LU Factorisation with Partial Pivoting

Filipe Oliveira* and Sérgio Caldas*
University of Minho
Computer Science Department
*Email: *a57816@alunos.uminho.pt, †a57779@alunos.uminho.pt*

***Abstract*—In order to solve a linear system, the Gaussian eliminations with partial pivoting is one of the most efficient and accurate ways to do it.**

In the absence of pivoting, the total error found on the final solution is sometimes undesired. A row-oriented implementation of Gaussian elimination with partial pivoting on a local-memory multiprocessor is described, and also a blocked row-oriented implementation of Gaussian elimination with partial pivoting.

A simple error and total time to solution analysis is made, providing the comparison between LU factorisation with and without partial pivoting, and also the comparation between blocked and non-blocked versions.

***Index Terms*—Gaussian elimination; Linear Algebra; multiprocessor; LU Factorisation[1] ; Linear Systems Pivoting**

1. Introduction

1.1. LU Factorisation without Partial Pivoting

The LU Factorisation[2] is a method used to solve a linear system of equations. This factorisation uses two matrices, one upper matrix (triangular superior matrix) and a lower matrix (triangular inferior matrix). The LU Factorisation can be viewed as the matrix form of Gaussian elimination. A math definition for LU factorisation is:

> Let A be a square matrix. An LU factorisation refers to the factorisation of A, with proper row and/or column orderings or permutations, into two factors, a lower triangular matrix L and an upper triangular matrix U.

The equation that translate this definition is:

$$A = LU \tag{1}$$

In the lower matrix, all elements above the diagonal are zeros, in the upper matrix all elements below the diagonal are zeros and in the upper matrix all elements of diagonal are 1.

1. Lecture 12 LU Decomposition
2. LU Decomposition

1.2. LU Factorisation with Partial Pivoting

In the LU Factorisation with Partial Pivoting, the factorisation refers often to the LU factorisation with row permutations only. The LU Factorisation can be defined for the following equation:

$$PA = LU \tag{2}$$

Where L and U, are the lower and the upper matrix respectively and P, refers to a permutation matrix. The permutation matrix, is a binary matrix where in each row and column only have one entry of 1, the others elements are zeros. When we multiply permutation matrix for other matrix, this produce the permutation in the rows or columns of the other matrix.

2. Hardware characterisation

The computing platform, our team laptop, is a Intel® Core™ i7-3635QM (Ivy Bridge architecture). The system features 16 GB of DDR3 RAM, supported at a frequency of 1600MHz, divided in 2 memory channels.

The characteristics of the CPU on the computing platform is presented in the table 1.

System	team laptop
# CPUs	1
CPU	Intel® Core™ i7-3635QM
Architecture	Ivy Bridge
# Cores per CPU	4
# Threads per CPU	8
Clock Freq.	2.4 GHz
L1 Cache	128KB 32KB per core
L2 Cache	1024KB 256KB per core
L3 Cache	6144KB shared
Inst. Set Ext.	SSE4.2 & AVX
#Memory Channels	2
Vendors Announced Peak Memory BW	25.6 GB/s
Measured Peak Memory BW	15.5 GB/s

Table 1: Architectural characteristics of the evaluation platform.

3. Changes made in the provided code

The first step of this work, was analysing the provided code by the professor, and understand it. After that, we changed the code in order to allow rows permutations (this changes were made in the ”BLAS2LU.m” file).

In order to allow row permutations in the blocked LU factorisation with partial pivoting, changes were made in the file ”BLAS3LU.m” producing a new code store in the file ”BLAS3LUPP.m”. This blocked version, in order to allow row permutation recurs to the prior defined ”BLAS2LUPP”. To reflect row permutations to all A matrix, the returned P (Permutation matrix) from the algorithm BLAS2LUPP was applied to the corresponding rows of the entire A matrix. An alternative solution would be to instead of passing a full Permutation matrix from BLAS2LUPP to pass only a permutation Array. The main difference lies on memory locality of the test machine and the number of operations required to achieve the solution – with P Matrix we have a matrix to matrix multiplication, with P Array we have only N operations, being N the P Array dimension.

