CD-adapco
A Siemens Business

... because performance matters

Klaus Iglberger – CppCon 2016 – September 21st

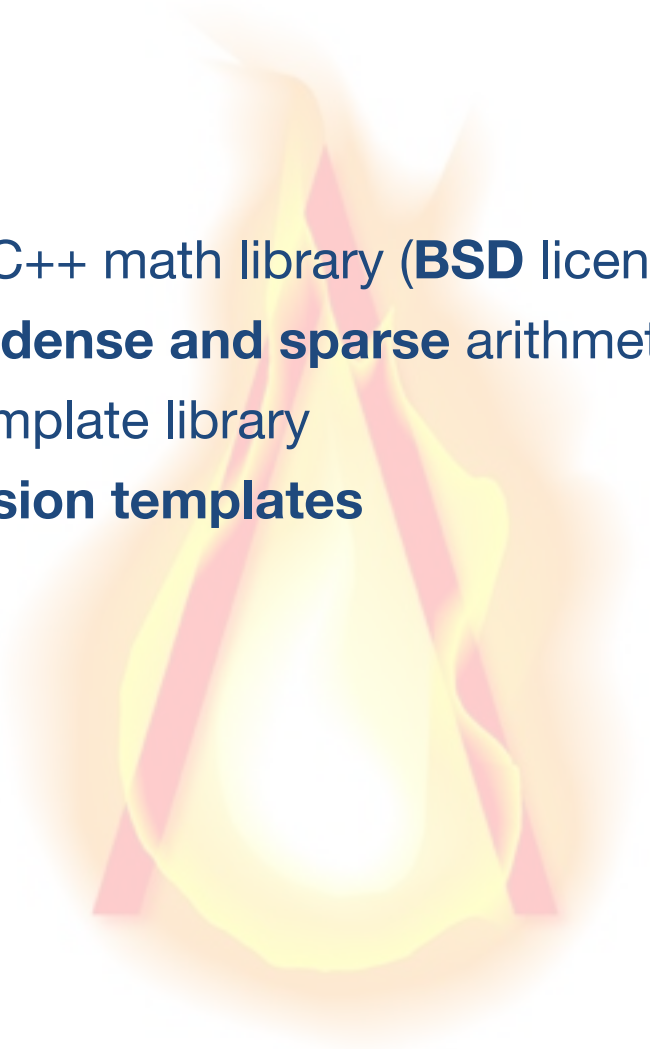
CD-adapco
A Siemens Business

```
using namespace blaze;

const size_t NN( N*N );

CompressedMatrix<double,rowMajor> A(NN,NN);
DynamicVector<double,columnVector> x(NN,1.0), b(NN,0.0), r(NN), p(NN), Ap(NN);
double alpha, beta, delta;

// ... Initializing the sparse matrix A

// Performing the CG algorithm
r = b - A * x;
p = r;
delta = (r,r);

for( size_t iteration=0UL; iteration<iterations; ++iteration )
{
   Ap = A * p;
   alpha = delta / (p,Ap);
   x += alpha * p;
   r -= alpha * Ap;
   beta = (r,r);
   if( std::sqrt( beta ) < 1E-8 ) break;
   p = r + ( beta / delta ) * p;
   delta = beta;
}

...
```

CD-adapco
A Siemens Business

Blaze is …

- … an **open source** C++ math library (**BSD** license)
- … a math library for **dense and sparse** arithmetic
- … a **header-only** template library
- … based on **expression templates**



CD-adapco
A Siemens Business

Blaze offers …

- … **high performance** through the integration of BLAS libraries and manually tuned HPC math kernels
- … **vectorization** by SSE, AVX, AVX-512, FMA, and SVML
- … **parallel execution** by OpenMP, C++11 threads and Boost threads
- … the **intuitive** and **easy to use** API of a domain specific language
- … **unified arithmetic** with dense and sparse vectors and matrices
- … **thoroughly tested** matrix and vector arithmetic
- … completely **portable, high quality** C++ source code

- Version 1.0 has been released in August 2012
- The current version 3.0 has been released in August 2016

CD-adapco
A Siemens Business

Come on, another C++ math library?

CD-adapco
A Siemens Business

Benchmarks

CD-adapco
A Siemens Business

Intel “Haswell” 10-core Xeon with 2.3 GHz

(turbo boost disabled, peak performance: 36.8 Gflops, Bandwidth STREAM: 55.6 GByte/s)

GNU compiler 6.1 / Clang 3.8

```
g++ -O3 -mavx -mfma -DNDEBUG -std=c++14 …
```

CD-adapco
A Siemens Business

```
::blaze::DynamicVector<double,columnVector> a( N ), b( N ), c( N );
::blaze::timing::WcTimer timer;

init( a );
init( b );

c = a + b;

for( size_t rep=0UL; rep<reps; ++rep )
{
   timer.start();
   for( size_t step=0UL; step<steps; ++step ) {
      c = a + b;
   }
   timer.end();
}
```

All benchmarks are run for at least 2 seconds

Samples are collected in each of several runs

All in-cache problems are hot

CD-adapco
A Siemens Business

Blaze 3.0

(August 2016)

Eigen 3.3-beta2

(July 2016, including support for AVX and FMA)

Armadillo 7.300.1

(June 2016)

Boost uBlas 1.61

(November 2014)

CD-adapco
A Siemens Business

Blitz++ 0.10
(March 2014)

GMM++ 5.0
(July 2015)

MTL 4.0.9555
(May 2014)

Intel MKL 14.0.1
(October 2013)

CD-adapco
A Siemens Business

BLAS Level 1

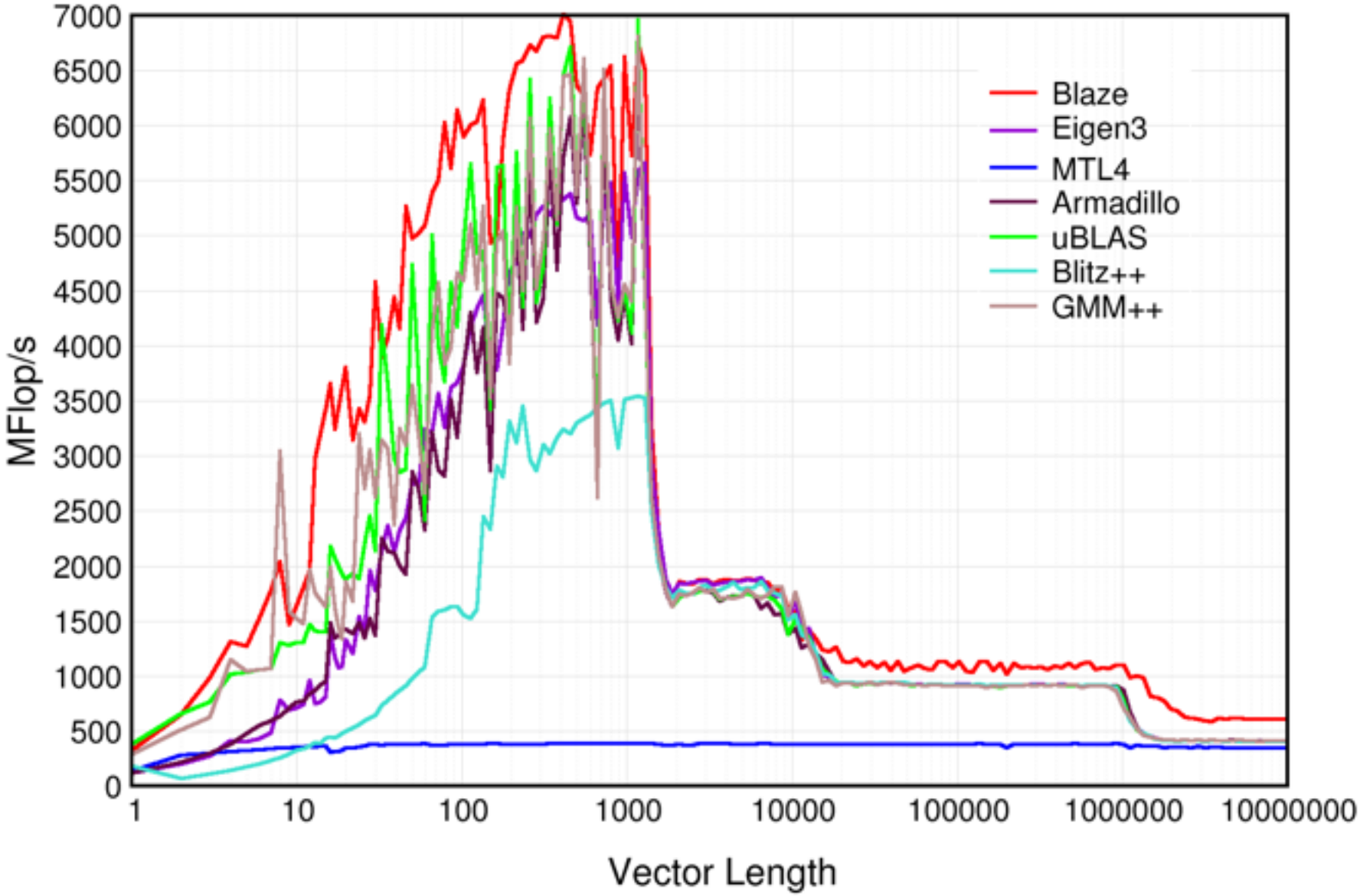
CD-adapco
A Siemens Business

```
template< typename Type, size_t N, bool TF >
class StaticVector;


template< typename Type, bool TF >
class DynamicVector;


template< typename Type, size_t N, bool TF >
class HybridVector;
```

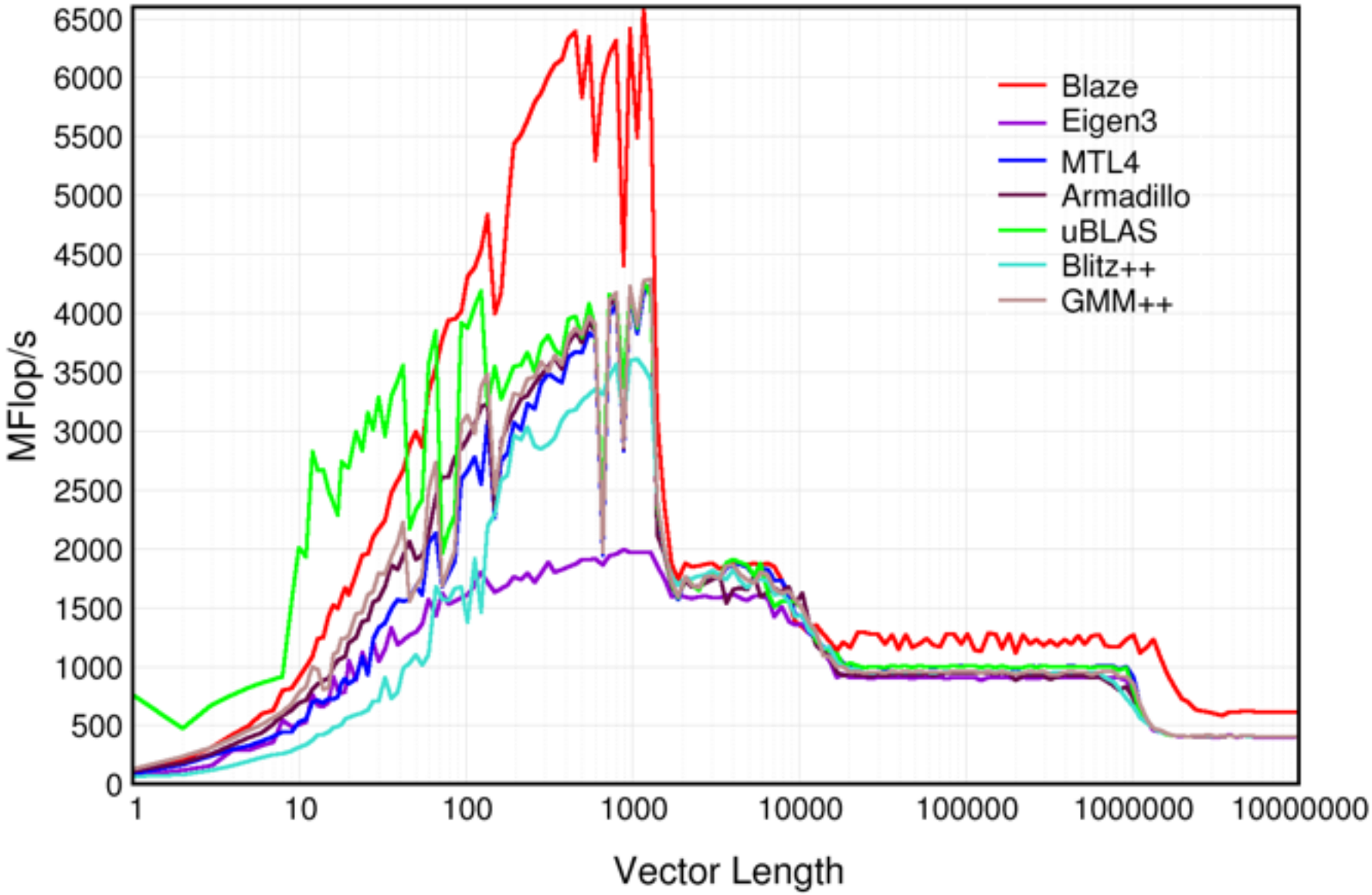
CD-adapco
A Siemens Business

```
using namespace blaze;

DynamicVector<double,columnVector> a(N), b(N), c(N);

// ... Initialization of the vectors

c = a + b;
```

Clang 3.8


MFlop/s
7000
6500
6000
5500
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1
10
100
1000
10000
100000
1000000
10000000
Vector Length
Blaze
Eigen3
MTL4
Armadillo
uBLAS
Blitz++
GMM++

