2022

Runtime Power Allocation Based on Multi-GPU utilization in GAMESS

Masha Sosonkina
Old Dominion University, msosonki@odu.edu

Vaibhav Sundriyal
Old Dominion University, vsundriy@odu.edu

Jorge Luis Galvez Vallejo

Original Publication Citation

Sosonkina, M., Sundriyal, V., & Vallejo, J. L. G. (2022). Runtime power allocation based on multi-GPU utilization in GAMESS. *Journal of Computer and Communications*, *10*(9), 66-80. https://doi.org/10.4236/jcc.2022.109005

Scientific Research Publishing

Journal of Computer and Communications, 2022, 10, 66-80
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Runtime Power Allocation Based on Multi-GPU Utilization in GAMESS

Masha Sosonkina[1], Vaibhav Sundriyal[1], Jorge Luis Galvez Vallejo[2]

[1]Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, USA
[2]Department of Chemistry, Iowa State University, Ames, USA
Email: msosonki@odu.edu, vsundriy@odu.edu, jg4@iastate.edu

How to cite this paper: Sosonkina, M., Sundriyal, V. and Galvez Vallejo, J.L. (2022) Runtime Power Allocation Based on Multi-GPU Utilization in GAMESS. *Journal of Computer and Communications*, **10**, 66-80.
https://doi.org/10.4236/jcc.2022.109005

Received: August 11, 2022
Accepted: September 19, 2022
Published: September 22, 2022

Abstract

To improve the power consumption of parallel applications at the runtime, modern processors provide frequency scaling and power limiting capabilities. In this work, a runtime strategy is proposed to maximize performance under a given power budget by distributing the available power according to the relative GPU utilization. Time series forecasting methods were used to develop workload prediction models that provide accurate prediction of GPU utilization during application execution. Experiments were performed on a multi-GPU computing platform DGX-1 equipped with eight NVIDIA V100 GPUs used for quantum chemistry calculations in the GAMESS package. For a limited power budget, the proposed strategy may deliver as much as hundred times better GAMESS performance than that obtained when the power is distributed equally among all the GPUs.

Keywords

Time Series Forecasting, ARIMA, Power Allocation, Performance Modeling, GAMESS, GPU Utilization

1. Introduction

Power and the subsequent energy consumption pose major challenges in the design of large scale systems. The power/energy constraints are due to many reasons with technical and economical costs being the primary. Therefore, power and energy consumption is a major obstacle to application scalability, availability, and affordability, and it is urgent to develop techniques that optimize energy consumption while maximizing performance. Hurdles to such optimization are in part due to 1) a great variability in modern high-performance application workloads and 2) complexity of modern hardware architectures. These two fac-

tors have to be accurately modeled to predict runtime performance under different power levels.

In authors' previous work [1], machine learning (ML) modeling was used to provide accurate performance models, which accounted for a multitude of hardware characteristic as they relate to the application runtime changes. For the accurate predictions, it is of utmost importance to provide accurate estimations for the said characteristics, which may serve as input features to the ML performance models executing dynamically. In [1], the authors considered *history-window* methods with averaging across a few past time intervals. In this work, time series forecasting methods are explored for feature value prediction. Given a real-world quantum chemistry application GAMESS [2], its feature value patterns are first explored as a function of time. Next, time series forecasting methods are applied to predict their future values during the runtime. To test the resulting solution in modern high-performance computing (HPC) scenarios, it was implemented as a foundation of a novel runtime strategy that allocates a given power budget among the multiple GPUs executing an application for maximum performance. The strategy operates in a manner transparent to the application and utilizes a time-slice based approach to set the power limits for the next timeslice. In a nutshell, the main contributions of this work are as follows:

- Explored the feature value patterns with respect to time in GAMESS GPU executions. In particular, considered GPU utilization feature.
- Selected and justified an appropriate time series forecasting method that may be applicable to a variety of HPC GPU calculations in GAMESS.
- Proposed a novel runtime strategy to maximize GPU execution performance under a given power budget.
- Deployed the chosen forecasting method in the runtime strategy and demonstrated its superior performance to standard power allocation approaches.

The structure of the rest of the paper is as follows. Section 2 provides literature review. Section overviews GPU calculations in GAMESS used in this work. Section 4 provides methodology to model runtime behavior of the GAMESS GPU workload and studies applicability of different time series forecasting algorithms. Section 5 provides methodology and description for the runtime strategy to allocate available power. Section 6 shows experimental research results and Section 7 concludes the paper.

2. Review of Published Related Work

Power consumption is one of the principal design constraints for the modern exascale systems. To mitigate the resulting operating costs and prohibitive failure rates, researchers have been devising strategies to budget power on system components. In this section, a brief discussion of previous work in power capping and closely related work in system-level power and energy savings is provided.

The strategies to budget the power consumption for modern computing sys-

tems come in two forms: 1) DVFS/CPU throttling for processor and memory and 2) hardware-enforced power bounds using such interfaces as Intel(R) RAPL [3]. The authors in [4] propose a multi-input multi-output (MIMO) power control algorithm to distribute a given power budget between the processor and memory domains to maximize the application performance. A machine learning technique to determine the sensitivity of application performance with respect to CPU and DRAM power capping was proposed in [5] and used to devise a strategy for power budgeting. A runtime system termed *conductor* was proposed in [6] which utilizes configuration space exploration and adaptive power balancing to maximize performance under a hardware-enforced power cap. A multilevel power distribution framework termed *CLIP* was proposed in [7], which estimates per node power budget using the workload characteristics and utilizes memory accesses to determine CPU and memory affinity.

