# Programme Structure

**1**. System accepts the first argument value and treat it as total number of vessels.
**2**. Declaration stage:
- Using [start] and [end] (clock_t type) to calculate the duration of initialization.
- Using [timeForInitialization] (double type) to record the initialization time.
- Declare a [location] (int type) to help system retrieve data.
- Declare a [numberOfRounds] (long type) to help root processor calculate which round is going on via this variable.
- Declare a [totalNumberOfStrikes] (long type) to help root processor calculate how many strikes occurs.
- Declare a [timeToMoveVessels] (double type): This is the first timestamp after the initialization.
- Declare a [previousTimestamp] (double type) to help root processor calculate timestamps that for printing on the terminal.
- Declare a [timestamp] (double type): Use this variable to print timestamp of every round.
- Declare [myId] & [size] (int type) to help processors get relative information.

**3**. Initialize the programme
- Every processor gets its own rank ID.
- Every processor knows the size of processors.

**4**. Prepare a seed to generate random number for slave processors.
**5**. Allocate vessels to different slave processors.
- Make sure that each slave processor has been allocated vessels equally.

**6**. Synchronization: Wait until every processor complete the initialization.
**7**. Root processor starts to calculate the initialization time and first timestamp.
**8**. Starting the while loop.
- Each slave processor get some random numbers and allocate them to the vessels.
- Using locationCount[location] to store the information about how many vessels are there in the same location.
- Using MPI_Reduce() to collect the information (how many vessels are there in each location) from slave processors and put all (sum the info up first) the data into the globalCount[]. (More information is in the comment, please look at the code.)

**9**. Root processor go through all the locations (1500).
- Root processor check if strikes can occur in each location.
- Accumulate total number of strikes.

**10**. Print relative information on the terminal.
**11**. Repeat step-h to step-j until the loop has executed for 60 seconds.
**12**. Exit from the loop and root processor print the final result.
**13**. Finalize the programme.

# Efficiency:

**1. Appropriate data type:**

```
short locationCount[TOTALLOCATIONS] = {0};
```

The array is defined as a [short] type. That's because the length of the array is constant and it won't exceed 1500. Hence, using a [short] data type will reduce the memory usage, which helps RAM to retrieve data more quickly.

**2. Using MPI_Wtime() to increase the accuracy:**

```
timeToMoveVessels = 0.0 - MPI_Wtime() + timeForInitialization;
```

That's because MPI_Wtime will return a floating-point number of seconds, which representing elapsed wall-clock time since a time point in the past. Therefore, system can record a more accurate timestamp.

**3. Using MPI_Reduce() to integrate data and reduce the inter-process communication:**

```
MPI_Reduce(locationCount, globalCount, TOTALLOCATIONS, MPI_UNSIGNED_SHORT, MPI_SUM, 0,
MPI_COMM_WORLD);
```

Each slave processor controls some vessels. Every vessel will go to a random location and then the slave processor will record the data, namely put specific information in the lcationCount[] array. Using MPI_Reduce(), the method will collect all the data from locationCount[] (each slave processor has its own locationCount[]). Because we use [MPI_SUM] as the reduce operation, MPI_Reduce will integrate all the data that comes from locationCount[] and put the integrated data to the globalCount[].

Due to each slave processor controls some vessels rather than each processor controls only one vessel. locationCount[] collects vessel's location information and integrate the information at first, then pass it to globalCount via MPI_Reduce(). We avoid passing a single data to root processor (we pass an array which contains some vessels' location information), so doing like this will dramatically reduce the inter-process communication.

