软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(7):1840−1852 [doi: 10.13328/j.cnki.jos.004706]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

多核处理器的功耗估算模型*

刘 辛, 沈 立, 苏 博, 王志英

(高性能计算国家重点实验室(国防科学技术大学 计算机学院),湖南 长沙 410073)

通讯作者: 刘辛, E-mail: liuxin12@nudt.edu.cn

摘 要: 精确的功耗估算能够为操作系统调度、软/硬件能效优化提供有效的指导.以往的研究表明:通过监测处理器内部相关硬件事件(如提交的指令数、Cache 访问次数等),可以对功耗进行估算.但是,已有的相关功耗模型的精度并不理想,误差通常在 5%以上.通过分析处理器提供的硬件事件,并在众多事件中筛选出一组与程序运行功耗密切相关的事件,使用逐步多元线性回归分析,建立了一个与应用无关的实时功耗估算模型,该模型可以直接移植到支持 SMT 的平台上.通过 PARSEC 和 SPLASH2 两个基准测试程序集进行了验证,估算误差分别为 3.01%和 1.99%.针对建模耗时长的问题,提出了基于两阶聚类的优化改进方法.所提出的估算模型能为构建具有动态平衡功耗和平滑峰值功耗的智能功耗感知系统提供借鉴.

关键词: 功耗估算;性能计数器;多核

中图法分类号: TP303

中文引用格式: 刘辛,沈立,苏博,王志英.多核处理器的功耗估算模型.软件学报,2015,26(7):1840−1852. http://www.jos.org.cn/1000-9825/4706.htm

英文引用格式: Liu X, Shen L, Su B, Wang ZY. Power estimation model on multi-core platforms. Ruan Jian Xue Bao/Journal of Software, 2015,26(7):1840−1852 (in Chinese). http://www.jos.org.cn/1000-9825/4706.htm

Power Estimation Model on Multi-Core Platforms

LIU Xin, SHEN Li, SU Bo, WANG Zhi-Ying

(State Key Laboratory of High Performance Computing (College of Computer, National University of Defense Technology), Changsha 410073, China)

Abstract: Accurate power consumption estimation can provide a significant guidance for OS scheduling and software/hardware power efficiency optimization. Previous researches have indicated that power consumption can be estimated by monitoring the related hardware events inside the CPU, such as instruction submission times and caches access times. However, those models which are based on hardware are not able to provide accurate results; they often come with an error over 5%. This study first analyzes the hardware events provided by the CPU, then chooses a set of events that are closely related to power consumption, and finally uses step by step multi-element linear regression analysis to build our run-time estimation model. This model is not related to any applications and can be directly transformed into the platforms that support SMT. The model is verified with the two benchmark suites PARSEC and SPLASH2, resulting in estimated errors of 3.01% and 1.99% respectively. To address the issue of high time consuming in modeling, an optimization scheme with two-step cluster is also presented in this article. The proposed estimation model can serve as a foundation for the intelligent power consumption perception systems that dynamically balance power assignment and smooth peak power consumption at run-time.

Key words: power estimation; performance counter; multi-core

“功耗墙”是当前计算机系统设计时面临的重要挑战之一.首先,随着工艺尺寸的减小和集成度的提高,处理器功耗急剧增加,已成为制约性能进一步提升的瓶颈;其次,高功耗带来的高热量会导致处理器温度上升,严重

* 基金项目: 国家高技术研究发展计划(863)(2012AA010905); 国家自然科学基金(61472431, 61272143)
收稿时间: 2014-05-07; 定稿时间: 2014-07-29

影响计算机系统的可靠性.相关研究成果表明:温度每升高 10°C[1],芯片失效概率便会提高 1 倍;再者,功耗还直接影响计算机系统的设计和运行成本.对于嵌入式系统,高功耗意味着大容量电池的需求,这会大幅增加成本并严重影响设备的便携性.对于桌面系统和高性能计算系统,功耗还决定了芯片的封装技术及散热装置.2013 年,天河 2 号超级计算机以惊人的计算能力荣登并卫冕 Top500[2]排行榜第一,其整机功耗为 17.808MV,打开水冷系统后达到 24MV,正常工作 1 小时的电费高达十几万元.在这个提倡"绿色、环保、节能"的新时代,如何降低功耗、开发环境友好型处理器,早已成为工业界和学术界关注的焦点.Green500[3]以性能功耗比作为超级计算机排行的依据,意在引导人们从单纯追求高性能转向高效能,在功耗和性能之间寻找平衡点.

精确地获得系统功耗数据,是研究功耗相关问题的一个重要前提.功耗数据可以通过在芯片上集成额外的功耗测量电路获得[4],但是这种方法需要额外的硬件电路,增加了处理器的制造成本.另外一种简单、易行的办法是:通过万用表探测主板供电线的电流电压,再通过计算获得功耗.但这种方法的精度受到测量仪器的限制,极易受到环境因素的干扰.在多核处理器中,由于多个处理器核心共享同一个电源层(power plane),上述两种方法均只能在芯片级测量功耗,难以区分不同处理器核的功耗.建立功耗模型则是另一种常用的方法,并且可以对单个处理器核心甚至功能部件的功耗进行建模.集成电路的功耗分析早已成为研究的热点,在电路级、逻辑门级、RT 级、体系结构级、系统级等抽象层次,都有大量的文献对相关功耗模型及优化方法进行了讨论[5−10].不同层次的功耗模型均存在着速度与精度之间的矛盾,抽象层次越高,模型的计算速度越快,但是精度越差;相反,抽象层次越低,能获取更多的电路层细节信息,因此更接近于真实的芯片,但速度较慢.

本文利用处理器中现有的硬件性能计数器,通过分析处理器的能耗分布,选择了一组与程序运行功耗密切相关的事件,应用数理统计的方法分别对动态功耗和静态功耗进行建模,最终建立了一个实时的、与应用无关的功耗估算模型,并对模型进行了验证和优化.

本文第 1 节对功耗模型作宏观介绍.第 2 节详细阐述参数模型的构造过程.第 3 节描述模型的具体实现,并进行验证、分析与优化.第 4 节介绍国内外相关工作的研究进展并进行对比分析.第 5 节对本文工作进行总结,并给出以后的研究方向.

