CITS3402, CITS5507 - First Project

Notes:

- This project is worth 15 marks.
- The due date is 11:59 pm on September 13.
- The project can be done either individually, or in a group of two.
- The submission is through `cssubmit` and will consist of a code file and a report file (only in pdf format). Both files must have the names and student numbers of group members.

1 Matrix-matrix multiplication of sparse matrices

I have explained these multiplications in the lecture, you should read this Wikipedia article if you do not know. I have also explained sparse matrices, you can also read this Wikipedia page. This project is about sparse matrix-matrix multiplication. A very small percentage of the entries of a matrix have non-zero elements in a sparse matrix. Sparse matrices are usually stored and processed in a compressed form and there are many different compression algorithms, you can read about a simple compression algorithm called *Yale format* from the article linked above. We will use a still simpler compression format that we call *row compression*, the idea is to store the column indices of the non-zero elements in each row. For example, the following matrix: $A = \begin{pmatrix} 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$ will be represented by two matrices $B = \begin{pmatrix} 2 & 1 \\ 3 & \\ 1 & \\ 1 & \end{pmatrix}$, and $C = \begin{pmatrix} 0 & 3 \\ 3 & \\ 1 & \\ 2 & \end{pmatrix}$. The B matrix stores the actual elements in each row, and the C matrix stores the indices in each row where these elements are located. All other elements are assumed to be 0. If there is no non-zero element in a row, the corresponding rows of the B and C matrices will store two consecutive

0s. For example, for the matrix $A = \begin{pmatrix} 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$, $B = \begin{pmatrix} 2 & 1 \\ 3 & \\ 0 & 0 \\ 1 & \end{pmatrix}$, and $C = \begin{pmatrix} 0 & 3 \\ 3 & \\ 0 & 0 \\ 2 & \end{pmatrix}$.

2 Description

- You have to generate three pairs of sparse matrices X and Y of size 100000×100000 when an entry of the matrix can be non-zero with probabilities 0.01, 0.02 and 0.05. For example, in the first case you can generate a random number between 1 and 100 for each entry of the matrix, and the entry is non-zero if the random number is between 0 and 1. Generate another random small integer between 1 and 10 and store it as the entry in that location of the matrix.

- Create the two arrays B and C for row compression for each matrix as in the example above. For example, if you are multiplying two matrices X and Y of size 100000×100000 each, you should generate two matrices XB, XC (for X) and YB and YC (for Y) and use these four matrices for multiplying X and Y.

3 Tasks

1. Write an ordinary matrix multiplication algorithm so that we can check your results. The number of rows, number of columns should be declared at the top of your code using `#define` directives.

2. Write separate code for generating the B and C matrices given a matrix X. You can generate the matrix X inside this code.

3. Multiply the three sets of matrices mentioned above, and evaluate their running times. You have to write OpenMP code and vary the number of threads. Though Setonix has 28 cores, you can use more threads

than 28. You have to find the number of threads that gives the best performance for each of the three cases.

4. Write a report by explaining your implementation. Use appropriate graphs and bar charts.

4 Allocation of marks

- 2 marks for ordinary matrix multiplication algorithm.
- 4 marks for correctly generating the B and C matrices.
- 6 marks for performance evaluation. This part should include experiments for determining the optimal number of threads for the best performance for the three matrix sizes. Also, you should include an analysis of performance using the four scheduling strategies for scheduling `for` loops for the matrix multiplication problem for one the three matrix sizes. Your code should write the final results for the B and C matrices in two separate files named `FileB` and `FileC`. Also you should allocate large matrices using dynamic allocation (we have discussed this in the lecture). The way to write to a file is

  ```
  FILE *fp, *fopen();
  fp=fopen("FileB","w"); /*opens the file named FileB for writing*/
  fprintf(fp,"%d", i); /* write variable i to the file*/
  ```

- 3 marks for quality of report.

Amitava Datta
August 2024