Journal of Supercomputing manuscript No.
(will be inserted by the editor)

A Case Study in Multi-core Parallelism for the Reliability Evaluation of Composite Power Systems

Robert C. Green II · Vishakha Agrawal

Received: date / Accepted: date

Abstract The probabilistic evaluation of composite power system reliability is an important but computationally intense task that requires the sampling/searching of a large search space. While multiple methods have been used for performing these computations, a remaining area of research is the impact that modern platforms for parallel computation may have on this computation. Studies have been performed in the past, but they have been primarily limited to cluster-based computing. In addition, the most recent works in this area have used outdated technology or been evaluated using smaller test systems. In the modern era, a wide variety of platforms are available for achieving parallelism in computation including options like multi-core processors, clusters, and accelerators. Each of these platforms provides unique opportunities for accelerating computation and exploiting scalability.

In order to fill this gap in the research, this study implements and evaluates two methods of parallel computation – Batch Parallelism and Pipeline Parallelism – using a multi-core architecture in a cloud computing environment on Amazon Web Services (AWS) using up to 36 virtual compute cores. Further, the methodologies are contrasted and compared in terms of computation time, speedup, efficiency, and scalability. Results are collected using IEEE Reliability Test Systems and speedups upwards of 5x are demonstrated across multiple test systems.

Keywords Power System Reliability · OpenMp · Pipeline parallelism · Monte Carlo Simulations · Multicore · Batch Parallelism

This is a pre-print of an article published in The Journal of Supercompuing. The final authenticated version is available online at: https://doi.org/10.1007/s11227-017-2073-z

R. Green and V. Agrawal
Dept. of Computer Science
Bowling Green State University
Bowling Green, OH 43402
E-mail: {greenr,vagrawa}@bgsu.edu

1 Introduction

The ability to quickly estimate the reliability of power system is an important aspect of the modern power grid, that often plays a role in choices regarding investments, power generation, operating costs, etc. [21] As the power system grows, operating constraints increase, making the process of reliability evaluation more computationally intensive, particularly as this problem suffers from curse of dimensionality [11]. As an example, the test systems currently used for research have state space sizes ranging from 2^{32} to 2^{99} while real-world systems are many times as large. To combat these issues, many paths have been followed ranging from the use of non-sequential Monte Carlo Simulation (MCS) to the inclusion of Population-based Metaheuristics (PBMs) [11] and high performance computing (HPC) [8,9]. For those works focusing squarely on the use and application of HPC, the main focus has been on the use of cluster computing using various master-slave configurations [5,6,4,13,17] or grid computing [2]. Throughout these works, there is a lack of exploration of modern, or multi-core centric, architectures or unique algorithmic implementations (almost all of these works focus on the well-known master-slave paradigm) [8, 9]. In order to address these shortcomings in the literature, this study explores the impacts and nuances of pipeline and batch parallel implementations of non-sequential MCS using multi-core architectures in a cloud-based environment for evaluating the reliability of the composite power system in a probabilistic manner. As such, this work is substantially different from previous works in three main ways: 1) A multi-core architectures, as opposed to cluster and grid computing, are the focus, 2) A batch and pipeline parallel methodology is implemented and evaluated, and 3) All benchmarks are run using cloud-based resources as opposed to locally built clusters or compute resources.

The remainder of this paper is structured as follows: Section 3 discusses the parallelization methodology and its implementation; Section 4 describes the results obtained from pipeline parallelism and batch parallelism; Section 4.3 discusses the results; and Section 5 summarizes the paper while presenting conclusions and future work.

2 Background

Background information related to this study include the fundamentals of MCS as applied to the probabilistic reliability evaluation of composite power systems as well as a review of those methods that have previously leveraged parallel computation when investigating and extending this algorithm.

2.1 Reliability Analysis of Composite Power Systems

MCS, and particularly non-sequential MCS, is often times the algorithm of choice for evaluating composite power system reliability. The algorithm itself

relies on three main steps – Sampling, classification, and determining convergence. In a composite power system, the state space is made up of all sources of generation and transmission, with the state of each being represented as a "1" for functioning and a "0" when failing. The combination of individual device states compose a single state, while all states that may exist in a given system compose the entire state space.

During the sampling stage, states are sampled using pseudo-random numbers and the forced outage rate, or (FOR_i), of each component, i, which is common probability measure referring to how often a single device may fail. Sampling a state, S, for each device/dimension, i, entails generating a random number, r_i that is compared to the FOR_i according to (1). The concatenation of these results leads to the complete binary representation of a single state.

$$S_i = \begin{cases} 0 & r_i \leq FOR_i \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

Once sampled, a DC Optimal Power Flow (OPF) may be used to classify a state as a failed or a functioning state [18,22], where the goal of the DC-OPF is to minimize load curtailment.

