Chombo 4 Performance Report

Simon Zhang, D. T. Graves, D. F. Martin

September 2023

Abstract

This technical report presents a performance study of the $EBINS$ example in Chombo 4. Chombo 4 provides a set of tools for implementing finite difference and finite volume methods for the solution of partial differential equations on block-structured adaptively refined rectangular grids. We provide insights into the computational performance of Chombo 4 when utilizing a second-order finite volume algorithm to solve the incompressible Navier-Stokes equations within a cut cell context. In the end, we ran weak scaling tests on $EBINS$ to determine how Chombo 4 scales as we increase its workload and computational resources. Consequently, we are able to determine the optimal parameters in terms of performance and convergence. This technical report is also a proof-of-concept for the need of geometric multigrid by uncovering the relationship between smoothing and geometric multigrid iterations.

1 Introduction

When solving partial differential equations (PDEs), problems with multiple length scales and strong spatial localizations are difficult to solve, analytically and numerically. Chombo provides a set of tools for implementing finite difference and the finite volume methods with block-structured adaptive mesh refinement (AMR) to compute solutions to these difficult PDEs. It includes both elliptic and time-dependent modules while supporting calculations in complex geometries with both embedded boundaries and mapped grids. In Chombo, the problem domain is discretized using a logically-rectangular (often Cartesian) grid and a solution is computed on that grid. Chombo identifies regions on that grid that require finer resolution by computing local indicators of the solution error. Then, Chombo covers those rectangles by a disjoint union of rectangles in the domain and refines this subset by some integer factor. The solution is then computed on the composite grid. This process may be applied recursively for both time and space [1].

In Chombo, geometric multigrid is implemented in this process as a means to compute the solution for elliptic systems of equations. Geometric multigrid is a numerical technique for solving differential equations, especially for when basic iterative solvers or direct solvers exhibit the smoothing property. Examples of such solvers are the Jacobi method and the Gauss-Seidel method. The smoothing property causes these solvers to eliminate the high-frequency or oscillatory components of the error effectively, but failing to do so for the low-frequency or smooth components of the error. To overcome the smoothing property, geometric multigrid coarsens the current grid or reduces the grid resolution. This transformation turns low-frequency components into high-frequency components. Then, it employs iterative solvers to eliminate these high-frequency components on the coarsened grid, repeating this process until the coarsest level is reached.

Furthermore, the algorithm to geometric multigrid can be summarized into the following steps [3]:

Relaxation/Smoothing: Solve using basic iterative solvers until high-frequency components of the error are eliminated

Restriction: Transform the error to a coarser grid and solve/relax/smooth

Coarsest grid is reached and error is computed

Prolongation: Interpolate the error from coarse grid to finer grid and the interpolation is added to the finer grid solution

Relaxation/Smoothing: Solve using basic iterative solvers until high-frequency components of the error are eliminated

This is also known as the v-cycle (Figure 1). In Chombo 4, *mac_proj.numSmooth* is how many times relaxation/smoothing is applied and *mac_proj.useWcycle* is whether w-cycle is used or not, which is a v-cycle with one or more intermediate grid levels between each coarser level and the finer level (Figure 1).

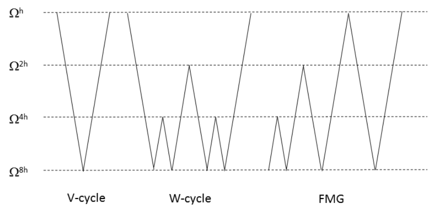


Figure 1: Schedule of grids for V-cycle, W-cycle, and FMG scheme all on four levels [2]

Another important concept to understand geometric multigrid in Chombo 4 is agglomeration: a technique used to create coarser grids from fine grids by grouping multiple fine grids boxes together to define coarser grid boxes. Agglomeration is implemented in Chombo 4 to reduce overall computational cost of multigrid methods, resulting in better performance.

To investigate the performance of Chombo 4, we ran weak scaling tests and manipulated inputs on the *EBINS* example: a second-order finite volume algorithm to solve the incompressible Navier-Stokes equations within a cut cell context.

