计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202111189
60(5): 1164−1176, 2023

针对冗余零的跨平台细粒度性能分析技术

游　心[1]　杨海龙[1]　雷克伦[1]　孔祥浩[1]　徐　筠[2]　栾钟治[1]　钱德沛[1]
[1](北京航空航天大学计算机学院　北京　100191)
[2](北京仿真中心航天系统仿真重点实验室　北京　100854)
(youxin2015@buaa.edu.cn)

A Cross-Platform Fine-Grained Performance Analysis Technique for Redundant Zeros

You Xin[1], Yang Hailong[1], Lei Kelun[1], Kong Xianghao[1], Xu Jun[2], Luan Zhongzhi[1], and Qian Depei[1]
[1] (*School of Computer Science and Engineering*, *Beihang University*, *Beijing* 100191)
[2] (*Science and Technology on Space System Simulation Laboratory*, *Beijing Simulation Center*, *Beijing* 100854)

Abstract　Software inefficiencies caused by redundant zeros will introduce massive zero values to be loaded or used for trivial computation, which significantly wastes memory and compute resources. However, the compiler toolchain still cannot effectively identify the redundant operations dealing with zeros and hardware optimizations handling redundant zeros have not been adopted in commercial hardware yet. Although ZeroSpy can detect the existence of redundant zero buried within software and report sufficient information for performance optimization, its detection is still limited in Intel platform as well as its large overhead. Therefore, we propose a cross-platform tool DrZero to overcome these limitations. DrZero can detect redundant zeros in both x86 and ARM platforms and it implements novel online analysis based on buffered tracing for lower overhead. For ARM platform, we propose floating-point estimation via dataflow analysis to estimate the data type of a memory operand for further detection. The evaluation results demonstrate that DrZero can detect redundant zeros with code-centric, data-centric analysis on both x86 and ARM platforms with 45.31×, 54.20× and 14.12×, 13.40× performance overheads, respectively. Besides, DrZero incurs 37.2% and 55.8% lower time overheads than ZeroSpy with code-centric and data-centric analysis on the x86 platform, respectively. Based on the optimization guidance revealed by DrZero, we can achieve 1.76× and 2.12× speedups at maximum on both x86 and ARM platforms after eliminating redundant zeros for evaluated applications. DrZero is open-source at https://github.com/buaa-hipo/zerospy-drcctprof.

Key words　binary instrumentation; redundant zero; software inefficiency; performance analysis and optimization; cross-platform

摘　要　冗余零造成的软件低效行为会导致大量的0值被读取甚至被用来重复地进行无用的计算,从而导致内存、计算资源的浪费.然而,现有的编译器工具链都不能够有效地识别并消除应用程序中0值相关的冗余操作,且冗余零相关的硬件优化方法仍然没有应用于商业处理器中.虽然ZeroSpy能够识别冗余零并提示足够的信息来指导优化,但其检测方法仍然局限于Intel平台,且其过大的性能开销阻碍了更加广泛的使用.针对冗余零的跨平台细粒度性能分析工具DrZero则可以克服上述限制.DrZero支持x86和ARM

收稿日期: 2021-11-30; **修回日期**: 2022-06-07
基金项目: 国家重点研发计划项目(2022ZD0117805); 国家自然科学基金项目(62072018, U22A2028); 中央高校基本科研业务费专项资金
This work was supported by the National Key Research and Development Program of China (2022ZD0117805), the National Natural Science Foundation of China (62072018, U22A2028), and the Fundamental Research Funds for the Central Universities.
通信作者: 杨海龙(hailong.yang@buaa.edu.cn)

平台，并实现在线细粒度缓存迹分析来减少性能开销. 为了支持 ARM 平台，基于数据流分析的数据类型推断方法可以自动推断内存读取值的数据类型. 经过评测，DrZero 的以代码、数据为中心的分析模式可以在 x86 和 ARM 平台上分别以平均 45.31 倍、54.20 倍和 14.12 倍、13.40 倍的性能开销识别冗余零并给出优化建议. 此外，在 x86 平台上与 ZeroSpy 所报告的性能开销相比，DrZero 的平均性能开销分别在以代码、数据为中心的分析模式下降低了 37.2%，55.8%.基于 DrZero 给出的性能优化指导，应用程序优化后在 x86 和 ARM 上分别达到了最高 1.76 倍和 2.12 倍的性能加速. DrZero 的实现代码已经开源：https://github.com/buaa-hipo/zerospy-drcctprof.

关键词 二进制插桩；冗余零；软件低效行为；性能分析与优化；跨平台

中图法分类号 TP311

如今，无论是生产软件还是科学计算应用都变得越来越复杂，其中软件包含的大量库依赖项以及复杂的控制和数据流使得软件效率难以得到保证. 并且，如此高的复杂性很容易导致意料之外的低执行效率，从而阻碍软件达到最佳性能. 软件效率低下往往涉及冗余操作，例如从内存中反复加载相同的值[1]、写入从未使用过的值[2]、使用从未使用过的中间值来覆盖同一位置的值[3]以及重复计算相同的值[4-5]. 此外，大量的应用将稀疏数据作为输入，而使用稠密数据结构处理该稀疏数据将会导致资源的大量浪费[6-8]. 以往的研究已经证明上述低效率的根本原因之一是一些指令和数据结构经常处理冗余零[9].

目前已经有大量的真实应用报告了大量冗余零的存在并对其进行针对性优化来达到更好的效果. 例如在深度神经网络领域，研究人员[7-8]已经提出了软件或者硬件的优化方法来实现对神经网络中稀疏性的自动检测以及特定的稀疏优化来达到更好的性能；在视频编码领域，研究人员[6]提出了一些全 0 块（all-zero block）检测方法来跳过这些块的计算从而达到更高的性能. 而这些方法都是针对特定领域上的工作，并不能对其他领域的应用提供冗余零的检测或优化指导.

此外，现代编译器优化无法识别访存和计算操作中涉及的冗余零以进行进一步的代码优化. 而诸如 perf[10], HPCToolkit[11], VTune[12], Gprof[13], CrayPat[14], Oprofile[15] 这些性能分析工具也不会识别与冗余零相关的低效率行为，从而无法提供相应的优化指导. ZeroSpy[9] 虽然能够正确识别冗余零相关的低效行为并给出相应的指导建议，其检测方法仍然局限于 Intel 平台，且不支持目前逐渐流行的 ARM 计算平台. 此外，ZeroSpy 的高性能开销也大大加剧了真实应用的性能分析开销. 而目前 ARM 架构在高性能领域已经越来越受到重视，在 Top500 中取得世界第一性能的富岳超级计算机[16]的中央处理器就是基于 ARMv8 架构设计的，而目前并没有一款编译器或性能分析工具可以在 ARM 平台识别冗余零相关的低效行为，从而错失大量潜在的性能优化机会.

因此，本文提出了 DrZero[17]，它是一款基于冗余零检测的可跨平台的、具有更低分析开销的性能分析工具. DrZero 实现了一种检测应用程序执行过程中冗余零的方法，包括：1）识别由于数据结构使用不当、数据宽度过大以及无用计算造成的冗余零；2）提示冗余零发生的源代码行与执行上下文来提供直观的优化指导；3）依据应用检测出的冗余零信息进行针对性优化显著提高应用的执行性能或能效. 此外，DrZero 基于 Dynamorio 二进制动态插桩框架[18]实现，并通过基于数据流分析的数据类型推断来识别 ARM 访存指令读取的数据类型，以及实现在线细粒度缓存迹分析方法来进一步减少性能开销. 相较于 ZeroSpy，DrZero 支持跨平台的冗余零检测，并拥有更低的开销以及更加友好的报告界面来提示足够充分的冗余零信息以及对应的优化建议.

1 冗余零的定义与来源

类似文献 [9] 中所提出的冗余零定义，本文也使用冗余映射（redmap）以及冗余零比例（redundant zeros fraction）作为冗余零的检测目标. 其中，冗余零为指令读取值的高位 0 值，因此理论上完全消除冗余零对值的表达与计算的正确性不会造成任何影响. 因此高冗余零比例表示具有较大的优化空间，冗余映射则展现读取值的模式，从而为优化策略的设计提供参考. 其中，冗余零主要来自 3 个方面：数据宽度过大、数据结构使用不当以及 0 值相关的无用计算.

