软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*,2016,27(4):943−954 [doi: 10.13328/j.cnki.jos.004968]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

# 面向 CPU+GPU 异构计算的多目标测试用例优先排序*

边 毅, 袁 方, 郭俊霞, 李 征, 赵瑞莲

(北京化工大学 计算机系,北京 100029)

通讯作者: 李征, E-mail: lizheng@mail.buct.edu.cn

**摘 要**: 测试用例优先排序是一种基于整个测试用例集以寻找最优测试用例执行序列的软件回归测试技术.由于其能够尽早地发现错误,同时应用灵活度高、不会漏掉重要测试用例等,在实际软件测试过程中可以有效提高测试效率.多目标测试用例优化排序是寻找同时覆盖多个测试准则的用例执行序列,通常采用演化算法优化求解,但执行时间较长,严重影响了在实际软件测试中的应用.采用先进的 GPU 图形卡通用并行计算技术,提出了面向 CPU+GPU 异构计算下的多目标测试用例优先排序技术,在 NSGA-II 算法中,实现了基于序列编码的适应度函数计算和交叉操作的 GPU 并行计算,在近 6 万行有效代码的工业界开源程序上实现了 30 倍的计算效率提升.同时,实验验证了不同并行策略的计算加速比,提出了切实可行的 CPU+GPU 异构计算模式,并提供了相应的原形工具.

**关键词**: 回归测试;测试用例优先排序;多目标优化;异构计算

**中图法分类号**: TP311

中文引用格式: 边毅,袁方,郭俊霞,李征,赵瑞莲.面向 CPU+GPU 异构计算的多目标测试用例优先排序.软件学报,2016,27(4):943−954. http://www.jos.org.cn/1000-9825/4968.htm

英文引用格式: Bian Y, Yuan F, Guo JX, Li Z, Zhao RL. CPU+GPU heterogeneous computing orientated multi-objective test case prioritization. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):943−954 (in Chinese). http://www.jos.org.cn/1000-9825/4968.htm

## CPU+GPU Heterogeneous Computing Orientated Multi-Objective Test Case Prioritization

BIAN Yi, YUAN Fang, GUO Jun-Xia, LI Zheng, ZHAO Rui-Lian

(Department of Computer Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China)

**Abstract**: Test case prioritization is a type of technique that aims at searching for the test case execution sequence to find faults early based on the whole test case suite. This technique is flexible and barely can miss important test cases, which contributes much benefit to regression testing. Multi-objective test case prioritization, where evolutionary algorithms have been widely used, aims to find a test case execution sequence that suits multiple test criteria. However, the drawback of large computation cost of the algorithms can greatly reduce the value of industrial application. This article proposes a CPU+GPU heterogeneous Computing orientation based multi-objective test case prioritization technique that utilizes advanced general purpose graphic process unit (GPGPU) technique to accelerate the process of test case prioritization. In experiment based on parallel structure, the sequential based parallel fitness and crossover operation computation is designed on NSGA-II and at last achieves 30 times speed-up rate on a well-known industrial open source project. Based on the systematic study on the benefit of different types of parallel strategies, a CPU+GPU heterogeneous computing framework is proposed and a prototype of tools is developed and available online.

**Key words**: regression testing; test case prioritization; multi-objective; heterogeneous computing

* 基金项目: 国家自然科学基金(61170082, 61472025); 教育部新世纪优秀人才计划(NCET-12-0757)

Foundation item: National Natural Science Foundation of China (61170082, 61472025); Program for New Century Excellent Talents in University of Ministry of Education of China (NCET-12-0757)

收稿时间: 2015-06-24; 修改时间: 2015-10-15; 采用时间: 2015-11-20; jos 在线出版时间: 2016-01-13

CNKI 网络优先出版: 2016-01-14 13:16:09, http://www.cnki.net/kcms/detail/11.2560.TP.20160114.1316.007.html

© 中国科学院软件研究所 http://www.jos.org.cn

软件回归测试是指在软件开发新版本的过程中,重新运行原有测试用例,来检测已发现的错误是否被修复和是否引入了新的错误,是保证软件质量、提高软件测试效率的重要技术手段.研究指出:软件开发过程中,软件回归测试已成为整个开发过程的“瓶颈”[1].例如,在软件模块整合过程中,软件回归测试耗时占到了整个软件开发过程的 1/3.因此,研究如何在保证软件质量的同时提高软件回归测试的效率,对提高软件开发中的迭代速度,缩短软件开发周期有着重大的意义.

研究人员提出了多种软件回归测试优化技术,依据其主要手段的不同,主要分为三大类:测试用例选择、测试用例集最小化[2,3]和测试用例优先排序[4].有研究发现:测试用例集最小化和测试用例选择都有可能遗漏关键测试用例,导致错误漏检[5].测试用例优先排序也称为测试用例预优化[6],是根据具体的测试需求对测试用例进行优先排序,以达到用最小代价满足测试准则的目的.由于测试用例优先排序技术的优化对象为整个测试用例集,即寻找一个最优的测试用例集执行顺序,因而可以避免漏检问题.但由于其过程是一个 NP 问题[7],通常采用启发式搜索算法寻找最优或次优序列,如模拟淬火算法、爬山算法、遗传算法等.有实验证实,遗传算法在解决测试用例优先排序问题方面可以获得更好的结果[7].近年来,粒子群算法[8]、蚁群算法[9]也被逐步研究和应用于这个领域,该领域被称为基于搜索的软件测试[10,11].

在回归测试过程中,通常需要满足多个目标.例如,要同时满足多个覆盖准则,还要尽量减少测试用例的执行时间[8].多目标优化在解决软件回归测试的相关问题上获得了良好的效果[12,13],其中以 Non-dominated Sorting Genetic Algorithm(NSGA-II)算法[14]为代表的多目标演化算法被广泛使用.该类算法会建立包含多个个体的种群,并进行多次迭代,实现个体进化,直到发现最优解.多目标演化算法具有良好的普适性和规模可扩展性,但随着问题规模的扩大,也会不可避免地出现计算效率降低、计算耗时急剧增加的现象.

效率问题是阻碍新技术进入产业界的一个重要瓶颈,通常可以通过科学和工程两个方面来解决:从科学的角度解决效率问题是用先进的方法改进算法,例如,通过将超体概念被引入遗传算法来加速优先排序[15],袁方等人将遗传算法的上位基因概念引入测试用例优先排序,提出新的遗传算子来加速优先排序的过程[16].同时,也有学者研究对于遗传算法中的参数调整,进而加速算法的收敛过程[17];从工程角度解决问题包括采用并行计算代替串行计算等.而演化算法实现过程中对种群中多个个体进行相同的计算,自身就拥有潜在并行性[9].在并行计算框架的选择上,由于软件测试已经占软件开发成本的很大一部分,若采用高性能并行计算集群,则会进一步增加开发成本.基于图形卡的统一计算框架 CUDA(compute unified device architecture),利用 GPU 图形卡的多处理内核实现单指令流多数据流的高性能并行计算,在浮点运算和大规模数据运算等方面,可以提供数十倍乃至上百倍于 CPU 性能的运算能力,具有高效和低成本的特点.但同时,CPU+GPU 异构架构模式会涉及到任务划分和协作等问题,所以针对具体问题,要科学地研究其使用方法.在 CPU+GPU 异构架构下的异构计算(heterogeneous computing),需要处理好不同类型指令集和体系架构的计算单元的协同计算,其中,CPU 擅长于处理不规则数据结构和不可预测存取模式的复杂指令调度、循环、分支、逻辑判断以及执行等,而 GPU 擅于处理规则数据结构和可预测存取模式的大规模简单运算.

