arXiv:2601.00974v1 [cs.NI] 2 Jan 2026

改进图挑战参考实现

Inna Voloshchuk、Hayden Jananthan、Chansup Byun、Jeremy Kepner，MIT

摘要—MIT/IEEE/Amazon 图挑战为个人和团队展示大规模图与稀疏数据分析领域的新创新提供了平台。匿名网络传感图挑战赛处理超过 1000 亿个网络数据包，以构建隐私保护流量矩阵，并通过 GraphBLAS 参考实现展示如何将超稀疏矩阵应用于这一问题。本研究对部分参考代码进行了重构和基准测试，以提升其清晰度、适应性和性能。原有的 Python 实现分布在 3 个文件中，约有 1000 行代码；经过精简后，代码被整理为两个专注的模块，共 325 行，在保持全部功能的同时，将代码量减少了 67%。借助 pMatlab 和 pPython 分布式数组编程库，新增的并行映射实现了数据的并行基准测试。结果展示了对流量矩阵进行大规模求和与分析时的可扩展性能。该实现为参赛者提供了清晰高效的基础，进一步提升了图挑战赛的潜在影响力。

索引词—**图挑战赛、互联网分析、并行计算、流量矩阵、网络流量、GraphBLAS**

I. 引言

MIT/IEEE/Amazon 图挑战是一个推动图和稀疏数据分析领域技术发展的平台。图挑战赛支持由社区推动的软件、硬件、算法和系统创新方法，以处理反映现实世界复杂性和规模的大型图及数据集 [1]。

匿名化网络感知挑战赛是这一更广泛计划中发布的一项挑战，重点关注兼顾隐私的网络流量数据分析。挑战赛的目标是改进匿名化流量矩阵的构建与分析。通过从流式网络数据包构建匿名化流量矩阵，研究人员可以在保护隐私的同时提取网络属性和统计信息 [1]。图 1 所示的流量矩阵表示法，能够利用线性代数高效计算网络属性。

本研究由美国空军部人工智能加速器资助，并依据合作协议完成编号 FA8750-19-2-1000。本文件所载的观点和结论均为作者所有，不应被解释为美国空军部或美国政府明示或暗示的官方政策。尽管本文包含任何版权声明，美国政府仍有权为政府目的复制和分发转载件。

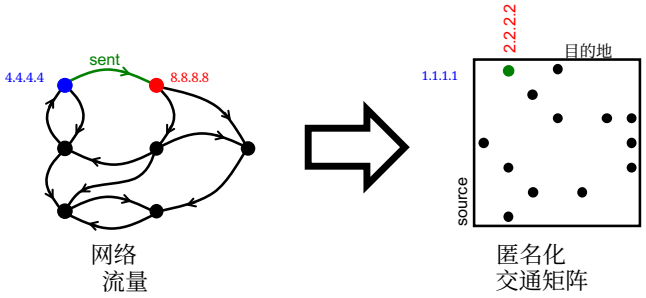

图1. **从网络图到匿名化流量矩阵**。 每个网络数据包都包含一个源 IP 地址和一个目的地 IP 地址。这些地址可以聚合为矩阵表示，其中行对应源，列对应目的地。经过修改的匿名化网络流量图挑战赛参考实现，重点在于读取和分析由网络流量构成的流量矩阵。图片来源： [1]。

随着系统日趋复杂，网络流量分析对于网络安全和异常检测变得愈发重要。这项挑战使用了超过 1000 亿个网络数据包，这些数据包由应用互联网数据分析中心（CAIDA）运营的互联网望远镜收集而来 [2]。为应对数据规模，所提供的参考实现采用 GraphBLAS 超稀疏矩阵，可实现无需降采样、高性能和开放标准 [3]–[16]。这一初始参考实现虽已具备完整功能，但仍有重构空间，以提升清晰度、简洁性和性能。

本文介绍了一种全新的精简参考实现，在保持功能不变的同时将代码量减少了 67%。此外，新的 Python 实现和现有的 Matlab/Octave 实现均适用于通过 pPython [17] 和 pMatlab [18] 进行分布式数组并行编程。我们对所有数据进行了基准测试，测试了每个进程使用不同线程数的情况，并将多节点扩展测试至 32 个节点。

