Accelerating optimization solvers on GPUs

François Pacaud
frapac.github.io

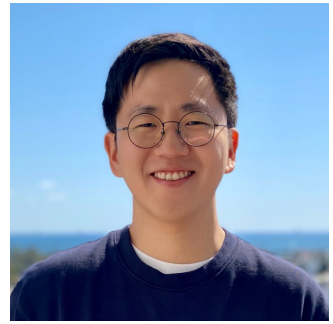
CAS, Mines Paris - PSL

GT optimisation
March 30, 2026

Who, what, why???

https://madsuite.org/





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

What do we do?

We prototype the next generation of optimization solvers

Research question

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

Programming on GPUs

```
for (int iy = 0; iy < 5; ++iy) {
    for (int ix = 0; ix < 20; ++ix) {
        for (int jy = 0; jy < 8; ++jy) {
            for (int jx = 0; jx < 16; ++jx) {
                foo(iy, ix, jy, jx);
            }
        }
    }
}
```

Figure: Programming on a CPU

```
__device__ void foo(int iy, int ix, int jy, int jx) {}

__global__ void mykernel() {
    int ix = blockIdx.x;
    int iy = blockIdx.y;
    int jx = threadIdx.x;
    int jy = threadIdx.y;
    foo(iy, ix, jy, jx);
}

int main() {
    dim3 dimGrid(20, 5);
    dim3 dimBlock(16, 8);
    mykernel<<<dimGrid, dimBlock>>>();
    cudaDeviceSynchronize();
}
```

Figure: Programming on a GPU

- CPUs have dozen of cores, while GPUs have **thousand** of them
- GPU are designed for **single instruction multiple data** (SIMD) patterns
- Many classical algorithms (including simplex) do not work well on GPUs!

Outline

Coming back to the basic: Newton method

Let $F : \mathbb{R}^n \to \mathbb{R}^n$ a smooth function. We aim to solve

$$F(x) = 0$$

Idea. For an iterate x_k, find a zero of the linear model:

$$F(x_k) + J_x F(x_k)(x - x_k) = 0$$

with $J_x F(x_k)$ Jacobian of F at x_k

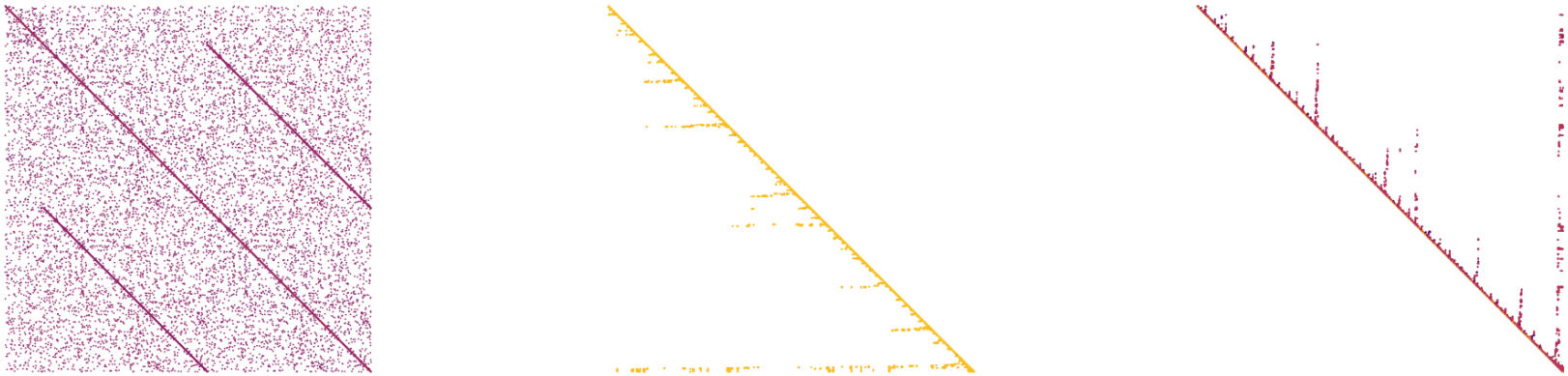
Newton method

Starting from x_0, proceed to the iterates

$$x_{k+1} = x_k - \left(J_x F(x_k)\right)^{-1} F(x_k)$$

- Require a non-singular Jacobian $J_x F(x_k)$;
- Quadratic convergence rate once we are in the basin of attraction;
- Require globalization outside the basin of attraction.

