BTL++: Mesure de performance et généricité

Laurent Plagne et Frank Hülsemann (EDF R&D)
{laurent.plagne,frank.hulsemann}@edf.fr

November 25, 2008

BTL++ ?

- BTL++ : *Benchmark Template Library in C++*

BTL++ ?

- BTL++ : *Benchmark Template Library in C++*
- Un outil pour mesurer les performances de noyaux de calculs définis par l'utilisateur (*kernels*).

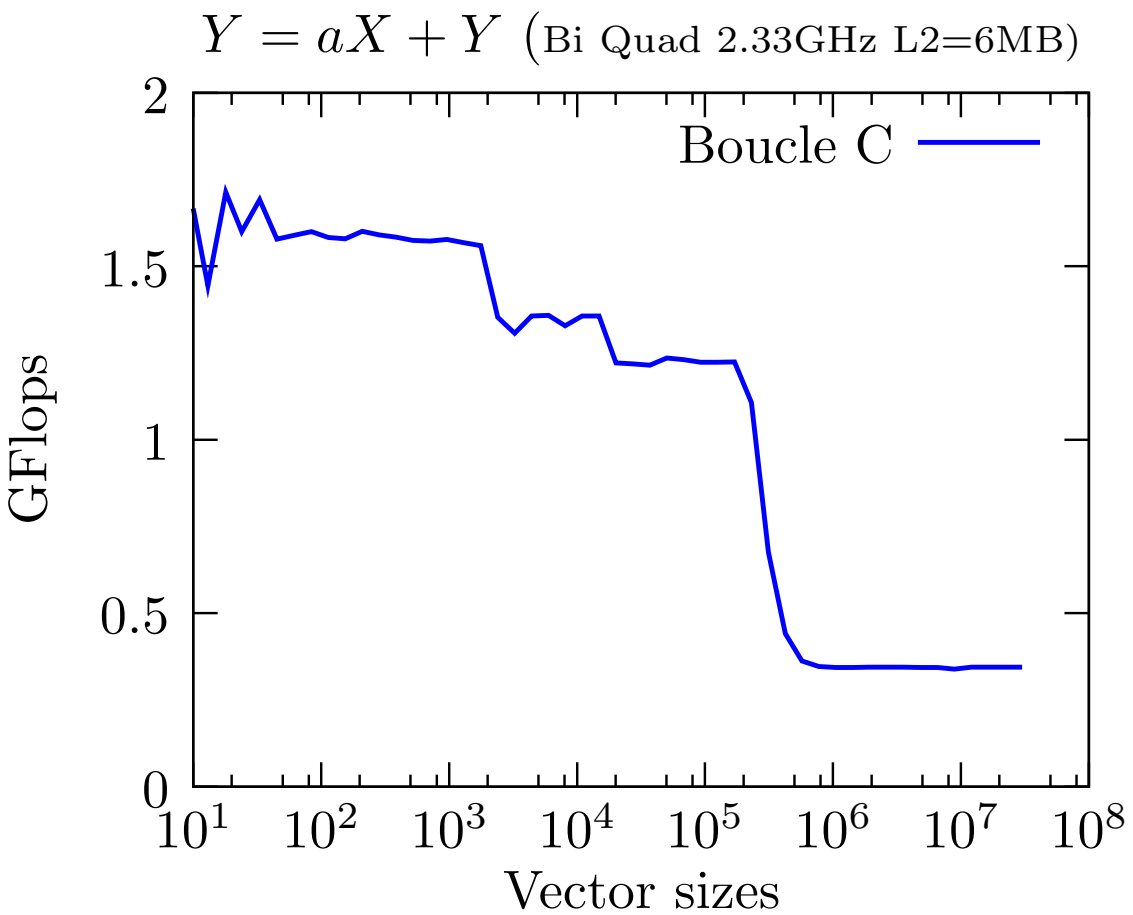
BTL++ ?

- BTL++ : *Benchmark Template Library in C++*
- Un outil pour mesurer les performances de noyaux de calculs définis par l'utilisateur (*kernels*).
- Un outil pour comparer différentes bibliothèques qui implémentent un noyau donné (*libraries*).

BTL++ ?

- BTL++ : *Benchmark Template Library in C++*
- Un outil pour mesurer les performances de noyaux de calculs définis par l'utilisateur (*kernels*).
- Un outil pour comparer différentes bibliothèques qui implémentent un noyau donné (*libraries*).
- Une bibliothèque générique qui met l'accent sur l'**extensibilité par l'utilisateur**.

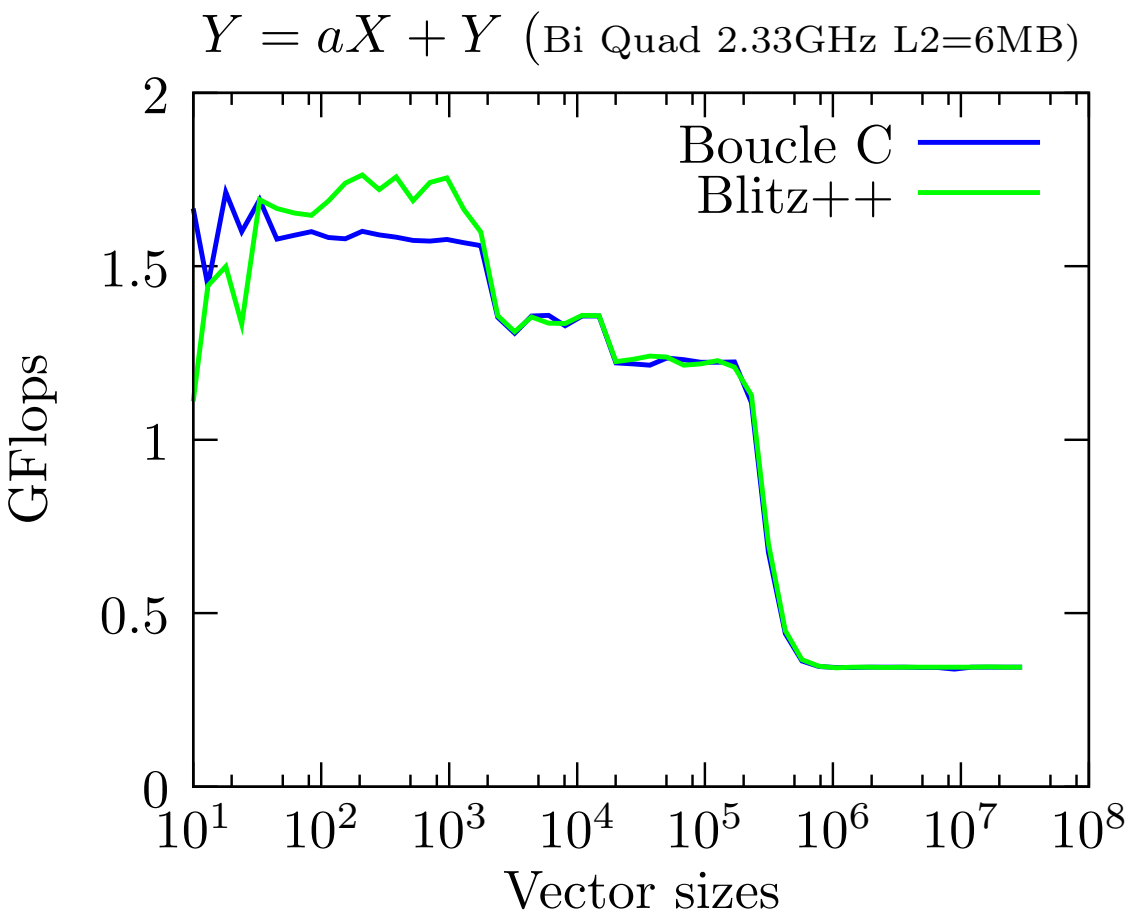
Exemple introductif (daxpy)