本文其余部分组织如下。首先介绍图挑战赛以及匿名化网络感知挑战赛的具体要求。随后，详细说明参考实现所采用的具体方案及所做的更改。接下来介绍分布式数组，以及它们在该问题中的应用。最后展示基准测试结果及其意义。

979-8-3315-5937-3/25/$31.00 ©2025 IEEE

```
ReadSumAnalyzeMatrices(
  Np, # 文件中的数据包 (2^30)
  Nv, # 每个矩阵的数据包 (2^17)
  NmatPerFile，# 每个输出文件的矩阵数 (2^6)
);
甲_t(:,:) = 0;
对于i = 0 至 (Np/(NmatPerFile*Nv))-1
  一个 =读取矩阵(i);
  对于j = 0 至NmatPerFile-1
    A_t +=A[j];
  end
end
# 对A_t进行分析
end
```

图 2. **匿名网络传感图挑战赛读取、求和与分析伪代码**。交通矩阵 $\mathbf{A}[j]$ 此前已使用匿名化的源 IP 地址和目的地 IP 地址创建。随后，将每组包含 2^6 个的交通矩阵保存为 .tar 归档文件中的单独文件。每个文件均保存了 2^7 个 .tar 归档文件。读取这些 .tar 文件，并将所有交通矩阵相加，得到一个交通矩阵 $\mathbf{A}_t$。然后，对$\mathbf{A}_t$ 执行《匿名网络感知图挑战赛论文》中表 1 所示的分析，并报告结果。

II. 匿名网络传感图挑战赛

网络流量可以表示为一个矩阵，其中每一行代表一个源，每一列代表一个目的地（图 1）。数据包数据流被划分为多个时间窗口，每个时间窗口包含 2^{30} 个数据包。矩阵中的源和目的地直接取自数据包头，可以对其进行匿名化处理，而不会丢失网络的本质，因为行或列的置换不会影响通过矩阵运算提取的网络属性。有关交通矩阵构建的更多信息，请参阅《匿名网络感知图挑战赛论文》 [1]。

匿名网络传感图挑战赛包含六个步骤：

1) 读取数据包捕获（PCAP）文件。2) 提取源和目的地互联网协议（IP）地址。3) 对地址进行匿名化。4) 构建交通矩阵。5) 将矩阵保存到文件中。6) 读取、求和并分析已存储的文件。

本工作重点关注第 6 步），其中 2^{13} 表示 2^{30} 数据包的流量矩阵被汇总为单个矩阵 $\mathbf{A}_t$ ，以便按照《匿名网络感知图挑战赛论文》 [1]中表 1 所述进行分析。图 2 展示了用于读取、求和并分析匿名化矩阵的参考伪代码。

还可以使用对角矩阵作为掩码来选择源地址和目的地地址的子集，从而对流量矩阵执行子范围分析。参考实现通过应用这些掩码创建的流量矩阵会以单独的 tar 归档文件保存，以便日后使用。

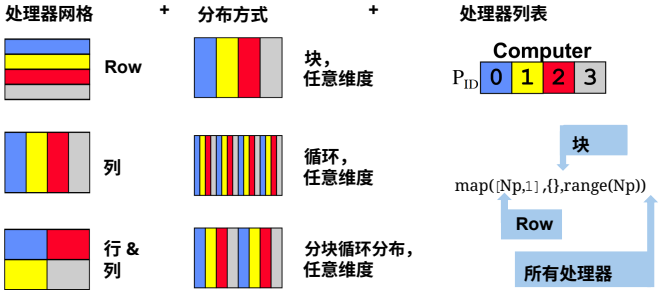

图 3. 引入映射来处理如何在各进程之间分配工作。映射有三个主要参数：处理器网格、分布方式和处理器列表。处理器网格构建一个应用于数组的网格，使数组可以按行、按列或同时按行和列划分。分布方式指数组各部分如何分配给处理器，可指定为块、循环、分块循环分布和块重叠（图中未示出）。处理器列表描述哪些处理器将接收通过此映射创建的分布式数组中的一部分。本文基准测试所用的映射将各行分成块，并将工作分配给所有处理器。

III. 实现改进

原始 Python 参考实现的代码约有 1000 行，分布在三个文件中，并包含分层更新、扩展统计选项以及通过 TrafficMatrix 类实现的面向对象抽象等功能。虽然这些功能为更多使用场景提供了灵活性，但也增加了复杂性，妨碍了提供清晰、易于理解的参考实现这一主要目标。

