计算机科学
COMPUTER SCIENCE

并行计时偏差评测指标及工具

廖秋承, 周洋, 林新华

引用本文

廖秋承, 周洋, 林新华. 并行计时偏差评测指标及工具[J]. 计算机科学, 2025, 52(5): 41-49.

LIAO Qiucheng, ZHOU Yang, LIN Xinhua. Metrics and Tools for Evaluating the Deviation in Parallel Timing [J]. Computer Science, 2025, 52(5): 41-49.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.241200053

并行计时偏差评测指标及工具

廖秋承[1]　周　洋[2]　林新华[1]
1 上海交通大学高性能计算中心　上海 200240
2 浙江省科技厅　杭州 310006
(keyliao@sjtu.edu.cn)

摘　要　在并行计算程序中插桩计时,是多核处理器中常用的性能测量和分析手段。然而,高精度并行计时的准确性受到计时方法、硬件配置和运行时环境等影响,测量结果不稳定,性能分析结论难以复现。近年来,高性能多核处理器的核心数量不断攀升,给多核心并行计时的准确性带来了更大挑战。目前,在真实计算程序中,高精度并行计时技术面临两大问题:1)无法定量比较不同计时函数的准确性;2)无法定量分析多种因素影响下微秒、毫秒级并行计时分布的偏差幅度。针对上述问题,首先设计了用于定量评测计时结果统计学分布偏差的指标,并开发了支持 X86 和 Armv8 指令集的多核心计时结果偏差评测工具 ParTES。ParTES 可以模拟真实计算场景的缓存特征和计时间隔,定量评测不同计时函数的测量偏差。其次,在鲲鹏、飞腾和海光高性能处理器上开展了微秒和毫秒级并行计时稳定性量化分析。实验结果表明,计时方法、缓存命中率、计时函数邻近指令和服务器硬件配置等因素,均会对并行计时结果的准确性产生影响。在鲲鹏、飞腾和海光处理器上,计时结果偏差最小且偏差幅度变化最稳定的计时方法分别是 PAPI 的计时函数、POSIX 的 clock_gettime 计时函数和 C86 指令集汇编计时指令 RDTSC。

关键词:高性能计算;并行计算;性能评测;性能分析;误差分析
中图分类号　TP302

Metrics and Tools for Evaluating the Deviation in Parallel Timing

LIAO Qiucheng[1], ZHOU Yang[2] and LIN Xinhua[1]
1 Center for High-Performance Computing, Shanghai Jiao Tong University, Shanghai 200240, China
2 Science and Technology Department of Zhejiang Province, Hangzhou 310006, China

Abstract　In parallel computing, instrumenting specific code segments is commonly used for performance evaluation on multicore processors. However, factors such as timing methods, hardware configurations, and runtime environments affect parallel timing accuracy, jeopardizing stability and reproducibility of performance measurements. As the core number of multicore processors grows, accurate parallel timing has grown more challenging. Two key problems remain: 1) current method cannot quantitatively compare the accuracy of different timing methods; 2) the root cause of parallel timing variability is not fully understood. This paper proposes metrics for evaluating the deviation in measurements and presents ParTES, a tool which emulates realistic cache conditions and timing intervals on X86 and Armv8 CPUs, allowing quantitative evaluation of timing variability across different timing methods. This study performed microsecond-level and millisecond-level analyses of parallel timing deviations on Kunpeng, Phytium, and Hygon processors. The results show that the performance of timing methods, cache status, nearby instructions, and server hardware configurations all influence accuracy is excellent. Among these CPUs, the most stable timing methods are PAPI on Kunpeng, POSIX's clock_gettime on Phytium, and the RDTSC instruction on Hygon.

Keywords　High performance computing, Parallel computing, Performance evaluation, Performance analysis, Error analysis

1　引言

并行计时,是现代高性能多核处理器性能测量的重要手段。随着处理器核心数量的迅速增加,现代高性能多核处理器中出现了多节点、多核心性能不一致[1-2]、多核资源竞争[3]以及长尾延迟[4-6]等性能问题。为了测量和分析这些问题,向被测程序的源代码插入计时函数(即"插桩")并进行多核心并行计时,已成为多核处理器常用的性能测量手段。通过观察和分析多核心计时结果的统计分布,可进一步开展性能分析。然而,计时函数也是被测程序的一部分,并行计时准确性容易受核心数量和缓存状态等因素的干扰[7]。

处理器核心数量的增加,给高精度并行计时结果分布的

到稿日期:2024-12-09　返修日期:2025-02-18
基金项目:国家自然科学基金(62072300)
This work was supported by the National Natural Science Foundation of China(62072300).
通信作者:林新华(james@sjtu.edu.cn)

准确性带来了挑战。以部分代表性型号为例，海光、鲲鹏等国产处理器已超过 64 核心，AMDEPYC 系列可达 128 核心。在此类处理器中，多核资源竞争、缓存误淘汰和通信长尾延迟等性能问题，直接影响了并行计算性能的扩展性。然而，上述现象的时间尺度均在纳秒、微秒级别，测量结果容易受到计时波动的影响。已有研究发现，当两次计时函数的调用间隔低于 1ms 时，由计时函数开销、缓存不命中等因素共同引起的计时波动，会导致计时结果分布在中位数、均值、四分位点等关键统计学指标上出现偏差，且分布的整体形态与真值之间出现差异[7-9]。这些现象会影响计时结果的准确性和稳定性，进而导致性能分析的结论出现偏差。

亚毫秒级并行计时的测量偏差及其影响因素尚未得到充分研究。该研究主要面临两个技术问题。

问题一：现有技术无法定量比较不同计时函数的准确性。目前，为避免计时函数自身的开销对计时结果的准确性产生影响，主要采用两种方法：1)拉长两次计时函数调用之间的时间间隔，确保计时函数带来的额外开销占被测程序耗时的比例低于 5%[8]；2)人为设定 1ms 或 10ms 等时间作为计时间隔的下限[10]。然而，根据中心极限定理，拉长计时间隔会使原有时间分布向正态分布收敛，从而掩盖真实的性能问题。此外，已有研究证明，计时开销本身具有不确定性，且与计时结果分布偏差程度之间不存在稳定的量化关系[7]。

问题二：现有技术无法定量分析多种因素影响下微秒、毫秒级并行计时分布的偏差幅度。包括本团队在内的多项研究已证明，计时波动具有不稳定性，并受到芯片参数、缓存状态和程序运行时参数的影响[9,11-13]。然而，这些研究未能定量分析常见影响因子对计时波动的具体影响程度，导致当前性能分析工作中缺乏对计时波动和计时分布偏差的定量参考，难以快速开展测量和分析工作。

