南開大學
Nankai University

MPI 并行化编程

姓名：钟坤原

学号：2212468

专业：计算机科学与技术

2024 年 6 月 9 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 MPI 多进程编程，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 MPI 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	C++
Arm	华为鲲鹏 920 处理器	48KB-1.25MB-24MB	C++

表 1: 实验环境

3 实验设计

高斯消去有两个阶段，一是在消元行行内除法过程，一是其余行减去消元行的过程。在其余行减去消元行的过程中，各个操作都是一样的，适合做并行操作。而当任务下发给不同的进程之后，在各个进程内又可以开启多条线程来处理任务。本次实验中，我做了以下工作：

- **MPI 并行处理**：高斯消去过程中，每一轮消去包括两个阶段：针对消元行的除法运算和对剩余被消元行的减法操作。除法运算和消元减法操作需要有严格的执行顺序，除法完成后，需要通过广播将结果通知给其他进程以同步数据。
- **数据划分设计**：考虑 MPI 是进程级的并行手段，实验设计了不同的数据划分方式，并对比了块划分和循环划分的性能。块划分分配连续行给进程，可能导致消元进程早期闲置。循环划分按步长分配行给每个进程，以保持负载均衡。

- **不同数据收发方式的实验对比**:探讨了进程间通信方式对性能的影响,对比了广播和流水线(pipeline)两种通信模式。流水线模式通过减少等待和阻塞来优化性能，每个进程接收到数据后转发给下一个进程，并开始自己的计算。
- **问题规模和进程数量的对比**：研究了问题规模和进程数量对并行性能的影响，预测在小规模问题中多进程带来的通信开销可能抵消并行优化效果。大规模问题下，进程通信所需时间相对于任务执行时间占比较低，能够更好地反映出并行优化的效果。

4 实验结果分析

4.1 X86 平台

4.1.1 MPI 性能分析

我测试了在不同问题规模下,串行算法,MPI 优化,MPI+SIMD,MPI+OpenMP 和 MPI+SIMD+OpenMP 优化的计算速度表现，MPI 使用了 8 进程进行并行，OpenMP 通过 8 线程进行并行，SIMD 使用四路向量化。不同算法表现如下所示

N	serial	MPI	MPI_SIMD	MPI_OMP	MPI_OMP_SIMD
100	1.404	2.151	4.644	5.742	6.03
200	11.214	5.526	9.099	12.222	11.628
300	37.656	13.644	15.309	21.24	19.377
400	89.046	26.838	23.994	38.835	36.279
500	173.96	48.024	35.964	53.136	46.71
600	303.102	77.148	50.994	72.396	59.373
700	480.87	114.39	70.704	93.096	76.986
800	713.646	174.159	95.346	121.14	98.955
900	1017.828	239.085	127.872	164.915	118.71
1000	1394.775	329.274	163.071	195.318	146.412
1500	4705.137	1124.046	528.723	529.146	360.459
2000	11292.57	2569.994	1108.305	1098.288	698.238
2500	22103.1	4934.646	2105.163	2121.813	1201.725
3000	38067.21	8963.343	3580.567	3320.298	1960.038

