**《并行计算》上机报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名： | 王章瀚 | | 学号： | PB18111697 | 日期： | 2021/5/29 |
| 上机题目： | Hadoop编程实验 | | | | | |
| 实验环境：  CPU架构: 鲲鹏通用计算增强型 kc1.large.2; 2块虚拟CPU;  内存: 4GB;  操作系统：OpenEular 20.03 64bit with ARM(40GB); | | | | | | |
| 1. **华为云配置** 2. 购买华为云   首先选择计费模式, CPU架构, 操作系统, 硬盘大小等信息      然后配置网络, 购买公网IP以便使用    配置密码等:    确认并购买:     1. 登录到各台机器进行基础配置   首先添加新用户, 并配置密码等:    然后配置IP地址别名:    切换用户:    对其他机器也做这样的操作.  然后用ssh-keygen生成公钥密钥对, 并配置到各个机器上, 以便免密登录:     1. **快速排序代码输入与编译**   复制好助教给出的代码及Makefile等:    新建并配置hostfile文件:     1. **快速排序运行结果分析**   运行代码    将得到:    如实验指导所述, 随着处理器数量增加, 快速排序的耗时从1243左右减少到了635左右.   1. **思考题**   问: 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：**  本次实验配置了华为云, 并用它运行了快速排序的算法. 非常有助于各种前沿科技的国产化进程! | | | | | | |
| 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;  } | | | | |