本 科 毕 业 论 文（设 计）

基于 Python 语言的并行技术在活体细胞影像定量分析中的应用

学生姓名 程明宇 学号 21160001011

指导教师 张临杰

院、系、中心 数学科学学院

专业年级 信息与计算科学 2021 级

论文答辩日期 年 月 日

中 国 海 洋 大 学

基于 Python 语言的并行技术在活体细胞影像定量分析中的应用

完成日期：_____

指导教师签字：_____

答辩小组成员签字：_____

_____

_____

_____

_____

基于 Python 语言的并行技术在活体细胞影像定量分析中的应用

摘 要

随着细胞成像技术的发展，细胞图像数据的规模不断增大，要求图像分析工具具有更高的计算效率和处理速度。Epi-Quant 作为一款开源工具，提供高效的图像分割、细胞分类和追踪功能，广泛应用于细胞动力学研究。然而，随着数据量的增加，Epi-Quant 在大规模数据处理时面临计算效率和资源利用率的瓶颈。

为解决这一问题，本文采用多线程技术对 Epi-Quant 工具包进行优化，重点对细胞特征计算、细胞分类和细胞追踪模块进行并行化改进。通过并行化细胞特征提取、轮廓分割和帧间匹配任务，显著提高了计算效率和实时追踪能力。

优化后的平台在处理大规模数据集时展现出显著的性能提升，能够更好地满足高通量细胞图像数据分析的需求，提供更高效的工具支持，为细胞动力学及相关领域的研究提供了有效的技术支持。

关键词：细胞图像分析，并行计算，多线程，细胞特征计算，细胞分类，细胞追踪

Application of Python-based Parallel Computing Techniques in Live Cell Imaging Quantitative Analysis

Abstract

With the advancement of cell imaging technologies, the volume of cell image data is growing rapidly, demanding higher computational efficiency and processing speed from image analysis tools. Epi-Quant, an open-source tool, offers efficient image segmentation, cell classification, and tracking functions, widely used in cell dynamics research. However, as data volumes increase, Epi-Quant faces performance bottlenecks in processing large-scale datasets.

To address this, this paper optimizes the Epi-Quant tool using multi-threading techniques, focusing on parallelizing key tasks such as cell feature calculation, cell classification, and cell tracking. By parallelizing the extraction of cell features, contour segmentation, and frame-to-frame matching, computational efficiency and real-time tracking capability are significantly improved.

The optimized platform shows significant performance improvements when processing large datasets, better meeting the demands of high-throughput cell image data analysis, and providing more efficient tool support for research in cell dynamics and related fields.

Keywords: Cell image analysis, Parallel computing, Multi-threading, Cell feature calculation, Cell classification, Cell tracking

目 录

1. 引言

1.1 课题研究背景及意义

近年来，随着活细胞成像与单细胞测序等技术的迅速发展，海量的细胞图像数据不断涌现，对图像分析工具的性能提出了更高要求。同时涌现出很多基于深度学习算法的细胞图像处理工具[1]，针对上皮细胞的量化分析，Epi-Quant 凭借结合 Cellpose 的高效分割能力、自主开发的基于主成分分析（PCA）与 K-means 聚类的细胞分类模块，以及基于差异度矩阵与匈牙利算法的追踪方法，成为细胞动力学研究中的重要工具[2]。Epi-Quant 能够对形态异质性的上皮细胞群体进行精准分类，并支持多组别的定量比较分析，在细胞发育、再生医学、疾病模型研究中具有广泛应用前景。

随着图像数据规模的不断扩大，Epi-Quant 在处理大规模数据集时，仍存在处理速度有限、计算资源利用率不足等问题。这不仅制约了其在高通量实验数据分析中的应用效率，也影响了对细胞行为细微动态变化的实时捕捉与深入解析。已有研究表明，传统串行处理方式在处理百万级细胞图像数据时，难以满足实时分析与交互式操作的需求。

并行计算技术，尤其是多核 CPU 并行（例如 OpenMP, multiprocessing）与 GPU 加速（例如 CUDA, TensorRT）技术[3][4]，已广泛应用于图像处理、深度学习和生物图像分析领域，能够显著提升程序执行速度与计算资源利用率。在细胞图像领域，Cellpose、DeepCell 等工具已经展示了 GPU 加速在大规模图像分割任务中的巨大潜力[5]。

将并行技术系统性引入 Epi-Quant，针对图像分割、细胞分类、细胞追踪等核心模块进行深度优化，不仅能够缩短整体计算时间，提高大规模数据处理能力，还能够扩展 Epi-Quant 在复杂实验场景中的适用性与竞争力，进一步推动细胞动力学领域的研究进展，提升生物医学基础研究与临床应用的效率与深度。

1.2 国内外研究现状

国内外在细胞图像并行处理领域的研究正呈现出蓬勃发展的态势，其核心目标在于借助高性能计算资源，有效加速对细胞图像的分析进程，以应对日益增长

的数据量和分析复杂性。[6] 目前，国内外的研究均在积极探索前沿技术与方法，并取得了一定的进展。

在国际上，多线程技术已被广泛应用于细胞图像分析的各个环节。例如，Hi-TIPS 平台集成了先进的图像处理和机器学习算法，实现了细胞和细胞核的自动分割、信号检测和追踪等功能，显著提升了大规模图像数据的处理效率[7]。此外，Cecelia 工具箱通过整合多个开源软件包，提供了一个统一的数据管理平台，使非专业人员也能进行定量的多维图像分析，推动了多线程技术在图像分析中的普及[8]。在算法层面，Goldberg 等人开发的增量广度优先搜索算法在图像分割中表现出更高的速度和鲁棒性，展示了多线程算法在提升图像处理性能方面的潜力[9]。

