算法能耗复杂度的定义与推导

宋 杰[1] 马忠义[1] 徐 澍[1] 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 计算机系统的性能优化研究早期关注硬件性能,后来更关注软件性能.能耗优化研究与之类似,近年来,面向软件或代码的系统能耗优化方法研究受到重视,而算法作为代码的抽象,其能耗评价技术更是一个研究重点.现有算法能耗研究大多针对特定算法以及特定运行环境,且和编程语言或硬件特性相关,并不具有普适性.比照算法的时间复杂度和空间复杂度,提出能耗复杂度是认知算法能耗特性的有效模型.首先,以图灵机为起点,建立更适于算法能耗分析的能耗图灵机,并定义算法能耗复杂度,为评价和优化算法能耗提供理论依据;然后,分析算法能耗与算法空间复杂度、时间复杂度、存储和运算语句的交叉度之间的关系,并设计利用后两者推导能耗复杂度的方法;最后,实验验证算法能耗复杂度的正确性.能耗复杂度的定义将为设计更低能耗的算法、算法选择以及算法能耗优化提供理论依据.

关键词 绿色计算;能耗度量;算法能耗;能耗复杂度;交叉度

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2018.00709

Define and Deduce Energy Consumption Complexity of Algorithms

SONG Jie[1] MA Zhong-Yi[1] XU Shu[1] 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 As far as the performance optimization of computer system is concerned, in the early years, hardware performance is well studied and then software efficiency is gradually focused. Also, the study of energy consumption optimization follows the same way, and software oriented energy consumption optimization is highlighted recently, for the energy consumption grows rapidly and the energy issues in IT industry is much harder to deal with. Evaluation and optimization on energy consumption of algorithms, which are abstractions of procedures, is becoming a new and hot research topic. However, most existing researches focus on specific algorithms or runtime environment, and they are closely related with the programming languages or hardware features. And there are some researches on energy complexity, but their models are just for specific environment such as only for parallel algorithms of multi-core environment. Therefore, they are not universal solutions. Referring to the time complexity, we propose that the energy consumption complexity, which represents the energy consumption features of an algorithm, as well as a key measurement for evaluating algorithms. Firstly, starting from the classical Turing machine, we design an improved Turing machine for analyzing energy consumption of an algorithm, and define

收稿日期:2016-05-15;在线出版日期:2017-01-20.本课题得到国家自然科学基金(61433008,61672143,61662057,61502090,61402090)资助.**宋 杰**,男,1980 年生,博士,副教授,中国计算机学会(CCF)高级会员,主要研究方向为高能效计算、大数据存储与管理、迭代计算.E-mail:songjie@mail.neu.edu.cn.**马忠义**,男,1994 年生,硕士研究生,主要研究方向为高能效计算.**徐 澍**,男,1991 年生,硕士研究生,主要研究方向为高能效计算.**鲍玉斌**,男,1968 年生,博士,教授,中国计算机学会(CCF)高级会员,主要研究领域为大数据存储与管理.**于 戈**,男,1962 年生,博士,教授,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为数据库理论.

its energy consumption complexity; we define the abstract energy consumption in this part, make the energy consumption of program into the dimensionless energy consumption that can deal with the energy consumption issue of an algorithm; and then we define the set and polynomial relationship of abstract energy consumption to make the abstract energy consumption comparable; and then combining with our definition of status transfer energy consumption and inertia energy consumption, we finally propose the energy consumption complexity of algorithm on the improved Turing machine. Secondly, we analyze the relationship among energy consumption complexity, space complexity, time complexity, and interleaving of calculation statements and storage statements, and then deduce the first by the latter two; the space and time complexity only have constant or linear effect on energy consumption complexity, which is meaningless to a definition of our new concept; but with the definition of cross factor that we propose in the chapter of Turing machine, we can combine these two complexity together, and deduce the energy consumption complexity with the cross factor; the cross factor is also an function of the input size, and the fundamental of the deducing process is based on DVFS and other equations on energy consumption, so the energy consumption complexity has great difference with other complexities. Finally, we validate the energy consumption complexity by experiments; the experiment was done on a cluster with java and the test case we used was seven distributed algorithms that their theoretical time complexity, energy consumption complexity and cross factor are given; with the experiment and the theoretical complexity of those algorithms, we validate the effect of cross factor and time on the energy consumption, and the correctness of our energy consumption complexity model. The energy consumption complexity is theoretical fundament which is contributed to designing energy efficient algorithms, choosing proper algorithms, and energy consumption on algorithms.

Keywords green computing; energy consumption measurement; algorithm energy consumption; energy consumption complexity; cross factor

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 大学的研究人员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;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[1].IT 企业已经属于能源密集型产业,计算机系统的能耗优化逐渐受到关注.

软件能耗优化研究通过优化软件降低整个计算机系统在完成给定运算时的能耗,例如优化软件本身的模块结构、代码、指令序列和算法[2].软件能耗优化研究面临的首要问题是软件能耗评估,其次是在系统功能和性能约束的前提下的能耗优化方法[3].软件能耗评估不仅要把软件执行所涉及的硬件能耗与软件各组成部分进行映射,更期望建立更高层次、更抽象的评估方法.软件由代码构成,而算法是代码的抽象表达,研究算法的能耗评价,可以更粗粒度地、脱离硬件环境地评价算法的能耗特性,指导低能耗的算法设计.

然而,脱离了执行环境,无法估计算法的真实能耗,但可以利用其他模型来评价算法的能耗特征.计算复杂性理论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[4].算法复杂度只关注算法本身,不考虑算法的上下文属性,具有高度的抽象性.算法和资源之间的关系通常用时间复杂度和空间复杂度表示,尚缺少算法规模和其消耗的电能之间的关系,即能耗复杂度研究.

算法是解决某个待定问题而定义的操作序列.本文研究的算法具有抽象性和无二义性:抽象性是

① Two Google searches equivalent to boiling a kettle, says scientist. http://www.computing.co.uk/ctg/news/1852749/two-google-searches-equivalent-boiling-kettle-scientist, 2016

指算法基于的计算模型和能耗度量均为抽象的概念，同图灵机算法具有相同的抽象层次；无二义性是指算法采用无二义的代码或指令表述. 算法的能耗复杂度可类比算法时间复杂度：脱离了执行环境，无法估计算法的执行时间/能耗，而只能采用时间/能耗复杂度这一模型，且时间/能耗复杂度中涉及的时间/能耗是抽象的、无量纲的，不能采用“秒/焦耳”度量. 时间/能耗复杂度研究算法消耗的时间/能耗随输入规模增加而增加的趋势，这种趋势通常采用大 O 渐进法（上界函数）表示.

本文首先扩展图灵机计算模型，定义图灵机算法的能耗复杂度. 但定义和求解图灵机算法的能耗复杂度步骤繁琐，因此，本文探求计算机算法的能耗复杂度求解方法. 众所周知，物理学中能耗等于实时功率在时间上的积分，计算机实时功率是关于时间的函数，功率浮动取决于计算机所运行的任务，即算法特征. 那么，算法能耗复杂度、时间复杂度和空间复杂度一定会既有联系又存在差异，如何定义三者之间的关系存在挑战. 本文通过易知的时间复杂度、空间复杂度和算法特征推导其能耗复杂度，极大地简化能耗复杂度的求解过程，该推导方法是本文的另一个贡献.

本文第 2 节介绍相关工作；第 3 节对经典图灵机进行扩展，定义能耗图灵机，作为能耗复杂度所依赖的计算模型；最后定义算法能耗复杂度；第 4 节设计根据算法时间和空间复杂度推导能耗复杂度的方法；第 5 节的验证实验共分 4 组：(1) 算法特征和算法能耗关系验证，以验证算法能耗复杂度中参数的正确性；(2) 算法执行时间与能耗关系验证，以验证算法的时间复杂度无法代替能耗复杂度；(3) 算法真实能耗随算法规模变化趋势与算法能耗复杂度是否一致，以验证算法能耗复杂度的正确性；(4) 算法时间复杂度和能耗复杂度与算法交叉度之间的关系分析. 第 6 节总结本文的研究工作，并且对研究进行展望.

2　相关工作

与本文最为相关的是算法能耗复杂度研究，现有研究成果非常少. 此外，算法或代码能耗评估研究以及基于形式化模型的软件系统能耗优化方法研究也与本文相关. 本节将从上述角度分析相关工作.

提及能耗复杂度概念的文献有 5 篇[5-10]. 文献[5]提出的“能量复杂度（Energy Complexity）”模型并非是算法能耗与算法输入规模之间的关系，而是算法能耗与性能的权衡关系，因此应称为“能耗的复杂特性”. 文献[6]提出的并非是一个通用的算法能耗复杂度模型，而是在一个有 n 个节点的网格（$\sqrt{n}\times\sqrt{n}$）上的计算问题的节点间通讯代价复杂度，尽管题目中提及“Energy”，但文中仅出现两次传感器节点能耗的描述，研究主要针对节点间的通讯代价与节点规模间的关系. 文献[7]将软件按照其能耗大小分类，采用的分类标准是“Complexity-Classification”，类似于亚线性 $O(\log n)$、线性 $O(n)$ 等类别，并非研究能耗复杂度. 本文与上述三篇论文存在明显的差异.

