Primal-Dual Hybrid Gradient Algorithm for Linear Programming

Zaiwen Wen

Beijing International Center For Mathematical Research
Peking University

http://bicmr.pku.edu.cn/~wenzw/bigdata2025.html

Acknowledgement: this slides is Prepared by Zhonglin Xie

Mathematical Formulation: LP Problem

General LP Problem:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & c^\top x \\ \text{s.t.} \quad & \ell_c \leq Ax \leq u_c \\ & \ell_v \leq x \leq u_v \end{aligned}$$

Deriving the Lagrangian:

- Introduce dual variables for constraints to form unconstrained problem
- Rewrite constraints: $Ax - u_c \leq 0$, $\ell_c - Ax \leq 0$, $x - u_v \leq 0$, $\ell_v - x \leq 0$
- Associate non-negative multipliers y^-, y^+, r^-, r^+ with each inequality

Lagrangian Function:

$$\mathcal{L}(x, y^-, y^+, r^-, r^+) = c^\top x + (y^-)^\top (Ax - u_c) + (y^+)^\top (\ell_c - Ax) \tag{1}$$

$$+ (r^-)^\top (x - u_v) + (r^+)^\top (\ell_v - x) \tag{2}$$

where all multipliers are non-negative

Deriving the Dual Problem

Derivation: Group x terms and form the dual function

$$\mathcal{L}(x, y^-, y^+, r^-, r^+) = x^\top (c + A^\top (y^- - y^+) + (r^- - r^+)) \tag{3}$$
$$- (y^-)^\top u_c + (y^+)^\top \ell_c - (r^-)^\top u_v + (r^+)^\top \ell_v \tag{4}$$

The minimum over x is $-\infty$ unless $c + A^\top (y^- - y^+) + (r^- - r^+) = 0$

Dual Problem: Using substitutions $y = y^+ - y^-$ and $r = r^+ - r^-$

$$\begin{aligned} \max_{y \in \mathbb{R}^m, r \in \mathbb{R}^n} \quad & - (y^-)^\top u_c + (y^+)^\top \ell_c - (r^-)^\top u_v + (r^+)^\top \ell_v \\ \text{s.t.} \quad & c - A^\top y = r \\ & y^-, y^+, r^-, r^+ \geq 0 \end{aligned}$$

Dual Problem Formulation

Simplified notation: Define $p(y; \ell, u) := u^\top y^+ - \ell^\top y^-$ where $y^+ = \max(y, 0)$ and $y^- = \max(-y, 0)$

Rewriting the dual:

$$\begin{aligned} \max_{y \in \mathbb{R}^m, r \in \mathbb{R}^n} \quad & -p(-y; \ell_c, u_c) - p(-r; \ell_v, u_v) \\ \text{s.t.} \quad & c - A^\top y = r \\ & y \in \mathcal{Y},\ r \in \mathcal{R} \end{aligned}$$

Dual feasible sets $\mathcal{Y}$ and $\mathcal{R}$: Based on constraint types

$$\mathcal{Y}_i := \begin{cases} \{0\} & (\ell_c)_i = -\infty, (u_c)_i = \infty \text{ (unconstrained)} \\ \mathbb{R}^- & (\ell_c)_i = -\infty, (u_c)_i \in \mathbb{R} \text{ (upper bound)} \\ \mathbb{R}^+ & (\ell_c)_i \in \mathbb{R}, (u_c)_i = \infty \text{ (lower bound)} \\ \mathbb{R} & \text{otherwise (both upper and lower bounds)} \end{cases} \quad \mathcal{R}_i := \begin{cases} \{0\} & (\ell_v)_i = -\infty, (u_v)_i = \infty \\ \mathbb{R}^- & (\ell_v)_i = -\infty, (u_v)_i \in \mathbb{R} \\ \mathbb{R}^+ & (\ell_v)_i \in \mathbb{R}, (u_v)_i = \infty \\ \mathbb{R} & \text{otherwise} \end{cases}$$

Saddle Point Formulation of LP

- Keeping the bounds on x, we obtain the Lagrangian function:

$$\mathcal{L}(x, y^-, y^+) = c^\top x + (y^-)^\top (Ax - u_c) + (y^+)^\top (\ell_c - Ax)$$