为解决上述问题，我们设计并开发了跨架构的并行计时偏差检测套件 ParTES(Parallel Timing Error Suites)，并利用该套件开展了多个计时波动影响因子的定量分析。具体工作如下：1)基于 Wasserstein 指标[14-15]，设计了计时结果分布偏差的量化评价算法；2)开发了支持多种芯片和计时方法的并行计时偏差检测套件 ParTES，其可模拟真实应用程序运行时的缓存与指令状态，检测计时波动并测量并行计时分布的偏差；3)利用 ParTES，在海光、鲲鹏和飞腾处理器上对硬件参数、计时方法和缓存状态等计时偏差影响因子进行了定量分析。

本研究涵盖了 3 种主流国产高性能处理器，并支持 C86(X86)和 Armv8 指令集。实验表明，在并行计算程序中，计时结果的影响因子包括但不限于：计时方法、计时点附近的缓存利用率、计时点附近的指令特征、计时点间隔和硬件配置等。ParTES 可以通过模拟测量环境，帮助研究者和工程师在开展毫秒、微秒级性能测量时，确定特定缓存条件下所使用的计时方法是否能够达到所需准确度。在鲲鹏、飞腾和海光处理器上，计时结果偏差最小且偏差幅度变化最稳定的计时方法分别为：PAPI 的计时函数、POSIX 的 clock_gettime 计时函数以及 C86 指令集的汇编计时指令 RDTSC。

本文的主要贡献如下：

1)设计了基于 Wasserstein 指标的计时波动量化评价算法；

2)开发了面向多款国产芯片，支持 X86，C86 和 Armv8 指令集的并行计时偏差检测套件 ParTES；

3)开展了多个常见影响因子的并行计时偏差定量分析，给出了 3 种国产芯片在典型测量场景下获得准确计时分布所需的计时间隔。

2 研究背景

2.1 并行计时中的计时波动

计时波动，指计算机系统中通过计时函数测量的时间与程序真实运行时间之差存在波动。其根本原因是现代处理器的复杂结构导致的指令顺序和指令耗时的不确定性。封装于晶振之上的时钟计数器通过计数定频晶振的震荡数，将处理器内部的“拍”转换为真实世界的墙钟时间戳。在时间测量中，各类计时方法会以不同的方式封装这些时钟源，并向上层函数提供调用接口。

一方面，不同处理器型号的时钟计数器在计时精度和读取方式上存在差异；另一方面，不同计时函数的功能、实现方式、封装方式和调用路径也各不相同。以 Armv8 处理器为例，用户既可以使用汇编指令读取 CNTVCT_EL0 寄存器，也可以使用 PAPI[16] 工具的 PAPI_get_real_nsec() 函数读取时间戳。前者直接将寄存器数据拷贝到内存中，而后者则需要经过多层函数抽象和 Linux 操作系统调用。因此，两者的开销差异较大，在真实计算程序中也具有完全不同的计时波动分布。总的来说，不同计时方法的差异与真实计算程序中处理器缓存、TLB、流水线等子系统的不确定性叠加，共同导致了处理器时间测量中的计时波动现象。

计时波动是影响性能测量准确性的重要因素。现有研究表明，计时波动分布会与被测程序耗时分布互相叠加，从而导致性能分析得出不准确的结论。此外，当测量者需要采集各类性能事件并与计算时间进行联合相关性分析时，计时波动导致的测量结果分布偏差会使性能计数器的采集数据(尤其是多次不同测试时的测量数据)无法与真实运行时间匹配，从而导致相关性分析失败。因此，在需要深度性能分析的性能工程场景(如处理器微架构研究、复杂系统性能分析和复杂应用建模与深度优化等场景)中，量化具体场景下计时波动对计时结果分布的影响，并确定适当的计时粒度和计时区间，是准确进行性能分析的前提。

2.2 Wasserstein 距离

本研究采用 Wasserstein 距离指标[14-15](Wasserstein Distance，以下简称“W 指标”)来量化两个时间分布之间的差异。W 指标用于度量可测空间中不同概率分布之间的距离，常被用于机器学习、图像处理等领域。在一维样本空间中，W 指标计算的是样本的累积分布函数曲线之间的面积。此外，在一些并行计算性能研究工作中，为便于观察特定运行时间出现的概率，通常使用分位函数(Quantile Function)(也称逆累积分布函数(Inverse Cumulative Distribution Function,

ICDF))，进行分析，即将累积分布函数的 X 轴和 Y 轴对调[1,8]。W 指标的计算方法是对所有分位点处两个分布之间的时间差异求和，并使用与原始数据相同的时间单位，从而定量评估两个时间分布之间的偏差。

一次实验中，计时结果 t 的集合 U 的概率密度函数可以表示为：

$$F_U(t):=Pr(U\leqslant t)=p \tag{1}$$

则分位函数可以表示为：

$$Q_U(p)=F_U^{-1}(p)=t \tag{2}$$

即输入分位点 p，求此时对应的时间 t。因此，W 指标的计算过程为：

$$W(U,V)=\frac{1}{n}\sum_{i=1}^{n}|Q_U(i)-Q_V(i)| \tag{3}$$

当测试次数足够多($n\to\infty$)时，分位函数可被近似为连续的。因此，W-metric 可以视为两个分位函数曲线围成的重叠面积。

3 测量分布偏差定量评测算法的设计与实现

3.1 计时基准分布生成算法

本研究通过控制变量法和参考计算过程来定量评估计时偏差。并行计算程序的运行时间波动普遍存在，且不同软硬件环境下的波动特征差异显著。现有方法不仅难以量化计时偏差，也难以评估处理器参数、缓存状态、软件环境等因素对计时偏差的影响。

本算法通过生成特定参数下的基准时间分布来实现时间标定。算法基于寄存器减法指令，具有耗时稳定、不易受性能波动影响的特点，可作为处理器运行时间的基本度量工具。其核心是“双减法内核”。首先将一个寄存器值初始化为整型 X，随后使用两条存在数据依赖的标量寄存器减法指令将其值递减至 0，即“$r1=r0-1$；$r0=r1-1$”。若标量减法的指令延迟为 1 拍，则该内核的运行时间为 X 拍；将处理器主频固定为 F 赫兹后，耗时为 $\frac{X}{F}$ s。对比计时结果和双减法内核理论耗时，即可定量分析测量偏差；并且，还可在计时区域外添加其他计算内核，研究计时函数附近的计算内核对计时结果偏差的影响。

需要指出，双减法内核并非完全没有运行时间波动。现代处理器的复杂性导致在循环初始的若干条指令中，运行时间仍可能受取指、流水线延迟等因素的影响，执行过程中也可能受到操作系统噪音的干扰。然而，该内核仅涉及两次标量加法器操作和一条分支操作。其中，分支操作延迟可被加法器操作掩盖，且标量计算存在无法流水化的数据依赖。因此，双减法内核的运行时间稳定性高于更复杂的计算指令，可视为无波动的时间参考。

