Comparing the Performance of Numba and CUDA for Historical Analysis of Synchrophasor Data

Nakul Rao
Nicholas Liebers
Aaron St. Leger
Suzanne J. Matthews
Department of Electrical Engineering & Computer Science
United States Military Academy
West Point, NY 10996
Email: [nakul.rao, nicholas.liebers, aaron.stleger, suzanne.matthews]@westpoint.edu

***Abstract*—Modern Wide Area Monitoring systems (WAMS) incorporating Phasor Measurement Unit (PMU) technology are producing big datasets. Historical analysis of PMU data is beneficial in development of online WAMS applications, quantifying baseline normal performance, and discovering anomalous events. Energy and time-efficient computational techniques are beneficial for historical analysis of PMU data. Application workflows that include historical analysis typically combine higher-level (but slow) languages like Python with faster (but older) languages like C. This paper compares the performance of Numba Python and C for historical analysis of PMU data, on both the CPU and GPU. We augment a known PMU anomaly detection scheme with linear state estimation, implement it separately in Numba and C, test the approaches on two real-world datasets, and measure their performance on the CPU and GPU of the NVIDIA Jetson Xavier single board computer, varying the available power modes. Results demonstrate that while Numba is significantly faster than traditional Python, simplifies application development, and holds promise for PMU applications, there is a noticeable performance gap between Numba and C on the GPU.**

***Index Terms*—Numba, CUDA, synchrophasor, smart grid, historical analysis, GPU**

I. Introduction

A wide variety of programming languages and computing platforms are used for analysis of synchrophasor data. For Wide Area Monitoring Systems (WAMS) applications, it is not uncommon for SCADA designers to implement time-sensitive components in compiled languages like C/C++, and auxiliary functionality such as database querying, machine learning, and data visualization in less performant (but feature-rich) languages like Python [1]–[3]. While concise, readable, and flexible enough to enable rapid development of smart grid applications, Python's interpreted nature typically makes it a poor choice for applications that require rapid analysis (e.g. historical analysis of synchrophasor data). Workflows incorporating multiple languages are significantly harder to maintain and secure; keeping a workflow entirely in one language greatly simplifies application development.

A variety of methods are available to expedite historical analysis of PMU data. Specifically, serial methods can be parallelized on multicore CPUs or manycore GPUs to speed up computation. Researchers have successfully applied parallel techniques to implementations of Weighted Least Squares (WLS) state estimation for improved performance [4]–[7].

Most GPU-based applications for the smart grid are implemented using NVIDIA's Compute Unified Device Architecture (CUDA), which is arguably the most popular API used for GPGPU applications, and typically implemented in C/C++ and Fortran. While C/C++ applications yield fast executables, they are significantly more difficult to secure and program than Python applications. In addition, CUDA is a notoriously difficult language for even experienced developers to leverage.

Given Python's critical role in many scientific computing applications, researchers have explored creating Python bindings for CUDA. The CuPy [8], [9] Python library allows programmers to embed C/C++ CUDA kernels in their Python code and launch the kernels using a provided API. While this method results in kernel speeds that are comparable to their C/C++ counterparts, programmers need to know enough CUDA C/C++ to write the kernels themselves, preventing the application from being written only in Python.

Numba Python [10] enables programmers to gain the benefits of a compiler while programming in nearly pure Python. Available as a Python library, Numba is a just-in-time compiler that enables developers to add special decorators to their Python applications to target specific sections of code for compilation. Additionally, Numba is able to create compiled code on the GPU, allowing developers a way to benefit from Python's features while also maintaining fast CPU/GPU code. Prior work [11], [12] demonstrates that Numba performs competitively to C in several classes of GPU applications.

In this paper, we evaluate Numba's ability to perform historical analysis of PMU data, by comparing it to traditional C and C CUDA on computing Linear State Estimation (LSE) and anomaly detection on synchrophasors. Software experiments were conducted to investigate performance, prove workflow, and identify strengths and limitations of each language.

