Mathematical Programming in the Era of AI

HORIZONS 2026, CEI VinUni

Yinyu Ye (SJTU and Stanford)

2026/07/2

Mathematical Programming (MP)/Optimization

$$\max \quad \sum r_t x_t$$

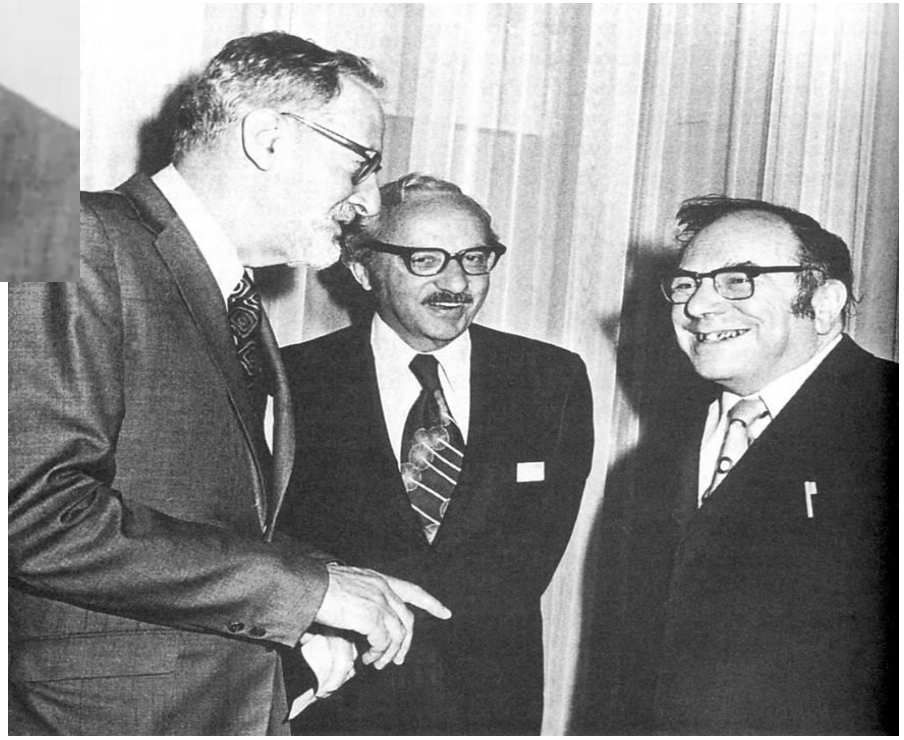
$$\text{s.t.} \quad \sum_t \boldsymbol{a}_t x_t \leq \boldsymbol{b}\,,$$

$$0 \leq x_t \leq 1 \quad \forall\ t = 1,\ldots,T$$

Linear Programming (**LP**)
x_t: decision variable
Data: number r_t, and vectors $\mathbf{a}_t$, $\mathbf{b}$

Von Neumann

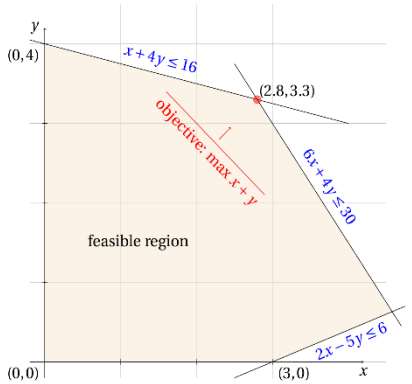
Koopman
Dantzig
Kantorovich



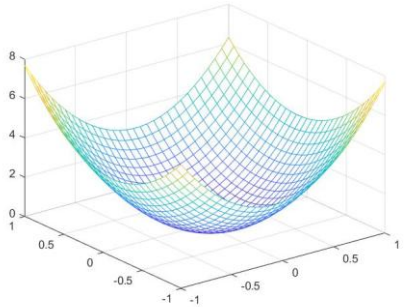
Mathematical Optimization Problems/Algorithms

Optimization models/problems are commonly classified under the theory and applications.

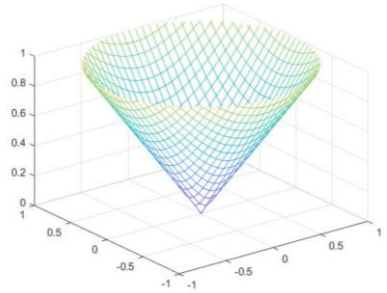
$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{Minimize}} \quad & c^\top x \\ \text{Subject to} \quad & Gx \geq h \\ & Ax = b \\ & l \leq x \leq u \end{aligned}$$



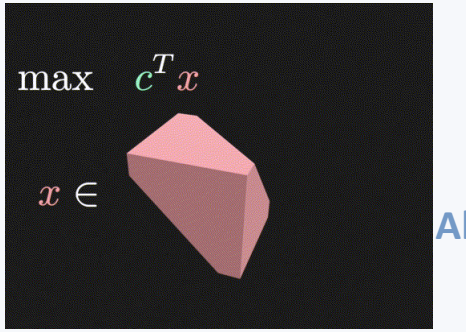
Linear Programming (LP)



Convex Quadratic Programming (QP)

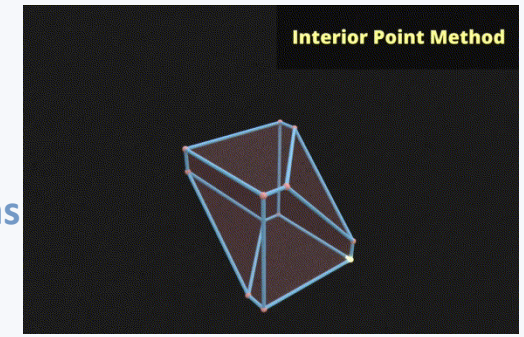


Second-Order Cone Programming (SOCP)

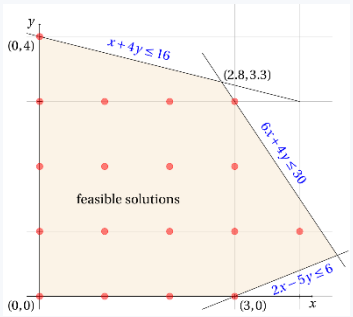


Simplex Method

Algorithms

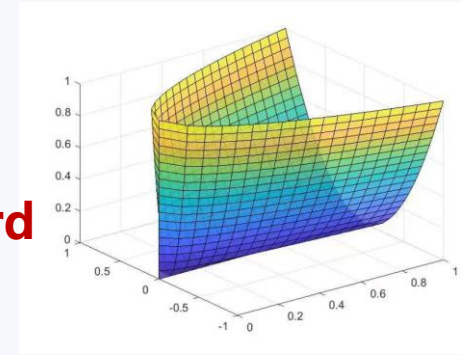


Interior Point Method



Integer Programming (IP)

NP-hard



Nonlinear Programming (NLP)

Mathematical Optimization Solver Developments

Milestones in Mathematical Programming Solvers

- **1939** Soviet Nobel Laureate Kantorovich invented Linear Programming (LP)
- **1979** University of Chicago — Lingo (the first LP solver)
- 1983 University of Edinburgh — XPRESS (Europe's first solver)
- 1987 French CPLEX (later acquired by IBM)
- 2000 Open-source CLP and CBC (well-known open-source solvers)
- 2005 Open-source SCIP (integer programming solver)
- 2008 GUROBI (the world's strongest solver at that time)
- **2017 Our team released the open-source solver LEAVES**
- 2018 Chinese Academy of Sciences (CAS) released CMIP
- **2019 Our team released the professional solver COPT;** subsequently, Alibaba and Huawei established their own solver teams
- 2021–Present Google, Oracle, and Microsoft began investing; intelligent computing is becoming an imperative of the era

Overview of Well-Known Mathematical Programming Solvers

GUROBI
CPLEX
XPRESS

One of the "Big Three" solver giants from the US and UK, with 30+ years of cumulative R&D and >95% market share. Fully equipped with linear, integer, and nonlinear modules

SAS — World's largest statistical software

CVX — World-renowned solver modeling platform

IPOPT — Famous open-source nonlinear programming software

Coin-OR — World's largest open-source solver organization

Baron — World's best nonlinear programming solver

NEOS — World's largest free service platform

SCIP — Open-source integer programming (ZIB)

CBC — Coin-OR's MILP solver

SOPLEX

MOSEK — Formerly world's #1 conic programming software

GLPK

USA | Germany | Denmark | Russia | China

XOPT — CUHK-Shenzhen

OPTV — Huawei

MDOPT — Alibaba

COPT — Cardinal Optimization

CMIP — CAS

LEAVES — SUFE

Optimization Solver Rank: LP Benchmark 2026 (https://plato.asu.edu/bench.html by Hans Mittelmann)

18 Feb 2026

LPfeas Benchmark (find PD feasible point) + ADDENDUM

++++++++++++++++++++ cuOpt runs in concurrent mode; the tolerance level for all codes used is 1e-6; COPTG has barrier accuracy (1e-8)

unscaled	25.3	83.3	272	431	421	146	18.0	26.3	15.1	39.3
scaled	1.67	5.50	18.0	28.5	27.8	9.63	1.19	1.74	1	2.60
solved	65	59	56	49	50	59	61	58	64	54
65 probs	COPT	MOSEK	HiGHS	KNTRO	PDLP	XOPT	CUOPT	CUPDX	COPTG	HPRLP
L1_sixm	2	3	115	t	88	8	3	1	6	1
Linf_520c	3	6	37	4858	233	15	3	2	2	2
a2864	1	1	703	117	10	1	1	1	1	2
bdry2	6	19	t	11	t	146	10	t	6	876
cont1	1	1	7	1	256	6	5	21	1	15
cont11	1	1	21	2	1988	23	48	60	1	36
datt256	1	2	9	6	2	5	1	1	1	1
dlr1	28	118	238	1663	t	279	15	19	15	t
ex10	1	1	49	7	1	1	1	1	1	1
fhnw-bin1	19	47	902	435	3827	86	86	28	14	19
fome13	1	1	19	26	10	1	1	1	1	1
graph40-40	1	1	31	5	2	1	1	1	1	1
irish-e	9	3	17	5	t	12	1	24	5	t
neos	5	8	155	2564	257	16	13	10	2	9
neos3	1	1	58	31	3	1	1	1	1	1
neos3025225	8	6	117	35	94	45	2	1	6	2
neos5052403	4	4	18	14	13	9	2	1	2	4
neos5251015	2	5	560	9	3	14	1	1	2	1
ns1687037	3	f	35	7	t	53	40	t	2	t
ns1688926	2	1	972	5408	t	47	12	566	4	t
nug08-3rd	1	1	60	318	1	1	1	1	1	1
pds-100	13	19	80	567	37	34	1	1	5	1
psched3-3	5	17	78	75	5165	15	9	19	2	t
qap15	1	1	21	48	3	1	1	1	1	1

8 Feb 2026

LPopt Benchmark (find optimal basic solution)

unscaled	1053	38.5	288	830	2296	3996	255
scaled	27.4	1	7.49	21.6	59.7	104	6.63
solved	40	65	52	48	33	31	52
probs	CLP	COPT	MOSEK	HiGHS	GLOP	SPLX	XOPT
L1_sixm	269	2	3	114	f	t	10
Linf_520c	36	4	63	t	239	t	21
a2864	1645	55	1307	4047	f	t	2001
bdry2	t	59	f	t	t	t	215
cont1	184	2	1415	421	259	276	5
cont11	963	5	f	528	1452	4250	35
datt256	106	3	174	551	471	t	99
dlr1	t	41	148	246	f	t	273
ex10	32	1	14	49	479	296	2
fhnw-bin1	t	26	870	2912	t	t	97
fome13	40	1	8	19	141	206	2
graph40-40	556	13	47	814	f	10923	28
irish-e	297	5	5	t	t	572	14
neos	23	5	9	196	45	47	19
neos3	18	1	8	361	248	3898	1
neos3025225	663	11	8	102	96	500	68
neos5052403	188	4	4	24	370	385	16
neos5251015	249	3	20	574	99	3860	36
ns1687037	233	3	9075	313	553	2824	53
ns1688926	12	2	1	t	19	82	28
nug08-3rd	97	3	65	345	f	845	17

QP Benchmark 2026

Convex Continuous QPLIB Benchmark (also on GPUs)

mean	2.38	1.95	12.2	1	61.8
solved	41	41	35	42	25
prob#	MOSEK	KNITRO	IPOPT	COPT	Mnotaur
2456	1	1	1	1	20
2468	3	1	2	1	295

Convex Discrete QPLIB Benchmark

mean	7.24	12.1	3.60	51.9	20.1	60.7	1.02	1
solved	20	15	22	7	14	8	25	24
prob#	MOSEK	KNITRO	BARON	BONMIN	SCIP	MNTAUR	SHOT	COPT
3547	286	204	137	t	t	t	2	111
3678	+	+	+	+	+	+	4280	+

Continuous Non-Convex QPLIB Benchmark

unscaled	1454	889	2721	4559	1331
scaled	1.64	1	3.06	5.13	1.50
solved	35	31	22	15	27
prob#	ANTIGONE	BARON	MINOTAUR	SCIP	COPT

Discrete Non-Convex QPLIB Benchmark (non-binary)

unscaled	657	83.0	2062	173
scaled	7.92	1	24.8	2.09
solved	66	99	40	88
prob#	BARON	SHOT	SCIP	COPT

Nonconvex QUBO-QPLIB Benchmark

mean	1.81	6.69	2.89	1	6.60	1.84	2.18
solved	13	8	12	15	9	12	13
prob#	BARON	SCIP	MCSPARSE	QUBOWL	BIQBIN	SHOT	COPT
3506	80	t	407	60	t	274	187

Binary Non-Convex QPLIB Benchmark

mean	3.95	29.5	63.1	74.9	1	10.3	2.47
solved	74	36	16	6	91	69	83
prob#	BARON	SCIP	ANTIGONE	COUENNE	SHOT	RAPOSa	COPT
0067	1	22	117	1158	151	11	35

But Classical Optimization Solvers Face Challenges…

Problem Scales in Real-World Scenarios Are Growing Rapidly, and Exceeding the Limits of CPU-Based Mathematical Programming Solvers

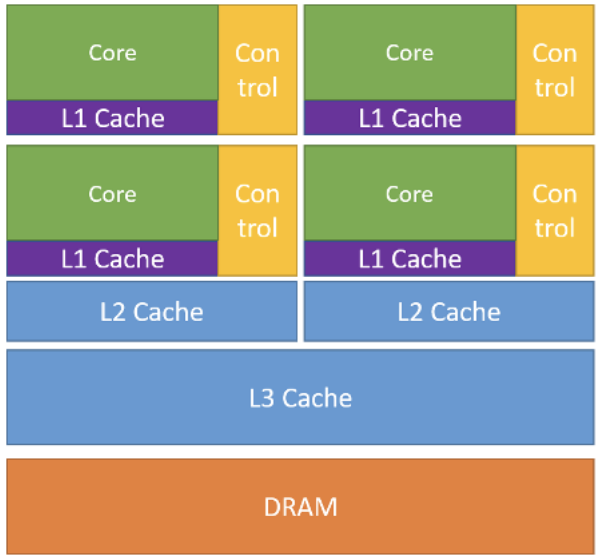
Category	Problem	Constraints	Variables	Nonzero Coefficients	CPU Solver Time (sec) — COPT
European EDA Design	zib03	19,731,970	29,128,799	104,422,573	59,400
Google PageRank	rand_1m_nodes	1,000,001	1,000,000	7,999,982	—
	rand_10m_nodes	10,000,001	10,000,000	79,999,982	—
	com-livejournal	3,997,963	3,997,962	77,358,302	—
	soc-livejournal1	4,847,572	4,847,571	78,170,533	—
Huawei Supply Chain Network Design	inv-10	4,035,449	3,758,458	15,264,380	—
	inv-20	8,368,795	7,810,584	31,718,673	—
	inv-40	13,186,756	12,066,105	49,528,729	—
	inv-60	16,227,780	14,544,689	60,372,404	—
CSG Market Clearing & Dispatch (Tolerance 1e-4)	Model A	Confidential	Confidential	Confidential	1,951.35
	Model B	Confidential	Confidential	Confidential	1,635.65

GPU-Based Optimization Solvers?

