Optimization of Renewable Energy Resources for Large-Scale Power Grids

John Poirier
Department of Electrical and Computer Engineering, University of Vermont | *Advised by Dr. Samuel Chevalier*

Abstract

The integration of renewable energy into large-scale power grids brings with it significant challenges to resource management and dispatching. This research aims to address those challenges by developing knowledge in the realm of optimization and utilization of renewable energy resources. Resources must be distributed more efficiently, and security measures must be robust to ensure the continuous operation of the power grid. This approach uses nonlinear AC power flow equations in combination with machine learning models and numerical solution algorithms for optimization. Through the utilization of a parallelized Adam-based numerical solver, this research aims to overcome challenges of security and reserve-constrained AC Unit Commitment . This research tested several methods to go about solving energy optimization problems, utilizing both CPU and GPU methods. Single-Threading and Multi-Threading was used to test CPU optimization, and CuArrays and Kernels were used to test GPU optimization. This research found that utilizing the GPU provided significant advantages in speed, accessibility, and controllability. Compared to CPU optimization, GPU optimization allowed for exact control of what GPU resources are used in calculations, as well as a much greater capacity for parallel calculations.

Methods

CPU Single-Threading

QuasiGrad was operated on the CPU in a single-threading configuration. This configuration used only serial calculations. This method proved to be slow and inefficient.

CPU Multi-Threading

CPU multi-threading uses many threads of the CPU at one time. This method is limited by the machine that the code is being run on, as the power and number of threads in a CPU varies. This method allows the optimization to run on any machine and utilize the resources available effectively. Multi-threading provided significant improvements to speed and effectiveness of optimization.

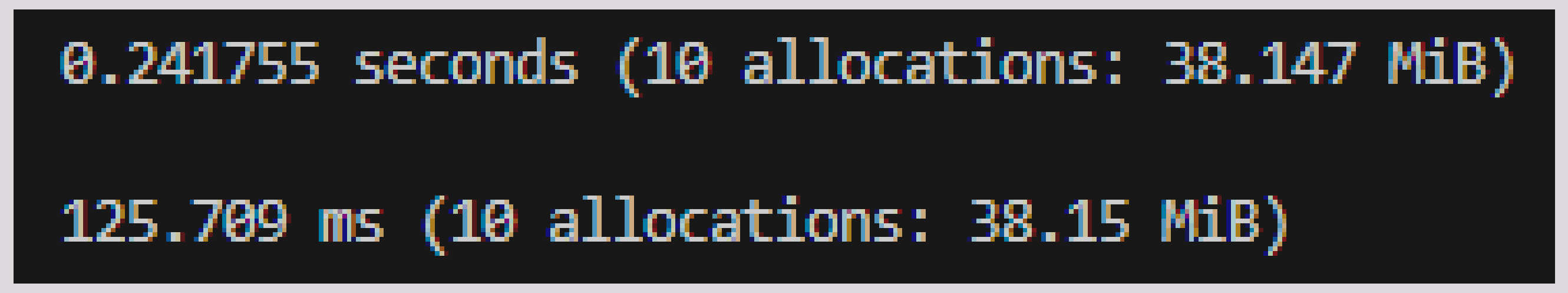
GPU

Operating on the GPU provides even further speed and efficiency increases, as well as greater control over resource allocation. GPUs contain multitudes more threads than CPUs, allowing for a greater capacity for parallelization. GPU power also varies by machine. To get the most out of every machine, we run a function that pulls the necessary device information, including number of threads, and use that information to automatically allocate resources appropriately. This method gave significantly improved efficiency to optimization.