After classification, the reliability of the system can be evaluated using a common measure known as $LOLP$, or loss-of-load probability. To accomplish this, $LOLP$ and $V(LOLP)$ (variance of $LOLP$) during each iteration and, once a preset limit is reached, the algorithm terminates. This is shown in (2) - (5), where K is the number of total states sampled.

$$LOLP = \frac{1}{K}\sum_{x=1}^{K} S_x \tag{2}$$

$$V(LOLP) = \frac{1}{K}(LOLP - LOLP^2) \tag{3}$$

$$\sigma = \frac{\sqrt{V(LOLP)}}{LOLP} \tag{4}$$

$$S_x = \begin{cases} 1 & \text{loss-of-load} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

2.2 Parallel Computing in Composite Power System Reliability Evaluation

The majority of the work relating HPC methodologies to the probabilistic evaluation of composite power system reliability using non-sequential MCS is encompassed in seven specific works [13,3,5,6,4,2,10], though one additional work may be considered if including parallel PBMs and their applications in this area [23,10].

The work in [13] is the fundamental basis of study in this area. In this work, three types of master-slave topology are implemented and evaluated using MPI

on a nCUBE processors. Insights are discussed regarding implications of the model and the impact of random number generation.

This work is continued by Borges et. al [3,5,6,4], though only two of the works focus on non-sequential MCS ([3,6]). These works evaluate course-grained methods using asymmetric parallelism. The developed algorithm is applied to five test systems (RTS79, NBS, NNE, SUL, and SE) using a IBM RS/6000 SP with 4 POWER2 processors. Speedups ranging from $1x$–$10x$ are achieved.

Considering modern parallel computing paradigms, the work in [2] develops a grid focused on the reliability and security evaluation of power systems. In this study results obtained from a prototype system are reported, though are centered on small signal stability analysis and not on reliability evaluation

While note purely MCS, the progress in [23] focuses on parallel implementations of a GA for calculating reliability indices. The multi-deme method is used. The algorithms is applied to the RTS79 test system using 3, 7, and 15 processors. Resultant speedups range from 2 to 12.

The work continued in [10] extends the Intelligent State Space Pruning (ISSP) algorithm to a parallel computing platform using MPI. The methodology leverages population-based metaheuristics, assigning different populations to different compute cores. The populations pass various members and other information, resulting in speedups from 1x-10x.

3 Proposed Methods

In this study, two methods – batch parallelism and pipeline parallelism – are proposed as described in the following two sections. Under both methods, thread level parallelism is used to divide the necessary computation into isolated tasks. As previously noted, these methods differ from the current state-of-the art as they are multi-core centric, leverage new methodologies that are different from those previously evaluated, and are evaluated using cloud-based, virtualized resources.

3.1 Batch Parallelism

Under this paradigm, the computational tasks present under MCS are divided on a per thread basis where N is the number of threads allocated. Each thread is also assigned a batch size, B, that informs each thread with regards to how many states it should Generate and then Classify. This method is graphically depicted in Fig. 1. After each thread Generates and Classifies all of the necessary states, all results are merged and the results are evaluated for convergence. If not converged, the algorithm launches another N threads that once again classify with the given value of B.

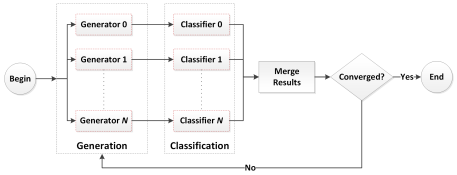

Fig. 1 Proposed process for batch parallel methodology.

3.2 Pipeline Parallelism

This paradigm uses task level parallelism in order to spawn a number of threads that continually perform all necessary tasks – Generation, Classification, and Computation – in a pipelined matter. The fundamental advantage in using this methodology is that those tasks that are more computationally intense are able to use a larger number of threads. For instance, Classification is typically computationally intense portion of the algorithm. This suggests that any given Generation thread should be able to produce or sample states more rapidly than any Classification thread can keep pace with. In order to reduce the computation time required, additional Classification threads may be added, allowing the sampled states to be processed more rapidly.

The number of threads performing each task are named as N_g for generation threads and N_c for classification threads. Computation threads require no definition as, due to the sequential nature of the process(result of convergence depends on previous iterations), the operation requires only a single thread. The entirety of this process is detailed in Fig. 2.

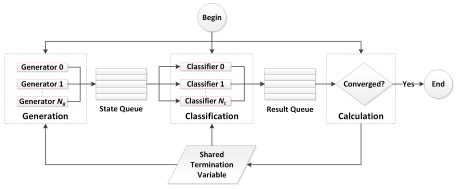

Fig. 2 Proposed process for pipeline parallel methodology.

