**Project 2, due on 03/03.**

1. Copy code ~zxu2/Public/Demo_pkg/20140216-driver.tar.gz

2. Implement all functions as the member functions of class Compute_1d.

3. Use "mpimkintel" command to compile your code. See README.txt file for details.

4. Modify the script "HPCC_1.sh" to submit your runs.

**Problem 1. Parallel Numerical Integration for Undergraduate Students.**

Evaluate $\int_{0.0}^{10.0} \ln(x+20)e^{\sqrt{x}}dx$. Use the Demo_Pkg code to implement a parallel program using composite Gaussian quadrature rule to approximate this definite integral.

Suppose *P* processes are used and the integration domain $[0.0, 10.0]$ is partitioned into *M* grid blocks. Each of the processes is assigned with a sub-region $[x_{i,l}, x_{i,u}]$, which is partitioned into $(M/P)$ blocks. Here $i = 0,1, \ldots, P$-1.

We apply the 1D 3-point Gaussian quadrature rule to each of these grid blocks to compute a numerical quadrature value. The approximation to the given integral is obtained by summing up these numerical quadrature values.

1. Use point-to-point communication, specifically, non-blocking send and blocking receive to transfer the partial sum of quadrature values computed by each of the processes to process 0 and let process 0 compute the sum of these quadrature values.

2. Use $M = 10000, 20000\ and\ 40000$ to do the calculation respectively. For each computation, use 2, 4 and 8 processors respectively. Find the overall the wall clock times spent by the computation, and the communication respectively. Make a table to list the results.

**Hand-In.** Turn in the hardcopy of all your source code, and the report which contains results and a description of your implementation on point-to-point communication. Email the source code.

**Coding Hints.**

*1. Defines an assignment of processes to subdomains.*

RECT_GRID is used to save this information. This information is initialized in constructor of class Compute_1d.

2. The workload assigned to each process is estimated in function Compute_1d::estimate_workload(). The sub-region $[x_{i,l}, x_{i,u}]$ on ith process is saved in L[0] and U[0] of variable rect_grid of RECT_GRID type.

3. Implement a member function Compute_1d::integrate_on_subdomain() to integrate $\int_{x_{i,l}}^{x_{i,u}} \ln(x+20)e^{\sqrt{x}}dx$.

4. Implement a member function Compute_1d::sum_partial_int() to add partial integration values together. Inside this function, implement the point-to-point communication. The nonblocking send function is u_pp_isend(). Either pp_test() or pp_wait() could be used to test the completion of the nonblockingsend.

**Problem 2. Parallel Explicit Finite Difference Scheme for Solving 1D**

**Heat Equation for Graduate Students.**

Consider to solve $\begin{cases} u_t(x,t) = u_{xx}(x,t), & 0 \le x \le 2\pi, \ t > 0 \\ u(x,0) = \sin(x) & 0 \le x \le 2\pi \end{cases}$ with periodic boundary condition by the explicit finite difference scheme. Compute the solution for $t = 2.0$.

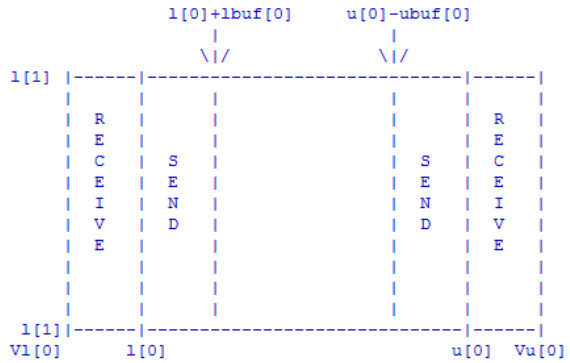
The exact solution is given by $u(x,t) = e^{-t}\sin(x)$.

Assume we use $M+1$ grid points. The grid space then is $\Delta x = \frac{2\pi}{M}$. The grid points are $x_k = k\Delta x, \ k = 0, \ldots, M$. Let $\Delta t$ be the time step size. For stability, we should satisfy $\frac{\Delta t}{\Delta x^2} \le 0.5$.