文献[8]针对并行算法，提出了一种渐进的能耗复杂度模型，将能耗复杂度表达为含并行算法中的核数量、单核最大频率与输入规模等参数的函数. 该模型类似于多核算法的时间复杂度，且对于非并行算法或单核环境，该模型与算法时间复杂度模型并无差异. 模型与硬件特征和算法特征之间的耦合度过高，普适性差. 算法并行化并不会改变算法的时间复杂度，而该文并未对比算法并行和串行时的能耗复杂度，也缺少实验验证. 本文与文献[8]均研究了计算机算法能耗复杂度，均采用大 O 渐进法，但在推导思路和适用环境上明显不同. 文献[9]将算法能耗分为处理器能耗与内存能耗，前者又分为处理单元能耗和处理器时钟能耗；后者又分为读写、休眠、待机和激活四种状态的能耗，并将内存能耗简化为读写、待机与激活状态的能耗之和. 该文采用的推导思路是估算能耗，然后进行约减. 例如在推导过程中，约简内存执行读写命令的能耗这一小项，并将内存功率归一化. 本文借鉴了其小项约简与适度近似的思路，也将内存功率视为常量. 文献[9]提出的能耗复杂度模型的参数与具体算法相关，不同算法参数不同，推导过程比较繁琐且通用性较差，而本文方法的优势在于通过容易获得的算法时间和空间复杂度来推导算法能耗复杂度.

文献[10]同样提出能耗图灵机和能耗复杂度，该文提出的能耗图灵机以九元组 $(Q,\Sigma,\Gamma,\delta,\xi,q_0,B,F,A)$ 表示，其中 ξ 为计算“图灵机在一次状态转移中消耗能量单元数量”的函数，将图灵机整体能耗表达为全部状态转移能耗之和，并通过大量的定理证明上述理论的正确性. 该文提出了将图灵机状态分为多个 ξ 函数单元的思路，讨论了时间复杂度与 I/O 复杂度极大情况下的 ξ 函数形式，但是并没有明确给出 ξ 函数表达或推导方法，而仅仅描述 ξ 函

数返回值必为正整数,虽然理论正确但是无法实际求解和应用.本文以八元组表征能耗图灵机 $M=(Q,\Sigma,\Gamma,\delta,q_0,B,F,\varepsilon)$,其中 δ 元组表示状态转移能耗,ε 表示读写头移动方向改变对能耗的影响,即3.3节提及的惯性能耗,随之定义了交叉度概念,模型较文献[10]更为细致.最重要的是,本文给出一般性的能耗复杂度推导方法,并有详细的实验验证.

代码能耗评价方法大多从程序的源代码着手,进而分析程序语句、算法和数据结构与能耗的关系.文献[11]通过源码分析建立解析树(ParseTree).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.该文还设计了解析树的等价变换方法,以优化代码能耗.文献[12]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.这些研究结论与特定的高级语言相关,并不具有抽象性.

算法是解题方法的一种抽象描述,代码是算法的实现.代码能耗研究与算法能耗研究相比,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.算法能耗评价和优化的现有研究成果很少,大多数研究或针对某一类算法展开,或是分析在特定执行环境下算法能耗评价规则.文献[13]研究了固态硬盘下的排序算法能耗优化;文献[14]对常见的排序算法的能耗进行量化地分析和比较,最后得出结论:"不同排序算法对应用软件能耗影响显著,且能耗影响与性能影响之间并无必然联系".但该文并未将排序算法的能耗规律抽象为能耗复杂度.

近些年涌现了很多基于自动机[15-16]、进程代数[17-18]和 Petri 网[19-20]等形式化模型优化软件系统能耗的研究成果.文献[15]在剖析传感器网络耗能原理的基础上,基于自动机理论设计网络控制层,调整传感器之间的交互,实现能耗优化;文献[16]设计了一种基于共享总线的、多核架构下的定时自动机,以实现多核系统的高性能和低能耗;文献[17]提出了一种适用于嵌入式软件系统能耗建模与分析的进程代数模型,扩展通信顺序进程(Communicating Sequential Processes,CSP)模型为 PTCSP(Priced Timed Communicating Sequential Process)模型,将系统能耗映射为模型代价;文献[18]同样是利用进程代数模型,针对实时系统设计了 RTCSP(Resource Timed Communicating Sequential Process)模型,并将资源与能耗映射.文献[19]基于有色 Petri 网提出了一种估计嵌入式实时系统能耗与性能的方法;文献[20]基于 Petri 网设计了一种自适应的框架,确保了系统的性能与能耗之间的平衡.上述模型大多包含形式化模型中"停止"、"等待"等算法能耗研究所不关注的状态,并且包含大量状态间转换细节,这些细节难以抽象并与算法能耗映射;多针对嵌入式软件能耗,应用范围窄,难以适用于一般性算法;虽然能较准确地估算系统能耗,但难以建立算法输入规模和其能耗之间的关系,无法用于推导算法能耗复杂度.与上述模型相比,图灵机模型抽象程度高,普适性好,更加适用于一般算法.因此本文基于图灵机模型定义算法能耗复杂度.

能耗优化研究成果可广泛应用到多个领域,例如:新能源供电的大型云数据中心[21]、支持细粒度能源监控的数据中心[22]、能耗与性能权衡的云计算平台[23]和智能网格与地理分布云计算平台[24]等应用场景,都有很大的能效优化空间.而这些应用场景中的软件与算法部分,都可以采用本文中的算法复杂度模型进行评价,并针对复杂度高的算法进行改进.

3 算法能耗复杂度

研究算法能耗复杂度,首先要解决两个问题.其一,能耗的度量单位是什么;其二,算法所依赖的计算模型,或抽象的执行环境是什么.

能耗复杂度以抽象能耗作为能耗度量,该抽象能耗并不以焦耳等物理概念为单位.能耗复杂度并不能给出一个算法执行过程中具体消耗的能量,其关注的是算法能耗随算法规模的增长趋势,以及如何比较这种趋势.

算法复杂度分析需要依托某种抽象计算模型,如图灵机、RAM① 以及 RASP②,而经典图灵机作为现代计算机的形式模型,具有很强的普适性,算法时间复杂度的研究首先基于经典图灵机,随后推广到其他计算模型.因此图灵机应作为本文研究算法能

① RAM(Random Access Machine)是一种抽象计算模型,是哈佛结构的一个示例,与图灵机等价.在 RAM 模型中,指令一条接一条地执行,没有并发操作,没有层次化的内存模型,指令及其执行代价可被精确定义,参见 https://en.wikipedia.org/wiki/Random-access_machine.

② RASP(Random-access Stored-program Machine)是存储了算法和其输入数据的 RAM,是冯诺依曼结构的一个示例,与通用图灵机等价,参见 https://en.wikipedia.org/wiki/Random-access_stored-program_machine.

耗复杂度的计算模型.但是,经典图灵机“七元组”中不包含表征算法能耗的元组,因此,难以将图灵机算法执行过程的每一步与其相应的抽象能耗关联,因此,经典图灵机不能直接用于能耗复杂度研究.

3.1　抽象能耗

抽象能耗与传统的、具有物理意义的、以焦耳为单位的能耗不同,抽象能耗是无量纲的正数,并不能代表算法执行过程中具体消耗的能量,但其与物理能耗是正相关的.抽象能耗以多项式为表达,并定义能耗以及能耗之间比较方式.本节所提及的“能耗”均为抽象能耗.

定义 1.　能耗集.能耗集 E 是多项式的集合,其包括常数级正整数、线性级多项式、对数级多项式、二次或高次多项式以及指数级多项式等,且每个量级的元素可以构成一个集合 E 的真子集,不同量级的多项式代表不同级别的能耗.

定义 2.　能耗(多项式)关系.能耗多项式中的小于等于关系“$\leq$”用于比较两个多项式的大小.以多项式中元素 n 为自变量,多项式的值为因变量,将多项式投影到二维坐标轴,当 n 相同时,若某一多项式曲线位于另一多项式曲线的上方,则认为该多项式较大,反之则该多项式较小.

关系“$\leq$”具有自反性、反对称性以及传递性,任意两个能耗集的元素均具有上确界和下确界,因此可得到定理 1 的结论.

定理 1.　在能耗集 E 上,关系“$\leq$”是偏序关系,并且此关系与能耗集 E 共同构成格$\langle E,\leq\rangle$.

另外,在格$\langle E,\leq\rangle$以及由该格所诱导的代数系统$\langle E,\vee,\wedge\rangle$中,二元运算 $\vee$(上确界)和 $\wedge$(下确界)具有交换律、结合律、幂等律等运算规律以及反对称性、保序性等运算性质.格中运算规则与算法复杂度渐进计法中大 O 表示法的计算规则一致,即若假设存在算法 A 的能耗复杂度为 $O(n^2)$,算法 B 的能耗复杂度为 $O(n^3)$时,当其二者顺序执行,则整个执行过程的能耗复杂度为两者之间较大者 $O(n^3)$,而此计算过程等价于 $n^2\vee n^3=n^3$.另外,本节仅使用多项式作为抽象能耗的数学表达,多项式内的 n 并无实际意义,但是后文将 n 赋予其与时间复杂度中的 n 相同的意义,即代表算法的输入规模.

3.2　能耗图灵机

经典图灵机依靠读写头移动以及对带上符号的读写操作执行算法,这种计算模型可以有效地分析算法是否可以在有限步内求解,可以定义 P 问题与 NP 问题;但是经典图灵机并不能将算法执行的每一步操作映射为算法能耗.因此,本文定义经典图灵机中转移函数与能耗的映射关系,并在图灵机“七元组”内增加与算法能耗相关的元素.

