CS301: Lab - 2

Submission Guidelines:

Doubts to be addressed through google-discussion-board.

Make your report brief and to the point; highlight the 10 important issues listed below (observations + supporting explanations).

Report generation using Using ***www.letshpc.org***

All the assignments have to be supplemented with a **brief write-up or ppt with the following details** (wherever necessary):

1. Context:
    - Brief description of the problem.
    - Complexity of the algorithm (serial).
    - Possible speedup (theoretical).
    - Profiling information (e.g. gprof). Serial time.
    - Optimization strategy.
    - Problems faced in parallelization and possible solutions.
2. Hardware details: CPU model, memory information details, no of cores, compiler, optimization flags if used, precision used.
3. Input parameters. Output. Make sure results from serial and parallel are same.
4. Parallel overhead time. (openmp version on 1 core vs serial without openmp)
5. Problem Size vs Time (Serial, parallel) **curve**. Speedup curve. Observations and comments about the results.
6. If more than one implementation, curves for all algorithms in the same plot.
7. Wherever necessary use log scale and auxiliary units.
8. Problem size vs. speedup curve.
9. No. of cores vs. speedup curve for a couple of problem sizes.
10. Measure performance in MFLOPS/sec.

Assignments include submission of codes [optimized serial and parallel codes (multiple versions if applicable) with necessary comments inside the code].

Basic Vector Operations

Parallelize the following codes using OpenMP:

1. Dot product of two vectors
2. Addition of two vectors

What is the difference in parallelization strategy used for these operations.

Lab 2

1. Write a parallel code (openMP) for print HELLO.
   The goal of thi exercise to understand how parallel programming works.

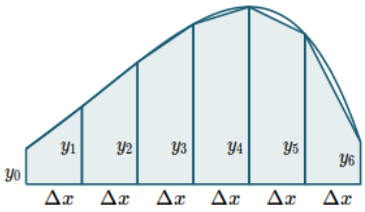
   - write simple c code for print HELLO world
   - Use #pragma omp parallel to make parallel code using <omp.h> library
   - Use threadId = omp_get_thread_num()
   - Print HELLO with threadId

2. Write a parallel code (openMP) for integration using trapezoidal rule. Serial code was discussed in the class.

   - Aim is to find integration of function f(x) , where x∈ [a,b]

     $$\int_b^a f(x)\,dx = \int_b^a f(x)\,dx \quad , f(x) = 5x^2$$

     Intregration is area under the curve. Divide area under the curve



Area under curve using trapezoidal Rule

Area of trapezoidal is = $(y_1-y_0)\Delta x/2$

Area≈ ½ $(y_0+y_1)\Delta x$+ ½$(y_1+y_2)\Delta x$+ ½ $(y_2+y_3)\Delta x$+…

Area≈$\Delta x(y_0/2 + y_1+\ldots+y_{n-1}+y_n/2)$

   - Use omp_get_wtime() for time measurement in openMP.

3. Use the parallel code to calculate PI and verify the implementation.

   - Use f(x) = $\sqrt{4-x^2}$ in above trapezoidal code.

In addition to 10 points discussed above, make comments about your observations and the most optimized implementation.

Start with a naïve implementation, and gradually improve it as discussed during the class (using critical, reduction etc.).

Report about the min. no changes required to convert the serial code to parallel.
Measure performance in MFLOPS/sec.

Code for print ‘HELLO’

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
     int nthreads, threadId;

     /* Fork a team of threads giving them their own copies of
variables */
     #pragma omp parallel private(nthreads, tid)
     {

          /* Obtain thread number */
          threadId = omp_get_thread_num();
          printf("Hello World from thread = %d\n", threadId);

          /* Only master thread does this */
          if (tid == 0)
          {
               nthreads = omp_get_num_threads();
               printf("Number of threads = %d\n", nthreads);
          }

