Proceedings of the 2019 Winter Simulation Conference
N. Mustafee, K.-H.G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, eds.

ENERGY CONSERVATION THROUGH CLONED EXECUTION OF SIMULATIONS

Srikanth B. Yoginath, Maksudul Alam, Kalyan S. Perumalla

Oak Ridge National Laboratory
Oak Ridge, TN

ABSTRACT

High-performance computing facilities used for scientific computing draw enormous energy, some of them consuming many megawatt-hours. Saving the energy consumption of computations on such facilities can dramatically reduce the total cost of their operation and help reduce environmental effects. Here, we focus on a way to reduce energy consumption in many ensembles of simulations. Using the method of *simulation cloning* to exploit parallelism while also significantly conserving the computational and memory requirements, we perform a detailed empirical study of energy consumed on a large supercomputer consisting of hardware accelerator cards (graphical processing units, GPUs). We build on previous insights from mathematical analysis and implementation of cloned simulations that result in computational and memory savings by several orders-of-magnitude. Using instrumentation to track the power drawn by thousands of accelerator cards, we report significant aggregate energy savings from cloned simulations.

1 Introduction

High-performance computing facilities used for scientific computing draw enormous amount of energy for their computing and subsequent cooling requirements. The current fastest supercomputer's ability to compute over $140,000$ trillion floating-point operations per second comes with a power requirement of around 10 MW (TOP500.org 2019). The energy needs of such massive computing infrastructures are currently in focus (Scogland, Azose, Rohr, Rivoire, Bates, and Hackenberg 2015; Pakin, Storlie, Lang, Fields III, Romero Jr, Idler, Michalak, Greenberg, Loncaric, Rheinheimer, et al. 2016). Several studies have been performed to reduce energy consumption of big systems using scheduling (Ren, Lan, and van der Schaar 2013), code perforation (Hoffmann, Misailovic, Sidiroglou, Agarwal, and Rinard 2009), and performance counters (Chetsa, Lefèvre, Pierson, Stolf, and Da Costa 2014).

Previously, we had successfully demonstrated that the dynamic cloning of simulations during execution, to evaluate several different "what-if" scenarios, result in the conservation of computation and memory by several orders-of-magnitude (Yoginath and Perumalla 2018). However, the equivalent energy conservation has not been quantified yet. So far, most of the energy efficiency considerations are based on low-level computing system optimization and efficient device usage, rather than the structure of simulation systems (Fujimoto 2015). In this paper, we instrument our framework to measure energy consumed during computations and empirically evaluate the energy conservation promise of the cloned simulation executions.

Simulation Cloning is a conceptual approach in which such a tree of many related simulations is efficiently executed by dynamically minimizing the duplication of memory and computation among the simulations. The efficiency is achieved by separating the logical view and physical manifestations of the simulations in terms of their memory usage and computational operations. Logically, each simulation is an entirely separate simulation of its own. However, using cloning, the *physical* manifestation of the simulations is optimized: the common shared content across state space and virtual time along the *clone tree* hierarchy is combined at the runtime, thereby dramatically reducing the aggregate amount of computation and memory consumed by the entire tree.

