南京航空航天大学

毕业设计程序源代码

题　　目　基于MPI和openMP的程序性能优化研究

学　　院　计算机科学与技术学院

专　　业　软件工程

学生姓名　胡思旺　学号　161330216

指导教师　陈哲　职称　副教授

毕设地点　南京航空航天大学

2017年 6 月

目录

第一章 代码目录

1.1 程序目录图

整个毕业设计在linux环境下面完成,编译工具为g++,文本编辑工具gedit,调试工具为gdb,代码组织列表如下:

```
Design
├── Matrix
│   ├── data.txt
│   ├── Matrix.cpp
│   ├── Matrix.sh
│   ├── myMatrix.h
│   ├── Omp_Matrix.cpp
│   ├── Omp_MPI_Matrix.cpp
│   └── Random.cpp
├── Max
│   ├── data.txt
│   ├── Max.cpp
│   ├── Max.sh
│   ├── myMax.h
│   ├── Omp_Max.cpp
│   ├── Omp_MPI_Max.cpp
│   └── Random.cpp
├── readme.txt
├── run.sh
├── Sort
│   ├── data.txt
│   ├── mySort.h
│   ├── Omp_MPI_Sort.cpp
│   ├── Omp_Sort.cpp
│   ├── Random.cpp
│   ├── Sort.cpp
│   └── Sort.sh
└── UI
    ├── exec.sh
    ├── Main.png
    ├── result.txt
    ├── UI.c
    ├── UI.glade~
    ├── UI.xml~
    └── wait.gif
```

1.2 文件说明

以下是各个文件的解释说明:

Design 项目主目录

run.sh 项目运行脚本

readme.txt 帮助文件

Matrix 实现 MPI,openMP,MPI 与 openMP 混合编程的并行矩阵乘法主目录

data.txt 用来保存用来计算的矩阵数据

myMatrix.h 头文件中定义了矩阵结构体和一些编译条件指令

Random.cpp 为产生随机矩阵源文件

Matrix.cpp 为 MPI 实现并行矩阵乘法源文件

Omp_Matrix.cpp 为 openMP 实现并行矩阵乘法源文件

Omp_MPI_Matrix.cpp 为 MPI 与 openMP 混合编程实现并行矩阵乘法源文件

Matrix.sh 为 linux 下测试脚本

Max 实现 MPI,openMP,MPI 与 openMP 混合编程的求序列最值主目录

data.txt 用来保存序列数据

myMax.h 头文件中定义了序列结构体和一些编译条件指令

Random.cpp 为产生随机序列数据源文件

Max.cpp 为 MPI 实现并行求序列最值源文件

Omp_Max.cpp 为 openMP 实现并行求序列最值源文件

Omp_MPI_Max.cpp 为 MPI 与 openMP 混合编程实现并行求序列最值源文件

Matrix.sh 为 linux 下测试脚本

Sort 实现 MPI,openMP,MPI 与 openMP 混合编程的并行排序主目录

data.txt 用来保存序列数据

mySort.h 头文件中定义了序列结构体和一些编译条件指令

Random.cpp 为产生随机序列数据源文件

Sort.cpp 为 MPI 实现并行排序源文件

Omp_Sort.cpp 为 openMP 实现并行排序源文件

Omp_MPI_Sort.cpp 为 MPI 与 openMP 混合编程实现并行排序源文件

Matrix.sh 为 linux 下测试脚本

UI 图形界面测试工具主目录

result.txt 用来保存实验结果

exec.sh 测试脚本

Main.png 图形界面图标文件

UI.c 图形界面测试工具源文件

Wait.gif 图形界面资源文件

第二章 Matrix

2.1 myMatrix.h

```
#pragma once
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#ifdef MYMPI
#include <mpi.h>
#endif

#ifdef OPENMP
#include <omp.h>
#include <pthread.h>
#endif

long DATASIZE;

long WIDTH_M;
long HIGH_M;

long WIDTH_N;
long HIGH_N;

#define TIMESIZE 1000000

typedef int keyType;

typedef struct
{
    keyType **data;
    int size;
}Matrix_M;

typedef struct
{
    keyType **data;
```

```
    int size;
}Matrix_N;

typedef struct
{
    keyType **data;
    int size;
}Matrix_Result;
```

2.2 Matrix.cpp

```
#define MYMPI
#include "myMatrix.h"

void Print(Matrix_M &M,Matrix_N &N)
{
    for (int i = 0; i < WIDTH_M; i++)
    {
        for (int j = 0; j < HIGH_M; j++)
        {
            printf("%d    ", M.data[i][j]);
        }
        printf("\n");
    }
    printf("-----------------------------------------------\n");
    for (int i = 0; i < WIDTH_N; i++)
    {
        for (int j = 0; j < HIGH_N; j++)
        {
            printf("%d    ", N.data[i][j]);
        }
        printf("\n");
    }
    printf("-----------------------------------------------\n");
}

void readFile(Matrix_M &M,Matrix_N &N)
{
    FILE *file = fopen("data.txt", "r");
    if (file==NULL)
    {
        printf("打开文件失败!");
    }
    for(int i=0;i<DATASIZE;i++)
```

```
    {
        for(int j=0;j<DATASIZE;j++)
        {
            fscanf(file,"%d ",&(M.data[i][j]));
        }
    }
    for(int i=0;i<DATASIZE;i++)
    {
        for(int j=0;j<DATASIZE;j++)
        {
            fscanf(file,"%d ",&(N.data[i][j]));
        }
    }
    fclose(file);
}

void matrixMulti(Matrix_Result &result,Matrix_M &M,Matrix_N &N)
{
    if (HIGH_M != WIDTH_N)
        return;
    else
    {
        int i, j, k;
        for (i = 0; i < WIDTH_M; i++)
        {
            for (j = 0; j < HIGH_N; j++)
            {
                int sum = 0;
                for (k = 0; k < WIDTH_N; k++)
                {
                    sum += (M.data[i][k]*N.data[k][j]);
                }
                result.data[i][j] = sum;
            }
        }
    }
}


void matrixMultiParallel(int argc,char * argv[],Matrix_Result &result, Matrix_M &M,
Matrix_N &N)
{
    int rank,size;
    MPI_Status status;
```

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
if(size==1)
{
    double first,last,time;
    first=MPI_Wtime();
    matrixMulti(result, M, N);
    last=MPI_Wtime();
    time=last-first;
    printf("TIME:%lf\n",time);
}
else
{
    if(rank==0)
    {
        int dest,source,row,offset,flag;
        row = WIDTH_M / size;
        offset=0;
        source=0;
        double first,last,time;
        first=MPI_Wtime();
        for (dest = 1; dest < size; dest++)
        {
            MPI_Send(&offset, 1, MPI_INT, dest, 1, MPI_COMM_WORLD);
            MPI_Send(&row, 1, MPI_INT, dest, 1, MPI_COMM_WORLD);
            offset = offset + row;
        }
        for (int i = offset; i < WIDTH_M; i++)
        {
            for (int j = 0; j < HIGH_N; j++)
            {
                int sum = 0;
                for (int k = 0; k < WIDTH_N; k++)
                {
                    sum += (M.data[i][k]*N.data[k][j]);
                }
                result.data[i][j] = sum;
            }
        }
        for(dest = 1;dest < size; dest++)
        {
            MPI_Recv(&flag, 1, MPI_INT, dest, 2, MPI_COMM_WORLD,
&status);
```

```
            }
            last=MPI_Wtime();
            time=last-first;
            printf("TIME:%lf\n",time);
        }
        if(rank>0)
        {
            int dest,source,row,offset;
            source=0;
            MPI_Recv(&offset, 1, MPI_INT, source, 1, MPI_COMM_WORLD,
&status);
            MPI_Recv(&row, 1, MPI_INT, source, 1, MPI_COMM_WORLD,
&status);
            for (int i = offset; i < (offset+row); i++)
            {
                for (int j = 0; j < HIGH_N; j++)
                {
                    int sum = 0;
                    for (int k = 0; k < WIDTH_N; k++)
                    {
                        sum += (M.data[i][k]*N.data[k][j]);
                    }
                    result.data[i][j] = sum;
                }
            }
            MPI_Send(&source, 1, MPI_INT, source, 2, MPI_COMM_WORLD);
        }
        MPI_Finalize();
    }
}

void createMatrix(char * argv[],Matrix_Result &R, Matrix_M &M, Matrix_N &N)
{
    DATASIZE=atol(argv[1]);
    WIDTH_M=DATASIZE;
    HIGH_M=DATASIZE;
    WIDTH_N=DATASIZE;
    HIGH_N=DATASIZE;
    M.data = (keyType **)malloc(sizeof(keyType *) * DATASIZE);
    for (int i = 0; i < DATASIZE; i++)
    {
        *(M.data + i) = (keyType *)malloc(sizeof(keyType) * DATASIZE);
    }
    N.data = (keyType **)malloc(sizeof(keyType *) * DATASIZE);
```

