计算机科学
COMPUTER SCIENCE

面向处理器设计的快速性能评测方法

邓林, 张瑶, 罗家豪

引用本文

邓林, 张瑶, 罗家豪. 面向处理器设计的快速性能评测方法[J]. 计算机科学, 2023, 50(11): 15-22.

DENG Lin, ZHANG Yao, LUO Jiahao. Fast Performance Evaluation Method for Processor Design[J]. Computer Science, 2023, 50(11): 15-22.

相似文章推荐（请使用火狐或 IE 浏览器查看文章）

Similar articles recommended (Please use Firefox or IE to view the article)

基于迭代轨迹划分的单分支循环程序终止性分析

Termination Analysis of Single Path Loop Programs Based on Iterative Trajectory Division

计算机科学, 2023, 50(9): 108-116. https://doi.org/10.11896/jsjkx.220700214

基于可逆数字水印的无线传感器网络可恢复数据聚合协议

Recoverable Data Aggregation Protocol for Wireless Sensor Networks Based on Reversible Digital Watermarking

计算机科学, 2023, 50(8): 333-341. https://doi.org/10.11896/jsjkx.220800089

基于 Coq 的逆矩阵运算的形式化

Formalization of Inverse Matrix Operation Based on Coq

计算机科学, 2023, 50(6A): 220400108-7. https://doi.org/10.11896/jsjkx.220400108

RSA算法在网络数据传输中的研究进展

Research Progress of RSA Algorithm in Network Data Transmission

计算机科学, 2023, 50(6A): 220300107-7. https://doi.org/10.11896/jsjkx.220300107

基于机器学习的SCADE模型组合验证环境假设自动生成方法

Machine Learning Based Environment Assumption Automatic Generation for Compositional Verification of SCADE Models

计算机科学, 2023, 50(6): 297-306. https://doi.org/10.11896/jsjkx.220500207

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220900250

面向处理器设计的快速性能评测方法

邓 林 张 瑶 罗家豪
国防科技大学计算机学院 长沙 410073

摘 要 面对日益复杂的处理器设计和有限的设计周期,如何有效地快速进行性能评估,是每一个处理器设计团队需要解决的问题。完整的性能测试集需要运行较长的时间,特别是在硅前验证阶段,高昂的时间成本导致设计团队无法使用完整的性能测试集进行性能评估分析。文中介绍了一种通用处理器快速性能评测方法(Fast-Eval),Fast-Eval性能评测方法基于SimPoint技术,使用FastParallel-BBV方法、最优模拟点的选取以及模拟点的热迁移等方法,显著缩短了BBV生成时间和性能测试时间。实验结果表明,相比完整运行SPEC CPU 2006 REF数据规模测试程序获得的性能数据,所提方法在ARM64处理器上BBV生成时间缩短为原来的16.88%,性能评估时间缩短为原来的1.26%,性能评估结果的平均相对误差为0.53%;在FPGA开发板上测试集的平均相对误差可以达到0.40%,运行时间仅为完整运行时间的0.93%。

关键词:快速BBV生成;性能评测;SimPoint;处理器;验证
中图法分类号 TP306

Fast Performance Evaluation Method for Processor Design

DENG Lin,ZHANG Yao and LUO Jiahao
College of Computer Science and Technology,National University of Defense Technology,Changsha,410073,China

Abstract In the face of increasingly complex processor design and limited design cycles,how to efficiently and quickly perform performance evaluation is a problem faced by each processor design team. The complete performance test suite requires longer run time,especially in the pre-silicon validation phase,and the high time cost makes it impossible for the design team to use the full performance test suite for performance evaluation analysis. In this paper,a general processor Fast-Eval method based on the SimPoint technique,using the Fast Parallel-BBV method,the selection of the optimal simulation points and the thermal migration of the simulation points,significantly reduces the performance test time and BBV generation time. Experimental results show that the performance evaluation time of the ARM64 processor is reduced to 16.88% of the original,and the performance evaluation time is reduced to 1.26% of the original,and the average relative error of the performance evaluation results is 0.53%. The average relative error of the test set on the FPGA board can reach 0.40%,and the running time is only 0.93% of the full running time.

Keywords Rapid BBV generation,Performance evaluation,SimPoint,Processor,Verification

1 引言

随着高性能处理器的设计越来越复杂,设计周期越来越短,如何在有限的处理器设计周期内快速且高精度地进行性能评测成为迫切需要解决的问题。处理器的性能评测贯穿整个处理器设计周期,主要分为硅前和硅后两个阶段。硅前性能评测主要在处理器架构设计、RTL设计、功能验证、后仿验证等阶段进行。

架构设计阶段的性能评测主要依赖体系结构模拟器,利用体系结构模拟器进行芯片配置空间探索。体系结构模拟器包括行为级模拟器和周期精确模拟器两类。与真实芯片相比,行为级模拟器的运行速度慢2个数量级,周期精确模拟器的速度慢3个数量级[1]。显然,在体系结构模拟器上运行完整的性能评测测试集(如SPEC CPU等),进行芯片配置空间探索的性能测试时间是设计团队无法接受的,因此架构设计师们采用SimPoint等技术对性能评测进行加速。虽然采用SimPoint技术提取程序每个阶段的代表性程序片段(Simulation Point,SP)能有效地提高架构设计阶段的性能评测效率,但周期精确模拟器与真实芯片之间存在很大的差异,导致模拟器无法反映芯片的真实性能,性能评测误差达到了约17.94%[2]。

RTL验证阶段主要采用RTL软件模拟器[3-5]、硬件仿真器[6-7]和FPGA原型系统[8-9]进行性能评测。这些平台频率在几百KHz到几十MHz之间,运行速度很慢。与真实芯片

到稿日期:2022-09-26 返修日期:2023-03-17
基金项目:高层次科技创新人才工程人选自主科研项目(22-TDRCJH-02-006)
This work was supported by the High-level Scientific and Technological Innovation Talent Project Candidates Independent Research Projects(22-TDRCJH-02-006).
通信作者:邓林(denglin@nudt.edu.cn)