基于 GPU 并行计算的多目标测试用例最小化[18]、多目标测试用例选择首先被提出[4],编码方式均为“0/1”二进制编码.Shin 等人提出将计算过程转换为矩阵运算,进而通过直接调用 CUDA 库函数来加以实现.

测试用例优先排序技术的操作对象为全部测试用例的执行次序,“0/1”的二进制编码方式不再适用,不能将计算过程转换为矩阵运算,所以不能通过简单调用 CUDA 库函数实现并行计算.而测试用例优先排序是一种重要的软件回归测试技术,要通过并行解决效率问题.研究、设计并实现其在 CPU+GPU 异构架构下的协同并行计算模式,是首先需要解决的问题.

在本文的研究中,首先实现了基于 NSGA-II 算法的测试用例多目标优先排序;在详细分析算法操作的计算效率和设计 GPU 并行策略的基础上,理论分析并实验验证了不同并行策略下的效率提升和成本花费,提出了一种基于 CPU+GPU 异构下的多目标测试用例优先排序并行计算模式.

本文的主要贡献是:

(1) 针对序列编码方式,设计并实现了 NSGA-II 算法中适应度函数和交叉操作的通用 GPU 并行方法;

© 中国科学院软件研究所 http://www.jos.org.cn

(2) 针对基于 CPU+GPU 异构模式下的多目标测试用例优先排序问题,设计了多种算法并行策略,并通过大规模开源程序在单显卡 Tesla M2050 上的并行实验,结果表明,最大能达到近 30 倍的计算效率提升;

(3) 实验结果同时表明:引入交叉操作的 GPU 并行虽然可以进一步提高计算加速比,但由于交叉操作 GPU 编程相对复杂,因此并不建议实际应用中实施交叉操作的 GPU 并行;

(4) CPU+GPU 异构模式下的多目标测试用例优先排序问题与程序规模成正比关系,越大规模的程序并行效率提升得越显著;

(5) 开发了相应工具,并提供相应的开源程序发布至 Github(https://github.com/themythfang/parellel-test-case-prioritization.git).

本文第 1 节介绍测试用例优先排序领域的问题及定义.第 2 节介绍 NSGA-II 算法在基于 CUDA 平台下的不同并行策略.第 3 节描述具体的实验设计和实验结果.第 4 节对实验结果进行详细分析.最后是总结和展望.

## 1 多目标测试用例优先排序相关问题描述

测试用例优先排序技术是在软件回归测试过程中,对原有的测试用例集合,寻找满足测试准则的最优测试用例执行序列,其定义如下.

**定义 1(测试用例优先排序)**. 已知一个测试用例集合 $T$、$T$ 的所有排列 $PT$ 以及一个能够度量 $PT$ 优劣的函数 $f$,$f$:$PT\to R$ 的函数,其中,$R$ 为一个实数.测试用例优先排序问题为:找出一个集合 $T'\in PT$,满足:

$$(\forall T'')(T''\in PT)(T''\neq T')[f(T')\geqslant f(T'')].$$

多目标测试用例优先排序是指,在测试用例优先排序中的度量函数 $f$ 并不唯一,而是由多个度量函数共同进行优化选择.在此,我们使用 Pareto 排序技术对测试用例个体进行多目标排序[19],其中,针对两个目标的 Pareto 排序规则定义如下.

**定义 2(多目标排序规则)**. 已知一个测试用例集合 $T$、$T$ 的所有排列 $PT$ 以及度量 $PT$ 优劣的函数 $f_1$ 和 $f_2$,$T'$ 和 $T''$分别是测试用例集合 $T$ 的两种排序,即,$T'\in PT$,$T''\in PT$,$T'$和 $T''$的相互关系如下:

(1) 当 $f_1(T')\leqslant f_1(T'')$且 $f_2(T')\leqslant f_2(T'')$,或 $f_1(T')\geqslant f_1(T'')$且 $f_2(T')\geqslant f_2(T'')$时,则可得到两个集合,集合 $a=\{T'\}$,集合 $b=\{T''\}$,集合 $a$ 中的各个元素对集合 $b$ 中的各个元素为支配关系,则集合 $a$ 为集合 $b$ 的支配集合;集合 $b$ 中的各个元素对集合 $a$ 中的各个元素为被支配关系,集合 $b$ 为集合 $a$ 中的被支配集合;

(2) 当 $f_1(T')\leqslant f_1(T'')$且 $f_2(T')\geqslant f_2(T'')$,或 $f_1(T')\geqslant f_1(T'')$且 $f_2(T')\leqslant f_2(T'')$时,则得到集合 $a=\{T',T''\}$,其中,集合 $a$ 中的元素互不支配.

为度量不同测试用例序列在回归测试过程中的效率,本文设定了两个优化指标,分别对应两个度量函数来评价一个测试序列的优劣.

- 一方面,为了保证代码覆盖率[7],本文采用了平均代码覆盖率(average percentage statement coverage,简称 APSC)来衡量测试用例序列的优劣.每个测试用例序列的 APSC 值在 0~1 之间,APSC 值越大,说明测试用例序列越有效;
- 另一方面,为了提高回归测试的效率,本文采用了测试用例执行时间(effective execute time,简称 EET)[20],即:在达到最大覆盖过程中执行测试用例所需要的时间.EET 越小,说明测试用例序列越有效.

**定义 3(Fitness 适应度函数)**. 假设被测程序拥有 $M$ 行代码,其测试用例集包括 $N$ 个测试用例,$TS_i$ 表示第 $i$ 行代码首次被覆盖时的测试用例序号,$i\in[1,N]$,测试用例集合执行到 $N'$个测试用例时覆盖率达到最大(即,$N'$等于 Max($TS_i$),$i\in[1,M]$),$ET_j$ 表示第 $j$ 个测试用例的执行时间,$j\in[1,N]$,则 APSC 与 EET 可表示如下:

$$APSC=1-\frac{TS_1+TS_2+\ldots+TS_M}{MN}+\frac{1}{2N} \tag{1}$$

$$EET=\sum_{j=1}^{N'}ET_j \tag{2}$$

在回归测试过程中,APSC 直接反映了测试用例序列达到测试目标的效果,即:APSC 值越高,说明相关测试

© 中国科学院软件研究所 http://www.jos.org.cn

序列有可能越早发现存在的错误.而 EET 衡量的是检测到所有的错误所需要的时间.

