Cholesky Factorization

CS F422: Parallel Computing

Guntaas Singh (2018A7PS0269P)
Harshit Garg (2018A7PS0218P)
K Vignesh (2018A7PS0183P)

1 Introduction

The Cholesky Factorization algorithm is used for factorizing a symmetric positive definite matrix into the form $A_{n\times n} = U^T U$.

```
procedure CHOLESKY(A)
begin
    for k = 0 to n - 1 do
        begin
        A[k, k] = sqrt(A[k, k])
        // DIVISION STEP
        for j = k + 1 to n - 1 do
            A[k, j] = A[k, j] / A[k, k]
        // REDUCTION STEP
        for i = k + 1 to n - 1 do
            for j = i to n - 1 do
                A[i, j] = A[i, j] - A[k, i] * A[k, j]
        endfor
end CHOLESKY
```

The serial algorithm involves three nested loops and carries out $n(n-1)/2$ divisions, n square root operations, $n(n^2-1)/6$ multiplications and subtractions, in total. Therefore,

$$T_{cholesky} = \frac{n(n+1)(2n+1)}{6}$$

The Cholesky Factorization can be used to solve a system of linear equations $Ax = b$, if A is symmetric and positive definite. The factorization yields $A = U^T U$, where U is an upper triangular matrix. We can then solve $U^T y = b$ by forward substitution and $Ux = y$ by back substitution to find the solution.

$$T_{\text{forward substitution}} = n(n-1)$$

$$T_{\text{back substitution}} = n(n-1)$$

$$T_s = \frac{n(n+1)(2n+1)}{6} + 2n(n-1) = O(n^3) \tag{1}$$

2 Decomposition

The input matrix can be row-wise partitioned among the processes (1D data decomposition). Row k ($0 \leqslant k \leqslant n-1$) is updated in the first $k+1$ iterations - subtraction in the first k iterations and division in the last iteration. Hence, the problem can be decomposed further on the basis of tasks for each row.

The resulting task-dependency graph and task-interaction graph are as follows. There are no

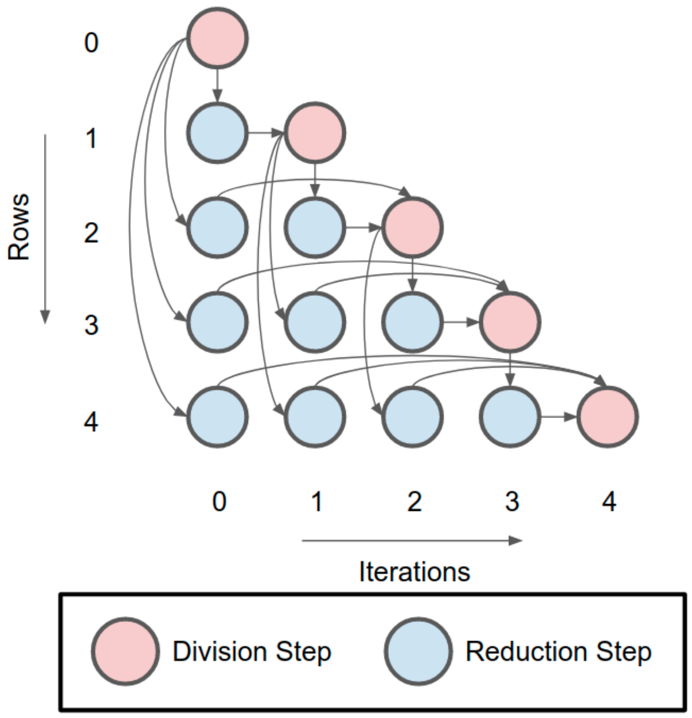

Fig. 1. Task-Dependency Graph and Task-Interaction Graph for a $n \times n$ input matrix ($n = 5$).

interactions between processes other than those necessitated by the task-dependency graph and partitioning of input data. Hence, the task-interaction graph is the same as the task-dependency graph.
The cost associated with a task corresponding to row i ($0 \leqslant i \leqslant n-1$) is $n-i$ for division tasks (red nodes) and $2(n-i)$ for reduction tasks (blue nodes).

$$\text{Critical Path Length} = n + \sum_{i=1}^{n-1} [(n-i) + 2(n-i)] = \frac{3n^2 - 2n}{2} \tag{2}$$

$$\text{Total Work} = \sum_{i=0}^{n-1} [i \times 2(n-i) + (n-i)] = \frac{n(n+1)(2n+1)}{6}$$

$$\begin{aligned} \text{Average Degree of Concurrency} &= \frac{\text{Total Work}}{\text{Critical Path Length}} \\ &= \frac{(n+1)(2n+1)}{3(3n-2)} \end{aligned} \qquad (3)$$

A finer 2D data decomposition of the matrix is also possible, but is expected to yield large communication overhead on account of data dependencies within rows.

