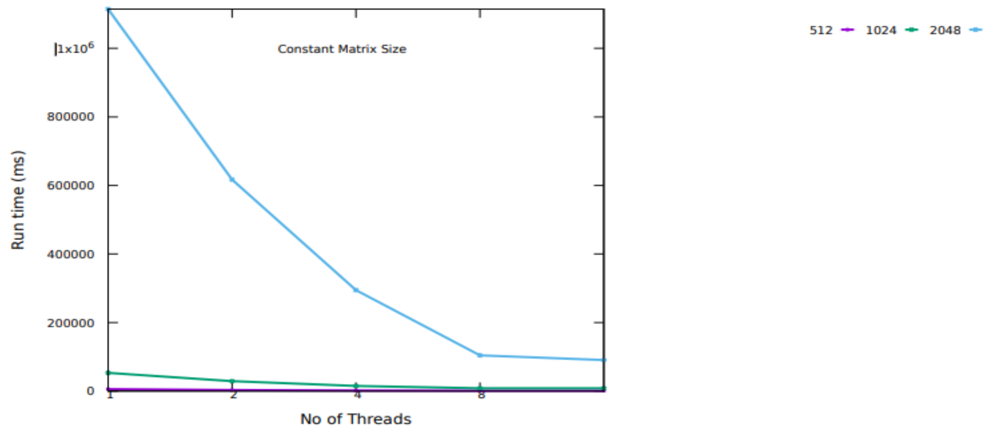
HPC ASSIGNMENT 1

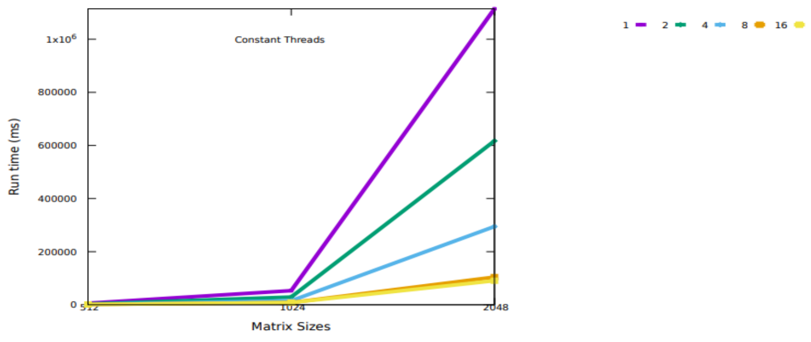
Group 14

1. Tanisha Rana SE20UCSE202
2. Jahnavi Siripurapu SE20UCSE178
3. Harshit Reddy SE20UCSE051
4. Ridhi Bigala SE20UCSE138
5. Amrutha Manne SE20UARI017
6. Agastya Gandra SE20UARI011

1)Ordinary Matrix Multiplication

1. At every iteration of the outer loop on i variable a single row of
2. Matrix A and all columns of matrix B are processed
3. m·l inner products are calculated to perform the matrix multiplication
4. The complexity of the matrix multiplication is O(mnl)





2)Block Matrix Multiplication

A fine-grained approach – the basic subtask is calculation of one element of matrix C. Number of basic subtasks is equal to n2. Achieved parallelism level is redundant!

As a rule, the number of available processors is less then n2 (p<n2), so it will be necessary to perform the subtask scaling

The aggregated subtask – the calculation of one row of matrix C (the number of subtasks is n)

Data distribution – row wise block-striped decomposition for matrix A and column wise block striped decomposition for matrix B

Analysis of Information Dependencies:

Each subtask hold one row of matrix A and one column of matrix B. At every iteration each subtask performs the inner product calculation of its row and column, as a result the corresponding element of matrix C is obtained

Then every subtask i, 0≤ i<n, transmits its column of matrix B for the subtask with the number (i+1) mod n.