Implémentation C

```
for (int i=0;i<N;i++){
  Y[i]+=coef*X[i];
}
```

Exemple introductif (daxpy)



Implémentation C

```
for (int i=0;i<N;i++){
  Y[i]+=coef*X[i];
}
```

Implémentation Blitz++

```
Y+=coef*X;
```

Les 3 ingrédients principaux

- Noyaux de calcul :

$$\text{BTL++ kernels } = \{\text{dot}, \text{MatMat prod}, X = aY + bZ + cW \ldots\}$$

Les 3 ingrédients principaux

- Noyaux de calcul :

  $$\text{BTL++ kernels} = \{\text{dot}, \text{MatMat prod}, X = aY + bZ + cW \ldots\}$$

- Bibliothèques/Implémentations :

  $$\text{BTL++ libraries} = \{\text{uBlas}, \text{MTL4}, \text{ATLAS}, \text{Goto}, \text{C}, \text{F77}, \ldots\}$$

Les 3 ingrédients principaux

- Noyaux de calcul :

  $$\text{BTL++ kernels} = \{\text{dot}, \text{MatMat prod}, X = aY + bZ + cW \ldots\}$$

- Bibliothèques/Implémentations :

  $$\text{BTL++ libraries} = \{\text{uBlas}, \text{MTL4}, \text{ATLAS}, \text{Goto}, \text{C}, \text{F77}, \ldots\}$$

- Timer et méthodes de mesure (`rdtsc`, `gettimeofday`, `PAPI`,...)

Les 3 *concepts* associés

- Noyaux de calcul
  → le concept BTL++ `Action`.

Les 3 *concepts* associés

- Noyaux de calcul
  → le concept BTL++ `Action`.
- Bibliothèques/Inplémentations
  → le concept BTL++ `Library_Interface`.

Les 3 *concepts* associés

- Noyaux de calcul
  → le concept BTL++ `Action`.
- Bibliothèques/Inplémentations
  → le concept BTL++ `Library_Interface`.
- Méthodes de mesures
  → le concept BTL++ `Performance_Analyser`.

Les 3 *concepts* associés

- Noyaux de calcul
  $\longrightarrow$ le concept BTL++ `Action`.
- Bibliothèques/Inplémentations
  $\longrightarrow$ le concept BTL++ `Library_Interface`.
- Méthodes de mesures
  $\longrightarrow$ le concept BTL++ `Performance_Analyser`.
- Exemple : évaluation de la performance de l'opération `axpy` ($Y = a * X + Y$) implémentée par la bibliothèque Blitz++.

BTL++ Library_Interface

public Types	
RT	Real Type (double or float)
GV	Generic Vector Type
GM	Generic Matrix Type
(static) functions	
std::string name(void)	
void vector_from_stl(GV &, const SV &)	
void vector_to_stl(const GV &,SV &)	
void matrix_from_stl(GM &, const SM &)	
void matrix_to_stl(const GM &, SM &)	
void axpy(RT, const GV &, GV &, int)	
+ dot, copy matrix_vector_product, ...	

blitz_interface *instance* de Library_Interface

```
template<class real>
struct blitz_interface{
  typedef real                    RT;
  typedef blitz::Vector<RT>       GV;
  static std::string name( void ){return "Blitz";}
  static void vector_from_stl(GV & B, const std::vector<RT
      > & B_stl){
    B.resize(B_stl.size()); // Note the () operator
    for (int i=0; i<B_stl.size() ; i++) B(i)=B_stl[i];
  }
  static void vector_to_stl(const GV & B, std::vector<RT>
      & B_stl){
    for (int i=0; i<B_stl.size() ; i++) B_stl[i]=B(i);
  }
  static void axpy(const RT coef, const GV & X, GV & Y,
      int N){
    Y+=coef*X; // Blitz++ Expression Template !
  }
  ...follows dot, matrix_vector_product,..
};
```

BTL++ Action concept

public Types	
`Interface`	*Library_Interface* model
methods	
`Ctor(int size)`	Ctor with problem size argument
`void initialize( void )`	
`void calculate( void )`	
`void check_result( void )`	
`double nb_op_base( void )`	
(static) functions	
`std::string name( void )`	

action_axpy instance de Action

```
template<class LIB_INTERFACE> class action_axpy {
public :
 typedef LIB_INTERFACE Interface;
 action_axpy(int size):size_(size)...{...} //Ctor
 static std::string name( void ){return "axpy";}
 double nb_op_base( void ){return 2.0*_size;}
 void calculate() {Interface::axpy(_coef,X,Y,_size);}
 void initialize(){Interface::copy_vector(Y_ref,Y,_size);}
 void check_result(){
        Interface::vector_to_stl(Y,resu_stl);
        STL_I::axpy(_coef,X_stl,Y_stl);
        assert(STL_I::norm_diff(Y_stl,resu_stl)<1.e-6);
 }
 ...
private :
 const int _size;
 typename STL_I::gene_vector  X_stl,Y_stl,resu_stl;
 typename LIB_INTERFACE::gene_vector  X,Y,Y_ref;
};
```

Exemples

```
action_axpy< blitz_interface<double> > a(1000);
a.calculate(); // Performs axpy
a.initialize();// Reset X and Y operand
a.calculate(); // Performs axpy (again)
a.check_result(); // Compare with STL
//
action_dot< ATLAS_interface<float> > b(1000);
b.calculate(); // Performs dot
b.initialize();// Reset X and Y operand
b.calculate(); // Performs dot (again)
b.check_result(); // Compare with STL

bench<portable_perf_analyzer,
      action_dot< ATLAS_interface<float> > >(sizes);
```

Plan

Formation

- Importance de la localité des données.

