# 通过自动格式选择和机器学习优化稀疏线性代数

Christodoulos Stylianou*EPCC*，爱丁堡大学爱丁堡，英国 c.stylianou@ed.ac.uk

Mich`ele Weiland*EPCC*，爱丁堡大学爱丁堡，英国 m.weiland@epcc.ed.ac.uk

arXiv:2303.05098v1 [cs.LG] 9 Mar 2023

摘要—**稀疏矩阵是科学模拟不可或缺的一部分。随着硬件不断发展，人们提出了新的稀疏矩阵存储格式，以利用针对新硬件的特定优化。在异构计算时代，为了使应用程序在各种可用硬件上都保持最优性能，用户往往需要使用多种格式，这会增加开发时间和维护开销。解决这一问题的一种潜在方案是采用由机器学习（ML）驱动的轻量级自动调优器，为用户从一组可用格式中选择最优格式，使其与稀疏模式、目标硬件以及待执行的操作相匹配。**

**本文介绍了*Morpheus-Oracle*，这是一个库，提供轻量级机器学习自动调优器，能够针对多个后端准确预测最优格式，并面向主流高性能计算架构，旨在免除终端用户进行格式选择。基于 2000 多个真实矩阵，我们在各个可用系统上分别实现了** $92.63\%$ **的平均分类准确率和**$80.22\%$ **的平衡准确率。采用该自动调优器后，CPU 上的平均加速比为** $1.1\times$ **，NVIDIA 和 AMD GPU 上为** $1.5\times$ **至** $8\times$ **，最大加速比分别可达** $7\times$ **和** $1000\times$ **。**

索引词—**稀疏矩阵存储格式、机器学习、自动格式选择**

## I. 引言

自诞生以来，稀疏矩阵便已成为计算科学与工程领域诸多科学模拟的关键组成部分 [1]–[3]。多年来，人们开发了 70 多种稀疏矩阵表示（稀疏矩阵存储格式），分别针对不同类型的矩阵，和／或适应硬件向多核与众核处理器及加速器演进的需求 [4]。文献表明，不同格式具有不同的性能特征，没有任何一种格式能在所有不同类型的矩阵和硬件上都达到最佳性能 [5]–[10]。

利用每种格式的特性，并调整稀疏矩阵底层的数据结构（而不是采用单一的存储格式），使其更好地匹配操作和目标硬件，可以获得优化机会，从而有望提升操作性能。由于矩阵通常在编译时未知，这种利用只能在运行时进行。此前的工作 [11]–[14] 提供了抽象和机制，有效支持在所支持的不同格式之间进行运行时切换。

尽管启用某种动态切换机制以便在运行时更改格式至关重要，但将选择最优格式的过程自动化同样重要。为了便于针对给定的矩阵、目标体系结构和操作选择最优格式，研究人员开发了自动调优器（例如 [9], [13], [15], [16]），重点选择稀疏矩阵-向量乘法（SpMV）的最优格式；SpMV 通常主导线性系统求解过程的运行时开销。这些研究成果证明了通过自动格式选择优化迭代求解器性能对单节点多核和众核系统的影响。

在这项工作中，我们开发了一种自动调优系统，能够利用机器学习，高效调优稀疏矩阵-向量乘法内核在不同厂商提供的各种先进系统上的性能，同时不依赖特定体系结构。该自动调优系统提供多种机器学习算法，各自具有不同的开销和选择准确率。我们介绍 *Morpheus-Oracle(Oracle)* [17],，这是一个支持自动格式选择的库，它实现了不同的自动调优器以及特征提取例程。第六节详细介绍*Oracle*。

总之 , 我们的贡献如下:

- 我们展示了在多种系统和目标硬件（包括多核与众核处理器及加速器）上，对 2000 多个矩阵而言，稀疏矩阵-向量乘法运算在 6 种不同矩阵存储格式中的最优格式分布。
- 对于同一组矩阵和系统，我们量化了与常用通用格式压缩稀疏行（CSR）相比，选择最优格式所带来的性能提升。CPU后端的平均运行时加速比最高可达 $\sim 2\times$，GPU后端最高可达 $\sim 10\times$ 。
- 我们提供了一个可复用的模型生成系统，其中包含来自实际应用的 2000 多个稀疏矩阵，用户可利用这些矩阵训练自己的模型。此外，我们还训练并调优了两种不同的机器学习算法，比较它们正确选择最优格式的能力，并量化引入自动调优机制所带来的开销相对于稀疏矩阵-向量乘法迭代运行时开销的大小。
- 我们介绍一个简单、灵活且可扩展的自动

调优库，以补充*Morpheus*——这个用于动态格式切换的库。该库提供调优器，可针对多种架构可靠地选出最优格式。

## II. 背景与动机

### A. 动机

每当新架构问世时，研究人员都会提出新的存储格式，以利用新硬件特有的优化。在异构计算时代，硬件日趋多样化，因此应用程序往往需要在不同类型的硬件上使用多种格式，才能保持最优性能。尽管已经提出了新格式（ [5], [6], [10] ，旨在缓解性能可移植性问题，但目前仍没有一种格式能在不同的稀疏模式、硬件架构和运算中都达到最优表现。正如我们在第VII-C节所展示的，即使某种格式总体表现良好，在某些情况下也可能性能严重下降，导致运行时性能不佳。这在图形处理器上尤为明显：错误的格式可能导致设备利用率不足，或因非合并内存访问而产生过多的内存请求。因此，为使应用程序达到最优性能，更好的解决方案是在运行时从一组候选格式中选出最优格式。

有经验的用户或许对自己经常使用的某类矩阵或某种类型矩阵的最优格式选择有所判断，但这类决策并非易事，因为它取决于多种因素。此外，先运行各种可用选项再选择最优格式，可能会带来显著开销。机器学习（ML）和人工智能（AI）已成功应用于各种优化任务，从代码优化到模型选择 [18],，其中包括选择最优稀疏矩阵存储格式 [13], [16]。采用机器学习模型有望为自动格式选择问题提供准确且开销低的解决方案，无需用户手动输入格式选择，从而使应用程序针对不同类型的硬件和稀疏模式，在执行任何感兴趣的操作时都能保持最优。

### B. 稀疏矩阵存储格式

多年来，人们开发了大量稀疏矩阵存储格式；每当出现新的架构时，都会有新的格式问世 [4]。每种格式的存储需求和计算特性各不相同，而且修改和操作矩阵条目时所使用的接口也各不相同 [19]。这些特性使得要事先确定某种格式在给定操作和目标硬件条件下能否为特定矩阵带来最佳性能并非易事，因为有多个因素会影响这一决策。本文考察了六种不同的存储格式：两种通用格式、两种专用格式和两种混合格式。

最基础且广为人知的格式是坐标格式（COO）和压缩稀疏行格式（CSR）。这两种格式都被视为通用格式，适用于各种稀疏模式和目标架构，而 CSR 通常是首选格式。COO 由三个数组构成，每个非零元素都与其行索引和列索引一同存储，且不保证元素的排列顺序。类似地，CSR 也显式存储非零值和列索引，但会压缩行索引，并改为存储一个指针数组来标记每行的边界，从而有效地使各行按自然顺序排列。