**4. Allocate vessels to slave processors equally:**

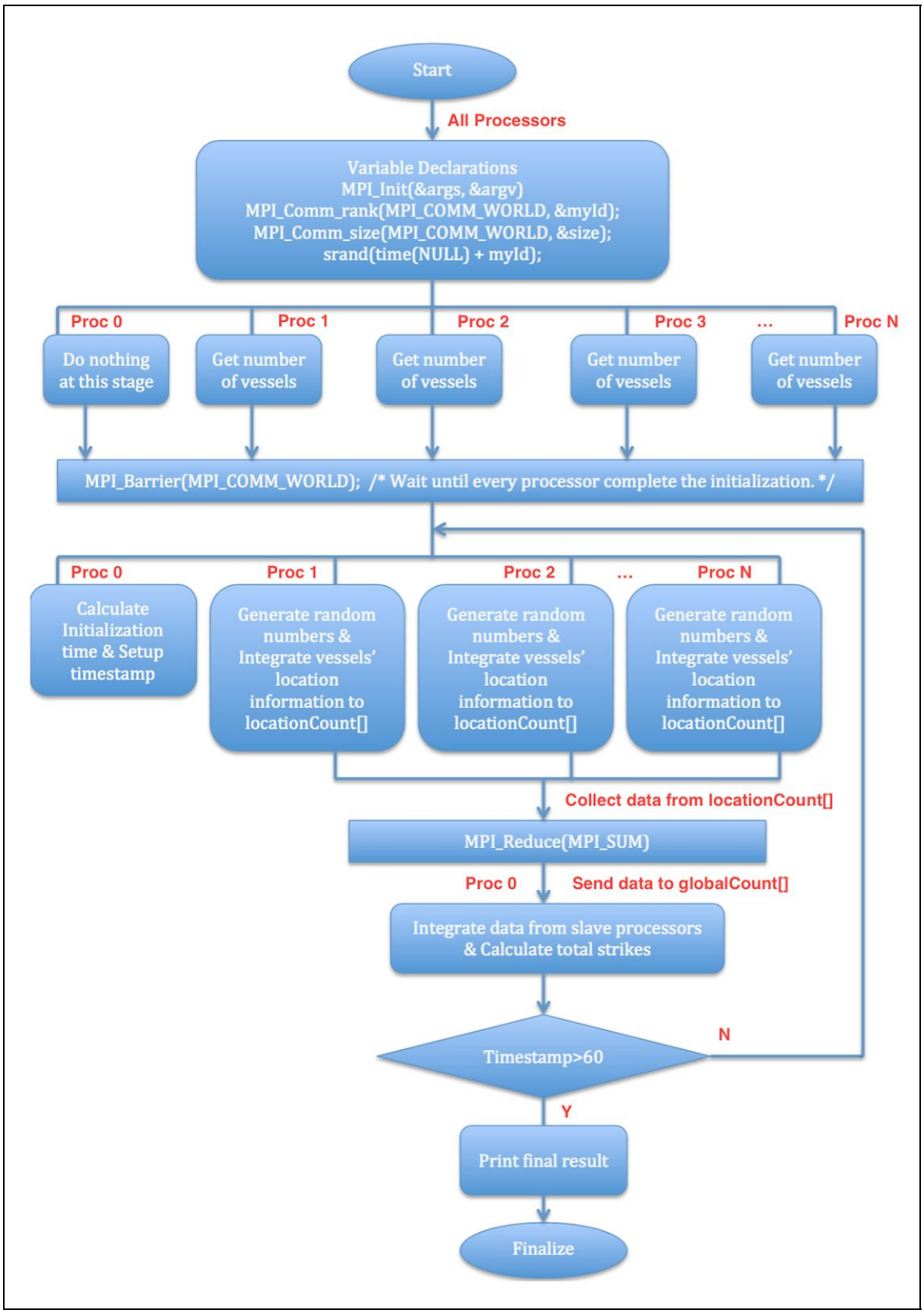
```
/* Allocate vessels to different slave processors. */
    int numOfVesselsForEachProc;
    if (myId != 0) {
        numOfVesselsForEachProc = numOfVessels / (size - 1);
        if (numOfVessels % (size - 1) > myId - 1) {
            numOfVesselsForEachProc += 1;
        }
    }
```

By using this method can let system allocate vessels to different processors as equal as possible. If we don't do this, for example, we have 3 slave processors. Proc A controls 100 vessels. Proc B controls 20 vessels. Proc C controls only 1 vessel. Proc A needs generate more random numbers and store more data in the array. When Proc A goes into a loop, it will take a longer time to iterate and other processors have to wait. Hence, allocate vessels to slave processor equally is fair to every slave processor and can reduce the waiting time.

P.S. If you want each processor to control only one vessel. For example, allocate 30 vessels to 30 processors. Run like this:

```
[sten11@fit-parrcomp PartB]$ mpicc Fleet_Sim.c
[sten11@fit-parrcomp PartB]$ mpirun -np 30 ./a.out 30
```

# Inter-Process Communication Schema



# Performance Matrix

1. Number of Processors: 30 & Number of Vessels: 30

| | Test 1 | Test 2 | Test 3 |
|---|---|---|---|
| Rounds | 189506 | 152163 | 222905 |
| Strikes | 0 | 0 | 0 |

2. Number of Processors: 50 & Number of Vessels: 50

| | Test 1 | Test 2 | Test 3 |
|---|---|---|---|
| Rounds | 69397 | 89517 | 62654 |
| Strikes | 0 | 7 | 0 |

3. Number of Processors: 100 & Number of Vessels: 100

| | Test 1 | Test 2 | Test 3 |
|---|---|---|---|
| Rounds | 44138 | 51638 | 49589 |
| Strikes | 3 | 1 | 0 |

4. Number of Processors: 150 & Number of Vessels: 150

| | Test 1 | Test 2 | Test 3 |
|---|---|---|---|
| Rounds | 17465 | 29781 | 32273 |
| Strikes | 4 | 6 | 2 |

Average-edition Table

| Number of Processors | 30 | 50 | 100 | 150 |
|---|---|---|---|---|
| Number of Vessels | 30 | 50 | 100 | 150 |
| Average rounds | 188191 | 73856 | 48788 | 26506 |
| Average strikes | 0 | 2 | 1 | 4 |