《并行计算》上机报告

姓名：	裴启智	学号：	PB18111793	日期：	2021.6.12
上机题目：	鲲鹏处理器上实现快速排序算法				

实验环境：华为鲲鹏云主机
CPU：2vCPUs　内存: 4GiB　操作系统：openEuler 20.03 64bit with ARM
软件平台：CloudShell

一、算法设计与分析：

通过在华为鲲鹏上，编译运行快排算法程序，掌握快排算法程序的编写。

二、核心代码：

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

#include <iostream>

#include "omp.h"

using namespace std;

void QuickSort(int *&array, int len) {
    if (len <= 1) return;
    int pivot = array[len / 2];
    int left_ptr = 0;
    int right_ptr = len - 1;
    while (left_ptr <= right_ptr) {
        while (array[left_ptr] < pivot) left_ptr += 1;
        while (array[right_ptr] > pivot) right_ptr -= 1;
        if (left_ptr <= right_ptr) {
            swap(array[left_ptr], array[right_ptr]);
            left_ptr += 1;
            right_ptr -= 1;
        }
    }
    int *sub_array[] = {array, &(array[left_ptr])};
    int sub_len[] = {right_ptr + 1, len - left_ptr};
```

```
#pragma omp task default(none) firstprivate(sub_array, sub_len)
    { QuickSort(sub_array[0], sub_len[0]); }
#pragma omp task default(none) firstprivate(sub_array, sub_len)
    { QuickSort(sub_array[1], sub_len[1]); }
    // for (int i = 0; i < 2; i++) QuickSort(sub_array[i], sub_len[i]);
}

int main(int argc, char *argv[]) {
    srand(time(NULL));
    if (argc != 3) {
        cout << "Usage: " << argv[0] << " thread-num array-len\n";
        exit(-1);
    }
    int t = atoi(argv[1]);
    int n = atoi(argv[2]);
    int *array = new int[n];
    omp_set_num_threads(t);

    unsigned int seed = 1024;
#pragma omp parallel for
    for (int i = 0; i < n; i++) array[i] = rand_r(&seed);

    struct timeval start, stop;
    gettimeofday(&start, NULL);

#pragma omp parallel default(none) shared(array, n)
    {
#pragma omp single nowait
        { QuickSort(array, n); }
    }
    gettimeofday(&stop, NULL);

    double elapse = (stop.tv_sec - start.tv_sec) * 1000 +
                    (stop.tv_usec - start.tv_usec) / 1000;
    cout << elapse << " " << n << endl;

    for (int i = 0; i < n - 1; i++) {
        if (array[i] > array[i + 1]) {
            cerr << "quick sort fails! \n";
            break;
        }
    }
    return 0;
}
```

三、结果与分析：

首先按照实验环境说明文档配置环境，安装完成后界面如下

```
/usr/bin/install -c -m 644 MPI_Wtime.html /usr/local/share/doc/mpich/www3/MPI_Wtime.html
/usr/bin/install -c -m 644 MPI_WTIME_IS_GLOBAL.htm /usr/local/share/doc/mpich/www3/MPI_WTIME_IS_GLOBAL
.htm
/usr/bin/install -c -m 644 MPIX_Comm_agree.html /usr/local/share/doc/mpich/www3/MPIX_Comm_agree.html
/usr/bin/install -c -m 644 MPIX_Comm_failure_ack.html /usr/local/share/doc/mpich/www3/MPIX_Comm_failur
e_ack.html
/usr/bin/install -c -m 644 MPIX_Comm_failure_get_acked.html /usr/local/share/doc/mpich/www3/MPIX_Comm_
failure_get_acked.html
/usr/bin/install -c -m 644 MPIX_Comm_revoke.html /usr/local/share/doc/mpich/www3/MPIX_Comm_revoke.html
/usr/bin/install -c -m 644 MPIX_Comm_shrink.html /usr/local/share/doc/mpich/www3/MPIX_Comm_shrink.html
if [ ! -e /usr/local/share/doc/mpich ] ; then mkdir -p /usr/local/share/doc/mpich ; fi
if [ -s ./doc/userguide/user.pdf ] ; then /usr/bin/install -c -m 644 ./doc/userguide/user.pdf /usr/loc
al/share/doc/mpich/user.pdf ; fi
if [ -s ./doc/installguide/install.pdf ] ; then /usr/bin/install -c -m 644 ./doc/installguide/install.
pdf /usr/local/share/doc/mpich/install.pdf ; fi
if [ -s ./doc/logging/logging.pdf ] ; then /usr/bin/install -c -m 644 ./doc/logging/logging.pdf /usr/l
ocal/share/doc/mpich/logging.pdf ; fi
 mkdir -p '/usr/local/include'
 /usr/bin/install -c -m 644 src/binding/fortran/use_mpi/mpi.mod src/binding/fortran/use_mpi/mpi_sizeof
s.mod src/binding/fortran/use_mpi/mpi_constants.mod src/binding/fortran/use_mpi/mpi_base.mod '/usr/loc
al/include'
 mkdir -p '/usr/local/include'
 /usr/bin/install -c -m 644 src/binding/cxx/mpicxx.h src/binding/fortran/mpif_h/mpif.h src/include/mpi
.h '/usr/local/include'
 mkdir -p '/usr/local/lib/pkgconfig'
 /usr/bin/install -c -m 644 src/packaging/pkgconfig/mpich.pc '/usr/local/lib/pkgconfig'
make[3]: Leaving directory '/home/pqz/mpich-3.3.2'
make[2]: Leaving directory '/home/pqz/mpich-3.3.2'
Making install in examples
make[2]: Entering directory '/home/pqz/mpich-3.3.2/examples'
make[3]: Entering directory '/home/pqz/mpich-3.3.2/examples'
make[3]: Nothing to be done for 'install-exec-am'.
make[3]: Nothing to be done for 'install-data-am'.
make[3]: Leaving directory '/home/pqz/mpich-3.3.2/examples'
make[2]: Leaving directory '/home/pqz/mpich-3.3.2/examples'
make[1]: Leaving directory '/home/pqz/mpich-3.3.2'
```

按照实验手册编写 quick_sort.cpp、Makefile、hostfile 和 run.sh

其中 hostfile 和源文件稍有区别，如下