定义 3.　能耗图灵机.能耗图灵机 E-TM 是一个八元组:$M=(Q,\Sigma,\Gamma,\delta,q_0,B,F,\varepsilon)$.其中:$Q$ 是状态的有穷集合;$q_0\in Q$ 是 M 的初始状态;$F\subseteq Q$ 是终止状态集合;Γ 是带上可用符号的有穷集合;$\Sigma\subseteq\Gamma-\{B\}$是输入符号集合;$B\in\Gamma$ 是空白符;δ 是 $Q\times\Gamma\times Q\times\Gamma\times\{R,L\}$上的 E 值子集,即 $\delta:Q\times\Gamma\times Q\times\Gamma\times\{R,L\}\to E$ 是 E-TM 的转移函数,其表征能耗图灵机执行算法每一步对执行算法的能耗影响.$\forall p,q\in Q$, $x,y\in\Gamma$, $\delta(q,\ x,\ p,\ y,\ d)$表示“$M$ 在状态 q 读入符号 x,状态改为 p,并在 x 所在的带方格中写入符号 y 然后将读写头向左($d=L$)或者向右($d=R$)移动一格”的能耗,此操作过程也可记作 $\delta(q,x)=(p,y,d)$;ε 是$\{R,L\}^*$ 上的 E 值子集,即 $\varepsilon:\{R,L\}^*\to E$ 是 E-TM 的惯性能耗函数,表征能耗图灵机读写头移动方向之间的顺序关系对能耗影响.

例如,$\varepsilon(L,L,R,R,L)$表示“能耗图灵机以读写头以向左移动两格,再向右移动两格,最后向左移动一格”的能量消耗.

定义能耗图灵机 M 的即时描述(Instantaneous Description):$X_1X_2\in\Gamma^*,q\in Q,X_1qX_2$称为 M 的即时描述(id).设 $X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n$是 M 的一个 id,如果此时存在 $\delta(q,X_i,p,Y,R)\in\delta$,那么 M 的下一个 id 将为:$X_1X_2\cdots X_{i-1}YpX_iX_{i+1}\cdots X_n$,此 id 转移过程可记作 $X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n\Rightarrow_M X_1X_2\cdots X_{i-1}YpX_iX_{i+1}\cdots X_n$;如果此时存在 $\delta(q,X_i,p,Y,L)\in\delta$,那么 M 的下一个 id 为:$X_1X_2\cdots pX_{i-1}YX_{i+1}\cdots X_n$.同理,将此 id 的转移过程记作 $X_1X_2\cdots X_{i-1}YpX_iX_{i+1}\cdots X_n\Rightarrow_M X_1X_2\cdots pX_{i-1}YX_{i+1}\cdots X_n$.另外,令 $id(M)=\Gamma^*\times Q\times\Gamma^*$表示 M 的所有 id 的集合,并可使用符号$\Rightarrow_M^*$表示多次此类 id 的转移的能量消耗过程集合,若存在$(id_1,id_2)\in\Rightarrow_M^*$,则表示在此图灵机过程中,存在瞬时描述由 id_1转移为 id_2的情况,可表示为$\Rightarrow_M(id_1,id_2)$,因此记$\Rightarrow_M(id_1,id_2)$为能耗图灵机读写头移动方向的转移.

图灵机的主耗能操作是转移函数执行时的两种操作:(1)读写头移动;(2)在带上擦写字符.并且每种操作都存在其对应的能耗值.因此可以以“操作次数×单次操作能耗”的方式计算图灵机执行时某操作的能耗,而整个图灵机的状态转移能耗为全部操

作能耗的和.为了更容易、更清晰地理解能耗图灵机 M,下面将描述 M 所识别的逻辑语言的两类原子能耗操作:

(1)"M 在状态 q 读入符号 x,状态改为 p,并在 x 所在的带方格中写入符号 y 然后将读写头向左($d=L$)或者向右($d=R$)移动一格",记作"$\delta(q,x,p,y,d)\in\delta$".第一类原子能耗操作的能耗可由式(1)求得.

$$\bigvee_{i=0}^{n}(\Rightarrow_M(id_i,id_{i+1})) \tag{1}$$

(2)"能耗图灵机的读写头移动顺序改变,即 M 执行算法过程其读写头移动顺序中出现$\langle L,R\rangle$或者$\langle R,L\rangle$的有序移动方向对".第二类原子能耗过程的能耗可由式(2)求得,其中 $d_1\neq d_2$.另外,若图灵机允许读写头在一次转移函数的执行过程中保持不动,即此时读写头仅进行读写操作,则可重新定义 $\delta:Q\times\Gamma\times Q\times\Gamma\times\{R,S,L\}\rightarrow E$.

$$\bigvee_{i=0}^{n}(\Rightarrow_M(id_1,id_2)) \tag{2}$$

综上所述,使用能耗图灵机计算算法能耗,即为对图灵机执行过程中两类原子能耗操作计数,并通过汇总计算得到.能耗图灵机有着广泛的适用性,稍加改动可以随之定义出多带能耗图灵机、不确定能耗图灵机以及通用能耗图灵机.关于能耗图灵机的适用性描述参见附录 A.

3.3 算法能耗复杂度

图灵机是一种计算模型,也是一种算法描述语言,其基本数据结构是字符串,基本语法是图灵机转移函数的语法,一个图灵机的所有转移函数共同组成某算法的执行过程.图灵机算法,即算法的图灵机描述,与算法的其他描述方式(如伪码和流程图)等价.算法能耗复杂度可由能耗图灵机为计算模型分析得到.根据前文能耗图灵机定义可知,图灵机执行过程中存在两种原子能耗操作,本节分别从这两种原子能耗操作出发,逐项考虑其与算法能耗复杂度的关系,最后定义能耗复杂度,并分析能耗复杂度渐进表达,对比能耗复杂度和时间复杂度.

(1)状态转移能耗

能耗集合 E 以及$\leq$关系共同构成的偏序集$\langle E,\leq\rangle$是一个格,能耗集合 E 中的元素为多项式;而能耗图灵机的转移函数 δ 是 $Q\times\Gamma\times Q\times\Gamma\times\{R,L\}$上的 E 的子集.因此图灵机中任意一个转移函数都对应着一个量级的能耗.由此可见,对于任意两个转移函数 δ_1 和 δ_2 所对应的能耗 a 和 b,其全部执行完成的能耗复杂度可以使用 $a\vee b$ 表示.设一个图灵机共存在 k 个转移函数,且第 i 个转移函数的执行能耗为常数 e_i,执行次数为 u_j,则图灵机状态转移能耗如式(3)所示:

$$E\approx(e_1\vee)^{u1}(e_2\vee)^{u2}\cdots(e_i\vee)^{ui}\cdots(e_k\vee)^{uk} \tag{3}$$

进一步细分,状态转移能耗包含的是读写头移动能耗、读写头在纸带上读写的能耗和存储能耗.存储能耗是指图灵机纸带保存数据所消耗的能量,在纸带条数固定的前提下,存储能耗仅与存储时间有关,而与存储数据量无关.若将纸带视为耗电设备,其功率是稳定的,能耗等于功率与工作时间的乘积.由此可见,存储能耗与算法的输入规模有关,输入规模越大,算法执行时间越长,存储能耗越高.

后文将介绍,读写头移动能耗对应 CPU 逻辑运算耗能,读写头在纸带上读写能耗对应 CPU 访问内存运算能耗和内存读写状态的能耗,存储能耗对应计算机内存存储数据的能耗.

(2)惯性能耗

状态转移能耗是图灵机单步操作能耗的累加,并没有考虑步骤之间的关系,因此本文提出交叉度的概念.图灵机中读写头移动时方向改变为一次交叉,而图灵机执行过程中的交叉次数与读写头移动总次数之间比例称为交叉频度,简称交叉度.交叉度与图灵机能耗正相关.从物理意义上而言,物体在静止或保持运动时,若改变其运动状态,则物体自身会对这种变化产生一定程度的阻力,这就是物体惯性的表现,克服该阻力需要能量.类比图灵机的读写头的移动过程,当图灵机改变其移动方向时会因为读写头惯性的存在而消耗额外能量.

定义 4. 交叉度 α(图灵机算法).图灵机算法包含图灵机读写头的移动操作,在其操作序列中包含$\langle L,L\rangle$,$\langle R,R\rangle$,$\langle L,R\rangle$或$\langle R,L\rangle$的有序对,其中$\langle L,R\rangle$和$\langle R,L\rangle$有序对的个数与所有移动操作个数之间的比值称为图灵机算法的交叉度.

图灵机算法交叉度 α 是关于算法输入规模 n 的函数,即 $\alpha=R(n)$.且即便每两次移动均产生交叉,交叉度也不大于图灵机读写头总移动次数.给定算法以及输入规模时,交叉度是一个确定值,且 $R(n)$最大为 n 的一阶多项式.

(3)能耗复杂度

基于前文所述,本文对算法的能耗复杂度作如下定义.

定义 5. 算法能耗复杂度.算法能耗为状态转移能耗与惯性能耗之和,因此算法能耗复杂度为一个与问题输入规模以及交叉度相关的多项式,前者

对应状态转移能耗，后者对应惯性能耗. 设算法输入规模为 n，交叉度为 α，算法能耗复杂度 $E(n,\alpha)$ 定量表征算法能耗以及算法能耗随算法的输入规模增长而增长的趋势.

