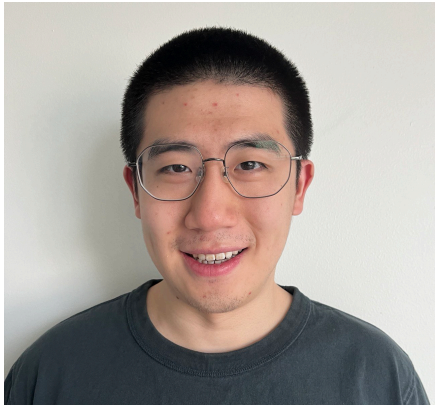
# GPU-Accelerated First-Order Methods for Linear Programming and Beyond

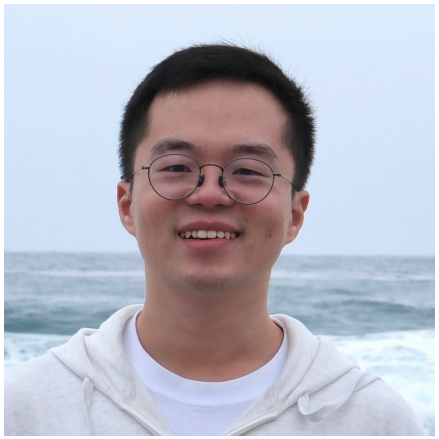
Haihao Lu

MIT

Stanford ISL, October 2026



Jinwen Yang



Zedong Peng

Acknowledgment of my early collaboration with Google

Mostly based on a series of papers:

- H Lu, J Yang (2025a). "cuPDLP.jl: A GPU Implementation of Restarted Primal-Dual Hybrid Gradient for Linear Programming in Julia." *Operations Research*.
- H Lu, Z Peng, J Yang (2025). "cuPDLPx: A Further Enhanced GPU-Based First-Order Solver for Linear Programming."
- H Lu, J Yang (2024). "Restarted Halpern PDHG for Linear Programming."
- H Lu, J Yang (2025b). "On the Geometry and Refined Rate of Primal-Dual Hybrid Gradient for Linear Programming." *Mathematical Programming*.
- H Lu, J Yang (2025c). "A Practical and Optimal First-Order Method for Large-Scale Convex Quadratic Programming." *Mathematical Programming*.
- L Ding, H Lu, J Yang (2025). "New Understandings and Computation on Augmented Lagrangian Methods for Low-Rank Semidefinite Programming."

# First, a brief history

# The starting point in 2017/2018

**My Google internship, in conversations with Miles Lubin**

**The prevailing view: LP was a "solved problem"**

- Mature simplex and interior-point solvers handled most LPs well.
- Many saw LP as a problem of the previous generation, with little room for breakthroughs.

# The starting point in 2017/2018

**My Google internship, in conversations with Miles Lubin**

**The prevailing view: LP was a "solved problem"**

- Mature simplex and interior-point solvers handled most LPs well.
- Many saw LP as a problem of the previous generation, with little room for breakthroughs.

**Google-scale LPs still relied on heuristic decomposition**

**Memory**
A 1 GB model could require 100 GB of solver memory due to factorization.

**Parallelism**
Solvers ran on a single CPU machine, with limited ability to use distributed computing.

# The starting point in 2017/2018

**My Google internship, in conversations with Miles Lubin**

**The prevailing view: LP was a "solved problem"**

- Mature simplex and interior-point solvers handled most LPs well.
- Many saw LP as a problem of the previous generation, with little room for breakthroughs.

**Google-scale LPs still relied on heuristic decomposition**

**Memory**
A 1 GB model could require 100 GB of solver memory due to factorization.

**Parallelism**
Solvers ran on a single CPU machine, with limited ability to use distributed computing.

**Could a different approach scale linear programming further?**

# A question

Can first-order methods (FOMs) and GPUs solve traditional mathematical optimization problems at scale?

# A question

Can first-order methods (FOMs) and GPUs solve traditional mathematical optimization problems at scale?

**Why the answer was unclear**

- FOMs were viewed as too slow or inaccurate.
- GPUs are not suitable for solving sparse linear systems, which is the bottleneck for the classic simplex and interior-point methods.

# A question

Can first-order methods (FOMs) and GPUs solve traditional mathematical optimization problems at scale?

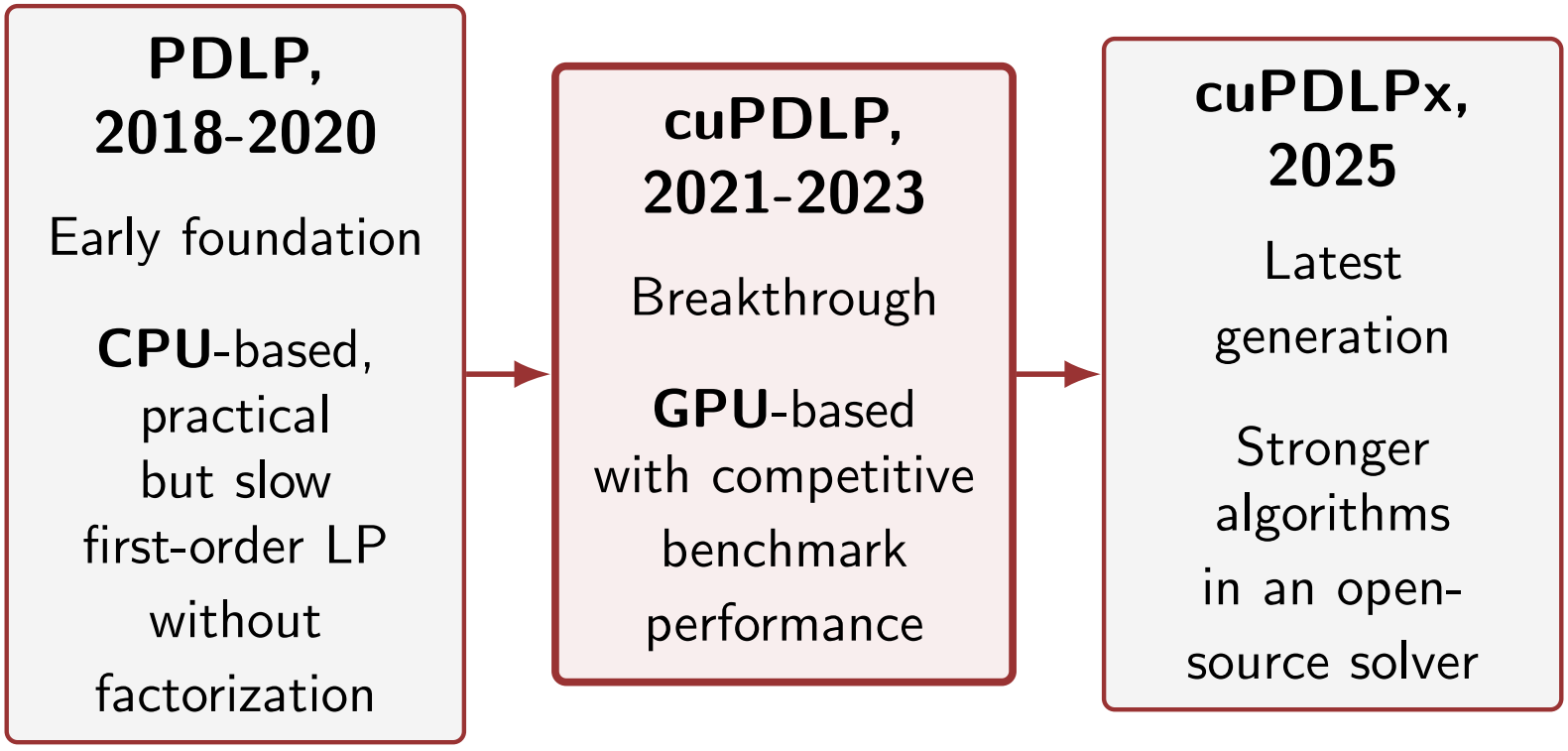
**Why the answer was unclear**

- FOMs were viewed as too slow or inaccurate.
- GPUs are not suitable for solving sparse linear systems, which is the bottleneck for the classic simplex and interior-point methods.

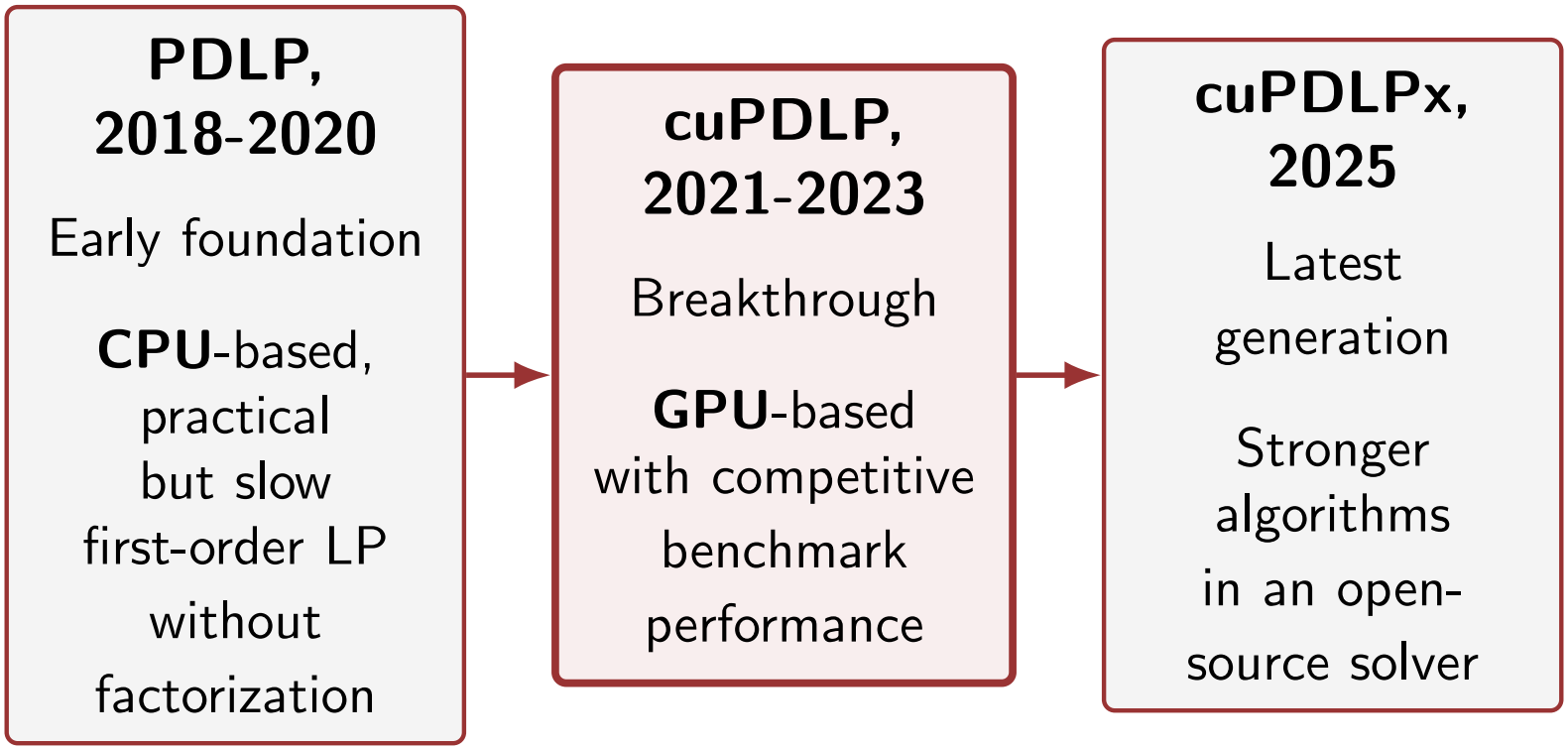
**What had to be demonstrated**

- Develop new FOMs for these problems that take advantages of GPUs.
- High accuracy and robustness, not only cheap iterations.
- Competitive end-to-end performance against established solvers.

# The development of PDLP



# The development of PDLP




**Back to the previous question**

I believe PDLP-line of research now provides strong evidence that the answer can be yes

PDLP established possibility; cuPDLP established competitiveness; cuPDLPx demonstrates continued development.

# Impact to optimization practice

## Gurobi's position in 2023

Gurobi reported that the available evidence suggested GPUs were not well suited to LP/MIP/QP solvers.

# Impact to optimization practice

## Gurobi's position in 2023

Gurobi reported that the available evidence suggested GPUs were not well suited to LP/MIP/QP solvers.

## Gurobi's subsequent developments

- **2025:** PDHG alpha and beta releases with optional GPU acceleration.
- **2026:** The PDHG implementation in Gurobi 13 closely follows the approach used in cuPDLPx

# Impact to optimization practice

## Gurobi's position in 2023

Gurobi reported that the available evidence suggested GPUs were not well suited to LP/MIP/QP solvers.

## Gurobi's subsequent developments

- **2025:** PDHG alpha and beta releases with optional GPU acceleration.
- **2026:** The PDHG implementation in Gurobi 13 closely follows the approach used in cuPDLPx

## "What changed since 2023?" (from Gurobi's blog)

"The ability to leverage libraries for sparse linear algebra on GPU and the implementation of the barrier method and **first-order LP algorithms (i.e., PDHG) on GPU**."

# Impact to optimization practice

## Gurobi's position in 2023

Gurobi reported that the available evidence suggested GPUs were not well suited to LP/MIP/QP solvers.

## Gurobi's subsequent developments

- **2025:** PDHG alpha and beta releases with optional GPU acceleration.
- **2026:** The PDHG implementation in Gurobi 13 closely follows the approach used in cuPDLPx

## "What changed since 2023?" (from Gurobi's blog)

"The ability to leverage libraries for sparse linear algebra on GPU and the implementation of the barrier method and **first-order LP algorithms (i.e., PDHG) on GPU**."

PDLP is now used by major optimization solvers and tech companies, including NVIDIA, Google, Gurobi, HiGHS, Knitro, COPT, and FICO Xpress.

# Impact to academic research

The performance gap once made competing with established commercial solvers previously is out of reach for academic general-purpose solvers.

**Our PDLP work demonstrated competitiveness on large-scale LPs**, validating the potential of **GPUs and first-order methods** and encouraging further academic solver research.

# Impact to academic research

The performance gap once made competing with established commercial solvers previously is out of reach for academic general-purpose solvers.

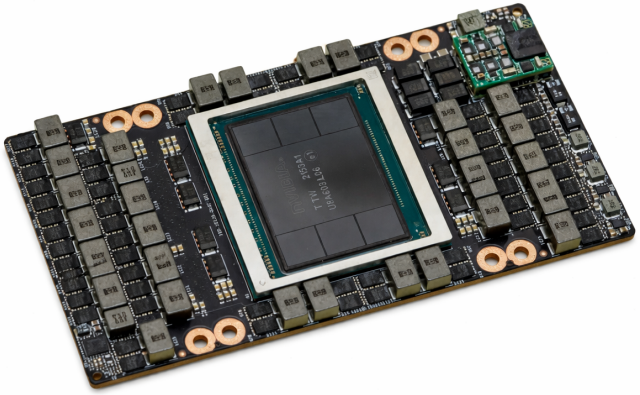
**Our PDLP work demonstrated competitiveness on large-scale LPs**, validating the potential of **GPUs and first-order methods** and encouraging further academic solver research.

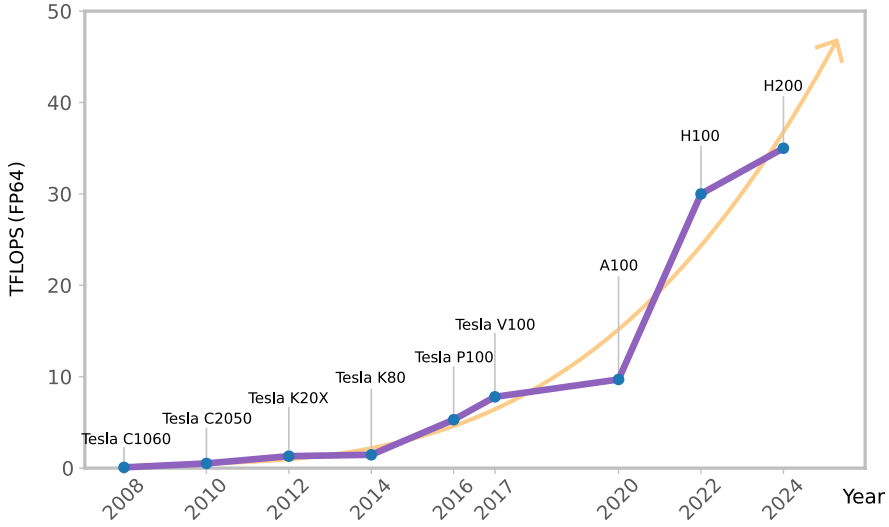
| **Selected groups** | **GPU solvers and methods** |
|---|---|
| **Stanford**<br>Yinyu Ye, et al. | cuPDLP-C (LP), D-PDLP (LP) PDHCG (QP), cuPDCS (SOCP/QP): PDHG-based; cuLoRADS (SDP): low-rank ADMM. |
| **ZIB, Berlin**<br>Koch Kempke, et al. | PDLP-based MIP heuristics use approximate LP relaxations; CHAP integrates cuPDLPx. |
| **Georgia Tech**<br>Renato Monteiro et al. | cuHALLaR (SDP): GPU implementation of a hybrid low-rank augmented Lagrangian method. |
| **PolyU**<br>Defeng Sun et al. | HPR-LP (LP): GPU solver using Halpern Peaceman–Rachford splitting. |

# Introduction

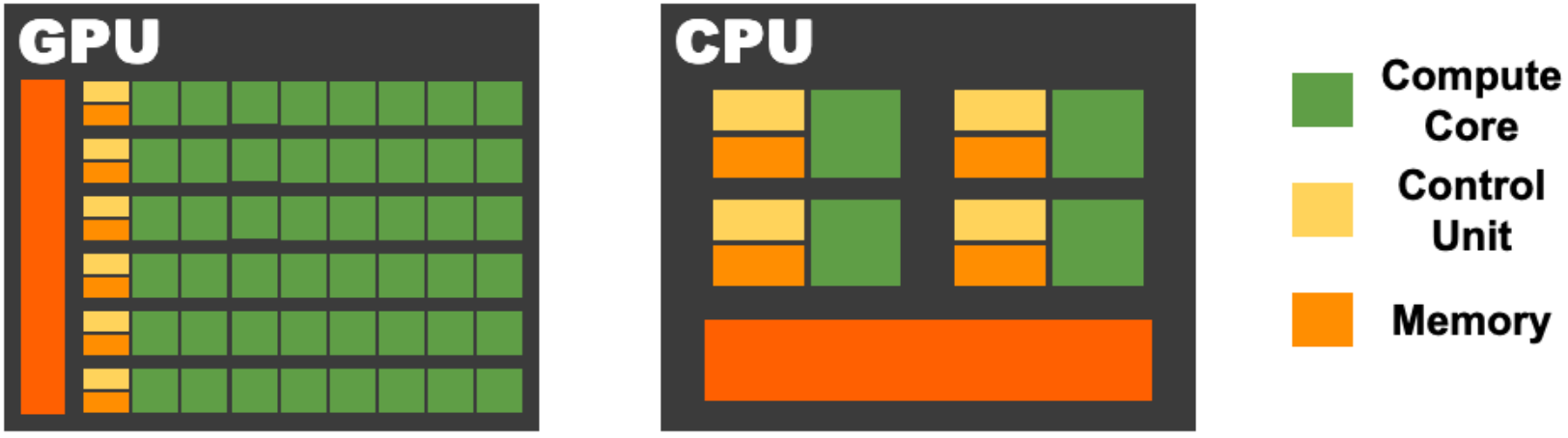
# What is a GPU?

