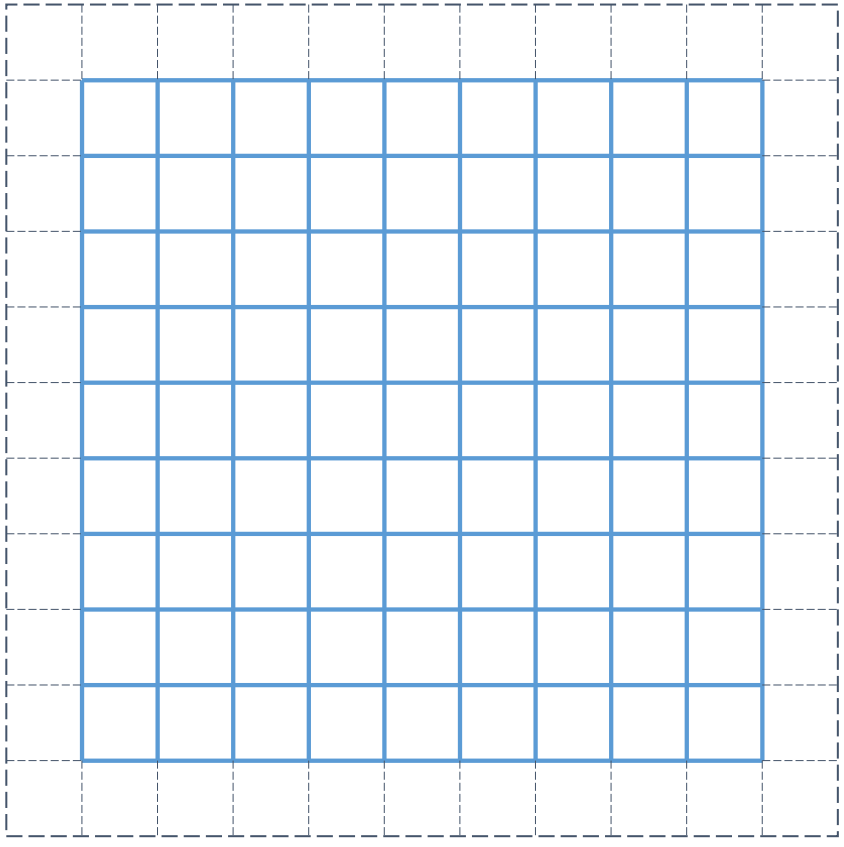
High Performance Computing: Tools and Applications

Edmond Chow
School of Computational Science and Engineering
Georgia Institute of Technology

