Rexsss Performance Analysis: Domain Decomposition Algorithm Implementations for Resilient Numerical Partial Differential Equation Solvers

K. M. Dahlgren[*], F. Rizzi[**], K. V. Morris[**], and B. J. Debusschere[**]
[*]*California State University, Stanislaus, One University Circle, Turlock, CA 95382, U.S.A.*
[**]*Sandia National Laboratories, 7011 East Avenue, Livermore, CA 94550, U.S.A.*

The future of extreme-scale computing is expected to magnify the influence of soft faults as a source of inaccuracy or failure in solutions obtained from distributed parallel computations. The development of resilient computational tools represents an essential recourse for understanding the best methods for absorbing the impacts of soft faults without sacrificing solution accuracy. The Rexsss (Resilient Extreme Scale Scientific Simulations) project pursues the development of fault resilient algorithms for solving partial differential equations (PDEs) on distributed systems. Performance analyses of current algorithm implementations assist in the identification of runtime inefficiencies.

I. Introduction

The Rexsss project represents a research and development effort dedicated to creating numerical PDE solvers implemented for parallel execution on distributed systems and designed to incorporate fault resiliency as an essential algorithmic trait. The Rexsss solver algorithms use generalized versions of the additive Schwarz domain decomposition approach for generating numerical PDE solutions. The additive Schwarz algorithm divides the PDE solution space into a number of subdomains and consolidates the results of a large number of sample PDE solutions generated within each subdomain per iteration until the numerical solve converges to the target PDE [1]. The current paper discusses the analysis of the scalability of various implementations of core sections of the Rexsss algorithms. The specifics of the algorithms that lead to resilience are discussed elsewhere [2]. Two of the major algorithms currently implemented within the Rexsss code base are the Distributed 1 Core Per Subdomain (D1CPS) and Task Manager (TM) approaches.

The D1CPS implementation produces numerical solutions for one-dimensional PDEs using a specified number of processors running in parallel. Computations for all solutions over a given subdomain occur within a single core. Accordingly, the number of processors utilized for the calculation of a PDE solution equals the number of subdomains partitioning the solution space.

In the TM implementation, a set of one or more server nodes divides the numerical PDE solve into a series of individual computations and communications and distributes the tasks among a collection of one or more client clusters such that each cluster contains one or more devoted cores. Each server core is responsible for

determining and distributing PDE solve tasks to a different subset of the client clusters. Additionally, each server is responsible for collecting PDE solutions over a different subset of subdomains. Updating communications occur between client clusters and server nodes and between server nodes. The server(s) also represent repositories for solution data obtained from client calculations. As a result, the algorithm inherently limits the impact of hard faults occurring on one or more of the client cores to the loss of data from single sample solutions or communications. The approach renders the destructive power of hard faults upon solution accuracy practically negligible. The parallel processing capabilities of the D1CPS and TM tools rely on the MPICH implementation of the MPI (Message Passing Interface) standard.

The performance analyses of the D1CPS and TM solvers explore the weak scaling behaviors of the code implementations. Valid weak scaling comparisons require test cases ensure the amount of computational work performed by each processor is equivalent for each processor involved in all of the tests, where the metric for computational work is real execution time in seconds [3]. Ideally, increasing the numbers of processors used in each test case equates to a linear increase in the total amount of work involved in solving each of the test problems.

The performance analyses calculate the weak scaling efficiencies of different subsections of the solver implementations using manual instrumentation statements gathering execution time data per isolated code section. Resulting scaling comparisons for the D1CPS implementation suggest the execution times devoted to sampling and updating processes do not scale well on the current hardware system as the number of processors increases. Data for the weak scaling analysis of the TM implementation exhibit similar efficiency trends.

II. Methods

The performance analyses focused upon test cases designed to provide insight into the weak scaling behaviors of the D1CPS and TM implementations of the Rexsss tools. Weak scaling represents a measure of the ability of a software tool to distribute computational work among a collection of processors consistently, regardless of the number of processors in the collection [3]. Calculations of weak scaling efficiency data result from the formula: [t1/ tN] * 100%, where t1 represents the execution time required for a test case containing a reference collection of processors and tN represents the execution time of a larger test case containing N times the number of reference collections of processors. Ideally, the addition of processors to a system only results in an increase of computational work equal to or less than the proportional amount of work expected of a reference collection of processors within the system. Software exhibiting good weak scaling manifests approximately similar percent efficiencies for test cases incorporating larger and larger collections of cores. Deviations from the ideal weak scaling efficiency trend indicate the presence of confounding factors relating increases in numbers of processing units to either negative or positive effects on tool execution within the context of individual processors.