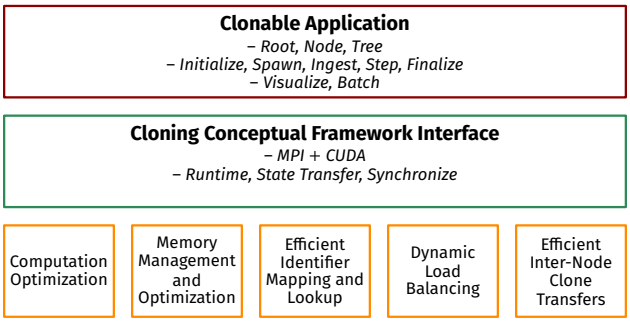


Figure 1: CloneX Software Architecture

CloneX is a novel GPU-based simulation cloning framework that we developed for large-scale high-performance computing systems, which efficiently and dynamically creates whole logical copies of simulations without full physical duplication. Figure 1 shows the software architecture of CloneX. The CloneX framework is developed over efficient GPU architecture specific parallel computing SIMD algorithms, novel memory management strategies, rapid clone identification and lookup algorithms, efficient communication algorithms and scalable load-balancing algorithms. This framework provides generic application programming interfaces through which several applications could exercise cloned simulation executions. Currently, CloneX can be readily interfaced with simulation applications in which the model is dependent on immediate neighborhood to update its simulation state at every time step. CloneX ensures that the results from any node of the simulation tree are exactly the same as one would obtain if that node is separately executed from beginning to end with its own independent copy of simulation state, and with its own initial conditions.

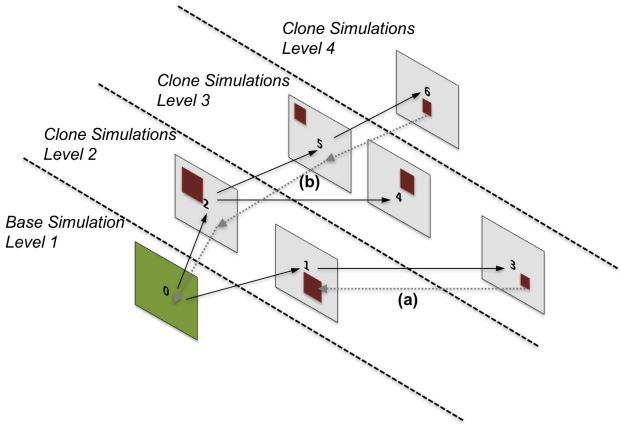
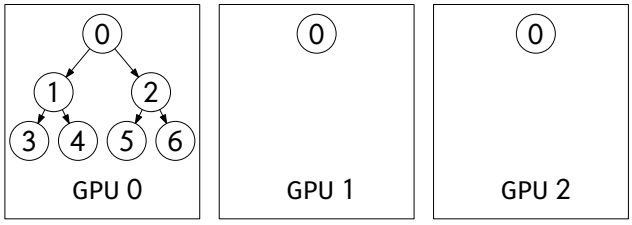


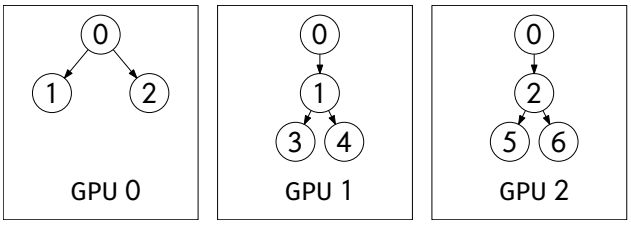
Figure 2: CloneX Architecture

In Figure 2, we show an execution snapshot of cloned simulation executions that use two-dimensional grids. The CloneX framework provides the flexibility for any executing simulation instance to *branch* into desired number of clones. In a time stepped two-dimensional grid-based simulation, it translates to the initial state of the *what-if* scenario that need to be super-imposed on the executing simulation. The memory occupied by this initial state might be as small as a single grid element, which grows in size as the simulation clone evolves in time. These simulation clone instances are represented by small brown rectangles in Figure 2, which can grow to the size of the base simulation. Further, every simulation clone is also able to branch at any time, thus increasing the depth of the simulation clone tree. The depth increases in terms of *levels* and larger the *level* number farther the simulation clone is from the base simulation in the simulation clone tree. The base simulation is at *level 1*.

Figure 3 gives a schematic of the load-balancing algorithm in CloneX. Here, the simulation starts with base-simulation executing on each GPU. As the base simulation *branches* and grows into a tree with six nodes, our dynamic load-balancing algorithm efficiently distributes the newly spawned nodes across lightly occupied nodes. This essentially results in multiple cloned simulation sub-trees with each GPU hosting one sub-tree.



