arXiv:2603.26576v1 [cs.DC] 27 Mar 2026

面向加速系统的硬件无关且直观的效率指标：TALP 中的定义与实现

Ghazal Rahimi[1 [0009−0000−4946−5229]]、Victor Lopez[1 [0000−0002−3113−9166]]、Marc Clasca[1 [0000−0002−5963−6658]]、Joan Vinyals Ylla Català[1 [0000−0002−4711−6815]]、Jesus Labarta[1 [0000−0002−7489−4727]] 和 Marta Garcia-Gasulla[1 [0000−0003−3682−9905]]

巴塞罗那超级计算中心，西班牙
{grahimi,vlopez,mclasca,jvinyals,jlabarta,martag}@bsc.es

摘要。 在高性能计算（HPC）中，将 CPU 与 GPU 等加速器相结合的异构平台日益普及，这为性能分析和优化带来了新的挑战。传统效率指标，例如性能优化与生产力（POP）卓越中心提出的指标，主要针对同构 CPU 系统设计，因此无法反映主机和设备资源之间复杂的交互。在本研究中，我们通过引入一套分别评估主机效率和设备效率的新指标层级，将 POP 效率框架扩展至异构架构。在主机端，我们量化混合执行和卸载操作的有效性。在设备端，我们提出了一个与主机层级类似的乘法层级，并定义了其中的并行效率分支。除了定义和公式化这些指标，我们还介绍了它们在 DLB 库 TALP 模块中的实现。TALP 是一个轻量级监控库，可在事后和运行时提供测量结果，并支持以文本和机器可读格式输出。我们通过合成基准测试和三个生产级高性能计算应用验证了所提出的框架，展示了这些指标如何揭示卸载、负载均衡和编排方面的效率低下。结果表明，扩展后的 TALP 指标能够提供切实可行的洞见，指导开发者优化异构高性能计算代码。

关键词： 高性能计算 · 加速器 · 效率指标 · 性能分析

1 引言

从地球与生命科学到计算机科学和工程学，高性能计算（HPC）正迅速改变世界，帮助科学家和工程师更高效地应对复杂挑战。自 1965 年以来，摩尔定律一直应验，预测大约每两年

集成电路上的晶体管数量就会翻一番，从而以几乎不增加成本的方式，带来效率更高、性能更强的处理器。

然而，由于暗硅等物理和技术限制，摩尔定律如今已接近极限。为了继续提升并行性和性能，人们引入了新型硬件——加速器，尤其是图形处理器（GPU）。

因此，采用 CPU 与通用 GPU 等加速器相结合架构的异构计算，已成为满足人工智能和高性能计算工作负载高计算需求的合适方案。配备 GPU 的系统可利用数千个简单的小型核心并发执行操作，从而实现极高的每秒浮点运算次数（FLOPS）。GPU 在高性能计算中的重要性，从最新的 TOP500 榜单 [22],中显而易见：排名前十的系统有九个包含加速分区。这反映了高性能计算设计中的一个更广泛趋势：从传统的通用处理器转向采用专用内存和通信模型的众核加速器。

随着异构计算系统的出现，高性能计算应用开发者和用户面临的首要挑战是将代码移植到这些新架构上。GPU 在 20 多年前改变了高性能计算领域的格局，此后已有相当一部分高性能计算代码成功适配，以充分利用 GPU。然而，当前的挑战在于开发和维护可移植代码，使其能够在各种异构系统和基于加速器的系统上高效运行。

因此，对运行在现代异构系统上的应用程序进行性能测量，不仅变得更具挑战性，也日益重要。硬件设备种类不断增加，例如不同厂商生产的 GPU，使分析过程变得更加复杂。现有的大多数性能工具都针对特定厂商，旨在提供有关相应硬件架构利用情况的详细信息。尽管这些细节有其价值，却往往令用户应接不暇，使非专业人士难以解读结果，并找出代码中的关键问题或瓶颈。缺乏标准化、可移植且易于使用的工具，进一步增加了任务的难度，导致许多开发者在异构环境中优化性能时缺乏明确指引。

本文介绍了一组适用于主机和设备架构的效率指标，扩展了 POP 并行效率框架。这些新指标旨在不受底层架构以及用于将工作卸载到设备上的编程模型的限制。它们能让用户全面了解应用程序中效率损失的主要来源，涵盖主机和设备上的执行情况。这些指标已在 TALP [18]（应用程序实时性能跟踪）中实现。TALP 是一款轻量级性能分析工具，专为在生产级高性能计算运行期间提供并行效率指标而开发。

本文的主要贡献如下：

– **将 POP 指标扩展至异构加速系统：**我们引入适用于基于加速器架构的硬件无关效率指标，补充了成熟的 POP 方法。– **对新指标进行形式化定义：**我们为其提供严谨的数学基础，以确保定义清晰、结果可复现，并适用于不同平台。– **在 TALP 框架中实现：**我们对所提出的指标进行了概念验证式集成，目前已应用于 NVIDIA GPU。– **在不同场景中进行评估：**我们通过一项合成基准测试和三个科学生产用例，验证了这些指标的有用性。

本文其余部分安排如下：第 2 节回顾性能工具的最新发展。第 3 节介绍原有的 MPI POP 指标和 TALP 框架。第 4 节说明新指标及其公式、解释，以及在 TALP 中的实现方式。第 5 节展示将这些指标应用于不同用例所得的结果。最后，第 6 节总结本文的结论和未来工作。

2 相关工作

在高性能计算领域，编写正确、功能完善且能够扩展并高效执行的代码正变得愈发复杂。并行性是在各种异构架构上运行应用程序的关键要求，这些架构既包括共享内存系统，也包括由 CPU（可连接或不连接 GPU）组成节点的分布式内存系统。欧洲高性能计算社区正推动计算领域快速发展，TOP500排名便是明证：榜单上的全球前 10 大系统中，有 5 个位于欧洲。这一进步的关键特征之一，是专注于提升并行计算性能和效率。PRACE [20]（欧洲先进计算伙伴关系）等倡议通过支持 POP 卓越中心（POP CoE）[25]开发的工具和方法，在推动高性能计算性能分析方面发挥了重要作用。在三个 POP 项目期间，他们帮助用户发现性能问题并采用最佳实践。这套方法的一个关键要素是使用性能分析工具，它们构成了 POP 优化高性能计算应用的基础。

每种性能工具都有其优势和局限，因此根据其功能、需求或经验，它们适用于不同的用例和开发者。Eriksson等人在白皮书中全面概述了欧洲高性能计算社区使用的工具[21]，，对所有性能分析和跟踪工具进行了比较。本文提及的工具如下：1. *HPCToolkit* [15]2. *ITAC* [5] 3. *Extrae* 和 *Paraver* [23] [19] 以及 4.*Scalasca* [14]。我们简要讨论最相关的跟踪工具，例如 Extrae、Paraver 和 Scalasca

[26]。最后，我们介绍最具影响力的性能分析工具，例如 TAU [24],gprof [16]和 HPCToolkit [15]。

Paraver 和 Extrae [23] 由巴塞罗那超级计算中心开发。Paraver 是一款用于定性分析和定量计算的数据可视化浏览器。Extrae 是一种跟踪工具，它使用多种机制以透明方式捕获性能数据，范围涵盖对未经修改的生产环境二进制文件进行动态插桩，以及静态链接。

Scalasca 由德国的于利希超级计算中心开发。用于 Scalasca [14] 分析的应用程序代码通过 Score-P [17] 插桩器进行准备，该插桩器会为各种测量接口配置适配器，并链接相关的测量库。

TAU [24] 性能系统是一款性能剖析与跟踪工具包，用于分析以 CUDA C、CUDA C++、OpenCL 编写，或使用 pyCUDA 或 OpenACC 的混合并行程序。TAU 通过对函数、方法、基本块和语句进行插桩来收集 GPU 计算的性能信息，并将其与其他应用程序性能数据相结合，从而全面呈现最终应用程序执行的性能情况。

gprof [16] 是一款性能分析工具，用于分析应用程序，以确定程序执行期间时间的消耗情况。大多数 Unix/Linux 实现都包含 gprof；它易于使用，能够快速显示应用程序中耗时最多的部分。gprof 的工作原理是在编译期间自动为代码插桩，然后在执行期间对应用程序的程序计数器进行采样。

最后，我们讨论 HPCToolkit [15], ，这是一个用于在从多核桌面计算机到 GPU 加速超级计算机等各种系统上，对应用程序进行性能测量和分析的集成工具套件。该工具包通过对 CPU 上的计时器和硬件计数器进行统计采样，以及监控 GPU 上的活动和事件，为 CPU 和 GPU 执行及其潜在低效之处提供洞察。