特定用途格式则旨在利用特定类别矩阵的特性，通常经过专门设计，可在特定体系结构（如图形处理器）上实现最佳性能。例如，对角线格式（DIA）旨在表示规则的稀疏模式，适用于类向量处理器。DIA 将非零元素存储在一个二维数组中，其中每一列存放矩阵一条对角线上的系数。此外，它还使用一个整数偏移数组来记录每条对角线的起始位置。另一个值得关注的例子是 ELLPACK（ELL）格式，该格式假设每行至多有 $K$ 个非零元素。因此，它使用一个二维数组表示矩阵的非零元素，另一个二维数组存储这些元素的列索引。因此，DIA 格式适用于表示对角线方向占主导地位的结构，例如带状矩阵；而 ELL 格式则适用于结构化或半结构化矩阵（即每行的非零元素数量相近）。需要注意的是，如果对角线数量或每行的非零元素数量非常大，这两种格式都可能因过度填充而造成问题。

混合格式通常旨在结合两种格式的优势，以消除其中一些缺点，例如前文提到的过度填充。首先介绍的混合格式是混合格式（HYB），它由 ELL 格式和坐标格式组成。HYB 使用参数 $K_H$ ，表示每行存储在 ELL部分中的非零元素数量。对于非零元素数量大于 $K_H$ 的行，多出的非零元素则存储在坐标格式部分。其次介绍的混合格式是 DIA/CSR混合格式（HDC）。它使用参数 $N_D$ ，表示对角线上非零元素的数量阈值；超过该阈值的对角线被视为“真实”对角线。矩阵中的真实对角线随后以对角线格式存储，其余部分则以压缩稀疏行格式存储。

### C. Morpheus

Morpheus [11] 是一个 C++ 库，通过 *DynamicMatrix* 这一单一动态“抽象”格式，支持在运行时切换稀疏矩阵存储格式。*Morpheus* 提供了一种透明机制，可高效切换到 *DynamicMatrix* 支持的不同格式，目前支持第 II-B 节中提到的六种格式。此外，*Morpheus* 为四种不同的后端提供了稀疏矩阵-向量乘法等算法，以支持大多数关键的高性能计算平台：1）串行（顺序），2）OpenMP（多线程），3）CUDA（NVIDIA GPU）和 4）HIP（AMD GPU）。

多线程），3）CUDA（NVIDIA GPU）和 4）HIP（AMD GPU）。此外，*Morpheus* 提供数据管理例程，并支持在各后端所支持的不同内存空间之间传输数据；同时采用主机-设备模型，以支持同构和异构平台（即仅配备 CPU 或配备 CPU+GPU 硬件的平台），并保持应用程序源代码不变。

通过将不同格式抽象为单一的动态格式，封装内部实现细节，并为跨后端的算法提供统一接口，用户便可使用简单直观的接口，无需处理其中的复杂性。在这项工作中，我们基于*Morpheus*的运行时切换机制，实现自动格式选择，从而彻底消除用户手动选择格式的需要。

## III. 自动调优流程高级概述

这项工作的重点是开发一种自动调优器，用于为 *DynamicMatrix* 选择最优格式。*Morpheus* 提供的 DynamicMatrix 可根据给定的矩阵、操作和目标硬件切换格式。实现这一任务最直接的方法是使用先运行调优器：对每种受支持的格式运行目标操作，测量所需指标，并选择性能最佳的格式作为最优格式。这种方法会牺牲整体运行时性能（尽管它能提供最准确的预测），因为它需要在不同格式之间进行多次耗时的转换，而且格式越多，开销越大。另一种更好的方法是使用机器学习模型来寻找最优格式，从而降低预测成本。图1展示了我们提出的自动调优流程的高级概述。

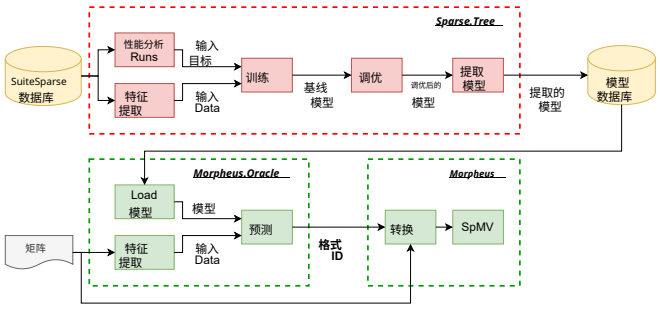



图 1：自动调优流程的高层概览。红色和绿色方框分别表示离线和在线运算。

自动调优流程分为离线阶段（红色）和在线阶段（绿色）。对于每种新的体系结构以及我们希望进行预测的运算，都必须执行一次离线阶段；该阶段的结果随后可在在线阶段重复使用。

### *A.* 离线阶段

流水线的第一阶段是实际的模型生成：我们针对特定架构和运算训练、调优并提取机器学习模型，将其保存到文件中，供自动调优器稍后使用。我们使用了约 2200 个实值方阵，它们的尺寸各异，具有不同的稀疏模式，涵盖不同的应用领域，取自 *SuiteSparseCollec-tion* [20]。

对于数据集中的每个矩阵，我们首先针对感兴趣的运算和体系结构进行性能分析运行，并测量运行时，以确定最优格式，即该运算运行时最短的格式，然后导出其格式 ID，供后续阶段使用。同时，我们对这些矩阵执行特征提取流程，具体内容详见第四节，以生成训练过程中使用的输入。输入特征（输入数据）和格式 ID（输入目标）都会用于训练和调优机器学习模型；模型的描述见第五节。获得调优后的模型后，我们将其导出到文件中并保存，以供后续使用。

为了简化用户的训练过程，我们将该过程封装在一个名为 *Python* 框架 *Sparse.Tree*[1] 中，该框架底层使用*scikit-learn* [21] 。用户可以使用 *Sparse.Tree* 为新系统生成模型，也可以使用模型数据库 中预先训练好的模型，这些模型适用于本研究使用的 x86 和 ARM CPU，或 NVIDIA 和 AMD GPU。

### *B.* 在线阶段

为了能够在应用程序中自动选择*Morpheus* 应使用的最佳格式，我们必须能够高效地在线做出决策，也就是说，在应用程序运行时做出决策。在流水线的第二阶段，我们实现了*Oracle*，这是一个与体系结构无关的 C++ 自动调优器，它使用*Morpheus* 提供的*DynamicMatrix* ，并加载运行时指定文件中的机器学习模型来做出决策。请注意，“与体系结构无关”是指调优器无需了解其所针对的目标硬件，因为这些信息已包含在运行时加载的模型中。

为了让自动调优器能够使用模型，需要以与第一阶段（训练阶段）相同的方式提取输入矩阵的特征。随后，通过遍历模型，*Oracle*会返回最优格式 ID，*Morpheus*据此切换格式并执行所需操作。*Oracle* 的详细介绍见第六节。

## IV. 特征提取

在本研究中，特征提取是指将原始稀疏矩阵转换为一组数值“特征”的过程，这些特征可由模型处理，同时保留原始矩阵稀疏模式的信息。针对所关注问题的相关特征有助于模型做出有依据的决策；然而，计算这些特征所需的开销与基于这些特征所做决策的准确率之间存在权衡。换言之，为模型提供大量与问题相关的信息

[1]网址：https://github.com/morpheus-org/sparse.tree

| 参数 | 描述 | 公式 |
|---|---|---|
| $M$ | 行数 | - |
| $N$ | 列数 | - |
| $NNZ$ | 非零元素数 | - |
| $\overline{NNZ}$ | 平均非零元素数<br>每行 | $\overline{NNZ} = \frac{NNZ}{M}$ |
| $\rho$ | 密度 | $\rho = \frac{NNZ}{M*N}$ |
| $max\,(NNZ)$ | 最大非零元<br>每行 | $max\,(NNZ) = max_{i=1}^{M} NNZ_i$ |
| $min(NNZ)$ | 最小非零元<br>每行 | $min(NNZ) = min_{i=1}^{M} NNZ_i$ |
| $\sigma_{NNZ}$ | 非零元标准差<br>每行 | $\sigma_{NNZ} = \frac{\sum_{i=1}^{M} \lvert NNZ_i - \overline{NNZ} \rvert^2}{M}$ |
| $N_D$ | 对角线数量 | - |
| $N_{TD}$ | # of<br>真实对角线 | - |

