南開大學
Nankai University

OpenMP&Pthread 多线程编程

姓名：钟坤原

学号：2212468

专业：计算机科学与技术

2024 年 5 月 26 日

目录

1 问题描述

高斯消去法是解多元线性方程组的一种常用算法。该方法通过逐行消去，将线性方程组的系数矩阵转化为上三角形矩阵，使主对角线上的元素为 1，非主对角线上的元素为 0。这一过程主要包括两个步骤：

对当前行进行处理，使得行首元素为 1，也就是将当前消元行的每个元素都除以行首元素。利用当前行将其下方各行的对应元素消去，确保当前列的下方元素为 0。例如，考虑如下的方程组：

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

经过高斯消去法处理后，方程组的系数矩阵将转化为：

$$\begin{bmatrix} 1 & a'_{12} & \cdots & a'_{1n} \\ 0 & 1 & \cdots & a'_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

本次实验，采用 pthread 和 OpenMP 多线程编程，针对上述过程进行并行优化本次实验还将同时设计 ARM 架构和 x86 架构两个平台的实验，对比在不同平台上多线程编程的性能差异。

2 实验环境

实验平台	CPU 型号	L1-L2-L3	语言
x86	i7-9750H	384KB-1.5MB-12MB	Rust
Arm	华为鲲鹏 920 处理器	48KB-1.25MB-24MB	Rust

表 1: 实验环境

3 实验设计

因为 rust 有 Thread 模块可以提供多线程支持，有 rayon 与 crossbeam 可以提供并行和并发控制支持，我选择使用 rust 来进行实验。

3.1 Pthread

高斯消去法的并行化主要涉及两个阶段：消元行的行内除法和剩余行的消元减法。除法操作由单个线程执行，而减法操作由多个线程并行执行。我选择使用信号量和 barrier 来同步不同线程之间的操作，确保所有减法操作在除法操作完成后才开始。

3.2 OpenMP

首先处理消元行的除法操作，由于这一操作规模相对较小，通常由单一线程执行以避免多线程带来的额外开销。对于消元操作（剩余行减去消元行的倍数），则由多个线程并行处理，因为这些操作彼此之间没有数据依赖，适合进行并行化。我还探索了使用 OpenMP 的 SIMD 自动向量化预编译选项来提高性能，并与手动设计的 SIMD 向量化进行性能比较。

3.3 数据划分

Pthead 实验采用块划分和动态划分。每个线程处理连续的若干行，以利用数据局部性和减少线程之间的通信开销。划分策略：考虑到缓存优化，选择按块划分数据，每个线程处理一块连续的行，以提高缓存命中率并减少访存延迟。采用动态任务划分，允许线程在完成分配任务后动态获取新任务，以保持所有线程的负载均衡，减少空闲时间。

OpenMP 探讨了不同的任务划分方式，包括静态、动态和指导式（guided）任务划分。静态划分因任务不均匀可能导致负载不均，而动态划分虽然可以提高负载均衡但可能增加调度开销。实验中采用了 OpenMP 的 schedule 参数来调整任务划分方式，优化负载均衡和性能。

3.4 不同数据规模和线程数的影响

本文探讨了两种算法不同数据规模对并行优化效果的影响，特别是在小规模数据上，多线程可能因调度开销而不及串行算法性能。本实验还考察了不同线程数量对性能的影响，以找出最优线程数配置。

4 实验结果分析

4.1 Pthread

4.1.1 X86 平台测试结果

在 x86 平台上，测量在不同任务规模下，pthread 并行优化算法对于普通串行算法和 SIMD 向量化优化算法的加速比。本实验在 SIMD 进行向量化处理的时候，采用的是四路向量化处理，而 pthread 多线程优化时，总共开启了 8 条线程，其中一条线程负责除法操作，剩余的 7 条线程负责做消元操作。本次实验使用 SSE 指令集来验证 pthread 多线程的优化效果，测量在不同问题规模下的运行时间，如下表所示。

N	serial	SSE	pthread
10	0.06	0.06	0.08
100	0.76	0.52	3.41
200	6.16	3.79	6.04
300	18.10	12.27	10.12
500	86.06	56.14	23.91
800	513.18	328.84	81.94
1000	1026.93	648.90	146.61
1500	3312.41	1947.13	432.78
2000	9349.11	6343.67	1593.84