基于以上对多目标测试用例优先排序相关问题的定义,适应度函数取 APSC 与 EET,实现基于多目标的测试用例优先排序的 CPU+GPU 并行算法框架设计.

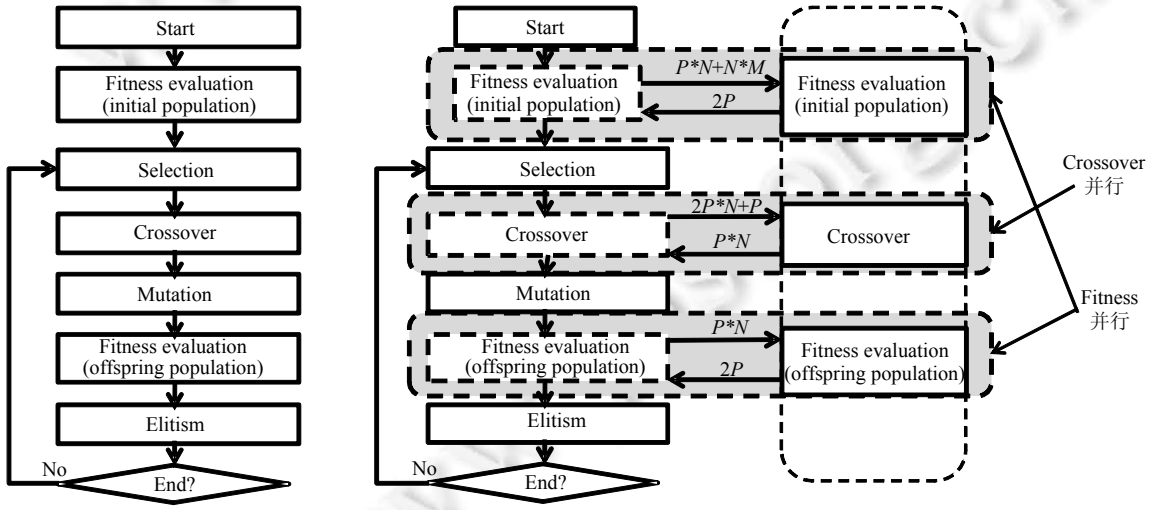
## 2 基于 CPU+GPU 的异构并行多目标测试用例优先排序

在验证基于 GPU 的并行策略对测试用例优先排序问题的计算效率影响时,采用了软件回归测试优化技术中广泛使用的多目标演化算法 NSGA-II,算法流程如图 1(a)所示.相较于普通的遗传算法,NSGA-II 算法也包含种群初始化,选择(selection)个体作为父代,通过交叉(crossover)和变异(mutation)操作演化出子代.但在选择哪些个体进入到下一轮迭代时,引入了非支配排序来处理多目标问题.同时,为了增加种群的多样性,采用了精英选择(elitism).在实现 NSGA-II 算法的过程中,个体通过编码方式表示.由于测试用例优先排序是一个序列问题,“0/1”编码方式无法应用,所以采用了序列编码,即:一个个体是一个测试用例执行序列,个体的基因为测试用例编号.

在 CPU+GPU 的异构模式下,GPU 擅长处理大规模的简单运算,而 CPU 擅长与逻辑运算.

- 针对 GPU 进行并行编程,首先要考虑程序的可并行性以及并行的意义大小.一般情况下,只选取算法中计算量大且计算方式单一的步骤进行并行化.在测试用例优先排序中,对每个个体都要计算 Fitness,这种大量的重复操作适合并行,优先考虑在 GPU 中进行处理;
- 其次,在每次迭代过程中,选择、交叉和变异操作也都在重复执行,理论上都可以并行执行,以进一步提高算法的并行度.但这些操作过程的计算相对复杂,需要精心设计 GPU 并行算法.我们选择了其中耗时最大的 crossover 操作,设计相应的 GPU 并行策略.针对变异操作,我们采用了交换变异(exchange mutation),该方法随机选取测试用例序列中的两个位置上的基因并进行交换.由于其运算成本相对较低,所以采用串行方式执行.

图 1(b)显示了在 CPU+GPU 异构模式下,NSGA-II 算法的并行策略,其中,Fitness 计算(包括初始种群和每次种群迭代)和交叉操作在 GPU 端并行,其他操作在 CPU 段执行.由于数据需要在 CPU 和 GPU 之间传输,图中还给出了数据传输量,这将在后文详细给出解释.




(a) 串行 NSGA-II 算法 (b) NSGA-II 并行框架

Fig.1 The framework of serial and parallel NSGA-II algorithm

图 1 串、并行 NSGA-II 算法框架

© 中国科学院软件研究所 http://www.jos.org.cn

在具体 GPU 的并行算法设计中,由于 GPU 并行编程不同于其他的多核程序、多线程程序并行,设计中主要需要考虑以下 3 点:(1) 任务划分的方式;(2) 数据所在的存储器类型;(3) GPU 与 CPU 交换的数据量.其中,任务划分决定了程序的并行粒度,直接影响并行程序的性能:如果并行程序的粒度过大,则会导致任务没有进行足够细致的划分,无法充分使用 GPU 的性能;而如果并行粒度过小,可能导致工作繁琐,而其并行的性能却不一定是最优的.例如,针对 Fitness 并行计算,可以选择一个 Block 计算一个个体的 Fitness 值,也可以选择一个 Block 计算一个种群的 Fitness 值,但是以上两种并行的粒度设计其效率的差异是巨大的.任务划分也直接决定了数据所在的存储器类型.GPU 有整体的显存(global memory),同时,计算核心按 Block 划分,并为每个 Block 提供了单独 Shared Memory 和 Cache,但 Block 之间是不能共享的,因此在设计 Block 所处理的任务时,需要考虑 Block 之间的数据独立.例如:多个 Block 间共享的数据可以存储在显存中,但是对其访问的速度是最慢的;一个 Block 内部所有 Thread 的共享数据可以存放在 Shared Memory 中,其访问速度次之;而 Cache 存储器的访问速度最高.由于显卡的显存与计算机内存相互独立,两者之间只能通过 PCI-E 总线进行通信,传输耗时要比内存与 CPU 间的通信速率低很多.而在很多情况下,CPU 与 GPU 之间交换数据的量是影响 GPU 并行程序性能的一个瓶颈,所以设计中要减少两者之间的数据传输次数.

