Solving Large-Scale, Sparse, Constrained Nonlinear Optimization Problems on GPUs

Pivoting-Free Interior Point Methods

Sungho Shin

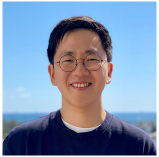
shin.mit.edu

Massachusetts Institute of Technology

2025 INFORMS Annual Meeting
October 27, 2025

Who are we?

https://madsuite.org/

- Alexis Montoison @ Argonne National Lab
- François Pacaud @ MINES Paris-PSL (an ANL alumnus)
- Sungho Shin @ MIT (an ANL alumnus)
- Mihai Anitescu @ Argonne National Lab
- and friends... Michael Saunders, Dominic Orban, Armin Nurkanović, Anton Pozharskiy, Jean-Baptiste Caillau, ...

GPU-based optimization

- What is GPU? —many-core accelerator with high memory bandwidth
- Traditionally used for graphics, now widely used for ML/AI workloads
- Classical optimization (mathematical program; not ML training problems) has been thought to be unsuitable for GPUs
- Many state-of-the-art solvers (on CPUs) rely on sparse direct factorizations, which has been considered difficult to implement efficiently on GPUs
- This status quo is changing—the focus of this talk

GPU-based optimization: current landscape

- Factoriation-free methods have been the dominant approach
  - PDLP: Primal-dual first-order methods (Applegate et al., 2022; Lu et al., 2025)
  - PCG-based algorithms for convex QPs (Schubiger et al., 2020)
  - ALM + IPM + PCG for nonlinear problems (Cao et al., 2016)
- Factoriztion-free methods only rely on GPU-friendly kernels—`spmv`, `axpy`, `map`, `mapreduce`—makes them easy to implement on GPUs
- Scale to extremely large problems but achieving high precision is difficult
- Can we implement second-order methods with sparse direct solvers on GPUs?

David Applegate et al. (2022). *Practical Large-Scale Linear Programming Using Primal-Dual Hybrid Gradient*. arXiv: 2106.04756 [math]
Haihao Lu et al. (2025). *cuPDLP+: A Further Enhanced GPU-Based First-Order Solver for Linear Programming*. arXiv: 2507.14051 [math]
Michel Schubiger et al. (2020). "GPU Acceleration of ADMM for Large-Scale Quadratic Programming". In: *Journal of Parallel and Distributed Computing*
Yankai Cao et al. (2016). "An Augmented Lagrangian Interior-Point Approach for Large-Scale NLP Problems on Graphics Processing Units". In: *Computers & Chemical Engineering*

Our problem setting

Large-scale, sparse, constrained nonlinear programs

- We consider:

$$\min_{x^\flat \le x \le x^\sharp} f(x) \quad \text{s.t. } g^\flat \le g(x) \le g^\sharp,$$

  … but will particularly focus on:
  - $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}^m$ twice continuously differentiable
  - n and m are large (millions or more)
  - $\nabla g(x)$ and $\nabla^2_{xx} L(x, \lambda)$ are sparse (tens of nonzeros per row)—$L(\cdot, \cdot)$ is the Lagrangian
- Generally nonconvex—we only seek local solutions
- Most important instances (e.g., energy systems, optimal control) fall into this category
- Most existing solvers (e.g., Ipopt, KNITRO) are optimized for such problems

Our problem setting

Large-scale, sparse, constrained nonlinear programs

- WLOG (via $a = b \iff a \geq b,\ b \geq a$), we consider:

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t. } g(x) \geq 0,$$

  ... but will particularly focus on:
  - $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}^m$ twice continuously differentiable
  - n and m are large (millions or more)
  - $\nabla g(x)$ and $\nabla^2_{xx} L(x, \lambda)$ are sparse (tens of nonzeros per row)—$L(\cdot, \cdot)$ is the Lagrangian
- Generally nonconvex—we only seek local solutions
- Most important instances (e.g., energy systems, optimal control) fall into this category
- Most existing solvers (e.g., Ipopt, KNITRO) are optimized for such problems

NLP software stack—either on CPUs or GPUs

(1) algebraic modeling system, (2) NLP solver, and (3) sparse, direct linear solver