Formation

- Importance de la localité des données.
- Choix d'une bibliothèque d'implémentation d'un noyau.

Formation

- Importance de la localité des données.
- Choix d'une bibliothèque d'implémentation d'un noyau.
- Usage d'une bibliothèque.

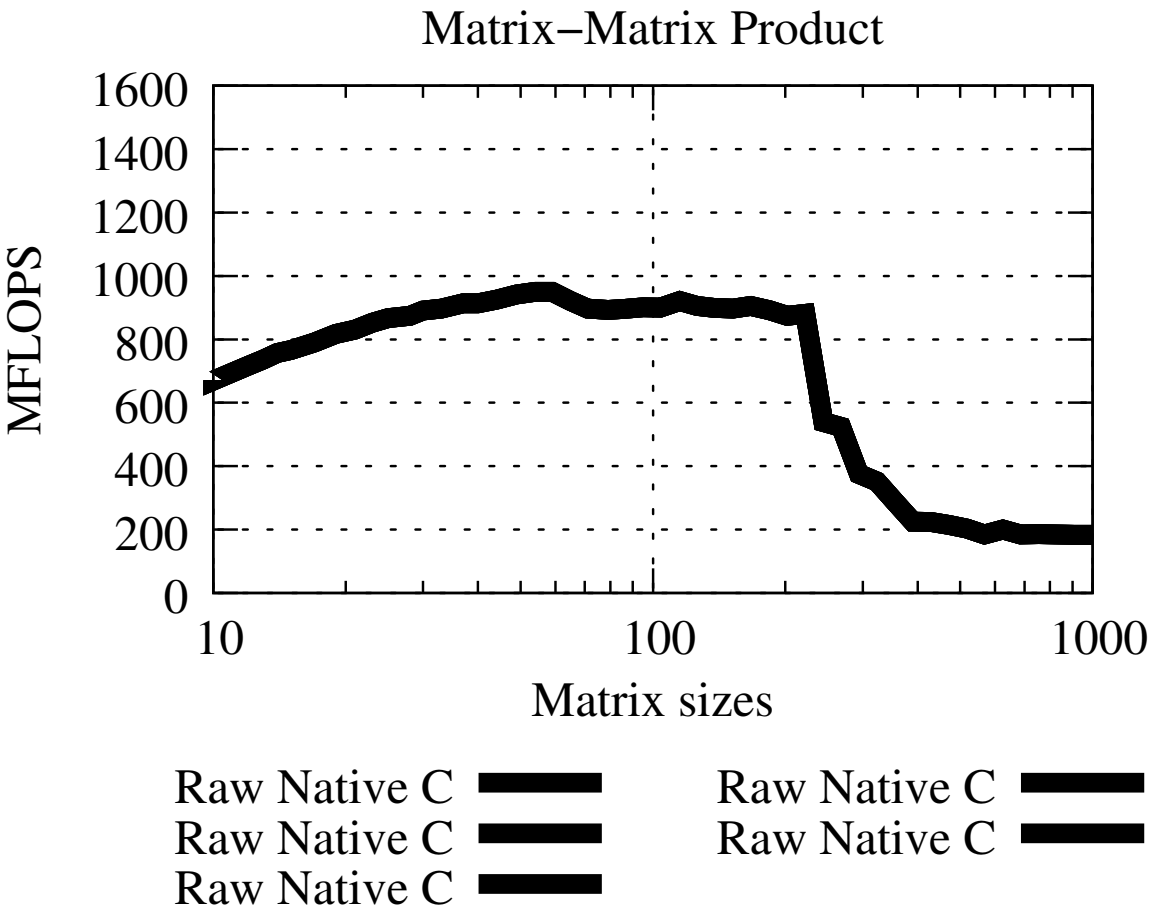
Formation

- Importance de la localité des données.
- Choix d'une bibliothèque d'implémentation d'un noyau.
- Usage d'une bibliothèque.
- Choix des machines cibles.

