



中山大學
SUN YAT-SEN UNIVERSITY

并行程序设计 with 算法实验

Lab0-环境设置与串行矩阵乘法

姓 名 _____ 林隽哲

学 号 _____ 21312450

学 院 _____ 计算机学院

专 业 _____ 计算机科学与技术

2025 年 4 月 1 日

1 实验目的

- 理解并行程序设计的基本概念与理论。
- 掌握使用并行编程模型实现常见算法的能力。
- 学习评估并行程序性能的指标及其优化方法。

2 实验内容

- 设计并实现以下矩阵乘法版本：
 - 使用 C/C++ 语言实现一个串行矩阵乘法。
 - 比较不同编译选项、实现方式、算法或库对性能的影响：
 - * 使用 Python 实现的矩阵乘法。
 - * 使用 C/C++ 实现的基本矩阵乘法。
 - * 调整循环顺序优化矩阵乘法。
 - * 应用编译优化提高性能。
 - * 使用循环展开技术优化矩阵乘法。
 - * 使用 Intel MKL 库进行矩阵乘法运算。
- 生成随机矩阵 A 和 B，进行矩阵乘法运算得到矩阵 C。
- 衡量各版本的运行时间、加速比、浮点性能等。
- 分析不同实现版本对性能的影响。

3 实验结果

我当前使用的 CPU 为 Intel 酷睿 i5 12490，拥有 6 核心 12 线程，CPU 主频 3GHz，最高睿频 4.6GHz。浮点测试用的矩阵大小为 $2000 \times 200 \times 1000$ 。

- 单核峰值性能为 $4.6\text{GHz} \times 8\text{FLOPS/cycle} = 36.8\text{GFLOPS}$ 。
- 全核峰值性能为 $36.8\text{GFLOPS} \times 6 = 220.8\text{GFLOPS}$ 。
- 总浮点操作数为 $2000 \times 200 \times 1000 \times 2 = 8 \times 10^8\text{FLOP}$ 。

版本	实现描述	运行时间	相对加速比	绝对加速比	浮点性能	峰值性能百分比
1	Python	67,627.9	1×	0.00005x	0.0118x	0.005%
2	C/C++	3,626	18.66×	0.001x	0.220x	0.1%
3	调整循环顺序	3,087	21.91×	0.001173x	0.259	0.117%
4	编译优化 (-Ofast)	194	348.6×	0.019x	4.123	1.87%
5	循环展开	4,103	16.48×	0.0009×	0.195	0.09%
6	Intel MKL	41.02	1,648.5×	0.088×	19.49	8.83%

4 实验分析

• Python 实现

```

1 def matrix_multiply(A, B):
2     rows_a, cols_a = len(A), len(A[0])
3     rows_b, cols_b = len(B), len(B[0])
4     if cols_a != rows_b:
5         raise ValueError("Invalid matrix dimensions")
6     C = [[0.0 for _ in range(cols_b)] for _ in range(rows_a)]
7
8     start_time = time.time()
9     for i in range(rows_a):
10        for j in range(cols_b):
11            for k in range(cols_a):
12                C[i][j] += A[i][k] * B[k][j]
13    end_time = time.time()
14    print(f"Time taken to multiply two matrices: {(end_time - start_time) * 1000} milliseconds.")
15
16    return C

```

```

kobayashi@DESKTOP-5V2290B:~/Code$ python3 2_Computer_Science/Parallel/ppt/lab0/m_mul1.py
Time taken to multiply two matrices: 67627.90179252625 milliseconds.

```

图 1: Python 实现

• C/C++ 实现

```

1 void matrix_multiply(const vector<vector<double>>& A, const
    vector<vector<double>>& B, vector<vector<double>>& C) {

```

```

2   int rows_a = A.size();
3   int cols_a = A[0].size();
4   int rows_b = B.size();
5   int cols_b = B[0].size();
6
7   if (cols_a != rows_b) {
8       throw invalid_argument("Invalid matrix dimensions");
9   }
10
11  C.resize(rows_a, vector<double>(cols_b, 0.0));
12
13  auto start_time = chrono::high_resolution_clock::now();
14  for (int i = 0; i < rows_a; ++i) {
15      for (int j = 0; j < cols_b; ++j) {
16          for (int k = 0; k < cols_a; ++k) {
17              C[i][j] += A[i][k] * B[k][j];
18          }
19      }
20  }
21  auto end_time = chrono::high_resolution_clock::now();
22  auto duration = chrono::duration_cast<chrono::milliseconds>(
23      end_time - start_time);
24  cout << "Time taken to multiply two matrices: " << duration.
      count() << " milliseconds." << endl;
}