All test cases pursued by the weak scaling performance analysis ran on an in-house four-node development cluster running Linux and designed such that each node contained 32 cores. For the D1CPS tests, cores reserved for the execution of each case were equally distributed among each of the four nodes. For example, when running a test using 8 cores, each of the four system nodes reserved 2 cores for the execution of the test. For the TM tests, one node contained all the server cores and the remaining 3 nodes shared a uniform distribution of all the client cores. For example, in each of the 8-core TM tests, the 2 server cores lived in one node and the 3 other nodes each contained 2 client cores.

Though a number of pre-processing functions and iteration post-processing directives contribute to the overall execution times of the tools, the associated performance costs are negligible and do not constitute crippling performance bottlenecks. Instead, processes related directly to the PDE solve and to inter-core communications constitute the most profitable areas for performance optimizations. Accordingly, the weak scaling analysis centralized around efficiency trends associated with the sampling and updating processes of the solver implementations. The sampling process encompassed all CPU time devoted exclusively to PDE solution calculations. In contrast, the updating process encompassed all CPU time devoted exclusively to the communication of subdomain iteration solutions to the processors of neighboring subdomains. Manual instrumentation statements written in C++, the native language of the Rexsss code base, isolated the sampling and updating processes and captured the associated real execution times per subdomain. The instrumentations output the average, minimum, and maximum real execution times for the total amount of time spent sampling and updating per test case for each process. The sampling and updating data per iteration result from dividing the total execution times for the processes by the number of iterations required by the test case to achieve convergence.

A. Solution Grid Parameters

All test cases solved the ODE described by $d^2y/dx^2 = 0$, where $y(xL) = 1$ and $y(xR) = yR = xR+1$. The boundary conditions at the x-coordinates xL and xR varied depending upon the test case parameters to ensure the solution grids possessed comparable characteristics as discussed further below. For all test cases, the boundary conditions were constructed such that the analytical solution was $y=x+1$. The number of points per subdomain for all tests was 517 with an overlap of 32 cells per subdomain. Each iteration of the tests required subdomains calculate exactly 30 inner samples of the ODE solution at different subdomain boundary conditions. The grid spacing for all tests was fixed at 0.0005. All tests ran until convergence measured in terms of a root-mean-square error tolerance with the true solution of the PDE obtained via analytical derivation methods.

For the performance analysis, work was defined in terms of the real execution time devoted to sampling and updating processes per subdomain. Configuration parameter restrictions ensured all cores performed the same amount of work across all tests by requiring each subdomain in each test case possess the characteristics of:

(a) Solving over a section of the same PDE.
(b) Containing the same number of points per subdomain.
(c) Containing the same number of overlapping grid cells.
(d) Enforcing the same length of grid spacing between all constituent points.

The absolute solution interval for the PDE must vary in length across test solution grids to ensure the grid spacing, number of points per subdomain, and overlap are constant across all test cases. The test configurations accommodated the requirement by fixing the left boundary point and extending the right boundary point slightly as the number of cores increased. Additionally, ensuring the different performance test cases solve at least a subsection of the same solution grid imposes a general consistency upon the complexity of the test case problems. As a result, the constraints ensured each subdomain experienced the same number of sampling calculations and the same problem complexities per test case.

B. Test Cases: Distributed 1 Core Per Subdomain

Test cases for the D1CPS implementation ran using 2, 4, 8, 16, 32, 64, and 128 cores. Since the D1CPS algorithm implementation relegates work performed within individual subdomains to work performed within individual cores, the consistent configuration parameters of the different test solution grids ensure all the processors involved in each test case engaged in approximately the same amount of sampling work, regardless of the number of cores involved in a particular test.

