Smart Expression Templates

Marcin Copik

Automation, Compilers, and Code-Generation
Summer semester 2015
RWTH Aachen

January 22, 2017

Outline

1. Motivation
   - Expression Templates
   - Limitations of ETs
   - Extensions
2. Smart Expression Templates
   - What are SETs?
   - How to improve ETs?
3. Evaluation
   - Test Cases
4. Summary

Outline

1 Motivation
- Expression Templates
- Limitations of ETs
- Extensions

2 Smart Expression Templates
- What are SETs?
- How to improve ETs?

3 Evaluation
- Test Cases

4 Summary

Why?

- Provide efficiency of C-style code for linear algebra:

  "...evaluates vector expressions at 95-99.5% efficiency of hand-coded C..."

  "The speed is 2-15 times that of a conventional C++ vector class."

- Avoid creation of temporary data objects:

  "...evaluate vector and matrix expressions in a single pass without temporaries..."

Why?

- Provide efficiency of C-style code for linear algebra:

  "...evaluates vector expressions at 95-99.5% efficiency of hand-coded C..."

  "The speed is 2-15 times that of a conventional C++ vector class."

- Avoid creation of temporary data objects:

  "...evaluate vector and matrix expressions in a single pass without temporaries..."

How?

Expression Templates are based on templatized expression tree.

- The node contains all information needed for evaluation of expression
- No temporaries are created during whole evaluation.
- Lazy evaluation - the expression is computed only when the value is accessed.
- Code is clean, natural and flexible.

Vector sum - tree node

```
template< typename A, typename B >
class Sum {
  const A& a;
  const B& b;
  ...
  Sum(const A& _a, const B& _b) :
          a(_a), b(_b) {}
  double operator[]( std::size_t i ) const {
     return a[i] + b[i];
  }
}
```

Vector sum - node creation

```
template< typename A, typename B >
Sum<A,B> operator+( const A& a, const B& b )
{
  return Sum<A,B>( a, b );
}
```

Outline

1 Motivation
- Expression Templates
- Limitations of ETs
- Extensions

2 Smart Expression Templates
- What are SETs?
- How to improve ETs?

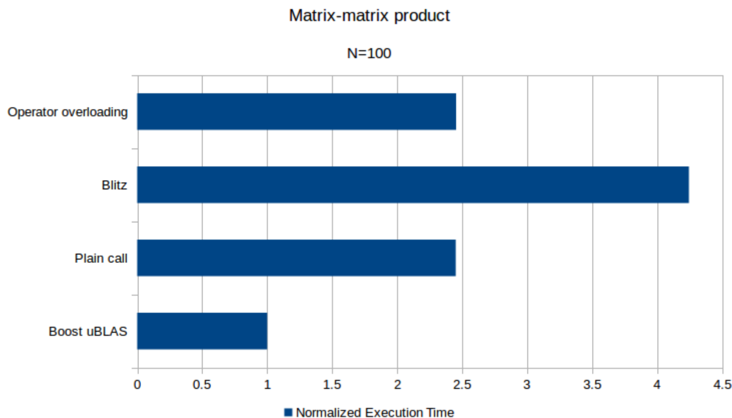
3 Evaluation
- Test Cases

4 Summary

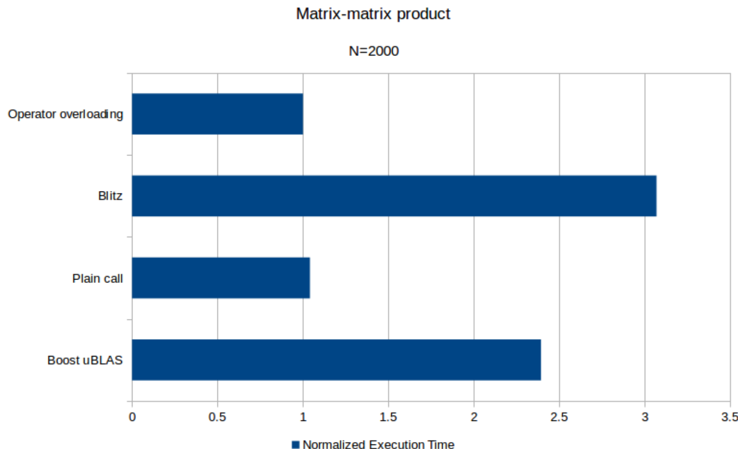
ETs as a performance optimization

- Todd Veldhuizen designed ETs for vector operations.
- Linear algebra heavily uses **matrix** operations.
- How does ETs perform on matrix operations?