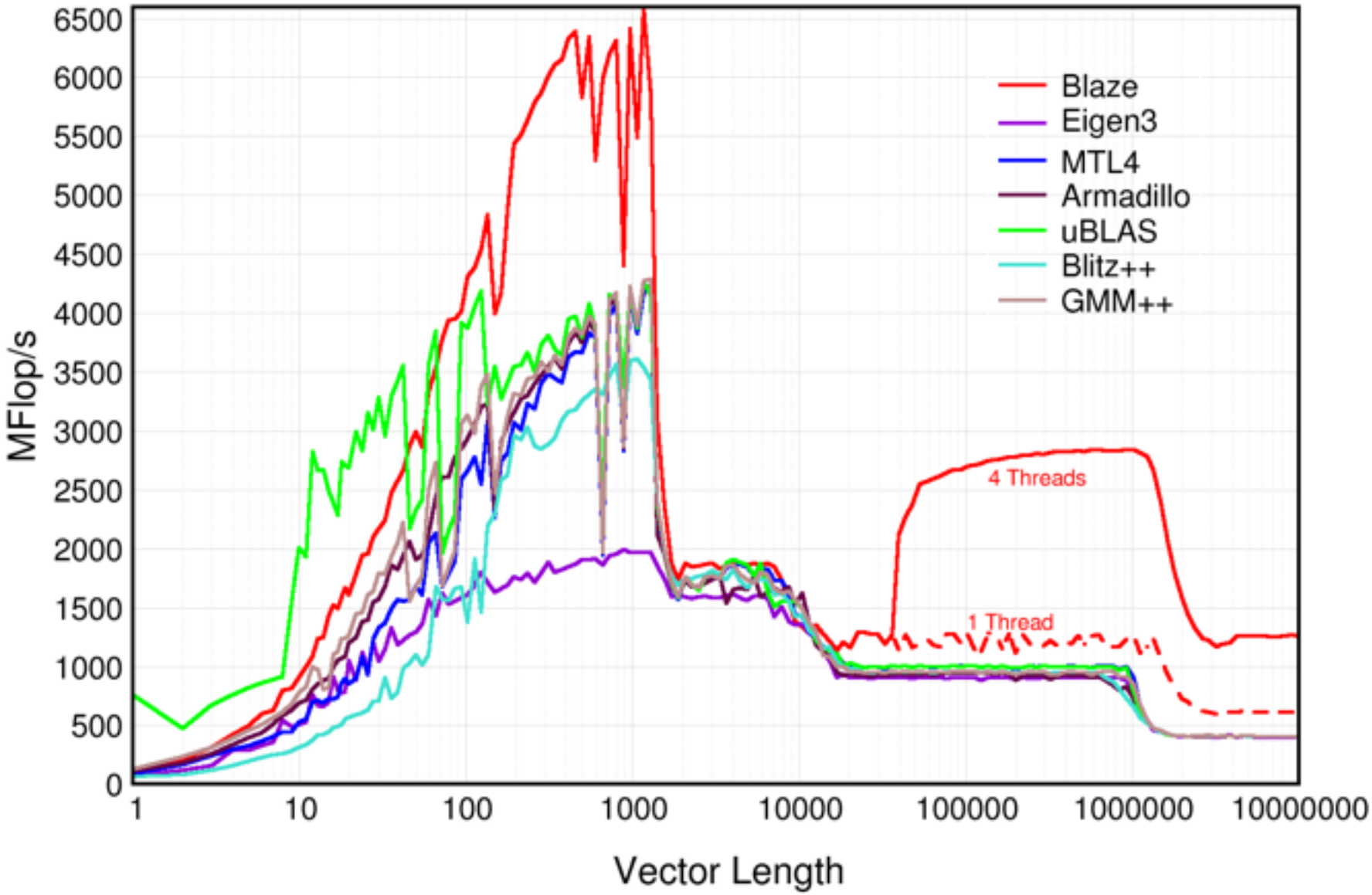
GNU compiler 6.1



CD-adapco
A Siemens Business

```
g++ -O3 -mavx -mfma -fopenmp -DNDEBUG …

    export OMP_NUM_THREADS=4
    export OMP_PLACES="{0,1,2,3}"
```

GNU compiler 6.1 with OpenMP



```
template< typename Type  // Data type of the vector
        , bool TF >      // Transpose flag
template< typename VT >  // Type of the right-hand side dense vector
inline EnableIf_<typename DynamicVector<Type,TF>::BLAZE_TEMPLATE VectorizedAssign<VT> >
   DynamicVector<Type,TF>::assign( const DenseVector<VT,TF>& rhs )
{
   BLAZE_CONSTRAINT_MUST_BE_VECTORIZABLE_TYPE( Type );

   BLAZE_INTERNAL_ASSERT( size_ == (~rhs).size(), "Invalid vector sizes" );

   const bool remainder( !usePadding || !IsPadded<VT>::value );

   const size_t ipos( ( remainder )?( size_ & size_t(-SIMDSIZE) ):( size_ ) );
   BLAZE_INTERNAL_ASSERT( !remainder || ( size_ - ( size_ % (SIMDSIZE) ) ) == ipos, "..." );

   size_t i=0UL;
   Iterator left( begin() );
   ConstIterator_<VT> right( (~rhs).begin() );

   if( useStreaming && size_ > ( cacheSize/( sizeof(Type) * 3UL ) ) && !(~rhs).isAliased( this ) )
   {
      for( ; i<ipos; i+=SIMDSIZE ) {
         left.stream( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; remainder && i<size_; ++i ) {
         *left = *right; ++left; ++right;
      }
   }
   else
   {
      for( ; (i+SIMDSIZE*3UL) < ipos; i+=SIMDSIZE*4UL ) {
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; i<ipos; i+=SIMDSIZE ) {
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; remainder && i<size_; ++i ) {
         *left = *right; ++left; ++right;
      }
   }
}
```

```
template< typename Type  // Data type of the vector
        , bool TF >      // Transpose flag
template< typename VT >  // Type of the right-hand side dense vector
inline EnableIf_<typename DynamicVector<Type,TF>::BLAZE_TEMPLATE VectorizedAssign<VT> >
   DynamicVector<Type,TF>::assign( const DenseVector<VT,TF>& rhs )
{
   BLAZE_CONSTRAINT_MUST_BE_VECTORIZABLE_TYPE( Type );

   BLAZE_INTERNAL_ASSERT( size_ == (~rhs).size(), "Invalid vector sizes" );

   const bool remainder( !usePadding || !IsPadded<VT>::value );

   const size_t ipos( ( remainder )?( size_ & size_t(-SIMDSIZE) ):( size_ ) );
   BLAZE_INTERNAL_ASSERT( !remainder || ( size_ - ( size_ % (SIMDSIZE) ) ) == ipos, "..." );

   size_t i=0UL;
   Iterator left( begin() );
   ConstIterator_<VT> right( (~rhs).begin() );

   if( useStreaming && size_ > ( cacheSize/( sizeof(Type) * 3UL ) ) && !(~rhs).isAliased( this ) )
   {
      for( ; i<ipos; i+=SIMDSIZE ) {
         left.stream( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; remainder && i<size_; ++i ) {
         *left = *right; ++left; ++right;
      }
   }
   else
   {
      for( ; (i+SIMDSIZE*3UL) < ipos; i+=SIMDSIZE*4UL ) {
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; i<ipos; i+=SIMDSIZE ) {
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; remainder && i<size_; ++i ) {
         *left = *right; ++left; ++right;
      }
   }
}
```

```
template< typename Type  // Data type of the vector
        , bool TF >      // Transpose flag
template< typename VT >  // Type of the right-hand side dense vector
inline EnableIf_<typename DynamicVector<Type,TF>::BLAZE_TEMPLATE VectorizedAssign<VT> >
   DynamicVector<Type,TF>::assign( const DenseVector<VT,TF>& rhs )
{
   BLAZE_CONSTRAINT_MUST_BE_VECTORIZABLE_TYPE( Type );

   BLAZE_INTERNAL_ASSERT( size_ == (~rhs).size(), "Invalid vector sizes" );

   const bool remainder( !usePadding || !IsPadded<VT>::value );

   const size_t ipos( ( remainder )?( size_ & size_t(-SIMDSIZE) ):( size_ ) );
   BLAZE_INTERNAL_ASSERT( !remainder || ( size_ - ( size_ % (SIMDSIZE) ) ) == ipos, "..." );

   size_t i=0UL;
   Iterator left( begin() );
   ConstIterator_<VT> right( (~rhs).begin() );

   if( useStreaming && size_ > ( cacheSize/( sizeof(Type) * 3UL ) ) && !(~rhs).isAliased( this ) )
   {
      for( ; i<ipos; i+=SIMDSIZE ) {
         left.stream( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; remainder && i<size_; ++i ) {
         *left = *right; ++left; ++right;
      }
   }
   else
   {
      for( ; (i+SIMDSIZE*3UL) < ipos; i+=SIMDSIZE*4UL ) {
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; i<ipos; i+=SIMDSIZE ) {
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; remainder && i<size_; ++i ) {
         *left = *right; ++left; ++right;
      }
   }
}
```

```
template< typename Type  // Data type of the vector
        , bool TF >      // Transpose flag
template< typename VT >  // Type of the right-hand side dense vector
inline EnableIf_<typename DynamicVector<Type,TF>::BLAZE_TEMPLATE VectorizedAssign<VT> >
   DynamicVector<Type,TF>::assign( const DenseVector<VT,TF>& rhs )
{
   BLAZE_CONSTRAINT_MUST_BE_VECTORIZABLE_TYPE( Type );

   BLAZE_INTERNAL_ASSERT( size_ == (~rhs).size(), "Invalid vector sizes" );

   const bool remainder( !usePadding || !IsPadded<VT>::value );

   const size_t ipos( ( remainder )?( size_ & size_t(-SIMDSIZE) ):( size_ ) );
   BLAZE_INTERNAL_ASSERT( !remainder || ( size_ - ( size_ % (SIMDSIZE) ) ) == ipos, "..." );

   size_t i=0UL;
   Iterator left( begin() );
   ConstIterator_<VT> right( (~rhs).begin() );

   if( useStreaming && size_ > ( cacheSize/( sizeof(Type) * 3UL ) ) && !(~rhs).isAliased( this ) )
   {
      for( ; i<ipos; i+=SIMDSIZE ) {
         left.stream( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; remainder && i<size_; ++i ) {
         *left = *right; ++left; ++right;
      }
   }
   else
   {
      for( ; (i+SIMDSIZE*3UL) < ipos; i+=SIMDSIZE*4UL ) {
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; i<ipos; i+=SIMDSIZE ) {
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; remainder && i<size_; ++i ) {
         *left = *right; ++left; ++right;
      }
   }
}
```

```
template< typename Type  // Data type of the vector
        , bool TF >      // Transpose flag
template< typename VT >  // Type of the right-hand side dense vector
inline EnableIf_<typename DynamicVector<Type,TF>::BLAZE_TEMPLATE VectorizedAssign<VT> >
   DynamicVector<Type,TF>::assign( const DenseVector<VT,TF>& rhs )
{
   BLAZE_CONSTRAINT_MUST_BE_VECTORIZABLE_TYPE( Type );

   BLAZE_INTERNAL_ASSERT( size_ == (~rhs).size(), "Invalid vector sizes" );

   const bool remainder( !usePadding || !IsPadded<VT>::value );

   const size_t ipos( ( remainder )?( size_ & size_t(-SIMDSIZE) ):( size_ ) );
   BLAZE_INTERNAL_ASSERT( !remainder || ( size_ - ( size_ % (SIMDSIZE) ) ) == ipos, "..." );

   size_t i=0UL;
   Iterator left( begin() );
   ConstIterator_<VT> right( (~rhs).begin() );

   if( useStreaming && size_ > ( cacheSize/( sizeof(Type) * 3UL ) ) && !(~rhs).isAliased( this ) )
   {
      for( ; i<ipos; i+=SIMDSIZE ) {
         left.stream( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; remainder && i<size_; ++i ) {
         *left = *right; ++left; ++right;
      }
   }
   else
   {
      for( ; (i+SIMDSIZE*3UL) < ipos; i+=SIMDSIZE*4UL ) {
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; i<ipos; i+=SIMDSIZE ) {
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; remainder && i<size_; ++i ) {
         *left = *right; ++left; ++right;
      }
   }
}
```

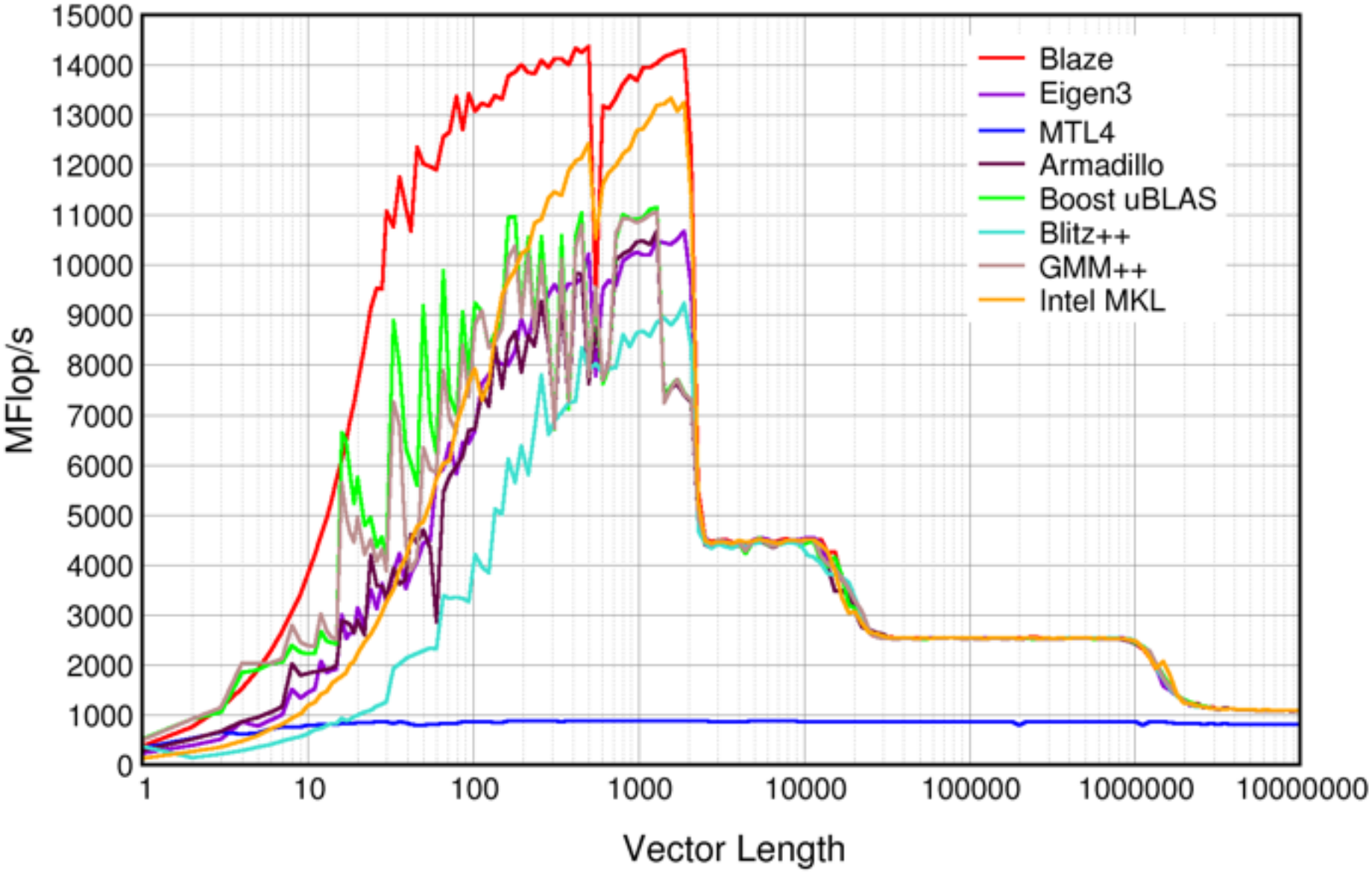
```
template< typename Type  // Data type of the vector
        , bool TF >      // Transpose flag
template< typename VT >  // Type of the right-hand side dense vector
inline EnableIf_<typename DynamicVector<Type,TF>::BLAZE_TEMPLATE VectorizedAssign<VT> >
   DynamicVector<Type,TF>::assign( const DenseVector<VT,TF>& rhs )
{
   BLAZE_CONSTRAINT_MUST_BE_VECTORIZABLE_TYPE( Type );

   BLAZE_INTERNAL_ASSERT( size_ == (~rhs).size(), "Invalid vector sizes" );

   const bool remainder( !usePadding || !IsPadded<VT>::value );

   const size_t ipos( ( remainder )?( size_ & size_t(-SIMDSIZE) ):( size_ ) );
   BLAZE_INTERNAL_ASSERT( !remainder || ( size_ - ( size_ % (SIMDSIZE) ) ) == ipos, "..." );

   size_t i=0UL;
   Iterator left( begin() );
   ConstIterator_<VT> right( (~rhs).begin() );

   if( useStreaming && size_ > ( cacheSize/( sizeof(Type) * 3UL ) ) && !(~rhs).isAliased( this ) )
   {
      for( ; i<ipos; i+=SIMDSIZE ) {
         left.stream( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; remainder && i<size_; ++i ) {
         *left = *right; ++left; ++right;
      }
   }
   else
   {
      for( ; (i+SIMDSIZE*3UL) < ipos; i+=SIMDSIZE*4UL ) {
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; i<ipos; i+=SIMDSIZE ) {
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; remainder && i<size_; ++i ) {
         *left = *right; ++left; ++right;
      }
   }
}
```

```
template< typename Type  // Data type of the vector
        , bool TF >      // Transpose flag
template< typename VT >  // Type of the right-hand side dense vector
inline EnableIf_<typename DynamicVector<Type,TF>::BLAZE_TEMPLATE VectorizedAssign<VT> >
   DynamicVector<Type,TF>::assign( const DenseVector<VT,TF>& rhs )
{
   BLAZE_CONSTRAINT_MUST_BE_VECTORIZABLE_TYPE( Type );

   BLAZE_INTERNAL_ASSERT( size_ == (~rhs).size(), "Invalid vector sizes" );

   const bool remainder( !usePadding || !IsPadded<VT>::value );

   const size_t ipos( ( remainder )?( size_ & size_t(-SIMDSIZE) ):( size_ ) );
   BLAZE_INTERNAL_ASSERT( !remainder || ( size_ - ( size_ % (SIMDSIZE) ) ) == ipos, "..." );

   size_t i=0UL;
   Iterator left( begin() );
   ConstIterator_<VT> right( (~rhs).begin() );

   if( useStreaming && size_ > ( cacheSize/( sizeof(Type) * 3UL ) ) && !(~rhs).isAliased( this ) )
   {
      for( ; i<ipos; i+=SIMDSIZE ) {
         left.stream( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; remainder && i<size_; ++i ) {
         *left = *right; ++left; ++right;
      }
   }
   else
   {
      for( ; (i+SIMDSIZE*3UL) < ipos; i+=SIMDSIZE*4UL ) {
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; i<ipos; i+=SIMDSIZE ) {
         left.store( right.load() ); left += SIMDSIZE; right += SIMDSIZE;
      }
      for( ; remainder && i<size_; ++i ) {
         *left = *right; ++left; ++right;
      }
   }
}
```

