Project 1, due on 02/20.

Problem 1. Parallel Numerical Integration for Undergraduate Students.

Evaluate $\int_{2.0}^{5.0}\int_{2.0}^{5.0} \ln(x+2y)e^{\sqrt{x^2+y^2}}\,dydx$. Implement a parallel code using composite Gaussian quadrature rule to approximate this definite integral.

Suppose P processes are used and the integration domain $[2.0,5.0]\times[2.0,5.0]$ is partitioned into $M\times M$ grid blocks. Each of the processes is assigned with a sub-region $[x_{i,l},x_{i,u}]\times[y_{i,l},y_{i,u}]$, which is partitioned into $(M/\sqrt{P})\times(M/\sqrt{P})$ blocks. Here $i=0,1,\ldots,\sqrt{P}$-1.

We apply the 2D 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.

Use the framework implemented in parallel_trapezoidal.c. Modify the script "HPCC_1.sh" to submit your runs.

1. Use point-to-point communication, specifically, non-blocking send and blocking receive to transfer the 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 4, 16 and 64 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.

```
----------------------------------------
|         |         |         |         |
|    8    |    9    |   10    |   11    |
|         |         |         |         |
|  (0,2)  |  (1,2)  |  (2,2)  |  (3,2)  |
----------------------------------------
|         |         |         |         |          -----------
|    4    |    5    |    6    |    7    |          |         |
|         |         |         |         |          |   id    |
|  (0,1)  |  (1,1)  |  (2,1)  |  (3,1)  |          |         |
----------------------------------------           | icoords |
|         |         |         |         |          -----------
|    0    |    1    |    2    |    3    |
|         |         |         |         |
|  (0,0)  |  (1,0)  |  (2,0)  |  (3,0)  |
----------------------------------------
```

find_Cartesian_coordinates(). This function identifies the coordinates of the subdomain with process rank "id". The identification of the id and subdomain coordinates are given by

id = icoords[0] + icoords[1]*gmax[0] + icoords[2]*gmax[0]*gmax[1].

This defines the natural lexographical assignment of id numbers to the subdomains, as illustrated for the above 4x3 partition.

```
typedef struct {
       int     gmax[3];     /* # of subdomains in each dir */
       int     nn;           /* total number of nodes            */
       int     dim;
} PP_GRID;
```

```
void find_Cartesian_coordinates(
       int             id,
       PP_GRID        *pp_grid,
       int             *icoords)
{
       int     dim =  pp_grid->dim;
       int     d, G;

       for (d = 0; d < dim; d++)
       {
          G = pp_grid->gmax[d];
          icoords[d] = id % G;
          id = (id - icoords[d])/G;
       }
}          /*end find_Cartesian_coordinates*/
```

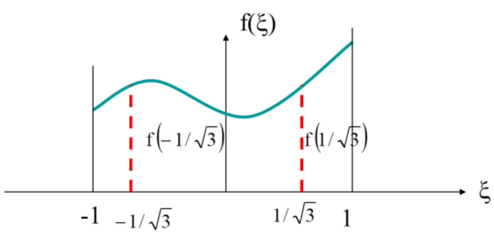
2. Compute subdomain boundaries

For process with rank "id" and coordinate (i, j) on the process grid,

$$x_{i,l} = 2.0 + i * (5.0 - 2.0)/\sqrt{P}$$

3. Quadrature rules

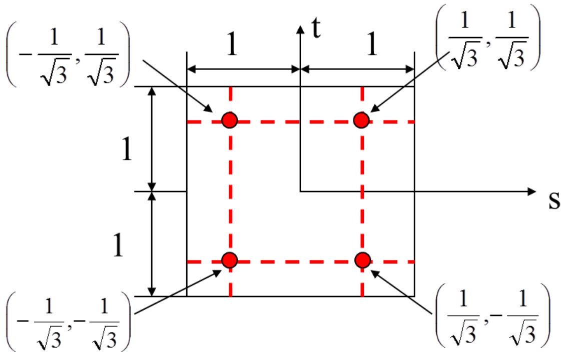
1D 2-point Gaussian quadrature rule



$$\int_{-1}^{1} f(\xi)\, d\xi \approx f(\frac{1}{\sqrt{3}}) + f(-\frac{1}{\sqrt{3}})$$