PRACE [21] 报告称，跟踪工具在高性能计算生态系统中占据主导地位，可用的独立性能剖析工具相对较少。然而，性能分析依赖于性能剖析工具和跟踪工具，因为二者承担着不同的职责。跟踪工具会在整个执行过程中记录细粒度、详尽的带时间戳事件，例如函数调用、通信事件、同步和内存访问。相比之下，性能分析工具会提供程序行为的高层统计摘要，尤其有助于发现 CPU 和 GPU 上的负载不均衡及低效问题。在大型高性能计算应用中，完整跟踪可能会产生海量数据，因此性能剖析首先突出关键性能热点，发挥着至关重要的作用。这使用户能够更有效地安排跟踪工作，从而让整体性能分析流程更具可扩展性且更易管理。HPCToolkit、TAU 和 Score-P 均支持性能剖析和跟踪；相比之下，gprof 仅提供性能剖析功能。这些工具可以测量内核执行时长、内存分配与释放、内存复制、同步事件、GPU 时间等。不过，它们本身无法表明测得的时间是否反映了正常行为，还是

性能瓶颈，因此其结果可能难以解读。相比之下，TALP 不仅会报告原始执行时间和指标，使性能工程师能够自行计算并创建自定义指标，还提供了实用且可直接使用的 POP 指标及其 GPU 指标扩展，这些对于分析高性能计算应用非常有帮助。

此外，还有一些厂商专属工具，例如 NVIDIA Nsight Systems[9]，，这是一款跟踪工具；以及 Nsight Compute [8],，这是一款性能分析工具。尽管这些性能工具能够为在支持 CUDA 的设备上运行的应用程序提供大量信息，但它们并不适用于其他加速器和厂商，例如 AMD 或 Intel。TALP 和全新的 TALP 指标不受厂商限制，可为性能分析提供支持和协助。此外，TALP 性能分析器和 DLB 库均为开源软件，因此任何开发者都能更轻松地使用这些指标并在必要时进行修改。

3 背景

3.1 DLB

DLB [11]（动态负载均衡）是一款模块化运行时库，旨在以非侵入且透明的方式提升并行应用的性能和可扩展性。它可与消息传递接口、OpenMP 和 OmpSs 等广泛使用的编程模型无缝集成，从而确保其在高性能计算环境中的可移植性。此外，DLB 提供用户级 API，使高级用户能够进行细粒度控制，并便于与资源管理器等外部组件集成。

当前版本的 DLB 包含三个模块：

– **LeWI**（空闲时借出 [12, 13]）：该模块利用第二级并行（OpenMP 或 OmpSs）改善消息传递接口应用的负载均衡。它会动态调整共享内存编程模型所使用的线程数，从而优化节点计算资源的利用率。这些调整对应用程序透明，无需用户进行任何修改。–**DROM**（动态资源所有权管理 [10]）：允许资源管理器、用户或应用程序在运行期间更改进程间的线程分配，以最大限度地提高性能或效率，而无需停止正在运行的进程。

– **TALP**（应用程序实时性能跟踪）：以极低的开销对应用程序性能进行轻量级在线监控。TALP 通过实时提供效率指标，使用户和工具能够识别瓶颈、指导优化工作并支持自适应运行时决策。该模块旨在做到无侵入、可移植，并与传统的离线性能分析方法互为补充。

3.2 TALP

TALP [18] 是一款可移植、可扩展、轻量且可伸缩的并行性能测量与监控工具。它支持在线和事后性能分析：可以在应用程序执行期间收集指标，以指导运行时决策；也可以在执行结束后检查指标，以评估完整生产运行的效率。

TALP 提供基于成熟 POP 方法的并行效率指标，并结合硬件计数器信息和原始指标，让用户能够定义并计算自定义指标。

该工具依赖 PMPI 和 OMPT 等标准机制，确保应用程序能够透明地使用，无需修改代码。对于更高级的场景，TALP 还提供用户级 API，使开发人员能够为代码添加注释，并获取特定感兴趣区域的指标。

部署 TALP 的唯一要求是，系统能够在运行时预加载或覆盖库。例如，在 Linux 系统上，动态链接器 ld 支持通过 LD_PRELOAD 环境变量预加载共享对象。实际上，要使用 TALP 收集所有性能指标，只需在解析消息传递接口符号之前动态加载 DLB 库即可。

TALP 的事后分析输出既可以采用便于阅读的纯文本格式，也可以采用 JSON 文件格式，从而支持自动化处理并与数据分析工作流集成。

3.3 POP MPI 指标

TALP 收集的主要效率指标源自 POP CoE 项目提出的方法。

POP 指标以乘法结构进行层次化组织，其中每个父指标都定义为其子指标的乘积。不同于传统的性能剖析方法，这些指标侧重于效率测量，突出显示性能损失，而非仅仅报告原始计时数据。POP MPI 指标的层次化组织如图 1 所示。

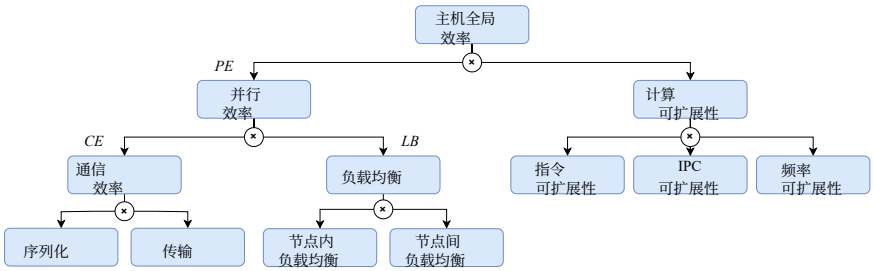


图 1. POP 消息传递接口指标的层次结构

这些指标基于一种简化的执行模型：每个进程会在两种状态之间交替切换：执行计算时处于 有用状态，停滞时（例如，等待通信）则处于 无用状态。本节聚焦于 MPI 应用，因此，无用状态特指在 MPI 调用中花费的时间。

该层次结构区分了两大类指标：效率和可扩展性。效率指标通过衡量无用时间所占的比例，量化给定执行过程中性能损失的主要来源。相比之下，可扩展性指标是相对于基准情况定义的；它们评估有用 计算量如何随不同运行而变化，并将偏差归因于特定的扩展因素。

由于 TALP 报告的是单次运行的指标，因此只能获得效率指标。不过，借助 TALP 收集的硬件计数器，用户可以计算多次 TALP 运行的可扩展性指标。

本节余下部分将定义效率指标，这些指标将在后续各节中扩展。所有新指标都属于并行效率分支。不过，完整的指标树还包括第二个分支——计算可扩展性，本文将不讨论这一分支。

我们将 $P = \{p_1, \ldots, p_n\}$ 定义为 MPI 进程集合，将 n 定义为 MPI 进程数量。对于每个 MPI 进程 p，我们将应用程序执行有效计算的时间间隔集合定义为 $U_p = \{u_1^p, u_2^p, \ldots, u_{|U|}^p\}$。我们将进程 p 中所有有用时间段的持续时间总和定义为 D_{U_p}（见公式 (2)）。类似地，我们可以为无用时间间隔定义 U_p 和 D_{U_p}。我们还将经过时间定义为 经过时间 E，如公式 1 所示。

$$E = max_{i=1}^{n} D_{U_i} + D_{\overline{U}_i} \tag{1}$$

$$D_{U_p} = \sum_{j=1}^{|U_p|} u_j^p \tag{2}$$

$$PE = \frac{\sum_{i=1}^{n} D_{U_i}}{E * n} \tag{3}$$

$$LB = \frac{\sum_{i=1}^{n} D_{U_i}}{n * \max_{i=1}^{n} D_{U_i}} \tag{4}$$

$$CE = \frac{\max_{i=1}^{n} D_{U_i}}{E} \tag{5}$$

并行效率指标（PE）表示并行化导致的时间损失，也可以等价地表示为有效计算时间与总经过时间之比，如公式 3 所示。

并行效率包含两个子指标：负载均衡（LB）和通信效率（CE），其定义分别见公式 4 和 5。

负载均衡衡量进程间不均衡工作负载（有效计算）造成的效率损失，其计算方式为平均有效持续时间与负载最重进程的有效持续时间之比。

通信效率量化了通信所耗时间中并非由负载不均衡导致的部分。

4 加速平台的 TALP 效率指标

4.1 指标的制定与说明

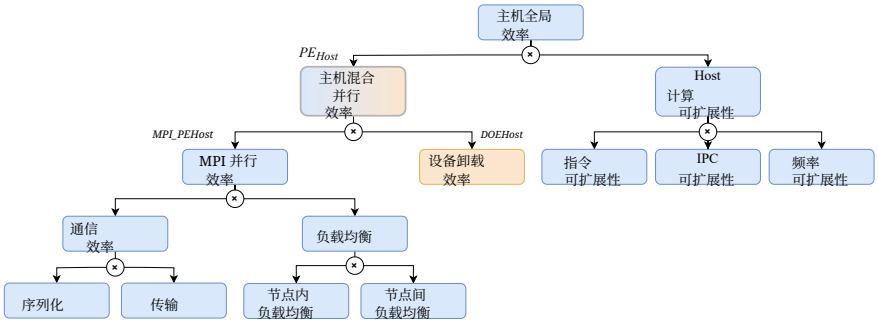


图2. 主机资源效率指标的扩展层级结构。橙色方框表示新增的指标

我们为异构平台提出的效率指标扩展了第3.3节中介绍的层次结构。与原始表述相比，关键区别在于，该层次结构被划分为两棵互不相交的树，分别对应主机和设备资源。

第一个层次结构扩展了原始 POP 模型，用于评估主机侧资源利用的效率。图 2 展示了这一结构，其中以橙色突出显示了新引入的指标：主机混合并行效率和设备卸载效率。主机混合并行效率定义为 MPI 并行效率与卸载效率的乘积。卸载效率指标用于衡量将工作从主机交由设备处理时损失的效率比例。这包括 CPU 因数据传输、内核启动和同步等 GPU 相关操作而处于阻塞状态的时段。因此，该指标量化了使用加速器所产生的效率低下情况。

