Batched Second-Order Adjoint Sensitivity for Reduced Space Methods

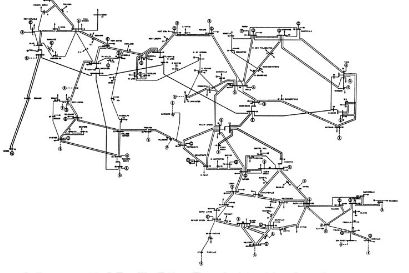
François Pacaud[♭] Michel Schanen[♭] Daniel Adrian Maldonado[♭]
Alexis Montoison[♮] Valentin Churavy[♯] Julian Samaroo Mihai Anitescu[♭]

[♭] Argonne National Laboratory (Mathematics and Computer Science Division)
[♮] Polytechnique Montréal (GERAD)
[♯] MIT CSAIL

SIAM PP22
Friday, February 25th

Motivation: solving optimal power flow problems on GPU architectures

Our research is funded by the ECP project: porting algorithms at exascale

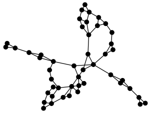


Challenge

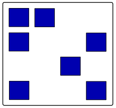
Handling unstructured sparsity on SIMD architectures is non trivial

Hardware
GPU centric (SIMD)

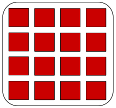
Physical model
unstructured



Idea



$\Longrightarrow$



Compress the graph structure with a nonlinear reduction

$$\begin{aligned}&\min_{\boldsymbol{x},\boldsymbol{u}} \; f(\boldsymbol{x},\boldsymbol{u}) \\ &\text{subject to} \;\; g(\boldsymbol{x},\boldsymbol{u}) = 0\end{aligned} \quad \Longrightarrow \quad \min_{\boldsymbol{u}} \; f(\underline{x}(\boldsymbol{u}),\boldsymbol{u})$$

Why?

- The functional $\underline{x}(\boldsymbol{u})$ satisfies *implicitly* $g(\underline{x}(\boldsymbol{u}),\boldsymbol{u}) = 0$!
- The second-order derivatives compress to a dense matrix

Reduction method: formalism

We denote $\boldsymbol{x} \in \mathbb{R}^{n_x}$ the state, $\boldsymbol{u} \in \mathbb{R}^{n_u}$ the control

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

$$\text{subject to} \quad g(\boldsymbol{x}, \boldsymbol{u}) = 0$$

<u>Assumptions:</u>

- Both the objective $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ and the physical equations $g : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ depend on $\boldsymbol{x}$ and $\boldsymbol{u}$
- Both f and g have smooth second-order derivatives

Implicit function theorem

Let $(\boldsymbol{x}, \boldsymbol{u}) \in \mathbb{R}^{n_x} \times \mathbb{R}^{n_u}$ such that $g(\boldsymbol{x}, \boldsymbol{u}) = 0$.
If $\nabla_x g(\boldsymbol{x}, \boldsymbol{u}) \in \mathbb{R}^{n_x \times n_x}$ is <u>invertible</u>, then there exists a local set $U \in \mathbb{R}^{n_u}$ and an unique differentiable function $\underline{x} : U \to \mathbb{R}^{n_x}$ such that locally

$$g(\underline{x}(\boldsymbol{u}), \boldsymbol{u}) = 0 \quad \forall u \in U$$

In practice $\underline{x}(\boldsymbol{u})$ is computed iteratively (e.g. with a Newton Raphson algorithm)

Reduced derivatives: first to second order

Let the reduced functional: $f_r(\boldsymbol{u}) := f(\underline{x}(\boldsymbol{u}), \boldsymbol{u})$
We note the Jacobians: $G_x = \nabla_x g \in \mathbb{R}^{n_x \times n_x}$, $G_u = \nabla_u g \in \mathbb{R}^{n_x \times n_u}$,

Reduced gradient

The chain rule gives directly:

$$\nabla f_r(\boldsymbol{u}) = \nabla_u f - G_u^\top G_x^{-\top} \nabla_x f$$

Complexity: one linear solve

For $\boldsymbol{\lambda} \in \mathbb{R}^{n_x}$, we define the Lagrangian

$$\ell(\boldsymbol{x}, \boldsymbol{u}; \boldsymbol{\lambda}) = f(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{\lambda}^\top g(\boldsymbol{x}, \boldsymbol{u})$$

