Energy-Efficient Analysis of Synchrophasor Data using the NVIDIA Jetson Nano

Suzanne J. Matthews
Electrical Engineering & Computer Science
United States Military Academy
West Point, NY USA
suzanne.matthews@westpoint.edu

Aaron St. Leger
Electrical Engineering & Computer Science
United States Military Academy
West Point, NY USA
aaron.stleger@westpoint.edu

***Abstract*—Smart Grid Technology is an important part of increasing resilience and reliability of power grids. Applying Phasor Measurement Units (PMUs) to obtain synchronized phasor measurements, or synchrophasors, provides more detailed, higher fidelity data that can enhance situational awareness by rapidly detecting anomalous conditions. However, sample rates of PMUs are up to three orders of magnitude faster than traditional telemetry, resulting in large datasets that require novel computing methods to process the data quickly and efficiently. This work aims to improve calculation speed and energy efficiency of anomaly detection by leveraging manycore computing on a NVIDIA Jetson Nano. This work translates an existing PMU anomaly detection scheme into a novel GPU-compute algorithm and compares the computational performance and energy efficiency of the GPU approach to serial and multicore CPU methods. The GPU algorithm was benchmarked on a real dataset of 11.3 million measurements derived from 8 PMUs from a 1:1000 scale emulation of a power grid, and two additional datasets derived from the original dataset. Results show that the GPU detection scheme is up to 51.91 times faster than the serial method, and over 13 times faster than the multicore method. Additionally, the GPU approach exhibits up to 92.3% run-time energy reduction compared to serial method and 78.4% reduction compared to the multicore approach.**

***Index Terms*—big data applications, manycore computing, GPGPU, single board computers, phasor measurement units, power system analysis computing, smart grids**

I. Introduction

Power grids have historically been susceptible to large-scale blackouts resulting from cascading failures. While rare, the impact of large-scale blackouts can be severe. Blackouts arise due to numerous factors such as equipment failures, operational errors, cyber-attacks, physical attacks on the power grid, or a combination thereof. One blackout incident in North America, the 2003 Northeast Blackout, occurred due to stress on the grid infrastructure and ineffective monitoring and analysis of the grid itself. This incident left approximately 50 million people in the Northeastern United States and Canada without power [1]. Subsequent analysis showed that more sophisticated monitoring and better situational awareness would have helped prevent this blackout [2].

Synchronized phasor measurments, or synchrophasors, are obtained by Phasor Measurement Units (PMUs). PMU implementation into Wide Area Monitoring Systems (WAMS) show promise in improving the situation awareness of power grids. Phasor Measurement Units (PMUs) provide data sample rates and measurement fidelity far greater than traditional methods. However, a challenge of PMUs is the large data that is produced. For example, PMUs operating at a sampling rate of 60 HZ collect data every 16.67 ms. Efficient algorithms are therefore required to quickly analyze synchrophasor data.

Generally, synchrophasor analysis can be classified into two categories: real-time or near real-time to support operation of the power grid, and offline analysis of historical PMU data to provide further insight into power system operation. Both categories require computational methods and algorithms that can process large amounts of data quickly. Real-time applications process less data but have stricter timing requirements while historical data analysis process significantly larger data sets with less strict timing requirements.

Recent research has shown promise in applying single board computers (SBCs) to the problem of detecting anomalies in synchrophasor data [3]–[5], an important component of WAMS. SBCs have several advantages over traditional computing platforms, including reduced cost, lower power and energy consumption, and smaller form factor. When deployed in an edge computing framework, SBCs are capable of analyzing data close to the PMU, resulting in lower network requirements, energy savings, and the ability to better match the computing hardware to the specific application at hand.

A key performance limitation of most modern SBCs lies with the ARM-based system-on-a-chip (SoC) that is prevalent on such systems, which make it near impossible to achieve real-time historical analysis of sychrophasor data. This paper is novel for two reasons. First, it presents a GPU anomaly detection scheme for historical synchrophasor analysis. Second, it demonstrates how the 128-core NVIDIA Jetson Nano SBC can analyze millions of synchrophasor measurements in less than six seconds while consuming less than 10 Watts of power. Our results highlight the enormous potential of manycore SBCs for centralized WAMS anomaly detection applications.

