分布式系统作业

第 5 次作业

姓名：唐晨轩

班级：人工智能与大数据

学号：19335182

一、 问题描述

1. Consider a simple loop that calls a function `dummy` containing a programmable delay （sleep）. All invocations of the function are independent of the others. Partition this loop across four threads using `static`, `dynamic`, and `guided` scheduling. Use different parameters for static and guided scheduling. Document the result of this experiment as the delay within the `dummy` function becomes large.

2. Implement a producer-consumer framework in OpenMP using sections to create a single producer task and a single consumer task. Ensure appropriate synchronization using locks. Test your program for a varying number of producers and consumers.

3. Consider a sparse matrix stored in the compressed row format (you may find a description of this format on the web or any suitable text on sparse linear algebra). Write an OpenMP program for computing the product of this matrix with a vector. Download sample matrices from the Matrix Market (http://math.nist.gov/MatrixMarket/) and test the performance of your implementation as a function of matrix size and number of threads.

二、 解决方案

1.

1. Consider a simple loop that calls a function `dummy` containing a programmable delay (sleep) . All invocations of the function are independent of the others. Partition this loop across four threads using `static`, `dynamic`, and `guided` scheduling. Use different parameters for static and guided scheduling. Document the result of this experiment as the delay within the `dummy` function becomes large.

具体代码见 static.cpp，dynami.cpp 和 guided.cpp。

大致思路是，设计了一个 12 次的循环，在循环中调用了 dummy 造成适当延迟。观察三种调度函数的结果。

其中线程数量设置为 4，chunksize 设置为 1/2。

运行结果如下：

Static 调度策略：

当线程数目设置为 4 时：

(1) Chunksize = 1

```
hadoop@ubuntu:~/hw5$ g++ -fopenmp static.cpp -o static
hadoop@ubuntu:~/hw5$ ./static
i = 3 , This is thread 3 of 4 .
i = 2 , This is thread 2 of 4 .
i = 1 , This is thread 1 of 4 .
i = 0 , This is thread 0 of 4 .
i = 6 , This is thread 2 of 4 .
i = 5 , This is thread 1 of 4 .
i = 7 , This is thread 3 of 4 .
i = 4 , This is thread 0 of 4 .
i = 9 , This is thread 1 of 4 .
i = 10 , This is thread 2 of 4 .
i = 11 , This is thread 3 of 4 .
i = 8 , This is thread 0 of 4 .
hadoop@ubuntu:~/hw5$
```

(2) Chunksize = 2

```
hadoop@ubuntu:~/hw5$ g++ -fopenmp static.cpp -o static
hadoop@ubuntu:~/hw5$ ./static
i = 4 , This is thread 2 of 4 .
i = 0 , This is thread 0 of 4 .
i = 6 , This is thread 3 of 4 .
i = 2 , This is thread 1 of 4 .
i = 1 , This is thread 0 of 4 .
i = 3 , This is thread 1 of 4 .
i = 7 , This is thread 3 of 4 .
i = 5 , This is thread 2 of 4 .
i = 8 , This is thread 0 of 4 .
i = 10 , This is thread 1 of 4 .
i = 9 , This is thread 0 of 4 .
i = 11 , This is thread 1 of 4 .
hadoop@ubuntu:~/hw5$
```

经过观察发现：

Chunksize=1 时：静态调度策略，按轮转的方式，每次把 1 个循环分配给 1 个线程，所以 i = 0/4/8 是线程 0 完成的，i = 1/5/9 是线程 1 完成的，i = 2/6/10 是线程 2 完成的，i = 3/7/11 是线程 3 完成的。

Chunksize=2 时：静态调度策略，按轮转的方式，每次把 2 个循环非陪给 1 个线程，所以 i = 0/1/8/9 是线程 0 完成的，i = 2/3/10/11 是线程 1 完成的，i = 4/5 是线程 2 完成的，i = 6/7 是线程 3 完成的。