4.1.2 X86 平台测试加速比

对于不同数据规模的计算，只做 SIMD 向量化处理和 Pthread 多线程处理结合 SIMD 向量化处理在不同数据规模的表现情况如图4.1所示

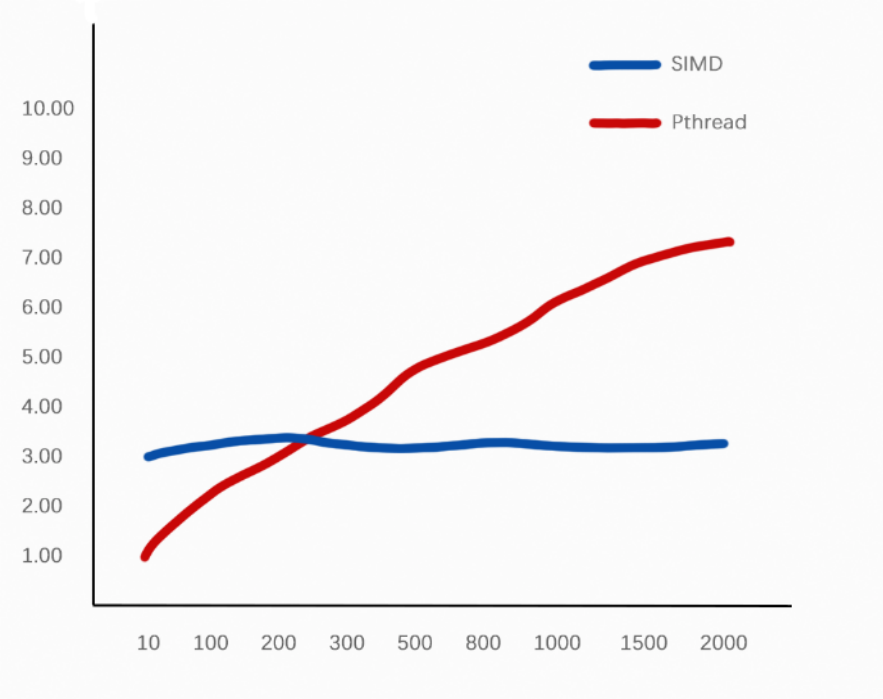

图 4.1: x86 平台随规模变化加速比

可以看出，在问题规模较小时，Pthread 加速效果并不如直接的向量化计算方法，在问题规模较大时，Pthread 的加速比会以类线性的速度增长。

4.1.3 X86 平台数据划分方式对比

本次实验在 x86 平台上分别对比了循环划分、块划分和动态划分三种方式的表现效果，表现情况如图4.2所示

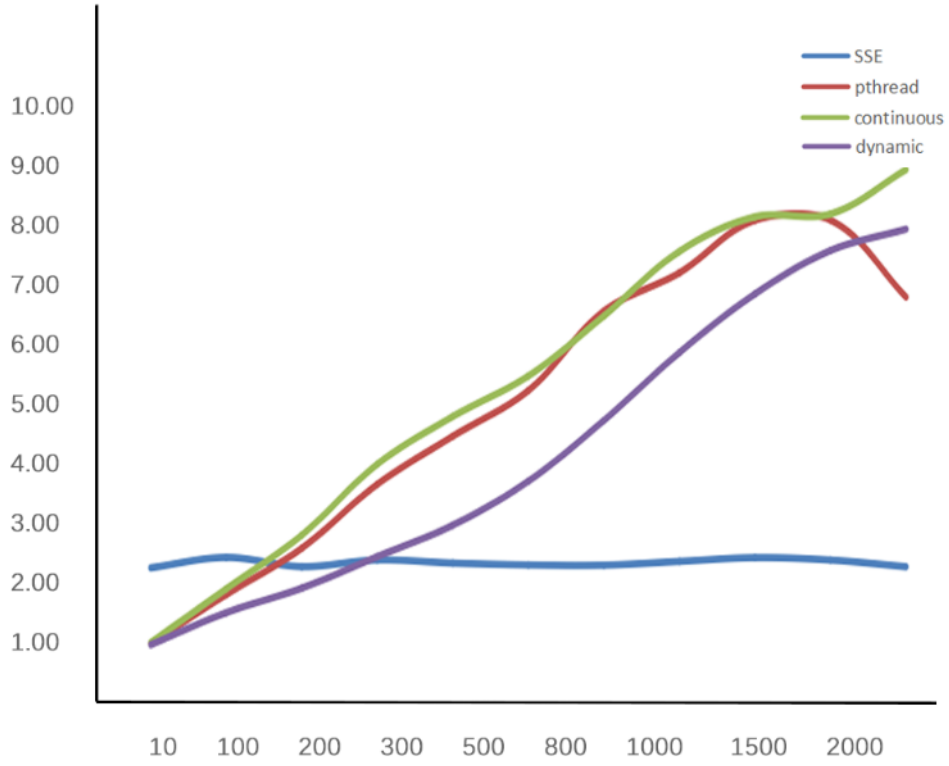

图 4.2: x86 平台数据划分方式对比

从缓存（cache）的角度来解释数据划分方法图像中观察到的性能差异，主要涉及到任务划分的不同方式对于缓存命中率的影响。按块划分的情况下，在任务开始前就决定了每个线程的工作量。这种方式通常按块划分数据，可以优化空间局部性，因为每个线程会连续处理内存中相邻的数据。这减少了跨远距离的内存访问，从而提高了缓存命中率。然而，在负载不均或数据不均匀的情况下，静态划分可能导致一些线程较早完成任务，从而等待其他线程，这降低了整体性能。动态划分的情况下，任务在运行时动态分配给线程。每当一个线程完成其任务，它会请求更多的任务。这种方式优化了时间局部性，因为每个线程总是在工作，减少了闲置时间。然而，动态划分可能导致频繁的缓存失效，因为线程可能被频繁分配到物理内存中位置较远的数据块。