表 I：用于训练模型的特征参数，以及在适用情况下用于计算各特征参数的相应公式。

有助于提升学习效果，但代价是必须计算这些信息。本研究选取了一组包含 10 个特征的集合（见第 IV-A 节），如表 I 所示，这些特征既描述了稀疏矩阵的基本结构，也描述了矩阵各行和各对角线上的非零元素分布。

### A. 特征选择

前三个特征——行数（$M$）、列数（$N$）和非零元素数（$NNZ$）——旨在概括矩阵的规模，而且易于获取，因为它们由 *DynamicMatrix* 提供。根据 Monakov 等人 [8], 的研究，坐标格式适用于具有许多空行的极稀疏矩阵，因此我们还将每行非零元素的平均数（$NNZ$）和矩阵的密度（$\rho$）加入特征集合。

另一方面，DIA 和 ELL 等专用格式的性能会受到非零元素分布的显著影响，因为不适合各自格式的分布会导致矩阵中零元素填充过多，从而损害整体性能。ELL 根据每行非零元素的最大数量分配内存，因此每行非零元素分布极不均匀的矩阵并不适合这种格式。通过测量每行非零元素的最大值和最小值（$max(NNZ)$ $min(NNZ)$），以及每行非零元素的标准差（$\sigma_{NNZ}$），可以捕捉这一信息。类似地，DIA 格式根据对角线数量分配内存，因此，对于对角线数量很多、但每条对角线仅包含少量元素的矩阵，同样会导致填充过多。因此，这里遍历矩阵的对角线，统计至少包含 1 个非零元素的对角线数量（$N_D$），以及非零元素数量超过阈值的主对角线数量（$N_{TD}$）。由于 HYB 和 HDC 是混合格式，现有特征仍具有代表性。关于如何针对不同格式提取特征的更多详情见第六节 C 部分。

## V. 机器学习模型

我们的目标是训练一个模型，用于预测给定稀疏输入矩阵的最优存储格式。这类问题属于多类分类问题。在训练期间，对于集合中的每个输入矩阵，我们都会提取一组包含 10 个特征的数据特征，以及一个目标属性；该属性对应于通过性能分析运行得到的最优格式索引。模型的目标是尝试确定输入特征与最优格式 ID 之间的映射关系，如公式 1 所示：

$$f(\vec{x_1}, \vec{x_2}, ..., \vec{x_n}) \rightarrow y_n(COO, CSR, ..., HDC) \quad (1)$$

其中，$\vec{x_i}$ 表示训练集中的 $i^{th}$ 稀疏矩阵的特征向量，$y_n$ 表示目标向量，其中每个条目都包含六种可用格式之一的索引。

为了训练模型预测目标值，我们采用决策树机器学习算法，该算法能够有效地从数据特征中推导并学习简单的决策规则。选择该算法有两个原因：首先，这种方法简单易懂，便于解释；其次，在训练模型或使用模型进行预测之前，几乎不需要进行数据准备。然而，决策树可能会通过生成过于复杂的树而发生过拟合，导致其无法很好地泛化数据，或在数据略有变化时变得不稳定。为避免这些问题并生成最优模型体系结构，我们在定义的超参数空间内进行穷举网格搜索，以寻找最优超参数值。此外，为了提高模型的稳健性，我们构建了一个由决策树组成的集成，称为“随机森林”；它通过将多个决策树分类器拟合到数据集的不同子样本上来实现这一点。虽然随机森林分类器能够提高模型的预测准确率并控制过拟合，但代价是预测时间更长，因为需要遍历多棵树，并将每棵树的决策合并为一个最终结果。

在这项工作中，我们特别关注训练一个模型，以预测 *Morpheus* 支持的每种后端在执行稀疏矩阵 - 向量乘法运算时应使用的最优格式。不过，此处采用的技术和算法也可迁移到其他稀疏运算。我们使用决策树和随机森林方法，在 Python 中针对多种 x86 和 ARM CPU 以及 NVIDIA 和 AMD GPU 训练并调优模型，然后将模型导出到文件中，以便在自动调优阶段使用。

## VI. MORPHEUS ORACLE——一种用于自动格式选择的自动调优器

为了系统地进行格式选择，我们开发了*Oracle*，这是一个仅包含头文件的 C++ 库，用于自动格式选择。开发它是为了补充*Morpheus* 的动态切换能力，并通过调优

性能，自动为给定操作和目标架构选择要使用的最优格式。*Oracle* 遵循与*Morpheus*类似的设计理念，即将容器与算法分离。此处的容器代表该软件包支持的不同调优器，负责封装每种调优器实现的具体细节，并仅向用户提供用于配置和运行调优器的接口。

### *A.* 调优器

目前，*Oracle* 支持三种调优器：1）*Run-first*、2）*DecisionTreeTuner* 和 3）*RandomForestTuner*。每种调优器都负责处理选择最优格式时涉及的复杂问题。例如，*Run-first* 调优器会记录每种格式在给定操作中执行*N*次迭代所需的迭代时间，并运用统计学方法确定哪种格式表现最佳。另一方面，*DecisionTreeTuner* 和 *RandomForestTuner* 需要从文件加载以树结构表示的机器学习模型，并遍历该模型以确定最优格式。*DecisionTreeTuner*只遍历一棵树，而*RandomForestTuner* 会遍历集成中的多棵树，然后采用投票方案来确定最优格式。在此情况下，使用多数投票方案，选择得票最多的格式作为最优格式。

这三种调优器的性能，直接体现为运行时开销与预测准确率之间的权衡。*Run-first* 调优器能够提供最准确的预测，但代价是需要在各个受支持的格式之间进行耗时的转换；*DecisionTreeTuner* 则能提供非常快速但准确度较低的预测。*RandomForestTuner* 通过使用多棵树提高 *DecisionTreeTuner* 的预测准确率，而预测过程的运行时间则与所用树的数量成正比。支持多种类型的调优器，用户便可根据自身需求进行选择。

### *B.* 运算

Morpheus 提供的每种稀疏算法都可以调优，以使用最优格式，从而有效优化算法。稀疏矩阵-向量乘法算法的接口通过 TuneMultiply 运算定义，任何其他运算也遵循相同的原则。通过编译时内省，我们可以为各种受支持的运算维护一个统一的高级接口，并针对每个调优器专门化实现。请注意，Oracle 包通过请求以下内容与 Morpheus 交互：1）要调优的 DynamicMatrix；2）运行该运算的执行空间；随后返回 DynamicMatrix 应切换到的最佳格式的 ID。

调优操作的输入需要包含*DynamicMa-trix* 和调优器，并将所需的执行空间作为模板参数。调优操作完成后，可以向调优器查询最优格式。如果将*Run-first* 调优器传递给 *TuneMultiply*，调优操作将针对每种格式执行*N*次稀疏矩阵-向量乘法，并由调优器跟踪运行时间。不过，当机器学习调优器被使用时，它们会评估模型并确定最优格式。为此，机器学习调优器必须实时进行特征提取。