The rest of the paper is organized as follows. Section II provides background information and discussion of related work. Section III provides an overview of the GPU algorithm. Experimental setup and results are in Section IV and V respectively, and concluding remarks are in Section VI.

DOD High Performance Computing Modernization Program

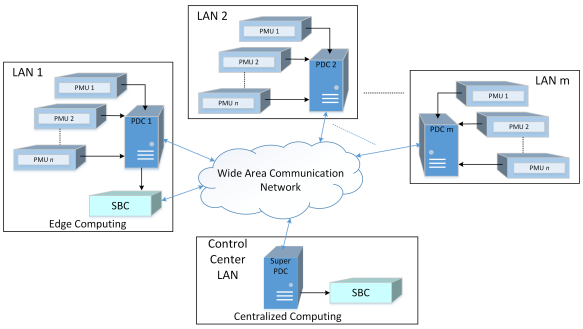

Fig. 1: PMU based wide area monitoring system with SBC edge computing

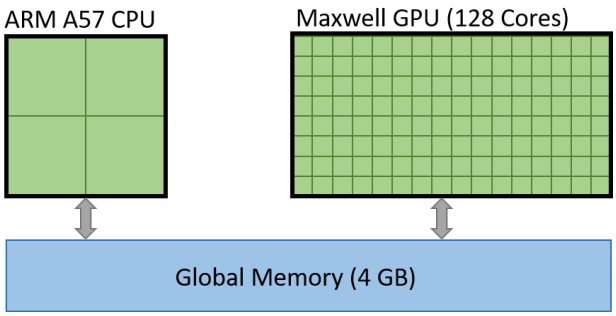

Fig. 2: Architectural Layout of Jetson Nano

II. Background & Related Work

PMU implementation is guided by the IEEE C37.118 Synchrophasor Standard [6] which defines the performance standards, and protocols for PMUs. PMUs are more advanced than traditional asynchronous sensors. They are time synchronized via the global positioning system (GPS) and provide time stamped measurements of voltage phasors, current phasors, frequency, and rate of change of frequency in the power grid. Reporting rates are defined by the standard between 10-60 Hz for a 60 Hz power system [6], three orders of magnitude faster than traditional telemetry, and provide the ability to more rapidly detect and respond to anomalies. However, significant communication and computing infrastructure, and computational algorithms and techniques, are required to handle the resulting large data in a timely manner.

PMU based WAMS can provide faster and more detailed monitoring. WAMS require networked sensors and computing devices integrated throughout the grid (see Figure 1 for an example). PMUs provide synchrophasors to the Phasor Data Concentrators (PDC), which archive and serve data. For centralized computing at the control center, all data must be sent through the wide area communication network. As an alternative, data is transferred across a local area network to a computing platform to perform analysis and send the results through the wide area network. The primary trade-off for edge vs. centralized computing is the quantity of data to be analyzed and the network requirements to transfer the data.

Real-time PMU data analysis supports operation of the power grid while offline analysis of historical PMU data provides insight into power system operation. Both categories require computational methods and algorithms that can process large amounts of data quickly. Real-time applications process less data, but have strict timing requirements while historical data analysis process larger data sets with less strict timing requirements. The anomaly detection approach presented in this work is based on earlier work [7] that performed constraint and temporal analysis of historical PMU data on a standard multicore server. For a particular type of measurand (e.g. voltage, current, etc.) there is a window of allowed variation. A measurement that falls outside its window of allowed variation is classified as a *constraint* anomaly [7]. Rapid fluctuations that occur inside the window of allowed variation are often indicators of anomalous behavior; these are classified as *temporal* anomalies [7] and are detected using a metric known as the Fano factor [7].

A concept of a WAMS, with single board computers for both edge and centralized computing, is shown in Figure 1. Compared to traditional computers, SBCs are inexpensive and power-efficient. Prior work demonstrates the efficacy of applying a Raspberry Pi SBC [3], [4] in an edge-computing context for the real-time detection of constraint and temporal anomalies. The Raspberry Pi approach resulted in real-time speeds at significant energy savings. An SBC cluster was explored for centralized, historical anomaly detection in [5] and showed some promise. While cost and power consumption compared very favorably to multicore servers, the cluster's data transfer and memory bottlenecks limited its usefulness.