表 2: x86 平台性能随问题规模的表现

下图显示了不同并行计算方法处理调整后数据时的性能变化，针对不同的问题规模（N）。

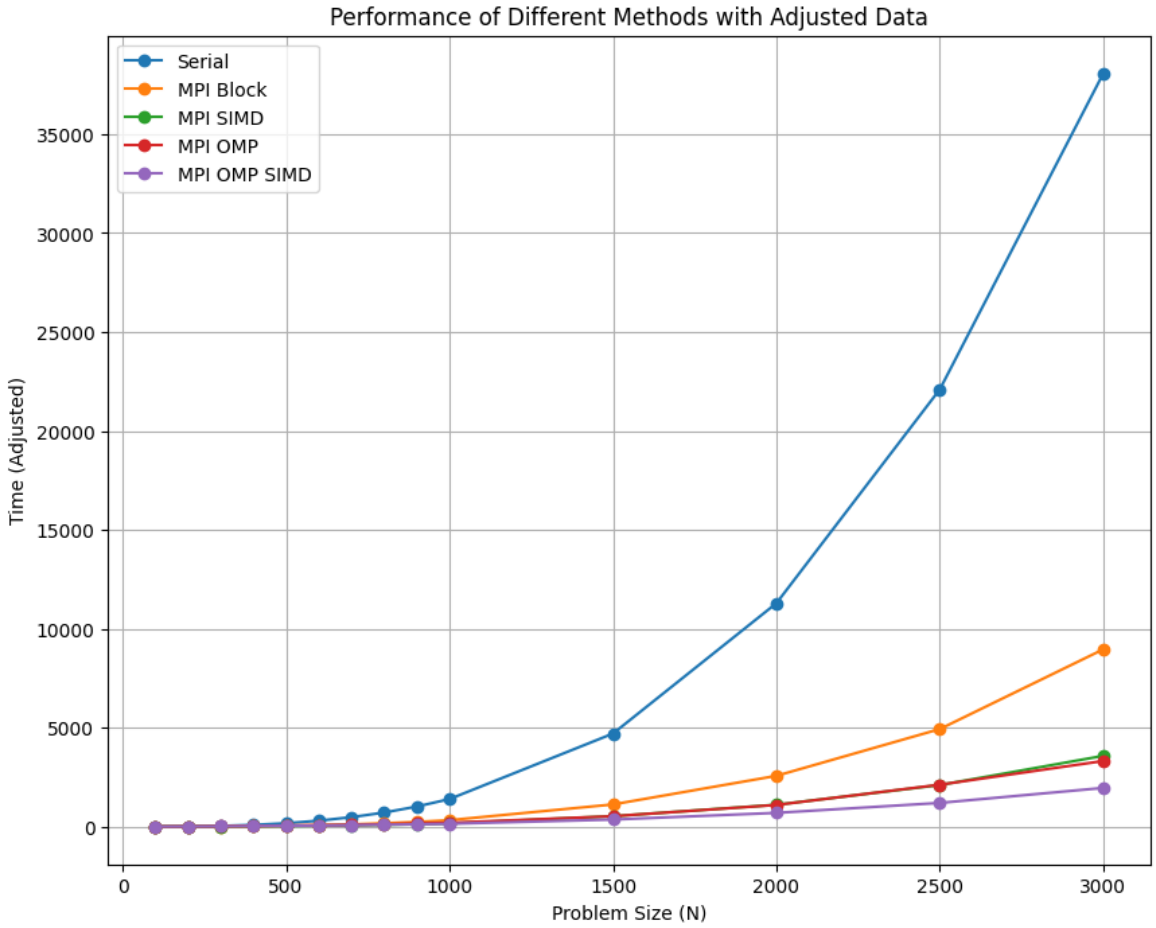

图 4.1: x86 平台运行时间随问题规模的表现

根据算法进行分类，我们可以得知不同的方法有以下特征：

- **序列方法（Serial）**：这是基线方法，没有并行优化。随着问题规模的增加，执行时间显著增长，特别是在问题规模超过 2000 之后，增长趋势更为急剧。这显示了对于大规模问题，序列方法的效率极低。
- **MPI 块分解**：使用 MPI 进行的块分解方法相比序列方法有显著的性能提升。其时间增长速度较慢，显示出较好的并行效益，尤其是在中等规模的问题上。
- **MPI+SIMD**：这种方法展示了更为平稳的性能增长曲线，与 MPI 块分解相比，其在较大问题规模下表现更好，这表明 SIMD 向量化对于加速计算效果显著。
- **MPI+OpenMP**：这种结合了 MPI 和 OpenMP 多线程优化的方法，在所有并行方法中表现较好，性能稳定且提升明显，尤其在问题规模增大时仍保持较低的增长率。
- **MPI+OpenMP+SIMD**：这是最复杂的并行优化组合，理论上应该提供最高的性能。图中显示，其在较小规模问题上与其他方法差异不大，但在规模超过 2500 时，其性能提升明显，处理时间的增长几乎趋于平缓。

