面向代码的软件能耗优化研究进展

宋 杰[1] 孙宗哲[1] 李甜甜[2] 鲍玉斌[2] 于 戈[2]

[1](东北大学软件学院 沈阳 110819)

[2](东北大学计算机科学与工程学院 沈阳 110819)

摘 要 面向代码的软件能耗优化从程序设计和编码角度优化软件系统的能耗,能够很好地弥补面向硬件和面向资源的能耗优化方法过多依赖硬件环境、普适性较差、粒度过大,且难以在软件开发过程中应用等缺点.该文综述了近年来面向代码软件能耗优化领域的主要研究成果,总结了能耗优化的基本方法和技术层次以及面向代码的软件能耗优化基本思路;随后从面向代码的软件能耗估算方法和优化方法两个方面对现有工作加以梳理,逐一介绍了相关优化工具.该文提出了若干进一步研究的问题.首先,该文重定义面向代码的软件能耗评估模型和方法应该满足的特性,并提出代码的运行时能耗(Runtime Energy Consumption)和视在能耗(Apparent Energy Consumption)的概念;其次,该文认为现有面向代码的能耗优化技术过于具体,或针对具体的代码,或针对具体的功能,缺乏抽象层次的优化技术,缺乏算法层面的能耗优化方法,且没有充分考虑编程语言特征,尤其是面向对象语言特征;最后,该文提出算法能耗复杂度这一新观点,指出仍然存在的问题和可能的解决办法.

关键词 面向代码;软件能耗;能耗估算;能耗优化;绿色计算

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2016.02270

Research Advance on Code Oriented Optimization of Software Energy Consumption

SONG Jie[1] SUN Zong-Zhe[1] LI Tian-Tian[2] BAO Yu-Bin[2] YU Ge[2]

[1](*Software College*, *Northeastern University*, *Shenyang* 110819)

[2](*School of Computer Science and Engineering*, *Northeastern University*, *Shenyang* 110819)

Abstract Code oriented optimization of software energy consumption focuses on aspects of programming and coding. It is a good complement to the hardware oriented and resource oriented optimizations because the latter two have the disadvantages of hardware environment depended, weak universality, coast granularity and inapplicable in the software development processes. In this paper, the primary research achievements of code oriented energy consumption optimization are summarized. Firstly, we introduce the basic approaches of energy consumption optimization, the layers of optimization techniques, and the basic principles of code oriented optimization. Then, the related works are explained from two aspects: estimation and optimization. Moreover, challenges in both aspects are reviewed as well as future research trends are predicted. In the paper, we propose several challenges for the further study. First, we redefine the characters of code oriented estimation model and approaches for software energy consumption, then propose the concepts of runtime energy consumption and apparent energy consumption of codes. Second, we think the existing code oriented energy consumption optimization approaches are more specific, and

收稿日期:2015-05-18;在线出版日期:2016-02-26.本课题得到国家自然科学基金重大项目(61433008)和青年基金(61402090,61502090)、中国博士后科学基金面上项目(2013M540232)和教育部博士点基金(20130042120006)资助.**宋 杰**,男,1980 年生,博士,副教授,主要研究方向为高能效计算、大数据管理、云计算.E-mail:songjie@mail.neu.edu.cn.**孙宗哲**,男,1991 年生,硕士研究生,主要研究方向为高能效计算.**李甜甜**,女,1989 年生,博士研究生,主要研究方向为高能效计算.**鲍玉斌**,男,1968 年生,博士,教授,主要研究领域为大数据管理.**于 戈**,男,1962 年生,博士,教授,主要研究领域为数据库理论.

they focus on fixed codes or functions. There is few abstract-level and generalized optimizations, few algorithm or optimizations, and few optimizations which taken the features of programming language, especially the object oriented language, into consideration. Finally, we propose the new challenge of this research topic, which is energy consumption complexity of algorithm, and then, the existing problems and possible solutions, which can be referred by researchers, are given.

Keywords　code orientation; energy consumption of software; energy consumption estimation; energy consumption optimization; green computing

1　引　言

当提及节能减排技术时，人们会想到制造业、交通运输等传统行业，殊不知，计算机等 IT 设备的电能消耗同样不容忽视. 美国 Harvard 大学研究人员就用户电脑耗能和服务器搜索一次的耗能进行计算，形象地指出人们每使用 Google 搜索一次，将消耗可以烧开半壶水的电能①；一个数据中心的能耗高于 100 个商业建筑的能耗[1]；数据中心每年耗电量已达近千亿千瓦时，电费占运营成本的 50% 以上②. 绿色和平组织预测，2020 年全球主要 IT 运营商的能耗将达到 2 万亿千瓦时，超过德、法、加和巴西等 4 国的能耗总和③. IT 企业已经属于能源密集型产业.

从成本角度看，随着计算机设备规模的不断扩大，耗电量剧增，导致运营成本大大增加，能源费用是各个 IT 企业的主要成本. 马化腾表示，2009 年腾讯数据中心全年电费已经等于腾讯所有员工的工资；中国联通数据中心运营成本约占总收入的 60%，能源消耗成本占运营成本的 70%④. 能源价格飙升导致运营成本同比增长，买得起设备供不起电的现象已经出现[2]. 从环境角度看，各种数据表明计算机在吞噬大量能源的同时，不知不觉中给环境带来了沉重压力. 传统能源驱动的 100 MW 发电站每年排放 5000 万吨 CO_2[3]. 在 2008 年，全球接入 Internet 的 IT 资源的年耗电量等同于 14 个大型发电站的年供电量，换算成二氧化碳排放量，相当于全球航空公司一年的碳排放量之和，占温室气体总排放量的 2%⑤. 而且，这些数据都在持续地增加.

成本和环境因素促使能耗优化成为 IT 企业界和学术界共同关注的热门话题，绿色计算逐渐产业化. 报道指出，2015 年全球绿色 IT 服务市场规模达 50 亿美元，亚太地区绿色 IT 服务市场规模达到 20 亿美元，中国、日本和澳大利亚将占据重要份额⑥. 综上，计算机能耗优化问题亟待解决.

能耗归根结底是硬件消耗的，但我们可以从软件层面来更灵活地优化能耗. 软件是能源的最终消费者[4]. 为研究软件特征对能耗的影响，以便从软件层面优化能耗，我们首先定义软件完成特定运算和服务时对应的硬件环境消耗的电能为软件能耗. 如果将系统简单的分为硬件层和软件层，那么软件能耗优化方法则位于软件层，采用软件相关的技术，优化软件能耗[5].

从软件层面优化能耗的方法有很多，本文着重研究面向代码的软件能耗优化技术，也即从编写代码角度来评估和优化软件能耗. 我们在下一节会描述这种优化方法的优势. 本文总结了近年来面向代码的软件能耗优化相关领域的研究进展和最新成果.

本文第 2 节介绍能耗优化的基本思路和分类；第 3 和 4 节分别分析代码能耗估算和优化方法，并在每节小结中提出现存问题和研究方向；第 5 节介绍面向代码的软件能耗优化工具；第 6 节从抽象算法角度提出面向代码的软件能耗优化的新思路和挑战；最后，第 7 节对全文进行总结.

2　优化思路

无论是采用何种优化方法，或是面向何种优化对象，能耗优化思路可以归结为两种：一是减少能源

① http://www.computing.co.uk/ctg/news/1852749/two-google-searches-equivalent-boiling-kettle-scientist, 2015, 4
② http://news.mydrivers.com/1/229/229399.htm, 2015, 4
③ http://www.greenpeace.org/international/en/, 2015, 4
④ http://www.mcplive.cn/index.php/article/index/id/7211/viewall/1, 2015, 4
⑤ http://www.greenpeace.org/international/Global/international/publications/climate/2012/iCoal/HowCleanisYourCloud.pdf, 2012
⑥ http://www.chinanews.com/it/it-itxw/news/2010/06-27/2365506.shtml, 2015, 4

的使用;二是提高能源利用率,减少能源浪费(如设备空载耗能)[6-8].例如:对于一个低功率硬件,若该硬件采用更加节能的材料制作,则属于前者;若该硬件可以在空闲时自动休眠,则属于后者.对于一个低能耗软件,若缩减功能以降低能耗,则属于前者,如果充分利用资源减少硬件空闲,则属于后者.基于该思路,我们采用不同分类规则对现有研究进行归类,一种按优化对象不同,分成软件能耗优化和硬件能耗优化;另一种按优化方法不同,分成静态能耗优化和动态能耗优化;两种分类互有交叉.本节首先介绍硬件和软件能耗优化,随后介绍静态与动态能耗优化,最后概述面向代码的软件能耗优化.

2.1 软件和硬件能耗优化

对于计算机系统,其能耗归根结底是由硬件产生的,但却可以从多个层面来优化.我们把系统分为硬件层和软件层,而又将软件分为平台软件层和应用软件层,图 1 展示了系统各个层面的能耗优化研究.

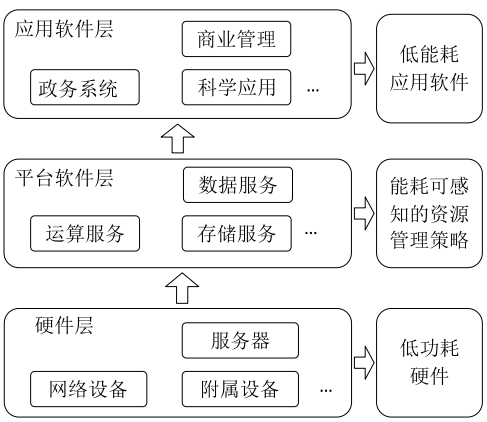


图 1 能耗优化方法位于不同的系统层次

由前文所述优化思路可知,电能由硬件层消耗,因此很多研究在该层考虑硬件设备的能耗特性,开发功耗更低的硬件;然而从另外一个角度,硬件为平台软件提供资源,平台软件是资源的"消费者",通过合理的资源管理能减少资源使用,或提高资源使用率,节能效果更好,尤其是在集群环境下,平台软件能耗优化效果显著[9];同理,平台软件为应用软件提供运算和存储服务,那么本质上应用软件是资源的最终消费者,研究低能耗的应用软件,同样可以达到节能的作用.

应用于平台软件层和应用软件层的能耗优化方法均为软件能耗优化方法.软件能耗优化是绿色软件研究的一个重要部分,软件能耗存在很大优化空间.在性能领域,软件的效率与硬件的效率增长之间的差距越来越大;同理,在能耗领域,硬件功耗降低速度较快,可是软件能耗降低则较为缓慢[10].此外,在可持续发展被高度重视的今天,IT 企业和学术组织更多地考虑计算机硬件和原材料的重复利用等,而较少地关注软件的"可持续发展"[11].而软件能耗优化正是实现"可持续发展"的重要途径,这一点是硬件能耗优化所无法替代的[12].以汽车节油作比喻,研究计算机硬件节能技术好比是研究发动机节油技术,而研究软件节能技术好比是研究驾驶员的习惯(静态)、行车最短路径(动态)、躲避交通高峰(动态)等等,后者能够大幅度地减少空闲油耗,也能够充分发挥各种发动机节油技术.

2.2 静态和动态能耗优化

无论是面向硬件还是软件系统,现有能耗优化方法均可以分为静态和动态两种.动态能耗优化方法根据上下文实时地调节硬件或软件的行为以达到节能的目的[13].动态能耗优化建立在 3 个假设的前提之上:一是系统边界清晰,系统能耗可以测量或是度量;二是系统可以实时动态地调整其能耗相关的状态,如工作状态,资源分配等;三是系统负载在一定程度上是可预测的.组件开关是动态能耗优化主要方法,即关闭空闲组件.硬件节能方法中的"组件"多为芯片;操作系统节能方法中的"组件"多为计算机部件;分布式系统节能方法中"组件"多为计算机节点.此外,动态地减少硬件空闲也是一种可选的方法,如避免 MapReduce 作业中 Reduce 节点等待所有 Map 节点执行结束[14].本文不再逐一详述这些方法.

静态能耗优化方法,顾名思义,是在系统设计之时即被采用的,在系统运行时无法改变的优化方法.从硬件角度,静态能耗优化方法包括针对电路设计、逻辑设计和硬件体系结构的优化.从软件角度,静态能耗优化可以分为面向代码的优化方法、面向软件架构的优化方法以及软件开发过程上的优化(绿色软件工程)[15].我们可以设计低能耗的软件模块,如采用低能耗的算法或低能耗的程序设计方法,直接降低能耗,避免资源空闲.

