GPU-accelerated nonlinear programming

Mihai Anitescu [1] **François Pacaud** [2] **Sungho Shin** [1]

[1]MCS, Argonne National Laboratory [2]CAS, Mines Paris - PSL

Autonomous Systems Workshop, Golden 2024

Who are we?

- An international team looking at the future of nonlinear programming

- Development of a nonlinear optimization solver: MadNLP.jl
  - Winner of the 2023 COIN-OR cup!

MadNLP: a structure exploiting interior-point solver

MadNLP

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

MadNLP

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

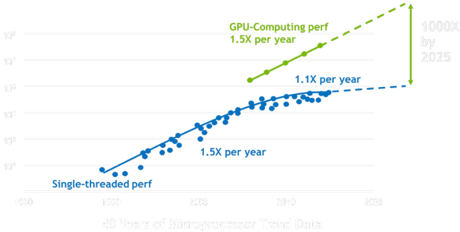
✓ CUDA-compatible

✓ MPI-compatible

Open-source:
https://github.com/MadNLP/MadNLP.jl/

Why GPUs?

- End of Moore's Law



- GPUs power AI and scientific computing (fluid, climate, bioinformatics)
- The newest generation of supercomputers are using GPUs

Source of the figure: NVIDIA

Outline

Nonlinear programming

GPU-accelerated automatic-differentiation

GPU-accelerated KKT linear solvers

Nonlinear programming: a reminder

n variables, m inequality constraints, p equality constraints

Continuous nonlinear problems

Objective

Equality cons.

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

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.

Nonlinear programming: a reminder

n variables, m inequality constraints, p equality constraints

Continuous nonlinear problems

Objective — Equality cons. — Slack

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

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

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

Dual variables

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

Complementarity cons., $S = \mathrm{diag}(s)$

Interior-point method

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

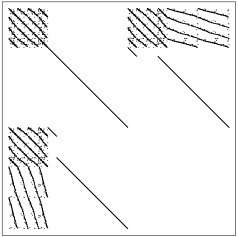


Figure: ∇F_μ

Augmented KKT system

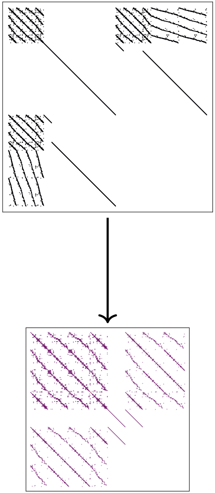
At iteration k, solve the Newton step $(\nabla F_\mu) d_k = -F_k$

∇F_μ

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

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

Condensed KKT system



Condensed KKT system

The augmented KKT system is equivalent to

$$\begin{bmatrix} K & \nabla g^\top \\ \nabla g & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} = - \begin{bmatrix} r_1 + (\nabla h)^\top (\Sigma_s r_4 + r_2) \\ r_3 \end{bmatrix}$$

with the *condensed matrix* $K = W + \nabla h^\top \Sigma_s \nabla h$.
We recover (d_s, d_z) as

$$d_s = -\Sigma_s^{-1}(r_3 + d_y) \, , \quad d_z = \Sigma_s (\nabla h \, d_x - r_4) - r_2 \, .$$

- Additional fill-in compared to augmented KKT system...
- Useful when the number of inequality constraints m is large

Identifying the computational bottlenecks

How to solve the Newton step?

$$(\nabla F_{\mu})d_k = -F_k$$

Two computational bottlenecks:

1. **Evaluate derivatives** and assemble KKT matrix ∇F_{μ}
2. **Solve KKT system** $\nabla F_{\mu} d_k = -F_k$

Outline

Nonlinear programming

GPU-accelerated automatic-differentiation

GPU-accelerated KKT linear solvers

Evaluating derivatives on the GPU

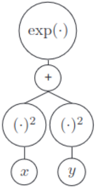


Figure: Expression tree for $\exp(x^2 + y^2)$ (credit: JuMP.jl)

Derivatives: Evaluate ∇F_μ requires Jacobian and Hessian
- Rely on *automatic differentiation* (AD)
- Usually we formulate the nonlinear program inside a *modeler*, computing automatically the derivatives using the expression tree
- **Software:** AMPL, GAMS, Pyomo, JuMP (all designed for CPU)

Challenge: evaluating sparse derivatives on the GPU

- GPU-accelerated AD frameworks already exist (Torch, Tensorflow, jax)
- But none of them have full support for **sparse** and **second-order**

A. Griewank, A. Walther. Evaluating derivatives: principles and techniques of algorithmic differentiation. SIAM, 2008.
I. Dunning, J. Huchette, M. Lubin. "JuMP: A modeling language for mathematical optimization." SIAM review 59, no. 2 (2017).

