Compile-time Is the New Constexpr

Leveraging Compile-time Sparsity for Vectors and Matrices

Daniel Withopf

2023

C++ now

Last year´s talk

Physical units in vectors and matrices*

If it compiles, it works!

* Design principle

- Move information to compile-time in order to *detect problematic usages and prevent them at compile-time*

Today

Compile-time sparsity for matrices*

If it compiles, it‘s efficient!

* Design principle

- Move information to compile-time in order to *save memory and run-time and make sparse linear algebra code efficient by design*

Daniel Withopf | 2023-05-03

Overview

- Recap: what is a TypeSafeMatrix?
  - Moves semantic information to compile-time (physical units, coordinate frame)
  - Moves all error handling to compile-time
- (Compile-time) sparse matrices in C++
  - Moving sparsity from run-time to compile-time
- Loops
  - Moving iteration from run-time to compile-time

- Experiments
  - Can we get a free lunch by moving calculations from run-time to compile-time?

Named index structs

Identify each entry with a unique name: **D**istance**X**

```
struct DX :
CartesianIdxType<VehicleRearAxleCoords,
                 tsm::CartesianXAxis,
                 si::Metre>{};
```

coordinate frame
axis identifier
si unit template

Physical quantity	Axis	Coordinate frame
Distance	X	_SENSOR
Distance	Y	_SENSOR
Velocity	X	_SENSOR
Velocity	Y	_SENSOR
Acceler.	X	_SENSOR
Acceler.	Y	_SENSOR

```
struct DY_SENSOR :
CartesianIdxType<SensorCoords,
                 tsm::CartesianYAxis,
                 si::Metre>{};
```


Daniel Withopf | 2023-05-03

One type for almost everything

```cpp
template<class Scalar, class RowList, class ColList, class MatrixTag>
class TypeSafeMatrix {
  ... // methods
  private:
    Eigen::Matrix<Scalar, SizeOf<RowList>::value, SizeOf<ColList>::value> m_matrix;
};
```

```cpp
tsm::TypeSafeMatrix<double, tsm::TypeList<DX, VX>, tsm::TypeList<DX, VX>,
                    tsm::CovarianceMatrixTag> covariance{ ... };
```

```cpp
// access physical quantity of an entry
si::Metre2PerSecond<double> quantity = covariance.coeffSi<DX, VX>();
```

```cpp
// how to read / write a plain scalar
covariance.at<DX, VX>() = other_covariance.at<DX, VX>();
```


Daniel Withopf | 2023-05-03

Important classes

Building blocks:

```
template<class Derived> class MatrixBase{};
```

```
template<class Promotion,    // infers resulting TypeSafeMatrix
         class LinalgExpression> // Eigen expression
class MatrixExpression : public
  MatrixBase<MatrixExpression<Promotion, LinalgExpression>>{};
```

```
template<class Scalar, class RowList, class ColList, class MatrixTag>
class TypeSafeMatrix : public MatrixBase<
    TypeSafeMatrix<ScalarT, RowList, ColList, MatrixTagT>>{};
```


Daniel Withopf | 2023-05-03

Important classes

Building blocks:

```cpp
template<class Derived> class MatrixBase{};
```

```cpp
template<class Promotion,    // infers resulting TypeSafeMatrix
         class LinalgExpression> // Eigen sparse expression
class MatrixExpression : public
  MatrixBase<MatrixExpression<Promotion, LinalgExpression>>{};
```

```cpp
template<class Scalar, class RowList, class ColList, class MatrixTag, class Functor>
class TypeSafeMatrix : public MatrixBase<
    TypeSafeMatrix<ScalarT, RowList, ColList, MatrixTag, Functor>>{};
```


Daniel Withopf | 2023-05-03

Matrix multiplication

```
cov_sensor = jacobian * cov_vehicle * jacobian.transpose();
```

$$
1\;
\begin{array}{c|c}
Jac & \begin{array}{cccc} \boxed{-1} \\ DX & DY & VX & VY \end{array} \\
\hline
\begin{array}{c} DX_{SEN} \\ DX_{SEN} \\ VX_{SEN} \\ VY_{SEN} \end{array} &
\begin{pmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 1.0 \end{pmatrix}
\end{array}
\; * 1\; 1 \;
\begin{array}{c|c}
Cov & \begin{array}{cccc} \boxed{1} \\ DX & DY & VX & VY \end{array} \\
\hline
\begin{array}{c} DX \\ DY \\ VX \\ VY \end{array} &
\begin{pmatrix} 3.1 & 0 & 0 & 0 \\ 0 & 2.4 & 0 & 0 \\ 0 & 0 & 8.5 & 0 \\ 0 & 0 & 0 & 6.4 \end{pmatrix}
\end{array}
\; * \; -1 \;
\begin{array}{c|c}
Jac^T & \begin{array}{cccc} 1 \\ DX_{SEN} & DY_S & VX_S & VY_{SEN} \end{array} \\
\hline
\begin{array}{c} DX \\ DY \\ VX \\ VY \end{array} &
\begin{pmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 1.0 \end{pmatrix}
\end{array}
$$

Daniel Withopf | 2023-05-03

BOSCH

Sparse matrices

Different kinds of sparsity

- Run-time sparse matrices
  - Sparseness information is only known at run-time
  - Only covers entries that are zero

- Compile-time sparse matrices
  - Sparseness information is known at compile-time due to physical constraints / modelling choices
  - Our goal: represent entries that are zero, one or remapped to another

(Compile-time) sparse matrices

1. Matrix shapes

- Diagonal matrix

$$\begin{pmatrix} * & 0 & 0 & 0 \\ 0 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & * \end{pmatrix}$$

- Upper triangular matrix

$$\begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \end{pmatrix}$$

- Upper uni-triangular matrix

$$\begin{pmatrix} 1 & * & * & * \\ 0 & 1 & * & * \\ 0 & 0 & 1 & * \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Symmetric matrix

$$\begin{pmatrix} * & a & b & c \\ a & * & d & e \\ b & d & * & f \\ c & e & f & * \end{pmatrix}$$

$*$ Entry with any run-time value
a Entry with storage
a Entry w/o storage, remapped
0 Compile-time 0 entry
1 Compile-time 1 entry

Daniel Withopf | 2023-05-03

(Compile-time) sparse matrices

2. Sparse matrices

- Diagonal block matrix

$$\begin{pmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{pmatrix}$$

$$\begin{pmatrix} R & & 0 & 0 \\ & & 0 & 0 \\ 0 & 0 & R & \\ 0 & 0 & & \end{pmatrix} * \begin{pmatrix} DX \\ DY \\ VX \\ VY \end{pmatrix}$$