3 Mapping

On the basis of data partitioning, tasks corresponding to the same row can be statically mapped to the same process. Since these densely interacting tasks operate on the same row, this mapping reduces communication overhead and eliminates data redundancy (each process only needs to store a disjoint subset of the rows).

Further, we can use block or cyclic distribution schemes for mapping n rows to p processes ($p < n$). Over n iterations, the active region of the matrix continuously shrinks as shown in Figure 2. As a result, a block distribution scheme leads to uneven load distribution producing large process idling overhead. In contrast, with a cyclic distribution scheme, load distribution is more even. Since reduction tasks belonging to the same iteration are independent, an even distribution of these tasks also reduces idling overhead. Hence, we use a cyclic distribution scheme for mapping rows (and associated tasks) to processes.

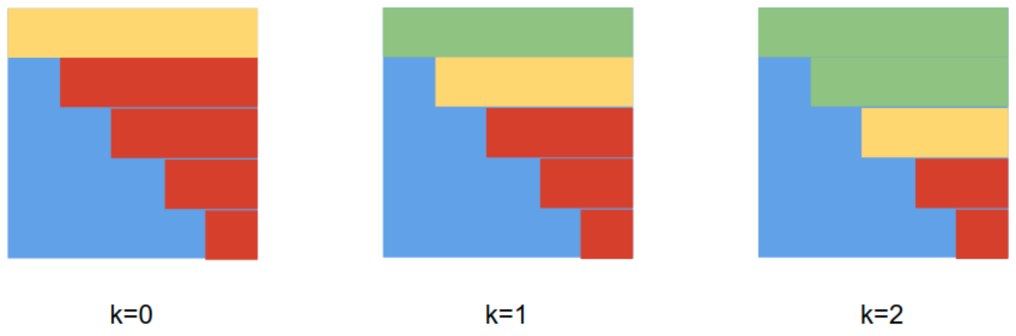

Fig. 2. Iterations of Cholesky Factorization on a 5×5 matrix. The active region for reduction is indicated in red.

4 Algorithm and Implementation

The parallel implementation performs n iterations over each row in order. Each iteration consists of three stages:

1. Computation: Division in row k
2. Communication: Broadcast of row k
3. Computation: Reduction (multiplication and subtraction) in rows $k+1$ to $n-1$

The process owning row k broadcasts its elements only once the division step is finished. Hence, there is no overlap between stages 1 and 2. Since one-to-all broadcast uses recursive doubling, processes receive the contents of row k at different points in time and can immediately proceed with stage 3. The process owning row $k+1$ will receive the broadcast at the end of $log(p)$ steps. Since, division on a row can only be performed after all reduction steps are finished, stage 1 for the next iteration (which will perform division on row $k+1$) will begin only after stage 3 finishes. Hence, there is no overlap between adjacent iterations.

On the basis of these observations, the parallel run-time can be estimated as follows.

$$T_{\text{division}} = \sum_{k=0}^{n-1}[n-k] = \frac{n(n+1)}{2}$$

$$T_{\text{communication}} = \sum_{k=0}^{n-1}[log(p) \times (t_s + t_w(n-k-1))] = log(p)(nt_s + t_w(\frac{(n-1)n}{2}))$$

$$T_{\text{reduction}} = \sum_{k=0}^{n-1}[2 \times \sum_{j=0}^{\lfloor \frac{n-k-2}{p} \rfloor}[n-k-2-jp]] = \frac{(n-2)(n-1)(2n-3)}{6p} + \frac{(n-2)(n-1)}{2}$$

$$T_{\text{data partitioning}} = 2n(t_s + t_w(n))$$

$$T_{\text{forward substitution}} = T_{\text{back substitution}} = n(n-1)$$

Adding these components,

$$T_P = 3n^2 + 2n(t_s + t_w(n)) - 3n + log(p)(nt_s + t_w(\frac{(n-1)n}{2})) + \frac{(n-2)(n-1)(2n-3)}{6p} \quad (4)$$

