第43卷 第12期 计 算 机 工 程 2017年12月
Vol.43 No.12 Computer Engineering December 2017

·体系结构与软件技术· 文章编号: 1000-3428(2017)12-0055-05 文献标志码: A 中图分类号: TP311

基于程序变异分析的软件错误定位

王 琦,孙文辉

(北京交通大学 计算机与信息技术学院,北京 100044)

摘 要: 基于覆盖的错误定位(CBFL)方法通过获取成功和失败测试用例的覆盖信息和执行结果对程序中的错误进行定位,但该方法未考虑偶然性成功测试用例的影响,降低了错误定位的准确率。为此,提出一种新的软件错误定位方法,通过分析程序变异减少偶然性成功测试用例的影响,改进怀疑度计算公式,并加入对变异影响的计算。实验结果表明,与传统CBFL方法相比,该方法能够有效提高错误定位的准确率。

关键词: 错误定位;程序变异;成功测试用例;代码覆盖;自动化测试

中文引用格式: 王 琦,孙文辉. 基于程序变异分析的软件错误定位[J]. 计算机工程,2017,43(12):55-59.

英文引用格式: WANG Qi, SUN Wenhui. Software Fault Localization Based on Program Mutation Analysis [J]. Computer Engineering, 2017, 43(12): 55-59.

Software Fault Localization Based on Program Mutation Analysis

WANG Qi, SUN Wenhui

(College of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China)

【Abstract】 Coverage-Based Fault Localization (CBFL) method can locate the fault by analyzing the information and results of success and failure test cases. However, CBFL ignores the impact of accidental successful test cases, and the existence of accidental successful test cases will reduce the accuracy of the fault location. Aiming at this problem, this paper presents a new fault localization method. It reduces the influence of accidental success test cases based on program mutation anatysis, improves the doubt degree calculation formula and adds calculation of the influence of variation. Experimental results show that this method can significantly improve the accuracy of fault localization compared with the traditional CBFL method.

【Key words】 fault localization; program mutation; successful test case; code coverage; automated test

DOI: 10.3969/j.issn.1000-3428.2017.12.010

0 概述

软件已经成为现代信息社会的重要组成部分,人类的日常生活越来越离不开计算机软件的支持。软件测试是为显示程序中存在的错误而执行程序的过程,是软件生命周期中的重要环节。软件调试则是找到错误并修复错误的过程。在软件开发与维护过程中,软件调试是一项成本昂贵且复杂耗时的工作,约占软件开发与维护过程代价的50%~80%[1]。软件调试是保证软件质量的重要手段。迄今为止,软件调试仍是软件工程领域的难题之一。其中,软件错误定位旨在自动检测出软件存在的缺陷,是提高软件调试效率最有效的手段之一。

近年来,软件错误定位问题深受工业界和学术界的广泛关注[2],其中,基于覆盖的错误定位(Coverage-Based Fault Localization, CBFL)方法是一种被广泛应用的错误定位方法[3]。基于覆盖的错误定位一般通过对比执行失败和执行成功的测试用例的覆盖信息(如语句覆盖信息、谓词覆盖信息[4]、方法覆盖信息和分支覆盖信息等)或者执行轨迹来定位错误。然而,CBFL忽略了偶然性成功测试用例对错误定位结果的负面影响。偶然性成功测试用例执行了错误的代码,但却没有引发失败的测试结果[5-6]。针对该问题,本文提出一种基于程序变异分析的软件错误定位方法。首先通过变异候选语句得到变异体集合,然后对所有的变异体执行测试用例得到变异语句的变异影响,最后将变异影响引入怀疑度计算公式,得到更精确的错误定位结果。

1 相关研究

1.1 基于覆盖的错误定位方法

基于覆盖的错误定位(CBFL)方法是一种利用程序覆盖信息来定位错误语句的错误定位技术,其

作者简介:王 琦(1992—),男,硕士研究生,主研方向为软件自动化测试;孙文辉,副教授。

收稿日期:2016-08-01 修回日期:2016-11-06 E-mail: qi@bjtu.edu.cn

过程一般分为 2 个阶段: 首先通过执行程序来收集每个测试用例的执行轨迹(覆盖信息) 和执行结果(实际输出与预期输出是否相同) ,以评估各个语句的怀疑度; 然后按照怀疑度的大小将各个程序语句依次排序,怀疑度最大的语句为错误语句的可能性越大。

