无源系统轻量级动态程序运行环境

李松璠[1)] 鲁 力[1)] 李 杰[2)]

[1)](电子科技大学计算机科学与工程学院 成都 611731)

[2)](铜仁市公安局科技信息通信处 贵州 铜仁 554300)

摘 要 无源系统能量紧缺导致系统难以长时间处于活动状态下工作,时常因能量不足而掉电.为保障程序完整执行,程序运行环境通常先让系统在低功耗状态从环境采集能量,然后切换至活动状态执行一个程序片段,循环往复直至完成.根据执行程序的不同策略,现有运行环境可分为两大类:一次执行和断点执行.前者在充能足够后一次性完整执行一个程序,适用于感知等轻量级程序;后者将程序拆分成若干片段,可间断执行,适用于长时间计算类程序.但在实际应用中,经常会出现以上两类程序混合执行的情况,使用任何现有运行环境都难以有效支持不同类型的程序混合执行,导致系统吞吐量下降.本文提出动态程序运行环境,根据不同的程序类型动态调整程序执行策略,全面支持现有两类程序高效率混合执行.我们优化了动态加载机制,实现了轻量化设计,降低了动态运行环境工作开销.我们在 Intel WISP 平台上对动态程序运行环境进行了实验评估,实验结果显示在多类程序混合执行时,动态运行环境的吞吐量比一次执行运行环境提高 102.8%,比断点执行运行环境提高 34%.

关键词 无源系统;运行环境;吞吐量;程序执行;能量效率

中图法分类号 TP393 **DOI 号** 10.11897/SP.J.1016.2021.01326

A Dynamic and Lightweight Runtime for Battery-Free Systems

LI Song-Fan[1)] LU Li[1)] LI Jie[2)]

[1)](*School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu* 611731)

[2)](*Department of Technology Information and Communication, Tongren Municipal Public Security Bureau, Tongren, Guizhou* 554300)

Abstract Due to the scarce energy in battery-free systems, the system cannot be active for a long time and frequently suffers from power outage. In order to protect programs executing completely, a program runtime typically first collects enough energy from the environment in low-power mode and then executes a part of programs in active mode, repeating until all programs are completed successfully. According to the strategy of executing the program, existing runtimes can be divided into two categories, burst execution and checkpoint execution. The former performs a complete program at a time after charging enough, and adapts to lightweight programs like sensing. On the other hand, the latter splits the program into several segments and can be executed intermittently, which is suitable for long-term computing programs. In practical applications, however, the above two types of programs exist simultaneously and are often mixed in program scheduling. Existing runtimes cannot effectively support mixed execution on different types of programs, resulting in a decline in system performance in terms of task throughput. This paper proposes a lightweight runtime that dynamically adjusts the program execution strategy according to different program categories, and fully supports the mixed execution with high efficiency. In order to achieve this, our basic idea is to assign a program description block (PDB) to each

收稿日期:2019-08-09;在线发布日期:2020-03-12.本课题得到国家自然科学基金面上项目(61872061)、国家重点研发计划(2017YFB1003000)资助.**李松璠**,博士研究生,主要研究方向为无源系统、能量管理、低功耗技术.E-mail:songfanli@std.uestc.edu.cn.
鲁 力(通信作者),博士,教授,中国计算机学会(CCF)会员,主要研究领域为物联网智能终端、无线系统及安全、RFID 技术.E-mail:luli2009@uestc.edu.cn.**李 杰**,学士,三级警督,主要研究方向为信息系统和网络.

program to indicate the type of program execution. Before executing each program, the runtime first determines the program type and then dynamically configures the corresponding execution strategy, and chooses the best way to execute the corresponding program. In doing so, however, it is quite challenging to implement such design in battery-free systems due to the scarce energy. To make the runtime dynamic, we have to redesign the runtime kernel that is currently static in existing battery-free systems, but the dynamic optimization will bring extra high energy overheads to the system. To achieve a lightweight design, we deeply investigate the root reason of the extra energy overheads. Essentially, the dynamic runtime needs to be aware of dynamicity of the harvested energy. However, we observe that energy sensing operations drain the major power consumption in the dynamic optimization, where the system has to keep detecting how much energy is available in the system. Since battery-free systems often exhaust and recharge their energy reservoirs in milliseconds, the frequent energy sensing operations result in high energy overheads to the system. To address this problem, we construct the dynamic runtime based on our proposed lightweight energy sensing component. Further, we build a system throughput model to describe the relationship between task throughput and different programs in the dynamic runtime, which shows a theoretical analysis to the throughput. Moreover, we discuss the compatibility of the dynamic runtime with existing programs and make efforts to design a program framework to support the mixed execution to all of existing programs. Finally, we implement our design on the Intel WISP platform and conduct comprehensive experiments to evaluate the improved performance in particular task throughput. The results show that the throughput of the adaptive runtime is 102.8% higher than that of the burst execution runtime, and 34% higher than that of the checkpoint execution runtime.

Keywords　battery-free system; runtime; throughput; program execution; energy efficiency

1　引　言

据市场研究公司 Machina Research 预测，物联网设备数量将在 2025 年增长到 270 亿①. 如何给这些终端提供能源是物联网面临的关键挑战之一. 由于电池技术发展还无法满足物联网终端设备的寿命需求，更换电池将带来高昂的维护开销. 因此，从环境光、电、温、振中获取能量的技术受到了广泛关注. 环境获能技术与现有物联网终端相结合，衍生出一类特殊的物联网嵌入式系统，称为无源感知和计算系统(下文简称为无源系统). 为了降低生产和维护成本，无源系统使用能量采集模块代替电池为设备供能，在收集足够能量后执行感知、计算和通信等任务. 现有主要的无源系统包括可计算式 RFID 标签如华盛顿大学/英特尔的 WISP[1] 和马萨诸塞大学的 MOO[2]，以及一些能量自治传感节点如加州大学埃尔文分校的 Everlast[3]，克莱姆森大学的 UFoP[4] 等. 与电池供电系统(如无线传感网)相比，无源系统具有轻巧便携、能量自治、独立免维护运行等突出优势，能以各种形式深度嵌入到物理环境中，有望在物联网中得到广泛应用.

无源系统的程序运行环境(runtime)是一种在能量短缺情况下保护程序完整执行的系统机制. 在介绍运行环境之前，让我们先了解无源系统的能量短缺问题. 由于嵌入式环境对无源系统物理尺寸的要求，无源系统一般采用微型能量采集模块，能量来源单一且转换效率低下，导致无源系统获取能量的能力低下. 另一方面，由于没有专门为无源系统设计的低功耗器件和架构，现有无源系统的感知、计算和通信等负载直接采用了有源系统所采用的设计，导致功耗过大，超过了微型采能模块的供能上限. 以 WISP 为例，其负载功耗常高于能量采集器输出功率 3～6 个数量级[5]. 以上因素导致了无源系统能量供不应求，程序执行频繁因中途掉电而失败，造成能量浪费进一步恶化能量短缺问题. 为了保障程序执

① Machina research: IoT global forecast & analysis 2015-25. https://machinaresearch.com/report/iot-global-forecast-analysis-2015-25/2016,08,05

行,程序运行环境根据当前系统能量状况规划程序间歇性执行.所谓间歇性是指,系统电量较少时,运行环境先使系统进入低功耗模式充电等待;当电量充足时(高于某个阈值),程序才开始执行.程序运行环境通过间歇性程序执行提高程序执行的成功率.

无源系统中现有程序运行环境可分为两类.第一类为一次执行运行环境[6-7],基本思路为充能足够后一次性将某程序完整执行.由于无源系统储能模块容量限制,一次执行运行环境一般适用于能量需求量少、完成时间短的程序(如温度测量),难以支持长时间的程序运行.第二类是断点执行运行环境[8-11],基本思路为在低电量时将此时程序执行的状态以断点的形式保存,当系统再次激活时读取断点继续执行,直至程序完成.断点执行运行环境一般适用于可设置断点的计算程序,难以支持片外传感器感知和外部通信.总之,现有的两类程序运行环境分别面向不同类型的程序,选择使用哪一类运行环境由具体程序类型和应用需求决定.

现有程序运行环境难以满足许多实际应用中的程序执行需求.无源系统常需要执行多个任务,其中任务程序可能是一次执行的感知任务,也可能是断点执行的计算任务,这就容易出现两类程序在同一运行环境下混合执行的情况.例如,系统先从温度传感器读取数据(一次执行),然后对温度数据进行CRC计算(断点执行),最后传给通信模块发送到接收器(一次执行).程序在不匹配的运行环境下执行会导致程序执行效率下降,进而降低系统吞吐量.我们通过实验发现,一次执行运行环境支持断点类程序会导致系统吞吐量平均下降43.5%,用断点执行运行环境支持一次类程序导致系统吞吐量平均下降27.7%.

