Analysis and Implementation of Parallel LU-Decomposition with Different Data Layouts

E. E. Santos M. Muraleetharan

June 2000

Abstract

In this paper we will analysis and implement the parallel LU-decomposition method for six different data layouts – column block, row block, column cyclic, row cyclic, blocked grid, and scattered grid. We use the LogP model to analysis the running time of algorithms since it's not depend on the structure of the network and implement the algorithms using MPI.

1 Introduction

Solving a set of simultaneous linear equations is a fundamental problem that occures in diverse applications and is of central importance in numerical analysis. A linear system can be expressed as a matrix equation in which each matrix or vector element belongs to a field, typically the real numbers $\Re$. In principle, there are two groups of methods for the solutions of linear systems:

1. *Direct methods or elemination methods*, the exact solution, in principle, is determined through a finite number of arithmetic operations (in real arithmetic leaving aside the influence of round-off errors).

2. *Iterative methods* generate a sequence of approximations to the solution by repeating the application of the same computational procedure at each step of the iteration.

A key consideration for the selection of a solution method for a linear system is its structure. Roughly speaking, direct methods are best for full (dense) matrices, whereas iterative methods are best for very large and sparse matrices. We start with a set of linear equations in n unknowns $x_1, x_2, \ldots, x_n$:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1, \\ a_{21}x_2 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2, \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n. \end{aligned}$$

We can write the equations as a matrix-vector equation by letting A $= (a_{ij}), x = (x_j)$, and $= (b_i)$, as

$$Ax = b \tag{1}$$

If A is nonsingular, $x = A^{-1}b$ is the unique solution vector.

Gaussian elemination is the classical procedure for solving linear equations. The very basic idea of the Gaussian elimination method is to use the first equation to eliminate the first unknown form the last $n-1$ equations, then use the new second equation to eliminate the second unknown form the last $n-2$ equations, etc. This way, by $n-1$ such eliminations the given linear system is transformed into an equivalent linear system that is of *triangular* form.

$$\begin{aligned} b_{11}x_1 + b_{12}x_2 + \ldots + b_{1n}x_n &= z_1 \\ b_{22}x_2 + \ldots + b_{2n}x_n &= z_2 \\ &\vdots \\ b_{n-1,n-1}x_{n-1} + b_{nn}x_n &= z_{n-1} \\ b_{nn}x_n &= z_n \end{aligned}$$

The triangular system can be solved recursively by first obtaining x_n from the last equation, then obtaining x_{n-1} from the second to last equation, etc. This procedure is known as *backward substitution*,

$$x_n = z_n/b_{nn},$$

$$x_i = \frac{1}{b_{ii}}(z_i - \sum_{k=i+1}^{n} b_{ik}x_k), \quad i = n-1, n-2, \ldots, 1$$

Upper Triangularizing:

Assume that A is an $n \times n$ matrix. *Gauss transformations* $M_1, \ldots, M_{n-1}$ can usually be found such that $M_{n-1} \ldots M_2 M_1 A = U$ is upper triangular. Observe that during the k^{th} step:

1. We are confronted with a matrix $A^{(k-1)} = M_{k-1}, \ldots M_1 A$ that is upper triangular in columns 1 to $k-1$.
2. The multipliers in M_k are based on the entries in column k, form rows k+1 to n of $A^{(k-1)}$. In particular, we need $a_{kk}^{(k-1)} \neq 0$ to proceed.

Noting that complete upper triangularization is achieved after (n-1) steps. It is easy to check that

$$A = LU$$

where, $L = M_1^{-1} \ldots M_{n-1}^{-1}$.

Definition: A factorization of a matrix A into a product

$$A = LU$$

of a lower triangular matrix L and upper triangular matrix U is called an *LU decomposition* of A.

Observe that the decomposition is not unique. We will make the choice, $L_{ii} = 1$. The solution to the original $Ax = b$ problem is then found by solving triangular systems:

$$Ly = b, and \quad Ux = y$$

The LU decomposition is a "high level" algebraic description of Gaussion elimination.

We wish to point out that not every nonsingular matrix allows an LU decomposition. For example,

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

has no LU decomposition.

However, since Gaussian elimination with row reordering always works, for each nonsingular matrix there is a permutation matix P such that PA has an LU decomposition. A *permutation matrix* is just the identity with its rows re-ordered.