为计算每个语句的怀疑度,研究者已经提出了很多种计算怀疑度的方法。最为经典的相似度系数方法 Tarantula [7] 。Tarantula 方法的怀疑度计算公式如式(1) 所示。

$$sus(S_i) = \frac{N_f(s)/N_f}{N_p(s)/N_p + N_f(s)/N_f} \tag{1}$$

式中: N_p 为成功测试用例的总数; N_f 为失败测试用例的总数; $N_p(S)$ 为覆盖语句 S 的成功测试用例数; $N_f(S)$ 为覆盖语句 S 的失败测试用例数; $sus(S_i)$ 为语句 S 的怀疑度,是一个 0 到 1 之间的概率值,表示语句 S 为错误语句的可能性大小。

1.2 变异测试

变异测试是一种基于程序错误的测试技术,这种测试技术主要用来评估测试用例集发现程序错误的有效性 [8] 。程序变异是指,选定一个源程序 P,按照规定的方法,对 P 的某一处源代码的进行符合语法的轻微变更,从而获得一个新的程序 P',称程序 P'为源程序的变异体。在变异测试中,错误会被人为地引入源程序,从而生成一系列程序包含错误的版本,这样每个版本中都只包含一个简单的错误。以下所示为源程序 P 的一个简单变异: 对语句源程序 P 中的 if($a>0$ && $b>0$) 语句进行变异后生成包含一个错误的变异体 P'。

源程序 P:

```
function( int a,int b) {
    if( a >0&&b >0)
      return 1;
    else
      return 0;
}
```

变异体 P':

```
function( int a,int b) {
    if( a >0||b >0) //mutant
      return 1;
    else
      return 0;
}
```

程序变异的基础是变异算子集,变异算子是一种变体产生式装置,也可以理解为应用于变异的操作运算符,源程序代码通过变异算子可以嵌入变异因子,实现轻微的程序变更产生变异体 [9] 。以变异对象的类型来区分,变异算子大致可分为语句、运算符、变量、常量 4 种常见类型 [10] 。上述示例所采用变异算子为运算符变异算子。

1.3 偶然性成功测试用例

文献 [11] 提出了偶然正确性的概念。偶然正确性是指测试用例执行过程中可能会出现以下情况: 程序因为执行了错误进入了异常的状态,但是程序的输出与预期输出一致,而导致这种情况的测试用例叫做偶然性成功测试用例。偶然性成功测试用例,就是指那些执行了包含缺陷语句导致程序进入异常状态,但是没有检测到错误的测试用例。研究表明这种测试用例在实际测试中广泛存在,因为 CBFL 方法的怀疑度值与成功测试用例所占比例密切相关,所以偶然性成功测试用例的存在会显著降低错误定位的准确率。

2 基于程序变异的错误定位

为减少偶然性成功测试用例的影响来提高错误定位的准确率,本文提出了一种基于程序变异分析的错误定位方法 TarantulaM,其在 Tarantula 方法的基础上引入变异分析,以降低偶然性成功测试用例的负面影响。

2.1 设计思想

基于程序变异的错误定位的核心思想是变异程序中不同语句后引发的执行结果的改变,为错误定位提供新的线索。该设计思想是因为笔者在实验中发现了存在这样的情况: 对于一个存在错误的程序,将错误代码变异后,即把原错误代码按规则替换成不同的代码,然后执行测试用例集,发现测试结果和原程序的测试结果趋于相同,因为程序中并没有增加新的错误,只是把原错误换成了一个新的错误,所以对每个测试用例的结果并没有很大的影响; 然而,假如将程序中的正确代码进行变异,再执行测试用例集,发现测试结果会有显著的改变,原先成功的测试用例很可能变为失败,这是因为相比原程序变异后的程序多了一个新的错误,所以就明显影响了测试用例的结果。因此,笔者认为变异程序中不同代码段后所引发的由成功变为失败的测试用例数为程序错误定位提供了新的线索。与基于代码覆盖的错误定位相比,偶然性成功测试用例对这种定位线索不会造成负面影响,用它来修正 CBFL 技术所计算出的结果,可有效减少偶然性成功测试用例的负面影响,提高错误定位的准确率。