2 Methodology

2.1 Weak Scaling Test

Weak scaling tests were carried out on NERSC Perlmutter CPU Nodes using the *EBINS* example. We started the tests using 1 processor of 1 CPU node in the domain of $nx = 32$ and maximum grid of 32. Increasing the domain by a factor of 2 and the number of processors by 2^3, we stop at a domain of $nx = 256$ and 516 processors. By doing so, we are ensuring for all of our runs, each processor is being dealt with the same workload of one box. We have also changed the *mac_proj.numSmooth* and *mac_proj.useWCycle* of the input files.

2.2 Optimal Parameters

Using the time tables of the same runs, we are able to investigate the optimal choice of *mac_proj.numSmooth* and *mac_proj.useWCycle* in terms of performance for different number of processors and problem sizes.

2.3 Convergence

Using the output from the same runs, we are able to investigate how the convergence of different problems can be impacted by the different choices of *mac_proj.numSmooth* and *mac_proj.useWCycle*.

3 Findings

3.1 Weak Scaling Test

We find that using *mac_proj.numSmooth* = 2 and *mac_proj.useWcycle* = false scales the best (Figure 2). In general, weak scaling tests with *mac_proj.useWcycle* = false tend to scale better than weak scaling tests with *mac_proj.useWcycle* = true and *mac_proj.numSmooth* = 2 tend to scale better than higher smoothing (Figure 3).

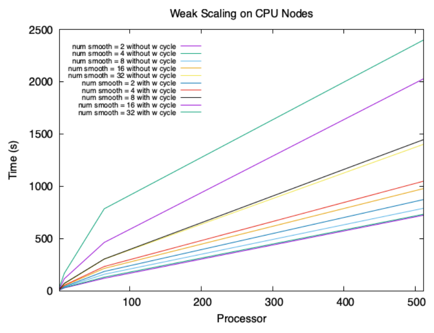
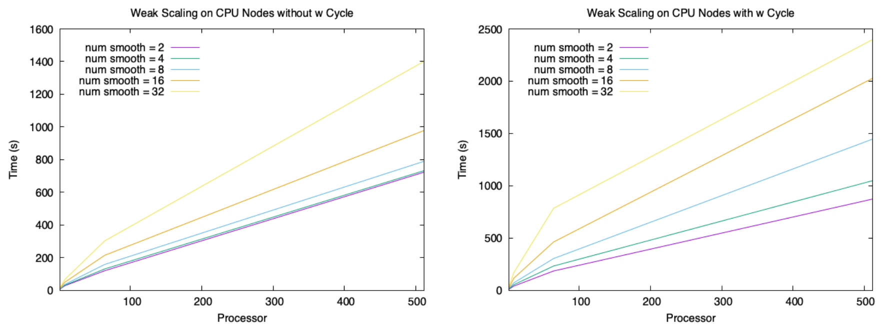


Figure 2: Weak Scaling on CPU Nodes



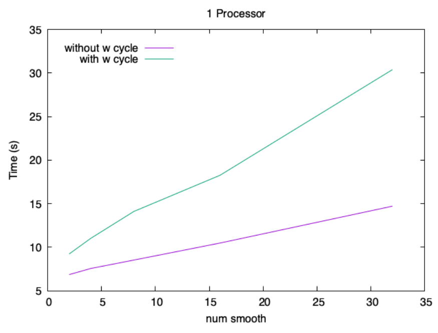
(a) Without w Cycle

(b) With w Cycle

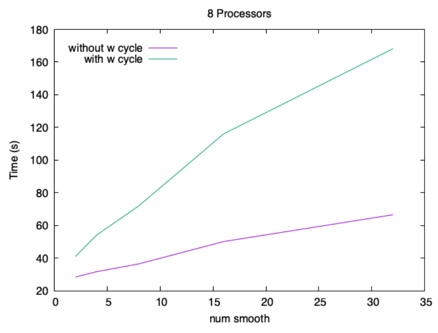
Figure 3: Weak Scaling on CPU Nodes with and without w Cycle

3.2 Optimal Parameters

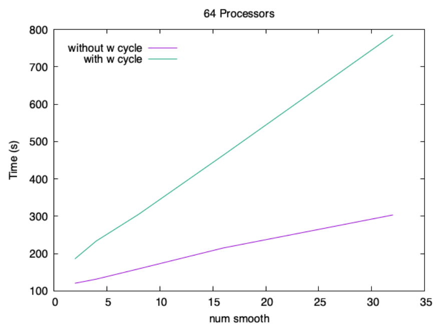
We find that using $mac\_proj.numSmooth = 2$ give us the best performance for both choices of $mac\_proj.useWcycle$ (Figure 4) and runs without w-cycles have quicker runtimes compared to runs with w-cycles.



