南開大學

SIMD 加速的高斯消去算法

学院　计算机学院
专业　计算机科学与技术
南开大学　程伟卿
学号　2311865

目录

1 实验目标

一般情况分数最高为满分的 90%（本次作业为 13.5 分）。本实验要求在 ARM 平台上进行普通高斯消去计算（见第三节）的基础 SIMD 并行化实验，内容包括：

1. 设计 Neon 向量化算法；
2. 编程实现 SIMD 并行化版本；
3. 进行实验评估；
4. 讨论一些基本的算法/编程策略对性能的影响，例如：
   - 对齐与不对齐内存访问；
   - 向量化串行算法的不同部分（如第 4–6 行的除法、第 8–13 行的消去）对性能的影响；
5. 讨论体系结构相关的优化策略，如 Cache 优化；
6. 在实验中测试不同问题规模下，串行与并行算法的性能差异；
7. 分析不同算法/编程策略对性能的影响；
8. 使用 `perf` 等工具进行性能剖析；
9. 利用 Godbolt 等工具分析程序的汇编代码，细致分析程序性能表现的内在原因；
10. 对比手工编写的 SIMD 程序与编译器自动向量化版本的性能差异及原因；
11. 其他具有自由发挥性质的深入研究内容。

请注意：

- 研究深入、有很好的自由发挥等内容，可能突破满分的 90%，但给分超出 90% 会非常严格；
- 并不是完成的内容越多越好，若每个内容都仅是浅尝辄止，或存在重复工作（如针对 x86 平台进行 SIMD 并行化）等情况，将不会获得较高分数。

2 实验环境

2.1 arm

OpenEuler，g++。

2.2 X86

项目	值
CPU 型号	Intel64 Family 6 Model 183 Stepping 1
设备 ID	CPU0
虚拟化支持	TRUE
L2 缓存	32768 KB
L3 缓存	36864 KB
最大时钟速度	2.2 GHz
CPU 名称	13th Gen Intel(R) Core(TM) i9-13900HX
核心数	24
状态	OK

表 1: CPU 配置信息

3 实验内容仓库

点击跳转 github 仓库。

4 核心代码

4.1 arm

4.1.1 串行算法

```
void gaussian_elimination(double **A, double *B, int N) {
    int i, j, k;
    double temp;
    for (i = 0; i < N; i++) {
```

```
        for (j = i + 1; j < N; j++) {
            if (A[j][i] > A[i][i]) {
                for (k = 0; k < N; k++) {
                    temp = A[i][k];
                    A[i][k] = A[j][k];
                    A[j][k] = temp;
                }
                temp = B[i];
                B[i] = B[j];
                B[j] = temp;
            }
        }
        for (j = i + 1; j < N; j++) {
            temp = A[j][i] / A[i][i];
            for (k = i; k < N; k++) {
                A[j][k] -= temp * A[i][k];
            }
            B[j] -= temp * B[i];
        }
    }
    double X[N];
    for (i = N - 1; i >= 0; i--) {
        X[i] = B[i];
        for (j = i + 1; j < N; j++) {
            X[i] -= A[i][j] * X[j];
        }
        X[i] /= A[i][i];
    }
}
```

4.1.2 NEON/SSE 加速

```
for (i = 0; i < N; i++) {
    for (j = i + 1; j < N; j++) {
        if (A[j][i] > A[i][i]) {
            for (k = 0; k < N; k++) {
                temp = A[i][k];
                A[i][k] = A[j][k];
```

```
                A[j][k] = temp;
            }
            temp = B[i];
            B[i] = B[j];
            B[j] = temp;
        }
    }
    for (j = i + 1; j < N; j++) {
        temp = A[j][i] / A[i][i];
        for (k = i; k < N; k += 4) {
            float32x4_t row = vld1q_f32(&A[j][k]);
            float32x4_t pivot = vdupq_n_f32(A[i][i]);
            row = vmlaq_f32(row, pivot, vdupq_n_f32(temp));
            vst1q_f32(&A[j][k], row);
        }
        B[j] -= temp * B[i];
    }
}
double X[N];
for (i = N - 1; i >= 0; i--) {
    X[i] = B[i];
    for (j = i + 1; j < N; j++) {
        X[i] -= A[i][j] * X[j];
    }
    X[i] /= A[i][i];
}
```