1）数据宽度过大是造成访存过程中某个指令或者从某个数据对象读取的值中高位字节总是为 0 的

主要原因之一. 例如, 访存指令频繁地从内存读取 64b 整型值(占 8B), 且该值的高 7B 总是为 0, 即该访存指令访问了大量的冗余零. 造成该冗余零现象的主要原因就是数据宽度过大, 该数据使用 8b 整型(占 1B)即可保证数值的等价性, 而不必使用过长的 64b 整型来存储、访问该数据. 这些冗余零会造成有限的多级缓存资源的浪费, 从而造成潜在的性能损失.

2)使用不当的数据结构也会造成大量的冗余零, 从而造成大量的资源浪费. 如图 1 所示, 使用稠密数据结构与稠密算法对该稀疏数据进行处理会造成大量的完全冗余零. 而如果应用稀疏数据结构以及对应的稀疏算法会大幅度减少存储、访存开销, 并避免大量无用的 0 值相关的计算, 从而得到更好的性能.

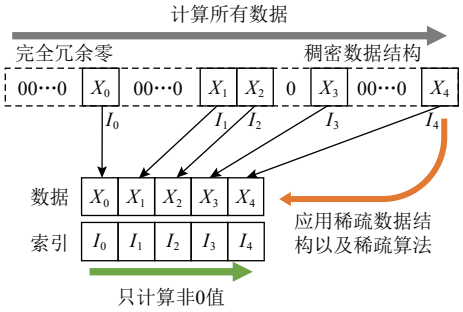

Fig. 1 An illustration of software inefficiency by inappropriate use of data structure

图 1 使用不当的数据结构导致的软件低效行为示例

3)0 值相关的无用计算则是另一类常见的冗余零来源. 图 2 展示了 bwaves 应用程序中 0 值相关的无用计算示例代码. 通过 DrZero 检测以及人工插桩等细致分析后, 该示例代码中 *u*,*v*,*w* 这 3 个变量常常读取完全冗余零(0 值), 从而进一步造成在计算 *mu* 变量值时引入繁重的 0 值相关的无用计算. 0 值相关的无用计算并不是所有的冗余零都值得进行优化,

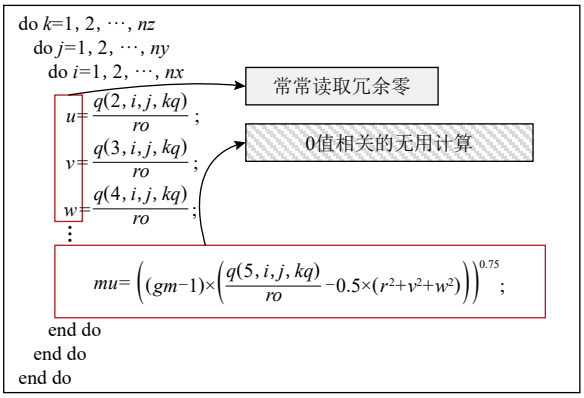

Fig. 2 An example of zero-agonistic computation

图 2 0 值相关的无用计算示例

原则上高冗余零比例的、复杂且高计算开销的操作更加值得进一步优化.

2 相关工作

值得注意的是, 传统的基于硬件计数器的性能分析工具[10-15,18]往往由于其中存在的大量冗余访存与计算反而会充分利用硬件资源, 从而得到较好的性能结果. 因此, 若含有大量冗余零的访存、计算操作浪费了大量资源, 基于硬件计数器的性能分析工具并不能有效地识别类似冗余零的资源浪费导致的软件低效行为, 从而无法给出有效的优化建议.

值分析器(value profiler)的开发是为了查明软件中含有的冗余计算. 值分析器最早由 Calder 等人[19]提出, 它可以检测程序代码并查明存储在寄存器或存储器中的不变变量或半不变变量. 其后续研究提出了此值分析器的一种变体[20]. Wen 等人[4]开发了一种细粒度的探查器(RedSpy)来识别静默写入(scilent writes)并给出相关指导优化. RedSpy 不但检测到计算结果以及数据移动中的冗余, 还可以通过报告调用上下文及其在源代码中的位置来报告冗余, 从而精确定位冗余代码区域. 此外, Su 等人[1]开发了另一个细粒度的探查器 LoadSpy, 以识别软件中的冗余负载, 他们声称, 在软件的指令中加载相同的值效率很低. LoadSpy 跟踪每个加载指令以查明这些冗余负载, 并报告涉及冗余负载的指令对. 此外, Tan 等人[21]开发了 CIDetector, 并证明即使使用现代的优化编译器, 无效操作和冗余操作仍然存在. ZeroSpy[9]可以查明与零相关的冗余算术和内存操作, 但 ZeroSpy 基于 Intel Pin[22]实现, 且 ARM 指令集与 x86 指令集不同, 访存与计算是分离的, 所以不能通过指令来识别访问内存数据的类型. 因此, ZeroSpy 的检测方法与实现手段都局限在 Intel 平台, 不能直接应用在跨平台冗余零检测中.

另外, 有一些方法致力于通过减少数据类型的长度来利用冗余零. Stephenson 等人[23]提出了一种位宽分析方法来静态标识变量的最大所需位宽, 从而尽可能减少数据类型的长度. 当在目标 FPGA 平台上进行评估时, 他们的方法可能对面积、速度和功耗产生积极影响. 另一方面, Precimonious[24]工具可用来权衡浮点精度和性能. 这些方法与工具都只能检测数据长度中的冗余, 而 DrZero 还可以检测不适当的数据结构以及零相关导致的冗余计算. 此外, DrZero 也不需要任何源代码即可进行冗余零检测.

3 冗余零检测方法

本文提出并实现了针对冗余零的跨平台细粒度性能分析工具DrZero. 具体来讲, DrZero 基于 Dynamorio 动态二进制插桩框架[25]实现. 相较 ZeroSpy 选用的 Intel Pin[22], Dynamorio 具有可跨平台(支持x86与ARM)、支持细粒度的分析与插桩、开源等特性, 从而可以支持跨平台更加细粒度的插桩、分析工具. Dynamorio 框架是基于事件来触发分析与插桩过程, 例如线程启动和终止、基础块载入等事件. 当事件发生时, Dynamorio 会触发初始化时注册的回调函数来对目标事件进行分析、插桩. 其中, 当基础块载入事件发生时, Dynamorio 可以获取该基础块将要执行的所有指令, 因此 DrZero 基于各个基础块对所有包含内存读取操作的指令进行分析, 并进行基础块级别的动态二进制插桩. DrZero 的跨平台冗余零检测技术总览如图 3 所示, 包含: ①基于数据流分析的数据类型推断; ②细粒度缓存迹插桩与在线分析. 其中, 根据检测目标的不同, 在线分析又分为以代码为中心的分析模式和以数据为中心的分析模式. 在目标程序执行完毕后, DrZero 会生成一系列报告文件, 并通过定制的 VSCode 扩展来将冗余零报告可视化, 并基于检测结果提供优化建议. 下文将对 DrZero 的各个核心技术进行详细讲解.

3.1 基于数据流的数据类型推断

与 x86 指令集不同, ARM 指令集集中访存与计算为不同的指令, 即计算操作不能直接访问内存, 而同一访存指令可能读取整型值或浮点值. 此外, 通用寄存器除了保存整型值以外, 也可能保存浮点值并在计算时移动到浮点寄存器来进行浮点计算, 因此也不能只通过内存读取操作的目标寄存器类型来判断读取值是浮点值还是整型值. 为了解决上述挑战, DrZero 采用基于数据流的数据类型推断来尽可能推断出读取值的数据类型, 从而可以采用对应的冗余零分析策略来获取冗余映射以及冗余零比例.

基于数据流的数据类型推断算法如算法 1 所示. 由于数据类型中的数据宽度可以直接由访存宽度直接获取, 因此算法 1 的主要目的在于推断该指令读取值是否为浮点数. 首先, 算法 1 根据目标寄存器类型推断是否为浮点指令(行①). 如果目标寄存器为浮点寄存器则推断为浮点指令, 返回真值. 由于本文主要关注访存指令中存在的冗余零现象, 因此只在指令读取内存中的值时才进一步基于数据流进行推断(行②). 若该指令读取内存值, 算法则枚举所有目标寄存器对象, 并对每个目标寄存器对象查找该基础代码块中在该指令后执行的所有指令的源寄存器中是否存在定义-使用关系(行③~⑫), 若存在, 则返回使用目标寄存器的指令的类型(行⑬). 如果上述数据流分析过程都没有找到定义-使用关系, 则返回先前推断的 *instr* 指令类型(行⑳).