3.3 Evaluation Methodology

For evaluation, all values are reported as average values over at least ten trials. In addition, various measures exist for the comparison of parallel algorithms, the most common two being speedup and efficiency. Speedup is a well known metric that is defined in (6) where T_1 is the execution time on a single processor, core, or node and T_m is the execution time on m processors, cores, or nodes. As is shown in [1] this is not a valid comparison for non-deterministic

algorithms such as MCS algorithms. In light of this, mean parallel execution time will be a better judge of algorithm improvement is shown in (7).

$$s = \frac{T_1}{T_m} \tag{6}$$

$$s_m = \frac{E[T_1]}{E[T_m]} \tag{7}$$

Another calculation often used to evaluate the benefit of parallel algorithms is the measure of efficiency [1]. This measure is defined in (8) and can be considered as the speedup per processor, a normalization of the speedup measure, or a measure of the effectiveness of resource usage and scalability. In this measure, an efficiency of one corresponds to a linear speedup while any value exceeding one is super-linear and any value less than one is sub-linear.

$$e_m = s_m/m \tag{8}$$

4 Results

For all results, evaluation occurred using a c4.8xlarge compute optimized EC2 instance on Amazon AWS which consists of 36 virtual CPUs, 60 GB of RAM, and 8 GB of solid state device (SSD) storage. The instance was initialized using Ubuntu Server 14.04 (64-bit). To achieve parallelism in both the methods, OpenMP [19] was used. OpenMP is an API which supports multi-platform shared-memory parallel programming in C/C++ and FORTRAN. All trials were run a total of 10 times and average values are reported in all cases. In tables and figures, N refers to the number of threads used in batch-level parallelism, B refers to batch size.In pipeline parallelism N_g refers to the number of generation threads, and N_c refers to the number of classification threads and only a single thread was ever used for performing the calculations involved in determining convergence. The software and all generated data used for this study was developed by the author and is available for free at Gitlab [1].

Simulations were completed using three standard test systems: RTS79 [14], MRTS [7,20] and RTS96 [12]. The IEEE Reliability test system, RTS79 contains 24 buses, 38 transmission lines, and 32 generating units with 10 buses connected to generators, and 38 transmission lines and 5 transformers. The annual peak load for the system is 2,850 MW and the total generating capacity is 3,405 MW. The MRTS (Modified Reliability Test System) is the modified version of RTS79.It is modified such that capacity of its generating units is twice of as that of RTS79 and annual peak load is 1.8 times of that of RTS79. As a result, annual peak load is 5,130 MW and total generating capacity is 6,810 MW. The RTS96 contains three interconnected areas, where each area is RTS79, connected via 5 tie lines. The system has 73 buses, 120 transmission

[1] https://gitlab.com/MCS-Power-System-Reliability/mcs-pruning

lines, and 96 generating units. Annual peak load of the system is 8,550 MW and total generating capacity is 10,215 MW.

Results from all trials were also submitted to a Wilcoxon rank-sum test as implemented by SciPy [15]. This test was used to determine the statistical significance of 1) Results in terms of computation time and 2) Resultant LOLP values. A value of $p \leq 0.05$ would suggest that the computation times were drawn from different distributions. In terms of LOLP, a $p > 0.05$ would suggest that LOLP values achieved were statistically similar. In any case, the p values referring to time will be referenced as p_{time} and the p value concerning LOLP will be referred to as p_{LOLP}.

4.1 Batch Parallelism

For the Batch Parallel method, the Wilcoxon rank-sum test indicated that 22 RTS79 experiments and 17 MRTS experiments produced computation times that were possibly not statistically significant, achieving a p_{time} value greater than 0.05. This was expected as some combinations of threads and batch size should produce computation times similar to that of the serial implementation.

Considering p_{LOLP}, 58 RTS79 experiments, 200 MRTS experiments, and 126 RTS96 experiments resulted in values suggesting that the p_{LOLP} achieved may vary from that achieved by the serial implementation. On manual inspection, it was easily seen that though the possibility for significant variation existed, the values did not differ substantially.

4.1.1 Computation Time

Results regarding computation time for the batch parallel method are shown in Figs. 3 - 5. Generally, these results are as expected. For the RTS79 and MRTS systems a very similar trend is shown where, initially, the increase in number of threads and batch size results in a decrease in computation time. As the number of threads and the batch size are increased, the combinations result in extended computation times. This is expected for systems such as this as they require so few samples to converge. For instance, the RTS79 requires roughly 18,000 samples to achieve convergence. Considering a batch size of 10,000 using 35 threads samples roughly 350,000 states – a sample size that is unnecessary and equivalent to roughly 20 times more samples than are needed. From Fig. 3 it is obvious that this situation results in the largest of computation times. This trend is similar for the MRTS and RTS96 systems, though the results are generally more exaggerated for the RTS96.

Optimal values occur for RTS79, MRTS, and RTS79 at thread–batch combinations of 12/1,500, 11/4,500, and 33/3,500 where computation times (in seconds) of 0.51 (vs. 2.25), 1.44 (vs. 5.83), and 8.24 (vs. 48.21) are achieved.

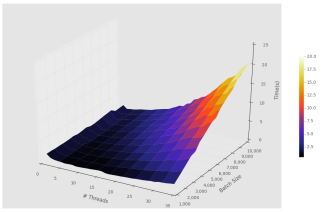

Fig. 3 Required computation time for various combinations of Number of Threads and Batch Size for RTS79 using Batch Parallelism.