加速比如下图所示。

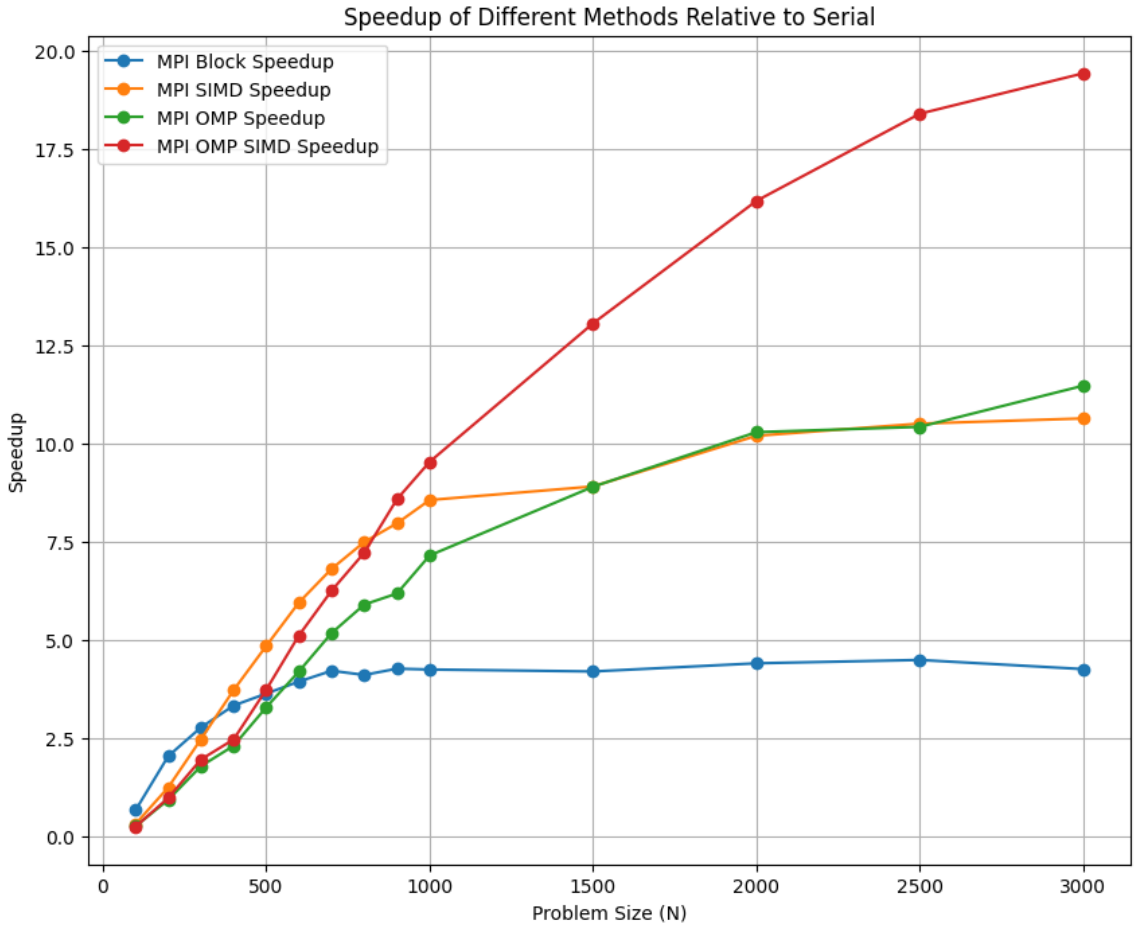

图 4.2: 不同算法加速比随问题规模的变化

所有并行方法都有效地减少了计算时间，特别是在问题规模较大时。尽管并行化可以显著加速计算，但经过计算，我们发现 MPI 的加速比最后稳定在 4.7 左右，并没有实现 8 线程的加速比为 8，原因可能是通信开销可能会随着进程数的增加而显著增加，特别是在大规模数据处理时。

4.1.2 数据划分

随着高斯消去法的进展，消元过程逐渐完成，前面的行不再参与后续计算。这导致在消元后期，那些早期完成任务的进程会处于闲置状态，从而导致资源浪费和效率下降。

在数据划分中，我实现了块划分和循环划分，块划分方法涉及将连续的行或数据块分配给进程。这种划分简单直接，易于实现。循环划分方法将数据分配给进程时采用跨行均匀分配的方式，即按照进程数量均匀分配所有行，确保每个进程获得的行数大致相等，分散在整个矩阵中。这种方法可以更均匀地分配工作负载，避免了块划分中出现的闲置问题。每个进程在整个消去过程中几乎都保持忙碌状态，这有助于提高整体的运算效率和加速比。