2.3 面向代码的软件能耗优化

软件能耗优化并非一个崭新的技术领域.传统的能耗优化技术主要应用于嵌入式软件,这些软件多运行在电池供电的硬件环境,在能源受限环境下优化软件能耗可以延长软件的运行时间.但近年来,随着云计算技术推广,能耗优化技术逐渐推广到一

般软件，尤其是那些需要依赖大量服务器的分布式软件.研究软件能耗优化，若按优化软件的种类不同，则会有嵌入式软件、分布式软件、移动应用软件等多种分类，但这些软件均由代码组成，抛开功能和运行环境的差异，面向代码的软件能耗优化具有较好的适用性.

本文针对性能已优化的代码，研究其能耗优化方法，称为面向代码的能耗优化，属于静态的软件能耗优化方法.按优化对象，面向代码的能耗优化又可以具体分为指令级、语句级和模块级三类.例如：指令级优化可采用英特尔编译器提供的“prof-gen”和“prof-use”优化代码能耗；语句级优化可以采用能耗更低的代码结构和数据结构来降低能耗；模块级优化可以通过根据上下文选择能耗更低的实现算法或模块设计方法，来优化能耗.无论优化对象是指令、语句还是模块，面向代码的能耗优化方法大多遵循两个原理.

其一，代码对应的指令越长，指令越复杂，代码的运行时间将越长，能耗越高.即使硬件存在功率波动，指令长度还是执行时间以及执行能耗的主导因素；此外，复杂性高的指令将需要更多的 CPU 资源，CPU 是计算机组件中最为耗能的部分，因此消耗了大量能源.应该尽量减少代码的复杂性，用简单的语句或逻辑替换复杂的语句或逻辑.例如模块封装会增加额外的数据转换和传递运算，如数据格式和数据位置的变换，这些运算都消费额外的能量.

第二，代码对资源的使用率越高，运行时内空闲资源越少，能耗越低.理论上，代码的运行能耗和代码完成的任务量线性正相关，但实际上这种关系是不准确的.代码运行会消耗一些额外能量，对资源的使用率越高，这部分额外能量就越少.这是由于硬件设备空载功率的存在，空闲的资源会浪费能源.给定完成某功能的代码，若资源需求总量不变，资源空闲越少，浪费的能量就越少，能耗就越低，或称为能效（能源效率）越高.比如，高内聚低耦合是软件模块化的重要目标，但弱耦合的模块会增加模块间数据同步的代价，在数据同步时，运算资源的使用率很低，造成能源浪费.

部分研究认为，因为代码编译过程的复杂性和指令的执行顺序对能耗影响的无法预知性，导致从代码级分析和优化大型软件产生的能耗非常困难；而本文梳理和总结了很多静态代码优化以降低能耗的理论和技术，证明该方法同样可行且有效.

3　能耗估算

本节将综述面向代码的软件能耗估算方法.作为软件能耗优化的关键支撑技术之一，软件能耗估算不仅要把软件执行涉及到的硬件能耗映射到软件各组成部分上，更期望建立高层软件特征与硬件能耗的关系，从而易于在软件层面评估和优化系统能耗[16].能耗测量最简单的方法是采用功率和电量测量仪器（如电量计）去直接测量软件执行时的系统能耗，然而，该方法存在以下不足[17]：(1) 电量计都有一定的功率测量范围，普通的电量计测量的最大功率在 2500 W 左右，如果软件的运行在分布式环境，如包含大量用电设备的集群系统，则需要大量电量计，并且电量计的控制、同步和数据汇总都存在难题；(2) 电量计难以区分代码各个部分的能耗差异，如存储代码和运算代码、业务代码和数据访问代码，难以定位高能耗语句，此外也无法区分在分布式环境下每个节点的能源消耗，也即电量计测量能耗的粒度难以控制；(3) 由于测量仪器的精度限制，能耗分析的范围、精度和准确度有限.由此可见，能耗测量是一种黑盒的测量方法，不利于对软件能耗进行细致的评价和优化，而对软件的能耗估算是能耗优化研究的一个重要课题.

本节描述了面向代码的软件能耗估算方法，我们将估算方法分为指令级、语句级和模块级共 3 类.指令级能耗估算将代码执行的每条指令能耗进行加和；语句级估算则以语句作估算单元，研究语句之间的逻辑结构对能耗的影响；模块级则以软件模块，如类、方法等作为估算单元，研究模块间关系对能耗的影响.现存大量指令级能耗估算研究，已较为成熟，本文简要介绍其基本思路，而着重描述语句级和模块级的能耗估算.

3.1　指令级

指令级能耗估算方法首先估计代码生成的指令在目标硬件（如处理器）上的执行能耗，并通过叠加指令能耗来估算代码能耗，这种方法又称做白盒方法，它要求预知硬件信息.白盒方法的优点是通过硬件预描述，能够相对快速和准确地评估代码能耗；缺点是与硬件环境绑定，在许多硬件环境，或解释型执行的语言环境，周期精确的指令集模拟器不可用.该方法大量应用到嵌入式软件[18]，文献[19]对相关技术做了很好的综述.

部分研究人员注意到白盒方法的难度在于代码和指令的映射关系,而视代码为黑盒,单纯地考虑对应指令的执行情况则会更容易.操作系统会给出大量的参数来统计这些指令的执行,但这样做的代价就是无法精确地确定代码和能耗的关系,而代码能耗并非是"估算"而是"测量和预测"得到,且度量粒度也随之变大.例如,文献[20]更加细致地对指令进行划分,并采用不同功能的测试代码(Burn CPU, MemLoop, Network, Tar Kernel, Disk Read, Disk Write)来分析指令和能耗之间的关系,以预测代码能耗.文献[21]研发了一套进程级功率测量工具,这能准确地评估每一个在 Linux 服务器上运行的进程的能源利用情况,进而评价软件系统的能耗.一系列实验表明,该方法对复杂电子商务应用软件的能耗评估准确率到达了 95%.

3.2 语句级

相对于指令级,语句级能耗估算粒度更大,该方法将代码以语句块的形式进行分割,评价每个语句块的能耗.语句级能耗估算需要考虑语句之间的逻辑关系,也即语句结构对能耗的影响.这一点有别于指令级能耗估计.在第 4 节会进一步介绍,语句结构对代码能耗影响是显著的,语句结构的调整是面向代码的能耗优化的重要途径.

语句级能耗估算的一般方法如图 2 所示.首先抽象每条语句以及语句结构的能耗特征,然后分别评价其能耗.对于前者,可以采用指令级方法估算;对于后者,尽管不同的研究做法不尽相同,但都是通过参数的定义和调整以及不同参数组合来表征语句结构对能耗的影响.

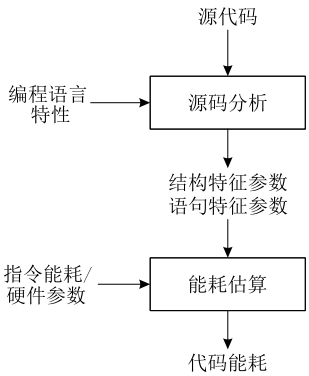


图 2 语句级能耗估算的一般方法

文献[22]遵循如图 2 所示的方法,属于语句级能耗估算的研究工作.该文献通过源码分析建立解析树(ParseTree).图 3 给出一段简单的 if-else 语句块建立的解析树.每个节点都是一个原子的能耗单元,而树的边则表示组合关系,以此来表征源码的结构特征和语句特征.If-else 分支结构特征带来的能耗由节点 1 表示,而 If 子句中两条语句的顺序结构特征由节点 3 和 7 表示.

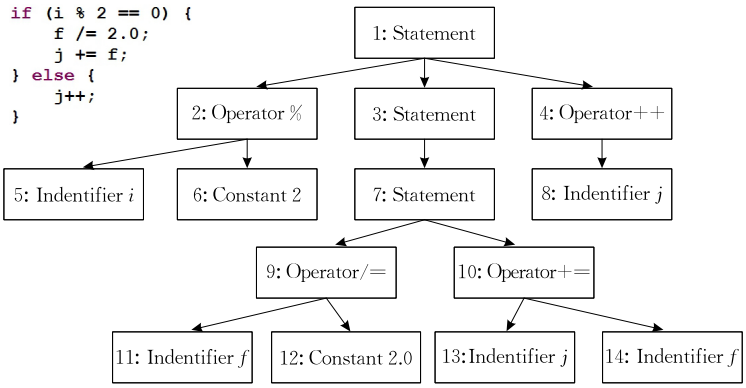


图 3 语句片段和其对应的解析树[23]

文献[22]还设计了解析树的等价变换方法,以优化代码能耗.在能耗估算部分,该文采用指令级的能耗估算技术,设计了一个编译和运行环境相关的处理引擎去逐节点、逐行、逐函数地计算代码的能耗、时间代价和空间代价.

文献[23]提出了一个 C 语言代码能耗模型.该文认为指令执行需要 3 个阶段:指令获取、解码以及执行,因此代码执行能耗不能等价于指令执行能耗,而应该等价获取、解码和执行阶段的能耗之和,如式(1)所示.

$$E_{\text{total\_system}} = E_{\text{total\_fetch}} + E_{\text{total\_decode}} + E_{\text{total\_execute}} \quad (1)$$

其中:$E_{\text{total\_system}}$ 为代码能耗;$E_{\text{total\_fetch}}$ 为指令获取能耗;$E_{\text{total\_decode}}$ 为指令解码能耗;$E_{\text{total\_execute}}$ 为指令执行能耗.获取指令时需要通过外部总线访问内存,因此该阶段的能耗近似等价于访问内存数据的能耗.因此指令获取的能耗可由式(2)估算:

$$E_{\text{total\_fetch}} = E_{\text{mem}} + N_{\text{mem\_cyc}} \quad (2)$$

式中,$N_{\text{mem\_cyc}}$ 表示内存在活动状态时的存储周期

的数量.

实验证明,该文提出的方法累积指令获取以及指令解码阶段的能耗,其估算值比传统仅仅考虑执行阶段的能耗估算值高出 90%.文章认为该估算值更为准确,遗憾的是该文并没有给出估算值与实际值的比较.

此外,部分研究工作采用执行代码的方式来评价其能耗,或称为代码运行时的能耗评价(Profiling)工具[24].通过执行代码,评价工具易获得 CPU 等计算机组件的工作状态,并采用对应的能耗模型,进而估算代码能耗[17].文献[25]设计了 Eprof 工具.Eprof 可以准确评估代码能耗,有助于开发人员定位高能耗代码并加以优化.该文认为传统的以 CPU 为中心的能耗评价方法没有考虑到计算机附属设备,如硬盘和网络设备,而这些设备同样耗能.文章指出,代码同步访问 CPU,因此代码执行与 CPU 能耗也是同步的;而代码对附属设备的访问则可能是异步的,因此语句执行与附属设备能耗也是异步的.该工具主要从 CPU 的同步能耗和设备(如磁盘和网卡)的异步能耗两个角度来考虑,编程人员可以在依赖 CPU 的代码和依赖 I/O 的代码中做出选择.例如,是否采用压缩的数据格式传输数据.整个系统结构如图 4 所示.

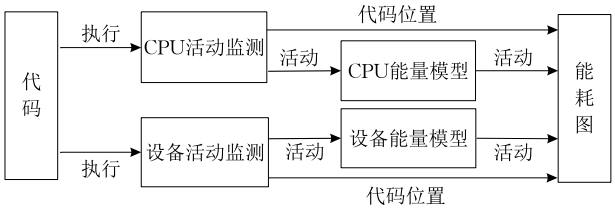


图 4　Eprof 系统结构

如图 4 所示,代码执行时会产生 CPU 行为和设备行为,这些行为能够被 Eprof 采集并构建 CPU 能耗模型和设备能耗模型.实验证明 Eprof 对能够准确定位语句的能耗,误差率为 3.6%.

类似的,通过进程级和设备级监控,文献[26]提出一种估算代码能耗的方法,并定位高能耗代码.该文认为传统解决方案主要采用粗粒度方法来监控设备和进程的能耗,而该文提出了一个细粒度的运行时能耗监控框架,以判断能耗热点的位置.框架包含两级监控组件:操作系统级和进程级能量监控器,前者主要监控硬件设备,后者监控 Java 代码的执行.框架将两个监控器的结果对应起来,就可以评估代码能耗,确定代码中的能耗热点.类似的研究还包括文献[27]和[28].

