Computational and energy efficiency performance tracking of GridQube's Distribution State Estimator

28 March 2024 [Draft]

1 Background and Introduction

GridQube's vision for computation engine development is to be explainable, validated, auditable, as these are necessary conditions for decision support support systems in critical infrastructure. Accuracy and precision end up enabling customers in the energy transition, as better decisions lead to more uptake or solar, faster EV charging, higher reliability. Conversely, shortcuts in data and the mathematical models lead to ambiguity, opaqueness, and lack of explainability.

Detailed models come with trade-offs in computational resource use though. More detailed models need more data, and therefore require more memory, which in turn often leads to increased computational requirements as well.

Therefore, the goal of this initiative is to track the resource intensity of GridQube's model-based computational technologies, and to communicate about developments and improvements.

1.1 Advantages and disadvantages of model-free vs model-based

Relative to model-free approaches, model-based approaches have a number of advantages:

- no training needed, just initial data cleaning. training of model-free approaches is highly resource intensive.
- once set up and deployed, alerts can be set up for the model getting out of sync with reality in a straightforward manner thanks to the residuals
- can easily be validated for correctness

However, model-based approaches have some down-sides as well:

- data needs to be cleaned and semantics established

- depend on a mostly complete electrical model
- model-free approaches are less expensive to evaluate than model-based approaches.

2 Energy and performance tracking

We will describe different versions of our engine, and the functional differences, and then develop performance and resource use comparisons across versions.

2.1 Methodology

- We use compiled artifacts of milestone versions of core (0.15.0 and 0.20.0)
- Each artifact is deployed on testbed machine (AMD EPYC 7313 16-Core Processor, 32 threads total, L3 Cache 128MB)
- Three feeders models are used: small, medium and large EQL feeders (GLY15A, CMA9A, LBH1B)
- Network model and measurements are uploaded, calibrated and state estimation of 144 time steps is run based on historical data, 10 min intervals for duration of 24 hours using web API calls (common to all versions tested).
- We track CPU, memory and total time of GridQube backend process using psrecord library `https://pypi.org/project/psrecord/`
- Each case was evaluated 50 times to establish variance

2.1.1 Feeder specifications

Table 1 lists the network model sizes for the three feeders.

Table 1: Features of the feeders used on the study.

	GLY15A	CMA9A	LBH1B
# buses initially	350	424	1331
# buses after Kron's reduction	35	63	157
# MV/LV transformers	17	31	78
admittance matrix (nodes)	1400x1400	1696x1969	5324x5324
Jacobian matrix	313x280	543x504	1358x1256

2.2 Overview of Artifact Versions compared

2.2.1 0.15.0 vs 0.20.0

Summary of Differences

- improved docker base images
- improved data model - lines and line types - including support for phase-coordinate NxN matrices
- highly optimised Kron's reduction of admittance matrix using Intel PARDISO routines
- refactored math routines into separate library for more focused development, optimisation and unit testing
- UMFPACK and SuiteSparse integration: experimental solvers for large networks

Key Learnings

- Optimised Kron reduction allows for larger feeders to be ingested, analysed, cleaned and solved more readily.
- High memory usage can cause crashes, particularly if multiple large feeders need to be solved at the same time.

2.3 Further Comparisons between Intel MKL and AMD BLIS

AMD BLIS "is a high-performant implementation of the Basic Linear Algebra Subprograms (BLAS). The BLAS was designed to provide the essential kernels of matrix and vector computation and are the most commonly used and computationally intensive operations in dense numerical linear algebra."

Our core software can be configured to use AMD BLIS or Intel MKL for BLAS dense matrix computations. Performance testing was also carried out to compare MKL and BLIS.

Note: AMD libFLAME has not been fully optimised for all LAPACK functions. Therefore MKL was retained for LAPACK operations while BLIS was used at the BLAS level.