Aggregating and Distributing the Subtasks among the

Processors:

In case when the number of processors p is less than the number of

basic subtasks n, calculations can be aggregated in such a way that

each processor would execute several inner products of matrix A rows

and matrix B columns. In this case after the completion of

computation, each aggregated basic subtask determines several rows

of the result matrix C,

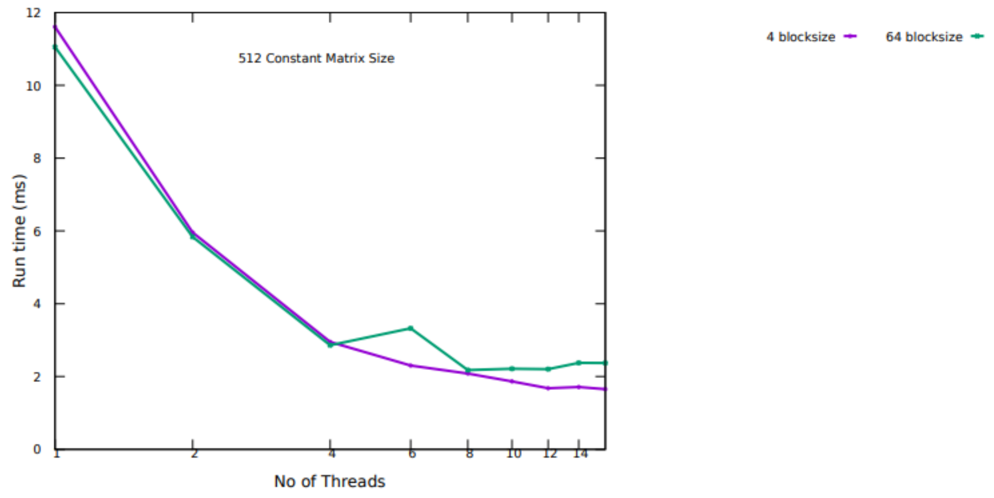
Under such conditions the initial matrix A is decomposed into p

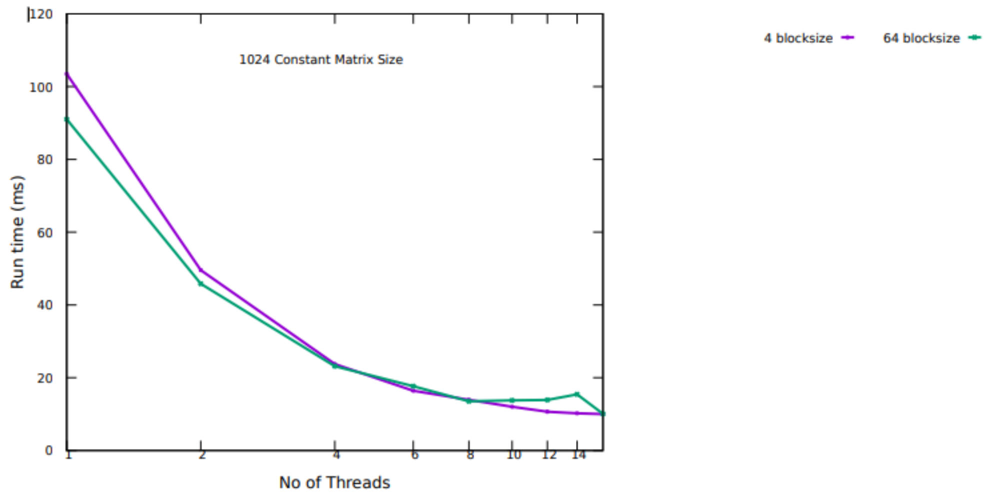
horizontal stripes and matrix B is decomposed into p vertical stripes,

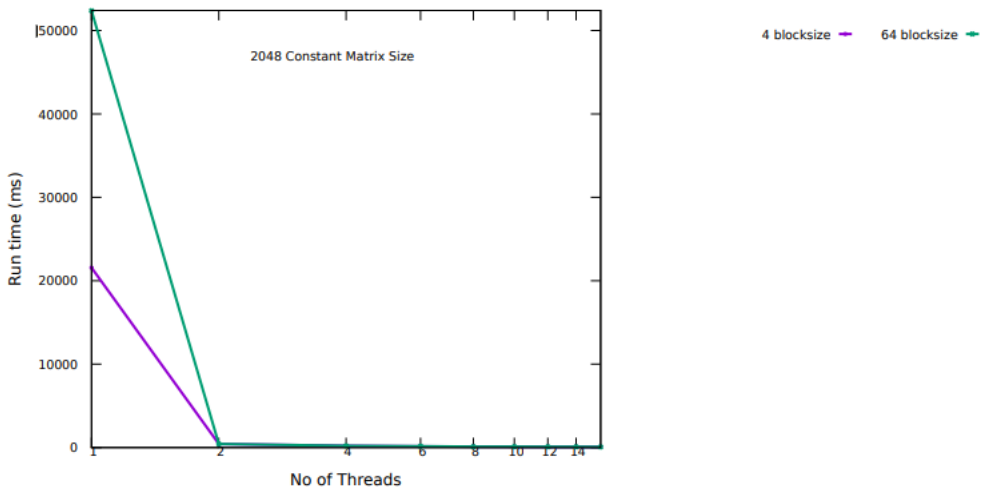
Subtasks distribution among the processors must meet the