- Using the notation $y = y^+ - y^-$ and the function $p(y; \ell, u) = u^\top y^+ - \ell^\top y^-$:

$$\begin{aligned}(y^-)^\top (Ax - u_c) + (y^+)^\top (\ell_c - Ax) &= -((y^-)^\top u_c - (y^+)^\top \ell_c) + (y^- - y^+)^\top (Ax) \\ &= -p(-y; \ell_c, u_c) - y^\top (Ax)\end{aligned}$$

- Then the saddle point problem is:

$$\min_x \max_y \{c^\top x + y^\top (Ax) - p(-y; \ell_c, u_c)\} \quad \text{s.t. } \ell_v \le x \le u_v$$

Final saddle point formulation:

$$\min_{x \in \mathcal{X}} \max_{y \in \mathcal{Y}} \mathcal{L}(x, y) := c^\top x + y^\top Ax - p\,(y; -u_c, -\ell_c)$$

where $\mathcal{X} := \{x \in \mathbb{R}^n : \ell_v \le x \le u_v\}$

Problems with Classical Solvers for Large-Scale LP

- **Simplex Method:**
  - Iterations potentially exponential in problem size
  - Poor parallelization on modern hardware
- **Interior Point Methods (IPMs):**
  - Memory requirements: $O(nnz(A))$ for matrix factorization
  - Often exceeds 1TB for problems with billions of nonzeros
- **First-Order Methods:**
  - Low memory requirements
  - Highly parallelizable matrix-vector operations
  - But historically struggle with achieving high accuracy
  - Small constraint violations can lead to significant errors

First-Order Methods for LP

- **State-of-the-art First-Order Method Solvers:**
  - SCS: ADMM-based solver with homogeneous self-dual embedding
  - OSQP: ADMM-based for convex quadratic programming
  - ECLIPSE: Gradient descent on smoothed dual formulation
  - ABIP/ABIP+: Interior-point solvers using ADMM
- **PDHG (Primal-Dual Hybrid Gradient) Advantages:**
  - Requires only matrix-vector products: Ax and $A^\top y$
  - No matrix factorization or systems of equations
  - Form of operator splitting (related to ADMM)
  - Linear convergence for LP established in theory

Generic Convex-Concave Saddle Point Problems

General Form:

$$\min_{x \in \mathcal{X}} \max_{y \in \mathcal{Y}} \mathcal{L}(x, y) = \langle Kx, y \rangle + g(x) - f^*(y)$$

Key components:

- K: Linear operator (matrix) mapping primal to dual space
- $g(x)$: Convex function (often includes constraints on x)
- $f^*(y)$: Convex conjugate of function f (handles dual constraints)

Convex Conjugate Definition: For any convex function f

$$f^*(y) = \sup_x \{\langle x, y \rangle - f(x)\}$$

This transforms constraints into penalties in the optimization

PDHG: Abstract Form

Primal-Dual Hybrid Gradient Algorithm:

$$x^{k+1} = \text{prox}_{\tau g}(x^k - \tau K^* y^k) \tag{5}$$

$$y^{k+1} = \text{prox}_{\sigma f^*}(y^k + \sigma K(2x^{k+1} - x^k)) \tag{6}$$

Proximal Operator: A generalization of projection

$$\text{prox}_{\tau g}(z) = \arg\min_x \left\{ g(x) + \frac{1}{2\tau}\|x - z\|^2 \right\}$$

Moreau Decomposition: Allows computing proximal operator of f^* using f

$$\text{prox}_{\sigma f^*}(y) = y - \sigma \cdot \text{prox}_{f/\sigma}(y/\sigma)$$

This is crucial for implementing PDHG efficiently without explicitly forming the conjugate function!

Applying PDHG to LP Problems

For LP saddle point problem:

$$\min_{x \in \mathcal{X}} \max_{y \in \mathcal{Y}} \mathcal{L}(x, y) := c^\top x + y^\top A x - p\left(y; -u_c, -\ell_c\right)$$

We identify:

- $K = A$ (linear constraint matrix)
- $g(x) = c^\top x + \delta_{\mathcal{X}}(x)$ (objective + variable bounds)
- $f^*(y) = p(y; -u_c, -\ell_c)$ (constraint bounds)