- Algebraic modeling systems (e.g., JuMP, AMPL, CasADi) generate callback functions—$f(\cdot)$, $g(\cdot)$, $\nabla f(\cdot)$, $\nabla g(\cdot)$, $\nabla^2_{xx} L(\cdot,\cdot)$—for NLP solvers
- NLP solvers (e.g., Ipopt, KNITRO) form KKT systems to find step direction:

$$\underbrace{\begin{bmatrix} \nabla^2_{xx} L(x,s,\lambda) + \delta_p I & & \nabla g(x)^\top \\ & S^{-1}\Lambda & -I \\ \nabla g(x) & -I & -\delta_d I \end{bmatrix}}_{\text{KKT matrix}} \underbrace{\begin{bmatrix} d_x \\ d_s \\ -d_\lambda \end{bmatrix}}_{\text{step direction}} = - \underbrace{\begin{bmatrix} \nabla f(x) - \nabla g(x)^\top \lambda \\ \Lambda e - \mu S^{-1} e \\ g(x) - s \end{bmatrix}}_{\text{residual to KKT conditions}},$$

  where s is the slack variable, $S = \mathrm{diag}(s)$, $\Lambda = \mathrm{diag}(\lambda)$, and $\delta_p, \delta_d \geq 0$ are primal-dual regularization parameters
- Sparse, direct linear solvers (e.g., MA27, PARDISO) factorize and solve KKT systems; iterative method is typically not an option due to ill-conditioning

Full NLP software stack on GPU—is it possible?

(1) algebraic modeling system, (2) NLP solver, and (3) sparse, direct linear solver

- Partially porting on GPU (e.g., AD or linear algebra only) is not sufficient—will be bottlenecked by CPU–GPU communication. We need all three components on GPU
- Porting algebraic modeling system is relatively straightforward. ExaModels.jl provides GPU-compatible modeling capabilities (Shin et al., 2024)—not the focus today
- Porting direct linear solver is more non-trivial; as of now, there is no drop-in replacement of MA27 on GPUs

Our approach

Adapt interior-point method for NLPs so that it can work with GPU direct linear solvers with (relatively) limited capabilities

Sungho Shin et al. (2024). "Accelerating Optimal Power Flow with GPUs: SIMD Abstraction of Nonlinear Programs and Condensed-Space Interior-Point Methods". In: *Electric Power Systems Research*

Sparse direct solvers within interior-point methods

LBL$^\top$ factorization

$$\begin{bmatrix} \nabla^2_{xx} L(x,s,\lambda) + \delta_p I & & \nabla g(x)^\top \\ & S^{-1}\Lambda & -I \\ \nabla g(x) & -I & -\delta_d I \end{bmatrix} = P \times Q \times L \times B \times L^\top \times Q^\top \times P^\top$$

- P: Fill-in-reducing (and parallel-friendly) re-ordering (re-used across IPM iterations)
- Q: Pivoting (computed on-the-fly during factorization)
- $LBL^\top$: Factorization of permuted matrix
- Additionally, pivots may be perturbed, which must be corrected via iterative refinement
- Factorization reveals inertia—number of $+$, 0, and $-$ eigenvalues; primal-dual regularization (δ_p, δ_d) is adjusted to achieve correct inertia (Wächter et al., 2006); this can be merged with pivot perturbation (Nocedal et al., 2006)

Andreas Wächter et al. (2006). "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming". In: *Mathematical Programming*

Jorge Nocedal et al. (2006). *Numerical Optimization*. 2nd ed. Springer Series in Operations Research. New York: Springer

Pivoting-free IPM

Montoison, Pacaud, Shin, et al., 2025

- Numerical pivoting is difficult to implement on GPUs (Świrydowicz et al., 2022) because row swapping can potentially destroy parallelism
- Existing GPU solvers (e.g., cuDSS) don't have adequately robust pivoting strategies; currently only have partial pivoting and pivot perturbation (NVIDIA, 2024)
- Factorization may fail without numerical pivoting, unless
  - Symmetric quasi-definite (SQD): factorization exists (Vanderbei, 1995)
  - Symmetric positive definite (SPD): factorization exists and numerically stable

  However, KKT systems for NLPs are generally indefinite
- When pivoting is not required, GPU sparse direct solvers can be very efficient

Can we make IPM pivoting-free?