算法 1. 数据类型推断算法.

输入: 目标访存指令 *instr*;

输出: 是否为浮点指令.

```
① bool is_float = instr_is_floating_self(instr);
② if(!is_float && instr_reads_memory(instr))
③   int num_dsts = instr_num_dsts(instr);
④   for(int i=0; i<num_dsts; ++i)
⑤     opnd_t opnd = instr_get_dst(instr, i);
⑥     if(opnd_is_reg(opnd))
⑦       reg_id_t reg_def = opnd_get_reg(opnd);
⑧       for (instr_t* ins = instr; ins != NULL;
          ins = instr_get_next(ins))
⑨         int num_srcs = instr_num_srcs(ins);
⑩         for(int j=0; j<num_srcs; ++j)
⑪           opnd_t opnd = instr_get_src(ins, j);
⑫           if (opnd_is_reg(opnd) &&
              opnd_get_reg(opnd) == reg_def)
⑬             return instr_is_floating_self(ins);
⑭           end if
⑮         end for
```

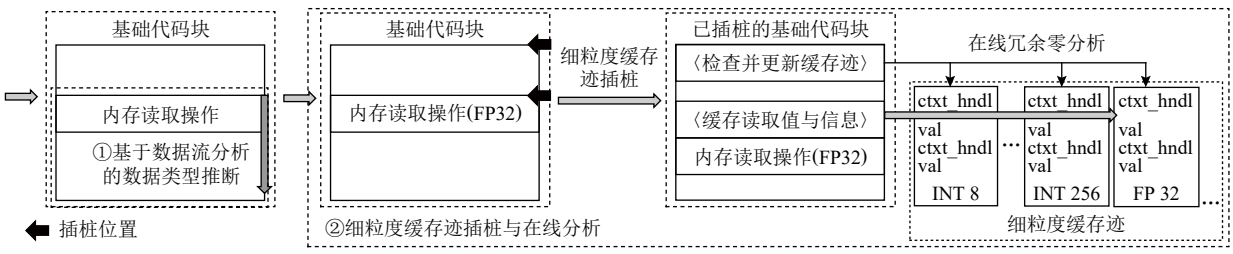

Fig. 3 An overview of the cross-platform redundant zeros detection techniques of DrZero

图 3 DrZero 跨平台冗余零检测技术总览

⑯ end for
⑰ end if
⑱ end for
⑲ end if
⑳ return *is_float*.

在 ARM 平台上，算法 1 可以有效地识别大部分浮点数内存访问指令，从而应用适当的冗余零分析方法来建立冗余映射并识别冗余零. 但由于程序执行过程的不确定性，基础代码块最后的跳转指令的目标地址往往无法静态确定，因此基于数据流的数据类型推断仅受限于单个基础代码块，从而某些存在跨基础代码块的定义-使用关系的浮点值读取指令时可能仍会推断为读取整型值. 然而，由于时空局部性的存在，这些推断错误依旧可以认为是较少的，不会影响最终报告的冗余映射与冗余零比例.

3.2 在线细粒度缓存迹分析

为了分析得到所有访存操作读取内存值的冗余映射与冗余零信息，本文提出了在线细粒度缓存迹分析在细粒度缓存区中缓存一定量的访存操作信息并基于该缓存区进行针对冗余零的在线分析来检测冗余零导致的软件低效行为. 具体来讲，该过程主要分为 4 步：细粒度缓存区的创建、细粒度缓存迹插桩、细粒度缓存迹更新以及在线冗余零分析.

在工具初始化时，DrZero 需要创建细粒度缓存区来保存程序运行过程中内存读取操作的信息，包括目标地址、调用上下文以及读取的值. 为了避免插桩后的代码执行时频繁检查读取值的数据类型从而造成繁重的分支操作，DrZero 对每个可能的数据类型都建立缓存区，包括 1b, 2b, 4b, 8b 整型值，单、双精度浮点值以及 128b, 256b 整型向量，单、双精度浮点向量. 对于每个缓冲区，DrZero 分配大小为 4 096 个元素的缓冲区.

在目标应用程序执行时，Dynamorio 会触发基础代码块载入事件. 在经过 3.1 节所述的数据类型推断后，DrZero 向基础代码块中插入细粒度缓存迹更新以及在线冗余零分析代码. 由于读取内存值的数据类型可静态推断，DrZero 根据推断的数据类型来插入相应的指令，向对应缓存区存储目标内存地址、调用上下文句柄(通过 DrCCTProf[26] 获取)以及读取值. 例如，图 3 中内存读取操作被推断为 32b 浮点类型(FP32)，然后插入指令将相应信息缓存到 FP32 对应的细粒度缓存迹中以便后续在线分析. 此外，在每个基础代码块级别都需要静态推断各个细粒度缓存迹将要填充的数量，并对每个将要填充的细粒度缓存迹都在基础代码块入口进行插桩：

1) 插入指令，检查是否将满或溢出；

2) 插入条件跳转指令与函数调用，从而保证在将满或溢出时调用在线冗余零分析代码；

3) 在在线冗余零分析后清空该缓存迹.

在插桩完成后，Dynamorio 将会负责执行插桩后的代码，因此所有内存读取操作的信息都会被记录到细粒度缓存迹中，并及时更新细粒度缓存迹，以及在将满或溢出时触发在线冗余零分析. 该过程也称为细粒度缓存迹更新. 在线冗余零分析过程根据检测模式的不同分为以代码为中心的在线分析和以数据为中心的在线分析. 由于具体 DrZero 的冗余零检测算法与 ZeroSpy[9] 中的冗余零检测算法类似，本文仅介绍在线分析的核心思想.

1) 以代码为中心(code-centric)的在线分析. 该分析过程旨在发现指令级别的冗余零现象，可以检测冗余零导致的无用计算以及数据宽度过长导致的资源浪费. 由于触发在线分析的细粒度缓存迹都已经静态确定其数据类型，在线分析过程中便不用检查其类型并直接进行冗余零检测. 在冗余零检测后，所有检测数据都被记录、累计在以调用上下文句柄为键值的散列映射表中.

2) 以数据对象为中心(data-centric)的在线分析. 该分析过程旨在发现数据对象级别的冗余零现象，可以检测数据结构使用不当以及数据宽度过长的问题. 类似以代码为中心的在线分析，在线分析过程中不用检查其数据类型. 数据对象信息则在在线分析过程中通过缓存迹中的目标内存地址从 DrCCTProf 获取数据对象信息，包括静态数据对象的变量名以及动态数据对象分配时的调用上下文. 由于只有静态数据对象与动态数据对象可以获取到足够的调试信息以供后续优化，本文只对静态与动态数据对象进行冗余零检测与报告.

相较 ZeroSpy[9] 的检测方法，在线细粒度缓存迹分析拥有 3 点优势：1)尽可能避免频繁、高开销的算数状态保存操作；2)避免对每次内存读取的值进行分析、记录，从而减轻工具对应用的缓存污染；3)集中批处理冗余零检测与记录操作，具有更好的时空局部性.

3.3 冗余零报告与性能优化指导的可视化界面

为了更好地展示冗余零检测结果，DrZero 提供基于 VSCode 的插件来将检测结果可视化. DrZero 的性能报告分为总览(metric overview)和按线程划分的详细报告(detailed metrics). 如图 4 所示，ZeroSpy 用户

界面的总览页面分为全局整数冗余零字节(total integer redundant byte)以及全局浮点数冗余零字节(total floating point redundant byte)两个部分，并分别列出了冗余零与总访问字节.此外，各个线程所统计的冗余零数量以及冗余零比例也可以在此页面查看.按线程划分的详细报告则可通过图 4 中“detail”链接查看. DrZero 可以分别生成以代码为中心的分析模式与以数据对象为中心的分析模式报告，其线程详细报告页面如图 5 所示.

