高性能计算应用实践

Lab6 实验报告

朱祉睿 220110501 大二秋

1. 朴素矩阵乘

```c
/* Create macros so that the matrices are stored in column-major order */

#define A(i,j) a[ (j)*lda + (i) ]
#define B(i,j) b[ (j)*ldb + (i) ]
#define C(i,j) c[ (j)*ldc + (i) ]

/* Routine for computing C = A * B + C */
// void print_row_matrix( int m, int n, double *a, int lda );
// void print_matrix( int m, int n, double *a, int lda );

void MY_MMult( int m, int n, int k, double *a, int lda,
                                    double *b, int ldb,
                                    double *c, int ldc )
{
  int i, j, p;
  // print_row_matrix(m,k,a,lda);
  // print_row_matrix(k,n,b,ldb);

  for ( i=0; i<m; i++ ){        /* Loop over the rows of C */
    for ( j=0; j<n; j++ ){        /* Loop over the columns of C */
    // C( i,j ) = 0;
      for ( p=0; p<k; p++ ){        /* Update C( i,j ) with the inner
                                       product of the ith row of A and
                                       the jth column of B */
        C( i,j ) =  C( i,j ) +  A( i,p ) * B( p,j );
      }
    }
  }
  // print_row_matrix(m,n,c,ldc);
}
```

2. openblas

调用 openblas 库函数 cblas_dgemm 实现矩阵乘

```c
#include <cblas.h>
void MY_MMult( int m, int n, int k, double *a, int lda,
                                    double *b, int ldb,
                                    double *c, int ldc )
{
        cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,n,n,n,1,a, n, b, n,1,c,n);
}


```

3. pthread

调用 pthread 库函数，多线程实现矩阵乘

```c
#include <assert.h>
#include <pthread.h>
#include <unistd.h>
typedef struct
{
    double *A;
    double *B;
    double *C;
    int alow;
    int ahigh;
    int blow;
    int bhigh;
    int k;
    int n;
}agv;

void *pthread(void *arg)
{
    agv *P = (agv *)arg;
    for (int i = P->alow-1; i < P->ahigh; i++)
    {
        for (int j = P->blow; j < P->bhigh; j++)
        {
            P->C[i*P->k+j] = 0;
            for (int p = 0; p < P->k; p++)
            {
                    P->C[i*P->k+j] += P->A[i*P->k+p] * P->B[p*P->n+j];
            }
        }
    }
}//double *A,double *B,double *C,int alow,int ahigh,int blow,int bhigh,int k ,int n

void MY_MMult( int m, int n, int k, double *a, int lda,
                                    double *b, int ldb,
                                    double *c, int ldc )
{
    int i=m/2;
    int rc;
    pthread_t  p1, p2;
    agv c1={a,b,c,1,i,1,n,k,n};
    agv c2={a,b,c,i+1,m,1,n,k,n};
    rc = pthread_create(&p1, NULL, pthread, &c1); assert(rc == 0);
    rc = pthread_create(&p2, NULL, pthread, &c2); assert(rc == 0);
```