更具体地说，尽管 TrafficMatrix 类在更广泛的场景中可能很有用，但对于图挑战赛的特定用例而言，它带来了不必要的开销。涉及子范围时，每个类实例都会为每个时间窗口保留过多信息，例如，在相关性最高的阶段过后，仍创建并跟踪所有生成的子范围掩码矩阵。这可能会在处理大量数据时导致内存问题。此外，尽管底层矩阵运算在数学上等价，却使用单独的函数处理子范围矩阵的分析计算。多个可选功能模糊了算法的核心结构，使新研究人员难以理解其基本方法。

精简后的实现由两个简洁的文件组成，共 325 行代码：

1) **Tar 和矩阵处理**（~ 85 **行**）：用于 tar 文件管理和 GraphBLAS 矩阵处理的函数。2) **挑战赛流程**（~ 240 **行**）：用于求和、分析和处理的较高层函数，包括主 process_filelist 函数，该函数完成单个时间窗口的完整步骤。

我们移除了额外的抽象，改用直接的 GraphBLAS 矩阵运算，尤其是在分析方面。现在，一个分析函数即可处理所有矩阵变体，并一并计算《匿名网络感知图挑战赛论文》 [1] al- 中表 1 所述的全部九项属性，重用相关值。此实现遵循

代码清单 1 (并行 Matlab 程序)。

```
% filelist 是要处理的 tar 文件名列表。
N = length(filelist)
Filemap =map([Np,1],{},0:Np-1); % 映射。
z = zeros(N, 1, map=Filemap);
my_i_global =global_ind(z,1);
for i_local = 1:length(my_i_global)
    i_global =my_i_global(i_local);
    file =filelist{i_global};
    % 继续读取、汇总并分析此文件
end
```

代码清单 2（并行 Python 程序）。

```
import pPython as GPC
# filelist 是要处理的 tar 文件名列表。
N =len(filelist)
Filemap = Dmap([Np,1],{},range(Np))# 映射。
z =zeros(N, 1, map=Filemap)
my_i_global =global_ind(z,0)[0]
for i_local in range(len(my_i_global)):
    i_global =my_i_global[i_local]
    file =filelist[i_global]
    # 继续读取、求和并分析此文件
```

一种清晰的顺序模式，反映了匿名网络传感图挑战赛的基本场景，为社区参与提供了更易于理解的基础，也让研究人员能够轻松展示他们的改进。

IV. 用于并行化的分布式数组

我们使用 pPython [17] 和 pMatlab [18] 对精简后的串行参考实现进行基准测试，以评估其在并行架构上的性能。这两个库都采用基于映射的分布式数组编程概念，将数据划分到多个处理器上，同时保持串行算法清晰且完整。

映射提供了一种抽象，将算法设计与并行化分离。映射定义了三个关键要素，用于描述数组如何在进程之间分布，如图 3 所示。处理器网格描述如何划分数组，划分可按任意维度进行，例如按行或列，也可以按多个维度进行。分布方式指定数组在进程之间的分布方式，可以采用块、循环、分块循环或分块重叠方式。处理器列表则描述哪些处理器将接收由所创建映射分布的数据。

代码清单 1 和 2 展示了用于基准测试图挑战赛实现的代码片段。这些实现使用 Matlab/Octave 和 Python 中的分布式数组来读取、求和和分析矩阵。它们使用 pMatlab 和 pPython 在多个进程间分配工作，进程数量用 N_P 表示。

该方法的工作原理如下。首先，创建一个映射，以便在处理器之间分配工作。该映射将数组的行平均分配，并用 {} 表示按默认块分布方式进行分布；同时，所有处理器（以 P_{ID}表示）都包含在内，范围从 $P_{ID} = 0$ 到 $P_{ID} = N_P - 1$ 。此外，还会初始化一个空数组结构，其第一个维度表示必须处理的文件时间窗口总数。这将代表待完成的工作；通过跟踪进程之间划分的索引，每个唯一的 P_{ID} 都会知道自己负责哪一部分工作。映射的创建方式及创建映射时可用的选项如图 3 所示。

所有进程都会执行代码清单 1 和 2 中的代码，但每个进程只处理分配给自己的本地文件。由于每个文件都可以独立处理，因此无需进程间通信，从而消除了通信开销或同步开销。