在图像中观察到的性能差异主要由于不同数据划分方式对缓存命中率的影响。块划分可能显示出更好的性能，尤其是在数据规模大且均匀的情况下，因为它减少了缓存失效的频率并优化了空间局部性。动态和指导式划分虽然提高了线程利用率，但在某些情况下可能因为频繁的数据位置切换而导致较高的缓存失效率。

从负载均衡的角度出发解释数据划分方法的性能差异，主要关注如何有效地分配任务以最大限度减少线程闲置时间，并确保所有处理单元都均匀地贡献计算资源。理想的负载均衡策略能够确保所有线程或处理单元在整个计算过程中尽可能均等地忙碌，从而提高程序的总体执行效率和减少计算时间。

负载均衡的性能影响在任务相对均匀的情况下，循环划分可能表现良好，因为减少了运行时的调度开销。对于具有不同处理时间的多变任务，动态划分能够更有效地利用所有线程的计算能力，理论上应该提供更优的性能。而当问题规模提升的时候，可以发现，动态划分方式的表现已经能够超越循环划分，负载均衡带来的收益已经抵消了线程调度的额外开销。

4.1.4 不同线程数量的效果

本次实验中，还探究了 pthread 多线程优化方法，在开启不同的线程数量时，优化效果的变化情况。为了能够显著体现 pthread 的优化效果，选取数据规模为 1000，调整线程数量，观测加速比的变化情况如图4.3所示。

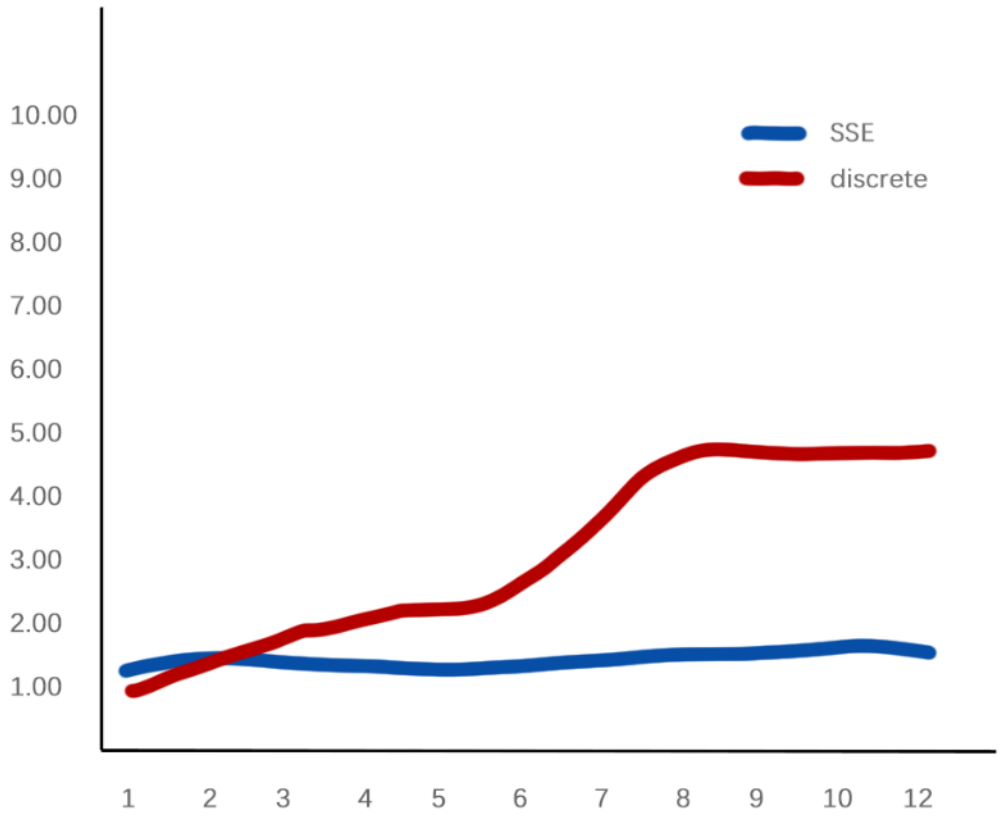

图 4.3: x86 平台优化效果随线程数变化

可以看到在线程数超过 8 时，优化效果不再提升，合理推测是因为本机是 8 核机器，线程数更高时为无效优化，暂时没有有更多内核的机器供测试这个。

4.1.5 Arm 平台测试结果

我测量在不同任务规模下 pthread 并行优化算法对于普通串行算法和 SIMD 向量化优化算法的加速比。在本次实验中，pthread 并行算法中，同样融合了 SIMD 的向量化处理。在 ARM 平台上，SIMD 的实现是基于 Neon 指令集架构的。

