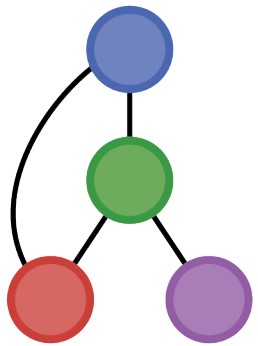
UCONN
UNIVERSITY OF CONNECTICUT

Scaling Deterministic Global Optimization with SIMT Architectures

Robert Gottlieb, Dimitri Alston, **Matthew Stuber**
University of Connecticut

March 22, 2026



Process Systems and Operations Research Laboratory

Atlanta, GA

Key Contributors

Robert X. Gottlieb
PhD Candidate
Primary Developer

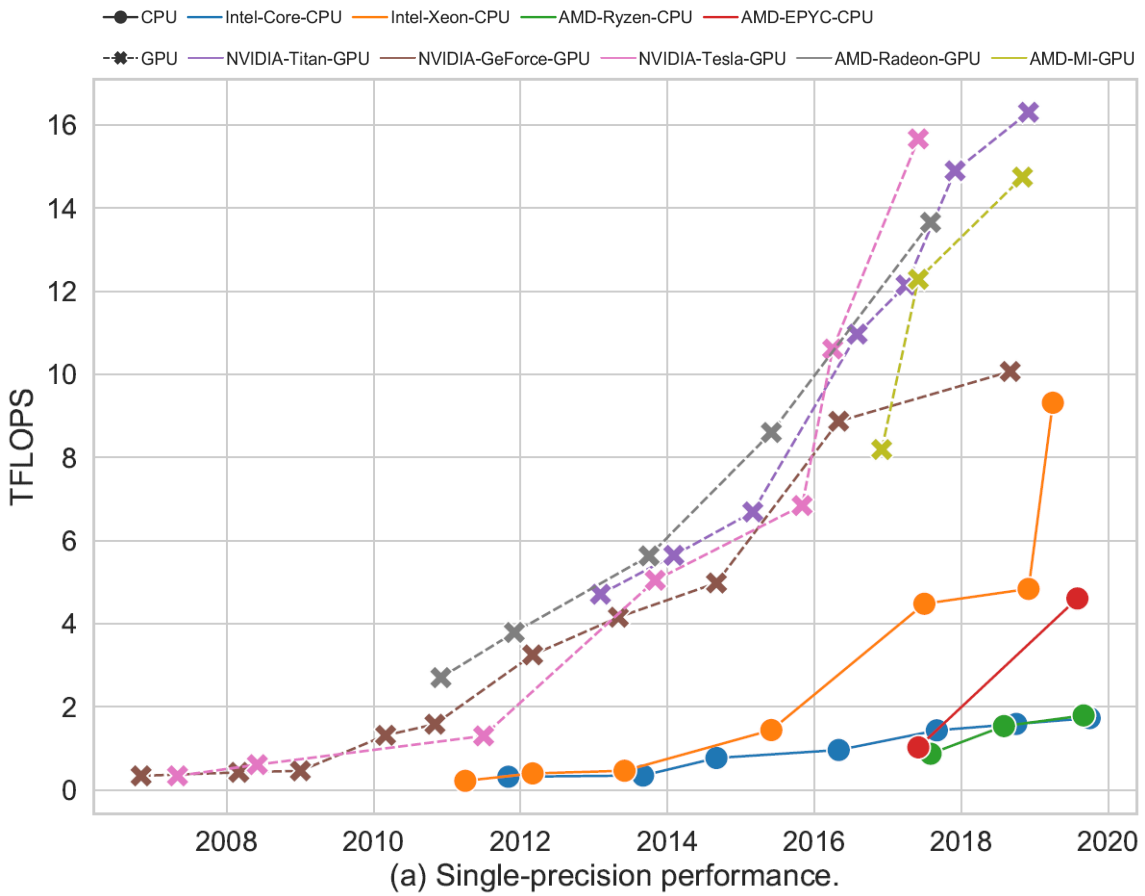
Dimitri Alston
PhD Student

PSOR Lab
Dept. of Chemical and Biomolecular Engineering
University of Connecticut

SIMT Architecture

Graphics Processing Units (GPUs)

- Graphics rendering
- Machine learning model training[2]
  - Generative AI
- Data analysis[3]
- Large-scale simulations[4]
  - Molecular dynamics
  - CFD modeling
- […]



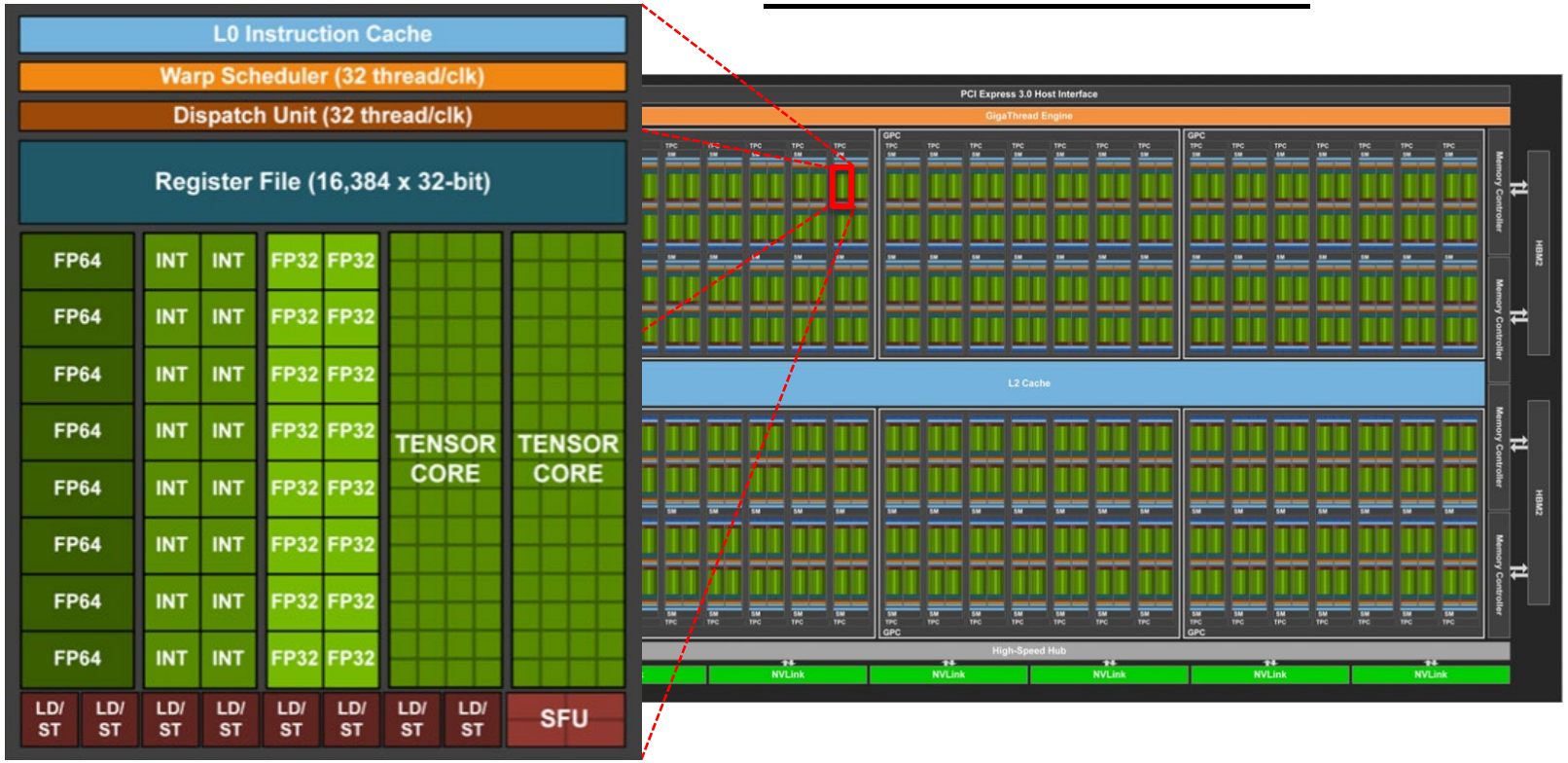
(a) Single-precision performance.

1. Sun, Y., et al. **Summarizing CPU and GPU design trends with product data.** *arXiv*, (2019). arXiv: 1911.11313
2. Steinkraus, D., et al. **Using GPUs for machine learning algorithms**. *Eighth International Conference on Document Analysis and Recognition*, Seoul, South Korea, 2: 1115-1120 (2005).
3. Singh, H., et al. **GPU and CUDA in Hard Computing Approaches: Analytical Review**. *Proceedings of ICRIC 2019*, 177-196 (2020).
4. Stone, J.E., et al. **GPU-accelerated molecular modeling coming of age.** *Journal of Molecular Graphics and Modelling*, 29(2):116-125 (2010).

GPU Architecture

- GPUs are composed of **thousands of "cores"**
- Single instructions are executed by **many cores simultaneously** on different portions of data
- Good performance requires **parallelization**

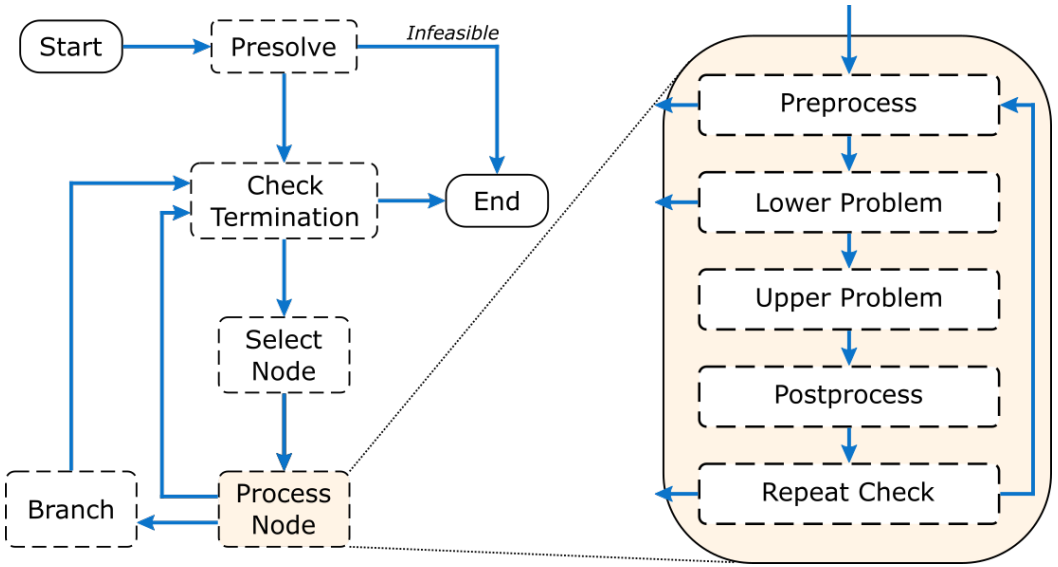
GPU Architecture[5]



5. NVIDIA. **NVIDIA Tesla V100 GPU Architecture: The World's Most Advanced Data Center GPU** [White paper]. NVIDIA (2017).

Aligning B&B with GPUs

B&B Algorithm[6]



?

GPU Architecture[5]

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

Aligning B&B with GPUs

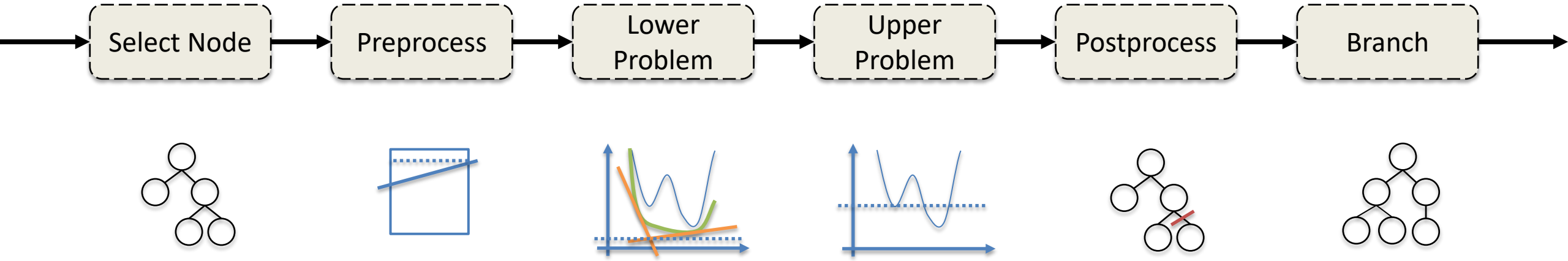
Global Solver	Uses GPU
BARON[7]	No
ANTIGONE[8]	No
SCIP[9]	No
MAiNGO[10]	Active Dev (2025)
EAGO[6]	Active Dev (2022)
[…]	[…]

?

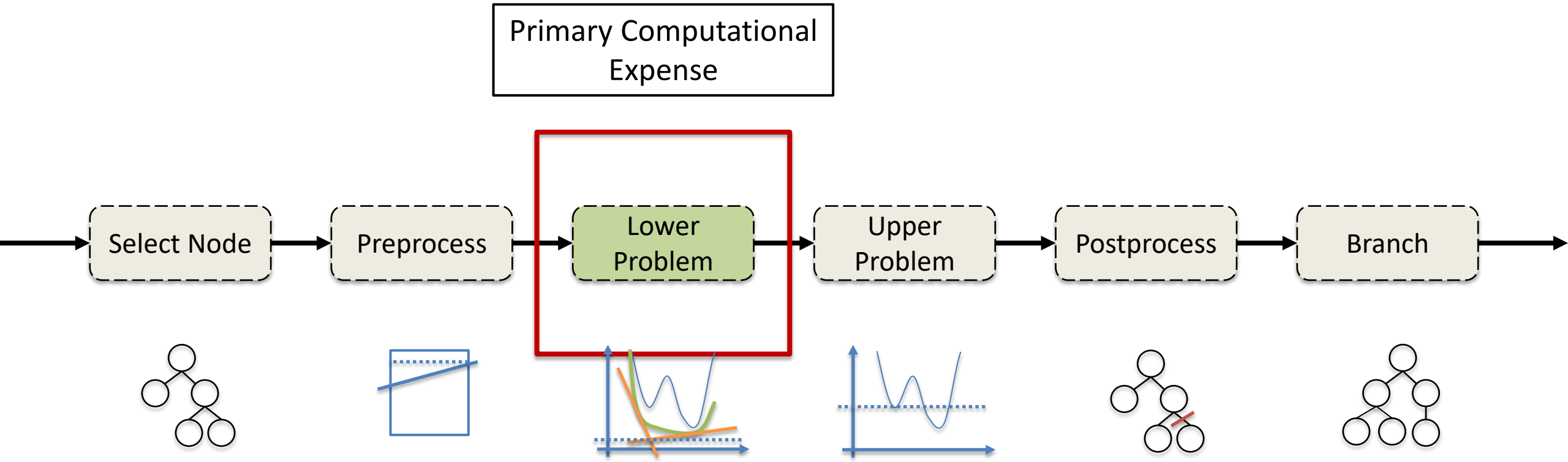
GPU Architecture[5]

5. NVIDIA. **NVIDIA Tesla V100 GPU Architecture: The World's Most Advanced Data Center GPU** [White paper]. NVIDIA (2017).
6. Wilhelm, M.E. and Stuber, M.D. **EAGO.jl: easy advanced global optimization in Julia.** *Optimization Methods and Software*, 37(2):425–450 (2022).
7. Sahinidis, N.V. **BARON: A general purpose global optimization software package.** *Journal of Global Optimization*, 8(2):201–205 (1996).
8. 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).
9. 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).
10. Bongartz, D., et al. **MAiNGO - McCormick-based Algorithm for mixed-integer Nonlinear Global Optimization.** Technical report, RWTH-Aachen (2018). URL https://www.avt.rwth-aachen.de/global/show_document.asp?id=aaaaaaaaabclahw.

