Name: Yuchao Zhang u6715243
Statement:
This report was done by myself. The discussion with Tianyi Wang u6636587 about the mathematical dependencies and the difference between non-blocking and blocking communication contributes to understanding of the requirements in some degree.

Assumption: For cyclic distribution, I apply DPOTRF(1 node), synchronization(1 send and O(nproc-k)), DTRSM(O(nproc-k) node can execute), synchronization in series, and DSYRK and DGEMM in parallel.

Q1.
I have tested on N = 2000, P =Q=1 , nproc=1. I tried to increase the wT by power 2 from 16 to 512. And then according to the cost, I try to find the minimal by dichotomy I find that when wT equals 144, I got the shortest time 8.40e-2s. And the total time for wT from 64 to 512 is waving around 9.20e-2.
I observed that a processor mesh of i by j has closed performance with j by i processor mesh. And I find too many processors does not give a good performance because of the communication overhead. And 2 by 2 processor mesh(gives 5.3e-2s) and 3 by 2(also 2 by 3) processor mesh, if I increase the processor numbers to 9(3 by 3), the cost will reach at least 2.5e-1s and 16 processor costs at least 2.3e-1s. 2 by 2 and 3 by 2(2 by 3) mesh give the best performance because:
1. The processor number is adequate, having less communication overhead.
2. 12 can divide both 2 and 3. So there's no waste processor. It leads to balanced workload in some degree. As a result of that, it shorten the waiting overhead(waiting for other).

Q2.
1. In the lazy model, each processor should send its data cell to other processor for synchronization. And the send and recv are all block send and block recv.
2.The communication cost id defined as (communication numer) *(ts + b(length of word))

DPOTRF calculation(Cholesky factor A[k][k] so tf*O(wT^3) for serial cholesky factor**):**
It taks tf*O(wT^3) to calculate.
Synchronization after DPOTRF: (A[i][k] = A[i][k] * A[k][k]^-T) so O(wT^3)
one process sends A[k][k] to other processors. So it takes (ts(nproc-1) + b wT*wT* wT*tw) to send a float number where b is a positive constant.

DTRSM calculation: (A[i][k] = A[i][k] * A[k][k]^-T**)**
Then the execution of (nproc-k+1) tiles will take 2 * wT^3 * tf * (nproc-k+1).
Synchronization after DTRSM:
The processor undergone the DTRSM will boardcast to the rest nodes. So it takes (ts*(nproc-1)+2 wT*wT* tw). The sending cannot be down in parallel since I use block send and block receive.
DSYRK and DGEMM calculation:(A[i][j] -= A[i][k] * A[j][k]^T) so O(wT^3)
Then, the dgemm calculation will take tf* O(wT^3) , every processor calculate it as parallel

Result
Because the k will increase from 0 to nproc, so the total cost for a paralled cholesky decomposition is (a * tf*(wT^3) + (ts(nproc-1) + b wT^3 * tw) + c * wT^3* tf* (nproc-k+1) + (ts*(nproc-1)+c* wT^2 * tw)+ d*tf*wT^3) / P for each processor

Q3.
At the point of synchronizations, I apply the barrier. So it will introduce load imbalance overhead. Each processor cannot go head until the global updating finishes. So let's assume that the first synchronization will vary from 0 to

Q4.
Testing conditions: N=2000 wT=144 nproc = 96
when p =2

Type	Generate RHS	Factorization	Backsolve time	Total
Cyclic	1.05e-01s	3.95e-01s	2.22e-02s	5.25e-01s
Block	7.92e-03s	2.53e-01s	6.19e-03s	2.82e-01s
Random	5.47e-02s	4.55e-01s	1.18e-02s	5.28e-01s

When p = 12

Type	Generate RHS	Factorization	Backsolve time	Total
Cyclic	2.72e-02s	4.92e-01s	7.59e-03s	5.30e-01s
Block	8.41e-02s	3.01e-01s	7.59e-03s	4.39e-01s
Random	2.05e-02s	4.48e-01s	1.41e-02s	4.83e-01s

random: Test condition nproc=16 p=4 N=1000 wT=144

Time	1	2	3	4
Random	8.11e-03s	2.59e-02s	3.03e-02s	6.48e-02s
Block	4.00e-02s	5.68e-02s	4.65e-02s	7.73e-02s
Cyclic	4.35e-02s	5.95e-02s	4.02e-02s	5.47e-02s

Block distribution outperforms on factorization calculation over other two when number of processor is large.
Random distribution's performance is not stable and it has the largest variance among these three distribution methods.
As I change p from 2 to 12, the performance of block drops but cyclic does not change a lot.
At the same time, backsolve speed of cyclic becomes higher while 'backsolve' speed of block and cyclic are not sensitive to that change.