PMUs increase measurement rates and enables measurements to be synchronized across the system using a GPS clock [13]. The time-aligned nature of the measurements they produce (synchrophasors) allows for linear solutions to the state estimation problem. While simplified, the required WLS

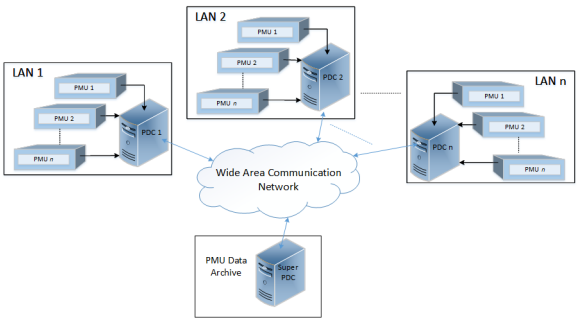

Fig. 1. Distributed PMU-based WAMS architecture

method is still is computationally intensive to benefit from GPU analysis, as it requires linear algebra on complex voltages and currents [14].

II. Methods

PMUs perform real-time monitoring of power grids and report data to a Phasor Data Concentrator (PDC) which processes and archives PMU data for other applications (e.g. historical analysis) as shown in Fig. 1. The LSE-anomaly detection workflow presented in this work can be applied to real-time WAMS [15] or historical analysis [16] of PMU data. Specifically, the anomaly detection technique processes PMU data for constraint violations and temporal anomalies, and is applicable to any level of PMU deployment, as outlined in [15]. LSE is applied to provide insight into PMU data matching the system model and is useful in determining if anomalous conditions are consistent with the system physics, or an artifact of the monitoring system.

Unlike real-time applications, which involve the analysis of smaller sets of PMU data with strict timing requirements, historical analysis requires the processing of large datasets with less strict timing requirements, though rapid analysis is still desirable. Additionally, energy efficient computing becomes ever more important as the quantity of PMU data, and subsequent offline PMU data analysis, continues to increase.

Depending on the application, the use of Single Board Computers (SBCs) can reduce the cost and power consumption of analyses that are typically performed on larger machines. For example, prior work [16], [17] has demonstrated the feasibility of real-time and historical anomaly detection of PMU data using decentralized SBCs at each PDC. Prior work [16] demonstrates that a CUDA-based historical anomaly detection approach running on the NVIDIA Jetson Nano SBC can detect anomalies in less than a second on a dataset containing 11.3 million PMU measurements. The feasibility of SBCs for centralized historical analysis is dependent on the required computational performance and dataset size.

For the approach in this paper, we augmented this prior work with a parallel version of a LSE described in [18]. The closed-form LSE solution for the WLS approach is:

$$x = (H^T W H)^{-1} H^T W z \tag{1}$$

where H is a matrix relating the states to measurements based on the power system model, W is a covariance weighted matrix based on measurement accuracy, z is a column matrix of time-aligned PMU measurements (consisting of voltages and current phasors), and x is the resulting column matrix of estimated power systems states (voltage phasors).

A key thrust of our work is to validate the workflow on a low-energy system, thus reducing the total energy required for the auditing process. To this end, we chose the NVIDIA Jetson Xavier NX [19], an SBC with a 6-core Carmel Arm CPU, a 384-core Volta GPU, and 8 GB of unified memory, as the platform of study. The NVIDIA Jetson Nano used in prior work [16] had insufficient memory for the LSE process. Most importantly, the Jetson Xavier NX features three optimized power budgets that limit the power consumption of the SBC to at most 20 Watts. In contrast, typical GPU cards alone consume upwards of 100 Watts, making the Jetson Xavier especially suitable for application spaces requiring a low-power footprint. The Jetson Xavier NX also features custom Tensor cores designed to accelerate deep learning applications that use TensorFlow [20], a popular ML package with a native Python API. These features make the Jetson Xavier NX a compelling platform for PMU-based applications, which may want to include machine learning as part of a larger workflow.