```
    for (int i = 0; i < DATASIZE; i++)
    {
        *(N.data + i) = (keyType *)malloc(sizeof(keyType) * DATASIZE);
    }
    R.data = (keyType **)malloc(sizeof(keyType *) * DATASIZE);
    for (int i = 0; i < DATASIZE; i++)
    {
        *(R.data + i) = (keyType *)malloc(sizeof(keyType) * DATASIZE);
    }
}

int main(int argc,char * argv[])
{
    if(argc==1)
    {
        printf("请输入数据量!\n");
    }
    else
    {

        Matrix_M M;
        Matrix_N N;
        Matrix_Result R;
        createMatrix(argv,R,M,N);
        readFile(M,N);
        matrixMultiParallel(argc,argv,R, M, N);
    }
}
```

2.3 Omp_Matrix.cpp

```
#define OPENMP
#include "myMatrix.h"

void Print(Matrix_M &M,Matrix_N &N)
{
    for (int i = 0; i < WIDTH_M; i++)
    {
        for (int j = 0; j < HIGH_M; j++)
        {
            printf("%d    ", M.data[i][j]);
        }
        printf("\n");
    }
```

```
    printf("-----------------------------------------------\n");
    for (int i = 0; i < WIDTH_N; i++)
    {
        for (int j = 0; j < HIGH_N; j++)
        {
            printf("%d    ", N.data[i][j]);
        }
        printf("\n");
    }
    printf("-----------------------------------------------\n");
}

void readFile(Matrix_M &M,Matrix_N &N)
{
    FILE *file = fopen("data.txt", "r");
    if (file==NULL)
    {
        printf("打开文件失败!");
    }
    for(int i=0;i<DATASIZE;i++)
    {
        for(int j=0;j<DATASIZE;j++)
        {
            fscanf(file,"%d ",&(M.data[i][j]));
        }
    }
    for(int i=0;i<DATASIZE;i++)
    {
        for(int j=0;j<DATASIZE;j++)
        {
            fscanf(file,"%d ",&(N.data[i][j]));
        }
    }
    fclose(file);
}

void matrixMultiOmpParallel(int argc,char * argv[],Matrix_Result &result, Matrix_M
&M, Matrix_N &N)
{
    if(argc==2)
    {
        printf("请输入线程数!\n");
    }
    else
```

```
    {
        double first,last,time;
        int thread=atol(argv[2]),i,j,k;
        omp_set_num_threads(thread);
        first=omp_get_wtime();
        #pragma omp parallel shared(result,M,N) private(i,j,k)
        {
            #pragma omp for schedule(dynamic)
            for(i=0;i<DATASIZE;i++)
                {
                for(j=0;j<DATASIZE;j++)
                {
                    result.data[i][j]=0;
                    for(k=0;k<DATASIZE;k++)
                    {
                        result.data[i][j]+=M.data[i][k]*N.data[k][j];
                    }
                }
            }
        }
        last=omp_get_wtime();
        time=last-first;
        printf("TIME:%lf\n",time);
    }
}

void createMatrix(char * argv[],Matrix_Result &R, Matrix_M &M, Matrix_N &N)
{
    DATASIZE=atol(argv[1]);
    WIDTH_M=DATASIZE;
    HIGH_M=DATASIZE;
    WIDTH_N=DATASIZE;
    HIGH_N=DATASIZE;
    M.data = (keyType **)malloc(sizeof(keyType *) * DATASIZE);
    for (int i = 0; i < DATASIZE; i++)
    {
        *(M.data + i) = (keyType *)malloc(sizeof(keyType) * DATASIZE);
    }
    N.data = (keyType **)malloc(sizeof(keyType *) * DATASIZE);
    for (int i = 0; i < DATASIZE; i++)
    {
        *(N.data + i) = (keyType *)malloc(sizeof(keyType) * DATASIZE);
    }
    R.data = (keyType **)malloc(sizeof(keyType *) * DATASIZE);
```

```
    for (int i = 0; i < DATASIZE; i++)
    {
        *(R.data + i) = (keyType *)malloc(sizeof(keyType) * DATASIZE);
    }
}

int main(int argc,char * argv[])
{
    if(argc==1)
    {
        printf("请输入数据量!\n");
    }
    else
    {

        Matrix_M M;
        Matrix_N N;
        Matrix_Result R;
        createMatrix(argv,R,M,N);
        readFile(M,N);
        matrixMultiOmpParallel(argc,argv,R, M, N);
    }
}
```

2.4 Omp_MPI_Matrix.cpp

```
#define MYMPI
#define OPENMP
#include "myMatrix.h"

void Print(Matrix_M &M,Matrix_N &N)
{
    for (int i = 0; i < WIDTH_M; i++)
    {
        for (int j = 0; j < HIGH_M; j++)
        {
            printf("%d    ", M.data[i][j]);
        }
        printf("\n");
    }
    printf("-----------------------------------------------\n");
    for (int i = 0; i < WIDTH_N; i++)
    {
        for (int j = 0; j < HIGH_N; j++)
```

```
            {
                printf("%d    ", N.data[i][j]);
            }
            printf("\n");
        }
        printf("-----------------------------------------------\n");
}

void readFile(Matrix_M &M,Matrix_N &N)
{
        FILE *file = fopen("data.txt", "r");
        if (file==NULL)
        {
             printf("打开文件失败!");
        }
        for(int i=0;i<DATASIZE;i++)
        {
            for(int j=0;j<DATASIZE;j++)
            {
                fscanf(file,"%d ",&(M.data[i][j]));
            }
        }
        for(int i=0;i<DATASIZE;i++)
        {
            for(int j=0;j<DATASIZE;j++)
            {
                fscanf(file,"%d ",&(N.data[i][j]));
            }
        }
        fclose(file);
}

void matrixMulti(Matrix_Result &result,Matrix_M &M,Matrix_N &N)
{
        if (HIGH_M != WIDTH_N)
            return;
        else
        {
            int i,j,k;
              #pragma omp parallel shared(result,M,N)    private(i,j,k)
              {
                    #pragma omp for schedule(dynamic)
                  for (i = 0; i < WIDTH_M; i++)
                  {
```

```
                    for (j = 0; j < HIGH_N; j++)
                    {
                        int sum = 0;
                        for (k = 0; k < WIDTH_N; k++)
                        {
                            sum += (M.data[i][k]*N.data[k][j]);
                        }
                        result.data[i][j] = sum;
                    }
                }
            }
        }
    }

void matrixMultiParallel(int argc,char * argv[],Matrix_Result &result, Matrix_M &M,
Matrix_N &N)
{
    int rank,size;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if(size==1)
    {
        double first,last,time;
        first=MPI_Wtime();
        matrixMulti(result, M, N);
        last=MPI_Wtime();
        time=last-first;
        printf("TIME:%lf\n",time);
    }
    else
    {
        if(rank==0)
        {
            int dest,source,row,offset,flag;
            row = WIDTH_M / size;
            offset=0;
            source=0;
            double first,last,time;
            first=MPI_Wtime();
            for (dest = 1; dest < size; dest++)
            {
                MPI_Send(&offset, 1, MPI_INT, dest, 1, MPI_COMM_WORLD);
```

```
                MPI_Send(&row, 1, MPI_INT, dest, 1, MPI_COMM_WORLD);
                offset = offset + row;
            }
              #pragma omp parallel shared(result,M,N)
             {
                    #pragma omp for schedule(dynamic)
                for (int i = offset; i < WIDTH_M; i++)
                {
                    for (int j = 0; j < HIGH_N; j++)
                    {
                        int sum = 0;
                        for (int k = 0; k < WIDTH_N; k++)
                        {
                            sum += (M.data[i][k]*N.data[k][j]);
                        }
                        result.data[i][j] = sum;
                    }
                }
              }
            for(dest = 1;dest < size; dest++)
            {
                MPI_Recv(&flag, 1, MPI_INT, dest, 2, MPI_COMM_WORLD,
&status);
            }
            last=MPI_Wtime();
            time=last-first;
            printf("TIME:%lf\n",time);
        }
        if(rank>0)
        {
            int dest,source,row,offset;
            source=0;
            MPI_Recv(&offset, 1, MPI_INT, source, 1, MPI_COMM_WORLD,
&status);
            MPI_Recv(&row, 1, MPI_INT, source, 1, MPI_COMM_WORLD,
&status);
              #pragma omp parallel shared(result,M,N)
             {
                    #pragma omp for schedule(dynamic)
                for (int i = offset; i < (offset+row); i++)
                {
                    for (int j = 0; j < HIGH_N; j++)
                    {
                        int sum = 0;
```