In order to be able to measure time and error for the several algorithms, changes were made to the return parameters of each function.

Every function, additionally to the returned A L U P, returns also vars **time** and **error** .

4. Error calculation for the final matrices

In order to calculate the error of the final solution, the following formula was used:

$$\frac{\| A - L * U \|}{\| A \|}$$

5. Analysing the solutions validity and error percentage

Several auxiliar scripts were produced in order to relate both error and time for solution for the several matrix sizes and block dimensions. The corresponding error analyse results are shown in figures 1 and 2.

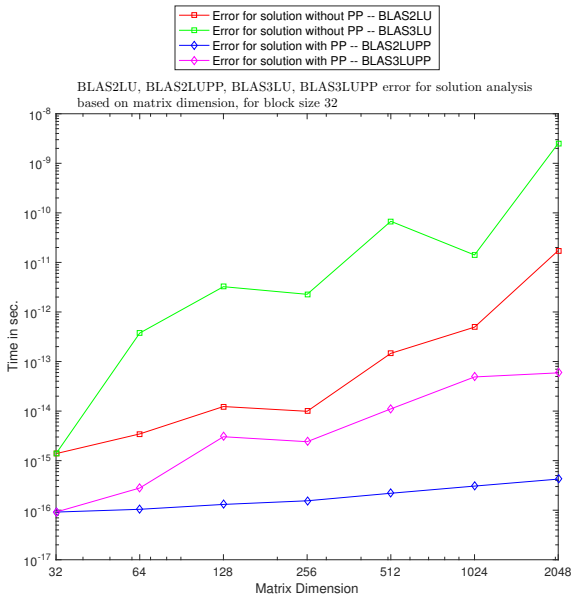

Figure 1: Error for solution analysis based on matrix dimension, for block size 32

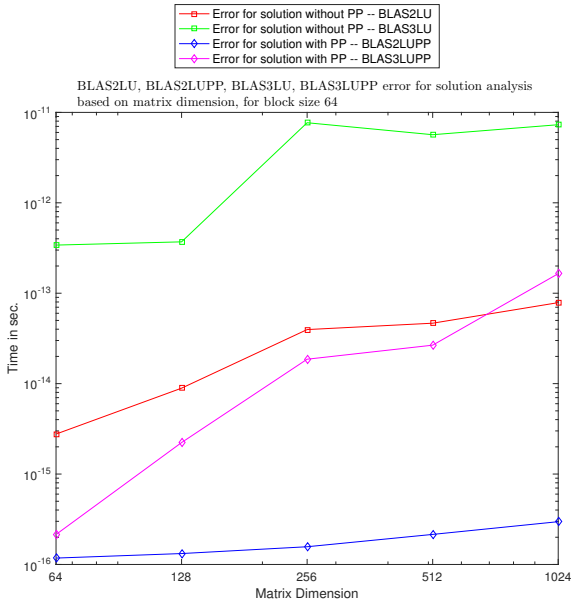

Figure 2: Error for solution analysis based on matrix dimension, for block size 64

As predicted, the error value for algorithms that recur to Partial Pivot presents a value several orders of magnitude below the one presented by algorithms without Partial pivoting.

6. Analysing the time for solutions based on algorithm and block dimension

Proved the algorithm correctness, we should analyse the total time for solution based on both the algorithm and the block dimension.

Please denote that the block sizes were chosen based on the average CPU cache line size.

The corresponding time analyse is shown in figures 3 and 4. A relation between matrix block dimension and time for solution is presented on figure 5.

