Homework2: Permutations and Sort

ECE 5720

Introduction to Parallel Computing

Code

Name: Yanling Wu

NetID: yw996

The description of codes of this word include:

1. yw996_hw2_openmp_sort_column.c (The problem 1, and including sequential and parallel code)
2. yw996_hw2_openmp_sort_block.c (The cyclic Row way)
3. Tile way's final_A Function
4. Block Row final_A Function
5. Block Column final_A Function
6. Cyclic Column final_A Function

The First two Parts are what we should upload, and the rest parts are what I used to explore the performance of the different ways but only includes the functions because the most part of the code are similar, and the swap function is same as the second part's.

1. yw996_hw2_openmp_sort_column.c

In this file, I firstly get the argument from the command line when you are executing the file. The arguments are the number of the row of the matrix and the number of the column of the matrix, and the number of threads that you want to open.

Then, I get the minimum of the row number and the column number as the dimension of matrix that need to be sorted. Initialize the matrix A, by rand() % 100 function using 1D pointer array.

Next, Start record the running time and set the number of the threads. Call the final_A(dim) function which is the parallel part.

> In final_A(dim) function, I used one variable maxVal to record the maximal value for every thread and one variable maxRow to record the according row number, and set them shared. Use one for-loop to control the number of column that need to be sorted, and then use "##pragma parallel omp for schedule(static) shared(maxRow, maxVal) " to declare the parallel part. In the parallel part, I used for-loop to control the row number and compare every value to the maxval, and if it is more than maxVal, then it will go into another omp critical part to compare the value to the current maxVal again and then give its row number to maxRow, give its value to maxVal, until it do the same thing to all rows. Finally, call the swap(j, i) function.
>
> In Swap(j, i) function, use "#pragma omp parallel for " to declare the parallel part and parallelly to swap the values of row.

Stop the time counting and print the running time.

Fourthly, initialize the A again, and start to count the running time and call the final_A_seq(dim) function.

> In final_A_seq(dim) function, most part of it is similar to the final_A(dim). The difference is final_A_seq(dim) does not use the parallel ways and call the swap_seq(I,j) function.
>
> Same as the swap_seq(I, j);

Finally, stop counting the running time and print the sequential running time.

2. yw996_hw2_openmp_sort_block.c

For this file, the main() function is similar to the first one. The different part is the final_A(dim) function and the swap(u, a, b) function.

In final_A(dim) function, use j to control the number of sorted elements in matrix, which means use one for-loop. And initiate the maxRow, maxCol, maxVal, using *"#pragma parallel omp for schedule(static, CHUNK_SIZE) shared(maxRow, maxVal, maxCol) collapse(2)"* to declare the parallel part. There are for loop following this declare statement to compare every element to the maxVal. If the element is more than maxVal, then using *"#pragma omp critical*" to change the values of maxRow, maxCol, maxVal. Do the same thing until compare all element in submatrix. And call the swap(j,maxRow, maxCol);

In swap(u, a, b) function, similar to the first problem, but this time, we need to swap, u-th row and a-th row and the u-th column and b-th column.

3. Tile way's final_A Function

```
void final_A(int dim) {
  int i = 0, j = 0, k = 0, a = 0, b = 0;
  int maxVal = -1;
  int maxRow = 0, maxCol = 0;
  for (j = 0; j < dim; j++) {
    maxRow = j,maxCol = j, maxVal = -1;
    for (a = j; a < m; a += tile_size)
      for (b = j; b < n; b += tile_size)
        #pragma parallel omp for schedule(static, CHUNK_SIZE) shared(maxRow, maxVal, maxCol) collapse(2)
          for(i = a; i < MIN(tile_size*(a+1), m); i++)
            for(k = b; k < MIN(tile_size * (b+1), n); k++)
              if (maxVal < abs(*(A + n * i + k))) {
                #pragma omp critical
                  if (maxVal < abs(*(A + n * i + k))) {
                    maxVal = abs(*(A + n * i + k));
                    maxRow = i;
                    maxCol = k;
                  }
              }
    swap(j, maxRow, maxCol);
  }
}
```

4. Block Row final_A Function

```
void final_A(int dim) {
  int i = 0, j = 0, k = 0;
  int maxVal = -1;
  int maxRow = 0, maxCol = 0;
  for (j = 0; j < dim; j++) {
    maxRow = 0,maxCol = 0, maxVal = -1;
    #pragma parallel omp for schedule(static) shared(maxRow, maxVal, maxCol) collapse(2)
      for(i = j; i < m; i++) {
        for(k = j; k < n; k++)
          if (maxVal < *(A + n * i + k)) {
            #pragma omp critical
            if (maxVal < *(A + n * i + k)) {
              maxVal = *(A + n * i + k);
              maxRow = i;
              maxCol = k;
            }
          }
      }

    swap(j, maxRow, maxCol);
  }
}
```

5. Block Column final_A Function

```
void final_A(int dim) {
  int i = 0, j = 0, k = 0;
  int maxVal = -1;
  int maxRow = 0, maxCol = 0;
  for (j = 0; j < dim; j++) {
    maxRow = 0,maxCol = 0, maxVal = -1;
    #pragma parallel omp for schedule(static) shared(maxRow, maxVal, maxCol) collapse(2)
      for(k = j; k < n; k++)
        for(i = j; i < m; i++) {
          if (maxVal < *(A + n * i + k)) {
            #pragma omp critical
```

```
                    if (maxVal < *(A + n * i + k)) {
                        maxVal = *(A + n * i + k);
                        maxRow = i;
                        maxCol = k;
                    }
                }
            }
        swap(j, maxRow, maxCol);
    }
}
```

6. Cyclic Column final_A Function

```
void final_A(int dim) {
    int i = 0, j = 0, k = 0;
    int maxVal = -1;
    int maxRow = 0, maxCol = 0;
    for (j = 0; j < dim; j++) {
        maxRow = 0,maxCol = 0, maxVal = -1;
        #pragma parallel omp for schedule(static, CHUNK_SIZE) shared(maxRow, maxVal, maxCol) collapse(2)
            for(k = j; k < n; k++) {
                for(i = j; i < m; i++) {
                    if (maxVal < *(A + n * i + k)) {
                        #pragma omp critical
                        if (maxVal < *(A + n * i + k)) {
                            maxVal = *(A + n * i + k);
                            maxRow = i;
                            maxCol = k;
                        }
                    }
                }
            }
        swap(j, maxRow, maxCol);
    }
}
```