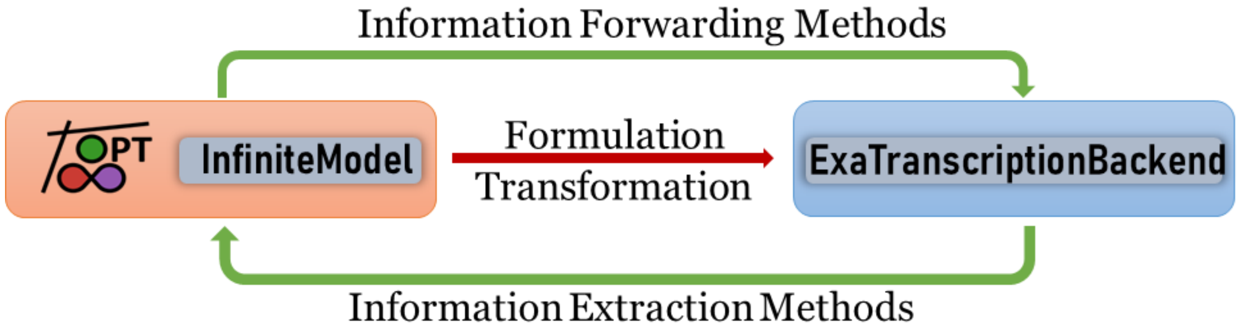
INFINITEEXAMODELS.JL: ACCELERATING INFINITE-DIMENSIONAL OPTIMIZATION PROBLEMS ON CPU & GPU

7/29/2024

Joshua Pulsipher and Sungho Shin



UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

ACKNOWLEDGEMENTS

Sungho Shin
MIT
Assistant Professor

François Pacaud
Mines Paris
Assistant Professor

Mihai Anitescu
Argonne
Senior Computational Mathematician

UNIVERSITY OF WATERLOO
Department of Chemical Engineering

DEPARTMENT OF ENERGY · UNITED STATES OF AMERICA

ECP
EXASCALE COMPUTING PROJECT

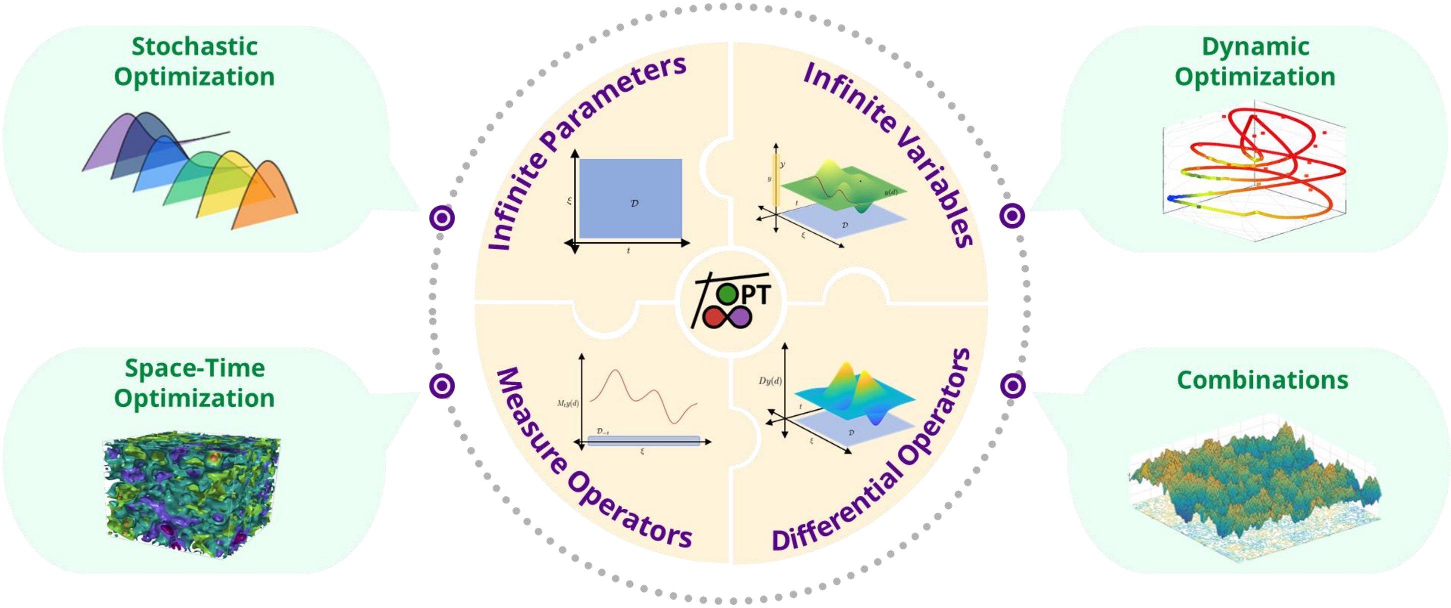
UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

OUTLINE

- InfiniteOpt
- ExaModels
- InfiniteExaModels

OUTLINE

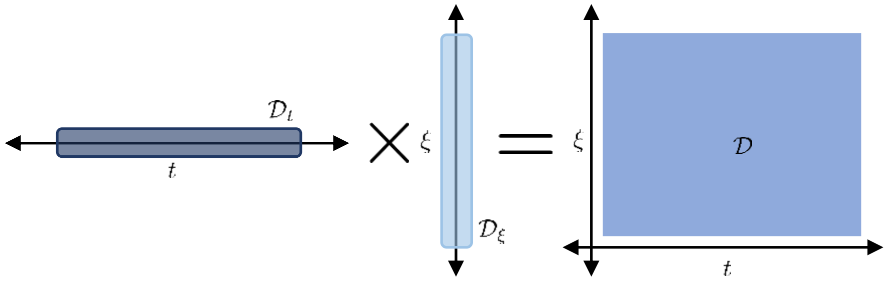
- **InfiniteOpt**
- ExaModels
- InfiniteExaModels



INFINITE-DIMENSIONAL OPTIMIZATION

Infinite Parameters

- Index over **continuous domains**



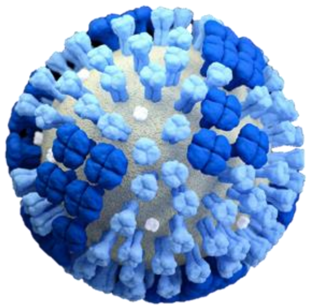
$t \in \mathcal{D}_t$ $\xi \in \mathcal{D}_\xi$ $d \in \mathcal{D} = \mathcal{D}_t \times \mathcal{D}_\xi$

- **Example:** Disease Control
  - Population dynamics

$$t \in [0, t_f]$$

  - Uncertain infection rates

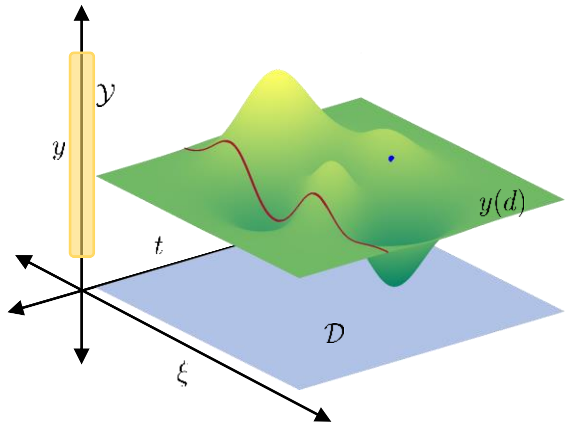
$$\xi \in (-\infty, \infty) \sim \mathcal{N}(\mu, \Sigma)$$



Infinite Variables

- **Decisions** indexed by infinite parameters

$y(d)$



- **Example:** Disease Control
  - Population of infected at a particular time and infection rate

$$y_i(t, \xi)$$

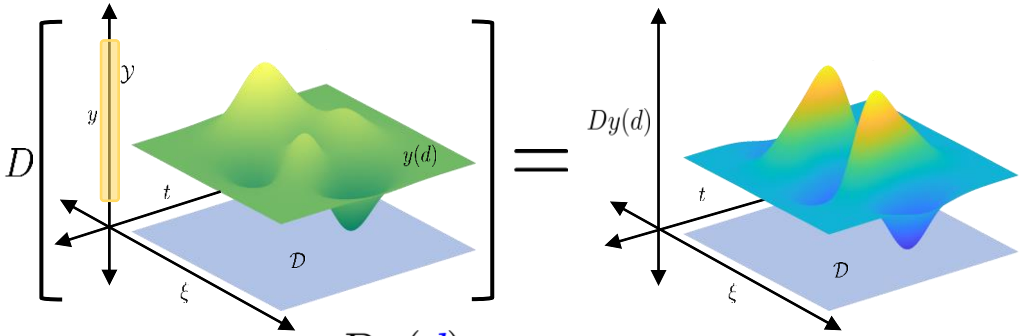
J. L. Pulsipher, W. Zhang, T. J. Hongisto, and V. M. Zavala. "A unifying modeling abstraction for infinite-dimensional optimization." 2022

UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

INFINITE-DIMENSIONAL OPTIMIZATION

Differential Operators

- Capture of **rate of change** in variables

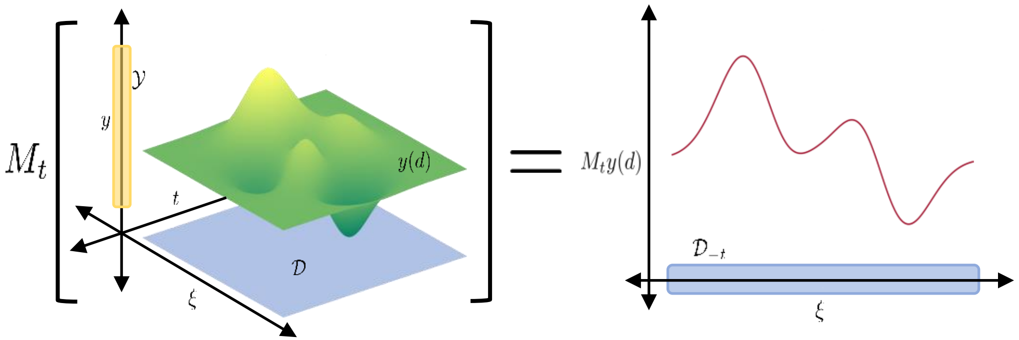


$Dy(d)$

- **Example:** Disease Control
  - Time derivative $$\frac{\partial y_i(t,\xi)}{\partial t}$$
  - SEIR model $$\frac{\partial y_i(t,\xi)}{\partial t} = \xi y_e(t) - \gamma y_i(t)$$

Measure Operators

- **Summarize variables** over continuous domains



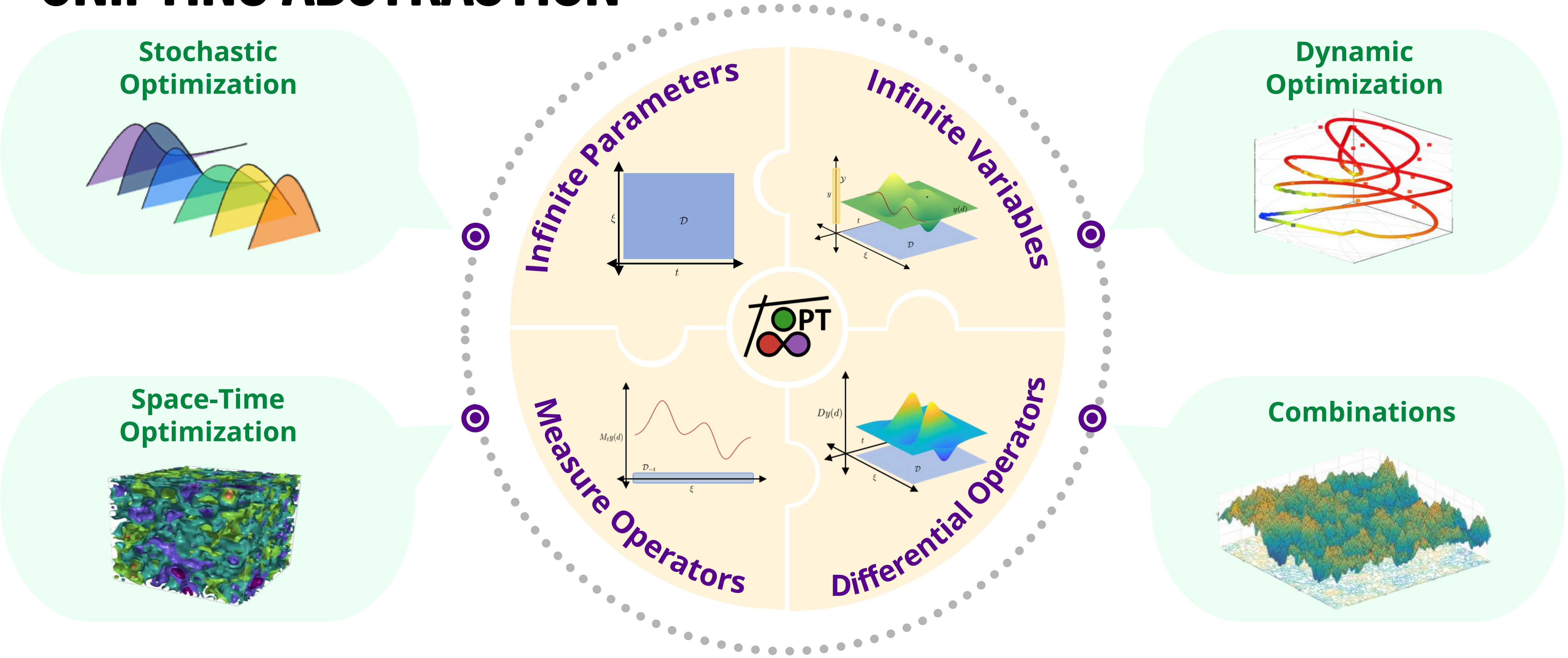
$M_d y(d)$

- **Example:** Disease Control
  - Summarize overall infections

$$\int_{t\in\mathcal{D}_t} \mathbb{E}_\xi[y_i(t,\xi)]dt \qquad \mathbb{E}_\xi\left[\int_{t\in\mathcal{D}_t} y_i(t,\xi)dt\right]$$

J. L. Pulsipher, W. Zhang, T. J. Hongisto, and V. M. Zavala. "A unifying modeling abstraction for infinite-dimensional optimization." 2022

UNIFYING ABSTRACTION



J. L. Pulsipher, W. Zhang, T. J. Hongisto, and V. M. Zavala. "A unifying modeling abstraction for infinite-dimensional optimization." 2022

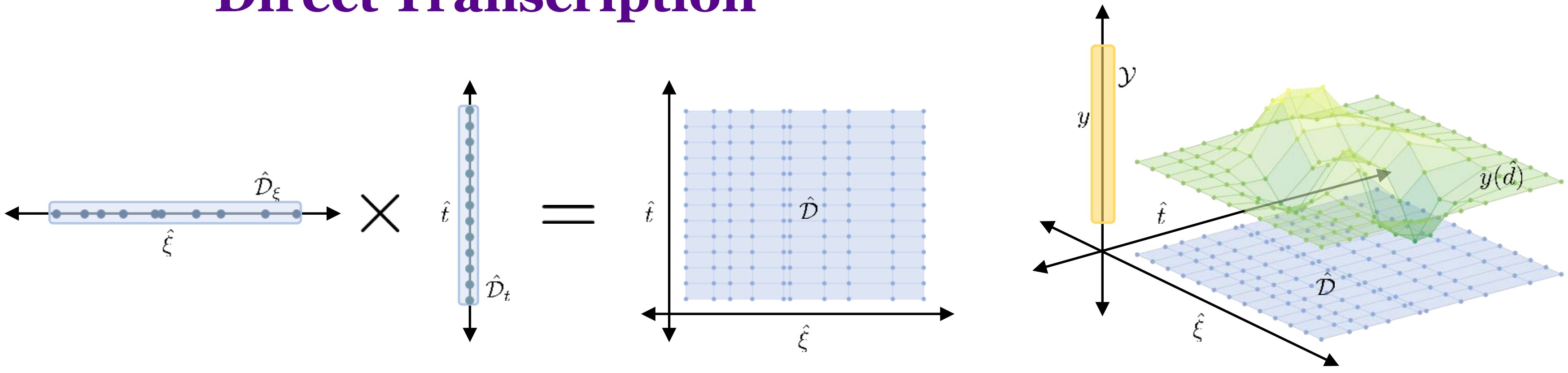
UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

