Wednesday, 05/02/12:

HW 5, 6, 7 approach:

HW5 = serial C = <u>lecture</u> today

idea of coding in C is to test <u>everything</u> fully in Matlab

$\Rightarrow$ <u>lab</u> today = Matlab code

"serial" here: framework of parallel code =

allocations of l_u (not u), use MPI_Wtime, Makefile

with mpicc, utility functions like parallel_dot

but: - run with one process only

- Ax() function in serial

HW 6 = parallel C <u>with</u> all performance studies done

<u>and</u> results assembled in tables and plots

also report written

<u>HW7 = feedback on</u> report, tables, and figures $\Rightarrow$ revision

Start with Matlab so far:

driver_ge.m and driver_cg.m

(same except for u=A\b vs. u=pcg(...))

setupA.m to set up sparse matrix A

$\Rightarrow$ Gauss elimination for sparse matrix A

CG method using sparse matrix A

We see that Gauss runs out of memory for some N value

CG might allow for one more N value

Two reasons for matrix-free implementation now:

- we might be able to solve for a larger N
- since setting up <u>sparse</u> matrix in C is non-trivial, it is actually easier to do matrix-free.

⇒ today: want a third version of Matlab code for CG using matrix-free function Ax.m
still use pcg.m in Matlab!

What do we need to do/have for C?

CG → download cg.c and use it
→ need to read and understand interface

Makefile: OBJS = ... cg.o
EXECUTABL = poisson

submisson script: change its name ...poisson...
change name to Poisson
./poisson N tol maxit

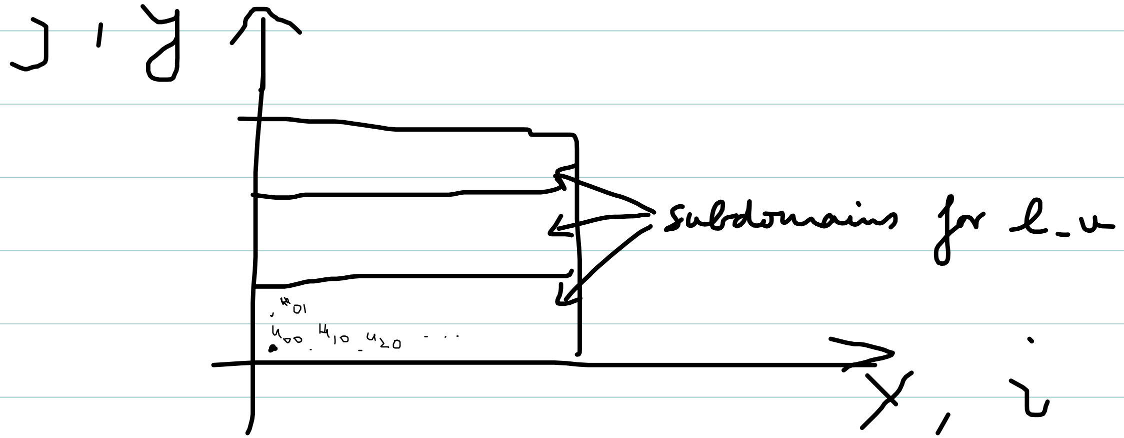
(for simplicity, you can simply use tol $= 10^{-6}$ and maxit = 50,000)

main() program: follow outline of driver_cg.m
→ understand interface to cg() and supply all needed information

Ax function !

We looked through cg() and noticed several things that we need to supply or set.

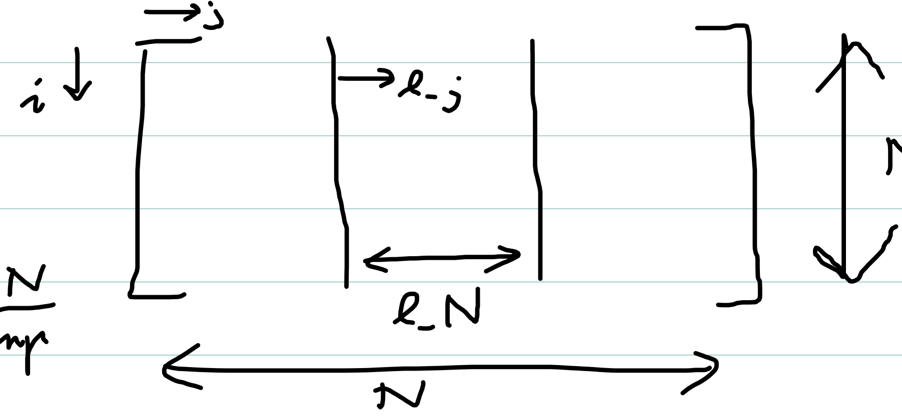
Datastructure:



$$\Longleftrightarrow \quad U = (u_{ij}) = \left[\begin{array}{c|c|c} u_{00}\ u_{01} \cdots & & \\ u_{10} & -l\_u & l\_u \\ u_{20} & & \\ \vdots & & \end{array}\right]$$

Notice PDE viewpoint with Ω split into subdomains is transpose of matrix viewpoint.

