ExaSGD – Optimization grid dynamics at exascale

C. G. Petra, I.Aravena Solis (LLNL); N. Gawande, V. Amatya, A. Li, J. Li, S. Abhyankar, S. Peles (PNNL); M. Schanen, K. Kim, A. Maldonado, M. Anitescu (ANL)

Project Description

Our goal is to optimize power grid's response in a near-term timeframe to a variety of under-frequency hazards via physical and control threat scenarios. We use comprehensive modeling and analyze a large number of hazards simultaneously to quickly discover strategies to reestablish demand-supply balance with minimal impact to loads served. Optimization of complex engineering systems under accurate uncertainty characterizations, large spatial and temporal coverage, and detailed contingency analysis are *extreme-scale problems* that require scalable algorithms and HPC optimization solvers.

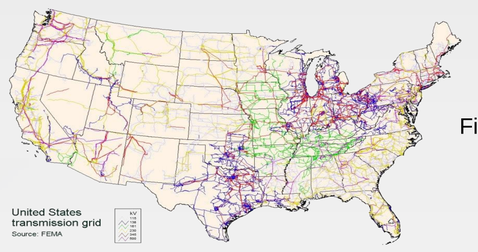


Fig. 1: The U.S. power grid.

Target application: cost-optimized generation and electricity power flow with security constraints

Technical Approach

We tackle the extreme scale of the problems by developing mathematically rigorous, second-order optimization algorithms and *specialized* HPC linear algebra kernels.

Julia for Prototyping

We use a framework written in Julia language to rapidly prototype and test novel optimization algorithms and automatic differentiation methods.

```
using CuArrays
n = 10
array   = CuArray{Float64,1,Nothing}(undef, n)
cuarray = Array{Float64,1}(undef, n)
for el in array
    el = el^2
end
# broadcast notation
array.^2
# Fast broadcast on GPU
cuarray.^2
```

- Use array/vector objects as hardware independent abstraction
- Execute broadcast operator '.' on GPUs

Toward derivative generation using CuArrays.jl

```
# Pg, coeff0, coeff2 = CuVector{Float64}(undef, ngen)
# Generate vectorized derivative code for Hessian
@objective(sum(coeff2 .* (baseMVA .*= Pg).^2 .+ coeff1 .*(baseMVA .* Pg) .+ coeff0))
```

Petascale Demo

We demonstrated petascale performance solving a problem with 768M variables and 524M constraints on 2048 KNL nodes of Theta supercomputer. Successful demonstration on a CPU-based architecture provided an evidence that the overall technical approach is sound.

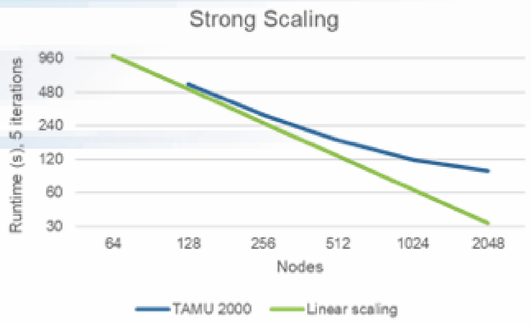


Fig. 2: Strong scaling of the contingency analysis computation on a CPU-based architecture.

The software stack used for the computation used Julia-based modeling framework StructJuMP[1], optimization solver PIPS[2], and MUMPS linear solver. Over **90%** of the computational cost is due to the linear solver.

1 https://github.com/StructJuMP/StructJuMP.jl
2 https://github.com/Argonne-National-Laboratory/PIPS

Modeling Frameworks

Model evaluation proxy app

Model evaluation of problems with irregular structure, such as power grids, is challenging on GPUs. A naïve model implementation typically leads to warp divergence and poor data coalescence, what creates a potential bottleneck for the entire analysis. To address this, we prototyped an approach to re-map model equations to create structures suitable for execution on GPU. Our preliminary results show that approach not only eliminates potential bottlenecks, but also produces significant performance gains versus traditional approach when running computations on GPU.

