Parallel computing contest: Problem A

Carlos Hoyos Barceló - karlos1982@hotmail.com
Computer Science Faculty - University of Murcia, Spain

ABSTRACT. The purpose of this document is to describe the method used to solve problem A featured in the first Spanish Parallel Computing Contest. This problem is a matrix multiplication in which matrices have one or more sub-matrices with all their values set to zero. The program makes use of the matrices' structure to reduce the execution time; it uses both OpenMP and MPI, and applies some optimizations to the basic algorithm to achieve a speed-up up to two orders of magnitude faster than the sequential program.

KEYWORDS: parallel computing, parallel algorithms, MPI, OpenMP.

1. PROBLEM STATEMENT

Given two square matrices A and B with 'holes' (sub-matrices with all their values set to zero) that may overlap, find the product C = A*B. For example, the following matrix:

$$\begin{Bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 0 & 0 & 0 & 2 & 3 \\ 1 & 0 & 0 & 0 & 2 & 3 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

has a 'hole' of size 3x3 starting at the coordinates [1,1], another one sized 4x3 starting at position [2,3], and one small 1x1 hole at position [0,5].

The problem can be solved by straight matrix multiplication, but it is possible to take advantage of the structure of the matrices to reduce the number of required operations. A sequential algorithm is provided, and the task is to parallelize it.

2. SOLVING THE PROBLEM

2.1 SEQUENTIAL ALGORITHM OPTIMIZATIONS

The solution applies the following optimizations to the basic algorithm:

- Transposing matrix B before the multiplication. This step greatly reduces the execution time for most cases, but specially so for dense matrices (<45% zeros) or with small 'hole' size (<2.5% the size of the matrix). The average speed-up provided by this optimization alone was 3.86x in the tests. Transposing the matrix results in a worse execution time in a few corner cases. The example ran by the judge program behaves this way, so this optimization can't be applied to it. Therefore, the speed-up registered in the contest doesn't reflect the maximum potential speed-up achievable by this algorithm.
- Tweaks to the inner loops. The algorithm avoids calculating index values unless required for the current iteration. It also performs an intelligent iteration of the indexes of Matrix B, so it only checks those blocks that are within the range of the current block of A in the loop. The average speed-up of this optimization is 3.75x, and it gets better in the case of highly fragmented matrices.

2.2 Parallelizing the algorithm

The partition of matrix A is done at row level. However, instead of assigning the same number of rows to each node, the algorithm assigns a number of rows such that the number of non-zero elements in each portion is roughly the same for each node. Thus, each node gets more or less the same amount of work.

$$\left\{\begin{matrix} 1&2&3&4&5&6 \\ 1&0&0&0&2&3 \\ 1&0&0&0&2&3 \\ 1&0&0&0&0&0 \\ 1&2&0&0&0&0 \\ 0&1&0&0&0&0 \end{matrix}\right\} \qquad \left\{\begin{matrix} 1&2&3&4&5&6 \\ 1&0&0&0&2&3 \\ 1&0&0&0&2&3 \\ 1&0&0&0&0&0 \\ 1&2&0&0&0&0 \\ 0&1&0&0&0&0 \end{matrix}\right\}$$

■: rows assigned to node 0
■: rows assigned to node 1
■: rows assigned to node 2

In the example, assigning the same number of rows to each node leads to a partition of [9, 4, 4] elements. The method used by the algorithm makes a partition of [4, 6, 6] elements. The last portion is typically smaller than the others, so the algorithm assigns it to node 0 to balance the extra work it does in managing the communications.

2.3 Optimization of the communications

All nodes receive a copy of matrix B, their corresponding portion of matrix A and the structure Filas_A. If the copies of matrix B are sent to the other nodes exclusively by the master node, the communication takes O(n) steps. This algorithm performs a tree distribution, so in each step not only the master node, but also every other node that already has a copy of B, sends copies to the other nodes. This completes the communication of matrix B in O(log n) steps.

The master node does not send the structure Columnas_B to the other nodes; that is created by each node to reduce the amount of communications. The master node uses OpenMP to reduce the work of transposing matrix B and creating the structure Columnas_A before sending them.

The algorithm uses MPI only for inter-node communication. For cores inside the same node it uses OpenMP, to take advantage of shared memory. Another optimization is to use N+1 threads for a node with N cores, as it leaves one thread with the single purpose of creating tasks, while all the cores can be used simultaneously to complete them.

3. Final remarks

* This algorithm could be made slightly faster by using a different "formarestructura" method for Columnas_B that does not count the number of non-zero elements in each row.
* Communications would be faster if the algorithm made use of a compressed matrix structure to represent matrices A and B.
* In some systems, broadcasting takes O(1) steps. In these systems it is better not to use the "tree distribution" optimization.
* Transposing matrix B makes the judge erroneously mark the result as invalid.