Questions on Chapter 3

1. **Write lines of code to time a block of MPI code and report a single elapsed time.**

The following code can be used to time a block of MPI code and report a single elapsed time:

```
double local_start, local_finish, local_elapsed, elapsed;
. . .
MPI_Barrier(comm);
local_start = MPI_Wtime();
/* Code to be timed */
. . .
local_finish = MPI_Wtime();
local_elapsed = local_finish - local_start;
MPI_Reduce(&local_elapsed, &elapsed, 1, MPI_DOUBLE,
   MPI_MAX, 0, comm);

if (my_rank == 0)
   printf("Elapsed time = %e seconds\n", elapsed);
```

2. **Why at some point, the run-times of parallel program can start to get worse?**

- Because there is a fairly-common relation between the run-time of serial programs and the run-time of parallel programs

$$T_{\text{parallel}}(n,p) = T_{\text{serial}}(n)/p + T_{\text{overhead}}.$$

- So if we increase input size n and put p to small values we notice the run time increases
- Also dividing work among processes could lead to over head and increase run-time

3. **Sort 5, 9, 4, 3 using quick sort and using odd-even transposition sort.**

Using quick sort:

- In quick sort we select 5 as pivot and partition the list into 2 lists one for elements less than 5: [4,3] and greater than 5: [9]
- We repeat the same step for the 2 lists
- [4,3] → [3] and [4]
- We then combine the 3 lists together after sorting [3,4,9]
- Finally we put the first pivot back in the middle [3,4,5,9]

Using odd-even transposition sort:

Start: 5,9,4,3
Even phase: Compare-swap (5,9) and (4,3), getting the list 5,9,3,4.
Odd phase: Compare-swap (9,3), getting the list 5,3,9,4.
Even phase: Compare-swap (5,3) and (9,4), getting the list 3,5,4,9.
Odd phase: Compare-swap (5,4), getting the list 3,4,5,9.

4. **Sort 15, 11, 9, 16, 3, 14, 8, 7, 4, 6, 12, 10, 5, 2, 13, 1 using parallel odd-even transposition sort (with and without lists' merge) on four processors.**

Table 3.8 Parallel Odd-Even Transposition Sort

Time	Process 0	1	2	3
Start	15, 11, 9, 16	3, 14, 8, 7	4, 6, 12, 10	5, 2, 13, 1
After Local Sort	9, 11, 15, 16	3, 7, 8, 14	4, 6, 10, 12	1, 2, 5, 13
After Phase 0	3, 7, 8, 9	11, 14, 15, 16	1, 2, 4, 5	6, 10, 12, 13
After Phase 1	3, 7, 8, 9	1, 2, 4, 5	11, 14, 15, 16	6, 10, 12, 13
After Phase 2	1, 2, 3, 4	5, 7, 8, 9	6, 10, 11, 12	13, 14, 15, 16
After Phase 3	1, 2, 3, 4	5, 6, 7, 8	9, 10, 11, 12	13, 14, 15, 16

5. Write lines of code for computing the partner rank in odd-even transposition sort?

First, how do we compute the partner rank? And what is the partner rank when a process is idle? If the phase is even, then odd-ranked partners exchange with my_rank−1 and even-ranked partners exchange with my_rank+1. In odd phases, the calculations are reversed. However, these calculations can return some invalid ranks: if my_rank = 0 or my_rank = comm_sz−1, the partner rank can be −1 or comm_sz. But when either partner = −1 or partner = comm_sz, the process should be idle. We can use the rank computed by Compute_partner to determine whether a process is idle:

```
if (phase % 2 == 0)      /* Even phase */
   if (my_rank % 2 != 0)     /* Odd rank */
      partner = my_rank - 1;
   else                      /* Even rank */
      partner = my_rank + 1;
```



```
else                     /* Odd phase */
   if (my_rank % 2 != 0)     /* Odd rank */
      partner = my_rank + 1;
   else                      /* Even rank */
      partner = my_rank - 1;
if (partner == -1 || partner == comm_sz)
   partner = MPI_PROC_NULL;
```

6. How to make the communication safe in the parallel odd-even sort program and show how the communication is safe within five processes?

- To make a communication safe it must be restructured as a cause of an unsafe communication is multiple processes sending and receiving messages to each other simultaneously so we need to restructure communication so that some of the processes receive before sending

Safe communication with five processes

```
void*          recv_buf_p      /* out */,
int            recv_buf_size   /* in  */,
MPI_Datatype   recv_buf_type   /* in  */,
int            source          /* in  */,
int            recv_tag        /* in  */,
MPI_Comm       communicator    /* in  */,
MPI_Status*    status_p        /* in  */);
```

Questions on Chapter 4

1. **Describe two applications of barriers.**
2. **Give the implementation of a barrier using busy-waiting and a mutex, semaphores, and condition variables.**
3. **Reusing counter in barrier implementation using busy waiting is problematic, while it is not problematic when semaphores are used, explain why.**
4. **Use insert function to add 2, 8, 7, 5 to a linked list. Then, use member function to find 5 and 9. Finally, delete 5 from the list.**
5. **Executing concurrent multiple read and write operations on a linked list can cause problems. What are these problems and how they can be solved?**
6. **How read-write locks can be implemented?**
7. **Explain how matrix dimensions can affect the efficiency of matrix-vector multiplication program.**
8. **Give an example where incorrect program can produce correct result.**