大 连 理 工 大 学 本 科 外 文 翻 译

使用 concolic 测试自动分析分支覆盖率和能耗

An Automated Analysis of the Branch Coverage and Energy Consumption Using Concolic Testing

学 部（院）：软件学院

专 业：软件工程

学 生 姓 名：唐智强

学 号：201192191

指 导 教 师：侯 刚

完 成 日 期：2020 年 2 月 28 日

大连理工大学

Dalian University of Technology

使用 concolic 测试自动分析分支覆盖率和能耗

Sangharatna Godboley[1];Subhrakanta Panda[1];Arpita Dutta[1];
Durga Prasad Mohapatra[1];
法赫德国王石油与矿业大学；minerals 期刊

摘要：计算机系统的能耗已成为一个重要的经济和环境问题。许多研究人员关注的是硬件的能耗，那么软件呢?软件能耗被广泛应用于科研实验室的绿色计算实践。但是，目前的研究人员未能为关键应用的软件能耗建立一致的概念基础。虽然分支覆盖和 concolic 测试是验证安全关键系统的非常关键的实践，但却很少对它们的能量消耗进行测量。这些技术的能耗计算是绿色 IT 和绿色软件工程中的一个重要问题。本文的贡献在于自动计算和分析测试技术的能耗，同时利用共曲线测试提高分支覆盖率。我们在一个工具中实现了我们提出的自动化框架，这个工具叫做分支覆盖增强的绿色分析。对 40 个 Java 程序的实证研究和评价结果表明，我们开发的工具实现了分行覆盖率的平均增长 13.5%。我们的自动化工具计算所有 40 个实验程序的分支覆盖率的平均能耗大约是 5.6 kJ。

关键词： 分支覆盖、测试、能耗、绿色 IT、绿色软件工程、绿色和可持续技术

1 介绍

ICT - literacy在2007年的一份技术报告中估计，信息和通信技术(ICT)占全球碳排放的2%以上。由1E在2007年赞助的《美国个人电脑能源报告》得出结论，由10,000台个人电脑组成的公司每年大约花费165,000美元用于电力供应。这一巨额账单是由于即使在没有人工作的情况下，电脑也整夜不关机而产生的。因此，个人电脑的能耗是一个需要解决的重要问题。

绿色信息技术(Green Information Technology, Green IT)认为ICT的资源和能源消耗从软件开发生命周期(SDLC)阶段开始，应该减少和最小化[2,3]。现有的方法主要集中在硬件方面，如数据中心的能耗。由于大多数ICT解决方案都是基于软件应用，因此对这些应用的能耗进行分析和控制是非常关键的。

软件测试的目的是检测和纠正软件中的一些bug，以确保软件的质量。在此之前，软件测试技术一直未能检测到软件中出现的所有错误。失败的原因之一是无法执行程序中的所有执行路径。由于存在条件表达式和许多其他循环结构，程序可能具有多种不同的路径。从技术上讲，生成能够覆盖所有执行路径的测试用例是一项困难的任务。为了解决这些问题，在文献中提出CONCOLIC (CONCrete + symbOLIC)测试或动态符号执行分析，以生成探究程序执行路径的

测试用例[5-7]。手工实现这种高覆盖率的过程是困难且不可行的。根据RTCA/DO178B标准[8]，基于覆盖的测试是在执行逻辑语句的上下文中对基于需求的测试的可接受性的度量。这就是为什么基于分支覆盖的测试已经成为航空航天和核应用等安全关键系统的强制性测试。尽管CONCOLIC和分支覆盖测试技术在许多场景中都很有用，但是它们的能耗问题还没有得到解决。而这正是绿色IT和绿色软件工程[9]关注的核心领域

在这篇论文中，我们提出了一个自动计算分支覆盖率并分析其能耗的框架来促进绿色软件测试。在it方面，我们在一个工具中实现了我们的框架，这个工具叫做绿色分支覆盖增强分析(GreenABCE)，它集成了一些用于不同目的的可用工具。我们提出的框架集成了Java程序代码转换器(JPCT)与Java Concolic Unit testing Engine (JCUTE)， JPCT是我们在早期工作[10]中提出的，它是程序代码转换器(PCT)[11]的Java版本。JPCT转换输入Java程序以改进它们的分支覆盖率，JCUTE可生成所需的测试用例，并计算Java程序的转换版本和非转换版本的分支覆盖率。请注意，尽管转换后的程序在语法上与原始程序不同，但由于插入的是不影响逻辑路径的额外表达式，两者在语义和功能上是等价的。为了避免混淆，我们在实验完成后从转换后的程序中去掉那些额外嵌套的if-else语句。利用焦耳计计算了总能耗。我们在表1中列出了本文中使用的缩写。

