Pseudocode-1

begin

commsize, rank;

MPI\_Init(NULL, NULL)

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank)

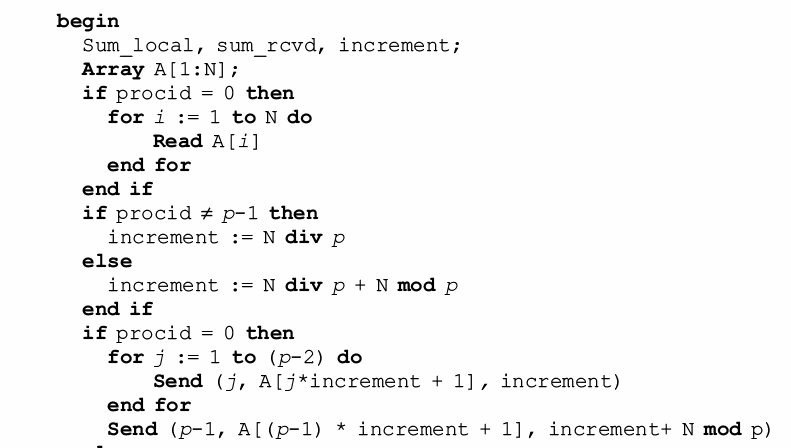
MPI\_Comm\_size(MPI\_COMM\_WORLD, &commsize)

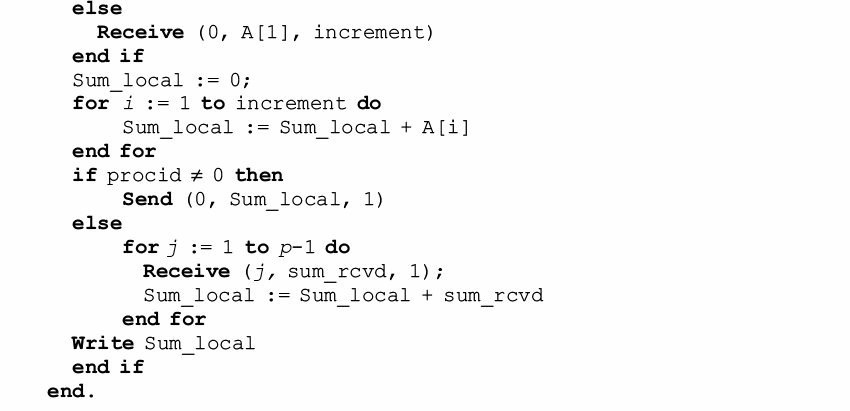
Print "Hello World from Process no. " rank

MPI\_Finalize();

end

Pseudocode-2





Pseudocode-3

begin

    NUM\_ROWS\_A,

    NUM\_COLUMNS\_A, NUM\_ROWS\_B, NUM\_COLUMNS\_B, rank, size, i, j, k, low\_bound, portion;

Array A[NUM\_ROWS\_A][NUM\_COLUMNS\_A], B[NUM\_ROWS\_B][NUM\_COLUMNS\_B], result[NUM\_ROWS\_A][NUM\_COLUMNS\_B];

MPI status, request;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

  if rank != size-1 then

portion = NUM\_ROWS\_A div size

end if

else

portion = NUM\_ROWS\_A div size + NUM\_ROWS\_A mod size

end if

  if rank = 0 then

      Read A

      Read B

      for i = 1 to size-1 do

          MPI\_Send(&A[i\*portion + 1][0], portion \* NUM\_COLUMNS\_A, MPI\_DOUBLE, i, 3, MPI\_COMM\_WORLD);

      end for

      MPI\_Send(&A[(size-1)\*portion + 1][0], (portion + NUM\_ROWS\_A mod size) \* NUM\_COLUMNS\_A, MPI\_DOUBLE, i, 3, MPI\_COMM\_WORLD);

      end if

          else MPI\_Recv(&A[1][0], portion \*NUM\_COLUMNS\_A, MPI\_DOUBLE, 0, 3, MPI\_COMM\_WORLD, &status);

      end if

end if

          MPI\_Bcast (&B, NUM\_ROWS\_B \* NUM\_COLUMNS\_B, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