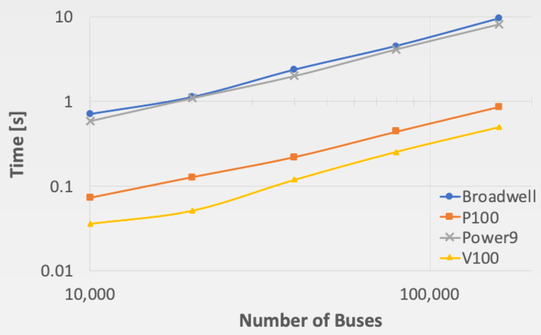


Fig. 3: Residual evaluation runtime vs. grid size (# of buses). Observed 10-20x speedup on GPU vs. 16-thread CPU runs.

Single residual evaluation computation is not large enough to flood Volta device. For 80,000-bus grid, the computation uses 20% SM and 50% memory bandwidth. Considering running multiple computations per device.

Optimization Solvers

Schur complement decomposition in stochastic optimization based on sparse linear algebra

$$\text{minimize } f_0(\boldsymbol{x_0}) + \sum_{i=1}^{N} f_i(\boldsymbol{x}_0, \boldsymbol{x}_i)$$

$$\text{subject to: } \boldsymbol{c_0}(x_0) = \boldsymbol{0}, \boldsymbol{c_i}(\boldsymbol{x_0}, \boldsymbol{x_i}) = \boldsymbol{0}, \boldsymbol{x_0} \geq \boldsymbol{0}, \boldsymbol{x_i} \geq \boldsymbol{0}.$$

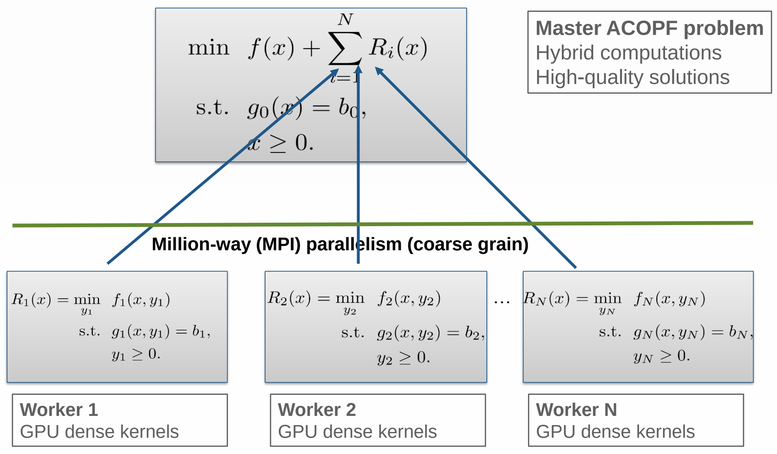
Many optimization problems in power grid, *e.g.,* multiperiod SC-ACOPF, have a large coupling $\boldsymbol{x}_0$ that negatively impact the efficiency of the scenario-based parallelization of the PIPS solver.

$$\boldsymbol{A} \approx \boldsymbol{K}_0 - \sum_{i=1}^{N} \boldsymbol{B}_i^T \boldsymbol{K}_i^{-1} \boldsymbol{B}_i$$

Experienced roadblock due to the poor performance of sparse direct solvers on GPUs.

New optimization decomposition and accelerator co-design developments

Outer optimization-based decomposition exploits the scenario/contingency structure



First transform ACOPF via Kron reduction:

$$\begin{bmatrix} i_\alpha \\ 0 \end{bmatrix} = \begin{bmatrix} Y^{\text{bus}}_{\alpha,\alpha} & Y^{\text{bus}}_{\alpha,\beta} \\ Y^{\text{bus}}_{\beta,\alpha} & Y^{\text{bus}}_{\beta,\beta} \end{bmatrix} \begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix}$$

$$i_\alpha = \underbrace{(Y^{\text{bus}}_{\alpha,\alpha} - Y^{\text{bus}}_{\alpha,\beta}(Y^{\text{bus}}_{\beta,\beta} \setminus Y^{\text{bus}}_{\beta,\alpha}))}_{Y^{\text{red}}} \; v_\alpha$$

