A Comparison of a Second-Order versus a Fourth-Order Laplacian Operator in the Multigrid Algorithm

Kaushik Datta (kdatta@cs.berkeley.edu)
Math 221 Project
May 19, 2003

Abstract

In this paper, the multigrid algorithm was used to solve Poisson's equation for various right-hand sides. However, in order to calculate the discretized laplacian value, both a second-order and a fourth-order laplacian operator was used. The fourth-order laplacian operator achieves higher accuracy at the cost of more calculations and a longer execution time. In order to compensate for this, the grid spacing for the fourth-order operator was doubled, and the convergence results with the second-order operator were compared for similar program running times.

1. Preliminaries

1.1 The Multigrid Algorithm

Multigrid is an iterative algorithm that is generally used to solve elliptic partial differential equations. In our case, it will be used to solve Poisson's equation. Specifically, we seek the solution φ in the equation:

$$\Delta\varphi = \frac{\partial^2\varphi}{\partial x^2} + \frac{\partial^2\varphi}{\partial y^2} + \frac{\partial^2\varphi}{\partial z^2} = \rho$$

Convergence to a fixed error in multigrid takes O(log N) steps, where N is the number of variables in grid φ. In addition, the algorithm operates in O(N) time since it can communicate data across many grid cells in one step.[1]

The three main operators in the multigrid algorithm are restrict (coarsen), interpolate, and relax. The restrict operation coarsens the grid to one with half the dimensions of the original grid. In the three-dimensional case, each value in the smaller grid is a weighted average of the eight values in the original grid that correspond to that point. Conversely, interpolation takes a coarse grid and distributes a weighted average of each value onto the eight corresponding points in the fine grid. Thus, restriction and interpolation are complementary operations. Finally, the relax operation is some iterative method that is

effective in smoothing the current solution, usually by subtracting off a fraction of the residual. Note that all three operators only do nearest-neighbor computations by averaging neighboring grid values. However, since adjacent points in the coarsened grids represent points that are far away in the fine grids, information still moves quickly across the grid.

Three-dimensional multigrid, in its simplest implementation, usually requires an equal, power-of-two number of cells in each dimension of the grid. The base two logarithm of the number of cells in one dimension can be considered the *level* of that grid. For instance, a 64 x 64 x 64 grid is a level 6 grid. The coarsen operation lowers the grid level by one, while interpolate increases it by one. The level of a grid is useful in understanding which phase of multigrid is being executed.

The multigrid algorithm begins by calculating the residual associated with the current solution. This residual is then iteratively refined using any relaxation operator, including Jacobi, Gauss-Seidel, and Gauss-Seidel red-black. In our case, three iterations of a modified weighted Jacobi method were used. To be precise, for the starting grid level *l*, each grid point in φ executed the following:

$$\varphi^{l}_{(i,j,k)} = \varphi^{l}_{(i,j,k)} + \lambda(\Delta\varphi^{l}_{(i,j,k)} - \rho^{l}_{(i,j,k)})$$

The calculation of the laplacian operator will be discussed later. The quantity $(\Delta\varphi^{l}_{(i,j,k)} - \rho^{l}_{(i,j,k)})$ is the negative of the residual, so a fraction of the residual is being subtracted from the solution during each iteration. The value of λ was chosen by using the rule-of-thumb formula $\lambda = h^2/12$, where *h* is the grid spacing. After relaxing the solution, a new residual grid is computed for this level, and this grid is then coarsened through the restrict operator. Mathematically, for each point in the coarsened residual grid *R* at level *l*, the following is performed:

$$R^{l}_{(i,j,k)} = \frac{1}{8}(R^{l+1}_{(2i,2j,2k)} + R^{l+1}_{(2i+1,2j,2k)} + R^{l+1}_{(2i,2j+1,2k)} + R^{l+1}_{(2i,2j,2k+1)} + R^{l+1}_{(2i+1,2j+1,2k)} + R^{l+1}_{(2i,2j+1,2k+1)} + R^{l+1}_{(2i+1,2j,2k+1)} + R^{l+1}_{(2i+1,2j+1,2k+1)})$$

The coarsened residual grid is recursively relaxed and coarsened is this manner until a level 0 grid is achieved. At that point, the grid is again relaxed three times. Then, the correction from this level 0 grid is interpolated onto the level 1 correction grid, which is initially all zeros. In general, for each point in the coarsened correction grid δ at level *l*, interpolate does the following:

$\varphi^{l+1}_{(2i,2j,2k)} = \varphi^{l+1}_{(2i,2j,2k)} + \delta^{l}_{(i,j,k)}$	$\varphi^{l+1}_{(2i+1,2j+1,2k)} = \varphi^{l+1}_{(2i+1,2j+1,2k)} + \delta^{l}_{(i,j,k)}$
$\varphi^{l+1}_{(2i+1,2j,2k)} = \varphi^{l+1}_{(2i+1,2j,2k)} + \delta^{l}_{(i,j,k)}$	$\varphi^{l+1}_{(2i+1,2j,2k+1)} = \varphi^{l+1}_{(2i+1,2j,2k+1)} + \delta^{l}_{(i,j,k)}$
$\varphi^{l+1}_{(2i,2j+1,2k)} = \varphi^{l+1}_{(2i,2j+1,2k)} + \delta^{l}_{(i,j,k)}$	$\varphi^{l+1}_{(2i,2j+1,2k+1)} = \varphi^{l+1}_{(2i,2j+1,2k+1)} + \delta^{l}_{(i,j,k)}$
$\varphi^{l+1}_{(2i,2j,2k+1)} = \varphi^{l+1}_{(2i,2j,2k+1)} + \delta^{l}_{(i,j,k)}$	$\varphi^{l+1}_{(2i+1,2j+1,2k+1)} = \varphi^{l+1}_{(2i+1,2j+1,2k+1)} + \delta^{l}_{(i,j,k)}$

So, the correction from the grid at each level is interpolated onto the correction grid of the next higher level (initially all zeros). The higher level grid is then relaxed by again performing three Jacobi iterations. This is done for consecutively higher grid levels until the correction is interpolated onto the original solution grid. After three more Jacobi iterations, one multigrid V-cycle is completed.

The key to multigrid is having the relax operation effectively suppresses the high frequency (oscillatory) Fourier modes of the residual *for that level*. By coarsening, the number of cells in each dimension is halved, so low frequency error becomes higher frequency error. Performing relax operations should again suppress the higher Fourier modes in the error. If this is executed over all grid levels, then multigrid should reduce error over all frequencies.[2]

1.2 The Laplacian Operators

In the multigrid algorithm, the laplacian value needs to be computed both in calculating the residual and in performing relaxes. There are two different laplacian operators that were used in doing so. Let us adopt the convention that for the 3 x 3 x 3 cube surrounding a given point in φ, then φ_m, φ_f, φ_e, φ_c represent a middle, face-centered, edge-centered, and corner point respectively. A similar convention holds for grid points in ρ. Then, the laplacian for the first formula is calculated by:

$$\Delta\varphi_{(i,j,k)} = \frac{1}{h^2}\left(-6\varphi_m + \Sigma\varphi_f\right)$$

This is a second-order formula. The second formula is of fourth-order, and calculates the laplacian as follows:

$$\Delta\varphi_{(i,j,k)} = \frac{1}{3h^2}\left(\Sigma\varphi_e + 2\Sigma\varphi_f - 24\varphi_m\right) - \frac{1}{6}\Sigma\rho_f$$

The fourth-order formula should allow multigrid to converge more rapidly than the second-order formula.[3] However, it will also take longer to execute a V-cycle since more calculations need to be performed. As a result, we will compare whether using twice the grid spacing for the fourth-order formula will result in better convergence results than the second-order formula for a given program running time.

1.3 The Programming Language and Platform

The multigrid code was written in the Titanium programming language. Titanium is an object-oriented high performance Java dialect written at UC Berkeley. To improve performance, Titanium code is first compiled to C, and then to native binary. While Titanium can be used for parallel processing, all the results in this paper are serial.[4]

This code was run on the distributed-memory IBM SP machine *seaborg.nersc.gov*. Each node of seaborg contains 16 IBM Power3 processors, each of which has a peak flop rate of 1.5 GFlops. In addition, each processor has two levels of cache. The L1 cache has a data cache size of 64 KB, with a cache line size of 128 B and a latency of 1 cycle. The L2 cache has a size of 8 MB with a cache line size of 128 B and a latency of 9 cycles.[5]

2. The Test Functions

There were four test functions used for the right-hand side that were solved using multigrid to compare the laplacian operators. In order to simplify matters, periodic boundary conditions were implemented in all three dimensions, so the chosen functions needed to conform to this constraint.

In addition, having functions with different Fourier modes would be useful in order to observe how well multigrid suppresses low, medium, and high frequency error. However, we need to make sure that the high frequency modes do not result in *aliasing*. Aliasing is a phenomenon where the wavelength of a Fourier mode is less than 2*h*. The resulting wave appears to have a wavelength greater than 2*h* on the grid since there are too few grid points to represent the wave accurately.

