Google Summer of Code 2018
Faster Matrix Algebra for ATLAS

Implementing Hermitian Matrices in Eigen

David Tellenbach

Overview

- Eigen
- Rectangular Full Packed Format
- Implementing Rectangular Full Packed Format in Eigen
- Matrix products
- Benchmarks
- Further implementation plans

Eigen

- C++ template library for linear algebra
- Used in a lot of major projects like Google‘s Tensorflow
- Modular structure: Eigen‘s native backend can be replaced by other high performance kernels like Intel‘s Math Kernel Library
- Expressive syntax: `matrix*matrix` instead of `gemm(&a,...)`
- Fast! At least as fast as any open source BLAS/LAPACK library
- Supports lazy evaluation and aliasing
- Heavy usage of static polymorphism (CRTP)

Lazy evaluation

```
int main() {
  Eigen::Matrix3i mat1;
  Eigen::Matrix3i mat2;
  mat1.setRandom();
  mat2.setRandom();
  auto product = mat1 * mat2;
  Eigen::Matrix3i evaluatedProduct = product;
}
```

Rectangular Full Packed Format

- Hermitian matrix completely determined by $\frac{n(n+1)}{2}$ of its elements
- Classical approach: Packed storage

$$\begin{bmatrix} \alpha_{00} & \alpha_{01} & \alpha_{02} \\ & \alpha_{11} & \alpha_{12} \\ & & \alpha_{22} \end{bmatrix} \quad \Rightarrow \quad [\alpha_{00}, \alpha_{01}, \alpha_{02}, \alpha_{11}, \alpha_{12}, \alpha_{22}]$$

- Bad access patterns, bad cache behaviour for complex operations
- Gustavson, Wasniewski, Dongarra, Langou (2009): Rectangular Full Packed Format (RFPF)

Rectangular Full Packed Format

- Distinguish between storage of upper or lower triangular part of the matrix and between even and odd dimension
- Example for storage of upper triangular part and even dimension:

$$\begin{pmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ & & \alpha_{33} & \alpha_{34} \\ & & & \alpha_{44} \end{pmatrix} \Rightarrow \begin{pmatrix} \alpha_{13} & \alpha_{14} \\ \alpha_{23} & \alpha_{24} \\ \alpha_{33} & \alpha_{34} \\ \alpha_{11} & \alpha_{44} \\ \alpha_{12} & \alpha_{22} \end{pmatrix}$$

Implementing Rectangular Full Packed Format in Eigen

- Goal: Hiding storage format and provide expressive (Eigen-like) syntax

```
template<typename _Scalar, int _Dimension, unsigned int _UpLo, int _Storage, int _MaxDimension>
struct traits<HermitianMatrix<_Scalar, _Dimension, _UpLo, _Storage, _MaxDimension, 0> >
  : traits<Matrix<_Scalar, _Dimension, _Dimension, _Storage, _MaxDimension, _MaxDimension> >
{
  typedef Matrix<_Scalar, _Dimension + 1, _Dimension / 2, _Storage,  _MaxDimension + 1, _MaxDimension / 2> NestedExpression;
  typedef _Scalar Scalar;
  typedef Matrix<_Scalar, _Dimension, _Dimension, _Storage, _MaxDimension, _MaxDimension> DenseType;
  typedef HermitianMatrix<_Scalar, _Dimension, _UpLo, _Storage, _MaxDimension> PlainObject;
  typedef HermitianShape StorageKind;
  typedef HermitianXpr XprKind;
  enum {⋯
  };
};
```

Implementing Rectangular Full Packed Format in Eigen

```
int main() {
  Eigen::HermitianMatrix<int, 2> hermitian;
  hermitian << 1, 2,  // 3 elements in memory
               2, 1;
  std::cout << hermitian(1, 0); // 2
  std::cout << hermitian;
}
```

