

Parallel-Programming Task2

刘森元, 21307289

中山大学计算机学院

Codes on <https://github.com/Myocardial-infarction-Jerry/Parallel-Programming/tree/main/Task2>.

Project built by CMake.

```
1 | > cd Task2
2 | > cmake . && make
3 | > mpirun -np 8 MPIMatMul // Customize running
```

1 Environment

11th Gen Intel(R) Core(TM) i7-11700KF @ 3.60GHz

NVIDIA GeForce RTX 3080 Ti O12G

Windows Subsystem for Linux @ Ubuntu 22.04 LTS

2 Task

改进上次实验中的 MPI 并行矩阵乘法 (MPI-v1), 并讨论不同通信方式对性能的影响.

Note

输入: m, n, k 三个整数, 每个整数的取值范围均为 $[128, 2048]$.

问题描述: 随机生成 $m \times n$ 的矩阵 A 以及 $n \times k$ 的矩阵 B , 并对这两个矩阵进行矩阵乘法运算, 得到矩阵 C .

输出: A, B, C 三个矩阵, 及矩阵计算所消耗的时间 t .

要求:

1. 采用 MPI 集合通信实现并行矩阵乘法中的进程间通信; 使用 `MPI_Typecreate_struct` 聚合 MPI 进程内变量后进行通信; 尝试不同数据/任务划分方式 (选做).
2. 对于不同实现方式, 调整并记录不同线程数量 (1-16) 及矩阵规模 (128-2048) 下的时间开销, 填写下页表格, 并分析其性能及扩展性.

3 Theory

由于矩阵的可分性, 使用并行通用矩阵乘法的朴素思想, 即将矩阵分割为 $size$ 等份, 在多个线程中进行计算, 最后由 master 核进行拼接.

a_{00}	a_{01}	\cdots	$a_{0,n-1}$	$\begin{matrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{matrix}$	$=$	y_0
a_{10}	a_{11}	\cdots	$a_{1,n-1}$			y_1
\vdots	\vdots		\vdots			\vdots
a_{i0}	a_{i1}	\cdots	$a_{i,n-1}$			$y_i = a_{i0}x_0 + a_{i1}x_1 + \cdots a_{i,n-1}x_{n-1}$
\vdots	\vdots		\vdots			\vdots
$a_{m-1,0}$	$a_{m-1,1}$	\cdots	$a_{m-1,n-1}$			y_{m-1}

如上图的逐行计算, 对于 $m \times n$ 的矩阵 A , 可将其切分为若干个 $subM \times n$ 子矩阵分别与矩阵 B 相乘, 最后进行拼接, 其中 $subM = (m + size - 1)/size$, 保证均分.

4 Code

⚠ Caution

源代码详见 [MPIMatMul.cpp](#).

4.1 朴素矩阵乘法

```

1 void matMul(float *A, float *B, float *C, int m, int n, int k) {
2     for (int i = 0; i < m; i++) {
3         for (int j = 0; j < k; j++) {
4             C[i * k + j] = 0;
5             for (int l = 0; l < n; l++) {
6                 C[i * k + j] += A[i * n + l] * B[l * k + j];
7             }
8         }
9     }
10 }
```

4.2 进行矩阵划分

```

1 // Set the matrix dimensions
2 std::cin >> M >> n >> k;
3 std::cerr << "Calculating for m = " << M << ", n = " << n << ", k = " << k << std::endl;
4 std::cerr << "Running on " << size << " processes\n";
5
6 // Calculate the submatrix size for each process
7 m = (M + size - 1) / size;
8 std::cerr << "subM = " << m << std::endl;
```

对于矩阵使用零补全的方式保证整除.

