计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2020.20190718
57(12): 2694-2702, 2020

基于 SMT 求解器的微处理器指令验证数据约束生成技术

谭 坚 罗巧玲 王丽一 胡夏晖 范 昊 徐 占
(江南计算技术研究所 江苏无锡 214083)
(tanjian131@163.com)

Data Constraint Generation Technology for Microprocessor Instruction Verification Based on SMT Solver

Tan Jian, Luo Qiaoling, Wang Liyi, Hu Xiahui, Fan Hao, and Xu Zhan
(*Jiangnan Institute of Computing Technology*, *Wuxi*, *Jiangsu* 214083)

Abstract During the development of the processor, it is necessary to fully verify the instructions datapaths. The existing simulation verification methods have insufficient functional coverage in terms of instruction result operands constraints, relationship constraints between operands, and internal constraints of instructions, etc. This paper proposes an instruction constraint solving method based on satisfiability modulo theory (SMT) solver. The SMT solver is introduced to convert the instruction function verification tasks into constraint satisfaction problems. Constraint satisfaction problem techniques are used to generate validation tuple data, which can be used to verify the functional correctness of the instructions set. The modeling processes and examples are given in four aspects: the instruction result operand constraints, the instruction operand constraints, the instruction internal constraints, and float-pointing instructions operand constraints. In order to improve the modeling efficiency, we propose two strategies. First, once the time threshold is reached, the current process is terminated; second, using process management and thread management technology, a parallel solution framework for instruction function constraints is implemented, and a series of serial solving tasks are assigned to multiple threads that can be executed in parallel, and the speed of solution is accelerated under the conditions of the same constraints coverage. Our experiences show that under the right circumstances, instruction constraint solving technology based on SMT provides technical support for system-level functional verification to achieve test coverage of complex scenarios.

Key words instruction function; datapaths; constraint solving; SMT solver; verification data; parallel acceleration

摘 要 处理器研制过程中需要对指令算术数据路径进行覆盖验证.针对现有模拟验证方法存在的不足,提出了一种基于可满足模理论(satisfiability modulo theory, SMT)的指令约束求解方法:利用可满足模理论求解器将指令级功能验证任务转化成数据约束求解满足问题.在结果操作数约束、操作数间约束、指令内部约束以及浮点操作数约束 4 个方面分别给出示例,并分别给出了利用 SMT 求解器进行约束建模的关键过程以及可以用于指令级功能验证的元组数据.为提高求解模型效率,提出了 2 种解决方

收稿日期:2019-10-11;**修回日期**:2020-02-13
基金项目:国家重点研发计划项目(2017YFB0202703)
This work was supported by the National Key Research and Development Program of China (2017YFB0202703).

法:首先利用时间阈值实现问题求解超时即终止的策略;其次是结合进程管理与线程管理技术,实现了指令功能约束并行求解框架,将串行求解任务分派给可并行执行的多个线程,提高了求解速度.该技术已成功应用于系统级验证中,有效提升了测试覆盖与质量,取得了很好的效益.

关键词　指令功能;数据路径;约束求解;SMT 求解器;验证数据;并行加速

中图法分类号　TP311

随着计算机体系结构理论与技术的迅速发展、制造工艺水平不断提升以及集成电路设计日趋复杂,行业内科研人员设计实现的处理器逻辑功能变得越来越复杂,对处理器测试验证质量的要求越来越高.为保证处理器功能正确性,硬件设计人员、验证人员、系统软件测试人员甚至用户都或多或少会承担着处理器功能与性能的测试验证任务.指令级功能测试是处理器正确性验证的重要组成部分,贯穿处理器研制过程中的各个阶段,在软件模拟指令验证、FPGA(field programmable gate array)实物验证、硬件模拟仿真验证以及 SoC(system on chip)硅后验证等阶段都需要进行充分验证;指令级测试方法具有通用性的特点,与具体的处理器类型无关,在专用处理器与通用处理器研制过程中都需要用到,其实现方法依赖于指令语义、内部规范以及指令上下文等.

在开展指令级功能测试的过程中,我们发现对于测试方案所要求覆盖的部分特殊场景,采用现有模拟验证方法较难达到目的.例如,当需要对单指令的结果进行各种数据类型的遍历验证时、在约定浮点操作数的中间结果为某特定数值时、在约定 2 个操作数之间满足特定关系时尤其是约定输入操作数与输出操作数之间满足特定的约束关系时,采用现有主流的模拟验证方法[1]通常显得比较低效.此外,很多的指令功能是隐含一些特定的约束关系,如加减法、乘法指令运算时存在进、借位情况发生的可能.这些内部关系对于测试人员来说不直接,部分约束甚至需要处理器设计人员协助制定.采用现有模拟方法,在制定测试验证计划或者生成测试程序的时候很难全面覆盖,即使能够实现代价也比较大.以上提及的场景都是比较常见的验证功能点,要求测试人员必须覆盖.

面对结果操作数与指令内部关系进行约束的验证场景,如果采用功能模拟测试方法,通常需要测试人员根据指令的语义模拟出该指令的相反语义,将输出转换成模拟程序的输入,反推出指令的输入操作数,形成验证元组数据,再根据验证数据生成相应的指令测试激励,验证指令功能正确性.部分指令其实是没有严格意义对应的求反过程的,例如浮点指令,假若存在浮点操作数 $A\times B=C$,但是由于舍入模式等原因,导致数据元组$\langle A,B,C\rangle$在很多时候并不满足 $C/B=A$ 的关系.