- Graphics Processing Units (GPUs) were originally designed to accelerate computer graphics and image processing
- Over time, GPUs have found applications far beyond graphics, such as large-scale AI development
- GPU hardware has undergone rapid and sustained development





# Comparison of GPU and CPU Architectures




- GPUs consist of tens of thousands of simpler cores, while CPUs typically have a small number of powerful cores (e.g., 16-128)
- GPUs excel at large-scale parallel workloads
- GPUs provide less memory capacity but higher memory bandwidth

# Which Linear Algebra Operations are GPU Good at?

Sparse system solves via Cholesky factorization

- Computational bottleneck in **interior-point methods**
- Parallelism on GPUs is highly challenging and non-trivial to exploit
- Triangular factors can become significantly denser

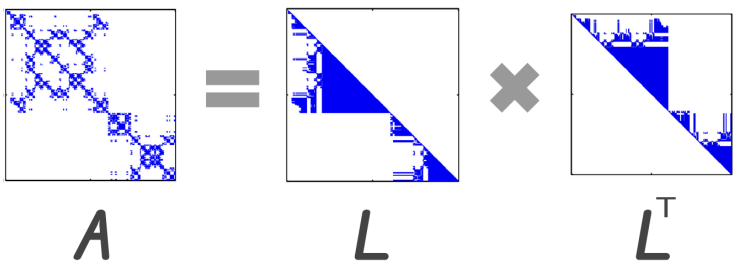



Figure: Sparse Cholesky factorization

# Which Linear Algebra Operations are GPU Good at?

Sparse system solves via Cholesky factorization

- Computational bottleneck in **interior-point methods**
- Parallelism on GPUs is highly challenging and non-trivial to exploit
- Triangular factors can become significantly denser

Sparse matrix-vector multiplication (SpMV)

- Dominant computational kernel in **first-order methods**
- Map naturally to GPU architectures

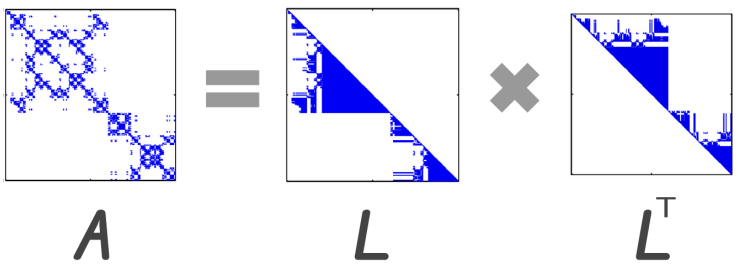



Figure: Sparse Cholesky factorization

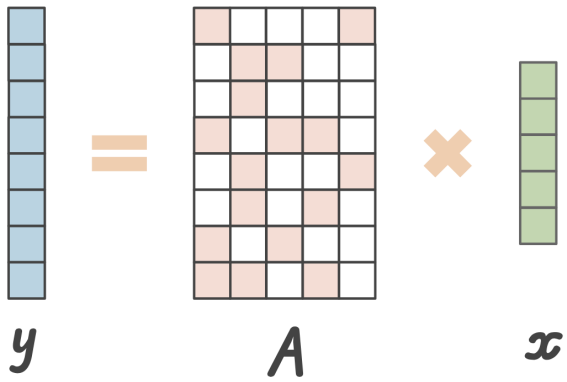
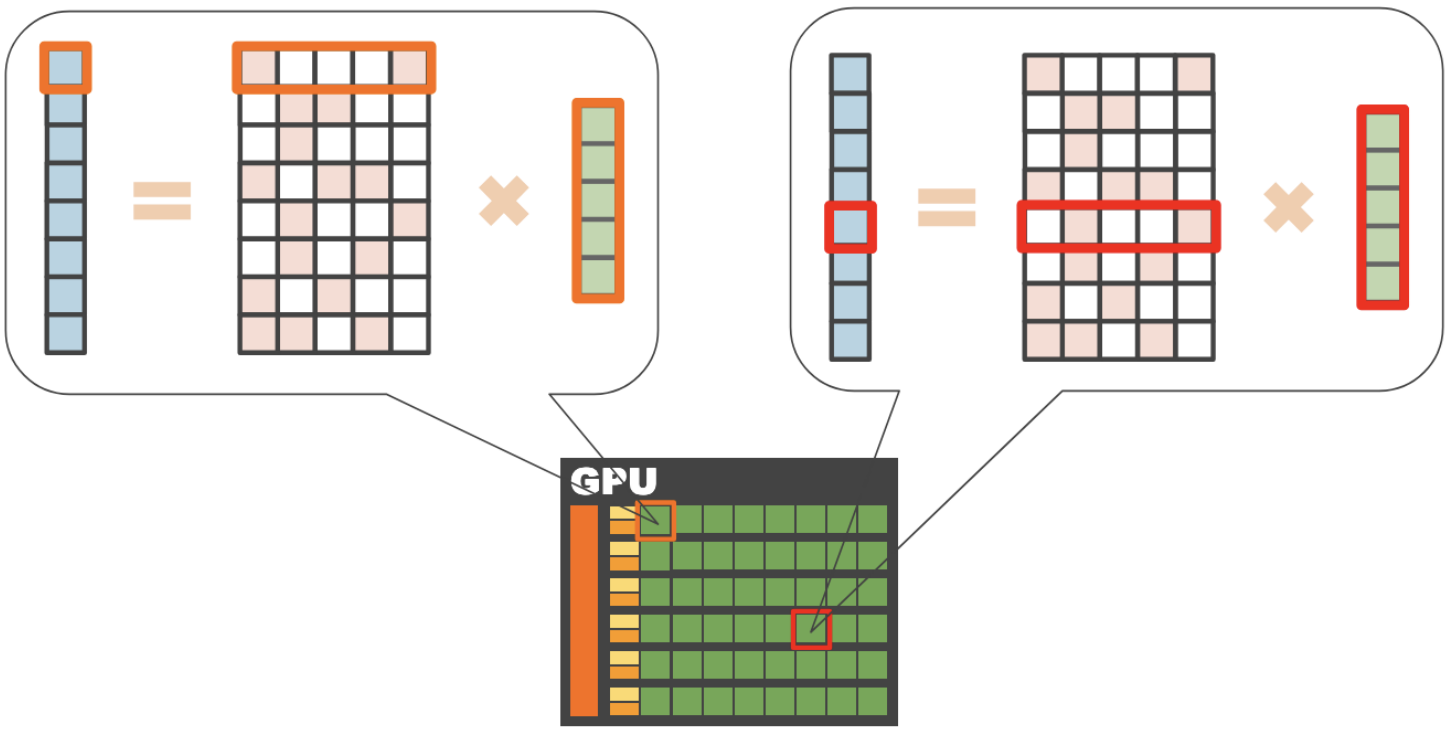



Figure: SpMV

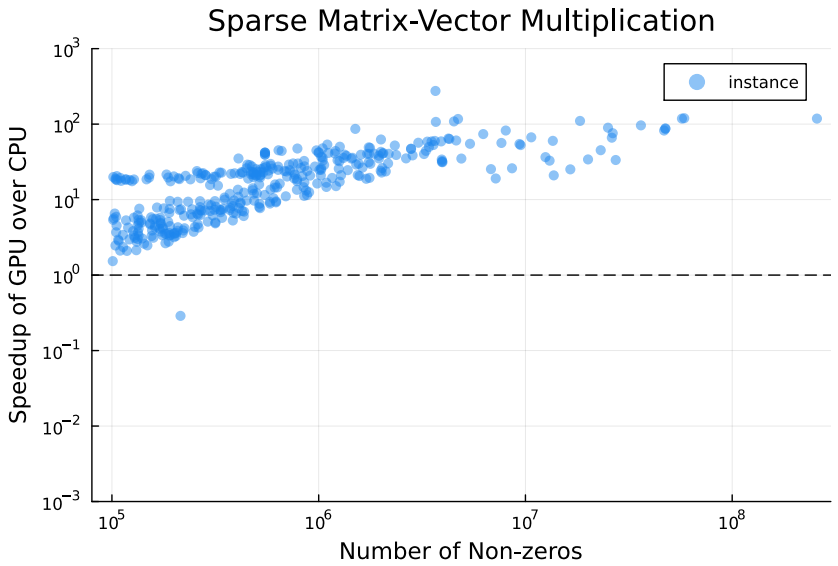
# Why is SpMV Well-Suited for GPUs?

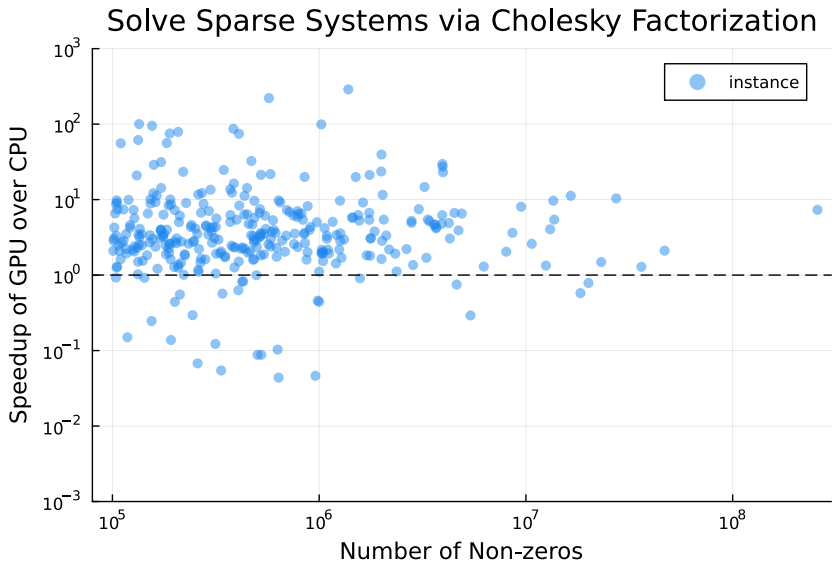
Sparse matrix-vector multiplication (SpMV)

- **Massively parallel:** Each row computes an independent dot product over nonzeros
- **Memory efficient:** No additional matrix storage is required



# Performance of SpMV and Cholesky Factorization




- GPU speedups for SpMV are consistently large
- GPU speedups for SpMV scale favorably with problem size
- GPU speedups for Cholesky factorization depend heavily on matrix structure and sparsity patterns

# Memory Usage of SpMV and Cholesky Factorization

| | | **Instances**: 4.915 MB |
|---|---|---|
| | **SpMV** | **Cholesky Factorization** |
| **Allocated Memory** | 0.971 MB | 481.067 MB |

- SpMV incurs negligible memory overhead
- Sparse Cholesky factorization requires substantially more memory
- GPU memory is more limited than CPU memory

# Memory Usage of SpMV and Cholesky Factorization

| | | **Instances**: 4.915 MB |
|---|---|---|
| | **SpMV** | **Cholesky Factorization** |
| **Allocated Memory** | 0.971 MB | 481.067 MB |

- SpMV incurs negligible memory overhead
- Sparse Cholesky factorization requires substantially more memory
- GPU memory is more limited than CPU memory

**GPUs provide greater advantages for SpMV than for sparse factorization, both in terms of performance and memory efficiency**

# First-Order Methods (FOMs)

**First-Order Methods (FOMs)**: Optimization algorithms that rely only on gradient information

**Examples**: Gradient descent (GD) and related variants

Many FOMs require only **SpMV** and avoid matrix factorization

- GPU speedups for SpMV are large and consistent
- Most runtime of FOMs is spent on SpMV operations
- Memory overhead is very low

# Outline

GPU-based first-order methods (FOMs) for linear programming (LP)

- **Algorithm**: choice of base algorithm
- **Computation**: empirical performance of cuPDLPx on GPUs

  [**L**-Yang, **OPRE**, 2025a], [**L**-Peng-Yang, 2025]
- **Applications**: large-scale LPs in practice and research
- **Theoretical insights**: theoretical acceleration of enhancements

  [**L**-Yang, 2024]
- **Trajectory-based analysis**: refined characterization of dynamics

  [**L**-Yang, **MAPR**, 2025b], [**L**-Yang, 2024]

Extensions

- Convex quadratic programming (QP) [**L**-Yang, **MAPR**, 2025c]
- Semidefinite programming (SDP) [Ding-**L**-Yang, 2025]

# Primal-Dual Hybrid Gradient (PDHG) Method for LP

# Projected Gradient Descent for LP

LP (primal form):

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b,\ x \geq 0\ . \end{aligned}$$

# Projected Gradient Descent for LP

LP (primal form):

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b,\ x \geq 0\ . \end{aligned}$$

A natural first-order method: projected gradient descent

$$x^{k+1} = \mathbf{proj}_{Ax=b, x\geq 0}(x^k - \eta c)$$

- $\eta$ denotes the stepsize

# Projected Gradient Descent for LP

LP (primal form):

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b,\ x \geq 0\ . \end{aligned}$$

A natural first-order method: projected gradient descent

$$x^{k+1} = \mathbf{proj}_{Ax=b, x\geq 0}(x^k - \eta c)$$

- $\eta$ denotes the stepsize

However, the projection can be computationally expensive

# Primal-Dual Form of LP

LP (primal form):

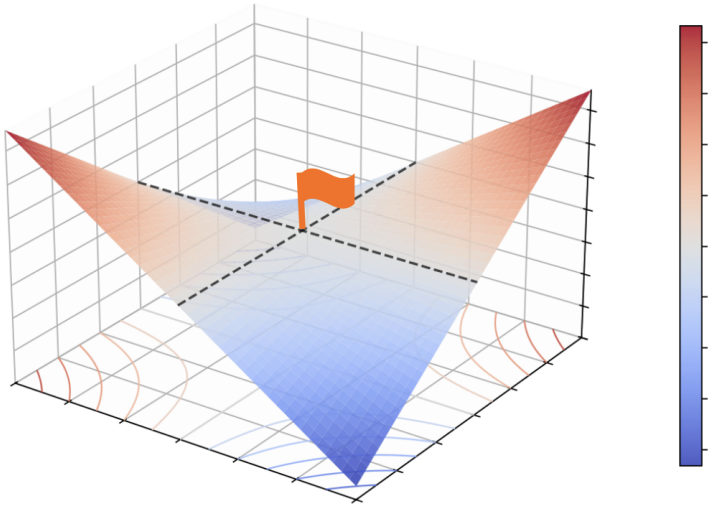
$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b,\ x \geq 0\ . \end{aligned}$$

LP (primal-dual form):

$$\min_{x \geq 0} \max_{y} c^\top x + b^\top y - y^\top A x =: L(x, y)$$

Our goal here is to (approximately) find a saddle point $(x^*, y^*)$

$$L(x^*, y) \leq L(x^*, y^*) \leq L(x, y^*)\ ,$$
for any $(x, y) \in \mathbb{R}^n_+ \times \mathbb{R}^m$



# Gradient Descent Ascent (GDA)

A natural first-order method: gradient descent ascent (GDA)

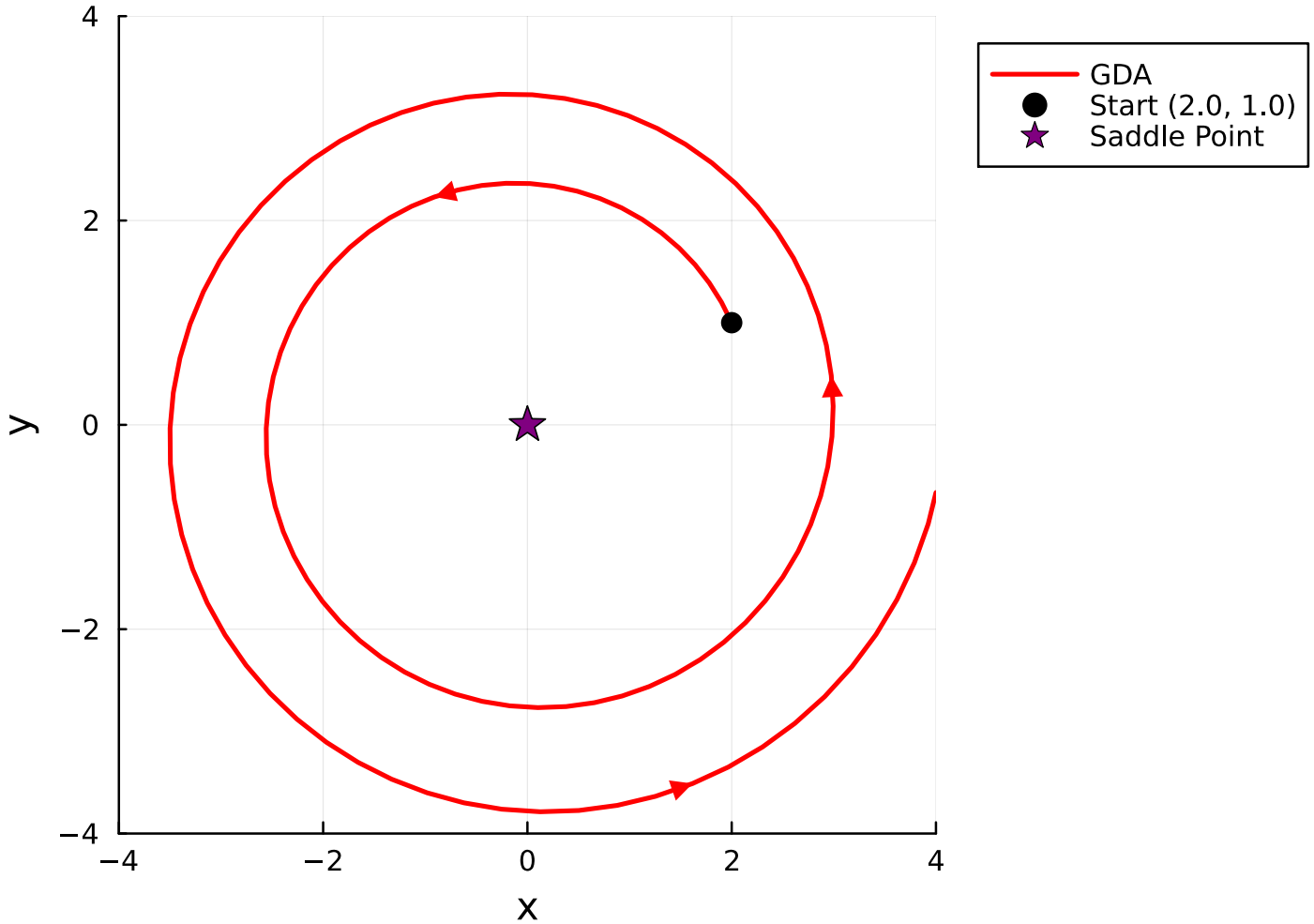
$$x^{k+1} = \mathbf{proj}_{\mathbb{R}^n_+}(x^k - \eta(c - A^\top y^k))$$
$$y^{k+1} = y^k + \tau(b - Ax^k)$$

- $\eta$ and $\tau$ are primal and dual stepsizes, respectively
- Simple projection (onto nonnegative orthant)
- The dominant computational cost is matrix-vector multiplication

# Behavior of GDA

Unfortunately, GDA fails to converge to the saddle point of bilinear problem

$$\min_x \max_y \; L(x, y) := xy$$



# Proximal Point Method (PPM)

Proximal point method (PPM) [Rockafellar, 1976]

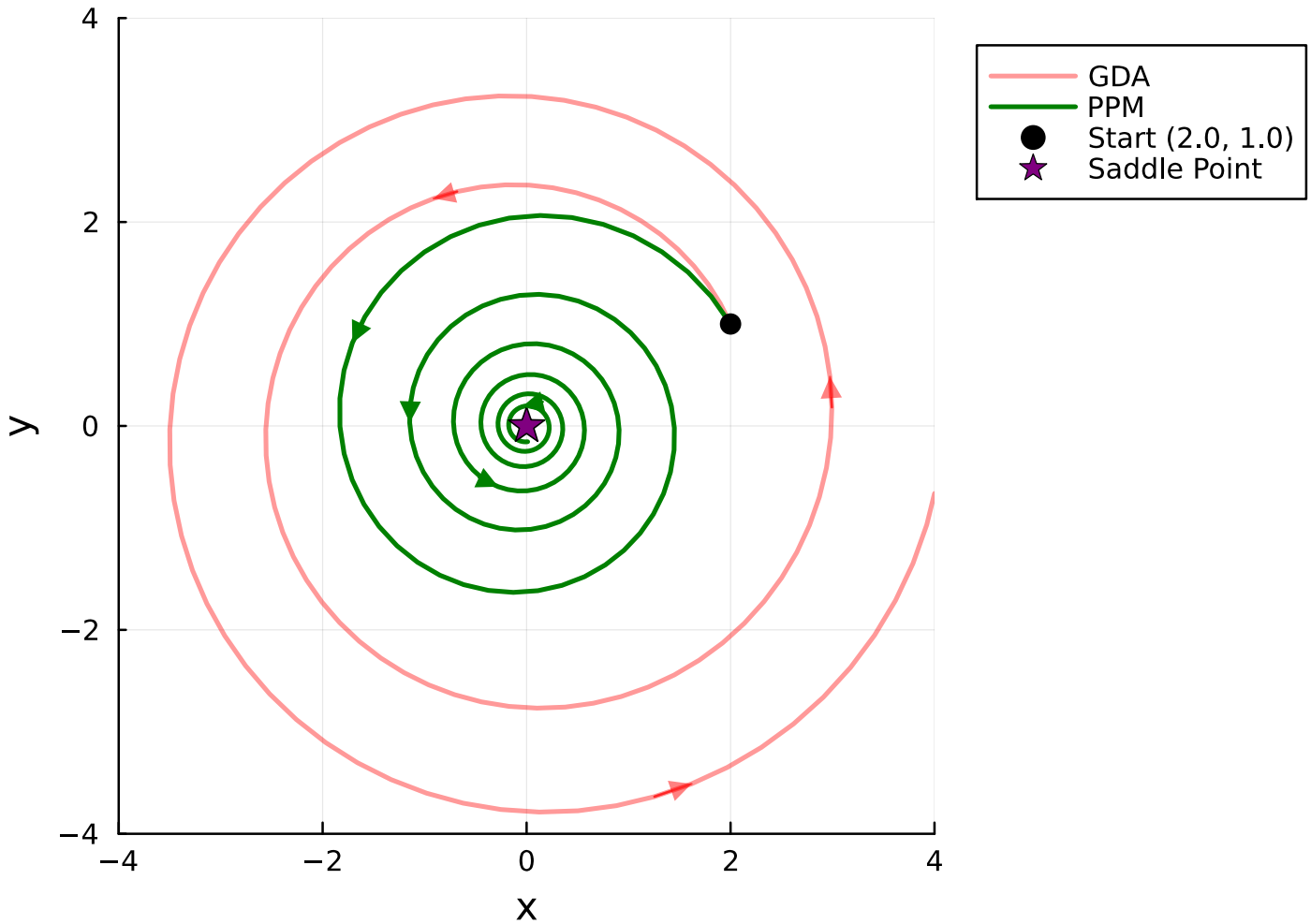
$$x^{k+1} = \mathbf{proj}_{\mathbb{R}^n_+}(x^k - \eta(c - A^\top y^{k+1}))$$
$$y^{k+1} = y^k + \tau(b - Ax^{k+1})$$