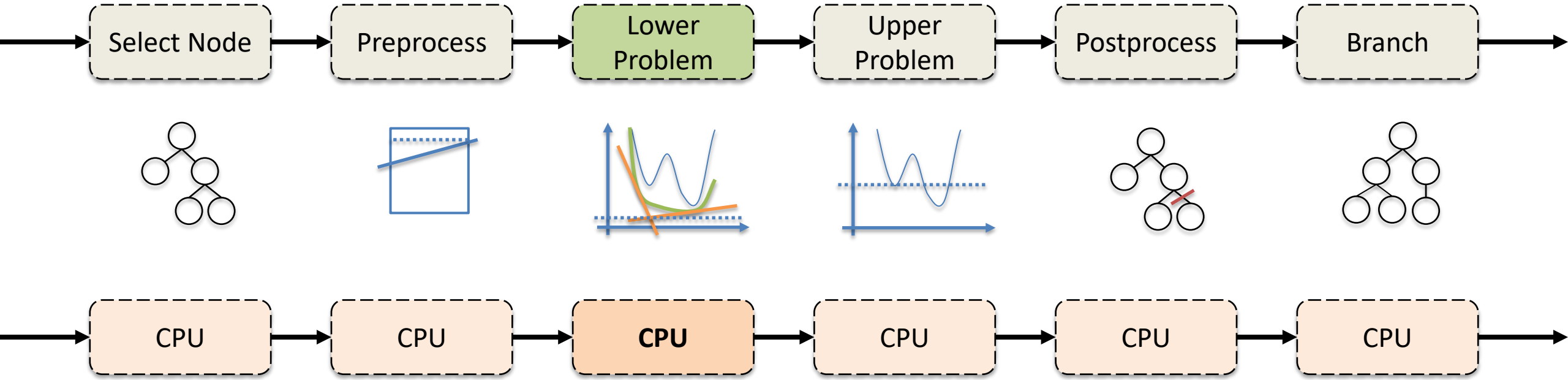
Serial CPU B&B



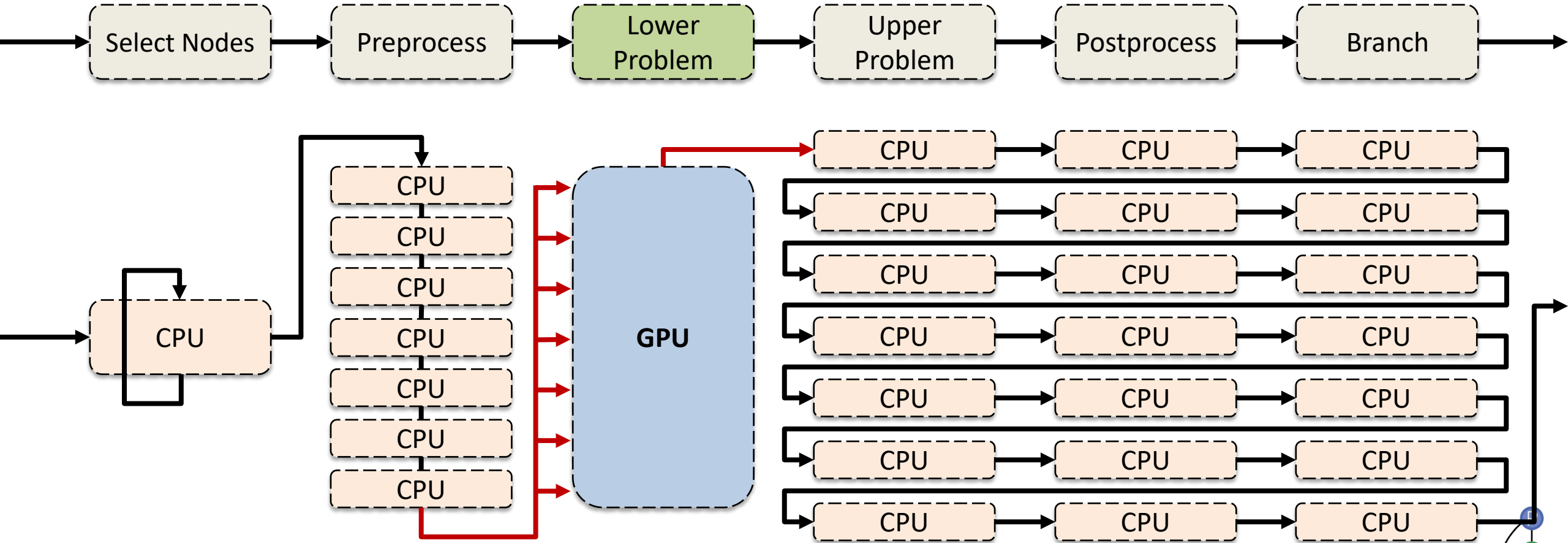
Serial CPU B&B



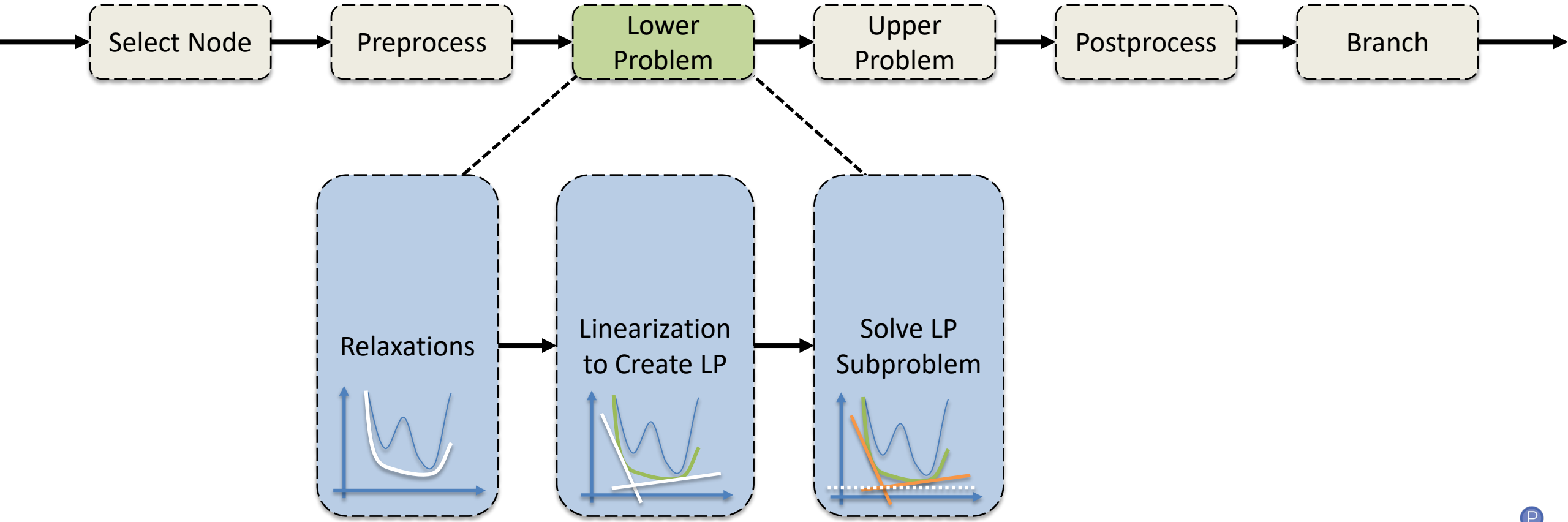
Serial CPU B&B



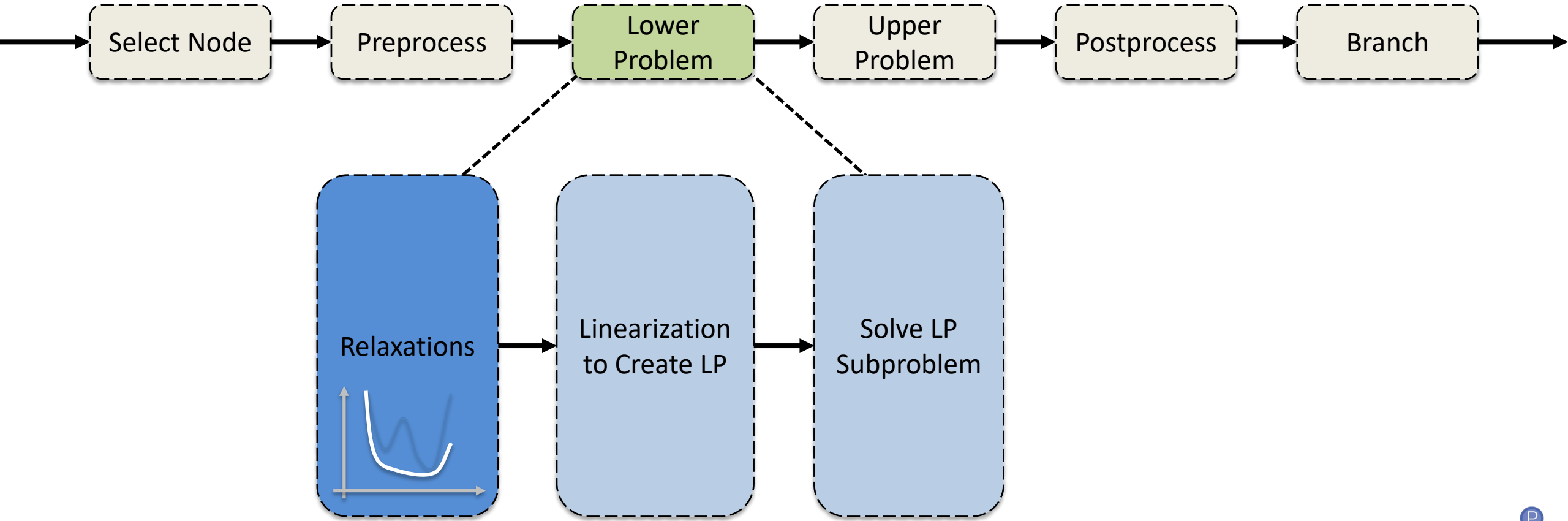
Integrated CPU-GPU B&B



Parallelization Targets



Parallelization Targets

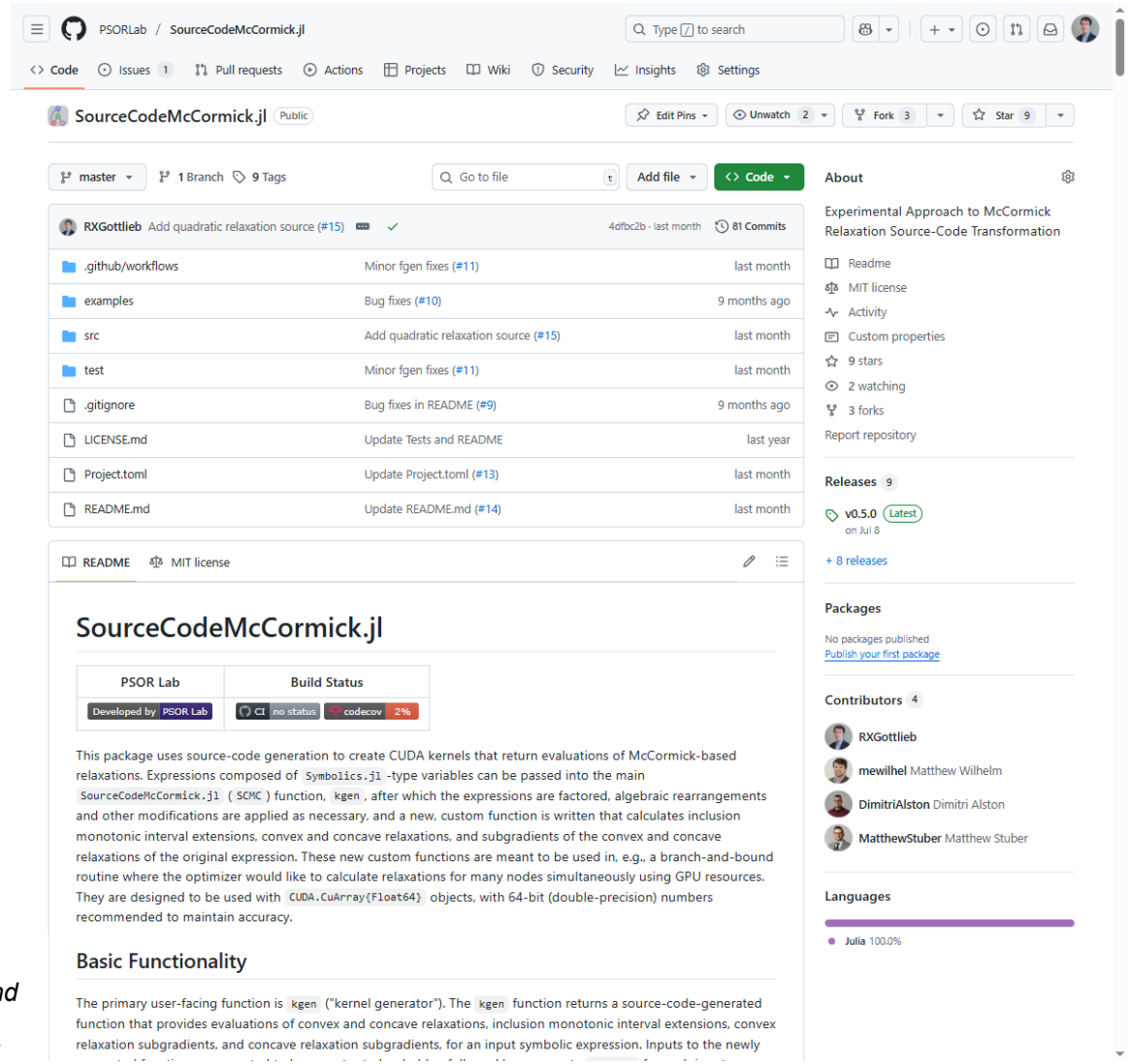


SourceCodeMcCormick.jl

SourceCodeMcCormick.jl (SCMC)[11,12]

- Open-source tool for GPU-accelerated McCormick relaxations

- Calculates:
  - Inclusion monotonic interval extensions
  - McCormick relaxations
  - Subgradients of McCormick relaxations

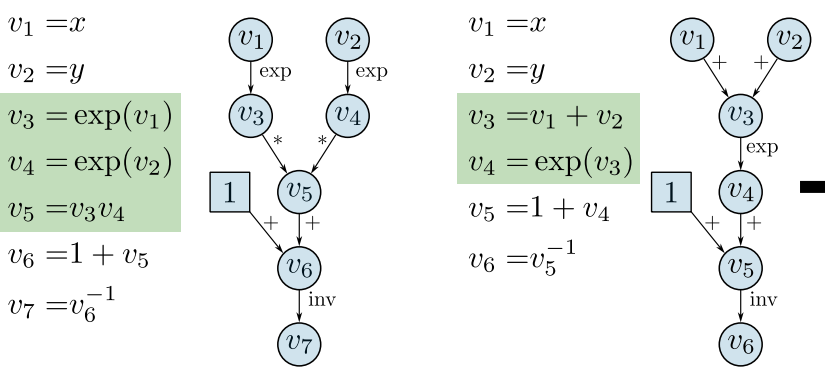


11. Gottlieb, R.X., Xu, P., and Stuber, M.D. **Automatic Source Code Generation for Deterministic Global Optimization with Parallel Architectures.** *Optimization Methods and Software*, 1–39 (2024).
12. Gottlieb, R.X. and Stuber, M.D. **Automatic Generation of GPU Kernels for Evaluators of McCormick-Based Relaxations and Subgradients.** Under Review, (2025).