本文将介绍动态程序运行环境设计,根据不同的程序类型动态调整程序执行策略,有效率地支持现有两类程序混合执行,如图1所示.基本思路是为每个程序分配一个程序描述块指示所属程序执行类型.运行环境执行程序前,先判断程序类型,然后动态配置相应的执行策略,选用最佳的方式执行相应程序.

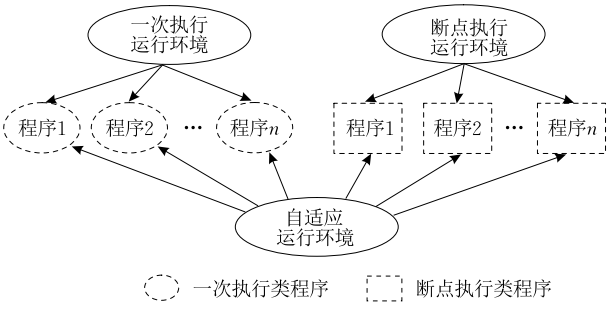


图1 本文提出动态程序运行环境支持目前无源设备中全类型程序混合执行

在无源系统能量短缺的背景下,如何实现动态又轻量级的程序运行环境是本文的主要难点.为了实现运行环境动态化,我们需要重新设计程序执行机制,使执行策略随不同程序类型进行动态调整.相比之下,现有运行环境是静态的,在系统初始化时已经配置完毕,执行策略无需在程序运行过程中改变.动态机制要求运行环境中各个模块单元根据不同程序类型动态配置,为多样性的程序执行提供可靠支撑.但动态机制势必会带来额外的开销,加重无源系统的能量问题.如何在保证动态程序执行的情况下,又不过分降低系统效率是研究的难点.

我们观察到能量探测单元是动态程序运行环境高功耗的根本原因.现有运行环境基于两类能量探测单元.其一是硬件探测(使用一个硬件探测单元探测电量是否达到某个阈值),优点是低功耗,缺点是阈值固定仅适用于静态程序运行环境.其二是软件轮询[6,12](软件控制数模转换器周期性探测系统电量,然后判断电量是否高于某个阈值),优点是阈值可调,缺点是高功耗.例如,WISP平台中数模转换器用于实现软件轮询的功耗占了系统总功耗近一半[5].总之,现有运行环境所使用的能量探测方法难以满足轻量级动态程序运行环境的设计要求.动态运行环境需要低功耗且阈值可调的能量探测单元.

我们基于自研的低功耗数字可调能量探测单元[5]构建了动态程序运行环境.该能量探测单元属于硬件探测,但它包含一个可以受控于CPU的数字接口,其能量阈值是数字可控的.运行环境会根据不同任务类型动态地配置能量探测单元的阈值,以支持不同程序执行.在后文中,我们将介绍动态程序运行环境的设计方法、系统框架和重要流程.此外,我们建立了系统吞吐量模型用于描述动态运行环境下吞吐量与不同程序之间的关系.最后,我们讨论了动态运行环境与现有程序的兼容性问题.

我们将动态程序运行环境实现在WISP平台上,并进行了实验评估.实验包括单程序评估和程序集评估两部分.单程序评估用于验证动态运行环境对无源系统中常见程序的支持效果.程序集指运行环境以不同顺序混合执行各类程序.程序集评估旨在测量动态运行环境对无源系统性能的提升情况.

我们以完成程序的吞吐量作为评价运行环境性能的指标.实验结果表面,动态程序运行环境在单程序评估时,吞吐量与对应原运行环境结果相似;在执行程序集时(混合执行),动态运行环境的吞吐量较一次执行运行环境提升 102.8%,较断点执行运行环境提升 34%.

本文主要创新点在于:(1)基于自研的低功耗数字可调能量探测单元构建了动态程序运行环境,用于支持无源系统各类型程序运行;(2)我们给出了动态程序运行环境的设计方法、框架和重要流程,并讨论了与现有程序的兼容性问题;(3)我们建立了系统吞吐量模型用于描述动态运行环境下吞吐量与不同程序之间的关系;(4)我们实现了动态程序运行环境的概念验证原型(Proof-of-concept)并部署在 WISP 平台,通过实验评估其性能.本文在第 2 节介绍相关工作;在第 3 节介绍轻量级动态程序运行环境的详细设计;在第 4 节描述实验设计、实验过程和评估结果;在第 5 节总结全文.

2　相关工作

在讨论相关工作前,我们以可计算式 RFID 为例先讨论无源系统的能量特性.当标签靠近阅读器时,能量采集模块输出功率大于系统负载功耗,程序可连续执行,不会因掉电被打断.当标签逐渐远离阅读器时,能量采集模块输出功率减少到低于负载功耗.此时,程序须先等待系统采集能量,然后当能量累积到阈值后开始执行(这种工作模式叫作间歇式模式).当标签过于远离阅读器时,能量采集模块输出功率甚至小于系统的静态功耗,系统无法启动.因此,根据能量状态,无源系统可分为能量充足,能量不足和无法启动三种情况,一般来说大部分无源系统在应用中处于能量不足的情况(不会离阅读器太近).

在能量不足情况下,无源系统工作在间歇式模式.程序在该模式下的执行策略可以分为一次执行和断点执行两类.一次执行指程序在一个工作周期内(系统充能后执行程序的时间段)开始执行到结束;断点执行指程序需要多个工作周期才能结束,如 CRC 计算需要 16 个周期[9].用于支撑程序执行的运行环境也因此分为对应两类.显然,用于支持一次执行的运行环境难以支持断点类程序运行,因为缺乏断点保护和恢复机制.反之,支持断点执行的运行环境虽然能支持一次类程序运行,但加载了断点机制的运行环境存在额外高功耗开销.我们的目标为设计动态运行环境,使得运行环境同时支持两类程序执行,避免无用地加载断点机制而带来不必要的开销.下面我们针对现有两类运行环境进行讨论.

(1)一次执行.无源系统在采集足够能量后一次性执行完一个任务程序而没有任何中断.为此,系统为每一个任务分配了一个能量阈值,以判断是否开始执行任务,如图 2(a)所示.这种运行环境将任务定义为能一次性执行完的小程序,因此适合那些短时、轻量级的任务,如读取温度传感器.文献[7]提出了在无源系统 WISP 上一次执行运行环境的雏形.作者探究了在 WISP 上能实现的最强计算性能并探索在不同供能环境下的计算性能.但是算法的能量阈值是固定的 2 V,并非一个最优值.文献[6]优化了程序的能量阈值,综合考虑了程序的执行成功率和能量效率,选取出最佳的能量阈值,优化了能量效率.

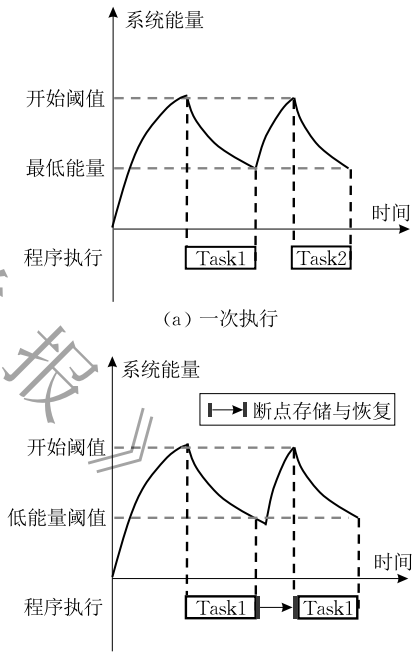


图 2　现有两种程序运行环境工作原理((a)一次执行环境每个周期仅执行一个任务程序.最低能量指能维持系统工作的最低电量,当能量低于该阈值时系统掉电关闭;(b)断点执行机制将一个程序分为多个周期执行直至程序结束.系统在能量达到低电量阈值时开始存储程序断点,并在能量回升到开始阈值时恢复程序断点继续执行)

一次执行运行环境具有轻量、算法简单、效率高等优点,但缺点也很明显,一次执行仅适用于那些短时间能完成的轻量级程序.一次执行运行环境的目标是在一个周期内完成程序执行,因此不包含低电量保护机制.出于对设备尺寸的考虑,无源系统中使用的微型能量存储器(一般为储能电容)容量仅为一

颗纽扣电池的万分之一[13].微能量使得无源系统很难一次执行完负载较高的程序.由于缺乏低电量保护机制,许多负载较高的程序将不停地因掉电而失败,永无成功完成之时.