Dynamic 调度策略：

当线程数目设置为 4 时：

(1) Chunksize = 1

```
hadoop@ubuntu:~/hw5$ g++ -fopenmp dynamic.cpp -o dynamic
hadoop@ubuntu:~/hw5$ ./dynamic
i = 0 , This is thread 3 of 4 .
i = 1 , This is thread 2 of 4 .
i = 2 , This is thread 1 of 4 .
i = 3 , This is thread 2 of 4 .
i = 4 , This is thread 1 of 4 .
i = 5 , This is thread 3 of 4 .
i = 6 , This is thread 2 of 4 .
i = 7 , This is thread 1 of 4 .
i = 8 , This is thread 3 of 4 .
i = 9 , This is thread 2 of 4 .
i = 10 , This is thread 1 of 4 .
i = 11 , This is thread 3 of 4 .
hadoop@ubuntu:~/hw5$
```

(2) Chunksize = 2

```
hadoop@ubuntu:~/hw5$ g++ -fopenmp dynamic.cpp -o dynamic
hadoop@ubuntu:~/hw5$ ./dynamic
i = 0 , This is thread 1 of 4 .
i = 2 , This is thread 2 of 4 .
i = 4 , This is thread 3 of 4 .
i = 6 , This is thread 0 of 4 .
i = 3 , This is thread 2 of 4 .
i = 1 , This is thread 1 of 4 .
i = 5 , This is thread 3 of 4 .
i = 7 , This is thread 0 of 4 .
i = 8 , This is thread 2 of 4 .
i = 10 , This is thread 1 of 4 .
i = 9 , This is thread 2 of 4 .
i = 11 , This is thread 1 of 4 .
hadoop@ubuntu:~/hw5$
```

经过观察发现：

Chunksize=1 时：动态调度策略，每次分配 1 个循环给当前空闲的线程，这种调度策略下的运行结果时不可预测的。

Chunksize=2 时：动态调度策略，每次分配 2 个循环给当前空闲的线程，这种调度策略下的运行结果时不可预测的。

Guided 调度策略：（较特殊所以修改了 chunksize 大小）

当线程数目设置为 4 时：

(1) Chunksize = 2

```
hadoop@ubuntu:~/hw5$ g++ -fopenmp guided.cpp -o guided
hadoop@ubuntu:~/hw5$ ./guided
i = 0 , This is thread 3 of 4 .
i = 6 , This is thread 2 of 4 .
i = 8 , This is thread 1 of 4 .
i = 3 , This is thread 0 of 4 .
i = 7 , This is thread 2 of 4 .
i = 9 , This is thread 1 of 4 .
i = 1 , This is thread 3 of 4 .
i = 4 , This is thread 0 of 4 .
i = 10 , This is thread 1 of 4 .
i = 2 , This is thread 3 of 4 .
i = 11 , This is thread 1 of 4 .
i = 5 , This is thread 0 of 4 .
hadoop@ubuntu:~/hw5$
```

(2) Chunksize = 4

```
hadoop@ubuntu:~/hw5$ g++ -fopenmp guided.cpp -o guided
hadoop@ubuntu:~/hw5$ ./guided
i = 0 , This is thread 1 of 4 .
i = 4 , This is thread 0 of 4 .
i = 8 , This is thread 3 of 4 .
i = 5 , This is thread 0 of 4 .
i = 9 , This is thread 3 of 4 .
i = 1 , This is thread 1 of 4 .
i = 6 , This is thread 0 of 4 .
i = 10 , This is thread 3 of 4 .
i = 2 , This is thread 1 of 4 .
i = 7 , This is thread 0 of 4 .
i = 11 , This is thread 3 of 4 .
i = 3 , This is thread 1 of 4 .
hadoop@ubuntu:~/hw5$
```

经过观察发现：

Chunksize=2 时：guided 调度策略，被分配的块的大小就是 chunksize（除最后一块），chunksize 为 2，所以第一个块的大小是 2，i = 0/1 被分配给了线程 3，当块完成后新块（此线程负责的新块）的大小变为原来的一半（1），线程 3 收到的下一个块就是 i = 2，接下来的 i = 3/4 被分配给了线程 0...