- General sparse matrices

$$\begin{pmatrix} a & b & 0 & 0 \\ b & c & 0 & 0 \\ 0 & 0 & d & e \\ 0 & 0 & e & f \end{pmatrix}$$

$$\begin{array}{c|c} Cov & \begin{matrix} DX & DY & VX & VY \end{matrix} \\ \hline \begin{matrix} DX \\ DY \\ VX \\ VY \end{matrix} & \begin{pmatrix} a & b & 0 & 0 \\ b & c & 0 & 0 \\ 0 & 0 & d & e \\ 0 & 0 & e & f \end{pmatrix} \end{array}$$

Daniel Withopf | 2023-05-03

Real-world use-case

- State transition in Kalman filter $DX' = DX + \Delta t * VX$

$$
\begin{array}{c|cccc}
Jac & DX & DY & VX & VY \\
\hline
\begin{array}{c} DX' \\ DY' \\ VX' \\ VY' \end{array} & \multicolumn{4}{c}{\begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}
\end{array}
* \begin{pmatrix} DX \\ DY \\ VX \\ VY \end{pmatrix}
$$

- Measurement matrix in Kalman filter

$$
\begin{array}{c|cccccccc}
 & DX & DY & DZ & VX & VY & AX & AY & OR \\
\hline
\begin{array}{c} DX_S \\ DY_S \\ VX_S \\ VY_S \end{array} & \multicolumn{8}{c}{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 1 & 0 & 0 & 0 & 0 \\ * & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}}
\end{array}
$$

Daniel Withopf | 2023-05-03

(Run-time) sparse matrix representation

- Sparse matrix representation in Eigen

$$\begin{pmatrix} \boxed{\begin{matrix}0\\22\\7\\0\\0\end{matrix}} & \begin{matrix}3\\0\\5\\0\\0\end{matrix} & \begin{matrix}0\\0\\0\\0\\14\end{matrix} & \begin{matrix}0\\0\\1\\0\\0\end{matrix} & \begin{matrix}0\\17\\0\\0\\8\end{matrix} \end{pmatrix}$$

$$\begin{array}{lcccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ values = & 22 & 7 & 3 & 5 & 14 & 1 & 17 & 8 \\ row\ idx = & 1 & 2 & 0 & 2 & 4 & 2 & 1 & 4 \end{array}$$

$$column\ start\ idx = 0 \quad 2 \quad 4 \quad 5 \quad 6 \quad 8$$

Daniel Withopf | 2023-05-03

BOSCH

Today´s goal

Make all sparse matrix operations efficient

$$A \quad * \quad B \quad = \quad C$$

$$\begin{pmatrix} \boxed{a_1 & a_2 & 0 & 1} \\ a_3 & a_4 & 0 & 0 \\ 0 & 0 & a_5 & a_6 \\ 0 & 0 & a_7 & a_8 \end{pmatrix} * \begin{pmatrix} b_1 & b_2 & b_3 & b_4 \\ b_5 & b_6 & b_7 & b_8 \\ b_9 & b_{10} & b_{11} & b_{12} \\ b_{13} & b_{14} & b_{15} & b_{16} \end{pmatrix} = \begin{pmatrix} \boxed{c_1} & c_2 & c_3 & c_4 \\ c_5 & c_6 & c_7 & c_8 \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{14} & c_{15} & c_{16} & c_{17} \end{pmatrix}$$

$$\boxed{\begin{matrix} a_1 \\ a_2 \\ 0 \\ 1 \end{matrix}} \begin{matrix} * \\ * \\ * \\ * \end{matrix} \boxed{\begin{matrix} b_1 \\ b_5 \\ b_9 \\ b_{13} \end{matrix}}$$

$$c_1 = a_1 * b_1 + a_2 * b_5 + b_{13}$$

Daniel Withopf | 2023-05-03

Translation to C++

```
template<typename... Pairs,        // Pairs contain the cartesian cross-product of rows and cols in C
         typename InnerIdxList>    // Inner index list for multiplication loop (cols of A and rows of B)
struct MatrixMultiplication<TypeList<Pairs...>, InnerIdxList>
{
  template<typename Left, typename Right, typename Out>
  static constexpr void run(Left&& A, Right&& B, Out&& C)
  {
    (..., (C.template at<typename Pairs::First, typename Pairs::Second>() =
              CalculateEntry<typename Pairs::First,  // row index of A
                             typename Pairs::Second, // col index of B
                             InnerIdxList>::run(A, B)));
  }
};
template<typename Row, typename Col, typename... InnerIdxs>
struct CalculateEntry<Row, Col, TypeList<InnerIdxs...>>
{
  template <typename Left, typename Right>
  static constexpr double run(Left&& A, Right&& B)
  {
    return (0 + ... + MultiplyTwoEntries<Row, InnerIdxs, Col>::run(A, B));
  }
};
```

No run-time loops

Daniel Withopf | 2023-05-03

BOSCH

Translation to C++

```cpp
template<typename Row, typename InnerIdx, typename Col>
struct MultiplyTwoEntries {
    template<typename Left, typename Right>
    static constexpr double run(Left&& A, Right&& B) {
        using LhsFunctor = typename std::decay_t<Left>::FunctorType;
        using RhsFunctor = typename std::decay_t<Right>::FunctorType;
        if constexpr (isZero(LhsFunctor::template getEntryKind<Row, InnerIdx>()) or
                      isZero(RhsFunctor::template getEntryKind<InnerIdx, Col>()) {
            return 0.;
        } else if constexpr (isOne(LhsFunctor::template getEntryKind<Row, InnerIdx>()) and
                             isOne(RhsFunctor::template getEntryKind<InnerIdx, Col>())) {
            return 1.;
        } else if constexpr (isOne(LhsFunctor::template getEntryKind<Row, InnerIdx>())
            return B.template at<InnerIdx, Col>();
        } else if constexpr (isOne(RhsFunctor::template getEntryKind<InnerIdx, C
            return A.template at<Row, InnerIdx>();
        } else {
            return A.template at<Row, InnerIdx>() * B.template at<InnerIdx, Co
        }
    }
};
```

If it compiles, it's efficient!

Focus on fold expression

```
template<typename... Pairs,        // Pairs contain the cartesian cross-product of rows and cols in C
         typename InnerIdxList>    // Inner index list for multiplication loop (cols of A and rows of B)
struct MatrixMultiplication<TypeList<Pairs...>, InnerIdxList>
{
  template<typename Left, typename Right, typename Out>
  static constexpr void run(Left&& A, Right&& B, Out&& C)
  {
    (..., (C.template at<Row, Col>() =
               CalculateEntry<typename Pairs::First, typename Pairs::Second, // row and c
                              RemoveZeros</*...*/, InnerIdxList>>::run(A, B)));
  }
};

template<typename Row, typename Col, typename... InnerIdxs>
struct CalculateEntry<Row, Col, TypeList<InnerIdxs...>>
{
  template <typename Left, typename Right>
  static constexpr double run(Left&& A, Right&& B)
  {
    return (0. + ... + MultiplyTwoEntries<Row, InnerIdxs, Col>::run(A, B));
  }
};
```

If it compiles, it's efficient!