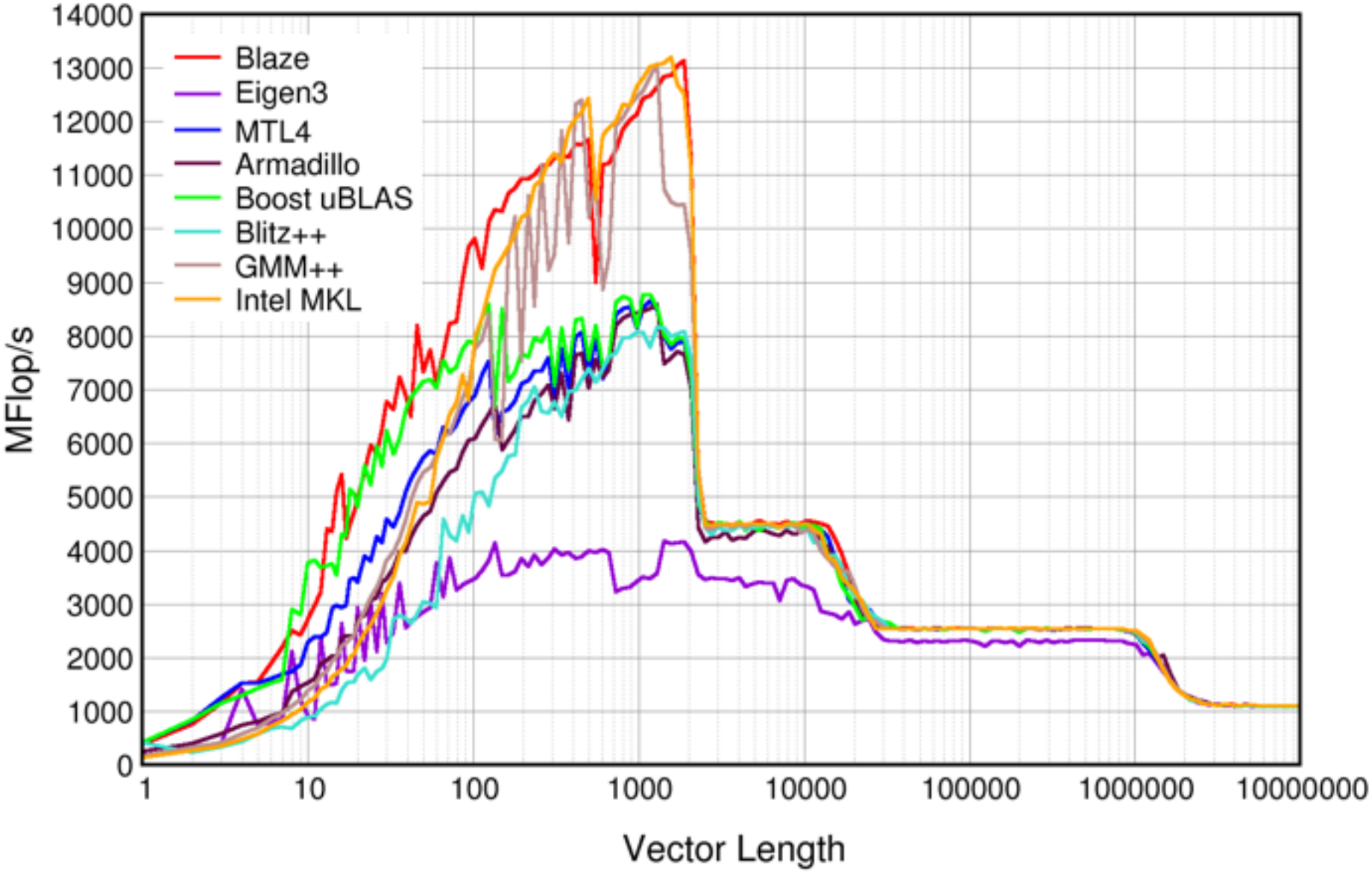
CD-adapco
A Siemens Business

```
using namespace blaze;

DynamicVector<double,columnVector> a(N), b(N);

// ... Initialization of the vectors

b += a * 3.0;
```

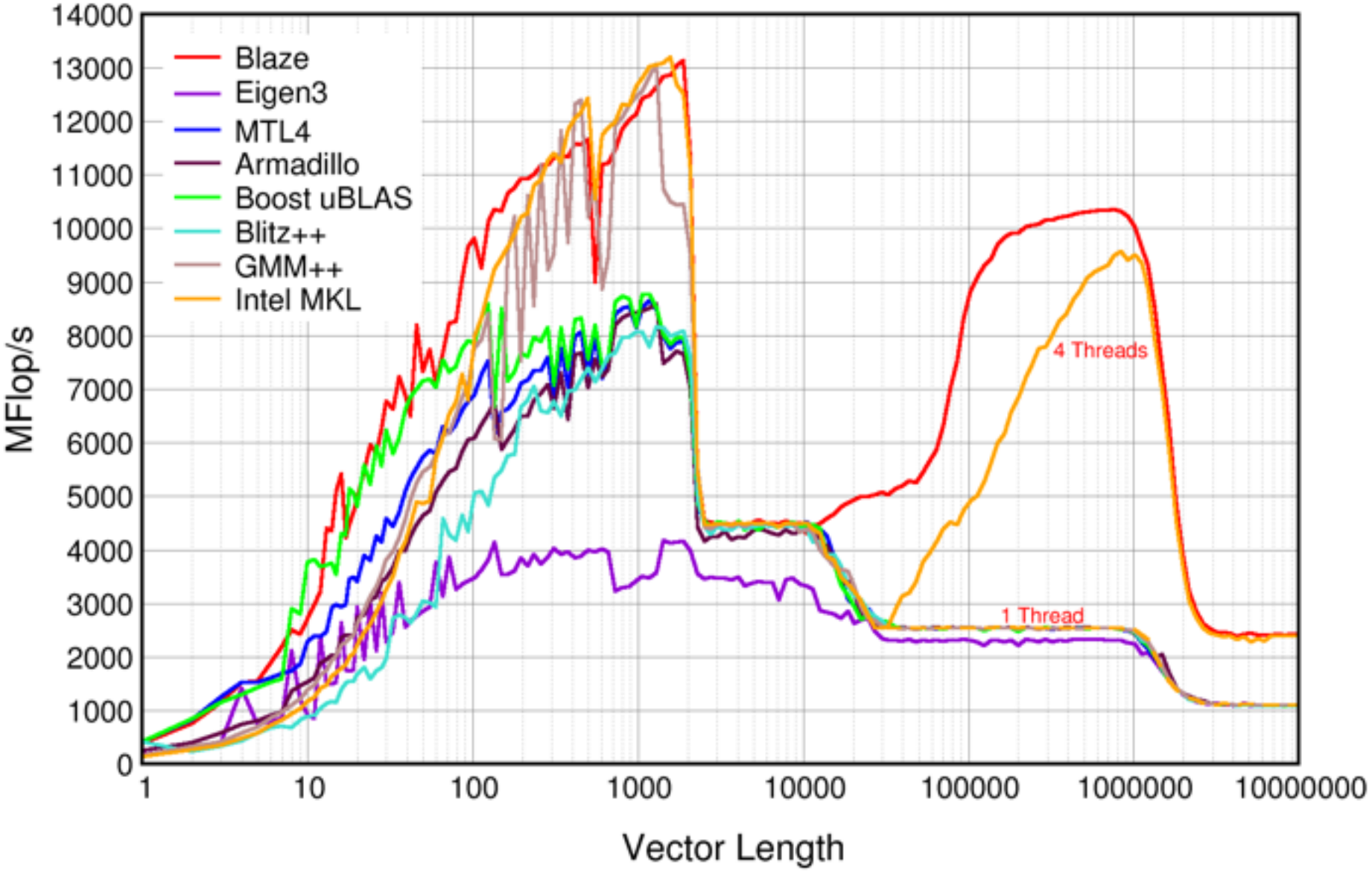
Clang 3.8


MFlop/s
Vector Length
15000
14000
13000
12000
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
1
10
100
1000
10000
100000
1000000
10000000
Blaze
Eigen3
MTL4
Armadillo
Boost uBLAS
Blitz++
GMM++
Intel MKL

GNU compiler 6.1



GNU compiler 6.1 with OpenMP



CD-adapco
A Siemens Business

BLAS Level 2

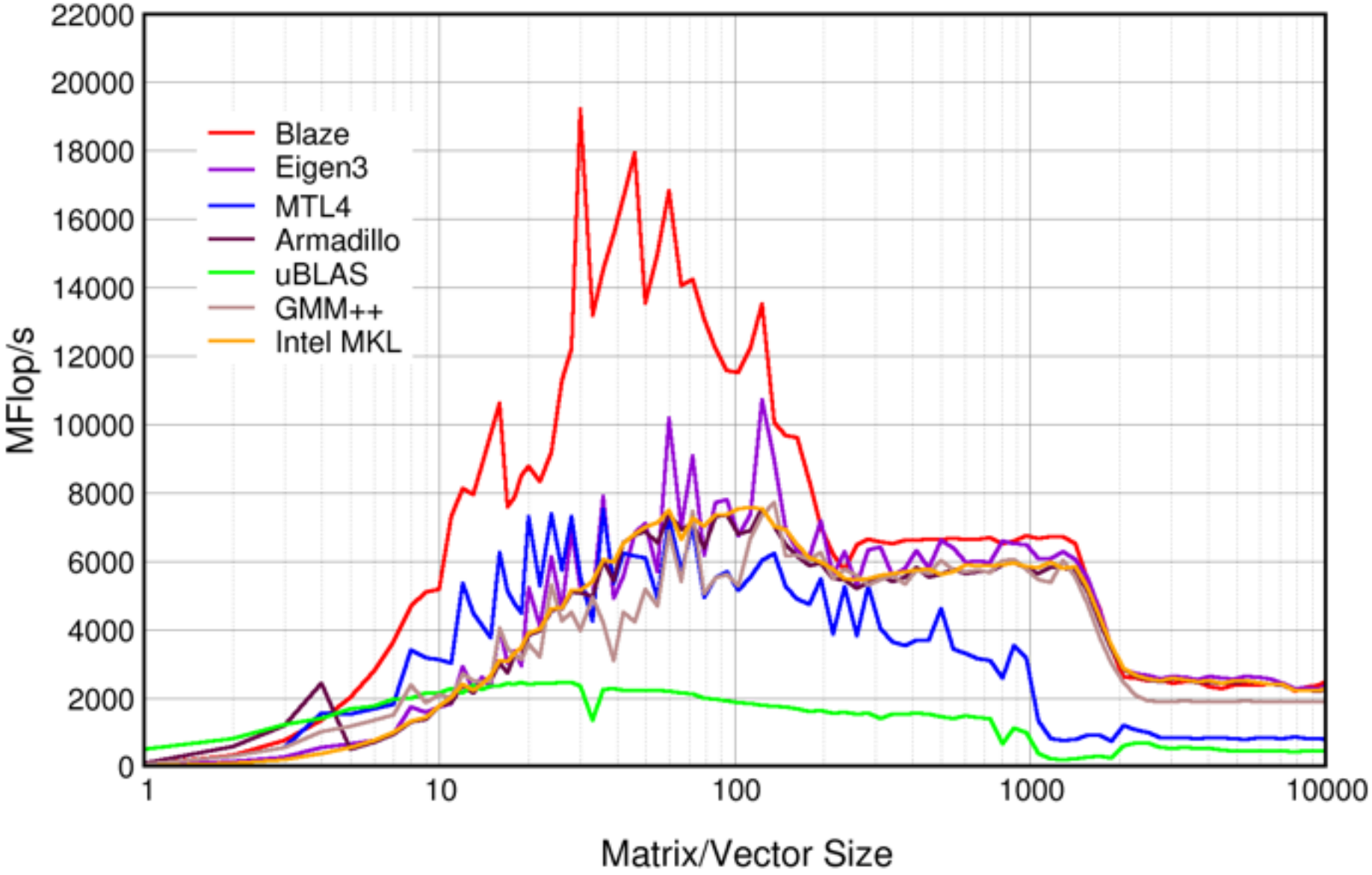
CD-adapco
A Siemens Business

```
template< typename Type, size_t M, size_t N, bool TF >
class StaticMatrix;


template< typename Type, bool TF >
class DynamicMatrix;


template< typename Type, size_t M, size_t N, bool TF >
class HybridMatrix;
```

