Parallel Matrix by Matrix Multiplication using MPI

Janko Strassburg
[janko.strassburg@bsc.es]

PRACE Training @ BSC, November 2012

1 Matrix-Matrix Multiplication

Multiplication of two matrices is well-defined only if the number of columns n of the first matrix A is the same as the number of rows n of the second matrix B. If A is an m-by-n matrix (m rows, n columns) and B is an n-by-p matrix (n rows, p columns), then their product AB is a matrix C with dimension m-by-p (m rows, p columns).

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix} \text{ and } B = \begin{pmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{np} \end{pmatrix}$$

$$AB = C = \begin{pmatrix} c_{11} & \cdots & c_{1p} \\ \vdots & \ddots & \vdots \\ c_{m1} & \cdots & c_{mp} \end{pmatrix}$$

Each element c_{ij} of the product matrix $C = AB$ can be calculated as the scalar product $c_{ij} = \vec{a_i}^T \cdot \vec{b_j}$ with the row vector $\vec{a_i} = (a_{i1}, a_{i2}, \dots, a_{in})$ and the column vector $\vec{b_j} = (b_{1j}, b_{2j}, \dots, b_{nj})^T$.
Keeping in mind that the definition of the scalar product is given by

$$s = \vec{x} \cdot \vec{y} = (x_1, \dots, x_n) \cdot \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \sum_{i=1}^{n} x_i y_i$$

we can derive the definition of the matrix product $C = AB$ as

$$c_{ij} = \sum_{k=1}^{r} a_{ik} b_{kj} \text{ with } i = 1, \dots, m \text{ and } j = 1, \dots, p$$

2 Hands On

1. Finalize a given example in C (or write your own version), to run on MPI for parallel matrix multiplication of two integer matrices A(n x n) and B(n x n) on a ring of r processors.

(a) Assume that your architecture is a ring with 2, 4, 6 and 8, etc... processors.

- Test the program on square matrices A and B of sizes (6 x 6), (8 x 8) and (12 x 12)
- Read the matrices from a file and save the results in a file (the format is given in 3.1).

(b) Test your program on larger matrices of different compatible sizes and compare the speed (in milliseconds) for different variations of matrix sizes and number of processors

- Use matrix size 250x250, 500x500, 1000x1000, 2000x2000, ...

(c) Pay attention to and analyse speedup as well as efficiency of the program execution.

3 Guidelines

3.1 Matrix File Format

The matrix files are to be in the following format. The first line of the file contains an integer which stands for the dimension n of the n-by-n matrix. The following n lines contain the n rows of the matrix.

eg.: having a 4-by-4 matrix

$$\begin{array}{cccc} 4 & & & \\ 1 & 1 & 7 & 23 \\ 3 & 6 & 7 & 11 \\ 77 & 1 & 0 & 1 \\ 2 & 0 & 34 & 1 \end{array}$$

3.2 Test Your Implementation

To test your implementation you can use the following property. The result of the multiplication of matrix A with the identity matrix I is matrix A again.

$AI = A$

An identity matrix is a matrix with ones in the diagonal and zeros for the rest of the elements.
eg.: a 4-by-4 identity matrix

$$I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Make sure that you test your code with more than 2 processors, since misdirected communication could be hidden.

4 Possible Parallel Algorithm

4.1 Algorithm

1. read matrices A and B from file
2. decision: is it possible to calculate with given number of processors?
3. calculate the size of the partitions
4. broadcast dimension of A and B to all slaves
5. send the appropriate rows of A and columns of B to each slave
6. compute parts of C
7. shift rows of A between slaves
8. collect all calculated parts of C from all slaves
9. save matrix C into a file

4.2 Example

Lets assume that we have 2 processors computing matrix C. Processor 1 would have to compute column 1 and 2 of matrix C. This will be done in two steps, STEP1 and STEP2. The master has to send the first two rows of A and the first two columns of B to the first processor and the rest of A and B to the second processor. After the processors computed STEP1 they will send it's part of A to the next processor in a ring which means that in the case of 2 processors they just exchange the parts. Then they can compute STEP2 and send the results back to the master.

Given matrix A and B.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{pmatrix}$$

We can compute C which is result of multiplication of A and B.

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{pmatrix}$$

$$C_{ij} = \sum_{k=1}^{r} A_{ik} B_{kj} \text{ with } i = 1, \ldots, p \text{ and } j = 1, \ldots, q$$

For the first column of C we have to calculate the following equations:

$$c_{11} = a_{11} \cdot b_{11} + a_{12} \cdot b_{21} + a_{13} \cdot b_{31} + a_{14} \cdot b_{41}$$
$$c_{21} = a_{21} \cdot b_{11} + a_{22} \cdot b_{21} + a_{23} \cdot b_{31} + a_{24} \cdot b_{41}$$
$$c_{31} = a_{31} \cdot b_{11} + a_{32} \cdot b_{21} + a_{33} \cdot b_{31} + a_{34} \cdot b_{41}$$
$$c_{41} = a_{41} \cdot b_{11} + a_{42} \cdot b_{21} + a_{43} \cdot b_{31} + a_{44} \cdot b_{41}$$

For the second column of C we have to calculate the following equations:

$$c_{12} = a_{11} \cdot b_{12} + a_{12} \cdot b_{22} + a_{13} \cdot b_{32} + a_{14} \cdot b_{42}$$
$$c_{22} = a_{21} \cdot b_{12} + a_{22} \cdot b_{22} + a_{23} \cdot b_{32} + a_{24} \cdot b_{42}$$
$$c_{32} = a_{31} \cdot b_{12} + a_{32} \cdot b_{22} + a_{33} \cdot b_{32} + a_{34} \cdot b_{42}$$
$$c_{42} = a_{41} \cdot b_{12} + a_{42} \cdot b_{22} + a_{43} \cdot b_{32} + a_{44} \cdot b_{42}$$

5 MPI Hints

5.1 Useful MPI Functions

Here are some useful functions you could use for solving the problem:

MPI_Init() Initialise the MPI execution environment

MPI_Comm_rank() Determines the rank of the calling process in the communicator

MPI_Comm_size() Determines the size of the group associated with a communicator

MPI_Get_processor_name() Gets the name of the processor

MPI_Bcast() Broadcasts a message from the process with rank "root" to all other processes of the group.

MPI_Scatter() Sends data from one task to all other tasks in a group

MPI_Send() Performs a basic send

MPI_Recv() Basic receive

MPI_Sendrecv_replace() Sends and receives using a single buffer. Note that this should be used while shifting big amount of data, the normal send and receive function can end in a deadlock situation.

MPI_Gather() Gathers together values from a group of processes

MPI_Finalize() Terminates MPI execution environment

6 References

6.1 MPI

MPI Homepage `http://www-unix.mcs.anl.gov/mpi/`

MPI Manual http://www-unix.mcs.anl.gov/mpi/mpich/index.html

MPI Reference http://www.netlib.org/mpi/index.html

MPI Tutorial http://www-unix.mcs.anl.gov/mpi/tutorial/gropp/talk.html

MPI Quick References http://web.umr.edu/~ercal/387/MPI/qref.html and http://www.georgetown.edu/research/arc/mpi-quick-ref.pdf

OpenMPI Homepage http://www.open-mpi.org/