5 Analysis

Theoretically,

$$\text{Speedup} = \frac{T_s}{T_p} = \frac{n(n+1)(2n+1) + 12n(n-1)}{18n^2 + 12n(t_s + t_w(n)) - 18n + 6log(p)(nt_s + t_w(\frac{(n-1)n}{2})) + \frac{(n-2)(n-1)(2n-3)}{p}}$$

$$\begin{aligned}\text{Efficiency} &= \frac{T_s}{pT_p} \\ &= \frac{n(n+1)(2n+1) + 12n(n-1)}{18n^2p + 12np(t_s + t_w(n)) - 18np + 6plog(p)(nt_s + t_w(\frac{(n-1)n}{2})) + (n-2)(n-1)(2n-3)}\end{aligned}$$

$$\begin{aligned}\text{Cost} = pT_p &= 3n^2p + 2np(t_s + t_w(n)) - 3np + plog(p)(nt_s + t_w(\frac{(n-1)n}{2})) + \frac{(n-2)(n-1)(2n-3)}{6} \\ &= \varTheta(n^2plog(p) + n^3)\end{aligned}$$

If $n = \varOmega(plog(p))$, $pT_p = \varTheta(T_s)$ and the system is cost optimal.

$$\begin{aligned}\text{Overhead Function}(T_o) &= pT_p - T_s \\ &= 3n^2p + 2np(t_s + t_w(n)) - 3np + plog(p)(nt_s + t_w(\frac{(n-1)n}{2})) - 4n^2 + 4n - 1 \\ &= O(n^2plog(p)) = O(W^{\frac{2}{3}}plog(p))\end{aligned}$$

$$\text{Isoefficiency function} = O(p^3(log(p))^3)$$

The following plots and tables present various performance measures for the parallel implementation. It should be noted that the implementation was tested on a machine with an Intel Core i5-8300H - a 4 core multi-processor. Hence, for $p > 4$, p processes were physically mapped to 4 processors only.

p	n = 10	n = 100	n = 500	n = 1000	n = 2000	n = 3000
1	0.000049	0.003562	0.122906	0.706596	4.771281	15.563150
2	0.000098	0.002945	0.098454	0.494983	2.944120	9.321857
3	0.000164	0.003077	0.095256	0.409568	2.489187	7.315904
4	0.000194	0.003679	0.078726	0.380961	2.250240	7.155480
5	0.000257	0.003762	0.093228	0.438014	2.738699	9.122288
6	0.000329	0.004316	0.085395	0.696560	2.887806	10.290178
7	0.000390	0.006526	0.089147	0.699248	3.566102	10.620403
8	0.000498	0.006742	0.139970	0.814681	3.935964	13.004947

Table 1. Execution time in seconds for a system of n linear equations in n variables for different values of p.

p	n = 10	n = 100	n = 500	n = 1000	n = 2000	n = 3000
1	1.000	1.000	1.000	1.000	1.000	1.000
2	0.500	1.210	1.248	1.428	1.621	1.670
3	0.299	1.158	1.290	1.725	1.917	2.127
4	0.253	0.968	1.561	1.855	2.120	2.175
5	0.191	0.947	1.318	1.613	1.742	1.706
6	0.149	0.825	1.439	1.014	1.652	1.512
7	0.126	0.546	1.379	1.011	1.338	1.465
8	0.098	0.528	0.878	0.867	1.212	1.197

Table 2. Speedup for a system of n linear equations in n variables for different values of p.

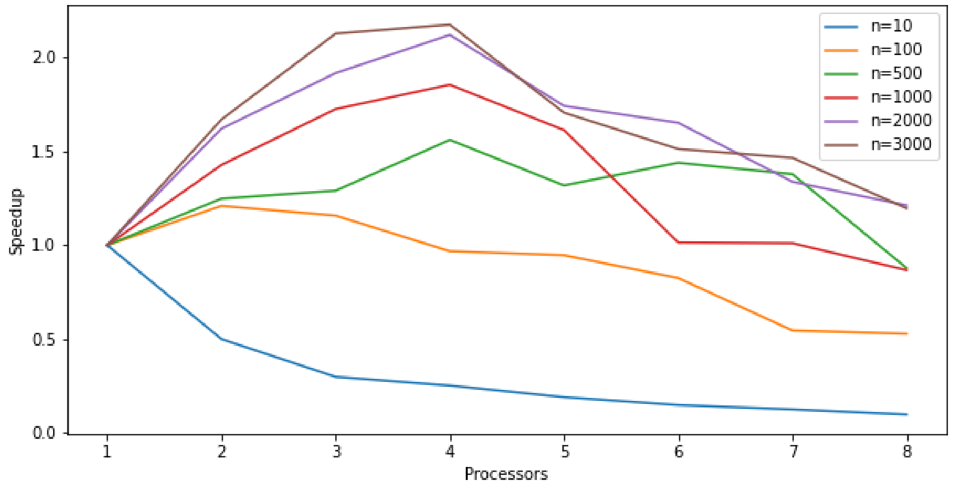

Fig. 3. Speedup plotted against number of processors for different input sizes.

p	n = 10	n = 100	n = 500	n = 1000	n = 2000	n = 3000
1	1.000	1.000	1.000	1.000	1.000	1.000
2	0.250	0.605	0.624	0.714	0.810	0.835
3	0.100	0.386	0.430	0.575	0.639	0.709
4	0.063	0.242	0.390	0.464	0.530	0.544
5	0.038	0.189	0.264	0.323	0.348	0.341
6	0.025	0.138	0.240	0.169	0.275	0.252
7	0.018	0.078	0.197	0.144	0.191	0.209
8	0.012	0.066	0.110	0.108	0.152	0.150

Table 3. Efficiency for a system of n linear equations in n variables for different values of p.