### *C.* 特征提取

处理稀疏矩阵时，最大的挑战在于每种格式在内存中的表示各不相同，而且格式之间可能差异很大。进行特征提取时，必须遍历矩阵以收集所需信息。若要实时提取第四节中定义的特征，通常需要多次遍历矩阵的行和对角线；对于行数较多的矩阵，这可能代价高昂。在训练过程中进行离线特征提取时，多次遍历矩阵或许成本较高，但并非无法接受，因为这一过程通常只执行一次。然而，在自动调优过程等在线场景中，这可能会抵消优化过程带来的任何收益。

为支持在线提取格式，*Morpheus* 已扩展为能够按每种格式以及在不同受支持的后端上提供矩阵统计信息，从而无需在不同空间之间传输任何数据。可以同时计算多项统计信息，减少遍历矩阵的次数，并降低此过程的运行时开销。随后，*Oracle*可以通过检查*DynamicMatrix* 当前使用的格式来在线提取特征，并通过查询调优器的模型来预测最优格式。

## VII. 结果与评估

本节将测量并评估*Oracle* 中可用的机器学习自动调优器的性能。首先，通过性能分析运行确定不同系统中每个矩阵所对应的最优格式分布，并评估切换到最优格式而非使用单一默认格式（压缩稀疏行格式）所带来的性能收益。接下来，通过比较基线模型和调优后的模型，量化机器学习模型的预测性能。然后，根据执行一次稀疏矩阵-向量乘法运算所需的时间，量化预测过程的开销。最后，评估 *Morpheus* 与*Oracle* 自动调优器协同运行时的性能。

### *A.* 实验设置

所有实验均在 ARCHER2 [22],Cirrus [23] 和 Isambard [24] 超级计算机上进行；其计算节点架构见表 II。实验涵盖 *Morpheus* 支持的全部四种后端，覆盖了具有代表性的主流硬件架构，例如 x86（英特尔和 AMD）与 ARM CPU，以及 NVIDIA 和 AMD GPU。

实验所用数据集由 SuiteSparse 库中的约 2200 个真实方阵组成，训练集与测试集按 80%-20% 划分。

| 系统 | 子系统队列 | | CPU | GPU |
|---|---|---|---|---|
| ISAMBARD | A64FX | A64FX | 1X 富士通 A64FX (48 个核心) | - |
| | P3 | INSTINCT | 1X AMD EPYC 7543P (32 个核心) | 4X AMD INSTINCT MI100 |
| | | AMPERE | | 4X 英伟达安培架构 A100 40GB |
| | XCI | ARM | 1X Marvell ThunderX2 ARM（32 个核心） | - |
| Cirrus | | 标准 | 2 颗英特尔至强 E5-2695（18 核） | - |
| | | GPU | 2 颗英特尔至强 Gold 6248（18 核） | 4 颗 NVIDIA Volta V100 16GB |
| ARCHER2 | | 标准 | 2 台 AMD EPYC 7742 (64 核) | - |

表 II：实验所用系统的节点配置。

### *B.* 格式分布

由于没有任何一种格式能在不同稀疏模式、目标硬件和运算中始终表现最佳，为了了解数据集中不同矩阵的最优格式，我们在可用平台上对稀疏矩阵-向量乘法运算进行了性能分析运行。对于数据集中的每个矩阵、每种受支持的格式以及每个可用平台，我们记录了重复执行 1000 次稀疏矩阵-向量乘法的运行时，并将运行时最短的格式设为该矩阵在该平台上的最优格式。图 2 展示了 SuiteSparse 数据集中所有矩阵在各系统和格式下的最优格式分布。

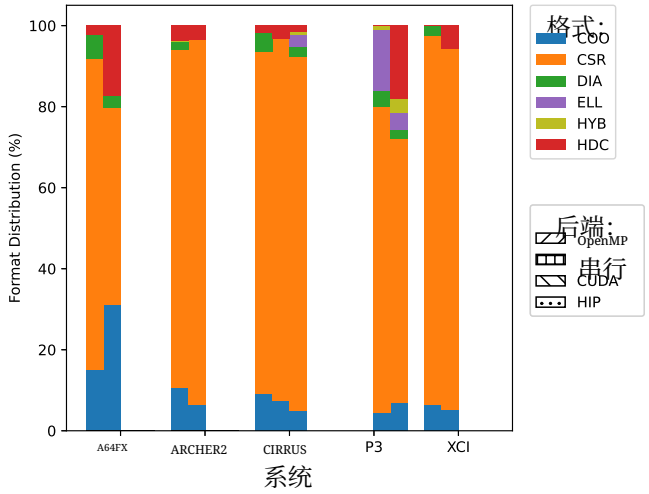



图 2：使用 SuiteSparse 数据集执行 1000 次稀疏矩阵-向量乘法时，最优格式的分布。每个矩阵的最优格式定义为运行时最短的格式。

从图 2 可以清楚地看出，在该集合中占比最大的矩阵在不同系统和后端上的最优格式都是压缩稀疏行格式，这验证了它作为最常用存储格式的地位。不过，我们确实观察到，即使在相同硬件上，格式分布也可能发生相当大的变化。在 A64FX 系统上，对于串行后端，使用 HDC 格式或坐标格式时性能更好的矩阵更多；而对于 OpenMP 后端，这些矩阵则采用压缩稀疏行格式。不过，在 Cirrus 和 Archer2 系统上观察到相反的情况：在串行后端上使用压缩稀疏行格式时性能最佳的矩阵，在 OpenMP 后端上实际上使用坐标格式或对角线格式时性能最佳。此外，与 CPU系统相比，图形处理器系统上的格式分布更加多样，所选最优格式几乎涵盖了所有可用格式类别。

这里的主要结论是，实验中每个系统/后端的格式分布都不平衡，其中压缩稀疏行格式占据明显多数。因此，第五节中的分类问题属于类别不平衡分类问题或稀有事件预测。大量真实矩阵使用压缩稀疏行格式时性能良好；不过，如果选择其他格式能够显著提升性能，那么能够预测稀有事件的自动调优器就很有用。

### *C.* 最优格式性能

作为第七节 B 部分所示实验的一部分，我们记录了稀疏矩阵-向量乘法运算的计时结果，以量化最优格式不是压缩稀疏行格式时的实际收益。图 3 对比了在 OpenMP 后端上，使用压缩稀疏行格式执行稀疏矩阵-向量乘法的运行时与数据集中每个矩阵使用最优格式时对应的运行时，并涵盖所有可用系统。为清晰起见，省略了最优格式为压缩稀疏行格式的矩阵。尽管许多矩阵的加速比低于 $1.5\times$，但仍有相当数量的矩阵加速比介于 $1.5\times$ 和 $10.5\times$ 之间；在 Cirrus 、 XCI 和 A64FX 上，平均加速比约为 $1.8\times$ ，在 ARCHER2 上则为 $1.3\times$ 。相同系统上的串行后端也得到了类似结果；由于篇幅有限，故未列出。

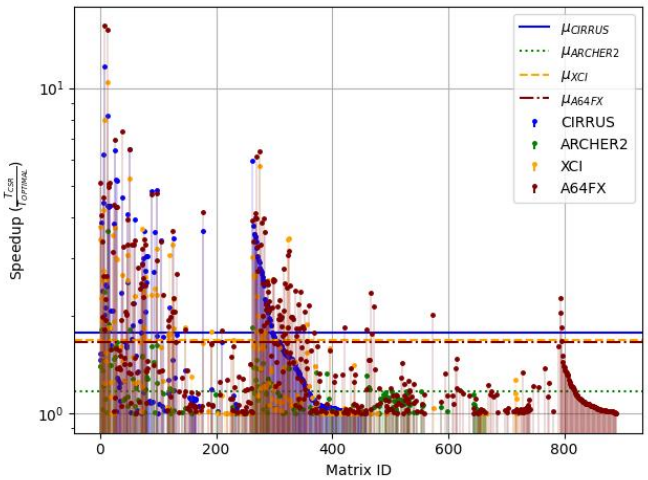



图 3：在可用系统的 OpenMP后端上，针对 SuiteSparse 数据集，使用最优格式的稀疏矩阵-向量乘法相对于压缩稀疏行格式的运行时加速比。为清晰起见，省略最优格式设置为压缩稀疏行格式的矩阵。

