面向天河新一代超算系统通用处理器的性能分析工具集

冯文韬 栾钟治 杨海龙 钱德沛
（北京航空航天大学计算机学院 北京 100191）

摘 要 天河新一代超算系统是继天河2号后天河系列的新一代超算系统.该系统拟采用通用处理器配合加速器的混合异构架构,其中通用处理器采用ARM架构.目前,面向ARM架构处理器的性能分析工具仍不够完善,而面向新一代超算的性能分析工具更是较为匮乏,实用性和效率还难以满足编程人员的需求.本文针对天河新一代超算系统的通用处理器,设计开发了一套性能分析工具集,包含缓存冲突检测、伪共享检测和内存缺陷检测三个子工具.工具集可以在天河新一代超算系统的普通用户权限下分析系统单节点内以及数据并行性较高的多节点程序的性能问题,并可以解决程序的内存问题.本文使用min-write、缓存行对齐填充、线程访问隔离等多种性能优化策略来提高工具性能,采用以上策略的工具的运行时间可至多减少至原先的1/20,同时使用新颖的红区检测法和红区隐藏与恢复机制来降低工具报告的假错误率.本文还开发了配套的可视化界面,使用户可以对程序的性能分析数据进行可视化的分析和处理,提高了工具的实用性和易用性.工具对程序执行带来的额外时间开销是40～100倍,额外内存开销是100～200倍,正确性和实用性得以保证,可以提高天河新一代超算系统的编程效率和程序性能.

关键词 性能分析工具;天河新一代超算系统;伪共享检测;内存缺陷检测;程序优化
中图法分类号 TP302 **DOI号** 10.11897/SP.J.1016.2024.00423

A Set of Performance Profiling Tools for the General Purpose Processors of TianHe New Generation Supercomputing System

FENG Wen-Tao LUAN Zhong-Zhi YANG Hai-Long QIAN De-Pei
(*Department of Computer Science and Engineering*, *Beihang University*, *Beijing* 100191)

Abstract TianHe new generation supercomputer system is a new generation of supercomputer system in the TianHe series after TianHe-2. The system is expected to adopt a hybrid heterogeneous architecture of general processor and accelerator, in which the general purpose processor adopts ARM architecture. At present, performance profiling tools for ARM architecture are still not perfect, and those for new generation supercomputers are even more scarce, and their practicability and efficiency are still difficult to meet the needs of programmers. For the general purpose processors of TianHe new generation of supercomputing system, this paper designs and develops a set of performance profiling tools, which contain cache conflict detection, false sharing detection and memory defect detection. The tool set can analyze the performance problems of the system's single node and the multi-node programs with high data parallelism, and solve the memory problems of the programs under the authority of ordinary users

收稿日期:2023-02-16;在线发布日期:2023-12-08. 本课题得到国家自然科学基金面上项目(62072018)资助.**冯文韬**,博士研究生,中国计算机学会(CCF)学生会员,主要研究领域为高性能计算、性能分析和计算机体系结构. E-mail: went_feng@buaa.edu.cn. **栾钟治**(通信作者),博士,副教授,中国计算机学会(CCF)高级会员,主要研究领域为分布式系统资源管理、异构计算系统应用软件栈、高性能计算性能分析和优化等. E-mail: luan.zhongzhi@buaa.edu.cn. **杨海龙**,博士,副教授,中国计算机学会(CCF)高级会员,主要研究领域为并行和分布式计算、高性能计算、性能优化和功耗优化等. **钱德沛**,教授,中国科学院院士,中国计算机学会(CCF)会士,主要研究领域为分布式计算、计算机体系结构、高性能计算等.

of TianHe new generation supercomputer system. Specially, the performance problems mentioned in this paper are mainly about the cache, which is always invisible to the programmers. This fact leads our work to great significance because the performance problems caused by cache are hard to disclose by programmers themselves only checking their codes. The memory defect detection tool proposed by this paper is able to detect five sub-problems including accessing invalid/illegal address space, use-after-free problem, read uninitialized space, double-free problem and memory leak problem. In this paper, a variety of performance optimization strategies such as min-write, cache line alignment fill, and thread access isolation are used to improve the tool performance, which can achieve 1.2 to 20 times faster than the unoptimized tool. Meanwhile, the novel red-zone detection method and red-zone hiding and recovery mechanism are used to reduce the false error rate reported by the tool. The red zone detection method is to set the red zone at the end of the memory allocation space to detect memory access errors. The design idea of this method comes from the summary of the common pattern that programmers write code, usually array bounds are concentrated in the array boundary. The purpose of the red zone hide and recover mechanism is to avoid false errors during continuous memory allocation and further reduce the false error rate generated by the tool. This paper also developed a supporting visual interface, users can perform visual analysis and processing of the program performance analysis data, improving the utility and usability of the tool. In the experiment, we use our tool-set to find a severe cache contention phenomenon in OCEAN-ncp in SPLASH-3, a famous parallel benchmark suite, which reveals a huge hidden optimizing opportunity. Later, we use the false-sharing detection tool to pinpoint the exact context also the line numbers in source code where the false-sharing happens and incurs great performance degradation. By gathering these information together and thoroughly exploiting this opportunity, we achieve a 3x speedup of parallel program OCEAN-ncp. The tools' time cost and space cost are about 40～100x and 100～200x. The tools have moderate overhead, correctness and practicability, which can improve the programming efficiency and program performance of TianHe new generation supercomputer system.

Keywords performance profiling tools; TianHe new generation supercomputing system; false sharing detection; memory defect detection; program optimization

1 引 言

随着近年来摩尔定律的失效,多核处理器成为提高处理器性能的一个主要手段.理论上,程序在多核处理器上获得的运行加速比应该等同于多核处理器所拥有的核心数量.但是现实情况并非如此,大部分并行程序并不能获得与核心数量成线性的加速效果.有时,甚至会出现线程数越多程序运行越慢的反常现象.这些性能问题的原因有很多,如同步、硬件资源竞争和缓存一致性等.本文工作首先关注于解决缓存的低效行为所导致的性能损失.缓存低效行为主要包含三大类:缓存冲突、长距离通信开销以及伪共享问题.缓存冲突是缓存缺失(cache miss)和缓存失效(cache invalidation)的统称,其中缓存失效是多核体系结构下特有的低效行为;长距离通信开销是由缓存失效问题引发的后续问题,发生缓存失效时需要更新缓存行,若两个线程处于两个通信开销较大的物理核上则更新操作需要浪费较多时间,造成性能损失;伪共享问题是著名的多核性能问题,即不合理的数据存放导致不必要的缓存失效所引发的性能损失.

这些缓存低效行为都会影响超算系统的性能,降低程序的运行效率.同时,内存错误具有滞后性、隐蔽性、复现性低等特点,也是经常困扰编程人员的一大问题,使得在超算系统上进行程序调试较为困难.

目前天河新一代超算系统还处于实验阶段,为了在正式投入使用后降低用户的调试难度、提高用户的平台体验,研发一套面向系统的通用处理器的缓存低效行为和内存缺陷的性能分析工具具有相当的必要性,也是天河新一代超算系统的迫切需求.

这样的一套工具也可以帮助开发人员更充分地利用天河新一代超算系统的性能.

现有的分析工具由于第二节中所介绍的原因无法在天河新一代超算系统上运行,因此本文设计实现了一套专门面向天河新一代超算系统通用处理器的性能分析工具集,包括检测缓存冲突、伪共享问题和内存缺陷三个功能模块. 工具采用运行时检测的方法,该方法准确率更高且分析粒度更细致,有助于更好地定位和解决性能问题. 工具基于DrCCTProf框架[1]实现,该框架是首个支持在ARM架构下提供详细指令调用信息的细粒度分析框架,这使得工具可以提供详细的错误报告. DrCCTProf框架在DynamoRIO[2]的基础上提供了上下文调用树(Context Calling Tree, CCT),使得定位问题所在的源代码更加方便. 在工具自身性能优化方面,工具使用了min-write、缓存行对齐填充和线程访问隔离等多种优化策略,整体性能良好,工具对程序执行所带来的额外时间开销平均为60倍,额外内存开销平均在120倍,使用工具检测并进行优化后的程序较原版加速4倍.

本文的主要贡献有以下几点:

(1) 对DrCCTProf框架进行了改进移植,使其适配天河新一代超算系统的分析需求;

(2) 提出了红区检测法和红区隐藏与恢复机制,提高内存缺陷检测效率的同时降低了假错误率;

(3) 使用min-write、缓存行对齐填充和线程访问隔离等性能优化策略实现工具,大大降低了工具的开销;

(4) 开发了配套的GUI,方便用户使用工具.

接下来的文章结构如下:第2节介绍相关工作;第3节将介绍面向天河新一代超算系统架构的DrCCTProf框架改进;第4节详细介绍了工具的检测原理、设计架构和工程实现;第5节将展示实验结果并对实验结果加以分析;第6节将对本文工作进行总结和展望.

2　相关工作

目前已有的关于伪共享问题检测和内存缺陷检测的工作如表1所示,SHERIFF修改pthread库的实现,将线程变为操作系统中的进程[3];Zhao等人的工作使用运行时位图检测法,可以报告程序中存在的缓存冲突、伪共享等信息[4];PREDATOR通过一次运行的数据,可以预测程序中潜在的伪共享问题[5];LASER是基于Intel的Haswell架构所提供的硬件数据实现的轻量级伪共享检测和修复框架[6];Cheetah和PerfMemPlus均基于PMU(Performance Monitoring Unit)数据,报告出潜在的修复位置,为编程人员提供详细的指导[7-8]. Pluto和CM-SIMICS都是使用运行时收集的数据,在运行后进行计算和模拟来给出结果[9-10].

Memcheck和Dr. Memory两个工具都基于动态二进制插桩工具来追踪每一个访存指令,进而通过访存地址和目前的地址状态进行比较来得出结果[11-12];Mudflap和CETS通过分析编译器的中间代码并进行插桩,可以收集到二进制文件中缺少的程序语义信息,帮助工具更好地分析数组越界访问等内存问题[13-14];AddressSanitizer则通过插桩LLVM的中间表示,并结合影子内存(shadow memory)可以更全面更高效地检测常见的内存问题[15].

