Q.1 Explain the working behavior of the following program segment and write your comments. Assume that there is no syntax error in the program segment. [4]

```
int a[10], b[10], npes, myrank;
MPI_status status;
...
MPI_Comm_size(MPI_COMM_WORLD, &npes);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Send(a, 10, MPI_INT, (myrank+1)%npes, 1, MPI_COMM_WORLD);
MPI_Recv(b, 10, MPI_INT, (myrank-1+npes)%npes, 1, MPI_COMM_WORLD);
...
```

Ans:

From Bookmarked website

http://parallelcomp.uw.hu/ch06lev1sec3.html

Q.2 Explain how Foster's design methodology can be applied to find the numerical integral of a given function. [4]

Ans:

https://www.mcs.anl.gov/~itf/dbpp/text/node22.html#SECTION02380000000000000000

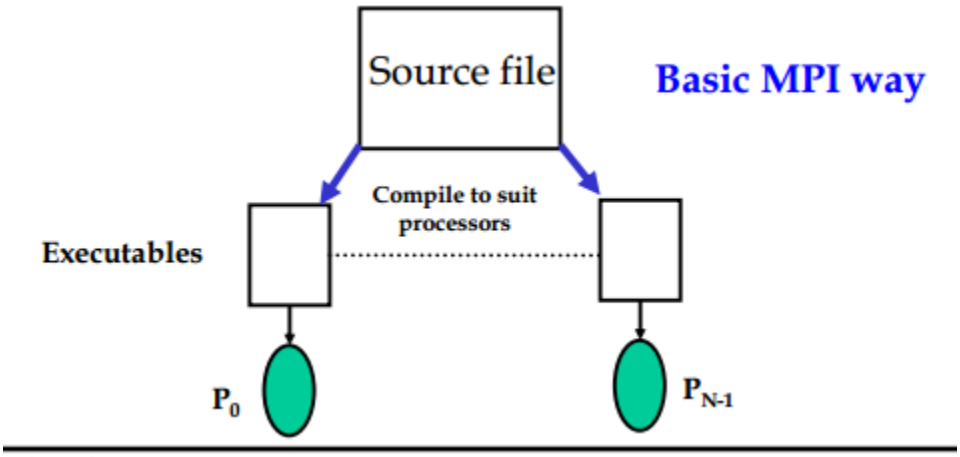
https://www.dartmouth.edu/~rc/classes/intro_mpi/

Q.3 What is Single Program Multiple Data (SPMD) model? Explain the general structure of a C program that uses SPMD model. [4]

Ans:

Single Program Multiple Data (SPMD) model

Different processes merged into one program. Within program, control statements select different parts for each processor to execute. All executables start together -static process creation.



Evaluating General Message

Message Passing SPMD : C program

```
main (int argc, char **argv)
{
    if (process is to become a controller process)
                {
                        Controller (/* Arguments /*);
                }
        else
                {
                        Worker (/* Arguments /*);
                }
}
```

Q.4 Write a C/MPI program Scatter sets of 50 integers from the root to each process in the group. [4]

14.8.4 Scatter (MPI_SCATTER)

MPI_SCATTER is the inverse operation to MPI_GATHER. The root sends a message buffer with MPI_SEND. This message is split into n equal segments; the i-th segment is sent to the i-th process in the group. The send buffer is ignored for all nonroot processes. Note that the amount of data sent must equal the amount received between each process and the root as shown in the following example:

```
/* MPI_Scatter Example:Scatter an array of 100 integers */
/* across multiple processes    */
/* Get the number of Processes  */
int gsize;

int root;
int rbuf[100];

MPI_Comm_size (MPI_COMM_WORLD, &gsize);

/* An array of 100 ints */
int *sendbuf = (int *) malloc ( gsize * 100 * sizeof(int) );

/**
* Scatter the array across all proceses
* All arguments to the function are significant on process root
* On the sub processes, only arguments:
     * recvbuf (rbuf),
     * recvcount (100),
     * recvtype (MPI_INT),
     *root and comm are significant
    **/
    MPI_Scatter( sendbuf, 100, MPI_INT
                , rbuf, 100, MPI_INT, root, comm);
```