4.3 数据聚合及传输

```

1 MPI_Datatype matrixSizeType;
2 MPI_Type_contiguous(3, MPI_INT, &matrixSizeType);
3 MPI_Type_commit(&matrixSizeType);
4
5 MatrixSize matrixSize;
6 if (rank == MASTER_RANK) {
```

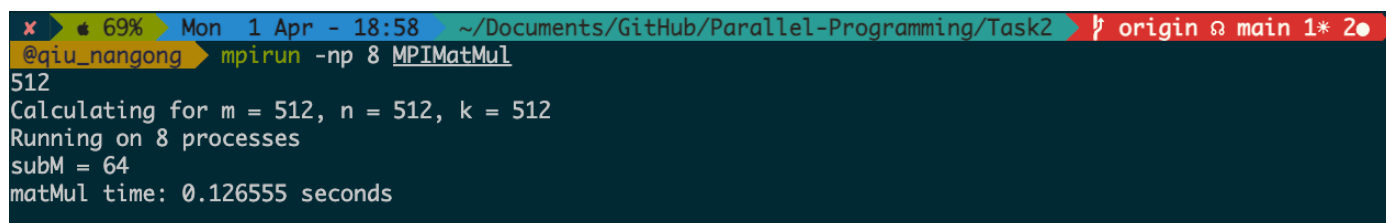
```

7     ...
8
9     // Set the dimensions of the matrixSize struct
10    matrixSize.m = m;
11    matrixSize.n = n;
12    matrixSize.k = k;
13 }
14
15 auto startTime = MPI_Wtime();
16
17 // Broadcast the matrixSize struct to all processes
18 MPI_Bcast(&matrixSize, 1, matrixSizeType, MASTER_RANK, MPI_COMM_WORLD);
19
20 // Update the local dimensions based on the received matrixSize
21 m = matrixSize.m;
22 n = matrixSize.n;
23 k = matrixSize.k;
24
25 // Allocate memory for the submatrices
26 float *subA = new float[m * n];
27 B = new float[n * k];
28 float *subC = new float[m * k];
29
30 // Scatter matrix A to all processes
31 MPI_Scatter(A, m * n, MPI_FLOAT, subA, m * n, MPI_FLOAT, MASTER_RANK, MPI_COMM_WORLD);
32
33 // Broadcast matrix B to all processes
34 MPI_Bcast(B, n * k, MPI_FLOAT, MASTER_RANK, MPI_COMM_WORLD);
35
36 // Perform matrix multiplication on the submatrices
37 matMul(subA, B, subC, m, n, k);
38
39 // Gather the submatrices C from all processes to the master process
40 MPI_Gather(subC, m * k, MPI_FLOAT, C, m * k, MPI_FLOAT, MASTER_RANK, MPI_COMM_WORLD);
41
42 auto endTime = MPI_Wtime();

```

5 Result

以 8 核心 512 矩阵为例



```

x 69% Mon 1 Apr - 18:58 ~/Documents/GitHub/Parallel-Programming/Task2 origin main 1* 2
@qiu_nangong mpirun -np 8 MPIMatMul
512
Calculating for m = 512, n = 512, k = 512
Running on 8 processes
subM = 64
matMul time: 0.126555 seconds

```

其中 `matMul time` 为并行矩阵乘法运行时间, `Running time` 为串行矩阵乘法运行时间.

进程数量/矩阵规模 (s)	128	256	512	1024	2048
1	0.00571813	0.0442355	0.378732	3.55125	30.4362
2	0.00325512	0.0249743	0.267912	2.40782	28.9294
4	0.00171017	0.0205275	0.14179	1.54117	20.3192
8	0.00107397	0.012676	0.114885	0.894698	12.0781
16	0.00573091	0.00768703	0.0707892	0.645633	12.8843

- 随着进程数量的增加, 程序的运行时间在大多数情况下都有所减少.
- 当进程数量增加到16时, 对于规模为 2048 的矩阵, 运行时间并没有显著减少. 这是因为进程间的通信开销开始超过了并行计算带来的收益, 即 Amdahl 定律.
- 随着矩阵规模的增加, 程序的运行时间也在增加. 矩阵乘法的计算复杂度为 $O(n^3)$, 所以当矩阵规模增加时, 所需的计算量也会显著增加.
- 程序在多进程下表现出了良好的性能提升, 但当进程数量增加到一定程度后, 通信开销可能会开始影响性能.