

An Accelerated Hybrid Low-Rank Augmented Lagrangian Method for Large-Scale Semidefinite Programming on GPU

Renato D.C. Monteiro

School of ISyE, Georgia Tech

June 4, 2026 (3:45 PM Edinburgh time)

SIOPT 2026 - University of Edinburgh (25 min)

Goals of my talk:

- Show that a first-order method for semidefinite programming (SDP), namely **HALLaR**, can be significantly accelerated using GPU
- Compare **HALLaR** and its GPU-accelerated version **cu-HALLaR** against other state-of-the-art SDP solvers for solving large-scale SDPs

SDP Formulation currently handled by HALLaR

- HALLaR currently handles

$$\min_X \{ \langle C, X \rangle : \mathcal{A}X = b, \quad X \in \Delta^n \} \quad (\text{P})$$

where $b \in \mathbb{R}^m$, $C \in \mathbb{S}^n$, $\mathcal{A} : \mathbb{S}^n \rightarrow \mathbb{R}^m$ is a linear map, and

$$\Delta^n := \{X \in \mathbb{S}^n : \text{tr } X \leq 1, X \succeq 0\}$$

- The **dual** of (P) is

$$\max_{p \in \mathbb{R}^m, \theta \in \mathbb{R}} \{ -b^T p - \theta \quad : \quad S := C + \mathcal{A}^* p + \theta I \succeq 0, \quad \theta \geq 0 \} \quad (\text{D})$$

where $\mathcal{A}^* : \mathbb{R}^m \rightarrow \mathbb{S}^n$ is the adjoint of $\mathcal{A}$

Computational difficulties in solving huge SDPs

- Projecting (usually dense) matrices onto $\mathbb{S}^n$ or Δ^n (which many projection-type methods do) is expensive since it requires full eigen-decomposition. Instead, HALLaR computes min eigen-pairs of structured matrices (e.g., sparse ones)
- Projecting onto $\{X : \mathcal{A}X = b\}$ (which a few methods do) is costly. Being a primal-dual method, HALLaR never performs this projection
- Storing the $n \times n$ matrix variable X requires huge amount of memory
 - a laptop with 32 GB of memory can only store a fully dense X with $n \approx 65,000$
 - HALLaR addresses this limitation by using a low-rank factorization scheme proposed by Burer and M^{**} (2003) where all iterates have the form $X = UU^T$ for some $n \times r$ matrix U with $r \ll n$, and only U is stored

Outline of HALLaR

HALLaR is an inexact augmented Lagrangian (AL) method

- Let initial multiplier $p_0 \in \mathbb{R}^m$ and penalty parameter $\beta > 0$ be given
- for every $t \geq 1$, its t -th iteration consists of

$$\begin{aligned} X_t &\approx \operatorname{argmin}_{X \in \Delta^n} \left\{ \mathcal{L}_\beta(X; p_{t-1}) := C \bullet X + \langle p_{t-1}, \mathcal{A}X - b \rangle + \frac{\beta}{2} \|\mathcal{A}X - b\|^2 \right\} \quad (\text{AL}_t) \\ p_t &= p_{t-1} + \beta(\mathcal{A}X_t - b) \end{aligned}$$

- HALLaR uses a hybrid low-rank (HLR) method to solve subproblem (AL_t)
- HLR generates X_t in factored form $X_t = Y_t Y_t^T$ where $\#$ of columns of $Y_t \ll n$
- Only Y_t is computed and stored (X_t is **never** computed)

Outline of the HLR method for solving subproblem (AL_t)

- 0) Let $U_0 \in \mathbb{R}^{n \times r_1}$ be the near-global solution Y_{t-1} found for subproblem (AL_{t-1}) and set $k = 1$
- 1) Starting from $U_{k-1} \in \mathbb{R}^{n \times r_k}$, use a nonconvex composite optimization solver, called ADAP-AIPP, to find a near stationary point $\tilde{U}_k$ of

$$\min_{U \in \mathbb{R}^{n \times r_k}} \left\{ \mathcal{L}_\beta(UU^T; p_{t-1}) : \|U\|_F \leq 1 \right\} \quad (P_k)$$

- 2) If $\tilde{U}_k \tilde{U}_k^T$ is a near-solution for subproblem (AL_t) , then set $Y_t = \tilde{U}_k$ and **stop**
- 3) Perform a Frank-Wolfe step w.r.t. subproblem (AL_t) at $\tilde{U}_k$ to obtain $U_k \in \mathbb{R}^{n \times r_{k+1}}$, set $k \leftarrow k + 1$, and go to step 1