代码清单 1 和 2 的说明如下：Python 代码的**第 1 行**导入所需的 Python 包。**第 4 行**创建并行映射 Filemap，指定将 $N\times1$ 元素列向量的各行，按照默认块分布（{}）平均分配给 Np 个进程，即分配给 P_{ID} = 0、.....、Np-1。

此基准测试脚本展示了分布式数组编程方法的优势。

- **代码影响小**。只需增加少量代码，即可将串行程序转换为并行程序。
- **并行库函数占用少**。只需使用两个并行库函数：map 和 global_ind。
- **可扩展**。只要满足 $N > N_P$，代码就可以用于任意问题规模或处理器数量。
- **映射独立性**。该程序适用于任何分布方式（即块、循环、分块循环分布）。
- **性能保证**。由于之后的任何操作都在处理器本地的独立文件上进行，因此运行这些代码行时可以保证不会产生隐性的性能损失。

基准测试时，通过将工作负载划分为较小的时间窗口来应对内存限制，确保每个进程处理的数据子集都在可管理范围内。代码清单 1 和 2 分别列出了并行 Matlab/Octave 和 Python 实现。

V. 性能测量

基准测试测量使用 MIT SuperCloud TX-E1 超级计算机 [19] 上配备双路 2.4 GHz Intel Xeon Platinum 8260 处理器的计算节点进行。重构后的代码分别使用 Python、Matlab 和 Octave 进行了测试，其中 Matlab/Octave 使用 pMat-lab 分布式数组，Python 则类似地使用 pPython。从单个节点上的一个进程和一个线程开始，线程数逐步翻倍，直至 16。随后

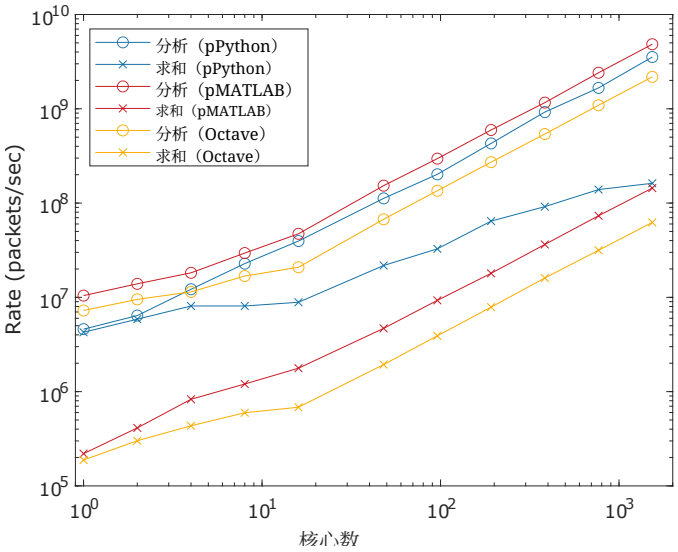

图 4. **性能比较**。三种语言——Python、MATLAB 和 Octave——中交通矩阵求和与分析代码的平均性能。使用 pMATLAB 分布式数组将文件分配给 3 个实例，每个实例配备 16 个 OpenMP 线程。求和与分析代码的内存占用较大；在配备 192 GB RAM 的 48 核节点上，可运行 3 个不同的实例，每个实例配备 16 个 OpenMP 线程，分别处理一个文件。多节点性能随着处理的不同文件数量增加而线性扩展。

完成了多节点基准测试：每个进程使用 16 个线程，每个节点运行三个进程，从单个节点开始，节点数逐步翻倍，直至 32 个节点。

图 4 展示了 Python、MATLAB 和 Octave 实现的性能比较。对交通矩阵求和与分析需要较大的内存占用，但可以通过线程加速。在这两种情况下，都可以同时处理多个文件，且性能随节点数线性扩展。

主要观察结果包括：

- 在所有语言中，求和所需时间始终长于分析。
- Python 的求和性能优于 MATLAB 和 Octave，这可能归功于经过优化的原地加法。各语言的分析耗时相近，因为它直接测试 GraphBLAS 的能力。
- 随着核心数和节点数增加，性能几乎呈线性扩展；但在更大规模下，Python求和的性能略有下降。