1 引 言

为了进行正确性调试和性能优化,现代主流处理器中都集成了硬件性能计数器(performance counter),用于监测系统中各种各样的硬件事件,例如 CPU 运行周期数、提交的微指令数、各级 Cache 的失效/命中次数等等.这些硬件事件的实时性强,精确度高,甚至可以达到时钟级精度.处理器厂商提供专门的 API 供用户读取这些计数器的值,并基于这些计数器开发了一系列性能优化工具,如 Intel VTune.此外,为了便于用户使用性能计数器,目前出现了许多用于简化性能计数器配置过程的工具.处理器为性能计数器预留了专用的统计模块,不会对流水线产生影响,也不会占用 Cache 等存储资源.设计这些计数器的最初目的是为了调节性能,直到后来,Frank 发现某些硬件事件与动态功耗之间存在强线性关系[11].在了解了各硬件事件的触发条件后,可以通过采样性能计数器,利用硬件事件的数据值构建功耗估算模型.一个优秀的功耗模型不仅可以反映被测程序的实时功耗,还可以作为优化调度策略的依据.例如:操作系统可以基于程序运行时的功耗进行线程分派和迁移,实现负载平衡.现有的基于性能计数器的功耗模型或多或少存在一些不足,有的是应用相关的,有的建模过程过于复杂,有的可移植性差.最重要的是,大部分模型的估算精度都不够理想.因此,建立一个应用无关、低开销、可移植性好的高精度模型,具有重要的现实意义.

本文构建功耗模型主要分为两个阶段.

1) 离线构建参数模型:将监测程序绑定到处理器核 0 上,核 1~核 3 运行应用程序,监测程序周期性地采样获取每一个核硬件事件信息、实时温度和总功耗,并将数据输入到一个日志文件中;然后,以日志文件作为训练输入,通过逐步多元线性回归拟合出模型参数;
2) 在线实时功耗估算:监测程序继续采样性能计数器和温度,并将数据带入线下建立的参数模型中,计算出实时功耗.

整个实验流程如图 1 所示.

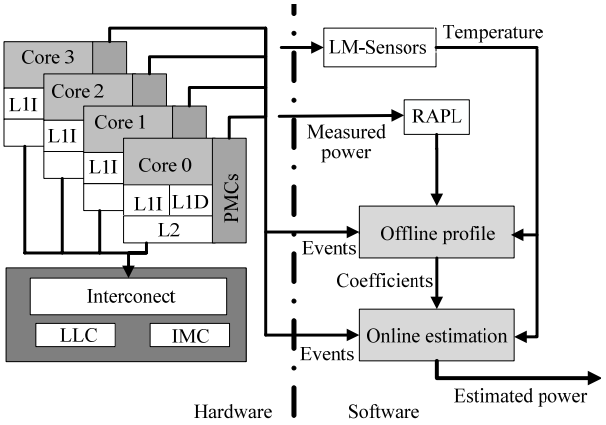


Fig.1 Framework of experiment

图 1 实验流程

2 参数模型

2.1 选择硬件事件

硬件事件的类型、数量直接决定了整个模型的精度和复杂度,所以,硬件事件的选择是构建功耗估算模型中最关键的步骤之一.如果监测的事件数量太多,会增大采样间隔,由此降低数据的精确性,同时也会增加模型的额外开销;相反地,如果事件太少,就不能达到通过硬件事件反映应用程序功耗特征的目的,降低了模型的可用性.现在的处理器通常会提供上百个程序员可见的硬件事件,如何在众多事件中寻找一个冗余度最小但又能充分表征应用功耗特征的事件组,是构建功耗模型的首要问题.

与功耗密切相关的事件通常来自于高能耗的部件,一个定义“良好”的事件组能够尽可能地反映出这些部件的功耗.在微处理器中,时钟逻辑的功耗最大,占总能耗的 40%左右(如图 2 所示)[12],主要包括时钟发生器、时钟驱动器、时钟树和时钟负载,其中,时钟负载所占比重最大;数据通路的功耗位居其次,它包括复杂的功能部件、高频度使用的寄存器文件以及总线,随着片上 Cache 容量的不断增加,其面积往往占据芯片面积的 50%以上,数量众多的晶体管使它的功耗也很高;其他控制逻辑和接口的功耗相对较低.

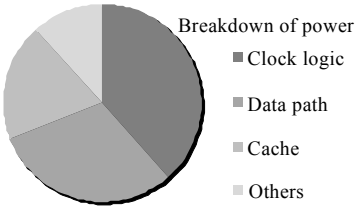


Fig.2 Distribution of power in processor

图 2 处理器中的功耗分布

选择硬件事件的具体步骤如算法 1 所示.第 1 步,根据前面的分析,选出主要能耗部件的相关事件;第 2 步,计算各个事件与动态功耗之间的 Pearson 相关系数,根据表 1[13]中所示的 Pearson 系数与相关性的关系,剔除掉无相关性的事件(相关系数绝对值小于 0.1);第 3 步~第 6 步,通过逐步多元回归分析选择最终的事件组.逐步回归分析的基本思想是:逐步向模型中加入算法第 2 步所确定的事件,进行多元线性回归分析,并对模型中各事件进行 F 检验,若经检验后该变量是显著的,便将这个事件加入到功耗模型中.同时,每次引入一个新的事件后都要

对模型中已有的事件进行检验,如果之前被引入的某个事件由于新事件的加入而变得对功耗的作用不再显著,便从模型中剔除.为了消除多元回归分析中的多重共线性问题,在每次引入新事件后,还要进行 T 检验,删除不显著的事件,直到既不能剔除已有事件也不能加入新的事件为止.这个反复迭代的过程就保证了模型中的所有事件都是显著的,而且回归方程是“最优”的.

算法 1. 事件选择算法.

输入:所有硬件事件集合;

输出:精简的事件集合.

Step 1. 根据图 2 选择出与主要能耗部件相关的硬件事件.

Step 2. 计算事件与动态功耗的 Pearson 相关系数 r,去掉$|r|<0.1$ 的事件.

Step 3. 每次向模型中增加入一个硬件事件,并以事件集作为解释变量,动态功耗作为被解释变量进行多元线性回归分析.