表1　现有工作总结

工具	类别	工具名称	粒度	时间	支持架构
伪共享检测工具	运行时	SHERIFF[3]	线程级	2011	X86
		Zhao[4]	指令级	2011	X86
		PREDATOR[5]	编译级	2014	LLVM
		LASER[6]	硬件事件	2016	X86
		Cheetah[7]	硬件事件	2016	\\
		PerfMemPlus[8]	硬件事件	2019	\\
	运行后	CM-SIMICS[9]	\\	2007	\\
		Pluto[10]	\\	2009	X86
内存缺陷检测工具	运行时	Mudflap[13]	编译级	2003	GCC
		Memcheck[11]	指令级	2005	ALL
		CETS[14]	编译级	2010	LLVM
		Dr. Memory[12]	指令级	2011	X86
		AddressSanitizer[15]	编译级	2012	LLVM

值得注意的是，以上提到的全部运行时检测工具除Memcheck在最新版本中刚完善ARM架构的支持外均不支持ARM架构，也就无法在天河新一代超算系统上使用，而Memcheck提供的检测范围和运行环境需求是制约其在系统上应用的重要原因. 除了Cheetah和PerfMemPlus这两个使用PMU数据的工具外，其余工具在设计之初便没有考虑ARM架构，仅支持X86架构. 而使用PMU数据的工具由于需要使用Perf来收集系统数据，因此存在一定的系统安全风险. 天河新一代超算系统对安全性要求较高，因此对这类数据收集行为权限限制较多，导致工具无法在普通用户权限下运行.

下面针对本文所参考和改进的Zhao等人的伪共享检测工具[4]和Dr. Memory内存缺陷检测工具[14]二者的优点和不足进行剖析. Zhao等人提出的位图检测方法可以很方便地实现运行时检测，带来的开销也较小. 虽然工具没有使用完整的缓存模拟机制，但其正确率仍是令人满意的. 它的缺点是无法区分冷缺失(cold miss)和其他缺失、最多只支持8个线程以及对影子内存(shadow memory)不进行线程安全的保护. 这三个问题会导致其工具的实用性较低且正确率也会受到线程访问不安全的影响. Dr. Memory工具提供了堆栈空间检测和内存泄露检测两大功能，它的优点是适用范围广、性能较好且假错误率较低. 它最严重的缺点是目前无法支持ARM架构，使得本文所面向的天河新一代超算系统无法运行该工具. 其他的缺点还有(1)实现机制复杂，需要实时监控栈帧的变化以及模拟系统调用的影响；(2)结果反馈仅提供发生错误的位置，而没有变量创建位置的提示，使得对工具所报告的问题进行修复的难度较高、体验较差.

本文设计的工具集基于DrCCTProf这一运行时细粒度程序分析框架，利用位图检测法和影子内存实现了缓存冲突和伪共享问题的检测；利用堆空间状态划分和创新性的红区检测法实现了高效的内存缺陷检测功能. 在天河新一代超算系统的普通用户权限下，工具可以分析在ARM架构的通用处理器上执行的二进制文件.

3 面向天河新一代超算系统架构的DrCCTProf框架改进

DrCCTProf框架是Zhao等人[4]提出的一个面向ARM架构，也支持X86架构的性能分析框架. 它是基于DynamoRIO——一个动态二进制插桩工具实现的. 它可以提供每一条指令的完整上下文调用树(CCT)，还可以将访存指令和其操作的数据对象联系起来. 这些特点使得该框架适合用于开发运行时细粒度分析工具.

天河新一代超算系统是我国最新一代的超算系统，拟采用混合异构架构. 它既有CPU+GPDSP类型的节点也有纯CPU类型的节点，系统会根据程序的不同特点分配最适合的节点去执行代码. 本文研发的工具适用于纯CPU类型的节点，对于CPU+GPDSP类型的节点也可以运行工具，但工具仅分析CPU端运行的情况而无法分析加速器内部的运行情况. 如图1所示为Lu等人[16]论文中介绍的天河新一代超算系统通用处理器节点可能的设计结构，文中将CPU部分称为通用区域(general purpose zone). 通用区域包含16个CPU物理核，每个核拥有独立的L1缓存和512 KB的L2缓存，核组之间使用MESI缓存一致性协议；16个核互相连通且与4个大小为48 MB的HBSM(High Bandwidth Shared

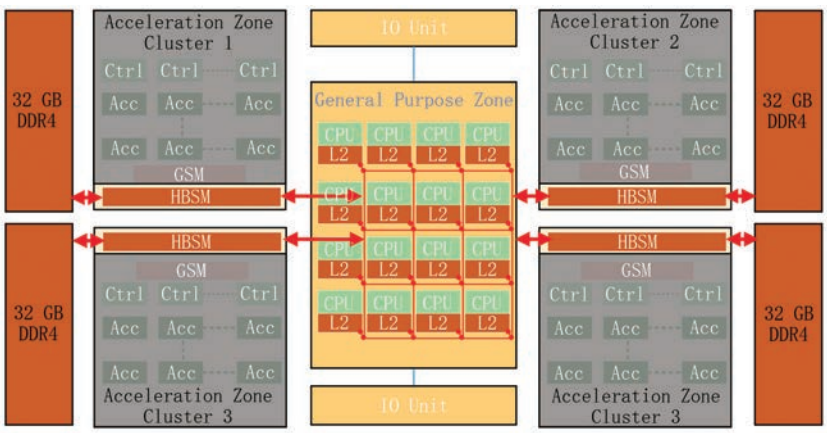

图1 天河新一代超算系统通用区域节点设计架构[16]

Memory，HBSM）相连，每一个HBSM外端连接一个32 GB的DDR4[16].

在该架构下，16个CPU物理核访问其他任何一个物理核的L2缓存的开销是大致相当的，且每一个核访问四个HBSM和四个DDR4的开销也是大致相当的. 这样就不存在因为内存层次架构和核组间通信距离的不同导致的访存开销的差异，使得本文的工具并不需要包含检测线程间交互情况的功能. 因为无论将交互较多的一对线程绑定到哪两个物理核上，通信带来的开销相差不多，最终对程序的性能影响也几乎可以忽略不计.

本文在使用DrCCTProf框架作为基础进行工具开发过程中，发现该框架存在一些与工具设计不适配的地方，接下来简要介绍框架存在的问题和相应的解决方案. 框架的整体结构以及每一点改进所处框架的层次结构如图2所示.

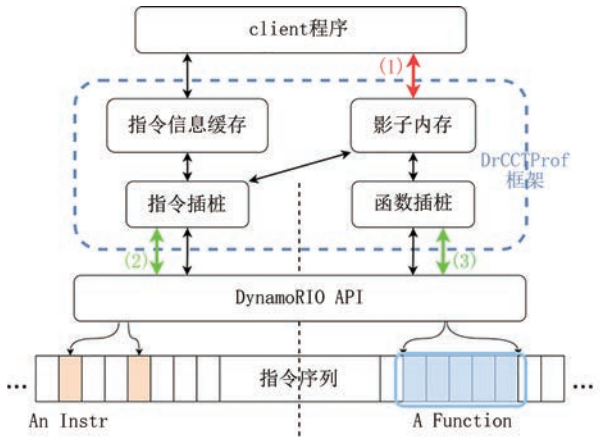

图2　DrCCTProf框架结构及相应修改示意图

（1）框架对影子内存（shadow memory）的操作接口不足. 为了解决Zhao等人工作[4]存在的无法区分冷缺失这一点不足，工具需要能够判断一个影子内存单元（Shadow Memory Unit，SMU）是否已经被分配，访问一个未分配的SMU即代表发生一次冷缺失. 影子内存单元（SMU）是对程序实际内存地址空间进行映射的最小单位，即一段实际内存空间的信息由一个SMU记录和一系列SMU组成整个影子内存. 本文工具设定SMU的大小为缓存行大小，即一个SMU结构体记录实际内存中长度为缓存行大小的一段地址的信息. 为了实现以上功能，本文为框架增加了三个操作影子内存的接口，分别用于初始化SMU、检测SMU是否已初始化和获取已初始化的SMU内容. 本文利用新增的接口实现了初始化探测机制，如图3所示. 该机制既可以实现区分冷缺失的功能，又可以节省大量内存开销，因为程序实际访问到的地址空间即需要分配SMU的空间仅占全部地址的一小部分.

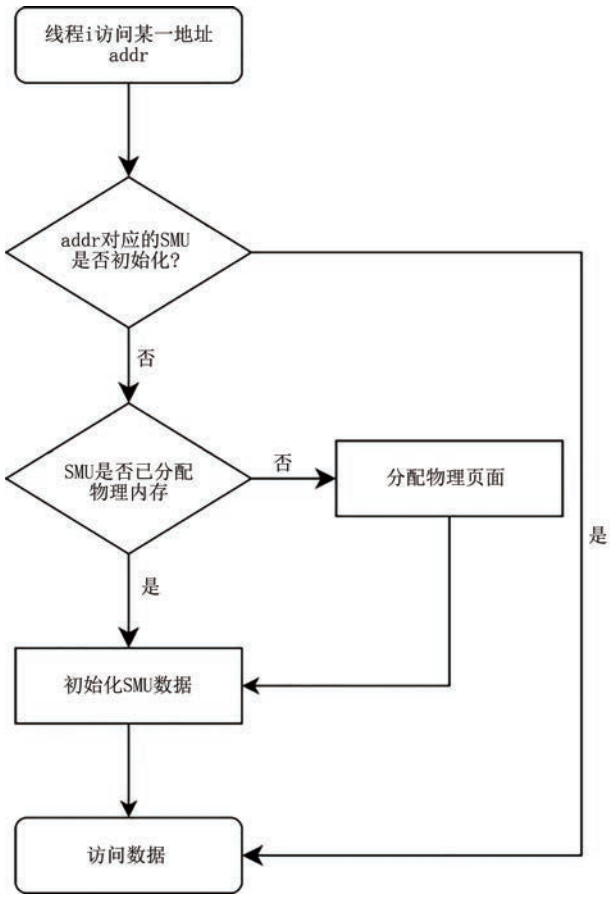

图3　初始化探测机制示意图

（2）框架收集的指令信息不够完善. 工具需要获取每一条访存指令的读写类型和访存数据的大小信息，而框架仅提供了指令的访存地址. 本文使用DynamoRIO提供的instr_get_src和instr_get_dst两个接口来获取指令的读写类型；使用DynamoRIO提供的drutil_opnd_mem_size_in_bytes接口来获取访存数据的大小. 获取到的信息使用MINSERT宏函数通过汇编指令存入相应的结构体中用于后续工具的计算和分析.