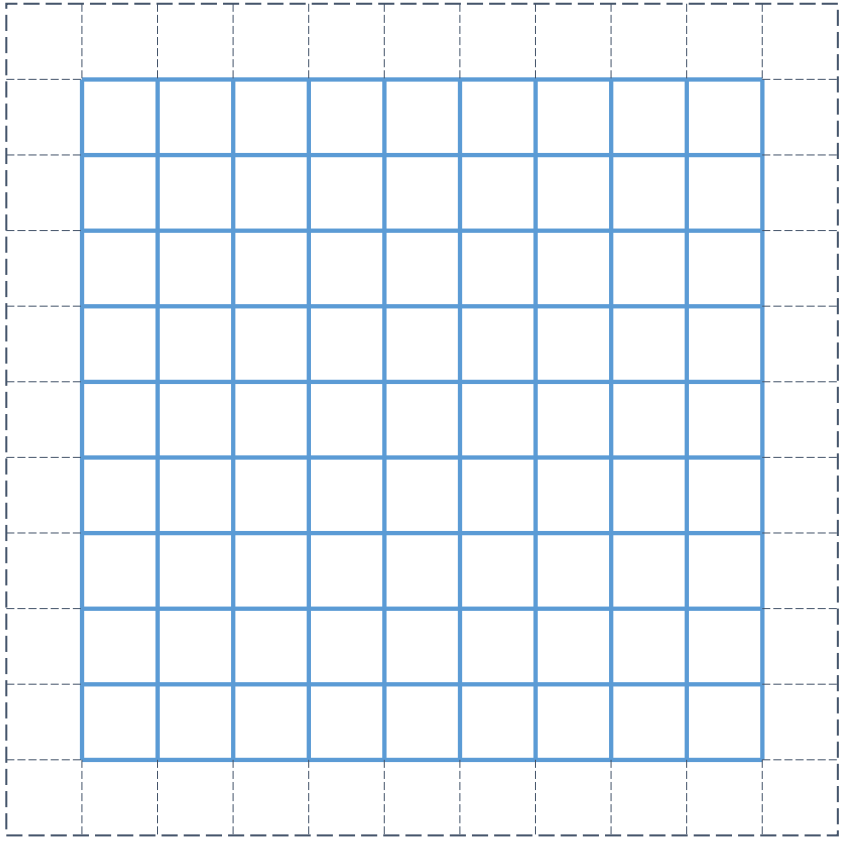
Lecture 15

Numerically solve a 2D boundary value problem



- Example: temperature distribution in a square room, given heat sources and sinks inside the room;
  many more complicated and realistic examples with similar communication and computation patterns
- Solve for the unknown (e.g., temperature) at each grid point
- Values on the boundary (dotted lines) are known.
  We will use zero on the boundary for simplicity.

Numerically solve a 2D boundary value problem



- A discrete formula relates the value at a grid point with the values at its four neighbors
- For temperature, the formula is

$$4u_C = u_N + u_W + u_S + u_E + f_C$$

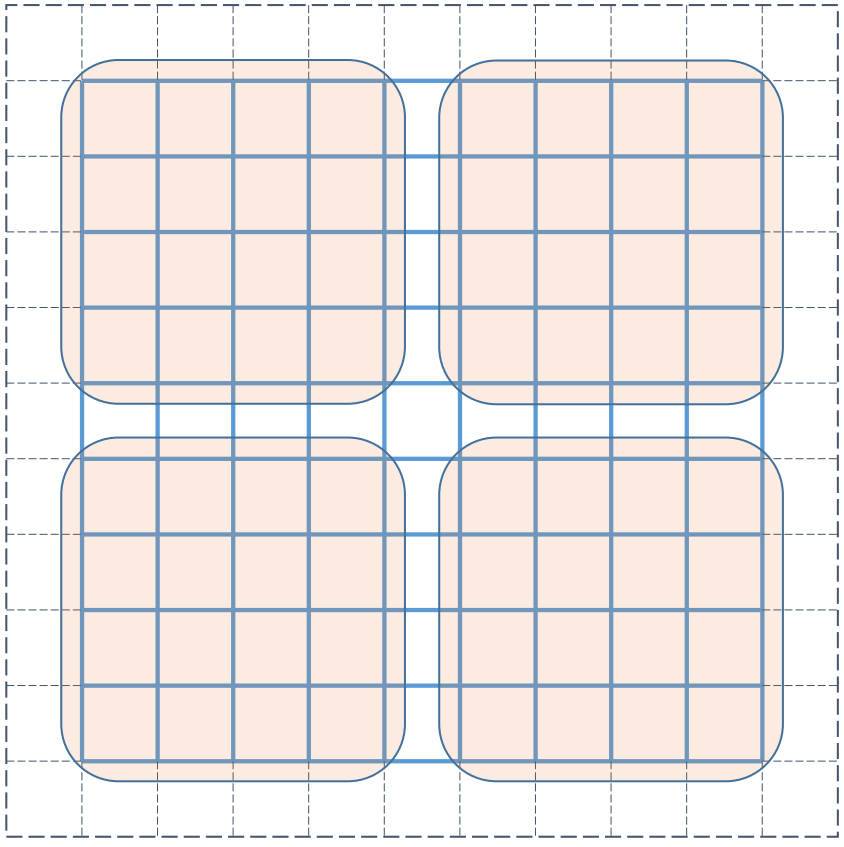
where f_C is the value of the source or sink at point C, and the other subscripts are the north, west, south, and east points.