Q5.
I have applied non-blocking send and receive. In the lazy algorithm, only when all data have been processed can the message being sent out. However, message listening and receiving also take time. So, before the root processor sends message, all the other processor will wait for the root node, leading to the waste of calculation. As I apply the non-blocking send and receive, it does not reduce cost for DPOTRF.

Q6. DPOTRF will send data to DTRSM, and DTRSM will send data to DSYRK and DGERK.
A DPOTRF cannot run in parallel, so it will do calculation and sequentially send data to DTRSM. But in this case, the processor 0 can only send the processor who has different rank with processor 0 and at most send a processor once. For example, if processor 2 deals with A[1][0] and A[2][0] in iteration 0, only send processor 2 with A[0][0] once rather multiple times to reduce the communication cost. And the message only sends to the node has the dependency. For example, if a A[2][0] will only send to A[2][j] (j<=2) and A[j][2] (for nT>j>2). These strategies can greatly reduce the communication cost. Further more, I need to use boardcast the data to specific ranks because the boardcast performs better than other
Q7.
Basic, static and Dynamic Cholesky factorization in Distributed Square Block Packed(DSBP) format have excellent performance(GUSTAVSON 2009). First, basic variant in DSBP halves the storage and make the factorization can be applied on PDPORTF(GUSTAVSON 2009). As for static algorithm variant, it can apply the non-blocking communication so that largely reduce the communication overhead. In terms of dynamic algorithm variant , it can be widely used in distributed and portable memory. The DSBP maps its squared block unit directly to processor so that a data block can directly call kernel function, for example – BLAS kernel that largely decrease the overhead for copying and transformation. What’s more, it also provide a flexible way to store data by implementing block decoupling algorithm, which also leads to high parallelism performance.

The **basic variant** takes advantages of this data structure structure. For example, two data blocks with dependency can be assigned to the same processor to reduce the communication overhead. And it also reuses some transmission rules from the static and dynamic variants.
Unlike the basic variant, the **static variant** can make a schedule(Lee 2000) for next paralleled computation commands and partially update the Cholesky factorization. But it fails to parallel the communication and computation since it splits them into two different phase.
As for **dynamic variant**, it can also apply the schedule given Lee’s statement. The dynamic method can accept in-order messages compared to the static variant requiring strict order message passing.

Some performance tests have been conducted on a AMD’s 192 nodes processors using MPI libraries. The performance of the static, dynamic and even the basic overshadow and ScaLAPACK method in terms of average floats point calculation. Compare simulation result and actual result, the basic model is not precise due to the communication costs while the basic model well predicts the total overhead. As for the scalability, dynamic and static model drops the average calculation from 3.5 Gf/s to 3 Gf/s while

the basic model drops from 3.4Gf/s to 2.7Gf/s as the processor number increases from 5 to 50. As a result, the static and dynamic model have better scalability performance over the basic model.
These three model applied on DSBP guarantees the same or even better performance of the distributed system over traditional full storage. And now these three variants have outperformed the popular ScaLAPACK method, showing that these variants are worth to optimize and study in.

Gustavson, F.K. 2009 , "Distributed SBP Cholesky factorization algorithms with near-optimal scheduling", ACM Transactions on Mathematical Software (TOMS), vol. 36, no. 2, pp. 1-25.

Lee, H., Kim, J., Hong, S. & Lee, S. 2000, "Task scheduling using a block dependency DAG for block-oriented sparse Cholesky factorization", ACM, pp. 641.

Q9.
For the lazy algorithm, we can apply boardcast to shorten the length of the code but the communication cost is almost 3 times over the manually send.
Defining a function is not a good idea in MPI programming, since the parameters and complex pointer (especially nested pointer type) will lead you to the a mistake.