GEMM: matrix-matrix product



GEMM: matrix-matrix product



Abstraction as a problem

Lazy evaluation of GEMM

ETs does not make any use of information about the expression - matrix multiplication is evaluated in the same manner as matrix addition:

```
template<typename A>
Matrix & operator=( const A& expr)
{
  ...
  data(i,j) = expr(i,j);
}
```

Abstraction as a problem

ETs implementation

```
for(i = 0; i < A.rows(); ++i) {
  for(j = 0;j < B.cols(); ++j) {
    for(k = 0;k < A.cols();++k) {
      C(i,j) += A(i,k) * B(k,j);
    }
  }
}
```

Abstraction as a problem

Operator overloading implementation

Simple, yet efficient computation of matrix-matrix product:

```
for(i = 0; i < A.rows(); ++i) {
  for(k = 0;k < B.cols(); ++k) {
    C(i,k) = A(i, 0) * B(0, k);
  }
  for(j = 1;j < A.cols(); ++j) {
    for(k = 0;k < B.cols();++k) {
      C(i,k) += A(i,j) * B(j,k);
    }
  }
}
```

Abstraction as a problem

Lazy evaluation of GEMM

ETs does **NOT** allow for proper optimization of matrix-matrix product.

- vectorization
- proper caching
- efficient data access scheme

Do we still need ETs?

C++11 move semantics

In C++11 one can perform operator overloading without additional copy:

```
class Vector {
  Vector(Vector &&);
  Vector operator+(const Vector & a, const
   Vector & b)
  {
    ...
    return c;
  }
}
```

Do we still need ETs?

ETs are not only about avoiding copy

Product of symmetric, dense matrix with sparse vector:

- **row-major matrix**
  non-zero elements of vector multiplied by selected positions at row
  impossible to vectorize
- **column-major matrix**
  whole column of matrix may be multiplied by non-zero element of vector
  efficiently vectorized

Symmetric row-major matrix may be transposed by library - result is still the same!

Do we still need ETs?

ETs are not only about avoiding copy

Product of symmetric, dense matrix with sparse vector:

- **row-major matrix**
  non-zero elements of vector multiplied by selected positions at row
  impossible to vectorize
- **column-major matrix**
  whole column of matrix may be multiplied by non-zero element of vector
  efficiently vectorized

Symmetric row-major matrix may be transposed by library - result is still the same!

Do we still need ETs?

ETs are not only about avoiding copy

Product of symmetric, dense matrix with sparse vector:

- **row-major matrix**
  non-zero elements of vector multiplied by selected positions at row
  impossible to vectorize
- **column-major matrix**
  whole column of matrix may be multiplied by non-zero element of vector
  efficiently vectorized

Symmetric row-major matrix may be transposed by library - result is still the same!

Do we still need ETs?

ETs are not only about avoiding copy

Dense matrices: A, B and dense vector x:

- $A \cdot B \cdot y$
  evaluated from left to right, one matrix-matrix product and one matrix-vector product
- $A \cdot (B \cdot y)$
  two matrix-vector products, **zero** matrix-matrix products

Do we still need ETs?

ETs are not only about avoiding copy

Dense matrices: A, B and dense vector x:

- $A \cdot B \cdot y$
  evaluated from left to right, one matrix-matrix product and one matrix-vector product
- $A \cdot (B \cdot y)$
  two matrix-vector products, **zero** matrix-matrix products

Do we still need ETs?

Domain Specific Languages

Expression templates are **NOT** only a tool for linear algebra operations.
ETs have been succesfully used for generation of **Domain Specific Languages**.
Smart ETs methodology doesn't affect this particular application.