因此,对于系统测试人员而言,很多的待验证场景采用现有的功能模拟等手段是比较难以实现的;即使采用算法模拟成功实现,得到验证数据的运行时间通常较长,生成的数据量也比较有限.

国内外处理器设计研发机构如 Intel,IBM 等公司,为满足处理器特殊约束场景的验证需求,组织了大量的科学家专门设计了满足各自处理器约束建模需求的求解器[2-3].根据 IBM 公司官网显示,以色列海法实验室研究开发出了 GEC(generation core),Stocs,Mage 等专用求解器[4],并在此基础上开发出了 Genesys-Pro[5],FPgen[6]等一系列系统,很好地满足了处理器验证过程中面临的各种功能验证需求.由于这些求解器的实现与处理器内部设计细节紧密相关,因此这些专用求解器仅限在公司内部以及合作单位进行应用.

由于与设计紧密关联并且需要参考模型等底层环境和技术的支撑配合,目前现有以功能模拟为基础的系统级测试很难全面覆盖带约束的待测场景.很多带约束的待测场景对系统级测试人员来说是可见的,因此将会使系统测试在功能覆盖方面存在不足.如果不加以覆盖,将会导致经过系统测试的目标处理器仍然存在设计错误的隐患,这对处理器研制来说是不能容忍的.随着形式化技术的快速发展,我们可以在系统级测试过程中引入形式化方法,对指令约束关系描述并建模求解.在验证场景与处理器功能测试覆盖任务之间建立可靠的技术桥梁,满足工程实际需要,提升处理器验证测试质量.

本文首先对一种形式化技术方法——可满足模理论(satisfiability modulo theory, SMT)[7]进行简要介绍;然后提出了一种基于 SMT 求解器的关系约束求解技术,分别从指令结果操作数约束、操作数相互关系约束、指令内部关系约束以及浮点操作数

约束4个方面,阐述SMT技术在解决指令功能关系约束建模需求时的具体实现方法,给出了具体约束建模求解算法;之后,指出了求解过程中存在的常见问题并有针对性地提出了优化策略;最后,我们对SMT技术在指令功能验证中的应用进行了总结,对后续进一步工作进行了展望.

1 可满足模理论概况

可满足模理论(SMT)是基于布尔命题可满足问题(Boolean satisfiability problem, SAT)发展起来的一种形式化技术.其中SAT求解器能够判定给定的合取范式(conjunctive normal form, CNF)是否存在可以满足的模型,即判别是否存在满足所有给定公式都为真的变量赋值.SAT求解技术在微处理器RTL(register transfer level)级约束验证中被广泛应用.研究人员开发了很多成熟的SAT求解器;这些求解器被当作一种基础支撑的问题判定技术,已经广泛应用于人工智能领域和其他问题求解项目中.SMT技术是在SAT命题逻辑的基础上扩充了量词和项,融合了多种背景知识理论,提升了问题描述能力,扩充了应用求解范围.

SMT求解器支持用户定义变量、声明变量之间约束关系,将用户自定义的约束描述自动转换成命题逻辑甚至谓词逻辑公式,然后通过一定的搜索策略、调用底层SAT求解器进行问题判定求解;问题可满足时将给出一个满足所有约束关系的随机模型,对用户定义的变量进行随机赋值,否则求解器将反馈问题是不可满足的.

SMT有很多种具体的、被广泛使用的求解器,各个求解器之间的背景理论存在差别,因此不同求解器有各自擅长的问题求解领域.其中比较通用的是Alt-Ergo,OpenSMT,veriT,Yices,CVC3/CVC4,Z3等,其中微软公司的Z3、斯坦福大学和爱荷华州立大学联合开发的CVC4求解器尤为突出.Z3支持比特位向量、整型、实数、浮点等数据类型的变量定义与约束关系描述.支持空理论、数组、字符串、线性算术运算、非线性运算、集合运算、差分逻辑、命题逻辑、谓词逻辑等运算关系表达,甚至可以将不同领域的知识进行组合,实现复杂问题的求解[8].CVC4支持实数和整型的线性算术运算、数组、元组、归纳数据类型、比特位向量、字符串、等式与未解释函数等理论运算[9].SMT求解器强大的表达能力在实际问题的建模与求解方面有着广泛的技术前景[10-12].

本文正是利用SMT求解器强大的问题描述能力,对指令功能验证中存在比较复杂的内部约束关系进行建模.借助SMT求解器自动判断与可满足情况下随机模型生成的能力特征,生成测试验证数据,达到在处理器系统级测试任务中覆盖复杂场景的目的.

2 基于SMT求解器的指令操作数约束求解技术

系统级测试中基于约束的指令功能验证可通过对指令操作数施加约束,利用求解器建模验证是否满足给定的约定操作数,生成验证元组数据形成对应的测试激励,验证目标机器是否满足该约束对应的验证数据.本节针对模拟验证方法不便覆盖的4种情形,说明采用SMT求解器实现指令功能约束求解技术的优越性.