相比，软件模拟器和硬件仿真器的运行速度慢 3 个数量级[10]，FPGA 原型系统的运行速度较真实芯片慢 1～2 个数量级[11]，运行完整的性能测试集（如 SPEC CPU 2006 等）需要几周甚至几个月。Guo 等[12]提出的 Proto-Perf 方法使用动态程序分析方法和基本块聚合技术抽取测试程序的特征程序片段进行测试，可以显著缩短性能测试时间。Proto-Perf 方法利用 Valgrind[13]生成程序 BBV 的时间为完整运行程序的 20～50 倍，该阶段耗费了大量的时间，降低了性能评估的效率。同时，该研究只进行了简单的 SimPoint 模拟点抽样，没有探索最优模拟点的选取。在最终生成检查点和实际产生的检查点上虽然使用了 warmup 来减小指令统计不准确所造成的误差，但在定位检查点生成处指令的统计仍然不准确，该方法的误差为 1.53%。

在硅后验证阶段的性能评测是在流片回来的真实芯片上进行测试，运行完整的性能测试集需要 1～2 天。使用主成分分析和聚类算法可以有效提高性能测试的效率，Phansalkar 等[14]利用主成分分析和聚类算法等对 SPEC CPU 2006 基准测试的相似度和冗余度进行分析，缩减得到了一组子集程序。CINT 组件中，4 个程序的子集显示平均误差为 5.8%，最大误差为 10.1%；6 个 FP 基准的子集显示平均误差为 3.8%，最大误差为 8%。虽然该方法能快速进行性能评估，但是误差偏大。

本文基于文献[5]的思想对该文献存在的问题进行了改进，提出了一种可用于处理器设计的 RTL 验证阶段快速性能评测方法 Fast-Eval。本文研究了快速 BBV 生成技术，同时将 SimPoint 选取的 SP 与性能监控单元（Performance Monitor Unit，PMU）结合使用，通过 PMU 得到指令数对 SP 进行定位，再通过 CRIU[15]对 SP 进行 checkpoint/restore，并将 SP 移植到硬件仿真器、FPGA 原型系统上运行，快速获得高精度模拟结果。

本文的主要贡献如下：

1）提出了一种可用于处理器设计的 RTL 验证阶段快速性能评测方法 Fast-Eval。

2）Fast-Eval 方法的核心是程序的 BBV 生成，通常生成 BBV 非常耗时。针对该问题，本文提出了一种 FastParallel-BBV 技术，能够快速生成 BBV，将 BBV 的生成时间缩短为原来的 16.88%。

3）针对运行程序 SP 的加权平均 CPI 与完整程序的 CPI 之间误差大的问题，提出了一种最优模拟点选取策略，通过对比不同 SP 数量的实验数据，选取最优模拟点数量组合，可根据时间和精度需求选取不同组合进行测试。

本文第 2 章主要介绍可用于处理器设计的 RTL 验证阶段快速性能评测方法 Fast-Eval；第 3 章介绍 Fast-Eval 在真实的 ARM64 架构处理器和 FPGA 原型系统上的实验结果；最后总结全文并展望未来。

2 Fast-Eval 方法

Fast-Eval 是一种可用于处理器设计的 RTL 验证阶段快速性能评测方法。该方法一共分为 4 个步骤：FastParallel-BBV、SP 选取、定位 SP 起始位置以及 SP 热迁移。总体方法流程如图 1 所示。

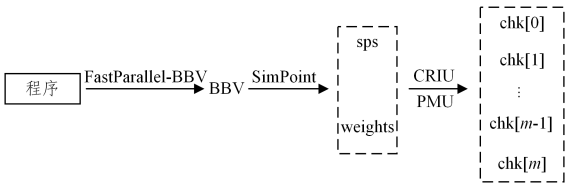

图 1 总体方法流程

Fig. 1 Overall flow of the proposed method

2.1 FastParallel-BBV 的生成

BBV 是 SimPoint 工具的输入，用于统计程序执行过程中的 BB 及它的执行次数。基本块（Basic Block，BB）是具有一个入口点和一个出口点的线性代码段。基本向量块（Basic Block Vector，BBV）是由程序执行过程中所有 BB 的列表以及每个 BB 运行次数的计数组成的向量（BB_index，BB_count）。BBV 文件格式如图 2 所示，每一行表示一个指令间隔的数据，指令间隔是在一段时间内程序运行的指令数量，每一行以一个数字和字母 T 开头，数字为指令间隔的编号，指令间隔是指程序执行过程中划分的等长片段，T 后面的是在一个间隔期间执行的每个 BB 的编号和它的执行次数计数对。本文基于 QEMU 生成 BBV，QEMU 以 TranslationBlock（TB）为单位执行程序，TB 是一个单入口单出口的代码段，其含义与 BB 相同。

```
93 T:1627:1116  :1624:2511  :1625:2232
94 T:1225:61712  :1226:92568  :1227:277704  :1261:264
95 T:1627:1240  :1624:2790  :1625:2480  :1629:3823409  :1628:30971367
96 T:1225:41880  :1226:62820  :1227:188460  :1261:264
```

图 2 BBV 文件格式

Fig. 2 BBV file format

QEMU[16]是一款能够模拟多种类型处理器的模拟器，有用户（user）和系统（system）两种运行模式。user 模式下，QEMU 不需要加载操作系统，能够直接运行应用程序；system 模式下，QEMU 能够运行完整的操作系统，模拟整个硬件平台，包括处理器及其他外围设备。为了得到性能评估测试程序进程的 BBV，需要打开 linux 内核的 PID_IN_CONTEXTIDR 配置选项，通过 QEMU 内部的 contextidr_el[1]变量准确得到目前正在执行的 TB 的 PID，进而得到该进程下的 BBV 文件。