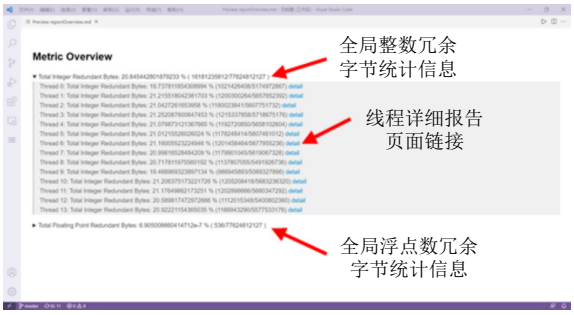

Fig. 4 Metric overview page reported by DrZero GUI

图 4 DrZero 用户界面报告的总览页面

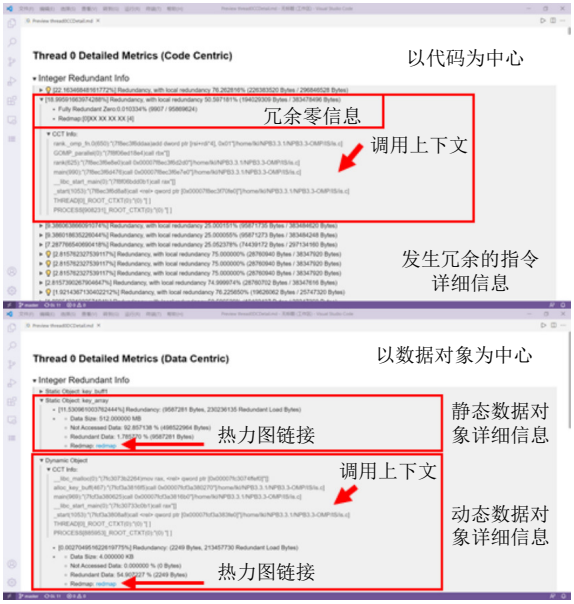

Fig. 5 The detailed report of each thread by DrZero GUI

图 5 DrZero 用户界面的线程详细报告页面

1) 以代码为中心的分析模式.如图 5 所示，各个线程的以代码为中心模式的报告页面分为整数冗余信息以及浮点冗余信息 2 个部分.对于以代码为中心的模式报告，页面将会显示发生冗余的程序指令的记录，且每条记录是按照检测到的冗余字节数量从高到低排列.对于每条记录，“Redundancy”是该指令的冗余零在所有检测到的冗余零中的比例，“local redundancy”则是该指令的冗余零在所有执行该特定指令检测到的冗余零中的比例.使用者可以进一步点击每条记录以显示更多冗余零信息，包括完全冗余零占比、指令的冗余映射(从最低到最高).此外，每条记录会给出包含源代码行号与文件路径的调用上下文信息，从而指导开发者进一步的代码优化.

2) 以数据对象为中心的分析模式.如图 5 所示，各个线程的以数据对象为中心的模式报告页面同样分为整数冗余信息以及浮点冗余信息 2 个部分.对于以数据对象为中心的模式报告，页面将会显示发生冗余的静态或动态数据对象的记录，且每条记录是按照检测到的冗余字节数量从高到低排列.对于静态数据对象，每一条记录都会显示对象的名称，以帮助使用者在源代码中定位该静态数据对象.对于动态分配的数据对象，报告会提供其调用上下文信息(CCT info)指示其动态数据对象创建的位置.每条记录均会显示静态或动态数据对象的数据大小、以字节为单位的未访问数据比例信息以及冗余零比例信息.此外，使用者还可以点击“redmap”字样以跳转到链接的数据对象的以字节为单位的冗余映射热力图，如图 6 所示.

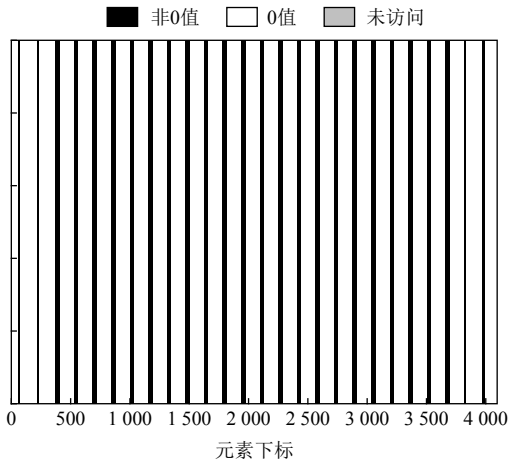

Fig. 6 The heatmap of the dynamic data object with significant redundant zeros in NPB-IS program

图 6 NPB-IS 程序中报告大量冗余零的动态数据对象的热力图

4 实验结果与分析

4.1 实验设置

本文使用如表 1 所示的 x86 和 ARM 实验平台对 DrZero 进行评测.为了检验 DrZero 的跨平台冗余零检测的性能开销以及检测效果，本文选用 NAS

Table 1 The Detailed Software and Hardware Configurations of Evaluated X86 and ARM Platform

表 1 x86 和 ARM 评测平台详细软硬件配置信息

平台	x86	ARM
处理器	Xeon E5-2680v4@2.40GHz	Cavium ThunderX2@2.50GHz
核数	14	32
内存	256GB DDR4	128GB DDR4
编译器	GCC 11.0	GCC 11.0
系统	Linux 5.8.0-55-generic #62~20.04.1-Ubuntu	Linux 5.4.0-74-generic #83-Ubuntu

Parallel Benchmarks(NPB-3.4)[27]和 Rodinia[28]基准测试程序集进行评测. 其中 NPB 基准测试使用 C CLASS 输入问题规模. 此外，为了评测实际应用规模的冗余零检测能力，本文也选用 SPEC CPU2017[29]中的 NAB 生命科学应用(ref 输入大小)以及 Fotonik3D 计算电磁学应用(ref 输入大小)作为案例研究来详细讲解 DrZero 的冗余零检测结果以及优化流程. 所有程序都使用 GCC 11.0-O3-g-fopenmp 编译，并在单个 CPU 上(x86 平台上 14 线程，ARM 平台上 32 线程)并行执行.

4.2 冗余零识别效果

DrZero 在 x86 和 ARM 平台上的冗余零检测结果分别如表 2 和表 3 所示. 其中，CC 表示以代码为中心的冗余零分析模式，其冗余零比例按照所有整数或浮点数累计的冗余零数量除以总访问字节数得到，CC 数值越大表明执行过程中冗余零相关的冗余计算、资源浪费越严重；DC 表示以数据为中心的冗余零分析模式，其冗余零比例按照所有数据对象中含有的冗余零数量除以所有数据对象的数量大小之和得到，DC 数值越大表明数据对象的稀疏性越高或潜在的数据宽度过长问题越严重. 此外，在 x86 和 ARM 平台上 DrZero 使用以代码为中心(CC)以及以数据为中心(DC)的冗余零分析模式的评测，结果对比分别如图 7、图 8 所示. 评测结果证明 DrZero 拥有跨平台检测冗余零导致的软件低效行为的能力. 此外，本文还展示了在 x86 平台上 ZeroSpy 与 DrZero 报告的冗余零比例的比较，如图 7 所示. 其中，由于 ZeroSpy 报告中冗余零比例在 CC 与 DC 模式下的计算方式相同，本文仅展示 CC 模式下的冗余零比例检测比较. DrZero 在大部分测试程序中检测的冗余零比例与 ZeroSpy 相当. 然而，有些程序的冗余零比例则有较明显的差距(如 heartwall). 通过细致分析发现，DrZero 与 ZeroSpy 所检测的冗余零数量差别不大. 而由于

Table 2 Fraction of Redundant Zero and Profiling Overheads Reported by DrZero on x86 Platform