3.3　模块级

从代码角度,具有适当内聚性且相互弱耦合的代码单元称为模块.在面向对象编程语言中,方法可以视为最小的模块,而类或包作为模块则粒度适中.模块级的软件能耗评估不仅考虑语句能耗、语句结构对能耗的影响,还要考虑模块间的依赖关系对能耗的影响,这种依赖关系通过具体代码的形式体现.因此,模块级的软件能耗估算也属于面向代码的估算方式.

模块级的能耗估算研究主要考虑模块和模块之间的依赖关系对能耗的影响,本文总结这种依赖关系为 4 类,它们及其对能耗的影响分别为:

(1)调用(Invocation).B 模块调用 A 模块的方法,在这种情况下,模块间依赖带来的能耗等同于语句级能耗估算中的函数调用,可以简单地采用内联技术将两者视为一个模块.

(2)通信(Communication).B 模块与 A 模块远程通讯,通信带来的能耗参照消息通信代码的能耗,按同步和异步、阻塞和非阻塞的消息通信,能耗影响均不同.

(3)转换(Conversion).B 模块与 A 模块交互需要接口转换,转换带来的能耗参照数据结构变换代码的能耗.

(4)多态(Polymorphism).B 模块中的方法是 A 模块中方法的多态版本,任何其他模块与 A 模块间的依赖关系都要叠加调用虚方法的能耗.

文献[29]提出一个架构用来评估软件不同模块之间的依赖对于软件能耗的影响.该文献假设模块之间通过“连接器”交互,且连接器本身也是模块,一个连接器也可以与其他连接器交互.由于没有考虑面向对象的特性,连接器被抽象为通信连接器、流程控制连接器、数据转换连接器和接口化简(Facilitation)连接器,其中前两者对应本文总结的“通信依赖”,后两者都对应模块间“转换依赖”.该文献给出了这些连接器的能耗,本文略去了繁杂的公式描述.为了证明能耗估算的准确性和平台无关性,该文估算了 MIDAS 系列的传感器应用软件,在不同的运行环境中估计值和测量值的误差范围在 7%之内.

以 Java 语言为例,文献[30]从软件模块角度提出了 Java 代码的能耗估算方法.包括 CPU 操作、内存访问、I/O 操作对应的代码能耗.此外还整合了模块和模块之间的通信能耗模型,模块的能耗包括运算能耗和通信能耗,而通信能耗又分为本地通信能耗和远程通信能耗.实验使用数字万用表来测量影

响设备能耗的电压和电流两种因子，并分别监控运算能耗和通讯能耗.经实验验证，在不考虑信息交互频率和平均信息大小的情况下，评估结果与实际消耗之间误差在5%之内.类似的面向对象语言编写的代码的能耗估算研究还有文献[31].

部分研究工作采用抽象模块而非具体代码，以评估代码能耗.文献[32]提出基于程序流程图来对软件系统进行模块级的能耗评估.流程图能够表征代码结构：流程图元素封装了程序模块，这些模块分为运算(Process)、数据读写(I/O)和流程控制(Decision)这3类；元素间关系为分支和迭代.该文基于流程图元素的特征和分支、迭代的执行概率估算代码能耗.该文实验部分采用AutoBench① 定义的3种基准用例对能耗估算方法进行了验证.实验结果表明误差范围在−11.9%和6.9%之间.文献[33]进一步的提出基于“存在并发软件模块的流程图”的能耗估算.文献[34]采用了更为抽象的UML和Petri网来表征模块和模块之间的调用关系，以估算代码能耗.上述方法很大程度上借鉴了代码性能估算和评价方法，其关键步骤是代码模块的划分，划分方法需满足：(1)按代码能耗特征划分模块，模块能耗已知，模块内没有显著的分支语句，具体可参照语句级的代码能耗估算方法；(2)模块的调用次数可以估算，流程中的分支判断可以推算执行概率.

由于软件模块的粒度较大，对于不同的编程语言，模块大小划分灵活，给能耗估算带来较大的不确定性，因此上述文献，如文献[30]，多定性的评价而非定量的估算能耗.我们认为这是面向代码的软件能耗估算的发展方向，本文将在下一小节阐述这一观点.

3.4 进一步研究

由前文可知，面向代码的软件能耗估算可以从指令级、语句级和模块级3种不同的粒度考虑.估算方法越靠近硬件平台则越精确，但普适性会越差；反之越抽象，估算误差越大，但普适性越好.大部分研究都强调语句结构和模块依赖关系对能耗的影响.能耗估算的目的并非是要取代仪器测量，而是为了定位高能耗代码、能耗优化或评价能耗优化效果.此外，无论是直接使用指令，还是将代码转换成指令，只要代码没有运行，都不能实现精确估算.因为，代码的逻辑结构复杂，除了运行代码，否则难以精确地确定代码的执行路径，如分支的选择，循环的执行次数，数据结构的规模等等.由此可见，一个精确的代码能耗估算难以实现，且未必满足要求，我们重定义面向代码的软件能耗评估模型和方法应该满足：

(1)环境无关性.与编译环境和运行环境无关；

(2)富代码特性.能够充分考虑代码的各种性质，包括丰富的语句和关键字、代码逻辑结构、数据结构、面向对象特性和模块间依赖关系等；

(3)静态度量.不需要代码编译或者运行，仅仅分析代码即可度量；

(4)适度精确.适当放宽估算的精确性要求，考虑估算值和实际值量级上的一致，变化趋势上的一致，或估算值是实际值的上(确)界或是下(确)界；

(5)可叠加.估算方法要可叠加，计算方法(函数)是分布的，每部分代码能耗估算值的加和应等于整个代码整体估算的能耗值；

(6)公平性.估算精度、实际运行环境等因素都不应该影响估算方法的公平性，在相同的上下文中，若干代码间的估算能耗差距和实际运行能耗差距应基本一致，或满足一定比例；

(7)简洁通用.估算方法应该突出代码的共性，弱化差异，化繁为简，保证方法的通用性.

既然估算方法不可能达到较高的精度，那么我们认为，适当的简化估算方法，突出代码特点，是面向代码的能耗估算方法的进一步研究方向.

众所周知，代码由若干语句以及它们之间的结构组成，这些语句的特征包括操作种类(如运算、赋值和方法调用等)、语句间不同的逻辑结构(如顺序、分支和循环)以及语句包含的面向对象特性(如方法、继承和多态)，我们将上述特征统称为代码特征.现有研究表明，语义相同但上述特征不同的代码运行能耗不同.我们定义运行时能耗(Runtime Energy Consumption)是指代码在特定硬件环境中以及特定输入下执行而消耗的能量，而代码的视在能耗(Apparent Energy Consumption)是由代码特征体现的能耗.当设定某种特征语句的运行时能耗为单位能耗时，代码的视在能耗即是其静态语句能耗的叠加，而忽略运行时和上下文信息.

为阐述上述观点，我们将代码能耗类比为代码性能.在代码性能优化中，我们可以反复执行代码，并调整代码特征以优化代码的“运行时性能”，但代码的运行时性能与输入数据的规模和分布以及硬件环境都有很大的关系，运行时性能的估计和优化需要考虑很多因素；而静态代码优化工具，如pclint②，

① http://www.eembc.org/techlit/datasheets/autobench/db.pdf, 2015

② https://en.wikipedia.org/wiki/PC-Lint, 2015

则可以按代码特征评价和优化代码的“视在性能”，这在代码编写时尤为重要，可以避免很多功能满足但性能较差的代码.同样，视在能耗也是一种静态的度量标准.诚然，代码运行时能耗和视在能耗会存在差异，但视在能耗度量能够有效地比较不同代码之间的能耗差异，或代码的能耗优化效果，且有利于总结面向代码的软件能耗优化方法.

然而，视在能耗的定义和估算也绝非易事，如何定义原子的能耗单元，是指令、运算还是语句；如果是语句，那么如何定义具有单位视在能耗的语句；其他语句的视在能耗如何表达；代码特征和视在能耗之间有何关系；如何确定语句间的调用关系；视在能耗如何比较以及和运行时能耗之间存在何种关系；这些都是亟待研究的问题.

4　能耗优化

本文第 2 节介绍了大量软件能耗优化的思路和具体做法；第 3 节介绍了面向代码的能耗评估，这为发现代码相关的能耗规律、定位耗能代码以及评价能耗优化效果提供了基础.本节着重介绍面向代码的软件能耗优化方法.在 2.3 节我们按优化对象不同，将面向代码的能耗优化又细分为指令级、语句级和模块级优化方法.现存大量指令级能耗优化研究，已较为成熟，本文简要介绍其基本思路，而着重描述语句级和模块级能耗优化.

4.1　指令级

在给定硬件环境下，指令的执行能耗是固定的，若要降低指令的能耗，需开发低功耗的硬件系统，这并非本文关注的内容.从代码角度，指令级能耗优化的基本思路是通过编译技术，减少代码生成的二进制指令数；或通过指令变换将功耗较高的指令用功能相同但功耗较低的指令来替换[35]；或使代码编译后的指令能够充分利用硬件的节能特性[36]；以降低代码能耗.本小节将简要列举部分优化技术.

在编译期间删除永远不会执行的冗余代码，可以减少代码编译后的二进制指令，从而降低 CPU 执行指令集的功耗.例如，可以在编译期间检查条件表达式的值，若值为常量，则可以减少分支循环指令.此外，加法移位操作的能耗比乘法操作的能耗低，因为前者的指令周期较少，因此编译时将代码的乘法操作变为加法操作，会一定程度上节省能耗.在多级存储中，数据的存储位置，如寄存器、高速缓存、主存等，会显著影响代码能耗；但效率高的存储容量有限，我们可以在编译阶段确定数据的重用特征，将反复读写的数据放入高速存储；还可以在编译阶段预测缓存的大小，让多余的缓存设备休眠，以降低能耗[37].

4.2　语句级

在程序设计方法学中，程序变换(Program Transformation)是指由现有程序 P 生成新程序 P' 的过程，且 P 和 P' 等价.一般的，我们通过程序变换将一个面向问题、结构清晰、易于理解但效率不高的正确程序，转变为一个面向过程、不太直观但效率较高的正确程序.语句级能耗优化研究借用程序变换的概念，在不改变代码可察行为的前提下，将能耗较高的代码优化为能耗较低的代码，称为代码变换(Code Transformation).

代码变换的优化原理可以归纳为以下 3 点：(1)化复杂运算为简单运算，化复杂数据结构为简单数据结构(但并非化复杂代码为简单代码)，减少运算能够减少资源的使用，降低能耗；(2)优先使用高性能的存储，如缓存优于内存、磁盘和网络存储，以此类推，无论是否读写数据，内存一直处于通电状态，而磁盘一直在旋转，优先使用高性能存储会减少数据交换，减少空载能耗，提高存储介质的能源效率；(3)充分利用 CPU 资源，在不增加运算量的前提下提高 CPU 运算的效率，减少 CPU 空闲时间，虽然 CPU 空闲时可以进入低功率状态，但这个空载功率也高于内存、硬盘等其他组件的功率，减少 CPU 的空闲，能够减少电能消耗，而 CPU 空闲的原因可以归结为“等待任务”和“等待数据”两种[38]，在代码的执行过程中 CPU 会因为上述原因而被动空闲.

代码变换的步骤主要为“变换”和“校验”两步，可以进一步归纳为如图 5 所示步骤：首先对源代码运用代码变换，使其转换为变换代码；然后对源代码

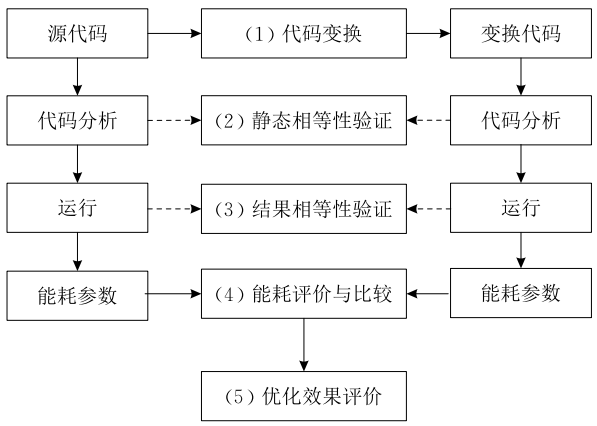


图 5　代码变换的步骤

和变换代码进行分析(或编译),验证两者的相等性,如通过"数据依赖图"表示数据流信息,"代数运算"表示运算信息,判定等价条件,计算输入与输出的映射等价[39];随后运行源代码和变换代码,对运行结果的相等性进行验证;最后通过两者运行期间采集的能耗参数评价能耗和能耗优化效果.图 5 中第 2 步和第 3 步至少应该完成一步,因为在实际运用中,代码变换会由程序自动完成,如开发工具集成的优化功能,那么需要一个正确性检测程序来确保变换后的代码正确和变换前后的语义等价.

