Solving Linear Equations Iteratively
(see also PP, sec. 6.3 & 10.4)

Our Matrix-Vector Multiply can be used to solve linear equations

\[
\begin{align*}
  a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \ldots + a_{0,n-1}x_{n-1} &= b_0 \\
  a_{10}x_0 + a_{11}x_1 + a_{12}x_2 + \ldots + a_{1,n-1}x_{n-1} &= b_1 \\
  a_{20}x_0 + a_{21}x_1 + a_{22}x_2 + \ldots + a_{2,n-1}x_{n-1} &= b_2 \\
  \vdots \\
  a_{m-1,0}x_0 + a_{m-1,1}x_1 + a_{m-1,2}x_2 + \ldots + a_{m-1,n-1}x_{n-1} &= b_{m-1}
\end{align*}
\]

can often be solved iteratively. A sufficient condition is diagonal dominance: The diagonal term is greater than the sum of the abs of the off-diagonal terms.

First symbolically solve the \(i\)th equation for \(x_i\):

\[
\begin{align*}
  x_0 &= (b_0 - (a_{01}x_1 + a_{02}x_2 + \ldots + a_{0,n-1}x_{n-1}))/a_{00} \\
  x_1 &= (b_1 - (a_{10}x_0 + a_{12}x_2 + \ldots + a_{1,n-1}x_{n-1}))/a_{11} \\
  x_2 &= (b_2 - (a_{20}x_0 + a_{21}x_1 + \ldots + a_{2,n-1}x_{n-1}))/a_{22} \\
  \vdots \\
  x_{m-1} &= (b_{m-1} - (a_{m-1,0}x_0 + a_{m-1,1}x_1 + \ldots + a_{m-1,n-2}x_{n-2}))/a_{m-1,m-1}
\end{align*}
\]

Each \(x\) on the LHS depends on all other \(x\)’s except itself.

The RHS entails a matrix-vector multiplication \(CX\).

Our Matrix-Vector Multiply can be used to solve linear equations

\[
\begin{align*}
  a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \ldots + a_{0,n-1}x_{n-1} &= b_0 \\
  a_{10}x_0 + a_{11}x_1 + a_{12}x_2 + \ldots + a_{1,n-1}x_{n-1} &= b_1 \\
  a_{20}x_0 + a_{21}x_1 + a_{22}x_2 + \ldots + a_{2,n-1}x_{n-1} &= b_2 \\
  \vdots \\
  a_{m-1,0}x_0 + a_{m-1,1}x_1 + a_{m-1,2}x_2 + \ldots + a_{m-1,n-1}x_{n-1} &= b_{m-1}
\end{align*}
\]

Matrix Equation

- The previous equation in matrix form is:
  \[
  X = B - CX
  \]
- Starting with an initial approximation to \(X\) (say all 0’s), we repeatedly iterate:
  \[
  X := B - CX;
  \]
  until the change in \(X\) (say, Euclidean distance), is within a desired tolerance.

Jacobi Method

- What we have just described is the Jacobi method for solving simultaneous equations.
- It is one of several forms of “relaxation”.
- The distinguishing feature of Jacobi is that all the new LHS’s are computed before installing them in place of the Xs.

Gauss-Seidel Method

- In this variation, the new LHS’s are installed as soon as they are computed, assuming that this is done sequentially.
- In its pure form, this method appears unattractive for parallel computing; it is too sequential.
- The following graph shows this.
Gauss-Seidel Method

- If the matrix is sparse and structured in certain regular ways, there is a way to pipeline this computation so that it is not so bad once the pipe is full.
- One such case is the Laplace equation.

Gauss-Seidel v. Jacobi

- On the other hand, Gauss-Seidel converges about twice as fast as Jacobi.
- Intuitively, this is because the effect of changes propagates faster.

Laplace Equation

- Consider the case where the array of unknowns is itself two-dimensional.
- This occurs for example in Laplace’s equation (aka Heat Equation).
- The update rule is that each point depends only on the previous value of its immediate neighbors.

Laplace Equation

- Typically the boundary points are fixed.
- Only the interior points are updated.
- So this is a “boundary value” problem.
- The algebraic problem is derived by discretizing a partial differential equation (Laplace’s equation).
- Let’s look at the Gauss-Seidel update.
Gauss-Seidel Method

- The alternate way of thinking about this is as a “wave-front”.
- If we update a point in Gauss-Seidel as soon as it is “ready” (when its predecessors have been computed), we get the following phenomenon.