表 2 DrZero 报告在 x86 平台上的冗余零比例以及检测分析开销

模式	名称及统计项	冗余零比例/%		时间开销	内存开销	模式	名称及统计项	冗余零比例/%		时间开销	内存开销
		整数	浮点数					整数	浮点数		
NPB CC	BT	2.02	20.30	54.83	1.25	NPB DC	BT	0.00	0.01	32.15	29.52
	CG	10.38	0.00	19.33	1.14		CG	0.86	0.00	57.04	28.68
	EP	5.44	0.00	15.61	6.88		EP	0.44	0.00	6.57	135.49
	FT	2.46	0.00	18.18	1.02		FT	0.00	0.00	18.46	25.88
	IS	22.96	0.00	14.40	1.17		IS	9.28	0.00	47.76	28.46
	LU	6.06	12.11	24.94	1.28		LU	0.00	0.01	29.36	31.01
	MG	10.33	7.38	4.45	1.03		MG	0.00	0.03	23.05	25.93
	SP	4.09	1.64	23.10	1.22		SP	0.00	0.01	39.74	29.32
	UA	23.54	5.08	33.98	1.47		UA	1.07	0.07	27.94	31.92
	中位数	6.06	1.64	19.33	1.22		中位数	0.00	0.01	29.36	29.32
	最大值	23.54	20.30	54.83	6.88		最大值	9.28	0.07	57.04	135.49
	平均值	9.70	5.17	23.20	1.83		平均值	1.29	0.01	31.34	40.69
Rodinia CC	中位数	38.41	0.13	64.76	3.4	Rodinia DC	中位数	34.17	0.01	64.98	82.42
	最大值	56.15	11.58	111.58	99.76		最大值	91.14	10.23	157.45	2 480.46
CC	中位数	23.25	0.17	40.32	2.95	DC	中位数	21.72	0.01	44.02	58.91
	平均值	24.47	2.83	45.31	7.96		平均值	27.14	2.71	54.20	188.40

Table 3 Fraction of Redundant Zero and Profiling Overheads Reported by DrZero on ARM Platform

表 3 DrZero 报告在 ARM 平台上的冗余零比例以及检测分析开销

模式	名称及统计项	冗余零比例/%		时间开销	内存开销	模式	名称及统计项	冗余零比例/%		时间开销	内存开销
		整数	浮点数					整数	浮点数		
NPB CC	BT	1.85	25.95	12.49	1.34	NPB DC	BT	0.00	0.01	2.82	32.64
	CG	12.74	0.00	4.29	1.16		CG	0.30	0.00	4.80	31.61
	EP	3.65	0.00	5.65	4.65		EP	1.57	0.00	4.03	151.07
	FT	0.44	0.00	7.96	1.03		FT				OOM
	IS	28.15	0.00	6.04	1.08		IS	7.25	0.00	24.84	31.61
	LU	70.53	1.69	4.47	1.39		LU	0.00	0.00	2.67	34.68
	MG	7.61	11.51	4.90	1.05		MG	0.01	0.01	21.99	26.85
	SP	3.99	1.52	5.23	1.30		SP	0.00	0.01	5.31	32.48
	UA	29.74	2.14	6.86	1.68		UA	0.25	0.01	7.09	38.40
	中位数	7.61	1.52	5.65	1.30		中位数	0.13	0.01	5.06	32.56
	最大值	70.53	25.95	12.49	4.65		最大值	7.25	0.01	24.84	151.07
	平均值	17.63	4.76	6.43	1.63		平均值	1.17	0.01	9.20	47.42
Rodinia CC	中位数	40.94	0.00	12.33	3.28	Rodinia DC	中位数	0.66	0.00	13.07	111.69
	最大值	58.09	10.59	60.47	47.75		最大值	8.63	0.01	36.86	2 794.66
CC	中位数	32.02	0.00	8.76	2.74	DC	中位数	0.53	0.00	10.20	55.68
	平均值	30.21	2.51	14.12	6.06		平均值	1.28	0.00	13.40	265.52

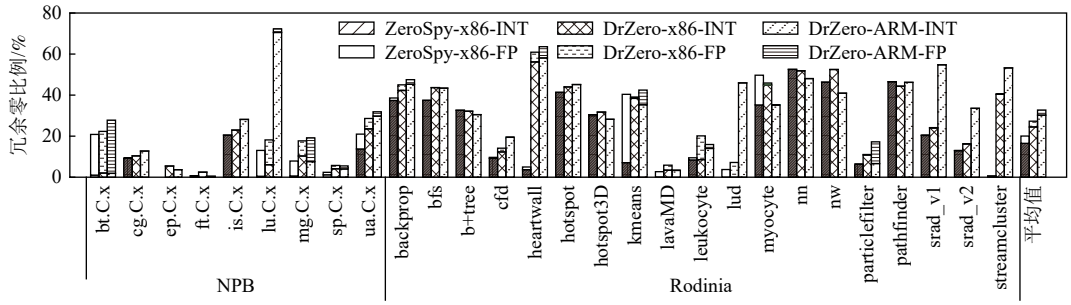

Fig. 7 Fraction of integer (INT) and floating point (FP) redundant zeros reported by DrZero CC mode on x86 and ARM platforms

图 7 DrZero CC 模式下分别在 x86 和 ARM 平台上检测的整型（INT）以及浮点（FP）类型的冗余零含量

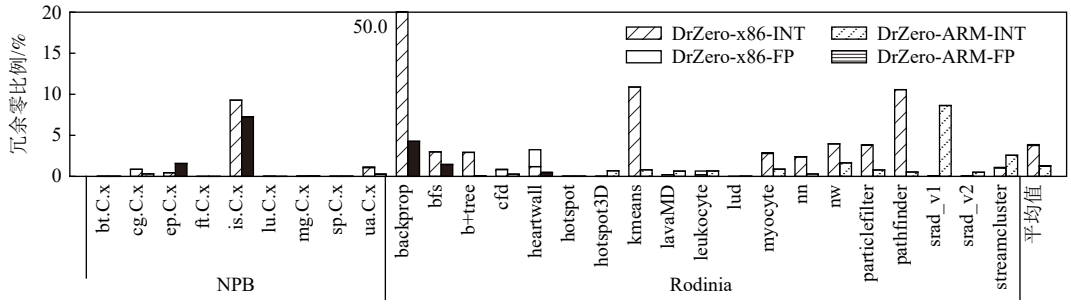

Fig. 8 Fraction of integer (INT) and floating point (FP) redundant zeros reported by DrZero DC mode on x86 and ARM platforms

图 8 DrZero DC 模式下分别在 x86 和 ARM 平台上检测的整型（INT）以及浮点（FP）类型的冗余零含量

Dynamorio 不能完全识别所有的指令，DrZero 可能会将有些特殊的访存指令识别为空指令(nop)，从而大大低估了总访问字节数. 此外，通过图 7 所示的同一代码在不同平台上的冗余零比例对比，本文发现虽然某些代码在不同平台上展现一定的冗余零比例的一致性，但仍然有一些基准测试程序在同样的代码、不同平台下的冗余零情况具有较大的差异性(例如 LU). 这表明 x86 和 ARM 指令集对程序的针对冗余零的低效行为拥有不同的敏感度. 不同指令集之间的冗余零发散现象需要进一步详细研究.

4.3 性能与内存开销

本文评测的 DrZero 的性能与内存开销如表 2 与表 3 所示. 实验结果表明 DrZero 在 x86 平台上的以 CC 与 DC 性能开销的中位数分别为 40.32 倍与 44.02 倍. 与 ZeroSpy 所报告的性能开销的中位数 64.17 倍(CC)与 99.52 倍(DC)相比[9], DrZero 的平均性能开销分别降低了 37.2% 和 55.8%. 此外, 在 ARM 平台, DrZero 的 CC 和 DC 的平均性能开销仅为 14.12 倍和 13.40 倍, 性能开销中位数则仅分别为 8.76 倍和 10.20 倍. DrZero 的性能开销与常见的二进制插装工具[1-4] 持平甚至更低, 因此具有良好的应用前景.

ZeroSpy(x86 平台运行)、DrZero-x86(x86 平台上运行)以及 DrZero-ARM(x86 平台上运行)之间在以代码为中心的和以数据对象为中心的分析模式下的详细性能开销对比分别如图 9 与图 10 所示. 实验结果表明, 本文提出的 DrZero 在大部分程序下 CC 和 DC 的性能开销都优于 ZeroSpy, 并且 DrZero 在 ARM 平台上相较在 x86 平台具有更低的性能开销. 更低的性能开销主要来自 2 个方面: 1)DrZero 调用上下文采集基于 DrCCTProf 实现, 使用性能开销更低的指令内联方式来获取调用上下文, 且 DrCCTProf 也针对 ARM 平台进行高度优化, 因此 DrZero 调用上下文采集开销相较依赖 Intel Pin 的 CCTLib[30] 更低; 2)DrZero 的在线细粒度缓存迹分析方法可以有效避免冗余且高开销的频繁算数、寄存器状态缓存操作, 从而带来可观的性能提升.