(2)断点执行.对于那些无法在短时间内完成的任务程序,可以选择断点执行运行环境.断点执行指的是将一个任务划分成多个短时轻量级程序片段,在每一次系统处于激活期间尽可能地执行多个程序片段.当系统能量将要耗尽时,系统将程序执行的断点(即执行状态和数据)存储在非易失存储器中,等到系统采集到足够的能量后,再将任务从非易失存储器中恢复,从之前的断点处继续执行,直到程序完成,如图 2(b)所示.

断点执行运行环境可以分为能量驱动式和关卡驱动式两类.能量驱动式指运行环境根据当前能量情况决定是否设置断点.文献[10]提出当系统能量低于预设阈值时保存程序断点.这种方法直截了当,根据当前系统电量情况确定断点保存时机,但缺点也很明显,此时系统已处于能量匮乏的状态,还要执行保存断点等操作无疑会引入更多的能量开销,可能会导致系统频繁掉电,降低系统性能.文献[14]引入了动态频率调节技术,根据系统能量动态调节工作频率和功耗,当系统电量较低时降低系统工作频率,从而一定程度上缓解了系统掉电问题.文献[15]引入了自适应阈值调节机制,根据程序执行成功率尝试寻找一个最佳的程序开始执行阈值,使得系统能最大化程序执行时间、最小化程序执行失败率.文献[11]优化了断点存储的方式,将传统的非易失 Flash 替换为铁电存储器,加快了断点存储的时间并降低了功耗.文献[16]提出将断点信息以无线通信的方式发送到远程收发机上进行存储.文献[13,17]将断点执行的方式用于反向散射通信中,加强了无源系统在能量微弱环境下与远程收发机通信的能力.总的来说,能量驱动式断点虽然设计简单,但要求系统在电量较低时执行高功耗的断点保护机制.这要求系统预留一部分能量,降低了程序能量利用率.

关卡驱动式与电子游戏中每次通关后的存档操作颇为相似,将断点设置在若干重要的程序片段之后,当对应片段执行完成后,若电量较低,运行环境立即保存断点.当系统掉电后重新上电时,从最近的断点保存处开始执行,直至程序完成.文献[8]首次将关卡驱动式断点机制引入到无源系统中,并设计出断电执行运行环境的雏形.在此基础上,文献[9]给出一套完整的断点设置和存储机制,提出完整的关卡驱动式断点机制并将其存储在非易失 Flash 中.关卡驱动式断点的性能非常依赖断点设置的位置和数量,过多的数量会降低系统性能,过少又可能无法保障程序执行的可靠性.文献[18]从开发环境和编译环境对关卡驱动式断点设置进行了优化,降低了功耗和冗余计算.虽然关卡驱动式断点安全可靠,但断点机制的高功耗缺点依然难以得到解决,收集和存储断点信息的开销较大,带来显著的能量开销.

现有工作的局限:我们观察到,现有的程序运行环境仅能有效支持对应类型的程序执行,不能很好地支持其它类型的程序.具体的,虽然一次执行运行环境可以执行断点执行类程序,断点执行运行环境也能执行一次执行类程序,但这样不匹配的策略会导致程序执行效率下降(第 4 节实验显示,系统性能分别降低 43.5%和 27.7%).在实际应用中,这两类程序常存在于同一个系统,作为子任务构成了若干系统目标.以一个应用场景为例,系统在一个周期内完成从传感器采集到温度数据(一次执行类程序);接着,为了保障数据的传输可靠性,系统计算循环冗余校验码并与温度数据封装成包后发至远程上位机.由于该过程需要较大的计算量,在无源系统中常需要多周期的断点执行.现有的运行环境还难以满足这类需求,导致无源系统难以支持更多功能的应用场景,进而限制了无源系统的发展.

国内相关工作:国内与本文相关的研究主要集中在两个方面.其一是无源可计算式 RFID 的通信协议优化.文献[19]优化了无源 RFID 通信物理层设计,提出了一种新的用于无源 RFID 系统的能量高效的数据编码方法,增加了系统 2.5 dB 的平均接收功率.文献[20]优化了通信盘存阶段的时隙选择机制,对访问阶段的数据包大小进行设计优化从而提高系统有效吞吐量.文献[21]通过重新设计数据重传机制实现对通信链路的优化,基于动态规划的思路将多次重传的信息存储在缓冲区,减少数据重传次数.此外,文献[22]介绍了射频供能无源系统在老人跌倒检测中的应用.

其二是无源系统程序运行与控制.文献[23]中设计了可计算式 RFID 的程序控制系统.随后,文献[24]在此基础上实现了远程无线固件重编程和控制.这些工作是实现无源系统运行环境的基础,但还

无法支持运行环境的所有功能.运行环境除了程序控制外,还需要有能量探测与管理,中断控制等模块.文献[25-26]研究了可计算式 RFID 标签中的能量管理与任务规划算法,针对不同位置的标签所剩余的能量来动态地进行任务的规划,提高了 27.3% 能量利用率.但是,这些工作还未考虑多类程序混合执行的情况,而混合执行又在实际应用中时常作为应用需求出现,这使得无源系统在应用中性能容易受限.本文专注于无源系统中轻量级动态程序运行环境的设计,通过动态加载任务支撑模块实现对多类程序的轻量化支持.

3　轻量级动态程序运行环境设计

本节内容介绍动态程序运行环境设计,从系统层面支持两类程序交替混合执行,并且优化运行环境的内部设计,进一步降低开销.接着,我们为动态运行环境构建了吞吐量模型.最后,我们给出兼容性程序框架设计,提高动态运行环境与现有程序的兼容性.

3.1　动态运行环境设计

动态程序运行环境的基本思路为融合两类程序运行环境,使得运行环境能同时支持两类程序执行.现有运行环境典型结构如图 3 所示.其中,能量探测和断点机制是保障无源系统程序执行的两个主要模块.能量探测赋能系统感知当前系统能量(电量)状况,在电量充足时执行程序.在此基础上,断点机制负责在低电量情况下保存程序状态用于下次继续执行,仅存在于断点执行运行环境.

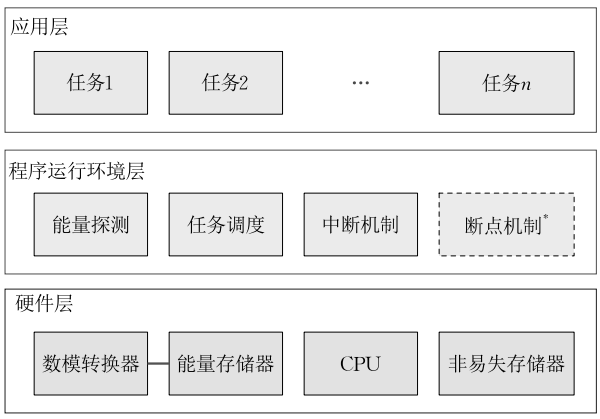


图 3　现有程序运行环境层次结构图
(其中断点机制* 仅存在于断点执行运行环境中)

现有程序运行环境是静态的,运行环境在设备启动时进行初始化,完成各个模块的加载(能量探测、任务调度、中断机制等).系统在运行过程中将不会再加载新的模块.一次和断点运行环境的区别主要体现在有无断点机制模块.具体而言,一次运行环境没有断点机制;断点运行环境虽加载了断点机制,但一直启用断点机制开销太大,特别是对于没有断点保护需求的程序来说,加载断点机制浪费了系统资源.

动态运行环境并非现有两种运行环境的简单组合.动态程序运行环境根据程序的类型动态加载断点机制模块,以降低系统开销,实现轻量级程序运行环境.图 4 给出了我们动态程序运行环境的关键结构设计.当任务调度模块确定执行的程序时,会根据程序类型(一次或断点执行)决定是否加载断点机制.能量探测模块为断点机制工作提供低电量事件探测.因此,动态加载断点机制也需要实现能量探测模块的动态化.具体而言,一次执行运行环境的能量探测一般只探测一个开始阈值,而断点执行在其基础上还要探测低电量预警阈值,系统需要随任务类型动态地添加/删除能量探测模块的功能.