To this end, we explore two main research questions in this work. First, (a) *What benefit (if any) does Numba bring to historical analysis of synchrophasor data?*; and, (b) *What power budgets on the Jetson NX are recommended for historical analysis of sychrophasor data?*.

A. Implementation Details

To explore question (a), we implement a LSE anomaly detection workflow designed for historical analysis of synchrophasor data for two power systems: a 7-bus and 57-bus power system. The algorithm requires three steps: data preparation, state estimation, and anomaly detection.

Data preparation is done entirely on the CPU, and involves the creation of a large hashtable that collates the PMU data into their respective measurement type (e.g, voltage magnitude, current magnitude, etc.), which we refer to as measurands. This hashtable is then transcribed into a static $m \times n$ matrix, where m is the number of measurands, and n in the number of measurements per measurand. This $m \times n$ matrix is transferred to the GPU for linear state estimation and anomaly detection, which are then run on the matrix as separate kernels. In addition to the input matrix, the H, W, and z matrices required for LSE are transferred to the GPU.

At a high level, the LSE kernel assigns t threads to each of n/t blocks of the input matrix. Using a grid-stride loop [21], each thread computes (1) on its assigned set of measurements. To avoid race conditions and to ensure that there is a consistent state of the system, the pre-populated z matrix is used only for reading, while the resulting x matrix is only ever written to. At the conclusion of the LSE step, a separate detection kernel (described in [16]) is run on the data.

The entire LSE-anomaly detection workflow described above was separately implemented in C CUDA and Numba CUDA Python. All implementations use 32-bit floating point data types to hold measurement and estimated values. Each implementation required a custom LSE model that corresponded to the associated power system architecture. Thus, the 7-bus and 57-bus implementations have separate formulations of (1). From a computational perspective, it is notable that 57-bus workflow requires matrices that are an order of magnitude larger than those for the 7-bus workflow.

To explore question (b), we experimentally identified three power modes, out of the eight available on the Jetson Xavier NX, which runs JetPack v. 5.02-b321. In all cases we picked the mode that maximized CPU frequency as the data preparation step took a non-trivial amount of time. For the 10 Watt budget, mode 5 was utilized, the default for the Jetson Xavier NX. This power mode provides 4 online CPUs at 1900 MHz, but limits GPU core frequency to 510 MHz. For the 15-watt budget, mode 0 was selected, which provides 2 online CPUs at 1900 MHz, but allows for a GPU Max Frequency of 1100 MHz. Finally, for the 20-Watt budget, mode 6 was selected, which also provides 2 online CPUs at 1900 MHz and GPU Max Frequency of 1100 MHz. In this latter case, the higher power utilization goes partially to an increase in memory frequency, which increases to 1866 MHz from the 1600 MHz utilized on the smaller power budgets.

B. Datasets

We ran experiments on two sets of data. The first dataset is real data derived from a 1000:1 scaled emulated smart grid test bed [22] of a three-phase 46 kV subtransmission system with 7 buses, 9 transmission lines and 8 IEEE C37.118-compliant PMUs that are GPS time synchronized with a reporting rate of 60 Hz. Each 1-second data frame consists of 82 measurements (10 voltage magnitudes, 11 current magnitudes, 10 voltages phases, 11 current phases, 8 frequencies, 8 rate of change of frequencies, and 24 additional pieces of information). In the context of LSE, 14 states are estimated (magnitude and phase of each bus voltage).