Alexis Montoison, François Pacaud, Sungho Shin, et al. (2025). *GPU Implementation of Second-Order Linear and Nonlinear Programming Solvers*. arXiv: 2508.16094 [math]
Robert J. Vanderbei (1995). "Symmetric Quasidefinite Matrices". In: *SIAM Journal on Optimization*
Kasia Świrydowicz et al. (2022). "Linear Solvers for Power Grid Optimization Problems: A Review of GPU-accelerated Linear Solvers". In: *Parallel Computing*
NVIDIA (2024). *NVIDIA cuDSS (Preview): A High-Performance CUDA Library for Direct Sparse Solvers — NVIDIA cuDSS Documentation*. https://docs.nvidia.com/cuda/cudss/index.html

Pivoting-free IPM

Montoison, Pacaud, Shin, et al., 2025

- Numerical pivoting is difficult to implement on GPUs (Świrydowicz et al., 2022) because row swapping can potentially destroy parallelism
- Existing GPU solvers (e.g., cuDSS) don't have adequately robust pivoting strategies; currently only have partial pivoting and pivot perturbation (NVIDIA, 2024)
- Factorization may fail without numerical pivoting, unless
  - Symmetric quasi-definite (SQD): factorization exists (Vanderbei, 1995)
  - Symmetric positive definite (SPD): factorization exists and numerically stable

  However, KKT systems for NLPs are generally indefinite
- When pivoting is not required, GPU sparse direct solvers can be very efficient

(For now) can we reformulate KKT systems to be SPD or SQD?

Alexis Montoison, François Pacaud, Sungho Shin, et al. (2025). *GPU Implementation of Second-Order Linear and Nonlinear Programming Solvers*. arXiv: 2508.16094 [math]

Robert J. Vanderbei (1995). "Symmetric Quasidefinite Matrices". In: *SIAM Journal on Optimization*

Kasia Świrydowicz et al. (2022). "Linear Solvers for Power Grid Optimization Problems: A Review of GPU-accelerated Linear Solvers". In: *Parallel Computing*

NVIDIA (2024). *NVIDIA cuDSS (Preview): A High-Performance CUDA Library for Direct Sparse Solvers — NVIDIA cuDSS Documentation*. https://docs.nvidia.com/cuda/cudss/index.html

Option 1: Lifted KKT system

Shin et al., 2024

- First, relax equalities by small enough τ (the same as solver tolerances):

$$-\tau \leq h(x) \leq \tau$$

- Condense the KKT system by eliminating slack/dual variables:

$$\begin{bmatrix} \nabla^2_{xx} L(x,s,\lambda) + \delta_p I & & \nabla g(x)^\top \\ & S^{-1}\Lambda & -I \\ \nabla g(x) & -I & -\delta_d I \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ -d_\lambda \end{bmatrix} = - \begin{bmatrix} \nabla f(x) - \nabla g(x)^\top \lambda \\ \Lambda e - \mu S^{-1} e \\ g(x) - s \end{bmatrix}$$

$$\iff \Big(\underbrace{\nabla^2_{xx} L(x,s,\lambda) + \delta_p I + \nabla g(x)^\top (\delta_d I + \Lambda^{-1} S)^{-1} \nabla g(x)}_{\text{SPD iff the original system has correct inertia}} \Big) d_x = -r_p$$

(and some easy-to-solve equations)

- Enables numerically stable factorization without pivoting or excessive regularization

Sungho Shin et al. (2024). "Accelerating Optimal Power Flow with GPUs: SIMD Abstraction of Nonlinear Programs and Condensed-Space Interior-Point Methods". In: *Electric Power Systems Research*

Option 1: Lifted KKT—numerical results (AC OPF)

Montoison, Pacaud, Shin, et al., 2025

	Tol	Solver	Small $\text{nnz} < 2^{18}$		Medium $2^{18} \leq \text{nnz} < 2^{20}$		Large $2^{20} \leq \text{nnz}$		Total	
			Solved	Time	Solved	Time	Solved	Time	Solved	Time
OPF	10^{-4}	MadNLP (GPU)	**31**	0.4166	**24**	**2.6380**	**11**	**3.7040**	**66**	**1.6979**
		Ipopt (CPU)	**31**	0.3970	**24**	5.0697	11	38.5053	**66**	5.3817
	10^{-8}	MadNLP (GPU)	30	2.5037	**24**	**4.6016**	10	**12.8040**	64	**4.6228**
		Ipopt (CPU)	**31**	0.5100	**24**	5.4292	11	37.7818	**66**	5.5541