ExaModels.jl: a prototype for sparse automatic differentiation on GPU

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

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

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

- Repeated patterns are made available by always 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**

Observation

ExaModels.jl is effective at evaluating the derivatives of practical nonlinear problems (e.g. optimal power flow)

S. Shin, F. Pacaud, and M. Anitescu. *Accelerating optimal power flow with GPUs: SIMD abstraction of nonlinear programs and condensed-space interior-point me*

Outline

Nonlinear programming

GPU-accelerated automatic-differentiation

GPU-accelerated KKT linear solvers

Solving the KKT system on the GPU

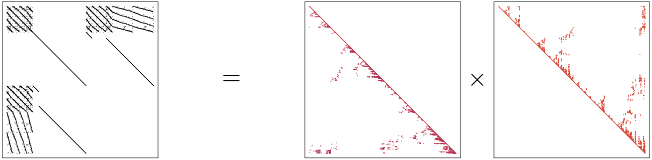


Figure: Matrix factorization using a direct solver

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

- Usually require factorizing ∇F_μ (convex: Cholesky, nonconvex: LBL)
- KKT system is highly *ill-conditioned* $\to$ numerical pivoting
- **Software:** HSL, Pardiso

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 numerical stability, an operation difficult to parallelize

B. Tasseff, C. Coffrin, A. Wächter, C. Laird. "Exploring benefits of linear solver parallelism on modern nonlinear optimization applications.", 2019

Roadmap

Solution 1: Densification

- Reduce the KKT system down to a dense matrix
- Akin to a null-space method (also known as reduced Hessian)
- Works well if the number of degrees of freedom is small

Solution 2: Condensation

- Reduce the KKT system to a sparse positive definite matrix
- Sparse Cholesky is stable without numerical pivoting
  $\rightarrow$ runs in parallel on the GPU (`cuDSS`)
- More versatile approach

Solution 1: Densification

- Split the decision variables into independent (=control) and dependent variables (=states)
- Reduce the KKT system to a dense matrix by eliminating the state variables

Problem with a physical structure

- u: control (=degrees of freedom)
- x: state

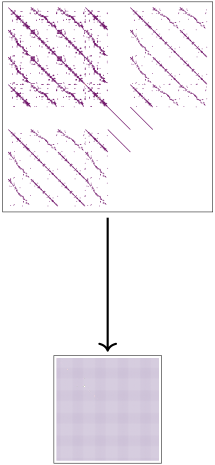
Physical cons.

$$\min_{x,u} \; f(x,u) \quad \text{s.t.} \quad \begin{cases} g(x,u) = 0 \\ h(x,u) \leq 0 \end{cases}$$

Operational cons.

L. Biegler, J. Nocedal, C. Schmid. "A reduced Hessian method for large-scale constrained optimization." SIAM Journal on Optimization 5, no. 2 (1995)

Null-space strategy



We can exploit the structure in the condensed KKT system (=split x from u)

$$\begin{bmatrix} K_{uu} & K_{ux} & G_u^\top \\ K_{xu} & K_{xx} & G_x^\top \\ G_u & G_x & 0 \end{bmatrix} \begin{bmatrix} d_u \\ d_x \\ d_y \end{bmatrix} = - \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}$$

Reduced KKT system

If the Jacobian G_x is *invertible*, then the condensed KKT system is equivalent to

$$\hat{K}_{uu} d_u = -r_1 + G_u^\top G_x^{-\top} r_2 + K_{ux} G_x^{-1} r_3$$

The reduced matrix $\hat{K}_{uu} \in \mathbb{R}^{n_u \times n_u}$ is *dense* and satisfies

$$\hat{K}_{uu} = \begin{bmatrix} I \\ -G_x^{-1} G_u \end{bmatrix}^\top \begin{bmatrix} K_{uu} & K_{ux} \\ K_{xu} & K_{xx} \end{bmatrix} \begin{bmatrix} I \\ -G_x^{-1} G_u \end{bmatrix}$$

