UCONN
UNIVERSITY OF CONNECTICUT

Source Code Transformation for GPU-Enhanced Deterministic Global Optimization

Robert Gottlieb, PhD Student
Matthew Stuber, Associate Professor

November 6th, 2023

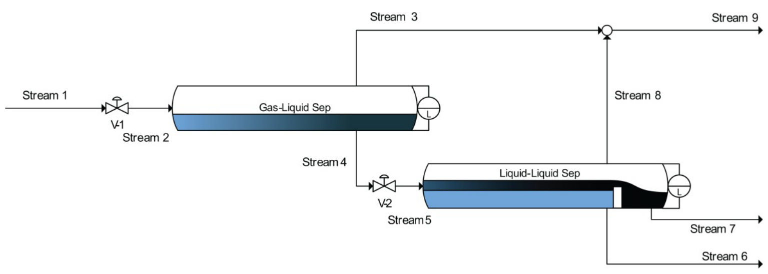
2023 AIChE ANNUAL MEETING

Process Systems and Operations Research Laboratory

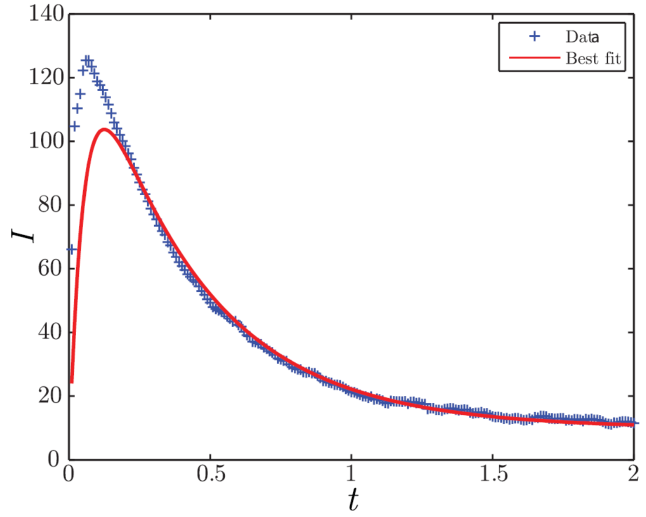
Deterministic Global Optimization

- **Nonconvex problems** naturally arise in many applications
- Guaranteed global solutions require specialized algorithms such as **branch-and-bound (B&B)**
- B&B is **computationally expensive**
  - Solvable problems typically have very few decision variables

***Worst-Case Design and Safety-Critical Systems*[1]**



***Parameter Estimation and Model Validation*[2]**

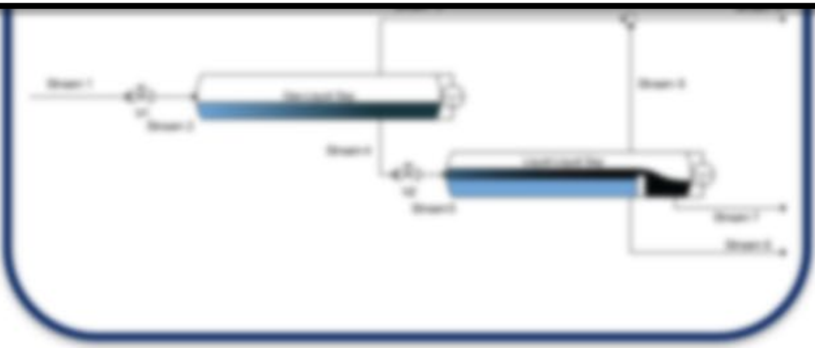


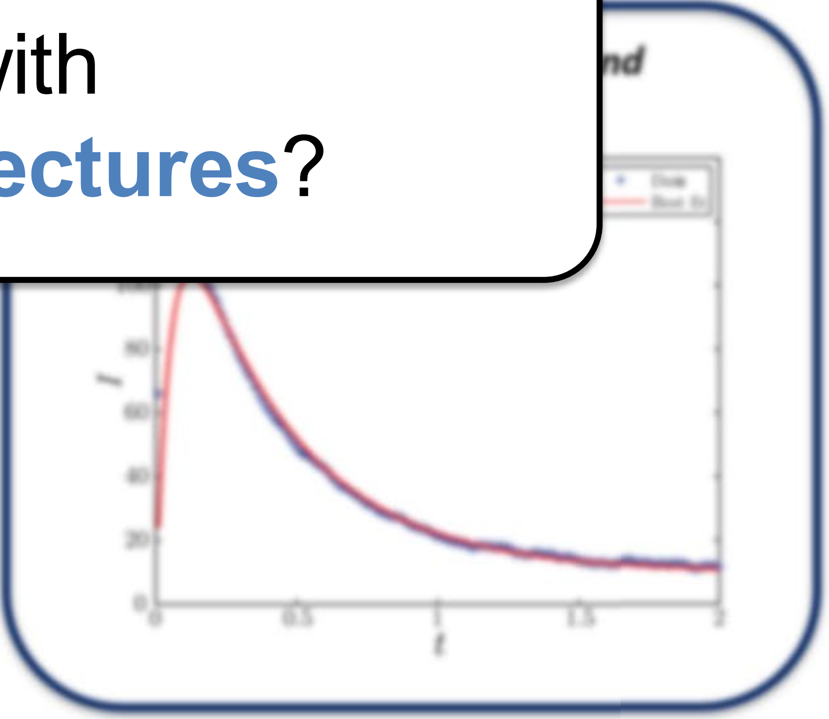
1. Stuber, M.D. et al. **Worst-case design of subsea production facilities using semi-infinite programming.** *AIChE Journal* (2014): 2513-2524.
2. Stuber, M.D. et al. **Convex and concave relaxations of implicit functions**. *Optimization Methods and Software* **30**(3), 424-460 (2014).

Deterministic Global Optimization

Can we **speed up B&B** with **parallelized computing architectures**?

- No…
  ari…
- Gu…
  rec…
  such as branch-and-bound (B&B)
- B&B is **computationally expensive**
  - Solvable problems typically have very few decision variables





1. Stuber, M.D. et al. **Worst-case design of subsea production facilities using semi-infinite programming.** *AIChE Journal* (2014): 2513-2524.
2. Stuber, M.D. et al. **Convex and concave relaxations of implicit functions.** *Optimization Methods and Software* **30**(3), 424-460 (2014).

Deterministic Global Optimization

Can we **speed up B&B** with **parallelized computing architectures**?

GPUs have been successfully used for **ML**, **data analysis**, etc.

1. Stuber, M.D. et al. **Worst-case design of subsea production facilities using semi-infinite programming.** *AIChE Journal* (2014): 2513-2524.
2. Stuber, M.D. et al. **Convex and concave relaxations of implicit functions.** *Optimization Methods and Software* **30**(3), 424-460 (2014).

Why GPUs?

Strengths

- Faster calculation speed
- More efficient energy utilization
- More cost effective than CPUs for scale-up

Why GPUs?

Strengths

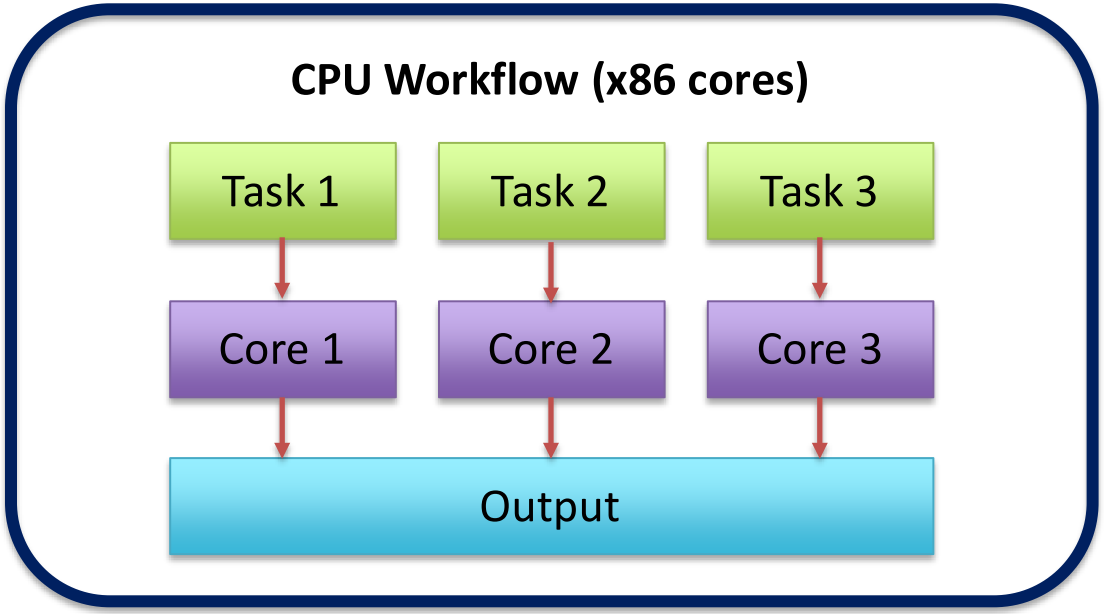
- Faster calculation speed
- More efficient energy utilization
- More cost effective than CPUs for scale-up