Computing proximal operators:

$$\text{prox}_{\tau g}(z) = \text{proj}_{\mathcal{X}}(z - \tau c) \tag{7}$$

$$\text{prox}_{\sigma f^*}(y) = y - \sigma \cdot \text{proj}_{[-u_c, -\ell_c]}(y/\sigma) \tag{8}$$

where projections enforce the constraints efficiently

PDHG Algorithm for LP

PDHG iterations for LP:

$$x^{k+1} = \text{proj}_{[\ell_v, u_v]}(x^k - \tau(c + A^\top y^k)) \tag{9}$$

$$\tilde{y}^{k+1} = y^k + \sigma A(2x^{k+1} - x^k) \tag{10}$$

$$y^{k+1} = \tilde{y}^{k+1} - \sigma \text{proj}_{[-u_c, -\ell_c]}(\tilde{y}^{k+1}/\sigma) \tag{11}$$

Key benefits:

- Only requires matrix-vector products (Ax and $A^\top y$)
- Projections computed element-wise (highly parallelizable)
- No matrix factorization or linear systems to solve
- Memory-efficient for very large-scale problems

Convergence Theory for PDHG on LP

Step Size Parameterization:

- $\tau = \eta/\omega$ and $\sigma = \omega\eta$ with $\eta \in (0, \infty), \omega \in (0, \infty)$
- Convergence guaranteed when $\eta < 1/\|A\|_2$
- ω: primal weight, controls scaling between primal and dual iterates

Special Norm for Convergence Analysis:

$$\|z\|_\omega := \sqrt{\omega\|x\|_2^2 + \frac{\|y\|_2^2}{\omega}}$$

for $z = (x, y)$ - Used in convergence theory, restart criteria, and primal-dual balance

Linear convergence: Under certain conditions, PDHG converges linearly for LP:

$$\|z^k - z^*\|_\omega \leq C(1 - \gamma)^k$$

where $\gamma \in (0, 1)$ depends on problem structure

Main Algorithm

Algorithm PDLP (Main Structure)

Input: Initial solution $z^{0,0}$
Initialize outer loop counter $n \leftarrow 0$, total iterations $k \leftarrow 0$
Initialize step size $\hat{\eta}^{0,0} \leftarrow 1/\|A\|_\infty$, primal weight $\omega^0 \leftarrow$ `InitializePrimalWeight`
repeat
  $t \leftarrow 0$ {Inner restart loop counter}
  repeat
    $z^{n,t+1}, \eta^{n,t+1}, \hat{\eta}^{n,t+1} \leftarrow$ `AdaptiveStepOfPDHG`$(z^{n,t}, \omega^n, \hat{\eta}^{n,t}, k)$
    $\bar{z}^{n,t+1} \leftarrow \frac{1}{\sum_{i=1}^{t+1} \eta^{n,i}} \sum_{i=1}^{t+1} \eta^{n,i} z^{n,i}$ {Step-size weighted average}
    $z_c^{n,t+1} \leftarrow$ `GetRestartCandidate`$(z^{n,t+1}, \bar{z}^{n,t+1}, z^{n,0})$
    $t \leftarrow t+1$, $k \leftarrow k+1$
  until restart or termination criteria holds
  restart the outer loop: $z^{n+1,0} \leftarrow z_c^{n,t}$, $n \leftarrow n+1$
  $\omega^n \leftarrow$ `PrimalWeightUpdate`$(z^{n,0}, z^{n-1,0}, \omega^{n-1})$
until termination criteria holds
Output: $z^{n,0}$

PDLP Key Improvements Overview

- **Adaptive step sizes:** Dynamic adjustment based on convergence conditions
- **Restart strategies:** Reset algorithm when progress slows
- **Primal weight updates:** Balance progress in primal and dual spaces
- **Special Norm for Convergence Analysis:**

$$\|z\|_\omega := \sqrt{\omega\|x\|_2^2 + \frac{\|y\|_2^2}{\omega}}$$

for $z = (x, y)$ - Used in convergence theory, restart criteria, and primal-dual balance

Adaptive Step Size

Traditional PDHG: Fixed step size $\eta = \frac{1}{\|A\|_2}$

- Overly pessimistic
- Requires estimation of $\|A\|_2$

PDLP Approach: Adaptive step size based on convergence condition

- Calculate maximum allowable step size:

$$\bar{\eta} = \frac{\|z^{k+1} - z^k\|_\omega^2}{2(y^{k+1} - y^k)^\top A(x^{k+1} - x^k)}$$