GPU (or manycore) computing holds significant promise for accelerating the anomaly detection process. Prior research has shown the applicability of manycore computing for power systems applications, such as non-linear AC Power flow analysis in transmission systems (Newton-Raphson) [8] and radial distribution systems (forward backward sweep) [9]. Performance of GPU compute on power flow and state estimation were presented in [10]–[12]. These works showed substantial improvements by leveraging GPU compute. The work in [13], [14] specifically highlighted the needs and motivations of High Performance Computing (HPC) to overcome the computational challenges of WAMS, and other, power systems applications. Applying GPU compute to AD in state estimation [15], security assessment [16], and distribution system optimization [17], showed improved performance.

Historically, GPU computing was not possible on most single board computers. If an SBC had a GPU, it was integrated into the system-on-a-chip, and was too small to perform anything beyond rudimentary graphics processing. The NVIDIA Jetson TK1, and subsequent Jetson TX1 and TX2 were the first single board computers with CUDA-enabled cores. However, these early SBCs had a large form-factor, and cost as much as $599.00. In 2019, NVIDIA released the Jetson Nano, a $99.00 SBC that is roughly the size of two credit cards with a maximum power consumption of 10 Watts. Figure 2 depicts the architecture of the system. The Jetson Nano features a

compute capability 5.3 Maxwell GPU (the "device") with one shared-memory multiprocessor (SMM) consisting of 128 CUDA cores. The board also features a quad-core ARM A57 "host" CPU @ 1.5 GHz and a single 4 GB bank of unified memory that is shared between the the host CPU and the GPU.

Our work is *novel* for two primary reasons. First we show the applicability of the NVIDIA Jetson Nano for fast historical analysis of synchrophasor data. Second, we present a novel GPU algorithm for performing anomaly detection in a centralized context. Our results support the use of the NVIDIA Jetson Nano for efficient centralized synchrophasor analysis in WAMS systems.

III. Overview of Algorithm

In designing a GPU algorithm for analyzing synchrophasor data, there are two main challenges. First, for real data gathered from actual PMUs, there is no guarantee that that when the instrument stops measuring, that there are an equal number of synchrophasor measurements collected from each PMU. Secondly (and more importantly) the time-aligned nature of synchrophasor data necessitates time-series dependencies between the elements. GPUs work the most optimally when each element can be operated on independently. Therefore, special care must be taken in the algorithmic design.

Suppose r is the number of unique measurands, or physical properties to be measured (e.g. each bus voltage is a measurand), captured in the time file. Each measurand r_i has an associated time series of measurements (numerical values). Let n_i be the number of measurements associated with measurand r_i. To simplifying indexing, we create an input measurement matrix of size $r \times N$, where $N > n_i$, $\forall n_i$. The remaining $N - n_i$ elements associated with each r_i in the matrix are 0s.

A second $r \times 5$ matrix stores the requisite parameters for analysis. For a particular measurand r_i, its row includes the total number of associated measurements n_i, its constraint boundaries (c_{imin}, c_{imax}), normalization factor, and maximum allowed Fano factor f_i.

Prior to the kernel run, the $r \times N$ and $r \times 5$ input matrices are transferred to the GPU, along with an 0-initialized $r \times N$ output anomaly matrix. The GPU is responsible for populating the anomaly matrix; for any measurement (i, j) in the input matrix, the GPU places a 1 in the corresponding (i, j) location of the anomaly matrix if a constraint anomaly is detected, and 2 if a temporal anomaly is detected. If no anomalies are detected, the (i, j) location in the anomaly matrix remains 0.

We initialize the kernel with t threads and $(r \times N)/t$ blocks. Thus, each thread is responsible for computing a single element in the anomaly matrix. For memory-efficiency, the matrix is passed to the GPU as a single-dimension array.

Algorithm 1 depicts pseudo-code for the GPU detect function. The input is a matrix (M) of measurements, while the output is a matrix (A) of detected anomalies. Every element is handled by a separate thread; when $r \times N$ exceeds the maximum allowable blocks and threads on the GPU, a grid-stride loop [18] is employed.

