Parallel Scaling Performance of MOOSE on TACC

Alexy Skoutnev

November 2021

1 Introduction

Modeling physics related problems has been an ongoing venture since the dawn of science. Mathematical principles are used to extrapolate reality to scientific concepts. In this work, mathematical operators, for a multi-physics problem, were approximated with the help of parallel computing. Parallel computing resources such as MPI and MOOSE were used to investigate the scaling performance on Frontera located at the University of Texas at Austin. Message Passing Interface (MPI) is a standard parallel programming library that is used to implement parallel communication within a computing architecture. MPI allows large computer architectures to efficiently communicate with each of its cores to maximize its computational power. Multiphysics Object-Oriented Simulation Environment (Moose) is a parallel finite element framework used to solve a range of partial differential equations driven physics problems. In this paper, MOOSE was utilized to approximate the linear and non-linear heat conduction problem via a simple 3 dimensional cube. Performance test statistics such as strong efficiency, and weak efficiency were performed to evaluate the scale ability of MPI and MOOSE. The MPI performance models are used as the base work comparison for MOOSE performance models. An internode and intranode study was performed on MPI and MOOSE to evaluate the scalability in one or multiple nodes. Within each test benchmark, a strong or weak scale performance test was executed with a similar problem size and starting conditions.

2 Requirements

MOOSE and MOOSE's internal module, Heat Conduction, was used to find and solve the steady linear, and non-linear heat conduction equation. The MOOSE application code is mostly written in C++ while being compiled and executed with the use of makefiles and Unix scripts. The MPI performance test was written in C while being compiled and executed with Unix scripts.

- Evaluate the scale performance of a simple MPI program to benchmark the performance of MPI on Frontera.
- Numerically approximate the heat conduction problem using finite element principles.
- Model the weak and strong scale ability of MOOSE using MPI on Frontera
- Perform scalability on linear and non-linear heat conduction via weak and strong scaling.
- Model the key performance differences between MPI scaling and MOOSE scaling.

3 Benchmarking MPI Performance on Frontera

Scalability relates the computational power of computing hardware as the problem size varies in magnitude. A common metric in displaying scaling performance is indicated by strong and weak scaling methods. In strong scaling, the number of processors are increased while the problem size stays constant. Consequently, the workload per processor is decreased. Strong scaling is used to evaluate long-term (time intensive) computational tasks, and determine if a large sized problem can be completed in reasonable amount of time. Strong scaling is gauged by the strong efficiency metric as seen below,

$$E_s = \frac{t(1)}{t(N) * N} \tag{1}$$

Where $t(1)$ is the execution time for one processor and $t(N)$ is the execution time for N processors. In optimal cases, the efficiency is around 1 resulting in a one to one scaling factor which is represented as the speedup. The speedup in parallel computing is defined as (for both strong and weak scaling),

$$S_p = \frac{t(1)}{t(N)} \tag{2}$$

In most cases, one-to-one scaling is not possible because of communication bias and hardware capabilities. In memory-intensive problems, weak scaling is used to evaluate the parallel computational performance. To perform weak scaling, both the number of processors and problem size are increased resulting in a constant workload per processors. Weak scaling is mainly used for memory prone problems where memory is the dominate issue in the computational task. Weak scaling is measured by the weak efficiency metric which is derived as,

$$E_w = \frac{t(1)}{t(N)} \tag{3}$$

Where $t(1)$ is the execution time for one processor and $t(N)$ is the execution for N number of processors. In using the weak scaling metric, theoretically, weak efficiency should be around one and have similar execution time as we increase the number of processors. Both weak and strong scaling metrics are theoretically bounded at one, but in practice this is not possible due to hardware capabilities. To establish a control sample to compare performance tests between different parallel computing libraries, the MPI parallel programming library was used. With MPI implemented, a simple C program calculated the average value from a set of random values between 0 and 1 where each processor computed it own average value. The simple MPI program was used to benchmark the scaling performance on Frontera. The two benchmarks were focused on intranode testing and internode testing.

3.1 Internode Scaling Performance

In this scaling performance testing, each node utilized all of its 56 cores to push its hardware to its full potential. The scaling performance test started with one node and incremented up by one node. Within the test, two sets of data were obtained with ranges of one to twenty nodes. The problem size was a 2.327×10^9 sized array and a 2.327×10^{10} sized array for the strong scaling case. For the weak scaling case, the problem size was 2.327×10^7 sized array and a 2.327×10^8 sized array. The following problem size was chosen so that the work-load per processor would be an integer when divided by the prime factors between 0-20. Each set of nodes retrieved the amount of seconds it took to complete the computational task, and in each test iteration, a additional node was added to perform the computational task with a respective decrease in work per processor. Figure 1 and 2 display the time needed to complete the computational task over the number of processors used to perform that task. This would be the standard in this paper by displaying the execution times and calculated efficiency over the amount of processors configured in parallel. Figure 1 displays strong scaling over a range of 1120 cores of problem size 2.327×10^9, while Figure 2 displays a equivalent configuration of problem size 2.327×10^{10}.

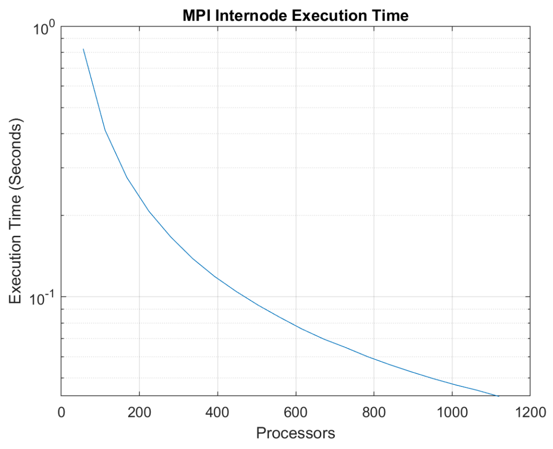

Figure 1: Problem Size 2.327×10^9 Strong Scaling

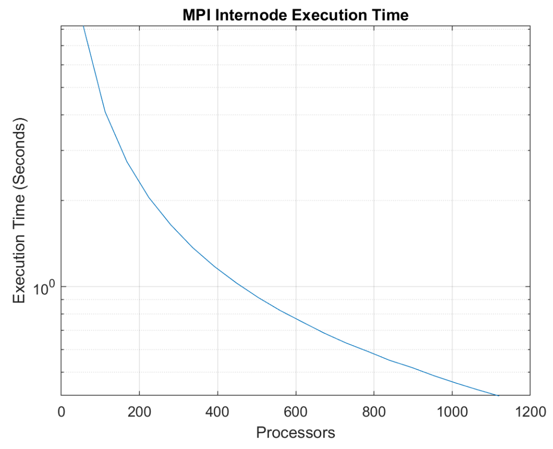

Figure 2: Problem Size 2.327×10^{10} Strong Scaling

As seen in many of the strong scaling times, the execution times decreases exponentially as the number of processors are increased. There are many memory limits with performance testing when evaluating extremely large problem sizes. Due to the limits, the problem size is set right before the memory issue occurs. In the MPI data sample, the execution times are fairly small which causes large time deviations if a slowdown occurs in the hardware or executing software. From the execution times, the strong scaling metric can be computed using (1) which results in Figure 3 and 4.

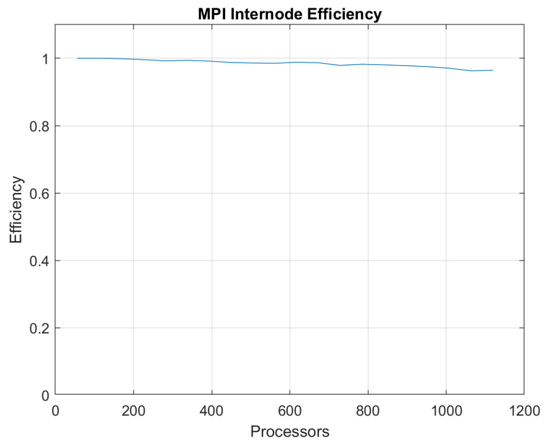

Figure 3: Problem Size 2.327×10^9 Strong Efficiency

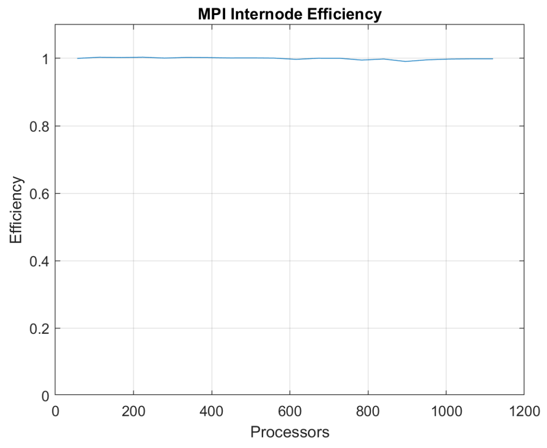

Figure 4: Problem Size 2.327×10^{10} Strong Efficiency

The strong scaling efficiency slowly decays as the number of processors are increased. Larger time deviations are seen when approaching the upper limit of the problem size due to communication bias between the processors however in MPI performance test it is barely noticeable.

Weak scaling was performed on the same range of nodes. The problem size was a 2.327×10^7 and 2.327×10^8 sized array where it represented the work amount per processor. The weak scale execution times are seen in Figure 5 and 6.

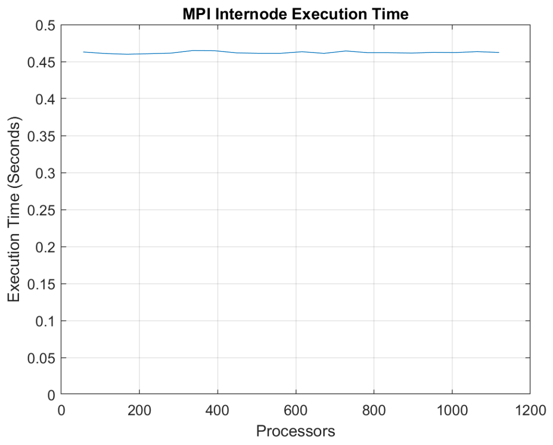

Figure 5: Problem Size 2.327×10^7 Weak Scaling

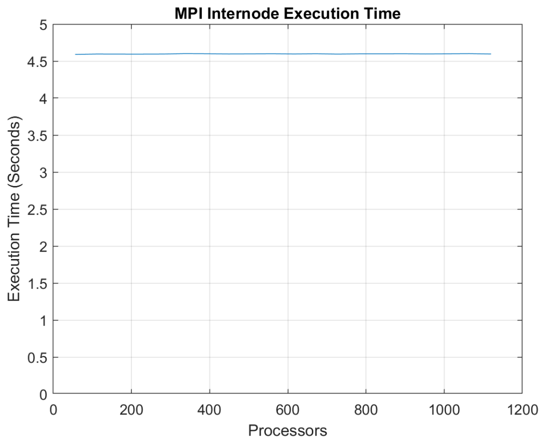

Figure 6: Problem Size 2.327×10^8 Weak Scaling

The execution times are expected to be similar since the work per processors are equivalent. It is noticed that increasing the work per processors flattens out the execution time curve in weak scaling. With the weak scaled execution times, the weak efficiency is computed via (3). The results are displayed in Figure 7 and 8 with a constant work per processor of 2.327×10^7 and 2.327×10^8 elements.

