ORBIT:一个基于智能手机的数据密集型嵌入式传感应用平台

Mohammad-Mahdi Moazzami* Dennis E. Phillips* Rui Tan* Guoliang Xing

密歇根州立大学计算机科学与工程系，密歇根州东兰辛，
美国先进数字科学中心，新加坡伊利诺斯州
浙江大学计算机学院软件工程 1801 班 3180103772 张溢弛 翻译

摘要：

由于丰富的处理方式，多模态传感和多功能的网络传输功能，智能手机被越来越多地用于构建数据密集型的嵌入式传感应用，然而在智能手机用作通用的嵌入式传感平台之前必须系统地面对各种各样的挑战，包括功耗过高，缺乏实时功能和对用户友好的嵌入式编程支持。本文着重介绍了 ORBIT——一个基于智能手机的数据密集型嵌入式传感应用平台。ORBIT 采用多层式的架构，智能手机可以连接到节省能耗的的外围主板或者云服务中。ORBIT 作为一个平台，在利用智能手机的优势的同时，解决了目前的智能手机使用时间短的问题。ORBIT 提供了基于配置文件的任务分区，允许其在各个功能层之间智能地处理与调度任务并借此来降低智能手机系统的功耗。ORBIT 还提供了包括两种机制的数据修正库，其一为自适应性去布局与权衡机制，其二为通过多线程来操作资源使用。除此之外，ORBIT 还为开发人员提供了拥有注释的编程 API 用来简化程序的开发并使得编程更加灵活。覆盖面广的微基准评估与两个具体案例研究(地震传感与多摄像机的三位重建)验证了 ORBIT 的强大功能。

分类和主题描述：C.3[特殊用途系统]：实时系统，嵌入式系统，信号处理系统

关键词：智能手机，嵌入式传感器，数据处理，数据密集型应用程序

1. 基本介绍

智能手机的普及以及其多模态的传感功能已经在移动传感应用中取得了广泛的应用，这些应用通常以人为中心，因为智能手机利用自带的传感器来感知人们的特征和他们的所处的情景。与这些以人为中心的传感应用不同，本文着重介绍了一种新兴的基于智能手机的数据密集型嵌入式传感应用。与以人为本的参与式传感不同，这些应用中的智能手机被嵌入到了环境中，以便长时间自主地感知物理世界并与之进行交互。例如在浮动传感器网络项目中[9]，配备智能手机的漂流器被快速部署用以收集一条河流中有关水流的实时数据。智能手机的 GPS 系统还允许漂流器根据实时位置测定水流的体积和方向并通过一共蜂窝网络将数据传输给服务器。智能手机还被用于监测地震[7]，火山[13]，甚至微型的卫星[22]。另一个重要的基于智能手机的嵌入式系统是云机器人[4][11]，通过集成智能手机，这些机器人可以利用大量的手机传感器来实现复杂的传感导航功能，并将图像和语音等计算密集型的认知任务加载到云端。

与传统的移动传感平台相比，智能手机具有几个显著的优势，这使得它们在上述嵌入式系统中得到应用。这些功能包括：能够执行高级的多核处理器数据处理算法，拥有多个网络接口和各种集成传感器，友好的用户界面和对各级编程语言的支持。此外，智能手机的价格在过去的十年中一直在显著先将，许多配置优秀的安卓手机（具有高达 800MHz 的 CPU 与 2GB 内存）成本更是低于 50 美元。

然而在智能手机被用作嵌入式传感应用的系统平台之前，我们必须面对一些挑战。首先，智能手机的电池在起初设计时就被设计为只能维持几天的使用时长，不适合运行需要有非常

多传感功能的程序，这些程序必须长时间在没有人监管的情况下运行，当今的许多嵌入式应用本质上都是数据密集型的，因为传感器必须以高频率采样。连续的传感器采样会阻止手机进入睡眠模式，导致电池在几个小时里迅速耗尽。此外当前主要的智能手机操作系统并不提供实时功能，例如恒定采样率，精确时间戳和用于表示时序要求的 API 接口，这对于许多嵌入式系统而言非常重要。例如，根据我们的测定结果，安卓手机中的 USB 硬件中断会遭受高达 5 毫秒的不可控延迟，因此无法实现高恒定采样频率。最后，智能手机的编程环境虽然简化了嵌入式系统生命周期中的许多编程任务，比如调试，远程数据记录，可视化和软件维护等功能，但是缺乏重要的嵌入式编程支持，如搞笑的信号处理库和外围传感器通信单元。

在本文中，我们迈出了应对这些挑战的第一步，我们将展示 ORBIT，一种基于智能手机的嵌入式传感系统，特别是现成的 ORBIT 智能手机可以满足数据密集型嵌入式传感的功耗和拥有较高的及时性要求的应用。ORBIT 基于分层式的结构，最多包含云端，智能手机，一个或多个与智能手机相交的节能外设板(被称为 extBoard)这样的三层。目前已经有许多 extBoard 平台(例如 Arduino 和 IOIO)供大家使用。如果智能手机上内置的传感器不适合传感应用，这些主板可以通过 USB 或者蓝牙接口将各种附件(如外部传感器)轻松集成到安卓手机中。我们进行了安卓智能手机和 extBoard 平台的延迟和功耗测定研究，结果表明，这两个平台具有高度异构但是互补的延迟配置文件：智能手机具有更高的能效，由于其更快的处理能力，而由于操作系统的耗用过高，时序精度比较差，这些结果对有效的任务分区有非常重要的影响。虽然智能手机和云端在处理计算密集型任务时需要花费较长时间，但是由于硬件计时器的存在，必须将高频率传感器采样和精确时间戳等关键的时间相关的功能转移到 extBoard 平台中进行高效的中断处理。

基于以上测定结果，我们提出了一个任务划分的框架，该框架根据任务的时间临界性，计算强度和异构延迟，功耗配置文件将任务分配到了不同的层。此外为了利用智能手机上性能逐渐提升的多核处理器，ORBIT 实现了一种数据分区方案，将基于矩阵的计算分解为多个县城。ORBIT 还集成了一个数据处理库，支持高级 Java 和标记应用程序编程。该库的设计通过促进延迟-质量平衡机制，简化了嵌入式应用程序的资源管理。为了支持动态任务的调度和运行时进行的任务分析，我们开发了一个由运行在每一层上的任务控制器租车给的轨道运行时环境。这些控制器通过统一的传输协议协调任务的执行。基于这些功能，ORBIT 是一个拥有强大功能的系统工具包，可以用于构建广泛的数据密集型嵌入式传感应用。

本文将做出如下贡献：首先，我们采用了系统地测定和建模方法，了解使用智能手机数据密集型的嵌入式传感应用，我们的测定结果对于广义上的基于智能手机传感系统的设计也很有用；其次，我们提供了作为库的多种数据处理算法的实现，以及一些用来提供智能手机和扩展板数据处理算法效率的机制，ORBIT 的几个组件与现有的嵌入式系统平台有些相似[5,10,23,28]，但据我们所知，ORBIT 是第一个通用的，可扩展的应用程序传感和端到端的传感处理平台，适用于基于智能手机的数据密集型嵌入式应用，最后，我们展示了 ORBIT 作为一个平台的通用性和灵活性，我们展示了 ORBIT 上建立两种应用的模型:地震传感和多机位的三维重建，灵活的任务分区和框架调度是的 ORBIT 能够适应不同的任务结构，应用程序截止时间时间需求和通信延误。实验表明，与基线的方法相比，ORBIT 可将能耗降低 50%。

2. 相关工作

基于智能手机的移动传感近来受到了极大的关注。大多数研究着重关注了以人为中心的环境相关的问题，包括多个并发感知应用程序之间的互相协调[15,16,17]和环境分类器[3]，等感知算法。最近没智能手机已经在许多嵌入式传感应用中得到了应用，在[7]中，智能手机被用来构建一个使用板上加速度计的地震预警系统。而在漂流传感器网络项目[9]中，配备智

能手机的漂流者被部署来监控水道，并根据手机的 GPS 实时收集水量和水流的方法。美国国家航天局(NASA)的电话项目[22]发射了搭载安卓智能手机的低成本卫星。在智能手机的控制下，这样的小型卫星可以执行各种任务，比如地球观测和空间碎片的跟踪。这几项研究的重点是构建集成智能手机和机器人的云机器人[11]。手机内置的传感器用于感知和导航，而图像语音的识别等计算密集型的任务被转移到云端进行。

智能手机的各种任务的卸载方案目前已经被开发出来，Spectra[8]允许程序员指定任务分区计划，给定面向应用程序的任务要求。Chroma[2]旨在减轻人工定义详细分区计划的负担，Medusa[25]具有分布式的运行时系统，用于协调智能手机和云端之间的任务执行。Turducken[28]采用分层电源管理架构，其中笔记本电脑可以将轻量级任务卸载到 PDA 和传感器。虽然它提供了用于分区的分层硬件体系结构，但是这项技术依赖于程序开发人员来设计实现能效的层数。