Non-parallel solution method

Solve the matrix equation

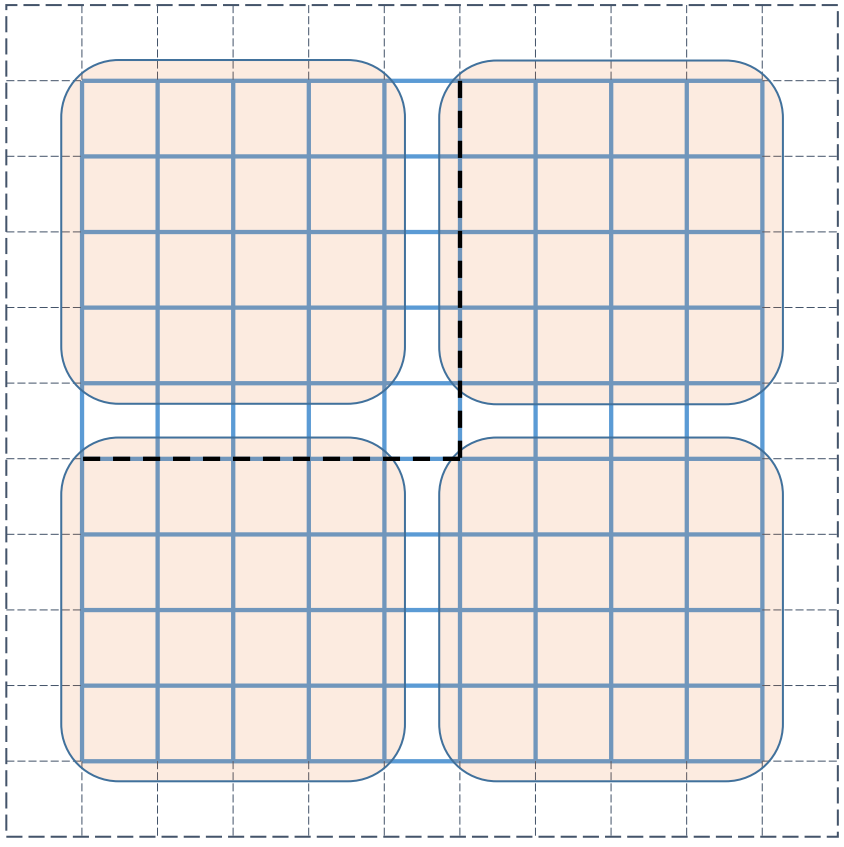
$$Au = f$$

Parallel Solution



- To solve the problem in parallel, partition the unknowns into subdomains, one for each process

Block Jacobi method



- Each subdomain is a boundary value problem
- Note that the boundary values are generally the unknowns stored in other processes
- Boundary for top-left subdomain is shown

Block Jacobi method

In a distributed computation, each process performs:

```
For s = 0, 1, 2, ... until convergence
  Send boundary values needed by other processes
  Receive boundary values needed by this process
  Solve local boundary value problem
  Compute the residual norm and stop if converged
Endfor
```

Important: no processor stores the entire global problem.
This allows very large problems to be solved by distributing it across many compute nodes.