这些发现凸显了不同语言与 GraphBLAS 功能之间的权衡。结果表明，分布式数组编程非常适合并行化此类问题，只需较少工作便可在更多节点上加速文件处理。

VI. 结论

本文介绍了对匿名化网络感知图挑战赛参考实现最后一步的重构，在保持全部功能的同时，将代码规模减少了 67%。通过移除抽象和冗余计算，这一简化提升了代码的清晰度和适应性，使该实现成为更有效的社区参考。本文为图挑战赛社区提供了更易使用且更高效的参考实现，降低了新研究人员参与该挑战赛的门槛。

分布式数组技术以及映射的加入，有效地在各进程间分配了工作，同时最大限度地减少了对代码的影响。基准测试研究表明，增加并行操作很容易实现性能的线性扩展；利用更多节点还可以了解不同语言和工具之间的权衡。

致谢

作者谨向以下人士致谢，感谢他们的贡献与支持：Daniel Andersen、LaToya Anderson、Bill Arcand、Sean Atkins、Bill Bergeron、Chris Berardi、Bob Bond、Alex Bonn、Bill Cash man、K Claffy、Tim Davis、Chris Demchak、Alan Edelman、Peter Fisher、Jeff Gottschalk、Thomas Hardjono、Chris Hill、Michael Houle、Michael Jones、Charles Leiserson、Piotr Luszczek、Kirsten Malvey、Peter Michaleas、Lauren Milechin、Chasen Milner、Sanjeev Mohindra、Guillermo Morales、Julie Mullen、Heidi Perry、Sandeep Pisharody、Christian Prothmann、Andrew Prout、Steve Rejto、Albert Reuther、Antonio Rosa、Scott Ruppel、Daniela Rus、Mark Sherman、Scott Weed、Charles Yee、Marc Zissman。

参考文献

[1] H. Jananthan、M. Jones、W. Arcand、D. Bestor、W. Bergeron、D. Burrill、A. Buluc、C. Byun、T. Davis、V. Gadepally、D. Grant、M. Houle、M. Hubbell、P. Luszczek、P. Michaleas、L. Milechin、C. Milner、G. Morales、A. Morris、J. Mullen、R. Patel、A. Pentland、S. Pisharody、A. Prout、A. Reuther、A. Rosa、G. Wachman、C. Yee 和 J. Kepner，“匿名网络感知图挑战赛”，载于*2024*年*IEEE*高性能极限计算会议 *(HPEC)*，第 1–8 页，IEEE，2024 年 9 月。[2] “加州大学圣迭戈分校网络望远镜。”https://www.caida.org/projects/network telescope/. [3] J. Kepner、P. Aaltonen、D. Bader、A. Buluc、F. Franchetti、J. Gilbert、D. Hutchison、M. Kumar、A. Lumsdaine、H. Meyerhenke、S. McMillan、C. Yang、J. D. Owens、M. Zalewski、T. Mattson 和 J. Moreira，“GraphBLAS 的数学基础”，载于*2016*年*IEEE*高性能极限计算会议 *(HPEC)*，第 1–9 页，2016 年。[4] A. Buluc、T. Mattson、S. McMillan、J. Moreira 和 C. Yang，“C 语言 GraphBLAS API 的设计”，载于*2017*年*IEEE*国际并行与分布式处理研讨会研讨会 *(IPDPSW)*，第 643–652 页，2017 年。[5] C. Yang、A. Buluc 和 J. D. Owens，“在 GraphBLAS 中高效实现推拉”，载于第*47*届国际并行处理会议，第 1–11 页，2018 年。[6] J. Kepner 和 H. Jananthan，大数据数学：电子表格、数据库、矩阵和图。麻省理工学院出版社，2018 年。[7] T. A. Davis，“算法1000：SuiteSparse:GraphBLAS：使用稀疏线性代数语言实现的图算法”，*ACM* 数学软件汇刊 *(TOMS)*，第 45 卷，第 4 期，第 1–25 页，2019 年。[8] T. Mattson、T. A. Davis、M. Kumar、A. Buluc、S. McMillan、J. Moreira 和 C. Yang，“LaGraph：基于 GraphBLAS 构建图算法的社区协作项目”，载于*2019*年*IEEE*国际并行与分布式处理研讨会研讨会 *(IPDPSW)*，第 276–284 页，IEEE，2019 年。[9] P. Cailliau、T. Davis、V. Gadepally、J. Kepner、R. Lipman、J. Lovitz 和 K. Ouaknine，“支持 GraphBLAS 的 RedisGraph 图数据库”，载于*2019*年*IEEE*国际并行与分布式处理研讨会研讨会 *(IPDPSW)*，第 285–286 页，IEEE，2019 年。