### 2.1 Fitness并行计算策略

在测试用例优先排序的 Fitness 计算中,APSC 和 EET 这两个目标被用来指导多目标测试用例优先排序.由公式(2)中 $N'$与 $TS_i$ 的关系,结合公式(1)可得所选取的两个优化目标均与 $TS_i$ 的计算相关,即,第 $i$ 行代码首次被覆盖的测试用例序号.在得到 $TS_i$ 之后,只需将相应的 $TS_i$ 代入公式即可求得 Fitness 值.相较于 $TS_i$ 的计算,将其代入公式(1)与公式(2)的过程计算量十分小,因此,Fitness 的并行核心是 $TS_i$ 计算的并行.下面将从任务划分、数据传输和数据存储这 3 个方面详细解释 $TS_i$ 计算的并行策略.

A. 任务划分

在并行 Fitness 计算时,利用此时个体与个体间相关性不大的特性,可以采用一个 Block 处理一个个体.此时,不同的 Block 之间不需要共享数据,可以达到数据独立.而在一个 Block 内部,需要考虑对于 $TS_i$ 计算的并行粒度设计.计算 Fitness 需要使用覆盖信息矩阵,矩阵中的每行代表一个测试用例,每列代表一行代码,行与列的交叉处为该行测试用例对该列代码的覆盖情况.计算 $TS_i$ 的过程就是在覆盖信息矩阵中,依次搜索每列(一行被测代码)第 1 次被覆盖的测试用例序号.而针对同一个体计算不同代码行的 $TS_i$ 过程是相互独立的,因此可以同时分配多个 Thread,每个 Thread 负责计算一行代码的 $TS_i$.在阅读相关 SDK 后,本实验所采用的平台单个 Block 内最优 Thread 数量为 512,如果测试用例数超过 512 这一数值,我们采用分组处理的方式,即:以 512 个测试用例为一组计算,分成多组进行,最后一组不足 512 的情况同样作为一组单独处理.

B. 数据传输

原测试用例集包含 $N$ 个测试用例,则种群中的每个个体包含 $N$ 个基因,对应 $N$ 个测试用例的编号.在算法初始化时,会传入 $N$ 测试用例对 $M$ 行语句的覆盖信息矩阵,传入数据量 $N\times M$;在每次迭代计算 Fitness 时,需要传入的数据为种群的测试用例序列信息,假定种群包含 $P$ 个测试用例序列,每个测试用例序列包含 $N$ 个测试用例的序号,则每次迭代传入数据量 $P\times N$;计算 Fitness 结束后,需要传出 $P$ 个个体的 Fitness 值,传出数据量为 $2P$.

C. 数据存储

在 Fitness 并行过程中,由于测试用例覆盖信息矩阵会被所有的 Block 访问,但该矩阵自身巨大,无法存储在 Shared Memory 中,因此将其存储于 GPU 的 Global Memory 中.由于父代的 id 会经常被访问,因此采用了读取速度最高的 Cache 进行存储.在计算公式(1)中的 $TS_1+TS_2+\ldots+TS_M$ 与公式(2)中的 $N'$时,采用了二分的计算方式,为了保证效率,采用了 Shared Memory 的存储方式.

综合考虑上述因素,Fitness 计算的并行策略如图 2 所示.在该策略中,一个 Block 处理一个个体的 Fitness 计算,而一个个体的 Fitness 计算由 $M$ 个 $TS_i$ 的计算构成,考虑到每个 Block 中的最优线程为 512 个,因此并行过程采用计算 512 个 $TS_i$ 为一组的分组策略.由图 2 中可知,每个 Block 独立计算一个按照测试用例序列重新排列后的代码覆盖矩阵.由于每个 Block 中的 Thread 在计算过程中必须达到时间同步,所以每个 Block 的计算时间为

© 中国科学院软件研究所 http://www.jos.org.cn

该组 Thread 最长的时间,即:当某一 Thread 搜索完 $TS_i$ 后,该线程需要等待该 Block 所有线程都完成搜索.即,单个 Block 计算中耗时最长的线程决定了整个计算过程的耗时.最差情况下,单个 Block 计算的时间复杂度为 $O(N)$,最好情况时间复杂度为 $O(1)$,即:在单个 Block 计算中,每个线程都在第 1 个元素位置找到了覆盖该行代码的测试用例.并行执行时,单个 Block 时间实际上为该 Block 耗时最长的线程所花费的时间,并行执行的次数大致为⌈$M$/512⌉,其中"⌈*⌉"为向上取整.但是并行计算需要相关数据传递.同时,GPU 每个运算单元的运算效率都远远低于 CPU.

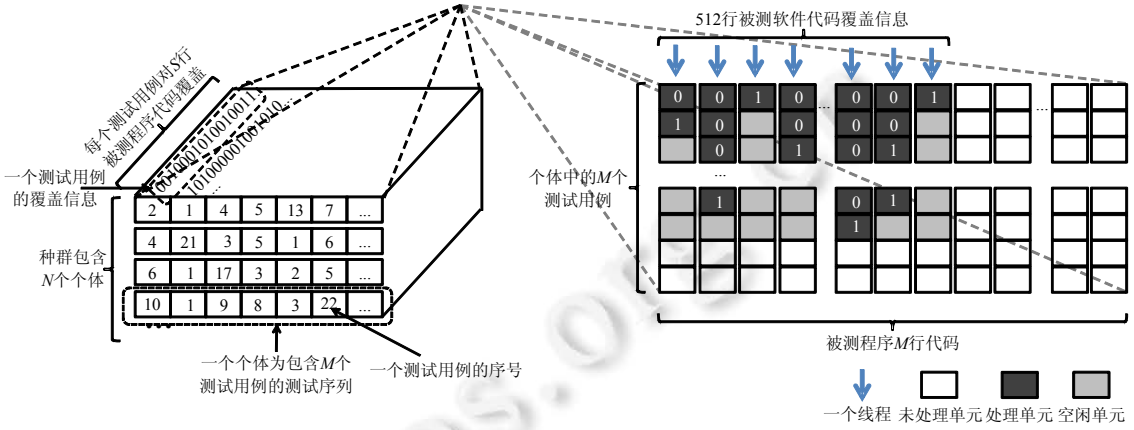



Fig.2 Parallel Fitness calculation strategy

图 2 并行 Fitness 运算

### 2.2 Crossover并行计算策略

本文所提出的 Crossover 并行计算策略针对的是序列编码,使用测试用例在原测试用例集中的序号作为该测试用例的标号,每个个体的序列即为测试用例的执行顺序.例如,原有测试用例集合中有 8 个测试用例,$T=\{t_0, t_1,t_2,t_3,t_4,t_5,t_6,t_7\}$,序列 5-3-2-7-6-0-1-4 表示 $t_5$ 测试用例第 1 个执行,$t_7$ 测试用例第 4 个执行等等.