与这些任务分区方案不同，ORBIT 调度基于智能手机的多层体系结构来执行传感和处理任务，以实现数据密集型应用程序的各项要求。ORBIT 可以最大限度地延长电池的寿命，但是受到特定应用程序的延迟限制。此外，为了支持跨层进行细粒度的任务分区，开发者通过 Java 注释或者基于 XML 的注释指定应用程序的任务结构以及实时要求 ORBIT 提供的应用模型。ORBIT 要提供了信息传递的结构以支持异构层之间和不同程序组件之间的统一数据传递机制。

MAUI[5]支持细粒度的卸载机制以延长智能手机的寿命，但是 MAUI 依赖于 Microsoft.NET 托管的代码环境属性来表示可以远程执行的功能。远程执行函数时，MAUI 会通过假设保存与其本地执行相关的能量。相反，ORBIT 不依赖于任何语言，特定的环境以及基于测量的功率配置文件，并且考虑到许多现实的电源特性，如 CPU 睡眠，唤醒和延迟时间。

Wishbone 系统还具有任务调度方案，与 Turducken 不同，Wishbone 使用基于配置文件的方法来查找最佳分区。它仅考虑网络内层和服务器层两个层面。与 MAUI 不同，Wishbone 仅依赖于计时配置文件，并不考虑功耗。ORBIT 与 Wishbone 在多个方面有所不同，Wishbone 只是用 CPU 和网络计时配置文件来查找最佳任务分区，而 ORBIT 则考虑测量的延迟和功率消耗从而使得任务分区更加节能，更多时候 Wishbone 取决于基于样本数据的计时配置文件，前提是示例数据可以表示实际运行时的数据。然而我们的研究表明，信号处理时序曲线在真实场景中可能表现出显著差异。为了解决这个问题，ORBIT 在运行时会测量统计计时配置文件的数量并定期进行优化。此外，Wishbone 将分区问题当作 0/1 整数的线性编程问题，因此只支持两个分层，而 ORBIT 作为非线性的优化问题，支持三个及更多的分层。

RTDroid[30]解决了安卓系统缺乏硬核实时能力的问题，通过重新设计和替换 Dalvik 中的几个安卓组件(比如 LooperHandler 和 AlarmManager)来解决这个问题，相比之下，ORBIT 不需要改变安卓系统，而是考虑了任务执行时的统计特性，通过任务划分机制找到最佳执行分配的方案，因此，尽管 RTDroid 和 ORBIT 解决的问题不同，但他们是互补的，事实上，ORBIT 可以运行在 RTDroid 上，而基于 ORBIT 的传感应用则可以从两者中受益。

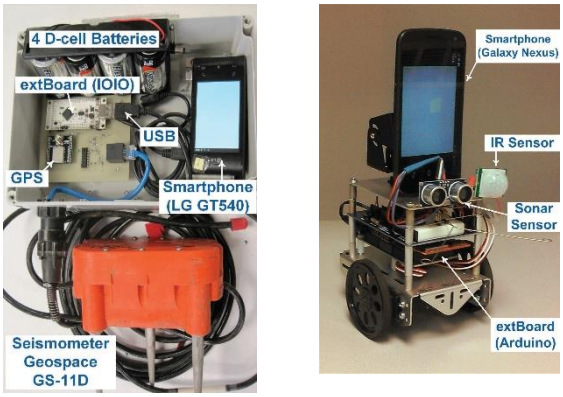
与 ORBIT 相似，EmStar 提供了一个环境，用于实现基于 Linux 类微型服务器的传感应用的分布式嵌入式系统，但是 ORBIT 则基于最初设计并非为了开发嵌入式系统的智能手机，提出了新的设计。底层技术的差异导致了完全不同的设计和实现。虽然 EmStar 和 ORBIT 具有类似的模块化设计，但与 ORBIT 不同，EmStar 没有分区机制，也没有严格的层次区分。更重要的是，ORBIT 提供了一个数据处理的算法库，这些算法在资源受限的智能手机上非常高效，但这些在 EmStar 设计时并没有成为设计的目标。

3. 动机与机制概述

在本节中，我们将讨论使用智能手机作为数据密集型嵌入式传感应用系统平台的动机和ORBIT 的设计目标

3. 1 动机与挑战

在过去的十年中，注入 TelosB 之类的莫特级传感平台已经被嵌入式传感应用程序广泛采用，然而，由于有限的处理和存储能力，它们不适合高采样率的传感应用。近年来，Gumstix[12]、Sheevaplug[20]，Raspberrt Pi[26]等具有丰富处理和存储能力的单板计算机在嵌入式应用中得到了越来越多的应用，然而，他们的设计并没有特别针对低功率传感进行优化。此外，因为没有车载传感器和无线接口，它们需要配备各种外围设备。



(a) 地震传感ORBIT节点　　(b) 机器人ORBIT节点

图片1：ORBIT节点.

与上述平台不同，商用现货智能手机提供了几个突出的优势，使它们成为数据密集型嵌入式传感应用的一个有前途的系统平台，具有计算量大，存储空间大，拥有多种网络接口和传感方式，可增加的可用多核架构和较低的成本。此外，智能手机具有先进的编程语言和有好的用户界面(如触摸屏)丰富了互动与显示，不同于有限的用户界面和嵌入式计算机(如 LED 和按钮)

然而，在构建基于 COTS 智能手机的嵌入式传感平台方面，我们仍然面临如下挑战：

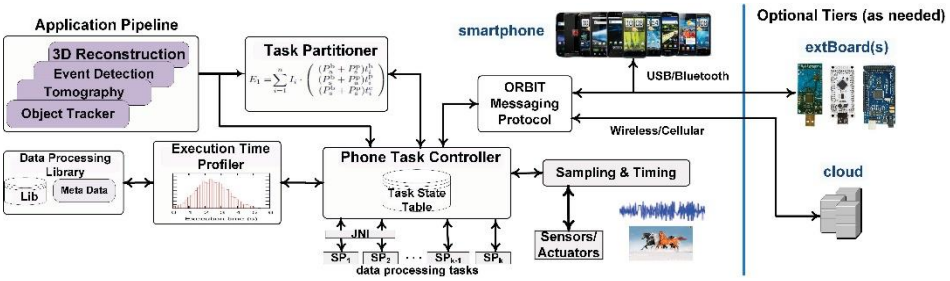
(1).高功耗：智能手机电源的管理方案被设计得适应用户活动，延长电池续航时间。然而，它们不适合无栓的嵌入式传感系统。如果智能手机持续对传感器进行采样，它的 CPU 就无法进入深度睡眠状态来节省能源，低功耗的协处理器可以处理连续的采样，但只有少数高端型号可以使用。

(2).实时性差：很多传感应用对实时性有着非常严格的要求，如采样率恒定，时间戳精确等。然而，现代智能手机操作系统并不是为满足这些实时需求而设计的。例如，传感器采样可以延迟高优先级的 CPU 任务，如安卓系统服务或用户图形界面的绘图。我们的测量结果表明，安卓提供的软件定时器可能会被安卓核心系统服务阻塞 1.1 亿分之一秒，此外，安卓编程库没有提供允许开发人员表达时间需求的本机接口。

(3).缺乏嵌入式编程的支持：智能手机的编程环境旨在促进以人为中心的网络化移动应用程序的发展，但是它缺乏重要的嵌入式编程支持，比如资源效率高的信号处理库以及用于控制与外部传感器等外围设备通信的统一语言

3. 2 系统概述

在本文中我们提出的 ORBIT 旨在解决上述三个主要的挑战，一个 ORBIT 节点包括一台安卓智能手机，一个 extBoard(例如 IOIO[14],Arduino[1]),可能还有一个在云端的运行时系统。extBoard 通过 USB 线或者蓝牙连接到智能手机上进行通信。它配备了一个低功耗的 MCU，例如，ATmega2560 具有 16mhz 的频率，8kb 的 RAM，和一个模拟到数字(a /D)转换器，可



以集成各类模拟传感器。

图片2：ORBIT系统结构

ORBIT 旨在满足以下三项要求：

(1).能效，同时考虑到及时性的要求：ORBIT 利用多层的异构延迟特性，将总体能耗降低到最低。它还对应用程序的计时延迟进行统计分析，并将这些模型应用于任务分区和执行，我们注意到，ORBIT 无法实现硬性实时保证，但是统计任务计时模型允许用更高的概率满足任务截止时间。

(2).可编程性：ORBIT 提供了基于 API 组件的编程环境，允许开发人无需构建传感应用处理系统设计的低级问题

(3).兼容性：ORBIT 仅依赖于 COTS 智能手机的开机即用功能，无需内核级的自定义或者设备进行刷机。这不仅最大限度地减少了应用程序开发人员的负担，而且确保了与各种智能手机型号的兼容性，下面我们将描述一些主要的 ORBIT 组件。