(a) Before Load Distribution



(b) After Load Distribution

Figure 3: Distribution of clones among available processors

In this paper, we present our preliminary work measuring, quantifying and evaluating energy gains made through cloned simulation executions on large scale computing systems. In section 2, we discuss the hardware and software platforms, the benchmark applications, the instrumentation of CloneX to measure the energy consumed by cloned simulation executions using NVML library. In section 3, we provide a detailed performance numbers for different benchmarks with varying number of *branches* and *levels* of simulation clone execution. We conclude the paper with section 4.

2 Experimental Setup and Benchmarks

2.1 Hardware and Software

Our performance study was conducted on ORNL's Titan machine, which is a supercomputer based on a hybrid architecture with a power requirement of 8.2 MW. Titan features 18,688 compute nodes, a total system memory of 710 terabytes and Cray's high-performance Gemini network. Each node comprises of 16-core AMD Opteron processor with 32GB of host memory and NVIDIA Tesla K20 GPU accelerator with 2688 cuda cores with 6 GB of device memory.

We use our CloneX software framework to measure, compare and evaluate the energy utilization of the cloned simulation executions. This C++ software is developed using CUDA and MPI libraries. We refer the reader to (Yoginath and Perumalla 2018) for the details on the design and development of CloneX. We used NVIDIA Management Library (NVML) for energy measurements. The NVML based power readings are verified to be within 5% of error margin for the Kepler architecture based GPUs (NVIDIA 2018).

2.2 Benchmark Applications

The following three benchmarks were used in our study

Heat Diffusion Simulation We use two-dimensional heat diffusion equation as our first benchmark. *Forward time central space* (FTCS), an explicit finite-difference scheme was employed for computation.

We use Dirichlet boundary conditions for this application benchmark. This benchmark simulates to 2D heat-diffusion across a thin sheet of iron (Flaherty 2016). For cloning, each *what-if* scenario is represented by a simulation clone that is created by randomly picking a part of the domain as a new heat source (Yoginath and Perumalla 2018).

Forest Fire Simulation We use the forest fire simulations as our other benchmark to evaluate the performance of cloned simulation executions. This simplistic realization of forest fire application completely follows the model of (Balbi, Santoni, and Dupuy 1999). The process of cloning involved modeling the ignition of a small block of cells. This is achieved by resetting the selected block of cells to the ignition temperature. Igniting different points in the forest area on fire creates new simulation clones (Yoginath and Perumalla 2018).

Epidemiological Simulation we use an epidemiological model based on geographical population distributions. The geographical domain is divided into cells in which each cell contains four key state variables, each of which is a population count: S for susceptible, E for exposed, I for infected, and R for recovered. This is known as the SEIR model, to which we also add movement of individuals across cells. The initial population densities in the cells are assigned based on population databases of countries made available by the United Nations. The base simulation tracks the propagation dynamics based on the SEIR model. The clones are spawned based on a variety of *what-if* scenarios, such as new outbreaks (cells with increased infected count), quarantines (restricted spatial movement), vaccination (reducing susceptible counts), and hospitalization (increasing recovered counts) (Perumalla and Seal 2012).

2.3 Benchmark Scenarios

All benchmarks work on 2048×2048 input grid sizes and the initial size of the clones is 64×64.

Single-node scenario This experimental scenario is used to quantify the energy conserved and compare it with the computational and memory savings achieved due to cloned execution. In this experiment, we run all the benchmark applications for 100 timesteps. Each simulation clone spawns 6 clones (*branches*) after every 20 timesteps. This essentially results in a 5-*level* simulation clone tree, with each clone *branching* off 6 ways at every level. This results in the spawning and execution of 1,555 simulation clones, including the base (full) simulation.