- $\eta$ and $\tau$ are primal and dual stepsizes, respectively
- Updates are implicit and require solving coupled equations

# Behavior of PPM

PPM converges to the saddle point of bilinear problem

$$\min_x \max_y \; L(x, y) := xy$$



# Primal-Dual Hybrid Gradient (PDHG)

Primal-Dual Hybrid Gradient (PDHG) [Chambolle-Pock, 2011]

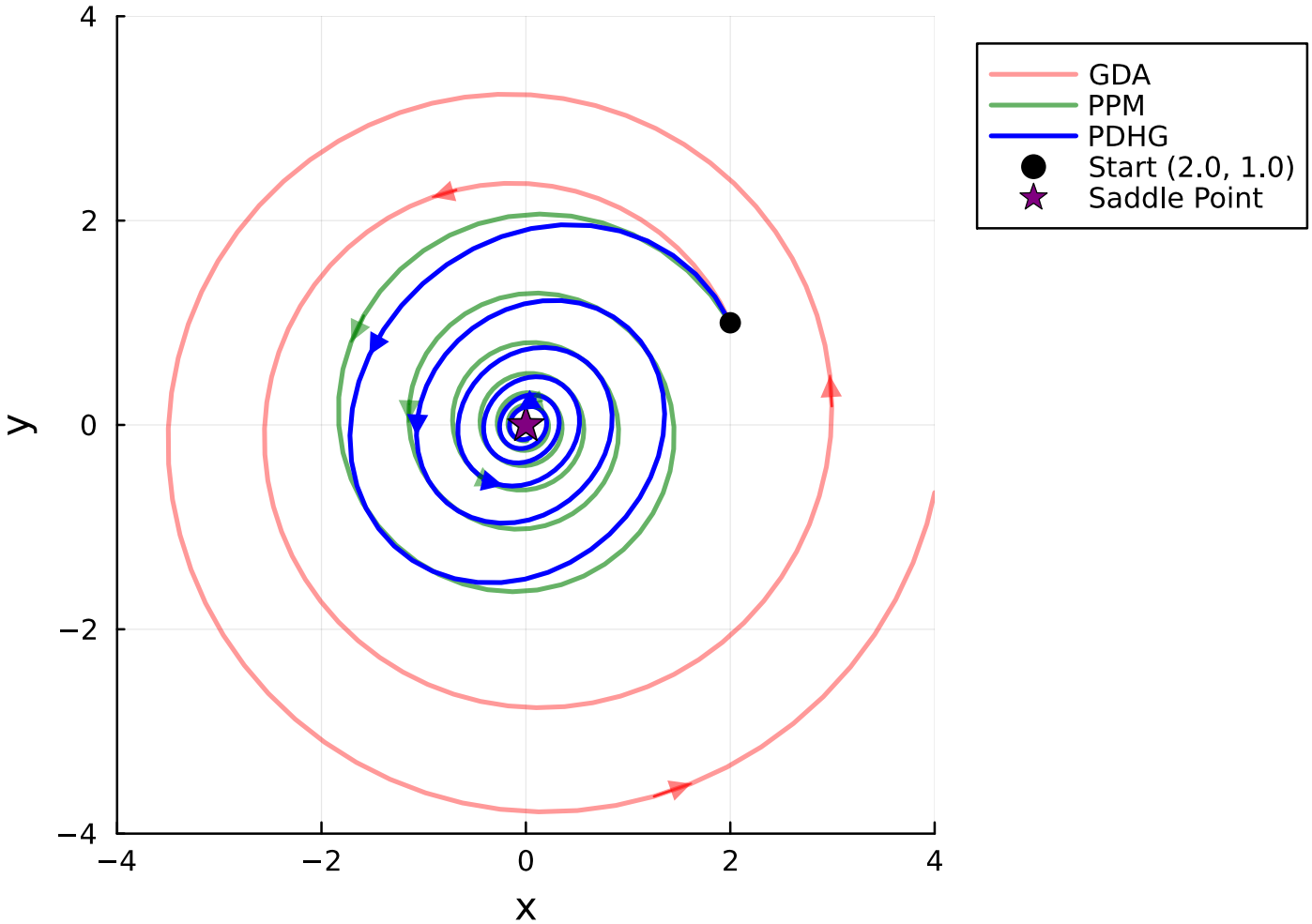
$$x^{k+1} = \mathbf{proj}_{\mathbb{R}^n_+}(x^k - \eta(c - A^\top y^k))$$
$$y^{k+1} = y^k + \tau(b - A(2x^{k+1} - x^k))$$

- $\eta$ and $\tau$ are primal and dual stepsizes, respectively
- The extrapolation makes PDHG a preconditioned PPM
- Simple projection (onto nonnegative orthant), as in GDA
- The dominant computational cost is matrix-vector multiplication, as in GDA
- Why PDHG work?
  - PDHG is a preconditioned PPM! It avoids implicit update.

# Behavior of PDHG

As PPM, PDHG converges to the saddle point of bilinear problem

$$\min_x \max_y \; L(x, y) := xy$$



# Recap

**Q: Why first-order methods (FOMs) for LP?**

- The dominant computational bottleneck is SpMV
- SpMV is highly parallelizable and scales efficiently on GPUs

# Recap

**Q: Why first-order methods (FOMs) for LP?**

- The dominant computational bottleneck is SpMV
- SpMV is highly parallelizable and scales efficiently on GPUs

**Q: Why solve the primal-dual problem?**

- Avoids expensive projections or linear system solves

# Recap

**Q: Why first-order methods (FOMs) for LP?**

- The dominant computational bottleneck is SpMV
- SpMV is highly parallelizable and scales efficiently on GPUs

**Q: Why solve the primal-dual problem?**

- Avoids expensive projections or linear system solves

**Q: Which method do we use?**

- Primal-dual hybrid gradient (PDHG) method

$$x^{k+1} = \mathbf{proj}_{\mathbb{R}^n_+}(x^k - \eta(c - A^\top y^k))$$
$$y^{k+1} = y^k + \tau(b - A(2x^{k+1} - x^k))$$

# Computational Results

Mostly based on

- [**L**-Yang, 2025a] **H Lu**, J Yang "cuPDLP.jl: A GPU Implementation of Restarted Primal-Dual Hybrid Gradient for Linear Programming in Julia", *Operations Research*, 2025
- [**L**-Peng-Yang, 2025] **H Lu**, Z Peng, J Yang "cuPDLPx: A Further Enhanced GPU-Based First-Order Solver for Linear Programming", *Major Revision at INFORMS Journal on Computing*, 2025

# Dataset and Setup

- MIPLIB Relaxations (383 instances)

| | **Small** | **Medium** | **Large** |
|---|---|---|---|
| **Number of nonzeros** | 100K - 1M | 1M - 10M | >10M |
| **Number of instances** | 269 | 94 | 20 |

Table: Scales of instances in MIPLIB Relaxations

- Hardware details

| | **CPU** | **GPU** |
|---|---|---|
| **Processor** | AMD EPYC 9335 | NVIDIA H100 SXM |
| **Launch date** | Q4, 2024 | Q4, 2022 |

- Termination criteria: relative primal feasibility, dual feasibility and duality gap below tolerance

# Is Vanilla PDHG Numerically Strong?

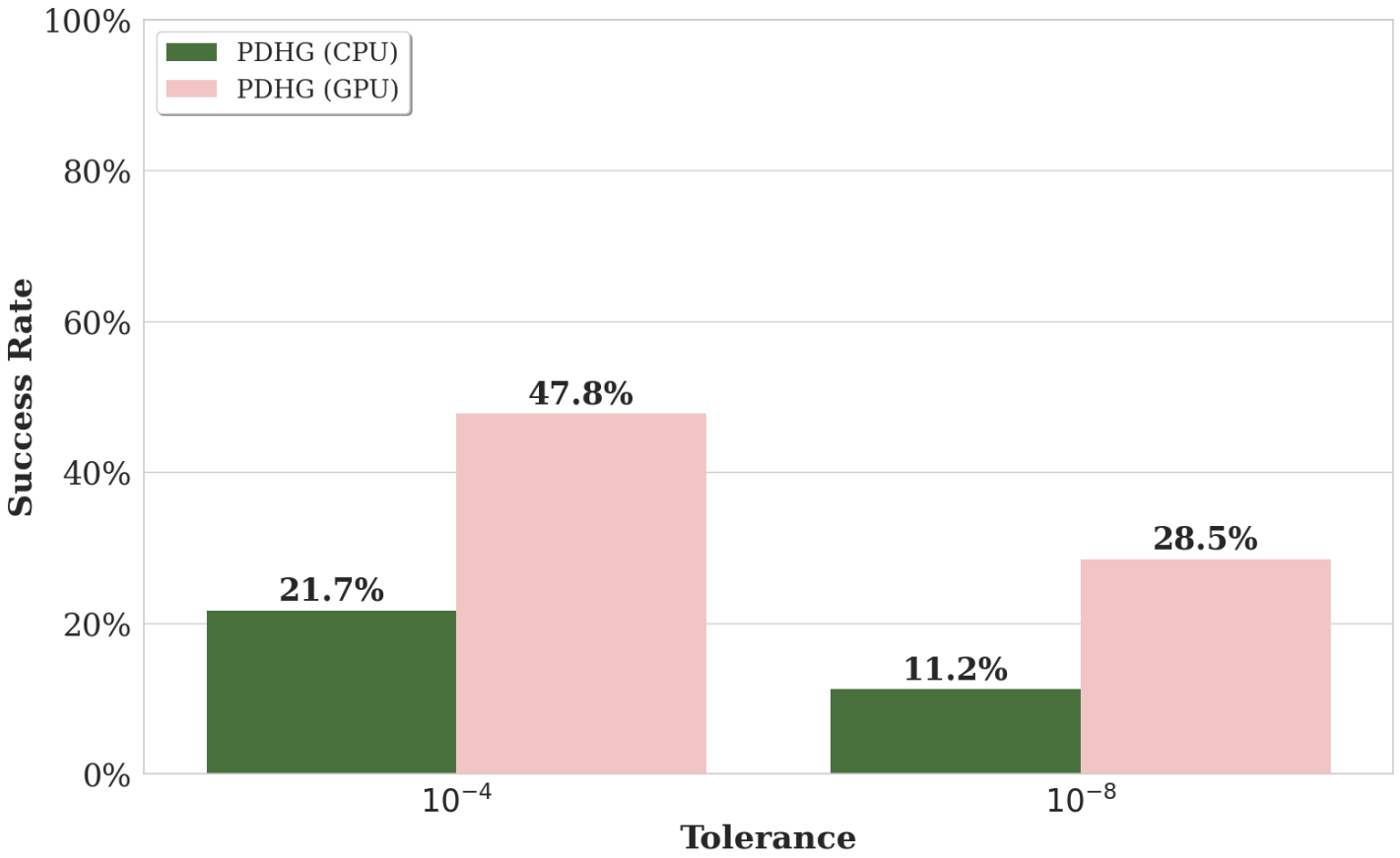



Figure: Solver success rate on instances of MIP Relaxations, with a 600s time limit

- PDHG on GPU is not sufficient on its own

# GPU-based LP Solvers

- cuPDLP ($\approx$ GPU-friendly variant of PDLP) [**L**-Yang, 2025a]
- **cuPDLPx** [**L**-Peng-Yang, 2025] [**L**-Yang, 2024]

**New Algorithm**

**Restarted Reflected Halpern PDHG ($r^2$HPDHG)**

+

**Numerical Techniques**

- **Diagonal preconditioning**
- **Primal weight update**
- **.....**

**Robust solvers depend on numerical techniques.**

The hard work: diagnose benchmark failures and design targeted fixes.

# A New Algorithm for LP: A High-Level Preview

- Restarted Reflected Halpern PDHG [**L**-Yang, 2024]

**PDHG**

$$x^{k+1} = \mathbf{proj}_{\mathbb{R}^n_+}(x^k - \eta(c - A^\top y^k))$$
$$y^{k+1} = y^k + \tau(b - A(2x^{k+1} - x^k))$$

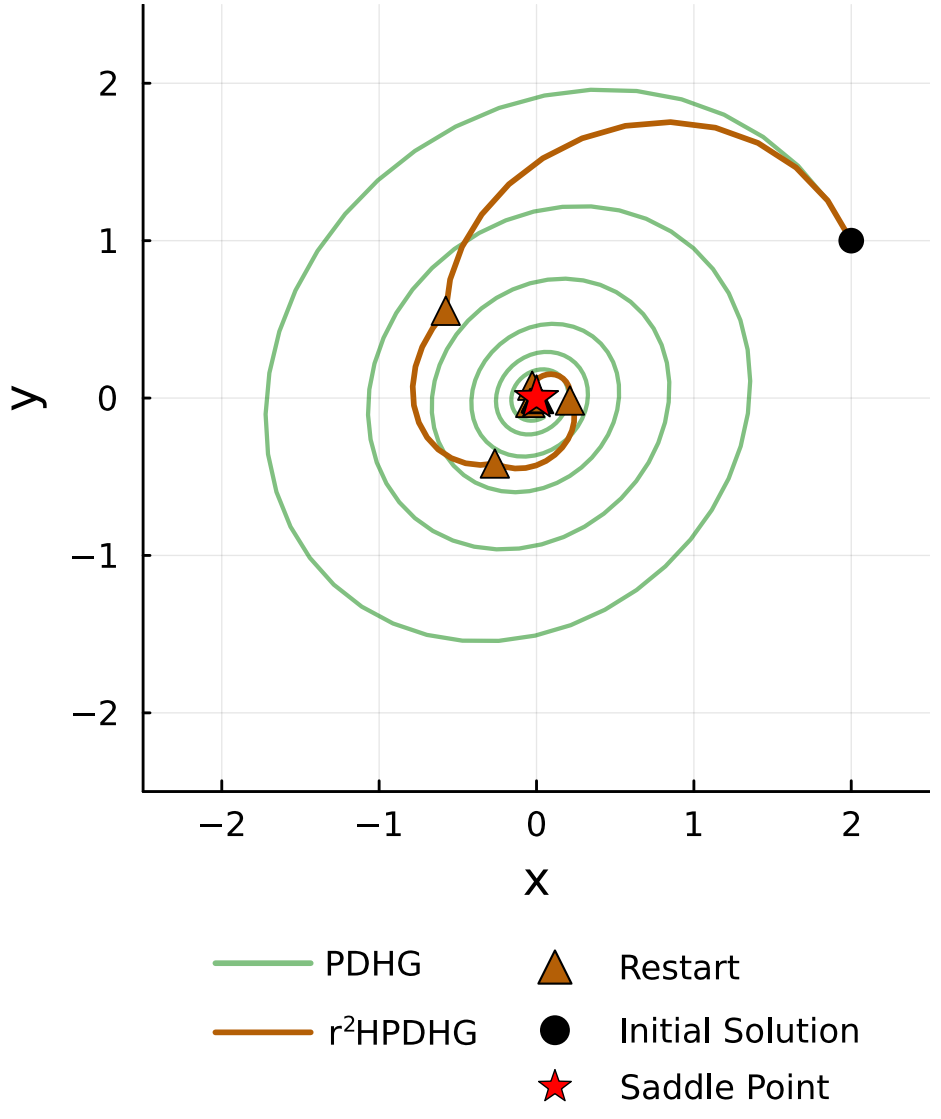
**r²HPDHG**

**Start Epoch** $n$

$$z^{n,t+1} \leftarrow \frac{t+1}{t+2}(2\mathrm{PDHG}(z^{n,t}) - z^{n,t}) + \frac{1}{t+2}z^{n,0}$$

**End**

$$z^{n+1,0} \leftarrow \mathrm{PDHG}(z^{n,t+1})$$



# cuPDLP/cuPDLPx versus PDHG

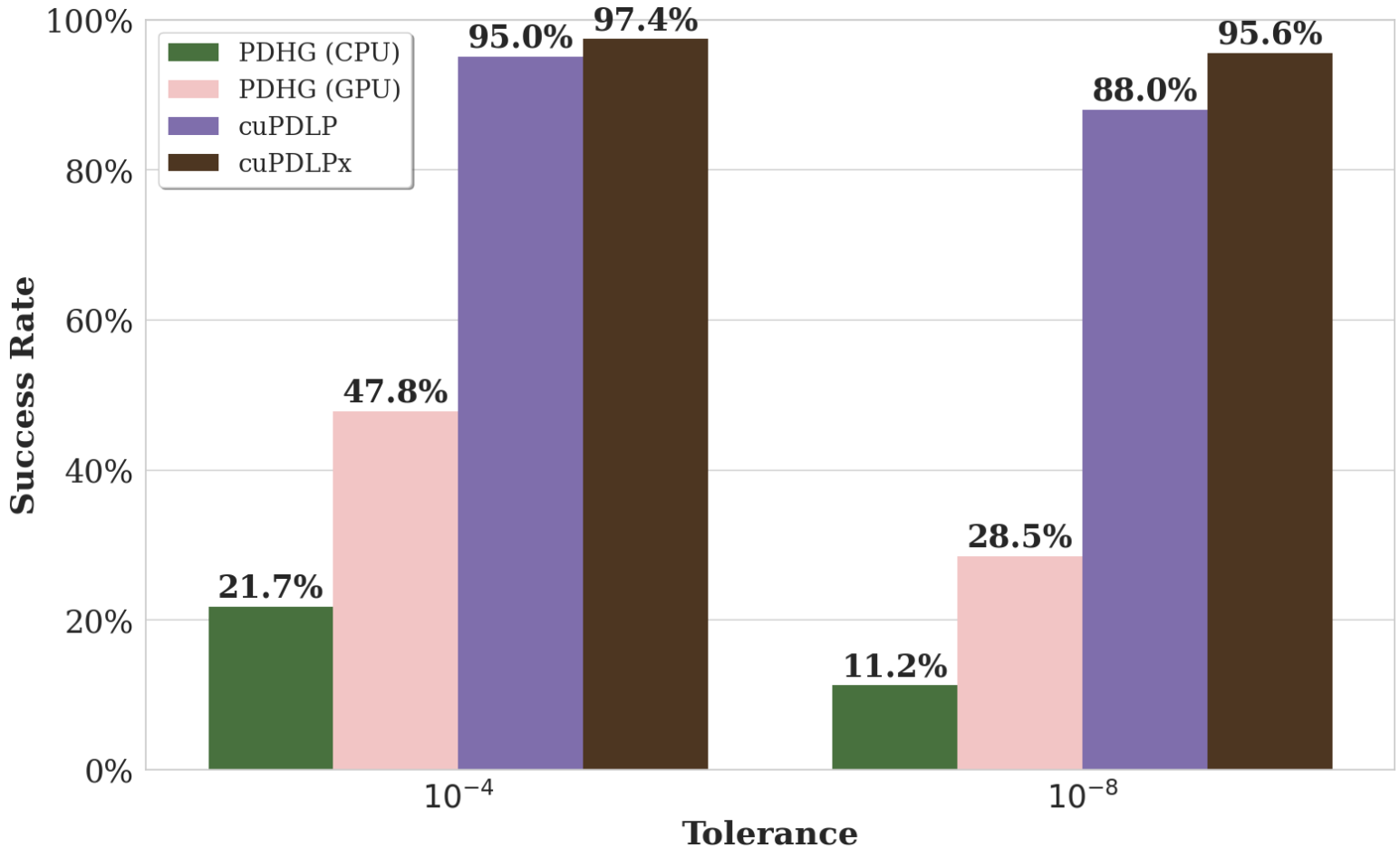



Figure: Solver success rate on instances of MIP Relaxations, with a 600s time limit

# cuPDLP/cuPDLPx versus Gurobi

| | | Small (269) (1-hour limit) | | Medium (94) (1-hour limit) | | Large (20) (5-hour limit) | |
|---|---|---|---|---|---|---|---|
| | | **Count** | **Time** | **Count** | **Time** | **Count** | **Time** |
| $10^{-4}$ | **Primal Simplex (Gurobi)** | 268 | 8.29 | 73 | 124.97 | 13 | 1582.04 |
| | **Dual Simplex (Gurobi)** | 266 | 6.10 | 84 | 46.76 | 15 | 372.81 |
| | **Interior-Point (Gurobi)** | 269 | 2.00 | 90 | 20.05 | 17 | 221.09 |
| | **cuPDLP** | 266 | 8.81 | 91 | 11.55 | 19 | 77.43 |
| | **cuPDLPx** | 269 | 3.50 | 93 | 5.51 | 19 | 62.04 |
| $10^{-8}$ | **Primal Simplex (Gurobi)** | 268 | 8.27 | 78 | 108.56 | 15 | 1167.18 |
| | **Dual Simplex (Gurobi)** | 268 | 5.41 | 86 | 41.43 | 16 | 413.23 |
| | **Interior-Point (Gurobi)** | 269 | 2.58 | 90 | 22.55 | 15 | 346.07 |
| | **cuPDLP** | 260 | 24.80 | 87 | 36.90 | 17 | 208.66 |
| | **cuPDLPx** | 264 | 8.94 | 91 | 16.43 | 17 | 194.91 |

Table: Solve time in seconds and SGM10 of different solvers on instances of MIP Relaxations

- Gurobi 12.0 (crossover disabled for IPM; presolve disabled)

# cuPDLP/cuPDLPx versus Gurobi, with Gurobi presolve

| | | Small (269) (1-hour limit) | | Medium (94) (1-hour limit) | | Large (20) (5-hour limit) | |
|---|---|---|---|---|---|---|---|
| | | **Count** | **Time** | **Count** | **Time** | **Count** | **Time** |
| $10^{-4}$ | **Primal simplex (Gurobi)** | 269 | 3.72 | 74 | 87.16 | 18 | 180.64 |
| | **Dual simplex (Gurobi)** | 268 | 2.66 | 87 | 26.23 | 18 | 90.63 |
| | **Interior-Point (Gurobi)** | 269 | 0.54 | 94 | 7.54 | 20 | 17.04 |
| | **cuPDLP** | 268 | 5.11 | 92 | 9.03 | 19 | 26.74 |
| | **cuPDLPx** | 269 | 1.41 | 94 | 2.75 | 19 | 14.07 |
| $10^{-8}$ | **Primal simplex (Gurobi)** | 269 | 3.45 | 78 | 72.53 | 18 | 140.82 |
| | **Dual simplex (Gurobi)** | 269 | 2.41 | 89 | 19.31 | 19 | 77.57 |
| | **Interior-Point (Gurobi)** | 269 | 0.59 | 94 | 8.15 | 20 | 19.15 |
| | **cuPDLP** | 264 | 18.50 | 90 | 29.40 | 19 | 63.68 |
| | **cuPDLPx** | 269 | 3.98 | 92 | 9.60 | 19 | 43.09 |

Table: Solve time in seconds and SGM10 of different solvers on instances of MIP Relaxations: cuPDLP/cuPDLPx versus Gurobi with presolve.