图 3 展示了设备效率指标的层次结构。与主机侧层次结构类似，设备指标分为两个分支：设备并行效率，用于衡量设备资源是否得到积极利用；设备计算效率，用于衡量这些资源得到有效利用的程度。本文重点讨论设备并行效率分支，将设备计算效率的定义留待未来工作。

设备并行效率衡量设备用于执行有效工作的时间占比，其中有效工作定义为内核执行。它表示为三个子指标的乘积：负载均衡、通信效率和编排效率。

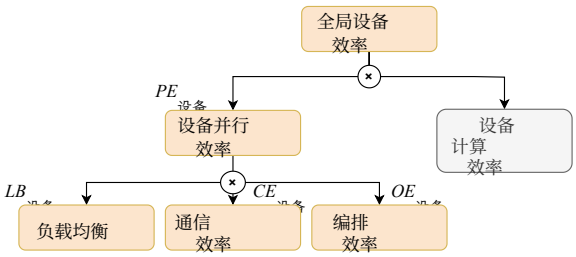


图3. 针对设备资源提出的效率指标层级结构

负载均衡量化设备间内核执行时间不均所导致的效率低下。通信效率衡量数据传输造成的效率损失，无论是在设备之间（GPU–GPU）还是在主机与设备之间（CPU–GPU）。编排效率反映没有安排任何有效工作的空闲时段，体现了在协调计算、通信和主机卸载方面的效率低下。

如第3.3节所述，原始 POP 效率指标通过将进程执行划分为两种状态来计算：有用 和无用。为了将这些指标扩展到异构平台，我们在主机和设备上分别定义三种状态。

对于在主机上运行的进程，这三种状态为：(i) 有用、(ii) 设备卸载和 (iii) *MPI* 调用。令 D_{U_p} 表示进程p的总有用 时间， D_{W_p} 表示总设备卸载 时间。

主机混合并行效率 (PE_{host}) 定义为有效计算时间与总经过 CPU 时间之比（公式 6）。

$$PE_{host} = \frac{\sum_{i=1}^{n} D_{U_i}}{E \cdot n} \tag{6}$$

MPI 并行效率 (MPI_PE_{host}) 同时计入有用时间和卸载时间，并将二者视为有用，再将它们的总和除以总经过 CPU 时间（公式 7）；其子指标的公式从略，但对状态采用相同的处理方式。

$$MPI_PE_{host} = \frac{\sum_{i=1}^{n} (D_{U_i} + D_{W_i})}{E \cdot n} \tag{7}$$

设备卸载效率 (OE_{host}) 是有效计算时间与有用时间和卸载时间之和的比值（公式 8）。

$$OE_{host} = \frac{\sum_{i=1}^{n} D_{U_i}}{\sum_{i=1}^{n} (D_{U_i} + D_{W_i})} \tag{8}$$

对于设备指标，我们同样将执行划分为三种状态，并将每个 GPU 视为单一资源（此处不区分流；这将在未来的设备计算效率指标中纳入）。这三种状态

分别为：(i) 内核计算 （有效工作）、(ii) 内存操作和 (iii) 空闲。值得注意的是，由于这些指标是在设备级别定义的，计算流与通信流之间的任何重叠都会计为计算时间。

设 D_{K_g} 为 GPU g 执行内核所花费的时间， D_{M_g} 为内存操作所花费的时间， m 为设备数量。设备效率指标定义如下：

$$PE_{device} = \frac{\sum_{i=1}^{m} D_{K_i}}{E \cdot m} \quad (9) \qquad LB_{device} = \frac{\sum_{i=1}^{m} D_{K_i}}{m \cdot \max_{i=1}^{m} D_{K_i}} \quad (10)$$

$$CE_{device} = \frac{\max_{i=1}^{m} D_{K_i}}{\max_{i=1}^{m}(D_{K_i} + D_{M_i})} \quad (11) \qquad OE_{device} = \frac{\max_{i=1}^{m}(D_{K_i} + D_{M_i})}{E} \quad (12)$$

设备并行效率是执行内核的有用时间与总经过时间的比值（公式 9）。

负载均衡是所有设备上的平均有用时长与最长内核持续时间的比值（公式 10）。

通信效率是负载最重的设备上的有用时间，与该设备有用时间和空闲时间最少的设备上的内存传输时间之和的比值（公式 11）。

编排效率是空闲时间最少的进程所占时间与总经过时间的比值（公式 12）。

4.2 实现

GPU 效率指标通过模块化的插件式设计集成到了 DLB 库的 TALP 模块中。当前实现支持通过 CUDA 和 OpenACC 使用 NVIDIA GPU，以及通过 HIP 使用 AMD GPU。

目前已实现两个后端：一个基于 NVIDIA 的 CUPTI[7]接口（可选地结合 OpenACC 运行时钩子），另一个基于 AMD 的 rocprofiler2 [1] 库。每个插件都会在主库启动期间初始化，并根据可用的运行时环境启用相应组件。例如，只有检测到 OpenACC 库时，才会启用 OpenACC 插桩。

每个插件都实现两种互补路径：(i) 通过运行时回调同步监控主机 API 调用；(ii) 通过运行时提供的缓冲区异步收集设备活动记录。主机端回调用于测量内核启动、内存传输或同步事件等运行时调用的持续时间，并立即将这些信息转发给 TALP 模块。另一方面，设备端记录会在缓冲区填满或显式刷新后以批次形式交付，并且需要经过后处理，才能分类为 TALP 指标。

在 NVIDIA 平台上，通过启用 CUPTI 的运行时回调域（ cuptiEnableDomain 和 CUPTI_CB_DOMAIN_RUNTIME_API 参数）来监控主机活动；每次调用 CUDA 运行时 API（例如内核启动、内存操作或同步调用）时，都会触发回调。设备活动则通过 CUPTI 活动 API（cuptiActivityEnable）收集，方法是启用多种活动类型，包括内核执行、内存传输和内存操作。

当 OpenACC 库可用时，系统会通过 acc_prof_register 接口拦截其他运行时事件，该接口用于注册 OpenACC 特定操作的回调。这使 TALP 能够扩展其可见范围，涵盖依赖基于指令的编程的应用程序。

在 AMD 平台上，我们使用 rocprofiler_create_filter，并通过 ROCPROFILER_API_TRACE 参数收集主机活动，同时调用 rocprofiler_set_api_trace_sync_callback 函数，以获取每次 HIP 运行时 API 调用的回调。对于设备活动，我们会使用 ROCPROFILER_DISPATCH_TIMESTAMPS_COLLECTION 参数创建单独的筛选器，该筛选器会异步传送内核执行记录。

设备活动记录传送到位后，插件会执行统一的后处理步骤，不受后端影响：

– 内核执行记录会被展平，将跨流重叠的启动合并为一个连续的执行区间。– 内存传输记录也会以相同方式展平，并移除与内核时间区间重叠的片段，以避免重复计数。– 其余未覆盖的区域会被归类为非活动时间，反映 GPU 利用不足。

最终，我们能够将 GPU 状态占用精确拆分为计算、通信和空闲时段。这种分类构成了 TALP 设备指标的基础，其中包括设备并行效率。

5 评估

本节将通过分析一组具有代表性的用例，展示 TALP 中实现的所提指标在实际中的适用性。对于每个用例，我们都会呈现 TALP 生成的指标，指出从效率测量中得出的见解，并将这些指标与同一执行过程的跟踪记录进行比较，以验证其准确性。