由定义 4 知，$\alpha=R(n)$，因此算法能耗复杂度 $E(n,\alpha)$ 是关于 n 的函数，可以简记为 $E(n)$. 能耗复杂度的渐进表达可比照算法时间复杂度相关知识. 给定算法的时间复杂度 $T(n)$ 和能耗复杂度 $E(n)$ 的渐进表达式是一致的，因为状态转移函数的执行时间和能耗为同阶多项式，而交叉度定义可知 $R(n)<T(n)$. 算法时间复杂度和能耗复杂度多项式的高阶项相等. 尽管如此，$T(n)$ 与 $E(n)$ 并不等价.

首先，能耗等于功率乘以时间，若功率为常量，则类比算法时间复杂度定义的能耗复杂度会和前者一致. 按现有实践经验，应不存在时间上可以完成但能耗上无法完成的算法，即不存在经典图灵机上的 NP 问题，而在能耗图灵机上为 P 问题. 然而，计算机功率并非恒定，相同时间复杂度算法会有不同的能耗特征. 因此，本文后续研究将不采用复杂度的渐进表达，而关注时间复杂度和能耗复杂度的差异.

其次，若已知算法的能耗复杂度和时间复杂度的高阶项相同，那么在研究算法能耗复杂度时即包含了时间复杂度的研究，且可以忽略高阶项（算法时间复杂度部分），仅考虑算法能耗复杂度的低阶项. 这种分析方法与现存算法能耗优化思路一致："在算法性能（执行时间）一定的情况下优化算法能耗"或"在算法性能（执行时间）已经优化的情况下考虑算法能耗优化问题". 若解决某一个问题的多个算法的时间复杂度相等时，可以通过能耗复杂度的低阶项，选择能耗更低（增长慢）的算法.

最后，由于算法的能耗复杂度和时间复杂度的高阶项相同，且易知一个算法的时间复杂度，若能建立算法时间复杂度与能耗复杂度之间的函数关系，则可以避免使用抽象的图灵机算法来推导能耗复杂度. 本文第 4 节将描述这一方法.

4　能耗复杂度推导方法

能耗图灵机的转移函数描述图灵机算法执行的每一个步骤，这种描述非常复杂，因此将现有计算机算法转换成图灵机算法很繁琐，求解算法能耗复杂度也很困难. 本节将针对计算机算法和 RAM 模型研究如何推导算法能耗复杂度. 拟从已知的算法空间复杂度、时间复杂度和交叉度入手，分析三者对能耗复杂度的影响，并确定它们之间函数关系，以推导能耗复杂度.

表 1 列出了本节涉及的主要符号及其含义. 若算法的时间复杂度、空间复杂度和交叉度已知，令 $E(n)=f(T(n),S(n),R(n))$，若能够定义 f 函数的表达式，则算法能耗复杂度可迎刃而解.

表 1　符号说明

描述项	符号
时间复杂度	$T(n)$
空间复杂度	$S(n)$
能耗复杂度	$E(n)$
交叉度	$R(n)$
影响计算机功率的参数集	W
计算机功率	$power(W)$
CPU 频率和使用率	f,ω
运算（CPU）语句	U
存储（I/O）语句	I
算法的能耗复杂度 $E(n)$、时间复杂度 $T(n)$、空间复杂度 $S(n)$ 和交叉度 $R(n)$ 之间的关系	$E(n)=f(T(n),S(n),R(n))$
时间复杂度 $T(n)$ 与时间 T 的函数关系	$T=g(T(n))$
能耗复杂度 $E(n)$ 与能耗 E 的函数关系	$E=h(E(n))$
频率 f 和使用率 ω 之间的函数关系	$f=scal(\omega)$
交叉度 $R(n)$ 和使用率 ω 间的函数关系	$\omega=idle(R(n))$

4.1　空间复杂度

本小节判断空间复杂度是否会独立影响能耗复杂度，即空间使用大小是否会影响算法能耗. 作为数据存储硬件，首先讨论内存的工作状态，一是无数据访问的空载状态（Idle），二是有数据访问时的工作状态（Loaded）. 前者消耗维持内存数据所需要的最小能量，后者则叠加内存读写数据时消耗的能量.

由此可见，内存功率包括两个部分，一是读写内存数据时内存功率，称为 I/O 功率，对应图灵机状态转移能耗中纸带读写能耗；二是存储数据态内存功率，称为存储功率，对应图灵机状态转移能耗中的存储能耗；有文献[①]认为，前者是后者的一倍. 但是内存的总功率很低，普通 DDR3（8 GB）内存的功率在 4～8 瓦特之间浮动，DDR4 的最大功率甚至低至 5 瓦特[②]. 绝大部分算法执行时内存均会处于读写状态，因此在实际计算中，可以近似地将内存功率视为

① Gottscho M, Gupta P, Kagalwalla A A, et al. Analyzing Power Variability of DDR3 Dual Inline Memory Modules, Available in https://www.researchgate.net/publication/262915126_Analyzing_Power_Variability_of_DDR3_Dual_Inline_Memory_Modules

② http://www.samsung.com/global/business/semiconductor/file/media/DDR4_Brochure-0.pdf

I/O 功率和存储功率之和,且为常量.

空间复杂度用来度量算法在运行过程中临时存储空间随输入规模增加而增加的趋势.首先,在硬件环境不变的前提下,内存空载状态的能耗与存储的数据量是无关的,存储功率为常量,空间复杂度对存储功率无影响.其次,空间复杂度可以表征内存写操作的频繁程度,而内存读操作实际包含在时间复杂度中,I/O 功率难以确定.

由于内存的总功率很小,因此将 I/O 功率也视为常量.那么,内存的能耗就仅取决于算法执行时间.由此可见,算法空间复杂度对能耗复杂度的影响近似地包含在算法时间复杂度对能耗复杂度的影响中.在后文的式(5)中,常量 a_4 即为内存、硬盘等功率较小或近似稳定的计算机组件功率.

综上所述,若视内存功率为常数,那么内存能耗仅与算法执行时间相关,因此 $S(n)$ 对 $E(n)$ 的影响将包含在 $T(n)$ 对 $E(n)$ 的影响中,可以令 $E(n)=f(T(n),R(n))$.

4.2 时间复杂度

无论是计算机算法还是图灵机算法,算法执行时间和执行能耗之间的关系应遵守"能耗等于时间和功率的乘积"①这一物理定理,该定理既适用于抽象的图灵机,也适用于具体的计算机,因此能耗复杂度和时间复杂度的潜在关联是求解 $E(n)=f(T(n),R(n))$ 的关键.但抽象的图灵机功率特性无法获得,因此本文考虑具体的计算功率特性:本节将计算机功率对应图灵机功率,算法在计算机上的执行时间/能耗对应算法在图灵机上的执行时间/能耗,在计算机模型下求解.

计算机都标有额定功率,而算法的执行时间已知.设影响计算机功率的参数集为 W,那么 T 时间内计算机能耗计算公式:

$$E=power(W)\times T \tag{4}$$

其中,$power$ 表示计算机功率与特征参数之间的函数关系.文献[25]指出计算机功率和 CPU 频率的关系为 $P=P_{fix}+P_f\times f^3$.其中,P_{fix} 为常数,表示除 CPU 以外其他设备的功率;P_f 为 CPU 功率系数,f 为 CPU 频率;而文献[26]认为计算机功率与 CPU 使用率 ω 和 CPU 频率 f 相关,定义函数 $power(f,\omega)=a_1f^3+a_2\omega f^3+a_3\omega+a_4$($a_1,a_2,a_3,a_4$)均为硬件相关的系数,其中常量 a_4 即为内存、硬盘等功率较小或近似稳定的计算机组件功率).大部分研究均假设除 CPU 以外的其他计算机组件功率稳定.除此之外,部分研究认为计算机的 CPU、主板、内存以及磁盘的工作状态均是计算机功率相关的特征属性[27].按文献[26],计算机功率采用 CPU 频率 f 函数、CPU 使用率等参数估算.

但计算机运行时功率是动态变化的,功率大小与计算机的繁忙程度相关:当计算机空闲时,功率较低;反之亦反.同理 W 值也会随时间变化.因此算法执行期间所消耗的能量与时间的关系并不是简单的线性正相关.因此式(4)的计算结果未必精确.在能耗复杂度的推导过程中,本文采用适度近似的方法,认为 W 的值为算法执行时间内的平均值或具有代表性的特征值,因此 $power(W)$ 即为计算机平均功率.本文在 4.4 节继续上述推导.

4.3 交叉度

计算机算法由语句构成,而语句可以分为运算语句和存储语句两类,其中运算语句是指 CPU 对缓存内数据进行运算的语句,而存储语句是指访问内存数据的语句.类比图灵机算法的交叉度定义,算法能耗不仅与运算语句和存储语句数量有关,还和两种语句之间的关系相关.不考虑具体语句特征,若运算(存储)语句后紧接一条存储(运算)语句,则认为两者有交叉关系,否则没有交叉关系.类比能耗图灵机,定义计算机算法中交叉度为运算语句和存储语句的交叉频率.交叉频率不同,语句排列方式就会不同,进一步会影响算法的能耗,原因如下:

(1)主要原因.CPU 的功率与 CPU 频率正相关,根据 CPU Frequency Scaling② 和 CPU Frequency Governor③ 技术,当 CPU 使用率较低时将主动降低频率以节省电能,因此,当执行连续的存储语句时,交叉度低,CPU 会处于连续空闲状态,Scaling 技术发挥作用;而当存储语句和运算语句密集交错执行时,交叉度高,CPU 空闲时间片很短,Scaling 技术难以实施④,CPU 一直处于高频率高功率状态.后文会介绍,算法在 CPU 高频执行阶段和低频执行阶段的比例,可由交叉度的定义以及交叉度和 CPU 使用率之间的关系估算.