Do we still need ETs?

Domain Specific Languages

- QT ($\geq$ 4.6) uses ETs in QStringBuilder class for more efficient creation of strings.
- Boost.Proto uses ETs for creation of **Embedded DSLs**.
  " In short, Proto is an EDSL for defining EDSLs. "
- Khronos SYCL is a DSEL used for automatic translation of C++11 code to OpenCL kernel.

Outline

1 Motivation
- Expression Templates
- Limitations of ETs
- Extensions

2 Smart Expression Templates
- What are SETs?
- How to improve ETs?

3 Evaluation
- Test Cases

4 Summary

How to improve performance of ETs?

Fast ETs

- $c = a \cdot b \cdot a$
  This arithmetical expression can be computed efficiently, if one knows that two vectors are efficiently the same.
- ```
  template<int num>
  class Vector { ... };
  ```
- Store pointers in special set or as a static field, because every class correspond to different vector.
  ```

How to improve performance of ETs?

Fast ETs

- $c = b + c \cdot d$
  FETs are faster for small vectors, because tree nodes doesn't have to store pointers to objects - they are uniquely determined by their enumerated type.
- $c = a + a \cdot a + a \cdot a \cdot a + \ldots$
  FETs are ten times faster than usual ETs.

How to improve performance of ETs?

ETs over GPGPUs

- **“CUDA Expression Templates“**
  Works only for vector operations, code released as open-source, no longer developed.
- **“Automatic Offloading C++ Expression Templates to CUDA Enabled GPUs“**
  Developed as a part of tool for quantum chromodynamics computations on GPU.
- **“Expression Templates and OpenCL“**
  Source code has never been published.
- **VexCL**
  Vector expression template library for CUDA/OpenCL, actively developed.

How to improve performance of ETs?

ETs over GPGPUs

- **“CUDA Expression Templates“**
  Works only for vector operations, code released as open-source, no longer developed.
- **“Automatic Offloading C++ Expression Templates to CUDA Enabled GPUs“**
  Developed as a part of tool for quantum chromodynamics computations on GPU.
- **“Expression Templates and OpenCL“**
  Source code has never been published.
- **VexCL**
  Vector expression template library for CUDA/OpenCL, actively developed.

How to improve performance of ETs?

ETs over GPGPUs

- **“CUDA Expression Templates“**
  Works only for vector operations, code released as open-source, no longer developed.
- **“Automatic Offloading C++ Expression Templates to CUDA Enabled GPUs“**
  Developed as a part of tool for quantum chromodynamics computations on GPU.
- **“Expression Templates and OpenCL“**
  Source code has never been published.
- **VexCL**
  Vector expression template library for CUDA/OpenCL, actively developed.

How to improve performance of ETs?

ETs over GPGPUs

- **“CUDA Expression Templates“**
  Works only for vector operations, code released as open-source, no longer developed.
- **“Automatic Offloading C++ Expression Templates to CUDA Enabled GPUs“**
  Developed as a part of tool for quantum chromodynamics computations on GPU.
- **“Expression Templates and OpenCL“**
  Source code has never been published.
- **VexCL**
  Vector expression template library for CUDA/OpenCL, actively developed.

Outline

1 Motivation
- Expression Templates
- Limitations of ETs
- Extensions

2 Smart Expression Templates
- What are SETs?
- How to improve ETs?

3 Evaluation
- Test Cases

4 Summary

Introduce 'smarter' ETs

SIAM J. Sci. Comput.
Vol. 34, No. 2, pp. C42–C69

© 2012 Society for Industrial and Applied Mathematics

EXPRESSION TEMPLATES REVISITED: A PERFORMANCE ANALYSIS OF CURRENT METHODOLOGIES*

KLAUS IGLBERGER[†], GEORG HAGER[‡], JAN TREIBIG[‡], AND ULRICH RÜDE[§]

Abstract. In the last decade, expression templates (ETs) have gained a reputation as an efficient performance optimization tool for C++ codes. This reputation builds on several ET-based linear algebra frameworks focused on combining both elegant and high-performance C++ code. However, on closer examination the assumption that ETs are a performance optimization technique cannot be maintained. In this paper we compare the performance of several generations of ET-based frameworks. We analyze different ET methodologies and explain the inability of some ET implementations to deliver high performance for dense and sparse linear algebra operations. Additionally, we introduce the notion of "smart" ETs, which truly allow for a combination of high performance code with the elegance and maintainability of a domain-specific language.

Key words. expression templates, performance optimization, high performance programming, linear algebra, Boost, uBLAS, Blitz++, MTL4, Eigen3, Blaze

AMS subject classification. 68

DOI. 10.1137/110830125

1. Introduction. Expression templates (ETs) as originally introduced by Veldhuizen in 1995 [22, 23] are intended to be a "performance optimization for array-based operations." The general goal is to avoid the unnecessary creation of temporary objects during the evaluation of arithmetic expressions with overloaded operators in C++. Commonly demonstrated using simple $\mathcal{O}(n)$ array operations like additions

Introduce 'smarter' ETs

- analyze performance of ETs
- find and study bottlenecks
- propose new methodology
- implement SETs in new library - Blaze

Introduce 'smarter' ETs

- analyze performance of ETs
- find and study bottlenecks
- propose new methodology
- implement SETs in new library - Blaze

Introduce 'smarter' ETs