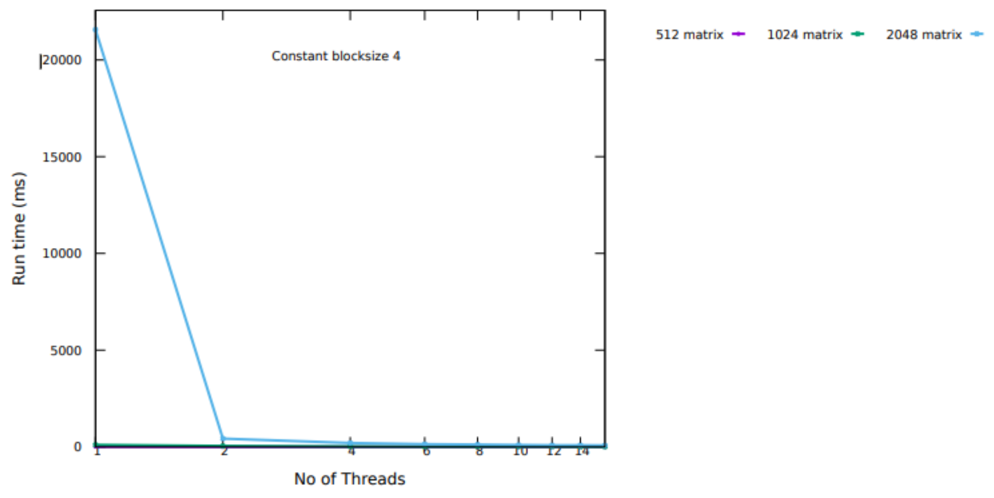
requirements of effective representation of the ring structure of subtask

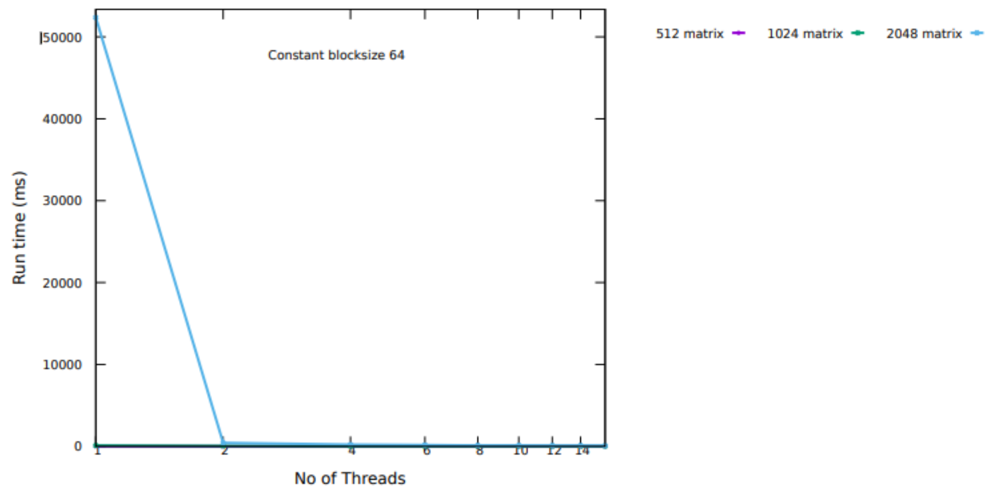
information dependencies.


