Nonlinear Optimization on Graphics Processing Units

Sungho Shin[*,1], **François Pacaud**[2], **Mihai Anitescu**[1,3], and **Exanauts**

[*]sshin@anl.gov

[1]Mathematics and Computer Science Division, Argonne National Laboratory
[2]Centre Automatique et Systèmes, Mines Paris - PSL
[3]Department of Statistics, University of Chicago

2023 AIChE Meeting, Orlando, Florida



Accelerated Computing on GPUs

- Accelerated computing has **driven the success of AI** (e.g., GPT models have 10^{12} pars).

Accelerated Computing on GPUs

- Accelerated computing has **driven the success of AI** (e.g., GPT models have 10^{12} pars).
- Accelerated computing **empowers scientific computing** (e.g., fluid, climate, bioinformatics).

Accelerated Computing on GPUs

- Accelerated computing has **driven the success of AI** (e.g., GPT models have 10^{12} pars).
- Accelerated computing **empowers scientific computing** (e.g., fluid, climate, bioinformatics).
- We're entering **exascale computing era** (10^{18} floating point operations per second).

Aurora Supercomputer @ Argonne (2024)

$=$ 1 million $\times$

iPhone 14 Pro (2023)

Accelerated Computing on GPUs

- Accelerated computing has **driven the success of AI** (e.g., GPT models have 10^{12} pars).
- Accelerated computing **empowers scientific computing** (e.g., fluid, climate, bioinformatics).
- We're entering **exascale computing era** (10^{18} floating point operations per second).

Aurora Supercomputer @ Argonne (2024)

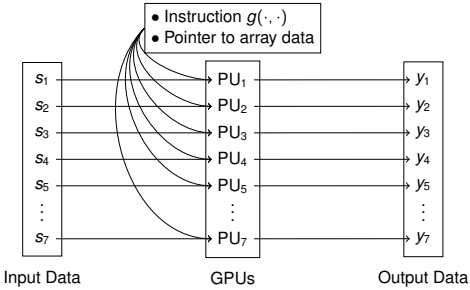
$=$ 1 million $\times$

iPhone 14 Pro (2023)

Can we harness these capabilities in the realm of **classical optimization**?
(e.g., energy systems, optimal control, operations research)

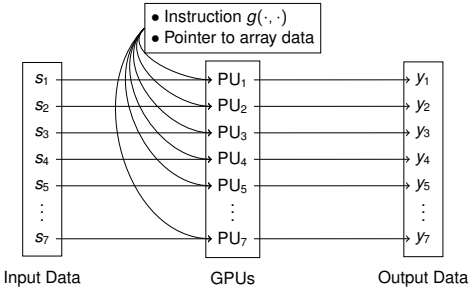
How Do GPUs Work? (or, how are they different from CPUs?)

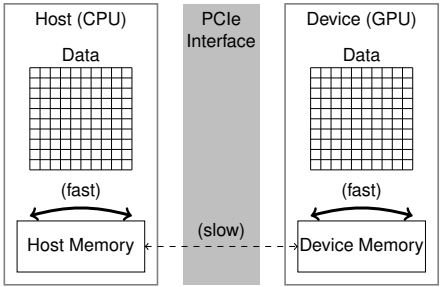
- Single Instruction, Multiple Data (**SIMD**) parallelism,



How Do GPUs Work? (or, how are they different from CPUs?)

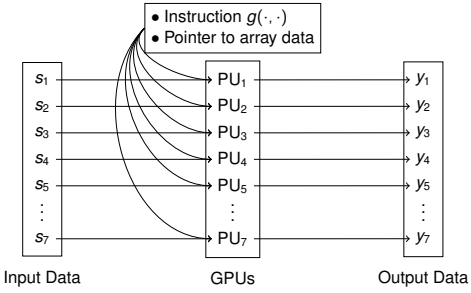
- Single Instruction, Multiple Data (**SIMD**) parallelism, on **dedicated memory**.

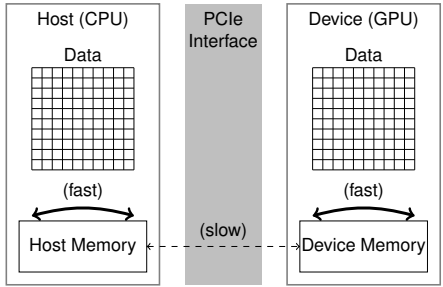




How Do GPUs Work? (or, how are they different from CPUs?)

- Single Instruction, Multiple Data (**SIMD**) parallelism, on **dedicated memory**.

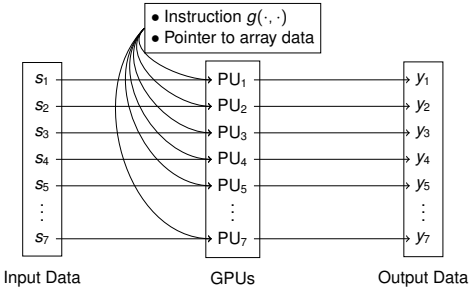


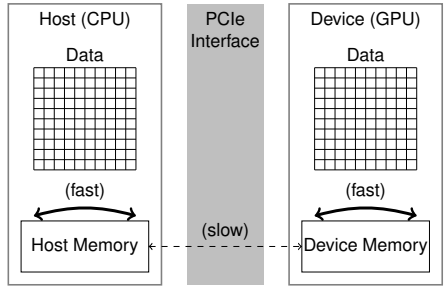


- To achieve optimal performance, all data should **reside exclusively on device memory**.

How Do GPUs Work? (or, how are they different from CPUs?)

- Single Instruction, Multiple Data (**SIMD**) parallelism, on **dedicated memory**.

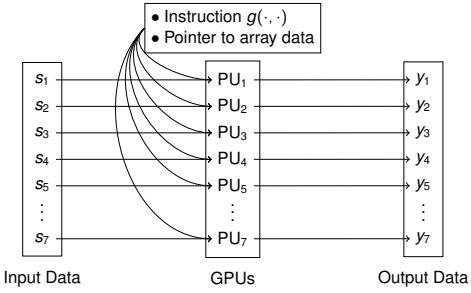


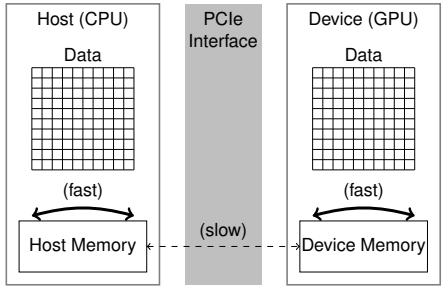


- To achieve optimal performance, all data should **reside exclusively on device memory**.
- Heterogeneous architectures due to **various hardware vendors** (NVIDIA, AMD, and Intel).