SourceCodeMcCormick.jl

- Generates customized CUDA kernels to calculate **many relaxations at once**

```
using SourceCodeMcCormick
@variables x, y
func = kgen(1 / (1 + exp(x)*exp(y)))
```

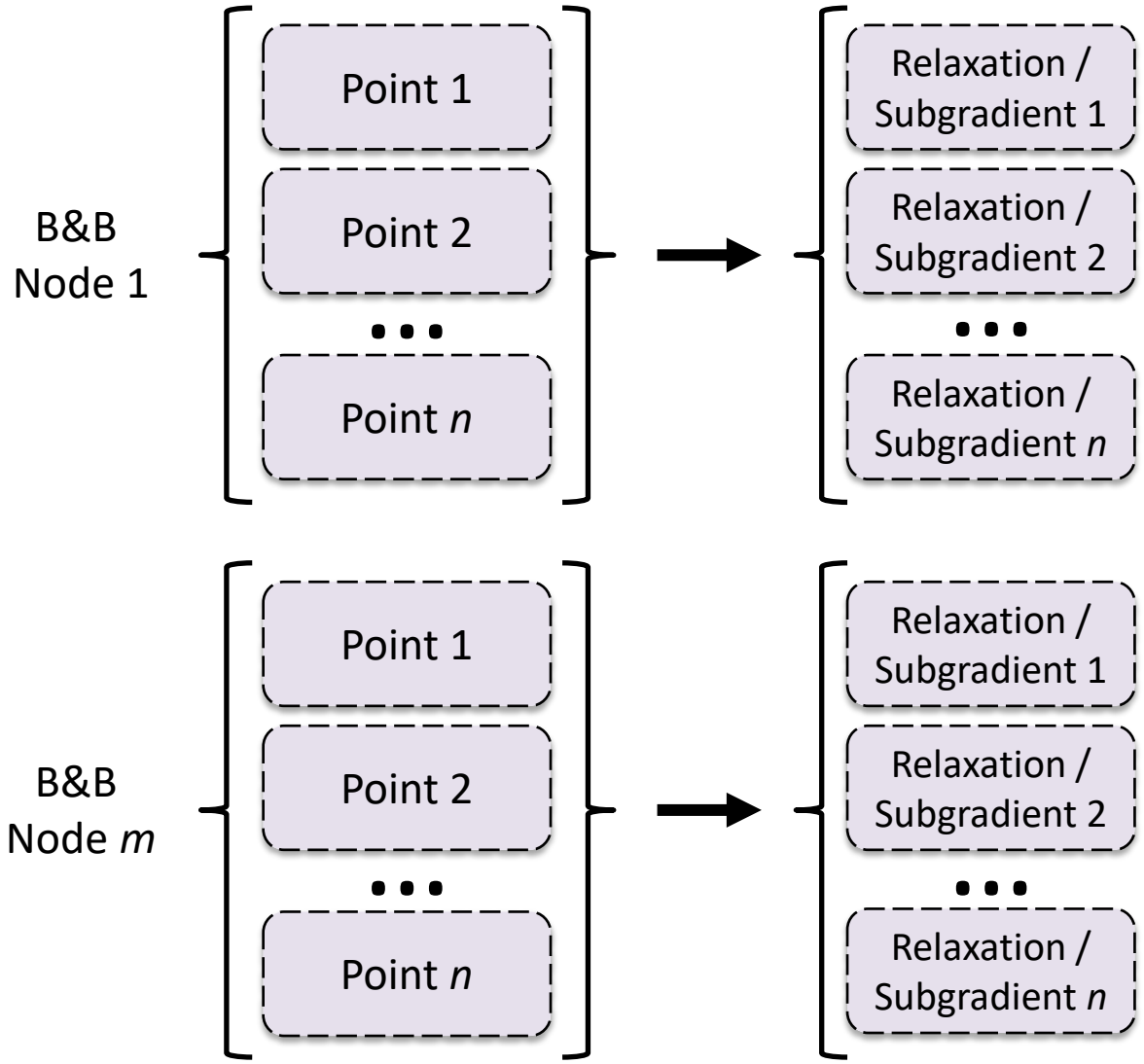



```julia
# Generated at 2025-08-22T17:22:03.345

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

function f_B3WZmzXUeWL_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)
    while idx <= Int32(size(OUT,1))
        ################################
        ## Addition of Two Variables ##
        ################################

        # Reset the column counter
        col = Int32(1)

        # Begin rule
```

Automatically Generated

11. 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).
12. Gottlieb, R.X. and Stuber, M.D. **Automatic Generation of GPU Kernels for Evaluators of McCormick-Based Relaxations and Subgradients.** Under Review, (2025).

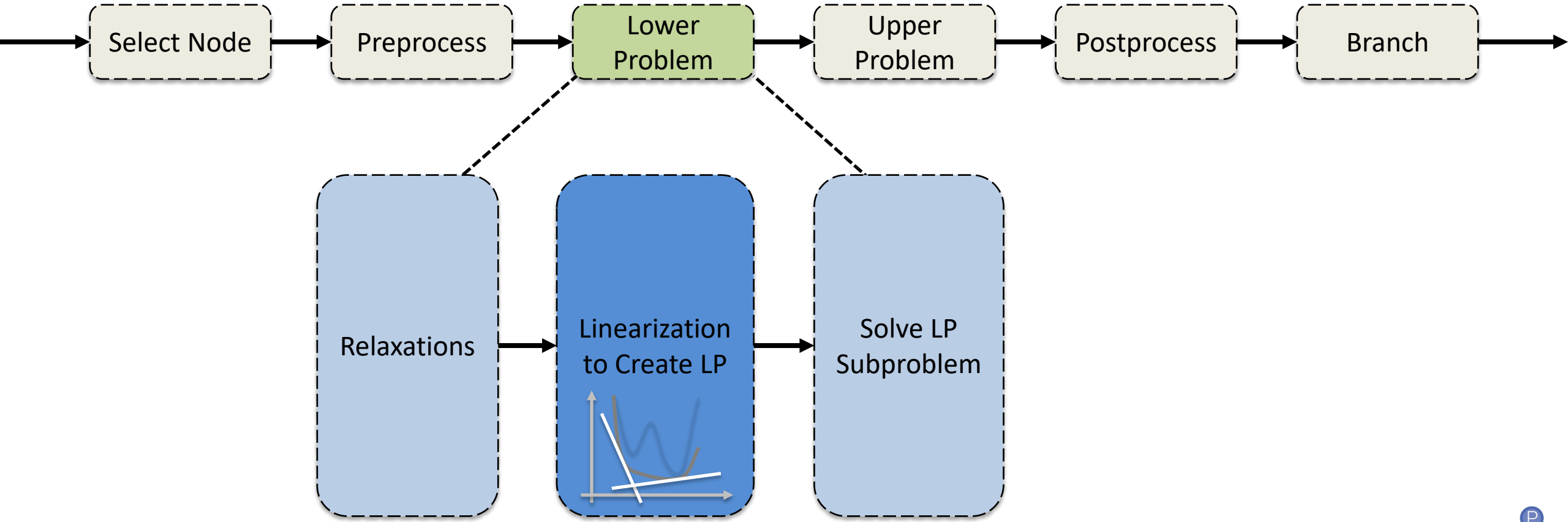
SourceCodeMcCormick.jl



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

11. Gottlieb, R.X. and Stuber, M.D. **Automatic Generation of GPU Kernels for Evaluators of McCormick-Based Relaxations and Subgradients.** Under Review, (2025).