Daniel Withopf | 2023-05-03

BOSCH

Element access

```
template<class Scalar, class RowList, class ColList, class MatrixTag, class Functor>
class TypeSafeMatrix {
  using NonTrivialElements = SelectNonTrivialElements<CrossProduct<RowList, ColList>>;
  template <typename RowIdx, typename ColIdx>
  constexpr double at() const
  {
    if constexpr (isZero(Functor::template getEntryKind<RowIdx, ColIdx>()))
      return 0.;
    else if constexpr (isOne(Functor::template getEntryKind<RowIdx, ColIdx>()))
      return 1.;
    else if constexpr (Functor::template isEntryRemapped<RowIdx, ColIdx>()) {
      using RemappedTo = FunctorRemappingTo<Functor, RowIdx, ColIdx>;
      using RemappedRow = typename RemappedTo::FirstType;
      using RemappedCol = typename RemappedTo::SecondType;
      constexpr int idx = index_of<Pair<RemappedRow, RemappedCol>, NonTrivialElements>;
      return m_data[idx];
    } else {
      constexpr int idx = index_of<Pair<RowIdx, ColIdx>, NonTrivialElements>;
      return m_data[idx];
    }
  }
};
```


Daniel Withopf | 2023-05-03

Defining a sparsity functor

```cpp
template<typename Rows, typename Cols>
struct SymmetricMatrixFunctor {
  template<typename RowIdx, typename ColIdx>
  static constexpr bool EntryKind getEntryKind() { return EntryKind::kNormal; // kZero, kOne }

  template<typename RowIdx, typename ColIdx>
  using RemappedIdxPair = Pair<ColIdx, RowIdx>;

  template<typename RowIdx, typename ColIdx>
  static constexpr bool isEntryRemapped() {
    if constexpr (index_of<RowIdx, Rows> > index_of<ColIdx, Cols>) { return true; }
    else { return false; }
  }

  template<typename T>
  using InverseFunctor = T;

  template<typename Row1, typename Col1, typename Row2, typename Col2>
  static constexpr bool isContentIdentical =
    isEntryRemapped<Row2, Col2> && is_same_v<Pair<Row1, Col1>, RemappedIdxPair<Row2, Col2>> ||
    isEntryRemapped<Row1, Col1> && is_same_v<Pair<Row2, Col2>, RemappedIdxPair<Row1, Col1>>;
};
```


Daniel Withopf | 2023-05-03

Determine the remapping of a compound expression

$$\begin{array}{cc} & \begin{array}{ccc} DX & DY & DZ \end{array} \\ \begin{array}{c} Idx0 \\ Idx1 \end{array} & \begin{pmatrix} a & a & a \\ b & b & b \end{pmatrix} \end{array}$$

$\Uparrow$ ().transpose()

$$\begin{array}{cc} & \begin{array}{cc} Idx0 & Idx1 \end{array} \\ \begin{array}{c} DX[m] \\ DY[m] \\ DZ[m] \end{array} & \begin{pmatrix} a & b \\ a & b \\ a & b \end{pmatrix} \end{array}$$

`auto res = (1.0[s] * mat).transpose();`

* 1.0[s] $\Uparrow$

$$\text{mat} = \begin{array}{cc} & \begin{array}{cc} Idx0 & Idx1 \end{array} \\ \begin{array}{c} VX\left[\frac{m}{s}\right] \\ VY\left[\frac{m}{s}\right] \\ VZ\left[\frac{m}{s}\right] \end{array} & \begin{pmatrix} a & b \\ a & b \\ a & b \end{pmatrix} \end{array}$$

Daniel Withopf | 2023-05-03

Determine the mapping of a compound expression

$$\begin{array}{cc} & \begin{matrix} DX & DY & DZ \end{matrix} \\ \begin{matrix} Idx0 \\ Idx1 \end{matrix} & \begin{pmatrix} a & a & a \\ b & b & b \end{pmatrix} \end{array}$$

().transpose()

$$\begin{array}{cc} & \begin{matrix} Idx0 & Idx1 \end{matrix} \\ \begin{matrix} DX[m] \\ DY[m] \\ DZ[m] \end{matrix} & \begin{pmatrix} a & b \\ a & b \\ a & b \end{pmatrix} \end{array}$$

/ 1.0[s]

$$\text{mat} = \begin{array}{cc} & \begin{matrix} Idx0 & Idx1 \end{matrix} \\ \begin{matrix} VX[\frac{m}{s}] \\ VY[\frac{m}{s}] \\ VZ[\frac{m}{s}] \end{matrix} & \begin{pmatrix} a & b \\ a & b \\ a & b \end{pmatrix} \end{array}$$

$$\begin{array}{cc} & \begin{matrix} DX & DY & DZ \end{matrix} \\ \begin{matrix} Idx0 \\ Idx1 \end{matrix} & \begin{pmatrix} a & a & a \\ b & b & b \end{pmatrix} \end{array}$$

().transpose()

$$\begin{array}{cc} & \begin{matrix} Idx0 & Idx1 \end{matrix} \\ \begin{matrix} DX[m] \\ DY[m] \\ DZ[m] \end{matrix} & \begin{pmatrix} a & b \\ a & b \\ a & b \end{pmatrix} \end{array}$$

* 1.0[s]

$$\text{mat} = \begin{array}{cc} & \begin{matrix} Idx0 & Idx1 \end{matrix} \\ \begin{matrix} VX[\frac{m}{s}] \\ VY[\frac{m}{s}] \\ VZ[\frac{m}{s}] \end{matrix} & \begin{pmatrix} a & b \\ a & b \\ a & b \end{pmatrix} \end{array}$$

```
auto res = (1.0[s] * mat).transpose().eval();
```

Inverse: / 1.0[s] .transpose()

Daniel Withopf | 2023-05-03

Inverting a sparsity functor

auto res = (1[s] * mat).transpose();

TransposeExprFunctor<MultWithUnitExprFunctor<*StorageFunctor*>>

```
template<typename Functor>
struct TransposeExprFunctor {
  template<typename T>
  using InverseFunctor = typename Functor::template InverseFunctor<TransposeExprFunctor<T>>;
};

TransposeExprFunctor<MultWithUnitExprFunctor<StorageFunctor>>::InverseFunctor<T> =>
  MultWithUnitExprFunctor<StorageFunctor>::InverseFunctor<TransposeExprFunctor<T>> =>
    StorageFunctor::InverseFunctor<DivideByUnitExprFunctor<TransposeExprFunctor<T>>> =>
      DivideByUnitExprFunctor<TransposeFunctor<T>>
```


Daniel Withopf | 2023-05-03

Matrix assignment $A = B$

$$\begin{pmatrix} * & a \\ a & * \end{pmatrix} = \begin{cases} \begin{pmatrix} ? & * \\ * & ? \end{pmatrix} \\ \begin{pmatrix} ? & b \\ b & ? \end{pmatrix} \\ \begin{pmatrix} ? & b \\ b & ? \end{pmatrix} \\ \begin{pmatrix} ? & 0 \\ 0 & ? \end{pmatrix} \\ \begin{pmatrix} 0 & b \\ b & 1 \end{pmatrix} \end{cases}$$

A	Correct?	Efficient?
$\begin{pmatrix} ? & * \\ * & ? \end{pmatrix}$	✗	
$\begin{pmatrix} ? & b \\ b & ? \end{pmatrix}$	✓	✓
$\begin{pmatrix} ? & b \\ b & ? \end{pmatrix}$	✓	
$\begin{pmatrix} ? & 0 \\ 0 & ? \end{pmatrix}$	✓	✗
$\begin{pmatrix} 0 & b \\ b & 1 \end{pmatrix}$	✓	

If it compiles, it works!