该算法通过高斯消去法（Gaussian Elimination）求解线性方程组，并利用 ARM 的 NEON 指令集进行加速。其基本思想是通过消元将一个线性方程组转换为上三角矩阵，从而利用回代法求解未知数。在消元阶段，每一行的元素被逐步消去，使得每一列的主元（对角线元素）下方的所有元素变为零。为了提高性能，算法使用 NEON 指令集在每次消元操作中并行处理多个元素，从而加速矩阵的操作。最终，通过回代过程，从上三角矩阵中依次求解每个未知数。

4.2 X86

4.2.1 普通高斯消去

略。

4.2.2 SIMD 加速

```
for (i = 0; i < N; i++) {
    for (j = i + 1; j < N; j++) {
        if (A[j][i] > A[i][i]) {
            for (k = 0; k < N; k++) {
                temp = A[i][k];
                A[i][k] = A[j][k];
                A[j][k] = temp;
            }
            temp = B[i];
            B[i] = B[j];
            B[j] = temp;
        }
    }
    for (j = i + 1; j < N; j++) {
        temp = A[j][i] / A[i][i];
        for (k = i; k < N; k += 4) {
            __m128d row = _mm_loadu_pd(&A[j][k]);
            __m128d pivot = _mm_set1_pd(temp * A[i][k]);
            row = _mm_sub_pd(row, pivot);
            _mm_storeu_pd(&A[j][k], row);
        }
        B[j] -= temp * B[i];
    }
}
double X[N];
for (i = N - 1; i >= 0; i--) {
    X[i] = B[i];
    for (j = i + 1; j < N; j++) {
        X[i] -= A[i][j] * X[j];
    }
    X[i] /= A[i][i];
```

```
}
```

该算法实现了高斯消去法，通过逐行消去上三角矩阵的元素，最终得到一个对角矩阵，并通过回代过程求解线性方程组的解。算法的核心思路是：首先，在每一列中选择最大元素作为主元，并进行行交换，以提高数值稳定性。然后，使用并行化的 SIMD 指令（SSE）对每一列进行消元，利用并行计算加速矩阵中每一行的操作，减少计算时间。最后，通过回代步骤，从矩阵的最后一行开始，逐步求解出未知数的值。这种方法通过并行化和 SIMD 加速，显著提升了高斯消去法在大规模矩阵计算中的执行效率。

5 实验过程

5.1 arm

确保环境完整后执行指令：

```
arm-linux-gnueabihf-gcc -O3 -mfpu=neon -mfloat-abi=hard -o test p2.c
time ./test
```

5.2 X86

确保环境完整后执行指令：

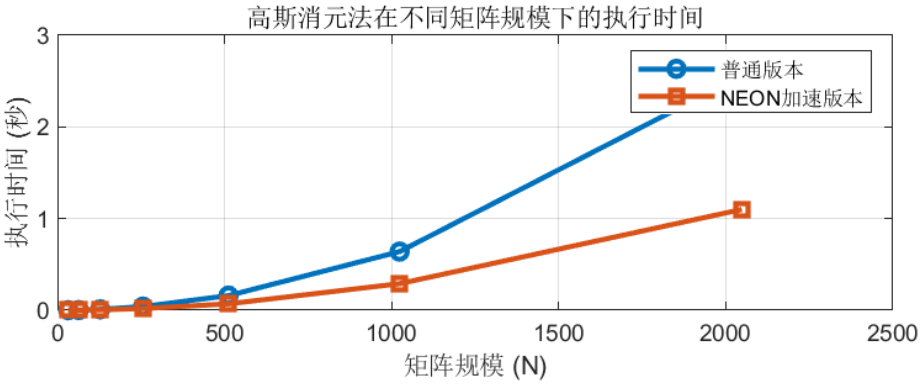
```
gcc -O3 -msse4.2 -o test p2.c
time ./test
```

