CZ4102 – High Performance Computing

Lecture 12 (Last):

Parallel Algorithms for Solving a System of Linear Equations

- Dr Tay Seng Chuan

Reference:

"Introduction to Parallel Computing" – Chapter 8.

1

Topic Overview

- Algorithm for Solving a System of Linear Equations
- Parallel Versions of the Algorithm (1D Partition)
- Performance Analysis

2

Solving a System of Linear Equations

- Consider the problem of solving linear equations of this kind:

$$\begin{array}{ccccccccc} a_{0,0}x_0 & + & a_{0,1}x_1 & + & \cdots & + & a_{0,n-1}x_{n-1} & = & b_0, \\ a_{1,0}x_0 & + & a_{1,1}x_1 & + & \cdots & + & a_{1,n-1}x_{n-1} & = & b_1, \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{n-1,0}x_0 & + & a_{n-1,1}x_1 & + & \cdots & + & a_{n-1,n-1}x_{n-1} & = & b_{n-1}. \end{array}$$

- This is written as $Ax = b$, where A is an n x n matrix with $A[i, j] = a_{i,j}$, b is an n x 1 vector [$b_0, b_1, \ldots, b_n$]^T, and x is the solution.

3

Solving a System of Linear Equations

Two steps in solution are: reduction to triangular form, and back-substitution. The triangular form is as:

$$\begin{array}{ccccccccccc} x_0 & + & u_{0,1}x_1 & + & u_{0,2}x_2 & + & \cdots & + & u_{0,n-1}x_{n-1} & = & y_0, \\ & & x_1 & + & u_{1,2}x_2 & + & \cdots & + & u_{1,n-1}x_{n-1} & = & y_1, \\ & & & & & & & & \vdots & & \vdots \\ & & & & & & & & x_{n-1} & = & y_{n-1}. \end{array}$$

We write this as: $Ux = y$.

A commonly used method for transforming a given matrix into an upper-triangular matrix is **Gaussian Elimination**.

4

Illustration

Given

```
2x + 3y +  z = 6     (1)
3x + 2y + 4z = 9     (2)
4x +  y + 3z = 8     (3)
```

We work on the 1st column first.

```
(1)/2          x  + 1.5y + 0.5z = 3     (1)
               3x +   2y +   4z = 9     (2)
               4x +    y +   3z = 8     (3)

                x + 1.5y + 0.5z = 3     (1)
(2) - (1)x 3    0 - 2.5y + 2.5z = 0     (2)
(3) - (1)x 4    0 -   5y +    z = -4    (3)
```


We have these equations at the end of 1st iteration.

```
x + 1.5y + 0.5z = 3     (1)
  - 2.5y + 2.5z = 0     (2)
    - 5y +    z = -4    (3)
```

We proceed with the 2nd iteration to work on the 2nd column.

```
               x + 1.5y + 0.5z = 3     (1)
(2)/(-2.5)            y -    z = 0     (2)
                   - 5y +    z = -4    (3)

               x + 1.5y + 0.5z = 3     (1)
                      y -    z = 0     (2)
(3) - (2)x(-5)        0 -   4z = -4    (3)
```

We have these equation at the end of 2nd iteration.

```
x + 1.5y + 0.5z = 3     (1)
       y -    z = 0     (2)
           - 4z = -4    (3)
```

We proceed with the 3rd iteration to work on the 3rd column.

```
          x + 1.5y + 0.5z = 3     (1)
                 y -    z = 0     (2)
(3)/(-4)                z = 1     (3)
```

We can now do a back substitution to solve for the values of y and x.