With these considerations in mind, the following four functions were chosen:

Freq.	φ (exact)	ρ (Test Functions)	Domain
Low	$\cos(2\pi x)\cos(2\pi y)\cos(2\pi z)$	$-12\pi^2\cos(2\pi x)\cos(2\pi y)\cos(2\pi z)$	(-.5,-.5,-.5) to (.5, .5, .5)
Med.	$\cos(8\pi x)\cos(8\pi y)\cos(8\pi z)$	$-192\pi^2\cos(8\pi x)\cos(8\pi y)\cos(8\pi z)$	(-.5,-.5,-.5) to (.5, .5, .5)
High	$\cos(26\pi x)\cos(26\pi y)\cos(26\pi z)$	$-2028\pi^2\cos(26\pi x)\cos(26\pi y)\cos(26\pi z)$	(-.5,-.5,-.5) to (.5, .5, .5)
All	$\frac{1}{4}(\cos(2\pi x)\cos(2\pi y)\cos(2\pi z)) +$ $\frac{1}{64}(\cos(8\pi x)\cos(8\pi y)\cos(8\pi z)) +$ $\frac{1}{676}(\cos(26\pi x)\cos(26\pi y)\cos(26\pi z))$	$-3\pi^2(\cos(2\pi x)\cos(2\pi y)\cos(2\pi z) +$ $\cos(8\pi x)\cos(8\pi y)\cos(8\pi z) +$ $\cos(26\pi x)\cos(26\pi y)\cos(26\pi z))$	(-.5,-.5,-.5) to (.5, .5, .5)

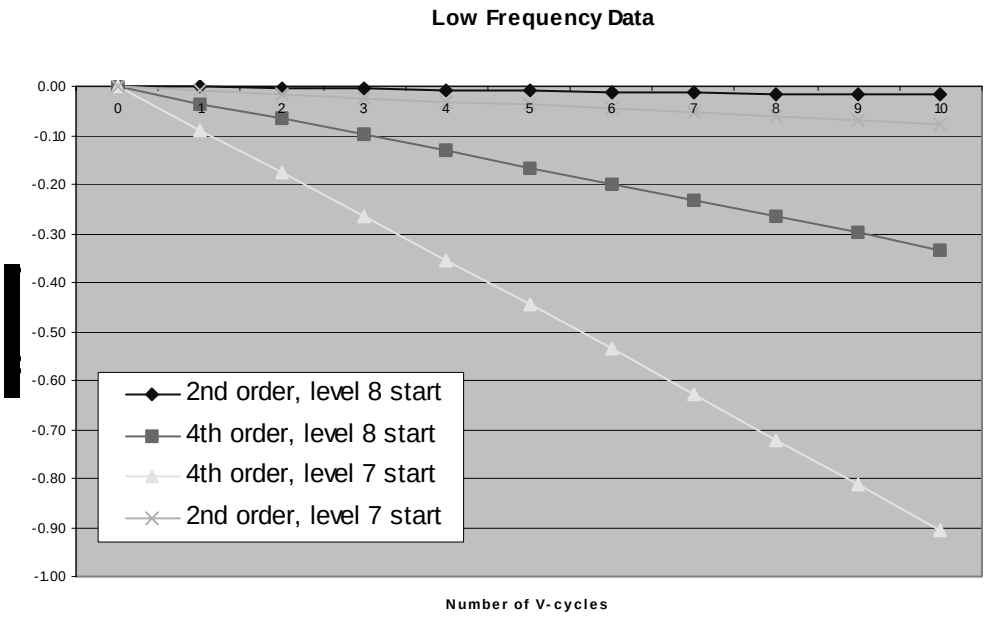
Having the exact solution φ allows us to calculate the error directly rather than use the residual.

3. Convergence Results

The following results plot the base 10 log of the maximum error against the number of multigrid V-cycles executed (execution time is not yet included). Examining the maximum error gives us a good sense of the worst-case convergence rate of multigrid. The four runs that are shown are for the second and fourth-order laplacian stencil being used for multigrid that begins on a level 7 grid and level 8 grid. Note that starting at level 7 results in twice the grid spacing as that of level 8.

