GPU-accelerated dynamic nonlinear optimization with ExaModels and MadNLP

François Pacaud **Sungho Shin**

CAS, Mines Paris - PSL & Chemical Engineering Department - MIT

CDC 2024
December, 18th 2024

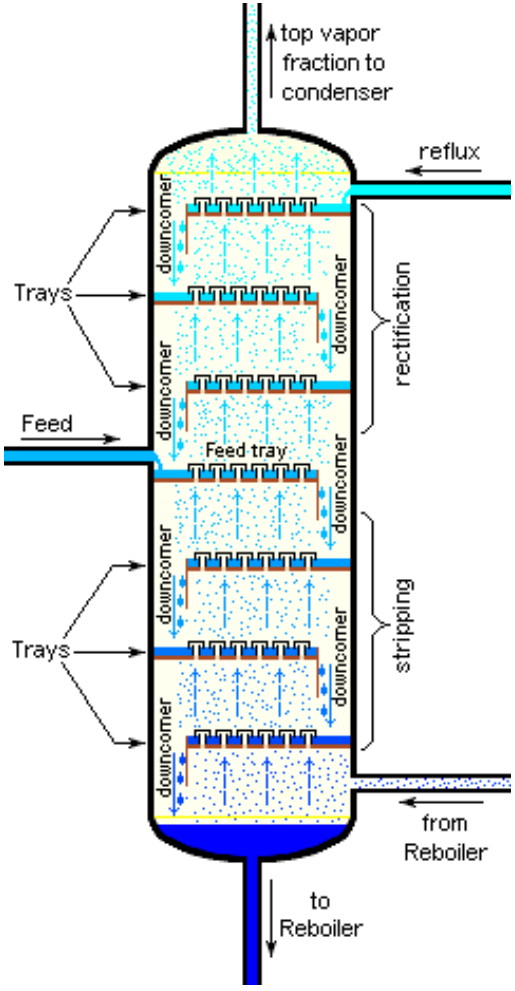

Figure: Image source: Wikipedia

Outline: dynamic nonlinear optimization

1. We present a tractable method to port the interior-point method on GPU
2. We leverage the newly released cuDSS linear solver for optimal performance
3. We study the performance of the method on the classical distillation column instance

Context

How to solve a dynamic nonlinear optimization problem?

1. Discretize the continuous dynamics
2. Formulate the problem in a modeler (casadi, JuMP, AMPL,...)
3. Solve it using a nonlinear solver (Ipopt, Knitro,...)

Pioneered by the chemical engineering community:

Fluid Mechanics and Transport Phenomena | Full Access

Large-scale DAE optimization using a simultaneous NLP formulation

A. Cervantes, L. T. Biegler

Ind. Eng. Chem. Res. **2004**, *43*, 6803−6812 **6803**

Nonlinear Optimization with Many Degrees of Freedom in Process Engineering

Maame Yaa B. Poku and Lorenz T. Biegler*

Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania 15123

Jeffrey D. Kelly

Honeywell Industrial Solutions, 300 Yorkland Boulevard, Toronto, Ontario M2J 1S1, Canada

Example: optimization of a distillation column

$$\min \sum_{t=1}^{N} \left(\gamma (x_{1,t} - \bar{x}_1)^2 + \rho (u_t - \bar{u})^2 \right)$$

subject to, for all $t = 1, \cdots, N$ and for a fixed time-step $\Delta t := 10/N$,

$$L_t = u_t D \,, \;\; V_t = L_t + D \,, \;\; S_t = F + L_t \,,$$

$$y_{n,t} = \frac{\alpha x_{n,t}}{1 + (\alpha - 1) x_{n,t}} \,, \quad \forall n \in \{1, .., 32\} \,,$$

$$\dot{x}_{1,t} = \frac{1}{M_1} V_t (y_{2,t} - x_{1,t})$$

$$\dot{x}_{n,t} = \frac{1}{M_n} \left(L_t (x_{n-1,t} - x_{n,t}) - V_t (y_{n,t} - y_{n+1,t}) \right)$$