If it compiles, it's efficient!

Daniel Withopf | 2023-05-03

BOSCH

„There's No Such Thing as a Free Lunch“

Experiment 1

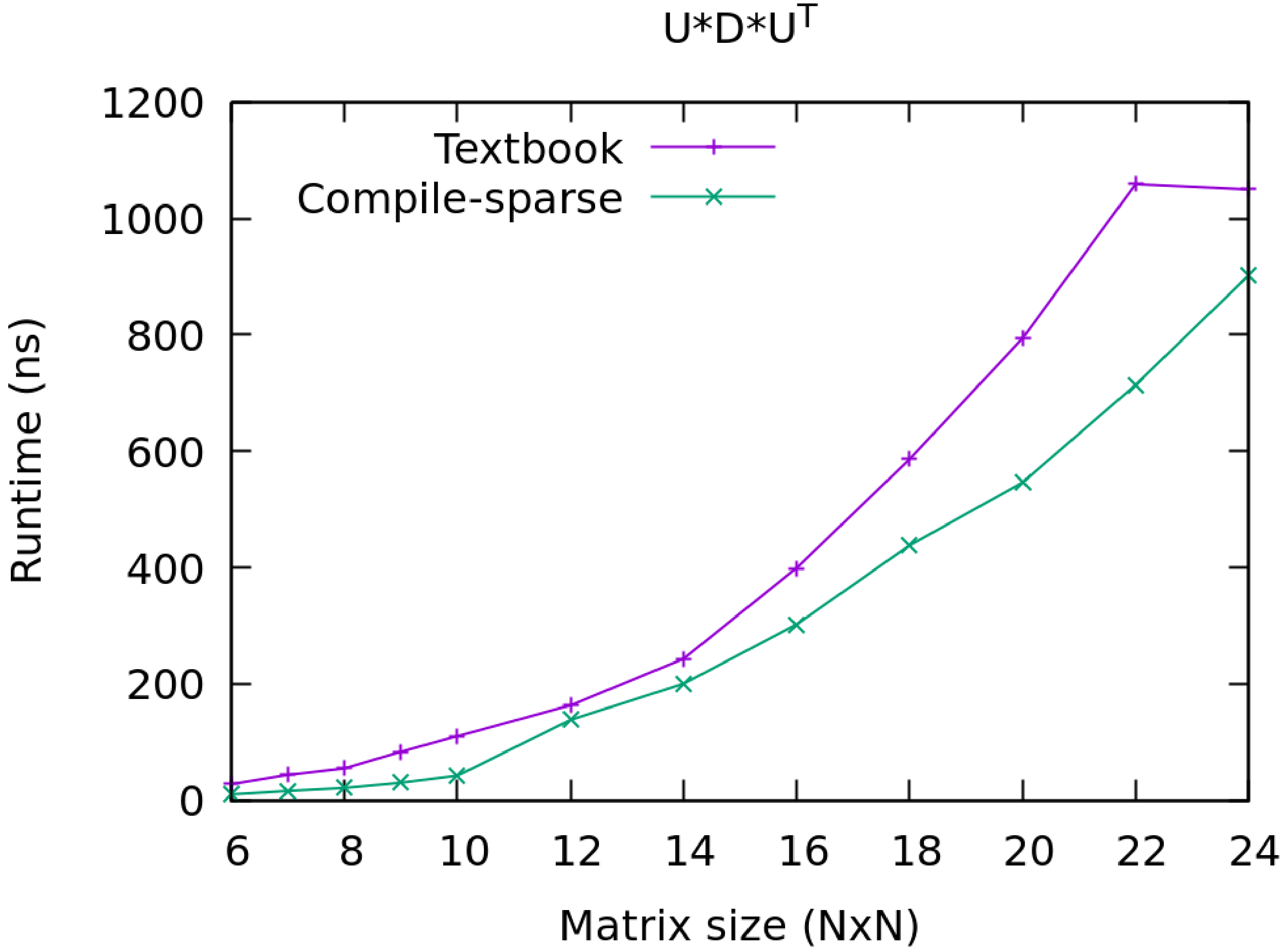
Upper triangular matrix / diagonal matrix multiplication

$$U \quad * \quad D \quad * \quad U^T$$

$$\begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \end{pmatrix} * \begin{pmatrix} * & 0 & 0 & 0 \\ 0 & * & 0 & 0 \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & * \end{pmatrix} * \begin{pmatrix} * & 0 & 0 & 0 \\ * & * & 0 & 0 \\ * & * & * & 0 \\ * & * & * & * \end{pmatrix}$$