GPU Parallel Acceleration Is Not Well-Suited for Classical Mathematical Optimization Computation

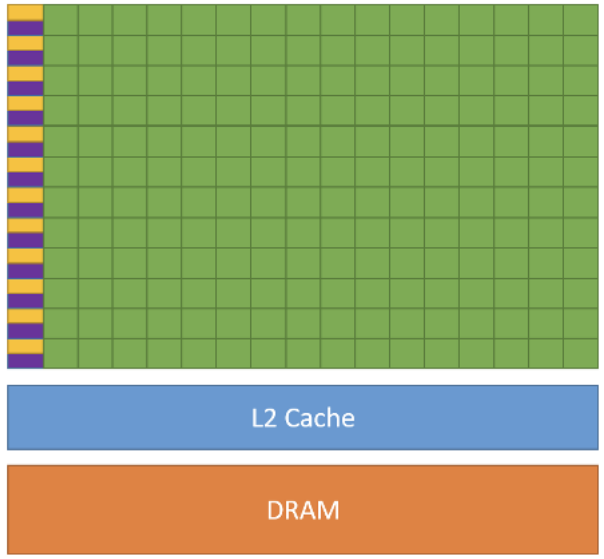
1. **First-order algorithms** suffer from low precision; numerically difficult problems converge slowly and unstably
2. **Second-order algorithms** involve matrix inversion/factorization, for which parallel acceleration on GPU yields limited improvement
3. **Integer programming algorithms** entail massive sequential and complex logical operations that are difficult to parallelize on GPU

But we made breakthroughs …



CPU

- Low parallelism, few threads
- Suitable for single-thread complex logic and computation
- e.g., matrix factorization, inversion, optimization task scheduling



GPU

- High parallelism, many threads
- Suitable for massively parallel simple tasks
- e.g., large-scale matrix-vector addition and multiplication

A Quick Summary of Recent GPU-Based Algorithms/Solvers

Category	Scenario	Problem	Constraints	Variables	Nonzeros	CPU COPT Time (sec)	GPU COPT Time (sec)
Large-Scale LP Team cuPDLP-C Algorithm	European EDA Design	zib03	19,731,970	29,128,799	104,422,573	59400	607
	Google Pagerank	rand_1m_nodes	1,000,001	1,000,000	7,999,982	-	3.56
		rand_10m_nodes	10,000,001	10,000,000	79,999,982	-	44.22
		com-livejournal	3,997,963	3,997,962	77,358,302	-	21.07
	Multi-layer Supply Chain Network Design	inv-10	4,035,449	3,758,458	15,264,380	-	1636
		inv-20	8,368,795	7,810,584	31,718,673	-	1157
		inv-40	13,186,756	12,066,105	49,528,729	-	6032
		inv-60	16,227,780	14,544,689	60,372,404	-	11102
	CSG Market Clearing & Dispatch (Tolerance 1e-4)	Model A	Sensitive Data			1951.35	254.93
		Model B				1635.65	141.01
Quadratic Programming Team PDHCG PDCS Algorithm	Multi-stage Portfolio Optimization	48期	101,808	410,256	10,891,238	9	7
		1440期	3,054,240	12,307,680	326,742,998	>3600	491
		2880期	6,108,480	24,615,360	653,486,198	>3600	581
	Fisher Market Equilibrium	100000/50	$5.2*10^6$	$4.0*10^5$	$1.0*10^6$	243	14
		100000/500	$5.02*10^7$	$4.0*10^5$	$1.0*10^7$	>3600	423
Semidefinite Programming Team cuLoARDS Algorithm	Max-Cut Problem		170m	170m		-	177
	Matrix Completion Problem		200m	20m		-	443
	Quantum Chemistry	Ground State Energy Estimation: Achieving 200× scale and speed improvement					

LP: PDHG (Primal Dual Hybrid Gradient) I

General Form Linear Programming Problem and Its Dual Problem:

$$(P)\quad \begin{aligned} \min \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

$$(D)\quad \begin{aligned} \max \quad & \mathbf{b}^\top \mathbf{y} \\ \text{s.t.} \quad & A^\top \mathbf{y} + \mathbf{s} = \mathbf{c} \\ & \mathbf{s} \geq 0 \end{aligned}$$

Transform LP into a min-max saddle-point problem:

$$\min_{x \in X} \max_{y \in Y} \mathcal{L}(x, y) = c^\top x - y^\top A\, x + y^\top b$$

PDHG

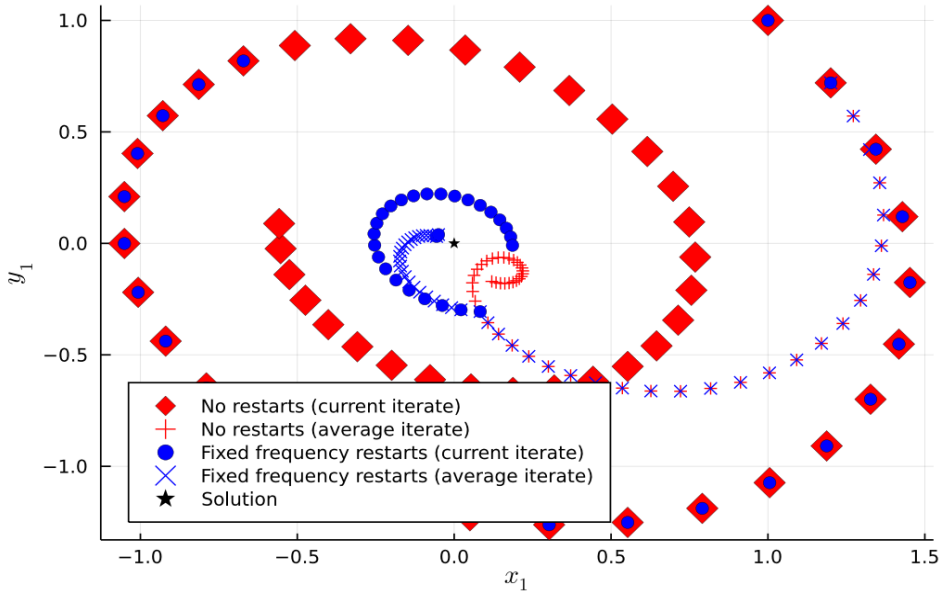
$$\begin{aligned} x^{k+1} &= \mathrm{Proj}_X\left(x^k - \tau\left(c - A^\top y^k\right)\right) \\ y^{k+1} &= \mathrm{Proj}_Y\left(y^k + \delta\left(b - A\left(2x^{k+1} - x^k\right)\right)\right) \end{aligned}$$

LP: PDHG（Primal Dual Hybrid Gradient）II

PDLP: Incorporating multiple acceleration techniques into PDHG (Applegate et al. 2021):

Theoretical advance (Applegate et al. 2023): Restart improves the iteration complexity to $O(\left(\frac{\gamma}{\alpha}\right)^2 \log\left(\frac{1}{\varepsilon}\right))$.

- Precondition,
- Adaptive Restart,
- Adaptive Stepsize,
- Primal weight update,
- Restart



In November 2023, in collaboration with the University of Chicago/MIT, the COPT team released cuPDLP-C, the world's first linear programming algorithm under a CPU+GPU heterogeneous computing architecture

cuPDLP-C: A Strengthened Implementation of cuPDLP for Linear Programming by C language,
H Lu, J Yang, H Hu, Q Huangfu, J Liu, T Liu, Y Ye, C Zhang, D Ge, arXiv preprint arXiv:2312.14832, 2023

D-PDLP: Ddistributed PDLP

D-PDLP: Scaling PDLP to Distributed Multi-GPU Systems

Hongpei Li[1], **Yicheng Huang**[2], **Huikang Liu**[3], **Dongdong Ge**[3], **Yinyu Ye**[4]
[1] Cardinal Operations
[2]Shanghai University of Finance and Economics
[3]Shanghai Jiao Tong University
[4]Stanford University
ishongpeili@gmail.com, hkl1u@sjtu.edu.cn, ddge@sjtu.edu.cn

Block-wise Random Permutation For SpMV Efficiency and Load Balancing

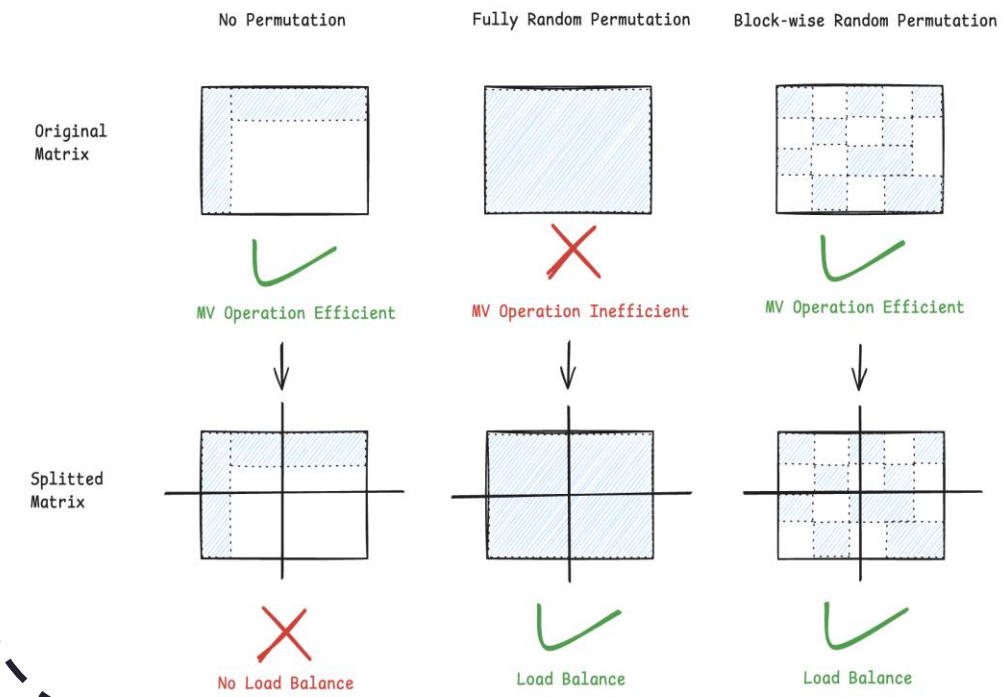


Table 2: Solving Time Comparison (seconds)

Instance	GPU	No Permutation		Full Random		Block Random	
		No NNZ	Yes NNZ	No NNZ	Yes NNZ	No NNZ	Yes NNZ
sdm_50k_500k_15_10	4	328.20	107.12	227.02	237.99	142.49	**92.33**
	8	527.91	377.41	69.06	91.09	78.19	**60.90**
zib04	4	80.80	80.95	93.07	92.54	79.70	**78.27**
	8	61.94	62.18	63.38	65.05	60.24	**58.61**
zib03	4	2926.91	2017.57	501.96	502.93	**336.53**	338.26
	8	7368.89	2012.84	300.60	291.65	**243.76**	245.29
ds1_lp	4	777.84	682.63	157.49	155.88	147.39	**142.67**
	8	293.32	690.24	129.47	127.95	124.65	**122.27**
ds2_lp	4	3742.87	3299.29	748.69	746.62	710.96	**690.54**
	8	1407.81	3325.82	621.12	617.57	594.08	**591.10**
lipa50a	4	33.48	30.95	20.84	20.60	**19.95**	**19.95**
	8	35.48	33.18	17.08	**16.77**	16.80	16.94
lipa50b	4	35.03	32.12	21.82	21.32	**21.06**	21.24
	8	36.26	35.29	17.50	17.52	**17.26**	17.31
tai50a	4	34.96	32.08	21.98	21.65	21.11	**20.80**
	8	36.95	34.97	**17.25**	17.36	17.86	17.77
tai50b	4	101.52	81.91	53.94	54.91	52.47	**52.42**
	8	92.80	88.73	44.29	43.60	43.72	**42.59**
SGM10		220.08	194.13	86.80	87.96	79.51	**75.85**

D-PDLP: Multi-GPU Acceleration of cuPDLP

Public Benchmark Results

The advantage of multi-GPU setups grows with problem scale.

Dataset	Metric	Number of GPUs 1	2	4	8
Mittelmann	Solved (< 3600s)	46	46	46	46
	Time (s) [SGM10]	21.26	19.00	18.58	19.24
	Iter/s [SGM10]	3233	3617	3800	3621
MIPLIB Small (100K - 1M)	Solved (< 1500s)	265	267	267	266
	Time (s) [SGM10]	5.81	5.77	6.12	6.80
	Iter/s [SGM10]	3453	3172	3019	2740
MIPLIB Medium (1M - 10M)	Solved (< 1500s)	92	92	92	92
	Time (s) [SGM10]	7.32	6.68	6.70	7.11
	Iter/s [SGM10]	3101	3169	3161	2965
MIPLIB Large (> 10M)	Solved (< 1500s)	13	14	16	16
	Time (s) [SGM10]	115.68	76.09	69.06	66.97
	Iter/s [SGM10]	1303	2049	2564	2758

Large Instance Results

Our Distributed PDLP exhibits **near-linear scaling with the number of GPUs** in several instances.

Source	Instance	m	n	nnz	Number of GPUs 1GPU	4GPU	8GPU
Koch	zib03	19,731,970	29,128,799	104,422,573	812	336	245
Pagerank	rand 10m nodes	10,000,001	10,000,000	79,999,982	10	5	4
	com-livejournal	3,997,963	3,997,962	77,358,302	3	2	1
	soc-livejournal1	4,847,572	4,847,571	78,170,533	2	1	1
multicommodity-flow	mcf_2500_100_500	1,512,600	126,250,100	253,750,100	2943	935	504
	mcf_5000_50_500	2,775,050	126,250,050	253,750,050	9114	2994	1677
	mcf_5000_100_250	1,775,100	127,500,100	257,500,100	9173	3159	1732
design-matching	sdm_50k_500k_15_10	5,500,135	10,000,000	690,000,000	377	92	61
QAP	wil50	3,437,600	6,252,500	19,125,000	43	31	25
	lipa50a				27	20	17
	lipa50b				28	21	17
	tai50a				30	21	18
	tai50b				82	52	43
Unit Com.	ds1	641,037	659,145	21,577,566	279	143	122
	ds2				1288	691	591

Solution Milestones of a Well-Known "Intractable" LP Instance

In a workshop in January 2008 on the *Perspectives in Interior Point Methods for Solving Linear Programs*, the instance `zib03` with 29,128,799 columns, 19,731,970 rows and 104,422,573 non-zeros was made public. As it turned out, the simplex algorithm was not suitable to solve it and barrier methods needed at least about 256 GB of memory, which was not easily available at that time. The first to solve it was Christian Bliek in April 2009, running CPLEX out-of-core with eight threads and converging in 12,035,375 seconds (139 days) to solve the LP without crossover. Each iteration took 56 hours! Using modern codes on a machine with 2 TB memory and 4 E7-8880v4 CPUs @ 2.20 GHz with a total of 88 cores, this instance can be solved in 59,432 seconds = 16.5 hours with just 10% of the available memory used. This is a speed-up of 200 within 10 years. However, when the instance was introduced in 2008, none of the codes was able to solve it. Therefore there was infinite progress in the first year. Furthermore, 2021 was the first time we were able to compute an optimal *basis* solution.

[1]Koch, Thorsten, et al. "Progress in mathematical programming solvers from 2001 to 2020." EURO Journal on Computational Optimization 10 (2022): 100031.