代码变换的形式多种多样,或修改代码直接实现代码变换,或不修改代码本身而是对编译器进行优化,间接实现代码变换.大部分研究属于前者,又可以进一步分为"循环结构变换"、"数据结构变换"、"操作和控制结构转换"和"程序内部变换"4 类.对代码变换技术做上述总结后,我们将描述具体的代码变换方法.文献[40-41]对代码变换方法加以总结:"循环结构变换"减少循环代码中变量的依赖,减少 CPU 因 I/O 操作而阻塞的可能,符合优化原理③;"数据结构变换"改变数据结构和存储的关系,减少数据结构的遍历次数,降低存储的访问量,符合优化原理②;"控制结构变换"采用函数内联技术,通过减少函数之间的调用而导致的上下文切换,符合优化原理①;"运算变换"对于条件表达式的每个子项进行了重新排列,能够减少测试条件运算,符合优化原理①;"函数调用预处理"通过宏命令包装库函数的调用,当结果通过先验知识可确定时可以直接去除函数调用,符合优化原理①;"子程序队列重新排序"和"按照地址参数缩小范围"两种变换方法改变堆栈的访问频率和访问顺序,提高缓存命中率,符合优化原理③.该文献选用 6 种测试用例对上述代码变换的能耗优化效果进行评价,如表 1 所示.此外,由于实验程序的启动、监测和退出消耗了较多能耗,因此代码的实际能耗优化效果应该优于测量值.

表 1 代码变换对能耗的优化效果[42]

应用程序	总能耗/%
CRC-16	−5.6
WAVE	−9.6
HASH	−4.0
Bubble Sort	−3.7
Matrix Multiplication	−6.2
IIR Filter	−5.8

文献[42]较早地研究了循环变换对能耗优化影响,此类变换均以减少缓存失效率和提高并行性为目标,符合优化原理②和③.总体上,循环变换方法如下:

(1)循环展开(Loop Unrolling).如代码 1 所示,循环展开通过多次展开"各语句数据无关"的循环体来增加循环语句,减少循环次数.循环展开旨在增加循环体指令级并行度,这种变换减少了循环控制语句执行的次数,为具有多个功能单元的处理器提供指令级并行,也有利于指令流水线的调度.

```
//Original Loop
for (i=0; i<=60; i++){
  a[i]=a[i]*b+c;
}
//Transformed Loop
for (i=0; i<=60; i+=3)
{
  a[i]=a[i]*b+c;
  a[i+1]=a[i]*b+c;
  a[i+2]=a[i]*b+c;
}
```

代码 1 循环展开示例

(2)软件流水线(Software Pipelining).如代码 2 所示(和循环展开联合使用),软件流水线去除循环,或改变循环体内相互独立语句的顺序,提高代码或数据缓存的命中率以降低能耗.

```
//Original Loop
for (i=0; i<=60; i++) {
  f(i);
  g(i);
  h(i);
}
//Transformed Loop
for (i=0; i<=60; i+=3){
  f(i);f(i+1);f(i+2);
  g(i);g(i+1);g(i+2);
  h(i);h(i+1);h(i+2);
}
```

代码 2 软件流水线示例

(3)循环合并(Loop Fusion).如代码 3 所示,循环合并将多组独立的循环合为一个循环.类似于循环展开,变换后代码更有利于减少缓存失效,降低能耗.

```
//Original Loop
for (i=0; i<=60; i++){
  a[i]=i;
}
for (i=0; i<=60; i++){
  b[i]=i;
}
//Transformed Loop
for (i=0; i<=60; i++){
  a[i]=i;
  b[i]=i;
}
```

代码 3 循环分块示例

(4)循环排列(Loop Permutation).如代码 4 所示,循环排列又称为循环互换,用来互换内层循环和外层循环,改变多个循环变量之间的嵌套顺序.它是一种高级别的循环变换,通常改善代码的引用局部性(Locality of Reference).

```
//Original Loop
for (i=0; i<=10; i++){
  for (j=0; j<=20; j++){
    a[i][j]=i+j;
  }
}
//Transformed Loop
for (j=0; j<=20; j++){
  for (i=0; i<=10; i++){
    a[i][j]=i+j;
  }
}
```

代码 4　循环排列示例

(5) 循环分块(Loop Tiling). 如代码 5 所示，循环分块将迭代空间分割成小块，对应的代码和数据也会分割成小块，以此确保循环内的数据能够驻留缓存，提高缓存命中率. 循环分块是针对内存的高级优化，同样以减少缓存失效率，提高 CPU 使用率为目标.

```
//Original Loop
int k=100;
for (int i=0; i<k; i++){
  c[i]=0;
for (int j=0; j<k; j++) {
    c[i]=c[i]+a[i][j] * b[j];
  }
}
//Transformed Loop
//Loop tiling using 2 * 2 blocks
for (int i=0; i<k; i+=2) {
  c[i]=0;
  c[i+1]=0;
for (int j=0; j<k; j+=2){
    for (int m=i; m<min(i+2,k);m++){
    for (int n=j; n<min(j+2,k); n++){
      c[m]=c[m]+a[m][n] * b[n];
    }
  }
 }
}
```

代码 5　循环分块示例

文献[42]采用 SPEC CPU92[43] 标准测试用例的矩阵乘法(Matrix Multiply，MXM)和并行化分块五对角矩阵求逆(Inverts Matrix Penta-diagonals in a Highly Parallel Fashion，VPENTA)对上述部分循环变换的能耗优化效果进行测试，实验结果如图 6 所示. 除软件流水线变换没有优化 VPENTA 用例的能耗，其他变换均能够优化 10%的软件能耗.

从图 6 和其他相关文献的实验结果可以看出，单纯的通过代码变换，能耗优化的效果并非显著，一方面说明测试用例的规模较小，但另一方面也促使人们研究优化效果更好的方法. 上述代码变换方法没有很好地考虑操作系统的影响，也没有很好的考虑系统并发性和多核应用程序中数据在多个处理器之间传递等特性. 因此文献[39]提出了以下方法：(1) 通过线程的合并或分割实现能耗优化的单处理器并发线程管理；(2) 通过处理器内部数据交互和

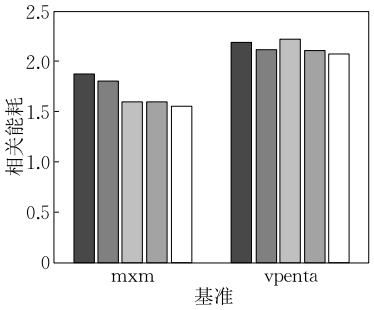


图 6　变换前后能耗相对值，每组数据自左向右分别是原代码、循环展开、软件流水线、循环分块和循环排列[43]

处理器与内存的数据交互之间权衡来优化能耗；(3) 将计算从一个处理器移动到另外一个处理器，以此减少数据访问，降低能耗；(4) 采用共享内存，管道和消息队列等特征来标识软件的通讯特征，通过给出能量优化模型以及源代码转换的整体流程来实现. 文献[39]提出一种兼顾系统特征的代码变换方法，这种变换方法与操作系统相关，且难以实现自动地变换代码，但优化效果明显. 该方法在系统缓存利用率和 IPC(Interprocess Communication)代价之间找到了平衡点. 该文献给出两个实际的应用案例：一是水下导航控制系统代码，改进后节省能耗 29.6%；二是以太网包控制系统代码，改进后节省能耗 27.9%.

综上所述，代码变换的主要方法是源代码的变换，变换同时要充分考虑一般运行环境或特定运行环境的特征. 代码变换是传统面向编译器的优化节能技术提供了有力的补充. 文章总结上述代码变换类型如表 2 所示.

然而，代码变换也有一定代价，不能简单地认为代码变换一定会降低能耗. 能耗和代码规模之间并非简单的正相关关系，因此需要在能耗和代码规模之间寻找到一个平衡点，并以用户约束条件来权衡性能和能耗之间的关系. 文献[44]充分考虑了这一点，提出一个代码级能耗微模型和能耗优化方法. 该方法是一种交互式的代码变换方法，有两方面的优势：(1) 采用“能耗/CPU 时钟周期”作为能耗评价指标，仅当必要时且很可能带来明显优化效果时才进行代码级的变换；(2) 提供强大的工具集来确定优化所需的参数. 文章提出优化代码的执行时间和能耗问题的解空间(代码变换方法)很大，找到最优解是一个 NP 完全问题，但没有给出证明. 代码变换问题的求解由若干步骤组成，每一个步骤用一个 5 元组 r 来描述：

$$r=\{codeSize, executionTime, energy, cacheMiss, slotUsage\}.$$

表 2 代码变换方法

转换类型	转换方法		解释
循环变换(Loop Transformations)	循环展开(Loop Unrolling) 循环合并(Loop Fusion) 循环分块(Loop Tiling) 循环判断外提(Loop Unswitch) While-do 循环变为 do-while 循环	变量扩展(Variable Expansion) 循环交换(Loop Interchange) 软件流水线(Software Pipelining) 零输出条件(Zero Output Condition) (While-do to Do-while)	修改循环体或者循环控制结构,循环重复次数,分离代码块的数量和大小,从源码角度考虑降低能耗的因素
数据结构变换(Data Structure Transformations)	数组声明排序(Array Declaration Sorting) 临时数组插入(Insertion of Temporary Arrays) 标量变量条目(Entries with Scalar Variables)	数组范围修改(Array Scope Modification) 数组替换(Replacement of Array)	研究数据和内存的关系,以最大化利用高速存储数据为目标
程序内部变换(Inter-Procedural Transformations)	子程序队列重排(Subroutines Queuing Reordering)	地址参数范围减少(Scope Reduction of by-address Parameters)	它的优点是通过调用自身来减少上下文切换并且通过消除函数边界来获得更积极的编译优化
操作和控制结构转换(Operators and Control Structure Transformations)	条件表达式重排(Conditional Expression Reordering)	函数调用预处理(Function Call Preprocessing)	优化对象为代码的特定操作或者是特定控制结构

元组中的每个元素为优化对象,设置为{1,0,−1}3 个值,对应含义分别为“增加”、“不变”和“缩小”,极大地缩小了解空间.例如 r 可以为{1,−1,0,−1,1},表明当前代码变换相对于前一步会扩大代码规模,降低执行时间,在能耗不变的情况下减少缓存失效率和提高 CPU 的使用率,增加并行性.而整个优化过程则采用“决策树”求解.

文献[45]采用代码变换的方法对嵌入式系统的能耗进行优化,并且辅以算法优化和内存访问策略的优化,以达到优化系统性能和能耗之双目标.文献[46]用实例验证了代码变换对软件能耗优化的效果.采用 C 语言编写代码求解八皇后问题,并对代码进行能耗优化,评价优化效果.具体采用的优化方法为优化循环结构和改善数据结构.实验结果表明,对比“循环结构”,“数据结构”和“二者结合”3 种优化方法,其能耗分别减少 16.5%,90.2%和 93.1%.由此可见,改善数据结构可以很大程度上降低能耗.事实上,改善数据结构属于一种算法级改进,与简单的代码变换相比更为抽象,更为通用.我们认为,面向算法的能耗优化与面向代码的能耗优化既有联系又有区别,前者更加具有普适性,是未来的研究方向.我们将在本文第 6 节提及算法级能耗优化思路.

4.3 模块级

在面向对象程序设计中,重构(Refactoring)是指在不改变软件系统外部行为的前提下,改善它的内部结构.模块级能耗优化研究借用重构的概念,在不改变模块功能的前提下,调整模块自身或模块间结构以优化代码能耗,称为模块重构(Module Refactoring).本小节将应用于模块的、粗粒度的能耗优化方法归为模块重构,是为了区别于语句级的代码变换方法,前者更着重于模块之间依赖关系的调整,粒度较粗;后者则着重于语句之间关系的调整,粒度较细.我们将按被重构的模块结构由小到大,从算法、组件、(模块)结构 3 个角度介绍现有工作.