How Do GPUs Work? (or, how are they different from CPUs?)

- Single Instruction, Multiple Data (**SIMD**) parallelism, on **dedicated memory**.





- To achieve optimal performance, all data should **reside exclusively on device memory**.
- Heterogeneous architectures due to **various hardware vendors** (NVIDIA, AMD, and Intel).

Adapting CPU algorithms into GPU algorithms is **not merely a matter of software engineering** and often requires a **complete redesign of the algorithm**.

Exascale Computing Project

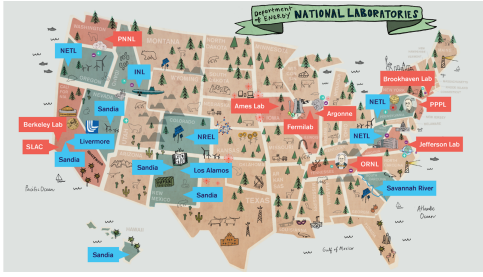
- **Mission**: Harness exascale computing capabilities to tackle **real-world challenges**.

Frontier @Oak Ridge (AMD GPUs)

Aurora @Argonne (Intel GPUs)

Exascale Computing Project

- **Mission**: Harness exascale computing capabilities to tackle **real-world challenges**.



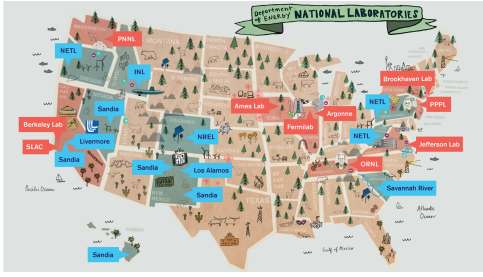
Frontier @Oak Ridge (AMD GPUs)

Aurora @Argonne (Intel GPUs)

- **Challenge**: There were no implementations of **sparse automatic differentiation, nonlinear optimization solver, nor sparse indefinite solvers** on GPUs.

Exascale Computing Project

- **Mission**: Harness exascale computing capabilities to tackle **real-world challenges**.



Frontier @Oak Ridge (AMD GPUs)

Aurora @Argonne (Intel GPUs)

- **Challenge**: There were no implementations of **sparse automatic differentiation, nonlinear optimization solver, nor sparse indefinite solvers** on GPUs.
- **Goal**: Build a **comprehensive software infrastructure** for nonlinear optimization on GPUs.
  - **Performance**: at least an order of magnitude speedup.
  - **Portability**: compatibility with NVIDIA, AMD, and Intel.
  - **Application**: energy infrastructure problems.

Language of Choice: Julia

- Julia resolves the “**two-language problem**”; we’re experiencing 10x faster development.

Language of Choice: Julia

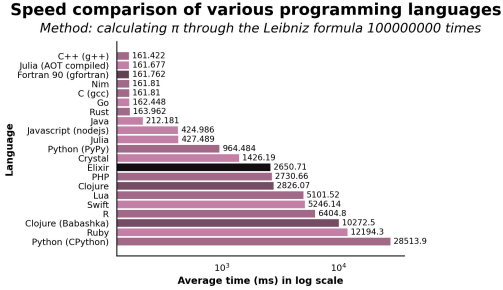
- Julia resolves the "**two-language problem**"; we're experiencing 10x faster development.
- **Multiple dispatch**: High-level abstraction while specializing for specific data types.

Language of Choice: Julia

- Julia resolves the "**two-language problem**"; we're experiencing 10x faster development.
- **Multiple dispatch**: High-level abstraction while specializing for specific data types.
- **Portable programming**: Compatibility across various architectures (NVIDIA, AMD, Intel, etc.)

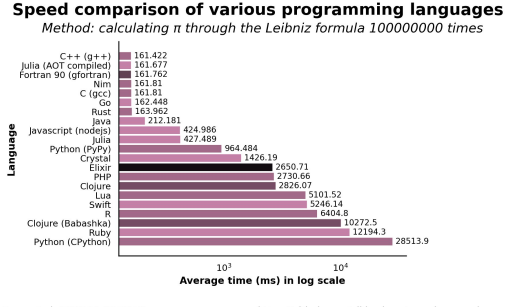
Language of Choice: Julia

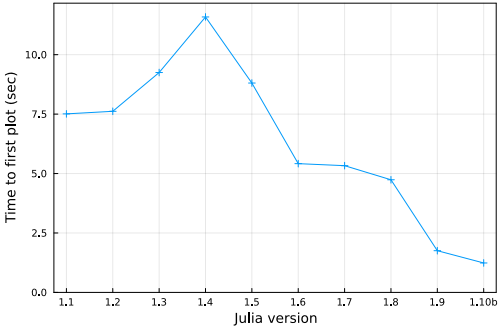
- Julia resolves the "**two-language problem**"; we're experiencing 10x faster development.
- **Multiple dispatch**: High-level abstraction while specializing for specific data types.
- **Portable programming**: Compatibility across various architectures (NVIDIA, AMD, Intel, etc.)
- **JIT compilation**: No performance compromises, running as fast as C/C++/Fortran.



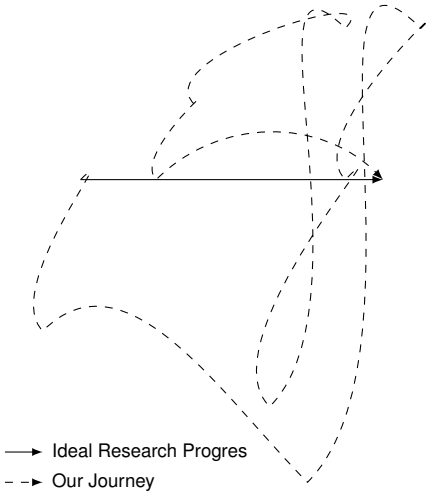
Language of Choice: Julia

- Julia resolves the "**two-language problem**"; we're experiencing 10x faster development.
- **Multiple dispatch**: High-level abstraction while specializing for specific data types.
- **Portable programming**: Compatibility across various architectures (NVIDIA, AMD, Intel, etc.)
- **JIT compilation**: No performance compromises, running as fast as C/C++/Fortran.
- The initial delay, often referred to as "**time-to-first-call**," has been significantly reduced.

