Min Jung, Parth Singh, Jonathan Peña, Shavleen Singh
CS 4230: Social Computing
April 29, 2024

Group Assignment # 1

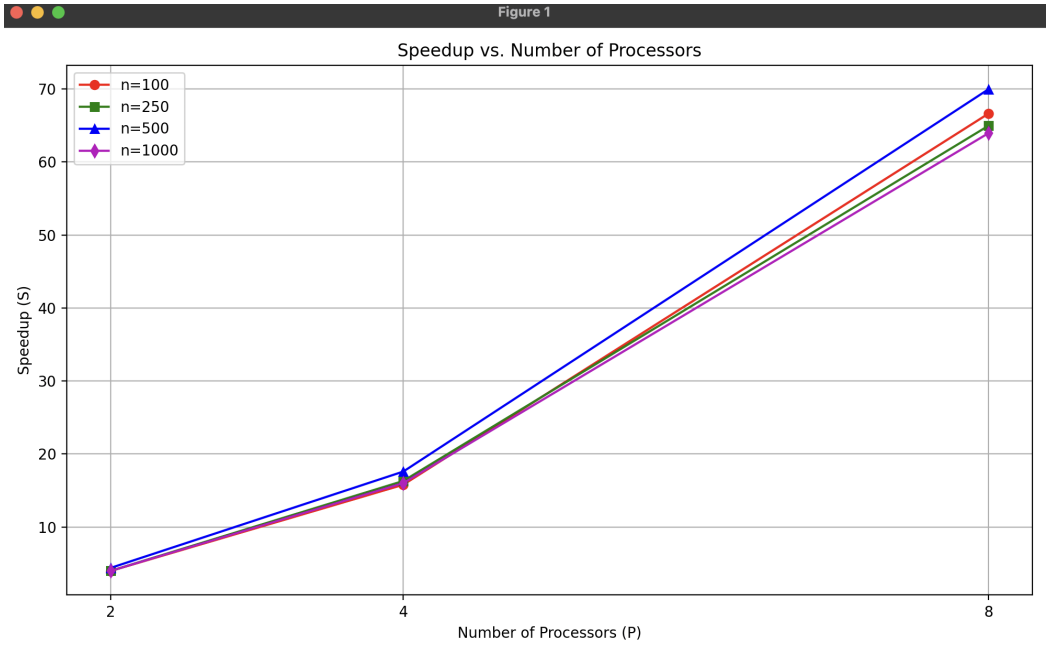
Problem 1:

Results:

Run time(secs)	P =1	P =2	P =4	P=8
n=100	3.546984	0.889845	0.224564	0.053233
n=250	47.417794	11.885332	2.907980	0.729848
n=500	405.365260	95.574428	23.901060	5.971873
n=1000	3575.580493	895.630170	223.260976	55.911144

Speedup (S)	P=2	P=4	P=8
n=100	3.98607	15.79499	66.63112
n=250	3.989606	16.30609	64.96939
n=500	4.40032	17.595798	69.96692
n=1000	3.99225	16.01525	63.95112

Graphs:



Analysis:

Speedup with Increasing Processes (P):

- As expected, the speedup generally increases with the number of processes. This behavior aligns with the parallel nature of the algorithm, where more processes lead to concurrent computation, thus reducing the overall execution time.

- For example, consider the case of n=100. As P increases to 2, 4, and 8, the speedup grows significantly, reaching 66.63. This indicates substantial gains in performance by parallelizing the computation across multiple processes.

Impact of Matrix Size (n):

- Larger matrix sizes generally result in longer computation times due to the increased number of operations required. This observation is evident from the provided runtimes, where the execution time increases notably as the matrix size (n) grows.

- For instance, comparing the runtime for n=100 and n=1000 with P=1, the execution time increases significantly from 3.54 seconds to 3575.58 seconds. This illustrates the impact of matrix size on computational complexity and, consequently, execution time.

Problem 2:

Matrix_Multiplication_Parallel

Input: $A(m \times n)$, $B(n \times q)$, $m \neq n \neq q$
Input: $Q = \{Q_0 \ldots Q_r \ldots Q_{p-1}\}$ // p processors
Output: $C(m \times q)$

For each processor Q_r Do:

$A_r(\frac{m}{p} \times n) \leftarrow \left(\frac{rm}{p} : \frac{(r+1)m}{p} - 1, :\right)$ // assign A slice to Q_r

$B_r(n \times \frac{q}{p}) \leftarrow \left(:, \frac{rq}{p} : \frac{(r+1)q}{p} - 1\right)$ // assign B slice to Q_r

$C_r(\frac{m}{p} \times \frac{q}{p}) \leftarrow 0$ // to every processor

For $t = 0$ to $p-1$
  IF $t \neq 0$
    $SPROC = (r + t) \bmod p$
    $RPROC = (r - t + p) \bmod p$
    SEND B_r to Processor Q_{SPROC}
    RECV B_{RPROC} from Processor Q_{RPROC}
  END IF
  For $i = 0$ to $\frac{m}{p} - 1$
    For $j = 0$ to $\frac{q}{p} - 1$
      For $K = 0$ to $n - 1$
        $C(i, j + RPROC \cdot \frac{q}{p}) \mathrel{+}= A_r(i,K) * B_{RPROC}(K,j)$
      End For
    End For
  End For
End For

Total Runtime Analysis: $O\left(\frac{mnq}{p} + P \cdot t_s + n \cdot \frac{q}{p} \cdot tw\right)$

Each processor is responsible for a slice of the matrix multiplication workload. The time complexity for this part of the process is $(mnq)/P$, reflecting that each processor handles $1/P$th of the computation, assuming an even distribution of the matrix A's rows and the full matrix B. On the communication side, there are the startup cost and the data transfer cost. The startup cost, denoted by $P \cdot ts$, aggregates the latency involved in initiating communication rounds across all P processors. The data transfer cost, $n \cdot (q/P) \cdot tw$, accounts for the time to transmit the relevant portions of matrix B each processor needs to perform its computation. This cost is incurred in each communication round, of which there are $P-1$. However, in the big O notation simplification, we account for this as a single round of communication due to the parallel nature of these operations. Altogether, the estimated total runtime complexity of the algorithm, considering both the computation and communication phases without overlap, is $O(((mnq)/P)+(P \cdot ts)+(n \cdot (q/P) \cdot tw))$.