2008: Instance zib03[1]
29,128,799 variables
19,731,970 constraints

2009: Cplex Barrier (without crossover)
139 days (56 hours/IPM-iteration)

2019: IPM on a more advanced machine
16.5 hours

2023-24: cuPDLP-C (to 1e-6 tolerance)
1.7 hours on NVIDIA A6000
27 minutes on NVIDIA H100

2026-03
247 Seconds on Nvidia 8 H100s
545 seconds on 8 X201s(Muxi)

Excluding hardware improvement, LP (COPT and others) speed becomes 3.5x faster on average in the past 4 years

GPU Interior-Point Method Acceleration: COPT

Large-Scale LP Model: GPU Barrier Acceleration Results (Convergence Tolerance: 1e-8)

Model Scale	225s	254s	460s
210,000 constraints, 10,000,000 variables, 30,000,000 non-zero elements	GPUMode=2	GPUMode=1	CPU Barrier

GPU High-Performance Mode 225s

```
COPT> set LpMethod 2
Setting parameter 'LpMethod' to 2
COPT> set GPUMode 2
Setting parameter 'GPUMode' to 2
COPT> optlp
Model fingerprint: e61dc167

Using Cardinal Optimizer v8.0.5 on Linux (Ubuntu 24.04.4 LTS - x86_64)
The CPU model is AMD Ryzen 9 9950X 16-Core Processor
Hardware has 16 physical cores and 32 logical cores. Using instruction set X86_NATIVE (1)
Minimizing an LP problem

The original problem has:
    210000 rows, 10000000 columns and 30000000 non-zero elements
The presolved problem has:
    208802 rows, 8002135 columns and 23763346 non-zero elements

Hardware has 1 supported GPU device with CUDA 12.9
  GPU 0: NVIDIA GeForce RTX 5090 (CUDA capability 12.0)

Starting barrier solver using 16 CPU threads and GPU 0

  18  -5.49802857e+12  -5.49802857e+12   2.49e+02    6.86e-07  5.68e-12    153s
  19  -5.49802857e+12  -5.49802857e+12   6.73e+01    4.06e-07  2.27e-12    160s
  20  -5.49802857e+12  -5.49802857e+12   4.05e+01    2.83e-07  1.59e-12    166s
  21  -5.49802857e+12  -5.49802857e+12   2.00e+01    1.27e-07  1.36e-12    173s
  22  -5.49802857e+12  -5.49802857e+12   8.21e+00    5.55e-08  1.14e-12    179s
  23  -5.49802857e+12  -5.49802857e+12   3.87e+00    5.78e-08  1.14e-12    185s
  24  -5.49802857e+12  -5.49802857e+12   6.01e-01    3.74e-08  1.14e-12    192s
  25  -5.49802857e+12  -5.49802857e+12   4.06e-01    3.15e-08  1.14e-12    199s
  26  -5.49802857e+12  -5.49802857e+12   1.75e-01    1.76e-08  1.14e-12    205s
  27  -5.49802857e+12  -5.49802857e+12   4.02e-03    7.11e-09  1.36e-12    212s
  28  -5.49802857e+12  -5.49802857e+12   1.03e-03    2.04e-09  1.14e-12    218s
  29  -5.49802857e+12  -5.49802857e+12   4.54e-05    1.46e-09  1.36e-12    225s

Barrier status:                OPTIMAL
Primal objective:              -5.49802857e+12
Dual objective:                -5.49802857e+12
```

GPU Standard Mode 254s

```
COPT> set LpMethod 2
Setting parameter 'LpMethod' to 2
COPT> set GPUMode 1
Setting parameter 'GPUMode' to 1
COPT> optlp
Model fingerprint: e61dc167

Using Cardinal Optimizer v8.0.5 on Linux (Ubuntu 24.04.4 LTS - x86_64)
The CPU model is AMD Ryzen 9 9950X 16-Core Processor
Hardware has 16 physical cores and 32 logical cores. Using instruction set X86_NATIVE (1)
Minimizing an LP problem

The original problem has:
    210000 rows, 10000000 columns and 30000000 non-zero elements
The presolved problem has:
    208802 rows, 8002135 columns and 23763346 non-zero elements

Hardware has 1 supported GPU device with CUDA 12.9
  GPU 0: NVIDIA GeForce RTX 5090 (CUDA capability 12.0)

Starting barrier solver using 16 CPU threads and GPU 0

  12  -5.49802810e+12  -5.49802909e+12   1.02e+06    2.10e-03  2.53e-08    128s
  13  -5.49802835e+12  -5.49802881e+12   4.72e+05    9.67e-04  1.17e-08    135s
  14  -5.49802848e+12  -5.49802866e+12   1.88e+05    4.02e-04  4.46e-09    142s
  15  -5.49802856e+12  -5.49802858e+12   1.92e+04    4.09e-05  4.54e-10    151s
  16  -5.49802857e+12  -5.49802857e+12   2.40e+03    1.16e-05  5.30e-11    160s
  17  -5.49802857e+12  -5.49802857e+12   1.37e+03    1.15e-05  2.95e-11    167s
  18  -5.49802857e+12  -5.49802857e+12   2.49e+02    3.39e-05  6.44e-12    176s
  19  -5.49802857e+12  -5.49802857e+12   6.73e+01    5.81e-05  3.50e-12    184s
  20  -5.49802857e+12  -5.49802857e+12   4.05e+01    3.82e-05  3.08e-12    191s
  21  -5.49802857e+12  -5.49802857e+12   2.00e+01    1.74e-05  2.76e-12    198s
  22  -5.49802857e+12  -5.49802857e+12   8.21e+00    7.59e-06  2.89e-12    205s
  23  -5.49802857e+12  -5.49802857e+12   3.87e+00    3.93e-06  2.52e-12    212s
  24  -5.49802857e+12  -5.49802857e+12   5.98e-01    3.03e-06  2.84e-12    220s
  25  -5.49802857e+12  -5.49802857e+12   2.16e-01    1.50e-06  2.72e-12    228s
  26  -5.49802857e+12  -5.49802857e+12   2.17e-02    3.50e-07  2.77e-12    236s
  27  -5.49802857e+12  -5.49802857e+12   4.67e-04    1.12e-07  3.00e-12    245s
  28  -5.49802857e+12  -5.49802857e+12   2.43e-05    3.35e-07  2.66e-12    254s

Barrier status:                OPTIMAL
Primal objective:              -5.49802857e+12
Dual objective:                -5.49802857e+12
```

CPU Baseline Mode 460s

```
COPT> set LpMethod 2
Setting parameter 'LpMethod' to 2
COPT> optlp
Model fingerprint: e61dc167

Using Cardinal Optimizer v8.0.5 on Linux (Ubuntu 24.04.4 LTS - x86_64)
The CPU model is AMD Ryzen 9 9950X 16-Core Processor
Hardware has 16 physical cores and 32 logical cores. Using instruction set X86_NATIVE (1)
Minimizing an LP problem

The original problem has:
    210000 rows, 10000000 columns and 30000000 non-zero elements
The presolved problem has:
    208802 rows, 8002135 columns and 23763346 non-zero elements

Starting barrier solver using 16 CPU threads

Problem info:
    Dualized in presolve:            No
    Range of matrix coefficients:    [1e+00,1e+00]
    Range of rhs coefficients:       [2e+04,4e+05]
    Range of bound coefficients:     [9e-01,1e+03]
    Range of cost coefficients:      [2e+00,2e+03]


  12  -5.49802856e+12  -5.49802858e+12   2.19e+04    6.81e-05  2.12e-10    239s
  13  -5.49802857e+12  -5.49802857e+12   3.54e+03    1.32e-05  3.93e-11    258s
  14  -5.49802857e+12  -5.49802857e+12   1.44e+03    7.97e-06  1.75e-11    271s
  15  -5.49802857e+12  -5.49802857e+12   7.78e+02    4.38e-06  9.43e-12    284s
  16  -5.49802857e+12  -5.49802857e+12   3.55e+02    2.12e-06  4.31e-12    297s
  17  -5.49802857e+12  -5.49802857e+12   5.14e+01    8.82e-07  1.51e-12    316s
  18  -5.49802857e+12  -5.49802857e+12   7.23e+00    1.02e-06  1.03e-12    332s
  19  -5.49802857e+12  -5.49802857e+12   5.04e+00    4.19e-06  1.09e-12    345s
  20  -5.49802857e+12  -5.49802857e+12   4.25e-01    8.42e-06  1.01e-12    363s
  21  -5.49802857e+12  -5.49802857e+12   1.47e-01    5.72e-06  8.55e-13    375s
  22  -5.49802857e+12  -5.49802857e+12   5.17e-02    3.05e-06  8.99e-13    387s
  23  -5.49802857e+12  -5.49802857e+12   4.45e-02    2.67e-06  9.64e-13    412s
  24  -5.49802857e+12  -5.49802857e+12   1.12e-02    1.30e-06  1.45e-12    425s
  25  -5.49802857e+12  -5.49802857e+12   1.06e-04    1.93e-06  1.33e-12    443s
  26  -5.49802857e+12  -5.49802857e+12   4.79e-06    1.41e-07  1.08e-12    460s

Barrier status:                OPTIMAL
Primal objective:              -5.49802857e+12
Dual objective:                -5.49802857e+12
```

GPU Interior-Point Method Acceleration: COPT and GUROBI

unscaled	25.3	83.3	261	431	421	146	18.0	26.3	15.1	39.3
scaled	1.67	5.50	17.2	28.5	27.8	9.63	1.19	1.74	1	2.60
solved	65	59	56	49	50	59	61	58	64	54
65 probs	COPT	MOSEK	HiGHS	KNTRO	PDLP	XOPT	CUOPT	CUPDX	COPTG	HPRLP
L1_sixm	2	3	98	t	88	8	3	1	6	1
Linf_520c	3	6	1063	4858	233	15	3	2	2	2
a2864	1	1	199	117	10	1	1	1	1	2
bdry2	6	19	t	11	t	146	10	t	6	876
cont1	1	1	6	1	256	6	5	21	1	15
cont11	1	1	15	2	1988	23	48	60	1	36
datt256	1	2	13	6	2	5	1	1	1	1

Single-GPU Comparison: Hans-Lpfeas-Benchmark Dataset Test

SGM	H100	BW1000	BW/NV	X201	X201/NV
barrier1	12.32	15.48	1.26	20.06	1.63
barrier2	9.22	12.26	1.33	22.61	2.45

SGM	AMD 7900X
COPT-CPU	13.58

Hybridizing PDHG and Interior-Point Methods

Edward Rothberg
Gurobi Optimization, LLC
rothberg@gurobi.com

March 3, 2026

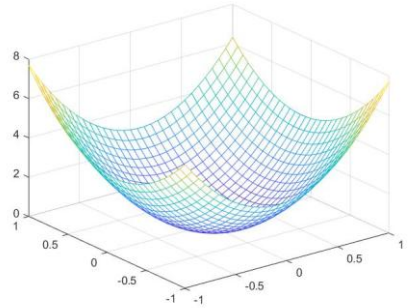
Abstract

The Primal-Dual Hybrid Gradient (PDHG) algorithm is a first-order method that can exploit GPUs to solve large-scale linear programming problems. The approach can often be faster than the alternatives, simplex and interior-point methods, typically at the cost of much lower accuracy. This paper looks at whether PDHG can be hybridized with an interior-point method to retain some of the speed advantages of the former while capturing the accuracy advantages of the latter.

Following a period of optimization, the performance of BW1000 on the interior-point method has improved significantly. Notably, barrier2 is already faster than the CPU version of COPT running on an AMD processor with a 4.7 GHz clock speed.

Quadratic Programming (QP): GPU-Based Algorithms/Solvers

Quadratic Programming



$$\min_{x \in \mathbb{R}^n} \quad \frac{1}{2} x^T Q x + c^T x$$

$$\text{s.t.} \quad Ax \leq b$$

Quadratic programming is widely used in various applications:

- Finance: mean–variance portfolio optimization model
- Machine Learning: Support Vector Machines (SVM)
- Control: Model Predictive Control (MPC)
- Nonlinear optimization: Sequential Quadratic Programming (SQP)

MPC Model:

$$\begin{aligned} \text{minimize} \quad & x_T^T Q_T x_T + \sum_{t=0}^{T-1} \left(x_t^T Q x_t + u_t^T R u_t \right) \\ \text{subject to} \quad & x_{t+1} = A x_t + B u_t \\ & x_t \in \mathcal{X},\ u_t \in \mathcal{U} \\ & x_0 = x_{\text{init}}, \end{aligned}$$

Markowitz Portfolio Model:

$$\begin{aligned} \text{maximize} \quad & \mu^T x - \gamma \left(x^T \Sigma x \right) \\ \text{subject to} \quad & 1^T x = 1,\ x \geq 0, \end{aligned}$$

Primal-Dual Hybrid Conjugate Gradient （PDHCG）I

Quadratic Programming

$$\min_{x\in\mathbb{R}^n} \quad \frac{1}{2}x^T Qx + c^T x$$
$$\text{s.t.} \quad Ax \le b$$

Min-Max Stationary Point Problem

$$\min_{x}\max_{y\ge 0} \quad \mathcal{L}(x,y) = \frac{1}{2}x^T Qx + c^T x + y^T Ax - b^T y$$

Primal-Dual Hybrid Conjugate Gradient, PDHCG

Advantages：

- **Major cost is matrix-vector multiplication, which is easy for parallel computation**
- **Use CG to solve primal subproblem, decreasing outer iteration complexity**
- **Suitable for extremely large scale problems with GPU acceleration.**

$$y^{k+1} = \text{proj}_{\mathbb{R}^m_+}\{y^k + \delta_k(A(x^k + \theta_k(x^k - x^{k-1})) - b)\},$$
$$x^{k+1} = \arg\min_x \frac{1}{2}x^T Qx + c^T x + (y^{k+1})^T Ax - b^T y^{k+1} + \frac{1}{2\tau_k}\|x - x^k\|^2.$$

Use CG to INEXACTLY solve primal subproblem

Restarted primal-dual hybrid conjugate gradient method for large-scale quadratic programming
Y Huang, W Zhang, H Li, D Ge, H Liu, Y Ye, Accepted, IJOC, 2024

Primal-Dual Hybrid Conjugate Gradient （PDHCG） II

Table 1 Comparison of the theoretical results between rAPDHG and PDHCG

Method	outer loop complexity	extra CG steps
rAPDHG	$\mathcal{O}\left(\left(\|A\| + \sqrt{\|Q\|} + \frac{\|Q\|}{\|A\|}\right)\log\frac{1}{\epsilon}\right)$	-
PDHCG-fixed	$\mathcal{O}\left(\left(\|A\| + \sqrt{\gamma_K^N\|Q\|} + \frac{\gamma_K^N\|Q\|}{\|A\|}\right)\log\frac{1}{\epsilon}\right)$	N
PDHCG-adaptive	$\mathcal{O}\left(\|A\| \cdot \log\frac{1}{\epsilon}\right)$	$\log_r \frac{\zeta}{2(1+\tau\|A\|)(1+\tau\|Q\|)}$

When the condition number of the quadratic term Q is poor (ill-conditioned), the PDHCG algorithm is **only mildly affected**.

Desired Properties：

- Linear Convergence
- Reduce dependency on the condition number of Q with CG

Table 2 Solving time (s) on Random QP with different condition numbers

Condition Number	PDHCG(GPU) Iteration	PDHCG(GPU) Time	PDHCG(GPU) CG extra	rAPDHG(GPU) Iteration	rAPDHG(GPU) Time	SCS(GPU) Time	OSQP Time	Gurobi Time
10^0	406.54	**1.74**	1375.84	795.62	2.00	4.66	439.58	4574.38
10^1	586.00	**2.08**	4092.57	1943.55	3.11	10.36	432.65	4074.02
10^2	675.51	**3.07**	7726.04	5462.20	5.72	43.75	435.40	2165.79
10^3	782.69	**3.75**	11357.09	23041.33	17.66	127.69	432.03	1743.56
10^4	1087.15	**5.68**	15733.61	58098.26	30.61	f	406.51	1891.62
10^5	1337.01	**7.21**	21614.69	t	t	f	396.84	1606.98

[1] "t" means the algorithm reached the time limit (7200s), while "f" means failure to find a correct solution.

[2] "CG extra" means the total number of CG iterations.

PDHCG: Financial QP Model Performance

Large-Scale Portfolio Problem

$$\begin{aligned} \max \quad & \mu^T x - \gamma(x^T \Sigma x) \\ \text{s.t.} \quad & \mathbf{1}^T x = 1 \\ & x \geq 0 \end{aligned} \quad \Rightarrow \quad \begin{aligned} \min \quad & x^T D x + y^T y - \gamma^{-1} \mu^T x \\ \text{s.t.} \quad & y = F^T x \\ & \mathbf{1}^T x = 1, x \geq 0 \end{aligned}$$

where $\Sigma = FF^\top + D$ is the risk model covariance matrix.

Random Portfolio Problems.

Dimension	PDHCG		PDQP		SCS			OSQP	GUROBI
	GPU	CPU	GPU	CPU	GPU	ID CPU	D CPU	CPU	CPU
1.00E+03	0.64	0.53	0.73	0.54	0.26	423.21	0.03	0.01	**0.01**
1.00E+04	1.37	1.63	2.10	4.35	5.71	1730.23	0.56	**0.11**	0.16
1.00E+05	**1.70**	33.55	3.60	58.30	172.42	f	f	f	f
1.00E+06	**8.25**	2309.99	39.58	f	f	f	f	f	f

PDHCG: Real Convex QP Performance

Large-Scale Real World QP Dataset：

Table 11 Solving time (s) over large-scale LASSO problems in LIBSVM and SuiteSparse Matrix Collection.

Problem	m	n	Sparsity	PDHCG	rAPDHG	SCS(GPU)	OSQP	COPT
SLS	1,748,122	62,729	6.21E-05	**3.35**	7.30	345.09	80.32	88.21
rcv1_test	677,399	47,236	1.55E-03	**7.12**	19.54	t	t	t
avazu-site.tr	23,567,843	1,000,000	1.50E-05	**1377.54**	5124.82	t	t	t
avazu-app	40,428,967	1,000,000	1.50E-05	**1429.55**	5557.97	t	t	t
avazu-site	25,832,830	1,000,000	1.50E-05	**4224.95**	t	t	t	t
kddb2010_test	748,401	1,163,024	7.74E-06	**11.10**	46.49	490.66	255.81	69.57
kdda2010_test	510,302	20,216,830	1.87E-05	**61.00**	148.01	t	t	t
kddb2010_train	19,264,097	1,163,024	7.97E-06	**387.00**	1715.50	t	t	t
kdda2010_train	8,407,752	20,216,830	1.80E-06	**2705.94**	t	t	t	t

[1] "t" means the algorithm reached the time limit (7200s).