$$\forall n \in \{2, .., 16\} \,,$$

$$\dot{x}_{17,t} = \frac{1}{M_{17}} \left(F x_f + L_t x_{16,t} - S_t x_{17,t} - V_t (y_{17,t} - y_{18,t}) \right)$$

$$\dot{x}_{n,t} = \frac{1}{M_n} \left(S_t (x_{n-1,t} - x_{n,t}) - V_t (y_{n,t} - y_{n+1,t}) \right)$$

$$\forall n \in \{18, .., 31\} \,,$$

$$\dot{x}_{32,t} = \frac{1}{M_{32}} \left(S_t (x_{31,t} - (F - D_t) x_{32,t} - V_t y_{32,t} \right)$$

$$\dot{x}_{n,t} = \frac{1}{\Delta t} \left(x_{n,t} - x_{n,t-1} \right) \,, \quad \forall n \in \{1, .., 32\} \,,$$

$$x_{n,0} = \bar{x}_{n,0} \,, \;\; 1 \leq u_t \leq 5 \,,$$

We rewrite the distillation column instance as a nonlinear program

n variables, m inequality constraints, p equality constraints

Continuous nonlinear problems

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{cases} g(x) = 0 \\ h(x) \leq 0 \end{cases}$$

Objective

Equality cons.

Inequality cons.

The functions f, g, h are smooth, *possibly nonconvex*

- Useful framework to solve practical engineering problems
- Usually, we are interested only at finding a *local optimum*
- Mature solvers exist since the 2000s (Ipopt, Knitro, LOQO)

J. Nocedal, SJ. Wright. Numerical optimization.

We rewrite the distillation column instance as a nonlinear program

n variables, m inequality constraints, p equality constraints

Continuous nonlinear problems

Objective

$$\min_{x \in \mathbb{R}^n, s \in \mathbb{R}^m} f(x) \quad \text{subject to} \quad \begin{cases} g(x) = 0 \\ h(x) + s = 0\,, \quad s \geq 0 \end{cases}$$

Equality cons.

Slack

The functions f, g, h are smooth, *possibly nonconvex*

- Useful framework to solve practical engineering problems
- Usually, we are interested only at finding a *local optimum*
- Mature solvers exist since the 2000s (Ipopt, Knitro, LOQO)

J. Nocedal, SJ. Wright. Numerical optimization.

Interior-point method (IPM)

KKT stationary equations

$$\begin{cases} \nabla f(x) + \nabla g(x)^\top y + \nabla h(x)^\top z = 0 \\ z - \nu = 0 \\ g(x) = 0 \\ h(x) + s = 0 \\ 0 \le s \perp \nu \ge 0 \end{cases}$$

Complementarity cons

Rewrite the (nonsmooth) KKT system as a *smooth* nonlinear system

Dual variables

$$F_\mu(x, s; y, z, \nu) := \begin{bmatrix} \nabla f(x) + \nabla g(x)^\top y + \nabla h(x)^\top z \\ z - \nu \\ g(x) \\ h(x) + s \\ S\nu - \mu e \end{bmatrix} = 0$$

Homotopy, $S = \operatorname{diag}(s)$

Primal-dual interior-point method

Solve $F_\mu(x, s; y, z, \nu) = 0$ using Newton method while driving $\mu \to 0$.