[10] T. A. Davis、M. Aznaveh 和 S. Kolodziej，“快速编写，快速运行：通过 SuiteSparse:GraphBLAS 在 20 分钟开发时间内构建稀疏深度神经网络”，载于 *2019 IEEE* 高性能极限计算会议 *(HPEC)*，第 1–6 页，2019 年。[11] M. Aznaveh、J. Chen、T. A. Davis、B. Hegyi、S. P. Kolodziej、T. G. Mattson 和 G. Sz´arnyas，“使用 OpenMP 的并行 GraphBLAS”，载于*2020* 年工业与应用数学学会组合科学计算研讨会论文集，第 138–148 页，工业与应用数学学会，2020 年。[12] B. Brock、A. Buluc、T. G. Mattson、S. McMillan 和 J. E. Moreira，“GraphBLAS 2.0 简介”，载于*2021 IEEE* 并行与分布式处理国际研讨会专题讨论会 *(IPDPSW)*，第 253–262 页，IEEE，2021 年。[13] M. Pelletier、W. Kimmerer、T. A. Davis 和 T. G. Mattson，“Julia 和 Python 中的 GraphBLAS：PageRank 与三角中心性”，载于*2021 IEEE* 高性能极限计算会议 *(HPEC)*，第 1–7 页，2021 年。[14] M. Jones、J. Kepner、D. Andersen、A. Buluc、C. Byun、K. Claffy、T. Davis、W. Arcand、J. Bernays、D. Bestor、W. Bergeron、V. Gadepally、M. Houle、M. Hubbell、H. Jananthan、A. Klein、C. Meiners、L. Milechin、J. Mullen、S. Pisharody、A. Prout、A. Reuther、A. Rosa、S. Samsi、J. Sreekanth、D. Stetson、C. Yee 和 P. Michaleas，“边缘端 GraphBLAS：网络流量的匿名化高性能流处理”，载于 *2022 IEEE* 高性能极限计算会议 *(HPEC)*，第 1–8 页，2022 年。[15] T. Trigg、C. Meiners、S. Pisharody、H. Jananthan、M. Jones、A. Michaleas、T. Davis、E. Welch、W. Arcand、D. Bestor、W. Bergeron、C. Byun、V. Gadepally、M. Houle、M. Hubbell、A. Klein、P. Michaleas、L. Milechin、J. Mullen、A. Prout、A. Reuther、A. Rosa、S. Samsi、D. Stetson、C. Yee 和 J. Kepner，“ 使用 GraphBLAS 对数据包进行超稀疏网络流分析 ”，载于 *2022 IEEE* 高性能极限计算会议 *(HPEC)*，第 1–7 页，2022 年。[16] T. A. Davis，“算法 1037：SuiteSparse:GraphBLAS：使用稀疏线性代数语言实现的并行图算法”，*ACM* 数学软件汇刊，第 49 卷，第 3 期，第 1–30 页，2023 年。[17] C. Byun、W. Arcand、D. Bestor、B. Bergeron、V. Gadepally、M. Houle、M. Hubbell、H. Jananthan、M. Jones、A. Klein、等，“pPython 性能研究”，载于*2023* 年 *IEEE* 高性能极限计算会议 *(HPEC)*，第 1–7 页，IEEE，2023 年。[18] J. Kepner，面向多核和多节点计算机的并行 *MATLAB*。工业与应用数学学会，2009 年。[19] A. Reuther、J. Kepner、C. Byun、S. Samsi、W. Arcand、D. Bestor、B. Bergeron、V. Gadepally、M. Houle、M. Hubbell、M. Jones、A. Klein、L. Milechin、J. Mullen、A. Prout、A. Rosa、C. Yee 和 P. Michaleas，“面向机器学习和数据分析的 40,000 核交互式超级计算”，载于 *2018 IEEE* 高性能极限计算会议 *(HPEC)*，第 1–6 页，2018 年。