- Gurobi 12.0 (crossover disabled for IPM; Gurobi presolve enabled)

# cuPDLPx: v0.1.0 versus v0.3.0

| | | Small (269) (1-hour limit) | | Medium (94) (1-hour limit) | | Large (20) (5-hour limit) | |
|---|---|---|---|---|---|---|---|
| | | **Count** | **Time** | **Count** | **Time** | **Count** | **Time** |
| $10^{-4}$ | **v0.1.0** | 269 | 2.77 | 94 | 4.62 | 19 | 57.18 |
| | **v0.3.0** | 269 | 1.05 | 94 | 2.967 | 19 | 16.95 |
| $10^{-8}$ | **v0.1.0** | 266 | 6.84 | 92 | 13.91 | 16 | 194.40 |
| | **v0.3.0** | 269 | 2.89 | 93 | 9.65 | 19 | 45.65 |

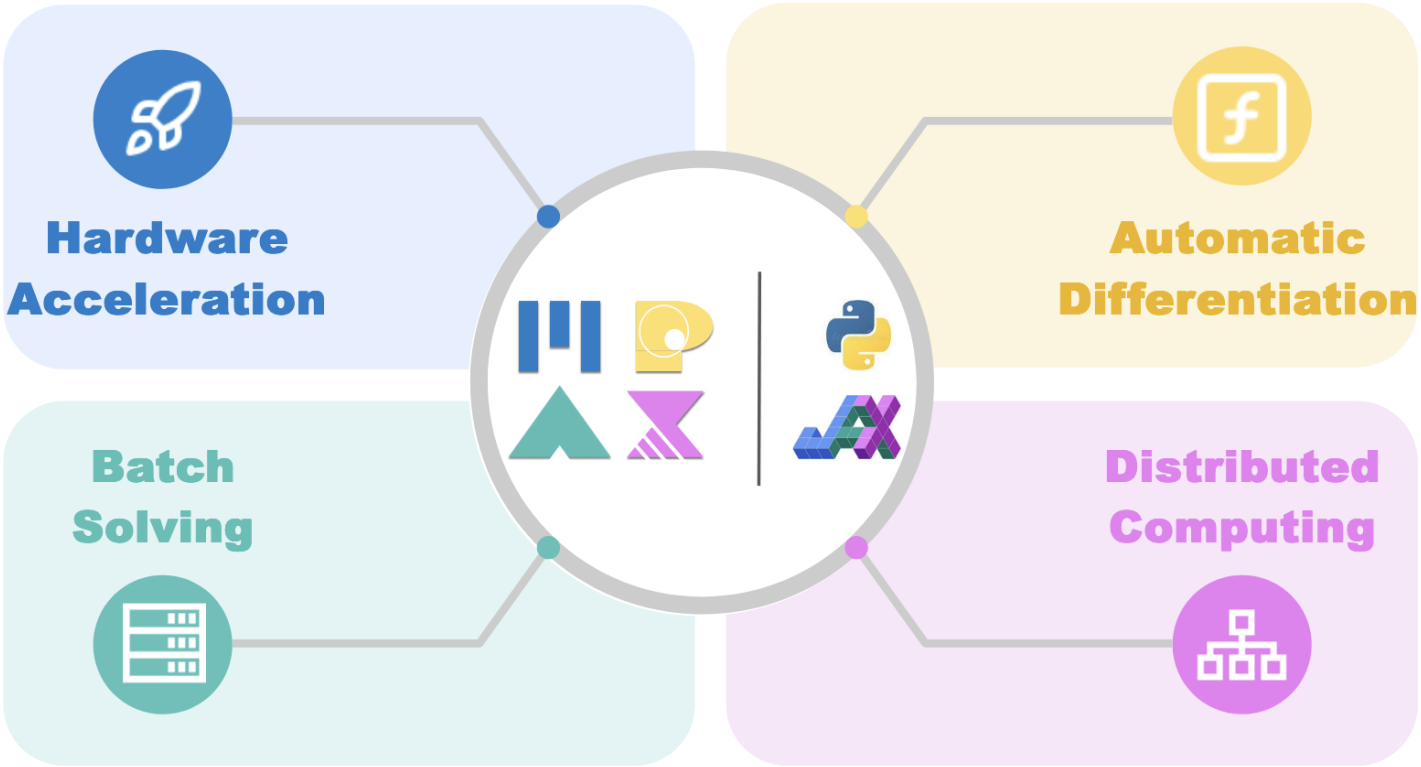
Table: Solve time in seconds and SGM10 of different solvers on instances of MIP Relaxations

- v0.1.0 -¿ v0.3.0: added PSLP presolver, CUDA graph, new SpMV backend, active set boost

# MPAX: Math Programming in JAX

cuPDLPx is also available in **MPAX** (**M**ath **P**rogramming in J**AX**) [**L**-Peng-Yang, 2024]

- JAX is a Python library designed for large-scale machine learning
- Motivation: enable optimization to be used as composable building blocks



# Major Message

- cuPDLPx demonstrates advantages in solving large LPs
- New LP-specific algorithms are critical for fast and robust convergence in practice

# Applications

1. **Applications in Practice**

   Google, NVIDIA, and Gridmatic / COPT.

2. **Direct Applications in Research Papers**

   Apply general-purpose GPU solvers such as cuPDLPx directly.
   Shipping (Google), energy systems (ZIB), targeted marketing, sponsored listings (eBay), and production inventory.

3. **Applications Using Problem Structure in Research Papers**

   Develop reformulations and specialized GPU algorithms.
   Portfolio optimization, QUBO, and joint replenishment.

# Applications in Practice

Large-scale optimization in industry

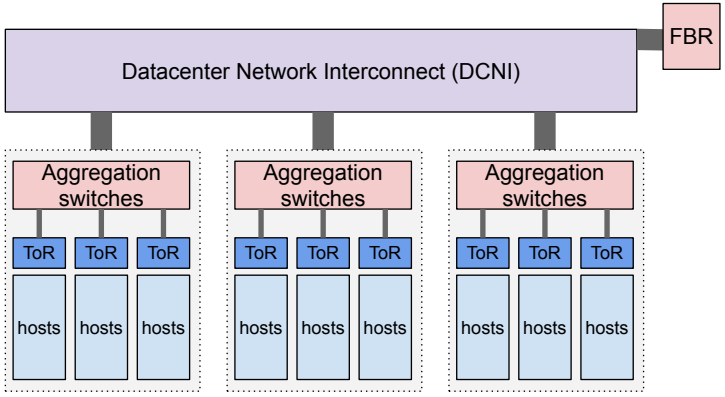
- Google: traffic engineering within a data center (DCNI)
- NVIDIA: supply chain planning with agentic AI and cuOpt
- Gridmatic / COPT: unit commitment and economic dispatch

# Google: Traffic Engineering Within a Data Center

## DCNI: Data Center Network Interconnect

**Goal:** fulfill application demand using the machines and network within a data center.

- Requests generate traffic among machine clusters.
- Decide how to split traffic across network paths.
- A variant of **multi-commodity flow**: demands share limited link capacity.




Google's data center network

$$\sum_k f_e^k \leq c_e$$

Total flow on link $e$ stays within capacity.

# NVIDIA: Supply Chain Planning

**Scale:** hundreds of partners, thousands of components, and nearly 3,000 products.

- **Planning decisions:** allocate available supply to meet customer commitments.
- **Changing conditions:** supplier delays, capacity disruptions, and changes in demand require new allocation plans.
- **What-if analysis:** compare fulfillment and allocation under many supply and demand scenarios.

**NVIDIA's operations team uses a cuOpt-powered AI planner** to run these scenarios through natural-language questions.

# Direct Applications in Research Papers

General-purpose GPU solvers on existing models

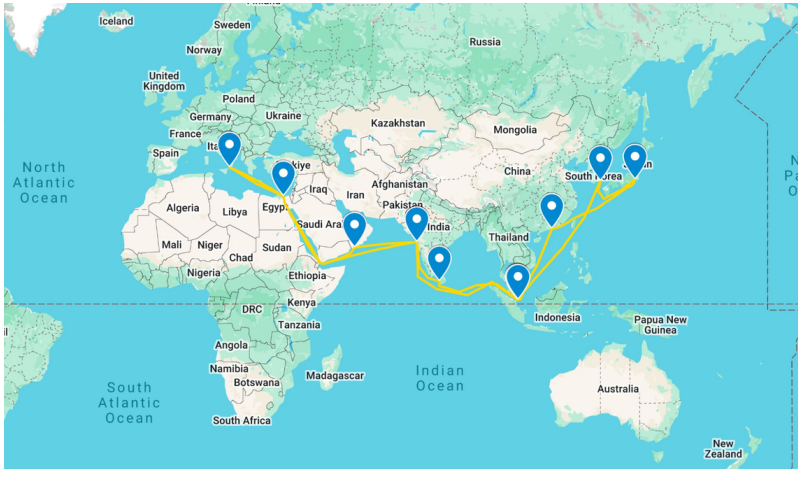
- Liner-Shipping Network Design and Scheduling Problem (Google)
- Energy System Optimization Problem (ZIB)
- Targeted Marketing Policies
- Sponsored Listings Ranking (eBay)
- Production Inventory Problem

# Liner-Shipping Network Design and Scheduling Problem

**Goal:** design a global liner-shipping network that moves containers from origins to destinations efficiently and profitably.

- **Network design:** determine the order in which vessels visit ports.
- **Network scheduling:** determine when vessels arrive at and leave each port.
- **Container routing:** choose the journey that containers take from origin to destination.



# Liner-Shipping Network Design Problem (Primarily by Google)

An LP relaxation of a MIP formulation of a LINERLIB instance [Brouer et al., 2014]

| Name | Variables | Constraints | Nonzeros | Memory Usage |
|---|---|---|---|---|
| world-shipping | 228,867,510 | 15,304,282 | 688,658,522 | 14.79 GB |

Table: Size of world-shipping instance

| Solver | Hardware | Solve Time |
|---|---|---|
| Interior-Point (Gurobi)* | CPU with 1TB memory | Out of memory |
| PDLP (Google OR-Tools)* | CPU with 256GB memory | 53.8 hours |
| cuPDLPx | GPU with 80GB memory | 422.7 seconds |

Table: Performance comparison on world-shipping instance. Terminate when primal and dual feasibility is less than $10^{-8}$ and the relative duality gap is less than 1%. Results marked with * are from [Applegate et al., 2025]

# Energy System Optimization Problem

**Goal:** optimize the operation and expansion of the German power system under hourly demand, generation, storage, and network constraints. [Sasanpour et al., 2025]

- **Expansion decisions:** renewable generation, gas-fired plants, storage, and transmission capacity.
- **Operational decisions:** hourly dispatch, storage levels, imports/exports, and power flows.
- **Unit commitment:** on/off decisions for conventional generators.





# Energy System Optimization Problem (By Zuse Institute Berlin)

LP relaxations of German energy system optimization models [Kempke-Koch, 2025]

- Model size varies due to different spatial aggregations

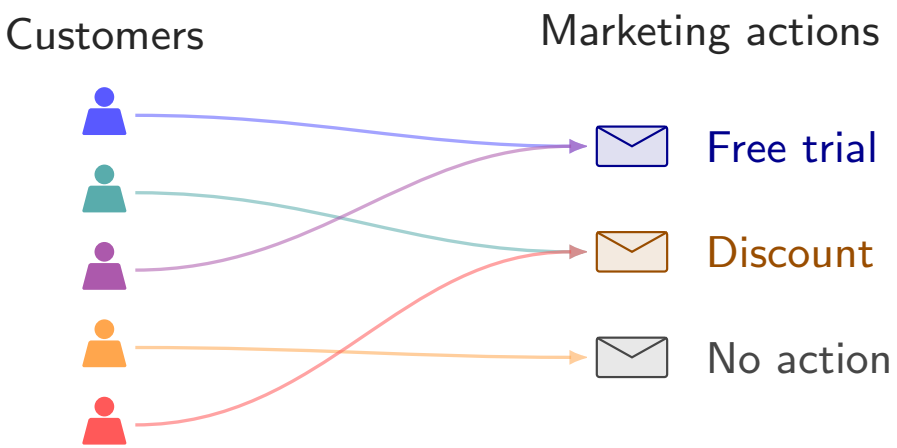
| Test Set | Nonzeros | cuPDLP | Interior-Point | Speedup |
|---|---|---|---|---|
| **X-Small** | 2,829,975 | 13.10 | 29.87 | 2.28× |
| **Small** | 4,590,593 | 9.18 | 72.34 | 7.88× |
| **Medium** | 47,310,667 | 104.21 | 1002.34 | 9.62× |
| **Large** | 82,773,747 | 394.82 | 4354.79 | 11.03× |
| **X-Large** | 105,960,838 | 148.0 | 11296.42 | 76.33× |
| **XX-Large** | 243,495,063 | 437.52 | TIMEOUT | - |

Table: Solve time in seconds across different test sets, with a two-day time limit [Kempke-Koch, 2025]

# Targeted Marketing Policies

**Goal**: For each prospective customer $i$, picks which marketing action $j$ to take to maximize total incremental profit. [**L** et al., 2023]

- **Volume:** min / max number of actions per segment (capacity, budget, supplier agreements).
- **Similarity (fairness):** similar customer segments should receive similar marketing actions.



# Targeted Marketing Policies

**Instance construction**: experiment data from a large American retailer in 2015. [Simester et al., 2020]

| Instance | Variables | Constraints | Nonzeros |
|---|---|---|---|
| OH | 5,307,190 | 1,061,828 | 122,065,370 |
| All | 11,850,480 | 2,372,040 | 604,374,480 |

Table: Sizes of the marketing LP instances.

| Instance | Tolerance | cuPDLPx (s) | Gurobi (s) | Speedup |
|---|---|---|---|---|
| OH | $10^{-4}$ | 15.01 | 270.53 | **18.02**× |
| | $10^{-8}$ | 46.94 | 299.49 | **6.38**× |
| All | $10^{-4}$ | 53.17 | 1234.07 | **23.21**× |
| | $10^{-8}$ | 259.22 | 1393.30 | **5.38**× |

Table: Solve time and speedup of cuPDLPx versus Gurobi.

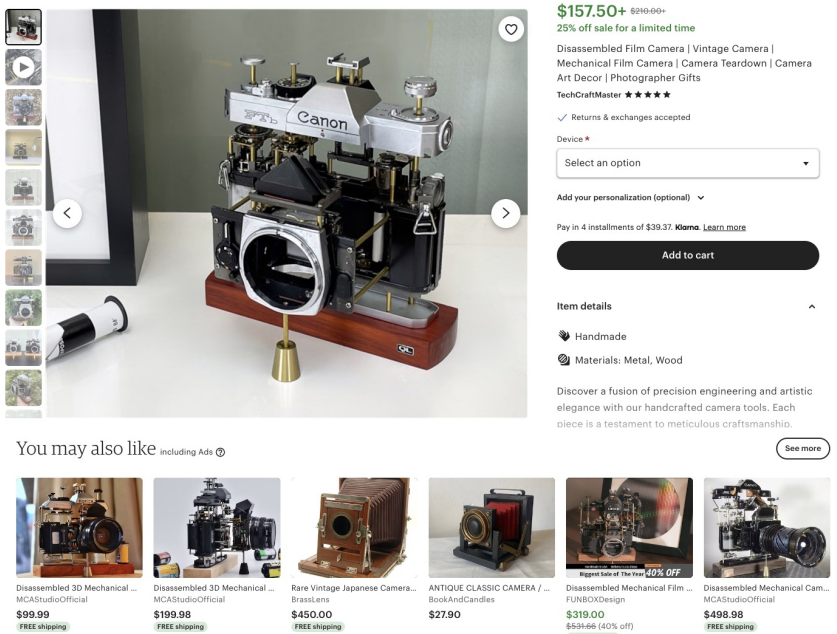
- **Gurobi**: 16 Intel Xeon Platinum 8462Y+ CPU cores with 256 GB RAM;
- **cuPDLPx**: one NVIDIA H100 GPU.

# Sponsored Listings Ranking (eBay)

**Per visit.** ranks candidate sponsored items into a few dozen slots, balancing revenue and relevance. [**L** et al., 2025]

**Across visits.** The marketplace also has **global business goals**:

- **Per seller:** inventory caps, revenue floors, exposure floors.
- **Per user:** diversity / fairness across the sellers they see.



# Sponsored Listings Ranking (eBay)

**Instance construction** : one full day of historical impression data; filtered to the top **5,408** sellers (most listed items).

| Instance | Variables | Constraints | Nonzeros |
|---|---|---|---|
| Overall planning LP | 131,984,284 | 13,129,841 | 428,565,283 |

Table: Instance size.

| Tolerance | cuPDLPx (s) | Gurobi (s) | Speedup |
|---|---|---|---|
| $10^{-4}$ | 121.26 | 2575.31 | **21.24**× |
| $10^{-8}$ | 3755.09 | 4520.83 | **1.20**× |

Table: Solve time and speedup of cuPDLPx versus Gurobi Barrier

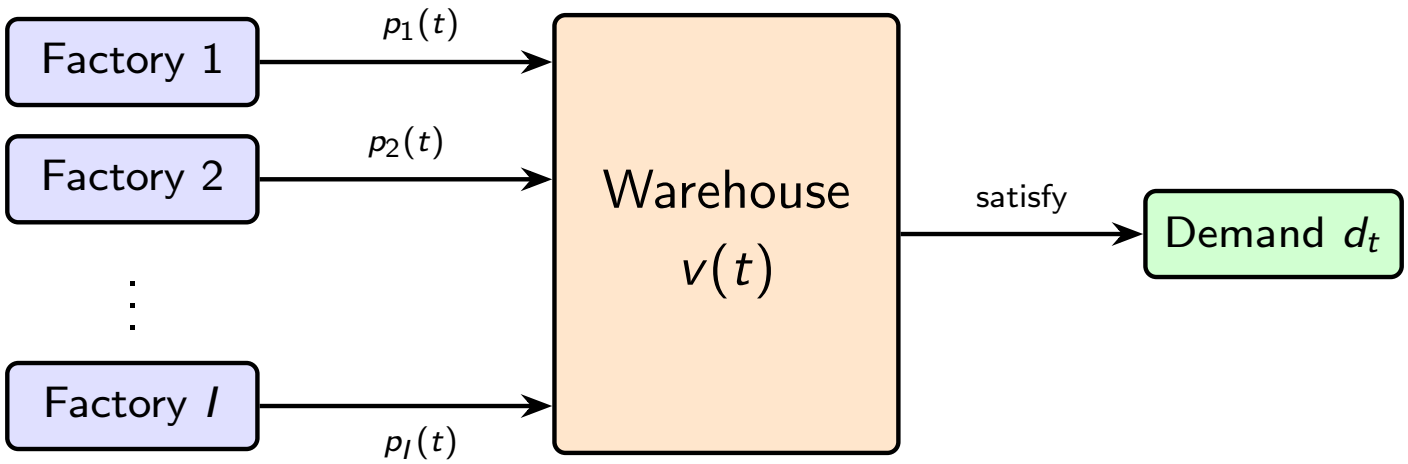
- **Gurobi**: 16 Intel Xeon Platinum 8462Y+ CPU cores with 256 GB RAM;
- **cuPDLPx**: one NVIDIA H100 GPU.