- Benchamrk set: AC optimal power flow (Babaeinejadsarookolaee et al., 2021)
- Hardware: NVIDIA GV 100 GPU / Intel Xeon Gold 6130 CPU
- Time: reported as SGM10:= $\prod_{i=1}^{n}(t_i + 10))^{1/n} - 10$, max wall time 900s

Observation

GPU solver is most effective for large-scale problems for moderate accuracy (10^{-4})

Alexis Montoison, François Pacaud, Sungho Shin, et al. (2025). *GPU Implementation of Second-Order Linear and Nonlinear Programming Solvers*. arXiv: 2508.16094 [math]

Sogol Babaeinejadsarookolaee et al. (2021). *The Power Grid Library for Benchmarking AC Optimal Power Flow Algorithms*. arXiv: 1908.02788 [math]

Option 1: Lifted KKT—numerical results (COPS)

Montoison, Pacaud, Shin, et al., 2025

	Tol	Solver	Small $\text{nnz} < 2^{18}$ Solved	Time	Medium $2^{18} \leq \text{nnz} < 2^{20}$ Solved	Time	Large $2^{20} \leq \text{nnz}$ Solved	Time	Total Solved	Time
COPS	10^{-4}	MadNLP (GPU)	**13**	**0.8665**	**15**	**4.8665**	**16**	**3.8194**	**44**	**3.2314**
		Ipopt (CPU)	**13**	5.2315	**15**	15.9701	15	45.8411	43	19.2243
	10^{-8}	MadNLP (GPU)	**13**	**0.8575**	**16**	**1.5572**	**16**	**8.3549**	**45**	**3.3797**
		Ipopt (CPU)	**13**	5.9413	**15**	17.6758	15	40.8639	43	19.2999

- Benchamrk set: COPS Benchmark (Dolan et al., 2001)
- Hardware: NVIDIA GV 100 GPU / Intel Xeon Gold 6130 CPU
- Time: reported as SGM10:$= \prod_{i=1}^{n}(t_i + 10))^{1/n} - 10$, max wall time 900s

Observation

GPU solver is most effective for large-scale problems for moderate accuracy (10^{-4})

Alexis Montoison, François Pacaud, Sungho Shin, et al. (2025). *GPU Implementation of Second-Order Linear and Nonlinear Programming Solvers*. arXiv: 2508.16094 [math]

Sogol Babaeinejadsarookolaee et al. (2021). *The Power Grid Library for Benchmarking AC Optimal Power Flow Algorithms*. arXiv: 1908.02788 [math]

Option 2: Hybrid KKT system

Golub et al., 2003; Pacaud et al., 2024; Regev et al., 2023

- We keep the equality constraints $h(x) = 0$ and eliminate the inequalities (slack/dual):

$$\begin{bmatrix} K_{\text{cond}} + \delta_p I & \nabla h(x)^\top \\ \nabla h(x) & -\delta_d I \end{bmatrix} \begin{bmatrix} d_x \\ -d_\lambda \end{bmatrix} = - \begin{bmatrix} r_x \\ r_\lambda \end{bmatrix}$$

$$\xrightarrow[\text{(w/o changing solution)}]{\text{Regularization}} \begin{bmatrix} K_{\text{cond}} + \delta_p I + \rho \nabla h(x)^\top \nabla h(x) & \nabla h(x)^\top \\ \nabla h(x) & -\delta_d I \end{bmatrix} \begin{bmatrix} d_x \\ -d_\lambda \end{bmatrix} = - \begin{bmatrix} r'_x \\ r_\lambda \end{bmatrix}$$

Then, the Schur complement can be solved with an iterative method (e.g., CG)

$$\Bigg(\underbrace{\nabla h(x)^\top \Big(\overbrace{K_{\text{cond}} + \delta_p I + \rho \nabla h(x)^\top \nabla h(x)}^{\text{SPD for sufficiently large } \rho \text{ iff correct inertia}} \Big)^{-1} \nabla h(x)}_{\text{converges to } I \text{ as } \rho \to \infty} + \delta_d I \Bigg) d_\lambda = r'_\lambda$$

Gene H. Golub et al. (2003). "On Solving Block-Structured Indefinite Linear Systems". In: *SIAM Journal on Scientific Computing*

Shaked Regev et al. (2023). "HyKKT: A Hybrid Direct-Iterative Method for Solving KKT Linear Systems". In: *Optimization Methods and Software*

François Pacaud et al. (2024). *Condensed-Space Methods for Nonlinear Programming on GPUs*. arXiv: 2405.14236 [math]

Options 3, 4, 5, ...