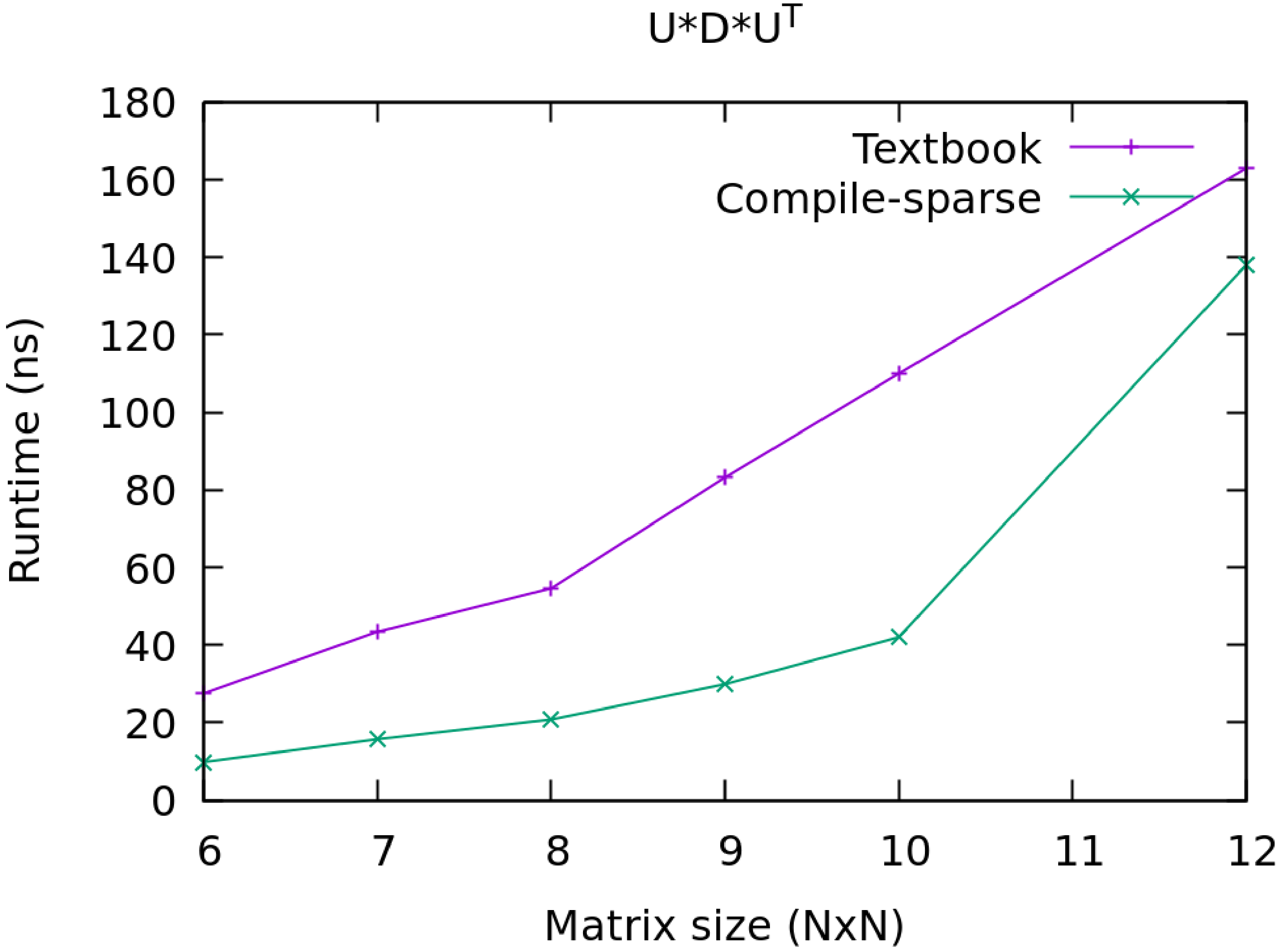
Daniel Withopf | 2023-05-03

Runtime (ns) over matrix dimension (NxN)



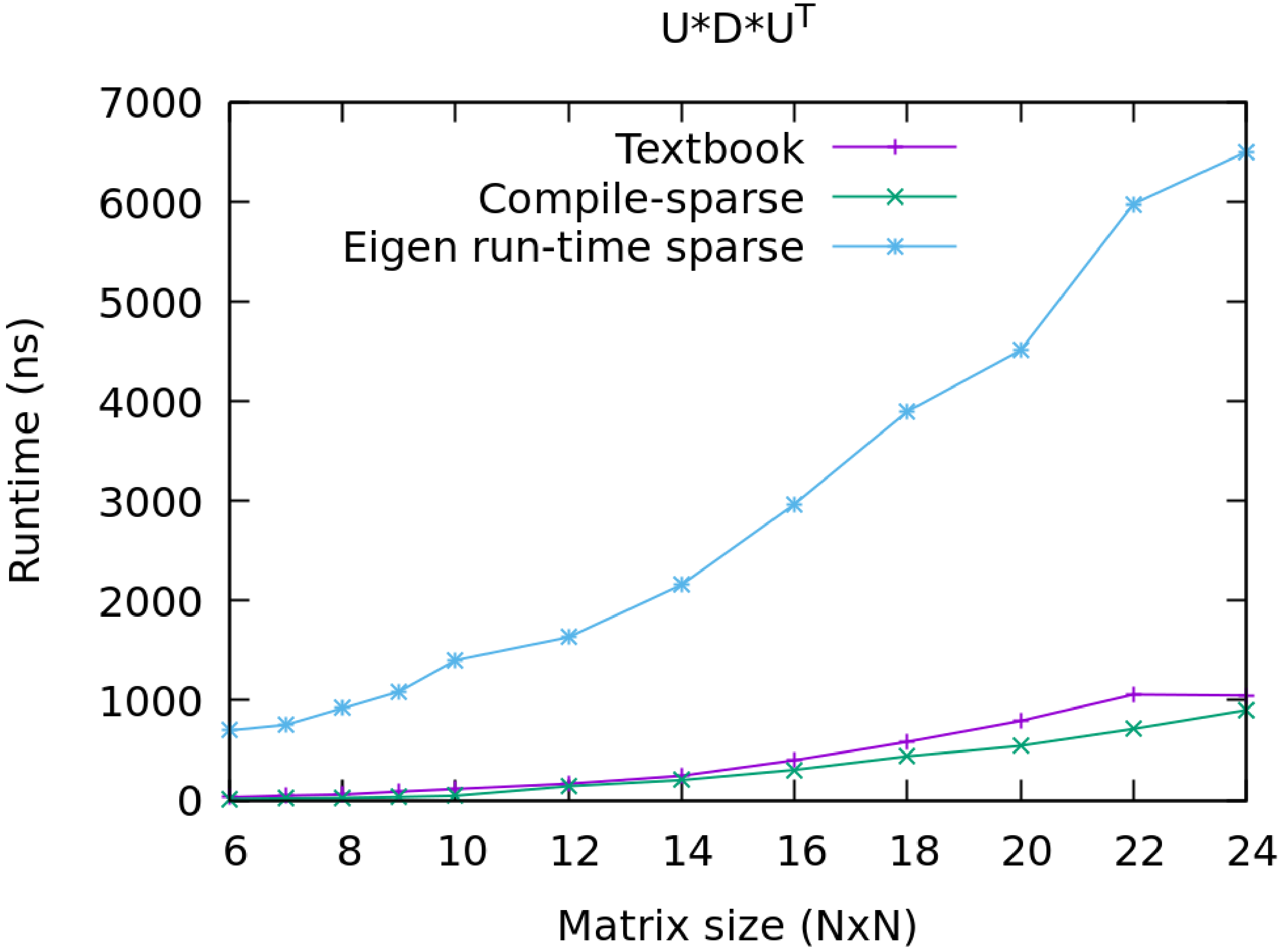
BOSCH

Runtime (ns) over matrix dimension (NxN)