How to solve efficiently a linear system $Kx = b$?



K = L x U

Figure: LU decomposition of the power flow Jacobian for 1354pegase (computed using KLU)

Usual workflow

1. *Symbolic factorization:* Analyse the matrix **sparsity pattern** and find a **static ordering** that reduces the fill-in. Instantiate the sparse factors in memory
2. *Numerical factorization:* Compute inplace the nonzero values in L and U
3. *Backsolve:* Solve the system $L^{-1}b$ and $U^{-1}d$ to get solution of the linear system

Matrix decompositions: a hall of fame

Denote by P, Q pivoting matrices

K is generic

LU decomposition:

$$K = PLUQ$$

with L lower triangular and U upper triangular

K is symmetric definite positive (SDP)

Cholesky decomposition:

$$K = PLDL^\top P^\top$$

with L lower triangular and D diagonal

K is symmetric indefinite

Bunch-Kaufman factorization:

$$K = PQLBL^\top Q^\top P^\top$$

with L lower triangular and B block diagonal (with blocks of size 1×1 or 2×2)

Solving sparse linear system on the GPU

Observation

- Numerical pivoting Q is difficult to handle in parallel (and should be avoided at all cost on the GPU)
- Use Cholesky whenever you can!

NVIDIA cuDSS

Released in November 2023

- Symbolic factorization on the CPU, the rest on the GPU
- Implement sparse LU, Cholesky and signed-Cholesky
- Support batch solution of linear systems!

Outline

Linear programming

Linear program (LP) in standard form

For given data $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$, solve

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

KKT conditions

A variable x is solution if there exists multipliers (y, z) such that

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

where $0 \leq x \perp z \geq 0$ encodes the complementarity constraints

$$x_i \times z_i = 0 \quad \forall i = 1, \cdots, n$$

Primal-dual interior-point method

Central path

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

$$c + A^\top y - z = 0$$
$$Ax - b = 0$$
$$XZe = \mu e$$

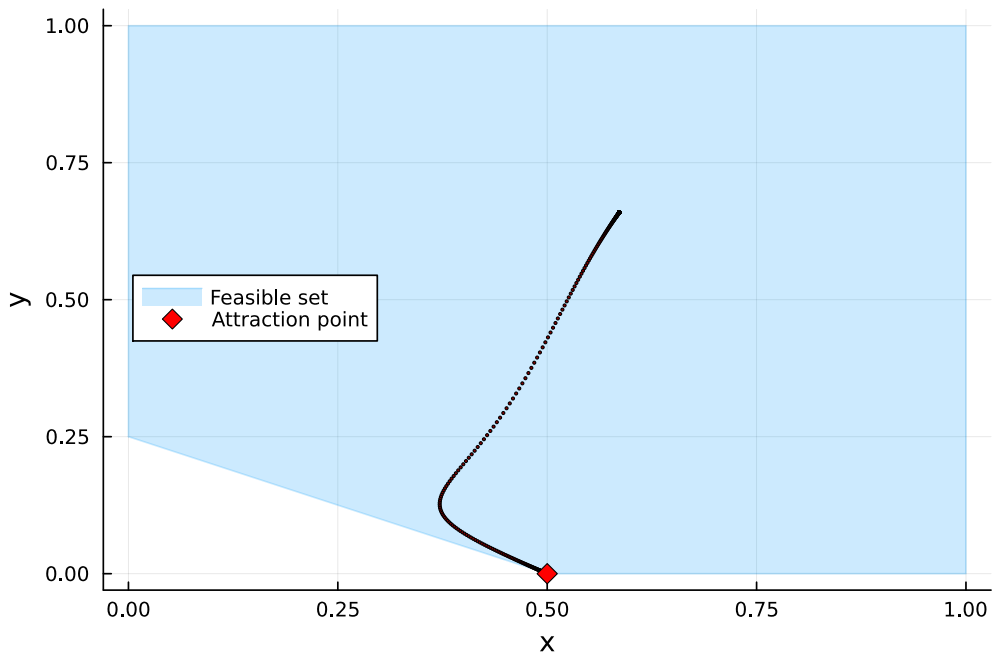


Figure: Track the central path using Newton method

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

Tracking the central path with the Mehrotra predictor-corrector

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.