- There can be lots of other ways to implement pivoting-free solvers
- MadIPM: Solver for convex quadratic programs; uses regularization to avoid pivoting (Montoison, Pacaud, Shin, et al., 2025)
- MadNCL: ALM-based algorithm—ALM is more GPU-friendly due to the quadratic penalty term. Uses nonlinearly-constrained Lagrangian (NCL) formulation and applies condensation techniques (Montoison, Pacaud, Saunders, et al., 2025)
- Directly solving augmented systems without pivoting? —currently under investigation

Concluding remarks

- It is an exciting time to develop optimization solvers!
- Existing GPU solvers have focused on factorization-free (e.g. first-order) methods, but

$$\text{GPU optimization solver} \supsetneq \text{Factorization-free solver.}$$

- With GPU direct solvers, second-order methods on GPUs has now become possible
- To harness these GPU direct solvers, we need pivoting-free algorithms
- We presented Lifted KKT and Hybrid KKT, but there can be many others

Check out our project page!

https://madsuite.org/

References

Applegate, David et al. (2022). *Practical Large-Scale Linear Programming Using Primal-Dual Hybrid Gradient*. arXiv: 2106.04756 [math].

Babaeinejadsarookolaee, Sogol et al. (2021). *The Power Grid Library for Benchmarking AC Optimal Power Flow Algorithms*. arXiv: 1908.02788 [math].

Cao, Yankai et al. (2016). "An Augmented Lagrangian Interior-Point Approach for Large-Scale NLP Problems on Graphics Processing Units". In: *Computers & Chemical Engineering* 85, pp. 76–83.

Dolan, E. D. et al. (2001). *Benchmarking Optimization Software with COPS*. Tech. rep. ANL/MCS-TM-246. Argonne National Lab., IL (US).

Golub, Gene H. et al. (2003). "On Solving Block-Structured Indefinite Linear Systems". In: *SIAM Journal on Scientific Computing* 24.6, pp. 2076–2092.

Lu, Haihao et al. (2025). *cuPDLP+: A Further Enhanced GPU-Based First-Order Solver for Linear Programming*. arXiv: 2507.14051 [math].

Montoison, Alexis, François Pacaud, Michael Saunders, et al. (2025). *MadNCL: A GPU Implementation of Algorithm NCL for Large-Scale, Degenerate Nonlinear Programs*. arXiv: 2510.05885 [math].

Montoison, Alexis, François Pacaud, Sungho Shin, et al. (2025). *GPU Implementation of Second-Order Linear and Nonlinear Programming Solvers*. arXiv: 2508.16094 [math].

Nocedal, Jorge et al. (2006). *Numerical Optimization*. 2nd ed. Springer Series in Operations Research. New York: Springer.

NVIDIA (2024). *NVIDIA cuDSS (Preview): A High-Performance CUDA Library for Direct Sparse Solvers — NVIDIA cuDSS Documentation*. https://docs.nvidia.com/cuda/cudss/index.html.

Pacaud, François et al. (2024). *Condensed-Space Methods for Nonlinear Programming on GPUs*. arXiv: 2405.14236 [math].

Regev, Shaked et al. (2023). "HyKKT: A Hybrid Direct-Iterative Method for Solving KKT Linear Systems". In: *Optimization Methods and Software* 38.2, pp. 332–355.

Schubiger, Michel et al. (2020). "GPU Acceleration of ADMM for Large-Scale Quadratic Programming". In: *Journal of Parallel and Distributed Computing* 144, pp. 55–67.

Shin, Sungho et al. (2024). "Accelerating Optimal Power Flow with GPUs: SIMD Abstraction of Nonlinear Programs and Condensed-Space Interior-Point Methods". In: *Electric Power Systems Research* 236, p. 110651.

Świrydowicz, Kasia et al. (2022). "Linear Solvers for Power Grid Optimization Problems: A Review of GPU-accelerated Linear Solvers". In: *Parallel Computing* 111, p. 102870.

Vanderbei, Robert J. (1995). "Symmetric Quasidefinite Matrices". In: *SIAM Journal on Optimization* 5.1, pp. 100–113.

Wächter, Andreas et al. (2006). "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming". In: *Mathematical Programming* 106.1, pp. 25–57.