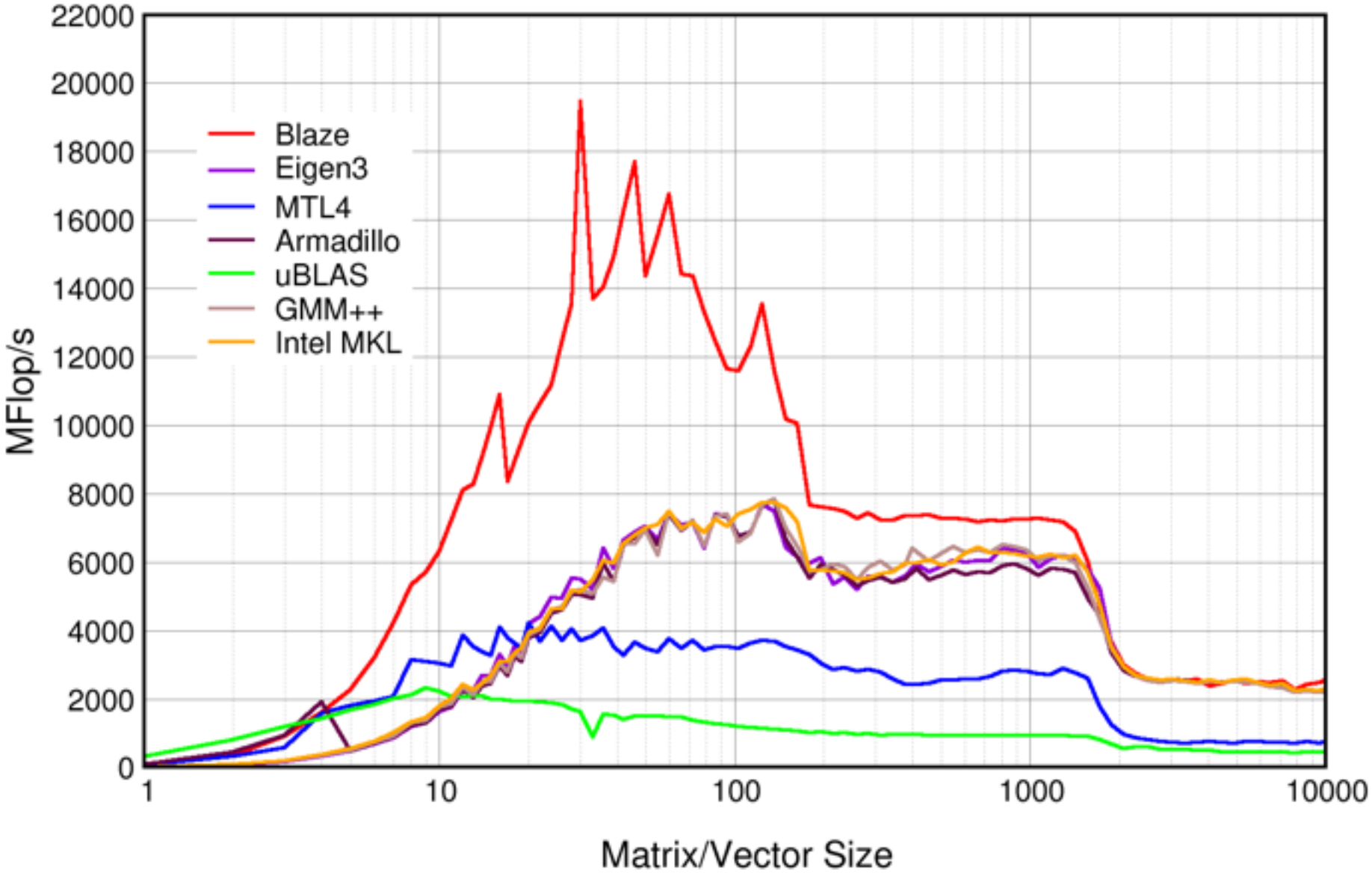
CD-adapco
A Siemens Business

```
using namespace blaze;

DynamicMatrix<double,columnMajor> A(N,N);
DynamicVector<double,columnVector> a(N), b(N);

// ... Initialization of the matrix and vector

b = A * a;
```

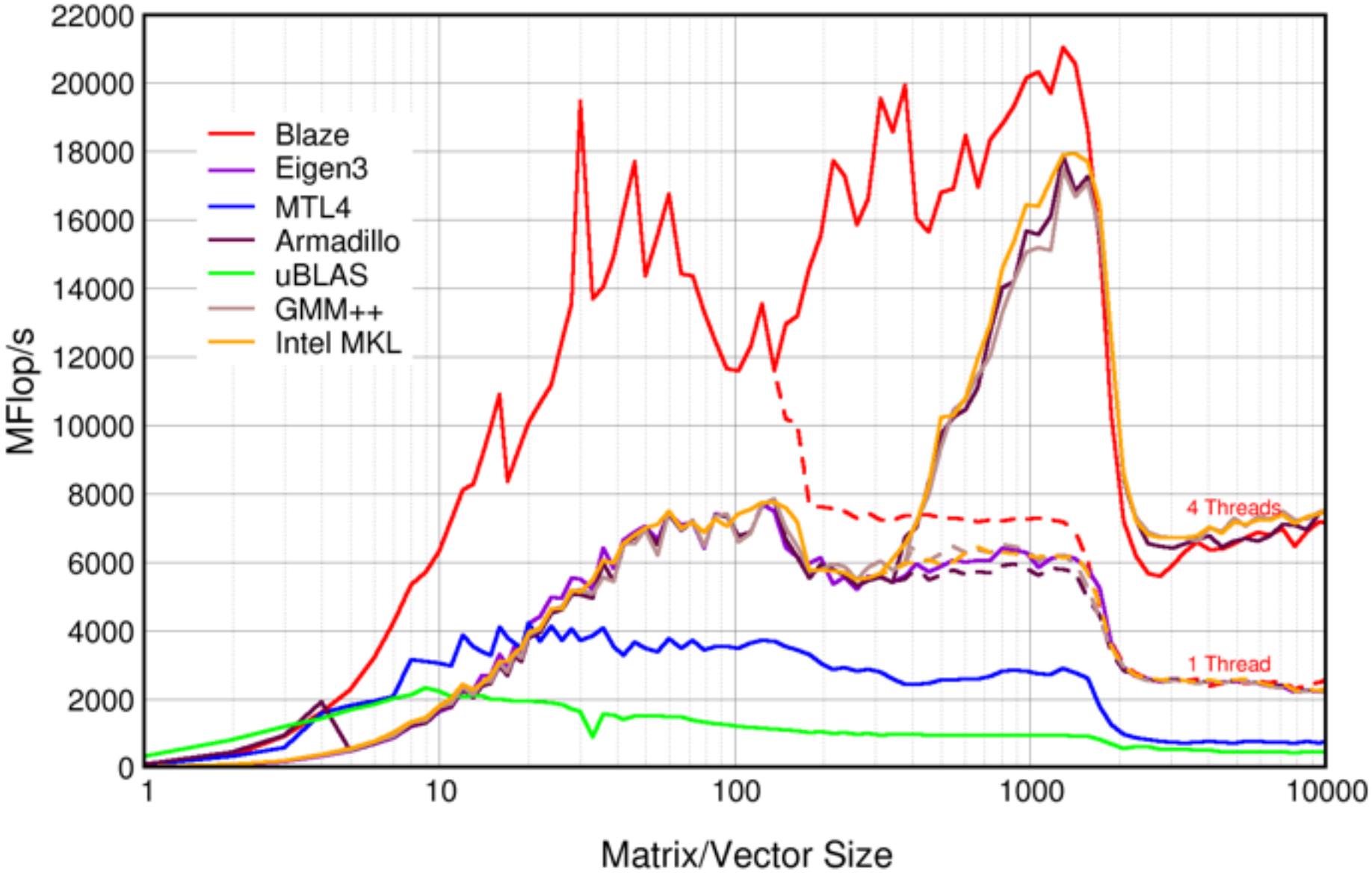
Clang 3.8


Blaze
Eigen3
MTL4
Armadillo
uBLAS
GMM++
Intel MKL
MFlop/s
Matrix/Vector Size
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
22000
1
10
100
1000
10000

GNU compiler 6.1



GNU compiler 6.1 with OpenMP



CD-adapco
A Siemens Business

```
template< typename VT1    // Type of the left-hand side target vector
        , typename MT1    // Type of the left-hand side matrix operand
        , typename VT2 >  // Type of the right-hand side vector operand
static inline void selectAssignKernel( VT1& y, const MT1& A, const VT2& x )
{
   if( ( IsDiagonal<MT1>::value ) ||
       ( IsComputation<MT>::value && !evaluateMatrix ) ||
       ( A.rows() * A.columns() < TDMATDVECMULT_THRESHOLD ) ) {
      selectSmallAssignKernel( y, A, x );
   }
   else {
      selectLargeAssignKernel( y, A, x );
   }
}
```

CD-adapco
A Siemens Business

```
template< typename VT1    // Type of the left-hand side target vector
        , typename MT1    // Type of the left-hand side matrix operand
        , typename VT2 >  // Type of the right-hand side vector operand
static inline void selectAssignKernel( VT1& y, const MT1& A, const VT2& x )
{
   if( ( IsDiagonal<MT1>::value ) ||
       ( IsComputation<MT>::value && !evaluateMatrix ) ||
       ( A.rows() * A.columns() < TDMATDVECMULT_THRESHOLD ) ) {
      selectSmallAssignKernel( y, A, x );
   }
   else {
      selectLargeAssignKernel( y, A, x );
   }
}
```

CD-adapco
A Siemens Business

```
template< typename VT1    // Type of the left-hand side target vector
        , typename MT1    // Type of the left-hand side matrix operand
        , typename VT2 >  // Type of the right-hand side vector operand
static inline void selectAssignKernel( VT1& y, const MT1& A, const VT2& x )
{
   if( ( IsDiagonal<MT1>::value ) ||
       ( IsComputation<MT>::value && !evaluateMatrix ) ||
       ( A.rows() * A.columns() < TDMATDVECMULT_THRESHOLD ) ) {
      selectSmallAssignKernel( y, A, x );
   }
   else {
      selectLargeAssignKernel( y, A, x );
   }
}
```

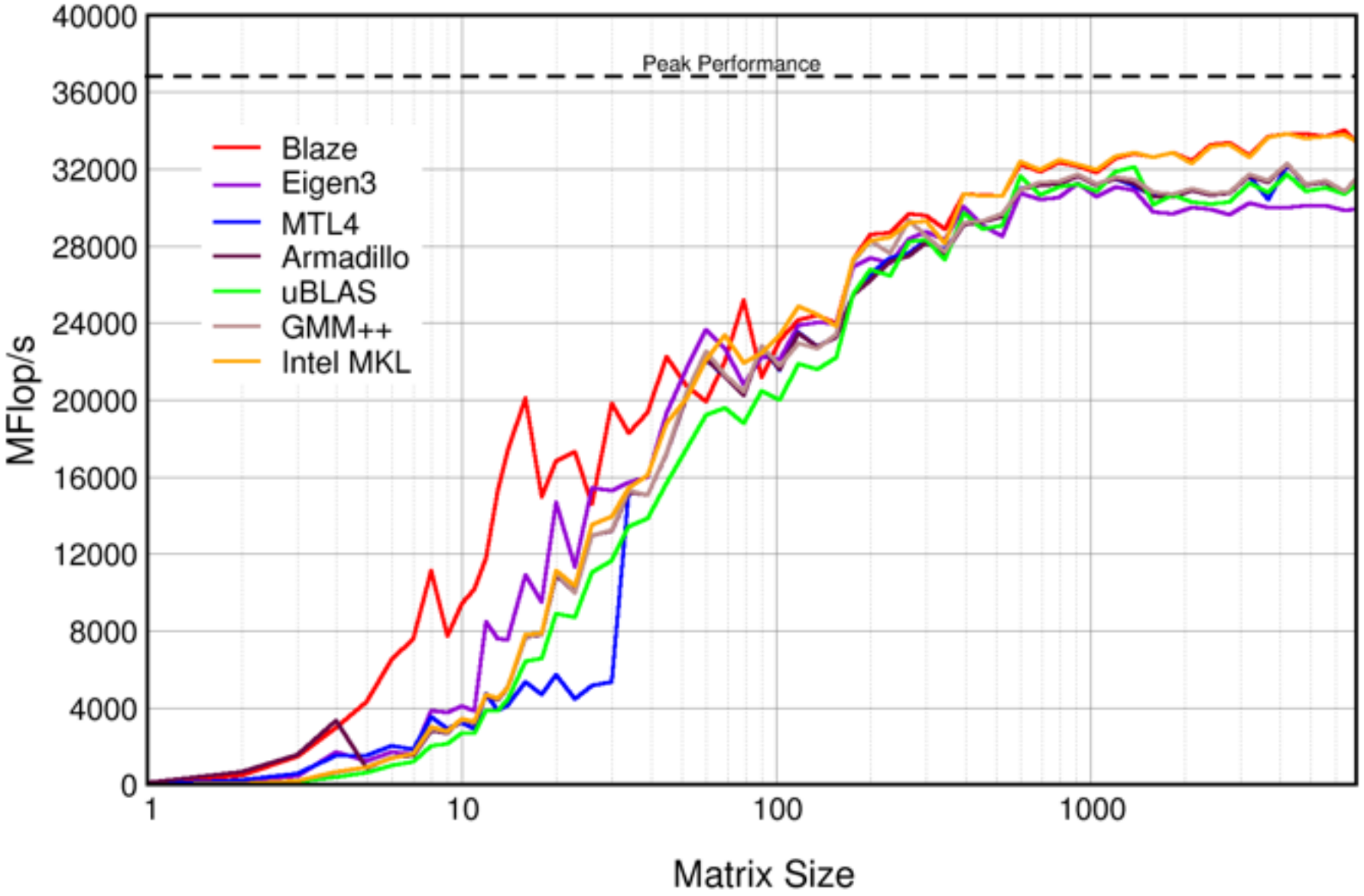
CD-adapco
A Siemens Business

BLAS Level 3

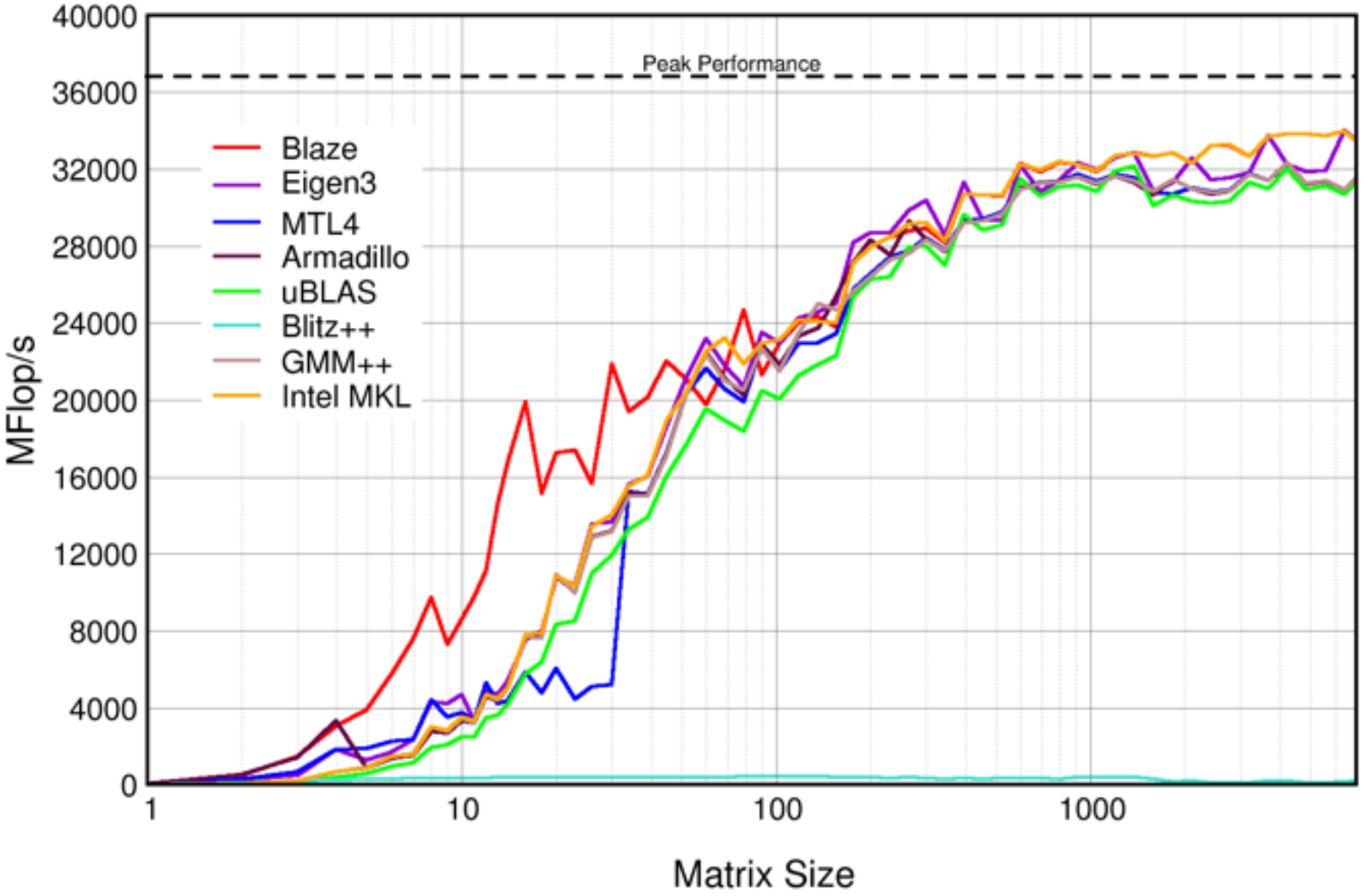
CD-adapco
A Siemens Business

```
using namespace blaze;

DynamicMatrix<double,columnMajor> A(N,N), B(N,N), C(N,N);

// ... Initialization of the matrices

C = A * B;
```