The 7-bus dataset was collected over a period of 70 minutes, containing $\approx$ 21 million measurements. During this time, the system state was perturbed to generate anomalies. To test the efficacy of our historical analysis approach we tested it over the following collection periods:

- A 5-minute period ($\approx$ 1 million measurements)
- A 15-minute period ($\approx$ 6 million measurements)
- A 50-minute period ($\approx$ 15 million measurements)
- A 70-minute period ($\approx$ 21 million measurements)

The 57-bus dataset was synthesized in software to simulate the IEEE standard 57-bus power system that is an approximation of American Electric Power system in the Midwestern United States in the 1960s [23]. This system consists of 80 transmission lines and a PMU reporting rate of 60 Hz. A true state of this system was obtained by performing power flow; PMU data, with and without anomalies, was subsequently synthesized by adding random measurement error to the power flow result. Each 1-second data frame consists of 388 measurements (57 voltage magnitudes, 80 current magnitudes, 57 voltage phases, 80 current phases, 57 frequencies, 57 rate of change frequencies). In the context of LSE, 114 states are estimated (the magnitude and phase for each bus voltage). Data was simulated to obtain similar numbers of measurements as stated above, correspond to a 1-minute (1.5 million measurements), 4-minute (6 million measurements), 10-minute (15 million measurements) and 14-minute (21 million measurements) collection periods.

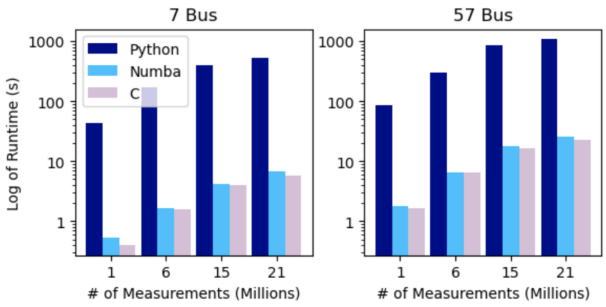

Fig. 2. Serial Performance

III. Results

For a particular dataset (7-bus or 57-bus), we varied the number of measurements (1 million, 6 million, 15 million or 21 million), and the implementation (Numba or CUDA). For the parallel benchmarks, we also varied the power budget (10 watt, 15 watt or 20 watt), and the number of threads (t) per block, which ranged from $t = 32 \ldots 768$. Each instance was run 5 times and results report the average time spent on the LSE and anomaly detection component. Memory consumption was also measured. The figures that follow depict the total time required for the state estimation and anomaly detection steps. Please note the hashtable building time is not included.

A. Serial Performance

For the 7-bus and 57-bus datasets, there was no appreciable difference in serial performance across the three power modes. For this reason, Fig. 2 shows the serial performance on the 7-bus and 57-bus datasets for pure Python, Numba Python, and C on the default 10W power envelope.

There is a significant performance difference between traditional "pure" Python and Numba Python. For example, the LSE+anomaly detection process implemented in pure Python takes 8.96 minutes on the 7-bus 21 million dataset, while Numba Python takes only 7.026 seconds, a speedup of 76.52. On the 57-bus dataset, the pure Python version takes 17.95 minutes to perform LSE+anomaly detection on 21 million measurements. In contrast, the Numba Python version takes only 25.111 seconds, a speedup of 42.88.

Consistent with prior work, the Numba Python serial implementations perform competitively to their serial C counterparts. For example, the C implementation took 5.718 seconds to perform LSE+anomaly detection on the 21 million