(2)次要原因.交叉度高会增加缓存失效的概率,导致额外的访存操作,消耗额外的能量.

(3)无关因素.CPU 的功率与 CPU 使用率正相关,执行存储语句时 CPU 使用率降低,CPU 功率随之降低.例如,多核 CPU 的使用率降低时部分核会关闭,功耗线性下降.但是,上述特性仅与存储语句

① 更精确的表述应该是实时功率函数在时间 T 上的积分,本文将积分形式简化为乘积形式.

② https://en.wikipedia.org/wiki/Frequency_scaling

③ https://en.wikipedia.org/wiki/Governor_%28device%29#Computing, 2016

④ Linux 提供的 CPU Frequency Governor 技术,CPU 使用率的采样间隔在毫秒级别.

的数量有关，而与存储语句的执行顺序无关；换言之，上述特性与交叉度无关，无论交叉度大或小，一个算法的存储语句数量是确定的，交叉度并没有改变计算总量，而只是改变了 CPU 使用率在时间上的分布.

定义 6.　交叉度 α(计算机算法). 计算机算法的交叉度 $\alpha=R(n)$是算法中运算语句和存储语句交叉的频率. 设算法由语句序列$\langle I_1,U_1,I_2,U_2,\cdots,I_k,U_k\rangle$组成，$U$ 代表运算语句集，I 代表存储语句集，下标为操作逻辑时钟. 则该语句序列中$\langle I,U\rangle$语句子序列个数为 $k(k\geq 1)$. 在输入规模为 n 的条件下，整个算法执行的语句交叉了 kn 次，设算法时间复杂度为 $T(n)$，定义算法执行时间为 $T=g(T(n))$，可知 kn 为运算语句和存储语句交叉次数，则交叉频率为 $kn/T=kn/g(T(n))$，该频率定义为计算机算法交叉度.

图 1 表述了语句顺序对交叉度的影响，5.1 节和 5.4 节实验证明了交叉度对算法能耗的影响.

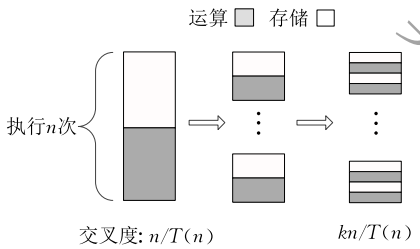


图 1　交叉度示意图

4.4　推导步骤

给定一个计算机算法，本小节根据公式 $E(n)=f(T(n),R(n))$推导 $E(n)$，其中 $T(n),R(n)$均为已知函数，因此推导的关键是确定 f 函数的表达.

首先，在不采用渐进记法的前提下，任意算法的时间复杂度 $T(n)$与运行时间 T 必然存在线性关系，记为 $T=g(T(n))$. 同理，算法的能耗复杂度 $E(n)$与算法能耗 E 也存在线性关系，即为 $E=h(E(n))$.

其次，函数 *power* 是表示计算机功率与功率相关参数集的函数，文献[25]中的 $power(W)$函数如式(5)所示：

$$\begin{aligned} power(W)&=power(f,\omega)\\ &=a_1f^3+a_2\omega f^3+a_3\omega+a_4 \end{aligned}\tag{5}$$

其中 f 为 CPU 频率，ω 为 CPU 使用率，a_1,a_2,a_3,a_4均为硬件相关的系数，其中常量 a_4即为内存、硬盘等功率较小或近似稳定的计算机组件功率. 式(5)中采用 f 和 ω 的均值来代替瞬时值.

再次，因为算法交叉度主要通过 CPU Frequency Scaling 技术影响 CPU 频率和功率，进而影响能耗. 如果 ω 持续较低时，CPU Frequency Scaling 将会降低 CPU 频率 f. 而交叉度 $R(n)$影响着降频的触发条件，交叉度越大，ω 持续较低的可能就越小，降频的可能也就越小. 设存在交叉度的临界值 R_0，使得当 $R(n)<R_0$时 CPU 会降频，且 $f=scal(\omega)$；当 $R(n)\geq R_0$时，$f=f_{\max}$. 由此可见，CPU 使用率 ω 与交叉度 $R(n)$之间存在函数关系，设 $\omega=idle(R(n))$，因此可以得出结论：n 与 $power(W)$之间存在函数关系 P，使得

$$\begin{aligned} power(W)&=power(f,\omega)=power(scal(\omega),\omega)\\ &=power(scal(idle(R(n))),idle(R(n)))\\ &=P(n) \end{aligned}\tag{6}$$

接着，式(6)中$R(n)=kn/g(T(n))$. 由文献①(参见上页脚注④)知，*scal* 函数为单调递增的线性函数；当 $R(n)\geq R_0$时，*idle* 函数值为常量. 当 $R(n)<R_0$，可以确定常量 r，使 $\omega=idle(R(n))=O(r\times R(n))$，采用 *idle* 函数的上界函数代替未知的 *idle* 函数. 式(6)中的 *power* 函数，*scal* 函数和 *idle* 函数均为已知函数.

最后，由式(6)可知：

$$\begin{aligned} &P(n)=power(scal(idle(kn/g(T(n))))),\\ &\quad idle(kn/g(T(n))),\ R(n)<R_0,\\ &P(n)=P_{\max},\ R(n)\geq R_0 \end{aligned}\tag{7}$$

结合式(5)，经合并系数，简单整理可得 $P(n)$函数的形式为

$$\begin{aligned} &P(n)=b_1n^4/g(T(n))^4+b_2n^3/g(T(n))^3+\\ &\quad b_3n/g(T(n))+b_4,\ R(n)<R_0,\\ &P(n)=P_{\max},\ R(n)\geq R_0 \end{aligned}\tag{8}$$

式中，b_1,b_2,b_3,b_4为常数，$P_{\max}>0$.

综上所述，已知：

$$\begin{aligned} &E(n)=f(T(n),R(n)),\\ &E=power(W)\times T,\\ &T=g(T(n)),\\ &E=h(E(n)),\\ &power(W)=P(n) \end{aligned}\tag{9}$$

那么可得

$$\begin{aligned} E&=h(E(n))=power(W)\times T\\ &\Rightarrow h(E(n))=P(n)\times T\\ &\Rightarrow E(n)=h^{-1}(P(n)\times e(T(n))) \end{aligned}\tag{10}$$

① CPU frequency and voltage scaling code in the Linux(TM) kernel. https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

最终，可将 f 函数整理为如下形式：

$$E(n)=c_1T(n)+c_2n^4/T(n)^3+c_3n^3/T(n)^2+c_4n,\quad R(n)<R_0,$$
$$E(n)=c_5T(n),\ R(n)\geq R_0 \tag{11}$$

式中，$c_1\sim c_5$ 为常数.

由式(11)可知，算法的能耗复杂度和时间复杂度高阶项相同，符合第 3 节的能耗图灵机模型；当 $R(n)\geq R_0$，两者线性相关，可以类比 c_5 为计算机稳定功率. 值得注意的是，当算法的时间复杂度为常数时，算法的能耗复杂度高阶项为 n^4，这显然与实践相悖. 事实上，此时算法能耗复杂度也为常数，算法在输入规模为 n 时执行的语句和输入规模为 1 时执行的语句相同，算法交叉数量为 k，交叉频率也为常量 k/T，均与 n 无关.

5 实验分析

本节实验共分 3 个部分：5.1 节验证了交叉度和算法能耗之间的关系；5.2 节验证了算法执行时间与能耗并不等价，算法的时间复杂度无法代替能耗复杂度；5.3 节证明算法真实能耗随算法规模变化趋势与算法能耗复杂度是一致的，以验证算法能耗复杂度的正确性；5.4 节分析了真实算法中不同交叉度对算法的时间复杂度和能耗复杂度的影响. 实验在真实环境下对计算机算法执行期间的能耗进行测量，实验环境包括实验节点、监控节点以及数据处理与分析节点. 具体实验环境如表 2 所示. 实验中用到的测试用例如表 3 所示.

表 2 实验环境

项	描述
计算机	清华同方超翔 Z900 计算机，Intel Core i5-2300 2.80 GHz，8 GB 内存，1 TB 硬盘，功率在 50 至 100 瓦特之间浮动.
操作系统	CentOS 5.6，Linux 2.6.18 内核，CPUFreq Governor 为 Conservative 模式.
功率计	HOPI-9800，自身功耗小于 1 W，采样时间间隔为 1～5 s，可调.
监控软件	用电检测仪数据分析系统，通过 USB 接口将 HOPI-9800 功率计获得的实验节点能耗数据传输至监控节点.
程序语言	Java(jdk-1.7.0)
IDE	Eclipse 4.3
测量单位	能耗单位：焦耳(Joule)；时间单位：秒(s)；交叉度：无量纲；算法规模：数据项个数.
参数	式(5)至(10)中的参数，如 $a_1\sim a_4$，r，g 函数，h 函数，最后可确定 $c_1\sim c_4$ 的值. 上述参数只是为了推导能耗复杂度的一般形式，$E(n)=c_1T(n)+c_2n^4/T(n)^3+c_3n^3/T(n)^2+c_4n$. 在实验中，令 $c_1\sim c_4$ 的值为 1，无论是能耗复杂度还是时间复杂度都是一种趋势，通常用上界函数表示，本身就非精确函数.

