Eigen

Bo Wang

January 9, 2014

1. What is Eigen?
2. How good is Eigen?
3. How is Eigen implemented?

What is Eigen?

- A library for linear algebra
  - but not limited to linear algebra, e.g. $A = a + B$
- A C++ template library
  - functions and classes need template arguments
- Consists of .h files
  - no .c files
  - no .so and .a
  - need include .h files correctly
- History
  - started in 2008
  - Eigen2 and Eigen3

Eigen is versatile

- Various matrices
  - arbitrary large
  - fixed / non-fixed size
  - with any standard scalar type + custom type
  - dense or sparse matrix
- Vectors
- Arrays
  - for coefficient-wise operations, such as $A = b + B$
- Functions to manipulate matrices, arrays, vectors
  - such as transpose(), sum(), decomposition ...
- Solvers for linear systems
  - i.e. $Ax = b$ using PartialPivLU(), FullPivLU(), ...

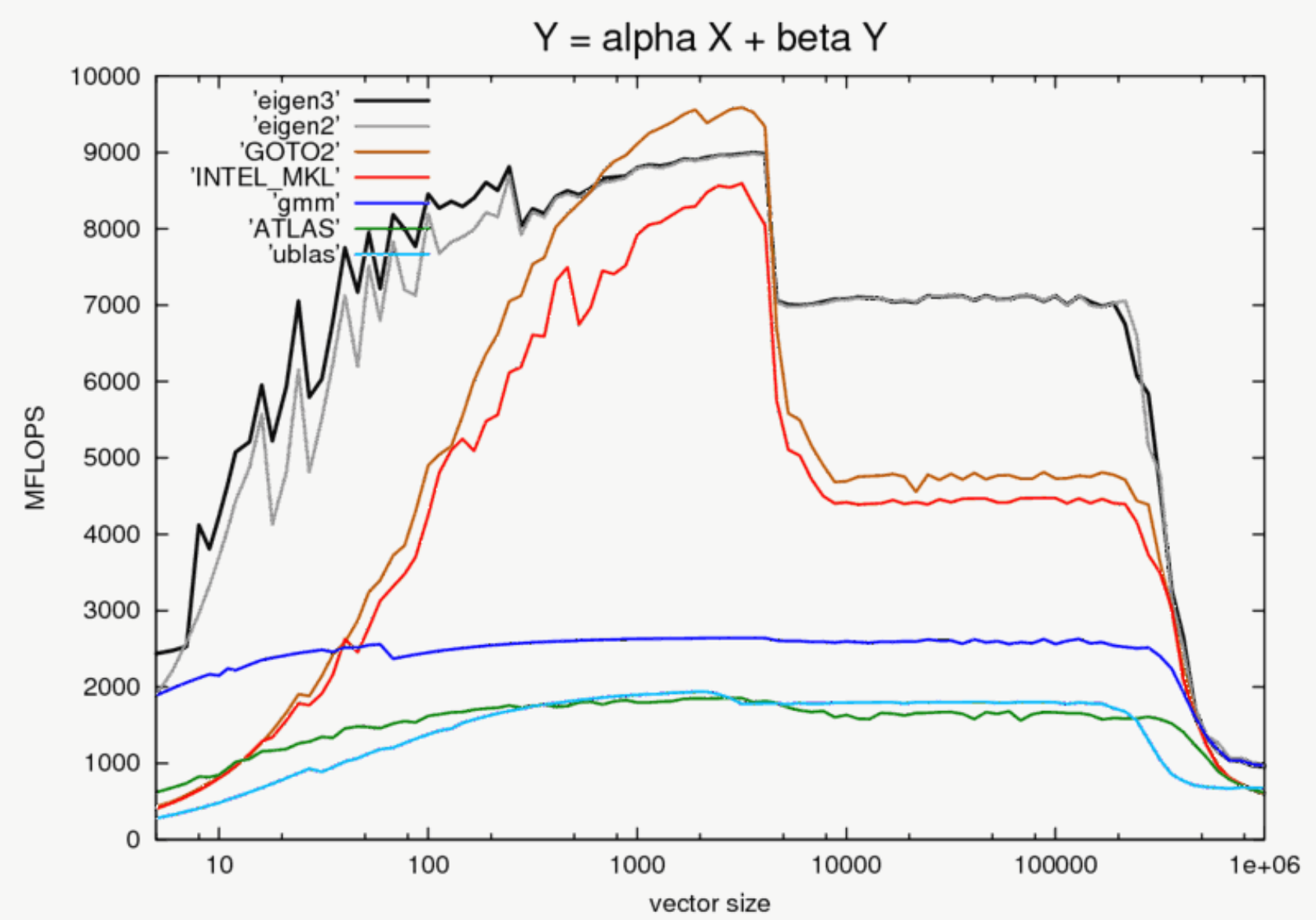
Eigen is easy to use - 1

```
1  // Eigen
2  Matrix<float, 3, 3, ColMajor, 3, 3> m;
3  Matrix3f m;
4  MatrixXf m(3, 3);
5  m << 1,2,3,4,5,6,7,8,9;
6
7  // BLAS in Intel MKL
8  m = (float *) mkl_malloc( size, boundary);
9  for( i=0; i<size; i++) { initialize one element);
```