ORBIT 库与应用模型：ORBIT 库提供了具有统一接口的信号处理算法库。它们可以很容易地组成各种先进的传感应用。该库提供了一个称为 connection 的编程原语，允许程序员在 XML 文件中或通过 Java 注释指定应用程序组合，每个算法都可以在任何层上执行从而支持灵活的任务分配。

任务/数据分区器和执行时间分析器：为了满足感知应用程序的最后期限，时间关键型任务应该在 extBoard 上执行，而计算密集型任务应该在智能手机或者云端执行，我们正式提出了一个任务划分问题，其墓地是在时间关键型任务的处理延迟约束之下最小化智能手机的能耗，任务分区器解决了这个问题，得到了最优的任务调度方案，该设计的一个挑战是信号处理任务可能具有高度可变的执行时间，我们设计了一个在线的分析器，在运行时测量任务执行时间并且动态运行任务分区器。此外，ORBIT 采用了一种数据分区方案，将基于矩阵的计算分解为了多个区域，以利用智能手机上不断进步的多核可用性。

任务控制器和统一的信息传递协议：在运行时不同层上的任务控制器卸载实例化任务并按照任务分配计划来执行这些任务，extBoard 运行低级别的实施功能，这些任务需要多个轨道节点的数据。为了方便这种灵活的任务调度和控制机制，我们开发了一个统一的信息传递协议用于在本机通信通道(如 USB 和 HTTP)上跨不同层进行通信。

4. 基于测量的延迟与功率分析

要将智能手机作为数据密集型感知应用程序的系统平台，了解它们的延迟和功耗特性非常重要，本节将对不同的智能手机的延迟和功耗进行测量研究。测量研究为智能手机的局限性提供了更深刻的见解并在轨道上激发了几个关键的设计决策。例如，在 ORBIT 消息协议中，任务分区器，执行时间分析器，自适应性延迟与质量权衡的设计都是基于本节讨论的测量研究结果。

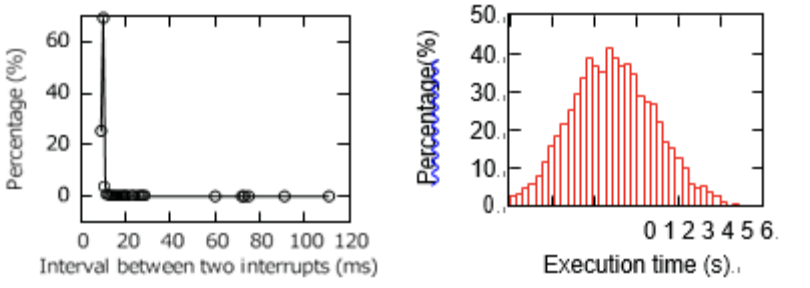
4. 1 时序精度和延迟分析

时序精度对于许多传感应用至关重要。例如，声学或地震源定位[19]通常需要重新求测传感器样品。在本节中，我们将测量安卓智能手机的软件计时器和事件时间戳的准确性，并讨论 ORBIT 设计的影响。首先，事件计时器通常用于实现恒定速率传感器采样，其精度决定了可支持的采样速率精度。其次，对外部事件进行时间戳的测定（可能由 GPS 接收器或通过 USB 连接到智能手机的传感器触发）对于许多嵌入式应用也至关重要。我们使用 LG GT540、Nexus S 和 Galaxy Nexus 进行测量，这分别代表了三种典型的中低端智能手机型号。它们运行三个版本的安卓发行版，2.1、4.0.4 和 4.2.2。这里讨论的 LGGT540 结果代表了这些测量的手机。在这里讨论的 LGGT540 的测量结果代表了这些测量手机的时序可变性水平。

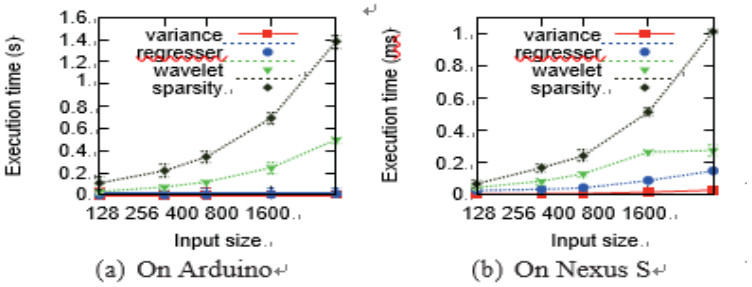
软件计时器：图 3 绘制了由软件周期计时器生成的两个间隔的分布，所需间隔为 10 毫秒，而只有安卓核心系统的服务正在运行。尽管大多数间隔接近 10 毫秒，但分布具有比较长的延迟， 最大间隔高于 110 毫秒

事件时间戳：然后，我们测量当 extBoard 的数字引脚接收脉冲信号时的时间实例之间的延迟（这将触发安卓中的 USB 中断）， 并在安卓应用程序中收到 USB 中断时。我们的测量表明，这种延迟是高度可变的，可以高达 5 毫秒。

由于这些结果表明安卓智能手机的定时精度较差，因此很难实现高恒定速率的传感器采样或精确的事件时间戳。相比之下,我们的测量表明,Arduino extBoard 的计时误差不大于 12μs,这是因为使用了硬件定时器使得中断处理效率具有可用性。



图片 3 Android 软件定时器引起的两个间隙分布。 图片 4 SIFT 算法执行时间

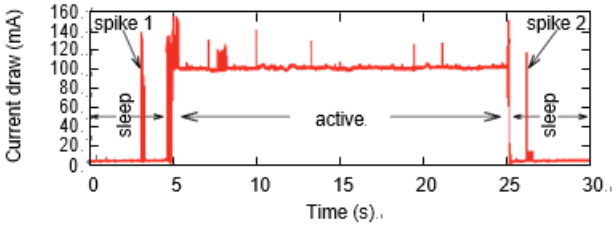


图片 5：信号处理算法的执行时间

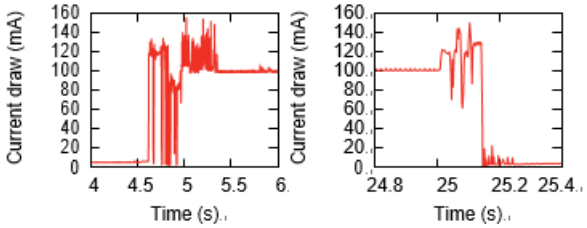
然后，我们研究了信号处理算法的执行时间。我们发现，对于固定的输入，大多数算法有相对固定的执行时间。然而，一些算法的执行时间取决于输入数据的复杂度。图 4 显示了执行时间的分布尺度不变特征的形式(筛选)用于检测特性不同的 640 x480 像素图像。这个例子表明,信号处理延迟的统计特性必须占在运行时,以确保应用程序的实时性能。任务的执行时间决定了它们的能耗，并极大地影响应用程序的实时性能。图 5 显示了 Arduino extBoard 和 Nexus S 智能手机上四种信号处理算法的执行时间与输入信号长度的关系。可以看出，extBoard 和智能手机的延迟时间分别为秒和毫秒。但是，它们的功耗是相当的，如 4.2 节所示。因此，智能手机可以以更少的能量和更短的延迟处理信号。

4. 2 功率分析

在数据密集型的传感应用中，计算通常是功耗的主要来源，因此我们重点关注智能手机的 CPU 性能。其他组件(如无线电)的功耗可以很容易地与测量的 CPU 功耗配置文件集成。我们使用 Agilent 34411A 万用表测量了几款安卓智能手机的当前电量。图 6(a)显示了三星 Nexus S 在不同处理状态下的当前状态。我们在多个智能手机模型中观察到类似的 CPU 状态转换和功耗特征。一开始，智能手机处于休眠状态，因此几乎不吸收电流(小于 5 毫安)。在第 5 秒，extBoard 会请求智能手机执行一个 FFT 算法。收到请求后，手机首先会接收一个唤醒锁，这是一种用来防止手机进入睡眠状态的 Android 机制。在第 25 秒，FFT 完成并释放唤醒锁。在手机完全醒来或进入睡眠状态之前，会有一个过渡阶段，会出现一些电量峰值。图 6(b)显示了这两个过渡阶段的扩展视图。我们将它们称为唤醒阶段和延迟阶段，分别持续约 200 毫秒和 755 毫秒。图 6(a)中还有两个峰值，是由电话和 extBoard 之间的通信引起的。由于这些峰值非常短，电流有限，它们的能量消耗可以忽略不计。基于这些结果，我们定义了四种 CPU 状态:睡眠，唤醒，活跃，延迟。



(a) Current draw in a 30-second experiment.



