

# CSE-557 Home Work No. 3

Anirudh Modi and Anupam Sharma  
Graduate Students, Penn State University

---

We have developed a library which shall be very useful in writing a parallel code for solving problems with iterative methods. The subroutines in the library are pretty general and are not constrained by *the number of processors* used for working in parallel, *grid size* and *relaxation technique*. The functions in the library have been made as robust as possible so as to take care of any absurd data input from the user. The grid discretization (among the processors) and message passing is all handled by the library routines and the user has almost no way to meddle with it. Two groups of Communicator have been used, one each for row and column communication.

The library subroutines can perform the following without the user having to worry about parallelizing etc.

- New Communicator definition and initialization, which is then used to do all the message passing.
- Memory allocation in the processors given the total grid size and the total number of processors.
- Message passing to/from each neighbor. Any number of cells can be passed by specifying the number of ghost cells in each direction.

## 1 The Library Structure

The library has been structured in the following fashion. All the files are under the main directory **GridComm**. GridComm has following subdirectories for convenient

storage and easy accessibility :

- **lib** : This directory contains the compiled library archive *libGC.a* which is to be linked to the test programs.
- **include** : This directory contains the following '.h' files which contain the declarations of the functions etc. - *GCcommon.h*, *GCerror.h* and *GCinclude.h*. In particular, *GCerror.h* contains the definitions for the functions which return error message if there is some error in any of the following processes : printing, memory allocation and memory freeing.
- **src** : Contains all the source files of the library. All the files are listed below with a brief description of their use.

1. *GCinit* - This is the initialization of the communicators - GridComm and MPI. GridComm is a new communicator defined in order to differentiate it from the MPI\_COMM\_WORLD. The communicator has two groups one each for communicating east-west and north-south. Arguments to GCinit :

- *int argc* - count of number of command line arguments
- *int args* - the command line arguments

2. *GCpartitionGrid* - This is useful for partitioning the grid among the various processors. Actually, the full grid is never read, so the term 'partition' is a little misleading. The initial grid and the boundary conditions are read in separate chunks by each processor. GCpartitionGrid just correlates the *local indices* to the *global indices* of the grid. Arguments to GCpartitionGrid :

- *int gridSize[2]* :: global grid size in 'x' and 'y' directions (input)
- *int procTotal[2]* :: number of processors in 'x' and 'y' directions (input)

- *int globalStartIndex[2]* :: global start index for each processor (output)
  - *int globalEndIndex[2]* :: global end index for each processor (output)
  - *int procIndex[2]* :: processor indexing in ‘x’ and ‘y’ directions (output)
3. *GCgetGridDataPtr* : Gets the pointer to the grid data
4. *GCinitGrid* : Does memory allocation for grid on each processor taking care of the ghost cells. Each grid size is  $(nx+ghostx) \times (ny+ghosty)$ . This does the memory allocation for the grid on each processor. Arguments to *GCinitGrid*.
- *GCgrid \*\*grid* :: double pointer to the Grid structure (since memory allocation is to be done) (input/output)
  - *ghostSize[2]*, *procIndex[2]*, *globalStartIndex[2]* & *globalEndIndex[2]* : standard definitions
  - *double (\*BCfunc)(double u, double v)* :: function for boundary conditions passed as an argument; ‘u’ and ‘v’ are calculated here which give the correlation between global and local grid (output)
5. *GCinitBC* : To specify the boundary conditions on each processor. Boundary conditions are input by the user in the main program in the parametric space, as functions of ‘x’ and ‘y’. Arguments to *GCinitBC*.
- *GCgrid \*grid* :: pointer to the Grid structure ‘GCgrid’ (output)
  - *double (\*BCfunc)(double u, double v)* :: function for boundary conditions passed as an argument (input)
  - *int \*procTotal* :: total number of proc. in ‘x’ and ‘y’ directions (input)
  - *\*globalStartIndex* & *\*globalEndIndex* :: global start/stop indices in ‘x’ and ‘y’ directions (input)

6. *GCupdateGrid* : This is function which takes care of all the memory passing among processors. *GCgrid \*grid* is the only input/output. Works for any number of ghost cells as prescribed by the user in the main program.
  7. *GCfreeGrid* : To free the memory allocated to 'grid' on each proc. Arguments :
    - *\*grid* :: pointer to the Grid structure 'GCgrid'
  8. *GCerror* - Contains routines which check if there is any error in printing, memory allocation or memory freeing. If there is any error these provide a safe exit to the program with the an 'understandable' error message.
  9. *GCutilities* - For the present this contains only one routine which checks if the processor is the master processor or not. This can be used to contain other routines which may be handy.
  10. *GCnorm* : This shall be used to calculate the error norm to check the convergence.
  11. *GCfinalize* - This just finalizes the MPI.
- **obj** : This just contains the object files obtained after compilation from the source files of the library routines.
  - **test** : For the present the Jacobi relaxation has been implemented in 'test1.c' file. The relaxation can be done on any grid and any number of processors.

## 2 Message Passing

In order to check if the message passing was working fine, each grid point was initialized with it's lexicographic value. Thus the numbering was from 1 - nx\*ny. For this case nx=ny=10 was chosen and the message passing was done for 2\*2 processors. As can be seen from the following, the message passing is working fine. The ghost cells which correspond to the boundary are not getting altered (as there

is no communication there) and the ghost cells which come due to the partitioning, are updated.

Before performing the sends and receives

	0	0	0	0	0			0	0	0	0	0	
0	91	92	93	94	95	0	0	96	97	98	99	100	0
0	81	82	83	84	85	0	0	86	87	88	89	90	0
0	71	72	73	74	75	0	0	76	77	78	79	80	0
0	61	62	63	64	65	0	0	66	67	68	69	70	0
0	51	52	53	54	55	0	0	56	57	58	59	60	0
	0	0	0	0	0			0	0	0	0	0	
	0	0	0	0	0			0	0	0	0	0	
0	41	42	43	44	45	0	0	46	47	48	49	50	0
0	31	32	33	34	35	0	0	36	37	38	39	40	0
0	21	22	23	24	25	0	0	26	27	28	29	30	0
0	11	12	13	14	15	0	0	16	17	18	19	20	0
0	1	2	3	4	5	0	0	6	7	8	9	10	0
	0	0	0	0	0			0	0	0	0	0	

After performing the sends and receives

	0	0	0	0	0			0	0	0	0	0	
0	91	92	93	94	95	96	95	96	97	98	99	100	0
0	81	82	83	84	85	86	85	86	87	88	89	90	0
0	71	72	73	74	75	76	75	76	77	78	79	80	0
0	61	62	63	64	65	66	65	66	67	68	69	70	0
0	51	52	53	54	55	56	55	56	57	58	59	60	0
	41	42	43	44	45			46	47	48	49	50	0
	51	52	53	54	55			56	57	58	59	60	
0	41	42	43	44	45	46	45	46	47	48	49	50	0
0	31	32	33	34	35	36	35	36	37	38	39	40	0
0	21	22	23	24	25	26	25	26	27	28	29	30	0
0	11	12	13	14	15	16	15	16	17	18	19	20	0
0	1	2	3	4	5	6	5	6	7	8	9	10	0
	0	0	0	0	0			0	0	0	0	0	