Step 4. 对回归方程中的每个事件进行 F 检验:当硬件事件与实测功耗的 F 检验概率大于 F_{in},则将该事件引入模型;当引入新的事件导致模型中已有的事件的 F 检验概率小于 F_{out},则将其从模型中剔除;

Step 5. 为消除多重共线性,如果引入一个新的事件后,旧的事件的 t 检验值不再显著,则将旧变量剔除.

Step 6. 重复 Step 3~Step 5,直到模型中事件数量不再发生变化为止.

Table 1 Explanation of correlation for Pearson coefficient

表 1 Pearson 系数的相关性解释

相关性	负	正
无	−0.09~0.0	0.0~0.09
弱	−0.3~−0.1	0.1~0.3
中	−0.5~−0.3	0.3~0.5
强	−1.0~−0.5	0.5~1.0

通过上述步骤,最终选取了 11 个典型的硬件事件(见表 2),用于建立功耗估算模型,它们的含义分别为:

- UNHALTED_CORE_CYC.CPU 的动态功耗在其处于不同状态时差别很大,必须加以区分.当 CPU 处于 halt 状态时,操作系统会通过 HLT 指令发送 Clock Gating 信号,使一部分部件进入休眠状态以达到节能的目的.这一事件描述了 CPU 处于 unhalt 状态的时钟周期数;
- INSTRUCTION_RETIRED.这个事件统计了已确认的指令数量,可用于度量指令流水线的吞吐率.需要注意的是:这个事件所记录的指令数不包含分支预测失败的指令,必须与其他计数器配合,才能准确描述指令流水线的功耗特点;
- MISPREDICTED_BRANCH_RETIRED.对于支持分支预测的指令流水线,一旦发现预测错误,就会排空(flush)流水线,并沿着正确的分支路径重新执行,这部分操作引发的功耗是不能被忽略的;
- BUS_CYCLE.系统总线是处理器和内存之间进行数据交换的通道.由于二者速度不相匹配,大量的总线周期说明 CPU 常常无法从 Cache 中获得所请求的数据,不得不访问速度慢很多的主存,严重影响了程序的性能;
- DTLB_MISS.TLB 一旦失效,为了进行虚拟地址到物理地址的映射转换,就必须要访问物理内存中所存放的页表,增加一次额外的访存操作;
- MEM_UOP_RETIRED:ALL_LOAD/ALL_STORE.这两个事件记录 Load 和 Store 操作次数,它们可以反映 load/store buffer 以及 L1 cache 的功耗;
- ICACHE:MISSES/DCACHE:MISSES.分别表示 L1 指令 Cache 和数据 Cache 的未命中次数.它们的和就是 L2 Cache 的访问次数,反映了 L2 Cache 的功耗;
- OFFCORE_REQUESTS/OFFCORE_REQUESTS_BUFFER.随着核外 Cache 容量的不断增大以及 SOC 部件集成度的提高,核外(offcore)部件的能耗越来越大[20].这两个事件记录了发送到核外的请求数.

Table 2 Selected hardware events in model

表 2 模型所选择的硬件事件

Unit	Event
Clock logic	UNHALTED_CORE_CYC
Data path	INSTRUCTION_RETIRED MISPREDICTED_BRANCH_RETIRED BUS_CYCLE DTLB_MISS
Cache	MEM_UOP_RETIRED:ALL_LOADS MEM_UOP_RETIRED:ALL_STORES ICACHE:MISSES DCACHE:MISSES
Others	OFFCORE_REQUESTS OFFCORE_REQUESTS_BUFFER

2.2 基于硬件事件的功耗模型

指令在访问各功能部件时将产生一定量的功耗,尽管这些部件的功耗大小依赖于电平的翻转,本文假设每次访问同一个功能部件所产生的功耗为一个常量.据此,该部件在一段时间内的动态功耗可以由访问频率乘以每次访问的功耗获得.用 $P(U_i)$表示部件 U_i 的动态功耗,则可表示为 $P(U_i)=Act_i\times avgP_i$.其中,$Act_i$ 为部件 U_i 的访问频率,可以代表部件活跃度;$avgP_i$ 为每次访问 U_i 的平均功耗.将各主要能耗部件的动态功耗线性相加,就可以获得总的动态功耗.

$$P_{dynamic}=\sum_{i=1}^{k}P(U_i) \tag{1}$$

以硬件事件为自变量、实测的实时功耗为因变量,通过逐步多元线性回归分析,可获得以下功耗估算模型:

$$\vec{P}_{dynamic}=\sum_{i=1}^{cores}\vec{A}_i\vec{W}+\vec{C}+\vec{\varepsilon} \tag{2}$$

其中,

- $\vec{P}_{dynamic}=\begin{pmatrix}P_1\\P_2\\\vdots\\P_m\end{pmatrix}_{m\times 1}$ 为被解释变量,也就是动态功耗;
- $\vec{W}=\begin{pmatrix}W_1\\W_2\\\vdots\\W_n\end{pmatrix}_{n\times 1}$ 为待估参数,用于表征能耗部件的平均访问功耗;
- $\vec{C}=\begin{pmatrix}C_1\\C_2\\\vdots\\C_m\end{pmatrix}_{m\times 1}$ 为常量,尽管所选的事件集并不能覆盖所有的处理器部件,但模型已包含了主要的能耗部件,未被监测的部件功耗相对较小,所以本文假设这部分功耗恒定不变,用 $\vec{C}$ 作为近似值;
- $\vec{A}_i=\begin{pmatrix}A_{i11}&A_{i12}&\cdots&A_{i1n}\\A_{i21}&A_{i22}&\cdots&A_{i2n}\\\vdots&\vdots&\ddots&\vdots\\A_{im1}&A_{im2}&\cdots&A_{imn}\end{pmatrix}_{m\times n}$ 为解释变量,是部件的访问频率,通过采样性能计数器获得.其中,$A_{ijk}=S_{ijk}/Sampling\ Interval$,$S_{ijk}$ 是第 i 个处理器核中第 j 个硬件事件的第 k 次观测值,$Sampling\ Interval$ 是采样周期;

- $\vec{\varepsilon}=\begin{pmatrix}\varepsilon_1\\ \varepsilon_2\\ \vdots\\ \varepsilon_m\end{pmatrix}_{m\times 1}$ 为随机误差.