Many strategies have been developed to improve GPU energy efficiency by utilizing DVFS, micro-architectural techniques, workload division and application specific information. Zamani *et al.* [8] propose a framework for matrix multiplication employing undervolting beyond minimum operating voltage to reduce energy consumption of GPUs. To guard against any rise in faults by such undervolting, a fault-tolerance algorithm was also proposed in [8]. Authors in [9] characterize and demonstrate the NP-hardness of optimal task scheduling problem on GPUs and propose a constant approximation algorithm assuming that GPU cores can scale with continuous frequencies.

Guerreiro *et al.* [10] develop microbenchmarks to devise GPU power models using an iterative approach and the performance counter parameters. The work in [11] proposes strategies to modify application and CUDA parameters, such as grid and thread dimensions, to improve GPU utilization and energy efficiency. Kernel fusion, which combines two kernels into a single thread, is proposed in [12] to improve GPU utilization and reduce energy consumption. Greengpu [13] involves low level programming and memory management with custom pthread-based kernel launches for the GPU to divide workload between CPU and GPU for synthetic benchmarks. The authors in [14] utilize software prefetching and DVFS to reduce GPU energy consumption.

The work in [15] proposes PowerCoord that dynamically controls the power of the PKG and GPUs to cap power consumption while seeking to maximize performance of the system in which multiple jobs are executing. The problem of job co-scheduling on an integrated system with both CPUs and GPUs under a power cap was studied in [16]. Factors, such as memory, power contention and job period, were observed to affect performance and considered in heuristic algorithms for performance enhancing co-schedules. Authors in [17] study the gaming workloads from the point of view of PID controllers, least mean squares (LMS) and autoregressive moving average (ARMA) to manage their power consumption. In [18], workload characterization is performed using time series for reducing CPU power consumption.

The work described in this paper differs from the related work in that it considers not only GPU power allocation *dynamically* but also the distribution of power among multiple GPUs for maximum performance. Furthermore, a time series approach was used to model GPU streaming-multiprocessor (SM) utilization during the execution.

3. Overview of GAMESS Calculations on GPUs

GAMESS [2] [19] is one of the most representative freely available quantum chemistry applications used worldwide for *ab initio* electronic structure calculations. A wide range of quantum chemistry computations may be accomplished using GAMESS, ranging from basic Hartree-Fock and Density Functional Theory computations to high-accuracy multi-reference and coupled-cluster computations.

The high performance multi-GPU capabilities in the GAMESS/LibCChem [20] suite of programs include a GPU-accelerated Fock build [21], a full implementation of the Self-Consistent-Field (SCF) method [22] including the one-electron integrals and the Direct-Inversion of the Iterative Subspace (DIIS) algorithm. These routines have been scaled up to the entirety of the Summit supercomputer [23] and have demonstrated excellent parallel efficiency when coupled with fragmentation methods [24] [25]. A brief discussion of the overall algorithm is included here, for more details refer to [21] [25]. The GPU-accelerated SCF program is currently available only in a non-publicly released version of GAMESS, such that GAMESS performs the basic routines of reading the input file, setting up the basis set information, and creating the guess matrix for the SCF calculation. The new GPU-accelerated SCF programs accepts the basis set information, the density matrix, and the coordinates of the system to begin the calculation.

The overarching scheme for the SCF program includes a coordinator/worker dynamic work balancing algorithm, the steps of which are as follows: Firstly, the basis set information is accepted, the shells and shell pairs are constructed. Secondly, the shell pairs are sorted and stored in the *binned* batch container [22], which ensures a load balanced distribution of the work among the processes. Thirdly, the coordinator process evaluates the one-electron integrals on the GPU and transfers the Hamiltonian (H_{core}) matrix back to the CPU. After the one-electron integrals have been computed, the process to evaluate and digest the two-electron integrals begins.

The coordinator process binds the workers to a GPU, binding one rank to one GPU exclusively. The batch-binned shell pair container is copied among the ranks. The coordinator process statically assigns the first batches of integrals to be evaluated to a GPU. Each batch contains a *batch-size* number of shell pairs, the value of which is set in the input file with the default of 2560 shell pairs. The batches are organized in a greedy fashion, having the most computationally expensive first.

In the GPU, each thread calculates a contracted integral and its results are digested into partial Fock matrices in order to avoid race conditions. After a worker process is done with its associated batch of work, it sends a signal to the coordinator process. The coordinator process then assigns the next available batch to that worker. This is repeated until all shell-pair batches are processed.

At this point, the partial Fock matrices are gathered in the host and the final Fock matrix is transferred back to the GPU for the DIIS algorithm to diagonalize the matrix and obtain the final Hartree-Fock energy of the respective iteration. This process continues until self-consistency is achieved, which is controlled by a user-defined convergence threshold with the default of 10^{-6}.

4. Methodology: Time Series Forecasting

Section 3 outlined the scheme of the SCF program executed on multiple GPUs that are dynamically load-balanced, from which it is inferred that all the participating GPUs have a similar pattern of their utilization. In particular, Figure 1 shows the streaming multiprocessor (SM) utilization for the eight V100 GPUs for a total of 300 seconds of an RHF calculation of a valinomycin molecule. Observe that utilization traces for all the GPUs are virtually indistinguishable in Figure 1, which graphically indicates that the design of the SCF program leads to load balanced calculations. Hence, an analysis for the first GPU only is shown in the rest of this section. To better focus on this analysis the utilization trace was extracted from Figure 1 into a separate figure, Figure 2.