```
                        for (int k = 0; k < WIDTH_N; k++)
                        {
                            sum += (M.data[i][k]*N.data[k][j]);
                        }
                        result.data[i][j] = sum;
                    }
                }
            }
            MPI_Send(&source, 1, MPI_INT, source, 2, MPI_COMM_WORLD);
        }
        MPI_Finalize();
    }
}

void createMatrix(char * argv[],Matrix_Result &R, Matrix_M &M, Matrix_N &N)
{
    DATASIZE=atol(argv[1]);
    WIDTH_M=DATASIZE;
    HIGH_M=DATASIZE;
    WIDTH_N=DATASIZE;
    HIGH_N=DATASIZE;
    M.data = (keyType **)malloc(sizeof(keyType *) * DATASIZE);
    for (int i = 0; i < DATASIZE; i++)
    {
        *(M.data + i) = (keyType *)malloc(sizeof(keyType) * DATASIZE);
    }
    N.data = (keyType **)malloc(sizeof(keyType *) * DATASIZE);
    for (int i = 0; i < DATASIZE; i++)
    {
        *(N.data + i) = (keyType *)malloc(sizeof(keyType) * DATASIZE);
    }
    R.data = (keyType **)malloc(sizeof(keyType *) * DATASIZE);
    for (int i = 0; i < DATASIZE; i++)
    {
        *(R.data + i) = (keyType *)malloc(sizeof(keyType) * DATASIZE);
    }
}

int main(int argc,char * argv[])
{
    if(argc==1)
    {
        printf("请输入数据量!\n");
    }
```

```
    else
    {
        if(argc==2)
        {
            printf("请输入线程数!\n");
        }
        else
        {
            Matrix_M M;
            Matrix_N N;
            Matrix_Result R;
            createMatrix(argv,R,M,N);
            readFile(M,N);
              int thread=atol(argv[2]);
              omp_set_num_threads(thread);
            matrixMultiParallel(argc,argv,R, M, N);
        }
    }
}
```

2.5 Random.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

long DATASIZE;

long WIDTH_M;
long HIGH_M;

long WIDTH_N;
long HIGH_N;


typedef int keyType;


typedef struct
{
    keyType **data;
    int size;
}Matrix_M;
```

```
typedef struct
{
	keyType **data;
	int size;
}Matrix_N;


void Print(Matrix_M &M,Matrix_N &N)
{
	for (int i = 0; i < WIDTH_M; i++)
	{
		for (int j = 0; j < HIGH_M; j++)
		{
			printf("%d	", M.data[i][j]);
		}
		printf("\n");
	}
	printf("-------------------------------------------------\n");
	for (int i = 0; i < WIDTH_N; i++)
	{
		for (int j = 0; j < HIGH_N; j++)
		{
			printf("%d	", N.data[i][j]);
		}
		printf("\n");
	}
	printf("-------------------------------------------------\n");
}


void createMatrix(Matrix_M &M,Matrix_N &N)
{
	srand((unsigned)time(NULL));
	for (int i = 0; i < WIDTH_M; i++)
	{
		for (int j = 0; j < HIGH_M; j++)
		{
			M.data[i][j] = rand() % 100;
		}
	}
	M.size = WIDTH_M*HIGH_N;

	srand((unsigned)time(NULL));
```

```
    for (int i = 0; i < WIDTH_N; i++)
    {
        for (int j = 0; j < HIGH_N; j++)
        {
            N.data[i][j] = rand() % 100;
        }
    }
    N.size = WIDTH_M*HIGH_N;
}

void printFile(Matrix_M &M,Matrix_N &N)
{
    FILE *file = fopen("data.txt", "w");
    if (file==NULL)
    {
        printf("打开文件失败!");
    }
    for(int i=0;i<DATASIZE;i++)
    {
        for(int j=0;j<DATASIZE;j++)
        {
            fprintf(file,"%d ",M.data[i][j]);
        }
    }
    for(int i=0;i<DATASIZE;i++)
    {
        for(int j=0;j<DATASIZE;j++)
        {
            fprintf(file,"%d ",N.data[i][j]);
        }
    }
    fclose(file);
}


int main(int argc,char *argv[])
{
    if(argc==1)
    {
        printf("请输入数据量!\n");
    }
    else
    {
        Matrix_M M;
```

```
        Matrix_N N;
        DATASIZE=atol(argv[1]);
        WIDTH_M=DATASIZE;
        HIGH_M=DATASIZE;
        WIDTH_N=DATASIZE;
        HIGH_N=DATASIZE;

        M.data = (keyType **)malloc(sizeof(keyType *) * DATASIZE);
        for (int i = 0; i < DATASIZE; i++)
        {
            *(M.data + i) = (keyType *)malloc(sizeof(keyType) * DATASIZE);
        }
        N.data = (keyType **)malloc(sizeof(keyType *) * DATASIZE);
        for (int i = 0; i < DATASIZE; i++)
        {
            *(N.data + i) = (keyType *)malloc(sizeof(keyType) * DATASIZE);
        }
        createMatrix(M,N);
        printFile(M,N);
    }
}
```

2.6 Matrix.sh

```
#!/bin/bash

yes="y"
no="n"
clear
mpic++ Matrix.cpp -o Matrix
g++ -fopenmp Omp_Matrix.cpp -o Omp_Matrix
mpic++ -fopenmp -o Omp_MPI_Matrix Omp_MPI_Matrix.cpp
g++ Random.cpp -o Random
echo ".............需要产生数据吗?(y/n)............."
read console
clear
if [ $console == $yes ]
then
    echo ".............请输入产生的数据量大小!............."
    read number
    clear
    ./Random $number
    echo ""
    echo ".............数据产生成功!............."
```

```
        echo ""
        echo ".............请输入进程个数!............."
        echo ""
        read process
        clear
        echo ".............请输入线程个数!............."
        echo ""
        read thread
        clear
        echo ".............程序开始运行!............."
        echo "MPI Program is running!"
        mpirun -np $process ./Matrix $number
        echo "openMP Program is running!"
        ./Omp_Matrix $number $thread
        echo "openMP&MPI Program is running!"
        mpirun -np $process ./Omp_MPI_Matrix $number $thread
fi

if [ $console == $no ]
then
        echo ".............请输入数据量大小!............."
        echo ""
        read number
        clear
        echo ".............请输入进程个数!............."
        echo ""
        read process
        clear
        echo ".............请输入线程个数!............."
        echo ""
        read thread
        clear
        echo "程序开始运行!"
        echo "MPI Program is running!"
        mpirun -np $process ./Matrix $number
        echo "openMP Program is running!"
        ./Omp_Matrix $number $thread
        echo "openMP&MPI Program is running!"
        mpirun -np $process ./Omp_MPI_Matrix $number $thread
fi
```

第三章 Max

3.1 myMax.h

```
#pragma once
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

#ifdef MYMPI
#include <mpi.h>
#endif

#ifdef OPENMP
#include <omp.h>
#endif

long DATASIZE;
#define TIMESIZE 1000000

typedef int keyType;

typedef struct
{
	keyType *data;
	int length;
}List;
```

3.2 Max.cpp

```
#define MYMPI
#include "myMax.h"

void Print(List &L)
{
	for (int i = 0; i < DATASIZE; i++)
	{
		printf("%d\n", L.data[i]);
```

```
    }
}

void readFile(List &L)
{
    FILE *file = fopen("data.txt", "r");
    if (file==NULL)
    {
        printf("打开文件失败!");
    }
    for(int i=0;i<DATASIZE;i++)
    {
        fscanf(file,"%d ",&(L.data[i]));
    }
    fclose(file);
}

int Max(List &L)
{
    int max = L.data[0];
    for (int i = 0; i < DATASIZE; i++)
    {
        if (L.data[i] > max)
        {
            max = L.data[i];
        }
    }
    return max;
}



void MaxParallel(int argc,char * argv[],List &L, keyType &max)
{
    max = L.data[0];
    MPI_Status status;
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (size == 1)
    {
        double    _first, _last;
        double _time;
```

```
        _first = MPI_Wtime();
        max = Max(L);
        _last = MPI_Wtime();
        _time=(_last-_first);
        printf("MAX:%d      TIME:%lf\n", max, _time);
    }
    else
    {
        if (rank == 0)
        {
            double    first, last;
            double time;
            first = MPI_Wtime();
            int dest, start, end, length, source, i,temp;
            start = 0;
            length = DATASIZE / size;
            end = start + length;
            for (dest = 1; dest < size; dest++)
            {
                MPI_Send(&start, 1, MPI_INT, dest, 1, MPI_COMM_WORLD);
                if(start+length<DATASIZE)
                    MPI_Send(&end,          1,          MPI_INT,          dest,          1,
MPI_COMM_WORLD);
                else
                    MPI_Send(&L.length,          1,          MPI_INT,          dest,          1,
MPI_COMM_WORLD);
                start += length;
                end += length;
            }
            for (i = start; i < end; i++)
            {
                if (L.data[i] > max)
                    max = L.data[i];
            }
            for (i = 1; i < size; i++)
            {
                source = i;
                MPI_Recv(&temp, 1, MPI_INT, source, 2, MPI_COMM_WORLD,
&status);
                if (max < temp)
                    max = temp;
            }
            last = MPI_Wtime();
            time=last-first;
```

```
            printf("MAX:%d      TIME:%lf\n", max, time);
        }
        if (rank > 0)
        {
            int start, end, length, dest,source, i, temp;
            source = 0;
            dest = 0;
            MPI_Recv(&start, 1, MPI_INT, source, 1, MPI_COMM_WORLD,
&status);
            MPI_Recv(&end, 1, MPI_INT, source, 1, MPI_COMM_WORLD,
&status);
            for (i = start; i < end; i++)
            {
                temp = L.data[i];
                if (L.data[i] > temp)
                    temp = L.data[i];
            }
            MPI_Send(&temp, 1, MPI_INT, dest, 2, MPI_COMM_WORLD);
        }
        MPI_Finalize();
    }
}



int main(int argc,char * argv[])
{
    if(argc==1)
    {
        printf("请输入数据量!\n");
    }
    else
    {
        keyType max;
        List L;
        DATASIZE=atol(argv[1]);
        L.data=(keyType*)malloc(sizeof(keyType)*DATASIZE);
        readFile(L);
        MaxParallel(argc, argv, L, max);
    }
}
```