- This ensures the iteration remains convergent

Adaptive Step Size Algorithm

Algorithm One step of PDHG with adaptive step size

function `AdaptiveStepOfPDHG`$(z^{n,t}, \omega^n, \hat{\eta}^{n,t}, k)$
$(x, y) \leftarrow z^{n,t}$, $\eta \leftarrow \hat{\eta}^{n,t}$
for $i = 1, \ldots, \infty$ **do**
  $x' \leftarrow \text{proj}_{\mathcal{X}}(x - \frac{\eta}{\omega^n}(c - A^\top y))$
  $y' \leftarrow y - A(2x' - x) - \eta\omega^n \text{proj}_{[\ell_c, u_c]}\left((\eta\omega^n)^{-1} y - A(2x' - x)\right)$
  $\bar{\eta} \leftarrow \frac{\|(x'-x, y'-y)\|^2_{\omega^n}}{2(y'-y)^\top A(x'-x)}$
  $\eta' \leftarrow \min\left((1 - (k+1)^{-0.3})\bar{\eta}, (1 + (k+1)^{-0.6})\eta\right)$
  if $\eta \leq \bar{\eta}$ **then**
    return $(x', y'), \eta, \eta'$
  end if
  $\eta \leftarrow \eta'$
end for

Key Properties:

- Guarantee convergence: Only accept step if $\eta \leq \bar{\eta}$
- Aggressive adaptation: Next step size η' can grow up to factor of $(1 + (k+1)^{-0.6})$

Normalized Duality Gap and Adaptive Restarts

Normalized Duality Gap Definition:

$$\rho_r^n(z) := \frac{1}{r} \max_{\|\hat{z}-z\|_\omega \le r} \{\mathcal{L}(x, \hat{y}) - \mathcal{L}(\hat{x}, y)\}$$

where $\mathcal{L}(x, y) = c^\top x + y^\top A x - p(y; -u_c, -\ell_c)$ is the Lagrangian.

Key Properties:

- Always finite (bounded by search radius)
- Zero if and only if solution is optimal
- Provides a meaningful measure of progress toward optimality

Notation: $\mu_n(z, z_{\text{ref}}) := \rho^n_{\|z-z_{\text{ref}}\|_{\omega^n}}(z)$, where z_{ref} is a user-chosen reference point (typically start of current restart)

Adaptive Restart Criteria

PDLP Parameters:

$$\beta_{\text{necessary}} = 0.9, \quad \beta_{\text{sufficient}} = 0.1, \quad \beta_{\text{artificial}} = 0.5$$

Restart triggered when any of these criteria hold:

- **Sufficient decay:**

  $$\mu_n(z_c^{n,t+1}, z^{n,0}) \leq \beta_{\text{sufficient}} \mu_n(z^{n,0}, z^{n-1,0})$$

  Guarantees linear convergence on LP problems

- **Necessary decay + no progress:**

  $$\mu_n(z_c^{n,t+1}, z^{n,0}) \leq \beta_{\text{necessary}} \mu_n(z^{n,0}, z^{n-1,0})$$
  $$\text{and} \quad \mu_n(z_c^{n,t+1}, z^{n,0}) > \mu_n(z_c^{n,t}, z^{n,0})$$

  Detects when progress begins to stall

- **Long inner loop:** $t \geq \beta_{\text{artificial}} k$

Restart Mechanism - Implementation Details

Restart Candidate Selection:

$$\text{GetRestartCandidate}(z^{n,t+1}, \bar{z}^{n,t+1}, z^{n,0}) = \begin{cases} z^{n,t+1} & \text{if } \mu_n(z^{n,t+1}, z^{n,0}) < \mu_n(\bar{z}^{n,t+1}, z^{n,0}) \\ \bar{z}^{n,t+1} & \text{otherwise} \end{cases}$$

Implementation Note:

- Restart criteria evaluated every 64 iterations to reduce overhead
- Makes minimal impact on total iteration count
- Running averages $\bar{z}^{n,t+1}$ weighted by step sizes

Primal Weight Updates

Motivation: Balance progress in primal and dual spaces

- For optimal convergence, we want equal progress in both spaces:

$$\|(x^{n,t} - x^*, \mathbf{0})\|_{\omega^n} = \|(\mathbf{0}, y^{n,t} - y^*)\|_{\omega^n}$$

- This yields the ideal primal weight:

$$\omega^n = \frac{\|y^{n,t} - y^*\|_2}{\|x^{n,t} - x^*\|_2}$$

Implementation:

- Estimate using consecutive iterates:

$$\Delta_x^n = \|x^{n,0} - x^{n-1,0}\|_2, \quad \Delta_y^n = \|y^{n,0} - y^{n-1,0}\|_2$$

- Apply log-scale exponential smoothing:

$$\omega^n = \exp\left(\theta \log\left(\frac{\Delta_y^n}{\Delta_x^n}\right) + (1-\theta)\log(\omega^{n-1})\right)$$

Primal Weight Update Algorithm

Algorithm Primal weight update

function `PrimalWeightUpdate`$(z^{n,0}, z^{n-1,0}, \omega^{n-1})$
$\Delta_x^n = \|x^{n,0} - x^{n-1,0}\|_2$, $\Delta_y^n = \|y^{n,0} - y^{n-1,0}\|_2$
if $\Delta_x^n > \varepsilon_0$ and $\Delta_y^n > \varepsilon_0$ **then**
    return $\exp\left(\theta \log\left(\frac{\Delta_y^n}{\Delta_x^n}\right) + (1-\theta)\log(\omega^{n-1})\right)$
else
    return ω^{n-1}
end if

Key innovation: Updates only occur after restarts

- Allows larger weight changes without causing instability
- Focuses on balancing distance traveled rather than residuals
- Significantly improves performance compared to per-iteration updates

GPU vs. CPU Architecture

CPU Design:

- Few cores (16-64) with deep pipelines
- Optimized for sequential processing
- Sophisticated branch prediction
- Limited memory bandwidth (100-200 GB/sec)

GPU Design:

- Thousands of cores (7296 on NVIDIA H100)
- Single Instruction Multiple Data (SIMD)
- Optimized for parallel computation
- Very high memory bandwidth (2 TB/sec)

Why GPUs for LP?

- Previous attempts failed with simplex/IPM methods
- First-order methods like PDHG rely on matrix-vector operations
- PDLP's core operations highly parallelizable

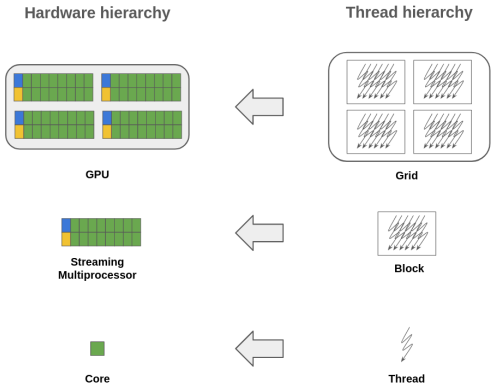
GPU Thread Hierarchy and Execution Model

Thread Hierarchy:

- **Thread:** Basic execution unit
- **Warp:** 32 threads executing in lockstep
- **Block:** Group of threads with shared memory
- **Grid:** Collection of blocks executing same kernel

Implications for PDLP:

- Matrix-vector operations highly parallelizable
- Each thread can process individual vector elements
- Challenge: Reducing CPU-GPU communication overhead



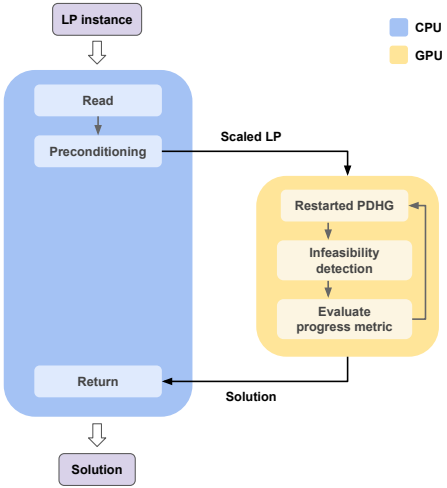
cuPDLP.jl: Design Principles

Minimizing CPU-GPU Communication:

- Initial transfer: Problem instance from CPU to GPU
- Final transfer: Solution from GPU to CPU
- All iterations computed entirely on GPU

Implementation Framework:

- Implemented in Julia using CUDA.jl
- Custom CUDA kernels for PDHG updates
- cuSPARSE library for sparse matrix operations



Key Acceleration Points

Matrix and Vector Operations:

- Sparse matrix stored in Compressed Sparse Row (CSR) format
- Matrix-vector multiplication via cuSPARSE library
- Custom CUDA kernels for vector operations and projections
- One thread per vector element for maximum throughput