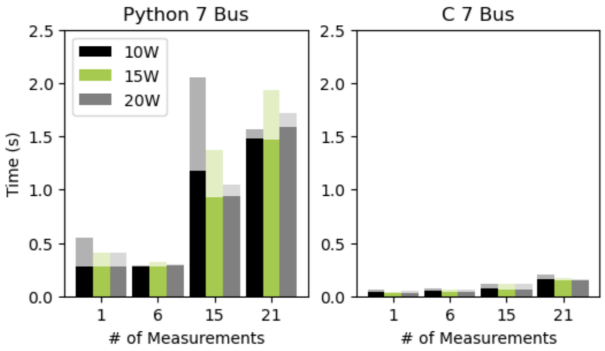

Fig. 3. 7-bus Dataset GPU performance

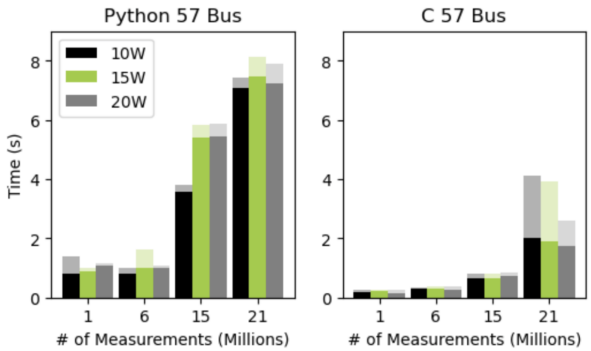

Fig. 4. 57-bus Dataset GPU Performance

measurements of the 7-bus dataset, roughly a one second difference. The performance difference narrows further on the 57-bus dataset, with the C version taking approximately 24.079 seconds, about a second faster than Numba. These results underscore Numba's ability to close the performance gap between serial Python applications and their C counterparts.

B. Parallel GPU performance

Any similarity in performance between Numba and C disappear when the approaches are run on the GPU. Fig. 3 and Fig. 4 depict the GPU performance of the Numba CUDA and C CUDA approaches on the 7-bus and 57-bus datasets respectively. We were unable to identify a particular number of threads that consistently produced the fastest runtime. As such, we depict the fastest (darker bars) and slowest (stacked lighter bars) observed parallel time on each power mode and measurement file. For the 7-bus dataset, the fastest times were consistently observed on the 15 W power envelope.

On the 7-bus datasets, it took Numba roughly 0.28 seconds to perform the LSE+anomaly detection process on 1 million and 6 million measurements, corresponding to respective speedups of 2.2 and 5.93 over the corresponding serial implementations. Numba takes 0.927 seconds on the 15 million measurements dataset and 1.446 seconds on the 21 million measurement dataset, or 4.7 times faster than their serial counterparts. In contrast, the C implementations are an order of magnitude faster. On 1 million measurements, the C implementation only takes 0.032 seconds to complete the LSE+anomaly detection process, a speedup of 13.84. On 21 million measurements, the C implementation takes only 0.144 seconds, a speedup of 40.82.

The speed difference between Numba and C is magnified on the 57-bus dataset. Unlike the 7-bus dataset, there was no consistency on which power envelope produced the fastest result. Therefore, in the discussion that follows, we refer only to the fastest observed time for each file. On the larger 15 million and 21 million measurement files, it takes Numba 3.559 seconds and 7.093 seconds to execute the LSE+anomaly detection process on the GPU, representing speedups of 5.04 and 3.54 over their serial counterparts respectively. In contrast, the C implementations took only 0.633 seconds to analyze 15 million measurements on the GPU, and 1.775 second to process 21 million measurements, representing respective speedups of 26.08 and 13.411 over the serial C counterparts. We also note that the C implementations are up to 5.62 times faster than their Numba counterparts on the 57-bus dataset.

Lastly, we note that the Python implementations use significantly more memory than their C counterparts. On the 15 and 21 million measurement files of both datasets, the Python implementations consume all the memory on the platform (8GB plus swap), significantly slowing down the execution of the Python implementations on these datasets. In contrast, the C versions comfortably fit into memory, with the 21 million (15 million) dataset consuming approximately 6.03 GB (5.29 GB) of memory. We believe this disparity in memory consumption is one of main reasons why there is such a significant performance difference between the Numba CUDA and C CUDA implementations.

IV. Discussion and Conclusion

For RQ1, there is compelling evidence that developers should look closer at Numba for speeding up serial applications of PMU data, especially those that heavily involve numerical computing. Compared to traditional Python, Numba's benefits are obvious: the LSE+anomaly detection process is infeasibly slow in traditional Python, but performs competitively to C when using Numba.