Gaussian Quadrature Rule in 2D

$$\boxed{I = \int_{-1}^{1}\int_{-1}^{1} f(s,t)\, dsdt}$$



$$
\begin{aligned}
I &= \int_{-1}^{1}\int_{-1}^{1} f(s,t)\,ds\,dt \\
&\approx \int_{-1}^{1}\left(\sum_{j=1}^{M} w_j f\left(s,t_j\right)\right) ds \\
&\approx \sum_{i=1}^{M}\sum_{j=1}^{M} w_i w_j f(s_i t_j) \\
&= \sum_{i=1}^{M}\sum_{j=1}^{M} w_{ij} f(s_i t_j)
\end{aligned}
$$

Where $w_{ij} = w_i w_j$.

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 Δt 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

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

Implement a parallel version of the scheme to solve the above problem by arbitrary number of grid points $M+1$ using P processors. Assume $M+1 \gg P$. Use non-blocking 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 4, 8, 16 processors respectively. Make a table to list the wall clock time space on computation and communication respectively. Hint: the code to compute work load assignment is work_division.c.

```
                l[0]+lbuf[0]      u[0]-ubuf[0]
                     |                  |
                    \|/                \|/
l[1] |------|----------------------------|------|
     |      |        |                  |      |      |
     |  R   |        |                  |      |  R   |
     |  E   |        |                  |      |  E   |
     |  C   |   S    |                  |  S   |  C   |
     |  E   |   E    |                  |  E   |  E   |
     |  I   |   N    |                  |  N   |  I   |
     |  V   |   D    |                  |  D   |  V   |
     |  E   |        |                  |      |  E   |
     |      |        |                  |      |      |
     |      |        |                  |      |      |
     |      |        |                  |      |      |
  l[1]|------|----------------------------|------|
Vl[0]      l[0]                          u[0]   Vu[0]
```

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.

The communication routine looks like the following:

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

    MPI_Comm_rank(MPI_COMM,&myid);
    find_Cartesian_coordinates(myid,pp_grid,me);

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

void    pp_send_interior_states(
     int          *me,
     int           side,
     double        *soln)
{
     int            myid, him[3], dst_id;

     MPI_Comm_rank(MPI_COMM,&myid);
     dst_id = neighbor_id(him,me,0,side,pp_grid);

     /* Next collect soln points to be sent and call MPI_Isend() to send the data
        to the process with rank dst_id */
}

void    pp_receive_interior_states(
     int          *me,
     int          side,
     double       *soln)
{
     int            myid, him[3], src_id;
     MPI_Comm_rank(MPI_COMM,&myid);
     src_id = neighbor_id(him,me,0,side,pp_grid);

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

```
int     neighbor_id(
        int             *him,
        int             *me,
        int             dir,
        int             side,
        PP_GRID         *pp_grid)
{
        int             *G = pp_grid->gmax;
        int             i, dim = pp_grid->dim;

        for (i = 0; i < dim; i++)
            him[i] = me[i];

        him[dir] = (me[dir] + 2*side - 1);
        if (him[dir] < 0)
            him[dir] = G[dir] - 1;
        if (him[dir] >= G[dir])
            him[dir] = 0;

        return domain_id(him,G,dim);
}               /*end neighbor_id*/

int domain_id(
        int             *icoords,
        int             *G,
        int             dim)
{
        int             tmpid;
        int             i;

        tmpid = icoords[dim-1];
        for (i = dim-2; i >= 0; i--)
            tmpid = icoords[i] + G[i]*tmpid;
        return tmpid;
}           /*end domain_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.