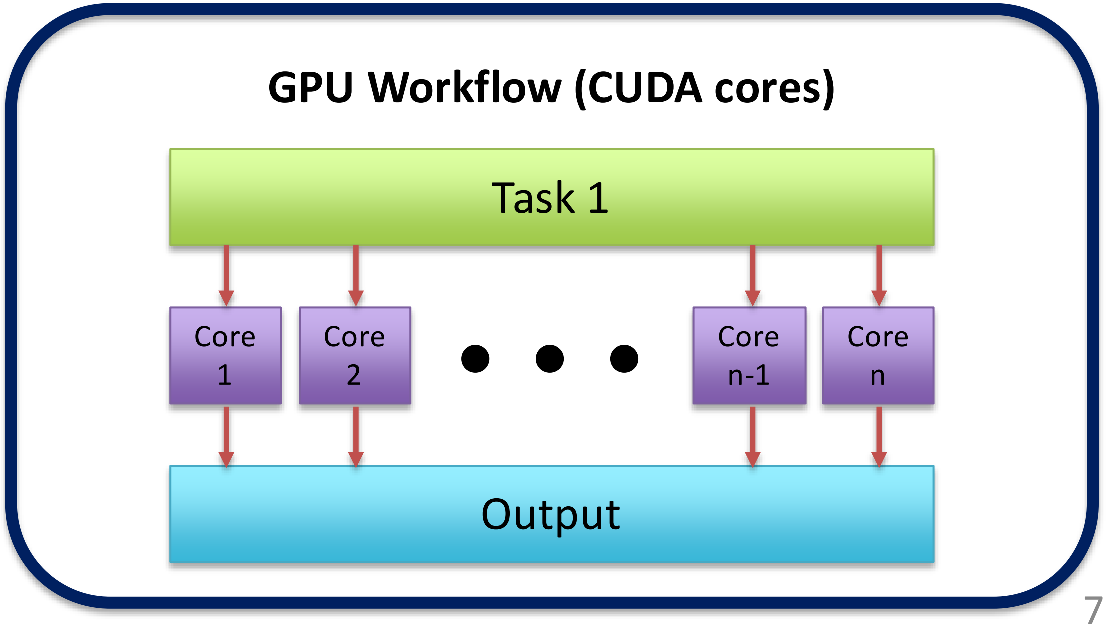
Weaknesses

- Standard B&B software not automatically compatible with GPUs
  - Requires re-architecting algorithms to be data-parallel
- "Branches" in code massively degrade performance

CPU vs. GPU Parallelism

- Multicore CPUs use **task parallelism (MIMD)**
  - Different cores perform **different tasks** independently
- GPUs use **data parallelism (SIMD)**
  - Different cores perform the **same task** on **different portions of data**
  - **Efficient** with a pipeline: minimal decision-making, minimal branches based on data

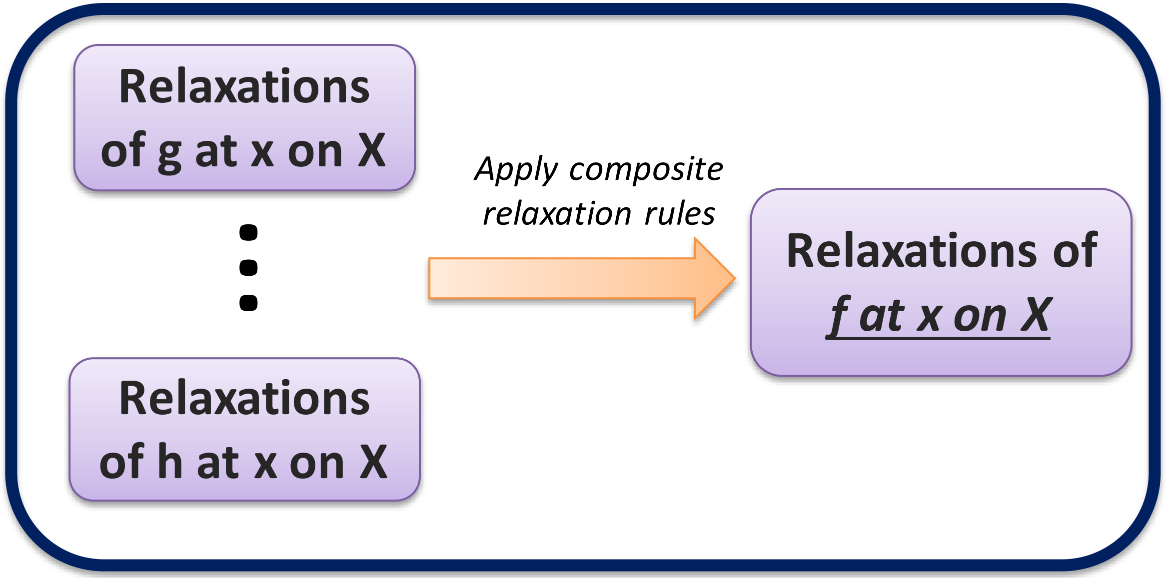




McCormick Relaxations of Factorable Functions

$$\mathbf{y} = \mathbf{f}(\mathbf{g}(\mathbf{x}), \ldots, \mathbf{h}(\mathbf{x}))$$

McCormick-Based Relaxations[5,6]



https://www.github.com/PSORLab/EAGO.jl

5. Mitsos, A., et al. **McCormick-based relaxations of algorithms.** *SIAM Journal on Optimization*, SIAM (2009) 20, 73-601.
6. Scott, J.K., et al. **Generalized McCormick relaxations.** *Journal of Global Optimization* 51.4 (2011): 569-606.

McCormick.jl

1) Create a library of math operators, overloaded* to apply McCormick rules

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

*Multiple dispatch, in Julia

McCormick.jl

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

1) Create a library of math operators, overloaded* to apply McCormick rules

2) Create “McCormick objects” for variables {*x*, *y*} with specified bounds and pointwise values

*Multiple dispatch, in Julia

McCormick.jl

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

1) Create a library of math operators, overloaded* to apply McCormick rules

2) Create “McCormick objects” for variables {*x*, *y*} with specified bounds and pointwise values

3) Evaluate the math expression using McCormick objects

*Multiple dispatch, in Julia

McCormick.jl

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

Relaxations at specified values/bounds of *x*, *y*

1) Create a library of math operators, overloaded* to apply McCormick rules

2) Create "McCormick objects" for variables {*x*, *y*} with specified bounds and pointwise values

3) Evaluate the math expression using McCormick objects

*Multiple dispatch, in Julia

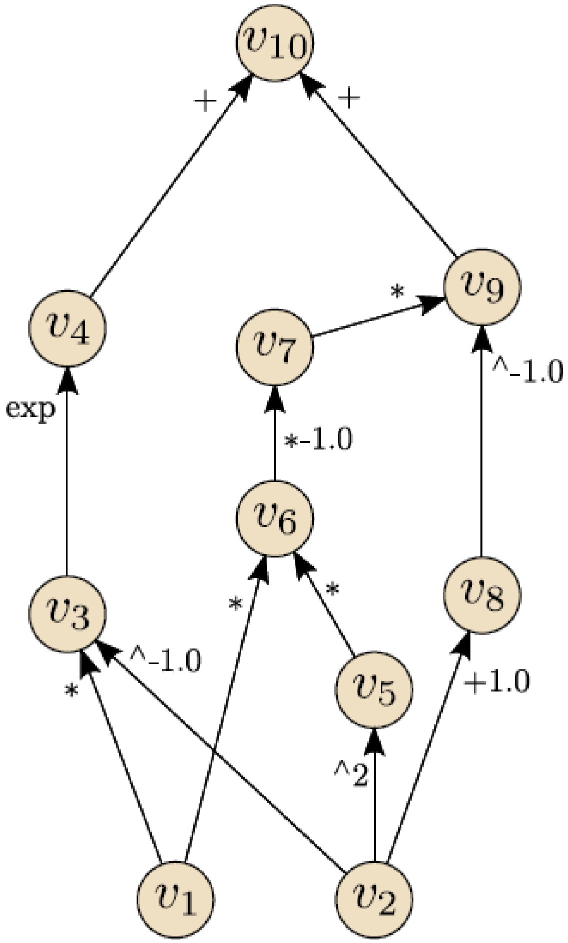
SourceCodeMcCormick.jl

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

SourceCodeMcCormick.jl

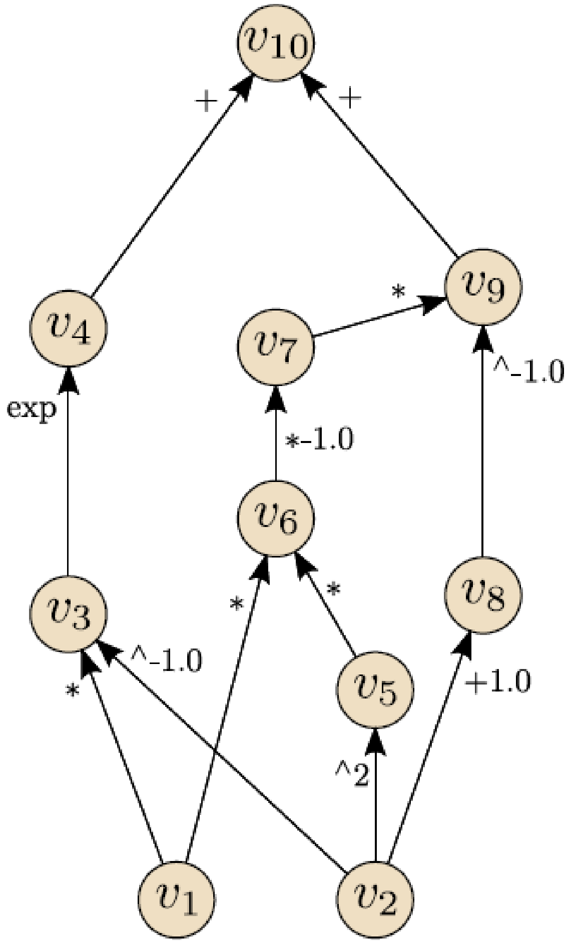
$$\exp\left(x \,/\, y\right) - xy^2 \,/\, \left(y + 1\right) \longrightarrow$$