TRANSFORMING INFINITEOPT PROBLEMS INTO FINITE ONES

Direct Transcription

Project onto set of **finite points** $\hat{\mathcal{D}}$

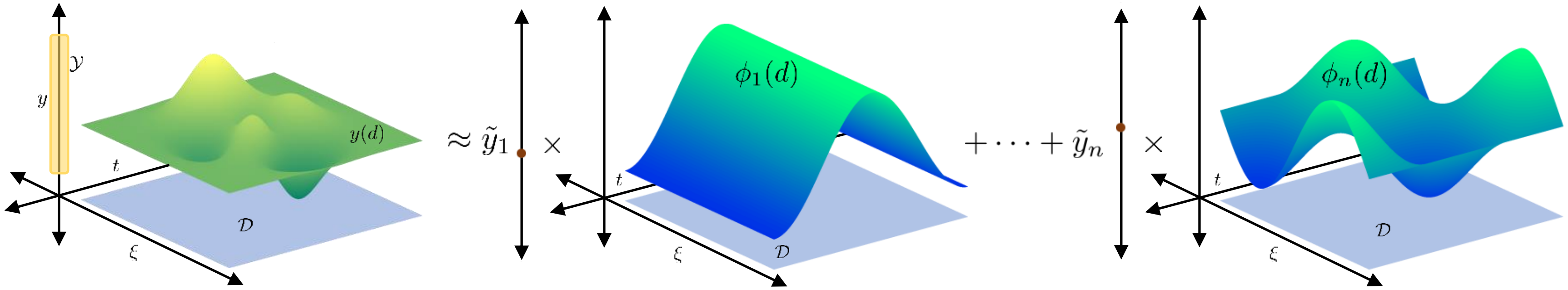
$$\hat{\mathcal{D}} := \prod_{\ell \in \mathcal{L}} \{\hat{d}_{\ell,i} : \hat{d}_{\ell,i} \in \mathcal{D}_\ell, i \in \mathcal{I}_\ell\}$$



Method of Weighted Residuals

Project onto set of known **basis functions**

$$y(d) \approx \sum_{i \in \mathcal{I}} \tilde{y}_i \phi_i(d)$$



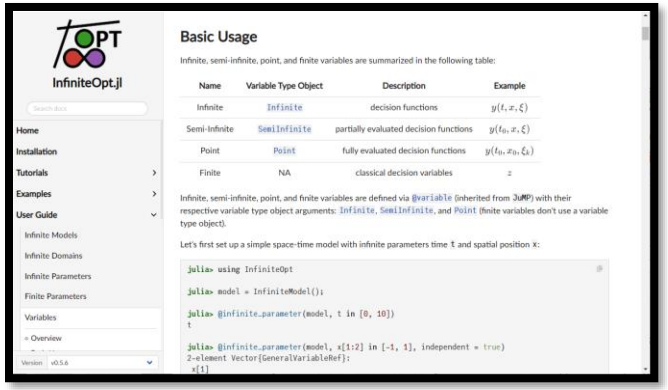
UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

J. L. Pulsipher, W. Zhang, T. J. Hongisto, and V. M. Zavala. "A *Unifying Modeling Abstraction for Infinite-Dimensional Optimization*." Computers & Chem. Eng. 2022

InfiniteOpt

Why is it Different?

- Implements **unifying abstraction**
  - Models a wide range of problems
  - Leverages structure to **accelerate solutions**
- Implemented in julia
  - Enables **intuitive** symbolic expressions
  - Highly **performant**
- **Extensive resources**
  - Documentation, tutorials, examples, forum, short courses, videos



InfiniteOpt.jl

Home
Installation
Tutorials
Examples
User Guide
Infinite Models
Infinite Domains
Infinite Parameters
Finite Parameters
Variables
Overview
Version v0.5.6

Basic Usage

Infinite, semi-infinite, point, and finite variables are summarized in the following table:

Name	Variable Type Object	Description	Example
Infinite	`Infinite`	decision functions	$y(t, x, \xi)$
Semi-Infinite	`SemiInfinite`	partially evaluated decision functions	$y(t_0, x, \xi)$
Point	`Point`	fully evaluated decision functions	$y(t_0, x_0, \xi_k)$
Finite	NA	classical decision variables	z

Infinite, semi-infinite, point, and finite variables are defined via `@variable` (inherited from `JuMP`) with their respective variable type object arguments: `Infinite`, `SemiInfinite`, and `Point` (finite variables don't use a variable type object).

Let's first set up a simple space-time model with infinite parameters time `t` and spatial position `x`:

```
julia> using InfiniteOpt

julia> model = InfiniteModel();

julia> @infinite_parameter(model, t in [0, 10])
t

julia> @infinite_parameter(model, x[1:2] in [-1, 1], independent = true)
2-element Vector{GeneralVariableRef}:
 x[1]
```

Intuitive Modeling API

$$\frac{\partial y_b(t,\xi)}{\partial t} = 2y_b(t,\xi)^2 + y_a(t) - z_1$$

$$\mathbb{E}_\xi\left[y_c(t,\xi)\right] \geq \alpha$$

$$y_a(0) + z_2 = \beta$$

```
@constraint(m, ∂(yb, t) == 2yb^2 + ya - z[1])
@constraint(m, 𝔼(yc, ξ) ≥ α)
@constraint(m, ya(0) + z[2] == β)
```

Impact

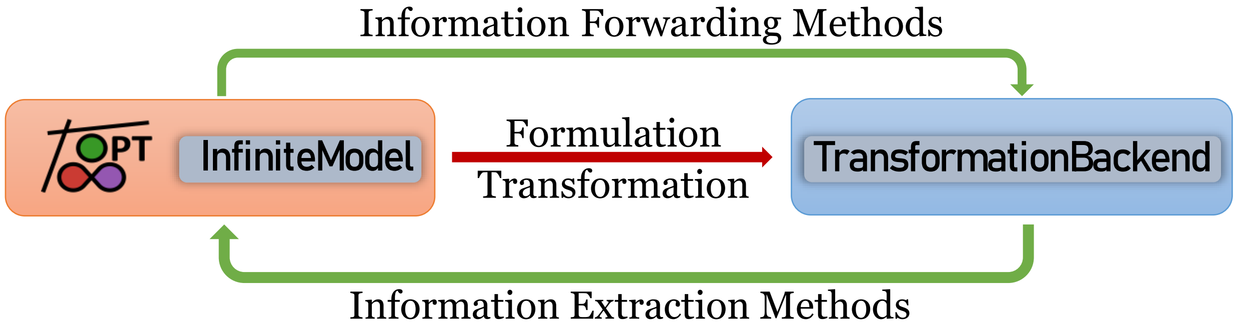
- 1000s of downloads
- Use cases in **diverse disciplines**
  - e.g., evolutionary biology, rocketry, economics, autonomous vehicles

UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

Try it @ https://github.com/infiniteopt/InfiniteOpt.jl

TRANSFORMING INFINITEOPT MODELS

Transformation Paradigm



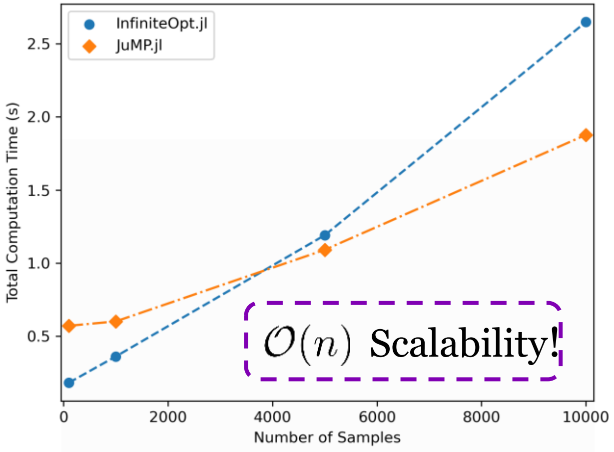
Transformation API

- Highly extensible to **make advanced solution techniques accessible/automated**
- Detailed templates, tutorials, and docs

Automated Transcription

(via TranscriptionOpt)

- Many **derivative/measure approximations**
  - Orthogonal collocation, Gauss quadrature, etc.
- **Performant**



InfiniteOpt.jl
↓
Automatic Transcription

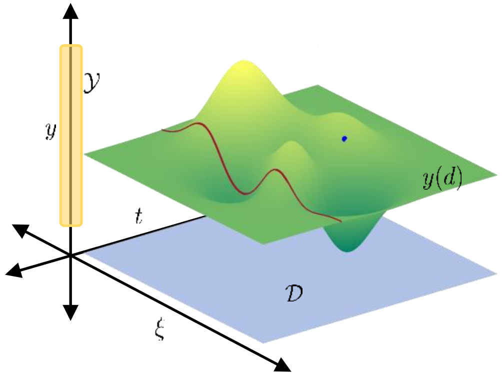
JuMP.jl
↓
Manual Transcription

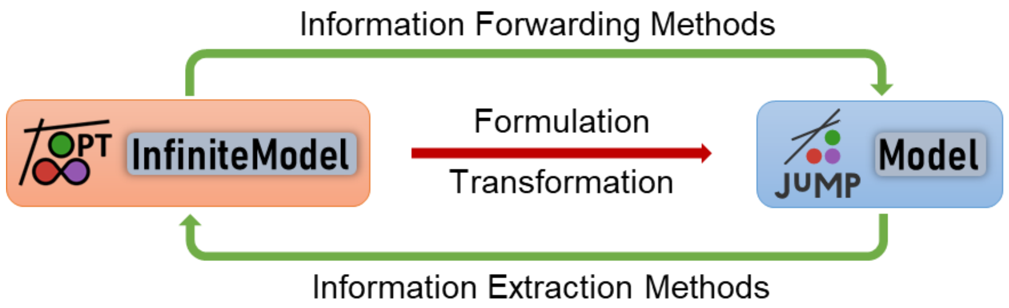
SOLVING INFINITEOPT PROBLEMS VIA TRANSCRIPTIONOPT