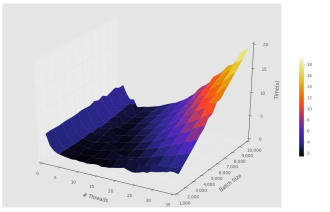

Fig. 4 Required computation time for various combinations of Number of Threads and Batch Size for MRTS using Batch Parallelism.

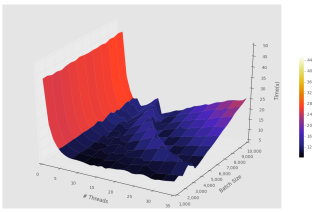

Fig. 5 Required computation time for various combinations of Number of Threads and Batch Size for RTS96 using Batch Parallelism.

4.1.2 Speedup & Efficiency

Results regarding speedup for the batch parallel method are shown in Figs. 6 - 8. The same general trend is shown for all three systems – an initial increase in speedup as the number of threads and batch sized are increased followed by various combinations of threads and batch sizes that produce mixed, yet near optimal, results, which are finally followed by a continual decrease in speedup. As expected, the optimal combination of threads and batch size is found in an area with a middle ground that does not simply maximize either parameter. This results in optimal speedup values of 4.36, 4.04, and 5.85 for the RTS79, MRTS, and RTS96, respectively, using batch sizes of 9,000, 4,500, and 3,500.

Efficiency for the batch parallel method is shown in Figs. 9 - 11. It is interesting to note that for all three systems a high level of efficiency (1.07, 0.99, and 1.12 for the RTS79, MRTS, and RTS96) is obtained using a single thread with a batch size of 9,000 or 2,000 for the RTS79, 1,000 for the MRTS, and 3,000 for the RTS96 (these values are shown in tabular format in Table 1). This suggests that the batch parallel methodology reaches its ideal performance in terms of utilization at a low thread count, though not in terms of time reduction. This is reasonable as the addition of either additional threads or the shift to larger batch sizes increases either the overhead required for parallel computation or the computation required of an individual thread. Again, looking at the figures, it can be noted that there is a general trend of a decrease in efficiency as 1) The number of threads is increased and 2) The batch size is enlarged for any given number of threads. This suggests that as both number of threads and/or batch size is increased, the utilization of resources by the batch parallel methodology slowly reduces, though this does not necessarily suggest that the speedup must decrease – it could only increase more slowly.

System	Threads	Batch Size	Time (s)	Speedup	Efficiency
RTS79	12	1,500	0.5162	4.3596	0.3633
RTS79	1	9,000	2.0874	1.078	1.078
MRTS	11	4,500	1.4416	4.0449	0.3677
MRTS	1	1,000	5.8383	0.9988	0.9988
RTS96	33	3,500	8.2404	5.8502	0.1772
RTS96	1	3,000	43.2123	1.1156	1.1156

Table 1 Statistics regarding the optimal combination of threads and batch size for achieving maximum Speedup and Efficiency using the batch parallel method.

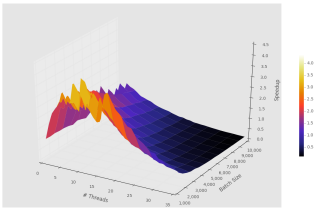

Fig. 6 Speedup achieved for various combinations of Number of Threads and Batch Size for RTS79 using Batch Parallelism.

4.2 Pipeline Parallelism

For the Pipeline Parallel method, the Wilcoxon rank-sum test indicated that seven RTS79 experiments, nine MRTS experiments, and 2 RTS96 experiments

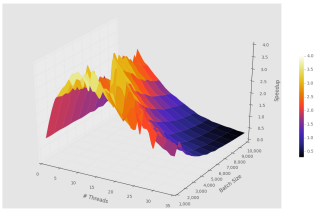

Fig. 7 Speedup achieved for various combinations of Number of Threads and Batch Size for MRTS using Batch Parallelism.

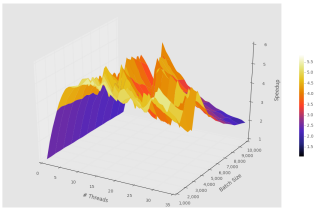

Fig. 8 Speedup achieved for various combinations of Number of Threads and Batch Size for RTS96 using Batch Parallelism.

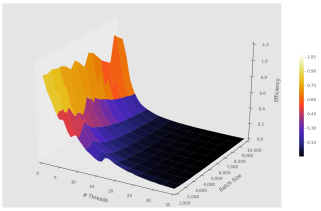

Fig. 9 Efficiency achieved for various combinations of Number of Threads and Batch Size for RTS79 using Batch Parallelism.

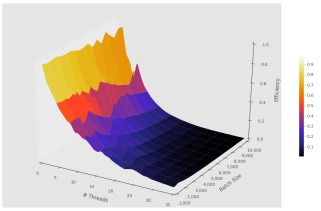

Fig. 10 Efficiency achieved for various combinations of Number of Threads and Batch Size for MRTS using Batch Parallelism.