3.1 Low Frequency Data

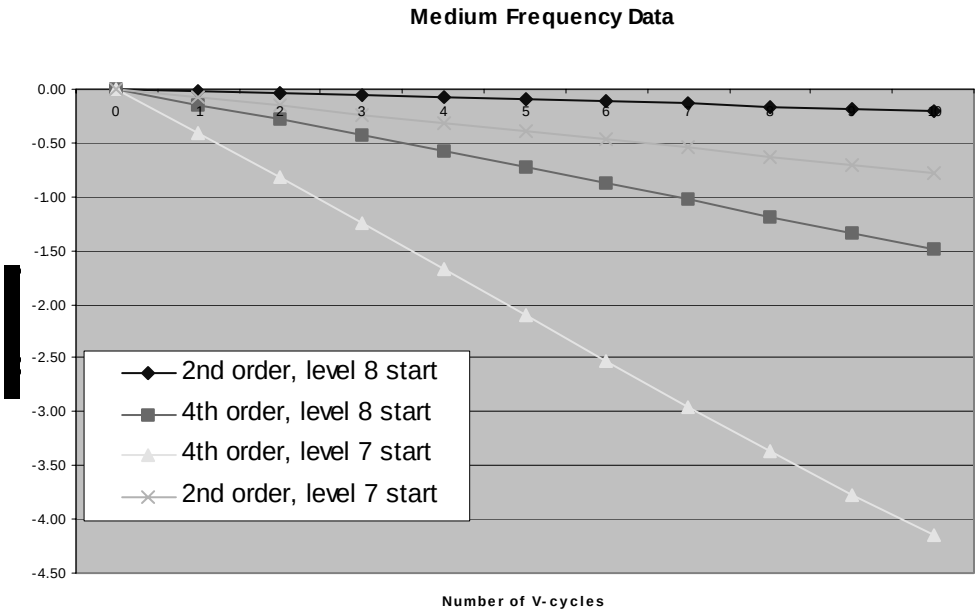
Here is the graph for low frequency data:



While all the runs seem to converge at a constant rate, the 4^{th} order method that starts on a level 7 grid definitively outperforms the other runs.

3.2 Medium Frequency Data

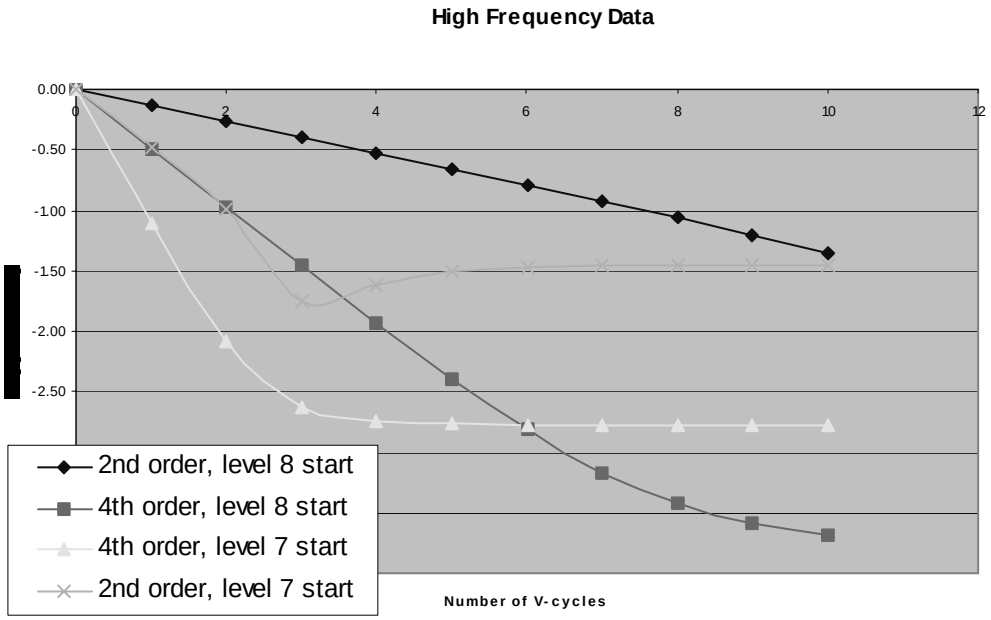
Here is the graph for medium frequency data:



While the rates of convergence are faster than for the low frequency data, the fourth-order laplacian operator that starts on a level 7 grid again outstrips the other runs.

