MadIPM

Alexis Montoison

François Pacaud, Sungho Shin, Mihai Anitescu

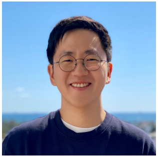
Argonne National Laboratory

November, 17^{th} 2025
JuMP-dev
Auckland

Who are we ?

https://madsuite.org/






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

What is MadSuite ?

MadSuite is a suite of open-source optimization software in **Julia** encompassing :

- algebraic modeling systems (ExaModels.jl)
- optimization solvers (MadIPM.jl, MadNLP.jl, MadNCL.jl)
- direct sparse linear solvers (CUDSS.jl)
- domain-specific modeling libraries (ExaModelsPower.jl)

We employ the latest advancements in **GPU computing**, to provide high-performance solutions for large-scale linear and nonlinear optimization problems.

What is MadSuite ?

- **MadNLP.jl** : A nonlinear programming solver based on the filter line-search interior point method (as in Ipopt) that can handle/exploit diverse classes of data structures, either on host or device memories.

- **MadIPM.jl** : It solves linear and convex quadratic programming. It implements the Mehrotra predictor-corrector method, leading to faster convergence than the default filter line-search algorithm used in MadNLP.

- **MadNCL.jl** : MadNCL.jl is another extension of MadNLP.jl. It combines Augmented Lagrangian method with IPM. It is particularly good at solving infeasible or degenerate nonlinear optimization problems.

MadIPM : Problem Formulation

$$\min_x c^\top x \quad \text{s.t.} \quad Ax = b, \;\; \ell \leq x \leq u$$

Lagrangian :

$$\mathcal{L}(x, y, z) = c^\top x + y^\top (Ax - b) - z_\ell^\top (x - \ell) + z_u^\top (x - u)$$

- KKT conditions define optimality.
- IPM reformulates complementarity constraints via barrier parameter $\mu > 0$.

For a given barrier parameter $\mu > 0$, IPM solves the system of nonlinear equations for $\ell < x < u$ and $z > 0$,

$$F_\mu(x, y, z) := \begin{bmatrix} c + A^\top y - z_\ell + z_u \\ Ax - b \\ X_\ell z_\ell - \mu e \\ X_u z_u - \mu e \end{bmatrix} = 0 \, ,$$

where $X_\ell := \mathrm{diag}(x - \ell)$ and $X_u := \mathrm{diag}(u - x)$.

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

$$\mu = \frac{z_\ell^\top (x - \ell) + z_u^\top (u - x)}{2n} .$$

Affine step : Compute Δ^{aff} by solving

$$\begin{bmatrix} 0 & A^\top & -I & I \\ A & 0 & 0 & 0 \\ Z_\ell & 0 & X_\ell & 0 \\ -Z_u & 0 & 0 & X_u \end{bmatrix} \begin{bmatrix} \Delta x^{\text{aff}} \\ \Delta y^{\text{aff}} \\ \Delta z_\ell^{\text{aff}} \\ \Delta z_u^{\text{aff}} \end{bmatrix} = - \begin{bmatrix} c + A^\top y - z_\ell + z_u \\ Ax - b \\ X_\ell z_\ell \\ X_u z_u \end{bmatrix} .$$

Corrector step : Compute Δ^{corr} using

$$\begin{bmatrix} 0 & A^\top & -I & I \\ A & 0 & 0 & 0 \\ Z_\ell & 0 & X_\ell & 0 \\ -Z_u & 0 & 0 & X_u \end{bmatrix} \begin{bmatrix} \Delta x^{\text{corr}} \\ \Delta y^{\text{corr}} \\ \Delta z_\ell^{\text{corr}} \\ \Delta z_u^{\text{corr}} \end{bmatrix} = - \begin{bmatrix} 0 \\ 0 \\ \sigma \mu e - \Delta Z_\ell^{\text{aff}} \Delta X^{\text{aff}} e \\ \sigma \mu e + \Delta Z_u^{\text{aff}} \Delta X^{\text{aff}} e \end{bmatrix} .$$

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

$$\begin{bmatrix} 0 & A^\top & -I & I \\ A & 0 & 0 & 0 \\ Z_\ell & 0 & X_\ell & 0 \\ -Z_u & 0 & 0 & X_u \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z_\ell \\ \Delta z_u \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix} .$$

The unsymmetric system reduces to the symmetric **augmented KKT system** :

$$\begin{bmatrix} \Sigma & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_1 + X_\ell^{-1} r_3 - X_u^{-1} r_4 \\ r_2 \end{bmatrix} ,$$

with the diagonal matrix $\Sigma := X_\ell^{-1} Z_\ell + X_u^{-1} Z_u$.

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

$$A \Sigma^{-1} A^\top \Delta y = A \Sigma^{-1} \left(r_1 + X_\ell^{-1} r_3 - X_u^{-1} r_4 \right) - r_2 .$$

MadIPM : challenges with sparse linear systems

The performance of the interior-point method depends on efficient linear solves. Key issues :

- Free variables ($\ell_i = -\infty, u_i = +\infty$) $\rightarrow$ $(\Sigma)_{ii} = 0$, singular matrix $\rightarrow$ treat separately.
- Rank-deficient Jacobian A (redundant constraints) $\rightarrow$ augmented/normal systems become singular.
- Dense rows in A $\rightarrow$ $A\Sigma^{-1}A^\top$ becomes dense $\rightarrow$ require special treatment (e.g., Sherman-Morrison).
- GPU-specific challenge : only augmented system handled; Schur complement is hard with dense columns and graph-based kernels.