Algorithm 1 Detect Algorithm Run by Each GPU Thread

```
1:  procedure DETECT(M, A)
2:      Get x;
3:      while x < r × N do                          ▷ if x in range do:
4:          i ← x/N
5:          j ← x mod N
6:          Look up n_i, c_imin, c_imax, and f_i
7:          if j < n_i then                          ▷ If valid element
8:              if x is a constraint anomaly then
9:                  A[x] ← 1                         ▷ constraint anomaly
10:             else
11:                 Check (x − HZ . . . x) for constraint anom.
12:                 if no constraint anomalies then
13:                     compute Fano factor (x − HZ . . . x)
14:                     if Fano factor is greater than f_i then
15:                         A[x] ← 2                 ▷ temporal anomaly
16:                     end if
17:                 end if
18:             end if
19:         end if
20:         Calculate new x
21:     end while
22: end procedure
```

The algorithm first derives the (i, j) position of measurement x in matrix M by setting $i = x/N$ and $j = x \mod N$. This identifies the measurement as being the j^{th} measurement associated with measurand i. After ensuring that x is a valid measurement, the algorithm performs a constraint check on x. If x is a constraint anomaly, the output matrix at location $A[x]$ is updated with the value 1. If x falls within the bounds of allowable variation, the algorithm next checks the previous HZ elements for constraint anomalies. If the entire range is clear, the Fano factor for that range is computed. If the computed Fano factor exceeds the Fano factor limit f_i, then the anomaly matrix at position $A[x]$ is updated with value 2.

Once the kernel finishes execution, the final anomaly matrix is transferred back to the host for final post-processing and printing.

IV. Experimental Setup

We use three datasets to benchmark the performance of the GPU detection algorithm: a real dataset derived from actual PMUs, and three "processed" datasets that were generated from the real dataset to better study performance. All benchmarks were run on the NVIDIA Jetson Nano SBC.

The real data is derived from the USMA Smart Grid Testbed [3], [19], a 1:1000 scaled version of 46kV power grid that consists of eight real PMUs, seven buses and nine transmission lines. Each PMU is configured to sample at 60 HZ and outputs new synchrophasor data ever 16.67 ms. The data capture in each time unit consists of standard quantities as outlined by the IEEE synchrophasor standard [6], including voltage phasors, current phasors, and the system frequency. Synchrophasor data is transferred to a Phasor Data

Concentrator (PDC) via TCP/IP. The PDC then time aligns and stores the data in a MySQL database. The real dataset used in the experiments in this paper were derived from a 87-minute run and consists of roughly 11.3 million measurements.

The processed datasets are generated using two different scripts. Suppose the original dataset has p PMUs and m measurements. The first script simulates $p \times x$ PMUs over a constant time period (in this case, 87 minutes) by replicating the dataset x times and assigning the new measurements to new PMU measurand identifiers. The final dataset then contains $p \times x$ total PMUs and $m \times x$ total measurements. Increasing the number of PMUs in the dataset is more representative of a centralized computing historical analysis case.

The second script scales up the number of PMUs by a factor of x, while holding the number of measurements constant. It accomplishes this by generating $p \times x$ PMUs and assigning a subset of m/x measurements to each. This script serves to simulate a "real-time" scenario [7], where analysis is being performed on a large number of PMUs over a small time period (52 seconds).

V. Results

Figures 3-8 depict the performance results. For each dataset, the GPU algorithm was benchmarked from 8 to 512 threads in powers of 2, as the fastest time is not always achieved at the highest number of GPU cores. For each thread-dataset combination, the GPU program was benchmarked 5 times, and the average time is reported. The serial version of the algorithm was also benchmarked 5 times. Each graph depicts the average speedup ($S = \frac{T_{serial}}{T_{parallel}}$) of the GPU approach over the serial algorithm.

As another point of reference, we also report the performance of a standard multicore implementation of the algorithm. The multicore version of the code is extremely straightforward. For r unique measurands and c cores, each thread simply performs the detection approach on the measurements of the r/c measurands assigned to it. We assign $c = 4$ since the ARM CPU on the Jetson Nano has 4 physical cores. Due to the embarrassingly parallel nature of the multicore approach, the measured speedup is always close to 4, and speedup is near-linear up to 4 cores.

A. Original dataset: 11.3 million measurements

Figure 3 shows the results of benchmarking on the the original dataset, consisting of 8 PMUs and 11.3 million measurements. The serial detection approach takes on average of 25.42 seconds to complete, while the multicore version takes 6.54 seconds. In contrast, the GPU implementation of the algorithm takes 0.50 seconds to execute on 64 threads, corresponding to a speedup of 50.95.