A set of data points ordered in time and equally spaced are considered *time series* and thus, corresponding analysis methods are applicable. In the rest of this section, the SM utilization will be treated as time series and a particular analysis

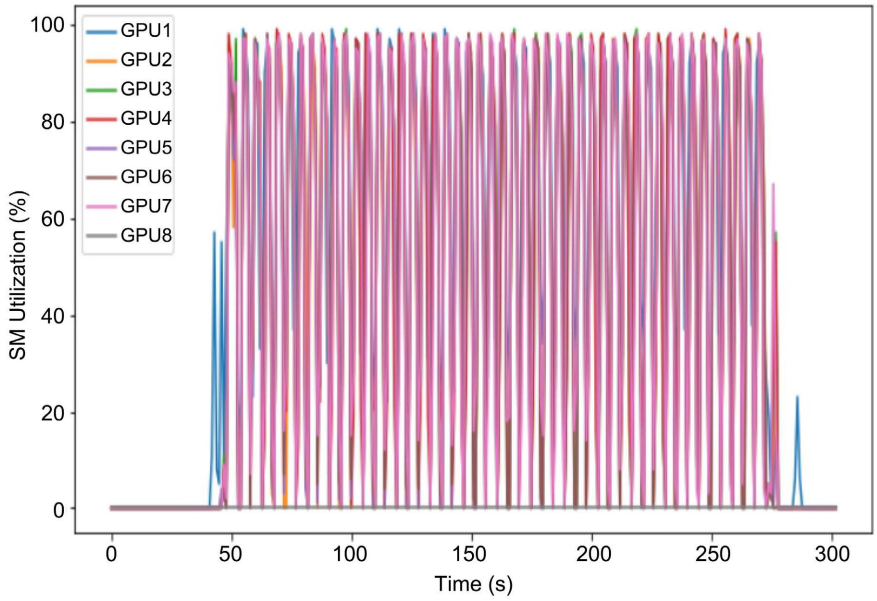

Figure 1. SM utilization on eight V100 GPUs executing GAMESS RHF calculation during 300 seconds.

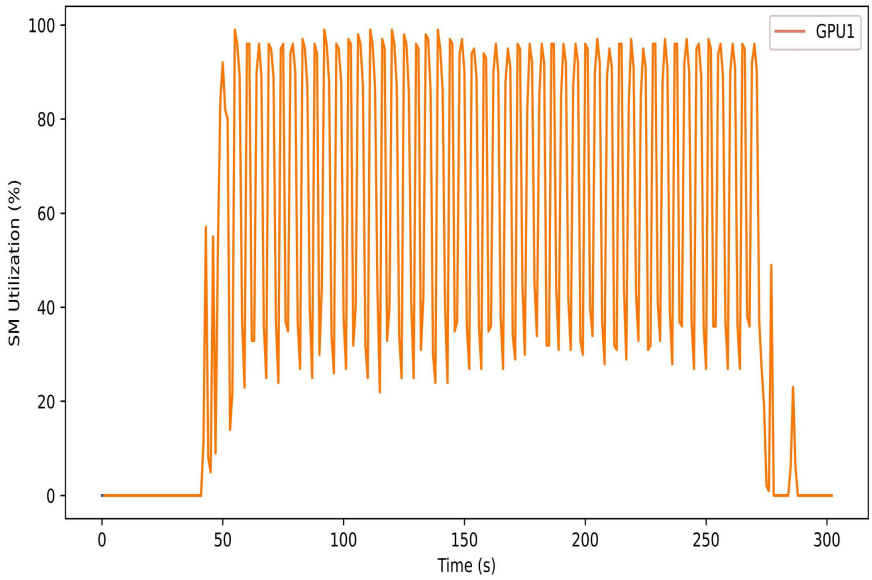

Figure 2. SM Utilization on V100 GPU #1 for the GAMESS RHF calculation executing on eight GPUs during 300 seconds (cf. **Figure 1**, which also includes this very utilization trace).

method chosen. Notice that the SM utilization, shown in **Figure 2**, exhibits a repeating pattern in intervals of similar duration, so time-series forecasting techniques can be employed to predict future utilization values. **Figure 3** outlines decision-flow stages as suggested in [26] to identify an appropriate method to predict a time series accurately. In particular, for the applicability of sophisticated forecasting models, it should be verified that the sequence of data considered is not a random walk, meaning that, in addition to its stationarity, the sequence points must be autocorrelated, thereby avoiding a random movement without any pattern. Note that, if the given time series is not stationary, differencing transformation can be applied to make it stationary, and that the augmented Dickey-Fuller (ADF) test can be used to determine if a time series is stationary by testing for the presence of a unit root. If a unit root is present, the time series is not stationary and a differencing transformation needs to be applied. This testing-differencing process is repeated. For the time series in **Figure 2**, the initial p-value output by ADF was determined to be 0.67, which is much higher than the threshold of 0.05 below which the series is considered stationary. Therefore, the first-order differencing transformation has been applied and the ADF p-value of 0.0012 was obtained indicating that this transformation made the time series stationary.

Next, per **Figure 3**, one has to check if any autocorrelation lies between the lagged values of the time series. **Figure 4** shows the autocorrelation plot with y-axis depicting the autocorrelation values and x-axis depicting the time lag. It can be observed from **Figure 4** that there is a significant autocorrelation even when the lag value is as large as 20, and hence, the time series is not a random walk, and the lag $q = 1$, where the autocorrelation is the maximum, may be used as a moving average parameter in the forecasting model.

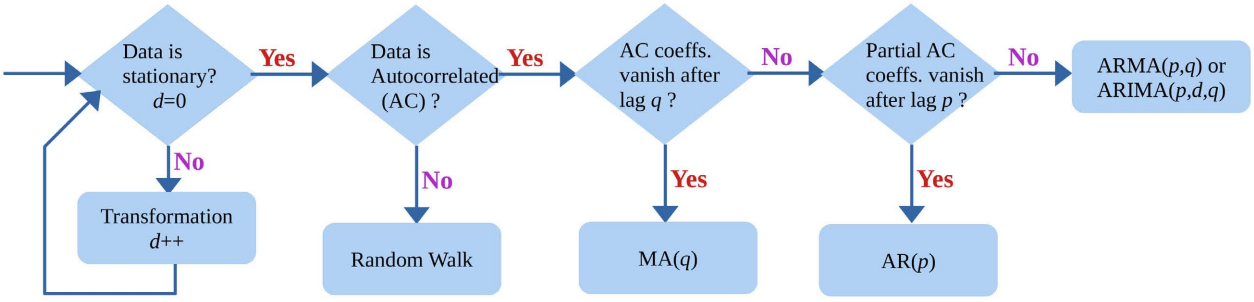

Figure 3. Decision sequence to determine which time series prediction process to select among Moving Average (MA), Autoregressive (AR), Autoregressive Moving Average (ARMA) and Autoregressive Integrated Moving Average (ARIMA), where p, q, and d are process parameters.