- analyze performance of ETs
- find and study bottlenecks
- propose new methodology
- implement SETs in new library - Blaze

Introduce 'smarter' ETs

- analyze performance of ETs
- find and study bottlenecks
- propose new methodology
- implement SETs in new library - Blaze

Outline

1 Motivation
- Expression Templates
- Limitations of ETs
- Extensions

2 Smart Expression Templates
- What are SETs?
- How to improve ETs?

3 Evaluation
- Test Cases

4 Summary

Remove abstraction from expressions!

Extended expression node

- specialized type, e.g. DMatDMatMultExpr, instead of abstract binary operation node
- result type of both expression sides
- composite type of both expression sides

Integrated Compute Kernels

- The classical linear algebra algorithms are **rarely** the best solution.
- There exist many highly-optimized solutions for linear algebra: BLAS, LAPACK, cuBLAS etc.

Kernels support in Blaze

- BLAS
- Shared Memory Parallelism:
  - Boost Threads
  - C++11 Threads
  - OpenMP

Integrated Compute Kernels

- The classical linear algebra algorithms are **rarely** the best solution.
- There exist many highly-optimized solutions for linear algebra: BLAS, LAPACK, cuBLAS etc.

Kernels support in Blaze

- BLAS
- Shared Memory Parallelism:
  - Boost Threads
  - C++11 Threads
  - OpenMP

Integrated Compute Kernels

- The classical linear algebra algorithms are **rarely** the best solution.
- There exist many highly-optimized solutions for linear algebra: BLAS, LAPACK, cuBLAS etc.

Kernels support in Blaze

- BLAS
- Shared Memory Parallelism:
  - Boost Threads
  - C++11 Threads
  - OpenMP

Integrated Compute Kernels

One can't use BLAS directly in expression templates! The subexpressions have to be evaluated before and stored in a temporary object.
The policy **"absolutely no temporaries"** needs to be reconsidered.

Intermediate temporaries

- What if the expression is nested?
  $a = A \cdot (a + b + c)$
- In ETs vector sum can't be stored in a temporary object
- Each vector element needs to be used several times in multiplication.
- Result: more arithmetical expressions and a lot of cache line replacements

Intermediate temporaries

- What if the expression is nested?
  $a = A \cdot (a + b + c)$
- In ETs vector sum can't be stored in a temporary object
- Each vector element needs to be used several times in multiplication.
- Result: more arithmetical expressions and a lot of cache line replacements

Intermediate temporaries

- What if the expression is nested?
  $a = A \cdot (a + b + c)$
- In ETs vector sum can't be stored in a temporary object
- Each vector element needs to be used several times in multiplication.
- Result: more arithmetical expressions and a lot of cache line replacements

Intermediate temporaries

- What if the expression is nested?
  $E = (A + B) \cdot (C - D)$
- No possibility of using efficient multiplication algorithm with temporaries.
- Result: more arithmetical expressions and much more cache line replacements (an increase of one order of magnitude, comparing to operator overloading).

Intermediate temporaries

- What if the expression is nested?
  $E = (A + B) \cdot (C - D)$
- No possibility of using efficient multiplication algorithm with temporaries.
- Result: more arithmetical expressions and much more cache line replacements (an increase of one order of magnitude, comparing to operator overloading).

Intermediate temporaries

- What if the expression is nested?
  $E = (A + B) \cdot (C - D)$
- No possibility of using efficient multiplication algorithm with temporaries.
- Result: more arithmetical expressions and much more cache line replacements (an increase of one order of magnitude, comparing to operator overloading).

Intermediate temporaries

- Avoid temporaries is the main goal of ETs.
- MTL4 and Eigen introduced intermediate temporaries to avoid these performance problems.

Intermediate temporaries

- Avoid temporaries is the main goal of ETs.
- MTL4 and Eigen introduced intermediate temporaries to avoid these performance problems.

Rearrange of operations

- Consider already introduced matrix-matrix-vector product: $A \cdot B \cdot x$
- Good library should be able to optimize these kind of expression.
- Consider another expression: $A = B + C * D$
- Naive introduction of temporary: $TMP = C * D$ $A = B + TMP$
- Smarter reordering: $A = B$ $A += C * D$

Rearrange of operations

- Consider already introduced matrix-matrix-vector product: $A \cdot B \cdot x$
- Good library should be able to optimize these kind of expression.
- Consider another expression:
  $A = B + C * D$
- Naive introduction of temporary:
  $TMP = C * D$
  $A = B + TMP$
- Smarter reordering:
  $A = B$
  $A += C * D$

Outline

1 Motivation
- Expression Templates
- Limitations of ETs
- Extensions

2 Smart Expression Templates
- What are SETs?
- How to improve ETs?

3 Evaluation
- Test Cases

4 Summary

Test Cases

- Evaluate different implementations:
  - C-like plain call
  - ETs - Boost uBLAS 1.58, Blitz++ 0.10
  - SETs - Blaze 2.3
  - BLAS - Intel MKL 11.2
- One thread, 10 repetitions, time measured with C++ std::chrono::high_resolution_clock
- Intel compiler with -O3, -mtune=core-avx2 and -inline-forceinline
  Intel Xeon E3-1231 3.4 GHz

Test Cases

- Single matrix-matrix product
- Complex expression: $(A + B) \cdot (D - E)$
- Dense matrix multiplication with a sparse matrix

Test Case 1

General matrix-matrix product