算法的整体流程如算法 1 所示。

算法 1 计时基准分布生成算法

输入：计时次数 N，分布参数 $P_D=[d_0,d_1,\cdots]$ 前冲刷内核参数 $P_F=[f_0,f_1,\cdots]$；后冲刷内核参数 $P_R=[r_0,r_1,\cdots]$

输出：理论耗时 $t\_arr=[t_0,t_1,\cdots]$；测量结果 $m\_arr=[m_0,m_1,\cdots]$

```
1. IF myrank==0 DO
2.     FOR i←0 to N−1 DO
3.         t_arr[i]←sample_rand(P_D) // 随机采样
4. MPI_Bcast(t_arr) // 广播随机序列
5. FOR i←0 to N−1 DO // 主循环
6.    MPI_Barrier(); // 进程同步
7.    flush_front(P_F) // 计时前缓存冲刷
8.    R0← t_arr[i] // 载入寄存器 R0
9.    ts← read_timestamp() // 起始时间戳
10.   WHILE R0 ≠ 0 DO //双减法内核
11.     R1← R0−1 // 标量减法
12.     R0← R1−1 //R0、R1 互相依赖
13.   te← read_timestamp() // 结束时间戳
14.   flush_rear(P_R) // 计时后缓存冲刷
15.   m_arr[i]=te − ts //计算耗时
16. print(t_arr);print(m_arr) // 结果输出
```

步骤 1　第 1－4 行，生成随机序列。基于典型概率分布，如帕累托分布、正态分布、泊松分布等，根据输入的分布参数随机采样，生成理论耗时数组 t_arr，并广播至所有进程。

步骤 2　第 5－15 行，主循环。遍历 t_arr，进行 N 次计时。

步骤 2.1　第 6 行，进程间同步。使用 MPI 同步，使不同进程同时开始测试。

步骤 2.2　第 7 行，前冲刷内核。调用自定义微型计算内核，模仿真实应用在计时前的计算。

步骤 2.3　第 8－13 行，对双减法内核进行计时。首先将 $t\_arr[i]$ 载入寄存器 $R0$。据前文所述，双减法内核的理论耗时取决于 $t\_arr[i]$。用自定义计时方法 $read\_timestamp()$ 在双减法内核前后分别取时间戳。

步骤 2.4　第 14 行，后冲刷内核。与步骤 2.2 类似，模仿真实应用在计时之后的计算。

步骤 2.5　第 15 行，计算耗时。

步骤 3　第 16 行，输出结果。

3.2 关键统计指标算法

本节基于 W 指标和常用的计时开销指标，推导出两个关键统计指标，用于量化两次测量结果分布在中低分位与高分位之间的差异，其分别表示为中低分位相对偏差 R_L 和高分位相对偏差 R_H。上述指标可帮助测量者定量评估测量结果偏差以及所选计时间隔的合理性。

对于两个由大量计时结果样本构成的分布 U 和 V，用 W 指标计算它们的总体差异，记作 W_A。令分位函数为 $F(i)$ $(0<i\leqslant N)$，则：

$$W_A(U,V)=\frac{1}{N-\delta}\sum_{i=1}^{N-\delta}|Q_V(i)-Q_U(i)| \tag{4}$$

其中，N 为测量者定义的分位点个数，常用值为 100(百分位)或 1 000(千分位)；δ 为舍弃的极大值所占的比例，用于排除部分不属于测量目标的极端异常值的干扰(如处理器被抢占、系统故障、中断事件等)。W_A 与时间量纲相同。

为了进一步评价统计分布的不同分位数区间的偏差，对于 x,y 两个分位点($0<x<y\leqslant N-\delta$)，根据式(4)进一步计算

局部 W 指标 $W_{x,y}$：

$$W_{x,y}(U,V)=\frac{1}{y-x+1}\sum_{i=x}^{y}|Q_V(i)-Q_U(i)| \tag{5}$$

目前，为了计算计时开销，通常反复运行计时函数，然后求计时函数耗时的平均值。在真实测量场景中，如果计时间隔较短，对精度要求高，测量者往往会人为减去计时开销。以 LIKWID[17] 为代表的一些工具中也支持此功能。记分布 V 相对于分布 U 的最小计时开销为 τ_{VU}：

$$\tau_{VU}=\min(V)-\min(U)$$

在此基础上，为了对齐两组数据的分位函数的起点，同时不改变分布特征，用 τ_{VU} 对 V 进行预处理，从而得到：

$$\hat{W}_{x,y}(U,V)=\frac{1}{y-x+1}\sum_{i=x}^{y}|Q_V(i)-Q_U(i)-\tau_{VU}| \tag{6}$$

特别地，当只有一个分布 U 时，在分位函数图上，也可以计算它与直线 $y=\min(U)$ 之间所围成的面积：

$$\hat{W}_{x,y}(U)=\frac{1}{y-x+1}\sum_{i=x}^{y}|F_U(i)-\min(U)| \tag{7}$$

1）中低分位相对偏差 R_L：计算中低分位数的累积差异。对于 U,V 两组数据，$t_{\min}=\min(P,Q)$。设分位点为 q，计算 R_L，以量化 U 和 V 分布中 $t_{\min}$ 至第 q 分位之间的样本分布偏差量占 $t_{\min}$ 的百分比：

$$R_L(U,V)=\frac{\hat{W}_{1,q}(U,V)}{\hat{W}_{1,q}(U)}\times 100(\%) \tag{8}$$

2）高分位相对偏差 R_H：计算高分位数的累积差异。设分位点为 q，计算 R_H，以量化 U 和 V 分布中第 $q+1$ 分位点至第 $N-\delta$ 分位之间的样本分布偏差量占 $t_{\min}$ 的百分比：

$$R_H(U,V)=\frac{\hat{W}_{q+1,N-\delta}(U,V)}{\hat{W}_{q+1,N-\delta}(U)}\times 100\% \tag{9}$$

其中，R_L 与 R_H 是一对既可独立使用，也可联合使用的指标，以 q 分位点为分界。独立使用时，R_L 与 R_H 可用于量化两个分布的差异在当前时间尺度下的占比。例如，当 R_L 或 R_H 低于 5%时，可认为两个时间分布在对应的分位点区间的偏差低于 5%。联合使用时，可在同一实验中对比 R_L 和 R_H 的值。例如，如果接受 5%的测量偏差，则有 4 种测量偏差情况。

情况 1 R_L 与 R_H 均小于 5%。证明两个分布从最小值到最大值，偏差都可接受。

情况 2 $R_L<0.5\%$，R_H 远大于 5%。中低分位的统计量（如中位数、四分位点）可接受，但计时波动导致测量结果分布出现长尾偏差，处于高分位的较大值的测量结果不准确。