Gauss-Seidel Wavefront

- As soon as a sufficiently-many elements of the current sweep are finished, the next sweep can be started in parallel.
- This is pipelining the matrix through a series of sweeps.
- Once the pipeline is “full”, all processors can be kept busy until the end.

Gauss-Seidel Wavefront

- As soon as the wave is one diagonal away, the next wave could be started.
- Since half the points are being updated at any time, just do all half; don’t wait for a wave to arrive.
Red-Black Method

- Degree of parallelism = 1/2 degree for Jacobi (insignificant drawback if large number of points)
- No extra storage required as with Jacobi
- Convergence = Gauss Seidel
- Program similar to iterated matrix-vector, except that red and black iterations alternate

Partitioning

- To partition the previous methods on a distributed-memory system, the number of points should be large compared to the number of processors.
- The grid is divided into per-processor regions.
- Only the borders of these regions need to be communicated to processors of neighboring regions.

Communication across the Border
(“ghost points”)

Embarassingly-Parallel Problems

Pleasantly-Parallel Problems

- The degree of parallelism is very high (grows linearly or better with problem size).
- There is little or no communication between operations or threads.

Graph of a Pleasantly-Parallel Problem
Deja Vu

- The initial segment of cost-optimal versions of problems such as:
  - vector inner-product
  - prefix sum
  - polynomial evaluation
  - Matrix-Vector multiplication, if distribution cost ignored

Examples

- Image Transformation
- IFS (Iterated Function Systems), e.g. “Mandelbrot Set”
- Monte Carlo methods

Image Transformation

- 2-D array (say 512 x 512) of pixels ("pixmap")
- Pixels are grayscale or RGB
- Typical function produces a new pixmap from an old one, by:
  - Coordinate translation, scaling, rotation, clipping, etc. (move pixels to new locations)
  - Masking, thinning (change pixels in place)

IFS (Iterated Function Systems)

- e.g. “Mandelbrot Set”
- The original naïve parallelism problem
- 2-D array of complex values
- Points are updated in place for some large number of iterations, to determine whether each point is “in” the set or not.
- No communication with other points

Mandelbrot Set & Iteration

If $|z| < 2$ after certain large number of iterations, point is declared to be in the set.
Monte Carlo Methods

- Random numbers for fun and profit
- Embarassingly parallel
- Embarassingly slow
- Sometimes the only recourse for complex problems

To Integrate

- Given random \((x, y)\) it is trivial to determine whether \(f(x) \leq y\).
- Simply counting the number of such points and dividing by the total number of points approximates the integral.

Random Number Considerations

- Ideally (from mathematical point of view) one central RNG (random-number generator)
- Central RNG would be bottleneck
- For parallelism, use separate independent RNGs.
- Problem is to keep these from getting synchronized, producing non-random results overall.

Typical RNG Iteration

Parallel version, \(k\) processors, \(A = a^k\) and \(C = c(a^{k-1} + \ldots + a^0)\):

\[
\begin{align*}
x_{i+1} &= (ax_i + c) \mod m \\
x_{i+1} &= (ax_i + c) \mod m \\
x_{i+1} &= (ax_i + c) \mod m
\end{align*}
\]

Processor/Work Pool Model

- aka Worker Model, or Processor Farm
- Can convert many relatively fine-grain problems to be somewhat pleasingly parallel
- Identify units of work, of which there should be a large number
- A unit of work is represented by a struct, such as one containing the parameters to a procedure
Typically one unit of work will spawn others.

- If a processor is available in the pool, the new work will be taken up by it.
- If no processor is available, the new work is put in the work pool until a processor completes some work.

At steady state, we have that at all times either:

- The work pool is empty, or
- The processor pool is empty

Having both work and processors is an unstable condition: the processor should take on the work.

This condition should only exist at start-up or transients.

Work and processors are like matter and anti-matter; they want to annihilate each other.
More Deja Vu

- This model used to schedule threads and processes.
- The work units can be other.
- The “processors” can be processes or threads themselves.

Processor/Work Pool Model

**Advantages**
- Low sensitivity to number of processors and work units
- Finer-grain work units simplify balancing of load
- Good way to hide latency
- Possibly a good basis for fault tolerance

**Disadvantages**
- Some overhead in managing work units
- May create a bottleneck if pools centralized