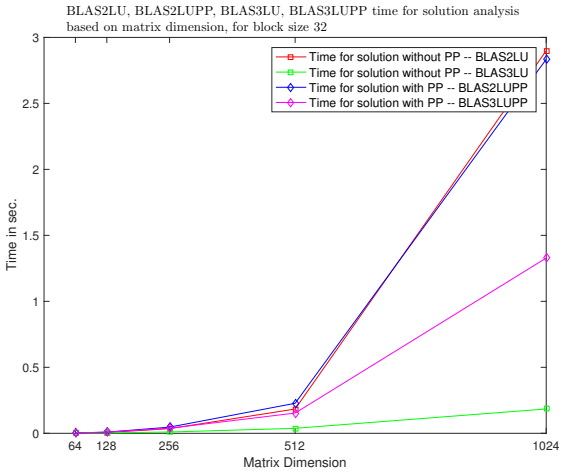

Figure 3: Time for solution analysis based on matrix dimension, for block size 32

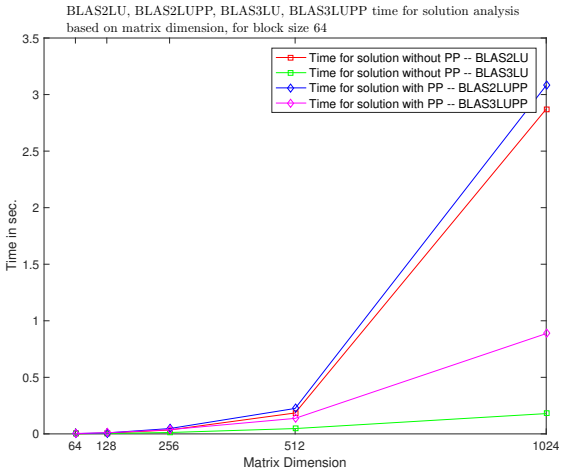

Figure 4: Time for solution analysis based on matrix dimension, for block size 64

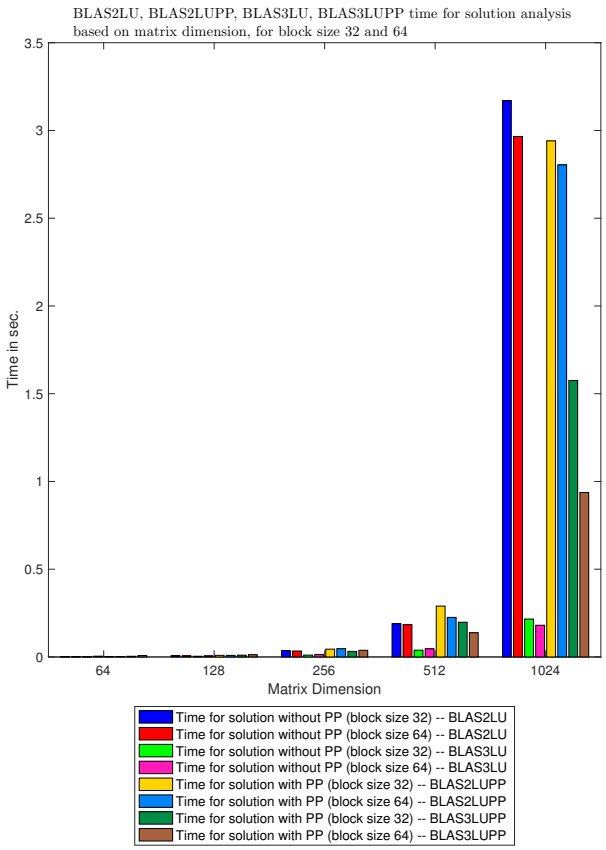

Figure 5: Time for solution analysis based on matrix dimension, for block size 32

As we can visually infer, a block dimension of 32 produces better overall results in terms of time for solution. That was as expected result, since the computing platform used in the tests has as Cache line size of 32 Bytes.

7. Conclusion

As we can see, in the previous results, the relative error associated to the LU Factorisation with partial pivoting is less then the LU Factorisation without partial pivoting, because of this we can conclude that the LU Factorisation with partial pivoting is more accurate then FU Factorisation without partial pivoting.

In terms of computation time the most expensive algorithms are the ones with partial pivoting. The total time for solution LU Factorisation with partial pivoting for BLAS2 has revealed to be the most expensive for block sizes of 32 and 64.

As predicted, the usage of blocks in the algorithms, implied a better time for solution.

During this work we have come across with some problems, essentially on the implementation of the BLAS3

LU factorisation with partial pivoting. The bigger problem was the row permutations on this algorithm. We think that this difficulties were surpassed.