We regularize the system using two small positive parameters $(\rho, \delta) > 0$. Once the primal-dual regularization is applied, the system becomes :

$$\begin{bmatrix} \Sigma + \rho I & A^\top \\ A & -\delta I \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_1 + X_\ell^{-1} r_3 - X_u^{-1} r_4 \\ r_2 \end{bmatrix} .$$

The matrix in the left-hand-side above is symmetric quasi-definite (SQD), meaning that it is strongly factorizable using a signed Cholesky factorization.

It is the key to performance on GPU !

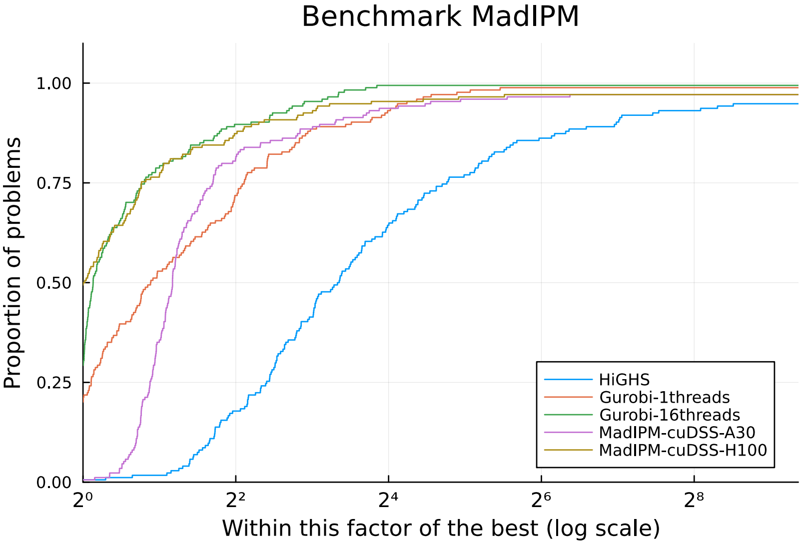


Figure – Benchmarking MadIPM, Gurobi and HiGHS on 174 large-scale LP instances from MIPLIB.

MadIPM

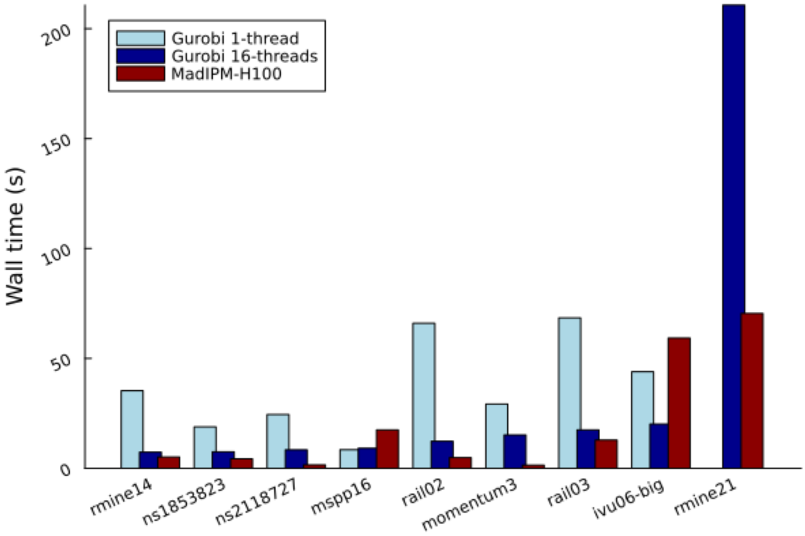


Figure – Benchmarking MadIPM and Gurobi.

MadIPM — CPU example

```julia
using JuMP
using MadIPM

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

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

JuMP.optimize!(model)
```

MadIPM — GPU example

```
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)
```

MadIPM on GPU

- Custom operators for sparse matrices in CSR format, optimized for GPU.
- Fraction-to-boundary linesearch redesigned for GPU efficiency.
- Preprocessing of the LPs / QPs (`QuadraticModels.jl`) still performed on CPU.

MadIPM – challenges

- We currenctly only handles the augmented system in `MadIPM.jl`.
- Implementing a GPU version of the Schur complement is hard : dense columns, graph-based kernels.
- Exploring more stable KKT formulations (Ghannad, Orban, Saunders 2022). Assuming bounds are $x \geq 0$, which leads to $\Sigma = X^{-1}Z$. By using $\Delta x = X^{1/2}\Delta\bar{x}$, we obtain a better-conditioned system :

$$\begin{pmatrix} Z & X^{1/2}A^T \\ AX^{1/2} & -\delta I \end{pmatrix} \begin{pmatrix} \Delta\bar{x} \\ \Delta y \end{pmatrix} = \begin{pmatrix} \bar{r}_1 \\ r_2 \end{pmatrix}$$

- Is it relevant to derive a similar variant of the Schur complement ?

Batch MadIPM

- **Goal :** Solve multiple LPs simultaneously, assuming the same KKT sparsity pattern.
- **Efficiency :** Reuse symbolic analysis across all sparse linear systems $\rightarrow$ GPU-friendly and SIMD-efficient.
- **API :** Through NLPModels.jl / MOI.jl, returns vectors of objective values, gradients, and non-zero entries of Jacobians and Hessians.
- **Parameter handling :** Smart management of JuMP / MOI parameters.
- **Applications :** DC Optimal Power Flow (DCOPF).
- **Challenge :** Different central paths $\rightarrow$ real-time rebalancing when some systems converge earlier.

Conclusion /next steps

- Collaboration with Georgia Tech (Michael Klamkin, Andrew Rosenberg)
- Batch MadIPM : faster multi-problem optimization
- Pave the way to batch MadNLP / MadNCL, require batch AD
- Applications : ACOPF with multiple loads, optimal control with multiple X0, ...