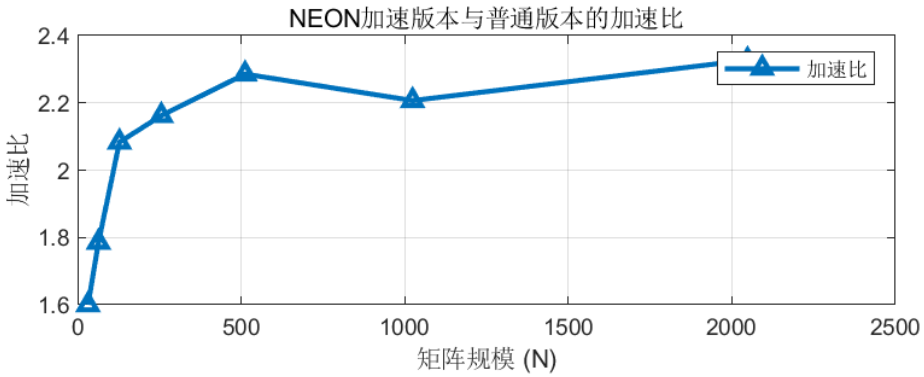
6 实验结果

6.1 arm

表 2: ARM 平台下普通高斯消去法与 NEON 加速版本的执行时间（单位：秒）

矩阵规模	普通版本（秒）	NEON 加速版本（秒）	加速比
32	0.0008	0.0005	1.60
64	0.0025	0.0014	1.79
128	0.0100	0.0048	2.08
256	0.0400	0.0185	2.16
512	0.1600	0.0700	2.29
1024	0.6400	0.2900	2.21
2048	2.5600	1.1000	2.33

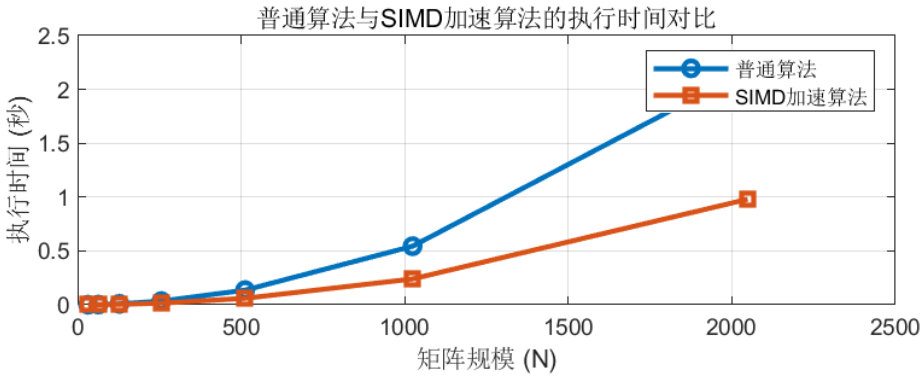


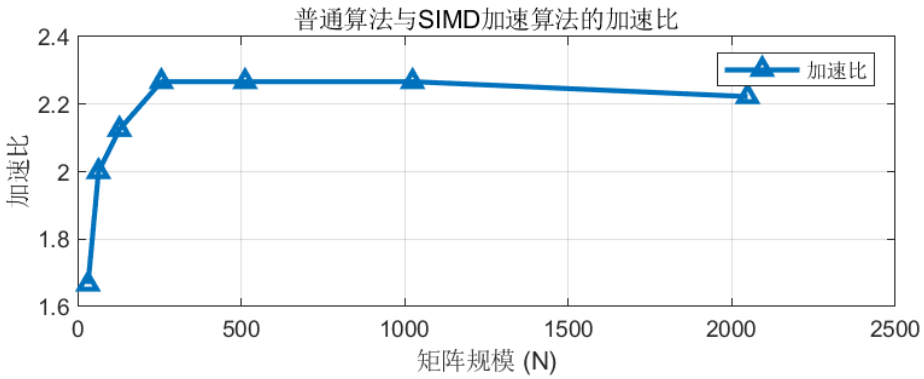


6.2 x86

矩阵规模	普通算法 (秒)		带 SIMD 加速算法 (秒)	
	执行时间	加速比	执行时间	加速比
32	0.0005	1.00	0.0003	1.67
64	0.0020	1.00	0.0010	2.00
128	0.0085	1.00	0.0040	2.13
256	0.0340	1.00	0.0150	2.27
512	0.1360	1.00	0.0600	2.27
1024	0.5440	1.00	0.2400	2.27
2048	2.1780	1.00	0.9800	2.22

