Fast Fourier Transform (FFT)

Computing DFT requires to compute N terms in the Fourier series (N multiplications), and sum them up, *for each of the numbers* ρ_k. This means the computation goes as $O(N^2)$. This is a lot - today cosmological simulations for example consider up to 10^{11} grid points on large volumes in order to do "precision cosmology" (interpret latest data from eg Cosmic Microwave Background Experiments)--> with direct DFT generating initial conditions would be impossible!

The Fast Fourier Transform (FFT), invented in the 60s, is a method to reduce the order of the calculation by splitting the individual sums recursively so that one has to compute a number of terms much lower then *N* for each of the *N* numbers. The calculation, if N can be expressed as power in base 2, reduces to $O(N\log_2 N)$

Base example: replace a N-point DFT with the sum of many 2 point DFTs

Let us begin by splitting the single summation over N samples into 2 summations, each with $\frac{N}{2}$ samples, one for k even and the other for k odd.
Substitute $m = \frac{k}{2}$ for k even and $m = \frac{k-1}{2}$ for k odd and write:

$$F[n] = \sum_{m=0}^{\frac{N}{2}-1} f[2m] W_N^{2mn} + \sum_{m=0}^{\frac{N}{2}-1} f[2m+1] W_N^{(2m+1)n}$$

Note that $W_N^{2mn} = e^{-j\frac{2\pi}{N}(2mn)} = e^{-j\frac{2\pi}{\frac{N}{2}}mn} = W_{\frac{N}{2}}^{mn}$

Therefore $F[n] = \sum_{m=0}^{\frac{N}{2}-1} f[2m] W_{\frac{N}{2}}^{mn} + W_N^{n} \sum_{m=0}^{\frac{N}{2}-1} f[2m+1] W_{\frac{N}{2}}^{mn}$

ie. $F[n] = G[n] + W_N^{n} H[n]$

Now for N=8 one has to compute:

- even input data $f[0]f[2]f[4]f[6]$
- odd input data $f[1]f[3]f[5]f[7]$

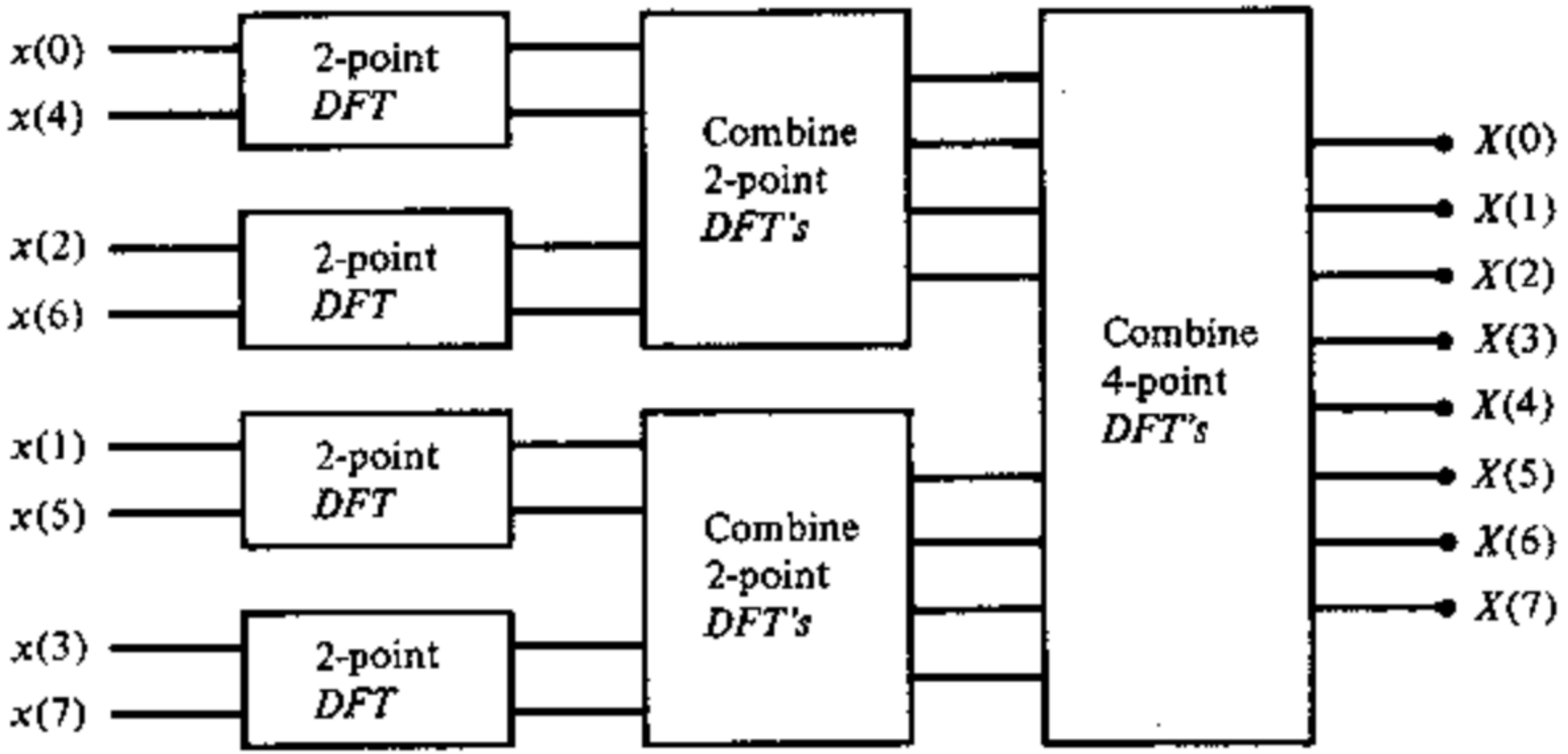
$$
\begin{aligned}
F[0] &= G[0] + W_8^0 H[0] \\
F[1] &= G[1] + W_8^1 H[1] \\
F[2] &= G[2] + W_8^2 H[2] \\
F[3] &= G[3] + W_8^3 H[3] \\
F[4] &= G[0] + W_8^4 H[0] = G[0] - W_8^0 H[0] \\
F[5] &= G[1] + W_8^5 H[1] = G[1] - W_8^1 H[1] \\
F[6] &= G[2] + W_8^6 H[2] = G[2] - W_8^2 H[2] \\
F[7] &= G[3] + W_8^7 H[3] = G[3] - W_8^3 H[3]
\end{aligned}
$$

Overall only 4 terms to be computed!

Assuming than N is a power of 2, we can repeat the above process on the two $\frac{N}{2}$-point transforms, breaking them down to $\frac{N}{4}$-point transforms, etc. . . , until we come down to 2-point transforms. For $N = 8$, only one further stage is needed (i.e. there are γ stages, where $N = 2^\gamma$)

Thus the FFT is computed by dividing up, or *decimating*, the sample sequence $f[k]$ into sub-sequences until only 2-point DFT's remain. Since it is the input, or time, samples which are divided up, this algorithm is known as the *decimation-in-time* (DIT) algorithm. (An equivalent algorithm exists for which the output, or frequency, points are sub-divided – the decimation-in-frequency algorithm.)

Computational scheme for FFT of N=8 sequence with decimation to 2-point DFTs
Three stages involved (log N stages)



The DFT requires N^2 complex multiplications. At each stage of the FFT (i.e. each halving) $\frac{N}{2}$ complex multiplications are required to combine the results of the previous stage. Since there are $(log_2 N)$ stages, the number of complex multiplications required to evaluate an N-point DFT with the FFT is approximately $N/2log_2 N$ (approximately because multiplications by factors such as W_N^0, $W_N^{\frac{N}{2}}$, $W_N^{\frac{N}{4}}$ and $W_N^{\frac{3N}{4}}$ are really just complex additions and subtractions).