KKT-Based Restart Scheme:

- Original PDLP: Trust-region algorithm for normalized duality gap
  - Sequential nature - poor fit for GPU architecture
- cuPDLP.jl innovation: KKT error-based restart
  - Highly parallelizable computation
  - Maintains convergence properties

KKT-Based Restart Details

KKT Error Definition:

$$\mathrm{KKT}_\omega(z) = \sqrt{\omega^2 \left\| \begin{pmatrix} Ax - b \\ [h - Gx]^+ \end{pmatrix} \right\|_2^2 + \frac{1}{\omega^2} \|c - K^\top y - \lambda\|_2^2 + (q^\top y + l^\top \lambda^+ - u^\top \lambda^- - c^\top x)^2}$$

Restart Candidate Selection:

$$z_c^{n,t+1} = \begin{cases} z^{n,t+1} & \text{if } \mathrm{KKT}_{\omega^n}(z^{n,t+1}) < \mathrm{KKT}_{\omega^n}(\bar{z}^{n,t+1}) \\ \bar{z}^{n,t+1} & \text{otherwise} \end{cases}$$

Restart Conditions: Algorithm restarts if any of these holds:

- **Sufficient decay:** $\mathrm{KKT}_{\omega^n}(z_c^{n,t+1}) \leq 0.2 \cdot \mathrm{KKT}_{\omega^n}(z^{n,0})$
- **Necessary decay + no progress:** $\mathrm{KKT}_{\omega^n}(z_c^{n,t+1}) \leq 0.8 \cdot \mathrm{KKT}_{\omega^n}(z^{n,0})$ and no improvement
- **Long inner loop:** Iteration count exceeds threshold

Primal and Dual Updates on GPU

Primal Update CUDA Kernel:

- **Input:** x^k, y^k, c, A, τ, lower/upper bounds
- **Parallel operations:**
  - Matrix-vector product: $A^\top y^k$ (via cuSPARSE)
  - Vector addition: $c - A^\top y^k$
  - Projection onto bounds: $\text{proj}_{\mathcal{X}}(x^k - \tau(c - A^\top y^k))$

Dual Update CUDA Kernel:

- **Input:** x^{k+1}, x^k, y^k, A, σ, constraint bounds
- **Parallel operations:**
  - Extrapolation: $2x^{k+1} - x^k$
  - Matrix-vector product: $A(2x^{k+1} - x^k)$ (via cuSPARSE)
  - Projection onto bounds for constraint relaxation
  - Final dual update computation

Performance vs. Gurobi: Moderate Accuracy (10^{-4})

	Small (269)		Medium (94)		Large (20)		Total (383)	
	Count	**Time**	**Count**	**Time**	**Count**	**Time**	**Count**	**Time**
cuPDLP.jl	266	8.61	92	14.80	19	111.19	377	12.02
Primal simplex	268	12.56	69	188.81	11	3145.49	348	39.81
Dual simplex	268	8.75	84	66.67	15	591.63	367	21.75
Barrier	268	5.30	88	45.01	18	415.78	374	14.92

Key Observations:

- cuPDLP.jl solves 377/383 instances (98.4%)
- Clear advantage on medium and large instances:
  - 3x faster than simplex on medium instances
  - 3.7x faster than barrier on large instances
- Especially strong for problems with complex structures

Performance vs. CPU PDLP

	Small (269)		Medium (94)		Large (20)		Total (383)	
	Count	**Time**	**Count**	**Time**	**Count**	**Time**	**Count**	**Time**
cuPDLP.jl	266	8.61	92	14.80	19	111.19	377	12.02
FirstOrderLp.jl	253	35.94	82	155.67	12	2002.21	347	66.67
PDLP (1 thread)	256	22.69	85	98.38	15	1622.91	356	43.81
PDLP (4 threads)	260	24.03	91	42.94	15	736.20	366	34.57
PDLP (16 threads)	238	104.72	84	142.79	15	946.24	337	127.49

GPU Speedup vs. CPU:

- vs. FirstOrderLp.jl: 4x on small, 10x on medium, 18x on large instances
- vs. PDLP with 4 threads: 2.9x overall speedup
- Solved 30 more instances than FirstOrderLp.jl at tolerance 10^{-4}
- Speedup increases with problem size