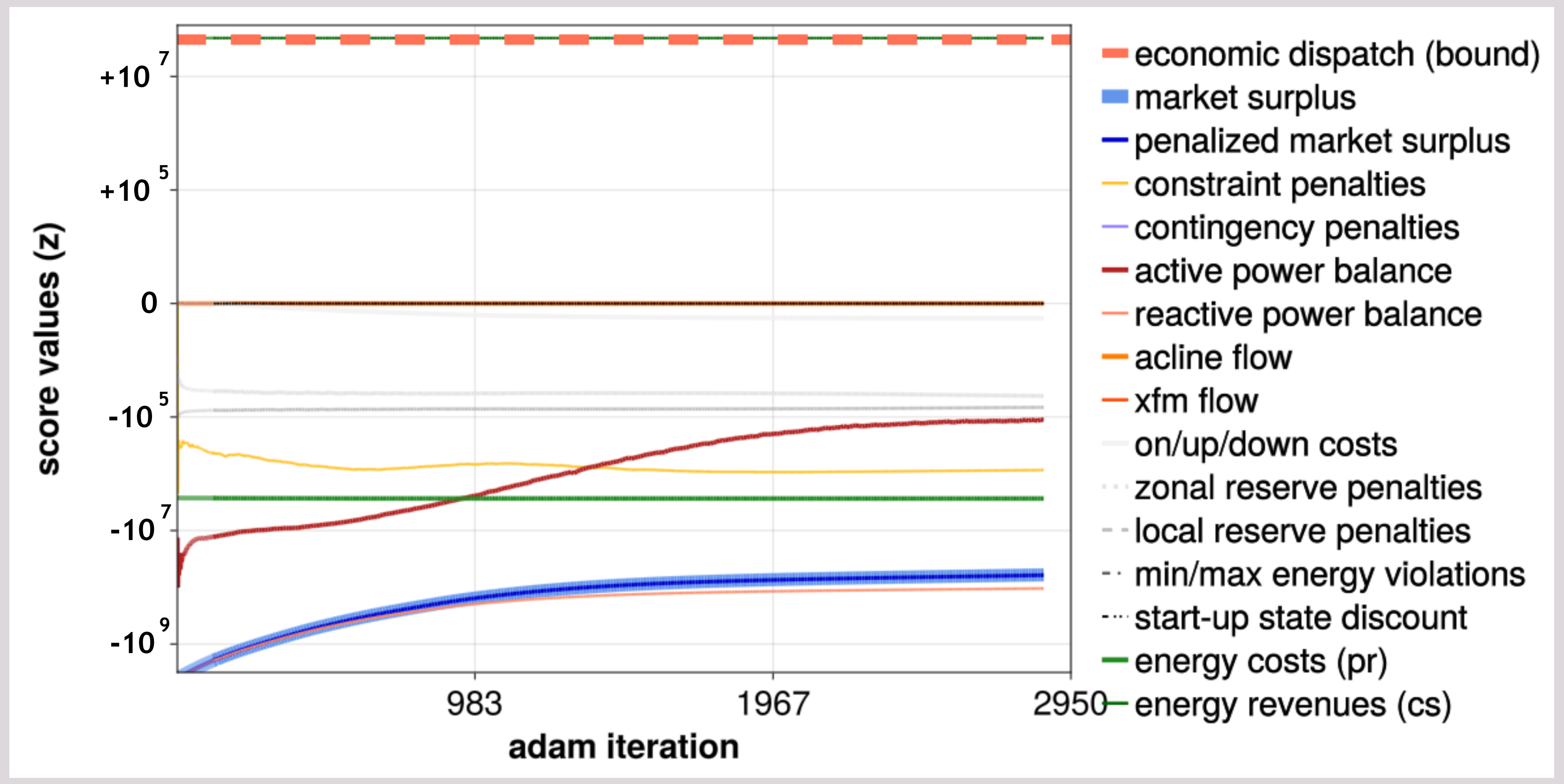
Results

Through many tests, we found that operating QuasiGrad on the GPU would provide the most benefits to speed, efficiency, and control. To test timing, we used large matrix-vector and matrix-matrix products for CPU single-threading, CPU multi-threading, and GPU operation. Shown below are the timings for each method, the first number being a single-run time and the second number being a 1000-run averaged time.