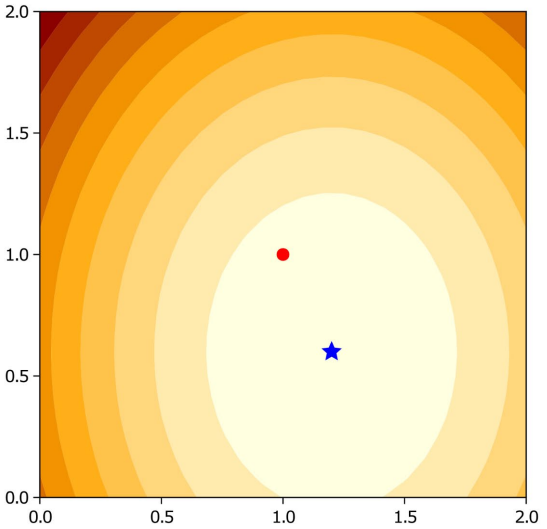
Parallelization Targets

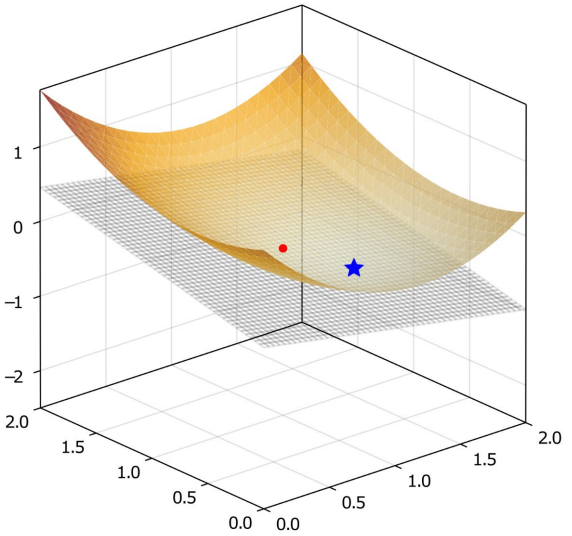


Kelley's Algorithm

- Approach utilized in EAGO

n	Lower Bound
1	-0.9830

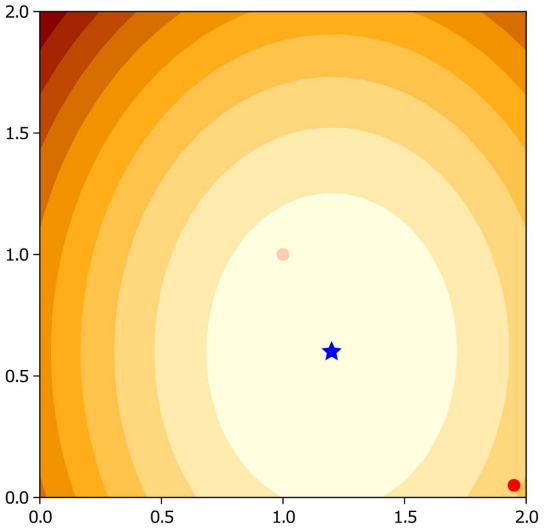


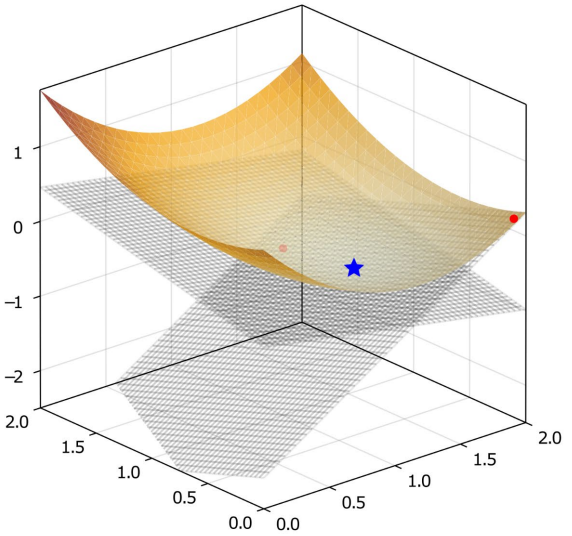


Kelley's Algorithm

- Approach utilized in EAGO

n	Lower Bound
1	-0.9830
2	-0.7100

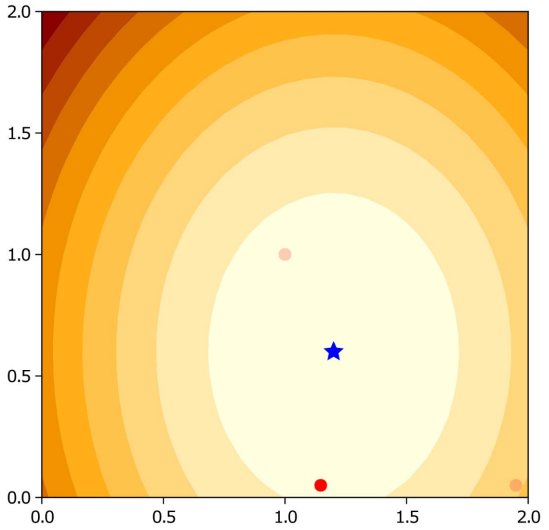


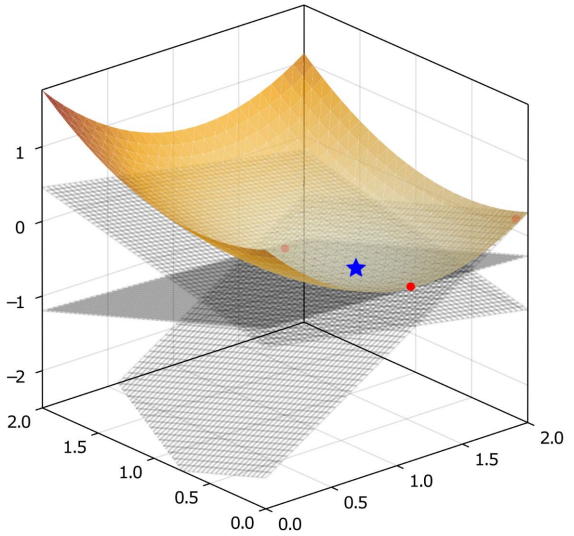


Kelley's Algorithm

- Approach utilized in EAGO

n	Lower Bound
1	-0.9830
2	-0.7100
3	-0.5815

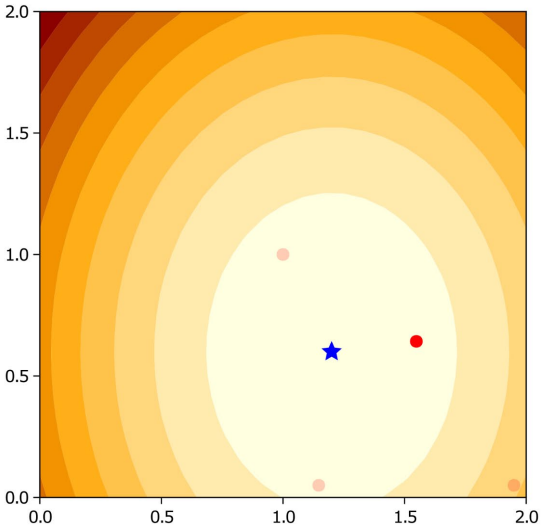


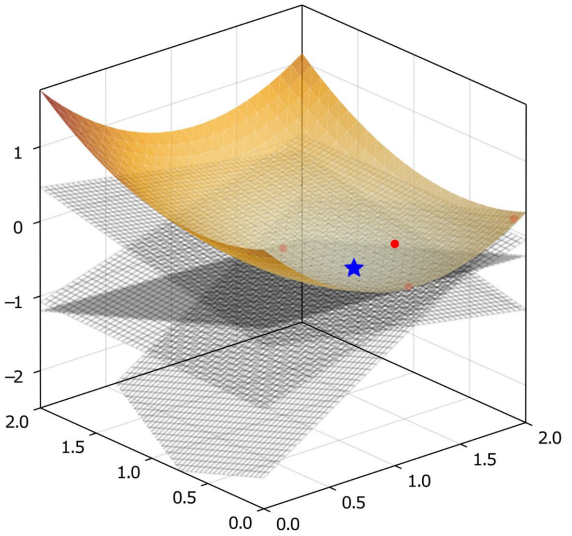


Kelley's Algorithm

- Approach utilized in EAGO

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

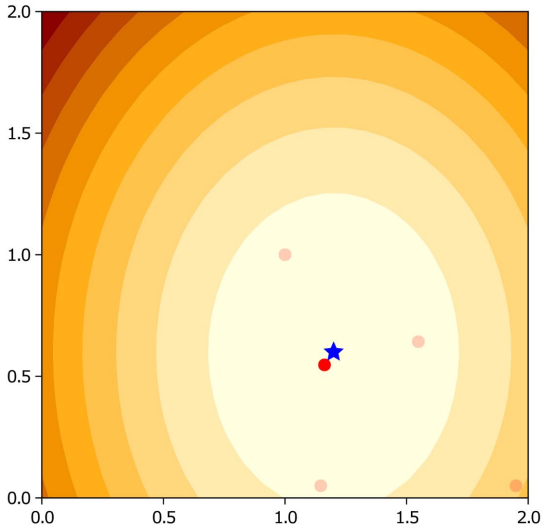


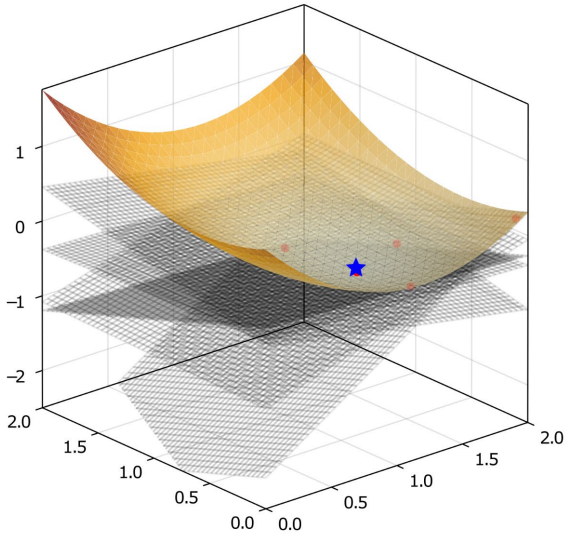


Kelley's Algorithm

- Approach utilized in EAGO

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

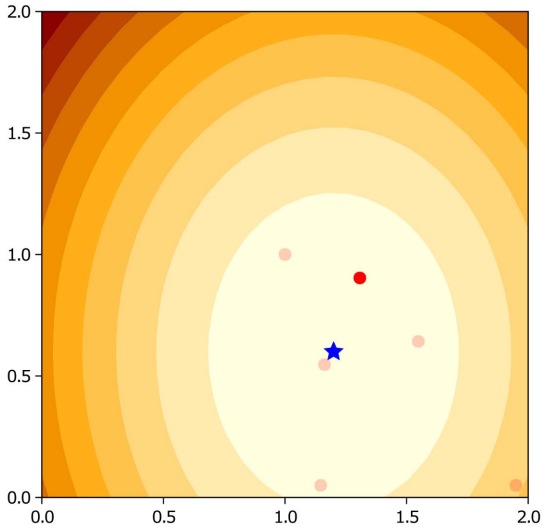


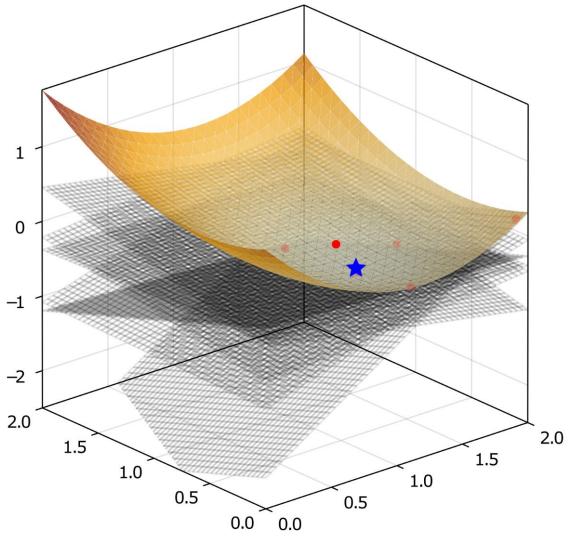


Kelley's Algorithm

- Approach utilized in EAGO

n	Lower Bound
1	-0.9830
2	-0.7100
3	-0.5815
4	-0.4962
5	-0.4001
6	-0.3891

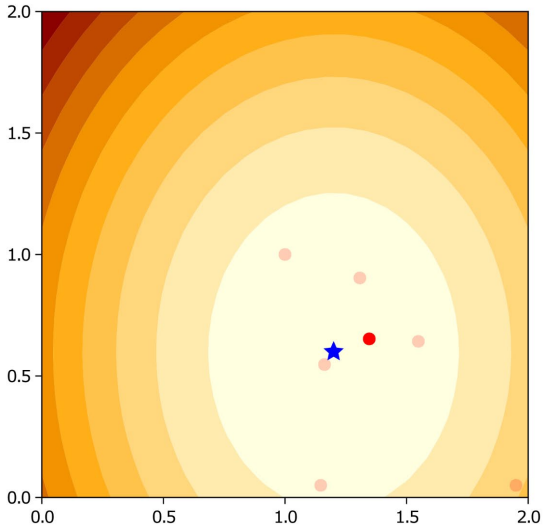


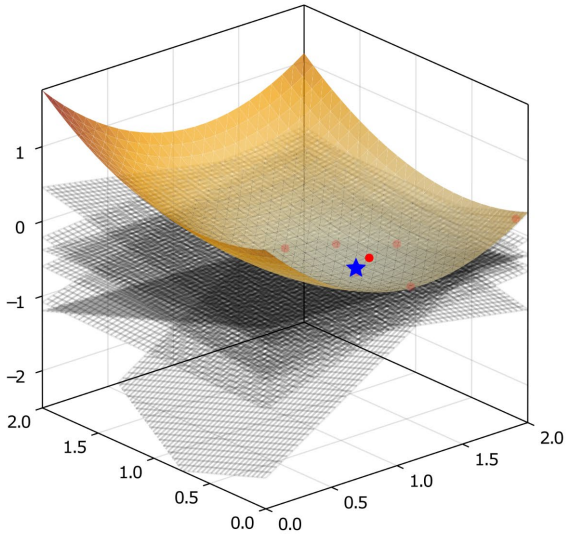


Kelley's Algorithm

- Approach utilized in EAGO

n	Lower Bound
1	-0.9830
2	-0.7100
3	-0.5815
4	-0.4962
5	-0.4001
6	-0.3891
7	-0.3854

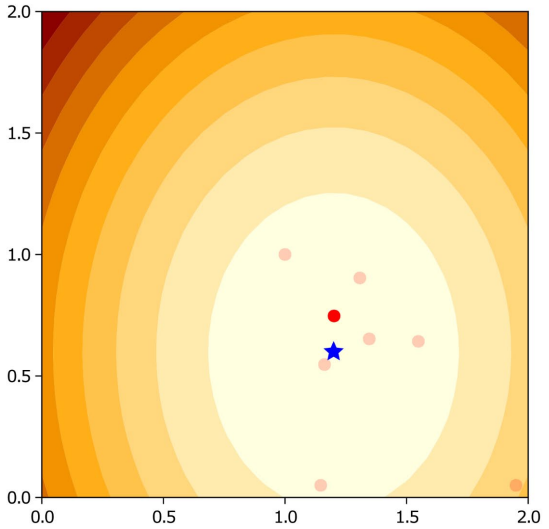


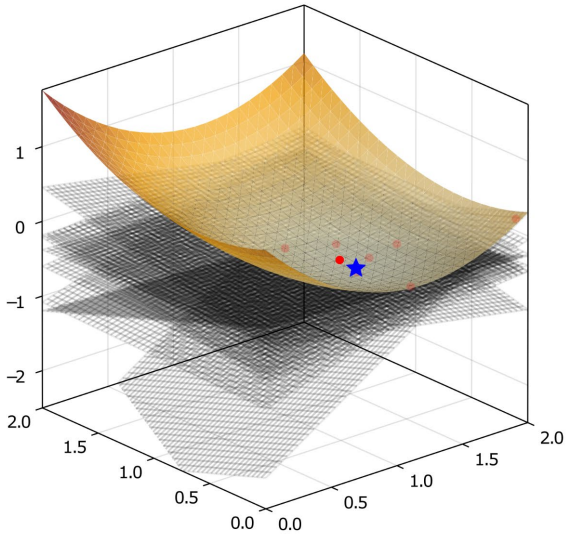


Kelley's Algorithm

- Approach utilized in EAGO

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

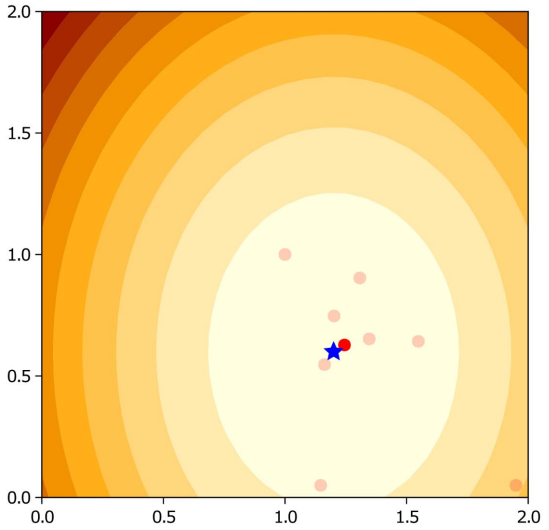


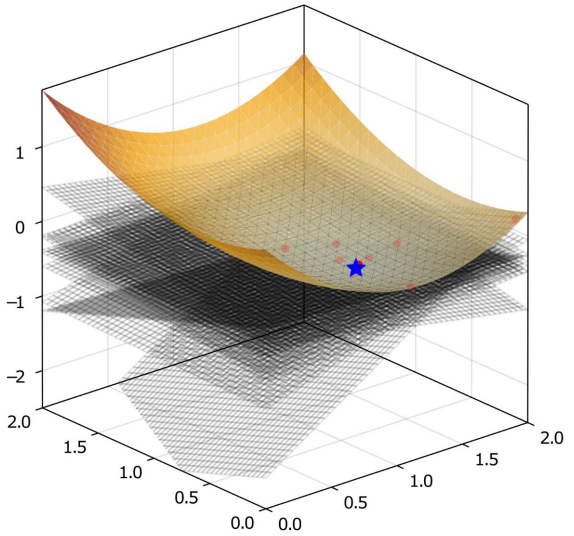


Kelley's Algorithm

- Approach utilized in EAGO

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

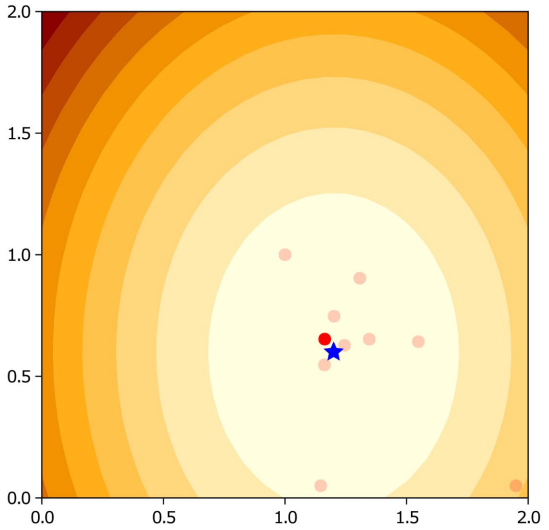


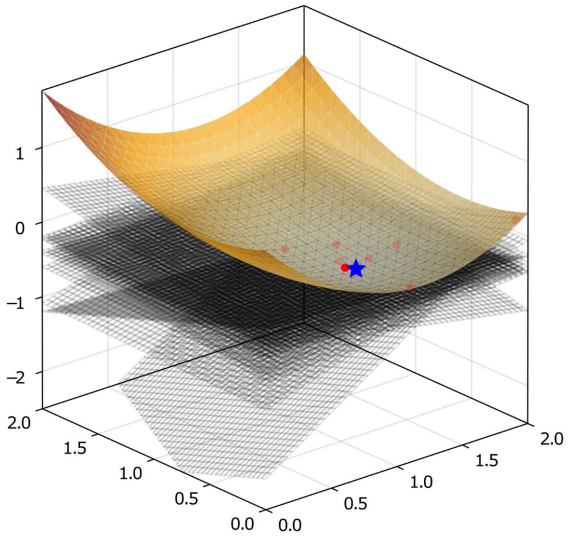


Kelley's Algorithm

- Approach utilized in EAGO

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

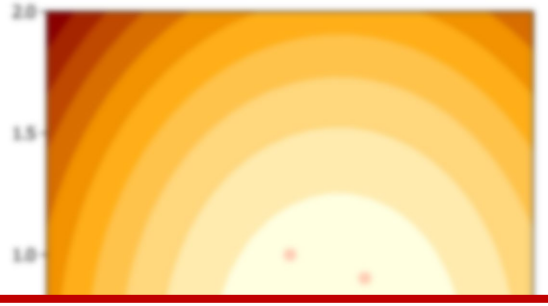


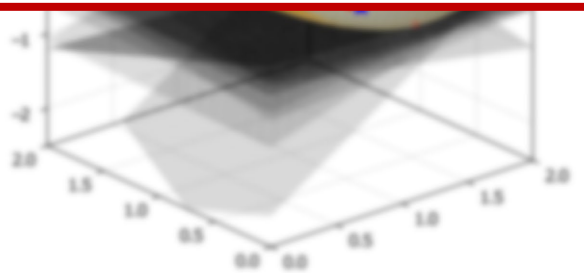


Kelley's Algorithm

- Approach utilized in EAGO

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





- Inherently sequential
- Different B&B nodes may terminate in different numbers of iterations
- Requires *n* LP solves per node

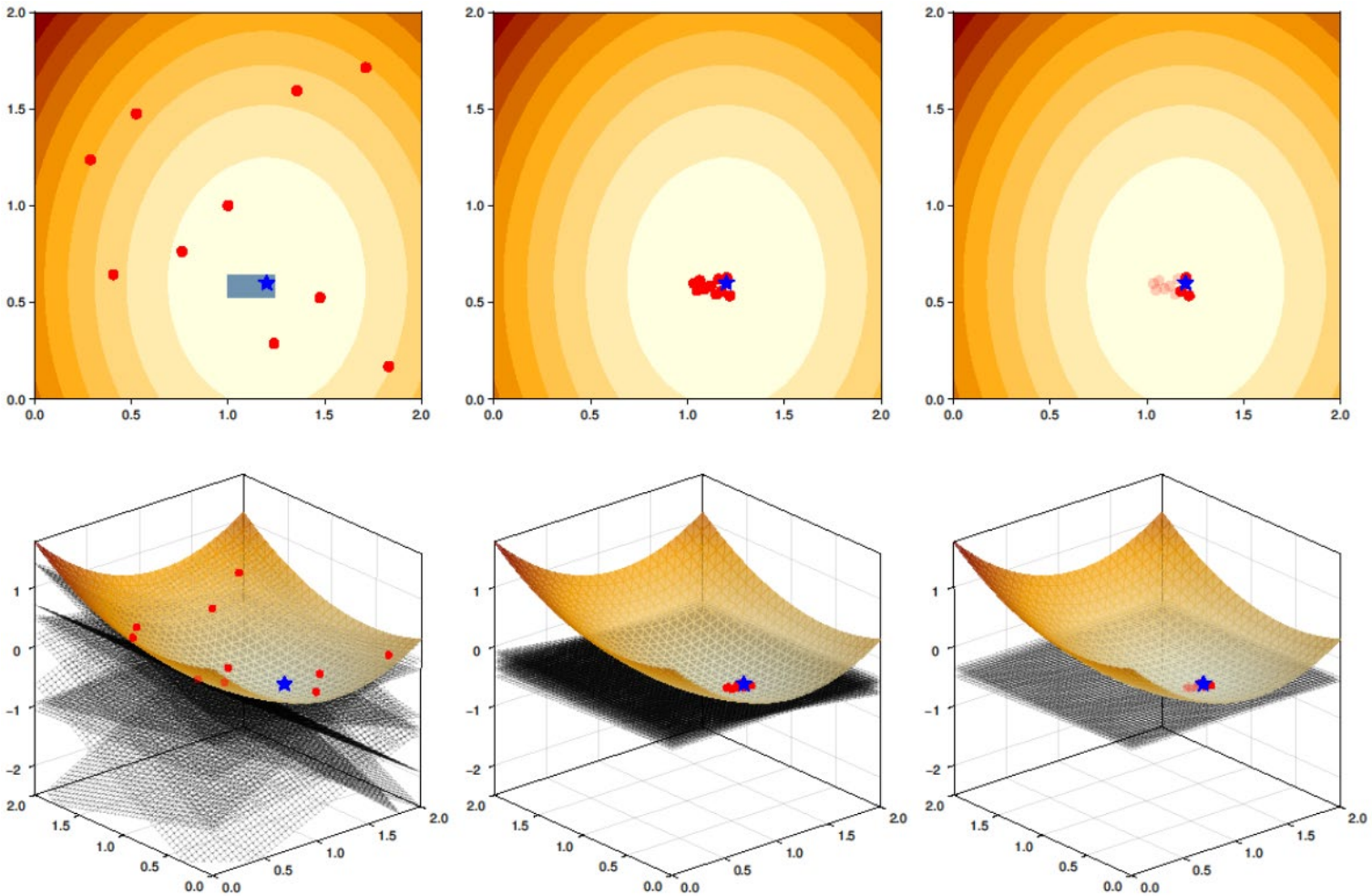
Linearization Point Selection Procedure

Algorithm 2 MultiSobol Scan Procedure

function `MultiSobolScan`$(\xi, \boldsymbol{\sigma}, n_{\text{Sobol}}, X)$
  Initialize $\mathbf{l}, \mathbf{u}$ such that $[\mathbf{l}, \mathbf{u}] = [X^{\text{L}}, X^{\text{U}}]$
  Scale $\boldsymbol{\sigma}$ to $[\mathbf{l} + \delta, \mathbf{u} - \delta]$, where δ is $2.5\% * (\mathbf{u} - \mathbf{l})$
  Evaluate McCormick relaxations and their subgradients for ξ at each point $\boldsymbol{\sigma}$ on the domain X, yielding $(\boldsymbol{\xi}^{\text{cv}}, \boldsymbol{\xi}^{\text{cc}}, \boldsymbol{\xi}^{\text{L}}, \boldsymbol{\xi}^{\text{U}}, \mathbf{s}^{\text{cv}}, \mathbf{s}^{\text{cc}})$
  for $i \in 1, \cdots, n$ **do**
    for $j \in 1, \cdots, n_{\text{Sobol}}$ **do**
      if $s^{\text{cv}}_{j,i} < 0$ and $\sigma_{j,i} > l_i$ **then**
        $l_i := \sigma_{j,i}$
      if $s^{\text{cv}}_{j,i} > 0$ and $\sigma_{j,i} < u_i$ **then**
        $u_i := \sigma_{j,i}$
    end for
    if $l_i > u_i$ **then**
      $l_i,\ u_i := u_i,\ l_i$
  end for
  return $\mathbf{l},\ \mathbf{u}$
end