2.2 怀疑度计算公式

基于上述思想,在程序变异的基础上,本文统计了每个代码段变异后原成功测试用例执行失败的平均数,并将此作为对语句 Si 的变异影响 $Effect(S_i)$。$Effect(S_i)$ 的计算公式如式(2) 所示。

$$Effect(S_i) = \frac{\sum_{j=1}^{k} Changep \rightarrow f(Ms_i, j)}{k} \tag{2}$$

式中: k 是每条语句的变异个数; $Changep \rightarrow f(Ms_i, j)$ 是第 S_i 条语句的第 j 个变异程序执行后成功测试用例的变化数。

在计算错误怀疑度值时引入变异影响可以有效提升 CBFL 技术的准确率。因此,本文在经典的基于覆盖的错误定位方法 Tarantula 的基础上加入变异影响影响因子,设计了 TarantulaM 方法的怀疑度计算公式,如式(3)所示。

$$sus(S_i) = \frac{N_f(S_i)/N_f}{N_p(S_i)/N_p + N_f(S_i)/N_f + Effect(S_i)/N_p(S_i)} \quad (3)$$

其中,各变量与式(1)中的变量含义相同。

3　错误定位的自动化实现

为验证 TarantulaM 方法的性能,针对错误定位中广泛使用的标准评估数据集 Siemens 套件的结构,本文在 Linux 系统下实现了错误定位的自动化。

3.1　框架结构

基于覆盖信息与程序变异分析的设计思想,设计支持本文算法的程序错误自动化定位系统框架。如图 1 所示,该系统的输入是程序的源代码和测试用例集,输出是错误语句在可疑语句列表中的位置。根据框架结构,工作过程如下: 1) 给定一个错误源程序和其对应的测试用例集,首先对源程序进行插装编译,对生成的可执行文件运行所有测试用例,收集覆盖信息和测试用例执行结果; 2) 根据测试用例覆盖信息以及变异语句挑选规则选出变异候选语句; 3) 对挑选出来的变异候选语句,应用所有类型变异算子来生成变异程序,然后从中随机挑选出一定数量的变异体样本; 4) 对每个被挑选出的变异程序执行所有的测试用例,收集成功测试用例执行失败的变化数,计算每个语句的变异影响; 5) 基于原错误程序的覆盖信息及执行结果,加入变异影响,计算每个语句最终的错误怀疑度值。