Chunksize=4 时：guided 调度策略，被分配的块的大小就是 chunksize（除最后一块），chunksize 为 4，所以第一个快的大小是 4，i = 0/1/2/3 被分配给了线程 1，i = 4/5/6/7

被分配给了线程 0，i = 8/9/10/11 被分配给了线程 3。

2.

2. Implement a producer-consumer framework in OpenMP using sections to create a single producer task and a single consumer task. Ensure appropriate synchronization using locks. Test your program for a varying number of producers and consumers.

单个生产者单个消费者：

Sleep 函数的作用是避免生产过快，导致生产结束了，消费的线程还没有开始创造，最终并行程序呈现出串行程序的效果。

全局变量：

```
vector<int> item;
omp_lock_t mylock;
const int number = 10;
int num = 0;
```

函数调用：

```
int main(){
    #pragma omp parallel sections
    {
        #pragma omp section
        producer();
        #pragma omp section
        consumer();
    }
    return 0;
}
```

生产者函数：

```
void producer(){
    for(int i = 0; i < number; i++){
        omp_set_lock(&mylock);
        item.push_back(num);
        printf("item %d was produced.\n",num);
        num ++;
        omp_unset_lock(&mylock);
        sleep(1);
    }
}
```

消费者函数：

```
void consumer(){
    for(int i = 0; i < number; i++){
        sleep(1);
        if(item.size() != 0){
            omp_set_lock(&mylock);
            printf("item %d was comsumed.\n",item[item.size()-1]);
            item.pop_back();
            omp_unset_lock(&mylock);
        }
        else printf("there is no item.\n");
    }
}
```

运行结果：（经测试，不用锁会出现段错误）

```
hadoop@ubuntu:~/hw5$ g++ -fopenmp p-c.cpp -o p-c
hadoop@ubuntu:~/hw5$ ./p-c
item 0 was produced.
item 0 was comsumed.
item 1 was produced.
item 1 was comsumed.
item 2 was produced.
item 2 was comsumed.
item 3 was produced.
item 3 was comsumed.
item 4 was produced.
item 4 was comsumed.
item 5 was produced.
item 6 was produced.
item 6 was comsumed.
item 7 was produced.
item 7 was comsumed.
item 5 was comsumed.
item 8 was produced.
item 8 was comsumed.
item 9 was produced.
item 9 was comsumed.
hadoop@ubuntu:~/hw5$
```

```
hadoop@ubuntu:~/hw5$ ./p-c
item 0 was produced.
item 0 was consumed.
item 1 was produced.
item 1 was consumed.
item 2 was produced.
item 2 was consumed.
item 3 was produced.
item 3 was consumed.
item 4 was produced.
item 5 was produced.
item 5 was consumed.
item 4 was consumed.
item 6 was produced.
item 6 was consumed.
item 7 was produced.
item 8 was produced.
item 8 was consumed.
item 9 was produced.
item 9 was consumed.
item 7 was consumed.
hadoop@ubuntu:~/hw5$
```

```
hadoop@ubuntu:~/hw5$ ./p-c
item 0 was produced.
item 0 was consumed.
item 1 was produced.
item 1 was consumed.
item 2 was produced.
item 2 was consumed.
item 3 was produced.
item 4 was produced.
item 4 was consumed.
item 3 was consumed.
item 5 was produced.
item 6 was produced.
item 6 was consumed.
item 5 was consumed.
item 7 was produced.
item 7 was consumed.
item 8 was produced.
item 8 was consumed.
item 9 was produced.
item 9 was consumed.
hadoop@ubuntu:~/hw5$
```

多个生产者多个消费者:

多添加了几个并行任务

函数调用:

```cpp
int main(){
    #pragma omp parallel sections
    {
        #pragma omp section
        producer();
        #pragma omp section
        consumer();
        #pragma omp section
        producer();
        #pragma omp section
        consumer();
    }
    return 0;
}
```

