IMPERIAL

ACCESS
AI Chip Center for Emerging Smart Systems
智能晶片與系統研發中心

PRINCETON UNIVERSITY

CENTER FOR STATISTICS AND MACHINE LEARNING

OSQP with GPUs & FPGAs

Accelerating quadratic programming on heterogeneous systems

Ian McInerney – Imperial College London
Maolin Wang – ACCESS, Hong Kong University of Science and Technology
Bartolomeo Stellato – Princeton University, ORFE
Vineet Bansal – Princeton University, CSML
Amit Solomon – Princeton University

SIAM Conference on Computational Science and Engineering
06/03/2025

Contributors

Ian McInerney

Vineet Bansal

Bartolomeo Stellato

Maolin Wang

Paul Goulart

Goran Banjac

Michel Schubiger

Hayden Kwok-Hay So

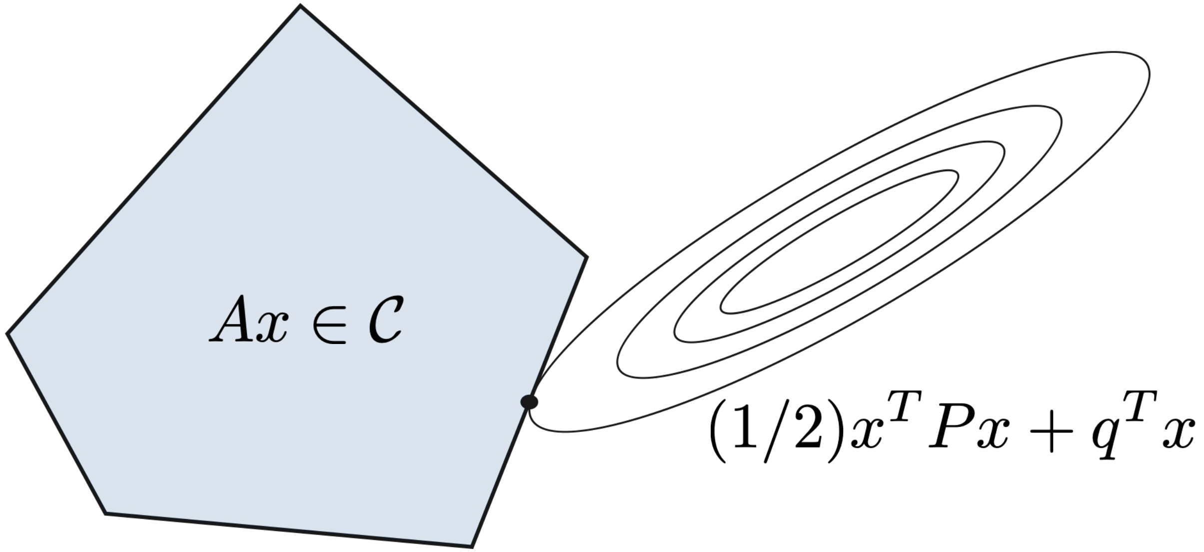
Agenda

- The OSQP Solver
- Linear Algebra Abstractions
- OSQP on FPGAs – RSQP
- Numerical Results
- The future

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



The OSQP Solver

ADMM – Alternating Direction Method of Multipliers

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

Splitting

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

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

$$
\text{minimize} \quad \overbrace{(1/2)\tilde{x}^T P \tilde{x} + q^T \tilde{x} + \mathcal{I}_{Ax=z}(\tilde{x}, \tilde{z})}^{f} + \overbrace{\mathcal{I}_{\mathcal{C}}(z)}^{g}
$$

$$
\begin{array}{ll}
\text{subject to} & \tilde{x} = x \longrightarrow \sigma \\
 & \tilde{z} = z \longrightarrow \rho
\end{array}
$$

Diagonal
step sizes

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

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

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

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

Solve using LDL factorization

Indirect method (large scale)

Positive-definite matrix

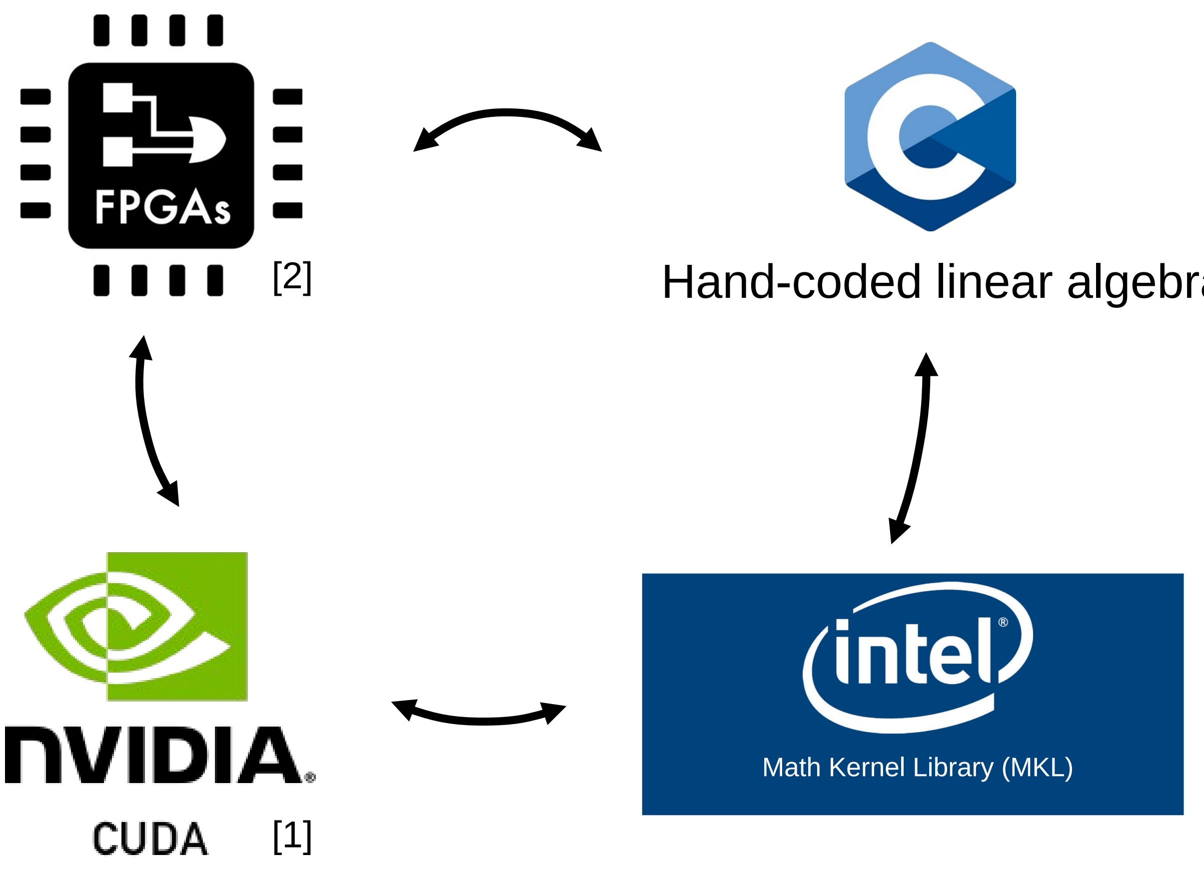
$$\left(P + \sigma I + \rho A^T A\right) x = \sigma x^k - q + A^T(\rho z^k - y^k)$$

Solve using conjugate gradient

Linear Algebra Abstractions

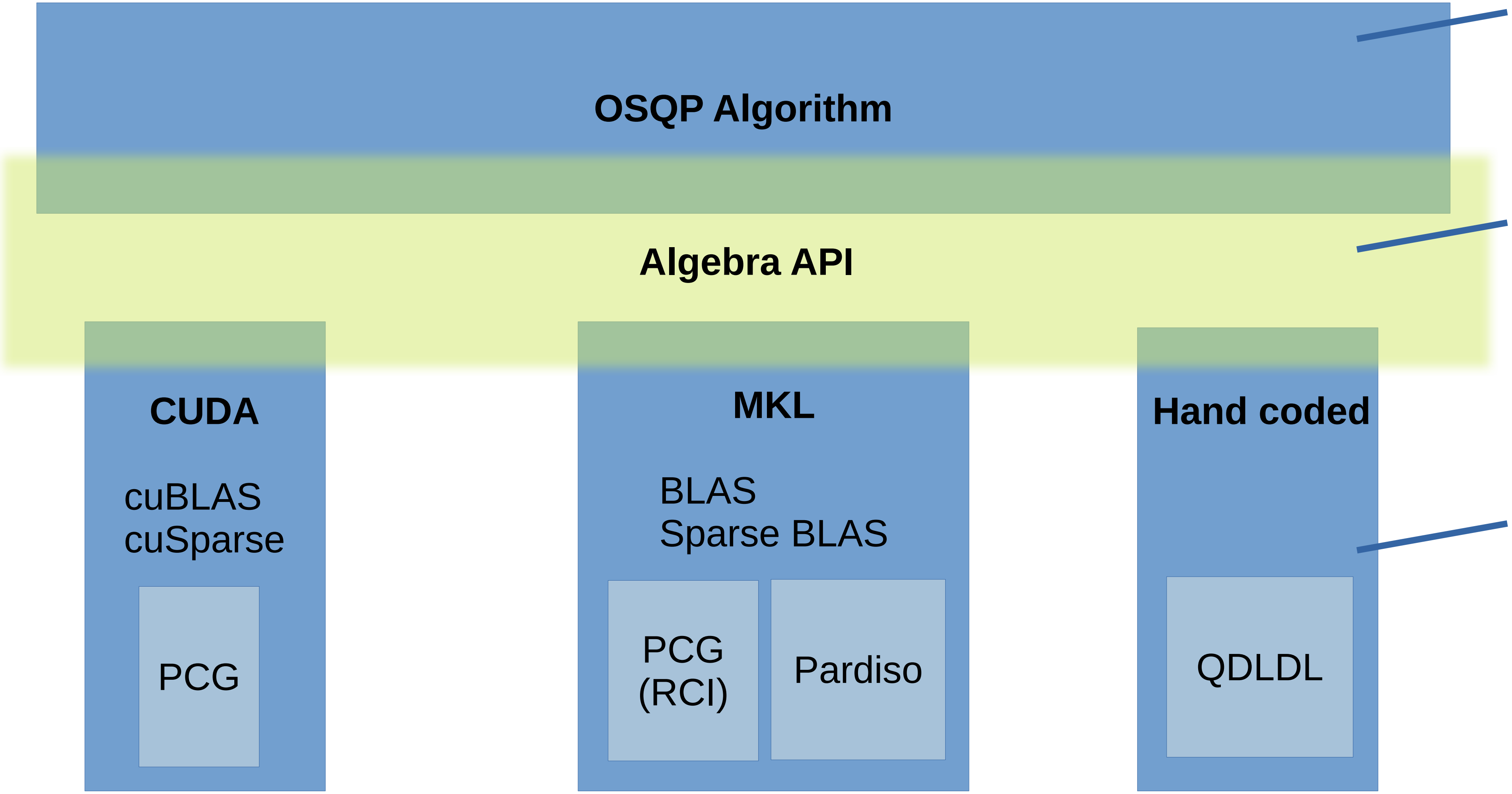
Modular Linear Algebra

Goal: easily switch between compute runtimes/systems



[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) 2023*, Orlando, FL, USA, Jun. 2023.

Modular Linear Algebra



Operates on opaque linear algebra types:
OSQPVector
OSQPMatrix

Defines operations on opaque types, e.g.,
OSQPVectorf_dot_prod
OSQPMatrix_Axpy

- Implements operations on opaque types
- Compile-time selected

Modular Linear Algebra – Key Points

Builtin

- Hand coded
- CSC-based sparse matrices
- Library free
- Linear system solvers
  - QDLDL

MKL

- CSC-based sparse matrices
- BLAS vector operations
- Linear system solvers
  - Pardiso
  - RCI Preconditioned conjugate gradient

CUDA

- cuSparse & cuBLAS libraries
- CSC & CSR matrices
- All data fully GPU resident
  - *osqp_setup* – Data copied to OSQP GPU workspace
  - *osqp_solve* – CPU-managed control flow, only transfer status values
- Linear system solvers
  - Preconditioned conjugate gradient

Same OSQP API for all backends

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

Modular Linear Algebra from Julia - Implementation

Macro to define the C function API

```julia
@cprototype osqp_solve(solver::Ptr{OSQP.OSQPSolver{Tfloat,Tint}})::Tint

@cprototype osqp_cleanup(solver::Ptr{OSQP.OSQPSolver{Tfloat,Tint}})::Tint

@cprototype osqp_warm_start(solver::Ptr{OSQP.OSQPSolver{Tfloat,Tint}}, x::Ptr{Tfloat}, y::Ptr{Tfloat})::Tint

@cprototype osqp_cold_start(solver::Ptr{OSQP.OSQPSolver{Tfloat,Tint}})::Nothing

@cprototype osqp_update_rho(solver::Ptr{OSQP.OSQPSolver{Tfloat,Tint}}, rho_new::Tfloat)::Tint
```

Macro to define mapping to specific type

```julia
Tdoubledict = Dict(
    :Tfloat => :Float64,
    :Tint => :Cc_int
)

@cdefinition OSQPBuiltinAlgebra{Float64} osqp_builtin_double osqp_solve          Tdoubledict
@cdefinition OSQPBuiltinAlgebra{Float64} osqp_builtin_double osqp_cleanup        Tdoubledict
@cdefinition OSQPBuiltinAlgebra{Float64} osqp_builtin_double osqp_warm_start     Tdoubledict
@cdefinition OSQPBuiltinAlgebra{Float64} osqp_builtin_double osqp_cold_start     Tdoubledict
@cdefinition OSQPBuiltinAlgebra{Float64} osqp_builtin_double osqp_update_rho     Tdoubledict
```

```julia
Tdoubledict = Dict(
    :Tfloat => :Float64,
    :Tint => :Cint
)
@cdefinition OSQPCUDAAlgebra{Float64} osqp_cuda_double osqp_solve          Tdoubledict
@cdefinition OSQPCUDAAlgebra{Float64} osqp_cuda_double osqp_cleanup        Tdoubledict
@cdefinition OSQPCUDAAlgebra{Float64} osqp_cuda_double osqp_warm_start     Tdoubledict
@cdefinition OSQPCUDAAlgebra{Float64} osqp_cuda_double osqp_cold_start     Tdoubledict
@cdefinition OSQPCUDAAlgebra{Float64} osqp_cuda_double osqp_update_rho     Tdoubledict
```

Modular Linear Algebra from Julia - Implementation

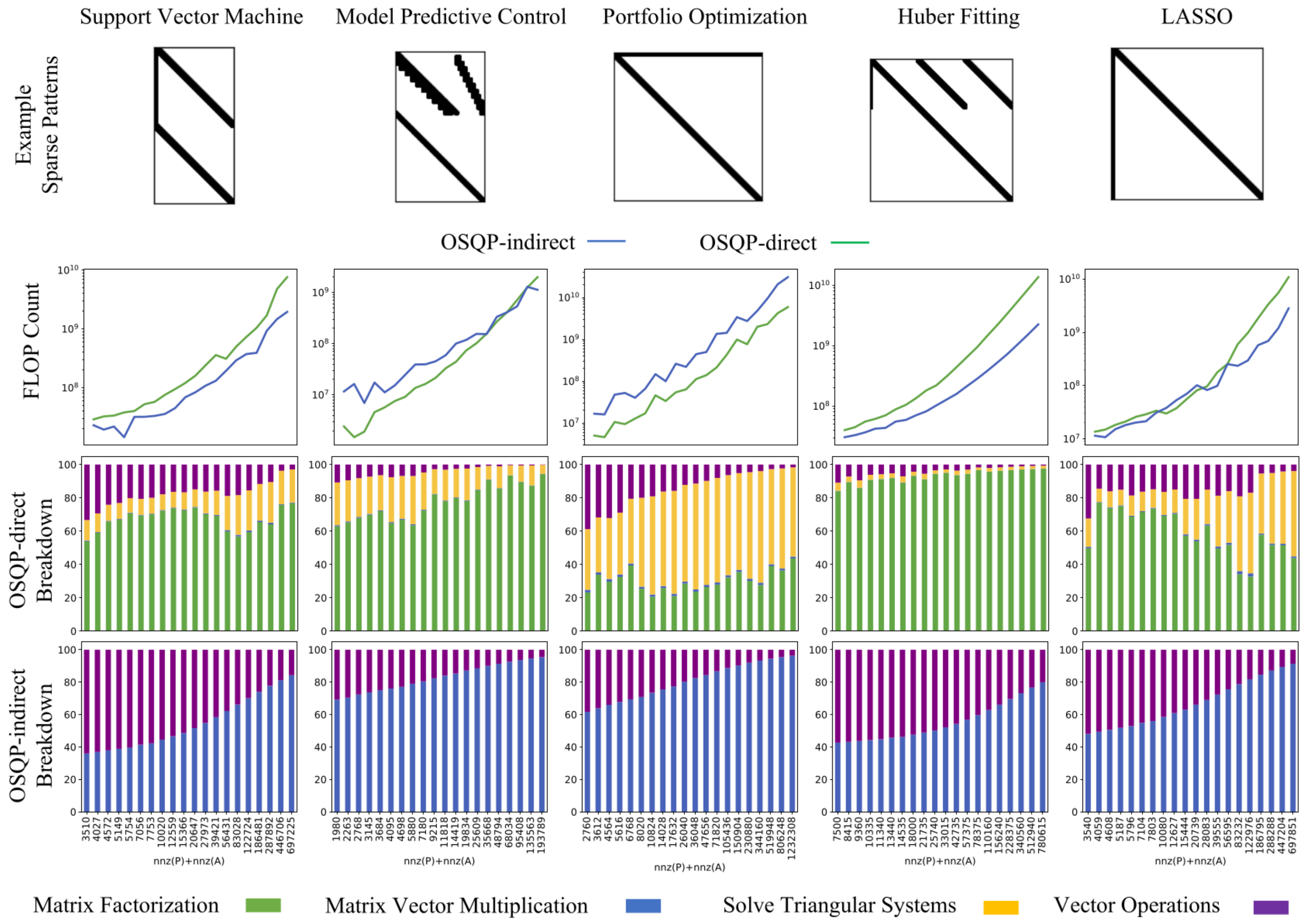
```
julia> @macroexpand OSQP.@cdefinition OSQP.OSQPBuiltinAlgebra{Float64} osqp_builtin_double osqp_warm_start            Tdoubledict
quote
    #= REPL[27]:1 =#
    import OSQP: _ccall_osqp_warm_start
    function _ccall_osqp_warm_start(::OSQP.OSQPBuiltinAlgebra{Float64}, solver, x, y)
        #= REPL[27]:1 =#
        return ccall((:osqp_warm_start, osqp_builtin_double), Cc_int, (Ptr{OSQP.OSQPSolver{Float64, Cc_int}}, Ptr{Float64}, Ptr{Float64}), solver, 
x, y)
    end
end

julia>
```

OSQP on FPGAs

The RSQP solver

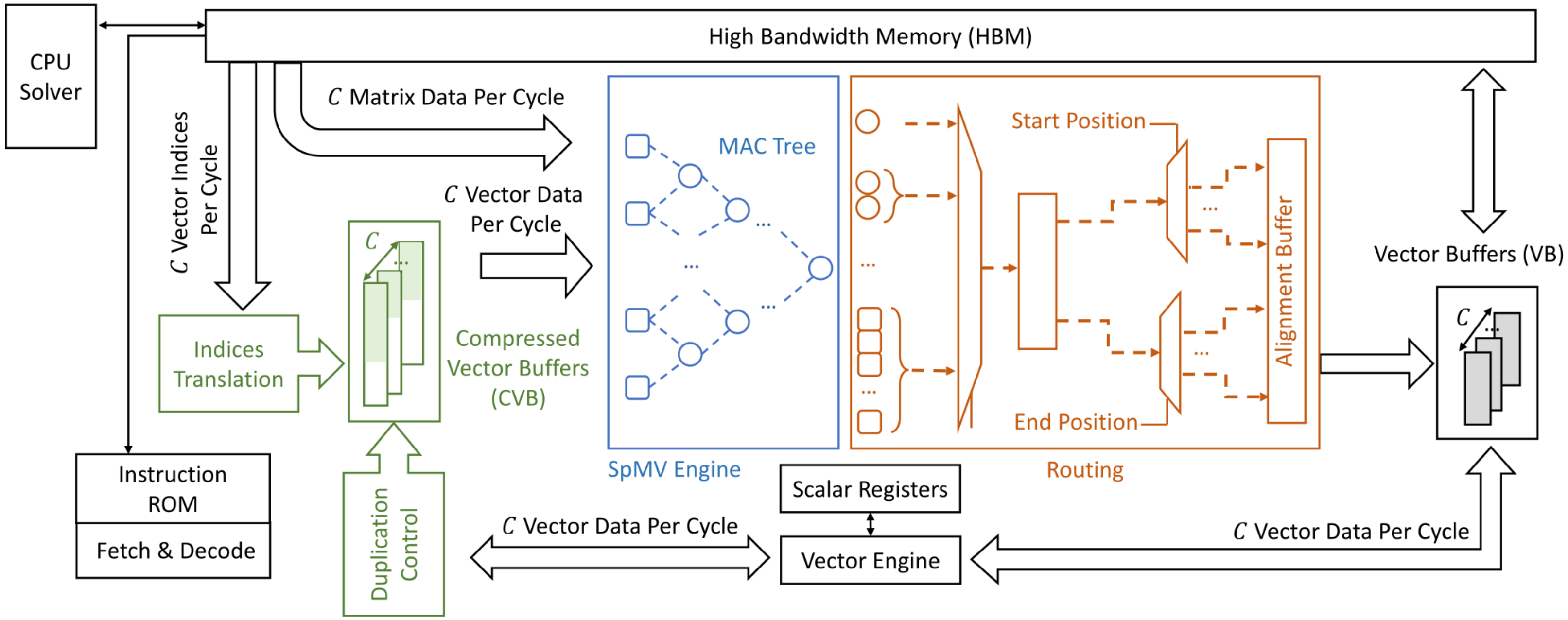
OSQP computational characteristics



RSQP – Hardware Design

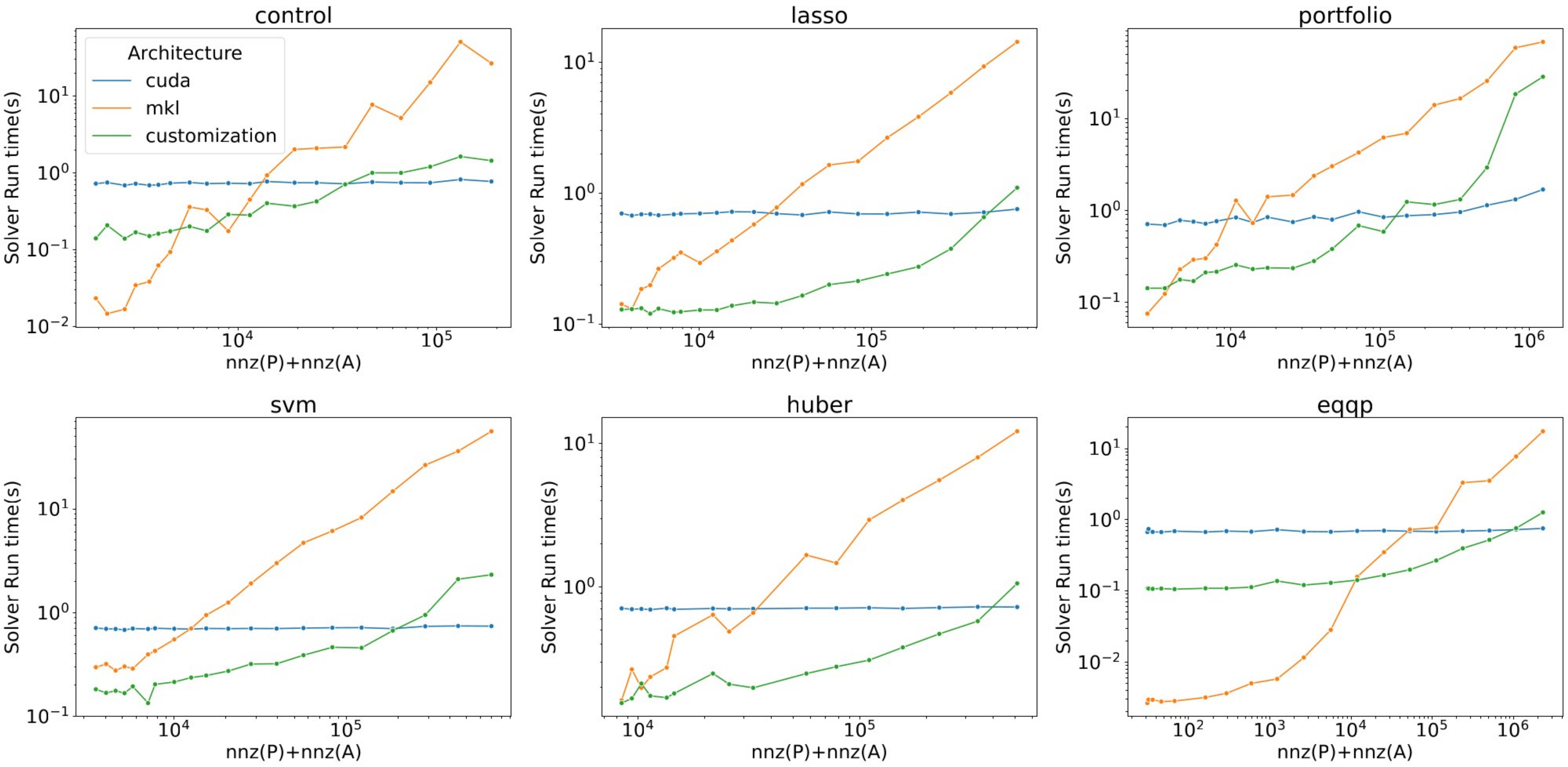
- FPGA-based design
  - Pros:
    - Custom logic
    - Reprogrammable
    - Power efficient
  - Cons:
    - Complicated to use
    - Not general purpose

- Implements OSQP indirect
  - Uses Preconditioned CG to solve the reduced KKT system
- Focus on accelerating the SpMV operation
- Implemented as an engine for *OSQPMatrix* and *OSQPVector* operations



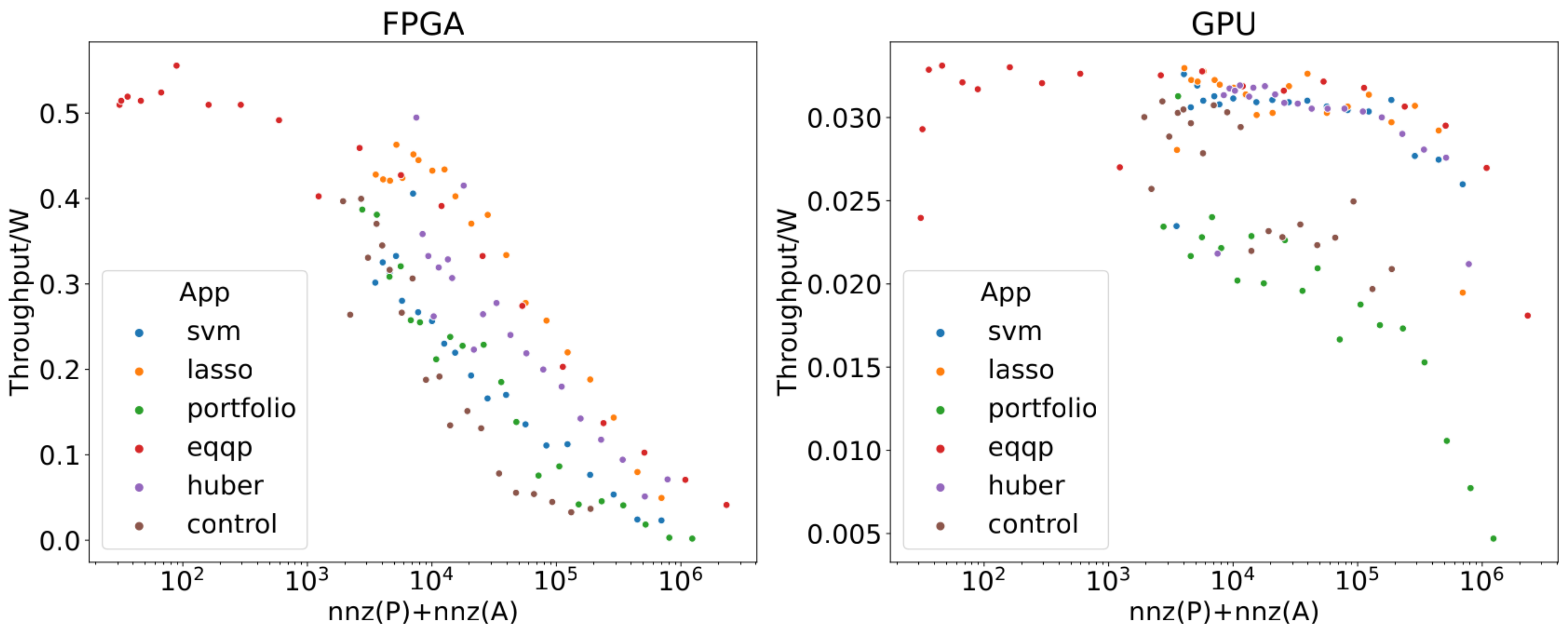
Numerical Results

RSQP – Performance



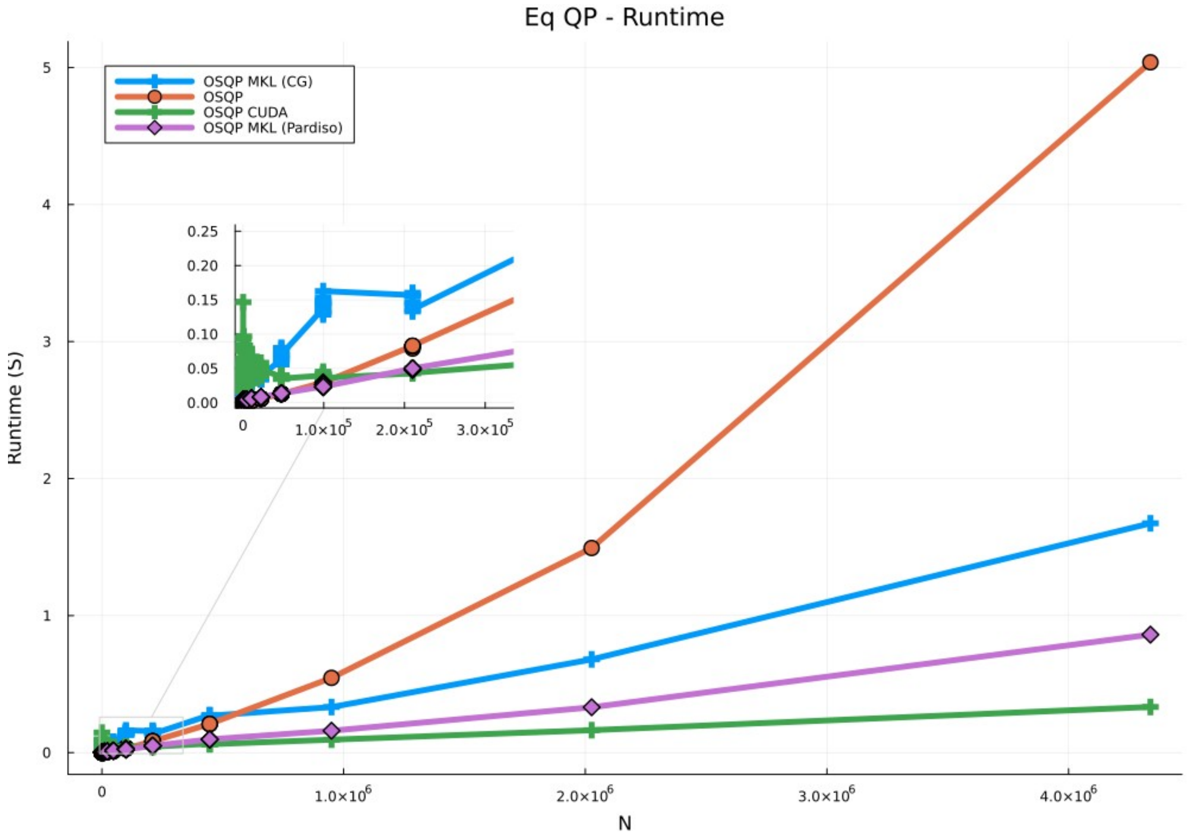
Customization: AMD(Xilinx) U50 FPGA, 8GB HBM
Cuda: NVidia RTX3070 8GB
MKL: Intel i7-10700KF

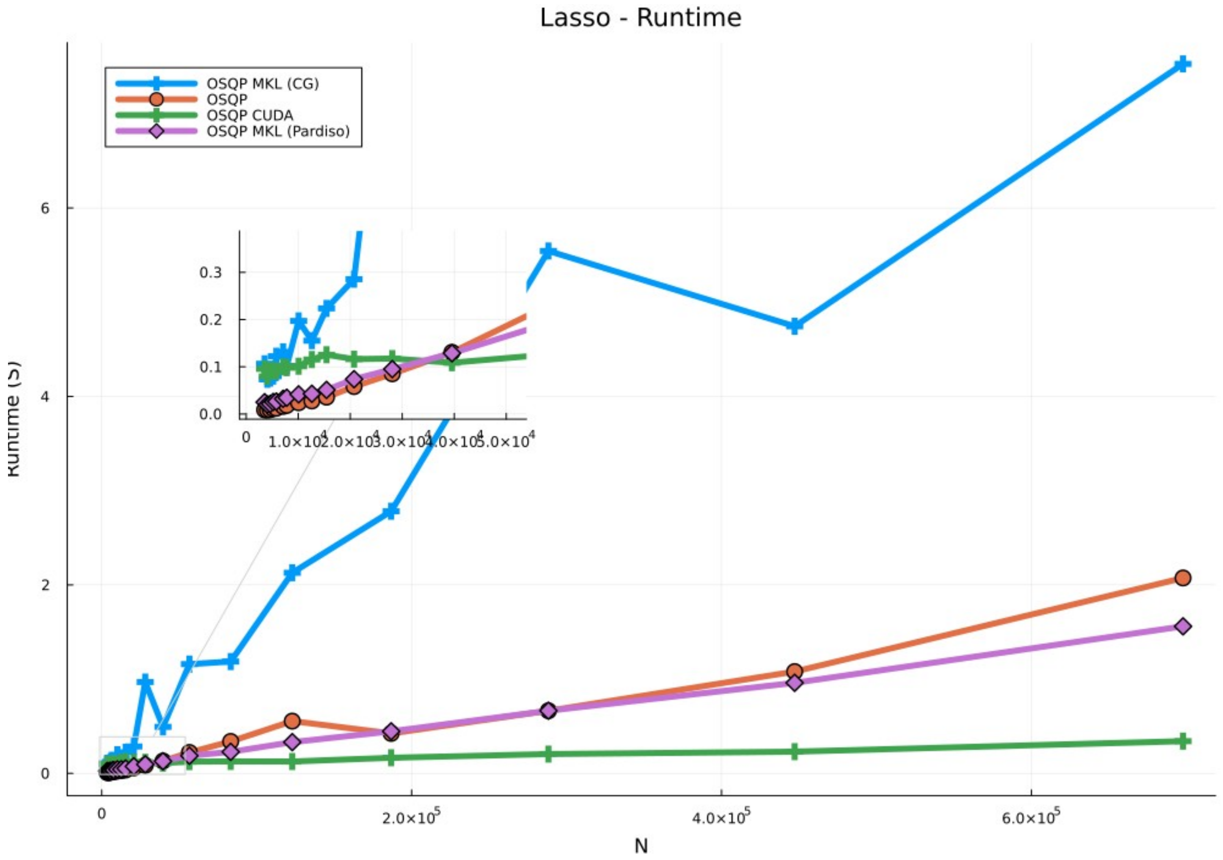
RSQP – Power



Customization: AMD(Xilinx) U50 FPGA, 8GB HBM
Cuda: NVidia RTX3070 8GB
MKL: Intel i7-10700KF

Numerical Example – Runtimes

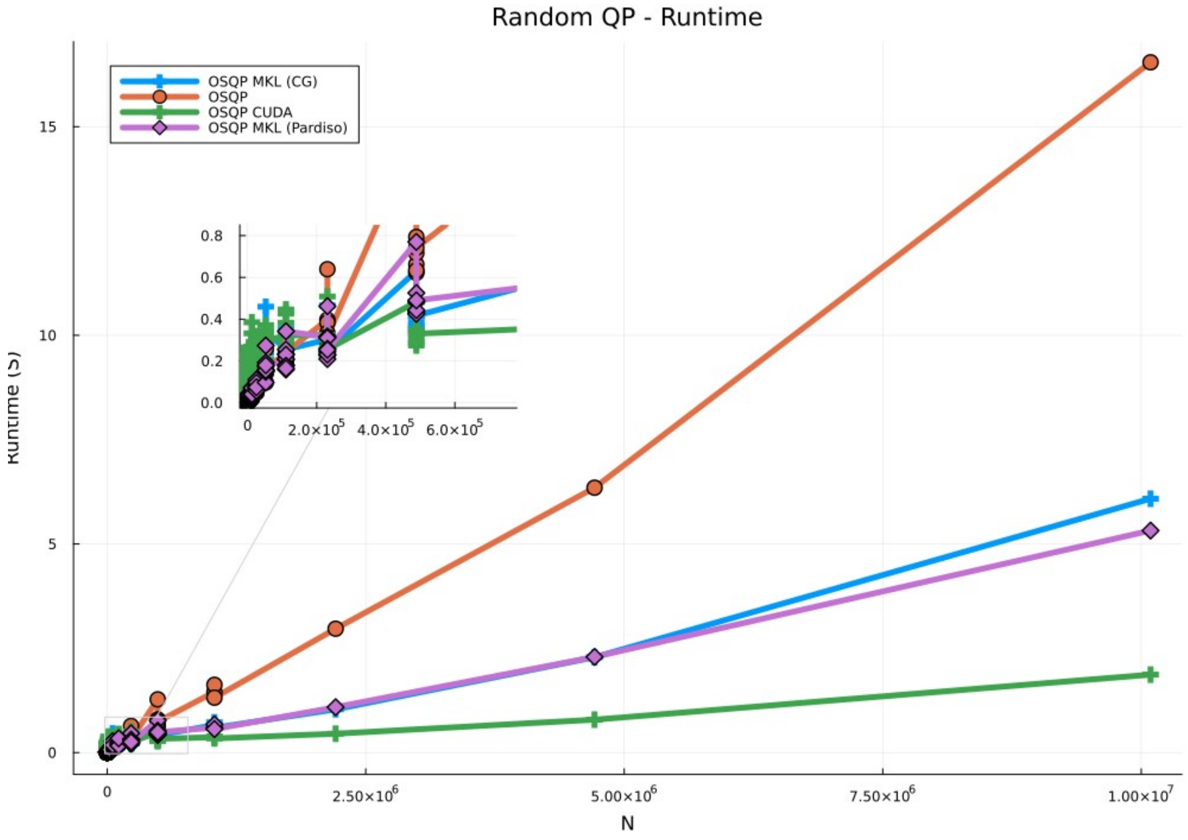


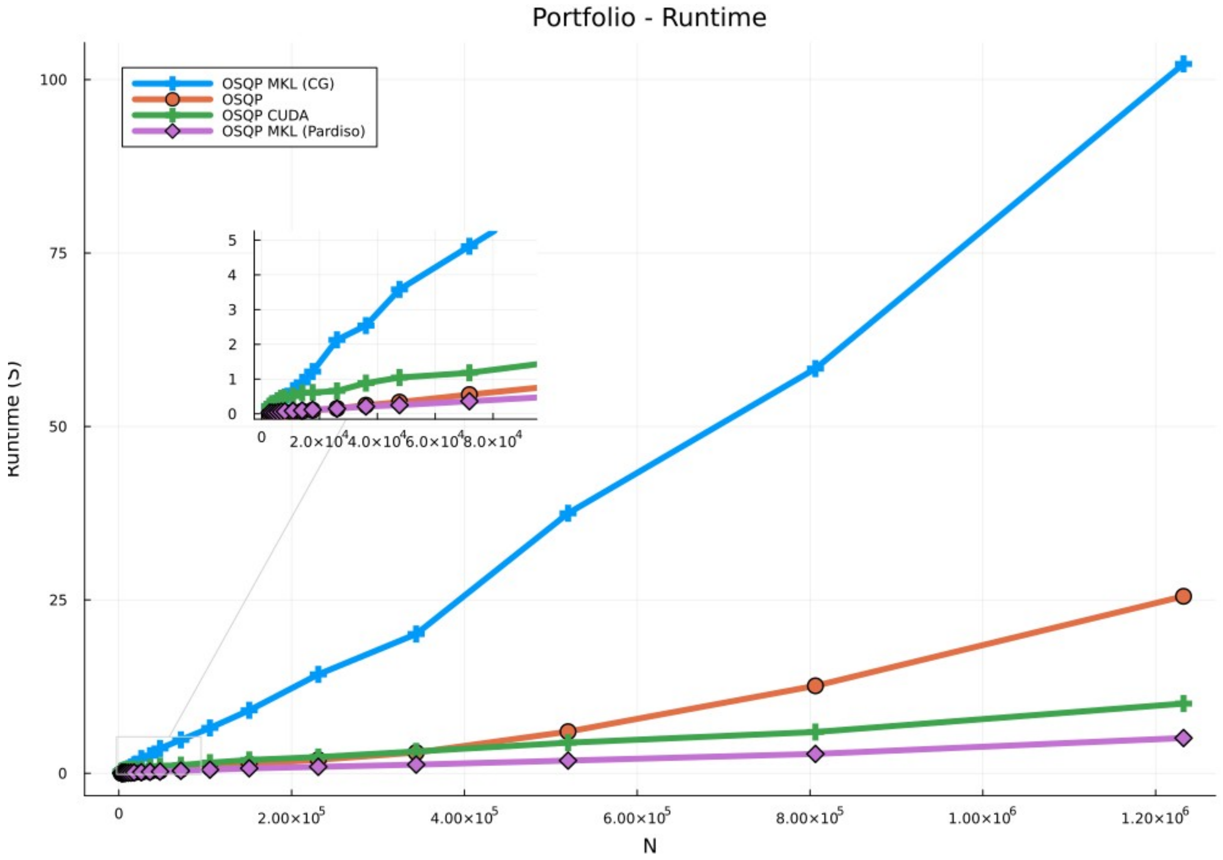


Solved to low accuracy: 1e-3

Cuda: NVidia T1000 4GB
CPU: Intel Xeon W-2255 @ 3.7GHz

Numerical Example – Runtimes

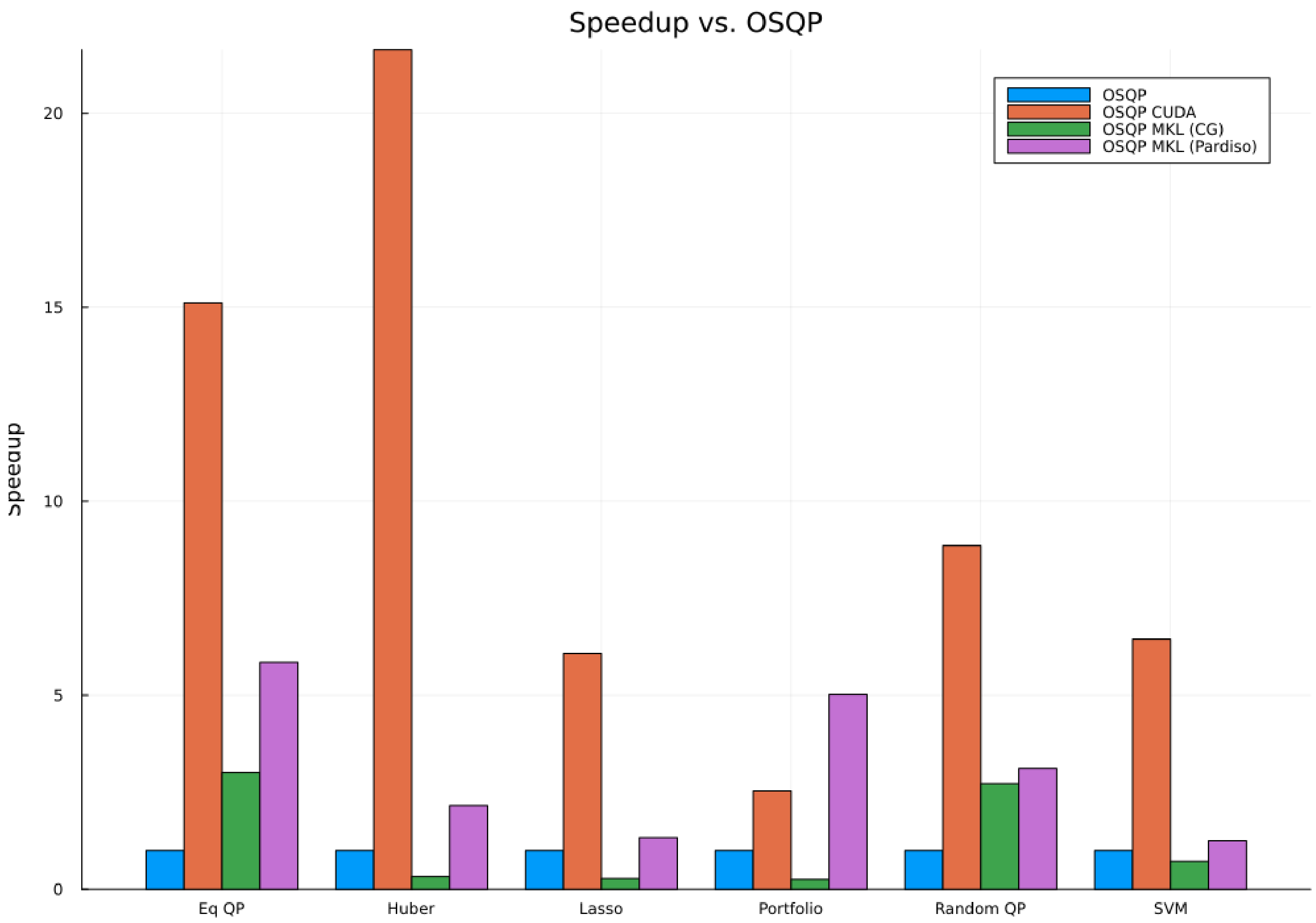




Solved to low accuracy: 1e-3

Cuda: NVidia T1000 4GB
CPU: Intel Xeon W-2255 @ 3.7GHz

Numerical Example – Speedup



Solved to low accuracy: 1e-3

Cuda: NVidia T1000 4GB
CPU: Intel Xeon W-2255 @ 3.7GHz

The future

Future Work

- Implementation details
  - Performance portable GPU implementations
    - HIP, OpenMP, Gingko, etc.
  - CUDA-specific
    - CUDA Streams and Graph solver definition
    - cuDSS direct solver
  - Batched mode
- Algorithmic improvements
  - Low/mixed precision implementations
  - Conjugate Residual solver
  - Sensitivity computations on the GPU

OSQP Papers

- B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, ‘OSQP: an operator splitting solver for quadratic programs’, *Mathematical Programming Computation*, vol. 12, pp. 637–672, 2020.
- G. Banjac, P. Goulart, B. Stellato, and S. Boyd, ‘Infeasibility Detection in Alternating Direction Method of Multipliers for Convex Quadratic Programs’, *Journal of Optimization Theory and Applications*, vol. 183, pp. 490–519, 2019.
- G. Banjac, B. Stellato, N. Moehle, P. Goulart, A. Bemporad, and S. Boyd, ‘Embedded Code Generation Using the OSQP Solver’, in *56th IEEE Conference on Decision and Control (CDC)*, Melbourne, Australia: IEEE, 2017, pp. 1906–1911.
- 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.
- 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) 2023*, Orlando, FL, USA, Jun. 2023.
- M. Wang, I. McInerney, B. Stellato, F. Tu, S. Boyd, H. Kwok-Hay So, K.T. Cheng, “Multi-Issue Butterfly Architecture for Sparse Convex Quadratic Programming,“ *57th IEEE/ACM International Symposium on Microarchitecture*, Austin, TX, USA, Nov. 2024.
- I. McInerney, A. Solomon, V. Bansal, P. Goulart, G. Banjac, & B. Stellato, “OSQP 1.0: A quadratic programming solver with code generation and selectable linear algebra backends,” (*In preparation).*

Backup slides

Modular Linear Algebra

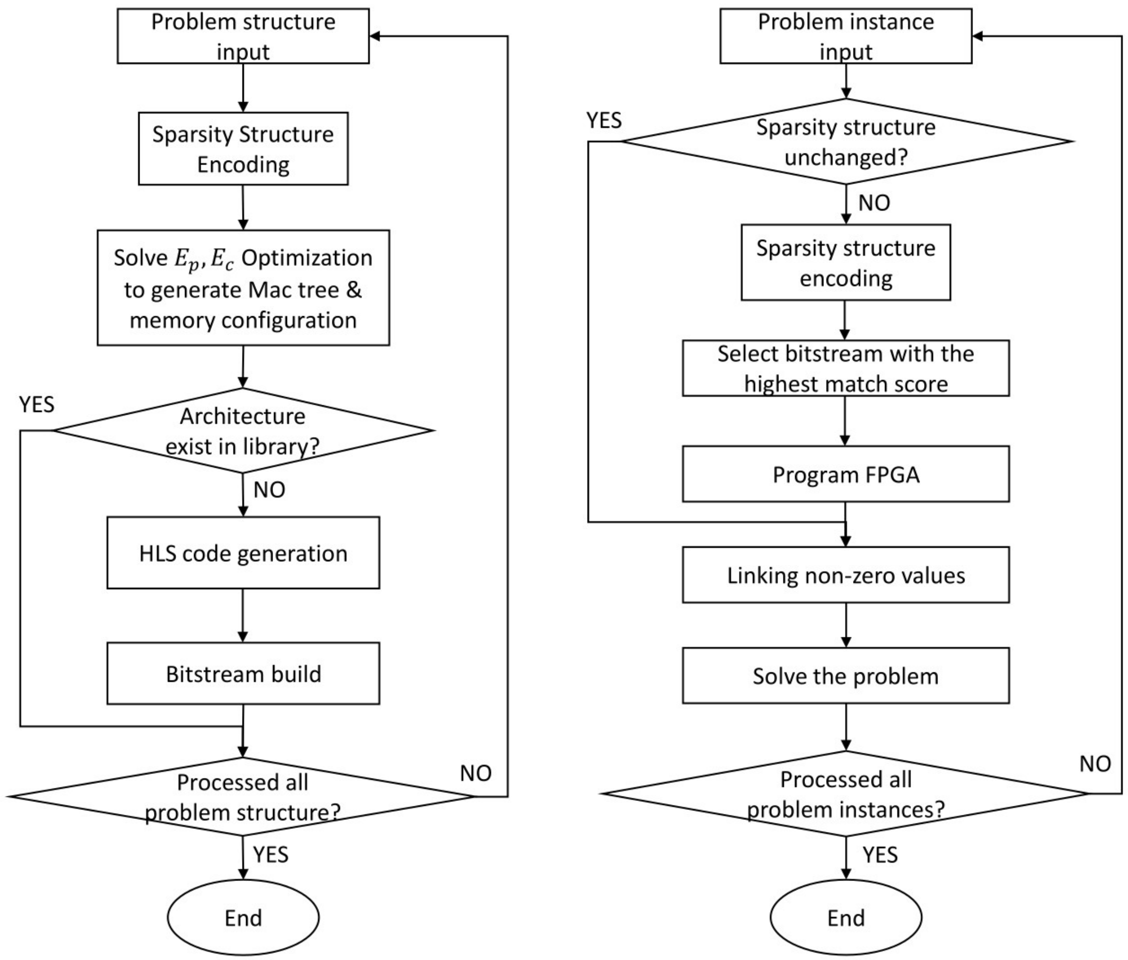
- Abstraction layer provides three categories of functions
  - *OSQPVector* operations
  - *OSQPMatrix* operations
  - Linear system solver

- *OSQPVector* and *OSQPMatrix* are opaque to the ADMM implementation

- Compile-time selection of linear algebra libraries for the C-library
- Run-time selection for Python/Julia interfaces

RSQP – Exploit the sparsity pattern

- Analyze sparsity pattern of all the matrices
- Compute problem-specific hardware design
  - Optimal compression of matrix data into memory
  - Optimal multiply-accumulate tree for sparsity pattern
  - Optimal data processing timeline



(a) Design Flow (b) Deployment Flow

cuOSQP - Technology stack

- Implemented using
  - Custom CUDA kernels
  - cuSparse (for SpMV)
  - cuBLAS (for vector operations)
- Data storage
  - *OSQPVector* – Single device array
  - *OSQPMatrix* – Two internal matrices, one CSC and one CSR
- Packaged/distributed using
  - Python wheels
  - Julia Yggdrasil