我们介绍两类用例。首先，我们使用合成基准测试构建简单的执行模式，并报告相应的指标。这有助于我们在受控场景中验证这些指标的表现。

其次，我们使用三种科研生产级高性能计算代码评估扩展版 TALP。这些实际应用展示了所提出的指标如何揭示大规模异构工作负载中的效率问题。

对于每个用例，我们都提供计算所得指标的表格以及一份或多份执行跟踪。跟踪记录可直观地确认所报告的指标与观察到的行为一致，从而确保这些指标既切合实际，又能代表工作负载特征。

5.1 PILS

PILS 是一种合成微基准测试，最初旨在模拟呈现各种负载不均衡情形的应用程序。其初始实现采用 C 语言开发，并使用消息传递接口、OpenMP 和 OmpSs 进行并行化。更新后的版本通过 CUDA 将 PILS 扩展至 GPU，同时保留基于消息传递接口的分布式执行。此扩展能够生成受控工作负载，涉及 CPU 和 GPU 计算以及内存操作。

在下文所示的实验中，所有用例均使用两个消息传递接口进程，每个进程绑定到一个 CPU核心和一个 GPU。在 PILS 基准测试跟踪中，CPU 状态的可视化方式如下：有效计算以蓝色表示，卸载至 GPU 的任务以橙色表示，消息传递接口通信以红色表示。设备状态则以蓝色表示有效内核执行，以绿色表示内存操作，以灰色表示空闲时段。

用例 1：GPU 满载，CPU 利用率不足，负载均衡良好.

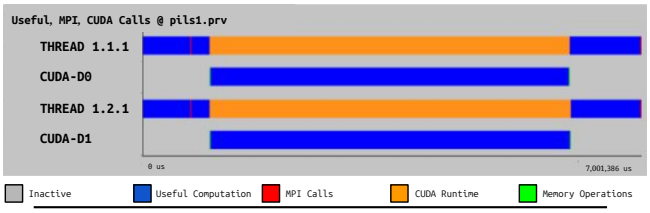
此用例展示了一个大部分工作都卸载到设备上执行的应用程序，CPU 仅用于初始化、收尾以及将工作卸载到 GPU；各 GPU 之间以及运行于 CPU 上的各消息传递接口进程之间的工作负载均衡良好。

图4（上）展示了此用例的执行跟踪记录。我们观察到，在各自的硬件上，CPU 和 GPU 工作负载几乎完美均衡，且 GPU 计算量约为 CPU 计算量的 $10\times$ 倍。图 4 下半部分展示了此次执行的 TALP 输出。除设备卸载效率和编排效率外，所有指标均显示效率为 100%。

从主机端指标来看，我们可以得出结论：CPU 主要用于将任务卸载到 GPU，而非自行执行有用计算，这一点反映在较低的设备卸载效率上。此用例中未观察到显著的消息传递接口通信。

从设备端指标来看，GPU 得到了高效利用。82% 的编排效率表明，GPU 在等待 CPU 卸载工作时会出现一些空闲时间。

针对这种模式，一种潜在的优化方案是让 CPU 计算与 GPU内核执行重叠。该策略可以更有效地利用主机资源，从而提高设备卸载效率，同时保持较高的 GPU 利用率。



	指标	Eff.
Host	并行效率	0.16
	├ 消息传递接口并行效率	1.00
	│ ├ 通信	1.00
	│ └ 负载均衡	1.00
	└ 设备卸载效率	0.16
Device	并行效率	0.82
	├ 负载均衡	1.00
	├ 通信效率	1.00
	└ 编排效率	0.82

图 4. 用例 1：GPU 负载较高、CPU 利用率不足，但整体均衡

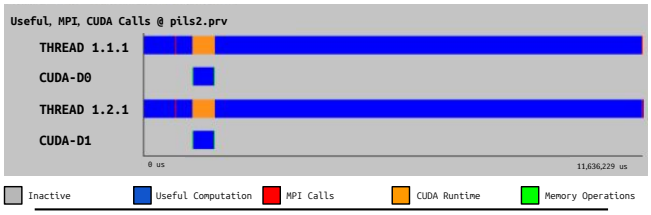
用例 2：CPU 负载较高、GPU 利用率不足，但整体均衡。

第二种用例代表这样一种场景：大部分工作负载都在主机上执行，卸载到设备的工作量不足。不过，工作负载在 CPU 和 GPU 之间仍保持良好平衡。

图 5（上）展示了此用例的执行跟踪记录。我们观察到，CPU 计算量约比 GPU 计算量高 $10\times$，同时 CPU 和 GPU 内部均衡良好。图 5（下）所示的 TALP 输出报告了出色的主机端效率指标，均接近 100%。设备卸载效率达到 94%，这表明 CPU 大部分时间都在执行有效计算，只有少部分时间用于卸载任务或等待 GPU 内核完成。

相比之下，设备端效率指标非常低，设备并行效率仅为 5%。这表明 GPU 仅在执行时间中微不足道的一小部分用于有效工作。

根据这些结果，我们可以得出结论：执行过程主要受主机端计算所支配，而加速器大多未得到充分利用。为了提升性能，应重新组织应用程序，增加卸载到 GPU 的工作负载比例，从而充分利用其计算能力。



	指标	Eff.
Host	并行效率	0.94
	├ 消息传递接口并行效率	1.00
	│ ├ 通信	1.00
	│ └ 负载均衡	1.00
	└ 设备卸载效率	0.94
Device	并行效率	0.05
	├ 负载均衡	1.00
	├ 通信效率	1.00
	└ 编排效率	0.05

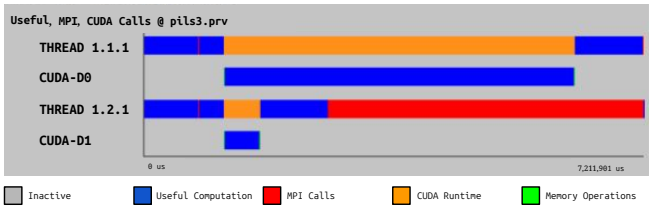
图 5. 用例 2：CPU 负载较高，GPU 利用率不足，负载均衡良好

用例 3：GPU 负载较高，GPU 计算不均衡，CPU 利用率不足 。

此用例模拟了一种应用程序，其中大部分计算都卸载到了 GPU，但各设备之间的工作负载严重不均衡。

在图 6 顶部所示的跟踪记录中，我们观察到，GPU0（来自 MPI 进程 0）执行的工作量比 GPU1 和两颗 CPU 各自多约 $10\times$ 。如橙色部分所示，MPI 进程 0 大部分时间都在执行 CUDA 相关调用，而 MPI 进程 1（运行在 CPU1 上）大部分时间都用于 MPI通信（红色），等待进程 0 完成。

图6底部显示的TALP 指标报告了非常低的总体效率。在主机端，设备卸载效率仅为26%，这表明CPU大部分时间都处于空闲状态，等待GPU完成其工作负载，而不是执行有效工作。同样，主机 MPI 并行效率也很低，反映出两个进程之间存在不均衡。值得注意的是，尽管CPU执行的有效工作不存在负载不均衡，但在消息传递接口层面仍显示出负载不均衡。这是因为在主机端计算负载不均衡时，我们将卸载工作到GPU所花费的时间也纳入了考量。这样设计是有意为之，因为我们假设卸载到GPU的工作属于分配给相应消息传递接口进程的工作，因此分配给不同消息传递接口进程的负载并不均衡，效率指标也体现了这一点。



	指标	Eff.
Host	并行效率	0.16
	├ 消息传递接口 并行效率	0.63
	│ ├ 通信	1.00
	│ └ 负载均衡	0.63
	└ 设备卸载效率	0.26
Device	并行效率	0.45
	├ 负载均衡	0.55
	├ 通信效率	1.00
	└ 编排效率	0.82

图 6. 用例 3：GPU 负载较高，GPU 计算负载不均衡，CPU 利用率不足

在设备端，负载均衡指标为55%，这证实了GPU 工作负载的分配并不均匀：一块GPU始终处于满负荷状态，而另一块在执行过程中的很长一段时间里基本处于空闲状态。

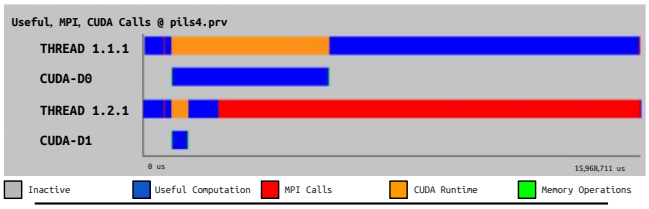
改善设备间的工作负载分配，可显著提升设备负载均衡和主机消息传递接口并行效率；更好地编排计算和数据传输，则有助于减少 GPU 空闲时间。

