MadIPM: a GPU-accelerated solver for linear programming

Why second-order methods remain relevant

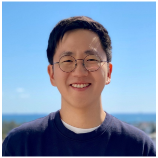
François Pacaud

CAS, Mines Paris - PSL

PGMO Days 2025

Who are we?

https://madsuite.org/

- Alexis Montoison @ Argonne National Laboratory
- François Pacaud @ MINES Paris-PSL
- Sungho Shin @ MIT
- Mihai Anitescu @ Argonne National Laboratory

And friends: Dominique Orban and JSO

What do we want to discuss today?

Observations:

- GPU-accelerated optimization is gaining momentum
- Most recent developments are using first-order methods (ADMM, PDLP)
- New generation of sparse linear solvers on the GPU (cuDSS)

Research question

Are second-order methods effective at solving large-scale LPs on the GPU?

Newest GPU-accelerated solvers for **LPs**:

- cuPDLP
- NVIDIA cuOPT
- cuClarabel
- MadIPM!

MadIPM — GPU example using JuMP

Available on github to solve your own LPs!

github.com/MadNLP/MadIPM.jl

```
using JuMP, MadIPM
using CUDA, KernelAbstractions, MadNLPGPU

c = rand(10)
model = Model(MadIPM.Optimizer)

# GPU settings
set_optimizer_attribute(model, "array_type", CuVector{Float64})
set_optimizer_attribute(model, "linear_solver", MadNLPGPU.
    CUDSSSolver)

@variable(model, 0 <= x[1:10], start=0.5)
@constraint(model, sum(x) == 1.0)
@objective(model, Min, c' * x)

JuMP.optimize!(model)
```

How to solve a LP with interior-point?

We define a LP in **standard form** as

$$\min_{x \in \mathbb{R}^n} \; c^\top x \quad \text{subject to} \quad Ax = b \, , \; x \geq 0 \, , \tag{LP}$$

with $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ problem's data

Any interior-point solver uses the three following items to solve (LP):

1. A good **pre-processing**
2. A Newton method to track the **central path**
3. A sparse linear solver to compute the **Newton direction**

The **Mehrotra's predictor-corrector** is the workhorse IPM method for (LP)

Mehrotra. On the implementation of a primal-dual interior point method. 1992.

Step 1: pre-processing

We reformulate (LP) in a suitable form for IPM (this step is **very** important to get a reliable convergence)

1. Call a pre-processing routine to remove fixed variables and dependent constraints
2. Scale the problem's data by applying the **Ruiz equilibration** method on the constraint matrix A
3. If appropriate, reformulate the problem in standard form
4. Find a good initial point using Mehrotra's heuristic
(push away from boundary to get long steps in the first iterations)

Andersen, Andersen. Presolving in linear programming. 1995.

KKT equations

We write the KKT equations for the LP in standard form

The primal-dual point $w := (x, y, z)$ is a solution of (LP) if and only if

$$
\begin{aligned}
A^\top y - z &= 0 \\
Ax - b &= 0 \\
0 \leq x &\perp z \geq 0
\end{aligned}
$$

For a barrier parameter $\mu > 0$, we say that w is on the central path if $(x, z) > 0$ and

$$
\begin{aligned}
A^\top y - z &= 0 \\
Ax - b &= 0 \\
XZe &= \mu e
\end{aligned}
$$

Primal-dual interior-point method

$\equiv$ Track the central path using Newton method

Wright. Primal-dual interior-point methods. 1997.

Step 2: Mehrotra predictor-corrector

We have to ensure that we track the central path

For a given primal-dual iterate $w = (x, y, z)$, define the **current barrier parameter** (average complementarity) as:

$$\mu = \frac{z^\top x}{n} .$$

Affine step: Compute Δ^{aff} by solving

$$\begin{bmatrix} 0 & A^\top & -I \\ A & 0 & 0 \\ Z & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x^{\text{aff}} \\ \Delta y^{\text{aff}} \\ \Delta z^{\text{aff}} \end{bmatrix} = - \begin{bmatrix} c + A^\top y - z \\ Ax - b \\ XZe \end{bmatrix} .$$

Corrector step: Compute Δ^{corr} by solving

$$\begin{bmatrix} 0 & A^\top & -I \\ A & 0 & 0 \\ Z & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x^{\text{corr}} \\ \Delta y^{\text{corr}} \\ \Delta z^{\text{corr}} \end{bmatrix} = - \begin{bmatrix} 0 \\ 0 \\ \sigma\mu e - \Delta Z^{\text{aff}} \Delta X^{\text{aff}} e \end{bmatrix} ,$$

with σ given by a heuristic

Update: For $\Delta^k = \Delta^{\text{aff}} + \Delta^{\text{corr}}$, set

$$w^{k+1} = w^k + \alpha^k \Delta^k$$

with α a step computed using a fraction-to-boundary rule

Mehrotra. On the implementation of a primal-dual interior point method. 1992.

Step 3: sparse linear solver

The affine step and the corrector step are both solving the **unsymmetric linear system**:

$$\begin{bmatrix} 0 & A^\top & -I \\ A & 0 & 0 \\ Z & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} .$$

Augmented KKT system

The unsymmetric system reduces to:

$$\begin{bmatrix} \Sigma & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_1 + X^{-1} r_3 \\ r_2 \end{bmatrix} ,$$

with the diagonal matrix $\Sigma := X^{-1} Z$.

Normal KKT system

We can also eliminate Δy to recover the positive-definite system:

$$\left(A \Sigma^{-1} A^\top \right) \Delta y = A \Sigma^{-1} \left(r_1 + X^{-1} r_3 \right) - r_2 .$$

Primal-dual regularization

It is the key for GPU performance!