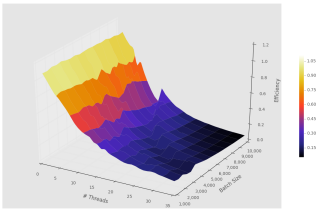

Fig. 11 Efficiency achieved for various combinations of Number of Threads and Batch Size for RTS96 using Batch Parallelism.

produced computation times that were possibly not statistically significant, achieving a p_{time} values greater than 0.05. This was expected as some combinations of generation and classification threads should produce computation times similar to that of the serial implementation.

Considering p_{LOLP}, four RTS79 experiments, 32 MRTS experiments, and 42 RTS96 experiments resulted in values suggesting that the p_{LOLP} achieved may vary from that achieved by the serial implementation. On manual inspection, it was easily seen that though the possibility for significant variation existed, the values did not differ substantially.

4.2.1 Computation Time

Results regarding computation time for the pipeline parallel method are shown in Figs. 12 - 14 where the results are as expected. The use of various combinations of generation and classification threads leads to reduced computation time as the number of each type of thread increases concurrently. Note that all methods graphically show a "dip" near their optimal values (using 1 generation thread for all methods). As is known, this clearly demonstrates that the portion of calculation requiring extended computation lies solely in the classification of states.

Optimal values for computation time occur for RTS79, MRTS, and RTS79 at Generation/Classification thread combinations of 1/15, 1/15, and 1/35 where computation times (in seconds) of 0.0.43 (vs. 2.25), 1.23 (vs. 5.83), and 8.68 (vs. 48.21) are achieved.

4.2.2 Speedup & Efficiency

Results regarding speedup and efficiency for the the pipeline parallel method are shown in Figs. 15 - 20. For the RTS79 and MRTS, a trend that clearly mimics that of computation time is shown, with a steep peak begin graphically evident at the optimal combination of generation and classification threads that then quickly decreases for other combinations, eventually leveling off near a speedup value of two. The RTS96, on the other hand, demonstrates a somewhat different trend, where combinations of generation and classification

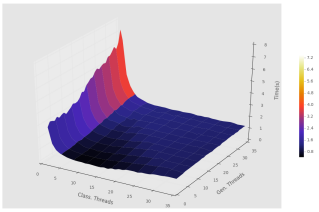

Fig. 12 Required computation time for various combinations of Generation and Computation threads for RTS79 using Pipeline parallelism.

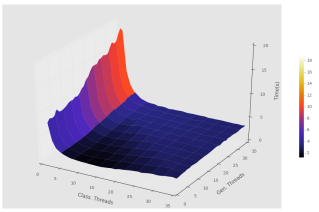

Fig. 13 Required computation time for various combinations of Generation and Computation threads for MRTS using Pipeline parallelism.

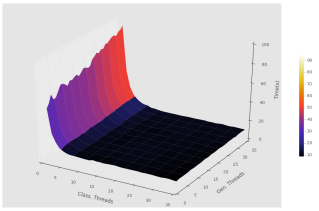

Fig. 14 Required computation time for various combinations of Generation and Computation threads for RTS96 using Pipeline parallelism.

threads result in a nearly constant speedup of roughly five, though the optimal speedup and computation time achieved are still obvious with one generation and 35 classification threads. This behavior is likely due to the large number of samples required by the RTS96 system to achieve convergence. This results in optimal speedup values of 5.22, 4.75, and 5.55 for the RTS79, MRTS, and RTS96, respectively.

In terms of efficiency, Figs. 18-20 show that for the RTS79 and MRTS systems, efficiency is maximized using generation/classification thread combinations of 1/4 and 1/5, resulting in efficiency values of 0.55 and 0.49. This is in opposition to the combinations of threads leading to optimal speedups that were listed previously. The RTS96 system shows similar results, yielding a maximal efficiency of 0.65 at a combination of 1/4 generation/classification

threads which is, again, in contrast to the combination leading to optimal time reduction and speedup as listed previously. The details of this data are also shown in Table 2. The overall trend for efficiency in all cases related to pipeline parallelism suggests that there is an increase in utilization by adding a reasonable number of threads for the classification task, though adding too many classification threads begins a decrease in efficiency. In other words, as more classifier threads are added, the benefit of each addition is reduced, eventually leading to a decline in resource utilization. In addition, the number of generation threads should be held at one.

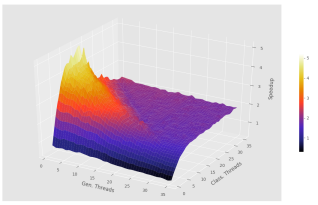

Fig. 15 Speedup achieved for various combinations of Generation and Computation threads for RTS79 using Pipeline parallelism.

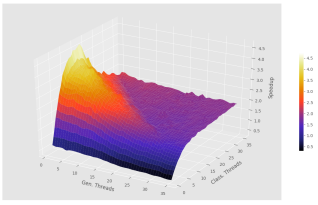

Fig. 16 Speedup achieved for various combinations of Generation and Computation threads for MRTS using Pipeline parallelism.

