Lab 6

Infomations

- 课程名:高性能计算
- 学期:2024秋
- 姓名:凌建杰
- 学号:2023311C08

Environment

- OS版本:5.15.153.1-microsoft-standard-WSL2
- gcc版本: 11.4.0
- CPU:
  - 型号:AMD Ryzen 7 6800H with Radeon Graphics
  - 频率:3.2 GHz
  - 物理核数:8

Methods

1. `Native`

- 无需多言,简单矩阵乘法

```
C(i, j) = C(i, j) + A(i, p) * B(p, j);
```

2. `Cblas`

- 调用 `cblas_dgemm` 即可

```
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, m, n, k, 1, a, lda, b, ldb, 1,
c, ldc);
```

3. `pthread`

- 按 `PTHREAD_NUM` 切分原矩阵
- 子线程内,按2*2,4*4或8*8的矩阵块计算

```
  pthread_t *threads;
  myarg_t *args;
  threads=(pthread_t *)malloc(PTHREAD_NUM*sizeof(pthread_t));
  args=(myarg_t *)malloc(PTHREAD_NUM*sizeof(myarg_t));
  // init pthread
  for (int i = 0; i < PTHREAD_NUM; i++)
  {
```

```
    pthread_t p=i;
    threads[i] = p;
  }
  int len = (int)sqrt((double)PTHREAD_NUM);
  if (len % 2)
  {
    printf("Thread num must be the square of 4!\n");
    return;
  }
  for (int i = 0; i < len; i++)
  {
    for (int j = 0; j < len; j++)
    {
      myarg_t arg = {a, b, c, (m / len) * (i + 1), (n / len) * (j + 1), k, lda, ldb,
ldc, (m / len) * (i + 0), (n / len) * (j + 0)};
      args[i * len + j] = arg;
    }
  }
  // create
  for (int i = 0; i < PTHREAD_NUM; i++)
  {
    if (m / len <= 2)
    {
      Pthread_create(&threads[i], NULL, MY_MMult_2, &args[i]);
    }
    else if (m / len <= 4)
    {
      Pthread_create(&threads[i], NULL, MY_MMult_4, &args[i]);
    }
    else if (m / len <= 8)
    {
      Pthread_create(&threads[i], NULL, MY_MMult_8, &args[i]);
    }
    else
    {
      Pthread_create(&threads[i], NULL, MY_MMult_8, &args[i]);
    }
  }
  // join
  for (int i = 0; i < PTHREAD_NUM; i++)
    Pthread_join(threads[i], NULL);
  free(threads);
  free(args);
```