TABLE 1 缩略字

本文的其余部分内容如下:第2节讨论了理解所提议的方法所需的一些基本思想。第3节解释了提出的框架并讨论了实现所需的算法。在第四部分，我们分析了实验结果。在第5节中，我们讨论并比较了与我们提议的工作相关的一些现有工作。我们在第6节中报告了我们工作的有效性受到的一些威胁。第7节总结了我们的工作，并对未来的工作提出了一些见解。

2 基本思想

在本节中，我们将讨论一些对我们工作所必须的重要概念和定义。

Fig. 1 greensoft 模型

2.1 绿色软件工程

我们从 Kern 等人的[9]项目中获得了推动节能绿色软件测试的动力。为了对绿色可持续软件及其工程技术的一些关注点进行分类整理,Kern 等人[9]开发了 GREENSOFT 模型，如图 1 所示。GREENSOFT 模型由以下四个部分组成:软件产品的生命周期、软件产品的可持续性标准、过程模型、行动和工具的建议。有

兴趣的读者可以参考[9]了解更多关于绿色软件工程(Green Software Engineering, Green SE)的细节。

因此，在符合绿色SE原则的基础上，我们的目标是开发一个工具，可以计算Concolic测试期间的能源消耗。对能源消耗的认识可以使测试人员在以后的测试中提出新的节能的技术，朝着绿色软件测试的方向发展。

2.2 能源消耗

本章节利用现有的joulemeter1来计算能量消耗。JouleMeter是用来计算所提出的工具在程序测试过程中的能量消耗(EC)的。JouleMeter给出了在记录时间戳的任何特定实例中正在使用的工具的功耗(PC)。我们使用这些功率读数和第一次使用的时间戳来计算焦耳的能量消耗。将功率读数转换为能耗的公式如下:

$$\mathrm{EC} = (T_m - T_n) \times \sum_{n}^{m} (PC_n) \qquad (1)$$

其中EC为能耗(焦耳)，PC为功耗(瓦特)，Ti为第i个瞬间的时间戳，n为第一个实例，m为最后一个实例。

2.3 分支覆盖

为了实现分支覆盖，每一个决策应该至少采取一次所有可能的结果，要么是对的，要么是错的。例如:If (m > 0)，那么最大可能有两个测试场景。这些测试场景是:(1)m > 0，和(2)m <= 0。因此，任何生成足够的测试用例来执行上述两个场景的测试技术都可以实现100%的分支覆盖率。

Fig. 2 一个例子程序来解释CONCOLIC测试

2.4 concolic测试

Concolic测试是一种跨界的程序验证方法，它可以巩固符号执行。Concolic测试执行所有典型的程序变量，以便在某些特定的输入上更具体地运行程序。在concolic测试中，我们引入了一些标准变量(带有随机值)来生成新的路径条件以及常规的执行路径(当条件满足true时执行)来探索可选的分支谓词。这些新生成的路径将初始谓词与其否定谓词连接起来。

图2中的程序检查学生取得的分数，并据此进行评分。对于这个程序的一致性测试，我们首先将随机输入分配给名为testscore的变量。程序的执行取决于testscore变量的赋值。假设testscore的第一个值是91，那么要分析的第一个谓词是：

(testscore >= 90) (2)

现在，条件2一定不能探索另一条可能到达下一个谓词的路径。通过否定考虑中的谓词来发现新的路径。这个否定操作如下图所示:

¬(testscore >= 90)　　(3)

否定操作后得到的新谓词如下:

(testscore >= 80)　　(4)

约束求解器检查每个约束，以发现程序中的适当路径。如果存在输入的路径，则将该值记录为测试数据，以便进一步执行程序。将原约束与反约束相结合，得到下一个新的路径约束，如式(5)所示。

¬(testscore >= 90) ∧ (testscore > 80)　(5)

上述过程将继续进行，直到条件得到评估。

3 提出框架

基于覆盖率的测试和能耗分析是许多安全关键系统所必需的。由于无法达成较高的分支覆盖率，许多软件测试人员不得不放弃某些软件。所以我们构思的框架旨在研究利用现有技术增加分支覆盖率的问题，但是这种分支覆盖率的提高却往往伴随着额外的能源消耗。

为了测量在增加分支覆盖率时所消耗的能量，一个能量计算工具，即，JouleMeter 与分支覆盖分析器一起集成到提议的框架中。因此，我们的框架是在一个名为 Green-ABCE 的工具中实现的。该工具的功能是提高共曲线测试中的分支覆盖率，并计算能量消耗。同时开发了 JPCT 以实现分支覆盖率的增加，并将其集成到 Green-ABCE 中，使用 JCUTE 作为动态符号执行工具，解决了诸如库代码不可用和 concolic 引擎缺陷等问题。