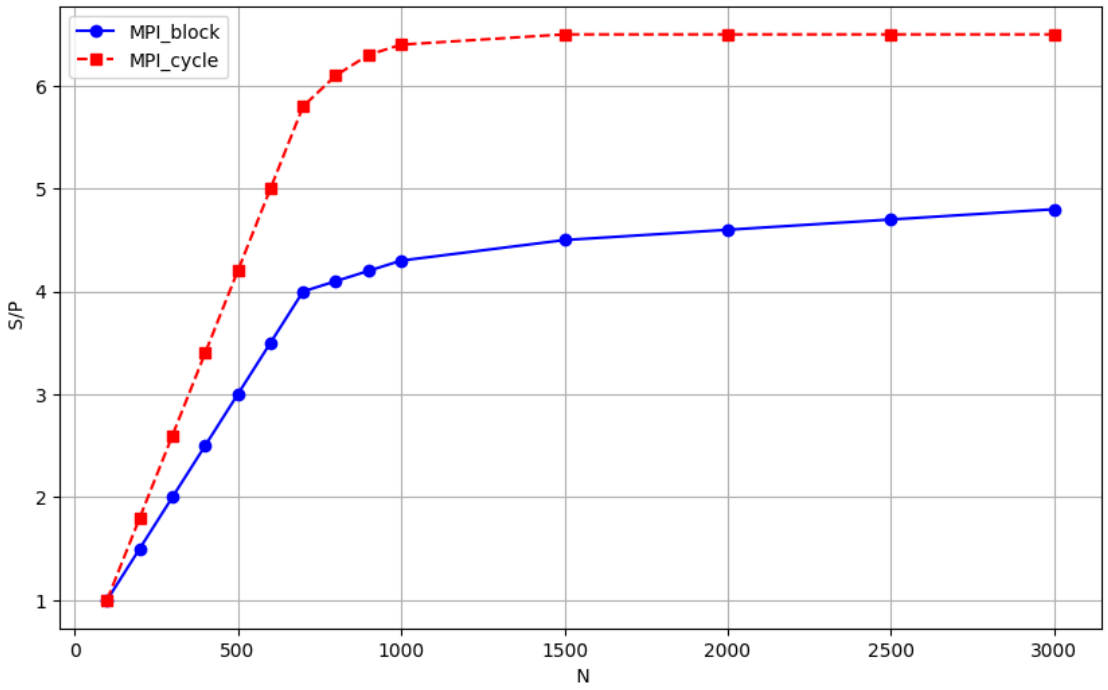

图 4.3: X86 平台不同数据划分方式加速比的变化

根据图像可以得知，块划分的性能随问题规模增大而先提升后趋于平稳，但通常无法达到理论最优加速比，尤其是在大规模问题处理时，由于进程闲置导致的性能损失更为显著。循环划分。循环划分方法显示出较块划分更稳定和高效的性能。随着问题规模的增加，循环划分的加速比逐步提升，并更接近理论加速比。

4.1.3 数据收发方式

MPI 在接收数据的时候处于阻塞状态，在这部分可以借助流水线的思想来进行。在流水线方式中，数据按顺序从一个进程传递到下一个进程。这种方式模拟了生产线的流程，数据逐步被处理和转发。这种方法可以减少等待时间，因为一旦一个进程完成其数据处理，它就会立即将数据传递给下一个进程，而不必等待其他进程。流水线方式在处理大规模数据时通常更有效，因为它允许数据在进程间连续流动，减少了整体的等待时间和阻塞。