Arm 平台加速比如图4.4所示。

N	serial	neon	pthread
10	0.04	0.03	0.04
100	2.18	1.46	2.57
200	17.34	11.46	5.89
300	61.34	38.34	11.84
500	274.91	180.69	42.32
800	1144.60	747.47	130.17
1000	2215.65	1460.97	266.14
1500	10856.681	7158.75	505.66
2000	17641.26	11503.62	1820.84

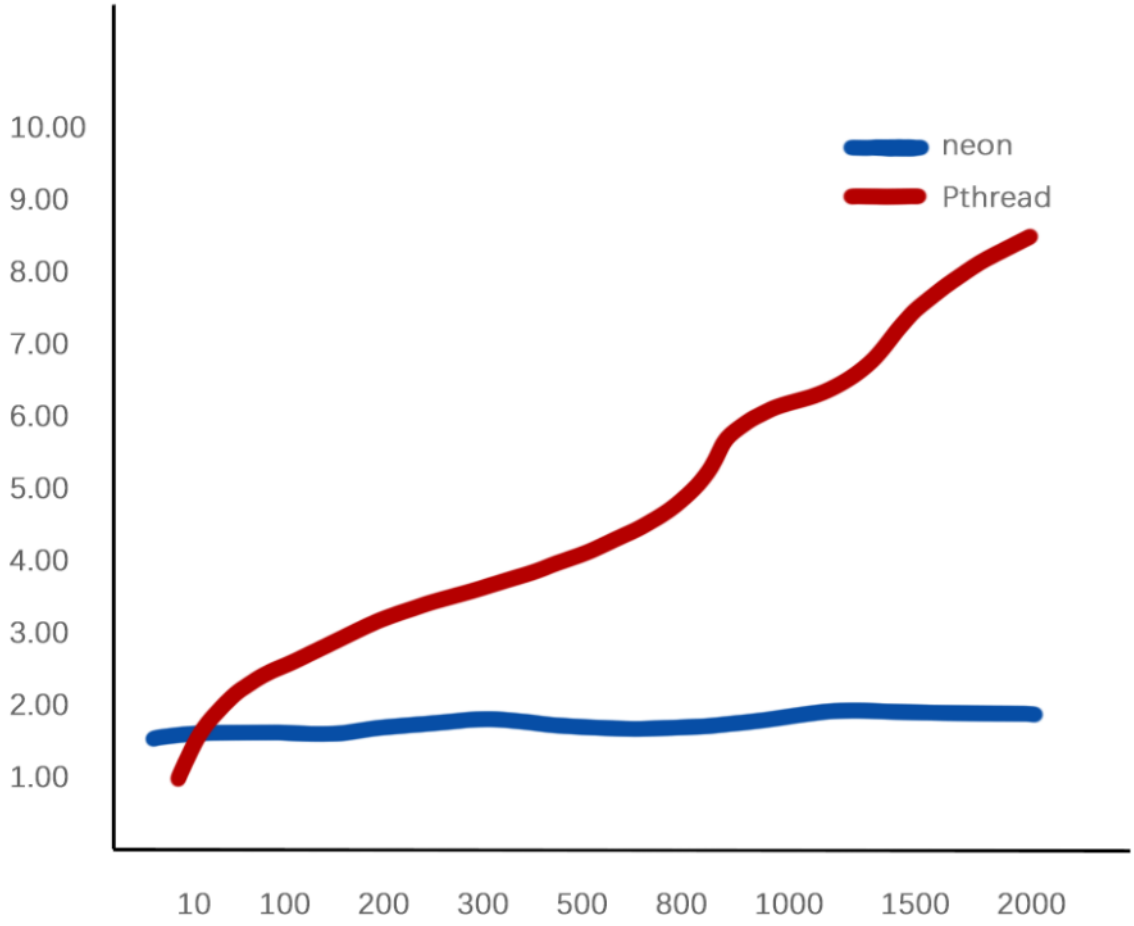

图 4.4: x86 平台优化效果随线程数变化

而从实验数据来看，当问题规模较小的时候，pthrad 多线程算法的时间性能甚至差于普通的串行算法。这是由于线程的创建，挂起，唤醒和切换等操作，所需要消耗的时钟周期数要远远多余简单的运算操作。因此当问题规模较小时，由于运算操作在整个问题求解的过程中所占比例较低，因此线程额外开销的副作用就会显现出来。而随着问题规模的增大，pthread 多线程的优势就能够显现出来。