建模过程实际上是对公式(2)中的参数进行估计,根据最小二乘原理求解 $\vec{W}$, 使得实际功耗与估算功耗的残差平方和 Q 达到最小值.

$$Q=\sum_{i=1}^{m}(P_{dynamic\_i}-\hat{P}_{dynamic\_i})^2 \tag{3}$$

3 实验与测评

3.1 测试平台

本文以 Intel Core I5 2300 和 I3 2130 两款处理器为实验对象,对第 2 节中的功耗模型进行了验证.其中,I5 2300 为 4 核 4 线程结构;I3 2130 支持 SMT,是双核 4 线程结构.表 3 列出了这两款处理器的详细参数.它们的 L1 Cache 均采用指令 Cache 和数据 Cache 分离的哈佛结构,每个核拥有私有的 L1 Cache 和 L2 Cache,L3 Cache 被所有处理器核共享.所有的实验均在 Ubuntu13.04 操作系统(内核版本 3.8.0)下完成.

Table 3 Configuration of experiment platform

表 3 实验平台配置

Processor	Intel i5 2300	Intel i3 2130
# of core	4	2
# of thread	4	4
L1 caches	64KB Instruction,64KB Data,私有	32KB Instruction,32KB Data,私有
L2 cache	256KB,私有	256KB,私有
L3 cache	6MB,共享	3MB,共享
Size	32mn	32mn
TDP	95W	65W
Clock speed	最高 2.8GHZ,最低 1.6GHZ	最高 3.4GHZ,最低 1.6GHZ

3.1.1 采样性能计数器

通过事件编码及掩码配置 MSR(model specific register)可以获取性能计数器的值,但目前多种成熟的采样性能计数器的工具已被广泛使用,如 PAPI,Perfsuit 和 Perfmon.本文使用了 libpfm4 函数库,它提供了线程级以及 CPU 级的计数器接口,将事件名称自动转换为事件编码.这些接口十分灵活,开销很低,因而可以方便地集成到用户程序中.

采样间隔与模型中的事件数量有关,并且直接影响了功耗模型的精度.由于模型中选用了 11 个硬件事件,但是实验平台只提供了 4 个可配置的性能计数器,为此,本文采用了分时复用的方法,即:为每个处理器核分配一个计数器,在同一个采样周期内各个核对某个事件同时进行采样,每个事件将获得 20ms 的时间片.由于需要对 11 个事件轮询采样,所以采样间隔为 11 个采样周期.之所以可以使用分时复用,是由于程序的时间局部性原理.我们可以认为:在采样间隔内,程序的运行行为是相似的.增加事件数量可以获得更多的功耗特征,从而更接近于真实的功耗,但另一方面会增大采样间隔,使得采样精度下降,所以我们需要对模型中事件的数量进行权衡.

3.1.2 实时功耗测量

Intel 在 Sandy Bridge 构架下集成了 RAPL[14]模块,提供了获取实时功耗的接口.通过 RAPL,我们可以监测和控制不同层次的功耗,例如 Package、DRAM 控制器、CPU 核等.我们将监测程序绑定到线程 0 上,其他 3 个线程负责运行应用程序.所测功耗可表示为 $P_{measure}=P_{dynamic}+P_{idle}$,其中,$P_{measured}$ 为 RAPL 获取的实时功耗;$P_{dynamic}$ 为动态功耗;P_{idle} 为空闲时的功耗,$P_{idle}=P_{os}+P_{monitor}+P_{static}$,由静态功耗、监测程序功耗以及操作系统功耗 3 部分组成.由于操作系统功耗比较小,而且很难单独分离出来,而监测程序只是重复记录实时功耗、温度和采样性能

计数器,其运行功耗十分稳定,因此可以假设这两部分功耗为常量.而静态功耗与电压和温度相关,实验中,频率/电压保持不变,所以只考虑其随温度的变化.我们首先让处理器接近满负载运行,使其快速升温.当温度恒定,处理器处于发热/散热平衡后,停止所有应用程序,监测程序开始记录实时功耗,并且调用 LM-sensors 获取 CPU 实时温度,直到温度不再发生变化.LM-sensors 是针对 Linux 系统的一款硬件监测工具,可以实时监测 CPU 的温度变化.测试之前,我们在 BOIS 中关闭了 Intel 的睿频技术,并且固定了 CPU 运行频率.通过线性回归来拟合 P_{idle} 与温度之间的关系,最终可获得 P_{idle} 与温度的关系式:$P_{idle}=C_1\times T+C_2$,C_1 和 C_2 均为常数,T 为所有核的平均温度.虽然在理论上静态功耗与温度呈指数关系,但是经过实测,在本文的实验环境下,处理器温度变化范围在 15°C 以内,而在温度变化范围较小的情况下将其简化成线性模型,不会引入太大误差[15].

3.1.3 测试用例

为了保证估算模型的通用性,训练集应尽可能全面地覆盖所选事件集的全空间.SPEC 2006[16]是一个综合性基准程序集,它包含了来自工作站、科学计算以及企业等多个领域的应用,能够对处理器的各个子系统进行强化训练.所以我们选取 SPEC 2006 作为训练集,但是由于该测试集均为单线程应用,我们在对每个程序进行测试时,分别运行 1~3 个程序副本(每个核绑定一个程序副本).

为了增加模型可信度,建立模型的训练集与验证模型所用的测试集应为不相交的程序.于是,我们使用了 PARSEC[17]+SPLASH2[18]作为测试集.PARSEC 基准测试集被广泛应用于评估 CMPs 的性能,由多线程程序组成.一些学者证明了 PARSEC 与 SPLASH2 在指令 footprint 以及工作集大小等结构特征方面存在很大差异[19],同时使用这两个基准程序集作为测试集,可以通过上述几个方面进行互补,更具说服力.