如图 11 所示, DrZero 在 CC 模式下, 在 x86 平台和 ARM 平台上内存开销都普遍低于 ZeroSpy. 然而, 如图 12 所示, DrZero 在 DC 模式下则相反, 其内存开销都普遍高于 ZeroSpy. 具体情况如表 2 所示, DrZero 在 x86 平台上的 CC 和 DC 的内存开销中位数分别为 2.74 倍和 55.68 倍. 相较 ZeroSpy 所报告的内存开销中位数 4.47 倍(CC)和 6.56 倍(DC), DrZero 在 CC 模式下具有更低的内存开销(降低 38.7%), 而在 DC 模式下具有更高的内存开销(升高 8.49 倍). 经过进一步分析, DC 模式下 DrCCTProf[26] 为了更高的性能, 将所有已静态、动态分配内存的数据对象的所有地址空间通过影射内存(shadow memory)方法按字节映射、存储其数据对象信息. 该方法相较 ZeroSpy 依赖的基于 Intel Pin 的 CCTLib[30] 支持的基于树的实现拥有更大的内存开销. 此外, 由于需要 DrZero 的 DC 模式下的冗余零检测需要数据对象的起、止内存地址, 在扩展 DrCCTProf 的数据对象信息实现后, 每个数据对象在每个影射位需要存储的信息变为原来的 3 倍, 使得内存开销问题变得更为严重, 甚至在 ARM 平台上 NPB FP 会超出内存容量(out of memory, OOM)从而

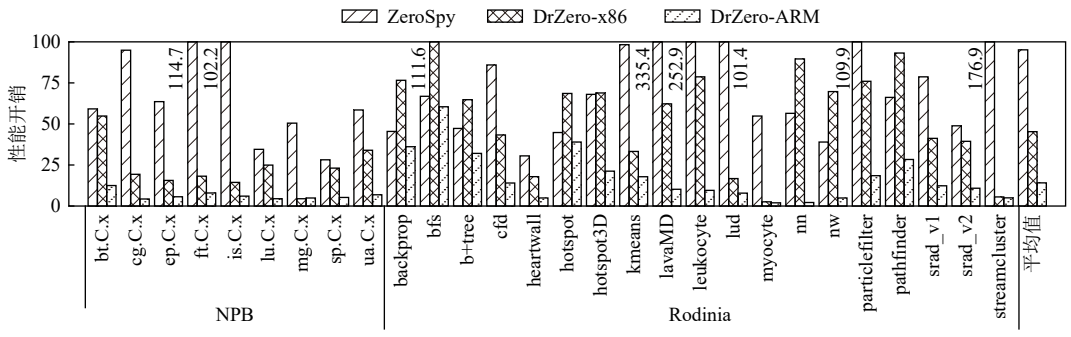

Fig. 9 Comparison of the performance overhead of CC mode caused by ZeroSpy, DrZero-x86, and DrZero-ARM

图 9 在 CC 模式下 ZeroSpy，DrZero-x86，DrZero-ARM 上的性能开销对比

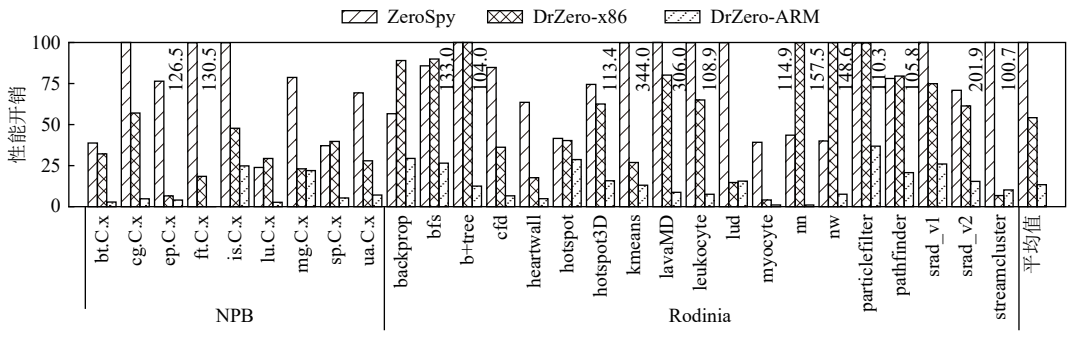

Fig. 10 Comparison of the performance overhead of DC mode caused by ZeroSpy, DrZero-x86, and DrZero-ARM

图 10 在 DC 模式下 ZeroSpy，DrZero-x86，DrZero-ARM 上的性能开销对比

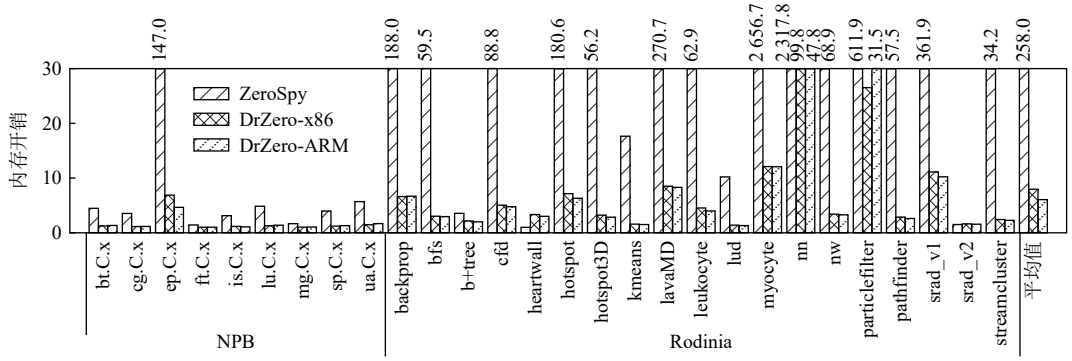

Fig. 11 Comparison of the memory overhead of CC mode caused by ZeroSpy, DrZero-x86, and DrZero-ARM

图 11 在 CC 模式下 ZeroSpy，DrZero-x86，DrZero-ARM 上的内存开销对比

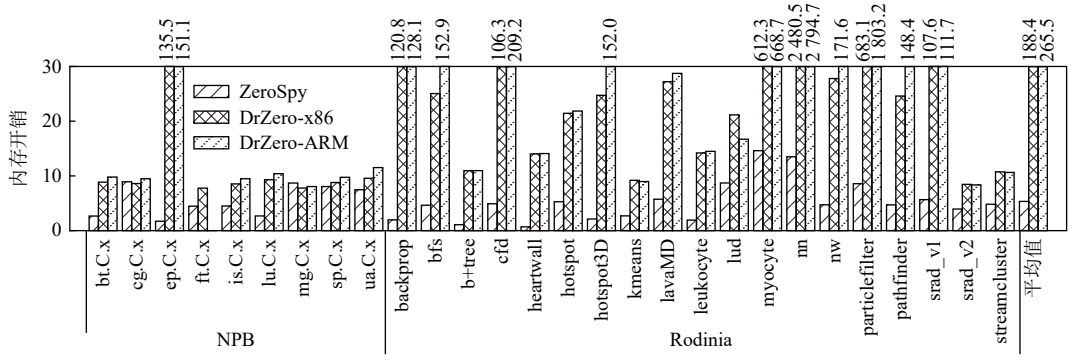

Fig. 12 Comparison of the memory overhead of DC mode caused by ZeroSpy, DrZero-x86, and DrZero-ARM

图 12 在 DC 模式下 ZeroSpy，DrZero-x86，DrZero-ARM 上的内存开销对比

无法采集冗余零信息. 因此, 未来工作需要提出更加内存友好且能保持低开销的数据对象采集方法来替代 DrCCTProf 的影射内存方法, 以获取更低的内存开销.

4.4 NAB 案例研究

NAB 是一个包含在 SPEC CPU2017 基准测试套件[29] 中的分子建模程序. NAB 是生命科学模拟领域典型的计算密集型程序. 由于 DrZero 报告的 NAB 冗余零情况在 x86 和 ARM 平台上都类似, 因此本文仅展示 x86 上的冗余零报告结果. DrZero 的以代码为中心的冗余零分析模式检测出 NAB 包含 11.52% 的整数冗余零和 6.57% 的浮点冗余零. DrZero 报告 NAB 中产生最多浮点冗余零的指令信息以及其对应于源代码中的位置如图 13 所示.