3.3 Omp_Max.cpp

```
#define OPENMP
#include "myMax.h"

void Print(List &L)
{
	for (int i = 0; i < DATASIZE; i++)
	{
		printf("%d\n", L.data[i]);
	}
}

void readFile(List &L)
{
	FILE *file = fopen("data.txt", "r");
	if (file==NULL)
	{
		 printf("打开文件失败!");
	}
	for(int i=0;i<DATASIZE;i++)
	{
		fscanf(file,"%d ",&(L.data[i]));
	}
	fclose(file);
}

void OmpMaxParallel(List &L,keyType &max)
{
	 double first,last,time;
	 first=omp_get_wtime();
	 max = L.data[0];
	 #pragma omp parallel for
	for (int i = 0; i < DATASIZE; i++)
	{
		if (L.data[i] > max)
		{
			max = L.data[i];
		}
	}
	 last=omp_get_wtime();
	time=last-first;
	 printf("MAX:%d	TIME:%lf\n", max, time);
```

```
}

int main(int argc,char * argv[])
{
    if(argc==1)
    {
        printf("请输入数据量!\n");
    }
    else
    {
        keyType max;
        List L;
        DATASIZE=atol(argv[1]);
        L.data=(keyType*)malloc(sizeof(keyType)*DATASIZE);
        readFile(L);
         int thread=atol(argv[2]);
        omp_set_num_threads(thread);
        OmpMaxParallel(L, max);
    }
}
```

3.4 Omp_MPI_Max.cpp

```
#define MYMPI
#define OPENMP
#include "myMax.h"

void Print(List &L)
{
    for (int i = 0; i < DATASIZE; i++)
    {
        printf("%d\n", L.data[i]);
    }
}

void readFile(List &L)
{
    FILE *file = fopen("data.txt", "r");
    if (file==NULL)
    {
        printf("打开文件失败!");
    }
    for(int i=0;i<DATASIZE;i++)
    {
```

```
            fscanf(file,"%d ",&(L.data[i]));
        }
        fclose(file);
}

int Max(List &L)
{
        int max = L.data[0];
         #pragma omp parallel for
        for (int i = 0; i < DATASIZE; i++)
        {
            if (L.data[i] > max)
            {
                max = L.data[i];
            }
        }
        return max;
}



void MaxParallel(int argc,char * argv[],List &L, keyType &max)
{
        max = L.data[0];
        MPI_Status status;
        int rank, size;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        if (size == 1)
        {
            double    _first, _last;
            double _time;
            _first = MPI_Wtime();
            max = Max(L);
            _last = MPI_Wtime();
            _time=(_last-_first);
            printf("MAX:%d      TIME:%lf\n", max, _time);
        }
        else
        {
            if (rank == 0)
            {
                double    first, last;
```

```
            double time;
            first = MPI_Wtime();
            int dest, start, end, length, source, i,temp;
            start = 0;
            length = DATASIZE / size;
            end = start + length;
            for (dest = 1; dest < size; dest++)
            {
                MPI_Send(&start, 1, MPI_INT, dest, 1, MPI_COMM_WORLD);
                if(start+length<DATASIZE)
                    MPI_Send(&end, 1, MPI_INT, dest, 1,
MPI_COMM_WORLD);
                else
                    MPI_Send(&L.length, 1, MPI_INT, dest, 1,
MPI_COMM_WORLD);
                start += length;
                end += length;
            }
              #pragma omp parallel for
            for (i = start; i < end; i++)
            {
                if (L.data[i] > max)
                    max = L.data[i];
            }
            for (i = 1; i < size; i++)
            {
                source = i;
                MPI_Recv(&temp, 1, MPI_INT, source, 2, MPI_COMM_WORLD,
&status);
                if (max < temp)
                    max = temp;
            }
            last = MPI_Wtime();
            time=last-first;
            printf("MAX:%d      TIME:%lf\n", max, time);
        }
        if (rank > 0)
        {
            int start, end, length, dest,source, i, temp;
            source = 0;
            dest = 0;
            MPI_Recv(&start, 1, MPI_INT, source, 1, MPI_COMM_WORLD,
&status);
            MPI_Recv(&end, 1, MPI_INT, source, 1, MPI_COMM_WORLD,
```

```
&status);
                  #pragma omp parallel for
                for (i = start; i < end; i++)
                {
                     temp = L.data[i];
                     if (L.data[i] > temp)
                          temp = L.data[i];
                }
                MPI_Send(&temp, 1, MPI_INT, dest, 2, MPI_COMM_WORLD);
           }
           MPI_Finalize();
     }
}



int main(int argc,char * argv[])
{
     if(argc==1)
     {
          printf("请输入数据量!\n");
     }
     else
     {
           if(argc==2)
          {
               printf("请输入线程数!\n");
          }
          else
           {
               keyType max;
               List L;
               DATASIZE=atol(argv[1]);
               L.data=(keyType*)malloc(sizeof(keyType)*DATASIZE);
               readFile(L);
                 int thread=atol(argv[2]);
                 omp_set_num_threads(thread);
               MaxParallel(argc, argv, L, max);
           }
     }
}
```

3.5 Random.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

long DATASIZE;

typedef int keyType;

typedef struct
{
	keyType *data;
	int length;
}List;

void Print(List &L)
{
	for (int i = 0; i < DATASIZE; i++)
	{
		printf("%d\n", L.data[i]);
	}
}

bool IsRepetition(List &L, int Length, keyType param)
{
	int i;
	for (i = 0; i < Length; i++)
	{
		if (L.data[i] == param)
			return true;
	}
	return false;
}

void Random(List &L)
{
	srand((unsigned)time(NULL));
	int i = 0;
	while (i < DATASIZE)
	{
		int temp = rand();
		L.data[i] = temp;
```

```
            i++;
        }
        L.length = DATASIZE;
}


void printFile(List &L)
{
        FILE *file = fopen("data.txt", "w");
        if (file==NULL)
        {
                printf("打开文件失败!");
        }
        for(int i=0;i<DATASIZE;i++)
        {
              fprintf(file,"%d ",L.data[i]);
        }
        fclose(file);
}

int main(int argc,char *argv[])
{
       if(argc==1)
       {
              printf("请输入要产生的随机数个数！\n");
       }
       else
       {
              List L;
              DATASIZE=atol(argv[1]);
              L.data=(keyType*)malloc(sizeof(keyType)*DATASIZE);
              Random(L);
              printFile(L);
       }
}
```

3.6 Max.sh

```
#!/bin/bash

yes="y"
no="n"
clear
mpic++ Max.cpp -o Max
```

```
g++    -fopenmp Omp_Max.cpp -o Omp_Max
mpic++ -fopenmp -o Omp_MPI_Max Omp_MPI_Max.cpp
g++ Random.cpp -o Random
echo ".............需要产生数据吗?(y/n)............."
read console
clear
if [ $console == $yes ]
then
    echo ".............请输入产生的数据量大小!............."
    read number
    clear
    ./Random $number
    echo ""
    echo ".............数据产生成功!............."
    echo ""
    echo ".............请输入进程个数!............."
    echo ""
    read process
    clear
    echo ".............请输入线程个数!............."
    echo ""
    read thread
    clear
    echo ".............程序开始运行!............."
    echo "MPI Program is running!"
    mpirun -np $process ./Max $number
    echo "openMP Program is running!"
    ./Omp_Max $number $thread
    echo "openMP&MPI Program is running!"
    mpirun -np $process ./Omp_MPI_Max $number $thread
fi

if [ $console == $no ]
then
    echo ".............请输入数据量大小!............."
    echo ""
    read number
    clear
    echo ".............请输入进程个数!............."
    echo ""
    read process
    clear
    echo ".............请输入线程个数!............."
    echo ""
```