Fig. 3 Green-ABCE 的整体视图

3.1 Green-ABCE 概述

图3显示了Green-ABCE的总体视图。框图主要由三个模块组成:JPCT、JCUTE和JouleMeter。JouleMeter开始对JPCT和JCUTE的每1秒功耗进行跟踪记录。JPCT将Java程序作为输入，并通过添加额外的嵌套空if-else表达式对其进行转换。然后将程序的转换版本提供给JCUTE，以自动生成测试用例[12]并计算分支覆盖率的百分比。完成此流程后，我们将停止对功耗的跟踪。最后，我们得到了绿色abce所花费的总时间，来找出jpct和jcutemodels的总覆盖率、总功率和总能耗。

Table 2　算法 1：Green-ABCE

3.2 Green-ABCE 详细描述

在本节中，我们将详细讨论Green-ABCE的不同组件如何工作。

3.2.1 Green-ABCE

算法表2给出了拟工作的总体伪代码。提出的算法表2以ABCE模块为输入，由一个Java程序、转换Java程序的JPCT和生成所需测试用例并计算分支覆盖率的JCUTE concolic tester组成。这里，JPCT表示Java程序代码转换器，而JCUTE表示Java Concolic单元测试引擎。关于JPCT和JCUTE的更多描述分别在3.2.2和3.2.3节中给出。最后，算法表2给出了输入程序的总能量消耗(EC)作为输出。前两个步骤由JouleMeter执行，以开始时间戳记录。算法的第三步计算所需的其他参数(即，生成测试用例，并计算其相应的分支覆盖率)。第4步调用名为JPCT的代码转换器，它转换输入的Java程序。代码转换器的伪代码如算法表3所示。现在，步骤5为转换后的Java程序找到测试用例和BC%(使用JCUTE+JPCT的分支覆盖率)。步骤6计算所有需要的参数，即，分支覆盖的计算时间，测试用例生成的速度和差异。步骤7和8用于停止JouleMeter功能。步骤9根据时间戳差异计算总功耗。最后，步骤10根据公式1计算能耗。

3.2.2 JPCT

Java程序代码转换器(Java Program Code Transformer, JPCT)[10,11]的功能是通过插入额外的if-else语句来转换输入的Java程序。变压器的伪代码如算法表3所示。转换程序的步骤如下:

1.这一步的目标(第1-3行)是检测Java程序中的谓词。逻辑是逐字符扫描Java程序，并将具有布尔运算符的代码行复制到其他文本文件中。重复此过程，直到发现程序中的所有谓词。

2.此步骤的目标(第5-6行)是通过计算已识别谓词的乘积和(SOP)来简化其表示形式。对程序中发现的每个谓词都执行这种简化。谓词的SOP被转换为minterm(二进制)形式，以便使用奎因-麦克卢斯基方法最小化它。这些minterm存储在变量P_Minterm中。

3.第7行使用quinn - mccluskey方法最小化SOP。奎因-麦克卢斯基方法在[10,13]中给出。选择Quine - McCluskey方法而不是另一种现有技术(称为Karnaugh map)来最小化谓词的原因很简单，因为前者比后者更有效。

4.插入嵌套的空 if-else 语句第 8 行生成空嵌套的 if-else 语句，以获得程序中的附加语句集。在[10]中定义了生成这些附加语句的方法。这些额外的语句集由空嵌套的 if-else 语句组成，这些语句被插入到输入程序的第 9 行，从而得到原始 Java 程序的转换版本。这些附加的条件表达式对于增加生成的测

试用例的数量很有用。一旦我们增加了测试用例的数量，我们就可以覆盖更多的分支。这帮助我们获得了高的分支覆盖率。

TABLE 3：算法二:JPCT 的 java 代码转换器

3.2.3 JCUTE

我们使用JCUTE为Java程序的原始和转换版本自动生成测试用例。算法表3的输出现在是JCUTE的输入。然后JCUTE计算分支覆盖率。我们还记录JCUTE计算分支覆盖率所花费的总时间。更多关于JCUTE工具的信息可以在[5,6]中找到。