（3）框架对内存分配函数的监控粒度不够细致. 工具中的内存缺陷检测功能需要对malloc、calloc、realloc和free四个内存空间管理函数实现不同的监控函数，而框架仅提供最基本的同一种监控函数. 本文所实现的监控函数将会在4.2节中详细讨论.

4 性能分析工具设计原理与实现方案

本节对工具集包含的三个功能模块的检测原理，设计框架以及具体的实现方案进行介绍．伪共享检测模块是基于缓存冲突检测模块进行的扩充，因此将缓存冲突检测和伪共享检测两个模块合称为缓存低效行为检测一并介绍．工具使用了min-write、缓存行对齐填充和线程访问隔离等性能优化策略，降低了工具自身的开销，取得了较好的效果．针对内存缺陷检测假错误率较高的问题，本文创新性地提出了红区检测法和红区隐藏与恢复机制，极大地降低了假错误率，提高了工具的实用性．

4.1 缓存低效行为检测

4.1.1 整体架构设计

缓存低效行为检测工具基于Zhao等人[4]的设计思路，使用DrCCTProf框架进行实现，使之可以在天河新一代超算系统上运行并对Zhao等人设计的工具中存在的(1)无法区分冷缺失(cold miss)和其他缺失、(2)最多只支持8个线程、(3)对影子内存(shadow memory)不进行线程安全的保护，这三点不足进行改进．还提出了min-write、缓存行对齐填充和线程访问隔离三个策略来优化工具性能．

缓存低效行为检测的整体架构如图4所示，其中有两个核心的功能模块：改进后的DrCCTProf框架和client程序．框架为工具提供访问影子内存的接口以及收集指令信息的支持．client程序根据所要检测的两种缓存低效行为，分别实现两个具体功能程序，即缓存冲突检测的client程序和伪共享检测的client程序．

工具的整体工作流程是DynamoRIO先读取二进制的源代码并进行反汇编，将其按基本块为单位进行划分，每次加载一个基本块到Code Cache里面；DrCCTProf框架分析Code Cache中的指令，获取所需信息并存储下来；DrCCTProf框架分析结束后，会通知client程序，调用client程序来进行下一步的分析；client程序被激活后，对Code Cache中的指令逐条检测，所需的信息通过指令本身和DrCCTProf框架存储的结果来获取，其对影子内存的操作需要通过DrCCTProf框架提供的接口来进行；最终，分析完所有基本块后client程序给出结果报告．

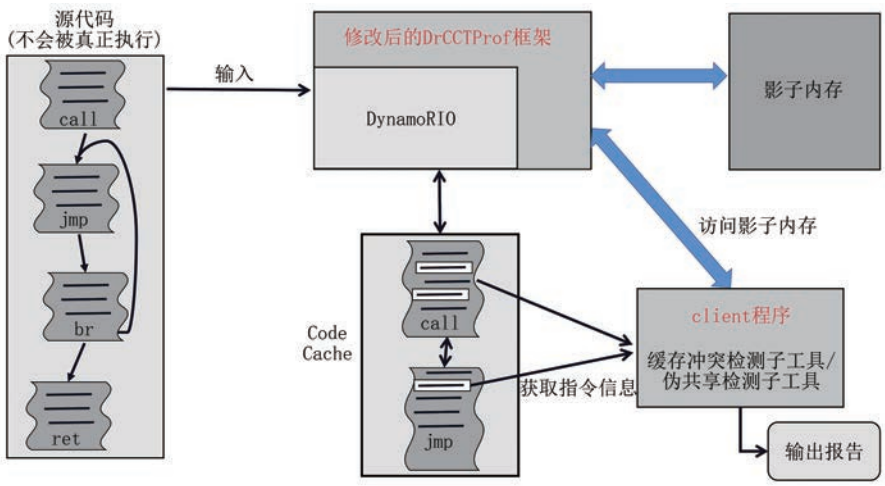

图4 缓存低效行为检测设计架构图

此处，对缓存低效行为检测提出两点简化假设：(1)假设不存在容量缺失；(2)假设物理核足够多，不会发生线程在物理核之间的迁移[4]．这两点假设给工具提供了良好的硬件独立性，即工具对缓存的替换策略、组织方式等微体系结构信息并不敏感，使得工具具有良好的通用性．这两点假设还有助于简化工具设计逻辑，且根据Lu等人论文[16]中所述天河新一代超算系统每个核拥有512 KB的L2私有缓存，全机可能拥有10万级的节点数，很好地满足了以上两点假设．在天河上运行的程序均由专业人员编写，因此工作集会经过严密设计以最大程度上减少容量缺失的发生，故做出第一点假设是符合实际情况的．当出现极少数的容量缺失时，检测到的缓存行为与实际缓存行为会存在少量的偏差，但并不影响整体结果的正确性和有效性．相较两点假设带来的优点，在少数情况下出现的轻微偏差应当是可以接受的．此处说明：由于假设不发生线程在物理核之间的迁移，为了简便，本文后续表述用“线程的

L2 Cache”代替“运行该线程的物理核的L2 Cache”.

4.1.2　缓存冲突检测模块

缓存冲突检测模块采用的运行时检测原理是位图(bitmap)标识法. 该模块对影子内存的使用和存储的SMU结构参考Zhao等人的思路[4],针对每一个缓存行,工具在影子内存中分配一块空间用于存储线程归属位图(ownership bitmap). 工具中设置的缓存行大小为16字(64字节),因为绝大部分机器的缓存行大小都为64字节. 本工具所设计的最大线程检测数为32个,因此位图大小为32 bit,其中每一个bit代表一个线程的归属属性. 若某缓存行对应的线程归属位图中的第i位为1,则说明该缓存行的副本存在于第i号线程的L2 Cache中;若其线程归属位图中的第i位为0,则说明第i号线程的L2 Cache中没有该缓存行的副本.

位图表示法的检测流程:当线程i读取一个地址时,检测该地址所在的缓存行对应的线程归属位图中第i位是否标记为1. 若为1,则说明该线程的L2 Cache中存在所读取地址对应的缓存行副本,没有发生缓存缺失;否则,发生一次缓存缺失. 当线程i写入一个地址时,检测该地址所在的缓存行对应的位图,是否有除了第i位外的其他位置1. 若为1,则说明其他线程的L2 Cache中也存在所写入地址对应的缓存行副本,发生缓存失效;否则,没有发生缓存失效. 对程序运行过程中的每一个访存指令进行如上的分析,即可得到整体的缓存缺失和缓存失效的数据.

缓存冲突检测client程序的具体实现流程如图5所示. (1)对采集到的访存地址进行缓存行地址对齐;(2)根据访存地址和访存数据大小,确定访存所涉及的所有缓存行;(3)判断该缓存行是否发生冷缺失,若发生冷缺失则记录数据并更新线程归属位图;(4)判断指令的读写类型,若为读指令转(5),若为写指令转(6);(5)判断是否发生了缓存缺失,若发生了则记录数据并更新位图;(6)判断是否发生了缓存失效,若发生了则记录数据并更新位图;(7)根据跨越的缓存行数目,循环执行(3)～(6).

缓存冲突检测模块的实现方案采用C++代码编写client程序,主要实现逻辑包括对读指令和写指令行为的判断两部分. 判断读指令是否发生缓存缺失的实现方案是使用按位与操作进行检测. 例如,某访存地址对应的缓存行的位图是0x00000001,那么说明仅0号线程的L2 Cache中,存在该访存地址对应的缓存行的副本. 当第i号线程访问该缓存行

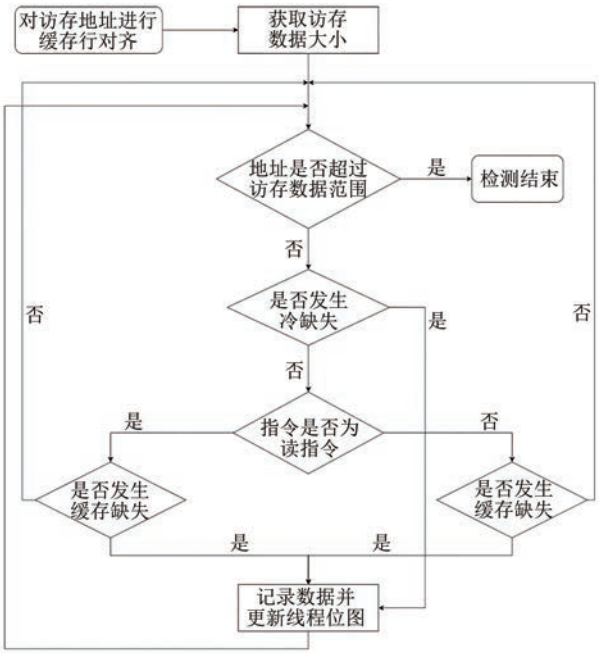

图5　缓存冲突检测client程序流程图

时,使用0x00000001 & (1 << i) > 0进行判断:若该不等式成立,则说明第i号线程的L2 Cache中存在该缓存行的副本,无需后续操作;若该不等式不成立,说明按位与的结果为0,则第i号线程的L2 Cache中不存在该缓存行的副本,发生缓存缺失.

判断写指令是否发生缓存失效的实现方案依旧是使用按位与操作. 例如,某访存地址对应缓存行的位图是0x00000005,那么说明仅0号和2号线程的L2 Cache中,存在该访存地址对应的缓存行的副本. 当第i号线程访问该缓存行时,使用0x00000005 & (～(1 << i) > 0进行判断:若该不等式成立,则说明除i号线程外,其他线程的L2 Cache中也存在该缓存行的副本,发生缓存失效;若该不等式不成立,说明按位与的结果为0,则说明仅i号线程的L2 Cache中存在该缓存行的副本,无需后续操作.