In order to control the influence of roundoff errors we want to keep the quotient $a_{kj}^{(k-1)}/a_{kk}^{(k-1)}$ small; i.e. we want to have large pivot element $a_{kk}^{(k-1)}$. Therefor, instead of only requiring $a_{kk}^{(k-1)} \neq 0$, in practice, either *complete pivoting* or *partial row or column pivoting* is employed. For complete pivoting, both the rows and columns are reorderd such that $a_{kk}^{(k-1)}$ has maximum absolute value in the $(n-k+1) \times (n-m+1)$ matrix remaining. For row(column) pivoting the rows(columns) are reorderd such that $a_{kk}^{(k-1)}$ has maximum absolute value in the $(k-1)^{th}$ column(row). In this paper, we are only using partial row pivoting.

2 Computing LU-decomposition

We wish to construct an LU decomposition using recursive strategy. If n=1, then $L = I_1$ and $U = A$. For $n > 1$, we break A into four blocks:

$$A = \left(\begin{array}{c|c} a_{11} & a_{12} \ldots a_{1n} \\ \hline a_{21} & a_{22} \ldots a_{2n} \\ a_{n1} & a_{n2} \ldots a_{nn} \end{array} \right) = \begin{pmatrix} a_{11} & w^T \\ v & A' \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 \\ \frac{v}{a_{11}} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & A' - \frac{vw^T}{a_{11}} \end{pmatrix}$$

Where v is a size (n-1) column vector, w^T is a size (n-1) row vector, and A' is an $(n-1) \times (n-1)$ matrix. vw^T formed by taking outer product of v and w. $A' - \frac{vw^T}{a_{11}}$ is called schur complement of A with respect to a_{11}. We now recursively find an LU decomposition of the schur complement. Let us say that $A' - \frac{vw^T}{a_{11}} = L'U'$. Where L' is unit lower-triangular and U' is upper triangular.

$$A = \begin{pmatrix} 1 & 0 \\ \frac{v}{a_{11}} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & L'U' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{v}{a_{11}} & L' \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & U' \end{pmatrix} = LU$$

LU-Decomposition Algorithm:

```
For k=1 to n-1
    $u_{kk} = a_{kk}$
    For i=k+1 to n
        $u_{ki} = a_{ki}$          $u_{ki}$ holds $w_i^T$
        $l_{ik} = \frac{a_{ik}}{a_{kk}}$          $l_{ik}$ holds $v_i$
        For i=k+1 to n
            For j=k+1 to n
                $a_{ij} = a_{ij} - l_{ik}u_{kj}$
```

The six permutation of the indices i, j, and k give six different organiztions of LU-decomposition, and we call these the "ijk"forms. The kij and kji forms are *immediate update* algorithms in that the elements of A are updated when the necessary multipliers are known. This is in opposition ot the other forms, which are *delayed update* algorithms

Generally, in solving a system of linear equations, $Ax = b$, we must pivot on off-diagonal elements of A to avoid dividing by 0. Dividing by any small value can result in mumerical instabilies in the computation. Therefore we try to pivot on a large value.

LU-Decomposition Algorithm with pivoting:

```
For k=1 to n-1
    choose $l$ so that $|a_{lk}| = \max_{k \le i \le n} |a_{ik}|$, swap rows $l$ and $k$
    For i=k+1 to n
        $a_{ik} = \frac{a_{ik}}{a_{kk}}$
        For j=k+1 to n
            $a_{ij} = a_{ij} - a_{ik}a_{kj}$
```

Here L and U are constructed by overwriting A. The parallelism in the above algorithm is trivial. At step k all $(n-k)^2$ scalar updates are independent. The pivoting, swapping, scaling, and updating steps could be parallelized with appropriate data layout.

3 LogP model

LogP is a model of a distributed-memory multiprocessor in which processors communicate by point-to-point messages. The model specifies the performance characteristics of the interconnection network, without describing the structure of the network. The main parameters of the model are:

L: an upper bound on the latency, or delay, incurred in communicating a message containing a numerical value from its source module to its target module.

o: the overhead, defined as the length of time that a processor is engaged in the transmission or reception of each message; during this time, the processor can not perform arithmetic operations.

g: the gap, defined as the minimum time interval between consecutive message transmissions or consecutive message receptions a message.