The D1CPS algorithm implementation requires the number of updating processes used to solve a PDE over a set of subdomains to be proportional to the number of subdomains partitioning the solution grid. The partitioning scheme divides the solution space into a set of inner subdomains flanked by two boundary subdomains. The inner subdomains communicate sample solution data to exactly two processors representing the left-hand and right-hand subdomains. The boundary subdomains communicate sample solutions to exactly one neighboring subdomain processor. Accordingly, the number of updating communications experienced by each individual boundary subdomain remains constant per test case. Likewise, the number of updating communications experienced by any one inner core remains constant per test case. As a result, the individual boundary cores involved in each test experience approximately the same amount of updating work, regardless of the number of cores involved in a particular test. Similarly, the individual inner cores involved in each test experience approximately the same amount of updating work, regardless of the number of cores involved in a particular test.

The proportionality of the test case solution grid conditions proffer the 2-core test as a valid base case for all weak scaling calculations. Since all the larger test cases utilize a number of cores equal to a power of 2, the 2-core case represents a reference collection of processors for the for the work involved in completing the PDE solves pursued by the larger test cases.

C. Test Cases: Task Manager

Test cases for the TM implementation ran with 8, 16, and 32 cores and client sizes of 1 and 3. Tests possessing a client size of 1 indicate constituent client clusters contain only 1 devoted core. On the other hand, tests possessing a client size of 3 indicate constituent client clusters contain the computing resources of 3 devoted cores. Table I in Appendix A lists the six different TM test case configurations of servers and clients.

Since the TM algorithm implementation utilizes the computing power of a group of one or more server processors to divide PDE solution work into a list of tasks distributed among a collection of one or more client clusters, the uniformity of the task determination and distribution process ensures all client cluster cores involved in each test case engage in approximately the same amount of sampling work, regardless of the number of cores involved in a particular test. Similarly, since the updating processes used to solve a PDE over a set of subdomains are distributed uniformly among the client processors, all client clusters involved in each test experience approximately the same amount of updating work.

Because the server processors do not perform calculations, the servers engage in work entirely different from the computational loads handled by client processors. Accordingly, test cases supporting valid weak scaling comparisons for TM test cases expand upon a reference collection of processors encompassing both server and client cores. Additionally, test cases involving different client sizes are inherently incompatible due to the drastic differences in the total number of cores sharing the computational work. Accordingly, TM weak scaling analyses compare the efficiencies of test cases sharing the same client size as the number of cores increases. The chosen base cases for the TM analyses with the different client sizes are the two 8-core tests in Table I. Since all the successively larger test cases utilize a number of server and client cores equal to a multiple of 2 greater than the numbers of server and client cores in the smaller tests, the 8-core case represents a basic unit of work for the PDE solves pursued by the larger test cases.

III. Results

The results of the performance analyses encompass weak scaling efficiency plots for the Distributed 1 Core Per Subdomain and the Task Manager algorithm implementations. Appendix B contains graphs of the weak scaling data for all performance tests.

A. Weak Scaling: Distributed 1 Core Per Subdomain

The results of the D1CPS performance analysis encompass data regarding the weak scaling efficiencies of the real sampling and updating times per iteration.

1. D1CPS: Sampling Time Per Iteration

Figure 1 in Appendix B shows the weak scaling efficiencies of the sampling functionality per iteration of the D1CPS algorithm implementation in relation to increasing numbers of cores. The dotted blue line represents the weak scaling data for the minimum sampling execution time experienced during an iteration by a

single core in the test cases. In contrast, the dashed blue line represents the weak scaling efficiencies of the maximum sampling execution time experienced during an iteration by a single core in a particular test case. The solid blue line represents the weak scaling efficiencies of the average sampling time per iteration calculated from the average time spent by each core in the sampling process.

The minimum, maximum, and average weak scaling data exhibit a downward trend with increasing numbers of processors. Accordingly, the deviation from ideally equivalent weak scaling efficiencies suggests the presence of factors causing the increasing numbers of processors to negatively impact the execution time devoted to solution sampling across all subdomains per iteration. Since the complexities of test case problems are equivalent, the increasing numbers of sampling processes required to support higher numbers of subdomains do not represent a factor decreasing the overall sampling efficiency. Instead, the results suggest the source of the negative impact upon sampling execution efficiency stems from an increased amount of time devoted to calculating individual sampling solutions. The increases in individual sample solution calculation times culminate in higher total sampling execution times per iteration and, consequently, decrease weak scaling efficiencies for test cases using larger collections of processors.