(1)算法

相同功能的不同算法需要不同的时间和空间,在给定上下文中,其实现代码量、内存访问量以及 I/O 量也会有明显差异,因而其能耗也不同.面向算法的能耗优化主要采用算法选择的方法,顾名思义,即在给定上下文中从功能相同的算法中选择能耗最低的算法.

由于排序算法复杂多样,很多算法选择研究都以排序算法为例展开[47-50].文献[47]认为不同的排序算法有不同的能耗,而且算法时间复杂度和能耗之间没有直接的联系,因此对于不同的排序算法,如冒泡排序、堆排序、插入排序、合并排序、快排序、选择排序、鸡尾酒排序和希尔排序,该研究建立一个测试平台来度量它们的能耗.实验主要考虑算法能耗和算法规模之间的关系,最后得出结论:一般意义上,算法运行速度越慢耗能越高,但排序算法需要综合考虑能耗和处理速度之间的关系,算法性能不是影响能耗的决定性因素.例如,尽管快排序是最快的排序方式但不是最节能的,插入排序相较于其他排序算法能耗更低,动态的选择排序算法能够显著的优化系统能耗.文献[49]进一步讨论了排序算法的选择标准.分别使用快排序、插入排序、优化算法(动态选择合适的排序算法)3 种不同算法来完成排序,

并测量能耗：快排序算法对运行速度进行优化；插入排序对能耗进行优化；优化算法平衡速度和能耗，优化效果符合预期. 文章提出数据类型对于排序能耗也有影响. 浮点数排序会比整数排序消耗更多的能量，因此编写低能耗代码很重要的一点是选择合适的数据类型. 上述文献提出了代码能耗和代码特征之间的关系，这些关系与 4.2 节描述的代码变换技术相吻合. 此外，相关研究还有评估使用高性能存储器设备(SSD)对算法能耗的影响[49]以及 MapReduce 环境下海量数据排序算法的选择和优化[50]，此处不再累述.

此外，还可以改写高能耗的算法，根据硬件环境和上下文特征，插入实现能耗优化功能的代码. 文献[51]提出的优化框架可自动检测功耗关键性代码区域(Power-CriticalCode Region)，并减少代码执行时的峰值功耗. 借助功耗评估技术，框架首先确定高功耗的代码区域，随后选择相应的节能策略来影响这些区域，如采用 CPU 频率调整(Frequency Scaling)技术可以降低 CPU 功耗，或增加非功能代码来降低功耗峰值. 图 7 给出了该框架的工作机理. 框架的基础组件为“标准软件开发流程(A)”和“运行时功耗分析方法(B)”，基于这两个组件，框架可以跟踪代码，分析并检测出功耗关键性代码区域(C). 随后，框架对这一代码区域进行优化，降低其功耗峰值(D).

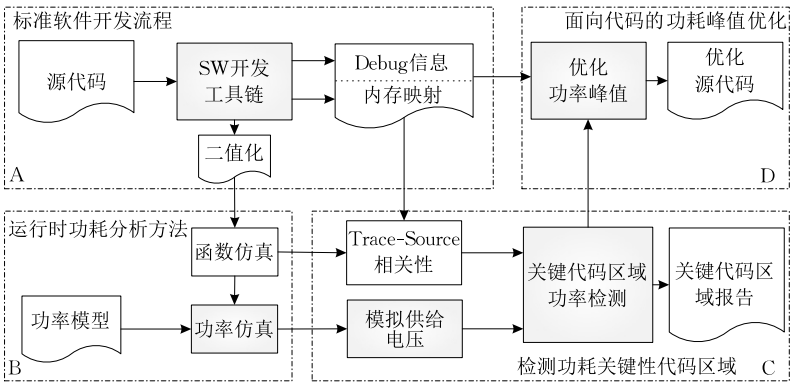


图 7　框架结构和工作机理

本小节重点描述的算法选择技术的基本思路是确定实现同一功能的若干算法的能耗特征以及能耗和上下文之间的关系，随后静态或动态的选择能耗最低的算法. 除此之外，面向算法的能耗优化还包括对算法自身的优化. 例如文献[46]，该文同时采用了语句级和模块级的能耗优化方法，而“低能耗算法选择”和“改写现有算法”是后者采用的主要方法. 但由于算法是一个抽象的概念，难以定义其能耗，因此现有的算法优化实际上是其实现代码的优化. 现有研究尚缺少对算法设计方法和设计准则的优化. 我们认为，从算法设计角度而非实现代码考虑能耗优化是未来的一个研究方向，我们在 4.4 节以及第 6 节中将对此加以论述.

(2)组件

应用软件会依赖很多基础组件，如数据库连接组件、开发框架组件、网络通讯组件等，它们也会对软件能耗产生影响. 首先，操作系统对软件能耗影响显著. 文献[52]以实验的方式证明了操作系统，特别是 Linux 操作系统，对能耗优化起重要作用. 其次，优化软件依赖的外部链接库或框架组件也可以优化其能耗. 文献[53]对 63 个开源应用软件进行了测试，探求软件运行环境是否影响以及如何影响软件能耗. 结果表明，基础组件和外部链接库的使用将额外增加能耗，且不同类型、完成不同业务流程的应用软件能耗级别有显著差异. 由于采用组件会缩短编码时间，上述规律暗示在软件开发效率和软件能耗上存一个平衡.

文献[54]分析了分布式编程抽象(Distributed Programming Abstractions，DPA)，如 Socket、RMI 等对软件能耗的影响，然后提出了一系列可实践的原则以指导程序员编程. DPA 可以理解为实现分布式功能的组件或中间件. 一个分布式软件执行时会经历几个阶段：初始化、执行阶段、空转阶段、终止状态，在每个状态都会产生能耗. 该文认为，分布式系统的整体能耗由应用软件以及 DPA 两个部分构成，在实现相同功能的前提下，从大量 DPA 中选择一个合适的 DPA 会优化软件的整体能耗. 例如同步通信，CPU 能耗取决于传输数据的大小和网络延迟，采用 XML-RPC 和 DOSGi 实现通讯，由于需要传输 XML 文档，因此能耗会略高. 又例如异步通讯，无论是采用 R-OSGi，MOM(Message Oriented Middleware)，还是 Sockets，都不会影响 CPU 能耗.

这是由于 CPU 在空载状态下仍然会消耗能量，而异步通讯能降低 CPU 空闲时间. 此结果也暗示网络传输数据量的大小也会明显影响整体能耗. 在例如复杂数据 Marshalling/Unmarshalling 功能，基于 XML 的 DPA 与基于本地 Java 序列化(例如 RMI，Sockets，and MOM)或其他优化的序列化机制(例如 RBI 和 R-OSGi)的 DPA 相比较，前者更加耗能. 总之，该文通过系统地测量和分析主要 DPA 的能耗特征，以指导软件设计者和分布式系统研究人员；然而文中的能耗测量只包含分布式交互中客户端的能耗，不包括服务器端的能耗.

文献[55]介绍了一种代码生成器，用于优化移动设备端代码，减轻移动终端的计算量，降低能耗. 代码生成器的结构如图 8 所示. 首先，移动设备端代码经过代码分析，定位代码中运算量大的组件，随后将该组件卸载，并生成对应的 Web Service，将 Web Service 自动部署在服务器端，代码通过调用该 Web Service 来实现原有组件的功能. 总之，将复杂的本地组件替换为服务器端组件远程执行，大大降低移动端代码能耗.

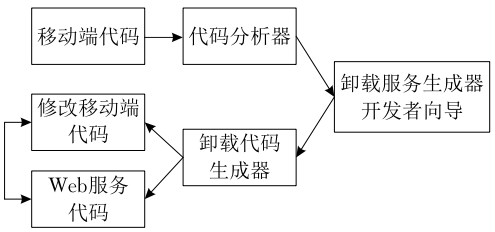


图 8 高能效代码生成器结构

实验采用斐波那契数列作为测试用例. 该用例在移动设备本地、Wifi 连接远程 Web 服务、3G 连接远程 Web 服务 3 种情况下运行. 测试结果表明组件远程服务化的方法能有效减少终端能耗，当远程组件执行时间越长，优势越为明显，该方法不仅缩减了能耗，而且由于服务器性能优于移动设备性能，这些组件的本地时间很可能会更长，此时代码性能同时得到优化. 通过“组件卸载为 Web 服务”的方式可以减少的移动设备能耗为 30%～32%.

文献[56]基于抽象的组件模型，对代码进行性能和能耗两个方面的优化. 由抽象组件模型生成代码需要 3 个步骤：建立组件模型、从模型中生成代码、对代码进行编译. 该文在每个步骤中加入了能耗优化策略，因此能够对生成的代码能耗加以优化. 该文没有给出详细的优化方法，仅以状态图为例举例说明：首先建立状态图，然后由状态图生成 C++源代码，最后对生成的源代码进行优化编译，期间的优化方法包括如图 9 所示的“移除不可达状态”、“移除无效的事务(与已完成状态相关的事务)”.

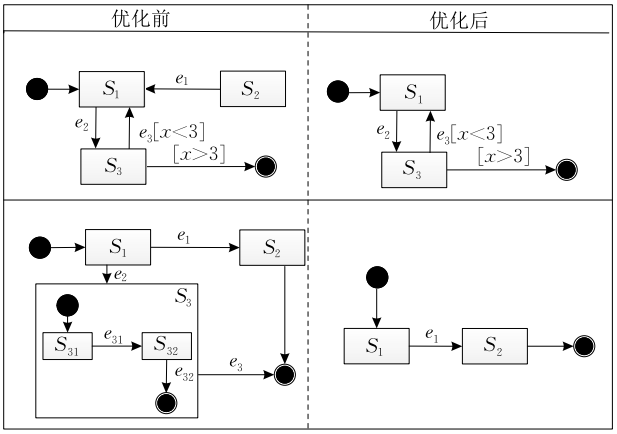


图 9 状态图优化举例

(3) 结构

模块结构可以反映代码的物理结构，表征模块(如源代码文件或动态链接库)之间的编译器和运行时依赖关系，如组件间的连接方法；或模块间的抽象结构，如细粒度的设计模式和粗粒度的软件体系结构. 这些结构都直接影响代码能耗，因此存在优化的空间.

首先，我们研究面向组件间结构的能耗优化. 基于 Fractal 组件模型，文献[57]从性能和能耗两个方面提出了一系列优化方法，简要介绍如下：

(1) 单一实例(Single Instance). 约束组件只有一个实例，以优化内存访问；

(2) 单一接口(Single Interface). 如果一个组件接口是唯一的，应该明显地标识出来，这样，在调用该接口的时候就无需参照上下文信息；

(3) 常量属性(Constant Attribute). 若属性为只读，该属性会被编译为字面量；

(4) 静态绑定(Static Binding). 两个组件之间动态的绑定改为静态的绑定，组件之间的接口会被简化，所有的间接调用会改为直接调用，减少了非功能操作，优化能耗；

(5) 内联绑定(Inline Bindings). 组件级的内联操作，参见文献[41].

其次，我们考虑细粒度的模块结构. 设计模式为构造软件系统的结构、行为和关键属性提供了设计模型和指导，是提高软件质量的常用手段，通常程序设计人员会编写或修改面向对象的代码使其符合某个设计模式. 文献[58]研究了不同设计模式的运用对代码能耗的影响，得出总体规律：软件系统的能耗主要分为处理器能耗和内存能耗两个部分，而内存

能耗又分为指令内存的能耗和数据内存的能耗，前者与代码规模正相关，后者则与处理的数据量正相关；小部分设计模式的运用对代码能耗没有影响，但大部分设计模式则会增加代码能耗.具体来说，工厂方法(Factory Method)模式应用前后代码规模小范围增加，但执行的指令数不会有明显变化，指令内存的能耗和数据内存的能耗也没有明显增加，因此整体能耗也没有明显增加(增加 0.23%)，这个结论也适用于适配器(Adapter)模式(增加 0.18%)，而观察者(Observer)模式应用前后，代码规模明显增加，且增加了很多中间变量的读写，能耗变化明显(增加 44.1%).上述实验结果是文献[58]采用 C++语言在嵌入式设备中测量得到的.但在文献[59]中采用 Java 语言在移动设备中测量出不同的结果，如表 3 所示.运用观察者(Observer)模式能耗变化不大(增加 0.9%)，而装饰器(Decorator)模式虽然并没有引入大量代码，但能耗明显提高(增加 132.40%).这是因为后者实例化大量对象，导致 Java 虚拟机频发触发垃圾回收机制，这将消耗大量能量.由此得出结论，尽管设计模式对于提高面向对象软件的可复用性和可扩展性有着明显的效果，但从能耗角度，有些设计模式需要更为平衡地考虑其能耗代价，且能耗的变化和编程语言以及运行环境均有关.