PDHCG: Convex Nonlinear Problem Performance

- Large-Scale Market Equilibrium Problem Solving
- Large-scale real-world problem:
  - real transaction data from JD.com in March 2018, covering 2.5 million customers and 30,000 products.
- Further exploration of market equilibrium problems:
  - the Arrow–Debreu problem.

n	m	Sparsity	PDHCG		PDHG		PDCS	Mosek	PDLS
			Iteration	Time	Iteration	Time			
10^3	400	0.2	2,612	1.7	4303.8	4.5	13.4	**0.6**	356.9
10^4	4,000	0.02	2,758	**3.6**	7604	7.1	1079.3	118.8	f
10^5	4,000	0.02	1,200	**8.3**	10463	29.9	f	1038.9	f
10^6	4,000	0.02	2,283	**116.3**	26793	739.1	f	f	f
10^7	4,000	0.01	2,423	**589.7**	29749	4,374.2	f	f	f

Note. "f" indicate failure due to exceeding the time limit (7200s) or returning errors.

n	m	Sparsity of E	Sparsity of U	Iterations	Time
10^3	400	0.5	0.2	12.0	2.7
10^4	4000	0.05	0.02	9.3	13.0
10^5	4000	0.05	0.02	9.2	62.4
10^6	4000	0.05	0.02	9.7	560.7
10^7	4000	0.02	0.01	33.1	2305.3

m	n	nonzeros	PDHCG		PDHG		Mosek	PDCS
			Iteration	Time	Iteration	Time		
1000	11716	82530	2471.30	3.32	4133.51	4.74	8.92	19.69
2000	13541	136877	2807.39	3.84	4136.93	4.85	25.12	44.33
5000	15887	265220	3119.08	4.43	6279.22	6.75	78.27	124.55
10000	17706	433607	3404.01	5.24	7430.19	9.56	172.19	314.90
20000	19449	698579	3668.47	6.10	7430.19	9.56	478.51	-
50000	21410	1278391	3925.15	7.52	14663.67	21.51	-	-
100000	22733	1967356	4336.00	11.51	24126.07	40.74	-	-
200000	23853	2937906	4584.44	16.11	45219.31	90.50	-	-
500000	25214	4679648	6120.00	29.85	68598.64	170.73	-	-
1000000	26115	6189526	7968.19	55.29	73132.75	215.25	-	-
2557841	27330	8073837	17452.47	165.76	103618.02	348.91	-	-

"-" means the algorithm fail to solve the problem within 7200s.

Semidefinite Programming (SDP): the Factor-Representation

Consider the standard SDP: $\langle A_i, X \rangle = b_i,\ i=1,\ldots,m$

$$\min_{X \in \mathbb{S}^n} \ \langle C, X \rangle \quad \text{s.t.} \quad \mathcal{A}(X) = b, \quad X \succeq 0,$$

$$\min_{U \in \mathbb{R}^{n \times r}} \ \langle C, UU^\top \rangle \quad \text{s.t.} \quad \mathcal{A}(UU^\top) = b.$$

The low-rank factorization helps save:

- **storage**
- **computation efforts**

A low-rank factorization method (Burer and Monteiro, 2003) uses the **Lagrangian Multiplier method** to solve the (**nonconvex**) problem above.

The **column-dimension** selection of *U* (**rank** of *X*) plays an important role.

Theorem 0. If the SDP has a solution, it has a solution with rank no more than $\sqrt{2m}$ (Caratheodory's theorem…)

SDP Solution Rank: Approximate $\mathbf{O(\log(m))}$

Theorem 1.1 of (So et al. 2008) Let $A_1, \ldots, A_m \in \mathbb{R}^{n \times n}$ be symmetric positive semidefinite matrices, and let $b_1, \ldots, b_m \geq 0$. Suppose that there exists an $X \succeq 0$ such that $\langle A_i, X \rangle = b_i$ for $i = 1, 2, \ldots, m$. Let $r = \min\{\sqrt{2m}, n\}$. Then, for any $d \geq 1$, there exists an $X_0 \succeq 0$ with $\operatorname{rank}(X_0) \leq d$ such that:

$$\beta(m,n,d) \cdot b_i \leq \langle A_i, X_0 \rangle \leq \alpha(m,n,d) \cdot b_i \qquad \text{for } i = 1, \ldots, m$$

where:

$$\alpha(m,n,d) = \begin{cases} 1 + \frac{12\ln(4mr)}{d} & \text{for } 1 \leq d \leq 12\ln(4mr) \\ 1 + \sqrt{\frac{12\ln(4mr)}{d}} & \text{for } d > 12\ln(4mr) \end{cases}$$

and

$$\beta(m,n,d) = \begin{cases} \frac{1}{e(2m)^{2/d}} & \text{for } 1 \leq d \leq 4\ln(2m) \\ \max\left\{ \frac{1}{e(2m)^{2/d}}, 1 - \sqrt{\frac{4\ln(2m)}{d}} \right\} & \text{for } d > 4\ln(2m) \end{cases}$$

Moreover, there exists an efficient randomized algorithm for finding such an X_0.

Extension of the Johnson–Lindenstrauss lemma

The Splitting Technique and ADMM (LoRADS, Han et al. 24 IJC)

$$\min_{U \in \mathbb{R}^{n \times r}} \ \langle C, UU^\top \rangle \ \text{ s.t. } \ \mathcal{A}(UU^\top) = b.$$

We "split" the variable U into

$$UU^T \rightarrow UV^T \quad s.t. \quad U = V$$

$$\min_{U,V \in \mathbb{R}^{n \times r}} \ \langle C, UV^\top \rangle + \frac{\gamma}{2}\|U - V\|_F^2 \ \text{ s.t. } \ \mathcal{A}(UV^\top) = b.$$

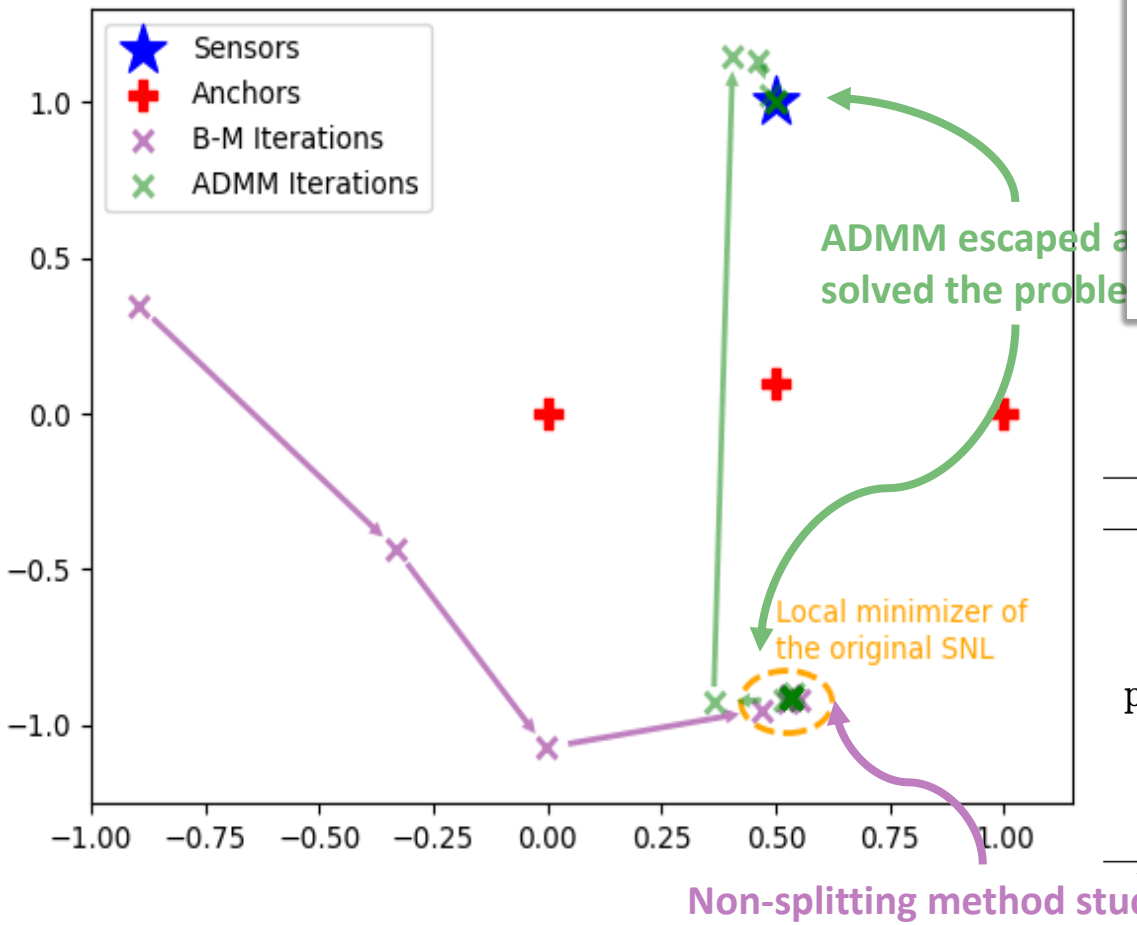
Then we apply the ADMM to tackle the problem on (iteratively) solving subproblems:

$$U^{k+1} \leftarrow \operatorname{argmin}_U \mathcal{L}_\rho(U, V^k, \lambda^k)$$
$$V^{k+1} \leftarrow \operatorname{argmin}_V \mathcal{L}_\rho(U^{k+1}, V, \lambda^k)$$
$$\lambda^{k+1} = \lambda^k + \rho\big(\mathcal{A}(U^{k+1}(V^{k+1})^\top) - b\big)$$

The Splitting Technique and ADMM

Benefits:

ADMM's "split search" may escape stuck points



The ADMM subproblem resulting in solving well conditioned linear system:

$$\left(\rho\sum_{i=1}^{m}\mathrm{vec}(A_iV^k)\mathrm{vec}(A_iV^k)^\top+\gamma I_{nr\times nr}\right)\mathrm{vec}(U^{k+1})$$
$$=-\mathrm{vec}\left(CV^k-\gamma V^k+\sum_{i=1}^{m}\lambda_i^kA_iV^k-\rho\sum_{i=1}^{m}b_iA_iV^k\right),$$

Typically, **warm-started CG** takes about $10^{-5}\mathbf{n}$ iterations for a $\mathbf{n}$-dimensional linear system:

Problem	$nr=n\sqrt{2m}$	ADMM Iterations	Total CG Iters	Avg CG Iters
G60	828251	100	845	8.45
MC_3000	4092176	17	9626	566.24
G40_mb	126586	104	2252	21.65
p_auss2_3.0	1230829	1656	11152	6.73
qap7	1338	1829	91128	49.82
qap10	4564	29452	226521	7.69
theta12	255011	1261	6938	5.50

Bridging the Split Solutions

We obtain a solution pair $(\overline{U}, \overline{V}, \overline{\lambda})$ of the penalized problem

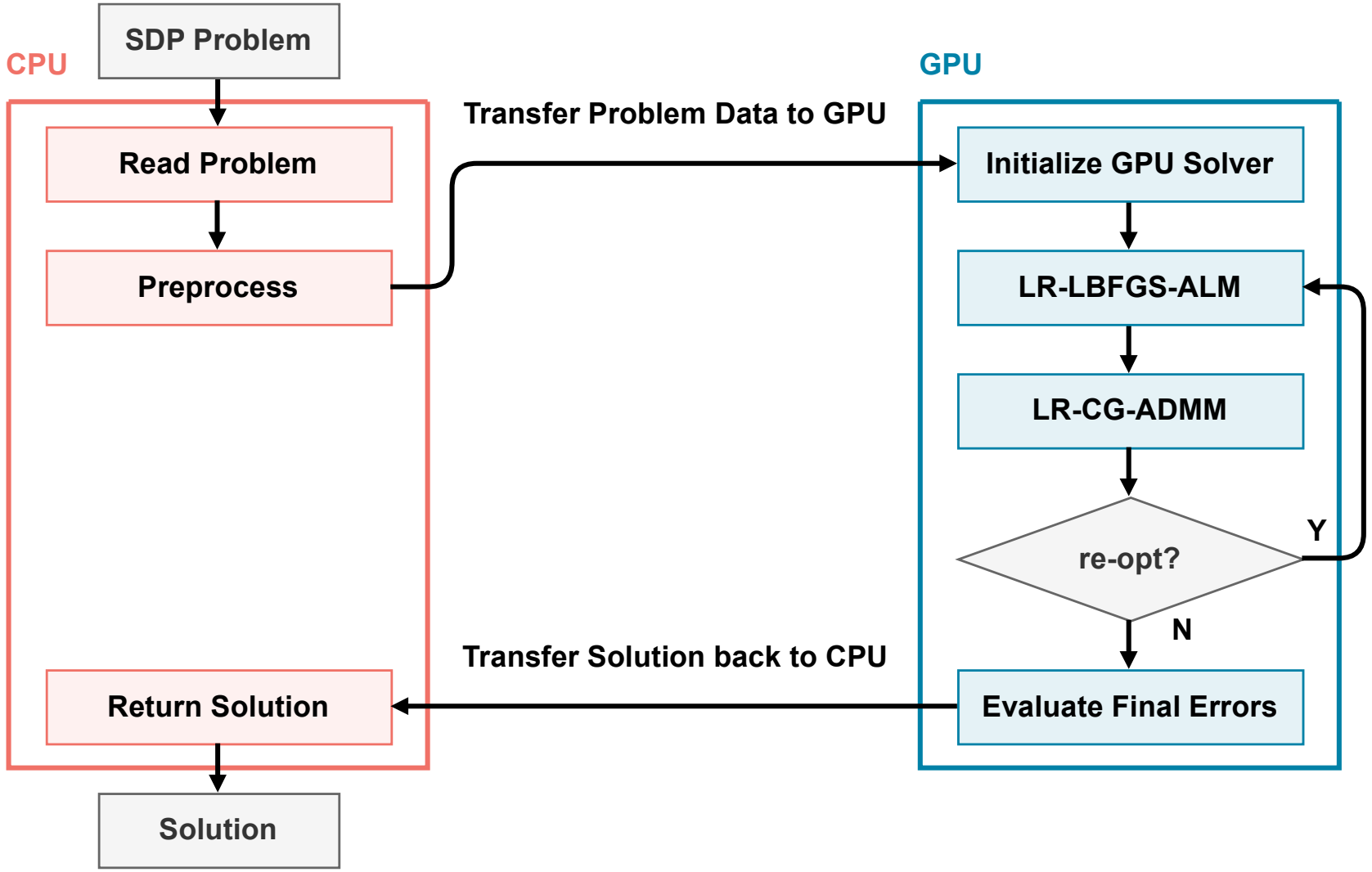
$$\min_{U,V \in \mathbb{R}^{n \times r}} \quad \langle C, UV^\top \rangle + \frac{\gamma}{2} \|U - V\|_F^2 \quad \text{s.t.} \quad \mathcal{A}(UV^\top) = b.$$

Theorem 2 **If $(\overline{U}, \overline{V}, \overline{\lambda})$ is a ε-KKT point of the problem above and γ is large enough, then $(\frac{\overline{U}+\overline{V}}{2}, 2\overline{\lambda})$ is a 3ε-KKT point of the original non-splitting factorization problem .**

$$\min_{U \in \mathbb{R}^{n \times r}} \quad \langle C, UU^\top \rangle \quad \text{s.t.} \quad \mathcal{A}(UU^\top) = b.$$

LoRADS Solving on GPUs (cuLoRADS, Han et al. 2024)

Structural Design → keeping most of computation and data on the GPU



Milestones of Solving SDP Instances

Solvers based on interior-point method:

Benchmark problem: Max Cut of size (n, m)

- **[1997] SDPA (Fujisawa et al.),**
  - **a (1250, 1250) in 31 hours**
- **[1998] DSDP (Benson, Ye and Zhan),**
  - **a (1000, 1000) in ~ 5 minutes**
- **[2008] DSDP5 (Benson and Ye),**
  - **a (2000, 2000) in < 4 minutes**
- **[2010] According to Yamashita et al., 2010, a (5000, 5000)**
  - **SDPA 7.3.1 in ~ 10 minutes**
  - **CSDP 6.0.1 in ~ 10 minutes**
  - **SDPT3 4.0β in ~ 20 minutes**
  - **SeDuMi 1.21 > 1 hour**
- **[2022] HDSDP (Gao, Ge & Ye),**
  - **a (5000, 5000) in < 1 minutes**
- **[Recent years] Modern SOTA solvers,**
  - **can solver ~ $(10^4, 10^4)$ in minutes**

