HeteroFuzz: Fuzz Testing to Detect Platform Dependent Divergence for Heterogeneous Applications：工具理解

191250219
邹英龙
2021.11.30

CONTENTS

背景

随着FGPA等硬件加速器日益普遍，在资源受限的场景下（如边缘计算等应用场景），为提高程序运行效率，考虑硬件特性设计的异构应用逐渐增多。异构应用在不同硬件下的运行情况存在显著差异。

异构应用可分为两个模块：用户程序和内核程序。内核程序是考虑硬件特性进行的程序预处理。

由于异构应用需要在FGPA等硬件加速器上运行，而非CPU上运行。因此异构程序存在专属错误，这些错误往往和所使用的硬件加速器的基本特性密切相关。

背景

传统的模糊测试方法的应用有两个首要前提：1.随机生成测试用例能够测出全部的程序缺陷。2.测试用例执行速度足够快（在毫秒量级）

对于异构应用的测试而言，上述两个前提均不满足。首先，由于漏洞与异构应用所使用的基于硬件的优化方法密切相关，因此基于随机生成测试用例能够检测到硬件相关问题的概率很低。

其次，由于异构应用分析所用时间相对较长（在数小时量级）单次测试用例的执行时间很长。

因此，需要提出专门测试异构应用的模糊测试方法。

实现原理

本文所提出的模糊测试方法——HeteroFuzz是一种专门用于测试异构程序的模糊测试方法。其复用基本模糊测试流程，在此基础上进行改造。

工具的主要流程和模糊测试十分类似：1. 选择合适的突变因子。2. 选择原始的测试输入集。3. 产生新的突变数据。4. 执行突变数据。5. 根据程序运行结果反馈突变因子的选择方法。 6. 执行下一轮迭代，直至达到中止条件。

在模糊测试基本流程的基础上，结合异构程序特点，该工具同时使用三种方法提高模糊测试效率：

实现原理

1. 传统的模糊测试方法多使用代码覆盖率与内存消耗等相关性能指标作为测试质量的度量。而本工具与之不同，考虑到异构程序多使用并发句柄或一些声明语句来来结合硬件加快程序运行，而这些异构程序的bug也出于这些加速过程。因此该方法通过检测异构程序中用于加速的句柄所声明的变量。检测这些变量的取值范围以及这些变量对应的空间是否被全部使用。（如FGPA所使用的数字位数，带宽，数组长度等）。对这些信息进行监控，作为反馈信息指导新测试输入的合成。

例如：若系统检测到FPGA中int类型为8位，该方法会倾向于合成超过2^8的数据。

实现原理

2. 使用一种动态的突变因子选择方法。该方法没有让各突变因子被选择到的概率相同。而判断所得测试输入是否扩大了原先的测试范围或者与硬件特性相关。其具体算法为（对测试用例的选择）：

$$P_i = \begin{cases} 1 & \text{if i is a kernel-sensitive input;} \\ a & \text{otherwise} \end{cases} \quad (1)$$

其中，kernel-sensitive input即为测试输入与所使用的硬件特性相关，该种测试用例一定会被执行。

实现原理

2. 对突变因子的选择:

$$P_m = \begin{cases} P_m + \alpha & \text{if } m \text{ is chosen and it} \\ & \text{increases spectra} \\ P_m - \frac{\alpha}{l-1} & \text{otherwise} \end{cases} \quad (2)$$

其中，判断条件为是否扩大了已有测试的数据范围。即：为了减少冗余数据，方法有意筛选能够产生扩大测试数据范围的数据的突变因子。这种筛选方法背后的基本理念为：程序漏洞往往在边界处容易发生，对于已经测试的数据范围内部的测试用例是降低测试性能的冗余测试用例。

实现原理

3. 剪枝

与上页所阐释的基本思想相同，对于已经产生的测试用例，如果该测试用例与硬件特性无关，且该测试用例并未扩大测试范围，则该测试用例不执行，从而可以提高测试的效率。

该测试需要建立在对FGPA进行高层分析的基础上。对硬件的高层分析会保存有硬件相关加速方法的基本信息，因此原理1的处理不再额外耗费资源。

创新点

充分考虑异构应用的硬件特性，通过三种方式引导产生程序硬件相关的测试用例。由于异构应用的bug更可能在硬件加速的预处理部分产生，因此，该种工具对于异构应用具有更好的测试效果。

功能模块及数据交互——执行流程梳理

整个工具的执行流程如下：

· （该步骤简化版本不含）运行异构程序，并对该异构应用进行高层分析，生成相应的数据流图。

· 调用fuzz方法作为整体程序入口。

· 启动afl方法，执行模糊测试基本流程。

· 调用HeteroFuzz方法，重新定义突变因子，用于测试数据的生成。

· 将控制权转交回afl方法，继续进行模糊测试。

· 调用重新定义的反馈函数，用于突变因子的选择。

· 将控制权交还给afl方法

· 调用HeteroFuzz方法，进而调用filter文件，用于筛去冗余的测试用例。

· 将控制权交还给afl方法，反复迭代上述过程，直至达到预定时间或预定次数。

功能模块及数据交互——功能模块梳理

该工具完整版本共分为如下功能模块：

- 对异构应用内核代码的高层分析模块
- 模糊测试基本流程模块
- 测试数据生成模块
- 突变因子选择模块
- 测试用例筛选剪枝模块
- 其他辅助模块

对异构应用内核代码的高层分析模块

该模块对内核代码相关硬件加速做法进行分析，得到相应的控制流图。确定例如字长、FGPA数组范围等基本信息。

由于商业机密，该模块未开源。由于缺少FGPA相关硬件加速器以及所使用的异构程序。本次实现采用该工具的一个简化版本，该简化版本可以实现工具的全部功能，但不需要在FGPA上进行运行，因此也无法根据对硬件加速相关句柄进行测试突变因子筛选的反馈。因此本次实现不再关注这一模块。