```
        read thread
        clear
        echo "程序开始运行!"
        echo "MPI Program is running!"
        mpirun -np $process ./Max $number
        echo "openMP Program is running!"
        ./Omp_Max $number $thread
        echo "openMP&MPI Program is running!"
        mpirun -np $process ./Omp_MPI_Max $number $thread
fi
```

第四章 Sort

4.1 mySort.h

```
#pragma once
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

#ifdef MYMPI
#include <mpi.h>
#endif

#ifdef OPENMP
#include <omp.h>
#endif

long DATASIZE;
#define TIMESIZE 1000000

typedef int keyType;

typedef struct
{
	keyType *data;
	int length;
}List;
```

4.2 Sort.cpp

```
#define MYMPI
#include "mySort.h"

void readFile(List &L)
{
	FILE *file = fopen("data.txt", "r");
	if (file==NULL)
	{
```

```
            printf("打开文件失败!");
        }
        for(int i=0;i<DATASIZE;i++)
        {
            fscanf(file,"%d ",&(L.data[i]));
        }
        fclose(file);
}


void EnumSort(List &L,List &S)
{
        for(int i=0;i<DATASIZE;i++)
        {
            int k=0,data=L.data[i];
            for(int j=0;j<DATASIZE;j++)
            {
                if(data>L.data[j])
                {
                    k++;
                }
            }
            S.data[k]=data;
        }
}


void EnumSortParallel(int argc, char *argv[],List &L,List &S)
{
        int rank, size;
        MPI_Status status;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        if(size==1)
        {
            double first,last,time;
            first=MPI_Wtime();
            EnumSort(L,S);
            last=MPI_Wtime();
            time=last-first;
            printf("TIME:%lf\n",time*TIMESIZE);
        }
        else
```

```
    {
        if(rank==0)
        {
            double first,last,time,flag;
            first=MPI_Wtime();
            int start,end,dest;
            start=0;
            end=DATASIZE/size;
            for(dest=1;dest<size;dest++)
            {
                MPI_Send(&start, 1, MPI_INT, dest, 1, MPI_COMM_WORLD);
                MPI_Send(&end, 1, MPI_INT, dest, 1, MPI_COMM_WORLD);
                start=start+end;
            }
            for(int i=start;i<DATASIZE;i++)
            {
                int k=0,data=L.data[i];
                for(int j=0;j<DATASIZE;j++)
                {
                    if(data>L.data[j])
                    {
                        k++;
                    }
                }
                S.data[k]=data;
            }
            for(dest=1;dest<size;dest++)
            {
                MPI_Recv(&flag, 1, MPI_INT, dest, 2, MPI_COMM_WORLD,
&status);
            }
            last=MPI_Wtime();
            time=last-first;
            printf("TIME:%lf\n",time*TIMESIZE);
        }
        if(rank>0)
        {
            int start,end,source;
            source=0;
            MPI_Recv(&start, 1, MPI_INT, source, 1, MPI_COMM_WORLD,
&status);
            MPI_Recv(&end, 1, MPI_INT, source, 1, MPI_COMM_WORLD,
&status);
            for(int i=start;i<end;i++)
```

```
            {
                int k=0,data=L.data[i];
                for(int j=0;j<DATASIZE;j++)
                {
                    if(data>L.data[j])
                    {
                        k++;
                    }
                }
                S.data[k]=data;
            }
            MPI_Send(&source, 1, MPI_INT, source, 2, MPI_COMM_WORLD);
        }
        MPI_Finalize();
    }
}


int main(int argc, char *argv[])
{
    if(argc==1)
    {
        printf("请输入数据量!");
    }
    else
    {
        List L,S;
            DATASIZE=atol(argv[1]);
        L.data=(keyType*)malloc(sizeof(keyType)*(DATASIZE));
        S.data=(keyType*)malloc(sizeof(keyType)*(DATASIZE));
        readFile(L);
        EnumSortParallel(argc,argv,L,S);
    }
}
```

4.3 Omp_Sort.cpp

```
#define OPENMP
#include "mySort.h"

void readFile(List &L)
{
    FILE *file = fopen("data.txt", "r");
    if (file==NULL)
```

```
    {
        printf("打开文件失败!");
    }
    for(int i=0;i<DATASIZE;i++)
    {
        fscanf(file,"%d ",&(L.data[i]));
    }
    fclose(file);
}

void OddEvenSort(List &L)
{
    int i, j;
     double first,last,time;
     first=omp_get_wtime();
    for (i = 1; i < DATASIZE + 1; i++)
    {
        if (1 == (i % 2 ))
        {
#pragma omp parallel for
            for (j = 1; j < DATASIZE + 1; j+=2)
            {
                if (L.data[j]>L.data[j + 1])
                {
                    int temp = L.data[j];
                    L.data[j] = L.data[j + 1];
                    L.data[j + 1] = temp;
                }
            }
        }
        else
        {
#pragma omp parallel for
            for (j = 2; j < DATASIZE; j += 2)
            {
                if (L.data[j]>L.data[j + 1])
                {
                    int temp = L.data[j];
                    L.data[j] = L.data[j + 1];
                    L.data[j + 1] = temp;
                }
            }
        }
    }
```

```
    last=omp_get_wtime();
    time=last-first;
    printf("TIME:%lf\n",time);
}

int main(int argc, char *argv[])
{
    if(argc==1)
    {
        printf("请输入数据量!");
    }
    else
    {
        List L,S;
            DATASIZE=atol(argv[1]);
        L.data=(keyType*)malloc(sizeof(keyType)*(DATASIZE));
        readFile(L);
          int thread=atol(argv[2]);
        omp_set_num_threads(thread);
        OddEvenSort(L);
    }
}
```

4.4 Omp_MPI_Sort.cpp

```
#define MYMPI
#define OPENMP
#include "mySort.h"

void readFile(List &L)
{
    FILE *file = fopen("data.txt", "r");
    if (file==NULL)
    {
        printf("打开文件失败!");
    }
    for(int i=0;i<DATASIZE;i++)
    {
        fscanf(file,"%d ",&(L.data[i]));
    }
    fclose(file);
}
```

```
void EnumSort(List &L,List &S)
{
	#pragma omp parallel private(i,j,k)
	{
		#pragma omp for
		for(int i=0;i<DATASIZE;i++)
		{
			int k=0,data=L.data[i];
			for(int j=0;j<DATASIZE;j++)
			{
				if(data>L.data[j])
				{
					k++;
				}
			}
			S.data[k]=data;
		}
	}
}


void EnumSortParallel(int argc, char *argv[],List &L,List &S)
{
	int rank, size;
	MPI_Status status;
	MPI_Init(&argc, &argv);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	MPI_Comm_size(MPI_COMM_WORLD, &size);
	if(size==1)
	{
		double first,last,time;
		first=MPI_Wtime();
		EnumSort(L,S);
		last=MPI_Wtime();
		time=last-first;
		printf("TIME:%lf\n",time*TIMESIZE);
	}
	else
	{
		if(rank==0)
		{
			double first,last,time,flag;
			first=MPI_Wtime();
			int start,end,dest;
```

```
            start=0;
            end=DATASIZE/size;
            for(dest=1;dest<size;dest++)
            {
                MPI_Send(&start, 1, MPI_INT, dest, 1, MPI_COMM_WORLD);
                MPI_Send(&end, 1, MPI_INT, dest, 1, MPI_COMM_WORLD);
                start=start+end;
            }
              #pragma omp parallel private(i,j,k)
              {
                    #pragma omp for
                 for(int i=start;i<DATASIZE;i++)
                 {
                      int k=0,data=L.data[i];
                      for(int j=0;j<DATASIZE;j++)
                      {
                           if(data>L.data[j])
                           {
                                k++;
                           }
                      }
                      S.data[k]=data;
                 }
              }
            for(dest=1;dest<size;dest++)
            {
                MPI_Recv(&flag, 1, MPI_INT, dest, 2, MPI_COMM_WORLD,
&status);
            }
            last=MPI_Wtime();
            time=last-first;
            printf("TIME:%lf\n",time*TIMESIZE);
        }
        if(rank>0)
        {
            int start,end,source;
            source=0;
            MPI_Recv(&start, 1, MPI_INT, source, 1, MPI_COMM_WORLD,
&status);
            MPI_Recv(&end, 1, MPI_INT, source, 1, MPI_COMM_WORLD,
&status);
              #pragma omp parallel private(i,j,k)
              {
                    #pragma omp for
```

