Large-Scale Semidefinite Programming through GPU-Accelerated First-Order Optimization

Antoine Groudiev
École Normale Supérieure

Heng Yang – Supervisor
Harvard University

September 9, 2025

ENS | PSL
Harvard John A. Paulson School of Engineering and Applied Sciences

Plan

Semidefinite Programming (SDP)

Standard form of SDP:

$$\min_{X} \langle C, X\rangle \quad \text{s.t.} \quad \begin{cases} \langle A_i, X\rangle = b_i, & i \in [m] \\ X \succeq 0 \end{cases} \tag{SDP}$$

- Many **applications** (combinatorial optimization, control, physics, etc.)
- Various **algorithms** (interior-point methods, first-order methods, etc.)
- Multiple **solvers** (MOSEK, SDPT3, SeDuMi, SCS, etc.)

Trajectory Optimization

Motivation in robotics:

$$\begin{aligned} \min_{\{x_k\}_{k=0}^{N},\{u_k\}_{k=0}^{N-1},\{\lambda_k\}_{k=0}^{N-1}} \quad & \ell_N(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k, \lambda_k) \\ \text{s.t.} \quad & \begin{cases} x_0 = x_{\text{init}} \\ F_k(x_{k-1}, u_{k-1}, \lambda_{k-1}, x_k) = 0, \quad k \in [N] \\ (u_{k-1}, \lambda_{k-1}, x_k) \in \mathcal{C}_k, \quad k \in [N] \end{cases} \end{aligned} \tag{Traj-Opt}$$

Can be expressed as a **Polynomial Optimization Problem (POP)** and then relaxed as an SDP using Lasserre's hierarchy and tools such as TSSOS or SPOT [2].

Multi-block SDP

Specific **multi-block** structure of the SDP relaxation:

- Ω is the Cartesian product of the symmetric blocks
- $\Omega_+ \subset \Omega$ is the subset of Ω for which all the blocks are PSD
- Example: $\Omega = \mathbb{S}^2 \times \mathbb{S}^3$ and $\Omega_+ = \mathbb{S}^2_+ \times \mathbb{S}^3_+$

Problem (SDP) becomes:

$$\min_X \langle C, X \rangle \quad \text{s.t.} \quad \begin{cases} \langle A_i, X \rangle = b_i, & i \in [m] \\ X \in \Omega_+ \end{cases} \tag{Block-SDP}$$

Contributions

Two main contributions:

- **cuADMM**: a GPU-accelerated implementation of a first-order method (sGS-ADMM) for solving large-scale multi-block SDPs (Groudiev et al., 2025, [1])
- A new method to **project a symmetric matrix onto the PSD cone** without computing any factorization (Kang et al., 2025, [3])

Plan

Dual problem

cuADMM uses the **symmetric Gauss–Seidel ADMM (sGS-ADMM)** variant of the ADMM algorithm, applied to the dual problem

Lagrangian dual of (Block-SDP):

$$\max_{y \in \mathbb{R}^m,\, S \in \Omega} \langle b, y \rangle \quad \text{s.t.} \quad \begin{cases} A^\top y + S = C \\ S \in \Omega_+ \end{cases} \tag{Dual-SDP}$$

PSD cone is self-dual, so $\Omega_+^* = \Omega_+$

sGS-ADMM

Each iteration:

- Update each variable in sequence
- Requires solving a linear system involving $AA^\top$
- Requires projecting onto the PSD cones Ω_+

sGS-ADMM variant of ADMM:

- Symmetric Gauss-Seidel (sGS) updates (update y twice)
- Multiple blocks of variables

sGS-ADMM iteration

Step 1. Update y

$$r_s^{k+\frac{1}{2}} := \frac{1}{\sigma}b - A\left(\frac{1}{\sigma}X^k + S^k - C\right),$$
$$y^{k+\frac{1}{2}} = \left(AA^\top\right)^{-1} r_s^{k+\frac{1}{2}}.$$

Step 2. Update S

$$X_b^{k+1} := X^k + \sigma(A^\top y^{k+\frac{1}{2}} - C),$$
$$S_b^{k+1} = \frac{1}{\sigma}\left(\Pi_{\Omega_+}\left(X_b^{k+1}\right) - X_b^{k+1}\right).$$

Step 3. Update y again (sGS step)

$$r_s^{k+1} := \frac{1}{\sigma}b - A\left(\frac{1}{\sigma}X^k + S^{k+1} - C\right),$$
$$y^{k+1} = \left(AA^\top\right)^{-1} r_s^{k+1}.$$