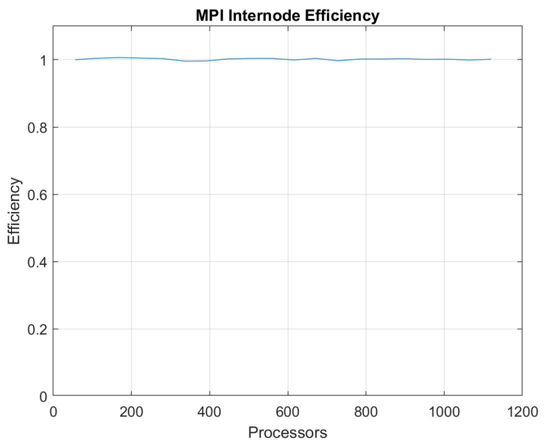

Figure 7: Problem Size 2.327×10^7 Weak Efficiency

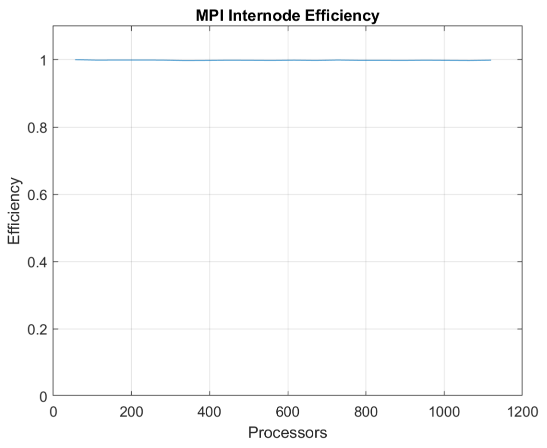

Figure 8: Problem Size 2.327×10^8 Weak Efficiency

3.2 Intranode Scaling Performance

Using one node is efficient in many cases where the computational size is not computing intensive and increasing the amount of nodes is detrimental in some cases. However, the problem size can not be larger than a single node memory capacity thus limits the potential computing power. The intranode section presents strong and weak scaling performance within one node. For the strong scaling case, the problem size was set to 2.327×10^8 and 2.327×10^9 which is smaller than the internode strong scaling problem size due to memory issues. Figure 9 and Figure 10 display the execution times respectively.

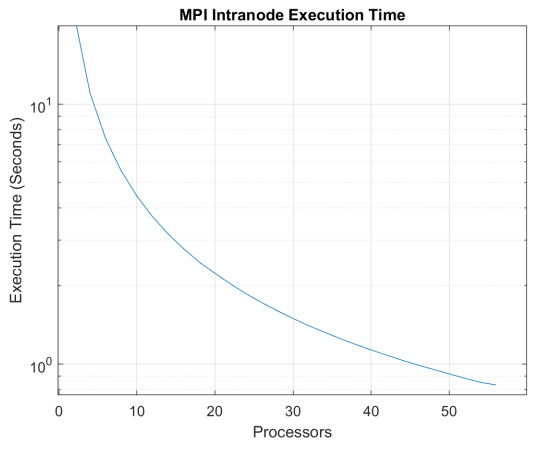

Figure 9: Problem Size 2.327×10^8 Strong Scaling

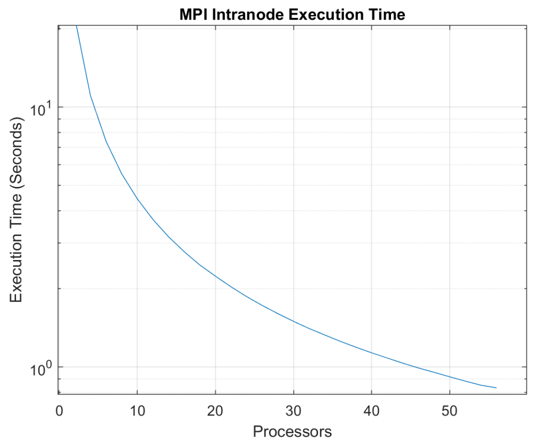

Figure 10: Problem Size 2.327×10^9 Strong Scaling

Internode and intranode strong scaling performance testing is similar because the problem requires one simple communication call between the processors once the average is found. The similarity can be seen in the efficiency values in Figure 11 and Figure 12 when compared to Figure 3 and 4.

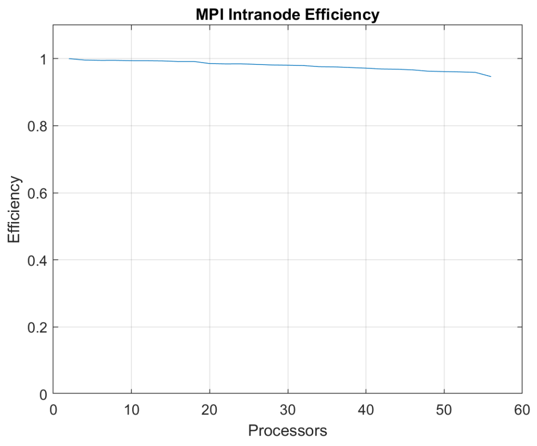

Figure 11: Problem Size 2.327×10^8 Strong Efficiency

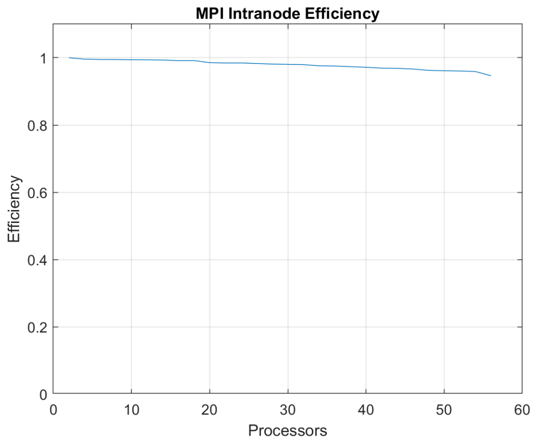

Figure 12: Problem Size 2.327×10^9 Strong Efficiency

As stated before, increasing the problem size causes the scaling efficiency to remain more stable as it is able to handle more sudden deviations in time with respect to execution time, communication travel time, etc. The execution time is more stable in Figure 14 compared to Figure 13 because the problem size is 10 times larger.

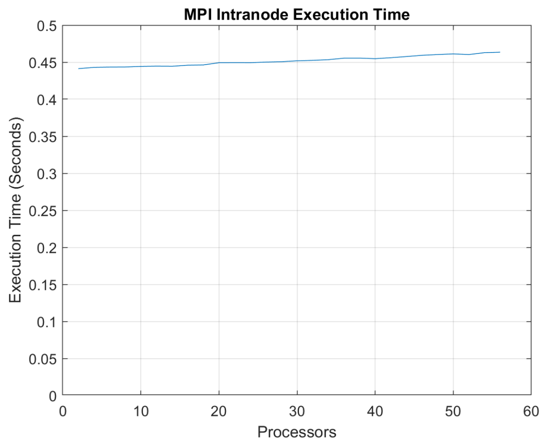

Figure 13: Problem Size 2.327×10^7 Weak Scaling

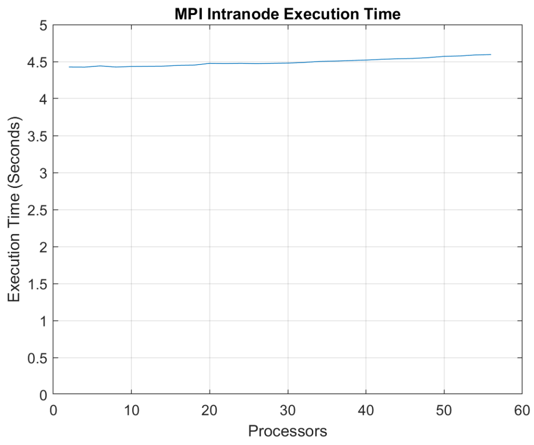

Figure 14: Problem Size 2.327×10^8 Weak Scaling

From the weak scaling efficiency figures seen in Figure 15 and 16, Figure 16 has a better weak scaling efficiency due to the increased workload per processor. Having the processor at loaded capacity is important in scaling benchmarks.

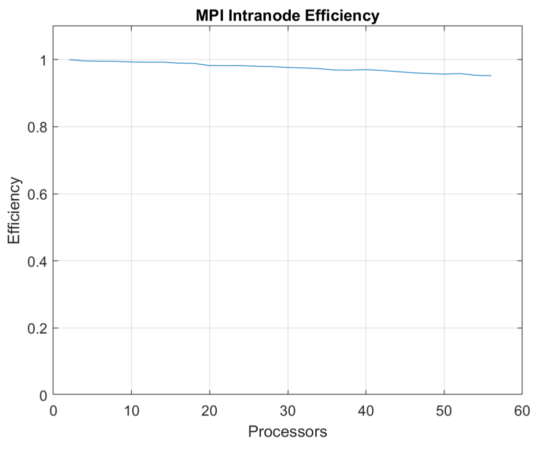

Figure 15: Problem Size 2.327×10^7 Weak Scaling Efficiency

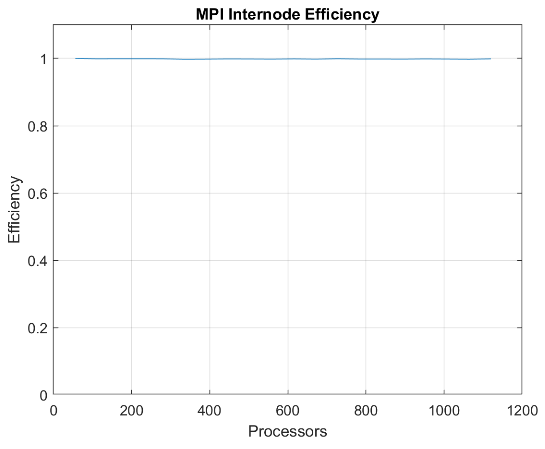

Figure 16: Problem Size 2.327×10^8 Weak Efficiency

All these figures will be used in concatenation with the scaling performance of MOOSE.