BOSCH

Run-time sparse vs. compile-time sparse



BOSCH

Experiment 2

A*B*A^T with different levels of sparseness for A

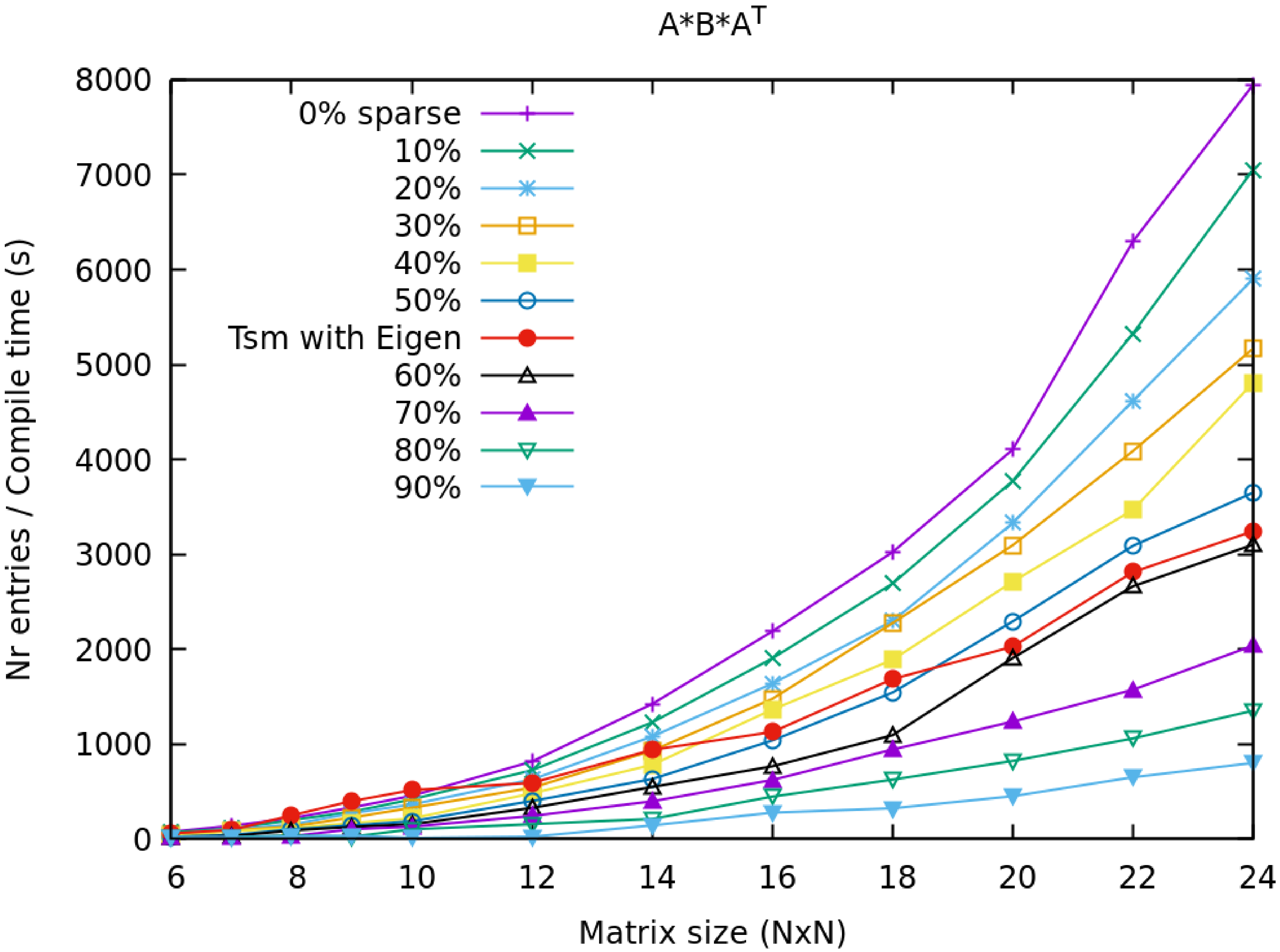
$$A * B * A^T$$

$$\begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ * & 0 & * & * \\ * & * & 0 & * \end{pmatrix} * \begin{pmatrix} a & b & c & d \\ b & e & f & g \\ c & f & h & i \\ d & g & i & j \end{pmatrix} * \begin{pmatrix} * & 0 & * & * \\ * & * & 0 & * \\ * & * & * & 0 \\ * & * & * & * \end{pmatrix}$$