Local boundary value problem

This is a sparse linear matrix equation to be solved

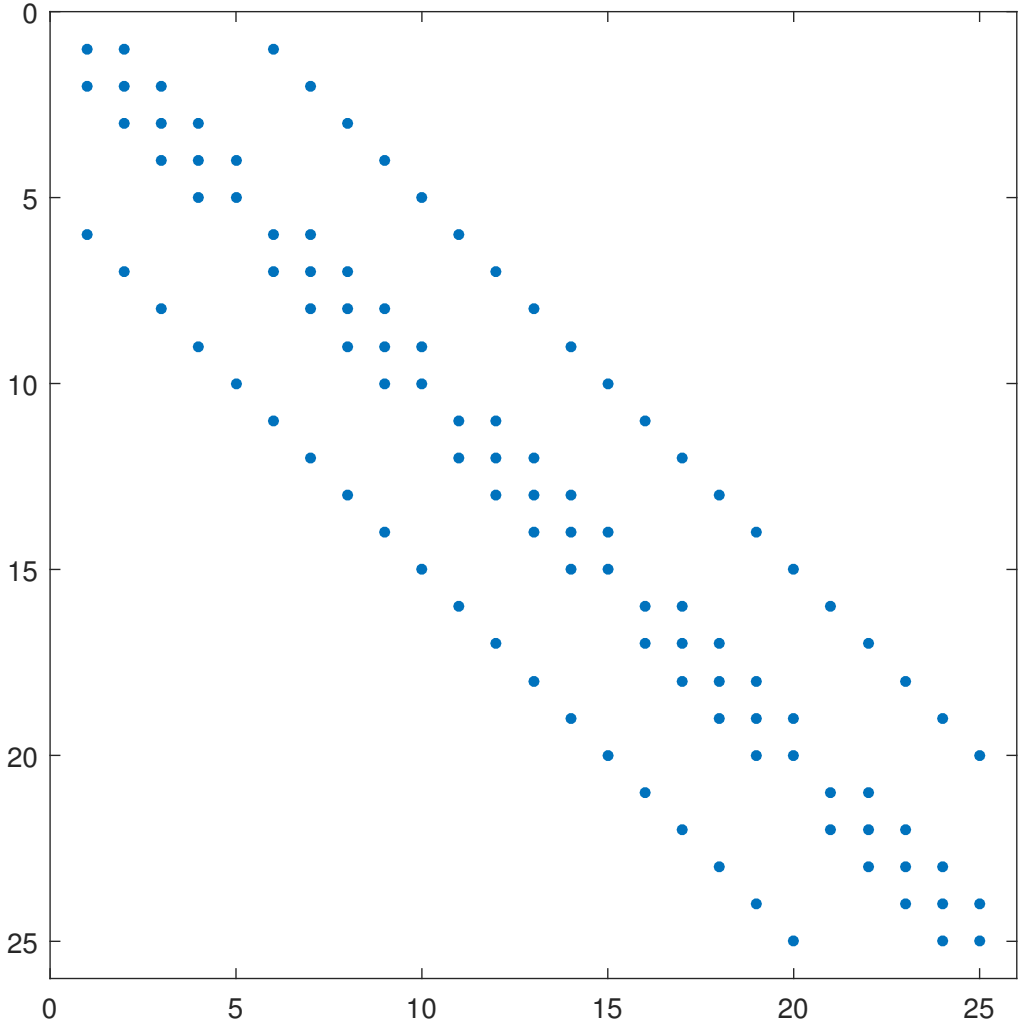
$$A_{local} u_{local} = f_{local}$$

where the number of equations is equal to the number of local grid points (or unknowns), assuming the f_{local} has been modified to incorporate the boundary values from other processors

Need to assemble the sparse matrix, and solve the equations, e.g., using MKL. (Hint: use CSR format, to be described in the next lecture.)

Nonzero pattern of A_{local}

25×25 sparse matrix



Vector f_{local}

The vector f_{local} is constructed from the local part of the global f, but must also be modified depending on the boundary values

Suppose we have the equation

$$-u_N - u_W + 4u_c - u_S - u_E = f_C$$

but that u_E is on the boundary and has value α.
Then the equation is modified as