We note $W = \nabla^2 \ell$ the Hessian of the Lagrangian in the full-space

Reduced Hessian

The reduced Hessian satisfies

$$\nabla^2 f_r(\boldsymbol{u}) = W_{uu} - W_{ux} G_x^{-1} G_u - G_u^\top G_x^{-\top} W_{xu} + G_u^\top G_x^{-\top} W_{xx} G_x^{-1} G_u$$

Complexity: n_u linear solves

Implementation

Reduction operation

$$\nabla^2 f_r(\boldsymbol{u}) = \begin{bmatrix} -G_x^{-1} G_u \\ I \end{bmatrix}^\top \begin{bmatrix} W_{xx} & W_{xu} \\ W_{ux} & W_{uu} \end{bmatrix} \begin{bmatrix} -G_x^{-1} G_u \\ I \end{bmatrix}$$

Complexity: either

- n_u linear solves if we store the $n_x \times n_u$ dense matrix $S = -G_u G_x^{-1}$
- $2n_u$ linear solves otherwise

Two successive operations:

1. Evaluate the Hessian W in the full space (automatic differentation)
2. Reduce the derivatives in the reduced space (linear algebra)

How-to: efficient linear algebra operations on the GPU

- `SpMV/SpMM` (sparse matrix - vector/matrix product
- `SpSV/SpSM` (sparse triangular solve)
- `SpRF` (sparse refactorization)

First step: Streamlining the automatic differentation

Factorizing the nonlinearities in the OPF problem (Lee et al., 2020)

There exists a basis $\psi(\boldsymbol{x}, \boldsymbol{u}) \in \mathbb{R}^{n_b}$ and two sparse matrices M, N such that

$$f(\boldsymbol{x}, \boldsymbol{u}) = M\psi(\boldsymbol{x}, \boldsymbol{u}) + b \,, \quad g(\boldsymbol{x}, \boldsymbol{u}) = N\psi(\boldsymbol{x}, \boldsymbol{u}) + c \,,$$

- Evaluate the basis ψ first, then recover f and g with `SpMV` operations
- We have implemented a GPU kernel for ψ and its adjoint $(\nabla\psi)^\top$

Computing the Hessian with forward-over-reverse

The Hessian W is a (super)-sparse matrix

- Find coloring associated to W
- Evaluate W with forward-over-reverse, knowing

$$(\nabla f)^\top = (\nabla\psi)^\top M^\top \,, \quad (\nabla g)^\top = (\nabla\psi)^\top N^\top$$

Performance: with a dual vector $\boldsymbol{d} \in \mathbb{D}_p^n$ with p partials, the linear operation $M^\top \boldsymbol{d}$ translates to a `SpMM` operation (RHS matrix with size $n \times (p+1)$)

First step: results

ExaPF: implemented in Julia, using `CUDA` and `KernelAbstractions.jl`

Our benchmark cases, ordered by size (obtained from MATPOWER):

Case	n_v	n_e	n_x	n_u	n_{colors}
IEEE118	118	186	181	107	27
IEEE300	300	411	530	137	24
PEGASE1354	1,354	1,991	2,447	519	28
PEGASE2869	2,869	4,582	5,227	1,019	35
PEGASE9241	9,241	16,049	17,036	2,889	85
ACTIVSg25K	25,000	32,230	47,246	6,531	36

Protocol

Compare time to evaluate W with

- Reference: JuMP's AD (open-source, Julia code)
- our algo on the CPU (ExaPF CPU)
- our algo on the GPU (ExaPF GPU)

First step: results

ExaPF: implemented in Julia, using `CUDA` and `KernelAbstractions.jl`

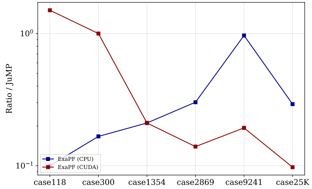
Our benchmark cases, ordered by size (obtained from MATPOWER):

Case	n_v	n_e	n_x	n_u	n_{colors}
IEEE118	118	186	181	107	27
IEEE300	300	411	530	137	24
PEGASE1354	1,354	1,991	2,447	519	28
PEGASE2869	2,869	4,582	5,227	1,019	35
PEGASE9241	9,241	16,049	17,036	2,889	85
ACTIVSg25K	25,000	32,230	47,246	6,531	36

Protocol

Compare time to evaluate W with

- Reference: JuMP's AD (open-source, Julia code)
- our algo on the CPU (ExaPF CPU)
- our algo on the GPU (ExaPF GPU)