用例 4：GPU 和 CPU 负载不均衡，CPU 的负载高于 GPU。

用例 4 展示了一个在主机和设备层面都存在负载不均衡的应用程序。

从图 7 上方所示的跟踪记录中可以看到，MPI 进程 0 将一大块工作卸载到其 GPU（蓝色），等待其完成（橙色），随后进行一段较长的计算（蓝色）。而 MPI 进程 1 卸载到 GPU 的工作少得多；等待 GPU 完成后，MPI 进程 1 进行一小段计算（蓝色），之后等待 MPI 进程 0（红色）。

图 7 底部所示的 TALP 输出表明，主机上的并行效率较低。负载均衡为 55%，说明两个 MPI 进程之间存在严重的负载不均衡。设备卸载效率表明，CPU 在等待 GPU 计算完成期间未得到充分利用。另一方面，设备指标显示负载均衡为 55%，说明各设备之间的工作负载分配不均。较低的



	指标	Eff.
Host	并行效率	0.36
	├ 消息传递接口并行效率	0.55
	│ ├ 通信	1.00
	│ └ 负载均衡	0.55
	└ 设备卸载效率	0.65
Device	并行效率	0.18
	├ 负载均衡	0.55
	├ 通信效率	1.00
	└ 编排效率	0.33

图 7. 用例 4：GPU 和 CPU 负载不均衡，CPU 负载高于 GPU

编排效率表明，GPU 未得到充分利用，因为卸载到 GPU 的工作量不足；从执行结束时 MPI 进程 0 完成的一大段工作也可以看出这一点。

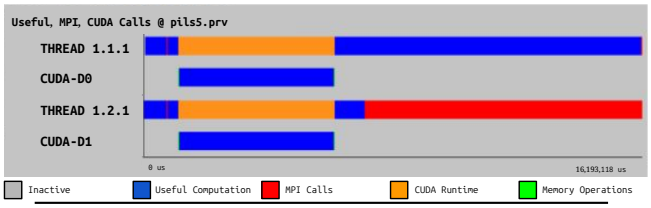
在这种情况下，至关重要的是更好地将工作负载分配给 CPU 和 GPU，以提高代码的性能。

用例 5：CPU 负载不均衡，CPU 和 GPU 的全局负载相同。

这种模式代表一个应用程序，其中 CPU 和 GPU 的工作负载相同，但 CPU 负载没有在两个 MPI 进程之间均匀分配。我们可以在图 8 的跟踪记录中观察到这种模式。可以看到，两个 MPI 进程都会将工作卸载到各自对应的 GPU，并等待 GPU 完成工作（橙色）。GPU 计算完成后，两个 MPI 进程之间的工作量出现不均衡，MPI 进程 1 会等待 MPI 进程 0 完成工作（红色）。

图 8 底部报告的 TALP 指标显示，主机和设备的并行效率都很低。对于主机，我们可以看到负载均衡较低（70%），这表明两个 MPI 进程之间的工作负载分配不均；设备卸载效率也较低，这表明在等待 GPU 完成工作时，CPU 没有用于执行有效计算。

设备效率指标显示，编排效率为 33%，这表明卸载到设备上的工作量不足，设备大部分时间都处于空闲状态，等待任务。



	指标	Eff.
Host	并行效率	0.36
	├ MPI 并行效率	0.70
	│ ├ 通信	1.00
	│ └ 负载均衡	0.70
	└ 设备卸载效率	0.51
Device	并行效率	0.33
	├ 负载均衡	1.00
	├ 通信效率	1.00
	└ 编排效率	0.33

图 8. 用例 5：CPU 负载不均衡，CPU 和 GPU 的全局负载相同

通过更好地分配 MPI 进程的工作负载，并将更多工作从 CPU 卸载到 GPU，可以改善这种模式。

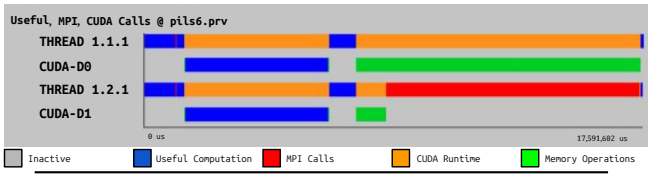
用例 6：工作均匀分布，主机与设备之间的数据传输量大。

用例 6 展示了一种模式：计算任务在 GPU 和 CPU 之间分布均匀，大部分工作卸载到了 GPU 上，但主机与设备之间存在大量数据传输。

在图 9（上图）中，我们可以看到代表此用例的跟踪记录。可以看出，两个消息传递接口进程的行为相同，直到执行结束时，消息传递接口进程 0 需要从设备（绿色）传输一大块数据，而消息传递接口进程 1 则在消息传递接口阻塞调用中等待（红色）。

获得的 TALP 指标（图 9 下图）显示，主机上的并行效率非常低。主要瓶颈是设备卸载效率，该效率仅为 9%，这表明 CPU 大部分时间都在等待 GPU。主机负载均衡为 72%，这表明两个消息传递接口进程的工作量并不相同；在本例中，这指向了仅由其中一个消息传递接口进程执行的数据传输。

设备指标显示，通信效率为36%，表明数据传输造成了显著的效率损失。编排效率为86%，这说明主机仍有空间通过更高效地将工作卸载到 GPU 来进一步提升效率。



	指标	Eff.
Host	并行效率	0.06
	├ 消息传递接口 并行效率	0.72
	│ ├ 通信	1.00
	│ └ 负载均衡	0.72
	└ 设备卸载效率	0.09
Device	并行效率	0.31
	├ 负载均衡	1.00
	├ 通信效率	0.36
	└ 编排效率	0.86

图 9. 用例 6：工作均匀分配，主机-设备数据传输量大

用例 7：CPU-GPU 计算重叠的比较 。

在这个用例中，我们演示了如何通过新引入的 POP 指标展示 CPU 和 GPU 工作负载的重叠情况。

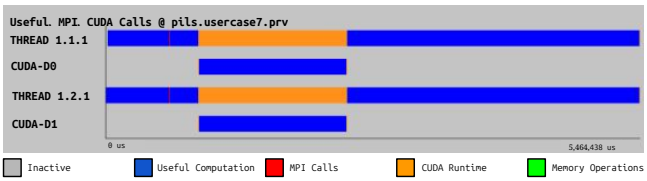
在图 10 的顶部和中部，我们可以看到代表此用例的两条跟踪记录。顶部的跟踪记录中，CPU 与 GPU 计算没有重叠；而中部的跟踪记录中，CPU 和 GPU 计算的执行存在重叠。在两条跟踪记录中，CPU 和 GPU 各自在其对应资源上的负载均衡情况都良好。

查看图 10 底部，可以看到两种执行方式对应的 TALP 指标。我们观察到，两种执行方式之间只有设备卸载效率和编排效率这两项指标有所不同，它们分别影响主机和设备上的并行效率。在存在重叠的执行方式中，设备卸载效率提高了 33%，达到近乎最优的水平。这是因为 CPU 会在短时间内异步地将工作卸载到 GPU，从而能够在剩余时间里进行有用的计算。另一方面，在存在重叠的执行方式中，编排效率接近 50%，因为 CPU 工作负载是 GPU 工作负载的两倍，导致 GPU 在一半的执行时间内处于非活动状态。

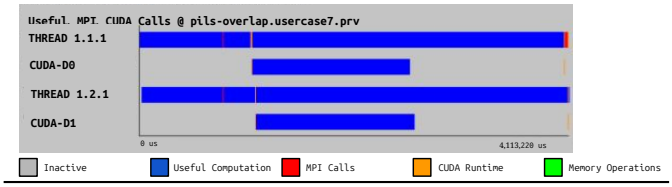
5.2 科学计算 HPC 应用

在本节中，我们使用三个高性能计算（HPC）科学应用程序来测试和验证所提出的指标及其实现。我们将在以下小节中简要介绍每个应用程序。

无重叠 CPU-GPU 计算



CPU-GPU 计算重叠



	指标	Seq	重叠
Host	并行效率	0.64	0.97
	├ MPI 并行效率	1.00	1.00
	│ ├ 通信 Eff.	1.00	1.00
	│ └ 负载均衡	1.00	1.00
	└ 设备卸载效率	0.64	0.97
Device	并行效率	0.33	0.49
	├ 负载均衡	1.00	1.00
	├ 通信效率	1.00	1.00
	└ 编排效率	0.33	0.49

图 10。用例 7：CPU-GPU 计算重叠的比较。

这些应用程序在 MareNostrum5 (MN5-Acc) 的加速分区上运行， [2], MN5-Acc 的每个节点均配备 $2\times$ Intel Sapphire Rapids 8460Y处理器和 $4\times$ NVIDIA H100 GPU。