本文中提及的建模描述以及约束求解实验是采用Z3求解器,原因是其能够提供多种语言的支持以及API接口,同时能够支持描述浮点、整型、比特位等数据类型;提供了广泛的逻辑运算接口,能够对指令集涉及到的多种微指令行为建模提供很好的支持,同时能够以字符串的方式对求解结果进行输出,方便用户获取求解结果以及后续数据的处理.实验求解器不仅限于Z3,有资料显示CVC4最新版本[13-14]已经能够支持浮点类型数据,因此也能够实现本文中提到的建模描述实验的所有过程.

2.1 结果操作数约束

模拟验证方式对于很多指令的结果操作数不能直接、高效地进行约束求解.例如无符号乘法指令MUL R_a, R_b, R_c 实现 $R_c=R_a\times R_b$ 的功能,这里3个操作数均为无符号整型类型.如果用户需要对 R_c 进行数据类型遍历,同时限定 R_b 是大于1的整数.那么需要求解 R_a,则需要测试人员模拟出形如 $R_a=R_b$ OP R_c,然后制定相应算法实现求解.然而很多时候存在实现代价太大甚至是无法求解的情形.但这时候采用SMT求解器则很容易表达与求解,因为其背景理论支持各种算术表达式的约束求解.对于约定 R_c==111,同时要求 R_b 为大于1的整型数据,那么要求解满足条件的 R_a,用户可以采用如下伪代码约束表达方式进行描述:

① $args[0]=mk\_gt(R_b,1)$;…/* 描述 $R_b>1$ 的约束 */;

② $args[1]=mk\_mul(R_a,R_b,R_c)$;…/* 描述 $R_c=R_a\times R_b$ 的约束 */;

③ $args[2]=mk\_eq(R_c,const\_111)$; …/* 描述 $R_c\equiv 111$ 的约束 */;

④ $clause=mk\_and(args,3)$; …/* 描述同时满足上述 3 个约束 */.

其中 mk_gt,mk_mul,mk_eq,mk_and 都是 SMT 求解器算子,R_a,R_b,R_c 则是用户使用 SMT 求解器 API 声明的变量;$mk_gt(R_b,1)$描述的是 $R_b>1$ 的约束关系;$mk_mul(R_a,R_b,R_c)$描述的是 $R_c=R_a\times R_b$ 的约束关系;$mk\_eq(R_c,111)$ 描述的是 $R_c\equiv 111$ 的约束关系.然后将以上 3 个约束关系分别赋值给 $args[0]$,$args[1]$,$args[2]$子公式中,$clause=mk_and(args,3)$描述的是将上述 3 个公式进行合取并赋值给 CNF 公式 $clause$.

上述结果操作数约束伪代码采用 Z3 求解器建模描述得到 CNF 范式以及随机求解模型结果如表 1 所示:

Table 1　Result Operand Constraint CNF Formula and Solution in Z3 Solver

表 1　结果操作数约束 Z3 建模 CNF 公式以及求解模型

Meaning	Content Example
Z3 CNF Formula	*and*((*bvugt*(R_b # x0000000000000001)) (=R_c(*bvmul* R_a R_b)) (=R_c # x00000000000006f))
Z3 Solution	R_a→ # x03b0a38c0203d963 R_b→ # x51ff99244304b085 R_c→ # x000000000000006f

最后,形成的公式 $clause$ 表示需要满足所有 3 个约束关系;利用 SMT 求解器进行求解,求解器在运算过程将始终保持上述 3 个约束条件,如果有解,则系统将返回满足所有上述约束条件的一组 R_a,R_b,R_c 的随机赋值.

对于其他复杂语义的指令,可以通过增加约束变量以及借助 SMT 支持的命题逻辑运算符进行描述,实现给定约束的建模描述.

2.2　操作数之间关系约束

模拟验证的方式在对操作数之间的约束关系描述时也存在不足之处.例如对测试要求对操作数之间关系进行约束.仍然以无符号乘法 MUL R_a, R_b, R_c 为例.如对测试数据进行约束,要求遍历 R_c 的数值中 1 的个数从 0～64 的所有情况,同时要求 R_a 或者 R_b 中 1 的个数不能多于 R_c 中 1 的个数.这时采用模拟验证的方式就很难高效地实现.

相对而言,采用 SMT 求解器虽然也是比较复杂但仍然能够相对方便地进行表述与求解.

用户可以增加变量 $N_1,N_2,\cdots,N_{64}$,分别对应 R_c 中最低 n 位中 1 的个数,可以采用 4 个步骤实现约束建模:

① $args[0]=mk\_ite(extract(R_c,0,0),(N_1=1),(N_1=0))$; …/* 抽取 R_c 的第 0 位进行判断:如果 $R_c[0]$=为 1,则 N_1 赋值为 1,否则赋值为 0 */;

② $args[j]=mk\_ite(extract(R_c,i,i),(N_{i+1}=N_i+1),(N_{i+1}=N_i))$;…/* 抽取 R_c 中的第 i 位判断:如果 $R_c[i]=1$,则 N_{i+1} 赋值为 N_i+1,否则 N_{i+1}赋值为 N_i */;

③ …/* 依此类推,建立 R_c 所有的位计数约束关系,再分别建立 R_a 或者 R_b 的位计数约束关系 $N\_R_a[i]$ */;

④ 最后 $args[k]=mk\_le(N\_R_a[64],N_{64})$.

其中 mk_ite,$extract$,mk_le 是 SMT 求解器算子:mk_ite 描述的是一种 if-then-else 的约束关系;$extract$ 实现的是抽取 SMT 变量的某几个位的功能;mk_le 描述的是一种小于等于的约束关系.