System	Gen. Threads	Class. Threads	Time (s)	Speedup	Efficiency
RTS79	1	4	0.81	2.77	0.55
RTS79	1	15	0.43	5.22	0.33
MRTS	1	15	1.23	4.75	0.30
MRTS	1	5	2.00	2.92	0.49
RTS96	1	35	8.68	5.55	0.15
RTS96	1	4	14.72	3.27	0.65

Table 2 Statistics regarding the optimal combination of threads and batch size for achieving maximum Speedup and Efficiency using the pipeline parallel method.

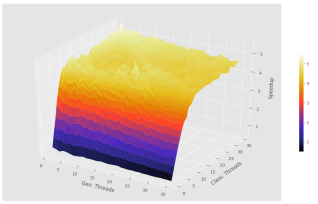

Fig. 17 Speedup achieved for various combinations of Generation and Computation threads for RTS96 using Pipeline parallelism.

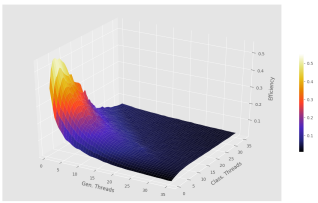

Fig. 18 Efficiency achieved for various combinations of Generation and Computation threads for RTS79 using Pipeline parallelism.

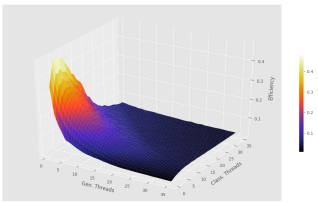

Fig. 19 Efficiency achieved for various combinations of Generation and Computation threads for MRTS using Pipeline parallelism.

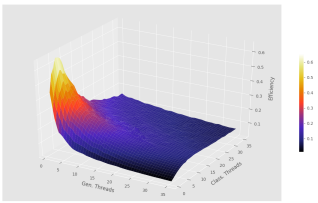

Fig. 20 Efficiency achieved for various combinations of Generation and Computation threads for RTS96 using Pipeline parallelism.

4.3 Discussion

The work presented in previous sections clearly demonstrates that the use of a modern, multi-core, cloud-based system of parallel computation results in speedups of roughly 4x-5x when considering the probabilistic evaluation of composite power system reliability. Yet, when considering both methodologies, two key questions should be asked. First, "What is the optimal configuration in order to achieve the most significant performance?", and, second, "How well will either of these algorithms scale?"

Optimal configuration concerns itself with the selection and organization of the number and types of threads used for processing. When considering the maximum speedups obtained via batch parallelism at 12, 11, and 33 threads using a batch size of 1,500, 4,500, and 3,500, respectively, it can safely be assumed that a key to achieving significant speedups is using a relatively small batch size as opposed to a larger batch size. This follows common sense as, when running multiple threads, utilizing a smaller batch size results in short, as opposed to long, bursts of computational work, leading to improved run times. It should also be noted that, in all cases, the combination of threads used and batch size results in a number of samples that is slightly greater than those required for this in the serial case (e.g. $3,500 \times 33 = 115,000$ which is roughly the number of samples required for convergence when evaluating RTS96). This suggests that, if it is possible, values should be chosen for number of threads used and batch size that result in sampling near the required number of samples. Overall, the results for the batch parallel method are as expected.

Upon inspecting the results of the Pipeline parallel method, it appears that the key to achieving significant speedup is using a very small number of threads for state generation (maximum speedup and maximum efficiency were both achieved when using only a single thread) and an appropriate, but typically small, number of threads for classification. This is a sensible conclusion as state generation is not a computationally intense process, allowing a single thread to generate a large number of states rapidly. On the other hand, the measured result is not expected. One would expect that the addition of generation and classifier threads would reduce time and increase speedup due to the rather parallel nature of the given problem. This should occur as more threads of each type should be able to produce and process a larger number of states more rapidly. Yet, it does not. Instead, there is an eventual plateau in time reduction, speedup, and efficiency for all three test systems. As the problem is known to be highly parallel, this suggests that the lack of continued speedup is either due to overhead – issues in synchronization between threads that could be optimized in the future – or that the parallel nature has been fully exploited at this problem size, leading to the need for larger test systems to further exploit the benefits of parallelization.

Considering the scalability of both methodologies, the major concern is formed around how each algorithm will scale with 1) Additional compute resources, 2) Increased problem/system size, and 3) A simultaneous increase in computational resources and problem/system size. For all three of these cases,

the trends in efficiency in Figs. 9-11 and Figs. 18-20 are of utmost interest. For the batch parallel method, the general trend is that efficiency decreases while either the number of threads used or the batch size is increased. This suggests that the method, while useful, may not scale well as the addition of threads or batch size leads to decreased efficiency, or utilization of resources. This makes sense based on the comments on optimal configuration previously made – if a large batch size is coupled with an excessive thread count, overhead is added to the process. In considering the pipeline parallel method, the trend is similar to that of the batch parallel method, though there is an initial increase in efficiency (and decrease in computation time) as the number of classification threads is initially increased to between five and ten when considering all test systems.