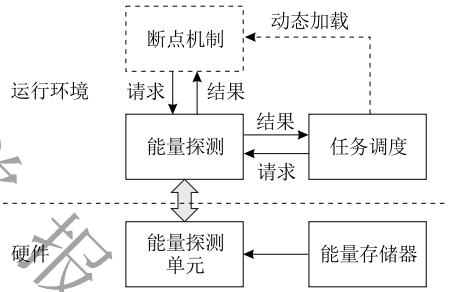


图 4　动态程序运行环境的关键结构设计

但是,无源系统中的能量探测模块难以满足轻量级动态运行环境的需求,或者是功耗太高,或者是难以实现动态化.无源系统中能量探测模块主要可以分成两类.第一类是传统的软件轮询,系统调用数模转换器 ADC 采集能量存储器中的电压,经过换算得到当前系统能量.由于该过程主要通过软件控制,所以软件轮询灵活可控,能满足动态化能量探测的需求.但其功耗也很高,例如在 WISP 平台上,片上 ADC 的功耗是计算单元功耗的 2 到 3 倍.第二类是硬件探测,原理为利用预设好阈值的比较器电路,当电压达到预设阈值时,比较器输出触发信号唤醒系统.硬件探测的优点在于低功耗,WISP 平台上的硬件探测器功耗 0.63 μW①,仅为片上 ADC 功耗的

① S-1000 series ultra-small package high-precision voltage detector. https://www.ablic.com/en/doc/datasheet/voltage_detector/S1000_E.pdf, 2015, 1, 8

千分之一.但传统的硬件探测缺点也颇为显著,其预设阈值在硬件出厂时固定,难以修改,因此无法实现能量探测动态化.

传统硬件探测仅适合一次执行运行环境,难以满足断点运行环境对能量探测的需求.断点运行环境由于要探测多个阈值,常使用软件轮询的能量探测.在动态程序运行环境中,只有软件轮询能实现动态能量探测,但其高功耗会导致一次执行类程序的效率降低.此外,简单地改造硬件探测电路虽可能实现阈值可调,但会带来显著的漏电流,造成系统静态功耗显著增加[5].能量紧缺导致现有无源系统在维持程序执行方面本已举步维艰,我们不希望以降低太多程序性能和能量效率为代价实现动态的程序运行环境.

我们基于自研的低功耗数字可调硬件探测单元[5]构建了动态程序运行环境.图5给出了该单元的结构图,其中电压比较器将能量存储器电压和阈值电压进行对比,若能量存储器中电压大于阈值,比较器则输出高电平,反之输出低电平.阈值电压可以通过数字接口进行调节,设默认阈值电压为V_{ref},数字可调阈值V_{th}可由以下公式得到.

$$V_{th}(c)=\frac{c}{s-c}\times V_{ref} \tag{1}$$

其中,c是发送给数字接口的正整数阈值调节码,$c\in(0,S]$,S为阈值探测单元支持的调节码最大值.低功耗数字可调能量探测单元与现有能量探测模块的能耗对比如表1所示.

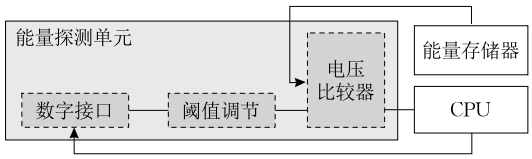


图5 数字可控低功耗能量探测单元结构图

表1 低功耗数字可调硬件探测与其他探测单元的对比[5]

能量探测单元	能耗类型描述	能耗/μJ
硬件探测	硬件探测	0.63(每秒)
软件轮询	软件轮询	225(每秒)
低功耗数字可调硬件探测[5]	硬件探测	11.7(每秒)
	设置一次阈值	39.7

我们在能量探测单元的基础上,实现了动态程序运行环境,其流程图如图6所示.运行环境根据不同程序类型动态加载系统模块.当系统上电后,若不存在未完成的程序,任务调度从程序集选择要调度的任务程序,并判断程序类型.若程序为断点执行程序,运行环境则动态加载断点机制.断点机制能在系统能量低于低电量阈值时,将运行环境和执行程序的状态信息(如寄存器、指针、堆在等)保存至非易失存储器中,以便当系统下次上电时进行恢复.随后,运行环境根据任务需求配置能量探测单元,设置开始阈值,接着进入低功耗模式等待系统充能.当电量高于开始阈值时,若当前程序类型是一次执行,则直接执行程序;若为断点执行,运行环境首先设置低电量预警阈值,使能执行断点机制,然后开始/恢复程序执行(根据程序是否未完成判断).当相应程序

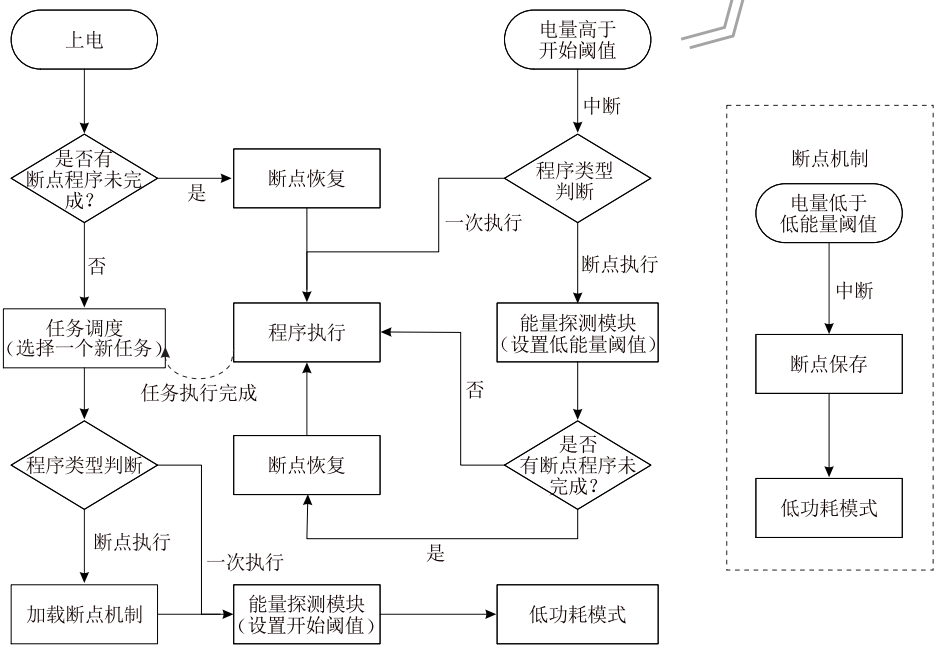


图6 动态程序运行环境流程图(在低功耗模式下系统处于休眠状态不执行程序,等待电量达到开始阈值或重新上电后才开始执行)

执行完成后，任务调度模块从程序集中再挑选新的任务程序用于执行，以此实现低功耗多类程序运行支撑.

3.2　系统吞吐量模型

设动态程序执行环境可执行程序集 Ψ 中包含 n 个独立子任务 $\Psi=\{\alpha_1,\alpha_2,\cdots,\alpha_n\}$. 我们定义执行任务 α_i 的吞吐量为 $H_i=1/T_{c,i}$，其中 $T_{c,i}$ 为任务 α_i 的完成时间(从任务开始到完成的时间). 在动态执行环境中，一次执行和断点执行程序的完成时间是不同的. 若任务 α_i 是一次类程序，其完成时间可由如下公式得到：

$$T_{c,i}=t_{\text{exe}}+t_{\text{charge}},$$

其中，t_{exe} 为程序在无中断情况下执行的时间，t_{charge} 指为执行程序无源系统所需的充能时间. 另一方面，断点执行程序完成时间可由如下公式得到：

$$T_{c,i}=t_{\text{exe}}+n_{\text{cycle}}\cdot t_{\text{charge}}+(n_{\text{cycle}}-1)\cdot t_{\text{cp}},$$

其中，n_{cycle} 为任务执行完成需要的工作周期数，t_{cp} 为执行一次断点机制(保存并恢复断点)所需要的时间开销. 在 WISP 平台上，每次断点机制开销为 5 ms(保存断点 2.8 ms，恢复断点 2.2 ms). 因此，平均吞吐量(单位时间内完成的子任务数量)可表示为

$$H_{\text{sys}}=\frac{n}{\sum_{1}^{n}T_{c,i}+t_{\text{dy}}\cdot n_{\text{dy}}},$$

t_{dy} 和 n_{dy} 分别为运行环境动态加载断点执行模块的时间开销和次数. 在 WISP 平台上，加载断点执行模块需要 $t_{\text{dy}}=120\,\mu$s 的时间. n_{dy} 与任务执行顺序有关.

3.3　兼容性程序框架设计

动态程序运行环境要求系统能够判断每个任务的执行类型. 为此，我们需要对每个程序进行离线标记，以便运行环境在调度和执行程序的过程中进行类型判断. 离线标记应是与程序独立的. 我们认为好的运行环境应该与现有无源系统中的程序相兼容，能将目前执行在其他运行环境下的程序移植过来，不需要修改现有程序代码和设计框架.

我们在开发阶段对程序进行标注，其过程如图 7 所示. 首先，程序源代码经过编译和链接之后生成无源系统中的可执行程序. 接着便是本文所提的离线标记环节，我们在程序的前端添加一个程序描述块 PDB，其主要元素如表 2 所示. 在程序标记后，程序描述块和程序会一并放入无源系统的非易失存储器，等待运行环境在运行阶段调度. 运行环境通过 PDB 识别和调度所有无源系统中的任务.