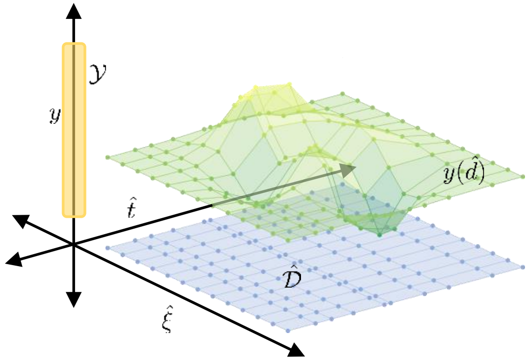
- Apply **transformation** to obtain finite JuMP model that can be solved
- InfiniteOpt has a large suite of **discretization** techniques
- Discretized InfiniteOpt problems have **repeated structure**
- Traditional modeling languages like JuMP do not leverage repeated structure

$$\sin^2(y(t)) \leq 42, \; t \in \mathcal{D}_t$$

$$\sin^2(y_k) \leq 42, \; k \in \mathcal{K}$$







How can we leverage the repeated structure to **accelerate solution performance**?

OUTLINE

- InfiniteOpt
- **ExaModels**
- InfiniteExaModels

Traditional Nonlinear Optimization: Software

Problem Formulation

$$\min_{x \geq 0} f(x)$$
$$\text{s.t. } c(x) = 0$$

Newton's Step Computation

$$\underbrace{\begin{bmatrix} W + \Sigma + \delta_w I & A^\top \\ A & -\delta_c I \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} p^x \\ p^\lambda \end{bmatrix}}_{\text{“KKT System”}}$$

Line Search

$$x^{(k+1)} = x^{(k)} + \alpha \Delta x$$
$$\lambda^{(k+1)} = \lambda^{(k)} + \alpha \Delta \lambda$$

Algebraic Modeling Systems
e.g., AMPL, JuMP, CasADi

Nonlinear Optimization Solvers
e.g., Ipopt, Knitro

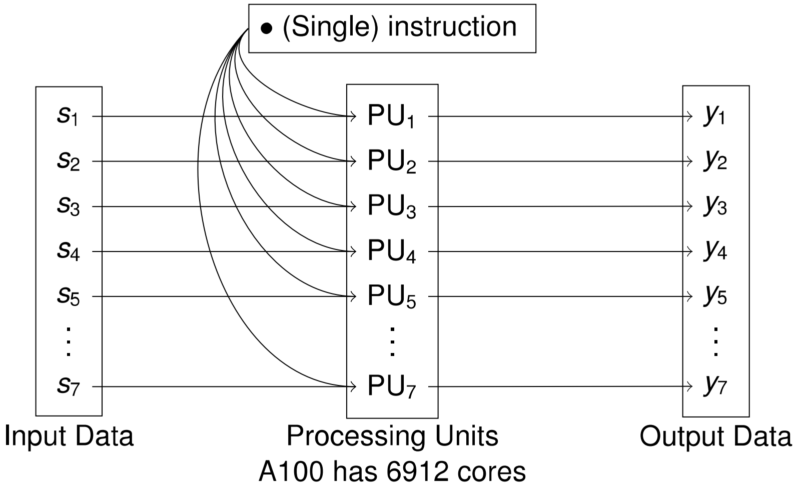
Sparse Linear Solvers
e.g., Ma27, Pardiso

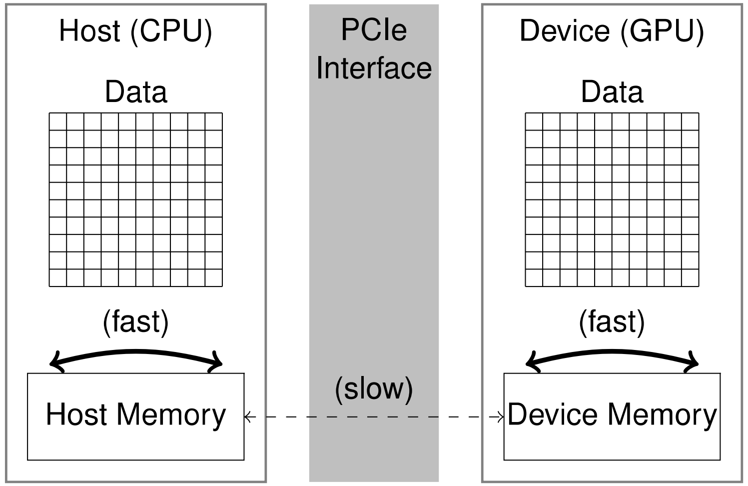
- **Algebraic modeling systems** provide **front-end** and **sparse derivative evaluation** capabilities
- **Nonlinear optimization solvers** apply **optimization algorithms**
- **Sparse linear solvers** resolve **KKT systems** using **sparse matrix factorization**
- Many of these tools are developed in the 1980s-2000s (not compatible with GPUs).

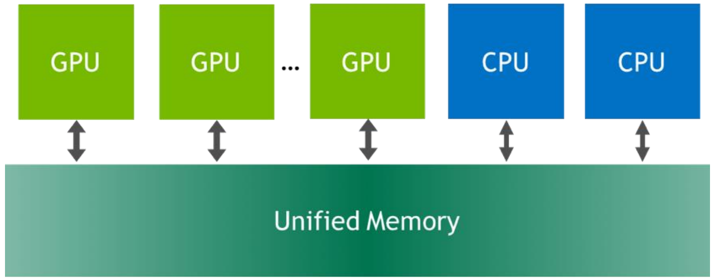
Sungho Shin sshin@anl.gov

How Does GPU Work?

- Single Instruction, Multiple Data **(SIMD) parallelism**
- **Dedicated device memory and slow interface:** all data should reside in device memory only
- **Emerging architectures** employ **unified memory.**

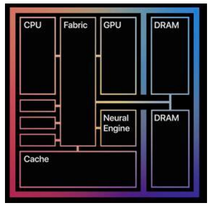






NVIDIA Grace Hopper

Apple M-series chips



Adapting CPU code to GPU code is not merely a matter of software engineering; it often requires the **redesign of the algorithm**

Sungho Shin sshin@anl.gov

SIMD Abstraction for NLPs

ExaModels

- **Large-scale optimization problems almost always have repeated patterns**
- **SIMD Abstraction** can capture such repeated patterns:

"single instruction"

$$\min_{x^\flat \le x \le x^\sharp} \underbrace{\sum_{l\in[L]}}_{\text{a small number of different patterns}} \sum_{i\in[I_l]} f^{(l)}(x; p_i^{(l)})$$

$$\text{s.t.} \sum_{n\in[N]} \sum_{k\in[K_n]} h^{(n)}(x; s_k^{(n)}) = 0$$

"over multiple data"

- Repeated patterns are inputted as **iterators** (data can be stored in structured format)

```
constraint(c, 3 * x[i+1]^3 + 2 * sin(x[i+2]) for i = 1:N-2)
```

"Instruction" "Data"

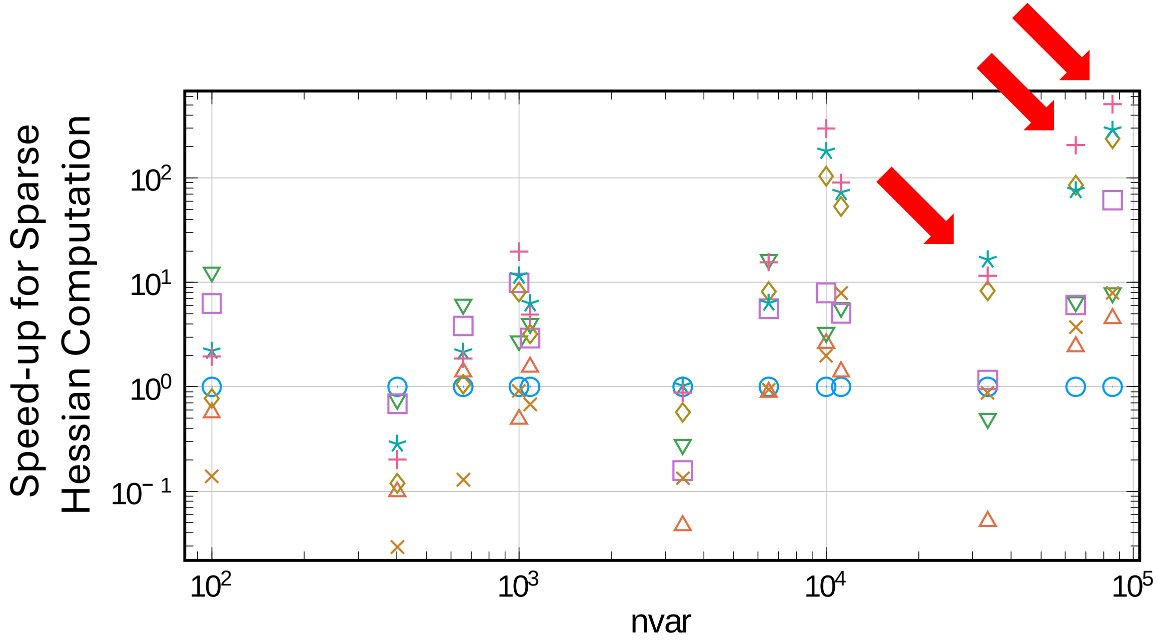
- **For each pattern**, the AD kernel is **compiled** and **executed over multiple data** in parallel