In order to further examine the issue of scalability and determine if issues regarding scalability are due to overhead/implementation or a lack of parallelism in the problem, the Karp-Flatt metric [16] is introduced as shown in (9). While this metric provides an experimental method of estimating the serial portion of a program, k_m, it also provides insight into the source causes of scalability issues. For instance, if the Karp-Flatt metric grows along with the total number of threads used, overhead is typically the issue blocking scalability. If this same metric acts otherwise when an increase in total number of threads is introduced, then the issue is typically related to the portion of the problem which is inherently sequential. For the batch parallel method, the Karp-Flatt metric is shown graphically in Figs. 21 - 23. As the trend of the metric is continually increasing (while the efficiency decreases), it can be assumed that the scalability issue related to the batch parallel method is one of overhead in computation, possibly due to synchronization between threads or the additional (and excessive) computational burden created by increased batch sizes.

Considering the pipeline parallel method, the Karp-Flatt metric is depicted in Figs. 24 - 26. In all of these figures, the initial trend is a decrease in the Karp-Flatt metric, suggesting that an initial increase in classifier threads leads to increased parallelism (i.e. the estimated serial portion of the code is smaller). This matches trends seen in terms of speedup and efficiency. In all cases there is a continual trend towards a slow increase after this initial decrease. As with the batch parallel method, this suggests that any scalability issues that occur above this point are likely due to overhead in computation – potentially interactions with the queue structures that allow generator threads, classifier threads, and the convergence calculations to share data.

$$k_m = \frac{\frac{1}{s_m} - \frac{1}{m}}{1 - \frac{1}{m}} \tag{9}$$

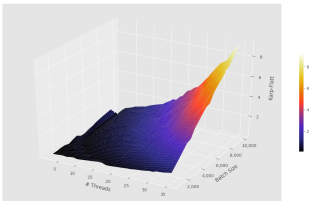

Fig. 21 Karp-Flatt metric for various combinations of Number of Threads and Batch Size for RTS79 using Batch Parallelism.

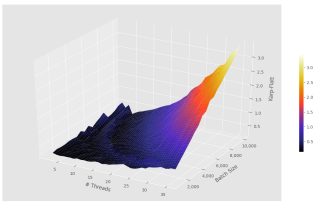

Fig. 22 Karp-Flatt metric for various combinations of Number of Threads and Batch Size for MRTS using Batch Parallelism.

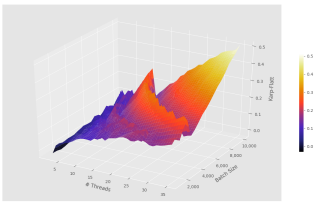

Fig. 23 Karp-Flatt metric for various combinations of Number of Threads and Batch Size for RTS96 using Batch Parallelism.

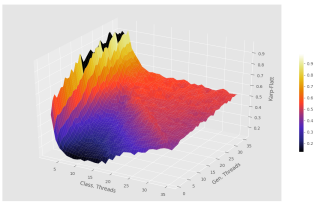

Fig. 24 Karp-Flatt metric for various combinations of Generation and Computation threads for RTS79 using Pipeline Parallelism.

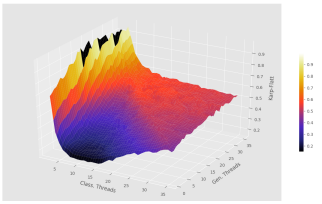

Fig. 25 Karp-Flatt metric for various combinations of Generation and Computation threads for MRTS using Pipeline Parallelism.

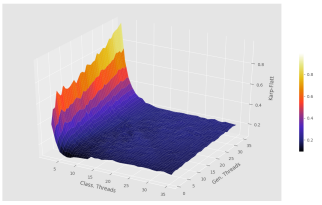

Fig. 26 Karp-Flatt metric for various combinations of Generation and Computation threads for RTS96 using Pipeline Parallelism.

5 Conclusion and Future Work

This work has introduced both pipeline and batch parallelism using OpenMP for the probabilistic reliability evaluation of composite power systems. The newly applied algorithms were benchmarked and analyzed using a cloud-based system and all source code has been made publicly available. Results show that these methodologies, while both effective, may result in a speedup upwards of $5x$ by choosing appropriate batch sizes or the proper combination of thread types. In terms of batch parallelism, this choice should accurately reflect any knowledge related to the convergence of the system, for instance how many iterations are required for convergence. For pipeline parallelism, these choices should generally use a single generator thread and roughly four to five classifier threads. While demonstrating improved results in terms of computation time and speedup, the algorithms due suffer from some issues related to scalability, but based on analysis using both efficiency and the Karp-Flatt metric, it can be shown that these inefficiencies are generally due to computational overhead, an issue that the authors hope to address in the future.