```
                    for(int i=start;i<end;i++)
                    {
                        int k=0,data=L.data[i];
                        for(int j=0;j<DATASIZE;j++)
                        {
                            if(data>L.data[j])
                            {
                                k++;
                            }
                        }
                        S.data[k]=data;
                    }
                }
            MPI_Send(&source, 1, MPI_INT, source, 2, MPI_COMM_WORLD);
        }
        MPI_Finalize();
    }
}


int main(int argc, char *argv[])
{
    if(argc==1)
    {
        printf("请输入数据量!");
    }
    else
    {
         if(argc==2)
        {
            printf("请输入线程数!\n");
        }
        else
         {
            List L,S;
                DATASIZE=atol(argv[1]);
            L.data=(keyType*)malloc(sizeof(keyType)*(DATASIZE));
            S.data=(keyType*)malloc(sizeof(keyType)*(DATASIZE));
            readFile(L);
              int thread=atol(argv[2]);
              omp_set_num_threads(thread);
            EnumSortParallel(argc,argv,L,S);
         }
    }
```

```
}
```

4.5 Random.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

long DATASIZE;
typedef int keyType;

typedef struct
{
	keyType *data;
	int length;
}List;

void Print(List &L)
{
	int i = 0;
	for (i = 0; i < DATASIZE; i++)
	{
		printf("%d\n", L.data[i]);
	}
	printf("-----------------------------------------\n");
}

void Random(List &L)
{
	srand((unsigned)time(NULL));
	int i = 0;
	while (i < DATASIZE)
	{
		int temp = rand();
		L.data[i] = temp;
		i++;
	}
	L.length = DATASIZE;
}

void printFile(List &L)
{
	FILE *file = fopen("data.txt", "w");
	if (file==NULL)
```

```
    {
        printf("打开文件失败!");
    }
    for(int i=0;i<DATASIZE;i++)
    {
      fprintf(file,"%d ",L.data[i]);
    }
    fclose(file);
}

int main(int argc,char *argv[])
{
    if(argc==1)
    {
        printf("请输入要产生的随机数个数！\n");
    }
    else
    {
        List L;
        DATASIZE=atol(argv[1]);
        L.data=(keyType*)malloc(sizeof(keyType)*(DATASIZE+1));
        Random(L);
        printFile(L);
    }
}
```

4.6 Sort.sh

```
#!/bin/bash

yes="y"
no="n"
clear
mpic++ Sort.cpp -o Sort
g++ -fopenmp Omp_Sort.cpp -o Omp_Sort
mpic++ -fopenmp -o Omp_MPI_Sort Omp_MPI_Sort.cpp
g++ Random.cpp -o Random
echo "..............需要产生数据吗?(y/n)............."
read console
clear
if [ $console == $yes ]
then
    echo "..............请输入产生的数据量大小!............."
    read number
```

```
        clear
        ./Random $number
        echo ""
        echo ".............数据产生成功!............."
        echo ""
        echo ".............请输入进程个数!............."
        echo ""
        read process
        clear
        echo ".............请输入线程个数!............."
        echo ""
        read thread
        clear
        echo ".............程序开始运行!............."
        echo "MPI Program is running!"
        mpirun -np $process ./Sort $number
        echo "openMP Program is running!"
        ./Omp_Sort $number $thread
        echo "openMP&MPI Program is running!"
        mpirun -np $process ./Omp_MPI_Sort $number $thread
fi

if [ $console == $no ]
then
        echo ".............请输入数据量大小!............."
        echo ""
        read number
        clear
        echo ".............请输入进程个数!............."
        echo ""
        read process
        clear
        echo ".............请输入线程个数!............."
        echo ""
        read thread
        clear
        echo "程序开始运行!"
        echo "MPI Program is running!"
        mpirun -np $process ./Sort $number
        echo "openMP Program is running!"
        ./Omp_Sort $number $thread
        echo "openMP&MPI Program is running!"
        mpirun -np $process ./Omp_MPI_Sort $number $thread
fi
```