表 3 测试用例集描述

测试用例	描述	时间复杂度(近似) 能耗复杂度($R(n)\geq R_0$)	能耗复杂度(近似) ($R(n)<R_0$)	交叉度
运算语句	计算 π^2，π 保留 8 位小数	n	n	0
存储语句	随机访问大链表中的一个大对象	n	n	0
Search	线性搜索	n	n	1
InsertSort	插入排序	$(n^2-n)/2$	$n(n-1)/2+8n/(n-1)^3+4n/(n-1)^2+n$	$4/(n-1)$
MergeSort	归并排序	$n\log n+n$	$n\log n+2n+n/(\log n+1)^3+n/(\log n+1)^2$	$1/(\log n+1)$
LCS	最长公共子序列算法	$3n^2+2$	$3n^2+n+n^4/(3n^2+2)^3+n^3/(3n^2+2)^2+2$	$4n/(3n^2+2)$
Floyd	Floyd-Warshall 最短路径算法	n^3+2n^2	$n^3+2n^2+n+1/n^2(n+2)^3+1/n(n+2)^2$	$1/(n^2+2n)$

5.1 交叉度和算法能耗关系验证

本实验验证交叉度会影响算法能耗. 实验设计一个含有固定语句条数的算法，探讨在不同的存储语句的比例下，两种语句的交叉度对算法能耗的影响. 因为实验用例并非实际算法，故将交叉度简单分为零交叉，半交叉和逐句交叉. 零交叉表示先执行所有的存储语句，后执行所有的运算语句；逐句交叉表示每条运算(存储)语句与若干存储(运算)语句交替出现①；半交叉则介于两者之间②.

图 2 的横轴为存储语句交叉比例，纵轴为这些语句在实验计算机上执行的测量能耗，单位焦耳(J). 由图 2 可知：相同的运算和存储语句规模下，随着交叉程度的增加算法能耗明显增加，近似为线性关系；相同交叉程度下，由于实验中存储语句比运算语句更加耗能，代码能耗与存储语句比例正相关. 实验结果证实了交叉度对算法能耗的影响.

① 逐句交叉：设有 2 条运算语句，8 条存储语句，则每 1 条运算语句后接 4 条存储语句交叉，反之则反.

② 半交叉：设有 2 条运算语句，8 条存储语句，则 4 条存储语句后接 2 条运算语句，再接 4 条存储语句.

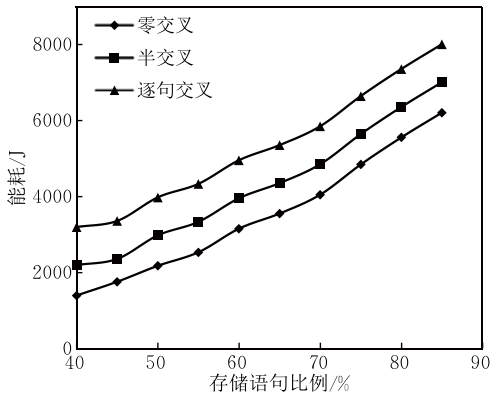


图 2　不同交叉度的语句能耗图

5.2　算法执行时间与能耗关系验证

本节证明算法执行时间与能耗并非为等比例关系. 首先，第 1 组实验对比了 InsertSort、MergeSort、Floyd 和 LCS 这 4 个用例能耗，实验通过调整算法规模(n)以保证用例执行时间大致相同，比较它们的能耗以证明执行时间相同的不同算法的能耗并不相同. 实验结果如图 3 所示.

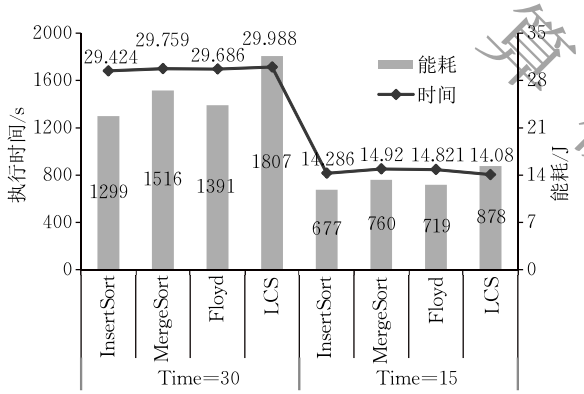


图 3　不同算法在执行时间相等时能耗对比图

从图 3 中可以看出，尽管 4 个算法执行时间近似相同，但能耗差距明显. 当算法执行时间约 30 s 时，LCS 算法的能耗比 Floyd 算法能耗高出 29.9%；同理，InsertSort 算法能耗比 MergeSort 算法能耗高出 16.6%；降低上述 4 个算法的输入规模，使执行时间约为 15 s，此时 4 个算法能耗规律不变，但能耗差距发生变化，LCS 算法的能耗比 Floyd 算法能耗高出 22.1%；同理，InsertSort 算法能耗比 MergeSort 算法能耗高出 12.3%. 因此可以得出如下结论：(1) 时间相同的算法能耗并不一定相同，不能简单地采用算法执行时间来表征执行能耗；(2) 不同算法的能耗差异也非稳定的，随着算法输入规模的变化，即使算法执行时间相同，能耗差异亦在变化；(3) 对于同一个算法，当通过改变输入规模而减小其执行时间，能耗并非等比例变化，时间复杂度无法替代能耗复杂度. 上述均证明了采用算法能耗复杂度而非时间复杂度来表征算法间的能耗特征差异的必要性.

5.3　算法能耗复杂度验证

验证一个算法的能耗复杂度是否是 $E(n)$ 的基本思路为：对于同一算法，比较在不同输入规模下，能耗的测量值与 $E(n)$ 计算值是否一致，$E/E(n)$ 是否为常量. 对 n 的某个范围计算 $E/E(n)$，其中 E 是实际运行时算法能耗测量值. 如果 $E(n)$ 是执行能耗的理想近似，那么 $E/E(n)$ 会收敛于正数；如果 $E(n)$ 估计过大，则 $E/E(n)$ 收敛于 0；如果 $E(n)$ 估计过小，则 $E/E(n)$ 的值逐渐增大且无法收敛. 在推导算法能耗复杂度时采用第 4 节提出的 $E(n)=f(T(n),R(n))(R(n)<R_0)$ 的方法.

文献[28]亦使用此方法评价算法时间复杂度的正确性. 本实验对时间复杂度为 $n^2\log n$ 的算法①测试不同输入规模下的执行时间 T 和不同 $T(n)$ 之间的比例. 表 4 给出运行时间以及 3 种 $T/T(n)$ 的值.

表 4　运行时间示例

n	time(T)	T/n^2	T/n^3	$T/(n^2\log n)$
100	22	0.002200	0.000022000	0.0004777
200	56	0.001400	0.000007000	0.0002642
400	207	0.001294	0.000003234	0.0002159
500	466	0.001294	0.000002157	0.0002024
800	846	0.001322	0.000001652	0.0001977
1000	1362	0.001362	0.000001362	0.0001972
1500	3240	0.001440	0.000000960	0.0001969
2000	5949	0.001482	0.000000740	0.0001947
4000	25720	0.001608	0.000000402	0.0001938

其中，当 $T(n)=n^3$ 时，$T/T(n)$ 的值渐进收敛于数值 0；当 $T(n)=n^2$ 时，$T/T(n)$ 随 n 逐渐增长，当 $T(n)=n^2\log n$ 时，$T/T(n)$ 收敛于 0.00019，实验结果与上述方法一致.

实验中操作系统和 Java 虚拟机都会消耗部分能耗. 从测量手段角度，当应用软件执行时，无法区分应用软件、中间件、操作系统的能耗，就好比无法区分它们的执行时间一样. 但 Java 虚拟机能耗不会改变算法能耗复杂度. 理由如下：(1) Java 语言是转换为指令在虚拟机中执行，虚拟机不会因为执行的程序不同，而导致同一指令的执行能耗有所差异. 虚拟机的能耗由指令类型和数量决定，恰好反映了算法能耗. 虚拟机能耗和算法能耗是一致的；(2) 实验避免调用 Java 类库代码，否则会使算法编译后的指令数量明显增加，改变算法能耗复杂度；(3) 算法能

① 归并排序增加了一个外层循环，即相当于 ν 次归并排序.

耗复杂度是一个抽象的数学模型，通过实验可以验证规律性，而非验证模型计算值和实验值的拟合.

实验选择线性搜索、插入排序、归并排序、最长公共子序列、最短路径算法为测试用例，考虑算法在不同输入规模下执行时的执行时间 T、能耗 E 以及 $E/E(n)$值. 为了便于算法间比较，将 5 个算法在不同规模下的 T、E 和 $E/E(n)$值按 Max-Min 法归一化到[0,1]区间. 图 4 展示了 5 个算法的输入规模与 $E/E(n)$值之间的关系.

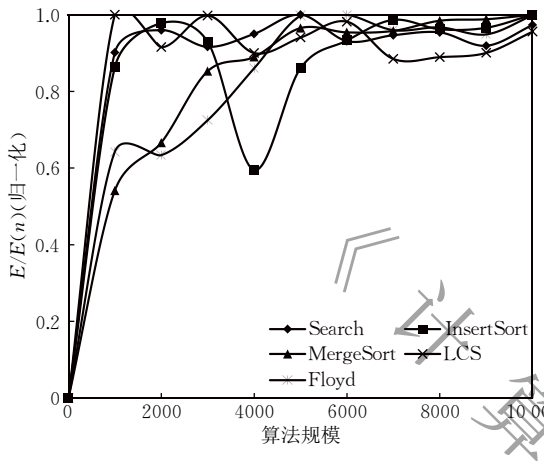


图 4 $E/E(n)$与算法输入规模之间的关系

由图 4 可知，Search、InsertSort、MergeSort、LCS、Floyd 算法的 $E/E(n)$值随输入规模(n)的变化很小，在 10 组输入下的 $E/E(n)$值方差分别为 2.09×10^{-7}、3.33×10^{-9}、7.67×10^{-6}、6.89×10^{-10}、5.88×10^{-15}. 由于对数据进行了归一化处理，$E/E(n)$的值均收敛于 1，当 n 较小时，测量能耗和估计能耗存在差距，而随着算法输入规模的不断增大 $E/E(n)$值趋于收敛. 可以证明本文提出的算法能耗复杂度能够反映能耗随输入规模的变化趋势.