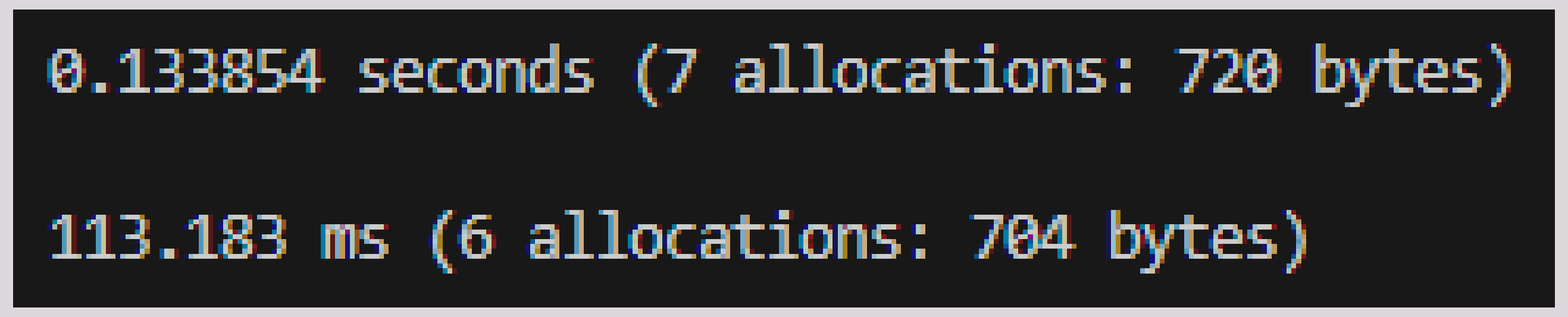
CPU Single-Threading



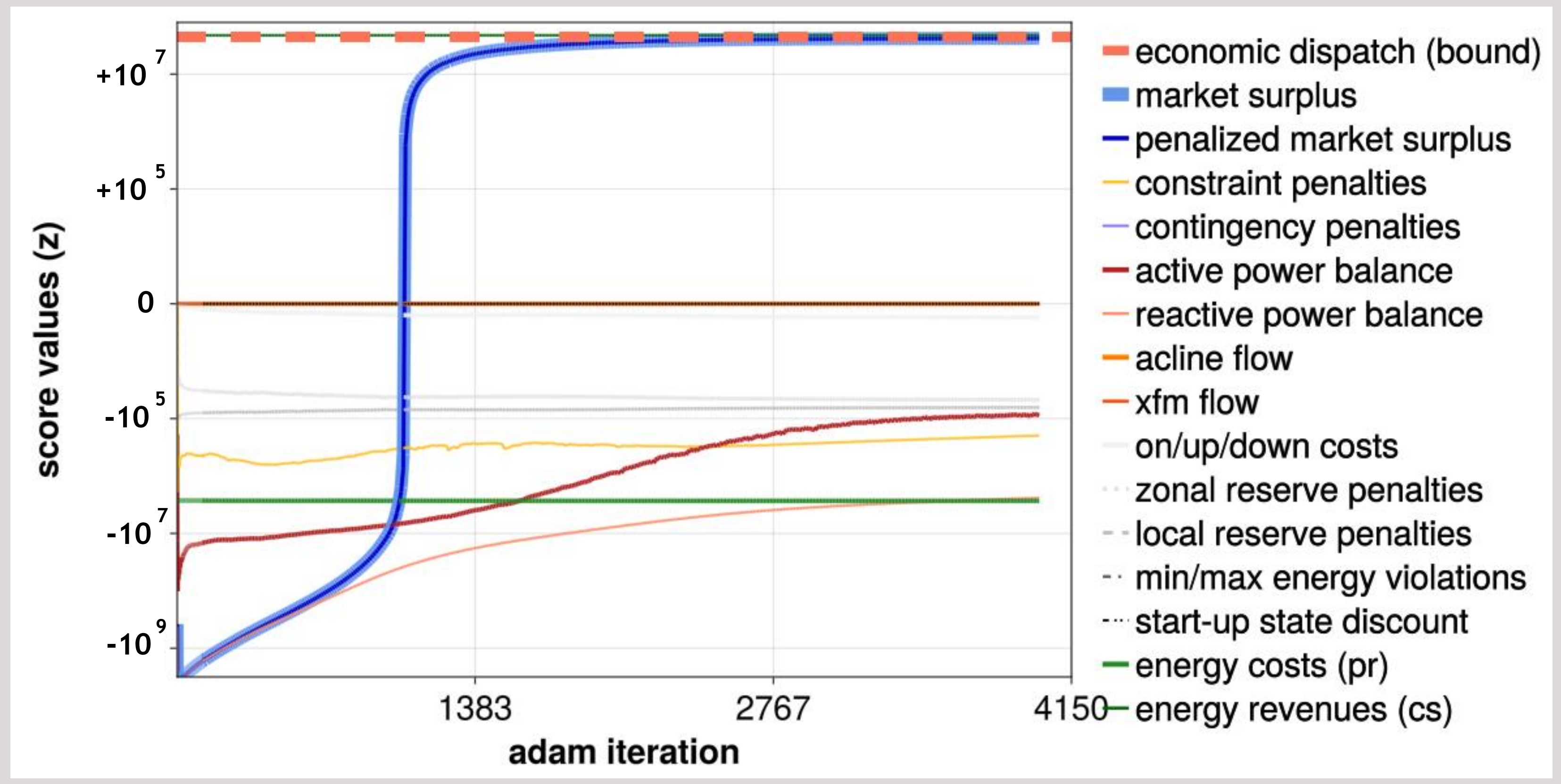
```
0.241755 seconds (10 allocations: 38.147 MiB)

125.709 ms (10 allocations: 38.15 MiB)
```



