22级超算优化方向考核

很高兴大家参与最后一轮学习考核，本次考核截止到**下学期开学**

本次考核目的主要在于 并行编程的初步学习 以及 动手实操优化一些基础代码

在学习过程中遇到疑惑都可以在群里提出，大家可以一起交流讨论

一、 Linux操作系统的学习

1.1 Linux下C编程

- 掌握Linux环境下C代码的编译与调试
- C语言基础知识
- 目录与文件系统

1.2 Linux 进程

- 理解进程的基本概念，特别是进程的各个属性的含义。
- 了解进程的运行环境，特别是要加深对命令行参数、环境变量等概念的理解。
- 了解进程的创建与运行

1.3 Shell入门

- 超算习堂 - Shell入门教程 https://easyhpc.net/lab/27

ps. 以上Linux内容会提供一些学习资料供大家参考学习，也可以选择自己查找资料学习

1.4 Shell编程作业

- SHELL1 统计文件的行数

编写一个shell脚本以输出一个文本文件 a.txt 中的行数
示例:
假设 a.txt 内容如下:

```
#include <iostream>
using namespace std;
int main()
{
 int a = 10;
 int b = 100;
 cout << "a + b:" << a + b << endl;
 return 0;
}
```

你的脚本应当输出:

```
9
```

- SHELL2 打印文件的最后5行

查看日志的时候，经常会从文件的末尾往前查看，请编写一个shell脚本以输出一个文本文件 a.txt 中的最后5行。
示例:
假设 a.txt 内容如 SHELL1 内容所示

你的脚本应当输出：

```
int a = 10;
int b = 100;
cout << "a + b:" << a + b << endl;
return 0;
}
```

- SHELL3 监控CPU使用率

编写shell脚本，实现对后台CPU使用率的的监控，并将CPU利用率前三的进程信息存储到文件中

二、入门并行编程理论学习

要求：学习 OpenMP/MPI/Pthreads 的基础知识，了解向量化（AVX/SSE）的使用

ps. OpenMP、MPI、Pthreads选择一个学习即可

2.2 向量化 和 2.3 并行编程方法论 目前仅了解即可

2.1.1 OpenMP

OpenMP API 用户指南：

https://math.ecnu.edu.cn/~jypan/Teaching/ParaComp/books/OpenMP_sun10.pdf

超算习堂 - OpenMP 编程实训：

https://easyhpc.net/problem/programming_lab/2

推荐学习视频：

https://www.bilibili.com/video/BV18M41187ZU/?spm_id_from=333.999.0.0

https://www.bilibili.com/video/BV1Ea411X78R/?spm_id_from=333.999.0.0&vd_source=d4d8725a1a30e189fa2cd9218fa9842a

推荐博客：

https://zhuanlan.zhihu.com/p/51173703

https://blog.csdn.net/fengbingchun/article/details/15027507?spm=1001.2014.3001.5506

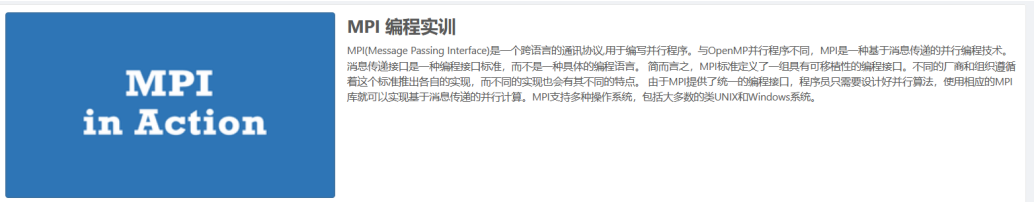
2.1.2 MPI

MPI tutorial：

https://mpitutorial.com/tutorials/

超算习堂 - MPI 编程实训：

https://easyhpc.net/problem/programming_lab/1



推荐学习视频：

https://www.bilibili.com/video/BV13v4y1v7y7/?spm_id_from=333.999.0.0

https://www.bilibili.com/video/BV1FN4y137GV/?spm_id_from=333.999.0.0&vd_source=d4d8725a1a30e189fa2cd9218fa9842a

2.1.3 Pthreads

pthreads介绍：

https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html

超算习堂 - Pthreads编程实训：

https://easyhpc.net/problem/programming_lab/3

推荐学习视频：

https://www.bilibili.com/video/BV1qQ4y1i7VC/?p=1&vd_source=f3af9f038101af118fe42051bd4a8229

2.2 向量化（了解）

向量化指令集：

Mirror of Intel® Intrinsics Guide (laruence.com)