在国内，研究者们也在积极探索多线程技术在细胞图像处理中的应用。清华大学戴琼海教授团队开发的 DeepCAD-RT 系统是国内在该领域的代表性成果之一。该系统通过引入多线程调度机制，实现了与显微镜硬件的深度融合，显著提升了荧光图像的实时处理能力。在神经元钙成像、免疫细胞迁移等多种生命过程的观测任务中，DeepCAD-RT 展现出强大的去噪性能，推动了高通量细胞图像分析的实时化进程[10]。这些研究成果表明，国内在多线程图像处理技术方面取得了积极进展，尤其在提高图像处理效率和精度方面展现出显著优势。未来，随着计算资源的进一步优化和算法的持续改进，多线程技术将在细胞图像处理领域发挥更加重要的作用。

综上所述，国内外在多线程技术应用于细胞图像并行处理方面均取得了显著成果。国际上，研究重点集中在工具平台的开发和算法优化上，而国内则在具体应用和方法实现方面取得了实质性进展。未来，随着图像数据规模的进一步扩大和分析需求的多样化，结合多线程技术与人工智能、深度学习等先进方法，将为细胞图像分析提供更强大的技术支持，推动生物医学研究的深入发展。

1.3 研究内容

本文的主要研究内容是在 Epi-Quant 平台中应用 Python 并行计算技术，以提高细胞图像分析任务的计算效率。首先，本文将详细介绍 Python 并行计算的相关原理，包括多线程和多进程的工作机制。接着，介绍 Epi-Quant 平台的整体流程，重点讲解细胞特征提取、细胞分类和细胞追踪的计算步骤，并进行详细的耗时分析，为后续并行化设计提供依据。然后，本文着重于 Epi-Quant 的并行化设计，

首先进行了可行性分析，指出哪些任务适合并行化，接着对细胞特征提取、细胞分类和细胞追踪部分的并行化进行了详细设计，并讨论了任务拆分和负载均衡问题，确保并行计算的高效性和正确性。第五章展示了并行计算实验的结果，包括实验目的、实验环境、实验数据、实验参数设置以及结果展示等内容。重点分析了各部分任务的加速比和并行效率，并对结果进行了深入讨论。最后，第六章总结了本文的研究成果，并对未来在细胞图像分析领域应用并行计算的研究方向进行了展望和讨论。

2. Epi-Quant 平台简述

2.1 流程简述

Epi-Quant 平台利用 Cellpose 实现高效的图像分割功能，为细胞形态的精确分析提供基础。Cellpose 是一种强大的深度学习工具，能够自动识别和分割显微镜图像中的细胞结构。在初次使用时，Epi-Quant 会通过 Cellpose 处理输入的细胞图像，并生成包含分割信息的 npy 文件作为后续分析的基础。

在完成图像分割后，Epi-Quant 采用主成分分析法（PCA）与 K-means 聚类算法对细胞进行分类。首先，PCA 被用来提取细胞形态的主要特征，简化数据维度的同时保留关键信息。然后，通过 K-means 聚类算法根据这些特征对细胞进行分组。分类完成后，Epi-Quant 会生成逐帧细胞分类结果的可视化图，并输出相关数据文件，包括逐帧细胞分类结果文件、PCA 主成分贡献度文件等。

为了实现精准的细胞追踪，Epi-Quant 计算细胞间的差异度矩阵以评估相似性，并结合匈牙利算法来确定同一细胞在不同时间点的最佳匹配。这种方法确保了细胞追踪过程的高效性和准确性。此外，Epi-Quant 还能生成细胞位移矢量图和均方位移（Mean Squared Displacement, MSD）箱型图，有助于深入理解细胞的运动模式和扩散行为。通过这一系列步骤，Epi-Quant 提供了一套完整的工具集，用于从图像分割到细胞动态变化的全面分析。