4 Defining a Diffusion Test Problem in MOOSE

With the implementation of MOOSE, many multi-physics problems can be approximated and solved. In this study, diffusion is the main focus since it's fundamental in computational nuclear engineering. In particular, the well known heat conduction equation, first developed by Joseph Fourier, was considered the basis for the diffusion performance experiment. The heat conduction equation is derived as,

$$-\nabla \cdot k\nabla T = 0 \tag{4}$$

Where k is the thermal conductivity, and ∇T is the temperature gradient in terms of spatial variables x, y, z. In this study, the transient response was not included where the temperature was related to time. Linear and non-linear partial differential equations were tested to compare scaling performance between them. For non-linearity, the thermal conductivity was coupled as a simple function depending on the temperature in the medium. The thermal conductivity in the non-linear problem was defined as,

$$k(T) = 5 + 10T \tag{5}$$

Where in the linear case, the thermal conductivity was set to a constant value of 18. The performance test made use of MOOSE's optimized automatic differentiation kernels for better execution times and scaling efficiency. In all of the test cases, the generated mesh was a 3-dimensional cube of unit length. The boundary condition was a Dirichlet boundary condition where the left-side was set to 500 kelvin, while the right-side was set to 1000 kelvin. The problem was a finite element problem that made use of the Newton-Raphson method to approximate the solution for the partial differential equation. Like in the MPI study, strong and weak scaling was performed for intranode and internode test evaluation. In comparison to the MPI performance test, the execution times for many MOOSE problem were 10 to 1000 times longer. Because of this, the MOOSE problem was optimized and configured to different specifications in order to reduce the performance time and help parallel efficiency. Most of the execution time is spend on building the mesh than finding the solution to the partial differential equation.

5 Single Node Scaling Performance

Similar to what is seen in the MPI intranode scaling performance section, the heat conduction equation was subject to strong and weak scaling performance. The strong scaling problem size was a 200^3 partitioned cube mesh and the weak scaling problem size started with a 90^3 partitioned cube mesh. At each iteration, each dimensional of the weak scaled cube mesh was multiplied by a factor of $\sqrt[3]{n}$, where n is the amount of processors in parallel so that the work per processors is constant. The input files made use of the distributed rectilinear mesh generation feature in MOOSE for better parallelism. There were 8,120,601 nodes and 8,000,000 elements for the strong scaling case, where the number of equally spaced partitions were based on the number of processors. Multiple execution times were measured in the performance test to evaluate what part of MOOSE is scaling properly. The four time measurements that were taken were total execution time, self FEProblem::solve, child FEProblem::solve, and total FEProblem::solve time. The total time was a combination of the mesh building process and the finite element solver using Newton-Raphson that represents the whole application execution time. The self FEProblem::solve (seen as Self FEProblem) represents a customized heat conduction kernel that receives each quadrature point computation from its parent class. Self FEProblem is mostly used for communication purposes. The parent class is represented by the child FEProblem::solve execution time (seen as Child FEProblem) where most of the computational work is present. Total FEProblem::solve (seen as Total FEProblem) is mostly the combination of Self FEProblem and Child FEProblem execution times.

5.1 Strong Scaling Analysis

In this section, the strong scaling performance will be displayed by four different timing metrics as discussed above. Below in Figure 17, the total execution is consisted of mesh generation and the solver iteration process.

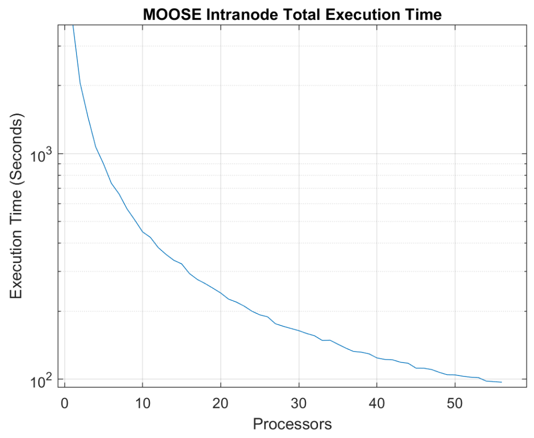

Figure 17: Problem Size 200^3 Strong Scaling

On average it took around 13-18 linear iterations and 3 nonlinear iterations for the approximation to converge. At each iteration, the residual decreased about an order in magnitude. Figure 18 displays the total strong efficiency and it can be seen that the efficiency is weaker compared to MPI one node strong scaling (Figure 11 and 12).

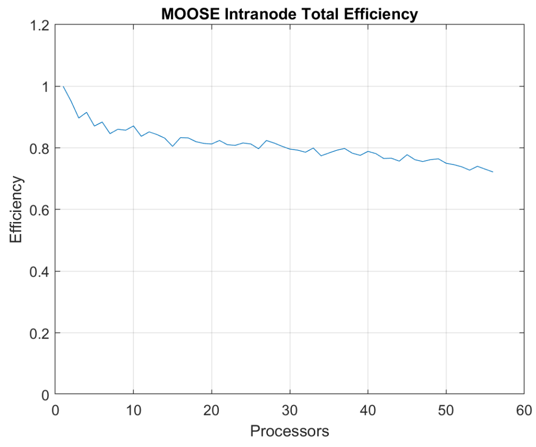

Figure 18: Problem Size 200^3 Strong Efficiency

Figure 19 displays Self FEProblem execution times and it can be seen that Self FEProblem execution times fluctuates more often compared to the other four time metrics that decreases steadily. Throughout the study, Self FEProblem performed the worst compared to all the other time metrics.

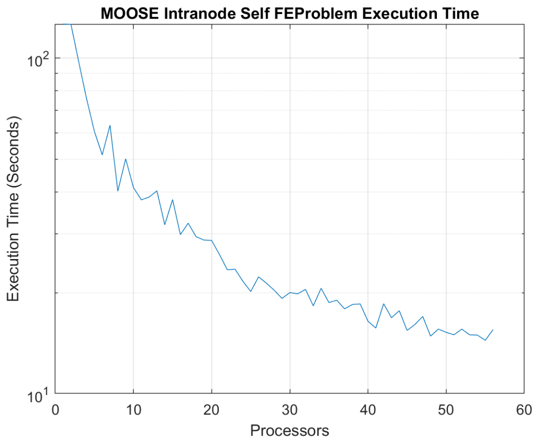

Figure 19: Problem Size 200^3 Strong Scaling

Self FEProblem poor performance can be seen Figure 20 where the efficiency drops down to 12% towards the end. At 56 processors, the node is utilizing it full potential and requiring each 56 core to communicate with each other. Since most of the communication between the processors is happening here, the execution time is shorter than child FEProblem execution time where most of the computationally work is seen.

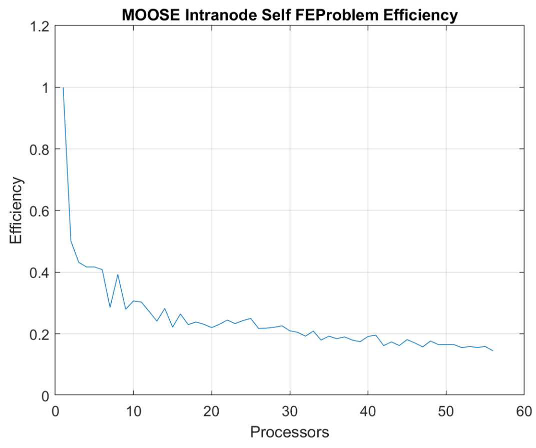

Figure 20: Problem Size 200^3 Strong Efficiency

The best scaling performance is seen in the Child FEProblem time metric where the efficiency stays around one. Child FEProblem is a MOOSE optimized class which is designed to have parallelism while being scaled. Figure 21 and 22 displays how Child FEProblem optimization takes advantage of parallel computing by cutting the execution time the most out of all the four time metrics.

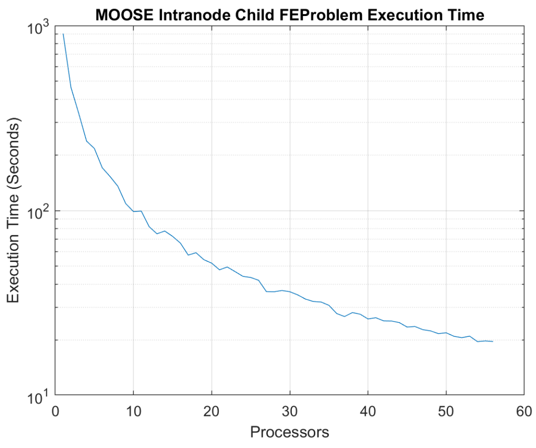

Figure 21: Problem Size 200^3 Strong Scaling

Figure 22 presents the strong efficiency of Child FEProblem where strong scaling best performs. Most of the computational work is seen in Child FEProblem and usually takes a longer process executing the code compared to Self FEProblem.

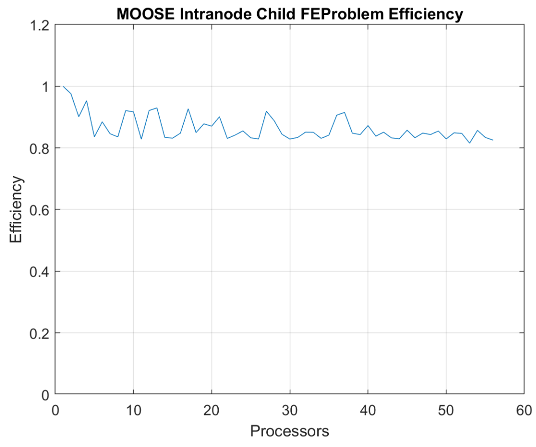

Figure 22: Problem Size 200^3 Strong Efficiency

Figure 23 and 24 is the combination of Child FEProblem and Self FEProblem displaying the overall scale ability of the executioner within MOOSE. If the Total FEProblem time was subtracted from the overall application execution time, the difference represents the amount of time it took for the mesh to be created. In most cases, the mesh generation time is larger than Total FEProblem time (time to solve the partial differential equation).