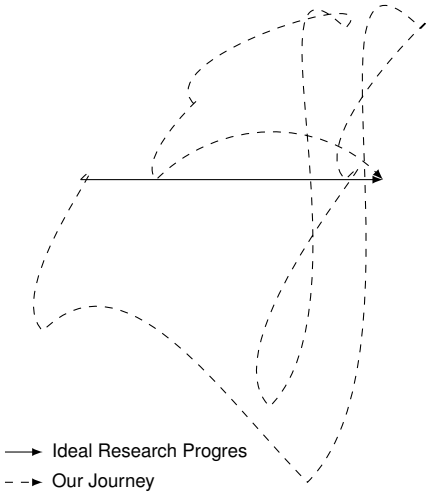




Summary of Exanauts' Journey

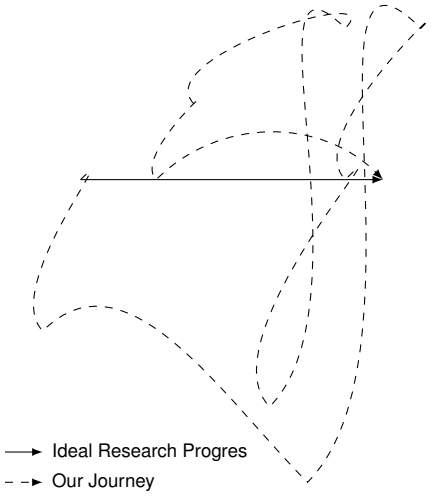


Summary of Exanauts' Journey



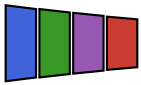
- **Nonlinear optimization/linear algebra**: distributed Jacobi, Lagrangian decomposition, ADMM, batched TRON solver, domain decomposition preconditioners, reduced-space interior point method, condensed interior-point methods w/ inequality relaxation, ...
- **Modeling/automatic differentiation**: batched second-order adjoint sensitivity, SIMD abstraction, ...
- **Packages developed**: ExaPf, Argos, ExaADMM, BlockPowerFlow, ProxAL, ExaTron, MadNLP, ExaModels, ...

Summary of Exanauts' Journey



- **Nonlinear optimization/linear algebra**: distributed Jacobi, Lagrangian decomposition, ADMM, batched TRON solver, domain decomposition preconditioners, reduced-space interior point method, condensed interior-point methods w/ inequality relaxation, ...
- **Modeling/automatic differentiation**: batched second-order adjoint sensitivity, SIMD abstraction, ...
- **Packages developed**: ExaPf, Argos, ExaADMM, BlockPowerFlow, ProxAL, ExaTron, MadNLP, ExaModels, ...

Two Award-Winning Open-Source Projects (2023 COIN-OR Cup)



ExaModels

MadNLP

https://github.com/sshin23/ExaModels.jl

https://github.com/MadNLP/MadNLP.jl

How {AMPL, IPOPT, Ma27} Have Succeeded on CPUs

Modeling

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

Barrier Reformulation

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

Newton's Step Comp.

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

Line Search

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

How {AMPL, IPOPT, Ma27} Have Succeeded on CPUs

Modeling

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

Barrier Reformulation

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

Newton's Step Comp.

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

Line Search

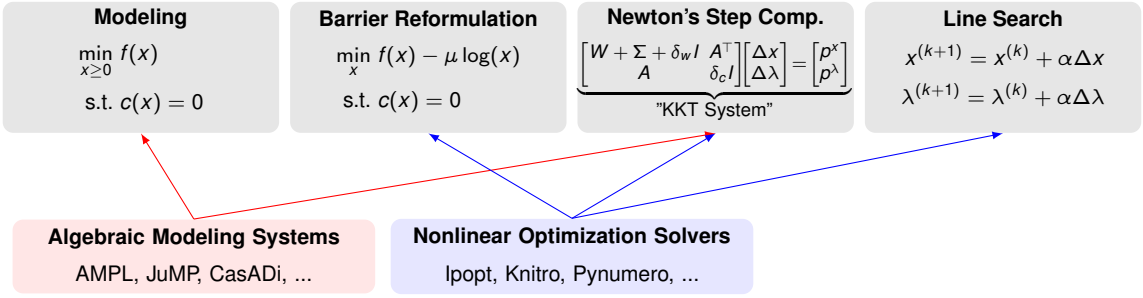
$$x^{(k+1)} = x^{(k)} + \alpha \Delta x$$
$$\lambda^{(k+1)} = \lambda^{(k)} + \alpha \Delta \lambda$$

Algebraic Modeling Systems

AMPL, JuMP, CasADi, ...

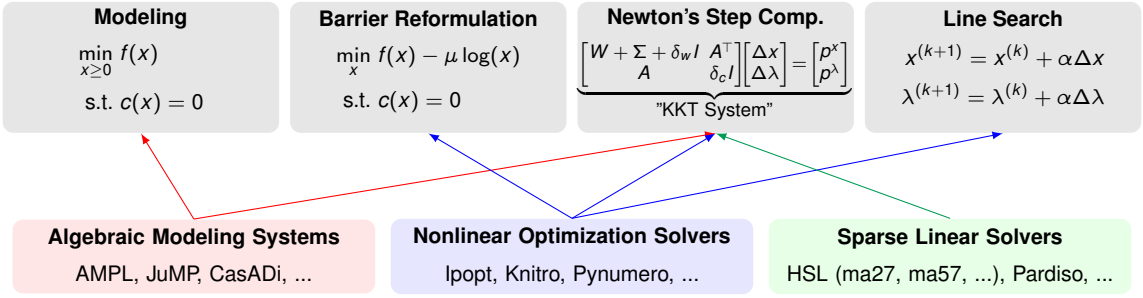
- Algebraic modeling systems have enabled efficient **sparse automatic differentiation**.

How {AMPL, IPOPT, Ma27} Have Succeeded on CPUs



- Algebraic modeling systems have enabled efficient **sparse automatic differentiation**.
- Interior-point solvers have removed the combinatorial complexities of inequality constraints; however, we pay the price by solving **extremely ill-conditioned linear systems**.