P: the number of processor/memory modules

Therefore, on the LogP model, sending a fixed sized message from one processor to another processor will require $2o + L$ time steps. All our parallel LU-decomposition algorithms will be analyzed on LogP.

4 Running time analysis on Sequential Algorithms based on LogP model

Sequential non-pivoting:

In non-pivoting sequential LU-decomposition, during kth iteration: scaling takes $(n-k)$ arithmetic operations and updating takes $(n-k)^2 \times 2$ arithmetic operations. So the total running time, $T_1 = \sum_{k=1}^{n-1}(n-k) + 2(n-k)^2 = \frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{n}{6}$

Sequential pivoting:

In pivoting seqntial LU-decomposition, during kth iteration: pivoting takes $n-k-1$; swapping rows takes n; scaling takes $(n-k)$; updating takes $2(n-k)^2$ operations. So the total running time, $T_2 = \sum_{k=1}^{n-1}(n-k-1) + n + (n-k) + 2(n-k)^2 = \frac{2}{3}n^3 + n^2 - \frac{8n}{9} + 1$

From the analysis on sequential LU-decomosition algorithm with pivoting and without pivoting, we found that the pivot and swap processing did not change the asymptotic growth rate of the running time. We also notice that the computation time is to the power three of the matrix size. Due to the huge amount of computation time when the size of matrix increase, parallel LU-decomposition algorithms were impelemented in order to achieve more efficient running time.

5 Running time analysis on Parallel Algorithms for Different Data Layouts based on LogP model

We will analysis the running time for the Parallel pivoting LU-decomposition algorithm on six different data layout. They are column block, column cyclic, row block, row cyclic, blocked grid, and scattered grid data layouts. Our goal is to show the influence of different datalayout to the same algorithm.

5.1 Column and Row Data Layouts

Column Block Data Layout

In column block data layout, contiguous n/P columns are allocated to each processor. Finding absolute maximum element of the column is a local operation. However, after the maxinum element is found or it is found that the matrix is singlar matrix, it needs to be broadcasted related information to all the other processors. All the processors will then perform the swap operation or terminate the algorithm based on the information it recieved. When current dominate processor P_k finished the scaling phase, it will broadcast the result to all the processor P_i $(i > k)$. Then, P_i $(i \geq k\ )$ will perform the update phase in parallel. We can find that P_i will be idel after ith iteration. This makes the parallel algorithm is not so effient because the load balancing is poor.

During the kth iteration: finding maximum element takes $(n - k - 1)$; broadcast swap information takes $L + g(P - 2) + 2o$; swapping rows takes $\frac{n}{P}$; scaling phase takes $(n - k)$; broadcast multiplier takes $(n - k - 1)g + (P - \lceil\frac{kP}{n}\rceil - 1)(L + 2o)$; updating phase takes $2\frac{n}{P}(n - k)$ operations. So the total running time, $T_3 = \sum_{k=1}^{n-1}[n - k - 1 + L + g(P - 2) + 2o + \frac{n}{P} + (n - k) + (n - k - 1)g + (P - \lceil\frac{kP}{n}\rceil - 1)(L + 2o) + \frac{n}{P}(n - k)] = \frac{n^3}{P} + (1 + \frac{g}{2})n^2 + O(n)$

Row Block Data Layout

In row block data layout, contiguous n/P rows are allocated to each processor . Finding max element of the column is not a local operation anymore. Assume processor P_k is the current dominate processor, then for each processor P_i $(i > k)$, it will first find the local maxinum number, which takes $\frac{n}{P} - 1$ steps. Then, by using tournament tree, we can get the maxinum number in $\log(P - \lceil\frac{kP}{n}\rceil - 1) \times (L + 20 + 1)$. After broadcasting pivoting related information to all the other processors, the processor who contains the maxninum number will swap the maxth row with current dominate processor with kth row. After pivoting, current dominate processor P_k will broadcast its current digonal element to all the processor P_i $(i > k)$ to let them perform the division phase in parallel. Finally,these processor need to broadcast the division result to all the processor P_i $(i > k)$. Then, P_i $(i \geq k\ )$ will perform the update phase in parallel. Same as the column block, we can find that P_i will also be idel after ith loop. Thus, the row block data layout also makes the parallel algorithm not so efficient.