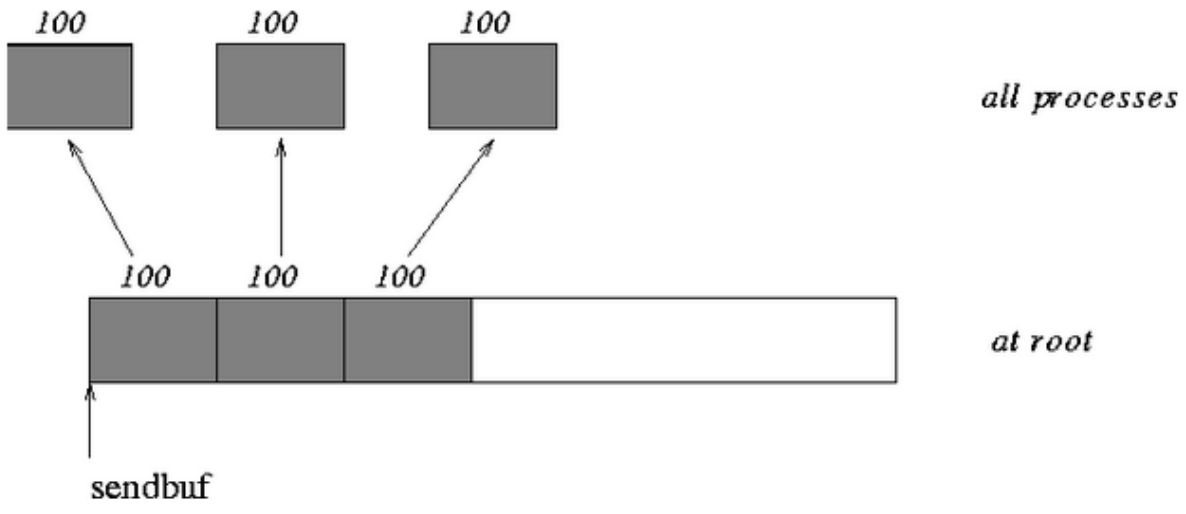


Figure 8: The root process scatters sets of 100 ints to each process in the group.

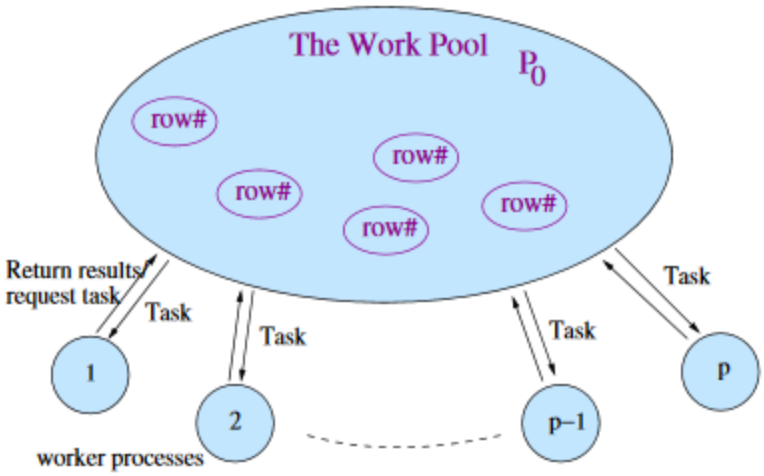
1. http://mpi-forum.org/docs/mpi-1.1/mpi-11-html/node72.html
2. (Programming Series) Vladimir Silva-Grid computing for developers-Charles River Media (2005).pdf
3. https://www.mcs.anl.gov/~itf/dbpp/text/node97.html

Q.5 Explain the method of Parallelizing Mandelbrot Set Computation using Dynamic Task Assignments. [4]

Parallel Mandelbrot Set

- Making each task be computing one pixel would be too fine-grained and lead to a lot of communication. So we will count computing one row of the display as one task.
- Two ways of assigning tasks to solve the problem in parallel.
  - **Static task assignment**. Each process does a fixed part of the problem. Three different ways of assigning tasks statically:
    - **Divide by groups of rows** (or columns)
    - **Round robin by rows** (or columns)
    - **Checkerboard mapping**
  - **Dynamic task assignment**. A **work-pool** is maintained that worker processes go to get more work. Each process may end up doing different parts of the problem for different inputs or different runs on the same input.

Dynamic Task Assignment: The Work Pool



The work pool holds a collection of tasks to be performed. Processes are supplied with tasks as soon as they finish previously assigned task. In more complex work pool problems, processes may even generate new tasks to be added to the work pool.