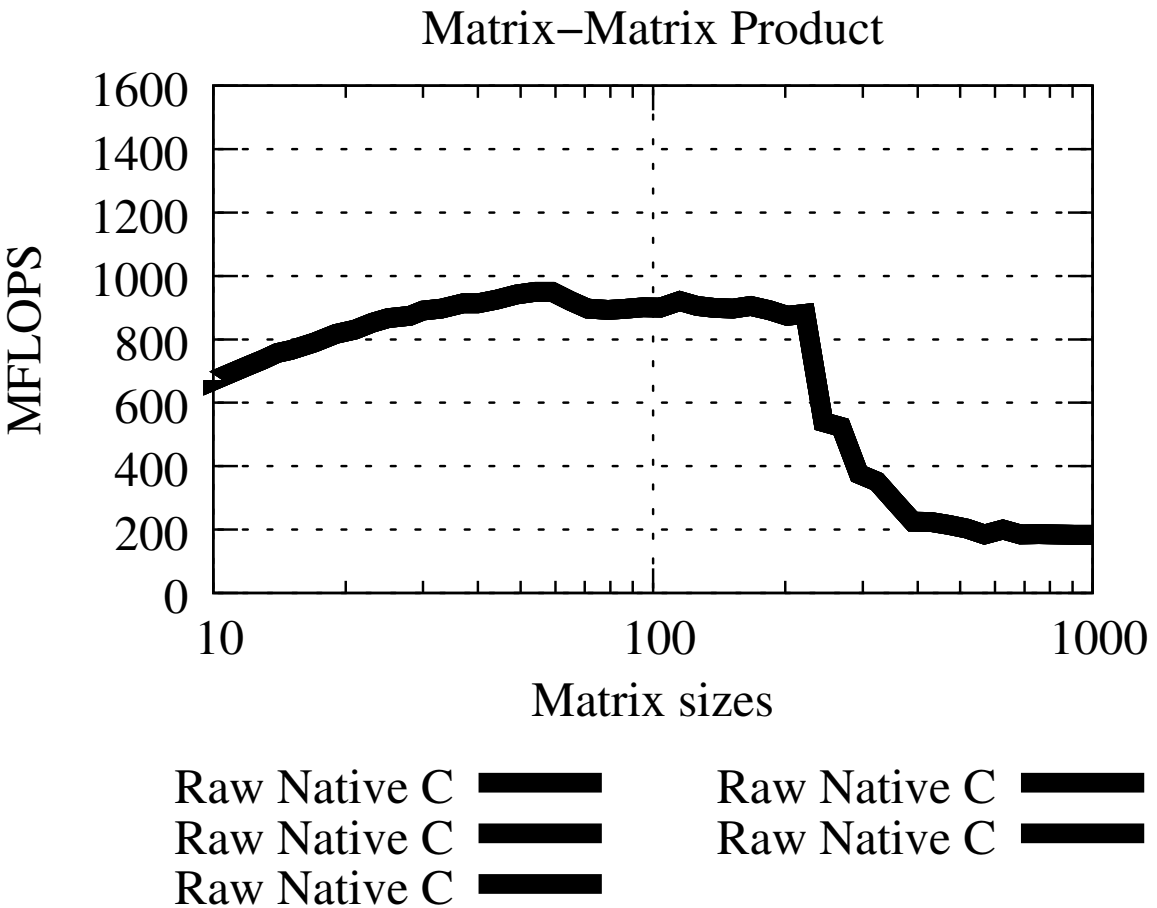
Formation

- Importance de la localité des données.
- Choix d'une bibliothèque d'implémentation d'un noyau.
- Usage d'une bibliothèque.
- Choix des machines cibles.
- Exemple : Produit $X = A * B$ où X, A et B sont des matrices denses de `double` .

Boucle C

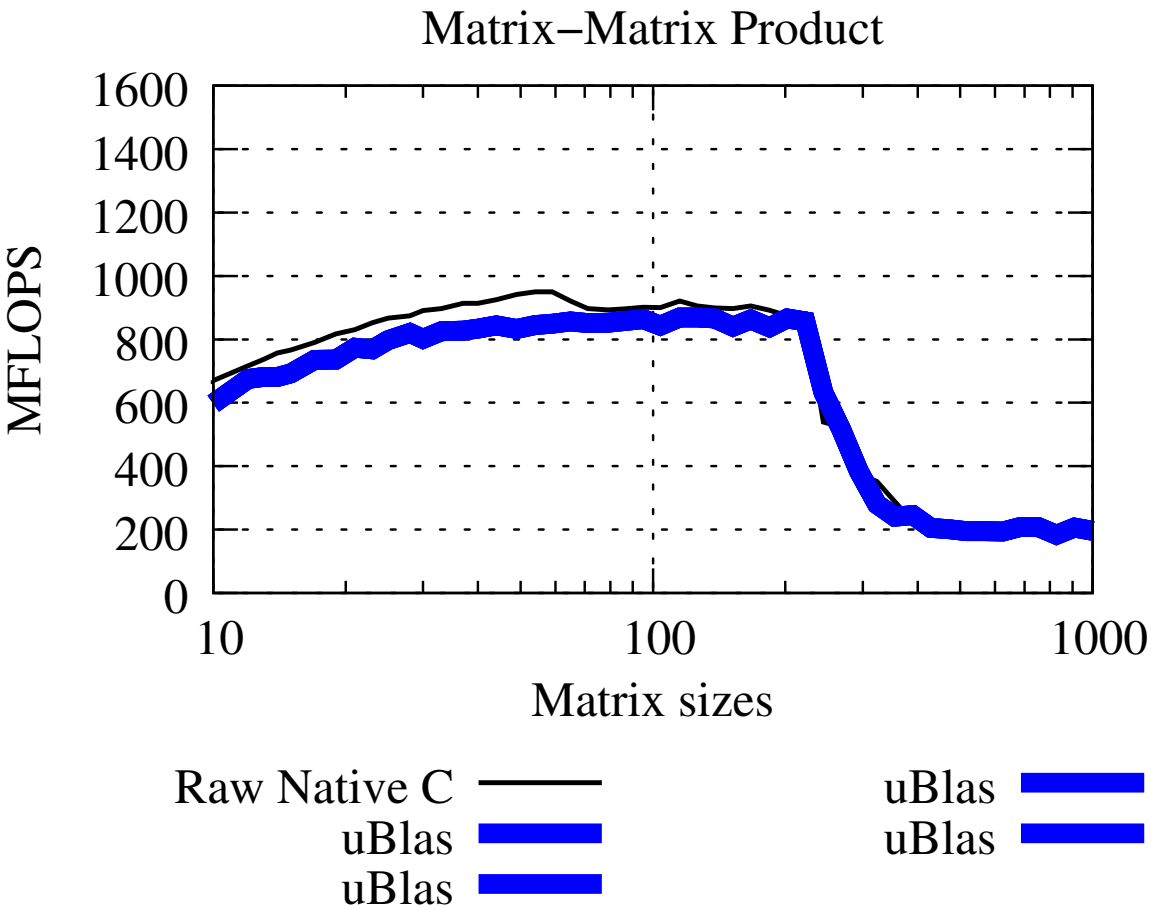

Matrix–Matrix Product