$mk_ite(extract(R_c,i,i),(N_{i+1}=N_i+1),(N_{i+1}=N_i))$描述的是抽取 R_c 中的第 i 个位并进行判断:如果 $R_c[i]=1$,则 N_{i+1}赋值为 N_i+1,否则 N_{i+1}赋值为 N_i.

通过上述关系的描述,系统将合取所有公式集合,形成满足所有上述约束条件的公式,以此作为求解器的输入进行约束求解.在约束可满足时,系统将随机给出满足所有子公式约束的模型,测试人员可以根据模型形成验证数据,并由此生成覆盖特殊场景的测试激励.

上述操作数之间关系约束伪代码采用 Z3 求解器建模描述得到 CNF 范式以及随机求解模型结果如表 2 所示.为简化描述,这里仅给出满足上述条件且在 R_c 的操作数位表示法中 1 的个数为 8 的情况.

2.3　指令内部关系约束

如果测试验证人员关心计算结果中是否出现进借位的场景,采用 SMT 求解也是能够实现该场景约束的表达并求解.本节以 64 b 的无符号加法 ADDL R_a,R_b,R_c 为例进行说明.

当约定 ADDL 操作为无符号长字加法以及所有操作数的值是无法确定结果是否存在进位的,因为结果约束最多就约束了 64 b.但是计算时可能

Table 2 The CNF Formula of Relational Constraints Between Operands and Solution in Z3 Solver

表 2 操作数之间关系约束 Z3 建模 CNF 公式以及求解模型

Meaning	Content Example
Z3 CNF Formula	*and*((*ite* (=((*_extract* 0 0) R_c) #b1) (=|N[0]| #0000000000000001) (=|N[0]| #0000000000000000)) (*ite* (=((*_extract* 0 0) R_a) #b1) (=|N_R_a[0]| #0000000000000001) (=|N_R_a[0]| #0000000000000000)) (*ite* (=((*_extract* 1 1) R_c) #b1) (=|N[1]|(*bvadd*|N[0]| #0000000000000001)) (=|N[1]||N[0]|)) (*ite* (=((*_extract* 1 1) R_a) #b1) (=|N_R_a[1]|(*bvadd*|N_R_a[0]| #0000000000000001)) (=|N_R_a[1]||N_R_a[0]|)) …//与 N[1],N_R_a[1]Similar, repeat 2 to 63 (=R_c(*bvmul* R_a R_b)) (=|N[63]| #x0000000000000008) (*bvule*|N_R_a[63]||N[63]|))
Z3 Solution	R_a→#x3300000000000000 R_b→#x0000000000000005 R_c→#x*ff*00000000000000

发生 R_a+R_b 得到的有效位实际上是超过 64 b 的情形.为达到加法进位场景覆盖验证的目的,我们继续给出这种情况 SMT 求解器的约束建模过程:

① *args*[0]=*mk_eq*(R_c_64, *extract*(R_c_65, 63,0));…/* 约束 R_c_64 与 R_c_65 的最低 64 b 相同 */;

② *args*[1]=*mk_eq*(R_a_64, *extract*(R_a_65, 63,0));…/* 约束扩展源操作数最低 64 b 与原始源操作数相同 */;

③ *args*[2]=*mk_eq*(R_b_64, *extract*(R_b_65, 63,0));…/* 同上 */;

④ *args*[3]=*mk_eq*(R_c_64,*mk_add*(R_a_64, R_b_64));…/* 分别建立结果操作数 R_c 与未扩展之前的 R_a 和 R_b 之间的关系 */;

⑤ *args*[4]=*mk_eq*(R_c_65,*mk_add*(R_a_65, R_b_65));…/* 建立扩展之后的结果操作数与扩展之后的源操作数之间仍然需要保持原约束关系 */;

⑥ *args*[5]=*mk_eq*(1, *extract*(R_c_65,64, 64));…/* 获取 Rc_65 的最高位并约定其为 1,描述发生进位约束的情况 */;

⑦ *args*[6]=*mk_not*(*mk_eq*(1, *extract*(R_a_65,64, 64)));…/* 获取 R_a_65 的最高位并约定其不等于 1 */;

⑧ *args*[7]=*mk_not*(*mk_eq*(1,*extract*(R_b_65,64, 64)));…/* 获取 R_b_65 的最高位并约定其不等于 1 */.

其中 *mk_add* 是 SMT 求解器算子,描述的是一种加法运算;*mk_not* 描述的是公式求反运算,*mk_eq*,*extract* 与 2.2 节描述功能相同.

上述操作数之间关系约束伪代码采用 Z3 求解器建模描述得到 CNF 范式以及随机求解模型结果如表 3 所示:

Table 3 CNF Formula of Instruction Internal Relationship Constraint and Solution in Z3 Solver

表 3 指令内部关系约束 Z3 建模 CNF 公式以及求解模型

Meaning	Content Example
Z3 CNF Formula	*and*((=Rc_64((*_extract* 63 0) R_c_65)) (=R_a_64((*_extract* 63 0) R_a_65)) (=R_b_64((*_extract* 63 0) R_b_65)) (=R_c_64(*bvadd* R_a_64 R_b_64)) (=R_c_65(*bvadd* R_a_65 R_b_65)) (=#b1((*_extract* 64 64) R_c_65)) (*not*(=#b1((*_extract* 64 64) R_a_65))) (*not*(=#b1((*_extract* 64 64) R_b_65))))
Z3 Solution	R_a→#x8000000000000000 R_b→#x8000000000000000 R_c→#x0000000000000000

2.4 浮点操作数约束表示

浮点部件作为处理器的重要部件,再加上浮点数据组成特殊性,因此浮点运算指令的正确性验证历来都是测试验证的重点.Z3 求解器能够同时支持位、整型、浮点等数据类型,另外还支持多种舍入模式,支持半精度、单精度、双精度、128 b 浮点以及其他自定义格式的浮点运算,并且支持 IEEE-754 2008 标准,因此可使用 Z3 求解器灵活构建多种测试需求的浮点运算约束求解模型.