Newton method

At iteration k,

1. Compute Newton step d^k as solution of the linear system

$$\nabla F_\mu(w^k)\, d^k = -F_\mu(w^k)$$

2. Update the primal-dual variable $w^k := (x^k, s^k, y^k, z^k, \nu^k)$ as

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

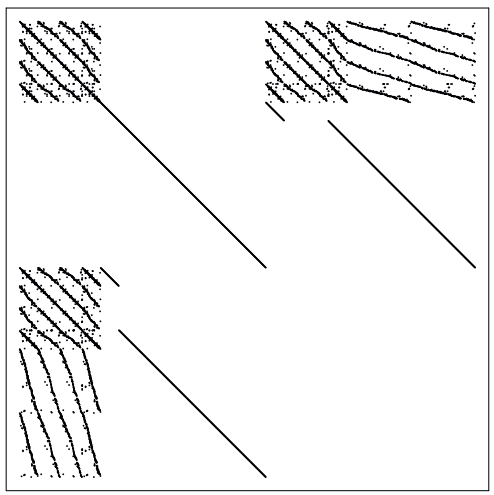

Figure: ∇F_μ

Augmented KKT system

After (slight) reformulation, the Newton step writes as

$$\begin{bmatrix} W & 0 & \nabla g^\top & \nabla h^\top \\ 0 & \Sigma_s & 0 & I \\ \nabla g & 0 & 0 & 0 \\ \nabla h & I & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ d_y \\ d_z \end{bmatrix} = - \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix}$$

with $W = \nabla^2_{xx} L(\cdot)$, $\Sigma_s = S^{-1}\mathrm{diag}(\nu)$

MadNLP: a structure exploiting interior-point solver

Winner of the 2023 COIN-OR cup!

MadNLP


```julia
using MadNLP, MadNLPTests
model = MadNLPTests.HS15Model()
solver = MadNLPSolver(model)
MadNLP.solve!(solver)
```

Fork on github!

https://github.com/MadNLP/MadNLP.jl/

https://github.com/exanauts/ExaModels.jl

MadNLP

- Written in pure Julia
- Filter line-search (ala Ipopt)
- Flexible & Modular

- ✓ CUDA-compatible
- ✓ MPI-compatible
- ✓ Interfaced with the vectorized modeler ExaModels.jl
- ✓ And now interfaced with Casadi, thanks to Tommaso Sartor!