对于 CUDA 和 HIP 后端，与在 CPU 后端上测得的运行时加速比相比，其加速效果更为显著。如图 4 所示，CUDA 和 HIP 后端的平均加速比分别为 $8\times$ 和 $10\times$ ，

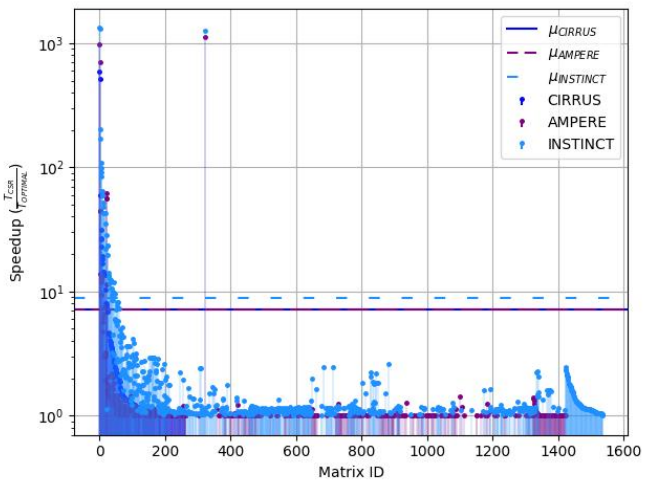



图 4：在 CUDA 和 HIP 后端上，采用最优格式的稀疏矩阵-向量乘法相对于压缩稀疏行格式的运行时加速比。在 Cirrus 和 Ampere（Isambard）系统上使用 NVIDIA V100 和 A100 图形处理器，在 Instinct（Isambard）系统上使用 AMD MI100。为便于阅读，省略了最优格式为压缩稀疏行格式的矩阵。在 CUDA（V100 和 A100 均如此）和 HIP 后端上，平均加速比分别为 $8\times$ 和 $10\times$ 。

最高加速比可达 $1000\times$。仔细检查内存读写请求，以及启动稀疏矩阵-向量乘法内核时达到的占用率后发现，对于其中一个矩阵（*mawi 201512020030*），压缩稀疏行格式版本发出的请求多出$5\times$，占用率则比采用最优格式的版本低 $10\times$ 。使用 CSR 格式时，该矩阵的稀疏模式会导致非合并访问，使图形处理器未能得到充分利用。

这些结果证明了开发和使用自动调优器（例如本文提出的自动调优器）的合理性，因为格式选择能够显著提升运算性能。不过，当最优格式为压缩稀疏行格式时，自动调优器必须足够轻量，以避免性能下降。

### *D.* 超参数调优

机器学习算法的超参数调优对机器学习模型的整体性能至关重要。这个过程更多依赖实验结果而非理论，而确定最优设置的最佳方法是尝试不同的超参数组合并评估模型性能。为避免过拟合并确保模型能够很好地泛化到未见过的数据，我们在训练集上进行五折交叉验证，并迭代拟合模型五次，每次使用其中四折进行训练，并在 $5th$ 上验证。

对于超参数调优，我们在预先定义的超参数空间中执行网格搜索，以寻找最优超参数值，每次都执行完整的五折交叉验证流程。我们比较所有生成的模型，并选出最佳模型，使用完整训练集进行训练，从而得到调优后的模型。在本例中，我们对决策树和随机森林 分类器都执行调优过程。

表 III 展示了每种可用后端和系统上用于生成基线（左侧子列）和调优后的随机森林 （右侧子列）的超参数，以及它们在测试集上达到的准确率和平衡准确率。决策树 分类器也取得了类似的定性结果，因此为节省篇幅不再列出。我们针对影响随机森林 分类器泛化能力的主要参数进行调优（估计器数量、树的最大深度以及叶节点上的最小样本数），同时也测试了不同的次要参数，例如自助采样（对数据点进行有放回或无放回抽样）、分裂前所需的最小样本数、分裂节点前考虑的最大特征数，以及用于衡量分裂质量的准则函数。

尽管调优过程得到的模型所报告的平均准确率（92.63%）与基线（92.36%）相近，但调优后的模型使用的估计器（树）数量显著更少、深度也更浅，因此预测时间大幅缩短。不过，由于数据集不平衡，更具参考价值的报告指标是平衡准确率

| 系统 | 后端 | 估计器 | | 自助采样 | | Max 深度 | | Min 样本 Leaf | | Min 样本 分裂 | | Max 特征 | | 准则 | | 准确率 (%) | | 均衡 准确率 (%) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ARC HER2 | 串行 | 100 | 40 | T | F | 21 | 21 | 1 | 3 | 2 | 2 | 10 | 4 | gini | 熵 | 95.13 | **95.86** | 83.65 | **88.49** |
| | OpenMP | | 40 | | T | 20 | 14 | | 1 | | 10 | | 9 | | 熵 | 91.94 | **92.18** | 83.39 | **85.12** |
| Cirrus | 串行 | | 50 | | T | 18 | 18 | | 2 | | 2 | | 6 | | 熵 | 93.60 | **94.08** | 79.13 | **81.71** |
| | OpenMP | | 30 | | T | 19 | 15 | | 1 | | 10 | | 8 | | gini | **92.65** | 91.71 | 68.36 | **78.39** |
| | Cuda | | 50 | | F | 17 | 16 | | 1 | | 10 | | 4 | | 熵 | 92.89 | **93.60** | 71.72 | **74.32** |
| A64FX | 串行 | | 90 | | T | 20 | 18 | | 1 | | 2 | | 5 | | gini | 87.93 | **87.93** | 84.56 | **86.22** |
| | OpenMP | | 30 | | T | 19 | 13 | | 1 | | 5 | | 6 | | gini | 91.26 | **91.75** | 89.29 | **91.62** |
| P3 | Cuda | | 40 | | F | 22 | 14 | | 2 | | 10 | | 4 | | 熵 | 86.46 | **87.17** | **84.95** | 83.82 |
| | HIP | | 40 | | T | 19 | 11 | | 1 | | 2 | | 6 | | 熵 | **93.38** | 92.67 | **90.82** | 87.92 |
| XCI | 串行 | | 60 | | F | 24 | 12 | | 2 | | 10 | | 6 | | 熵 | 96.21 | **96.92** | 91.34 | **95.72** |
| | OpenMP | | 20 | | T | 16 | 10 | | 1 | | 5 | | 10 | | 熵 | 94.54 | **95.01** | 55.25 | **75.31** |
| | | | | | | | | | | | | | | | **Mean** | 92.36 | 92.63 | 80.22 | 84.42 |
| | | | | | | | | | | | | | | | 标准差 (±) | 2.93 | 3.02 | 11.04 | 6.64 |

表 III：随机森林分类器调优过程中使用的超参数。还报告了测试集上的准确率和平衡准确率。我们报告了基线（左侧子列）和调优后（右侧子列）的分类器参数及得分。调优过程采用五折交叉验证（交叉验证）方法。