交叉操作的核心是选取个体中优秀的基因遗传到后一代的新个体中,设计主要包括如何确定交叉点和如何进行部分基因交换.针对序列编码的交叉方式要保证交叉后的新个体中每个测试用例只出现 1 次,所以计算过程更复杂一些,在 CUDA 环境下实现并行,必然要使用到相应的通信机制来保证上述特性,编程实现难度较大.通过分析,我们把测试用例优先排序的交叉算子操作分为两个步骤:(1) 获取子代测试用例的新位置;(2) 将对应测试用例放在新的位置上.不同的序列编码交叉操作,第 1 步通常各不相同,但第 2 步操作都是相同的.从通用性的角度考虑,本文设计的并行交叉操作只并行第 2 部分,第 1 部分以串行代码实现.

在具体交叉算子选择过程中,我们采用了一种单点交叉方式验证并行的有效性.本文选取的交叉方式具体过程为:给出一个父代的基因,随机地选取父代基因中的 $K$ 个位置,将其按照原有的次序作为子代的前 $K$ 个基因;子代剩余的基因由父代的其余基因按照原有的位置组成.图 3 给出了交叉算子的示例,随机地选取 3 个基因位,将这些位置上的基因 2,0,1 按照原有的顺序作为子代的前 3 个基因,其余的基因按照原有的次序作为子代的其余基因,最后得到子代的基因为 2,0,1,5,3,7,6,4.

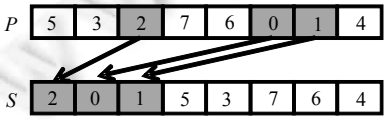



Fig.3 Crossover operator

图 3 交叉算子

© 中国科学院软件研究所 http://www.jos.org.cn

A. 任务划分

在并行交叉时,与并行 Fitness 相似,同样可以利用此时个体与个体间相关性不大的特性,使用一个 Block 处理一个个体.此时,同样地,不同的 Block 之间不需要共享数据,可以达到数据独立.而在一个 Block 内部,需要考虑如何按照基因位置的映射生成子代的基因.此时,采用一个 Thread 处理一个基因位的方式来生成子代的基因.一个 Block 内同样采用了 512 个 Thread,每个 Thread 可以独立处理一个基因位,即如果种群包含 $P$ 个个体,每个个体有 $N$ 个基因位,则其并行化算法可以通过使用 $P\times512$ 个线程进行计算(如果 $N<512$,则可使用 $P\times N$ 个线程计算).

与并行计算 Fitness 相同,每个 Block 中的线程在计算过程需要同步,单个 Block 计算中耗时最长的线程决定了整个计算过程的耗时.由于 Block 中的计算用了 512 为一组的策略,整个并行交叉的算法复杂度为$\lceil M/512\rceil$.

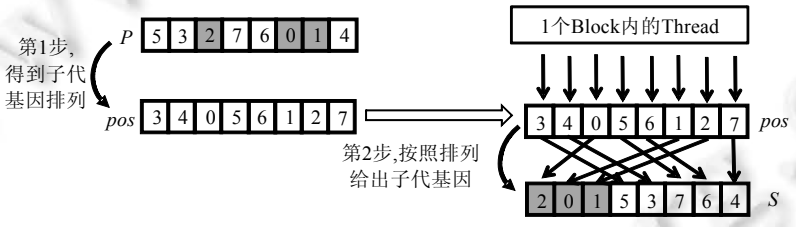
B. 数据传输

在串行算法得到所有的 *pos* 向量之后,产生子代需要父代的基因以及 *pos* 向量,传入数据量为 $P\times N$;而并行策略中对于父代信息的访问是通过 id 与种群矩阵的方式,即需要传入 $P\times N+P$.因此,最终的传入数据量为 $2P\times N+P$.在产生了子代之后,需要将子代的测试用例序列传出,传出数据量为 $P\times N$.

C. 数据存储

在交叉并行过程中,由于子代与父代的测试用例序列信息会被所有的 Block 访问,而该信息本身较大,因此将其存储于 GPU 的 Global Memory 中.在组成子代的测试用例序列过程中,由于父代的 id 会经常被访问,因此采用了读取速度最高的 Cache 进行存储.

针对以上单点交叉算法,具体并行策略如图 4 所示,其中,图 4(a)所示为通过串行计算得到子代基因的新位置.例如,5 号基因对应的 *pos* 中的数值为 3,表示 5 号基因在子代中的新位置为 3(位置从 0 开始计算),以此类推.图 4(b)所示为一个 Block 内的 8 个 Thread 分别处理 8 个位置上的基因,将原有基因按照 *pos* 的指导放在对应的位置上,最终得到子代的基因序列.在并行交叉过程中,由于个体中每个基因位交叉后的位置为输入,因此每个 Block 在处理一个个体内的基因时是相互独立的,即,GPU 只需要将基因按照 *pos* 的映射生成子代的基因.




(a) Crossover 第 1 步(串行) (b) Crossover 第 2 步(并行)

Fig.4 Parallel Crossover strategy

图 4 Crossover 并行策略

## 3 实验及结果

为验证本文提出的基于CPU+GPU异构模式下的多目标测试用例优先排序技术,设计并计算不同并行计算模式在效率上的差异,我们进行了一系列的实验,在不同规模软件的测试用例优先排序过程中,度量了不同并行模式下 NSGA-II 算法各个操作执行效率的差异.

### 3.1 实验环境

表 1 给出了实验中所用到的被测程序及其代码有效行数和测试用例集的大小,其中,v8 为 Google 浏览器 Chrome 的 JavaScript 引擎,属开源项目,其余 3 个都来自 SIR 测试用例库中程序规模相对较大的程序.为了获取测试用例集,用随机算法从测试用例池中抽取测试用例,直至达到最大覆盖.针对 flex,space,bash 实验程序共抽取了 100 组测试用例集;针对 v8 抽取了 30 组测试用例集.表 1 中,最后 3 行给出了同一被测程序所有测试用例

© 中国科学院软件研究所 http://www.jos.org.cn

集所包含的测试用例的最大、最小和平均个数.实验平台为一台 GPU 服务器,配有 16 核英特尔 Xeon(R) CPU E5620 主频 2.40GHz,并行平台所使用的显卡为 Nvidia 公司的一块 Tesla M2050,CUDA 版本为 5.5,编程语言为 C++.

通过 Nvidia SDK 自带的 Benchmark 程序测出实验平台数据传输速率具体为:从内存到显存的带宽为 2 961.1MB/s,从显存到内存的带宽为 3 198.8MB/s.遗传算法种群大小设定为 64,交叉概率与变异概率为 100%,迭代次数为 100.

**Table 1** The statistics of software under test

**表 1** 被测程序及其规模

| 被测程序 | | flex | space | bash | v8 |
|---|---|---|---|---|---|
| 代码有效行 | | 3 016 | 3 815 | 6 181 | 59 412 |
| 测试用例集 | 最小 | 1 047 | 1 208 | 764 | 2 454 |
| | 最大 | 1 470 | 3 229 | 1 467 | 5 585 |
| | 平均 | 1 350.17 | 1 894.29 | 1 063.17 | 3 737.23 |

### 3.2 实验设计

在基于 CPU+GPU 的异构并行多目标测试用例优先排序中,我们设计了 Fitness 和 Crossover 的并行策略实验,分别进行了完全串行、单独并行 Fitness、单独并行 Crossover 以及两者同时并行的 4 组实验.为保证实验的有效性,针对 flex,space,bash 的每个集合,实验重复执行了 30 次;针对 v8,由于程序规模巨大,每个实验只重复了 10 次.实验结果取平均值.