# Production Inventory Problem

**Setting**: A product inventory system with a warehouse and $I$ factories.

**Goal**: minimize the total production cost over all factories and the entire planning period. [Ben-Tal et al., 2004]

- All the demand must be satisfied.
- Per-period and cumulative production capacity of factories.
- Storage capacity of the warehouse.



# Production Inventory Problem

**Instance construction**: data is randomly generated. [Applegate et al., 2025]

| Instance | Variables | Constraints | Nonzeros |
|---|---|---|---|
| Production Inventory | 18,270,600 | 4,650,850 | 504,639,850 |

Table: Instance size.

| Tolerance | cuPDLPx (s) | Gurobi (s) | Speedup |
|---|---|---|---|
| $10^{-4}$ | 1,233.79 | OOM | – |
| $10^{-8}$ | 5,297.88 | OOM | – |

Table: Solve time and speedup of cuPDLPx versus Gurobi Barrier.

- **Gurobi**: 16 Intel Xeon Platinum 8462Y+ CPU cores with 256 GB RAM;
- **cuPDLPx**: one NVIDIA H100 GPU.

# Applications Using Problem Structure in Research Papers

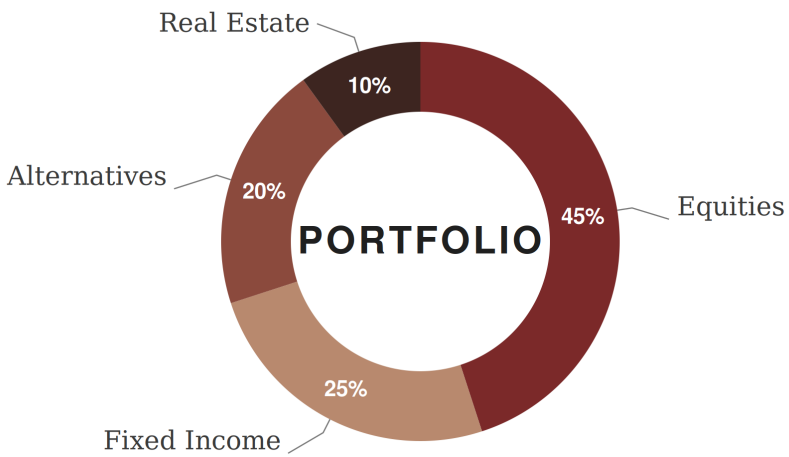
Reformulations and specialized GPU algorithms

- Portfolio optimization: a specialized GPU solver (FlashFolio)
- QUBO: reformulate the LP relaxation (PRAQ)
- Joint replenishment: exploit inventory structure (PRIA)

# Portfolio Optimization

**Goal:** optimize portfolio construction (single-period) or trade execution (multi-period) under realistic market and investment constraints.

- **Objective:** balance expected return, portfolio risk, and transaction costs.
- **Constraints:** enforce portfolio exposure limits and investment budget constraints.



# Portfolio Optimization

**Instances construction:** U.S. stock market data (∼4000 assets).

| Parameters | | Single-period model (T=1) | | | | | Multi-period model (T=10) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lambda_2, \lambda_3$ | $\lambda_1$ | FlashFolio | | MOSEK | | **Speedup** | FlashFolio | | MOSEK | | **Speedup** |
| | | Count | Time | Count | Time | | Count | Time | Count | Time | |
| 1 | $10^{-4}$ | 21 | 0.24 | 21 | 0.71 | **3.0**× | 21 | 0.88 | 21 | 13.88 | **15.8**× |
| | $10^{-6}$ | 21 | 0.14 | 21 | 1.80 | **12.9**× | 21 | 0.93 | 21 | 21.84 | **23.5**× |
| | $10^{-8}$ | 21 | 0.64 | 21 | 1.77 | **2.8**× | 21 | 1.77 | 21 | 30.87 | **17.4**× |
| 100 | $10^{-4}$ | 21 | 0.30 | 21 | 0.94 | **3.1**× | 21 | 1.06 | 21 | 29.25 | **27.6**× |
| | $10^{-6}$ | 21 | 0.22 | 21 | 1.22 | **5.5**× | 21 | 1.10 | 21 | 31.22 | **28.4**× |
| | $10^{-8}$ | 21 | 0.22 | 21 | 1.71 | **7.8**× | 21 | 1.11 | 21 | 40.93 | **36.9**× |
| 10000 | $10^{-4}$ | 21 | 0.63 | 21 | 1.49 | **2.4**× | 21 | 1.44 | 21 | 28.47 | **19.8**× |
| | $10^{-6}$ | 21 | 0.61 | 21 | 1.69 | **2.8**× | 21 | 1.44 | 20 | 34.96 | **24.3**× |
| | $10^{-8}$ | 21 | 0.63 | 21 | 1.66 | **2.6**× | 21 | 1.45 | 17 | 33.73 | **23.3**× |

Table: Performance comparison of FlashFolio and MOSEK. (Time in seconds) $\lambda_1, \lambda_2, \lambda_3$ are the penalty factors for risk, bid-offer spread, and market impact, respectively.

- MOSEK: 16 Intel Xeon Platinum 8462Y+ CPU cores with 256 GB RAM;
- FlashFolio: one NVIDIA H100 GPU.

# QUBO: Background and LP Reformulation

**Quadratic unconstrained binary optimization:**

$$\min_{x \in \{0,1\}^n} x^\top W x$$

Binary decisions with pairwise interactions; the binary problem is NP-hard.

**LP relaxation**

- Linearize off-diagonal products $x_i x_j$ and relax $x \in [0,1]^n$.
- The **roof-dual LP** provides lower bounds and variable-fixing information for branch-and-bound.

**PRAQ** [Dentes–**L**–Yang, 2026]:

- Absorb linearization constraints into simple box projections.
- Degree-based scaling bounds the coupling norm by 1; GPU iterations use clipping, edge gathers, and scatters.

# QUBO: Numerical Results

**Benchmarks:** 2,282 generated instances, spanning five edge-count groups.

PRAQ [Dentes–**L**–Yang, 2026] versus cuPDLPx on the standard LP.

| Edges | Instances | PRAQ (s) | cuPDLPx (s) | Speedup |
|---|---|---|---|---|
| $10^3$–$10^4$ | 897 | 0.0054 | 0.052 | 9.6× |
| $10^4$–$10^5$ | 767 | 0.012 | 0.080 | 6.8× |
| $10^5$–$10^6$ | 385 | 0.048 | 0.36 | 7.5× |
| $10^6$–$10^7$ | 193 | 0.59 | 3.7 | 6.4× |
| $10^7$–$10^8$ | 40 | 3.7 | 28.4 | 7.7× |

Table: Solve time: shifted geometric mean with a 1 ms shift.

Same H100 80 GB GPU; relative duality gap $10^{-8}$.

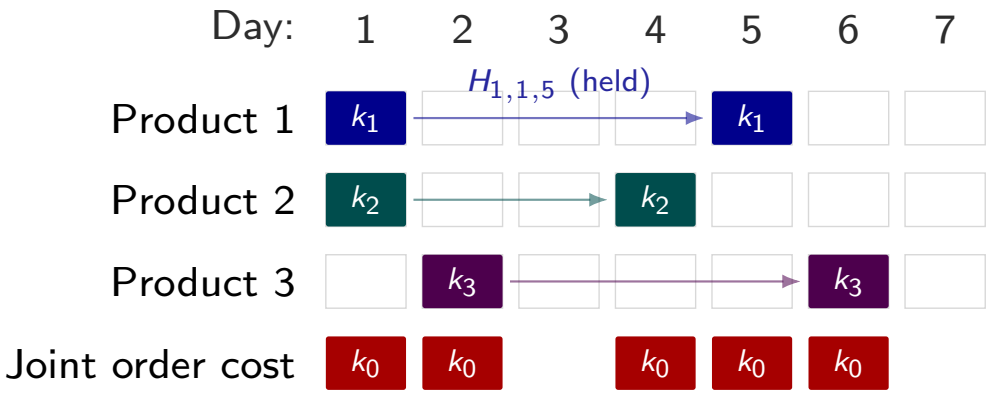
Both solvers solve all 2,282 instances and return the same LP bound.

**PRAQ is 6.4–9.6× faster on these LP relaxations.**

# Joint Replenishment Problem

Joint replenishment is an inventory management problem

- We can produce $N$ products over $T$ time periods
- We pay the fixed order cost $\boldsymbol{k}_i$ for each day we produce product $i$
- We pay the holding cost $\boldsymbol{H}_{ist}$ to hold over any amount of product $i$ from day $s$ to day $t$
- We pay the joint order cost $k_0$ for each day we produce any product



# Joint Replenishment: Reformulation and New Results

**Reformulation** [Dentes–**L**–Yang, 2026]:

- Eliminate order variables and absorb assignment constraints into simplex projections.
- **PRIA**: coupling norm $\sqrt{2}$, with no sparse matrix-vector products; projected outputs are feasible for the LP relaxation.

| **Assignment variables** | **PRIA (s)** | **cuPDLPx (s)** | **Speedup** |
|---|---|---|---|
| $10^3$–$10^4$ | 0.031 | 0.187 | 6.1× |
| $10^4$–$10^5$ | 0.136 | 1.32 | 9.7× |
| $10^5$–$10^6$ | 0.61 | $>$ 17.5 | $>$ 28.9× |
| $10^6$–$10^7$ | 2.88 | $>$ 80.4 | $>$ 27.9× |
| $10^7$–$10^8$ | 11.3 | $>$ 234 | $>$ 20.8× |

Table: Shifted geometric mean with a 0.01 s shift.

363 generated instances; one H100 80 GB GPU for each solver.
300 s limit per instance; $>$ marks lower bounds from timeouts.
cuPDLPx solves the original LP formulation.

# Theoretical Insights

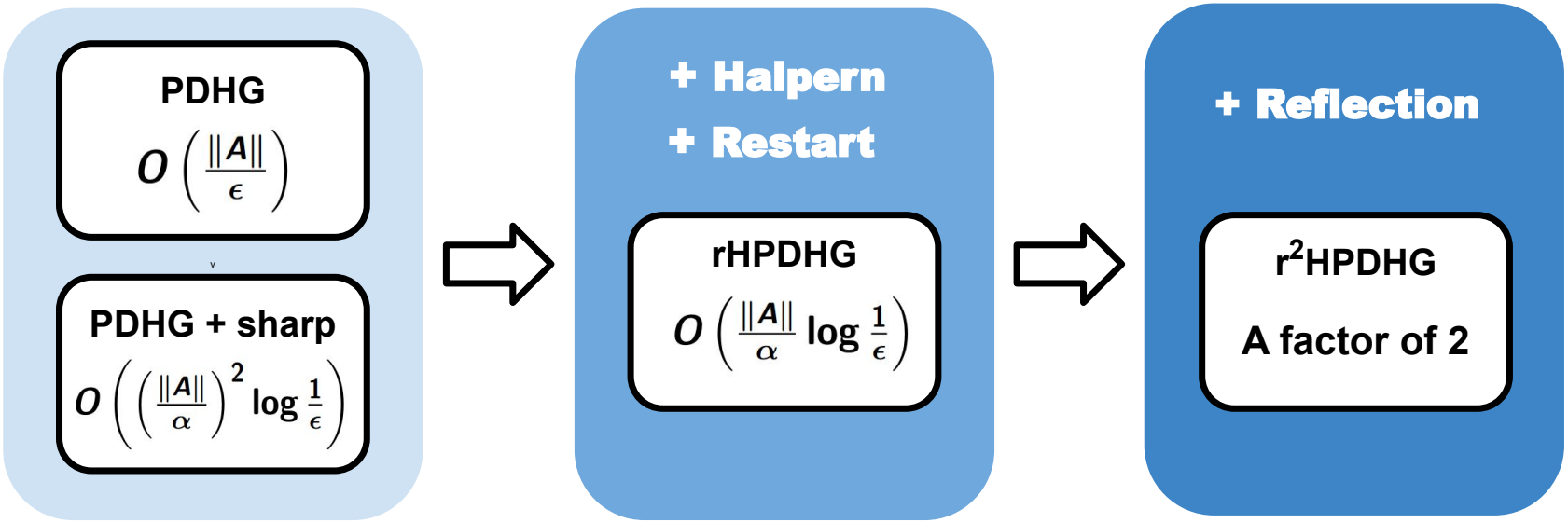
Mostly based on

- [**L**-Yang, 2024] **H Lu**, J Yang "Restarted Halpern PDHG for Linear Programming", *Under Review at Mathematics of Operations Research*, 2024.

# How to Accelerate PDHG?

cuPDLPx uses restarted reflected Halpern PDHG [**L**-Yang, 2024]

- Halpern update with restart scheme
- Reflection



# Progress Measure: KKT Error

Denote

$$K = \begin{pmatrix} A & 0 \\ -A & 0 \\ I & 0 \\ 0 & -A^\top \\ -c^\top & b^\top \end{pmatrix} \quad \text{and} \quad h = \begin{pmatrix} b \\ -b \\ 0 \\ -c \\ 0 \end{pmatrix}$$

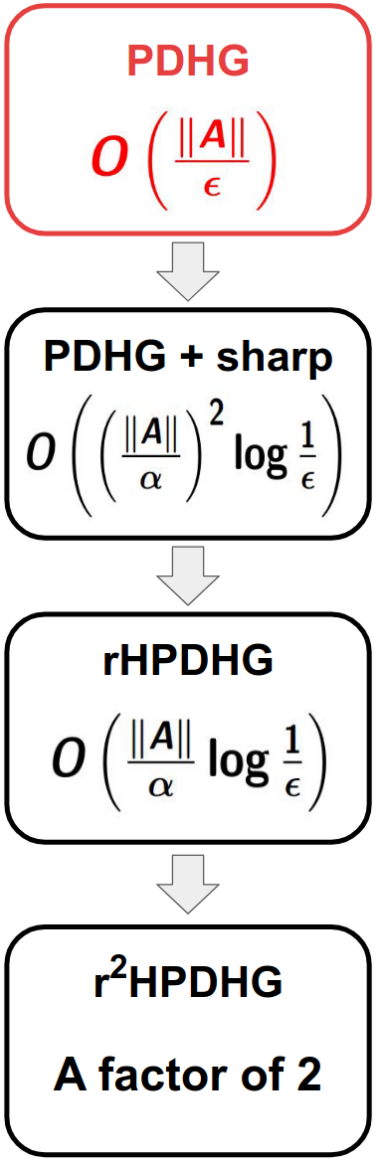
Then $Kz \geq h$ is the KKT system of LP and solutions $Z^* = \{z \mid Kz \geq h\}$

Progress measure: KKT error of standard LP at solution $z = (x, y)$

$$\mathrm{KKT}(z) = \|(h - Kz)^+\| = \left\| \begin{pmatrix} Ax - b \\ (-x)^+ \\ (A^\top y - c)^+ \\ (c^\top x - b^\top y)^+ \end{pmatrix} \right\|$$

- $(\cdot)^+$ denotes the coordinate-wise positive part of a vector

# Existing Results of PDHG for LP



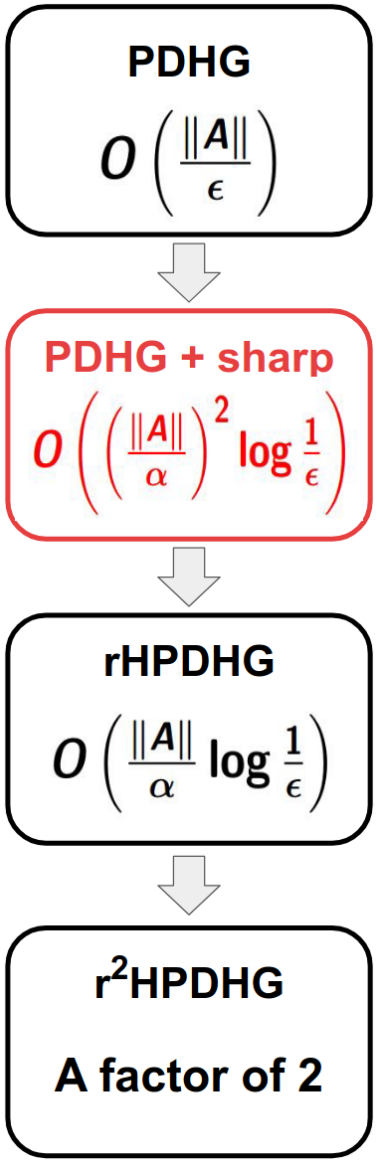

Classical convergence guarantees for PDHG on LP are mostly sublinear [Chambolle-Pock, 2011]

- PDHG finds a solution $z$ s.t. $\mathrm{KKT}(z) \leq \epsilon$ within $O(1/\epsilon)$ iterations
- Treat LPs as general convex-concave problems

Many LP applications require high-accuracy solutions

- Sublinear convergence is often insufficient
- Linear convergence rates are needed

# Linear Convergence of PDHG for LP

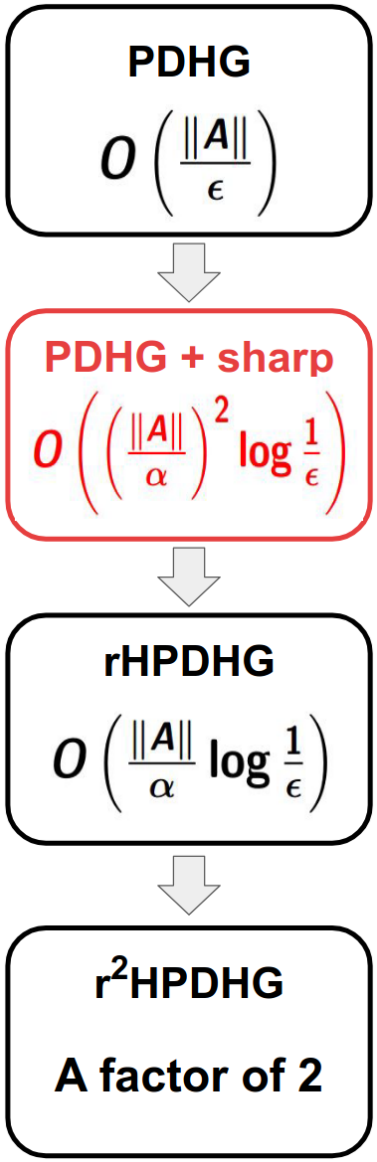



**Definition: Sharpness of the KKT System of LP**

$\alpha$ is the sharpness constant of the KKT system, if for any $z = (x, y), x \geq 0$,

$$\alpha \text{dist}(z, Z^*) \leq \|(h - Kz)^+\| .$$

# Linear Convergence of PDHG for LP




**Definition: Sharpness of the KKT System of LP**

$\alpha$ is the sharpness constant of the KKT system, if for any $z = (x, y), x \geq 0$,

$$\alpha \text{dist}(z, Z^*) \leq \|(h - Kz)^+\| .$$

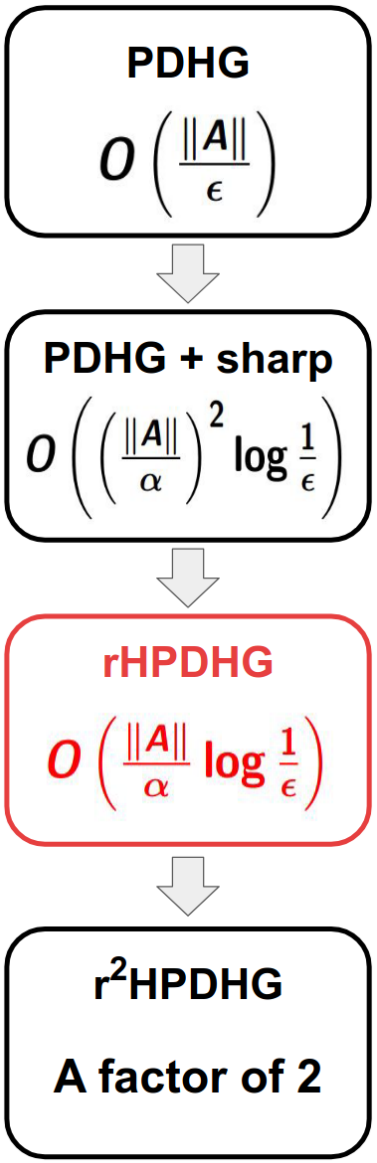
**Theorem (informal): Linear convergence of PDHG [L-Yang, 2023]**

Consider LP in primal-dual form: $\min_{x\geq 0}\max_y \; c^\top x + y^\top b - y^\top Ax$. Then PDHG finds a solution $z$ such that $\text{KKT}(z) \leq \epsilon$ within

$$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\left(\frac{1}{\epsilon}\right)\right)$$

iterations.