N	N^2 (DFT)	$\frac{N}{2}log_2 N$ (FFT)	saving
32	1,024	80	92%
256	65,536	1,024	98%
1,024	1,048,576	5,120	99.5%

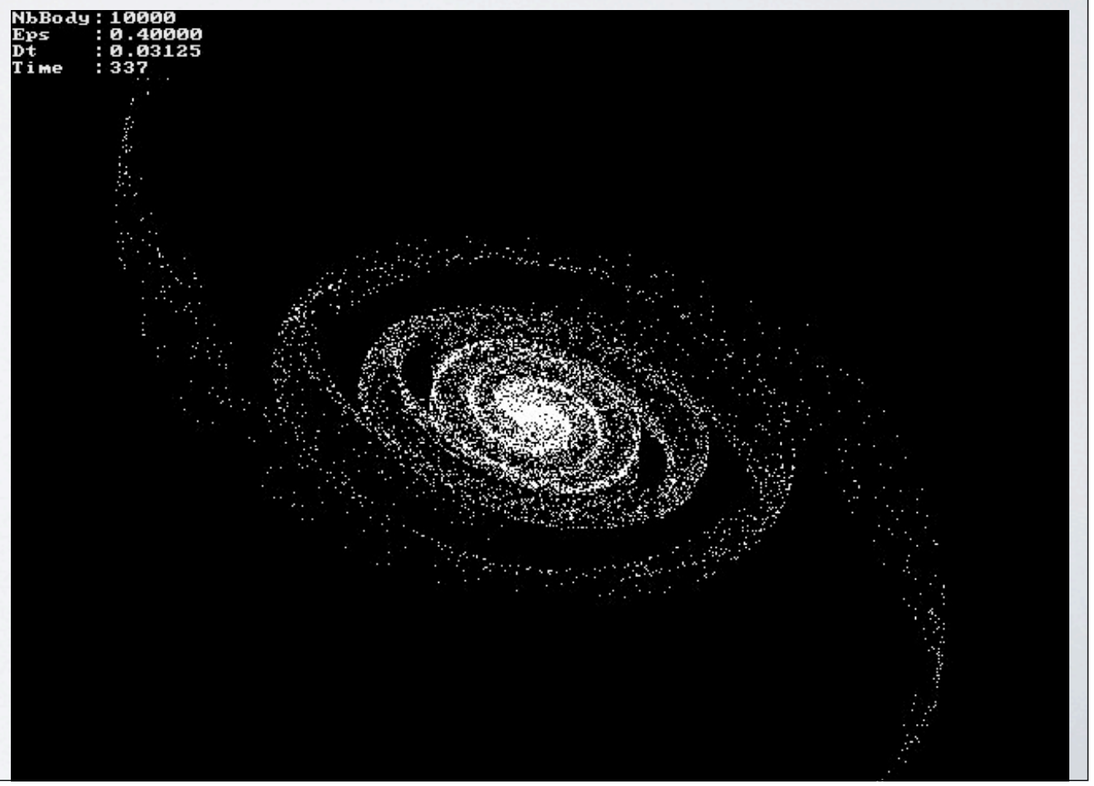
If the number of sampling points for the DFT is not a power 2 one can use alternative symmetries, eg for N=48 one can reduce to 3-point DFTs (16 x 3),or can add zeroes to adjust the number of terms to what required by power of 2.

But in general (eg cosmology) one decides to sample the physical function (eg density) with 2^N mesh points for conveniency

FFT method for non-periodic problems

As we saw the "cosmological simulation" problem is naturally periodic because of the wave-like nature of density perturbations assumed in standard theory. However most other problems in astrophysics are not naturally periodic as they often deal with the model of an "isolated density distribution" around which one can assume *void/vacuum boundary conditions* (i.e. $\rho_b=0$, $\Phi_b=0$)

Staying in the real of collisionless gravitational systems, obvious examples are a numerical model of a galaxy or star cluster.
Can we use FFT in this case?
Or better, can we re-formulate the problem as a fictitious periodic problem?