20 % sparse | symmetric | 20 % sparse

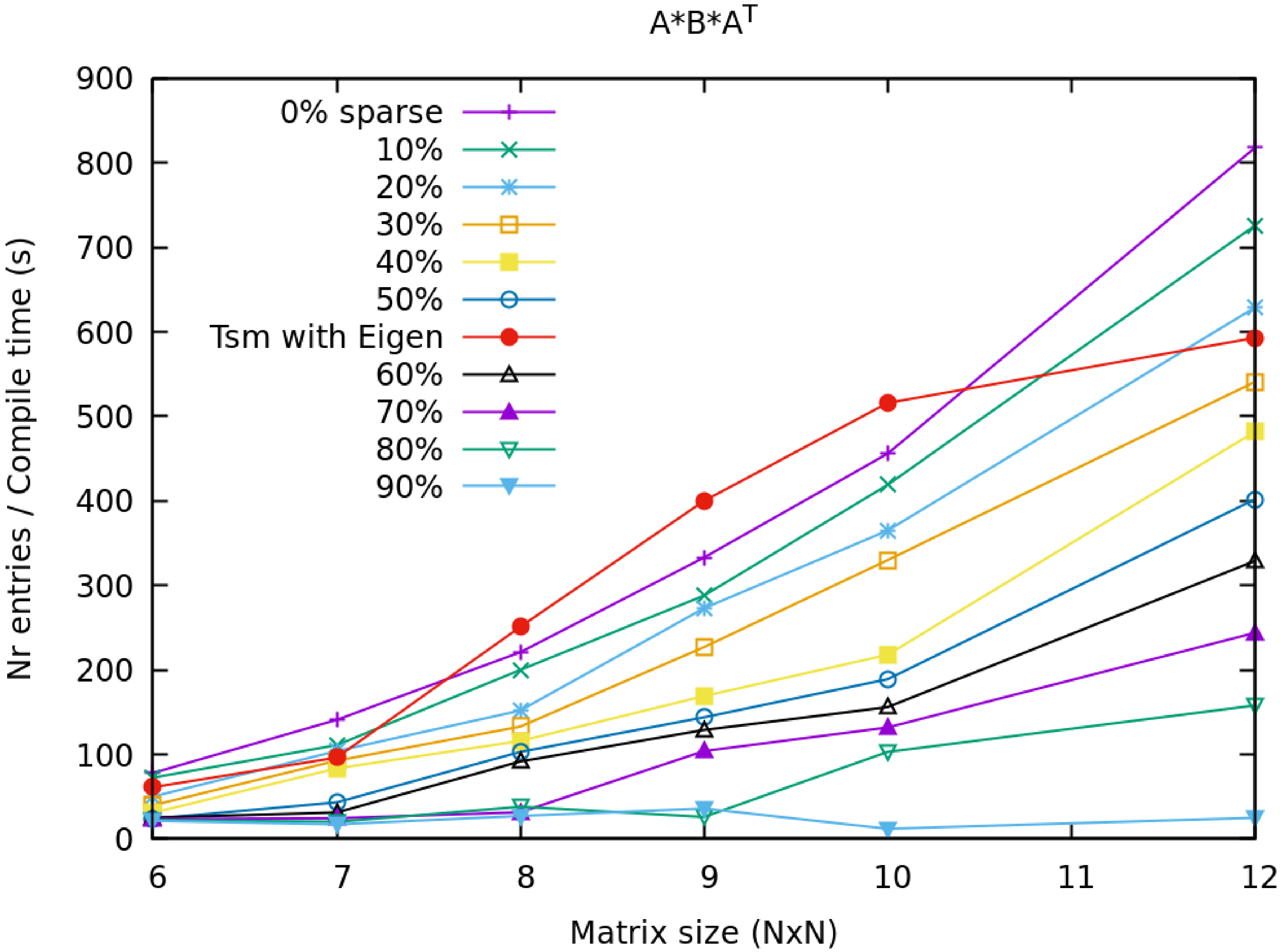
Daniel Withopf | 2023-05-03

A*B*A^T



Daniel Withopf | 2023-05-03

A*B*A^T



Daniel Withopf | 2023-05-03

Experiment 3

Nested sparse multipliations

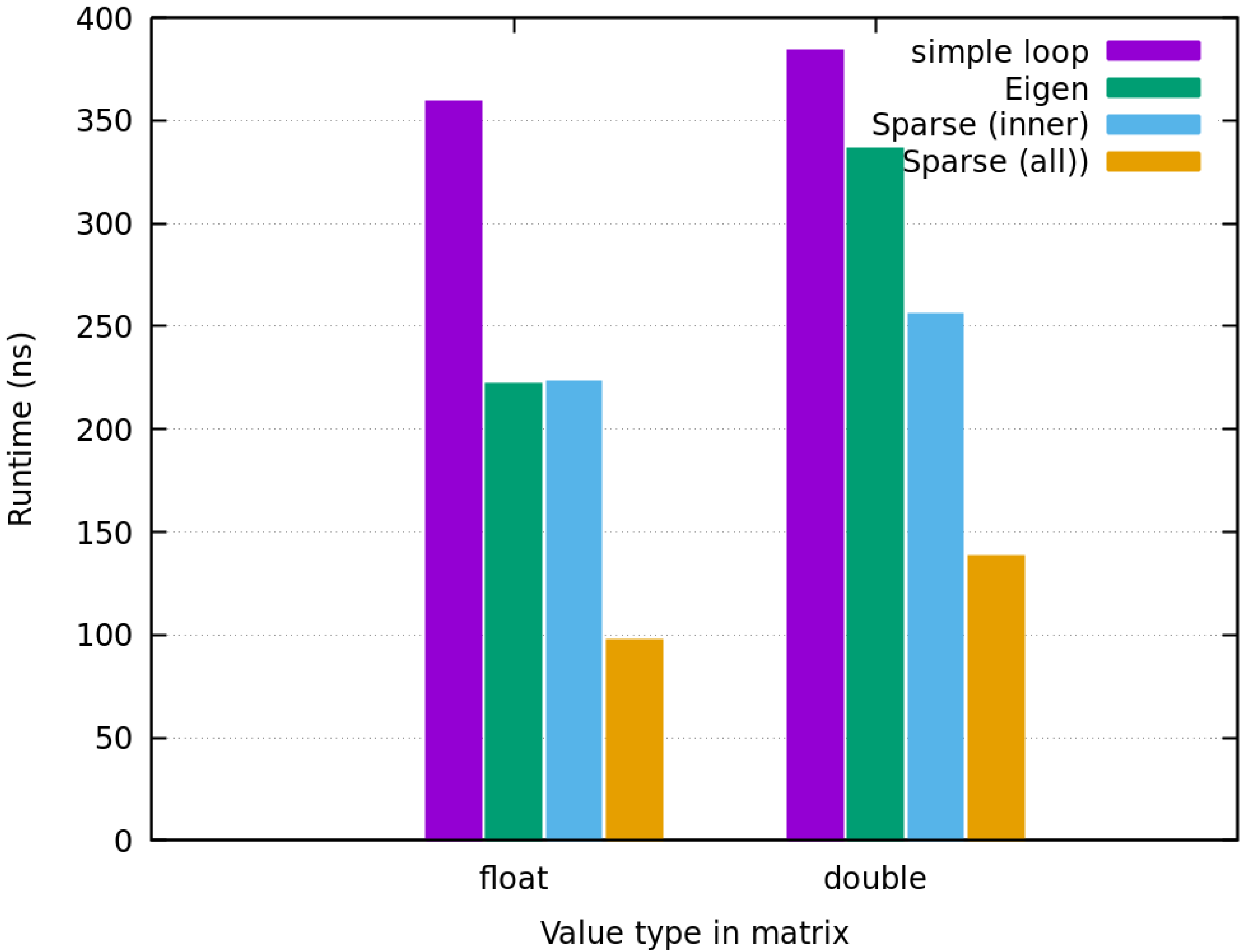
$$(I - gain * meas\_matrix) * cov * (I - gain * meas\_matrix)^T$$

Identity full symmetric

$$\begin{array}{c c} & \begin{array}{cccccccc} Dx & Dy & Dz & Vx & Vy & Ax & Ay & Or \end{array} \\ \begin{array}{c} Dx \\ Dy \\ Vx \\ Vy \end{array} & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 1 & 0 & 0 & 0 & 0 \\ * & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Daniel Withopf | 2023-05-03

Kalman



BOSCH

How is this related to `std::mdspan`?

mdspan layouts

- Map integers (i, j, ...) to a storage location

```
template<class T, class Extents,
         class LayoutPolicy=std::layout_right,
         class Accessor=std::default_accessor>
class mdspan;
```

Access is done via *run-time integers*

```
md_span[i, j] = 5.;
```

Sparsity functors provide information used to

- Map index types (or std::integral_constant) to a storage location

Access is done via *compile-time index types*

```
sparse_matrix.at<RowIdx, ColIdx>() = 5.;
```