第五章 UI

5.1 UI.c

```
#include <gtk/gtk.h>
#include <gtk/gtkmain.h>
#include <unistd.h>

pid_t pid;
int file;
int boolean;
int mytime;
int readFlag;
long times;
guint timer;

GtkWidget *window;
GtkWidget *dialog;

GtkWidget *openFileButton;
GtkWidget *runButton;
GtkWidget *cancelButton;

GtkWidget *fileNameLabel;
GtkWidget *processLabel;
GtkWidget *threadLabel;
GtkWidget *dataLabel;
GtkWidget *patternLabel;

GtkWidget *fileNameEntry;
GtkWidget *processEntry;
GtkWidget *threadEntry;
GtkWidget *dataEntry;
GtkWidget *patternEntry;

GtkWidget *image;
GtkWidget *textView;

GtkWidget *FileSelection;
```

```
GtkWidget *hbox_one;
GtkWidget *hbox_two;
GtkWidget *hbox_three;
GtkWidget *hbox_four;
GtkWidget *hbox_five;
GtkWidget *vbox;


char string[500];
char span[1000];
char _span[10];
char _time[100];
char pattern[100];
char processName[100];
char processNumber[100];
char threadNumber[100];
char dataNumber[100];



void initData()
{
	memset(processNumber,0,100);
	memset(threadNumber,0,100);
	memset(dataNumber,0,100);
	memset(pattern,0,100);
	memset(string,0,500);
	memset(span,0,1000);
	memset(_span,0,10);
	memset(_time,0,100);
	memset(processName,0,100);
}

char *readFile()
{
	FILE *fp;
	 char *str;
	 char result[1000];
	 int filesize;
	 if ((fp=fopen("result.txt","r"))==NULL){
		printf("打开文件失败\n");
		return;
	 }
```

```
    fseek(fp,0,SEEK_END);
    filesize = ftell(fp);
    str=(char *)malloc(filesize);
   memset(str,0,filesize);
    str[filesize]=0;
    rewind(fp);
    while((fgets(result,1000,fp))!=NULL){
        strcat(str,result);
    }
    fclose(fp);
    return str;
}


gboolean runTimer(gpointer data)
{
   if(boolean==0)
   {
      memset(_time,0,100);
      if(file!=1)
         strcpy(string,"  计时器准备就绪");
   }
   else
   {
      sprintf(_time,"%ld", times);
      strcpy(string,"计算中:");
      times++;
      if(times==mytime)
      {
         strcpy(string,readFile());
         file=1;
         boolean=0;
         times=0;
         readFlag=1;
         gtk_widget_hide(image);
      }
   }
   if(readFlag==1)
      strcpy(string,readFile());
   strcpy(span,"<span foreground='black' font_desc='14'>");
   strcpy(_span,"</span>");
   strcat(span,string);
   strcat(span,_time);
   strcat(span,_span);
```

```
	gtk_label_set_markup(GTK_LABEL(textView),span);
	 return TRUE;
}

void createProcess()
{
	char temp[500];
	memset(temp,0,500);
	strcat(temp,processName);
	strcat(temp," ");
	strcat(temp,pattern);
	strcat(temp," ");
	strcat(temp,processNumber);
	strcat(temp," ");
	strcat(temp,threadNumber);
	strcat(temp," ");
	strcat(temp,dataNumber);
	strcat(temp," ");
	strcat(temp,"&");
	system(temp);
}


void hideWindow(GtkWidget *widget, gpointer data)
{
	gtk_widget_hide(FileSelection);
}


void getFile(GtkWidget *widget, gpointer data)
{
	char temp[255]="程序名:";
	strcpy(processName,gtk_file_selection_get_filename  (GTK_FILE_SELECTION
(FileSelection)));
	strcat(temp,processName);
	gtk_entry_set_text(fileNameEntry,processName);
	gtk_widget_hide(FileSelection);
}

void openFile(GtkWidget *widget, gpointer data)
{
	if(FileSelection!=NULL)
	{
		gtk_widget_show (FileSelection);
```

```
    }
    else
    {
        FileSelection = gtk_file_selection_new("程序选择");
        gtk_file_selection_set_filename (GTK_FILE_SELECTION (FileSelection),
"exec.sh");
        g_signal_connect (GTK_OBJECT(FileSelection), "destroy",G_CALLBACK
(hideWindow), NULL);

    g_signal_connect(GTK_OBJECT(GTK_FILE_SELECTION(FileSelection)->ok_
button),"clicked",G_CALLBACK(getFile),FileSelection);

    g_signal_connect(GTK_OBJECT(GTK_FILE_SELECTION(FileSelection)->can
cel_button),"clicked",G_CALLBACK(hideWindow),FileSelection);

    gtk_file_selection_set_filename(GTK_FILE_SELECTION(FileSelection),"*");
    }

}

void runProgram(GtkWidget *widget, gpointer data)
{
    if(strlen(processName))
    {
        strcpy(pattern,gtk_entry_get_text(patternEntry));
        if(strlen(pattern))
        {
            strcpy(processNumber,gtk_entry_get_text(processEntry));
            if(strlen(processNumber))
            {
                strcpy(threadNumber,gtk_entry_get_text(threadEntry));
                if(strlen(threadNumber))
                {
                    strcpy(dataNumber,gtk_entry_get_text(dataEntry));
                    if(strlen(dataNumber))
                    {

    if(strcmp(pattern,"a")==0||strcmp(pattern,"b")==0||strcmp(pattern,"c")==0)
                            {
                                boolean=1;
                                times=0;
                                file=0;
                                readFlag=0;
```

```
                    mytime=rand()%10+8;

                    memset(string,0,100);
                    memset(span,0,300);
                    memset(_span,0,10);

                    strcpy(string,"计算中.....");
                    strcpy(span,"<span                    foreground='black'
font_desc='14'>");
                    strcpy(_span,"</span>");
                    strcat(span,string);
                    strcat(span,_span);

                    gtk_label_set_markup(GTK_LABEL(textView),span);
                    gtk_widget_show(image);

                    if(strcmp(pattern,"a")==0)
                        strcpy(pattern,"Matrix");
                    if(strcmp(pattern,"b")==0)
                        strcpy(pattern,"Max");
                    if(strcmp(pattern,"c")==0)
                        strcpy(pattern,"Sort");

                    createProcess();

                    }
                    else
                    {
                        dialog          =          gtk_message_dialog_new
(GTK_WINDOW (window),
                                                   GTK_DIALOG_MODAL |

    GTK_DIALOG_DESTROY_WITH_PARENT,
                                                   GTK_MESSAGE_INFO,
                                                   GTK_BUTTONS_OK,
                                                   "必须填写合法的类型{a 代
表 Matrix,b 代表 Max,c 代表 Sort}！");
                        gtk_message_dialog_format_secondary_text
(GTK_MESSAGE_DIALOG (dialog),"%s\n", "\n\nProduce By SiwangHu");
                        gtk_dialog_run (GTK_DIALOG (dialog));
                        gtk_widget_destroy (dialog);
                    }
                }
                else
```

```
                    {
                        dialog = gtk_message_dialog_new (GTK_WINDOW
(window),
                                                         GTK_DIALOG_MODAL |

    GTK_DIALOG_DESTROY_WITH_PARENT,
                                                         GTK_MESSAGE_INFO,
                                                         GTK_BUTTONS_OK,
                                                         "必须填写数据量！");
                        gtk_message_dialog_format_secondary_text
(GTK_MESSAGE_DIALOG (dialog),"%s\n", "\n\nProduce By SiwangHu");
                        gtk_dialog_run (GTK_DIALOG (dialog));
                        gtk_widget_destroy (dialog);
                    }
                }
                else
                {
                    dialog = gtk_message_dialog_new (GTK_WINDOW (window),
                                                         GTK_DIALOG_MODAL |

    GTK_DIALOG_DESTROY_WITH_PARENT,
                                                         GTK_MESSAGE_INFO,
                                                         GTK_BUTTONS_OK,
                                                         "必须填写线程数目！");
                    gtk_message_dialog_format_secondary_text
(GTK_MESSAGE_DIALOG (dialog),"%s\n", "\n\nProduce By SiwangHu");
                    gtk_dialog_run (GTK_DIALOG (dialog));
                    gtk_widget_destroy (dialog);
                }

            }
            else
            {
                dialog = gtk_message_dialog_new (GTK_WINDOW (window),
                                                         GTK_DIALOG_MODAL |

    GTK_DIALOG_DESTROY_WITH_PARENT,
                                                         GTK_MESSAGE_INFO,
                                                         GTK_BUTTONS_OK,
                                                         "必须填写进程数目！");
                gtk_message_dialog_format_secondary_text
(GTK_MESSAGE_DIALOG (dialog),"%s\n", "\n\nProduce By SiwangHu");
                gtk_dialog_run (GTK_DIALOG (dialog));
                gtk_widget_destroy (dialog);
```

```
            }
        }
        else
        {
                dialog = gtk_message_dialog_new (GTK_WINDOW (window),
                                                 GTK_DIALOG_MODAL |

    GTK_DIALOG_DESTROY_WITH_PARENT,
                                                 GTK_MESSAGE_INFO,
                                                 GTK_BUTTONS_OK,
                                                 "必须填写类型{a 代表 Matrix,b 代表
Max,c 代表 Sort}！ ");
                gtk_message_dialog_format_secondary_text
(GTK_MESSAGE_DIALOG (dialog),"%s\n", "\n\nProduce By SiwangHu");
                gtk_dialog_run (GTK_DIALOG (dialog));
                gtk_widget_destroy (dialog);
        }
    }
    else
    {
        dialog = gtk_message_dialog_new (GTK_WINDOW (window),
                                         GTK_DIALOG_MODAL |

    GTK_DIALOG_DESTROY_WITH_PARENT,
                                         GTK_MESSAGE_INFO,
                                         GTK_BUTTONS_OK,
                                         "必须先选择运行的程序！ ");
        gtk_message_dialog_format_secondary_text     (GTK_MESSAGE_DIALOG
(dialog),"%s\n", "\n\nProduce By SiwangHu");
        gtk_dialog_run (GTK_DIALOG (dialog));
        gtk_widget_destroy (dialog);
    }
}

void stopProgram(GtkWidget *widget, gpointer data)
{
    boolean=0;
    times=0;
    memset(string,0,500);
    memset(span,0,1000);
    memset(_span,0,10);
    strcpy(string," 计时器准备就绪");
    strcpy(span,"<span foreground='black' font_desc='14'>");
    strcpy(_span,"</span>");
```

```
    strcat(span,string);
    strcat(span,_span);
    gtk_label_set_markup(GTK_LABEL(textView),span);
    gtk_widget_hide(image);
}

gint delete_event( GtkWidget *widget,GdkEvent *event,gpointer data )
{
    gtk_main_quit ();
    return TRUE;
}

void destroy(GtkWidget *widget,gpointer data)
{
    gtk_main_quit ();
}


GdkPixbuf *loadImage(const gchar* filename)
{
     GdkPixbuf *pixbuf;
     GError *error = NULL;
     pixbuf = gdk_pixbuf_new_from_file(filename, &error);
     if(!pixbuf) {
          fprintf(stderr, "%s\n", error->message);
          g_error_free(error);
     }
     return pixbuf;
}


int main(int argc, char *argv[])
{
    initData();
    gtk_init (&argc, &argv);
     window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    timer = g_timeout_add(1000, (GSourceFunc)runTimer, NULL);

    FileSelection = gtk_file_selection_new("程序选择");
      g_signal_connect (G_OBJECT (FileSelection), "delete_event",G_CALLBACK
(hideWindow), NULL);
    g_signal_connect(GTK_OBJECT(GTK_FILE_SELECTION(FileSelection)->ok_
button),"clicked",G_CALLBACK(getFile),FileSelection);
    g_signal_connect(GTK_OBJECT(GTK_FILE_SELECTION(FileSelection)->can
```

```
cel_button),"clicked",G_CALLBACK(hideWindow),FileSelection);
    gtk_file_selection_set_filename(GTK_FILE_SELECTION(FileSelection),"*");

    gtk_window_set_title(GTK_WINDOW(window),"并行计算");
    g_signal_connect (G_OBJECT (window), "delete_event",G_CALLBACK
(delete_event), NULL);
     g_signal_connect (G_OBJECT (window), "destroy",G_CALLBACK (destroy),
NULL);
    gtk_widget_set_size_request(window,500,380);
    gtk_window_set_resizable(GTK_WINDOW(window),FALSE);
    gtk_container_set_border_width(GTK_CONTAINER(window),5);
    gtk_window_set_icon(GTK_WINDOW(window), loadImage("./Main.png"));


    fileNameLabel=gtk_label_new("文件");
    gtk_label_set_markup(GTK_LABEL(fileNameLabel),                "<span
foreground='black' font_desc='15'>文件</span>");
    gtk_widget_show (fileNameLabel);


    fileNameEntry=gtk_entry_new();
    gtk_editable_set_editable(fileNameEntry,FALSE);
    gtk_entry_set_max_length(fileNameEntry, 255);
    gtk_widget_show (fileNameEntry);

    openFileButton=gtk_button_new_with_label("打开");
    g_signal_connect(G_OBJECT(openFileButton),
"clicked",G_CALLBACK(openFile), "clicked");
    gtk_widget_show (openFileButton);

    patternLabel=gtk_label_new("类型");
    gtk_label_set_markup(GTK_LABEL(patternLabel),   "<span foreground='black'
font_desc='15'>类型</span>");
    gtk_widget_show (patternLabel);


    patternEntry=gtk_entry_new();
    gtk_entry_set_max_length(patternEntry, 255);
    gtk_widget_show (patternEntry);

    hbox_one= gtk_hbox_new(FALSE,2);
    gtk_box_pack_start(hbox_one,fileNameLabel,FALSE,FALSE,5);
    gtk_box_pack_start(hbox_one,fileNameEntry,TRUE,TRUE,5);
    gtk_box_pack_start(hbox_one,openFileButton,FALSE,FALSE,5);
```

```
    gtk_box_pack_start(hbox_one,patternLabel,FALSE,FALSE,5);
    gtk_box_pack_start(hbox_one,patternEntry,FALSE,FALSE,5);



    processLabel=gtk_label_new("进程数");
    gtk_label_set_markup(GTK_LABEL(processLabel),  "<span foreground='black'
font_desc='15'>进程数</span>");
    gtk_widget_show (processLabel);


    processEntry=gtk_entry_new();
    gtk_editable_set_editable(processEntry,TRUE);
    gtk_entry_set_max_length(processEntry, 255);
    gtk_widget_show (processEntry);


    threadLabel=gtk_label_new("线程数");
    gtk_label_set_markup(GTK_LABEL(threadLabel),  "<span  foreground='black'
font_desc='15'>线程数</span>");
    gtk_widget_show (threadLabel);


    threadEntry=gtk_entry_new();
    gtk_editable_set_editable(threadEntry,TRUE);
    gtk_entry_set_max_length(threadEntry, 255);
    gtk_widget_show (threadEntry);

    hbox_two=gtk_hbox_new(FALSE,2);
    gtk_box_pack_start(hbox_two,processLabel,FALSE,FALSE,5);
    gtk_box_pack_start(hbox_two,processEntry,TRUE,TRUE,5);
    gtk_box_pack_start(hbox_two,threadLabel,FALSE,FALSE,0);
    gtk_box_pack_start(hbox_two,threadEntry,FALSE,FALSE,0);

    dataLabel=gtk_label_new("数据量");
    gtk_label_set_markup(GTK_LABEL(dataLabel),  "<span  foreground='black'
font_desc='15'>数据量</span>");
    gtk_widget_show (dataLabel);


    dataEntry=gtk_entry_new();
    gtk_editable_set_editable(dataEntry,TRUE);
    gtk_entry_set_max_length(dataEntry, 255);
    gtk_widget_show (dataEntry);
```

```
    runButton=gtk_button_new_with_label("运行");
    g_signal_connect(G_OBJECT(runButton),
"clicked",G_CALLBACK(runProgram), "clicked");
    gtk_widget_show (runButton);

    cancelButton=gtk_button_new_with_label("停止");
    g_signal_connect(G_OBJECT(cancelButton),
"clicked",G_CALLBACK(stopProgram), "clicked");
    gtk_widget_show (cancelButton);

    hbox_three=gtk_hbox_new(FALSE,2);
    gtk_box_pack_start(hbox_three,dataLabel,FALSE,FALSE,5);
    gtk_box_pack_start(hbox_three,dataEntry,TRUE,TRUE,5);
    gtk_box_pack_start(hbox_three,runButton,FALSE,FALSE,5);
    gtk_box_pack_start(hbox_three,cancelButton,FALSE,FALSE,5);


    hbox_four=gtk_hbox_new(FALSE,2);
    textView=gtk_label_new("  计时器准备就绪");
    memset(string,0,100);
    memset(span,0,300);
    memset(_span,0,10);
    strcpy(string,"  计时器准备就绪");
    strcpy(span,"<span foreground='black' font_desc='15'>");
    strcpy(_span,"</span>");
    strcat(span,string);
    strcat(span,_span);
    gtk_label_set_markup(GTK_LABEL(textView), span);
    gtk_widget_show (textView);
    gtk_box_pack_start(hbox_four,textView,TRUE,TRUE,0);

    hbox_five=gtk_hbox_new(FALSE,2);
    image = gtk_image_new_from_file("./wait.gif");
    gtk_widget_hide(image);
    gtk_box_pack_start(hbox_five,image,TRUE,TRUE,0);

    vbox=gtk_vbox_new(FALSE,2);
    gtk_box_pack_start(vbox,hbox_one,FALSE,FALSE,5);
    gtk_box_pack_start(vbox,hbox_two,FALSE,FALSE,5);
    gtk_box_pack_start(vbox,hbox_three,FALSE,FALSE,20);
    gtk_box_pack_start(vbox,hbox_four,FALSE,FALSE,5);
    gtk_box_pack_start(vbox,hbox_five,FALSE,FALSE,5);
```

```
    gtk_container_add(window,vbox);
    gtk_widget_show (hbox_one);
    gtk_widget_show (hbox_two);
    gtk_widget_show (hbox_three);
    gtk_widget_show (hbox_four);
    gtk_widget_show (hbox_five);
    gtk_widget_show (vbox);
     gtk_widget_show (window);
    gtk_main ();

     return 0;
}
```