During the kth iteration : finding maximum element takes $\frac{n}{P} - 1 + \log(P - \lceil\frac{kP}{n}\rceil - 1) \times (L + 2o + 1)$; broadcast swap information takes ¿$L + g(P - 2) + 2o$; swap rows takes $2[L + (n - 1)g + 2o] + n$; broadcast matrix[k][k] takes $L + g(P - \lceil\frac{kP}{n}\rceil - 1) + 2o$; division phase takes $\frac{n}{P}$; broadcast division information takes $(n - k - 1)g + (P - \lceil\frac{kP}{n}\rceil - 1)(L + 2o)$; update phase takes $2\frac{n}{P}(n - k)$ operations.

So the total running time, $T_4 = \sum_{k=1}^{n-1}[\frac{n}{P} - 1 + \log(P - \lceil\frac{kP}{n}\rceil - 1) \times (L + 20 + 1) + L + g(P - 2) + 2o + 2[L + (n - 1)g + 2o] + n + \frac{n}{P} + (n - k - 1)g + (P - \lceil\frac{kP}{n}\rceil - 1)(L + 2o) + \frac{n}{P}(n - k)] = \frac{n^3}{P} + (1 + \frac{5g}{2} + \frac{1}{P})n^2 + O(n)$

Column Cyclic Data Layout

In column cyclic data layout, finding max element of the column is a local operation in current dominate processor. All the processors will then perform the swap operation or terminate the algorithm based on the broadcasted maxinum information it recieved. When current dominate processor P_k finished the division phase, it will broadcast the result to all the processor P_i $(i > k)$. Then, P_i $(i \geq k\ )$ will perform the update phase in parallel. We can notice that since we assign the column cyclicly, the running time of this algorithm is more efficient.

During the kth iteration : finding maximum element takes $n-k-1$; broadcast swap information takes $L+g(P-2)+2o$; swap rows takes $\frac{n}{P}$; division phase takes $(n-k)$; broadcast division information takes $(n-k-1)g+(P-1)(L+2o)$; update phase takes $2\frac{(n-k)^2}{P}$ operations.

So the total running time, $T_5 = \sum_{k=1}^{n-1} n-k-1+L+g(P-2)+2o+\frac{n}{P}+(n-k)f_d+(n-k-1)g+(P-1)(L+2o)+\frac{(n-k)^2}{P}f_d = \frac{2n^3}{3P}+(1+\frac{g}{2})n^2+O(n)$.

Row Cyclic Data Layout

In row cyclic data layout, similar to the row block data layout, finding max number of the column is not a local operation, either. Then, by using tournament tree, we can get the maxinum number in $\log(P-\lceil\frac{kP}{n}\rceil-1)\times(L+2o+1)$. After broadcasting pivot related information to all the other processors, the processor who contains the maxninum number will swap the maxth row with current dominate processor with kth row. After pivoting, current dominate processor P_k will broadcast its current digonal element to all the processor P_i $(i>k)$ to let them perform the division phase in parallel. Finally,these processor need to broadcast the division result to all the processor P_i $(i>k)$. Then, P_i $(i\geq k\ )$ will perform the update phase in parallel. Similar to the column cyclic data layout, it is also more effient.

During the kth iteration : find max element takes $\lceil\frac{n-k}{P}\rceil-1+\log P\times(L+20+1)$; broadcast swap information takes $L+g(P-2)+2o$; swap rows takes $2[L+(n-1)g+2o]+n$; broadcast matrix[k][k] takes $L+g(P-2)+2o$; division phase takes $\lceil\frac{n-k}{P}\rceil$; broadcast division information takes $(n-k-1)g+(P-1)(L+2o)$; update phase takes $2\lceil\frac{n-k}{P}\rceil(n-k)$ operations.

So the total running time, $T_6 = \sum_{k=1}^{n-1}\lceil\frac{n-k}{P}\rceil-1+\log P\times(L+20+1)+L+g(P-2)+2o+2[L+(n-1)g+2o]+n+\lceil\frac{n-k}{P}\rceil+(n-k-1)g+(P-1)(L+2o)+\lceil\frac{n-k}{P}\rceil(n-k) = \frac{2n^3}{3P}+(1+\frac{5g}{2})n^2+O(n)$.