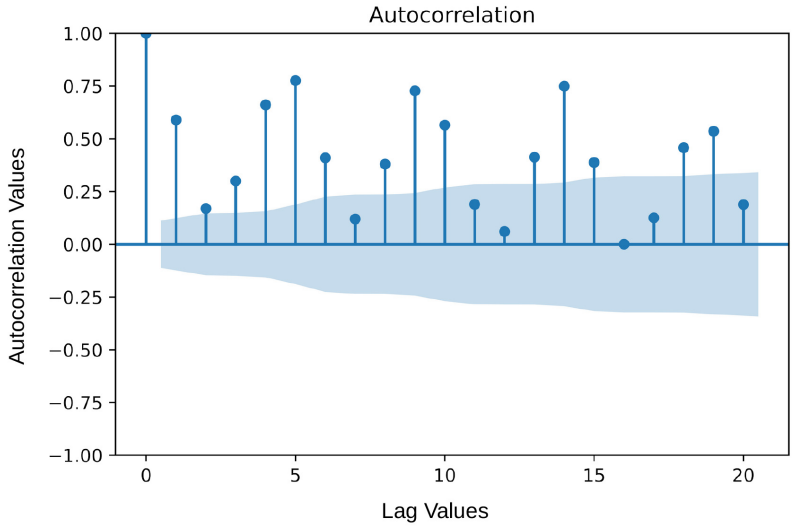

Figure 4. Autocorrelation of the SM utilization time series.

Since the autocorrelation values are not abruptly going down after a certain lag value, the given time series is not a moving average process. Following the steps in Figure 3, partial autocorrelation needs to be evaluated. Figure 5 shows the partial autocorrelation values of the time series. The values do not decrease abruptly after a certain lag, and the lag $p = 1$, where the partial autocorrelation is maximum, may be used as the order of the autoregressive process. Therefore, following Figure 3, it can be concluded that the given time series is an Autoregressive Moving Average (ARMA(p, q)) process. ARMA requires that the data is first transformed by differencing into a stationary series. Then this transformation must be reversed after the forecasting is done to obtain predictions on the original scale. Instead, one may model a non-stationary time series directly with a modified ARMA model, Autoregressive Integrated Moving Average ARIMA(p, d, q) model, where the parameter d is the integration order. The parameter d may be determined from the differencing order. The SM utilization time series considered here was differenced once, hence $d = 1$.

Figure 6 shows the observed and predicted SM utilization values when an ARIMA(1, 1, 1) process was used to predict the time series. In the same Figure 6,

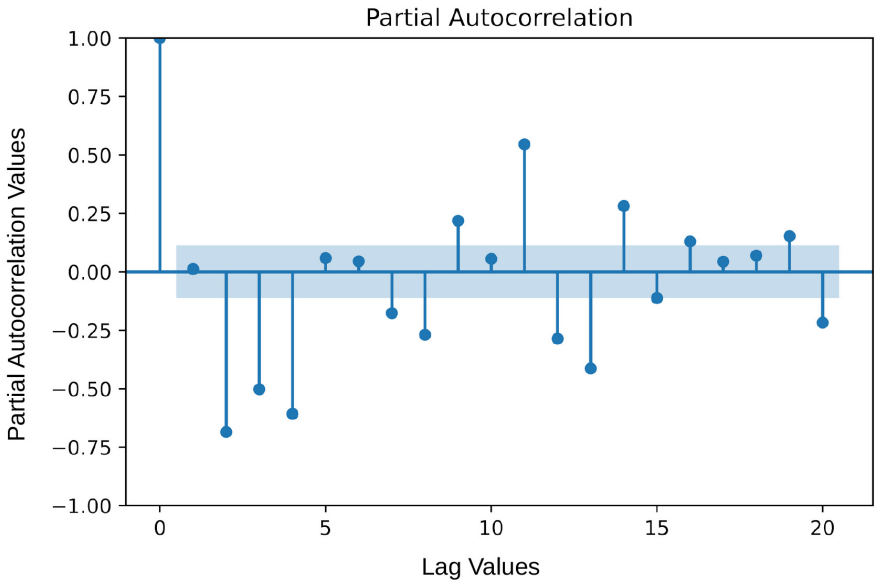

Figure 5. Partial autocorrelation of the SM utilization time series.

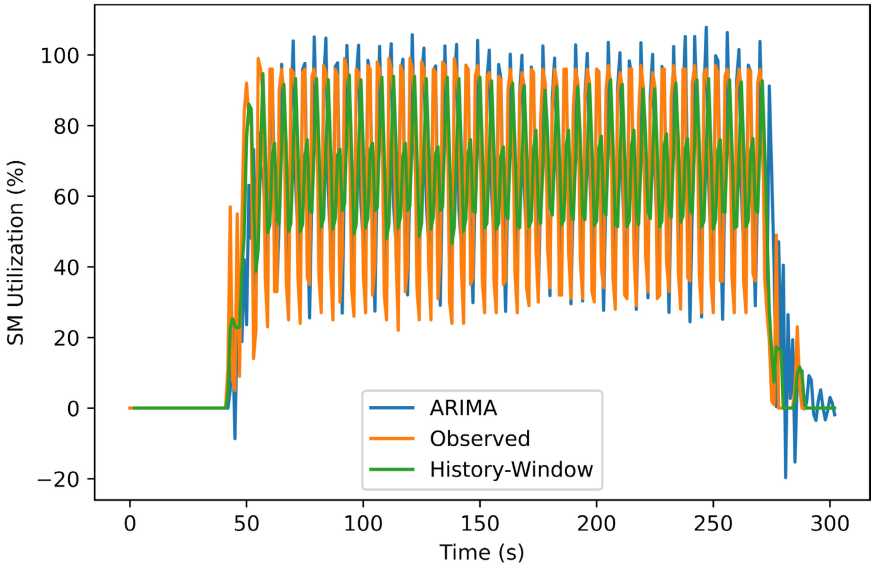

Figure 6. Comparison of the observed and predicted SM utilization values using the ARIMA(1, 1, 1) process and the history-window mechanism for the execution shown in **Figure 2**.