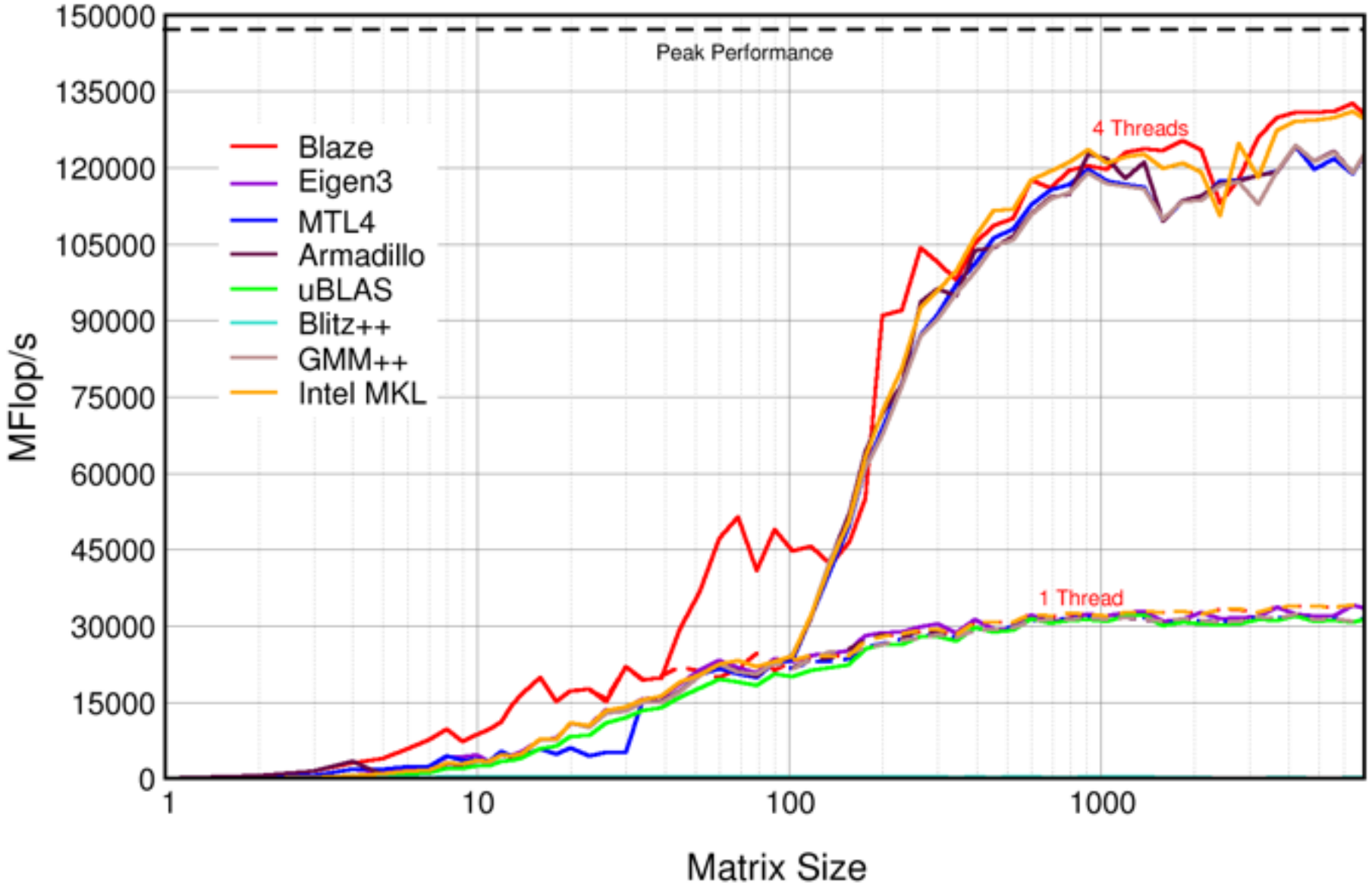
Clang 3.8


40000
36000
32000
28000
24000
20000
16000
12000
8000
4000
0
MFlop/s
Peak Performance
Blaze
Eigen3
MTL4
Armadillo
uBLAS
GMM++
Intel MKL
1
10
100
1000
Matrix Size

GNU compiler 6.1



GNU compiler 6.1 with OpenMP



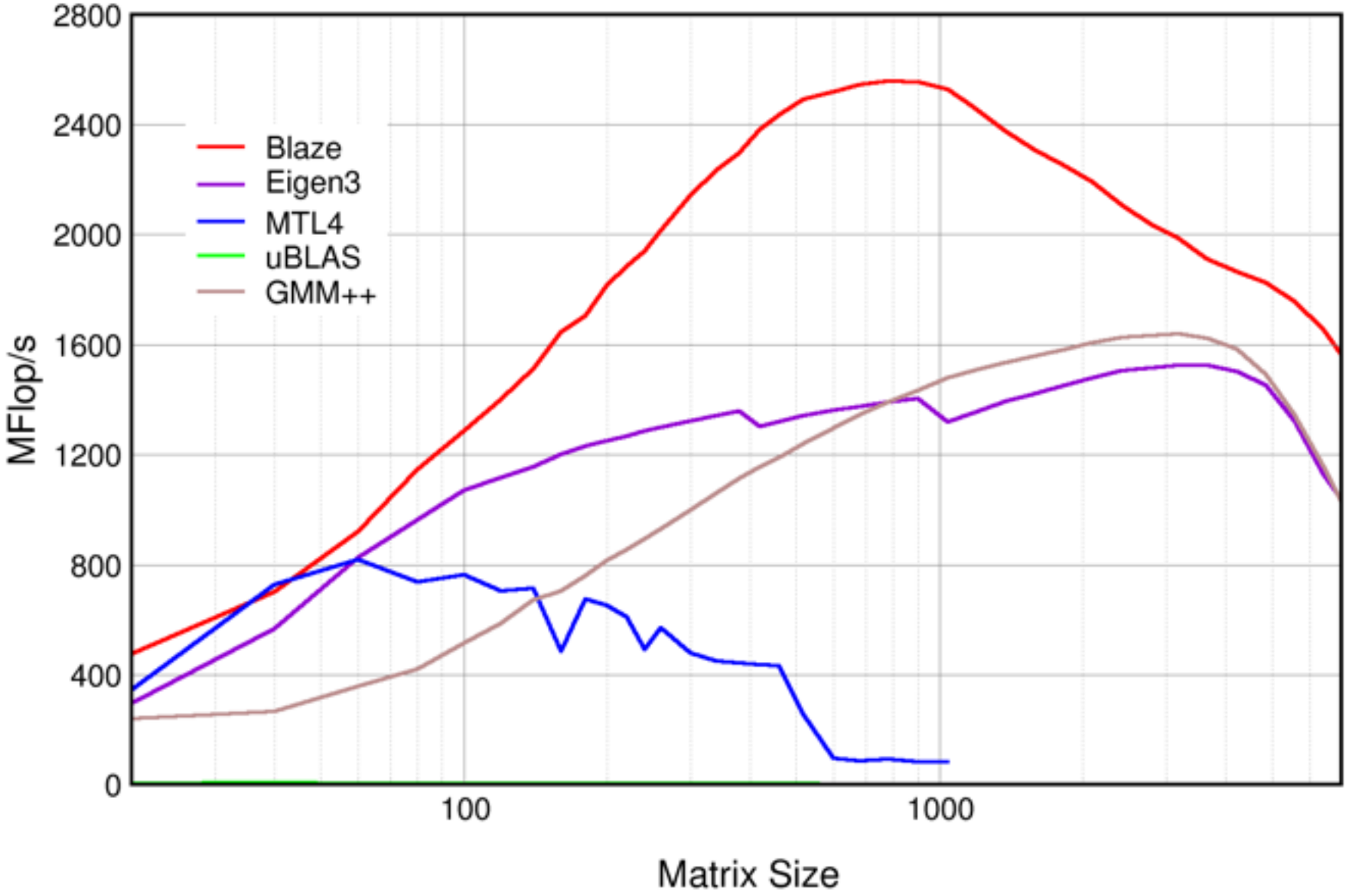
CD-adapco
A Siemens Business

```
template< typename Type, bool TF >
class CompressedMatrix;
```

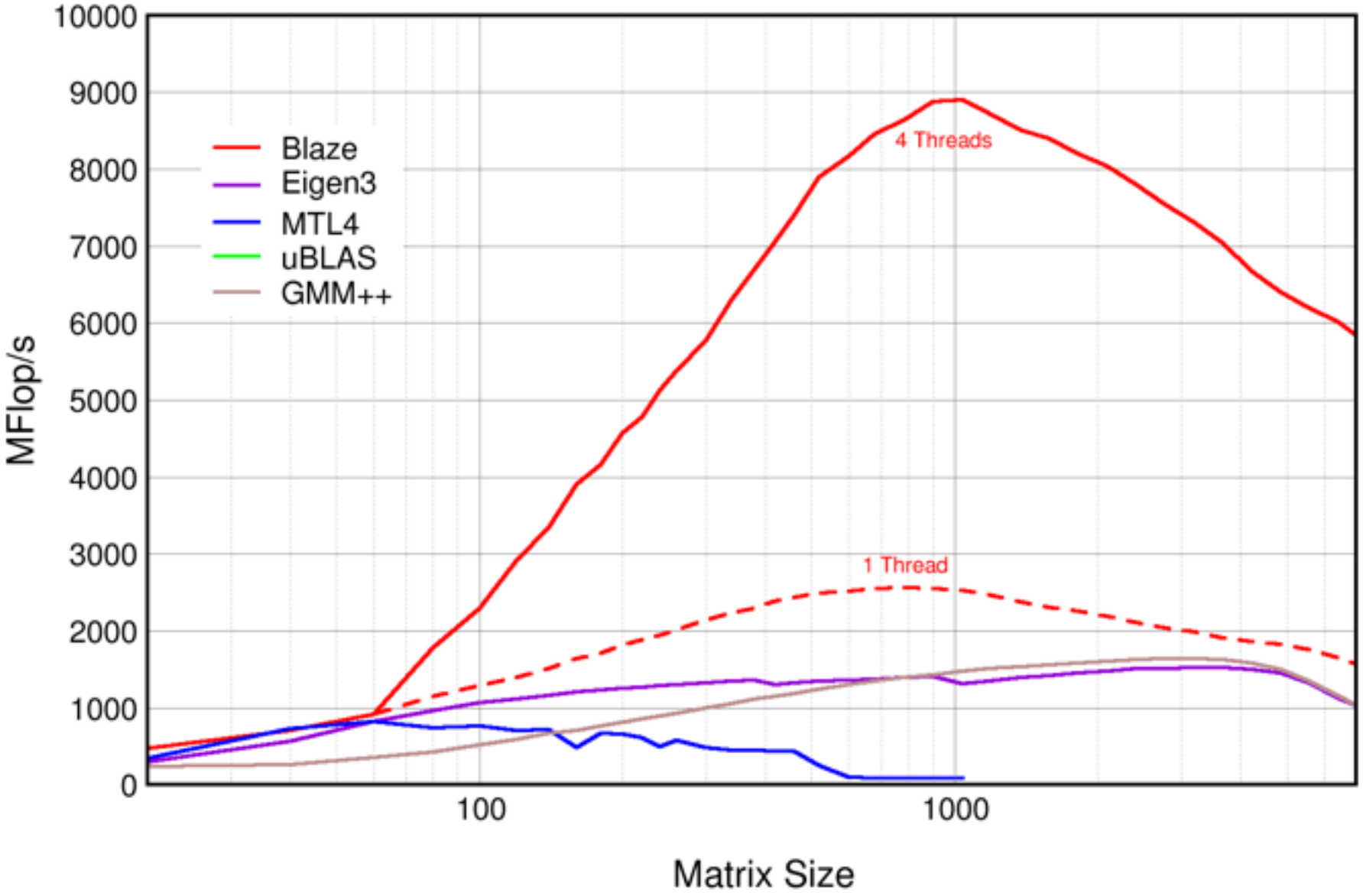
CD-adapco
A Siemens Business

```
using namespace blaze;

CompressedMatrix<double,columnMajor> A(N,N);
DynamicMatrix<double,columnMajor> B(N,N), C(N,N);

// ... Initialization of the matrices

C = A * B;
```