- 0) Let $Y_{t-1} \in \mathbb{R}^{n \times r_1}$ be the near-solution found for subproblem (AL_{t-1}) and set $U_0 = Y_{t-1}$ and $k = 1$
- 1) Starting from $U_{k-1} \in \mathbb{R}^{n \times r_k}$, use ADAP-AIPP to find a near stationary point $\tilde{U}_k$ of

$$\min_{U \in \mathbb{R}^{n \times r_k}} \left\{ \mathcal{L}_\beta(UU^T; p_{t-1}) : \|U\|_F \leq 1 \right\} \quad (P_k)$$

- 2) If $\tilde{U}_k \tilde{U}_k^T$ is a near-solution for subproblem (AL_t) , then set $Y_t = \tilde{U}_k$ and **stop**
- 3) Perform a FW step w.r.t. (AL_t) at $\tilde{U}_k$ to obtain $U_k \in \mathbb{R}^{n \times r_{k+1}}$, set $k \leftarrow k + 1$, and go to step 1

Remarks:

- Every iteration k of HLR computes a near-stationary solution $\tilde{U}_k$ of (P_k) (using U_{k-1} as a warm start) and performs a FW step to escape from $\tilde{U}_k$ in case it is spurious
- All inner iterates generated by ADAP-AIPP, and hence $\tilde{U}_k$, have r_k columns
- U_k generally has $r_k + 1$ columns, i.e., $r_{k+1} = r_k + 1$ (but $r_{k+1} = 1$ is possible but very unlikely)
- In practice, HLR usually perform only one iteration to solve most of the AL subproblems, in which case, its output has the same number of columns as its input

Step 2 - Check near optimality of $\tilde{U}_k \tilde{U}_k^T$ for subproblem (AL_t)

- Compute a min eigenpair $(\lambda_k^{\min}, v_k^{\min})$ of $\tilde{G}_k := \nabla_X \mathcal{L}_\beta(\tilde{U}_k \tilde{U}_k^T; p_{t-1})$ and set

$$\theta_k = \max \{ -\lambda_k^{\min}, 0 \}, \quad u_k = \begin{cases} v_k^{\min} & \text{if } \theta_k > 0, \\ 0 & \text{otherwise} \end{cases}$$

- If $(\tilde{G}_k \tilde{U}_k) \bullet \tilde{U}_k + \theta_k \leq \epsilon$, then declare $\tilde{U}_k$ to be a near-solution of (AL_t)

Remark:

- The matrix $u_k u_k^T$ solves the linear minimization oracle

$$u_k u_k^T = \operatorname{argmin}_{X \in \Delta^n} \langle \tilde{G}_k, X \rangle$$

Step 3 - Performing FW step at $\tilde{U}_k \tilde{U}_k^T$ for $(\mathbf{A}\mathbf{L}_t)$

- Given $\tilde{U}_k$ and (u_k, θ_k) obtained in steps 1 and 2, respectively, compute

$$\alpha_k = \operatorname{argmin}_{\alpha} \left\{ \mathcal{L}_{\beta} \left(\alpha u_k (u_k)^T + (1 - \alpha) \tilde{U}_k \tilde{U}_k^T; p_{t-1} \right) : \alpha \in [0, 1] \right\}$$

and set

$$U_k = \begin{cases} u_k & \text{if } \alpha_k = 1 \\ \left[\sqrt{1 - \alpha_k} \tilde{U}_k, \sqrt{\alpha_k} u_k \right] & \text{otherwise;} \end{cases}$$

- α_k has the closed form expression

$$\alpha_k := \min \left\{ \frac{[C + \mathcal{A}^* q_k] \tilde{U}_k \bullet \tilde{U}_k + \theta_k}{\beta \|\mathcal{A}(\tilde{U}_k \tilde{U}_k^T) - \mathcal{A}(u_k u_k^T)\|^2}, 1 \right\}$$

where

$$q_k := p_{t-1} + \beta \left[\mathcal{A}(\tilde{U}_k \tilde{U}_k^T) - b \right]$$

