并行与分布式作业

Foster 并行程序设计方法

第四次作业

姓名：TRY

班级：18 级计科 7 班

学号：

一、 问题描述

利用 Foster 并行程序设计方法计算 1000x1000 的矩阵与 1000x1 的向量之间的乘积，要求清晰地呈现 Foster 并行程序设计的四个步骤。

二、 解决方案

1. Foster 并行程序方法介绍

Foster 并行程序设计方法分为 4 步：

- Partition 划分
  - 分类：任务划分、数据划分
  - 标准：原子任务至少比 CPU 数量多 10 倍，且任务大小均衡

- Communication 通信
  - 分类：本地通信、局部通信
  - 标准：通信尽量减少、任务之间通信减少、并发通信、并发计算

- Agglomeration 组合
  - 把小任务归并为大任务
  - 目标：提高性能、保持可扩展性、简化程序
  - 标准：提高局部性，扩展性好，减少通信损耗

- Mapping 映射
  - 为处理器分配任务

2.设计矩阵乘向量的程序

本次实验，我利用 OpenMP 来进行编程。一开始，也有考虑过别的方法，如 pthread\mpi 等，但由于老师曾在课上详细讲过 OpenMP,所以打算用 OpenMP 进行练习。

而由 OpenMP 来设计矩阵和向量之间的乘法的多线程并行程序并不难，只需要加上“pragma omp parallel for num_threads(ThreadNumber)”即可，且由于 OpenMP 是用共享内存的编程框架来进行多线程并发，所以不存在数据依赖，不需要通信，十分方便。

代码详细如下：

```cpp
#include <iostream>
#include <omp.h>
#include<time.h>
using namespace std;

#define size 1000
#define ThreadNumber 8

int firstMatrix[size][size];//矩阵
int secondMatrix[size][1];//向量
int resultMatrix[size][1];//结果

void calculate(int row, int col)//计算单元的乘积
{
    int result=0;
    for (int i = 0; i < size; i++)
    {
        result += firstMatrix[row][i] * secondMatrix[i][col];
    }
    resultMatrix[row][col] = result;
}

void matrixInit()//矩阵、向量初始化
{
    #pragma omp parallel for num_threads(ThreadNumber)//并行
    for (int row = 0; row < size; row++) {
        for (int col = 0; col < size; col++)
            firstMatrix[row][col] = rand() % 10;
        secondMatrix[row][0] = rand() % 10;
    }
}

void matrixMulti()//矩阵乘法
{
    #pragma omp parallel for num_threads(ThreadNumber)//并行
    for (int row = 0; row < size; row++)
    {
        for (int col = 0; col < size; col++)
            calculate(row, col);
    }
}

int main()
{
    matrixInit();
    clock_t t1 = clock();
    matrixMulti();
    clock_t t2 = clock();
    cout << "time:" << t2 - t1;
    /* ... */
    system("pause");
    return 0;
}
```

三、 实验结果

以下将按照“Foster 并行程序设计方法”来分析上面的程序。

1. 分离（Partitioning）

对于一个 1000×1000 的矩阵和一个 1000×1 的向量相乘，结果是一个 1000×1 的向量。而对于结果向量的每一个分量来说，它是分别相乘，并将 1000 个值相加。所以，分离就是将结果向量的每一个分量分成 1000 个乘积来处理，每个乘积作为一个任务，最后相加得到一个分量的结果。

2. 通信（Communication）

对于 OpenMP 来说，实现通过共享内存，且这个问题本身比较简单，各计算任务间没有数据依赖，故不需要通信。且题目没有说民要保存在原来的 vector 中，所以可以把结果保存在一个新的数组 result 中。在对 result[i]的更新过程，并不牵扯到其他子任务的计算，所以任务之间没有通信。

3. 聚合（Agglomeration）

对于一个 1000×1000 的矩阵和一个 1000×1 的向量相乘，若按照每个分量 1000 个任务来处理，他们之间是存在数据竞争的，所以，要把每个分量的 1000 个乘积聚合起来，成为一个任务。

并且，若按照矩阵的行划分任务，则共有 1000 个任务，这大大超出了计算机上 CPU 的数量。所以，为了平衡 CPU 负载的问题，我们将多个行与向量相乘的任务合并为一个任务，以便将任务分配给 CPU。因为各个分量的计算没有数据依赖，所以这种合并可以是任意的。但考虑到负载均衡性和数据局部性，可以将连续的行均分到合并后的任务中。

4. 映射（Mapping）

我的计算机有 8 个 CPU 核心。使用 OpenMP 将每个计算任务映射为一个线程，共 8 个线程，每个线程负责 125 行。线程到 CPU 硬件执行单元的映射由硬件、操作系统和调度器完成。

四、 遇到的问题及解决方法

本次实验操作并不难，特别是使用 OpenMP 编程，十分方便。

但难点在于对题目的理解。就是该如何用 OpenMP 编程方法，体现“foster 并行设计思想”。经过了与很多同学的讨论，认识到了共享内存是不需要通信的，因为不存在数据依赖。同时，也加深了对“聚合（Agglomeration）”的理解，明白了聚合类似于一种优化，是用来“减少通信、提高数据局部性、发挥硬件效能”的操作。且并不是在编程的最后完成的，而是在一开始就设计好的。