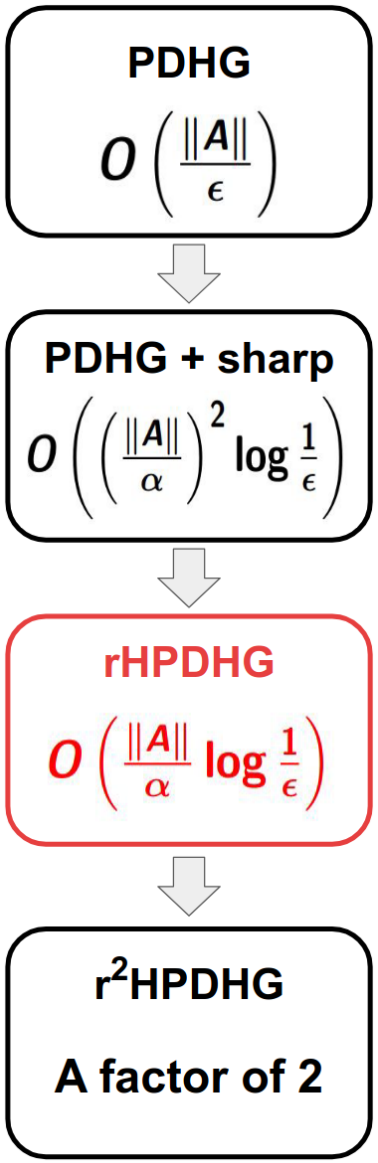
# Halpern PDHG (HPDHG)




Halpern PDHG (HPDHG)

$$z^{t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^t) + \frac{1}{t+2}z^0$$

# Halpern PDHG (HPDHG)

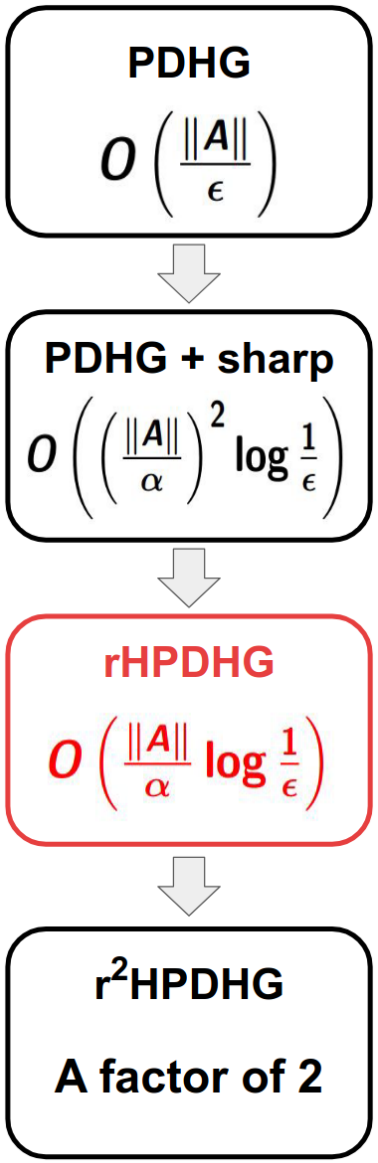



Halpern PDHG (HPDHG)

$$z^{t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^t) + \frac{1}{t+2}z^0$$

$z^0$

# Halpern PDHG (HPDHG)



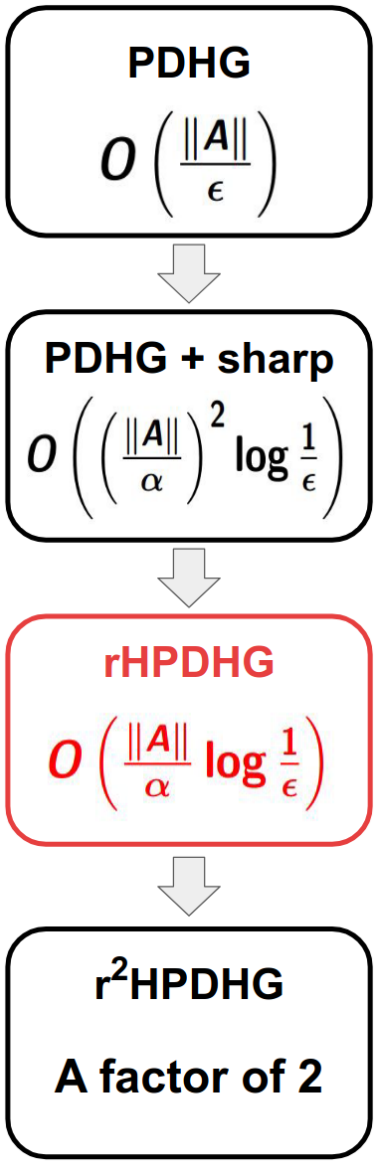

Halpern PDHG (HPDHG)

$$z^{t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^t) + \frac{1}{t+2}z^0$$

$z^0 \bullet$

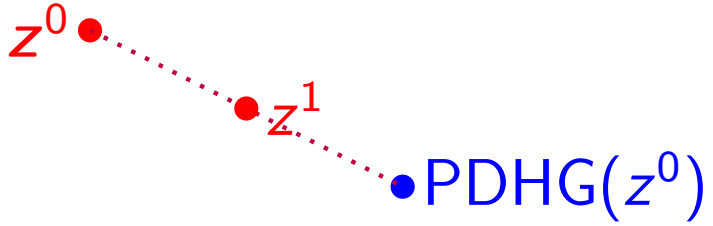
$\bullet \mathrm{PDHG}(z^0)$

# Halpern PDHG (HPDHG)

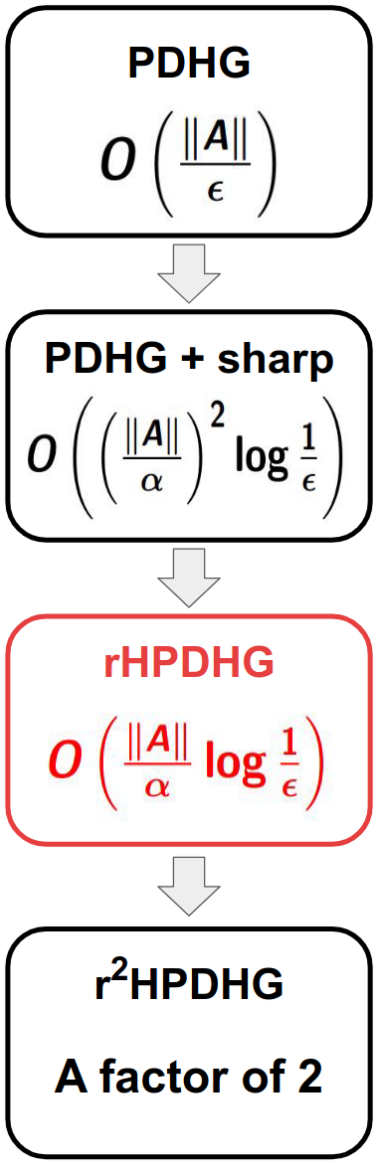



Halpern PDHG (HPDHG)

$$z^{t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^t) + \frac{1}{t+2}z^0$$



# Halpern PDHG (HPDHG)




Halpern PDHG (HPDHG)

$$z^{t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^t) + \frac{1}{t+2}z^0$$

$z^0$ • • $z^1$

• $\mathrm{PDHG}(z^1)$

# Halpern PDHG (HPDHG)

**PDHG**

$$O\left(\frac{\|A\|}{\epsilon}\right)$$

**PDHG + sharp**

$$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$$

**rHPDHG**

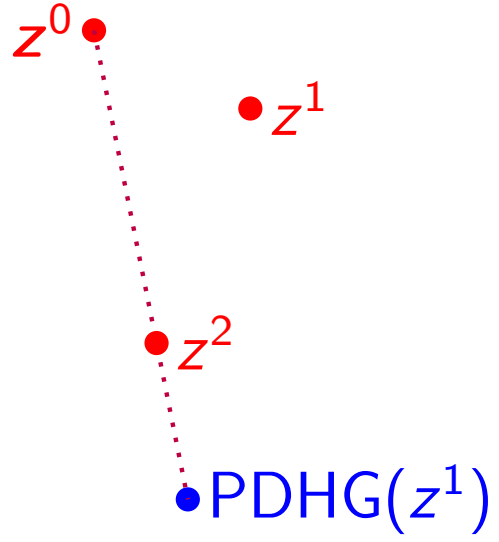
$$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$$

**r$^2$HPDHG**

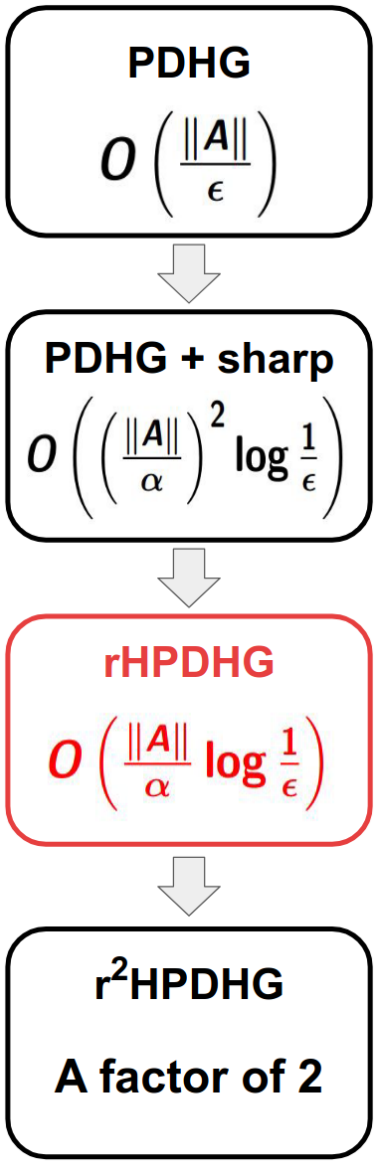
**A factor of 2**

Halpern PDHG (HPDHG)

$$z^{t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^t) + \frac{1}{t+2}z^0$$



# Halpern PDHG (HPDHG)




Halpern PDHG (HPDHG)

$$z^{t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^t) + \frac{1}{t+2}z^0$$

$z^0$ • • $z^1$ • $z^2$ • $\mathrm{PDHG}(z^2)$

# Halpern PDHG (HPDHG)

**PDHG**

$$O\left(\frac{\|A\|}{\epsilon}\right)$$

**PDHG + sharp**

$$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$$

**rHPDHG**

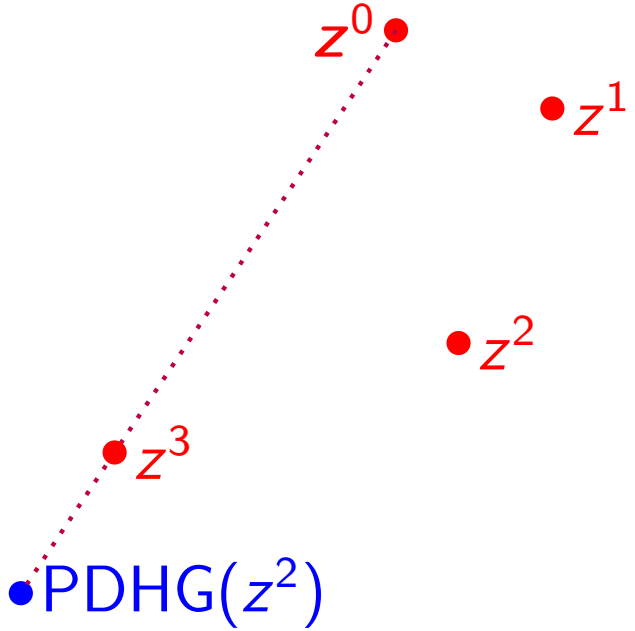
$$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$$

**r$^2$HPDHG**

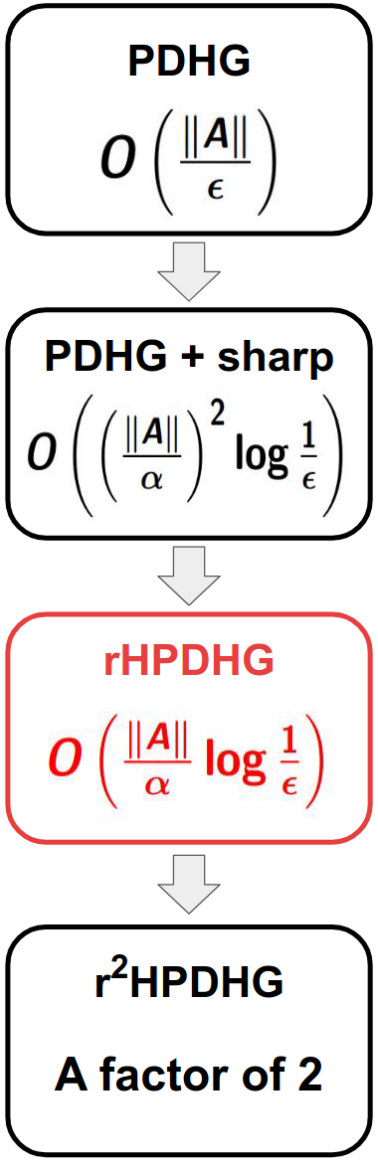
**A factor of 2**

Halpern PDHG (HPDHG)

$$z^{t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^t) + \frac{1}{t+2}z^0$$



# Restarted Halpern PDHG (rHPDHG)




Restarted Halpern PDHG (rHPDHG)

**initialize the inner loop.** inner loop counter $t \leftarrow 0$ ;
**repeat**
  $z^{n,t+1} \leftarrow \frac{t+1}{t+2}\mathrm{PDHG}(z^{n,t}) + \frac{1}{t+2}z^{n,0}$;
**until** $\|z^{n,t+1} - \mathrm{PDHG}(z^{n,t+1})\| \le \|z^{n,0} - \mathrm{PDHG}(z^{n,0})\|/2$;
**restart the outer loop.** $z^{n+1,0} \leftarrow \mathrm{PDHG}(z^{n,t+1})$;

- Let $z^{n,t}$ denote the $t$-th iterate within the $n$-th epoch
- Each epoch runs HPDHG
- Restart inner loop when fixed-point error is halved
  - Reset the initial point for next epoch
  - Reset the inner loop counter $t \leftarrow 0$

# Restarted Halpern PDHG (rHPDHG)

**PDHG**

$$O\left(\frac{\|A\|}{\epsilon}\right)$$

**PDHG + sharp**

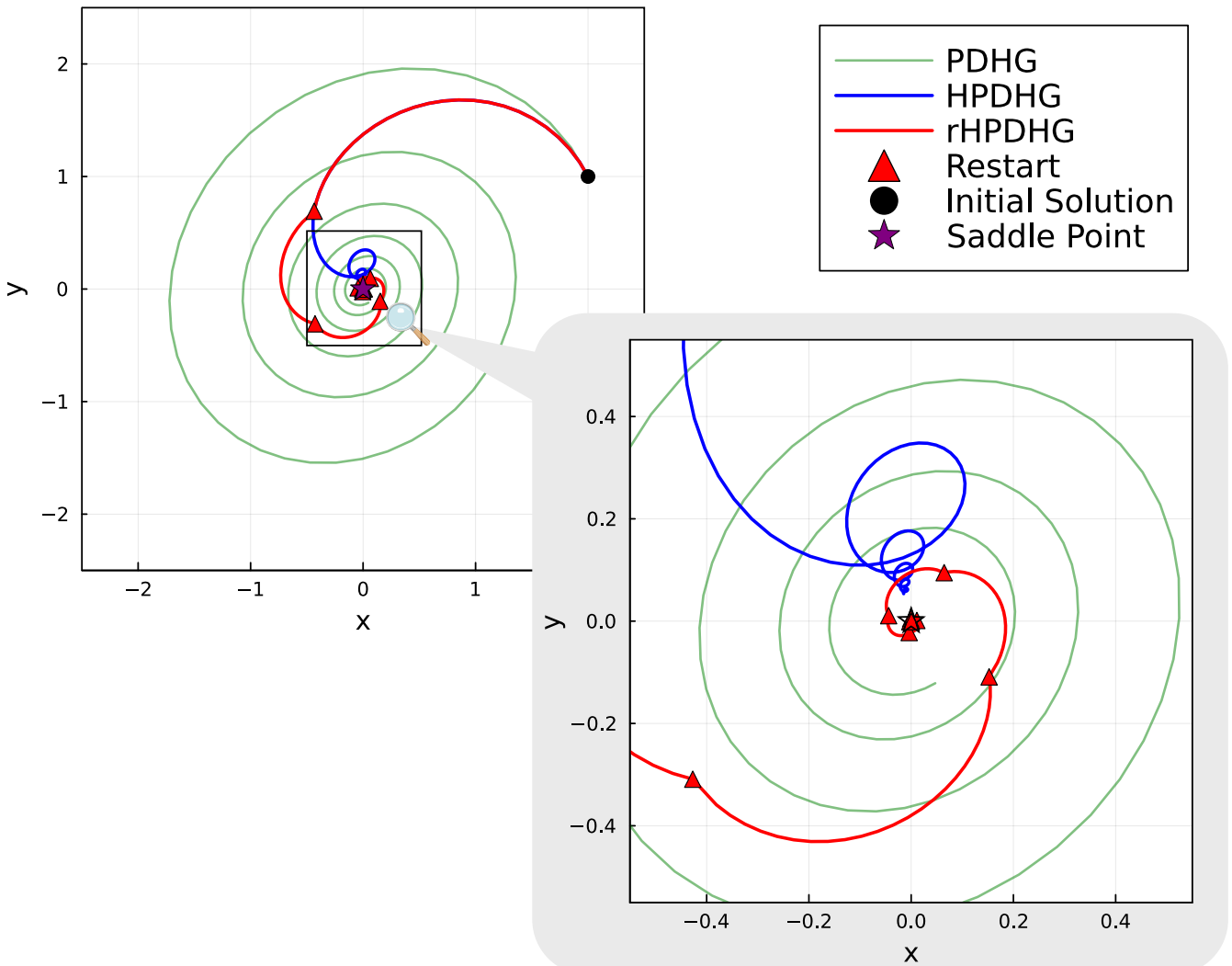
$$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$$

**rHPDHG**

$$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$$

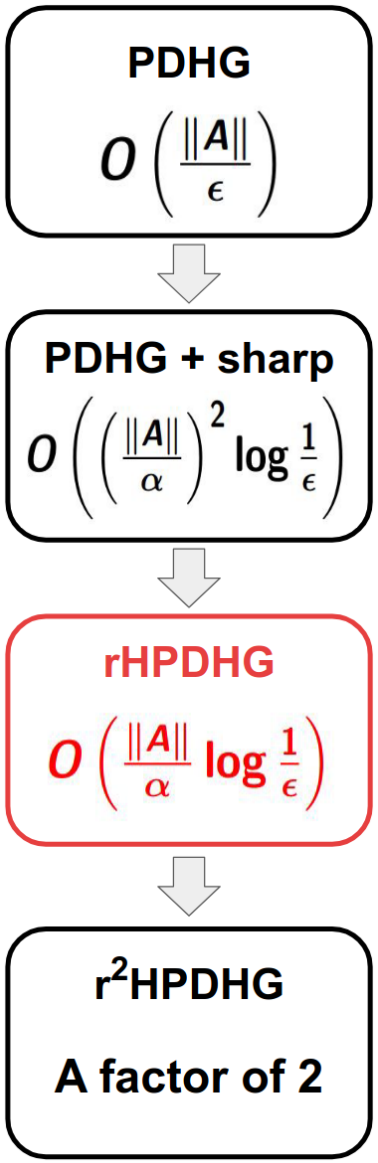
**r$^2$HPDHG**

**A factor of 2**




- Each new epoch of rHPDHG starts closer to the optimal solution

# Restarted Halpern PDHG (rHPDHG), continued




**Theorem (informal): Linear convergence of rHPDHG [L-Yang, 2024]**

Consider LP in primal-dual form: $\min_{x\geq 0}\max_y \; c^\top x + y^\top b - y^\top Ax$. Then rHPDHG finds a solution $z$ such that $\mathrm{KKT}(z)\leq\epsilon$ within

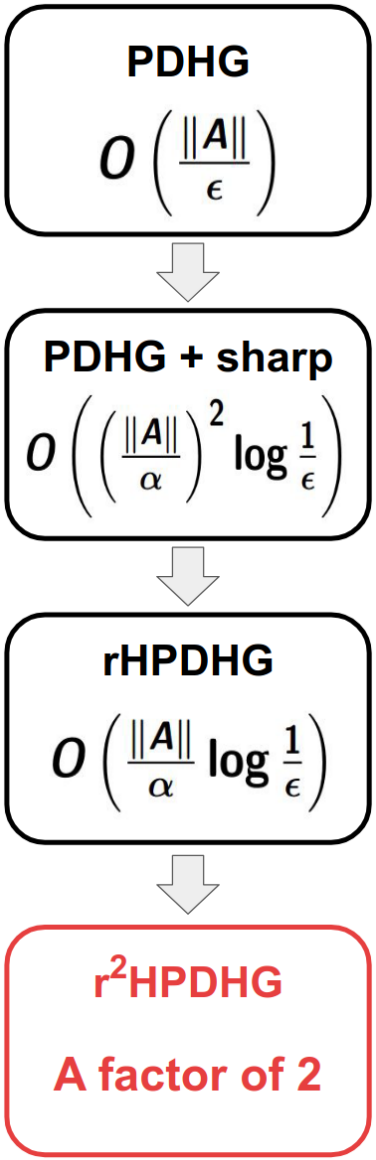
$$O\left(\frac{\|A\|}{\alpha}\log\left(\frac{1}{\epsilon}\right)\right)$$

iterations.