3.2 实验结果及分析

3.2.1 模型的精度

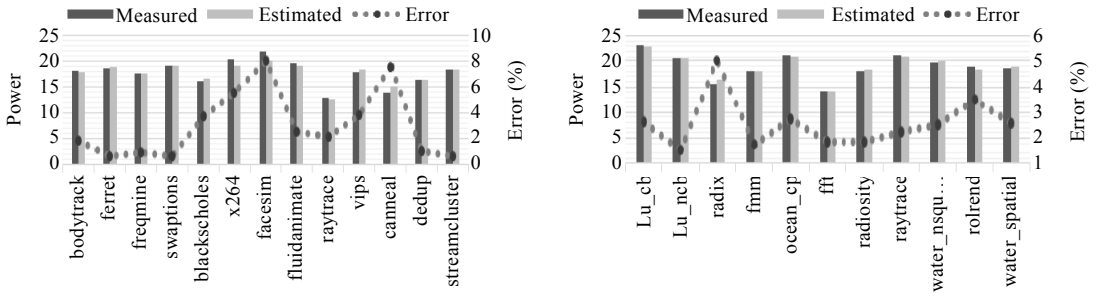
图 3(a)、图 3(b)比较了在 I5 平台上测试程序的估算功耗和实测功耗.总体上看:我们的模型能够精确估算动态功耗,对 PARSEC 和 SPLASH2 两个程序集的平均估算误差(百分比)分别为 3.01%和 1.99%.83%的测试程序的估算误差在 5%以内.运算密集型程序的估算功耗更接近于真实功耗,而访存频繁的程序估算误差相对较大,但是对于精度最差的 facesim 程序,误差仍低于 8%.图 4 为运行 1~3 个 Blackscholes(左)和 Streamcluster(右)程序副本的测试结果,从图中可以看到:随着程序副本的增加,功耗逐步上升,形成 3 个明显的功耗 phase[21];而本文提出的模型可自动适应程序运行时不同的 phase.

造成模型误差的主要原因分析如下:

1) 估算模型基于简单的多元线性回归,忽略了非线性因素,例如复杂的访存过程.这是导致产生误差的最主要的因素;
2) 粗粒度的模型对于一些复杂的部件可能并不合适.例如,我们并没有区分加法指令和乘法指令.缺乏对指令类型的区分,将降低 ALU 功耗的估算精度;
3) 由于训练集不够完备,导致模型的学习过程不够充分.我们尽可能地保证训练集的多样性,但是 SPEC2006 主要侧重于 ALU 和 Memory 测试,缺乏 I/O Bound 类应用;
4) 由于硬件计数器数量不足而引入的多路复用机制,将导致采样数据不够精确;
5) 简化的线性模型拟合空闲时的功耗将会产生误差,并且会被传递到动态功耗模型中,甚至在模型学习过程中被放大.

基于以上原因,对于少数应用,本文提出模型的精度还有待提高,例如 canneal 的误差接近 8%.通过分析可以发现:canneal 程序在执行过程中,swap_cost()函数占了 90%的执行时间.该函数主要完成数据交换,由于工作集较大,各级 Cache 的失效率很高,LLC 的作用不够明显,产生大量片外数据传输.上述分析中的第 1)点、第 3)点原因导致估算结果出现较大误差.

为了进一步验证模型的应用无关性,我们交换了测试集和训练集.以 PARSEC 和 SPLASH2 作为训练集,对 SPEC 2006 中程序进行验证,实验结果(如图 5 所示)与前面所获结果十分相似:最差情况下,估算误差仍然没有超过 8%;平均误差为 3.39%.



(a) PARSEC 功耗及估算误差 (b) SPLASH2 功耗及估算误差

Fig.3

图 3

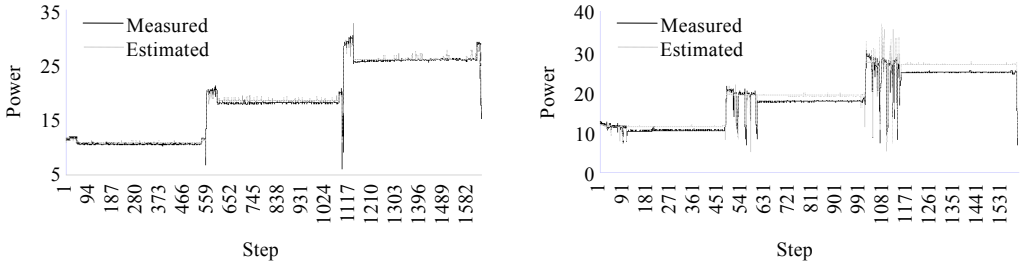


Fig.4 Power phase of Blackscholes (left) and Streamcluster (right)

图 4 Blackscholes(左)和 Streamcluster(右)的功耗 phase

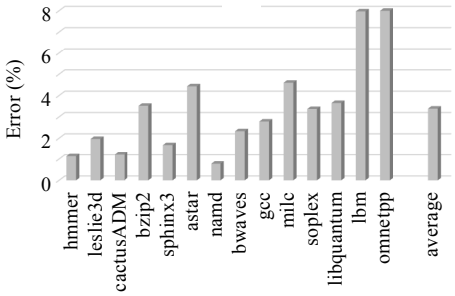


Fig.5 Estimation error of SPEC2006

图 5 SPEC2006 估算误差

尽管存在一定的误差,本文提出的功耗模型仍然可以较为准确地估算处理器的实时动态功耗,而且该模型是独立于应用的,它不需要被测应用的先验信息.也就是说,我们无需在测试前对程序进行预处理.更重要的是,整个建模过程可以很容易地移植到其他具有硬件性能计数器的平台上.

3.2.2 SMT 的影响

I3 2130 支持同时多线程技术,两个逻辑核共享一个物理核,以便开发线程级并行.由于性能计数器支持线程级采样,本文建立的模型可以几乎不用修改就移植到 I3 处理器上.并且在实验过程中发现以下两点规律.

规律 1. 将多个应用实例绑定到同一个物理核中的不同逻辑核,比运行在不同物理核的功耗要低.

这一点是显而易见的,图 6 是 I3 2130 的逻辑核分布图,逻辑核 0、逻辑核 2 和逻辑核 1、逻辑核 3 分别共

享一个物理核.如图 7 所示,设定将应用程序分配到同一个物理核中为紧凑策略,而分布到不同物理核中为松散策略.通过实验发现,使用紧凑策略的功耗总会比松散策略低.虽然同一个物理核中的不同逻辑核拥有自己的取指部件等私有资源,但它们共享了运算单元、L1 缓存、L2 缓存,提高了资源利用率.将应用紧凑地分布在一个物理核中,会使另一个核由于处于空闲状态而进入功耗较低的 C-STATE[22],进而使功耗降低.

规律 2. 对于 Memory Bound 应用,紧凑策略可以有效地减少能耗;但对于 CPU Bound 应用而言,松散策略更节能.