Solvers based on First-Order type methods:

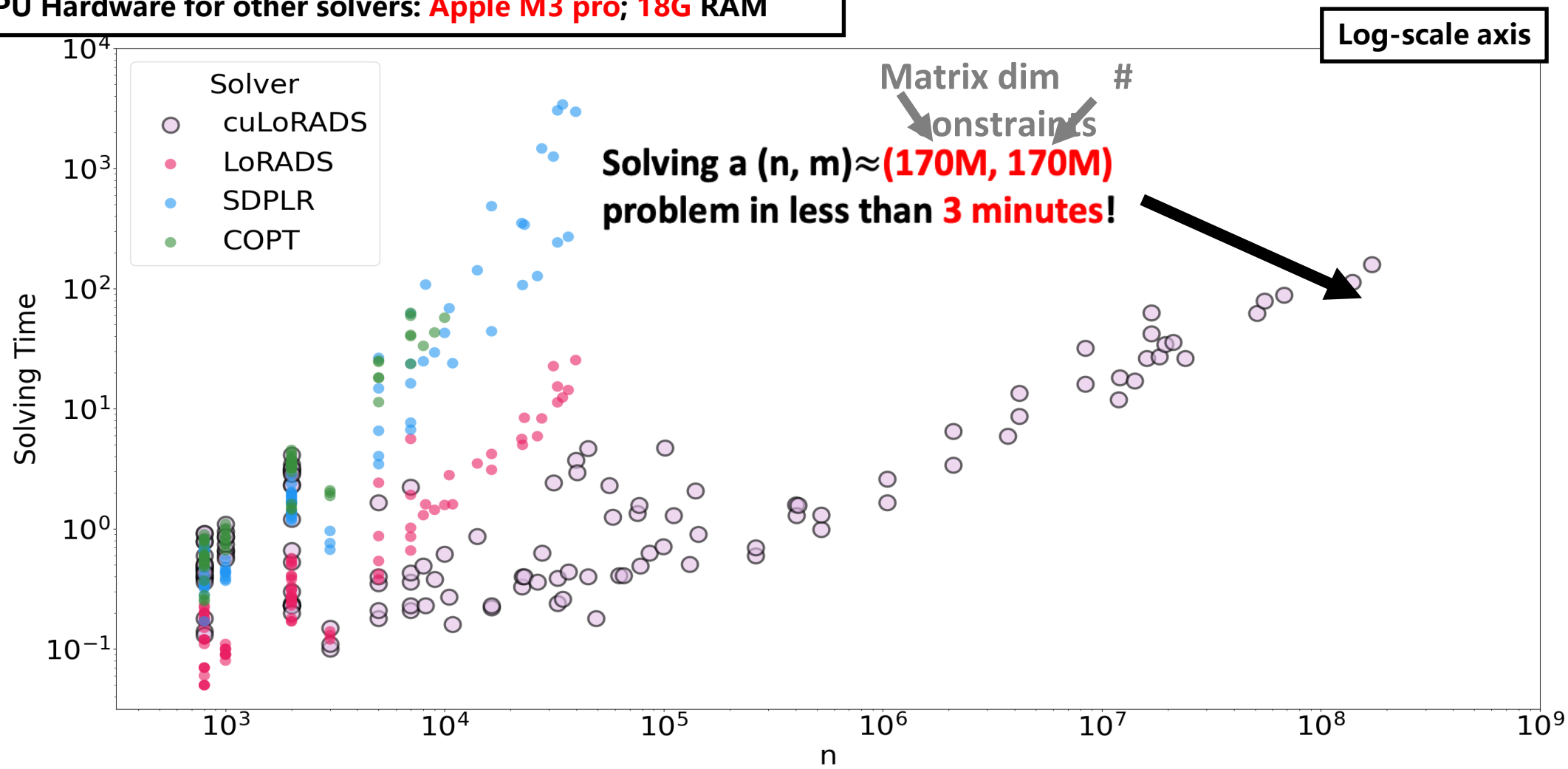
- **[2001] The Burer-Monteiro low-rank method (Burer and Monteiro, 2003),**
  - **a (20000, 20000) Max Cut in ~ 8 minutes**
- **[2021] SketchyCGAL (Yurtsever et al., 2021),**
  - **~ $(10^7, 10^7)$ Max Cuts in ~ 30 - 50 hours**
- **[Mar 2024] HALLaR (Monteiro et al., 2024),**
  - **a $(2 \times 10^5, 2 \times 10^7)$ Matrix Completion in 7.8 hours**

[June 2024] cuLoRADS (Han et al.):

- **Same ~ $(10^7, 10^7)$ Max Cuts in (Yurtsever et al., 2021) solved to high accuracy**
  - **in 10 seconds – 1 minutes (vs. 30 - 50 hours)**
- **A ~ $(4 \times 10^5, 1.6 \times 10^7)$ Matrix Completion similar to (Monteiro et al., 2024) solved to high accuracy**
  - **in ~ 2 seconds (vs. 7.8 hours)**

Computational Results on Max-Cut SDP Problems

GPU Hardware for cuLoRADS.jl: Nvidia H100; 80G VRAM
CPU Hardware for other solvers: Apple M3 pro; 18G RAM



Extension to "Pdcs: A primal-dual large-scale conic programming solver with gpu enhancements," Lin et al., arXiv preprint arXiv:2505.00311

Impact across Multiple Disciplines!

Open sourced cuLoRADS?

NR

Nicholas Rubin <nickrubin@google.com>

Thursday, August 1, 2024 at 14:49

To: Han, Josh; yinyu-ye@stanford.edu

Hi Josh and Prof. Ye,

My name is Nicholas Rubin and I'm a research scientist at Google working on quantum computing. I saw your paper on the GPU implementation of cuLoRADS and it looked like a really promising implementation for large scale SDP solving. I was wondering if you were planning on open sourcing this code.

You may wonder why someone in quantum algorithms is interested in semidefinite programming. In fact, there are a number of reasons why large scale SDP solving is important as a preprocessing step for quantum algorithms. I currently use my own implementation of RRSDP and the boundary point algorithm (BPSDP) for this preprocessing. It is also useful in lower-bound methods for quantum chemistry simulation (your table 7 seems to have molecular systems) which is the area I am primarily interested in.

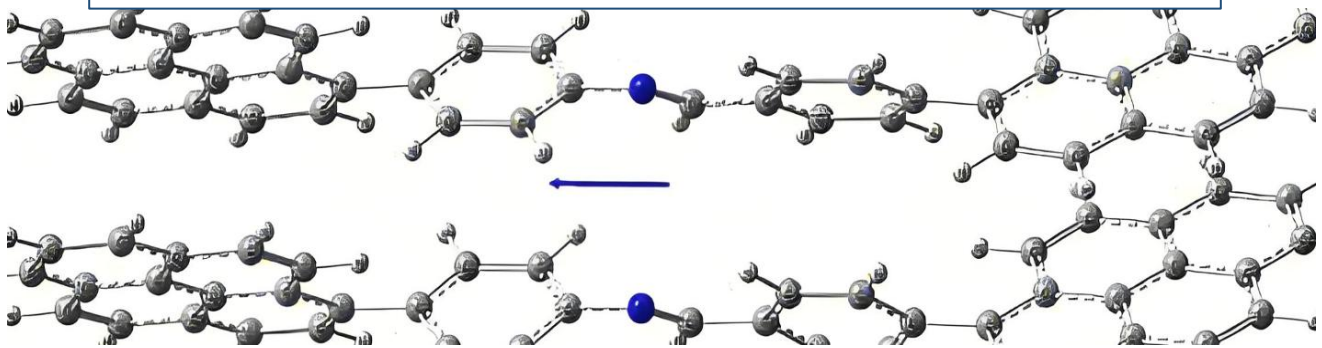
Thanks for the info,

Nicholas

--

Dr. Nicholas Rubin
Google Quantum AI
Research Scientist
Email: nickrubin@google.com
Phone: (408) 393-1064

Google
Quantun & AI Lab



Harvard
Robotics

Hi, I am Shucheng Kang, a first-year Ph.D. student in the Harvard EE department, and my advisor is Prof. Heng Yang. I just saw your exciting work on GPU-accelerated low-rank ADMM solver and was surprised by the huge scale you've achieved.

I worked on large-scale SDPs generated from Moment-SOS Hierarchy. Recently, we also developed a GPU-based (in CUDA and C++) sGS-ADMM SDP solver for multiple-cone SDP problems from trajectory optimization in the robotics area (https://www.arxiv.org/abs/2406.05846). Since the relaxation is tight, the primal SDP is low-rank. However, we tried different low-rank solvers, like ManiSDP and STRIDE, and found they performed poorly in our problem. So, we have to resort to general first-order methods. Since your solver shows unprecedented scalability, I wonder if you can share the Julia codes (or you can mask the core part / send me the binary code) and let me test our problem on your solver. If your solver works well in trajectory optimization, it will be a huge breakthrough in robotics!

SDP4S：Quantum Chemistry

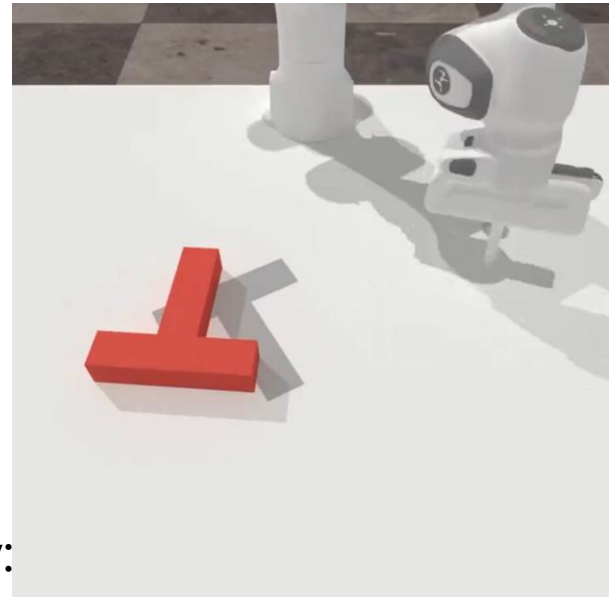
Low GH, King R, Berry DW, Han Q, DePrince III AE, White AF, Babbush R, Somma RD, Rubin NC. Fast Quantum Simulation of Electronic Structure by Spectral Amplification. Physical Review X. 2025 Oct 1;15(4):041016.

- It brings quantum simulation of real molecules from a “theoretical concept” to a **truly engineering-realizable framework**.
- It requires solving a massive semidefinite programming (SDP) problem, which has been widely considered computationally intractable for the past two decades.
- Using traditional CPU-based SDP methods, only electronic orbital models with 6–8 dimensions can be handled.
- In contrast, with cuLoRADS:
  - The solvable scale of SOS–SDP is increased to the level of “20–40 orbitals”.
  - The number of feasible Hamiltonian terms is raised from (10^4) to (10^5)–(10^6).
  - The quantum simulation resource estimates can, for the first time, provide meaningful lower bounds for medium-sized molecular systems.

SDP4S： Robotics MPC Problem I

A very important class of problems in robot control is trajectory optimization:

$$\begin{aligned} \min_{\{u_k\}_{k=0}^{N-1},\{x_k\}_{k=0}^{N}} \quad & l_N(x_N) + \sum_{k=0}^{N-1} l_k(x_k, u_k) \\ \text{subject to} \quad & x_0 = x_{\text{init}} \\ & F_k(x_{k-1}, u_{k-1}, x_k) = 0, \ \forall k \in [N] \\ & (u_{k-1}, x_k) \in \mathcal{C}_k, \ \forall k \in [N] \end{aligned}$$



This can be relaxed into an SDP via the Sum-of-Squares (SOS) hierarchy:

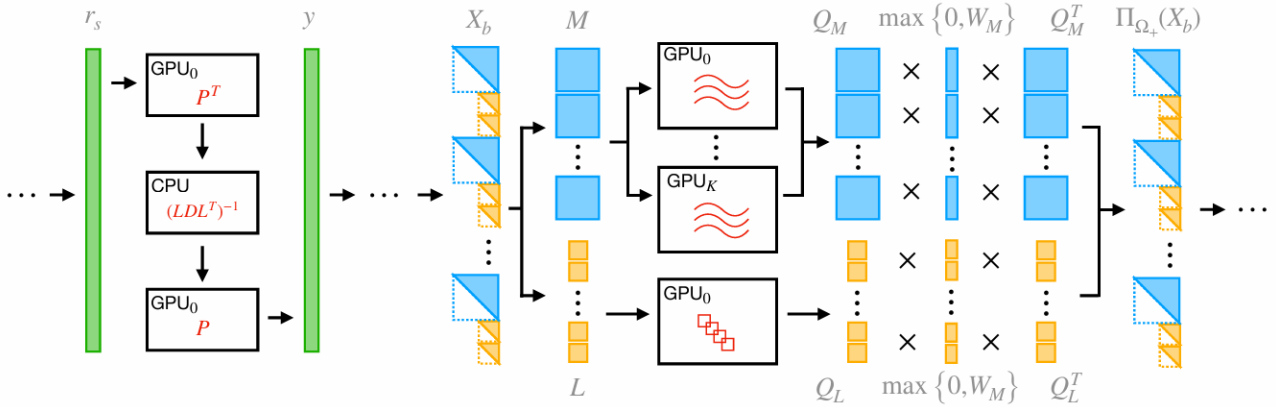
$$\begin{aligned} \inf_{y} \quad & L_y(f) \\ \text{s.t.} \quad & M_r(y, I_k) \succeq 0, \quad k = 1, \ldots, p \\ & M_{r-r_j}(g_j\, y, I_k) \succeq 0, \quad j \in J_k; \quad k = 1, \ldots, p \\ & y_0 = 1, \end{aligned}$$

- **Main Advantages:** Compared to neural network algorithms, higher control precision and stronger interpretability.
- **Main Difficulty:** As the SOS hierarchy level increases, the problem scale grows **exponentially**.
- **Research Direction:** Leveraging Primal-Dual first-order algorithms and GPU acceleration increase the solvable problem scale.