4.1.6 Arm 平台数据划分方式对比

本次实验在 Arm 平台上分别对比了循环划分、块划分和动态划分三种方式的表现效果，表现情况如图4.5所示

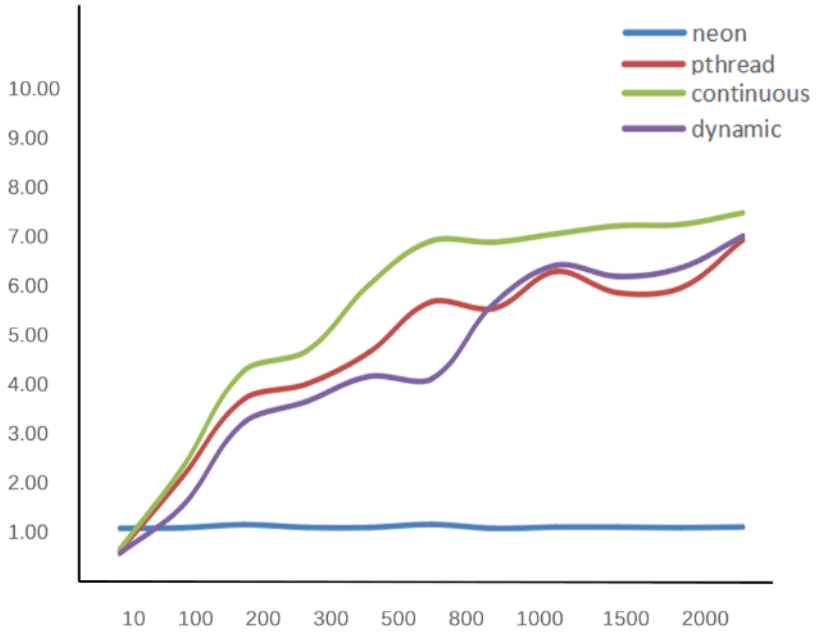

图 4.5: x86 平台数据划分方式对比

缓存优化关键在于最大限度减少缓存失效（cache miss）的发生，以加快数据访问速度和提高程序性能。在并行计算中，选择合适的数据划分策略对于缓存命中率至关重要。循环划分通常将数据按照固定间隔分配给每个线程，如每个线程处理的元素之间间隔为线程数。这种划分可能导致较大的缓存失效，特别是当处理的数据项大于 L1 缓存容量时。由于每个线程处理的数据在内存中是分散的，CPU 预取逻辑难以有效预测和加载即将访问的数据，从而导致高缓存失效率。按照实验数据，循环划分的缓存命中率通常低于块划分。块划分将连续的数据块分配给每个线程处理，有助于保持高空间局部性。这种方法允许 CPU 预取逻辑有效地预加载数据到缓存中，因为线程连续访问接近的内存位置。结果显示块划分可以实现更高的缓存命中率。因此，从缓存优化的角度看，块划分通常优于循环划分，尤其是在数据集较大时更为明显。

从负载均衡角度总结负载均衡的目标是确保所有处理单元（如线程）在计算过程中尽可能均衡地工作，避免某些线程过早完成任务而处于闲置状态，从而提高整体效率。循环划分可能导致工作负载不均。因为循环划分的方式是静态的，固定间隔地分配任务，这可能导致不同线程之间处理速度的差异，特别是当任务的处理时间不均匀时。在问题规模小或工作负载不均匀的情况下，某些线程可能较快完成任务并等待其他线程，这样会导致 CPU 资源的浪费。动态划分策略在运行时动态分配任务给线程，可以更好地适应不均匀的任务负载。线程在完成当前任务后会请求新的任务，这有助于保持所有线程的持续工作，最小化空闲时间。然而，动态划分可能会带来更高的线程管理开销，尤其是在任务分配和同步方面。在较小的问题规模下，这种开销可能会抵消由负载均衡带来的性能增益。在问题规模较大时，动态划分的性能通常会优于循环划分，因为负载更均衡，能够更充分利用每个线程的计算能力。

4.2 OpenMP

4.2.1 X86 平台并行实现及测试结果

本实验测试了在 X86 架构下，在 SSE 指令集上 openmp 的不同测试集规模下的运行结果，如下表格所示

N	SSE	openmp
100	1.674	1.305
200	1.683	3.610
300	1.692	6.020
400	1.674	7.566
500	1.683	8.016
600	1.665	8.893
700	1.683	9.981
800	1.683	10.056
900	1.683	10.537
1000	1.683	10.986
1500	1.692	11.633
2000	1.692	12.243
2500	1.674	13.011
3000	1.692	13.996

