Revisiting structure-exploiting optimal power flow methods

François Pacaud
Assistant professor @ Mines Paris-PSL

LANS seminar
May 24, 2023

Who I am?

Short bio

$\rightarrow$ numerical optimizer by heart

- Former LANS postdoc, supervised by Mihai Anitescu
- Now assistant professor at Mines Paris-PSL

Today, presenting the work I did during my postdoc @ LANS (2020-2022)

Research question

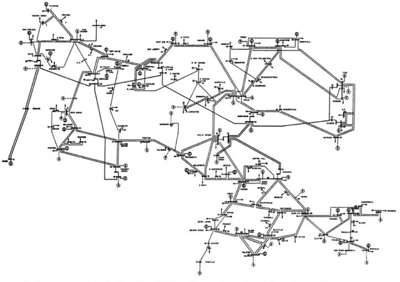
Can we solve large-scale nonlinear optimization problems on GPUs?

Fellow collaborators :

- Mihai Anitescu
- Adrian Maldonado
- Michel Schanen
- Sungho Shin

Motivation: solving optimal power flow problems on GPU architectures

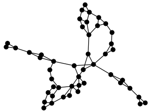
Our research is funded by the Exascale Computing Project (ECP)



Observation

Handling unstructured sparsity on SIMD architectures is non trivial

Physical model
unstructured



Upcoming hardware
GPU centric (SIMD)

Why GPUs are hard for optimizers?

Observation

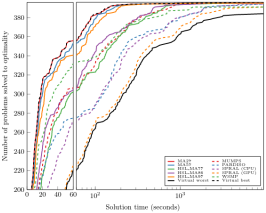
- GPUs are SIMD architecture (single instruction, multiple data)
- Excellent for *dense* and *batch* operations

On their hand, numerical optimization depends on two key routines

1. **Derivatives:** Evaluate derivatives using *Automatic Differentiation*
2. **Linear solve:** Solve KKT system to compute descent direction d_k as

$$(\nabla^2_{xx}\ell_k)d_k = -\nabla_x \ell_k$$

where $(\nabla^2_{xx}\ell_k)$ is *sparse symmetric indefinite*



(from [Tasseff et al., 2019])

- No good *sparse symmetric indefinite* solver on GPU
- Usual workarounds:
  1. Use decomposition algorithms (ADMM, [Kim et al., 2021])
  2. Use iterative solver (CG-based) [Cao et al., 2016, Schubiger et al., 2020]

Our solution: densification

Intuition: dense is easy on the GPU

Idea: Exploit the available degrees of freedom

Densify the problem using the *reduced Hessian*

$$\hat{H}_{uu} = Z^\top H Z$$

- Approach widely used during the 1980s/1990s
  - Summarized in [Fletcher, 1987, Section 12.5]: "Nonlinear elimination and feasible direction methods"
  - Also known as "Projected Hessian" [Nocedal and Overton, 1985, Gurwitz and Overton, 1989]
  - The reduced Hessian $\hat{H}_{uu}$ is often approximated [Biegler et al., 1995]
- The optimization community moved away from this technique in the 2000s:
  - "Many degrees of freedom" approaches [Poku et al., 2004]
  - Efficient resolution with interior-point combined with generic indefenite sparse linear solver (HSL [Duff and Reid, 1983], Pardiso [Schenk and Gärtner, 2004])
  - Lead to the development of mature NLP solvers (Ipopt [Wächter and Biegler, 2006], Knitro [Waltz et al., 2006])

MadNLP: a GPU-ready interior-point solver

MadNLP

MadNLP [Shin et al., 2020]

- Port of Ipopt in Julia
- Filter line-search interior-point method
- Open-source: `https://github.com/MadNLP/MadNLP.jl`

- **Derivatives:**
  - Custom automatic-differentation backend for OPF: ExaPF.jl
  - Derivatives evaluated in parallel on the GPU
- **Linear solver:** We compare two linear solvers for the KKT system
  1. *The reference:* HSL ma27 running on the CPU
  2. *The contender:* our reduction algorithm, using `cusolver` to factorize the reduced matrix with dense Cholesky on the GPU

Outline

Optimal power flow (OPF)

Security-Constrained Optimal power flow (SC-OPF)

Transient stability constraint OPF (TS-OPF)