Complexity Result for HALLaR

Theorem

Given a tolerance $\epsilon > 0$, HALLaR performs at most

$$\mathcal{O}\left(\frac{1}{\epsilon^2}\right)$$

eigen-pair computations (and total FISTA calls) to find an ϵ -solution of our primal/dual SDP pair, i.e., a triple $(\bar{X}, \bar{p}, \bar{\theta})$ satisfying

$$(\epsilon\text{-primal feasibility}) \quad \|\mathcal{A}(\bar{X}) - b\| \leq \epsilon, \quad \text{tr}(\bar{X}) \leq 1, \quad \bar{X} \succeq 0$$

$$(\text{dual feasibility}) \quad \bar{S} := C + \mathcal{A}^* \bar{p} + \bar{\theta} I \succeq 0, \quad \bar{\theta} \geq 0$$

$$(\epsilon\text{-complementarity}) \quad \langle \bar{X}, \bar{S} \rangle + \bar{\theta}(1 - \text{tr} \bar{X}) \leq \epsilon$$

GPU Acceleration: cuHALLaR

- **cuHALLaR** can achieve up to **135x** speedup compared to HALLaR (joint work with J. Aguirre, D. Cifuentes, V. Guigues, M***, V.H. Nascimento and A. Sujanani)
- Its main computational bottlenecks consist of sparse matrix-matrix and matrix-vector multiplications, which can be suitably parallelized in GPU
- The gradient of $\mathcal{L}_\beta(\cdot; p_{t-1})$ at $X = UU^T$ is given by

$$\nabla_X \mathcal{L}_\beta(UU^T; p_{t-1}) = C + \mathcal{A}^* p, \quad p = p_{t-1} + \beta [\mathcal{A}(UU^T) - b]$$

Its minimum eigenvalue can be computed using Lanczos method by evaluating

$$(C + \mathcal{A}^* p)v, \quad \text{where } v \in \mathbb{R}^{n \times 1}$$

- ADAPT-AIPP requires the computation of the gradient of $U \mapsto \mathcal{L}_\beta(UU^T; p_{t-1})$

$$(C + \mathcal{A}^* p)U \in \mathbb{R}^{n \times r}$$

Solvers Included for Comparison

We compare **HALLaR** and **cuHALLaR** against the following SDP solvers:

- CSDP: Julia solver based on an interior point method (Borchers 1999)
- COSMO: Julia solver based on an ADMM (Garstka et al. 2019)
- SDPNAL+: MATLAB solver based on an augmented Lagrangian/ semismooth Newton method (Sun et al. 2019)
- T-CGAL: thin variant of FW-AL method that generates the iterates X_t in factored form (Yurtsever et al. 2018)
- r -Sketchy ($r - Sk$) : low-rank variant of T-CGAL method that only stores a sketch of $\pi(X_t) \in \mathbb{R}^{n \times r}$ of the iterates $X_t \in \mathbb{S}^n$ (Yurtsever et al. 2019)
- cuLoRADs: augmented Lagrangian/ADMM method that solves the block ADMM subproblems using conjugate gradient (Han et al. 2024)

Computational Results for **Max Stable Set**

- Given a graph $G = ([n], \mathcal{E})$, the maximum stable set problem consists of finding largest set of vertices such that no two of them are connected by an edge
- Can be formulated as the IP:

$$\alpha(G) = \max_x \left\{ \sum_{i=1}^n x_i : x_i + x_j \leq 1 \ \forall (i,j) \in \mathcal{E}, \ x_i \in \{0,1\} \ \forall i = 1, \dots, n \right\}$$

- Letting $e = (1, \dots, 1)^T$ and solving the SDP

$$\max_{X \in \mathbb{S}^n} \left\{ ee^T \bullet X : X_{ij} = 0 \ \forall (i,j) \in \mathcal{E}, \ \text{tr } X = 1, \ X \succeq 0 \right\}$$

yields an upper bound on $\alpha(G)$

Maximum Stable Set Results

Graph		Runtime (seconds)						
$ \mathcal{V} $	$ \mathcal{E} $	HALLaR	10-Sk	100-Sk	T-CGAL	CSDP	COSMO	SDPNAL+
9k	18k	5.77s	* /.74e-02	* /.74e-02	* /.82e-01	2076.17s	* /.77e-03	* /.10e+01
10k	20k	5.87s	* /.70e-02	* /.71e-02	* /.13e-01	7599.80s	**	* /.47e+01
14k	28k	8.08s	* /.60e-02	* /.60e-02	* /.24e00	**	**	* /.99e00
20k	40k	10.89s	* /.91e-01	* /.71e-01	* /.10e00	**	**	* /.10e+01
69k	138k	40.64s	* /.63e00	* /.29e00	* /.38e00	**	**	**

Table: Runtimes (in seconds) and final accuracies for the Maximum Stable Set problem. Time limit of 3 hours is given. An entry marked with */ (resp., **) means corresponding method found solution (resp., crashed) with accuracy strictly larger than 10^{-5} within 3 hours.