Porting IPM to the GPU

1. Fast evaluation of the derivatives using ExaModels.jl
2. Fast linear solves using NVIDIA cuDSS

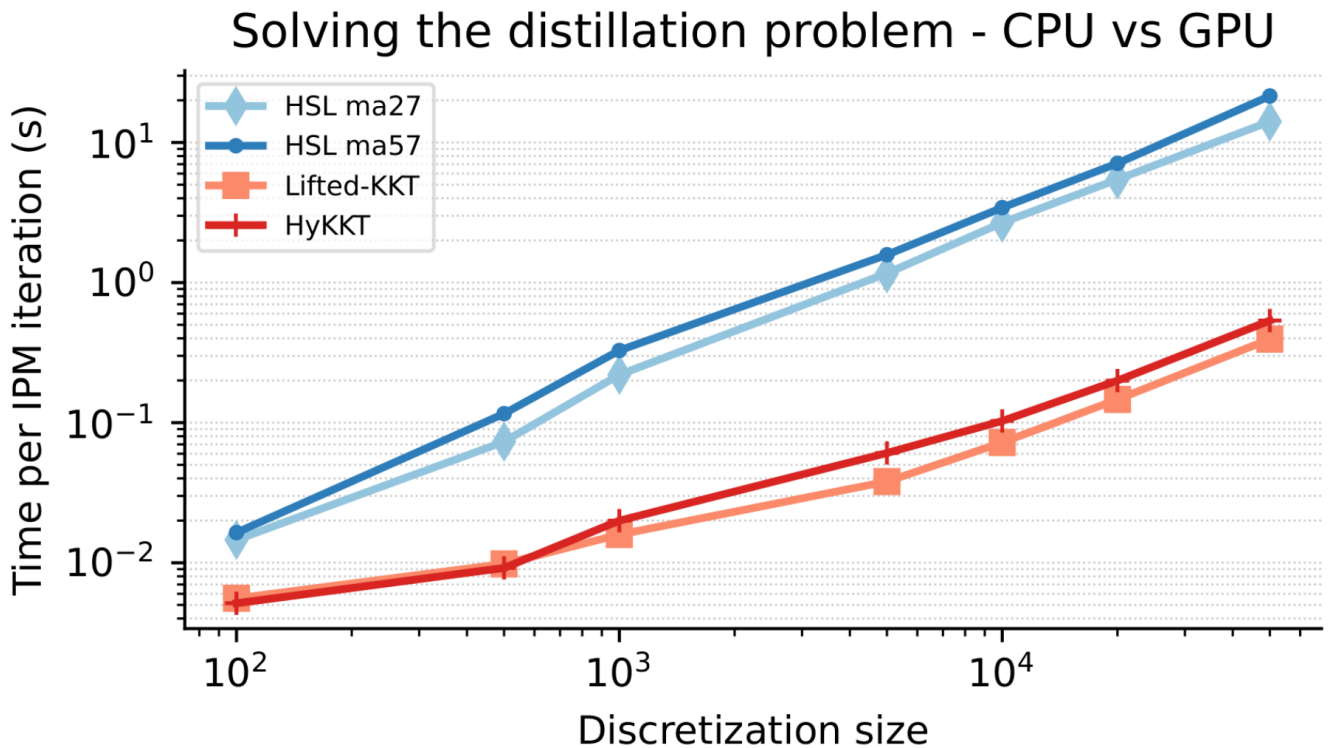

Figure: Time per IPM iteration (s), CPU versus GPU. Log-log scale.

F. Pacaud, S. Shin, A. Montoison, M. Schanen, M. Anitescu. "Condensed-space methods for nonlinear programming on GPUs."

First step: factorable programming with ExaModels.jl

- Large-scale optimization problems **almost always have repetitive patterns**

$$\min_{x^\flat \le x \le x^\sharp} \sum_{l \in [L]} \sum_{i \in [I_l]} f^{(l)}(x; p_i^{(l)}) \qquad \textbf{(SIMD abstraction)}$$

$$\text{subject to } \left[g^{(m)}(x; q_j) \right]_{j \in [J_m]} + \sum_{n \in [N_m]} \sum_{k \in [K_n]} h^{(n)}(x; s_k^{(n)}) = 0, \quad \forall m \in [M]$$

- Repeated patterns are made available by specifying the models as **iterable objects**

```
constraint(c, 3 * x[i+1]^3 + 2 * sin(x[i+2]) for i = 1:N-2)
```

- **For each repeatitive pattern**, the derivative evaluation kernel is constructed & compiled, and **executed in parallel over multiple data**

Second step: Solving the KKT system on the GPU

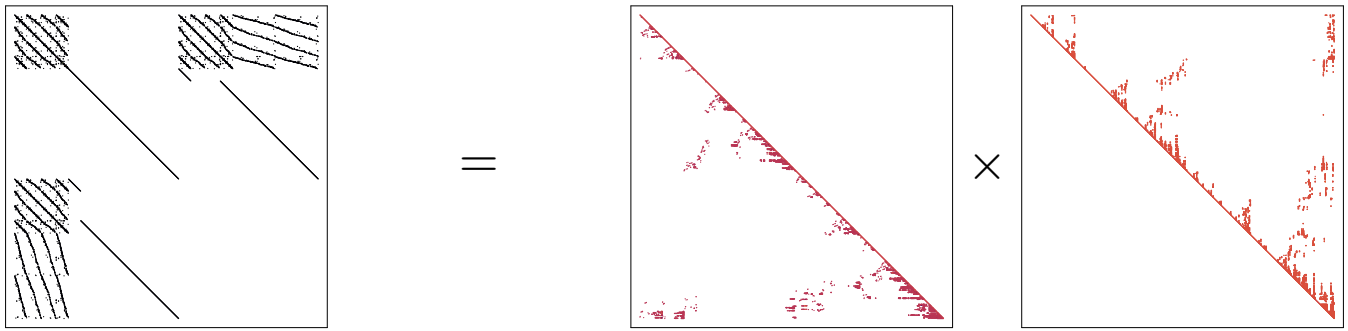

Figure: Matrix factorization using a direct solver

Linear solve: Solve the KKT system $\nabla F_\mu d_k = -F_k$

- Usually require factorizing ∇F_μ (symmetric indefinite: LBL)
- KKT system is highly *ill-conditioned* $\rightarrow$ numerical pivoting