  for i = 1 to portion do

      for j = 1 to NUM\_COLUMNS\_B do

          for k = 1 to NUM\_ROWS\_B do

              result[i][j] := result[i][j] + (A[i][k] \* B[k][j]);

          end for

      end for

  end for

  if rank !=0

    MPI\_Send(&result[rank\*portion + 1][0], portion \* NUM\_COLUMNS\_B, MPI\_DOUBLE, 0, 6, MPI\_COMM\_WORLD);

  end if

  else

    for i = 1 to size-1 do

          MPI\_Recv(&result[i\*portion + 1][0], portion \* NUM\_COLUMNS\_A, MPI\_DOUBLE, i, 6, MPI\_COMM\_WORLD);

    end for

    MPI\_Recv(&result[(size-1)\*portion + 1][0], (portion + NUM\_ROWS\_A mod size) \* NUM\_COLUMNS\_A, MPI\_DOUBLE, i, 6, MPI\_COMM\_WORLD);

    print result

  end if

end

Pseudocode-4

begin {main}

  global ARRAY\_SIZE

  number\_of\_elements, chunk\_size, own\_chunk\_size,number\_of\_process, rank;

  Array data, chunk;

  MPI\_Status status;

  MPI\_Init(&argc, &argv);

  MPI\_Comm\_size(MPI\_COMM\_WORLD, &number\_of\_process);

  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

  if rank = 0 then

      number\_of\_elements := ARRAY\_SIZE;

      for i = 0 to < number\_of\_elements do

          Read data[i]

      end for

  end if

  MPI\_Barrier(MPI\_COMM\_WORLD);

  MPI\_Bcast(&number\_of\_elements, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

  if rank != number\_of\_process-1 then

    chunk\_size = ARRAY\_SIZE div number\_of\_process

  end if

  else

    chunk\_size = ARRAY\_SIZE div number\_of\_process + ARRAY\_SIZE mod number\_of\_process

  end if

  chunk[chunk\_size];

    if rank = 0 then

      for i = 1 to number\_of\_process-1 do

          MPI\_Send(&data[i\*chunk\_size + 1], chunk\_size, MPI\_DOUBLE, i, 3, MPI\_COMM\_WORLD);

      end for

      MPI\_Send(&A[(number\_of\_process-1)\*chunk\_size + 1], (chunk\_size + ARRAY\_SIZE mod number\_of\_process), MPI\_DOUBLE, i, 3, MPI\_COMM\_WORLD);

    end if

    else

      MPI\_Recv(&chunk[1], chunk\_size, MPI\_DOUBLE, 0, 3, MPI\_COMM\_WORLD, &status);

    end if

    quicksort(chunk, 1, chunk\_size);

    if rank !=0 then

      MPI\_Send(&chunk[1], chunk\_size, MPI\_DOUBLE, 0, 3, MPI\_COMM\_WORLD, &status));

    end if

  if rank = 0 then

    for i = 1 to number\_of\_process-1 do

          MPI\_Recv(&data[i\*chunk\_size + 1], chunk\_size, MPI\_DOUBLE, i, 3, MPI\_COMM\_WORLD);

          merge(data[1], i\*chunk\_size, data[i\*chunk\_size + 1], chunk\_size);

      end for

      MPI\_Recv(&data[(number\_of\_process-1)\*chunk\_size + 1], (chunk\_size + ARRAY\_SIZE mod number\_of\_process), MPI\_DOUBLE, i, 3, MPI\_COMM\_WORLD);

      merge(data[1], (number\_of\_process-1)\*chunk\_size, data[i\*chunk\_size + 1], chunk\_size + ARRAY\_SIZE mod number\_of\_process);

  end if

  MPI\_Finalize();

  return 0;

end {main}

swap(input: arr[], i, j, output: void)

begin

  local t;

  t := arr[i];

  arr[i] := arr[j];

  arr[j] := t;

end

quicksort(input: arr[start, end], output: void)

begin

    pivot, index;

    if end <= start then

        return

    pivot := arr[start + end / 2];

    swap(arr, start, start + end / 2);

    index = start;

    for i = start + 1 to < start + end do

        if arr[i] < pivot then

            index := index+1;

            swap(arr, i, index);

        end if

    end for

    swap(arr, start, index);