- *task* (for Mandelbrot set): one row to be calculated
- *coordinator process (process 0)*: holds the work pool, which simply consists of number of rows that still need to be done

The Work Pool Pseudo-Code

```
mandelbrot(h, w, p, id)
// p+1 processes, numbered id=0,...,p.

  if (id == 0) {
    count = 0
    row = 0
    for(k=1; k<=p; k++) {
      send(&row, P_k, data)
      count++; row++;
    }
    do {
      recv(&id, &r, color, P_any, result)
      count–
      if (row < h) {
        send(&row, P_id, data)
        row++; count++
      } else {
        send(&row, P_id, termination)
      }
      display(r, color)
    } while (count > 0)
```

The Work Pool Pseudo-Code (contd.)

```
} else {   // the worker processes
    recv(&y, P0, ANYTAG, &source_tag)
    while (source_tag == data) {
        c.imag = imin + y * scale_imag
        for (x=0; x<w; x++) {
            c.real = rmin + x * scale_real
            color[x] = cal_pixel(c)
        }
        send(&id, &y, color, P0, result)
        recv(&y, P0, ANYTAG, &source_tag)
    }
}
```

Analysis of the Work-Pool Approach

- Let m be the maximum number of iterations in `cal_pixel()` function. Then sequential time is $\boxed{T^*(m,n) = O(mn)}$.
- **Phase I**. $T_{\text{comm}}(n) = p(t_{\text{startup}} + t_{\text{data}})$.
- **Phase II**. Additional $h-p$ values are sent to the worker processes. $T_{\text{comm}}(n) = (h-p)(t_{\text{startup}} + t_{\text{data}})$. Each worker process computes $\leq n/p$ points. Thus we have $T_{\text{comm}}(n) \leq (m \times n)/p = O(mn/p)$.
- **Phase III**. A total of h rows are sent back, each w elements wide.

$$\begin{aligned} T_{\text{comm}}(n) &= h(t_{\text{startup}} + wt_{\text{data}}) \\ &= ht_{\text{startup}} + hwt_{\text{data}} = O(ht_{\text{startup}} + nt_{\text{data}}) \end{aligned}$$

- The overall parallel run time:

$$\boxed{T_p(n) = O(ht_{\text{startup}} + (n+h)t_{\text{data}} + \tfrac{mn}{p})}$$

- The speedup is:

$$\boxed{S_p(n) = O\left(\frac{p}{1 + \frac{hp}{mn}t_{\text{startup}} + \frac{(n+h)p}{mn}t_{\text{data}}}\right)}$$