发生缓存缺失后,缺失的缓存行会被加载进入物理核的L2私有缓存中. 因此,若线程i发生了一次缓存缺失,那么在该次访存后,将该线程的位图第i位设置为1以更新位图,这一操作表明i号线程的L2 Cache中已存入该缓存行的副本. 实现方案是使用按位或操作,将原有的bitmap | (1 << i).

发生缓存失效后,失效的缓存行会被写回到下一层缓存,导致其它物理核内该缓存行的副本都变为无效的. 因此,若线程i发生了一次缓存失效,那么在该次访存后,将该位图中的每一位清零后,将其第i位设置为1. 表明除了线程i的L2 Cache中的缓

存行副本，其他副本都是无效的．实现方案是使用赋值操作，将原有的bitmap＝(1＜＜i)．

4.1.3 伪共享检测模块

伪共享检测在缓存冲突检测原理的基础上，额外使用读写更新法来实现功能．读写更新法更新的是缓存行中每一个字的读写线程记录．检测伪共享需要对其他线程的访问情况进行判断，因此需要对每一个线程的读写访问进行记录．该工具对影子内存的操作和SMU的结构参考Zhao等人的思路[4]，在缓存冲突检测的线程归属位图的基础上，对于缓存行中的每一个字(word)都需要额外分配两个位图，一个是读位图(read bitmap)，一个是写位图(write bitmap)．二者分别记录着读取和写入该字的线程号．

伪共享的判断流程：当发生一次缓存缺失时，可以通过检查相关字的位图，确定哪些线程之前更新过该字，如果除当前线程外其他线程的写位图的相应位没有置位，则发生一次伪共享；否则发生一次真共享．当发生一次缓存失效时，我们可以通过检查相关字的位图来确定哪些线程之前读或写过该字，如果除当前线程外，其他线程的读位图和写位图的相应位均没有置位，那么发生一次伪共享；否则发生一次真共享．对于一次写回(store)操作，需要清空当前缓存行对应的所有位图的所有置位，同时对所访问的字的写位图中对应当前线程的位进行置位．

伪共享检测client程序具体实现流程和缓存冲突检测的流程类似，都采用C＋＋代码编写一个单独的client程序．不同之处在于对伪共享数据的记录和读写位图的更新，如图6所示，图中标蓝部分是与缓存冲突检测流程不同的步骤．在前述的步骤(5)和(6)中都增加了对伪共享数据的统计和更新；步骤(5)和(6)中又分别增加了对读位图和写位图的更新操作．以上操作均通过C＋＋代码，进行线程局部变量的累加和影子内存数据读取更新．

由于工具需要输出发生伪共享时的上下文信息，在运行过程中就需要记录上下文信息．程序在实际运行中可能会出现以下三种情况：同一个变量可能会在不同的上下文处发生伪共享；不同的变量也可能会在同一个上下文处发生伪共享；同一个变量也可能会在同一个上下文处发生伪共享．为了能够区分并分别统计以上三种情况的数据，工具在client程序中创建了一个全局map．map的键(key)是将32位的变量创建信息和32位的伪共享发生的上下文信息进行拼接所组成的64位整数，如图7所

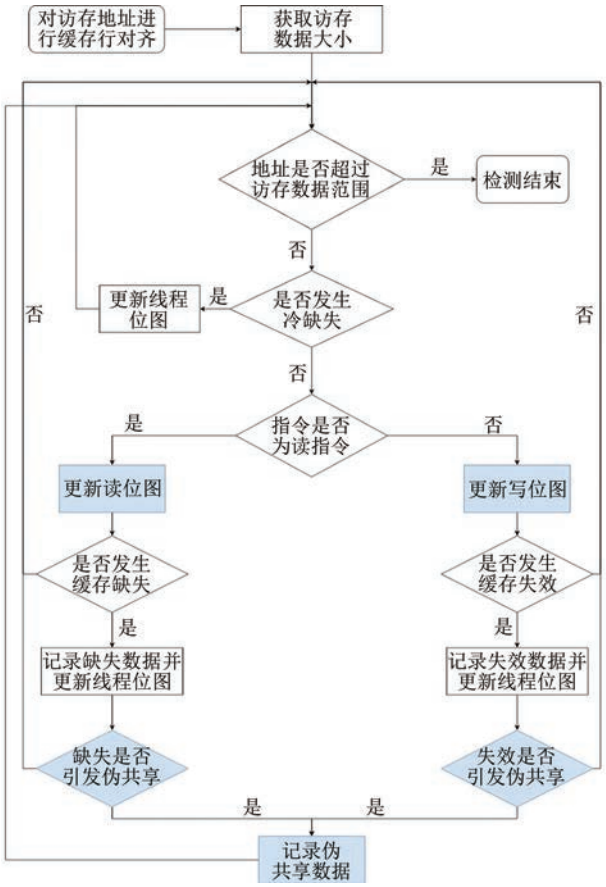

图6 伪共享检测client程序流程图

示．其中高32位存储的是发生伪共享时的上下文信息，低32位存储的是引发伪共享的变量的创建信息；值(value)存储的是该变量在当前上下文处发生的伪共享次数．这样的设计可以保证不是同一个变量在同一个上下文处发生的伪共享，工具都可以区分开并分别计数．

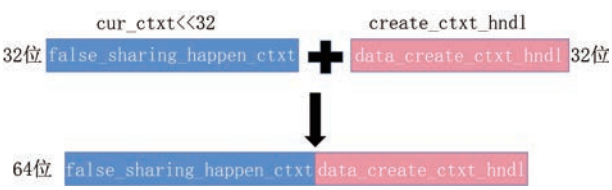

图7 键值构成图

为了方便用户获取全局信息，工具额外提供了两个全局视角的输出：(1)global-all-threads-info提供的是所有线程的整合数据，每一条结果按发生伪共享的次数进行降序排列；(2)global-all-threads-compact-info提供的是按变量分类的压缩视图．性能优化时，用户更多关注于对引发伪共享的变量进行优化而不会考虑该变量在具体的上下文处的伪共享行为，因此该视图将变量信息相同的输出结果进

行合并，对伪共享次数进行求和并丢弃发生伪共享时的上下文信息记录.

4.2　内存缺陷检测

4.2.1　整体设计思路

内存缺陷检测模块可以检测5类内存缺陷问题：访问非法/未定义的地址空间、访问释放(free)后的地址空间、使用已定义但未初始化的空间、两次释放(double free)问题和内存泄露(memory leak)问题. 所实现的工具与现有的成熟工具Dr. Memory相比[12]，可以支持ARM架构平台的检测、实现机制更简单且提供完整的调用树和变量创建信息.

工具需要对内存的每一个字节都使用影子内存存储相应的状态元数据，以标识该字节当前所处的状态. 如图8所示，堆空间的内存地址被划分为四种状态：(1) unaddr状态代表未使用malloc/calloc/realloc三个内存分配函数进行分配的内存地址空间，是非法的地址空间；(2) uninit状态代表已分配的空间，但未初始化任何值，是不安全的访问空间；(3) defined状态代表已分配并进行了初始化的空间，是合法的安全的访问空间；(4) freed状态代表之前已分配后又被free函数释放掉的空间，是不安全的访问空间.

工具所检测的内存主要是堆空间. 对于栈空间而言，除了栈空间不足外不会引发segment fault问题. 栈空间内的变量未进行初始化导致的影响作用范围并不大，一般限制在同一个作用域内，如一个函数、一个循环等. 因此为了提高工具性能并简化实现机制，工具不会对栈空间内的地址空间区分四种状态，仅标注栈空间对应的地址范围为defined. 如果工具未检测到其他内存问题，但程序发出SIGSEV信号，那么说明该段错误极大可能是由栈空间内的变量导致的. 通过这样的方式，该工具在避免Dr. Memory复杂的监控和模拟机制的同时，也保留了核心的堆变量检测功能.

图8　堆空间状态划分示意图

4.2.2　整体实现架构

内存缺陷检测工具的整体架构如图9所示，其中核心模块有两个：改进后的DrCCTProf框架和加载到框架上的client程序模块. 工具整体工作流程可以视为两个核心模块并行执行的过程，两个模块都需要实时监控加载到指令缓存块中的每一条指令.

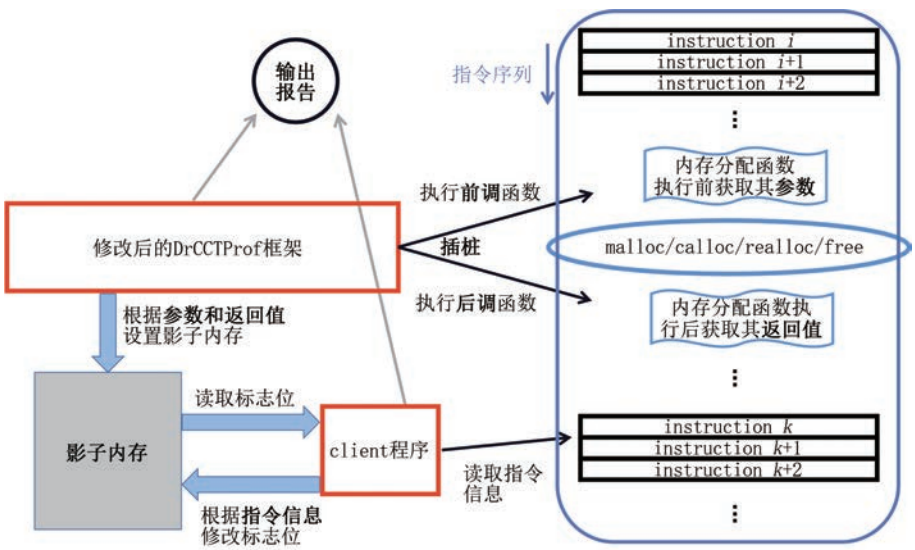

图9　内存缺陷检测设计架构图

对于改进后的DrCCTProf这一模块，它需要监控每一条指令，判断是否有malloc/calloc/realloc/free四个内存空间管理函数将要执行. 如果任何一个函数将要执行，就会触发框架的捕捉机