3.2.4 JouleMeter

JolueMeter 是一种测量计算机系统功耗的工具。可以计算整个系统或关键部件的功耗数据。读数可能取决于不同应用程序的个别跟踪。JouleMeter 通过一个称为 power model 的模型来计算电力使用情况。该模型显示了计算机资源的使用情况和硬件电源状态(处理器利用率、屏幕亮度、进程频率、监视器开/关状态、内存利用率)来绘制电源值。但是在我们提出的工作中，我们没有考虑其他硬件的功耗状态，我们只是跟踪特定软件应用程序的功耗。在我们的研究工作中，软件应用程序指的是我们提出的软件测试工具。为了学习功率模型，可以输入单个功率数的手动估计来查看功率使用数据。JouleMeter 为功率模型提出了一些功率数参数。这些功率数参数是:基准(空闲)功率、处理器峰值功率(高频)、处理器峰值功率(低频)和监视器功率。请注意，应用程序的功耗仅表示应用程序在 CPU 上消耗的功耗。在这里，我们可以考虑监视器电源以及计算机的基本电源，只要计算机已经打开，只运行该应用程序。能量消耗可以定义为整个运行过程中使用的总能量，可以通过计算执行期间的能量消耗值(当前的瓦特数)来得出。由于一秒钟的电功率是恒定的，所以这个和代表了以焦耳为单位的总能量消耗。整个过程以伪代码的形式表示，如算法表 2 所示。

4 实验研究

我们在一台个人电脑(PC)配置 4GB 内存(RAM)， Intel (R) Core (TM) i5 CPU 650@3.20 GHz 3.19 GHz, 32 位操作系统，Windows 7 操作系统上进行了实验。使用我们提出的工作方案，我们可以处理任何语言，如 SQL 和 c++。由于 JouleMeter 可测量和记录任何应用程序的功耗，我们可以跟踪任何语言开发的任何工具的功耗。此外，能耗计算器也是一个开发的工具，它可以执行应用程序/工具，而不受任何语言的限制。我们的目标是计算我们提出的改进 concolic 曲线测试的能耗。该测试技术使用了 JPCT 和 JCUTE;因此，我们使用 Java 语言。

实验在 40 个基准 Java 程序上进行，这些程序取自开放系统实验室储存库 2 和一些学生作业。

表4列出了我们在实验研究中考虑的Java程序的特征。从表4的第3和第4列可以得出结论，原来的程序总共分析了6064行代码。类似地，对转换后的程序分析了9601行代码。代码行(LOC)给出了原始程序的大小，而LOC`表示程序的转换版本的大小。由于我们主要关注基于覆盖的测试，所以计算应该覆盖的谓词和分支的总数是很重要的。这两个特征分别在第7栏和第8栏中提到。

TABLE 4：不同实验方案的特点

表5和6显示了实验Java程序所需的测试数据[例如生成测试用例的数量(TC,TC`)、计算时间(时间，时间`)和生成测试用例的速度(速度，速度`)]。这些测试数据在两个场景下计算:场景1(表5)对应于使用JCUTE为原始程序生成的测试数据，而场景2(表6)对应于使用JCUTE + JPCT为转换程序计算的测试数据。从表中可以推断，通过集成JPCT，我们增加了测试用例的生成,这有助于覆盖更多的分支，从而获得较高的分支覆盖率。在这两种情况下所花费的计算时间(生成测试用例和计算分支覆盖率)显示为时间和时间`。在场景1中，40个程序的平均计算时间为23,678.93 ms，而在场景2中，40个程序的平均计算时间为27,176.42 ms。因此，测试用例生成的速度被指定为每秒生成测试用例的数量(TC/s)。场景1计算的平均速度是speed = 4.66 TC/s,场景2的平均速度是speed`= 2.27 TC/s为40个程序。测试用例生成速度的降低是由于程序转换时间的增加。但是，速度的降低并不是一个决定性的因素，因为它导致了生成测试用例和分支覆盖率的数量的增加。

分支覆盖率的分析如表 7 所示。这里，BC%对应于情形 1 和 BC`%对应于场景 2。BC 的值范围从 0 到 100%。请注意，有些程序(如 CTest1、ErrorTest、FinallyTest、OneWrite 和 StackTest)的 BC%为 0%;这是因为生成的测试用例没有覆盖这些程序的任何分支。这清楚地表明了场景 1 中生成的测试用例的低效性。但是，经过项目转换后，我们能够成功地提高上述项目的分支覆盖率。例如,对于程序 OneWrite(参见第 4 列)，它大约从 0 提高到了 66%。请注意，BC%值仅为 11 个程序的 100%，而 BC?%是十五个程序的 100%。这意味着所有可能的分支都被覆盖了。表 7 最后一栏给出了这两个分支覆盖率的差异。

