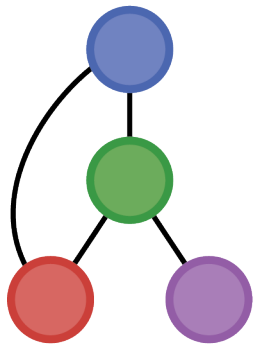
UCONN
UNIVERSITY OF CONNECTICUT

Deterministic Global Optimization Using GPUs for the Lower-Bounding Problem

Dimitri Alston, Robert Gottlieb
Matthew Stuber, P&W Associate Professor in Advanced Systems Engineering

June 3rd, 2026

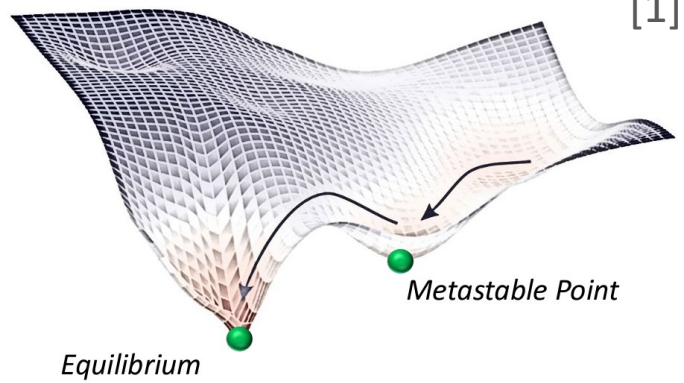


Process Systems and Operations Research Laboratory

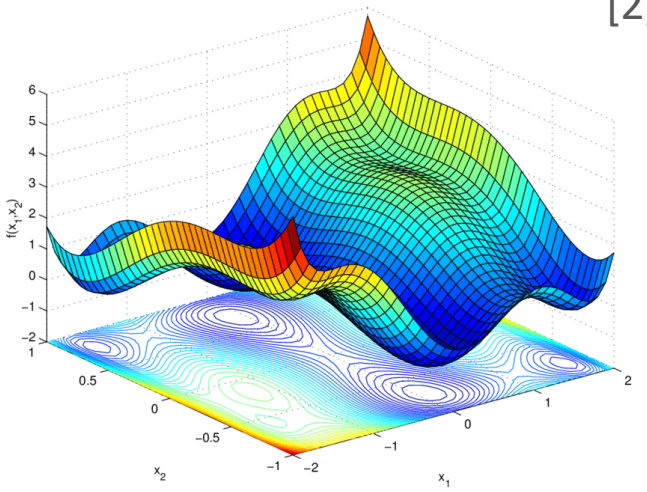
Nonconvex Optimization

- Nonconvex problems are widespread in practice
- Convex solution methods may miss the global optimum
  - Multiple feasible solutions may exist
- **Deterministic global optimization** guarantees optimality or infeasibility
  - Techniques for nonconvex problems are more expensive

[1]



[2]



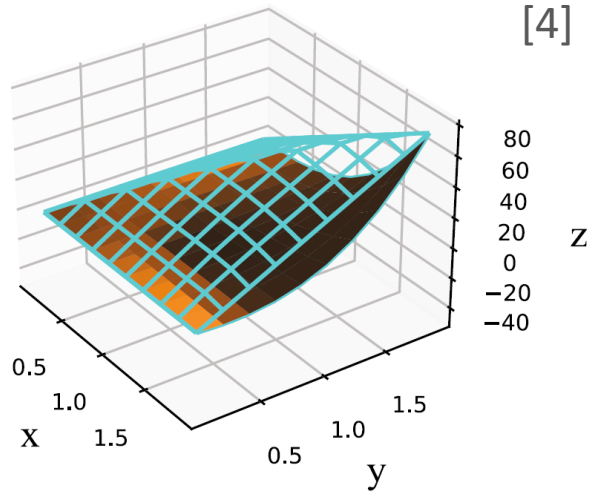
[1] Grajcarova, L. Simulations of structural phase transitions in crystals using ab initio metadynamics. INIS-IAEA. (2013).
[2] Henrion, D. and Lasserre, J.-B. GloptiPoly: global optimization over polynomials with MATLAB and SeDuMi. Proceedings of the 41st IEEE Conference on Decision and Control. 1(2): 747-752 (2002).

EAGO.jl

Easy Advanced Global Optimization

- Deterministic global solver for nonconvex MINLPs
  - Semi-infinite programs
  - Dynamic optimization
  - User-defined functions
- Easy to formulate complicated problems
- Applies McCormick-based relaxations for convex lower-bounding problems
- Open-source and extensible

[4]



[3] Wilhelm, M.E. and Stuber, M.D. EAGO.jl: easy advanced global optimization in Julia. Optimization Methods and Software. 37(2): 425-450 (2022).
[4] Wilhelm, M.E., and Stuber, M.D. Improved Convex and Concave Relaxations of Composite Bilinear Forms. Journal of Optimization Theory and Applications. 197, 174-204 (2023).

EAGO.jl

A Research Platform

- Focus on unsolved problems
- Designed for custom routines
  - Anyone can implement and test new ideas

```
using JuMP, EAGO

import EAGO: lower_problem!
function EAGO.lower_problem!(t::ExtendGPU, m::EAGO.GlobalOptimizer)···
end
```

[3] Wilhelm, M.E. and Stuber, M.D. EAGO.jl: easy advanced global optimization in Julia. Optimization Methods and Software. 37(2): 425-450 (2022).

High-Performance Computing

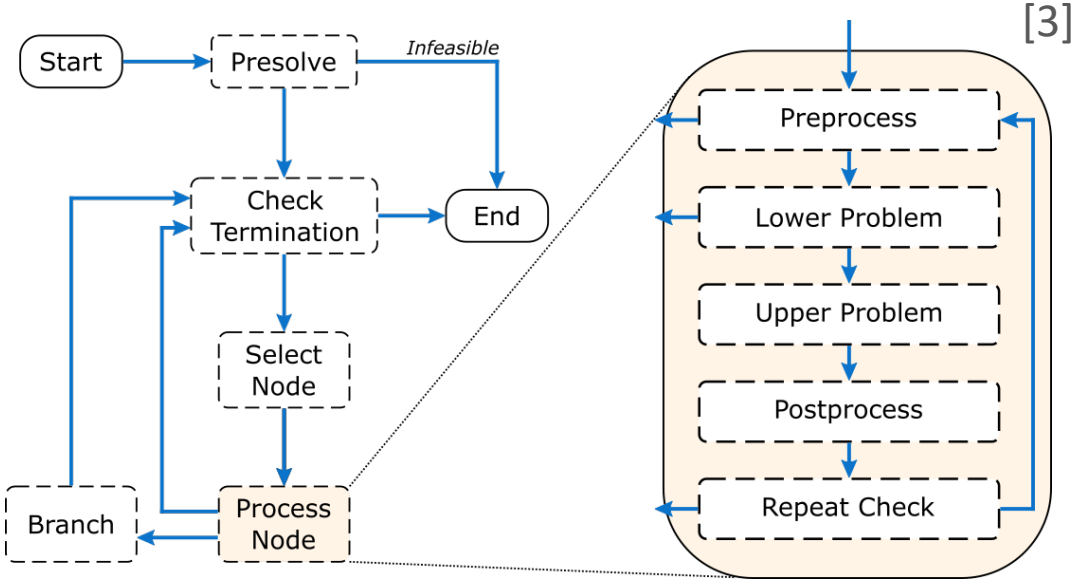
Global Solver	Uses GPU
BARON[5]	**No**
ANTIGONE[6]	**No**
SCIP[7]	**No**
MAiNGO[8]	**Yes (2025)**
EAGO[3]	**Yes (2022)**
[…]	[…]

[3] Wilhelm, M.E. and Stuber, M.D. EAGO.jl: easy advanced global optimization in Julia. Optimization Methods and Software. 37(2): 425-450 (2022).
[5] Sahinidis, N.V. BARON: A general purpose global optimization software package. Journal of Global Optimization. 8(2): 201–205 (1996).
[6] Misener, R. and Floudas, C.A. ANTIGONE: Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations. Journal of Global Optimization. 59(2-3): 503–526 (2014).
[7] Vigerske, S. and Gleixner, A.. SCIP: global optimization of mixed-integer non-linear programs in a branch-and-cut framework. Optimization Methods and Software. 33(3): 563–593 (2017).
[8] Bongartz, D., et al. MAiNGO - McCormick-based Algorithm for mixed-integer Nonlinear Global Optimization. Technical report, RWTH-Aachen (2018).

Aligning Branch-and-Bound with GPUs

Branch-and-Bound Algorithm

[3]



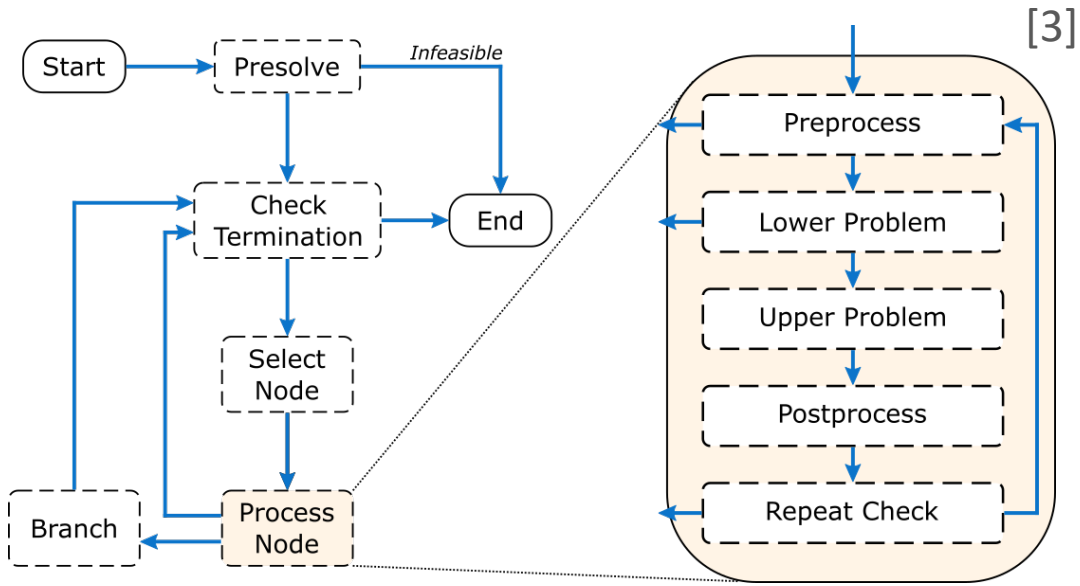
?

GPU Architecture

[9]

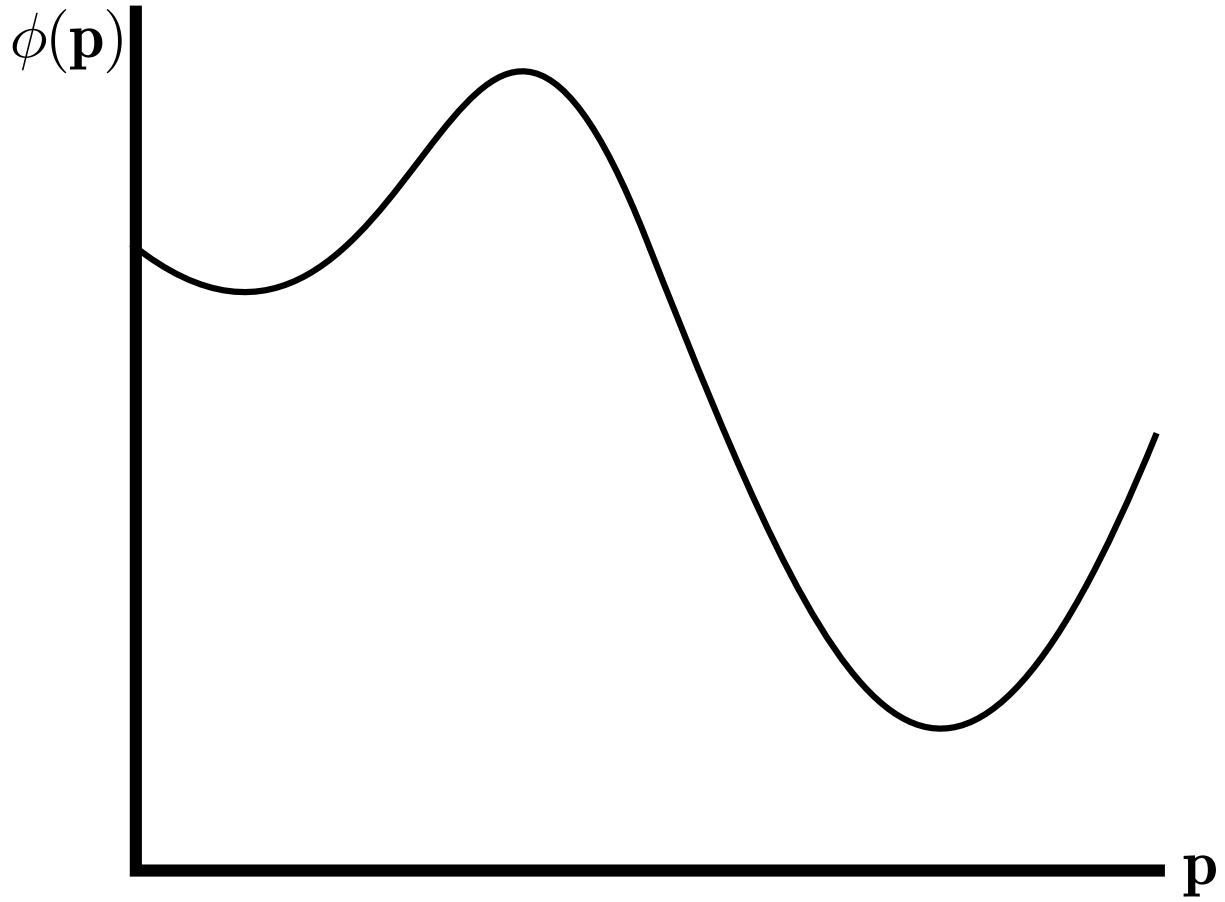
[3] Wilhelm, M.E. and Stuber, M.D. EAGO.jl: easy advanced global optimization in Julia. Optimization Methods and Software. 37(2): 425-450 (2022).
[9] NVIDIA. NVIDIA Tesla V100 GPU Architecture: The World's Most Advanced Data Center GPU. White paper. (2017).

Branch-and-Bound Algorithm



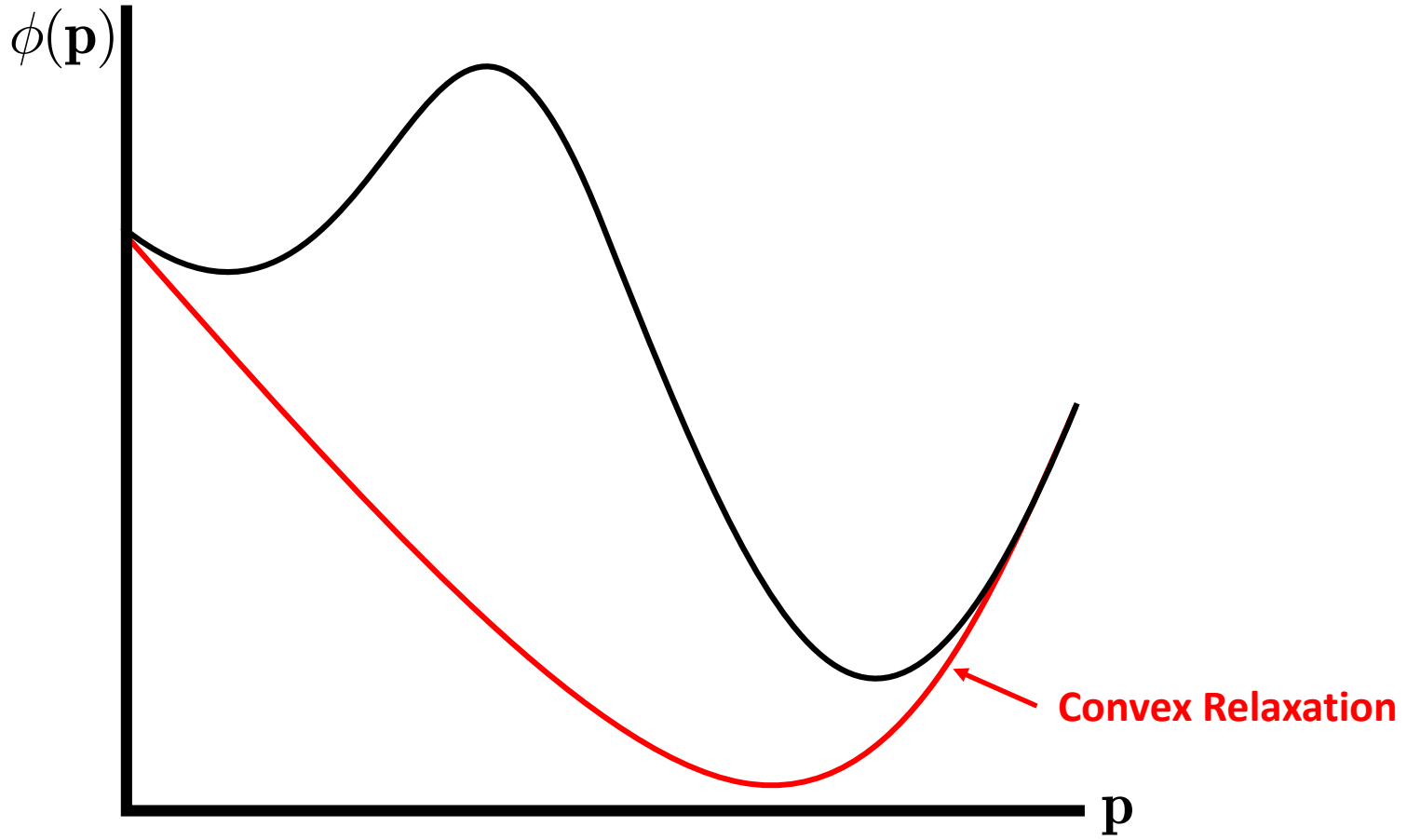
[3] Wilhelm, M.E. and Stuber, M.D. EAGO.jl: easy advanced global optimization in Julia. Optimization Methods and Software. 37(2): 425-450 (2022).