Multinode scenario This experimental scenario was designed to evaluate the strong scaling ability of CloneX. Here, each simulation clone spawns 4 clones (*branches*) after every 10 timesteps and the benchmark applications are run for 100 timesteps. This essentially results in a 10 *level* simulation clone tree, with each clone *branching* off 4 ways at every level. This results in the spawning and execution 349,525 simulation clones, including the base simulation. This experimental scenario is executed on varying number of GPUs: 128, 256, 512, 1024 and 2048.

2.4 Power and Energy Measurement

Instrumentation To measure the energy consumption of the simulation runs executing on GPUs, we use NVML API to obtain the power usage of the device in milli-Watts. The NVML call to query power was used after launching the simulation specific cuda kernels and before *cudaDevicesyncronize*. Since, CloneX is developed to execute on large-scale GPU-based computational platforms, we only measure the energy consumption of the GPUs.

Measurement In Figure 4, we show the power curve (red dotted-lines) plotted using the queried power values during the CloneX benchmark execution scenario involving 5 *levels* with 6 *branches*. We calculate the energy for this execution scenario by measuring the area under power curve (shaded green). To achieve this, we recorded the time just before and after the execution of *cuda* kernels. We calculate the area under the rectangle formed by period of kernel execution and queried power value to compute energy consumed at during cuda kernel execution. These individual energy computations are aggregated to obtain the total energy consumed by a simulation benchmark execution.

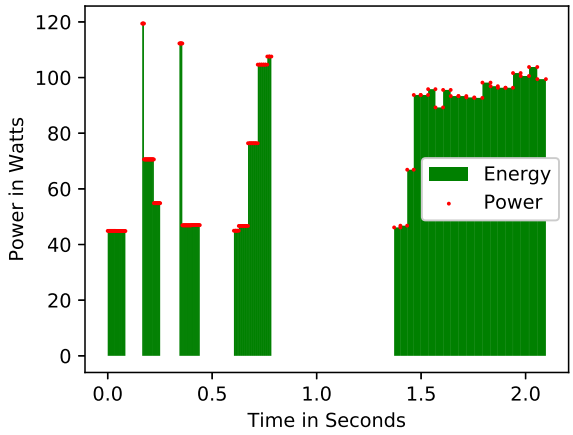


Figure 4: Power readings from NVML

Every measured point in Figure 4 represents the instance at which the power was measured. From Figure 4, the non-uniformity in the power query frequency can be observed. It is at these gaps the CloneX framework creates the cloned instances. During this period, the data pertaining to cloned instances are copied to the device memory and there is not much GPU computation being performed. In the multi-node runs, each GPU queries its power utilization and the energy consumed by each GPU is calculated.

We would like to note here that the energy utilized by the GPUs is estimated while executing the simulation benchmarks. The energy lost in the form of heat due to these computations is not measured. The temperature of the device recorded during computations remained almost constant and this was expected because of the cooling system. As part of future work, we intend to account for the energy lost as heat, which provides an increased amount of precision of our measurements. Note that, for this same reason, the energy consumption estimates are conservative in this paper, and the savings from cloned simulations could be higher when heat loss is also included.

3 Performance Results

3.1 Performance Metrics

We used the following metrics to evaluate the performance of cloned simulation benchmark executions.

Computational Savings (C_s) is calculated as

$$C_s = \frac{N_c \times C_1}{P \times C_c}$$

Where N_c is the number of simulation clones, C_1 is the computational requirement of a single simulation execution, P is the number of compute nodes (GPUs) used and, C_c is the computational time required for cloned simulation execution. Here, the fraction $\frac{N_c \times C_1}{P}$ linearly scales computation time $N_c \times C_1$ across P processors.

Memory Savings (M_s) is calculated as

$$M_s = \frac{N_c \times \mu_1}{\mu_c}$$

Where, μ_1 is the memory requirement of a single simulation execution and μ_c is the aggregate memory used by cloned simulation execution. In multinode execution, the aggregate memory is the summation of memory utilized across all the GPUs.