Algorithm 1 MultiSobol Linearization Point Selection Process

Require: The objective function, ξ, from the original NLP
Require: The first n_{Sobol} points of a Sobol sequence $\sigma_j \in \mathbb{R}^n,\ \forall j = 1, \cdots, n_{\text{Sobol}}$
Require: The node subdomain $X \in \mathbb{IR}^n$
$\mathbf{l},\ \mathbf{u} =$ `MultiSobolScan`$(\xi, \boldsymbol{\sigma}, n_{\text{Sobol}}, X)$
Scale $\boldsymbol{\sigma}$ to $[\mathbf{l} + \delta, \mathbf{u} - \delta]$, where δ is $2.5\% * (\mathbf{u} - \mathbf{l})$
Evaluate McCormick relaxations and their subgradients for ξ at each point $\boldsymbol{\sigma}$ on the domain X, yielding $\{\boldsymbol{\xi}^{\text{cv}}, \boldsymbol{\xi}^{\text{cc}}, \boldsymbol{\xi}^{\text{L}}, \boldsymbol{\xi}^{\text{U}}, \mathbf{s}^{\text{cv}}, \mathbf{s}^{\text{cc}}\}$
Store relaxations and subgradients in a set $\Omega_j = \{\xi^{\text{cv}}_j, \xi^{\text{cc}}_j, \xi^{\text{L}}_j, \xi^{\text{U}}_j, \mathbf{s}^{\text{cv}}_j, \mathbf{s}^{\text{cc}}_j\}$
Set d as the dimensionality of ξ
for $i = 1, \cdots, d+1$ **do**
  if $\boldsymbol{\Omega}$ is nonempty **then**
    Select the Ω_j that minimizes $\sum_{i=1}^{n} (s^{\text{cv}}_{j,i})^2 \times (u_i - l_i)$
    Add an LP constraint that the epigraph variable lies above the subtangent hyperplane described by ξ^{cv}_j and $\mathbf{s}^{\text{cv}}_j$
    Remove from $\boldsymbol{\Omega}$ all Ω_k where the signs of values $\mathbf{s}^{\text{cv}}_k$ match the signs of values $\mathbf{s}^{\text{cv}}_j$
end for



Lower Bound: -0.4506

Lower Bound: -0.3758

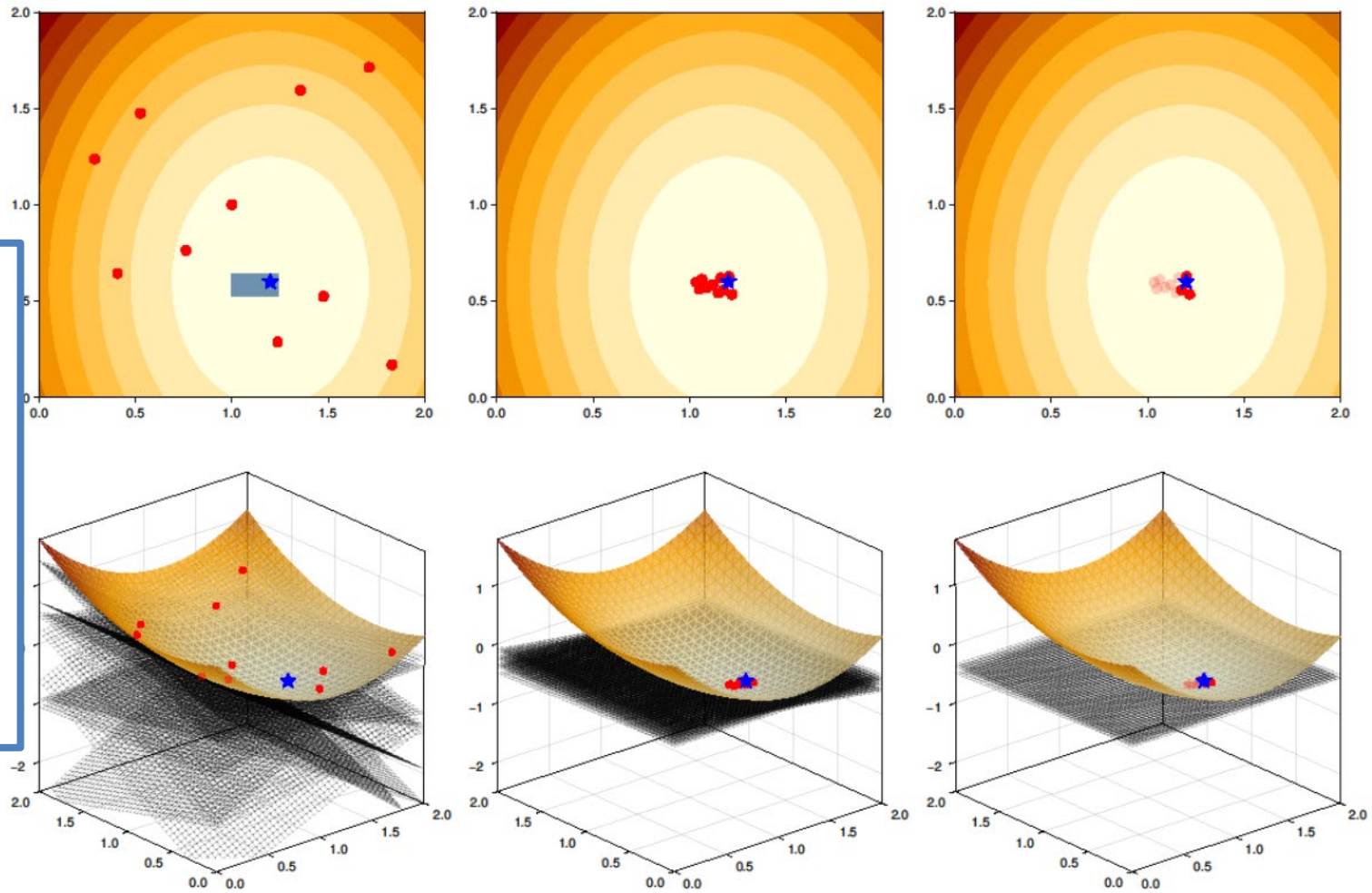
Lower Bound: -0.3758

Linearization Point Selection Procedure

Algorithm 2 MultiSobol Scan Procedure

function MultiSobolScan($\xi, \boldsymbol{\sigma}, n_{\text{Sobol}}, X$)
  Initialize $\mathbf{l}, \mathbf{u}$ such that $[\mathbf{l}, \mathbf{u}] = [X^{\text{L}}, X^{\text{U}}]$
  Scale $\boldsymbol{\sigma}$ to $[\mathbf{l} + \delta, \mathbf{u} - \delta]$, where δ is $2.5\% * (\mathbf{u} - \mathbf{l})$
  Evaluate McCormick relaxations and their subgradients for ξ at each point $\boldsymbol{\sigma}$ on the domain X, yielding $(\xi^{\text{cv}}, \xi^{\text{cc}}, \xi^{\text{L}}, \xi^{\text{U}}, s^{\text{cv}}, s^{\text{cc}})$
  for $i \in 1, \cdots, n$ **do**
    for $j \in 1, \cdots, n_{\text{Sobol}}$ **do**
      if $s^{\text{cv}}_{j,i} < 0$ and $\sigma_{j,i} > l_i$ **then**
        $l_i := \sigma_{j,i}$
      if $s^{\text{cv}}_{j,i} > 0$ and $\sigma_{j,i} < u_i$ **then**
for $i = 1, \cdots, d+1$ **do**
  if $\boldsymbol{\Omega}$ is nonempty **then**
    Select the Ω_j that minimizes $\sum_{i=1}^{n} (s^{\text{cv}}_{j,i})^2 \times (u_i - l_i)$
    Add an LP constraint that the epigraph variable lies above the subtangent hyperplane described by ξ^{cv}_j and $\mathbf{s}^{\text{cv}}_j$
    Remove from $\boldsymbol{\Omega}$ all Ω_k where the signs of values $\mathbf{s}^{\text{cv}}_k$ match the signs of values $\mathbf{s}^{\text{cv}}_j$
end for

- \+ Points evaluated in parallel
- \+ Every node requires same points, known a priori
- \+ Requires 1 LP solve per node

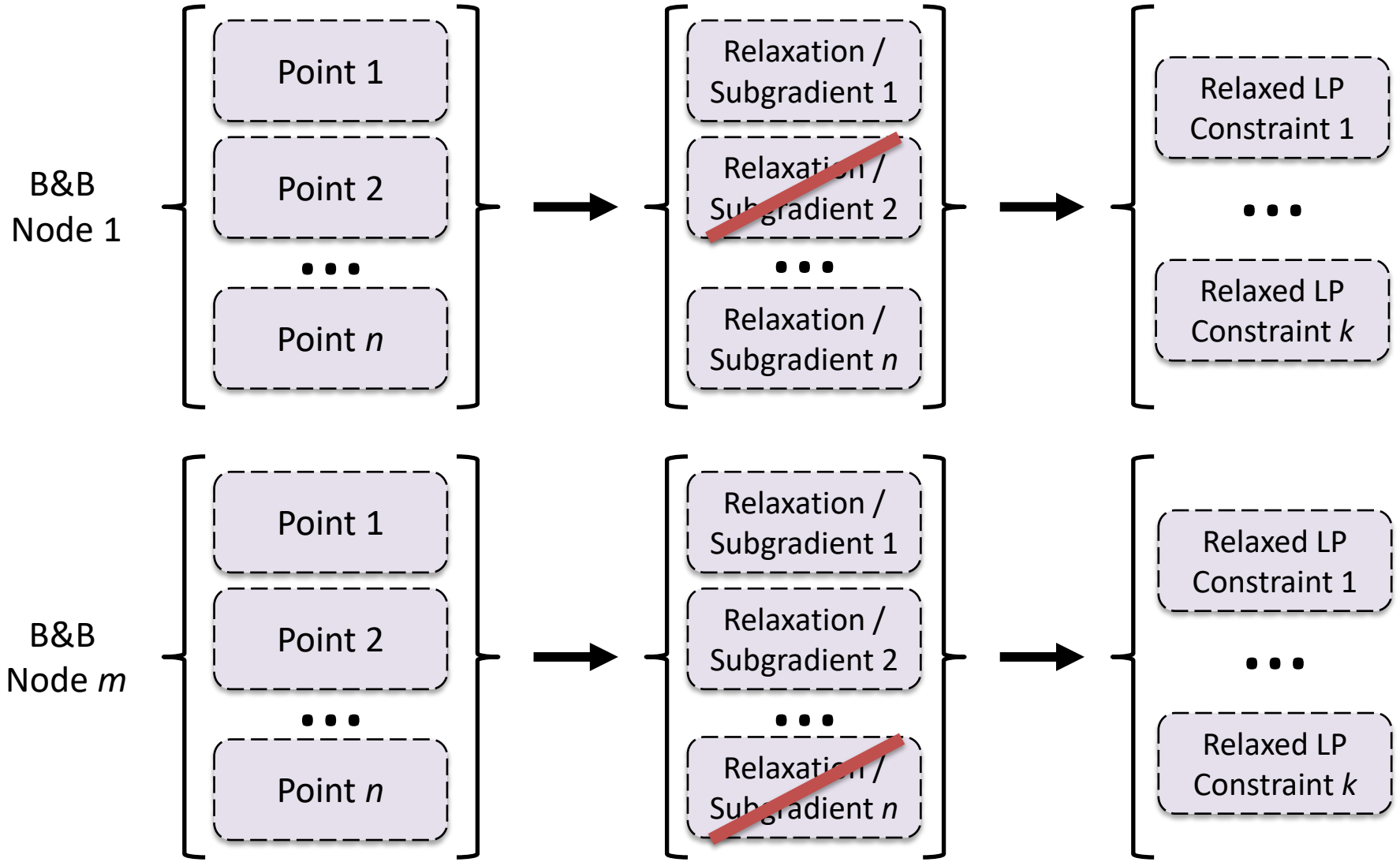


Lower Bound: -0.4506

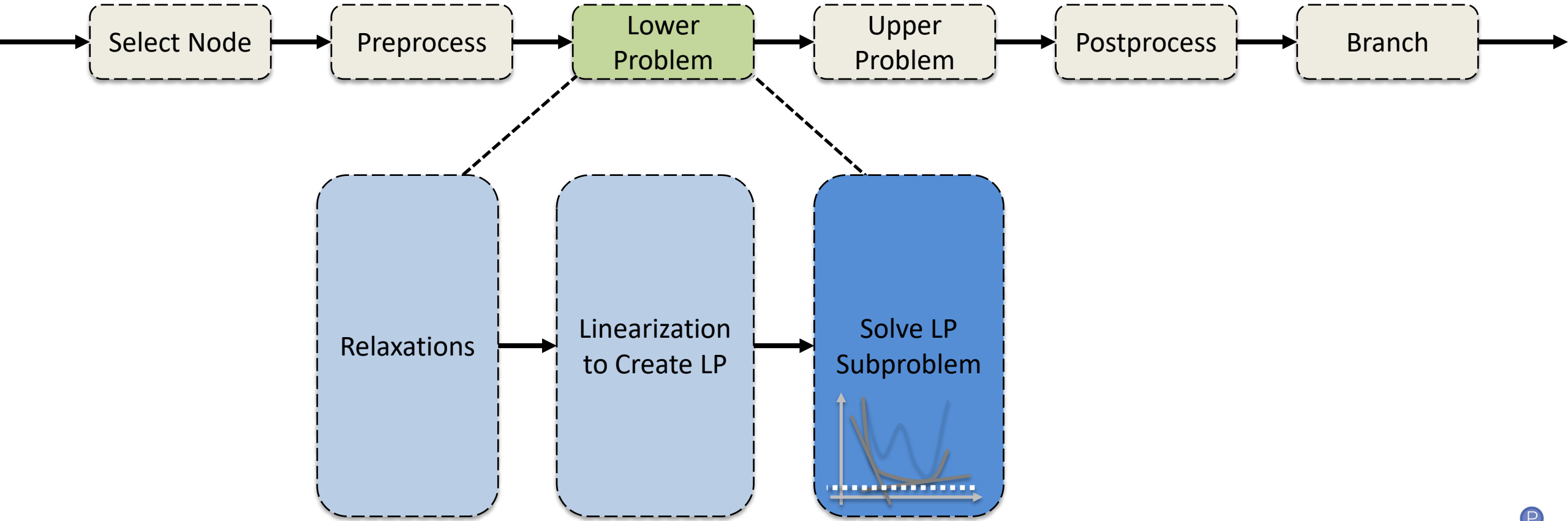
Lower Bound: -0.3758

Lower Bound: -0.3758

Linearization Point Selection Procedure



Parallelization Targets



PDLP

Primal Dual Hybrid Gradient for LP (PDLP)[13]

- Applegate et al. (2021) developed a competitive first-order method (FOM) for solving LPs

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

$$\downarrow$$

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

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.

