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 $L_y = P_b$ for y.
2. Second, we solve the equation $U_x = y$ 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 -o0 -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

`-o0`: without optimizing

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

- ## System Information's:

CPU: core i5 8th generation

RAM: 8GB

OS: windows 10

Cache: 1L = 256KB, 2L = 1MB, 3L = 6MB

- ## 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 -> ***7MB*** stack size

(1000 * 1000) Matrix

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

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

double data type -> ***28MB*** stack size

(500 * 500) Matrix

int data type:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 68.18      0.30     0.30                             LUPinverse
 27.27      0.42     0.12                             LUPdecompose
  2.27      0.43     0.01        1    10.00    10.00  initial_matix
  2.27      0.44     0.01                             __chkstk_ms
  0.00      0.44     0.00        1     0.00     0.00  print_system_data_type_info
```

float data type:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 71.11      0.32     0.32                             LUPinverse
 26.67      0.44     0.12                             LUPdecompose
  2.22      0.45     0.01                             __chkstk_ms
  0.00      0.45     0.00        1     0.00     0.00  initial_matix
  0.00      0.45     0.00        1     0.00     0.00  print_system_data_type_info
```

double data type:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 71.74      0.33     0.33                             LUPinverse
 26.09      0.45     0.12                             LUPdecompose
  2.17      0.46     0.01                             __chkstk_ms
  0.00      0.46     0.00        1     0.00     0.00  initial_matix
  0.00      0.46     0.00        1     0.00     0.00  print_system_data_type_info
```

(1000 * 1000) Matrix

float data type:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 72.30      2.48     2.48                             LUPinverse
 27.41      3.42     0.94                             LUPdecompose
  0.29      3.43     0.01                             __chkstk_ms
  0.00      3.43     0.00        1     0.00     0.00  initial_matix
  0.00      3.43     0.00        1     0.00     0.00  print_system_data_type_info
```

double data type:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 73.65      2.60     2.60                             LUPinverse
 25.50      3.50     0.90                             LUPdecompose
  0.57      3.52     0.02                             __chkstk_ms
  0.28      3.53     0.01        1    10.00    10.00  initial_matix
  0.00      3.53     0.00        1     0.00     0.00  print_system_data_type_info
```

- Performance profiling with Vtune:

The first function address always is = ***LUPinverse***

The second function address always is = ***LUPdecompose***

(500 * 500) Matrix

int data type:

Elapsed Time: 0.496s
- CPU Time: 0.470s
  - Total Thread Count: 2
  - Paused Time: 0s

Top Hotspots

This section lists the most active functions in your application. overall application performance.

Function	Module	CPU Time
func@0x401b6e	int_500_out.exe	0.335s
func@0x40190e	int_500_out.exe	0.124s
_math_exit	msvcrt.dll	0.011s
func@0x4029e0	int_500_out.exe	0.000s

*N/A is applied to non-summable metrics.

float data type:

Elapsed Time: 0.518s
- CPU Time: 0.485s
  - Total Thread Count: 2
  - Paused Time: 0s

Top Hotspots

This section lists the most active functions in your application. overall application performance.

Function	Module	CPU Time
func@0x401b44	float_500_out.exe	0.344s
func@0x4018d3	float_500_out.exe	0.123s
_math_exit	msvcrt.dll	0.008s
_close	msvcrt.dll	0.006s
_sin_default	msvcrt.dll	0.005s
[Others]	float_500_out.exe	0.000s

*N/A is applied to non-summable metrics.

double data type:

Elapsed Time: 0.832s
- CPU Time: 0.493s
  - Total Thread Count: 2
  - Paused Time: 0s

Top Hotspots

This section lists the most active functions in your application. overall application performance.

Function	Module	CPU Time
func@0x401b56	double_500_out.exe	0.342s
func@0x4018e5	double_500_out.exe	0.131s
_sin_default	msvcrt.dll	0.012s
_close	msvcrt.dll	0.008s
func@0x4029c0	double_500_out.exe	0.000s

*N/A is applied to non-summable metrics.

(1000 * 1000) Matrix

float data type:

Elapsed Time: 3.852s

CPU Time:	3.772s
Total Thread Count:	2
Paused Time:	0s

Top Hotspots

This section lists the most active functions in your application. typically results in improving overall application performance.

Function	Module	CPU Time
func@0x401b5c	float_1000_out.exe	2.712s
func@0x4018eb	float_1000_out.exe	0.974s
_sin_default	msvcrt.dll	0.034s
_math_exit	msvcrt.dll	0.020s
printf	msvcrt.dll	0.010s
[Others]		0.022s

*N/A is applied to non-summable metrics.

double data type:

Elapsed Time: 15.370s

CPU Time:	3.953s
Total Thread Count:	2
Paused Time:	0s

Top Hotspots

This section lists the most active functions in your application. typically results in improving overall application performance.

Function	Module	CPU Time
func@0x401b56	double_1000_out.exe	2.804s
func@0x4018e5	double_1000_out.exe	1.065s
_math_exit	msvcrt.dll	0.027s
func@0x4014da	double_1000_out.exe	0.019s
_sin_default	msvcrt.dll	0.019s
[Others]		0.019s

*N/A is applied to non-summable metrics.

- ## 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)). $M(n) \geq n^a$ 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 matrix size and comparison accuracy increases the execution times growth.

Matrix space always is a good application for parallelism due to high ILP property of matrix-based problems.

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

This method solving mathematical equation iteratively over each column.

We can divide this work over multi thread tasks 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.in other word the partial pivoting process that is comparison-based process across each column could be done in parallel manner.