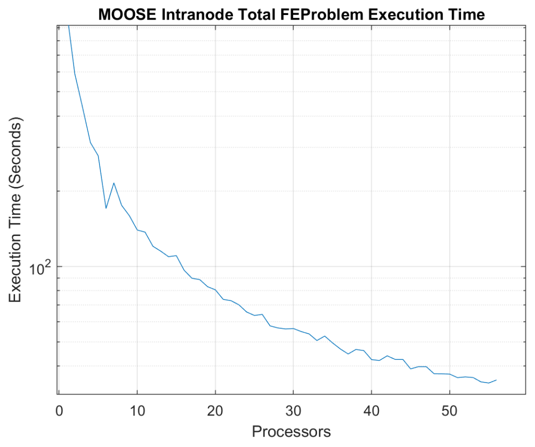

Figure 23: Problem Size 200^3 Strong Scaling

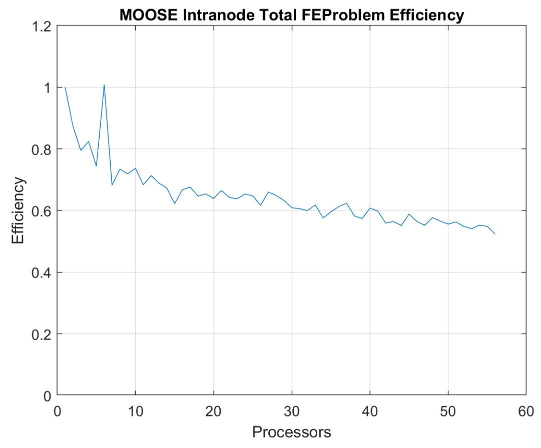

Figure 24: Problem Size 200^3 Strong Efficiency

5.2 Weak Scaling Analysis

The same four time metrics were applied to weak scaling as seen in the strong scaling intranode analysis. The problem size started with a 90^3 partitioned cube and in each iteration the problem size increased by the root cubic factor. With having the problem size starting at 90^3, this allowed for weak scaling to utilize most of the processor's computational power. If the starting problem size was larger than 90^3, then the memory allocated for a node will be overfilled and cause the program to crash. While in weak scaling, each processor handled 729,000 elements each iteration. Figure 25 displays the total execution times for the weak scale analysis.

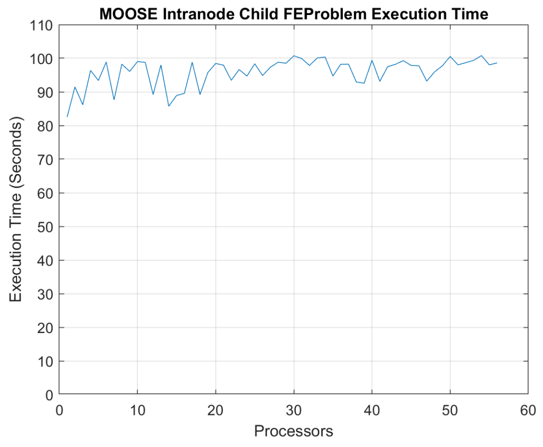

Figure 25: Problem Size 90^3 Weak Scaling

The execution time increases around 20% when utilizing most of the processors in the node unlike what is seen in MPI weak scaling test (Figure 13 and 14). The weak efficiency is presented by Figure 26 seen below where the weak efficiency drops to 69% towards the end.

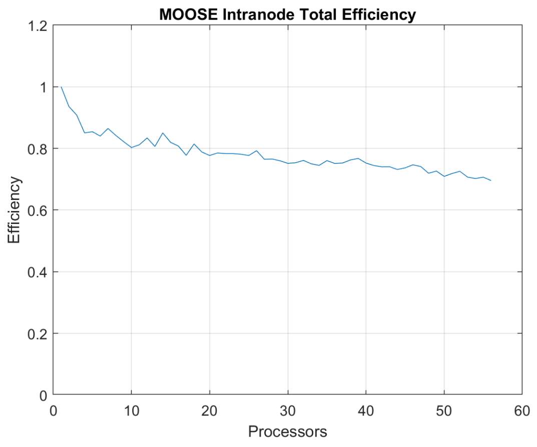

Figure 26: Problem Size 90^3 Weak Efficiency

Most of the slowdown occurs in Self FEProblem where there are noticeable increases in execution times as the number of processors are increased. It can be noticed that MOOSE's Self FEProblem scales inefficiently compared to MPI performance benchmarks, and MOOSE's Child class. By looking at Figure 20 and 28, intranode strong scaling also performs the worst in its Self FEProblem class. Self FEProblem seems to cause the largest slowdowns while being scaled by examining the weak scaling results in Figure 27 and Figure 28.

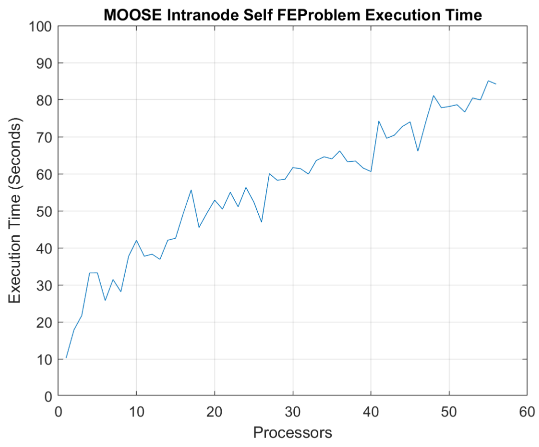

Figure 27: Problem Size 90^3 Weak Scaling

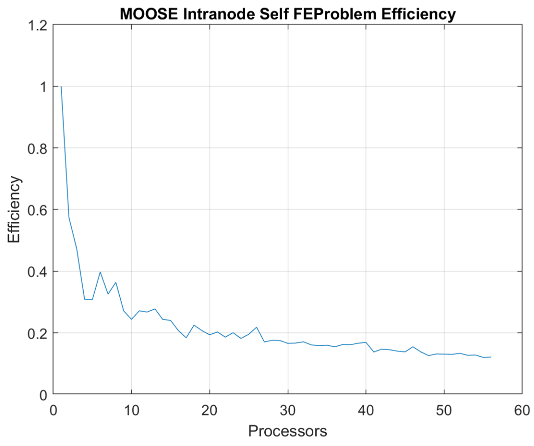

Figure 28: Problem Size 90^3 Weak Efficiency

On the contrast to FEProblem Self benchmarks, FEProblem Child maintains good parallelism while being weak scaled. Figure 29 depicts intranode weak scaling of MOOSE.

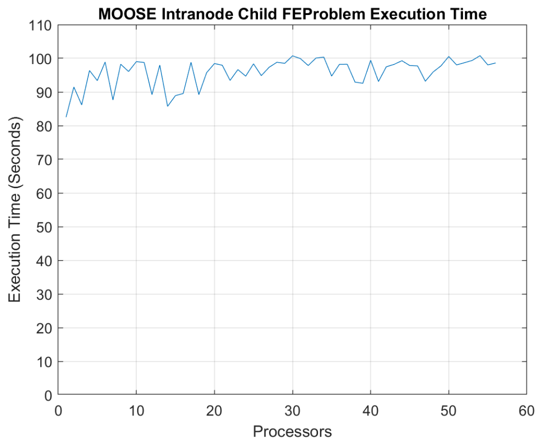

Figure 29: Problem Size 90^3 Weak Scaling

From Figure 29, Figure 30 displays intranode weak efficency as the number of processors are increased from one to 56. The weak scaling efficiency of Figure 30 mirrors the efficiency seen in Self FEProblem in strong intranode scaling by viewing Figure 22. It should be noted that Figure 30 does not follow perfect scaling efficiency as in the MPI case (Figure 14).

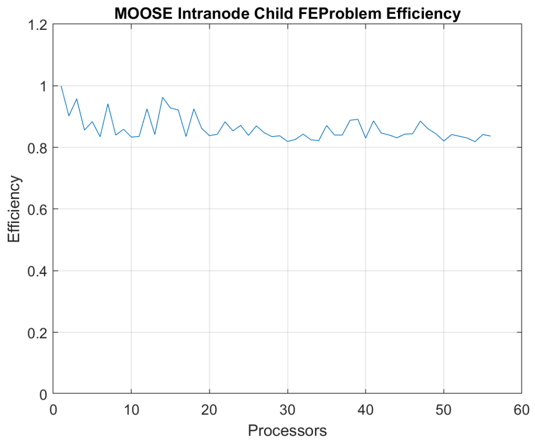

Figure 30: Problem Size 90^3 Weak Scaling

Total FEProblem is the combination of Self FEProblem and Child Problem execution times. The draw backs appearing in Self FEProblem is reflected in Total FEProblem causing the executioner time to rise faster than the overall application time. Figure 31 and 32 displays the total FEProblem execution time and weak efficiency respectively.

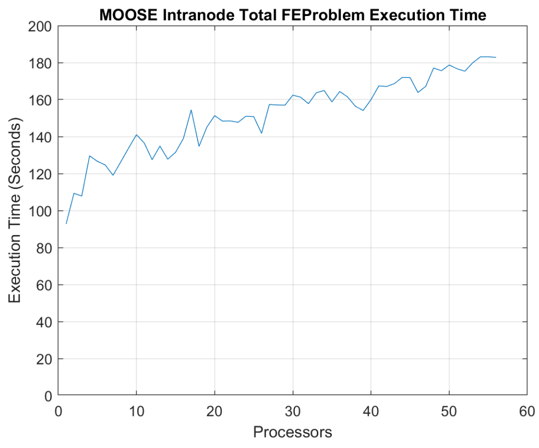

Figure 31: Problem Size 90^3 Weak Scaling

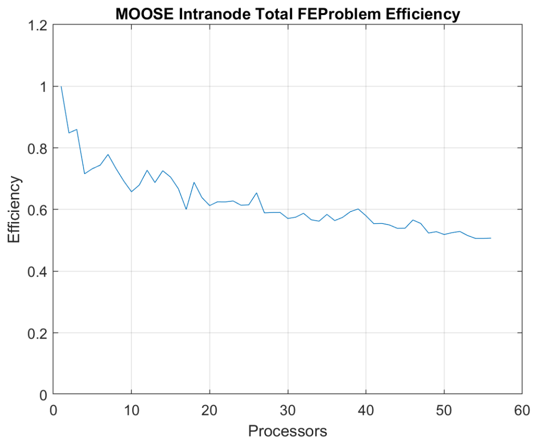

Figure 32: Problem Size 90^3 Weak Efficiency

6 Multi-Node Scaling Performance

Extreme sized computational problems requires the use of many nodes where a complex mesh excels the limits in computational power and memory allocation for a single node. Parallel computing between multiple nodes is a common solution in solving immense computational problems. To have a link between internode and intranode scaling, the relative mesh sizes were equal to each other. For the internode strong case, the problem size was a 200^3 partitioned cube that scaled from 1 node to 50 nodes. Likewise, for the weak scaling test, the starting problem size was a 90^3 partitioned cube that scaled from 1 node to 50 nodes. A distributed mesh was used to spilt up the work between the processors and each core did a small portion of the mesh. However, the overall mesh was loaded onto each core which limits memory capabilities for very large meshes. With the increase of cores, more communication bias and parallel inefficiency will be seen in the performance test. As done before, four time metrics will be used to evaluate the scale ability of each segment of the MOOSE solver.