5.4 算法时间/能耗复杂度与交叉度分析

本实验进一步分析了算法交叉度与算法时间复杂度和能耗复杂度之间的关系. 图 5 和图 6 分别给出了 5 种算法的执行时间和能耗(归一化)与算法规模之间的关系. 图中的曲线可以认为是算法的时间复杂度和能耗复杂度. 参考表 4 的算法时间复杂度和能耗复杂度，可以看出算法的时间复杂度和能耗复杂度基本一致，仅当 n 较小时曲线走势会存在差别. 而每个算法的交叉度均不相同.

Search 和 InsertSort 算法的交叉度最小，CPU 一直处于稳定的低使用率低频率和低功率状态；与之对应的是 LCS 算法，其交叉度最高，因此 CPU 一直处于稳定的高使用率高频率和高功率状态；

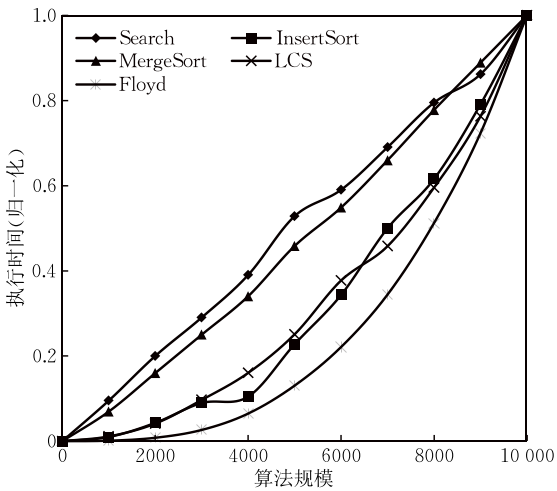


图 5 执行时间与算法输入规模之间的关系

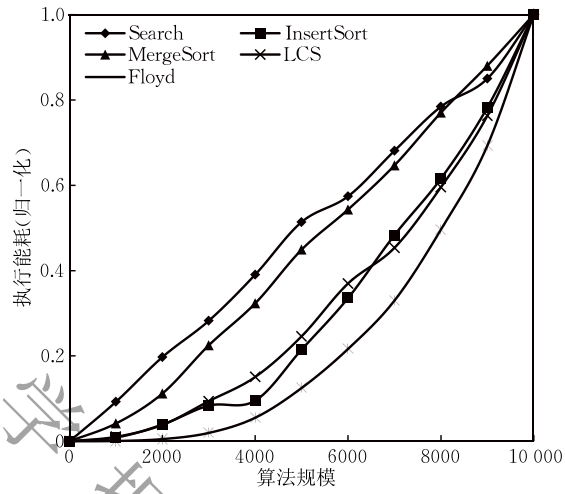


图 6 执行能耗与算法输入规模之间的关系

换而言之，这 3 个算法的 R_0 值很小，当 n 增长时，$R(n)\geq R_0$ 的条件很快满足，因此，能耗复杂度曲线和时间复杂度曲线高度一致. 仔细观察 MergeSort 和 Floyd 算法的时间复杂度曲线和能耗复杂度曲线，当 n 较小时，算法能耗复杂度曲线略低于算法时间复杂度曲线，此时计算机功率低于额定功率，也反映了在 $R(n)<R_0$ 条件下能耗复杂度和时间复杂度的差别.

分析算法的时间复杂度和能耗复杂度，可得如下结论：(1) 与时间复杂度不同，算法能耗复杂度考虑了算法“交叉度”带来的额外能耗，虽然能耗复杂度与时间复杂度的高阶项相同，但当算法规模较小时，两者仍存在差异；(2) 实验采用了较大的算法规模，且每次增加的步长也很大，这样做是为了减少误差，更好地反映时间或能耗随规模增长的趋势，但没有凸显算法规模较小时两者的差异，从时间复杂度和能耗复杂度的数学表达中也可以看出，两者的高

阶项相同，而差别之处在于低阶项，当 n 较小时这种差异更为明显；(3) 尽管在推导过程中使用大量的参数和化简操作，且交叉度的定义存在主观性，这都会影响能耗复杂度的准确性，但实验结果表明本文提出的能耗复杂度是“算法能耗与算法规模之间关系”的理想近似，且有别于算法时间复杂度；(4) 经实验结果推测，仅就交叉度对能耗的影响大小而言，外存算法比内存算法更为明显，因为前者的存储语句更耗时. 不基于 RAM 模型的外存算法的能耗复杂特征有待进一步研究.

6　结论和进一步工作

本文以算法为能耗评价对象，通过分析算法特征，设计算法能耗复杂度理论模型以及推导方法，主要工作有：

(1) 修改和扩展传统图灵机，设计更适合能耗分析的能耗图灵机；分析能耗图灵机算法的两种耗能操作，定义算法能耗复杂度；

(2) 分析计算机算法的空间复杂度、时间复杂度、算法交叉度和算法能耗的关系，设计由时间复杂度和交叉度推导能耗复杂度的方法；

(3) 通过一系列实验证明时间复杂度、算法交叉度和算法能耗的关系，并验证能耗复杂度的正确性.

算法能耗复杂度是算法能耗的评价模型，定义了算法能耗随算法规模的增长趋势. 在实际的软件开发过程中，通过比较多个算法的能耗复杂度，尤其是比较时间复杂度相同的一组算法的能耗复杂度，进而选择能耗复杂度低的算法，降低软件产品的能耗. 本研究还处于起步阶段，对交叉度的准确定义和适用性研究是亟待开展的工作，例如进一步区分“存储语句”和“运算语句”、考虑哪些类算法其“交叉度”对能耗影响更为明显(例如外存算法)，从而精确定义 4.4 中推导过程中的 *idle* 函数. 此外，本文仅考量了算法能耗复杂度的定义，并未涉及其优化方法. 算法能耗复杂度优化方法研究需要结合算法执行过程中已经存在的编译器优化原则以及程序设计方法等诸多因素，这也是下一步研究的主要工作.

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] 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)

[3] Song Jie, Sun Zong-Zhe, Li Tian-Tian, et al. Research advance on code oriented optimization of software energy consumption. Chinese Journal of Computers, 2016, 39(11): 2270-2290(in Chinese)
(宋杰，孙宗哲，李甜甜等. 面向代码的软件能耗优化研究进展. 计算机学报，2016，39(11)：2270-2290)

[4] Goldreich O. Computational complexity: A conceptual perspective. ACM SIGACT News, 2008, 39(3): 35-39

[5] Martin A J. Towards an energy complexity of computation. Information Processing Letters, 2001, 77(2-4): 181-187

[6] Karamchandani N, Appuswamy R, Franceschetti M. Time and energy complexity of function computation over networks. IEEE Transactions on Information Theory, 2011, 57(12): 7671-7684

[7] Höpfner H, Bunse C. Towards an energy-consumption based complexity classification for resource substitution strategies //Proceedings of the 22nd Workshop on Foundations of Databases. Bad Helmstedt, Germany, 2010: session7.1

[8] Korthikanti V A, Agha G, Greenstreet M. On the energy complexity of parallel algorithms//Proceedings of the 2011 40th International Conference on Parallel Processing Workshops. Taipei, China, 2011: 562-570

[9] Roy S, Rudra A, Verma A. An energy complexity model for algorithms. Dissertations & Theses-Gradworks, 2013: 283-304

[10] Jain R, Molnar D, Ramzan Z. Towards a model of energy complexity for algorithms//Proceedings of the 2005 IEEE Wireless Communications and Networking Conference. Los Angeles, USA, 2005: 1884-1890

[11] Brandolese C. Source-level estimation of energy consumption and execution time of embedded software//Proceedings of the 12th Euromicro Conference on Digital System Design: Architectures, Methods and Tools. Parma, Italy, 2008: 115-123

[12] Schubert S, Kostic D, Zwaenepoel W, et al. Profiling software for energy consumption//Proceedings of the IEEE International Conference on Green Computing and Communication. Besancon, France, 2012: 515-522

[13] 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

[14] Bunse C, Höpfner H, Roychoudhury S, et al. Energy efficient data sorting using standard sorting algorithms//Cordeiro J, Ranchordas A K, Shishkov B eds. Software and Data Technologies. Berlin Heidelberg, Germany: Springer, 2009: 247-260

[15] Berthier N, Maraninchi F, Mounier L. Synchronous programming of device drivers for global resource control in embedded operating systems. ACM Transactions on Embedded Computing Systems, 2013, 12(1s): 81-90

[16] Lv M, Yi W, Guan N, et al. Combining abstract interpretation with model checking for timing analysis of multicore software// Proceedings of the 2010 IEEE 31st Real-Time Systems Symposium. California, USA, 2010: 339-349

[17] Zhu Y. Modeling energy-aware embedded software using process algebra//Shen G, Huang X eds. Advanced Research on Computer Science and Information Engineering. Berlin Heidelberg, Germany: Springer, 2011: 389-394

[18] Zhu Y Z Y, Huang Z H Z, Zhang G Z G. Modeling and analysis of real-time software based on resource communicating sequential process. Advanced Materials Research, 2011, 225-226: 802-806

[19] Callou G, Maciel P, Tavares E, et al. Energy consumption and execution time estimation of embedded system applications. Microprocessors & Microsystems, 2011, 35(4): 426-440

[20] Perez-Palacin D, Mirandola R, Merseguer J. QoS and energy management with Petri nets: A self-adaptive framework. Journal of Systems & Software, 2012, 85(12): 2796-2811.

