并行计算实验三

PB19030888 张舒恒

问题描述

输入一个稀疏矩阵(非零元比例<1%)，一个向量，求两者的乘积。参考现有的并行稀疏矩阵向量乘法实现，或者自行设计一种并行稀疏矩阵向量乘法实现。使用MPI实现。

算法设计

设矩阵M为$DIM \times DIM$矩阵，向量N为DIM维向量，运算结果P为DIM维向量，线程数为num。则可以将矩阵M划分为$num-1$个$(DIM/(num-1)) \times DIM$个条带子矩阵以及1个$(DIM\%(num-1)) \times DIM$剩余子矩阵。$MPI$开启$num$个线程，线程0先将向量$N$发送给每个线程，再将$num-1$个条带子矩阵分别分发给其余$num-1$个线程，然后从其余每个线程收取运算结果，最后计算剩余子矩阵与向量N乘积，将结果汇总到向量P中。其余$num-1$个线程接受0号线程发送的条带子矩阵和向量N，进行乘积运算后将运算结果传回0号线程。[1]伪代码如下：

```
line = dimension / (numprocs - 1);

thread 0:
    for i from 1 to numprocs:
        SEND N to other threads;
    for i from 1 to numprocs:
        SEND the submatrixes of M to other threads;
    for i from 1 to numprocs:
        RECEIVE the result subvectors from other threads;
    for i from (numprocs - 1) * line to dimension:
        CALCULATE the product of N and the remaining submatrixex;
    Output the results;

thread 1,2,...,numprocs:
    RECEIVE N from thread 0;
    RECEIVE the submatrixes of M from thread 0;
    CALCULATE the product of N and the submatrixes of M;
    SEND the result subvectors to thread 0;
```

考虑到本问题中矩阵M为稀疏矩阵，则可以仅记录矩阵M中非0元素及其位置，0号线程发送条带子矩阵也可以优化为发送非0元素，交给其他线程进行计算。

实验评测

实验配置

软硬件配置

CPU参数：

Processor (CPU)	
CPU Name	11th Gen Intel® Core™ i7-11600H @ 2.90GHz
Threading	1 CPU - 6 Core - 12 Threads
Frequency	4192.09 MHz (42 * 99.81 MHz) - Uncore: 3493.4 MHz
Multiplier	Current: 42 / Min: 8 / Max: 46
Architecture	Tiger Lake / Stepping: R0 / Technology: 10 nm
CPUID / Ext.	6.D.1 / 6.8D
IA Extensions	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, AES, AVX, AVX2, AVX512F, FMA3, SHA
Caches	L1D : 48 KB / L2 : 1280 KB / L3 : 18432 KB
Caches Assoc.	L1D : 12-way / L2 : 20-way / L3 : 12-way
Microcode	Rev. 0x34
TDP / Vcore	45 Watts / 1.117 Volts
Temperature	94 °C / 201 °F
Type	Retail
Cores Frequencies	#00: 4192.09 MHz #01: 4192.09 MHz #02: 4192.09 MHz #03: 4192.09 MHz #04: 4192.09 MHz #05: 4192.09 MHz

GPU参数：

Graphic Card (GPU)	
GPU #1 Type	NVIDIA GeForce RTX 3050 Laptop GPU (GA107) @ 1500 MHz
GPU #1 Brand	ASUSTeK Computer Inc.
GPU #1 VRAM	4096 MB @ 6001 MHz
GPU #2 Type	Intel(R) UHD Graphics
GPU #2 Brand	ASUSTeK Computer Inc.