对浮点操作数约束通常存在 2 种方式:第 1 种是直接通过求解器接口构建不同类型的浮点数据类型,例如正负零、规格化数、非规格化数、无穷大以及非数等;第 2 种是采用位方式分别构建浮点的符号位、指数位与尾数位,然后通过拼接以上 3 部分形成需要验证的浮点数据,如首先拼接成 64 b 的长整型数据,然后采用内部接口转换成浮点数据类型,通过约束符号位、指数位、尾数位的方式实现浮点数据约束的目的.结合前述分析以及对浮点结果操作数的约束覆盖需求,采用模拟验证方式也是存在不足的.

以双精度浮点加法运算 FADDD 为例,假设测试人员需要验证舍入模式为向零舍入、浮点加法

结果操作数的尾数为 0x123456 的非规格化数且输入操作数都是规格化数的情况.我们继续以伪代码方式给出这种验证需求时采用 SMT 求解器的约束建模过程:

① $args[0]=mk\_fp\_normal(x)$;…/* 约束第 1 个输入操作数为规格化数 */;

② $args[1]=mk\_fp\_normal(y)$;…/* 约束第 2 个输入操作数为规格化数 */;

③ $args[2]=mk\_fp\_subnormal(x\_plus\_y)$;…/* 约束输出操作数为非规格化数 */;

④ $args[3]=mk\_eq(x\_plus\_y, mk\_fp\_add(mk\_fp\_rtz, x, y))$;…/* 约束输出操作数 x_plus_y 等于 x 与 y 在舍入模式为向零舍入下的浮点加法运算结果 */;

⑤ $args[4]=mk\_eq(x\_plus\_y, mk\_fpa\_to\_fp\_bv(mk\_concat(sign\_x, mk\_concat(exp\_x, mant\_x)), dp\_sort))$;…/* 约束将 $sign\_x$, exp_x, $mant\_x$ 进行拼接并形成 dp_sort 类型的浮点数据,同时进一步约束拼接之后的结果等于 x_plus_y */;

⑥ $args[5]=mk\_eq(mant\_x, mk\_unsigned\_int64(0x123456, bv\_sort(52)))$;…/* 约束尾数为操作数 $0x123456$ */;

⑦ $clause=mk\_and(6, args)$;

最后对前 6 个约束进行与运算,即同时满足以上 6 个约束,形成公式 *clause* 作为求解器的输入进行约束求解.对应的 Z3 建模过程以及求解结果如表 4 所示:

Table 4　CNF Formula of Float-Point Instruction FADDD Constraint and Solution in Z3 Solver

表 4　浮点指令 FADDD 约束的 Z3 建模 CNF 公式以及求解模型

Meaning	Content Example
Z3 CNF Formula	(and (fp.isNormal x) (fp.isNormal y) (fp.isSubnormal x_plus_y) (= x_plus_y (fp.add rm_rtz x y)) (= x_plus_y ((_ to_fp 11 53) (concat sign_x (concat exp_x mant_x)))) (= mant_x #x0000000123456))
Z3 Solution	mant_x → #x0000000123456 y → #x0008fffffffff7000 x → #x8020000000049a2b x_plus_y → #x8000000000123456

通过以上 4 种典型情况的展示,测试人员可基于该技术进一步扩展并解决绝大多数计算类指令操作数约束测试覆盖的验证求解需求.在解决约束表达与求解的问题之后,测试人员则可以将更多的时间精力放到测试场景的构建中,确保覆盖更多、更特殊情况的测试场景,提高测试针对性和测试质量,有效提升测试覆盖率.

3　求解优化技术

采用 SMT 对测试场景进行约束建模与求解的过程中,往往会遇到约束遍历求解耗时较长的问题,尤其是在涉及较多约束变量、关系较为复杂的情况下,求解时间可能会非常长.

在指令约束建模编码的实践过程中,通过 SMT 技术建立的求解程序通常是由若干个子问题的求解过程组成.而单次求解是 1 个单线程串行计算过程.当某个求解耗时很长甚至时无解时,求解器可能长时间悬挂在某一次求解的情况,导致后续求解必须等待前面的求解结束之后才能继续的问题.经过作者分析与实践,发现很多求解器都提供了超时终止求解的功能.针对该问题,SMT 用户在构建求解框架时,通过设定时间阈值,系统将在求解开始之后进行计时,若给定时间耗费完之后仍然不能判定出问题是否可满足时,系统将终止本次求解过程;转而继续后续的求解判定.这样就可以避免整个求解模型挂死在单个无效问题求解的情形.通过实现参数化框架,用户可以根据模型求解的特征,对不同的约束模型设定不同的时间阈值,保证约束求解不会因为时间阈值的限制导致求解质量明显下降.