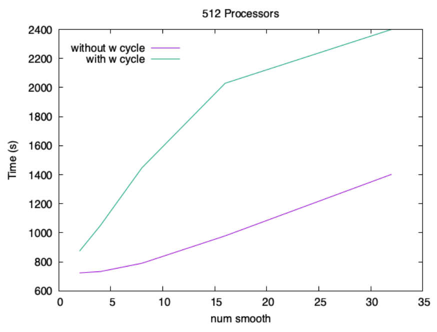
(a) 1 Processor



(b) 8 Processors



(c) 64 Processors



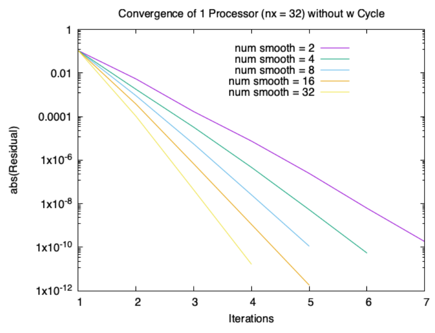
(d) 512 Processors

Figure 4: Performance of 1, 8, 64, and 512 Processors under various $mac\_proj.numSmooth$

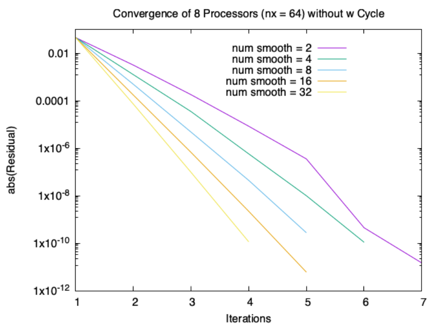
3.3 Convergence

From figure (5) and figure (6), we observe that higher $mac\_proj.numSmooth$ has the quickest rate of convergence. In particular, as we increase the number of smoothing, the number of geometric multigrid iteration decreases. Furthermore, there is an approximate 4:1 ratio between the number of smoothing and the number of geometric multigrid iteration, where increasing the number of smoothing by a factor of 4 results in one less geometric multigrid iteration.

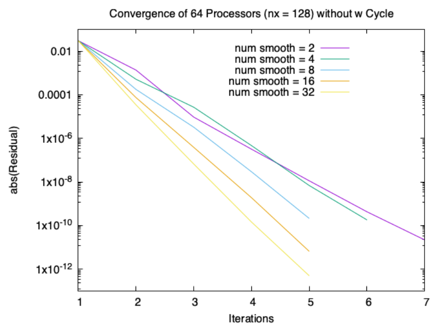
In section 3.2, $mac\_proj.numSmooth = 32$ returns the slowest run times and in section 3.3 $mac\_proj.numSmooth = 32$ returns the quickest rates of convergence with the lowest count of geometric multigrid iterations. On the other hand, $mac\_proj.numSmooth = 2$ returns the quickest run times, slowest rates of convergence and highest count of geometric multigrid iterations.