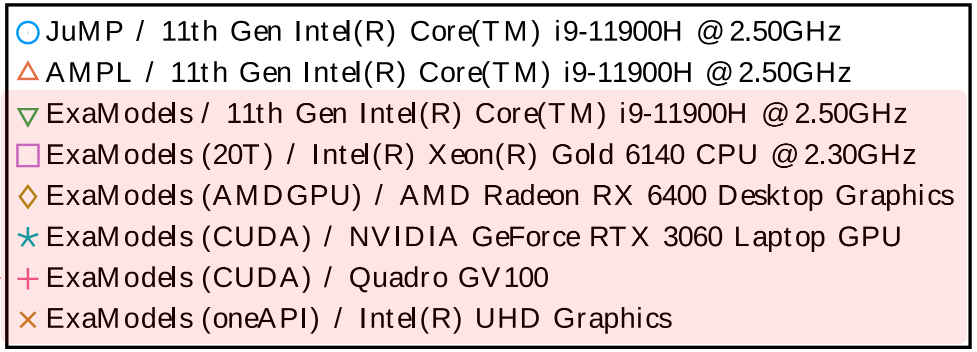
Shin, Pacaud, and Anitescu
Accelerating optimal power flow with GPUs: SIMD abstraction of nonlinear programs and condensed-space interior-point methods.
PSCC 2024

Sungho Shin sshin@anl.gov

Sparse AD Benchmark

ExaModels





*Benchmark problems: AC OPF, Luksan-Vlcek problem, quadrotor control, distillation column control

- For the largest case, **ExaModels on GPU** is **100× faster** than the state-of-the-art tools on CPUs
- ExaModels runs on **all major GPU architectures** and **single/multi-threaded CPUs**

Sparse AD with **SIMD abstraction** enables **efficient derivative computations on GPUs**

Nonlinear Optimization Framework on GPUs

Problem Formulation

$$\min_{x \geq 0} f(x)$$
$$\text{s.t. } c(x) = 0$$

Newton's Step Computation

$$\underbrace{\begin{bmatrix} W + \Sigma + \delta_w I & A^\top \\ A & -\delta_c I \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} p^x \\ p^\lambda \end{bmatrix}}_{\text{"KKT System" (ill-conditioned)}}$$

Line Search

$$x^{(k+1)} = x^{(k)} + \alpha \Delta x$$
$$\lambda^{(k+1)} = \lambda^{(k)} + \alpha \Delta \lambda$$

Algebraic Modeling Systems

Nonlinear Optimization Solvers

Sparse Linear Solvers

ExaModels

- Parallel AD with SIMD abstraction
- Runs on GPU architectures

MadNLP

- Lifted & Hybrid KKT System
- Runs on NVIDIA GPUs

NVIDIA CUDA

- Parallel Cholesky factorization
- Runs on NVIDIA GPUs

https://github.com/exanauts/ExaModels.jl
https://github.com/MadNLP/MadNLP.jl
https://docs.nvidia.com/cuda/cudss

Sungho Shin sshin@anl.gov

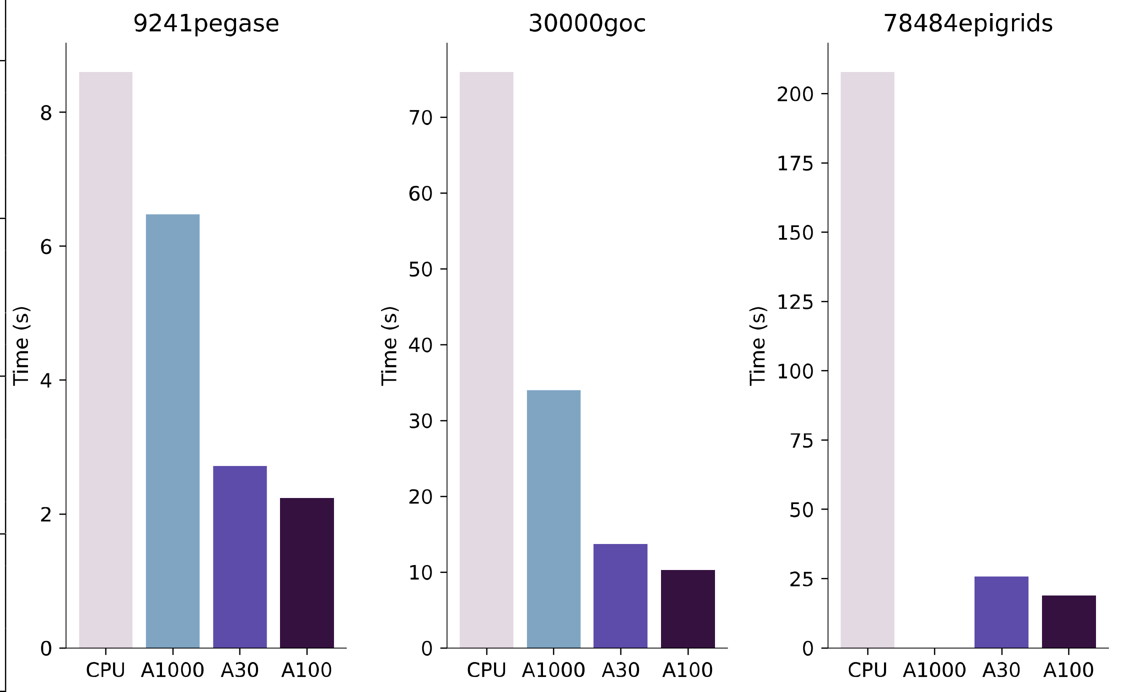
AC Optimal Power Flow

ExaModels + MadNLP + NVIDIA CUDA

	CPU				Lifted KKT on GPU				Hybrid KKT on GPU			
Case	it	init	lin	total	it	init	lin	total	it	init	lin	total
89_pegase	32	0.00	0.02	**0.03**	29	0.03	0.12	**0.24**	32	0.03	0.07	**0.22**
179_goc	45	0.00	0.03	**0.05**	39	0.03	0.19	**0.35**	45	0.03	0.07	**0.25**
500_goc	39	0.01	0.10	**0.14**	39	0.05	0.09	**0.26**	39	0.05	0.07	**0.27**
793_goc	35	0.01	0.12	**0.18**	57	0.06	0.27	**0.52**	35	0.05	0.10	**0.30**
1354_pegase	49	0.02	0.35	**0.52**	96	0.12	0.69	**1.22**	49	0.12	0.17	**0.50**
2000_goc	42	0.03	0.66	**0.93**	46	0.15	0.30	**0.66**	42	0.16	0.14	**0.50**
2312_goc	43	0.02	0.59	**0.82**	45	0.14	0.32	**0.68**	43	0.14	0.21	**0.56**
2742_goc	125	0.04	3.76	**7.31**	157	0.20	1.93	**15.49**	-	-	-	**-**
2869_pegase	55	0.04	1.09	**1.52**	57	0.20	0.30	**0.80**	55	0.21	0.26	**0.73**
3022_goc	55	0.03	0.98	**1.39**	48	0.18	0.23	**0.66**	55	0.18	0.23	**0.68**
3970_goc	48	0.05	1.95	**2.53**	47	0.26	0.37	**0.87**	48	0.27	0.24	**0.80**
4020_goc	59	0.06	3.90	**4.60**	123	0.28	1.75	**3.15**	59	0.29	0.41	**1.08**
4601_goc	71	0.09	3.09	**4.16**	67	0.27	0.51	**1.17**	71	0.28	0.39	**1.12**
4619_goc	49	0.07	3.21	**3.91**	49	0.34	0.59	**1.25**	49	0.33	0.31	**0.95**
4837_goc	59	0.08	2.49	**3.33**	59	0.29	0.58	**1.31**	59	0.29	0.35	**0.98**
4917_goc	63	0.07	1.97	**2.72**	55	0.26	0.55	**1.18**	63	0.26	0.34	**0.94**
5658_epigrids	51	0.31	2.80	**3.86**	58	0.35	0.66	**1.51**	51	0.35	0.35	**1.03**
7336_epigrids	50	0.13	3.60	**4.91**	56	0.45	0.95	**1.89**	50	0.43	0.35	**1.13**
8387_pegase	74	0.14	5.31	**7.62**	82	0.59	0.79	**2.30**	75	0.58	7.66	**8.84**
9241_pegase	74	0.15	6.11	**8.60**	101	0.63	0.88	**2.76**	71	0.63	0.99	**2.24**
9591_goc	67	0.20	11.14	**13.37**	98	0.63	2.67	**4.58**	67	0.62	0.74	**1.96**
10000_goc	82	0.15	6.00	**8.16**	64	0.49	0.81	**1.83**	82	0.49	0.75	**1.82**
10192_epigrids	54	0.41	7.79	**10.08**	57	0.67	1.14	**2.40**	54	0.67	0.66	**1.81**
10480_goc	71	0.24	12.04	**14.74**	67	0.75	0.99	**2.72**	71	0.74	1.09	**2.50**
13659_pegase	63	0.45	7.21	**10.14**	75	0.83	1.05	**2.96**	62	0.84	0.93	**2.47**
19402_goc	69	0.63	31.71	**36.92**	73	1.42	2.28	**5.38**	69	1.44	1.93	**4.31**
20758_epigrids	51	0.63	14.27	**18.21**	53	1.34	1.05	**3.57**	51	1.35	1.55	**3.51**
30000_goc	183	0.65	63.02	**75.95**	-	-	-	**-**	225	1.22	5.59	**10.27**
78484_epigrids	102	2.57	179.29	**207.79**	101	5.94	5.62	**18.03**	104	6.29	9.01	**18.90**