HALLaR vs cuHALLaR: CPU vs GPU comparison

Graph		Runtime		Ratio
$ \mathcal{V} $	$ \mathcal{E} $	HALLaR	cuHALLaR	HALLaR/cuHALLaR
1mil	10.5mil	249.70s	6.62s	37.72
2.1mil	22.0mil	592.70s	12.96s	45.73
4.2mil	46.1mil	1702.26s	18.84s	90.35
8.4mil	96.5mil	4205.74s	51.75s	81.27
524k	1.0mil	6437.05s	171.81s	37.47
1.0mil	2.1mil	16176.30s	119.50s	135.37

Table: Runtimes (in seconds) for Maximum Stable Set problem on Hamming graphs (top) and network graphs (bottom). Tolerance of 10^{-5} is set.

Matrix Completion

- Consider the problem of retrieving a low rank matrix $M \in \mathbb{R}^{n_1 \times n_2}$, where $n_1 \leq n_2$, by observing a subset of its entries: M_{ij} , $ij \in \Omega$. One approach solves the nuclear norm relaxation:

$$\min_{Y \in \mathbb{R}^{n_1 \times n_2}} \{ \|Y\|_* : Y_{ij} = M_{ij} \quad \forall (i,j) \in \Omega \}$$

- The above problem can be rephrased as the following SDP:

$$\min_{X \in \mathbb{S}^{n_1+n_2}} \left\{ \frac{1}{2} \text{tr}(X) : X = \begin{pmatrix} W_1 & Y \\ Y^T & W_2 \end{pmatrix} \succeq 0, Y_{ij} = M_{ij} \quad \forall (i,j) \in \Omega \right\}$$

cuLoRADS vs cuHALLaR

Problem Size		Runtime		Ratio
n	m	cuLoRADS	cuHALLaR	cuLoRADS/cuHALLaR
400k	46.4milă	40.65ă	12.24	3.32
600k	71.8milă	67.11	20.76	3.23
1.2mil	151.2mil	159.7	46.21	3.46
1.4mil	178mil	192.75	95.53	2.02
1.6mil	206mil	319.17	100.77	3.17
1.8mil	233mil	258.16	119.29	2.16
2.0mil	261mil	293.65	114.11	2.57

Table: Runtimes (in seconds) for the Matrix Completion Problem. Tolerance of 10^{-5} is set.

HALLaR vs cuHALLaR

Problem Size		Runtime		Ratio
<i>n</i>	<i>m</i>	HALLaR	cuHALLaR	HALLaR/cuHALLaR
400k	46.4mil	1419.26s	12.24s	115.95
600k	71.8mil	1949.80s	20.76s	93.92
1.2mil	151.2mil	4403.04s	46.21s	95.28
1.4mil	178.3mil	6400.22s	95.53s	67.00
1.6mil	205.7mil	7374.16s	100.77s	73.18
1.8mil	233.3mil	8003.14s	119.29s	67.09
2.0mil	261.2mil	8771.27s	114.11s	76.87

Table: Runtimes (in seconds) for the Matrix Completion Problem using cuHALLAR SDPA results. Tolerance of 10^{-5} is set.

Phase Retrieval

- Given m pairs $\{(a_i, b_i)\}_{i=1}^m \subseteq \mathbb{C}^n \times \mathbb{R}_+$, consider the problem of finding a vector $x \in \mathbb{C}^n$ such that

$$|\langle a_i, x \rangle|^2 = b_i, \quad i = 1, \dots, m$$

- Goal is to retrieve x from the magnitude of m linear measurements
- By creating the complex Hermitian matrix $X = xx^H$, this problem can be approached by solving the complex-valued SDP relaxation

$$\min_x \quad \left\{ \text{tr}(X) \quad : \quad \langle a_i a_i^H, X \rangle = b_i, \quad X \succeq 0, \quad X \in \mathbb{S}^n(\mathbb{C}) \right\}$$

Phase Retrieval Results

Problem Size		Runtime		Ratio
n	m	HALLaR	cuHALLaR	HALLaR/cuHALLaR
10k	120k	23.64	4.35	5.44
100k	1.2mil	303.99	10.40	29.24
100k	1.2mil	330.00	8.64	38.18
100k	1.2mil	380.05	9.33	40.75
100k	1.2mil	434.10	9.28	46.78
3.2mil	38mil	*/4.1E-03	583.78	>74

Table: Runtimes (in seconds) for Phase Retrieval problem. An entry marked with * means that the method did not find a solution with 10^{-5} accuracy within 12 hours.