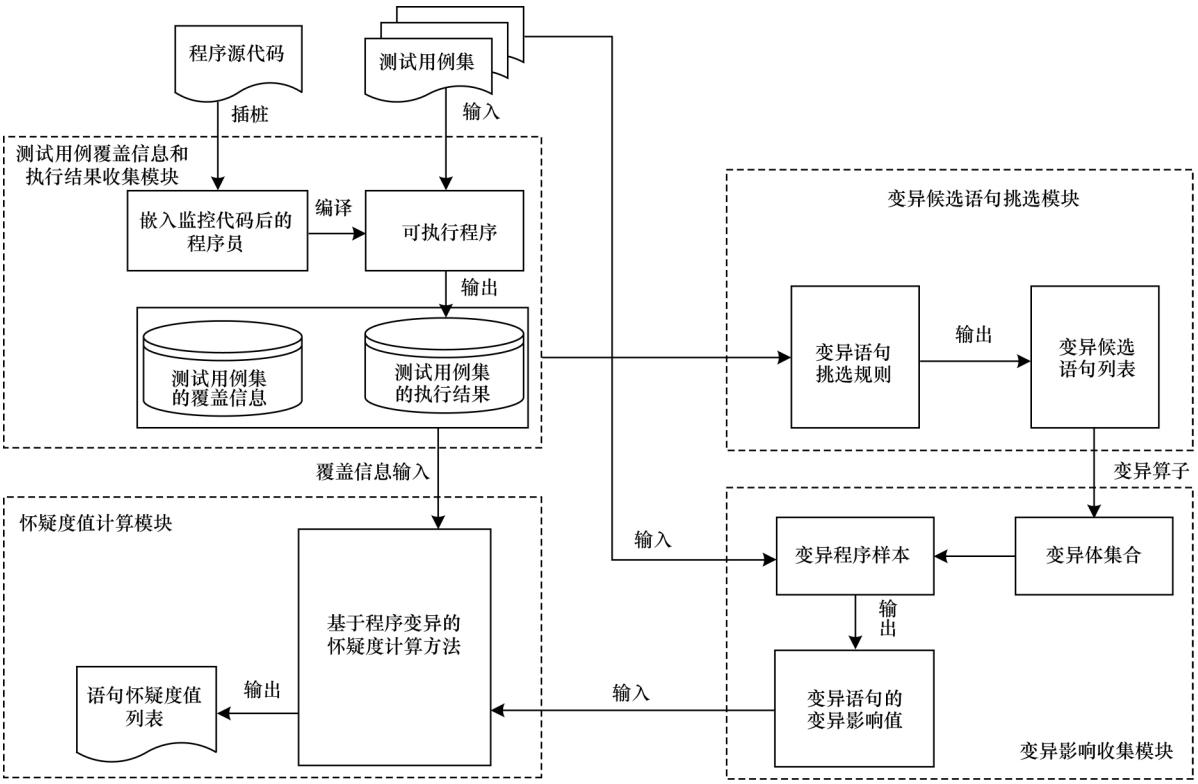

图 1　基于程序变异的错误定位方法框架

3.2　自动化脚本实现

本文采用脚本语言实现自动化。运行环境为 Linux Ubuntu 14.10,使用了 gcc 编译器,gcov 插装工具以及 Proteum/IM2.0 [12] 程序变异工具。命令执行、运行调度使用 shell 脚本编写,变异语句挑选算法和怀疑度值计算算法使用 c 语言编写。

实现自动化的脚本伪代码描述如下:

```
Input( sourcePro,testCase) ; //输入源程序和测试用例集
Gcc( sourcePro) ; //源程序编译插装
Run( testCase) ; //执行测试用例
Gather( ) ; //获取覆盖信息和执行结果
Select( cover,results) ; //挑选变异语句
Proteum( ) ; //执行变异
Effect( s) ; //计算语句变异影响
Suspect( s) ; //语句怀疑度值计算
```

3.2.1　变异候选语句挑选

在变异候选语句选择模块,为了在提高错误定位准确率的同时保证时间效率,本文的筛选策略是选取那些被所有失败用例覆盖的语句进行变异分析。

筛选语句的核心 C 代码如下:

```
void failed( void * arg) {
if( ispassed( j) ! = 1) { //判断用例是失败测试用例
    FILE * fp;
    fp = fopen( gcovname, "r") ; //读. gcov 文件
    if( fp = = NULL)
      return;
    char * buf = malloc( 1024* sizeof( char) ) ;
    int l =0;
    while( fgets( buf,1024,fp) ! = NULL) {
      if( buf [0] > =48 && buf [0] < =57) {
      //数字的 ascii 码,以判断该条语句是否被覆盖
        pthread_mutex_lock( &mut) ;
        fail [l] + +; //覆盖数 +1
        thread_mutex_unlock( &mut) ;
        }
      l + +;
     }
    fclose( fp) ;
    }
}
```

3.2.2 语句怀疑度计算

自动化的输出结果为一个错误语句怀疑度值列表文件,文件格式如下:

line < 程序行数 > : < 错误怀疑度值 >

顺序按照错误怀疑度值的大小从上到下排序。在计算语句怀疑度值时,可以替换怀疑度计算公式实现其他 CBFL 方法的计算,方便比较分析。实现语句怀疑度值计算的核心 C 代码如下:

```
void suspect( ) { //怀疑度计算
int i,j;
for( i =0; i < linenum; i + + ) { //计算可疑度
    ranked [i] . line = i -4;
    if( fail [i] = =0 && pass [i] = =0)
      ranked [i] . p = -1; //无则赋值为 -1
    else{
      ranked [i] . p = ( ( float) fail [i] /failnum) /( ( float) fail [i] /failnum + ( float) pass [i] /passnum + effect( i) /( float) pass [i]) ;
      //TarantulaM 方法,也可改变公式实现其他算法
      }
  }
  sort( ranked [linenum]) ; //排序
}
```

4 实验与结果分析

为了验证本文提出的基于程序变异的错误定位方法的效果,实验部分使用了错误定位中常用的评测基准集 Siemens Suite[13],它包含了 7 组实现不同功能的 C 程序以及相应的错误版本。所有程序及其错误版本均从软件基础设施库的网站下载[14]。同时完成 Tarantula 方法的错误定位,方便对定位结果比较分析。