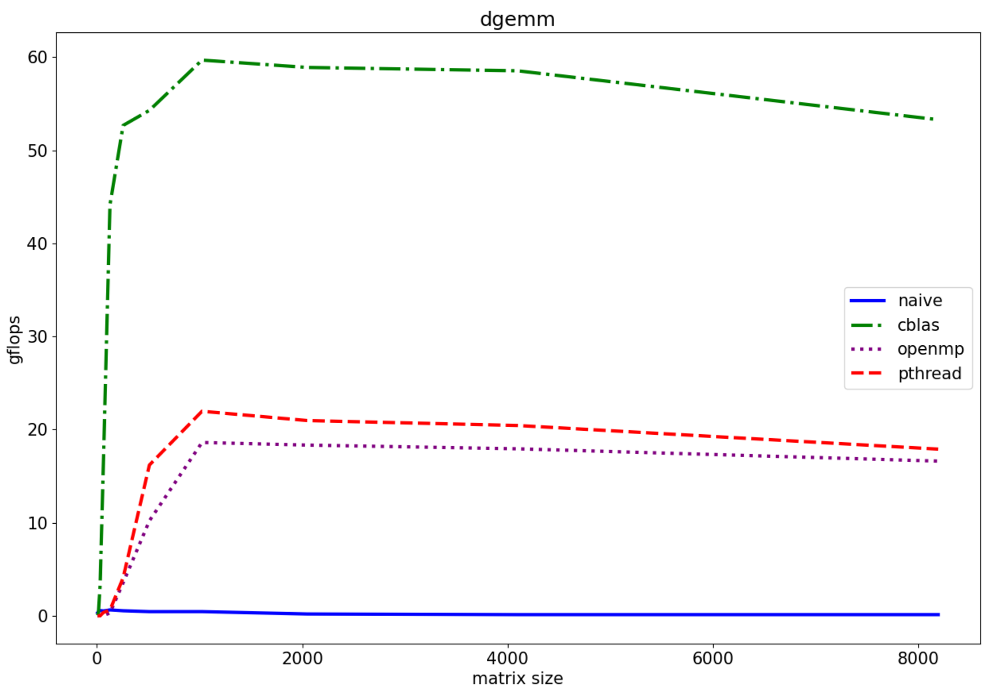
4. `OpenMP`

- 按8*8的块计算,openmp使用collapse(2),将外层两个循环合并.

```
int i, j, p, r, s;
    if (m < 8 || n < 8 || k < 8)
    {
        printf("Matrix can't small than 8*8!\n");
        return ;
    }
    int len = 8;
    // int len = m >= 8 ? 8 : BLOCK_LEN;
```

```
    fprintf(stderr, __FILE__ "\n");
    // printf("start\n");
#pragma omp parallel for collapse(2) private(i, j, p) shared(a,b,c) schedule(static)
    for (i = 0; i < m; i += len) /* Loop over the rows of C */
    {
        for (j = 0; j < n; j += len) /* Loop over the columns of C */
        {
            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 + 0, j + 0) = C(i + 0, j + 0) + A(i + 0, p) * B(p, j + 0);
                C(i + 0, j + 1) = C(i + 0, j + 1) + A(i + 0, p) * B(p, j + 1);
                C(i + 0, j + 2) = C(i + 0, j + 2) + A(i + 0, p) * B(p, j + 2);
                C(i + 0, j + 3) = C(i + 0, j + 3) + A(i + 0, p) * B(p, j + 3);
                C(i + 0, j + 4) = C(i + 0, j + 4) + A(i + 0, p) * B(p, j + 4);
                ...
            }
        }
    }
```

Result



OpenMP

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 6268 root      20   0 2920168   2.0g   4556 R  95.3  27.5   0:17.77 test_MMult.x
 6269 root      20   0 2920168   2.0g   4556 S  88.0  27.5   0:15.46 test_MMult.x
 6278 root      20   0 2920168   2.0g   4556 S  88.0  27.5   0:14.82 test_MMult.x
 6274 root      20   0 2920168   2.0g   4556 S  87.4  27.5   0:14.90 test_MMult.x
 6277 root      20   0 2920168   2.0g   4556 S  87.4  27.5   0:15.50 test_MMult.x
 6273 root      20   0 2920168   2.0g   4556 S  87.0  27.5   0:14.76 test_MMult.x
 6282 root      20   0 2920168   2.0g   4556 S  86.7  27.5   0:14.73 test_MMult.x
 6272 root      20   0 2920168   2.0g   4556 S  86.0  27.5   0:15.48 test_MMult.x
 6280 root      20   0 2920168   2.0g   4556 S  86.0  27.5   0:14.69 test_MMult.x
 6271 root      20   0 2920168   2.0g   4556 S  85.0  27.5   0:14.68 test_MMult.x
 6283 root      20   0 2920168   2.0g   4556 S  83.7  27.5   0:14.73 test_MMult.x
 6279 root      20   0 2920168   2.0g   4556 S  83.1  27.5   0:14.75 test_MMult.x
 6270 root      20   0 2920168   2.0g   4556 S  82.7  27.5   0:15.04 test_MMult.x
 6276 root      20   0 2920168   2.0g   4556 S  82.7  27.5   0:14.71 test_MMult.x
 6275 root      20   0 2920168   2.0g   4556 S  82.1  27.5   0:14.66 test_MMult.x
```

```
   PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
145792 root      20   0  954056 671224   4596 R  1600   8.7   2:11.95 test_MMult.x
```

Pthread

```
   PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
154051 root      20   0 3051448   2.5g   4724 S  1559  34.3   5:00.34 test_MMult.x
   529 root      20   0   21.5g 304152  46560 S   1.7   3.9   7:05.16 node
```

```
        └─11*[{node}]
 ├─node─┬─4*[zsh]
        ├─zsh───top
        ├─zsh───make───sh───test_MMult.x───31*[{test_MMult.x}]
        ├─zsh───pstree
        └─17*[{node}]
```

Problems

> 多个c代码中有相同的MY_MMult函数，怎么判断可执行文件调用的是哪个版本的MY_MMult函数？是makefile中的哪行代码决定的？

- `OBJ` 中的 `NEW` 变量代表了要编译的版本,因此调用 `NEW` 版本的MY_MMult函数

> 性能数据_data/output_MMult0.m是怎么生成的？c代码中只是将数据输出到终端并没有写入文件。

- 通过shell指令将 `test_MMult.x` 可执行文件的结果写入了 `output_$(NEW).m` 中,并将内容复制到 `output_new.m` 里