How {AMPL, IPOPT, Ma27} Have Succeeded on CPUs



- Algebraic modeling systems have enabled efficient **sparse automatic differentiation**.
- Interior-point solvers have removed the combinatorial complexities of inequality constraints; however, we pay the price by solving **extremely ill-conditioned linear systems**.
- Sparse indefinite solvers have enabled **the solution of ill-conditioned linear systems**.

Why it is Challenging: Sparse Automatic Differentiation

- **Automatic differentiation** is **more efficient** than *finite difference* or *symbolic differentiation*, and is **less prone to error** than *hand-written derivative codes*.

```
f(x){ ... }
```

```
df(x){ ... }
```

$$y = f(x)$$

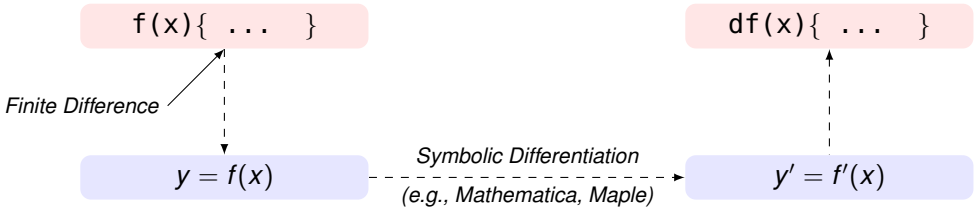
$$y' = f'(x)$$

Why it is Challenging: Sparse Automatic Differentiation

- **Automatic differentiation** is **more efficient** than *finite difference* or *symbolic differentiation*, and is **less prone to error** than *hand-written derivative codes*.

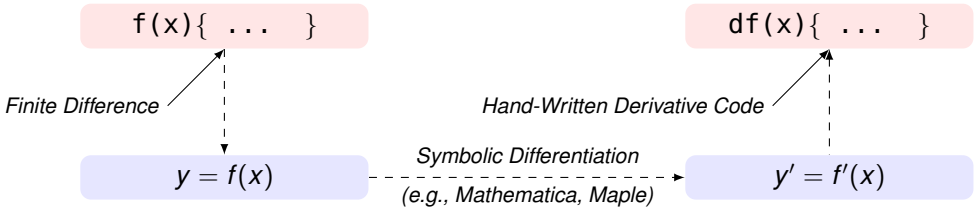
Why it is Challenging: Sparse Automatic Differentiation

- **Automatic differentiation** is **more efficient** than *finite difference* or *symbolic differentiation*, and is **less prone to error** than *hand-written derivative codes*.



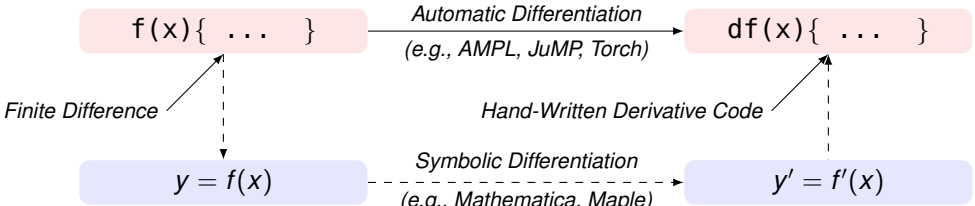
Why it is Challenging: Sparse Automatic Differentiation

- **Automatic differentiation** is **more efficient** than *finite difference* or *symbolic differentiation*, and is **less prone to error** than *hand-written derivative codes*.



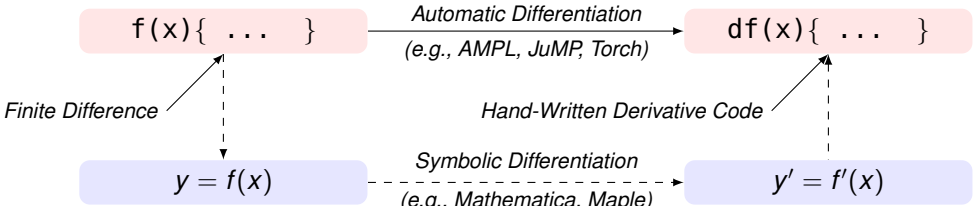
Why it is Challenging: Sparse Automatic Differentiation

- **Automatic differentiation** is **more efficient** than *finite difference* or *symbolic differentiation*, and is **less prone to error** than *hand-written derivative codes*.



Why it is Challenging: Sparse Automatic Differentiation

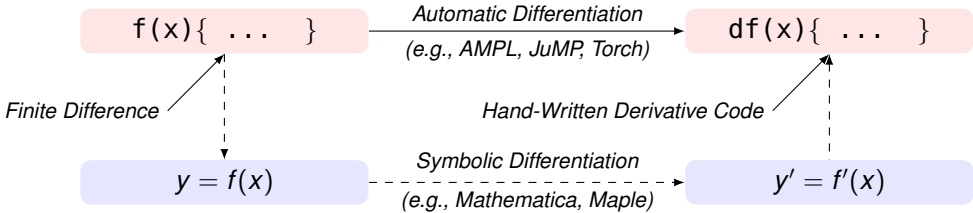
- **Automatic differentiation** is **more efficient** than *finite difference* or *symbolic differentiation*, and is **less prone to error** than *hand-written derivative codes*.



- Existing **algebraic modeling systems** (e.g., AMPL and JuMP) are **designed for CPUs**.

Why it is Challenging: Sparse Automatic Differentiation

- **Automatic differentiation** is **more efficient** than *finite difference* or *symbolic differentiation*, and is **less prone to error** than *hand-written derivative codes*.



- Existing **algebraic modeling systems** (e.g., AMPL and JuMP) are **designed for CPUs**.
- **Machine learning frameworks** (e.g., Torch) are not effective for **sparse problems**.