通过 QEMU 生成完整程序的 BBV 是性能测试程序在真实芯片上运行时间的 40 倍左右[17]，本文研发了一种 FastParallel-BBV 技术，加速了 BBV 的生成。FastParallel-BBV 是一种并行化 BBV 生成技术，其过程如图 3 所示。第 1 步，在真实芯片上使用 CRIU 工具将程序分成 n 个片段；第 2 步，同时启动 n 个 QEMU 的 system 模式，在 QEMU system 模式下将 n 个程序段恢复运行，QEMU-BBV 工具就会将第 i 个程序片段转换成 1 个 BBV_i 文件，共 n 个 BBV 文件，BBV_1，…，$BBVn$；最后将得到的 n 个 BBV 文件合并成程序完整的 1 个 BBV 文件。显然，BBV_i 与 BBV_j 的生成过程之间没有相关性，可并行执行。并行生成段 BBV_i 时，BB 编号均从 0 开始，每个 BB 都有其唯一的编号以及与之对应的地址。为了将所有段的 BBV_i 合并为 1 个完整的 BBV 文件，需要将相同地址的 BB 编号修改为第 1 次出现的 BB 编号，若没有相同的地址，则分配新的 BB 编号。

FastParallel-BBV 能将 BBV 的生成过程并行化，有效节约 BBV 生成时间。假设，T_0 为使用 QEMU-BBV 生成完整

程序 BBV 所需要的时间，是完整运行程序的约 40 倍。若将程序分为 n 个片段，则 BBV 的生成时间为：

$$T=T_1+T_2+T_3 \tag{1}$$

T_1 为程序分成 n 个片段的时间。

$$T_1=\sum chk[n-1]+Tx<<T_0 \tag{2}$$

其中，T_x 是正常运行程序所需要的时间，$\sum chk[n-1]$为分块保存 checkpoint 的时间。T_2 为 n 个程序片段生成 BBV 的时间。

$$T_2=restore[n]+T_0/n \tag{3}$$

其中，$restore[n]\ll T_0$，$restore[n]$为 checkpoint 在 QEMU 上的 restore 时间。T_3 为 n 个 BBV 文件的合并时间，$T_3\ll T_0$。

当$T_1+restore[n]+T_3<T_0/n$ 时，通过 FastParallel-BBV 得到的 BBV 文件需要的时间$T=T_1+T_2+T_3\cong\frac{T_0}{n}$，可以达到加速 BBV 生成的目的。

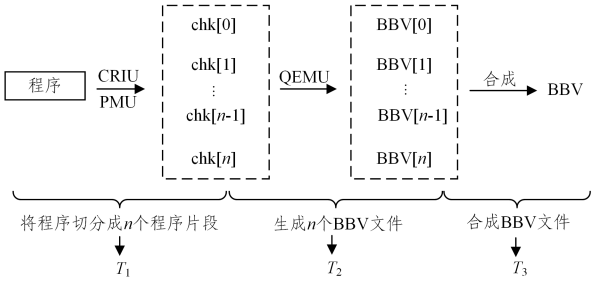

图 3　FastParallel-BBV

Fig. 3　FastParallel-BBV

2.2　SP 选取

SimPoint[18-19]是一个挑选具有代表性的 SP 进行模拟仿真的工具。如图 4 所示，该工具通过随机线性投影来降低 BBV 的维数，并使用 K-means[20]算法将得到的 BBV 划分为若干个具有相似性的聚类，从这些聚类中选出 n 个 SP 和相应的权重。通过运行这些 SP，计算 SP 的加权平均 CPI 并与完整运行程序的 CPI 进行相对误差分析，从而确定是否可以使用抽样 SP 的执行来代表完整程序的执行过程。

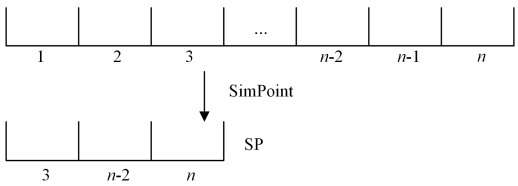

图 4　SimPoint 挑选 SP

Fig. 4　SimPoint chooses SP

SimPoint 的参数 k 可以设置 SP 生成的个数，其中 max k 用来设置生成 SP 个数的最大值。通过 SimPoint 生成 SP 的个数过少则无法很好地推断出程序的总体行为，SP 的个数过多又无法达到缩短程序运行时间的目的。本文进行了最优 SP 个数的选取，如图 5 所示。第 1 步，在设置 SP 最大生成个数的前提下，使用 SimPoint 对不同间隔大小的 BBV 文件自动生成 SP；第 2 步，分析自动生成 SP 的 CPI 与完整运行程序的 CPI 之间的相对误差；第 3 步，分析生成 SP 的个数集中出现的数量范围；第 4 步，在该 SP 个数的基础上减少 SP 继续进行测试；第 5 步，重复上一步的操作直到程序的相对误差大于预想值时停止测试；最后通过对运行时间和精度这两个维度进行分析，选出最优的 SP 个数。

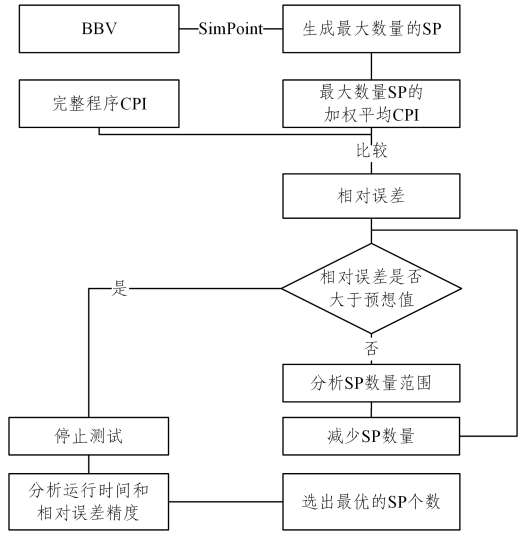

图 5　最优 SP 数量选取

Fig. 5　Selection of optimal number of SP

2.3　定位 SP 起始位置

为了得到 SimPoint 工具抽样的模拟点的 CPI，需要准确定位到模拟点的位置，对模拟点进行测试，如图 6 所示。本文定位 SP 起始位置的方法基于指令计数，即利用程序从启动至到达 SP 起始位置之间所统计的指令总数定位 SP。由于体系结构模拟器主频低，无法快速定位程序所有 SP 的起始位置，因此本文通过在真实芯片上进行定位来缩短时间。应用程序可以通过 ARM64 架构芯片的 PMU 事件计数寄存器得到测试程序执行的指令数。PMU 的 32 位事件计数器最大只能统计 2^{32} 条指令，指令数超过该值会产生计数溢出，导致指令计数不准确，从而无法精准定位 SP 的位置。