Branch-and-Bound Algorithm



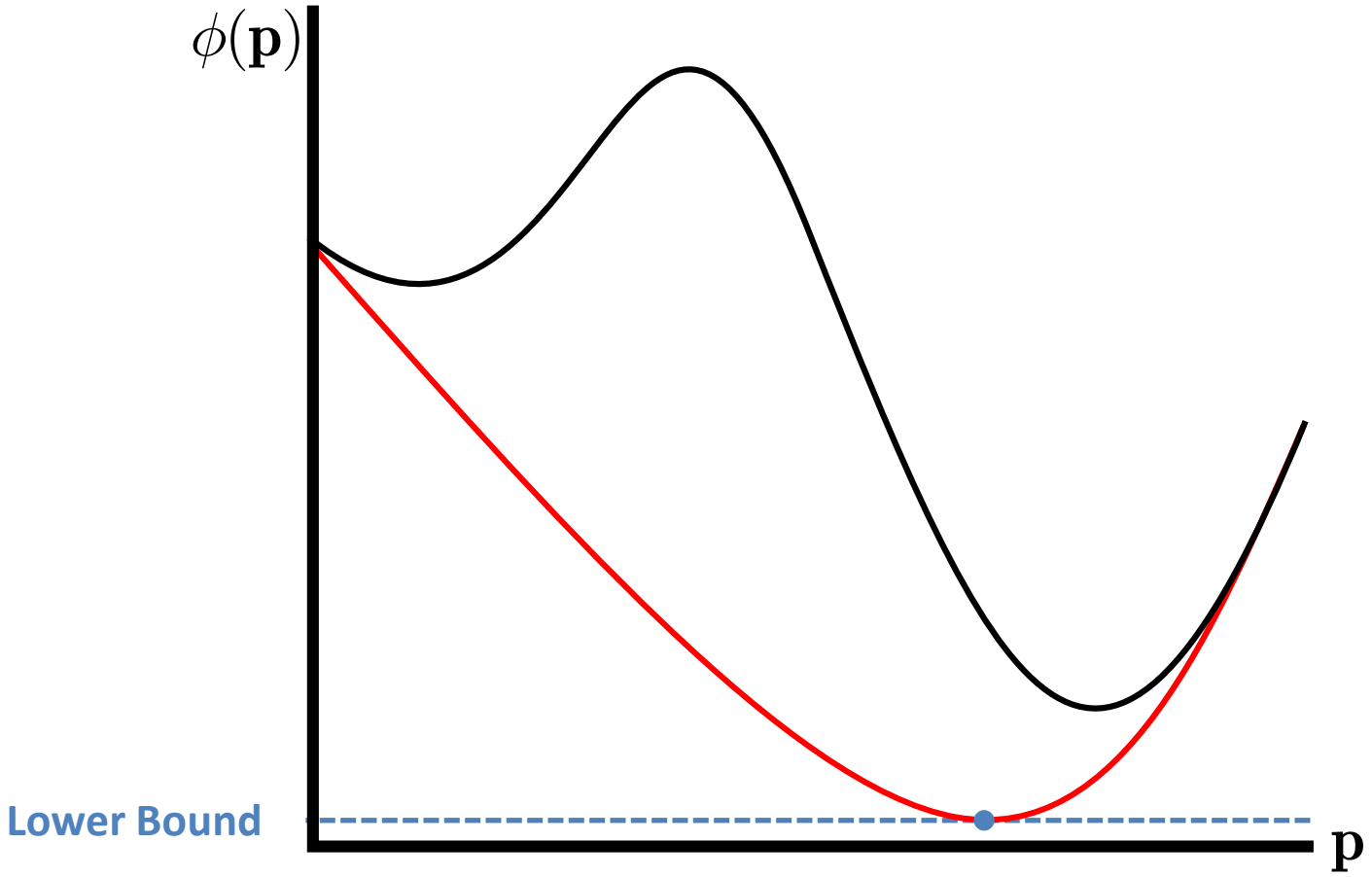
[10] Horst, R., and Tuy, H. Global optimization: Deterministic approaches. Springer Science & Business Media (2013).

Branch-and-Bound Algorithm



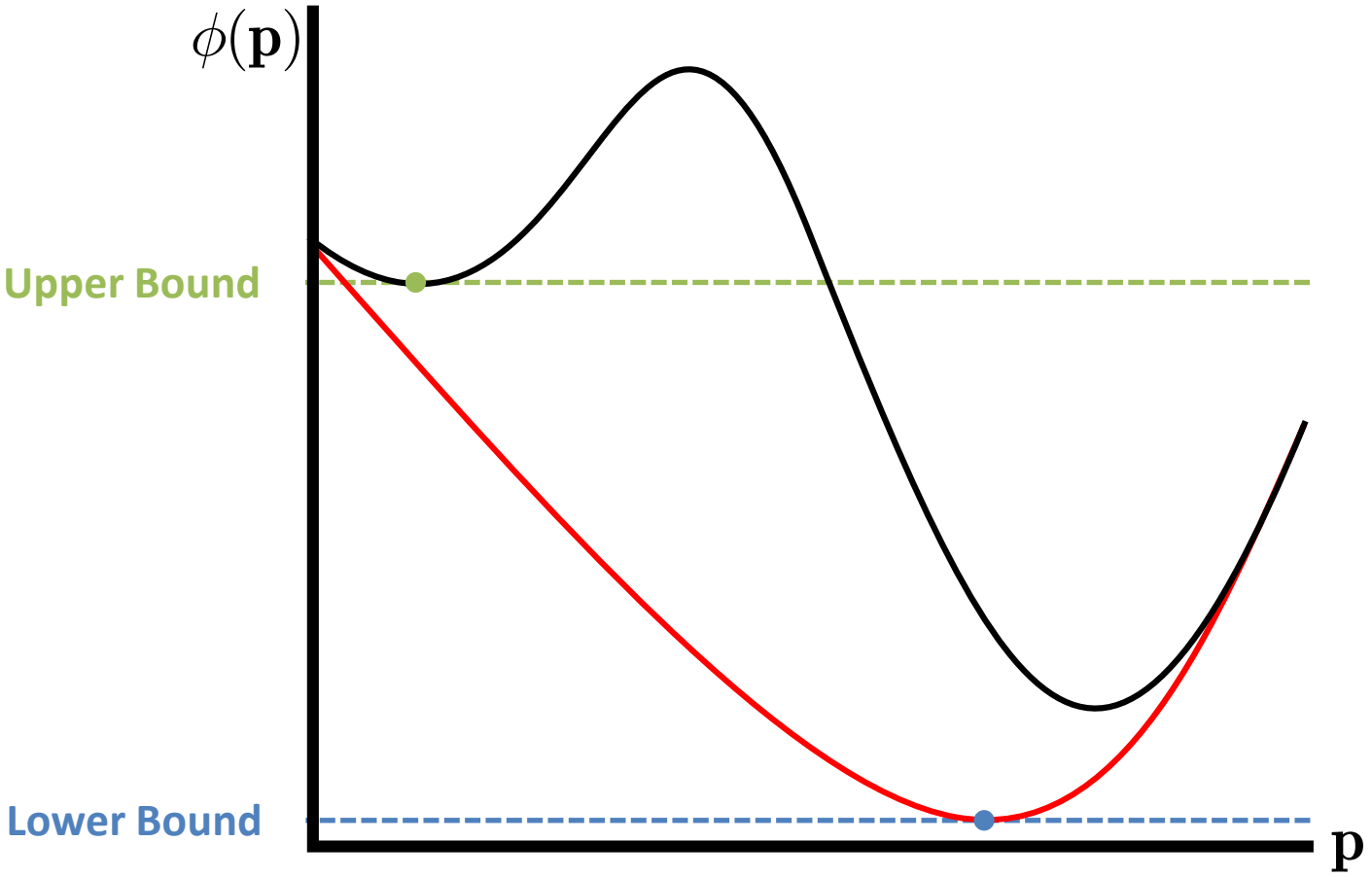
[10] Horst, R., and Tuy, H. Global optimization: Deterministic approaches. Springer Science & Business Media (2013).

Branch-and-Bound Algorithm



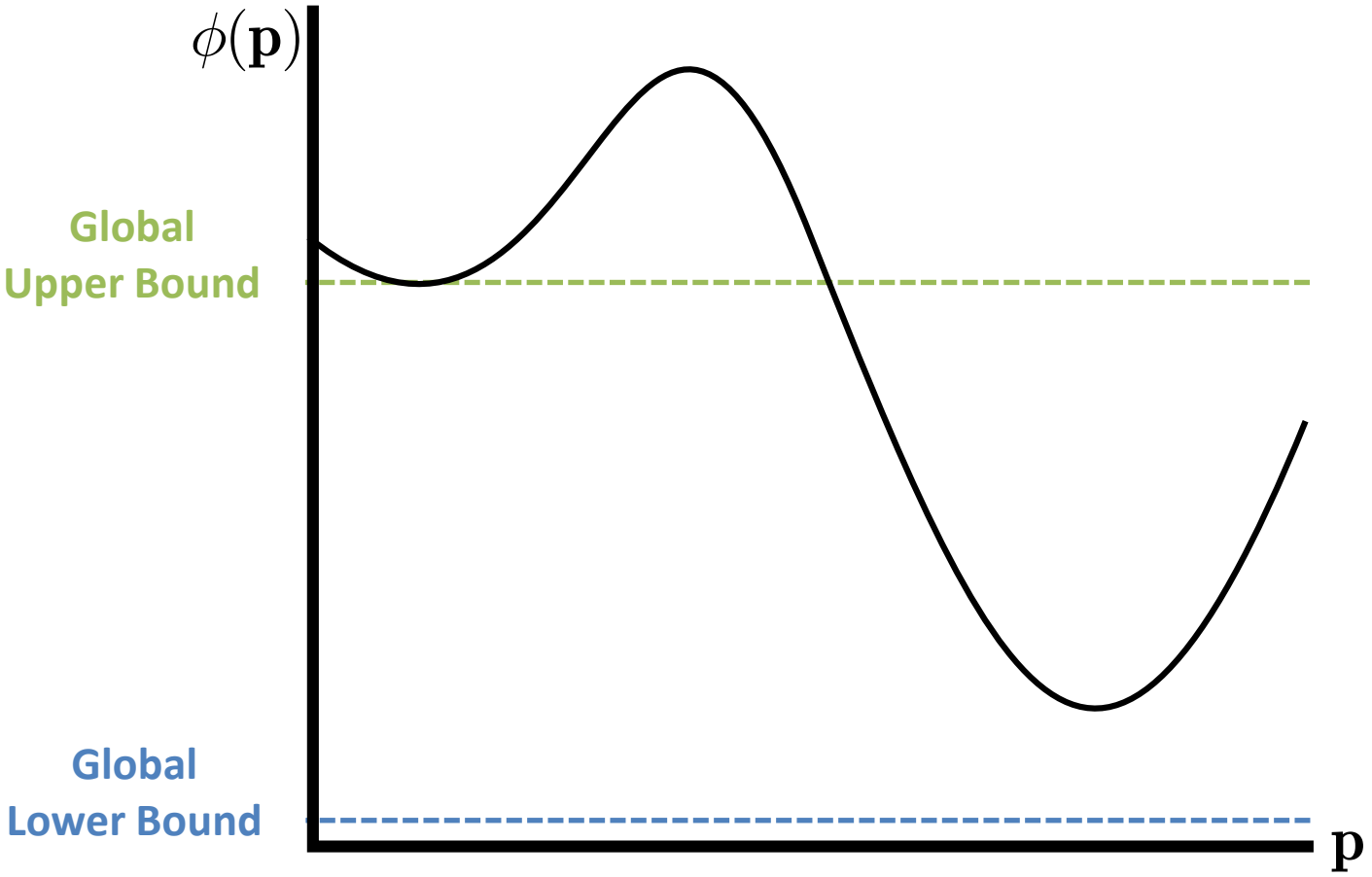
[10] Horst, R., and Tuy, H. Global optimization: Deterministic approaches. Springer Science & Business Media (2013).

Branch-and-Bound Algorithm



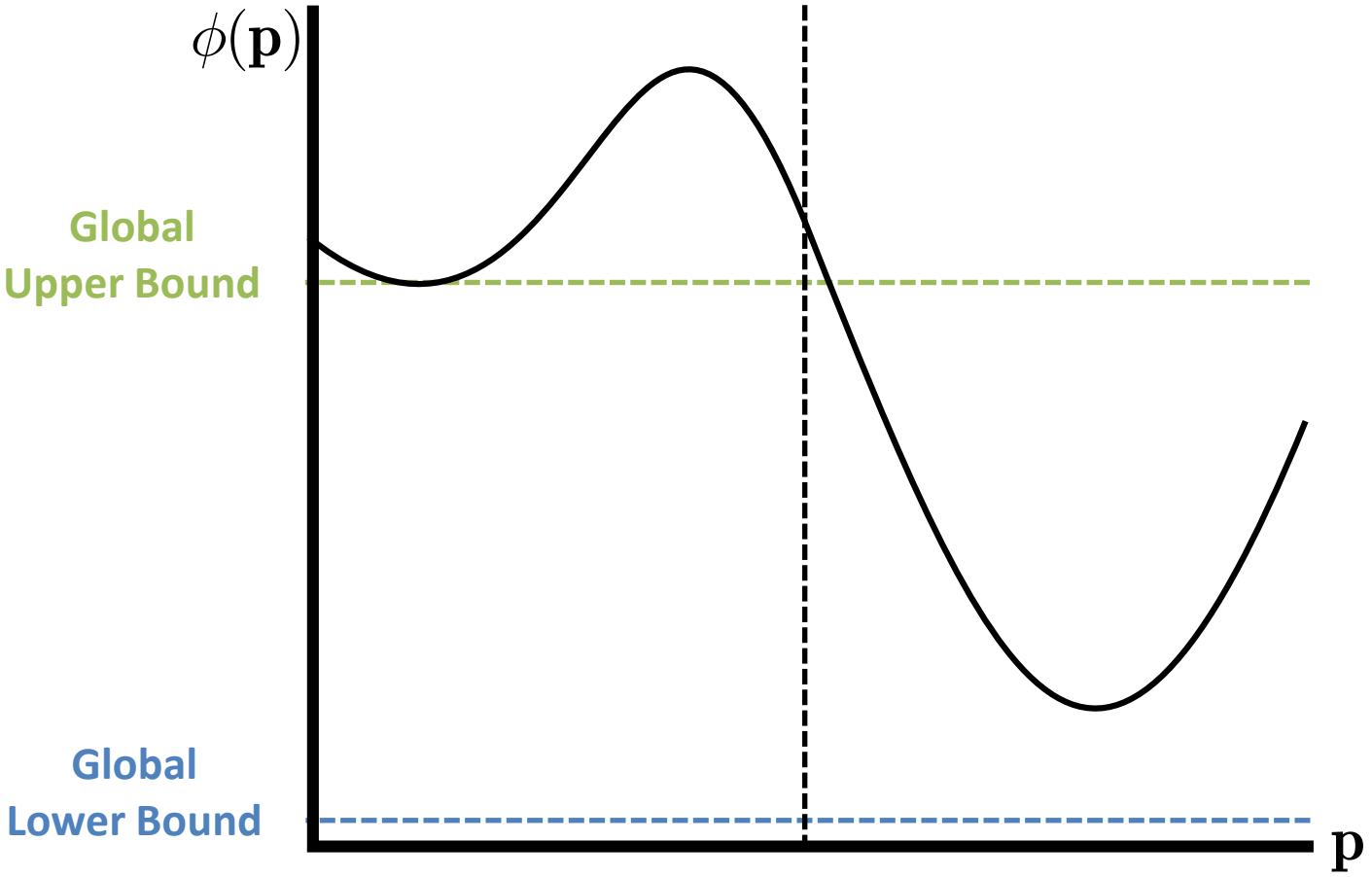
[10] Horst, R., and Tuy, H. Global optimization: Deterministic approaches. Springer Science & Business Media (2013).

Branch-and-Bound Algorithm



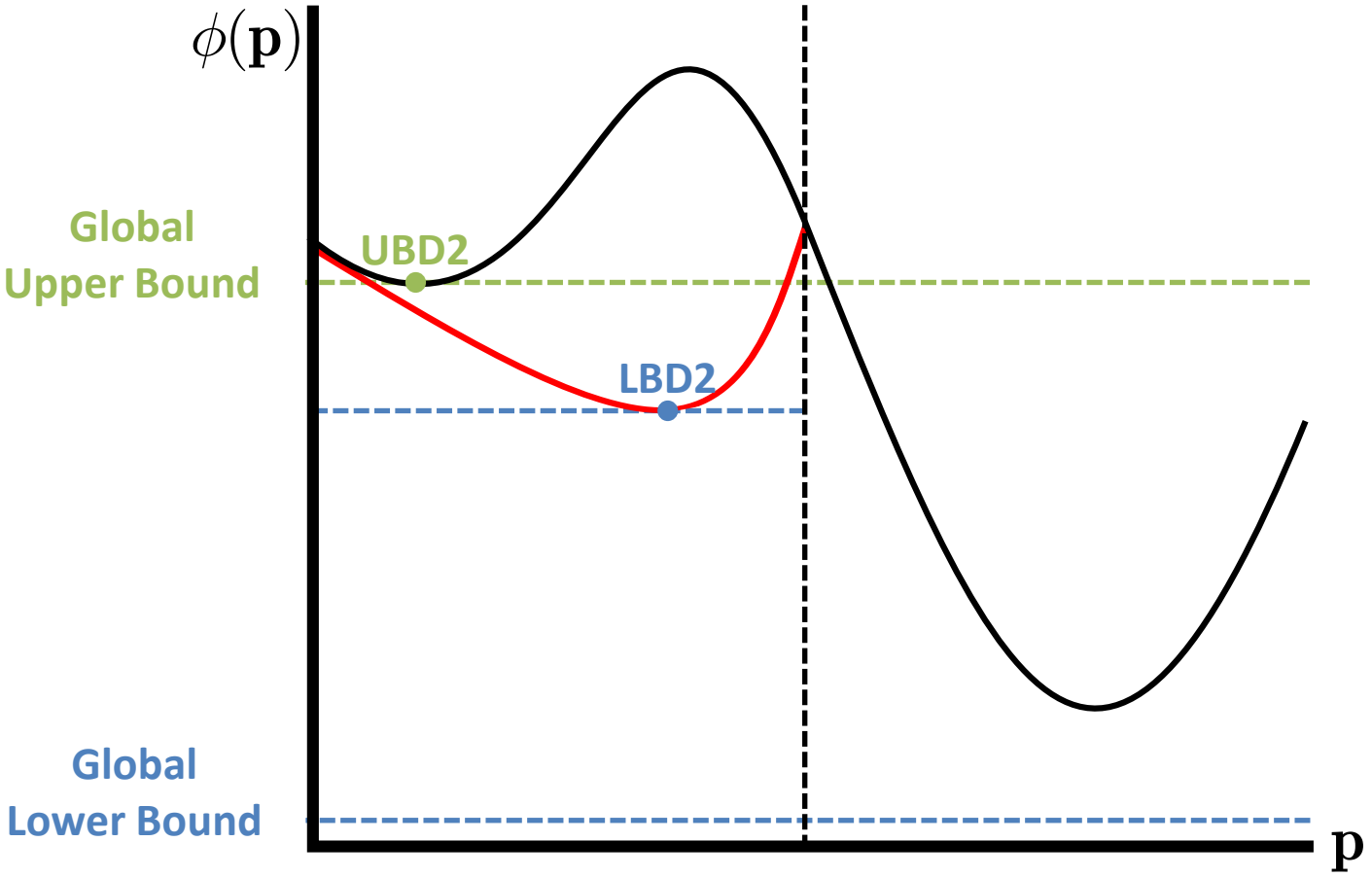
[10] Horst, R., and Tuy, H. Global optimization: Deterministic approaches. Springer Science & Business Media (2013).