[21] Deng Wei, Liu Fang-Ming, Jin Hai, et al. Leveraging renewable energy in cloud computing datacenters: State of the art and future research. Chinese Journal of Computers, 2013, 36(3): 582-598(in Chinese)
(邓维, 刘方明, 金海等. 云计算数据中心的新能源应用: 研究现状与趋势. 计算机学报, 2013, 36(3): 582-598)

[22] Tang G, Jiang W, Xu Z, et al. Zero-cost, fine-grained power monitoring of datacenters using non-intrusive power disaggregation//Proceedings of the 2015 ACM/IFIP/USENIX MIDDLEWARE Conference. British Columbia, Canada, 2015: 271-282

[23] Zhou Z, Liu F, Jin H, et al. On arbitrating the power-performance tradeoff in SaaS clouds//Proceedings of the 32nd IEEE International Conference on Computer Communications. Turin, Italy, 2013: 872-880

[24] Zhou Z, Liu F, Li Z, et al. When smart grid meets geo-distributed cloud: An auction approach to datacenter demand response//Proceedings of the 2015 IEEE International Conference on Communications. London, UK, 2015: 2650-2658

[25] Elnozahy E N, Kistler M, Rajamony R. Energy-Efficient Server Clusters: Power-Aware Computer Systems. Berlin Heidelberg Germany: Springer, 2003

[26] 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)

[27] Chen Q, Grosso P, Veldt K V D, et al. Profiling energy consumption of VMs for green cloud computing//Proceedings of the IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing. Sydney, Australia, 2011: 768-775

[28] Weiss M A. Data Structures and Algorithm Analysis in Java. Boston, USA: Pearson Addison-Wesley, 2007

附录 A. 能耗图灵机的适用性描述.

单带图灵机很难处理复杂变化的计算问题,因此在经典单带图灵机的研究基础上还提出图灵机的多种变形,使得经典图灵机可以执行多样的算法.

定义 3 是一个单带确定性能耗图灵机.本附录将对照经典图灵机变形,定义其他几种类型的能耗图灵机.面对特定算法时可选择更合适的能耗图灵机类型,分析算法能耗复杂度,以证明本文提出的能耗图灵机的适用性.本附录简要证明不同类型能耗图灵机求解算法能耗复杂度过程的同理性,即证明不同类型能耗图灵机的等价性.

多带能耗图灵机与单带能耗图灵机的不同点主要体现为前者具有多条符号带,每条符号带有各自的读写头.开始时,输入出现在第一条符号带上,其他符号带都是空白的.转移函数改为允许在不同带上进行读写和移动读写头操作.k 带能耗图灵机 $M=(Q,\Sigma,\Gamma,\delta,q_0,B,F,\varepsilon)$,其中函数 δ 定义为

$$\delta: Q\times\Gamma^k\times Q\times(\Gamma\times\{R,L\})^k\to E \tag{12}$$

可以使用和经典图灵机中证明单带图灵机等价于多带图灵机类似的证明方法:用单带能耗图灵机的有限步移动来模拟多带能耗图灵机的一步移动,证明对任何 k 带能耗图灵机 M 都存在一个单带能耗图灵机 M_1 与之等价.

非确定能耗图灵机与确定能耗图灵机的区别仅在于其移动规则不能被转移函数唯一地确定,因为其转移函数具有多值性的特点,也可以将一个转移函数的多个函数值看作一个子集,即将非确定能耗图灵机的转移函数当作一个集值函数.非确定能耗图灵机 $M=(Q,\Sigma,\Gamma,\delta,q_0,B,F,\varepsilon)$,其中函数 δ 定义为

$$\delta: Q\times\Gamma\times 2^{Q\times\Gamma}\times\{R,L\}\to E \tag{13}$$

证明非确定能耗图灵机与确定能耗图灵机等价,则只需要证明:确定图灵机与非确定图灵机接受同一语言.

定理 1. 非确定能耗图灵机与确定能耗图灵机等价.

证明. 设 $M=(Q,\Sigma,\Gamma,\delta,q_0,B,F,\varepsilon)$ 是一个非确定能耗图灵机,将按照如下的方法构造与之等价的确定能耗图灵机 M_1.

$$\delta: Q\times\Gamma\times 2^{Q\times\Gamma}\times\{R,L\}\to E.$$

取 $m=\max\{|\delta(q,x)|\ |\ (q,x)\in Q\times\Gamma\}$,可得

$$\forall(q,x)\in Q\times\Gamma,$$

$$\delta(q,x)=\{(p_1,y_1,d_1),(p_2,y_2,d_2),\cdots,(p_n,y_n,d_n)\}.$$

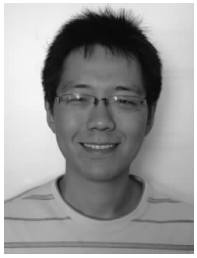
使用 $1,2,\cdots,n$ 分别代表选择 $(p_1,y_1,d_1),(p_2,y_2,d_2),\cdots,$

(p_n, y_n, d_n). 对任意输入的字符串，则 M_1 用来处理它的操作序列都要使用 $\{1,2,\cdots,m\}$ 上的数字序列表示. 由于 $m \geq n$，因此其中存在无效的操作序列.

令等价的 M_1 具有 3 条符号带，第 1 条带用来存放输入，第 2 条用来存放生成的用于处理输入串的操作序列，由于第 2 条带上存放的操作序列是系统生成的，因此依照该序列对输入串进行处理并一定能够保证成功. 为了解决此问题，M_1 的第 3 条带作为草稿纸，对系统生成的每一个操作序列，M_1 将输入串抄写到它的第 3 条带上，然后按照第 2 条带上的操作序列对第 3 条带上的内容进行处理，若 M_1 并没有进入终止状态，则表示当前这个操作序列无效. 此时，M_1 重新在其第 2 条带上系统生成下一条操作序列，并将第 1 条带上的内容复制到第 3 条带，并进入下一次循环.

如果能导致 M 接受输入串的操作序列，M_1 最终将会接受输入串；否则，M_1 将不会接受此输入串.　　证毕.

另外，在实际应用能耗图灵机时，图灵机输入的内容常常不仅只有数据，还应该有用于处理数据的程序(算法). 前面讨论的图灵机都等同于一个已经安装固定程序且此程序不可被改变的计算机(这里的程序指的是转移函数)，因此需要一个可以模拟其他任意类型的能耗图灵机的计算的图灵机，即通用能耗图灵机. 而对于通用能耗图灵机，与经典通用图灵机相比，本研究并未改变其最为关键的编码规则，即通用能耗图灵机继续研究其原有编码规则，而其他内容则与本文定义的单带能耗图灵机、多带能耗图灵机等类似.



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

MA Zhong-Yi, born in 1994, M. S. candidate. His current research focuses on energy-efficient computing.

XU Shu, born in 1991, M. S. candidate. His current research focuses on energy-efficient computing.

BAO Yu-Bin, born in 1968, Ph. D., professor. His research focuses on big data storage and management.

YU Ge, born in 1962, Ph. D., professor, Ph. D. supervisor. His research interests focus on database theory.

Background

Aiming at solving the energy issue in IT industry, software energy efficiency optimization is gradually focused by researchers in recent years. The researches on software energy efficiency optimization are done via reducing energy consumption of a given task, such as the optimization on software structure, code, instruction sequence or algorithm. Algorithms are the description of solution to the problem, while code is the implementation of algorithm. The study on algorithms in energy consumption is more abstract than study on code, so that the generality of the result is higher. The study on energy consumption of algorithms is valuable, although researches on energy consumption of algorithms are few at present. Starting from an improved Turing machine for energy consumption analyzing, a model of energy consumption complexity is proposed in this paper comparing with time complexity and space complexity of algorithms.

Recently, there have been few studies on energy consumption complexity. Some researchers proposed their energy consumption complexity model in some particular environment, such as a model for multi-core parallel algorithms; And some research also use Turing machine in energy consumption complexity study and deduce the relationship between energy consumption complexity and time and IO complexity. In this paper, researchers start with improved 8-tuple Turing machine for energy consumption, and define the energy consumption complexity combining with state shift energy consumption and inertia energy consumption. The feasibility of our Turing machine is also discussed in the appendix of this paper. With the theory base of Turing machine, we define the cross factor, which exists in real computers theoretically and shows consistency with the DVFS of CPU. Combining the cross factor with space complexity and time complexity, we deduce the mathematical equation of energy consumption complexity—the relationship between input size and energy consumption. In the final part of this paper, experiments analysis of energy consumption of several algorithms is discussed. We verified the existence of cross factor, and prove the correctness of our energy consumption complexity model.

This work is mainly supported by the National Natural Science Foundation of China (Nos. 61433008, 61672143, 61662057, 61502090, 61402090). Our group has worked in the area of green computing for many years and published many papers.