Solution: SIMD Abstraction

- Large-scale optimization problems **almost always have repetitive patterns**.
  e.g., optimal power flow w/ millions of variables can be expressed with a handful of patterns.
  - referencebus voltage angle
  - active and reactive power flow
  - voltage angle difference limits
  - apparent power flow limits
  - generation cost
  - power balance equations

Solution: SIMD Abstraction

- Large-scale optimization problems **almost always have repetitive patterns**.
  e.g., optimal power flow w/ millions of variables can be expressed with a handful of patterns.
  - referencebus voltage angle
  - active and reactive power flow
  - voltage angle difference limits
  - apparent power flow limits
  - generation cost
  - power balance equations

$$\min_{x^{\flat} \le x \le x^{\sharp}} \sum_{l \in [L]} \sum_{i \in [I_l]} f^{(l)}(x; p_i^{(l)}) \qquad \textbf{(SIMD abstraction)}$$

$$\text{s.t.} \quad \left[g^{(m)}(x; q_j) \right]_{j \in [J_m]} + \sum_{n \in [N_m]} \sum_{k \in [K_n]} h^{(n)}(x; s_k^{(n)}) = 0, \quad \forall m \in [M]$$

Solution: SIMD Abstraction

- Large-scale optimization problems **almost always have repetitive patterns**.
  e.g., optimal power flow w/ millions of variables can be expressed with a handful of patterns.
  - referencebus voltage angle
  - active and reactive power flow
  - voltage angle difference limits
  - apparent power flow limits
  - generation cost
  - power balance equations

$$\min_{x^\flat \le x \le x^\sharp} \sum_{l \in [L]} \sum_{i \in [I_l]} f^{(l)}(x; p_i^{(l)}) \qquad \textbf{(SIMD abstraction)}$$

$$\text{s.t.} \quad \left[g^{(m)}(x; q_j) \right]_{j \in [J_m]} + \sum_{n \in [N_m]} \sum_{k \in [K_n]} h^{(n)}(x; s_k^{(n)}) = 0, \quad \forall m \in [M]$$

- Repeated patterns are made available by always specifying the models as **iterable objects**.

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

Solution: SIMD Abstraction

- Large-scale optimization problems **almost always have repetitive patterns**.
  e.g., optimal power flow w/ millions of variables can be expressed with a handful of patterns.
  - referencebus voltage angle
  - active and reactive power flow
  - voltage angle difference limits
  - apparent power flow limits
  - generation cost
  - power balance equations

$$\min_{x^\flat \le x \le x^\natural} \sum_{l \in [L]} \sum_{i \in [I_l]} f^{(l)}(x; p_i^{(l)}) \qquad \textbf{(SIMD abstraction)}$$

$$\text{s.t.} \quad \left[g^{(m)}(x; q_j) \right]_{j \in [J_m]} + \sum_{n \in [N_m]} \sum_{k \in [K_n]} h^{(n)}(x; s_k^{(n)}) = 0, \quad \forall m \in [M]$$

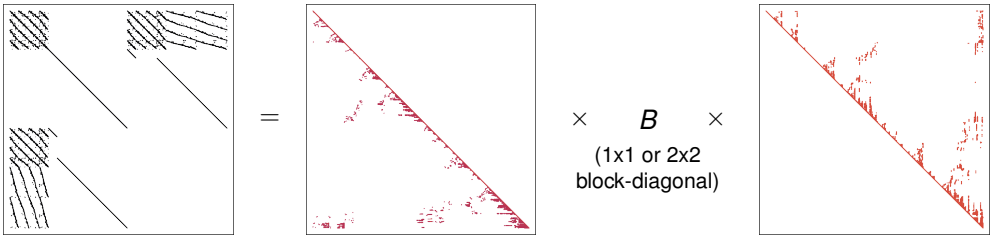
- Repeated patterns are made available by always specifying the models as **iterable objects**.

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

- **For each repeatitive pattern**, the derivative evaluation kernel is constructed & compiled, and it is **executed in parallel over multiple data**.

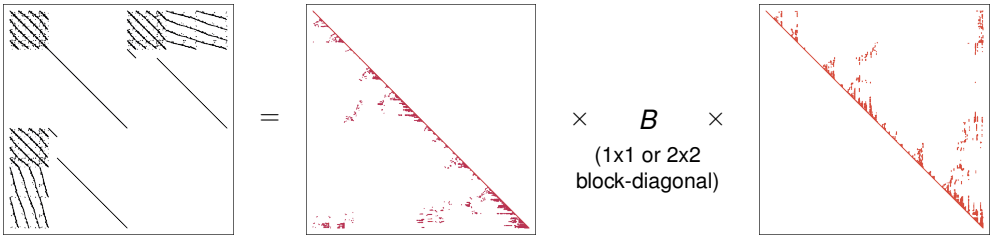
Why it is Challenging: Sparse Linear Solvers

- Solving KKT systems on CPUs has traditionally relied on **direct LBL$^\top$ factorization**.



Why it is Challenging: Sparse Linear Solvers

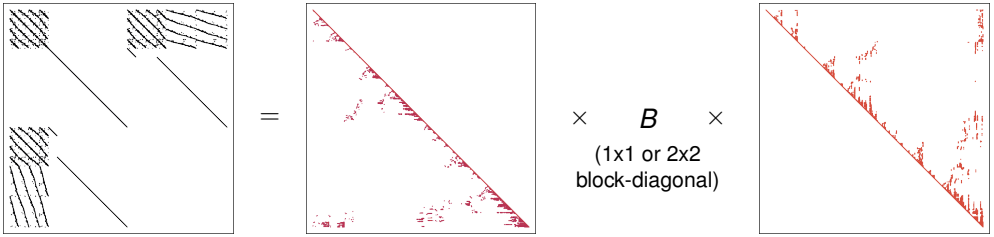
- Solving KKT systems on CPUs has traditionally relied on **direct LBL$^\top$ factorization**.



- Achieving numerical stability necessitates **numerical pivoting**, a process that is notoriously **challenging to parallelize**.

Why it is Challenging: Sparse Linear Solvers

- Solving KKT systems on CPUs has traditionally relied on **direct LBL$^\top$ factorization**.



- Achieving numerical stability necessitates **numerical pivoting**, a process that is notoriously **challenging to parallelize**.
- Due to the ill-conditioning of the KKT system, **iterative methods** are generally impractical, although there have been some promising results with **specialized preconditioners**.

Curtis, Huber, Schenk, Waechter. *A note on the implementation of an interior-point algorithm for nonlinear optimization with inexact step computations.* MathProg. (2012)
Cao, Seth, Laird. *An augmented Lagrangian interior-point approach for large-scale NLP problems on graphics processing units.* CACE. (2016)
Gondzio, Matrix-free interior point method, Comput Optim Appl (2012)

Solution #1: Condensed-Space Interior Point Method

- We aim to transform the KKT systems into a **sparse positive definite system**.