Branch-and-Bound Algorithm



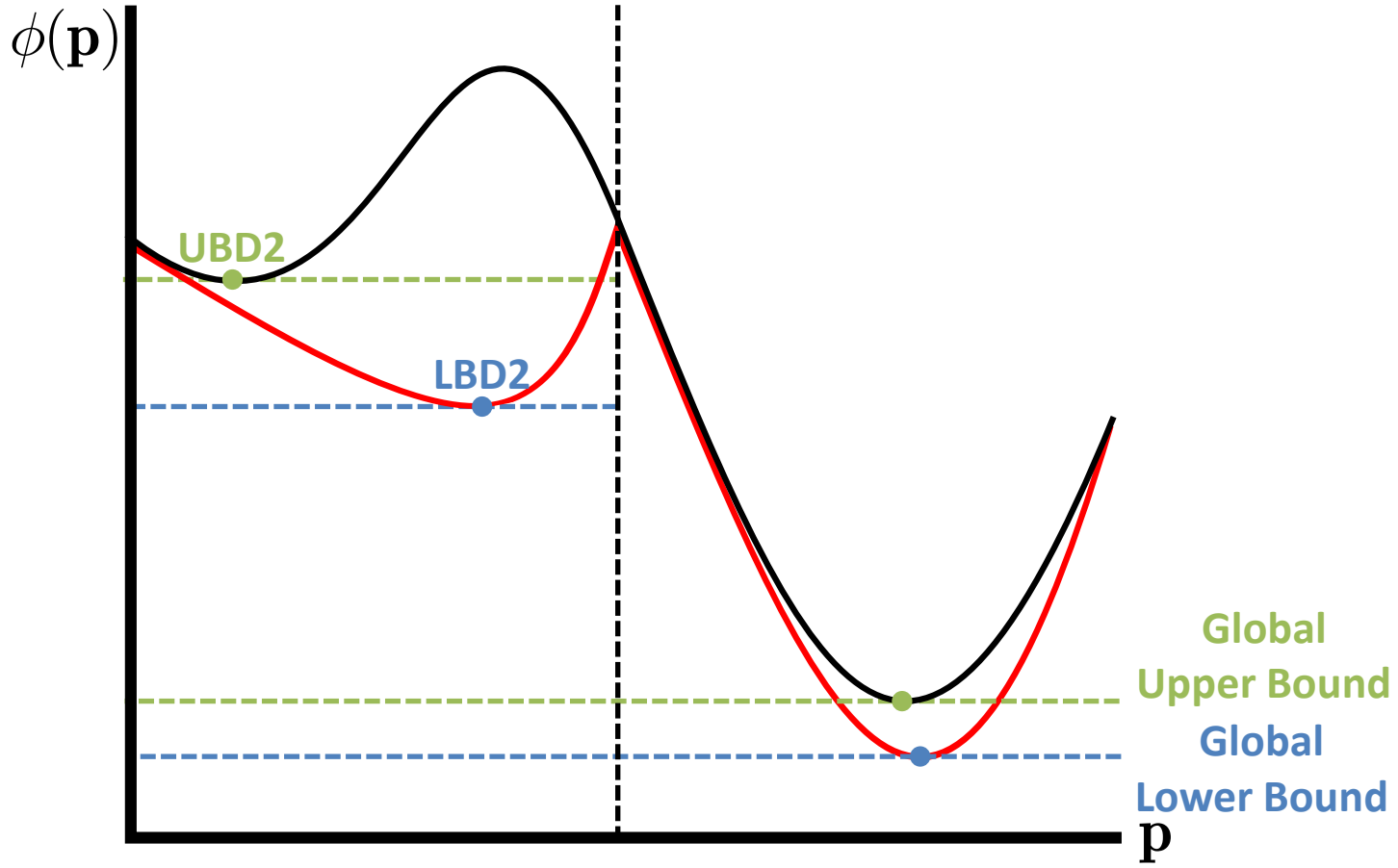
[10] Horst, R., and Tuy, H. Global optimization: Deterministic approaches. Springer Science & Business Media (2013).

Branch-and-Bound Algorithm



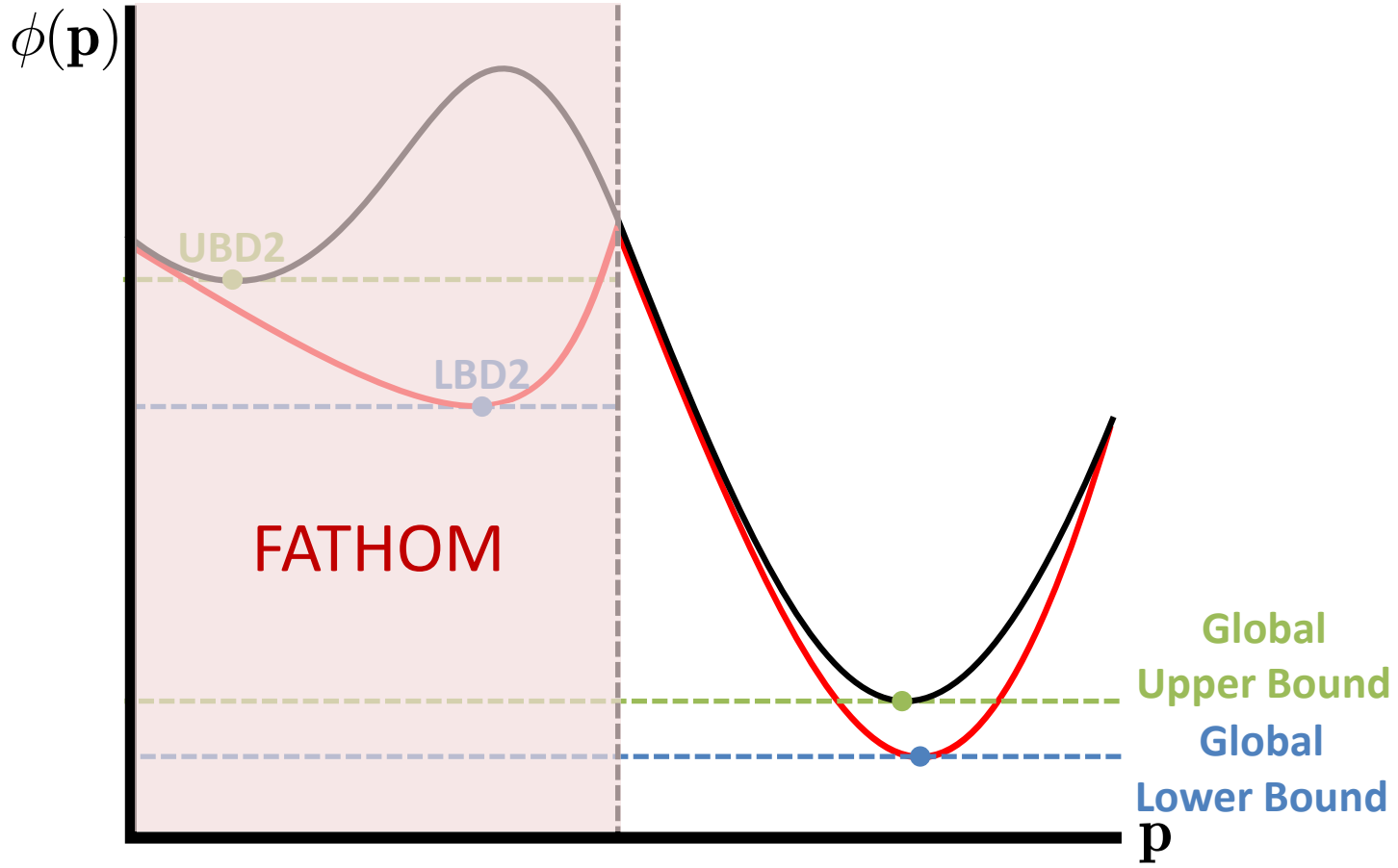
[10] Horst, R., and Tuy, H. Global optimization: Deterministic approaches. Springer Science & Business Media (2013).

Branch-and-Bound Algorithm



[10] Horst, R., and Tuy, H. Global optimization: Deterministic approaches. Springer Science & Business Media (2013).

Branch-and-Bound Algorithm

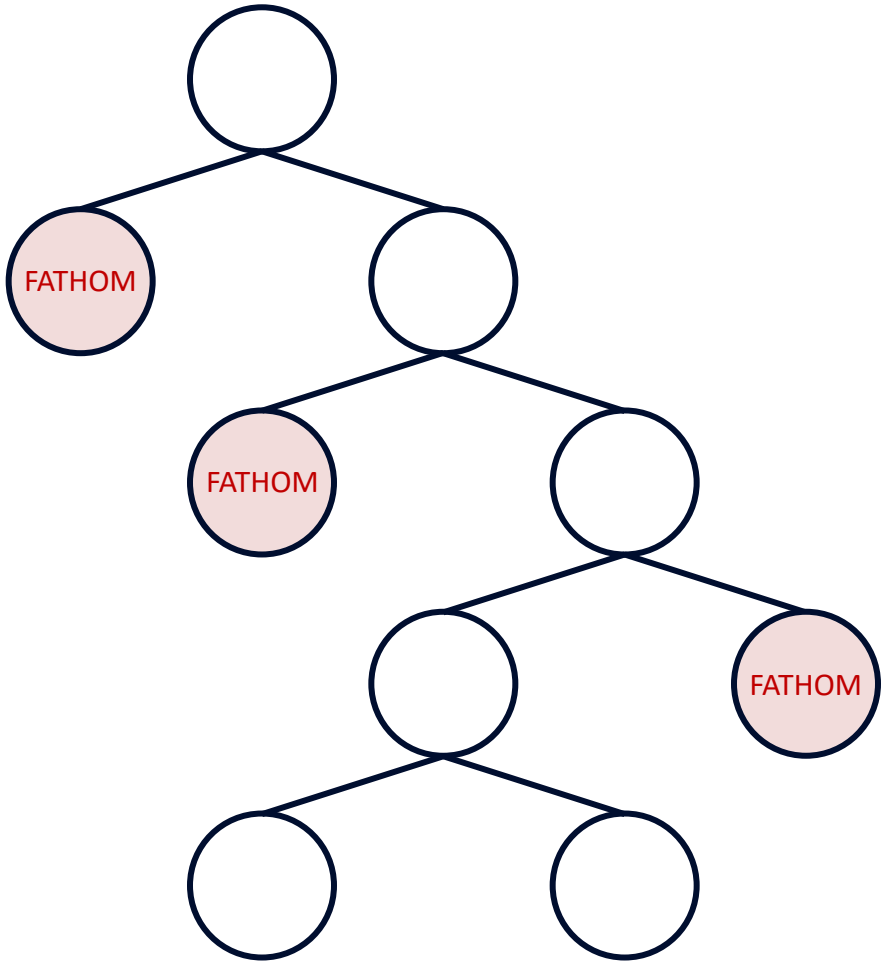


[10] Horst, R., and Tuy, H. Global optimization: Deterministic approaches. Springer Science & Business Media (2013).

Branch-and-Bound Algorithm

Worst-Case Exponential Runtime

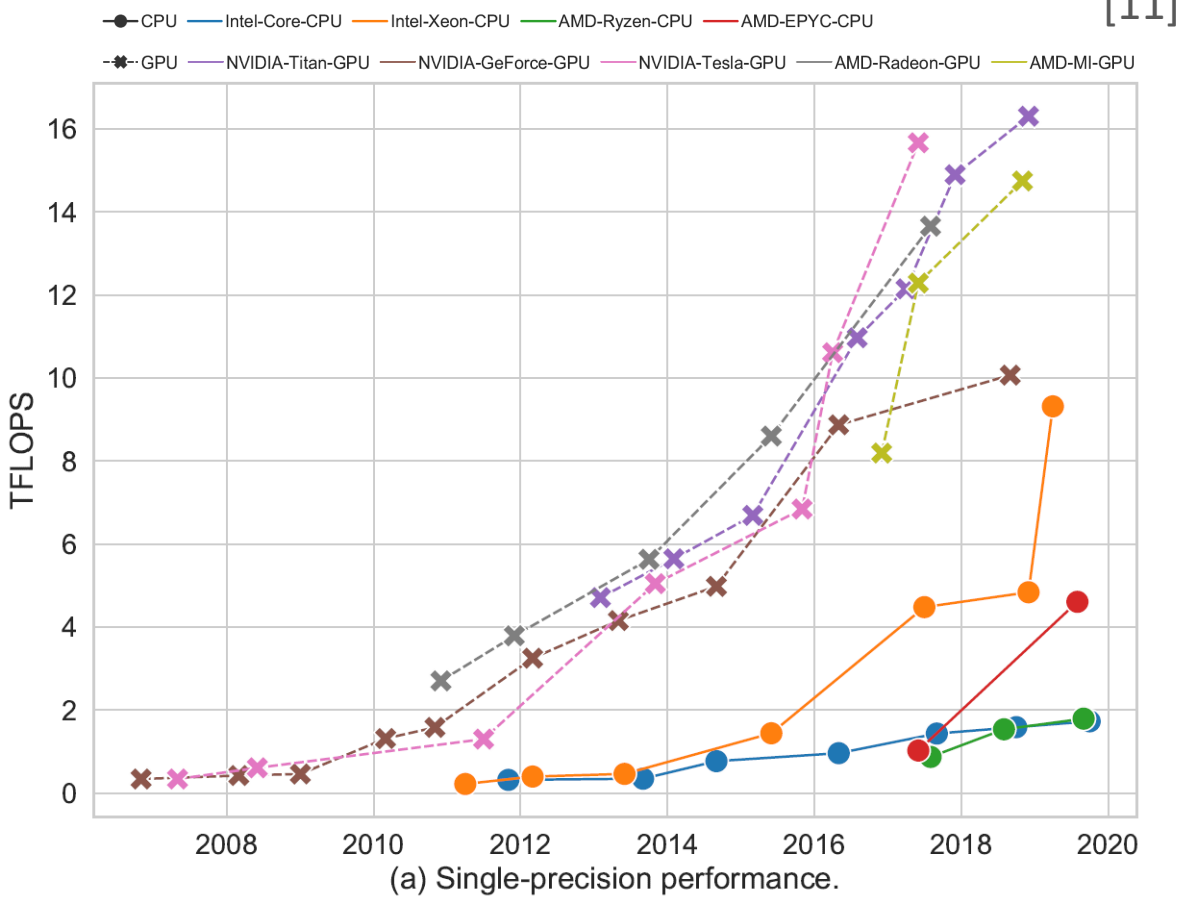
$O(2^n)$



GPU Architecture

- Single-instruction, multiple threads (SIMT) execution mode performs well on specialized tasks
  - Machine learning model training[12]
  - Data analysis[13]
  - Large-scale simulations[14]
  - Etc.

[11]

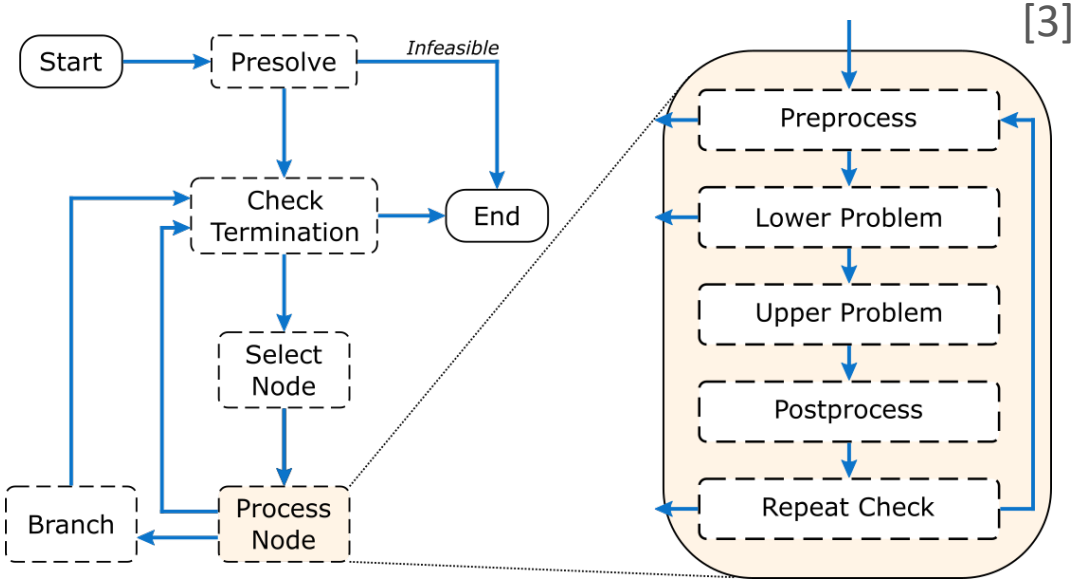


[11] Sun, Y., et al. Summarizing CPU and GPU design trends with product data. (2019).
[12] Steinkraus, D., et al. Using GPUs for machine learning algorithms. Eighth International Conference on Document Analysis and Recognition. 2: 1115-1120 (2005).
[13] Singh, H., et al. GPU and CUDA in Hard Computing Approaches: Analytical Review. Proceedings of ICRIC 2019. 177-196 (2020).
[14] Stone, J.E., et al. GPU-accelerated molecular modeling coming of age. Journal of Molecular Graphics and Modelling. 29(2): 116-125 (2010).

Aligning Branch-and-Bound with GPUs

Branch-and-Bound Algorithm

[3]



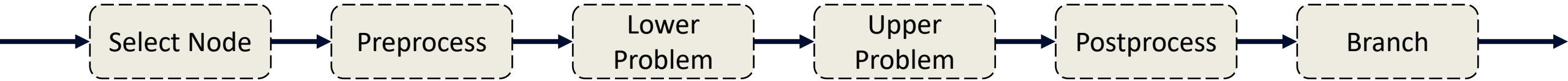
?

GPU Architecture

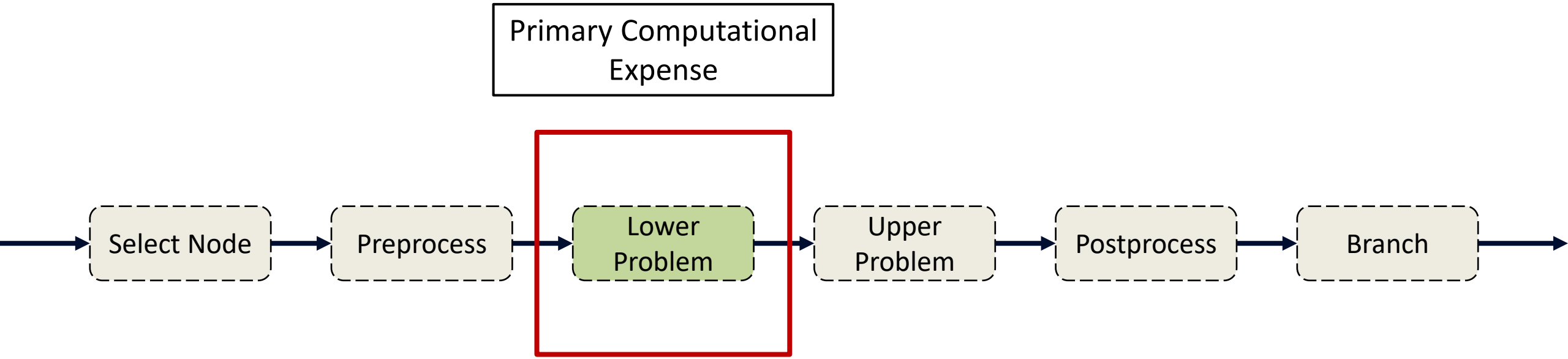
[9]

[3] Wilhelm, M.E. and Stuber, M.D. EAGO.jl: easy advanced global optimization in Julia. Optimization Methods and Software. 37(2): 425-450 (2022).
[9] NVIDIA. NVIDIA Tesla V100 GPU Architecture: The World's Most Advanced Data Center GPU. White paper. (2017).