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 $\Sigma := X^{-1}Z$.

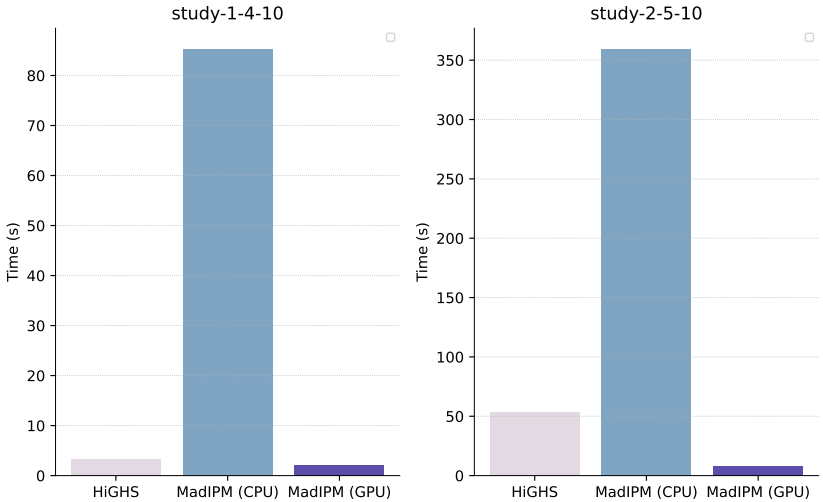
Normal KKT system

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

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

Practical results on large-scale LPs

Large-scale energy systems from RTE (Antares) [1]



	LP-1-4-10	LP-2-5-10
HiGHS	3.2	53.2
MadIPM (CPU)	85.2	359.0
MadIPM (GPU)	2.0	7.6

Observation

On the largest instance, 7x speed-up compared to HiGHS (dual-simplex)

[1] Courtesy of Thomas Bittar and Antoine Oustry @ RTE

Towards solving LPs in batch

Equivalent of ray tracing for LPs

Batch LPs

For a list of parameters $\{(c_i, b_i, A_i)\}_{i=1,\cdots,N}$ with similar sizes, solve

$$\begin{aligned} \min_{x_i \in \mathbb{R}^n} \quad & c_i^\top x_i \\ \text{subject to} \quad & A_i x_i = b_i \qquad \forall i = 1, \cdots, N \\ & x_i \geq 0 \end{aligned}$$

- The number N defines the batch sizes
- The solution of all LPs can proceed simultaneously
- The constraint matrices A_k should have the same sparsity pattern for efficient implementation

Solving DC-OPF LPs in batch

Recent results obtained by Michael Klamike @ Georgia Tech

batch size N	MadIPM (CPU)	MadIPM (GPU)	speedup
1	0.10	0.16	0.6x
2	0.20	0.16	1.2x
4	0.41	0.17	2.5x
8	0.81	0.17	4.7x
16	1.63	0.20	8.1x
32	3.31	0.24	13.8x
64	7.20	0.32	22.2x
128	15.05	0.49	30.5x

Table: Solve N LPs in parallel on the CPU (using multithreading) and on the GPU. Instance is 1354pegase, from the PGLIB-OPF benchmark

Observation

GPUs are very good at solving LPs in batch!