2. D1CPS: Updating Time Per Iteration

Figure 2 graphs the weak scaling efficiencies of the updating functionality of the D1CPS algorithm implementation in relation to increasing numbers of processors used to solve the test cases. The dotted red line represents the weak scaling data for the minimum updating execution time experienced during an iteration by a single core in a test case. In contrast, the dashed red line represents the weak scaling efficiencies of the maximum updating execution time experienced during an iteration by a single core in a particular test case. The solid red line represents the weak scaling efficiencies of the average updating time per iteration calculated from the average time spent by each core pursuing updating communications.

The updating time per iteration efficiencies reflect the trends revealed by the weak scaling analysis of sampling execution times per iteration. The minimum, maximum, and average weak scaling data exhibit a rapid downward descent with increasing numbers of processors. The deviation from ideally equivalent weak scaling efficiencies indicates the presence of factors causing the increasing numbers of processors to negatively impact the execution times devoted to updating communications per iteration. Since the complexities of all the test case problems are equivalent, the increasing numbers of updating processes required to support higher numbers of subdomains do not represent a factor decreasing the overall efficiencies of updating processes per iteration. Consequently, the source of the negative impact upon updating costs per iteration is an increased amount of time devoted to performing individual update communications. The increases in individual update communication costs result in higher total updating execution times per iteration and, consequently, decrease weak scaling efficiencies for test cases utilizing larger collections of processors.

B. Weak Scaling: Task Manager

The results of the TM performance analyses encompass data regarding the weak scaling efficiencies of the real sampling time per iteration and the weak scaling efficiencies of the real updating time per iteration for tests cases sharing a client size of 1 and for test cases sharing a client size of 3.

1. TM: Sampling Time Per Iteration

Figure 3 shows the weak scaling efficiencies of the sampling functionality per iteration of the TM algorithm implementation in relation to increasing numbers of processors. The blue line represents the weak scaling data for the average sampling execution time per iteration for the test cases using a client size of 1. The red line represents the weak scaling efficiencies of the average sampling time per iteration for test cases using a client size of 3.

Both the client size 1 and the client size 3 weak scaling data exhibit a downward trend with increasing numbers of processors. Given the consistent complexities of problem definitions across test cases, the results indicate the source of the decreasing sampling efficiencies centers around increasing amounts of time required to calculate individual sampling solutions. The increased execution times result in higher total sampling costs per iteration as the number of processors increases.

2. TM: Updating Time Per Iteration

Figure 4 displays the weak scaling efficiencies of the updating processes for the TM algorithm implementation in relation to increasing numbers of processors. The blue line represents the weak scaling data for the average total updating communication time per iteration for the test cases using a client size of 1. The red line represents the weak scaling efficiencies of the average total updating communication time per iteration for test cases using a client size of 3.

The efficiencies of updating costs per iteration reflect the trends revealed by the weak scaling analysis of sampling costs. The average weak scaling data exhibit a rapid downward trend as the number of processors increases. Since all the server cores are located within a single node, inter-node communication hardware exerts physical constraints upon updating processes conducted between servers and client clusters. The increased frequency of updating communications required to support higher numbers of subdomains increases the number of inter-node communications. Accordingly, increasing the number of signals traveling down the same fixed-bandwidth communication channel renders hardware issues a potential source of the observed decreased updating efficiencies.

IV. Discussion

The weak scaling analysis results for sampling and updating processes occurring within both the D1CPS and TM implementations exhibit downward efficiency trends with increasing numbers of cores. For the D1CPS tool, the solid comparability of the test case problems suggest the inefficiencies originate as a consequence of increasing the amount of time necessary to complete individual

sampling and updating functions. For the TM approach, the problem comparability indicates sampling inefficiencies stem from increased time costs associated with completing individual sampling processes. Accordingly, decreased sampling efficiencies imply that the observed execution cost increases result from issues relating to the balancing of concurrent computing demands from basic node maintenance and runs of the D1CPS and TM tools. On the other hand, decreased efficiencies associated with updating processes point to confounding factors stemming from hardware issues and/or MPICH communication settings on the current hardware. As mentioned before, all tests in this report used an in-house four-node/128-core development cluster. Preliminary tests on Edison at the DOE NERSC Leadership Computing facility show weak scaling efficiencies much closer to 100% for test cases simulating the behaviors of sampling and updating processes for the distributed 1 core per subdomain implementation. While the results are too preliminary to draw any solid conclusions, the difference in scalability between the two platforms suggests that the specifics of the communication hardware and protocols on the four-node development cluster will need to be examined. These comparisons are the subject of ongoing work.