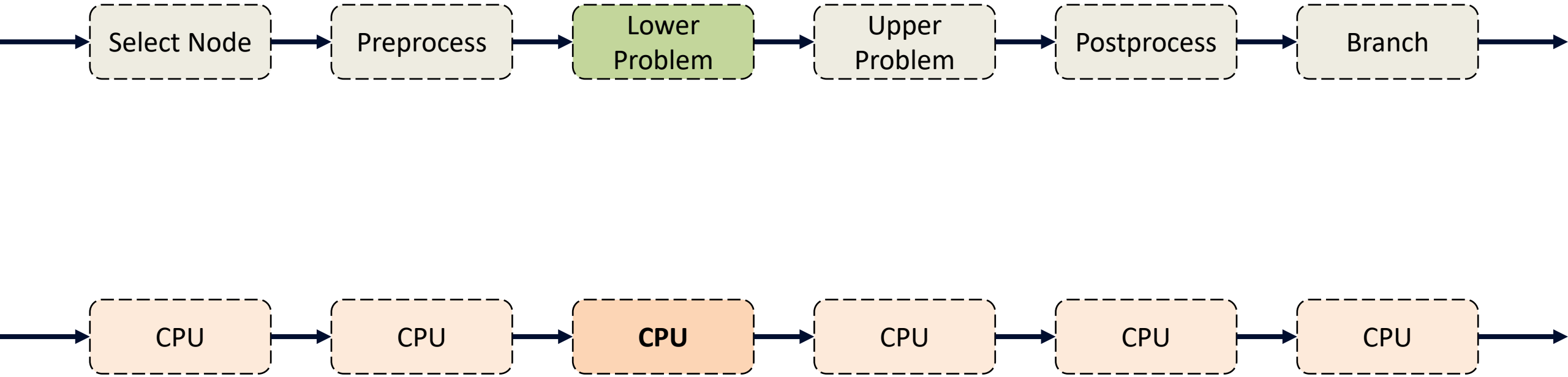
Serial CPU Branch-and-Bound



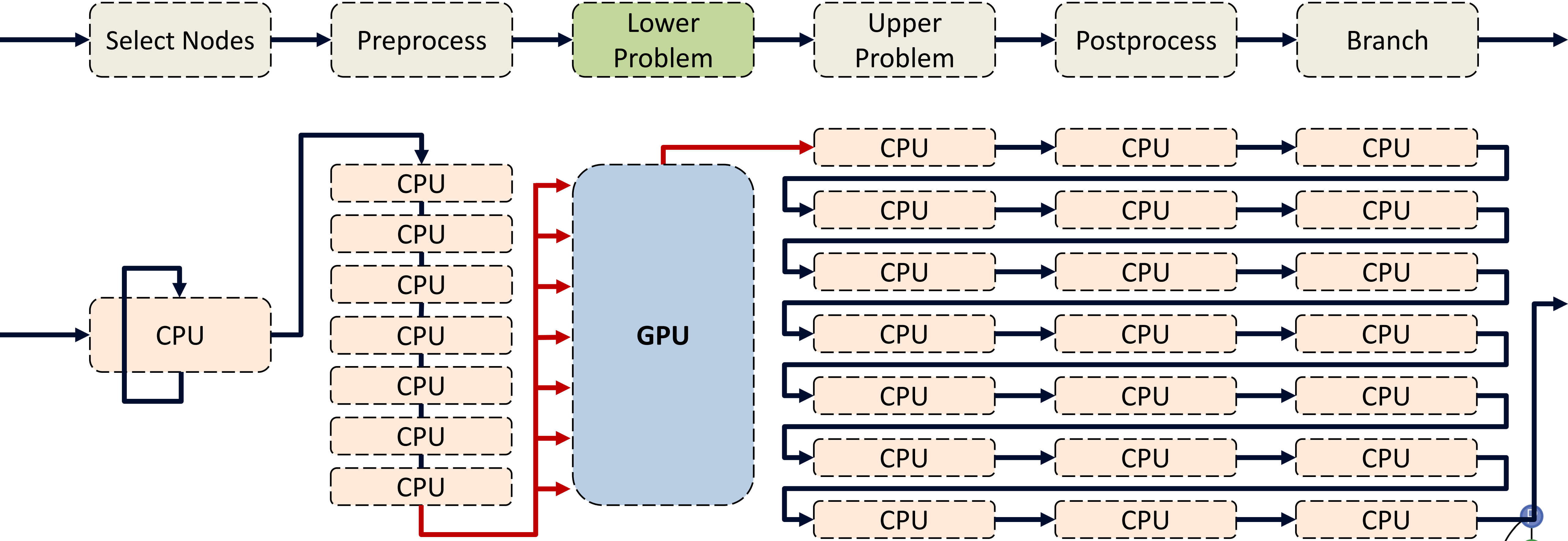
Serial CPU Branch-and-Bound



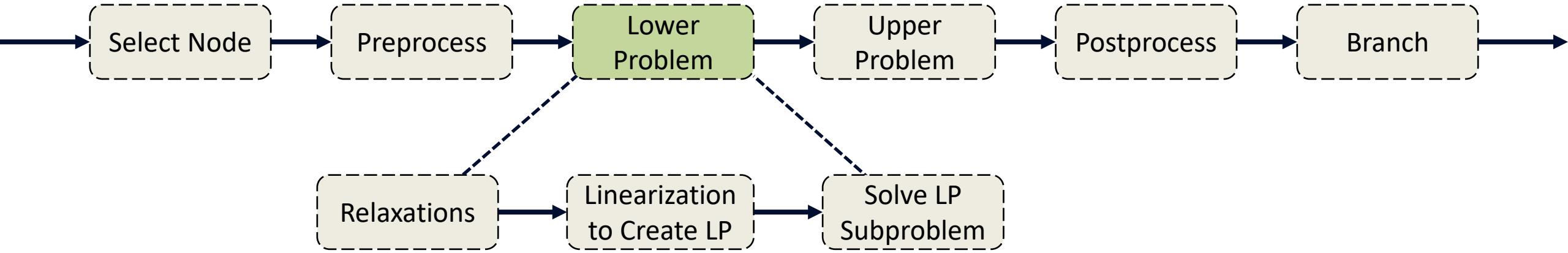
Serial CPU Branch-and-Bound



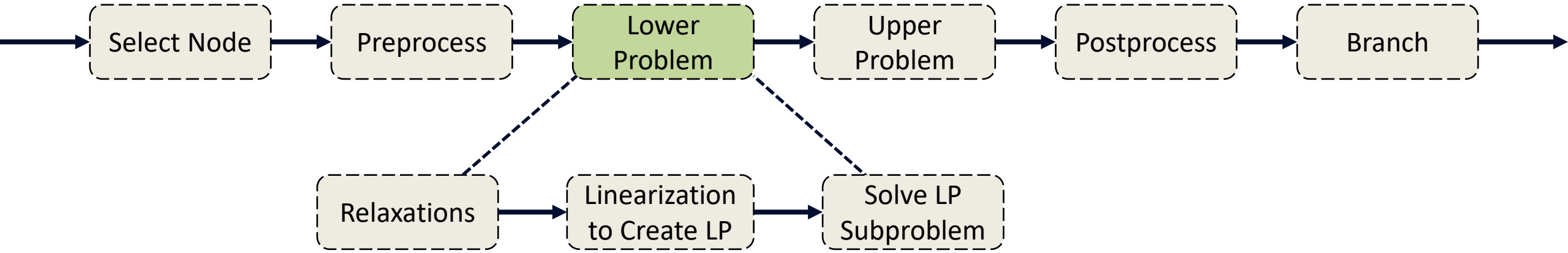
Serial CPU Branch-and-Bound



Serial CPU Branch-and-Bound

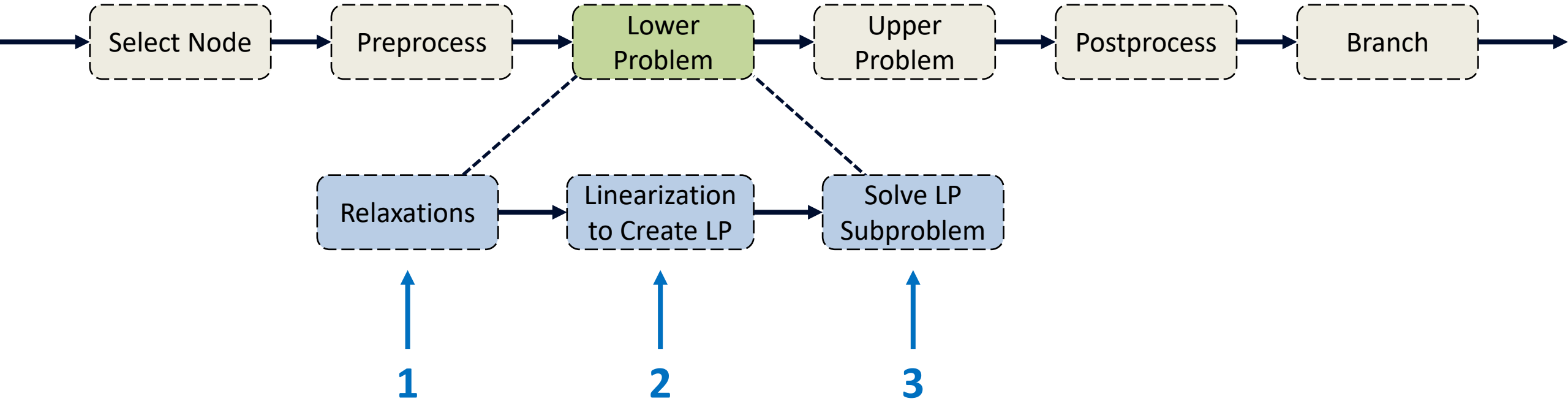


Serial CPU Branch-and-Bound




```
using JuMP, EAGO, HiGHS

model = JuMP.Model(EAGO.Optimizer(SubSolvers(; r=HiGHS.Optimizer())))
```

Tasks for GPU Parallelization

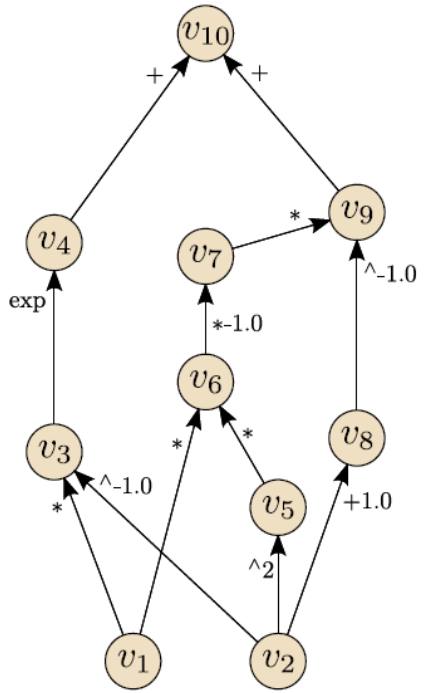


McCormick Relaxations

- Provide convex underestimators of nonlinear functions over a specified domain
- Built recursively for factorable functions

$$\exp\left(x \,/\, y\right) - xy^2 \,/\, \left(y+1\right)$$

$$\begin{aligned}
v_1 &= x \\
v_2 &= y \\
v_3 &= v_1 / v_2 \\
v_4 &= \exp(v_3) \\
v_5 &= v_2^2 \\
v_6 &= v_1 v_5 \\
v_7 &= -v_6 \\
v_8 &= v_2 + 1.0 \\
v_9 &= v_7 / v_8 \\
v_{10} &= v_4 + v_9
\end{aligned}$$



[15] McCormick, G. Computability of global solutions to factorable nonconvex programs: Part I - Convex underestimating problems. Mathematical Programming. 10 (1): 147-175 (1976).
[16] Mitsos, A., et al. McCormick-based relaxations of algorithms. SIAM Journal on Optimization, SIAM. 20, 73-601 (2009).
[17] Scott, J. K., et al. Generalized McCormick relaxations. Journal of Global Optimization. 51(4): 569-606 (2011).

McCormick Relaxations

$$x \in [x^L, x^U],\ y \in [y^L, y^U]$$

Interval Arithmetic

$$x \cdot y = [\min\{x^L y^L, x^L y^U, x^U y^L, x^U y^U\}, \max\{x^L y^L, x^L y^U, x^U y^L, x^U y^U\}]$$

McCormick Relaxations

$$x \in [x^L, x^U],\ y \in [y^L, y^U]$$

Interval Arithmetic

$$x \cdot y = [\min\{x^L y^L, x^L y^U, x^U y^L, x^U y^U\}, \max\{x^L y^L, x^L y^U, x^U y^L, x^U y^U\}]$$

McCormick

$$(x \cdot y)^{cv} = \max\{x^L y + x y^L - x^L y^L, x^U y + x y^U - x^U y^U\}$$
$$(x \cdot y)^{cc} = \min\{x^U y + x y^L - x^U y^L, x^L y + x y^U - x^L y^U\}$$

McCormick.jl

$$x \in [x^L, x^U],\ y \in [y^L, y^U]$$

Interval Arithmetic

$$x \cdot y = [\min\{x^L y^L, x^L y^U, x^U y^L, x^U y^U\}, \max\{x^L y^L, x^L y^U, x^U y^L, x^U y^U\}]$$

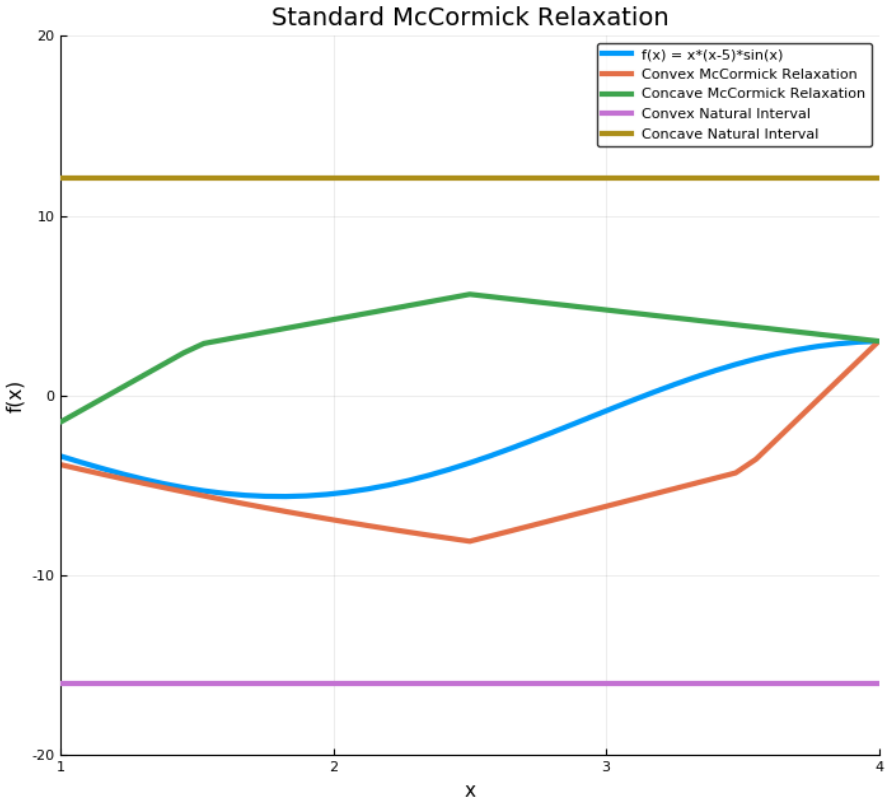
McCormick

$$(x \cdot y)^{cv} = \max\{x^L y + x y^L - x^L y^L, x^U y + x y^U - x^U y^U\}$$
$$(x \cdot y)^{cc} = \min\{x^U y + x y^L - x^U y^L, x^L y + x y^U - x^L y^U\}$$

```
@inline function *(x1::MC{N,T}, x2::MC{N,T}) where {N, T<:Union{NS,MV}}
    mult_kernel(x1, x2, x1.Intv*x2.Intv)
end
```

McCormick.jl

```
using McCormick

x = 2.0
xIntv = interval(1.0, 4.0)
xMC = MC{1,NS}(x, xIntv, 1)

f(x) = x*(x - 5.0)*sin(x)
f(xIntv)
f(xMC)
```



SourceCodeMcCormick.jl

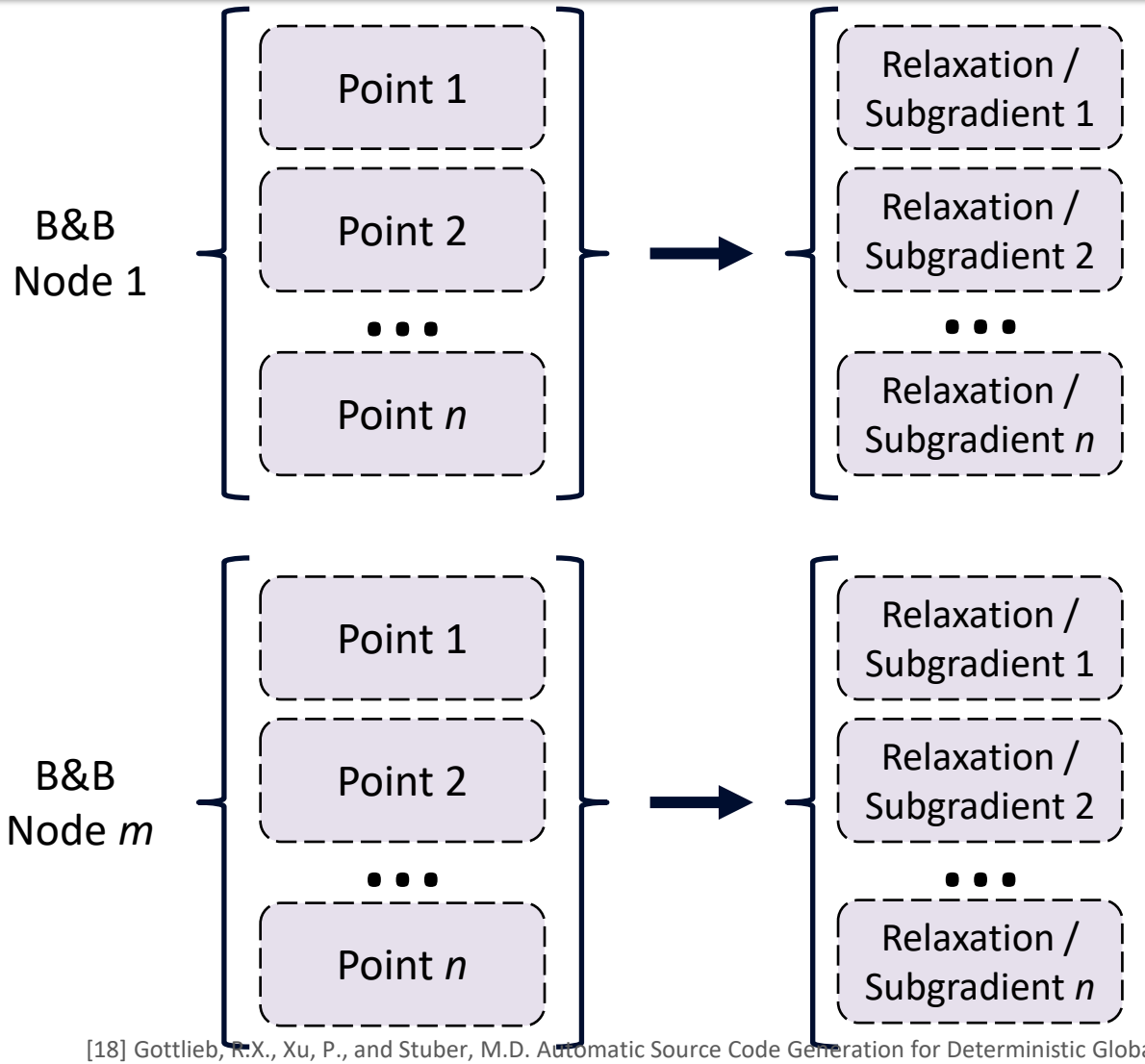
- Enables parallel evaluation of relaxations on GPUs
- Generates customized CUDA kernels

```
using SourceCodeMcCormick

Symbolics.@variables x, y
expr = exp(x/y) - x*y^2/(y+1)
func = kgen(expr)
```

```
# Kernel(s) generated for the expression: exp(x / y) + (-x*(y^2)) / (1 + y)

function f_5hzkcjneJVK_1(OUT, x, y)
    idx = threadIdx().x + (blockIdx().x - Int32(1)) * blockDim().x
    stride = blockDim().x * gridDim().x
    col = Int32(1)
    colmax = Int32(2)
    temp1_lo = 0.0
    temp1_hi = 0.0
    temp1_cv = 0.0
    temp1_cc = 0.0
    temp1_cvgrad = @MVector zeros(Float64, 2)
    temp1_ccgrad = @MVector zeros(Float64, 2)
    temp2_lo = 0.0
    temp2_hi = 0.0
    temp2_cv = 0.0
    temp2_cc = 0.0
    temp2_cvgrad = @MVector zeros(Float64, 2)
    temp2_ccgrad = @MVector zeros(Float64, 2)
    temp3_lo = 0.0
    temp3_hi = 0.0
    temp3_cv = 0.0
    temp3_cc = 0.0
    temp3_cvgrad = @MVector zeros(Float64, 2)
    temp3_ccgrad = @MVector zeros(Float64, 2)
    while idx <= Int32(size(OUT,1))
        ##############
        ## Squared ##
        ##############

        # Reset the column counter
        col = Int32(1)

        # Begin rule
        if y[idx,3] <= 0.0
            eps_min = y[idx,3]
```