制：在函数执行前进行插桩，将实现的前调函数插入到内存空间管理函数之前，使得前调函数能够在内存空间管理函数真正执行之前执行，获取其传入的参数信息；在函数执行后也进行插桩，将实现的后调函数插入到内存空间管理函数之后，使得后调函数能够在内存空间管理函数执行后执行，获取其执行的结果即函数的返回值．框架在获取到内存分配函数的参数和返回值等信息后，还需要根据这些信息来更新影子内存．影子内存是框架和client程序唯一的交互方式，通过影子内存，框架可以实时通知client程序目前的内存分配情况．最后，通过新加入的alloc_map_now和alloc_map_history数据结构，框架可以记录内存分配的当前和历史信息来检测两次释放和内存泄露两个内存问题并输出错误报告．

对于client程序这一模块，它同样需要监控每一条访存指令，根据指令的访存地址去访问影子内存．client程序根据框架之前设置好的四种状态标志位来判断这一段内存空间的状态，并结合当前指令的读写类型来判断该条指令是否有非法的访存行为．在执行写指令时，client程序除了读取影子内存的内容之外，还会修改影子内存．如果写指令写入了一段uninit状态的地址，即已分配但是未经过初始化的地址，那么client程序将会修改该段地址对应的SMU的标志位，将该段地址的状态设置为defined，即已分配且已经过初始化的状态．

4.2.3 具体实现流程

下面介绍检测五类内存问题的具体流程：(1)访问非法地址空间的错误，client检测每一条访存指令，获取其访存地址对应的SMU信息，判断其数据类型是否为动态分配数据：若不是则忽略该条指令；若是则检测SMU中的alloc标志位是否为1，若不为1则发生错误；(2)访问未初始化空间的错误，client检测每一条访存指令，判断其读写类型：若为读指令则判断其SMU的init标志位是否为1，若为0则发生错误；若为写指令则更新标志位；(3)访问已释放空间的错误，client检测每一条访存指令，判断其访存地址对应的SMU中的freed标志位是否为1，若为1则说明发生错误；(4)两次释放问题，改进后的框架内部free函数的前调函数进行识别；(5)内存泄露问题，工具在整个程序运行结束后对alloc_map_now进行遍历，若其不为空则说明发生内存泄露，将map中的全部内容输出以提示用户发生内存泄露的位置．以上提及的alloc、init和freed三个标志位，均通过malloc、calloc、realloc和free函数的前后调函数访问相应的SMU进行设置．

下面详细介绍两次释放问题的检测过程：(1)free函数的前调函数首先获取其参数，即待释放的变量的地址；(2)若地址为NULL则输出提示信息，若不为NULL则去alloc_map_now中查找是否有相应的元素；(3)若有相应元素则代表该变量未被释放过，那么就将该元素从alloc_map_now转移到alloc_map_history中，如图10所示．之后刷新该段内存所对应的影子内存，将SMU的freed标志位置为1；(4)若没有相应元素则代表该变量已经被释放过或者从未申请过空间，则前往alloc_map_history中查找是否有相应元素；(5)若有则发生了两次释放问题，将该地址对应的全部变量信息输出，说明这些变量中的某一个引发了问题，进而可以缩小排查范围；(6)若没有则输出提示信息，表明释放的是一个未申请过空间的变量．

4.2.4 假错误消除策略

在实现的过程中我们发现：若使用原有DrCCTProf框架提供的数据类型进行判断——将全部Unknown类型的地址认定为非法的内存空间，则会产生数以十万计的假错误报告．因为程序会访问很多内核地址空间，而这些地址全部都是Unknown类型的．为了解决该问题，本文提出红区检测法，如图11所示：该方法仅考虑通过动态内存分配申请的内存空间，判断其SMU的alloc标志位

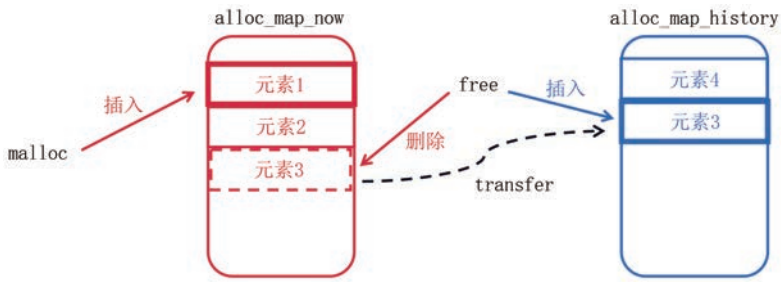

图10 free函数的前调函数操作示意图

是否为1. 若不为1，则说明程序访问到了红区，是非法的内存空间，报告这起错误. 本文提出这种检测方案是考虑到通常编程人员对malloc/calloc/realloc申请的地址的错误访问集中于数组越界访问，且通常越界不会太多，一般仅会访问边界元素. 为了降低工具假错误率的同时提高工具效率，工具仅监视程序是否访问了已在堆中申请的变量空间附近的非法空间，而忽视访问其他部分地址的指令的行为.

进一步研究发现，新的一次malloc函数执行，会访问上一次malloc函数分配的空间的后续一段地址. 该现象会导致多次执行malloc时，工具报告malloc函数自身访问了非法的地址空间，但非法的空间其实只是之前执行完的malloc函数的红区，这也是一种假错误. 为了避免这样的情况产生，工具引入了红区隐藏与恢复机制，如图12所示. 对于隐藏机制，工具会抓取每一个malloc/calloc/realloc函数，在其前调函数，中暂时地将上一次内存分配区域结尾的一段红区的alloc和init标志位修改为1，表明该段地址暂时是合法的，以便当前执行的malloc函数访问时不会报告错误；对于恢复机制，在其后调函数中，再将上一次分配的内存空间地址的红区中，没有被新分配的空间所覆盖的部分的alloc和init标志位恢复为0. 如图12中的(a)、(b)、(c)所示，分别代表新分配的起始地址在红区之后、新分配的起始地址与红区有重合部分和新分配的起始地址在红区之前三种情况. 根据新分配地址与红区之间相对位置的不同，对红区进行不同形式的恢复，以保证后续检测的正确性. 这样就可以进一步减少由malloc函数引发的假错误.

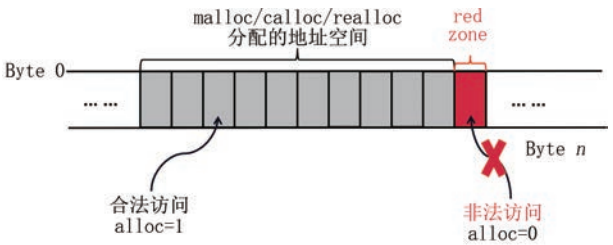

图11　红区检测法示意图

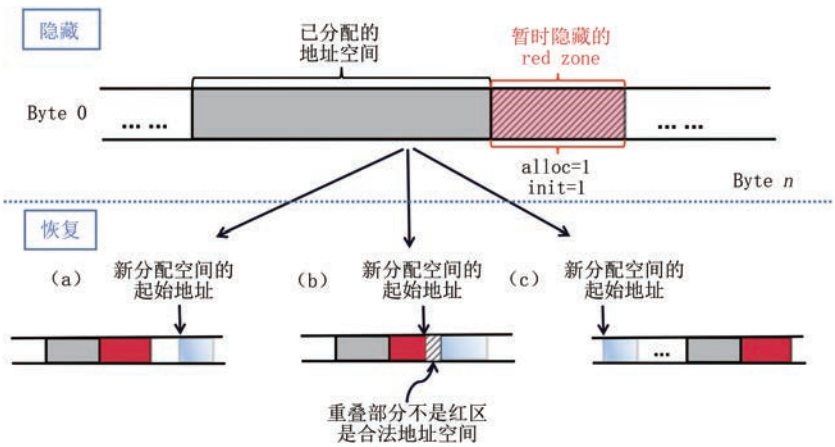

图12　红区隐藏与恢复机制示意图

为了进一步减少呈现给用户的无效信息，本文设计了一个输出过滤器来减少用户无法解决的错误报告. 过滤器的设计较为简单，过滤器会遍历将要输出的上下文信息，过滤掉前三条调用情况中源代码行数均为0的输出. 因为这是用户无法解决的问题，所以不必输出给用户. 用户需注意过滤器的使用需要结合-g编译，保证二进制代码中包含调试信息.

4.3　工具性能优化策略

(1)min-write策略[4]. 频繁地写入影子内存也会带来额外的缓存失效，导致工具自身运行效率降低，因此本工具采用min-write的思路，即尽可能地减少写入影子内存的次数. 这可以通过设置test环节来解决. test方法的核心思路是先检测进行修改的必要性，再进行必要的修改. 对于每次的指令读写操作，首先读取要修改的变量，只有在旧值与新值不同时，才需要做出后续的写入操作来更新变量的值. 由于读的开销远远小于写的开销，将大部分的写入改为读取就可以大大提高工具自身的性能.

(2)缓存行对齐填充(padding)策略. 为了进一步避免读写影子内存发生的缓存冲突，本工具使用了填充的方式来保证每个SMU的大小是缓存行对齐的.