表 3　设计模式运用前后代码能耗对比[59]

模式	模式运用	执行时间/s	能耗/J	差异/%	
				时间	能耗
Facade	否	15.40	395.60	1.80	2.50
	是	15.70	405.60		
Abstract Factory	否	13.50	342.10	14.20	15.90
	是	15.40	396.60		
Observer	否	15.10	373.70	0.90	0.10
	是	15.20	373.90		
Decorator	否	15.20	374.00	132.40	133.60
	是	35.40	873.80		
Prototype	否	11.20	271.80	33.00	33.20
	是	14.90	362.00		
Template Method	否	15.00	366.40	0.30	0.10
	是	15.10	366.70		

较之于设计模式，软件体系结构更粗粒度地描述软件模块间依赖关系.从软件体系结构角度同样可以优化软件能耗.文献[60]测量了不同系统结构的 MIS(Management Information Systems)能耗，包括两个 ERP 系统、两个 CRM 系统和 4 个 DBMS 系统，证实了：不仅仅是基础设施(硬件)对能耗有影响，软件体系结构(如 N 层结构)和中间件(如 Web 容器和数据库管理系统)对 MIS 的能耗也有巨大的影响，满足相同功能需求的不同 MIS 具有截然不同的能耗.文献[61]提出了一个基于 3 层(视图层、业务层、数据层)体系结构的能源消耗模型，从一个新角度来理解能源消耗，该模型让软件设计人员理解如何实现低能耗的软件.文献[62]从体系结构的角度研究海量数据的高性能、低功耗管理方法，给出一个 3 层数据库系统的原型设计与实现.文献[63]研究数据仓库的体系结构和性能以及能耗之间的联系以及现有能耗优化技术在各个部分上的应用.

4.4　进一步研究

代码变换能够降低软件能耗，是面向代码的软件能耗优化的重要途径，但是很多变换方法都存在缺陷；与此同时，代码变换也是有代价的，不能简单地认为代码变换一定会降低能耗.其次，相对于代码变换，还有一些其他的优化方法，如资源替换和算法选择.除此之外，基于模块结构的能耗优化可以指导编写代码.但我们认为面向代码的软件能耗优化现有研究还存在两处空白，是进一步研究的方向.

第一个方面，现有面向代码的能耗优化技术过于具体，或针对具体的代码，或针对具体的功能，缺乏抽象级别的优化技术，缺乏算法级能耗优化方法.所谓算法级，是指通过类似于“递归算法变为循环算法”、“贪心算法改为动态规划算法求解”、“Cooper 变换”之类的算法变换，来优化一类算法的能耗.

前文介绍了“程序变换”的定义，我们借用此定义，提出以能耗优化为目标的算法变换研究，在不改变算法的可察行为的前提下通过调整算法类型、设计方法、内部结构等，降低算法实现代码的能耗而不去考虑平台和编程语言等实现技术.数学上，变换(映射)是集合 X 到另一个集合 Y 的函数 f，$f:X\to Y$，因此算法变换可以视为原算法 A 上的函数 T_e，有 T_e：$A\to A'$，且 A 和 A' 行为等价 $A\leftrightarrow A'$(功能不变)，但算法实现代码的执行能耗 $E_A\geq E_{A'}$.我们认为可以从以下两个方面研究 T_e.

首先，可以研究算法的分类，将算法按设计思路、结构特征、所需资源等与能耗相关的因素划分为若干类.算法类大小应该适中，不能相互重叠.算法的分类是为了设计变换 T_e，毕竟我们无法针对所有算法设计有效的变换，可行的做法是将各种算法分类，对每一类进行抽象描述.由于变换前后算法是语义等价的，因此还应该设计算法语义等价的验证方法.我们无法定义算法这一抽象概念的能耗，需要将

算法用特定语言在特定平台上实现,度量实现代码的能耗.而实现方法不同必然会影响能耗,因此算法变换必须较粗粒度地优化能耗,一些实现代码造成的微小能耗差异应被忽略.

其次,应该研究变换 T_e.由于算法复杂且难以穷举差异,T_e或许只能手工地且在算法分析之后被定义,但应该研究指导性法则和变换规则集.这一点可以类比传统的程序变换,我们无法定义将递归算法变换为尾递归算法的通用程序变换,但可以定义 Cooper 变换规则来指导此类变换的设计.还可以进一步总结面向算法的能耗优化方法,考虑给定一个定义清晰的算法(如可以用清晰的伪码表示),那么可以通过算法变换优化其能耗的条件是什么,优化效果是否可评估,优化代价是什么;可否总结面向算法的能耗优化方法,得到一种低能耗算法设计方法或规律,指导算法设计.

第二个方面,现有面向代码的能耗优化技术没有充分考虑编程语言特征,尤其是主流的面向对象语言特征,如内存管理、垃圾回收、对象创建等.对于面向对象的编程语言,语句种类繁多,语法和关键字庞杂,完成的功能各异,若研究它们的能耗,则需要首先对面向对象编程语言进行很好的抽象.本文提出一种研究思路,首先将语句分为运算语句、逻辑语句、存储语句和调用语句.调用语句表征了面向对象中"类与类的关系",因为类之间的依赖、继承、聚合等关系在软件执行过程中表征为语句的调用、虚调用、构造函数调用等调用语句.

有了语句的分类和抽象,就可以参照 4.2 节中介绍的代码变换方法,对面向对象编程语言编写的代码进一步分析,找出优化规律和方法.这里需要考虑的是,面向对象编程语言的语句语义丰富,在研究时需要充分考虑编译和运行环境完成的功能.可以对显式方法和隐式方法分别研究.隐式方法是指那些实际发生但没有代码定义或采用特殊格式定义的方法,如对象创建和销毁会隐式调用所有父类的构造函数和析构函数,调用虚方法时会隐式调用虚方法的查找方法.隐式方法更多代表面向对象语言的特性,充分研究隐式方法的能耗评估和优化,将是面向代码的能耗研究的进一步研究内容.

5 优化工具

本文第 3 节至第 4 节综述了面向代码的能耗估算和优化方法,列举了很多现有的研究成果.本节将描述一些软件能耗估算和优化的工具,这些工具可以辅助开发人员编写代码,并能够极为快速准确地定位需要优化的代码,大大减少优化代码所需工作量,也能够避免因为能耗过高而重新设计软件.

软件能耗评估和优化工具可以按多种角度分类.按设计目标可以分为评估工具和优化工具;而按照评估优化层次可以分为硬件层工具和代码层工具;按评估和优化方法可以分为黑盒(无需源码)方法和白盒(需源码)方法;按照评估和优化粒度可以分为指令级工具,语句级工具,模块级工具,应用软件级工具;按是否需要运行时支持可以分为静态(无需执行)方法和动态(需执行)方法.我们对现有的工具做简要介绍,其中评估工具较多,而优化工具则较少,如表 4 所示.

表 4 面向代码的能耗评估和优化工具

名称	目标	层次	方法	粒度	运行	描述
Green Tracker[64]	评估	硬件	黑盒	应用软件	动态	Green Tracker 通过软件运行时测量 CPU 等硬件的能耗来评估软件能耗,并帮助软件研发人员关注能耗问题.Green Tracker 首先执行基准测试来确定哪一个软件系统是最节能的,进而对用户提出建议.
JouleMeter①	评估	硬件	黑盒	模块,应用软件	动态	JouleMeter 用于评估软件使用时 CPU 等硬件的能耗.对 Windows PC 下运行的程序进行能耗预测,概述资源组件的能源使用情况.工具包括 Workload Manager, Event Logger 和 Energy Profiler 等组件.
JouleTrack[65]	评估	硬件	白盒	语句,模块	动态	Joule Track 没有涉及指令级能耗,而是对于基准程序的能耗进行估算.它可以作为一种在线资源使用,从硬件层面对不同软件进行能耗评估.
EECoMark②	评估	硬件	黑盒	应用软件	动态	EECoMark 是一个节能基准测试工具,使用真实的应用程序来评定计算机的综合性能,但优先考虑系统功耗.
SEProf[66]	评估	代码	白盒	语句,模块	动态	SEProf 是一个高层次的代码能耗分析工具,主要用于评价嵌入式软件代码的能耗.以支持多线程的、支持电源管理功能的、多任务的操作系统作为运行环境.
WattsOn[67]	评估	代码	白盒	模块	动态	WattsOn 是一种 App 能耗评估工具,可以在程序运行时确定能源缺乏的部分代码以及耗能最多的组件.这个工具可以在不同环境和操作下测量 App 的能耗.

① http://research.microsoft.com/en-us/projects/joulemeter/default.aspx, 2015
② https://bapco.com/products/eecomark-v2/, 2015

（续　表）

名称	目标	层次	方法	粒度	运行	描述
pTop[68]	评估	代码	白盒	语句	动态	pTop 是一种跨硬件平台、轻量的、易用的进程级能耗估算工具，在充分考虑资源组件差异的情况下评估运行时进程的功耗. pTop 提供 API 帮助软件开发者建立能耗优化策略.
Safari[69]	评估	代码	白盒	模块	动态	Safari 是一个函数级的能耗评估工具，通过采集函数运行时的资源消耗来评估能耗，定位高能耗函数，并且工具的额外开销很小.
HMSim[70]	评估	代码	白盒	语句	静态	HMSim 是高精度的指令级嵌入式软件能耗仿真器，可灵活方便地获得嵌入式软件的能耗值.
SoftExplorer①	优化	代码	白盒	指令，语句	动态	SoftExplorer 是一个在汇编层和源码层执行的函数级别的能耗评估和优化工具，能够快速和准确地分析和评估每部分代码的能耗情况.
Green[71]	优化	代码	白盒	语句	静态	Green 是一个简单灵活的框架，它允许程序员以“近似机会（Approximation Opportunities）”的方式执行复杂的循环和函数，优化代码，降低能耗，同时提供服务质量保证.
Code optimizations[53]	优化	代码	白盒	语句，模块	静态	该工具能够快速准确地定位代码中需要优化的模块，大大减少优化代码所需的工作量，避免重新设计软件. 找到耗能模块后，工具分层次修改代码，从算法、代码、模块、内存 4 个方面实施优化.
PowerScope②	优化	代码	白盒	语句	动态	PowerScope 是一种语句级能耗优化工具. 它将能耗对应到程序结构，通过工具发现网络和磁盘的空闲状态，随后或修改对应的代码，或在合适的时候休眠部件，达到能耗优化效果.
EnergyAnalyzer[72]	优化	代码	白盒	模块	动态	EnergyAnalyzer 是一个在线能耗分析工具，主要针对云计算系统和高性能计算系统. 采用粒子群优化算法优化上述系统的能耗.

在商用领域，Intel、Microsoft 针对绿色计算开发了一些专用工具，供不同领域的开发人员使用，这些工具包括 Power-awareness Intel Power Check, Intel Power Informer, Intel Application Energy Toolkit, Windows Driver Kit, Windows Event Viewer, Windows Performance Monitoring Framework, Power Top, Battery Life Toolkit. 这些软件均能够收集操作系统的各种参数并加以调整，但它们并不能针对代码进行能耗优化. 就我们所知，未见成熟的、商业的面向代码的能耗优化工具.

与编程工具高度集成的、完全面向代码的能耗优化工具目前还较少. Trepn 是 App 软件能耗估算工具，它可以运行在 Eclipse 中，辅助开发人员进行低功耗 App 软件开发③. 与此同时，我们团队目前正在开发一个基于 Eclipse 插件的开发工具 AecProfiler，可以在开发人员编写 Java 代码的过程中提示代码能耗和能耗优化方法.

6　能耗复杂度

前文分别描述了面向代码的软件能耗估算和能耗优化研究的进一步工作（3.4 节和 4.4 节），作为研究展望，本节我们提出一个与之相关的新问题，即算法的能耗复杂度研究. 算法是解题方法的一种抽象描述，代码是算法的实现. 面向代码的能耗优化与面向算法的能耗优化相比，前者更为具体而后者更为抽象. 若能够从算法角度研究能耗优化问题，则抽象层次更高，得出更为普适的结论，更具有指导意义. “算法实现”的能耗，也即代码能耗是可以清晰度量的，但难以定义算法这一抽象概念的能耗，类比我们也无法定义算法的执行时间. 面向算法研究能耗优化，需要一个新的能耗评价模型.