Gaussian Elimination

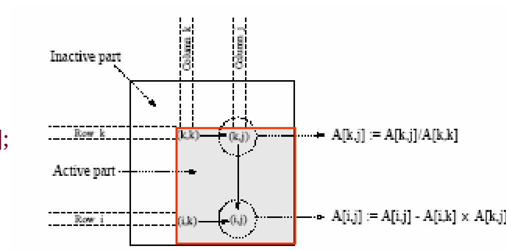
```
1.  procedure GAUSSIAN_ELIMINATION (A, b, y)
2.  begin
3.    for k := 0 to n - 1 do          /* Outer loop */
4.    begin
5.      for j := k + 1 to n - 1 do
6.        A[k, j] := A[k, j]/A[k, k];  /* Division step */
7.      y[k] := b[k]/A[k, k];
8.      A[k, k] := 1;
9.      for i := k + 1 to n - 1 do
10.     begin
11.       for j := k + 1 to n - 1 do
12.         A[i, j] := A[i, j] - A[i, k] x A[k, j]; /* Elimination step */
13.       b[i] := b[i] - A[i, k] x y[k];
14.       A[i, k] := 0;
15.     endfor;          /* Line 9 */
16.   endfor;            /* Line 3 */
17. end GAUSSIAN_ELIMINATION
```

```
(1)/2   x + 1.5y + 0.5z = 3
       3x +   2y +   4z = 9
       4x +    y +   3z = 8
```

```
               x + 1.5y + 0.5z = 3     (1)
(2) - (1)x 3   0 - 2.5y + 2.5z = 0     (2)
(3) - (1)x 4   0 -   5y +    z = -4    (3)
```

Gaussian Elimination

```
1.  procedure GAUSSIAN_ELIMINATION (A, b, y)
2.  begin
3.     for k := 0 to n - 1 do          /* Outer loop */
4.     begin
5.        for j := k + 1 to n - 1 do
6.           A[k, j] := A[k, j]/A[k, k];  /* Division step */
7.        y[k] := b[k]/A[k, k];
8.       A[k, k] := 1;
9.        for i := k + 1 to n - 1 do
10.        begin
11.          for j := k + 1 to n - 1 do
12.             A[i, j] := A[i, j] - A[i, k] x A[k, j];
13.          b[i] := b[i] - A[i, k] x y[k];
14.          A[i, k] := 0;
15.        endfor;          /* Line 9 */
16.    endfor;              /* Line 3 */
17. end GAUSSIAN_ELIMINATION
```



9

Gaussian Elimination

$$
\begin{array}{lllllll}
a_{0,0}x_0 & + & a_{0,1}x_1 & + & \cdots & + & a_{0,n-1}x_{n-1} = b_0, \\
 & + & a_{1,1}x_1 & + & \cdots & + & a_{1,n-1}x_{n-1} = b_1, \\
 & & & & & & \vdots \qquad \vdots \\
 & & & & & + & a_{n-1,n-1}x_{n-1} = b_{n-1}.
\end{array}
$$

```
1.  procedure GAUSSIAN_ELIMINATION (A, b, y)
2.  begin
3.     for k := 0 to n - 1 do          /* Outer loop */
4.     begin
5.        for j := k + 1 to n - 1 do
6.           A[k, j] := A[k, j]/A[k, k];  /* Division step */
7.        y[k] := b[k]/A[k, k];
8.       A[k, k] := 1;
9.        for i := k + 1 to n - 1 do
10.        begin
11.          for j := k + 1 to n - 1 do
12.             A[i, j] := A[i, j] - A[i, k] x A[k, j]; /* Elim */
13.          b[i] := b[i] - A[i, k] x y[k];
14.          A[i, k] := 0;
15.        endfor;          /* Line 9 */
16.    endfor;              /* Line 3 */
17. end GAUSSIAN_ELIMINATION
```

- Gaussian elimination involves approximately $n^2/2$ divisions (line 6).
- The number of subtractions and multiplications is $(n-1)^2 + (n-2)^2 + .. + 2^2 + 1$ in line 12.
- Given that $\sum_{r=1}^{n} r^2 = n(n+1)(2n+1)/6$.

  The number of subtractions and multiplications is approximately $n^3/3 - n^2/2$. (ignore n and below)
- Assume that each scalar arithmetic operation takes unit time. With this assumption, the sequential run time of the procedure is approximately

  $n^2/2 + 2(n^3/3 - n^2/2)$

  $= 2n^3/3$ (for large n).

multiply and substract

Parallel Gaussian Elimination

We work on the 1st column first.