(3)线程访问隔离策略. 为了避免工具本身进

行数据统计时引起过多的缓存冲突问题，工具收集全局数据时采用线程内部独立采集，最后全局汇总的方式，如图13所示．这使得线程内部采集数据时可以使用线程私有变量而不会共享访问全局变量．

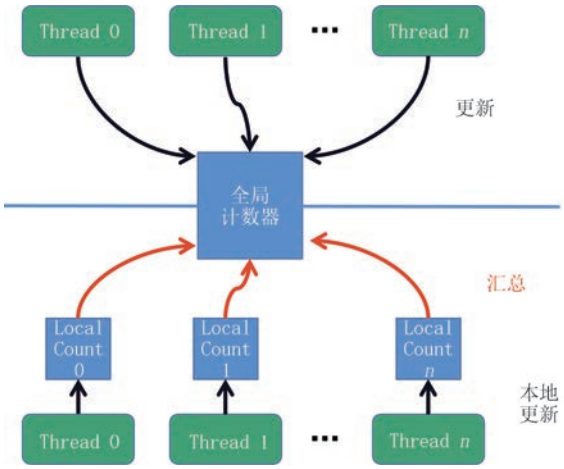

图13 线程访问隔离策略示意图

5 实验结果与分析

5.1 实验环境

本次实验使用飞腾CPU计算节点，每个节点拥有16个物理核，每个核拥有512 KB的L2缓存，缓存行大小为64字节．16个物理核共享128 GB的DDR4内存．下述的实验测试集和工具均使用天河新一代超算系统的普通用户权限运行和测试．

实验所使用的测试集融合了三个经典测试集：SPLASH3[17]、NPB①和PARSEC-3.0[18]，它们都是为测试多核体系结构而开发的并行程序集．本文根据实际情况从每个测试集中选取部分程序加入本文实验测试集，测试程序共13个，如表2所示．SPLASH3是在SPLASH2的基础上针对当代共享内存的多核处理器架构进行了一定的修改，以保证内存一致性并消除了数据冒险．因为其输入数据规模较小，所以其中的Barnes、Cholesky、Fmm、Lu、Raytrace、Volrend和Water程序的运行时间过短，波动较大，故排除在本次实验数据集之外．NPB是美国航空航天局(NASA)开发的一套测试超算性能的测试集，拥有不同的输入数据大小，本文选取其全部5个核心程序加入测试集．PARSEC是2008年由Bienia等人[18]提出的，它针对于新的多核内存架构，拥有不同输入数据大小．由于其中部分程序受硬件环境限制无法运行，本文选取可以运行的部分程序加入测试集．

表2 测试集程序汇总表

数据集名称	具体测试用例名称	功能描述
NPB-OMP	EP	并行浮点计算
	MG	多网格计算，非本地内存访问
	CG	共轭梯度计算，不规则访存和通信
	FT	离散三维快速傅里叶变换，密集的远距离通信
	IS	整数排序
SPLASH3	OCEAN(cp)	大规模洋流运动模拟(网格连续划分)
	OCEAN(ncp)	大规模洋流运动模拟(网格不连续划分)
	radiosity	场景中光的平衡分布模拟
	fft	离散一维快速傅里叶变换
	radix	基数排序
PARSEC-3.0	blackscholes	欧洲期权价格分析
	bodytrack	追踪人体姿态的计算机视觉模型
	fluidanimate	不可压缩流体的交互式动画

根据表2中的功能描述一列可知，本文实验所选取的测试用例广泛覆盖各个领域，例如物理、金融、计算机视觉和数值计算等．这些测试程序涉及的计算类型全面，具有相当的代表性．

5.2 工具性能分析

该部分对工具的三个功能模块即子工具：缓存冲突检测、伪共享检测和内存缺陷检测工具进行性能评测实验，分析其时间、内存开销．此时，工具处于完全优化状态，即实验获得的时间和内存开销是在工具使用三种优化策略后的最优情况．进一步，对4.3节中提到的min-write策略、缓存行对齐填充策略和线程访问隔离策略3种优化方法进行消融试验，证明其对工具性能提升的有效性．最后，对工具的可扩展性进行验证．

实验使用表2中所列的13个测试程序，程序运行的线程数量均为16线程．NPB测试集的输入大小除FT外均为CLASS B，由于内存空间有限，FT程序的输入使用CLASS A；SPLASH3测试集程序的输入大小均使用所提供输入的最大值；PARSEC-3.0测试集程序的输入大小均为simlarge．

5.2.1 时间开销(Time Cost)分析

该部分程序运行时间的获取通过Linux提供的gettimeofday函数完成．实验结果如图14所示，结果

① NAS Parallel Benchmarks，https://www.nas.nasa.gov/index.html，2022-05-23

表明伪共享检测子工具在大部分测试集上的时间开销为缓存冲突检测子工具的3倍. 这是因为伪共享检测子工具需要在影子内存中对缓存行中的每个字进行记录，同时还需要进行全局信息的汇总. 缓存冲突检测与伪共享检测的时间开销变化趋势相同，这是因为发生伪共享次数较多的程序其存在的缓存冲突问题也较为严重；而内存缺陷检测的时间开销变化趋势与前两者无关，表明内存缺陷问题和缓存低效行为是相互独立的. 从整体趋势来看，程序存在的问题越多，其花费的检测时间越长.

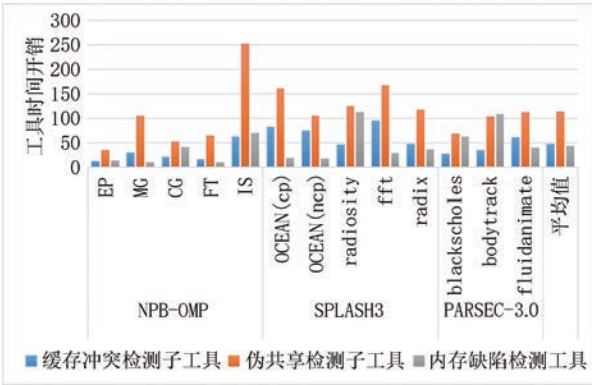

图14　工具的时间开销

缓存冲突检测和内存缺陷检测工具的平均时间开销为40倍，伪共享检测子工具的平均时间开销为110倍. 相较于在X86架构上对MemCheck、Dr. Memory等进行测试最高达200倍的时间开销，本文工具的平均时间开销合理适中、稳定性和实用性更好.

5.2.2　内存开销(Memory Cost)分析

该部分程序运行占用内存空间大小的获取通过Linux提供的time工具完成，使用time -f "%M %t %K"命令就可以获取程序的执行时所占用的实体存储器的最大值、执行时所占用的实体存储器的平均值和所占用的存储器总量的平均大小. 由于实验执行的程序都是单进程多线程程序，因此后两个值均为0. 本文实验将程序执行时所占用的实体存储器的最大值作为内存开销的衡量指标. 但由于天河新一代超算系统还不支持time指令的使用，因此实验数据来自于与天河新一代超算系统的CPU采用相同的ARMv8架构的飞腾S2500处理器上的运行数据.

实验结果如图15所示，结果表明缓存冲突检测和伪共享检测子工具的内存开销均相差无几. 这是因为程序运行中实际访问到的地址空间仅占全部地址空间的一小部分. 因此，虽然SMU中存储的信息量并不相同，但是其对内存开销的影响并不大. 缓存冲突检测和伪共享检测的内存开销整体趋势较为平稳，除了parsec-3.0的bodytrack程序，其余程序半数内存开销均在150倍以下. 内存缺陷检测的平均开销在80倍. 内存缺陷检测工具的内存开销波动较大，原因是若程序存在的问题较多，则存储错误记录所需的内存空间也随之增加. 图15中有一个峰值——bodytrack，该程序的三个子工具的内存开销都明显高于其他程序，原因是程序的功能为计算机视觉模拟，程序所需要的输入数据的量相比其他程序要大得多，因此内存开销也更大.

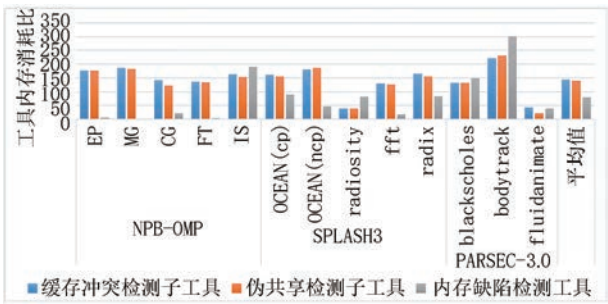

图15　工具的内存开销

5.2.3　优化策略有效性分析

该部分实验采用消融实验的思路，即分别将不使用min-write策略、缓存行对齐填充策略和线程访问隔离策略的工具运行时间与完全优化的工具运行时间作比较，得出某一优化策略的加速效果. 图16～18中的纵坐标"消融加速比"表示，仅不使用某一优化策略的工具运行程序的时间除以完全优化的工具运行程序的时间的比值.

(1) min-write策略. 其优化效果如图16所示，缓存低效行为检测平均加速20倍，内存缺陷检测平均加速10倍. 该策略避免了大量由于工具更新影子内存造成的缓存失效的开销，因此大大提高了工具的性能. 内存缺陷检测不需要在影子内存中额外存储线程标识位图和其他位图，所以其对影子内存的操作频率较低，因此该优化策略带来的加速比也较低.

(2) 缓存行对齐填充策略. 其优化效果如图17所示，工具的三个功能模块平均加速1.2倍. 该策略对SMU进行填充使其大小是缓存行大小的整数倍，这样可以避免工具更新影子内存带来的伪共享问题. 但min-write策略已经大大减少了工具本身带来的缓存失效，其中能够发生伪共享的情况也就

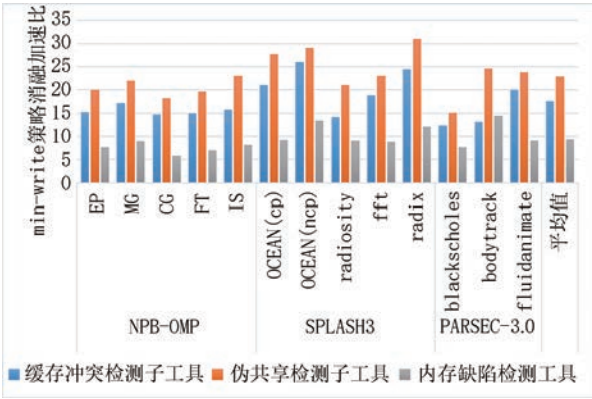

图16 不使用min-write策略较完全优化的时间开销

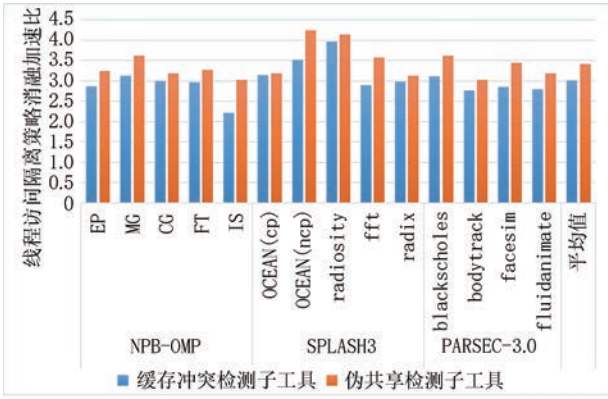

图18 不使用线程访问隔离策略较完全优化的时间开销

更少. 因此使用缓存行对齐填充策略来消除极少量的伪共享所带来的加速效果并不明显.