情况 3 R_L 和 R_H 均大于 5%，但两者相差较小。说明在中低分位出现较大偏差，但未出现长尾偏差。此时可能由于程序在运行过程中出现概率较低的事件，导致最小偏差很小，其余分位点偏差较大，但偏差幅度较稳定。

情况 4 R_L 和 R_H 均大于 5%，且两者相差较大。说明两个分布之间的特征差异较大，测量结果偏差无法接受。

3.3 测量分布偏差评测套件 ParTES

ParTES 套件包含对并行计时基准进行测量的程序 dsub.c 和关键统计指标计算脚本 calc_w.py。其整体流程如图 1 所示，共分为配置理论时间分布、配置计时方法和配置冲刷内核 3 个部分。

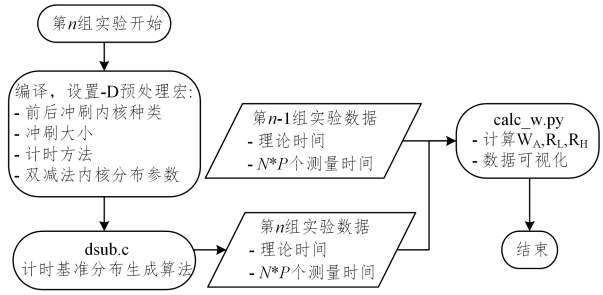


图 1 使用 ParTES 量化测量分布差异的流程

Fig. 1 Diagram of using ParTES to quantify the difference between measurements

1）配置理论时间分布。ParTES 通过 GSL 库（GNU Scientific Library）的 gsl_rng 随机数生成模块生成特定的概率分布，接受分布种类、计时间隔基准时间 t_b 和分布参数 3 个输入。通过在时间系数 α 上生成性能波动来模拟应用程序性能波动。例如，假设生成正态分布，输入 $t_b=1000$，正态分布参数 $\mu=0,\sigma=0.015$，那么，理论运行时间 t 可以表示为：

$$t=t_b\times\sigma,\sigma\sim N(\mu=0,\sigma=0.015) \tag{10}$$

将生成 $\mu=1000$ ns，$\sigma=15$ ns 的正态分布时间。

2）配置计时方法。通过编译时的宏定义（gcc-D），配置计时方法、冲刷内核的数组大小。本研究使用的计时方法和冲刷内核如表 1 所列，支持 4 种计时方法。

表 1 ParTES 支持的计时方法

Table 1 Timing methods supported by ParTES

名称	伪代码
ASM	Arm 处理器： “mrs %0,cntvct_el0” 海光处理器： “RDTSC” “mov %%edx,%0” “mov %%eax,%1” :“=r”(ch),“=r”(cl)
CGT	clock_gettime()
PAPI	PAPI_get_real_nsec()

PAPIX6	// 测量前，添加事件
	PAPI_add_named_event()
	// 测量中，先读时间戳，后读事件
	PAPI_get_real_nsec()
	PAPI_read() // 读 6 个性能事件

（1）ASM：X86 和 Armv8 指令集的汇编计时指令，在 Linux 用户态程序中通过内嵌汇编，读取时钟计数器至通用寄存器，然后转储到内存。Armv8 架构处理器的计时精度取决于处理器型号。例如，鲲鹏处理器的计数器每 10 ns 增加 1，而飞腾处理器上每 1 ns 增加 1。对于 X86/C86 架构，RDTSC 计数器基于 TSC 晶振，TSC 每震荡一次，计数增加 1。

（2）CGT：LinuxGLIBC 封装的 POSIX 标准计时函数 clock_gettime()。该函数通过 linux-vdso 库转换为虚拟系统调用，无需进入 Linux 内核态就可以获取处于内核态的时钟计数器。与 ASM 方法相比，本方法的计时器读取、转储过程对用户不可见，且存在函数跳转开销。

(3)PAPI:PAPI 性能测量工具中的计时函数 PAPI_get_real_nsec(),被广泛用于更高级的性能工具如 ScoreP[18] 和 HPCToolkit[19] 中。本方法对不同处理器的计时接口和 Linux 内核态计时器进行了不同的抽象和封装,抽象程度更高,计时开销不确定性更大。

(4)PAPIX6:在使用 PAPI_get_real_nsec 函数读取时间戳后,额外使用 PAPI_read 函数读取 6 个性能事件。这种方式常见于深度分析和优化工作中。然而,PAPI 在采集性能事件时,其函数调用终点是根据事件种类和硬件种类动态变化的,比 PAPI 开销的波动更大。

3)配置冲刷内核和冲刷量。设置每个核心进行冲刷内核计算的数据量大小,ParTES 根据不同计算内核的数组的个数和维度,将设定的数据大小均匀分配至各个数组。若设定的大小不能被整除,将下取整为 8 字节的整数倍。目前,ParTES 支持的冲刷内核如表 2 所列。

表 2 ParTES 支持的冲刷内核

Table 2 Flush kernels supported by ParTES

名称	伪代码	类型
COPY	$a[i]=b[i]$	连续访存
ADD	$a[i]=b[i]+c[i]$	连续访存
POW	$a[i]=pow(a[i],x)$	浮点计算
DGEMM	$C=1.0*A*B+0*C$	浮点计算
BCAST	*MPI_Bcast*()	集合通信

程序运行完成后,每核心将自己测量得到的计时结果输出至 csv 文件中,供后续分析。

4 国产芯片并行计时偏差检测实验

4.1 实验设计

实验平台的软硬件配置如表 3 所列。所有服务器均关闭超线程,并将核心频率固定在处理器标称默认频率。

表 3 实验平台配置

Table 3 Configurations of experimental platforms

平台	鲲鹏	飞腾	海光
CPU	2 * 海思鲲鹏 920 (64 核,2.4 GHz)	2 * 飞腾 S5000C (64 核,2.1 GHz)	2 * 海光 C86 7490 (64 核,2.7 GHz)
缓存大小	L1:64 KiB,核心独占 L2:512 KiB,核心独占 L3:128 MiB,多核心共享	L1:64KiB,核心独占 L2:512KiB,核心独占 L3:32 MiB,多核心共享	L1:32 KiB,核心独占 L2:512 KiB,核心独占 L3:256 MiB,多核心共享
指令集	Armv8.2	Armv8.2	C86
RAM	16 * 16 GiB DDR4 2933 MT/s	16 * 32 GiB DDR5 4000 MT/s	16 * 32 GiB DDR5 4400 MT/s
操作系统	openEuler 22.03 (Linux 5.10.0)	Rocky Linux 8.10 (Linux 4.18.0)	Rocky Linux 9.4 (Linux 5.14.0)
编译器	GCC 10.3.1	GCC 8.5.0	GCC 11.4.1
MPI 库	OpenMPI 4.1.6	OpenMPI 5.0.5	OpenMPI 5.0.5
其它软件	PAPI 7.1.0 OpenBLAS 0.3.28 GSL 2.7.0	PAPI 7.1.0 OpenBLAS 0.3.28 GSL 2.7.0	PAPI 7.1.0 OpenBLAS 0.3.28 GSL 2.7.0