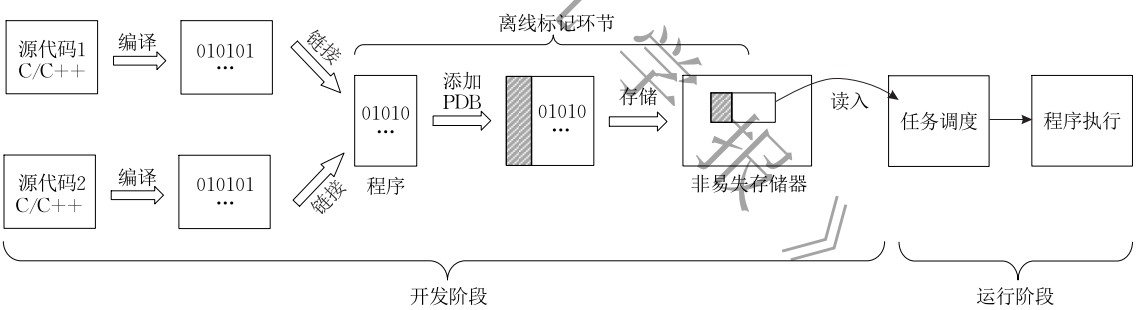


图 7　离线标记用于实现兼容性程序框架

表 2　程序描述块 PDB 记载的程序信息

程序信息	描述	数据类型
PID	程序标识号	无符号 1 字节
P_type	程序类型. 1 表示一次执行；2 表示断点执行	无符号 1 字节
P_state	程序状态/执行计数器. −1 表示正在执行，正整数为已执行次数	有符号 1 字节
P_num_block	程序代码存储所占用的区块数	无符号 1 字节
reset_vector	重置向量地址	函数指针

在程序存储方面，我们借鉴了文献[27]中的存储结构. 将存储器的高位区域设定为运行环境存储区，低位区域存储各类任务程序. 由于上电时系统默认从高位区域加载代码，这种存储结构会让系统首先执行运行环境相关代码，然后再执行程序，保证了程序在无源系统环境下可靠执行. 与该文献不同的是，我们在每个程序前添加了程序描述块，用于描述对应程序的相关信息，以便于运行环境进行调度和配置. 我们将所设计的存储结构实现在 WISP 平台，其固件存储组织结构如图 8 所示. WISP 平台的微控制器使用了片上 Flash 存储器，存储空间为 8K 字节，分为 16 个区块. 当系统上电时，系统默认从区块 12 的起始地址加载代码开始执行. 换句话说，系统上电后首先会执行运行环境的固件代码. 在程序执行方面，运行环境首先判断程序类型，并动态加载相关系统模块完成对程序执行的支撑. 然后跳转到程序代码开始任务执行. 当程序代码执行完成后，将执行重置中断代码. 这段代码将执行指针引导至从区块 12 的起始地址开始，进而重复以上过程.

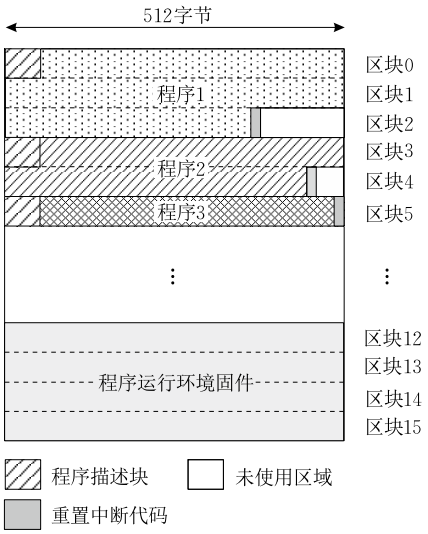


图 8 运行环境和程序固件组织结构(WISP 平台)

4 实验评估和结果分析

4.1 实验平台

我们将所设计的动态运行环境实现在英特尔公司/华盛顿大学的 WISP 平台[1]上. WISP 是款无线射频供电的无源设备,支持 EPC C1G2 协议无线通信,能通过现有商用 RFID 读写器进行读写访问,工作频段在 902 MHz～928 MHz. WISP 具有计算和感知能力,板上集成有 16 位低功耗微控制器,温度传感器,加速度传感器等,具有体积小、部署简单等优点,能深度嵌入到物理环境中,在物联网和普适计算中具有广阔的应用前景.

程序运行环境中的断点机制需要将程序断点保存在 Flash 存储器中. 原版 WISP 工作在 1.8 V 电压下,但 Flash 存储器稳定工作要求 2.2 V 电压,原版 WISP 难以支持动态运行环境工作. 由于 WISP 平台软件和硬件均是开源的①,在原版 WISP 基础上,我们替换了平台的稳压器模块,将原有的 1.8 V 输出变更为 2.5 V,用于支持片上 Flash 存储器稳定工作. 此外我们也优化了平台的硬件布局以提高平台性能,使其拥有更远的读写距离. 原版和改进版 WISP 平台如图 9 所示.

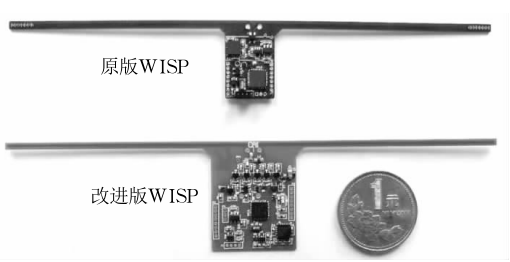


图 9 实验平台为射频供电的改进版 WISP

此外,改进版 WISP 集成了低功耗数字可调阈值机制. 板上处理器和能量探测单元通过 I^2C 总线进行通信. 在实验中,阈值调节机制主要可分为两个部分. 第一是设置阈值,我们根据任务的阈值需求,通过式(1)计算得到传输给能量探测单元的数字调节码并通过总线进行发送. 为了降低系统开销,调节码的计算可离线计算好再写入任务程序中. 第二部分为阈值触发. 当能量探测单元发现当前系统能量高于(或低于)之前设置的阈值时,探测单元输出中断信号给处理器. 随后,中断处理程序捕获到中断信号后,唤醒对应任务,并将中断事件消息发送给任务消息接口.

4.2 实验设置

我们使用英频杰公司的 Speedway R420 阅读器为 WISP 平台供能并与之通信. 阅读器安装有 6 dBi 增益的天线,发射功率为 30 dBm. 实验部署如图 10 所示,阅读器天线和 WISP 平台在同一水平面,离地约 1.2 m. 实验中,WISP 采集足够能量后在运行环境的支撑下执行若干程序. 我们将程序吞吐量作为评估运行环境的指标. 程序吞吐量指在运行环境中给定时间内完成的程序执行总量. 为了准确地测量吞吐量,我们将 WISP 平台中一根微控制器引脚配置成输出模式,当程序执行时引脚输出高电平,执行完成后置为低电平——这样我们就能通过一个数字示波器来记录 WISP 输出的高低脉冲个数,实现吞吐量测量. 此外,我们保持这个实验环境,调节天线和 WISP 之间的距离,在不同的距离下测量运行环境的吞吐量. 然而保持实验环境理想化不变非常困难,如人体自然抖动、周围行人走动,都可能影响实验测量结果. 为了最大化实验测量的准确性,我们对每组实验都进行了 50 次测量,并最终取平均值.

在评估环节,为了与本文设计的运行环境进行对比,我们同样实现了现有的两类运行环境,即一次执行和断点执行运行环境,一同参与实验评估. 它们的简介如下:

(1)一次执行运行环境. 我们将文献[6]中的运行环境实现在改进版 WISP 平台上作为一次执行运行环境的示例. 该文献提出的一次运行环境能量效率是目前同类运行环境中最高的,达到了 64.7%[5].