Case	JuMP (CPU)	ExaPF (CPU)	ExaPF (CUDA)
IEEE118	0.002	0.0002	0.003
IEEE300	0.003	0.0005	0.003
PEGASE1354	0.019	0.004	0.004
PEGASE2869	0.043	0.013	0.006
PEGASE9241	0.150	0.145	0.029
ACTIVSg25K	0.359	0.105	0.035

Table: Results: evaluation time in seconds

Second step: reduction operation

We have computed the Hessian W in the first step

$$\nabla^2 f_r(\boldsymbol{u}) = \begin{bmatrix} -G_x^{-1} G_u \\ I \end{bmatrix}^\top \begin{bmatrix} W_{xx} & W_{xu} \\ W_{ux} & W_{uu} \end{bmatrix} \begin{bmatrix} -G_x^{-1} G_u \\ I \end{bmatrix}$$

We should avoid allocating the sensitivity matrix $S = -G_u G_x^{-1}$ (size $n_x \times n_u$)!
Instead, use batched `HessMat` product $\nabla^2 f_r(\boldsymbol{u}) V$

`HessMat` kernel: batch adjoint-adjoint reduction

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

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

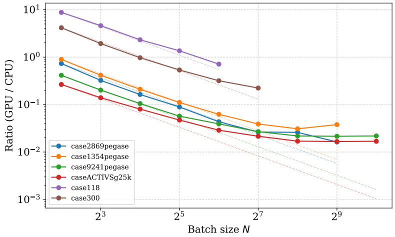
1. Solve $Z = G_x^{-1}(G_u V)$ (3 `SpMM`, 2 `SpSM`)
2. Evaluate $\begin{bmatrix} \Psi \\ H_u \end{bmatrix} = \begin{bmatrix} W_{xx} & W_{xu} \\ W_{ux} & W_{uu} \end{bmatrix} \begin{bmatrix} Z \\ V \end{bmatrix}$ (1 `SpMM`)
3. Solve $H_x = G_x^{-\top} \Psi$ (2 `SpMM`, 2 `SpSM`)
4. Output $\nabla^2 f_r(\boldsymbol{u}) V = H_u - G_u H_x$ (1 `SpMM`)

- G_x first factorized on the CPU with KLU,
  then refactorized entirely on the GPU with `cusolverRF` (fast) [1]
- $div(n_u, N) + 1$ `HessMat` products required to get full $\nabla^2 f_r(\boldsymbol{u})$

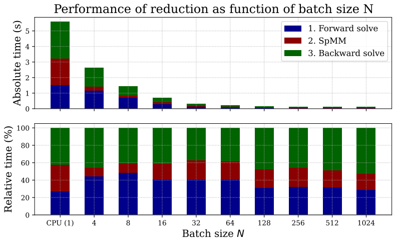
[1] Credits to Kasia Swirydowicz, PNNL, for the idea and the RF wrapper (Świrydowicz et al., 2021)

Second step: results

- **Question 1:** What is the appropriate batch size N?

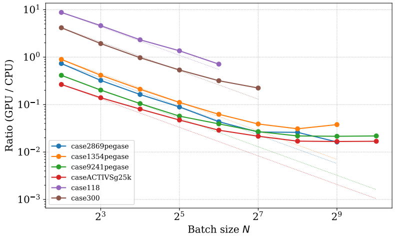


- **Question 2:** What is the bottleneck in the reduction algorithm?

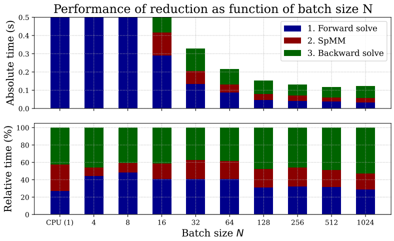


Second step: results

- **Question 1:** What is the appropriate batch size *N*?



- **Question 2:** What is the bottleneck in the reduction algorithm?



References I

Lee, D., Turitsyn, K., Molzahn, D. K., and Roald, L. A. (2020). Feasible path identification in optimal power flow with sequential convex restriction. IEEE Transactions on Power Systems, 35(5):3648–3659.

Świrydowicz, K., Darve, E., Jones, W., Maack, J., Regev, S., Saunders, M. A., Thomas, S. J., and Peleš, S. (2021). Linear solvers for power grid optimization problems: a review of gpu-accelerated linear solvers. Parallel Computing, page 102870.