Step 4. Update X

$$X^{k+1} = X^k + \tau\sigma\left(S^{k+1} + A^\top y^{k+1} - C\right).$$

Plan

Standard method: eigenvalue decomposition

PSD projection of a symmetric matrix $M \in \mathbb{S}^n$:

$$\Pi_{\mathbb{S}^n_+}(M) \ := \ \operatorname*{arg\,min}_{Y \in \mathbb{S}^n_+} \frac{1}{2} \, \|Y - M\|_{\mathsf{F}}^2 \, ,$$

Standard method: eigenvalue decomposition (EVD). If $M = Q\Lambda Q^\top$, then:

$$\Pi_{\mathbb{S}^n_+}(M) = Q \, \mathrm{diag}[\max\{\lambda_1, 0\}, \ldots, \max\{\lambda_n, 0\}] Q^\top .$$

Drawbacks:

- not really GPU-friendly
- cannot be computed in half-precision

Idea: apply ReLU using a polynomial

ReLU function: $f_{\mathsf{ReLU}}(x) = \max\{x, 0\}$. Then:

$$\Pi_{\mathbb{S}_+^n}(M) = Q\,\mathrm{diag}[f_{\mathsf{ReLU}}(\lambda_1), \ldots, f_{\mathsf{ReLU}}(\lambda_n)]Q^\top = f_{\mathsf{ReLU}}(M).$$

Projection amounts to applying f_{ReLU} to the eigenvalues of M!

Idea: apply ReLU using a polynomial

ReLU function: $f_{\text{ReLU}}(x) = \max\{x, 0\}$. Then:

$$\Pi_{\mathbb{S}^n_+}(M) = Q\,\mathrm{diag}[f_{\text{ReLU}}(\lambda_1), \ldots, f_{\text{ReLU}}(\lambda_n)]Q^\top = f_{\text{ReLU}}(M).$$

Projection amounts to applying f_{ReLU} to the eigenvalues of M!

Idea: approximate f_{ReLU} using a polynomial p:

$$\Pi_{\mathbb{S}^n_+}(M) \approx p(M) = Q\,\mathrm{diag}[p(\lambda_1), \ldots, p(\lambda_n)]Q^\top.$$

Benefits:

- only requires matrix multiplications and additions $\implies$ GPU-friendly
- can be computed in half-precision

Designing a good polynomial

For performance purposes, we look for p as a **composite polynomial** of depth T:

$$p(x) = f_T \circ f_{T-1} \circ \cdots \circ f_1(x),$$

Designing a good polynomial

For performance purposes, we look for p as a **composite polynomial** of depth T:

$$p(x) = f_T \circ f_{T-1} \circ \cdots \circ f_1(x),$$

Note that:

$$f_{\mathsf{ReLU}}(x) = \frac{1}{2}x(1+\operatorname{sign}(x)) \quad \text{with} \quad \operatorname{sign}(x) := \begin{cases} 1, & x > 0, \\ 0.5, & x = 0, \\ -1, & x < 0. \end{cases}$$

Since approximating sign is easier than approximating f_{ReLU}, we use a two-steps approach.

Two steps to design p

Step 1. Choose the optimal coefficients for sign:

$$\{f_t^\star\}_{t=1}^T = \operatorname*{arg\,min}_{f_1,\ldots,f_T} \max_{x\in[-1,-\varepsilon]\cup[\varepsilon,1]} \left|f_T \circ f_{T-1} \circ \cdots \circ f_1(x) - \operatorname{sign}(x)\right|$$

$$\text{subject to} \qquad f_t \in \mathbb{R}_{d_t}^{\text{odd}}[x],\ t = 1,\ldots,T,$$

which can be solved using the **Remez algorithm**.

Step 2. Refine the coefficients for f_{ReLU}, with the loss function:

$$\ell(f_T,\ldots,f_1) := \max_{x\in[-1,1]} \left|\frac{1}{2}x(1 + f_T \circ f_{T-1} \circ \cdots \circ f_1(x)) - f_{\mathsf{ReLU}}(x)\right|.$$

In practice, we use S_{float} instead of $[-1,1]$.

Plan

GPU Implementation

GPU Implementation of sGS-ADMM, Lanczos, LOBPCG, and PSD cone projection

- C++ as host language
- CUDA kernels for acceleration
- Rely on `cuSOLVER` and `cuBLAS` libraries
- Open-sourced at:

  https://github.com/ComputationalRobotics/psd_projection

  https://github.com/ComputationalRobotics/cuADMM

Implementation of cuADMM

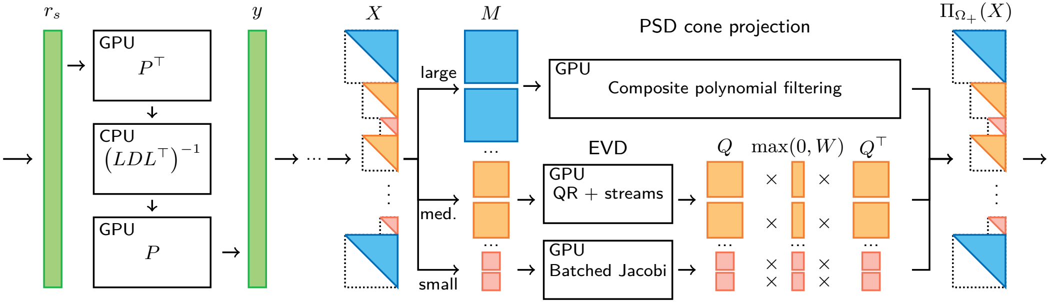


Figure 1: Illustration of the GPU implementation of ADMM.