我们不能简单地从功能和性能角度来衡量一个算法，而应该考虑其“能耗特征”. 那么可否使用“时间特征”来代替“能耗特征”呢？前期研究[70]中我们发现：当优化软件模块性能时，并不能确定性地优化其能耗，在某些情况，能耗和性能成反比：如增加 Web 服务器数量可以缩短请求的响应时间，但增加了系统能耗；或采用存储过程执行部分算法可以减少应用服务器的压力，在请求密集时可以提高性能，但能耗几乎不变；或采用密集的数据库索引能同时优化性能和能耗. 因此不能简单地使用性能度量代替能耗度量.

计算复杂性理论（Computational Complexity Theory）研究计算问题求解时所需资源（比如时间和空间）的界以及如何尽可能地节省这些资源. 对资源的界的深入理解可以更加明确求解算法和资源之间的规律，一方面可以很好地对算法进行优化，另一方面还可以确定一个能或不能被计算机求解的问题所具有的实际限制. 我们通常研究算法的时间复杂度和空间复杂度，尚缺少对其能耗复杂度的研究，即研究算法规模和其消耗的能量之间的关系. 基于上

① http://www.softexplorer.fr/, 2015
② https://code.google.com/p/powerscope/, 2015
③ https://developer.qualcomm.com/software/trepn-plug-in-eclipse, 2015

述背景，我们认为如何定义普适的算法能耗复杂度，并以此提出能耗优化方法是亟待解决的难题，具体可以从以下几个方面着手研究.

首先，研究算法能耗复杂度模型. 针对一个具体算法，利用数学工具研究其复杂程度，称之为复杂度分析. 目前对算法的复杂度分析主要集中的时间和空间两个角度，即时间复杂度 $T(n)$ 和空间复杂度 $S(n)$，两者是问题规模 n 的函数，可采用渐进记法 $T(n)=O(f_t(n))$ 和 $S(n)=O(f_s(n))$. 分析一个算法的能耗复杂度，亦应该遵照这种方式. 可以采用的基本思路是找到能耗复杂度 E 的函数表达，然后利用数学知识化简. 那么：(1) 能耗仅仅与问题规模有关，还是与执行环境、并行程度等其他因素相关，如何表达这种关系；(2) 如何最小化 E 函数的变量个数，把一些上下文相关的变量定义为常量，可否采用 $E(n)=O(f_e(n))$ 的形式表达能耗复杂度；(3) 能耗与时间复杂度 $T(n)$ 和空间复杂度 $S(n)$ 存在何种关系，相同时间复杂度的算法其能耗复杂度是否相同；(4) 如何借助 $T(n)$ 和 $S(n)$ 的表达来推导 $E(n)$. 上述问题都是算法能耗复杂度研究中亟待解决的问题.

其次，需要着手解决一个根本性的问题：能耗等于功率乘以时间，若功率在小范围内浮动，则遵照算法时间复杂度定义的能耗复杂度可能会和前者一致，而相同时间复杂度算法的不同能耗特征无法表现. 事实上，按现有实践经验，应该不存在需要大量时间但很少能耗的算法，也不存在有限时间可以完成但有限能耗无法完成的算法，反之亦然. 因此，要研究定义算法能耗复杂度的新思路，放大时间复杂度相同的算法的能耗差异.

再次，我们研究能耗复杂度的分类. (1) 对于算法的时间复杂度，采用同阶多项式渐进记法，可以分为常数级 $O(1)$，亚线性级 $O(n^k)(0<k<1)$，对数级 $O(\log n)$，多项式对数级 $O((\log n)^k)(k>1)$，线性级 $O(n)$，拟线性级 $O(n\log n)$，多项式级 $O(n^k)(k>1)$，指数级 $O(k^n)$ 和阶乘级 $O(n!)$ 等，那么，能耗复杂度是否也可以分为这些级别，或者哪些级别不会存在；(2) 传统的时间复杂度若是多项式级，则认为该算法为 P 问题，是多项式时间内可解的确定性问题，这一规律在算法的能耗复杂度中是否适用以及适用范围是什么；(3) 时间复杂性理论把问题分成 P 类和 NP 类，那么能耗复杂性是否可以比照此分类，这种分类是否有意义，是否存在无法在有限能量下求解的问题，如果有，这些算法呈何种特征，如果时间复杂度理论很好地诠释了问题的可解性，那么，能耗复杂度是否可以在此基础上进一步以合适的粒度来评价求解所需的能耗大小，以指导能耗优化.

最后，研究能耗复杂度优化. (1) 若某算法的能耗复杂度已知，那么采用何种数学方法能够优化该复杂度，且这种优化方法可否实际运用，应用效果如何. 现有时间复杂度优化方法在能耗复杂度中是否可行；(2) 是否存在能耗复杂度的规约方法，将不同的算法规约到同一能耗复杂度可定义、可优化的算法上；(3) 研究特定问题求解算法的能耗的界，及任何经过精妙设计的算法，或任何优化能耗的程序变换均无法达到的最优能耗复杂度. 对于一个能耗复杂度为 $E(n)=O(f_e(n))$ 的算法 X，能耗复杂性研究企图找到一个尽可能大的函数 $g_e(n)$，并以此证明对于该算法，无论实例大小如何，任意与 X 语义等价的算法 X' 的消耗的能量不会小于 $\Omega(g_e(n))$.

7 结束语

随着信息产业的不断发展，软件已经深入到人们的日常工作和生活中. 面向代码的软件能耗优化从程序设计和编码角度研究能耗优化问题，是硬件层和资源层能耗优化的一个很好的补充. 该技术能够很好地指导软件设计和实现. 众所周知，能源是计算机系统的重要运行成本，马化腾表示腾讯的能源成本已经等于或超过人力成本①. 软件能耗优化技术的研究和其在软件设计和开发领域的应用，尤其是在手机等能源受限硬件环境之上的软件中得以应用，可以提高软件的竞争力，带来市场经济效应. 此外，目前国内电能主要是以火力发电为主，软件能耗的降低不仅节省了开销，也减少了二氧化碳的排放量，保护了环境，符合目前倡导的节约型经济和低碳经济.

本文对近几年国内外在面向代码软件能耗优化研究方面的主要研究成果进行了综述：总结了软件能耗优化的思路，优化技术层次，软件能耗优化的优势，和具体的面向代码的软件能耗优化方法；随后从面向代码的软件能耗估算方法和优化算法两个方面对现有工作加以梳理，并分别提出进一步的研究方向；最后，由于现有面向代码的软件能耗研究过于具体，需要更为抽象的优化方法，本文提出算法能耗复

① http://group.vsharing.com/News/Diary.aspx?id=25268, 2015

杂度这一新观点，指出仍然存在的问题和可能的解决办法，以供研究人员参考. 总的来说，面向代码的软件能耗优化研究仍然处于刚刚起步的阶段，仍然有大量具有挑战性的关键问题需要深入研究，为国内的绿色计算研究者提供了广阔的研究空间.

参 考 文 献

[1] Poess M, Nambiar R O. Energy cost, the key challenge of today's data centers: A power consumption analysis of TPC-C results. Proceedings of the VLDB Endowment, 2008, 1(2): 1229-1240

[2] Duy T V T, Sato Y, Inoguchi Y. Performance evaluation of a green scheduling algorithm for energy savings in cloud computing//Proceedings of the 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum. Atlanta, USA, 2010: 1-8

[3] Deng W, Liu F, Jin H, et al. Harnessing renewable energy in cloud datacenters: Opportunities and challenges. IEEE Network, 2014, 28(1): 48-55

[4] Amsel N, Ibrahim Z, Malik A, Tomlinson B. Toward sustainable software engineering (NIER track)//Proceedings of the 33rd International Conference on Software Engineering (ICSE). Honolulu, USA, 2011: 976-979

[5] Sabharwal M R. Software power optimization: Analysis and optimization for energy-efficient software//Proceedings of the 17th IEEE/ACM International Symposium on Low-Power Electronics and Design. La Jolla, USA, 2011: 63-64

[6] Lin Chuang, Tian Yuan, Yao Min. Green network and green evaluation: Mechanism, modeling and evaluation. Chinese Journal of Computers, 2011, 34(4): 593-612(in Chinese)
(林闯，田源，姚敏. 绿色网络和绿色评价：节能机制，模型和评价. 计算机学报，2011，34(4)：593-612)

[7] Zhang Wei, Song Ying, Ruan Li, et al. Resource management in Internet-oriented data centers. Journal of Software, 2012, 23(2): 179-199(in Chinese)
(张伟，宋莹，阮利等. 面向 Internet 数据中心的资源管理. 软件学报，2012，23(2)：179-199)

[8] Ye Ke-Jiang, Wu Zhao-Hui, Jiang Xiao, He Qin-Ming. Power management of virtualizd cloud computing platform. Chinese Journal of Computers, 2012, 35(6): 1262-1285(in Chinese)
(叶可江，吴朝晖，姜晓，何钦铭. 虚拟化云计算平台的能耗管理. 计算机学报，2012，35(6)：1262-1285)

[9] Trinh T A, Hlavacs H, Talia D. Energy efficiency in large-scale distributed systems. Future Generation Computer Systems, 2012, 28(5): 743-744

[10] Sun Y, Zhao Y, Song Y, et al. Green challenges to system software in data centers. Frontiers of Computer Science in China, 2011, 5(3): 353-368

[11] Watson R T, Boudreau M-C, Chen A J. Information systems and environmentally sustainable development: Energy informatics and new directions for the IS community. Management Information Systems Quarterly, 2010, 34(1): 23-38

[12] Velte T, Velte A, Elsenpeter R C. Green IT: Reduce Your Information System's Environmental Impact While Adding to the Bottom Line. New York, USA: McGraw-Hill Education, 2008

[13] Song Jie, Liu Xue-Bing, Zhu Zhi-Liang, et al. An energy-efficiency optimized resource ratio model for MapReduce. Chinese Journal of Computers, 2015, 38(1): 59-73 (in Chinese)
(宋杰，刘雪冰，朱志良等. 一种能效优化的 MapReduce 资源比模型. 计算机学报，2015，38(1)：59-73)

[14] Song Jie, Liu Xue-Bing, Zhu Zhi-Liang, et al. A novel task scheduling approach for reducing energy consumption of MapReduce cluster. IETE Technical Review, 2014, 31(1): 65-74

[15] Shenoy S S, Eeratta R. Green software development model: An approach towards sustainable software development// Proceedings of the 2011 Annual IEEE. Hyderabad, India, 2011: 1-6

[16] Zhao Xia, Guo Yao, Chen Xiang-Qun. Research progresses on energy-efficient software optimization techniques. Journal of Computer Research and Development, 2011, 48(12): 2308-2316(in Chinese)
(赵霞，郭耀，陈向群. 软件能耗优化技术研究进展. 计算机研究与发展，2011，48(12)：2308-2316)

[17] Song Jie, Li Tian-Tian, Yan Zhen-Xing, et al. Energy-efficiency model and measuring approach for cloud computing. Journal of Software, 2012, 23(2): 200-214(in Chinese)
(宋杰，李甜甜，闫振兴等. 一种云计算环境下的能效模型和度量方法. 软件学报，2012，23(2)：200-214)

[18] Bazzaz M, Salehi M, Ejlali A. An accurate instruction-level energy estimation model and tool for embedded systems. IEEE Transactions on Instrumentation and Measurement, 2013, 62(7): 1927-1934

[19] Luo Gang, Guo Bing, Shen Yan, et al. Analysis and optimization method of energy consumption characteristics in embedded software based on source-code and algorithm level. Chinese Journal of Computers, 2009, 32(9): 1869-1875(in Chinese)
(罗刚，郭兵，沈艳等. 源程序级和算法级嵌入式软件功耗特性的分析与优化方法研究. 计算机学报，2009，32(9)：1869-1875)

[20] Da Costa G, Hlavacs H. Methodology of measurement for energy consumption of applications//Proceedings of the 2010 11th IEEE/ACM International Conference on Grid Computing (GRID). Brussels, Belgium, 2010: 290-297

[21] Singh V K, Dutta K, VanderMeer D. Estimating the energy consumption of executing software processes//Proceedings of

the Green Computing and Communications (GreenCom). Beijing, China, 2013: 94-101

[22] Brandolese C. Source-level estimation of energy consumption and execution time of embedded software//Proceedings of the 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools. Parma, Italy, 2008: 115-123