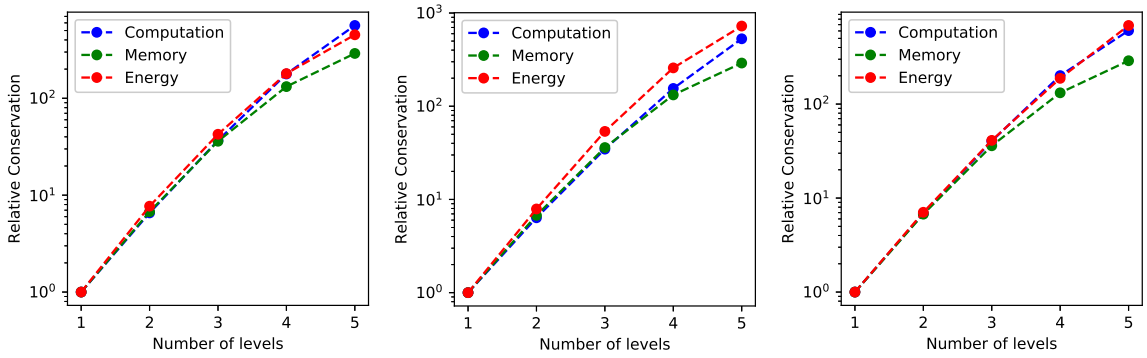
Energy Savings is calculated as

$$E_s = \frac{N_c \times \varepsilon_1}{\varepsilon_c}$$

Where, ε_1 is the energy requirement of a single simulation execution and ε_c is the aggregate energy utilized by cloned simulation execution. In multinode execution, the aggregate energy is the summation of energy utilized across all the GPUs.

3.2 Single Node Execution

In Figure 5a, Figure 5b, and Figure 5c we show the performance trends of the benchmark diffusion, forest fire and epidemiological simulations.



(a) Diffusion (b) Forest Fire (c) Epidemiological

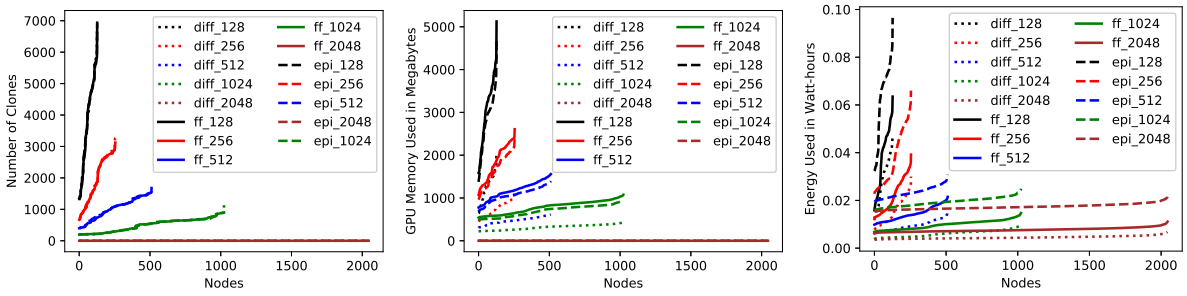
Figure 5: Relative computation, memory and energy conserved with each clone spawning six branches at every level, for (a) Diffusion simulations (b) Forest fire simulations (c) Epidemiological simulations.

These plots confirm that the amount of energy savings due to cloned simulation executions closely follow the trends of computation and memory, and can be expected to result in orders-of-magnitude energy savings on a single GPU. As seen in the Figure 5a, Figure 5b and Figure 5c, the energy conservation appear to be either equivalent or slightly better than computational gains.