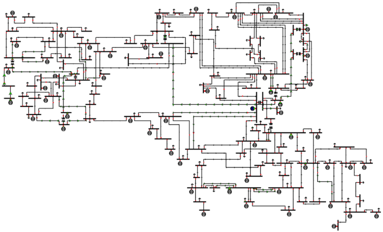
$\rightarrow$ the reduction runs efficiently in parallel on the GPU

F. Pacaud, S. Shin, M. Schanen, DA. Maldonado, M. Anitescu. "Accelerating condensed interior-point methods on SIMD/GPU architectures." JOTA (2023)

Application to the optimal power flow

The problem has a **graph structure** we can exploit:

- u: power generations
- x: voltage magnitudes and angles



Optimal power flow

Power flow balance

$$\min_{x,u} \; f(x,u) \quad \text{s.t.} \quad \begin{cases} g(x,u) = 0 \\ h(x,u) \leq 0 \end{cases}$$

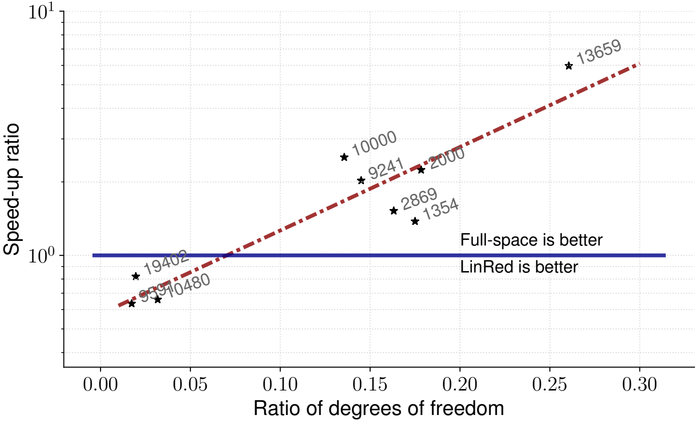
Line flow constraints

Structure is explicit!

Numerical results on large-scale OPF instances

Observations

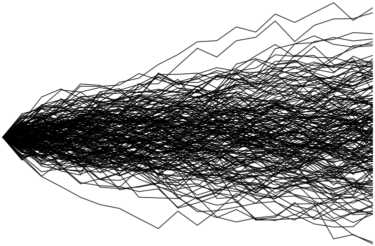
- The performance depends on the number of controls in the problem (the less, the better)
- Results on the AC OPF problem: the reduction gives better results than SOTA if ratio $< 7\%$



F. Pacaud, S. Shin, M. Schanen, DA. Maldonado, M. Anitescu. "Accelerating condensed interior-point methods on SIMD/GPU architectures." JOTA (2023)

Security-constrained optimal power flow

- N scenarios, with *one coupling* $\boldsymbol{u}$ (power generations)
- One recourse per scenario: states $x_1, \cdots, x_N$



Stochastic optimal power flow

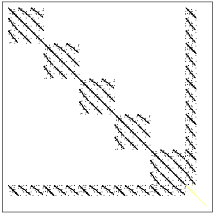
$$\min_{x_i, u} \sum_i f_i(x_i, \boldsymbol{u})$$

Power flow balance

$$\text{s.t.} \quad g_i(x_i, \boldsymbol{u}) = 0 \quad \forall i = 1, \cdots, N$$

$$h_i(x_i, \boldsymbol{u}) \le 0 \quad \forall i = 1, \cdots, N$$

Line flow constraints

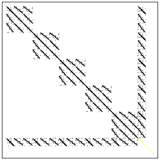


Fact

The condensed KKT system has a block-arrowhead structure

$$K = W + \nabla h^\top \Sigma_s \nabla h = \begin{bmatrix} K_{x_1 x_1} & & & K_{x_1 u} \\ & \ddots & & \vdots \\ & & K_{x_N x_N} & \\ K_{u x_1} & & \dots & K_{uu} \end{bmatrix}$$