### 3.3 实验结果

在实验中,我们统计了完全串行、单独并行 Fitness、单独并行 Crossover 以及两者同时并行时的总体运行时间和一次迭代中所执行的各个操作所需要的时间.在本节中,我们以表格的形式列出所有实验数据,在下一节中,我们将详细分析实验结果.

表 2 列出了完全串行的实验结果:所需总体时间以及各个操作所需要的平均时间.迭代平均时间是指迭代一次各个操作所消耗的时间和的平均值,而总时间为遗传算法迭代 100 次所消耗的时间.其中,初始化操作包含了覆盖信息的读取、一些变量的初始化以及首次的 Fitness 计算.从表 2 可以看出:Fitness 计算所消耗的时间占到了每次迭代过程时间的绝大部分,其次是交叉操作的时间.这个数据显示:本文的并行策略设计是正确的,把串行中最耗时的部分并行化,理论上可以极大地提高效率.

**Table 2** Time consumption of serial process (ms)

**表 2** 串行操作所需时间(单位:毫秒)

| 程序 | 初始化 | 选择时间 | 交叉时间 | 变异时间 | Fitness 计算 | 精英选择 | 迭代平均时间 | 总时间 |
|---|---|---|---|---|---|---|---|---|
| flex | 817.08 | 0.07 | 40.01 | 0.02 | 461.30 | 6.49 | 507.82 | 51 598.81 |
| space | 4 486.93 | 0.07 | 55.32 | 0.02 | 722.38 | 7.11 | 784.83 | 82 969.75 |
| bash | 2 053.04 | 0.07 | 28.40 | 0.02 | 1 632.39 | 5.57 | 1 666.38 | 168 690.93 |
| v8 | 64 423.36 | 0.07 | 99.67 | 0.02 | 44 794.12 | 8.39 | 44 902.20 | 4 574 940.76 |

表 3~表 5 分别给出了单独并行 Fitness、单独并行 Crossover 以及两者同时并行时各个操作的具体时间消耗.注意:由于并行产生了数据传输,不同的并行策略中数据传输的次数不同.在表 3~表 5 中,同时统计了其产生的时间消耗.

**Table 3** Time consumption of different operators in parallel Fitness process (ms)

**表 3** 并行 Fitness 时,各个操作的时间消耗(单位:毫秒)

| 程序 | 初始化 | 选择时间 | 交叉时间 | 变异时间 | 传入时间 | Fitness 计算 | 传出时间 | 精英选择 | 迭代平均时间 | 总时间 |
|---|---|---|---|---|---|---|---|---|---|---|
| flex | 805.11 | 0.08 | 44.43 | 0.02 | 3.88 | 14.25 | 4.31 | 7.66 | 129.51 | 13 756.37 |
| space | 1 095.03 | 0.08 | 58.80 | 0.02 | 2.20 | 50.09 | 0.98 | 7.97 | 202.44 | 21 339.20 |
| bash | 945.86 | 0.08 | 33.27 | 0.02 | 1.42 | 25.15 | 9.05 | 6.88 | 154.55 | 16 400.95 |
| v8 | 13 427.14 | 0.08 | 122.53 | 0.03 | 4.92 | 547.99 | 3.20 | 10.72 | 1501.99 | 163 626.60 |

© 中国科学院软件研究所 http://www.jos.org.cn

**Table 4** Time consumption of different operators in parallel Crossover process (ms)

**表 4** 并行 Crossover 时,各个操作的时间消耗(单位:毫秒)

| 程序 | 初始化 | 选择时间 | 传入时间 | 交叉时间 | 传出时间 | 变异时间 | Fitness计算 | 精英选择 | 迭代平均时间 | 总时间 |
|---|---|---|---|---|---|---|---|---|---|---|
| flex | 899.65 | 0.08 | 0.47 | 20.69 | 0.39 | 0.02 | 458.47 | 6.64 | 486.74 | 49 573.84 |
| space | 4 377.85 | 0.07 | 0.63 | 28.87 | 0.53 | 0.02 | 673.76 | 7.56 | 711.44 | 75 521.51 |
| bash | 2 221.37 | 0.07 | 0.34 | 15.42 | 0.34 | 0.02 | 1 677.34 | 6.57 | 1 700.11 | 172 231.87 |
| v8 | 899.65 | 0.07 | 1.09 | 54.37 | 0.92 | 0.02 | 51 280.87 | 18.84 | 51 356.17 | 5 203 800.74 |

**Table 5** Time consumption of different operators in parallel Fitness & Crossover process (ms)

**表 5** 同时并行 Fitness 与 Crossover 时,各个操作的时间消耗(单位:毫秒)

| 程序 | 初始化 | 选择时间 | 交叉传入时间 | 交叉时间 | 交叉传出时间 | 变异时间 | Fitness传入时间 | Fitness计算 | Fitness传出时间 | 精英选择 | 迭代平均时间 | 总时间 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| flex | 863.57 | 0.08 | 0.47 | 24.94 | 2.87 | 0.02 | 0.65 | 15.25 | 1.29 | 8.55 | 88.66 | 9 730.02 |
| space | 1 215.92 | 0.08 | 0.63 | 34.63 | 0.97 | 0.02 | 0.64 | 49.57 | 0.53 | 9.70 | 158.25 | 17 041.35 |
| bash | 983.48 | 0.08 | 0.34 | 19.19 | 4.23 | 0.02 | 0.46 | 25.81 | 5.56 | 8.66 | 123.73 | 13 356.08 |
| v8 | 12 086.06 | 0.08 | 1.13 | 68.63 | 1.13 | 0.02 | 2.20 | 604.11 | 0.75 | 22.03 | 1 438.92 | 155 978.49 |

## 4 实验结果分析

本文提出了基于 CPU+GPU 的异构并行多目标测试用例优先排序,所以,并行所带来的整体效率提升是我们首要关注的问题;其次,我们设计了多目标优化算法 NSGA-II 中 Fitness 和 Crossover 的并行策略,不同的并行策略对整体效率提升的效果是实验分析的另一个重点;最后,实验中采用了具有不同测试用例集大小的不同规模的程序,影响并行效率提升的相关因素分析也是实际中应用并行要考虑的.