Let $v_k^n \approx u(k\Delta x, n\Delta t)$ be the approximate solution. The explicit scheme is

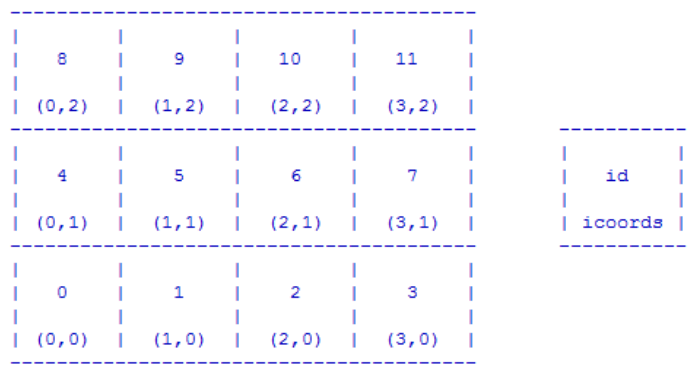
$v_k^{n+1} = v_k^n + \frac{\Delta t}{\Delta x^2}(v_{k+1}^n - 2v_k^n + v_{k-1}^n)$ for $k = 0, \ldots, M$.

Use the Demo_Pkg code to implement a parallel program using the above scheme to solve the above diffusion equation problem by arbitrary number of grid points $M+1$ using $P$ processors. Assume $M+1 \gg P$. Use buffered send and blocking receive for message passing. Use $M = 1000, 2000, 4000, 8000$ respectively to do the mesh refinement study. Compute $L_{2,\Delta x}$ error with respect to the mesh refinement. Do each of these calculations with 2, 4, 8 processors respectively. Make a table to list the wall clock time space on computation and communication respectively.




Let $[l[0], u[0]]$ be a subdomain assigned to a process. For convenience of computation, a virtual domain is defined to hold the ghost points for updating solutions defined on grid points within $[l[0], u[0]]$. This virtual domain is defined as $[vl[0], vu[0]]$. Here $vl[0] = l[0] - N * \Delta x$, and $vu[0] = u[0] + N * \Delta x$. $N$ is the number of ghost points.

*1. Defines an assignment of processes to subdomains.*



```
void scatter_states(double *soln)
{
    int          myid, side;
    int          me[3];

    MPI_Comm_rank(MPI_COMM,&myid);

    for (side = 0; side < 2; ++side)
    {
        MPI_Barrier(MPI_COMM);
        pp_send_interior_states(myid, side,soln);
        pp_receive_interior_states(myid ,(side+1)%2,soln);
    }
}

void   pp_send_interior_states(
      int          *me,
      int           side,
      double       *soln)
{
      int           myid,  dst_id, ntasks;

     MPI_Comm_rank(MPI_COMM,&myid);
     MPI_Comm_size(MPI_COMM_WORLD, & ntasks);
     dst_id = (myid + 2*side - 1);
     if(dst_id < 0)
         dst_id = ntasks-1;
     if(dst_id>= ntasks)
         dst_id = 0;
     /* Next collect soln points to be sent and call MPI_bsend() to send the data
        to the process with rank dst_id */
}

void   pp_receive_interior_states(
      int          *me,
      int          side,
     double       *soln)
{
      int           myid,  src_id,  ntasks;
       MPI_Comm_size(MPI_COMM_WORLD, & ntasks);
      MPI_Comm_rank(MPI_COMM,&myid);
      src_id = (myid + 2*side - 1);
      if(src_id < 0)
         src_id = ntasks-1;
     if(src_id>= ntasks)
         src_id = 0;

     /* Next call MPI_Recv() to receive the data
        from the process with rank src_id */
}
```

**Hand-In.** Turn in the hardcopy of all your source code, and the report which contains results and algorithmic notes on both computation and communication. Email the source code.