本文将两个 32 位事件计数寄存器拼接来解决指令溢出问题。假设某一个 SP 的起始位置的指令数为 1 billion，采用忙轮询的方式从 PMU 读取计数值，若指令数达到 1 billion，则立即停止程序的运行，生成该 SP 的 CRIU checkpoint 点。Linux 进程切换时间为千分之一秒，统计 1 billion 的指令数大约会有 1 million 的指令误差，误差值只占了 1 billion 的 1%，对测试结果的影响非常小。

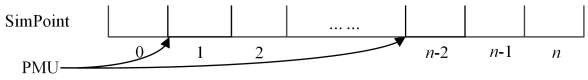

图 6　PMU 定位 SP 位置

Fig. 6　PMU locates SP position

2.4　SP 的热迁移

热迁移通常是将整个程序的运行状态进行完整的保存，同时支持将程序状态快速地恢复到原有平台上或者其他的硬件平台上，可实现程序在相同架构不同平台上的迁移。为了实现不同平台上的热迁移，被迁移的程序需要静态编译，防止运行环境不一致引起的迁移失败。本文通过 CRIU 的 restore 功能实现在其他平台上解析 checkpoint 过程产生的镜像文件，以此来恢复程序冻结前的状态，让程序从冻结前的状态继续执行。在一个 SP 运行结束时终止程序，通过 PMU 得到 instruction 和 cycle 计数，进而得到该 SP 的 CPI。

3　测试结果与分析

本文实验的软硬件环境如表 1 所列。

表 1 软硬件环境

Table 1 Hardware and software environment

名称	版本	功能参数
SPEC CPU 2006	1.2	行业标准化的 CPU 测试基准套件
Arm64 处理器	1	主频:2.5 GHz 内存容量:128 GB 核数:64 核
FPGA 开发板	1	主频:100 MHz 时钟频率:50 MHz 内存容量:8 GB 核数:单核
OS 版本	Ubuntu 18.04	内核版本 4.19.15
Qemu	4.1.0	
CRIU	3.6	
Simpoint	3.2	

本文在主频为 2.5 GHZ、内存容量为 128 GB 的 ARM64 处理器上进行快速评测,同时,在主频为 100 MHz、内存容量为 8GB 的单核 FPGA 开发板上进行验证。OS 版本为 Ubuntu 18.04,内核版本是 4.19.15。使用 SPEC CPU 2006[21] 的 REF 规模,该套件是一个获得了工业界和学术界广泛认可的专门用来测试和评价通用处理器、内存子系统和编译器性能的综合基准测试套件。该测试集包含了 12 道 int 型和 17 道 float 型程序。QEMU 的版本为 4.1.0,CRIU 的版本为 3.6,SimPoint 的版本为3.2。

图 7 给出了对 SPEC CPU 2006 测试集使用 FastParallel-BBV 并行化生成 BBV 与使用 qemu 完整生成 BBV 的时间对比结果。

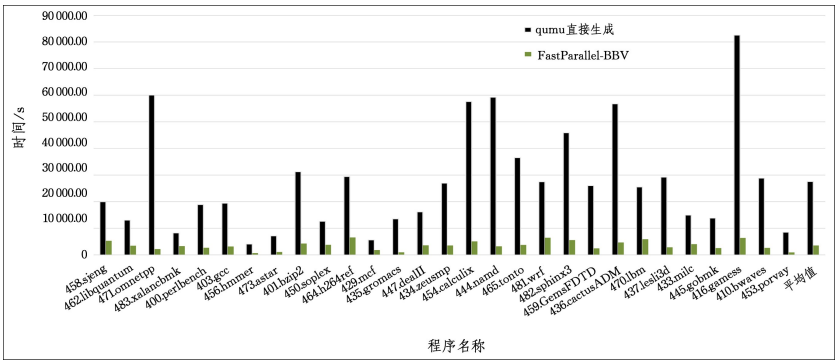

图 7 BBV 生成时间对比

Fig. 7 BBV generation time contrast

第 1 步 通过 FastParallel-BBV 并行化可以显著缩短 BBV 的生成时间。由于测试程序具有特异性,使用 FastParallel-BBV 并行化所用时间最大可缩短至 qemu 直接生成所用时间的 3.64%;最小可缩短至直接生成所用时间的 40.33%;平均所用时间是直接生成 BBV 所用时间的 16.88%。

第 2 步 根据 SimPoint 选取 SP 的方式,对第 1 步得到的 BBV 文件进行自动选取和手动选取 SP 两种测试。对于自动选取 SP,SimPoint 在设置最大 max $k=30$ 的条件下自行生成 SP。同时,为了探索最优模拟点的个数,验证是否可以使用更少的 SP 来模拟完整程序的运行,我们通过手动减少 k 值来得到相应的 SP 进行验证。

第 3 步 使用 PMU 确定每个 SP 的位置,并计算出其 CPI。

第 4 步 使用 CRIU 对每个 SP 进行备份与恢复。以 482.SPhinx3 这道程序为例,由表 2 可知,PMU 确定 SP 的精度可以达到 0.01%以下,对于每个模拟点进行 3 次测试,从中选取 PMU 确定 SP 位置最准确的一组数据(见图中灰色标注)。通过计算该组数据的指令数和周期数以及所占的权重,计算出每个 SP 的加权 CPI。