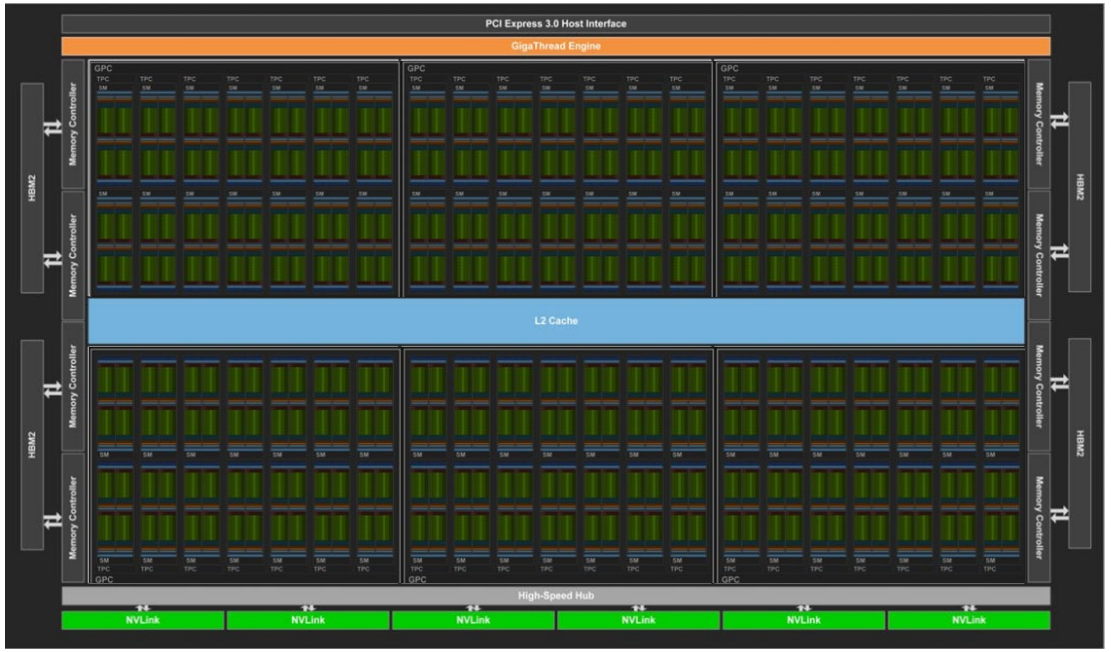
13. Applegate, D., et al. **Practical Large-Scale Linear Programming using Primal-Dual Hybrid Gradient.** *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*, (2021).

BatchPDLP.jl

- Custom novel implementation of PDLP, designed for **small-scale LPs** **(unique!)**
- **Each LP** 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



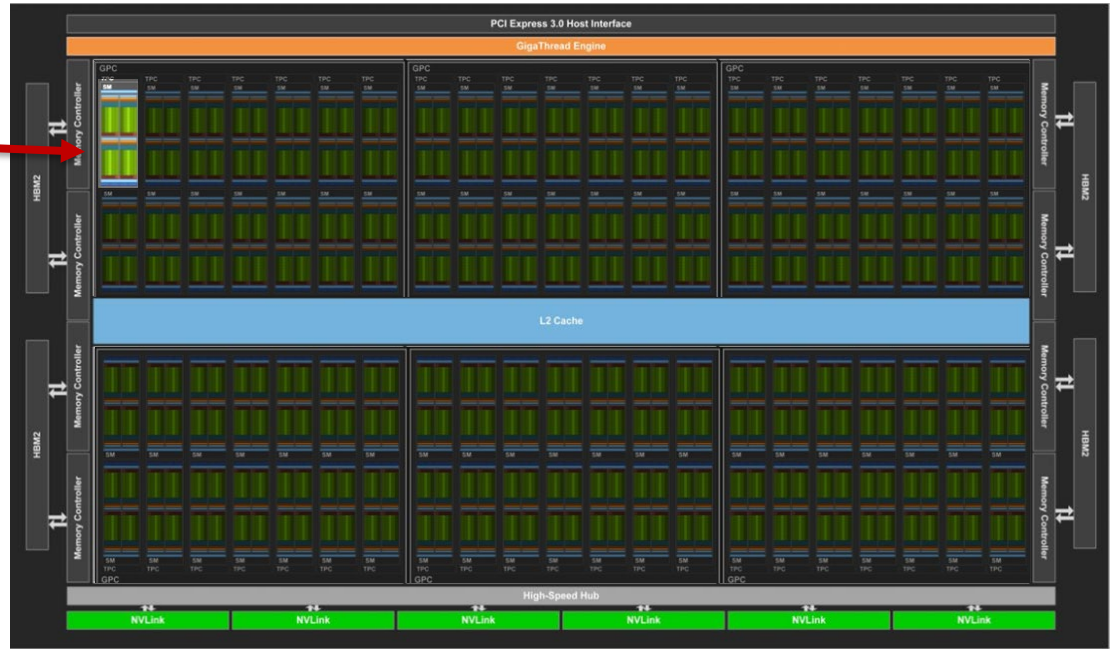
14. Gottlieb, R.X., Alston, D., and Stuber, M.D. **Re-Architected PDLP for Batch Parallelization of Linear Programs.** *Under Review.*

BatchPDLP.jl

- Custom novel implementation of PDLP, designed for **small-scale LPs** **(unique!)**
- **Each LP** 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



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

BatchPDLP.jl

- Custom novel implementation of PDLP, designed for **small-scale LPs** **(unique!)**
- **Each LP** 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

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

BatchPDLP.jl

- Custom novel implementation of PDLP, designed for **small-scale LPs** **(unique!)**
- **Each LP** solved by a **portion** of the GPU

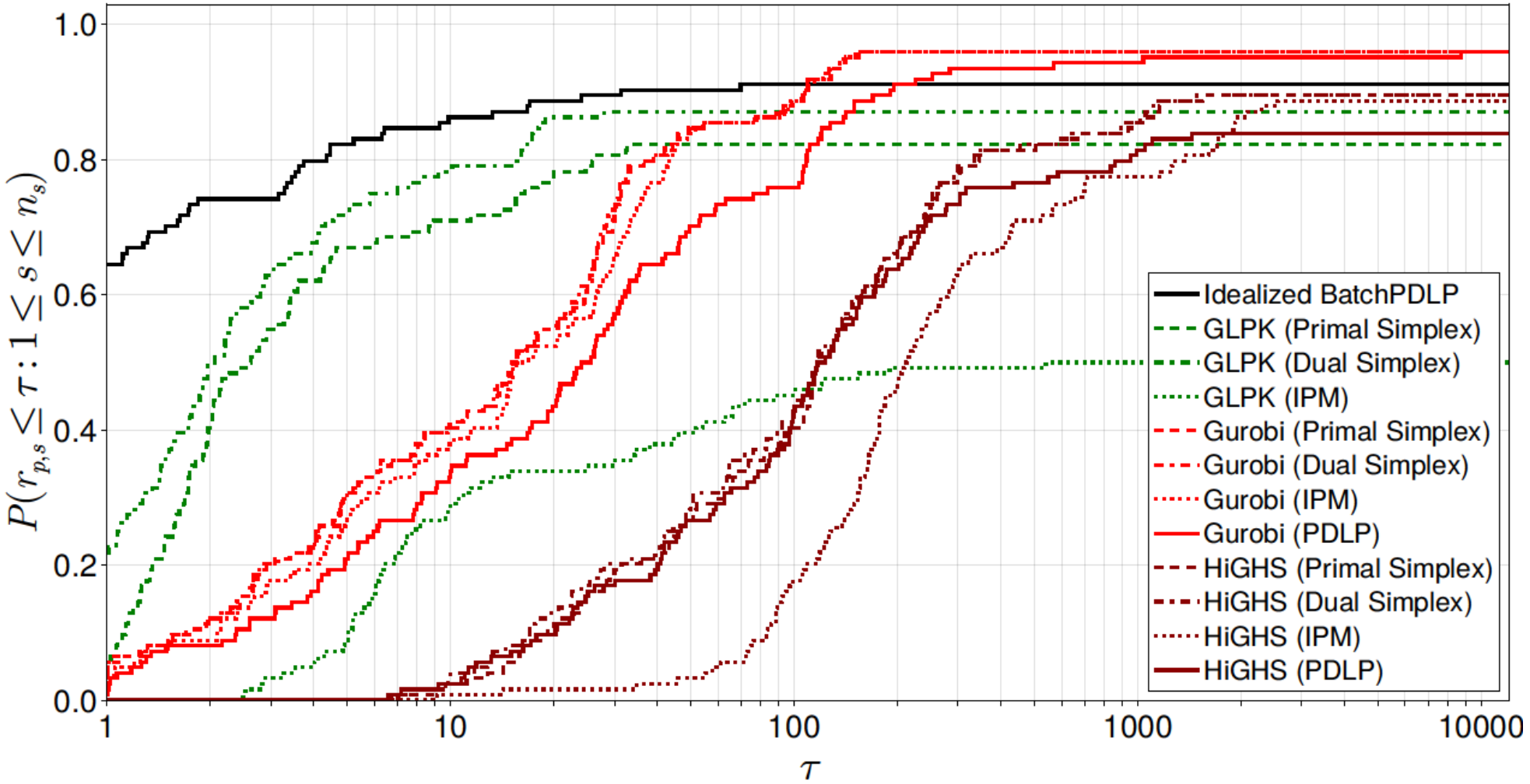
Algorithm 1: Single-Kernel PDLP

1 **foreach** *LP* **do**
2 **Input:** An initial solution $z_i^{0,0}$;
3 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 $\omega_i^0 \leftarrow \texttt{InitializePrimalWeight}(c_i, q_i)$;
4 **repeat**
5 $t_i \leftarrow 0$;
6 **repeat**
7 $z_i^{n_i,t_i+1}, \eta_i^{n_i,t_i+1}, \hat{\eta}_i^{n_i,t_i+1} \leftarrow \texttt{AdaptivePDHGStep}(z_i^{n_i,t_i}, \omega_i^{n_i}, \hat{\eta}_i^{n_i,t_i}, k_i)$;
8 $\bar{z}_i^{n_i,t_i+1} \leftarrow \frac{1}{\sum_{j=1}^{t_i+1} \eta_i^{n_i,j}} \sum_{j=1}^{t_i+1} \eta_i^{n_i,j} z_i^{n_i,j}$;
9 $z_{c,i}^{n_i,t_i+1} \leftarrow \texttt{GetRestartCandidate}(z_i^{n_i,t_i+1}, \bar{z}_i^{n_i,t_i+1}, \omega_i^{n_i})$;
10 $t_i \leftarrow t_i + 1, k_i \leftarrow k_i + 1$;
11 **until** *restart or termination criteria holds*;
12 **Restart the outer loop.** $z_i^{n_i+1,0} \leftarrow z_{c,i}^{n_i,t_i}, n_i \leftarrow n_i + 1$;
13 $\omega_i^{n_i} \leftarrow \texttt{PrimalWeightUpdate}(z_i^{n_i,0}, z_i^{n_i-1,0}, \omega_i^{n_i-1})$;
14 **until** *termination criteria holds*;
15 **Output:** $z_i^{n_i,0}$.
16 **end**

Other LPs continue even if one finishes

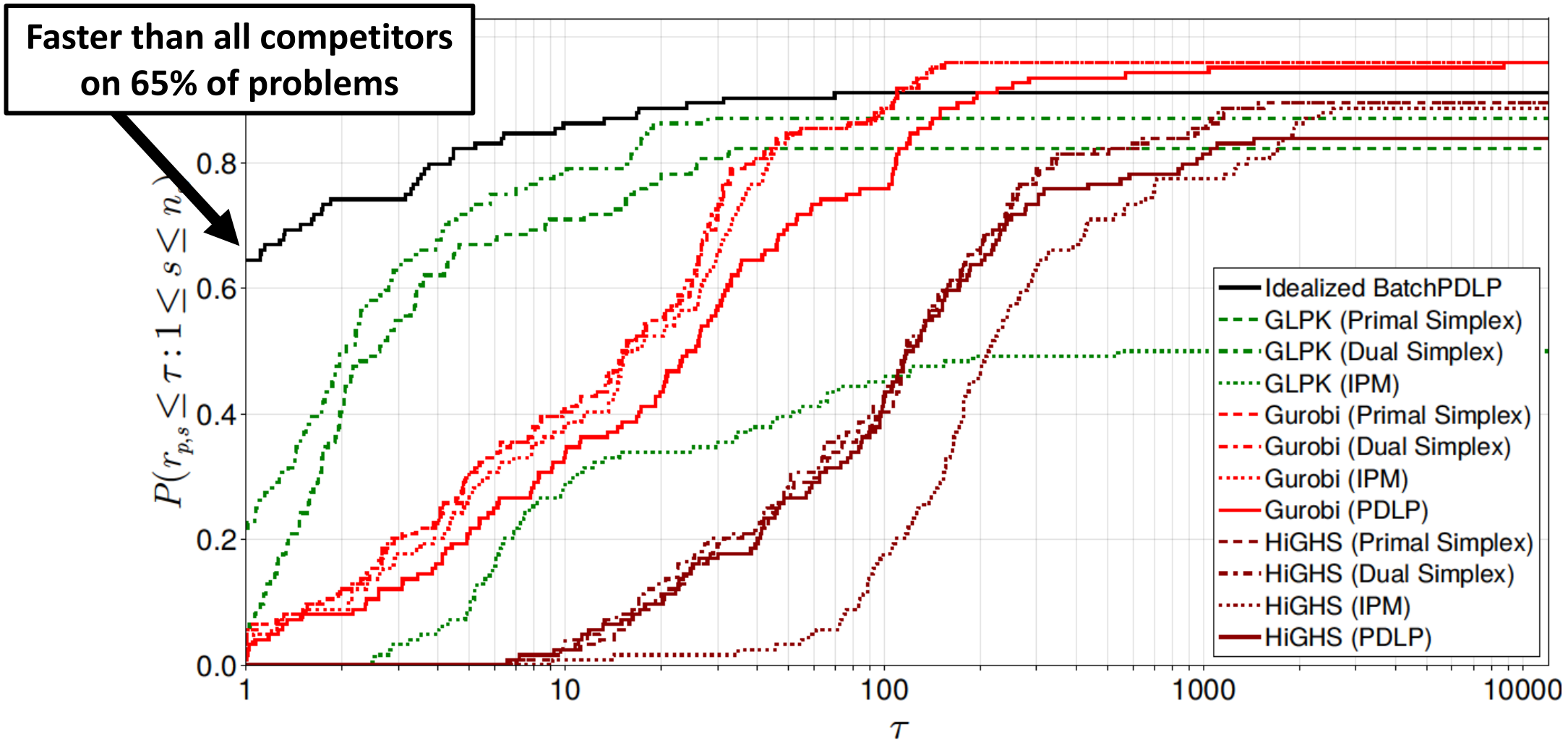
14. Gottlieb, R.X., Alston, D., and Stuber, M.D. **Re-Architected PDLP for Batch Parallelization of Linear Programs.** *Under Review.*