GNU compiler 6.1



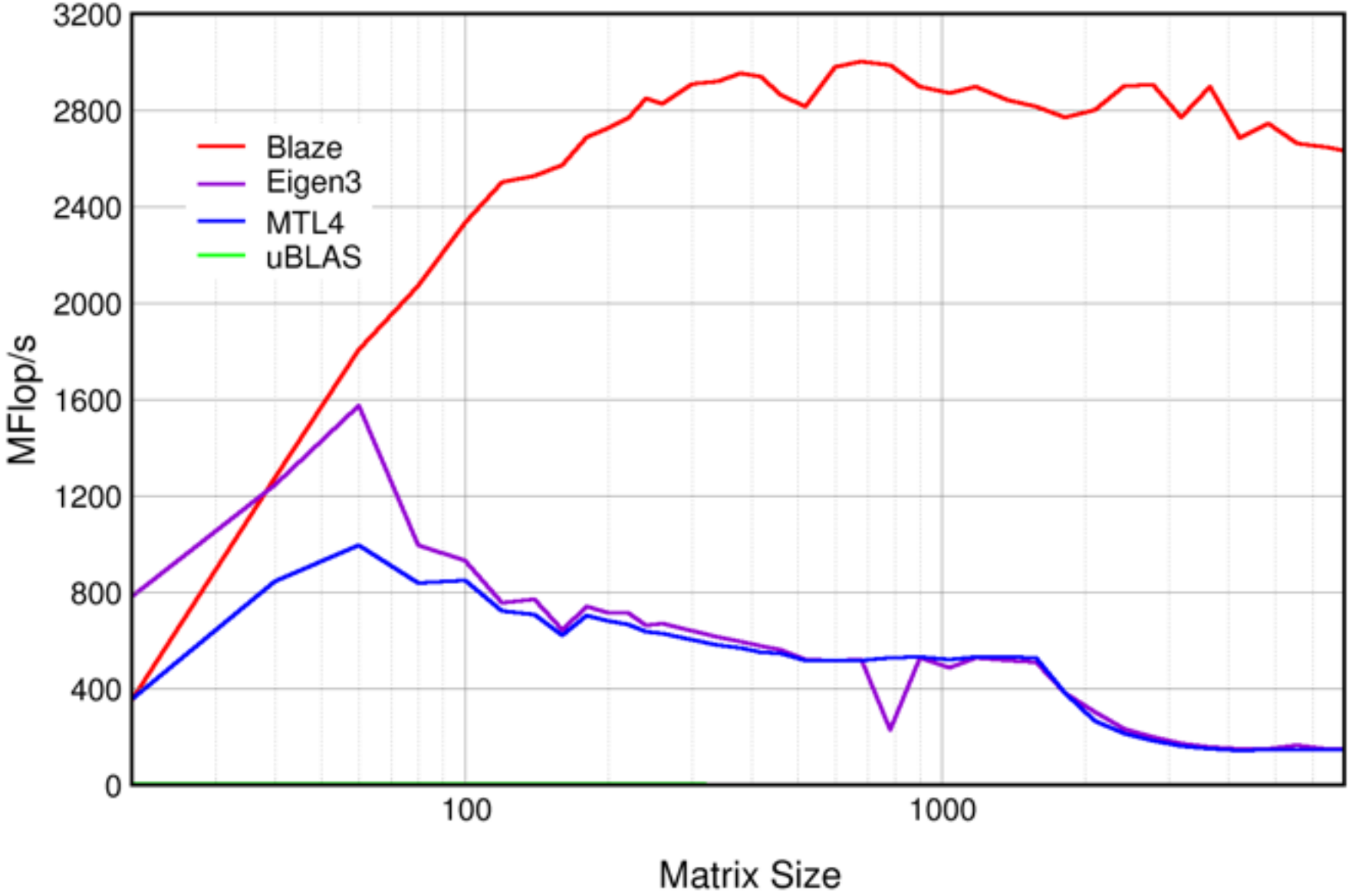
GNU compiler 6.1 with OpenMP



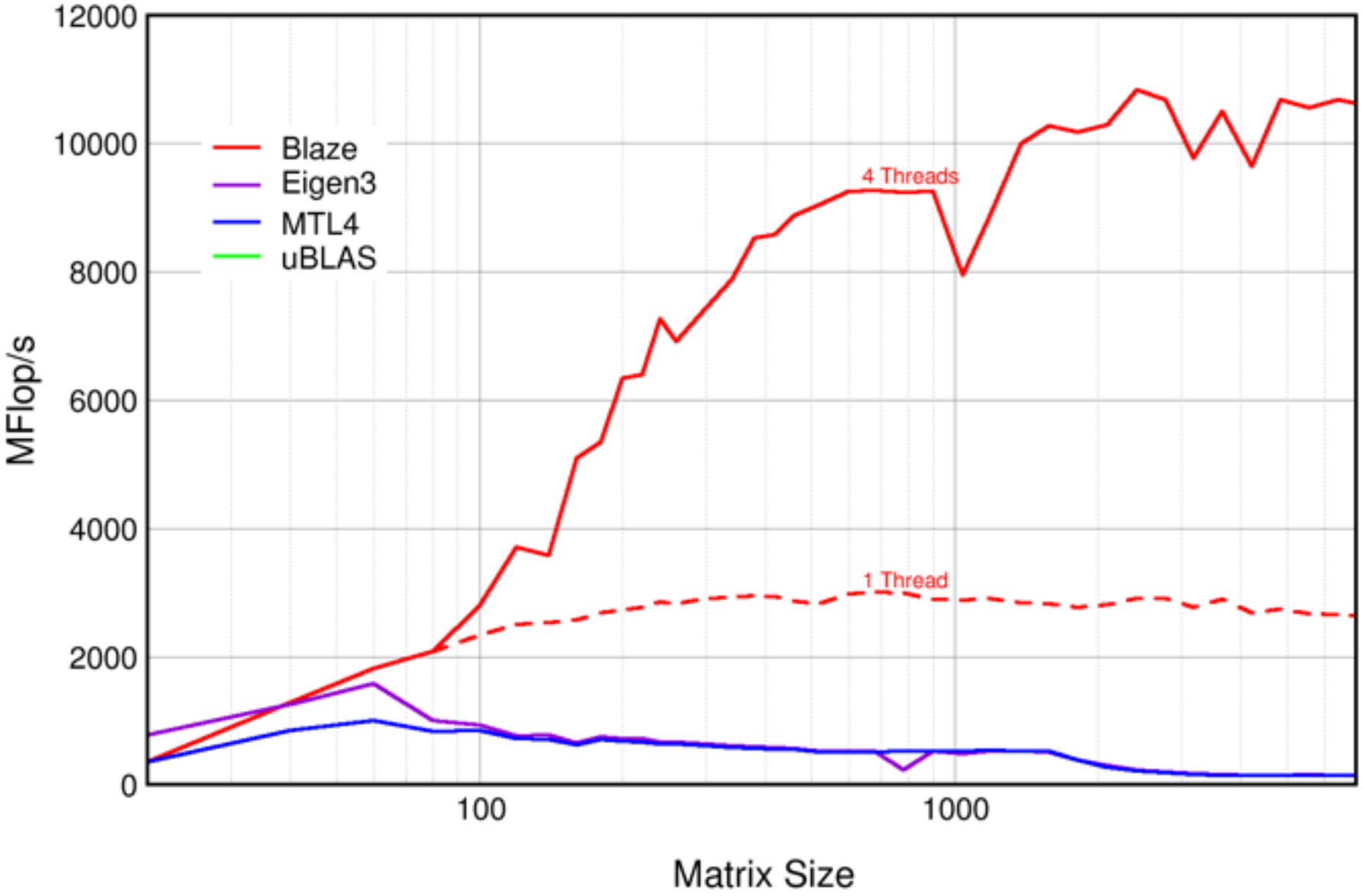
CD-adapco
A Siemens Business

```
using namespace blaze;

CompressedMatrix<double,columnMajor> A(N,N);
DynamicMatrix<double,rowMajor> B(N,N);
DynamicMatrix<double,columnMajor> C(N,N);

// ... Initialization of the matrices

C = A * B;
```

GNU compiler 6.1



GNU compiler 6.1 with OpenMP



CD-adapco
A Siemens Business

```
template< typename MT3    // Type of the left-hand side target matrix
        , typename MT4    // Type of the left-hand side matrix operand
        , typename MT5 >  // Type of the right-hand side matrix operand
static inline EnableIf_< UseOptimizedKernel<MT3,MT4,MT5> >
   selectLargeAssignKernel( MT3& C, const MT4& A, const MT5& B )
{
   BLAZE_CONSTRAINT_MUST_BE_ROW_MAJOR_MATRIX_TYPE( OppositeType_<MT4> );

   const OppositeType_<MT4> tmp( serial( A ) );
   assign( C, tmp * B );
}
```

CD-adapco
A Siemens Business

```
template< typename MT3    // Type of the left-hand side target matrix
        , typename MT4    // Type of the left-hand side matrix operand
        , typename MT5 >  // Type of the right-hand side matrix operand
static inline EnableIf_< UseOptimizedKernel<MT3,MT4,MT5> >
   selectLargeAssignKernel( MT3& C, const MT4& A, const MT5& B )
{
   BLAZE_CONSTRAINT_MUST_BE_ROW_MAJOR_MATRIX_TYPE( OppositeType_<MT4> );

   const OppositeType_<MT4> tmp( serial( A ) );
   assign( C, tmp * B );
}
```

CD-adapco
A Siemens Business

Complex expressions

CD-adapco
A Siemens Business

```
using namespace blaze;

DynamicMatrix<double,columnMajor> A(N,N), B(N,N);
DynamicVector<double> a(N), b(N), c(N);

// ... Initialization of the matrices and vectors

c = A * B * ( a + b );
```

CD-adapco
A Siemens Business

```
using namespace boost::numeric::ublas;

matrix<double,column_major> A(N,N), B(N,N);
vector<double> a(N), b(N), c(N);
vector<double> tmp(N);

// ... Initialization of the matrices and vectors

tmp = prod( B, ( a + b ) );
noalias( c ) = prod( A, tmp );
```

CD-adapco
A Siemens Business

```
using namespace gmm;

dense_matrix<double> A(N,N), B(N,N);
std::vector<double> a(N), b(N), c(N);
std::vector<double> tmp1(N), tmp2(N);

// ... Initialization of the matrices and vectors

add( a, b, tmp1 );
mult( B, tmp1, tmp2 );
mult( A, tmp2, c );
```

CD-adapco
A Siemens Business

```
using namespace mtl;

using parameters = mat::parameters<tag::col_major>;
using dense2D = dense2D<double,parameters>;

dense2D A(N,N), B(N,N), C(N,N);
dense_vector a(N), b(N), c(N);
dense_vector tmp1(N), tmp2(N);

// ... Initialization of the matrices

tmp1 = a + b;
tmp2 = B * tmp1;
c    = A * tmp2;
```

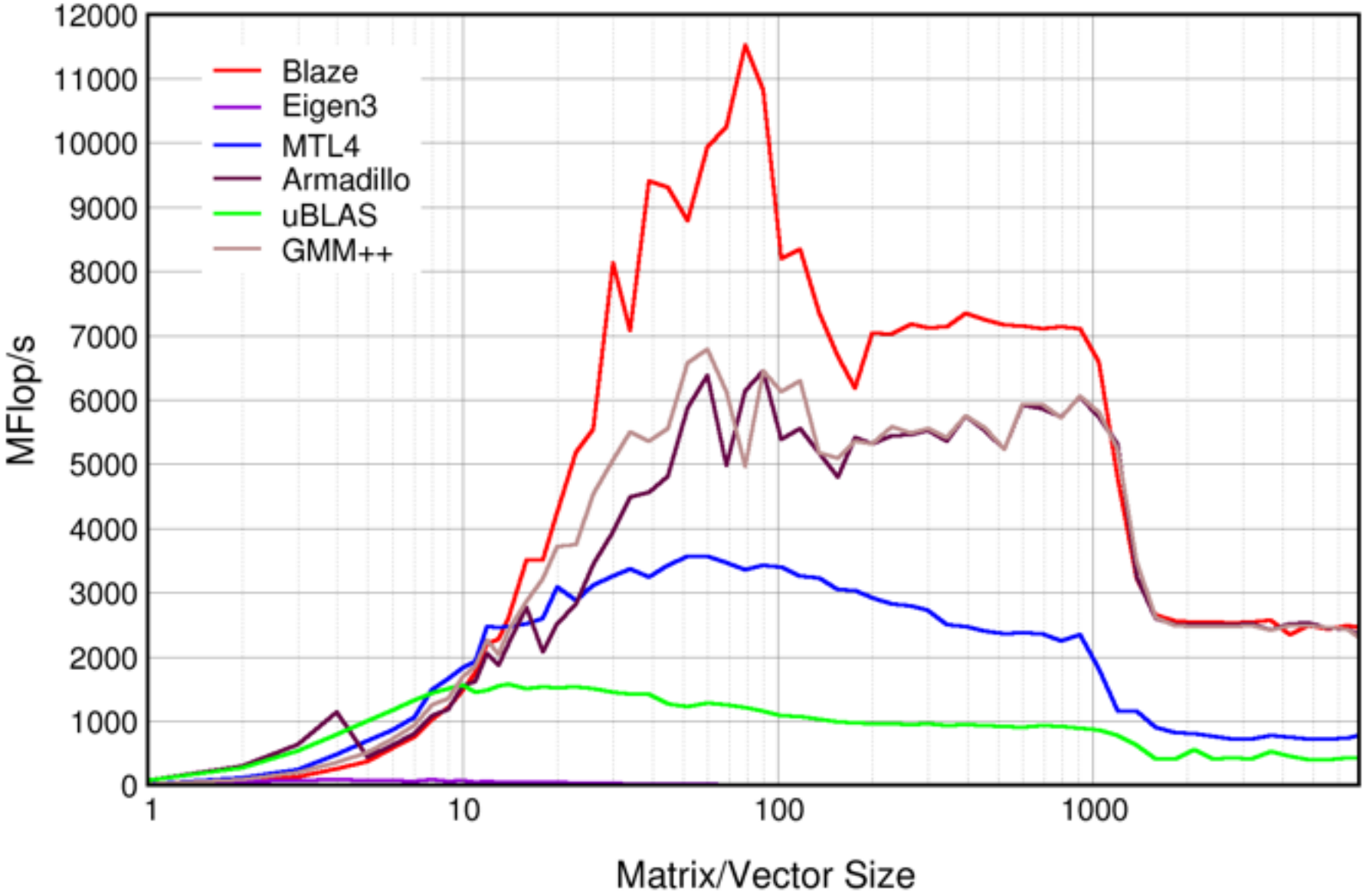
CD-adapco
A Siemens Business

```
using namespace arma;

Mat<double> A(N,N), B(N,N);
Col<double> a(N), b(N), c(N);

// ... Initialization of the matrices and vectors

c = A * B * ( a + b );
```

CD-adapco
A Siemens Business

```
using namespace Eigen;

Matrix<double,Dynamic,Dynamic,ColMajor> A(N,N), B(N,N);
Matrix<double,Dynamic,1> a(N), b(N), c(N);

// ... Initialization of the matrices and vectors

c.noalias() = A * B * ( a + b );
```