512 Constant Matrix Size
Run time (ms)
No of Threads
4 blocksize
64 blocksize
0
2
4
6
8
10
12
1
2
4
6
8
10
12
14


1024 Constant Matrix Size
Run time (ms)
No of Threads
4 blocksize
64 blocksize
0
20
40
60
80
100
120
1
2
4
6
8
10
12
14


2048 Constant Matrix Size
Run time (ms)
No of Threads
4 blocksize
64 blocksize
0
10000
20000
30000
40000
50000
1
2
4
6
8
10
12
14





3)Ordinary Matrix Multiplication Transpose

We assume that the matrix is distributed over a P × Q processor template with a block cyclic data distribution. P, Q, and the block size can be arbitrary

The communication schemes of the algorithms are determined by the greatest common divisor (GCD) of P and Q.

If P and Q are relatively prime, the matrix transpose algorithm involves complete exchange communication.

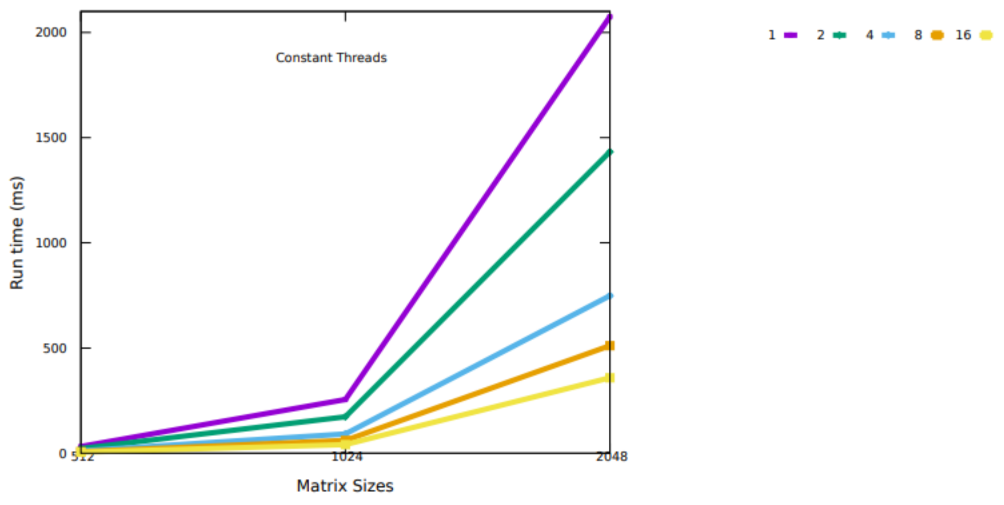
If P and Q are not relatively prime, processors are divided into GCD groups and the communication operations are overlapped for different groups of processors.

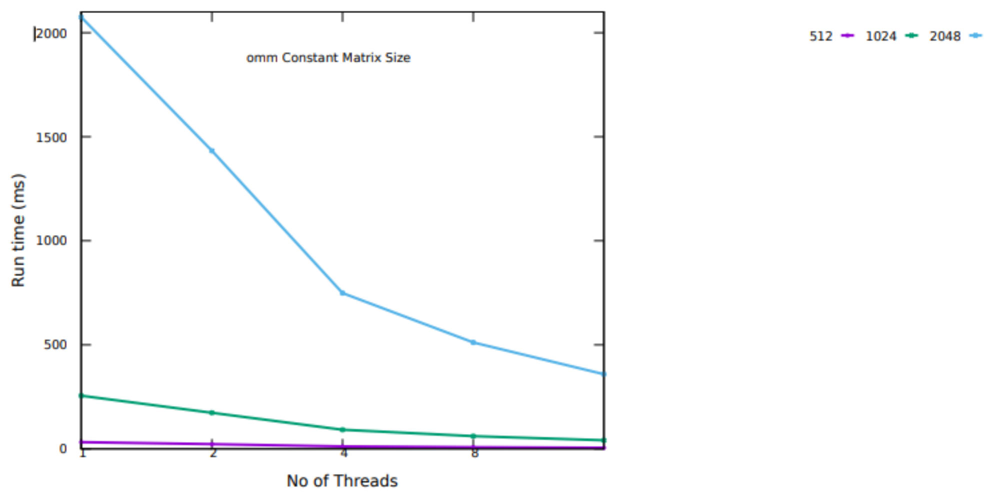
Processors transpose GCD wrapped diagonal blocks simultaneously, and the matrix can be transposed with LCM/GCD steps, where LCM is the least common multiple of P and Q. We make use of non-blocking, point-to-point communication between processors.