Next, we processed the 11.3 million dataset to contain 800 PMUs. This case emulates the scenario in which data is collected from 800 PMUs over a 52 second period. Figure 4 shows the result. The maximum speedup of 51.91 is achieved with 256 GPU threads, corresponding to an average run time of 0.48 seconds.

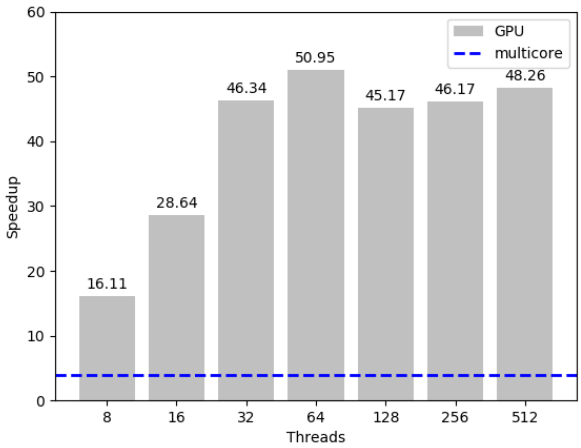

Fig. 3: Speedup Results on 8 PMUs, 11.3 Million Measurements

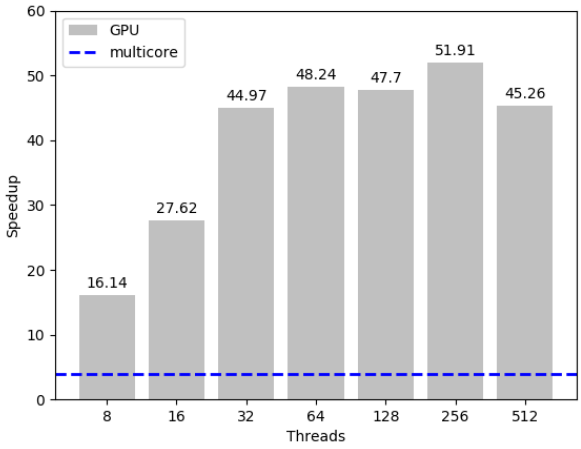

Fig. 4: Speedup Results on 800 PMUs, 11.3 Million Measurements

B. Large dataset: 33.9 million measurements

To study the performance on even larger datasets, we first processed the original 8 PMU dataset to represent a larger dataset of 24 PMUs consisting of measurements taken over a 87-minute collection period. This larger dataset contains 33.9 million measurements and is roughly 2.5 GB in size.

Figure 5 depicts the benchmarking results of the 24-PMU dataset. The GPU version of the algorithm took just 1.69 seconds to complete on 64 threads, compared to 76.49 seconds serially, a speedup of 45.09. It is worth noting that the GPU approach is also 11.5 times faster than the multicore approach on this same dataset, which averaged at 19.50 seconds.

Consistent with our first set of experiments, we next processed the 33.9 million measurement dataset to contain 2400 PMUs, emulating the scenario when 2400 PMUs were collecting data over a 52 second time frame. The results are shown

