Matrix Sort Code explanation

Jaime Luengo Rozas : jl3752

March 10, 2019

1 column sort and block sort together

Let's define m as number of columns and n as number of rows.

Before the for loop global index(es) and value variables are initialized that will be shared accross all threads

The outer for loop is executed $min(m, n)$ since:

- if m > n when we have swapped n rows, there will be no more rows to swap; and in the block sort, the submatrix will be empty as well
- if n < m when we have searched m columns, there will not be more columns or submatrixes to search

The first thing in the for loop is an update of the global index and value variable that we just want to be done by a single thread, hence the directive 'single'. This has an implicit barrier (since there isn't 'nowait' directive) so that no thread reaches the critical section without a correct global value to compare with.

Then each thread will get a copy of a local index and value that is initialized to -1 so that it must be updated at least once within the first inner loop.

Now the first inner loop has an static schedule so that the threads get divided equitatively the iterations and also as cache concsious approach. In the $block_sort$ code the 'collapse' directive is used so the nested loop is unwrapped and works similar to the single loop. Finally the 'nowait' is used so that the threads aren't waiting to enter the critical section, since if they waited they would still need to take turns to enter the critical section and would be very inefficient.

The critical section is used so that, even if the variables are shared, there are no race conditions when looking up the global value and updating it. I could have the critical section within the first inner loop and the code would have been correct, but really inefficient.

Afterwards an omp barrier is used so that there is no swapping until all the threads have had the opportunity to update the global index and we are ready for swapping.

Then the swappings use the 'static' schedule for the same reason as the previous. But it is important to note the implicit barrier formed since there isn't 'nowait' directive. This prevents any thread that finishes swapping and goes back up again and reaches the 'single' section and updates the global index, messing up the current swapping.

One of the trickiest race conditions maybe to realize is the one that happens in block_sort if a 'nowait' directive is given to the first swapping loop. This race condition could happen if one thread finished swapping columns first and went to swap row elements and reach the element that is in the intersection of row and column at the same time as the other thread in the previous swapping section. This is avoided with the implicit barrier.

```
void openmp_sort_colum_v2(int n_threads, int m, int n, int *mat[n]){

    int maxRow = n<=m ? n : m;
    int argmax_all, max_total = -1;
    double start_time = omp_get_wtime();
    #pragma omp parallel shared(argmax_all, max_total) num_threads(n_threads)
    {
    for(int i=0; i<maxRow; i++){
      #pragma omp single
      {
        argmax_all = i, max_total = mat[i][i];
      }
      int argmax_local = i, max_local = -1;

      #pragma omp for schedule(static) nowait
      for(int j=i+1;j<n; j++){
        if(max_local < mat[j][i]){
          max_local = mat[j][i];
          argmax_local = j;
        }
      }
      #pragma omp critical
      {
        if(max_total < max_local){
          max_total = max_local;
          argmax_all = argmax_local;
        }
      }

      #pragma omp barrier

      #pragma omp for schedule(static)
      for(int l=0; l<m; l++){
        int temp = mat[i][l];
        mat[i][l] = mat[argmax_all][l];
        mat[argmax_all][l] = temp;
      }

    }
  }
  double time = omp_get_wtime() - start_time;
  printf("%f\n", time*1000);
}
```

Listing 1: Column sort code

```
void omp_sort_block(int n_threads, int m, int n, int *mat[n]){
    int maxRow = n<=m ? n : m;
    double start_time = omp_get_wtime();
    int colmax_total, rowmax_total, max_total;
   #pragma omp parallel shared(colmax_total, rowmax_total, max_total)
   num_threads(n_threads)
    {
    for(int i=0; i<maxRow; i++){
        #pragma omp single
        {
          colmax_total = i, rowmax_total = i, max_total = mat[i][i];
        }
        int colmax_local = i, rowmax_local = i, max_local = -1;
        #pragma omp for schedule(static) collapse(2) nowait
        for(int k = i; k<m; k++){
          for(int j=i; j<n; j++){
            if(max_local < mat[j][k]){
              max_local = mat[j][k];
              colmax_local = k;
              rowmax_local = j;
            }
          }
        }

        #pragma omp critical
        if(max_total < max_local){
          max_total = max_local;
          colmax_total = colmax_local;
          rowmax_total = rowmax_local;
        }

      #pragma omp barrier

        #pragma omp for schedule(static)
      for(int l=0; l<n; l++){
        int temp = mat[l][i];
        mat[l][i] = mat[l][colmax_total];
        mat[l][colmax_total] = temp;
      }
      #pragma omp for schedule(static)
      for(int l=0; l<m; l++){
        int temp = mat[i][l];
        mat[i][l] = mat[rowmax_total][l];
        mat[rowmax_total][l] = temp;
      }
    }
  }
    double time = omp_get_wtime() - start_time;
  printf("%f\n", time*1000);
}
```

Listing 2: BLock sort code