Table 5 使用 JCUTE 为原始 Java 程序生成测试数据

Table 6 使用(JPCT + JCUTE)为转换后的 Java 程序生成测试数据

可以看到，一些程序的分支覆盖率没有变化。这些程序的分支覆盖率没有增强的可能原因可能是，要么所有可能的分支都被覆盖了，要么两个场景中覆

盖的分支数量相同。其余的程序成功地提高了分支覆盖率。平均BC%是66.5%，BC?%是40个程序的80%。在这里，我们得出结论，对于实验项目来说，分支覆盖率的平均增长率大约为13.5%。图4显示了这两个场景的分支覆盖率。x轴表示程序编号，Y轴表示BC%。

为了关注功耗，表8和表9分别显示了场景1和场景2的能源消耗详细信息。表9的第3至5列显示了记录的时间戳。第6列显示了我们的工具在计算每个实验程序的测试数据时的功耗。图5显示了不同实验方案相对于时间的功耗(见表9第5列)。表9显示，我们的工具的功耗为7.5%的实验程序超过100 W。当87.5%的程序功耗低于50w时，只有5%的程序功耗在50w到100w之间。我们给出了与转换程序的代码行数(LOC)、谓词数量和总功耗时间相关的功耗箱形图。图6的结果表明，功耗并没有随着LOC的增加而明显增加(即由于代码转换。因此，通过JPCT进行代码转换来增强分支覆盖率不会消耗太多的电能。因此，JPCT是高效节能的，而程序中谓词数量的小幅增加则会导致功耗的急剧上升(参见图7)。因此，程序中谓词数量越多，功耗越大。当工具花费一些时间来覆盖更多的谓词时，功耗就会增加。从图9可以明显看出，功耗随分析程序所用时间的增加而增加。

使用公式1计算出的所有40个程序的能耗如表8和表9的最后一列所示。能量消耗范围从0.1 kJ到70 kJ不等。

图 8 显示了 x 轴上的程序和 Y 轴上相应的能耗。结果表明，25%的程序能耗均在 1kj 以上。无编码变压器工具的平均能耗为 5.2 kJ，有编码变压器工具的平均能耗为 5.6 kJ。因此，代码转换带来的能耗增加是 0.4 kJ。相对于计算时间的增加，能源消耗的增加(表 5、6)如图 10 所示。根据绿色软件工程的原则，实现分支覆盖率增长 13.5%所需的能源开销仅为 0.4 kJ。因此，测试人员可以选择代码转换，同时仍然支持绿色软件测试。

Table 7 计算分支覆盖率

5 与相关工作比较

Fig. 4 分支覆盖率比较

在这一节中，我们将讨论和比较一些与我们提议的工作相关的现有工作。

Li等人提出了一个框架，称为能量定向测试套件优化(EDTSO)。他们通过考虑他们的能源效率来最小化测试用例。他们提出了一种新的测试套件最小化技术，以降低测试套件的能耗。他们的方法基于将能量和测试覆盖率需求转换为ILP问题。他们使用两个应用程序来进行实验，即， K-9邮件和MyTracks。对于K-9邮件，最昂贵的测试用例消耗了近5000兆焦耳，第二个测试用例消耗了近

3000兆焦耳，其余的用例从大约1000兆焦耳到接近5兆焦耳。同样，MyTracks测试用例的能耗范围为≈5m J 到≈400,000 mJ。从这个结果中，他们得出结论，来自测试套件的测试用例可能在它们的能量消耗中有很大的差异。与[2]中的方法不同，我们提出的方法是计算共曲线测试工具的能耗，而不是计算单个测试用例的能效，目前该工具还不支持这种方法。

Amsel等人[1]提出了GreenTracker来估计任何软件的能耗。他们认为这个工具的开发对于确定哪种软件系统在环境上是最可持续的至关重要。它们还旨在传播与软件相关的潜在环境危害的认识，并改进软件工程技术以减少软件的能源消耗。在结果和评估方面，他们已经在Mozilla Firefox 3.0.10和Safari 4.0.3网络浏览器上进行了试验。他们发现，在他们的系统上，Firefox的平均CPU功耗比Safari低。Firefox在5次测试中的平均CPU为28.63%，而Safari为30.98%。GreenTracker不适合用于实际测试功能，而我们提出的工具侧重于增强分支覆盖率以进行有效的concolic测试，并显示了在此过程中所消耗的能量。

Dick 等人提出了一种方法来测量和评估桌面计算机上独立应用程序和服务器上基于交互事务的应用程序的软件诱导能耗。在这两种情况下，都应用了实际的工作负载。他们的测试平台包括测试系统、适当的功率计(PM)、工作负载生成器(WG)和数据评估系统(DES)。他们的方法旨在支持软件开发人员、购买者、管理员和用户对软件架构和实现以及他们使用或计划使用的软件产品做出明智的决策。在他们的实验中，他们考虑了 Firefox 和 Internet Explorer 浏览器。差异具有统计学意义 t(30.686) =-10.981,p < 0.01(未假设的等方差)。在这种情况下，Firefox 与 Internet Explorer 的平均节能约为 19%，而我们提出的方法是计算软件测试工具的功耗和能耗。