rHPDHG achieves the **optimal rate** for LPs among FOMs

- There exists an LP instance such that any FOM requires at least $\Omega\left(\frac{\|A\|}{\alpha}\log\left(\frac{1}{\epsilon}\right)\right)$ iterations to find a solution $z$ such that $\mathrm{KKT}(z)\leq\epsilon$ [Applegate et al., 2023]

# Restarted Reflected Halpern PDHG ($\mathrm{r}^2$HPDHG)

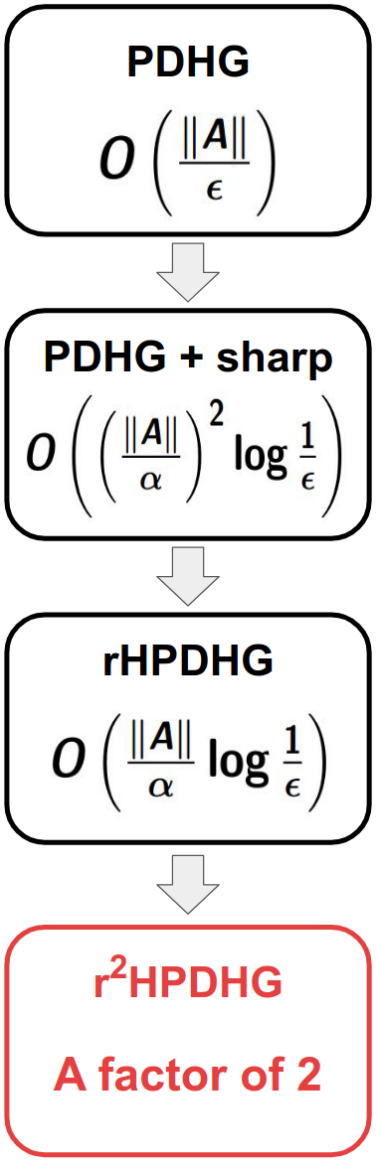



Reflected Halpern PDHG

$$z^{t+1} = \frac{t+1}{t+2}(2\mathrm{PDHG}(z^t) - z^t) + \frac{1}{t+2}z^0$$

- Take a more aggresive step
- Improve a factor of 2 theoretically

# Restarted Reflected Halpern PDHG ($\mathrm{r}^2$HPDHG)



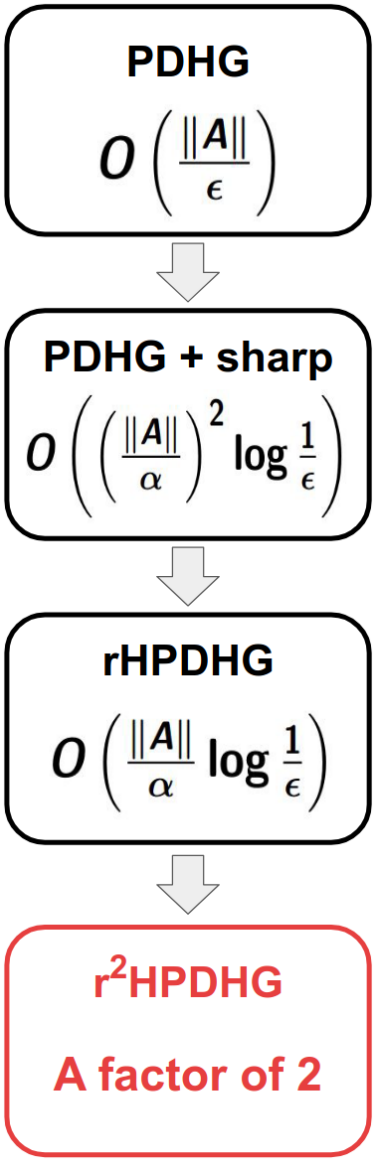

Reflected Halpern PDHG

$$z^{t+1} = \frac{t+1}{t+2}(2\mathrm{PDHG}(z^t) - z^t) + \frac{1}{t+2}z^0$$

- Take a more aggresive step
- Improve a factor of 2 theoretically

$z^0$ $z^t$

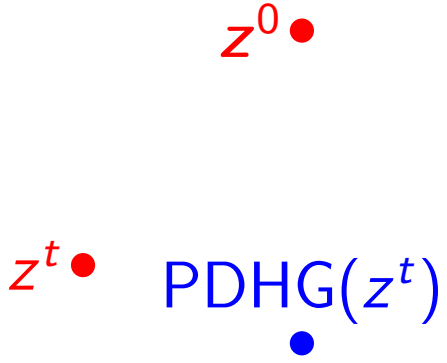
# Restarted Reflected Halpern PDHG ($\mathrm{r}^2$HPDHG)




Reflected Halpern PDHG

$$z^{t+1} = \frac{t+1}{t+2}(2\mathrm{PDHG}(z^t) - z^t) + \frac{1}{t+2}z^0$$

- Take a more aggresive step
- Improve a factor of 2 theoretically



# Restarted Reflected Halpern PDHG ($\mathrm{r}^2$HPDHG)

**PDHG**

$$O\left(\frac{\|A\|}{\epsilon}\right)$$

**PDHG + sharp**

$$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$$

**rHPDHG**

$$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$$

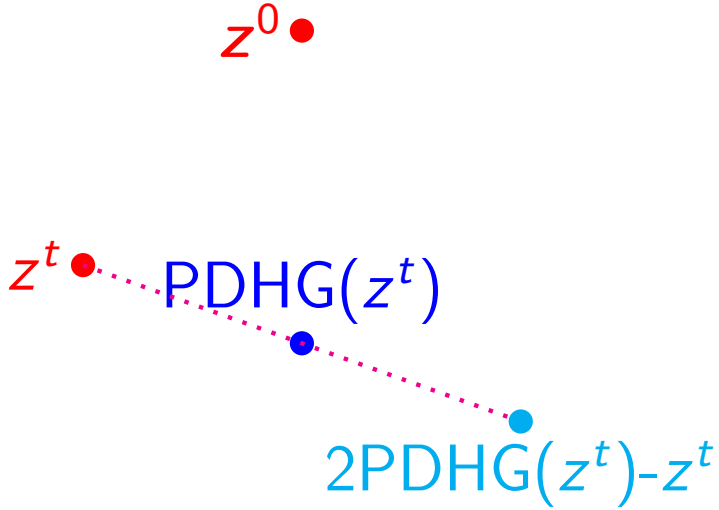
**$\mathrm{r}^2$HPDHG**

**A factor of 2**

Reflected Halpern PDHG

$$z^{t+1} = \frac{t+1}{t+2}(2\mathrm{PDHG}(z^t) - z^t) + \frac{1}{t+2}z^0$$

- Take a more aggresive step
- Improve a factor of 2 theoretically



# Restarted Reflected Halpern PDHG ($\mathrm{r}^2$HPDHG)

**PDHG**
$O\left(\frac{\|A\|}{\epsilon}\right)$

**PDHG + sharp**
$O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$

**rHPDHG**
$O\left(\frac{\|A\|}{\alpha} \log\frac{1}{\epsilon}\right)$

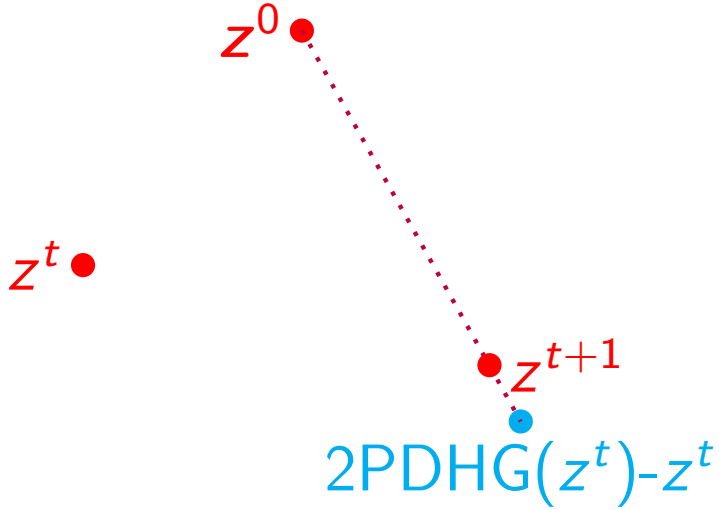
**$\mathrm{r}^2$HPDHG**

**A factor of 2**

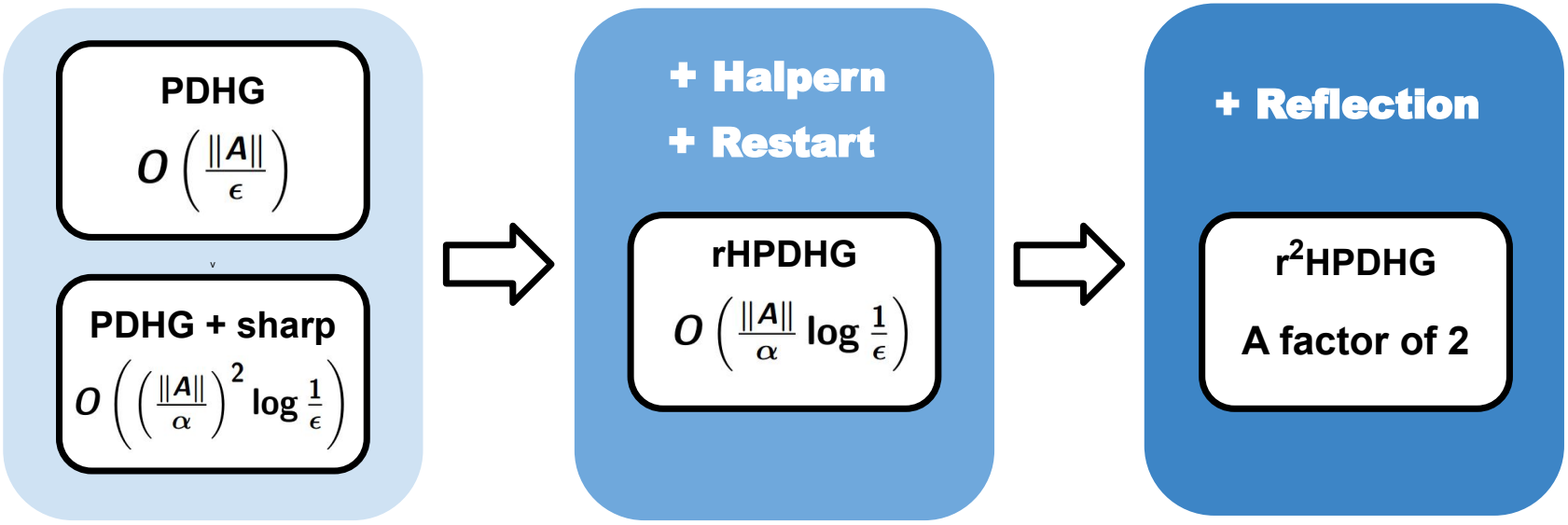
Reflected Halpern PDHG

$$z^{t+1} = \frac{t+1}{t+2}(2\mathrm{PDHG}(z^t) - z^t) + \frac{1}{t+2}z^0$$

- Take a more aggresive step
- Improve a factor of 2 theoretically



# Complexity Theory behind cuPDLPx




- $\alpha$ is global sharpness of the KKT system of LP

# Trajectory-Based Analysis

Mostly based on

- [**L**-Yang, 2025b] **H Lu**, J Yang "On the Geometry and Refined Rate of Primal-Dual Hybrid Gradient for Linear Programming", *Mathematical Programming*, 2025.
- [**L**-Yang, 2024] **H Lu**, J Yang "Restarted Halpern PDHG for Linear Programming", *Under Review at Mathematics of Operations Research*, 2024.

# Issues of Global Sharpness

We have shown that for solving LP,

- Vanilla PDHG achieves a convergence rate of $O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$
- $\mathrm{r}^2\mathrm{HPDHG}$ improves this to $O\left(\frac{\|A\|}{\alpha}\log\frac{1}{\epsilon}\right)$

# Issues of Global Sharpness

We have shown that for solving LP,

- Vanilla PDHG achieves a convergence rate of $O\left(\left(\frac{\|A\|}{\alpha}\right)^2 \log\frac{1}{\epsilon}\right)$
- $\mathrm{r}^2$HPDHG improves this to $O\left(\frac{\|A\|}{\alpha}\log\frac{1}{\epsilon}\right)$

However, the global sharpness $\alpha$ can be overly conservative

- A characterization of sharpness for linear inequality system $Kz \geq h$ [Pena et al, 2021] is

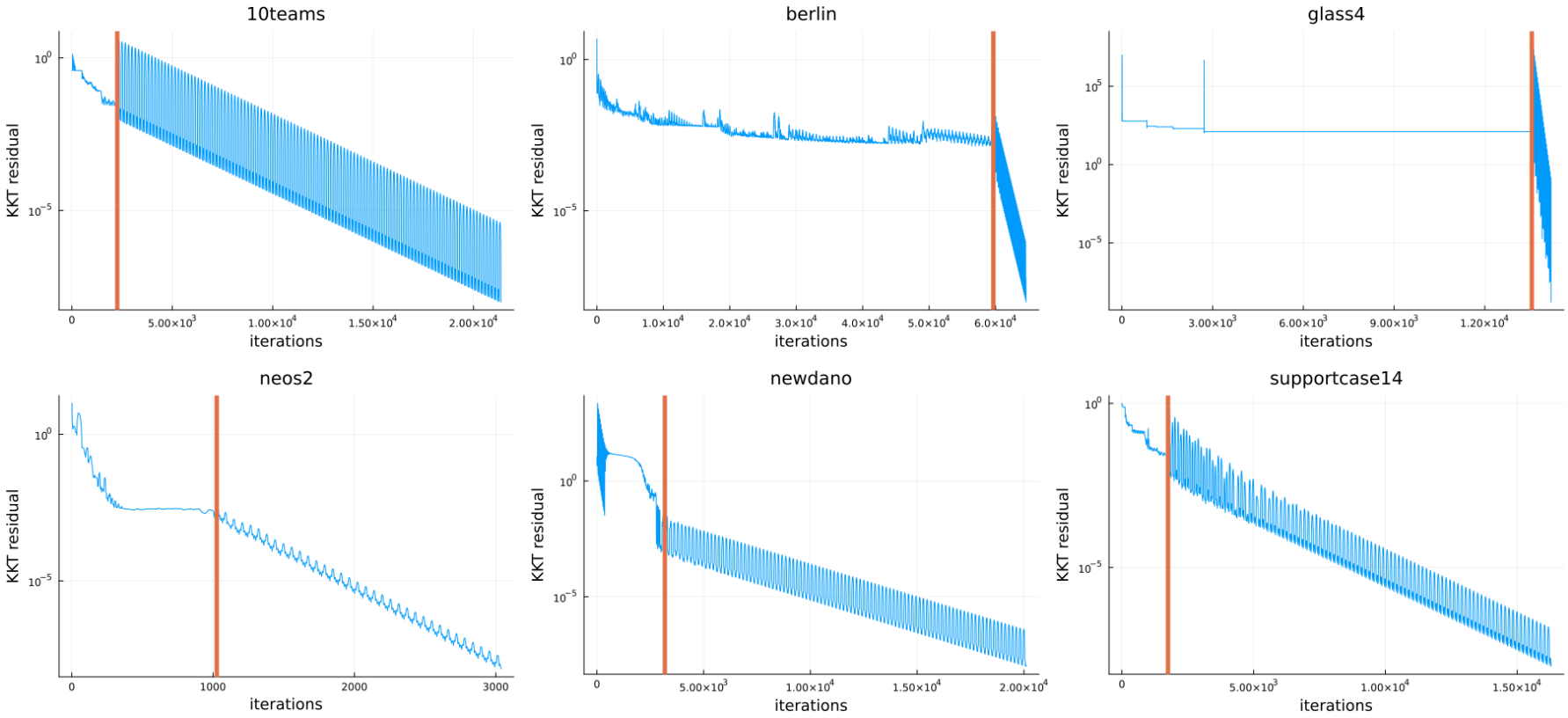
$$\alpha = \min_{J\subseteq\{1,\ldots,m\}, K_J \text{ full row rank}} \min_{v\in\mathbb{R}^J_+, \|v\|=1} \|K_J^T v\|$$

- Informally,

$$\alpha \approx \min_{J\subseteq\{1,\ldots,m\}} \sigma^+_{\min}(K_J)$$

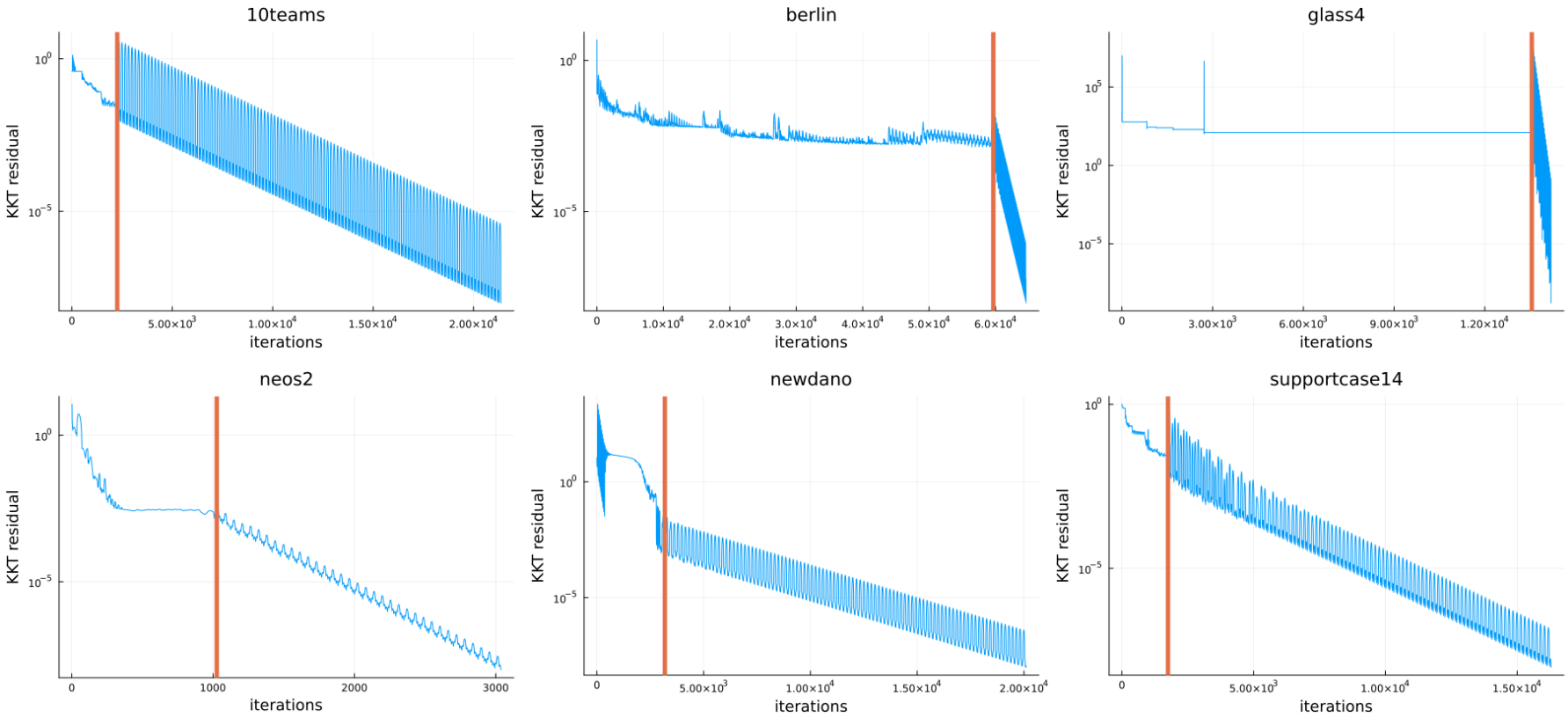
**Can we derive a more refined convergence analysis that avoids relying on global sharpness?**

# Typical Behaviors of PDHG for LP




- There are two stages of convergence
- Initial slow phase followed by fast (linear) convergence

# Two Questions

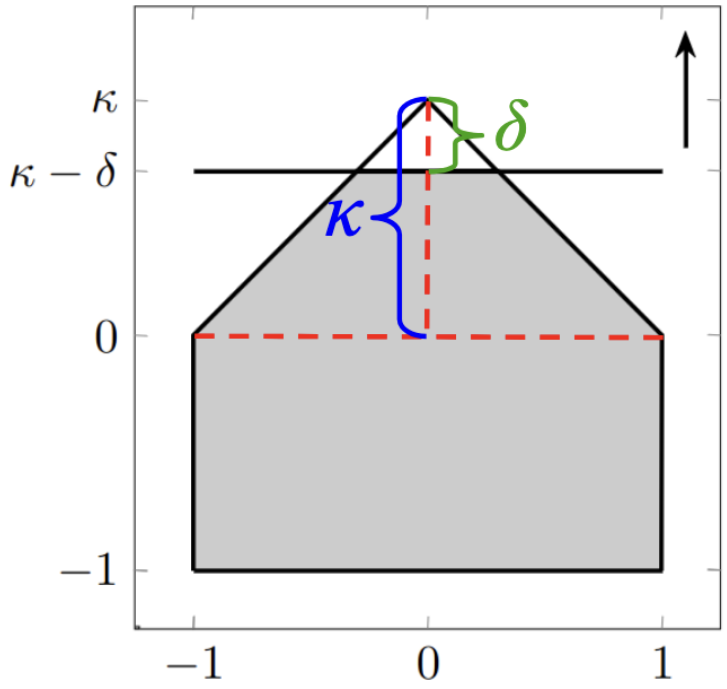



- What are the geometric driving forces of the two stages?
- Can we derive a refined convergence rate that reflects this behavior?