本文共设计 4 组实验,用于量化评测多种影响因子下计时波动对计时结果分布准确性的影响,并评估 ParTES 的有效性。其中,前 3 组实验使用鲲鹏平台,实验四使用飞腾和海光平台。每个核心重复计时 1 000 次。在所有实验中,设置 1 000 个分位点($N=1000$),舍去 0.5%的极大值($\delta=5$),以第 900 个分位点作为 R_L 和 R_H 的分界点($q=900$)。在所有实验中,使用常见的正态分布,参数为 N($\mu=0,\sigma=0.015$)。

实验 1 不同计时方法的计时偏差的评测。本实验评测不同计时方法在不同的缓存冲刷量和不同的计时间隔基准时间下的计时偏差。实验在 1 个鲲鹏处理器上进行,使用 64 个进程,依序绑定至 64 个物理核心。所有实验的前后冲刷内核均为 COPY。本实验有 2 组测量数据。第一组,将计时间隔基准时间设定为 10 μs,缓存冲刷量设定为 2048KiB,运行 ParTES,通过柱状图和分位函数图观察 ASM,CGT,PAPI,PAPIX6 计时方法的测量值分布与理论运行时间分布的差异。第二组,将计时间隔基准时间设为 10 μs,在[32,64,128,…,4096]KiB 冲刷量下运行 ParTES,计算 R_L 和 R_H,观察计时点附近缓存占用率越来越高时计时偏差的变化趋势。

实验 2 计时间隔对计时偏差影响的评测。评测不同计时间隔基准时间和冲刷内核的组合下,计时偏差的变化趋势。第一步,将冲刷量设为 4096KiB,冲刷内核设为 COPY,在[1,10,…,10 000] μs 的计时间隔基准时间下分别用 4 种计时方法运行 ParTES,计算 R_L 和 R_H。

实验 3 冲刷内核种类对计时偏差影响的评测。冲刷量设为 4096KiB,计时间隔基准时间设为 1 000 μs,将前后冲刷内核依次设置为 COPY,ADD,POW,DGEMM 和 BCAST,计算 R_L 和 R_H,观察不同冲刷内核对计时结果偏差的影响。

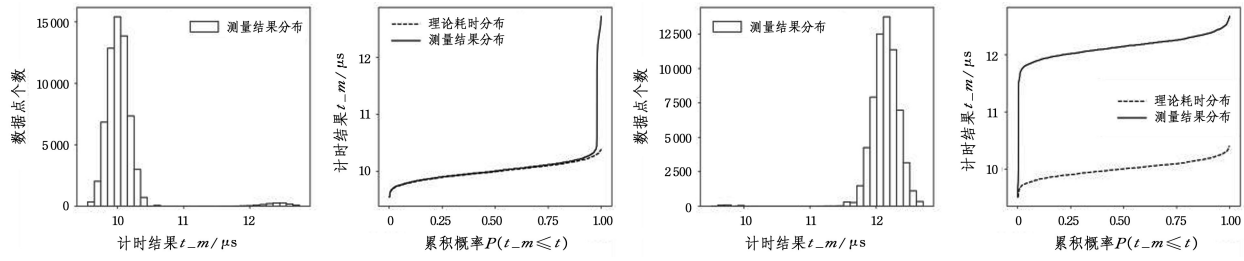
实验 4 不同处理器计时偏差对比的评测。在飞腾、海光平台上重复实验 1 和实验 2。

4.2 实验 1 不同计时方法的计时偏差的评测

2048KiB 冲刷量下,测量结果与理论分布之间的对比如图 2 所示。每种测量方法包含 2 张子图:左图为所有核心的计时结果分布柱状图;右图为理论耗时分布(虚线)和测量结果分布(实线)的分位函数图,横轴为累积概率分位点,纵轴为满足当前累积概率的计时结果。实验表明,鲲鹏平台上 4 种计时方法的计时波动特征有明显差异。

从图 2 中可以直观地对比不同计时方法的测量结果与理论分布的偏差幅度和偏差特征的差异。在该测试环境下,

ASM 和 PAPI 与理论耗时分布匹配得较好(见图 2(a)),但在接近最大测量值时,它们均出现了长尾偏差;CGT 计时开销较不稳定(见图 2(b)),在低分位存在测量结果分布跳变;PAPIX6 的计时结果呈明显的双峰分布(见图 2(d)),无法与理论运行分布匹配。这可能是因为采集了额外 6 个性能计数器后,其自身运行时间在当前时间尺度上体现出较大波动。如果测量者在性能分析中直接使用了 PAPIX6 的结果,将无法得出准确的性能分析结论。



(a)ASM 测量值柱状图(左)和分位函数图(右)

(b)CGT 测量值柱状图(左)和分位函数图(右)

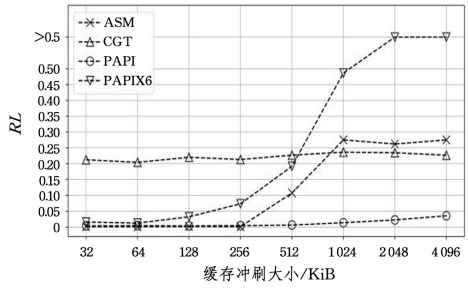
(c)PAPI 测量值柱状图(左)和分位函数图(右)

(d)PAPIX6 测量值柱状图(左)和分位函数图(右)

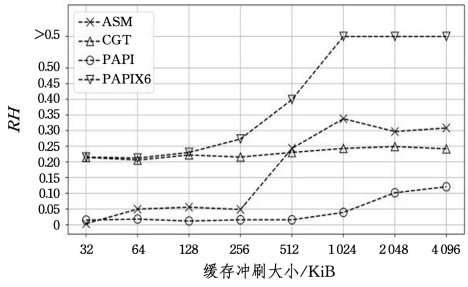
图 2 2048 KiB 冲刷量下,ASM,CGT,PAPI 和 PAPIX6 对 10μs 计时间隔基准时间计时结果的分布

Fig. 2 Plotting of histogram and quantile function of ASM, CGT, PAPI and PAPIX6's timing results' distribution with 2048 KiB cache flush

图 3 展示了缓存冲刷数据量从 32 KiB 逐渐增长至 4096 KiB 时,对 4 种计时方法的测量结果相对偏差指标的影响。随着冲刷数据量的不断增长,不同计时方法体现出不同的计时波动变化规律。