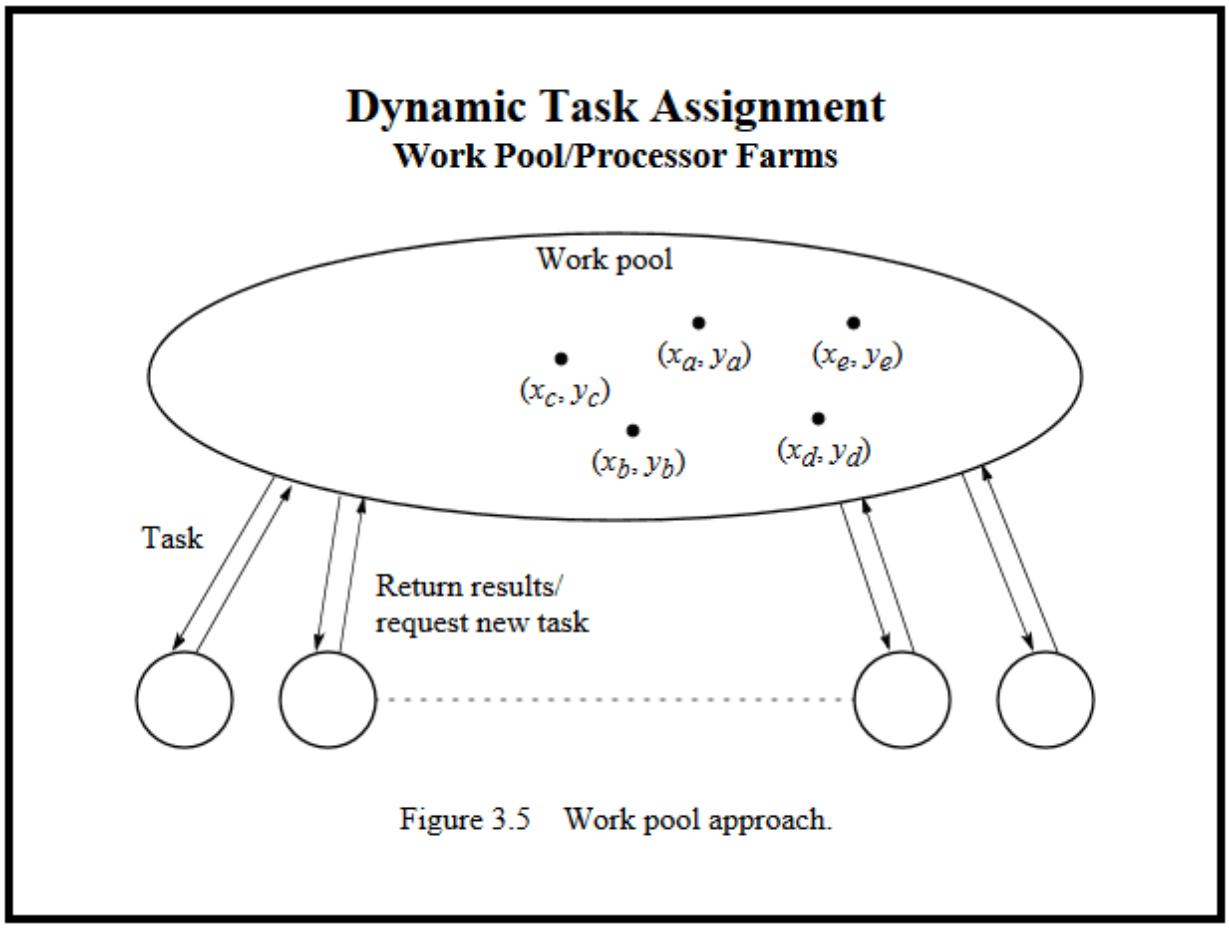


Figure 3.5 Work pool approach.

Coding for Work Pool Approach

Master

```
count = 0;                          /* counter for termination*/
row = 0;                            /* row being sent */
for (k = 0; k < procno; k++) {      /* assuming procno<disp_height */
    send(&row, Pk, data_tag);       /* send initial row to process */
    count++;                        /* count rows sent */
    row++;                          /* next row */
}

do {
    recv (&slave, &r, color, PANY, result_tag);
    count--;                    /* reduce count as rows received */
    if (row < disp_height) {
      send (&row, Pslave, data_tag);        /* send next row */
      row++;                                /* next row */
      count++;
    } else
      send (&row, Pslave, terminator_tag); /* terminate */
    rows_recv++;
    display (r, color);                     /* display row */
} while (count > 0);
```


Slave

```
recv(y, Pmaster, ANYTAG, source_tag);/* receive 1st row to compute */
while (source_tag == data_tag) {
    c.imag = imag_min + ((float) y * scale_imag);
    for (x = 0; x < disp_width; x++) {/* compute row colors */
      c.real = real_min + ((float) x * scale_real);
      color[x] = cal_pixel(c);
    }
    send(&i, &y, color, Pmaster, result_tag);/* row colors to master */
    recv(y, Pmaster, source_tag);       /* receive next row */
};
```

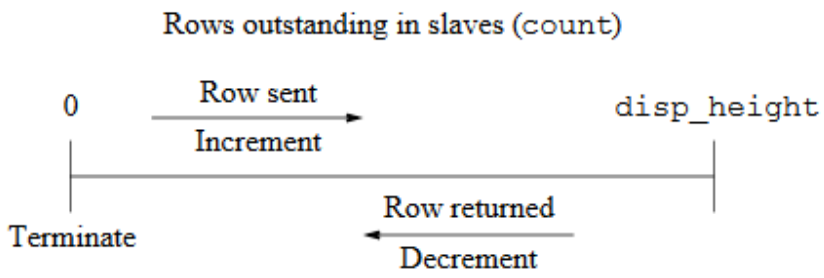


Figure 3.6 Counter termination.

Analysis

Sequential

$t_s \leq \max \times n = O(n)$

Parallel program

Phase 1: Communication - Row number is sent to each slave

$t_{comm1} = s(t_{startup} + t_{data})$

Phase 2: Computation - Slaves perform their Mandelbrot computation in parallel

$$t_{comp} \leq \frac{\max \times n}{s}$$

Phase 3: Communication - Results passed back to master using individual sends

$$t_{comm2} = \frac{n}{s}(t_{startup} + t_{data})$$

Overall

$$t_p \leq \frac{\max \times n}{s} + \left(\frac{n}{s} + s\right)(t_{startup} + t_{data})$$

http://cs.boisestate.edu/~amit/teaching/430/handouts/ep.pdf

http://www.cs.nthu.edu.tw/~ychung/slides/para_programming/slides3.pdf

Barry Wilkinson, Michael Allen-Parallel Programming_ Techniques and Applications Using Networked Workstations and Parallel Computers (2nd Edition) -Prentice Hall (2004) – pdf downloaded