Table 3 OPF benchmark, solved with a tolerance `tol=1e-6`. (A100 GPU)

Optimizing entire eastern interconnection



- For large-scale cases (> 20k vars), GPU becomes **significantly faster than CPU** (up to ×10)
- **Reliable convergence for** tol=10^{-6}, but still less reliable than CPUs

Pacaud, Shin, Montoison, Schanen, and Anitescu. *Approaches to nonlinear programming on GPU architectures*. In preparation.

Distillation Column

ExaModels + MadNLP + NVIDIA CUDA

#time steps	CPU init (s)	CPU it	CPU solve (s)	Lifted KKT on GPU init (s)	Lifted KKT on GPU it	Lifted KKT on GPU solve (s)	Hybrid KKT on GPU init (s)	Hybrid KKT on GPU it	Hybrid KKT on GPU solve (s)
100	0.1	7	0.1	0.1	11	0.1	0.1	7	0.0
500	0.1	7	0.5	0.2	12	0.1	0.2	7	0.1
1,000	0.1	7	1.5	0.4	12	0.2	0.4	7	0.1
5,000	0.6	7	8.2	2.3	13	0.5	2.3	7	0.4
10,000	1.3	7	18.7	5.2	13	0.9	5.3	7	0.7
20,000	4.3	7	38.2	10.7	14	2.1	11.3	7	1.4
50,000	15.9	7	98.8	30.4	14	5.5	31.2	7	3.8

- "**Symbolic analysis**" is often the bottleneck on GPUs, but this can be computed "**off-line**" thus, **online computation performance** can be **even greater**
- The distillation column control problem can be solved more than 20x faster

ExaModels, **MadNLP**, and **CUDSS** provide **efficient and reliable** solution framework for large-scale nonlinear optimization problems

Pacaud and Shin. *GPU-accelerated nonlinear model predictive control with ExaModels and MadNLP*. arXiv:2403:15913

Sungho Shin sshin@anl.gov

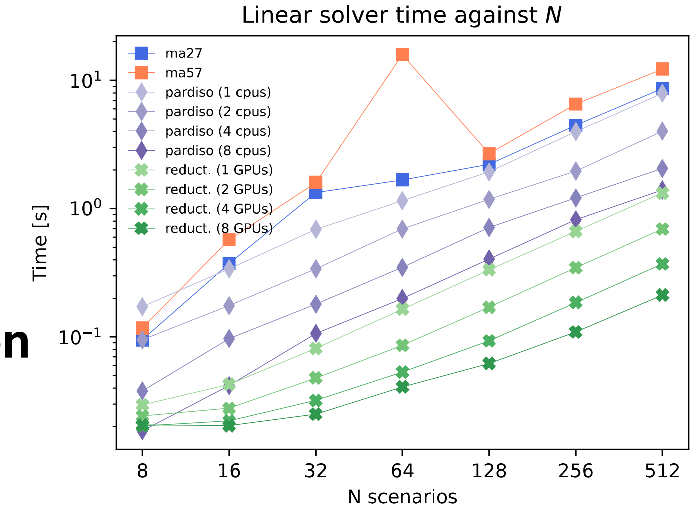
Remaining Challenges

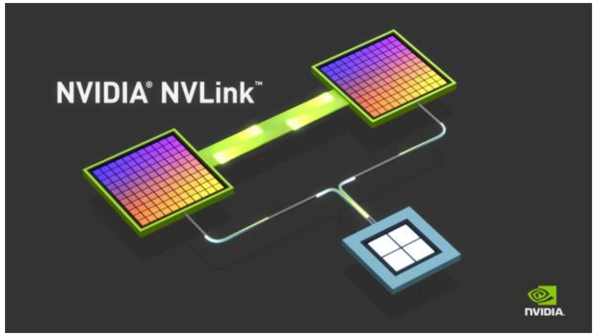
- **Portable sparse Cholesky factorization**

		CPU (single)	CPU (multi)	NVIDIA GPU	AMD GPU	Intel GPU
Algebraic Modeling Platforms	**AMPL**	✓	✗	✗	✗	✗
	JuMP	✓	✗	✗	✗	✗
	ExaModels	✓	✓	✓	✓	✓
NLP Solvers	**Ipopt**	✓	✗	✗	✗	✗
	MadNLP	✓	✗	✓	✗	✗

  - Currently, we are relying on a **proprietary** Cholesky solver (CUDSS)
  - An open-source, **portable Cholesky solver is needed** to run on Exascale

- **Multi-GPU optimization tools**
  - A **single GPU is sometimes limited** in computation & storage capacity
  - Our recent results suggest that there are significant opportunities in **multi-GPU utilization**





Pacaud et. al. *Parallel interior-point solver for block-structured nonlinear programs on SIMD/GPU architectures*, OMS (2024).

Sungho Shin sshin@anl.gov

EXAMODELSMOI.JL

- Provides an **MOI optimizer** for JuMP models
  - Can use either `ExaModels.IpoptOptimizer` or `ExaModels.MadNLPOptimizer`

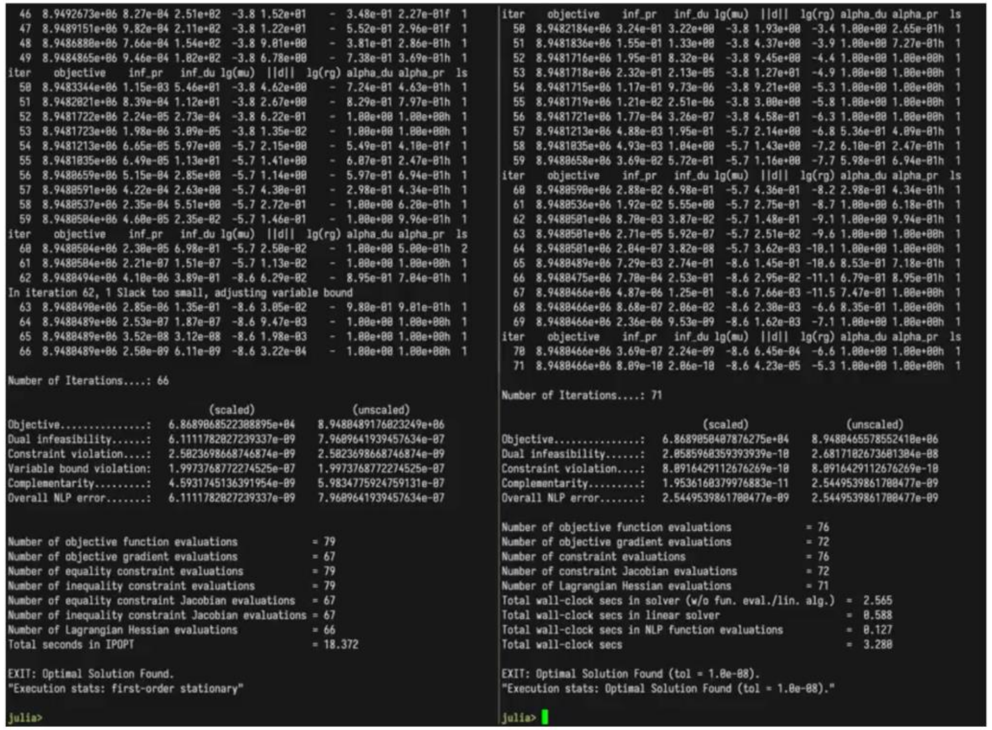
```julia
using ExaModels, JuMP, CUDA, MadNLPGPU

model = Model(() -> ExaModels.MadNLPOptimizer(CUDABackend()))
```