我们使用 TALP 对所有这些项目进行性能分析，以识别性能瓶颈，并通过 Paraver 跟踪数据验证分析结果。

SOD2D 。

SOD2D 实现了用于描述三维可压缩流动和不可压缩流动的控制方程的数值求解方法。该方法基于谱元法（SEM），旨在用于尺度分辨模拟，包括大涡模拟（LES）和直接数值模拟（DNS）。该实现采用 Fortran 编写，并使用消息传递接口在不同节点和核心之间分配工作，同时使用 OpenACC 将工作卸载到 GPU。

表 1 展示了使用 TALP 运行 SOD2D、在 1 至 8 个节点上执行 1001 个时间步时获得的效率指标。可以看出，该应用程序针对 GPU 执行进行了优化，这体现在较高的设备并行效率上；而主机设备卸载效率则极低。这表明

表 1。SOD2D 在 1 至 8 个节点上的 TALP 输出

	指标	节点			
		1	2	4	8
Host	并行效率	0.06	0.05	0.04	0.04
	├─ MPI 并行效率	0.94	0.88	0.79	0.67
	│ ├─ 通信效率	0.95	0.89	0.80	0.68
	│ └─ 负载均衡	1.00	0.98	0.99	0.99
	└─ 设备卸载效率	0.06	0.05	0.06	0.06
Device	并行效率	0.87	0.81	0.72	0.59
	├─ 负载均衡	1.00	0.98	0.99	0.99
	├─ 通信效率	0.99	0.99	0.99	0.99
	└─ 编排效率	0.88	0.84	0.73	0.60

在最多 8 个节点上，GPU 始终保持活跃并从事有用工作，而 CPU 仅用于将工作卸载到 GPU，且大部分时间处于空闲状态。

在 8 个节点时，编排效率显著下降。这种情况可能是因为 CPU 花费更多时间进行 CPU 间通信，主机端通信效率的降低也证明了这一点，从而延迟了向 GPU 分配工作的过程。

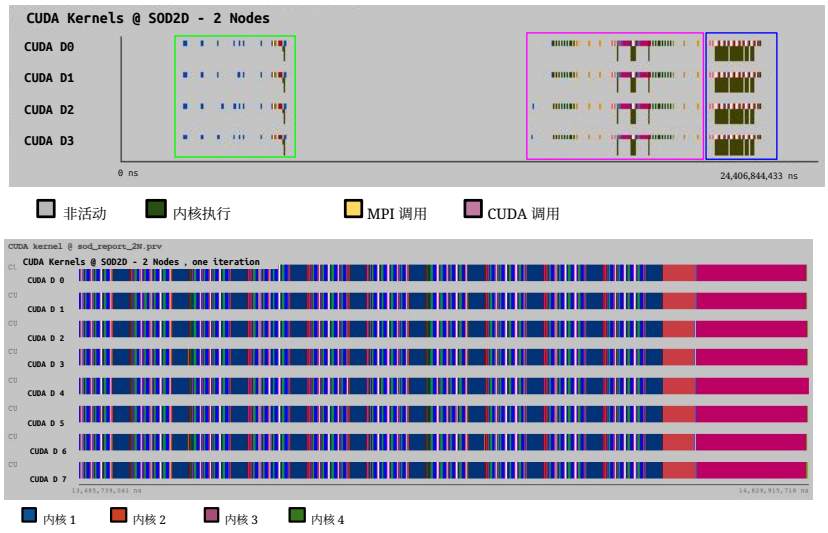


图 11. SOD2D 跟踪记录：CUDA 内核活动。上图：使用 2 个节点和 8 个 GPU 的完整跟踪记录（11 个时间步）。下图：同一执行过程中的单个时间步详细视图。

我们使用 NVIDA Nsight Systems 跟踪了 SOD2D 在 2 个和 4 个节点上执行 11 个步骤的过程，并使用 nsys2prv [4, 3] 将其转换为跟踪记录，以便在 Paraver 中进行可视化。由于使用与 TALP 性能分析相同的迭代次数会产生难以管理的数据量，因此我们减少了详细跟踪的步骤数。图 11 展示了整个执行过程中的 CUDA 内核，其中可以观察到三个明显的代码区域：(a) 初始化，位于开头并用绿色方框标出，此时数据从主机复制到设备；(b) 一个时间步（图中用紫色方框标出），在一段较长的间隔之后执行，此时 CPU 忙于初始化所有内容，而 GPU 没有任何活动；最后，(c) 11 个时间步，在上方图像中以蓝色矩形标出。

为了更好地理解 TALP 通过 1001 个步骤观察到的情况，图 11 底部的图像展示了使用 8 个设备执行 1 个步骤时的 CUDA 内核执行情况，每一行对应一个 CUDA 设备。在此跟踪记录中，我们可以看到，在时间步执行期间，GPU 几乎始终处于活动状态。

由于 SOD2D 的所有计算都在 GPU 上进行，因此设备卸载效率较低且稳定，同时编排效率较高，也是可以理解的。

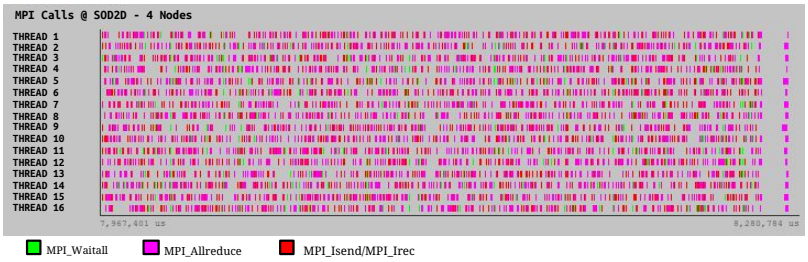


图 12. SOD2D 在 4 个节点、16 个进程上的 MPI 调用跟踪记录。

图 12 展示了在 4 个节点和 16 个 GPU 上执行的 MPI 调用。调用次数相当多，可以将其视为一种障碍，阻碍线程投入更多时间进行计算或将工作卸载到 GPU 上。因此，正如 TALP 输出所示，主机端 MPI 通信效率下降是意料之中的。

FALL3D 。

FALL3D 是一种用于大气被动输运和沉积的欧拉模型，基于平流–扩散–沉降（ADS）方程构建。该模型已扩展出多项新功能，包括支持集合预报以及模拟多种大气物种（例如火山

灰、气体、矿物粉尘和放射性核素）。FALL3D 使用 Fortran 实现，并通过消息传递接口和 CUDA 实现并行化。

表 2. FALL3D 使用 1 到 8 个节点的 TALP 输出

	指标	节点			
		1	2	4	8
Host	并行效率	0.26	0.16	0.10	0.07
	├ MPI 并行效率	0.44	0.27	0.16	0.11
	│ ├ 通信效率	0.85	0.83	0.82	0.89
	│ └ 负载均衡	0.52	0.32	0.20	0.12
	└ 设备卸载效率	0.59	0.61	0.63	0.64
Device	并行效率	0.14	0.08	0.04	0.03
	├ 负载均衡	0.98	0.97	0.96	0.96
	├ 通信效率	0.78	0.77	0.76	0.75
	└ 编排效率	0.19	0.11	0.06	0.04

表 2 展示了 FALL3D 在 1 至 8 个节点上运行时使用 TALP 得到的效率指标。TALP 输出揭示了 FALL3D 中的多个性能瓶颈。MPI 并行效率明显偏低，主要是由于负载不均衡。在主机级别，我们还观察到设备卸载效率较低，这意味着 GPU 执行内核时，CPU 并未用于执行有效工作。

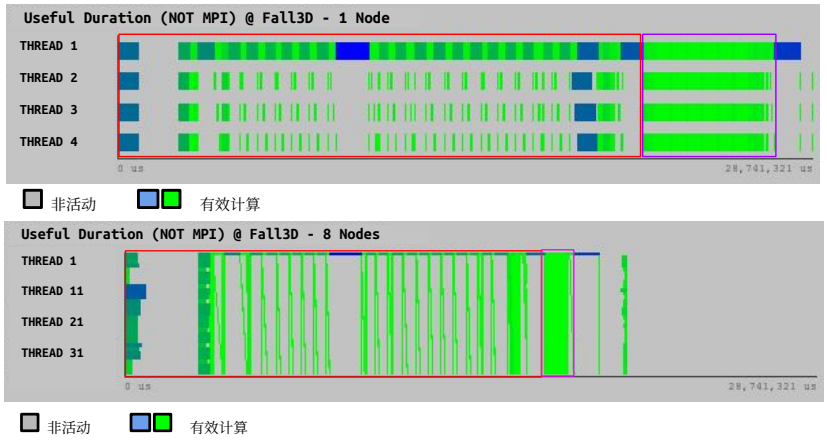


图 13。FALL3D 跟踪记录显示有用时长事件。上图：1 个节点，4 个 GPU。下图：8 个节点，32 个 GPU。

在设备级别，我们观察到编排效率较低，这意味着卸载到 GPU 的工作不足，无法让其持续执行有效工作。此外，随着应用程序扩展到更多节点，编排效率会下降，这可归因于 MPI 并行效率降低。由于 MPI 效率低下，主机无法为 GPU 提供工作。相反，设备卸载效率有所提高，这表明 CPU 花在 CUDA 运行时上的时间比例有所降低。