本文作者还发现在构建遍历所有约束条件的模型求解程序中,各个子约束求解过程相互之间是松耦合的.通过约束模型求解的合理设计与组织,不同求解过程是可以拆解成相互独立的计算过程.这为采用并行化技术加速整个框架求解提供了可能.作者将耦合度不高的模块分布到不同的线程,由各个线程并行独立求解,所有线程求解结束之后将结果进行整合,形成统一的约束求解数据集合.

本文作者在实践过程中成功实现了一种基于多进程与多线程相结合的并行求解框架:顶层使用 *fork* 函数创建多个进程、在单个进程中创建多个线程的方式,实现将原本顺序、串行的遍历约束求解过程分散到 $N\times M$ 个线程中,充分利用当前服务器的计算资源进行问题求解,利用消耗空间资源换取时间,结合超时终止求解的技术,采用参数控制的形式,实现有限时间内对指令约束模型的并行求解框架.框架实现流程如图 1 所示:

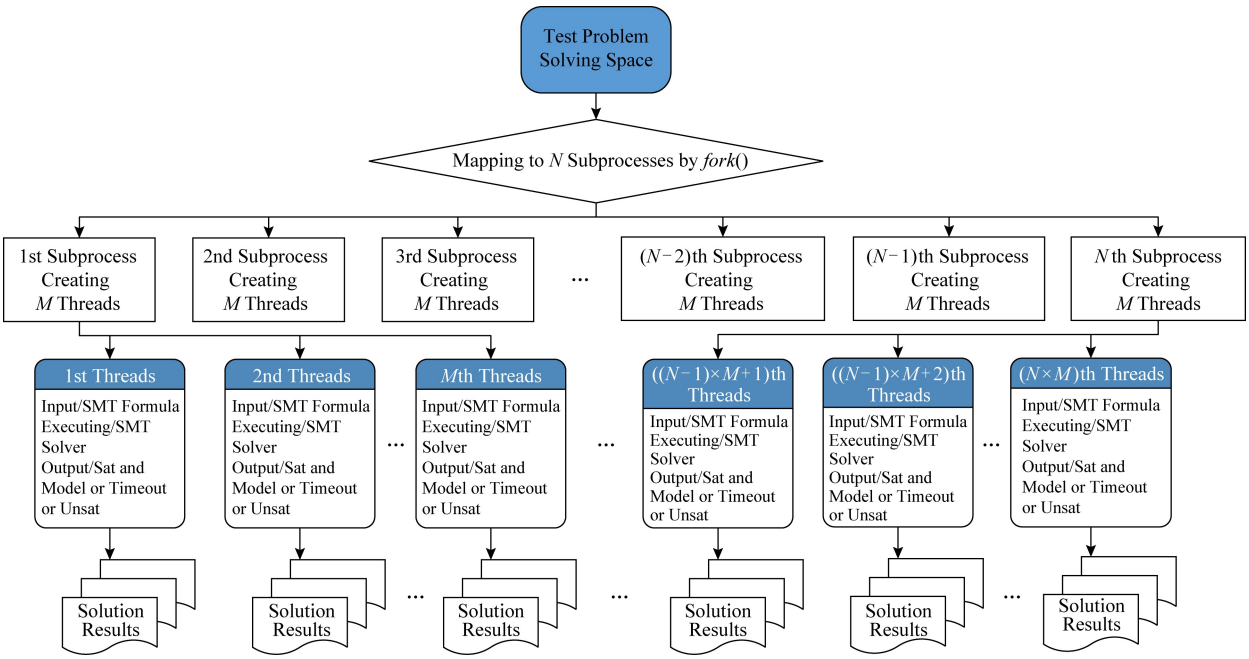

Fig. 1 Schematic diagram of parallel acceleration solution model based on *fork* and thread technologies

图 1 基于 *fork*+thread 的并行加速求解模型示意图

使用 4.8.4 版本的 Z3 求解器,在 1 台具有 2 个物理 CPU、累计 56 个逻辑核心、每个核心支持双线程的 Intel 服务器上进行实验,初始时将并行求解框架中进程数 N 与线程数 M 分别设置 8 和 4.

同时我们选择了 ADDL、MULL 以及逻辑运算指令 LOG2XX 这 3 条指令,分别对指令的约束求解模型加速前后的效果进行比较分析.其中 ADDL 模型相对简单,求解建模实现时仅有 6 个约束变量、针对 5 个约束关系形成了 5 个公式,运行过程中单次求解时间相对较短;MULL 指令复杂度次之,建模约束个数较多、求解时间较 ADDL 长.LOG2XX 指令则较为复杂,根据指令语义每种数据类型遍历的情况下存在 16 个不同的操作码,分别定义了 8 个约束变量,对约束变量建立的约束关系超过 128 个,求解时间相对较长.