3.3 Multinode Execution

Effects of Load Balancing In Figure 6a, we provide the curves that show the distribution of simulation clones across the parallel nodes for all the three benchmarks executing the multinode benchmark scenario. This plot confirms that distribution of number of clones on the GPUs are same across all the benchmarks, suggesting the load-balancing algorithm behaved in a consistent manner across all the benchmark applications. The load-balancing algorithm executes when new simulation clones are spawned and takes the memory occupancy of each node involved in the parallel computation into consideration before moving the new simulation clones from the spawning node to the execution node. The memory-occupancy of each node at the end of the benchmark execution is shown in Figure 6b. This load-balancing strategy appears to do better in terms of per-node energy utilization for large number of execution nodes (512 to 2048), as seen in Figure 6c

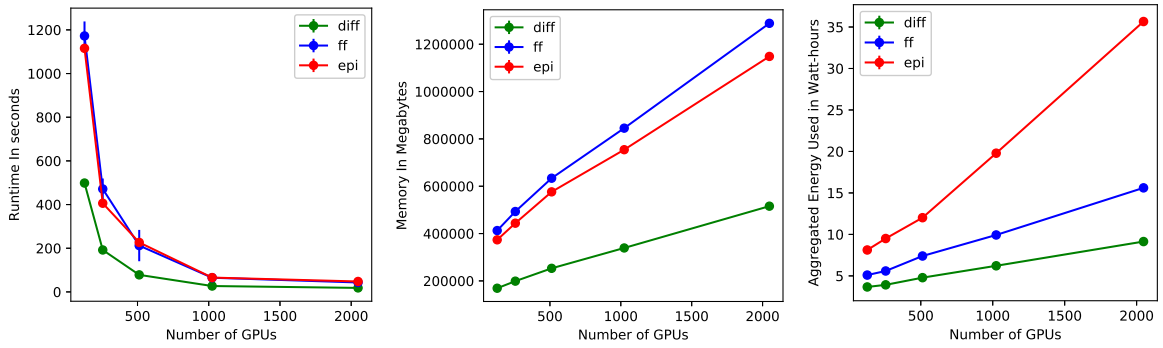
Further, we also observed that the inclusion of power polling instrumentation to the source code made no significant impact on the runtime performance of the benchmarks. This is significant because new load balancing algorithms can be designed directly based on the energy measurements made. This can be helpful when newly spawned simulation clones significantly differ from each other and the simulation clone branching are random and runtime dependent. .



(a) Number of Clones (b) GPU memory (c) Energy consumption

Figure 6: (a) Number of Clones (b) GPU memory (c) Energy consumption, of each execution node in strong-scaling evaluation of all the benchmarks using 128 to 2048 nodes. Here and in the following plots *diff, ff, epi* represent diffusion, forest fire and epidemiological simulations, respectively.

Strong Scaling Results In Figure 7a the parallel execution show good runtime gains with the increase in number of GPUs. Each of the simulation runs were replicated 5 times to avoid random noises. In this plot we use the average runtime of the replicated runs with a 95% confidence interval error bars. As seen from the plot, the runtime is very consistent among replicated runs.



(a) Execution Time (b) Memory Utilization (c) Aggregated Energy

Figure 7: Execution time, Memory utilization, and Aggregated energy, utilized by all the nodes in the strong-scaling evaluation of all the benchmarks using 128 to 2048 nodes