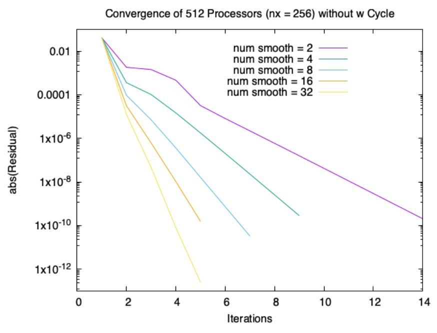
(a) 1 Processor without w cycle



(b) 8 Processors without w cycle

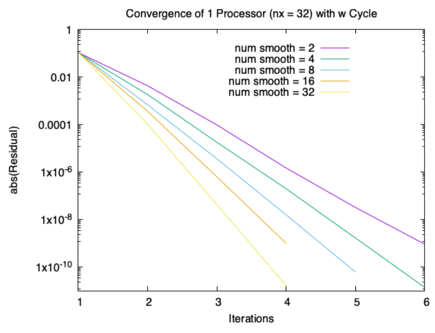


(c) 64 Processors without w cycle

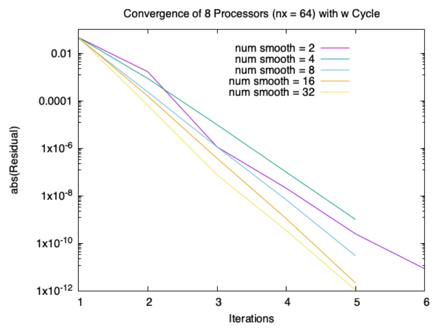


(d) 512 Processors without w cycle

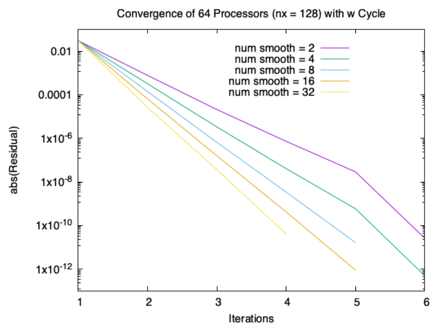
Figure 5: Convergence of 1, 8, 64, and 512 Processors without w cycle



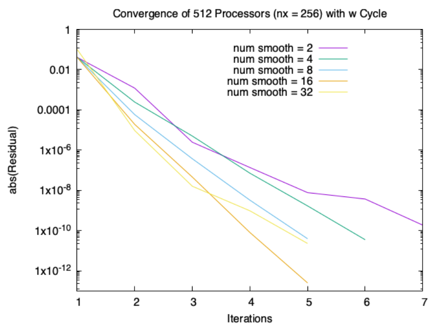
(a) 1 Processor with w cycle



(b) 8 Processors with w cycle



(c) 64 Processors with w cycle



(d) 512 Processors with w cycle

Figure 6: Convergence of 1, 8, 64, and 512 Processors with w cycle

From figure (7), we claim that Chombo 4 converge quicker with w-cycles than without. Compared to v-cycles, w-cycles has more smoothing due to itself having more levels, thus, it makes sense for examples with w-cycles to converge quicker than those without. Meanwhile, in figure (4), runs with w-cycle have longer runtimes, showing that more smoothing is more computationally expensive.

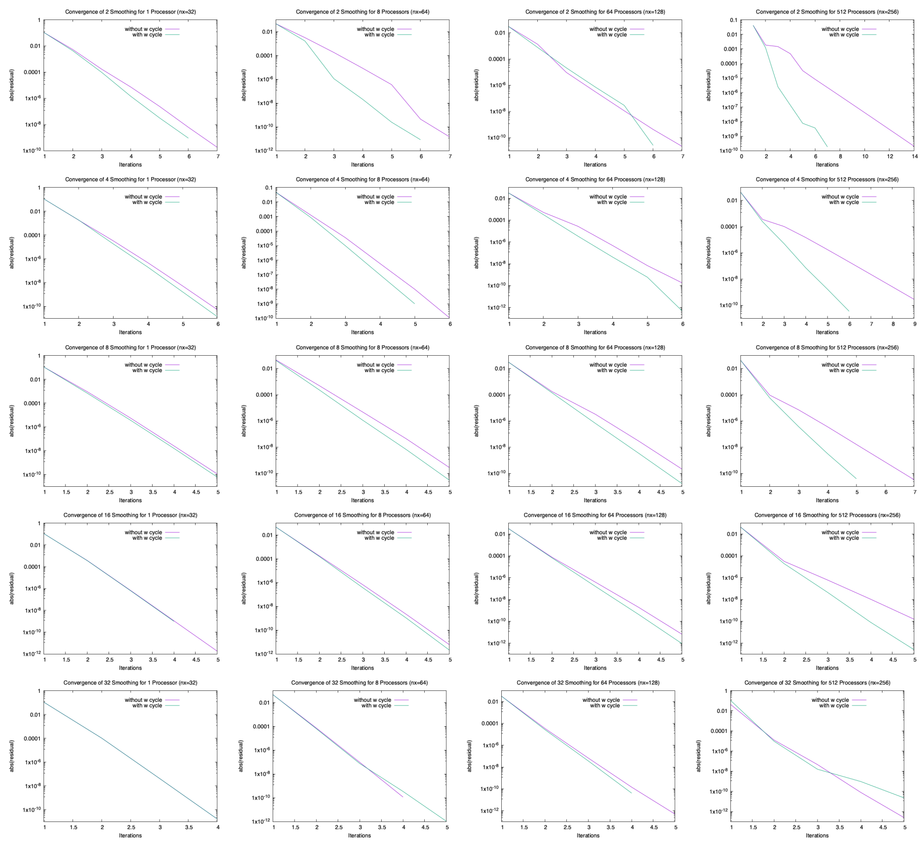


Figure 7: Convergence of $mac\_proj.numSmooth = 2$ (first row), 4 (second row), 8 (third row), 16 (fourth row), 32 (fifth row) for 1 (first column), 8 (second column), 64 (third column), and 512 Processors (fifth column). Green represents runs with w-cycle and purple represents runs without w-cycle