(a)R_L指标的变化规律



(b)R_H指标的变化规律

图 3 不同缓存冲刷量下,4 种计时方法的 R_H 和 R_L 指标的变化趋势

Fig. 3 Trends of R_H and R_L of four timing methods when measuring with different cache flush size

ASM 计时方法在冲刷量不超过 256 KiB 时,R_L 低于 0.2%,R_H 低于 6%;然而,当冲刷量达到 512 KiB 后,R_L 和 R_H 分别突增至 10% 和 20% 以上。CGT 在低分位点发生计时结果的跳变,故在所有缓存冲刷大小下,R_L 和 R_H 均高于 20%。PAPI 在 32~4096 KiB 的冲刷量下,R_L 都低于 5%,中低分位点的偏差幅度是 4 种计时方法中最低的。然而,PAPI 受长尾偏差的影响较大,在 4096 KiB 冲刷量下,R_H 达到 12.1%,此时 R_L 为 3.5%。如果测量者使用 PAPI 进行与性能波动、长尾延迟相关的研究,则会得到高于真实值的性能下降幅度。

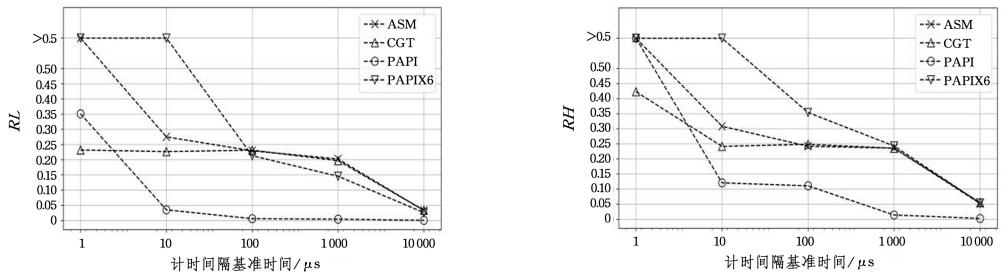
与 PAPI 相比,PAPIX6 在读取时间戳后,还连续读取了 6 个性能事件计数器。这带来了更大的开销和更多的内存操作,从而增加了计时过程中的不确定性,导致严重的长尾偏差。从冲刷量 32 KiB 起,其 R_H 高于 20%。

结果表明,低开销并不意味着低偏差。在 10μs 级计时间隔下,数据位于 L1 和 L2 缓存时,KP920 使用 PAPI 和 ASM 计时方法的计时结果较准确;否则,PAPI 计时方法有更小的偏差和更好的稳定性。

4.3 实验 2 计时间隔对计时偏差影响的评测

两次计时之间的时间间隔逐渐拉长时,R_H 和 R_L 的变化情况如图 4 所示。缓存冲刷量每核心 4096 KiB、基准时间为 1μs 时,所有方法的 R_L>20%,计时结果均不可信。PAPI 从 10μs 基准起,R_L<5%,但此时 R_H 仍高于 10%;基准时间高于 1000μs 时,R_H 低于 5%;基准时间达到 10000μs 时,所有计时方法的偏差均下降到 5% 附近。

该实验表明,拉长计时间隔可以降低计时偏差在计时结果中的占比。ParTES 通过量化指标,结合测量者对偏差的接受程度,合理地判断当前计时间隔是否能帮助计时结果达到足够低的偏差。



(a)R_L指标的变化规律 (b)R_H指标的变化规律

图 4 不同计时间隔基准时间下,4 种计时方法的 R_L 和 R_H 指标的变化趋势

Fig. 4 Trends of R_L and R_H of four timing methods when measuring with different t_b

4.4 实验 3 冲刷内核种类对计时偏差影响的评测

本实验评测冲刷内核种类对计时偏差的影响,评测结果如表 4 所列。

在 25 种情况中,R_L 均值为 13.5;最小值为 0.2,出现在前后冲刷内核均为 DGEMM 时;最大值为 19.3,出现在前后冲刷内核分别为 DGEMM 和 ADD 时。R_H 均值为 23.4;最小值为 17.1,出现在前后冲刷内核均为 DGEMM 时;最大值为 24.4,出现在前后冲刷内核分别为 COPY 和 BCAST 时。观察 PAPIX6 计时偏差可发现,当前后冲刷内核均为 DGEMM 时,计时偏差低于其他情况,这可能是由于 DGEMM 内核中浮点计算指令占比较大,而计时函数中访存指令数量占比较大,两者使用的处理器后端端口不同,容易流水线并行。

表 4 不同的前后冲刷内核下的计时相对偏差指标

Table 4 R_L and R_H with different flush kernels

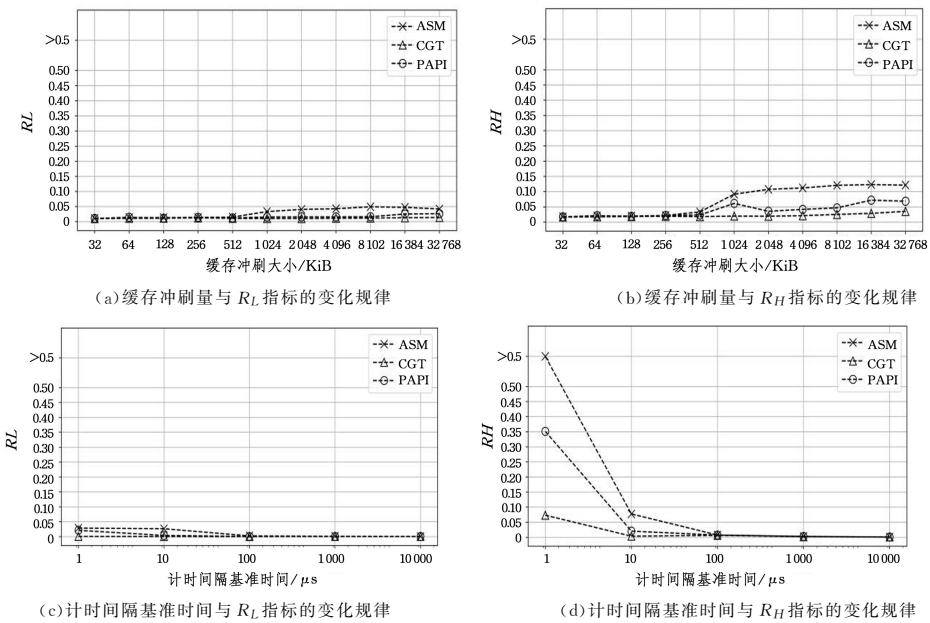
(%)