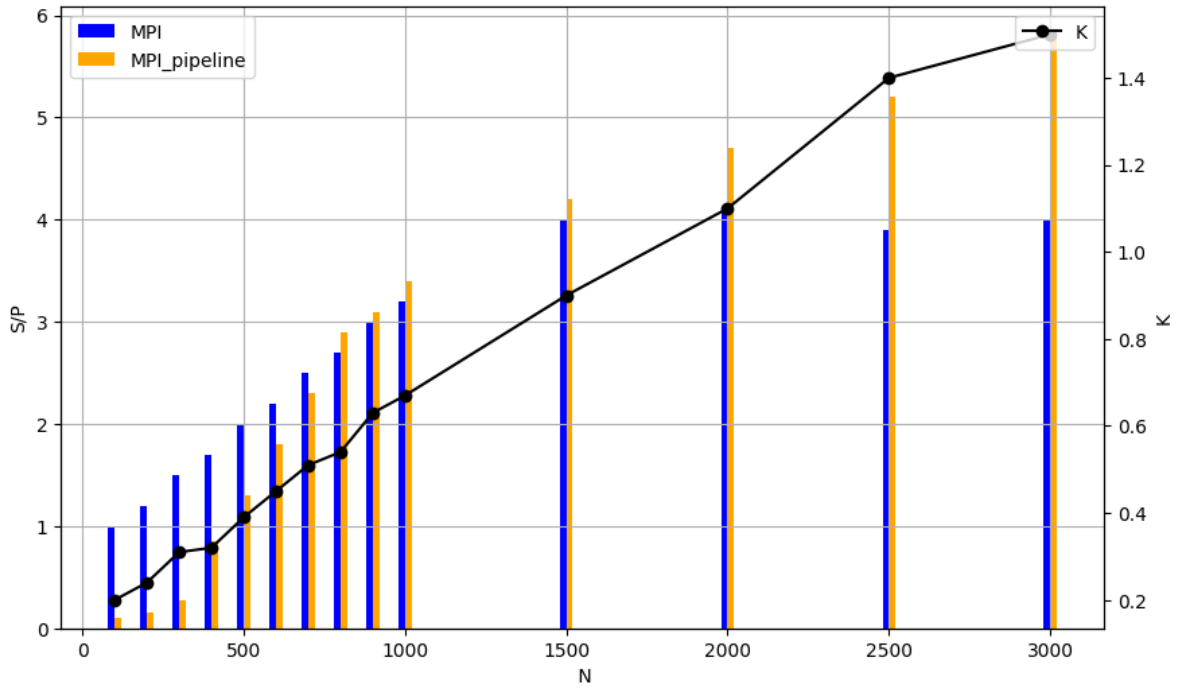

图 4.4: X86 平台不同数据收发方式加速比的变化

由图像可知，流水线方式在大多数情况下比正常方式更高效。特别是在处理大规模数据集时，流水线方式因其减少了阻塞和等待时间而显著提高了性能。流水线方式特别适用于数据依赖性强的应用，其中数据处理可以分阶段进行，而每个阶段的输出可以立即用作下一阶段的输入。尽管流水线可以减少等待和阻塞，但在某些情况下，它可能引入额外的通信开销，比如数据量较小的时候，因为每个数据包都需要在进程间传递，这在进程数量非常多时可能成为新的瓶颈。