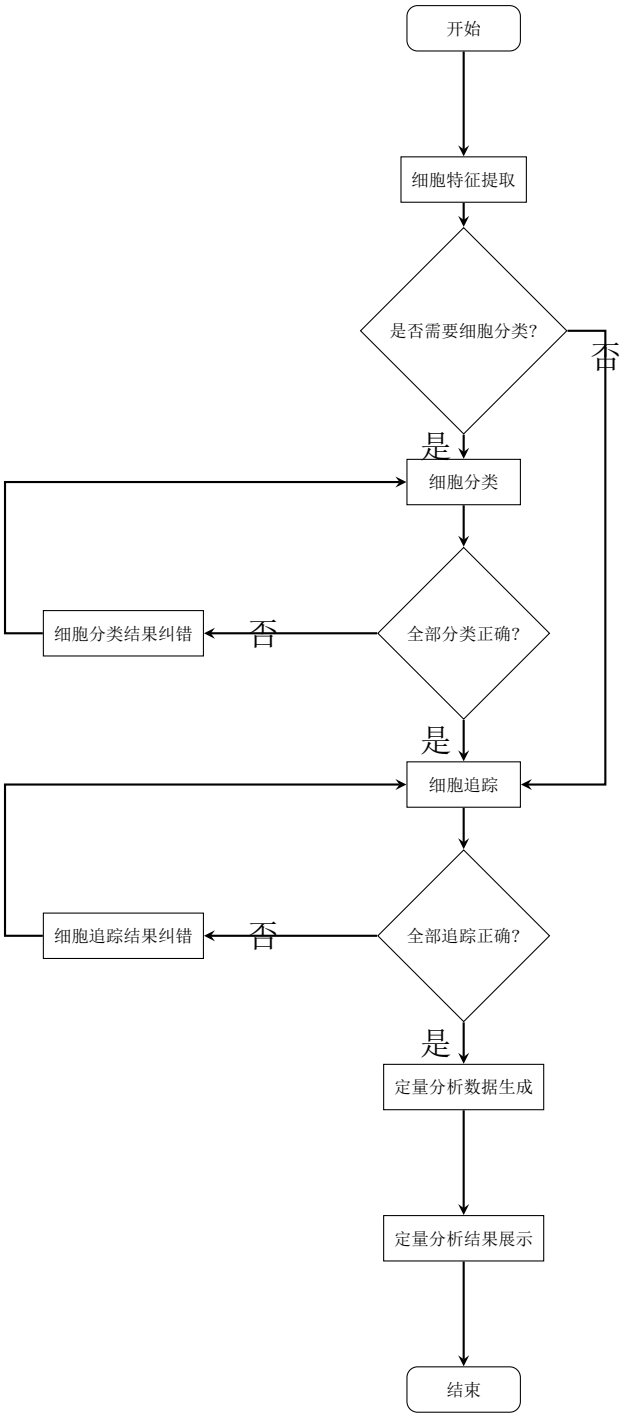

图 2-1: Epi-Quant 平台工作流程

2.2 耗时分析

为了全面评估细胞特征提取、细胞分类和细胞追踪三个模块的性能表现，本文分别对每个流程进行了五次独立测试，并取其平均值作为最终结果。这一方法能够有效减小由于系统环境波动或随机因素带来的误差，提高实验数据的稳定性与可靠性，从而为后续分析提供更具代表性的参考依据。同时，多次实验也有助于观察各阶段运行时间的一致性，验证程序执行的稳定性，增强整体实验结论的科学性与说服力。将所得数据列为下表：

表 2-1: 实验时间统计

实验次数	细胞特征提取（秒）	细胞分类（秒）	细胞追踪（秒）
第 1 次	28.8228	45.9401	114.5828
第 2 次	25.8424	41.7089	106.1937
第 3 次	24.9972	41.3358	106.9872
第 4 次	26.5087	41.5460	104.9517
第 5 次	27.0539	43.1250	107.0588
平均值	26.6450	42.7312	107.9548

可以看出，在串行的程序设计下，各部分耗时均有着较大的提升空间。

对于细胞特征提取部分，将代码中的各个部分时间作进一步的记录，随机选取执行一次细胞特征提取任务，将时间记录为下表:

表 2-2: 细胞特征提取-各步骤耗时统计

步骤	耗时（秒）	占比（%）
文件排序时间	0.0003	0.001
数据加载时间	0.5266	2.007
UI 更新时间	0.0717	0.273
特征计算时间	24.7285	94.251
DataFrame 合并时间	0.0276	0.105
Excel 保存时间	0.6098	2.323
总执行时间	26.2361	100.000

可以看出，细胞特征提取时间中占比部分大的主要是特征提取部分，涉及到对于细胞面积以及其他形态学特征的计算，串行会让计算过程变得较为缓慢，本文将代码运行改进的重点放在特征提取的部分，通过并行技术将计算效率进一步提升，达到理想的效果，为后续该工具的拓展应用奠定基础。

对于细胞分类部分，将代码中的各个部分时间作进一步的记录，随机选取执

行一次细胞分类任务，将时间记录为下表：

表 2-3: 细胞分类-各步骤耗时统计

步骤	耗时（秒）	占比（%）
PCA 执行耗时	1.3139	2.735
K-means 执行耗时	1.6279	3.389
图像处理耗时	44.9853	93.646
总执行耗时	48.0328	100.000

根据细胞分类过程中各阶段的耗时分析，图像处理作为整个流程中最耗时的部分主要集中在 update_image 函数中。该函数执行了多项任务，包括从.npy 文件加载数据、依据帧号匹配并加载相应的图像文件、对图像进行预处理（如转换为 RGB 格式和数据类型调整）、基于聚类结果对不同类型的细胞进行着色、计算相邻细胞间的邻接性以及使用 KDTree 算法计算细胞中心到边界点的最短距离等。具体来说，在一次运行中，update_image 函数总共耗时约 44.9853 秒，占总执行时间（48.0328 秒）的约 93.646%

这表明优化此函数中的操作，特别是那些计算密集型的任务（例如图像预处理和邻接性分析），将显著减少总体执行时间，进而提高细胞分类的整体效率。通过采用适当的并行化策略，如多进程处理，可以有效加速这些步骤，从而大幅缩短细胞分类所需的时间。

对于细胞追踪部分，将代码中的各个部分时间作进一步的记录，随机选取执行一次细胞追踪任务，将时间记录为下表：

表 2-4: 细胞追踪-各步骤耗时统计

步骤	时间 (秒)	占比 (%)
追踪所有细胞	77.5437	71.64
保存追踪数据	0.0769	0.07
生成对比图像	30.6142	28.29
总执行耗时	108.2348	100.00

run_cell_tracking 是整个细胞追踪流程中的核心函数，负责对给定的细胞信息（如位置、形态特征等）进行逐帧匹配与追踪。该函数首先根据细胞类型（如羊浆膜细胞或表皮细胞）划分处理逻辑，并为每一帧图像提取对应的细胞数据。随后，它通过计算当前帧与下一帧中细胞之间的多维特征距离（如面积、长宽比、中心坐标等），使用匈牙利算法（linear_sum_assignment）实现最优匹配。最终，将