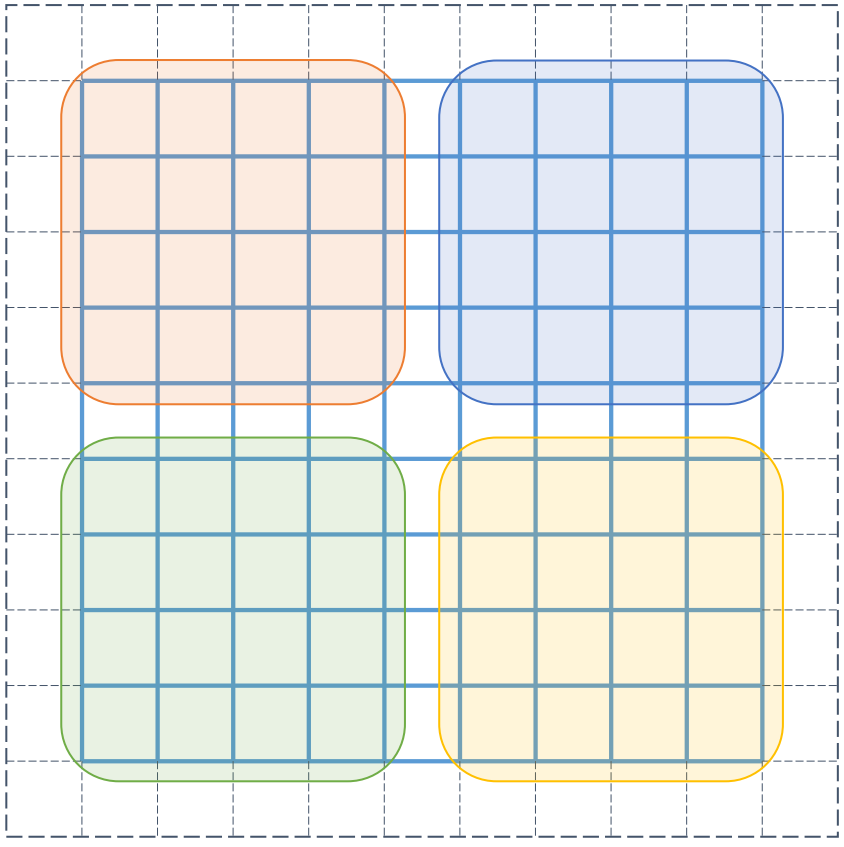
$$-u_N - u_W + 4u_c - u_S = f_C + \alpha$$

i.e., the boundary value is added to the appropriate component of f_{local}

Solving the local equations $A_{local} u_{local} = f_{local}$

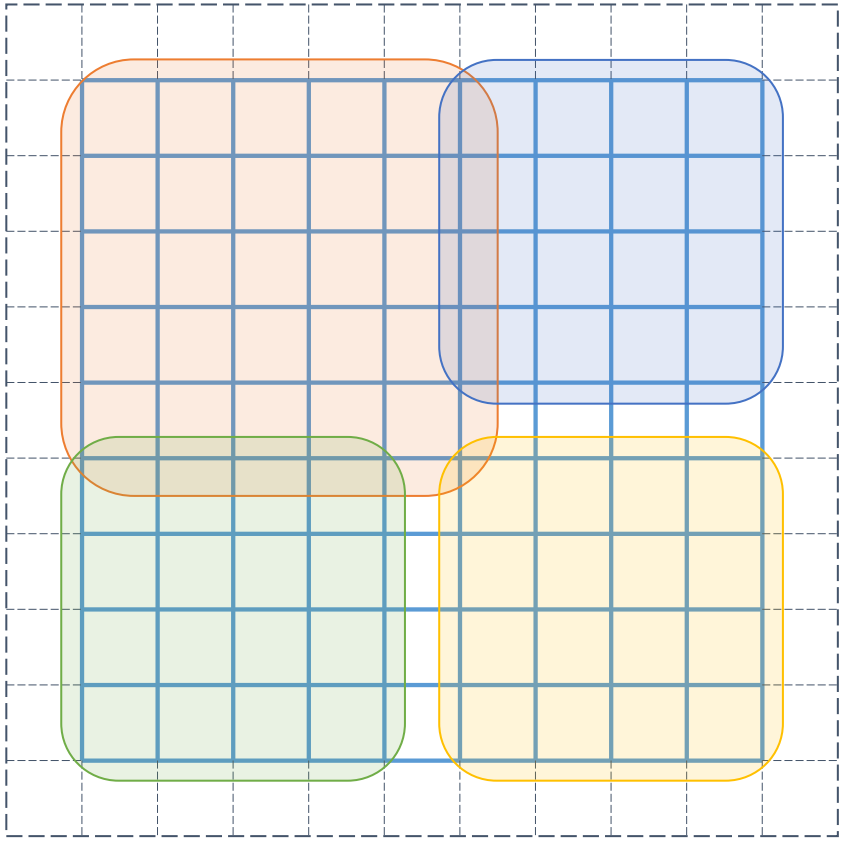
- For mini-project 3, simplest approach is to use a “direct” method implemented in MKL
- MKL provides the *PARDISO* method. It can be accessed through its native interface or the DSS interface
- Example programs
  - `dss_unsym_c.c`
  - `dss_sym_c.c`
  - `pardiso_unsym_c.c`
  - `pardiso_sym_c.c`
- Note, our matrices are symmetric and positive definite, but you can also use the unsymmetric interface functions if you wish