4.1.4 进程数量对比

我尝试了在 2-8 个进程下，MPI 算法的表现效果，如下图所示

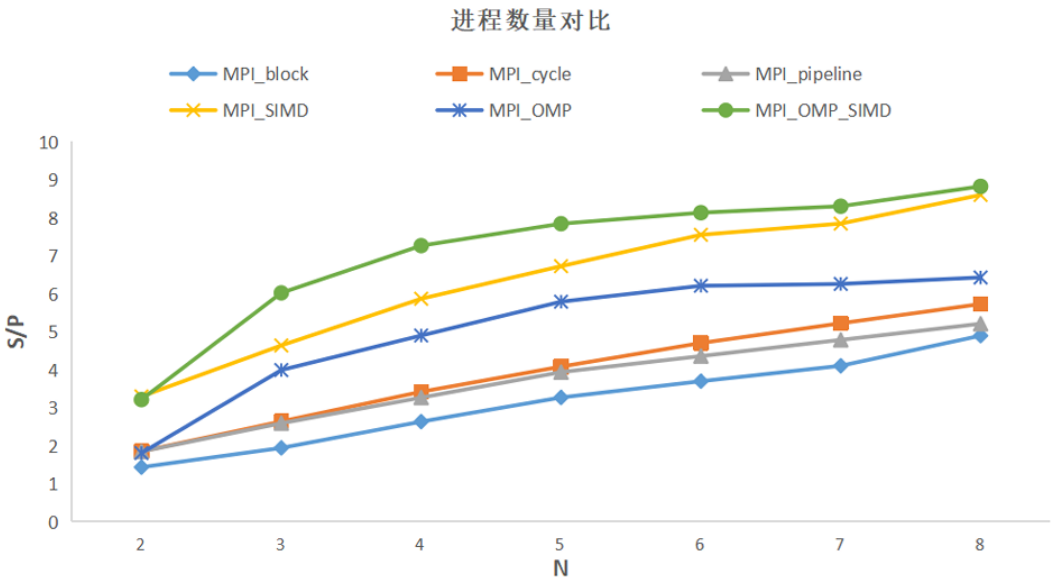

图 4.5: X86 平台不同进程下加速比

由图像我们可以看出，当进程数量较多时，如 MPI_block 和 MPI_cycle 在问题规模较大时的性能增长可能受到通信延迟的影响。特别是块划分，可能由于进程间数据依赖和同步导致性能不是线性增长。优化方法如 MPI_OMP_SIMD 显示了较高的加速比，这表明其计算与通信比更优，即它们在增加计算负载的同时有效减少了相对通信成本的影响。MPI_cycle 的持续良好表现说明了循环划分在保持进程工作负载均衡方面的效果，特别是在进程数量较多的情况下。

4.2 ARM 平台

4.2.1 MPI 性能分析

Arm 平台下我依然是测试了在不同问题规模下，串行算法，MPI 优化，MPI+SIMD，MPI+OpenMP 和 MPI+SIMD+OpenMP 优化的计算速度表现，MPI 使用了 8 进程进行并行，OpenMP 通过 8 线程进行并行，SIMD 使用四路向量化。多进程需要在鲲鹏服务器下申请多个结点来进行。不同算法表现如下所示

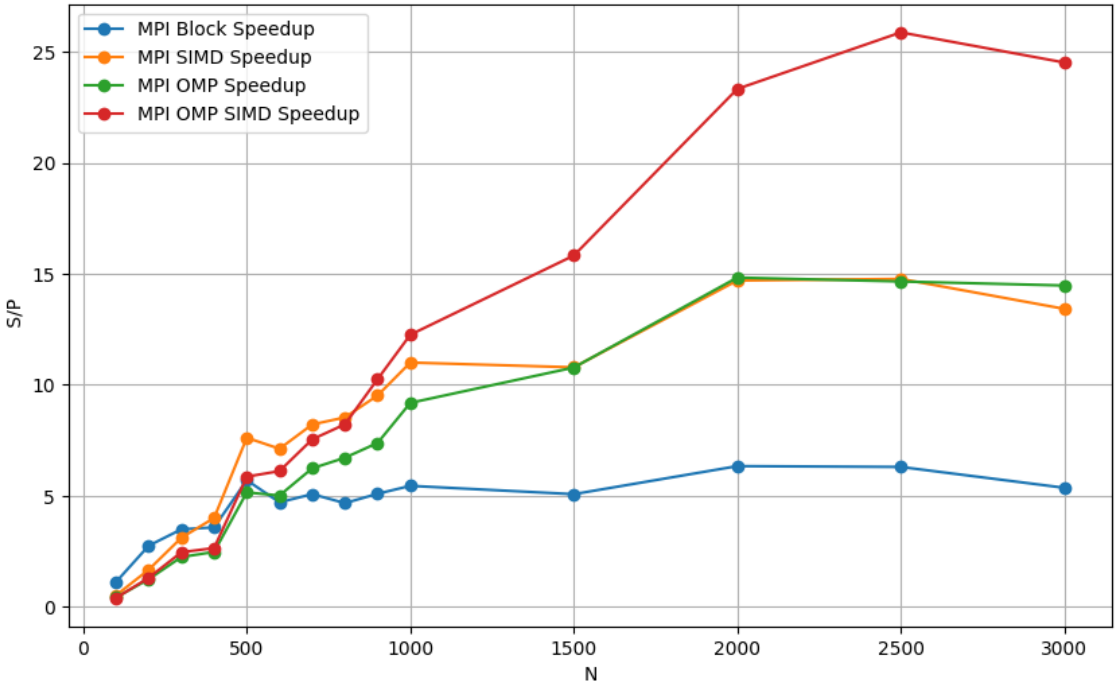

图 4.6: ARM 平台下加速比情况

上图显示结合 OpenMP 和 SIMD 的 MPI_OMP_SIMD 在较大规模数据处理上具有显著的性能优势。这反映了并行编程中，多种优化策略联合使用的重要性。SIMD 和 OpenMP 各自能带来性能提升，但它们的组合提供了更加显著的加速。但是 SIMD 和 MPI 都没有能够达到理想的加速比，可能是通信等原因造成的问题，体现出硬件特性和编程模型需要紧密配合才能达到最佳性能。这种多样化的优化方法可以根据不同的硬件配置和问题规模进行调整，以适应各种性能需求。