所有帧间的匹配结果进行合并，生成完整的细胞运动轨迹。函数中最关键的部分是逐帧匹配细胞关系并生成追踪路径。这部分不仅决定了细胞在时间序列上的连续性，也直接影响了追踪的准确性和鲁棒性。尤其是利用多个细胞特征加权后的欧氏距离作为匹配依据，并结合匈牙利算法求解最小代价匹配，使得系统能够有效应对细胞分裂、移动、遮挡等复杂情况。由于每一组相邻帧之间的匹配操作是相对独立的（即 Frame i 与 i+1 的匹配不影响 Frame j 与 j+1 的匹配，只要没有跨帧依赖），因此可以将帧间匹配部分（即主循环内的内容）拆分为多个子任务，通过多线程或多进程方式进行并行处理。这样可以显著提升大规模数据集下的运行效率，缩短整体追踪耗时。

3. Python 并行原理

3.1 串行和并行的原理

传统的串行计算方式，是大多数计算机早期应用所采用的执行模型。在这种模式下，所有的指令都是依次进行处理的，整个程序的执行流程像是一条单行道，每次只能处理一个任务。虽然串行执行逻辑清晰、结构简单，也便于调试和维护，但它的处理能力很大程度上受限于单个处理器的性能。一旦遇到数据量庞大或任务复杂的场景，处理速度就会成为瓶颈。例如，在图像分析中，如果每个像素都要一个接一个地处理，那么处理一张高分辨率图片往往需要较长时间，这在对实时性要求较高的应用中就显得尤为不便。与之相比，并行计算的出现为提高计算效率提供了新的思路。它的基本原理是将一个大任务拆分成若干个小任务，分派给多个计算单元同时进行处理。这样，原本需要数分钟才能完成的任务，可能在几秒钟内就可以完成。以图像处理为例，如果将每个像素的处理任务分配给不同的线程或 GPU 核心，那么整个图像就可以在短时间内高效完成分析。近年来，随着多核 CPU 和 GPU 等硬件的发展，并行计算在科学研究、金融建模、人工智能等多个领域得到了广泛应用。不过需要注意的是，并行程序的开发并不总是简单的。在设计并行程序时，必须仔细考虑任务划分是否合理、各处理单元之间是否存在数据依赖关系，以及如何进行有效的同步与通信等问题[11]

3.2 多线程原理

在现代计算体系中，多线程技术是一种提高程序执行效率的重要手段。所谓多线程，指的是在同一进程内部创建多个线程，每个线程独立执行不同的任务，多个线程之间共享该进程的内存空间及其他资源。相较于多进程模型，多线程由于资源共享程度高、创建与切换开销小，成为当前并发程序设计中最常用的一种方式。线程通常由操作系统调度器管理，调度器根据一定的策略（如时间片轮转、优先级调度等）将各个线程分配至处理器核心上运行，从而实现真正的并发或并行处理。尤其是在多核处理器环境下，多线程程序能够显著提升系统的整体吞吐量与响应速度。

多线程程序的运行机制包括线程的创建、调度、执行和终止等多个环节。在程序

运行过程中，主线程可以通过系统调用或语言标准库接口（如 C++ 的 std::thread、Java 的 Thread 类等）创建新的线程。每个线程有独立的程序计数器和调用栈，但共享堆空间和全局变量。这种资源共享带来了高效的线程间通信机制，同时也使得程序更易受到资源竞争和数据不一致的影响。为了保障数据安全和程序的正确性，必须引入同步机制来协调多个线程之间对共享资源的访问。常见的线程同步工具包括互斥锁、信号量和条件变量等，它们通过控制线程之间的执行顺序和互斥访问来避免竞态条件的发生。

尽管多线程具有明显的性能优势，但其设计与实现并不简单。线程的非确定性执行顺序可能导致程序逻辑出现意料之外的结果，调试也因此变得更加困难。此外，若线程间资源申请和释放顺序处理不当，可能造成死锁；若调度策略不合理，也可能导致部分线程长期得不到执行机会，从而产生“饥饿”现象。因此，编写一个高效且可靠的多线程程序不仅要求开发者理解操作系统的调度机制，还需要具备良好的并发程序设计经验。

在实际应用中，多线程被广泛用于图像处理、科学模拟、服务器响应、并行计算等领域。例如，在图像处理任务中，可以将图像分为多个区域，由不同的线程并行处理各自区域的数据，最终合并结果。这种方式不仅能加快处理速度，还能更好地利用多核系统的并行能力。随着多核处理器和图形处理单元（GPU）的广泛应用，多线程技术也在不断演进，线程池、任务队列、异步事件驱动等机制的引入，使得多线程程序的性能与可维护性得到了进一步提升。

3.3 多线程原理工作流程

在整个程序启动时，首先接收一个待处理的任务，这个任务可能是细胞图像分析中的特征提取等操作。系统首先评估此任务是否可以被分解为多个独立的小任务，以便于并行处理。如果确定任务具有可并行化的特性，例如不同细胞之间的信息提取互不影响，则开始进行下一步。接下来，原始任务会被拆解成若干个小任务，并为每个小任务创建一个对应的线程。这些线程随后由操作系统调度器分配到不同的 CPU 核心上运行。这样做的目的是充分利用多核处理器的优势，实现真正意义上的并行计算，从而加快整个任务的处理速度。

一旦所有线程都被分配到了相应的 CPU 核心，它们便开始并发执行各自的任务。在这个阶段，由于各个子任务之间相对独立，因此可以在不同的核心上同时

进行计算，极大地提高了处理效率。

然而，在实际操作中，某些情况下多个线程可能需要访问相同的资源，比如共享变量或数据结构。为了避免这种情况导致的数据不一致或其他问题，程序会在必要时使用同步机制，如互斥锁或信号量，确保对共享资源的安全访问。在所有线程都完成自己的工作后，系统会检查每个线程的状态以确认是否全部任务已经完成。如果还有未完成的任务，主控流程将继续等待；一旦所有任务结束，系统将收集所有线程的结果并将它们整合起来。最终，汇总后的结果会被保存或者展示给用户，这标志着整个多线程处理过程的结束。通过这种方式，不仅提高了程序的执行效率，还能有效地管理和利用多核 CPU 的强大计算能力，特别适合用于加速像细胞定量分析这样的大规模数据处理任务。