it is compared with the history-window prediction mechanism, in which the future value is predicted as a rolling mean of the past three values. Note that the window size of three has shown the best performance in authors' previous work on runtime power and performance modeling strategies [1] [27]. It can be observed from **Figure 6** that the predicted values significantly match the observed whereas the history-window prediction fails to capture the pattern of the observed values. Specifically, an R-squared value of 0.917 was obtained for the model predictions in **Figure 6**, which demonstrates that the ARIMA process can predict the SM utilization time series with high accuracy.

5. Methodology: Runtime Strategy to Allocate Available Power

In Section 4 the ARIMA-based prediction of the SM utilization time series was developed for GAMESS GPU execution. Here, the selected ARIMA prediction is employed in a proposed novel runtime strategy to maximize parallel application performance executing on multiple GPUs under a given power budget. In particular, Figure 7 displays the algorithmic steps of this strategy. Initially, at Step 1, the given power budget P_B is divided equally to all the available N GPUs by setting their individual power budgets P_i to $\frac{P_B}{N}$. In Step 2, the application executes for the duration τ of a timeslice. Note that, similar to authors' previous work [1] [27], the timeslice duration was set at 250 ms, which has been found here as acceptable for the accuracy of predictions as evidenced in Section 6 describing research results. In Step 3, given the P_i, the resulting SM utilization U_i is recorded for each GPU i, $i = 1, \cdots, N$. Then, the ARIMA model is employed (Step 4) in each GPU i to predict the utilization $\hat{U}_i$ for in the current timeslice r followed by collecting the predicted utilization values from all the N GPUs (Step 5). In Step 6, the power budget of each GPU is set in proportion to its predicted utilization. Step 7 executes the application for the duration τ in the timeslice r. Then, the strategy proceeds to consider the next timeslice $r + 1$.

6. Experimental Research Results

The experiments were performed on the DGX-1 compute node at Old Dominion University, having two Intel(R) Xeon(R) CPU E5-2698 20 core Broadwell-EP processors, 64 GB of DDR4 and eight NVIDIA V100 GPUs. The V100 GPU has 80 SMX units with a total of 5120 CUDA cores and 16 GB global memory.

Input Parameters:

τ	:	Duration of timeslice r.
V	:	Number of timeslices.
N	:	Number of GPUs.
P_B	:	Given power budget for all the GPUs.
i	:	This GPU number.

Algorithm:

Step 1. Set power budget $P_i = \frac{P_B}{N}$.

Step 2. Execute application during timeslice $r = 1$.
For $(r = 2,\ r \le V,\ r++)$ **do**

Step 3. Record SM utilization U_i for this GPU using the `nvidia-smi` utility.

Step 4. Predict the SM utilization $\hat{U}_i$ for timeslice r by the ARIMA model.

Step 5. Gather all $\hat{U}_i$, $i = 1, \ldots, N$.

Step 6. Set power budget $P_i = \frac{\hat{U}_i}{\sum_{i=1}^{N} \hat{U}_i}$.

Step 7. Execute application for the duration τ.
EndFor

Figure 7. Strategy pseudo-code executed in each GPU of a multi-GPU application.

Four different GAMESS/LibCChem inputs were chosen to evaluate the efficacy of the proposed strategy. Table 1 provides their input names along with descriptions. In order to select a (reduced) power budget P_B for the calculations, their total GPU unrestricted power consumption at the maximum GPU frequency was measured and is also shown in Table 1 (Column P_{max}). It can be observed, that the power consumption ranged from 1206 W to 1220 W. Therefore, same limits may be applied to all four calculations. In particular, three power budgets P_B were chosen as follows: 1000 W, 900 W, and 800 W, which represent similar percentages of the unrestricted power P_{max} for the calculations, about 81%, 73%, and 65% for the three budgets P_B, respectively.

To evaluate performance of the four inputs under the proposed runtime strategy, a baseline strategy termed allhigh was considered. In the allhigh strategy, the power limits for all GPUs were raised to their maximum values. The proposed strategy was also compared with another power-limiting strategy, termed naïve, which allocates the given power budget among GPUs based on their respective thermal design power (TDP) limits. In particular, the TDP of the V100 GPU is 300 W [28]. There are eight identical V100 GPUs in the DGX-1 compute node, so under the naïve strategy, the power budget is divided equally among the eight V100 GPUs.

Quantitatively, the performance of a strategy is proportional to the inverse of the execution time T under that strategy. Therefore, as authors suggested previously in [27], the performance degradation δ_s of a strategy s relative to the allhigh strategy is calculated as

$$\delta_s = \frac{T(\mathrm{s}) - T(\mathrm{allhigh})}{T(\mathrm{s})}.$$

Figure 8 shows the performance degradation δ for the four inputs operating under the naïve strategy for the three chosen power budgets with respect to the allhigh strategy. As expected, in Figure 8, the highest power budget of 1000 W results in the least amount of performance degradation for all inputs. Specifically, their average performance degradation was 48.9% for the 1000 W power budget. For a lower power budget of 900 W, the average performance degradation increased to 65.3%. This larger performance degradation comes from the reduction in GPU frequency due to power limiting through nvidia-smi. Further reduction in power budget to 800 W yielded even larger performance degradation for all the inputs with an average of 85.5%.