BatchPDLP.jl Benchmarking



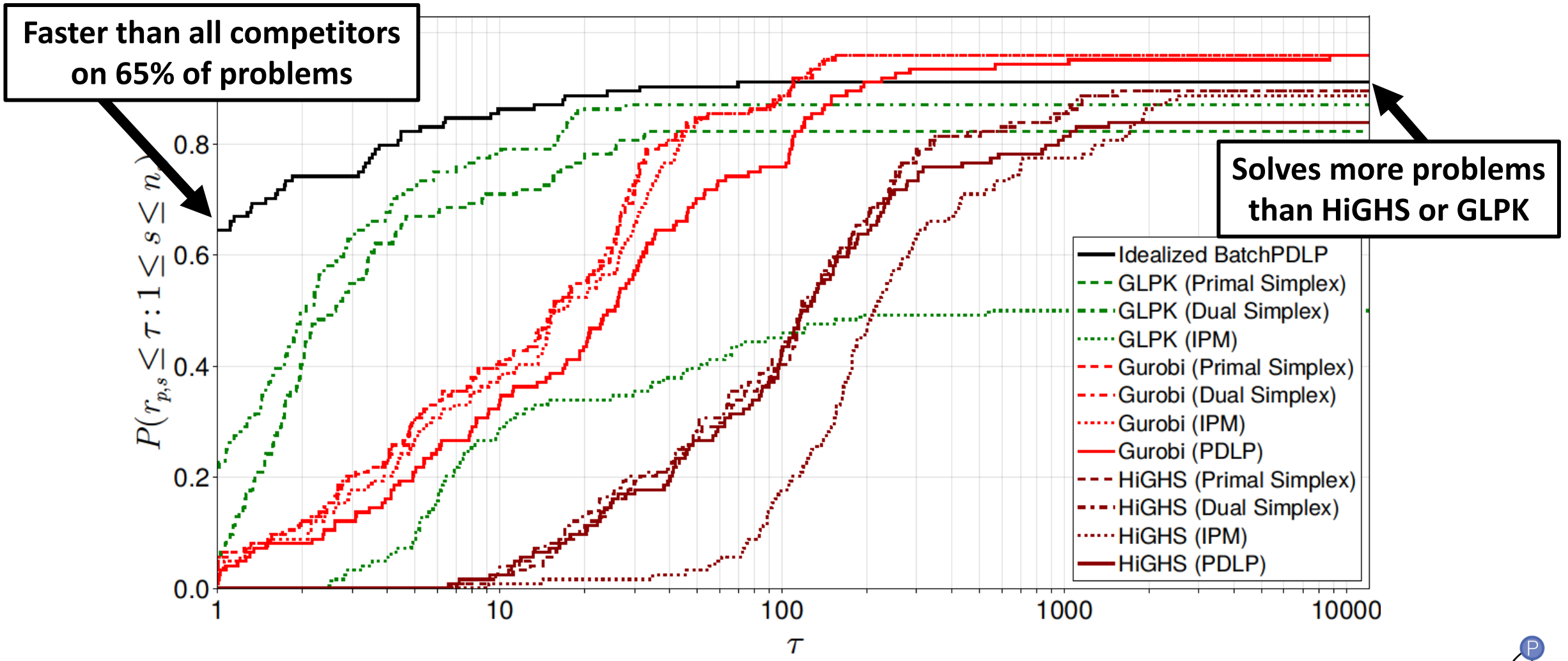
14. Gottlieb, R.X., Alston, D., and Stuber, M.D. **Re-Architected PDLP for Batch Parallelization of Linear Programs.** *Under Review.*

BatchPDLP.jl Benchmarking



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

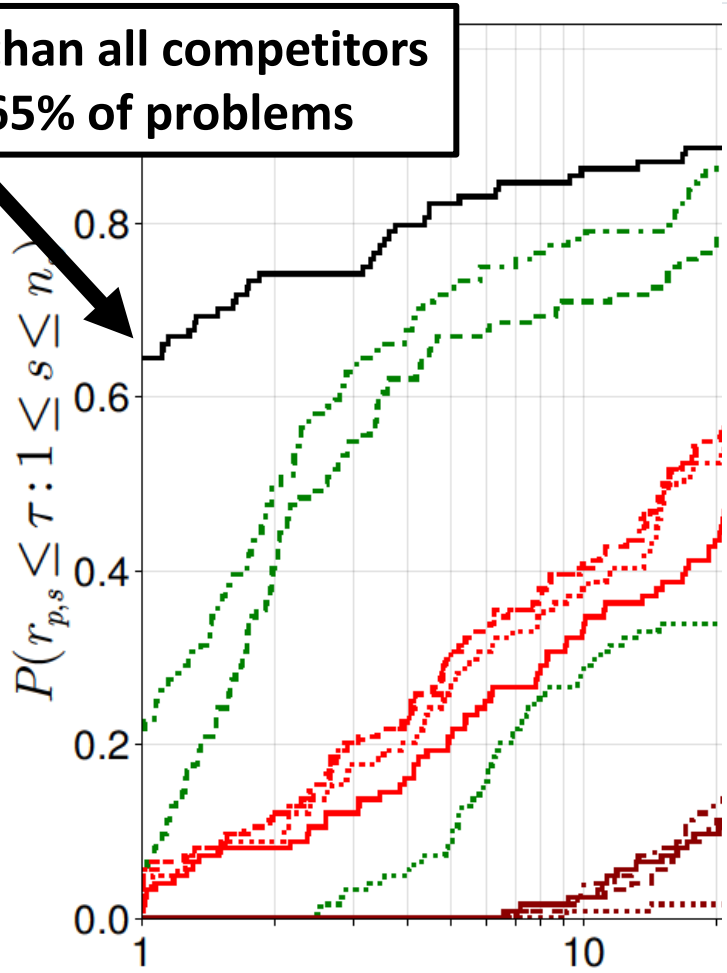
BatchPDLP.jl Benchmarking



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

BatchPDLP.jl Benchmarking

Faster than all competitors on 65% of problems



lems GLPK

PSORLab / BatchPDLP.jl

Type / to search

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

BatchPDLP.jl Public

Edit Pins Watch 0 Fork 0 Star 6

main 1 Branch 1 Tag Go to file t Code

RXGottlieb Remove CUDA use from top-level scope b796515 · 3 weeks ago 12 Commits

.github/workflows	Update Benchmarks	3 weeks ago
benchmarks	Update Benchmarks	3 weeks ago
src	Remove CUDA use from top-level scope	3 weeks ago
.gitignore	Initial commit	2 months ago
LICENSE.md	v0.1.0	2 months ago
Project.toml	v0.1.0	2 months ago
README.md	Update Benchmarks	3 weeks ago

README License

BatchPDLP.jl

This package applies the PDLP algorithm [1] to solve batches of small, structurally similar linear programs (LPs) simultaneously using NVIDIA GPU resources through CUDA.jl [2]. It assumes that LPs are provided (or can be generated) in GPU memory, and only stores solution information in GPU memory. I.e., this package does not manage the transfer of LP information to or from the CPU. `BatchPDLP.jl` is meant to be used within a global optimizer such as `EAGO.jl` [3] to solve large numbers of LPs of sizes typically seen within global optimization. `BatchPDLP.jl` was tested on batches of LPs with up to roughly 100 variables and 1000 constraints each.

Usage Note

`BatchPDLP.jl` is primarily designed to work with `SourceCodeMcCormick.jl` [4], which calculates McCormick relaxations and their subgradients for factorable expressions. These relaxations and their subgradients can then be converted into LP constraints through `BatchPDLP.jl`, after which the PDLP algorithm can be run. Internally, `BatchPDLP.jl` works by launching a kernel with the number of blocks set equal to the number of LPs being solved, meaning the number of LPs per batch should generally be larger than the number of streaming multiprocessors (SMs) in the GPU.

More specifically, `BatchPDLP.jl` is not meant to handle single LP instances. It is explicitly not designed to handle

About

A GPU-accelerated batched LP solver based on PDLP

Readme
View license
Activity
Custom properties
6 stars
0 watching
0 forks
Report repository

Releases 1

v0.1.0 Latest
3 weeks ago

Packages

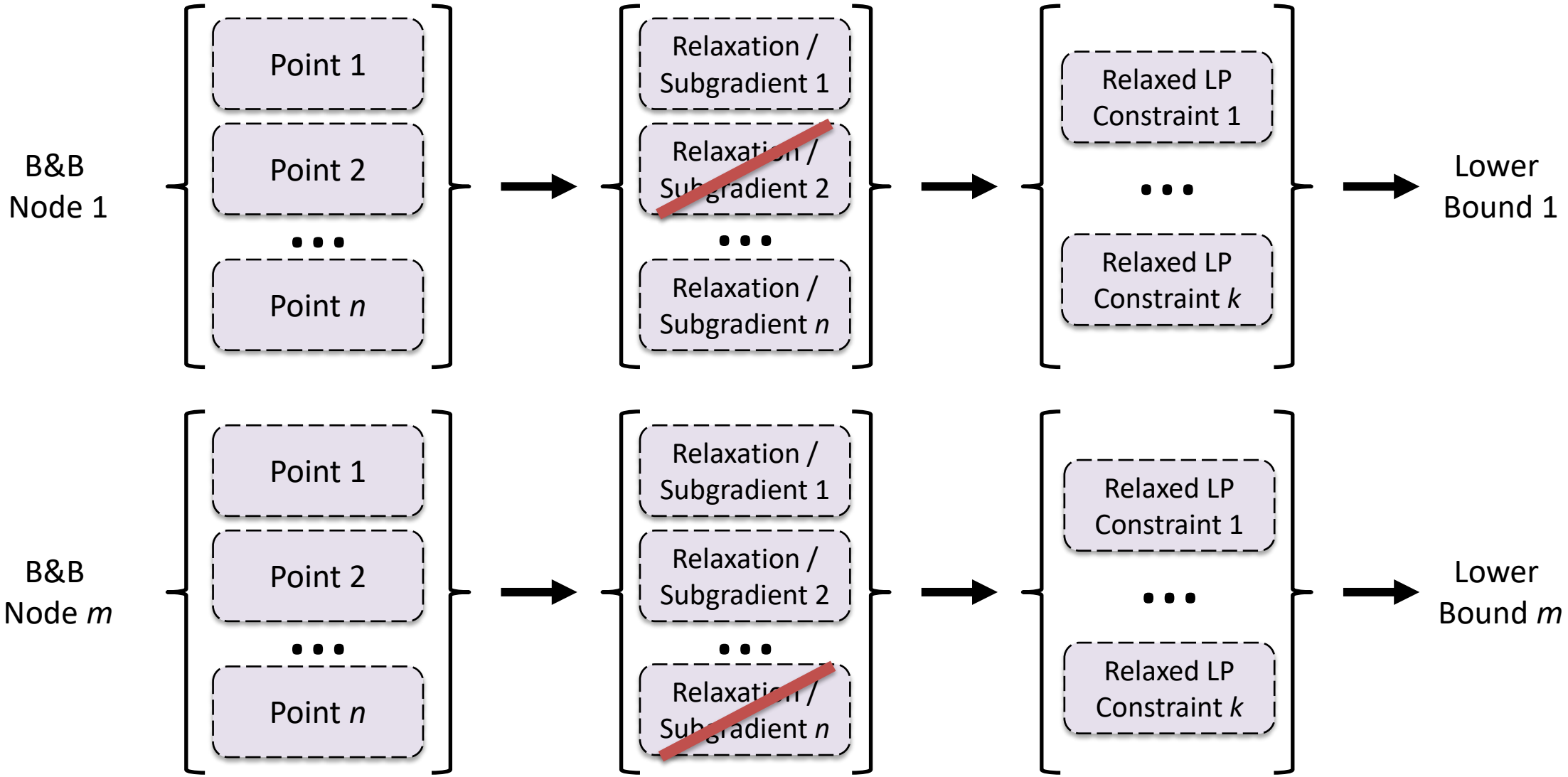
No packages published
Publish your first package