Running a nonlinear solver on multiple GPUs with CUDA-MPI



Solution

Nested reduction using **hierarchical Schur complement** on multiple GPUs

Apply directly to the solution of two-stage nonlinear programs

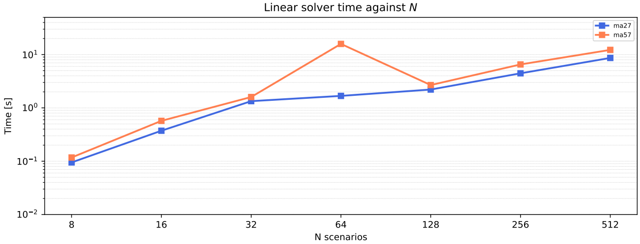
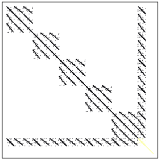


Figure: The 2000s: frontal solve using sparse LDL factorization (HSL)

F. Pacaud, M. Schanen, S. Shin, DA. Maldonado, M. Anitescu. "Parallel Interior-Point Solver for Block-Structured Nonlinear Programs on SIMD/GPU Architectur

Running a nonlinear solver on multiple GPUs with CUDA-MPI



Solution

Nested reduction using **hierarchical Schur complement** on multiple GPUs

Apply directly to the solution of two-stage nonlinear programs

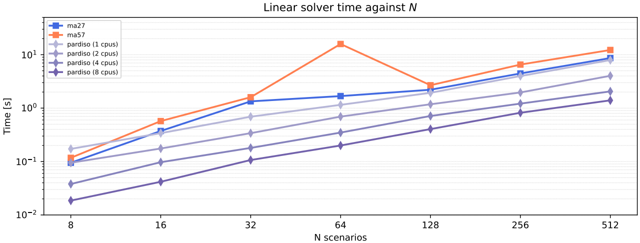
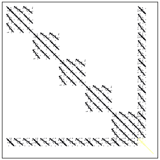


Figure: The 2010s: Schur with incomplete augmented factorization (Pardiso)

F. Pacaud, M. Schanen, S. Shin, DA. Maldonado, M. Anitescu. "Parallel Interior-Point Solver for Block-Structured Nonlinear Programs on SIMD/GPU Architectur

Running a nonlinear solver on multiple GPUs with CUDA-MPI



Solution

Nested reduction using **hierarchical Schur complement** on multiple GPUs

Apply directly to the solution of two-stage nonlinear programs

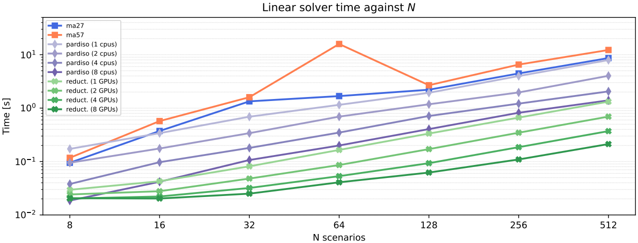


Figure: The 2020s: Schur complement with multiple RHS on GPUs

F. Pacaud, M. Schanen, S. Shin, DA. Maldonado, M. Anitescu. "Parallel Interior-Point Solver for Block-Structured Nonlinear Programs on SIMD/GPU Architectur

Solution 2: Condensation of the linear system

We look again at the condensed KKT system:

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

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

$\rightarrow$ Two strategies to reduce it down to a positive definite matrix:

1. LiftedKKT
2. HyKKT

S. Shin, F. Pacaud, and M. Anitescu. *Accelerating optimal power flow with GPUs: SIMD abstraction of nonlinear programs and condensed-space interior-point me*
S. Regev et al., "HyKKT: a hybrid direct-iterative method for solving KKT linear systems." Optimization Methods and Software 38, no. 2 (2023)

LiftedKKT

Idea: equality relaxation

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

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

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 $h^{\tau}(x) = (|g(x)| - \tau, h(x))$

Evaluating the descent direction using the 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!