     }  /* All threads join master thread and disband */

}
```

Trepezoidal Code

```
/*****************************************************************/
/****Find area under curve f(x) = 5x*x using Trepezoidal Rule ****************/
#include<stdio.h>
#include<omp.h>   //OpenMp library
#include<math.h>
int main()
{
     int n,i,id;
     float a,b,fa,fb,h,area,x;
     double t;
     double start= omp_get_wtime();
     a=0;
     b=1;
     n=1000000000;
     fa=5*a*a;
     fb=5*b*b;
     h=(b-a)/n;
     area=(fa+fb)/2.0;
     for(i=1; i<n; i++)
     {
          x=a+(i*h);
          area=area+(5*x*x);
     }
     area=area*h;
     double end= omp_get_wtime();
     t=end-start;
     printf("Area is %f and time is %f\n", area,t);
     return 0;
}
```

QR Decomposition using OpenMP

1 Introduction

In linear algebra, QR decomposition is the decomposition of a matrix A into a product of an orthogonal matrix Q and an upper triangular matrix R.

$$\underset{A}{\begin{pmatrix}1 & 2 & 4\\ 0 & 0 & 5\\ 0 & 3 & 6\end{pmatrix}} = \underset{Q}{\begin{pmatrix}1 & 0 & 0\\ 0 & 0 & 1\\ 0 & 1 & 0\end{pmatrix}} \underset{R}{\begin{pmatrix}1 & 2 & 4\\ 0 & 3 & 6\\ 0 & 0 & 5\end{pmatrix}}$$

QR decomposition is widely used in various other algorithms:

- Efficiently solve systems of linear equations.
- Used to solve the linear least squares problem.
- Basis for eigenvalue algorithm.
- Automatic removal of an object from an image.

There are several methods for computing the QR decomposition, such as Gram–Schmidt process, Householder transformations and Givens rotations. In this assignment, we will be using the Gram–Schmidt process to find the QR decomposition.

2 Implementation

You will start with a $n \times n$ 2-D matrix with real entries, whose columns represent a vector of dimension n. In the i^{th} iteration of the Gram-Schmidt algorithm, i^{th} vector is normalized to unit length, and the subsequent vectors (*i* + 1 to *n*) are updated in that iteration by subtracting the multiple of the i^{th} vector, so as to obtain the orthogonal matrix. The entries of the upper triangular matrix, r_{ij} is the projection of the j^{th} vector on i^{th} vector.

$$r_{ij} = \frac{a_i^T a_j}{\sqrt{a_i^T a_i}}, \quad j = i, \ldots, n \tag{1}$$

$$a_j \leftarrow a_j - r_{ij} a_i, \quad j = i+1, \ldots, n \tag{2}$$

$$a_i \leftarrow a_i / r_{ii} \tag{3}$$

Here a_i is the column i of matrix *A*, r_{ij} are the elements of the upper triangular matrix *R*. Matrix *A* obtained at the end of the algorithm is the orthogonal matrix *Q*.

2.1 Pseudo Code

Algorithm Gram-Schmidt process

1: **for** $i \leftarrow 1$ **to** n **do**
2: $s \leftarrow 0$
3: **for** $j \leftarrow 1$ **to** n **do**
4: $s \leftarrow s + a_{ji}^2$
5: **end for**
6: $r_{ii} \leftarrow sqrt(s)$
7: **for** $j \leftarrow 1$ **to** n **do**
8: $q_{ji} \leftarrow a_{ji}/r_{kk}$
9: **end for**
10: **for** $j \leftarrow i + 1$ **to** n **do**
11: $s \leftarrow 0$
12: **for** $k \leftarrow 1$ **to** n **do**
13: $s \leftarrow s + a_{kj} * q_{ki}$
14: **end for**
15: $r_{ji} \leftarrow s$
16: **for** $k \leftarrow 1$ **to** n **do**
17: $a_{kj} \leftarrow a_{kj} - r_{ij} * q_{ki}$
18: **end for**
19: **end for**
20: **end for**

- Using the above pseudo code, implement the serial QR Decomposition Algorithm in C.
- Vary the value of n as $64, 128, 256, 512, 1024$ and 2048 and measure the runtime of the code.
- Time complexity of the algorithm is $\mathcal{O}(n^3)$. Thus if we double the value of n, the runtime must become eight times. Does your practical observations corroborate the above theoretical calculation? Why or Why not?
- Without parallelizing the code, try to reduce the runtime of the code. (*Hint: Optimize cache performance by improving the locality*)

3 Parallelization

Using OpenMP library, parallelize your code for shared memory systems.

- Find the dependency present in the algorithm. Can you remove it?
- What are the problem division strategies that you can use? Also find its granularity level. Which of these strategies will give the best performance and why? (*Hint: Once again try to maintain locality to optimize cache performance*)
- For all the above strategies listed, calculate the theoretical compute to memory access ratio.
- Does your parallel implementation take care of load balancing? Try different scheduling mechanisms to achieve ideal load balance.
- As done above in the serial implementaion, vary the value of of n as $64, 128, 256, 512, 1024$ and 2048 and measure the runtime of the code. Plot the graph of this speedup obtained and comment on the trend observed.