Table 1. Description and maximum power consumption P_{max} of four GAMESS inputs used in this work.

Input name	P_{max}, W	Description
w150_pcseg0	1206	Water molecule; PCSeg-0 basis set with 1950 basis functions and 450 atoms.
w150_pcseg1	1220	Water molecule; PCSeg-1 basis set with 3150 basis functions and 450 atoms.
valinomycin	1210	Naturally occurring dodecadepsipeptide used in potassium transport and as antibiotic; PCSeg-0 basis set with 882 basis functions and 168 atoms.
gly5_pcseg0	1218	Glycine molecule; PCSeg-0 basis set with 223 basis functions and 38 atoms.

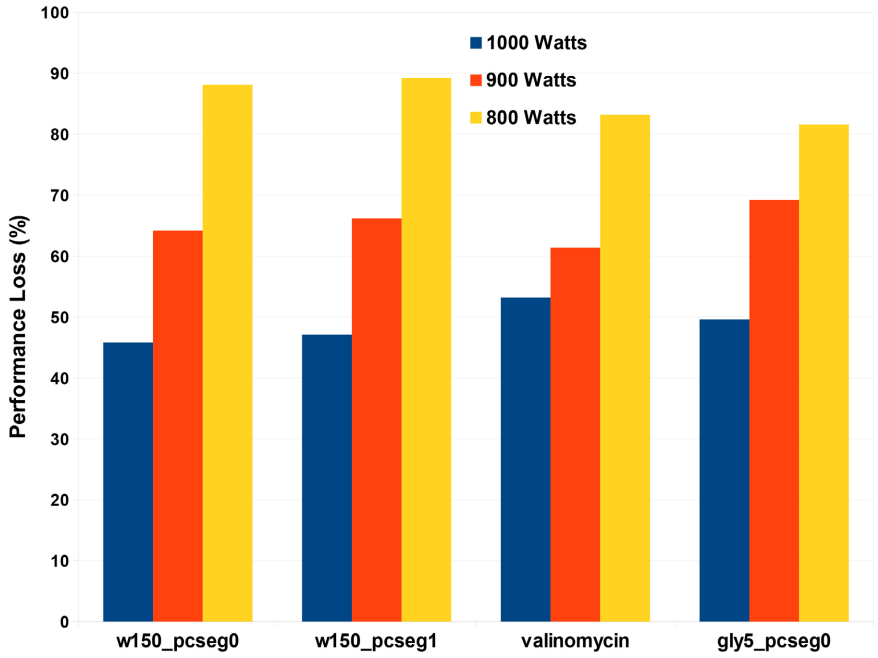

Figure 8. Performance degradation for the four inputs operating under the naïve strategy for three power budgets (1000, 900, and 800 W) with respect to the allhigh strategy.

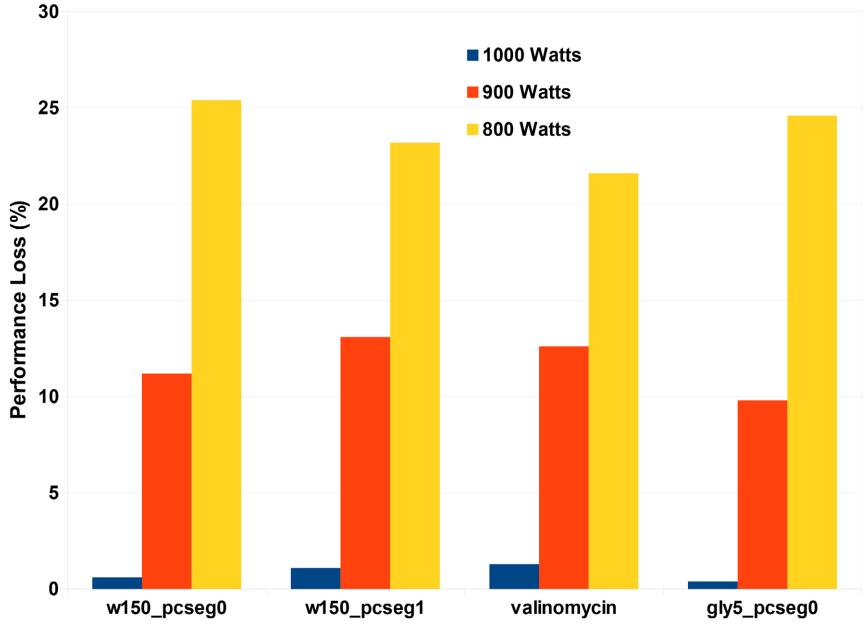

Figure 9. Performance degradation for the four inputs operating under the proposed strategy for three power budgets (1000, 900, and 800 Watts) with respect to the allhigh strategy.

Figure 9 shows the performance degradation for the four inputs operating under the proposed strategy with respect to the allhigh strategy. Here, the average performance degradation is only 0.9% for the power budget of 1000 W, which is 54.3 times less than that of the naïve strategy for the same power budget. The highest performance improvement of 124 times was observed for the

gly5_pcseg0 input by the proposed strategy as compared to the naïve one when the power budget was kept at 1000 W. This significant improvement in performance comes primarily from distributing the power budget among the eight GPUs in proportion to their utilization in the proposed strategy and from the high prediction accuracy of the underlying ARIMA process for the SM utilization. For the tight budgets of 900 W and 800 W (73% and 65% of the maximum required, respectively), the proposed strategy still performed 5.6 and 3.6 times better on average than the naïve one did so, respectively. Additionally, it is worth noting that, for lower power consumption (~1040 W) than that shown in Table 1 no performance degradation was observed for any input when the proposed strategy was employed.