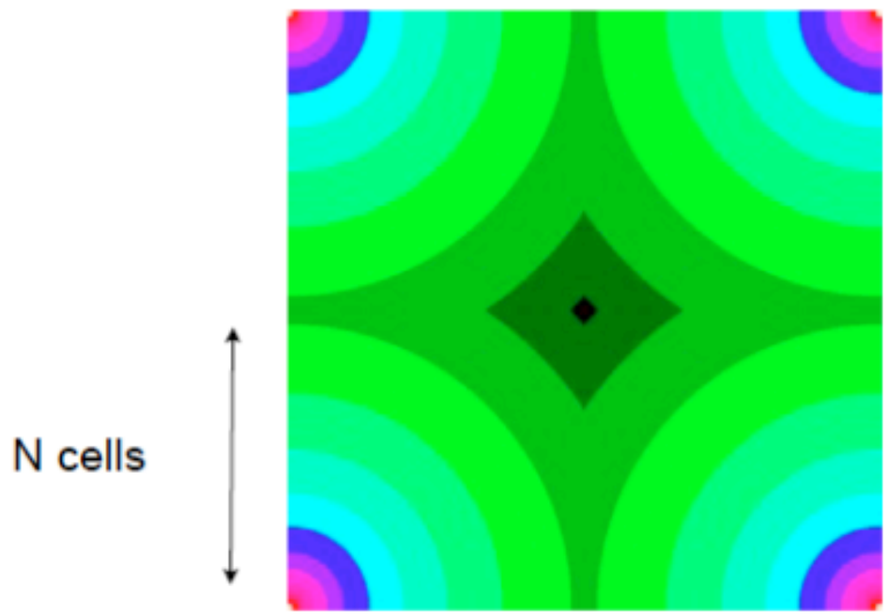


What we need to do is to obtain a periodic potential function despite the non-periodicity of the density exploiting the fact that the potential is convolution of density and Green's function. Using a 2D example may start by defining a periodic Green's function within a finite domain (2N x 2N grid) that is larger than the computational domain (N x N grid).

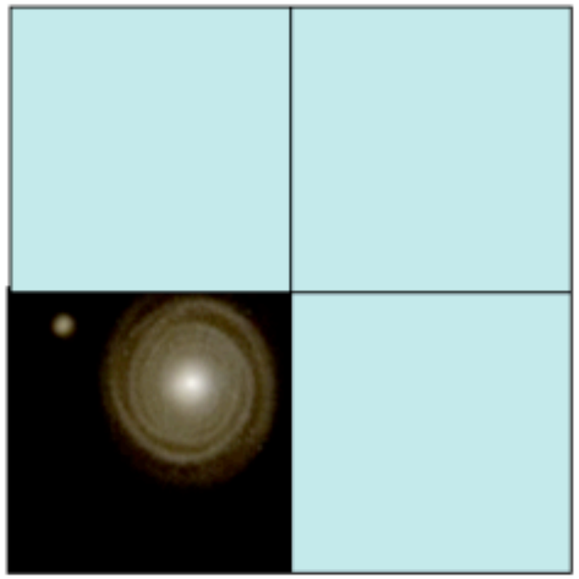
The "zero padding" trick (Hockney & Eatwood 81) for a NxN grid.

1- Compute the Green's function in real space, in the NxN grid.

2- Duplicate and mirror the Green's function in the 3 "ghost" quadrants, so that the Green's function is now periodic in the 2Nx2N grid.



3- Set the density field to zero in the 3 ghost quadrants.



4- Perform the convolution using Fourier analysis on the 2Nx2N grid.

Very simple and flexible ! It is quite expensive (8 times longer) !

Despite its simplicity it clearly can become very unnecessarily expensive for large N. Indeed in the numerical galaxy dynamics community other methods have been preferred since quite some time. For example, a method that can be comparably as fast as FFT but requires no artificial increase in the number of grid points would be preferrable, because in this case if one increases the number of grid points it allows to reach higher resolution rather than "wasting" parts of the computational domain.