- Searches for repeated algebraic structure via a **bin search**

- Doesn’t necessary yield the most efficient ExaModel structure

ACCELERATING NLP PERFORMANCE ON CPUS AND GPUS

- ExaModels + MadNLP is highly performant for problems with repeated patterns
- Translating InfiniteOpt problems to SIMD is nontrivial
- TranscriptionOpt + ExaModelsMOI has to ignore structure while building the model



```
  46  8.9492673e+06 8.27e-04 2.51e+02  -3.8 1.52e+01    -  3.48e-01 2.27e-01f  1
  47  8.9489151e+06 9.82e-04 2.11e+02  -3.8 1.22e+01    -  5.52e-01 2.96e-01f  1
  48  8.9486880e+06 7.66e-04 1.54e+02  -3.8 9.01e+00    -  3.81e-01 2.86e-01h  1
  49  8.9484865e+06 9.46e-04 1.82e+02  -3.8 6.78e+00    -  7.38e-01 3.69e-01h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  50  8.9483344e+06 1.15e-03 5.46e+01  -3.8 4.62e+00    -  7.24e-01 4.63e-01h  1
  51  8.9482021e+06 8.39e-04 1.12e+01  -3.8 2.67e+00    -  8.29e-01 7.97e-01h  1
  52  8.9481722e+06 2.24e-05 2.73e-04  -3.8 6.22e-01    -  1.00e+00 1.00e+00h  1
  53  8.9481723e+06 1.98e-06 3.09e-05  -3.8 1.35e-02    -  1.00e+00 1.00e+00h  1
  54  8.9481213e+06 6.65e-05 5.97e+00  -5.7 2.15e+00    -  5.49e-01 4.10e-01f  1
  55  8.9481035e+06 6.49e-05 1.13e+01  -5.7 1.41e+00    -  6.07e-01 2.47e-01h  1
  56  8.9480659e+06 5.15e-04 2.85e+00  -5.7 1.14e+00    -  5.97e-01 6.94e-01h  1
  57  8.9480591e+06 4.22e-04 2.63e+00  -5.7 4.30e-01    -  2.98e-01 4.34e-01h  1
  58  8.9480537e+06 2.35e-04 5.51e+00  -5.7 2.72e-01    -  1.00e+00 6.20e-01h  1
  59  8.9480504e+06 4.60e-05 2.35e-02  -5.7 1.46e-01    -  1.00e+00 9.96e-01h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  60  8.9480504e+06 2.30e-05 6.98e-01  -5.7 2.50e-02    -  1.00e+00 5.00e-01h  2
  61  8.9480504e+06 2.21e-07 1.51e-07  -5.7 1.13e-02    -  1.00e+00 1.00e+00h  1
  62  8.9480494e+06 4.10e-06 3.89e-01  -8.6 6.29e-02    -  8.95e-01 7.04e-01h  1
In iteration 62, 1 Slack too small, adjusting variable bound
  63  8.9480490e+06 2.85e-06 1.35e-01  -8.6 3.05e-02    -  9.80e-01 9.01e-01h  1
  64  8.9480489e+06 2.53e-07 1.87e-07  -8.6 9.47e-03    -  1.00e+00 1.00e+00h  1
  65  8.9480489e+06 3.52e-08 3.12e-08  -8.6 1.98e-03    -  1.00e+00 1.00e+00h  1
  66  8.9480489e+06 2.50e-09 6.11e-09  -8.6 3.22e-04    -  1.00e+00 1.00e+00h  1

Number of Iterations....: 66

                                   (scaled)                 (unscaled)
Objective...............:   6.8689068522308895e+04    8.9480489176023249e+06
Dual infeasibility......:   6.1111782827239337e-09    7.9609641939457634e-07
Constraint violation....:   2.5023698668746874e-09    2.5023698668746874e-09
Variable bound violation:   1.9973768772274525e-07    1.9973768772274525e-07
Complementarity.........:   4.5931745136391954e-09    5.9834775924759131e-07
Overall NLP error.......:   6.1111782827239337e-09    7.9609641939457634e-07


Number of objective function evaluations             = 79
Number of objective gradient evaluations             = 67
Number of equality constraint evaluations            = 79
Number of inequality constraint evaluations          = 79
Number of equality constraint Jacobian evaluations   = 67
Number of inequality constraint Jacobian evaluations = 67
Number of Lagrangian Hessian evaluations             = 66
Total seconds in IPOPT                               = 18.372

EXIT: Optimal Solution Found.
"Execution stats: first-order stationary"

julia>
```

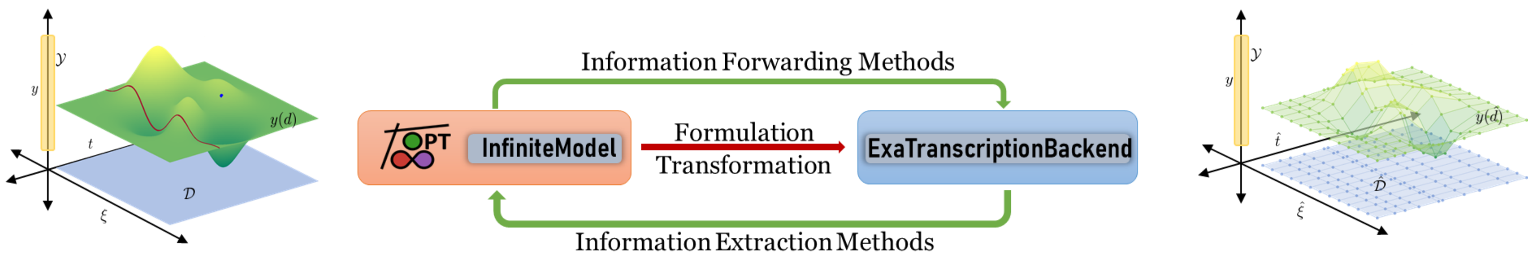
```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  50  8.9482184e+06 3.24e-01 3.22e+00  -3.8 1.93e+00  -3.4 1.00e+00 2.65e-01h  1
  51  8.9481836e+06 1.55e-01 1.33e+00  -3.8 4.37e+00  -3.9 1.00e+00 7.27e-01h  1
  52  8.9481716e+06 1.95e-01 8.32e-04  -3.8 9.45e+00  -4.4 1.00e+00 1.00e+00h  1
  53  8.9481718e+06 2.32e-01 2.13e-05  -3.8 1.27e+01  -4.9 1.00e+00 1.00e+00h  1
  54  8.9481715e+06 1.17e-01 9.73e-06  -3.8 9.21e+00  -5.3 1.00e+00 1.00e+00h  1
  55  8.9481719e+06 1.21e-02 2.51e-06  -3.8 3.00e+00  -5.8 1.00e+00 1.00e+00h  1
  56  8.9481721e+06 1.77e-04 3.26e-07  -3.8 4.58e-01  -6.3 1.00e+00 1.00e+00h  1
  57  8.9481213e+06 4.88e-03 1.95e-01  -5.7 2.14e+00  -6.8 5.36e-01 4.09e-01h  1
  58  8.9481035e+06 4.93e-03 1.04e+00  -5.7 1.43e+00  -7.2 6.10e-01 2.47e-01h  1
  59  8.9480658e+06 3.69e-02 5.72e-01  -5.7 1.16e+00  -7.7 5.98e-01 6.94e-01h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  60  8.9480590e+06 2.88e-02 6.98e-01  -5.7 4.36e-01  -8.2 2.98e-01 4.34e-01h  1
  61  8.9480536e+06 1.92e-02 5.55e+00  -5.7 2.75e-01  -8.7 1.00e+00 6.18e-01h  1
  62  8.9480501e+06 8.70e-03 3.87e-02  -5.7 1.48e-01  -9.1 1.00e+00 9.94e-01h  1
  63  8.9480501e+06 2.71e-05 5.92e-07  -5.7 2.51e-02  -9.6 1.00e+00 1.00e+00h  1
  64  8.9480501e+06 2.04e-07 3.82e-08  -5.7 3.62e-03 -10.1 1.00e+00 1.00e+00h  1
  65  8.9480489e+06 7.29e-03 2.74e-01  -8.6 1.45e-01 -10.6 8.53e-01 7.18e-01h  1
  66  8.9480475e+06 7.70e-04 2.53e-01  -8.6 2.95e-02 -11.1 6.79e-01 8.95e-01h  1
  67  8.9480466e+06 4.87e-06 1.25e-01  -8.6 7.66e-03 -11.5 7.47e-01 1.00e+00h  1
  68  8.9480466e+06 8.68e-07 2.06e-02  -8.6 2.30e-03  -6.6 8.35e-01 1.00e+00h  1
  69  8.9480466e+06 2.36e-06 9.53e-09  -8.6 1.62e-03  -7.1 1.00e+00 1.00e+00h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  70  8.9480466e+06 3.69e-07 2.24e-09  -8.6 6.45e-04  -6.6 1.00e+00 1.00e+00h  1
  71  8.9480466e+06 8.09e-10 2.06e-10  -8.6 4.23e-05  -5.3 1.00e+00 1.00e+00h  1

Number of Iterations....: 71

                                   (scaled)                 (unscaled)
Objective...............:   6.8689050407876275e+04    8.9480465578552410e+06
Dual infeasibility......:   2.0585960359393939e-10    2.6817102673601304e-08
Constraint violation....:   8.0916429112676269e-10    8.0916429112676269e-10
Complementarity.........:   1.9536160379976883e-11    2.5449539861700477e-09
Overall NLP error.......:   2.5449539861700477e-09    2.5449539861700477e-09

Number of objective function evaluations             = 76
Number of objective gradient evaluations             = 72
Number of constraint evaluations                     = 76
Number of constraint Jacobian evaluations            = 72
Number of Lagrangian Hessian evaluations             = 71
Total wall-clock secs in solver (w/o fun. eval./lin. alg.)  =  2.565
Total wall-clock secs in linear solver                      =  0.588
Total wall-clock secs in NLP function evaluations           =  0.127
Total wall-clock secs                                       =  3.280

EXIT: Optimal Solution Found (tol = 1.0e-08).
"Execution stats: Optimal Solution Found (tol = 1.0e-08)."

julia>
```