### 4.1 整体效率提升分析

根据表 5 的数据,我们计算了串行总体时间以及同时并行 Fitness 和 Crossover 时所花费的总体时间的比值,即加速比,如图 5 所示.可以看出:在针对程序规模较小的 flex 与 space 时(有效代码行 3 000 左右),加速比在 4 左右;在针对 bash 的实验中,加速比达到了 10 以上;而在针对 v8 的实验中,程序规模增加到近 6 万行有效代码,加速比接近 30.再进一步通过表 5 数据可以看出:v8 程序在本文提出的基于 CPU+GPU 异构并行模式下执行多目标测试用例优先排序,可以在不到 3 分钟内(160 秒内)完成排序,这个实验结果已经可以被工业界所接受.

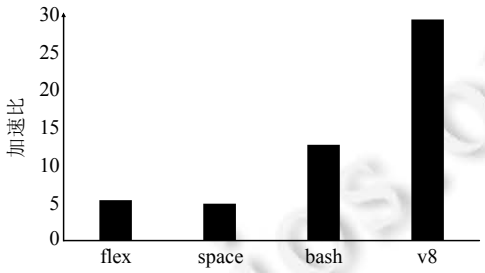



Fig.5 Speed-Up rates of parallel Fitness and Crossover process

图 5 同时并行 Fitness 与 Crossover 的整体加速比

### 4.2 不同并行策略的比较

从表 2 的数据中可以看出:Fitness 计算占总体时间的绝大部分,而 Crossover 是所有遗传操作中时间花费最大的.所以在 Fitness 并行的基础上,进一步设计了 Crossover 的并行.图 6 根据表 3 和表 5 的数据,计算了单独并行 Fitness 和同时并行 Fitness+Crossover 的加速比.可以明显看出:针对 4 个被测程序,后者比前者加速比略高.虽然两者差距不大,但可以看出,差距随着程序规模的扩大而增大.所以,在 Fitness 并行的基础上增加 Crossover

© 中国科学院软件研究所 http://www.jos.org.cn

的并行,可以进一步提高计算效率.随着程序规模的扩大,效果增加得越显著.

为了进一步分析并行交叉的效率提升情况,图 7 给出了 4 个程序单独并行交叉相对于串行交叉的加速比.可以看出:总体加速比在 1.6~1.8 之间,其中,v8 程序规模最大,单独并行交叉的加速比反而最小.与 Fitness 并行不同,并行交叉操作的加速比随程序规模大小的变化不够明显.具体原因如第 2.2 节所述:为了保证交叉算子并行策略的通用性,将交叉操作分为两步,第 1 步串行执行,仅第 2 步并行执行,所以加速比不大.综合考虑表 4 中的交叉时间,只占单次迭代时间的 5%左右,提升并行交叉算子的加速比对整体加速比影响不大.

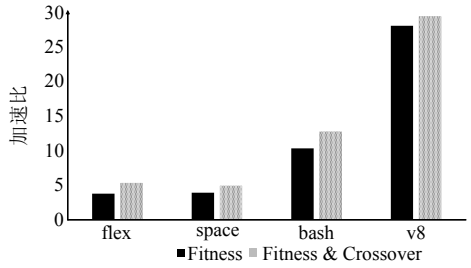



Fig.6 Comparison of speed-up rates beteeen Fitness parallellization and Fitness & Crossover parallelization

图 6 单独并行 Fitness 和同时并行 Fitness 与 Crossover 的整体加速比

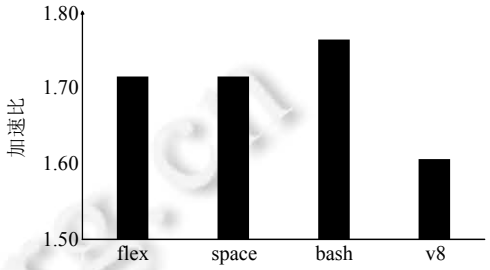



Fig.7 Speed-up rates of parallel Crossover process over serial Crossover process

图 7 并行 Crossover 相对串行 Crossover 的加速比

综合以上结果可以得到:针对 Fitness 计算并行加速的整体收益非常大;而在此基础上增加交叉的并行,其整体收益有增加但并不明显.同时,从第 2.2 节中可以看到:由于交叉操作的复杂性,在实现 GPU 并行编程时的算法难度较大.因此,建议在 CPU+GPU 异构模式下,测试用例排序并行计算可以只考虑单独并行 Fitness 计算.

### 4.3 算法加速比影响因素分析

在 CPU+GPU 异构并行计算中,数据要在 CPU 和 GPU 进行交换.为了分析数据交换对并行加速比的影响,表 3~表 5 给出了数据传入时间与传出时间,可以发现:并行交叉算子的数据传输中,数据的传入与传出时间基本上均小于 1ms;在并行 Fitness 计算的数据传输中,针对被测程序为 bash 与 v8,由于程序规模和测试用例集规模均较大,所以耗时较大,但是传入时间基本上在 5ms 左右,数据交换的耗时占整体运行时间的比例不大,对于整个加速比的影响几乎可以忽略.从图 6 中可以看出,并行加速比基本与程序规模成正比.即:随着程序规模的扩大,并行加速比随之增大.所以在回归测试中,程序规模越大,采用测试用例优先排序技术的时间花费越大,采用 GPU 并行的效率提升情况越明显.

由于 Fitness 计算占整体时间的绝大部分,我们进一步分析了单独并行 Fitness 的加速比.由表 3 可以得出,程序针对 Fitness 的加速比与被测程序规模成正相关.这是由于,Fitness 计算需要遍历测试用例的覆盖矩阵,而 $TS_i$ 的计算次数等于被测程序的代码行数.一次计算的 $TS_i$ 次数越多,并行实现所带来的额外开销在总时间中所占比重就越低,图 6 中计算并行的加速比就越大.可以注意到,space 程序加速比略微偏小可能与测试用例集相对较大有关.

### 4.4 实验结果影响因素分析

在本文实验中的 Fitness 采用了平均语句覆盖 APSC,实际应用中可以采用平均分支覆盖(APDC)或平均语句块覆盖(APBC).后两种从覆盖矩阵规模上远远小于 APSC,但计算公式相似,可以直接采用本文的并行策略.为了更好地验证规模对并行效率提升的影响,本文采用了 APSC.

SIR 库提供了十几个开源程序和测试用例集合,本文实验中只选择了 3 个规模较大的程序,同时补充了 1 个工业界的开源程序 v8,程序规模是 SIR 库中最大规模程序的 10 倍,可以有效地分析程序规模和并行加速效率提升之间的关系.为保证实验的有效性,本文采用了 SIR 抽取测试用例集的方法,对 4 个程序重新抽取了测试用例

© 中国科学院软件研究所 http://www.jos.org.cn

集合,保证了程序之间的一致性.

由于本文所设计的实验较多,如果单独运行需要花费近 1 年的时间,因此大部分实验采取了在同一台服务器上多线程并行运行的方式,会给实验结果带来一定的误差.但经过我们的分析,实验所带来的误差并不影响最终结论的得出.