MFLOPS
1600
1400
1200
1000
800
600
400
200
0
10
100
1000
Matrix sizes
Raw Native C
Raw Native C
Raw Native C
Raw Native C
Raw Native C

Boucle C

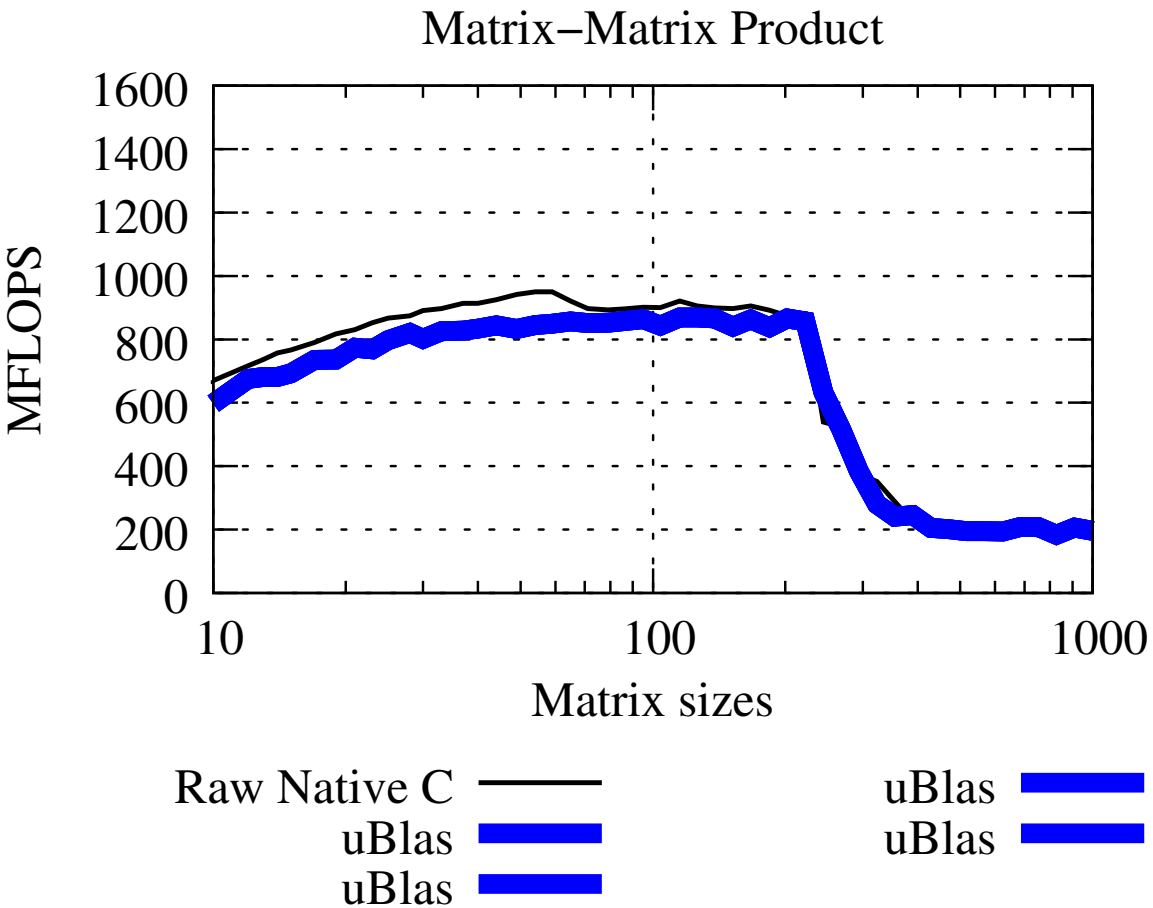



```
double sum;
for (int i=0;i<N;i++){
  for (int j=0;j<N;j++){
    sum=0.0;
    for (int k=0;k<N;k++){
      sum+=A[i][k]*B[k][j];
    }
    X[i][j]=sum;
  }
}
```