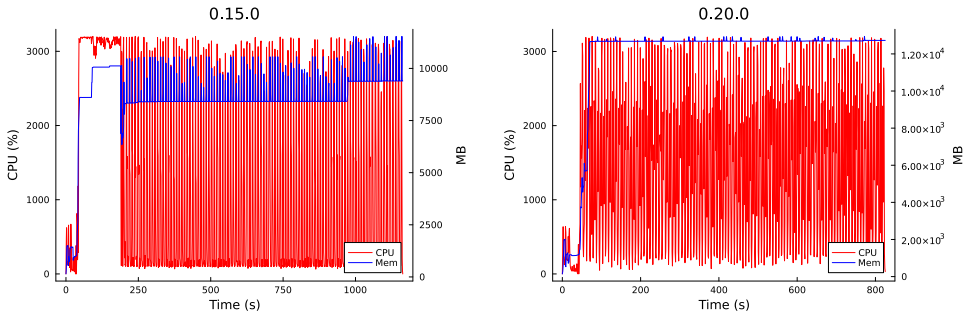
More information : `https://www.amd.com/en/developer/zen-software-studio/applications/spack/spack-aocl.html`

2.4 Results

2.4.1 Sample Run profiles - from upload to estimation

Below are sample run profiles of an estimation run with around 144 time steps on the larger LBH1B feeder. Note the high CPU utilization in 0.15.0 at the

beginning of the run up to roughly 200s. This is the unoptimised Kron reduction. Also note, the total run time is longer. More on this in later sections.



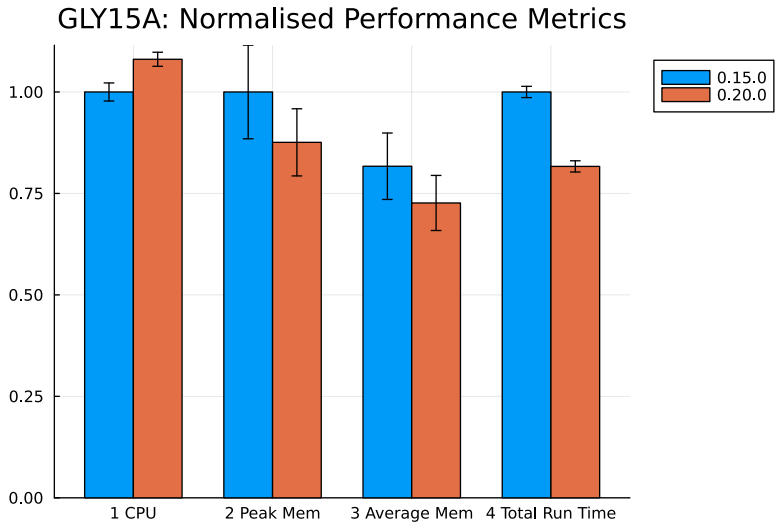
2.4.2 Summary Metrics, Mean (Std Dev)

Table 2 lists the observed computational aspects for the 3 feeders.

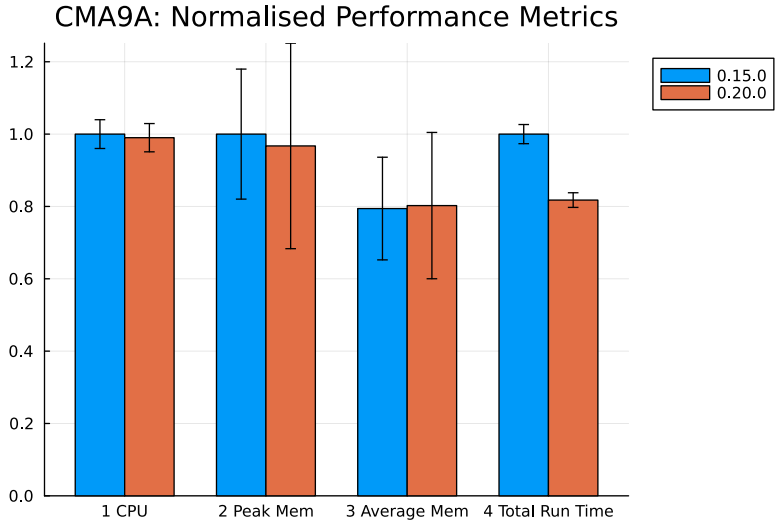
Table 2: Computational observations for the 3 feeders