V. Conclusion

The performance analysis investigated the weak scaling efficiencies of the D1CPS and TM code implementations in relation to increasing numbers of processors. The study required manually instrumenting the sampling and updating functionalities of the tools to capture real execution times gathered from test runs as the solvers converged toward the true solution of the test PDE. Results of sampling and updating efficiencies per iteration suggest poor weak scaling on the current four-node development cluster.

Acknowledgements

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internship (SULI) program. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award Number 13-016717. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

1. M. Garbey, "Acceleration of the Additive Schwarz Method for Elliptic Problems," SIAM J. Sci. Comput., 26(6), 1871-1893 (2005).
2. K. Sargsyan, F. Rizzi, C. Safta, K. Morris, H. Najm, O. Knio, B. Debusschere, "Fault resilient probabilistic preconditioner method for one-dimensional PDEs", in preparation (2014).
3. "Measuring Parallel Scaling Performance," https://www.sharcnet.ca/help/index.php/Measuring_Parallel_Scaling_Performance.

APPENDIX A: TM TEST CONFIGURATION TABLE

TABLE I: Parameter configurations for the 6 Task Manager test cases.

# of Subdomains	# of Cores	# of Servers	Client Size	Total # of client core sets
8	8	2	1	6
8	8	2	3	2
16	16	4	1	12
16	16	4	3	4
32	32	8	1	24
32	32	8	3	8