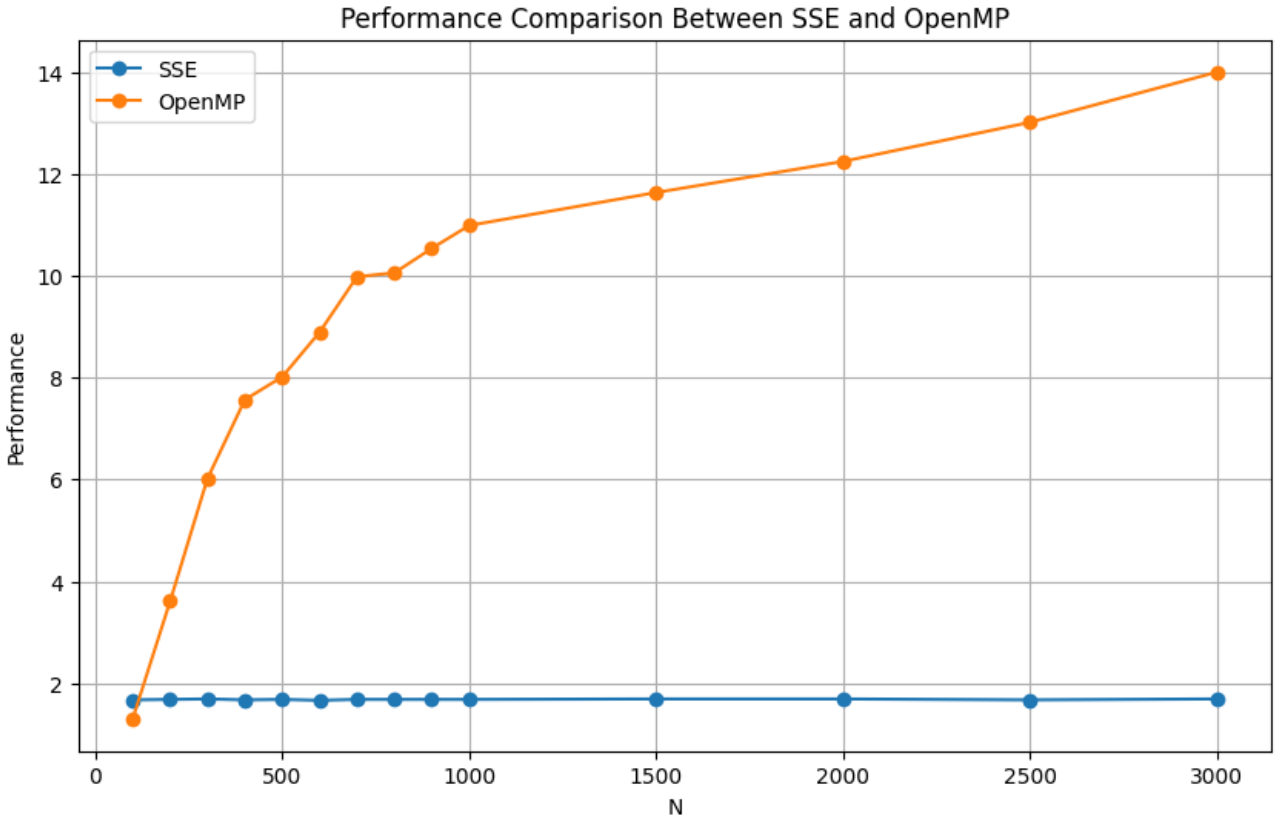

图 4.6: x86 平台测试结果

从数据中可以看出，随着问题规模的增加，SSE 指令集架构的加速比保持在一个比较稳定的水平上，这说明向量化的优化已经达到了一个瓶颈，而之所以没有能够达到理论实验结果分析加速比是因

为整个程序并不能完全进行向量化的展开，因此其余未能够向量化的部分极大的影响了整体的加速比。而横向对比结合了 SIMD 的 openmp 方法，则可以发现，由于在实验中总共拉起了 8 条线程，因此随着问题规模的增加，当问题规模达到 3000 时，这三种指令集架构的 openmp 算法相对于 SIMD 算法的加速比已经能够逼近 9 倍了，说明在 devcloud 平台上，多线程的性能能够得到充分的释放。

4.2.2 Arm 平台下测试结果

我测试了在不同任务规模下，串行算法，手动 SIMD 算法，手动 pthread 算法以及 openmp 版本的 SIMD 算法、openmp 版本的多线程算法的时间性能表现。测试结果如下

表 2: Adjusted Data Table

N	Serial	SIMD	OpenMP_SIMD	Pthread	OpenMP
100	2.39	1.55	1.84	2.58	0.97
200	18.88	12.07	14.18	6.19	3.38
300	63.90	40.52	47.74	13.49	8.42
400	152.73	96.72	113.97	26.30	18.78
500	306.52	192.94	227.40	46.39	35.17
600	537.80	337.67	397.07	75.39	55.52
700	856.91	536.17	632.12	110.08	89.94
800	1283.08	801.31	936.65	158.45	131.35
900	1839.90	1137.55	1337.76	220.70	184.99
1000	2463.67	1540.38	1809.94	335.67	248.03
1500	8543.98	5167.09	6095.91	879.25	798.42
2000	20861.01	12303.54	14461.65	1993.96	1846.77