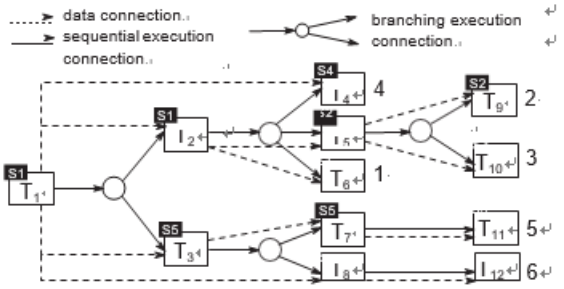
(b) Zoomed-in view of the wake-up and tail states.

图片 6：Nexus S 绘制当前配置文件

而 Arduino extBoard 有三种状态，活跃、闲置和睡眠。其在这些状态下的平均电流分别为 90 mA、66 mA 和接近于零。与智能手机,Arduino 的过渡状态很短(毫秒的精确范围内),因此他们的能耗是可以忽略不计的。

4. 3 概述

上述性能分析结果表明，不同分层(extBoard 和智能手机)的性能和延迟性能具有显著的异质性。虽然类似的测量研究已经在文献[23,5]中进行了报道，但我们公布了我们的测量结果，并表明这些发现如何为 ORBIT 设计的挑战和机遇提供重要的启示。首先，由于 Android 系统的定时精度较差，由于其硬件定时器和有效的中断处理，所以必须将高速率传感器采样和精确传感器事件时间戳等时间关键功能转移到 extBoard 中。其次，信号处理算法可能具有动态的执行时间，这需要在线分析来确保满足应用程序的关键时间期限。第三，与 extBoard 相比，智能手机具有更低的延迟和更高的能效。然而，如果 extBoard 必须保持活跃，以不断采样传感器，最好利用它的空闲时间来处理信号，这样智能手机就可以通过开启睡眠模式来节省能源。最后，过渡阶段(唤醒和延迟)以及层与层之间的数据传输在能耗和延迟方面都产生了不可忽视的花销。当将信号处理任务分配到不同的层时，必须要考虑这些重要的特性，以便在满足应用程序延迟限制的同时最小化系统总能耗。



图片 7：一个 ORBIT 应用案例

5. 设计和实现

本节将介绍 ORBIT 的设计，以实现 3.1 节所讨论的目标。

5. 1 应用程序的传递路径

ORBIT 应用管线可以用图形表示，其中节点是处理任务，边是数据流。下面的应用流程图定义了任务的执行顺序，应用于基于组件的编程模型和轨道任务划分模块。每个任务构成一个基本的感知或处理操作，如计算均值、FFT 或将图像转换为灰度。例如，一个应用管道可以是:

采样传感器(相机)→低通滤波→人脸识别→写入文件。

每个任务本身都可以由几个更小的任务组成。这种应用模式有两个优点。首先，根据任务的概念，我们可以构建每个任务的延迟配置文件(如 4.1 节所解释的那样)，并将其用于任务分区(如下一节所描述的那样)。其次，ORBIT 应用程序模型可以极大地简化应用程序开发，减少用户开发应用程序的工作量，特别是对于不熟悉嵌入式系统设计的用户。特别是，ORBIT 为应用程序开发人员提供了一个简单的 API，而不用负担底层的细节，如任务在哪里以及如何执行，以及如何在不同的层之间进行通信。

ORBIT 支持两种指定应用程序的方法。应用程序开发人员可以用 ORBIT 提供的 API 编写 Java 代码，也可以编写 XML 文件。应用程序传输通道以任何一种方式指定使用什么任务，为每个任务提供什么参数，以及如何连接任务以形成传输路径。从现在开始，我们将使用图 7 中所示的运行示例来说明如何将任务连接到构建应用程序，以及后面几节中的自动执行化和操作。示例应用程序有 12 个任务(即 T_1 至 T_{12})。

开发应用程序的主要方法是使用 ORBIT 提供的 API。ORBIT 使用 Java 注释[24]为应用程序开发人员提供了一个 API。通过使用此 API，应用程序开发人员可以将应用程序传输路径实现为 Java 类，将管道中的每个任务指定为字段，并使用 ORBIT 提供的注释来注释每个任

务。通过注释，开发人员可以指示哪个任务连接到另一个任务以及源任务中的哪个输出数据针连接到目标任务中的哪个输入数据针。例如,一个 Java 类生成应用程序的管道在图 7 中可以简单的实现如清单 1 所示,任务是实现一个在 ORBIT 数据库中的算法,指定每个任务的输入和输出参数,包括输入、输出数据和数据大小(样品)和其他算法的具体参数,如阈值,窗口大小等。@Next 注释由 ORBIT API 定义，应用开发人员使用它来连接任务并形成管道。注释@source 和@sink 用于指示管道中的源任务和接收任务。

基于注释的 ORBIT 编程模型的一个关键优势是,开发人员使用 Java 的高级功能移植到 Android 和利用 Java 语言的易用性设置应用程序的传输路径而不会背负容易出错的嵌入式编程使用低级语言所带来的风险。

```
/** import ORBIT API **/
public class Sample_application_pipeline
      extends ORBIT_pipeline_model {
@Source @Next{T_2, T_3}
private Task T_1 = new Task_1(param_1,param_2,...,param_N); @Next{T_4, T_5{2},
T_6{1}}
private Task T_2 = new Task_2(param_1,param_2, ...,param_N);
@Next{T_7,T_8}
private Task T_3 = new Task_3(param_1,param_2,...,param_N);
. . .
@Sink
private Task T_10 = new Task_10(param_1,param_2,...,param_N);
}
```

表 1：数据传输通道的伪代码表述

5.2 数据处理库

ORBIT 提供了一个数据处理算法库，从常用的学习算法和实用工具(如分类、回归、聚类、过滤和降维)到梯度优化等基本算法。使用这些经过良好测试的函数和提供的 API，开发人员可以通过 ORBIT 应用程序传输路径模型简单地连接不同的构建块，从而快速构建传感应用程序。这个库有两个主要的设计目标。首先，它是可扩展的，因此开发人员可以很容易地添加更多的算法或端口遗留信号处理库。其次，它被设计为对智能手机和 extBoard(如果应用程序使用)的资源友好的。一些算法是用 Java 语言实现的，而另一些算法是用 C++实现的，并通过 Java 本地接口(JNI)桥与其余的 ORBIT 组件连接。ORBIT 规划库设计的一个关键问题是，许多 ORBIT 应用对时间和开销有严格的要求。ORBIT 库包含两种机制来优化资源使用，同时提高编程的灵活性，即自适应延迟-质量权衡和通过多线程进行数据分区。这些机制允许程序员在智能手机平台上开发对资源友好的应用程序。

5.2.1 自适应性延迟-质量权衡

该特性的目标是缩短任务的执行时间，而不会严重影响它们的输出质量和精度。ORBIT 通过利用许多算法共有的特性来实现这个目标。也就是说，许多算法是迭代的，并且基于一个优化函数。最常用的解决优化问题的方法，包括梯度下降法和牛顿法，在轨道库中被实现为低级汇编语言。梯度下降是一个迭代的过程，在每一步(或迭代)中都是沿着负导数方向移动以减少损失。一旦损失小于阈值，算法就会停止。类似地，牛顿法使用二阶导数来选择更好的路径。因此，为了找到目标函数的最佳解决方案而进行更多迭代的任务会经历更长的执行时间，从而导致应用程序在智能手机上消耗更多的能量。缩短延迟从而降低能量消耗的一

种方法是简单地提前停止算法，例如，当第 t 步的解是令人满意的。这种方法的动机是随时算法[31]的原则。这种对轨道上迭代优化任务的提前停止机制由 stepSize、numofiteration、samplingFraction 三个参数控制，其中 samplingFraction 是每次迭代中采样的总数据的分数，用来计算梯度方向。在 ORBIT 库中，这些参数被用作每个任务的质量控制员的输入参数，同时仍然满足整个应用程序的传输路径的质量标准。

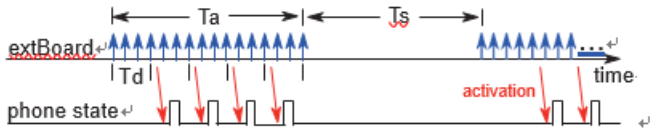
5. 2. 2 通过多线程进行数据分区

与莫特类平台相比，智能手机的一个关键优势是可以使用高速的多核处理器。如今，许多智能手机都有两个及以上的核。例如，Moto G 售价不到 110 美元，有 4 个内核。然而，尽管多核 CPU 的可用性非常高，多线程编程仍然具有挑战性。ORBIT 可以自动将长时间运行和计算密集型的任务分配到不同的线程中，并在不同的核上运行它们。这允许用户在为他们的传感应用程序设计任务结构时关注领域的特定方面。将应用程序转换为多线程有两种不同的方法。首先，我们可以将应用程序的不同任务调度到工作线程池中执行。特别是，ORBIT 可以解析任务结构，并相应地将任务调度到不同的线程。然而，许多嵌入式应用程序在信号处理管道中存在少量的“瓶颈”任务，其执行时间占总延迟的绝大部分。因此，这样的任务级多线程策略不会非常显著地降低端到端的延迟。ORBIT 采用数据驱动的多线程方法来划分这些任务。现在我们以矩阵向量乘法操作为例来说明这种方法。