运行结果：（数字太大不好截图，所以我把 number 从 10 修改为 8）

```
hadoop@ubuntu:~/hw5$ g++ -fopenmp p-c.cpp -o p-c
hadoop@ubuntu:~/hw5$ ./p-c
item 0 was produced.
item 0 was comsumed.
item 1 was produced.
item 2 was produced.
item 2 was comsumed.
item 3 was produced.
item 3 was comsumed.
item 1 was comsumed.
item 4 was produced.
item 4 was comsumed.
item 5 was produced.
item 5 was comsumed.
item 6 was produced.
item 6 was comsumed.
item 7 was produced.
item 7 was comsumed.
item 8 was produced.
item 9 was produced.
item 9 was comsumed.
item 8 was comsumed.
item 10 was produced.
item 10 was comsumed.
item 11 was produced.
item 11 was comsumed.
item 12 was produced.
item 13 was produced.
item 13 was comsumed.
item 14 was produced.
item 14 was comsumed.
item 12 was comsumed.
item 15 was produced.
item 15 was comsumed.
hadoop@ubuntu:~/hw5$
```

```
hadoop@ubuntu:~/hw5$ ./p-c
item 0 was produced.
item 0 was consumed.
item 1 was produced.
item 2 was produced.
item 2 was consumed.
item 3 was produced.
item 3 was consumed.
item 4 was produced.
item 4 was consumed.
item 1 was consumed.
item 5 was produced.
item 5 was consumed.
item 6 was produced.
item 7 was produced.
item 7 was consumed.
item 6 was consumed.
item 8 was produced.
item 8 was consumed.
item 9 was produced.
item 10 was produced.
item 10 was consumed.
item 11 was produced.
item 11 was consumed.
item 9 was consumed.
item 12 was produced.
item 12 was consumed.
item 13 was produced.
item 13 was consumed.
item 14 was produced.
item 15 was produced.
item 15 was consumed.
item 14 was consumed.
hadoop@ubuntu:~/hw5$
```

可以看到，无论是针对单个生产者单个消费者，或者多个生产者多个消费者，程序的执行都没有出错，互斥锁起到了很好的效果。

3.

3. Consider a sparse matrix stored in the compressed row format (you may find a description of this format on the web or any suitable text on sparse linear algebra). Write an OpenMP program for computing the product of this matrix with a vector. Download sample matrices from the Matrix Market (http://math.nist.gov/MatrixMarket/) and test the performance of your implementation as a function of matrix size and number of threads.

先从 Matrix Market 网站上下载对应的 1138 BUS 的 mtx.gz 文件并且解压为 mtx 文件:

1138_bus.mtx	1996/10/5 7:22	Manual Test Tex...	74 KB
1138_bus.mtx.gz	2021/11/28 11:56	WinRAR 压缩文件	21 KB

大致思路为:

先读取 1138_bus.mtx 文件，得到矩阵。然后通过 openmp 并行编程，完成矩阵乘向量的运算。关键代码如下:

```
omp_lock_t mylock;
int sum = 0;

int main(int argc, char *argv[])
{
    vector<vector<double> > m;  //假设m存放的是存进来的矩阵 x*y
    vector<double> v;           //v为向量 y*1
    vector<double> ans;         //存放结果 x*1

    int m_x = m.size();        //矩阵宽度
    int m_y = m[0].size();     //矩阵长度
    int v_l = v.size();        //向量长度

    #pragma omp parallel for num_threads(4) private(i)
    for(int i = 0; i < m_x; i++){
        omp_set_lock(&mylock);
        for(int j = 0; j < m_y; j++){
            for(int k = 0; k < v_l; k++){
                sum += m[i][j]*v[k];
            }
        }
        ans.push_back(sum);
        sum = 0;
        omp_unset_lock(&mylock);
    }
}
```

遗憾的是我没有完成 m 矩阵的读入，所以这个作业没有完全完成。