Approaches to Nonlinear Programming on GPU Architectures

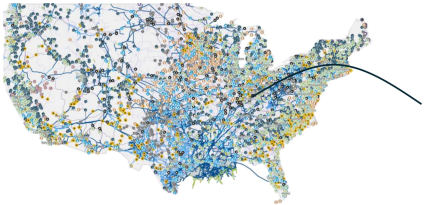
Alexis Montoison[1], Sungho Shin[2], François Pacaud[3], Michel Schanen[2], and Mihai Anitescu[2,4]

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

April 17, 2024
GEEKS – Gurobi

Takeaways

- **Accelerated computing** offers significant opportunities for **systems engineering**
  - A nonlinear solver on GPU is 10x faster than the state-of-the-art open-source solver on CPU [Shin, 2024]



Can optimize the operation of the **entire eastern interconnection** in <20 seconds

  - A linear programming solver on GPU is comparable to state-of-the-art commercial solvers on CPU [Lu, 2024]
- We envision **expanding the application scope** of **optimization** with accelerated computing.
  - **Extremely large-scale** problems (coupled infrastructures, multi-stage, multiscale)
  - Optimization with **AI models** (e.g., surrogates) and **digital twins**

Shin, Pacaud, and Anitescu. *Accelerating optimal power flow with GPUs: SIMD abstraction of nonlinear programs and condensed-space interior-point methods*, PSCC 2024.
Lu and Yang. *cuPDLP.jl: A GPU Implementation of Restarted Primal-Dual Hybrid Gradient for Linear Programming in Julia*, arXiv:2311.12180.

Outline

Outline

Accelerated Computing in 2024

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

Accelerated Computing in 2024

- 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 in 2024

- 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

Mostly powered by GPUs

$=$ 1 million $\times$

iPhone 14 Pro

Accelerated Computing in 2024

- 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

Mostly powered by GPUs

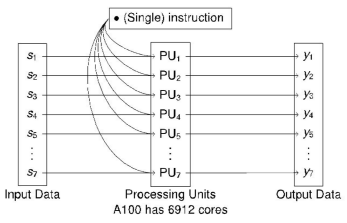
$=$ 1 million $\times$

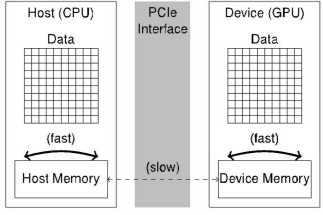
iPhone 14 Pro

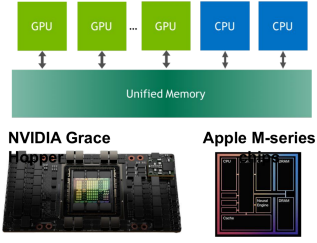
Can we harness these capabilities in the realm of **classical nonlinear optimization** (e.g., energy infrastures, optimal control, operations research)?

How Do GPUs 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.**





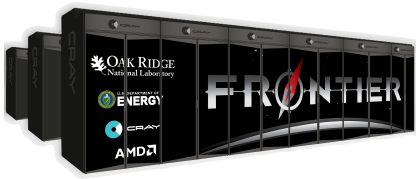


Adapting CPU code to GPU code is not merely a matter of software engineering; it often requires the **redesign of the algorithm**

Exascale Computing Project

- **Mission**: Tackle **real-world computational problems** with exascale computing.

Frontier @Oak Ridge

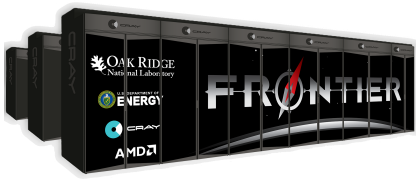


Aurora @Argonne

Exascale Computing Project

- **Mission**: Tackle **real-world computational problems** with exascale computing.

Frontier @Oak Ridge



Aurora @Argonne

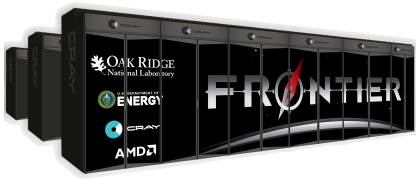
- **Goal**: Build a **comprehensive software infrastructure** for nonlinear optimization on GPUs.

M. Schanen, D. A. Maldonado, F. Pacaud, A. Montoison, M. Anitescu, K. Kim, Y. Kim, V. Rao, A. Subramanyam.
Julia as a portable high-level language for numerical solvers of power flow equations on GPU architectures, 2020

Exascale Computing Project

- **Mission**: Tackle **real-world computational problems** with exascale computing.

Frontier @Oak Ridge (AMD GPUs)



Aurora @Argonne (Intel GPUs)

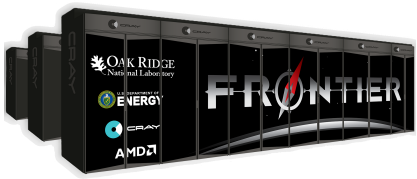
- **Goal**: Build a **comprehensive software infrastructure** for nonlinear optimization on GPUs.
- **Challenge #1**: No software infrastructure for **classical nonlinear optimization** on GPUs.

M. Schanen, D. A. Maldonado, F. Pacaud, A. Montoison, M. Anitescu, K. Kim, Y. Kim, V. Rao, A. Subramanyam.
Julia as a portable high-level language for numerical solvers of power flow equations on GPU architectures, 2020

Exascale Computing Project

- **Mission**: Tackle **real-world computational problems** with exascale computing.

Frontier @Oak Ridge (AMD GPUs)



Aurora @Argonne (Intel GPUs)

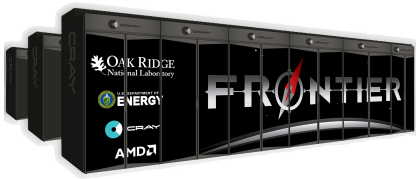
- **Goal**: Build a **comprehensive software infrastructure** for nonlinear optimization on GPUs.
- **Challenge #1**: No software infrastructure for **classical nonlinear optimization** on GPUs.
- **Challenge #2**: **Heterogeneous** development environment (NVIDIA, AMD, and Intel).

M. Schanen, D. A. Maldonado, F. Pacaud, A. Montoison, M. Anitescu, K. Kim, Y. Kim, V. Rao, A. Subramanyam.
Julia as a portable high-level language for numerical solvers of power flow equations on GPU architectures, 2020

Exascale Computing Project

- **Mission**: Tackle **real-world computational problems** with exascale computing.

Frontier @Oak Ridge (AMD GPUs)



Aurora @Argonne (Intel GPUs)

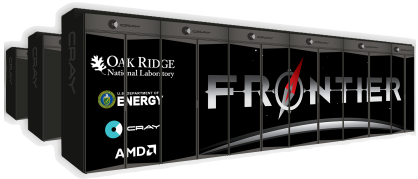
- **Goal**: Build a **comprehensive software infrastructure** for nonlinear optimization on GPUs.
- **Challenge #1**: No software infrastructure for **classical nonlinear optimization** on GPUs.
- **Challenge #2**: **Heterogeneous** development environment (NVIDIA, AMD, and Intel).
- Furthermore, we want to achieve
  - **Performance**: at least an order of magnitude speedup.

M. Schanen, D. A. Maldonado, F. Pacaud, A. Montoison, M. Anitescu, K. Kim, Y. Kim, V. Rao, A. Subramanyam.
Julia as a portable high-level language for numerical solvers of power flow equations on GPU architectures, 2020

Exascale Computing Project

- **Mission**: Tackle **real-world computational problems** with exascale computing.

Frontier @Oak Ridge (AMD GPUs)



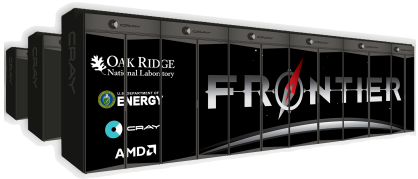
Aurora @Argonne (Intel GPUs)

- **Goal**: Build a **comprehensive software infrastructure** for nonlinear optimization on GPUs.
- **Challenge #1**: No software infrastructure for **classical nonlinear optimization** on GPUs.
- **Challenge #2**: **Heterogeneous** development environment (NVIDIA, AMD, and Intel).
- Furthermore, we want to achieve
  - **Performance**: at least an order of magnitude speedup.
  - **Portability**: compatibility with NVIDIA, AMD, and Intel.

Exascale Computing Project

- **Mission**: Tackle **real-world computational problems** with exascale computing.

Frontier @Oak Ridge (AMD GPUs)

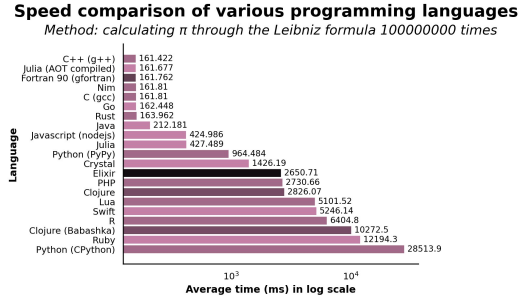


Aurora @Argonne (Intel GPUs)

- **Goal**: Build a **comprehensive software infrastructure** for nonlinear optimization on GPUs.
- **Challenge #1**: No software infrastructure for **classical nonlinear optimization** on GPUs.
- **Challenge #2**: **Heterogeneous** development environment (NVIDIA, AMD, and Intel).
- Furthermore, we want to achieve
  - **Performance**: at least an order of magnitude speedup.
  - **Portability**: compatibility with NVIDIA, AMD, and Intel.
  - **Application**: energy infrastructure problems (AC optimal power flow, in particular).

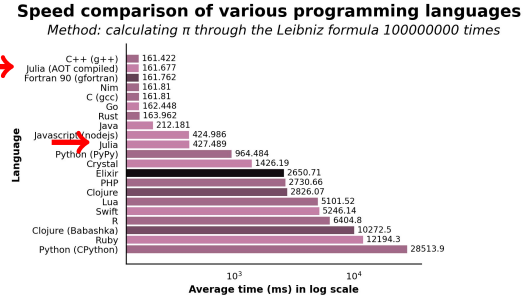
Language of Choice: Julia

- Runs **as fast as C/C++/Fortran**, but is as simple as Python/R/MATLAB.



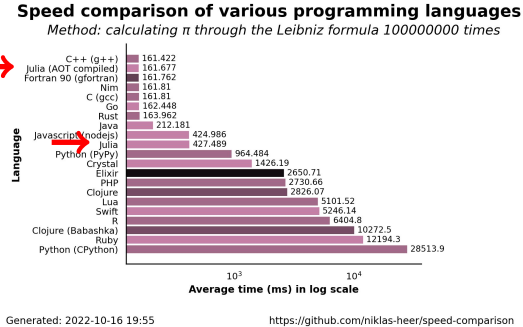
Language of Choice: Julia

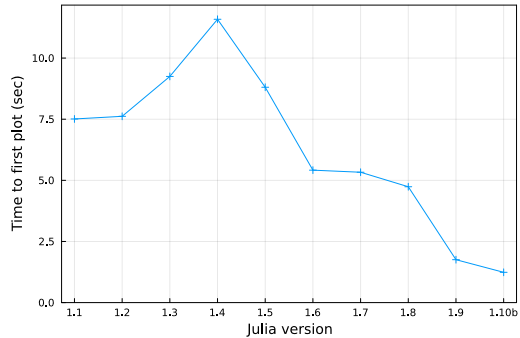
- Runs **as fast as C/C++/Fortran**, but is as simple as Python/R/MATLAB.



Language of Choice: Julia

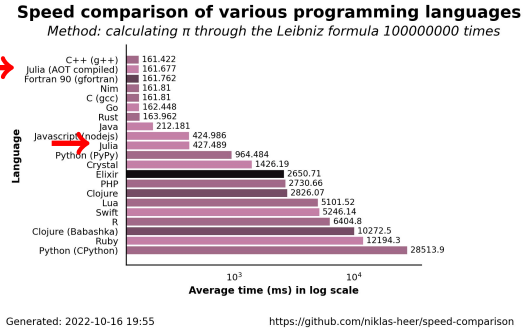
- Runs **as fast as C/C++/Fortran**, but is as simple as Python/R/MATLAB.

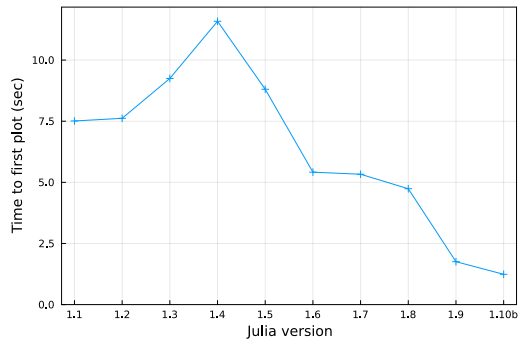




Language of Choice: Julia

- Runs **as fast as C/C++/Fortran**, but is as simple as Python/R/MATLAB.





- **Fast development**: Julia resolves the “**two-language problem**”.

Language of Choice: Julia

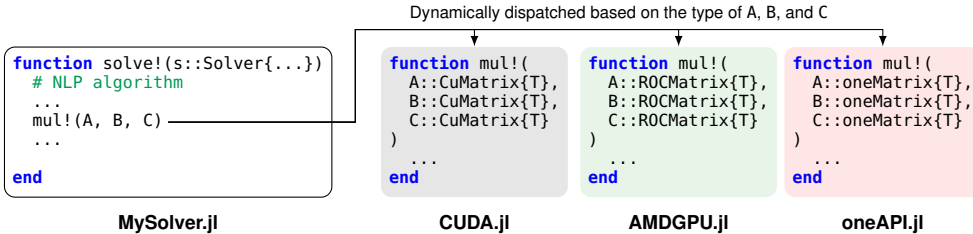
- **Multiple dispatch**: High-level abstraction while specializing for specific data types.

```
function solve!(s::Solver{...})
  # NLP algorithm
  ...
  mul!(A, B, C)
  ...

end
```

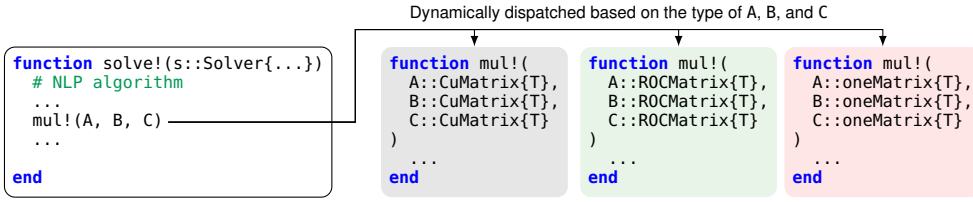
Language of Choice: Julia

- **Multiple dispatch**: High-level abstraction while specializing for specific data types.



Language of Choice: Julia

- **Multiple dispatch**: High-level abstraction while specializing for specific data types.



- **Portable kernel programming**: Compatibility across various architectures.

```
@kernel function _mul!(
    A,B,C
  )
  # portable GPU kernel
  ...
end
```

Language of Choice: Julia

- **Multiple dispatch**: High-level abstraction while specializing for specific data types.

Dynamically dispatched based on the type of A, B, and C

```
function solve!(s::Solver{...})
  # NLP algorithm
  ...
  mul!(A, B, C)
  ...
end
```

MySolver.jl

```
function mul!(
  A::CuMatrix{T},
  B::CuMatrix{T},
  C::CuMatrix{T}
)
  ...
end
```

CUDA.jl

```
function mul!(
  A::ROCMatrix{T},
  B::ROCMatrix{T},
  C::ROCMatrix{T}
)
  ...
end
```

AMDGPU.jl

```
function mul!(
  A::oneMatrix{T},
  B::oneMatrix{T},
  C::oneMatrix{T}
)
  ...
end
```

oneAPI.jl

- **Portable kernel programming**: Compatibility across various architectures.

Portable GPU kernels can be executed on different accelerator backends

```
@kernel function _mul!(
    A,B,C
  )
  # portable GPU kernel
  ...
end
```

Frontier @Oak Ridge (AMD GPUs)

Aurora @Argonne (Intel GPUs)

Summit @Oak Ridge (NVIDIA GPUs)

Summary

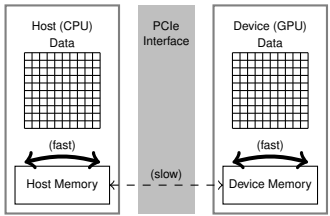
GPU/HPC

- **Motivation**: **Harness accelerated computing** for **nonlinear programming**.

Summary

GPU/HPC

SIMD Architecture

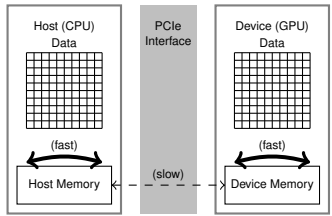


- **Motivation**: **Harness accelerated computing** for **nonlinear programming**.
- Adapting CPU code into GPU code is **not merely an issue of software engineering**.

Summary

GPU/HPC

SIMD Architecture



Performance, Fast Development, & Portability

julia

- **Motivation**: **Harness accelerated computing** for **nonlinear programming**.
- Adapting CPU code into GPU code is **not merely an issue of software engineering**.
- **Goal**: Build a **comprehensive software infrastructure** for nonlinear programming on GPUs on **Julia Language**, for its **fast performance**, **fast development speed**, and **portability**.

Outline

Nonlinear optimization framework: state-of-the-art on CPU

Problem Formulation

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

Nonlinear optimization framework: state-of-the-art on CPU

Problem Formulation

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

- In classical problems (e.g., optimal power flow),
  - the objective and constraints are **smooth**
  - **large number of variables and constraints**
  - the problem is **highly sparse**.

Nonlinear optimization framework: state-of-the-art on CPU

Problem Formulation

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

Newton's Step Computation

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

- In classical problems (e.g., optimal power flow),
  - the objective and constraints are **smooth**
  - **large number of variables and constraints**
  - the problem is **highly sparse**.
- Interior-point methods
  - Inequalities $x \geq 0$ replaced by smooth log-barrier functions $f(x) - \mu \sum_i \log(x[i])$.
  - **Newton's Step** is computed by solving a "**KKT system**" (large, sparse, symmetric indefinite, ill-conditioned system).

Nonlinear optimization framework: state-of-the-art on CPU

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

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

- In classical problems (e.g., optimal power flow),
  - the objective and constraints are **smooth**
  - **large number of variables and constraints**
  - the problem is **highly sparse**.
- Interior-point methods
  - Inequalities $x \geq 0$ replaced by smooth log-barrier functions $f(x) - \mu \sum_i \log(x[i])$.
  - **Newton's Step** is computed by solving a "**KKT system**" (large, sparse, symmetric indefinite, ill-conditioned system).
  - Line search (along with several additional heuristics) ensures **global convergence**.

Nonlinear optimization framework: state-of-the-art on CPU

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

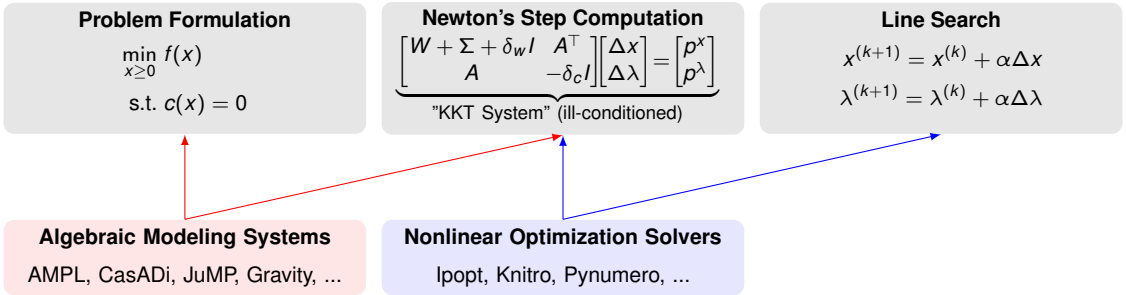
$$x^{(k+1)} = x^{(k)} + \alpha \Delta x$$
$$\lambda^{(k+1)} = \lambda^{(k)} + \alpha \Delta \lambda$$

Algebraic Modeling Systems

AMPL, CasADi, JuMP, Gravity, ...

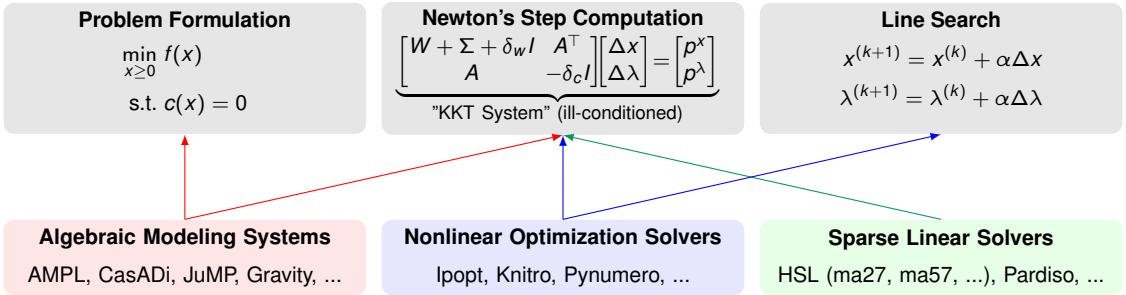
- Algebraic modeling systems provides **front-end** to specify models and (often) provides **derivative computation capabilities**.

Nonlinear optimization framework: state-of-the-art on CPU



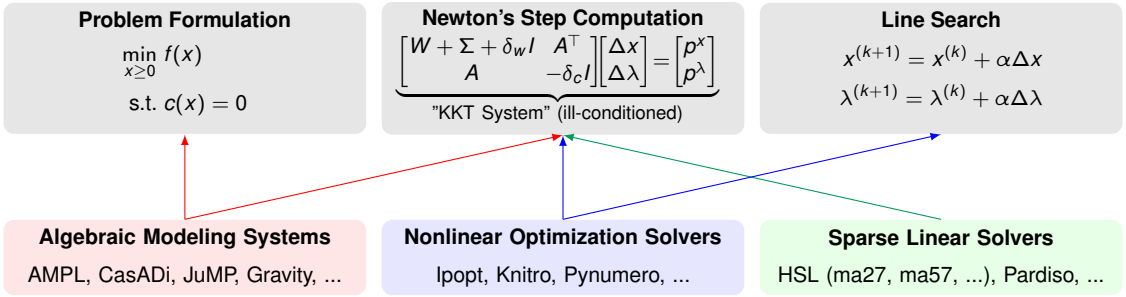
- Algebraic modeling systems provides **front-end** to specify models and (often) provides **derivative computation capabilities**.
- Nonlinear optimization solvers apply iterations of optimization algorithms.

Nonlinear optimization framework: state-of-the-art on CPU



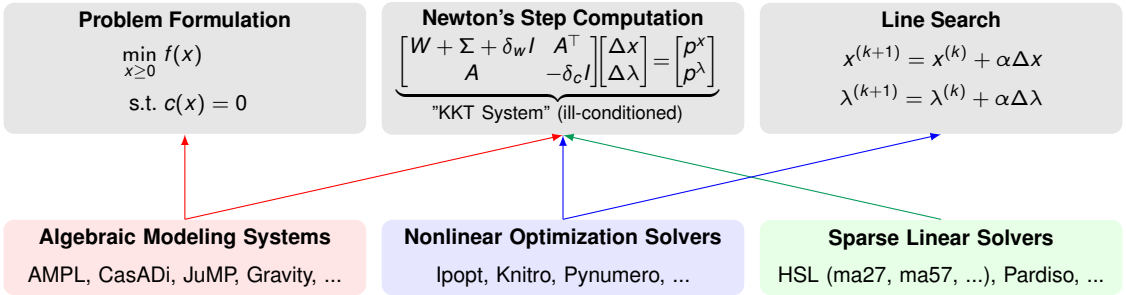
- Algebraic modeling systems provides **front-end** to specify models and (often) provides **derivative computation capabilities**.
- Nonlinear optimization solvers apply iterations of optimization algorithms.
- Sparse linear solvers solves KKT systems using **sparse matrix factorization**.

Nonlinear optimization framework: state-of-the-art on CPU



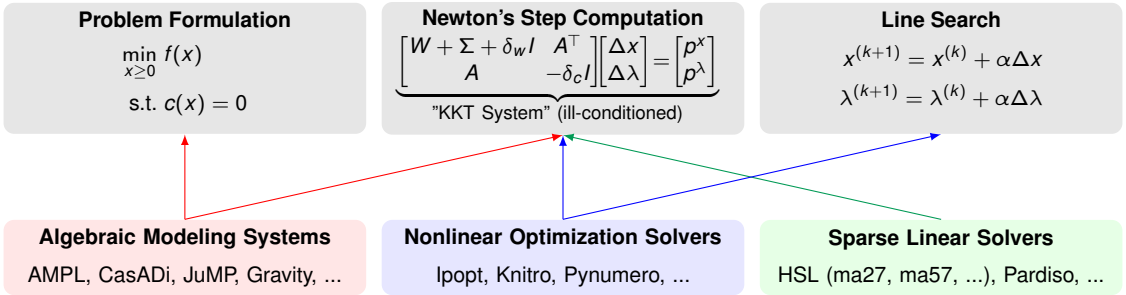
- These software tools have enabled the success of nonlinear optimization on CPUs.

Nonlinear optimization framework: state-of-the-art on CPU



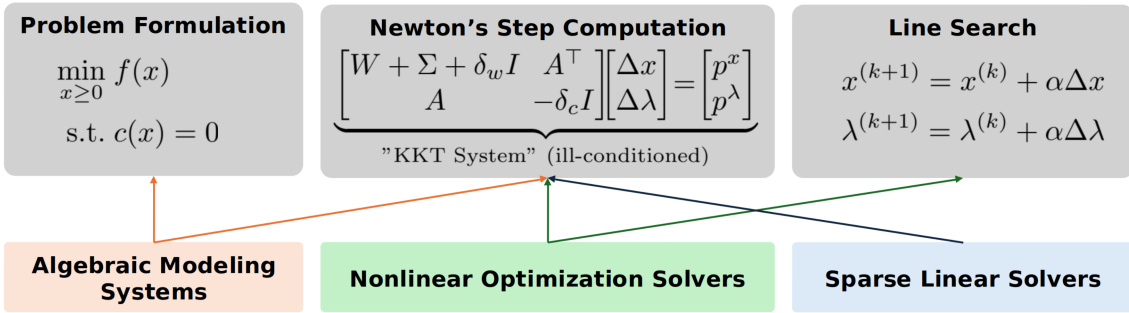
- These software tools have enabled the success of nonlinear optimization on CPUs.
- Many software tools have been developed in 80s-90s (**heavily optimized for CPUs**).

Nonlinear optimization framework: state-of-the-art on CPU



- These software tools have enabled the success of nonlinear optimization on CPUs.
- Many software tools have been developed in 80s-90s (**heavily optimized for CPUs**).
- Now we need **GPU-equivalent** of these tools.

Nonlinear optimization framework on GPUs



ExaModels

- Parallel AD with SIMD abstraction
- Runs on GPU architectures

MadNLP

- Lifted & Hybrid KKT System
- Runs on NVIDIA GPUs

NVIDIA CUDA

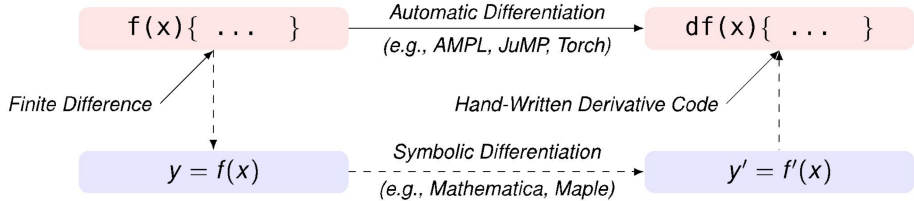
- Parallel Cholesky factorization
- Runs on NVIDIA GPUs

https://github.com/exanauts/ExaModels.jl
https://github.com/MadNLP/MadNLP.jl
https://github.com/exanauts/CUDSS.jl

Outline

Sparse Automatic Differentiation (AD)

- **Different paradigms** for **derivative computation**:



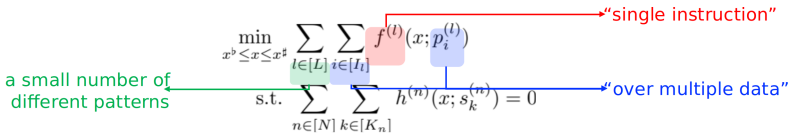
  - AD is superior in terms of **efficiency**, **accuracy**, and **convenience**
  - AD has been the **state-of-the-art method in numerical optimization**

- **Parallelizing** sparse automatic differentiation on GPUs is **non-trivial** due to **irregularities**

SIMD Abstraction

ExaModels

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



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

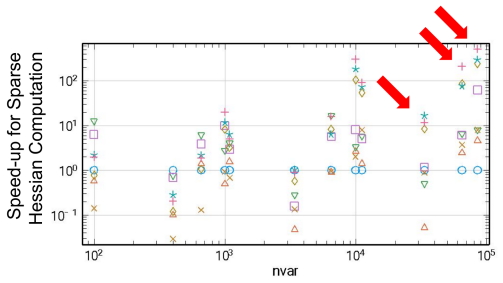
```
constraint(c, 3 * x[i+1]^3 + 2 * sin(x[i+2]) for i = 1:N-2)
```

"Instruction" "Data"

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

Sparse AD Benchmark

ExaModels



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

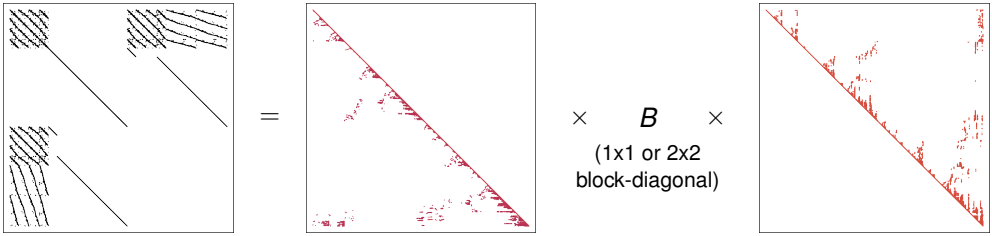
- For the largest case, **ExaModels on GPU** is **100× faster** than the state-of-the-art tools on CPUs
- ExaModels runs on **all major GPU architectures** and **single/multi-threaded CPUs**

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

Outline

Why it is Challenging: Sparse Linear Solvers

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



- LBL$^\top$ factorization requires **numerical pivoting**, which is **challenging to parallelize** [Swirydowicz, 2024].
- First-order methods [Lu, 2023] and iterative solvers[Schubiger, 2020] [Cao, 2016] have been proposed but can be ineffective/less reliable for nonconvex problems.
- We aim to develop a "pivoting-free" interior-point method.

Swirydowicz et. al. GPU-resident sparse direct linear solvers for alternating current optimal power flow analysis (2024).
Schubiger, Banjac, and Lygeros. GPU acceleration of ADMM for large-scale quadratic programming, JPDC (2020).
Cao, Seth, Laird. An augmented Lagrangian interior-point approach for large-scale NLP problems on graphics processing units, CACE (2016).

LiftedKKT

- We aim to transform the KKT system into an equivalent **positive definite system**, which can be **factorized without pivoting** based on **Cholseky factorization** (solver available for NVIDIA)
- This can be achieved by:
  - **Lift and relax** the equalities into **inequalities:** $g(x) = 0 \implies g(x) - s = 0, \quad s^\flat \le s \le s^\sharp,$ $\pm\epsilon_{\text{tol}} \approx 10^{-8}$
  - **Eliminate the slack variables** (so-called condensation):

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

Nonsingular

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

Positive-definite

- **Condensation** causes **extreme but relatively benign ill-conditioning** [M. Wright, 1998]

HybridKKT

- **Direct + iterative** approach, with **regularized equalities**
  - **Condense** KKT systems by **eliminating inequalities**:

$$\begin{bmatrix} W_k + \delta_x I & & H_k^\top & G_k^\top \\ & D_s + \delta_x I & I & \\ H_k & I & -\delta_c I & \\ G_k & & & -\delta_c I \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ d_z \\ d_y \end{bmatrix} = - \begin{bmatrix} r_1 \\ r_2 \\ r_4 \\ r_3 \end{bmatrix} \longrightarrow \overbrace{\begin{bmatrix} K_k & G_k^\top \\ G_k & 0 \end{bmatrix}}^{K_{cond}} \begin{bmatrix} d_x \\ d_y \end{bmatrix} = - \begin{bmatrix} r_1 + H_k^\top (D_s r_4 - r_2) \\ r_3 \end{bmatrix}$$

Nonsingular

Still indefinite

  - **Regularize** equalities with sufficiently large γ:

Positive-definite $\longleftarrow$ $K_\gamma := \begin{bmatrix} K_k + \gamma G_k^\top G_k & G_k^\top \\ G_k & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} = - \begin{bmatrix} r_\gamma \\ r_3 \end{bmatrix}$

  - **Solve the Schur complement system** with iterative method (e.g., conjugate gradient):

Converges to I as $\gamma \to \infty$ $\longleftarrow$ $(G_k K_\gamma^{-1} G_k^\top) d_y = r_3 - G_k K_\gamma^{-1} r_\gamma$

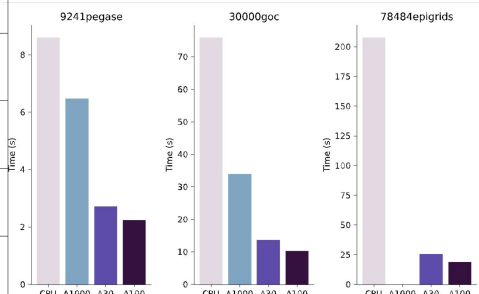
- Similarly to the “Lifted KKT System”, we have **extreme but relatively benign ill-conditioning**

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	0.33	**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	1.95	**3.57**	51	1.35	1.55	**3.51**
30000_goc	183	0.65	63.92	**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	9.91	**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)

Distillation Column

ExaModels + MadNLP + NVIDIA CUDA

	CPU			Lifted KKT on GPU			Hybrid KKT on GPU		
#time steps	init (s)	it	solve (s)	init (s)	it	solve (s)	init (s)	it	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

Outline

Accuracy

- We have **avoided indefinite systems** by replacing them with **positive definite systems**.

Accuracy

- We have **avoided indefinite systems** by replacing them with **positive definite systems**.
- Challenges arise from **ill-conditioning of condensed KKT systems**.

$$\begin{bmatrix} H + \Sigma_x & & J^\top \\ & \Sigma_s & -I \\ J & -I & \end{bmatrix} \xrightarrow{\text{condensation}} H + \Sigma_x + J^\top \Sigma_s^{-1} J$$

Accuracy

- We have **avoided indefinite systems** by replacing them with **positive definite systems**.
- Challenges arise from **ill-conditioning of condensed KKT systems**.

$$\begin{bmatrix} H+\Sigma_x & & J^\top \\ & \Sigma_s & -I \\ J & -I & \end{bmatrix} \xrightarrow{\text{condensation}} H+\Sigma_x+J^\top\Sigma_s^{-1}J$$

Causes ill-conditioning

Accuracy

- We have **avoided indefinite systems** by replacing them with **positive definite systems**.
- Challenges arise from **ill-conditioning of condensed KKT systems**.

$$\begin{bmatrix} H + \Sigma_x & & J^\top \\ & \Sigma_s & -I \\ J & -I & \end{bmatrix} \xrightarrow{\text{condensation}} H + \Sigma_x + J^\top \Sigma_s^{-1} J$$

Causes ill-conditioning

- **Alternative methods**, such as penalty method or augmented Lagrangian, **rely on similar manipulations**.

Accuracy

- We have **avoided indefinite systems** by replacing them with **positive definite systems**.
- Challenges arise from **ill-conditioning of condensed KKT systems**.

$$\begin{bmatrix} H+\Sigma_x & & J^\top \\ & \Sigma_s & -I \\ J & -I & \end{bmatrix} \xrightarrow{\text{condensation}} H+\Sigma_x+J^\top\Sigma_s^{-1}J$$

Causes ill-conditioning

- **Alternative methods**, such as penalty method or augmented Lagrangian, **rely on similar manipulations**.
- Work-arounds?
  - Using CPUs as we approach the solution.
  - Quadruple precision—challenging due to low-level kernel supports.
  - Hopefully, some breakthrough in sparse linear algebra (e.g., scalable parallel pivoting).

Portability

Table: GPU Compatibility of Nonlinear Optimization Frameworks

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

- **ExaModels** has **full compatibility** with multi-threaded CPUs, NVIDIA, AMD, and Intel GPUs.

Portability

Table: GPU Compatibility of Nonlinear Optimization Frameworks

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

- **ExaModels** has **full compatibility** with multi-threaded CPUs, NVIDIA, AMD, and Intel GPUs.
- **MadNLP** is currently **only compatible with NVIDIA**,

Portability

Table: GPU Compatibility of Nonlinear Optimization Frameworks

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

- **ExaModels** has **full compatibility** with multi-threaded CPUs, NVIDIA, AMD, and Intel GPUs.
- **MadNLP** is currently **only compatible with NVIDIA**, but making it portable is not difficult.

Portability

Table: GPU Compatibility of Nonlinear Optimization Frameworks

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

- **ExaModels** has **full compatibility** with multi-threaded CPUs, NVIDIA, AMD, and Intel GPUs.
- **MadNLP** is currently **only compatible with NVIDIA**, but making it portable is not difficult.
- **Main obstacle**: AMD and Intel GPUs are limited in terms of sparse linear solvers.

Portability

Table: GPU Compatibility of Nonlinear Optimization Frameworks

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

- **ExaModels** has **full compatibility** with multi-threaded CPUs, NVIDIA, AMD, and Intel GPUs.
- **MadNLP** is currently **only compatible with NVIDIA**, but making it portable is not difficult.
- **Main obstacle**: AMD and Intel GPUs are limited in terms of sparse linear solvers.
- Work-arounds?
  - Preconditioned iterative solvers.
  - Development of sparse linear solvers on GPU architectures?
    Work-in-Progress: A.Montoison and I. Duff. *MA57 on GPU* !!!

A. Montoison, D.Orban. *Krylov.jl: A Julia basket of hand-picked Krylov methods*, JOSS 2023
https://github.com/JuliaSmoothOptimizers/Krylov.jl

Outline

Conclusion

- Optimization on GPUs is a growing area (LPs, QPs, domain-specific problems)!

Conclusion

- Optimization on GPUs is a growing area (LPs, QPs, domain-specific problems)!
- **GPU hardware** offers significant potential for **accelerating large-scale optimization**.

Conclusion

- Optimization on GPUs is a growing area (LPs, QPs, domain-specific problems)!
- **GPU hardware** offers significant potential for **accelerating large-scale optimization**.
- Porting algorithms on GPUs often **requires complete redesign of the algorithms**.

Conclusion

- Optimization on GPUs is a growing area (LPs, QPs, domain-specific problems)!
- **GPU hardware** offers significant potential for **accelerating large-scale optimization**.
- Porting algorithms on GPUs often **requires complete redesign of the algorithms**.
- We have achieved promising results: up to **20x faster solutions with moderate accuracy**.

Conclusion

- Optimization on GPUs is a growing area (LPs, QPs, domain-specific problems)!
- **GPU hardware** offers significant potential for **accelerating large-scale optimization**.
- Porting algorithms on GPUs often **requires complete redesign of the algorithms**.
- We have achieved promising results: up to **20x faster solutions with moderate accuracy**.
- Chellenges remain: **Ill-conditioning of condensed KKT system** and **portability**.

Conclusion

- Optimization on GPUs is a growing area (LPs, QPs, domain-specific problems)!
- **GPU hardware** offers significant potential for **accelerating large-scale optimization**.
- Porting algorithms on GPUs often **requires complete redesign of the algorithms**.
- We have achieved promising results: up to **20x faster solutions with moderate accuracy**.
- Chellenges remain: **Ill-conditioning of condensed KKT system** and **portability**.
- We envision **expanding the application scope of nonlinear programming**.
  - **Extremely large-scale** problems (coupled infrastructures, multi-stage, multiscale).
  - Problems involving **expensive surrogate models** (neural nets, simulations).