	GLY15A	CMA9A	LBH1B
Total CPU Usage (%)	217k (3.5k)	337k (13k)	1664k (51k)
Peak Memory (MB)	2254 (211)	3010 (759)	8959 (2137)
Average Memory (MB)	2717 (257)	3629 (1066)	10920 (2686)
Total Time (s)	154 (2.6)	221 (5.5)	1179 (23)

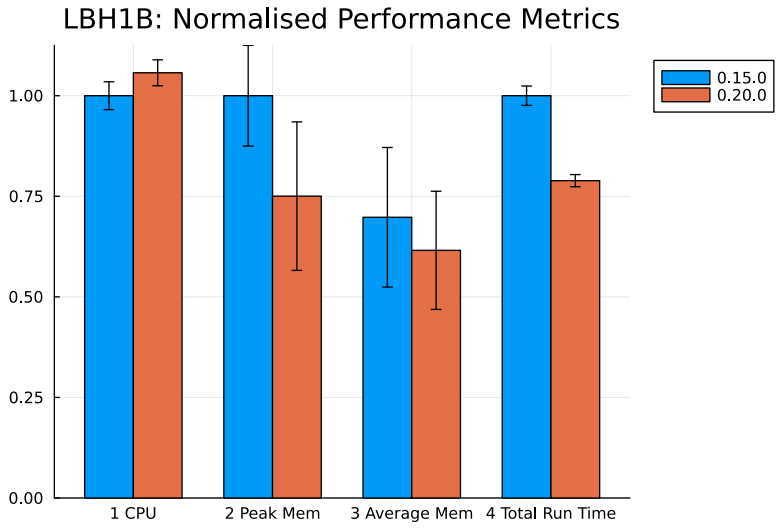
2.4.3 Small feeder - GLY15A - 17 transformers



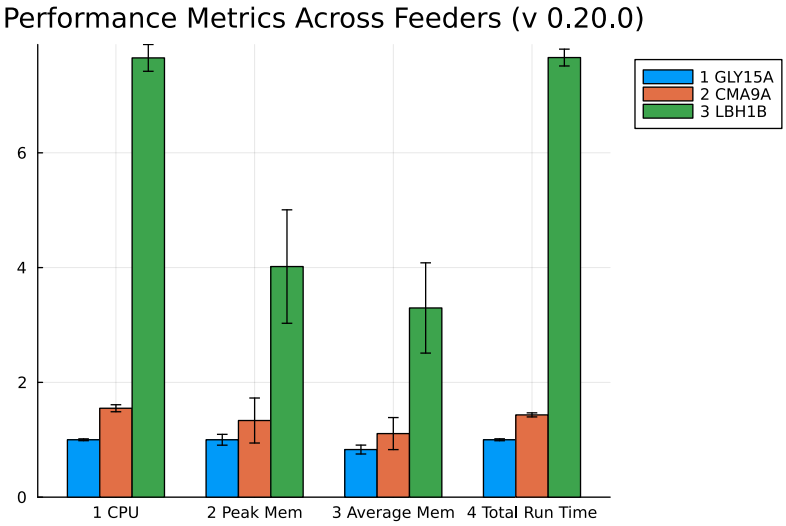
2.4.4 Medium feeder - CMA9A - 31 transformers