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



SourceCodeMcCormick.jl

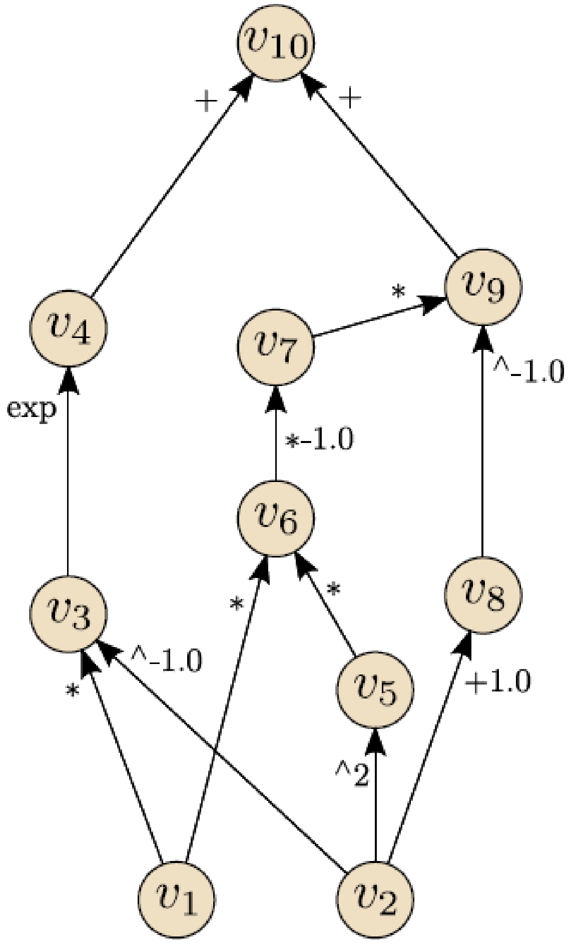
$$v_1 = x$$
$$v_2 = y$$
$$v_3 = v_1/v_2$$
$$v_4 = \exp(v_3)$$
$$v_5 = v_2^2$$
$$v_6 = v_1 v_5$$
$$v_7 = -v_6$$
$$v_8 = v_2 + 1.0$$
$$v_9 = v_7/v_8$$
$$v_{10} = v_4 + v_9$$



1) Factor original math expression

SourceCodeMcCormick.jl

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

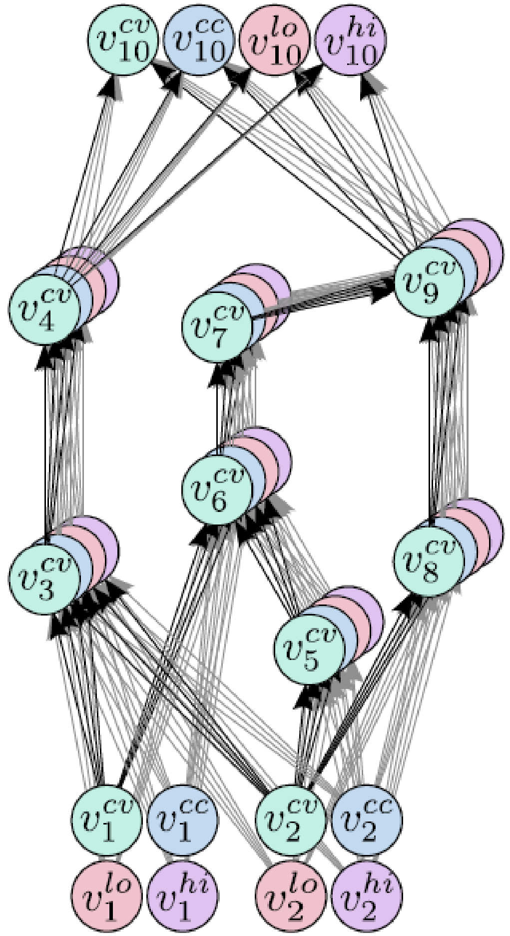


1) Factor original math expression

2) Replace each factor with code capturing all variations of that McCormick rule

SourceCodeMcCormick.jl

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

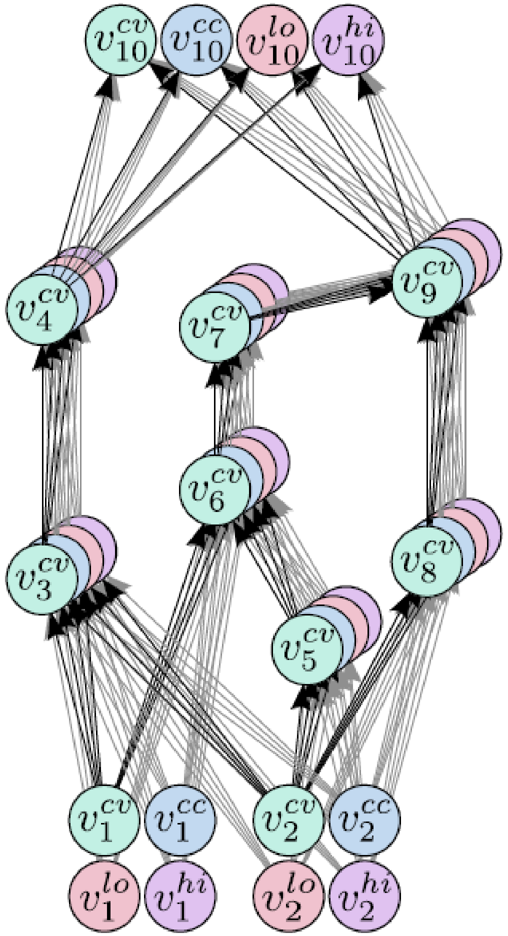


1) Factor original math expression

2) Replace each factor with code capturing all variations of that McCormick rule

SourceCodeMcCormick.jl

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



1) Factor original math expression
2) Replace each factor with code capturing all variations of that McCormick rule
3) Compile code into an “evaluator function”

SourceCodeMcCormick.jl

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

↓

new_func(x^{cv}, x^{cc}, x^L, x^U, y^{cv}, y^{cc}, y^L, y^U)

↓

Plug in values/bounds of *x*, *y* to obtain relaxations

1) Factor original math expression

2) Replace each factor with code capturing all variations of that McCormick rule

3) Compile code into an "evaluator function"

SourceCodeMcCormick.jl

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

1) Factor original math expression
2) Replace each factor with code

to obtain relaxations

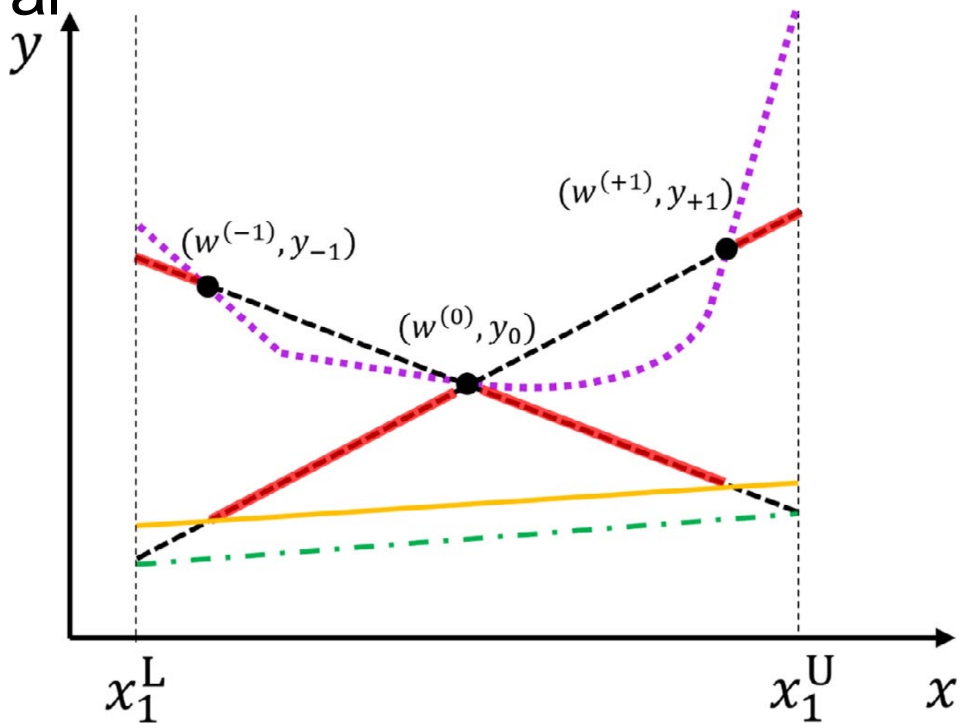
Fully compatible with GPUs

Pointwise evaluations **~3 OOM faster** than McCormick.jl

Past Hurdles

Past Hurdles

1) Only returned **relaxations** and natural interval extensions (**no subgradients**)
   - ➢ Reliant on subgradient-free lower-bounding methods
   - ➢ Cannot handle non-trivial constraints

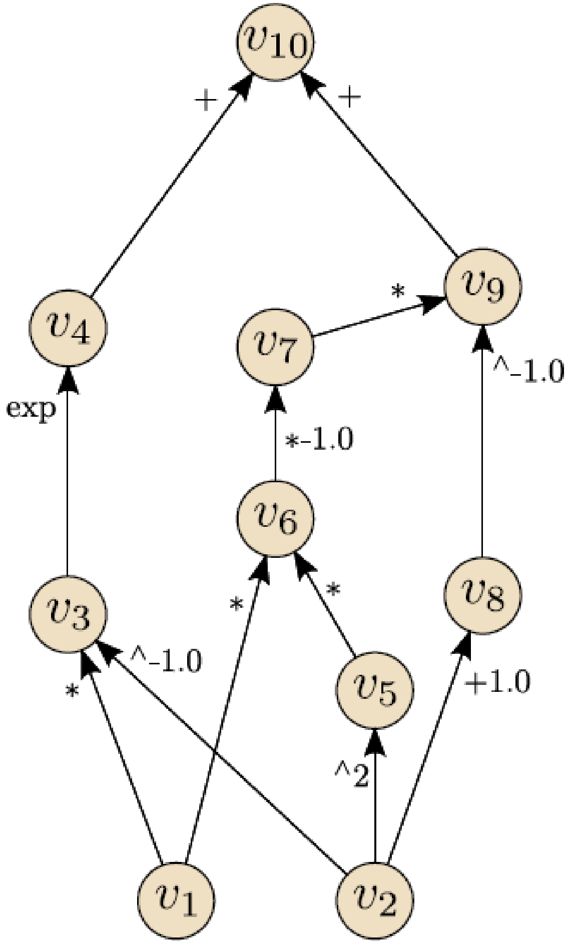


4. Song, Y., et al. **Bounding convex relaxations of process models from below by tractable black-box sampling.** *Computers & Chemical Engineering* 153 (2021), 107413.