后冲刷	前冲刷									
	COPY		ADD		POW		DGEMM		BCAST	
	R_L	R_H	R_L	R_H	R_L	R_H	R_L	R_H	R_L	R_H
COPY	14.6	24.3	11.5	23.0	17.2	23.9	14.6	23.8	14.5	23.6
ADD	14.0	23.8	13.8	23.3	15.2	23.5	19.3	23.8	11.2	23.5
POW	17.5	24.2	17.3	23.4	16.6	23.6	8.3	22.9	15.1	23.6
DGEMM	6.6	22.5	15.5	23.9	11.8	23.5	0.2	17.1	8.7	23.5
BCAST	15.1	24.4	17.2	24.2	11.3	23.4	15.2	24.0	15.1	24.2

总体而言,前后计算内核的指令和访存特性,均会影响计时偏差的幅度。对并行计算程序局部代码进行细粒度的性能分析,对测量稳定性和准确度要求较高时,可以通过 ParTES 构造与实际应用指令和访存特征相似的冲刷内核,测定计时结果的偏差幅度。

4.5 实验 4 不同处理器计时偏差对比的评测

本实验在海光、飞腾两款国产处理器上评测了 ASM,CGT 和 PAPI 这 3 种计时方法的计时偏差。计时间隔基准时间从 1μs 延长至 10μs 时,不同缓存冲刷量下 3 种计时方法的 R_H 和 R_L 指标如图 5 和图 6 所示。

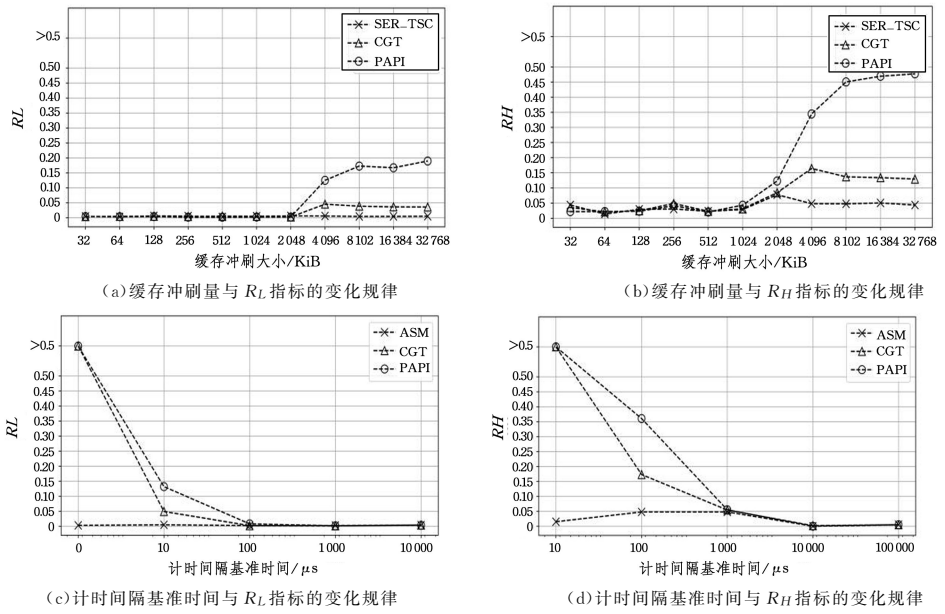


(a)缓存冲刷量与 R_L 指标的变化规律 (b)缓存冲刷量与 R_H 指标的变化规律

(c)计时间隔基准时间与 R_L 指标的变化规律 (d)计时间隔基准时间与 R_H 指标的变化规律

图 5 飞腾处理器上,不同计时间隔基准时间下,4 种计时方法的 R_L 和 R_H 指标的变化趋势

Fig. 5 On the Phytium processor, trends of R_L and R_H of four timing methods when measuring with different t_b



(a)缓存冲刷量与 R_L 指标的变化规律 (b)缓存冲刷量与 R_H 指标的变化规律

(c)计时间隔基准时间与 R_L 指标的变化规律 (d)计时间隔基准时间与 R_H 指标的变化规律

图 6 海光处理器上,不同计时间隔基准时间下,4 种计时方法的 R_L 和 R_H 指标的变化趋势

Fig. 6 On the Hygon processor, trends of R_L and R_H of four timing methods when measuring with different t_b

可以看出,飞腾处理器 3 种计时方法从 1 μs 基准时间起,R_L 始终低于 5%;R_H 指标反映出 3 种计时方法在 1 μs 和 10 μs 均受到长尾偏差的影响,但在 100 μs 后逐渐下降至 5%以下。在海光处理器上,ASM 计时方法最为稳定,R_H 和 R_L 在所有基准时间均低于 5%。总体而言,CGT 和 ASM 分别是飞腾、海光处理器上最准确和最稳定的计时方法。

上述结果体现出,由于指令集、内存子系统、计时方法自身等诸多因素的影响,不同处理器适用的计时方法有明显差别。ParTES 通过兼容多种计时方法,构造理论时间分布和缓存状态的多维度实验,帮助确定了鲲鹏、飞腾和海光处理器上特定测量情形下的最优计时方法。

5 相关工作

在并行计算领域,已有大量研究探讨了性能测量和分析中计时精度与准确性的影响因素。Weaver 等[20-22]发现并研究了并行计算环境中性能计数器读数不稳定的现象,并将其归因于操作系统噪音和应用程序访存特性等多种因素。随后,Rohl 等[7]进一步研究了 PAPI 和 LIKWID 在 STREAM[23]评测中因缓存污染、多核心资源竞争等因素导致的计时波动现象。这些研究定性展示了并行计时中的波动现象,但缺乏对不同处理器普遍规律的深入分析与总结。本研究首次实现了在多种架构、多种处理器上对并行计时波动的定量分析与评价,直观展示了不同计时方法在不同处理器上的稳定性。此外,Hunold 等[11-12]发现 MPI 程序中的性能测量结果不稳定性已影响并行计算实验结果的可复现性。然而,该研究未深入探讨多种不稳定性来源对计时结果的具体影响,且主要针对以英特尔处理器为代表的主流 X86 架构。本研究提出的指标和工具适用于多种计时方法,可评测鲲鹏、海光和飞腾等多种处理器在不同缓存大小下的计时偏差,具有更高的工程实用价值。

针对计时波动造成的计时结果偏差,已有研究者提出多种应对方法。Hoefler 等[8]和 Chen 等[24]提出通过延长计时间隔和回归分析等方法减少计时波动对性能评测的影响。Mccalpin[1]和 Abel 等[25]则建议使用低开销的汇编计时指令并进行大量重复测量,以提高测量指令和微型计算内核耗时的准确度和稳定性。这些研究虽然分析和总结了计时开销与计时波动对性能测量结果的影响,但仍将计时开销作为计时准确性的主要评判标准,在测量准确性分析思路上存在局限性。本研究通过实验证实了"低开销并不意味着低偏差"这一重要结论,并展示了汇编计时指令、clock_gettime()等低开销计时方法在不同处理器型号和运行时环境下未必优于 PAPI 等性能工具提供的计时方法。