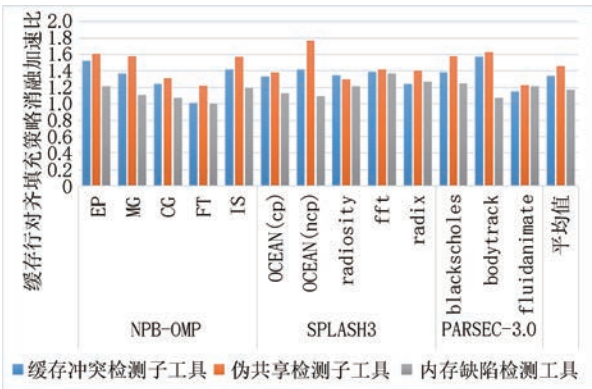

图17 不使用缓存行对齐填充策略较完全优化的时间开销

(3)线程访问隔离策略. 其优化效果如图18所示,缓存冲突和伪共享检测子工具平均加速3倍. 因为内存缺陷检测工具并不需要统计全局数据,所以该策略并不适用于内存缺陷检测工具. 对于多个线程对同一个全局变量进行更新的行为,该策略将其转变为各个线程使用线程本地计数器计数,在线程运行结束后将本地计数器的值累加到全局计数器上的方式. 这样可以避免大量的真共享导致的缓存失效,提高工具效率.

5.2.4 可扩展性分析

在天河超算系列中,天河新一代超算系统通用处理器首次采用ARM架构. 上一代天河二号通用处理器采用Ivy Bridge-E Xeon E5 2692处理器,天河一号通用处理器采用Intel Quad Core Xeon E5540处理器,二者处理器均采用X86架构. 工具在X86架构上均可以正确高效地执行.

5.3 工具正确性验证

该节使用含特定问题的测试程序对工具三个功能模块进行测试,目的是证明工具能够正确发现程序中存在的问题,证明工具的正确性和有效性.

5.3.1 缓存低效行为检测

测试程序使用pthreads线程库,由主线程开启4个工作线程,每个工作线程对一个全局静态变量循环累加,四个全局静态变量存储于同一个缓存行中. 以文本文件形式输出的缓存冲突检测子工具的检测结果如图19所示. 结果表明缓存冲突所占的比例极高,达到了20%左右,这符合程序执行的预期.

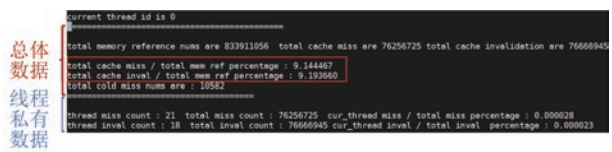

图19 缓存冲突检测结果

以GUI形式输出的伪共享检测子工具的检测结果如图20所示. 使用该GUI的方式是运行GUI版本的工具,工具会将各项数据存储并打包成

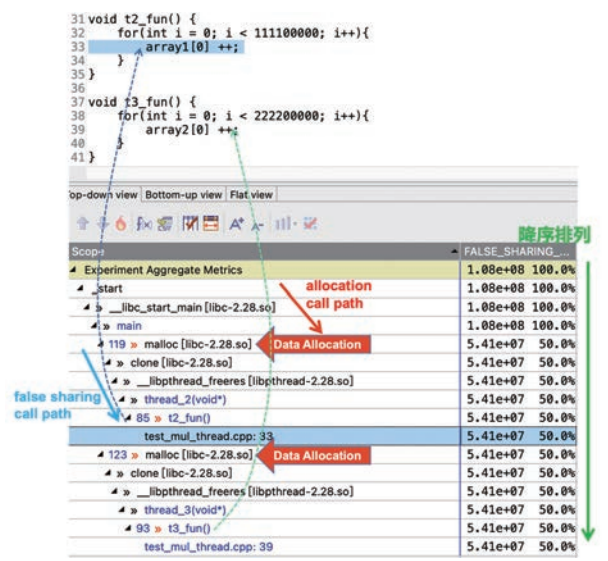

图20 伪共享检测结果

hpcrun 格式数据；用户再使用 hpcprof 对运行时数据进行解析，得到 database 数据；最后使用 HPCViewer 打开 database 文件即可．HPCViewer 的界面分为上下两部分，上半部分展示程序源代码，下半部分展示工具记录统计的性能数据．下半部分的第一栏展示程序的调用路径，后几栏展示各调用位置的累计数据，点击调用路径上的行数可以定位到源代码文件的相应行．图 20 下半部分的第二栏展示的结果表明四个全局静态变量的伪共享次数很多．根据工具提供的伪共享发生位置的调用树信息，结合变量名称或者创建位置的源代码行数，可以很容易地定位问题变量的位置并进行相应的优化修改．

针对程序存在的伪共享问题，使用缓存行对齐填充策略将四个全局静态变量分置于四个不同的缓存行，可以获得 4 倍的加速比．证明工具检测出来的问题是真实存在的，解决该问题可以获得相当的性能提升．

5.3.2　内存缺陷检测

测试程序如图 21 所示，(1)～(5)分别代表访问非法地址空间、访问已释放的地址空间、访问未初始化的空间、两次释放问题和内存泄露这五类问题的具体示例代码片段．

图 21　内存缺陷测试程序

工具能够正确检测五类内存缺陷问题，但由于篇幅所限，仅展示访问非法地址空间和两次释放问题的运行结果．工具的结果报告如图 22 和图 23 所示．图 22 和 23 分别对应图 21 的(1)和(4)，工具的结果报告中，绿色框标出的错误类型信息指明了问题的种类，并提供了详细的调用树信息．通过点击调用树信息，用户可以方便地定位问题所在的源代码位置以进行后续的修改调整，足以证明工具的正确性和有效性．

5.4　案例分析

该部分介绍用户使用工具进行程序的缓存低效

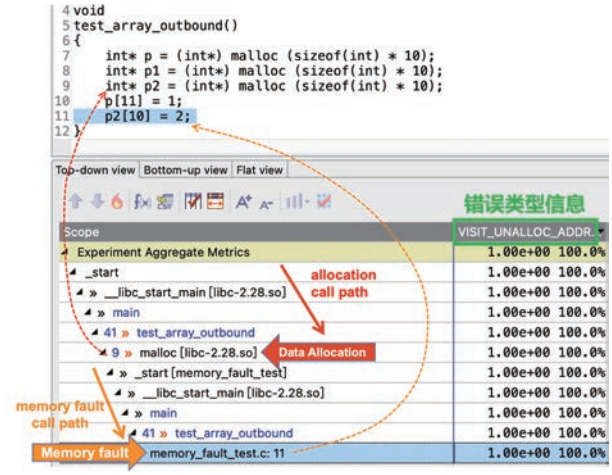

图 22　访问非法地址空间检测结果

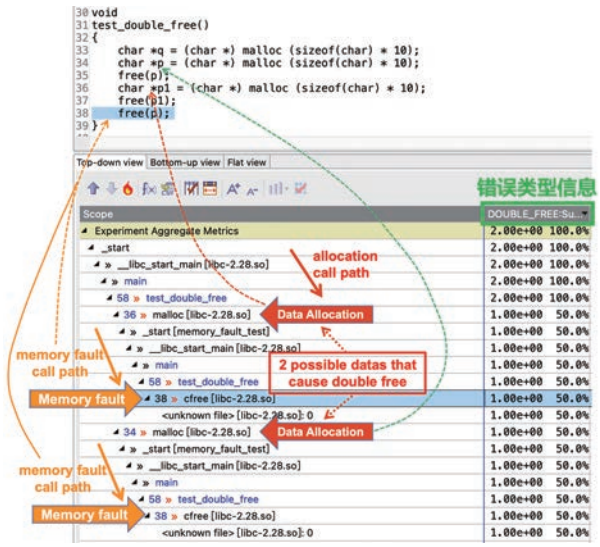

图 23　两次释放问题检测结果

行为检测的一套流程：即从用户的视角，介绍使用工具检测一系列程序并针对存在问题的程序使用工具提供的信息一步步优化，最终取得一定的性能提升的全过程．

首先，对于大量程序，用户应首先使用开销最小的缓存冲突检测子工具来检测某一程序是否存在大量的缓存冲突情况，以至于有必要进行进一步的伪共享检测．使用缓存冲突检测子工具对测试集的全部程序进行检测的结果如表 3 所示，其中 ocean (ncp)的 16 线程加速比是最低的，且其缓存冲突占全部访存指令的比例是最高的．因此可以判定该程序必定存在性能问题，存在优化的空间．

接下来使用伪共享检测子工具对该程序进行检测，结果如图 24 所示．可以发现程序中存在一个发生较多次数伪共享的变量，它的创建位置位于源代码中的第 229 行．

表3 测试集程序16线程加速比和缓存冲突检测结果表

数据集名称	具体测试用例名称	16线程加速比	缓存冲突占全部访存指令比例
NPB-OMP	EP	15.51	1.90e-7
	MG	6.36	5.76e-3
	CG	5.6	3.18e-3
	FT	6.36	9.43e-3
	IS	12.81	9.91e-3
SPLASH3	OCEAN(cp)	3.78	1.06e-2
	OCEAN(ncp)	3.53	1.27e-2
	radiosity	8.28	2.70e-3
	fft	7.87	7.39e-3
	radix	12.02	1.85e-2
PARSEC-3.0	blackscholes	11.01	6.29e-5
	bodytrack	8.86	1.51e-3
	fluidanimate	7.22	1.46e-3

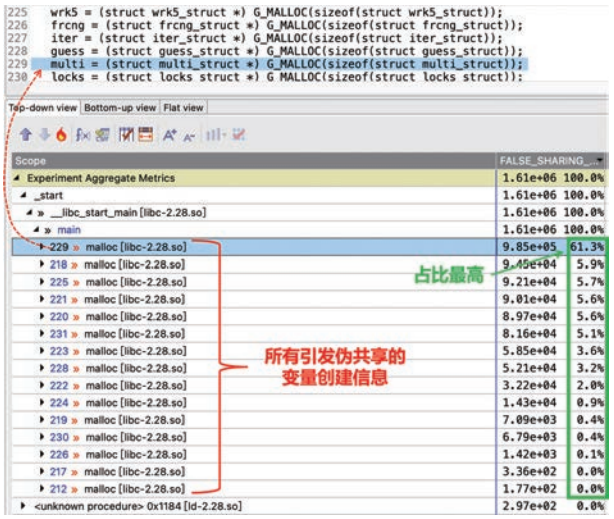

图24 伪共享检测工具分析OCEAN(ncp)结果1

接下来点击该变量,即可得到如图25所示的详细的由该变量引发的伪共享的上下文位置列表,每一位置都可以对应到源代码的相应行数.对该变量使用填充策略,可以在16线程情况下获得3倍的加速比.这一结果证明使用工具对现实应用进行检测并对检测出的问题进行优化可以获得相当的性能提升,证明工具是有效且重要的.