Future extensions of this work may include 1) Investigations into improved data sharing, implementation methods, and optimization to ease computational overhead and increase performance of the proposed method, 2) Implementation of similar approaches using MPI or a hybrid of OpenMP and MPI, 3) Implementing the algorithms using various accelerators, 4) Evaluation of the

proposed methods using larger computational resources, and 5) The extension of the methodologies to larger test systems.

Acknowledgements This work was supported in part by an Amazon Web Service (AWS) in Education Research Grant award.

References

1. Alba, E., Luque, G.: Evaluation of parallel metaheuristics. In: Parallel Problem Solving From Nature, pp. 9–14. Reykjavik, Iceland (2006)
2. Ali, M., Dong, Z.Y., Li, X., Zhang, P.: RSA-Grid: A Grid Computing based Framework for Power System Reliability And Security Analysis. In: IEEE/PES General Meeting, pp. 1–7. Montreal, CA (2006)
3. Borges, C., Falcão, D.: A parallelisation strategy for power systems composite reliability evaluation. In: V. Hernández, J. Palma, J. Dongarra (eds.) Vector and Parallel Processing, pp. 640–651. Springer Berlin / Heidelberg (1999)
4. Borges, C., Falcao, D.: Power system reliability by sequential monte carlo simulation on multicomputer platforms. In: J. Palma, J. Dongarra, V. Hernández (eds.) Vector and Parallel Processing — VECPAR 2000, pp. 242–253. Springer Berlin / Heidelberg (2001)
5. Borges, C., Falcao, D., Mello, J., Melo, A.: Composite reliability evaluation by sequential monte carlo simulation on parallel and distributed processing environments. IEEE Transactions on Power Systems **16**(2), 203–209 (2001)
6. Borges, C.L.T., Falcao, D.M., Mello, J.C.O., Melo, A.C.G.: Concurrent composite reliability evaluation using the state sampling approach. Electric Power Systems Research **57**(3), 149–155 (2001)
7. EPRI: Final report on research project 2473-10. Tech. rep., EPRI (1987)
8. Green, R., Wang, L., Alam, M.: High performance computing for electric power systems: Applications and trends. In: IEEE/PES General Meeting, pp. 1–8. Detroit, Michigan (2011)
9. Green, R., Wang, L., Alam, M.: Applications and trends of high performance computing for electric power systems: Focusing on smart grid. IEEE Transactions on Smart Grid **4**(2), 922–931 (2013)
10. Green, R., Wang, L., Alam, M., Singh, C.: Intelligent and parallel state space pruning for power system reliability analysis using MPI on a multicore platform. In: IEEE Conference on Innovate Smart Grid Technologies, pp. 1–8. Anaheim, California (2011)
11. Green, R., Wang, L., Alam, M., Singh, C.: Intelligent state space pruning for monte carlo simulation with applications in composite power system reliability. Engineering Applications of Artificial Intelligence **26**(7), 1707–1724 (2013)
12. Grigg, C., Wong, P., Albrecht, P., et al.: The IEEE reliability test system-1996. IEEE Transactions on Power Systems **14**(3), 1010–1020 (1999)
13. Gubbala, N., Singh, C.: Models and considerations for parallel implementation of monte carlo simulation methods for power system reliability evaluation. IEEE Transactions on Power Systems **10**(2), 779–787 (1995)
14. IEEE Committee Report: IEEE reliability test system. IEEE Transactions on Power Apparatus and Systems **PAS-98**(6), 2047–2054 (1979)
15. Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: Open source scientific tools for Python (2001–). URL http://www.scipy.org/
16. Karp, A.H., Flatt, H.P.: Measuring parallel processor performance. Commun. ACM **33**(5), 539–543 (1990). DOI 10.1145/78607.78614. URL http://doi.acm.org/10.1145/78607.78614
17. Li, F.: Distributed processing of reliability index assessment and reliability-based network reconfiguration in power distribution systems. IEEE Transactions on Power Systems **20**(1), 230–238 (2005)
18. Mitra, J., Singh, C.: Incorporating the DC load flow model in the decomposition-simulation method of multi-area reliability evaluation. IEEE Transactions on Power Systems **11**(3), 1245–1254 (1996)

19. OpenMP Architecture Review Board: Openmp application program interface. Specification (2008). URL http://www.openmp.org/mp-documents/spec30.pdf
20. Pereira, M., Balu, N.: Composite generation/transmission reliability evaluation. Proceedings of the IEEE **80**(4), 470–491 (1992)
21. Prada, J.: The value of reliability in power systems-pricing operating reserves. Tech. rep., Massachusetts Institute of Technology, Cambridge, Massachusetts (2005)
22. Singh, C., Mitra, J.: Composite system reliability evaluation using state space pruning. IEEE Transactions on Power Systems **12**(1), 471–479 (1997)
23. Wang, L., Singh, C.: Multi-deme parallel genetic algorithm in reliability analysis of composite power systems. In: IEEE/PES Power Systems Conference and Exposition, pp. 1–7. Seattle, Washington (2009)