# A Simple yet Representative Example

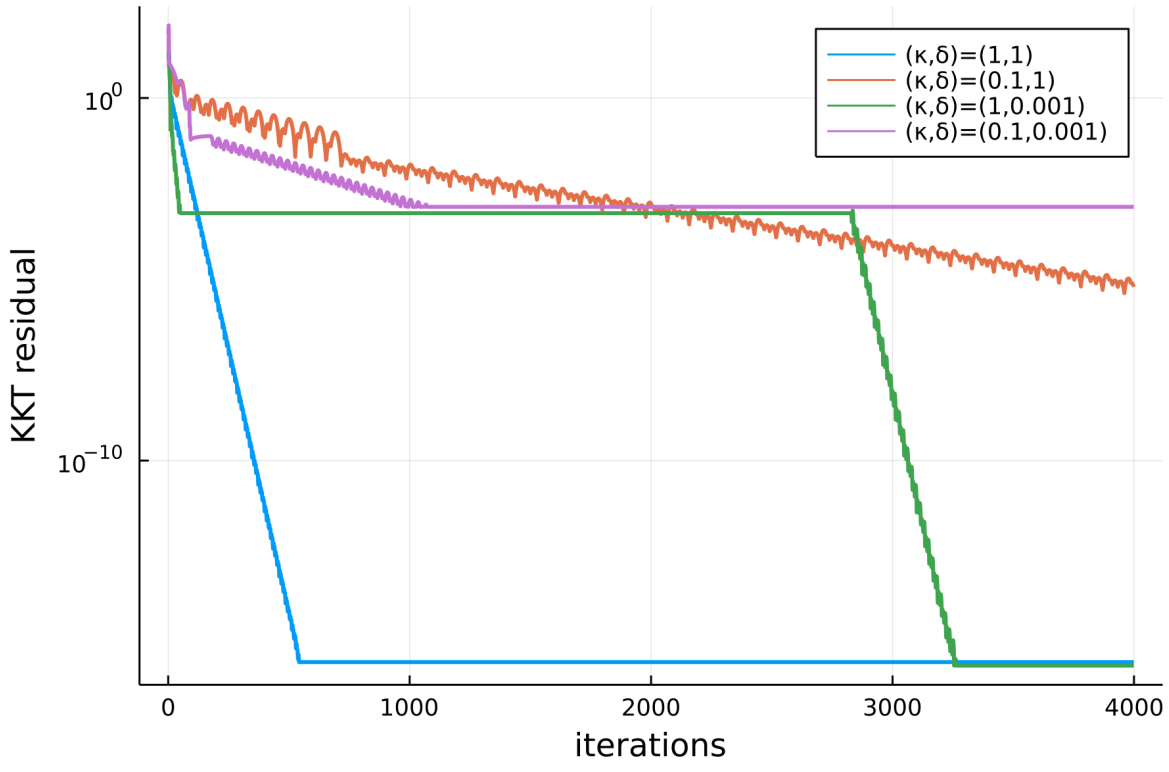
Consider a class of 2-D dual LP with parameter $(\kappa, \delta)$:

$$
\begin{aligned}
\underset{y_1, y_2}{\text{maximize}} \quad & y_2 \\
\text{subject to} \quad & y_1 \geq -1 \\
& y_1 \leq 1 \\
& y_1 + \frac{1}{\kappa} y_2 \leq 1 \\
& -y_1 + \frac{1}{\kappa} y_2 \leq 1 \\
& y_2 \leq \kappa - \delta
\end{aligned}
$$




- $\kappa$: controls the condition number of the constraint matrix
- $\delta$: controls proximity to degeneracy

# A Simple yet Representative Example

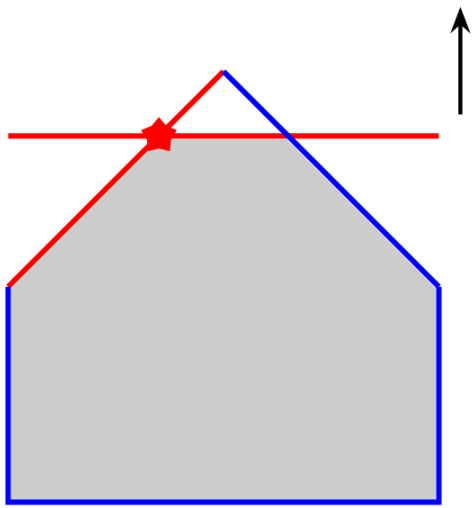



- $\delta$ small: slow first stage
- $\kappa$ small: slow second stage

# What Is Happening in the Two Stages? (Informal)

Stage 1: Slow initial convergence

- Identification of the active set $S$
- Convergence speed driven by proximity to degeneracy (i.e., $\delta$)



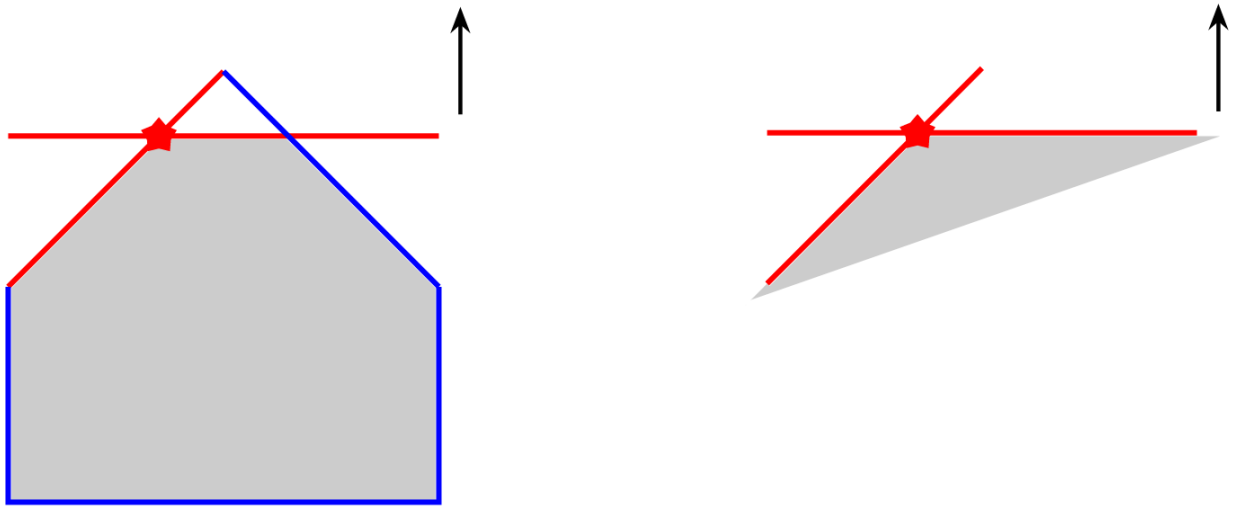
# What Is Happening in the Two Stages? (Informal)

Stage 1: Slow initial convergence

- Identification of the active set $S$
- Convergence speed driven by proximity to degeneracy (i.e., $\delta$)

Stage 2: "Fast" eventual convergence

- Once the active set is identified, dynamics become linearly convergent
- Convergence rate governed by the condition number of the reduced matrix $A_S$ (i.e., $\|A_S\|/\alpha_S$)
- Typically $\alpha_S \approx \sigma^+_{\min}(A_S) \gg \alpha$



# Refined Complexity for PDHG (Informal)

**Theorem [L-Yang, 2025b]: Two-Stage Convergence of PDHG for LP**

Consider PDHG for solving LP, then we have

- **(Finite time identification)** PDHG will identify the active set in

$$k \geq K := O\left(\left(\frac{\|A\|/\alpha_S}{\delta}\right)^2\right)$$

iterations.

# Refined Complexity for PDHG (Informal)

**Theorem [L-Yang, 2025b]: Two-Stage Convergence of PDHG for LP**

Consider PDHG for solving LP, then we have

- **(Finite time identification)** PDHG will identify the active set in

$$k \geq K := O\left(\left(\frac{\|A\|/\alpha_S}{\delta}\right)^2\right)$$

  iterations.
- **(Linear convergence after identification)** After identification, PDHG will compute a solution $z$ such that $\mathrm{KKT}(z) \leq \epsilon$ in

$$O\left(\left(\frac{||A_S||}{\alpha_S}\right)^2 \log\left(\frac{1}{\epsilon}\right)\right) .$$

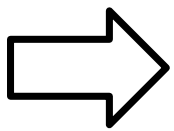
  iterations.

# Refined Accelerated Complexity (Informal)

A refined two-stage characterization:

**PDHG**

$$O\left(\left(\frac{\|A\|/\alpha_S}{\delta}\right)^2 + \left(\frac{\|A_S\|}{\alpha_S}\right)^2 \log\frac{1}{\epsilon}\right)$$



**rHPDHG**

$$O\left(\frac{\|A\|/\alpha_S}{\delta} + \frac{\|A_S\|}{\alpha_S} \log\frac{1}{\epsilon}\right)$$

- Similar acceleration over PDHG achieved by Halpern and restart [**L**-Yang, 2024]

# Extensions

- **LP infeasibility detection: HoPDLP**
  [**L**–Tang, 2026]
- **Convex QP: PDQP**
  [**L**–Yang, 2025c]
- **Nonconvex QP: PDNQP**
  [Chen–**L**, 2026]
- **Semidefinite programming**
  [Ding–**L**–Yang, 2025]

# HoPDLP: Optimal Solutions and Infeasibility Certificates

**Goal:** a GPU LP solver that returns an optimal primal-dual pair or a verifiable infeasibility certificate.

$$\min_x c^\top x \quad \text{s.t.} \quad \ell_c \le Ax \le u_c, \quad \ell_v \le x \le u_v$$

**Homogeneous self-dual embedding** [**L**–Tang, manuscript]:

- Generalized PDHG with restarted reflected Halpern iteration.
- Two-sided bounds enter through box cones whose projections reduce to one-dimensional root finding.
- Sparse matrix-vector products and projections keep iterations factorization-free and suitable for GPUs.

At a conclusive embedding solution:

$\tau > 0$ : recover an optimal primal-dual pair,

$\kappa > 0$ : recover a primal or dual infeasibility certificate.

# HoPDLP: Infeasible and Unbounded LP Benchmarks

**MIPLIB 2017 Infeas:** 229 infeasible and 105 unbounded LPs.

| | No presolve | | With PSLP | |
|---|---|---|---|---|
| **Solver** | **Detected** | **Time (s)** | **Detected** | **Time (s)** |
| Gurobi HSD barrier | 257 | 42.84 | 288 | 15.59 |
| OR-Tools PDLP | 230 | 371.66 | 290 | 87.31 |
| HoPDLP ($10^{-8}$) | 326 | 6.17 | 334 | 2.10 |

Table: Detected counts out of 334; time is SGM10.

HoPDLP: one H100 80 GB GPU; baseline stopping criteria differ.
$10^{-8}$ is HoPDLP's optimality tolerance; certificate tolerances stay fixed.

**HoPDLP also detects all 29 infeasible Netlib instances.**

# PDQP: GPU Methods for Convex QP

Convex quadratic programming

$$\min_x \ \tfrac{1}{2} x^\top Q x + c^\top x \quad \text{s.t.} \quad Ax \le b$$

$Q \succeq 0$. **Goal:** solve large convex QPs on GPUs.

**PDQP** [**L**–Yang, 2025c]:

- Restarted accelerated PDHG.
- Combine **restart** for LP geometry with **momentum** for the quadratic term.
- Linear convergence under a QP error bound.

Factorization-free GPU iterations: SpMV and vector operations.

# PDQP: Numerical Results

**Benchmarks:** 63 synthetic convex QPs in seven classes; 21 instances per scale.

Small / medium / large: approximately 0.3M / 3M / 30M nonzeros.

| | Small | | Medium | | Large | |
|---|---|---|---|---|---|---|
| **Solver** | **Solved** | **Time** | **Solved** | **Time** | **Solved** | **Time** |
| PDQP (GPU) | 21 | 2.08 | 21 | 3.40 | 21 | 12.17 |
| SCS (GPU) | 21 | 5.69 | 21 | 24.38 | 18 | 196.75 |
| SCS (CPU-indirect) | 21 | 23.90 | 20 | 267.27 | 12 | 1593.95 |
| SCS (CPU-direct) | 21 | 3.09 | 21 | 30.25 | 20 | 395.85 |
| OSQP (CPU) | 21 | 3.31 | 21 | 30.70 | 21 | 375.24 |
| Gurobi barrier | 21 | 1.17 | 21 | 10.48 | 18 | 131.85 |

Table: Time: SGM10 in seconds; tolerance $10^{-6}$.

**PDQP solves all 63 instances; large-instance SGM10 is 12.17 s.**

# PDNQP: GPU Methods for Nonconvex QP

Nonconvex quadratic programming

$$\begin{aligned} \min_x \quad & \tfrac{1}{2} x^\top Q x + c^\top x \\ \text{s.t.} \quad & A_E x = b_E, \quad A_I x \geq b_I, \quad \ell \leq x \leq u \end{aligned}$$

$Q$ may be indefinite. **Target:** a stationary (KKT) point.

**PDNQP** [Chen–**L**, manuscript]:

- Proximal augmented-Lagrangian outer iterations produce strongly convex QP subproblems.
- Retain residual variables to avoid penalty-dependent normal matrices such as $A^\top \Sigma A$.
- Solve inner QPs with restarted accelerated PDHG on a fixed sparse operator, using warm starts and adaptive inner accuracy.

GPU kernels use SpMV, projections, and vector operations.

# PDNQP: Numerical Results

**Benchmarks:** 107 nonconvex CUTEst QPs and 30 synthetic QPs with approximately one million variables each.

| | **CUTEst** $10^{-4}$ | | **Million** $10^{-4}$ | |
|---|---|---|---|---|
| **Solver** | **Success** | **Time** | **Success** | **Time** |
| PDNQP | 107 | 4.4 | 27 | 76.0 |
| QPALM | 101 | 23.0 | 0 | 3600 |
| Ipopt | 95 | 26.3 | 0 | 3600 |
| MadNLPGPU | 101 | 11.3 | 0 | 3600 |
| Knitro | 89 | 26.4 | 0 | 3600 |

Table: Time: SGM10 in seconds; unsuccessful runs charged 3,600 s.

CUTEst successes include accepted KKT points and infeasibility classifications. PDNQP and MadNLPGPU use an H100 80 GB GPU; others use CPUs.

**Verified KKT points on 27/30 million-variable QPs.**

# A GPU-Based SDP Solver

Semidefinite Programming (SDP)

$$\begin{aligned} \text{minimize} \quad & \langle C, X \rangle \\ \text{subject to} \quad & \mathcal{A}(X) = b, \ X \succeq 0 \ , \end{aligned}$$

where $X$ is a positive semidefinite matrix variable

We are building up a new GPU-based SDP solver

- Still exploring different FOMs for SDP
- Preliminary numerical results of our developing solver are suprisingly promising [Ding-**L**-Yang, 2025]

# SDP Relaxation of Large MaxCut

| instance | dimension | time | instance | dimension | time |
|---|---|---|---|---|---|
| NACA0015 | 1039183 | 11.485 | hugetric-00000 | 5824554 | 77.006 |
| delaunay_n20 | 1048576 | 7.333 | hugetric-00010 | 6592765 | 79.448 |
| kron_g500-logn20 | 1048576 | 131.911 | italy.osm | 6686493 | 40.709 |
| rgg_n_2_20_s0 | 1048576 | 9.888 | adaptive | 6815744 | 96.847 |
| belgium.osm | 1441295 | 8.283 | hugetric-00020 | 7122792 | 83.734 |
| delaunay_n21 | 2097152 | 15.262 | great-britain.osm | 7733822 | 61.270 |
| kron_g500-logn21 | 2097152 | 335.199 | delaunay_n23 | 8388608 | 67.985 |
| rgg_n_2_21_s0 | 2097152 | 21.679 | rgg_n_2_23_s0 | 8388608 | 164.510 |
| packing-500x100x100-b050 | 2145852 | 26.731 | germany.osm | 11548845 | 94.783 |
| netherlands.osm | 2216688 | 12.081 | asia.osm | 11950757 | 108.128 |
| M6 | 3501776 | 46.567 | hugetrace-00010 | 12057441 | 186.480 |
| 333SP | 3712815 | 40.248 | road_central | 14081816 | 174.989 |
| AS365 | 3799275 | 49.985 | hugetrace-00020 | 16002413 | 314.768 |
| venturiLevel3 | 4026819 | 41.529 | delaunay_n24 | 16777216 | 189.024 |
| NLR | 4163763 | 51.739 | rgg_n_2_24_s0 | 16777216 | 412.733 |
| delaunay_n22 | 4194304 | 33.287 | hugebubbles-00000 | 18318143 | 387.761 |
| rgg_n_2_22_s0 | 4194304 | 47.507 | hugebubbles-00010 | 19458087 | 280.580 |
| hugetrace-00000 | 4588484 | 50.669 | hugebubbles-00020 | 21198119 | 308.113 |
| channel-500x100x100-b050 | 4802000 | 80.858 | road_usa | 23947347 | 308.473 |

Table: Solve time of our developing solver in seconds with tolerance $10^{-5}$

- Instances constructed from graphs in the DIMACS10 collection

# SDP Formulation of Large Matrix Completion

| $r = 3$ | | | $r = 5$ | | |
|---|---|---|---|---|---|
| $n$ | $m$ | time | $n$ | $m$ | time |
| 10000 | 828659 | 0.379 | 10000 | 2300917 | 0.504 |
| 20000 | 1782453 | 0.726 | 20000 | 4952616 | 0.969 |
| 50000 | 4868248 | 1.409 | 50000 | 13522024 | 3.025 |
| 100000 | 10361604 | 2.944 | 100000 | 28777073 | 6.762 |
| 200000 | 21961921 | 5.511 | 200000 | 61013229 | 14.786 |
| 350000 | 40223331 | 12.759 | 350000 | 111700922 | 28.282 |

Table: Solve time of our developing solver in seconds with tolerance $10^{-5}$

- Synthetic data following the same scheme as [Monteiro et al., 2024]
  - $n$ is problem dimension
  - $m$ is number of constraints
  - $r$ is the true rank

# Takeaways

GPU can be effective for large-scale mathematical programming

# Takeaways

GPU can be effective for large-scale mathematical programming

- cuPDLP/cuPDLPx demonstrate the potential of GPUs for solving large linear programs
  - Providing a GPU-native option for LP solving

# Takeaways

GPU can be effective for large-scale mathematical programming

- cuPDLP/cuPDLPx demonstrate the potential of GPUs for solving large linear programs
  - Providing a GPU-native option for LP solving
- The underlying methodology extends beyond linear programming
  - Examples include convex QP and SDP

# Takeaways

GPU can be effective for large-scale mathematical programming

- cuPDLP/cuPDLPx demonstrate the potential of GPUs for solving large linear programs
  - Providing a GPU-native option for LP solving
- The underlying methodology extends beyond linear programming
  - Examples include convex QP and SDP

Fully exploiting GPUs may require rethinking optimization algorithms

# Takeaways

GPU can be effective for large-scale mathematical programming

- cuPDLP/cuPDLPx demonstrate the potential of GPUs for solving large linear programs
  - Providing a GPU-native option for LP solving
- The underlying methodology extends beyond linear programming
  - Examples include convex QP and SDP

Fully exploiting GPUs may require rethinking optimization algorithms

- Algorithms tailored to GPU architectures $\Rightarrow$ faster iteration speed

# Takeaways

GPU can be effective for large-scale mathematical programming

- cuPDLP/cuPDLPx demonstrate the potential of GPUs for solving large linear programs
  - Providing a GPU-native option for LP solving
- The underlying methodology extends beyond linear programming
  - Examples include convex QP and SDP

Fully exploiting GPUs may require rethinking optimization algorithms

- Algorithms tailored to GPU architectures $\Rightarrow$ faster iteration speed
- Algorithms aligned with problem structures $\Rightarrow$ faster convergence

# Takeaways

GPU can be effective for large-scale mathematical programming

- cuPDLP/cuPDLPx demonstrate the potential of GPUs for solving large linear programs
  - Providing a GPU-native option for LP solving
- The underlying methodology extends beyond linear programming
  - Examples include convex QP and SDP

Fully exploiting GPUs may require rethinking optimization algorithms

- Algorithms tailored to GPU architectures $\Rightarrow$ faster iteration speed
- Algorithms aligned with problem structures $\Rightarrow$ faster convergence
- Careful, GPU-aware implementations $\Rightarrow$ high utilization and scalability