The use of nonblocking communication allows a processor to overlap the messages that it sends to different processors, thereby avoiding unnecessary synchronization.

Combined with the matrix multiplication routine, C = A · B, the algorithms are used to compute parallel multiplications of transposed matrices, C = AT · BT, in the PUMMA package [5].

Details of the parallel implementation of the algorithms are given, and results are presented for runs on the Intel Touchstone Delta computer.





4)Block Matrix Multiplication Transpose

We assume that a matrix is distributed over a two-dimensional processor mesh, or template, so that in general each processor has several blocks of the matrix. For example, a matrix with 12 x12 blocks is distributed over a 2x3 template.

The concept of the LCM block is very useful for implementing algorithms that use a block scattered data distribution. Blocks belong to the same processor if their relative locations are the same in each square LCM block. An algorithm may be developed for the first LCM block, and then it can be directly applied to the other LCM blocks, which all have the same structure and the same data distribution as the first LCM block. That is, when an operation is executed on a block of the first LCM block, the same operation can be done simultaneously on other blocks, which have the same relative location in each LCM block.

A matrix A, distributed over a P Q processor template, has Mb Nb blocks and each block consists of r s elements, where r and s are arbitrary.

If A is transposed, the transposed matrix AT

is distributed over the same P Q template, and it has Nb Mb blocks and each block has s r elements. The elements of each block remain in the same block, but may be in a different processor, and each block is itself transposed. blocks in the first processor P0 are scattered to all processors. On a 3 3 square template, the blocks in each processor are not dispersed, but they are moved as one entity to a different processor. Parallel matrix transpose algorithms for the block scattered data distribution have several communication patterns determined by the greatest common divisor (GCD) of P and Q.

P and Q : relatively prime

We start with the simple case where P and Q are relatively prime, i. e. GCD = 1. In this case blocks in P0 are scattered to all processors after being locally transposed case blocks in P0 are scattered to all processors after being locally transposed. This case involves the two-dimensional complete exchange communication. That is, every processor needs to communicates with every other processor. The complete exchange problem is implemented by direct communication between sender and receiver. P hp; qi represents

PMOD(p;P);MOD(q;Q) in the processor template. Processor P hp; qi (0 p<P and 0 q<Q) starts to transpose blocks whose transposed blocks belong to itself.

Then it deals with blocks whose transposition are in processors in the same column of the template (P hpi; qi, 0 i<P).The processor sends blocks to its top neighbour, P hp 1; qi, and receives blocks from its bottom neighbour, P hp + 1; qi. Before sending the blocks, it is necessary to copy the blocks to be sent into a contiguous message buer. Next it sends blocks to the next top processor, P hp 2; qi, and receives blocks from the next bottom processor, P hp + 2; qi.

After it completes its operations with the processors in the same column, it sends blocks to the processors to the left in the template (P hp i; q 1i, 0 i<P), and receives blocks from the processors to the right (P hp + i; q + 1i). All operations are completed in P Q = LCM steps.

We interpret the algorithm from the matrix point-of-view. In the first LCM block, the above algorithm performs the operation by transposing one (wrapped) diagonal blocks at one step.

(Actually, the algorithm transposes every LCM diagonal blocks of the matrix at each step.)

P and Q : not relatively prime

In the previous section, we have investigated the case when P and Q are relatively prime, which

involves complete exchange communication. In this section we consider the case of GCD > 1.

The former algorithm is a special case (GCD = 1) of this algorithm.