Past Hurdles

1) Only returned **relaxations** and natural interval extensions (**no subgradients**)
   - Reliant on subgradient-free lower-bounding methods
   - Cannot handle non-trivial constraints

2) Relaxation rules have **combinatorial complexity**
   - Large compile times
   - Greater chance of branching in rules

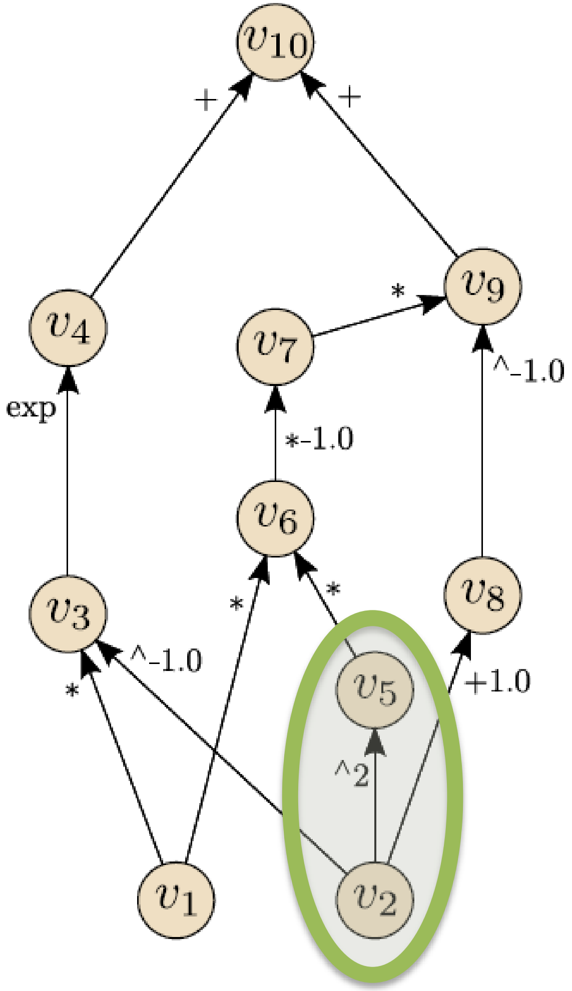


4. Song, Y., et al. **Bounding convex relaxations of process models from below by tractable black-box sampling.** *Computers & Chemical Engineering* 153 (2021), 107413.

Past Hurdles

1) Only returned **relaxations** and natural interval extensions (**no subgradients**)
   - ➢ Reliant on subgradient-free lower-bounding methods
   - ➢ Cannot handle non-trivial constraints

2) Relaxation rules have **combinatorial complexity**
   - ➢ Large compile times
   - ➢ Greater chance of branching in rules

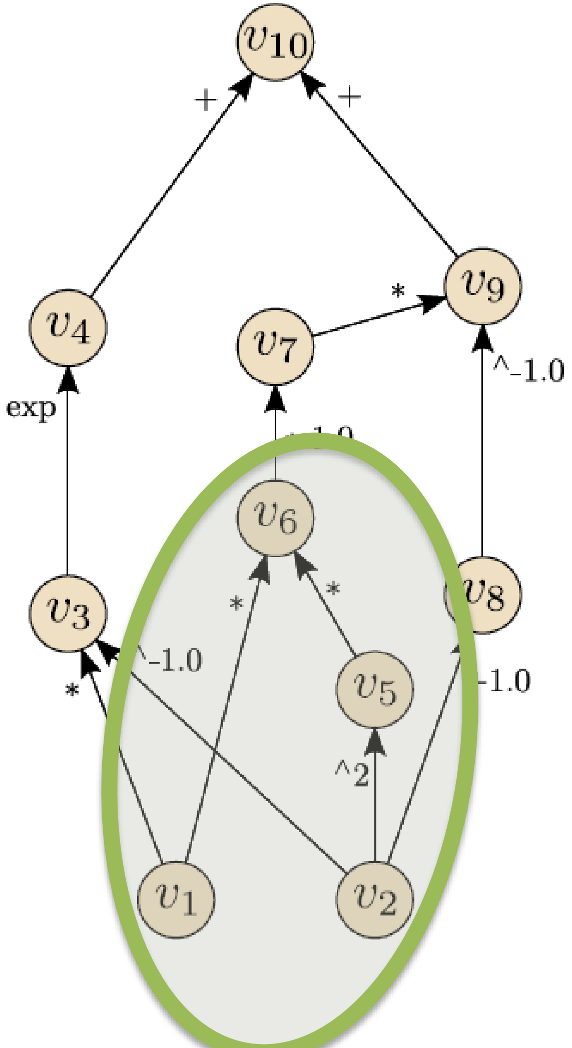


4. Song, Y., et al. **Bounding convex relaxations of process models from below by tractable black-box sampling.** *Computers & Chemical Engineering* 153 (2021), 107413.

Past Hurdles

1) Only returned **relaxations** and natural interval extensions (**no subgradients**)
   - ➢ Reliant on subgradient-free lower-bounding methods
   - ➢ Cannot handle non-trivial constraints

2) Relaxation rules have **combinatorial complexity**
   - ➢ Large compile times
   - ➢ Greater chance of branching in rules

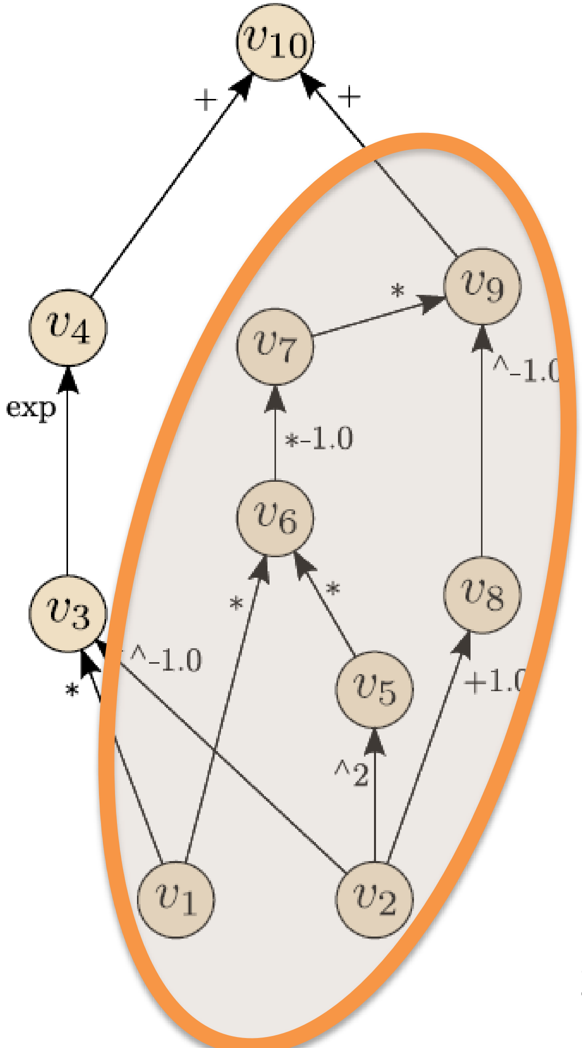


4. Song, Y., et al. **Bounding convex relaxations of process models from below by tractable black-box sampling.** *Computers & Chemical Engineering* 153 (2021), 107413.

Past Hurdles

1) Only returned **relaxations** and natural interval extensions (**no subgradients**)
   - ➢ Reliant on subgradient-free lower-bounding methods
   - ➢ Cannot handle non-trivial constraints

2) Relaxation rules have **combinatorial complexity**
   - ➢ Large compile times
   - ➢ Greater chance of branching in rules

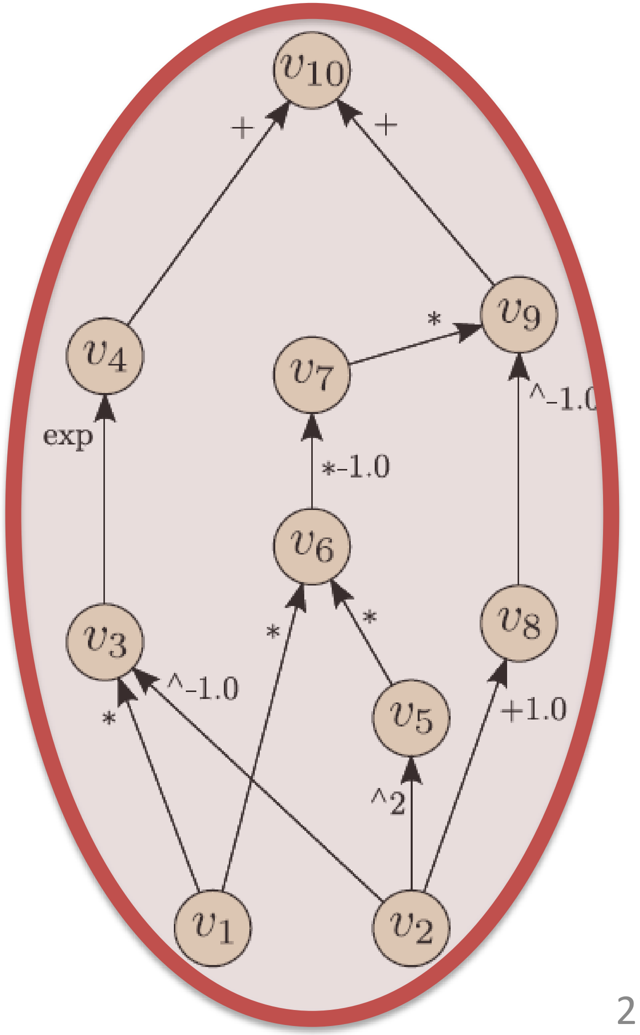


4. Song, Y., et al. **Bounding convex relaxations of process models from below by tractable black-box sampling.** *Computers & Chemical Engineering* 153 (2021), 107413.