许多信号处理算法(如各种变换和压缩采样)都是基于矩阵乘法的。输出 y 是矩阵乘法，$y = Ax$，其中 $A \in Z^{m \times l}$ 是对输入 $x \in Z^{l \times 1}$.的运算。假设矩阵 A 被均匀地分成子矩阵，即， $A = [A1, A2, \cdots, AK]$，其中 $AK \in Z^{m/k \times l}$.。第 k 个子任务计算 $y_k = A_k x$，最终结果为 y = [y1, y2，…],即]。第 k 个子任务也执行矩阵向量乘法。或者根据手机上可用内核的数量(可以通过 Android API 查询)来选择 k 的值。ORBIT 动态地创建计算线程，并为它们分配最高的优先级，以确保它们不会与设备上运行的其他线程争夺资源。基于这种方式，ORBIT 将所有基于矩阵的信号处理任务分配给智能手机。

许多基于矩阵运算的信号处理算法都可以从 ORBIT 的数据划分机制中受益，包括奇异值分解(SVD)、特征值分解、主成分分析(PCA)、均值和平均。这些基本算法常用于设计其他更高级的算法。但是由于 extBoard 不支持多线程，所以这些版本是用 c++实现的，不使用任何矩阵库。

多线程的一个关键设计考虑是最小化线程间通信的消耗。在 ORBIT 中，通过引用将矩阵传递给线程，每个线程计算结果的部分和不重叠(不相交)部分。换句话说，不同的线程访问相同的数据结构，但不连接数据结构的各个部分。例如，线程 k 计算 $y_k = A_k x$,，子矩阵 yk 不重叠。矩阵 $y = [y_1, y_2, \ldots y_k]$主线程也以类似的方式访问，没有线程之间的冲突或内存复制。避免轨道上的线程间通信对于处理大矩阵的数据密集型任务非常重要。



图片 8：电源管理方案

5. 3 任务划分和能耗管理

ORBIT 的一个关键设计目标是提供一个基于智能手机的节能平台。为此，ORBIT 采用了一个任务分区框架，该框架利用了不同层的功耗和延迟特性的异构性。任务划分算法在满足传感应用的处理期限的同时，使系统能耗最小化。此外，为了减少应用程序的延迟，ORBIT 实现了一个数据分区方案，该方案将基于矩阵的计算分解为多个线程，这些线程计划在不同

的 CPU 内核上执行。

5. 3. 1 电源管理模式

从第 4 节基于测量的研究中获得的关键观测数据来看，ORBIT 对不同的层采用了不同的能耗管理策略。具体地说，extBoard 在一个工作循环中运行，它在一个循环中保持 T_a 秒的活动状态和 T_s 秒的休眠状态。在活动期间，分机板以恒定的速率对传感器进行采样。一个信号段的采样时间称为采样时间，记作 T_d。活动周期包含多个采样周期。当前采样期间采集的信号段将在下一个采样期间由轨道应用程序(如图 7 所示)进行处理。T_a、T_s和 T_d的值是根据传感应用的预期系统寿命和及时性要求确定的。此外，在 extBoard 上的采样和处理常常受到严格的限制。现代微处理器还提供低功耗休眠状态，并在中断时唤醒，这可以进一步减少取样期间的 extBoard 功耗。与 extBoard 不同的是，智能手机采用了按需睡眠策略，除非 extBoard 或云消息去激活智能手机，否则智能手机将一直处于睡眠状态。图 8 显示了 extBoard 的工作周期和智能手机按需睡眠时间表。

5. 3. 2 执行时间分析器

在应用程序的生命周期内，extBoard和智能手机的电源配置一般不会发生非常显著的变化。然而，任务的延迟配置文件中可能包含一些错误，并且在部署后可能发生更改，如图4所示。为了解决这个问题，ORBIT在运行时不断地测量每个任务的延迟，并定期地重新运行任务分区程序来更新任务分区方案。特别地，我们设计了一个执行时间分析器，它可以基于运行时测量，为所有的任务建立延迟时间的统计模型。它通过使用任务执行前和执行后的系统时间来度量每个任务的执行时间。它还维护一个高斯分布模型为每个任务的执行时间,$T_i \sim N(\mu_i \sigma_2)$。该分布的参数被每一个新的度量t更新为:

$$\mu_i' = \mu_i + \frac{1}{n}.(t - \mu_i) \text{ and } \sigma_i'^2 = \frac{1}{n}((n-1)\sigma_i^2 + (t - \mu_i)(t - \mu_i')).$$

在这些模型的基础上，OTBIT采用了高秩百分位数设定执行时间。在这种方法下,我们的测量研究表明，智能手机消耗唤醒和尾部阶段有相当多的能量在满足条件下，轨道可以实现最优的分区方案时间需求统计。

5. 3. 3 顺序执行的分区

正如第5.3.1节所讨论的，extBoard具有固定的占空比，因此需要消耗相对恒定的能量。因此，ORBIT的目标是将智能手机的总能耗降至最低，这个目标可以细分为每一层的处理延迟上限。考虑一个包含n个任务的传感应用程序(用T1，…，Tn来表示)，将执行管道表示为一组连续的任务:T = T1→T2→…令li表示Ti的执行层，其中:li∈{(1,0,0)，(0,1,0)，(0,0,1)}分别表示extBoard、智能手机和云端。让$\tau_b\tau_p$、$\tau_c\tau_A$表示extBoard,智能手机,云,和结束所需的执行时间,结束整个应用程序的延迟(或部分应用程序的延迟),分别在一个采样周期。现在我们来研究一下顺序执行的任务分区问题。在技术报告[21]中讨论了分支执行的情况。任务划分问题。对于顺序执行T = T1→T2→…→Tn，任务分区器找到一个执行分配集S = {l1, l2，…，以最小化采样时间内智能手机的总能耗(表示为E)受τb≤Db,τp≤Dp,τc≤特区和$\tau_A \leqslant D_A$

处理延迟上限D_b、D_p、D_c和D_A通常是根据应用程序的时效性要求设置的，例如，传感器采样的恒定速率、在移动对象离开之前检测它的时间周期等。

如文章[23]和[6]所示，这个分区问题被定义为一个整数线性程序(ILP)，它网络带宽和CPU消耗的线性组合进行了最小化，并且满足这些资源的上限。需要注意的是，在传统的

ILP分区下，模型只需要执行时间延迟(即考虑层与层之间的张力(如网络带宽)和数据复制)。反过来，ORBIT通过在划分模型上增加了两个附加项来扩展该模型。这些术语是智能手机的唤醒和延迟时间以及每一层的(即时)功耗。另外，在执行时间分析器的帮助下，ORBIT还考虑了另外一个因素，即执行时间的不确定性。因此，ORBIT提供了一种更现实的分区模型。

5.4任务控制器

智能手机、extBoard和云端上的任务控制器(TCs)根据任务分区器计算的分配来执行整个传感应用程序。图2所示为TCs与轨道上其他组件之间的相互作用。

5.4.1智能手机的任务控制器

智能手机的任务控制器被设计成一个Android的后台服务器，它可以控制任务的执行，并与extBoard和云端进行通信。当ORBIT应用程序发射时，智能手机任务控制器创建任务的实例，并为所有输入和输出分配缓冲区。在这个初始化阶段之后，任务控制器检查分区分配并开始执行第一个任务。当智能手机不执行任务时，它会切换到睡眠状态以节省能源。当任务执行需要返回到智能手机时，会发送一条通知消息来唤醒智能手机并激活它的任务控制器来执行分配给它的下一个任务。智能手机的任务控制器还不断更新任务的元信息(比如执行时间)以及分支优先级。

我们的测定结果表明智能手机在唤醒和延迟阶段会消耗相当大的能量。我们优化了任务控制器的设计，使之能够在最短的时间内完成一项任务——让智能手机醒来或者让智能手机睡眠，只要没有更多的任务需要运行。任务控制器在智能手机上执行任务T后，检查是否有任何任务分配给以T的输出作为输入的其他两层。如果有，任务控制器的消传递协议(在5.5节中具体提到)会将T的输出数据发送到另一层。这使得其他层可以在不重新激活智能手机数据的情况下运行任务，避免额外的唤醒和额外的能耗。然而，这种设计的副作用是，如果应用程序有分支，则传送到另一层的数据可能不被使用。然而，典型的信号处理传输路径可能包含有限数量的分支。