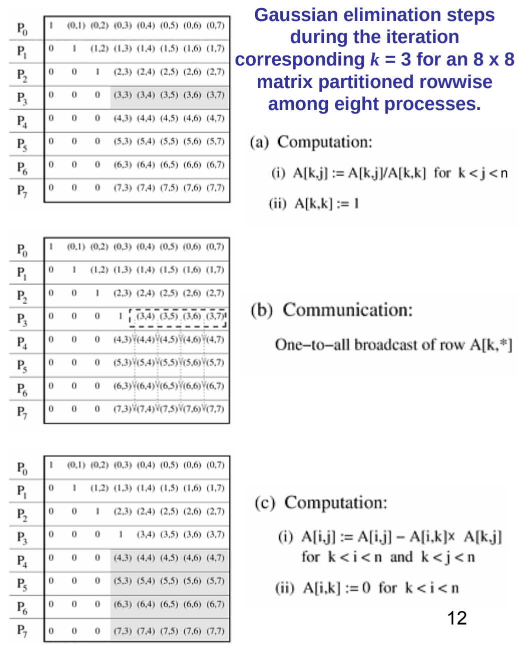
```
(1)/2          x + 1.5y + 0.5z = 3     (1)
              3x +   2y +   4z = 9     (2)
              4x +    y +   3z = 8     (3)
```

Once the **normalization** is done on a row, the **elimination** done on the subsequent rows can proceed in parallel.

Parallel Gaussian Elimination

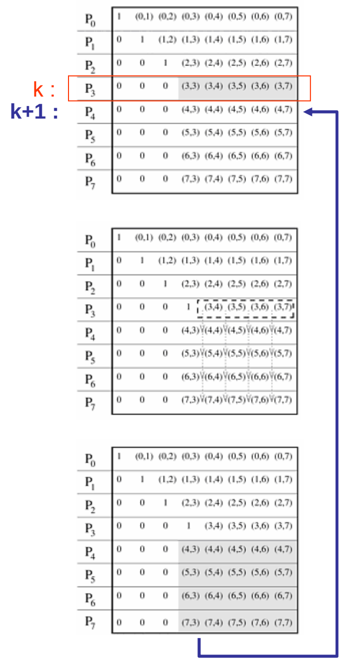
- Assume $p = n$ with each row assigned to a processor.
- The first step of the algorithm normalizes the row. This is a serial operation and takes time $(n-k)$ in the kth iteration.
- In the second step, the normalized row is broadcast to all the processors. This takes time $(t_s + t_w(n-k-1))\log n$
- Each processor can independently eliminate this row from its own. This requires $(n-k-1)$ multiplications and subtractions.
- The total parallel time can be computed by summing from $k = 1 \ldots n-1$, giving $T_P = \frac{3}{2}n(n-1) + t_s n \log n + \frac{1}{2} t_w n(n-1)\log n.$
- The formulation with process-time product of $O(n^3 \log n)$ is not cost optimal because of the t_w term.

Gaussian elimination steps during the iteration corresponding $k = 3$ for an 8 x 8 matrix partitioned rowwise among eight processes.



Parallel Gaussian Elimination: Pipelined Execution

- In the previous formulation, the (k+1)st iteration starts only after all the computation and communication for the kth iteration is complete.
- In the pipelined version, there are three steps - normalization of a row, communication, and elimination. These steps are performed in an asynchronous fashion.
- A processor P_k waits to receive and eliminate all rows prior to k.
- Once it has done this, it forwards its own row to processor P_{k+1}. No waiting.

