Analyzing Matrix inversion algorithm

* Algorithm:

For calculate matrix inversion we use LUP decomposition method which describe as bellow:

For given matrix A:

LU factorization with partial pivoting as:

***PA = LU***

***L*** and ***U*** are lower and upper triangular matrices. unique factorization for matrix ***A***

require the lower triangular matrix *L* to be a unit triangular matrix.

***P*** is a permutation matrix which reorders the rows of ***A***.

Then for calculating matrix invers we solve bellow expression in defined manner as bellow:

***PA = LU => AA-1 = LU A-1 = PI:***

We Iteratively move over columns of ***I*** as ***b*** and solve equations:

1. First, we solve the equation ***Ly = Pb*** for y.
2. Second, we solve the equation ***Ux = y*** for x.

* Implementation:

Our code has two main methods:

*static int LUPdecompose(int size, Type A[MAX][MAX], int P[MAX]);*

which return LU matrix in A and permutation matrix in P.

*static int LUPinverse(int size, int P[MAX], Type LU[MAX][MAX],*

*Type B[MAX][MAX], Type X[MAX], Type Y[MAX]);*

which return invers of matrix in A in LU.

* Compiling:

Space complexity of algorithm is ***O(n2)*** which for large size of ***n*** may cause problem due to default stack size per application as ***2MB*** in my OS and compiler base config.

Because of that I preserve more space for stack size to prevent segment fault of code that cause sudden execution termination at the start of running.

|  |
| --- |
| gcc -Wl,--stack,4000000 -Wall -pg lup\_matrix\_inverse.c -o2 -o int\_500\_out.exe |

-Wl,option: pass option as an option to the linker.

--stack, <size>: where <size> is in bytes to set the stack size.

-o: specify output exe file name

-o2: optimizing more

-pg: Generate extra code to write profile information suitable use gprof.

* System Information’s:

CPU: core i5 8th generation

RAM: 8GB

OS: windows 10

* Performance profiling with Gprah:

|  |
| --- |
| gprof int\_500\_out.exe > int\_500\_profile-data.txt |

(500 \* 500) Matrix

int data type -> ***4MB*** stack size

float data type -> ***4MB*** stack size

double data type -> ***6MB*** stack size

(1000 \* 1000) Matrix

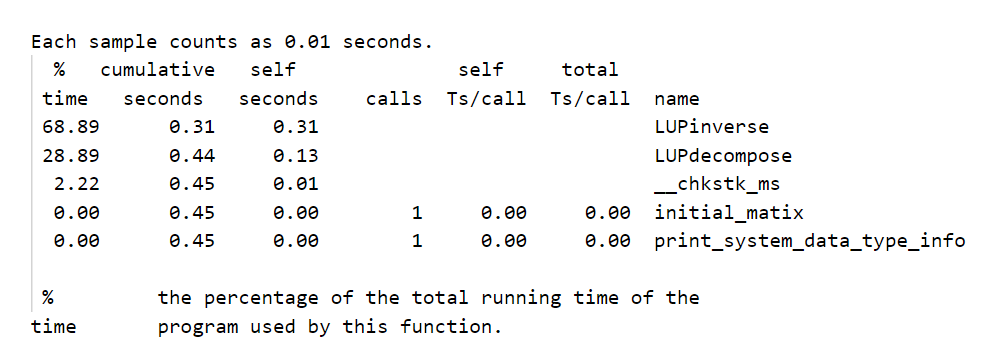
int data type -> ***12MB*** stack size

float data type -> ***12MB*** stack size

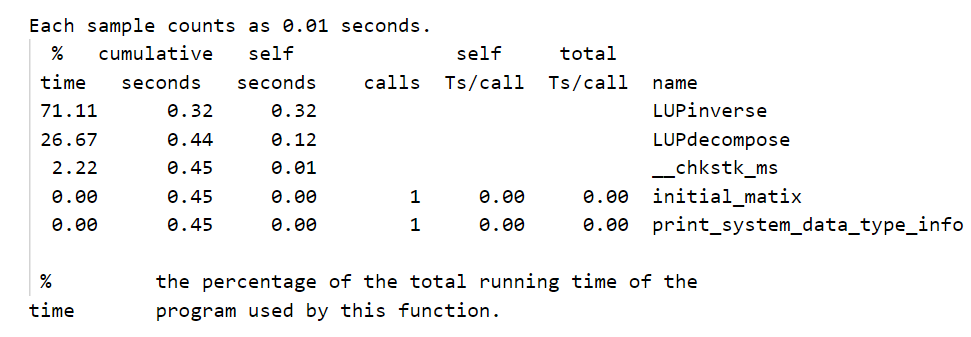
double data type -> ***26MB*** stack size