Iterative (relaxation) methods and multigrid

Central idea is to turn the Poisson equation, which is a non-linear PDE, into a set of linear equations that can be solved on the mesh using methods from matrix calculus. Eg in 1D:

$$\frac{\partial^2 \Phi}{\partial x^2} = 4\pi G\rho(x)$$

can be approximated into:

$$\left(\frac{\partial^2 \Phi}{\partial x^2}\right)_i \simeq \frac{\Phi_{i+1} - 2\Phi_i + \Phi_{i-1}}{h^2},$$

where the values of Φ are approximtated by finite differencing at nearby mesh points (*h* being the mesh size)

So for each of the N points of a mesh one has to solve a linear equation of the type:

$$\frac{\Phi_{i+1} - 2\Phi_i + \Phi_{i-1}}{h^2} = 4\pi G\rho_i.$$

The problem is thus now that of solving a system of *N* linear equations for *N* unknowns (the *N* values of at the mesh points), which is in principle feasible. We thus solve the matrix equation;

$$\mathbf{Ax} = \mathbf{b}$$

WITH: $\mathbf{x} = (\Phi_i)$, $\mathbf{b} = 4\pi G \rho h^2$

$$\mathbf{A} = \begin{pmatrix} -2 & 1 & & & 1 \\ 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & \cdots & & \\ & & & 1 & -2 & 1 \\ 1 & & & & 1 & -2 \end{pmatrix}$$

In 1D

But using matrix decomposition methods this becomes an $O(N^3)$ matrix inversion problem, unfeasible!

The way forward: iterative approaches

Idea:speed up time-to-solution of linear matrix equation by (i) splitting the **A** matrix into smaller matrices and (ii) invert the result to get the solution x^n (-->Φ^n) using repeated iterations after initial "guess" until convergence is reached.
CPU time cost depends on how many iteration steps are needed for each value of the potential at the mesh point (3 x N values for an N x N x N mesh in 3D)

Different iterative methods can achieve $O(N^2)$ and even O(NlogN).
One stops at truncation/round-off error for "exact" convergence
The fastest ones use the multi-grid approach, whose basic idea is to start iteration from solution obtained on a coarser mesh, map them to the finer grid to iterate there, and repeat the "cycle" several times (with different types of "cycle" between coarse and fine grid possible). Multi-grid iterative Poisson solvers are natural choice for Adaptive Mesh Refinement (AMR) techniques for fluid dynamics

Mathematical Steps: Jacobi Method

$$\mathbf{A} = \mathbf{D} - (\mathbf{L} + \mathbf{U}).$$

$$[\mathbf{D} - (\mathbf{L} + \mathbf{U})]\mathbf{x} = \mathbf{b},$$

Decompose **A** into diagonal **(D),** lower diagonal **(L)** and upper diagonal **(U)**

$$\mathbf{x} = \mathbf{D}^{-1}\mathbf{b} + \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}. \longrightarrow \mathbf{x}^{(n+1)} = \mathbf{D}^{-1}\mathbf{b} + \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}^{(n)}.$$

Convergence requires spectral radius of matrix **M** less than unity

$$\rho_s(\mathbf{M}) \equiv \max_i |\lambda_i| < 1.$$

eigenvalues

$$\mathbf{M} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})$$

This follows from $|\mathbf{e}^{(n)}| \leq [\rho_s(\mathbf{M})]^n |\mathbf{e}^{(0)}|$ where error vector is:

$$\mathbf{e}^{(n+1)} = \mathbf{x}_{\text{exact}} - \mathbf{x}^{(n+1)} = \mathbf{x}_{\text{exact}} - \mathbf{D}^{-1}\mathbf{b} - \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}^{(n)} = \mathbf{M}\mathbf{x}_{\text{exact}} - \mathbf{M}\mathbf{x}^{(n)} = \mathbf{M}\mathbf{e}^{(n)}$$

For specific case of Poisson equation:

In 2D: $$\phi_{i,j}^{n+1} = \frac{1}{4}\left(\phi_{i+1,j}^{n} + \phi_{i-1,j}^{n} + \phi_{i,j+1}^{n} + \phi_{i,j-1}^{n}\right) - \frac{1}{4}\rho_{i,j}\, 4\pi g h^2$$

Note in Jacobi method for a certain iteration step *n* is completed for all cells of the mesh (all mesh points) and *then* the next iteration step is performed.