Both the augmented and normal KKT systems have their own issues if:

- Free variables
- Rank-deficient Jacobian A
- Dense rows in A leads to a dense matrix $A\Sigma^{-1}A^\top$

Instead, we regularize the system using two small positive parameters $(\rho, \delta) > 0$.
The system becomes:

$$\begin{bmatrix} \Sigma + \rho I & A^\top \\ A & -\delta I \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_1 + X^{-1} r_3 \\ r_2 \end{bmatrix} .$$

The matrix is **symmetric quasi-definite** (SQD), meaning that it is strongly factorizable using a signed Cholesky factorization.

Proposition

This is equivalent to solve the **regularized LP**:

$$\min_{x,r} \; c^\top x + \frac{\rho}{2}\|x\|^2 + \frac{1}{2\delta}\|r\|^2$$
$$\text{subject to } Ax + r = b \; , \; x \geq 0$$

Altman, Gondzio. Regularized symmetric indefinite systems in interior point methods for linear and quadratic optimization. 1999.
Friedlander, Orban. A primal–dual regularized interior-point method for convex quadratic programs. 2012.
Gondzio. Matrix-free interior point method. 2012.

MadIPM: performance profile

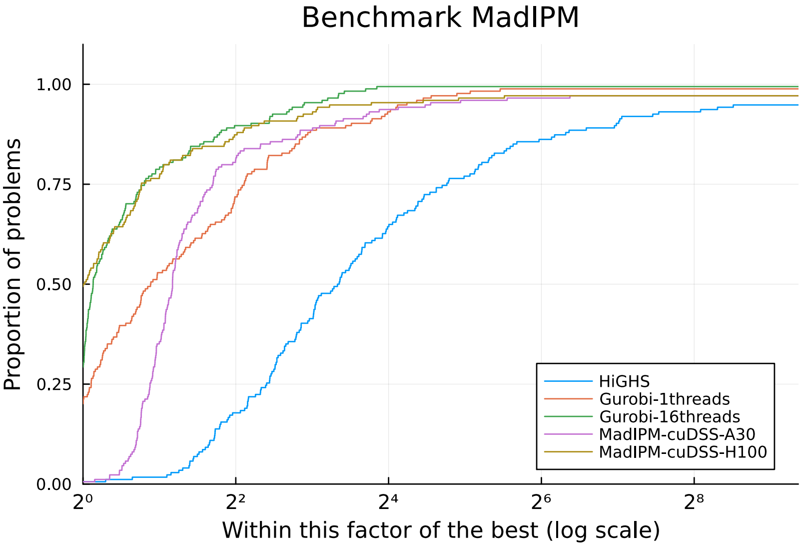


Figure: Benchmarking MadIPM, Gurobi and HiGHS on 174 large-scale LP instances from MIPLIB

The upper the better

MadIPM: raw performance

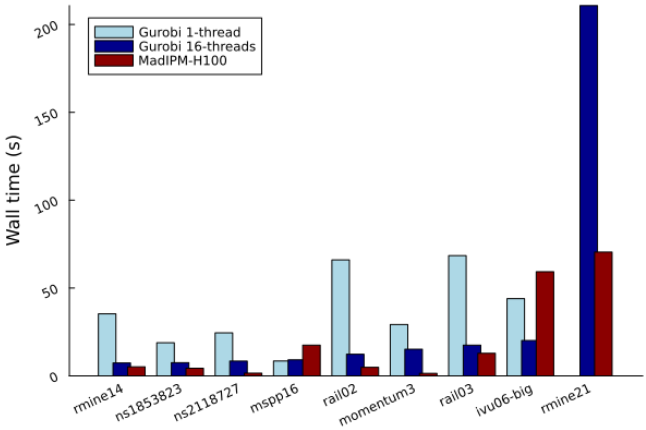


Figure: Zoom on the largest instances

The lower the better

How expensive should be your GPU?

Image courtesy of Sungho Shin

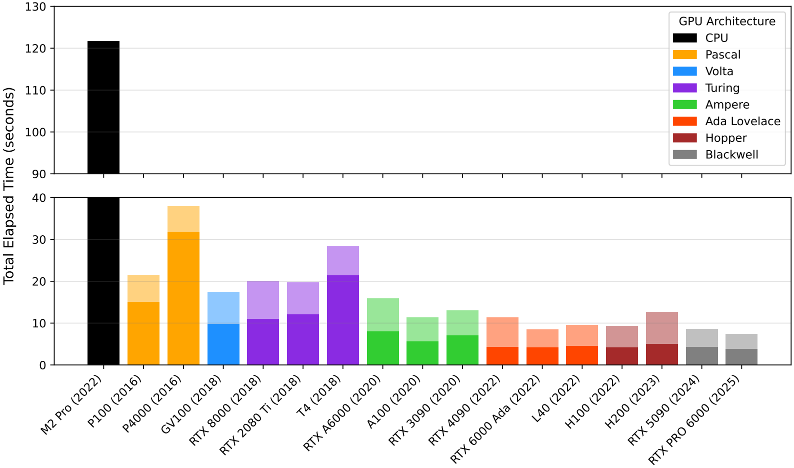


Figure: Time to optimality, here for a large-scale optimal power flow instance.

No need to buy a professional GPU for fast performance!

Conclusion

If you are interested, you can read the full article:

Montoison, Pacaud, Shin, Anitescu.
GPU Implementation of Second-Order Linear and Nonlinear Programming Solvers.
2025.

Next step

Solve LPs in **batch** on the GPU!

- Solve multiple LPs **simultaneously**, assuming the same KKT sparsity pattern
- Reuse **symbolic analysis** across all sparse linear systems
- Different central paths $\rightarrow$ real-time rebalancing when some systems converge earlier.

The one-billion dollars question

- How to solve large-scale MILPs on the GPU?