Matrix products

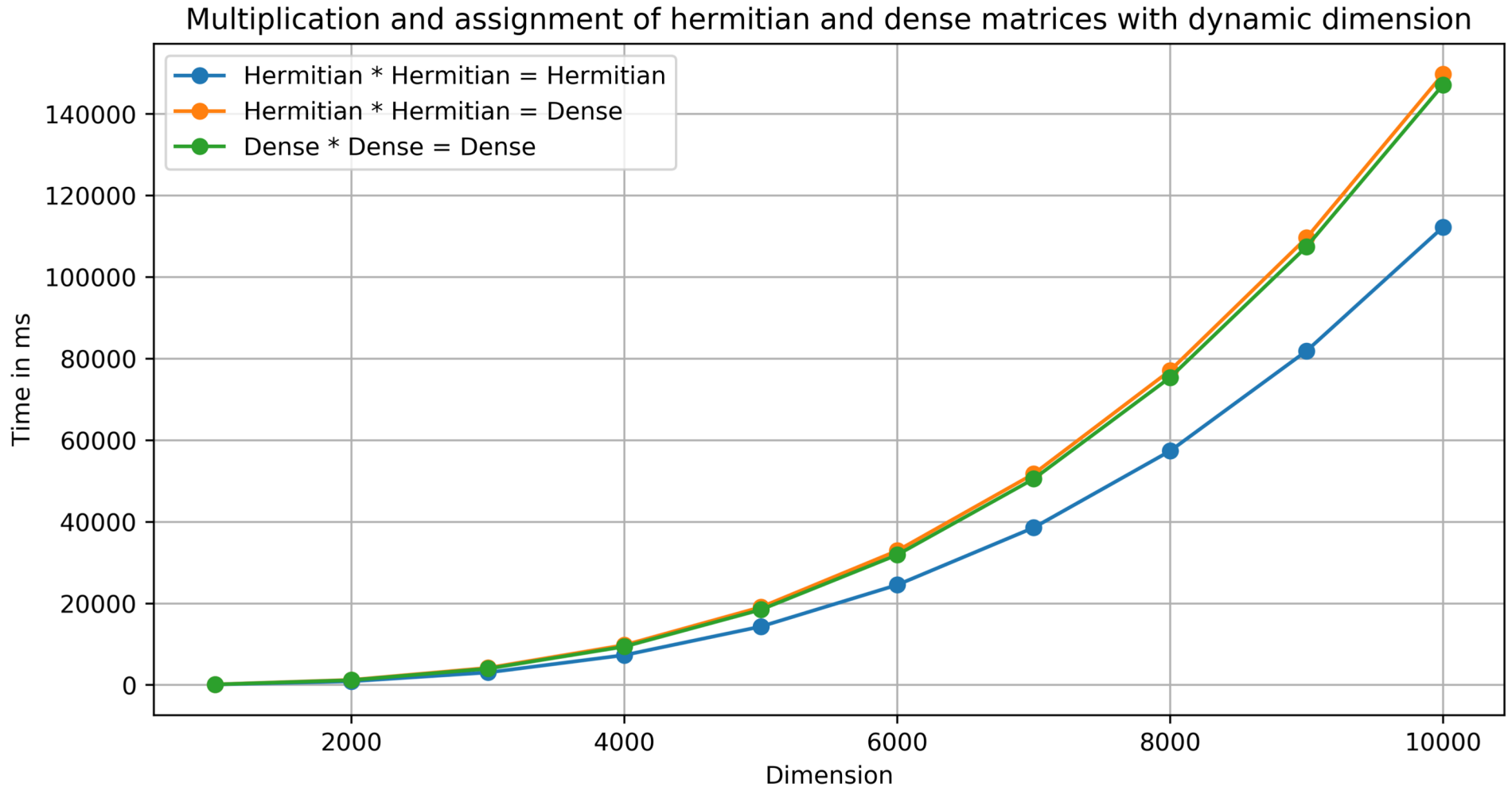
- Eigen provides a highly efficient product implementation that can be replaced by even more efficient backends (Intel MKL)
- Goal: Use existing implementation to multiply hermitian matrices
- 4 cases:
  - Hermitian * Dense = Dense
  - Dense * Hermitian = Dense
  - Hermitian * Hermitian = Dense
  - Hermitian * Hermitian = Hermitian

Matrix products

$$\begin{pmatrix} C_{11} & C_{12} \\ & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^{\mathrm{T}} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{12}^{\mathrm{T}} & B_{22} \end{pmatrix}$$

$$\begin{pmatrix} \gamma_{13} & \gamma_{14} \\ \gamma_{23} & \gamma_{24} \\ \gamma_{33} & \gamma_{34} \\ \gamma_{11} & \gamma_{44} \\ \gamma_{12} & \gamma_{22} \end{pmatrix} = \begin{pmatrix} \alpha_{13} & \alpha_{14} \\ \alpha_{23} & \alpha_{24} \\ \alpha_{33} & \alpha_{34} \\ \alpha_{11} & \alpha_{44} \\ \alpha_{12} & \alpha_{22} \end{pmatrix} \begin{pmatrix} \beta_{13} & \beta_{14} \\ \beta_{23} & \beta_{24} \\ \beta_{33} & \beta_{34} \\ \beta_{11} & \beta_{44} \\ \beta_{12} & \beta_{22} \end{pmatrix}$$

Benchmark


Multiplication and assignment of hermitian and dense matrices with dynamic dimension
Hermitian * Hermitian = Hermitian
Hermitian * Hermitian = Dense
Dense * Dense = Dense
Time in ms
0
20000
40000
60000
80000
100000
120000
140000
2000
4000
6000
8000
10000
Dimension

Further Implementation Plans

- Different product evaluation strategies for different matrix sizes
- Algorithms for hermitian matrices (Cholesky-Factorization)