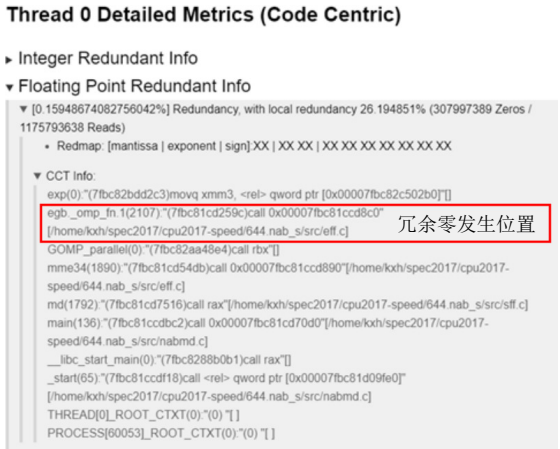

Fig. 13 The operation with the most significant redundant zeros in NAB reported by DrZero CC mode

图 13 DrZero CC 模式报告中 NAB 产生最多浮点冗余零的指令

产生冗余零的代码区域如图 14 所示, 其中 **kappa*的值频繁为 0. 由于完全冗余零的比例很高, 本文使用基于条件判断的方法(如图 15 所示的 if/else 语句)来跳过处理冗余零的无用计算. 此外, 该优化方法还可以通过在进入 2 个嵌套循环之前判断 **kappa*的值来进一步减少分支开销. 优化后的代码如图 15 所示, 类似的优化也应用于具有类似冗余的其他代码区域. 经过我们的优化, NAB 在 x86 平台实现 9.7% 的性能加速, ARM 平台上实现 6.08% 的性能加速.

4.5 Fotonik3D 案例研究

Fotonik3D 是一个包含在 SPEC CPU2017 基准测试套件[29] 中的计算电磁学程序, 其中包含了计算电

```
for (j = 0; j < prm->Natom; j++) {
 for (k = 0; k < npairs; k++) {
    ⋮
  fgbk = -(*kappa) × KSCALE / fgbi;
  expmkf = exp(fgbk) / (*diel_ext);
  temp6 = qiqj × temp4 × (dielfac+ fgbk × expmkf);
    ⋮
 }
}
```

Fig. 14 Code snippet containing the most significant redundant zeros in the *egb* function of NAB

图 14 NAB 的函数 *egb* 中包含最多冗余零的程序片段

```
if (*kappa==0) {
 for (j = 0; j < prm->Natom; j++) {
  for (k = 0; k < npairs; k++) {
      ⋮
   expmkf = 1 / (*diel_ext);
   temp6 = qiqj × temp4 × dielfac;
      ⋮
  }
 }
} else {
 for (j = 0; j < prm->Natom; j++) {
  for (k = 0; k < npairs; k++) {
      ⋮
   fgbk = -(*kappa) × KSCALE / fgbi;
   expmkf = exp(fgbk) / (*diel_ext);
   temp6 = qiqj × temp4 × (dielfac+ fgbk × expmkf);
      ⋮
  }
 }
}
```

Fig. 15 Code snippet in the *egb* function of NAB after optimization

图 15 NAB 的函数 *egb* 优化后的程序片段

磁学程序中常见的计算模式. Fotonik3D 使用麦克斯韦方程组的有限差分时域（FDTD）方法计算光子波导的传输系数. 在本文的评测中，Fotonik3D 使用 SPEC 套件中提供的 ref 输入数据集进行评测. 由于 Fotonik3D 在 x86 和 ARM 上检测的冗余零含量类似，本文仅给出 x86 上的 DrZero 的冗余零检测结果. DrZero 的以代码为中心的冗余零检测结果表明，Fotonik3D 的执行过程中含有 8.76% 的整型冗余零以及 32.30% 的浮点冗余零. 通过进一步分析发现，其中数组 ***Ex***, ***Ey***, ***Ez*** 相关的计算都含有大量的完全冗余零. 具体情况如图 16 中左图所示，在计算过程中，数组 ***Ex***, ***Ey***, ***Ez*** 由大量完全冗余零构成，且其中大量 *x*-*z* 平面数据往往为 0，即该数组是结构化稀疏的. 因此，稀疏数据使用稠密数据结构以及稠密算法是造成 Fotonik3D 应用程序中大量完全冗余零及其相关无用计算的根本原因.

为了优化并消除冗余零造成的冗余计算，本文针对该应用数据特点对结构化稀疏数据结构以及对应的算法进行设计. 如图 16 中右图所示，原 ***Ex***, ***Ey***, ***Ez*** 数组的三维数据存储格式由原来的 (x, y, z) 的存储顺序改为按照 (x, z, y) 顺序存储，使得 *x*-*z* 平面在内存中得以连续存储；此外，数组稀疏存储格式按照类似稀疏矩阵格式[31]进行存储，即数组中仅存储非零 *x*-*z* 平面，并将对应 *y* 轴方向上的坐标值记录在下标向量 $\boldsymbol{Index}_y$ 中. 经过本文所提出的优化，Fotonik3D 在 x86 平台上实现 1.76 倍的性能加速，在 ARM 平台上实现 2.12 倍的性能加速.

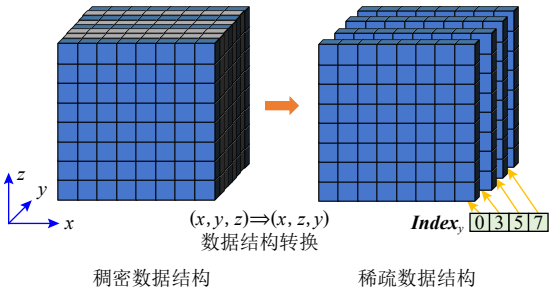

Fig. 16 Large fraction of redundant zeros in Fotonik3D

图 16 Fotonik3D 中含有大量冗余零

5 结束语

本文提出了一个针对冗余零的跨平台细粒度性能分析工具 DrZero. 为了适配访存、计算指令分离的 ARM 指令集，本文提出了基于数据流分析的数据类型推断方法来自动推断访存指令读取内存数据的数据类型. 此外，为了更低的性能开销，本文也提出了在线细粒度缓存迹分析来检测、记录冗余零相关的指标. DrZero 也提供了基于 VSCode 的可视化插件来展示冗余零检测报告以及相应的优化建议. 本文的实验展示了 DrZero 跨平台检测冗余零的能力. DrZero 在以代码、数据为中心的冗余零分析中，分别以 x86 和 ARM 平台 45.31 倍、54.20 倍和 14.12 倍、13.40 倍平均性能开销检测冗余零并给出优化建议. 基于 DrZero 给出的性能优化指导，本文优化的应用程序在 x86 和 ARM 上分别达到了最高 1.76 倍和 2.12 倍的性能加速. DrZero 的实现代码已经开源：https://github.com/buaa-hipo/ZeroSpy-drcctprof.

在未来，我们认为还有 3 个问题需要解决. 首先，在 ARM 平台上数据对象信息采集需要过大的内存开销，这导致运行使用较多内存的应用无法使用以数据为中心的分析模式进行分析，从而错失一些潜在的优化机会. 其次，本文发现的 x86 和 ARM 平台之间同一代码下冗余零的发散现象需要进一步研究其原因，以便挖掘更多的性能优化机会. 最后，希望将

来更多的ARM平台(例如天河三号原型机中使用的国产ARM处理器FT2000+)中测试本文提出的DrZero工具来发现更多的性能优化机会.

作者贡献声明：游心提出研究思路，设计实现研究方案，撰写论文；杨海龙负责论文起草以及最终版本修订；雷克伦负责采集实验数据并实现可视化界面；孔祥浩负责实现ARM平台部分功能；徐筠、栾钟治、钱德沛负责最终版本修订.