(500 \* 500) Matrix

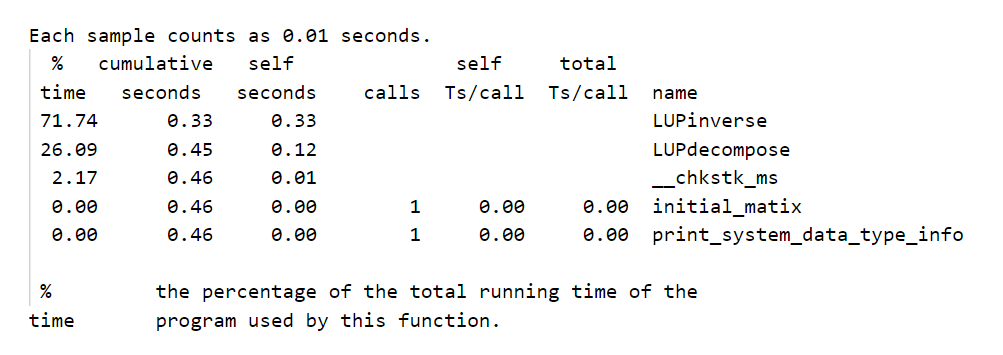
**int data type:**



**float data type:**

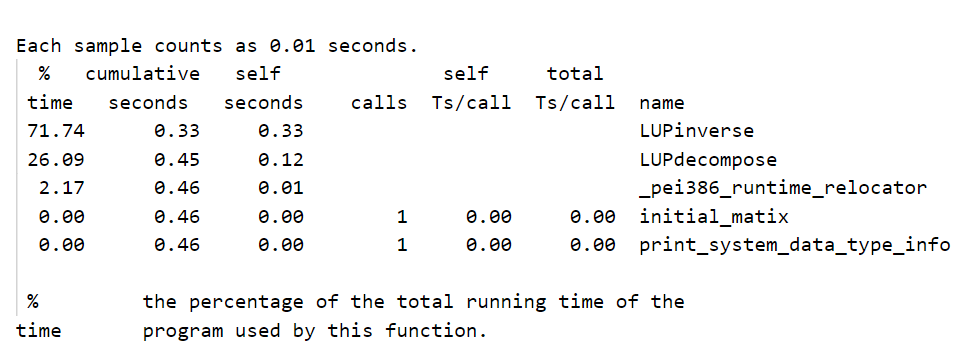


**double data type:**

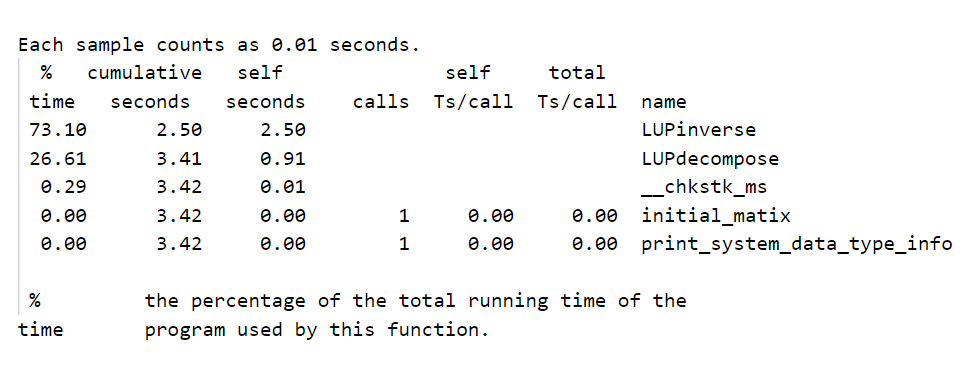


(1000 \* 1000) Matrix

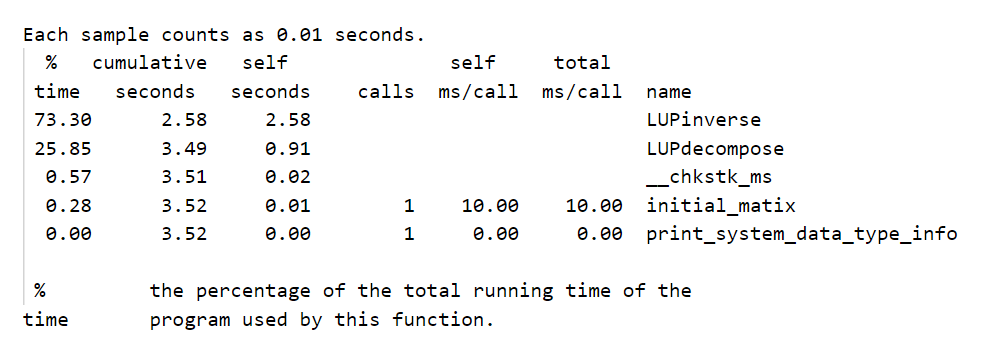
**int data type:**

****

**float data type:**



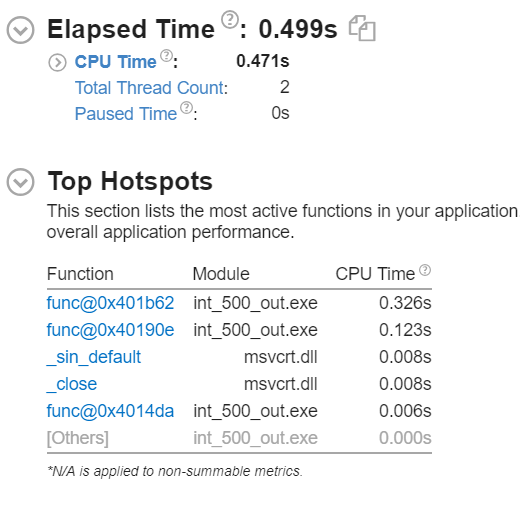
**double data type:**



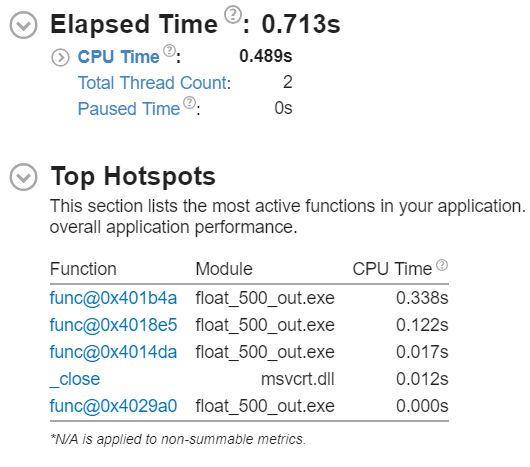
* Performance profiling with Vtune:

(1000 \* 1000) Matrix

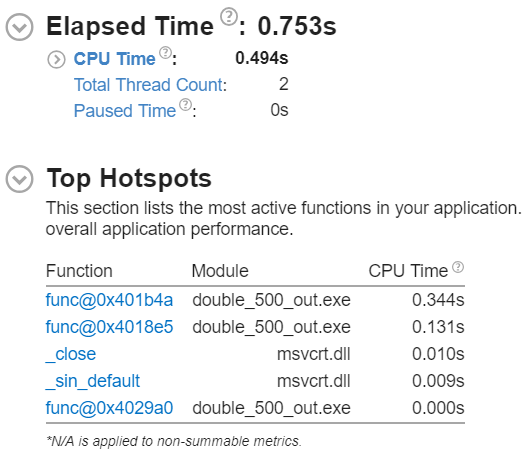
**int data type:**



**float data type:**



**double data type:**



* Execution analysis:

Program has two main method as:

* ***LUPdecompose:***

With time complexity as O(n2) based on code reviewing and space complexity as O(n+n2) =O(n2).

But according to the algorithm documents, LU decomposition can be computed in time O(M(n)). M(n) ≥ na where a > 2. It means O (n2.376)

* ***LUPinverse:***

With time complexity as O(n3) and space complexity as O(3n+2n2) =O(n2)

And according to code result On the used machine, size (in bytes) and precision (in number of decimal digits) of

float: 4 and 6,

double: 8 and 15

* Proposed improvement:

As the result shows ***LUPinverse*** takes the most of execution time.

This method solving mathematical equationiteratively over each column.

We can divide this work over multi threads that independently solve equation for specific column vector and in this way make the code much faster.

For another method ***LUPdecompose*** that take O(n2) time we can split the outer loop in specific sizes and pass them to some thread and make it faster.