Past Hurdles

1) Only returned **relaxations** and natural interval extensions (**no subgradients**)
   - ➢ Reliant on subgradient-free lower-bounding methods
   - ➢ Cannot handle non-trivial constraints

2) Relaxation rules have **combinatorial complexity**
   - ➢ Large compile times
   - ➢ Greater chance of branching in rules

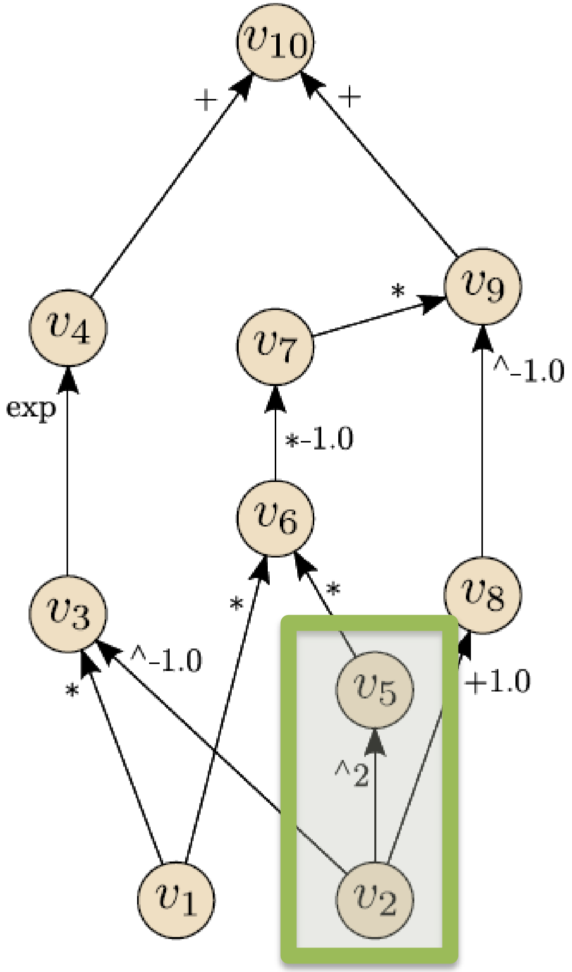


4. Song, Y., et al. **Bounding convex relaxations of process models from below by tractable black-box sampling.** *Computers & Chemical Engineering* 153 (2021), 107413.

Past Hurdles

1) Only returned **relaxations** and natural interval extensions (**no subgradients**)
   - ➢ Reliant on subgradient-free lower-bounding methods
   - ➢ Cannot handle non-trivial constraints

2) Relaxation rules have **combinatorial complexity**
   - ➢ Large compile times
   - ➢ Greater chance of branching in rules

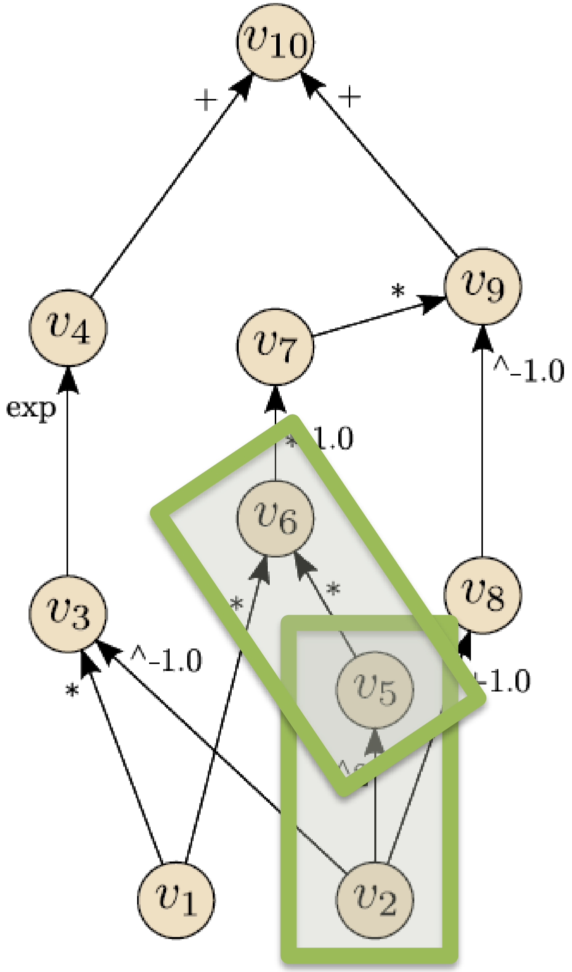


4. Song, Y., et al. **Bounding convex relaxations of process models from below by tractable black-box sampling.** *Computers & Chemical Engineering* 153 (2021), 107413.

Past Hurdles

1) Only returned **relaxations** and natural interval extensions (**no subgradients**)
   - ➢ Reliant on subgradient-free lower-bounding methods
   - ➢ Cannot handle non-trivial constraints

2) Relaxation rules have **combinatorial complexity**
   - ➢ Large compile times
   - ➢ Greater chance of branching in rules

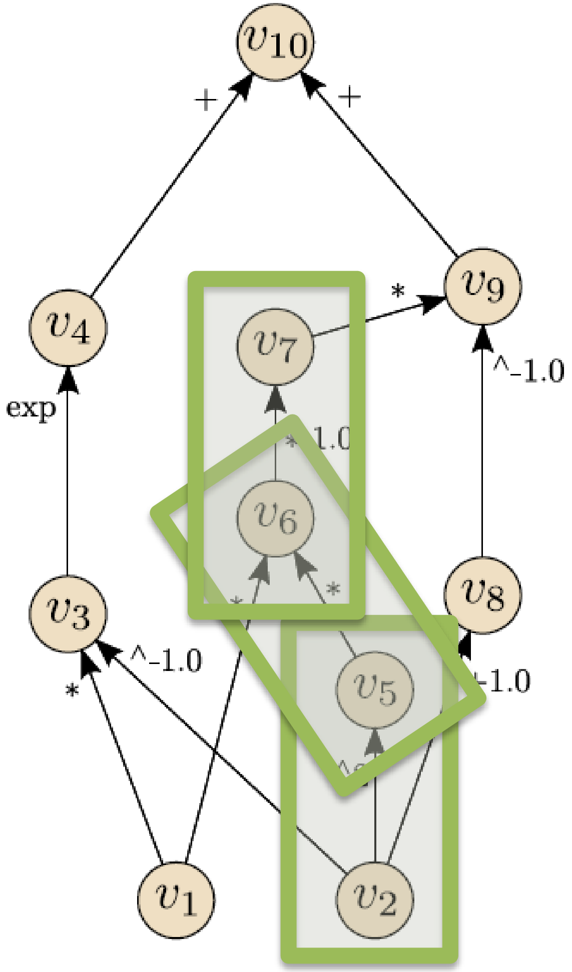


4. Song, Y., et al. **Bounding convex relaxations of process models from below by tractable black-box sampling.** *Computers & Chemical Engineering* 153 (2021), 107413.

Past Hurdles

1) Only returned **relaxations** and natural interval extensions (**no subgradients**)
   - ➢ Reliant on subgradient-free lower-bounding methods
   - ➢ Cannot handle non-trivial constraints

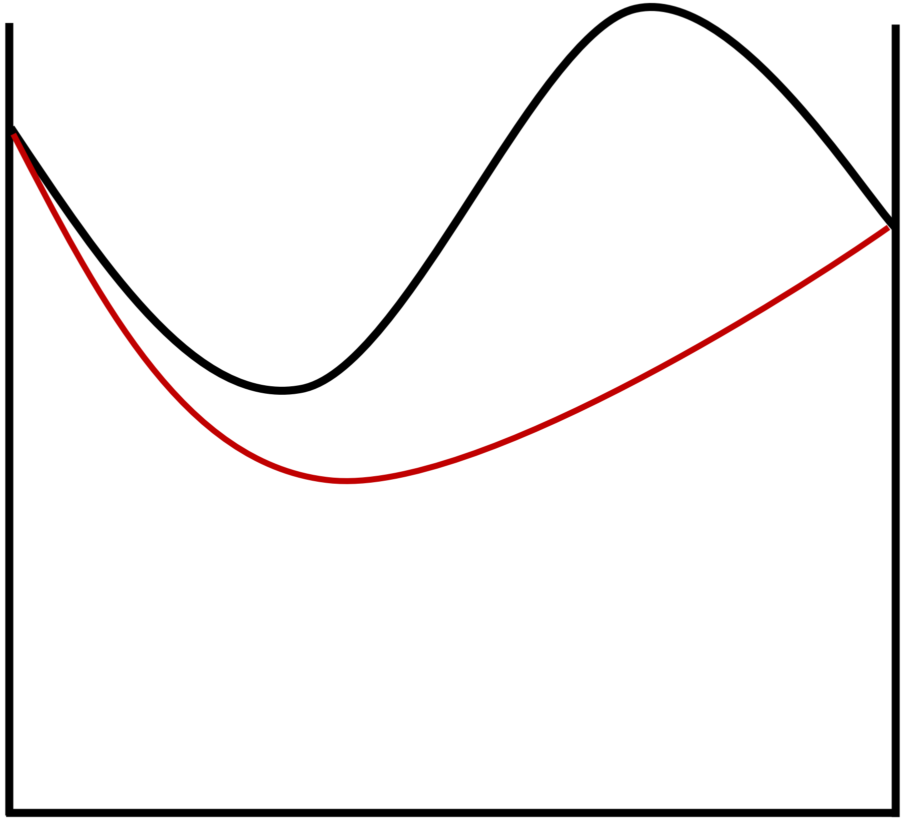
2) Relaxation rules have **combinatorial complexity**
   - ➢ Large compile times
   - ➢ Greater chance of branching in rules



4. Song, Y., et al. **Bounding convex relaxations of process models from below by tractable black-box sampling.** *Computers & Chemical Engineering* 153 (2021), 107413.