## 5 总 结

本文针对软件回归测试用例优先排序的效率问题,提出了面向 CPU+GPU 异构计算的多目标测试用例优先排序,并就不同的异构并行策略进行研究分析.实验测试了 4 个规模较大的被测程序,研究了 NSGA-II 算法中序列编码的 Fitness 并行策略和 Crossover 并行策略,并根据实验结果分析,提出一个可行 CPU+GPU 异构架构下并行模式.实验结果显示:针对近 6 万行有效代码的工业界开源程序 v8,测试用例集的大小接近 4 000,在本文提出的 CPU+GPU 异构并行框架下,单 GPU 就实现了近 30 倍的加速提升,可以在 3 分钟内完成测试用例的多目标优先排序,可以有效地推动测试用例优先排序技术在工业界的实际应用.

**References**:


[1] Jiang B, Chan W. On the integration of test adequacy, test case prioritization, and statistical fault localization. In: Proc. of the 10th Int'l Conf. on Quality Software (QSIC). IEEE, 2010. 377−384. [doi: 10.1109/QSIC.2010.64]

[2] Zhang XF, Xu BW, Nie CH, Shi L. Approach for optimizing test suite based on testing requirement reduction. Ruan Jian Xue Bao/Journal of Software, 2007,18(4):821−831 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/20070404.htm

[3] Gejian D, Yanni Z, Lu Z. Study of test suite minimization based on ant colony algorithm. Computer Engineering, 2009,35(6): 213−218 (in Chinese with English abstract).

[4] Yoo S, Harman M. Regression testing minimization, selection and prioritization: A survey. Software Testing, Verication and Reliability, 2012,22(2):67−120. [doi: 10.1002/stv.430]

[5] Wong WE, Horgan JR, London S, Mathur AP. Effect of test set minimization on fault detection effectiveness. In: Proc. of the 17th Int'l Conf. on Software Engineering (ICSE'95). IEEE, 1995. 41. [doi: 10.1145/225014.225018]

[6] Chen X, Chen JH, Ju XL, Gu Q. Survey of test case prioritization techniques for regression testing. Ruan Jian Xue Bao/Journal of Software, 2013,24(8):1695−1712 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4420.htm [doi: 10.3724/SP.J.1001.2013.04420]

[7] Li Z, Harman M, Hierons RM. Search algorithms for regression test case prioritization. IEEE Trans. on Software Engineering, 2007, 33:225−237. [doi: 10.1109/TSE.2007.38]

[8] Hla KHS, Choi YS, Park JS. Applying particle swarm optimization to prioritizing test cases for embedded real time software retesting. In: Proc. of the IEEE 8th Int'l Conf. on Computer and Information Technology Workshops (CIT Workshops 2008). IEEE, 2008. 527−532. [doi: 10.1109/CIT.2008.Workshops.104]

[9] Singh Y, Kaur A, Suri B. Test case prioritization using ant colony optimization. ACM SIGSOFT Software Engineering Notes, 2010, 35(4):1−7. [doi: 10.1145/1811226.1811238]

[10] McMinn P. Search-Based software testing: Past, present and future. In: Proc. of the IEEE 4th Int'l Conf. on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2011. 153−163. [doi: 10.1109/ICSTW.2011.100]

[11] Harman M, McMinn P, de Souza JT, Yoo S. Search based software engineering: Techniques, taxonomy, tutorial. In: Empirical Software Engineering and Verification. LNCS 7007, 2012. 1−59. http://www0.cs.ucl.ac.uk/staff/mhaman/laser.pdf

[12] Srinivas N, Deb K. Multi-Objective function optimization using non-dominated sorting genetic algorithms. IEEE Trans. on Evolutionary Computation, 1994,2(3):221−248. [doi: 10.1162/evco.1994.2.3.221]

[13] Li H, Zhang Q. Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II. IEEE Trans. on Evolutionary Computation, 2009,13(2):284−302. [doi: 10.1109/TEVC.2008.925798]

[14] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast elitist multi-objective genetic algorithm: NSGA-II. IEEE Trans. on Evolutionary Computation, 2000,1917:849−858. [doi: 10.1109/4235.996017]


© 中国科学院软件研究所 http://www.jos.org.cn

[15] Nucci DD, Panichella A, Zaidman A, Lucia AD. Hypervolume-Based search for test case prioritization. Lecture Notes in Computer Science, 2015,9275:157−172. [doi: 10.1007/978-3-319-22183-0_11]

[16] Yuan F, Bian Y, Li Z, Zhao R. Epistatic genetic algorithm for test case prioritization. In: Proc. of the Search-Based Software Engineering. Springer-Verlag, 2015. 109−124. [doi: 10.1007/978-3-319-22183-0_8]

[17] Srinivas M, Patnaik LM. Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Trans. on Systems, Man and Cybernetics, 1994,24(4):656−667. [doi: 10.1109/21.286385]

[18] Yoo S, Harman M, Ur S. Highly scalable multi objective test suite minimisation using graphics cards. In: Proc. of the Search Based Software Engineering. Springer-Verlag, 2011. 219−236. [doi: 10.1007/978-3-642-23716-4_20]

[19] Epitropakis MG, Yoo S, Harman M, Burke EK. Empirical evaluation of Pareto efficient multi-objective regression test case prioritization. In: Proc. of the 2015 Int'l Symp. on Software Testing and Analysis. ACM, 2015. 234−245.

[20] Walcott KR, Soffa ML, Kapfhammer GM, Roos RS. Timeaware test suite prioritization. In: Proc. of the 2006 Int'l Symp. on Software Testing and Analysis. ACM Press, 2006. 1−12. [doi: 10.1145/1146238.1146240]

**附中文参考文献**:

[2] 章晓芳,徐宝文,聂长海,史亮.一种基于测试需求约简的测试用例集优化方法.软件学报,2007,18(4):821−831. http://www.jos.org.cn/1000-9825/20070404.htm

[3] 丁革建,郑燕妮,张璐.基于蚁群算法的测试用例集最小化研究.计算机工程,2009,35(6):213−215.

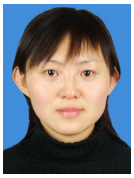
[6] 陈翔,陈继红,鞠小林,顾庆.回归测试中的测试用例优先排序技术述评.软件学报,2013,24(8):1695−1712. http://www.jos.org.cn/1000-9825/4420.htm [doi: 10.3724/SP.J.1001.2013.04420]

**边毅**(1986－),男,宁夏银川人,博士,CCF 学生会员,主要研究领域为基于搜索的软件回归测试.

**李征**(1974－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为基于搜索的软件工程,程序源代码分析.

**袁方**(1990－),男,硕士,主要研究领域为演化计算,软件回归测试.

**赵瑞莲**(1964－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件测试,软件可靠性分析.



**郭俊霞**(1977－),女,博士,讲师,CCF 高级会员,主要研究领域为网络信息定向抽取技术及测试.

© 中国科学院软件研究所 http://www.jos.org.cn