6.1 Internode Strong Scaling Analysis

The performance tests were similar to what was seen in intranode strong scaling analysis where the total problem size was set to 200^3 however in each iteration the number of cores were increased by 56 instead of one. The total execution times over the range of processors is seen in Figure 33.

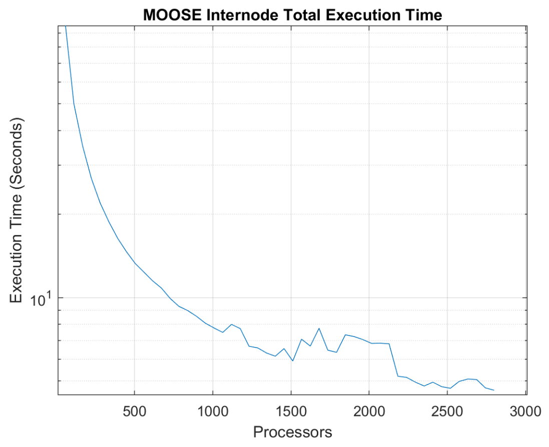

Figure 33: Problem Size 200^3 Strong Scaling

There were 13-15 linear iterations and 3 nonlinear iterations on average before the executioner converged to a low enough residual of 10^{-7} magnitude. Figure 34 shows the overall strong efficiency of the internode scaling performance below,

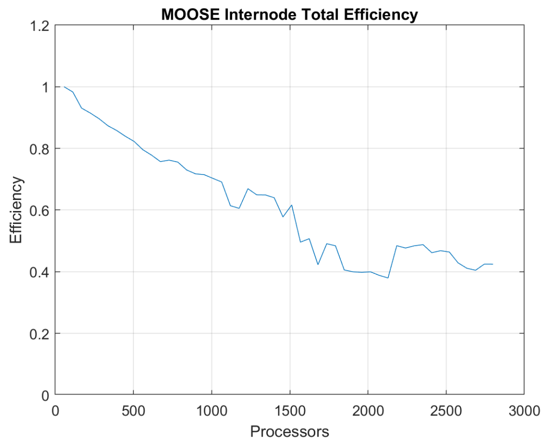

Figure 34: Problem Size 200^3 Strong Scaling Efficiency

The strong efficiency as seen in Figure 34 performs poorly compared to Figure 18 of similar problem size in the intranode strong scaling study. About midway, the parallel computing advantage diminishes as seen by the drop of efficiency. The effects of communication bias can be seen between the efficiency discrepancy of Figure 34 and Figure 18. The communication problem within internode scaling can be noticed in Figure 35 where Self FEProblem execution times are displayed.

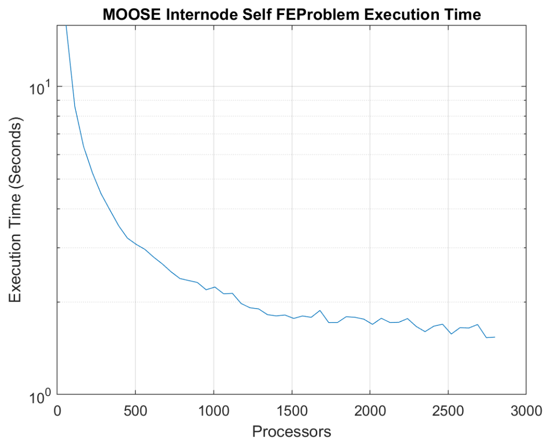

Figure 35: Problem Size 200^3 Strong Scaling

Most of the slowdowns in the executioner is seen in Self FEProblem where communication between the processors is poor when utilizing more than one node. Opposite results were seen in Figure 1 and Figure 2 where MPI was used for internode scaling benchmarks. Figure 36 shows internode scaling efficiency for MOOSE's communication class.

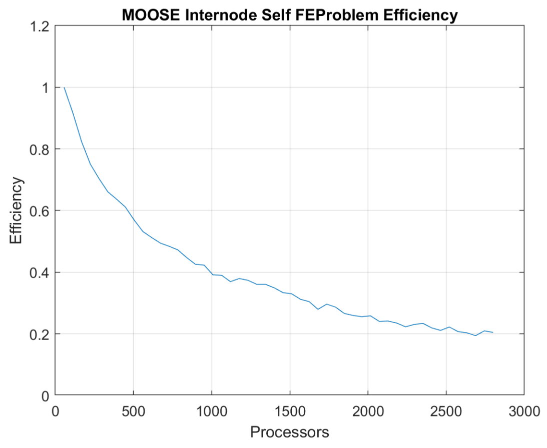

Figure 36: Problem Size 200^3 Strong Scaling Efficiency

On the other hand, Child FEProblem scales significantly better than Self FEProblem and resembles the efficiency seen in the intranode case (Figure 22). Figure 37 displays Child FEProblem execution times below.

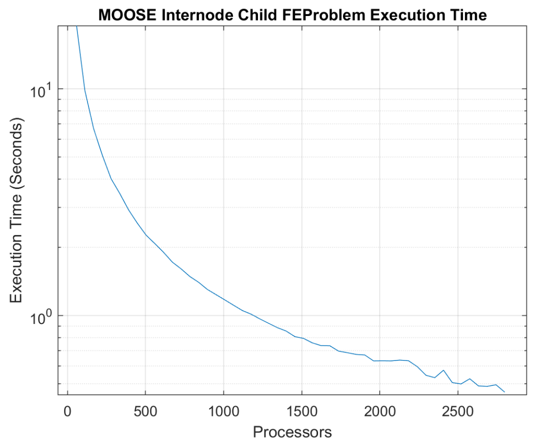

Figure 37: Problem Size 200^3 Strong Scaling

Parallel computing advantage is noticeable in Figure 37, where each iteration cuts the execution time by significant amount. Figure 38 displays the strong scaling efficiency over the range of 2,800 processors. Surprisingly, Figure 38 displays internode Child FEProblem strong efficiency and it is slightly better than the intranode case (Figure 22).

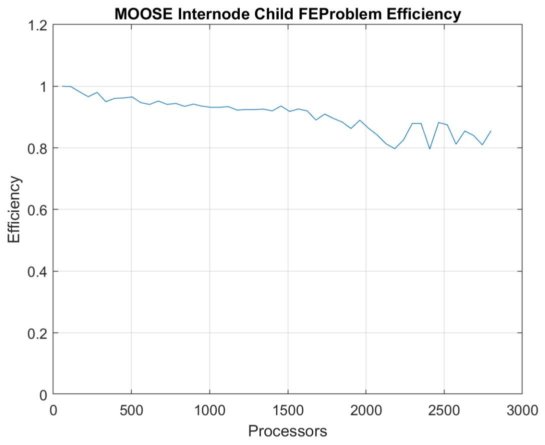

Figure 38: Problem Size 200^3 Strong Scaling Efficiency

Total FEProblem reflects the the poor scaling performance from Self FEProblem execution times, and Figure 39 displays the slow downs. Total FEProblem can be interpreted as the middle ground between Self FEProblem and Child FEProblem.

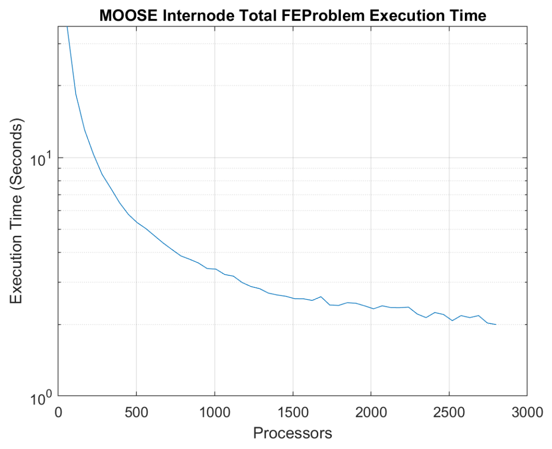

Figure 39: Problem Size 200^3 Strong Scaling

Figure 40 presents the strong scaling efficiency of Total FEProblem scaling where the efficiency goes down to 35% at full 50 node capacity.

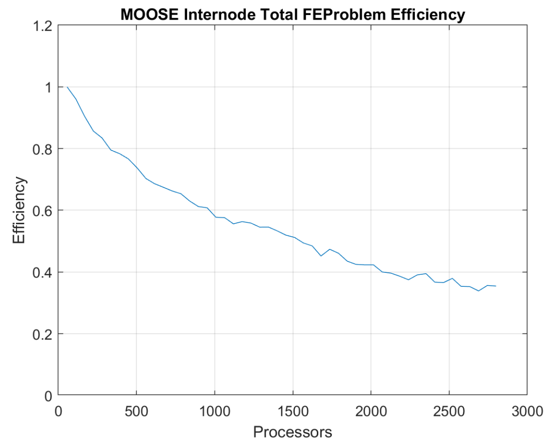

Figure 40: Problem Size 200^3 Strong Scaling Efficiency

6.2 Internode Weak Scaling Analysis

Internode weak performance scaling tests large scale memory bound computational problems. In this section, a 90^3 partitioned cube, equivalent to what was seen in intranode weak scaling, was used. Each processor had a constant 13,017 elements per processor as the number of processors were increased. All solver configurations were identical to the strong internode scaling test. Memory allocation issue occurred after the starting size of 90^3, and the range of scaling was from 1 node to 50 nodes. It should be noted that the mesh generator is in serial configuration and is not able to split parts of the mesh between processors effectively. Figure 41 displays the total execution time for the MOOSE solver which includes the mesh building process, and the executioner time.