Plan

Setup

Comparison of the performances of three solvers:

- **MOSEK** (Interior-point method, CPU)
- **ADMM+** (First-order method, CPU)
- **cuADMM** (First-order method, GPU), both sGS-ADMM and standard ADMM

Datasets:

- Trajectory optimization problems generated by SPOT [2]
- Generic multi-block SDPs collected by Mittelmann [4]

Solve up to accuracy 10^{-3} or timeout after 10 hours.

Results on SPOT problems

Problem	Solver	Time (s)	Max. KKT residual η	Primal distance
Push Box	MOSEK	**213.4**	8.47×10^{-7}	9.20×10^{-4}
	ADMM+[1]	$1,649$	2.20×10^{-3}	1.85×10^{-4}
	cuADMM	905.1	9.98×10^{-4}	2.96×10^{-4}
	cuADMM (sGS)	278.0	9.97×10^{-4}	4.97×10^{-4}
Push T	MOSEK	**36.6**	2.44×10^{-7}	8.3×10^{-4}
	ADMM+	—	—	—
	cuADMM	142.0	1.61×10^{-5}	9.03×10^{-4}
	cuADMM (sGS)	186.6	8.24×10^{-4}	8.99×10^{-4}
Planar Hand	MOSEK	$12,017$	1.91×10^{-6}	—
	ADMM+	—	—	—
	cuADMM	—	—	—
	cuADMM (sGS)	$\mathbf{1,281}$	9.95×10^{-4}	—
Tunnel	MOSEK	$32,593$	7.09×10^{-6}	—
	ADMM+	—	—	—
	cuADMM	—	—	—
	cuADMM (sGS)	$\mathbf{2,491}$	9.97×10^{-4}	—

Method

Two metrics to evaluate the quality of the projection:

- **Execution time**
- **Relative error** w.r.t. the ground truth (computed with `cuSOLVER` `FP64`):

$$\frac{\left\|A_{+} - \Pi_{\mathbb{S}^n_+}(A)\right\|_{\mathsf{F}}}{\left\|\Pi_{\mathbb{S}^n_+}(A)\right\|_{\mathsf{F}}}$$

In the context of ADMM, execution time is often more important than accuracy

Setup

Projection methods:

- `cuSOLVER FP64`: classical factorization-based method with `cuSOLVER`(ground truth)
- `cuSOLVER FP32`: same as `cuSOLVER FP64`, single precision
- `Composite FP32`: our filtering-based method in `FP32` with 31 GEMMs
- `Composite FP32` (em.): same as `Composite FP32`, with `BF16x9` emulation
- `Composite FP16`: our filtering-based method in `FP16` with 22 GEMMs

Datasets: matrices generated using the Matrix Depot package [5]

Results: execution time

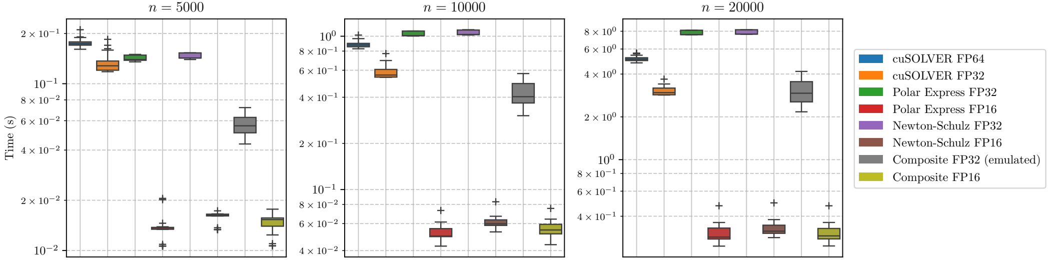


Figure 2: Boxplots for different PSD cone projection methods' *execution time* on B200 GPU.