(2)断点执行运行环境. 我们将文献[9]所提出的断点执行运行环境同样实现在改进版 WISP 平台

① WISP 平台开源 Github. https://github.com/wisp, 2018, 10, 5

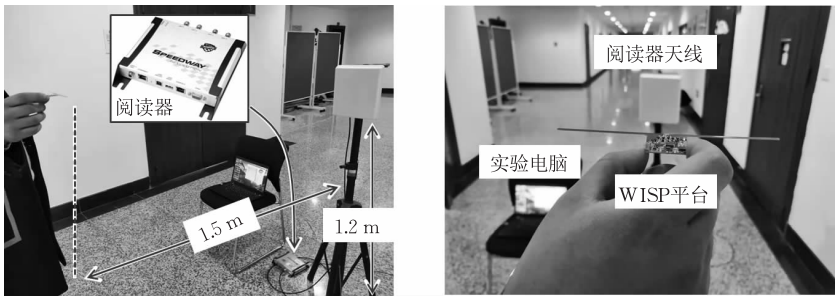


图 10　实验评估环境

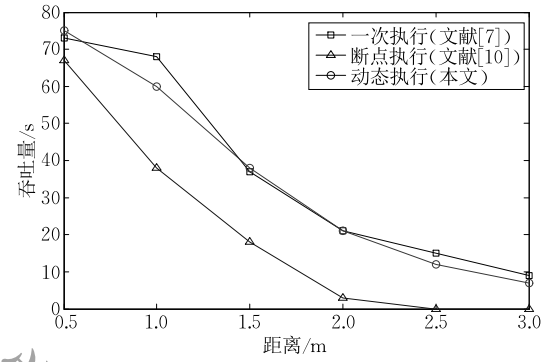
上. 该运行环境使用的是关卡式断点机制，比能量驱动式有更高的效率.

在后续的实验评估中，我们主要从两个方面对本文设计的运行环境进行评估. 第一，单程序执行评估，通过测量程序吞吐量评估运行环境对无源设备中常见类型程序的支持情况. 从程序的分类来看，我们考虑了无源系统中常见的三类任务，即感知、计算和通信. 第二，程序集执行评估，评估运行环境对多个程序(程序集)执行的支持情况. 其中程序集为混合执行，不仅包含一次执行类程序，也包含断点执行类程序. 我们测量不同程序集在不同运行环境下的执行吞吐量，以评估运行环境的性能.

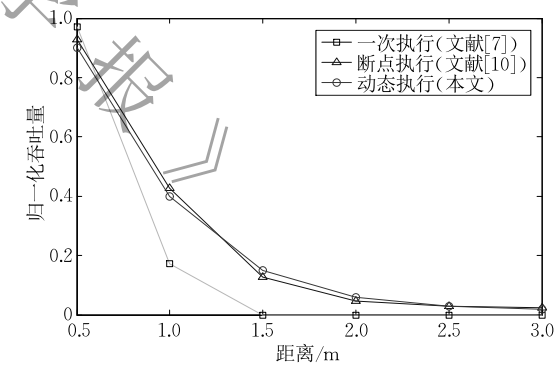
4.3　单程序评估

无源系统中的绝大多数程序可以分为感知、计算和通信三类任务. 感知类任务主要指微控制器从传感器采集数据；计算类任务主要负责对数据进行预处理，如压缩或加密等；通信类任务主要为通信模块与远程设备的数据交换，通信协议栈的实现等. 在单程序评估中，我们考虑这三类程序在不同运行环境下的执行情况(吞吐量). 每种类型的实验评估如下：

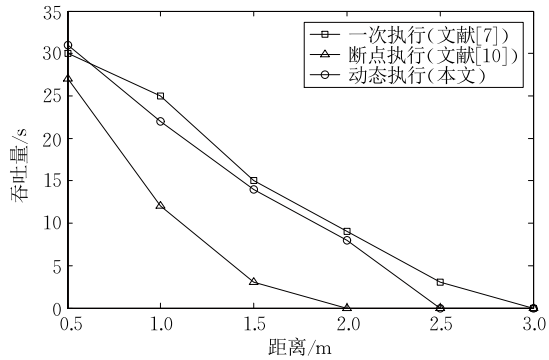
(1) 感知类任务. 板上 MCU 每隔 10 ms 从温度传感器采集温度信息. 由于该任务无法中途暂停(掉电会导致传感器数据丢失)且执行时间较短，因此认定该任务属于一次执行类程序. 我们将感知类任务放在一次执行、断点执行和本文的动态运行环境中执行，其结果如图 11(a)所示. 我们发现感知类任务在一次执行运行环境和动态运行环境中的吞吐量差别不大，这是因为动态运行环境在识别出任务类型后，会以一次执行的方式来运行此任务. 而在断点执行运行环境下，感知类任务的吞吐量有着明显的下降. 这其中暗含两个信息. 其一，断点执行环境并非无法执行一次类程序，而是无法支持该类型程序高效的执行，在性能上有所降低，系统吞吐量平均下降 43.5%. 其二，断点执行运行环境程序吞吐量降低的原因在于，运行环境需要在每次电量较低时存储程序断点并在下次上电时恢复断点继续执行，感知类任务无法进行恢复，因此该过程是无效的，反而带来额外的开销.



(a) 感知任务



(b) 计算任务



(c) 通信任务

图 11　单程序实验评估结果

(2)计算类任务.板上 MCU 对 2 KB 的数据进行 CRC16-CCITT 校验和计算.由于该任务在无源系统环境下需要的时间较长,甚至能达到数秒,并且计算可以暂停,因此该任务属于断点执行类程序.与上面感知类任务的实验类似,我们将 CRC 计算放在三种运行环境下运行,测量程序执行吞吐量,结果如图 11(b)所示.请注意,由于部分供能距离下 CRC 计算会超过 1 s 执行时间,为了方便读者阅读,我们将测量的结果进行了归一化处理.实验结果显示,本文所提出的动态执行效果与断点执行效果相差无几,这是因为动态执行检测到程序为断点类后加载了断点存储和恢复机制,实现了程序断点执行.此外,一次执行运行环境虽然也能支持断点类的 CRC 计算,但吞吐量平均下降了 27.7%.这是因为当 WISP 平台距离阅读器很近时,WISP 供能充足,CRC 计算能在无中断的情况下一次性执行完毕,因此一次运行环境执行的 CRC 计算在 0.5 m 时的吞吐量与其他两种环境差距不大,甚至还可能实现更高的吞吐量(毕竟一次执行环境较为简单,具有更少的开销).但随着 WISP 与阅读器的距离增加,一次执行环境的吞吐量急速下降,在距离为 1.5 m 时吞吐量为 0,无法完成 CRC 计算.这是因为一次执行运行环境不会在电量降低时保护程序断点,一旦掉电数据将会丢失,需要重新计算 CRC.在能量较少的情况下,WISP 平台会频繁掉电,导致 CRC 计算任务无法完成.

(3)通信类任务.板上 MCU 执行 EPC C1G2 协议栈,通过控制天线阻抗进行反向散射通信.与其他通信模块不同,WISP 平台采用软件定义无线电.虽然软件计算可断续执行,但是商业阅读器所使用的协议并不支持通信断点式进行,因此通信过程必须在一个执行周期内完成,属于一次执行类程序.我们在对应三种运行环境的支持下实现了 EPC 通信,具体的,阅读器通过 Query 命令获取 WISP 平台的 ID.其中一次执行环境我们使用了原版 WISP 的 EPC 示例程序.我们在阅读器端记录每秒读取到的 ID 次数作为通信类任务的吞吐量,其结果如图 11(c)所示.我们可以发现,动态执行的吞吐量与原版程序效果接近,仅有少量差距.主要原因是工程实现方面的问题,原版程序的通信核心部分广泛使用了汇编语言进行设计,虽然效率高但模块化不及 C 语言.在动态执行环境中,将原版的核心通信代码转换过来会存在一定的额外开销.此外,断点执行环境 EPC 通信任务的性能较低,主要原因还是断点执行所需要的开销太大,并且远程的商业阅读器不支持断点通信.如果阅读器支持断点通信,我们相信吞吐量和最大通信距离会有所改善.

4.4 程序集评估

本节评估运行环境在多任务情况下对程序执行的支持情况.我们在实验评估中考虑四个任务,分别为温度感知、加速度感知、CRC 计算和 RC5 加密.其中温度感知和 CRC 计算与上一节相同,分别属于一次执行类程序和断点执行类程序.加速度感知类似于温度感知,通过板上 MCU 每隔 10 ms 采集加速度传感器的数据,属于一次执行类程序.RC5 加密使用 RC5-32 算法,设计和实现见文献[7].RC5 加密计算可间断执行,属于断点执行类程序.我们将这四个程序进行分类和组合,构建了四个程序集,如表 3 所示."全一次执行"程序集用于评估运行环境对多个一次执行类程序的支持,"全断点执行"程序集同理.