C++ Generic Library implementation : uBlas

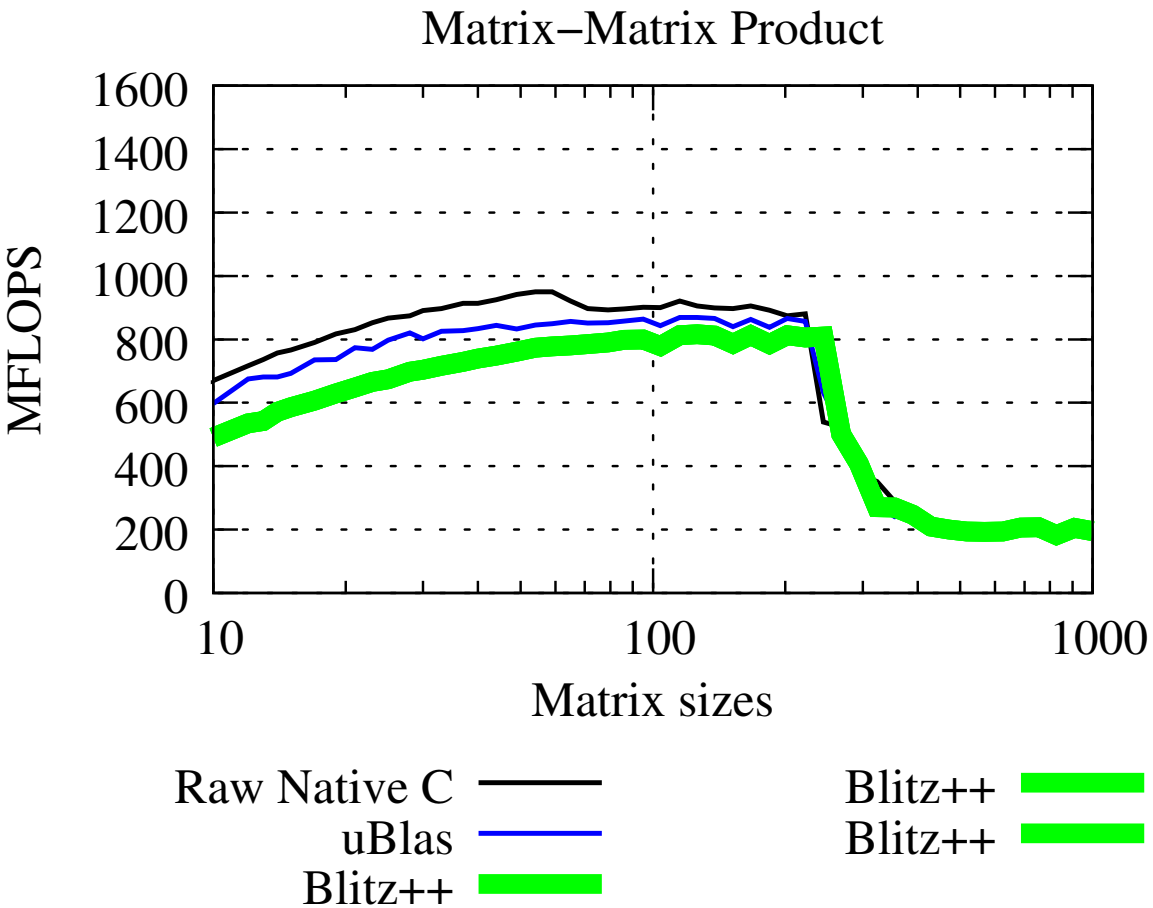


C++ Generic Library implementation : uBlas

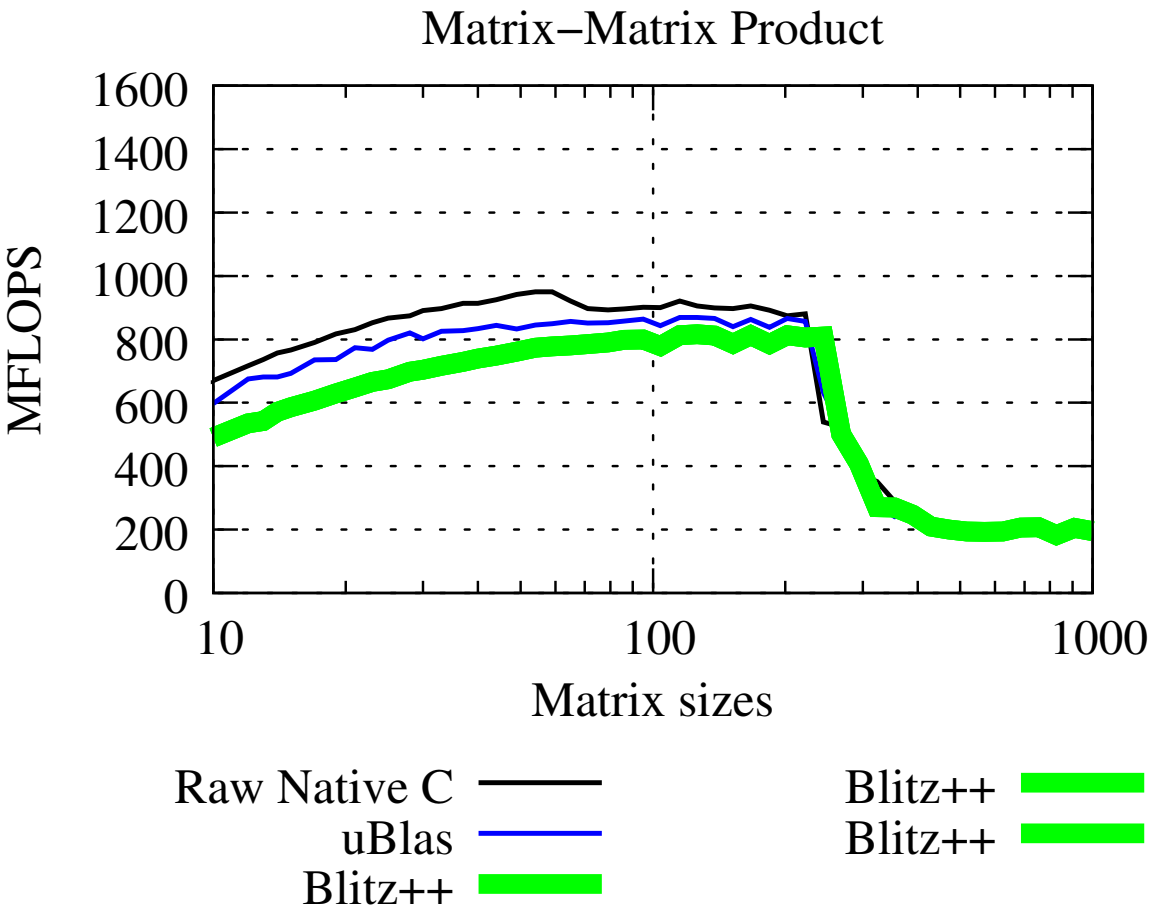



```
X.assign(prod(A,B));
//faster than X=prod(A,B);
```