3.3 High Frequency Data

Here is the same graph for high frequency data:

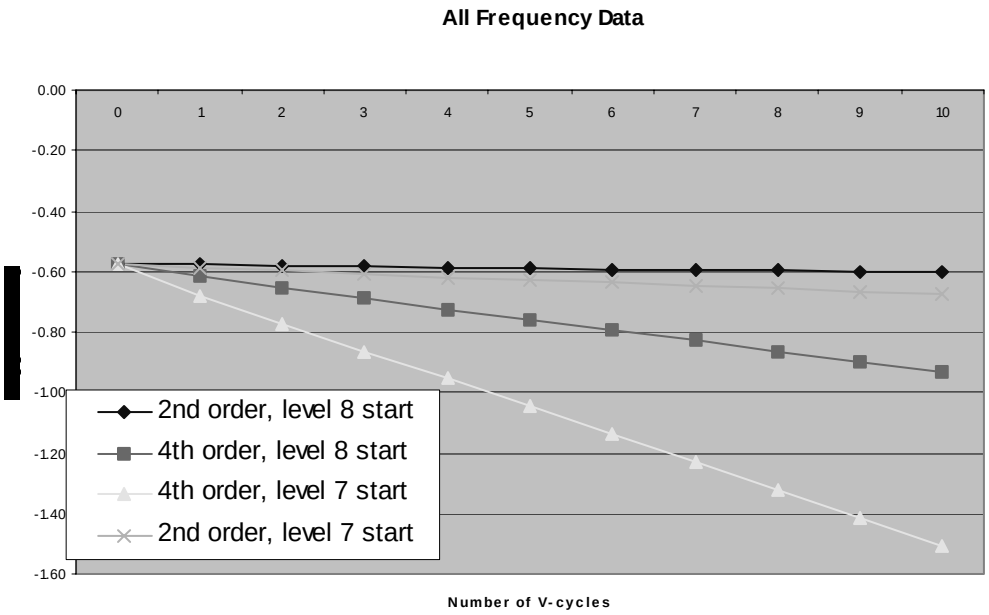


The suppression of the oscillatory Fourier modes, while converging at a relatively rapid constant rate for a number of V-cycles, seems to stop converging at a certain point. This seems to indicate that while high-frequency error is quickly suppressed, it is difficult to

completely eliminate. Only the second-order laplacian operator that starts at level 8 continues to converge at a constant rate for at least ten V-cycles, but it is also the slowest converging run.

3.4 All Frequency Data

Here is the graph for all frequency data:



Like the low and medium frequency data graphs, the fourth-order laplacian operator that starts on the level 7 grid seems to do better than the other runs.

4. Analysis

It is unnecessary to factor time into the analysis since the results seem to indicate that, for the same number of V-cycles, starting on a level 7 grid produces faster convergence than a level 8 grid. This is a somewhat surprising result, and there are several possible explanations.

First, the appropriate norm was not chosen for the problem at hand. While this analysis uses the maximum error over the solution grid, it may have been better to choose the L2 norm of the solution grid instead. The L2 norm is a better indication of the error over the entire grid, and is more typically used in this type of analysis. If there were a small area of the grid that produced unusually large error, the maximum error norm would be large, while the L2 norm would not increase significantly. This type of error could possibly account for the results above.

In addition, a good value for λ may not have been used for the level 8 grids. While $\lambda = h^2/12$ is typically a good value, a Fourier error analysis would have to be performed

to make sure that the high-frequency error is being effectively damped at each level of multigrid.

In addition, other test functions should be used. While each function listed isolates a specific Fourier mode, it may be worthwhile to use non-trigonometric functions as the right-hand side, as they may exhibit different types of convergence.

Finally, the condition number of the matrix needs to be accounted for. In this case, starting at a level 7 grid produces faster convergence for both the second and fourth order laplacian operators than starting at a level 8 grid, so that still does not explain our results. However, it still needs to be factored into the analysis.

5. Further Analysis

As stated above, the analysis should be attempted first with the L2 norm, as that may explain why level 7 grids converge more rapidly than level 8 grids. If that produces similar results, then the other options listed above should be tried. Using a larger grid spacing should not result in faster convergence, as this would imply that, for the same number of V-cycles, using fewer mesh points would increase the rate of convergence.

References

[1] Yelick, Kathy. http://www-inst.eecs.berkeley.edu/~cs267, Lecture 23.

[2] Briggs, William L., Van Emden Henson, Steve F. McCormick. *A Multigrid Tutorial, 2nd edition*. SIAM, Philadelphia, PA, 2000.

[3] Collatz, Lothar. *The Numerical Treatment of Differential Equations, 3rd Edition*. Springer-Verlag, Berlin, Germany, 1960, pp. 546.

[4] Titanium programming language. http://www.cs.berkeley.edu/projects/titanium.

[5] IBM SP. http://hpcf.nersc.gov/computers/SP.