Jacobi-Schwarz method (faster convergence)



- Partition the unknowns as before

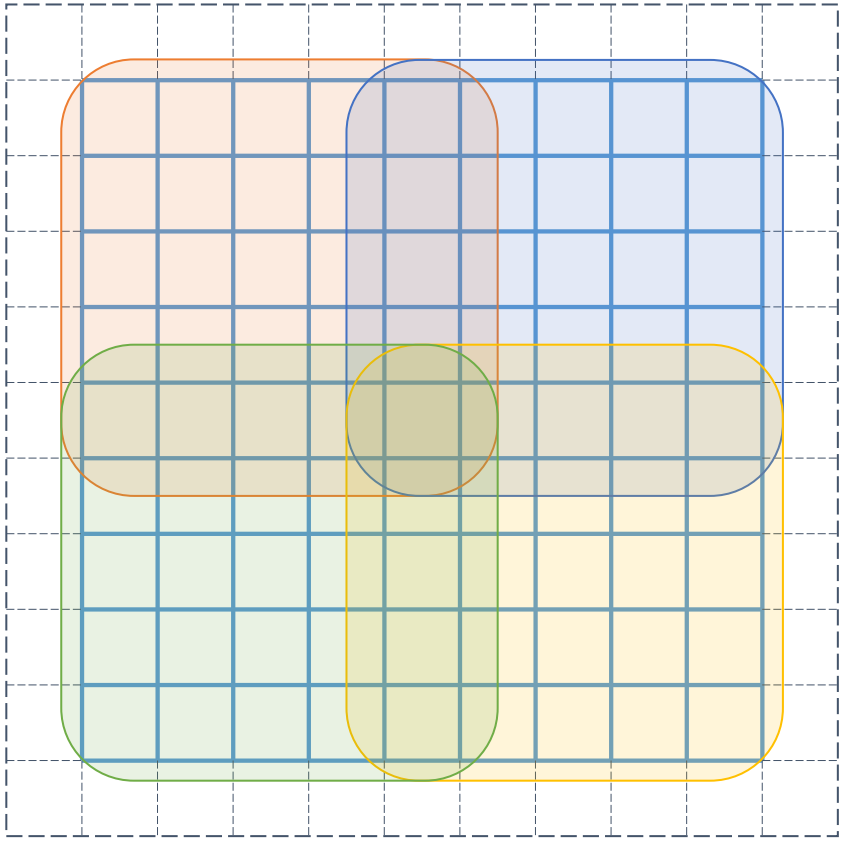
Jacobi-Schwarz method (faster convergence)



Top left subdomain is grown by 1 grid spacing in each direction

- Partition the unknowns as before
- Grow the subdomains (grow by 1 or more in each direction except at the real boundaries)