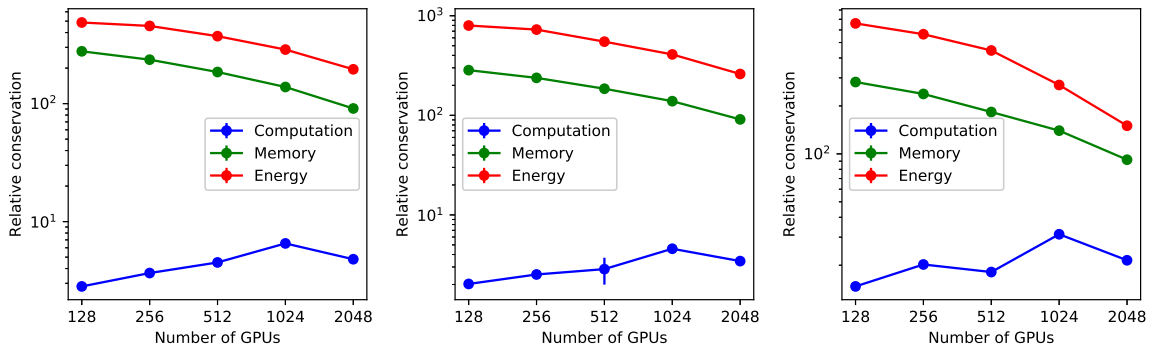
In Figure 7b the aggregate memory utilization increases with the increase in number of GPUs used in cloned memory execution, this is because of the replicated root simulation along with other replicated parent clones that result with the increase in the number of GPUs used in cloned simulation execution. We aggregated the memory used by each GPU during parallel computation of cloned simulation execution and calculated M_s using single simulation execution memory requirement and number of clones the scenario.

Similar to aggregated memory calculations, we aggregated the energy consumed by each GPU in the parallel execution for each multinode benchmark execution and the corresponding plots are shown in Figure 7c. Each of the simulation runs were replicated 5 times to avoid noise. The average aggregated energy of the replicated runs along with the 95% confidence interval is shown in the plot.

In these multinode execution runs, we observe decrease in the overall energy consumption with increase in number of nodes, as seen in Figure 7c. This behavior can be expected since each of the nodes and execute the root simulation along with other simulation clones. A higher energy consumption with the increase

in number of GPUs are observed in epidemiological simulations. In contrast the diffusion and forest fire simulation benchmarks demonstrate low energy consumption with increase in the number of GPUs.

3.4 Overall Conservation



(a) Diffusion (b) Forest Fire (c) Epidemiological

Figure 8: Relative computation, memory and energy conserved in multinode execution for simulation benchmark scenarios (a) Diffusion simulations (b) Forest fire simulations (c) Epidemiological simulations.

In this section we quantify the computational, memory and energy gains as we move beyond single node executions. Figure 8a, Figure 8b and Figure 8c, show the conservation trends for computation, memory and energy in relation to replicated runs. These curves were generated utilizing the runtime, memory and energy utilization values of a single simulation run. As mentioned in Section 2.3 the multinode benchmark scenario computes 349,525 simulation clones. By multiplying the single simulation performance values with 349525, we obtained the base line to compute multinode performance gains. We observe that even with tapering energy conservation curve across all the simulation benchmarks, but we still see an energy gain of over two orders of magnitude with cloned simulation executions over replicated simulation executions.

We observe that the memory gains slightly taper as the number of GPUs used in CloneX simulation executions increase. However the memory gains of over couple of orders of magnitude can be seen across all the cloned simulation benchmark executions over replicated simulation executions. However, the runtime appears to relatively suffer due to replicated root simulation executions and communication costs incurred in the multinode executions. With the increase in number of GPU nodes the best observed gain in computational savings over replicated runs is just over an order-of-magnitude for multinode execution scenarios. Further, the strong scaling performance evaluation scenario yield in relatively low computational savings with the increase in number of GPUs.

4 Conclusion

Using actual implementation of simulation cloning in software, we empirically evaluated the energy benefits from cloned execution of ensembles of GPU-based simulations. We found that the energy savings observed in actual runs are in line with those predicted from theoretical analyses of computational savings and memory conservation, which essentially result in orders-of-magnitude reduction in energy consumed on a single GPU. The results show that simulation cloning is an effective approach to perform ensembles of *what-if* decision simulations in an energy-efficient simulation. This is especially pertinent on high-performance computing systems that extensively rely on accelerators such as GPUs to achieve peak performance. On large-scale executions, we found that the instrumentation overhead for probing the power consumption is negligible. Due to the low overhead of tracking the power draw across many processors, it is possible to adopt new designs of energy-efficient load-balancing algorithms based on the energy consumption data

of each GPU device across thousands of computational nodes. Further, we observed retention of energy savings with the increase in the number of GPUs for large-scale runs are more promising in comparison with the computation and memory savings.

