Laboratory for Parallel Programming
Prof. Dr. Felix Wolf

TECHNISCHE UNIVERSITÄT DARMSTADT

Aamer Shah, shah@cs.tu-darmstadt.de

Large-Scale Parallel Computing (WS 15/16)

Exercise 6

This is a hands on exercise. The solution will be developed by the tutor and the students together in the class on February 2nd, 2016. However, the students are encouraged to solve the problem before attending the session. Skeleton code of the tasks is provided along with the task sheet, and can also be copied from `/home/as65huly/public/ex06.tgz` on the Lichtenberg cluster.

Task 1

In this task, we will revisit the matrix multiplication problem, discussed in Exercise 3. The main idea was to divide the workload among the processes in such a way that each process calculates the product of a tile of the resulting solution matrix. In Exercise 3, in the first parallel version, the column was distributed element-by-element as it is not contiguous in memory. In the second version, the second matrix was stored as a transpose to address this problem.
In this task, we will use `MPI_Datatype`s to distribute the rows and columns, without the need to store the matrix as a transpose. Similarly, the final product tile calculated by the processes will also be collected together at the root using `MPI_Datatype`s.
In the task, assume that the number of processes will always be a squared value (4, 16 or 64), and that the matrices will be distributed equally along rows and columns (the same number of tiles in both direction). A skeleton code is provided for the task in the accompanying archive (`matmul_dist.c`). Perform the task in the following steps:

a) Create datatypes for matrix rows, columns and tiles. For simplicity, create the datatypes with extent large enough to transfer the rows/columns/tiles in one communication call.

b) Distribute the rows among processes. Can this be done through a collective call or should a point-to-point communication mechanism be used?

c) Distribute the columns among processes. Can this be done through a collective call or should a point-to-point communication mechanism be used?

d) Each process calculates the product

e) Collect all the tiles at the root process. Can this be done through a collective call or should a point-to-point communication mechanism be used?

f) Free up the created datatypes

After completing the code, do the following:

a) Set the `DIM_LEN` to `16`.

b) Run the code for 4, 16 and 64 process counts and print the result matrix.

c) For correction, compare the matrices and see if the result is the same.

d) Remove the matrix printing, set the `DIM_LEN` to `64` and run it again for 4, 16 and 64 process counts.

e) Record the total execution time

Task 2

In this task, the solution of Task 1 will be extended to use non-blocking point-to-point communication to distribute rows and columns. To keep the task short, each process's product tile will be collected using the same method as Task 1.

The task will use an element-wise matrix multiplication algorithm in which computation can be overlapped with communication. First, the datatypes of rows and columns should be modified to cover only two rows and two columns respectively. The datatype of product tile remains the same as it is collected using the same method as Task 1. Then, each process (other than root) first receives a row datatype and then a column data type. For each received row/column, the process calculates the element-wise product of the received data. After calculating the product, the process then receives a new row datatype and a new column datatype. The following presents the algorithm:

Algorithm 1 Element-wise multiplication algorithm

```
1:  procedure ELEMENT-WISE MULTIPLICATION
2:      //dim_count is the number of row datatypes or column datatypes each process receives. They
    have the same value as the number of processes is always a square.
3:      for i = 0 to dim_count do
4:          Receive row[i]
5:          if i > 0 then //If there is already a column received
6:              //Multiplication can be done using a single call to mat_mul function as rows are contigu-
    ous in memory
7:              multiply row[i] with columns[ 0 to i − 1], storing result in product[i][0 to i − 1]
8:          Receive column[i]
9:          //Use a for loop as columns are not contiguous in memory
10:         for j = 1 to i do
11:             multiply column[i] with rows[j], storing result in product[j][i]
```

The accompanying skeleton code already implements the multiplication algorithm, with marked places left for communication. Perform the task in the following steps:

a) Create row, column and tile datatypes. The row and column datatypes should cover 2 rows and 2 columns respectively

b) Rank 0 sends all the rows and columns to other processes using non-blocking communication

c) Rank 0 calculates its product after sending

d) All other processes initiate non-blocking receives for row and column datatypes

e) The other processes then use the element-wise multiplication algorithm to receive row/column datatypes and to calculate the product.

f) Rank 0 waits for all the sends to complete

g) The product is collected at root process using the same method as Task 1.

After completing the code, do the following:

a) Set the `DIM_LEN` to `16`.

b) Run the code for 4, 16 and 64 process counts and print the result matrix.

c) For correction, compare the matrices and see if the result is the same.

d) Remove the matrix printing, set the `DIM_LEN` to `64` and run it again for 4, 16 and 64 process counts.

e) Record the total execution time and compare it with Task 1. Did we achieve better performance?