Second, the optimization linear system is further compacted to a denser and smaller linear system to enable efficient dense linear algebra on GPUs.

$$\begin{bmatrix} H^s & 0 & (J^s)^T \\ 0 & H^d & (J^d)^T \\ J^s & J^d & 0 \end{bmatrix} \begin{bmatrix} \Delta x^s \\ \Delta x^d \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}$$

(sparsity is further compacted)

$$\left[H^d + (J^d)^T \left(J^s H^s (J^s)^T\right)^{-1} J^d\right] \Delta x^d = r^d$$

Example:
ACOPF 10K buses, 13K lines, 2K generators

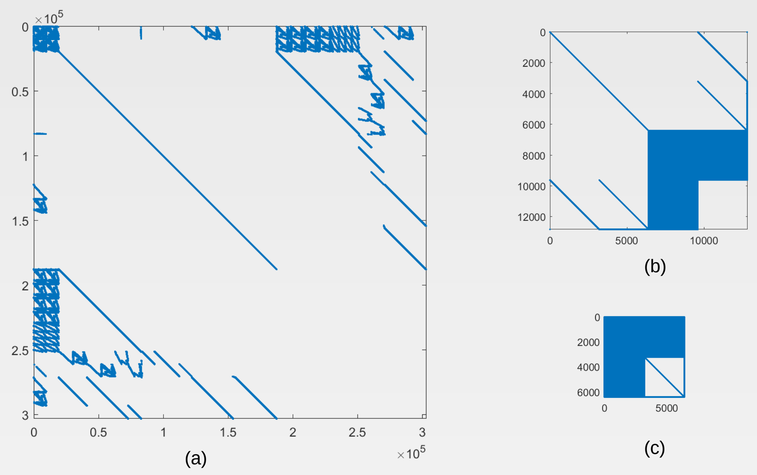


Fig. 4: Sparsity patterns linear systems for sparse ACOPF (a), ACOPF with Kron reduction (b), and of the final HiOp linear system (c).

The approach is implemented in HiOp and reaches 4 Tflops (approx. 70% of peak) on the GPU using Magma. In work is a device transfer hiding implementation.

HiOp is on Github https://github.com/LLNL/hiop

Synergetic Activities

Sparse direct linear solvers

Linear algebra of PIPS needs efficient sparse direct linear solvers on accelerators. ECP projects STRUMPACK[3] and SuperLU[4] provide promising solutions.

Dense direct linear solvers

Need *numerically stable* dense symmetric factorizations capable of considerable accelerator peak FLOPS. ECP project Magma[5] develops needed capability. The team is investigating other options, as well.

Programming models

The main programming model is based on C++ and MPI. The portability and on-node parallelism will be facilitated using hardware abstraction libraries such as Kokkos[6] and RAJA[7]. Currently, we are testing these two portability libraries.

Next Steps

Future work will be devoted to developing optimization algorithms that support computing at exascale systems Frontier and Aurora. The ultimate goal is to optimize and design the U.S. power grid with the necessary amount of geographical coverage, time resolution, and uncertainty resolutions.

Impact

The present work enables transmission grid planning and operation at unprecedented scale involving large number of uncertainty scenarios and contingencies. The computational capability developed is highly scalable; it can be deployed on DOE leadership HPC machines, as well as on a laptop.

3 https://portal.nersc.gov/project/sparse/strumpack/
4 https://github.com/xiaoyeli/superlu
5 https://icl.cs.utk.edu/magma/
6 https://github.com/kokkos/kokkos
7 https://github.com/llnl/raja

exascaleproject.org

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

LLNL-POST-738190

U.S. DEPARTMENT OF ENERGY | Office of Science | National Nuclear Security Administration