```
blitz::Array<double, 2> C(m,l), B(k,l), A(m,
   k);
blitz::firstIndex i;
blitz::secondIndex j;
blitz::thirdIndex n;
C = blitz::sum(A(i,n) * B(n,j), n);
```

Test Case 1

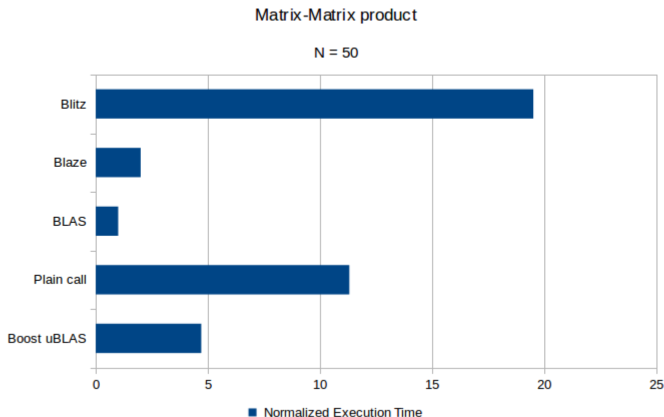
General matrix-matrix product

```
boost::numeric::ublas::matrix<double> A(m,k)
    , B(k,l), C(m,l);
noalias(C) = prod( A, B );
```

```
blaze::DynamicMatrix<double,blaze::rowMajor>
    A(m, k), B(k, l), C(m, l);
C = A * B;
```

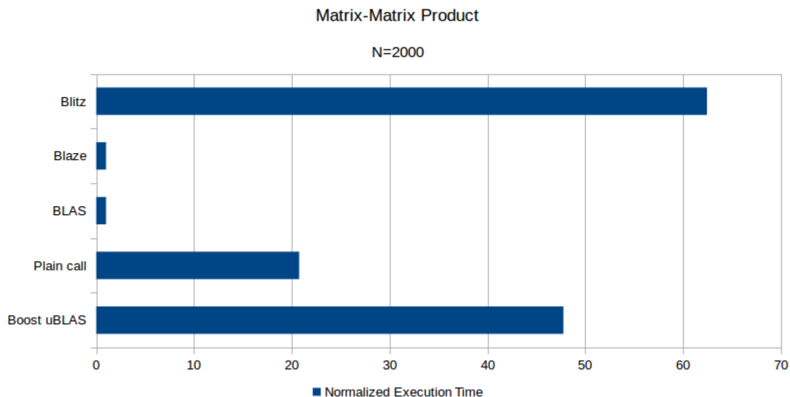
Test Case 1

General matrix-matrix product



Test Case 1

General matrix-matrix product



Test Case 2

Nested matrix operations

```
blitz::Array<double, 2> A(m, m), B(m, m),
C(m, m), D(m, m), E(m,m);
blitz::firstIndex i;
blitz::secondIndex j;
blitz::thirdIndex n;
A += B;
C -= D;
E = blitz::sum(A(i,n) * C(n,j), n);
```

Test Case 2

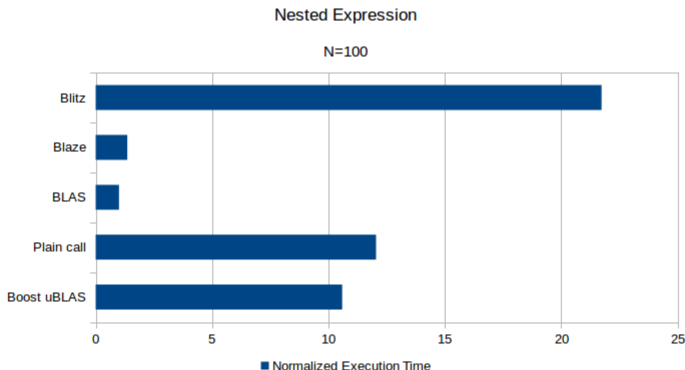
Nested matrix operations

```
boost::numeric::ublas::matrix<double> A(m,m)
    , B(m, m), C(m, m), D(m, m), E(m,m);
noalias(E) = prod( A + B, C - D );
```

```
blaze::DynamicMatrix<double,blaze::rowMajor>
      A(m, m), B(m, m), C(m, m), D(m, m),
 E(m,m);
E = (A - B) * (C - D);
```

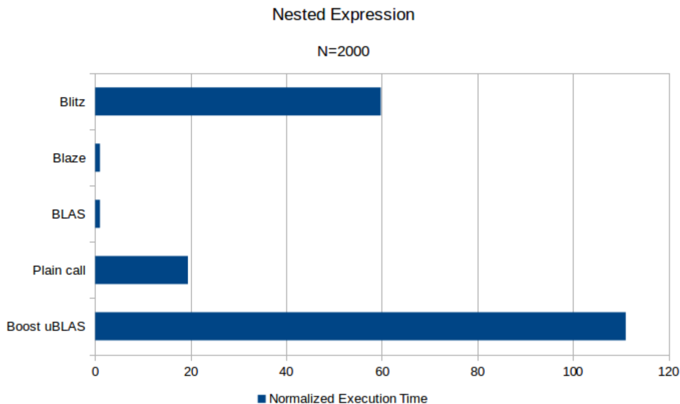
Test Case 2

Nested matrix operations



Test Case 2

Nested matrix operations



Test Case 3

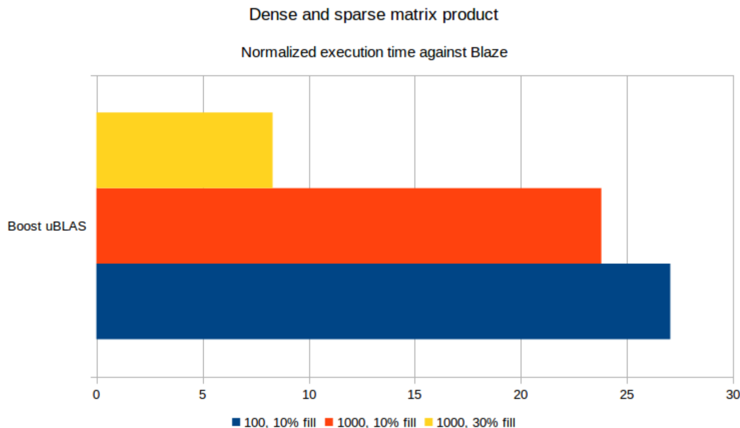
Sparse matrices

```
boost::numeric::ublas::matrix<double> A(m,m)
    , C(m, m);
boost::numeric::ublas::compressed_matrix<
    double> B(m, m);
noalias(C) = prod( A, B );

blaze::DynamicMatrix<double,blaze::rowMajor>
     A(m, m), C(m, m);
blaze::CompressedMatrix<double> B(m, m);
C = A * B;
```

Test Case 3

Sparse matrices



Summary

- General Expression Templates are **not** suitable for linear algebra libraries in C++
- Introduction of temporaries improves the performance of nested expressions.
- Don't reinvent the wheel - use tested and well-performing libraries for numerical computations.
- Expression Templates are a valuable tool for scientific computing in C++, which allow for many important optimizations.
- Remember to set all flags which are necessary to enable building in release mode - pure *NDEBUG* may not be enough!

Summary

- General Expression Templates are **not** suitable for linear algebra libraries in C++
- Introduction of temporaries improves the performance of nested expressions.
- Don't reinvent the wheel - use tested and well-performing libraries for numerical computations.
- Expression Templates are a valuable tool for scientific computing in C++, which allow for many important optimizations.
- Remember to set all flags which are necessary to enable building in release mode - pure *NDEBUG* may not be enough!

Summary

- General Expression Templates are **not** suitable for linear algebra libraries in C++
- Introduction of temporaries improves the performance of nested expressions.
- Don't reinvent the wheel - use tested and well-performing libraries for numerical computations.
- Expression Templates are a valuable tool for scientific computing in C++, which allow for many important optimizations.
- Remember to set all flags which are necessary to enable building in release mode - pure *NDEBUG* may not be enough!

Summary

- General Expression Templates are **not** suitable for linear algebra libraries in C++
- Introduction of temporaries improves the performance of nested expressions.
- Don't reinvent the wheel - use tested and well-performing libraries for numerical computations.
- Expression Templates are a valuable tool for scientific computing in C++, which allow for many important optimizations.
- Remember to set all flags which are necessary to enable building in release mode - pure *NDEBUG* may not be enough!

Summary

- General Expression Templates are **not** suitable for linear algebra libraries in C++
- Introduction of temporaries improves the performance of nested expressions.
- Don't reinvent the wheel - use tested and well-performing libraries for numerical computations.
- Expression Templates are a valuable tool for scientific computing in C++, which allow for many important optimizations.
- Remember to set all flags which are necessary to enable building in release mode - pure *NDEBUG* may not be enough!

References I

Benchmark used to generate results.
https://github.com/mcopik/SmartETBenchmark

"Expression Templates Revisited"
Klaus Iglberger @ Meeting C++ 2014
https://www.youtube.com/watch?v=hfn0BVOegac

References

T. Veldhuizen.
Expression Templates.
C++ Report, 7 (1995), pp. 26–31.

K. Iglberger, G. Hager, J. Treibig, U. Ruede
Expression Templates Revisited: A Performance Analysis of the Current ET Methodology.
SIAM Journal on Scientific Computing, Volume 34, Issue 2 (2011).