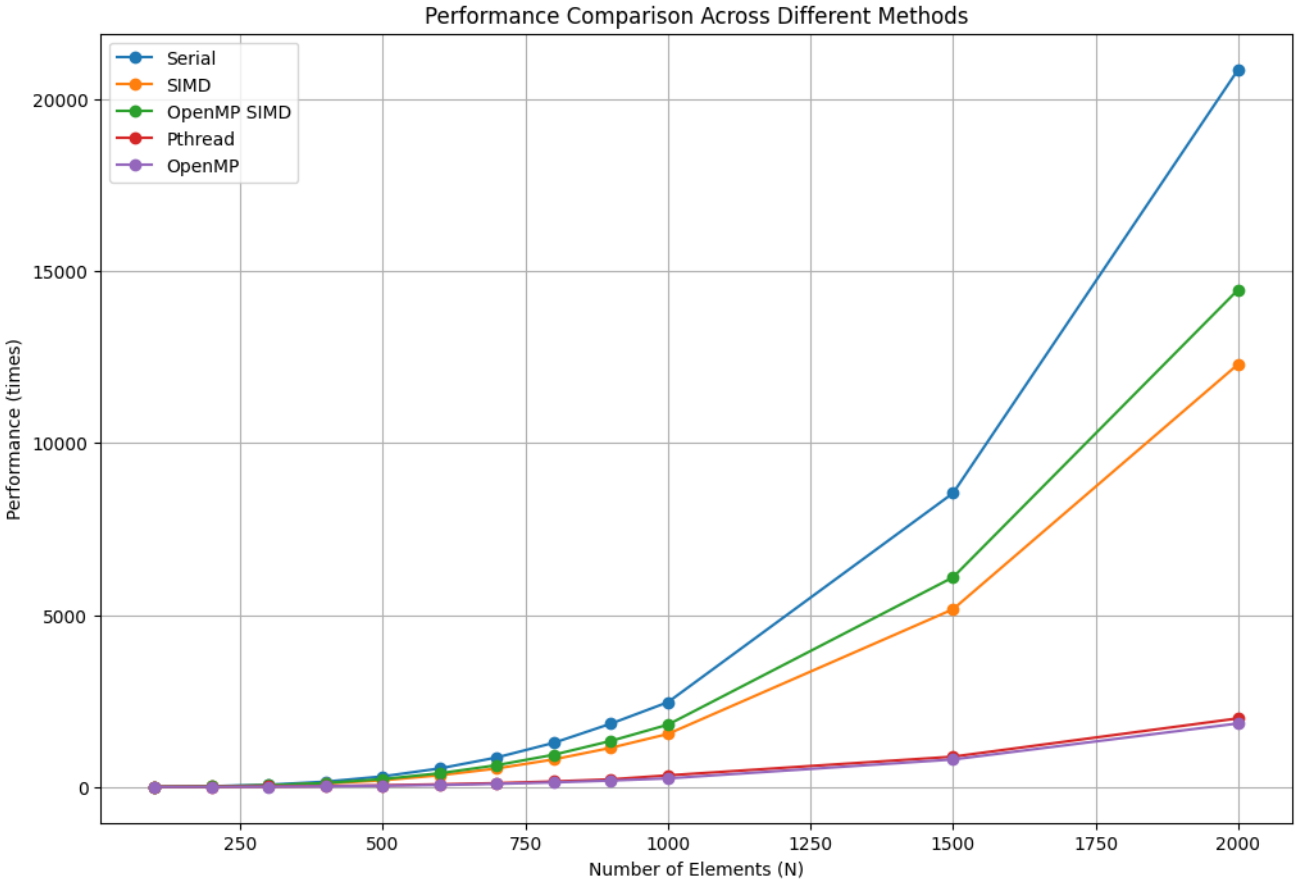

图 4.7: Arm 平台数据测试结果

如预期，串行实现的速度一致慢于并行版本。这一结果突出了并行处理在可以在多个处理器之间分配操作的计算任务中的好处。SIMD 配置通常显示出比串行配置显著的性能提升。SIMD 特别有效，因为它允许同时处理多个数据点，这是处理高斯消去中的矩阵运算的理想选择。OpenMP 配置通常优于基于 pthread 的实现。这可能是由于 OpenMP 更有效地管理了线程和资源，它从开发者手中抽象出了许多线程复杂性，提供了优化的同步和负载平衡。当使用 OpenMP 来管理 SIMD 操作时，与基本的 SIMD 实现相比，性能有明显的提升。这表明，协调 SIMD 操作的开销被 OpenMP 有效管理，从而更好地利用了硬件能力。所有并行实现与串行方法相比，在增加问题规模时都表现出更好的扩展性。这种扩展性是并行计算的一个关键方面，其中较大的问题规模通常提供更多的并行化空间，从而比小问题更显著地减少了计算时间。

4.2.3 数据划分测试情况

实验中关于数据划分的测试情况主要探讨了不同的任务划分方式对并行计算性能的影响。具体分析了三种常用的数据划分方法——静态（static）、动态（dynamic）和指导式（guided）——并通过实验比较了它们在高斯消去法中的表现。实验情况如图所示：