表 2 482.SPhinx3 间隔大小为 1 billion,$k=8$ 的测试结果

Table 2 Test results when 482.SPhinx3 interval size is 1 billion, $k=8$

	SP number		SP1	SP2	SP3	SP4	SP5	SP6	SP7	SP8
	SimPoint		57	113	295	349	547	719	735	795
	weight		0.13987	0.17731	0.05727	0.16410	0.08370	0.04736	0.07819	0.25220
测试次数	1	SP(inst)	57 005 531 594	113 001 342 498	295 014 091 802	349 005 549 956	547 014 282 034	719 018 096 269	735 014 713 179	795 008 023 250
		accuracy of (sp(inst))/%	0.97	0.12	0.48	0.16	0.26	0.25	0.20	0.10
		cycle	2 630 688 367	2 175 591 404	1 993 421 450	2 018 421 248	1 543 716 368	1 692 245 279	1 633 713 174	1 396 223 925
		instruction	1 001 506 409	990 669 912	1 018 193 346	1 004 318 226	1 008 088 113	1 008 079 501	1 010 183 334	1 015 622 515
		CPI	0.36740	0.38939	0.11212	0.32979	0.12817	0.07950	0.12646	0.34672
	2	SP(inst)	57 005 713 880	113 006 681 186	295 009 107 232	349 011 285 516	547 011 070 940	719 006 572 566	735 007 748 665	795 016 997 789
		accuracy (sp(inst))/%	1.00	0.59	0.31	0.32	0.20	0.09	0.11	0.21
		cycle	2 642 381 596	2 175 503 745	1 994 044 125	1 995 348 682	1 593 056 282	1 789 273 884	1 578 394 142	1 346 626 607
		instruction	1 008 595 738	990 685 497	1 008 386 713	1 013 237 873	1 006 367 979	1 006 265 563	1 007 876 285	1 000 098 896
		C PI	0.36643	0.38937	0.11325	0.32315	0.13250	0.08421	0.12246	0.33959
	3	SP(inst)	57 009 100 371	113 003 665 773	295 007 378 567	349 011 625 551	547 011 523 018	719 009 809 412	735 001 383 353	795 011 085 055
		accuracy of (sp(inst))/%	1.60	0.32	0.25	0.33	0.21	0.14	0.02	0.14
		cycle	2 640 372 452	2 147 791 050	1 943 674 229	1 992 370 868	1 593 031 937	1 738 904 804	1 578 900 635	1 396 523 740
		instruction	990 636 974	991 931 851	999 460 332	996 683 892	1 022 723 401	1 004 697 761	995 233 381	1 020 224 443
		CPI	0.37279	0.38393	0.11137	0.32803	0.13037	0.08196	0.12405	0.34523

通过上述方法对 SPEC CPU 2006 中的 29 道程序进行性能评估测试。

3.1　SimPoint 自动选择 SP 的个数

在指定最大模拟点个数 max $k=30$ 的条件下,间隔大小为 1billion 时,SimPoint 自动选择 SP 的测试结果如图 8 所示。29 道程序选取的 SP 与完整测试集程序的 CPI 的最大的相对误差为 7.25%,最小相对误差为 0.26%,平均相对误差为 2.89%。对于 SPEC CPU 2006 的 29 道测试程序,当间隔大小为 1 billion 时,在芯片验证阶段可以通过运行程序的一小部分的 SP 样本来推断总体程序的行为特点。完整运行 29 道程序需要的时间为 21226.35s,而使用 SP 只需要 466.80s,评估时间变成原来的 2%,达到了缩短时间的目的。

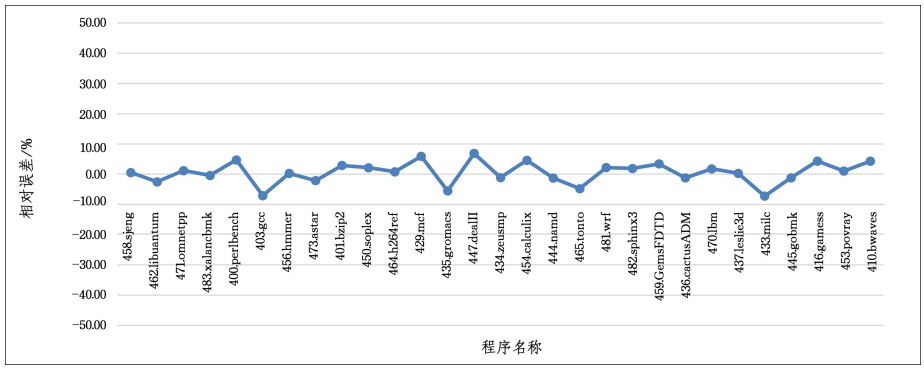

图 8　间隔大小为 1 billion 时自动选择 SP 的加权平均 CPI 与真实 CPI 的相对误差结果

Fig. 8　Relative error results between weighted average summation CPI and real CPI when the interval is 1 billion

如表 3 所列,在间隔大小为 2 billion 时,410. bwaves 的测试误差急剧增大,测试结果不理想。这是由于 SimPoint 选取 410. bwaves 在间隔大小为 2 billion 时,各个间隔的基本块的使用频率基本相似,因此无论是增加还是减少 SP 的个数,测试的 CPI 误差依旧很大。其余的 28 道程序的测试结果中,最大的相对误差为 6.42%,最小为 0.14%,平均相对误差为 2.02%。从测试数据来看,96.55%的程序当间隔大小为 2 billion 时可以使用 SimPoint 抽取的 SP 来代表完整程序的运行。