```
ecs-1794:2
ecs-1795:2
```

指定不同的处理进程数，运行时间如下

```
[pqz@ecs-1794 quicksort]$ bash run.sh quicksort 1
1239 8000000
[pqz@ecs-1794 quicksort]$ bash run.sh quicksort 2
619 8000000
[pqz@ecs-1794 quicksort]$ bash run.sh quicksort 3
617 8000000
[pqz@ecs-1794 quicksort]$ bash run.sh quicksort 4
622 8000000
[pqz@ecs-1794 quicksort]$ bash run.sh quicksort 5
618 8000000
[pqz@ecs-1794 quicksort]$ bash run.sh quicksort 6
625 8000000
[pqz@ecs-1794 quicksort]$ bash run.sh quicksort 7
648 8000000
[pqz@ecs-1794 quicksort]$ bash run.sh quicksort 8
646 8000000
```

通过上述运行，可以看出快排算法程序已经在集群中并行运行起来。大致从整体可以看出，随着进程数量的增加，耗时越来越少。从开始的 1239 减少到 646 左右。

四、备注（* 可选）：

有可能影响结论的因素：服务器的数目、指定的进程数

思考题：链接过程进行了什么操作？静态链接器和动态链接器的区别是什么？
答：链接就是将不同部分的代码和数据收集和组合成为一个单一文件的过程,这个文件可被加载或拷贝到存储器执行，链接是由链接器自动执行的。
静态链接器以一组可重定位目标文件和命令行参数作为输入，生成一个完全链接的可以加载和运行的可执行目标文件作为输出。
共享库是一个目标模块,在运行时,可以加载到任意的存储器地址，并在存储器中和一个程序链接起来。这个过程称为动态链接，是由动态链接器完成的。

总结：

通过本次实验，我对如何在华为鲲鹏处理器上实现并运行快速排序程序有了更加深入的了解，对 OpenMP 的并行编程框架也有了更加深入的体会。收获颇丰

附录（源代码）

算法源代码（C/C++/JAVA 描述）

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

#include <iostream>

#include "omp.h"

using namespace std;

void QuickSort(int *&array, int len) {
    if (len <= 1) return;
    int pivot = array[len / 2];
    int left_ptr = 0;
    int right_ptr = len - 1;
    while (left_ptr <= right_ptr) {
        while (array[left_ptr] < pivot) left_ptr += 1;
        while (array[right_ptr] > pivot) right_ptr -= 1;
        if (left_ptr <= right_ptr) {
            swap(array[left_ptr], array[right_ptr]);
            left_ptr += 1;
            right_ptr -= 1;
        }
    }
    int *sub_array[] = {array, &(array[left_ptr])};
    int sub_len[] = {right_ptr + 1, len - left_ptr};

#pragma omp task default(none) firstprivate(sub_array, sub_
```

```
len)
    { QuickSort(sub_array[0], sub_len[0]); }
#pragma omp task default(none) firstprivate(sub_array, sub_
len)
    { QuickSort(sub_array[1], sub_len[1]); }
    // for (int i = 0; i < 2; i++) QuickSort(sub_array[i],
sub_len[i]);
}

int main(int argc, char *argv[]) {
    srand(time(NULL));
    if (argc != 3) {
        cout << "Usage: " << argv[0] << " thread-num array-
len\n";
        exit(-1);
    }
    int t = atoi(argv[1]);
    int n = atoi(argv[2]);
    int *array = new int[n];
    omp_set_num_threads(t);

    unsigned int seed = 1024;
#pragma omp parallel for
    for (int i = 0; i < n; i++) array[i] = rand_r(&seed);

    struct timeval start, stop;
    gettimeofday(&start, NULL);

#pragma omp parallel default(none) shared(array, n)
    {
#pragma omp single nowait
        { QuickSort(array, n); }
    }
    gettimeofday(&stop, NULL);

    double elapse = (stop.tv_sec - start.tv_sec) * 1000 +
                    (stop.tv_usec - start.tv_usec) / 1000;
    cout << elapse << " " << n << endl;

    for (int i = 0; i < n - 1; i++) {
        if (array[i] > array[i + 1]) {
            cerr << "quick sort fails! \n";
            break;
        }
```

	`` } return 0; } ```