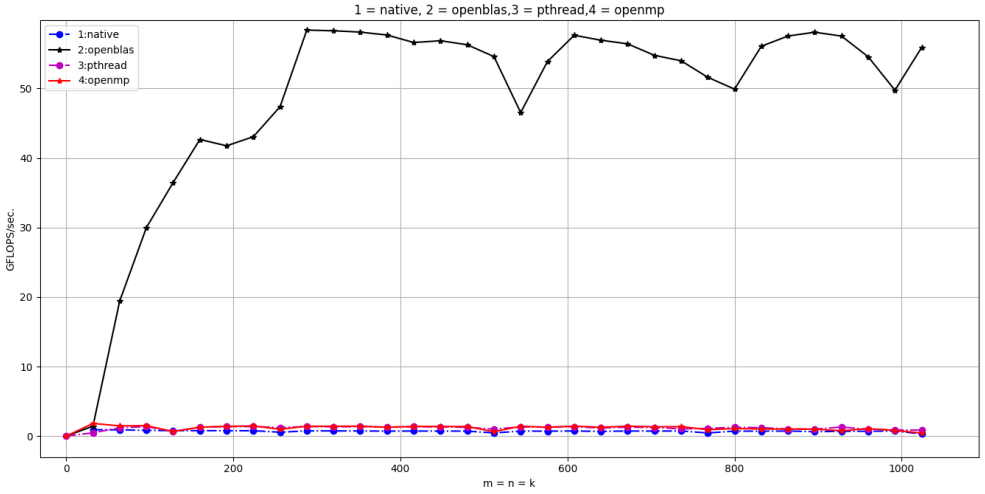
4. openmp

运用 openmp 多线程实现矩阵乘

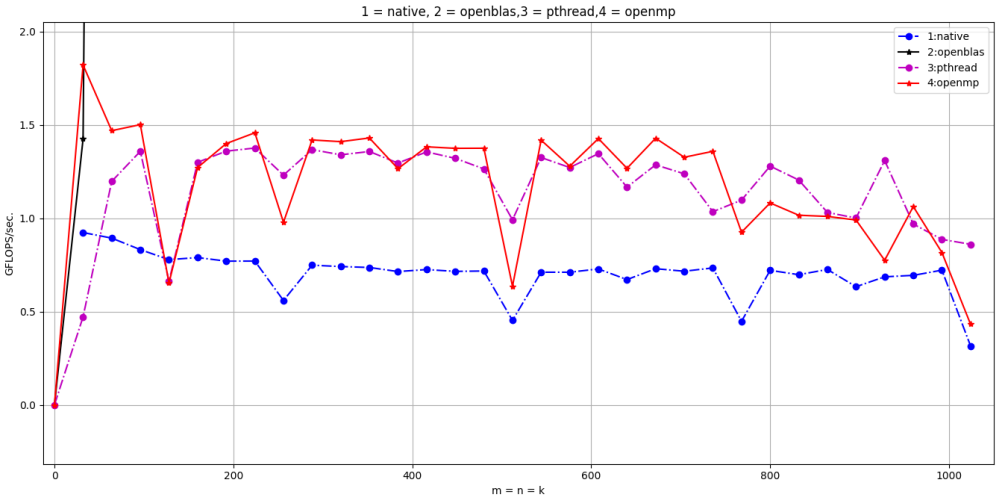
```c
#include <omp.h>
#define A(i,j) a[ (j)*lda + (i) ]
#define B(i,j) b[ (j)*ldb + (i) ]
#define C(i,j) c[ (j)*ldc + (i) ]
void MY_MMult( int m, int n, int k, double *a, int lda,
                                    double *b, int ldb,
                                    double *c, int ldc )
{
  #pragma omp parallel for
  for (int i=0; i<m; i++ ){        /* Loop over the rows of C */
    for (int j=0; j<n; j++ ){        /* Loop over the columns of C */
    // C( i,j ) = 0;
      for (int p=0; p<k; p++ ){
              C( i,j ) =  C( i,j ) +  A( i,p ) * B( p,j );
      }
    }
  }
}

```

5. gflops 曲线图



放大版



由图可见，调用 openblas 库实现矩阵乘的效率远大于其他方式，而运用 pthread 和 openmp 多线程实现矩阵乘的效率几乎相同，且都为朴素矩阵乘的两倍（线程数为 2）.

6.碰到的问题

1.在运用 openmp 时，cpu 占用率一直小于 100%

解决：在 makefile 中环境变量 OMP_NUM_THREADS 被设为 1，故一直是单线程，改为 2 即可

2.用 export 命令设置环境变量时，此修改只对当前终端有效，对其它终端和一个新的终端均无效。

3.想将四种矩阵乘方法画在一张图中

解决：修改 makefile 使之产生四个 output.m 文件，修改 PlotAll.py 使之画出四条曲线，并修改为不同颜色。