OpenMP Implementation of Gauss Jordan

Gabriel Dimitriu

1 Introduction

Let

$$Ax = b \tag{1}$$

be our system of linear equations, with $A = (a_{ij})_{i,j \in \{1,...,m\}}$, $b = (b_1, ..., b_m)$ and $x = (x_1, ..., x_m)^T$.

Algorithm 1 *Gauss-Jordan*

```
1.  procedure gauss_jordan
2.  for k=0 to m do
3.      for i=k+1 to m do
4.          for j=k+1 to m do
5.              $a_{ij} = a_{ij} - a_{ik} * a_{kj} / a_{kk}$
6.          endfor
7.          $b_i = b_i - a_{ik} * b_k / a_{kk}$
8.      endfor
9.      for i=0 to k do
10.         for j=k+1 to m do
11.             $a_{ij} = a_{ij} - a_{ik} * a_{kj} / a_{kk}$
12.         endfor
13.         $b_i = b_i - a_{ik} * b_k / a_{kk}$
14.     endfor
15. endfor
16. for i=0 to m do
17.     $x_i = b_i / a_{ii}$
18. endfor
19. end procedure
```

2 Pipeline Communication and Computation

The matrix is scattered to all processor so two consecutive row are in two consecutive processors.

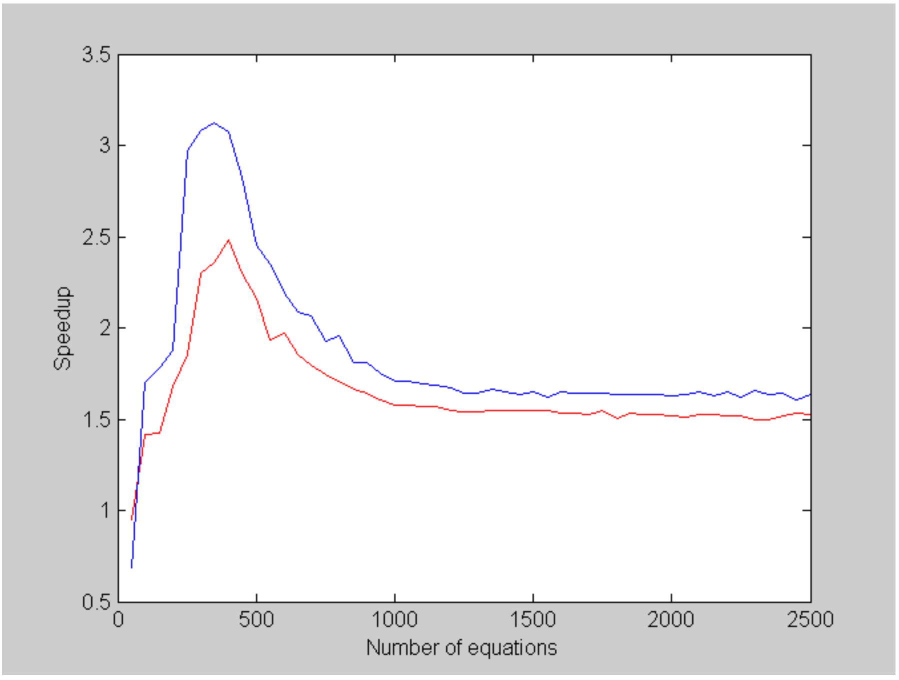
During the k^{th} iteration processor P_k broadcast part of the k^{th} row of the matrix to processors $P_{k+1}, ..., P_{p-1}$. Assume that the processors $P_0...P_{p-1}$ are connected in a linear array, and P_{k+1} is the first processor to receive the k^{th} row from processor P_k. Then the processor P_{k+1} must forward this data to P_{k+2}. However, after forwarding the k^{th} row to P_{k+2}, processor P_{k+1} need not wait to perform the elimination step until all the processors up to P_{p-1} have received the k^{th} row. Similarly, P_{k+2} can start its computation as soon as is has forwarded the k^{th} row to P_{k+3}, and so on. Meanwhile, after completing the computation for the k^{th} iteration, P_{k+1} can perform the division step, and start the broadcast of the $(k+1)^{th}$ row by sending it to P_{k+2}. In this case of shared memory the receive, send and broadcast from MPI are replaced by OpenMP lock procedures but without pipeline communication. The backward implementation is made similar to the forward implementation.

3 Results

I have compile the parallel program with two OpenMP compilers: Omni 1.6 and Intel C Compiler 8.0 for LINUX and the serial with gcc and Intel C Compiler 8.0 for LINUX both with maximum optimization "-O3" and for Intel C Compiler I've put also "-mcpu=pentiumpro -tpp6" for maximum optimization.

The executable were run on a dual pentium II at 500MHz with 256MB RAM and with LINUX Fedora Core 1.

The following results were made for a average or 10 runs for serial and parallel programs and with red is plotted the results from ICC and with blue the results from Omni.


Speedup
Number of equations