Expliciting the physical constraints in the optimization problem

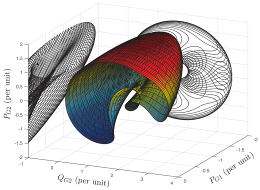


Figure: Nonlinear power flow (from [Hiskens and Davy, 2001])

Most real-life nonlinear problems encompasses a set of physical constraints

$$g(\boldsymbol{x}, \boldsymbol{u}) = 0$$

with $\boldsymbol{x}$ a state and $\boldsymbol{u}$ a control

Domain	g
Optimal control	Dynamics
PDE-constrained optimization	PDE
Optimal power flow	**Power flow**

Physically-constrained optimization problem

$$\min_{x,u} f(\boldsymbol{x}, \boldsymbol{u})$$
$$\text{s.t. } g(\boldsymbol{x}, \boldsymbol{u}) = 0 \, , \quad h(\boldsymbol{x}, \boldsymbol{u}) \leq 0$$

Well-known method [Cervantes et al., 2000, Biros and Ghattas, 2005]

Interior-point in a nutshell

Notations

- W: Hessian of Lagrangian
- G: Jacobian of equality constraints (power flow)
- A: Jacobian of inequalities (operational constraints)

The interior-point method (IPM) rewrites the problem in the standard form:

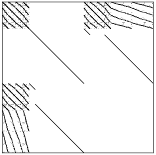
$$\min_{x,u,s} f(\boldsymbol{x}, \boldsymbol{u})$$
$$\text{s.t. } g(\boldsymbol{x}, \boldsymbol{u}) = 0 \,, \quad h(\boldsymbol{x}, \boldsymbol{u}) + s = 0, \quad s \geq 0$$

At each iteration, it solves the **augmented KKT linear system**

$$\begin{bmatrix} W + \Sigma_p & 0 & G^\top & A^\top \\ 0 & \Sigma_s & 0 & I \\ G & 0 & 0 & 0 \\ A & I & 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_v \\ \boldsymbol{p}_s \\ \boldsymbol{p}_\lambda \\ \boldsymbol{p}_y \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \\ \boldsymbol{r}_3 \\ \boldsymbol{r}_4 \end{bmatrix}$$

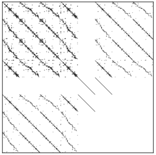
- in olive, blocks associated to the inequality constraints
- in violet, blocks associated to the equality constraints

Condense step: we remove the inequality constraints



1323 x 1323

$\Downarrow$



469 x 469

We remove the blocks associated to the inequality constraints by taking the Schur-complement

Condensed KKT

We define the *condensed KKT matrix* as

$$K := W + A^\top \Sigma_s A$$

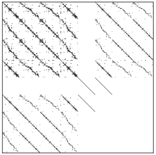
The augmented KKT system is equivalent to

$$\begin{bmatrix} K + \Sigma_p & G^\top \\ G & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_p \\ \boldsymbol{p}_\lambda \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_1 + A^\top (\Sigma_s \boldsymbol{r}_4 + \boldsymbol{r}_2) \\ \boldsymbol{r}_3 \end{bmatrix}$$

N.B.: This step is usually discarded because of additional fill-in in left-hand-side matrix, but here our goal is to densify the KKT system

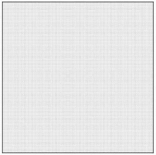
Reduce step: we remove the equality constraints

Idea: exploit the structure of the power flow equations $g(\boldsymbol{x}, \boldsymbol{u}) = 0$



469 x 469

$\Downarrow$



107 x 107

Expending the structure of the condensed KKT system:

$$\begin{bmatrix} K_{xx} + \Sigma_x & K_{xu} & G_x^\top \\ K_{ux} & K_{uu} + \Sigma_u & G_u^\top \\ G_x & G_u & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_x \\ \boldsymbol{p}_u \\ \boldsymbol{p}_\lambda \end{bmatrix} = \begin{bmatrix} \widehat{\boldsymbol{r}}_1 \\ \widehat{\boldsymbol{r}}_2 \\ \widehat{\boldsymbol{r}}_3 \end{bmatrix}$$

Reduced KKT

If the Jacobian G_x is invertible, the reduced Hessian is defined as

$$\hat{K}_{uu} := Z^\top K Z \quad \text{with} \quad Z := \begin{bmatrix} -G_x^{-1} G_u \\ I \end{bmatrix}$$