3.4 多进程原理

多进程（Multiprocessing）是操作系统层面的一种并行计算方式，指的是在同一时间内由多个独立进程协同运行，各自拥有独立的内存资源和执行环境。这种机制特别适合用于处理资源密集型或任务互不依赖的计算问题，因其具备良好的隔离性和扩展性，被广泛应用于科学计算、图像处理和高性能计算等领域[12]。

在细胞图像分析方面，多进程技术因其对大规模图像数据的高效处理能力而受到关注。细胞图像的分割、配准、特征提取等流程往往具有高度的并行性，将图像划分为若干部分由不同进程独立处理，可以充分利用多核处理器的计算能力，显著减少运行时间并提升整体效率。

相比多线程技术，多进程模型在 Python 等语言中表现出更优的并发性能，主要得益于其规避了全局解释器锁（GIL）的限制。这使得它特别适用于图像处理这类 CPU 密集型任务。近年来，多进程并行框架已被集成到多个细胞图像处理工具或自定义分析脚本中，并在显微图像自动分析、生物标志物提取及细胞行为定量研究等方向中得到实际应用。

3.5 多进程工作流程

多进程（Multiprocessing）的工作流程主要包括任务划分、进程创建、任务执行以及结果合并四个阶段，是一种典型的并行计算模型。在运行程序之初，主进程会根据待处理的数据或计算任务的结构特性，将整体任务划分为若干个相互独

立的子任务。这些子任务之间没有共享状态，彼此可以独立运行，这为后续并行执行打下基础。

接下来，主进程通过操作系统接口创建多个子进程，并为每个子任务分配相应的计算资源。每个子进程在独立的内存空间中运行，不同于线程之间的共享内存结构，因此具备更强的隔离性和稳定性。这些子进程并行执行各自的任务，并在运行过程中通过进程间通信（如管道、队列或共享内存）将中间结果反馈或传输。

待所有子进程完成各自任务之后，主进程将接收并整合来自各子进程的结果。这一阶段需要处理进程同步、结果排序与合并等操作，以确保最终输出的完整性和一致性。整个流程自始至终由主进程统一调度和管理，必要时还可动态调整子进程的数量，以适应不同硬件平台的并行能力。

Algorithm 1 多线程任务处理算法

输入：一个待处理的任务
输出：处理后的结果（保存或展示）

```
接收待处理的任务
if 任务可并行化 then
    分解为多个小任务
    创建线程和分配 CPU 核心
    并发执行子任务
    使用同步机制管理共享资源
else
    直接执行任务
end if
while 不是所有线程已完成 do
    监视线程执行情况
    if 所有线程已完成 then
        整合所有线程的结果
        跳出循环
    else
        继续执行剩余子任务
    end if
end while
保存或展示最终结果
```

图 3-1: 多线程任务处理算法

Algorithm 2 多进程任务处理算法

输入：一个待处理的任务
输出：处理后的结果（保存或展示）

```
接收待处理的任务
if 任务可并行化 then
    分解为多个子任务
    创建多个进程并分配计算资源
    启动并行进程执行子任务
else
    直接执行任务
end if
while 不是所有进程已完成 do
    监视进程执行情况
    if 所有进程已完成 then
        整合所有进程的结果
        跳出循环
    else
        继续执行剩余子任务
    end if
end while
保存或展示最终结果
```

图 3-2: 多进程任务处理算法

4. Epi-Quant 并行化设计

4.1 可行性分析

在 Epi-Quant 中，细胞特征计算、细胞分类的细胞轮廓提取，以及细胞追踪中的帧间匹配任务，都需要处理大量计算数据。随着数据集的扩大，传统的串行计算方式在执行这些任务时逐渐变得不够高效，特别是当数据量非常大时，处理时间会显著增加，影响整体分析效率。因此，采用并行计算来优化这些步骤，不仅可以提升计算速度，还能更好地利用现有的计算资源[13]。首先，细胞特征计算是整个工具包中的核心任务之一，涉及从细胞轮廓中提取各种形态学特征（比如面积、周长等）。如果按传统串行方式，每次只能处理一个细胞，计算进度会比较缓慢。而引入并行计算后，细胞的特征提取可以分配给不同的计算单元，彼此独立地处理每个细胞，这样可以显著加快整个计算的速度。其次，细胞分类过程中，轮廓提取同样是关键。传统的方法一般需要逐图像处理，这对于大数据集来说极其低效。通过并行计算，可以把多个图像或者细胞的轮廓提取任务分配到多个计算单元上并行处理，从而加快整个过程。在处理具有较大形态差异的细胞时，并行计算特别有效，因为它能够快速处理不同细胞的特征，从而保持分类的准确性和效率。最后，细胞追踪中的帧间匹配是一个具有挑战性的任务，涉及到时间序列的匹配。由于每帧的匹配任务独立性较强，因此非常适合并行计算。通过将帧间的匹配任务拆解，多个计算单元可以同时执行任务，这样不仅提升了匹配速度，也保证了长时间追踪过程中动态变化的及时捕捉。并行计算的引入，使得 Epi-Quant 在图像处理中的整体效率得到了显著提升。通过将计算任务拆解并分配到多个处理单元，可以更好地利用 CPU 或 GPU 的多核能力。在处理大规模数据时，系统能够更智能地分配任务，避免某些计算单元负荷过重，从而最大化计算效率。而且并行计算大大降低了串行计算中的瓶颈，能加快数据处理的速度，进一步缩短了整体计算的时间。