2.4.5 Large feeder - LBH1B - 78 transformers



2.4.6 Comparison across feeders for version 0.20.0



2.4.7 Comparison Across core Versions

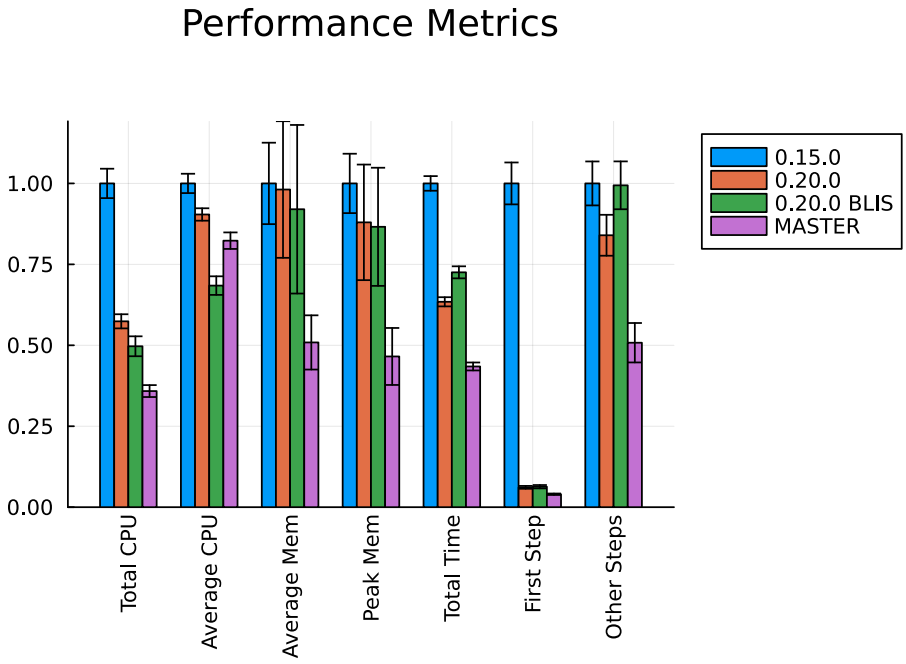
In this section we compare 4 different version of the core system: 0.15.0, 0.20.0 (default MKL), 0.20.0 with AMD BLIS and the latest version (MASTER) deployed on internal test systems. Experimental set up here was 36 time steps, 50 iterations of the LBH1B feeder. Table 3 lists the observed computational features.

Table 3: Comparison of computational features. Standard deviation in parentheses.

	0.15.0	0.20.0	0.20.0 BLIS	LATEST
Tot. CPU (%)	755k (34k)	433k (16k)	375k (23k)	350k (18k)
Avg. CPU (%)	1.4k (0.04k)	1.3k (0.03k)	0.98k (0.04k)	0.82k (0.03k)
Avg. Mem (GB)	7.44 (0.94)	7.30 (1.6)	6.85 (1.9)	3.45 (0.98)
Peak Mem (GB)	11.2 (1.0)	9.82 (2.0)	9.67 (2.0)	4.62 (1.1)
Tot. Time (s)	527 (12)	334 (7)	382 (10)	425 (13)
First Step (s)	159 (10)	9.74 (0.7)	10.1 (0.8)	11.5 (0.5)
Subs. Steps (s)	8.77 (0.6)	7.34 (0.6)	8.72 (0.6)	9.89 (0.7)

Notes:

- All metrics have been normalised against version 0.15.0.
- Total CPU % is the sum of instantaneous CPU % samples taken every second from the time of starting the core until finishing the estimation.
- Average CPU % is the Total CPU % divided by the total number of samples. This metric should be read in conjunction with the total time: a certain version may be more computationally intensive but finishes faster, while another version may have lower computation per unit time, but take longer to complete.
- "First step" and "Subsequent (abbreviated to subs.) steps" are the normalised times of the averages of the durations of the first estimation step and subsequent estimation steps respectively. Version 0.20.0 with MKL BLAS implementation is the fastest, but results indicate that 0.20.0 with AMD BLIS is more computationally efficient - although BLIS takes longer, the overall computational load is significantly less. Similarly the latest version deployed on internal test systems is even more computationally and memory efficient. Each timestep solving the LBH1B feeder is slightly longer, but much more computationally and thus energy efficient.



3 Conclusions

This study illustrates the progress GridQube is making in terms of improving the resource-intensity of its technologies. We pay attention to not just speed, but also necessary CPU cycles and RAM usage. These aspects matter a lot when running computations in parallel on servers, and for overall energy efficiency.