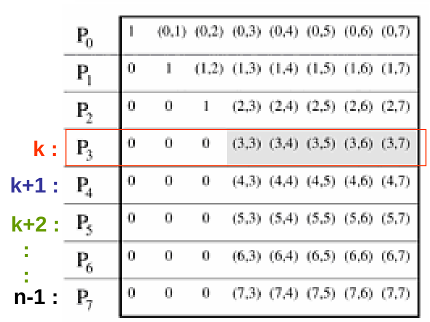


13

Parallel Gaussian Elimination: Pipelined Execution

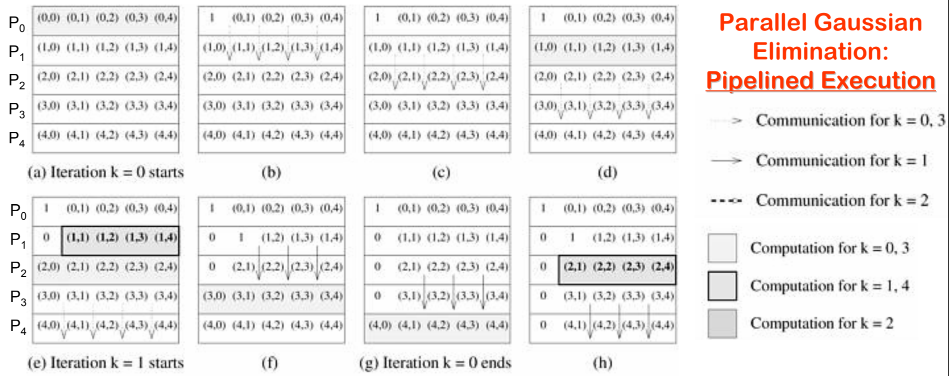
- Assuming that the processes form a logical linear array, and P_{k+1} is the first process to receive the kth row from process P_k. Then process P_{k+1} must forward this data to P_{k+2}.
- However, after forwarding the kth row to P_{k+2}, process P_{k+1} needs not wait to perform the elimination step (line 12) until all the processes up to P_{n-1} have received the kth row.
- Similarly, P_{k+2} can start its computation as soon as it has forwarded the kth row to P_{k+3}, and so on. Meanwhile, after completing the computation for the kth iteration, P_{k+1} can perform the division step (line 6), and start the broadcast of the (k + 1)th row by sending it to P_{k+2}.

```
1.  procedure GAUSSIAN_ELIMINATION (A, b, y)
2.  begin
3.     for k := 0 to n - 1 do                    /* Outer loop */
4.     begin
5.        for j := k + 1 to n - 1 do
6.           A[k, j] := A[k, j]/A[k, k];   /* Division step */
7.        y[k] := b[k]/A[k, k];
8.        A[k, k] := 1;
9.        for i := k + 1 to n - 1 do
10.       begin
11.          for j := k + 1 to n - 1 do
12.             A[i, j] := A[i, j] - A[i, k] × A[k, j];
13.          b[i] := b[i] - A[i, k] × y[k];
14.          A[i, k] := 0;
15.       endfor;          /* Line 9 */
16.    endfor;             /* Line 3 */
17. end GAUSSIAN_ELIMINATION
```



14

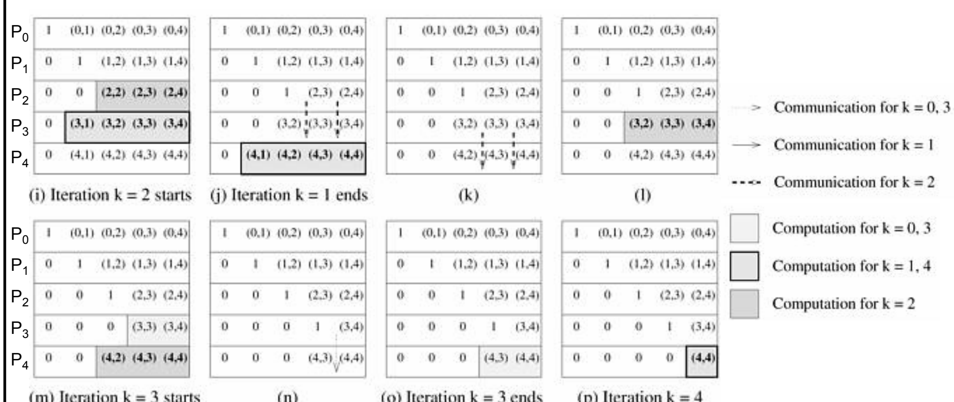
Parallel Gaussian Elimination: Pipelined Execution



- The first step (Figure a) is to perform the division on row 0 at process P_0. The modified row 0 is then sent to P_1 (Figure b), which forwards it to P_2 (Figure c).
- Now P_1 is free to perform the elimination step using row 0 (Figure d).
- In the next step (Figure e), P_2 performs the elimination step using row 0. In the same step, P_1, having finished its computation for iteration 0, starts the division step of iteration 1.
- At any given time, different stages of the same iteration can be active on different processes. For instance, in Figure h, process P_2 performs the elimination step of iteration 1 while processes P_3 and P_4 are engaged in communication for the same iteration.