4.2 并行程序设计

4.2.1 细胞特征提取部分的并行化

细胞特征计算步骤采用了基于线程池的并行处理策略，以提高细胞信息提取的效率。具体而言，在对每个输入的.npy 文件所包含的细胞区域进行处理时，系统将所有独立细胞（由 unique_ids 标识）分配至多个并发线程中，每个线程独立执行一个细胞的特征计算任务。该任务包括细胞边界识别、面积与周长计算、最大水平与垂直长度分析、拟合椭圆参数提取、多边形近似顶点数统计、以及圆度与 P 值等形态学特征的推导。通过 ThreadPoolExecutor 实现的多线程机制，多个细胞的特征计算过程得以并发执行，从而显著缩短了整体数据处理的时间，提升了系统的响应速度与运算效率。各线程完成计算后，其结果被统一收集至共享列表中，为后续的数据整合与输出提供支持。

4.2.2 细胞分类部分的并行化

在细胞分类过程中，并行处理被用于加速对大量细胞轮廓信息的分析和处理。具体来说，当一个新帧图像被加载后，系统首先识别出该帧中所有独特的细胞 ID。针对每个细胞 ID，系统会启动一个独立的任务或线程，这些任务并发地执行细胞处理函数 Process_Cell。在此函数中，每个细胞根据其聚类标签（如大细胞或小细胞）被分配特定的颜色标记，并且该细胞的轮廓点被提取并保存到相应的集合中（例如，蓝色细胞轮廓集或黄色细胞轮廓集）。通过利用线程池来管理这些并发任务，程序可以同时处理多个细胞的数据，从而显著提升整体处理效率。一旦所有线程完成其执行，后续步骤包括计算相邻细胞之间的最短距离等进一步分析也会基于这些初步结果进行。这种并行策略确保了即使面对高通量数据时，细胞分类和分析过程也能高效、准确地完成

4.2.3 细胞追踪部分的并行化

首先，程序提取所有有效帧号，并通过进程池机制（ProcessPoolExecutor）对相邻帧之间的细胞配对任务并行处理。每个进程独立完成一组帧之间的细胞对应计算，去除异常匹配并输出结果文件。匹配过程基于几何特征差异，确保配对的准确性。

随后，采用线程池（ThreadPoolExecutor）对生成的匹配文件并行读取。程序依次将各帧之间的匹配结果组合成跨帧追踪表，逐步构建细胞在时序图像中的连续轨迹。最终，整合结果被统一保存为 Excel 文件。

通过在不同阶段采用进程与线程并行机制，此方法在保证追踪精度的同时，显著缩短了处理大批量图像数据的时间。

Algorithm 3 多线程细胞信息提取算法

输入： 包含细胞轮廓的.npy 文件路径
输出： 提取后的细胞特征数据（Excel 文件）

```
获取所有.npy 文件并按编号排序
for 每个文件 do
    加载图像数据并获取细胞 ID 列表
    初始化用于存储结果的列表
    启动线程池
    for all 每个细胞 ID do
        计算细胞边界点坐标
        if 排除边缘细胞且该细胞位于边缘 then
            跳过该细胞
        else
            填充轮廓区域并计算面积
            查找轮廓并计算周长
            拟合椭圆，获取长短轴及角度
            计算最小外接圆和最大内接圆半径
            计算圆度和 P 值
            收集细胞中心坐标与边界信息
            将结果组织为 DataFrame 并返回
        end if
    end for
    收集所有线程的结果并合并
end for
将最终结果保存为 Excel 文件
```

图 4-1: 多线程细胞信息提取算法流程

Algorithm 4 多线程细胞分类算法流程

输入：标注信息文件路径、图像路径、分类结果、输出目录
输出：分类后并标注的图像结果

```
读取当前帧的标注文件并提取相关数据
if 图像未嵌入标注文件 then
    自动查找并载入对应图像
end if
初始化用于存储不同类别轮廓的容器
提取所有单元编号
启动线程池
for all 每个单元编号 do
    提交处理任务以判断类别并提取轮廓
    if 编号为背景或无效 then
        跳过
    else if 属于第一类 then
        存储轮廓至对应容器
        着色并可视化该区域
    else if 属于第二类 then
        同上操作，使用不同颜色
    end if
end for
等待所有线程完成处理
后处理分析：寻找配对关系与几何关系
计算指定边缘之间的距离
展示并保存结果图像
```

图 4-2: 多线程细胞分类算法流程

Algorithm 5 多进程细胞追踪算法

输入： 所有帧的细胞信息 cells_info，进度范围 progress_start 至 progress_end
输出： 合并后的细胞追踪表格（Excel 文件）

统一 cells_info 的索引为字符串格式
根据 leading_edge 字段内容，确定细胞类型与输出路径
if 需要读取已有细胞追踪结果 **then**
    获取现有结果中最大编号，作为新的起始索引
end if
初始化输出目录，创建输出路径
提取所有帧编号 fig_ids，并排序
if 帧编号为空 **then**
    报错并终止
end if
初始化 all_matching_results 列表
{第一阶段：使用多进程并行计算帧间匹配}
启动进程池
for 每个 fig_id **do**
    将匹配任务提交给进程池执行 parallel_frame_processing(fig_id)
end for
for 每个完成的任务 **do**
    获取 fig_id 与匹配结果 matching_result
    为结果添加唯一编号
    去除匹配误差等于最大阈值的行
    保存匹配结果至对应 Excel 文件
    累加结果并更新索引
end for
{第二阶段：主线程更新进度条}
for 每个 fig_id **do**
    根据已完成帧数更新进度条显示
end for
{第三阶段：使用多线程读取所有匹配文件并整合}
获取所有匹配结果文件
启动线程池
for 每个文件名 **do**
    并行读取并解析文件内容为 DataFrame
end for
按帧序整合结果，构建完整追踪链
将最终结果保存为 Excel 文件