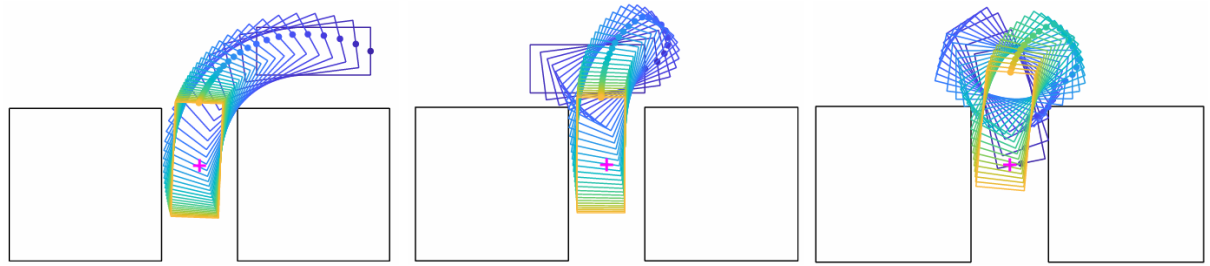
PushT Preliminary Experimental Results

κ	N	m	n	Iterations	Time (s)	Mosek Time (s)
2	5	25,056	50,708	5,880	35.64	**9.03**
2	20	102,576	214,913	5,240	42.34	**21.52**
2	30	154,256	324,383	5,880	55.17	**38.12**
3	5	352,038	688,387	10,080	**204.2**	1330.07
3	10	729,143	1,441,892	9,920	**458.32**	2,492.54
3	20	1,483,353	2,948,902	12,040	**940.79**	OOM

SDP4S: Robotics MPC Problem II



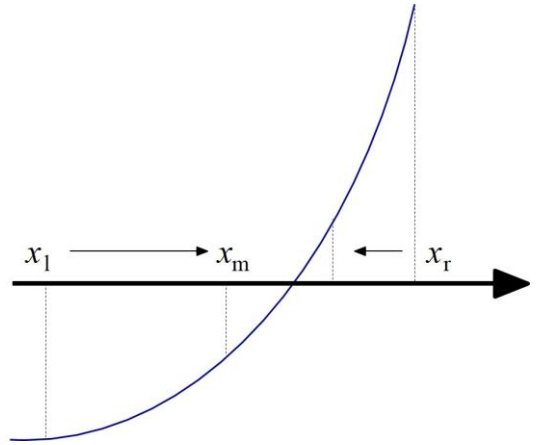
- Leverage the block-structured chain-like architecture of the problem
- Exploit GPU hardware characteristics to design highly parallel algorithms
- Significantly increase the scale of solvable problems



	d_x	d_u	size(M)	$\#M$	size(L)	m	$\log_{10}\xi$	Total Time			
								MOSEK	CDCS	SDPNAL+	cuADMM
Pendulum	4	1	55	30	10	47351	**−2.58 (−2.93)**	9.5s	110.3s	cold: 264.4s	cold: 31.2s (14.6s)
							−2.71 (−2.97)			warm: 2.95s	warm: **0.66s (0.62s)**
Cart-Pole	5	1	105	30	14	168961	**−2.89 (−2.98)**	**112.0s**	1471s	3671s	183.0s (173.1s)
Car Back-in	7	4	190	30	19	509141	**−1.87 (−1.99)**	> 5h	6123s	> 5h	**431.5s (536.8s)**
Vehicle Landing	8	2	190	50	19	946326	**−2.47 (−2.42)**	> 5h	2.9h	> 5h	**919.2s (1165s)**
Flying Robot	8	4	231	60	21	1595001	**−2.57 (−2.49)**	**	> 5h	> 5h	**1648s (2027s)**

SDP4S：the Quantum Ordered Search Problem

OSP is a fundamental computational task: find a specific item in a sorted list of N elements



- **Classically: The optimal binary search requires $[log_2 N]$ queries.**
- **Quantum computers offer a constant-factor speedup for the OSP.**
- **Known Lower Bound: Any exact quantum algorithm requires at least $\frac{lnN-1}{\pi} \approx 0.221 log_2 N$ queries.**
- **Existing Upper Bounds: historically established by constructing explicit algorithms within the translation-invariant algorithms framework.**

SDP Approach to Establish Upper bounds

- In 2007, Childs, Landahl, and Parrilo showed that the upper bound can be found by checking the feasibility of an Semidefinite Program (SDP): a set of (increasing) $N \times N$ PSD matrices $X_0, X_1, \cdots, X_k$ (k: query number) that satisfy:

$$
\begin{aligned}
X_0 &= E/N \\
\mathcal{T}_t X_t &= \mathcal{T}_t X_{t-1} \qquad \forall t \in \{1, \ldots, k\} \\
X_k &= I/N \\
\mathrm{Tr}(X_t) &= 1 \qquad \forall t \in \{0, \ldots, k\}
\end{aligned}
$$

k: number of queries

E: all-one matrix

I: identity matrix

$\mathcal{T}_t$: linear operator

$$(\mathcal{T}_t X)_i := \mathrm{Tr}_i X + (-1)^t \mathrm{Tr}_{i-N} X \qquad \mathrm{Tr}_i X = \begin{cases} \sum_{\ell=1}^{N-i} X_{\ell,\ell+i} & \text{if } i \geq 0 \\ \sum_{\ell=1}^{N+i} X_{\ell-i,\ell} & \text{if } i < 0. \end{cases}$$

- The optimal 4-query algorithm was found using this approach, searching up to (largest) N=605 items and establishing an upper bound of $4log_{605}N \approx 0.433log_2N$.
- For the 5-query case, the SDP approach was considered "out of reach of SDP solvers" for over 18 years

Recent Breakthroughs on Quantum OSP

- **Feasibility for k=5, N*=7265:**
  - **The SDP problem was numerically solved by cuLoRADS to an absolute tolerance of 10^{-8} in about 21 hours.**
- **Infeasibility "Certificate" for k=5, N=7266:**
  - **cuLoRADS produced a feasible dual solution with a positive dual objective value (1.92×10^{3}), which establish a upper bound 0.390**

$$\min_{X \in \mathbb{S}^n_+} \quad \langle 0, X \rangle \quad \text{subject to} \quad \mathcal{A}(X) = b\,,$$

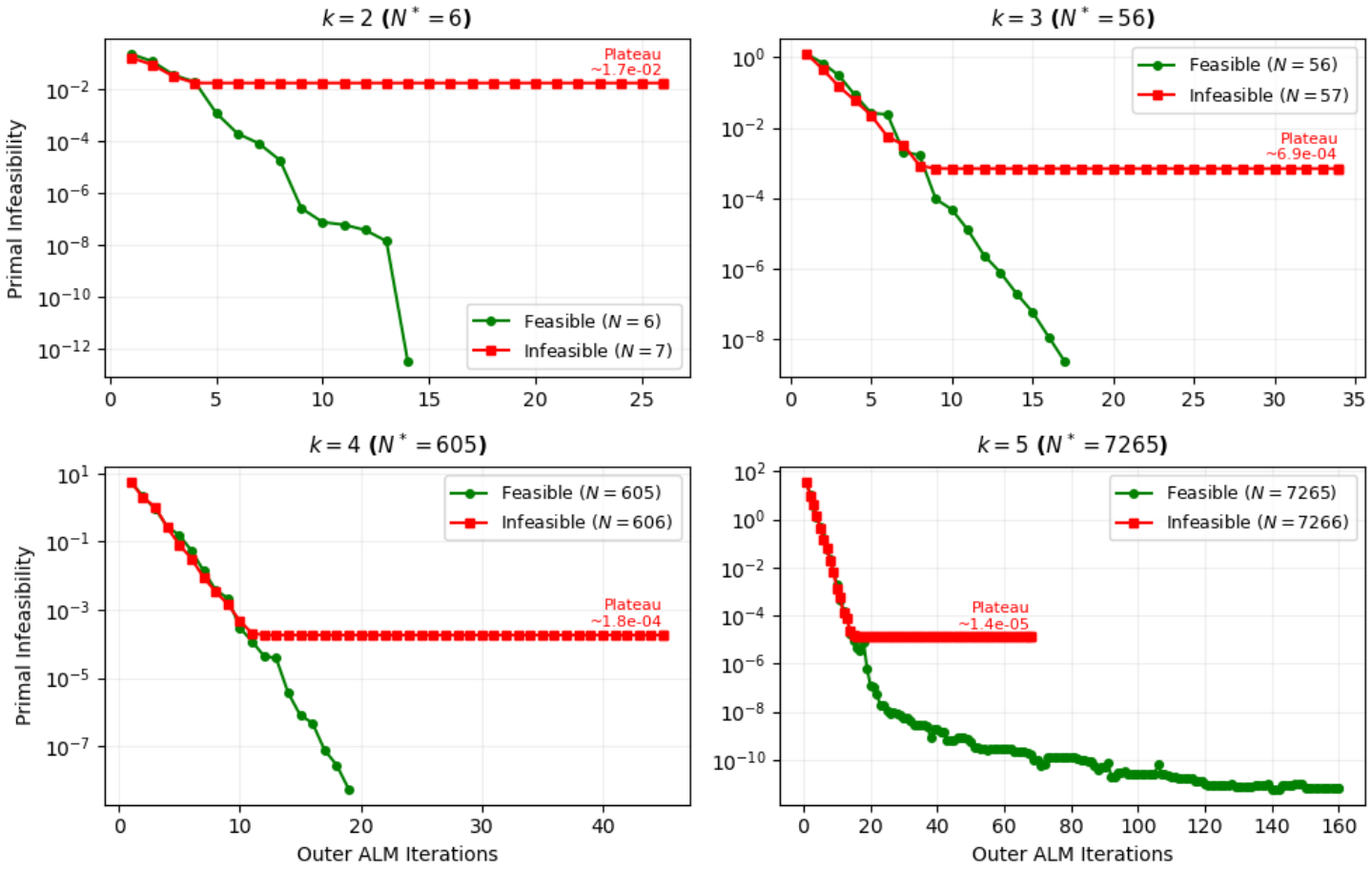
Infeasibility Certificate: a dual solution with positive objective value.

Last month, for k=6, N>=92400, cuLoRADS found a feasible solution 10^{-8} in 15 days, reducing the upper bound to 0.364…
Last week, for k=6, N>=92520, cuLoRADS found a feasible solution 10^{-8} in … days, reducing the upper bound to 0.3637…

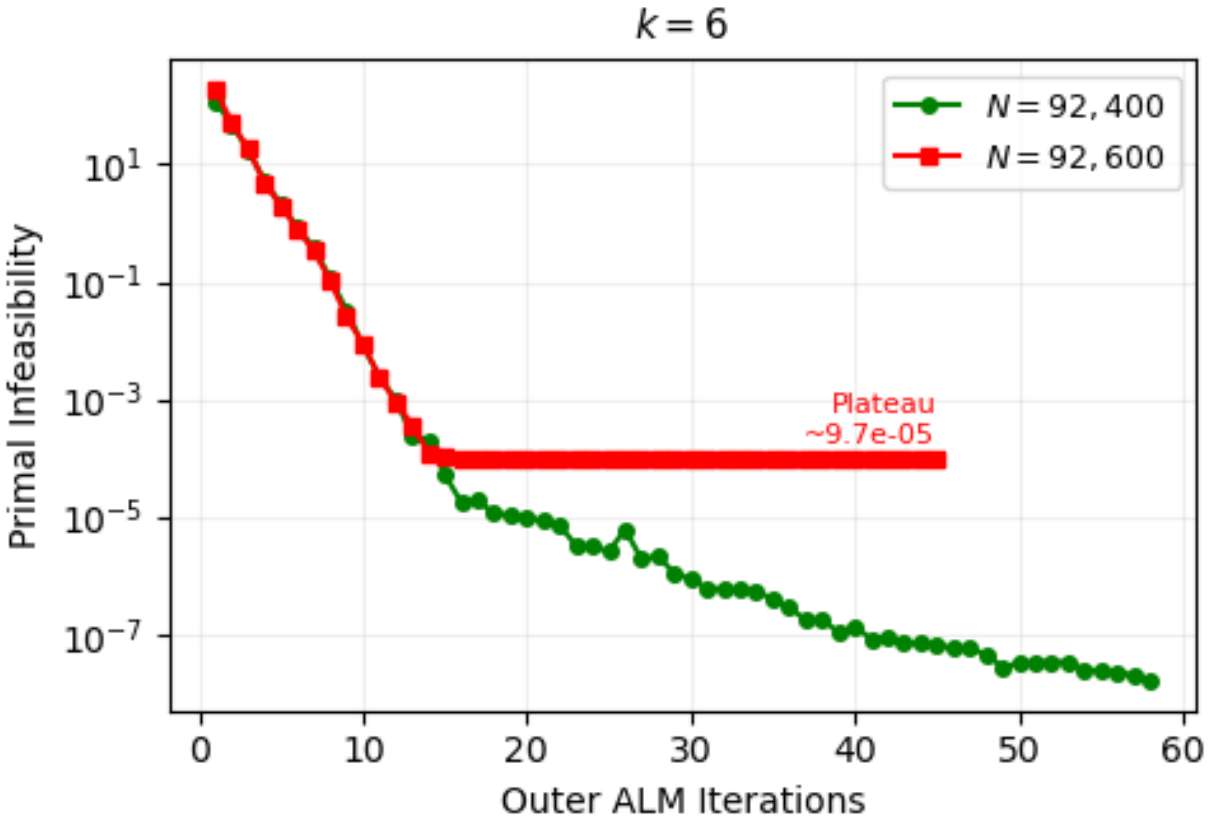
How to "prove" a math theorem using (inexact) numerical Algorithms?

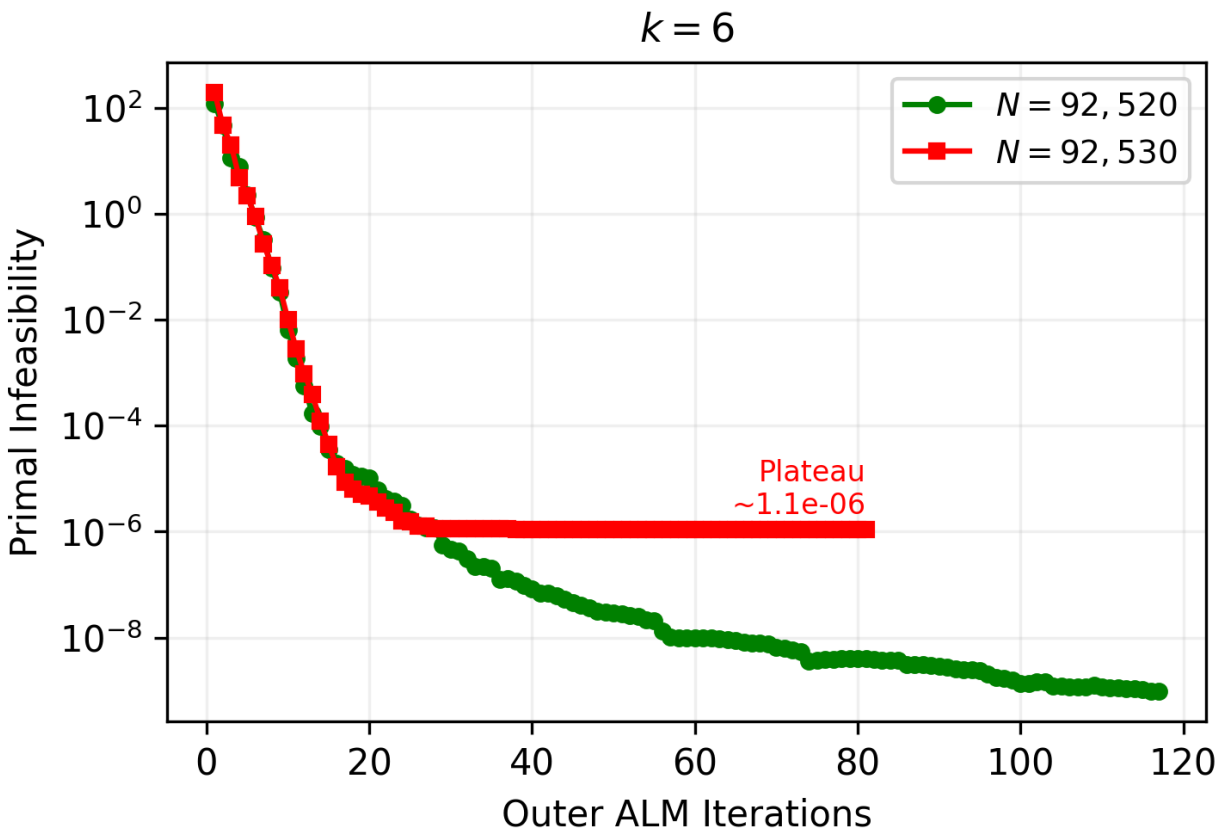
“Proof” based on Algorithmic Convergence Behavior I ?

The solver correctly identifies the convergence phase transition between Feasible (N*) and Infeasible (N* + 1).



“Proof” based on Algorithmic Convergence Behavior II ?