HyKKT

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
We can solve the condensed KKT system using the normal equations:

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

- Once K_γ factorized with Cholesky, HyKKT solves 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)

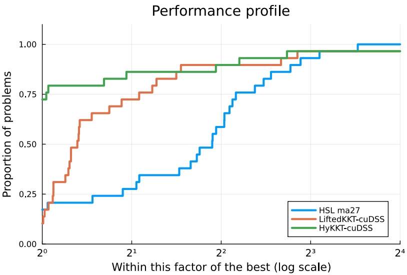
Results on the AC-OPF problem

Observations

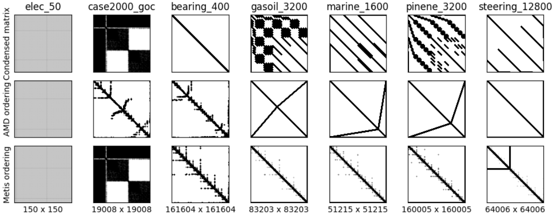
- We use the newly released `cuDSS` solver (sparse Cholesky)
- Up to 10x speed-up compared to Ipopt

	HSL MA27				LiftedKKT+cuDSS				HyKKT+cuDSS			
Case	it	init	lin	**total**	it	init	lin	**total**	it	init	lin	**total**
13659_pegase	63	0.45	7.21	**10.14**	75	0.83	1.05	**2.96**	62	0.84	0.93	**2.47**
19402_goc	69	0.63	31.71	**36.92**	73	1.42	2.28	**5.38**	69	1.44	1.93	**4.31**
20758_epigrids	51	0.63	14.27	**18.21**	53	1.34	1.05	**3.57**	51	1.35	1.55	**3.51**
78484_epigrids	102	2.57	179.29	**207.79**	101	5.94	5.62	**18.03**	104	6.29	9.01	**18.90**

Table: OPF benchmark, solved with a tolerance `tol=1e-6`. (A100 GPU)



Results on the COPS benchmark



Observation

- LiftedKKT and HyKKT remain competitive, but are not significantly faster on the COPS benchmark

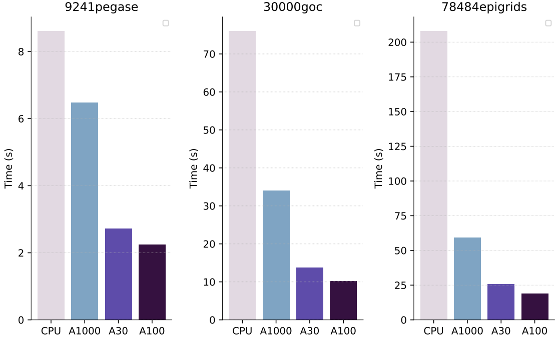
			HSL MA57				LiftedKKT+cuDSS				HyKKT+cuDSS			
	n	m	it	init	lin	**total**	it	init	lin	**total**	it	init	lin	**total**
bearing_800	643k	3k	13	0.94	14.59	**16.86**	14	3.31	0.18	**4.10**	12	3.32	1.98	**5.86**
camshape_12800	13k	38k	34	0.02	0.34	**0.54**	33	0.05	0.02	**0.16**	34	0.06	0.03	**0.19**
elec_800	2k	0.8k	354	2.36	337.41	**409.57**	298	2.11	2.58	**24.38**	184	1.81	2.40	**16.33**
gasoil_12800	333k	333k	20	1.78	11.15	**13.65**	18	2.11	0.98	**5.50**	22	2.99	1.21	**6.47**
marine_12800	410k	410k	11	0.36	3.51	**4.46**	146	2.80	25.04	**39.24**	11	2.89	0.63	**4.03**
pinene_12800	640k	640k	10	0.48	7.15	**8.45**	21	4.50	0.99	**7.44**	11	4.65	3.54	**9.25**
robot_12800	115k	77k	35	0.54	4.63	**5.91**	33	1.13	0.30	**4.29**	35	1.15	0.27	**4.58**
rocket_51200	205k	154k	31	1.21	6.24	**9.51**	37	0.83	0.17	**8.49**	30	0.87	2.67	**10.11**
steering_51200	256k	205k	27	1.40	9.74	**13.00**	15	1.82	0.19	**5.41**	28	1.88	0.56	**11.31**

Table: COPS benchmark , solved with a tolerance `tol=1e-6` (A100 GPU)

How expensive should be your GPU?

Benchmarking different GPUs

- A100 (80GB) HPC ($10,000)
- A30 (24GB) workstation ($5,000)
- A1000 (4GB) laptop



Perspective

Summary

Two practical methods to solve large-scale nonlinear programs on GPU:

- Condense & Densify
- Relax equality & condense

Take away

1. Large-scale optimization is practical on modern GPU hardware
2. On some problems, we observe a **x10** speed-up compared to state-of-the-art
3. Exciting new developments are coming!