C++ Generic Library implementation : Blitz++

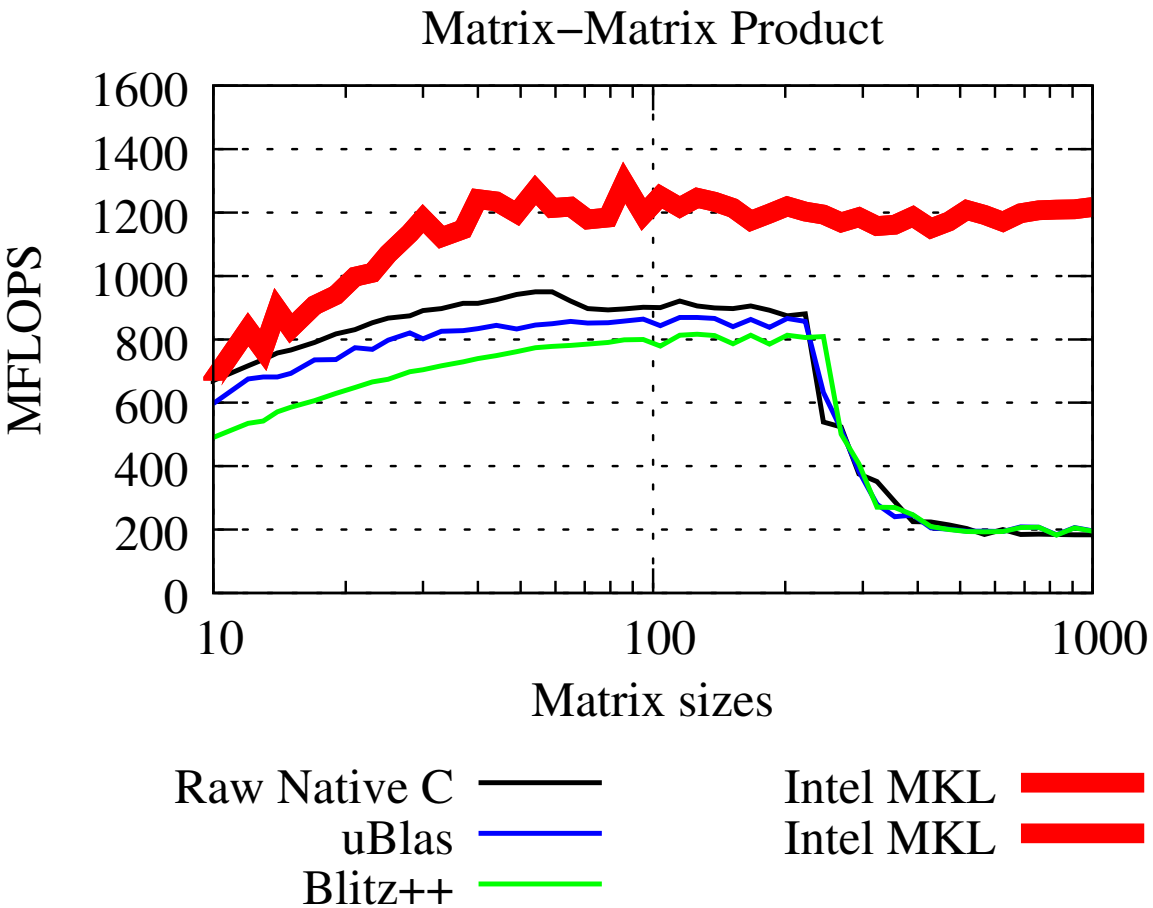


C++ Generic Library implementation : Blitz++

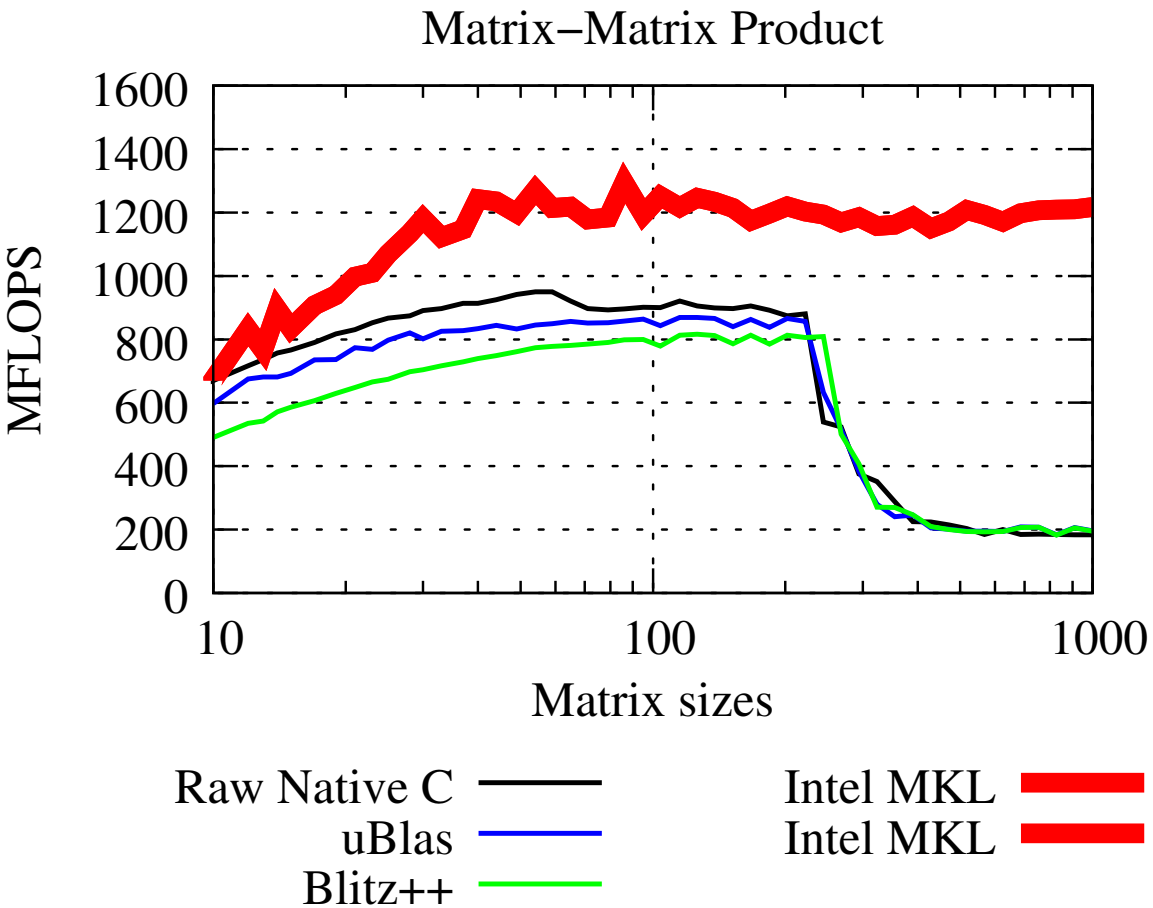



```
real sum;
for (int i=0;i<N;i++){
  for (int j=0;j<N;j++){
    sum=0.0;
    for (int k=0;k<N;k++){
      sum+=A(i,k)*B(k,j);
    }
    X(i,j)=sum;
  }
}
//X=sum(A(i,j)*B(j),j);
```