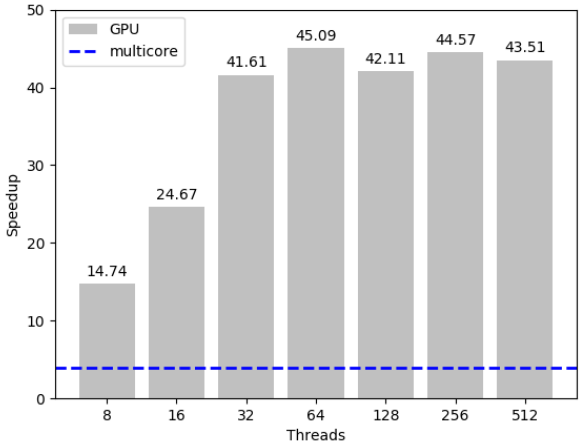

Fig. 5: Speedup Results on 24 PMUs, 33.9 Million Measurements

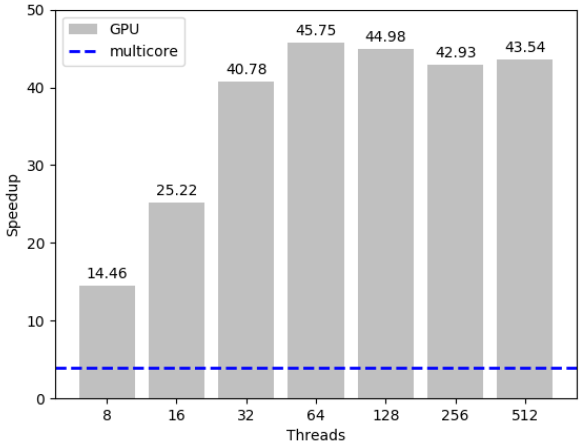

Fig. 6: Speedup Results on 2400 PMUs, 33.9 Million Measurements

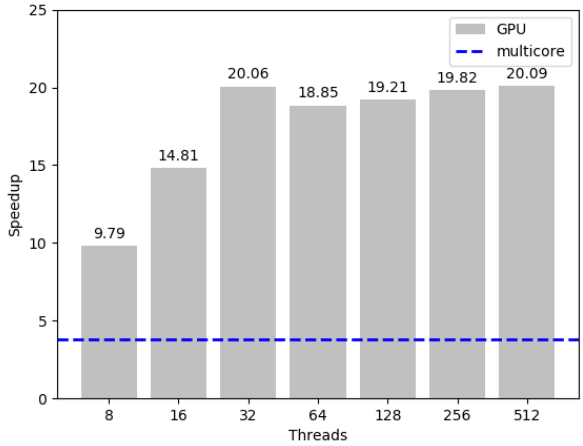

Fig. 7: Speedup Results on 32 PMUs, 45.2 Million Measurements

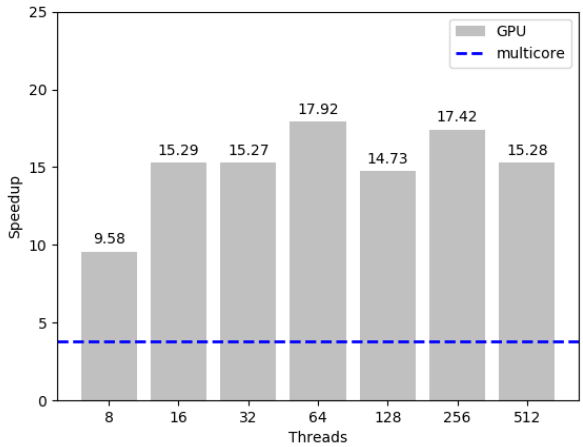

Fig. 8: Speedup Results on 3200 PMUs, 45.2 Million Measurements

in Figure 6. The maximum speedup of 45.75 was achieved on 64 threads, corresponding to a run time of 1.64 seconds. In contrast, the serial implementation took 74.87 seconds, while the multicore version took 19.15 seconds.

C. Memory intensive dataset: 45.2 million measurements

A key bottleneck discovered during benchmarking is memory. As previously mentioned, the NVIDIA Jetson Nano has 4 GB of unified Random Access Memory that is shared between the GPU and CPU. Roughly 1.2 GB of memory is used by the CPU for kernel-related processes.

In our prior benchmarks, our datasets fit within the available memory of the Jetson Nano. In this last set of benchmarks we process the real dataset to generate a new dataset representing measurements from 32 different PMUs over 87 minutes. This new dataset contains 45.2 million measurements (3.2 GB) and is the largest that would run on the Jetson Nano (any datasets of larger size were automatically killed by the kernel).

Figure 7 shows the benchmarking results of the 45.2 million measurement dataset. The speedup results on this dataset are considerably lower than the prior datasets, with the maximum speedup of 20.06 (5.07 seconds) achieved on 32-threads. It is worth noting however that the GPU approach is still up to 5.25 times faster than the multicore approach, which completed the detection process in 26.65 seconds.

