Terminology

Scaling: function as a number of processing elements (cores, nodes, computers)
- x-axis: number of processing elements
- y-axis: performance
- Linear scaling
- Sublinear scaling, as long as performance goes up with number of cores
- Superlinear scaling possible, happens because of cache effects
- Difficult to have perfect scaling

Performance?

Speedup = (sequential runtime) / (parallel runtime)
- > 1 ideal

Strong scaling: add more processors with fixed size problem, speedup goes up
- FFT of 1M points
  - 1 processors: 1M points / processors
  - 10 processors: 100,000 points / processor
  - 1M processors: 1 point / processor
Work per processor goes down, performance goes up (hard)

Weak scaling: fix amount of work per processor, add more processors, performance continues to go up
- FFT of 1M points
  - 1 processor: 1M points / processor
  - 2 processors: 2M points / processor
Gauss-Seidel

Jacobi

Systems of Linear Equations

Ax = b (solve for x)

\[ a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \]

\[ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \]

\[ \vdots \]

\[ a_{nn}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \]

PDEs, ODEs

Gaussian elimination/row elimination/... = \( O(n^3) \)

- \( n \) rows to complete * \( n \) rows to scale and subtract * \( n \) per row

Jacobi - Equation 4.4

\[ x_i(t + 1) = -\frac{1}{a_{ii}} \left[ \sum_{j \neq i} a_{ij}x_j(t) - b_i \right] \]

- Limit from \( t \rightarrow \infty \) of \( x_i(t + 1) \) equals equation 4.3 means it converges and OK to do approximation
- One iteration = \( O(n) \)
- To solve entire system = \( O(n^2) \), but how many iterations?
  - Usually set number of iterations to a constant
- All to all communication

Gauss-Seidel

\[ x_i(t + 1) = -\frac{1}{a_{ii}} \left[ \sum_{j < i} a_{ij}x_j(t + 1) + \sum_{j > i} a_{ij}x_j(t) - b_i \right] \]

- Breaks up into indices less than and indices greater than \( i \) to make it converge faster by using estimates already computed
- \( O(n^2) \), may take fewer iterations

Gaussian elimination

- Can parallelize mult + sub to reduce runtime to \( O(n^2) \)

Jacobi

- Can parallelize all multiplications, can’t parallelize summation
- On \( O(n) \) processors: run each index on one processor, runtime \( O(n) \).
- On \( O(n^2) \) processors: use sum tree for summation, runtime \( O(\log n) \).
- On \( O(n^3) \) processors: doesn’t really help.

Gauss-Seidel

- Basically has no parallelism even though it converges faster

Easier to achieve than strong scaling
Poisson’s Equation
- Uses: solve steady state temps inside 2D metallic sheet given boundary conditions
- Sparse, communication pattern is four diagonals because only need to talk to nearest neighbor
- Can break up in sections, assign processor to each section, and only communicate on the boundaries

Communication pattern depends on topology. **Processors’ interconnect topology may not match algorithm topology.**

Overrelaxation: converges faster
- Communication pattern fine since communicating with self
- Can change gamma to weight self more as it converges

**Weather Code**

Weather predictions models haven’t changed a lot
- Now have boundary conditions, includes poles, finer-grained/smaller grid
- Tertiary variables

Discretization of the world - ignore the poles, wrap surface of the earth around a cylinder and cutting into cubes
- Cubes are not all the same size because of shape of the Earth

Time also discretized

Parallelization
- Each processor gets a chunk of blocks to minimize communication to just at the boundary
- Temp and pressure of one cube of space is only affected by six neighboring cubes
  - N-body: force at a distance, every body is affected by all others

Mapping to Ultracomputer with butterfly communication pattern

Mapping to 3D mesh of processors
- No wraparound - long communication paths at boundaries

Improvements:
- Reduce timestep
- Add secondary and tertiary parameters
- Larger cubes -> less communication overhead due to volume vs surface area growth
  - Leads to fewer processors, but more processors means you can do more in parallel
  - More like weak scaling due to communication overhead
Synchronization - processors must all reach a point before continuing on
  ● Remove barrier: wouldn’t work since some processors might work at the wrong timestep
  ● Stronger synchronization than necessary, easier to write the code this way
  ● Map out dependency of iterations, some can work at faster timesteps than others
    ○ Footnote 22