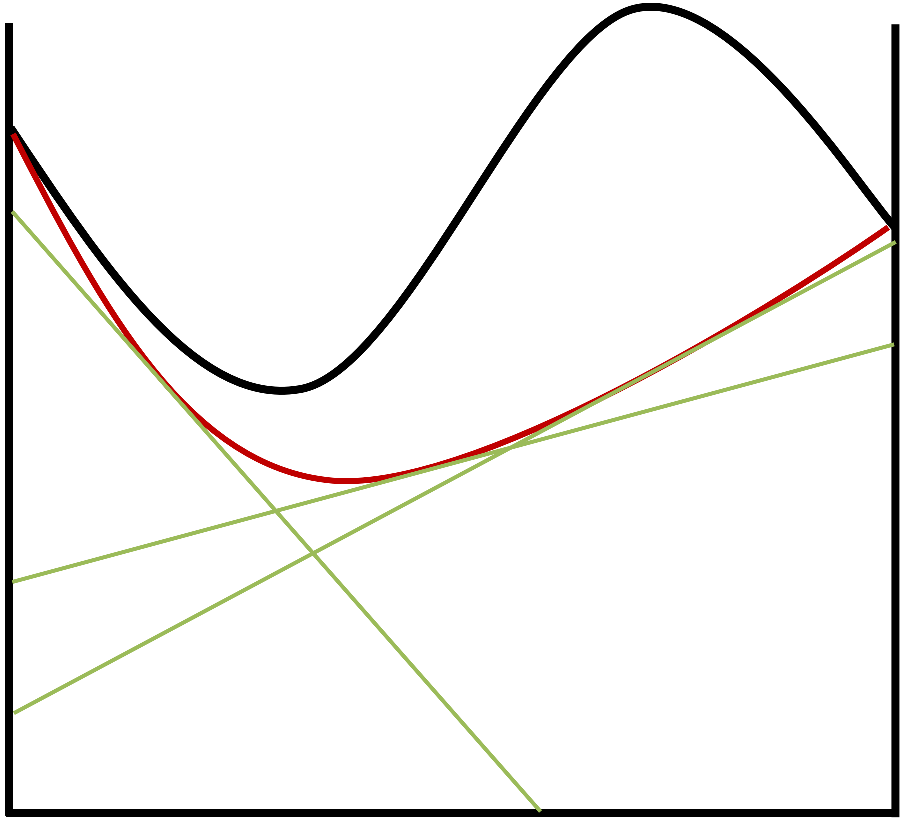
Newest Improvements

1) Can now handle subgradients!



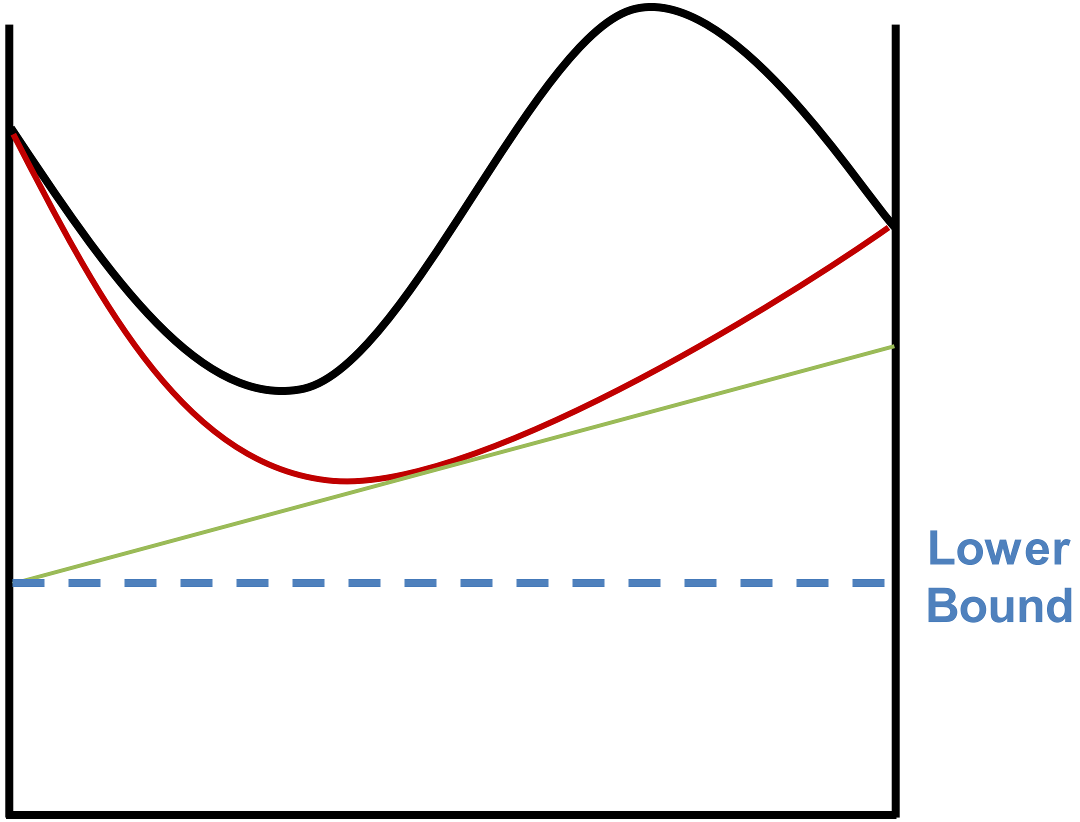
Newest Improvements

1) Can now handle subgradients!



Newest Improvements

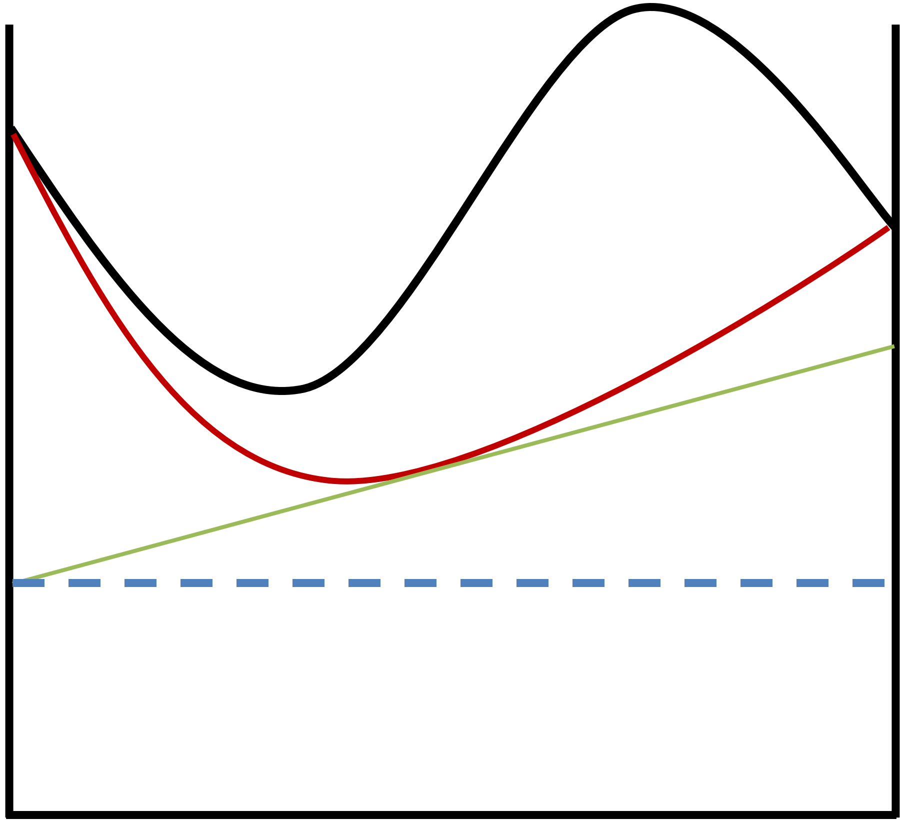
1) Can now handle subgradients!



5. Najman, J., Mitsos, A. **Tighter McCormick relaxations through subgradient propagation.** *J Glob Optim* **75**, 565–593 (2019).

Newest Improvements

1) Can now handle subgradients!
   - Does not address nontrivial constraints

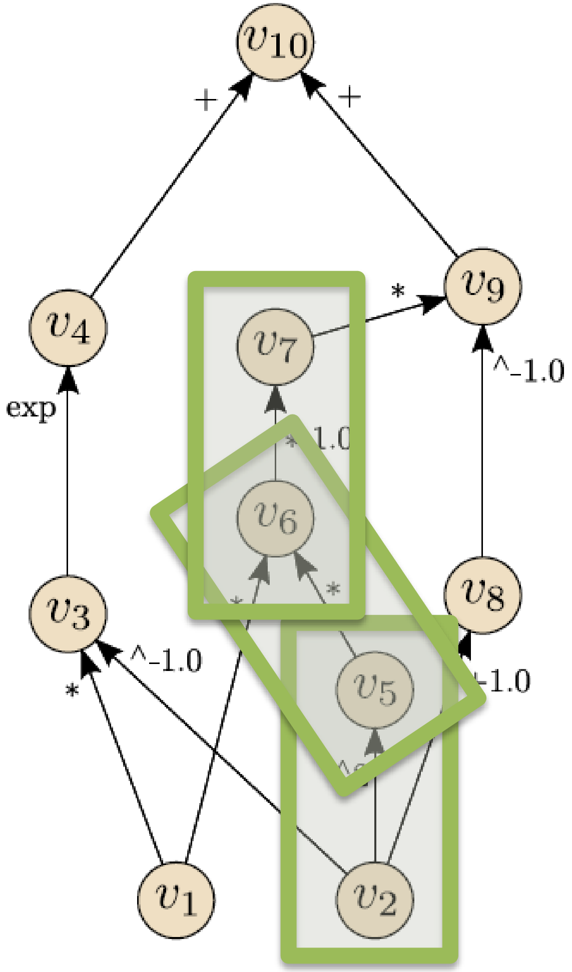


5. Najman, J., Mitsos, A. **Tighter McCormick relaxations through subgradient propagation.** *J Glob Optim* **75**, 565–593 (2019).