其计算方法是分别求出每个类别中正确分类样本所占比例，再取平均值。在本例中，调优后的模型（84.42% ± 6.64 %）的表现明显优于基线（80.22% ± 11.04%）。值得指出的是，对于某些系统与后端的组合，平衡准确率的变化幅度相当大（例如，使用 OpenMP 后端的 XCI 和 Cirrus 系统，准确率分别提升了 10% 和 20%）。

供参考，调优后的决策树 达到的准确率和平衡准确率分别为 90.85% ± 7.87% 和 78.12% ±4.91%。这一结果证明了开发 *DecisionTreeTuner* 和 *RandomForestTuner* 的合理性，因为使用 *DecisionTreeTuner* 时可以更快地进行预测，同时不会显著牺牲预测准确率。

### *E.* 自动调优器性能

将参数设置如表 III 所示的调优后随机森林 分类器部署在 C++ 中，并在合成基准测试中使用*RandomForestTuner*在 *Oracle* 上运行。该基准测试使用一个稀疏矩阵执行 1000 次稀疏矩阵-向量乘法运算，并在运行时将矩阵切换为调优器选定的最优格式。基准测试所用的数据集包含测试集中的所有矩阵。对于每个系统和后端组合，以采用 CSR 格式的等效稀疏矩阵-向量乘法运算来衡量自动调优器的运行时性能，计算方式为 $T_{tuning} = \frac{T_{CSR}}{T_{FE}+T_{PRED}}$ $T_{CSR}$ $T_{FE}$ $T_{PRED}$ ，其中 、 和 分别为单次 CSR 稀疏矩阵-向量乘法、特征提取和预测运算的运行时。

| 系统 | 后端 | Mean | Std | Min | Q1 | Q2 | Q3 | Max |
|---|---|---|---|---|---|---|---|---|
| ARC<br>HER2 | 串行<br>OpenMP | 10<br>25 | 19<br>20 | 2<br>2 | 4<br>14 | 7<br>21 | 10<br>31 | 303<br>179 |
| Cirrus | 串行<br>OpenMP<br>CUDA | 10<br>64<br>7 | 30<br>72<br>3 | 2<br>2<br>1 | 3<br>21<br>6 | 4<br>33<br>6 | 7<br>83<br>8 | 359<br>643<br>29 |
| A64FX | 串行<br>OpenMP | 6<br>45 | 9<br>40 | 1<br>1 | 3<br>23 | 4<br>28 | 5<br>59 | 120<br>246 |
| P3 | CUDA<br>HIP | 2<br>15 | 3<br>9 | 1<br>1 | 2<br>7 | 2<br>18 | 2<br>23 | 42<br>30 |
| XCI | 串行<br>OpenMP | 12<br>17 | 28<br>29 | 2<br>2 | 6<br>3 | 7<br>6 | 9<br>18 | 335<br>203 |

表 IV：使用自动调优器的运行时开销，以采用 CSR 的稀疏矩阵-向量乘法运算次数表示。测量方式为$T_{tuning} = \frac{T_{CSR}}{T_{FE}+T_{PRED}}$，其中 $T_{CSR}$、 $T_{FE}$和 $T_{PRED}$分别表示单次 CSR 稀疏矩阵-向量乘法、特征提取和预测运算的运行时。各列展示运行时开销的统计数据，Q1 到 Q3 表示四分位数。

表 IV 以每个系统与后端组合使用 CSR 执行的稀疏矩阵-向量乘法运算次数的形式，展示了调优器的运行时开销统计数据。我们确实观察到，无论系统如何，OpenMP后端运行自动调优器的平均耗时都更长，尽管与对应的串行后端相比，它使用的估计器可能更少。这表明，使用 OpenMP后端的随机森林相比串行后端，包含的估计器更少，但复杂度更高。另一方面，图形处理器上的*RandomForestTuner* 运行自动调优器所需的时间更短，因为其平均运行时开销仅相当于少数几次重复，而 CPU后端则不然。至少 75% 测试集中的矩阵在调优过程中所需的重复次数少于 100 次。在实际应用中，例如求解随时间变化的偏微分方程（PDE），可能需要执行数千次稀疏矩阵-向量乘法运算，这意味着所提出的自动调优器不会产生明显的开销。即使在观测到的最大情况中，运行调优器的开销仍处于可接受的范围内。

### *F.* 调优后的稀疏矩阵-向量乘法性能

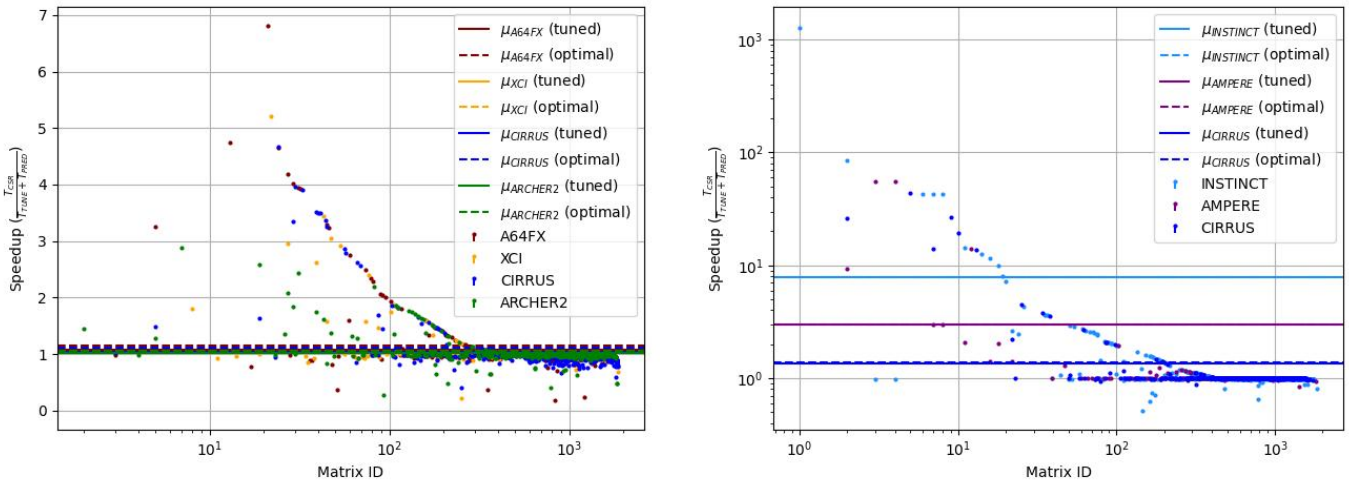
采用与第 VII-E 节相同的实验设置，我们还量化了使用所提出的自动调优器相较于使用 CSR 进行稀疏矩阵-向量乘法所获得的运行时加速比。加速比的测量方式如公式 2 所示:

$$Speedup = \frac{T_{CSR}}{T_{TUNE} + T_{OPT}} = \frac{T_{CSR}}{T_{FE} + T_{PRED} + T_{OPT}} \tag{2}$$

其中，$T_{CSR}$、$T_{OPT}$、$T_{FE}$和 $T_{PRED}$分别表示采用 CSR 和预测格式执行 1000 次稀疏矩阵-向量乘法运算，以及特征提取和预测运算的运行时。

图 5 展示了测试集中的所有矩阵在可用系统上使用 OpenMP、CUDA 和 HIP 后端所获得的加速比。请注意，受篇幅限制，图中未列出串行后端；其趋势与 OpenMP 后端相似。在 CPU 上，引入自动调优器并选择预测格式所带来的性能与使用 CSR 格式时相近。所有可用系统上的结果均一致，平均加速比接近 $1.1\times$。对于大多数矩阵，自动调优器的开销不会降低整体性能，因为大部分样本集中在 1 附近。对于少数性能显著低于 1 的矩阵，我们确实观察到错误分类最优格式所造成的影响。不过，在许多情况下，自动调优过程带来了明显的加速，在 A64FX 系统上达到的最大加速比为 $7\times$ 。