对基于程序变异的错误定位方法和 Tarantula 方法进行全面比较。图 2 描述了测试的所有错误版本中是否能够检查到错误语句的情况,其中,纵坐标是当根据怀疑度值结果检查少于某个比例的代码时可以检测到程序错误的错误版本数占总数的比例; 横坐标是当按错误语句怀疑度值来检查语句时检测到错误语句时所检测的代码比例。

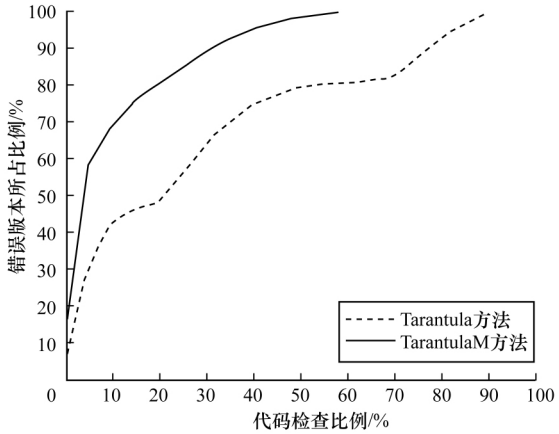

图 2 Tarantula 和 TarantulaM 方法的实验结果比较

从图 2 中可以看出: 检查 1% 的语句,Tarantula 方法可以找到约 11% 程序版本的错误,本文提出的 TarantulaM 方法可以找到约 26% 程序版本的错误; 而如果检查的语句数为 10% 时,Tarantula 方法可找到约 44% 程序版本的错误,TarantulaM 方法可找到约 70% 程序版本的错误。同时由图 2 可以发现,在相同的代码检查比时 TarantulaM 方法总比 Tarantula 方法能够定位到更多错误,并且在代码检查比超过 60% 时,就能够定位到所有错误。注意到 TarantulaM 方法和 Tarantula 方法的唯一区别在于 TarantulaM 方法在计算中加入了变异影响,减少了偶然性成功测试用例对测试结果的影响。相比传统的基于覆盖的错误定位方法,加入变异影响的 TarantulaM 方法可以显著提高错误定位的准确率。

5 结束语

降低偶然性成功测试用例的影响是提高错误定位精度的关键问题。为此,本文通过分析程序变异增加新的定位线索,以减少偶然性成功测试用例的负面影响,同时改进原有的怀疑度计算公式,加入变异影响的计算,提出基于程序变异的错误定位方法并给出自动化脚本,使其同样适用于其他 CBFL 方法。实验结果表明,相比于 Tarantula 方法,本文方法能够提高错误定位的准确率,但目前还不适用于多错误的定位[15],下一步将对此进行改进。

参考文献

[1] COLLOFELLO J S, WOODFIELD S N. Evaluating the Effectiveness of Reliability-assurance Techniques [J]. Journal of Systems & Software, 1989, 9(3): 191-195.

[2] 郝　鹏,郑　征,张震宇,等. 基于谓词执行信息分析的自适应缺陷定位算法 [J]. 计算机学报, 2014, 37(3): 500-511.

[3] SHIN Y. Evolving Human Competitive Spectra-based Fault Localisation Techniques [C]//Proceedings of the 4th International Conference on Search Based Software Engineering. New York, USA: ACM Press, 2012: 244-258.

[4] 辛　良,姜淑娟. 基于关键谓词的程序错误定位方法 [J]. 计算机工程, 2010, 36(14): 54-55, 58.

[5] ARTZI S, DOLBY J, TIP F, et al. Practical Fault Localization for Dynamic Web Applications [C]// Proceedings of the 32th International Conference on Software Engineering. New York, USA: ACM Press, 2010: 265-274.

[6] 贺　韬,王欣明,周晓聪,等. 一种基于程序变异的软件错误定位技术 [J]. 计算机学报, 2013, 36(11): 2236-2244.

[7] JONES J A. Empirical Evaluation of the Tarantula Automatic Fault-localization Technique [C]//Proceedings of the 20nd IEEE/ACM International Conference on Automated Software Engineering. Washington D. C., USA: IEEE Press, 2005: 273-282.

[8] DEMILLO R A, LIPTON R J, SAYWARD F G. Hints on Test Data Selection: Help for the Practicing Programmer [J]. Computer, 1978, 11(4): 34-41.

[9] 陈　翔,顾　庆. 变异测试: 原理、优化和应用 [J]. 计算机科学与探索, 2012, 6(12): 1057-1075.