4.2.2 数据划分

在本次实验中，从负载均衡的角度触发，探究不同的任务划分方式对于算法性能的影响。由于 MPI 是进程级的并行方式，因此进程间的通信开销是很大的，所以要尽可能地利用每一个进程，并且减少进程之间的通信次数。实验对比了两种不同的任务划分方式，块划分和循环划分，实验结果如图所示。

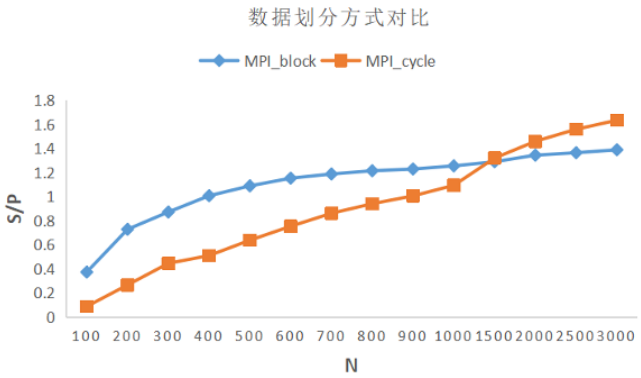

图 4.7: ARM 平台不同数据划分方式加速比的变化

从图像中可以看出，随着任务规模的增加，块划分方式率先达到了性能瓶颈，并且只达到了理论加速比的一半左右。这很符合预期，从整个过程来看，采用块划分的方式，随着消元的推进，负责前面行的进程会逐渐空闲下来，因此在后续的计算过程中，实际工作的进程会越来越少。平均下来每个进程只有一半的时间在有效工作。而对于循环数据划分的方式而言，由于其从计算量的角度去划分任务，因此整体来看每个线程的有效工作时间几乎相同，达到了比较好的负载均衡，充分利用了每个线程的计算资源，因此其加速比逐步逼近理论加速比 2。

4.2.3 数据收发方式

下图是 ARM 平台下不同数据收发方式加速比变化

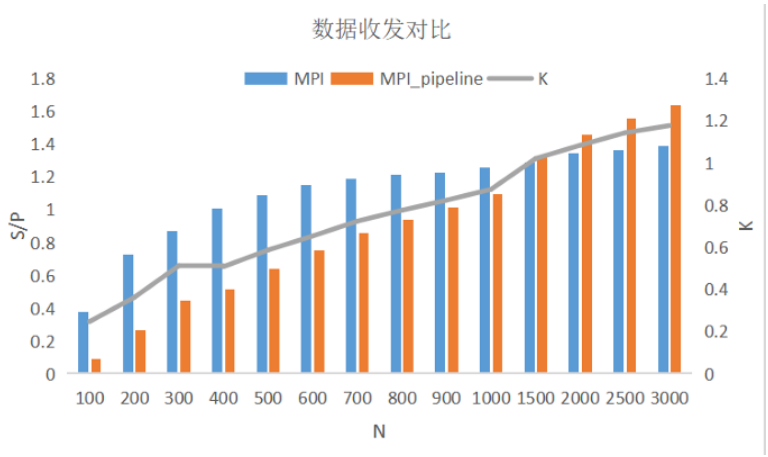

图 4.8: ARM 平台不同数据收发方式加速比的变化

从图像中可以看出，随着问题规模的增加，采用 pipeline 方式的数据收发可以提高性能瓶颈，普通的收发方式加速比只能够达到 1.2 左右，而采用了 pipeline 的加速比可以达到 1.6 左右。同时也注意到了，由于 ARM 平台上只启用了两个进程，因此不同的数据收发方式在本次实验中的效果并不明显，而且当数据规模较小的时候，由于 pipeline 的方式增加的数据收发的次数，反倒是表现出了更差的性能。