There is evidence that Numba has the potential to accelerate GPU compute tasks for PMU applications, allowing developers to create their applications completely in Python. The LSE+anomaly detection process requires nearly 18 minutes to analyze 21 million measurements in traditional Python; Numba CUDA analyzes the same dataset in < 8 seconds. While the equivalent CUDA C implementation is nearly 6 times faster than Numba, the 8 seconds needed for Numba CUDA is sufficiently fast for historical analysis of PMU data.

However, Numba's current limitations on optimizing certain critical data types can limit any advantage that Numba-based PMU applications may have over their C counterparts. Critically, Numba currently has very limited support for the Python dictionary data type, which is needed to build the initial

measurand hashtable prior to the LSE+anomaly detection process; again, table building was not included in our performance analysis above. Since this critical component had to be written in traditional Python for our Numba implementations, the table building process is an order of magnitude slower in the Numba implementations (327 s on the 7-bus, 680 s on the 57-bus) than their C counterparts (34 s on the 7-bus, 99 s on the 57-bus), eliminating any advantage that Numba may have for this specific application. While multithreading libraries like OpenMP can further reduce the table building preprocessing step in C, Python does not currently support multithreading, forcing this to remain a serial process for Python applications.

From a programming perspective, the serial and parallel historical analysis approaches were easier to implement in Numba Python than in C. However, optimization was significantly harder in Numba, as its abstractions made it difficult to optimize targeted sections of code. Furthermore, Numba's ecosystem of profilers and debuggers is still relatively immature; it is significantly more difficult to detect errors in a Numba implementation than in C. We also note that most of the observed performance differences in the GPU implementations in C and Numba had to do with the ability of the CUDA C implementation to leverage the unified memory architecture of the Jetson Xavier. The equivalent steps in Numba CUDA resulted in slower code, and therefore was omitted.

For RQ2, we had predicted that the higher power envelopes would yield significantly better performance on the NVIDIA Jetson NX. This assumption turned out to be false; the performance difference between power envelopes was negligible. Our results suggest that the 15W power envelope is a good choice for SCADA developers to achieve consistently good performance on the Jetson NX. Lastly, we believe the NVIDIA Jetson Xavier NX is an exciting platform for WAMS applications, and that adopting such power-efficient devices reduce the total energy consumption of the auditing process, making the process nearly "free" from an energy perspective, compared to traditional servers.

ACKNOWLEDGMENT

Funding for this work was provided by the Department of Defense (DOD). The views expressed in this article are those of the authors and do not reflect the official policy or position of the Department of the Army, DOD or the U.S. Government.

REFERENCES

[1] S. Schütte, S. Scherfke, and M. Tröschel, "Mosaik: A framework for modular simulation of active components in smart grids," in *2011 IEEE First International Workshop on Smart Grid Modeling and Simulation (SGMS)*, 2011, pp. 55–60.

[2] K. Anderson, J. Du, A. Narayan, and A. E. Gamal, "Gridspice: A distributed simulation platform for the smart grid," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2354–2363, 2014.

[3] L. Thurner, A. Scheidler, F. Schäfer, J.-H. Menke, J. Dollichon, F. Meier, S. Meinecke, and M. Braun, "Pandapower—an open-source python tool for convenient modeling, analysis, and optimization of electric power systems," *IEEE Transactions on Power Systems*, vol. 33, no. 6, pp. 6510–6521, 2018.

[4] H. Karimipour and V. Dinavahi, "Accelerated parallel WLS state estimation for large-scale power systems on GPU," in *2013 North American Power Symposium (NAPS)*, Sep. 2013, pp. 1–6.

[5] Z. Li, V. D. Donde, J.-C. Tournier, and F. Yang, "On limitations of traditional multi-core and potential of many-core processing architectures for sparse linear solvers used in large-scale power system applications," in *2011 IEEE Power and Energy Society General Meeting*, 2011, pp. 1–8.

