设计了一组实验.由于图片上传、网页浏览交互行为的响应时间都比较短,为减少实验误差,在实验中将每种操作重复 10 次,每 10 次作为一次循环,一共循环 4 次.实验的硬件环境仍为 Odroid-XU3 开发板.

**Table 2 The Tests Table**

**表 2 测试实验统计表**

| Test | Application | Loops | Execution Times in One Loop |
|---|---|---|---|
| Image Uploading | WeChat | 4 | 10 |
| Webpage Loading | Web Browser | 4 | 10 |
| Music Playing | Kuwo Music | 4 | 1 |
| Video Playing | Strom Player | 4 | 1 |
| Book Reading | Duokan Reader | 4 | 1 |

由于缺少商用 Android 应用程序的源代码,跨层通信机制难以实现,因而本文设计了新的程序实现对这几种交互行为的模拟测试.为了减少其他任务的影响,实验程序包含 6 个 Activity 界面,其中一个为主界面,从主界面可以跳转到实现这些交互行为的测试界面上.使用第 4 节描述的 Java 类库来进行跨层通信,仅需要在这些实验程序模块中各自添加两行代码标识交互事件开始、交互事件结束即可实现,大大减少了跨层通信的代码工作量,有效降低了开发人员在使用此调度算法时的负担.

对于图片上传测试,本文搭建了一个基于 J2EE 的服务器用以接受客户端发来的图片.每次上传大小均为 2 097 152 B 的一张图片.测试程序与服务器处于局域网中来减少网络波动对实验的影响.对于网页浏览测试,选择淘宝网站主页(www.taobao.com)作为加载页面.对于音乐播放测试,本文选取了播放长度为 27 s 的 MP3 音乐文件作为测试对象.对于视频播放测试,测试对象为播放长度为 55 s 的 MP4 视频文件.由于 MP3,MP4 文件格式能够被 Android 系统在无第三方应用的情况下支持,所以使用这 2 种文件作为测试对象,能够在最大程度上减少其他因素的影响.对于电子书阅读测试,测试程序读取大小为 5 MB 的 TXT 文件,并在屏幕上显示 20 行文字.每隔 2 s 时间,屏幕会自动翻页并显示接下来的 20 行文字,一次测试一共执行 20 次自动翻页.

在每个测试执行过程中,启用一个采样线程来获取测试实验的能耗数据以及响应时间.由于采样线程无论以何种采样频率进行采样,都会有一定的能耗,所以本文在测试实验结束之后将采样线程本身的能耗从结果中减去,以减少采样线程所带来的实验误差.

### 5.2 实验分析

实验结束后,本文通过采样线程分别收集了 2 种算法的实验能耗数据与响应时间数据.

图 9 为实验能耗统计结果,5 种实验测试在 XOS 算法下的能耗相对于 GTS 算法分别降低了 48.5%,12.1%,9.8%,28.2%与 8.0%.

测试程序的执行时间如图 10 所示,图片上传的执行时间增加了 15.9%,而网页浏览的执行时间增加了 7.0%.由于音乐播放、视频播放与电子书阅读行为属于长任务,其 QoE 值已经很低,所以对于这 3 种操作,本文仅统计其能耗结果.根据式(1),将响应时间转换为用户体验 *MOS* 值的统计结果如表 3

所示,XOS 算法对于图片上传操作的用户体验降低了 7.3%左右,而对于网页浏览操作降低了 2.7%.

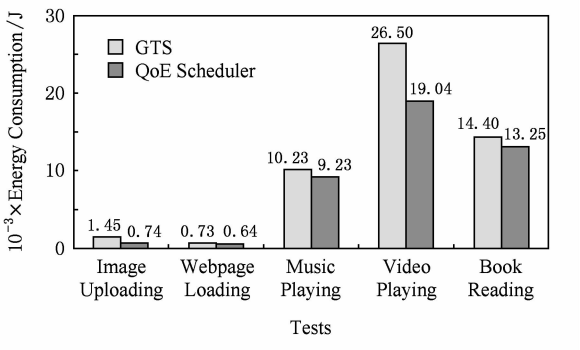


Fig. 9 Energy consumption with GTS and XOS.

图 9 实验能耗结果

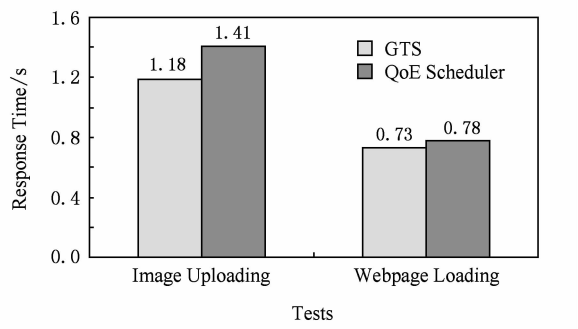


Fig. 10 Response time of tasks with GTS and XOS.