Challenge: solving the sparse linear system on the GPU

- Ill-conditioning of the KKT system
  (= iterative solvers are often not practical)
- Direct solver requires **numerical pivoting** for stability
  (= difficult to parallelize)

Strategy 1: LiftedKKT

Idea: equality relaxation

For a $\tau > 0$ small enough, solve the relaxed problem

$$\min_{x \in \mathbb{R}^n} \; f(x) \quad \text{subject to} \quad \begin{cases} -\tau \leq g(x) \leq \tau \\ h(x) \leq 0 \end{cases}$$

Reformulating the problem with slack variables:

$$\min_{x \in \mathbb{R}^n, s \in \mathbb{R}^{m+p}} \; f(x) \quad \text{subject to} \quad h^\tau(x) + s = 0 \;,\; s \geq 0$$

with $h^\tau(x) = (g(x) - \tau,\; -g(x) - \tau,\; h(x))$

Condensed KKT system

The augmented KKT system is equivalent to

$$K_\tau \; d_x = -r_1 + (\nabla h^\tau)^\top (\Sigma_s r_4 + r_2)$$

with the *condensed matrix* $K = W + (\nabla h^\tau)^\top \Sigma_s (\nabla h^\tau)$.

$\rightarrow$ the condensed KKT system can be solved without numerical pivoting!

Strategy 2: HyKKT (aka Golub & Greif method)

Idea: augmented Lagrangian reformulation

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

$$\begin{bmatrix} K_\gamma & \nabla g^\top \\ \nabla g & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} = - \begin{bmatrix} w_1 + \gamma \nabla g^\top w_2 \\ w_2 \end{bmatrix}$$

with $K_\gamma = K + \gamma \nabla g^\top \nabla g$

- ☑ For γ large-enough the matrix K_γ is positive definite
  ☞solve the condensed KKT system using the normal equations:

$$(\nabla g) K_\gamma^{-1} (\nabla g)^\top d_y = w_2 - K_\gamma^{-1} (w_1 + \gamma \nabla g^\top w_2)$$

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

S. Regev et al., "HyKKT: a hybrid direct-iterative method for solving KKT linear systems." Optimization Methods and Software 38, no. 2 (2023)

How does it compare with state-of-the-art?

Table: Performance comparison of MadNLP on CPU and GPU

	HSL ma27				Lifted-KKT				HyKKT			
N	init	AD	linsolve	**total**	init	AD	linsolve	**total**	init	AD	linsolve	**total**
1,000	0.1	0.0	1.7	**1.8**	0.5	0.0	0.4	**0.9**	0.4	0.0	0.2	**0.6**
10,000	1.6	0.2	20.6	**22.4**	4.8	0.0	0.9	**5.8**	4.9	0.0	0.8	**5.7**
50,000	15.4	0.9	109.1	**125.5**	27.9	0.5	5.4	**33.8**	29.7	0.1	4.6	**34.5**

- init: Pre-processing
- AD: automatic differentiation
- linsolve: numerical factorization
- total: total solving time

Observations

- Initial symbolic analysis is expensive
- The time per IPM iterations is reduced by x10
- More effective for large-scale problems!

Take-away

1. Large-scale optimization is practical on modern GPU hardware
2. NVIDIA cuDSS could be a game changer for your own application!