[6] Y. Xia, Y. Chen, Z. Ren, S. Huang, M. Wang, and M. Lin, "State estimation for large-scale power systems based on hybrid CPU-GPU platform," in *2017 IEEE Conference on Energy Internet and Energy System Integration (EI2)*, Nov. 2017, pp. 1–6.

[7] C. V. Zabala-Oseguera, A. Ramos-Paz, and C. R. Fuerte-Esquivel, "Parallelization of The Two-Stage State Estimation Method Using GPU-Based Parallel Computing," in *2020 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*, vol. 4, Nov. 2020, pp. 1–6.

[8] NVIDIA, "CUDA python documentation - overview," https://nvidia.github.io/cuda-python/overview.html, accessed: 2023-06-15.

[9] R. Okuta, Y. Unno, D. Nishino, H. Shohei, and C. Loomis, "Cupy: A numpy-compatible library for NVIDIA GPU calculations," in *31st Conference on Neural Information Processing Systems (NIPS 2017)*. Neural Information Processing Systems, 2017.

[10] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A llvm-based python jit compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, ser. LLVM '15. New York, NY, USA: Association for Computing Machinery, 2015.

[11] L. Oden, "Lessons learned from comparing c-cuda and python-numba for gpu-computing," in *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2020, pp. 216–223.

[12] D. Di Domenico, G. G. H. Cavalheiro, and J. V. F. Lima, "Nas parallel benchmark kernels with python: A performance and programming effort analysis focusing on gpus," in *2022 30th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2022, pp. 26–33.

[13] Y.-F. Huang, S. Werner, J. Huang, N. Kashyap, and V. Gupta, "State Estimation in Electric Power Grids: Meeting New Challenges Presented by the Requirements of the Future Grid," *IEEE Signal Processing Magazine*, vol. 29, no. 5, pp. 33–43, Sep. 2012.

[14] A. S. Dobakhshari, S. Azizi, M. Paolone, and V. Terzija, "Ultra Fast Linear State Estimation Utilizing SCADA Measurements," *IEEE Transactions on Power Systems*, vol. 34, no. 4, pp. 2622–2631, Jul. 2019.

[15] S. J. Matthews and A. St. Leger, "Leveraging mapreduce and synchrophasors for real-time anomaly detection in the smart grid," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 3, pp. 392–403, 2019.

[16] S. J. Matthews and A. S. Leger, "Energy-efficient analysis of synchrophasor data using the nvidia jetson nano," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, 2020, pp. 1–7.

[17] S. J. Matthews and A. St. Leger, "Leveraging single board computers for anomaly detection in the smart grid," in *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, Oct 2017, pp. 437–443.

[18] S. D. Hassak, A. St. Leger, H. Oh, and D. F. Opila, "Implementing a pmu based linear state estimator on a single board computer," in *2022 North American Power Symposium (NAPS)*, 2022, pp. 1–6.

[19] NVIDIA Edge Computing, "Jetson xavier nx series," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/, 2023, accessed: 2023-06-15.

[20] M. Abadi, "Tensorflow: Learning functions at scale," *SIGPLAN Not.*, vol. 51, no. 9, p. 1, sep 2016. [Online]. Available: https://doi.org/10.1145/3022670.2976746

[21] J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-purpose GPU programming*, 1st ed. Addison-Wesley Professional, 2010.

[22] A. St. Leger, J. Spruce, T. Banwell, and M. Collins, "Smart grid testbed for wide-area monitoring and control systems," in *2016 IEEE/PES Transmission and Distribution Conference and Exposition (T&D)*, 2016, pp. 1–5.

[23] IEEE 57-bus system. Illinois Center for a Smarter Electric Grid (ICSEG). [Online]. Available: https://icseg.iti.illinois.edu/ieee-57-bus-system/