图 10 实验响应时间结果

**Table 3 The *MOS* Result of QoE in the Experiment**

**表 3 实验中 QoE 的 *MOS* 统计结果**

| Algorithm | Image Uploading | Webpage Loading |
|---|---|---|
| GTS | 2.48 | 2.98 |
| XOS | 2.30 | 2.90 |

总体而言,XOS 算法使得任务的执行时间略有延长,用户体验轻微下降,但是实现了能耗的显著降低.特别是对于执行时间不受硬件影响的任务(如音乐播放等),XOS 能够明显降低其能耗.

## 6 相关工作

在先前的研究中,许多研究者聚焦于异构多核架构的任务调度.现有调度算法可大致分为 2 类:1)粗粒度算法,其调度对象为线程或进程;2)细粒度算法,其调度对象为代码段[14].

粗粒度调度算法中有公平算法与不公平算法 2 种.Van Craeynest 等人提出了一种公平调度算法,试图保证每个线程在每种类型的核心上运行相等的时间,从而达到公平的目的[15].但是这个调度算法仅试图将所有的核心利用起来,并未对性能与能耗进行优化.该团队同时还提出了另一种非公平调度算法 PIE(performance impact estimation),通过收集任务运行时 CPI 栈、MLP、ILP 的信息决定任务适合在何种类型的核心上运行[9].此外,Li 等人也提出了一种大核优先算法,尽可能让新的任务运行在大核上以充分利用大核的计算资源[8].以上这 2 种算法仅考虑了性能优化而忽视了系统的能耗约束.Koufaty 等人提出了一种偏好算法,能够识别一个线程的运行是偏向于大核还是小核,继而选择最合适的核心来运行[16].该算法能够在一定程度上兼顾性能与能耗,但是仍旧没有考虑用户体验.

细粒度的调度算法侧重于代码段的执行,这种类型的调度算法能够对任务进行更精细的分割.Suleman 等人提出了 ACS(accelerated critical sections)算法,能够加速线程中临界区代码段的执行速度[17],但是对于交互敏感的非互斥任务作用不大.Sondag 等人提出了一种基于代码分段的算法,将代码分段,根据预先建立的信息库匹配代码段的特性,用于指导该段代码的核心分配[18].研究显示,高并行化的代码在大量低性能核心上运行的效率是在少量高性能核心上的 2 倍左右[19].鉴于此,Saez 等人提出了一种轻量算法,能够快速识别出线程并发度高的代码和处理器密集型代码,从而针对性地分配核心以发挥优势[20].蒋建春等人设计了静态任务调度算法,能够使得任务在最短时间内完成并实现大小核心的负载均衡[21].这些算法能够加速任务执行、缩短响应时间,但是这些算法侧重于性能的提升,没有考虑智能移动设备的能耗约束和应用软件的复杂性,所以不能很好地应用于商业产品中.彭蔓蔓等人在使用改进的启发式方法进行任务分组的同时,又使用遗传算法进行任务的调度,能够实现在实时性要求较低的情况下以较小的时间代价来节省较多的能耗[22].然而对于交互性任务,其实时性要求较高,所以这种调度算法并不能满足用户的交互体验.

## 7 总 结

本文设计了一种面向用户体验的高能效异构多核调度算法 XOS,试图兼顾智能移动设备的能效与用户体验.首先本文使用一个仿真实验验证了算法的有效性,并对算法模型进行了改进,同时确定了调

度时间临界点.随后,在 Android 平台和在 Linux 系统内核中实现了该算法,并通过应用程序实测检测算法的性能.实验测试结果显示,XOS 能够在保障用户体验的前提下使系统能耗得到明显的降低.

## 参 考 文 献


[1] Kumar R, Farkas K I, Jouppi N P, et al. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction [C] //Proc of the 36th Annual IEEE/ACM Int Symp on Microarchitecture. Los Alamitos, CA: IEEE Computer Society, 2003: 81-92

[2] Kumar R, Tullsen D M, Ranganathan P, et al. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance [J]. ACM SIGARCH Computer Architecture News, 2004, 32(2): 64-64

[3] Duran A, Ayguadé E, Badia R M, et al. Ompss: A proposal for programming heterogeneous multi-core architectures [J]. Parallel Processing Letters, 2011, 21(2): 173-193

[4] Wang P H, Collins J D, Chinya G N, et al. EXOCHI: Architecture and programming environment for a heterogeneous multi-core multithreaded system [C] //Proc of the 28th ACM SIGPLAN Conf on Programming Language Design and Implementation. New York: ACM, 2007: 156-166

[5] Chen Fangyuan, Zhang Dongsong, Wang Zhiying. Research of the heterogeneous multi-core processor architecture design [J]. Computer Engineering & Science, 2011, 33(12): 27-36 (in Chinese)
(陈芳园, 张冬松, 王志英. 异构多核处理器体系结构设计研究[J]. 计算机工程与科学, 2011, 33(12): 27-36)