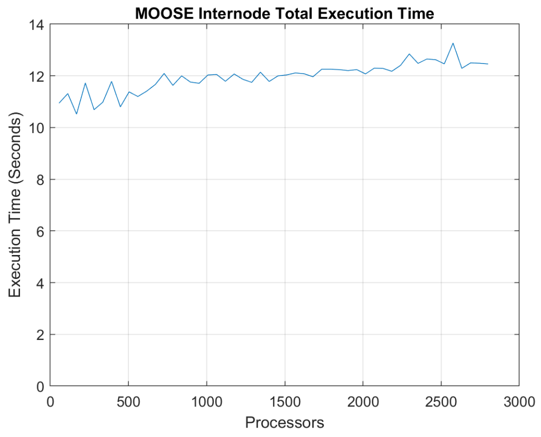

Figure 41: Problem Size 90^3 Weak Scaling

Towards 50 nodes, the completion time is 10% slower than what is seen in one node. It is expected that slowdowns should occur because the mesh generation is in a serial configuration and doesn't take advantage of parallelism. Because of this, every processor creates the entire mesh locally on one processors, and as we increase in the problem size, more of the mesh must be loaded onto the processor. Figure 42 displays the effects of a large serial mesh however the efficiency is better in the internode case compared to the intranode case.

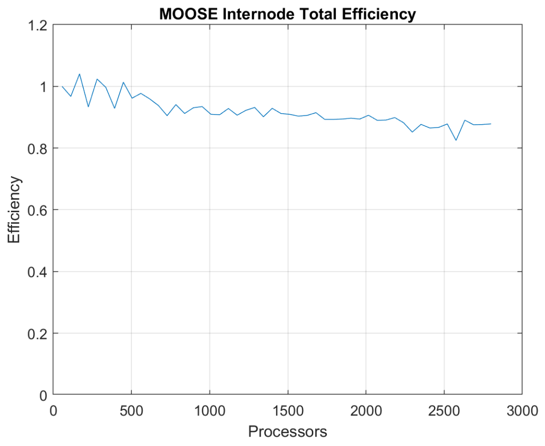

Figure 42: Problem Size 90^3 Weak Scaling Efficiency

As expected, Self FEProblem has the largest time deviations compared to the other three time metrics. In Figure 43, the execution time almost doubles from the start to end. Similar trends were seen in the intranode weak scaling study however in internode, Self FEProblem scaling perform significantly better.

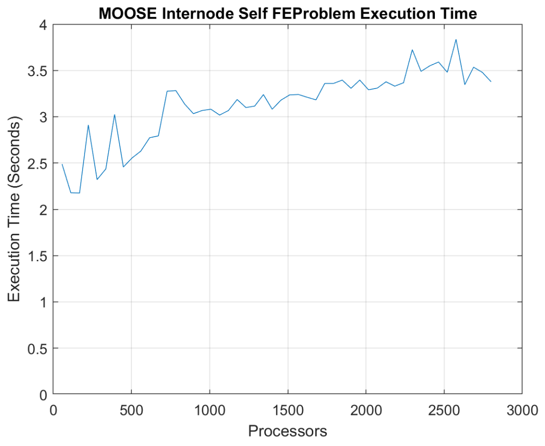

Figure 43: Problem Size 90^3 Weak Scaling

Figure 43 shows Self FEProblem efficiency and how the it slowly drop to 70% at 50 nodes compared to 12% in the intranode study (Figure 28).

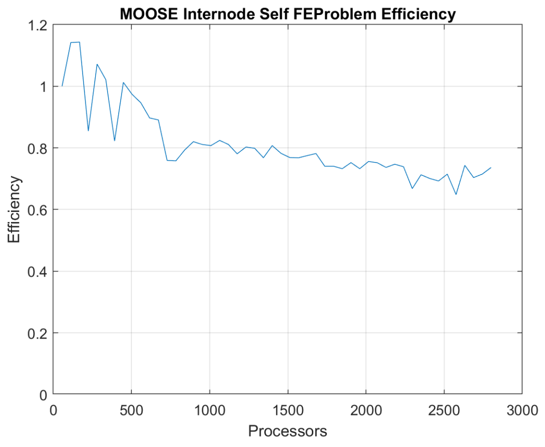

Figure 44: Problem Size 90^3 Weak Scaling Efficiency

Child FEProblem stays around 1.8 - 2.0 seconds of execution as the number of processors were increase. Child FEProblem result is comparable to MPI weak performance scaling as seen in Figure 5, 6, 13, and 14. Figure 45 displays internode weak scaling execution times.

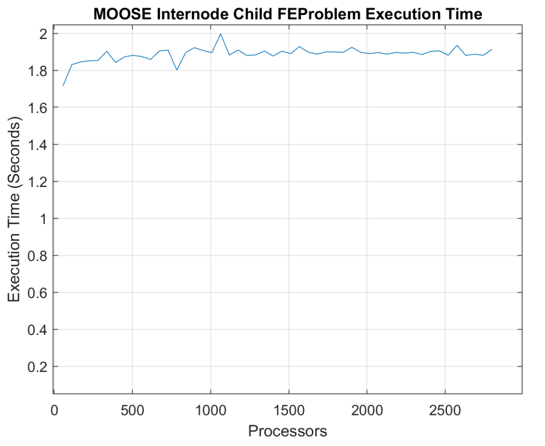

Figure 45: Problem Size 90^3 Weak Scaling

MOOSE is capable of computing in parallel effectively as seen in Figure 46, however communication between the boundary layers causes slowdowns in performance. The efficiency tends to level out towards upper middle half of the number of processors.

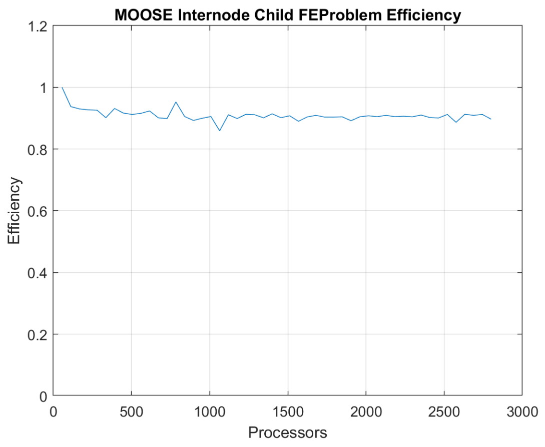

Figure 46: Problem Size 90^3 Weak Scaling

Total FEProblem represents the middle ground of MOOSE's communication class and computational class. Compared to intranode weak scaling (Figure 31 and 32), internode weak scaling performs notably better. Figure 47 presents Total FEProblem execution times below.

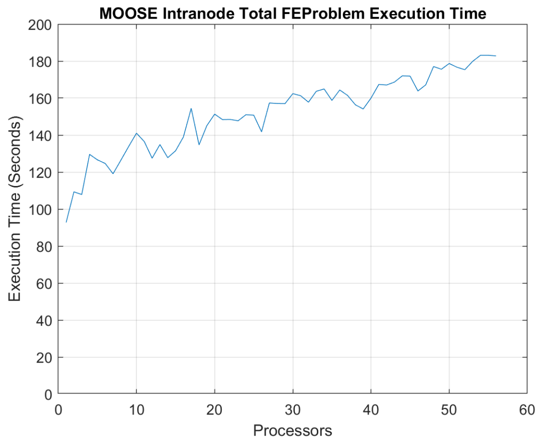

Figure 47: Problem Size 90^3 Weak Scaling

In terms of weak efficiency, internode Total FEProblem efficiency performs to the levels of intranode Child FEProblem efficiency by comparing Figure 32 and 48. Figure 48 displays intranode Total FEProblem efficiency.

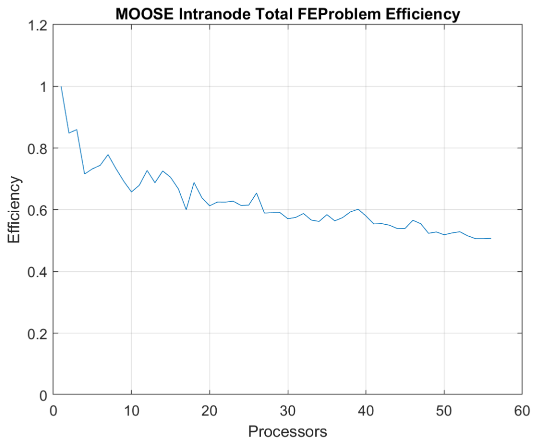

Figure 48: Problem Size 90^3 Weak Scaling Efficiency

7 Additional Explorations

MOOSE's vast capabilities allowed the testing of different performance structures. To test the potential of the solver and allow more intensive computations, the diffusion equation was made non-linear. By setting the diffusion coefficient as a function of time, as constructed by equation 5, the executioner evaluated the solution through non-linear iterations. Non-linearity caused more work per processor in terms of the number of iterations in the hope of increasing the efficiency of MOOSE's scalability. The processing amount of elements and nodes through each processor is the same for linear and nonlinear cases. Different mesh configurations were also tested to evaluate the best performing scaling benchmark. Only inbuilt MOOSE meshes were tested in this paper, such as those generated by the mesh object within MOOSE. Each type of mesh object had its corresponding partitioner. Those mesh partitions were made either by manually inputting into the input file or automatically determined by the type of mesh object when executed.

7.1 Nonlinear Diffusion

The nonlinear diffusion equation was subject to strong and weak scaling as seen in the previous sections of the paper. Similar configuration in problem size and timing metrics, total application time and total FEProblem::solve, were used to compare nonlinear diffusion scaling and linear diffusion scaling.

7.1.1 Single Node Strong Scaling Analysis

Figure 49 displays the execution time for intranode strong scaling, where the diffusion equation was nonlinear.