Chen 等人开发了一个名为 StreesCloud 的工具来测量应用程序的能耗。他们观察到部署配置为“3Small(S)”的能耗比“3Small(D)”增加了 0.8%。“3Small(S)”的系统吞吐量较“3Small(D)”下降 2.1%。这是因为他们在测试中模拟的客户工作负载中，大多数任务都是通信密集型的。他们的实验结果表明，在应力云的支持下，可以有效地收集和分析现实云环境中云应用程序的性能和能耗。与度量云基础应用程序性能的 StreesCloud 工具不同，我们的工具度量简单的独立 Java 应用程序的能耗。

Table 8 使用 JCUTE 提出的工具的功耗和能耗

Capra等人提出了一种测量应用软件能效的方法，主要关注管理信息系统(MIS)。他们的方法是定义应用软件能效的第一步，并为开发人员提供设计节能代码的指导方针。他们还开发了一个Java工具，称为工作负载模拟器(workload

simulator, WLS)。对于给定的应用程序，他们的工具可以模拟操作流，并为给定数量的同时用户执行一定数量的操作，从而生成基准工作负载。他们使用电表夹(AC)和一个他们称之为虚拟仪器机(VIM)的系统来测量功耗。他们的实验表明，应用软件对能源消耗有着不可忽视的影响。运行应用程序的服务器消耗的电能比空闲服务器消耗的电能高出72%。此外，不同的系统需要不同的能量来完成相同的操作。例如，OpenBravo需要17.5 W h，而Adempiere在Windows上只需要7.1 W h，差距大于100%。这些结果在应用程序类别和操作系统之间是一致的，平均差距为79%。在我们提出的工作中，我们还使用Java语言开发了我们的工具。我们的工具能够执行Java程序的测试任务。我们提出的工作需要消耗总功率和能量。

Brown 等人在他们的文章中讨论了计算系统中能源效率的整体方法。他们提出了在系统软件中实现能源优化机制，为系统硬件配备了电源模型，并根据应用程序建议进行资源供应调整，以实现所需的吞吐量水平和/或完成期限。例如，当前的 4 个 socket 服务器系统(基于 Sun Niagara2 的 8 核 CPU)使用 DDR2 双通道内存技术，每个 socket 有 16 个 DIMMs，总共有 64 个 DIMMs。如果它的更快的后继版本使用 edddr3 三通道内存，这将增加到每个插槽 24 个调光(总数为 96 个)。代表 DDR2 DIMM 消耗 1.65 w(或 3.3 wperpair)，其中当前 DDR3 DIMMs 的最低功率版本消耗 1.3W(或 3.9W / trio)。其结果似乎只是增加了 20%的电力消耗从约 100-120W 的总在他们的例子。在我们提出的工作中，我们关注的是软件应用程序的计算能力和能耗，而不是硬件组件。

Saxe 等人提出并开发了 PowerTop 工具。PowerTop 显示了浪费的程度，同时还显示了是哪个软件造成的。然后，系统用户可以将观察到的浪费报告为 bug 和/或选择运行更高效的软件。作为一个电力使用的例子，他们估计使用 OpenSolaris PowerTop 1.1 版本充电 3 小时需要 13.616 W。他们表示，PowerTop 为程序员和管理员提供了观察软件效率低下的能力，为优化提供了一个参考点，并使他们认识到软件在节能计算中扮演的重要角色，这是一个伟大的第一步。通过我们提出的工作，我们计算了不同软件测试工具的能耗。无论哪种工具消耗的能量更少，都更适合用于进一步的任务。

Table 9 JPCT+JCUTE 的功耗和能耗

Sen等人[5,12]分别为C和Java程序开发了concolic tester concolic Unit Testing Engine (CUTE)和JCUTE。在本文中，我们使用了与Sen等人提出的JCUTE相同的版本[5,12]来自动生成需要的测试用例。

Das 等人使用程序转换技术实现了 C 程序的高修改条件/决策覆盖率(MC/DC)。Das 等人提出了布尔码转换器(Boolean Code Transformer, BCT)来增强覆盖率。他们已经使用 CREST 来生成测试用例，并提出了覆盖率分析仪(CA)来测量 MC/DC%。但是，他们没有对代码转换所带来的时间开销进行分析。Godboley 等人使用 Quine-McCluskey 最小化技术扩展了[19]中的代码转换器，该技术后来在[10]中扩展到转换 Java 程序。另一种用于分布式协同测试的代码转换方法可参考[20,21]。