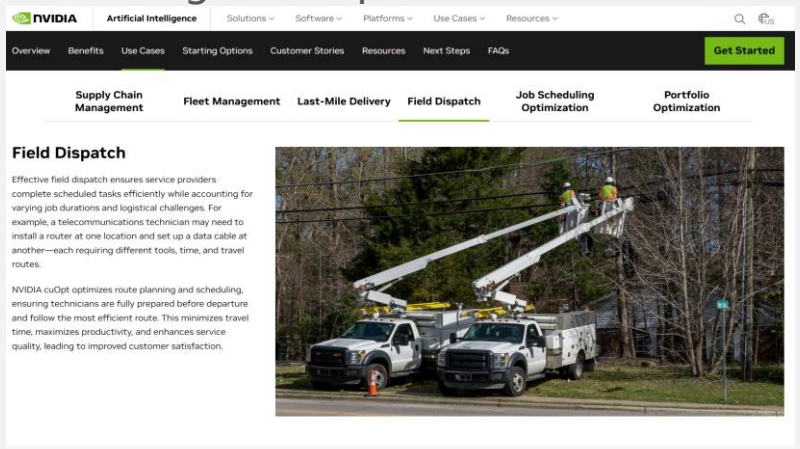


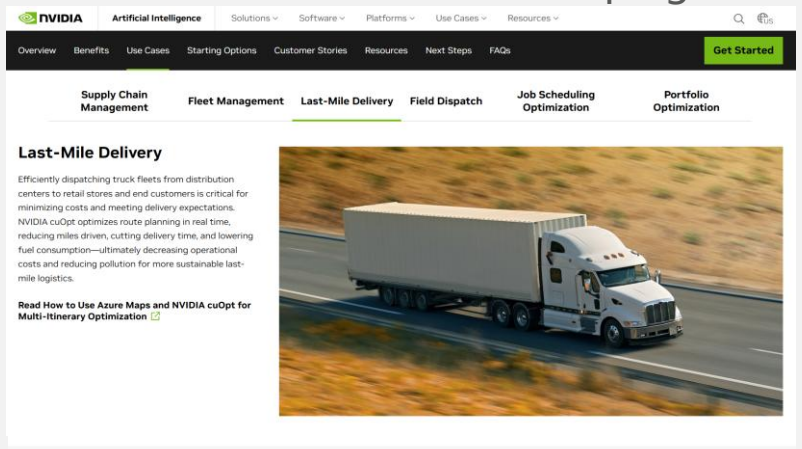


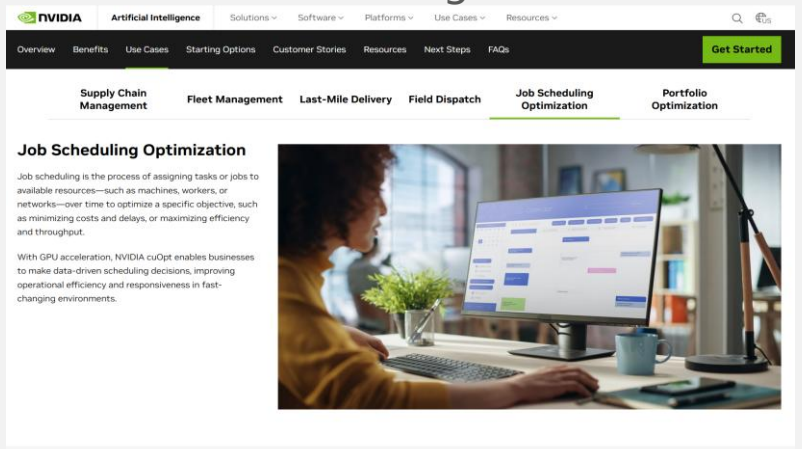
Industry Impact and Follow-up: Nvidia cuOpt

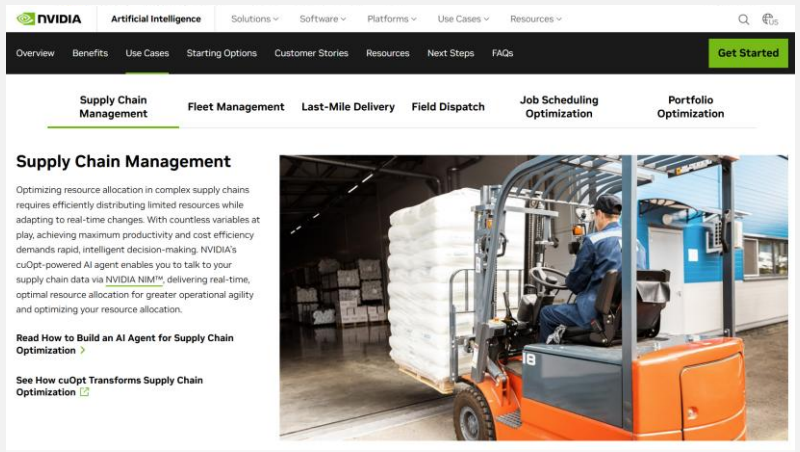
About NVIDIA cuOpt

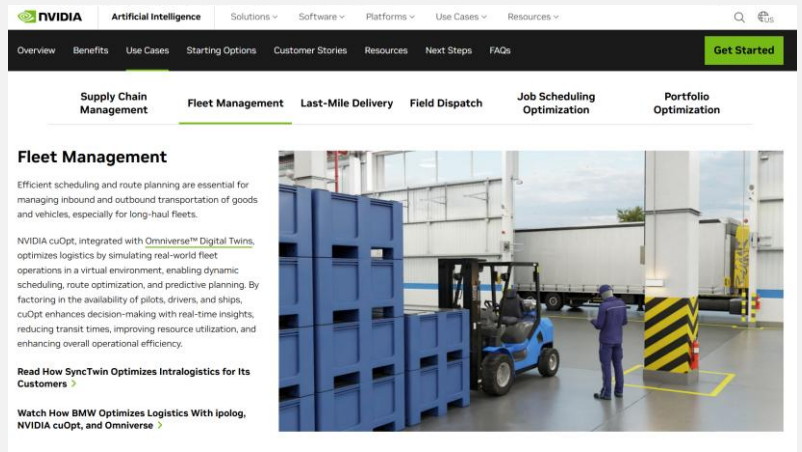
NVIDIA® cuOpt™ is a GPU-accelerated open-source engine for decision optimization that excels in Mixed Integer Programming (MIP), Linear Programming (LP), Vehicle Routing Problems (VRP), and Quadratic Programming (QP). cuOpt solves large-scale problems with millions of variables and constraints, helping to accelerate decision-making.

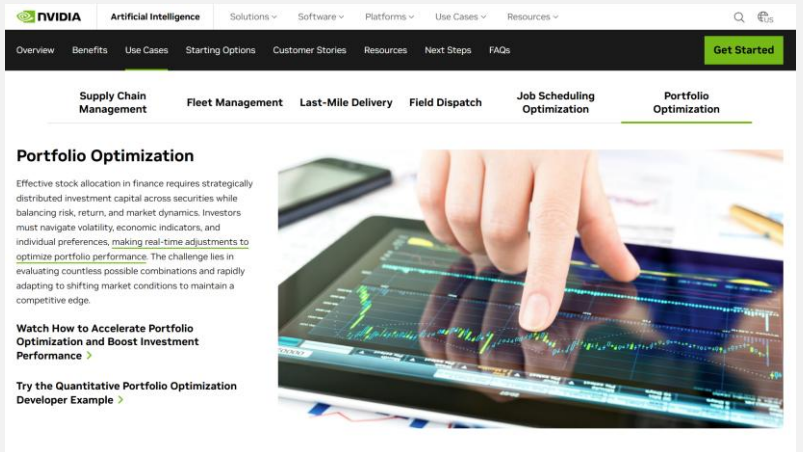












cuOpt Development Trajectory: "Industrial Intelligence" + "Accelerated Computing"

cuPDLP-C Publish 2023

- cuPDLP-C GPU first-order solver by Ge, Lu, and others shows that GPUs can achieve over **100×** speedup over CPU-based solver in high-precision scientific computing.

cuOpt Follow 2024

- NVIDIA cuOpt, announced at GTC, is a native **accelerated computing solver**, forming a productized stack from continuous to integer optimization for **industrial intelligence**.

Ecosystem Development.

- cuOpt is open-sourced and integrated into NVIDIA AI Enterprise / CUDA-X, becoming a foundational operator library for industrial systems such as Palantir.

Decision Foundation

- Extending QP, conic constraints, parallel branching, and PDLP components, aiming to serve as a GPU-native decision optimization foundation.

2023 **2024** **2025** **2026+**

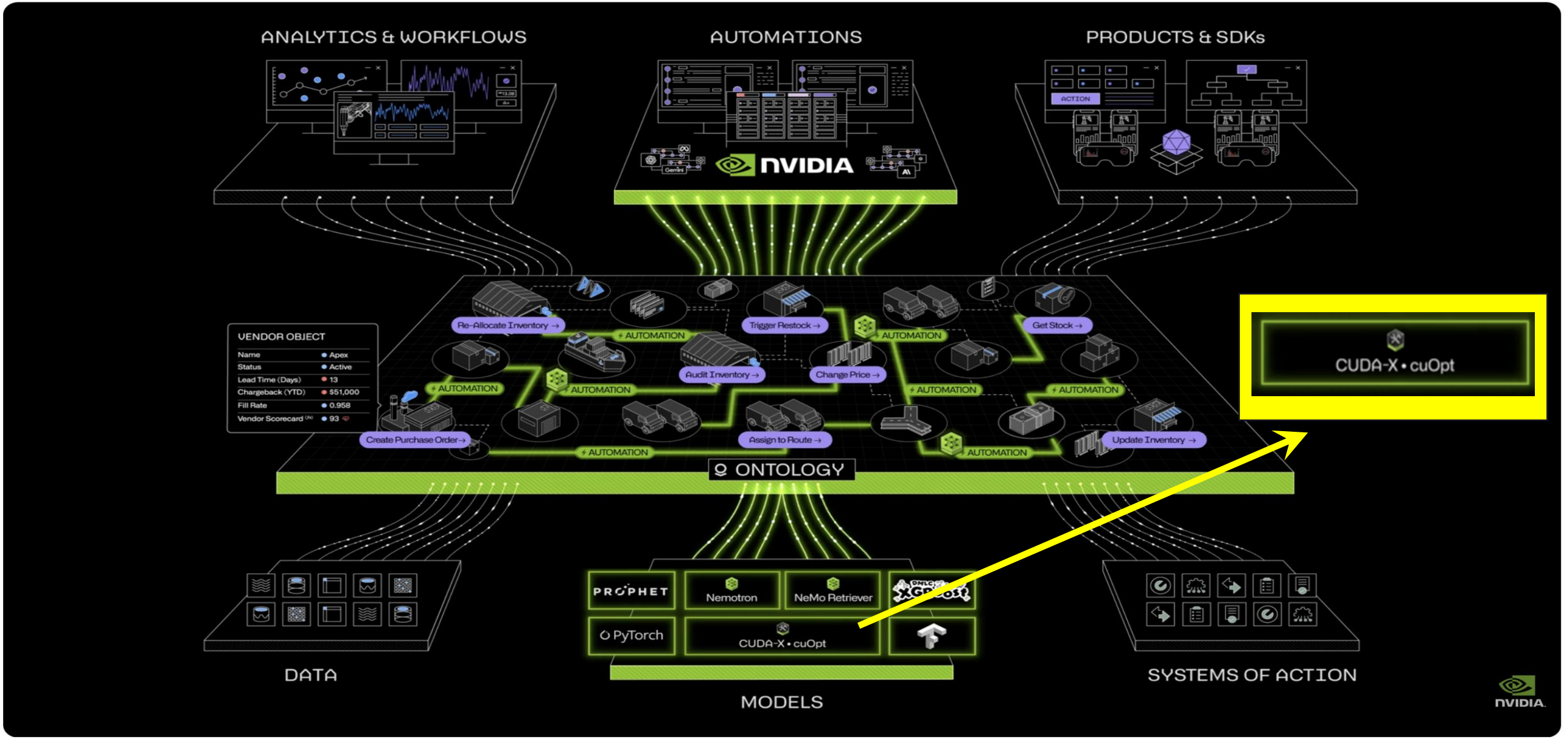
- NVIDIA cuOpt is partly inspired by the 2023 cuPDLP-C work of Ye, Ge, and Lu, with its early codebase and design influenced by cuPDLP-C.
- Over the past three years, it has been a key internal focus at NVIDIA, serving as a core component for **industrial intelligence** and **accelerated computing**.
- It aims to bridge high-precision scientific computing gaps—improving accuracy, reliability, and interpretability beyond conventional AI systems—and is being deployed with partners like Palantir across supply chain, manufacturing, and finance.

cuOpt’s emergence indicates that AI leaders have incorporated optimization solvers into AI competition.

For China to achieve leadership in AI4E and intelligent industrial software, it requires support for...

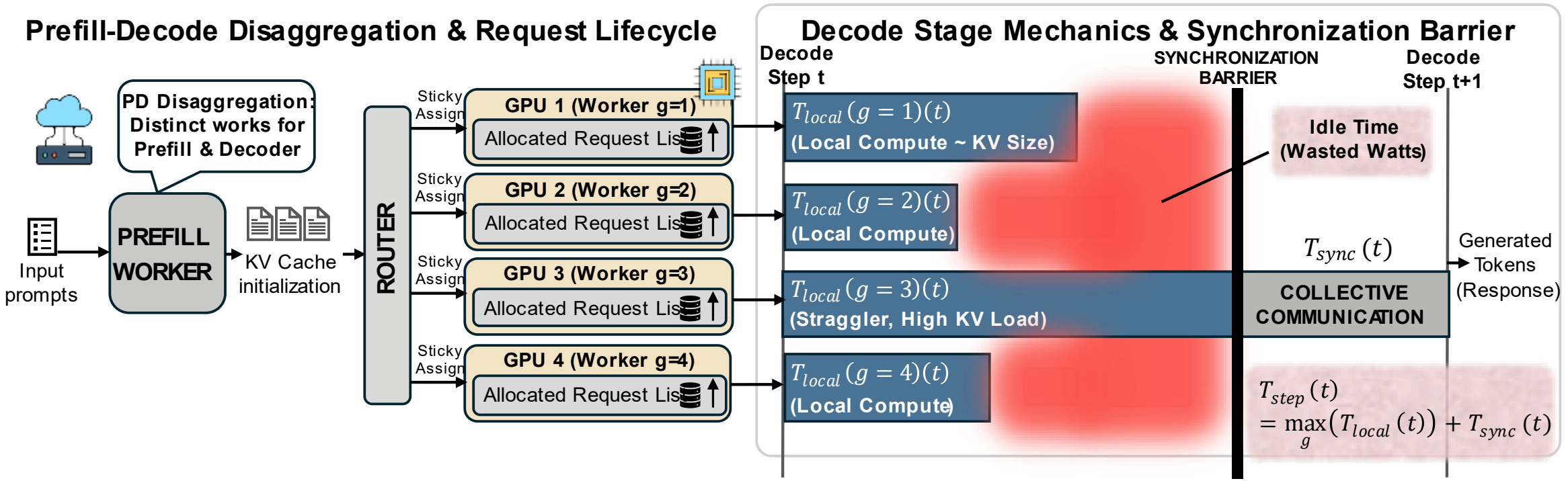
A coordinated stack of GPU ecosystem & optimization solvers & intelligent decision systems.

GTC 2025： Nvidia + Palantir



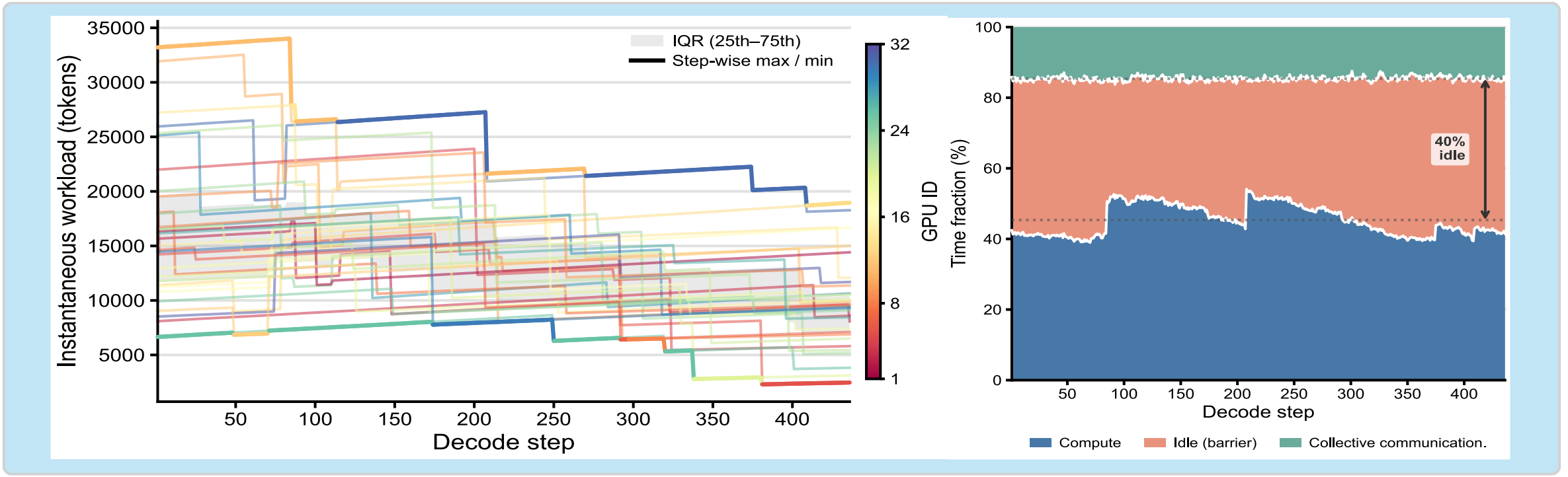
Palantir and NVIDIA Optimize Lowe's Operations

Impact Case: Load Balancing in Large-Scale LLM Serving



- In PD disaggregation, the prefill worker encodes input prompts and initializes each request's KV cache, then hands requests to a router that assigns them to decode workers.
- During each decode step, workers perform local computation with runtime **proportional to** their resident KV load.
- Due to expert/tensor parallelism (often deployed for large scale LLM inference), a **synchronization barrier** enforces that all workers must finish before collective communication can proceed.
- Faster workers **idle** at the barrier (red shaded region).