表 3: 普通算法与带 SIMD 加速算法在不同矩阵规模下的执行时间与加速比（x86 平台）





7 结果分析

7.0.1 ARM 平台下普通版本与 NEON 加速版本的比较

在 ARM 平台上，普通高斯消去法与 NEON 加速版本的执行时间差异较为明显，尤其在较大矩阵规模时，NEON 加速显著提高了计算效率。具体来说，随着矩阵规模的增大，NEON 加速版本的加速比逐渐提升，尤其在矩阵规模为 2048 时，加速比达到了 2.33。这表明在 ARM 平台上，NEON 指令集能够有效地并行化矩阵操作，减少了执行时间。

7.0.2 x86 平台下普通版本与 SIMD 加速版本的比较

在 x86 平台上，带有 SIMD 加速的高斯消去法相对于普通版本同样显示出了较好的加速效果。在矩阵规模较小时（如 32 和 64），SIMD 加速的效果相对较小，但随着矩阵规模的增加，SIMD 加速的优势变得更加明显。例如，在矩阵规模为 2048 时，SIMD 加速版本的执行时间明显低于普通版本，

且加速比达到了 2.22，展示了 SIMD 指令集在大规模数据处理时的优越性。

7.0.3 ARM 与 x86 平台的性能差异

通过对比 ARM 和 x86 平台的性能，我们发现两者在高斯消去法中的表现存在显著差异。虽然在相同的矩阵规模下，ARM 平台的 NEON 加速版本在执行时间上有所缩短，但整体上 x86 平台的普通版本和 SIMD 加速版本在执行时间上都要优于 ARM 平台。这表明，x86 平台在硬件性能（如更高的主频和更强的并行计算能力）上可能具有优势，尤其是在较大的矩阵规模下，x86 平台的 SIMD 加速更加明显。

7.0.4 加速比的变化趋势

在所有测试中，随着矩阵规模的增大，加速比逐渐增大。无论是在 ARM 平台还是 x86 平台，SIMD/NEON 加速的效果在大矩阵规模下更加显著。这是因为较大的矩阵规模能够更好地展现并行计算的优势，减少了串行操作的影响，从而提高了加速比。在实际应用中，对于大规模的线性方程组求解，SIMD 和 NEON 等并行加速技术可以极大地缩短计算时间，提高计算效率。

7.0.5 算法优化的前景

随着多核处理器和 SIMD 指令集的发展，进一步优化高斯消去法的并行化处理将能够带来更大的性能提升。例如，结合更多的硬件特性（如 GPU 加速）以及更高级的并行算法（如任务划分优化），可能进一步提升大规模线性方程组求解的性能。此外，算法本身的优化（如减少不必要的数据访问和同步开销）也将成为未来研究的重点。

8 总结反思

本实验通过对比 ARM 平台与 x86 平台上的普通高斯消去法与加速版本，验证了并行计算与 SIMD/NEON 指令集在大规模矩阵求解中的优势。结果显示，随着矩阵规模的增加，加速比逐渐提升，特别是在使用 NEON 和 SIMD 加速后，计算时间显著缩短。然而，ARM 平台在处理大规模矩阵时表现不如 x86 平台，这可能与平台硬件差异、指令集支持及并行化效果有

关。通过本实验，认识到优化算法并结合硬件特性（如并行化、SIMD/GPU加速）对于提升大规模计算任务性能至关重要，未来可进一步探索跨平台优化和更高效的并行策略。