[18] Gottlieb, R.X., Xu, P., and Stuber, M.D. Automatic Source Code Generation for Deterministic Global Optimization with Pa

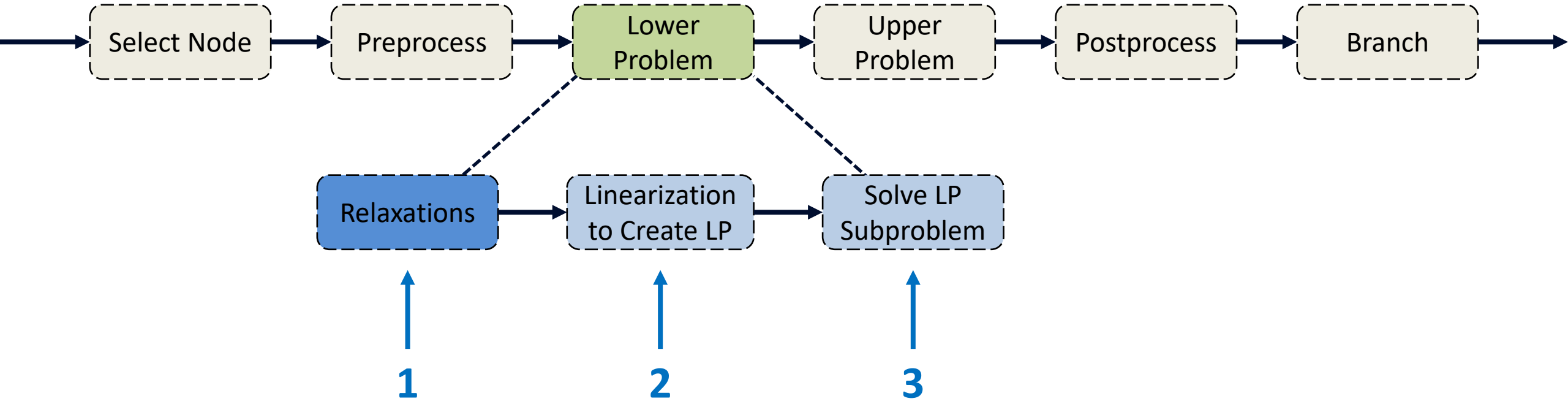
SourceCodeMcCormick.jl



Expression Form	SCMC Avg. Evaluation Time (ns)	McCormick.jl Avg. Evaluation Time (ns)	Evaluation Time Speed-Up
$\sum_{i=1}^{10} x_i$	1.9×10^0	1.90×10^2	99.9x
$\exp(\sum_{i=1}^{10} x_i)$	3.0×10^0	2.72×10^2	90.6x
$(\sum_{i=1}^{10} x_i)^2$	3.1×10^0	2.36×10^2	76.2x
$\prod_{i=1}^{10} x_i$	1.15×10^1	5.79×10^2	50.3x
$\exp(\prod_{i=1}^{10} x_i)$	1.19×10^1	6.38×10^2	53.6x
$(\prod_{i=1}^{10} (x_i)^2)$	1.02×10^1	9.40×10^2	92.1x
$x_1/(\prod_{i=2}^{10} x_i)$	6.2×10^0	5.11×10^2	82.4x
alkyl objective	$7. \times 10^{-1}$	1.61×10^2	229.9x
ex6_2_10 objective	9.04×10^1	1.37×10^4	151.1x
arki0002 objective	2.58×10^3	1.62×10^7	6269.5x
Trained ANN	1.90×10^1	5.51×10^3	290.0x
Training ANN	9.59×10^1	1.28×10^4	133.4x

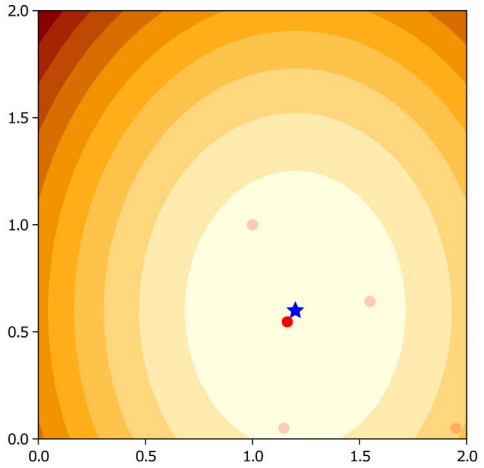
[18] Gottlieb, R.X., Xu, P., and Stuber, M.D. Automatic Source Code Generation for Deterministic Global Optimization with Parallel Architectures. Optimization Methods and Software. 1-39 (2024).

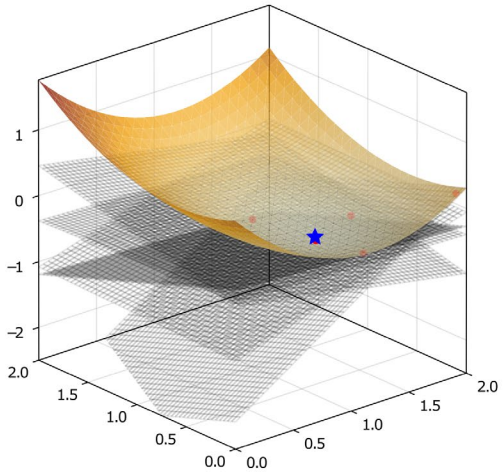
Tasks for GPU Parallelization



Kelley's Method

n	Lower Bound
1	-0.9830
2	-0.7100
3	-0.5815
4	-0.4962
5	-0.4001

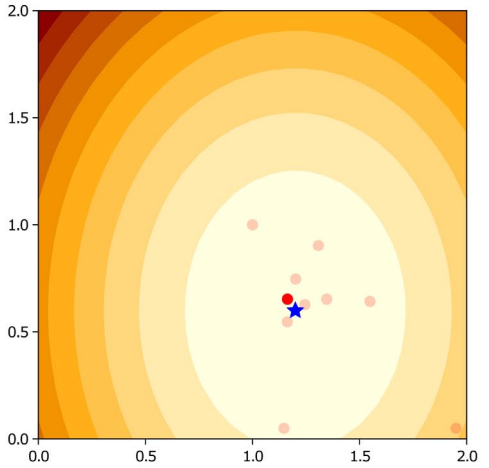


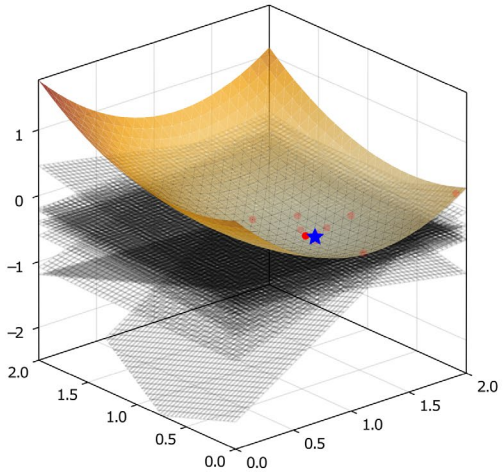


[19] Kelley, J. E. The Cutting-Plane Method for Solving Convex Programs. Journal of the Society for Industrial and Applied Mathematics. 8(4): 703-712 (1960).

Kelley's Method

n	Lower Bound
1	-0.9830
2	-0.7100
3	-0.5815
4	-0.4962
5	-0.4001
6	-0.3891
7	-0.3854
8	-0.3817
9	-0.3781
10	-0.3773



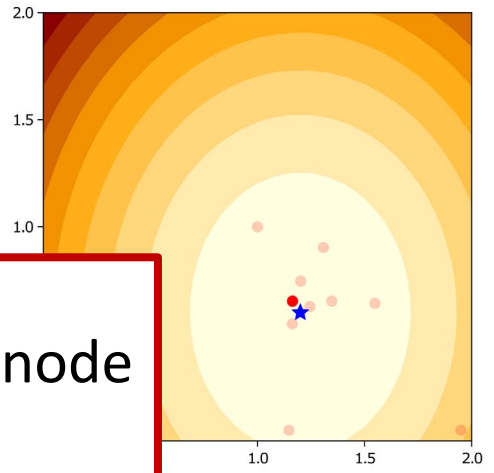


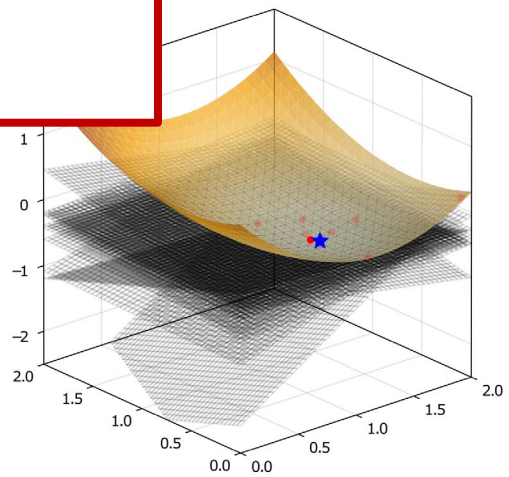
[19] Kelley, J. E. The Cutting-Plane Method for Solving Convex Programs. Journal of the Society for Industrial and Applied Mathematics. 8(4): 703-712 (1960).

Kelley's Method

n	Lower Bound
1	-0.9830
2	
3	
4	
5	
6	
7	-0.3854
8	-0.3817
9	-0.3781
10	-0.3773

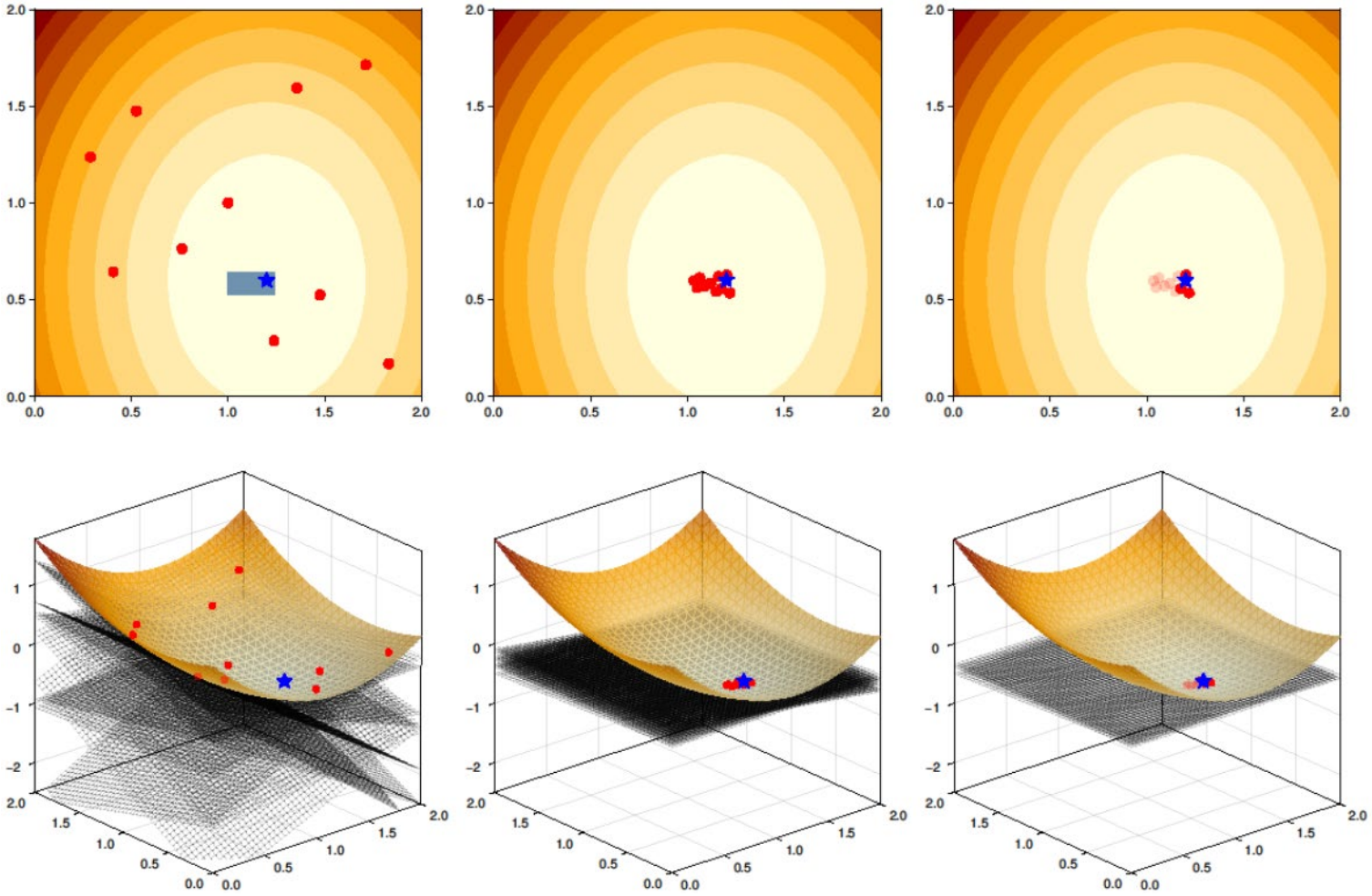
- Requires *n* LP solves per node
- Sequential





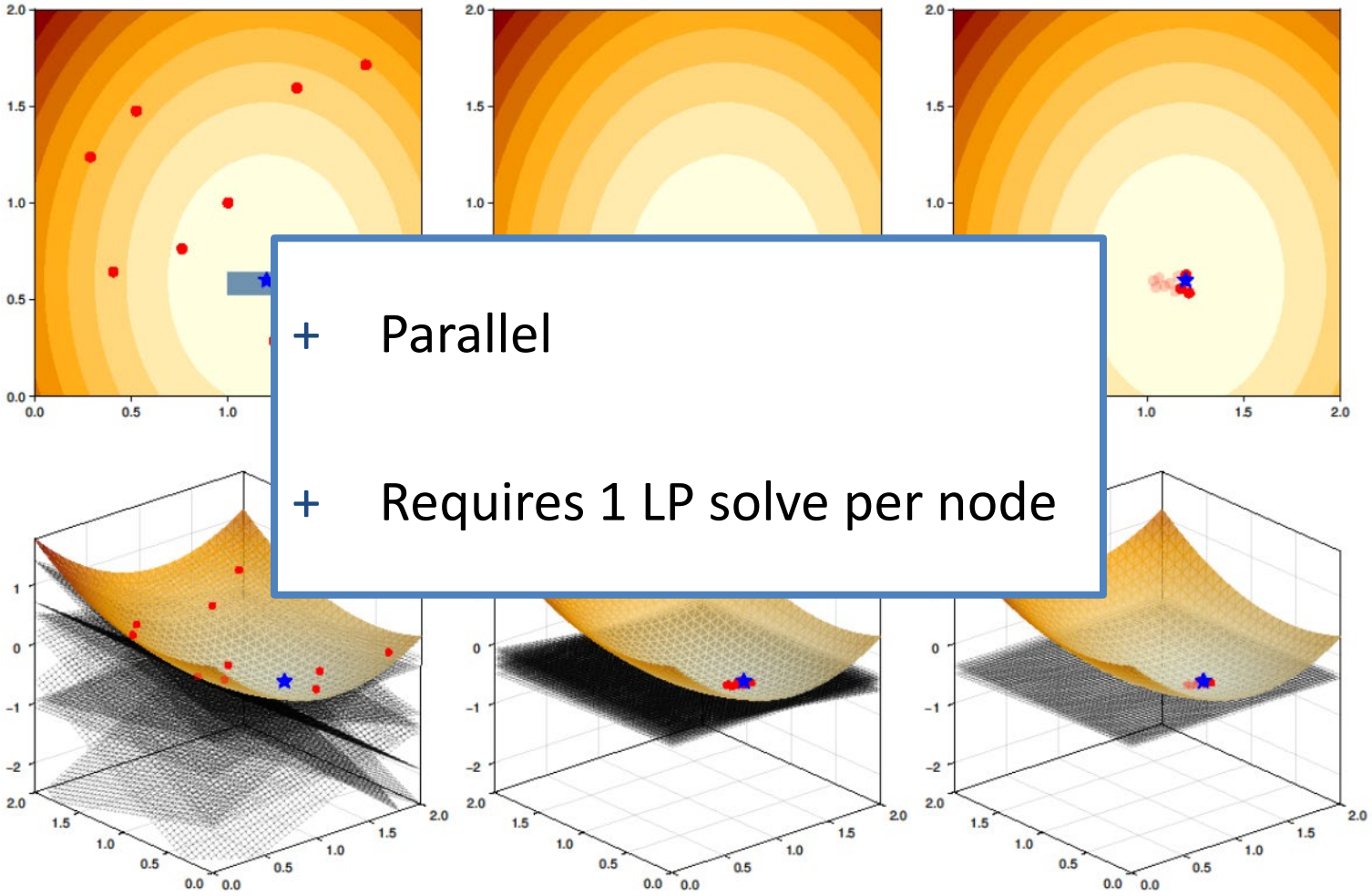
[19] Kelley, J. E. The Cutting-Plane Method for Solving Convex Programs. Journal of the Society for Industrial and Applied Mathematics. 8(4): 703-712 (1960).

Sobol Sampling



Lower Bound: -0.4506	Lower Bound: -0.3758	**Lower Bound: -0.3758**

Sobol Sampling



+ Parallel

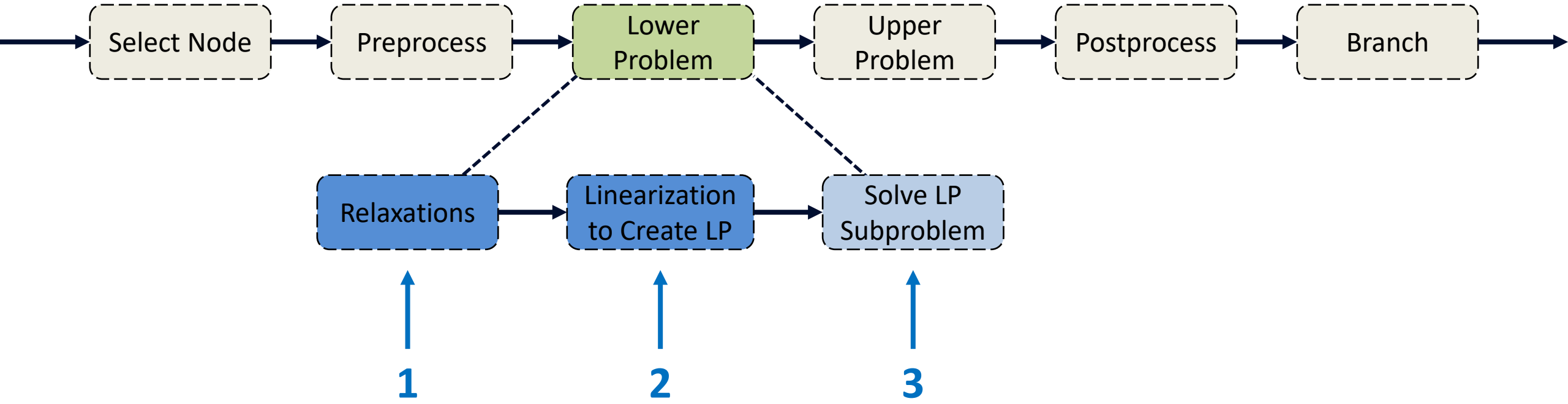
+ Requires 1 LP solve per node

Lower Bound: -0.4506

Lower Bound: -0.3758

Lower Bound: -0.3758

Tasks for GPU Parallelization



PDLP

- Competitive first-order method for solving LPs

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & c^\top x \\ \text{subject to:} \quad & Gx \geq h \\ & Ax = b \\ & l \leq x \leq u \end{aligned} \qquad \begin{aligned} \underset{y\in\mathbb{R}^{m_1+m_2},\lambda\in\mathbb{R}^n}{\text{maximize}} \quad & q^\top y + l^\top \lambda^+ - u^\top \lambda^- \\ \text{subject to:} \quad & c - K^\top y = \lambda \\ & y_{1:m_1} \geq 0 \\ & \lambda \in \Lambda\,, \end{aligned}$$

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

$$\begin{aligned} x^{k+1} &= \underset{X}{\mathbf{proj}}(x^k - \tau(c - K^\top y^k)) \\ y^{k+1} &= \underset{Y}{\mathbf{proj}}(y^k + \sigma(q - K(2x^{k+1} - x^k))) \end{aligned}$$

Practical Large-Scale Linear Programming using Primal-Dual Hybrid Gradient

David Applegate
Google Research
dapplegate@google.com

Mateo Díaz
California Institute of Technology*
mateodd@caltech.edu

Oliver Hinder
Google Research
University of Pittsburgh
ohinder@pitt.edu

Haihao Lu
University of Chicago†
haihao.lu@chicagobooth.edu

Miles Lubin
Google Research
mlubin@google.com

Brendan O'Donoghue
DeepMind
bodonoghue@deepmind.com

Warren Schudy
Google Research
wschudy@google.com

Abstract

We present PDLP, a practical first-order method for linear programming (LP) that can solve to the high levels of accuracy that are expected in traditional LP applications. In addition, it can scale to very large problems because its core operation is matrix-vector multiplications. PDLP is derived by applying the primal-dual hybrid gradient (PDHG) method, popularized by Chambolle and Pock (2011), to a saddle-point formulation of LP. PDLP enhances PDHG for LP by combining several new techniques with older tricks from the literature; the enhancements include diagonal preconditioning, presolving, adaptive step sizes, and adaptive restarting. PDLP improves the state of the art for first-order methods applied to LP. We compare PDLP with SCS, an ADMM-based solver, on a set of 383 LP instances derived from MIPLIB 2017. With a target of 10^{-8} relative accuracy and 1 hour time limit, PDLP achieves a 6.3x reduction in the geometric mean of solve times and a 4.6x reduction in the number of instances unsolved (from 227 to 49). Furthermore, we highlight standard benchmark instances and a large-scale application (PageRank) where our open-source prototype of PDLP, written in Julia, outperforms a commercial LP solver.