CPU Multi-Threading



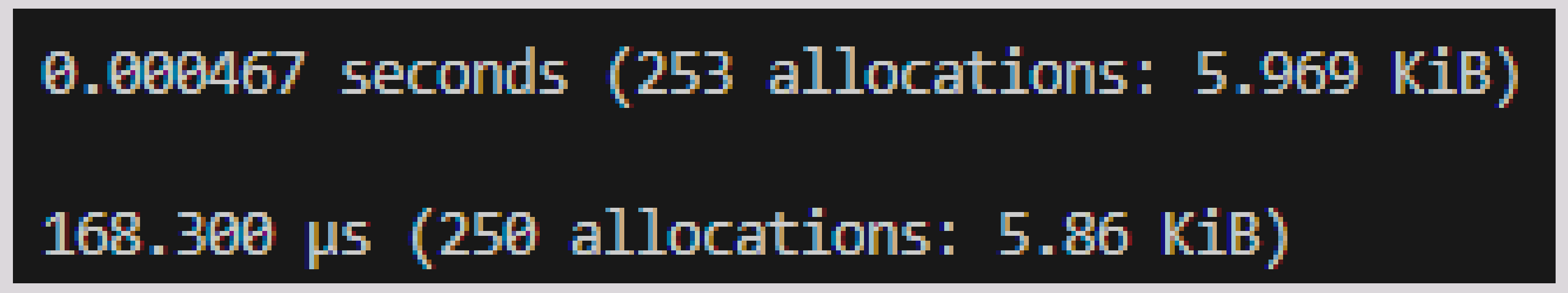
```
0.133854 seconds (7 allocations: 720 bytes)

113.183 ms (6 allocations: 704 bytes)
```