此外,实验过程中将单次求解的时间阈值设置为 1 s,即 1 s 内如果约束公式不能判定是否有解则终止,转而对下一约束关系模型进行求解.如图 2 所示,每条指令均选择了 4 组数据,分别对应了单种约束情况下求解得到 1,10,100,1 000 组随机模型的情况.这里的 1,10,100,1 000 分别对指令的每一个操作数、每一种操作数数据类型进行完全遍历时在每种情况下运行的求解次数.需要说明的是,实验中设置的每种情况 1 次随机模型时,ADDL 与 MULL 对应的模型将累计产生 32 次有效求解,LOG2XX 指令将累计产生超过 480 次有效求解.3 条指令的在串、并行模式下运行求解时的加速比分别如图 2 所示:

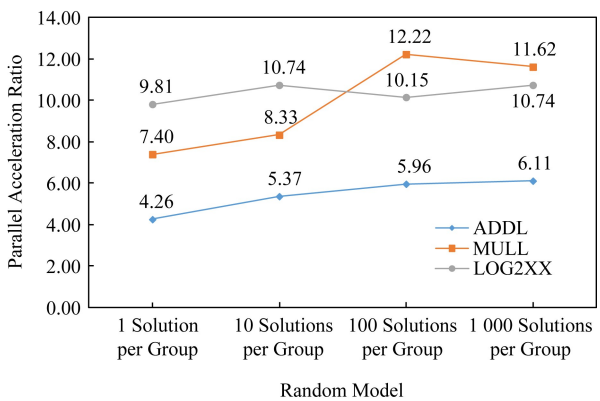

Fig. 2 Parallel acceleration diagran in 8×4 configuration

图 2 在 8×4 配置下的并行加速示意图

通过分析,我们看到在累计使用 32 个线程并行加速的情况下,ADDL 在 4 种情况下实际得到的单次有效求解加速比最高可达 6.11,MULL 指令模型最大可达 12.22,LOG2XX 最大可达 10.74.数据显示,在相同约束求解模型内,随着求解次数的增加,总体加速效果越好.尽管没有达到 $N\times M=32$ 的加速比,但是相比串行顺序求解的方式,并行求解模型的加速效果依然非常明显.

为了进一步获取更高的加速比,我们调整 N 与

M 的数值关系,分别在针对 3 种指令模型在单种约束情况下得到 1 000 组随机求解模型的前提下开展对比试验,得到的加速效果如图 3 所示.从图 3 中还可以看到,在支持双线程的服务器上,采用 16×2 模式的并行求解框架能够获得更好的加速效果.同时也可以看到,不合理的进程数与线程数设置甚至可能起不到加速的效果,并且在 3 种指令模型求解加速效果均支持这个结论.因此应根据实际的服务器配置采用不同的进程数与线程数的设置,达到更快生成指令验证元组数据的效果.

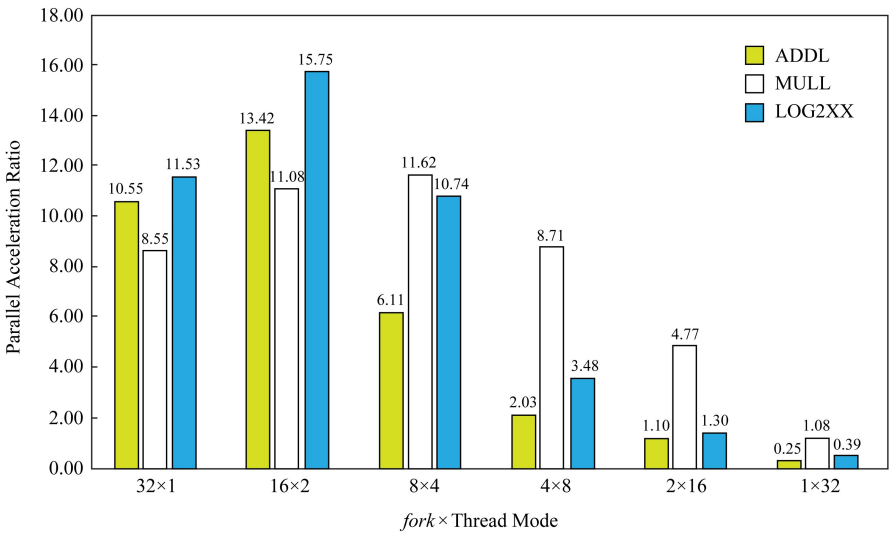

Fig. 3　Parallel acceleration diagram in multiple configurations

图 3　多种配置下的并行加速示意图

综上所述,通过比较 3 种指令约束模型的求解时间,我们看到复杂模型的求解时间较长,在约束复杂度相同的情况下,求解时间基本与求解数目成线性关系;不同模型之间,复杂度较大的模型加速效果较为明显;初步结论显示,随着约束模型的复杂度增加和求解次数增多,采用合理进程数与线程数配置的并行求解模型可得到较为理想的加速效果.

4　结论与未来工作

基于模拟验证的指令功能测试方法在结果操作数约束、操作数之间的约束、指令内部约束以及浮点操作数约束情形中存在不足,我们提出了一种基于 SMT 求解器的指令操作数约束求解技术.针对这 4 种场景,分别给出了详细的建模描述过程与求解结果示意.针对模型求解过程中存在求解时间较长的问题,采用时间阈值机制、超时则终止当前求解的策略;同时利用进程与线程管理技术,牺牲 CPU 计算资源换取时间的策略,对指令约束模型实现求解加速.实验结果表明,时间阈值策略与合理的进程加线程的并行框架提升了模型求解效率.

