NobleProg

Parallel Programming with OpenMP

Online Training

Final Project

Author:
Dr. Ayaz ul Hassan Khan

March 19, 2022

NobleProg

PARALELEIZING ADI ON SHARED-MEMORY USING OPENMP

The Alternating Direction Integration (ADI) program is a small piece of code with its structure and recurrences represents a scientific iterative algorithm. The C code shown below represents the ADI four loops with its iterative structure:

```
//***************ADI code: main loop start****************************
//Outer Iterater
for(iter = 1; iter <= MAXITER; iter++){
    //////ADI forward & backword sweep along rows//////
    //L1: Starts
    for (i = 0; i < N; i++){
        for (j = 1; j < N; j++){
            x[i][j] = x[i][j]-x[i][j-1]*a[i][j]/b[i][j-1];
            b[i][j]= b[i][j] - a[i][j]*a[i][j]/b[i][j-1];
        }
        x[i][N-1] = x[i][N-1]/b[i][N-1];
    }
    //L2: Starts
    for (i = 0; i < N; i++){
        for (j = N-2; j > 1; j--){
            x[i][j]=(x[i][j]-a[i][j+1]*x[i][j+1])/b[i][j];
        }
    }
    ////// ADI forward & backward sweep along columns//////
    //L3: Starts
    for (j = 0; j < N; j++){
        for (i = 1; i < N; i++){
            x[i][j] = x[i][j]-x[i-1][j]*a[i][j]/b[i-1][j];
            b[i][j]= b[i][j] - a[i][j]*a[i][j]/b[i-1][j];
        }
        x[N-1][j] = x[N-1][j]/b[N-1][j];
    }
    //L4: Starts
    for (j = 0; j < N; j++){
        for (i = N-2; i > 1; i--){
            x[i][j]=(x[i][j]-a[i+1][j]*x[i+1][j])/b[i][j];
        }
    }
}
//***************ADI code: main loop end****************************
```

The objective of this assignment is to write a parallel ADI program for Shared-Memory Systems using OpenMP directives.

The assumptions are following:

- Arrays x, a, and b have been properly initialized and are initially stored in the local memory of the main process with N=1000. Data initialization should not be included in the performance (speedup) calculations.
- The primary parallel program is to be written for one process having 4 threads which are interconnected using shared memory. The number of processes (threads) is then varied from 1 (for getting the sequential execution time), 2 threads, 4 threads, and 8 threads.

Answer each of the following questions:

1. Write the parallel C program for ADI using the most optimized OpenMP directives. The program should target the most efficient (least overhead) implementation and removal any possible overhead. For this determine:
    - how the parallel ADI program should implement the iterative structure to minimize overhead in thread entry and exit
    - how "omp for" should be used in order to minimize the overhead associated with the default barrier in accordance with the data dependence
2. Run the above program while scaling:
    - the parallel machine (number of cores: 1 core for sequential, P=2 cores, 4 cores, and 8 cores) and the problem size (use NxN arrays where N=200, 400, 600, 800, and 1000).
    - Plot the obtained speedup using one single plot where X axis is the problem size, the Y axis is the speedup of P-core over 1 core, and label the three plots with number of cores (S2/1, S4/a, and S8/1). Also give the sequential time for each problem size in the text.
3. One may notice that the OpenMP ADI program does not scale well with increasing the number of cores (threads). For this we may build a partial parallel ADI containing the first two loops ADI12 and another containing the last two loops ADI34. The running under OpenMP shows that ADI12 scales very well while ADI34 does not. The reason is the column major access in the loops 3 and 4. In order to overcome the column-major access which is causing low data reuse in the fetched data blocks because L34 runs over columns and the cache fetches rows which lead to low cashed data reuse. To overcome this problem we may discard the "omp for" from the original OpenMP ADI program and implement L34 as simple "omp parallel" where each range of data as required by the original problem for ADI, but the control flow is changed to access rows of data instead of columns while respecting the recurrence. For this the student is to carry out the following tasks:
    (a) re-write the L34 using "omp parallel", use the thread ID to determine the thread data range, and enforce the control flow (loop) to work as a row-major loop.

   (b) Plot the obtained speedup using one single plot where X axis is the problem size, the Y axis is the speedup of P-core over 1 core, and label the three plots with number of cores (S2/1, S4/a, and S8/1). Also give the sequential time for each problem size in the text. Compare the speedup obtained after eliminating the column major access to the original parallel ADI which has the column major access in its L34.

4. Can you optimize ADI implementation for more speedups using tasks taking into considerations the data dependence in the loop iterations? (Hint: Divide the matrix into square blocks to overlap iterations among L1, L2 and L3/L4)