对于 CPU Bound 应用,紧凑策略虽然也可以降低功耗,但是由于不同逻辑核之间竞争共享资源,引发 Cache 失效率上升,导致性能降低,运行时间大为增加,使得总能耗上升.而 Memory Bound 程序可以开发大量线程级并行,通过延时隐藏技术降低性能的损失,运行时间增加的负面影响不足以抵消掉功耗降低带来的好处,使得总能耗降低.

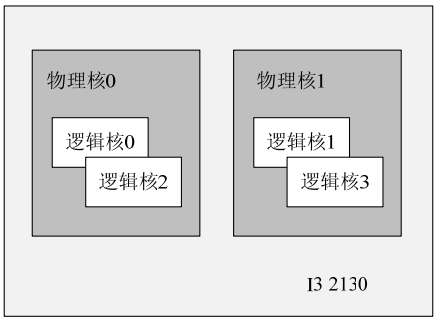


Fig.6 Schematic plot of I3 2130

图 6 I3 2130 示意图

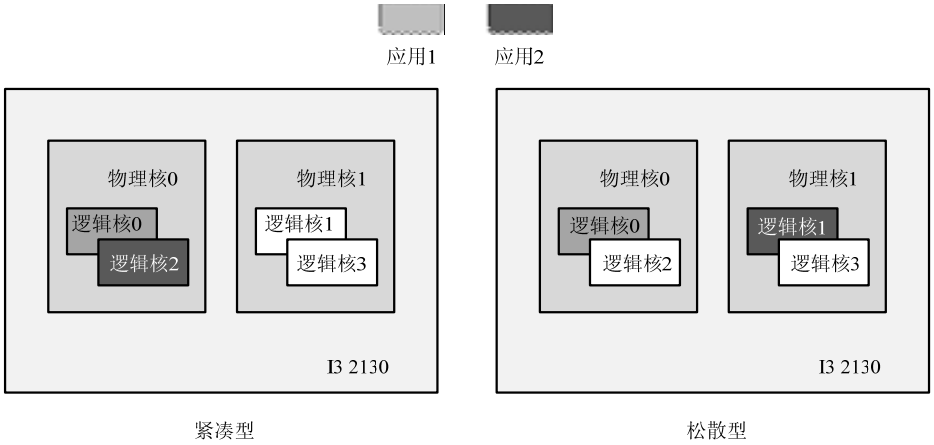


Fig.7 Two policies of task assignment

图 7 两种任务分配策略

3.3 训练时间优化

在实验过程中,我们发现建立模型的过程比较耗时.为了使训练集能够覆盖更多的程序特征,本文选择了程序数量多且类型较为全面的 SPEC 2006 作为训练集,每个程序分别测试了 1~3 个实例,运行完所有程序需要 2~3 天的时间.在选择硬件事件阶段,需要对不同的事件进行不断的尝试,而每次修改都需要对所有程序进行重新测试,这将花费大量的时间.本节针对上述问题,基于聚类分析方法将建模的时间缩减到原来的 1/3.

导致建模时间过长的最主要的原因是测试的程序太多,精简训练集中程序的数量可以有效地提升建模的

速度.但是从前面的分析中可以看出:训练集必须尽可能多地反映各种功耗特征,否则会导致功耗模型对特定的某类程序欠学习,出现较大的误差.为此,我们可以分析训练集中各程序的功耗特征,剔除掉冗余的程序,保留最小集.一方面可以减少训练集中程序的数量,缩短建模时间;另一方面,防止对某类程序的过度学习.通过分析SPEC 2006 中各程序的运行时的硬件事件值,最终以浮点操作数量、Cache 失效数和吞吐率作为分类变量,分别进行两阶聚类,将 SPEC 2006 中的程序分为 8 类(见表 4).再在每一类中随机选择一个代表程序,最终将训练集中的程序精简为 8 个,如此,便在很大程度上缩短了训练时间.

Table 4 Classification of SPEC 2006

表 4 SPEC 2006 的分类结果

分类	程序
1	400.perlbench,464.h264ref,456.hmmer
2	445.gbmk,458.sjeng
3	416.gamess,444.namd,454.calculix
4	434.zeusmp,459.GemsFDTD,433.milc,437.leslie3d,450.soplex,470.lbm
5	435.gromacs,436.cactusADM
6	401.bzip2
7	410.bwaves,482.sphinx3
8	403.gcc,462.libquantum,471.omnetpp,473.astar

两阶聚类主要分为如下两步:

(1) 通过聚类特征(CF)构造高度稳定的聚类特征树(cluster feature tree,简称 CFT),将大样本粗略分为若干子类.聚类特征定义为 $CF=(N,\vec{LS},\vec{SS})$, 其中,N 为样本数量, $\vec{LS}=\sum_{i=1}^{N}\vec{x}_i, \vec{SS}=\sum_{i=1}^{N}\vec{x}_i^2$.

首先初始化一个根节点,根据对数似然相似性 $L(J)$,将后续观察量依次归并到与之最相似的节点中,如果没有找到这样的节点,便为该观测量建立一个新的节点;

(2) 以聚类特征树为输入,使用分层聚类进行再聚类,通过施瓦兹贝叶斯信息准则(BIC)以确定最终的聚类数.

为了验证精简训练集后的模型仍然具有良好的估算精度,我们用上述方法从 SPEC 2006 的每一类程序中随机选择一个程序组成训练集(见表 5),分别以 PARSEC,SPLASH2 和未被选作训练集的 SPEC 2006 程序作为测试集进行了验证,结果见表 5.与之前的精度相比,估算误差有所增大,但仍优于大部分现有的功耗模型.

Table 5 Training set and test result

表 5 训练集和测试结果

组号	程序	误差(%)		
		剩余 SPEC 2006	PARSEC	SPLASH2
1	400.perlbench,445.gbmk,416.gamess,434.zeusmp, 435.gromacs,401.bzip2,410.bwaves,403.gcc	2.91	3.10	3.76
2	464.h264ref,458.sjeng,444.namd,459.GemsFDTD, 436.cactusADM,410.bwaves,482.sphinx3,462.libquantum	3.20	3.53	2.84
3	456.hmmer,445.gbmk,454.calculix,437.leslie3d, 436.cactusADM,401.bzip2,482.sphinx3,473.astar	2.92	3.58	2.34

4 相关工作

Frank Bellsoa 是借助硬件事件建立功耗模型的鼻祖,他首次观察并通过实验验证了硬件事件与功耗之间存在线性关系[10],随后,使用硬件活跃度来估算线程级功耗.自此以后,国外越来越多的学者开始致力于通过硬件事件建立功耗估算模型,但是国内相关研究相对缺乏.

