Enhancing nonlinear optimization through GPU computing

Alexis Montoison[1], **Sungho Shin**[2], **François Pacaud**[3], **Mihai Anitescu**[2,4], **and Exanauts Team**[⋆]

[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
[⋆]exanauts.github.io

alexis.montoison@polymtl.ca

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

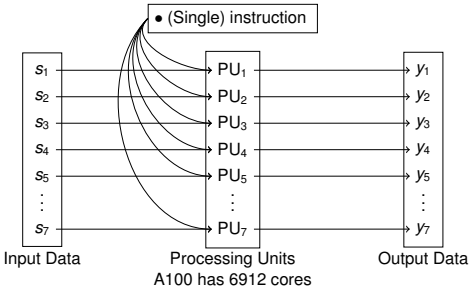
$=$ 1 million $\times$

iPhone 14 Pro

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

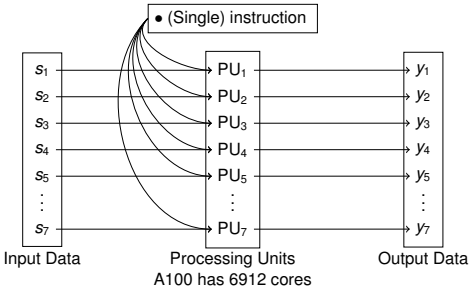
How Do GPUs Work? (or, how are they different from CPUs?)

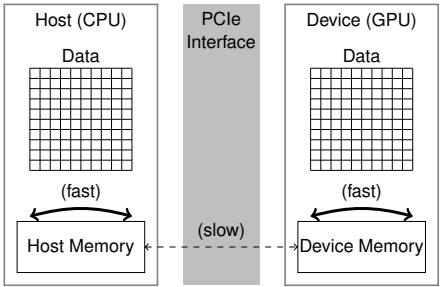
- Single Instruction, Multiple Data (**SIMD**) parallelism,



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

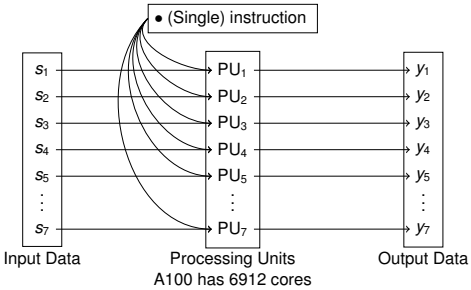
- Single Instruction, Multiple Data (**SIMD**) parallelism, on **(dedicated) device memory**.

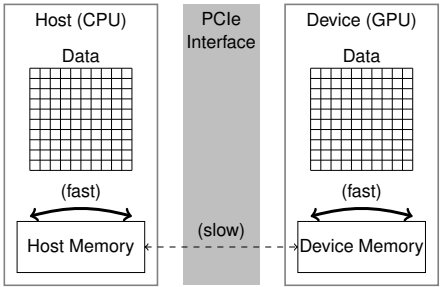




How Do GPUs Work? (or, how are they different from CPUs?)

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

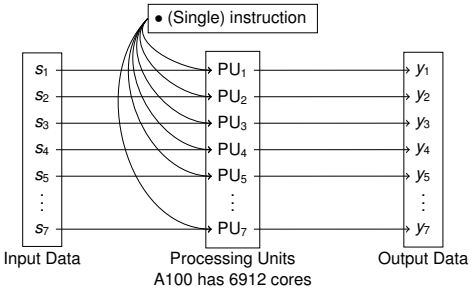


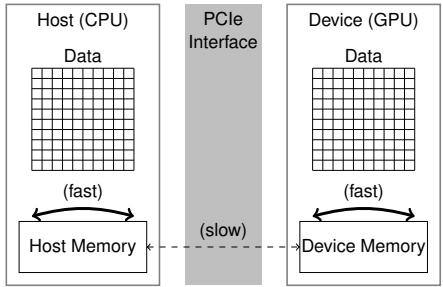


- All data **should reside exclusively on device memory**, and all operation **should be executed by GPU only**.

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

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

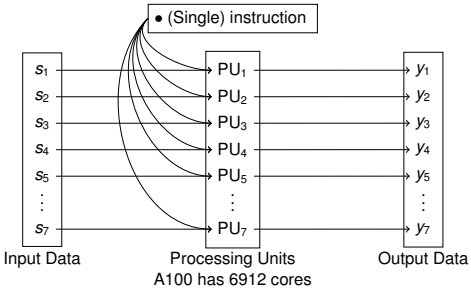


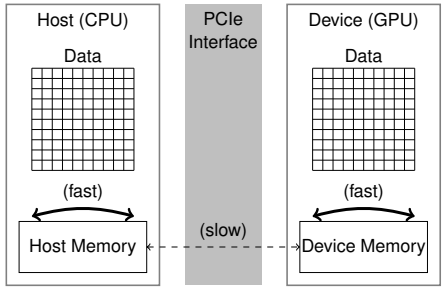


- All data **should reside exclusively on device memory**, and all operation **should be executed by GPU only**.
- Designing GPU algorithms sometimes require a **complete redesign** of the algorithm.

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

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





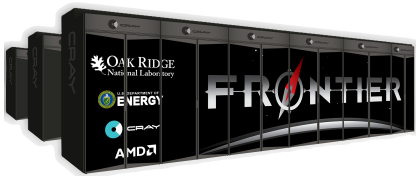
- All data **should reside exclusively on device memory**, and all operation **should be executed by GPU only**.
- Designing GPU algorithms sometimes require a **complete redesign** of the algorithm.

Adapting CPU code into GPU code is **not merely a matter of software engineering**.

Exascale Computing Project

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

Frontier @Oak Ridge

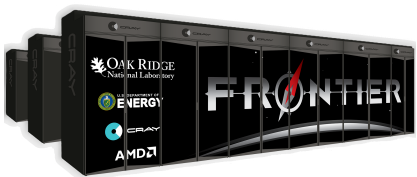


Aurora @Argonne

Exascale Computing Project

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

Frontier @Oak Ridge



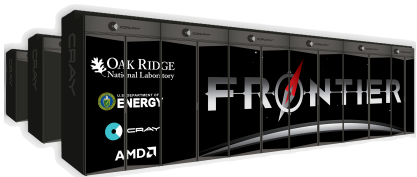
Aurora @Argonne

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

Exascale Computing Project

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

Frontier @Oak Ridge (AMD GPUs)



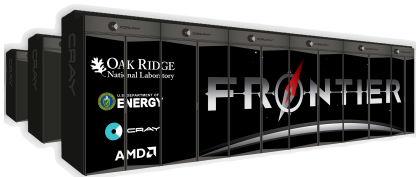
Aurora @Argonne (Intel GPUs)

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

Exascale Computing Project

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

Frontier @Oak Ridge (AMD GPUs)



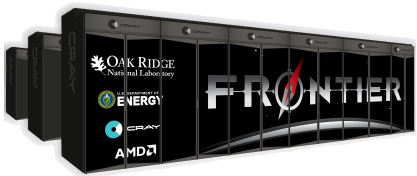
Aurora @Argonne (Intel GPUs)

- **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).

Exascale Computing Project

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

Frontier @Oak Ridge (AMD GPUs)



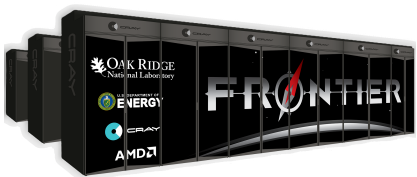
Aurora @Argonne (Intel GPUs)

- **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.

Exascale Computing Project

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

Frontier @Oak Ridge (AMD GPUs)



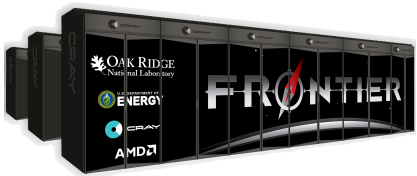
Aurora @Argonne (Intel GPUs)

- **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)

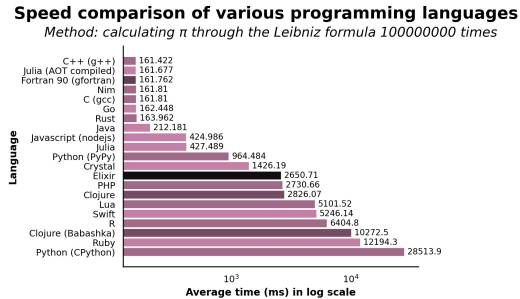


Aurora @Argonne (Intel GPUs)

- **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).

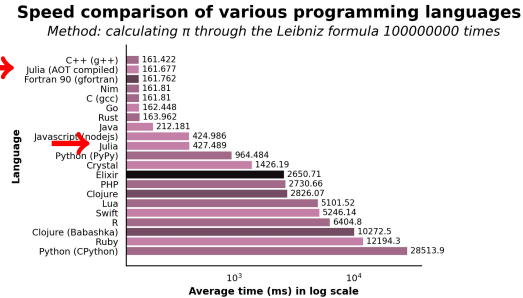
Language of Choice: Julia

- Runs **as fast as C/C++/Fortran**,



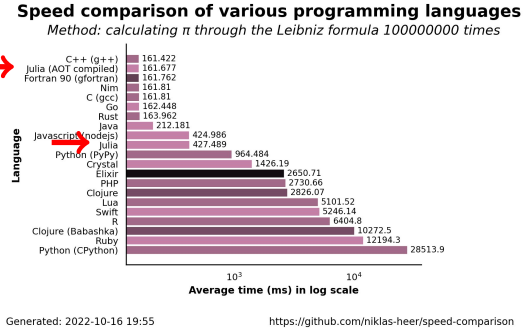
Language of Choice: Julia

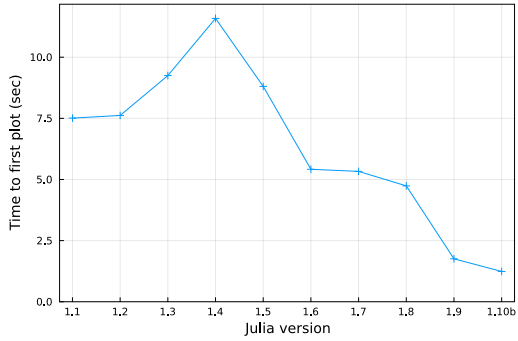
- Runs **as fast as C/C++/Fortran**,



Language of Choice: Julia

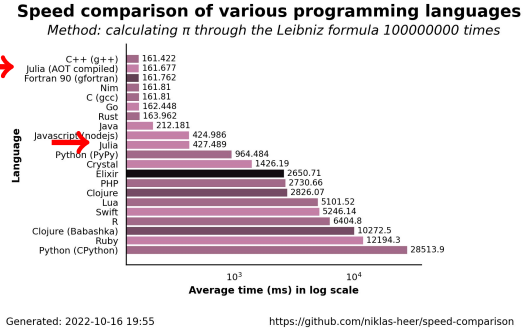
- Runs **as fast as C/C++/Fortran**, and "**Time-To-First-Call**" has been significantly improved.

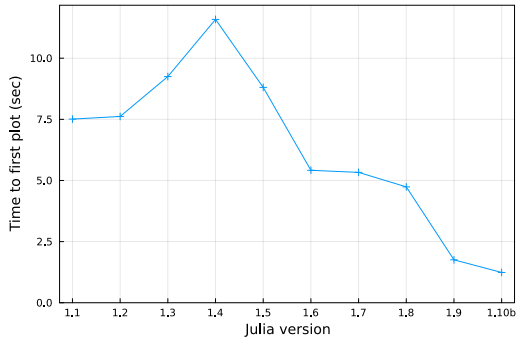




Language of Choice: Julia

- Runs **as fast as C/C++/Fortran**, and "**Time-To-First-Call**" has been significantly improved.





- **Fast development** (like Python, R, Matlab): Julia resolves the "**two-language problem**".

Language of Choice: Julia

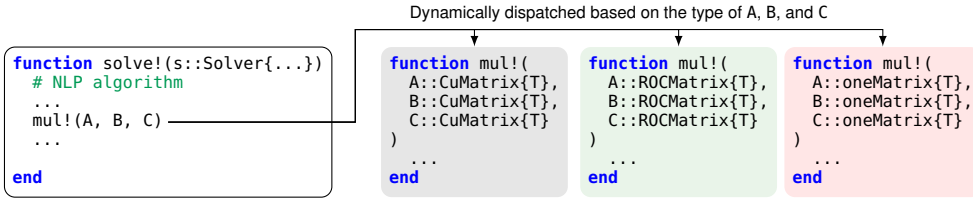
- **Multiple dispatch**: High-level abstraction while specializing for specific data types.

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

end
```

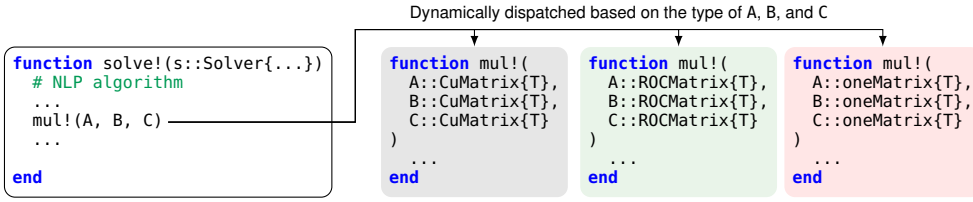
Language of Choice: Julia

- **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

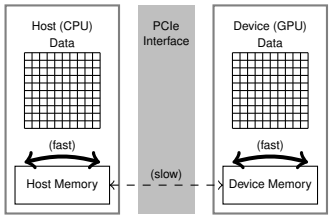
GPU/HPC

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

Summary

GPU/HPC

SIMD Architecture

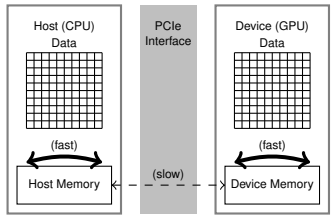


- **Motivation**: **Harness accelerated computing** for **nonlinear programming**.
- Adapting CPU code into GPU code is **not merely an issue of software engineering**.

Summary

GPU/HPC

SIMD Architecture



Performance, Fast Development, & Portability

- **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

CUDA.jl

- CUDA.jl is a Julia library that enables developers to harness the parallel computing capabilities offered by NVIDIA GPUs.
- CUDA.jl provides a native Julia interface for programming GPU kernels with Julia wrappers.
- CUDA.jl functions can be called directly from Julia code, making development and maintenance of code easier.
- CUDA.jl generates efficient and optimized CUDA code, fully leveraging the computing power of NVIDIA GPUs.

CUDSS.jl

NVIDIA cuDSS provides three factorizations (LDU, LDL^T, LL^T) for solving sparse linear systems.

cuDSS follows a well-established three phases approach commonly used in sparse direct solvers:

- reordering and symbolic factorization;
- numerical factorization;
- solve linear system using the computed factors.

This modular approach allows cuDSS to efficiently handle sparse linear systems by reusing the analysis and factorization stages, facilitating the solution of the KKT system in interior point methods.

Outline

Nonlinear Optimization Software: State-of-the-Art on CPU

Problem Formulation

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

Nonlinear Optimization Software: 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 Software: 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 Software: 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 Software: 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

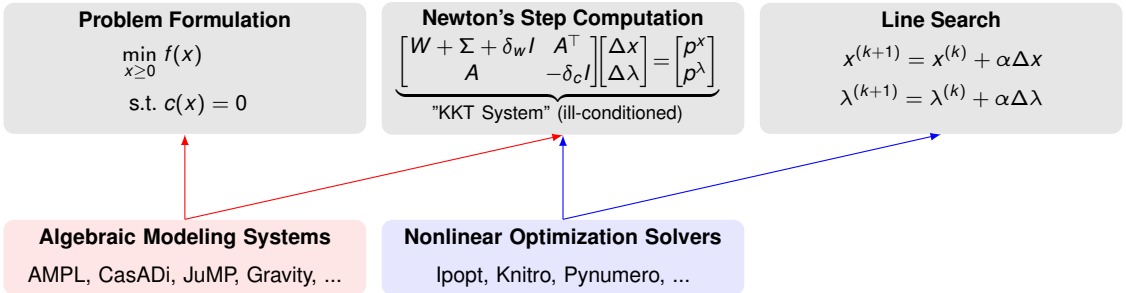
$$x^{(k+1)} = x^{(k)} + \alpha \Delta x$$
$$\lambda^{(k+1)} = \lambda^{(k)} + \alpha \Delta \lambda$$

Algebraic Modeling Systems

AMPL, CasADi, JuMP, Gravity, ...

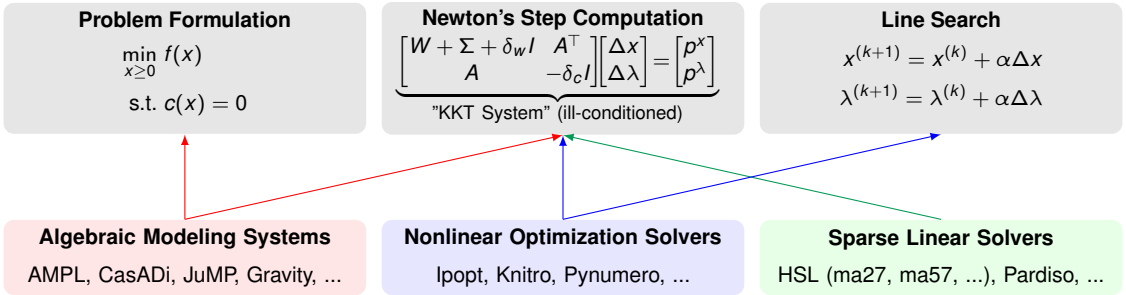
- Algebraic modeling systems provides **front-end** to specify models and (often) provides **derivative computation capabilities**.

Nonlinear Optimization Software: State-of-the-Art on CPU



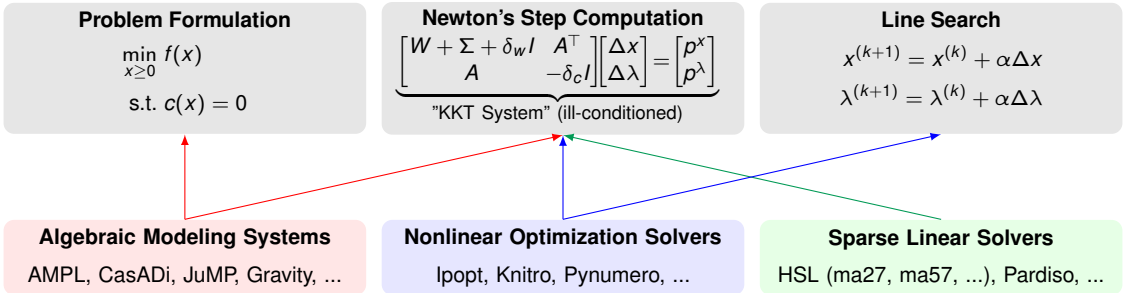
- Algebraic modeling systems provides **front-end** to specify models and (often) provides **derivative computation capabilities**.
- Nonlinear optimization solvers apply iterations of optimization algorithms.

Nonlinear Optimization Software: State-of-the-Art on CPU



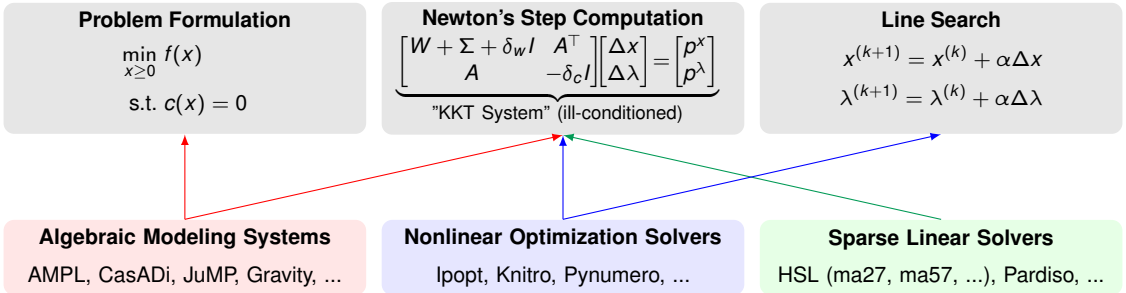
- Algebraic modeling systems provides **front-end** to specify models and (often) provides **derivative computation capabilities**.
- Nonlinear optimization solvers apply iterations of optimization algorithms.
- Sparse linear solvers solves KKT systems using **sparse matrix factorization**.

Nonlinear Optimization Software: State-of-the-Art on CPU



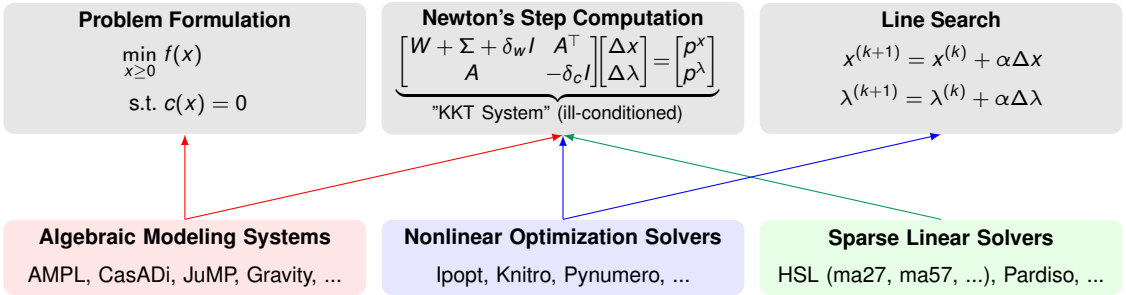
- These software tools have enabled the success of nonlinear optimization on CPUs.

Nonlinear Optimization Software: 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 Software: State-of-the-Art on CPU

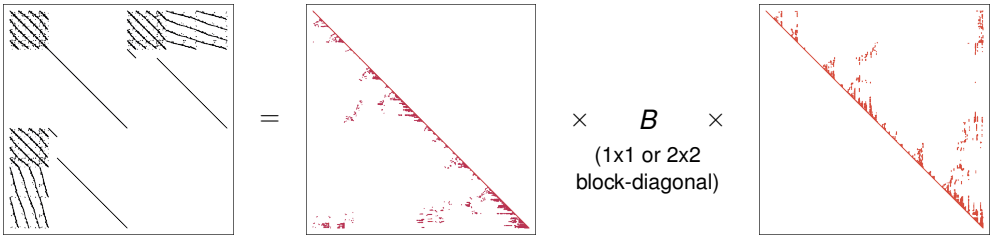


- These software tools have enabled the success of nonlinear optimization on CPUs.
- Many software tools have been developed in 80s-90s (**heavily optimized for CPUs**).
- Now we need **GPU-equivalent** of these tools.

Outline

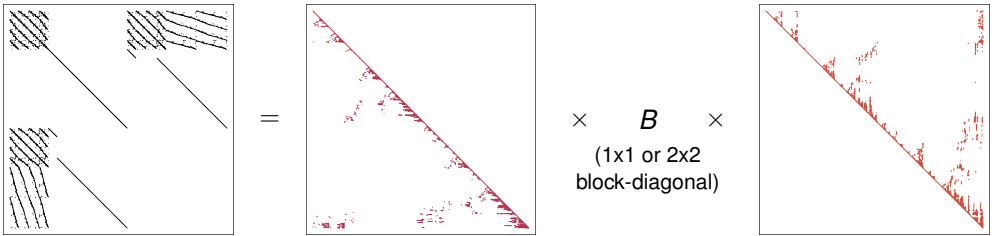
Why it is Challenging: Sparse Linear Solvers

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



Why it is Challenging: Sparse Linear Solvers

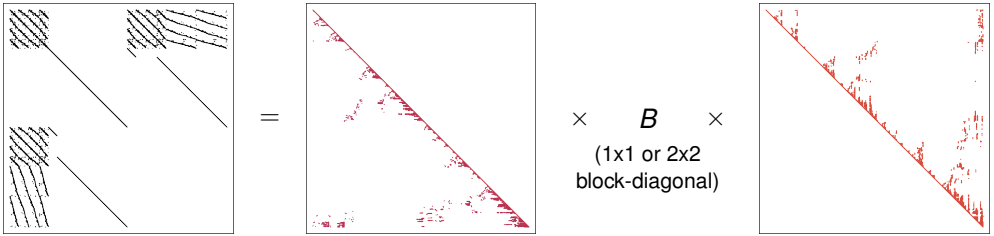
- Solving KKT systems on CPUs has traditionally relied on **direct LBL$^\top$ factorization**.



- LBL$^\top$ factorization requires **numerical pivoting**, which is **challenging to parallelize**.

Why it is Challenging: Sparse Linear Solvers

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

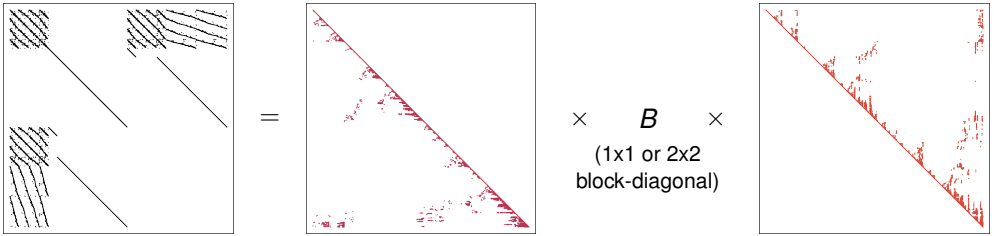


- LBL$^\top$ factorization requires **numerical pivoting**, which is **challenging to parallelize**.
- Then, how about **iterative solvers** (e.g., GMRES)?

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)

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**.
- Then, how about **iterative solvers** (e.g., GMRES)? Due to ill-conditioning of the KKT system, iterative methods are generally not effective, unless **specialized preconditioners** are used.

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)

Solution #1: Condensed-Space Interior Point Method

- To avoid **numerical pivoting**, we transform the KKT systems into **positive definite systems**.

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

Solution #1: Condensed-Space Interior Point Method

- To avoid **numerical pivoting**, we transform the KKT systems into **positive definite systems**.
- Cholesky factorization can be computed with static pivoting, and available in CUDA.

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

Solution #1: Condensed-Space Interior Point Method

- To avoid **numerical pivoting**, we transform the KKT systems into **positive definite systems**.
- Cholesky factorization can be computed with static pivoting, and available in CUDA.
- Can be achieved by (i) converting the **equalities into inequalities**:

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

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

Solution #1: Condensed-Space Interior Point Method

- To avoid **numerical pivoting**, we transform the KKT systems into **positive definite systems**.
- Cholesky factorization can be computed with static pivoting, and available in CUDA.
- 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*):

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

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

Solution #1: Condensed-Space Interior Point Method

- To avoid **numerical pivoting**, we transform the KKT systems into **positive definite systems**.
- Cholesky factorization can be computed with static pivoting, and available in CUDA.
- 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*):

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

Invertible

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

Solution #1: Condensed-Space Interior Point Method

- To avoid **numerical pivoting**, we transform the KKT systems into **positive definite systems**.
- Cholesky factorization can be computed with static pivoting, and available in CUDA.
- 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*):

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

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

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

Solution #1: Condensed-Space Interior Point Method

- To avoid **numerical pivoting**, we transform the KKT systems into **positive definite systems**.
- Cholesky factorization can be computed with static pivoting, and available in CUDA.
- 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*):

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

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

- The relaxation and condensation may cause **ill-conditioning** in the KKT system.

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

Highlight: AC Optimal Power Flow (single GPU)

MadNLP

- Standard polar form AC optimal power flow (AC OPF) problems.

Highlight: AC Optimal Power Flow (single GPU)

MadNLP

- Standard polar form AC optimal power flow (AC OPF) problems.
- For large-scale cases, GPU becomes **significantly faster than CPU** (up to $\times$20 speedup).

Case	nvars	ncons	MadNLP+ExaModels+cuDSS (GPU)				Ipopt+ExaModels+Ma27 (CPU)			
			iter	deriv.†	lin.†	total†	iter	deriv.†	lin.†	total†
89_pegase	1.0k	1.6k	28	0.02	0.03	0.15	31	0.01	0.06	0.06
179_goc	1.5k	2.2k	35	0.03	0.07	0.24	46	0.01	0.10	0.11
500_goc	4.3k	6.1k	40	0.05	0.11	0.40	40	0.02	0.21	0.23
793_goc	5.4k	8.0k	39	0.03	0.06	0.28	37	0.03	0.25	0.28
1354_pegase	11.2k	16.6k	55	0.06	0.23	0.62	49	0.07	0.69	0.76
2312_goc	17.1k	25.7k	47	0.04	0.24	0.70	46	0.10	1.12	1.22
2000_goc	19.0k	29.4k	42	0.04	0.12	0.53	43	0.13	1.26	1.38
3022_goc	23.2k	35.0k	53	0.05	0.33	0.90	58	0.18	1.93	2.11
2742_goc	24.5k	38.2k	231	0.28	0.59	2.92	106	0.49	5.63	6.13
2869_pegase	25.1k	37.8k	61	0.06	0.14	0.73	59	0.21	2.22	2.43
3970_goc	35.3k	54.4k	124i	0.13	1.93	3.73	66	0.36	4.87	5.24
4020_goc	36.7k	57.0k	60	0.06	0.42	1.35	60	0.35	6.82	7.17
4917_goc	37.9k	56.9k	59	0.06	0.41	1.15	65	0.34	3.98	4.32
4601_goc	38.8k	59.6k	82	0.08	0.69	1.81	73	0.44	5.84	6.28
4837_goc	41.4k	64.0k	56	0.06	0.30	1.14	59	0.39	4.78	5.17
4619_goc	42.5k	66.3k	53	0.07	0.19	1.20	50	0.34	5.93	6.27
5658_epigrids	48.6k	74.8k	49	0.05	0.31	1.17	51	0.41	5.46	5.87
7336_epigrids	62.1k	95.3k	55	0.06	0.38	1.46	50	0.49	7.08	7.57
10000_goc	76.8k	112.4k	65	0.08	0.48	3.64	84	0.96	12.51	13.47
8387_pegase	78.7k	118.7k	110a	0.13	0.47	2.72	79	0.98	12.01	12.99
9591_goc	83.6k	130.6k	67	0.09	0.95	2.85	70	0.96	21.24	22.20
9241_pegase	85.6k	130.8k	500f	1.57	4.17	13.26	73	0.93	12.96	13.89
10192_epigrids	89.8k	139.5k	58	0.10	0.79	2.55	59	0.99	16.38	17.37
10480_goc	96.8k	150.9k	68	0.09	0.61	2.71	70	1.02	22.63	23.65
13659_pegase	117.4k	170.6k	77	0.11	0.79	3.01	66	1.14	16.06	17.20
20758_epigrids	179.2k	274.9k	200a	0.44	5.24	12.59	51	1.63	29.84	31.47
19402_goc	179.6k	281.7k	72	0.14	0.75	4.42	72	2.26	60.80	63.06
24464_goc	203.4k	313.6k	65	0.14	0.90	4.41	63	2.11	41.28	43.39
30000_goc	208.6k	307.8k	179	0.43	3.65	9.57	153	6.17	93.02	99.20
78484_epigrids	674.6k	1.0m	105	0.61	4.37	20.38	104	21.50	345.93	367.43

Outline

Future Outlook: Towards 10^{-8} Precision

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

Future Outlook: Towards 10^{-8} Precision

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

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

Future Outlook: Towards 10^{-8} Precision

- 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

Future Outlook: Towards 10^{-8} Precision

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

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

$J^\top\Sigma_s^{-1}J$ → Causes ill-conditioning

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

Future Outlook: Towards 10^{-8} Precision

- 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, augmented Lagrangian, and hybrid KKT, **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).

Future Outlook: Towards 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.

Future Outlook: Towards 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**,

Future Outlook: Towards 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.

Future Outlook: Towards 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 sparse (Cholesky or LU) linear solvers.

Future Outlook: Towards 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 sparse (Cholesky or LU) linear solvers.
- Work-arounds?
  - Preconditioned iterative solver with reduction.
  - Domain-specific linear solvers (e.g., Riccati solver for optimal control).

Krylov.jl: A Julia Basket of Hand-Picked Krylov Methods `https://github.com/JuliaSmoothOptimizers/Krylov.jl`
Cole et. al., Exploiting GPU/SIMD Architectures for Solving Linear-Quadratic MPC Problems, ACC (2023).

Summary

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

Summary

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

Summary

- 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**.

Summary

- 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**.

Summary

- 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**.

Summary

- 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).