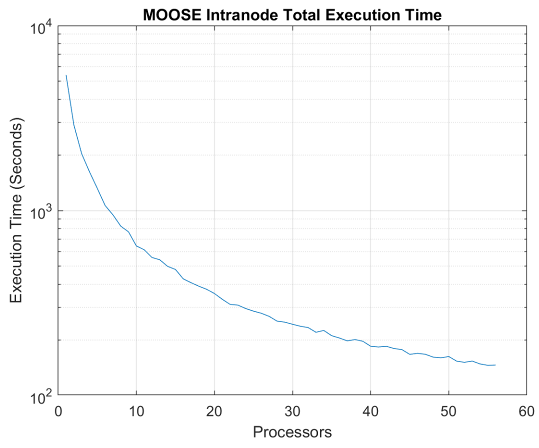

Figure 49: Problem Size 200^3 Strong Scaling

Figure 50 represents the efficiency seen in MOOSE's nonlinear solver. The efficiency were nearly identical between the linear and nonlinear diffusion cases as presented by the difference between Figure 50 and 18.

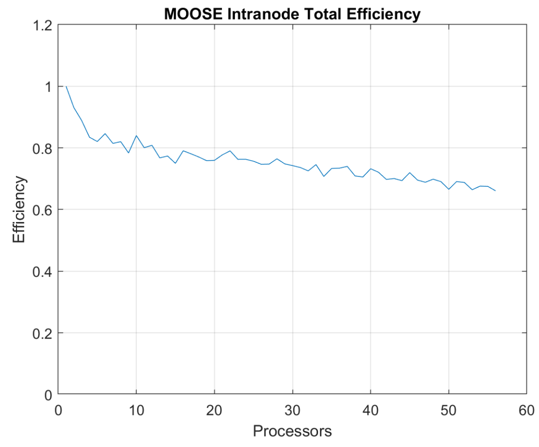

Figure 50: Problem Size 200^3 Strong Scaling Efficiency

In relation to the linear diffusion equation, nonlinear efficiency of the executioner is nearly identical to linear efficiency executioner even though there is more computations involved as seen by Figure 51 and 52.

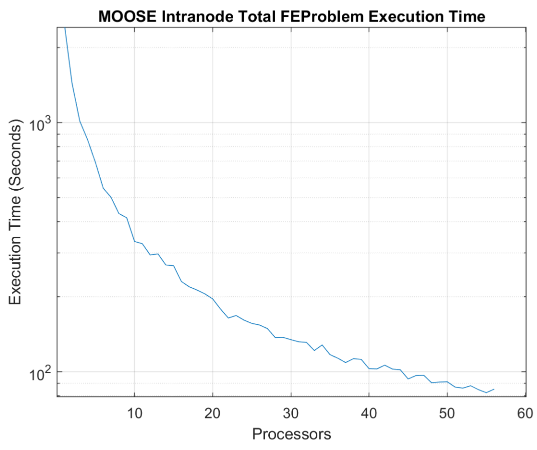

Figure 51: Problem Size 200^3 Strong Scaling

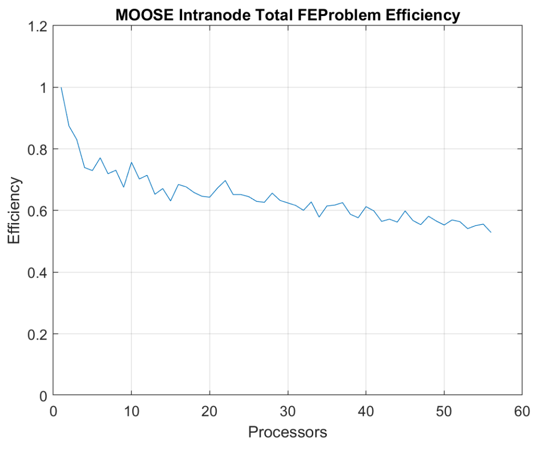

Figure 52: Problem Size 200^3 Strong Scaling Efficiency

7.1.2 Single Node Weak Scaling Analysis

In this section, weak intranode scaling was performed on the nonlinear diffusion equation, as seen in the previous sections. Figure 53 displays the execution time for a starting problem size of a 90^3 partitioned cube.

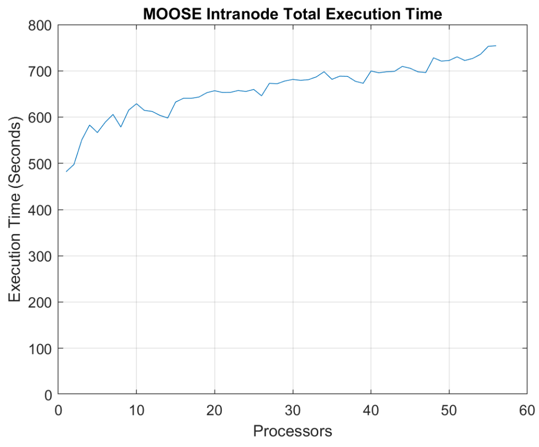

Figure 53: Problem Size 90^3 Strong Scaling

Figure 54 displays the weak efficiency calculated from the execution times of Figure 53, and the efficiency trend is slightly worse than Figure 26 (representing the linear case).

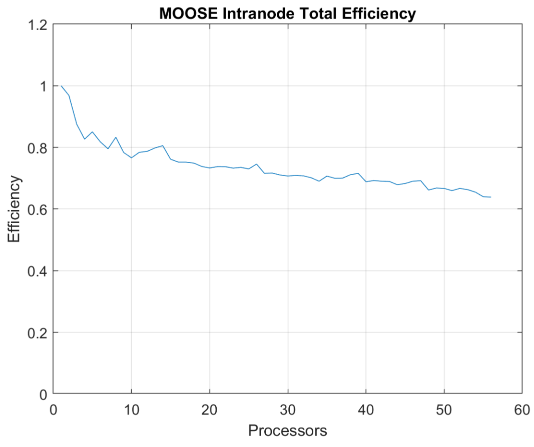

Figure 54: Problem Size 90^3 Strong Scaling Efficiency

Each nonlinear iteration had 729,000 elements being processed on each processor, however the required time to approximate the solution was longer due to higher iteration amounts. Figure 55 display larger weak scaling execution times compared to the linear case.

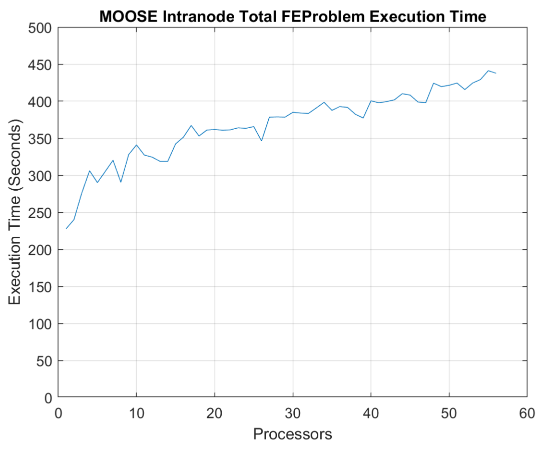

Figure 55: Problem Size 90^3 Weak Scaling

By comparing Figure 56 with Figure 32, the efficiency plot between nonlinear and linear diffusion is nearly identical. MOOSE's executioner performance does not suffer between linear and nonlinear partial differential equations.

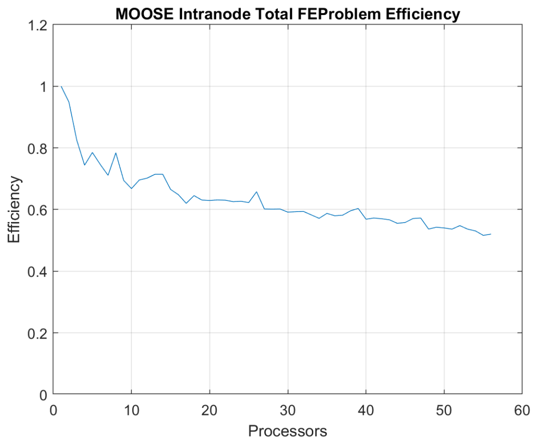

Figure 56: Problem Size 90^3 Weak Scaling Efficiency

7.1.3 Multi Node Strong Scaling Analysis

Internode strong scaling analysis was performed on MOOSE's nonlinear solver over the problem size of 200^3. Figure 57 displays internode strong scaling execution times as the number of processors varied from 56 to 2,800.

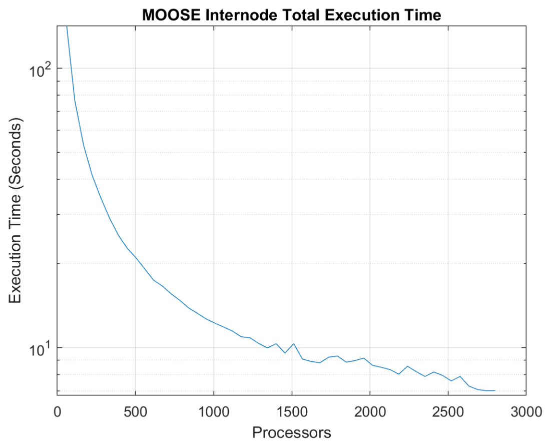

Figure 57: Problem Size 200^3 Strong Scaling

As in the intranode nonlinear diffusion section, the scaling efficiency closely mimics the performance of internode strong scaling of MOOSE. Figure 58 resembles similar performance results as Figure 34, where linear internode strong efficiency was plotted.

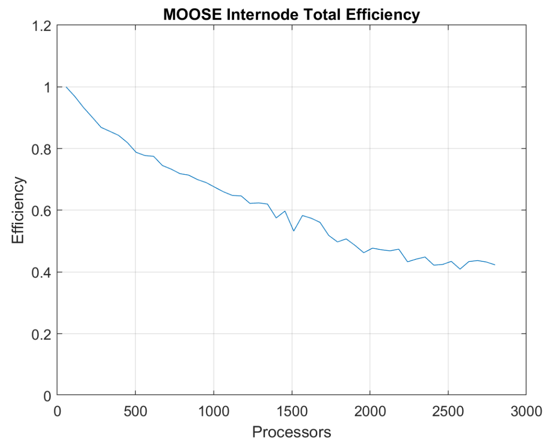

Figure 58: Problem Size 200^3 Strong Scaling Efficiency

Likewise, the nonlinear executioner performed similar to its linear counterpart, as seen in Figure 59 and 60. It is evident that multi-node scaling between the different computational classes (linear or nonlinear solvers) in MOOSE does not effect the scaling potential.