Newest Improvements

1) Can now handle subgradients!
   - ➢ Does not address nontrivial constraints

2) Automatic function generation!
   - ➢ Evaluator functions stitched into larger function
   - ➢ Faster compilation times



Newest Improvements

1) Can now handle subgradients!
   - Does not address nontrivial constraints

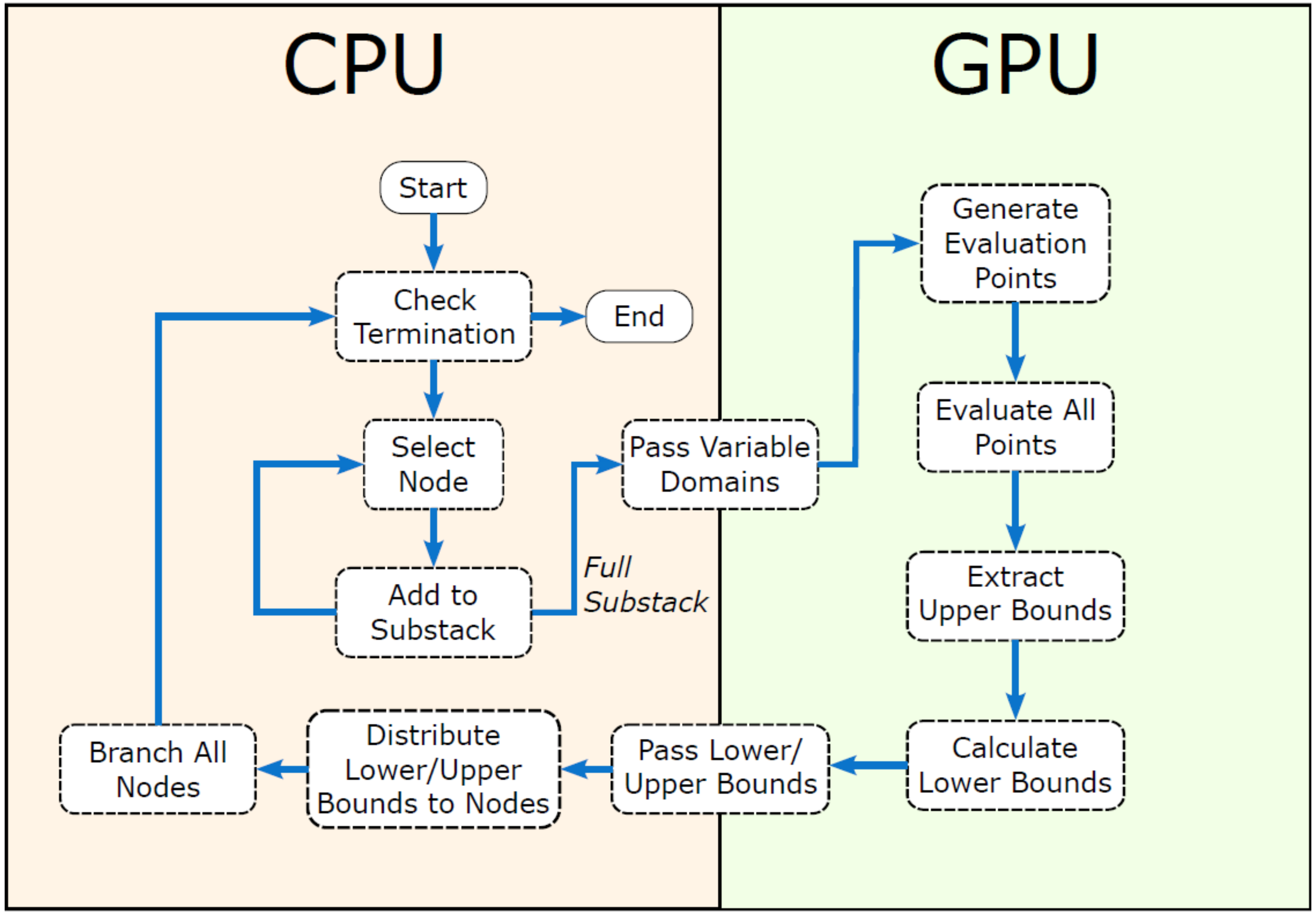
2) Automatic function generation!
   - Evaluator functions stitched into larger function
   - Faster compilation times

$$\log(\pi^{\text{calc}}) = \sum_{i=0}^{2} a_i w^i + \frac{\sum_{i=0}^{2} b_i w^i}{T}$$

```
@variables a0, a1, a2, b0, b1, b2, data, W, T
expr = exp(a0 + a1*W + a2*W^2 + (1/T)*(b0 + b1*W + b2*W^2))

new_func = fgen(((expr-data)/data)^2, constants=[data, W, T])
```

ParBB



6. Gottlieb, R.X., Xu, P., Stuber, M.D. **Automatic source code generation for deterministic global optimization with parallel architectures.** *Under Review.*

Kinetic Parameter Estimation

Concentrations after an initial laser flash pyrolysis are modeled using the system of ODEs:[8]

$$\frac{dx_A}{dt} = k_1 x_Z x_Y - c_{O_2}(k_{2f} + k_{3f})x_A + \frac{k_{2f}}{K_2}x_D + \frac{k_{3f}}{K_3}x_B - k_5 x_A^2,$$

$$\frac{dx_B}{dt} = c_{O_2}k_{3f}x_A - \left(\frac{k_{3f}}{K_3} + k_4\right)x_B,$$

$$\frac{dx_D}{dt} = c_{O_2}k_{2f}x_A - \frac{k_{2f}}{K_2}x_D,$$

$$\frac{dx_Y}{dt} = -k_{1s}x_Z x_Y,$$

$$\frac{dx_Z}{dt} = -k_1 x_z x_Y, \quad x_A(0) = x_B(0) = x_D(0) = 0, \quad x_Y(0) = 0.4, \quad x_Z(0) = 140.$$

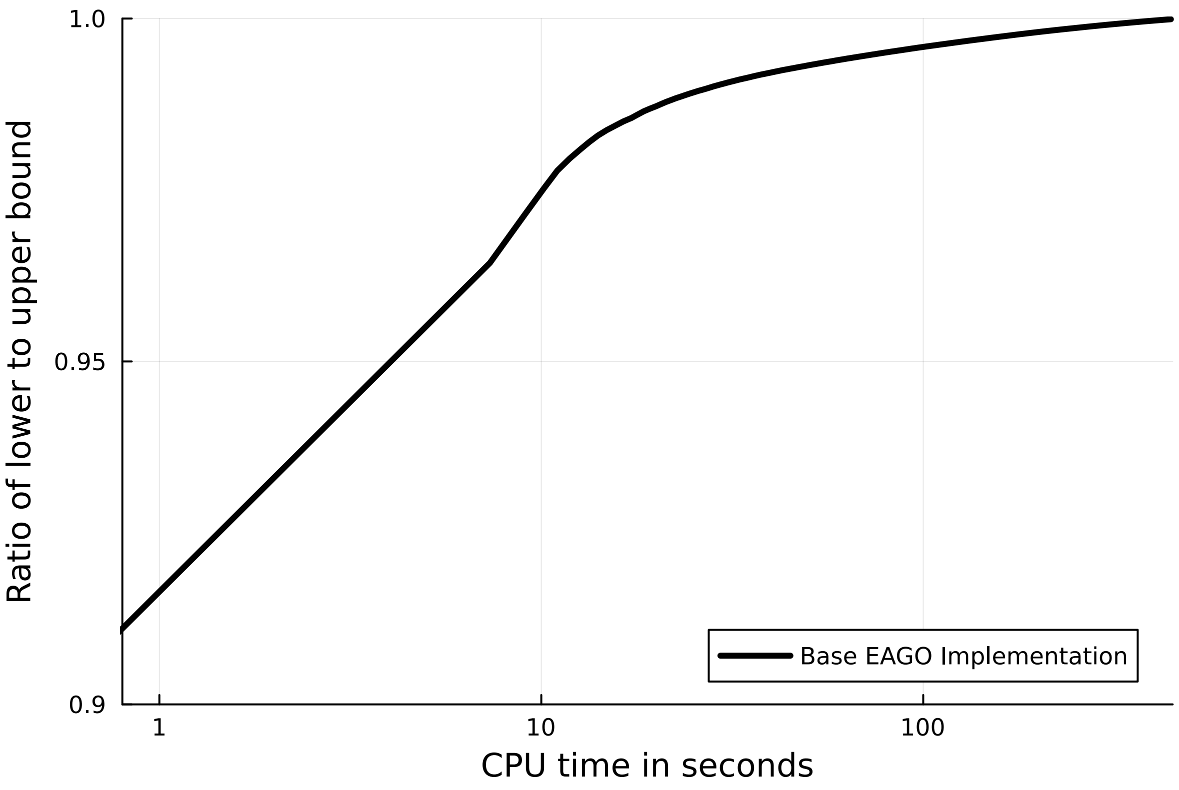
$$I = x_A + \frac{2}{21}x_B + \frac{2}{21}x_D$$

$$f(\mathbf{p}) = \sum_{i=0}^{N}\left(I^{\mathrm{calc}}(\mathbf{x}_i, \mathbf{p}) - I_i^{\exp}\right)^2$$

7. Taylor, J. W. **Direct Measurement and Analysis of Cyclohexadienyl Oxidation.** Ph.D. thesis, Massachusetts Institute of Technology.