[6] Lin F X, Wang Z, Zhong L. K2: A mobile operating system for heterogeneous coherence domains [J]. ACM Trans on Computer Systems, 2015, 33(2): 4:1-4:27

[7] Greenhalgh P. big. LITTLE processing with ARM Cortex-A15 & Cortex-A7 [EB/OL]. [2016-03-01]. http://www.eetimes.com/document.asp?doc_id=1279167

[8] Li T, Baumberger D, Koufaty D A, et al. Efficient operating system scheduling for performance-asymmetric multi-core architectures [C] //Proc of the 2007 ACM/IEEE Conf on Supercomputing. New York: ACM, 2007: 53-53

[9] Van Craeynest K, Jaleel A, Eeckhout L, et al. Scheduling heterogeneous multi-cores through performance impact estimation (PIE) [C] //Proc of the 39th Annual Int Symp on Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2012: 213-224

[10] ARM. big. LITTLE Technology: The Future of Mobile [EB/OL]. [2016-03-01]. http://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf

[11] Kuipers F, Kooij R, De Vleeschauwer D, et al. Techniques for measuring quality of experience [C] //Proc of the 8th Int Conf on Wired/Wireless Internet Communications. Berlin: Springer, 2010: 216-227

[12] Kopytov A. SysBench: A system performance benchmark [EB/OL]. [2016-03-01]. http://sysbench.sourceforge.net

[13] Agawi App Glimpse. TouchMarks I: Smartphone Touchscreen Latencies [EB/OL]. [2016-03-01]. http://appglimpse.com/blog/touchmarks-i-smart-phone-touch-screen-latencies

[14] Bambagini M, Marinoni M, Aydin H, et al. Energy-aware scheduling for real-time systems: A survey [J]. ACM Trans on Embedded Computing Systems (TECS), 2016, 15(1): 7:1-7:34

[15] Van Craeynest K, Akram S, Heirman W, et al. Fairness-aware scheduling on single-ISA heterogeneous multi-cores [C] //Proc of the 22nd Int Conf on Parallel Architectures and Compilation Techniques. Piscataway, NJ: IEEE, 2013: 177-188

[16] Koufaty D, Reddy D, Hahn S. Bias scheduling in heterogeneous multi-core architectures [C] //Proc of the 5th European Conf on Computer Systems. New York: ACM, 2010: 125-138

[17] Suleman M A, Mutlu O, Qureshi M K, et al. Accelerating critical section execution with asymmetric multi-core architectures [C] //Proc of the 14th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2009: 253-264

[18] Sondag T, Rajan H. Phase-guided thread-to-core assignment for improved utilization of performance-asymmetric multi-core processors [C] //Proc of the 2009 ICSE Workshop on Multicore Software Engineering. Los Alamitos, CA: IEEE Computer Society, 2009: 73-80

[19] Hill M D, Marty M R. Amdahl's law in the multicore era [J]. Computer, 2008, 41(7): 33-38

[20] Saez J C, Prieto M, Fedorova A, et al. A comprehensive scheduler for asymmetric multicore systems [C] //Proc of the 5th European Conf on Computer Systems. New York: ACM, 2010: 139-152

[21] Jiang Jianchun, Wang Tongqing. Task scheduling algorithm for heterogeneous multi-core processor [J]. Computer Engineering and Applications, 2009, 45(33): 52-56 (in Chinese)
(蒋建春, 汪同庆. 异构多核处理器的任务调度算法[J]. 计算机工程与应用, 2009, 45(33): 52-56)

[22] Peng Manman, Xu Lichao, Wang Ying. Task allocation and energy on heterogeneous multi-core processors [J]. Application Research of Computers, 2010, 27(5): 1729-1736 (in Chinese)
(彭蔓蔓, 徐立超, 王颖. 异构多核处理器的任务分配及能耗的研究[J]. 计算机应用研究, 2010, 27(5): 1729-1736)

**Gong Xiaoli**, born in 1983. PhD and assistant professor in Nankai University. Member of China Computer Federation. His main research interests include embedded system design and optimization.

**Li Tao**, born in 1977. PhD and associate professor in Nankai University. Member of China Computer Federation. His main research interests include high performance computing and parallelized computing (litao@nankai.edu.cn).

**Yu Haiyang**, born in 1991. Bachelor in Nankai University. His main research interests include energy-efficient computing and human-computer interaction (oceanisher@mail.nankai.edu.cn).

**Zhang Jin**, born in 1979. PhD and associate professor in Nankai University. Member of China Computer Federation. His main research interests include mobile cloud computing (nkzhangjin@nankai.edu.cn).

**Sun Chengjun**, born in 1992. Bachelor in Nankai University. Her main research interests include energy-efficient computing and human-computer interaction (suncheng jun2015@mail.nankai.edu.cn).

**Ma Jie**, born in 1957. Bachelor and professor in Nankai University. Member of China Computer Federation. His main research interests include new media technology (majie1765@nankai.edu.cn).