Parallel Gaussian Elimination

Rahul Ganesh Prabhu

2018A7PS0193P

1 The Problem Statement

Assuming an input of the augmented matrix of a set of N linear equations A and a system having p processors, implement a parallelized Gaussian Elimination algorithm to solve the given linear equations.

2 Partition

The given problem is partitioned into the following primitive tasks:

- t_{read} : The file is read for input.
- t_{pivot}: The maximum element in a column is found. Rows are exchanged to make it the pivot.
- $t_{distribute}$: The rows of the matrix are distributed to the processors
- t_{reduce}: The rows are reduced with respect to the pivot.
- t_{merge}: The reduced rows are merged back into a full array.
- $t_{subsititute}$: Back substitute and find the results.

Besides t_{read}, these tasks are all performed N times. Out of these tasks, t_{pivot} and $t_{distribute}$ cannot be efficiently parallelized. $t_{subsititute}$ also cannot be efficiently parallelized due to loop carried dependence.

3 Communication

$t_{distribute}$ is performed by the root process performing collective communication via *MPI_Scatter*. Other information is also broadcasted by the root process, like N and the value of the pivot.

t_{merge} is done via centralized communication - every process sends it's reduced rows to the root process. This communication scheme was chosen over a hypercube or a tree structure since they were found to be slower, probably due to the additional overhead of communication and memory allocation.

4 Agglomeration

It is possible that the number of rows N is greater than p. So, we agglomerate t_{reduce} so that the rows are distributed to all the processors evenly. The processor then reduces all the rows given to it and ocmmunicates them to the root process for t_{merge}. It is important to remember that due to the nature of Gaussian elimination, not all rows will be reduced in a given iteration, so over time more and more processors will idle.

5 Mapping

Since t_{read}, t_{pivot} and $t_{distribute}$ cannot be parallelized, they are performed sequentially by the root process. t_{reduce} is performed by every processor, though over time, the load balancing becomes unequal since an unequal number of rows are sent to each processor. t_{merge} is also performed by every processor, though the root processor acts as the reciever while every other processor acts as a sender.

6 Theoretical Speedup

The sequential algorithm involves three nested loops each iterating order of N times, so the running time is $T_{seq}(N) = \mathcal{O}(N^3)$. The parallel algorithm takes N steps. In the i^{th} step, $N - i$ rows are distributed among the p processors, each who perform an $\mathcal{O}(N)$ operation to reduce their rows. So, the running time becomes:

$$
\begin{aligned}
T_{pll}(N) &= \sum_{i=0}^{N} \frac{N-i}{p} \cdot \mathcal{O}(N) \\
T_{pll}(N) &= \frac{\mathcal{O}(N)}{p} \sum_{i=0}^{N} (N-i) \\
T_{pll}(N) &= \frac{\mathcal{O}(N)}{p} \cdot \frac{N \cdot (N+1)}{2} \\
T_{pll}(N) &= \mathcal{O}(\frac{N^3}{p})
\end{aligned}
\tag{1}
$$

The running times calculated above do not consider the running time for $t_{subsititute}$, which will be $\mathcal{O}(N)$

Hence the speedup will be:

$$
\begin{aligned}
Speedup &= \frac{T_{seq}(N)}{T_{pll}(N)} \\
Speedup &= \frac{N^3 + N}{\frac{N^3}{p} + N}
\end{aligned}
\tag{2}
$$

7 Results

The program was run on an input of N=1600. The results were as follows:

- 1 processor: 9.9 seconds
- 2 processors: 6.7 seconds
- 3 processors: 5.5 seconds
- 4 processors: 4.9 seconds

This does not reflect the ideal speedup calculated, but I theorize this is due to extra communication and allocation overheads not considered in the theoretical calculations.