这次的任务主要是让大家初步接触并行计算

本次学习任务时间截止至 11月 2日，大家注意安排好自己的学习时间，如果感觉学习上有困难可以及时提出来

前置知识以及将要学习的知识

- 线性代数
- 线、进程模型，调度算法等
- Linux OS
- pthread 多线程编程
- 指令集向量化，SIMD
- 学习使用vtune 进行性能分析
- shell脚本的编写

任务一

- 学习使用Linux系统，掌握常用的命令。
- 学习了解Linux系统的内核，整体架构等，例如环境变量，root权限，用户权限，目录名称的含义（bin目录一般 是存放二进制文件的）等。
- 了解程序编译、执行的过程。
- 学习进、线程的相关概念。

任务二

通过所学的线代知识编写矩阵加减乘除算法。可以适用于不同规格的矩阵，以及不同的数据类型（整形，浮点型，短整型等）

任务三

以数据类型为浮点数的矩阵算法作为之后测试，调试的baseline（基准时间），该任务分为几个小任务如下：

- 学习使用vtune进行性能分析，并贯彻整个优化性能过程。
- 使用pthread编写多线程，来加速矩阵算法。
- 使用指令集向量化对矩阵算法进行手动向量化操作。
- 编写shell脚本，实现对后台CPU使用率的的监控，并实时将CPU利用率前三的进程信息存储到文件中。可以试试把前三进程的所有线程的cpu利用率也记录下来。

任务四

在本机完成hpl的性能调优

1. 了解HPL.dat文件中每一行的含义。
   可参考https://blog.csdn.net/gyx1549624673/article/details/86551466
   （更多的可自行百度）
2. 了解HPL的调优手段，发挥你们的搜索能力去了解吧。
3. 如果对于Linux系统基本指令不熟悉的可以接着学习与实践,可以顺带了解一下Linux操作系统的原理。
4. Gfloat = 核心数(CPU几核) 主频(GHz) 每时钟周期浮点运算次数

   ！！！ 该公式用于计算理论峰值，实际峰值为HPL跑出来的结果

5. 多进程运行命令 mpirun -np 线程数 ./可执行文件。 例如:4个进程运行xhpl，指令为 mpirun -np 4 ./xhpl

友情提醒： 在测试调优的时候要保持电源连接，后台少应用甚至无应用，网络连接，如果电脑有性能模式的话也可以打开，不然有可能会产生较大的数值波动。还有一个就是在调参的时候不要一开始就把数值开太大了不然会跑很久。最后要测出自己的实际峰值与理论峰值的比值，需要提交（最原始的比值，以及调优最终的比值）

学习链接

- 向量化指令
- 参考书目《现代操作系统》、《深入理解计算机系统》、《鸟哥的Linux私房菜》、《并行算法设计与性能优化-刘文志》
- pthread介绍
- pthreads并行编程
- shell编程
- vtune

拓展练习

（1）结构体数组相加

```
struct A{
    int a;
    float b;
    double c;
    char d;
};
union B{
    int a;
    float b;
    double c;
    char d;
};
```

实现两个长度为100的数组相加，数组的数据类型有两种分别为结构体A以及联合体B，尝试考虑不同情况下的相加场景，例如当前B中的类型为int或者当前B中类型为double等。

- 通过指令集对该算法的不同相加场景进行向量化。
- 用vtune性能能分析每一步算法调整。

（2） 关于Dijkstra算法的并行实现

问题简述：
针对经典的Dijkstra算法的串行实现，设计其共享内存并行机制，分析优化的思路和流程并使用OpenMP实现其并行化，最终给出实验结果，包括并行结果正确性说明，**2/4/8/16线程并行执行的时间（Wall Time）、加速比（并行时间/串行时间）和效率（加速比/线程数）**。

详细说明：
单源最短路径问题可以定义为给定一个图G=(V，E)，找出从某个给定源顶点s到每个顶点v的最短路径。该图可以是有向图也可以是无向图，可以是加权值的也可以是非加权的，其表示方法可以是邻接表也可以是邻接矩阵。Dijkstra算法如果用来解带权有向图的单源最短路径问题，则要求权值非负。
Dijkstra算法已知一个顶点集合S，若为有权图则已知源点s到集合中的顶点的最终最短路径的权值。算法反复选择具有最短路径估计的顶点u，u属于V-S集合，并将u加入到S中，对u的所有出边进行松弛操作。另外还需用到顶点的最小优先队列Q，其中的顶点是依据顶点的d值排序的。算法如下：

```
// 针对不带权无向图的串行Dijkstra算法的C/C++实现代码,其中图使用邻接矩阵的方式表示。

// A C++ program for Dijkstra's single source shortest path algorithm.
// The program is for adjacency matrix representation of the graph
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>
// Number of vertices in the graph
#define V 9
// A utility function to find the vertex with minimum distance value, from
// the set of vertices not yet included in shortest path tree
int minDistance(int dist[], bool sptSet[])
{
// Initialize min value
int min = INT_MAX, min_index;
for (int v = 0; v < V; v++)
    if (sptSet[v] == false && dist[v] <= min)
        min = dist[v], min_index = v;
return min_index;
}

// A utility function to print the constructed distance array
int printSolution(int dist[], int n)
{
printf("Vertex Distance from Source\n");
for (int i = 0; i < V; i++)
    printf("%d \t %d\n", i, dist[i]);
}

// Function that implements Dijkstra's single source shortest path algorithm
// for a graph represented using adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
    int dist[V];     // The output array. dist[i] will hold the shortest
                     // distance from src to i
    bool sptSet[V]; // sptSet[i] will true if vertex i is included in shortest
                     // path tree or shortest distance from src to i is finalized
    // Initialize all distances as INFINITE and stpSet[] as false
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;
    // Distance of source vertex from itself is always 0
    dist[src] = 0;
    // Find shortest path for all vertices
    for (int count = 0; count < V-1; count++){
    // Pick the minimum distance vertex from the set of vertices not
    // yet processed. u is always equal to src in the first iteration.
    int u = minDistance(dist, sptSet);
    // Mark the picked vertex as processed
    sptSet[u] = true;
```

```
    // Update dist value of the adjacent vertices of the picked vertex.
    for (int v = 0; v < V; v++)
        // Update dist[v] only if is not in sptSet, there is an edge from
        // u to v, and total weight of path from src to v through u is
        // smaller than current value of dist[v]
        if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                                    && dist[u]+graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
    }
    // print the constructed distance array
    printSolution(dist, V);
}
// driver program to test above function
int main(){
/* Let us create the example graph discussed above */
int graph[V][V] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},
                   {4, 0, 8, 0, 0, 0, 0, 11, 0},
                   {0, 8, 0, 7, 0, 4, 0, 0, 2},
                   {0, 0, 7, 0, 9, 14, 0, 0, 0},
                   {0, 0, 0, 9, 0, 10, 0, 0, 0},
                   {0, 0, 4, 14, 10, 0, 2, 0, 0},
                   {0, 0, 0, 0, 0, 2, 0, 1, 6},
                   {8, 11, 0, 0, 0, 0, 1, 0, 7},
                   {0, 0, 2, 0, 0, 0, 6, 7, 0}
                   };
    dijkstra(graph, 0);
    return 0;
}
```

注意：

建议自己实现串行Dijkstra算法代码（不使用STL）；

并行的结果需要和串行等价；

多次实验记录平均运行时间作为实验结果。

并行手段不限（pthread、openmp、MPI等方式均可）