- FP32: composite is up to $2\times$ faster than `cuSOLVER`
- FP16: composite is up to $10\times$ faster than `cuSOLVER` (single)

Results: relative error

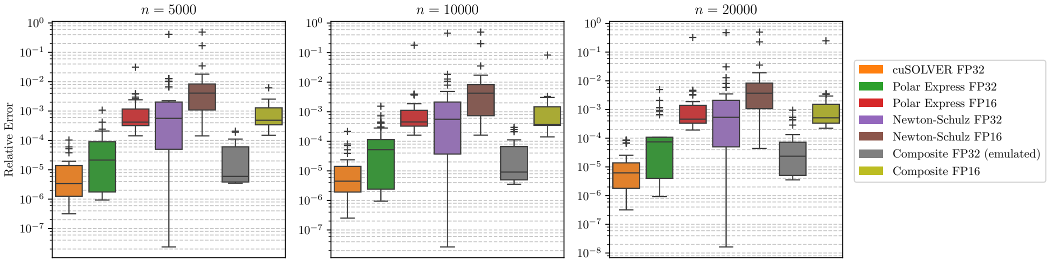


Figure 3: Boxplots for different PSD cone projection methods' *relative error* on B200 GPU.

- `FP32`: `cuSOLVER` is 2.5 to $10\times$ more accurate than composite
- `FP16`: `cuSOLVER` (single) is up to $200\times$ more accurate than composite

Warm-starting ADMM with low-precision projections

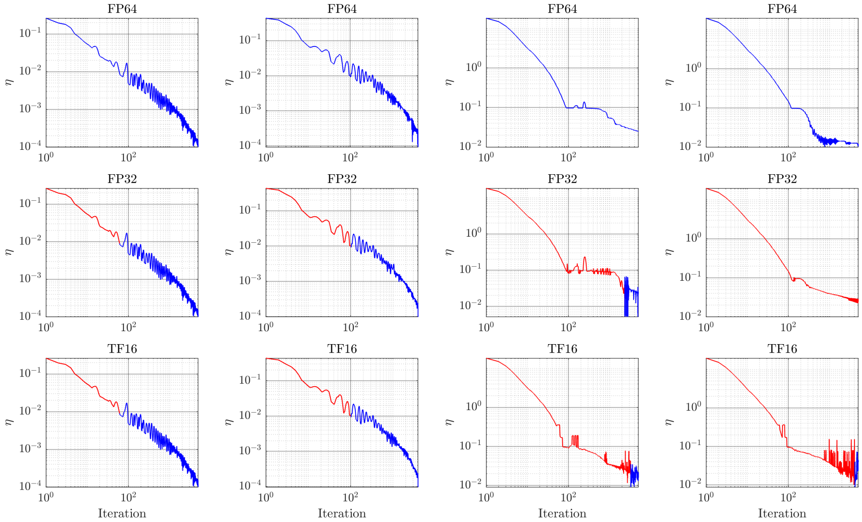
At first, speed is more important than projection accuracy; later, accuracy is more important than speed for convergence

Does using low-precision projections hurt convergence?

Two-phase approach for the experiments on Mittelmann's single-block datasets:

- First phase: use `Composite FP16` instead of `cuSOLVER FP64` for the projection
- Then switch to `cuSOLVER FP64` once $\eta \leqslant 10^{-2}$

Results: comparison of convergence



(a) `G55mc`, $n = 5000$ (b) `G59mc`, $n = 5000$ (c) `G60_mb`, $n = 7000$ (d) `G60mc`, $n = 7000$

Conclusion

cuADMM, a GPU-accelerated implementation of sGS-ADMM for solving large-scale multi-block SDPs:

- outperforms CPU-based interior-point and first-order methods on large-scale problems

Composite polynomial filtering for PSD cone projection:

- **up to** $10\times$ **faster** than factorization-based methods on large matrices
- **less accurate** than factorization-based methods
- can be used to warm-start ADMM with low-precision projections $\Longrightarrow$ **no convergence degradation**

References I

[1] Antoine Groudiev, Shucheng Kang, and Heng Yang. "cuADMM: GPU-Accelerated First-Order Optimization for Large-Scale Multi-Block Semidefinite Programs". In: *2025 RSS Workshop on Fast Motion Planning and Control in the Era of Parallelism*. 2025. URL: https://openreview.net/forum?id=SCrhEJ29HO.

[2] Shucheng Kang, Guorui Liu, and Heng Yang. "Global Contact-Rich Planning with Sparsity-Rich Semidefinite Relaxations". In: *Robotics: Science and Systems (RSS)*. 2025.

[3] Shucheng Kang et al. "Factorization-free Orthogonal Projection onto the Positive Semidefinite Cone with Composite Polynomial Filtering". In: *arXiv preprint* (2025). URL: https://arxiv.org/abs/2507.09165.

[4] Hans D Mittelmann. *Several SDP-codes on sparse and other SDP problems*. 2006.

References II

[5] Weijian Zhang and Nicholas J Higham. "Matrix Depot: an extensible test matrix collection for Julia". In: *PeerJ Computer Science* 2 (2016), e58.