2.3 并行编程方法论（了解）

https://www.bilibili.com/video/BV1794y1U7Rx/?spm_id_from=333.999.0.0&vd_source=d4d8725a1a30e189fa2cd9218fa9842a

https://www.bilibili.com/video/BV1mT411V7gp/?spm_id_from=333.999.0.0

https://www.bilibili.com/video/BV1AB4y1Q72p/?spm_id_from=333.999.0.0

三、实操优化代码

3.1 矩阵乘法优化

算法流程伪代码：

Algorithm 1 Matrix-Matrix Multiplication

```
1: procedure MMM
2:     for i from 1 to N do
3:         for j from 1 to M do
4:             for p from 1 to K do
5:                 C(i,j) += A(i,p)· B(p,j)
6:             end for
7:         end for
8:     end for
9: end procedure
```


ps. 会提供Baseline（基准）代码，以及数据，只能修改指定部分的优化代码以及.sh文件

```
//--------------初始矩阵乘法运算--------------------//
        t0 = omp_get_wtime();
        //mul
        for(int i=0;i<N;i++)
                for(int j=0;j<N;j++)
                        for(int k=0;k<N;k++)
                                c_0[i][j] += a[i][k]*b[k][j];

        t1 = omp_get_wtime();
        T0 = (t1-t0)*1000;
        printf("初始矩阵乘法运行耗时: %f ms\n", T0);
//------------------------------------------------//
        t0 = omp_get_wtime();

//-----------优化矩阵乘法运算（需修改部分）-----------//
//--------------- c = a * b -------------------------//
        for(int i=0;i<N;i++)
                for(int j=0;j<N;j++)
                        for(int k=0;k<N;k++)
//------------------------------------------------//
        t1 = omp_get_wtime();
        T1 = (t1-t0)*1000;
        printf("优化后矩阵乘法运行耗时: %f ms\n", T1);
        printf("加速比为: %f\n",T0/T1);
```

可修改部分

要求：需采用 OpenMP/MPI/Pthreads（任选一个） 操作进行代码优化

3.2 稀疏矩阵向量乘

背景介绍：

https://xupsh.github.io/pp4fpgas-cn/06-Sparse-Matrix-Vector-Multiplication.html

采用CSR存储格式：

https://zhuanlan.zhihu.com/p/342942385

$$\begin{bmatrix} 10 & 20 & 0 & 0 & 0 & 0 \\ 0 & 30 & 0 & 40 & 0 & 0 \\ 0 & 0 & 50 & 60 & 70 & 0 \\ 0 & 0 & 0 & 0 & 0 & 80 \end{bmatrix}$$

$$V = [10, 20, 30, 40, 50, 60, 70, 80]$$

$$COL\_INDEX = [0, 1, 1, 3, 2, 3, 4, 5]$$

$$ROW\_INDEX = [0, 2, 4, 7, 8]$$

核心代码：

```
inline void smmSerial(const int numRows,const int *rowIndex,const int
*columnIndex,const float *val,const float *x,float *r){
    int rowStart;
    int rowEnd;
    for(int i=0;i<numRows;i++){
        rowStart = rowIndex[i];
        rowEnd = rowIndex[i+1];
        float sum = 0.0f;
        for(int j=rowStart;j<rowEnd;j++){
            sum += val[j] * x[columnIndex[j]];
        }
        r[i] = sum;
    }
}
```

提示：由于每行的非零元素的个数可能差异巨大，为了减少负载均衡的问题，使用动态负载均衡策略，size大小的确定需要考虑

要求：对上述代码进行并行化操作，可自行构造数据

3.3 中值滤波代码优化（附加题）

中值滤波代码背景介绍：

https://baike.baidu.com/item/%E4%B8%AD%E5%80%BC%E6%BB%A4%E6%B3%A2/5031069?fr=aladdin

核心代码：

```
void medianFilter(int height,int width,unsigned char* __restrict__ src,unsigned
char* __restrict__ dst){
    for(int i=1;i<height-1;i++){
        for(int j=1;j<width-1;j++){
            unsigned char a[9];
            a[0] = src[i*width+j];
            a[1] = src[i*width+j+1];
            a[2] = src[i*width+j-1];
            a[3] = src[(i+1)*width+j];
            a[4] = src[(i+1)*width+j+1];
            a[5] = src[(i+1)*width+j-1];
            a[6] = src[(i-1)*width+j];
            a[7] = src[(i-1)*width+j+1];
```

```
            a[8] = src[(i-1)*width+j-1];
            for(int ji=0;ji<5;ji++){
                for(int jj=ji+1;jj<9;jj++){
                    if(a[ji] > a[jj]){
                        unsigned char tmp = a[ji];
                        a[ji] = a[jj];
                        a[jj] = tmp;
                    }
                }
            }
            dst[i*width+j] = a[4];
        }
    }
    for(int i=0;i<width;i++){
        dst[i] = src[i]; // first now
        dst[(height-1)*width+i] = src[(height-1)*width+i];
    }
    for(int i=0;i<height;i++){
        dst[i*width] = src[i*width];
        dst[i*width+width-1] = src[i*width+width-1];
    }
}
```

要求：已提供了代码及数据，需对代码进行并行化，并记录优化过程