在图形处理器后端（图 5b）上，与 OpenMP 后端相比，自动调优方法的收益要大得多，因为实现了更高的平均加速比。NVIDIA A100（Ampere）和 V100（Cirrus）图形处理器的平均加速比分别为 $1.5\times$和 $3\times$ ，这一结果与两种架构之间的带宽比相符。然而，AMD M150（Instinct）图形处理器是唯一明显受益于引入自动调优器的架构，其报告的平均加速比为 $8\times$。请注意，与 CPU 后端相比，在图形处理器上，误分类造成的性能损失较轻，因为显著低于 1 的样本较少。此外，对于若干矩阵，所实现的加速比使性能提升了数个数量级，这凸显了采用自动调优方法进行格式选择的重要性。图 5 所示的三个后端中，使用自动调优器所实现的平均加速比与选取最优格式时的平均最优加速比相当（无需执行任何自动调优），这表明随着稀疏矩阵-向量乘法重复次数增加，自动调优器引入的开销会变得可以忽略不计。




(a) OpenMP

(b) CUDA/HIP

图 5：对于测试集中的每个矩阵，在可用系统上执行 1000 次稀疏矩阵-向量乘法运算时，使用自动调优器和预测格式相对于使用压缩稀疏行格式所获得的运行时加速比。使用预测格式所获得的平均加速比（$\mu$，其中也包括调优器进行预测所需的时间）与直接使用最优格式所获得的平均加速比相当，这表明使用调优器引入的平均开销极小。

## VIII. 相关工作

多年来，机器学习已被证明是执行各种优化任务的有效方法，例如代码优化、任务调度和模型选择 [18]。应用自动调优技术来优化稀疏线性代数仍是一个活跃的研究领域，相关进展涵盖从特定格式的参数调优（ [5], [10], [18]）到面向当前及新兴架构的自动格式选择等主题。

Benatia 等人 [25] 提出了一种支持向量机（SVM）分类器，用于从四种可用格式中为图形处理器上的稀疏矩阵-向量乘法选择最优格式，报告的分类准确率最高可达 88%。类似地，Sedeghati 等人 [26] 使用基于决策树的分类器，从五种可用格式中进行选择，在图形处理器上的准确率达到 81%。另一方面，Li 等人 [13] 使用了一种基于规则集分类的输入自适应稀疏矩阵-向量乘法自动调优器，该调优器为每个测试样本维护一个置信值。如果分类器的预测低于设定阈值，自动调优器就会选择最优格式和稀疏矩阵-向量乘法内核；否则则切换到先运行方法来做出决策，报告的准确率最高可达 85%。

为缓解手动定义待提取特征所带来的问题，Zhao 等人 [16]使用卷积神经网络（CNN）模型，在 CPU 和 GPU 平台上为稀疏矩阵-向量乘法选择最优格式。这种方法需要将输入矩阵转换/压缩为固定大小（$128 \times 128$）的类图像表示，以便输入网络；不过，它的准确率最高（在 CPU 和 GPU 平台上分别为 93% 和 90% ），代价是预测时间比其他机器学习方法更长。

Zhao 等人 [27] 进一步推进了格式选择方法，提出了一种注重开销的选择机制，适用于基于 SpMV 的应用程序，并将格式转换的开销也纳入考量。通过构建多个回归模型，该方法报告的准确率最高可达 88%，应用程序的平均运行时加速比介于 $1.14\times$ - $1.43\times$。

我们提出的自动调优器成功地将机器学习方法（例如 Li 等人提出的方法）的速度与 Zhao 等人报告的准确率结合起来，并适用于多种存储格式和大多数关键高性能计算平台。不过需要注意的是，尽管先前工作报告的数据集存在类别不平衡问题，但没有作者报告所达到的平衡准确率，因此无法仅依据准确率公平比较各项工作的贡献。

## IX. 结论与后续工作

选择最优稀疏矩阵存储格式，对于确保应用程序在现有硬件架构上始终保持最优性能至关重要。然而，选择过程并非易事。机器学习通过将这一问题视为分类任务，为其提供了系统化的解决方案。就格式选择而言，由于数据中存在类别不平衡，这一问题可归类为稀有事件预测问题。通过训练、调优并部署由决策树组成的集成模型，我们能够准确预测在主要高性能计算架构上执行稀疏矩阵-向量乘法运算时应采用的最优格式。我们发现，尽管大多数情况下压缩稀疏行格式是最佳选择，但在某些情况下，切换到最优格式可使运行时性能提升数个数量级。最后，我们提出的轻量级自动调优方法会增加稀疏矩阵-向量乘法的总体运行时开销，但这一开销通常只需执行少量稀疏矩阵-向量乘法运算便可摊销，

平均而言，几次稀疏矩阵-向量乘法运算后即可摊销这一开销，对图形处理器的收益尤为显著。

下一步，我们将探索进一步提高模型准确率的方法，例如通过平衡数据集或采用其他机器学习方法，如梯度提升决策树。此外，消除对手动特征提取的需求仍是未来研究的一个方向。

## 致谢

本研究是英国工程与自然科学研究委员会项目 ASi-MoV (EP/S005072/1) 的一部分。我们使用了 ARCHER2 英国国家超级计算服务 (https://www.archer2.ac.uk)、由爱丁堡大学和英国工程与自然科学研究委员会 (EP/P020267/1) 资助的 EPCC Cirrus英国国家二级高性能计算服务 (http://www.cirrus.ac.uk)，以及由 GW4 和英国气象局运营、由英国工程与自然科学研究委员会 (EP/T022078/1) 资助的 Isambard 2英国国家二级高性能计算服务 (http://gw4.ac.uk/isambard)。