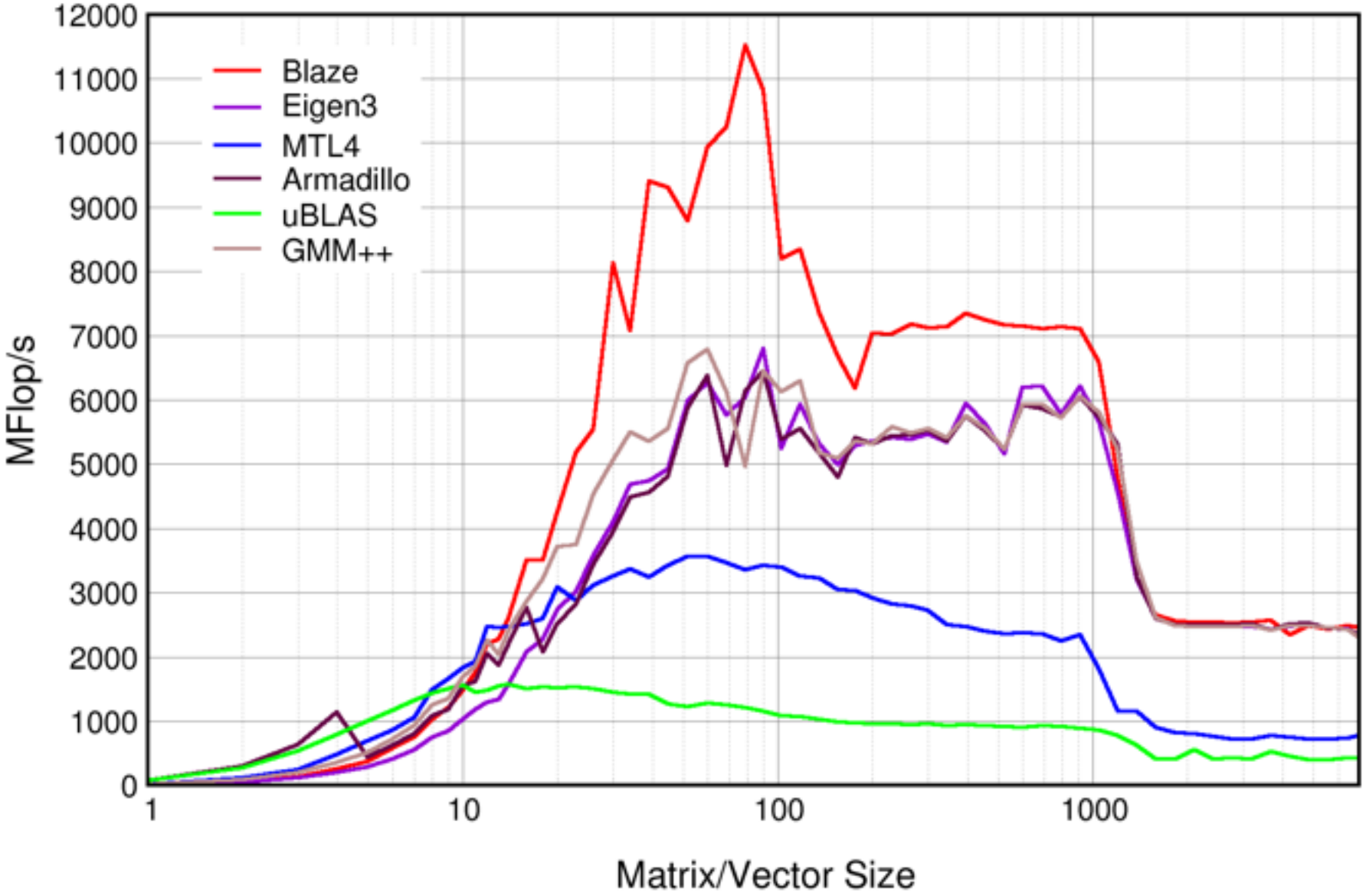
GNU compiler 6.1



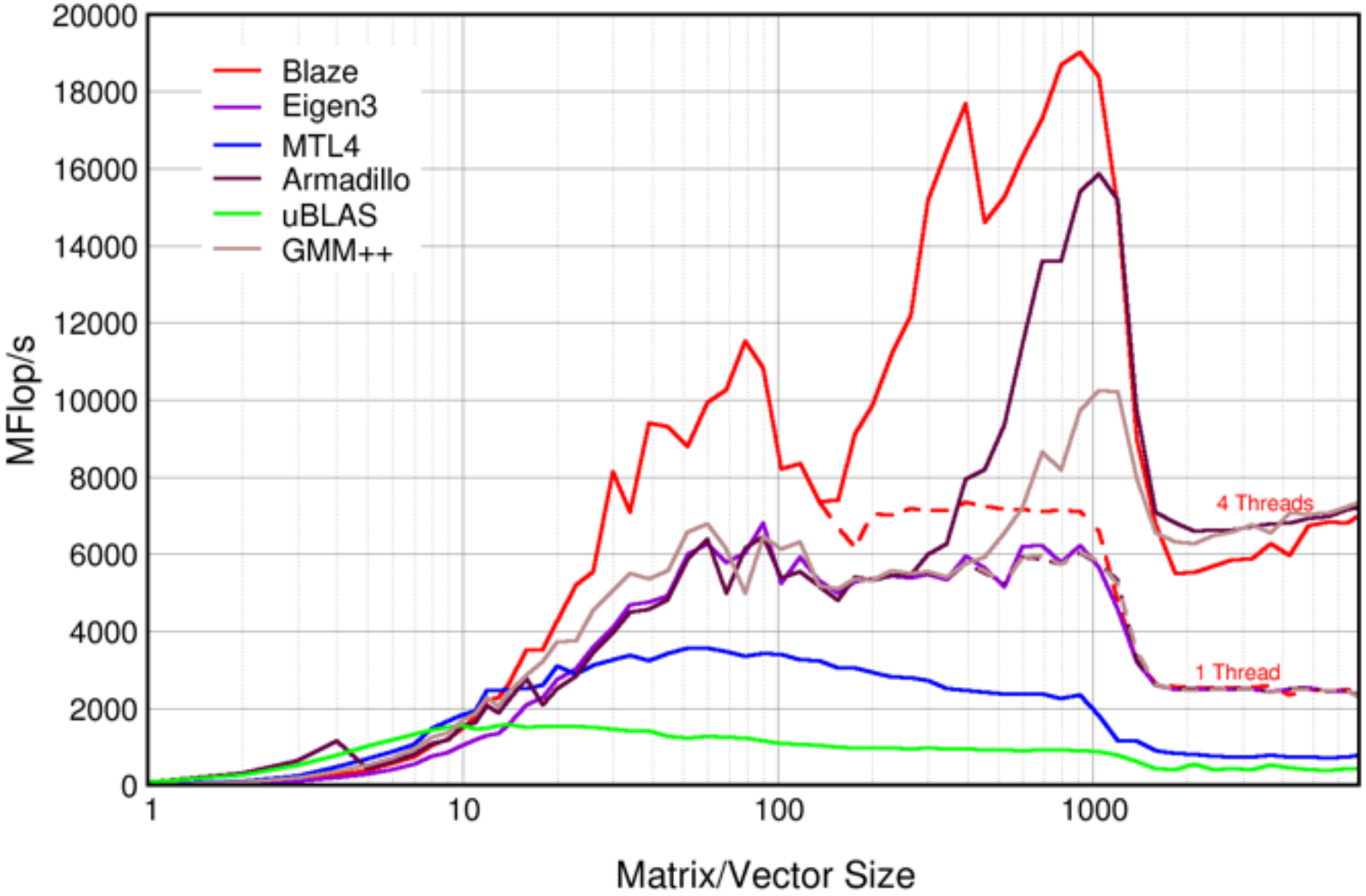
CD-adapco
A Siemens Business

```
using namespace Eigen;

Matrix<double,Dynamic,Dynamic,ColMajor> A(N,N), B(N,N);
Matrix<double,Dynamic,1> a(N), b(N), c(N);
Matrix<double,Dynamic,1> tmp1(N), tmp2(N);

// ... Initialization of the matrices and vectors

tmp1 = a + b;
tmp2 = B * tmp1;
c.noalias() = A tmp2;
```

GNU compiler 6.1



GNU compiler 6.1 with OpenMP



CD-adapco
A Siemens Business

```
template< typename T1    // Type of the left-hand side dense matrix
        , bool SO        // Storage order of the left-hand side dense matrix
        , typename T2 >  // Type of the right-hand side dense vector
inline const EnableIf_< IsMatMatMultExpr<T1>, MultExprTrait_<T1,T2> >
   operator*( const DenseMatrix<T1,SO>& mat, const DenseVector<T2,false>& vec )
{
   BLAZE_FUNCTION_TRACE;

   BLAZE_CONSTRAINT_MUST_NOT_BE_SYMMETRIC_MATRIX_TYPE( T1 );

   return (~mat).leftOperand() * ( (~mat).rightOperand() * vec );
}
```

CD-adapco
A Siemens Business

```
template< typename T1    // Type of the left-hand side dense matrix
        , bool SO        // Storage order of the left-hand side dense matrix
        , typename T2 >  // Type of the right-hand side dense vector
inline const EnableIf_< IsMatMatMultExpr<T1>, MultExprTrait_<T1,T2> >
   operator*( const DenseMatrix<T1,SO>& mat, const DenseVector<T2,false>& vec )
{
   BLAZE_FUNCTION_TRACE;

   BLAZE_CONSTRAINT_MUST_NOT_BE_SYMMETRIC_MATRIX_TYPE( T1 );

   return (~mat).leftOperand() * ( (~mat).rightOperand() * vec );
}
```

CD-adapco
A Siemens Business

Views

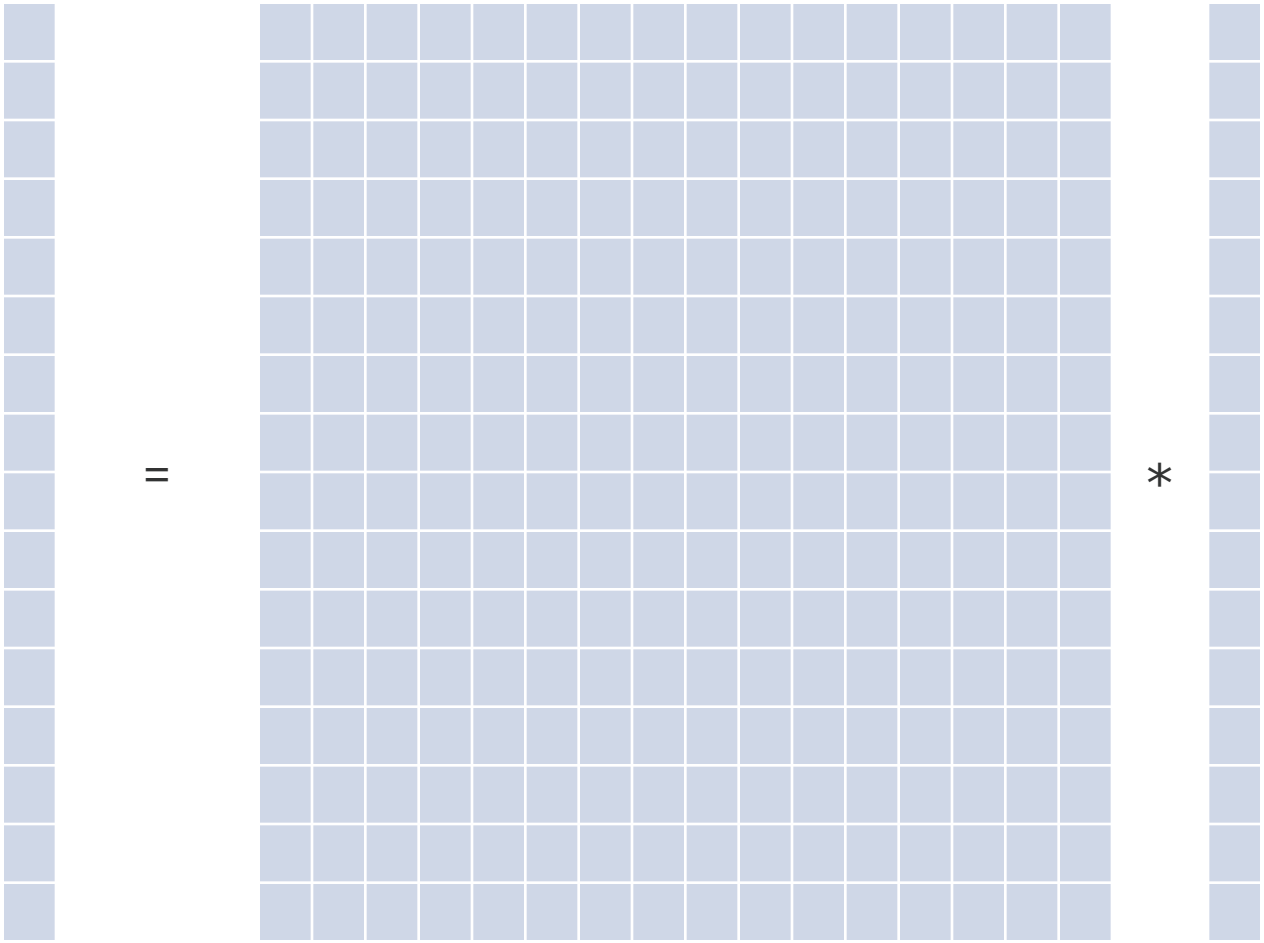
CD-adapco
A Siemens Business

```
template< typename VT >
class Subvector;


template< typename MT >
class Submatrix;


template< typename MT >
class Row;


template< typename MT >
class Column;
```

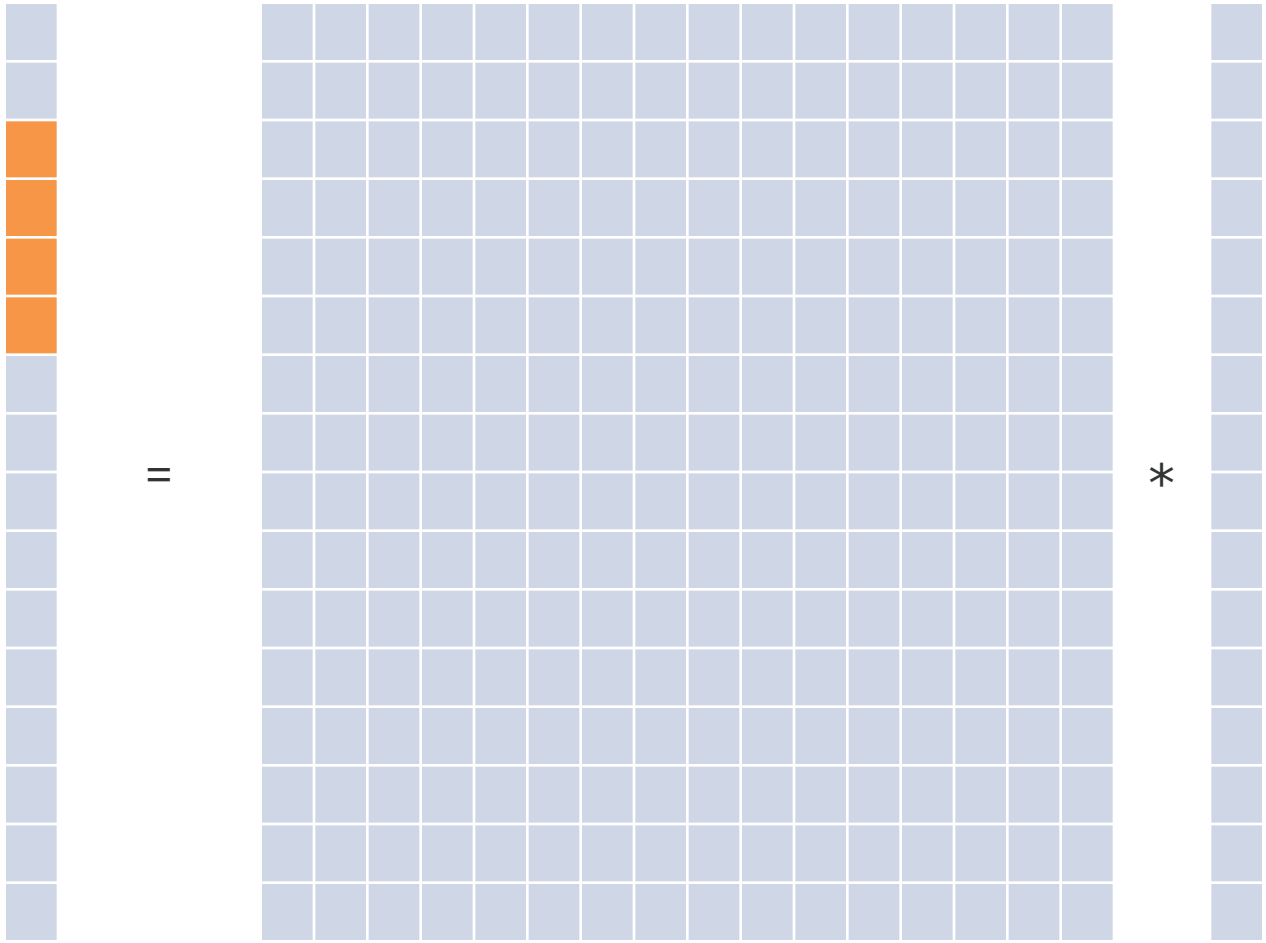
CD-adapco
A Siemens Business

```
row( A, 2U ) = subvector( x, 5U, 8U );

row( submatrix( A, 4U, 16U, 8U, 4U ), 3U ) = { 1, 2, 3, 4 };

subvector( y, 2U, 4U ) = subvector( A * x, 2U, 4U );
```

```
subvector( y, 2U, 4U ) = subvector( A * x, 2U, 4U );
```