Next, we scaled up the number PMUs in the memory-intensive set to 3200, emulating the scenario where 3200 PMUs are generating data over a 52-second period. The results are shown in Figure 8. Interestingly the speedup results are even *lower* in this set of experiments; the maximum speedup of 17.92 (5.58 seconds) was achieved on 64 GPU threads. However, even in this case, the GPU approach is

TABLE I: Peak Power Consumption (Watts)

	8	800	24	2400	32	3200
Serial	5.1	5	5.2	5.2	6.0	6.0
Multicore	7.9	7.8	7.9	7.9	8.2	8.2
GPU	7.1	7.1	8.5	8.5	9.3	9.4

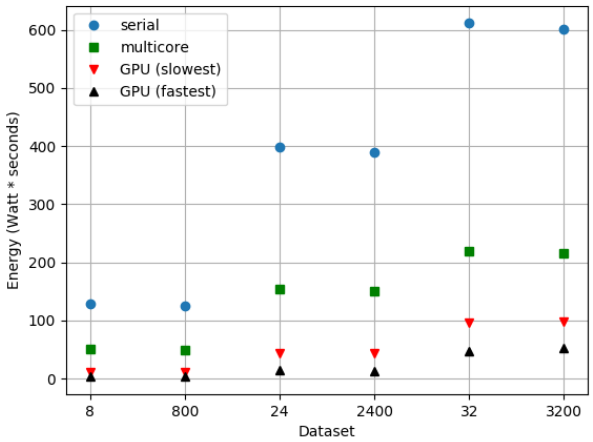

Fig. 9: Comparison of Energy Consumption

still 4.69 times faster than the multicore approach, which took 26.21 seconds on average.

D. Energy consumption analysis

We measured the power consumption of the NVIDIA Jetson Nano using a KillAWatt [20] electricity use monitor. Table I shows the peak power consumption of each algorithm measured over each dataset. The listed observations for the serial and multicore approaches are derived from the peak power consumption reading for each dataset over five runs. The listed observation for the GPU approach are derived from five runs of each dataset and each thread combination.

Larger datasets require more power than smaller datasets. The Jetson Nano SBC consumes approximately 3.8 Watts of power when idle, and between 5.0 to 6.0 Watts when running the serial approach. The multicore approach draws between 7.8 to 8.2 Watts of power. Lastly, the GPU power consumption spikes to 9.4 Watts on the largest datasets. All of these measurements are well within the NVIDIA Jetson Nano's power budget of 10 Watts, which is comparable to an LED light bulb.

Figure 9 compares the total energy consumed (in Watt-seconds) by each approach on the NVIDIA Jetson Nano. Given the fidelity, and lack of data logging of the KillAWatt, peak power values were used to estimate energy consumption. Specifically, energy was estimated by the product of peak power consumption and average run-time in seconds (Watt-seconds, or W-s). In addition to the serial and multicore approaches, Figure 9 also depicts the energy consumption of the slowest measured GPU approach (worst case) and the fastest measured GPU approach (best case) for each dataset.

Unsurprisingly, the serial dataset consumes the most energy, with up to 611 W-s being consumed during the benchmark of the 32-PMU dataset. The multicore approach uses significantly less energy, consuming up to 218 W-s on the 32-PMU dataset (64% energy reduction). However, the GPU approaches consistently use the least amount of energy, with the fastest approach using just 47.23 W-s of energy on the 32-PMU dataset (92.3% energy reduction). These results highlight the energy efficiency of the GPU algorithm when run on the NVIDIA Jetson Nano. Note that the GPU numbers represent a 78.4% reduction in energy from the multicore approach.

VI. Conclusion

Power grids require rapid anomaly detection approaches in order to maximize situational awareness and minimize the risk of power outages. While high-fidelity devices such as PMUs can increase the situational awareness of the grid, new computational methods must be developed to efficiently analyze the resulting glut of data. This paper presents a novel GPU algorithm for performing historical analysis of synchrophasor data and presents its efficacy on the recently released NVIDIA Jetson Nano single board computer.

Our results show that the Jetson Nano can analyze 33.9 million synchrophasor measurements in under 2 seconds for anomalies, which is 45.09 times faster than a CPU-bound serial approach, and 11.67 times faster than a multicore approach. While our experiments suggest that the Jetson Nano can analyze up to 45.2 million synchrophasor measurements, the size of this dataset causes the Jetson Nano to utilize quite a bit of its swap space, reducing the maximum speedup to 23.11. However, in all cases, the GPU algorithm is several times faster than multicore approach, and uses up to 92.3% less energy than the serial approach and up to 78.4% less energy than the multicore approach.