In matrix viewpoint



$l\_N = \frac{N}{np}$

CG for $Au=b$ has dimension $n = N^2$

and therefore $l\_n = \frac{n}{np} = N * l\_N$

Main program:

l_u = (double*) malloc(l_n * sizeof(double)

l_b = ...

l_p = ...

l_q = ...

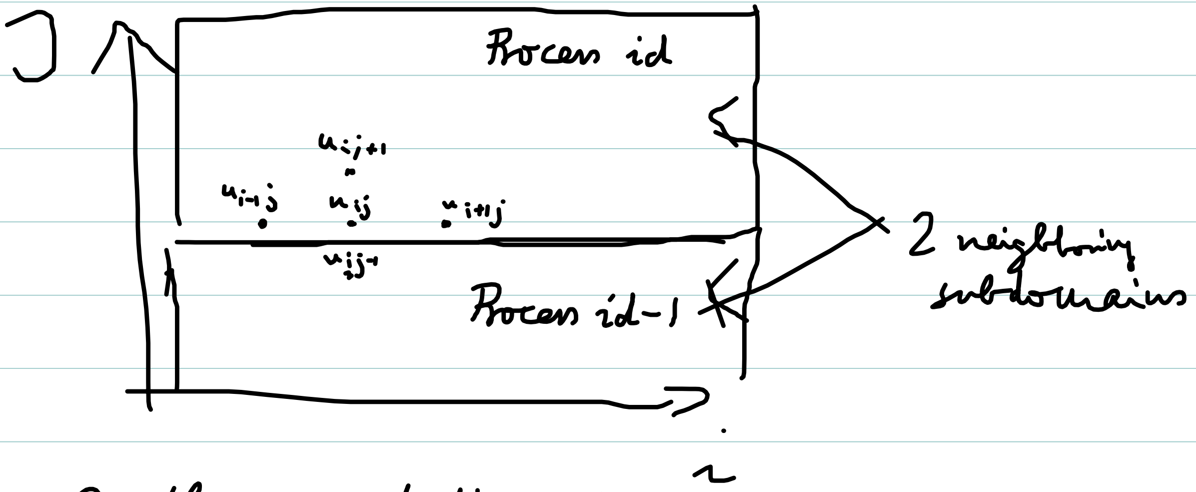
⇒ only exactly 4 vectors of large size N × l_N

$$gl = (double)\, malloc(\underbrace{(l\_n / l\_N)}_{(= N \text{ in 2-D})} * sizeof(double))$$

gr = ...

⇔ amount of data to communicate between neighboring processes

Understand this from PDE viewpoint of FD stencil



In the computation of u_{ij} on Process id, we need data $u_{i-1,j}$, $u_{i+1,j}$, and $u_{i,j+1}$ that is also on Process id, and $u_{i,j-1}$ that is on Process id-1

⟹ We need to communicate between neighboring processes, l_n/l_N = N in 2-D double numbers

⟹ gl, gr are vectors to hold the data that is communicated (length l_n/l_N) and we need idleft = id-1 and idright = id+1

More precisely, do this:

```
if ( id > 0 )
      idleft = id - 1
else
      idleft = MPI_PROC_NULL
if (id < np-1)
      idright = id+1
else
      idright = MPI PROC_NULL
```

Some ideas on code:

```
h = 1.0/ ((double)(N+1))
x = (double*) malloc ( N * sizeof(double))
y = . . .
for (i=0; i<N; i++)   x[i] = h*((double)(i+1))
for(j . . . .        )  y[j] =       - - -     j+1
for (l_j=0 ; l_j < l_N ; l_j++)
   for (i=0; i<N; i++)
      j = l_j + l_N * id
      l_r[i+N*l_j] = (h*h) * f(x[i], y[j])
```

Heart of a matrix-free code: Ax()
Here: serial version!

```
void Ax (double *v, double *u, int l_n,
  int l_N, int N, int id, int idleft,
  int idright, int np , MPI_Comm comm,
  double *gl, double *gr)
```

Want to implement: $(i+Nj)$ component of $v=Au$

$$\Leftrightarrow \quad v_{ij} = -u_{i,j-1} - u_{i-1,j} + 4u_{ij} - u_{i+1,j} - u_{i,j+1}$$

in C:

```
double tmp
int i, j

for (j=0; j<N ; j++) {
   for (i=0; i<N; i++) {
                  tmp = 4.0*u[i    +N*j    ]
        if (j>0)  tmp -=   u[i    +N*(j-1)]
        if (i>0)  tmp -=   u[(i-1)+N*j    ]
        if (i<N-1) tmp -=  u[(i+1)+N*j    ]
        if (j<N-1) tmp -=  u[i    +N*(j+1)]
        v[i+N*j] = tmp
   }
}
```