但是，上述方法[10,11,19]均不关注能源功耗。在这篇论文中，作者扩展了[10]中的方法，以支持concolix测试中的能量感知。在我们提出的方法中，我们分别使用JPCT和JCUTE来转换代码和生成测试用例。在这项工作中，实验是在40个Java程序上进行的，而在[10]中是5个程序。

Fig.5 记录功耗与执行时间

Fig.6 LOC`与 power(注意:LOC`是被转换程序中的代码行数）

Sarkar 等人提出了两种新的度量方法 RepQ 和 RDI 来量化跟踪中发现的重复是否足以用于循环重构。实证研究还表明，RepQ 和 RDI 值可以识别数据并行循环，其中循环结构将是无效的。

Fig. 7 谓词和功率

Fig. 8 40 个项目的能源消耗

结果表明，该算法在数据并行循环中取得了显著的性能改善，有利于消除发散分支。他们已经转换了应用程序的GPU，以实现更好的分支覆盖。在我们的方法中，我们为Java程序提出了一种转换技术。因此，我们通过将这个转换器集成到共扼测试器来实现高的分支覆盖率。

本尼迪克特等人提出了一种算法和能量调谐机制。它帮助有能源意识的科学应用和工具开发人员。EnergyAnalyzer是HPCCLoud研究实验室正在开发的一种基于在线的能源消耗分析工具。同样，我们的工具也旨在使软件测试人员能够保持对测试过程中所消耗的能量的关注。

Hassine等人提出了一套asmetal特异性突变算子。这些被分为四类。根据不同的测试覆盖标准，他们提出的操作符用于评估使用ATGT (ASM测试生成工具)生成的测试套件的充分性。他们通过8个公开的金属案例研究来衡量他们的方法的适用性。他们的案例研究结果表明，所提出的突变算子可以用来比较不同的基于asmetal的测试覆盖标准，并成功地检测出测试套件中的缺陷。在我们提出的工作中，我们进行了concolic测试并计算了分支覆盖率。然而，我们通过评估时间、速度、动力和能量消耗来扩展我们的实验。

Mao等人针对软件结构测试提出了一种基于搜索的测试数据生成技术。他们提出了一个框架，其中采用了粒子群优化(PSO)技术，因为它简单，收敛速度快。针对测试数据生成问题，将被测程序的输入编码成粒子。生成测试数据输入后，覆盖率信息由测试驱动程序收集。然后根据这些信息计算分支覆盖率的适应度值。因此，PSO使用适应度来调整搜索方向。最后得到覆盖率最高的测试数据集。在我们的方法中，我们使用动态符号执行来自动选择输入值,JCute工具生成测试用例并生成分支覆盖率作为输出报告,tongsh我们还计算了测试用例生成的速度。

Fig. 9 时间与功率

Fig. 10 能量消耗的增加相对于时间的增**加**

Varshneyetal。[26]提出了一种基于遗传算法的优化技术，利用优势度和分支距离的概念来生成一个程序的数据流依赖关系(通用准则)的测试数据。它们的方法可以用于有/没有循环和过程的程序。在我们提出的工作中，我们使用共曲线测试工具来生成测试用例，这些用例将被进一步用于计算分支覆盖率。

Khan等人提出了一种从UCM符号到统一建模语言(UML)序列图的模型转换方法，以促进从需求到高级设计构件的转换。他们还介绍了该模型转换在电梯控制系统中的应用。在我们提议的工作中，我们将一个Java程序转换为它的转换版本。这个转换后的程序由嵌套的if-else语句组成。通过使用输入Java程序的转换版本，我们实现了更高的分支覆盖率。

表10总结了我们对不同作者提出的一些可用的相关工作的调查。第三栏名为框架，列出了研究人员在各自的工作中采用的不同工具和技术，第四栏给出了简要描述。S1. 1至7号中列出的研究工作是基于电力和能源消耗的研究。这些工作的重点是传播绿色IT和绿色软件工程的意识。研究工作列在第8至17号的S1中，涉及共渗和基于覆盖的测试。表10中的最后一行显示，我们提议的工作是在一个名为Green-ABCE的工具上实现的，它确认了能量感知和一致性测试。

Table 10 基于 concolic 和覆盖的测试方法的总结

Table 11 基于 concolic 和覆盖的测试方法的特点

我们根据调查中确定的一些特征，将我们提出的工作与现有的相关工作进行了比较。该表清楚地显示了列中显示的这些特征是否被(对号)合并到(X)所引用的相关工作中。在现有的工作中，只有Li等人对软件测试的能耗进行了分析。请注意，S1. 2至7中提到的研究工作已经提出了一些计算能源和电力消耗