[20] Applegate, D., et al. Practical Large-Scale Linear Programming using Primal-Dual Hybrid Gradient. 35th Conference on Neural Information Processing Systems (NeurIPS 2021). (2021).

cuPDLP

- GPU implementation of PDLP
- Performance comparable to commercial LP solvers

[9]

Algorithm 1: Restarted PDHG (after preconditioning)

Input: Initial point $z^{0,0}$;

Initialize outer loop counter $n \leftarrow 0$, total iterations $k \leftarrow 0$, step-size $\hat{\eta}^{0,0} \leftarrow 1/\|K\|_\infty$, primal weight $\omega^0 \leftarrow \text{InitializePrimalWeight}(c, q)$;

repeat

  $t \leftarrow 0$;

  repeat

    $z^{n,t+1}, \eta^{n,t+1}, \hat{\eta}^{n,t+1} \leftarrow \text{AdaptiveStepPDHG}(z^{n,t}, \omega^n, \hat{\eta}^{n,t}, k)$;

    $\bar{z}^{n,t+1} \leftarrow \frac{1}{\sum_{i=1}^{t+1} \eta^{n,i}} \sum_{i=1}^{k+1} \eta^{n,i} z^{n,i}$;

    $z_c^{n,t+1} \leftarrow \text{GetRestartCandidate}(z^{n,t+1}, \bar{z}^{n,t+1})$;

    $t \leftarrow t+1$, $k \leftarrow k+1$;

  until restart or termination criteria holds;

  restart the outer loop $z^{n+1,0} \leftarrow z_c^{n,t}$, $n \leftarrow n+1$;

  $\omega^n \leftarrow \text{PrimalWeightUpdate}(z^{n,0}, z^{n-1,0}, \omega^{n-1})$;

until termination criteria holds;

Output: $z^{n,0}$.

[9] NVIDIA. NVIDIA Tesla V100 GPU Architecture: The World's Most Advanced Data Center GPU. White paper. (2017).
[21] Lu, H. and Yang, J. cuPDLP.jl: A GPU implementation of restarted primal-dual hybrid gradient for linear programming in Julia. (2024).

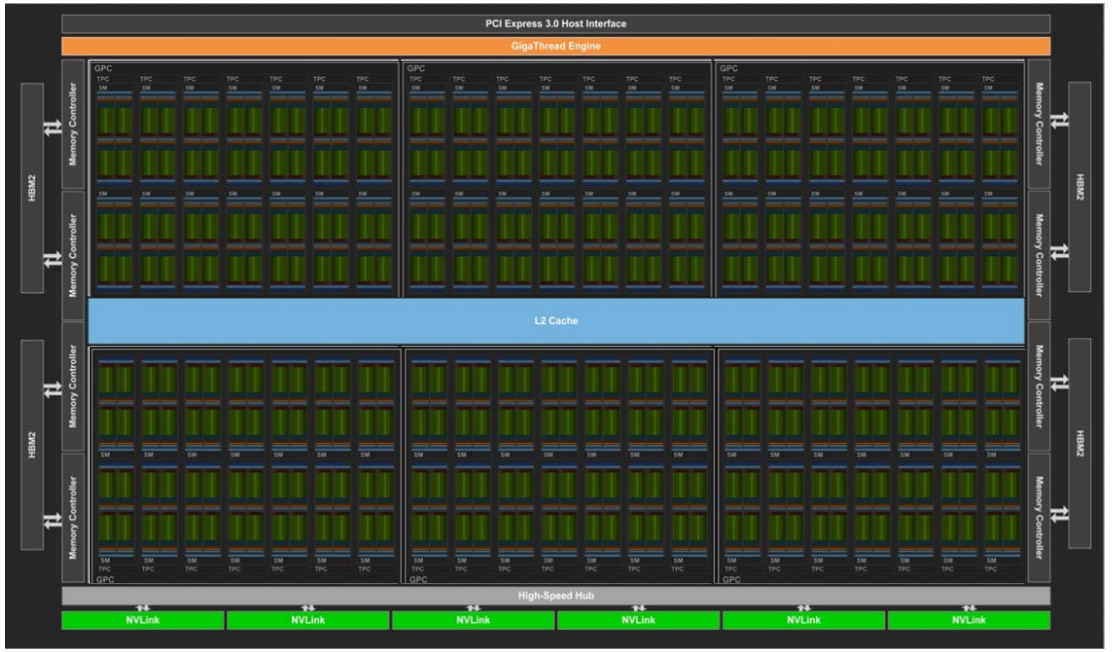
BatchPDLP.jl

- Custom implementation of PDLP designed for small-scale LPs
- Each LP is solved by a portion of the GPU

Algorithm 1: Single-Kernel PDLP

foreach *LP* **do**
  Input: An initial solution $z_i^{0,0}$;
  Initialize outer loop counter $n_i \leftarrow 0$, total iterations $k_i \leftarrow 0$, step size $\hat{\eta}_i^{0,0} \leftarrow 1/\|K_i\|_\infty$, primal weight $\omega_i^0 \leftarrow \texttt{InitializePrimalWeight}(c_i, q_i)$;
  repeat
    $t_i \leftarrow 0$;
    repeat
      $z_i^{n_i,t_i+1}, \eta_i^{n_i,t_i+1}, \hat{\eta}_i^{n_i,t_i+1} \leftarrow \texttt{AdaptivePDHGStep}(z_i^{n_i,t_i}, \omega_i^{n_i}, \hat{\eta}_i^{n_i,t_i}, k_i)$;
      $\bar{z}_i^{n_i,t_i+1} \leftarrow \frac{1}{\sum_{j=1}^{t_i+1} \eta_i^{n_i,j}} \sum_{j=1}^{t_i+1} \eta_i^{n_i,j} z_i^{n_i,j}$;
      $z_{c,i}^{n_i,t_i+1} \leftarrow \texttt{GetRestartCandidate}(z_i^{n_i,t_i+1}, \bar{z}_i^{n_i,t_i+1}, \omega_i^{n_i})$;
      $t_i \leftarrow t_i + 1, k_i \leftarrow k_i + 1$;
    until *restart or termination criteria holds*;
    Restart the outer loop. $z_i^{n_i+1,0} \leftarrow z_{c,i}^{n_i,t_i}, n_i \leftarrow n_i + 1$;
    $\omega_i^{n_i} \leftarrow \texttt{PrimalWeightUpdate}(z_i^{n_i,0}, z_i^{n_i-1,0}, \omega_i^{n_i-1})$;
  until *termination criteria holds*;
  Output: $z_i^{n_i,0}$.
end

[9]



[9] NVIDIA. NVIDIA Tesla V100 GPU Architecture: The World's Most Advanced Data Center GPU. White paper. (2017).
[22] Gottlieb, R.X., Alston, D., and Stuber, M.D. Re-Architected PDLP for Batch Parallelization of Linear Programs. Under Review

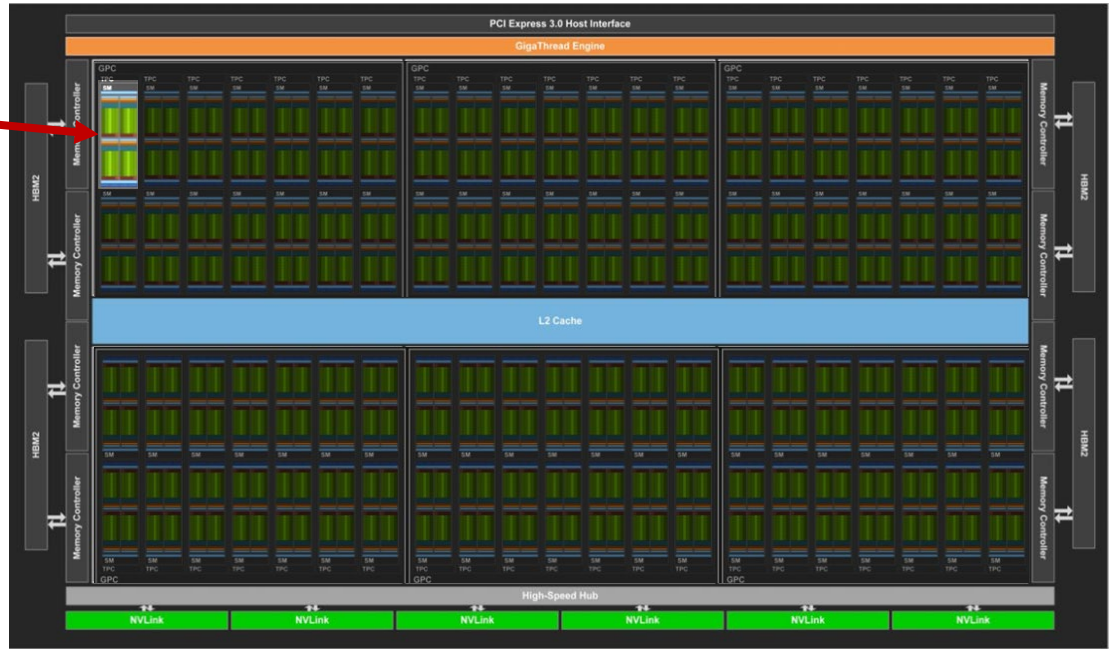
BatchPDLP.jl

- Custom implementation of PDLP designed for small-scale LPs
- Each LP is solved by a portion of the GPU

Algorithm 1: Single-Kernel PDLP

foreach *LP* **do**
  Input: An initial solution $z_i^{0,0}$;
  Initialize outer loop counter $n_i \leftarrow 0$, total iterations $k_i \leftarrow 0$, step size $\hat{\eta}_i^{0,0} \leftarrow 1/\|K_i\|_\infty$, primal weight $\omega_i^0 \leftarrow$ `InitializePrimalWeight`(c_i, q_i);
  repeat
    $t_i \leftarrow 0$;
    repeat
      $z_i^{n_i,t_i+1}, \eta_i^{n_i,t_i+1}, \hat{\eta}_i^{n_i,t_i+1} \leftarrow$ `AdaptivePDHGStep`$(z_i^{n_i,t_i}, \omega_i^{n_i}, \hat{\eta}_i^{n_i,t_i}, k_i)$;
      $\bar{z}_i^{n_i,t_i+1} \leftarrow \frac{1}{\sum_{j=1}^{t_i+1} \eta_i^{n_i,j}} \sum_{j=1}^{t_i+1} \eta_i^{n_i,j} z_i^{n_i,j}$;
      $z_{c,i}^{n_i,t_i+1} \leftarrow$ `GetRestartCandidate`$(z_i^{n_i,t_i+1}, \bar{z}_i^{n_i,t_i+1}, \omega_i^{n_i})$;
      $t_i \leftarrow t_i + 1, k_i \leftarrow k_i + 1$;
    until *restart or termination criteria holds*;
    Restart the outer loop. $z_i^{n_i+1,0} \leftarrow z_{c,i}^{n_i,t_i}, n_i \leftarrow n_i + 1$;
    $\omega_i^{n_i} \leftarrow$ `PrimalWeightUpdate`$(z_i^{n_i,0}, z_i^{n_i-1,0}, \omega_i^{n_i-1})$;
  until *termination criteria holds*;
  Output: $z_i^{n_i,0}$.
end

[9]



[9] NVIDIA. NVIDIA Tesla V100 GPU Architecture: The World's Most Advanced Data Center GPU. White paper. (2017).
[22] Gottlieb, R.X., Alston, D., and Stuber, M.D. Re-Architected PDLP for Batch Parallelization of Linear Programs. Under Review

BatchPDLP.jl

- Custom implementation of PDLP designed for small-scale LPs
- Each LP is solved by a portion of the GPU

Algorithm 1: Single-Kernel PDLP

foreach *LP* **do**
  Input: An initial solution $z_i^{0,0}$;
  Initialize outer loop counter $n_i \leftarrow 0$, total iterations $k_i \leftarrow 0$, step size $\hat{\eta}_i^{0,0} \leftarrow 1/\|K_i\|_\infty$, primal weight $\omega_i^0 \leftarrow \texttt{InitializePrimalWeight}(c_i, q_i)$;
  repeat
    $t_i \leftarrow 0$;
    repeat
      $z_i^{n_i,t_i+1}, \eta_i^{n_i,t_i+1}, \hat{\eta}_i^{n_i,t_i+1} \leftarrow \texttt{AdaptivePDHGStep}(z_i^{n_i,t_i}, \omega_i^{n_i}, \hat{\eta}_i^{n_i,t_i}, k_i)$;
      $\bar{z}_i^{n_i,t_i+1} \leftarrow \frac{1}{\sum_{j=1}^{t_i+1} \eta_i^{n_i,j}} \sum_{j=1}^{t_i+1} \eta_i^{n_i,j} z_i^{n_i,j}$;
      $z_{c,i}^{n_i,t_i+1} \leftarrow \texttt{GetRestartCandidate}(z_i^{n_i,t_i+1}, \bar{z}_i^{n_i,t_i+1}, \omega_i^{n_i})$;
      $t_i \leftarrow t_i + 1, k_i \leftarrow k_i + 1$;
    until *restart or termination criteria holds*;
    Restart the outer loop. $z_i^{n_i+1,0} \leftarrow z_{c,i}^{n_i,t_i}, n_i \leftarrow n_i + 1$;
    $\omega_i^{n_i} \leftarrow \texttt{PrimalWeightUpdate}(z_i^{n_i,0}, z_i^{n_i-1,0}, \omega_i^{n_i-1})$;
  until *termination criteria holds*;
  Output: $z_i^{n_i,0}$.
end

[9]

[9] NVIDIA. NVIDIA Tesla V100 GPU Architecture: The World's Most Advanced Data Center GPU. White paper. (2017).
[22] Gottlieb, R.X., Alston, D., and Stuber, M.D. Re-Architected PDLP for Batch Parallelization of Linear Programs. Under Review

BatchPDLP.jl

- Custom implementation of PDLP designed for small-scale LPs
- Each LP is solved by a portion of the GPU

Algorithm 1: Single-Kernel PDLP

foreach *LP* **do**
  Input: An initial solution $z_i^{0,0}$;
  Initialize outer loop counter $n_i \leftarrow 0$, total iterations $k_i \leftarrow 0$, step size $\hat{\eta}_i^{0,0} \leftarrow 1/\|K_i\|_\infty$, primal weight $\omega_i^0 \leftarrow \texttt{InitializePrimalWeight}(c_i, q_i)$;
  repeat
    $t_i \leftarrow 0$;
    repeat
      $z_i^{n_i,t_i+1}, \eta_i^{n_i,t_i+1}, \hat{\eta}_i^{n_i,t_i+1} \leftarrow \texttt{AdaptivePDHGStep}(z_i^{n_i,t_i}, \omega_i^{n_i}, \hat{\eta}_i^{n_i,t_i}, k_i)$;
      $\bar{z}_i^{n_i,t_i+1} \leftarrow \frac{1}{\sum_{j=1}^{t_i+1} \eta_i^{n_i,j}} \sum_{j=1}^{t_i+1} \eta_i^{n_i,j} z_i^{n_i,j}$;
      $z_{c,i}^{n_i,t_i+1} \leftarrow \texttt{GetRestartCandidate}(z_i^{n_i,t_i+1}, \bar{z}_i^{n_i,t_i+1}, \omega_i^{n_i})$;
      $t_i \leftarrow t_i + 1, k_i \leftarrow k_i + 1$;
    until *restart or termination criteria holds*;
    Restart the outer loop. $z_i^{n_i+1,0} \leftarrow z_{c,i}^{n_i,t_i}, n_i \leftarrow n_i + 1$;
    $\omega_i^{n_i} \leftarrow \texttt{PrimalWeightUpdate}(z_i^{n_i,0}, z_i^{n_i-1,0}, \omega_i^{n_i-1})$;
  until *termination criteria holds*;
  Output: $z_i^{n_i,0}$.
end

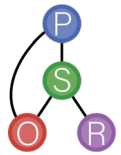
Other LPs continue even if one finishes

[9]

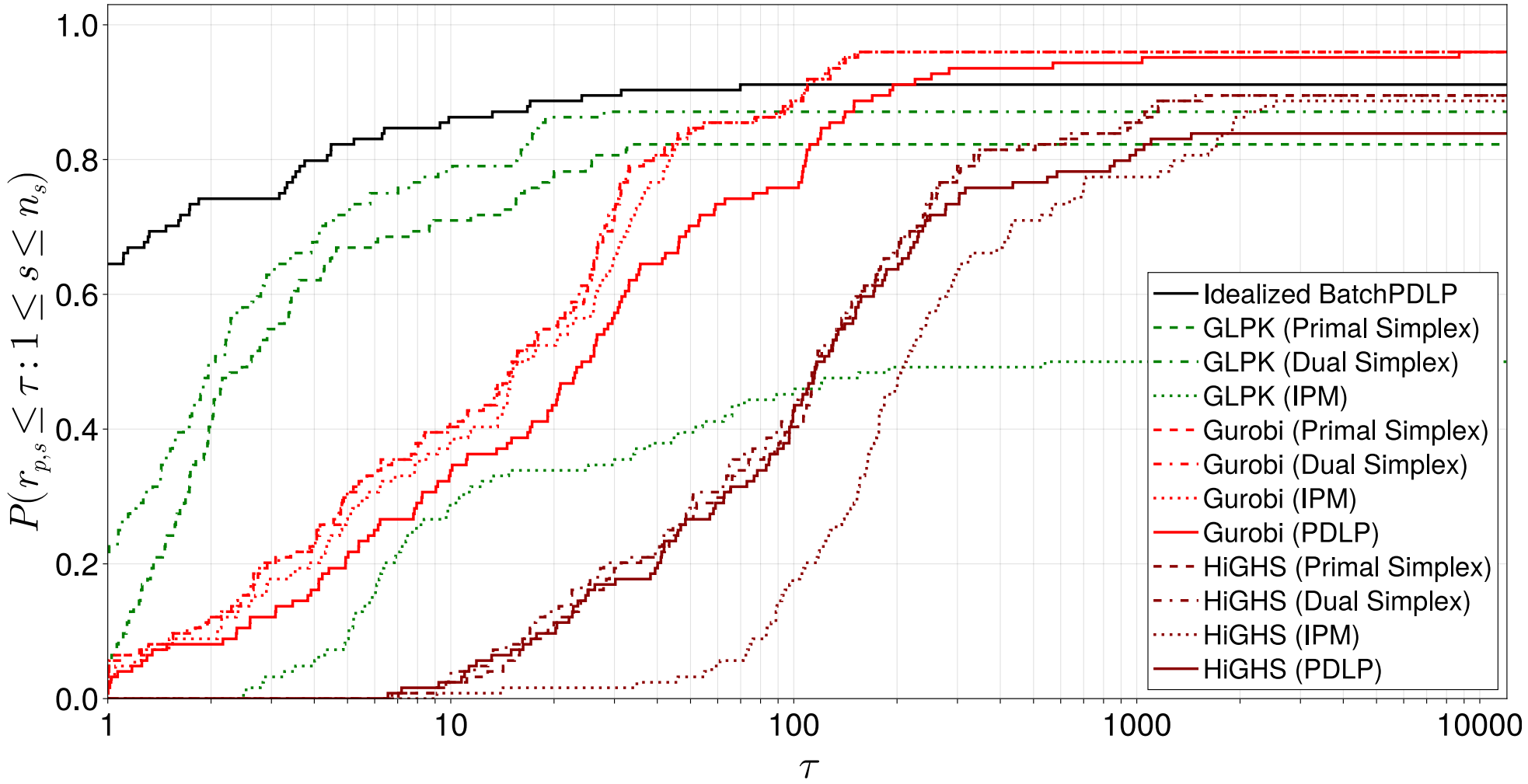
[9] NVIDIA. NVIDIA Tesla V100 GPU Architecture: The World's Most Advanced Data Center GPU. White paper. (2017).
[22] Gottlieb, R.X., Alston, D., and Stuber, M.D. Re-Architected PDLP for Batch Parallelization of Linear Programs. Under Review