The condensed KKT system is equivalent to

$$\hat{K}_{uu}\, \boldsymbol{p}_u = \hat{\boldsymbol{r}}_2 - G_u^\top G_x^{-\top} \hat{\boldsymbol{r}}_1 - (K_{ux} - G_u^\top G_x^{-\top} K_{xx}) G_x^{-1} \hat{\boldsymbol{r}}_3$$

- Assembling $\hat{K}_{uu}$ can proceed in parallel once the sparse Jacobian G_x factorized
- The matrix $\hat{K}_{uu}$, dense, can be factorized efficiently on the GPU

Step 1: factorizing G_x

Message 1: `cusolverRF` is efficient to refactorize the matrix G_x on CUDA GPU

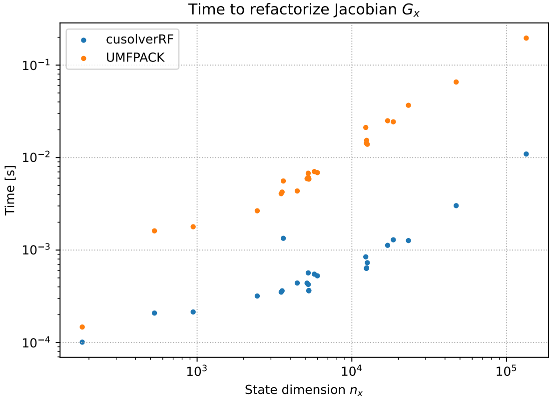


Figure: *Time to refactorize the Jacobian G_x against the dimension of the network.*

Step 2: parallel reduction algorithm (1)

We suppose given the condensed KKT matrix $K = W + A^\top \Sigma_s A$

$$\hat{K}_{uu} = \begin{bmatrix} -G_x^{-1} G_u \\ I \end{bmatrix}^\top \begin{bmatrix} K_{xx} & K_{xu} \\ K_{ux} & K_{uu} \end{bmatrix} \begin{bmatrix} -G_x^{-1} G_u \\ I \end{bmatrix}$$

We should avoid allocating the sensitivity matrix $S = -G_x^{-1} G_u$ (dense, size $n_x \times n_u$)!

$\rightarrow$ Instead, use batched `HessMat` product $\hat{K}_{uu} V$

`HessMat` kernel: batch adjoint-adjoint reduction

Input: LU factorization, such that $P G_x Q = L\, U$ (2 SpMM, 2 SpSM)

For every RHS $V \in \mathbb{R}^{n_u \times N}$

1. Solve $Z = G_x^{-1}(G_u V)$ (3 SpMM, 2 SpSM)
2. Evaluate $\begin{bmatrix} \Psi \\ H_u \end{bmatrix} = \begin{bmatrix} K_{xx} & K_{xu} \\ K_{ux} & K_{uu} \end{bmatrix} \begin{bmatrix} Z \\ V \end{bmatrix}$ (1 SpMM)
3. Solve $H_x = G_x^{-\top} \Psi$ (2 SpMM, 2 SpSM)
4. Output $\hat{K}_{uu} V = H_u - G_u H_x$ (1 SpMM)

- G_x first factorized on the CPU with KLU, then refactorized entirely on the GPU with `cusolverRF` (fast)
- $div(n_u, N) + 1$ `HessMat` products required to get full $\hat{K}_{uu}$

Step 2: parallel reduction algorithm (2)

Message 2: 7 seconds to evaluate $\hat{K}_{uu}$ for the largest instance (ACTIVSg70k)

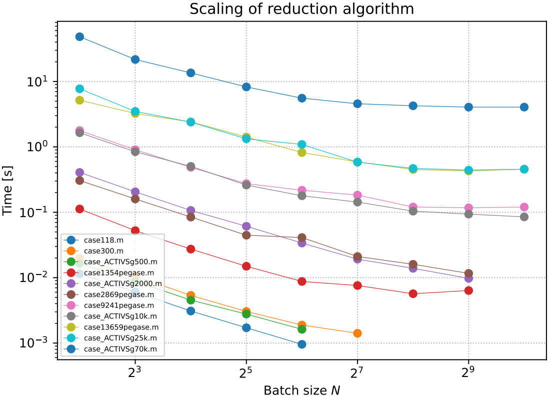


Figure: *Time to evaluate $\hat{K}_{uu}$ against the number of batch on the GPU, for various networks.*

Benchmark 1: Porting automatic differentiation on the GPU

Setting

- Knitro+ma27
  - **Derivatives:** MATPOWER (matlab)
  - **Linear solver:** ma27
- MadNLP+ma27
  - **Derivatives:** GPU-accelerated AD
  - **Linear solver:** ma27

	Knitro+ma27		MadNLP+ma27	
Case	#it	Time (s)	#it	Time (s)
9241pegase	102	**31.7**	69	**10.7**
ACTIVSg25k	47	**36.1**	86	**24.7**
ACTIVSg70k	101	**242.0**	90	**89.8**
9591goc	37	**22.5**	43	**11.7**
10480goc	40	**25.7**	50	**14.0**
19402goc	45	**66.5**	47	**30.8**

Benchmark 2: Porting the linear solver on the GPU

Setting

- MadNLP+ma27
  - **Derivatives:** GPU-accelerated AD
  - **Linear solver:** ma27
- MadNLP+reduced KKT
  - **Derivatives:** GPU-accelerated AD
  - **Linear solver:** reduction on GPU

		The reference MadNLP+ma27			*The contender* MadNLP+reduced KKT			
Case	DOF	#it	Time (s)	ma27 (s)	#it	Time (s)	Chol. (s)	Reduction (s)
		Problems with many degrees of freedom						
9241pegase	0.14	69	**10.7**	6.1	69	**23.7**	1.2	16.2
ACTIVSg25k	0.10	86	**24.7**	16.9	86	**85.0**	4.3	68.1
ACTIVSg70k	0.08	90	**89.8**	65.7	85	**658.2**	21.5	606.5
		Problems with few degrees of freedom						
9591goc	0.02	43	**11.7**	10.4	43	**7.7**	2.1	1.6
10480goc	0.03	50	**14.0**	12.0	50	**11.5**	3.9	3.3
19402goc	0.02	47	**30.8**	26.8	47	**19.5**	4.9	7.2

Table: *Comparing ma27 with reduced KKT linear solver. DOF is the % of degrees of freedom.*

When is reduced better than full-space?

Observation

The smaller the number of degrees of freedom n_u,
the more efficient is the reduction of the KKT system

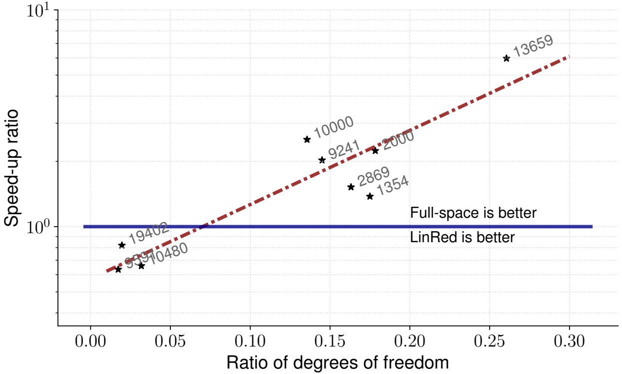


Figure: Illustrating the breakeven point

Summary for the classical OPF problem

Where we are at?

- **Derivatives:**
  - $\approx$ 2 times speed-up compared to MATPOWER just by evaluating the derivatives on GPU
- **Linear solver:**
  - Reduced KKT is a new structure-exploiting linear solver
  - Implemented entirely on the GPU (without memory transfer / the host)
  - Tractable in the large-scale regime, and beat state-of-the-art if # DOF is small

Outline

Optimal power flow (OPF)

Security-Constrained Optimal power flow (SC-OPF)

Transient stability constraint OPF (TS-OPF)

Extension to SC-OPF

SCOPF

Add N contingency scenarios (line tripping) to the base case OPF

In **preventive mode**, the SCOPF formulates as

$$\min_{x_0, x_1, \cdots, x_N, u} f(x_0, u) \quad \text{subject to} \quad \begin{cases} g(x_0, u) = 0 \\ h(x_0, u) \leq 0 \\ g(x_c, u) = 0 & \forall c = 1, \cdots, N \\ h(x_c, u) \leq 0 & \forall c = 1, \cdots, N \end{cases}$$

Conservative formulation: the control u (=power generations) is shared across all contingencies

Observations

- The derivatives can be evaluated in batch on the GPU
- The associated KKT system has a block arrowhead structure we can exploit in the reduced KKT solver

We follow the same procedure as before: *condense* and *reduce*

Fact 1: the condensed matrix has a block-arrowhead structure

The Hessian and the Jacobian of the SCOPF problems have all a *block-arrowhead structure*

$$W = \begin{bmatrix} W_{x_1 x_1} & & & W_{x_1 u} \\ & \ddots & & \vdots \\ & & W_{x_N x_N} & W_{x_N u} \\ W_{u x_1} & \dots & W_{u x_N} & W_{uu} \end{bmatrix}, \quad A = \begin{bmatrix} A^1_{x_1} & & A^1_u \\ & \ddots & \vdots \\ & A^N_{x_N} & A^N_u \end{bmatrix}$$

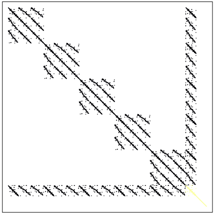


Figure: Condensed matrix K

Proposition

The condensed matrix $K = W + A^\top \Sigma_s A$ inherits the block-arrowhead structure of the Hessian W and the Jacobian H, with

$$K = \begin{bmatrix} K_{x_1 x_1} & & & K_{x_1 u} \\ & \ddots & & \vdots \\ & & K_{x_N x_N} & K_{x_N u} \\ K_{u x_1} & \dots & K_{u x_N} & K_{uu} \end{bmatrix}$$

Fact 2: we can reduce the KKT system by blocks

Idea: Exploit the block-arrowhead structure in the reduction

Proposition

If the Jacobian $G_x^0, \cdots, G_x^N$ are invertible, the reduced Hessian is defined as

$$\hat{K}_{uu} := Z^\top K Z \quad \text{with} \quad Z := \begin{bmatrix} -(G_x^0)^{-1} G_u^0 \\ \vdots \\ -(G_x^N)^{-1} G_u^N \\ I \end{bmatrix}$$

The KKT system is equivalent to

$$\hat{K}_{uu}\, p_u = -\hat{r}_2 + \sum_{i=1}^{N} \Big[(G_u^i)^\top (G_x^i)^{-\top} \hat{r}_1^i + \big[K_{ux_i} - (G_u^i)^\top (G_x^i)^{-\top} K_{x_i x_i} \big] (G_x^i)^{-1} \hat{r}_3^i \Big]$$

- The Jacobians $G_x^0, \cdots, G_x^N$ can be factorized in parallel on the GPU using `cusolverRF`
- The reduced Hessian $\hat{K}_{uu}$ is dense, with dimension $n_u \times n_u$

Numerical results on a single GPU (NVIDIA V100)

Setting: same as before

- MadNLP+ma27
  - **Derivatives:** GPU-accelerated AD
  - **Linear solver:** ma27
- MadNLP+reduced KKT
  - **Derivatives:** GPU-accelerated AD
  - **Linear solver:** reduction on GPU

		MadNLP+ma27				MadNLP+reduced KKT			
#bus	N	#it	Tot (s)	AD (s)	ma27 (s)	#it	Tot (s)	AD (s)	reduction (s)
1354	8	61	**10.8**	0.9	9.9	61	**7.9**	0.9	7.0
1354	16	54	**26.2**	1.1	25.1	54	**13.3**	1.2	12.1
1354	32	253	**1302.0**	9.0	1293.0	233	**172.2**	9.0	163.2
1354	64	135	**411.7**	8.6	403.1	236	**357.3**	14.5	342.8
9241	8	190	**1400.5**	31.7	1368.8	187	**1017.0**	30.5	986.5
9241	16	121	**3947.0**	38.1	3908.9	123	**1091.4**	34.4	1067.0

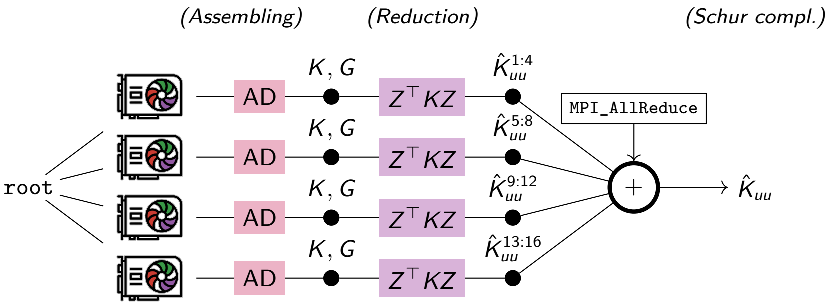
Table: Comparing the performance of the KKT linear solvers

Extension to multiple GPUs

Observation

We can gain additional speed-up by dispatching the solution on multiple GPUs

- ✓ Porting OPF to the exascale regime
- ✓ The approach is equivalent to the Schur-decomposition implemented previously in PIPS-NLP



Performance of automatic differentiation on multiple GPUs

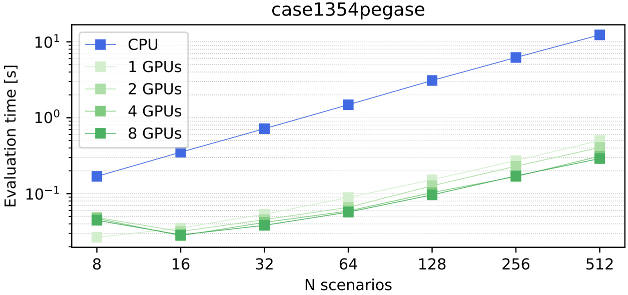


Figure: Time spent to evaluate the derivatives of `case1354pegase` with automatic differentiation

Performance of parallel reduced KKT solver on multiple GPUs

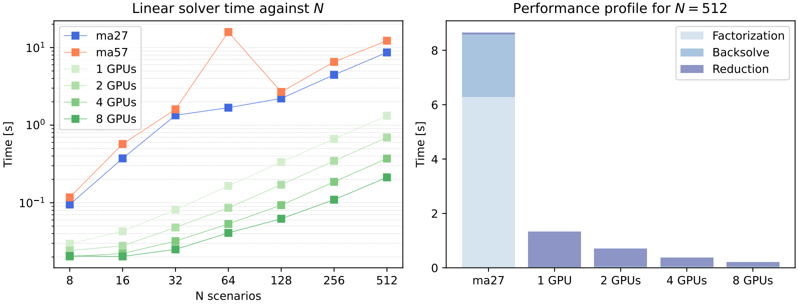


Figure: Time spent to solve the KKT system for `case1354pegase`

Solving SCOPF on multiple GPUs

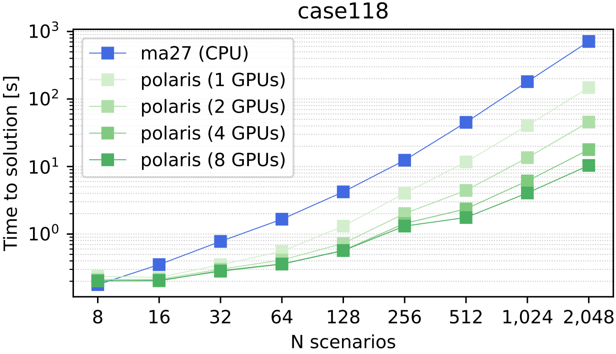


Figure: Time to solve the block-structured OPF problem case118 as a function of the number of scenarios N

Summary

Where we are at?

- **Derivatives:**
  - Very fast evaluation of the derivatives in batch on the GPU
- **Linear solver:**
  - Reduced KKT offers very good performance for SCOPF, as #DOF remains small (control u shared across contingencies)
  - Reduction can run in parallel on multiple GPUs
- **Generic observation:**
  - ✓ GPUs are well adapted to solve large-scale block-NLP problems with a shared structure (here: topology of the network is fixed +/- one line contingency)

Outline

Optimal power flow (OPF)

Security-Constrained Optimal power flow (SC-OPF)

Transient stability constraint OPF (TS-OPF)

Our next-step: transient-stabilit constraint problems

Extension to transient stability constraint OPF is under way

- Formulate as a NLP with DAE constraints
- Reduced-space approach for TS-OPF already studied in [Jiang and Geng, 2010]

Observation

The Jacobian inherits the dynamics structure of the problem:

$$G_x = \begin{bmatrix} G_{M,x_1} & & & & \cdots & \\ G_{S,x_2} & G_{M,x_2} & & & \cdots & \\ & G_{S,x_3} & G_{M,x_3} & & \cdots & \\ \vdots & \cdots & \cdots & \cdots & \ddots & \vdots \\ & & & & G_{S,x_T} & G_{M,x_T} \end{bmatrix}$$

It can also be re-interpreted as an optimal control problem

- [Wright, 1993, Cervantes et al., 2000]
- Parallel linear solver for dynamic problems already exist [Wright, 1990, Wright, 1991]

References I

Biegler, L. T., Nocedal, J., and Schmid, C. (1995).
A reduced Hessian method for large-scale constrained optimization.
SIAM Journal on Optimization, 5(2):314–347.

Biros, G. and Ghattas, O. (2005).
Parallel Lagrange–Newton–Krylov–Schur Methods for PDE-Constrained Optimization. Part I: The Krylov–Schur Solver.
SIAM Journal on Scientific Computing, 27(2):687–713.

Cao, Y., Seth, A., and Laird, C. D. (2016).
An augmented Lagrangian interior-point approach for large-scale NLP problems on graphics processing units.
Computers & Chemical Engineering, 85:76–83.

Cervantes, A. M., Wächter, A., Tütüncü, R. H., and Biegler, L. T. (2000).
A reduced space interior point strategy for optimization of differential algebraic systems.
Computers & Chemical Engineering, 24(1):39–51.

Duff, I. S. and Reid, J. K. (1983).
The multifrontal solution of indefinite sparse symmetric linear.
ACM Transactions on Mathematical Software (TOMS), 9(3):302–325.

References II

Fletcher, R. (1987).
Practical methods of optimization.
John Wiley & Sons.

Gurwitz, C. B. and Overton, M. L. (1989).
Sequential quadratic programming methods based on approximating a projected Hessian matrix.
SIAM Journal on Scientific and Statistical Computing, 10(4):631–653.

Hiskens, I. A. and Davy, R. J. (2001).
Exploring the power flow solution space boundary.
IEEE transactions on power systems, 16(3):389–395.

Jiang, Q. and Geng, G. (2010).
A reduced-space interior point method for transient stability constrained optimal power flow.
IEEE Transactions on Power Systems, 25(3):1232–1240.

Kim, Y., Pacaud, F., Kim, K., and Anitescu, M. (2021).
Leveraging gpu batching for scalable nonlinear programming through massive lagrangian decomposition.
arXiv preprint arXiv:2106.14995.

References III

Nocedal, J. and Overton, M. L. (1985).
Projected Hessian updating algorithms for nonlinearly constrained optimization.
SIAM Journal on Numerical Analysis, 22(5):821–850.

Poku, M. Y. B., Biegler, L. T., and Kelly, J. D. (2004).
Nonlinear optimization with many degrees of freedom in process engineering.
Industrial & engineering chemistry research, 43(21):6803–6812.

Schenk, O. and Gärtner, K. (2004).
Solving unsymmetric sparse systems of linear equations with pardiso.
Future Generation Computer Systems, 20(3):475–487.

Schubiger, M., Banjac, G., and Lygeros, J. (2020).
GPU acceleration of ADMM for large-scale quadratic programming.
Journal of Parallel and Distributed Computing, 144:55–67.

Shin, S., Coffrin, C., Sundar, K., and Zavala, V. M. (2020).
Graph-based modeling and decomposition of energy infrastructures.
arXiv preprint arXiv:2010.02404.

References IV

Tasseff, B., Coffrin, C., Wächter, A., and Laird, C. (2019).
Exploring benefits of linear solver parallelism on modern nonlinear optimization applications.
arXiv preprint arXiv:1909.08104.

Wächter, A. and Biegler, L. T. (2006).
On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming.
Mathematical Programming, 106(1):25–57.

Waltz, R. A., Morales, J. L., Nocedal, J., and Orban, D. (2006).
An interior algorithm for nonlinear optimization that combines line search and trust region steps.
Mathematical Programming, 107(3):391–408.

Wright, S. J. (1990).
Solution of discrete-time optimal control problems on parallel computers.
Parallel Computing, 16(2-3):221–237.

Wright, S. J. (1991).
Partitioned dynamic programming for optimal control.
SIAM Journal on optimization, 1(4):620–642.

Wright, S. J. (1993).
Interior point methods for optimal control of discrete time systems.
Journal of Optimization Theory and Applications, 77(1):161–187.