5.2 Grid Data Layouts

Blocked Grid Data Layout

In blocked grid data layout each processor is assigned $n/\sqrt{P}\times n/\sqrt{P}$ submatrix block of A is assigned to each processor. This assignment leads to a load imbalance.

During the k^{th} iteration the current dominate processor first has to get the part of k^{th} column from other processors and then finds the abslout maximum. It then broadcast the this information to all other processors. If needed the swapping may occur between coresponding processors. Then scaling and sending appropriate partial rows and columns (multipliers) to corresponding processor will happen. Finally all the active processors will update the submatrix.

Getting the part of the k^{th} column from other processors will take $L+2o+(P-2-\lfloor\frac{k\sqrt{P}}{n}\rfloor)g$; finding maximum takes $n-k-1$; Sending pivoting information takes $\lceil lgP\rceil(L+2o)+(\lceil lgP\rceil-1)g$; swapping takes $L+2o+n/\sqrt{P}$; Sending partial k^{th} row and column (multipliers) for updating takes $4[L+2o+(P-2-$

$\lfloor k\sqrt{P}/n \rceil)g+n/\sqrt{P}]$; updating takes $2(n/\sqrt{P}\times n/\sqrt{P})$. $\sum_{k=1}^{n-1} L+2o+(P-2-\lfloor \frac{k\sqrt{P}}{n} \rfloor)g+(n-k-1)+\lceil lgP \rceil(L+2o)+(\lceil lgP \rceil -1)g+L+2o+n/\sqrt{P}+4[L+2o+(P-2-\lfloor k\sqrt{P}/n \rceil)g+n/\sqrt{P}]+2(n/\sqrt{P}\times n/\sqrt{P}) = \frac{2n^3}{P}+(5/\sqrt{P}+1/2-2/P)n^2+O(n)$

Scattered Grid Data Layout

In scattered grid data layout, each processor receives a submatrix of A determined by a set of $n/\sqrt{P}$ rows and columns, and they are scattered $\sqrt{P}$ apart.

During the k^{th} iteration the current dominate processor first has to get the part of k^{th} column from other processors and then finds the abslout maximum. It then broadcast the this information to all other processors. If needed the swapping may occur between coresponding processors. Then scaling and sending appropriate partial rows and columns (multipliers) to corresponding processor will happen. Finally all the active processors will update the submatrix.

Getting the part of the k^{th} column from other processors will take $L+2o+(\sqrt{P}-1)g$; finding maximum takes $n-k-1$; Send pivoting information takes $\lceil lgP \rceil(L+2o)+(\lceil lgP \rceil-1)g$; swapping takes $L+2o+n/\sqrt{P}$; Sending partial k^{th} row and column (multipliers) for updating takes $4[L+2o+(\sqrt{P}-1)g+n/\sqrt{P}]$; updating takes $2(n-k)^2/P$. $\sum_{k=1}^{n-1} L+2o+(\sqrt{P}-1)g+(n-k-1)+\lceil lgP \rceil(L+2o)+(\lceil lgP \rceil-1)g+L+2o+n/\sqrt{P}+4[L+2o+(\sqrt{P}-1)g+n/\sqrt{P}]+2(n-k)^2/P) = \frac{2n^3}{3P}+(1/2+2/\sqrt{P}+1/P)n^2+O(n)$

6 Implementation and Result

We implement two sequential LU-decomposition and six parallel LU-decomosition by the MPI on a seriel of SUN workstation. We ranomly creat matrix with 7 different sizes of 4×4, 8×8, 16×16, 32×32, 64×64, 128×128, 256×256 on 1,2,4,8 processors. For some data layouts we even use the 16 virtual processors. The following tables show the parallel running time divided by corresponding serial running time of our implementation.

7 Conclusion

In this paper, we saw the effects from different type of data layout to the same algorithm. The results we got both from theoretical analysis part and the implementation part are more matched each other when the matrix size increases. This shows that communication time dominates when the matrix size is small, and computation time dominates when the matrix size is large. Thus, to solve very large size matrix, parallel implementation is much better than the sequential implementation.

From the theoretical analysis results and our implementation results, we can see that the different of data layout do effects the running time of the parallel algorithm. While by using the LogP model, we successfully predict the running time of the algorithm. Hence, the data layout should also be carefully chosen since it takes an important role in parallel implemenation.