BatchPDLP.jl Benchmarking

- 124 NLPs from MINLPLib
  - $\leq$ 100 variables
  - $\leq$ 1,000 constraints
  - Generate 10,000 LP subproblems for each

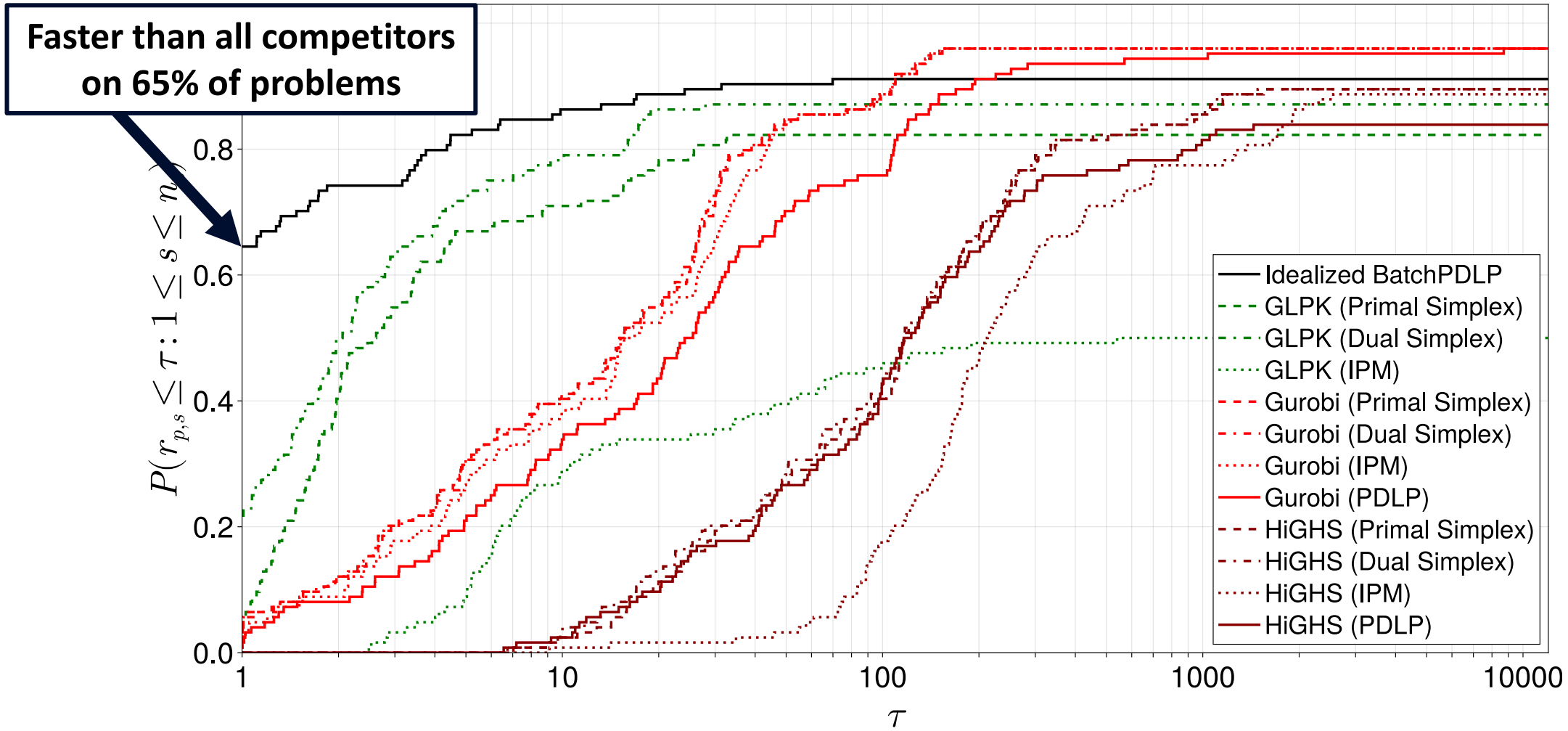


BatchPDLP.jl Benchmarking



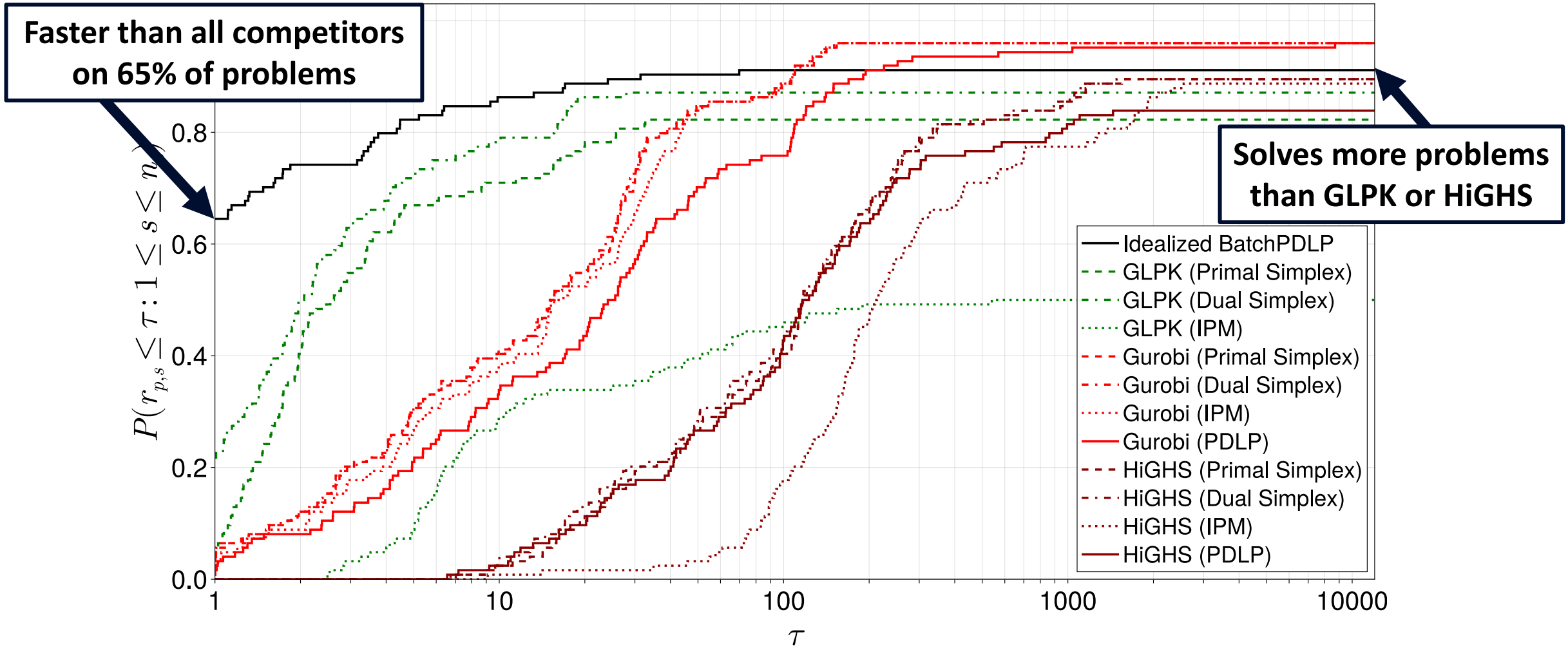
[22] Gottlieb, R.X., Alston, D., and Stuber, M.D. Re-Architected PDLP for Batch Parallelization of Linear Programs. Under Review.

BatchPDLP.jl Benchmarking



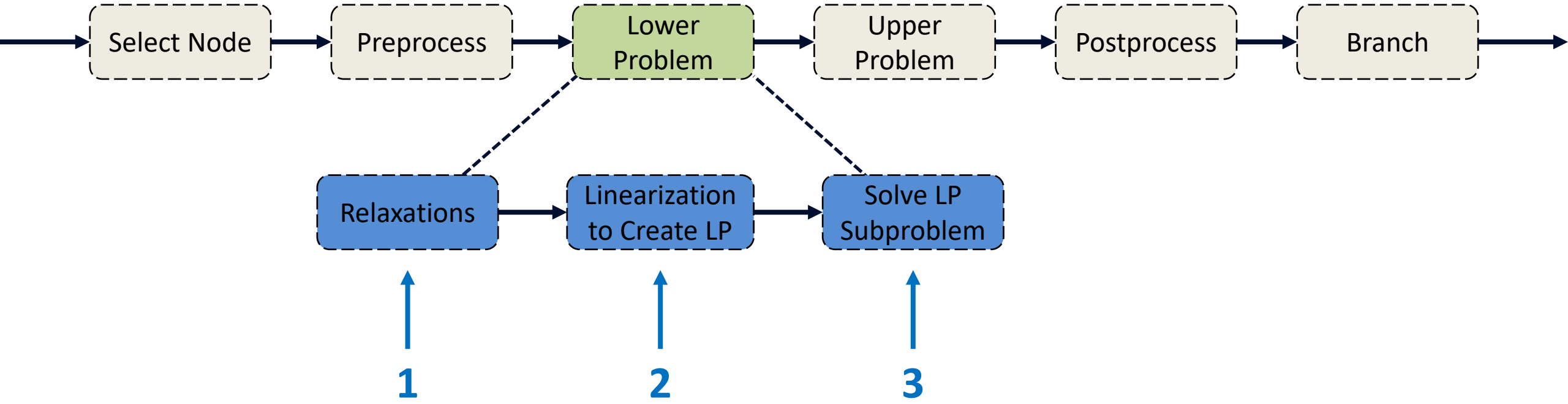
[22] Gottlieb, R.X., Alston, D., and Stuber, M.D. Re-Architected PDLP for Batch Parallelization of Linear Programs. Under Review.

BatchPDLP.jl Benchmarking

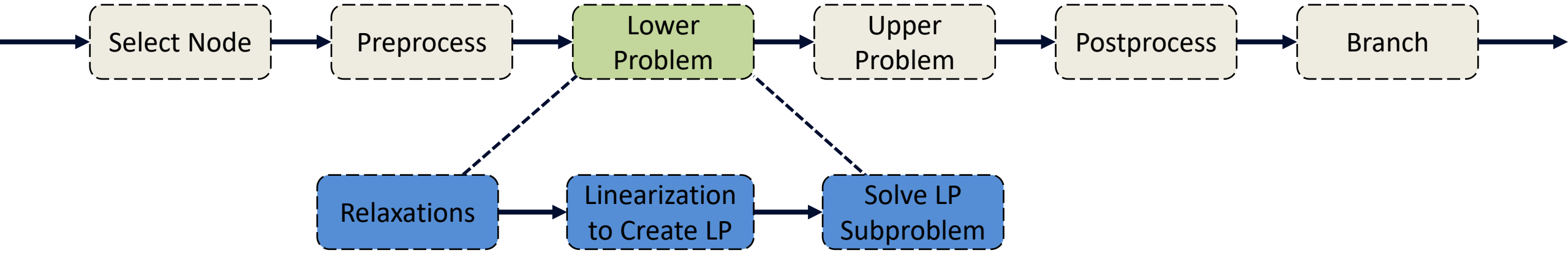


[22] Gottlieb, R.X., Alston, D., and Stuber, M.D. Re-Architected PDLP for Batch Parallelization of Linear Programs. Under Review.

Tasks for GPU Parallelization



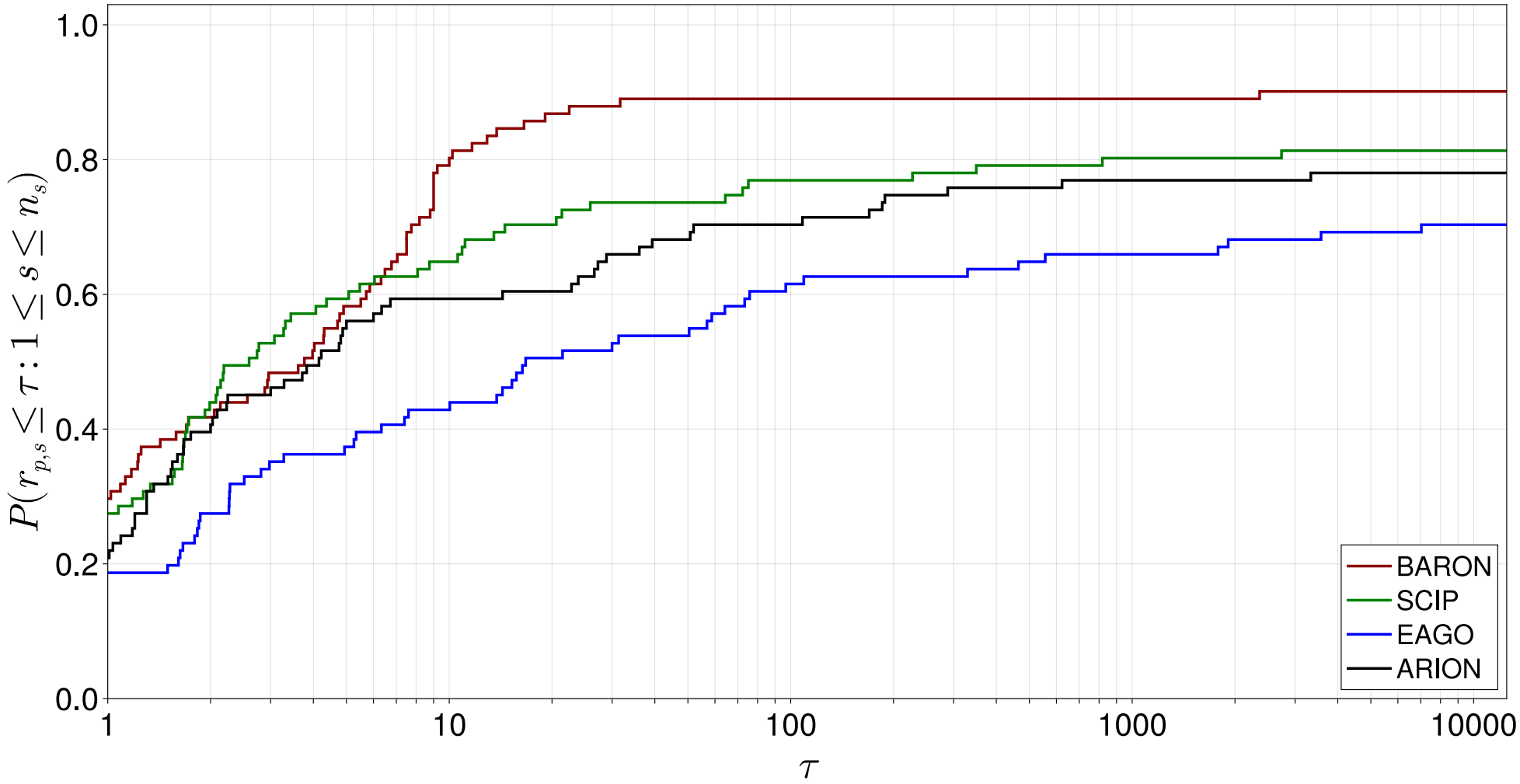
ARION.jl



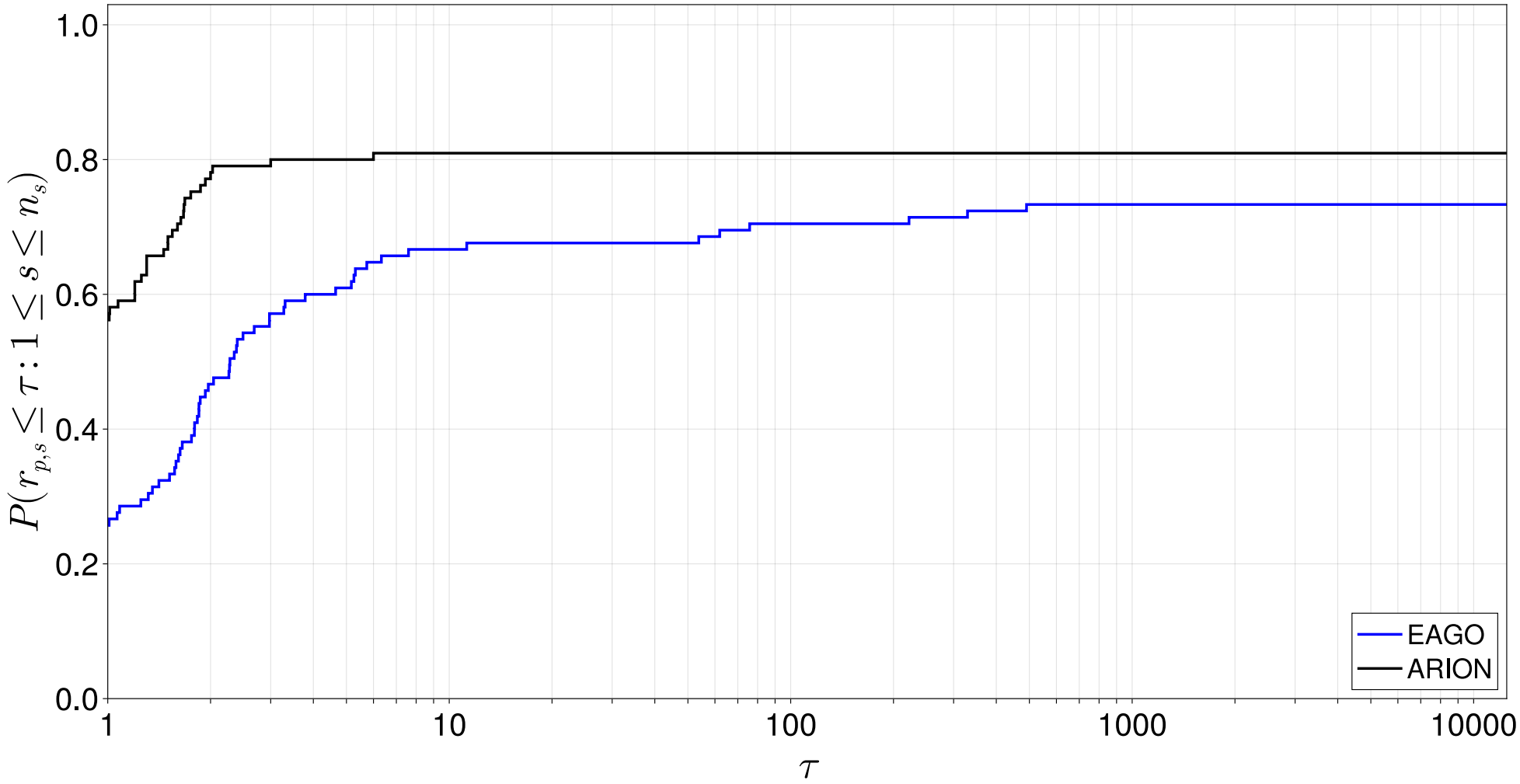
ARION.jl Private

Accelerated and Readily-scalable Integrated CPU-GPU Optimization of Nonlinear programs