    quicksort(arr, start, index - start);

    quicksort(arr, index + 1, start + end - index - 1);

end

merge(input: arr1[], n1, arr2[], n2, output: result[])

begin

    Array result[n1 + n2];

    i = 0,j = 0, k;

    for k = 0 to < n1 + n2 do

        if i >= n1 then

            result[k] := arr2[j];

            j := j+1;

        end if

        else if j >= n2 then

            result[k] = arr1[i];

            i:=i+1;

        end if

        else if arr1[i] < arr2[j] then

            result[k] := arr1[i];

            i := i+1;

        end if

        else

            result[k] := arr2[j];

            j := j+1;

        end if

    end for

    return result;

end

Pseudocode-5

begin {main}

  global thread\_id

  global Array data

  local n;

  pthread\_create(&thread\_id, NULL, threadfunction1, &n)

  pthread\_join(thread\_id, NULL);

  for i = 0 to n do

    print data[i]

  end for

  pthread\_exit(NULL)

end {main}

threadfunction1(input: void \*args, output: void\*)

begin

  local n

  print pthread\_self()

  n := \*((int \*)args);

  Array data[n];

  if n > 0 then

    data[0] = 0

  end if

  if n > 1 then

    data[1] = 1;

  end if

  if n > 2 then

    for i = 2 to n do

      data[i] := data[i - 1] + data[i - 2]

    end for

  end if

  pthread\_detach(pthread\_self());

  pthread\_exit(NULL);

end

pseudocode-6

begin {main}

  global pthread\_mutex\_t lock, n=1000;

  if pthread\_mutex\_init(&lock, NULL) != 0 then

      print error

      return;

  end if

  print n

  pthread\_t thread\_id1;

  pthread\_t thread\_id2;

  pthread\_create(&thread\_id1, NULL, threadfunction1, NULL);

  pthread\_create(&thread\_id2, NULL, threadfunction2, NULL);

  pthread\_join(thread\_id1, NULL);

  pthread\_join(thread\_id2, NULL);

  print n

  pthread\_mutex\_destroy(&lock);

  pthread\_exit(NULL);

end {main}

threadfunction1(input: void \*args, output: void\*)

begin

    pthread\_mutex\_lock(&lock);

    n := n + 1;

    print n

    pthread\_mutex\_unlock(&lock);

    pthread\_detach(pthread\_self());

    pthread\_exit(NULL);

end

threadfunction2(input: void \*args, output: void\*)

begin

    pthread\_mutex\_lock(&lock);

    n := n - 1;

    print n

    pthread\_mutex\_unlock(&lock);

    pthread\_detach(pthread\_self());

    pthread\_exit(NULL);

end

pseudocode-7

begin {main}

  #pragma omp parallel

  #pragma omp single

      begin

        local n

        n = atoi(argv[1]);

        print fib(n);

      end

end {main}

fib(input: n, ouput: res)

begin

    local res, a, b;

    if n == 0 or n == 1 then

        res = n

    end if

    else

        int a, b;

#pragma omp task shared(a)

        a = fib(n - 1);

#pragma omp task shared(b)

        b = fib(n - 2);

#pragma omp taskwait

        res = a + b;

    end if

    print omp\_get\_thread\_num();

    return res;

end

pseudocode-8

begin {main}

  local i, N = 10, THREAD\_COUNT = 3, CHUNK\_SIZE = 3;

  print "Default Scheduling"

  #pragma omp parallel for num\_threads(THREAD\_COUNT)

      for i = 0 to N do

          print "ThreadID: %d, iteration: %d", omp\_get\_thread\_num(), i;

      end for

  print "\nStatic Scheduling\n"

  #pragma omp parallel for num\_threads(THREAD\_COUNT) schedule(static, CHUNK\_SIZE)

      for i = 0 to N do

          print "ThreadID: %d, iteration: %d", omp\_get\_thread\_num(), i;

      end for

  print "\nDynamic Scheduling\n";

  #pragma omp parallel for num\_threads(THREAD\_COUNT) schedule(dynamic, CHUNK\_SIZE)

      for i = 0 to N do

          print "ThreadID: %d, iteration: %d", omp\_get\_thread\_num(), i;

      end for

end {main}