为了验证 TALP 输出，我们对 FALL3D 进行了跟踪记录，分别使用 1 个和 8 个节点，如图 13 和图 14 的两个窗口所示。两幅图中的窗口都显示相同的时间刻度。图 13 中，每一行对应一个 MPI 进程（为清晰起见，我们省略 GPU），两份跟踪记录中的红色方框标记初始化阶段，紫色方框标记应用程序的迭代区域。两个窗口中都可以看到主机上的负载不均衡：MPI 进程秩 0（标记为线程 1）在初始化阶段的大部分时间里，都用于向其他 MPI 进程分配工作负载。随着资源数量增加，FALL3D 中的负载均衡变得更加明显。

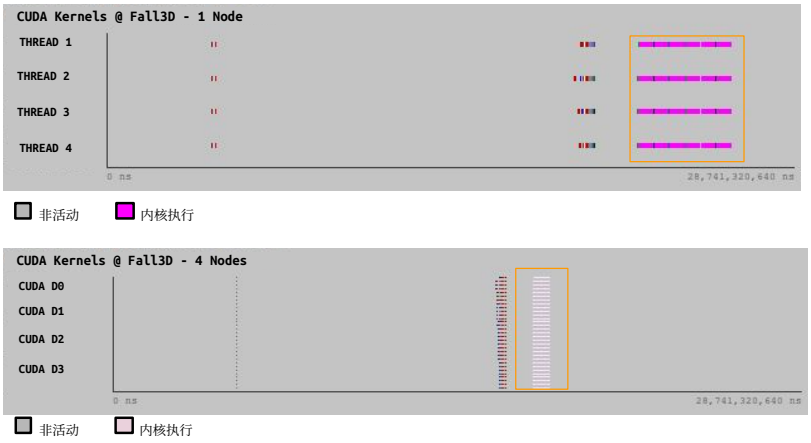


图 14。 FALL3D 跟踪记录显示 CUDA 内核执行情况。上：1 个节点，4 个 GPU。下：8 个节点，32 个 GPU。

从图 14 可以看出，顶部和底部两张图中黄色方框标出的 CUDA 内核执行，其设备数量从 1 个节点扩展到 8 个节点。因此，随着 GPU 数量增加，内核执行时间相应缩短，设备卸载效率的提升是合理的。随着节点数量增加、内核执行规模扩大，GPU 用于计算的时间减少，因此编排效率也随之降低。最后，由于初始化阶段耗时较长，且初始化期间 GPU 未被使用，在扩展应用程序时，GPU 处于非活动状态的时间占比会更高——

这一点可以从 TALP 输出中编排效率的较低数值看出。

XSHELLS 。

Xshells 是一款用于模拟球壳中旋转纳维–斯托克斯方程的高性能代码，也可选择与感应方程和温度方程耦合。XSHELLS 使用 C++ 编写，并使用消息传递接口实现分布式内存并行，使用 CUDA 实现 GPU 卸载。

表 3. XSHELLS 在 1 至 8 个节点上的 TALP 输出

	指标	节点			
		1	2	4	8
Host	并行效率	0.36	0.29	0.26	0.15
	├─ MPI 并行效率	0.90	0.64	0.51	0.25
	│ ├─ 通信效率	0.91	0.66	0.52	0.27
	│ └─ 负载均衡	0.98	0.97	0.98	0.93
	└─ 设备卸载效率	0.40	0.45	0.51	0.60
Device	并行效率	0.52	0.35	0.24	0.10
	├─ 负载均衡	1.00	1.00	1.00	1.00
	├─ 通信效率	0.98	0.97	0.96	0.94
	└─ 编排效率	0.54	0.36	0.25	0.10

表 3 展示了在 1 到 8 个节点上运行 XSHELLS 时的 TALP 输出。TALP 输出表明，CPU 和 GPU 上的计算负载均衡良好，因为主机端和设备端的负载均衡值都非常高。

另一方面，从 1 个节点扩展到 8 个节点时，主机通信效率急剧下降。在主机端，我们还观察到，使用一个节点时，设备卸载效率指标约为 40%，这意味着主机有 60% 的时间在执行有用工作，而有 40% 的时间在等待 GPU；我们还看到，随着节点数量增加，该指标也随之上升。这意味着，随着节点数量增加，CPU 完成的工作量也在增加。

在设备指标中，我们观察到良好的负载均衡和通信效率，而编排效率是唯一数值较低的指标。

单节点时，编排效率已经处于较低水平，仅为 54%；节点数增至 8 个时，更降至 10%。整体设备效率下降，主要是由于主机端指标较低，尤其是设备卸载效率和 MPI 通信效率。主机可能正忙于 MPI 通信和 CPU 有效工作，因此无法卸载足够的工作，导致 GPU 的利用率无法进一步提高。

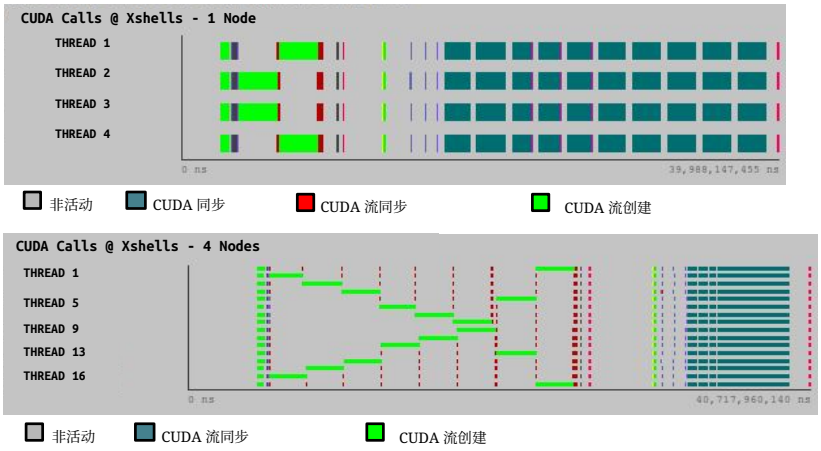


图 15. XSHELLS 显示 CUDA 调用的跟踪记录。上：1 个节点，4 个 GPU。下：4 个节点，16 个 GPU。

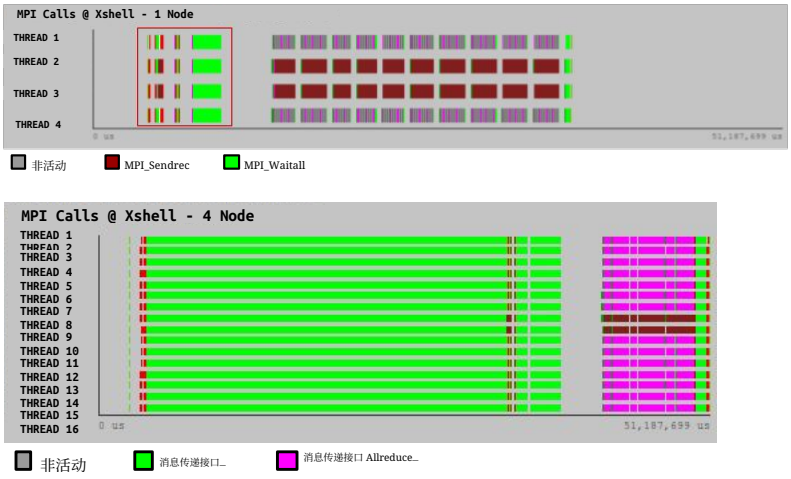


图 16. XSHELLS 显示 MPI 调用的跟踪记录。上：1 个节点，4 个 GPU。下：4 个节点，16 个 GPU。

图 15 展示了 XSHELLS 的两条跟踪记录，每条记录仅显示执行 CUDA 调用时的 MPI 进程：上方为 1 个节点，下方为 4 个节点。两个时间窗口的执行时长和刻度相同。

图 15 中单节点的顶部跟踪记录显示，主机仅在总时间线约一半的时间内向 GPU （青绿色）卸载工作。这与单节点测得的 54% 编排效率直观地相对应。

在对应 8 节点执行的底部跟踪记录中，我们看到初始化阶段的扩展性不佳，因此在 4 个节点上比在 1 个节点上耗时更长；而启动内核的迭代阶段扩展性良好，增加节点数后所需时间会减少。在 4 节点的跟踪记录中，迭代阶段耗时为四分之一，这与观测到的 25% 编排效率相对应。