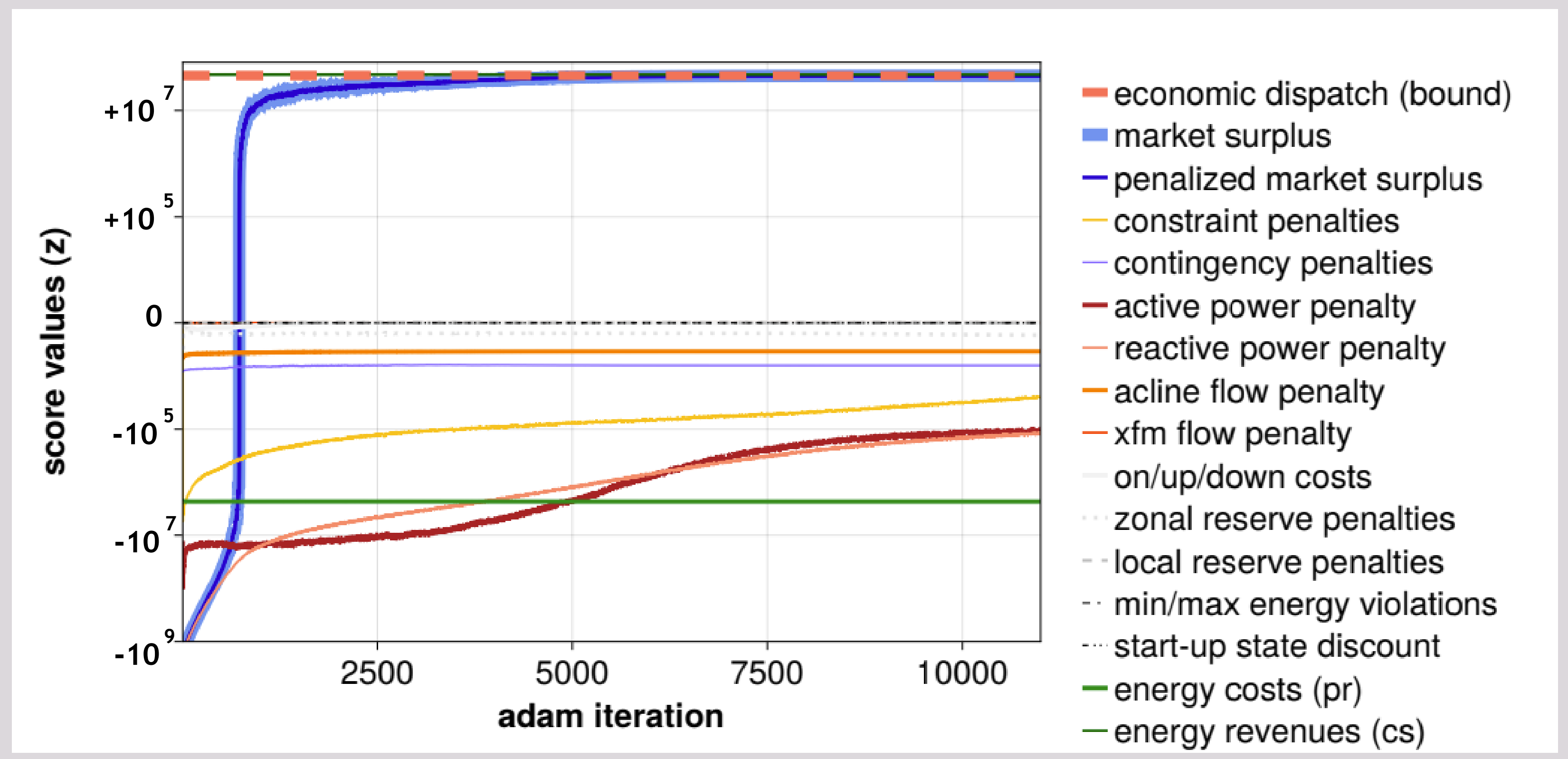
Conclusions

Operating on the GPU resulted in the greatest speed for the tests and had lower total allocation size that CPU single-threading, meaning it had to commit less information to memory. Thus, we concluded that converting QuasiGrad to operate on the GPU instead of the CPU would be beneficial. This would require transitioning all grid data to the GPU, then running calculations on the GPU rather than the CPU, then broadcasting back to the CPU to print results. This communication between hardware does slow the optimization process down, but the increased speed of calculations is significant enough for this to still be more effective. Further work on this research includes converting as much of QuasiGrad to operate on the GPU as possible, as well as testing this updated code with various simulated power grid conditions. Additionally, eliminating as much memory allocation and communication broadcasting as possible will further increase efficiency.

GPU



```
0.000467 seconds (253 allocations: 5.969 KiB)

168.300 μs (250 allocations: 5.86 KiB)
```



Acknowledgements

Samuel Chevalier, Department of Electrical and Computer Engineering

Drs. Luis Garcia, Donna Rizzo, and Mandar Dewoolkar

Graduate Students Ryan van der Heijden, Joy Onuh, and Shaurya Swami

The Richard Barrett Foundation

The University of Vermont College of Engineering, Mathematics, and Sciences