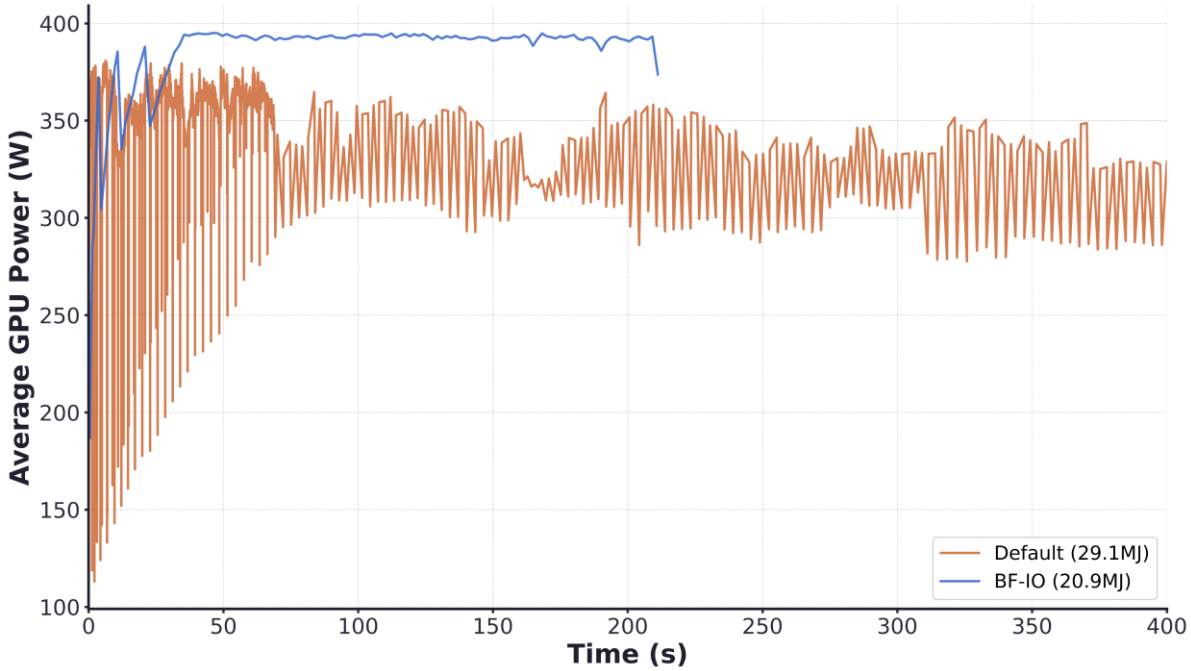
Prompt Scheduling in LLM Inferencing - Idle Time Caused by Load Unbalancing

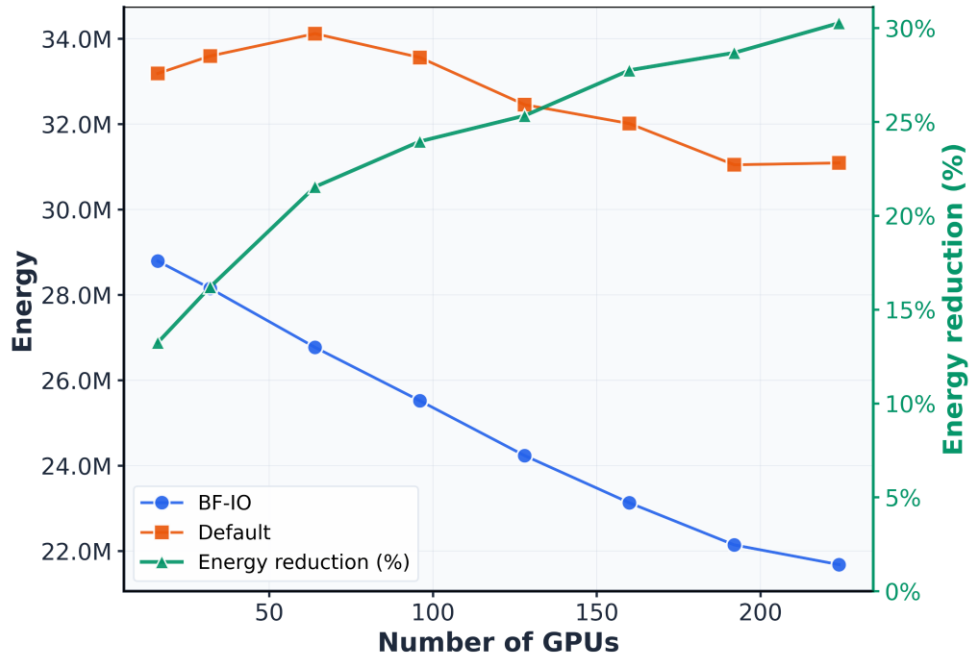


Huawei's Real LLM Serving Trace

- Due to the synchronization barrier, the processing time for each GPU is determined by the one with the heaviest workload (the highest curve in the left plot).
- The orange bars represent the average percentage of time (40%) that GPUs spend waiting during each decode step.

How Much Optimization can Help: Real Case in one of AI Services





- The default policy uses lower power, but it **takes a long time** to process requests due to significant idle time at each step. As a result, the cumulative energy consumption ends up being higher.
- Statistically, load imbalance **worsens as the number of GPUs increases**, making algorithm improvements more impactful in large-scale real production systems with thousands of GPUs.

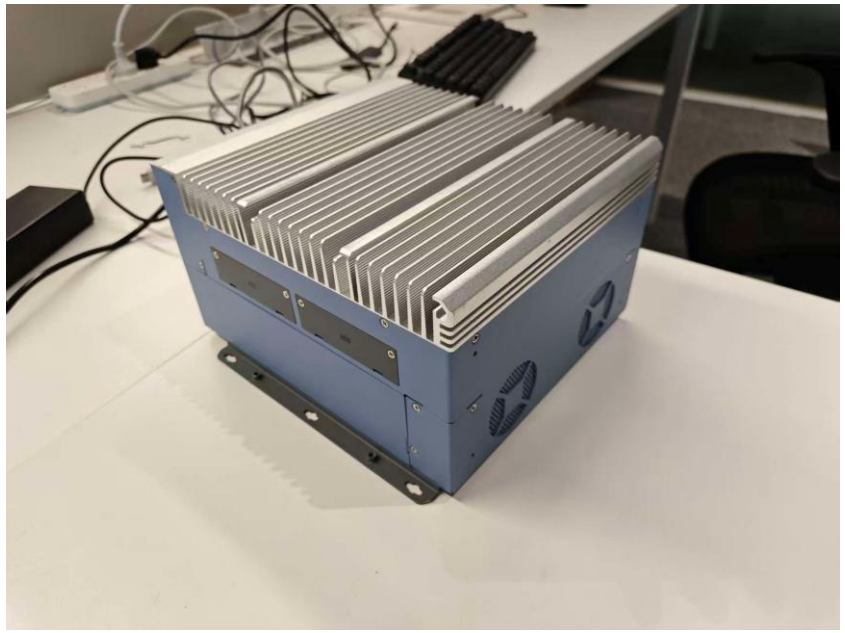
New Paradigm: (Magnetic Quantum) Chips + Algorithms/Solvers

Emerging Computation Hardware/Chip Units:

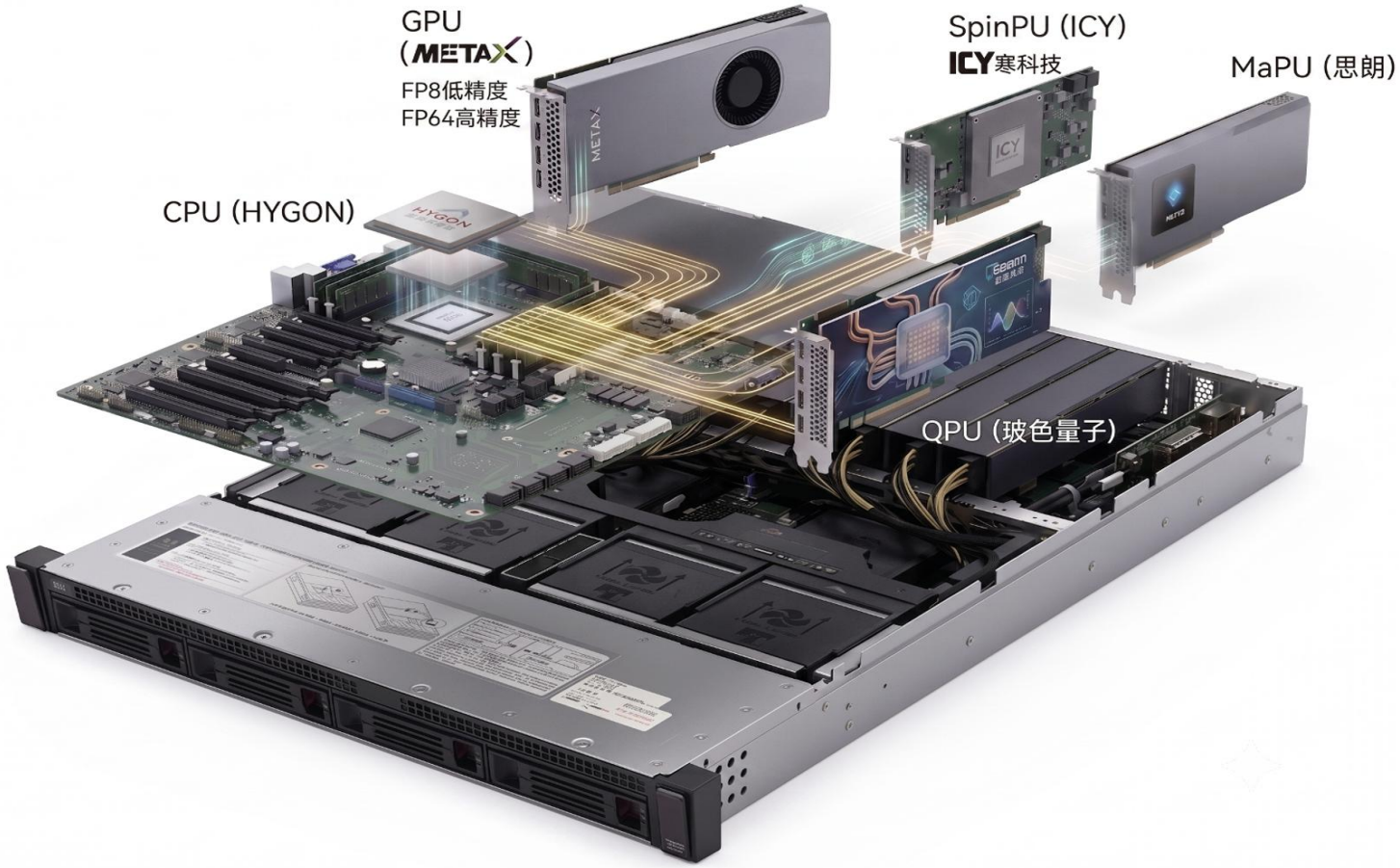
ICY 寒序科技 icy technology SpinPU

QDoson 玻色量子 QPU

- **"Solvable" Problems** $H(\sigma) = -\sum_{\langle ij \rangle} J_{ij}\sigma_i\sigma_j - \sum_j h_j\sigma_j$
  - Ising Problem in Physics
  - QUBO (Quadratic Unconstrained Binary Optimization) problems (transformed into Ising formulation for solution)
- **Our Approach**
  - Warm-Start + Provable Algorithms/Solvers



New Paradigm: Heterogeneous Platforms + Algorithms/Solvers



HYGON 中科海光 CPU/DCU

METAX 沐曦集成电路 GPU

思朗科技 SMART LOGIC MaPU

ICY寒序科技 icy technology SpinPU

QBOSON 玻色量子 QPU

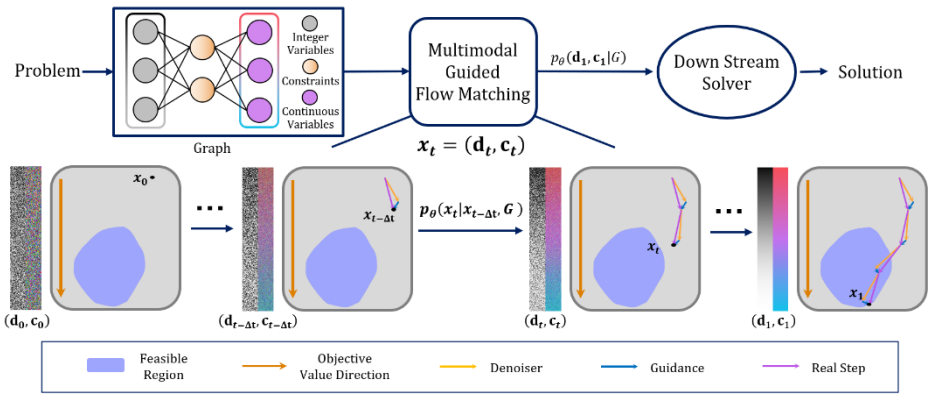
COPT Cardinal Optimizer
Mathematical Programming Solver

cuBLAS	cuDSS	cuSOLVER	cuSPARSE	cuOpt
cuRAND	cuTENSOR	cuFFT	CUDA Math API	AmgX

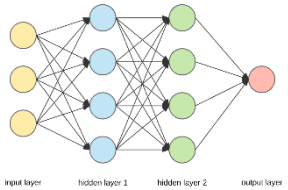
New Paradigm: AI + Algorithms/Solvers (a Showcase for MILP)

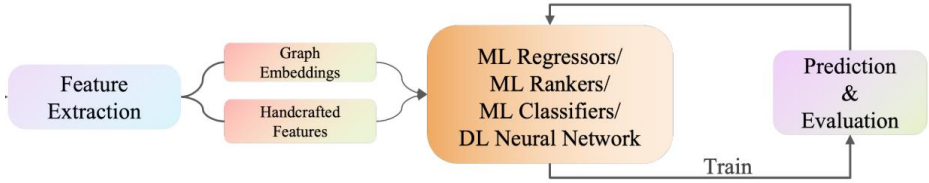
Presolveing

Predictions by diffusion & other AI Models

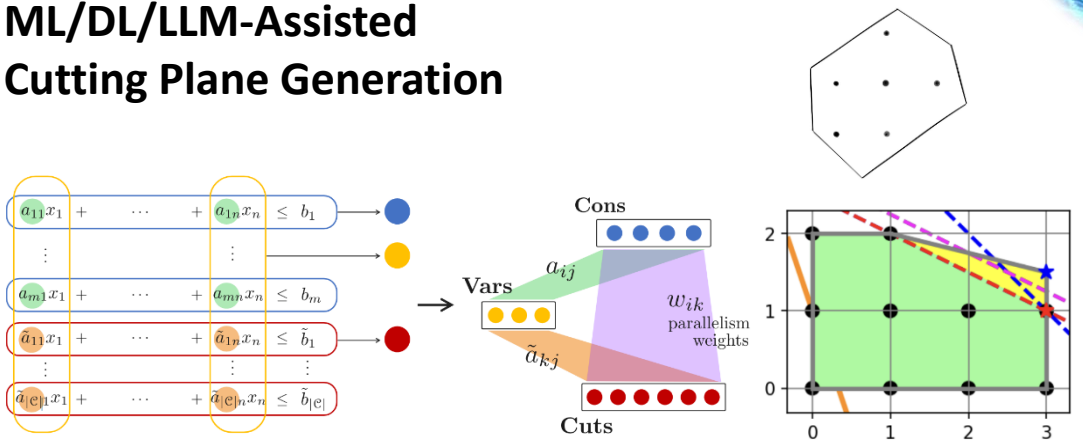


ML-Based or LLM-Based Hyperparameter Tuning

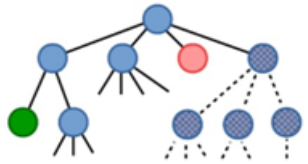


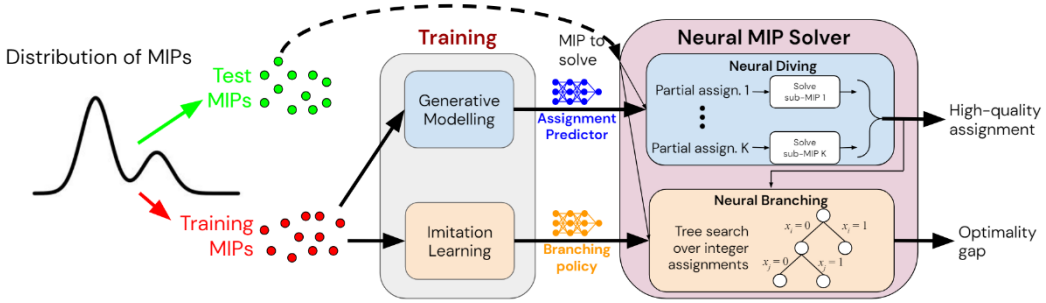


ML/DL/LLM-Assisted Cutting Plane Generation



AI-Guided Branching and Search Path Evolving





Long Live Optimization

THANKS