6 总结与未来工作

本文针对天河新一代超算系统的通用处理器设计开发了一套性能分析工具集,包含缓存冲突检测、伪共享检测和内存缺陷检测三个子工具.工具可以在天河新一代超算系统的普通用户权限下,分析系统单节点内以及数据并行性较高的多节点程序的性

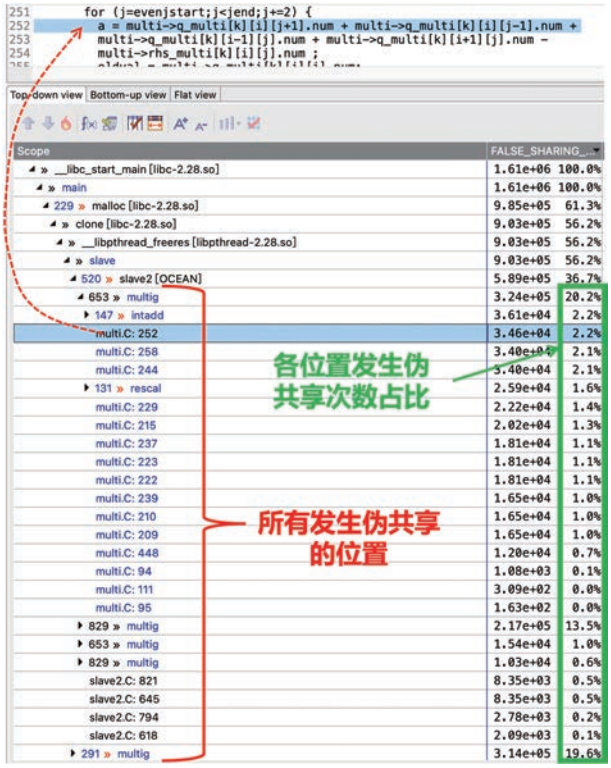

图25 伪共享检测工具分析OCEAN(ncp)结果2

能问题,并可以解决程序的内存问题.工具的目标是提高编程人员在天河新一代超算系统上编写并行程序的体验,并使其更容易编写出高性能的并行程序.工具使用min-write、缓存行对齐填充和线程访问隔离策略来优化工具性能,降低工具时间开销;还创新性地提出了红区检测法和红区恢复与隐藏机制,极大地降低了内存缺陷检测的假错误率.

经过实验验证,本文所开发的工具的时间和内存开销均较为良好,大大增加其实用性.经过针对性的测试证明工具的正确性和有效性是可靠的,并使用现实应用的实际案例介绍了使用工具检测伪共享的流程,结果表明解决伪共享问题可以获得巨大的性能提升,工具发挥了重要的作用.

然而,工具目前还存在一些问题:内存缺陷检测工具仍存在一定的假错误;缓存低效行为检测工具的两个子工具的内存开销过大导致部分程序无法运行;对人工智能领域的相关程序支持不足等.对于存在的问题,本文会在未来针对这些问题改进工具,力求为编程人员更精准、更高效、更友好地报告程序问题.

未来,我们将基于目前工作进一步探索对进程间交互进行性能分析的方法,将其融入本文工具中,实现对MPI的支持.针对天河新一代超算系统独特的加速器架构,今后也需要继续研究获取加速器内部数据、分析加速器内程序运行行为的方法.实现

以上两个目标后，本文所实现的工具就可以支持天河新一代超算系统全机多节点程序的性能分析，帮助天河新一代超算系统充分利用加速器的效果，发挥更大的效用.

参 考 文 献

[1] Zhao Q, Liu X, Chabbi M. DrCCTProf: A fine-grained call path profiler for arm-based clusters//Proceedings of the SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. Phoenix, USA, 2020: 1-16

[2] Bruening D, EfficientAmarasinghe S., transparent, and comprehensive runtime code manipulation [Ph. D. Thesis]. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Massachusetts, USA,2004

[3] Liu T, Berger E D. Sheriff: precise detection and automatic mitigation of false sharing//Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications. New York, USA, 2011: 3-18

[4] Zhao Q, Koh D, Raza S, et al. Dynamic cache contention detection in multi-threaded applications//Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Newport Beach, USA, 2011: 27-38

[5] Liu T, Tian C, Hu Z, et al. Predator: Predictive false sharing detection//Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York, USA, 2014: 3-14

[6] Luo L, Sriraman A, Fugate B, et al. Laser: Light, accurate sharing detection and repair//Proceedings of the 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA). Barcelona, Spain, 2016: 261-273

[7] Liu T, Liu X. Cheetah: detecting false sharing efficiently and effectively//Proceedings of the 2016 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). Barcelona, Spain, 2016: 1-11

[8] Helm C, Taura K. Perfmemplus: A tool for automatic discovery of memory performance problems//Proceedings of the International Conference on High Performance Computing. Frankfurt/Main, Germany, 2019: 209-226

[9] Schindewolf M. Analysis of cache misses using SIMICS [Master's Thesis]. Institute for Computing Systems Architecture, University of Edinburgh, Edinburgh, UK, 2007

[10] Günther S M, Weidendorfer J. Assessing cache false sharing effects by dynamic binary instrumentation//Proceedings of the Workshop on Binary Instrumentation and Applications. New York, USA, 2009: 26-33

[11] Seward J, Nethercote N. Using valgrind to detect undefined value errors with bit-precision//Proceedings of the USENIX Annual Technical Conference, Marriott Anaheim, Canada, 2005: 17-30

[12] Bruening D, Zhao Q. Practical memory checking with Dr. Memory// Proceedings of the International Symposium on Code Generation and Optimization (CGO). Chamonix, France, 2011: 213-223

[13] Eigler F C. Mudflap: Pointer use checking for c/c++// Proceedings of the First Annual GCC Developers' Summit. Ottawa, Canada, 2003: 57-70

[14] Nagarakatte S, Zhao J, Martin M M K, et al. CETS: compiler enforced temporal safety for C//Proceedings of the 2010 International Symposium on Memory Management. Toronto, Canada,2010: 31-40

[15] Serebryany K, Bruening D, Potapenko A, et al. {AddressSanitizer} : A fast address sanity checker// Proceedings of the 2012 USENIX Annual Technical Conference (USENIX ATC). Boston, USA, 2012: 309-318

[16] Lu K, Wang Y, Guo Y, et al. MT-3000: a heterogeneous multi-zone processor for HPC. CCF Transactions on High Performance Computing, 2022,(1): 1-15

[17] SakalisC., LeonardssonC., KaxirasS., and RosA.. Splash-3: A properly synchronized benchmark suite for contemporary research// Proceedings of the Performance Analysis of Systems and Software (ISPASS). Uppsala, Sweden, 2016: 101-111

[18] Bienia C, Kumar S, Singh J P, et al. The PARSEC benchmark suite: Characterization and architectural implications//Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques. New York, USA, 2008: 72-81

FENG Wen-Tao, Ph. D. candidate. His main research interests include high performance computing, performance profiling and computer architecture.

LUAN Zhong-Zhi, Ph.D., associate professor. His main research interests include resource management in distributed environment, supporting methods and technologies for application development on heterogeneous computing systems, performance analysis and optimization of high performance computing applications, etc.

YANG Hai-Long, Ph. D., associate professor. His research interests include parallel and distributed computing, HPC, performance optimization and energy efficiency.

QIAN De-Pei, professor, Member of Chinese Academy of Sciences. His research interests include distributed computing, high performance computing and computer architecture.

Background

Program performance analysis is an important means to improve program performance. As programs get larger and their execution mode changes from serial to parallel, performance bottlenecks usually become more pronounced. This problem is particularly important for large-scale computing systems such as supercomputers, where low-performance programs not only take up a large number of hours, but also lead to unnecessary waste of energy. Therefore, it is of great significance to develop performance analysis tools that can be used in supercomputer systems.

At present, the international performance analysis is mainly divided into two categories: static analysis and dynamic analysis. Dynamic analysis is divided into coarse-grained analysis and fine-grained analysis. Since static analysis cannot obtain runtime data, the analysis accuracy of existing studies of this kind cannot reach a high level. Coarse-grained analysis in dynamic analysis At present, the latest research in the world has reached a relatively perfect level. A series of analysis tools represented by HPCToolkit can well analyze the hot spot of program execution, communication bottleneck and other problems. However, due to the limitations of the operating environment (instruction set, hardware support, etc.) and pile driving tools, there is not a completely universal fine-grained analysis tool in the world.

In this paper, a performance analysis tool set is designed and developed for the general processor of TianHe new generation supercomputer system, which can analyze and detect cache inefficiency and memory defects efficiently and accurately. This set of tools ensures the stability and robustness of the program running on TianHe system, and helps to improve the performance of the program. The appearance of this set of tools fills the blank of TianHe new generation supercomputer system performance analysis tool chain and helps TianHe new generation supercomputer system to give full play to its performance.