One can imagine using immediately the new values to get one level up in the iteration for a mesh point, i.e. before the particular iteration step is completed for all cells/mesh points. This to speed up the overall calculation for *N* cells/mesh points.
This idea leads to the Gauss-Seidel iteration method.

$$(\mathbf{D}-\mathbf{L})\mathbf{x}=\mathbf{U}\mathbf{x}+\mathbf{b}, \longrightarrow \mathbf{x}=(\mathbf{D}-\mathbf{L})^{-1}\mathbf{U}\mathbf{x}+(\mathbf{D}-\mathbf{L})^{-1}\mathbf{b},$$

w/a few algebraic steps

$$\mathbf{x}^{(n+1)}=\mathbf{D}^{-1}\mathbf{U}\mathbf{x}^{(n)}+\mathbf{D}^{-1}\mathbf{L}\mathbf{x}^{(n+1)}+\mathbf{D}^{-1}\mathbf{b}.$$

Note in the iterative equation if we compute elements in the first row $i=1$ no values x^{n+1} are needed to start the iteration because by definition L contains only elements below the diagonal. So we can readily compute x^{n+1} if we have x^n and then use the results to move on with the iteration for other rows (the other rows also will need x^{n+1} only for some elements, i.e those above the diagonal, so the same trick can be exploited several times).

Practical issue: with this way of proceeding one needs a specific sequential order on the mesh to solve the Poisson equation, which is a problem if we want to parallelize the computation (domain decomposition).

Solution: create efficient update rules by reordering matrix elements into disjoint sets within which potential is updated only iterating between cells of other sets. Each processor computes solution at one of the mesh point in a set, then same for the other set etc..

Efficient update rule: Red-black ordering

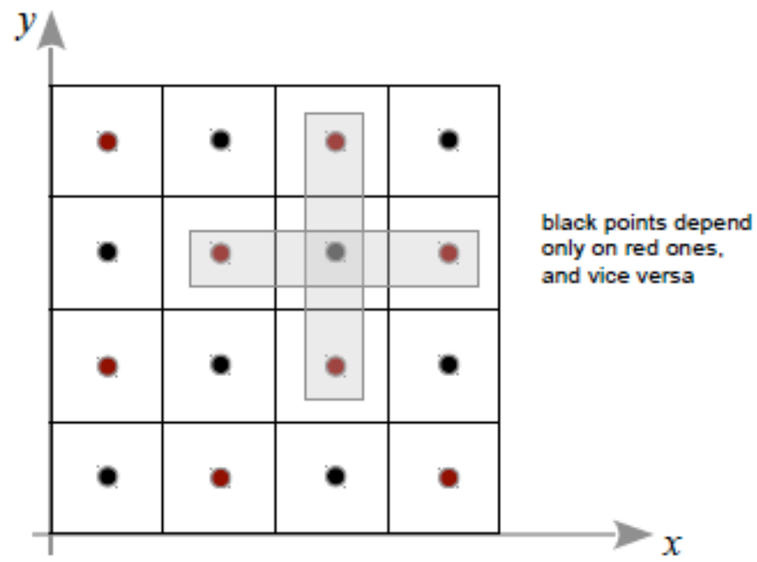


Fig. 11 Red-black ordering in which two interleaved chessboard-like patterns are formed that can be independently processed with immediate updating.

Rule: update first all black points, then red points. In second update one uses values computed for first update. The second update will then be much faster, indeed almost twice as fast to convergence relative to Jacobi. Easily parallelizable as it uses all domain at all steps.

Parallellization; each processor computes potential at one of the black points in first step because evaluations are independent and should be done simultaneously (natural synchoniszation). Same for red points later.

The multigrid technique

With standard iteration methods several issues:

- convergence slow, at best *O(N)*, so computation for N points goes as $O(N^2)$

- convergence depends on "wavelength" (slower for long wavelengths on the mesh, namely for contribution to the potential from most distant mesh points)

- at every iteration only nearby cells communicate (eg in red black ordering scheme), but fast convergence requires information on updates communicated efficiently through the whole domain

Idea: go to a coarser mesh to achieve faster convergence and produce guess value for iteration on finer mesh
(eg if coarser mesh has 2x less resolution convergence speed ups by ~ 1/8 order of magnitude since there would be ~ 1/8 mesh points at which to compute potential)