5.2 exec.sh

```
#!/bin/bash

Matrix="Matrix"
Max="Max"
Sort="Sort"
pattern=$1
processNumber=$2
threadNumber=$3
dataNumber=$4

if [ $pattern == $Matrix ]
then
      rm -f ./result.txt
      touch result.txt
      cd ..
      cd Matrix
      mpic++ Matrix.cpp -o Matrix
      mpic++ Omp_Matrix.cpp -o Omp_Matrix -fopenmp -lpthread
      mpic++ -fopenmp -o Omp_MPI_Matrix Omp_MPI_Matrix.cpp
      g++ Random.cpp -o Random
      ./Random $dataNumber
      echo "MPI Program" >> ../UI/result.txt
     mpirun -np $processNumber ./Matrix $dataNumber >> ../UI/result.txt
     echo "openMP Program" >> ../UI/result.txt
     ./Omp_Matrix $dataNumber $threadNumber >> ../UI/result.txt
     echo "openMP and MPI Program" >> ../UI/result.txt
```

```
    mpirun -np $processNumber ./Omp_MPI_Matrix $dataNumber
$threadNumber >> ../UI/result.txt
fi

if [ $pattern == $Max ]
then
      rm -f ./result.txt
      touch result.txt
      cd ..
      cd Max
      mpic++ Max.cpp -o Max
      mpic++    Omp_Max.cpp -o Omp_Max -fopenmp -lpthread
      mpic++ -fopenmp -o Omp_MPI_Max Omp_MPI_Max.cpp
      g++ Random.cpp -o Random
      ./Random $dataNumber
      echo "MPI Program" >> ../UI/result.txt
     mpirun -np $processNumber ./Max $dataNumber>> ../UI/result.txt
     echo "openMP Program" >> ../UI/result.txt
     ./Omp_Max $dataNumber $threadNumber >> ../UI/result.txt
     echo "openMP and MPI Program" >> ../UI/result.txt
     mpirun -np $processNumber ./Omp_MPI_Max $dataNumber
$threadNumber >> ../UI/result.txt
fi

if [ $pattern == $Sort ]
then
      rm -f ./result.txt
      touch result.txt
      cd ..
      cd Sort
      mpic++ Sort.cpp -o Sort
      mpic++ Omp_Sort.cpp -o Omp_Sort -fopenmp -lpthread
      mpic++ -fopenmp -o Omp_MPI_Sort Omp_MPI_Sort.cpp
      g++ Random.cpp -o Random
      ./Random $dataNumber
      echo "MPI Program" >> ../UI/result.txt
     mpirun -np $processNumber ./Sort $dataNumber >> ../UI/result.txt
     echo "openMP Program" >> ../UI/result.txt
     ./Omp_Sort $dataNumber $threadNumber >> ../UI/result.txt
     echo "openMP and MPI Program" >> ../UI/result.txt
     mpirun -np $processNumber ./Omp_MPI_Sort $dataNumber
$threadNumber >> ../UI/result.txt
fi
```

第六章 run.sh

6.1 run.sh

```
#!/bin/bash

yes="y"
no="n"
matrix="1"
max="2"
sort="3"
clear
echo ".............需要图形化测试界面吗?(n/y)............."
read console
clear
if [ $console == $no ]
then
	clear
	echo ".............请输入操作(1.matrix/2.max/3.sort)!............."
	read operation
	if [ $operation == $matrix ]
	then
		cd Matrix
		./Matrix.sh
	fi
	if [ $operation == $max ]
	then
		cd Max
		./Max.sh
	fi
	if [ $operation == $sort ]
	then
		cd Sort
		./Sort.sh
	fi
fi
if [ $console == $yes ]
then
	cd UI
```

```
    gcc ./UI.c -o ./UI `pkg-config --cflags --libs gtk+-2.0`
    clear
    ./UI
fi
```