图 4-3: 多进程细胞追踪算法流程

5. 并行计算实验

在本章中，本文将完成并行计算数据实验，对实验结果进行分析，评估并行算法的性能和效果，为进一步的研究和应用提供参考和指导。

5.1 实验目的

本研究的实验旨在基于多进程与多线程相结合的策略，对 Epi-Quant 工具进行并行化优化，以解决其在处理大规模细胞图像数据时存在的效率瓶颈问题。

本实验的核心目标是通过将任务细化为可并发执行的单元，并在不同层级引入多线程（如在特征提取或图像处理中的像素级并行）与多进程（如在多个细胞区域或图像帧之间的独立计算）相结合的调度机制，充分利用多核 CPU 资源，加快处理速度。在此基础上，实验将系统评估优化前后的性能差异，重点考察任务运行时间，验证并行机制对整体流程的加速比表现。

5.2 实验环境

表 5-1 是基于 DirectX 诊断工具给出的系统参数以及 Intel 官网给出的规格参数：

表 5-1: Intel Core i9-13900H 处理器规格

参数	规格说明
核心总数	14
性能核（P-core）数量	6
能效核（E-core）数量	8
线程总数	20
最大睿频频率	5.40 GHz
睿频加速 Max 技术 3.0 频率	5.40 GHz
性能核最大睿频频率	5.40 GHz
能效核最大睿频频率	4.10 GHz
缓存	24 MB Intel® Smart Cache
处理器基础功耗	45 W
最大睿频功耗	115 W
最小保证功率	35 W

5.3 实验结果

5.3.1 细胞特征提取并行化结果

将并行化的程序在 Python 平台运行 5 次，取平均值，记录为下表：

表 5-2: 并行化细胞特征提取平均耗时统计

步骤	平均耗时（秒）
文件排序时间	0.0002
加载时间	0.7621
处理时间	9.7935
UI 更新时间	0.1829
DataFrame 合并时间	0.0384
Excel 保存时间	0.7020
总时间	12.0433

已知并行效率的计算公式[14][15] 为：

$$\text{加速比} = \frac{\text{串行时间}}{\text{并行时间}} \tag{5-1}$$

$$\text{并行效率} = \frac{\text{加速比}}{\text{线程/进程数}} \tag{5-2}$$

将原代码所记录的时间记录为串行时间，由（5-1）和（5-2）计算得出：

表 5-3: 加速比与并行效率

指标	结果
加速比（Speedup）	2.179
并行效率（Efficiency）	1.089

5.3.2 细胞分类并行化结果

将并行化的程序在 Python 平台运行 5 次，取平均值，记录为下表：

表 5-4: 并行化细胞分类平均耗时统计

步骤	耗时（秒）
PCA 执行耗时	1.6210
K-means 执行耗时	1.6701
图像处理耗时	28.4078
总执行耗时	31.7604

将原代码所记录的时间记录为串行时间，由（5-1）和（5-2）计算得出：

表 5-5: 加速比与并行效率

指标	结果
加速比（Speedup）	1.51
并行效率（Efficiency）	0.755

5.3.3 细胞追踪并行化结果

将并行化的程序在 Python 平台运行 5 次，取平均值，记录为下表：

表 5-6: 并行化细胞追踪平均耗时统计

步骤	时间（秒）	占比（%）
追踪所有细胞	46.5262	60.24
保存追踪数据	0.0769	0.10
生成对比图像	30.6142	39.64
总执行耗时	77.2173	100.00

将原代码所记录的时间记录为串行时间，由（5-1）和（5-2）计算得出：

表 5-7: 加速比与并行效率

指标	结果
加速比（Speedup）	1.401
并行效率（Efficiency）	0.701

5.4 结果分析

将三个部分的并行效果进行汇总，记录为下表：从数据中可以看出：

表 5-8: 细胞图像处理各阶段加速比与并行效率汇总

阶段	串行时间（秒）	并行时间（秒）	加速比 / 并行效率
特征提取	26.2361	12.0433	2.179 / 1.089
细胞分类	48.0328	31.7604	1.512 / 0.756
细胞追踪	108.2348	77.2173	1.401 / 0.701

细胞特征提取阶段的并行优化效果最为显著。串行执行时间为 26.2361 秒，并行优化后降为 12.0433 秒，对应加速比为 2.179，效率为 1.089。这一结果甚至略超线性加速，可能得益于任务划分精细、内存访问局部性良好，以及多线程带来的缓存命中率提升。说明该阶段具备高度可并行性，且资源调度效率较高。

细胞分类阶段的加速效果相对一般，串行时间为 48.0328 秒，并行时间为 31.7604 秒，加速比 1.512，并行效率 0.756。该阶段涉及的 PCA 和 K-means 操作具有一定计算强度，但部分处理过程可能存在串行瓶颈，影响整体加速效果。任务划分和线程负载均衡仍有优化空间。

在细胞追踪阶段，通过对帧间匹配任务进行并行化处理，使总耗时由串行的 108.2348 秒减少至并行的 77.2173 秒，获得了 1.401 的加速比和 0.701 的并行效率。该阶段的性能提升主要来源于对不同帧之间细胞轮廓比对的并行处理，显著压缩了核心计算时间。然而，由于帧与帧之间仍存在一定的数据依赖关系，限制了并行深度，加之其他步骤（如图像生成）未实现并行，整体加速效果受到制约。因此，尽管当前优化取得了一定成效，仍有进一步提升并行效率的空间。