参 考 文 献

[1] Su Pengfei, Wen Shasha, Yang Hailong, et al. Redundant loads: A software inefficiency indicator[C]//Proc of the 41st Int Conf on Software Engineering (ICSE). Piscataway, NJ: IEEE, 2019: 982−993

[2] Lepak K M, Lipasti M H. On the value locality of store instructions[C]//Proc of the 27th Int Symp on Computer Architecture. New York: ACM, 2000: 182−191

[3] Chabbi M, Mellor-Crummey J. DeadSpy: A tool to pinpoint program inefficiencies[C]//Proc of the 10th Int Symp on Code Generation and Optimization. New York: ACM, 2012: 124−134

[4] Wen Shasha, Chabbi M, Liu Xu. RedSpy: Exploring value locality in software[C]//Proc of the 22nd Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2017: 47−61

[5] Wen Shasha, Liu Xu, Chabbi M. Runtime value numbering: A profiling technique to pinpoint redundant computations[C]//Proc of the 24th Int Conf on Parallel Architecture and Compilation (PACT). Los Alamitos, CA: IEEE Computer Society, 2015: 254−265

[6] Lee B, Jung J, Kim M. An all-zero block detection scheme for low-complexity HEVC encoders[J]. IEEE Transactions on Multimedia, 2016, 18(7): 1257−1268

[7] Peng Kuoyou, Fu Shengyu, Liu Yuping, et al. Adaptive runtime exploiting sparsity in tensor of deep learning neural network on heterogeneous systems[C]//Proc of the 17th Int Conf on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS). Piscataway, NJ: IEEE, 2017: 105−112

[8] Delmas Lascorz A, Judd P, Stuart D M, et al. Bit-tactical: A software/hardware approach to exploiting value and bit sparsity in neural networks[C]//Proc of the 24th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2019: 749−763

[9] You Xin, Yang Hailong, Luan Zhongzhi, et al. ZeroSpy: Exploring software inefficiency with redundant zeros[C]//Proc of the Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2020[2021-02-22].https://ieeexplore.ieee.org/document/9355303

[10] Perf Wiki. perf: Linux profiling with performance counters[EB/OL]. 2006[2021-07-29].https://perf.wiki.kernel.org/index.php/Main_Page

[11] Adhianto L, Banerjee S, Fagan M, et al. HPCToolkit: Tools for performance analysis of optimized parallel programs[J]. Concurrency and Computation: Practice and Experience, 2010, 22(6): 685−701

[12] Reinders J. VTune Performance Analyzer Essentials[M]. Danvers, MA: Intel Press, 2005

[13] Graham S L, Kessler P B, McKusick M K. Gprof: A call graph execution profiler[J]. ACM SIGPLAN Notices, 1982, 17(6): 120−126

[14] DeRose L, Homer B, Johnson D, et al. Cray performance analysis tools[C]//Proc of the 2nd Int Workshop on Parallel Tools for High Performance Computing. Berlin: Springer, 2008: 191−199

[15] Levon J. OProfile [EB/OL]. 2002 [2021-07-29].https://oprofile.sourceforge.io/news

[16] Nakao M, Ueno K, Fujisawa K, et al. Performance evaluation of supercomputer Fugaku using breadth-first search benchmark in Graph500[C]//Proc of the IEEE Int Conf on Cluster Computing (CLUSTER). Los Alamitos, CA: IEEE Computer Society, 2020: 408−409

[17] YouXin.DrZero[CP/OL].2021[2021-07-28].https://github.com/buaa-hipo/zerospy-drcctprof

[18] Zheng Zhen, Zhai Jidong, Li Yan, et al. Workload analysis for typical GPU programs using CUPTI interface[J]. Journal of Computer Research and Development, 2016, 53(6): 1249−1262 (in Chinese)
(郑祯, 翟季冬, 李焱, 等. 基于CUPTI接口的典型GPU程序负载特征分析[J]. 计算机研究与发展, 2016, 53(6): 1249−1262)

[19] Calder B, Feller P, Eustace A. Value profiling[C]//Proc of the 30th Annual Int Symp on Microarchitecture. Los Alamitos, CA: IEEE Computer Society, 1997: 259−269

[20] Watterson S, Debray S. Goal-directed value profiling[C]//Proc of the 10th Int Conf on Compiler Construction. Berlin: Springer, 2001: 319−333

[21] Tan Jialiang, Jiao Shuyin, Chabbi M, et al. What every scientific programmer should know about compiler optimizations?[C]//Proc of the 34th ACM Int Conf on Supercomputing. New York: ACM, 2020[2020-06-29].https://dl.acm.org/doi/10.1145/3392717.3392754

[22] Luk C K, Cohn R, Muth R, et al. Pin: Building customized program analysis tools with dynamic instrumentation[J]. ACM SIGPLAN Notices, 2005, 40(6): 190−200

[23] Stephenson M, Babb J, Amarasinghe S. Bidwidth analysis with application to silicon compilation[J]. ACM SIGPLAN Notices, 2000, 35(5): 108−120

[24] Rubio-González C, Nguyen C, Nguyen H D, et al. Precimonious: Tuning assistant for floating-point precision[C]//Proc of the Int Conf on High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2013[2021-07-28].https://ieeexplore.ieee.org/document/6877460

[25] Bruening D, Amarasinghe S. Efficient, transparent, and comprehensive runtime code manipulation[D]. Cambridge, MA: MIT Press, 2004

[26] Zhao Qidong, Liu Xu, Chabbi M. DrCCTProf: A fine-grained call path profiler for ARM-based clusters[C]//Proc of the Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ : IEEE, 2020[2021-02-22].https://ieeexplore.ieee.org/document/9355248

[27] NASA Ames Research Center. NAS parallel benchmarks[EB/OL]. 1991[2021-07-28].https://www.nas.nasa.gov/software/npb.html

[28] Che Shuai, Boyer M, Meng Jiayuan, et al. Rodinia: A benchmark suite for heterogeneous computing[C]//Proc of the 12th IEEE Int Symp on Workload Characterization (IISWC). Piscataway, NJ: IEEE, 2009: 44−54

[29] Bucek J, Lange K D, Kistowski J. SPEC CPU2017: Next-generation compute benchmark[C]//Proc of the 9th ACM/SPEC Int Conf on Performance Engineering (ICPE). New York: ACM, 2018: 41−42

[30] Chabbi M, Liu Xu, Mellor-Crummey J. Call paths for pin tools[C]//Proc of the 12th IEEE/ACM Int Symp on Code Generation and Optimization. New York : ACM, 2014: 76−86

[31] Buluç A, Fineman J T, Frigo M, et al. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks[C]//Proc of the 21st Annual Symp on Parallelism in Algorithms and Architectures. New York : ACM, 2009: 233−244

You Xin, born in 1997. PhD candidate. His main research interests include high performance computing, performance analysis tools, compile optimization.

游　心, 1997 年生. 博士研究生. 主要研究方向为高性能计算、性能分析工具和编译优化.

Yang Hailong, born in 1985, PhD, associate professor. Member of CCF. His main research interests include high performance computing, distributed and parallel computing, computer architecture, deep learning compilation.

杨海龙, 1985 年生. 博士，副教授. CCF 会员. 主要研究方向为高性能计算、分布式和并行计算、计算机系统结构、深度学习编译优化技术.

Lei Kelun, born in 2000. Undergraduate. His main research interest includes performance analysis tools.

雷克伦, 2000 年生. 本科生. 主要研究方向为性能分析工具.

Kong Xianghao, born in 1999. Undergraduate. His main research interest includes high performance computing.

孔祥浩, 1999 年生. 本科生. 主要研究方向为高性能计算.

Xu Jun, born in 1984. Senior engineer. Her main research interest includes modeling and simulation of weapon equipment system.

徐　筠, 1984 年生. 高级工程师. 主要研究方向为武器装备系统仿真与建模.

Luan Zhongzhi, born in 1971. PhD, associate professor. His main research interests include distributed computing, high performance computing, parallel computing, computer architecture, cloud computing, and big data.

栾钟治, 1971 年生. 博士，副教授. 主要研究方向为分布式计算、高性能计算、并行计算、计算机系统结构、云计算和大数据.

Qian Depei, born in 1952. PhD, professor. Academician of Chinese Academy of Sciences. His main research interests include distributed computing, high performance computing and computer architecture.

钱德沛, 1952 年生. 博士，教授，中国科学院院士. 主要研究方向为分布式计算、高性能计算和计算机体系结构.