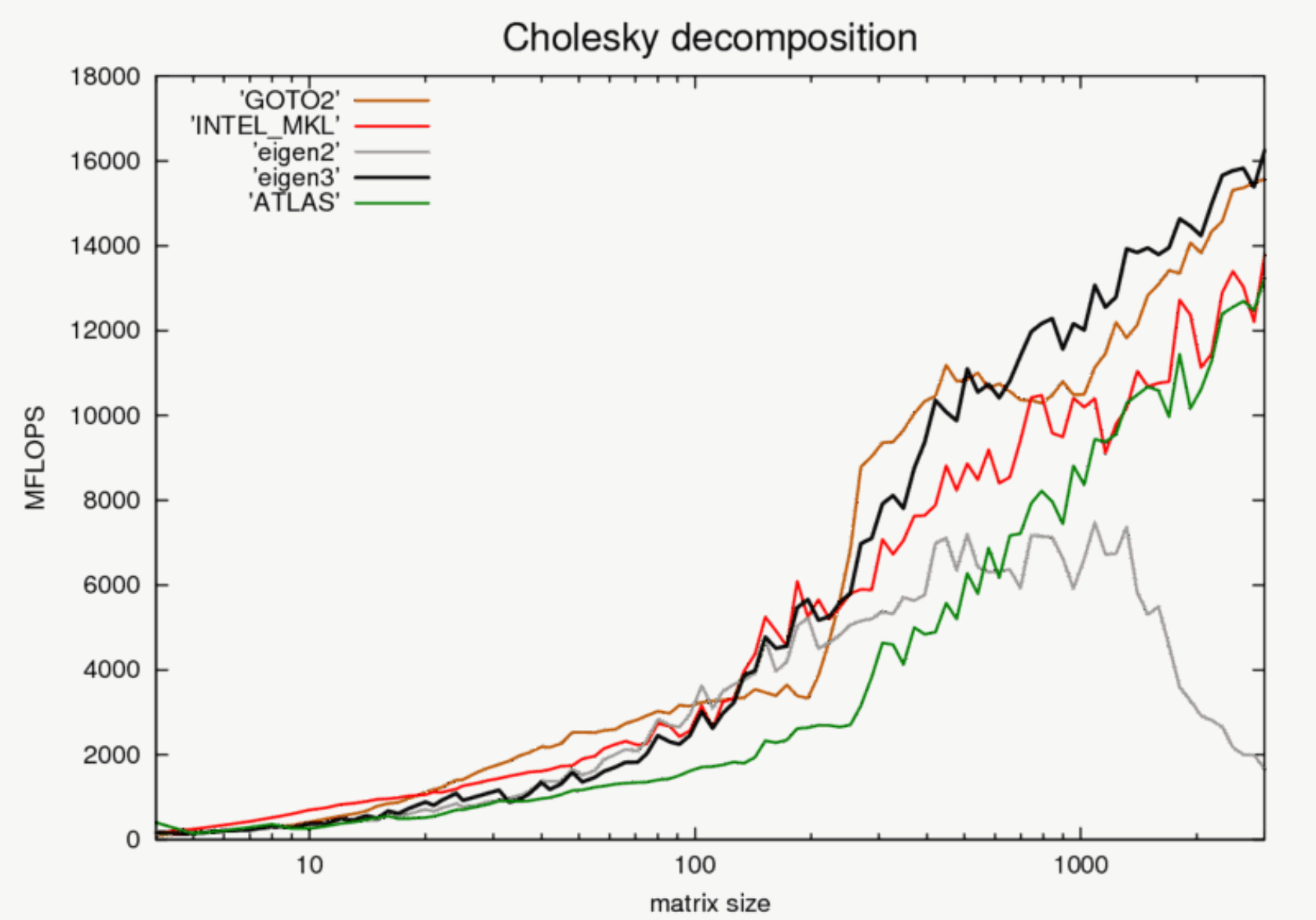
Eigen is easy to use - 2

```
1  // Eigen
2  A = B * C;
3  // solve Ax = b and Ay=c
4  PartialPivLU <Matrix3f> dec(A);
5  x = dec.solve(b);
6  y = dec.solve(c);
7
8  // BLAS in Intel MKL
9  cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTra
10             m, n, k, alpha, A, k, B, n, beta, C, n);
11 LAPACKE_cpptrs(matrix_order, uplo, n, nrhs,
12             *ap, *b, ldb);
```