根据本文提出的方法实现生成的测试验证数据与程序,已成功应用于国产神威系列处理器的测试验证工作中,发现了一些隐藏较深的设计问题;同时能够支持在系统级甚至用户级层面更加深入地验证设计弱项、实现复杂场景的覆盖,如辅助特殊边界场景的异常测试验证.实践表明,采用 SMT 指令级约束求解技术基本能够实现系统级测试指令功能验证任务中各种场景的建模描述与求解的功能需求,为测试实施人员提供可靠的技术支撑.

通过与公开报道的 IBM,Intel 公司关于指令级功能与数据约束求解方面的技术特征进行比较,利用本文提出的方法与思路基本可实现指令集的功能约束、数据路径约束等主要指令级约束建模求解.同时由于借助了 SMT 强大的约束表达能力,测试人员甚至可以仅仅根据设计规范说明文档进行功能与数据路径的约束建模,弱化约束求解对设计实现的依赖,更加灵活地在处理器系统级测试验证工作中实现约束求解功能,进一步提升测试质量.

在今后的工作中,我们将继续探索 SMT 技术在复杂约束生成、指令序列约束等验证场景中的应用前景;进一步探索优化约束求解技术,充分利用现有商用处理器资源,寻找加速模型求解的有效途径;尽可能在短时间内生成更多的验证测试数据,更全面、更高效地满足处理器研制过程中的功能正确性覆盖需求,不断提升处理器系统测试的质量.

参 考 文 献

[1] Shen Haihua, Wei Wenli, Chen Yunji. A survery on coverage directed generation technology [J]. Journal of Computer-Aided Design and Computer Graphics, 2009, 21(4): 419-431 (in Chinese)
(沈海华, 卫文丽, 陈云霁. 覆盖率驱动的随机测试生成技术综述[J]. 计算机辅助设计与图形学学报, 2009, 21(4): 419-431)

[2] Kaivola R, Ghughal R, Narasimhan N, et al. Replacing testing with formal verification in Intel CoreTMi7 processor execution engine validation [G] //LNCS 5643: Proc of the 21st Int Conf on Computer Aided Verification. Berlin: Springer, 2009: 414-429

[3] Naveh Y, Rimon M, Jaeger I, et al. Constraint-based random stimuli generation for hardware verification [J]. AI Magazine, 2007, 28(3): 13-30

[4] IBM. IBM constraint solver [OL]. Haifa, Israel: IBM R&D in Israel. [2019-10-01]. https://www.research.ibm.com/haifa/dept/vst/csp_solver.shtml

[5] Adir A, Almog E, Fournier L, et al. Genesys-Pro: Innovations in test program generation for functional processor verification [J]. IEEE Design & Test of Computers, 2004, 21(2): 84-93

[6] Aharoni M, Asaf S, Fournier L, et al. FPgen—A test generation framework for datapath floating-point verification [C] //Proc of the 8th IEEE Int High-Level Design Validation and Test Workshop. Piscataway, NJ: IEEE, 2003: 17-22

[7] Moura L D, Bjorner N. Satisfiability modulo theories [J]. Communications of the ACM, 2011, 54(9): 69-77

[8] Moura L D, Bjorner N. Z3: An efficient SMT solver [C] // Proc of the 14th Int Conf on Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer, 2008: 337-340

[9] Barrettl C, Conway C L, Deters M, et al. CVC4 [G] // LNCS 6806: Proc of the 23rd Int Conf on Computer Aided Verification. Berlin: Springer, 2011: 171-177

[10] Jin Jiwei, Ma Feifei, Zhang Jian. Brief introduction to SMT solving [J]. Journal of Frontiers of Computer Science and Technology, 2015, 9(7): 769-780 (in Chinese)
(金继伟, 马菲菲, 张健. SMT 求解技术简述[J]. 计算机科学与探索, 2015, 9(7): 769-780)

[11] Wang Chong, Lü Yinrun, Chen Li, et al. Survey on development of solving methods and state-of-the-art applications of satisfiability modulo theories [J]. Journal of Computer Research and Development, 2017, 54(7): 1405-1425 (in Chinese)
(王翀, 吕荫润, 陈力, 等. SMT 求解技术的发展及最新应用研究综述[J]. 计算机研究与发展, 2017, 54(7): 1405-1425)

[12] Reynolds A, Tinelli C, Moura L D. Finding conflicting instances of quantified formulas in SMT [C] //Proc of the 14th Conf on Formal Methods in Computer-Aided Design. Piscataway, NJ: IEEE, 2014: 195-202

[13] Barrett C, Barbosa H, Brain M, et al. CVC4 at the SMT competition 2018 [EB/OL]. (2018-06-20) [2019-10-01]. https://arxiv.org/abs/1806.08775v1

[14] Brain M, Schanda F, Sun Youcheng. Building better bit-blasting for floating-point problems [G] //LNCS 11427: Proc of Int Conf on Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer, 2019: 79-98

Tan Jian, born in 1985. Master, research assistant. His main research interests include processor function verification, formalization techniques and data mining.

Luo Qiaoling, born in 1984. Master, engineer. Her main research interests include operating system analysis and data mining.

Wang Liyi, born in 1979. Master, assistant professor. Her main research interests include high performance computing, microprocessor verification, parallel computing and artificial intelligence.

Hu Xiahui, born in 1995. Bachelor, assistant engineer. His main research interests include microprocessor verification, automatic program generation and computer algorithm design.

Fan Hao, born in 1982. Master, research assistant. His main research interests include computer system software environment and program tuning.

Xu Zhan, born in 1982. Master, engineer. His main research interest is high performance computing.