表 3　SPEC CPU 2006 间隔为 1 billion,2 billion 时的测试结果

Table 3　Test results when SPEC CPU 2006 interval size is 1 billion and 2 billion

程序名称	max $k=30$		CPI 相对误差	
	1 billion	2 billion	1 billion/%	2 billion/%
458. sjeng	$k=16$	$k=14$	0.55	0.92
462. libquantum	$k=15$	$k=8$	−2.54	−1.32
471. omnetpp	$k=7$	$k=8$	1.19	−0.45
483. xalancbmk	$k=19$	$k=18$	−0.42	−2.77
400. perlbench	$k=11,k=4,k=16$	$k=6,k=6,k=9$	4.72	0.25
403. gcc	$k=11,k=10,k=9$ $k=8,k=15,k=11$ $k=12,k=7,k=8$	$k=5,k=9,k=24$ $k=6,k=8$ $k=9,k=14,k=5,k=6$	−7.08	−1.43
456. hmmer	$k=10,k=9$	$k=14,k=7$	0.26	0.74
473. astar	$k=18,k=8$	$k=13,k=9$	−2.11	−6.23
401. bzip2	$k=11,K=22,K=22$ $k=7,k=17,k=10$	$k=11,k=4,k=9$ $k=7,k=11,k=8$	2.91	2.87
450. soplex	$K=7,k=12$	$k=12,k=7$	2.14	−0.55
464. h264ref	$k=11,k=17,k=14$	$k=7,k=12,k=17$	0.81	0.79
429. mcf	$k=10$	$k=14$	5.93	−4.16
435. gromacs	$k=20$	$k=16$	−5.50	1.24
447. dealII	$k=18$	$k=20$	6.88	−6.43
434. zeusmp	$k=24$	$k=19$	−1.12	−0.14
454. calculix	$k=11$	$k=12$	4.58	0.82
444. namd	$k=24$	$k=23$	−1.28	−2.56
465. tonto	$k=19$	$k=21$	−4.78	−0.76
481. wrf	$k=23$	$k=9$	2.18	3.58
482. sphinx3	$k=18$	$k=18$	1.90	0.94
459. GemsFDTD	$k=15$	$k=12$	3.42	6.02
436. cactusADM	$k=19$	$k=11$	−1.21	−1.51
470. lbm	$k=16$	$k=13$	1.78	1.02
437. leslie3d	$k=27$	$k=14$	0.31	−5.93
433. milc	$k=13$	$k=15$	−7.25	−0.28
445. gobmk	$k=6,k=12,k=13$ $k=9,k=12$	$k=5,k=7$ $k=7,k=6,k=9$	−1.19	1.19
416. gamess	$k=14,k=18,k=18$	$k=15,k=15,k=19$	4.35	1.05
453. povray	$k=15$	$k=15$	1.03	0.48
410. bwaves	$k=15$	$k=6$	4.31	146.69
平均值	—	—	2.89	7.00

如图 9 所示,对比 1 billion 和 2 billion 间隔可以看出,SP 的个数并没有随着间隔的增大而减少。这主要因为 SP

的选取是依据 BBV 的相似性来进行聚类划分的。从测试的准确度来看，2 billion 比 1 billion 的相对误差小 0.77%，测试结果更加准确，所选取的模拟点更能代表完整程序的行为特点。而运行效率却相反，以间隔为 1 billion 指令大小进行测试，程序的测试时间可以缩短至原来的 2%左右，而以间隔为 2 billion 指令大小进行测试，时间缩短了 3%左右。从提高效率和准确度来讲，运行 1 billion 间隔大小的 SP 效果更好。

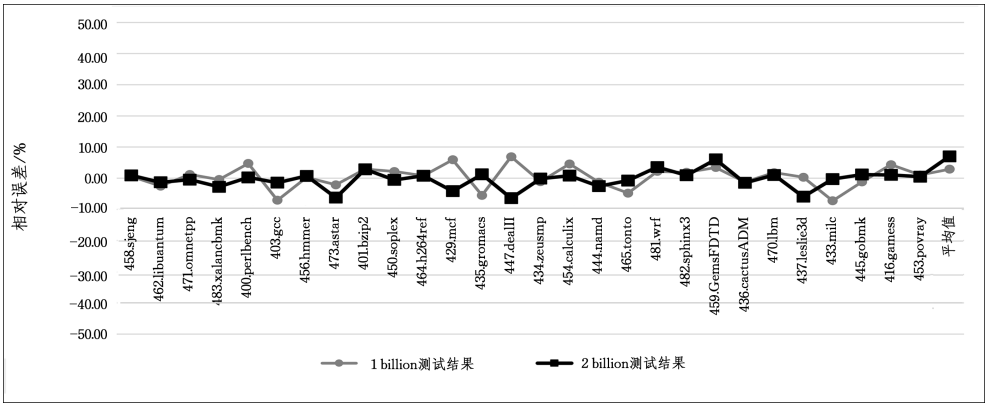

图 9 SPEC CPU 2006 间隔大小 1 billion 和 2 billion 指令自动选取 SP 的相对误差结果

Fig. 9 Relative error results when SPEC CPU 2006 automatically selects SP with interval sizes of 1billion and 2billion instructions

3.2 手动选择 SP 个数的测试结果及分析

SimPoint 自动生成 SP 的个数一般会有几个到几十个，SP 数量过多，在硬件仿真器上运行所花费的时间较长，同时也不利于进行迁移。为了探索少量的 SP 的个数是否可以实现较小的误差，本文通过手动选择 SP 的个数来探索符合要求的最少 SP 数量。根据 SimPoint 自动生成的 SP 的个数集中在 8 个左右，在 SP 为 8 的基础上对每道程序进行了 $k=8$，$k=7$，$k=6$，$k=5$ 时的测试。表 4 列出了 4 道 int 型和 6 道 float 型的程序的测试结果。