Solution #1: Condensed-Space Interior Point Method

- We aim to transform the KKT systems into a **sparse positive definite system**.
- Can be achieved by (i) converting the **equalities into inequalities**:

$$g(x) = 0 \quad \Longrightarrow \quad g(x) - s = 0, \quad s^{\flat} \leq s \leq s^{\sharp},$$

Solution #1: Condensed-Space Interior Point Method

- We aim to transform the KKT systems into a **sparse positive definite system**.
- Can be achieved by (i) converting the **equalities into inequalities**:

$$g(x) = 0 \quad \Longrightarrow \quad g(x) - s = 0, \quad s^\flat \leq s \leq s^\sharp,$$

(ii) **eliminating the slack variables** (so-called *condensation*):

$$\begin{bmatrix} W^{(\ell)} + \delta_w^{(\ell)} I & & A^{(\ell)\top} & -I & I & & & \\ & \delta_w^{(\ell)} I & -I & & & -I & I & \\ A^{(\ell)} & -I & -\delta_c^{(\ell)} I & & & & & \\ Z_x^{(\ell)\flat} & & & X^{(\ell)} - X^\flat & & & & \\ -Z_x^{(\ell)\sharp} & & & & X^\sharp - X^{(\ell)} & & & \\ & Z_s^{(\ell)\flat} & & & & S^{(\ell)} - S^\flat & & \\ & -Z_s^{(\ell)\sharp} & & & & & S^\sharp - S^{(\ell)} & \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta s \\ \Delta y \\ \Delta z_x^\flat \\ \Delta z_x^\sharp \\ \Delta z_s^\flat \\ \Delta z_s^\sharp \end{bmatrix} = \begin{bmatrix} p_x^{(\ell)} \\ p_s^{(\ell)} \\ p_y^{(\ell)} \\ p_{z_x^\flat}^{(\ell)} \\ p_{z_x^\sharp}^{(\ell)} \\ p_{z_s^\flat}^{(\ell)} \\ p_{z_s^\sharp}^{(\ell)} \end{bmatrix}$$

$$\Longrightarrow \quad (W + \delta_w I + \Sigma_x + A^\top D A)\Delta x = q_x + A^\top (C q_s + D q_y)$$

Solution #1: Condensed-Space Interior Point Method

- We aim to transform the KKT systems into a **sparse positive definite system**.
- Can be achieved by (i) converting the **equalities into inequalities**:

$$g(x) = 0 \quad \Longrightarrow \quad g(x) - s = 0, \quad s^\flat \leq s \leq s^\sharp,$$

(ii) **eliminating the slack variables** (so-called *condensation*):

$$\begin{bmatrix} W^{(\ell)} + \delta_w^{(\ell)} I & & A^{(\ell)\top} & -I & I & & & \\ & \delta_w^{(\ell)} I & -I & & & -I & I & \\ A^{(\ell)} & -I & -\delta_c^{(\ell)} I & & & & & \\ Z_x^{(\ell)\flat} & & & X^{(\ell)} - X^\flat & & & & \\ -Z_x^{(\ell)\sharp} & & & & X^\sharp - X^{(\ell)} & & & \\ & Z_s^{(\ell)\flat} & & & & S^{(\ell)} - S^\flat & & \\ & -Z_s^{(\ell)\sharp} & & & & & S^\sharp - S^{(\ell)} & \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta s \\ \Delta y \\ \Delta z_x^\flat \\ \Delta z_x^\sharp \\ \Delta z_s^\flat \\ \Delta z_s^\sharp \end{bmatrix} = \begin{bmatrix} p_x^{(\ell)} \\ p_s^{(\ell)} \\ p_y^{(\ell)} \\ p_{z_x^\flat}^{(\ell)} \\ p_{z_x^\sharp}^{(\ell)} \\ p_{z_s^\flat}^{(\ell)} \\ p_{z_s^\sharp}^{(\ell)} \end{bmatrix}$$

$$\Longrightarrow \quad (W + \delta_w I + \Sigma_x + A^\top D A)\Delta x = q_x + A^\top (C q_s + D q_y)$$

- These systems can be solved **without numerical pivoting** (routines available in CUDA).

Solution #2: Reduction & Dense Reformulation

- Oftentimes, the decision variables can be **splitted into state** and **control variables**.

L. Biegler, J. Nocedal, C. Schmid. *A reduced Hessian method for large-scale constrained optimization.* SIOPT (1995).
F. Pacaud, S. Shin, M. Schanen, D. A. Maldonado, and M. Anitescu. *Accelerating condensed interior-point methods on SIMD/GPU architectures.* JOTA (2023).
D. Cole, S. Shin, F. Pacaud, V. M. Zavala, and M. Anitescu. *Exploiting GPU/SIMD architectures for solving linear-quadratic MPC problems.* ACC (2023).

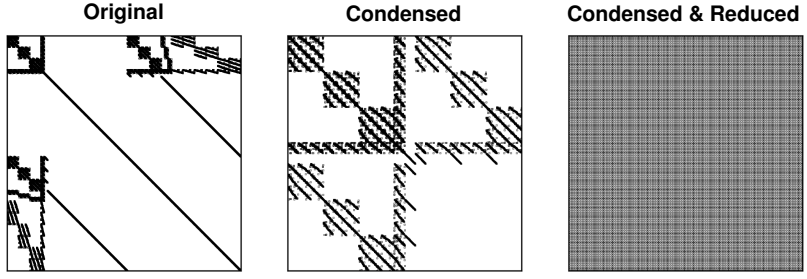
Solution #2: Reduction & Dense Reformulation

- Oftentimes, the decision variables can be **splitted into state** and **control variables**.
- We **reduce the KKT systems** by **elimintating the state variables** [Biegler, 1995].

L. Biegler, J. Nocedal, C. Schmid. *A reduced Hessian method for large-scale constrained optimization.* SIOPT (1995).
F. Pacaud, S. Shin, M. Schanen, D. A. Maldonado, and M. Anitescu. *Accelerating condensed interior-point methods on SIMD/GPU architectures.* JOTA (2023).
D. Cole, S. Shin, F. Pacaud, V. M. Zavala, and M. Anitescu. *Exploiting GPU/SIMD architectures for solving linear-quadratic MPC problems.* ACC (2023).

Solution #2: Reduction & Dense Reformulation

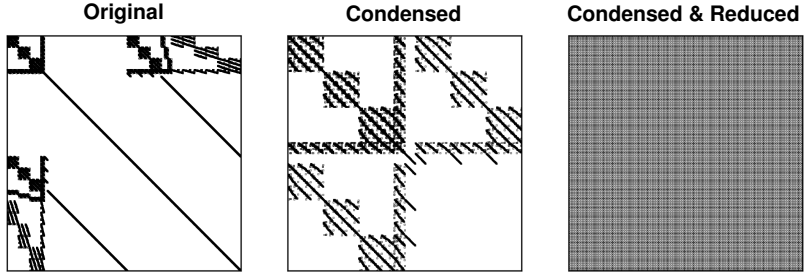
- Oftentimes, the decision variables can be **splitted into state** and **control variables**.
- We **reduce the KKT systems** by **elimintating the state variables** [Biegler, 1995].
- This **reduction** transforms the KKT system into a compact, dense, positive definite system.



L. Biegler, J. Nocedal, C. Schmid. *A reduced Hessian method for large-scale constrained optimization.* SIOPT (1995).
F. Pacaud, S. Shin, M. Schanen, D. A. Maldonado, and M. Anitescu. *Accelerating condensed interior-point methods on SIMD/GPU architectures.* JOTA (2023).
D. Cole, S. Shin, F. Pacaud, V. M. Zavala, and M. Anitescu. *Exploiting GPU/SIMD architectures for solving linear-quadratic MPC problems.* ACC (2023).

Solution #2: Reduction & Dense Reformulation

- Oftentimes, the decision variables can be **splitted into state** and **control variables**.
- We **reduce the KKT systems** by **elimintating the state variables** [Biegler, 1995].
- This **reduction** transforms the KKT system into a compact, dense, positive definite system.



- **The elimination is a bottleneck**, but can be accelerated by exploiting structures (e.g., Riccati recursion for optimal control).

L. Biegler, J. Nocedal, C. Schmid. *A reduced Hessian method for large-scale constrained optimization.* SIOPT (1995).
F. Pacaud, S. Shin, M. Schanen, D. A. Maldonado, and M. Anitescu. *Accelerating condensed interior-point methods on SIMD/GPU architectures.* JOTA (2023).
D. Cole, S. Shin, F. Pacaud, V. M. Zavala, and M. Anitescu. *Exploiting GPU/SIMD architectures for solving linear-quadratic MPC problems.* ACC (2023).

Highlight: AC Optimal Power Flow (single GPU)

- Standard form polar form AC optimal power flow (AC OPF) problems.
- The condensed-space interior point method is employed.
- Solved up to 10^{-4} precision (note that by default, Ipopt solves up to 10^{-8} precision).

Highlight: AC Optimal Power Flow (single GPU)

- Standard form polar form AC optimal power flow (AC OPF) problems.
- The condensed-space interior point method is employed.
- Solved up to 10^{-4} precision (note that by default, Ipopt solves up to 10^{-8} precision).
- For large-scale cases, GPU becomes **significantly faster than CPU**

Case	# vars	# cons	**MadNLP+ExaModels+cuSOLVER (GPU*)**			**Ipopt+AMPL+Ma27 (CPU**)**		
			# iter	AD†	total†	# iter	AD†	total†
10480_goc	96.8k	150.9k	70	0.13	14.26	64	16.93	38.04
13659_pegase	117.4k	170.6k	63	0.12	7.15	64	19.70	35.66
19402_goc	179.6k	281.7k	79	0.17	23.28	70	36.50	95.34
24464_goc	203.4k	313.6k	63	0.11	70.63	58	33.50	70.15
30000_goc	208.6k	307.8k	162	0.33	22.05	180	101.98	249.81

AD = Automatic Differentiation

†Wall time (sec) measured by Julia. ‡CPU time (sec) reported by Ipopt.

*NVIDIA Quadro GV100 **Intel Xeon Gold 6140

Highlight: AC Optimal Power Flow (single GPU)

- Standard form polar form AC optimal power flow (AC OPF) problems.
- The condensed-space interior point method is employed.
- Solved up to 10^{-4} precision (note that by default, Ipopt solves up to 10^{-8} precision).
- For large-scale cases, GPU becomes **significantly faster than CPU** (up to $\times 10$ speedup).

Case	# vars	# cons	**MadNLP+ExaModels+cuSOLVER (GPU*)**			**Ipopt+AMPL+Ma27 (CPU**)**		
			# iter	AD†	total†	# iter	AD†	total†
10480_goc	96.8k	150.9k	70	0.13	14.26	64	16.93	38.04
13659_pegase	117.4k	170.6k	63	0.12	7.15	64	19.70	35.66
19402_goc	179.6k	281.7k	79	0.17	23.28	70	36.50	95.34
24464_goc	203.4k	313.6k	63	0.11	70.63	58	33.50	70.15
30000_goc	208.6k	307.8k	162	0.33	22.05	180	101.98	249.81

AD = Automatic Differentiation

†Wall time (sec) measured by Julia. ‡CPU time (sec) reported by Ipopt.

*NVIDIA Quadro GV100 **Intel Xeon Gold 6140

Highlight: AC Optimal Power Flow (single GPU)

- Standard form polar form AC optimal power flow (AC OPF) problems.
- The condensed-space interior point method is employed.
- Solved up to 10^{-4} precision (note that by default, Ipopt solves up to 10^{-8} precision).
- For large-scale cases, GPU becomes **significantly faster than CPU** (up to $\times 10$ speedup).

Case	# vars	# cons	MadNLP+ExaModels+cuSOLVER (GPU*)			Ipopt+AMPL+Ma27 (CPU**)		
			# iter	AD†	total†	# iter	AD†	total†
10480_goc	96.8k	150.9k	70	0.13	14.26	64	16.93	38.04
13659_pegase	117.4k	170.6k	63	0.12	7.15	64	19.70	35.66
19402_goc	179.6k	281.7k	79	0.17	23.28	70	36.50	95.34
24464_goc	203.4k	313.6k	63	0.11	70.63	58	33.50	70.15
30000_goc	208.6k	307.8k	162	0.33	22.05	180	101.98	249.81

AD = Automatic Differentiation

†Wall time (sec) measured by Julia. ‡CPU time (sec) reported by Ipopt.

*NVIDIA Quadro GV100 **Intel Xeon Gold 6140

- Automatic differentiation is $\times 300$ faster, and the rest of computation is $\times 8$ faster.

Highlight: Security-Constrained AC OPF (multi GPU)

- Standard security-constrained AC optimal power flow (SC AC OPF) problems.
- The **reduction & dense formulation** and the **classical Schur-complement method for block-structured problems** are employed.

Highlight: Security-Constrained AC OPF (multi GPU)

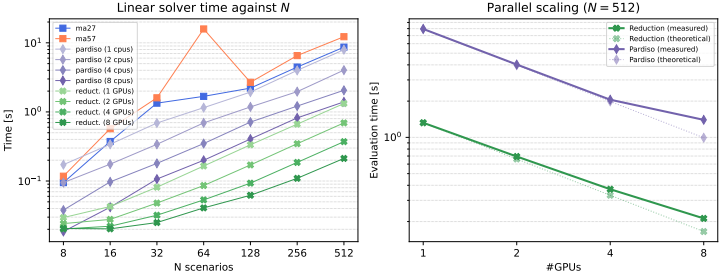
- Standard security-constrained AC optimal power flow (SC AC OPF) problems.
- The **reduction & dense formulation** and the **classical Schur-complement method for block-structured problems** are employed.
- Leveraging **multiple GPUs** allows us to **overcome the limitations** of a single GPU.

	1354pegase				**2869pegase**				**9241pegase**			
	# iter	AD	LA	total	# iter	AD	LA	total	# iter	AD	LA	total
CPU*	44	2.6	4.2	7.0	77	11.9	27.4	40.3	136	205.6	771.8	984.1
1 GPU**	44	0.3	1.8	2.1	93	1.1	11.7	12.8	98	5.5	112.3	117.8
2 GPUs**	44	0.3	1.1	1.4	93	0.8	7.4	8.2	98	3.4	56.8	60.2
4 GPUs**	44	0.3	1.0	1.3	93	0.8	5.7	6.5	98	2.3	35.8	38.1
8 GPUs**	44	0.2	1.0	1.2	93	0.6	5.1	5.7	98	1.4	26.4	27.7
# vars	20k				42k				139k			
# cons	53k				119k				404k			

AD = Automatic Differentiation, LA = Linear Algebra (KKT System Solver)
*AMD CPU Epyc "Milan" **NVIDIA A100 (4 GPUs per node)

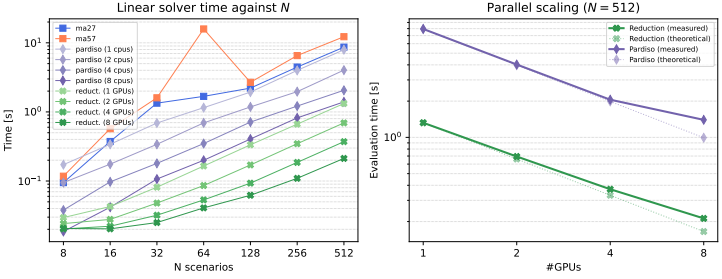
Highlight: Security-Constrained AC OPF (multi GPU)

- Our approach is $\times 6$ faster than commercial multi-CPU solver.

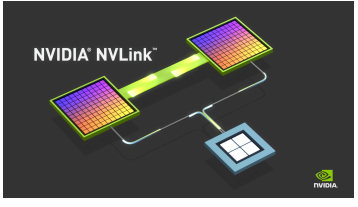


Highlight: Security-Constrained AC OPF (multi GPU)

- Our approach is $\times 6$ faster than commercial multi-CPU solver.



- Single-node, multi-GPU parallelism is more efficient than multi-node, multi-GPU.



Conclusions

- Modern GPU hardware offers significant potential for accelerating large-scale optimization.
- Implementing algorithms on GPUs can be challenging and often **requires reevaluating key aspects of existing algorithms**.
- We have achieved promising results: up to **10x faster solutions with moderate accuracy**, though challenges remain for high-accuracy solutions.
- GPU-based optimization opens up opportunities to address previously intractable problems:
  - **Extremely large-scale** problems (multi-stage, multiscale).
  - Problems involving **expensive surrogate models** (neural nets, simulations).

Conclusions

- Modern GPU hardware offers significant potential for accelerating large-scale optimization.
- Implementing algorithms on GPUs can be challenging and often **requires reevaluating key aspects of existing algorithms**.
- We have achieved promising results: up to **10x faster solutions with moderate accuracy**, though challenges remain for high-accuracy solutions.
- GPU-based optimization opens up opportunities to address previously intractable problems:
  - **Extremely large-scale** problems (multi-stage, multiscale).
  - Problems involving **expensive surrogate models** (neural nets, simulations).

Shin Group @ MIT is hiring postdocs! Find more details here:
`https://shin.mit.edu`

Acknowledgement

U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (DE-AC02-06CH11347)

U.S. Department of Energy, Office of Science and National Nuclear Security Administration, Exascale Computing Porject (17-SC-20-SC)

[1] M. Anitescu, K. Kim, Y. Kim, A. Maldonado, F. Pacaud, V. Rao, M. Schanen, S. Shin, and A. Subramanian. Targeting Exascale with Julia on GPUs for multiperiod optimization with scenario constraints. *SIAG/OPT Views and News*, 2021.

[2] F. Pacaud, M. Schanen, S. Shin, D. A. Maldonado, and M. Anitescu. Parallel interior-point solver for block-structured nonlinear programs on SIMD/GPU architectures, 2023, arXiv:2301.04869. Under Review.

[3] F. Pacaud, S. Shin, M. Schanen, D. A. Maldonado, and M. Anitescu. Accelerating condensed interior-point methods on SIMD/GPU architectures. *Journal of Optimization Theory and Applications*, pages 1–20, 2023, arXiv:2203.11875.

[4] S. Shin, F. Pacaud, and M. Anitescu. Accelerating optimal power flow with GPUs: SIMD abstraction of nonlinear programs and condensed-space interior-point methods, arXiv:2307.16830. Under Review.

Nonlinear Optimization on Graphics Processing Units

Sungho Shin[*,1], **François Pacaud**[2], **Mihai Anitescu**[1,3], and **Exanauts**

[*]sshin@anl.gov

[1]Mathematics and Computer Science Division, Argonne National Laboratory
[2]Centre Automatique et Systèmes, Mines Paris - PSL
[3]Department of Statistics, University of Chicago

2023 AIChE Meeting, Orlando, Florida