ARION.jl Benchmarking



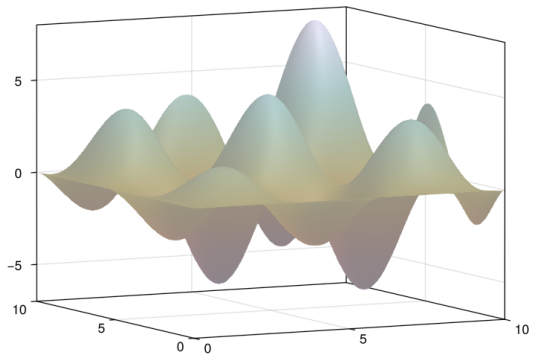
ARION.jl Benchmarking

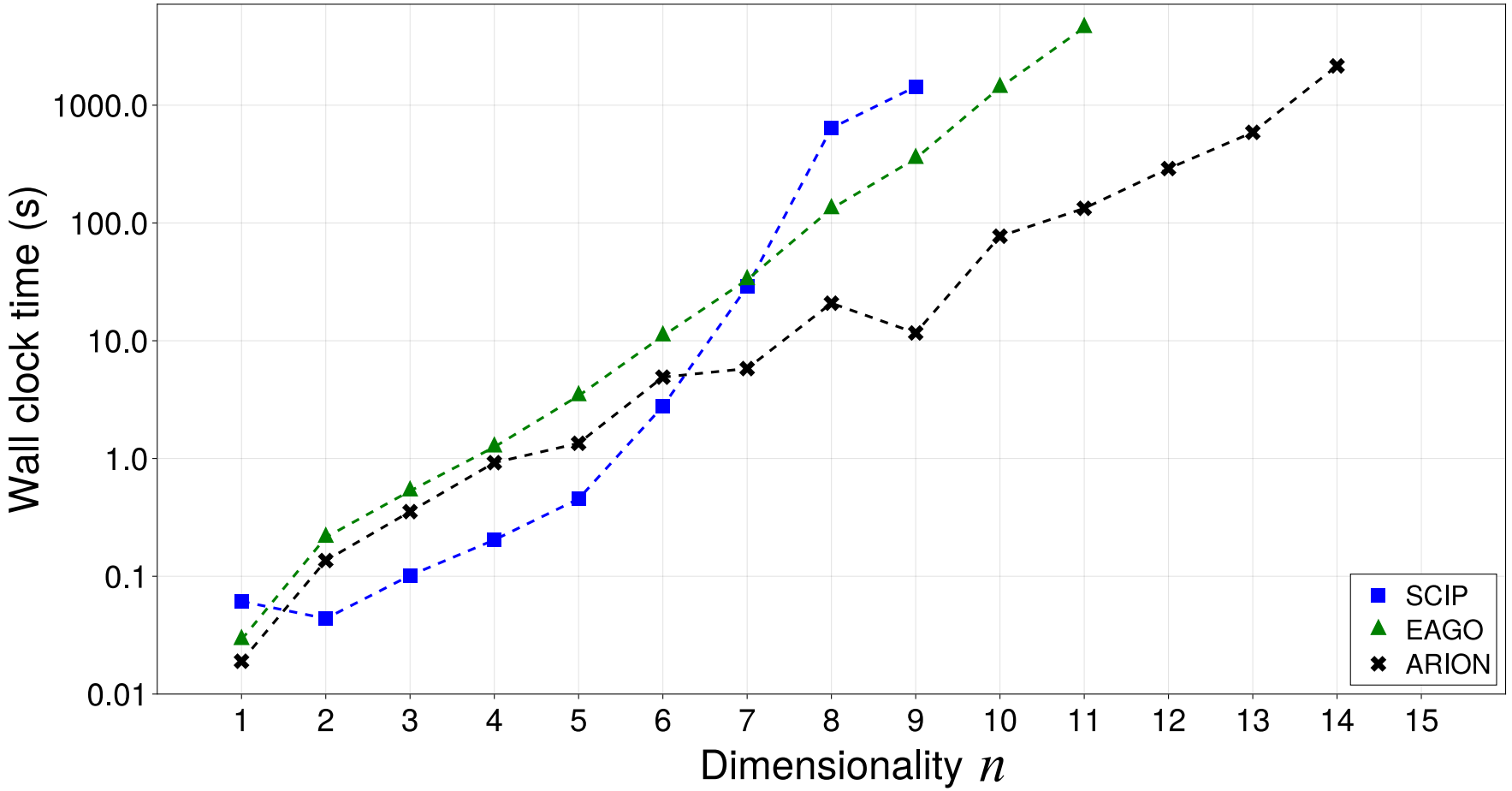


Scalable Benchmark Problem

$$f^* = \min_{\mathbf{x}} \prod_{i=1}^{n} \sin(x_i)\sqrt{x_i}$$
$$\text{s.t. } \mathbf{x} \in [0, 10]^n$$

- Difficulty scales with dimension



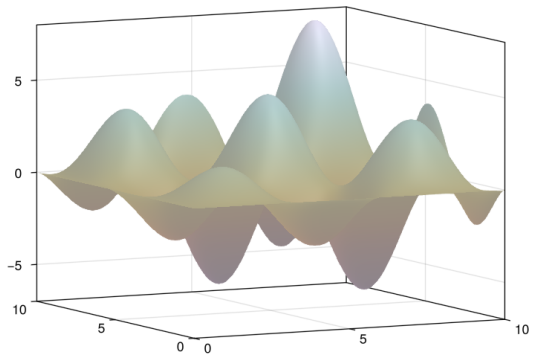


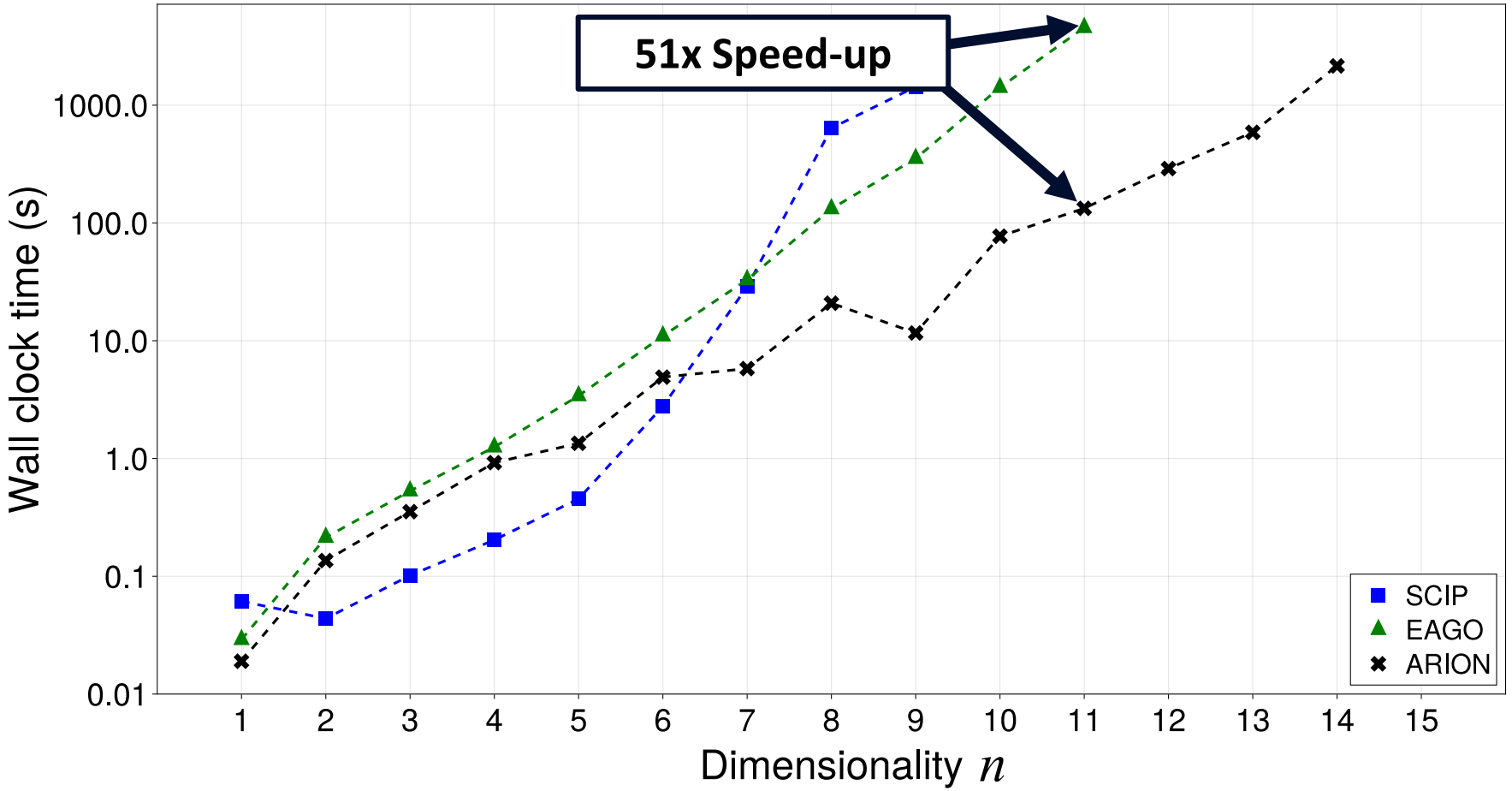
Scalable Benchmark Problem

$$f^* = \min_{\mathbf{x}} \prod_{i=1}^{n} \sin(x_i)\sqrt{x_i}$$

$$\text{s.t. } \mathbf{x} \in [0, 10]^n$$

- Difficulty scales with dimension

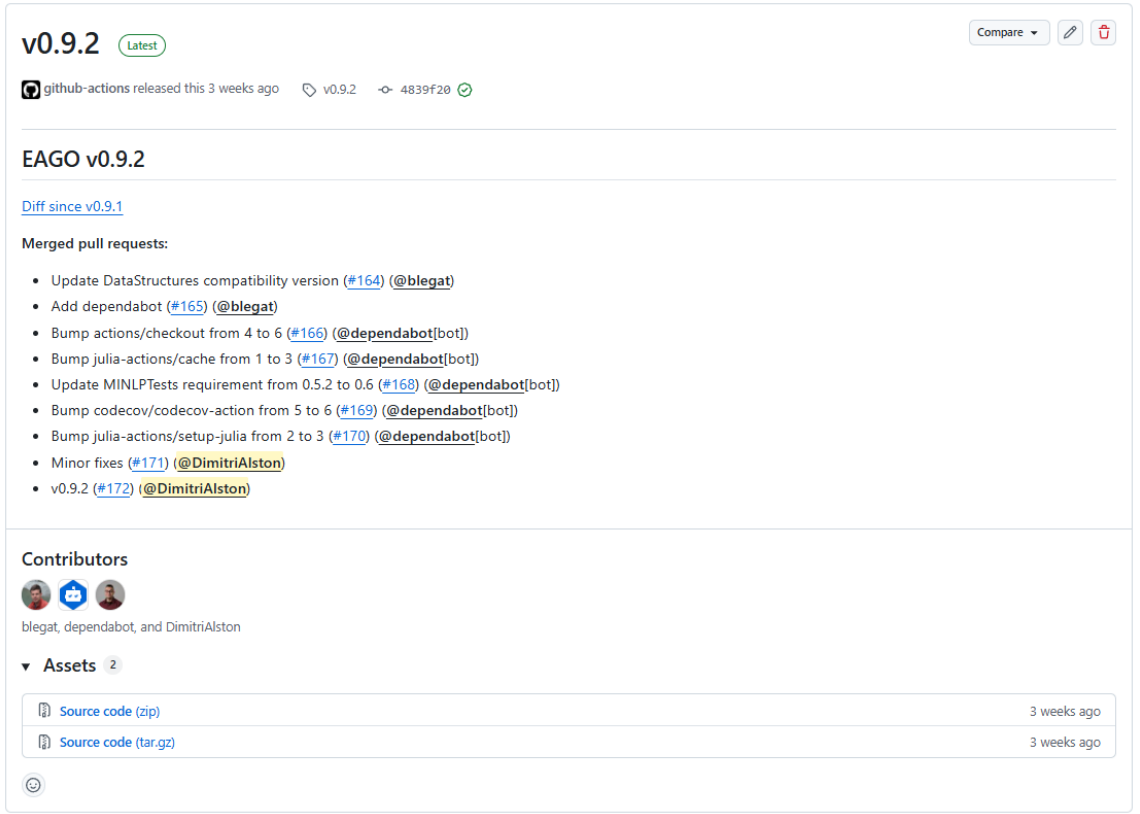




Current State of EAGO

EAGO v0.9.x

- Updated domain reduction routines
  - Constraint propagation
  - Optimality-based bounds-tightening
  - Feasibility-based bounds-tightening



v0.9.2 Latest

Compare

github-actions released this 3 weeks ago · v0.9.2 · 4839f20

EAGO v0.9.2

Diff since v0.9.1

Merged pull requests:

- Update DataStructures compatibility version (#164) (@blegat)
- Add dependabot (#165) (@blegat)
- Bump actions/checkout from 4 to 6 (#166) (@dependabot[bot])
- Bump julia-actions/cache from 1 to 3 (#167) (@dependabot[bot])
- Update MINLPTests requirement from 0.5.2 to 0.6 (#168) (@dependabot[bot])
- Bump codecov/codecov-action from 5 to 6 (#169) (@dependabot[bot])
- Bump julia-actions/setup-julia from 2 to 3 (#170) (@dependabot[bot])
- Minor fixes (#171) (@DimitriAlston)
- v0.9.2 (#172) (@DimitriAlston)

Contributors

blegat, dependabot, and DimitriAlston

Assets 2

Source code (zip) 3 weeks ago

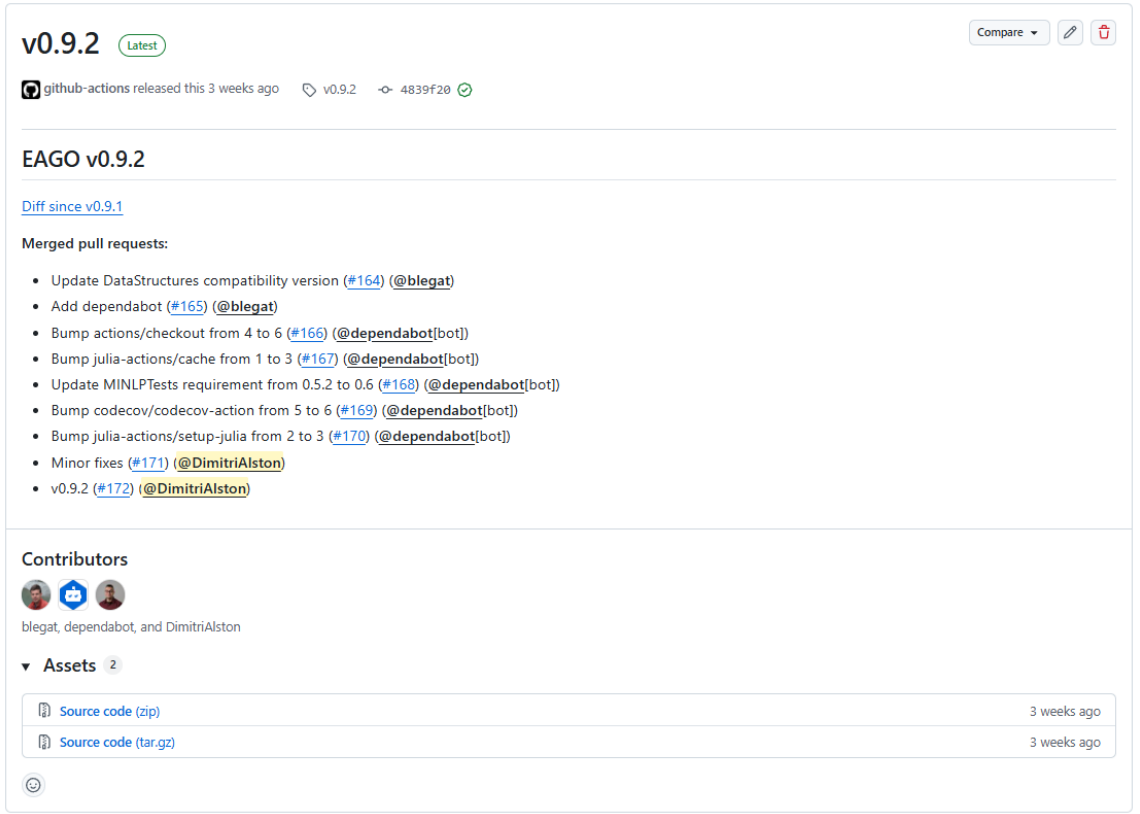
Source code (tar.gz) 3 weeks ago

Current State of EAGO

EAGO v0.9.x

- Updated domain reduction routines
  - Constraint propagation
  - Optimality-based bounds-tightening
  - Feasibility-based bounds-tightening

EAGO v0.10.x



v0.9.2 Latest

Compare

github-actions released this 3 weeks ago · v0.9.2 · 4839f20

EAGO v0.9.2

Diff since v0.9.1

Merged pull requests:

- Update DataStructures compatibility version (#164) (@blegat)
- Add dependabot (#165) (@blegat)
- Bump actions/checkout from 4 to 6 (#166) (@dependabot[bot])
- Bump julia-actions/cache from 1 to 3 (#167) (@dependabot[bot])
- Update MINLPTests requirement from 0.5.2 to 0.6 (#168) (@dependabot[bot])
- Bump codecov/codecov-action from 5 to 6 (#169) (@dependabot[bot])
- Bump julia-actions/setup-julia from 2 to 3 (#170) (@dependabot[bot])
- Minor fixes (#171) (@DimitriAlston)
- v0.9.2 (#172) (@DimitriAlston)

Contributors

blegat, dependabot, and DimitriAlston

Assets 2

Source code (zip) 3 weeks ago

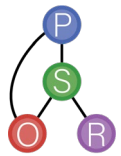
Source code (tar.gz) 3 weeks ago

Conclusion

- SIMT architecture can accelerate B&B lower-bounding tasks
  - Simultaneous relaxation evaluation
  - Simultaneous solutions to small-scale LPs

- Integration within EAGO platform allows minimal CPU-GPU data transfer

- General nonconvex NLPs can be accelerated using specialized SIMT architecture (GPUs)



Acknowledgements

Members of the Process Systems and Operations Research Laboratory at the University of Connecticut (https://psor.uconn.edu)

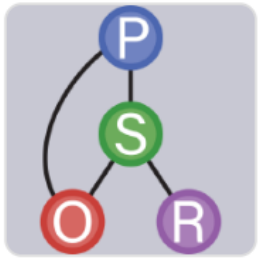
Funding:

National Science Foundation, Award No.: 2051084

Pratt & Whitney Endowed Professorship

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the United States Government.

Questions?



Process Systems and Operations Research Laboratory

The PSOR Laboratory at UConn develops numerical analysis methods and software for process systems engineering applications.

27 followers | University of Connecticut, Storrs, CT, ... | https://psor.uconn.edu | stuber@uconn.edu

https://psor.uconn.edu

https://www.github.com/PSORLab