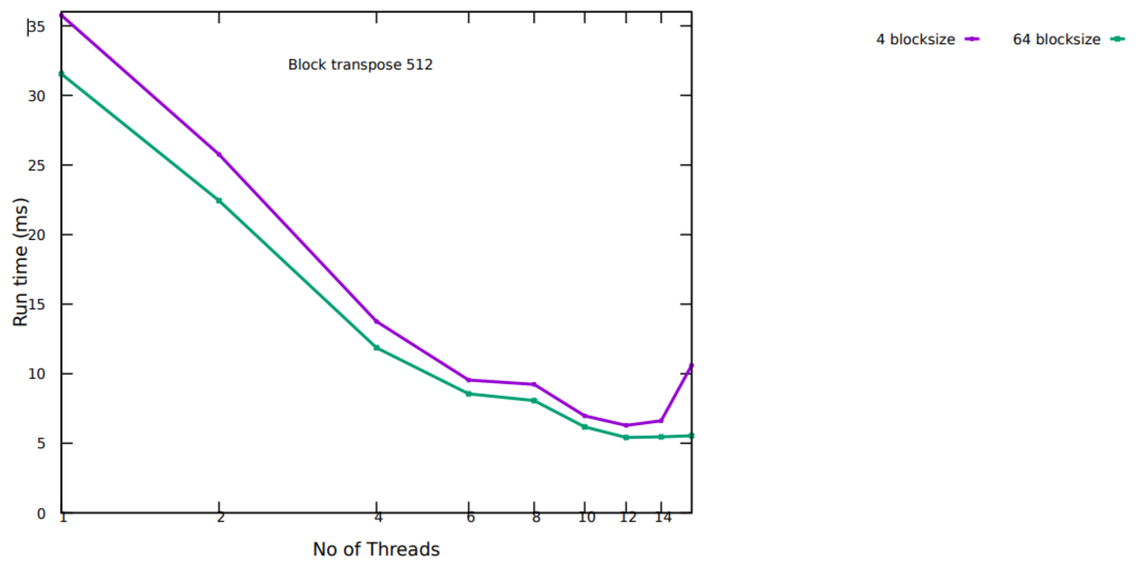
Each processor has its own 3 2 (= LCM=P LCM=Q) array of

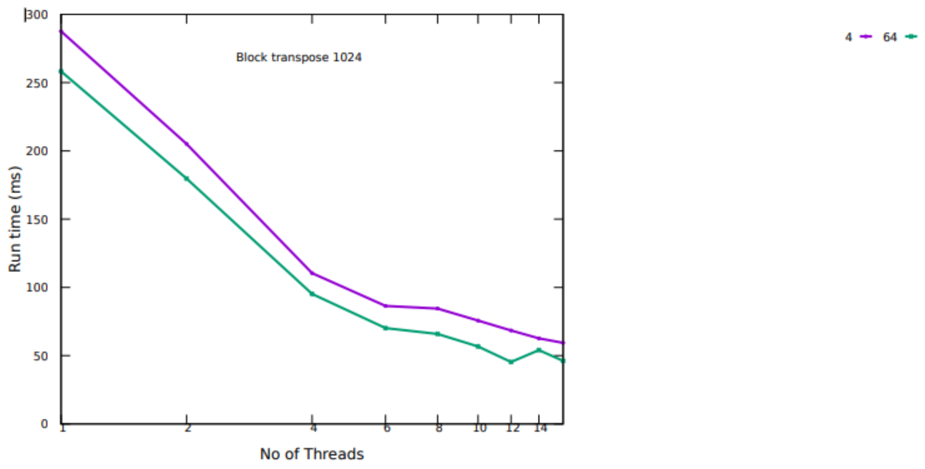
blocks. The processors can transpose the matrix with 6 (= LCM=P LCM=Q = LCM=GCD)

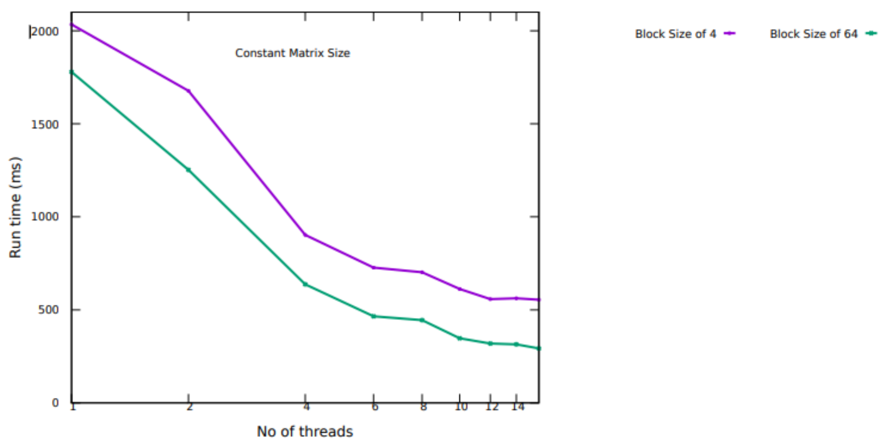
communications steps. a processor P hp; qi starts to communicate