5.4.2 extboard和云任务控制器

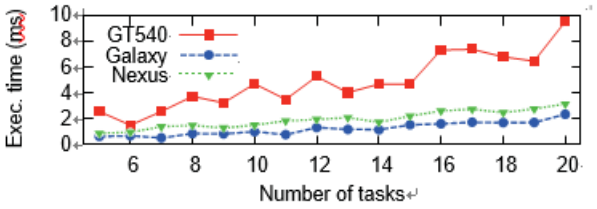
extBoard 任务控制器不断检查来自智能手机的消息是否到达。当它从智能手机接收到启动任务执行消息时，它开始执行任务分配中的第一个任务。在开始采样任务的情况下，extBoard创建一个定期计时器来控制采样。定时器中断处理例程从ADC中读取传感器样本，对其进行时间戳，然后将其插入循环缓冲区。这个过程只涉及几个指令，并且经过优化以减少中断处理延迟。一旦采样任务获得了输入参数中指定的采样数量，任务执行就会根据执行树T继续执行采样之后的任务。

云端的任务控制器被实现为一个Linux守护进程，用来检查来自一个或多个智能手机的到达消息。有两种类型的任务在云中运行;任务分区器分配的计算密集型任务，以及从多个轨道节点获取输入数据的任务。云中的任务T完成后，云任务控制器将T的输出发送给所有需要输出的智能手机。如果任何智能手机处于睡眠模式，它们在接收到云消息时就会醒来。云任务控制器和智能手机任务控制器一样，不断更新任务元信息(例如执行时间)和分支优先级。这确保将新的元信息用于任务分区。

5.5 ORBIT信息传输协议

ORBIT支持通过有线和无线路径连接到各种外部传感器的传感应用程序，以及许多内

置的智能手机传感器。然而，确保合适的设备进行协同工作将会是一项繁重的工作，特别是当一个应用程序需要使用不同的传输通道和数据格式集成多个传感器时。当应用程序必须在多层体系结构中的不同层之间传输数据时，就会更加繁琐了。本节介绍了ORBIT信息传递协议(OMP)，它通过抽象的方式将extBoard、智能手机和云平台之间的职责进行了分离，从而简化了数据传输的方式。



图片9：任务分区程序的执行时间

智能手机通过两个USB(USB配件和USB主机)和蓝牙与外围设备通信。Android蓝牙和USB通信只提供一个字节传输通道。这个通道既不提供传递的保证，也不支持信息的概念。另一方面，如5.1节所述，在以任务结构构成的传感应用程序中，不同任务之间的通信单元是一个消息。为了实现这一目标，ORBIT消息传递协议补充了简单的信息帧和信息校验，从而提供了更健壮的通信通道。另外，ORBIT还提供了长报文的分段和排队机制。为了使与层之间的通信具有同质性，ORBIT在HTTP协议之上为云通信实现了相同的信息传递协议。这促进了一个模块化框架的形成，并且允许开发人员专注于编写最小的数据传输片段和传感器特定的代码。

6. 微基准测试

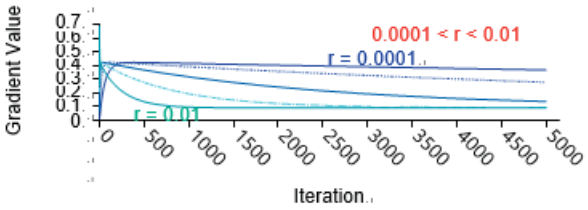
在本节中，我们将评估ORBIT的总体内存和CPU使用情况，以及在线任务分区所带来的内存消耗。我们还评估了多线程的模式在减少任务处理延迟方面的效果。

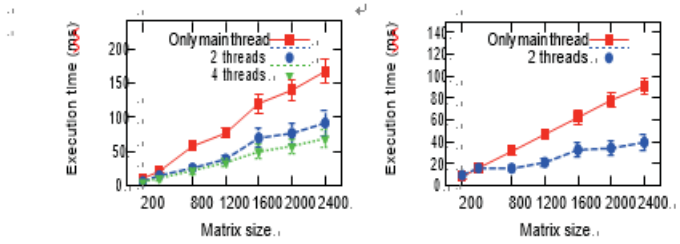
智能手机的CPU和内存占用：我们将测量ORBIT的CPU和内存占用。我们使用Android utility应用程序，系统监视器，来测量CPU和内存的使用情况。我们选择不同的应用程序来运行ORBIT。ORBIT作为一个应用程序运行的时候，它的CPU利用率可能会根据智能手机的硬件、安卓操作系统的版本和其他运行在智能手机上的应用程序而有所不同。我们通过运行任务时增加的CPU利用率来度量ORBIT的CPU占用情况。我们的测量显示，ORBIT的CPU占用率从10%到15%不等。在沉默期间，内存使用量约为22.5 MB，但达到了最大值。感知应用程序的33.8 MB内存，因为堆空间是为处理任务动态定位的。ORBIT的总大小只有2.84 MB。

在线分区的消耗:正如第5.3节所讨论的，任务分区器会在线运行以适应任务的可变执行时间。图9显示了任务分区器在不同智能手机上的执行延迟与任务数量的关系。我们可以看到，当有20个任务时，任务分区器花费的时间少于10毫秒。此外，仅在任务执行时间发生重大更改时才调用任务分区程序。因此，在线任务分区不会在智能手机上引入显著的运行时间消耗。。

延迟-质量权衡:在第5.2.1节中，我们讨论了如何优化算法，以便在质量和延迟之间进行适

当的权衡。图10为不同步长r和迭代次数下梯度下降算法的收敛情况。如图所示，随着迭代次数的增加，梯度值逐渐减小，直到最终收敛到解决方案。步长越大，梯度收敛越快。然而，对于每个步骤大小，在特定的迭代之后，降低的速度会变慢，这意味着任务不会从更多的迭代中受益。因此，与作为输入参数提供的迭代次数相比，梯度下降通常可以在更少的迭代次数中找到足够好的解决方案，从而允许轨道在不丢失显著精度的情况下提前停止。支持向量机、线性回归和k均值聚类等算法可以从这一特征中获益。该特性不仅为应用程序管道中的每个任务选择更好的参数值提供了见解，而且为ORBIT提供了电源和系统的机制来终止任务，同时仍然将结果保持在预期的精度范围内。





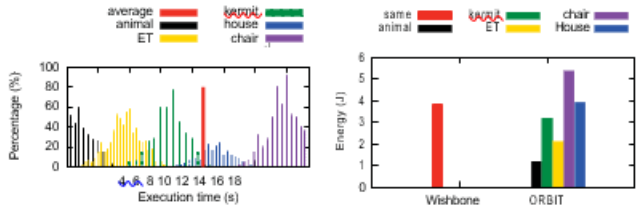
(a) Moto-G (4 cores)　(b) Galaxy Nexus (2 cores)

图片10：延迟-质量管理

图片11：智能手机多线程减少了对计算密集型任务的处理

数据分区和多线程的效果:如5.2.2节所述，智能手机任务管理器可以将计算密集型任务划分为多个线程，以减少处理延迟。图11显示了矩阵向量乘法任务y = Ax在两种不同智能手机上的性能增益，Moto-G采用四核处理器，Galaxy Nexus采用双核处理器。在这个例子中，向量$x \in Z_{l\times 1}$是输入信号，矩阵$A \in Z^{m\times l}$是计算矩阵。l的固定值为2000个数据样本，m随着不同的操作而变化(图中的横轴)。m的值越大，表示计算的数据量越大。结果表明，当任务被划分为2个线程时，Mote-G的计算延迟平均降低了44.7%，Galaxy Nexus的计算延迟平均降低了36.2%。当计算被分为4个线程时，Mote-G的计算延迟平均值也减少了56.1%。

从图中可以看出，对于更大的矩阵(数据密集型的计算)，多线程可以更大程度上降低计算延迟，这与我们的设计目标是一致的。这个图的另一个重要结果是Moto-G中的4个线程并没有比2个线程有明显的改进。这是因为，一旦将计算划分为两个线程，问题的大小就减少了一半。因此，当每个线程被进一步分成两个新线程时，它只会影响一个更小的问题，因此减少的计算延迟也更小。这与多线程对较小的问题提供较少的改进是一致的。



(a) Execution time distribution of SIFT　(b) Energy consumption of SIFT for different input images

图12：数据依存性的算法

数据依存性的影响:ORBIT的一个显著特点是在对任务的能耗和划分进行模型建立时，考虑了输入数据的大小和内容等信息。与之相反，在传统的任务建模和分区方案中，时间延迟是在脱机状态下测量的，其平均值通常被假定为时间延迟，而不考虑执行时间中观察到的差异。然而，我们的测量研究表明，对于具有不同输入大小和内容的数据依赖性算法，执行时间可能会发生显著变化。现在我们用几个例子来说明数据依赖对系统能耗的影响。图12(a)为具有不同维数和SIFT特征数的输入图像的SIFT算法的执行时间分布。由图12(b)可知Wishbone方法下SIFT算法的能耗估计与ORBIT方法的能耗估计的差异。由于Wishbone没有考虑输入数据之间的差异，所以将脱机测量的平均值作为执行时间。因此，当SIFT对图像的执行时间接近平均值时，如对房屋图像，两种方法估计的能量都是相似的。然而，当图像的执行时间小于平均时间时，如ET、kermit和动物图像，ORBIT的能耗估计值的性能优于Wishbone。另一方面，如果执行时间比平均执行时间长，例如，椅子图像，虽然ORBIT估计的能量比Wishbone大，但ORBIT可以更接近真实值的估计。因此，ORBIT为数据密集型算法的执行时间和能耗建模提供了一种更加现实的方法。