结束语 本研究提出了用于定量评价并行计时测量结果分布偏差的量化指标,设计并实现了开源工具 ParTES[1)],可在模拟真实应用程序的并行规模和缓存状态下定量测量计时偏差。本研究对鲲鹏、飞腾和海光 3 个代表性高性能处理器进行了计时偏差测试,首次直观展示了计时波动对这些处理器并行计时结果分布的影响,并通过量化评价确定了不同处理器上计时偏差最低、稳定性最佳的计时方法。

通过 4 组实验,本研究证实了"低开销并不意味着低偏差"的结论,表明性能测量方法的准确性和稳定性需要结合计时方法、硬件配置、缓存状态、计时前后的计算内核种类等因素进行量化评测后才能客观评价。

未来研究方向包括:对 GPU 和 DPU 等异构计算芯片的性能测量工具的准确性展开研究;深入研究处理器变频状态下的计时稳定性;进一步拓展 ParTES 功能,使其兼容 LoongArch 和 RISC-V 架构,并支持更多常见计时方法。

参 考 文 献

[1] MCCALPIN J D. HPL and DGEMM Performance Variability on the Xeon Platinum 8160 Processor[C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Dallas: IEEE Press, 2018: 225-237.

1) https://gitee.com/keyliao/partes

[2] CHUNDURI S,HARMS K,PARKER S,et al. Run-to-run variability on Xeon Phi based cray XC systems[C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. New York: Association for Computing Machinery,2017:1-13.

[3] COOK B,KURTH T,AUSTIN B,et al. Performance Variability on Xeon Phi[C]// International Conference on High Performance Computing. Hamburg:Springer International Publishing, 2017:419-429.

[4] BHATELE A,THIAGARAJAN J J,GROVES T,et al. The Case of Performance Variability on Dragonfly-based Systems [C]// Proceedings 2020 IEEE 34th International Parallel and Distributed Processing Symposium(IPDPS). New Orleans:IEEE Press,2020:896-905.

[5] BHATELE A,MOHROR K,LANGER S H,et al. There goes the neighborhood: Performance degradation due to nearby jobs [C]// Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. Denver: Association for Computing Machinery,2013:1-12.

[6] DAS R,MUTLU O,MOSCIBRODA T,et al. Aergia: exploiting packet latency slack in on-chip networks[J]. ACM SIGARCH Computer Architecture News,2010,38(3):106-116.

[7] RÖHL T,TREIBIG J,HAGER G,et al. Overhead Analysis of Performance Counter Measurements[C]// Proceedings of the 2014 43rd International Conference on Parallel Processing Workshops. Minneapolis: IEEE Computer Society, 2014: 176-185.

[8] HOEFLER T,BELLI R. Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results[C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Austin: Association for Computing Machinery, 2015:1-12.

[9] LIAO Q,LIN J. TacVar: Tackling Variability in Short-Interval Timing Measurements on X86 Processors[C]// 2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid 2024). Philadelphia: IEEE Computer Society, 2024:496-506.

[10] ZHAI J,ZHENG L,SUN J,et al. Leveraging Code Snippets to Detect Variations in the Performance of HPC Systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(12):3558-3574.

[11] HUNOLD S,CARPEN-AMARIE A,TRÄFF J L. Reproducible MPI Micro-Benchmarking Isn't as Easy as You Think[C]// Proceedings of the 21st European MPI Users' Group Meeting. New York, NY, USA: Association for Computing Machinery, 2014:69-76.

[12] HUNOLD S,CARPEN-AMARIE A. Reproducible MPI Benchmarking is Still Not as Easy as You Think[J]. IEEE Transactions on Parallel and Distributed Systems,2016,27(12):3617-3630.

[13] PAOLONI G. How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures[EB/OL]. (2010-09-01)[2024-12-07]. https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf.

[14] KANTOROVICH L V. Mathematical Methods of Organizing and Planning Production[J]. Management Science,1960,6(4): 366-422.

[15] LEONID NISONOVICH VASERSTEIN. Markov Processes over Denumerable Products of Spaces,Describing Large Systems of Automata[J]. Problemy Peredachi Informatsii,1969,5(3): 64-72.

[16] TERPSTRA D,JAGODE H,YOU H,et al. Collecting Performance Data with PAPI-C[C]// Tools for High Performance Computing 2009. Berlin:Springer,2010:157-173.

[17] TREIBIG J,HAGER G,WELLEIN G. LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments[C]// 2010 39th International Conference on Parallel Processing Workshops. San Diego:IEEE,2010:207-216.

[18] KNÜPFER A,RÖSSEL C,MEY D A,et al. Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope,Scalasca,TAU,and Vampir[C]// Tools for High Performance Computing 2011. Berlin:Springer,2012:79-91.

[19] 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.

[20] WEAVER V M,DONGARRA J. Can hardware Performance Counters be Trusted? [C]// 2008 IEEE International Symposium on Workload Characterization. Seattle:IEEE,2008:141-150.

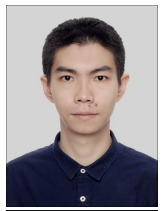
[21] WEAVER V,DONGARRA J. Can Hardware Performance Counters Produce Expected,Deterministic Results? [EB/OL]. (2010-12-01)[2024-12-07]. https://icl.utk.edu/files/publications/2010/icl-utk-451-2010.pdf.

[22] WEAVER V M,TERPSTRA D,MOORE S. Non-determinism and Overcount on Modern Hardware Performance Counter implementations[C]// 2013 IEEE International Symposium on Performance Analysis of Systems and Software(ISPASS). Austin:IEEE,2013:215-224.

[23] MCCALPIN J. Memory Bandwidth and Machine Balance in High Performance Computers[C]// IEEE Technical Committee on Computer Architecture Newsletter. 1995:19-25.

[24] CHEN T,GUO Q,TEMAM O,et al. Statistical Performance Comparisons of Computers[J]. IEEE Transactions on Computers,2015,64(5):1442-1455.

[25] ABEL A,REINEKE J. nanoBench: A Low-Overhead Tool for Running Microbenchmarks on x86 Systems[C]// 2020 IEEE International Symposium on Performance Analysis of Systems and Software(ISPASS). Boston:IEEE,2020:34-46.



LIAO Qiucheng, born in 1994, engineer, is a member of CCF(No. P6171M). His main research interests include high-performance computing and so on.

LIN Xinhua, born in 1979, Ph.D, senior engineer, Ph.D supervisor, is a distinguished member of CCF(No. 23737D). His main research interests include high-performance computing and so on.

(责任编辑:李亚辉)