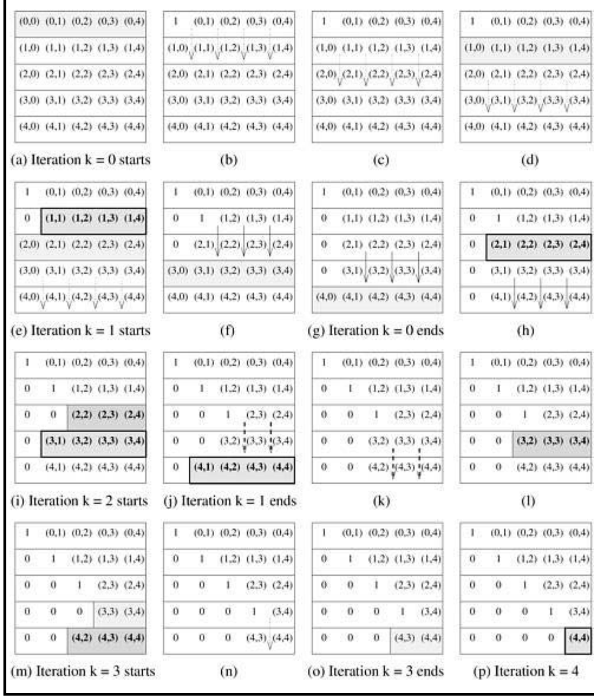
15

(P.T.O)



- Furthermore, more than one iteration may be active simultaneously on different processes. For instance, in Figure (i), process P_2 is performing the division step of iteration 2 while process P_3 is performing the elimination step of iteration 1.

16

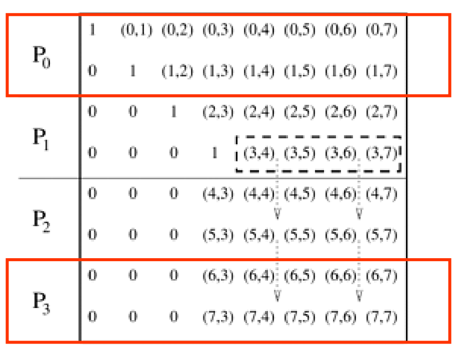


Parallel Gaussian Elimination: Pipelined Execution

- The total number of steps in the entire pipelined procedure is $O(n)$.
- In any step, either $O(n)$ elements are communicated between directly-connected processes, or a division step is performed on $O(n)$ elements of a row, or an elimination step is performed on $O(n)$ elements of a row.
- There are n equations (n rows). The parallel time is therefore of $O(n^2)$.
- This is cost optimal.

17

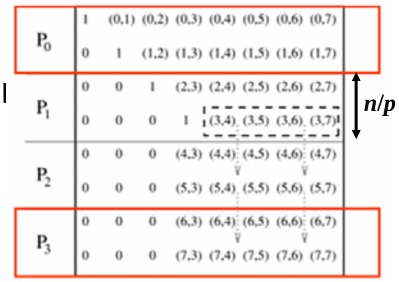
Parallel Gaussian Elimination using 1D Horizontal Block with $p < n$



The communication in the Gaussian elimination iteration corresponding to $k = 3$ for an 8 x 8 matrix distributed among four processes using **1-D block** partitioning.

18

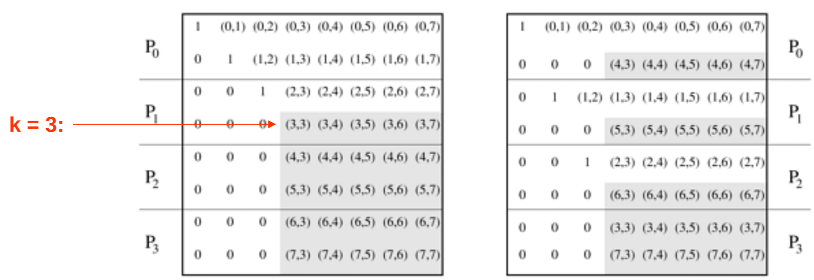
Parallel Gaussian Elimination (Pipelined Execution): 1D Block with $p < n$



- The above algorithms can be easily adapted to the case when $p < n$.
- In the kth iteration, a processor with all rows belonging to the active part of the matrix performs $(n - k - 1)\, n/p$ multiplications and subtractions.
- In the pipelined version, for $n > p$, computation dominates communication.
- The parallel time is given by: $2(n/p)\Sigma_{k=0}^{n-1}(n-k-1)$ or approximately, n^3/p.
- While the algorithm is cost optimal in term of order, the cost of the parallel algorithm is higher than the sequential run time of Gaussian Elimination ($2n^3/3$) by a factor of 3/2 due to uneven workload distribution (P.T.O.).