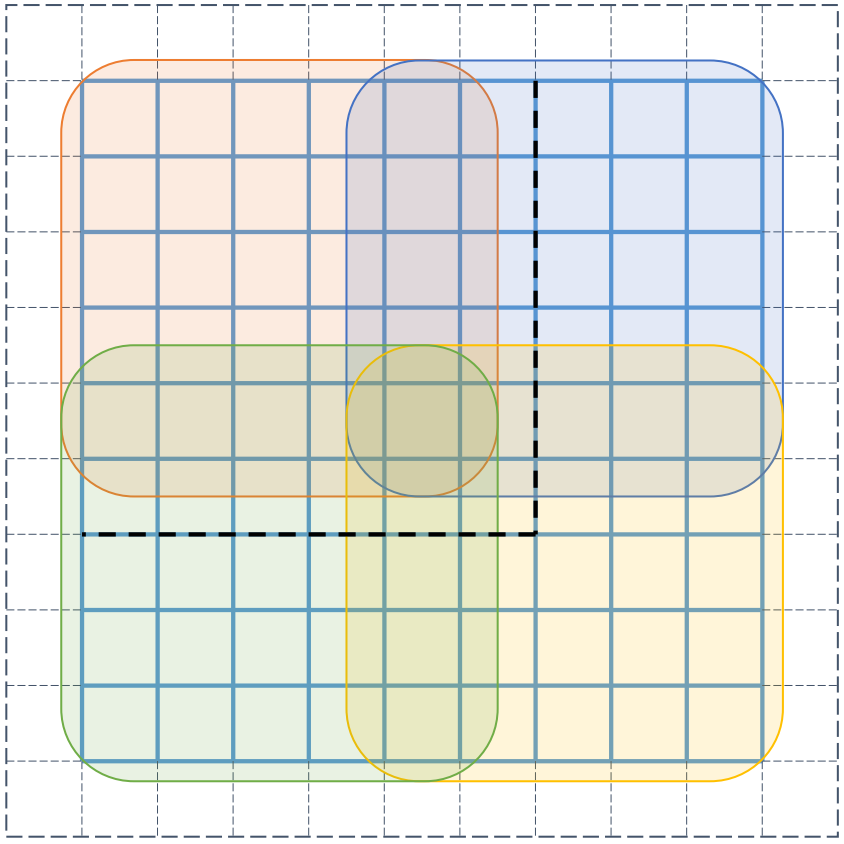
Jacobi-Schwarz method (faster convergence)



All subdomains are grown by 1 grid spacing

- Partition the unknowns as before
- Grow the subdomains (grow by 1 or more in each direction except at the real boundaries)
- Now the subdomains overlap

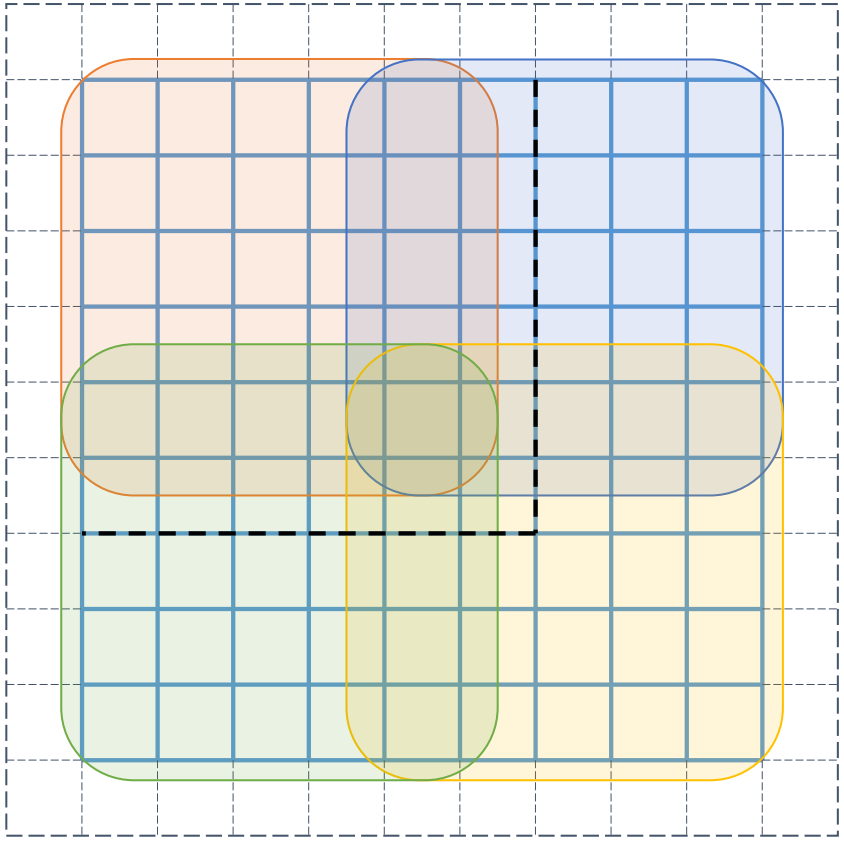
Jacobi-Schwarz method (faster convergence)