The 4 gigabytes of RAM on the Jetson Nano is a key performance bottleneck. This paper concentrates on the detection component of the anomaly detection process, the most time-intensive component. However, before anomaly detection occurs, the measurements must be read from the input file and aggregated into a global hashtable that organizes measurements in a time-synchronized manner by measurand. GPUs were not designed to build dynamic data structures like hashtables; the existing literature recommends that such structures be populated on the CPU [18]. We note that the Jetson Nano's 4 GB memory limit and parallel read bottlenecks of microSD cards limit the efficacy of a multicore hashtable building approach on the CPU. Future work will explore how to reduce hashtable build times on the Jetson Nano.

Acknowledgment

Funding for this project is provided by the DOD High Performance Computing Modernization Program (HPCMP) and the Army Futures Command. The views expressed in this article are those of the author and do not reflect the official policy or position of the Department of the Army, Department of Defense or the U.S. Government.

REFERENCES

[1] J. Chadwick, "How a smarter grid could have prevented the 2003 U.S. cascading blackout," in *2013 IEEE Power and Energy Conference at Illinois (PECI)*, 2016, pp. 65–71.

[2] A. Muir and L. J., "Final report on the August 14, 2003 blackout in the United States and Canada : causes and recommendations," Apr 2004.

[3] S. Drakontaidis, M. Stanchi, G. Glazer, J. Hussey, A. St. Leger, and S. J. Matthews, "Towards energy-proportional anomaly detection in the smart grid," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, 2018, pp. 1–7.

[4] 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)*, 2017, pp. 437–443.

[5] K. Candelario, C. Booth, A. St. Leger, and S. J. Matthews, "Investigating a raspberry pi cluster for detecting anomalies in the smart grid," in *2017 IEEE MIT Undergraduate Research Technology Conference (URTC)*, 2017, pp. 1–4.

[6] "IEEE standard for synchrophasor measurements for power systems," *IEEE Std C37.118.1-2011 (Revision of IEEE Std C37.118-2005)*, pp. 1–61, 2011.

[7] 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, first published in 2017.

[8] D. J. Sooknanan and A. Joshi, "GPU computing using CUDA in the deployment of smart grids," in *2016 SAI Computing Conference (SAI)*, 2016, pp. 1260–1266.

[9] D. Ablakovic, I. Dzafic, and S. Kecici, "Parallelization of radial three-phase distribution power flow using GPU," in *2012 3rd IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe)*, 2012, pp. 1–7.

[10] Z. Li, V. D. Donde, J. 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.

[11] H. Karimipour and V. Dinavahi, "Extended kalman filter-based parallel dynamic state estimation," *IEEE Transactions on Smart Grid*, vol. 6, no. 3, pp. 1539–1549, 2015.

[12] ——, "Parallel relaxation-based joint dynamic state estimation of large-scale power systems," *IET Generation, Transmission Distribution*, vol. 10, no. 2, pp. 452–459, 2016.

[13] R. C. Green, L. Wang, and M. Alam, "High performance computing for electric power systems: Applications and trends," in *2011 IEEE Power and Energy Society General Meeting*, 2011, pp. 1–8.

[14] ——, "Applications and trends of high performance computing for electric power systems: Focusing on smart grid," *IEEE Transactions on Smart Grid*, vol. 4, no. 2, pp. 922–931, 2013.

[15] H. Karimipour and H. Leung, "Relaxation-based anomaly detection in cyber-physical systems using ensemble kalman filter," *IET Cyber-Physical Systems: Theory Applications*, vol. 5, no. 1, pp. 49–58, 2020.

[16] D. Chen, H. Jiang, Y. Li, and D. Xu, "A two-layered parallel static security assessment for large-scale grids based on GPU," *IEEE Transactions on Smart Grid*, vol. 8, no. 3, pp. 1396–1405, 2017.

[17] E. Belič, N. Lukač, K. Deželak, B. Žalik, and G. Štumberger, "GPU-based online optimization of low voltage distribution network operation," *IEEE Transactions on Smart Grid*, vol. 8, no. 3, pp. 1460–1468, 2017.

[18] J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley, 2011.

[19] 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.

[20] P3 International, "Killawatt," Internet Website, last accessed 11/12/2019, 2018, http://www.p3international.com/products/p4400.html.