Daniel Withopf | 2023-05-03

BOSCH

Summary

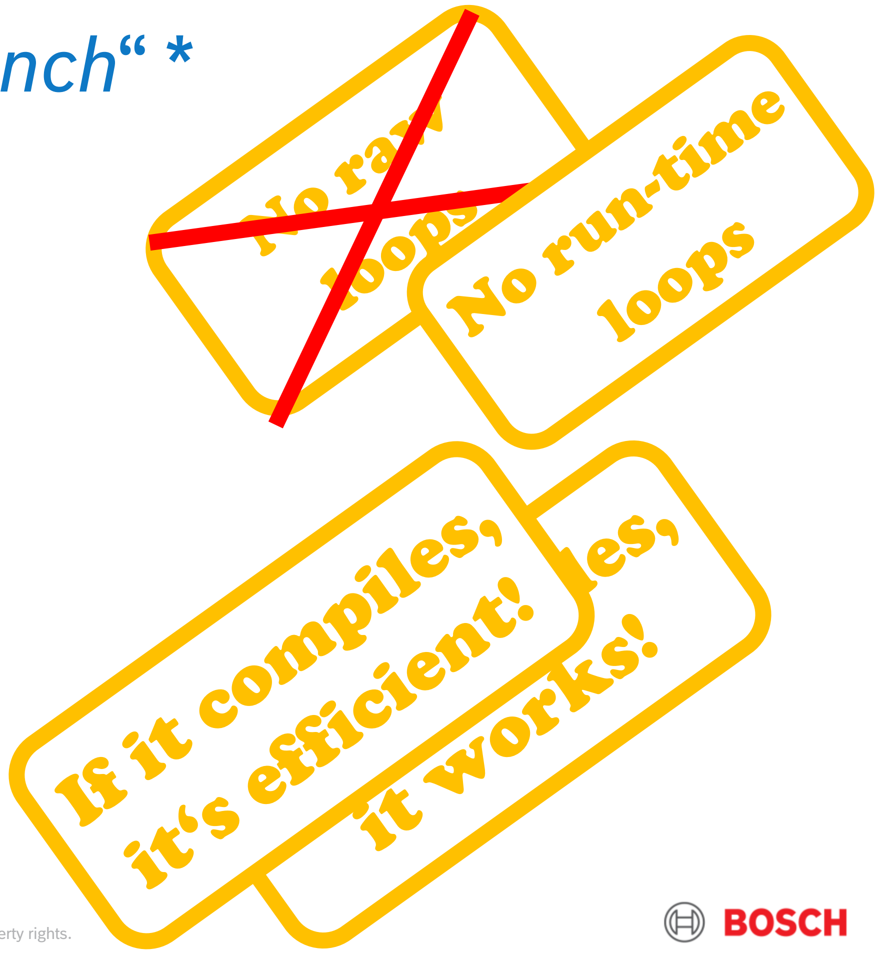
- We´ve combined memory efficiency with run-time efficiency

*„There's ~~No~~ Such a Thing as a Free Lunch" **

- Preconditions
  - Iteration is moved from run-time to compile-time

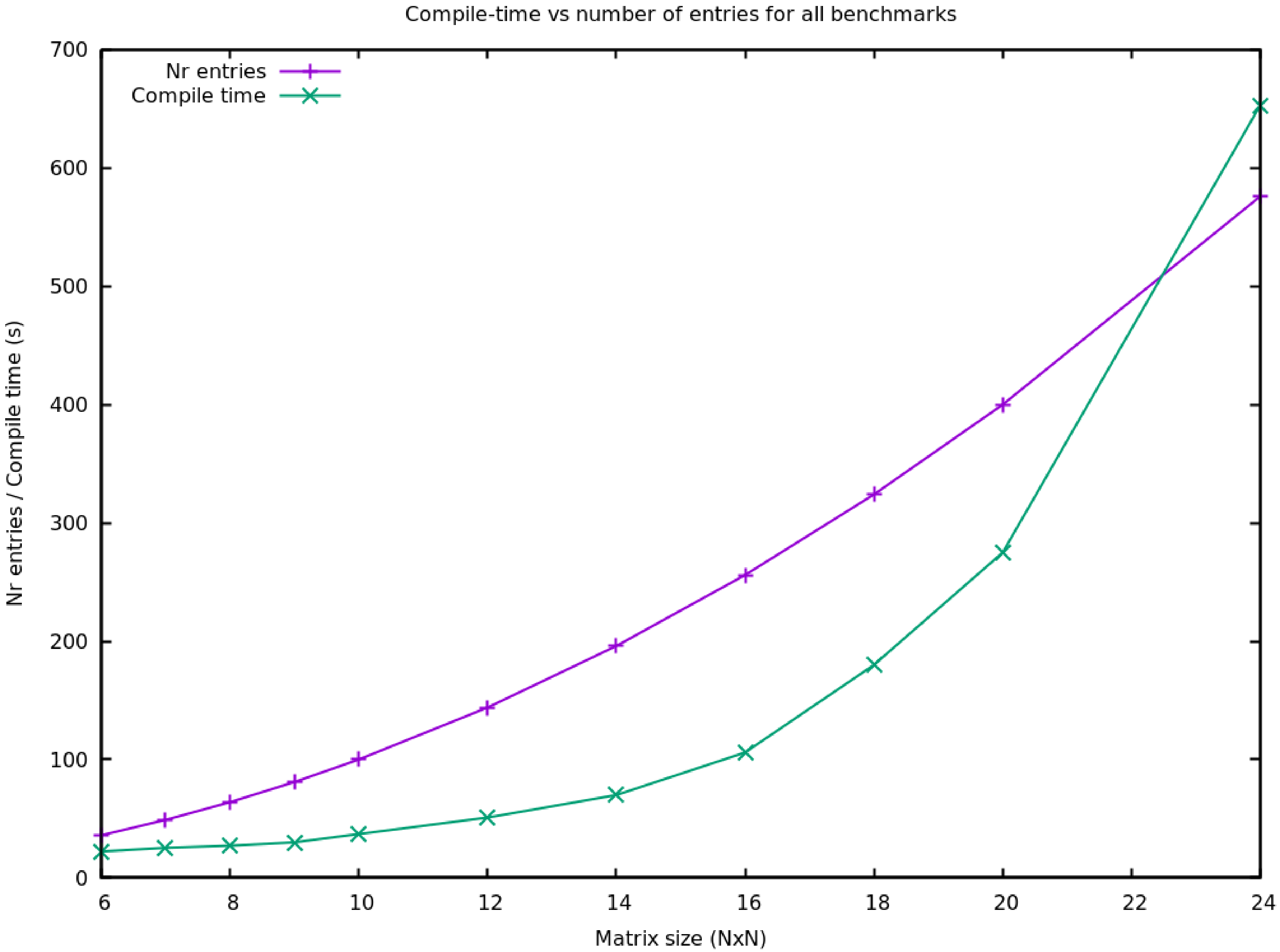
- We can enforce (at compile-time)
  - (Sparsity) correctness
  - Maximum (sparsity) efficiency

* Restrictions apply



Thank you for listening, looking forward to your questions!

Compile time


Compile-time vs number of entries for all benchmarks
Nr entries
Compile time
Nr entries / Compile time (s)
Matrix size (NxN)
0
100
200
300
400
500
600
700
6
8
10
12
14
16
18
20
22
24

BOSCH