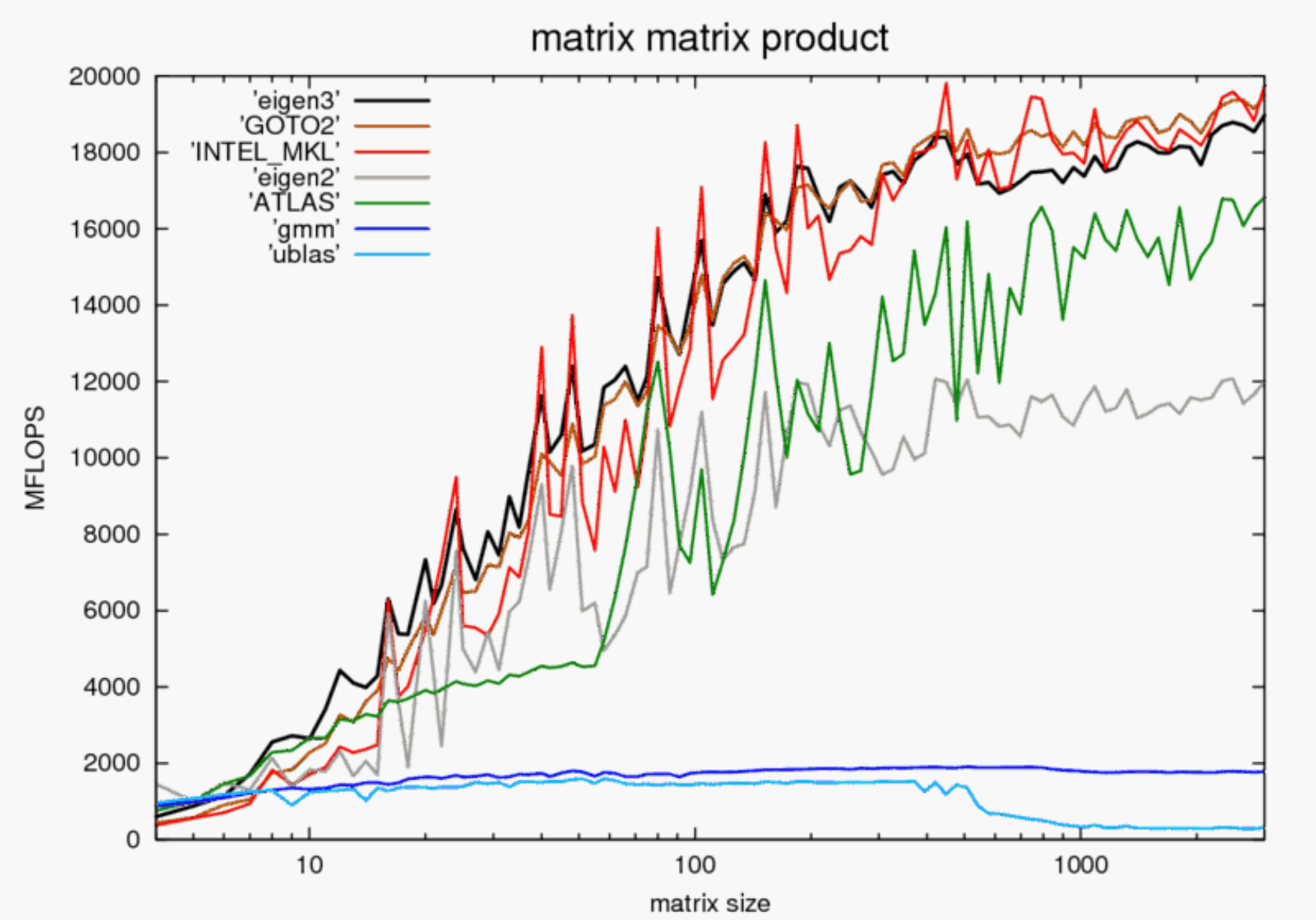
Eigen is fast - 1



Eigen is fast - 2



Eigen is fast - 3



Why is Eigen fast?

- Storage order
  - Row-major for A, Column-major for B in $A \times B$
- Exploit SIMD Infrastructure
  - SSE, SSE2, SSE3, IBM's AltiVec, ARM NEON
- Exploit multi-core processor using OpenMP
  - only some functions are parallelized
- Compute in blocks
- Call functions of other libraries
  - e.g. call functions of Intel MKL
  - like a wrapper, but more than a wrapper
- Two another techniques

Aims of Eigen's design

- all the complexity gets resolved at compile-time, if possible
- avoid unnecessary temporary data
- avoid unnecessary loop

Naive implementation using overloaded operator

$$A = B + C + D$$

- A, B, C, D are matrices
- The best implementation:

```
for i in matrix do:
    A[i] = B[i] + C[i] + D[i]
```

- Naive implementation using overloaded operator
  - After Compilation:

    $T0 = B + C$ // T0 is a temporary matrix
    $T1 = T0 + D$ // T1 is a temporary matrix
    $A = T1$

  - One loop for each statement

$\rightarrow$ unnecessary temporary matrices and unnecessary loops

Eigen's implementation - idea

- Alternative implementation using function calls
  - not well extendable
- Eigen's implementation: Lazy evaluation using expression template
  - An operator is an expression
  - Evaluate an expression as late as possible

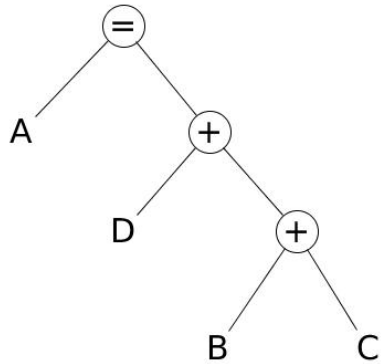
Eigen's implementation - steps 1

- Retrieving
  1. retrieve the right side from left to right, get:

     Sum< Sum< B, C>, D>

  2. retrieve the =, get:

     A.assign(Sum< Sum< B, C>, D> >)