编译器参数：

```
C:\Users\凝雨>gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=c:/mingw/bin/../libexec/gcc/mingw32/9.2.0/lto-wrapper.exe
Target: mingw32
Configured with: ../src/gcc-9.2.0/configure --build=x86_64-pc-linux-gnu --host=mingw32 --target=mingw32 --disable-win32-regi
stry --with-arch=i586 --with-tune=generic --enable-static --enable-shared --enable-threads --enable-languages=c,c++,objc,obj
-c++,fortran,ada --with-dwarf2 --disable-sjlj-exceptions --enable-version-specific-runtime-libs --enable-libgomp --disable-l
ibvtv --with-libiconv-prefix=/mingw --with-libintl-prefix=/mingw --enable-libstdcxx-debug --disable-build-format-warnings --
prefix=/mingw --with-gmp=/mingw --with-mpfr=/mingw --with-mpc=/mingw --with-isl=/mingw --enable-nls --with-pkgversion='MinGW
.org GCC Build-2'
Thread model: win32
gcc version 9.2.0 (MinGW.org GCC Build-2)
```

数据集配置

本实验采用随机生成的数据，为了控制矩阵稀疏程度即非零元比例$< 1\%$，先生成一个$0-1$的随机数，若此随机数小于0.01则生成非0随机值赋给矩阵元素，否则将矩阵元素设为0。

```
srand((unsigned)time(NULL));
float rand_num = (float)rand() / RAND_MAX;
for (int i = 0; i < N * N; ++i){
    if(rand_num < 0.01)
        matrix[i] = (float)rand();
    else
        matrix[i] = 0;
}
```

实验结果

正确性验证

MPI并行算法的结果正确性验证将由传统的串行两层循环代码来完成，这里给出验证的代码：

```
void gemm_baseline(float **A, float *B, float *C) {
    for (auto i = 0; i < N; i++){
        C[i] = 0.0;
        for (auto k = 0; k < N; k++)
            C[i] += A[i][k] * B[k];
    }
}

bool verify(float *A, float *B) {
    for (auto i = 0; i < N; i++)
        if (abs(A[i] - B[i]) > EPS){
             cout << "Error!" << endl;
             return 0;
         }
    return 1;
}
```

加速比分析

这里演示的是矩阵规模4000，线程数16时的运行结果，可以通过$verify = 1$看到结果的正确性。

```
zsh@LAPTOP-CU2KU731:/mnt/c/CA-lab5/bingxing$ mpirun -np 16 ./test 4000
The num of process is 16
The DIM is 4000
my_rank = 1 time = 0.292593s
my_rank = 2 time = 0.295203s
my_rank = 3 time = 0.295195s
my_rank = 4 time = 0.32213s
my_rank = 6 time = 0.335463s
my_rank = 5 time = 0.329898s
my_rank = 7 time = 0.353183s
my_rank = 8 time = 0.375276s
my_rank = 9 time = 0.376027s
my_rank = 10 time = 0.377872s
my_rank = 11 time = 0.379914s
my_rank = 12 time = 0.381134s
my_rank = 13 time = 0.381857s
my_rank = 15 time = 0.414553s
my_rank = 14 time = 0.396054s
my_rank = 0 time = 0.416061s
verify = 1
zsh@LAPTOP-CU2KU731:/mnt/c/CA-lab5/bingxing$ mpirun -np 2 ./test 1000
```

统计矩阵规模1000，2000，4000，8000，线程数2，4，8，16时的运行结果：

矩阵规模1000：

线程数	运行时间	相对于2线程的加速比
2	0.000156s	1
4	0.000245s	0.637
8	0.000214s	0.729
16	0.000230s	0.678

矩阵规模2000：

线程数	运行时间	相对于2线程的加速比
2	0.001451s	1
4	0.001476s	0.983
8	0.001390s	1.044
16	0.001383s	1.049

矩阵规模4000：

线程数	运行时间	相对于2线程的加速比
2	0.027819s	1
4	0.022952s	1.212
8	0.016648s	1.671
16	0.011212s	2.481

矩阵规模8000：

线程数	运行时间	相对于2线程的加速比
2	1.606828s	1
4	1.163524s	1.381
8	0.627667s	2.560
16	0.416061s	3.862

矩阵规模较小时加速效果不明显，甚至有负加速的情况出现，这主要的原因是MPI通信开销大于线程并行所节省的时间开销。当矩阵规模增大后，开始出现一定的加速效果，原因是随着矩阵规模增大，运算时间占总时间的比重增大，这时采取多线程并行将会降低运行时间。

参考文献

[1] Chapter 27 - Sparse Matrix-Vector Multiplication: Parallelization and Vectorization