APPENDIX B: WEAK SCALING PERFORMANCE GRAPHS

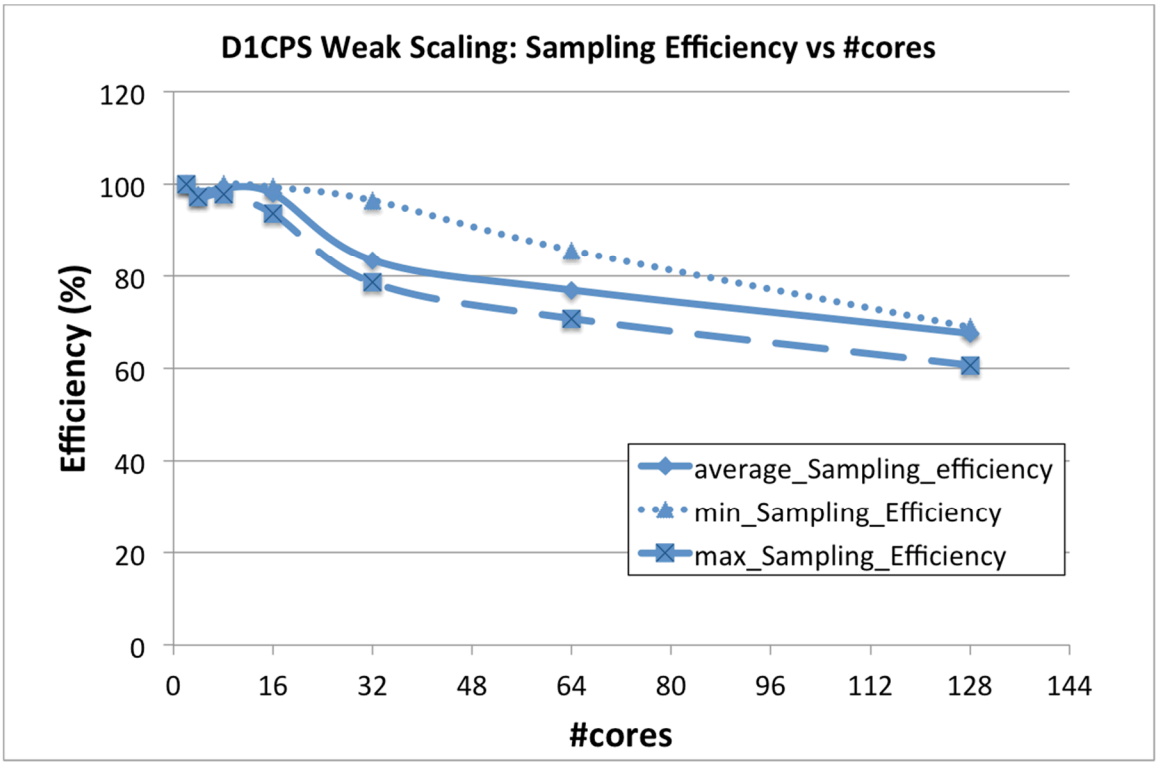

FIG. 1. Minimum, average, and maximum weak scaling efficiencies for the D1CPS sampling process graphed in relation to increasing numbers of processors.

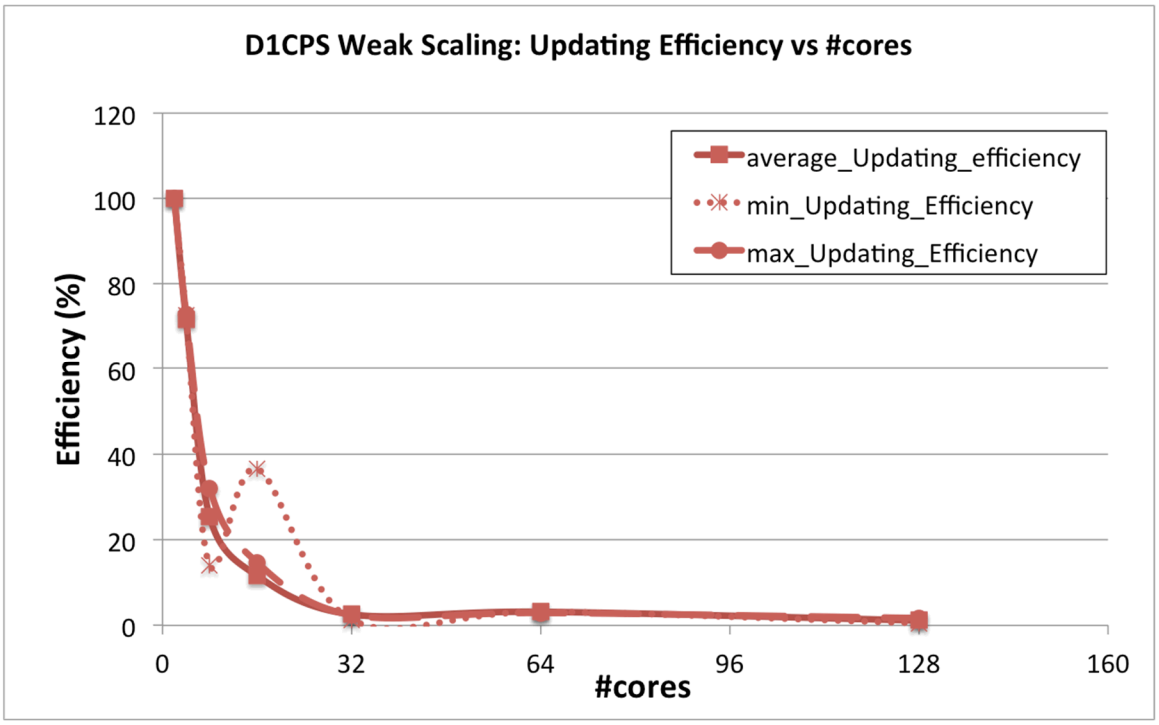

FIG. 2. Minimum, average, and maximum weak scaling efficiencies for the D1CPS updating process per iteration graphed in relation to increasing numbers of cores.