Results

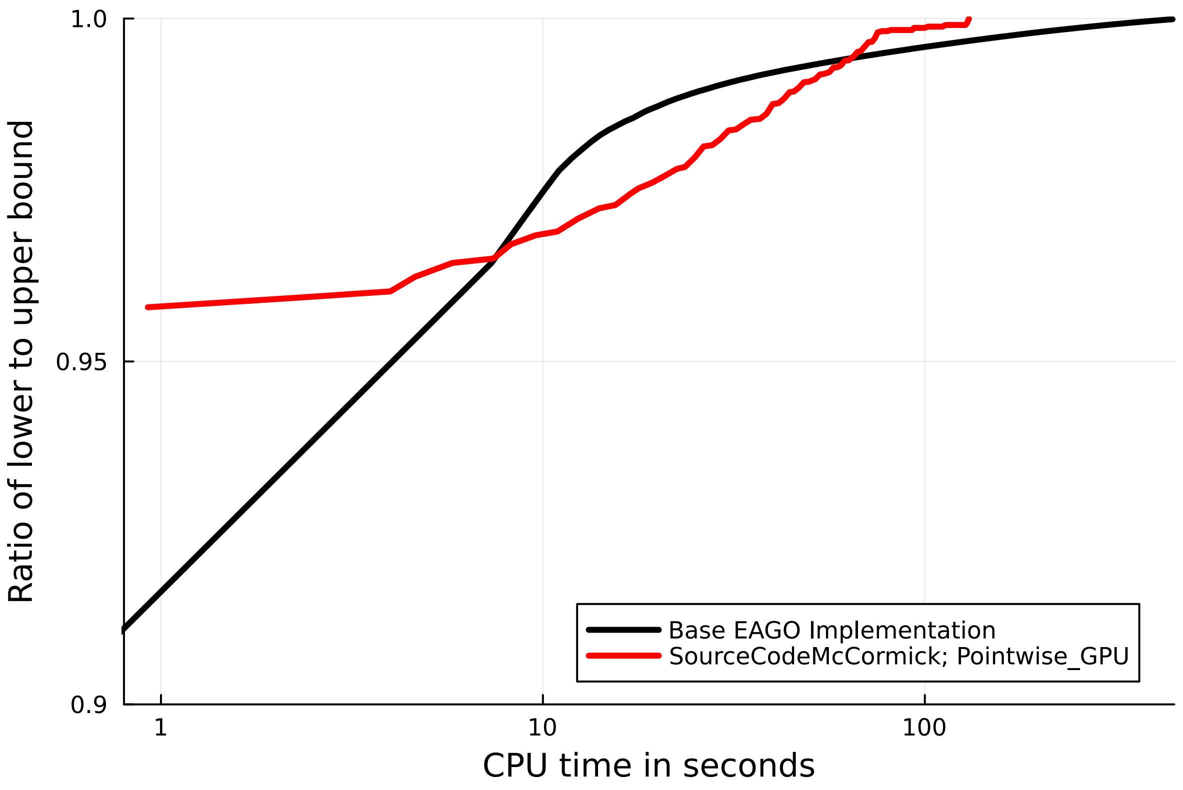
Solution Method	Convergence Time (s)	Nodes Accessed
Base EAGO	445.1	8.4E5



CPU: Intel W-2195
GPU: NVIDIA Quadro GV100

Results

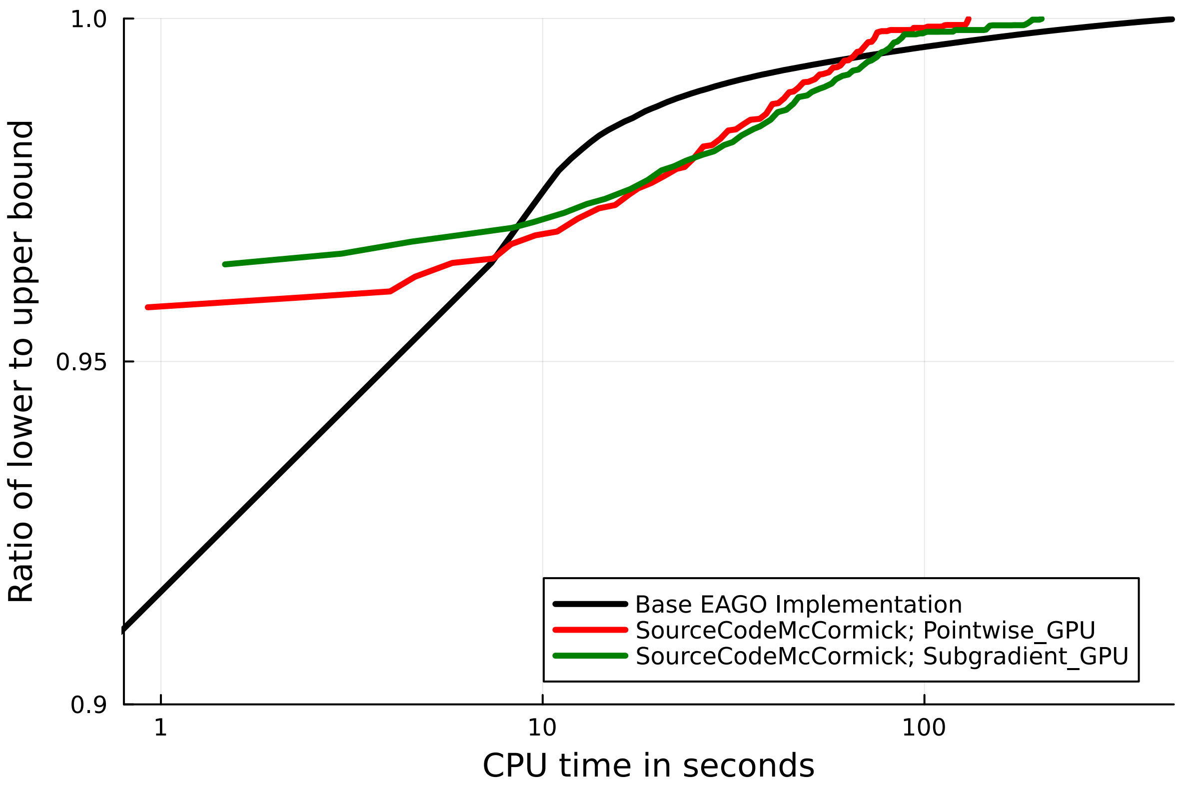
Solution Method	Convergence Time (s)	Nodes Accessed
Base EAGO	445.1	8.4E5
Pointwise GPU	130.4	4.5E6



CPU: Intel W-2195
GPU: NVIDIA Quadro GV100

Results

Solution Method	Convergence Time (s)	Nodes Accessed
Base EAGO	445.1	8.4E5
Pointwise GPU	130.4	4.5E6
Subgradient GPU	202.7	5.2E6



CPU: Intel W-2195
GPU: NVIDIA Quadro GV100

Conclusions

- Evaluations of relaxations and subgradients **performant on GPU**
- GPU-based B&B algorithm **implemented** in SourceCodeMcCormick.jl
- Current method **cannot handle non-trivial constraints**
  - Would require batch parallelized GPU LP solver

Conclusions

- Evaluations of relaxations and subgradients **performant on GPU**
- GPU-based B&B algorithm **implemented** in SourceCodeMcCormick.jl
- Current method **cannot handle non-trivial constraints**
  - Would require batch parallelized GPU LP solver*

Acknowledgements

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

UCONN
UNIVERSITY OF CONNECTICUT

Process Systems and Operations Research Laboratory

Funding:

National Science Foundation, Award No.: **1932723**

DOE / EERE / AMO Award No.: **DE-EE0009497**

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, the Department of Energy, or the United States Government.

Questions?

Process Systems and Operations Research Laboratory

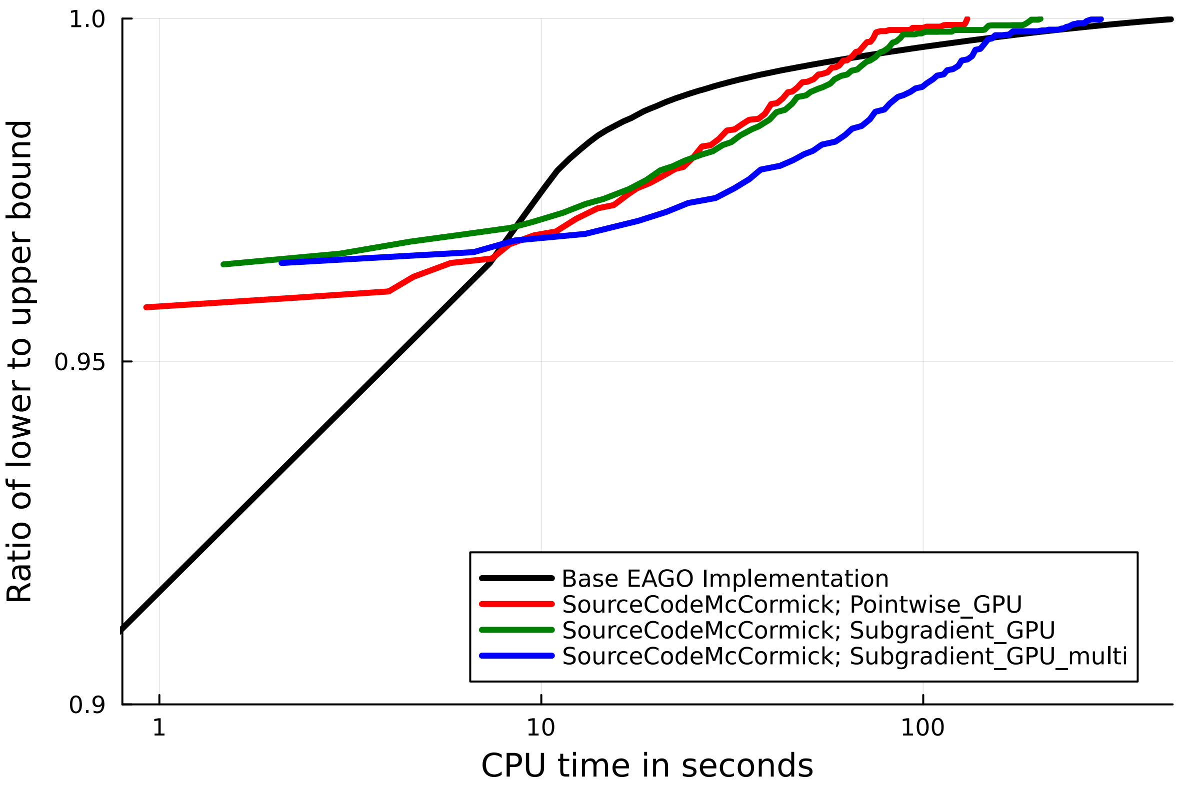
https://www.psor.uconn.edu

EAGO

https://www.github.com/PSORLab/EAGO.jl

Results

Solution Method	Convergence Time (s)	Nodes Accessed
Base EAGO	445.1	8.4E5
Pointwise GPU	130.4	4.5E6
Subgradient GPU	202.7	5.2E6
Subgradient GPU (multi)	291.9	4.3E6



CPU: Intel W-2195
GPU: NVIDIA Quadro GV100