Frame Title

- **cu-HALLaR** handles SDPs with bounded trace constraint
- We are currently working on variants that remove not only the trace constraint but also handle inequality constraints and SDPs with block-structure

Computational Results: Matrix Completion & Stable Set

Instance		Runtime (seconds)		Ratio
Problem	Size	cuHALLaR	cuHALLaR+	
<i>Matrix completion: rank r; size $(n_1, n_2; m)$</i>				
$r = 3$	3,000, 7,000; 828,931	0.77	0.41	1.88
$r = 5$	3,000, 7,000; 2,302,586	1.28	0.83	1.54
$r = 3$	10,000, 21,623; 2,948,996	1.22	0.58	2.10
$r = 5$	10,000, 21,623; 8,191,654	2.05	1.14	1.80
$r = 2$	25,000, 50,000; 3,367,574	1.20	0.49	2.45
$r = 3$	25,000, 50,000; 7,577,040	1.82	0.97	1.88
<i>Maximum stable set: size $(n; m)$</i>				
G11	800; 1,600	4.23	2.81	1.51
G43	1,000; 9,990	35.84	14.26	2.51
G55	5,000; 12,498	81.11	38.45	2.11
Oregon-2	11,806; 32,730	467.75	184.30	2.54
delaunay-n14	16,384; 49,122	102.43	53.17	1.93
$H_{14,2}$	16,384; 114,688	3.17	1.92	1.65

Table: Runtimes for the baseline cuHALLaR and the new version cuHALLaR+

Conclusion

- We have developed CPU and GPU solvers, HALLaR and cuHALLaR, for solving huge SDPs
- HALLaR (resp., cuHALLaR) can often solve SDPs with dimension pair on the order of millions in under a few hours (resp., few minutes)

References

This talk is based on the papers:

- R.D.C. M***, A. Sujanani, D. Cifuentes. A low-rank augmented Lagrangian method for large-scale semidefinite programming based on a hybrid convex-nonconvex approach. *Revision in Mathematical Programming* (2024).
- J. Aguirre, D. Cifuentes, V. Guigues, R.D.C. M***, V.H. Nascimento, A. Sujanani. cuHALLaR: A GPU Accelerated Low-Rank Augmented Lagrangian Method for Large-Scale Semidefinite Programming. *arXiv preprint arXiv:2505.13719* (2025).
- cu-HALLaR code can be downloaded from <https://github.com/OPThALLaR>

Thank you all!

Low-Rank Idea: Burer-M*** Method (2003)

- Burer-M*** proposed a low-rank method in 2003 to circumvent all 3 of these issues
 - Especially the issue of storing X .
- Idea: restrict SDP to matrices of rank r . That is, they consider:

$$\min_U \left\{ C \bullet UU^T \quad : \quad \mathcal{A}(UU^T) = b, \quad \|U\|_F \leq 1, \quad U \in \mathbb{R}^{n \times r} \right\} \quad (P_r)$$

- Advantage: matrix variable U has $n \times r$ entries. X in (P) has $n(n+1)/2$
- Due to results from Pataki and Shapiro, if r is chosen large enough then if U^* is optimal for (P_r) then $U_* U_*^T$ is optimal for original problem (P).
 - Problem: (P_r) is nonconvex. Finding optimum is NP-hard.
- Nonconvexity is benign
 - If r large enough, then almost always, local minima of (P_r) are global minima of (P_r) .
- Still, limited study of complexity/convergence rate guarantees

Step 1: Solving the Nonconvex Subproblem (P_k)

- Recall that, using U_{k-1} as initial iterate, the goal of ADAP-AIPP is to find a near-stationary point $\tilde{U}_k$ of

$$\min_{U \in \mathbb{R}^{n \times r_k}} \{ \mathcal{L}_\beta(UU^T; p_{t-1}) : \|U\|_F \leq 1 \} \quad (P_k)$$

- ADAP-AIPP is an adaptive accelerated inexact proximal point method which is completely parameter-free; it is based on ideas from Carmon et al. '18, Paquette et al. '18, Kong et al. '19, and Sujanani & M** '23)
- More specifically, started from $V_0 = U_{k-1}$, ADAP-AIPP iterates as

$$V_i \approx \operatorname{argmin} \left\{ \mathcal{L}_\beta(UU^T; p_{t-1}) + \frac{1}{2\lambda_i} \|U - V_{i-1}\|^2 : \|U\|_F \leq 1 \right\}$$

until a near-stationary point $V_i =: \tilde{U}_k$ is obtained

- ADAP-AIPP uses an adaptive version of FISTA to solve its prox subproblems