3.4 Discussion

In this report, we conduct weak scaling tests on Chombo 4's $EBINS$ example. These tests are carried out for problems with a maximum grid size of 32 on domains ranging from $nx = 32$ to $nx = 256$ with processors varying from 1 to 512, respectively. We explore different values for $mac\_proj.numSmooth$, ranging from 2 to 32 and the impact of having $mac\_proj.useWCycle$ turned on or off.

From our weak scaling tests, we were able to understand how well Chombo 4 scales as we increase the problem size and the amount of computational resources. Ideally, we would want to ensure the runtimes to remain roughly constant as we increase the computational resources and workload. We found that Chombo 4 scales the best when the number of smoothing is at a minimum and w-cycle is turned off. The runtimes exhibit a monotonically increasing relationship that appears roughly linear for lower smoothing counts but demonstrates an increase in the slope of the relationship for higher smoothing counts at 64 processors ($nx = 128$).

Furthermore, we are able to use the same time tables to determine the optimal parameters for $mac\_proj.numSmooth$ and $mac\_proj.useWCycle$ that will return the quickest runtimes. For all of our problem sizes, $mac\_proj.numSmooth$ = 2 and $mac\_proj.useWCycle$ = false runs the quickest compared to the same problem sizes with higher smoothing count and w-cycle turned on. Thus, these are the optimal parameters for future users of Chombo 4 if they wish to run $EBINS$ using the least computational resource. In the same plots, we suspect that agglomeration in Chombo 4 is functioning correctly. The best performance is achieved when $mac\_proj.numSmooth$ is set to 2, suggesting that geometric multigrid is effectively mitigating the low-frequency error components without the need for numerous iterations by basic iterative solvers.

From the output of the weak scaling tests, we find that higher smoothing results in less iterations of geometric multigrid or quicker rate of convergence. Specifically, when we increase the number of smoothing by a factor of 4, the number of geometric multigrid iteration decreases by 1. Thus, $mac\_proj.numSmooth = 32$ gives us the quickest rate of convergence and $mac\_proj.numSmooth = 2$ gives us the slowest rate of convergence in our weak scaling tests. However, higher smoothing performs poorer than the lower smoothing. Meanwhile, tests with w-cycles converge with fewer geometric iterations and have longer runtimes than tests without w-cycles due to w-cycles having more smoothing. Thus, we show the need for geometric multigrid since having more geometric multigrid iterations is less computationally expensive than having more smoothing.

For future work, we hope to investigate the performance of Chombo 4 on different examples to gain a more comprehensive understanding of its performance bottlenecks. As of right now, one limitation of Chombo 4 is that the geometric multigrid fails to converge for the $\_inflow\_outflow$ input of $EBINS$ as we increase the problem size to $nx = 64$ and bigger. In summary, we have determined the optimal parameters and assessed the weak scaling of Chombo 4 using the $EBINS$ example. Our study also highlights the necessity of employing geometric multigrid when numerically solving partial differential equations

References

[1] M. Adams, P. Colella, D. T. Graves, J.N. Johnson, N.D. Keen, T. J. Ligocki, D. F. Martin, P.W. McCorquodale, D. Modiano, P.O. Schwartz, T.D. Sternberg and B. Van Straalen, Chombo Software Package for AMR Applications - Design Document, Lawrence Berkeley National Laboratory Technical Report LBNL-6616E.

[2] “On Solvers: The V-Cycle Multigrid.” COMSOL. Accessed October 18, 2023. https://www.comsol.com/blogs/on-solvers-v-cycle-multigrid/.

[3] William L. Briggs, Van Emden Henson, and Steve F. McCormick. 2000. A multigrid tutorial (2nd ed.). Society for Industrial and Applied Mathematics, USA.