当 SP 的个数为 8 时，1billion 仍然有 3，4 道程序的相对误差结果不理想。但当间隔大小为 2 billion 时，只有 473. astar 的误差略大，为 8.84%，时间缩短到之前的 2.53%。当 SP 的个数为 7、间隔大小为 1 billion 时，483. xalancbmk，403. gcc，454. calculix 的 CPI 的相对误差较大，最大可达到 17.48%；间隔大小为 2 billion 时，473. astar，447. dealII，433. milc 这 3 道程序的相对误差大小超过 8%，最大为 37.56%。

在 $k=5$ 和 $k=6$ 时，指定选择的 SP 的个数较少，有一半以上的程序的相对误差超过了 8%。随着指定的 SP 的个数增加，大部分程序的测试结果越来越理想，同时也存在个别程序出现误差太大的现象。整体程序的测试结果表明，可以用极少数的 SP 来代表完整程序的行为。

表 4 SPEC CPU 2006 相对误差和时间测试结果

Table 4 SPEC CPU 2006 relative error and time results

程序题目	间隔大小 sp 的个数		1 billion					2 billion				
			max $k=30$	$k=5$	$k=6$	$k=7$	$k=8$	max $k=30$	$k=5$	$k=6$	$k=7$	$k=8$
程序题目	483. xalancbmk	相对误差	−0.00420	0.15995	0.15757	0.17482	0.17315	−0.02769	0.25399	0.22004	0.03729	0.04432
		时间	12.19799	7.72358	8.07201	8.43693	8.84369	35.31014	13.66914	14.39141	13.36946	14.44989
	473. astar	相对误差	−0.02108	−0.12912	−0.08064	−0.03493	−0.03232	−0.06231	0.05206	0.10578	0.08842	0.08327
		时间	21.97406	11.57571	13.61718	11.18116	11.89383	48.27402	23.37400	27.15041	34.51990	37.16922
	462. libquantum	相对误差	−0.02537	−0.00986	0.04598	−0.01102	0.00904	−0.01317	0.00998	−0.00248	−0.07888	−0.01317
		时间	19.03912	6.25558	7.77845	7.94910	8.28183	15.62246	11.53485	14.31543	15.81395	15.62246
	400. perlbench	相对误差	0.04716	0.00463	0.01238	−0.00113	−0.00297	0.97231	−0.01073	0.00836	0.01215	0.00943
		时间	9.06692	5.20424	6.31221	6.38409	7.82611	12.94966	10.30538	11.57165	14.07536	15.83067
	437. leslie3d	相对误差	0.00311	−0.06340	−0.06369	−0.05979	−0.05871	−0.05927	−0.03378	−0.06247	−0.05895	−0.04416
		时间	30.85647	5.64139	6.76851	7.93204	9.06302	31.21914	11.23977	13.45296	15.73696	17.75423
	454. calculix	相对误差	0.04583	−0.01268	−0.00221	0.14002	0.00904	0.00825	0.00532	0.00862	0.04115	0.00027
		时间	3.44893	1.55671	1.84829	2.32114	2.75288	7.17963	2.98902	3.66893	4.10484	4.83034
	436. cactusADM	相对误差	−0.01206	0.00830	−0.00617	−0.00978	0.00682	−0.01507	−0.01113	−0.00748	−0.01206	−0.01739
		时间	18.58530	4.73484	5.70042	6.69428	7.74680	21.34503	9.58025	11.59115	13.54406	15.49018
	447. dealII	相对误差	0.06882	0.20595	0.15431	0.04906	0.06905	−0.06428	−0.23096	−0.20062	−0.37564	0.03785
		时间	8.69690	3.05483	3.55069	3.93141	5.19393	16.93180	3.67348	5.33787	9.62320	8.17441
	470. lbm	相对误差	0.01782	0.00705	0.01144	0.00890	0.01559	0.01025	0.00974	0.01434	0.00124	0.02014
		时间	17.62125	5.28803	6.36527	7.60301	17.06238	28.01798	11.08854	13.49855	16.57945	17.39900
	453. povray	相对误差	0.01028	0.02395	0.02085	0.00171	0.00495	0.00478	0.00946	0.00556	−0.00338	−0.00338
		时间	5.12875	1.70599	2.05916	2.40517	2.73106	9.98518	3.43180	3.99945	4.67200	5.30499

由表 4 中的测试数据可以得出，要得到更为准确的测试程序，则选择浅灰色数据所代表的程序片段组合，其平均相对误差可以达到 0.53%，最小相对误差为 0.03%，最大相对误差为 3.78%。要快速得到程序的性能，则选择深灰色数据所

代表的程序片段组合，运行只需要 267 s，即评估时间变为原来的 1.26%，测试时间大大缩短。

由于目前 FPGA 的工作频率低，完整运行 SPEC CPU 2006 所有程序需要较高的时间成本，因此，作为验证 Fast-Eval 性能评估方法有效性的实验，本文选取 10 个程序在 FPGA 开发板上分别使用 Fast-Eval 性能评估方法和完整运行程序的方法进行测试。图 10 给出了使用 Fast-Eval 方法在间隔为 1 billion 时利用选取的最优 SP 测试得到的 CPI 与实际运行得到的 CPI 的相对误差。

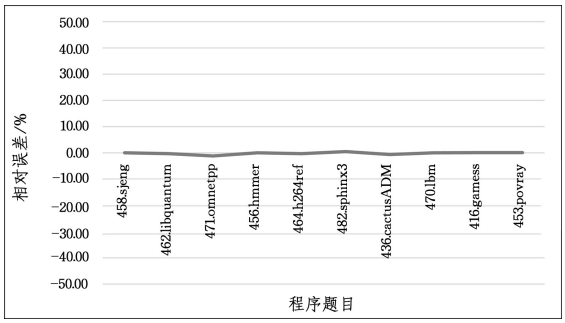

图 10　FPGA 开发板的 CPI 的相对误差

Fig. 10　CPI relative error of FPGA development board

选取的 10 道程序实验所得到的 CPI 的最大相对误差为 1.14%，最小相对误差为0.04287%，平均相对误差为0.40%。完整运行 10 道程序的时间为 162 246.05s，使用 SP 只需要 1517.66 s，评估时间变成原来的 0.93%，测试的时间显著缩短。