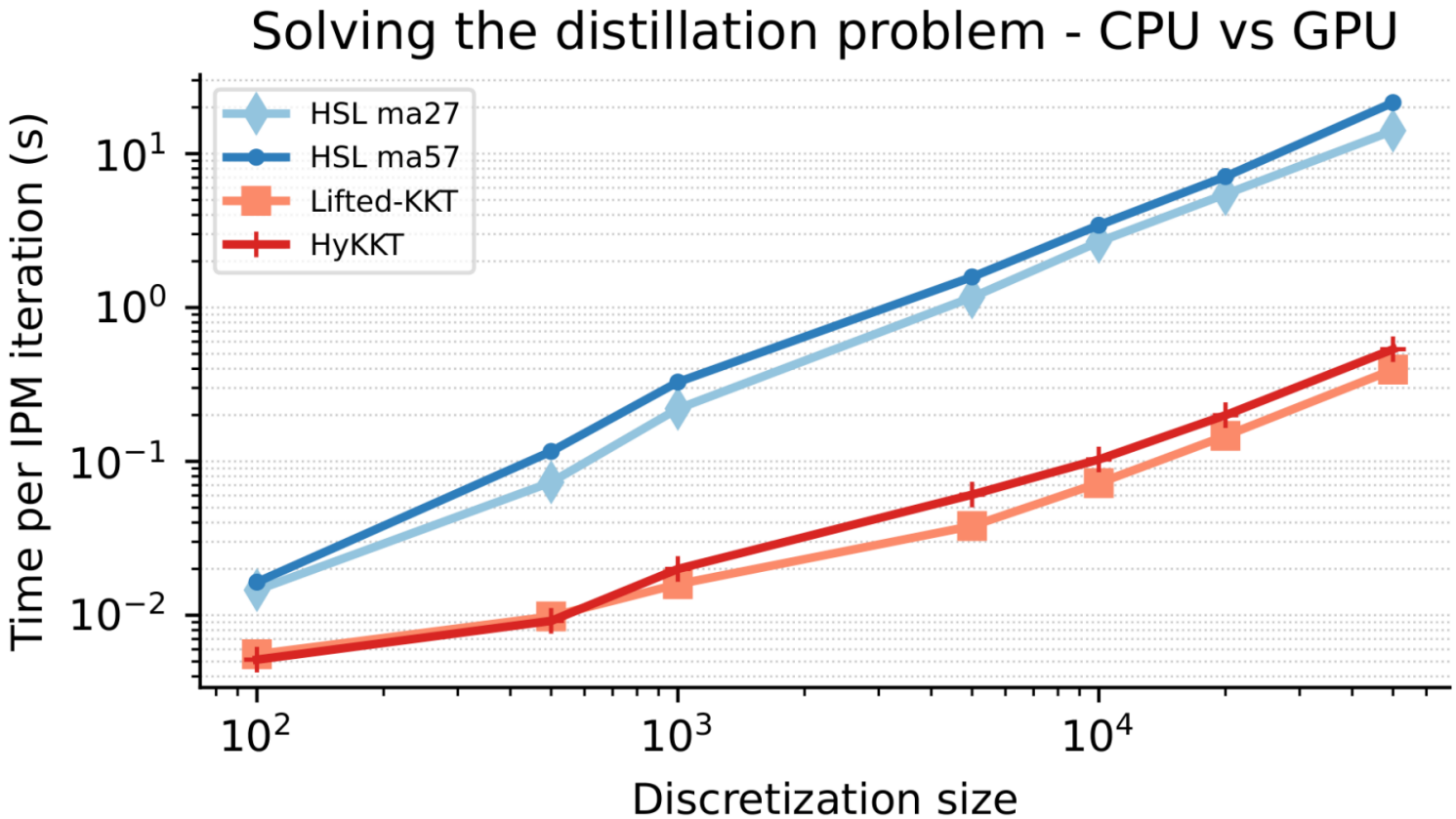

Figure: Time per IPM iteration (s), CPU versus GPU. Log-log scale.