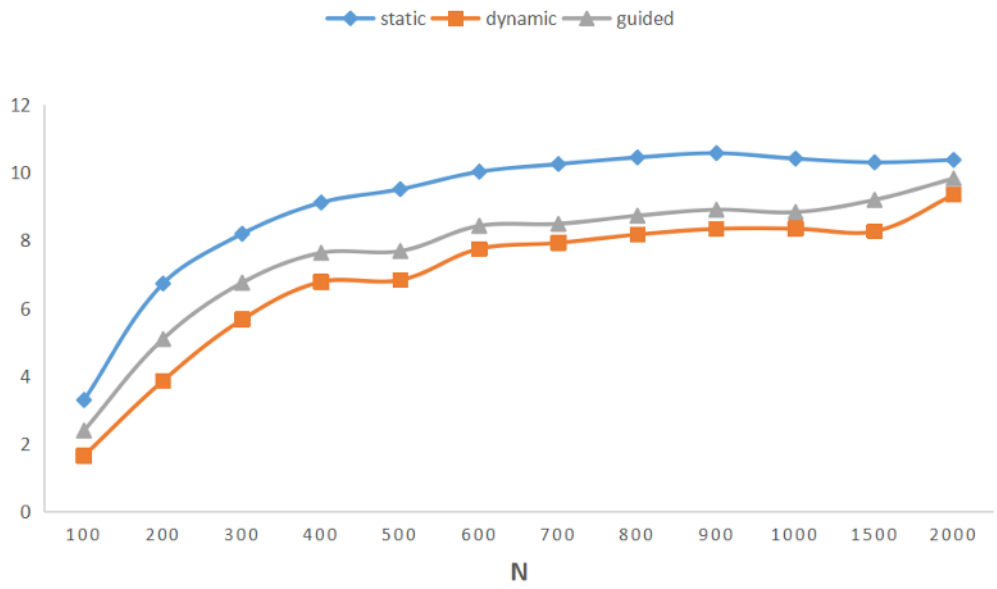

图 4.8: 不同划分方式加速比变化情况

实验通过对比这三种数据划分方法在不同数据规模下的性能，揭示了它们的优缺点。在较小的数据规模下，由于任务划分和线程同步的开销，所有并行方法的性能都不如串行方法显著。在数据规模增大时，各种并行方法的性能均有提升，其中动态划分由于能更好地适应任务的不均匀性，通常表现最优。静态划分在任务相对均匀时表现良好，因为它减少了运行时的调度开销，而指导式划分则在某些情况下提供了更好的负载均衡。总之，这些数据划分方法的选择和应用依赖于具体的计算任务和环境，合理的选择可以显著影响并行程序的性能和效率。

5 总结

5.1 Pthread

在这次并行程序设计的实验中，主要研究了基于 pthread 多线程编程对高斯消去法的并行优化。实验围绕不同的数据划分方法、多线程的实施效率以及在不同硬件平台上的表现进行了深入探索。实验的目标是通过多线程技术优化高斯消去法的计算效率，特别是在处理大规模数据时。通过将计算任务分配给多个处理器，可以显著减少解决问题所需的总时间。实验中设计了几种不同的数据划分策略，包括静态划分、动态划分和块划分，以探索哪种方法最能提高并行计算的效率。实验结果表明，考虑缓存优化的块划分方法在性能上优于简单的循环划分，特别是在大规模数据处理时更能有效利用缓存，减少缓存未命中导致的额外访存开销。动态划分方法在负载均衡方面表现更佳，特别是在不同线程处理速度不一致时，能够有效减少线程闲置时间，提高整体效率。通过与串行算法和简单的 SIMD 优化算法比较，pthread 多线程优化显示出显著的性能提升，尤其是在数据规模较大时。线程数量的增加初期能显著提高性能，但超过一定数量后性能提升会趋于平稳，这主要受限于 CPU 核心和线程的物理限制。本实验通过详细的设计和严格的测试，展示了多线程和数据划分策略在并行计算中的重要性和效果。实验结果不仅提高了高斯消去法的计算效率，也为未来在不同硬件平台上应用并行计算提供了宝贵的经验和数据支持。实验的所有代码和文档已经上传至 GitHub，供进一步的研究和开发使用。

5.2 OpenMP

在本次并行程序设计的实验中，我们主要围绕高斯消去法的 OpenMP 并行优化进行了深入的研究和实验。本实验的目的是通过多线程优化来提高高斯消去法处理大规模数据的效率。通过设计和执行一系列的实验，我们探索了不同的数据划分方法对并行计算性能的影响，并在 ARM 和 x86 两种不同的硬件平台上进行了实验，以比较不同平台上并行计算的性能差异。

实验设计实验设计包括了多个方面：

OpenMP 并行处理：使用 OpenMP 库实现高斯消去法的并行化，包括除法和减法运算的并行处理。不同数据规模和线程数下的性能探究：分析了问题规模和线程数量对性能优化效果的影响。任务卸载尝试：尝试使用 GPU 进行部分计算任务的卸载，以探索 GPU 加速的潜力。性能提升：并行优化显著提高了高斯消去法的计算速度，尤其是在大规模数据处理时。与串行版本相比，OpenMP 并行版本能够有效地缩短计算时间。数据划分的影响：动态数据划分因其能够更好地适应不同线程的处理速度而显示出较好的性能，而静态数据划分在负载较为均匀时也表现出色。本次实验成功地展示了 OpenMP 在并行化高斯消去法中的应用效果，特别是在处理大规模问题时的优势。通过合理的数据划分和线程管理，可以显著提高程序的运行效率。实验还揭示了在不同的硬件平台上进行并行程序设计时需要考虑的特定因素，如缓存管理和线程调度等。实验相关的源代码和文档已上传至 GitHub。**Github 地址：** Github