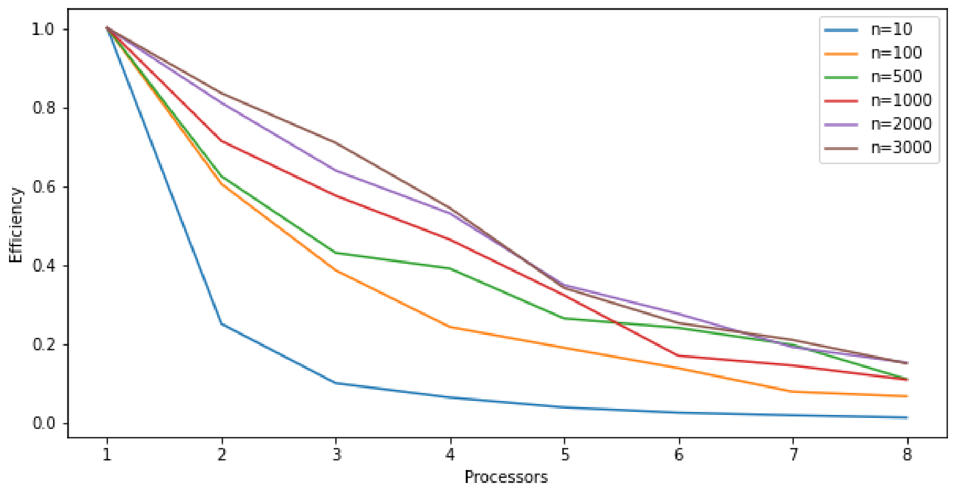

Fig. 4. Efficiency plotted against number of processors for different input sizes.

p	n = 10	n = 100	n = 500	n = 1000	n = 2000	n = 3000
1	0.000	0.004	0.123	0.707	4.771	15.563
2	0.000	0.006	0.197	0.990	5.888	18.644
3	0.000	0.009	0.286	1.229	7.468	21.948
4	0.001	0.015	0.315	1.524	9.001	28.622
5	0.001	0.019	0.466	2.190	13.693	45.611
6	0.002	0.026	0.512	4.179	17.327	61.741
7	0.003	0.046	0.624	4.895	24.963	74.343
8	0.004	0.054	1.120	6.517	31.488	104.040

Table 4. Cost for a system of n linear equations in n variables for different values of p.

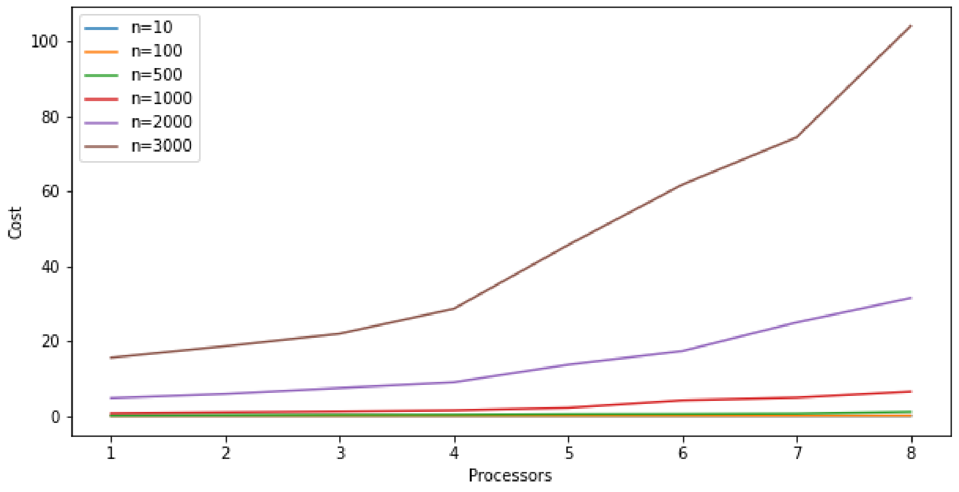

Fig. 5. Cost plotted against number of processors for different input sizes.