```

```

Time taken to multiply two matrices: 3626 milliseconds.
kobayashi@DESKTOP-5V2290B:~/Code$

```

图 2: C/C++ 实现

- 调整循环顺序

```

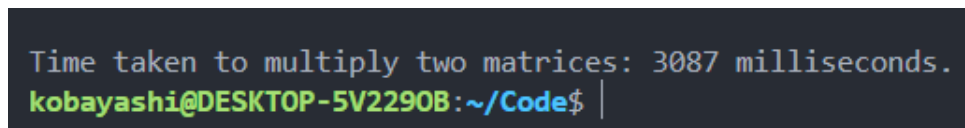
1 void matrix_multiply(const vector<vector<double>>& A, const
   vector<vector<double>>& B, vector<vector<double>>& C) {
2     int rows_a = A.size();
3     int cols_a = A[0].size();
4     int rows_b = B[0].size(); // change B from k*n to n*k
5     int cols_b = B.size();
6

```

```

7   if (cols_a != rows_b) {
8       throw invalid_argument("Invalid matrix dimensions");
9   }
10
11   C.resize(rows_a, vector<double>(cols_b, 0.0));
12
13   auto start_time = chrono::high_resolution_clock::now();
14   for (int i = 0; i < rows_a; ++i) {
15       for (int j = 0; j < cols_b; ++j) {
16           for (int k = 0; k < cols_a; ++k) {
17               C[i][j] += A[i][k] * B[j][k];
18           }
19       }
20   }
21   auto end_time = chrono::high_resolution_clock::now();
22   auto duration = chrono::duration_cast<chrono::milliseconds>(
23       end_time - start_time);
24   cout << "Time taken to multiply two matrices: " << duration.
25       count() << " milliseconds." << endl;
26 }

```



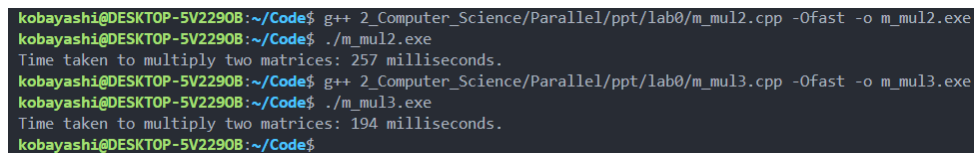
```

Time taken to multiply two matrices: 3087 milliseconds.
kobayashi@DESKTOP-5V2290B:~/Code$

```

图 3: 调整循环顺序

- 编译优化



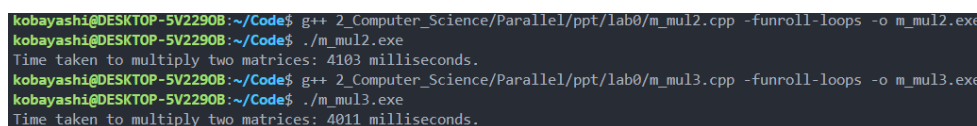
```

kobayashi@DESKTOP-5V2290B:~/Code$ g++ 2_Computer_Science/Parallel/ppt/lab0/m_mul2.cpp -Ofast -o m_mul2.exe
kobayashi@DESKTOP-5V2290B:~/Code$ ./m_mul2.exe
Time taken to multiply two matrices: 257 milliseconds.
kobayashi@DESKTOP-5V2290B:~/Code$ g++ 2_Computer_Science/Parallel/ppt/lab0/m_mul3.cpp -Ofast -o m_mul3.exe
kobayashi@DESKTOP-5V2290B:~/Code$ ./m_mul3.exe
Time taken to multiply two matrices: 194 milliseconds.
kobayashi@DESKTOP-5V2290B:~/Code$

```

图 4: 编译优化

- 循环展开



```

kobayashi@DESKTOP-5V2290B:~/Code$ g++ 2_Computer_Science/Parallel/ppt/lab0/m_mul2.cpp -funroll-loops -o m_mul2.exe
kobayashi@DESKTOP-5V2290B:~/Code$ ./m_mul2.exe
Time taken to multiply two matrices: 4103 milliseconds.
kobayashi@DESKTOP-5V2290B:~/Code$ g++ 2_Computer_Science/Parallel/ppt/lab0/m_mul3.cpp -funroll-loops -o m_mul3.exe
kobayashi@DESKTOP-5V2290B:~/Code$ ./m_mul3.exe
Time taken to multiply two matrices: 4011 milliseconds.

```

图 5: 循环展开

- Intel MKL

```
Computing matrix product using Intel(R) MKL dgemm function via CBLAS interface  
Computations completed in 41.02 milliseconds.
```

图 6: Intel MKL