[10] AGRAWAL H, DEMILLO R A, HATHAWAY B, et al. Design of Mutant Operators for the C Programming Language: SERC-TR-41-P [R]. West LafayeRe, USA: Department of Computer Science, Purdue University, 2006.

[11] BUDD T A, ANGLUIN D. Two Notions of Correctness and Their Relation to Testing [J]. Acta Informatica, 1982, 18(18): 31-45.

[12] DELAMARO M E, MALDONADO J C, VINCENZI A M R. Proteum/IM 2.0: An Integrated Mutation Testing Environment [M]//WONG W E. Mutation Testing for the New Century. Berlin, Germany: Springer, 2001: 91-101.

[13] 虞　凯,林梦香. 自动化软件错误定位技术研究进展 [J]. 计算机学报, 2011, 34(8): 1411-1422.

[14] DO H, ELBAUM S, ROTHERMEL G. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and Its Potential Impact [J]. Empirical Software Engineering, 2005, 10(4): 405-435.

[15] WONG W E, DEBROY V, LI Y, et al. Software Fault Localization Using DStar(D^*) [C]//Proceedings of the 6th IEEE International Conference on Software Security and Reliability. Washington D. C., USA: IEEE Press, 2012: 21-30.

编辑　金胡考

(上接第 54 页)

[3] HU J, MARCULESCU R. Energy/Performance-aware Mapping for Regular NoC Architectures [J]. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, 2005, 24(4): 551-562.

[4] MURALI S, de MICHELI G. Bandwidth-constrained Mapping of Cores onto NoC Architectures [C]// Proceedings of Design, Automation and Test in Europe Conference and Exhibition. Washington D. C., USA: IEEE Computer Society Press, 2004: 1-6.

[5] CHEN Guangyu, LI Feihui, SON S W, et al. Application Mapping for Chip Multiprocessors [C]//Proceedings of Design Automation Conference. New York, USA: ACM Press, 2008: 620-625.

[6] LIN L Y, WANG C Y, HUANG P J, et al. Communication-driven Task Binding for Multiprocessor with Latency Insensitive Network-on-chip [C]//Proceedings of Asia and South Pacific Design Automation Conference. New York, USA: ACM Press, 2005: 39-44.

[7] KANDEMIR M, OZTURK O, MURALIDHARA S P. Dynamic Thread and Data Mapping for NoC Based CMPs [C]//Proceedings of Design Automation Conference. New York, USA: ACM Press, 2009: 852-857.

[8] MANOLACHE S, ELES P, PENG Z. Fault and Energy-aware Communication Mapping with Guaranteed Latency for Applications Implemented on NoC [C]//Proceedings of Design Automation Conference. New York, USA: ACM Press, 2005: 266-269.

[9] YU Zhiyi, XIAO Ruijin, YOU Kaidi, et al. A 16-Core Processor with Shared-Memory and Message-Passing Communications [J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2014, 61(4): 1081-1094.

[10] 程爱莲,潘　赟,严晓浪. 基于竞争概率的片上网络带宽分配方法 [J]. 计算机工程, 2012, 38(22): 55-58.

[11] 全　励,潘　赟,丁　勇,等. 基于双维序路由策略的低能耗 NoC 网络分配 [J]. 计算机工程, 2012, 38(13): 13-16, 21.

[12] 尤凯迪,肖瑞瑾,权　衡,等. 适用于多核处理器的簇状片上网络设计 [J]. 计算机工程, 2011, 37(21): 211-213.

[13] LOIOLA E M, de ABREU N M M, BOAVENTURA-NETTO P O, et al. A Survey for the Quadratic Assignment Problem [J]. European Journal of Operational Research, 2007, 176(2): 657-690.

[14] DICK R P, RHODES D L, WOLF W. TGFF: Task Graphs for Free [C]//Proceedings of International Conference on Hardware Software Codesign. Washington D. C., USA: IEEE Computer Society Press, 1998: 97-105.

[15] COLE R. Parallel Merge Sort [J]. SIAM Journal on Computing, 1988, 17(4): 770-785.

[16] SAHU P K, CHATTOPADHYAY S. A Survey on Application Mapping Strategies for Network-on-chip Design [J]. Journal of Systems Architecture, 2013, 59(1): 60-76.

编辑　陆燕菲