[23] Zhou X, Guo B, Shen Y, et al. Design and implementation of an improved C source-code level program energy model//Proceedings of the Embedded Software and Systems. Hangzhou, China, 2009: 490-495

[24] Noureddine A, Bourdon A, Rouvoy R, et al. A preliminary study of the impact of software engineering on greenit//Proceedings of the 2012 1st International Workshop on Green and Sustainable Software. Zurich, Switzerland, 2012: 21-27

[25] Schubert S, Kostic D, Zwaenepoel W, et al. Profiling software for energy consumption//Proceedings of the Green Computing and Communications. Besançon, France, 2012: 515-522

[26] Noureddine A, Rouvoy R, Seinturier L. Monitoring energy hotspots in software. Automated Software Engineering, 2015, 22(3): 291-332

[27] Noureddine A, Rouvoy R, Seinturier L. Unit testing of energy consumption of software libraries//Proceedings of the 29th Annual ACM Symposium on Applied Computing. Gyeongju, Republic of Korea, 2014: 1200-1205

[28] Noureddine A, Bourdon A, Rouvoy R, et al. Runtime monitoring of software energy hotspots//Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. Essen, Germany, 2012: 160-169

[29] Seo C, Edwards G, Malek S, Medvidovic N. A framework for estimating the impact of a distributed software system's architectural style on its energy consumption//Proceedings of the 7th Working IEEE/IFIP Conference on Software Architecture. Vancouver, Canada, 2008: 277-280

[30] Seo C, Malek S, Medvidovic N. Component-Level Energy Consumption Estimation for Distributed Java-Based Software Systems. Switzerland: Springer, 2008

[31] Seo C, Malek S, Medvidovic N. An energy consumption framework for distributed Java-based software systems//Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering. Atlanta, USA, 2007: 421-424

[32] Heinrich P, Bergler H, Eilers D. Energy consumption estimation of software components based on program flowcharts //Proceedings of the High Performance Computing and Communications. Paris, Franc, 2014: 542-545

[33] Heinrich P, Bergler H, Oswald E. Early energy estimation of networked embedded systems executing concurrent software components. International Journal of Modeling and Optimization, 2015, 5(2): 119

[34] Shorin D, Zimmermann A, Maciel P. Transforming UML state machines into stochastic Petri nets for energy consumption estimation of embedded systems//Proceedings of the Sustainable Internet and ICT for Sustainability (SustainIT). Pisa, Italy, 2012: 1-6

[35] Schulte E, Dorn J, Harding S, et al. Post-compiler software optimization for reducing energy//Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems. Salt Lake City, USA, 2014, 42(1): 639-652

[36] Ozturk O, Kandemir M T, Chen Guang-Yu. Compiler-directed energy reduction using dynamic voltage scaling and voltage islands for embedded systems. IEEE Transactions on Computers, 2013, 62(2): 268-278

[37] Tavarageri S, Sadayappan P. A compiler analysis to determine useful cache size for energy efficiency//Proceedings of the Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW). Cambridge, USA, 2013: 923-930

[38] Song J, Li T, Wang Z, Zhu Z. Study on energy-consumption regularities of cloud computing systems by a novel evaluation model. Computing, 2013, 95(4): 269-287

[39] Fei Y, Ravi S, Raghunathan A, Jha N K. Energy-optimizing source code transformations for operating system-driven embedded software. ACM Transactions on Embedded Computing Systems, 2007, 7(1): 2

[40] Brandolese C, Fornaciari W, Salice F, Sciuto D. The impact of source code transformations on software power and energy consumption. Journal of Circuits, Systems, and Computers, 2002, 11(5): 477-502

[41] Kim B, Cho Y, Hong J. An efficient function inlining scheme for resource-constrained embedded systems. Journal of Information Science and Engineering, 2012, 28(5): 859-874

[42] Yang H, Gao G R, Marquez A, et al. Power and energy impact by loop transformations//Proceedings of the Workshop on Compilers & Operating Systems for Low Power Parallel Architecture & Compilation Techniques. New Orleans, USA, 2001: 1-12

[43] Phansalkar A, Joshi A, Eeckhout L, John L K. Measuring program similarity: Experiments with SPEC CPU benchmark suites//Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software. Texas, USA, 2005: 10-20

[44] Zafar N, Rupp M. Energy-aware source-to-source transformations for a VLIW DSP processor//Proceedings of the 17th International Conference on Microelectronics. Islamabad, Pakistan, 2005: 133-138

[45] Park I, Lee H, Lee S, et al. Source code optimization for embedded processing software of tactical communication system//Proceedings of the 2012 7th International Conference on Computing and Convergence Technology. Seoul, South Korea, 2012: 803-806

[46] Luo G, Guo B, Shen Y, et al. Analysis and optimization of embedded software energy consumption on the source code and algorithm level//Proceedings of the 4th International Conference on Embedded and Multimedia Computing. Jeju, Republic of Korea, 2009: 1-5

[47] Bunse C, Hopfner H, Mansour E, Roychoudhury S. Exploring the energy consumption of data sorting algorithms in embedded and mobile environments//Proceedings of the 10th International Conference on Mobile Data Management: Systems, Services and Middleware. Taipei, China, 2009: 600-607

[48] Bunse C, Höpfner H, Roychoudhury S, Mansour E. Energy Efficient Data Sorting Using Standard Sorting Algorithms. Switzerland: Springer, 2011

[49] Beckmann A, Meyer U, Sanders P, Singler J. Energy-efficient sorting using solid state disks. Sustainable Computing: Informatics and Systems, 2011, 1(2): 151-163

[50] Song J. Performance and energy optimization on Terasort algorithm by task self-resizing. Information Technology and Control, 2015, 44(1): 30-40

[51] Bachmann C, Genser A, Steger C, et al. An Automated Framework for Power-Critical Code Region Detection and Power Peak Optimization of Embedded Software. Berlin Heidelberg: Springer, 2011

[52] Orgerie A C, Lefevre L, Gelas J-P. Demystifying energy consumption in grids and clouds//Proceedings of the Green Computing Conference. Chicago, USA, 2010: 335-342

[53] Capra E, Francalanci C, Slaughter S A. Is software "green"? Application development environments and energy efficiency in open source applications. Information and Software Technology, 2012, 54(1): 60-71

[54] Kwon Y W, Tilevich E. The impact of distributed programming abstractions on application energy consumption. Information and Software Technology, 2013, 55(9): 1602-1613

[55] Fekete K, Pelle A, Csorba K. Energy efficient code optimization in mobile environment//Proceedings of the Telecommunications Energy Conference (INTELEC). Vancouver, Canada, 2014: 1-6

[56] Charfi A, Mraidha C, Gérard S, et al. Toward optimized code generation through model-based optimization//Proceedings of the Conference on Design, Automation and Test in Europe, European Design and Automation Association. Dresden, Germany, 2010: 1313-1316

[57] Lobry O, Navas J, Babau J P. Optimizing component-based embedded software//Proceedings of the Computer Software and Applications Conference. Seattle, USA, 2009: 491-496

[58] Litke A, Zotos K, Chatzigeorgiou A, Stephanides G. Energy consumption analysis of design patterns. World Academy of Science, Engineering and Technology, International Science Index, 2011, 1(11): 547-551

[59] Bunse C, Schwedenschanze Z, Stiemer S. On the energy consumption of design patterns//Proceedings of the 2nd Workshop EASED@BUIS Energy Aware Software-Engineering and Development. Oldenburg, Germany, 2013: 7-8

[60] Capra E, Formenti G, Francalanci C, Gallazzi S. The impact of MIS software on IT energy consumption//Proceedings of the ECIS 2010 PROCEEDINGS. Pretoria, South Africa, 2010: 95

[61] Jwo J S, Wang J Y, Huang C H, et al. An energy consumption model for enterprise applications//Proceedings of the 2011 IEEE/ACM International Conference on Green Computing and Communications (GreenCom). Sichuan, China, 2011: 216-219

[62] Ou Y, Härder T. Trading memory for performance and energy//Xu Jianliang, Yu Ge, Zhou Shuigeng, et al. eds. Database Systems for Advanced Applications. Switzerland: Springer, 2011

[63] Poess M, Nambiar R O. Tuning servers, storage and database for energy efficient data warehouses//Proceedings of the 2010 IEEE 26th International Conference on Data Engineering (ICDE). Long Beach, USA, 2010: 1006-1017

[64] Amsel N, Tomlinson B. Green tracker: A tool for estimating the energy consumption of software//Proceedings of the 28th International Conference on Human Factors in Computing Systems. Atlanta, USA, 2010: 3337-3342

[65] Sinha A, Chandrakasan A P. JouleTrack: A web based tool for software energy profiling//Proceedings of the 38th Design Automation Conference. Las Vegas, USA, 2010: 220-225

[66] Tsao S L, Chen J J. SEProf: A high-level software energy profiling tool for an embedded processor enabling power management functions. Journal of Systems and Software, 2012, 85(8): 1757-1769

[67] Mittal R, Kansal A, Chandra R. Empowering developers to estimate app energy consumption//Proceedings of the 18th Annual International Conference on Mobile Computing and Networking. Istanbul, Turkey, 2012: 317-328

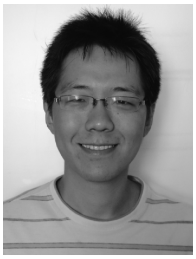
[68] Do T, Rawshdeh S, Shi W. pTop: A process-level power profiling tool//Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower'09). Malo, France, 2009: 1-5

[69] Wang Shi-Nan, Li Youhuizi, Shi Wei-Song, et al. Safari: Function-level power analysis using automatic instrumentation //Proceedings of the 3rd International Conference on Energy-Aware Computing. Cyprus, 2012: 1-6

[70] Zhou Xue-Mei, Guo Bing, Shen Yan, et al. Design and implementation of I/O power consumption simulation modules of power consumption simulator HMSim. Journal of Computer Applications, 2010, 30(7): 1987-1990(in Chinese)
(周雪梅，郭兵，沈艳等. 功耗仿真器 HMSim 的 I/O 接口功耗仿真模块设计与实现. 计算机应用，2010，30(7)：1987-1990)

[71] Baek W, Chilimbi T M. Green: A framework for supporting energy-conscious programming using controlled approximation. ACM SIGPLAN Notices, 2010, 45(6): 198-209

[72] Benedict S, Rejitha R S, Bright C B. Energy consumption-based performance tuning of software and applications using Particle Swarm Optimization//Proceedings of 2012 CSI 6th International Conference on the Software Engineering. Indore, India, 2012: 1-6



SONG Jie, born in 1980, Ph. D., associate professor. His research interests include energy-efficient computing, big data management, and cloud computing.

SUN Zong-Zhe, born in 1991, M. S. candidate. His current research interest is energy-efficient computing.

LI Tian-Tian, born in 1989, Ph. D. candidate. Her current research interest is energy-efficient computing.

BAO Yu-Bin, born in 1968, Ph. D., professor. His research interest is big data management.

YU Ge, born in 1962, Ph. D., professor, Ph. D. supervisor. His research interest is database theory.

Background

The energy consumption of the growing large-scale computing systems has already become an urgent problem. So in recent years, the energy consumption has increasingly become a new measurement of the software. With regard to the study of computer system performance, in the early phase researchers focused on hardware performance and later extended to software, algorithms and procedures. The studies of energy consumption are experiencing the same process above. In recent years, the idea of reducing the energy consumption of software through code level draws wide attention, and the energy evaluation of procedures, a kind of fine grain programming technology, is becoming an intensive research topic. In this paper, researches of software energy consumption and code-oriented optimizations are summarized. Most of literatures focus on only one algorithm or a kind of algorithms in the special situation, and meanwhile in general these literatures are related with the high-level languages or some specific information of the hardware. Based on these, new ideas of code oriented optimization are proposed.

This work is mainly supported by the National Natural Science Foundation of China (No. 61433008), named as "Research on High Efficiency of Big Data Storage and Management System". The project aims to develop a set of big data storage and management technologies, including the way to construct I/O-specific and energy efficient application on big data storage systems. This work is also supported by the National Natural Science Foundation of China (Nos. 61402090, 61502090), named as "Research on Energy-Consumption Optimization Approaches for Cloud Database Systems". Our group has been working in the area of energy-efficient computing for 6 years and has published many papers.