7. 案例研究

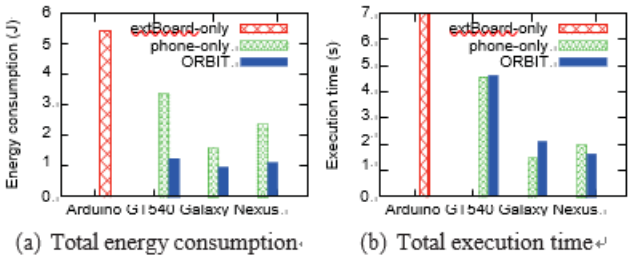
Application	Event Timing	Multi Camera 3D Reconstruction	Robotic Sensing
Script Length	27	20	35
Number of Tasks	7	10	11
Sensors	GPS, Geophone	Camera, GPS	IR, Camera, Ultrasound
Tiers	extBoard, smartphone	extBoard, smartphone, cloud	extBoard, smartphone
Data Fidelity	100Hz	640*480px	5fps

表2：基于ORBIT的应用程序

演示ORBIT应用程序脚本以及轨道作为一个平台的通用性和灵活性,我们设计并开发了三个不同的嵌入式传感数据密集型应用程序(表1)。每个应用程序演示了不同方面的轨道不同数量的任务在任务结构,不同的传感器的使用,层之间的应用程序分区的数量,和应用程序的数据保真度要求。我们的目标是展示该平台的功能和有效性，而不是展示新的应用程序。由于空间的限制，这里只介绍了事件定时和多摄像机三维重建的案例。机器人传感案例研究可在技术报告[21]中找到。

7. 1事件定时

这个应用的功能是估计一个声波/地震事件的到达时间，这是许多声学/地震监测应用的基础，如分布式事件定时[19]和震源失配。地震事件源定位要求事件时间精确到毫秒以下，节点之间的时间同步控制在几微秒内。首先对输入信号进行均值去除和带通滤波预处理。然后对滤波后的信号进行小波变换。基于小波系数计算信号的稀疏性和粗略的到达时间。这个应用程序需要100Hz的采样率。在早期地震探测的情况下，系统的响应时间必须在几秒左右。以下部分给出了评估结果。



(a) Total energy consumption (b) Total execution time

图13:各种分区方案的结果

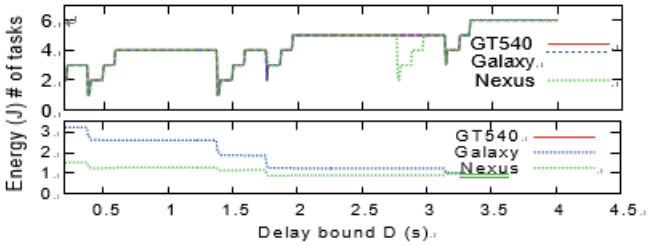
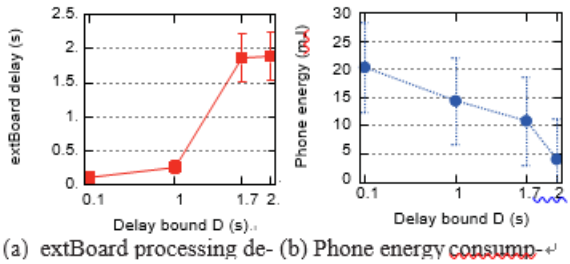


图14:延迟边界设置对任务分配和总能耗的影响。

任务分区器的有效性:我们首先评估5.3.3节中提出的任务分区算法的有效性，通过比较以下分区方法:仅考虑extBoard、仅考虑phone、贪心算法。贪心算法会把尽可能多的处理任务分配给extBoard，只要能够得到延迟限制的支持。图13显示了使用1.8 s的延迟边界D的分区方法的任务分区结果。除extBoard方法外，extBoard进程延迟满足此边界。图13(a)和图13(b)描绘了估计的总能耗和总执行时间(即一个轨道节点在不同分区方式下的一个执行周期。由于分机速度慢、功耗低，不能满足时延限制，是能耗最大的设备。我们的分区方法("ORBIT")在测试的智能手机上实现了最低的能耗。

然后我们来评估延迟约束D对任务分配和智能手机能耗的影响。图14的顶部显示了分配给extBoard与d的任务数量。我们可以看到，对于较大的d，任务分区器通常分配给extBoard的任务更多。这与我们在5.3.3节中的分析是一致的。然而，在图14的顶部我们可以看到一些水滴。例如，当D从1.37秒到1.38秒，extBoard任务的数量从4个下降到1个。这是由于一个计算密集型任务取代了前四个轻量级任务，以增加extBoard的CPU利用率和减少智能手机的能源消耗。图14的底部部分是总能耗与D的对比图。图中显示总能耗随着D的增加而减少，这与我们的分析是一致的。

测量执行时间和能量消耗:基于接收到的任务划分结果，我们使用一个Nexus ORBIT节点在实时传感器读数上运行应用程序。图15描绘了测量的extBoard处理延迟和智能手机能耗与指定延迟边界的关系。注意，智能手机处理延迟小于5毫秒的所有设置延迟边界。因此，在extBoard中处理延迟占主导地位。从图15(a)可以看出，总是满足指定的延迟边界。此外，extBoard处理延迟的能耗随着延迟边界的增加而增加，这证明了extBoard 对CPU时间进行了有效利用。从图15(b)可以看出，智能手机的能耗随着延迟边界的增大而减小，这与我们的分析是一致的。



(a) extBoard processing de- lay (b) Phone energy consump- tion

图15:测量的extBoard处理延迟和智能手机能耗与延迟边界

暂存板的占空比和寿命:根据测量的功耗，我们计算了4节D-电池(容量:1.2×10^4毫安时)与暂存板在不同时延限制下的占空比下的预计节点寿命。当占空比为100%时，预计的生存时间为5.8天，当占空比为20%时，节点可生存长达2个月。如图15(b)所示，智能手机的能耗为数十毫焦耳，而当占空比为100%时，extBoard的能耗约为1焦耳。由于extBoard和智能手机的有源功率是相当的(参见第4节)，所以当占空比较大时，extBoard的能耗占主要地位。在这种情况下，智能手机的主要作用是帮助满足严格的延迟边界，并且对于不同的延迟边界，节点生存期是相似的。然而，当占空比为20%时，如果延迟边界从0.1秒增加到2.0秒，则寿命可延长18.4%。然而，在智能手机的帮助下，ORBIT节点可以满足严格的延迟界限，这是许多需要连续传感器采样的传感应用成功的关键。

7. 2多摄像头三维重建

最后一个案例研究的灵感来自于光旅游[27]并涉及到机会感知，其中配备智能手机的机器人捕获基于位置的图像，并协作重建一个3D结构。与前面的两个案例研究相比，这个应用程序是跨三层划分的。捕获的图像在手机上进行部分处理，其余的处理以及分布式任务被卸载到云服务器。一旦一个图像被处理，机器人会被引导到一个新的地点去捕捉一个新的图像。除了CPU，我们还考虑了这个案例研究中的无线电功耗。图16展示了该应用程序的任务结构。云服务器由Sun Ultra 20工作站模拟。

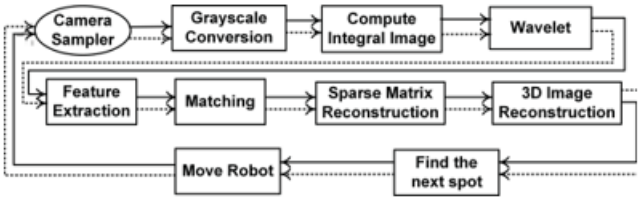


图16:多摄像头三维侦察应用的框图

任务分区器的有效性:在本案例研究中，智能手机与云服务器之间的通信延迟和输入数据的复杂性会影响分区结果，因为感知应用中加入了云层和更复杂的输入数据。我们通过比较轨道与纯电话和纯云基线来评估任务分配算法的有效性。在图17(a)和图17(b)中，比较了两种不同输入图像的情况:a) 房子图像:复杂度较低的较大图像(以像素数表示);b) kermit图像:复杂度较高的较小图像。这两种情况下分区分配的不同之处在于SIFT任务的分配。房子的图像结果表明,它更节能运行在电话里筛选,因为:1)图像不太复杂,因此筛选运行更快,因此导致应用程序消耗更少的能量,和2)它将消耗更多的能量传输的大型图像到云筛选处理。对于kermit图像，在云端执行SIFT算法会更节能，因为它是一个更小的图像，具有更多的SIFT特性。因此，在这两种情况下，轨道提出了最节能的分区。此外，这一结果表明，ORBIT不