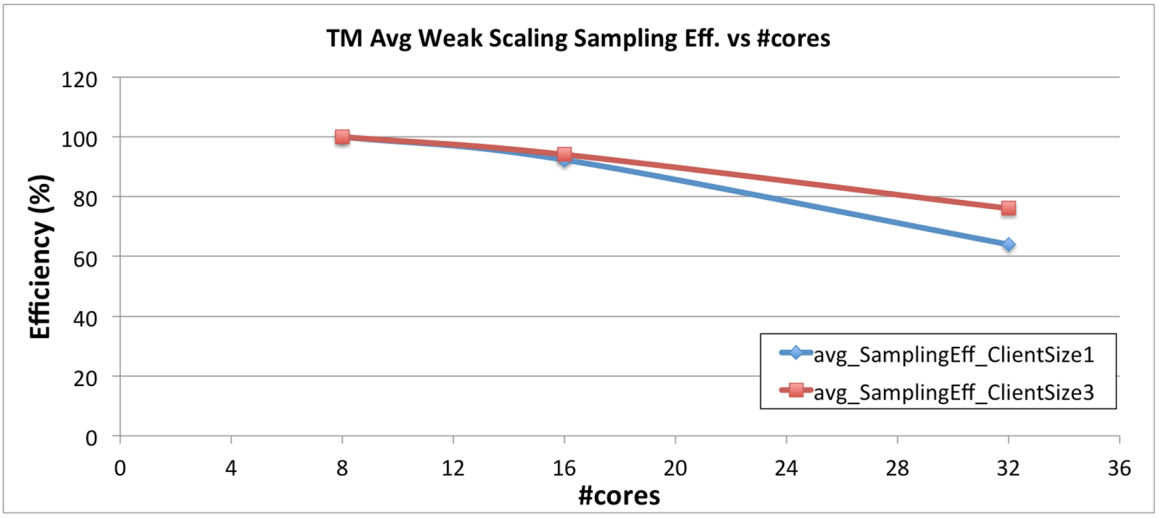

FIG. 3. Average weak scaling efficiencies for the TM sampling process per client size graphed in relation to increasing numbers of cores

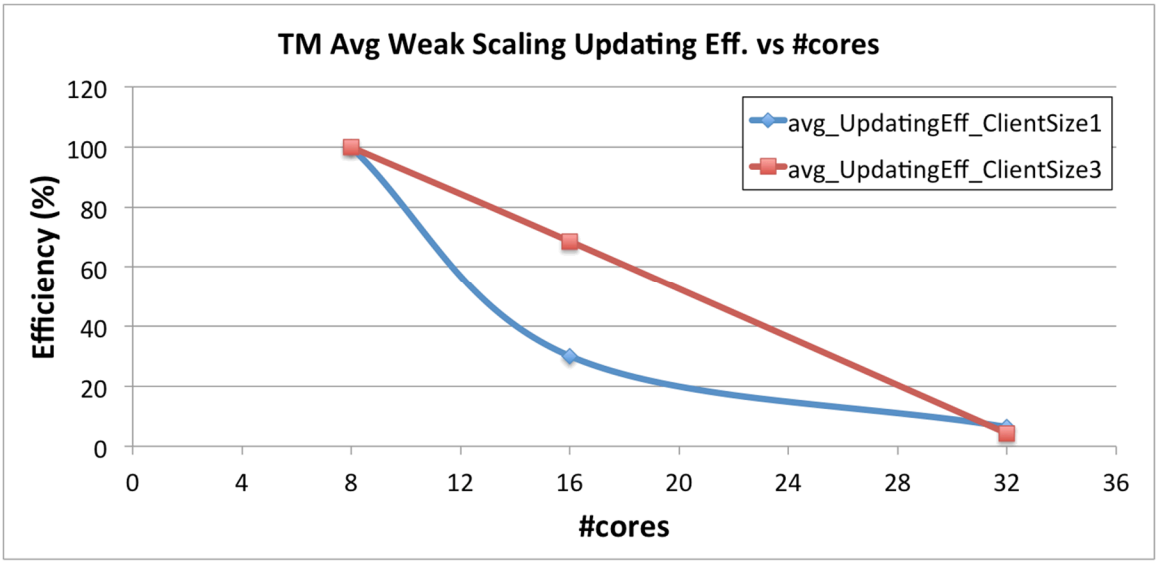

FIG. 4. Average weak scaling efficiencies for the TM updating process per iteration graphed in relation to increasing numbers of cores.