Recent Advances in the OSQP Solver:

Differentiable Optimization, Accelerated Linear Algebra, and More

Ian McInerney - Imperial College London/The University of Manchester
Vineet Bansal – Princeton University, CSML
Paul Goulart – The University of Oxford
Bartolomeo Stellato – Princeton University, ORFE

With Goran Banjac and Rajiv Sambharya

SIAM Conference on Optimization, May 31, 2023

Contributors

Ian McInerney

Vineet Bansal

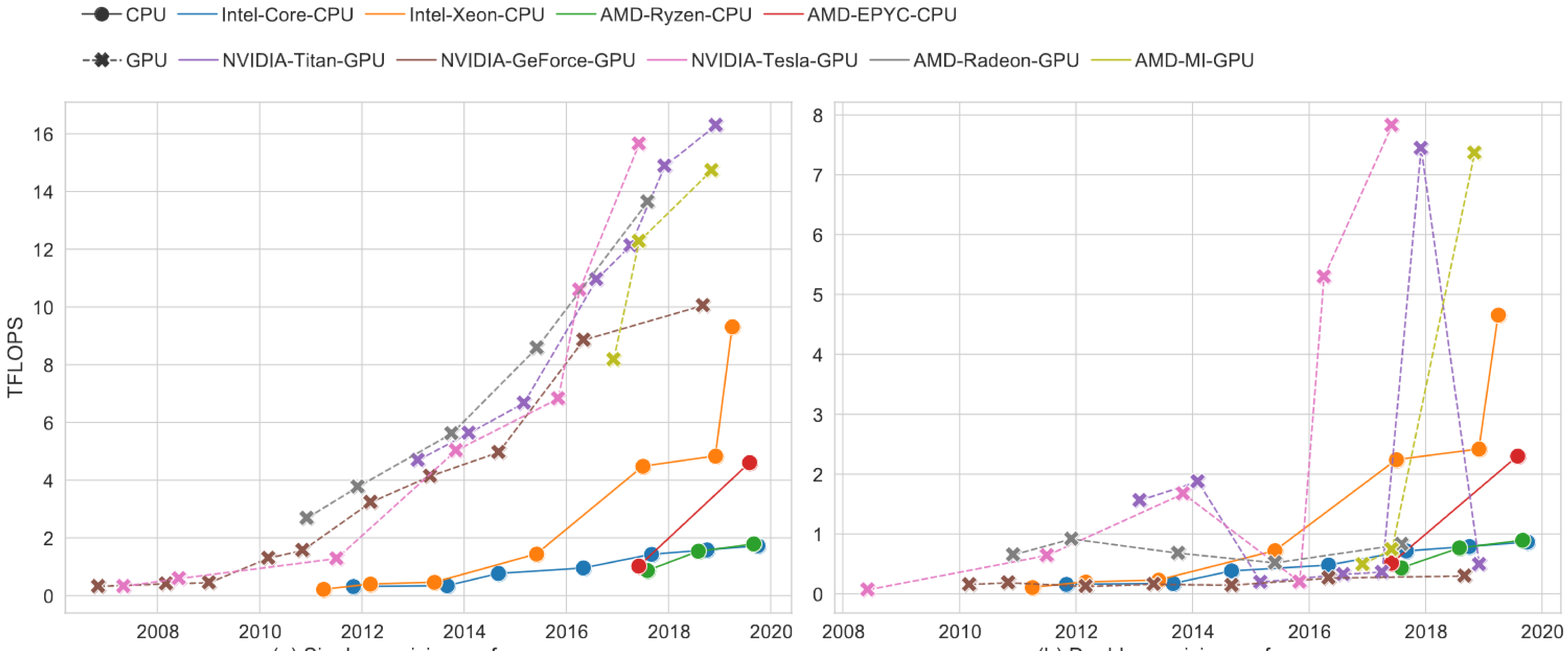
Bartolomeo Stellato

Paul Goulart

Goran Banjac

Rajiv Sambharya

Tremendous progress in compute



(a) Single-precision performance.

(b) Double-precision performance.

[Y. Sun, N. B. Agostini, S. Dong, and D. Kaeli, “Summarizing CPU and GPU Design Trends with Product Data”, 2020, arXiv:1911.11313v2]

123.74
145
140
135
130
125
Pct. chg
Day
Chg.
1.64
1.98
Tues.
142.78

First-order methods

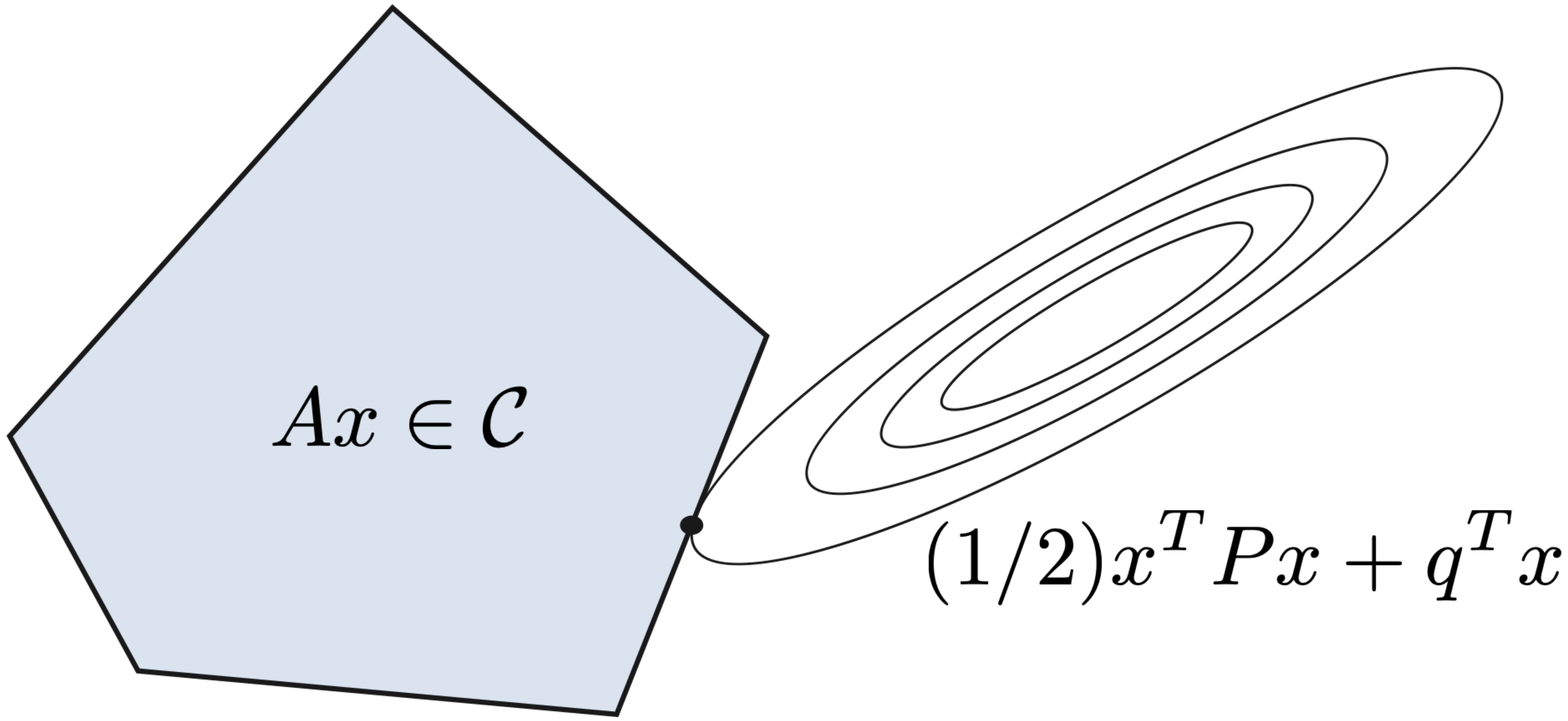
Wide popularity

Pros	Cons	OSQP
Warm-starting	Low quality solutions	High-quality solutions
Large-scale problems	Can’t detect infeasibility	Detects infeasibility
Embeddable	Problem data dependent	Robust

The problem

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Ax \in \mathcal{C} \end{array}$$

Quadratic program: $\mathcal{C} = [l, u]$



ADMM

Alternating Direction Method of Multipliers

Splitting

minimize $\quad f(x) + g(x) \quad \longrightarrow$

$$\begin{array}{ll} \text{minimize} & f(\tilde{x}) + g(x) \\ \text{subject to} & \tilde{x} = x \end{array}$$

Iterations

$$\tilde{x}^{k+1} \leftarrow \underset{\tilde{x}}{\text{argmin}} \left(f(\tilde{x}) + \rho/2 \left\| \tilde{x} - (x^k - y^k/\rho) \right\|^2 \right)$$

$$x^{k+1} \leftarrow \underset{x}{\text{argmin}} \left(g(x) + \rho/2 \left\| x - (\tilde{x}^{k+1} + y^k/\rho) \right\|^2 \right)$$

$$y^{k+1} \leftarrow y^k + \rho \left(\tilde{x}^{k+1} - x^{k+1} \right)$$

How do we split the QP?

$$\begin{array}{lll} \text{minimize} & (1/2)x^T P x + q^T x & \\ \text{subject to} & Ax = z & f \\ & z \in \mathcal{C} & g \end{array}$$

Splitting formulation

$$\begin{array}{ll} \text{minimize} & \underbrace{(1/2)\tilde{x}^T P \tilde{x} + q^T \tilde{x} + \mathcal{I}_{Ax=z}(\tilde{x}, \tilde{z})}_{f} + \underbrace{\mathcal{I}_{\mathcal{C}}(z)}_{g} \\ \text{subject to} & \tilde{x} = x \\ & \tilde{z} = z \end{array}$$

Diagonal
step sizes

σ

ρ

Complete algorithm

Problem

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & l \leq Ax \leq u \end{array}$$

Algorithm

Linear system solve

$$(x^{k+1}, \nu^{k+1}) \leftarrow \text{solve} \begin{bmatrix} P + \sigma I & A^T \\ A & -\frac{1}{\rho} I \end{bmatrix} \begin{bmatrix} x^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \sigma x^k - q \\ z^k - \frac{1}{\rho} y^k \end{bmatrix}$$

Easy operations

$$\tilde{z}^{k+1} \leftarrow z^k + (\nu^{k+1} - y^k)/\rho$$

$$z^{k+1} \leftarrow \Pi\left(\tilde{z}^{k+1} + y^k/\rho\right)$$

$$y^{k+1} \leftarrow y^k + \rho\left(\tilde{z}^{k+1} - z^{k+1}\right)$$

Solving the linear system

Direct method (small to medium scale)

Quasi-definite matrix

$$\begin{bmatrix} P + \sigma I & A^T \\ A & -\frac{1}{\rho} I \end{bmatrix} \begin{bmatrix} x \\ \nu \end{bmatrix} = \begin{bmatrix} \sigma x^k - q \\ z^k - \frac{1}{\rho} y^k \end{bmatrix}$$

Well-defined LDL^T factorization

Factorization caching

$\longrightarrow$ **QDLDL**
Free quasi-definite linear system solver

[https://github.com/osqp/qdldl]

Solving the linear system

Indirect method (large scale)

Positive-definite matrix

$$\left(P + \sigma I + \rho A^T A\right) x = \sigma x^k - q + A^T(\rho z^k - y^k)$$

Conjugate gradient

Solve very large systems

GPU & FPGA implementation

Complete algorithm

Problem

minimize $\quad (1/2)x^T P x + q^T x$

subject to $\quad l \leq Ax \leq u$

Algorithm

Linear system solve

$$x^{k+1} \leftarrow \text{Solve} \quad (P + \sigma + \rho A^T A)x = \sigma x^k - q + A^T(\rho z^k - y^k)$$

Easy operations

$$z^{k+1} \leftarrow \Pi(Ax^{k+1} + \rho^{-1} y^k)$$

$$y^{k+1} \leftarrow y^k + \rho(Ax^{k+1} - z^{k+1})$$

always solvable!

OSQP

Operator Splitting solver for Quadratic Programs

Embeddable (can be division free!)

Supports warm-starting

Detects infeasibility

Solves large-scale problems

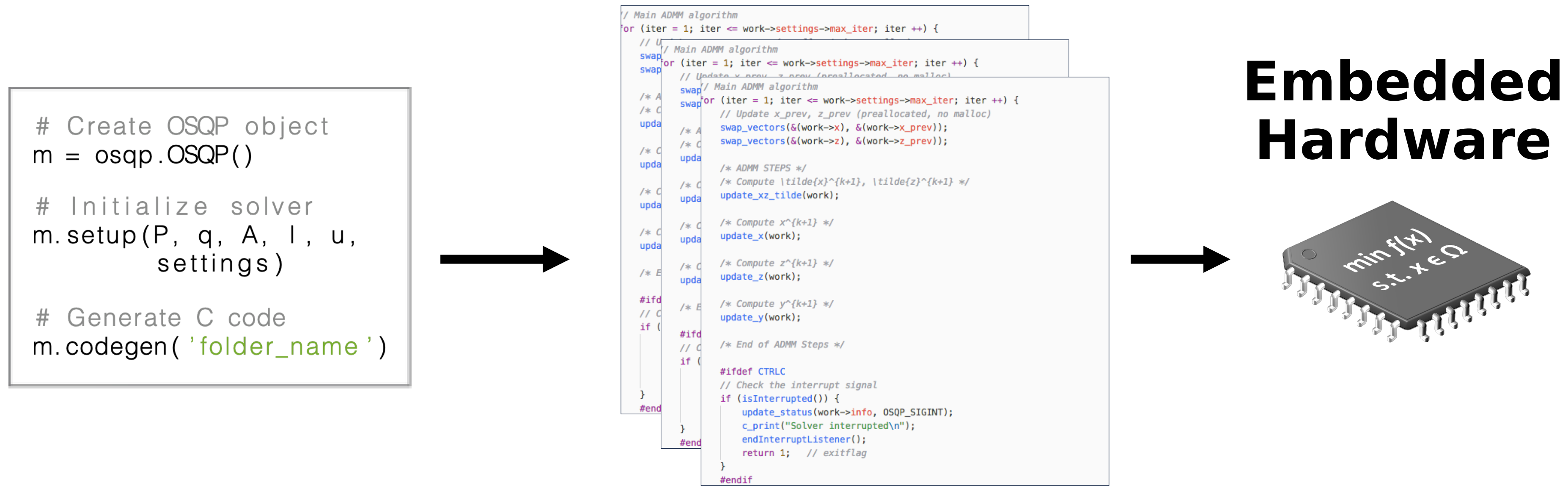
Users

More than 18 million downloads!

[pepy.tech/project/osqp]

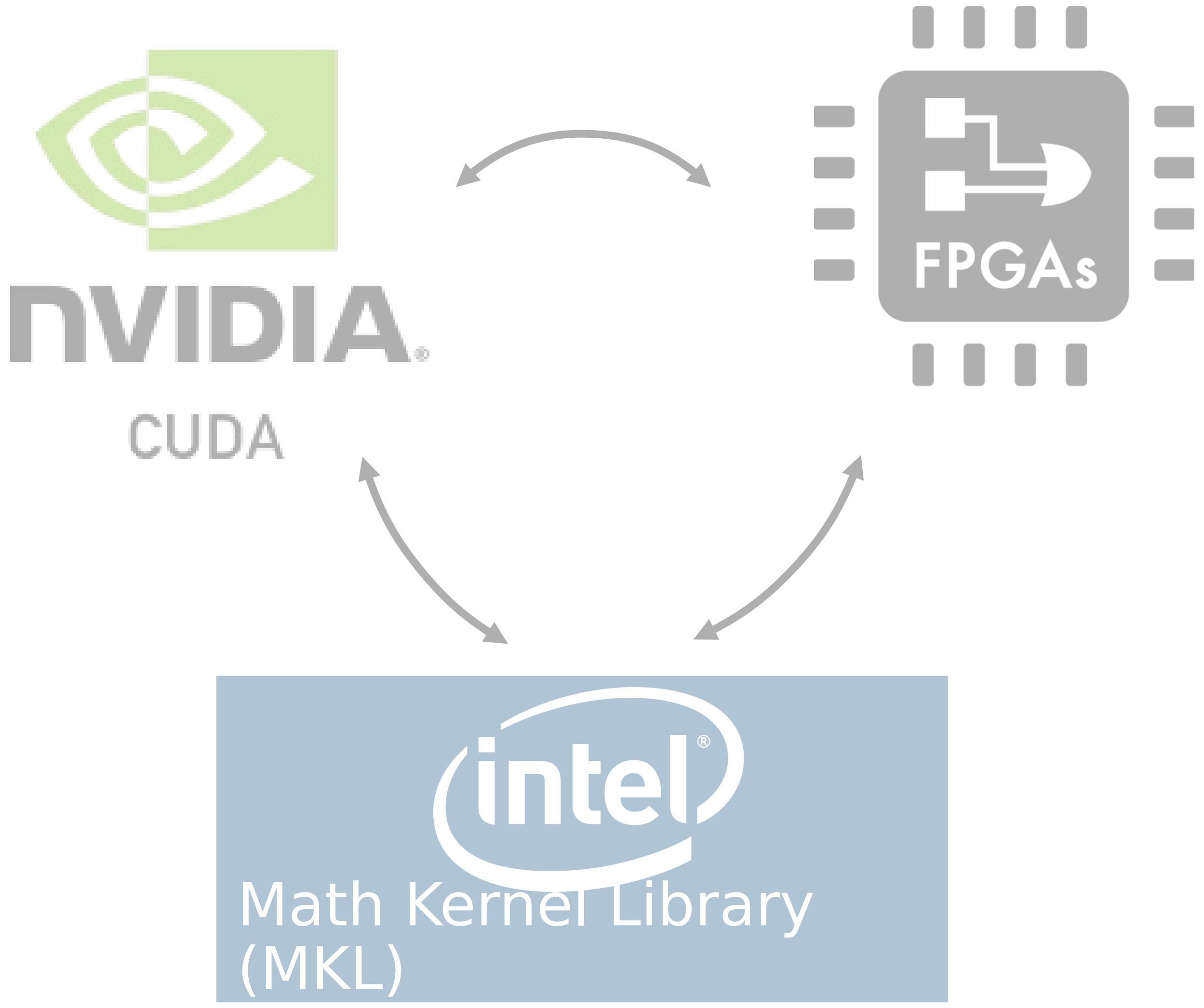
What’s new in OSQP 1.0

Improved embedded code generation



Code generation from C to C

Modular linear algebra



Differentiable layers

Code generation — Python API and result

Python API calls C code generation

```python
# Create an OSQP object
prob = osqp.OSQP()

# Setup workspace and change alpha parameter
prob.setup(P, q, A, l, u, alpha=1.0)

# Generate C code
prob.codegen(
    'folder',                   # Output folder for auto-generated code
    prefix='mysolver_',         # Prefix for filenames and C variables

    parameters='vectors',       # What do we wish to update in the generated code?
                                # 'vectors'/'matrices'
    use_float=False,            # Use single precision in generated code?
    printing_enable=False,      # Enable solver printing?
    profiling_enable=False,     # Enable solver profiling?
    interrupt_enable=False,     # Enable user interrupt (Ctrl-C)?
    include_codegen_src=True,   # Include headers/sources/Makefile in the output folder,
                                # creating a self-contained compilable folder?

    compile=False,              # Compile the python wrapper?
    python_ext_name='pyosqp',   # Name of the generated python extension
)
```

Naming

Solver options

Codegen wrapper

Code generation from C to C

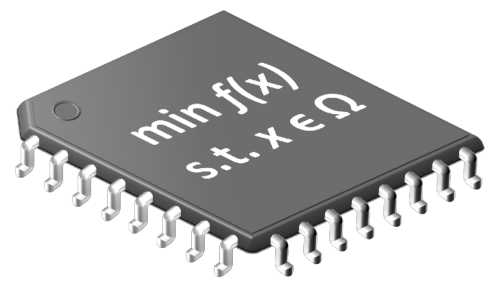
This is what gets called…

```
exitflag = osqp_setup(&solver, P, q, A, l, u, m, n, settings);

/* Test codegen */
OSQPCodegenDefines *defs = (OSQPCodegenDefines *)malloc(sizeof(OSQPCodegenDefines));

defs->float_type = 0;         /* Use doubles */
defs->printing_enable = 0;    /* Don't enable printing */
defs->profiling_enable = 0;   /* Don't enable profiling */
defs->interrupt_enable = 0;   /* Don't enable interrupts */

/* Generate code that allows only vector updates */
defs->embedded_mode = 1;
osqp_codegen(solver, vecDirPath, "vec_prefix_", defs);

/* Generate code that allows both vector and matrix updates */
defs->embedded_mode = 2;
osqp_codegen(solver, matDirPath, "mat_prefix", defs);
```

Desktop C solver

Embeddable code

- No dynamic memory allocation
- Division-free



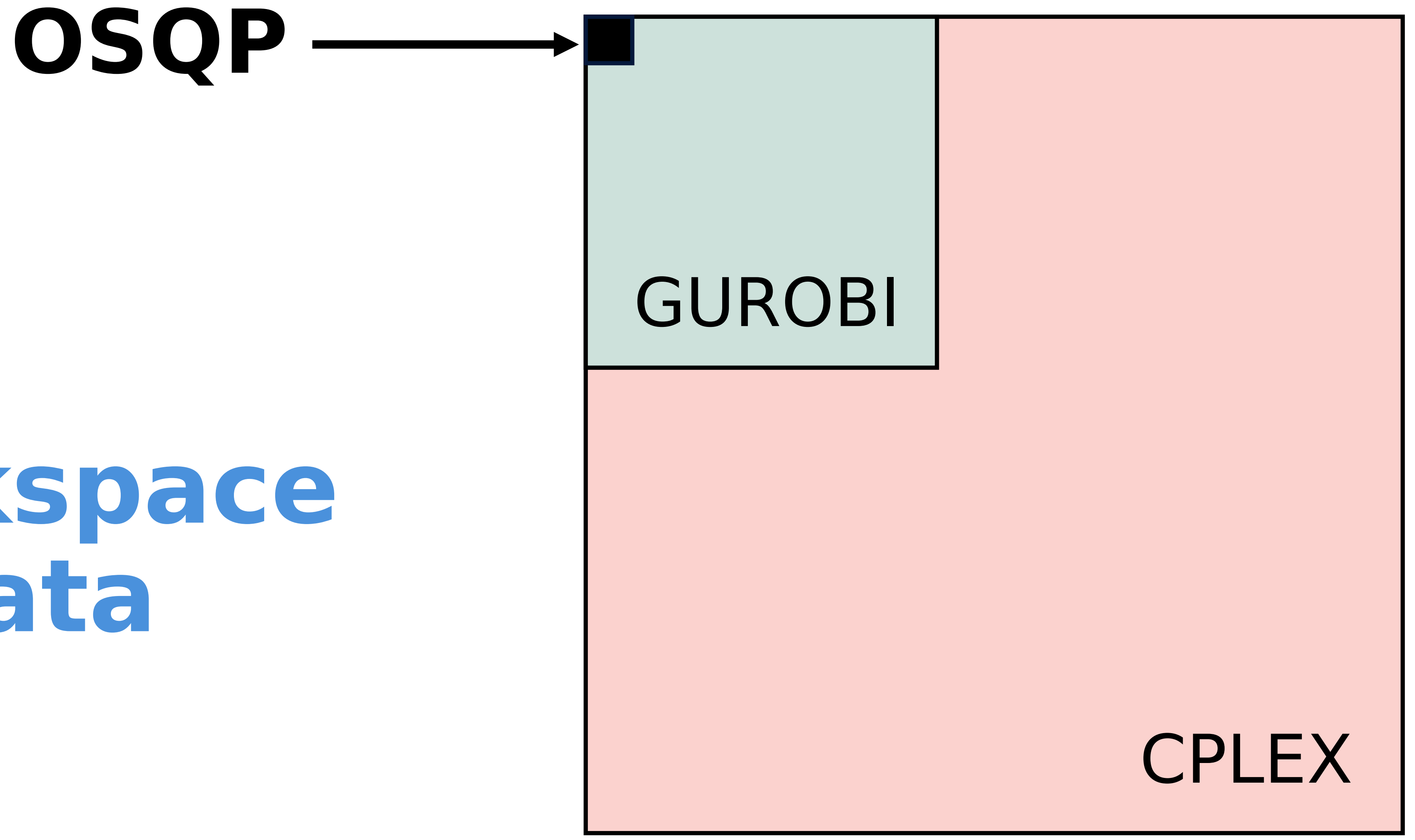
Code generation results

Self-contained and simplified directory structure

```
$ tree out
out
├── emosqp.c
├── inc
│   ├── private
│   │   ├── algebra_impl.h
⋮   ⋮   ⋮
│   │   └── version.h
│   └── public
│       ├── csc_type.h
│       ├── osqp_api_constants.h
│       ├── osqp_api_functions.h
│       ├── osqp_api_types.h
│       ├── osqp_api_utils.h
│       ├── osqp_export_define.h
│       └── osqp.h
├── Makefile
├── osqp_configure.h
├── mysolver_workspace.c
├── mysolver_workspace.h
└── src
    ├── algebra_libs.c
    ⋮
    ├── osqp_api.c
    ⋮
    └── vector.c
```

Workspace data

Compiled code size ~80kb (low footprint)

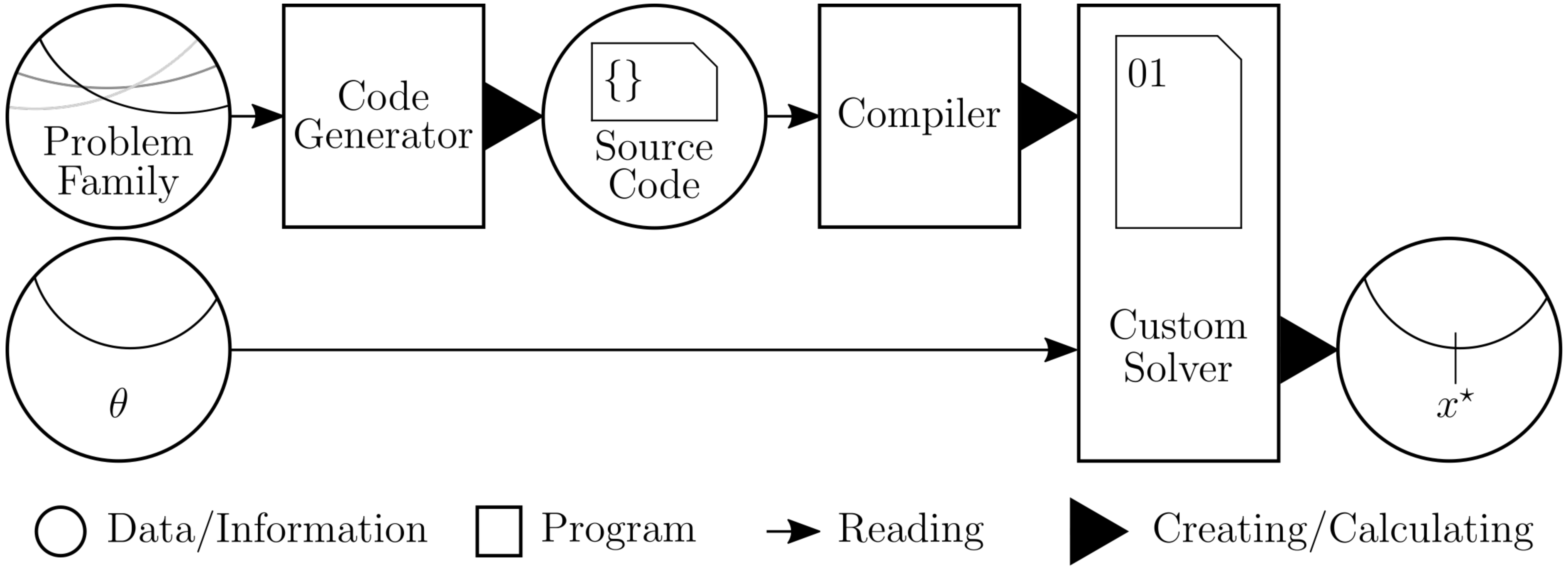


300x Reduction !

Code generation for parametric convex optimization

OSQP is integrated in CVXPYgen

https://pypi.org/project/cvxpygen/



Example

minimize $\|Gx - h\|^2$

subject to $x \geq 0$

$$\theta = (G, h)$$

```python
import cvxpy as cp
from cvxpygen import cpg

# model problem
x = cp.Variable(n, name='x')
G = cp.Parameter((m,n), name='G')
h = cp.Parameter(m, name='h')
p = cp.Problem(cp.Minimize(cp.sum_squares(G@x-h)),
               [x>=0])
# generate code
cpg.generate_code(p)
```

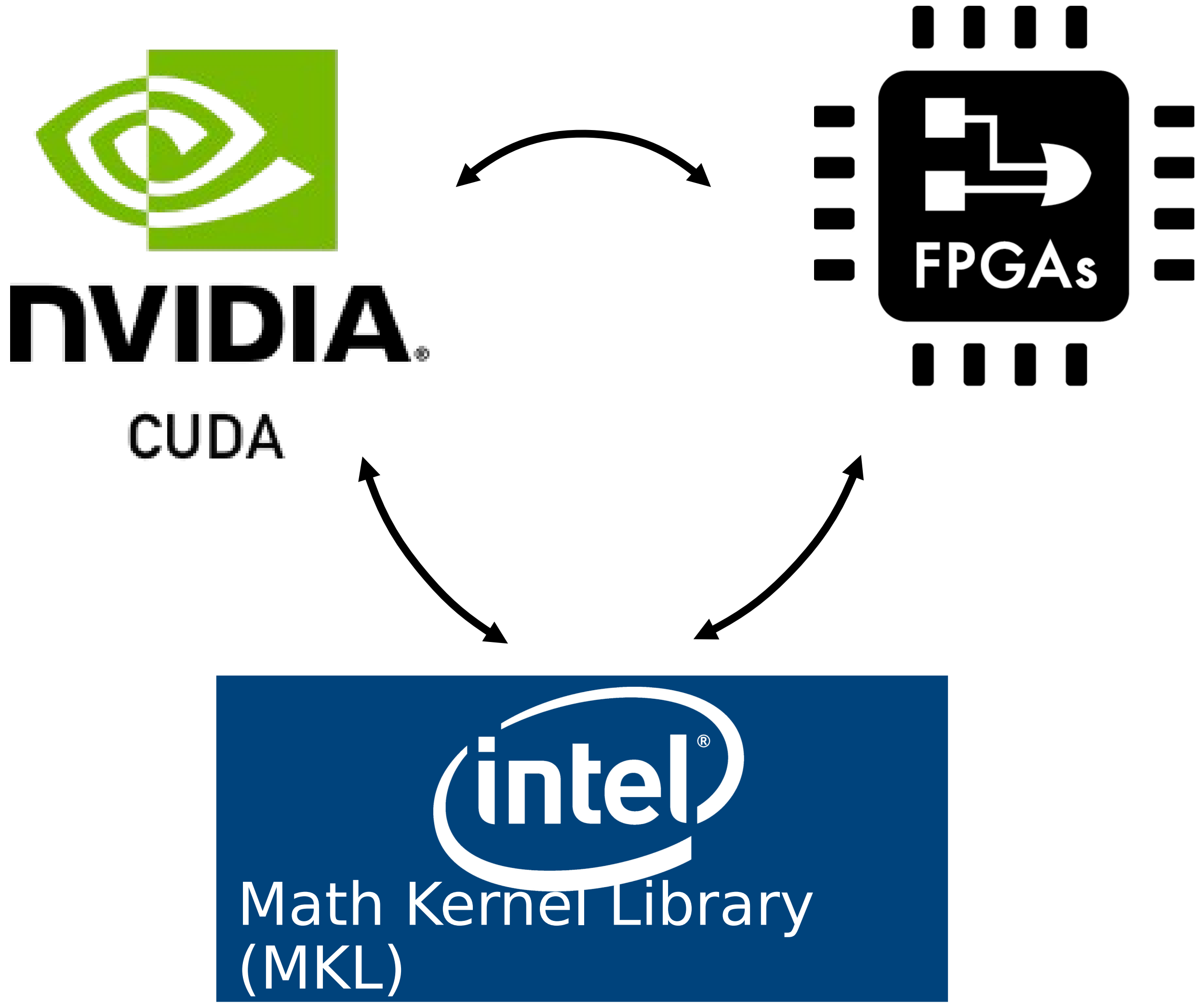
[Embedded Code Generation with CVXPY. Schaller, Banjac, Diamond, Agrawal, Stellato, and Boyd IEEE Control Systems Letters 2022]

What’s new in OSQP 1.0

Improved embedded code generation

Code generation from C to C

Modular linear algebra

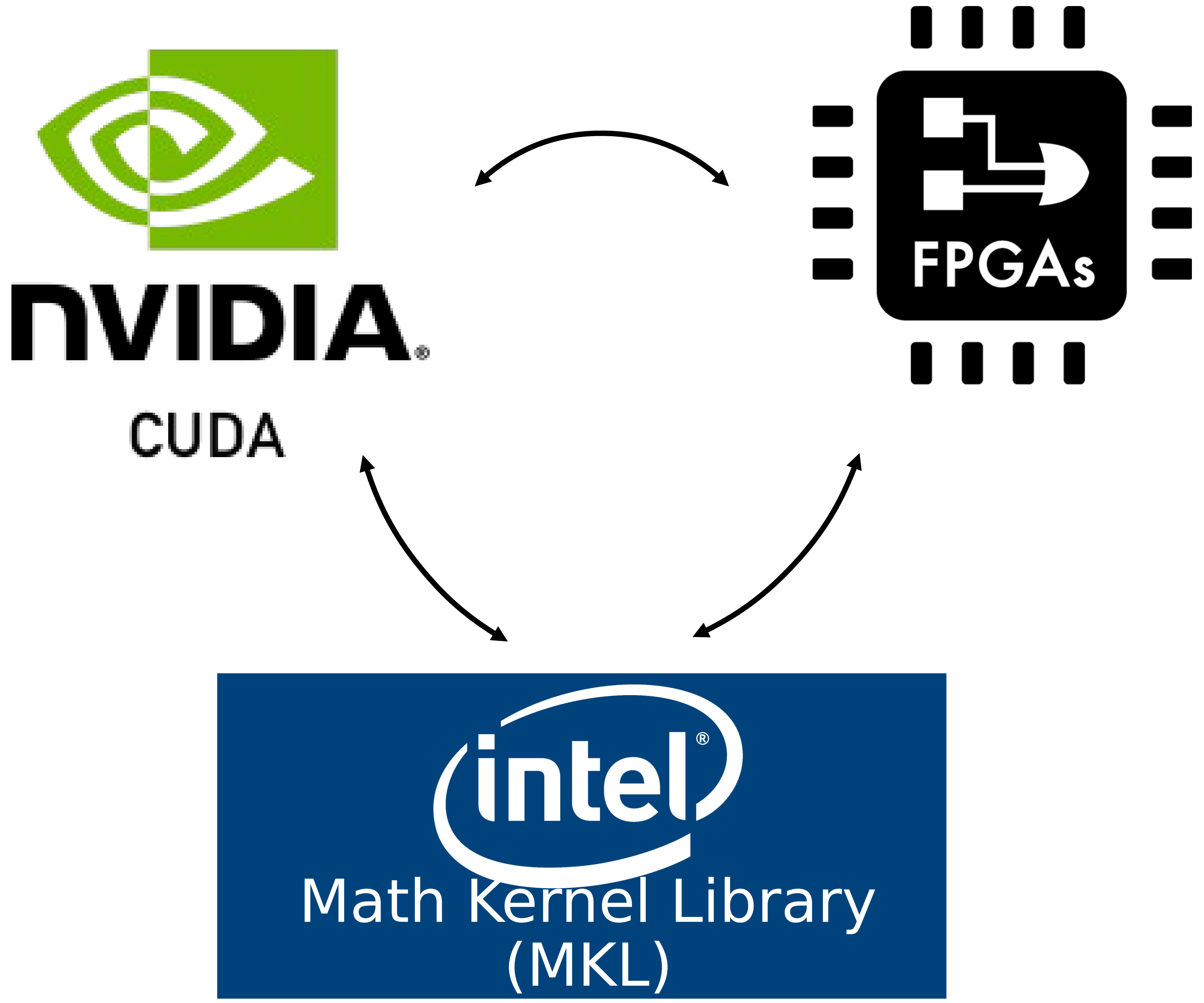


Differentiable layers

PyTorch

Modular Linear Algebra

Goal: easily switch between compute runtimes/systems



Modular Linear Algebra Backends

Available in 1.0:

- Standard CSC (hand-coded C)
- Nvidia CUDA[1]
- Intel MKL

Experimental:

- Sparse FPGA kernels[2]

Future:

- GraphBLAS
- Sycl/oneAPI
- ROCm
- …

[1] M. Schubiger, G. Banjac, and J. Lygeros, “GPU acceleration of ADMM for large-scale quadratic programming,” *Journal of Parallel and Distributed Computing*, vol. 144, pp. 55–67, 2020.
[2] M. Wang, I. McInerney, B. Stellato, S. Boyd, & H. Kwok-Hay So, “RSQP: Problem-specific Architectural Customization for Accelerated Convex Quadratic Optimization,“ *International Symposium on Computer Architecture (ISCA) 20*
Orlando, FL, USA, Jun. 2023. *(To appear).*

Modular Linear Algebra from Python

One-line import change

```python
# Import OSQP from a specific algebra backend module
from osqp.mkl import OSQP as OSQP_mkl
from osqp.cuda import OSQP as OSQP_cuda

prob_mkl = OSQP_mkl()
prob_cuda = OSQP_cuda()

# Setup workspace and change alpha parameter
prob_mkl.setup(P, q, A, l, u, alpha=1.0)

# Solve problem
res = prob_mkl.solve()
```

Setting in object constructor

```python
# Create an OSQP object with a specific algebra backend
if osqp.algebra_available('cuda'):
    # 'builtin' (default), 'mkl', or 'cuda'
    prob = osqp.OSQP(algebra='cuda')
else:
    prob = osqp.OSQP()

# Setup workspace and change alpha parameter
prob.setup(P, q, A, l, u, alpha=1.0)

# Solve problem
res = prob.solve()

...

# Solve with OSQP cuda on CVXPY
import cvxpy as cp

problm = cp.Problem(...)
problem.solve(solver=OSQP, algebra="cuda")
```

It works with CVXPY

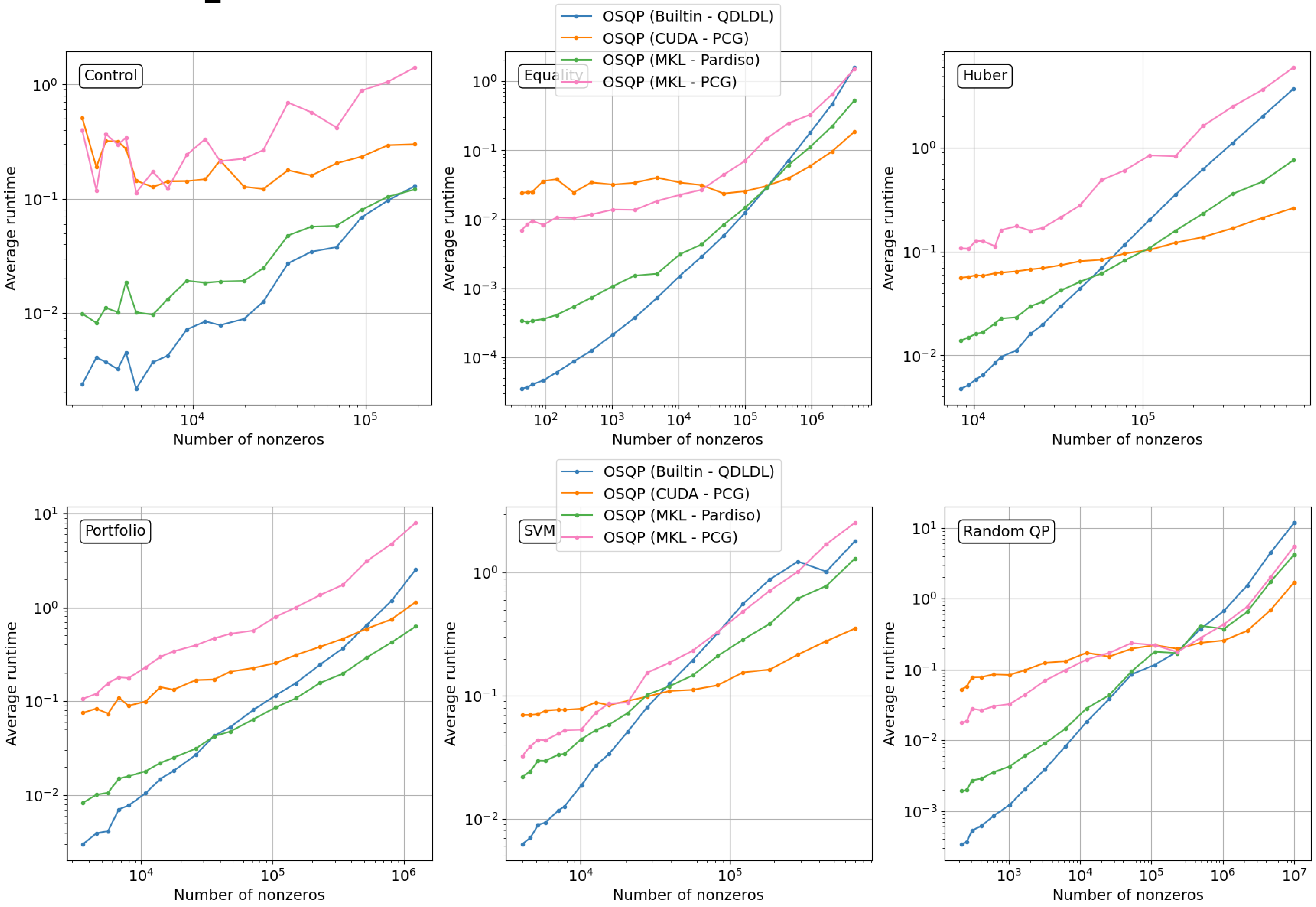
Modular Linear Algebra from Julia

One-line import change

```julia
using JuMP
using OSQP
using OSQPMKL

model = Model( () -> OSQP.Optimizer(OSQPMKLAlgebra()) )

@variable(model, x >= 0)
@variable(model, 0 <= y <= 3)
@objective(model, Min, 12x + 20y)
@constraint(model, c1, 6x + 8y >= 100)
@constraint(model, c2, 7x + 12y >= 120)
print(model)
optimize!(model)
```

It works with JuMP

Comparisons of different algebras



$\mathbf{nnz} > 10^5$

CUDA is much faster

CPU: Intel Xeon W-2255, 3.70GHz

CUDA GPU: NVIDIA T1000

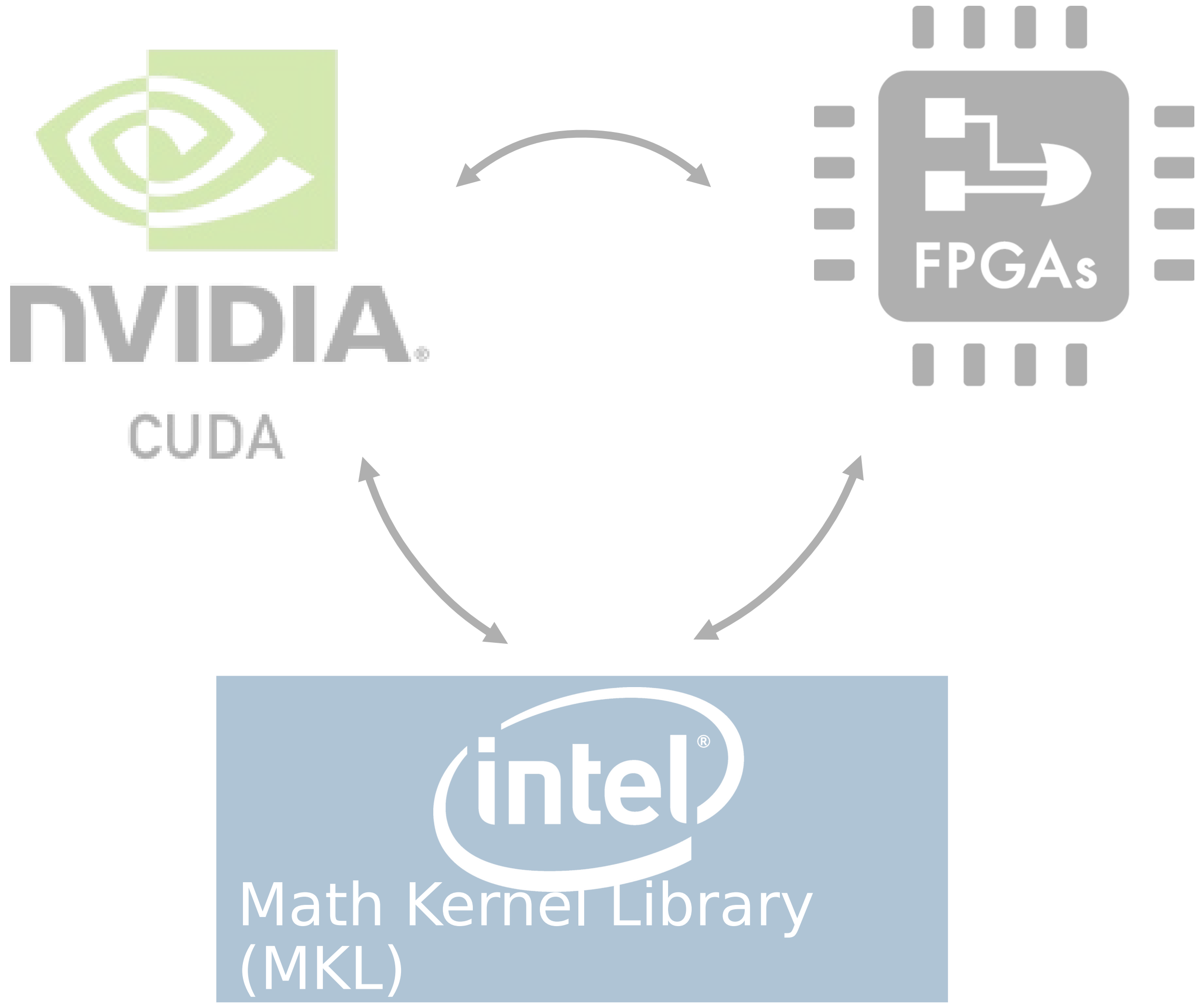
MKL: AVX512 support with 10 threads

What's new in OSQP 1.0

Improved embedded code generation

Code generation from C to C

Modular linear algebra



Differentiable layers

PyTorch

Derivatives computation in C

PyTorch

Implicit neural network architecture

Predictions $\xrightarrow{\theta = (P, q, A, l, u)}$

$$
\begin{array}{ll}
\text{minimize} & (1/2)x^T P x + q^T x \\
\text{subject to} & l \leq Ax \leq u
\end{array}
$$

$\xrightarrow{x^\star(\theta)}$ Loss

Can model decision-making and constraints → **Many applications** control, robotics, optimal-transport, meta-learning…

For learning, we need to compute derivatives (backpropagate)

$$Dx^\star(\theta)$$

However, no QP solver supports derivatives internally (from C)!

Differentiating through QPs

$$\begin{array}{ll} \text{minimize} & (1/2)x^TPx + q^Tx \\ \text{subject to} & Ax \leq b \end{array}$$

- x variable
- $\theta = (P, q, A, b)$ parameter

Optimality conditions

stationarity

complementary slackness

$$\left.\begin{array}{r} \boxed{Px^\star + q + A^Ty^\star = 0} \\ \boxed{\mathbf{diag}(y^\star)(Ax^\star - b) = 0} \end{array}\right] \longrightarrow F(z^\star(\theta), \theta) = 0$$

$$Ax^\star - b \leq 0$$

$$y^\star \geq 0$$

primal/dual solution

$$z^\star = (x^\star(\theta), y^\star(\theta))$$

Goal

Compute $Dz^\star(\theta)$

Differentiating through convex optimization problems

$$F(z^\star(\theta), \theta) = 0$$

primal/dual solution

$$z^\star = (x^\star(\theta), y^\star(\theta))$$

Implicit function theorem

$$D_z F(z^\star, \theta) D z^\star(\theta) + D_\theta F(z^\star, \theta) = 0$$

$$\downarrow$$

$$D z^\star(\theta) = -\left(D_z F(z^\star, \theta)\right)^{-1} D_\theta F(z^\star, \theta)$$

($D_z F(z^\star, \theta)$ must be invertible) **linear system solution**

We plug $Dz^\star(\theta)$ in AD (automatic differentiation)

[Differentiating through a cone program. Agrawal, Barratt, Boyd, Busseti, Moursi. Journal of Applied and Numerical Optimization 2019]

[Differentiable Optimization-Based Modeling for Machine Learning. Amos. PhD Thesis 2019]

Derivatives computation directly in C

Import and define Pytorch layer

```python
from osqp.nn import OSQP  # Import Torch module

...

# Initialize NN layer with sparsity pattern
qp_layer = OSQP(P_idx, P_shape, A_idx, A_shape)

...

# Define architecture
x_star = qp_layer(P, q, A, l, u)

...

#  Loss based on x_star
l = loss(x_star)
```

Inside, it is calling this

```c
OSQPInt osqp_adjoint_derivative_compute(OSQPSolver* solver,
                                        OSQPFloat*  dx,
                                        OSQPFloat*  dy_l,
                                        OSQPFloat*  dy_u)
                                        { ... }

OSQPInt osqp_adjoint_derivative_get_mat(OSQPSolver*    solver,
                                        OSQPCscMatrix* dP,
                                        OSQPCscMatrix* dA);
                                        { ... }

OSQPInt osqp_adjoint_derivative_get_vec(OSQPSolver* solver,
                                        OSQPFloat*  dq,
                                        OSQPFloat*  dl,
                                        OSQPFloat*  du);
                                        { ... }
```

Still work in progress!

(To be integrated with CVXPYLayers and Flux.jl)

Contributors

Ian McInerney

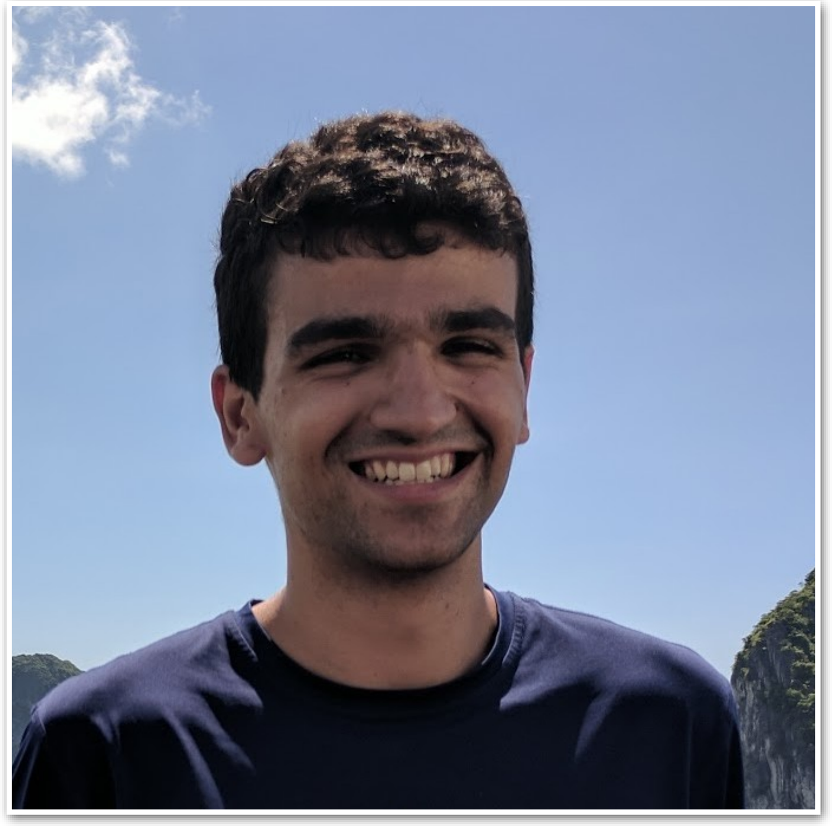
Vineet Bansal

Bartolomeo Stellato

Paul Goulart

Goran Banjac

Rajiv Sambharya



OSQP 1.0 – Beta Released!

20 minutes ago

imciner2

v1.0.0.beta0

3b3d162

Compare

v1.0.0.beta0 Pre-release

First beta release of OSQP v1.0

New features:

- Introduced new linear algebra backend system allowing compute framework to be changed at compile time.
- Merged cuOSQP project into main OSQP project (inside algebra/cuda directory).
- Introduced an Intel MKL-based algebra backend using the MKL sparse BLAS API, Vector Math Library. This backend contains both the Pardiso solver and an RCI conjugate gradient implementation.
- Added code generation capabilities to the C-level API (note, only problem export is in the C API, no file copying is done).
- Added initial adjoint derivative computation to the C-level API.

Main changes:

[github.com/osqp/{osqp,osqp-python,OSQP.jl}](https://github.com/osqp/)