目前已有的基于性能计数器的功耗模型可以被分为两类:一类利用 Top-Down[23]方法,使用数量较少的硬件事件(通常与性能计数器数量一致),以建立一个快速、简单、低开销的模型为目标,建立一个系统级的功耗模型;与之相反,另外一种 Bottom-Up 模型[23]依据微体系结构级的耗能部件,以增加模型复杂度为代价换取模型精度,通过大量的硬件事件来收集尽可能多的信息,以反映应用程序的功耗特征.两种模型各有利弊,本文提出的

模型结合了这两者的优点,在模型复杂度和精度之间进行了折中.

Singh[24]采用 Top-Down 的方法,针对多线程、多道程序负载建立了处理器核级的功耗模型.由于其实验平台只有 4 个性能计数器,Singh 将硬件事件分为浮点部件、指令确认、停顿和存储器相关这 4 类,分别计算每个事件与功耗的 Spearman 秩相关系数,在每一类中选取相关系数最大的事件进行建模.通过收集执行 Micro-benchmark 的事件信息,建立了一个分段的线性回归模型,以 NAS,SPECOMP 和 SPEC 2006 作为测试集,分别获得 5.8%,3.9%和 7.2%的平均误差.与本文所建立的模型相比,他们的功耗模型依赖于程序的运行时特征,由分段函数构成,而且他们并没有给出相应的理论依据.

Isci 和 Martonosi[25]基于功能部件将芯片划分为 22 个耗能模块,再分别以相关的硬件事件为每一个模块建立功耗模型.他们有针对性地编写了 Micro-benchmarks,强化训练各个子模型.这是典型的 Bottom-Up 建模方法,通过引入大量的硬件事件可以提高模型精度,但是由于现代处理器越来越精细、越来越复杂,各功能部件相互紧密耦合,很难找到一个合适的划分模块的粒度,建模复杂度大大提高;另一方面,同时监测大量硬件事件势必带来较大的额外开销,不利于进行在线估算;再者,该模型严重依赖于实验平台的微体系结构,可移植性较差.而本文在更高的层面上统一构建模型,避免过度细化功能部件之间的差异性,简化了建模的过程.

将本文所建立的模型与已有模型进行对比(见表 6),可以看到,我们的模型集成了 Top-Down 和 Bottom-Up 两种方法所具有的优点.首先,通过自下而上筛选事件,再通过数理统计方法自顶向下建立最终的模型,避免了为每一个功能部件单独建模,有效地降低了建模的复杂度;更重要的是,我们获得了与 Bottom-Up 方法相当的精度;通过对训练集中的程序进行聚类分析,精简训练程序,进一步减少了模型离线学习所需要的时间,而误差只增加了 1%左右.

Table 6 Comparison between different models

表 6 不同功耗模型之间的对比

模型	方法	平台	事件数	误差	应用相关
Karan 的模型[21]	Top-Down	AMD Phenom 9500	4	5.63%	是
TDC 模型[20]	Top-Down	Intel Core TM 2 Duo	4	4.53%	否
BU 模型[20]	Bottom-Up	Intel Core TM 2 Duo	9	2.53%	是
Isci 的模型[22]	Bottom-Up	P4	22+	Around 3W	否
MICRO 模型[26]	Bottom-Up	Intel Core TM 2 Duo	20	2.85%	否
本文的模型	综合	Intel I5	11	2.5%	否

5 结束语

目前的 DVFS(dynamic voltage and frequency scaling)和线程迁移均可以利用实时功耗数据作为调度依据,而实时的功耗数据可以来自功耗模型或实际测量.一般来说,所获得的功耗数据越精确,就越能有效地避免调度误差或抖动,从而达到理想的调度效果.另一种新兴的应用场景是 Power Capping[27],在这种情况下,总功率具有上限.通过本文中的模型求出实时功耗,并与功耗上限相比较,使实际总功耗和功耗上限的差距保持在一个比较小的范围内,使两者在总体上保持一致.另外,本文中的模型可以区分各个不同处理器核的功耗,可以实现实时地、按比例地将功耗分配给不同的应用,这在功耗成本很高的数据中心中将获得巨大的经济效益.

本文提出了一种通过硬件性能计数器估算处理器动态功耗的通用方法,实现了一个简单而准确的估算模型.该模型不需要对被测程序进行预处理,而且可以移植到支持同时多线程的平台上.实测结果表明,该模型的精度优于目前已有的大部分功耗估算模型(见表 6).在今后的工作中,我们将从两个方面重点优化访存密集型应用的估算精度:一是在选择训练集时引入访存密集型的程序,这样,经过前期的学习阶段后,模型将更加适应该类应用.另外,还可以增加访存相关的硬件事件,以获取更多的访存特征,减少访存密集型应用的功耗估算误差,但这有可能增加模型的复杂度.为了进一步验证本文所提方法的平台独立性,我们还计划将该模型移植到 AMD 处理器上进行测试和验证.

References:

[1] Yang P, Chern JH. Design for reliability: The major challenge for VLSI. Proc. of the IEEE, 1993,81(5):730−744. [doi: 10.1109/5.220904]

[2] TOP 500 SUPERCOMPUTER SITES. 2013. http://www.top500.org/list/2013/06/

[3] THE GREEN500 SITES. 2013. http://www.green500.org/

[4] Lefurgy C, Wang X, Ware M. Server-Level power control. In: Proc. of the 4th IEEE Int'l Conf. on Autonomic Computing (ICAC). IEEE, 2007. 4−4. [doi: 10.1109/ICAC.2007.35]

[5] Joseph R, Martonosi M. Run-Time power estimation in high performance microprocessors. In: Proc. of the 2001 Int'l Symp. on Low Power Electronics and Design. ACM Press, 2001. 135−140. [doi: 10.1145/383082.383119]

[6] Smith LD, Anderson R, Roy T. Power plane SPICE models and simulated performance for materials and geometries. IEEE Trans. on Advanced Packaging, 2001,24(3):277−287. [doi: 10.1109/6040.938294]