仅将数据处理任务的执行时间作为输入大小的函数，而且还作为输入内容的函数。以前的任务划分方法[23,5]常常不能解决这两个影响因素。

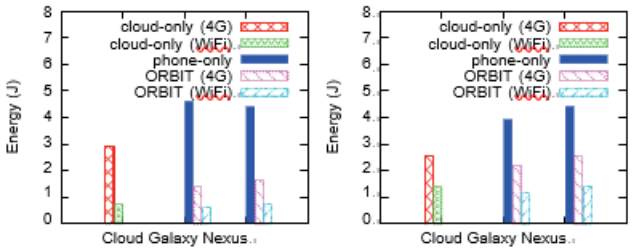


图17:各种分区方案的结果

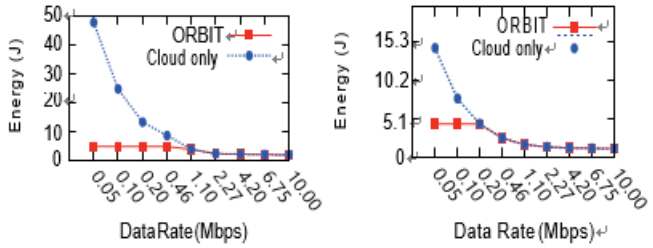


图18:数据率对任务分配和总能耗的影响

网络吞吐量的影响:然后我们评估通信数据速率R对任务分区的影响。图18显示了Nexus s的智能手机能耗。我们可以看到，在这两幅图中，能耗都随着R的增加而降低。原因是，当轨道通过更快的连接到云端时，智能手机保持活跃所需的时间更短。该图还表明，ORBIT的能量消耗与纯云的能量消耗是一致的。这是因为随着数据速率的增加，将任务转移到云端进行计算会变得更加节能。从图18(a)和图18(b)可以看出，在不同的上传延迟下，由于图像大小的不同，导致了云方案的收敛速度不同。总的来说，这些结果表明，ORBIT可以利用网络延迟和任务负载的不同特性来最小化系统的能耗。

7. 3讨论

这些案例研究证明了ORBIT设计的普遍性。特别是，这两个示例应用程序在任务结构、任务的计算强度、延迟要求、输入数据和任务分区所涉及的层上存在显著差异。总的来说，与基准方法相比，ORBIT方法可以节省高达50%的能源。

从这两个案例研究中得出的一个有趣的结论是，系统功耗具有很高的概率。然而，这样的运行时动态是未知的ORBIT在设计时。因此，按照最坏的情况来划分任务。一个可能的改进是提供轨道运行时关于系统性能的反馈。这将使轨道优化任务的布线和任务的优先次序，以减少电力消耗。由于ORBIT具有灵活的任务划分和调度框架，所以它很容易支持这种运行时的调整。

实例研究2表明，层与输入数据之间的通信延迟对分区结果有重要影响。例如，当3G或Wi-Fi网络条件较差的情况下，如果应用程序施加了严格的延迟限制，任务可能不会被转移到云中。此外，图像处理这样的高级感知任务通常具有高度可变的执行时间，这验证了ORBIT所采用的在线任务划分框架的合理性。

8. 结论与未来要做的工作

本文介绍了基于智能手机的数据密集型嵌入式传感应用平台ORBIT。ORBIT采用分层架构，其中智能手机可选内嵌节能外围板和云服务器。通过充分利用多层系统的功率/延迟特性的异构性，ORBIT将系统的能量消耗降到最低，但会受到上界处理延迟的影响。ORBIT还集成了一个支持高级Java注释应用程序编程的数据处理库。该库的设计方便了嵌入式应用程序的资源管理，并通过自适应延迟-质量权衡和多线程数据分区机制提供了编程灵活性。轨道评估通过几个基准和三个案例研究:地震感知，多摄像机三维重建和视觉跟踪使用轨道机器人。本文介绍了前两个案例的研究结果，最后一个案例的研究结果可以在技术报告[21]中找到。这种广泛的评价证明了轨道设计的普遍性。此外，我们的结果表明，ORBIT可以节省高达50%的能源消耗比基线方法。未来的计划包括使用额外的嵌入式传感应用程序对ORBIT进行评估，扩展其数据处理库，并将其用于大规模部署。

致谢

作者感谢shepherd博士Jack Stankovic和匿名审稿人为本文提供了有价值的反馈。这项工作得到了美国国会的部分支持。美国国家科学基金会资助的项目有:CNS-1218475、OIA-1125163和CNS-0954039 (CA- REER)

9．参考文献

[1] Arduino board. http://www.arduino.cc.
[2] R. K. Balan, M. Satyanarayanan, S. Y. Park, and T. Okoshi. Tactics-based remote execution for mobile computing. In MobiSys, 2003.
[3] D. Chu, N. D. Lane, T. T.-T. Lai, C. Pang, X. Meng, Q. Guo, F. Li, and F. Zhao. Balancing energy, latency and accuracy for mobile sensor data classification. In SenSys, 2011.
[4] Cloud robotics.
http://goldberg.berkeley.edu/cloud-robotics/.
[5] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu,
R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In MobiSys, 2010.
[6] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu,
R. Chandra, and P. Bahl. Maui: Making smartphones last longer with code offload. In Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM.
[7] M. Faulkner, M. Olson, R. Chandy, J. Krause, K. M. Chandy, and
A. Krause. The next big one: Detecting earthquakes and other rare events from community-based sensors. In IPSN, 2011.
[8] J. Flinn, S. Park, and M. Satyanarayanan. Balancing performance, energy, and quality in pervasive computing. In ICDCS, 2002.
[9] Floating sensor network. http://float.berkeley.edu.
[10] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and

D. Estrin. Emstar: A software environment for developing and deploying wireless sensor networks. In USENIX Annual Technical Conference, 2004.
[11] E. Guizzo. Robots with their heads in the clouds. IEEE Spectrum, 48(3):16–18, 2011.
[12] Gumstix. https://www.gumstix.com.
[13] Innovative monitoring systems help researchers look inside volcanos. www.cse.msu.edu/About/Notable.php?Nid=423.
[14] IOIO for Android. www.sparkfun.com.
[15] Y. Ju, Y. Lee, J. Yu, C. Min, I. Shin, and J. Song. Symphoney: a coordinated sensing flow execution engine for concurrent mobile sensing applications. In SenSys, 2012.
[16] S. Kang, J. Lee, H. Jang, H. Lee, Y. Lee, S. Park, T. Park, and J. Song. Seemon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. In MobiSys, 2008.
[17] S. Kang, Y. Lee, C. Min, Y. Ju, T. Park, J. Lee, Y. Rhee, and J. Song. Orchestrator: An active resource orchestration framework for mobile context monitoring in sensor-rich mobile environments. In PerCom, 2010.
[18] LG Optimus Net. http: //www.gsmarena.com/lg_optimus_net-4043.php.
[19] G. Liu, R. Tan, R. Zhou, G. Xing, W.-Z. Song, and J. M. Lees. Volcanic earthquake timing using wireless sensor networks. In IPSN, 2013.
[20] Marvell sheevaplug. www.plugcomputer.org.
[21] M.-M. Moazzami, D. E. Phillips, R. Tan, and G. Xing. A smartphone-based system platform for embedded sensing applications. Technical Report MSU-CSE-13-11, Dept. CSE, Michigan State University, 2013. http://www.cse.msu.edu/ publications/tech/TR/MSU-CSE-13-11.pdf.
[22] Nasa phonesat project. http://open.nasa.gov/plan/phonesat/.
[23] R. Newton, S. Toledo, L. Girod, H. Balakrishnan, and S. Madden. Wishbone: Profile-based partitioning for sensornet applications. In NSDI, 2009.
[24] Oracle. Java annotations. http://docs.oracle.com/ javase/tutorial/java/annotations/.
[25] M.-R. Ra, B. Liu, T. F. La Porta, and R. Govindan. Medusa: a programming framework for crowd-sensing applications. In MobiSys, 2012.
[26] Raspberry pi. http://www.raspberrypi.org.
[27] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: Exploring photo collections in 3d. In ACM SIGGRAPH, 2006.
[28] J. Sorber, N. Banerjee, M. D. Corner, and S. Rollins. Turducken: Hierarchical power management for mobile devices. In MobiSys, 2005.
[29] R. Tan, G. Xing, J. Chen, W. Song, and R. Huang. Quality-driven volcanic earthquake detection using wireless sensor networks. In RTSS, 2010.
[30] Y. Yan, S. Cosgrove, V. Anand, A. Kulkarni, S. H. Konduri, S. Y. Ko, and L. Ziarek. Real-time android with rtdroid. In MobiSys, 2014.
[31] S. Zilberstein. Using anytime algorithms in intelligent systems. AI magazine, 17(3):73, 1996.