Eigen's implementation - steps 2

- Evaluating

```
A.assign( Sum< Sum< B, C>, D>> )
```

  - assign():

```
for i in Matrix:
    A[i] = Sum< Sum< B, C>, D>.evaluate(i)
```

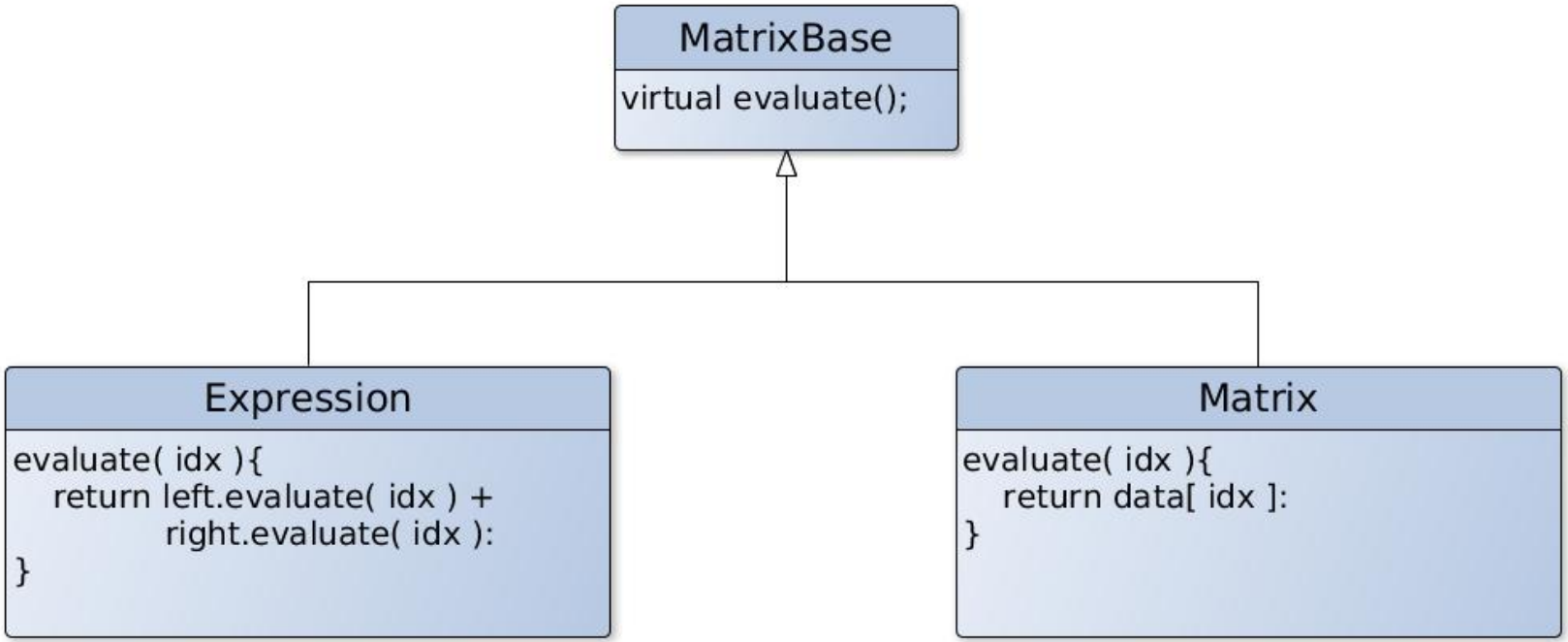
  - Sum:

```
inline evaluate(i):
    return left.evaluate(i) + right.evaluate(i)
```