在图16中，我们展示了两条跟踪记录，呈现主机执行的 MPI 调用。每一行对应一个 MPI 进程，颜色表示正在执行的 MPI 调用。上方的跟踪记录对应于在 1 个节点（4 个 MPI 进程）上运行的情况，下方的跟踪记录则对应于在 4 个节点（16 个 MPI 进程）上运行的情况。我们用红色矩形标出了初始化阶段。

我们观察到，初始化过程中存在一个 MPI 密集型阶段，且该阶段无法扩展；这可通过 MPI 通信效率的下降识别出来。

6 结论与未来工作

GPU 等加速器在高性能计算（HPC）中发挥着日益重要的作用。然而，在大规模高性能计算代码中高效使用加速器仍是一项艰巨挑战，因此性能测量与分析至关重要。由此，引入新的 GPU 专用指标势在必行。

在本研究中，我们提出了新指标，在现有 POP 效率框架的基础上扩展了指标范围，以分别捕捉异构架构中主机和设备的行为。通过区分效率指标，加速应用的性能分析变得更加简单有效。区分主机指标与设备指标，有助于更清晰地理解执行模式，并使分析能够贴合开发者针对目标硬件的意图。

所提出的指标不依赖特定厂商，适用于 AMD 和 NVIDIA GPU 等加速器平台，因此能够满足现代高性能计算集群和超级计算机的需求。具体而言，我们引入了一项主机端指标——设备卸载效率，以及三项设备端指标：负载均衡、通信效率和编排效率。

我们在 TALP 框架中实现了这些指标，使高性能计算用户能够以透明且轻量的方式测量加速平台的效率。目前的实现支持在 NVIDIA GPU 上使用 CUDA 或 OpenACC 运行的代码。未来，我们计划实现图的第二部分，即计算效率，并将实现扩展到 AMD GPU，以提升其在各种加速器上的适用性。

最后，我们使用合成基准测试和三个科学生产代码评估了所提出的指标及其在 TALP 中的集成。合成基准测试使我们能够对受控的工作负载和分布模式进行建模，而生产代码则展示了其在现实场景中的适用性。在这两种情况下，我们都将这些指标与详细的执行跟踪记录进行比较，验证了所提出的指标能够为性能行为提供准确且可操作的洞见。

致谢。 本工作获得了欧洲高性能计算联合事业（JU）以及西班牙、意大利、冰岛、德国、挪威、法国、芬兰和克罗地亚的资助，资助协议编号为 101093038，项目名称为 ChEESE-2P ； 还获得了 MCIN/AEI/10.13039/501100011033 资助的项目 PCI2022-134980-2 的支持，并由欧盟 NextGenerationEU/PRTR 提供资助。

本项目获得欧洲高性能计算联合事业计划（JU）以及西班牙、德国、法国、葡萄牙和捷克共和国的资助，资助协议编号为 101143931；此外还获得 POP3 项目 PCI2024-153419 的资助，该项目由 MICIU/AEI 资助。

参考文献

1. AMD：AMD ROCm 性能分析器（rocprofiler）（2025），
https://rocm.docs.amd.com/projects/rocprofiler/en/latest/，访问日期：2025 12 2 09- - . Banchelli, F., Garcia-Gasulla, M., Mantovani, F., Vinyals, J., Pocurull, J., Vi- cente, D., Eguzkitza, B., Galeazzo, F.C.C., Acosta, M.C., Girona, S.：介绍 MareNostrum5：一套面向多种科学工作负载的欧洲前百亿亿次级能效系统。《未来一代计算机系统》p. 108125（2025）。https://doi.org/https://doi.org/10.1016/j.future.2025.108125，
https://www.sciencedirect.com/science/article/pii/S0167739 25004194 3 2 X . Clascà, M.：nsys prv：将 NVIDIA Nsight Systems 跟踪数据转换为 Paraver 跟踪数据，
https://pypi.org/project/nsys2prv/ 4. Clascà, M., Labarta, J., Garcia-Gasulla, M.：nsys2prv：使用 Paraver 对大规模 GPU 执行跟踪数据进行详细的定量分析。载于：2025 IEEE 国际集群计算会议（CLUSTER）。IEEE（2025）5. Corporation, I.：Intel 跟踪分析器与收集器（ITAC）（2025），
h
ttps://
www.intel.com/content/www/us/en/developer/tools/oneapi/trace-analyzer.htm，访问日期：2025 09 6 09- - . Corporation, I.：摩尔定律（2025），
https://newsroom.intel.com/press-kit/moores-law，访问日期：2025 09 7 09- - . Corporation, N.：CUDA 性能分析工具接口（CUPTI）文档（2025），
https://docs.nvidia.com/cupti/，访问日期：2025 12 8 09- - . Corporation, N.：NVIDIA Nsight Compute（2025），
https://docs.nvidia.com/nsight-compute/ReleaseNotes/index.html，访问日期：2025年9月9日 9. Corporation, N.：NVIDIA Nsight Systems（2025），
https://docs.nvidia.com/nsight-systems/，访问日期：2025 09 10 09- - . D’Amico, M., Garcia-Gasulla, M., López, V., Jokanovic, A., Sirvent, R., Corbalan, J.：DROM：为资源管理器实现高效、轻松的可伸缩性。载于：第47届国际并行处理会议论文集（伴随会议）。第41页。ACM（2018）11. DLB 库在线资源：（2018年3月），https://pm.bsc.es/dlb 12. Garcia, M., Labarta, J., Corbalan, J.：利用 LeWI 提示改进混合应用的自动负载均衡。《并行与分布式计算杂志》**74**(9), 2781–2794（2014）13. Garcia-Gasulla, M., Houzeaux, G., Ferrer, R., Artigues, A., López, V., Labarta, J., Vázquez, M.：Mpi+ x：有限元组装的基于任务的并行化与动态负载均衡。《国际计算流体动力学杂志》**33**(3), 115–136（2019）

14. Geimer, M., Wolf, F., Wylie, B.J., Ábrahám, E., Becker, D., Mohr, B.：Scalasca 性能工具集架构。《并发与计算：实践与经验》**22**(6), 702–719（2010）15 09 16 2025 2025 09 1 . HPCToolkit：HPCToolKit（），https://hpctoolkit.org/，访问日期：- - . Kessler, S.L.G.P.B., McKusick, M.K.：gprof：调用图执行分析器。载于：编译器构造研讨会论文集，第–页。出版社，纽约„。引文见第.. Greco, Gianluigi。Citeseer（1982）17. Knüpfer, A., Rössel, C., an Mey, D., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A., 等：Score-P：为 Periscope、Scalasca、TAU 和 Vampir 提供的联合性能测量运行时基础设施。载于：《高性能计算工具 2011》，第79–91页。施普林格（2012）18. Lopez, V., Ramirez Miranda, G., Garcia-Gasulla, M.：TALP：揭示大规模执行并行效率的轻量级工具。载于：2021年性能工程、建模、分析与可视化策略会议论文集。第3–10页。PERMAVOST '21，美国计算机协会，纽约，NY，美国（2021）。https://doi.org/10.1145/3452412.3462753, https://doi.org/10 3462753 19 1145 3452412 . / . . Pillet, V., Labarta, J., Cortes, T., Girona, S.：Paraver：用于可视化和分析并行代码的工具。载于：WoTUG-18：Transputer 与 occam 开发论文集。第44卷，第17–31页。Citeseer（1995）20. PRACE：欧洲先进计算伙伴关系（PRACE）（2025），https://prace-ri.eu/，访问日期：2025 09 21 09- - . Profiling, for Performance Analysis of Large Scale Applications, T.T.：D7.4：未来百亿亿次级系统工具与技术评估最终报告。PRACE（2015），https://prace-ri.eu/wp-content/uploads/4IP-D7.4.pdf 22. Project, T.：TOP500：全球最快超级计算机排行榜（2025），https://top500 09 23 2025 09 .org/，访问日期：- - . Servat, H., Llort, G., Huck, K., Giménez, J., Labarta, J.：高效性能优化框架。《并行计算》**39**(8), 336–353（2013）24. Shende, S.S., Malony, A.D.：TAU 并行性能系统。《国际高性能计算应用杂志》**20**(2), 287–311（2006）25. Wagner, M., Mohr, S., Gi ménez, J., Labarta, J.：性能分析的结构化方法。载于：高性能计算并行工具国际研讨会。第1–15页。施普林格（2017）26. Wylie, B.J., Giménez, J., Feld, C., Geimer, M., Llort, G., Mendez, S., Mercadal, E., Visser, A., García-Gasulla, M.：使用 Scalasca/Score-P 和 Paraver/Extrae 工具集开展联合并行应用性能分析与工具培训的 $15+$ 年。《未来一代计算机系统》**162**, 107472（）。https://doi.org/https://doi.org/ . /j.future. . . https://www.sciencedirect.com/science/article/pii/S0167739X24004187