7. Conclusions and Future Work

In this paper, a runtime strategy is designed and implemented to distribute available power among multiple GPUs in order to maximize performance of the quantum chemistry application GAMESS. The foundation of the strategy is the workload prediction model that considers utilization as time series and applies Autoregressive Integrated Moving Average (ARIMA) process dynamically to predict the GPU utilization in the next timeslice to be executed. Then, in each GPU, the available power is allocated in accordance with the predicted utilization as a fraction of the total utilization predicted across all the GPUs participating in the calculation. Experiments, performed on the DGX-1 compute node having eight V100 GPUs, and demonstrated that the proposed strategy provided a near maximum performance even with substantially limited power budget for four GAMESS calculations. Specifically, for the calculation of the glycine molecule, as much as a 18% reduction in the available power resulted in only a 0.4% of performance loss. It was also observed that, for the four inputs, the proposed strategy improved performance several-fold, ranging from 3.6 to 54.3 times on average, from an "equitable" power distribution strategy based on the ratio of GPU thermal design power (TDP) limits. Using the proposed strategy, a lower overall unrestricted power amount was found for which none of the four inputs exhibited a performance loss and which was ~14% less than the average unrestricted power consumption observed initially.

Future work will focus on extending the proposed strategy to a distributed system with multiple nodes and multiple GPUs to facilitate its usage on exascale platforms.

Acknowledgements

This work was supported in part by the U.S. Department of Energy (DOE) Office of Science, Office of Basic Energy Sciences, Computational Chemical Sciences (CCS) Research Program under work proposal number AL-18-380-057 and the Exascale Computing Project (ECP) through the Ames Laboratory, operated by Iowa State University under contract No. DE-AC00-07CH11358, and by the Na-

tional Science Foundation under grant CNS-1828593. The authors appreciate helpful reviewer comments.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Sundriyal, V. and Sosonkina, M. (2022) Runtime Energy Savings Based on Machine Learning Models for Multicore Applications. *Journal of Computer and Communications*, **10**, 63-80. https://doi.org/10.4236/jcc.2022.106006

[2] Schmidt, M.W., Baldridge, K.K., Boatz, J.A., Elbert, S.T., Gordon, M.S., Jensen, J.H., Koseki, S., Matsunaga, N., Nguyen, K.A., Su, S., Windus, T.L., Dupuis, M. and Montgomery Jr., J.A. (1993) General Atomic and Molecular Electronic Structure System. *Journal of Computational Chemistry*, **14**, 1347-1363. https://doi.org/10.1002/jcc.540141112

[3] Intel (n.d.) Intel® 64 and IA-32 Architectures Software Developer Manuals. https://software.intel.com/en-us/articles/intel-sdm

[4] Chen, M., Wang, X.r. and Li, X. (2011) Coordinating Processor and Main Memory for Efficient Server Power Control. *Proceedings of the International Conference on Supercomputing* (ICS '11), Tucson, 31 May-4 June 2011, 130-140. https://doi.org/10.1145/1995896.1995917

[5] Tiwari, A., Schulz, M. and Carrington, L. (2015) Predicting Optimal Power Allocation for CPU and Dram Domains. 2015 *IEEE International Parallel and Distributed Processing Symposium Workshop*, Hyderabad, 25-29 May 2015, 951-959. https://doi.org/10.1109/IPDPSW.2015.146

[6] Marathe, A., Bailey, P.E., Lowenthal, D.K., Rountree, B., Schulz, M. and de Supinski, B.R. (2015) A Run-Time System for Power-Constrained HPC Applications. *Proceedings of International Conference on High Performance Computing*, Frankfurt, 12-16 July 2015, 394-408. https://doi.org/10.1007/978-3-319-20119-1_28

[7] Zou, P., Allen, T., Davis, C.H., Feng, X. and Ge, R. (2017) Clip: Cluster-Level Intelligent Power Coordination for Power-Bounded Systems. 2017 *IEEE International Conference on Cluster Computing* (*CLUSTER*), Honolulu, 5-8 September 2017, 541-551. https://doi.org/10.1109/CLUSTER.2017.98

[8] Zamani, H., Liu, Y.L., Tripathy, D., Bhuyan, L. and Chen, Z.Z. (2019) Greenmm: Energy Efficient GPU Matrix Multiplication through Undervolting. *Proceedings of the ACM International Conference on Supercomputing* (ICS '19), Phoenix, 26-28 June 2019, 308-318. https://doi.org/10.1145/3330345.3330373

[9] Chau, V., Chu, X.W., Liu, H. and Leung, Y.-W. (2017) Energy Efficient Job Scheduling with DVFS for CPU-GPU Heterogeneous Systems. *Proceedings of the* 8*th International Conference on Future Energy Systems* (e-Energy '17), Hong Kong, 16-19 May 2017, 1-11. https://doi.org/10.1145/3077839.3077855

[10] Guerreiro, J., Ilic, A., Roma, N. and Tomas, P. (2018) GPGPU Power Modeling for Multi-Domain Voltage-Frequency Scaling. 2018 *IEEE International Symposium on High Performance Computer Architecture* (HPCA), Vienna, 24-28 February 2018, 789-800. https://doi.org/10.1109/HPCA.2018.00072

[11] Yang, Y., Xiang, P., Mantor, M. and Zhou, H. (2012) Fixing Performance Bugs: An Empirical Study of Open-Source GPGPU Programs. 2012 41*st International Confe-*

rence on Parallel Processing, Pittsburgh, 10-13 September 2012, 329-339. https://doi.org/10.1109/ICPP.2012.30

[12] Wang, G., Lin, Y. and Yi, W. (2010) Kernel Fusion: An Effective Method for Better Power Efficiency on Multithreaded GPU. 2010 *IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, Hangzhou, 18-20 December 2010, 344-350. https://doi.org/10.1109/GreenCom-CPSCom.2010.102

