Revisiting structure-exploiting optimal power flow methods

François Pacaud
CAS @ Mines Paris-PSL

3rd workshop of the RTE Chair at CentraleSupélec
May 25, 2023

Who am I?

Numerical optimizer by heart

- Former postdoc at Argonne National Lab
- Now assistant professor at Mines Paris-PSL

These slides are summarizing the work we did during my postdoc at Argonne National Lab between 2020 and 2022

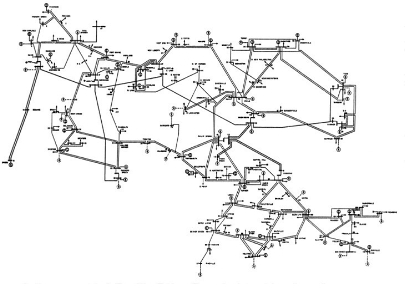
Argonne NATIONAL LABORATORY

Joint work with:
Mihai Anitescu, Adrian Maldonado, Michel Schanen, Sungho Shin

Broader research question

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

Motivation: solving optimal power flow problems on GPU architectures

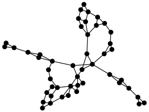


Observation

Handling unstructured sparsity on SIMD architectures is non trivial

Physical model

unstructured



Upcoming hardware

GPU centric (SIMD)

Why GPUs are hard for optimizers?

Observation

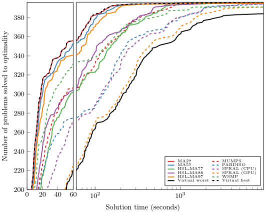
- GPUs = SIMD architectures (single instruction, multiple data)
- Excellent for *dense* and *batch* operations

On their hand, numerical optimization depends on two key routines

1. **Derivatives:** Evaluate derivatives using *Automatic Differentiation*
2. **Linear solve:** Compute the descent direction d_k by solving the KKT system

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

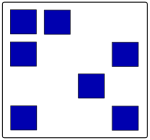
where $(\nabla^2_{xx}\ell_k)$ is *sparse symmetric indefinite*



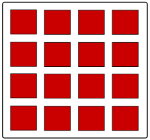
(from [Tasseff et al., 2019])

→ **Problem:** there is no good *sparse symmetric indefinite* solver on GPU

Our solution: densification



sparse



dense

Idea: Exploit the available degrees of freedom

Densify the problem using the *reduced Hessian*

$$\hat{H}_{uu} = Z^\top (\nabla^2_{xx} \ell_k) Z$$

Intuition: dense is easy on the GPU

MadNLP: a GPU-ready interior-point solver

MadNLP [Shin et al., 2020]

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

- **Linear solver:** We compare two linear solvers for 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

Expliciting the physical constraints in the optimization problem

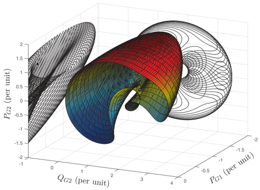


Figure: Nonlinear power flow (from [Hiskens and Davy, 2001])

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, e.g. line flows)*

The interior-point methods (IPM) reformulates the problem in the standard form:

$$\min_{x,u,s} f(\boldsymbol{x}, \boldsymbol{u})$$

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

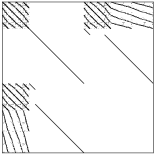
At each iteration, IPM solves the **augmented KKT linear system**

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

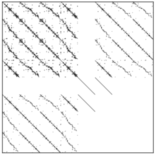
(in olive, blocks associated to the inequality constraints)

$\rightarrow$ the KKT system has a very specific structure!

Condense step: we remove the inequality constraints



$\Downarrow$



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

Condensed KKT

We define the *condensed KKT matrix* as

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

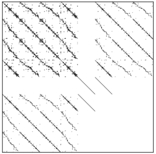
The augmented KKT system is equivalent to

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

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

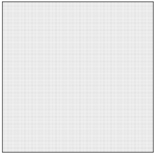
Reduce step: we remove the equality constraints

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 $\hat{K}_{uu}$ requires only the factorization of the sparse Jacobian G_x
- The matrix $\hat{K}_{uu}$, dense, can be factorized efficiently on the GPU

Numerical results: CPU or GPU?

Setting

- MadNLP+ma27
  - **Derivatives:** GPU-accelerated AD
  - **Linear solver:** ma27
- MadNLP+reduced KKT (full-GPU approach)
  - **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 ratio 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

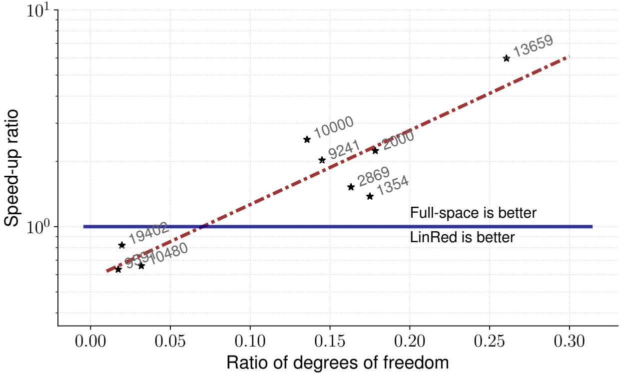


Figure: Illustrating the breakeven point

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* then *reduce*

Numerical results on a single GPU (NVIDIA V100)

Setting

- 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

Conclusion

Broader research question

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

✓ Yes, we can get a practical algorithm

✓ The more structure, the better

Next

- We showed the approach is practical
- Now, optimize it! (target: x10 speed-up)

References I

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.

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.

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

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

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.