Languages

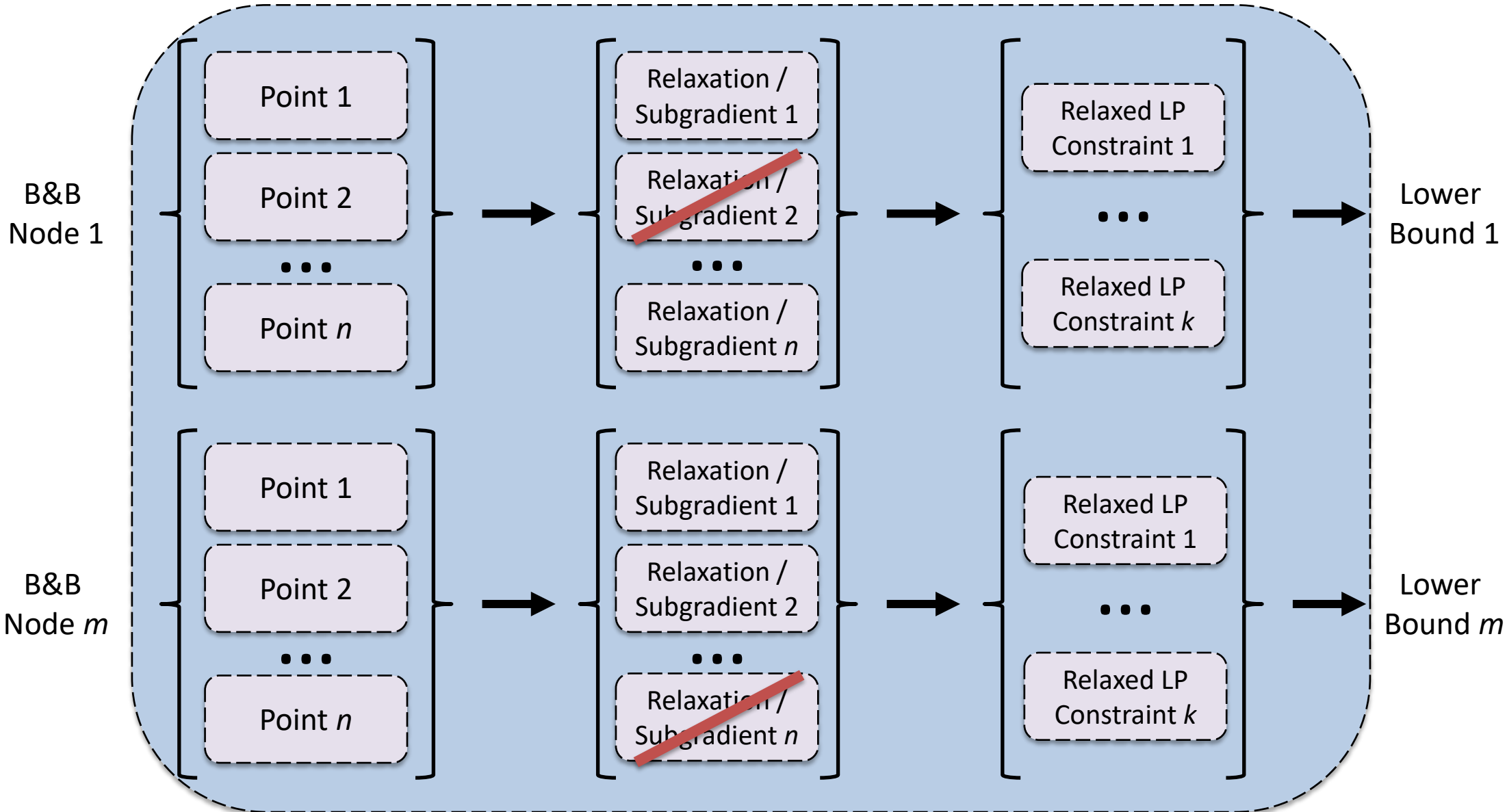
Julia 100.0%

14. Gottlieb, R.X., Alston, D., and Stuber, M.D. **Re-Architected PDLP for Batch**

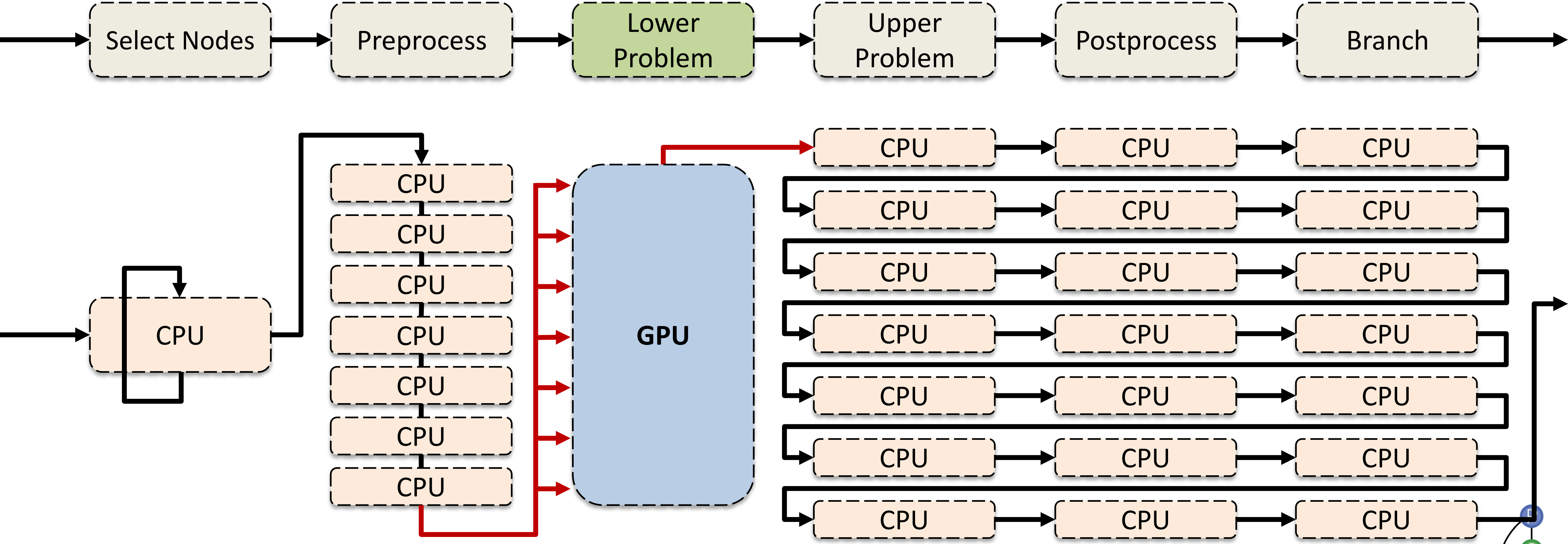
Integrated GPU Components



Integrated GPU Components



Integrated CPU-GPU B&B

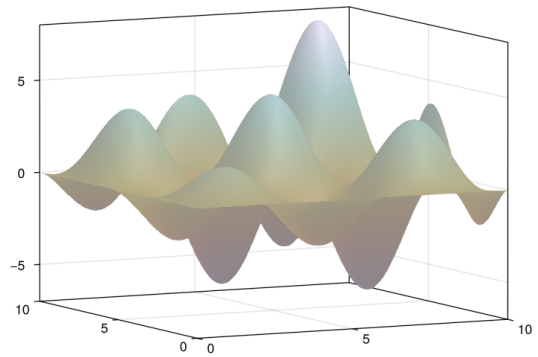


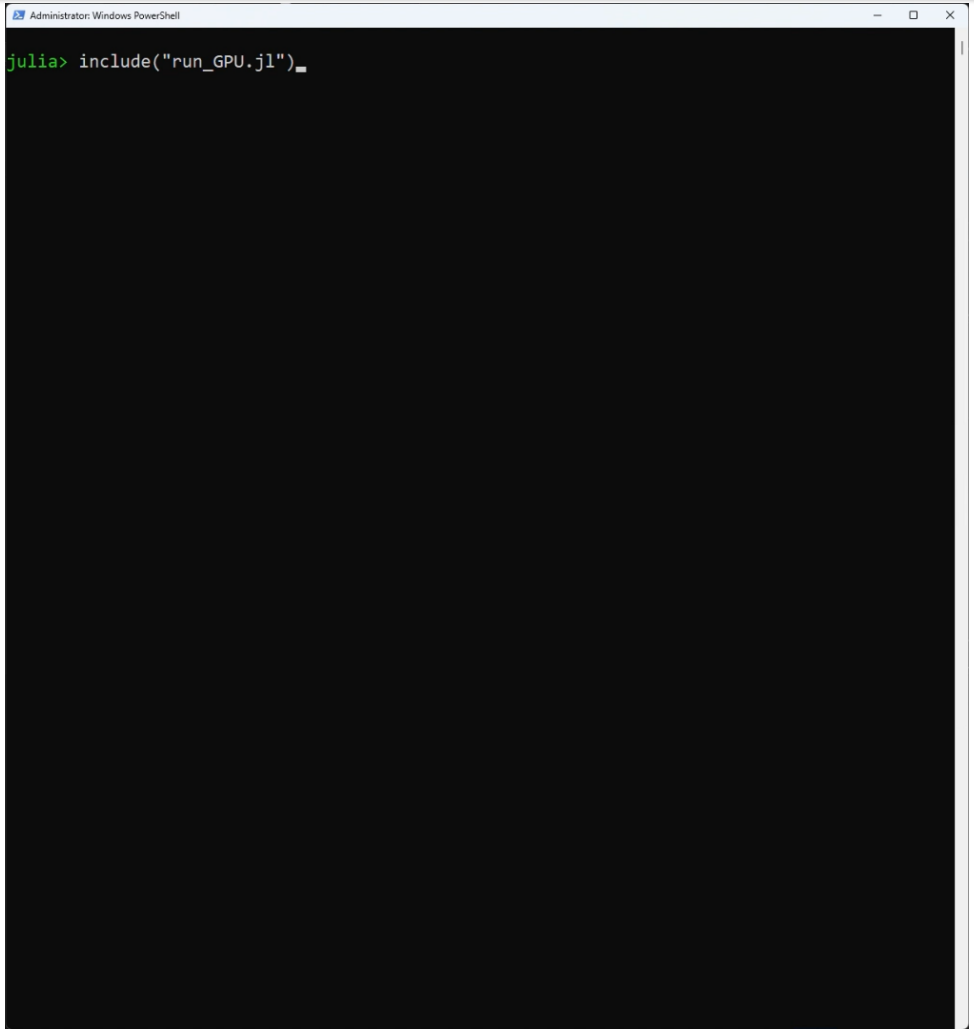
Scalable Benchmark Problem

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

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

- Difficulty scales with dimension



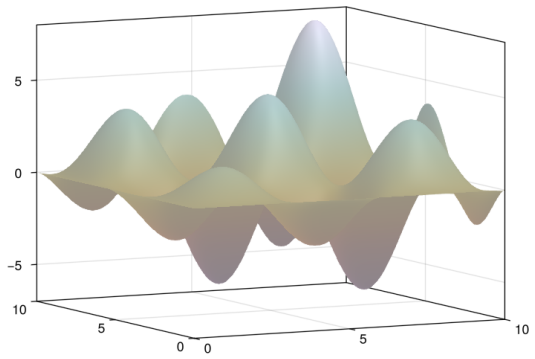


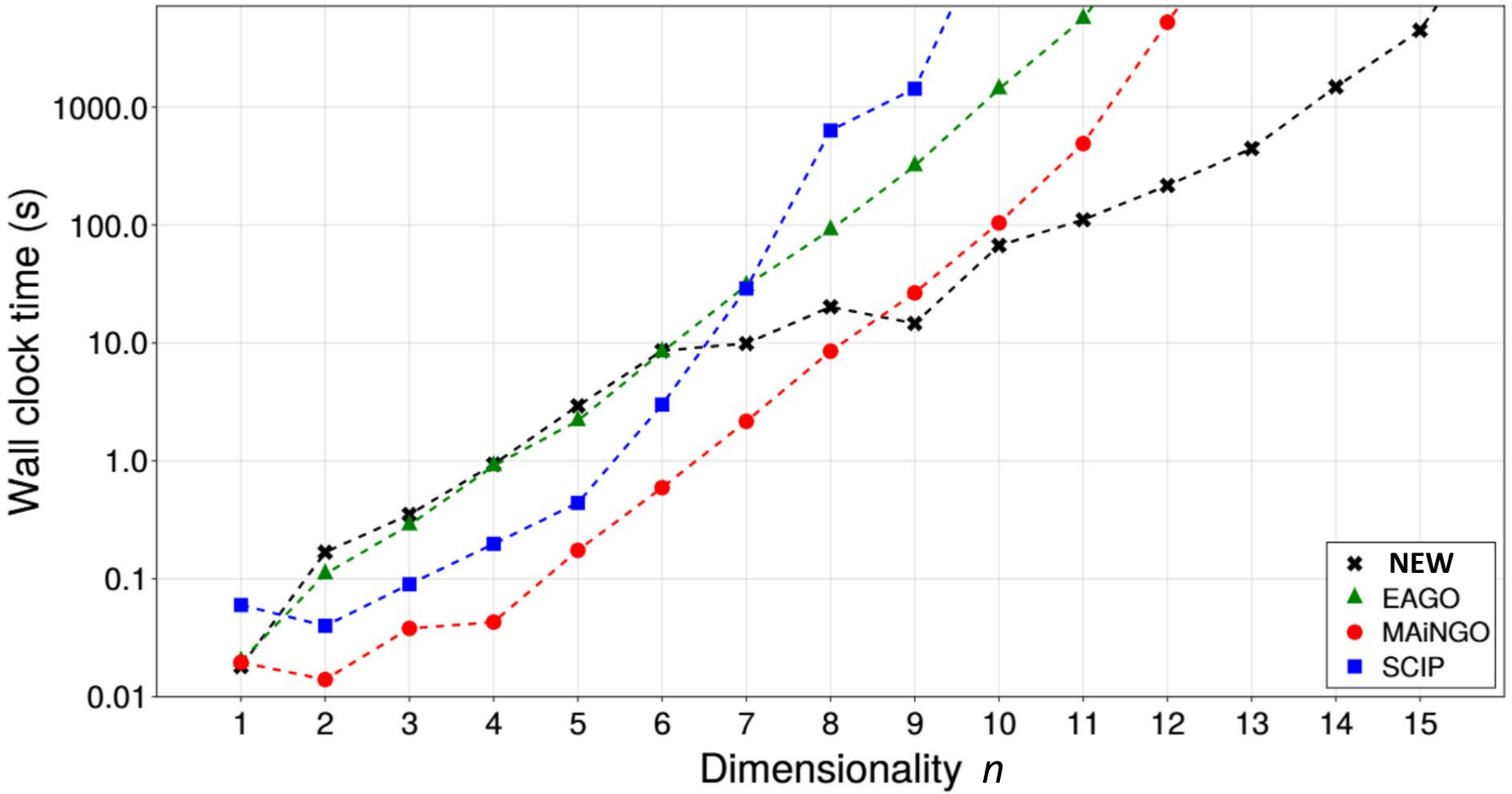
Scalable Benchmark Problem

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

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

- Difficulty scales with dimension



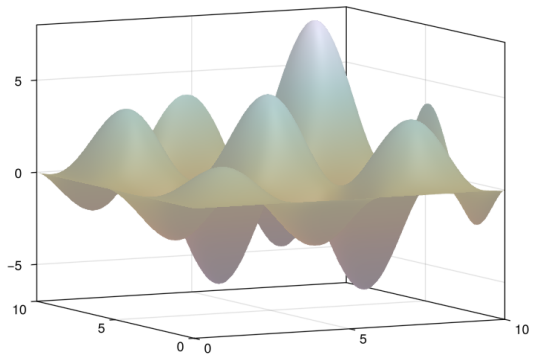


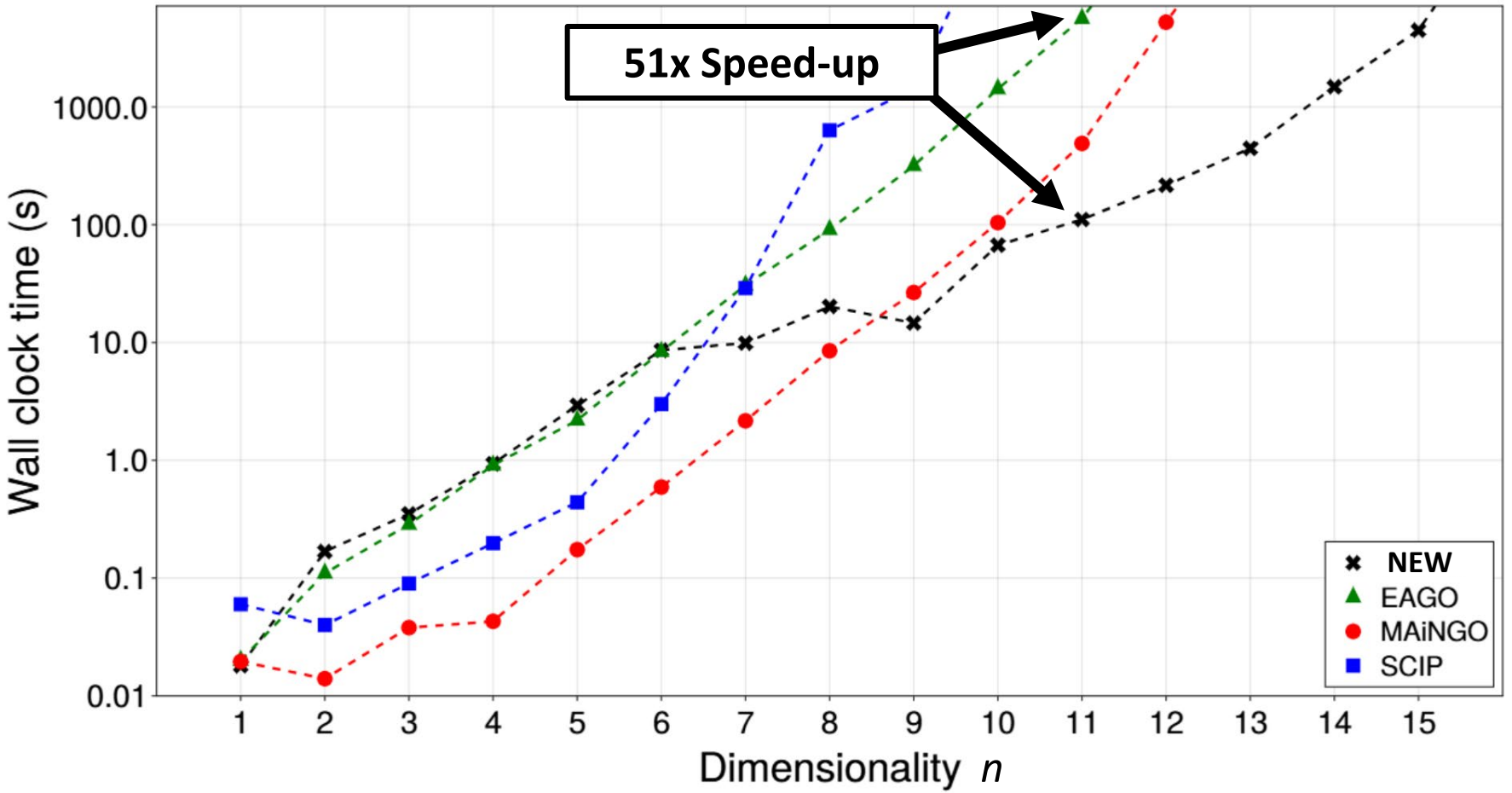
Scalable Benchmark Problem

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

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

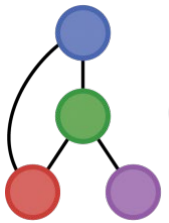
- Difficulty scales with dimension





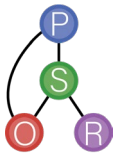
Wrapping Up

- 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)



Process Systems and Operations Research Laboratory

https://www.psor.uconn.edu



Acknowledgements

Members of the Process Systems and Operations Research Laboratory at the University of Connecticut

Process Systems and Operations Research Laboratory

https://www.psor.uconn.edu

Funding:

National Science Foundation, Award No.: **2330054**

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.