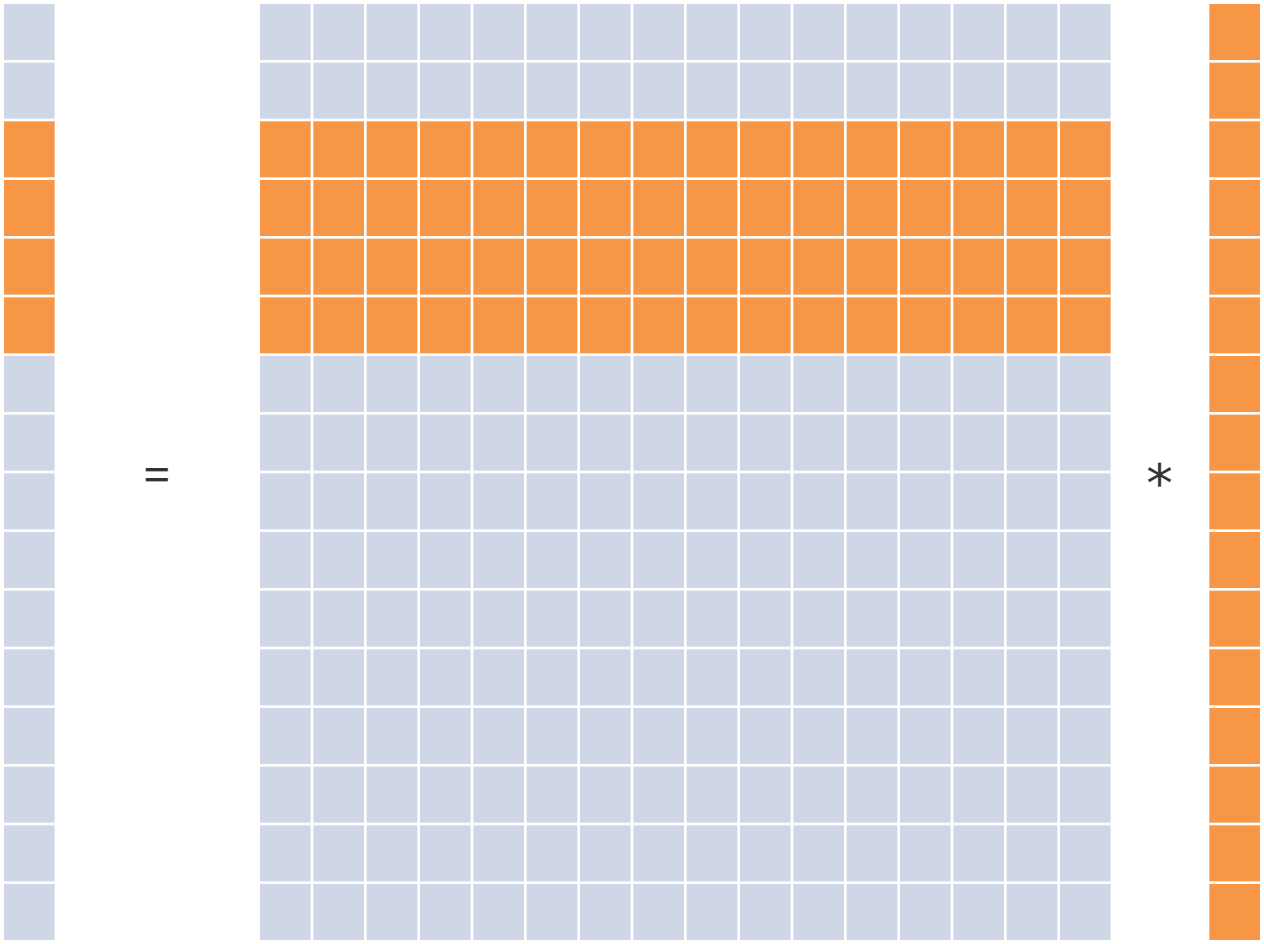


```
subvector( y, 2U, 4U ) = subvector( A * x, 2U, 4U );
```



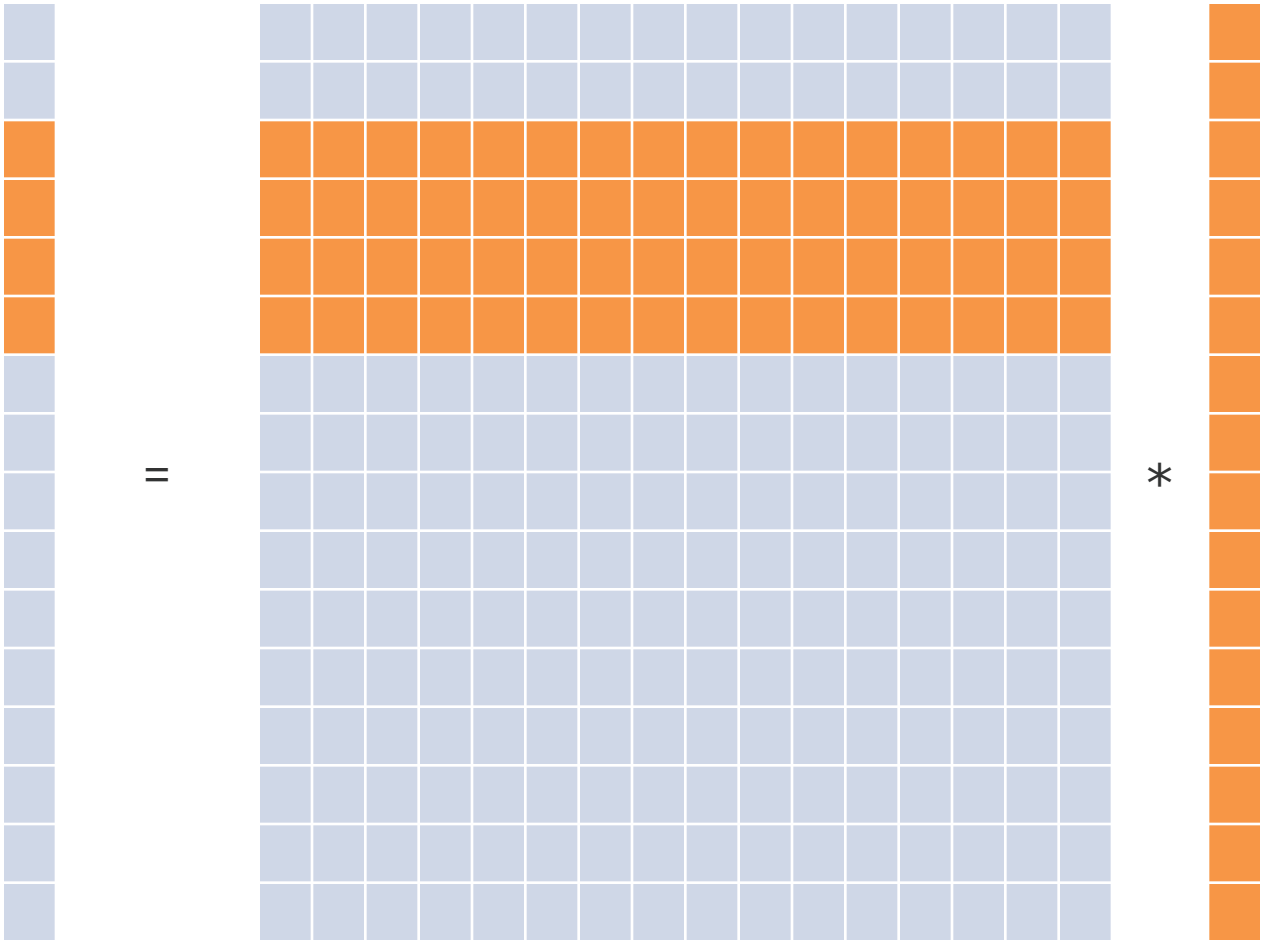
CD-adapco
A Siemens Business

```
subvector( y, 2U, 4U ) = subvector( A * x, 2U, 4U );
```



CD-adapco
A Siemens Business

```
subvector( y, 2U, 4U ) = subvector( A * x, 2U, 4U );
subvector( y, 2U, 4U ) = submatrix( A, 2U, 0U, 4U, N ) * x;
```



```
template< typename VT1  // Type of the left-hand side dense vector
        , bool TF1      // Transpose flag of the left-hand side dense vector
        , typename VT2  // Type of the right-hand side dense vector
        , bool TF2 >    // Transpose flag of the right-hand side dense vector
void smpAssign_backend( DenseVector<VT1,TF1>& lhs, const DenseVector<VT2,TF2>& rhs )
{
   BLAZE_FUNCTION_TRACE;

   BLAZE_INTERNAL_ASSERT( isParallelSectionActive(), "Invalid call outside a parallel section" );

   typedef ElementType_<VT1>                    ET1;
   typedef ElementType_<VT2>                    ET2;
   typedef SubvectorExprTrait_<VT1,aligned>     AlignedTarget;
   typedef SubvectorExprTrait_<VT1,unaligned>   UnalignedTarget;

   enum : size_t { SIMDSIZE = SIMDTrait< ElementType_<VT1> >::size };

   const bool simdEnabled( VT1::simdEnabled && VT2::simdEnabled && IsSame<ET1,ET2>::value );
   const bool lhsAligned ( (~lhs).isAligned() );
   const bool rhsAligned ( (~rhs).isAligned() );

   const int    threads      ( omp_get_num_threads() );
   const size_t addon        ( ( ( (~lhs).size() % threads ) != 0UL )? 1UL : 0UL );
   const size_t equalShare   ( (~lhs).size() / threads + addon );
   const size_t rest         ( equalShare & ( SIMDSIZE - 1UL ) );
   const size_t sizePerThread( ( simdEnabled && rest )?( equalShare - rest + SIMDSIZE ):
( equalShare ) );

#pragma omp for schedule(dynamic,1) nowait
   for( int i=0UL; i<threads; ++i )
   {
      const size_t index( i*sizePerThread );

      if( index >= (~lhs).size() )
         continue;

      const size_t size( min( sizePerThread, (~lhs).size() - index ) );
```

```
   const bool rhsAligned ( (~rhs).isAligned() );

   const int    threads      ( omp_get_num_threads() );
   const size_t addon        ( ( ( (~lhs).size() % threads ) != 0UL )? 1UL : 0UL );
   const size_t equalShare   ( (~lhs).size() / threads + addon );
   const size_t rest         ( equalShare & ( SIMDSIZE - 1UL ) );
   const size_t sizePerThread( ( simdEnabled && rest )?( equalShare - rest + SIMDSIZE ):
( equalShare ) );

#pragma omp for schedule(dynamic,1) nowait
   for( int i=0UL; i<threads; ++i )
   {
      const size_t index( i*sizePerThread );

      if( index >= (~lhs).size() )
         continue;

      const size_t size( min( sizePerThread, (~lhs).size() - index ) );

      if( simdEnabled && lhsAligned && rhsAligned ) {
         AlignedTarget target( subvector<aligned>( ~lhs, index, size ) );
         assign( target, subvector<aligned>( ~rhs, index, size ) );
      }
      else if( simdEnabled && lhsAligned ) {
         AlignedTarget target( subvector<aligned>( ~lhs, index, size ) );
         assign( target, subvector<unaligned>( ~rhs, index, size ) );
      }
      else if( simdEnabled && rhsAligned ) {
         UnalignedTarget target( subvector<unaligned>( ~lhs, index, size ) );
         assign( target, subvector<aligned>( ~rhs, index, size ) );
      }
      else {
         UnalignedTarget target( subvector<unaligned>( ~lhs, index, size ) );
         assign( target, subvector<unaligned>( ~rhs, index, size ) );
      }
   }
}
```

CD-adapco
A Siemens Business

```
   const bool rhsAligned ( (~rhs).isAligned() );

   const int    threads      ( omp_get_num_threads() );
   const size_t addon        ( ( ( (~lhs).size() % threads ) != 0UL )? 1UL : 0UL );
   const size_t equalShare   ( (~lhs).size() / threads + addon );
   const size_t rest         ( equalShare & ( SIMDSIZE - 1UL ) );
   const size_t sizePerThread( ( simdEnabled && rest )?( equalShare - rest + SIMDSIZE ):
( equalShare ) );

#pragma omp for schedule(dynamic,1) nowait
   for( int i=0UL; i<threads; ++i )
   {
      const size_t index( i*sizePerThread );

      if( index >= (~lhs).size() )
         continue;

      const size_t size( min( sizePerThread, (~lhs).size() - index ) );

      if( simdEnabled && lhsAligned && rhsAligned ) {
         AlignedTarget target( subvector<aligned>( ~lhs, index, size ) );
         assign( target, subvector<aligned>( ~rhs, index, size ) );
      }
      else if( simdEnabled && lhsAligned ) {
         AlignedTarget target( subvector<aligned>( ~lhs, index, size ) );
         assign( target, subvector<unaligned>( ~rhs, index, size ) );
      }
      else if( simdEnabled && rhsAligned ) {
         UnalignedTarget target( subvector<unaligned>( ~lhs, index, size ) );
         assign( target, subvector<aligned>( ~rhs, index, size ) );
      }
      else {
         UnalignedTarget target( subvector<unaligned>( ~lhs, index, size ) );
         assign( target, subvector<unaligned>( ~rhs, index, size ) );
      }
   }
}
```

CD-adapco
A Siemens Business

Custom Vectors and Matrices

CD-adapco
A Siemens Business

```
template< typename Type, bool AF, bool PF, bool TF >
class CustomVector;


template< typename Type, bool AF, bool PF, bool SO >
class CustomMatrix;
```

CD-adapco
A Siemens Business

```
using namespace blaze;

int* specialMemory = ...;

CustomVector<int,unaligned,unpadded,columnVector> a(
   specialMemory,
   42UL
);

int* alignedMemory = ...;

CustomMatrix<int,aligned,padded,rowMajor> B(
   alignedMemory,
   6U,
   7U,
   16U,
   Deleter()
);
```

CD-adapco
A Siemens Business

Custom Operations

CD-adapco
A Siemens Business

```
using namespace blaze;

DynamicMatrix<double> A, B;

// ... Initialization of the matrix A

B = forEach( A, []( double d )
                { return std::sqrt( d ); } );
```

CD-adapco
A Siemens Business

```
struct Sqrt
{
   double operator()( double a ) const
   {
      return std::sqrt( a );
   }
};

B = forEach( A, Sqrt() );
```

CD-adapco
A Siemens Business

```
struct Sqrt
{
   double operator()( double a ) const
   {
      return std::sqrt( a );
   }

   simd_double_t load( simd_double_t a ) const
   {
      return _mm256_sqrt_pd( a.value );
   }
};

B = forEach( A, Sqrt() );
```

CD-adapco
A Siemens Business

```
struct Sqrt
{
   double operator()( double a ) const
   {
      return std::sqrt( a );
   }

   template< typename T >
   T load( T a ) const
   {
      return _mm256_sqrt_pd( a.value );
   }

   template< typename T >
   static constexpr bool simdEnabled() {
#if defined(__AVX__)
      return true;
#else
      return false;
#endif
   }
};

B = forEach( A, Sqrt() );
```

Error Reporting Customization

CD-adapco
A Siemens Business

```
#define BLAZE_THROW( EXCEPTION ) \
   throw EXCEPTION
```

CD-adapco
A Siemens Business

```
#define BLAZE_THROW( EXCEPTION ) \
   log( "..." ); \
   abort()

#include <blaze/Blaze.h>
```

CD-adapco
A Siemens Business

Come on, another C++ math library?

CD-adapco
A Siemens Business

Yes, another C++ math library because there is still room for improvement.

Blaze …

- … fulfills HPC **performance** expectations
- … is less dependent on compiler optimization
- … is providing **support for multi-core CPUs**
- … is "**smart**" enough for the average user

Blaze's participation in the race benefits everyone!

CD-adapco
A Siemens Business

... because performance matters