Outline

Nonlinear programming

Nonlinear program (NLP) in standard form

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad c(x) = 0\,,\ x \geq 0\,, \tag{NLP}$$

with $f(\cdot)$ and $c(\cdot)$ twice continuously differentiable functions

Most use-cases in power engineering, chemical engineering and optimal control

KKT conditions

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

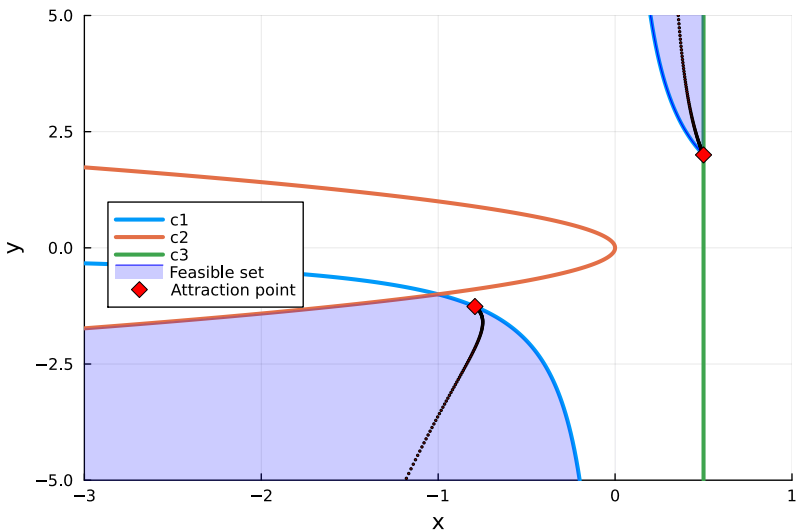
$$\begin{aligned} &\nabla f(x) + \nabla c(x)^\top y - z = 0 \\ &c(x) = 0 \\ &0 \leq x \perp z \geq 0 \end{aligned}$$

Primal-dual interior-point method

Homotopy continuation

For a barrier parameter $\mu > 0$, IPM aims to solve:

$$\nabla f(x) + \nabla c(x)^\top y - z = 0$$
$$c(x) = 0$$
$$XZe = \mu e \ , \ (x, z) > 0$$



$$\begin{aligned} \min \quad & 100\left(-x1^2 + x2\right)^2 + (x1 - 1)^2 \\ \text{subject to} \quad & x1 \times x2 \geq 1 \\ & x2^2 + x1 \geq 0 \\ & x1 \leq 0.5 \end{aligned}$$

Figure: Multiple solutions, multiple central paths

Filter line-search IPM (ala Ipopt)

Barrier update: For a given primal-dual iterate $w = (x, y, z)$, update the barrier parameter using a *monotone* rule.

Sensitivities: Compute the Jacobian $J := \nabla c(x)^\top$ and the Hessian $W := \nabla^2_{xx} L(x, y)$ using *sparse* automatic differentiation

Newton system: Compute the descent direction Δw by solving

$$\begin{bmatrix} W + \rho I & J^\top & -I \\ J & 0 & 0 \\ Z & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = - \begin{bmatrix} \nabla f(x) + J^\top y - z \\ c(x) \\ XZe \end{bmatrix} .$$

with ρ a regularization parameter ensuring that

$$Z^\top (W + \rho I) Z \succ 0 \quad \text{for} \quad Z \text{ a basis for } \mathrm{Ker}(J)$$

(ρ usually computed using an *inertia-correction* procedure)

Next iterate: Set

$$w_{k+1} = w_k + \alpha_k \Delta w$$

with α_k a step computed using a filter line-search

Again, we rely on sparse linear solvers

This time, the **unsymmetric linear system** writes as:

$$\begin{bmatrix} W + \rho I & J^\top & -I \\ J & 0 & 0 \\ Z & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}$$

Augmented KKT system

The unsymmetric system reduces to:

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

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

- By default, most solvers look at the *symmetric indefinite* augmented KKT system
- Require using a LBL factorization with numerical pivoting...
- Can we get an equivalent of the normal KKT system for NLPs?

Golub & Greif: Reformulating as a positive definite system

Idea: augmented Lagrangian reformulation

For $\gamma > 0$, the augmented KKT system is equivalent to

$$\begin{bmatrix} W_\gamma & J^\top \\ J & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = - \begin{bmatrix} r_1 + X^{-1} r_3 + \gamma J^\top r_2 \\ r_2 \end{bmatrix}$$

with $W_\gamma := W + \Sigma + \rho I + \gamma J^\top J$

- Suppose LICQ hold and the reduced Hessian is positive definite:
  Then for γ large-enough the matrix W_γ is positive definite
- The condensed KKT system reduces to the normal equations:

$$J W_\gamma^{-1} J^\top \Delta y = r_2 - W_\gamma^{-1} \gamma$$

  But W_γ^{-1} is likely dense!
- Keep W_γ^{-1} implicit by solving the normal equations *iteratively* with a conjugate gradient (CG) algorithm!
- For large γ, CG converges in few iterations

Regev et al., HyKKT: a hybrid direct-iterative method for solving KKT linear systems. 2023.

Results on large-scale AC-OPF instances

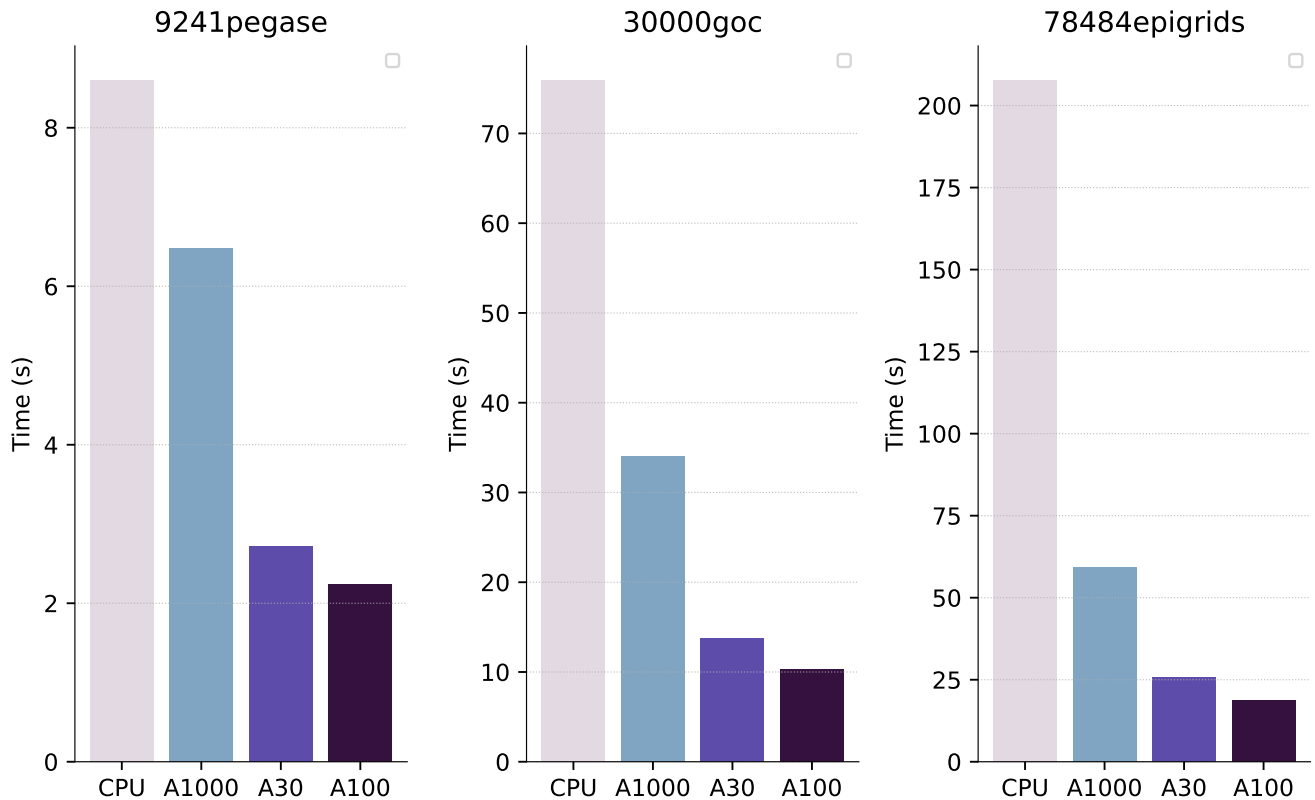


Figure: Testing the solutions of large-scale OPFs with different GPUs

Results on large-scale SC-OPF instances

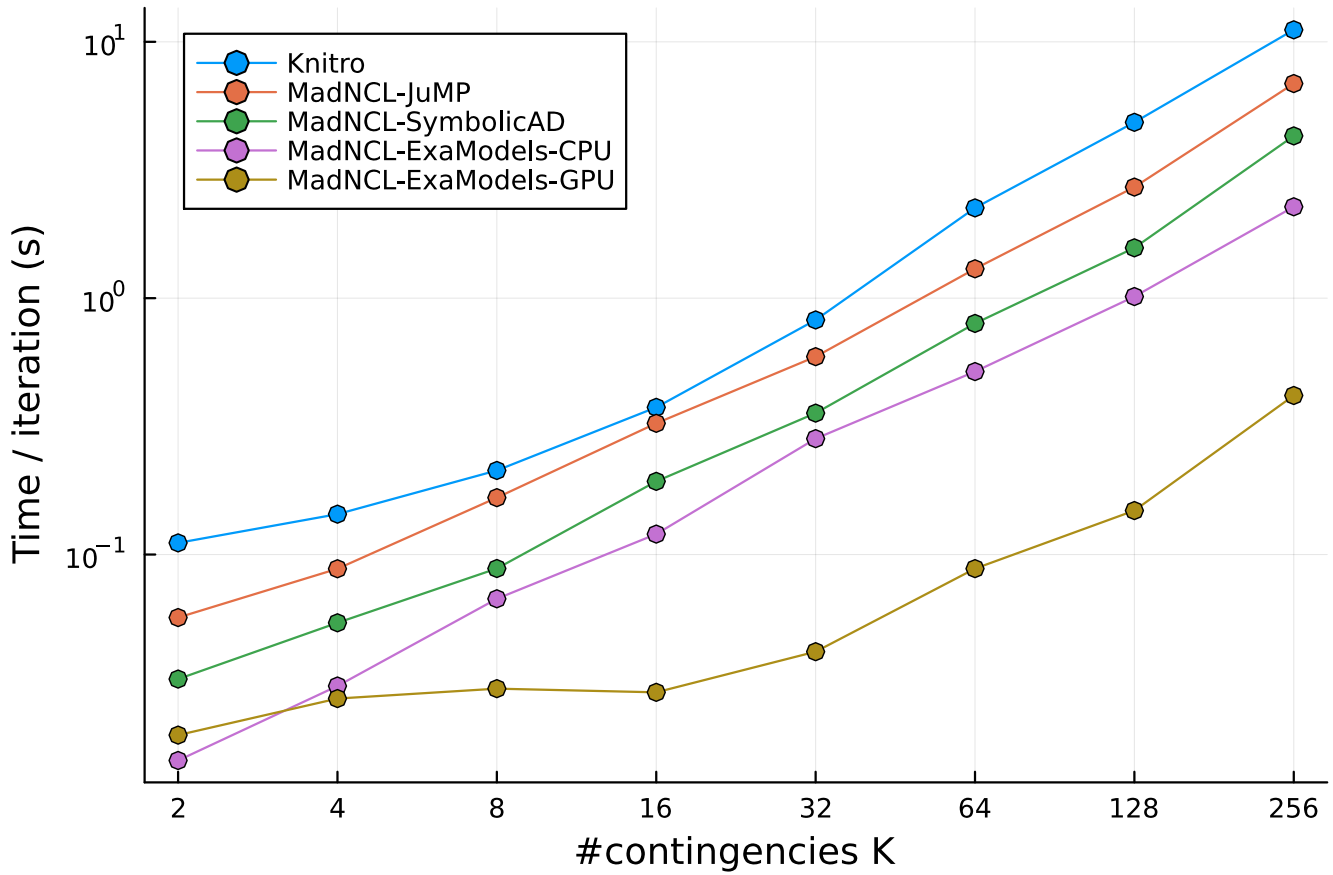


Figure: Solution time as we increase the number of contingencies K

Conclusion

Take-away

- Yes, Newton method is practical on the GPU!
- Sparse direct linear solvers are key
- With GPUs, you can expect up to x10-x100 speed-up on difficult instances, but not always!

Next step

Solve optimization problems in **batch** on the GPU!

- Reuse **symbolic analysis** across all sparse linear systems
- Different central paths $\rightarrow$ real-time rebalancing when some systems converge earlier.
- Exploit low-rank structure in solution matrix?