COL380 - Parallel and Distributed Programming : Assignment - 2

Harish Kumar Yadav - 2018CS10337
Param Khakhar - 2018CS10362

05 April 2021

Introduction

The following is a writeup for the Assignment-2, Parallel and Distributed Programming (COL380). There are sections corresponding to each of the four strategies implemented, which also contains the answers to the questions.

Changes to the Base Code

The code provided in the assignment specification was modified in order to achieve better results after using open mp for parallelism. The two versions are as presented below:

```
void crout(double const **A, double **L, double **U, int n) {

  int i, j, k;
  double sum = 0;

  for (i = 0; i < n; i++) {
    U[i][i] = 1;
  }

  for (j = 0; j < n; j++) {
    for (i = j; i < n; i++) {
      sum = 0;
      for (k = 0; k < j; k++) {
        sum = sum + L[i][k] * U[k][j];
      }
      L[i][j] = A[i][j] - sum;
    }

    for (i = j; i < n; i++) {
      sum = 0;
      for(k = 0; k < j; k++) {
        sum = sum + L[j][k] * U[k][i];
      }
      if (L[j][j] == 0) {
        exit(0);
      }
      U[j][i] = (A[j][i] - sum) / L[j][j];
    }
  }
}

void crout3(double **S, double const **A, double **L, double **U, int n)
{
      //S is an nxn auxiliary array initialized with 0's.

      int i, j, k;
      for (i = 0; i < n; i++) {
          U[i][i] = 1;
      }

      for(j = 0; j < n; ++j){
          L[j][j] = A[j][j]-S[j][j];
          U[j][j] = (A[j][j]-S[j][j])/L[j][j];
          double val = L[j][j];
          for(i = j+1; i < n; ++i){
              L[j][i] = A[i][j] - S[i][j];
              U[j][i] = (A[j][i] - S[j][i])/val;
          }
          for(int row1 = j; row1 < n; ++row1){
              for(int row2 = j; row2 < n; ++row2){
                  S[row1][row2] += L[j][row1]*U[j][row2];
              }
          }
      }
  }
```

Following are the salient features for the modified code:

- The indexing in the array is done in consistence with the **row-major** order in order to have maximum cache hits.
- The overall structure of the code is very simple, thus enabling **compiler optimizations** as well.

Strategy-1 : Using Parallel For

The following statistics were observed for our implementation when run on a diagonal matrix of size 2000. We executed the program for 50 iterations and computed the mean, std, and median time for the executions. We found that the median is a better statistic for the measured time as the outliers misrepresented the mean time as well as standard deviation.

# Threads	Median Time in s	Speedup	Efficiency
1	12.498	1.00	1.00
2	7.783	1.60	0.80
4	4.624	2.70	0.675
8	4.68	2.667	0.333
16	5.556	2.249	0.141

Description:

```
int i, j, k;
```

```
#pragma omp parallel for
for (i = 0; i < n; i++) {
  U[i][i] = 1;
}
for(j = 0; j < n; ++j){
      #pragma omp parallel
      {
          L[j][j] = A[j][j]-S[j][j];
          U[j][j] = (A[j][j]-S[j][j])/L[j][j];
          double val = L[j][j];
          #pragma omp for
          for(i = j+1; i < n; ++i){
              L[j][i] = A[i][j] - S[i][j];
              U[j][i] = (A[j][i] - S[j][i])/val;
          }
          #pragma omp for
          for(int row1 = j; row1 < n; ++row1){
              for(int row2 = j; row2 < n; ++row2){
                  S[row1][row2] += L[j][row1]*U[j][row2];
              }
          }
      }
}
```

In this approach we used parallel for all the double nested loops in the function crout3. The outer loop index (j) is the common variable among all the nested for loops. However, no two memory locations are accessed more than once, thereby ruling out the possibility of data race. Since, there are no loop carried dependencies as well, we used #pragma omp for to parallelize them. The new proposed algorithm crout3, requires the nested for loops to be completed in order and therefore we haven't specified the nowait clause in the pragma omp for. Static allocation would be better as the tasks are more or less similar, and using dynamic would also lead to other maintenance overheads.

Correctness:

Strategy-2 : Using Sections

# Threads	Median Time in s	Speedup	Efficiency
1	12.498	1.00	1.00
2	6.665	1.875	0.938
4	5.610	2.228	0.557
8	5.986	2.088	0.261
16	5.847	2.09	0.131

Description:

```
void sections(double** S, double const **A, double **L, double **U, int n, int t

  for (int i = 0; i < n; i++) {
      U[i][i] = 1;
  }

  for(int j = 0; j < n; ++j){
    L[j][j] = A[j][j]-S[j][j];
        double val = L[j][j];

    #pragma omp parallel sections
    {
          #pragma omp section
          {
              for(int i = j+1; i < n; ++i){
                  L[j][i] = A[i][j] - S[i][j];
              }
          }
          #pragma omp section
          {
              for(int i = j; i < n; ++i){
                  U[j][i] = (A[j][i] - S[j][i])/val;
              }
```

```
            }
}
if(t == 2){
  int fir = (j+n)/2;
  #pragma omp parallel sections
  {
  #pragma omp section
  {
      for(int row1= j; row1 < fir; ++row1){
          for(int row2 = j; row2 < n; ++row2){
              S[row1][row2] += L[j][row1]*U[j][row2];
          }
      }
  }
#pragma omp section
{
      for(int row1= fir; row1 < n; ++row1){
          for(int row2 = j; row2 < n; ++row2){
              S[row1][row2] += L[j][row1]*U[j][row2];
          }
      }
}
}
}
    .
    .
    .
    .
```

In this approach, we split the statements of the first nested for loop in crout3, into two different loops which can run concurrently and thus enabling us to use the sections clause specified by OpenMP. For the last triple nested for loop, we again forced parallelism by splitting the iterations into different mutually exclusive ranges. The implementation uses the fact that the number of threads is hard-coded at 2, 4, 8, and 16. The ranges and the updates are mutually exclusive and the addresses accessed are different for all the threads, therefore the condition of data race doesn't arise and doesn't need to be addressed.

Correctness:

Strategy-3 : Using both Parallel For and Sections

# Threads	Median Time in s	Mean Time in s	Std in s
2	20284	523s	
4	277	5.51s	
6	7	1.23s	
8	2	38.4s	

Description:

Correctness:

Strategy-4 : Distributed Version

# Threads	Median Time in s	Mean Time in s	Std in s
2	20284	523s	
4	277	5.51s	
6	7	1.23s	
8	2	38.4s	

Description:

Correctness: