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

INFORMS Annual Meeting
23/10/2024

Contributors

Ian McInerney

Vineet Bansal

Bartolomeo Stellato

Maolin Wang

Paul Goulart

Goran Banjac

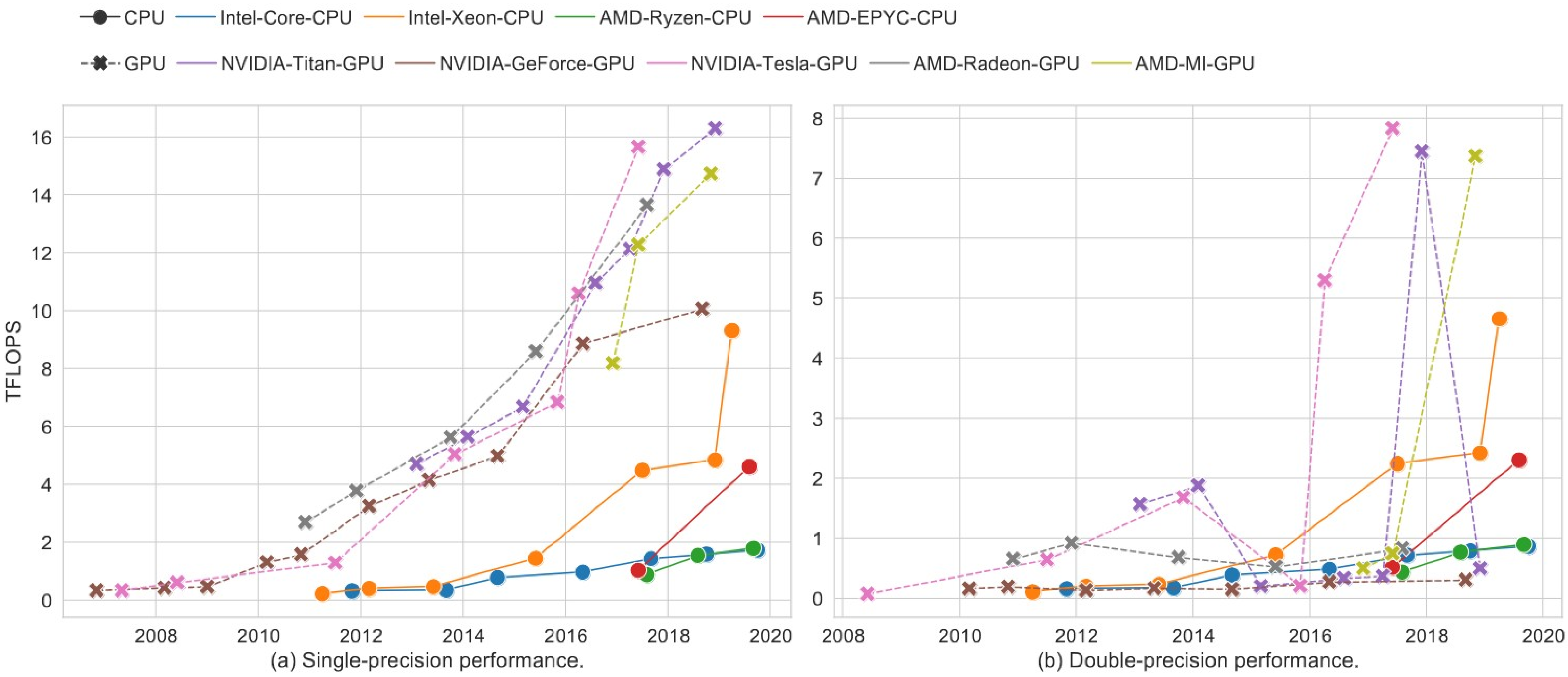
Michel Schubiger

Hayden Kwok-Hay So

Agenda

- The OSQP Solver
- Linear Algebra Abstractions
- OSQP on FPGAs – RSQP
- OSQP on GPUs – cuOSQP
- The future

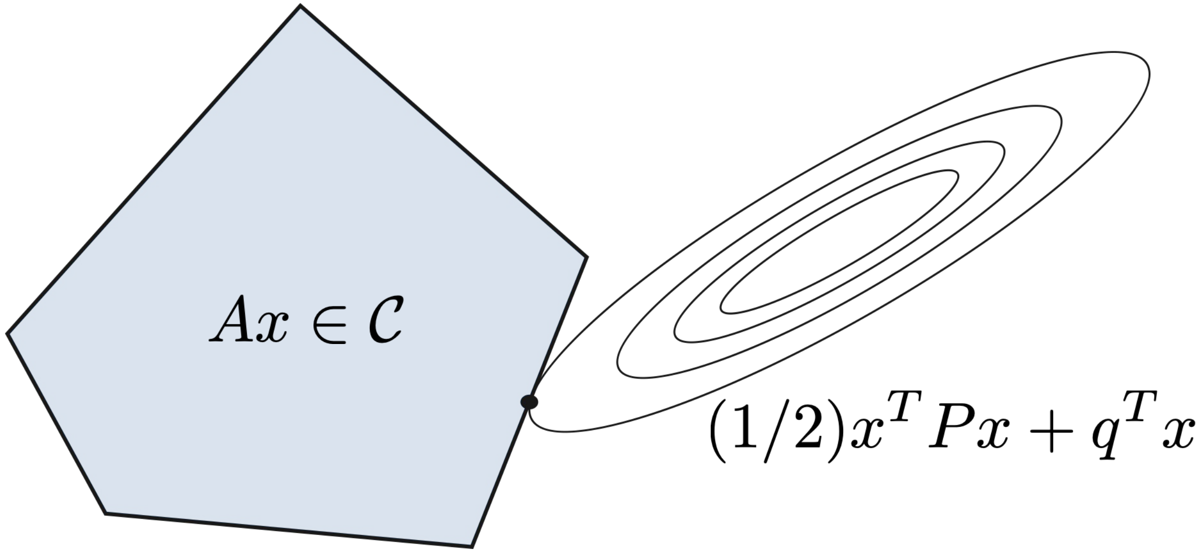
Tremendous progress in compute



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

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

First-order Methods

Pros

- Warm-starting
- Large-scale problems
- Embeddable

Cons

- Low quality solutions
- Can't detect infeasibility
- Problem data dependent

OSQP

- High-quality solutions
- Detects infeasibility
- Robust
- Embeddable (division free)

ADMM – Alternating Direction Method of Multipliers

Splitting

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

Iterations

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

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

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

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

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

Diagonal
step sizes

σ

ρ

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

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

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

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

Conjugate gradient

Solve very large systems

→ **GPU & FPGA implementation**

Complete algorithm – Indirect method

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

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

Easy operations

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

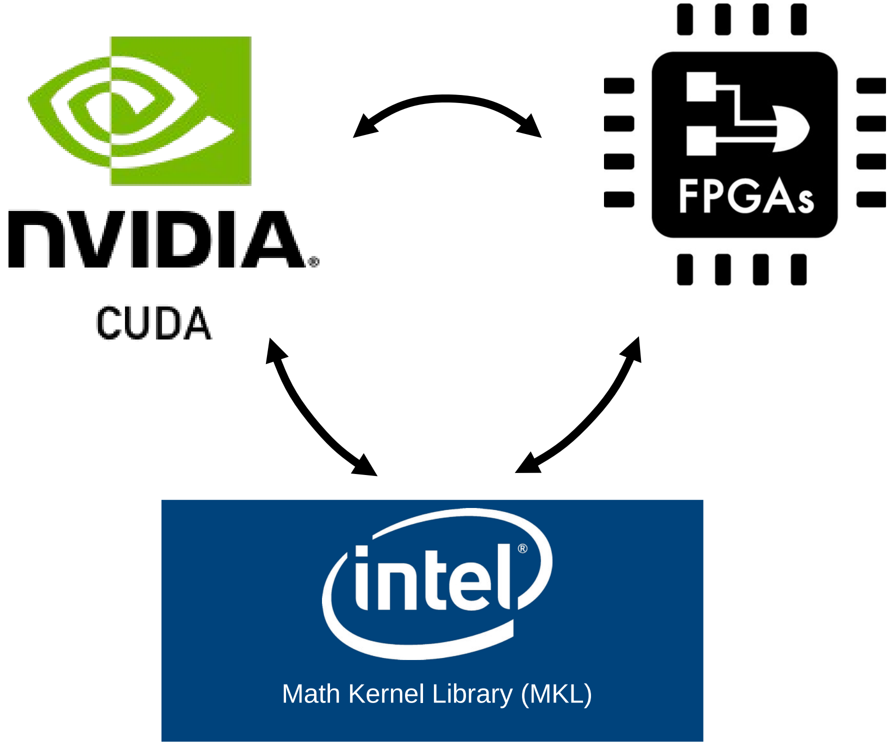
$$y^{k+1} \leftarrow y^k + \rho(Ax^{k+1} - z^{k+1})$$

always solvable!

Linear Algebra Abstractions

Modular Linear Algebra

Goal: easily switch between compute runtimes/systems



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

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

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

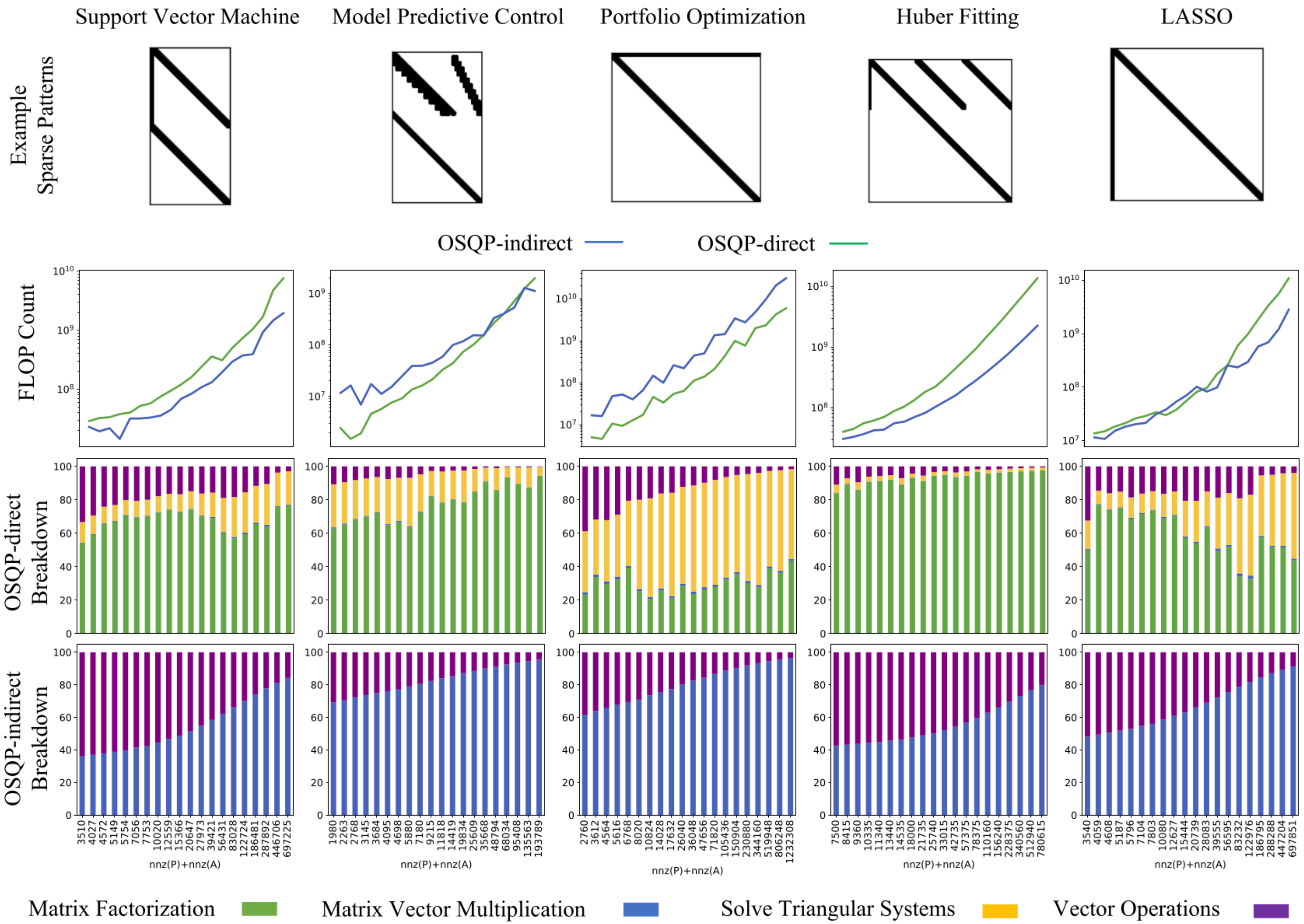
```julia
using JuMP
using OSQP
using OSQPMKL

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

OSQP on FPGAs

The RSQP solver

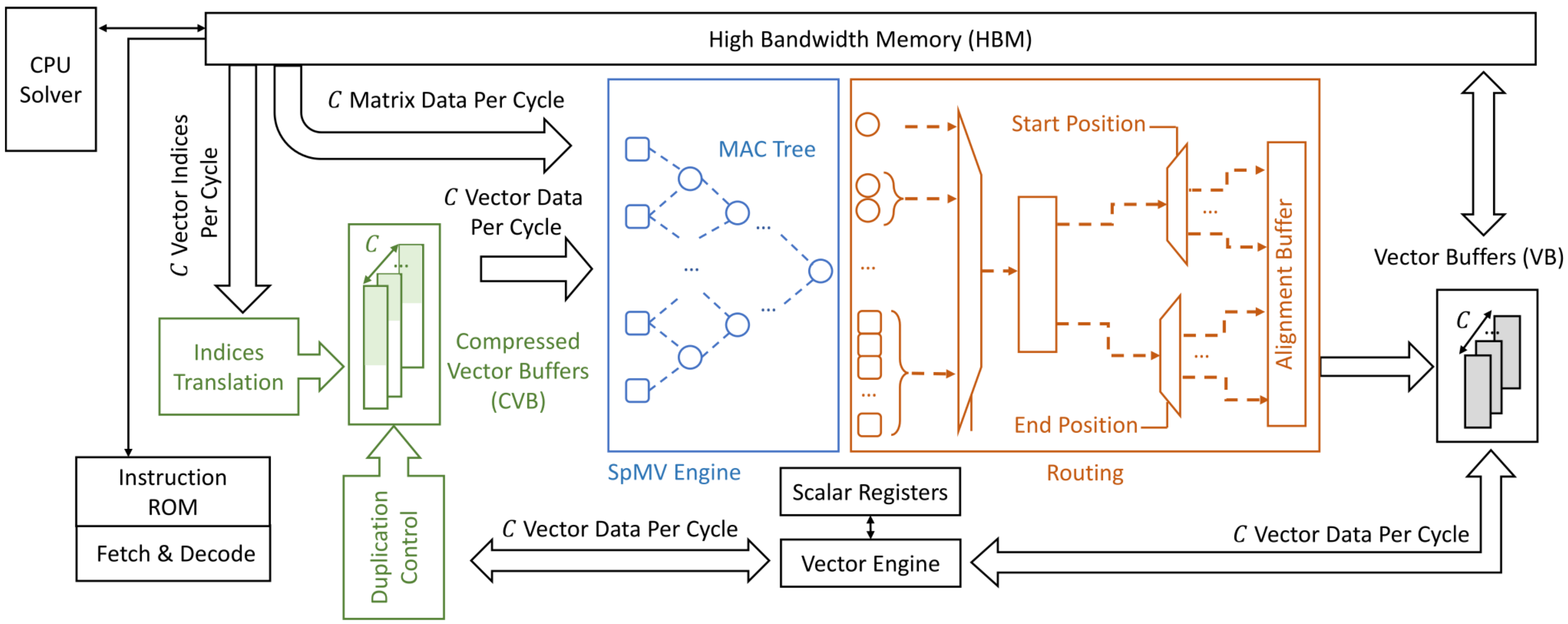
OSQP computational characteristics



RSQP – Hardware Design

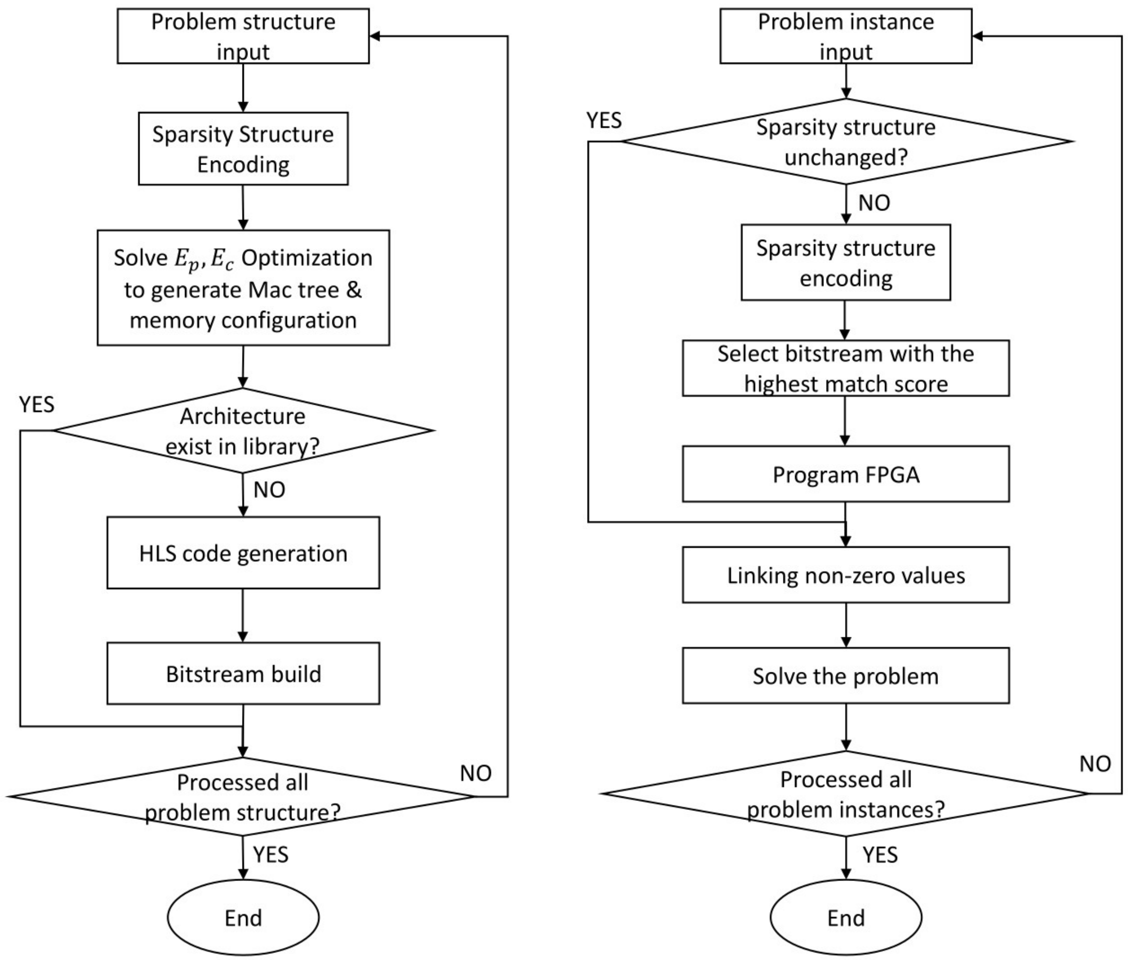
- FPGA-based design
  - Pros:
    - Custom logic
    - Reprogrammable
    - Power efficient
  - Cons:
    - Complicated to use
    - Not general purpose

- 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



RSQP – Exploit the sparsity pattern

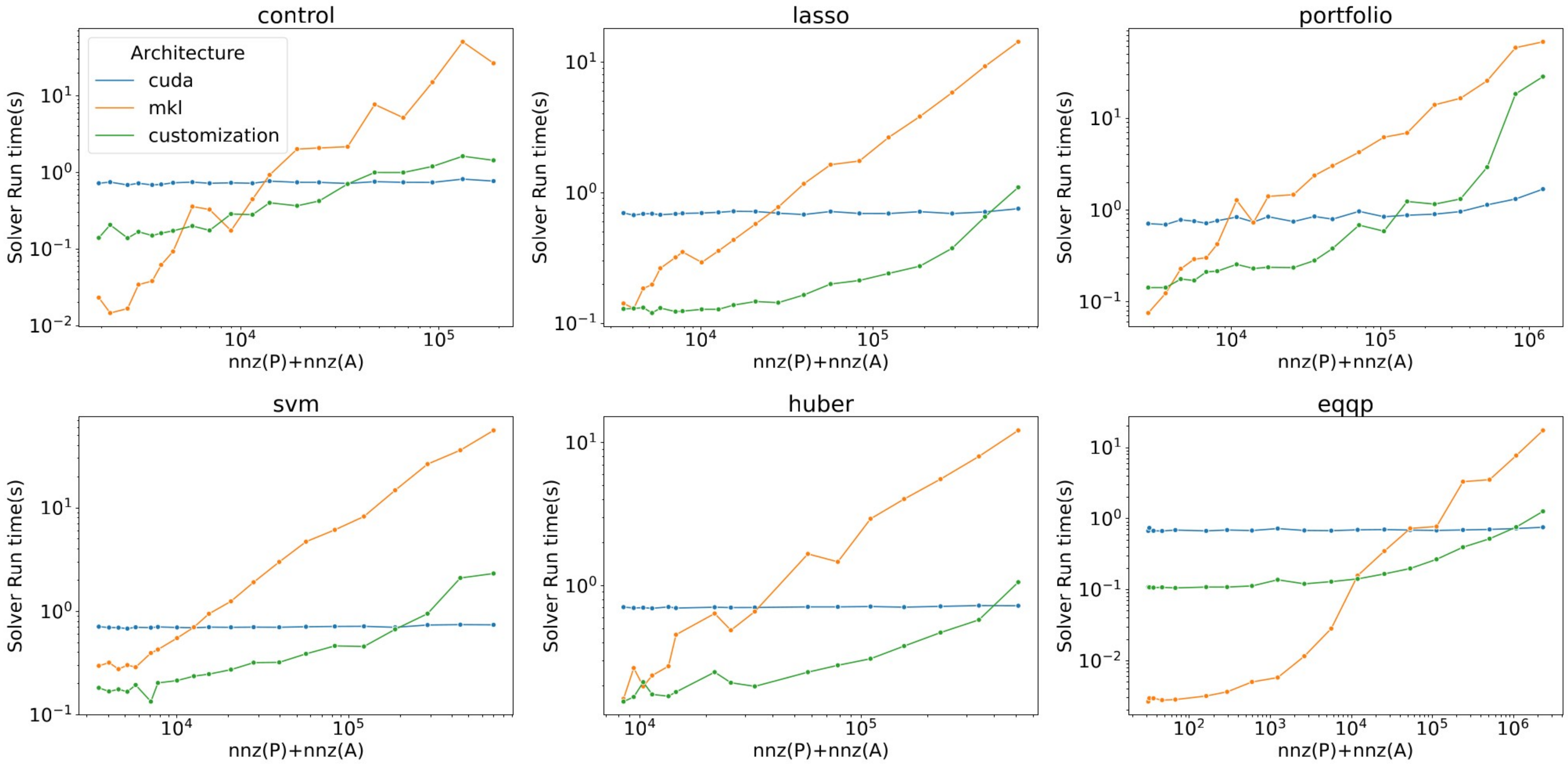
- Analyze sparsity pattern of all the matrices
- Compute problem-specific hardware design
  - Optimal compression of matrix data into memory
  - Optimal multiply-accumulate tree for sparsity pattern
  - Optimal data processing timeline



(a) Design Flow

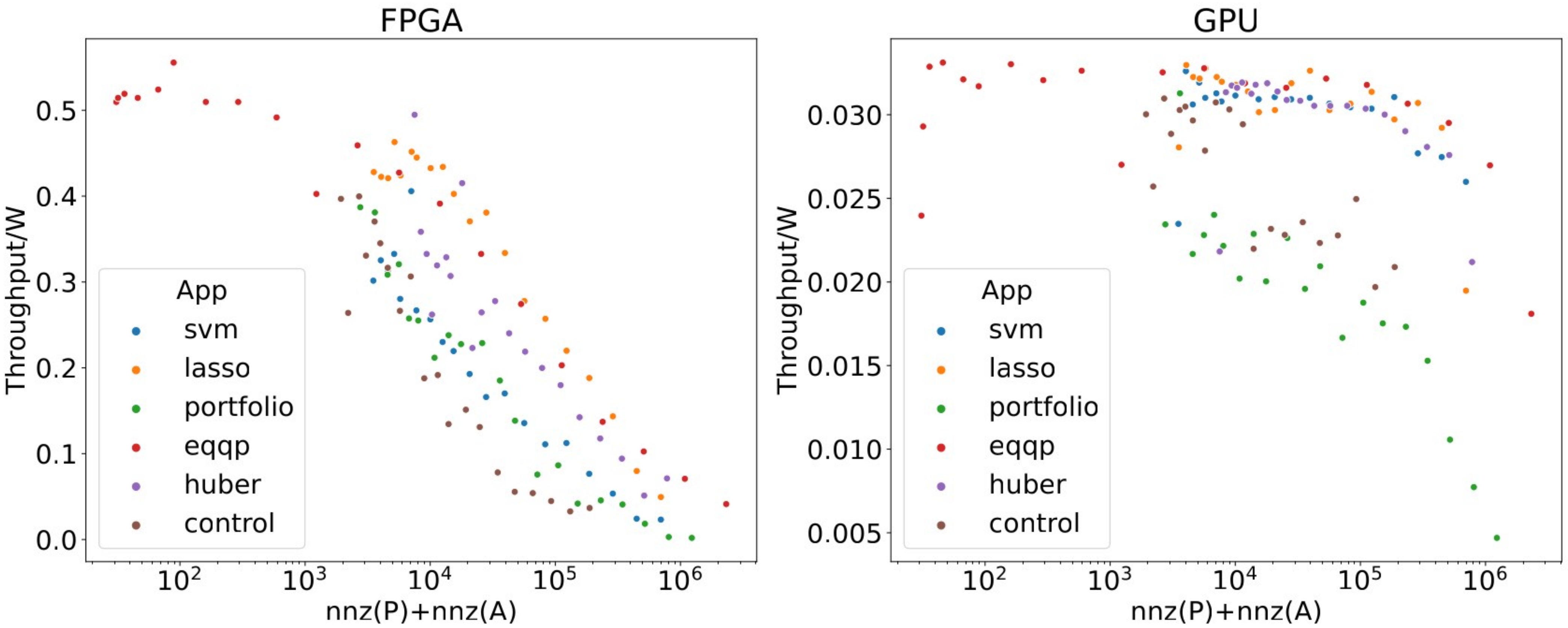
(b) Deployment Flow

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

OSQP on GPUs

cuOSQP

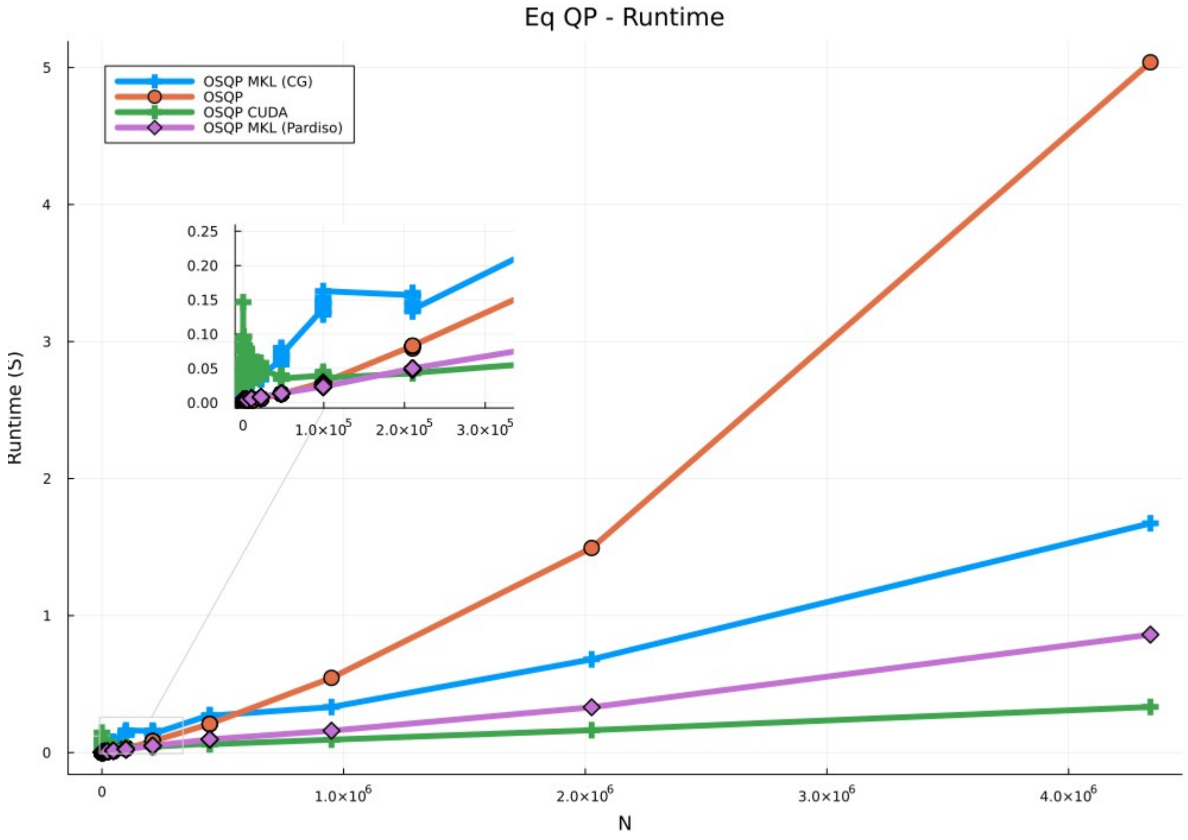
cuOSQP - Overview

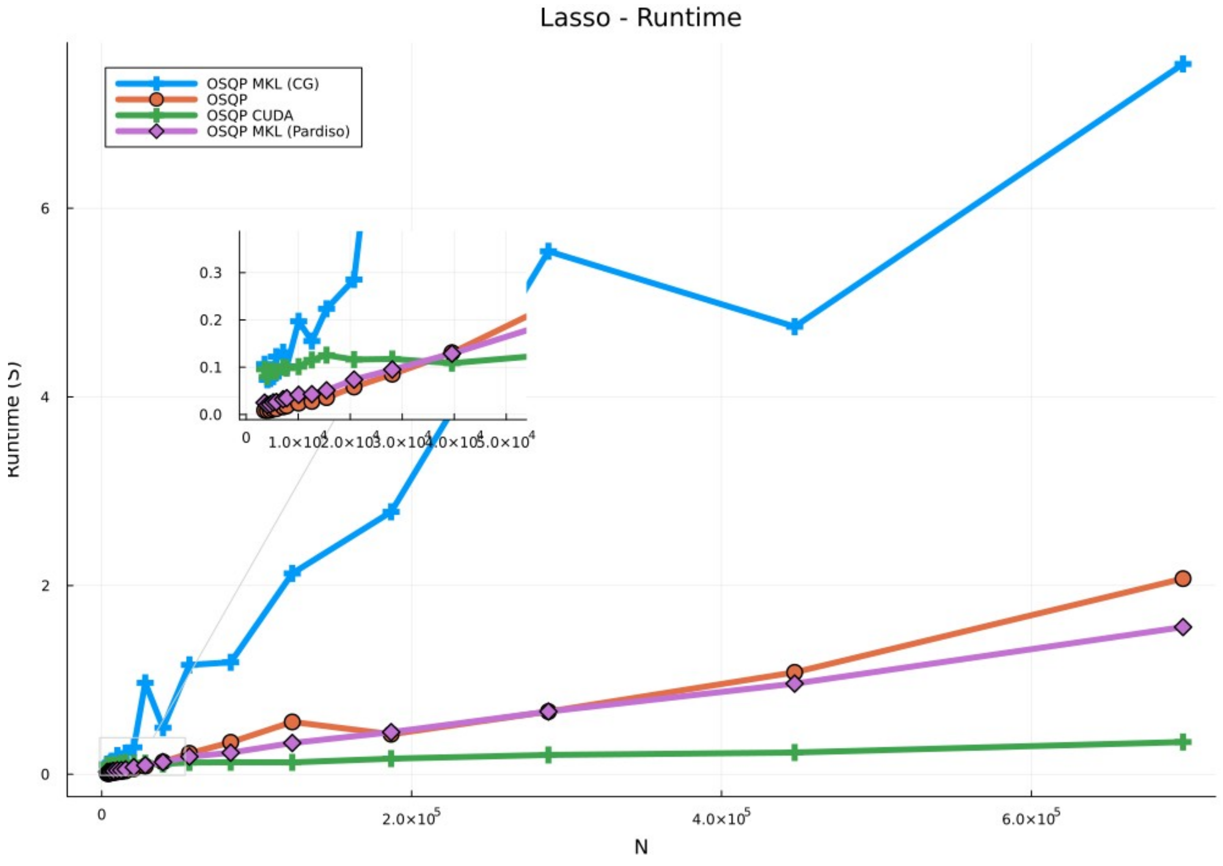
- Implements OSQP Indirect
  - Preconditioned CG linear system solver w/ tapered termination
  - Uses reduced KKT system
- Exact same API as default built-in algebra backend
  - Can drop-in/re-link OSQP to get GPU offload
- All data is GPU-resident
  - *osqp_setup* – Data copied to internal OSQP GPU workspace
  - *osqp_solve* – CPU-managed control flow, only transfer status values

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

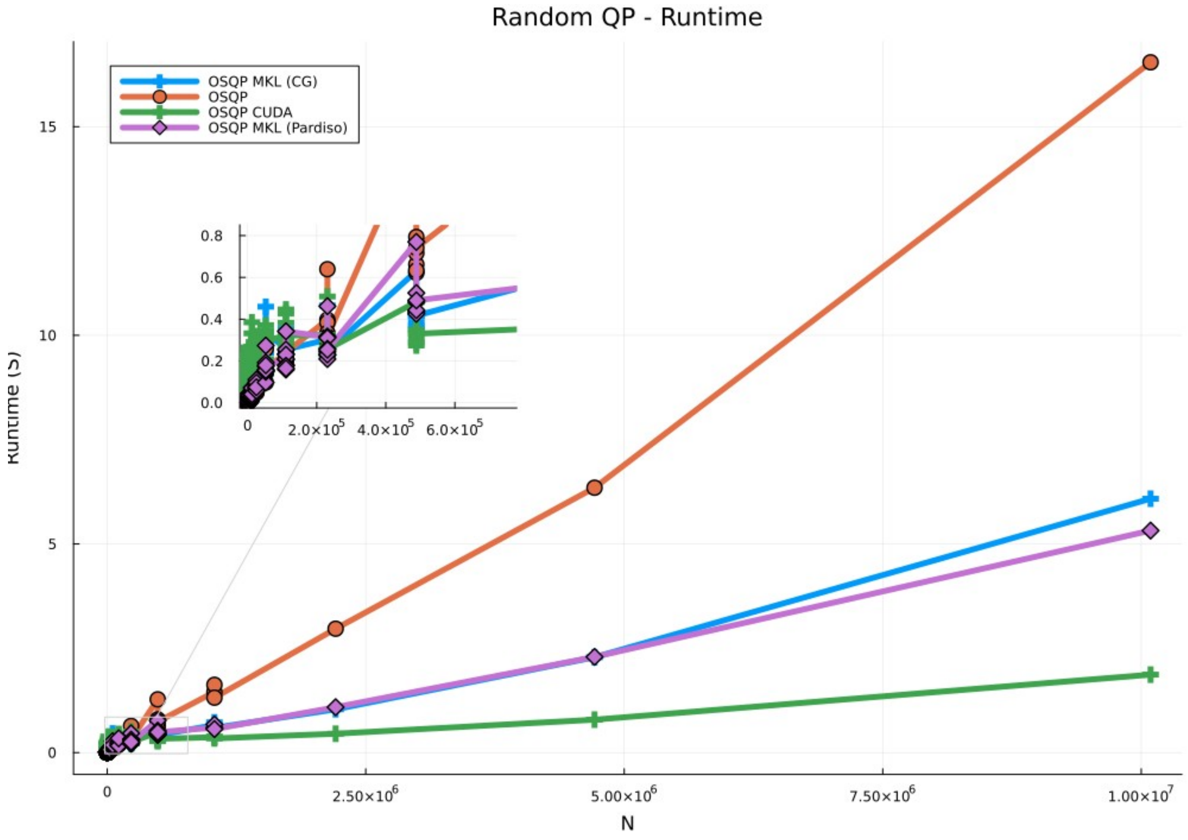
Numerical Example – Runtimes

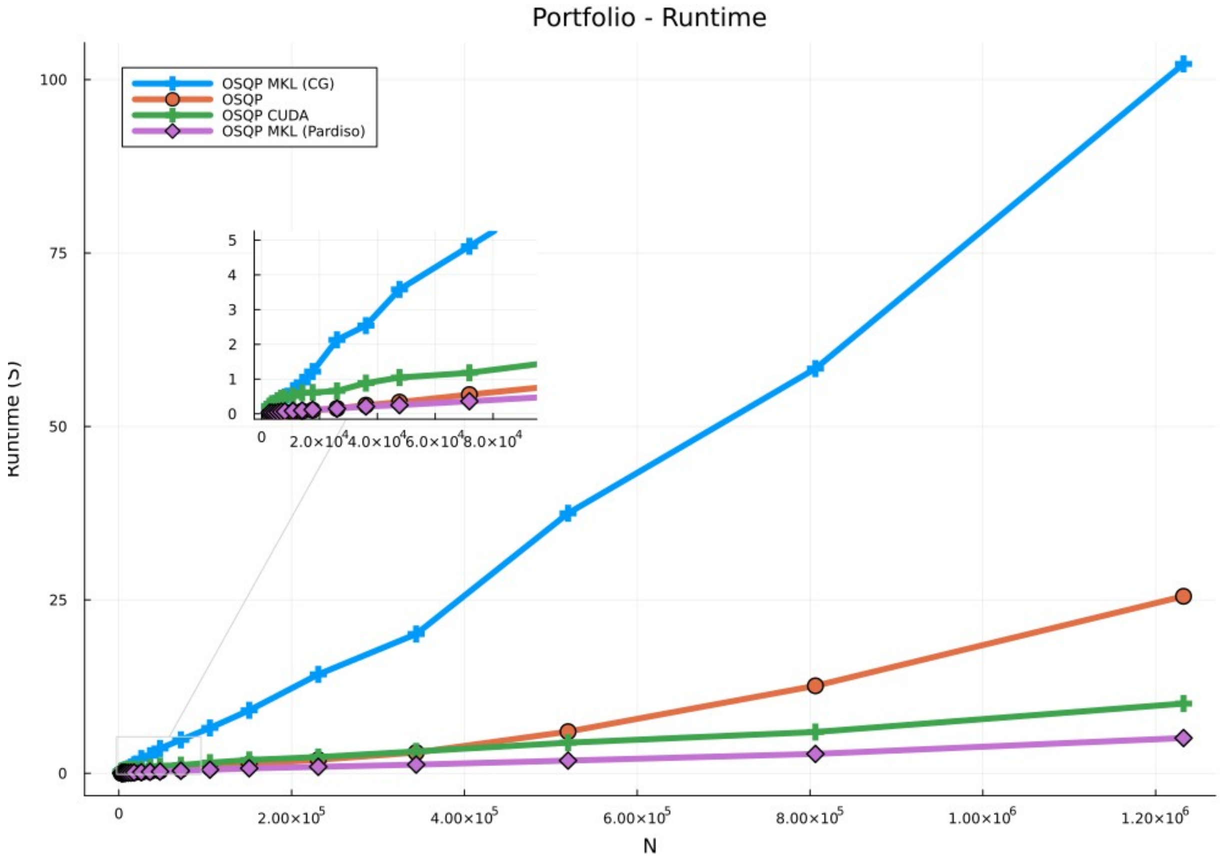




Cuda: NVidia T1000 4GB
CPU: Intel Xeon W-2255 @ 3.7GHz

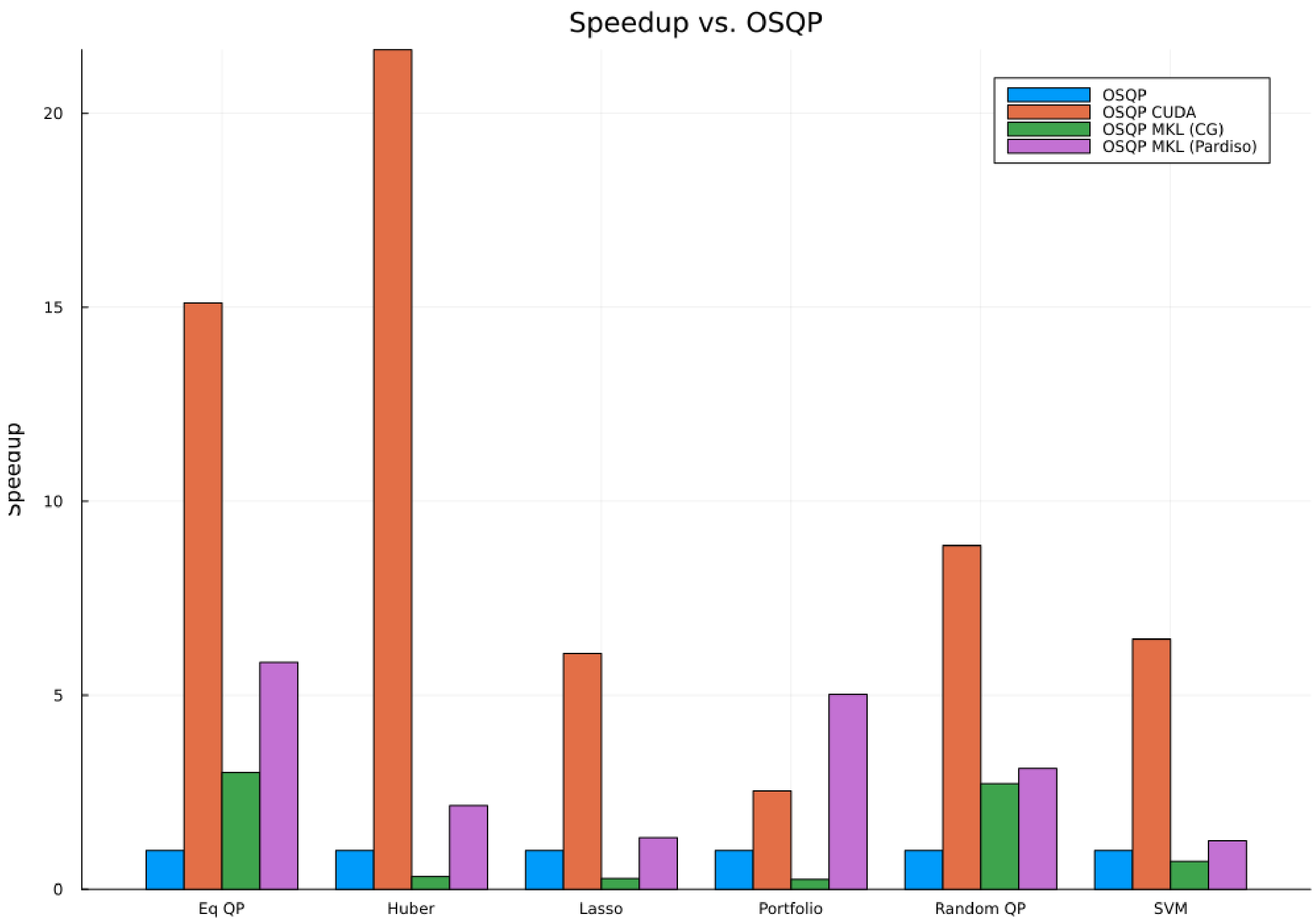
Numerical Example – Runtimes





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

Numerical Example – Speedup



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
  - MINRES solver

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).*