## 参考文献

[1] T. Ichimura, K. Fujita, T. Yamaguchi, A. Naruse, J. C. Wells, T. C. Schulthess, T. P. Straatsma, C. J. Zimmer, M. Martinasso, K. Nakajima, M. Hori 和 L. Maddegedara，“利用人工智能与混合精度计算，在低阶非结构化有限元上求解非线性地震城市问题的快速可扩展隐式求解器”，收录于*SC18*：国际高性能计算、网络、存储与分析大会，2018年11月，第 627–637 页。[2] A. M. B. Owenson, S. A. Wright, R. A. Bunt, Y. K. Ho, M. J. Street 和 S. A. Jarvis，“用于预测生产级 CFD 代码性能的非结构化 CFD 迷你应用程序”，刊于《并发与计算：实践与经验》，第 32 卷，第 10 期，文章编号 e5443，2020 年，e5443 cpe.5443。 [在线]。获取地址：https://onlinelibrary.wiley.com/doi/abs/10 1002 5443 。/cpe。P. Colella, [3] “定义科学计算的软件需求”，http://view.eecs.berkeley.edu/w/images/temp/6/6e/20061003235551!DARPAHPCS.ppt，2006年。[4] S. Filippone、V. Cardellini、D. Barbieri 和 A. Fanfarillo，“通用图形处理器上的稀疏矩阵-向量乘法”，刊于《*ACM* 数学软件汇刊》，第 43 卷，第 4 期，2017年1月。 [在线]。获取地址：https: //doi.org/10 1145 3017994 。/ E. Coronado-Barrientos、M. [5] Antonioletti 和 A. Garcia-Loureiro，“一种利用 AVX-512指令和 CUDA 实现高效 SpMV 运算的新型 AXT格式”，刊于《工程软件进展》，第 156 卷，文章编号 102997，2021 年。 [在线]。获取地址：https://www.sciencedirect.com/science/article/pii/S0965997821000260 [6] 5 W. Liu 和 B. Vinter，“CSR：一种适用于跨平台稀疏矩阵-向量乘法的高效存储格式”，收录于第*29*届 *ACM* 超级计算国际会议论文集，丛书 ICS ’15。美国纽约州纽约：计算机协会，2015 年，第 339 350 [ ] 10 1145 2751205 2751209 – 页。[7] 在线。获取地址：https://doi.org/。/。W. Yang、K. Li、Y. Liu、L. Shi 和 L. Wan，“图形处理器上准对角线矩阵-向量乘法的优化”，刊于《国际高性能计算应用杂志》，第 28 卷，第 2 期，第 183–195 页，2014年。 [在线]。获取地址：https://doi.org/10.1177/ 1094342013501126[8] A. Monakov、A. Lokhmotov 和 A. Avetisyan，“针对图形处理器架构自动调优稀疏矩阵-向量乘法”，收录于第*5*届高性能嵌入式架构与编译器国际会议论文集，丛书 HiPEAC’10。德国柏林、海德堡：施普林格出版社，2010 年，第 111–125 页。 [在线]。获取地址：https://doi.org/10 1007 978 3 642 11515 8 10 。/ - - - - B.-Y. Su 和 K. Keutzer, [9] “ClSpMV：一种面向图形处理器的跨平台 OpenCL SpMV 框架”，收录于第*26*届*ACM*国际超级计算大会论文集，丛书 ICS ’12。美国纽约州纽约：计算机协会，2012 年，第 353–364 页。 [在线]。获取地址：https://doi.org/10.1145/2304576.2304624 [10] M. Kreutzer、G. Hager、G. Wellein、H. Fehske 和 A. Bishop，“一种适用于现代宽 SIMD 单元处理器的通用稀疏矩阵-向量乘法数据格式”，《SIAM 科学计算杂志》，第 36 卷，2013 年 7 月。[11] C. Stylianou 和 M. Weiland，“利用动态稀疏矩阵实现可移植高性能线性代数运算”，收录于2022 年 IEEE/ACM 高性能计算性能、可移植性与生产力国际研讨会（P3HPC）。美国加利福尼亚州洛斯阿拉米托斯：IEEE 计算机学会，2022年11月，第 47–57 页。 [在线]。获取地址：https://doi.ieeecomputersociety。org/10 1109 3 56579 2022 00010 。/P HPC。/。S. Filippone 和 A. [12] Buttari，“Fortran 2003 中稀疏矩阵计算的面向对象技术”，《ACM 数学软件汇刊》，第 38 卷，第 4 期，2012 年 8 月。 [在线]。获取地址：https: // doi.org/10 1145 2331130 2331131 。/。J. Li、G. Tan、M. Chen 和 N. Sun，“SMAT：一种输入自适应稀疏矩阵-向量乘法自动调优器”，收录于第 [13] 34届 ACM SIGPLAN 编程语言设计与实现会议论文集，丛书 PLDI ’13。美国纽约州纽约：计算机协会，2013 年，第 117–126 页。 [在线]。获取地址：https://doi.org/10 1145 2491956 2462181 。/。H. [14] Anzt、T. Cojean、G. Flegar、F. Gobel、T. Grutzmacher、P. Nayak、T. Ribizel、Y. M. Tsai 和 E. S. Quintana-Ort´ı，“Ginkgo：一个面向高性能计算的现代线性算子代数框架”，《ACM 数学软件汇刊》，第 48 卷，第 1 期，第 2:1–2 33, 2022 [ ] 10 1145 3480935 页：2 月。[15] 在线。获取地址：https://doi.org/。/ G. Tan、J. Liu 和 J. Li，“面向多核与众核架构的自适应 SpMV 库的设计与实现”，《ACM 数学软件汇刊》，第 44 卷，第 4 期，2018 年 8 月。 [在线]。获取地址：https://doi.org/10 1145 3218823 。/ Y. Zhao、J. Li、C. Liao 和 X. Shen，“弥 [16] 合深度学习与稀疏矩阵格式选择之间的差距”，收录于第23届 ACM SIGPLAN 并行编程原理与实践研讨会论文集，丛书 PPoPP ’18。美国纽约州纽约：计算机协会，2018 年，第 94–108 页。 [在线]。获取地址：https://doi.org/10 1145 3178487 3178495 。/。C. Stylianou，“ [17] Morpheus-Oracle：一个用于高效自动选择稀疏矩阵存储格式的库”，2023 年 2 月。 [在线]。获取地址：https://github.com/morpheus-org/morpheus-oracle [18] S. Chen、J. Fang、D. Chen、C. Xu 和 Z. Wang，“新兴众核架构上稀疏矩阵-向量乘法的自适应优化”，收录于2018 IEEE 第20届高性能计算与通信国际会议、IEEE 第16届智慧城市国际会议及 IEEE 第4届数据科学与系统国际会议（HPCC/SmartCity/DSS），2018 年，第 649–658 页。[19] N. Bell 和 M. Garl，“CUDA 上高效的稀疏矩阵-向量乘法”，NVIDIA，技术报告，2008 年。[20] T. A. Davis 和 Y. Hu，“佛罗里达大学稀疏矩阵集合”，《ACM 数学软件汇刊》，第 38 卷，第 1 期，2011 年 12 月。 [ ] 10 1145 2049662 2049663 在线。获 [21] 取地址：https://doi.org/。/。F. Pedregosa、G. Varoquaux、A. Gramfort、V. Michel、B. Thirion、O. Grisel、M. Blondel、P. Prettenhofer、R. Weiss、V. Dubourg、J. Vander-plas、A. Passos、D. Cournapeau、M. Brucher、M. Perrot 和 E. Duch-esnay，“Scikit-learn：Python 中的机器学习”，《机器学习研究杂志》，第 12 卷，第 2825–2830 页，2011 年。[22] EPCC，“ARCHER2英国国家超级计算服务”，2022年。[在线]。获取地址：http://www.archer2.ac.uk/ [23] EPCC，“Cirrus英国国家二级高性能计算服务”，2020年。 [在线]。获取地址：http://www.cirrus.ac.uk/ [24] GW4 和英国气象局，“Isambard 2英国国家二级高性能计算服务”，2022年。 [在线]。获取地址：http://gw4.ac.uk/isambard/ [25] A. Benatia、W. Ji、Y. Wang 和 F. Shi，“利用多类支持向量机为图形处理器上的稀疏矩阵-向量乘法选择稀疏矩阵格式”，收录于2016年第45届并行处理国际会议（ICPP），2016 年，第 496–505 页。[26] N. Sedaghati、T. Mu、L.-N. Pouchet、S. Parthasarathy 和 P. Sadayappan，“在图形处理器上自动选择稀疏矩阵表示”，收录于第29届 ACM 超级计算国际会议论文集，丛书 ICS ’15。美国纽约州纽约：计算机协会，2015 年，第 99–108 页。 [在线]。获取地址：https://doi.org/10 1145 2751205 2751244 。/。Y. Zhao、W. Zhou、X. Shen [27] 和 G. Yiu，“面向 SpMV 应用程序的注重开销的格式选择”，收录于2018 IEEE 并行与分布式处理国际研讨会（IPDPS），2018 年，第 950–959 页。