S. Shin, F. Pacaud, and M. Anitescu. "Accelerating optimal power flow with gpus: Simd abstraction of nonlinear programs and condensed-space interior-point methods." 2023

OUTLINE

- InfiniteOpt

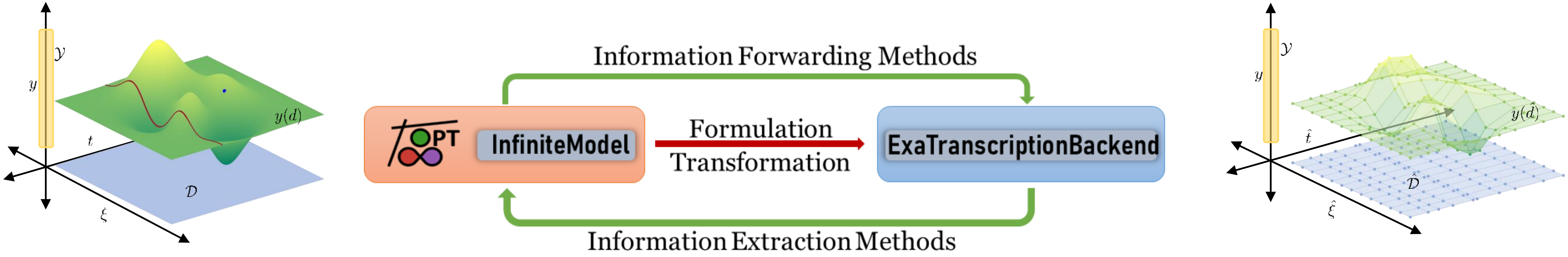
- ExaModels



- **InfiniteExaModels**

INFINITEEXAMODELS.JL

- Bridges the gap between InfiniteOpt & ExaModels
- **Automates transcription** via established transformation interface
- Leverages repeated structure to **drastically reduce model creation time**
  - More efficient than manual transcription directly given to ExaModels



J. Pulsipher and S. Shin. "Scalable Modeling of Infinite-Dimensional Nonlinear Programs with InfiniteExaModels.jl." 2023

IMPLEMENTATION DETAILS

- Supports the use of **JSO NLP solvers** (e.g., Ipopt, MadNLP, KNITRO)

- Defined via an `ExaTranscriptionBackend`

```
using InfiniteOpt, InfiniteExaModels, NLPModelsIpopt
model = InfiniteModel(ExaTranscriptionBackend(IpoptSolver))
```

- Rapidly transcribes infinite model into **efficient ExaModels**

- **Model build** time is nearly **independent of the discretization** size

BENCHMARK PROBLEMS

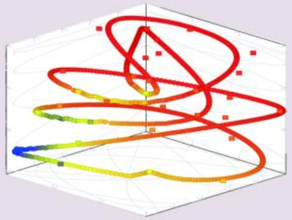
- Compare performance with JuMP, AMPL, ExaModels, and InfiniteExaModels
- Run on CPU with Ipopt and GPU with MadNLP

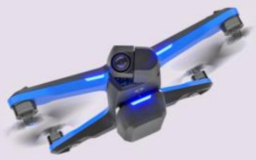
2-Stage Stochastic Program

- Stochastic optimal power flow
- 1,000 to 16,000 random scenarios

Optimal Control

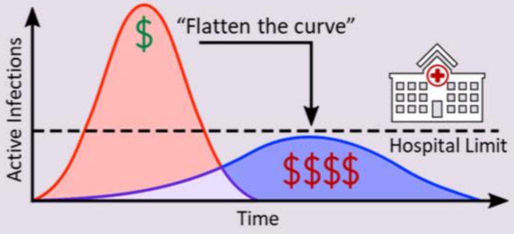
- Model predictive control of quadcopter
- Track trajectory setpoint and vary grid size





Stochastic Optimal Control

- Control isolation policy to combat disease
- Uncertain transmission rate



J. **Pulsipher** and S. Shin. "Scalable Modeling of Infinite-Dimensional Nonlinear Programs with InfiniteExaModels.jl." 2023

NUMERICAL RESULTS (CPU W/ IPOPT)

- AD is **5 – 20 times faster**
- Model build time is **1 – 2 orders-of-magnitude faster**

	Stochastic 2-Stage				Optimal Control				Stochastic Control			
Approach	**Build**	**AD**	**Solve**	**Tot.**	**Build**	**AD**	**Solve**	**Tot.**	**Build**	**AD**	**Solve**	**Tot.**
JuMP.jl	87.1	21.4	63.7	151	143	6.3	28.6	172	10.9	60.7	386	397
AMPL	99.3	11.5	90.3	190	153	5	26.4	179	10.6	23.4	364	375
ExaModelsMOI.jl	86.6	3.05	56.3	143	139	1.1	23.1	162	8.63	3.45	368	376
InfiniteExaModels.jl	1.94	2.03	43	45	9.34	1	22.6	31.9	0.12	3.01	369	369

```julia
using InfiniteOpt, InfiniteExaModels, NLPModelsIpopt
model = InfiniteModel(ExaTranscriptionBackend(IpoptSolver))
```

J. Pulsipher and S. Shin. "Scalable Modeling of Infinite-Dimensional Nonlinear Programs with InfiniteExaModels.jl." 2023

NUMERICAL RESULTS (GPU W/ MADNLP)

- All AD and solve times are up to **~20 faster on GPU**
- InfiniteExaModels.jl builds models **orders-of-magnitude faster** than ExaModels

	Stochastic 2-Stage				Optimal Control				Stochastic Control			
Approach	**Build**	**AD**	**Solve**	**Tot.**	**Build**	**AD**	**Solve**	**Tot.**	**Build**	**AD**	**Solve**	**Tot.**
ExaModelsMOI.jl	83	0.09	3.15	86.1	133	0.1	6.55	140	8.21	0.51	20.4	28.6
InfiniteExaModels.jl	1.8	0.14	3.03	4.82	8.63	0.1	6.44	15.1	0.06	0.74	21	21

(151 on CPU) ***(172 on CPU)*** ***(397 on CPU)***

```julia
using InfiniteOpt, InfiniteExaModels, MadNLPGPU, CUDA
transform_backend = ExaTranscriptionBackend(MadNLPSolver, backend = CUDABackend())
model = InfiniteModel(transform_backend)
```

TRY IT OUT!

InfiniteOpt

InfiniteExaModels

ExaModels

UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

InfiniteExaModels

UNIVERSITY OF WATERLOO

FACULTY OF ENGINEERING

YOU+WATERLOO

Our greatest impact happens together.

InfiniteOpt