19

Parallel Gaussian Elimination: 1D Block with $p < n$ (Uneven Workload Distribution)



(a) Block 1-D mapping (b) Cyclic 1-D mapping

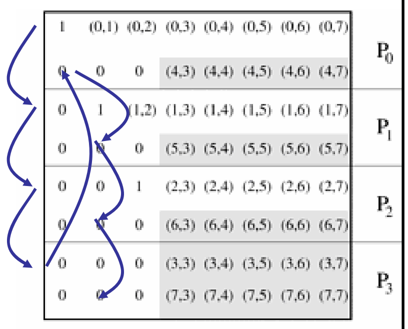
Computation load on different processes in block and cyclic 1-D partitioning of an 8 x 8 matrix on four processes during the Gaussian elimination iteration corresponding to k = 3.

For example, during the iteration corresponding to k = 3 (see figure (a)), one process is completely idle (P_0), one is partially loaded (P_1), and only two processes (P_2 and P_3) are fully active. By the time half of the iterations of the outer loop are over, only half the processes are active. The remaining idle processes make the parallel algorithm costlier than the sequential algorithm. The solution is cyclic mapping (see figure (b)). **(P.T.O.)**

20

Parallel Gaussian Elimination: 1D Block with *p* < *n* (Uneven Workload Distribution)

- The load imbalance problem can be handled (but not completely solved) by using a cyclic mapping where the row assignment to process is on a round robin basis.
- In this case, other than processing of the last p rows, there is no idle process. The largest work imbalance is not more than 1 row in all processes for k = 0 to n-1-p.
- This corresponds to a reduced cumulative idle time of $O(n^2)$ x p = $O(n^2 p)$ (instead of $O(n^3)$ in the previous case).



(b) Cyclic 1-D mapping

21

Solving a Triangular System: Back-Substitution

- In the second phase to solve the equations, the upper triangular matrix U undergoes back-substitution to determine the vector x.

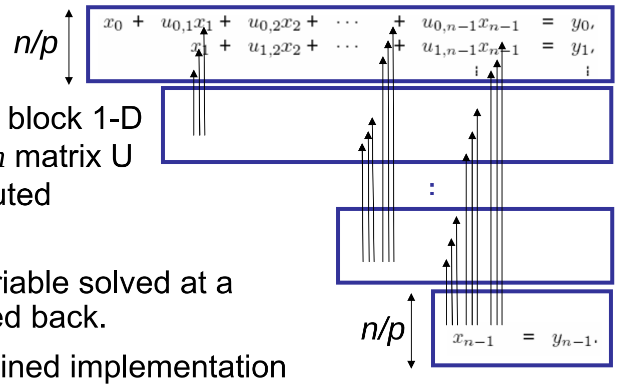
$$
\begin{array}{rcl}
x_0 + u_{0,1}x_1 + u_{0,2}x_2 + \cdots + u_{0,n-1}x_{n-1} & = & y_0, \\
x_1 + u_{1,2}x_2 + \cdots + u_{1,n-1}x_{n-1} & = & y_1, \\
\vdots & & \vdots \\
x_{n-1} & = & y_{n-1}.
\end{array}
$$

```
procedure BACK_SUBSTITUTION (U, x, y)
begin
   for k := n − 1 downto 0 do    /* Main loop */
      begin
         x[k] := y[k];
         for i := k − 1 downto 0 do
            y[i] := y[i] − x[k] × U[i, k];
      endfor;
end BACK_SUBSTITUTION
```

The serial algorithm performs approximately $n^2/2$ multiplications and subtractions.

22

Solving a Triangular System: Back-Substitution (Parallel Version)



- Consider a rowwise block 1-D mapping of the n x n matrix U with vector y distributed uniformly.
- The value of the variable solved at a step can be pipelined back.
- Each step of a pipelined implementation requires a constant amount of time for communication, and $\Theta(n/p)$ time for computation.
- The parallel run time of the entire algorithm is $O(n^2/p)$, or $O(n)$ if $p = n$.

23

2-D Partition for Parallel Gaussian Elimination Method produces fine granularity so it is not promising for implementation.

We have completed the course. All the best.

24