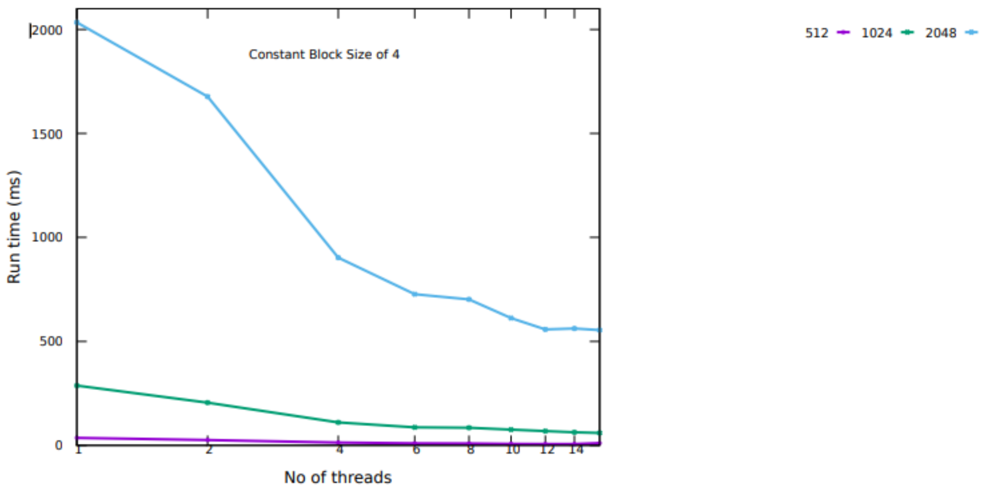
with P hp~; q~i, where ~p and ~q are computed from p and q (details are explained later of this

section). Once P hp; ~ q~i is determined, it communicates with other processors, whose vertical and horizontal intervals are GCD from P hp

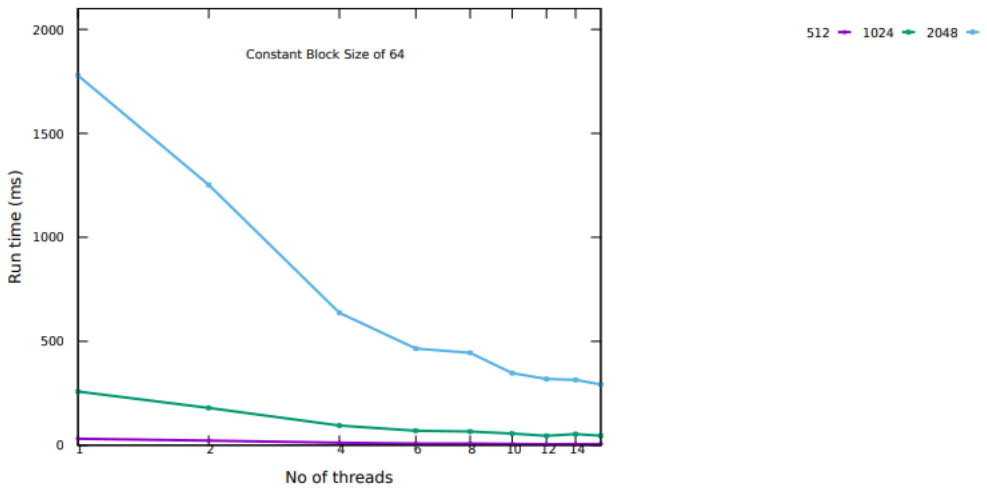

Block transpose 512
4 blocksize
64 blocksize
Run time (ms)
No of Threads
0
5
10
15
20
25
30
35
1
2
4
6
8
10
12
14


Block transpose 1024
4
64
Run time (ms)
No of Threads
0
50
100
150
200
250
300
1
2
4
6
8
10
12
14


Constant Matrix Size
Block Size of 4
Block Size of 64
Run time (ms)
No of threads
0
500
1000
1500
2000
1
2
4
6
8
10
12
14



Constant Block Size of 4
Run time (ms)
No of threads
512
1024
2048
0
500
1000
1500
2000
1
2
4
6
8
10
12
14



Constant Block Size of 64
Run time (ms)
No of threads
512
1024
2048
0
500
1000
1500
2000
1
2
4
6
8
10
12
14