  - Matrix:

```
inline evaluate(i):
    return data[i]
```

Dynamic polymorphism

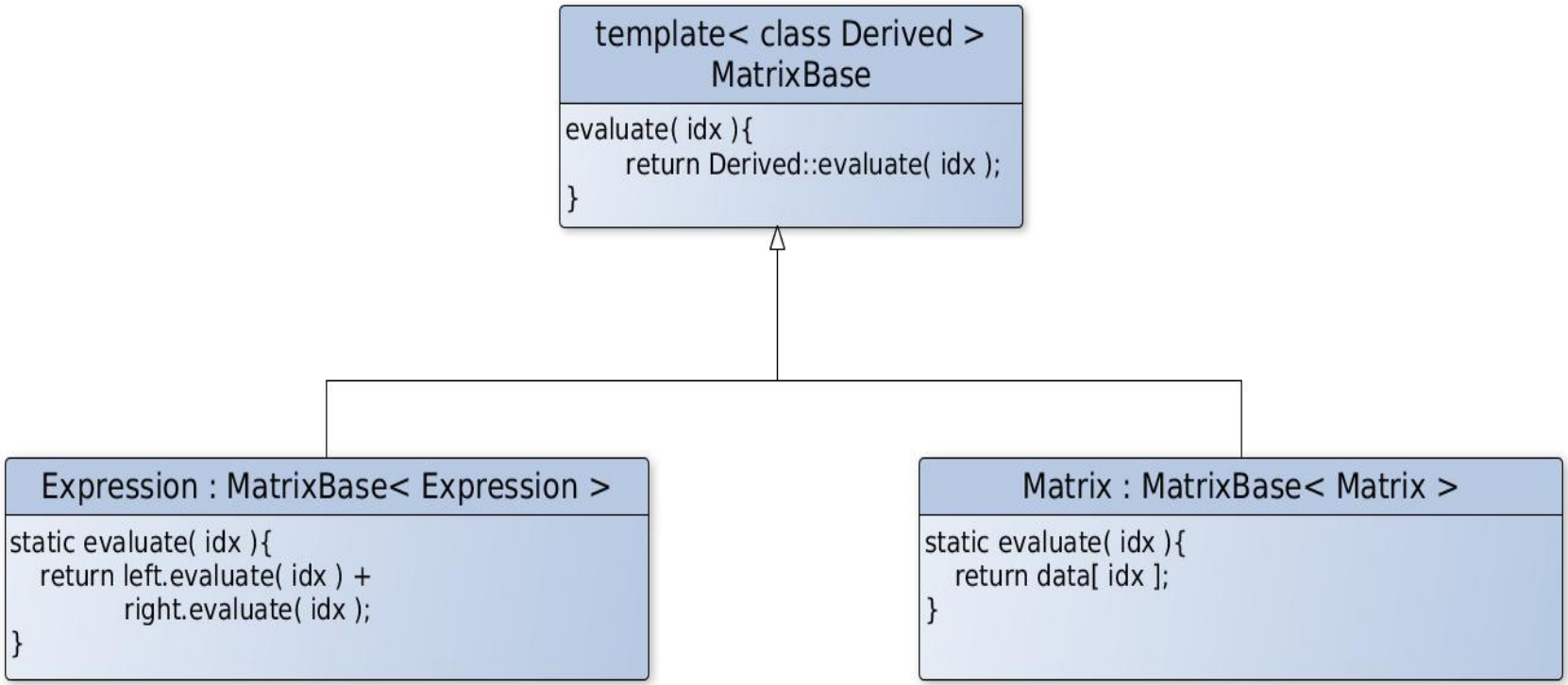


obj: object of MatrixBase:

$$obj.evaluate() = \begin{cases} Expression.evaluate() \\ Matrix.evaluate() \end{cases}$$

$\rightarrow$ decided at run-time

Static polymorphism



$$obj.evaluate() = \begin{cases} Expression.evaluate() \\ Matrix.evaluate() \end{cases}$$

$\rightarrow$ decided at compile-time

Eigen's implementation

After compilation:

```
for i in matrix do:
    A[i] = B[i] + C[i] + D[i]
```

Live showing:

$$A = b \cdot B + c \cdot C + d \cdot D$$

- Eigen vs Intel MKL
- size: 20000 * 20000
- run-time: initialization, computing

Problem

$$A = A * A$$

→Coefficient on top left will be changed before used again
→call .eval()

Conclusion

Eigen is worth using:

- versatile, easy to use, fast and so on

Eigen is worth exploring:

- elegant design

Eigen is worth keeping up; it is in development

- to use AVX
- to parallelize more functions
- ect.

Thanks for your attention