模糊测试基本流程模块

该模块为前人的工作基础。属于该模块的全部文件为：afl开头的全部文件以及config、debug、hash、types等文件。这些文件实现了模糊测试从给定测试种子到筛选测试因子，从产生突变测试用例，到筛选并运行测试用例的全部过程。值得注意的是，由于最终工具的测试效果要与随机测试来比较。因此对于突变因子的筛选仍保留了最初的随机选择方法。该方法存储在randumfuzz.c中。

模糊测试基本流程模块

另外，值得说明的是，由于该种方法要保证其正确性。因此其能够生成模糊测试所能生成的全部测试用例。即为：该种方式所使用的测试用例是基于随机的测试用例的超集。

补充说明：接下来三个模块是工具自主实现的模块。由于该工具实质上是对经典模糊测试工具的改造，因此下面三个模块的调用方法均为替换该基本流程模块原有部分。

测试数据生成模块

该模块与工具原理中所说第一种方法相对应。主要存储在fuzz.cpp文件和Heterofuzz.cpp两个文件中。该方法的实现与原始的模糊测试方法十分相似。只是改变了突变因子的类型。

本文所使用的突变因子类型在下页介绍。

测试数据生成模块——所使用的突变因子类型

异构程序测试所用测试输入多为元组类型。如[1,2,3]。
本文所使用的突变因子类型共有6种：

1.数据规模的突变：如将[1,2,3]突变为[1,2,3,4]。
2.数据维度的突变：如将[[1,2,3],[3,2,1]]突变为[1,2,3]
3.某一元素的突变：如将[1,2,3]突变为[1,2,4]。
4.类型的突变：如将整数转为浮点数。
5.某一位的突变
6.某几位的突变

突变因子选择模块

该方法储存在Heterofuzz.cpp文件中。该方法进行突变因子的选择。选择的逻辑在实现原理第二点中进行了介绍。

补充说明：bubble相关方法和sort相关方法不是待测程序，而是突变因子选择时用于优先级比较所使用的冒泡排序方法。

测试用例筛选剪枝模块

该方法储存在Heterofuzz.cpp文件和filter.cpp中。该方法进行突变因子的选择。选择的逻辑在实现原理第二点中进行了介绍。

filter.cpp方法被heterofuzz.cpp调用，而后将控制权交还给afl-gcc执行模糊测试的其他流程。

其他辅助模块

程序开发所用的库和依赖。（所包含的文件将在本章节末进行梳理）

对数据交互的说明

fuzz.c为整个工具的程序入口，该工具会启动afl-gcc。通过afl-gcc调用afl系列方法，以搭建基本的模糊测试流程。在流程的执行过程中，自主实现的三个环节（自主实现的文件会在本部分末进行梳理）替换模糊测试的相应环节被afl-gcc调用。

对程序执行流程和数据交互的说明

初始种子保存在good-seeds文件夹中。生成的满足要求的测试能够用例保存在good-outputs文件夹中。执行这些测试用例分析错误类型是后续的人工处理过程，未在工具中体现出来。

在执行过程中，由突变因子产生的所有测试用例都会被保存到good-outputs文件夹中，其中不符合原理部分所述规则的测试用例会被删除。

各模块包含文件概览

高层分析模块：在简化版本中未体现。

待测程序：hello.c

输入输出：good-seeds和good-outputs文件夹

模糊测试基本流程模块：afl-系列文件、alloc-inl.h、config.h、debug.h、fuzz.cpp、hash.h、heterofuzz.cpp、test.c、test-instr.c

测试数据生成模块：heterofuzz.cpp、fuzz.cpp

突变因子选择模块：heterofuzz.cpp、fuzz.cpp

剪枝模块：filter.cpp、input-identfier.cpp、input-test.cpp

其他辅助模块：good-seeds和good-outputs外的全部文件夹、bubble.cpp、bubble.h、sort.cpp

输入输出分析

本工具是一个对异构程序进行模糊测试的工具。准确来讲，是一个有效模糊测试用例的产生工具。在good-seeds文件夹中指定模糊测试种子，反复运行程序，会在good-outputs文件夹中的得到符合要求的测试用例。对于测试用例的再次执行和分析可选用任意方法。本文中采用人工分析的方式，不再属于工具本身的范畴。因此，本工具的输入为：模糊测试种子，输出为：符合要求的测试用例。其中输入的种子保存在good-seeds中，输出的有效测试用例保存在good-outputs中。每次模糊测试的相关信息（包括测试用例名，测试用例的值，该次测试的路径覆盖的哈希）会以半结构化的形式输出到控制台。并会计算测试的总用时。调用工具时需要制定工具的调用次数。

一些个人理解

本工具的功能是有效测试用例的产生。定义了六种突变因子。工具的核心创新点是该工具充分考虑了异构程序硬件优化的特性。将硬件优化相关方法作为测试突变因子的筛选和测试用例筛选以突变因子概率计算的评价标准。因此，在其采用的三种方式中，工具的核心是第一种和第二种。而第三种方法（对测试用例的筛选）只是为了加快测试速度所进行的剪枝操作。

由此可见，模糊测试在随机的基础上，还可以进行限定和引导，使之符合不同的应用场景，发挥出更好的效果。

THANKS