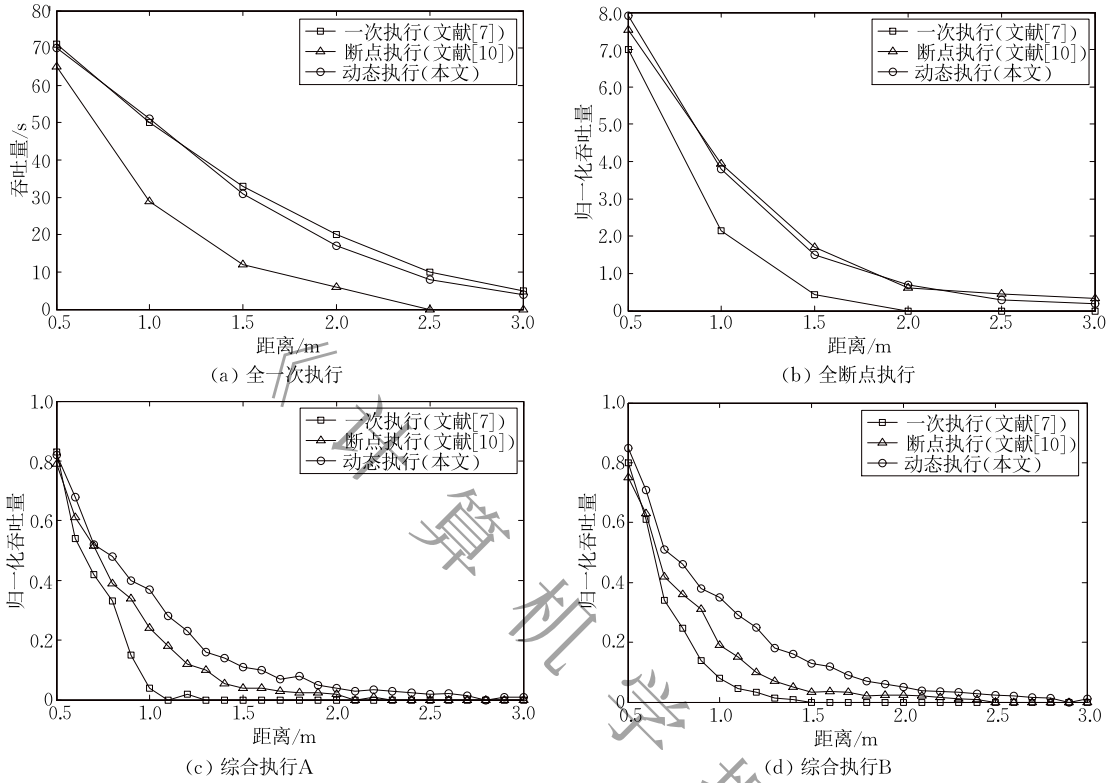
表 3 程序集组成和执行顺序

程序集名称	包含任务和执行顺序
全一次执行	温度感知、加速度感知
全断点执行	CRC 计算、RC5 加密
综合执行 A	温度感知(一)、加速度感知(一)、CRC 计算(断)、RC5 加密(断)
综合执行 B	RC5 加密(断)、加速度感知(一)、CRC 计算(断)、温度感知(一)

"综合执行 A"和"综合执行 B"程序集包含了以上两类程序,区别仅在于程序执行顺序.值得注意的是,四个独立任务的排列组合本应产生"综合执行 A,B,C,D,…"多达 24 个程序集.我们只选择了两个典型组合来进行评估,原因如下.动态程序运行环境会根据程序类型来加载运行环境的模块.当相邻程序类型一致时,运行环境无需进行调整,这种情况已在"全一次执行"和"全断点执行"程序集中进行了实验评估.只有当前程序类型与下个程序类型不同时,系统进行动态模块调整.具体而言,模块调整可分为"一次→断点"和"断点→一次"两类,"综合执行 A"考虑到了这两种情况(程序集内的任务是循环执行的,RC5 执行完之后重新从温度感知开始执行)."综合执行 B"在此基础上,调整了两类程序的分布,提高了动态模块调整的频率.其他程序集达到的效果类似.因此,我们选择了"综合执行 A"和"综合执行 B"作为典型程序集用于实验评估.

同上一节,我们对比一次执行、断点执行和动

态执行三类运行环境对以上程序集的支持.“全一次执行”程序集执行的吞吐量如图 12(a)所示，随着供能距离增加，程序吞吐量均出现相应的下降.其中断点执行运行环境的吞吐量在 2.5 m 时降低到 0，无法完成程序执行.一次执行和动态执行在 3 m 处依然能支持程序运行.该现象主要是因为断点机制无法有效应用于一次执行程序，系统需要的启动能量较高，导致工作距离减少.动态运行环境和一次执行运行环境的执行效果相似，因为动态运行环境在执行一次执行类型任务时，不会加载断点机制，此时运行环境与一次执行运行环境一致.



(a) 全一次执行　(b) 全断点执行　(c) 综合执行A　(d) 综合执行B

图 12　程序集评估实验结果(动态运行环境能高效率地支持两类程序执行)

“全断点执行”程序集执行的吞吐量如图 12(b)所示.由于一次执行运行环境不能很好地支持断点类程序执行，其吞吐量随距离迅速降低.当距离较近时，能量供应较为充足，系统的执行时间较长，因此一次执行运行环境可以在一个执行周期内完成断点类程序.但随着距离增加，单个周期时间减少，无法满足断点类程序的计算量需求，频繁掉电导致任务失败，难以完整执行程序.此外，断点执行运行环境和动态运行环境的结果非常类似.这说明动态运行环境在加载断点执行机制后，与传统的断点执行运行环境在功能上没有区别.

“综合执行 A”程序集执行的吞吐量如图 12(c)所示.为了更加准确地评估不同运行环境，我们将实验中距离变化的步长细致到 0.1 m.由图所知，所执行的程序集包含一次执行和断点执行两类程序时，不同运行环境的吞吐量有明显的区别.当距离靠近供能源(阅读器)时，能量较为充足，三类运行环境的执行效果差距不大.随着距离增加，各运行环境的吞吐量在 1 m～2 m 时差别较大.首先，一次执行环境的吞吐量随着距离增加迅速降低.追其缘由，虽然一次执行对程序集中的温度感知和加速度感知支持性能较好，但在执行 CRC 和 RC5 计算时缺乏断点保护机制，导致任务大概率执行失败.在实验中，我们的设定是系统需要成功执行完成一个任务后才能执行下一个任务.短缺的能量会导致一次执行环境需要大量重复地执行 CRC 和 RC5 计算，直至其成功完成后，才循环到程序集开头执行温度感知任务，从而大大降低了系统吞吐量，在 1.3 m 之后已无法支持程序集运行.然后，断点执行运行环境的吞吐量整体上要高于一次执行运行环境，因为断点运行环境对断点类程序有较好的支持力度，避免了一次运行环境所遇到的重复执行问题.但是断点执行运行环境在执行温度、加速度感知等一次执行类任务时开销较大.本文所提出的动态运行环境会根据程序类

型动态加载对应运行环境,因此能更加高效地支持两类程序执行.

"综合执行 B"程序集用于评估程序集中不同任务执行顺序对吞吐量的影响,其结果如图 12(d)所示.同图 12(c)对比可知,任务的执行顺序对吞吐量的影响很小.这是因为,对一次执行和断点执行运行环境来说,各运行环境执行程序的策略是不变的,不会受执行顺序影响.对动态运行环境来说,不同的执行顺序确实会影响系统在各运行环境之间切换的频率,导致额外的切换开销.但是,本文所提出的轻量级运行环境切换机制将切换的开销降低至无源系统可以承受的程度(仅 120 μs),从吞吐量上来看,这样的开销是可以忽略的.

综上所述,动态运行环境能较好地支持一次类和断点类程序执行,在执行某一类型任务时,其效率和对应运行环境一致.在执行两类任务组成的程序集时,动态运行环境的吞吐量比一次执行运行环境高 102.8%,比断点执行运行环境高 34%.

5 结束语

针对无源系统中现有运行环境仅适用于单类型程序执行,无法同时支持所有程序类型的问题,本文提出动态程序运行环境,轻量地支持所有程序类型混合执行.基本实现思路根据程序执行类型,动态地加载程序运行策略,包括断点保护和恢复机制,实现对不同类型程序的高效支持.为了降低运行环境的能量开销,本文运行环境基于低功耗数字可控能量探测单元实现.我们分析了动态程序运行环境的设计方法、框架和重要流程,并且构建了系统吞吐量模型.此外,本文介绍了兼容性程序设计框架以兼容无源系统已有程序.我们在 WISP 平台进行了实验,评估并对比了运行环境对单程序执行和多程序集执行的支持效果.实验结果显示,动态程序运行环境能良好地支持无源系统中所有类型的程序执行.在各类型程序混合的程序集执行评估中,动态运行环境的吞吐量比一次执行运行环境高 102.8%,比断点执行运行环境高 34%.本文的后续研究主要集中于研究如何轻量化断点存储和恢复机制,提高系统能量效率.

参 考 文 献

[1] Sample A P, et al. Design of an RFID-based battery-free programmable sensing platform. IEEE Transactions on Instrumentation and Measurement, 2008, 57(11): 2608-2615

[2] Zhang H, et al. Moo: A batteryless computational RFID and sensing platform. University of Massachusetts Computer Science: Technical Report UM-CS-2011-020, 2011: 1-4

[3] Simjee F, Chou P H. Everlast: Long-life, supercapacitor-operated wireless sensor node//Proceedings of the International Symposium on Low Power Electronics and Design. Tegernsee, Germany, 2006: 197-202

[4] Hester J, Sitanayah L, Sorber J. Tragedy of the coulombs: Federating energy storage for tiny, intermittently-powered sensors//Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems. Seoul, South Korea, 2015: 5-16

[5] Li S, et al. Sentinel: Breaking the bottleneck of energy utilization efficiency in RF-powered devices. IEEE Internet of Things Journal, 2019, 6(1): 705-717

[6] Buettner M, Greenstein B, Wetherall D. Dewdrop: An energy-aware runtime for computational RFID//Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation. Boston, USA, 2011: 197-210

[7] Chae H-J, et al. Maximalist cryptography and computation on the WISP UHF RFID tag//Proceedings of the Wirelessly Powered Sensor Networks and Computational RFID. New York, USA, 2013: 175-187

[8] Ransford B, et al. Getting things done on computational RFIDs with energy-aware checkpointing and voltage-aware scheduling//Proceedings of the Workshop on Power Aware Computing and Systems. San Diego, USA, 2008: 1-6

[9] Ransford B, Sorber J, Fu K. Mementos: System support for long-running computation on RFID-scale devices//Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems. Newport Beach, USA, 2011: 159-170

[10] Balsamo D, et al. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. IEEE Embedded Systems Letters, 2015, 7(1): 15-18

[11] Jayakumar H, et al. QuickRecall: A HW/SW approach for computing across power cycles in transiently powered computers. ACM Journal on Emerging Technologies in Computing Systems, 2015, 12(1): 1-19

[12] Gomez A, et al. Dynamic energy burst scaling for transiently powered systems//Proceedings of the Conference on Design, Automation & Test in Europe. Dresden, Germany, 2016: 349-354

[13] Zhang P, Ganesan D. Enabling bit-by-bit backscatter communication in severe energy harvesting environments//Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation. Seattle, USA, 2014: 345-357

[14] Balsamo D, et al. Graceful performance modulation for power-neutral transient computing systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2016, 35(5): 738-749

[15] Balsamo D, et al. Hibernus++: A self-calibrating and adaptive system for transiently-powered embedded devices. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2016, 35(12): 1968-1980

[16] Salajegheh M, et al. CCCP: Secure remote storage for computational RFIDs//Proceedings of the 18th USENIX Security Symposium. Montreal, Canada, 2009: 215-230

[17] Zhang P, Ganesan D, Lu B. QuarkOS: Pushing the operating limits of micro-powered sensors//Proceedings of the 14th USENIX Conference on Hot Topics in Operating Systems. Santa Ana Pueblo, USA, 2013: 1-6

[18] Mirhoseini A, Songhori E M, Koushanfar F. Idetic: A high-level synthesis approach for enabling long computations on transiently-powered ASICs//Proceedings of the IEEE International Conference on Pervasive Computing and Communications. San Diego, USA, 2013: 216-224

[19] Wang Hong-Gang, Pei Chang-Xing, Yi Yun-Hui. Energy efficient physical layer design for passive RFID system. Chinese Journal of Computers, 2009, 32(7): 1356-1364(in Chinese)
(王宏刚，裴昌幸，易运晖. 无源 RFID 系统中能量有效的物理层设计. 计算机学报，2009，32(7)：1356-1364)

[20] Wu Hao. Research on Energy Utilization and Communication Goodput in CRFID System [M. S. dissertation]. Taiyuan University of Technology, Taiyuan, 2019(in Chinese)
(吴昊. CRFID 系统中能量利用率与通信吞吐量研究[硕士学位论文]. 太原理工大学，太原，2019)

[21] Lin Xiao-Jie. The Research on the Optimization of Perceptual Link Technology in Passive System [M. S. dissertation]. Taiyuan University of Technology, Taiyuan, 2018(in Chinese)
(林小洁. 无源系统中感知链路技术的优化研究[硕士学位论文]. 太原理工大学，太原，2018)

[22] Yan Yu-Juan, Li Hua, Zhao Ju-Min, et al. Fall detection system based on CRFID and pattern recognition. Computer Engineering, 2019, 45(6): 297-302, 309(in Chinese)
(闫玉娟，李化，赵菊敏等. 基于 CRFID 和模式识别的跌倒检测系统. 计算机工程，2019，45(6)：297-302，309)

[23] Wen Bin-Yu. Real-Time Program Control in CRFID Systems [M. S. dissertation]. University of Electronic Science and Technology of China, Chengdu, 2014(in Chinese)
(文彬宇. 可计算 RFID 实时程序控制系统[硕士学位论文]. 电子科技大学，成都，2014)

[24] Wu Die. Research on Firmware Control Methods of Computational RFID Systems [Ph. D. dissertation]. University of Electronic Science and Technology of China, Chengdu, 2018(in Chinese)
(吴迭. 无源可计算 RFID 系统固件控制方法研究[博士学位论文]. 电子科技大学，成都，2018)

[25] Guo Ru-Ru. The Research on Energy Optimization and Task Planning on WISP[M. S. dissertation]. Taiyuan University of Technology, Taiyuan, 2018(in Chinese)
(郭茹茹. WISP 系统中能量优化与任务规划的研究[硕士学位论文]. 太原理工大学，太原，2018)

[26] Guo Ru-Ru, Gong Jian-Ping, Zhao Ju-Min, et al. Research on energy management and task planning algorithm in CRFID. Journal of North University of China(Natural Science Edition), 2018, 39(6): 726-732, 751(in Chinese)
(郭茹茹，巩建平，赵菊敏等. CRFID 中的能量管理与任务规划算法研究. 中北大学学报(自然科学版)，2018，39(6)：726-732，751)

[27] Wu D, et al. Remote firmware execution control in computational RFID systems. IEEE Sensors Journal, 2017, 17(8): 2524-2533

LI Song-Fan, Ph. D. candidate. His research interests include battery-free systems, energy management, low-power technology.

LU Li, Ph. D., professor. His research interests include smart terminals in IoT, security in wireless systems, RFID technology.

LI Jie, B. S., police supervisor (class III). His research interests include information systems and networks.

Background

As a result of scarce energy, battery-free systems work in discrete time-slots instead of running all the time. In this situation, a program runtime is proposed to protect program executions under unpredictable ambient energy conditions and further improve the energy efficiency to relieve the energy problem.

Recently, a number of runtimes have been proposed in battery-free systems. According to the program execution, the runtimes can be categories into two types. First is called One-Cycle runtimes, which the system accomplishes a complete program at one-time once upon accumulating enough energy for task execution. Second is known as Multi-Cycle runtimes

that the system slices a task into several pieces and executes the pieces as many as possible according to the energy state. When the power is using up, the system will save the status of task execution and recover it once upon the system is powered up again. The two types of runtimes adapt to different programs of characteristics. The One-Cycle runtimes is suitable to execute uninterruptible programs such as operating sensors and communicating with a module, because these programs cannot be sliced into multi-pieces. Besides, the Multi-Cycle runtimes adapt long-term computational programs that are recoverable and hardly accomplish in a single cycle.

However, existing runtimes hardly satisfy program executions in practice. Many applications require executing sensing and computing programs, while neither the One-Cycle runtimes nor the Multi-Cycle runtimes cannot support all of them effectively. Specifically, the One-Cycle runtimes scarcely support long-term computing programs, while the Multi-Cycle runtimes cannot deal with un-recoverable programs.

This work attempts to design a dynamic and lightweight runtime environment for battery-free systems, which enables all types of programs executing in the runtime effectively. In order to achieve this, we mark the execution type for programs and enable the runtime dynamically loading the system's modules according to the type of programs. Further, we present a lightweight runtime loading technique that makes use of a low-power digitally controllable energy supervisor to support the runtime operations on the demands of applications. We conduct experiments to evaluate our design on the Intel WISP platform. The results show that the throughput of the adaptive runtime is 102.8% higher than that of the burst execution runtime, and 34% higher than that of the checkpoint execution runtime.

This work is supported in part by the National Natural Science Foundation of China under Grant Nos. 61872061 and 61976047, in part by the National Key Research and Development Program of China under Grant No. 2017YFB1003003, and in part by the Science & Technology Department of Sichuan Province of China under Grant No. 2019YFG0122. This work belongs to a part of the project entitled "Energy Theory and Key Technology in Passive Sensing and Computation Systems" that aims to address the energy problems in battery-free systems.