### **Stock Market Simulation**

#### **1. Introduction**

This report analyzes the performance of a parallel flower market simulation using four programming models: **Serial**, **OpenMP**, **MPI**, and a **Hybrid (MPI + OpenMP)** approach. The goal is to evaluate how different parallelization techniques improve execution time and simulate more realistic trading dynamics between buyers and sellers in a competitive market environment.

The simulation models a dynamic trading market where **multiple buyers** attempt to purchase different types of flowers (e.g., roses, tulips, sunflowers) from **multiple sellers**. Each buyer has specific demands and a budget, while each seller has limited stock and price settings. The simulation runs for multiple rounds or until all buyer demands are fulfilled or seller stocks are depleted.

#### **2. Programming Concepts**

##### **2.1 Serial Approach**

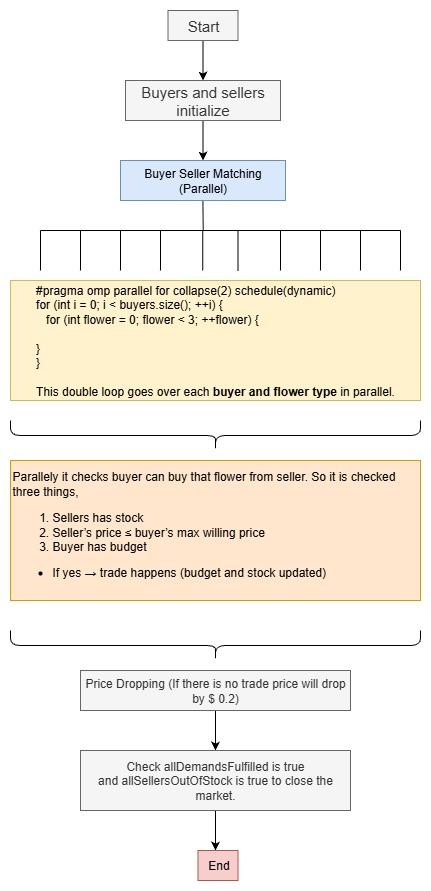
In the **serial version**, all buyer-seller interactions are handled by a **single process on one CPU core**. Buyers go through sellers sequentially to purchase flowers based on availability, price, and budget. This version is straightforward but **lacks performance and scalability** for large numbers of buyers and sellers.

##### 

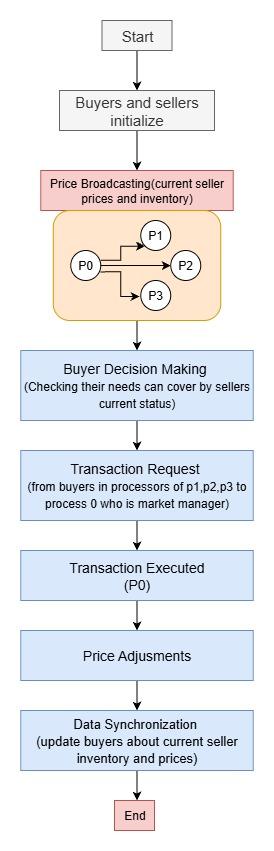
##### **2.2 Shared Memory Programming (OpenMP)**

In the **OpenMP version**, the simulation runs as a **single process** using **multiple threads** to exploit shared memory on multicore CPUs. Flower trading operations (e.g., processing each buyer or flower type) are parallelized. This allows:

* Parallel processing of buyers or flower types.
* Reduced simulation time on multicore machines.
* Shared memory access, avoiding explicit communication overhead.



##### **2.3 Distributed Memory Programming (MPI)**



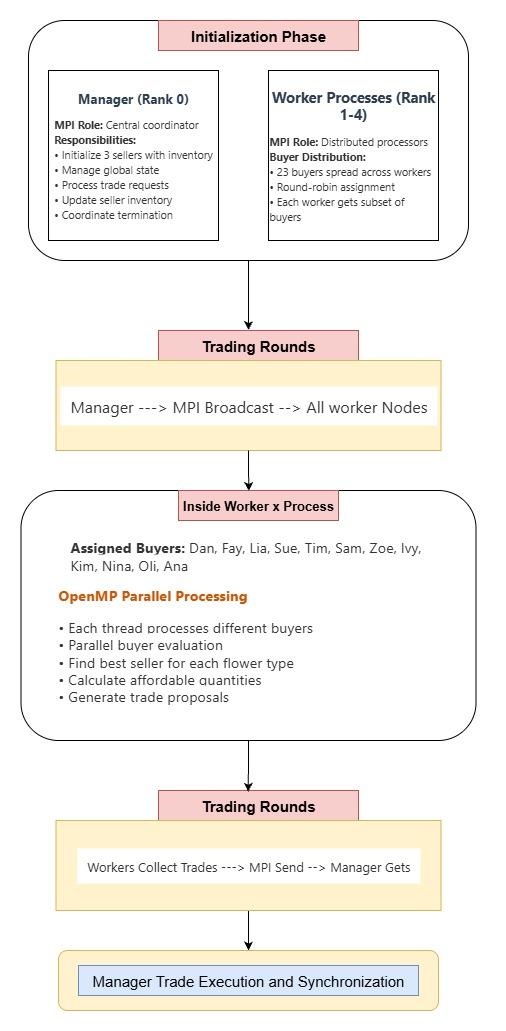
In the **MPI version**, the simulation is **distributed across multiple processes**, which may run on separate CPU cores or machines. Each MPI process is assigned a subset of buyers, while sellers may be centralized or replicated.

MPI is used for:

* Distributing buyer workloads.
* Coordinating seller stock updates.
* Communicating purchase results or boundary market information using “MPI\_SendRecv”.

This model enables **scalability across clusters or multi-node systems**, though communication overhead must be carefully managed.

##### **2.4 Hybrid Programming (MPI + OpenMP)**



The **hybrid approach** combines **MPI** (for inter-process communication and workload distribution) with **OpenMP** (for intra-process thread-level parallelism). For example:

* Each MPI process handles a group of buyers.
* Within each MPI process, OpenMP threads process different flower types or individual buyers concurrently.

#### **3. Accuracy of Parallel Code Compared to Serial Code**

##### **3.1 RMSE Calculation**

The accuracy of the parallel implementations was measured by comparing their results to the serial code. The **Root Mean Squared Error (RMSE)** was used as a metric to quantify the difference between the parallel and serial methods:

The RMSE values were found to be very small for all parallel methods, indicating that the parallel solutions are accurate and the boundary conditions are correctly maintained across the simulations.

|  |  |
| --- | --- |
| **Method** | **RMSE** |
| OpenMP | 4.55 |
| MPI | 9.76 |
| Hybrid | 4.33 |

#### **4. Timin****g Results**

We measured the execution time for each version of the simulation (Serial, OpenMP, MPI, Hybrid).

|  |  |
| --- | --- |
| **Method** | **Time(s)** |
| Serial | 19.35 |
| OpenMP | 15.2675 |
| MPI | 9.76 |
| Hybrid | 12.4056 |

.

|  |  |
| --- | --- |
| **Number of Threads** | **Time(s)** |
| 4 | 15.776 |
| 8 | 15.3044 |
| 12 | 15.2675 |
| 16 | 15.1984 |

OpenMP executes times against Number of threads.