4.2.4 进程数量对比

我尝试了在 2-8 个进程下，MPI 的表现效果，如下图所示

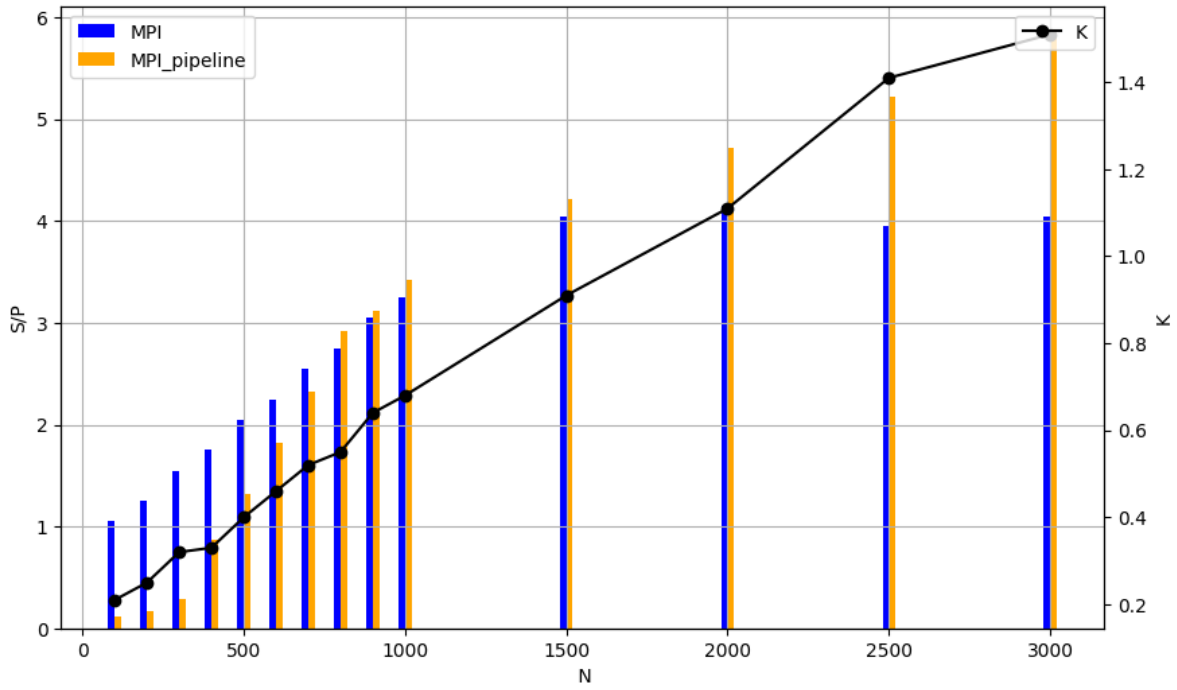

图 4.9: ARM 平台不同进程下加速比

图像显示在进程数量较少时，加速比逐渐提升，这表明增加进程有效地分散了计算负载并减少了总体执行时间。在进程数量达到某个点后加速比增长放缓或停止增长，这可能表明达到了并行效率的瓶颈。此时，进程间通信和同步可能成为主要限制因素。特别高效的曲线（如结合了 OpenMP 和 SIMD 的 MPI 策略）可能显示出更好的负载管理和降低的通信影响，这表明复合优化策略能够更有效地利用增加的进程数。

5 总结

在本次实验中，我对 X86 以及 ARM 平台上 MPI，MPI+SIMD，MPI+OMP 等多种并行编程策略的性能进行了深入分析。通过使用消息传递接口（MPI），结合单指令多数据（SIMD）和 OpenMP 技术，我探索了不同优化方法对并行加速比（S/P）的影响。结合 SIMD 的 MPI 实现在提高数据处理能力方面显示了显著效果，特别是在处理大规模数据时。MPI 和 OpenMP 的结合使用展示了优异的多线程处理能力，特别是在并行任务中有效分配计算负载方面。增加进程数量通常会提升加速比，但这种提升在达到一定的进程数后会逐渐放缓，这可能是由于通信开销和数据同步的限制。在进程数量较多时，特别是在结合了 OpenMP 和 SIMD 技术的 MPI 策略中，加速比的增长尤为显著，反映了优化的并行算法在有效利用硬件资源方面的潜力。数据划分策略（如块划分与循环划分）和数据通信方式（如广播与流水线）对性能有显著影响。

代码与图片文件已上传 GitHub **Github 地址：** Github