References

[1] Bertsekas, D. P., and Tsitsiklis, J. N. *Parallel and Distributed Computation: Numerical Methods.* Athena Scientific, Belmont, Massachusetts, 1997.

[2] Chern, M.-Y., and Murata, T. A fast algorithm for concurrent lu decomposition and matrix inversion. *IEEE* (1983).

[3] Culler, D., Karp, R., Patterson, D., Sahay, A., Santos, E. E., Schauser, K. E., Subramonian, R., and von Eicken, T. Logp: a practical model of parallel computation. *Communications of the Association for Computing Machinery 39*, 11 (November 1996), 78–85.

[4] Dekel, E., Nassimi, and Sahni, S. Parallel matrix and graph algorithms. *SIAM Journal of Computing 10*, 4 (November 1981).

[5] Demmel, J. W., Heath, M. T., and van der Vorst, H. A. Parallel numerical linear algebra. Tech. Rep. UCB//CSD-92-703, U. C. Berkeley, October 1992.

[6] Gallivan, K., Plemmons, R., and Sameh, A. Parllel algorithms for dense linear algebra computations. *SIAM Review 32*, 1 (March 1990), 54–135.

[7] Geist, G. A., and Romine, C. H. Lu factorization algorithms on distributed memory multiprocessor architectures. *SIAM J. Sci. Stat. Comput. 9*, 4 (July 1988).

[8] Golub, G. H., and van Loan, C. F. *Matrix Computations*, 3rd ed. The Jones Hopins University Press, Baltimore and London, 1996.

[9] Kress, R. *Neumerical Analysis.* Springer-Verlag New York Inc., 1998.

[10] Liu, Z., and Cheung, D. Efficient parallel algorithm for dense matrix lu decomposition with pivoting on hypercubes. *Computers Math. Applic 33*, 8 (1997), 39–50. Great Britain.

[11] Ortega, J. The ijk forms of factorization methods i. vector computers. *Parallel Computing 7* (1988), 135–147. North-Holland.

[12] Ortega, J., and Romine, C. The ijk forms of factorization methods ii. parallel systems. *Parallel Computing 7* (1988), 149–162. North-Holland.

[13] Toledo, S. Locality of reference in lu decomposition with partial pivoting. *SIAM J. Matrix Anal. Appl. 18*, 4 (October 1997), 1065–1081.

[14] von Laszewski, G., Parashar, M., Mohamed, A. G., and C.Fox, G. On the parallelization of blocked lu factorization algorithms on distributed memory architectures. *IEEE* (1992).

matrix size	P=2	P=4	P=8	P=16
4*4	61.18	202.16		
8*8	48.9	88.98	3021.9	
16*16	44.98	53.18	3281.75	30534.41
32*32	14.72	23.08	469.95	2576.09
128*128	3.84	3.96	66.7	118.98
256*256	1.89	1.87	34.67	56.8

Column Block Data Layout

matrix size	P=2	P=4	P=8	P=16
4*4	52.78	94.99		
8*8	47.56	88.09	6076.42	
16*16	44.67	83.85	6818.69	15710.64
32*32	21.01	32.67	1156.09	2454.09
128*128	2.09	4.67	85.47	175.68
256*256	1.75	2.67	33.07	68.38

Column Cyclic Data Layout

matrix size	P=2	P=4	P=8	P=16
4*4	148.09	299.48		
8*8	50.67	99.38	2365.02	
16*16	43.98	55.69	3286.09	30550.09
32*32	14.9	25.08	479.03	2557.98
128*128	2.94	2.64	64.46	119.45
256*256	1.91	0.67	33.79	50.89

Row Block Data Layout

matrix size	P=2	P=4	P=8	P=16
4*4	143.09	421.67		
8*8	125.58	236.99	10004.46	
16*16	111.57	159.49	13062.34	35965.05
32*32	61.99	91.56	1441.67	2007.09
128*128	15.8	22.69	238.45	346.63
256*256	8.78	12.94	117.45	165.9

Row Cyclic data Layout

matrix size	P=4	
4*4	1091.59	
8*8	54.39	
16*16	27.97	
32*32	4.51	
128*128	0.86	
256*256	0.6	

Blocked Grid data Layout