[13] Ma, K., Li, X., Chen, W., Zhang, C. and Wang, X. (2012) GreenGPU: A Holistic Approach to Energy Efficiency in GPU-CPU Heterogeneous Architectures. 2012 41*st International Conference on Parallel Processing*, Pittsburgh, 10-13 September 2012, 48-57. https://doi.org/10.1109/ICPP.2012.31

[14] Lin, Y., Tang, T. and Wang, G. (2011) Power Optimization for GPU Programs Based on Software Prefetching. 2011 *IEEE* 10*th International Conference on Trust, Security and Privacy in Computing and Communications*, Changsha, 16-18 November 2011, 1339-1346. https://doi.org/10.1109/TrustCom.2011.184

[15] Azimi, R., Jing, C. and Reda, S. (2018) PowerCoord: A Coordinated Power Capping Controller for Multi-CPU/GPU Servers. 2018 9*th International Green and Sustainable Computing Conference* (IGSC), Pittsburgh, 22-24 October 2018, 1-9. https://doi.org/10.1109/IGCC.2018.8752132

[16] Zhu, Q., Wu, B., Shen, X., Shen, L. and Wang, Z. (2017) Co-Run Scheduling with Power Cap on Integrated CPU-GPU Systems. 2017 *IEEE International Parallel and Distributed Processing Symposium* (IPDPS), Orlando, 29 May-2 June 2017, 967-977. https://doi.org/10.1109/IPDPS.2017.124

[17] Dietrich, B., Goswami, D., Chakraborty, S., Guha, A. and Gries, M. (2015) Time Series Characterization of Gaming Workload for Runtime Power Management. *IEEE Transactions on Computers*, **64**, 260-273. https://doi.org/10.1109/TC.2013.198

[18] Cioara, T., Anghel, I., Salomie, I., Copil, G., Moldovan, D. and Grindean, M. (2011) Time Series Based Dynamic Frequency Scaling Solution for Optimizing the CPU Energy Consumption. 2011 *IEEE* 7*th International Conference on Intelligent Computer Communication and Processing*, Cluj-Napoca, 25-27 August 2011, 477-483. https://doi.org/10.1109/ICCP.2011.6047919

[19] Gordon, M.S. and Schmidt, M.W. (2005) Advances in Electronic Structure Theory: GAMESS a Decade Later. In: Dykstra, C.E., Frenking, G., Kim, K.S. Scuseria, G.E., Eds., *Theory and Applications of Computational Chemistry*: *The First Forty Years*, Elsevier Science, Amsterdam, 1167-1189. https://doi.org/10.1016/B978-044451719-7/50084-6

[20] Barca, G.M.J., Bertoni, C., Carrington, L., Datta, D., De Silva, N., Emiliano Deustua, J., Fedorov, D.G., Gour, J.R., Gunina, A.O., Guidez, E., Harville, T., Irle, S., Ivanic, J., Kowalski, K., Leang, S.S., Li, H., Li, W., Lutz, J.J., Magoulas, I., Mato, J., Mironov, V., Nakata, H., Pham, B.Q., Piecuch, P., Poole, D., Pruitt, S.R., Rendell, A.P., Roskop, L.B., Ruedenberg, K., Sattasathuchana, T., Schmidt, M.W., Shen, J., Slipchenko, L., Sosonkina, M., Sundriyal, V., Tiwari, A., Galvez Vallejo, J.L., Westheimer, B., Włoch, M., Xu, P., Zahariev, F. and Gordon, M.S. (2020) Recent Developments in the General Atomic and Molecular Electronic Structure System. *The Journal of Chemical Physics*, **152**, Article ID: 154102. https://doi.org/10.1063/5.0005188

[21] Barca, G.M.J., Galvez-Vallejo, J.L., Poole, D.L., Rendell, A.P. and Gordon, M.S. (2020) High-Performance, Graphics Processing Unit-Accelerated Fock Build Algorithm. *Journal of Chemical Theory and Computation*, **16**, 7232-7238. https://doi.org/10.1021/acs.jctc.0c00768

[22] Barca, G.M.J., Alkan, M., Galvez-Vallejo, J.L., Poole, D.L., Rendell, A.P. and Gordon, M.S. (2021) Faster Self-Consistent Field (SCF) Calculations on GPU Clusters. *Journal of Chemical Theory and Computation*, **17**, 7486-7503. https://doi.org/10.1021/acs.jctc.1c00720

[23] Summit (Supercomputer). https://en.wikipedia.org/wiki/Summit_(supercomputer)

[24] Barca, G.M.J., Poole, D.L., Vallejo, J.L.G., Alkan, M., Bertoni, C., Rendell, A.P. and Gordon, M.S. (2020) Scaling the Hartree-Fock Matrix Build on Summit. SC20: *International Conference for High Performance Computing, Networking, Storage and Analysis*, Atlanta, 9-19 November 2020, 1-14. https://doi.org/10.1109/SC41405.2020.00085

[25] Barca, G.M.J., Galvez Vallejo, J.L., Poole, D.L., Alkan, M., Stocks, R., Rendell, A.P. and Gordon, M.S. (2021) Enabling Large-Scale Correlated Electronic Structure Calculations: Scaling the RI-MP2 Method on Summit. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (SC '21), St. Louis, 14-19 November 2021, Article No. 40. https://doi.org/10.1145/3458817.3476222

[26] Peixeiro, M. (2022) Time Series Forecasting in Python. Manning, Shelter Island.

[27] Sundriyal, V., Sosonkina, M., Poole, D. and Gordon, M.S. (2020) Runtime Power Allocation Approach for Gamess Hybrid CPU-GPU Implementation. *Concurrency and Computation: Practice and Experience*, **32**, Article No. e5917. https://doi.org/10.1002/cpe.5917

[28] NVIDIA Tesla V100 GPU Accelerator. https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf