

Final project performance analysis

Luis Francisco Flores Romero A01328937

Daniela Alvarado Pereda A01329233

06/05/2019

Programación Avanzada

ITESM Campus Puebla

**Final project performance analysis**

The program calculates the inverse of a square matrix using the following formula.

The type matrix\_t was defined to better handle matrices as a type. The following functions were created in order to perform the calculation.

* getInverseMatrix
* getDeterminant: used to obtain inverse matrix
* getAdjugateMatrix: used to obtain inverse matrix
* getCofactorMatrix: used to obtain adjugate matrix
* getCofactor: used to obtain cofactor matrix
* getMinor: used to obtain cofactor
* transpose: used to obtain adjugate matrix

The following helper functions were also included for testing purposes and to handle common type operations.

* getValue
* setValue
* printSerializedMatrix
* printMatrix
* destroyMatrix
* newIdentity
* newSquareMatrix
* newDummyMatrix
* import
* newMatrix

In order to make implementation easier, all these functions where included in the static library “matrices”. Several operations included in the library can be parallelized without producing incorrect results, these operations are listed bellow.

* Transposing a matrix: each element can be repositioned independently.
* Obtaining the cofactor matrix: the cofactor of each element can be obtained independently from the others.
* Obtaining the inverse of a matrix: each element of the adjugate matrix can by divided by the determinant independently.
* Other operations related to setting values of a matrix.

The use of OpenMP directives causes overhead related to thread handling. Because of this, some of the operations listed above are not well suited for parallelization, they are too lightweight and would cause the program to take longer because of the overhead.

After excluding these operations, OpenMP directives where added before the following loops inside the functions related to obtaining the cofactor matrix and the inverse of a matrix. These directives provide parallel computing for the operations.

1. #pragma omp parallel for collapse(2)
2. for(int i = 0; i < rows; i++) {
3. for(int j = 0; j < cols; j++) {
4. setValue(comatrix, i, j, getCofactor(m\_ptr, i, j));
5. }
6. }
7. #pragma omp parallel for collapse(2)
8. for(int i = 0; i < rows; i++) {
9. for(int j = 0; j < cols; j++) {
10. float element = getValue(adjugate, i, j) / determinant;
11. setValue(inverse, i, j, element);
12. }
13. }

The following tables compare execution times for the operations with and without the use of OpenMP directives.

|  |  |  |
| --- | --- | --- |
| Parallel cofactor matrix | With parallel inverse | Without parallel inverse |
|  | 13.85 s | 14.16 s |
|  | 13.72 s | 14.57 s |
|  | 13.80 s | 14.71 s |
|  | 13.89 s | 14.84 s |
|  | 13.74 s | 14.27 s |
| **Average** | **13.8 s** | **14.51 s** |

**Table 1.** Running time for computing the inverse of a

10x10 matrix. Use of parallel programming in the

inverse function enhances average performance.

|  |  |  |
| --- | --- | --- |
| Parallel inverse, parallel cofactor matrix | With parallel transpose | Without parallel transpose |
|  | 13.88 | 13.85 |
|  | 14.34 | 13.72 |
|  | 15.10 | 13.80 |
|  | 14.84 | 13.89 |
|  | 13.97 | 13.74 |
| **Average** | **14.426 s** | **13.8 s** |

**Table 2.** Running time for computing the inverse of a

10x10 matrix. The use of parallel programming in

the transpose function does not improve performance.

As seen in the tables above, the use of parallel programming does not always improve performance. For the function listed in Table 2, transpose, the operations performed within the function are too lightweight for it to take advantage in the use of several threads. Thus, the overhead related to the creation of the threads takes more time than the operation would without using parallelism.

However, including the OMP directives for functions *getCofactorMatrix* and *getInverse* do enhance performance, and such a thing was expected. These functions have not been chosen arbitrarily, but rather according to their suitability for parallelism. Key factor considered include having a non-trivial work load to be distributed, data independency, and few potential syncing conflicts. The operations that need to be performed in these functions are more complex and can be independently solved in separated threads, without need to wait for intermediate data created on other threads. This distribution of load allows for shorter time in performing the inverse operation.