REFERENCES

Balbi, J., P. Santoni, and J. Dupuy. 1999. "Dynamic modelling of fire spread across a fuel bed". *International Journal of Wildland Fire* 9(4):275–284.

Chetsa, G. T., L. Lefèvre, J.-M. Pierson, P. Stolf, and G. Da Costa. 2014. "Exploiting performance counters to predict and improve energy performance of HPC systems". *Future Generation Computer Systems* 36:287–298.

Flaherty, Joseph E Accessed on: 2016. "Multi-Dimensional Parabolic Problems".

Fujimoto, R. 2015. "Parallel and distributed simulation". In *Winter Simulation Conference (WSC), 2015*, 45–59. IEEE.

Hoffmann, H., S. Misailovic, S. Sidiroglou, A. Agarwal, and M. Rinard. 2009. "Using code perforation to improve performance, reduce energy consumption, and respond to failures".

NVIDIA 2018. "NVML API Reference Guide". https://docs.nvidia.com/deploy/nvml-api/index.html. [Online; accessed Nov 2018].

Pakin, S., C. Storlie, M. Lang, R. E. Fields III, E. E. Romero Jr, C. Idler, S. Michalak, H. Greenberg, J. Loncaric, R. Rheinheimer et al. 2016. "Power usage of production supercomputers and production workloads". *Concurrency and Computation: Practice and Experience* 28(2):274–290.

Perumalla, K. S., and S. K. Seal. 2012. "Discrete event modeling and massively parallel execution of epidemic outbreak phenomena". *SIMULATION* 88(7):768–783.

Ren, S., C. Lan, and M. van der Schaar. 2013, May. "Energy-efficient design of real-time stream mining systems". In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 3592–3596.

Scogland, T., J. Azose, D. Rohr, S. Rivoire, N. Bates, and D. Hackenberg. 2015. "Node Variability in Large-scale Power Measurements: Perspectives from the Green500, Top500 and EEHPCWG". In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, 74:1–74:11. New York, NY, USA: ACM.

TOP500.org Accessed on: 2019. "TOP500 List - November 2018".

Yoginath, S. B., and K. S. Perumalla. 2018, January. "Scalable Cloning on Large-Scale GPU Platforms with Application to Time-Stepped Simulations on Grids". *ACM Trans. Model. Comput. Simul.* 28(1):5:11–5:26.

ACKNOWLEDGEMENTS

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

AUTHOR BIOGRAPHIES

Srikanth B. Yoginath is a Research Staff Member in the Computer Science and Mathematics division of the Oak Ridge National Laboratory, Oak Ridge, TN. Dr. Yoginath earned his PhD in Computational Sciences and Engineering from Georgia Institute of Technology in 2014. His research areas include large-scale parallel and distributed simulations, cyber physical system security, machine learning and deep learning.

Maksudul Alam is a Postdoctoral Research Associate at the Oak Ridge National Laboratory (ORNL). He obtained a PhD from Virginia Tech, Blacksburg, in 2016 in the area of parallel algorithms for high performance computing systems. At ORNL, Dr. Alam works on several research problems addressing

image processing, network generation, deep learning, and simulation.

Kalyan S. Perumalla is a Distinguished Research Staff Member and Manager at the Oak Ridge National Laboratory (ORNL). Dr. Perumalla founded and currently leads the Discrete Computing Systems Group in the Computer Science and Mathematics Division at ORNL. He is as an Adjunct Professor in the School of Computational Sciences and Engineering at the Georgia Institute of Technology (Georgia Tech) and a Joint Professor in the Department of Industrial and Systems Engineering at the University of Tennessee, Knoxville.