5.5 实验结论与展望

本研究采用了并行计算对 Epi-Quant 工具包中的细胞特征提取、细胞分类和细胞追踪等任务进行了优化。实验结果表明，细胞特征提取阶段通过并行化处理

显著提升了计算效率，尤其在特征计算步骤中取得了显著加速。这表明特征提取的计算任务具有较强的并行化潜力，能够有效提高整体处理速度。

在细胞分类阶段，尽管并行计算提升了部分计算效率，但由于 PCA 和 K-means 算法的迭代过程较为复杂，导致加速效果不如细胞特征提取阶段明显。这也表明，对于一些依赖性较强的计算任务，虽然并行化能加速部分处理过程，但整体效果仍有改进的空间。

在细胞追踪阶段，通过并行计算优化了帧间匹配的过程。尽管存在较大的数据依赖性，合理的线程分配依然提高了计算速度，并增强了处理大规模图像数据的能力。尽管加速比有限，但并行计算在稳定性和效率提升方面仍然发挥了重要作用。

未来的工作可以考虑进一步优化并行化策略，特别是针对迭代依赖较强的算法，可能需要改进算法或采用更高效的并行方法。随着细胞图像分析规模的扩大，探索更高效的硬件加速方法（如 GPU 或分布式计算）也是提高计算效率的重要方向。此外，结合智能资源调度和任务划分策略，进一步提升系统的整体性能，预期能为大规模细胞图像数据的处理提供更高效的解决方案。

参考文献

[1] Moen E, Bannon D, Kudo T, et al. Deep learning for cellular image analysis. *Nature Methods*, 2019, 16(12): 1233–1246. doi:10.1038/s41592-019-0403-1.

[2] Stringer C, Wang T, Michaelos M, et al. Cellpose: a generalist algorithm for cellular segmentation. *Nature Methods*, 2021, 18(1): 100–106. doi:10.1038/s41592-020-01018-x.

[3] Alda Kika, Silvana Greca. Multithreading Image Processing in Single-core and Multi-core CPU using Java. *International Journal of Advanced Computer Science and Applications*, 2013, 4(9): 174–180. doi:10.14569/IJACSA.2013.040926.

[4] Dilla D S, Pustovalov E V, Artemeva I L. Applying GPU parallel programming for image processing and clustering. *Computational Nanotechnology*, 2024, 11(4): 77–86. doi:10.33693/2313-223X-2024-11-4-77-86.

[5] Bannon D, Moen E, Schwartz M, et al. DeepCell Kiosk: scaling deep learning-enabled cellular image analysis with Kubernetes. *Nature Methods*, 2021, 18(1): 43–45. Erratum in: Nat Methods, 2021, 18(2): 219. doi:10.1038/s41592-020-01023-0.

[6] Zuvieta H, Torres Cruz F. Efficiency of Image Processing with Parallel Techniques. [EB/OL] 2024. Preprint at: arXiv:doi:10.20944/preprints202408.0178.v1.

[7] Keikhosravi A, Almansour F, Bohrer C H, et al. High-throughput image processing software for the study of nuclear architecture and gene expression. *Sci Rep*, 2024, 14: 18426. doi:10.1038/s41598-024-66600-1.

[8] Schienstock D, Hor J L, Devi S, et al. Cecelia: a multifunctional image analysis toolbox for decoding spatial cellular interactions and behaviour. *Nat Commun*, 2025, 16: 1931. doi:10.1038/s41467-025-57193-y.

[9] Bokhari SH, Çatalyürek ÜV, Gurcan MN. Massively Multithreaded Maxflow for Image Segmentation on the Cray XMT-2. *Concurr Comput*, 2014, 26(18): 2836-2855. doi:10.1002/cpe.3181. PMID: 25598745; PMCID: PMC4295505.

[10] Li, X., Li, Y., Zhou, Y. et al. Real-time denoising enables high-sensitivity fluorescence time-lapse imaging beyond the shot-noise limit. *Nat Biotechnol*, 41, 282–292 (2023). doi:10.1038/s41587-022-01450-8.

[11] Grama A, Gupta A, Karypis G, et al. *Introduction to Parallel Computing* (2nd ed.). Addison-Wesley, 2.

[12] Baer J L. A survey of some theoretical aspects of multiprocessing. *ACM Computing Surveys (CSUR)*, 1973, 5(1): 31-80.

[13] Rathod A B, Khadse R, Bagwan M F. Serial computing vs. parallel computing: A comparative study using MATLAB. *International Journal of Computer Science and Mobile Computing*, 2014, 3(5): 815-820.

[14] Hennessy, John L. and Patterson, David A. *Computer Architecture, Fifth Edition: A Quantitative Approach* (5th ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.

[15] 陈国良, 毛睿, 蔡晔. 高性能计算及其相关新兴技术. * 深圳大学学报: 理工版 *, 2015, 32(1): 25-28. DOI: CNKI:SUN:SZDL.0.2015-01-004.

致谢

在四年大学时光的最后，我想感谢一切给予过我帮助的人，在毕业论文的完成过程中，郭雅倩学姐和张临杰老师曾为我提供了无私细致的指导，在一次次的困难挫折中，我坚信从哪里跌倒就从哪里站起来，学习数学不是一件轻松的事情，同时我也感谢我自己，坚定地选择这条并不轻松的路，并坚持到现在。

遥望未来，我将继续在计算机领域深造，愿海大园锲而不舍的学习精神与我相伴，继续在学术道路上披荆斩棘，不断战胜过去的自己。