的方法。但是他们不关注软件测试技术。S1. 8 - 17中所列的研究工作只关注软件测试，没有涉及到能耗。表11中的最后一行显示，我们提议的工作(在Green-ABCE工具上实现)满足所有提到的特性。我们提出的工作涉及软件测试和软件工程的绿色计算方面。据我们所知，我们开发的工具(Green-ABCE)是第一个能够显示所消耗的能量，同时增强面向对象(Java)程序的分支覆盖率的concolic测试工具。因此，这是我们迈向绿色软件测试的第一步。

总的来说，通过我们的文献研究，我们得出结论，一些现有的工作是为了传播绿色软件工程的意识。但是，根据我们的知识和研究，我们提出的工作是迈向绿色软件测试的新步骤。我们集成了一些开源工具，并开发了一些工具来执行绿色软件测试过程。

我们提出的工具产生了几个有用的成果，这些成果在测试领域和绿色计算领域都很重要。我们可以看到，只有Li等人的[2]测试参数计算与能源报告。表11中列出的作者没有一个同时从事这两个领域的工作(软件测试和绿色计算)。这是我们与其他公司的主要区别。我们的主要贡献是实现与电力和能源报告一起的高分支覆盖率。因此，我们使用一些开源的和我们开发的工具(JPCT,jCUTE,JouleMeter和Energy Calculator)开发了Green-ABCE，这些在前面的第3节已经详细讨论过了。在我们的工作中，在场景1中，40个程序的平均计算/执行时间为23,678.93 ms，而在场景2中，40个程序的平均计算/执行时间为27,176.42 ms。场景1的平均计算速度为，速度= 4.66 TC/s，场景2的平均计算速度为?= 2.27 TC/s为40个程序。我们提出的方法为我们测试的40个程序实现了大约13.5%的更高的分支覆盖率，并且开销为0.4 kJ的能源消耗。所有程序的平均能耗约为5.6 kJ。

6 对工具有效性的威胁

与现有的许多作品一样，这一作品也存在一定的局限性。以下提到的是对Green-ABCE有效性的威胁。

1.我们的方法面临的主要威胁与目标项目有关。为这个实验所选择的程序是可接受共线测试的，并且不显示特性(如多线程、分布式程序)，如果存在，可能不适合提议的方法。虽然共融测试对于安全临界系统是必不可少的，但我们的实验方案并不代表安全临界类别。

2. JCUTE中使用的符号执行引擎有一定的局限性，这对我们方法的有效性构成了另一个威胁。

3.该工具中使用的约束求解器可能不够强大，无法计算在非常大的工业应用程序中满足约束的具体值。我们仍在努力克服这一限制。

4.在生成测试用例时，Green-ABCE(由于代码转换)的速度较慢，这是该建议工作的另一个问题。虽然结果并没有建立速度下降与程序大小增加之间的直接关系，但是对于非常大的程序，由于实验程序不能很好地代表工业程序，速度预计会变慢。

5.JouleMeter是为windows 7操作系统开发的。因此，我们的工具目前仅限于这个特定的平台。

6.另一个对有效性的威胁是，JolueMeter的读数仅仅是功耗的参考值。但JouleMeter手册建议集成电子专业模式，如Wattsup，以获得更准确的读数。因为电子专业模式的成本和当地市场的不可用，拟议的工作没有利用任何这种模式。我们的框架只使用JouleMeter的代理读数。

7 总结与未来工作计划

我们提出了一个用于Java程序的concolic测试框架，并在一个名为Green-ABCE的工具中实现。提出的框架是设计和开发的，以衡量能源消耗，同时分析增加的分支覆盖率使用共曲线测试。从整体上介绍了绿色- abce的概念，并对其不同组成部分的工作进行了详细的讨论。实验结果表明，与现有方法相比，本文提出的测试用例生成方法具有更好的分支覆盖率。对于开销为0.4 kJ的能源消耗的40个程序，我们提出的方法实现了大约13.5%的分支覆盖率增长。所有程序的平均能耗约为5.6 kJ。

在未来，我们将重点克服我们的方法的有效性所面临的威胁。我们将把这项工作扩展到实施其他节能改造技术，如Java版本的专用或编码变压器(EX-NCT)，以实现高分支覆盖率。我们还计划对MC/DC测试方法进行能源消耗方面的工作。此外，我们还打算开发一个分布式框架来增加我们方法的可伸缩性。

参考文献：略