[7] Chen B, Nedelchev I. Power compiler: A gate-level power optimization and synthesis system. In: Proc. of the '97 Int'l Conf. on Computer Design: VLSI in Computers & Processors. Austin: IEEE, 1997. 74−79. [doi: 10.1109/ICCD.1997.628852]

[8] Martonosi M, Tiwari V, Brooks D. Wattch: A framework for architectural-level power analysis and optimizations. In: Proc. of the 27th Int'l Symp. on Computer Architecture (ISCA). Vancouver: ACM Press, 2000. 83. [doi: 10.1145/339647.339657]

[9] Tiwari V, Malik S, Wolfe A, Lee MTC. Instruction level power analysis and optimization of software. Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology, 1996,13(2-3):223−238. [doi: 10.1109/ICVD.1996.489624]

[10] Tiwari V, Malik S, Wolfe A. Power analysis of embedded software: A first step towards software power minimization. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 1994,2(4):437−445. [doi: 10.1109/92.335012]

[11] Bellosa F. The benefits of event: Driven energy accounting in power-sensitive systems. In: Proc. of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System. Kolding: ACM Press, 2000. 37−42. [doi: 10.1145/566726.566736]

[12] Zhang J, Fan XY, Liu SH. Research of lowpower design techniques for multi-core and multithreading microprocessor. Computer Science, 2007,34(10):301−305 (in Chinese with English abstract). [doi: 10.3969/j.issn.1002-137X.2007.10.079]

[13] 2013. http://zh.wikipedia.org/wiki/%E7%9A%AE%E5%B0%94%E9%80%8A%E7%A7%AF%E7%9F%A9%E7%9B%B8%E5%85%B3E7%B3%BB%E6%95%B0#cite_note-Buda-4

[14] David H, Gorbatov E, Hanebutte UR, Khanna R, Le C. RAPL: Memory power estimation and capping. In: Proc. of the 2010 ACM/IEEE Int'l Symp. on Low-Power Electronics and Design (ISLPED). Austin: ACM/IEEE, 2010. 189−194. [doi: 10.1145/1840845.1840883]

[15] Liu Y, Dick RP, Shang L, Yang HZ. Accurate temperature-dependent integrated circuit leakage power estimation is easy. In: Proc. of the Conf. on Design, Automation and Test in Europe. Nice: ACM Press, 2007. 1526−1531. [doi: 10.1109/DATE.2007.364517]

[16] Henning JL. SPEC CPU2006 benchmark descriptions. ACM SIGARCH Computer Architecture News, 2006,34(4):1−17. [doi: 10.1145/1186736.1186737]

[17] Bienia C, Kumar S, Singh JP, Li K. The PARSEC benchmark suite: Characterization and architectural implications. In: Proc. of the 17th Int'l Conf. on Parallel Architectures and Compilation Techniques. Toronto: ACM Press, 2008. 72−81. [doi: 10.1145/1454115.1454128]

[18] Stanford Parallel Applications for Shared Memory. The modified SPLASH-2 benchmark suit. 2007. http://www.capsl.udel.edu/splash/

[19] Bienia C, Kumar S, Li K. PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors. In: Proc. of the IEEE Int'l Symp. on Workload Characterization (IISWC). Seattle: IEEE, 2008. 47−56. [doi: 10.1109/IISWC.2008.4636090]

[20] Gupta V, Brett P, Koufaty DA, Reddy D, Hahn S, Schwan K, Srinivasa G. The forgotten 'uncore': On the energy-efficiency of heterogeneous cores. In: Heiser G, ed. Proc. of the 2012 USENIX Annual Technical Conf. USENIX Association Berkeley, 2012. 367−372.

[21] Isci C, Martonosi M. Identifying program power phase behavior using power vectors. In: Proc. of the 2003 IEEE Int'l Workshop on Workload Characterization (IISWC). White Plains: IEEE, 2003. 108−118. [doi: 10.1109/WWC.2003.1249062]

[22] Intel® 64 and IA-32 Architectures Software Developer's Manual. Volume 3A: System programming guide, part 1. 2011. http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.html

[23] Bertran R, Gonzàlez M, Martorell X, Navarro N, Ayguadé E. Counter-Based power modeling methods: Top-Down vs. bottom-up. The Computer Journal, 2013,56(2):198−213. [doi: 10.1093/comjnl/bxs116]

[24] Singh K, Bhadauria M, McKee SA. Real time power estimation and thread scheduling via performance counters. ACM SIGARCH Computer Architecture News, 2009,37(2):46−55. [doi: 10.1145/1577129.1577137]

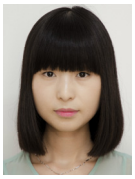
[25] Isci C, Martonosi M. Runtime power monitoring in high-end processors: Methodology and empirical data. In: Proc. of the 36th Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO), Vol.93. Vancouver: IEEE/ACM, 2003. [doi: 10.1109/MICRO.2003.1253186]

[26] Bertran R, Gonzalez M, Martorell X, Navarro N, Ayguadé E. Decomposable and responsive power models for multicore processors using performance counters. In: Proc. of the 24th ACM Int'l Conf. on Supercomputing (ICS). Tsukuba: ACM Press, 2010. 147−158. [doi: 10.1145/1810085.1810108]

[27] Ma K, Wang X. PGCapping: Exploiting power gating for power capping and core lifetime balancing in CMPs. In: Proc. of the 21st Int'l Conf. on Parallel Architectures and Compilation Techniques. New York: ACM Press, 2012. 13−22. [doi: 10.1145/2370816.2370821]

附中文参考文献:

[12] 张骏,樊晓桠,刘松鹤.多核、多线程处理器的低功耗设计技术研究.计算机科学,2007,34(10):301−305. [doi: 10.3969/j.issn.1002-137X.2007.10.079]



刘辛(1990－),女,四川泸州人,硕士生,主要研究领域为高性能处理器体系结构,运行时技术.

苏博(1987－),男,博士生,主要研究领域为计算机系统能效优化,低功耗电路设计.

沈立(1974－),男,博士,副教授,CCF 高级会员,主要研究领域为多核/众核体系结构,编译技术,运行时技术.

王志英(1956－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为高性能处理器体系结构,运行时技术,异步处理器设计.