结束语　本文提出的基于 SimPoint 技术使用快速 BBV 生成、最优模拟点的选取以及模拟点的热迁移的 Fast-Eval 方法，测试结果表明，本文方法显著地缩短了 BBV 生成时间和性能测试时间。通过该方法，SPEC CPU 2006 测试集测试在 ARM64 处理器上获取 BBV 的时间相比原来缩短 16.88%，测试集的 CPI 的平均相对误差可以达到0.53%，运行时间仅为完整运行的 1.26%。在 FPGA 开发板上测试集的平均相对误差可以达到 0.40%，运行时间仅为完整运行时间的 0.93%。Fast-Eval 性能评估方法能够在有限的设计周期内，快速有效地进行性能评估，对硬件仿真器测试加速有着重要的意义，但目前仍然存在一些需要改进的地方。

1）CRIU 的 checkpoint 功能产生的镜像文件太大，大小在几百兆到几千兆左右，占用的内存空间太大，不利于相同处理器架构在不同平台上进行迁移。未来的工作可以研究将 SP 做成一个裸机的 ELF 程序，这样需要迁移的程序就会小很多。

2）Fast-Eval 方法是基于操作系统实现的，CRIU 恢复程序时根据程序的 PID 进行恢复运行，如果在恢复程序运行时程序的 PID 被占用，则无法恢复程序运行。未来的研究工作可以尝试解决操作系统下程序 PID 冲突的问题。

参 考 文 献

[1] ZHANG Q L，HOU R，YANG S B，et al. The role of architecture simulator in processor design process[J]. Computer Research and Development，2019，56(12)：2702-2719.

[2] BUTKO A，GARIBOTTI R，OST L，et al. Accuracy evaluation of gem5 simulator system[C]// 7th International Workshop on Reconfigurable and Communication-centric Systems-on-chip(ReCoSoC). IEEE，2012：1-7.

[3] HEIRMAN W，CARLSON T，EECKHOUT L. Sniper：Scalable and accurate parallel multi-core simulation[C]// 8th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES-2012). High-Performance and Embedded Architecture and Compilation Network of Excellence(HiPEAC)，2012：91-94.

[4] BINKERT N，BECKMANN B，BLACK G，et al. The gem5 simulator[J]. ACM SIGARCH Computer Architecture News，2011，39(2)：1-7.

[5] TA T，CHENG L，BATTEN C. Simulating multi-core RISC-V systems in gem5[C]// Workshop on Computer Architecture Research with RISC-V. 2018.

[6] LUO T，WANG X，QU C，et al. An FPGA-based hardware emulator for neuromorphic chip with RRAM[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems，2018，39(2)：438-450.

[7] PATEL H V，RATHOD S S，SHAH P H. An FPGA based Hardware Emulator for Neuromorphic Chip[C]// 2020 International Conference on Electronics and Sustainable Communication Systems(ICESC). IEEE，2020：1131-1136.

[8] LIU S，LAU F C M，SCHAFER B C. Accelerating FPGA prototyping through predictive model-based HLS design space exploration[C]// Proceedings of the 56th Annual Design Automation Conference. 2019：1-6.

[9] DENNIS D K，PRIYAM A，VIRK S S，et al. Single cycle RISC-V micro architecture processor and its FPGA prototype[C]// 2017 7th International Symposium on Embedded Computing and System Design(ISED). IEEE，2017：1-5.

[10] JIANG X Z. Software-Hardware Co-emulation Automation Verification Platform Design[D]. Xi'an：Xidian University，2019.

[11] SUKHWANI B，ROEWER T，HAYMES C L，et al. Contutto：A novel FPGA-based prototyping platform enabling innovation in the memory subsystem of a server class processor[C]// Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. 2017：15-26.

[12] GUO H，HUANG L B，ZHENG Z，et al. Proto-perf：A fast and accurate performance evaluation method for general purpose processor prototype system[J]. Computer Engineering and Science，2021，43(4)：579-585.

[13] Valgrind. Valgrind Documentation [EB/OL]. (2022-10-24) [2022-09-26]. https://valgrind.org/docs/manual/valgrind_manual.pdf.

[14] PHANSALKAR A，JOSHI A，JOHN L K. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite[C]// Proceedings of the 34th Annual International Sympo-

sium on Computer architecture. 2007:412-423.
[15] CRIU. Checkpoint/restore in user space[EB/OL]. (2013-12-26)[2022-09-26]. https://criu.org/CRIU:About.
[16] QEMU. QEMU is a generic and open source machine emulator and virtualizer[EB/OL]. (2020-07-07)[2022-09-26]. https://wiki.qemu.org/Main_Page.
[17] WEAVER V M, MCKEE S A. Using dynamic binary instrumentation to generate multi-platform simpoints: Methodology and accuracy[C]// International Conference on High-Performance Embedded Architectures and Compilers. Berlin: Springer, 2008: 305-319.
[18] CALDER B, SHERWOOD T, HAMERLY G, et al. Simpoint: Picking representative samples to guide simulation[J/OL]. https://sites.cs.ucsb.edu/~sherwood/pubs/CHAPTER-simpoint.pdf.
[19] SHERWOOD T, PERELMAN E, HAMERLY G, et al. Automatically characterizing large scale program behavior[J]. ACM SIGPLAN Notices, 2002, 37(10): 45-57.
[20] LIKAS A, VLASSIS N, VERBEEK J J. The global k-means clustering algorithm[J]. Pattern Recognition, 2003, 36(2): 451-461.
[21] HENNING J L. SPEC CPU2006 benchmark descriptions [J]. ACM SIGARCH Computer Architecture News, 2006, 34(4): 1-17.

DENG Lin, born in 1980, Ph. D, associate research fellow, is a member of China Computer Federation. His main research interests include computer architecture, microprocessor design, integrated circuit verification and basic software.

(责任编辑:何杨)