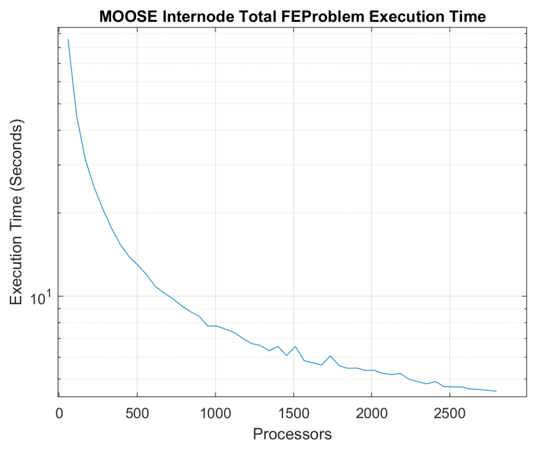

Figure 59: Problem Size 200^3 Strong Scaling

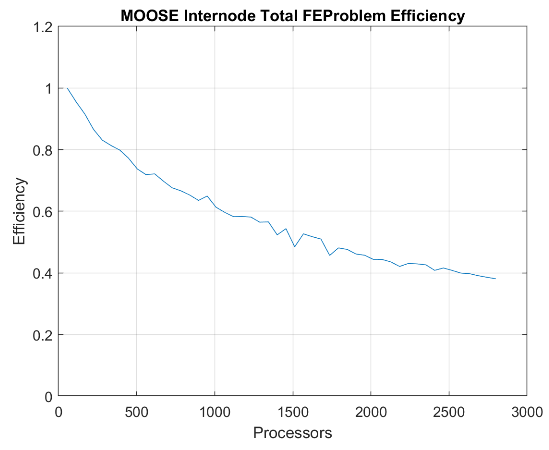

Figure 60: Problem Size 200^3 Strong Scaling Efficiency

7.1.4 Multi Node Weak Scaling Analysis

In this section nonlinear weak scaling was performed to compare with linear weak scaling. On each iteration, the processor had to perform more computations compared to the linear case. Figure 61 displays the execution times for 1 to 50 nodes while being weak scaled with the nonlinear diffusion problem.

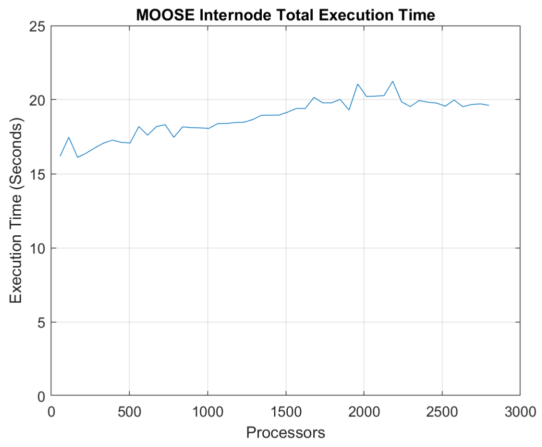

Figure 61: Problem Size 90^3 Weak Scaling

Figure 62 presents the weak efficiency of Figure 61's scaling and it can be noticed that the weak efficiency performs worse than the linear weak efficiency trend seen in Figure 42. Opposite results were seen in strong scaling between the nonlinear and linear scaling problems.

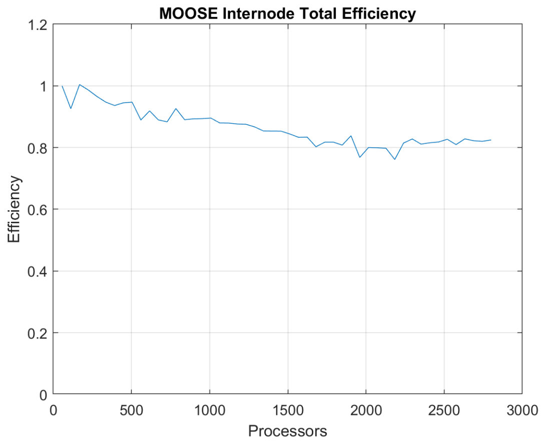

Figure 62: Problem Size 90^3 Weak Scaling Efficiency

The nonlinear diffusion equation caused the solver to take twice as long because of the added complexity. The following solver execution times are seen in Figure 63, where the execution time tops out at thirteen seconds.

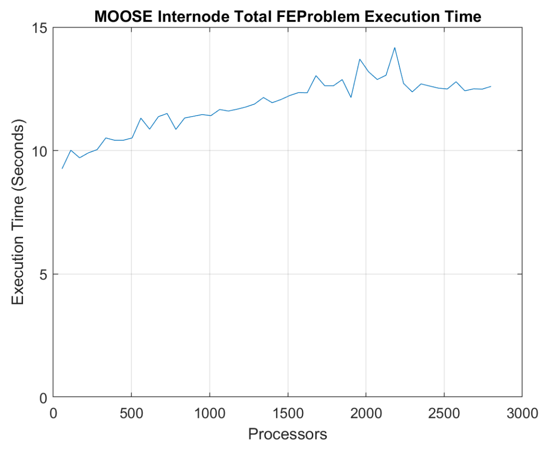

Figure 63: Problem Size 90^3 Weak Scaling

In terms of the solver, it is able to handle nonlinear and linear problems equivalently when examining the difference between linear and nonlinear scaling efficiency. Figure 64 presents the weak efficiency for internode scaling.

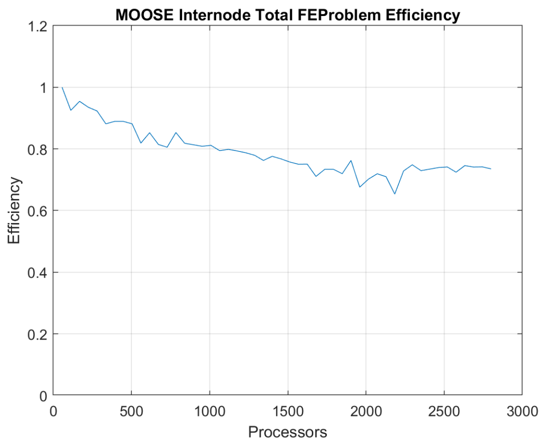

Figure 64: Problem Size 90^3 Weak Scaling Efficiency

The number of linear and nonlinear iterations is higher for the nonlinear diffusion equation because of the increased dependencies between the temperature gradients and temperature dependent thermal conductivity. This coupling effect requires the solver to perform more approximations to find the particular solution.

7.2 Meshing and Partitioning

Meshes allows the creation of computable geometries for many multi-physics problems. MOOSE has the ability to create inbuilt meshes for many multi-physics problem and use partitions to spilt up the mesh for parallel computing. Basic MOOSE mesh generation objects like GeneratedMeshGenerator and DistributedRectilinearMeshGenerator were used in this study. MOOSE's default mesh object is GeneratedMeshGenerator, where basic shapes like a sphere, cylinder, or cube can be constructed in the input file. GeneratedMeshGenerator was used at the beginning when experimenting with the supercomputer architecture, however memory bound issues prevented proper performance scaling. GeneratedMeshGenerator mesh evaluates the whole mesh on each processor which leads to memory issues once the mesh overfills the total allocated memory on the processor. [1] GeneratedMeshGenerator utilizes METIS to split the mesh, which is extremely fast but works poorly when being scaled. DistributedRectilinearMeshGenerator partially counteracts the memory issue by reading the entire mesh on each processor and deleting the parts that are not assigned to the processor. Each assigned mesh part is then sent to FEProblem solve to find the approximated solution to the partial differential equation of its corresponding mesh domain. DistributedMeshGenerator was the primary mesh object used in this study and allowed mesh sizes reaching 40,000,000 elements to be evaluated. At the beginning of each performance benchmark, the distributed mesh was equally cut by the number of processors in parallel performing the MPI task. ParMETIS is the default partitioner used in DistributedMeshGenerator and many other MOOSE mesh objects. [1] ParMETIS is an MPI-based library that is used to partition meshes in parallel and works well with large meshes as seen in this study. GridPartitioner is used if manual partitions are needed within the mesh. However, most cases prove GridPartitioner to be detrimental to the performance, since ParMETIS is optimized to give the best partitioned mesh. After many performance benchmarks, a 200^3 partitioned cube resulted in the best strong scaling results. For the weak scaling case, a starting problem size of a 90^3 partitioned cube performed the best in weak scaling performance benchmarks without memory issues.

8 Conclusions

The MPI benchmarks was used as the base comparison in this study because the scaling efficiency was practically one for all the scaling performance tests. In relation to the MPI benchmarks, MOOSE did not perform to MPI's ideal scaling performance. However, MOOSE was able to scale in relatively consistent manner in three out of four main performance indicators used each test set. The two major drawbacks in scaling performance within MOOSE were the inability to create a proper parallel mesh and MOOSE's communication class not scaling when collecting all the data vectors together. It was observed that putting more work per processor did indeed increase the total efficiency of the scaling problem, however the memory allocation problems for MOOSE's internal meshes limit the potential for truly large meshes. Across all the performance tests, Child FEProblem scaled the best and Self FEProblem scaled the worst. In relation between intranode and internode scaling, intranode scaled better in strong scaling while internode and intranode scaled about the same in weak scaling. There was no difference in efficiency between the linear and nonlinear diffusion problem except that the solver execution time doubled in some cases. Each diffusion problem (linear or nonlinear) was able to be solved in reasonable amount of time in MOOSE however non-linearity did not increase the efficiency as hoped. ParMETIS combined with DistributedMeshGenerator mesh object was the optimal configuration for loading and splitting MOOSE's meshes. In most cases, a single node utilizing its 56 cores were sufficient enough to solve the 200^3 element diffusion problem with the help of ParMETIS and a distributed mesh. The most substantial problem with the parallel scaling performance of MOOSE is its serial structured internal mesh objects that cause major slowdowns in scaling performance as indicted in the performance figures.

9 Future Work

The ability for a computer to perform a large computational task reliably and effectively has been the task of many high performance software. Many high performance libraries need scaling benchmarks to evaluate their performance on a large computer cluster like Frontera. Performing strong and weak scaling on different high performance software is a possible extension of this study. For more improvements within MOOSE, pregenerated meshes via a third party software is a potential solution to MOOSE's poor scaling internal meshes. With a pregenerated mesh, complex mesh geometries can be evaluated against different types of mesh generation algorithms. Each mesh algorithm has the possibility to improve the scaling performance of MOOSE and other high performance software requiring mesh geometries.

References

[1] G. Karypis, "Karypis lab," 2020. Last accessed 8 September 2021.