Boundary for top-left subdomain is shown

- Partition the unknowns as before
- Grow the subdomains (grow by 1 or more in each direction except at the real boundaries)
- Now the subdomains overlap
- Solve the boundary value problem on each subdomain

Jacobi-Schwarz method (faster convergence)



Boundary for top-left subdomain is shown

- Note that some points (which will be used as boundary points) are defined by more than one subdomain.
  What value should be used?
- Could use an average value
- Could use the value defined by the “owner” subdomain, i.e., original partitioning before growing subdomains (better choice)

Jacobi-Schwarz method (faster convergence)

Each process performs:

```
For s = 0, 1, 2, ... until convergence
  Send boundary values needed by other processes
  Receive boundary values needed by this process
  Solve local boundary value problem
  Only store the part of the solution corresponding to
    the original subdomain (before growing)
  Compute the residual norm and stop if converged
Endfor
```

Again, no process stores the global problem

Matlab examples

- `jacobi_schwarz.m`

Mini-Project 3

- Write a MPI program for solving a 2D Poisson boundary value problem using the Jacobi-Schwarz method
- Use a single thread per MPI process (i.e., no multithreading, including in MKL function calls)
- Input f is zero, i.e., solution is zero
- Initial approximation is random, distributed between -0.5 and 0.5.

Stopping criterion

- For mini-project 3, the solution is the zero vector
- After each iteration, compute the 2-norm of the error, and stop if this quantity is small enough, i.e.,

$$\left(\sum e_i^2\right)^{1/2} < \text{tolerance}$$

- Hint: use `MPI_Allreduce` to sum a vector across all MPI processes and put the result on all processes
- Another hint, `MPI_Wtime` can be used to measure wall-clock time

Mini-Project 3: Grading

- 0-7 points: correctness and programming style
  - must use scalable data structures (no data structures with size proportional to the global problem size)
- 0-2 points: execution time
- 0-2 points for error norm graphs: Plot the error norm (log scale) vs. iteration count (for 1 to 100 iterations) for:
  - 10 by 6 processor mesh, and 100 by 100 local grid, grow=0
  - 10 by 6 processor mesh, and 100 by 100 local grid, grow=1
  - 10 by 6 processor mesh, and 100 by 100 local grid, grow=10
- 0-2 points: table of time per iteration and number of iterations to reduce the error norm to less than 10^{-3}:
  - 1 by 1 processor mesh, and 700 by 700 local grid (no grow)
  - 2 by 2 processor mesh, and 350 by 350 local grid, grow = 1
  - 4 by 4 processor mesh, and 175 by 175 local grid, grow = 1
  - 7 by 7 processor mesh, and 100 by 100 local grid, grow = 1
- 0-2 points: speedup graph (relative to 1 process) for above 4 cases

Mini-Project 3: Due date

Sunday, Oct. 30, at 10 p.m.