Analyzing Matrix inversion algorithm(heap mem mode)

- ## 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, int *P);
```

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

```
static int LUPinverse(int size, int *P, Type** LU, Type **B, Type *X, Type *Y);
```

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 -Wall -pg lup_matrix_inverse.c -o0 -o int_500_out.exe
```

`-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 8^{th} 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:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 63.41      0.26     0.26                             LUPinverse
 36.59      0.41     0.15                             LUPdecompose
  0.00      0.41     0.00        4     0.00     0.00  allocate_2d
  0.00      0.41     0.00        1     0.00     0.00  initial_matix
```

float data type:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 68.09      0.32     0.32                             LUPinverse
 31.91      0.47     0.15                             LUPdecompose
  0.00      0.47     0.00        4     0.00     0.00  allocate_2d
  0.00      0.47     0.00        1     0.00     0.00  initial_matix
```

double data type:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 70.00      0.35     0.35                             LUPinverse
 30.00      0.50     0.15                             LUPdecompose
  0.00      0.50     0.00        4     0.00     0.00  allocate_2d
  0.00      0.50     0.00        1     0.00     0.00  initial_matix
```

(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
 68.83      2.54     2.54                             LUPinverse
 31.17      3.69     1.15                             LUPdecompose
  0.00      3.69     0.00        4     0.00     0.00  allocate_2d
  0.00      3.69     0.00        1     0.00     0.00  initial_matix
```

double data type:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 69.05      2.61     2.61                             LUPinverse
 30.69      3.77     1.16                             LUPdecompose
  0.26      3.78     0.01        1    10.00    10.00  initial_matix
  0.00      3.78     0.00        4     0.00     0.00  allocate_2d
```

- Performance profiling with Vtune:

1st function address always is ***LUPinverse*** , 2nd function address always is ***LUPdecompose***

(500 * 500) Matrix

int data type:

Elapsed Time: 0.475s

CPU Time:	0.455s
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@0x401d5d	int_500_out.exe	0.288s
func@0x401aaa	int_500_out.exe	0.148s
_sin_default	msvcrt.dll	0.010s
_open	msvcrt.dll	0.009s

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

float data type:

Elapsed Time: 0.539s

CPU Time:	0.515s
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@0x401d39	float_500_out.exe	0.347s
func@0x401a75	float_500_out.exe	0.168s

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

double data type:

Elapsed Time: 0.546s

CPU Time:	0.525s
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@0x401d41	double_500_out.exe	0.357s
func@0x401a7d	double_500_out.exe	0.157s
_sin_default	msvcrt.dll	0.011s

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

(1000 * 1000) Matrix

float data type:

Elapsed Time: 4.141s

- CPU Time: 4.097s
- 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@0x401d39	float_1000_out.exe	2.781s
func@0x401a75	float_1000_out.exe	1.250s
_sin_default	msvcrt.dll	0.029s
_math_exit	msvcrt.dll	0.020s
_87except	msvcrt.dll	0.010s
[Others]	msvcrt.dll	0.007s

N/A is applied to non-summable metrics.

double data type:

Elapsed Time: 4.229s

- CPU Time: 4.189s
- 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@0x401d41	double_1000_out.exe	2.837s
func@0x401a7d	double_1000_out.exe	1.279s
_math_exit	msvcrt.dll	0.039s
_sin_default	msvcrt.dll	0.021s
Sleep	KernelBase.dll	0.011s
[Others]		0.002s

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.