BLAS API Implementation : MKL

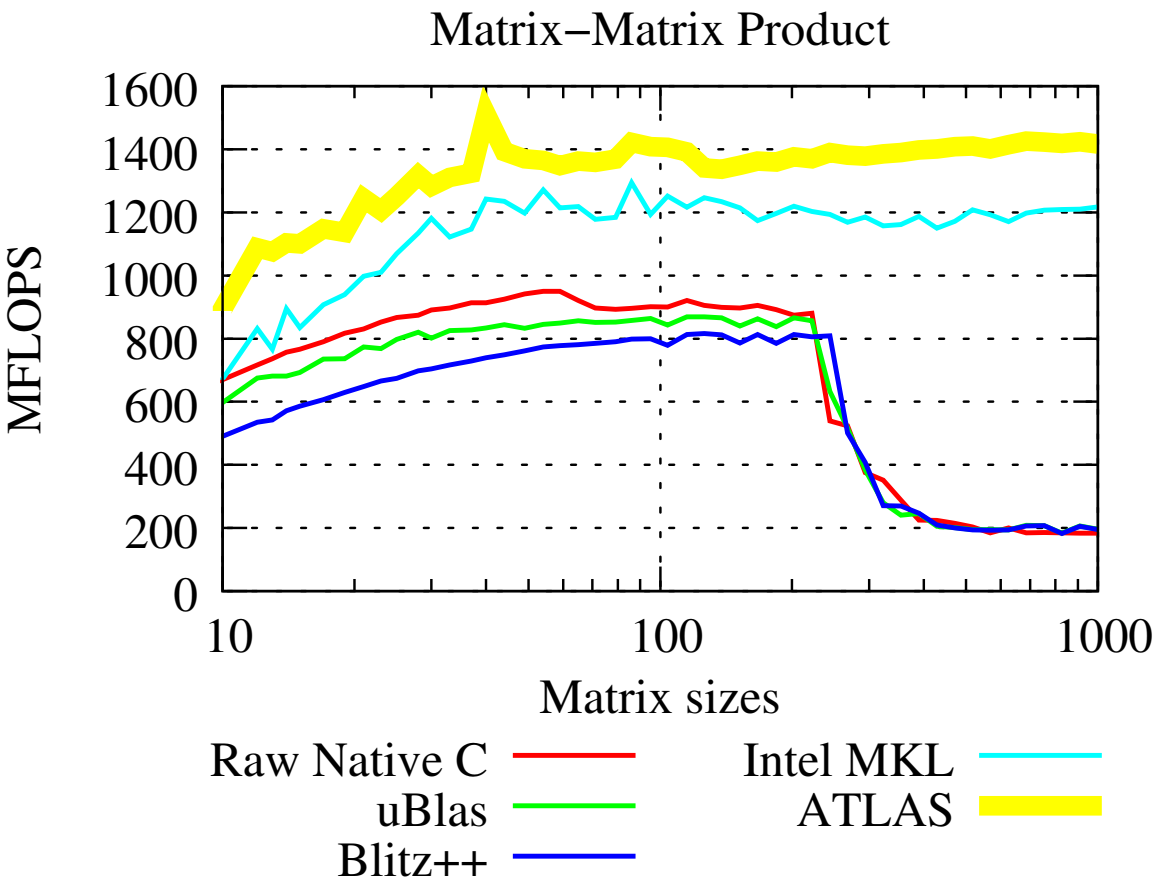


BLAS API Implementation : MKL



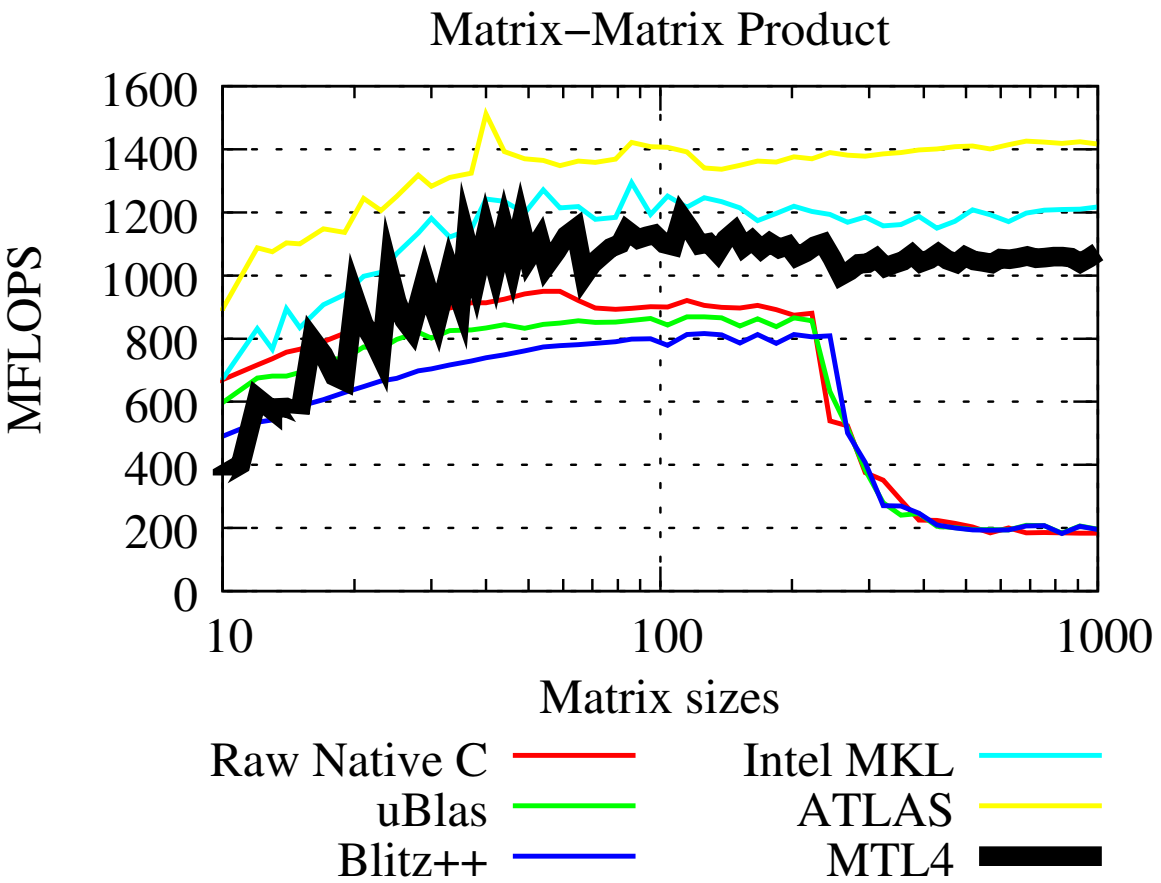

```
cblas_dgemm(CblasColMajor,
            CblasNoTrans,
            CblasNoTrans,
            N,N,N,1.0,A,N,
            B,N,0.0,X,N);
```

BLAS API Implementation : ATLAS



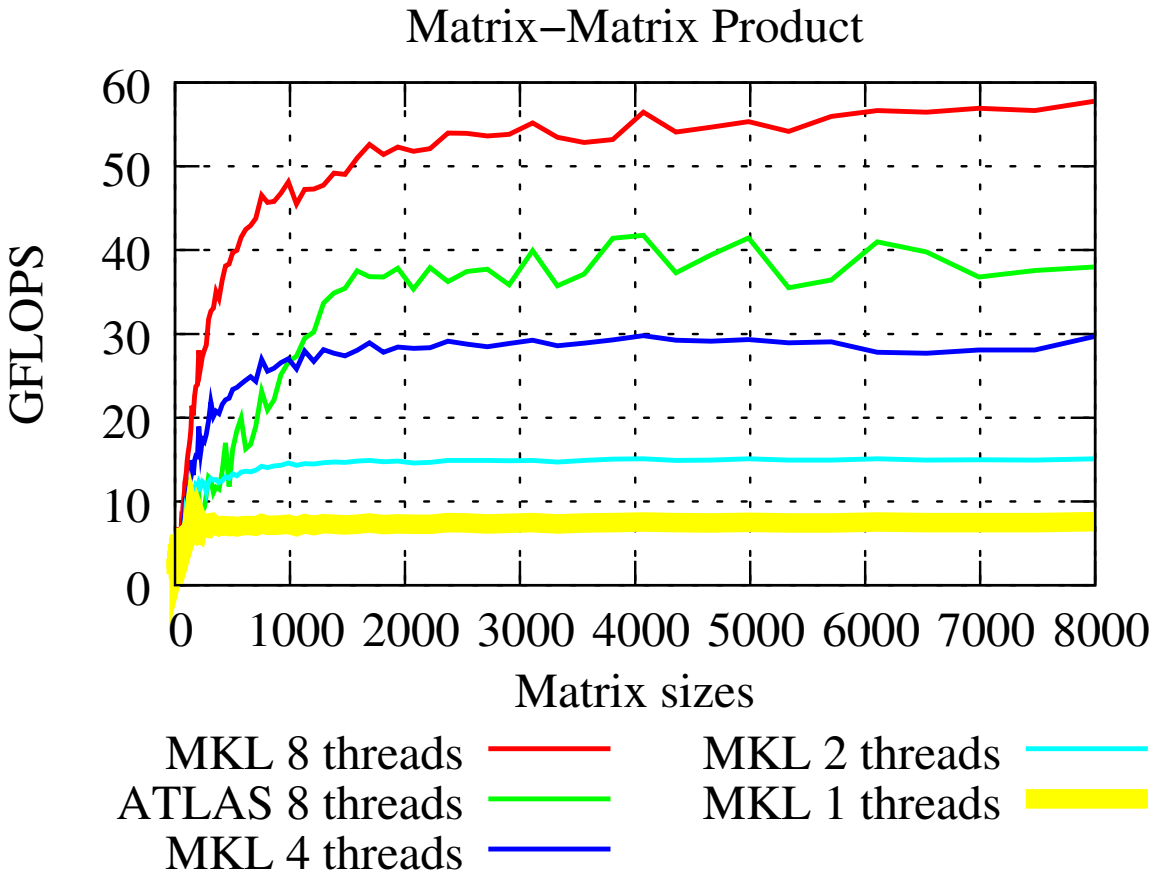

```
cblas_dgemm(CblasColMajor,
            CblasNoTrans,
            CblasNoTrans,
            N,N,N,1.0,A,N,
            B,N,0.0,X,N);
```

MTL4 (Morton Order)



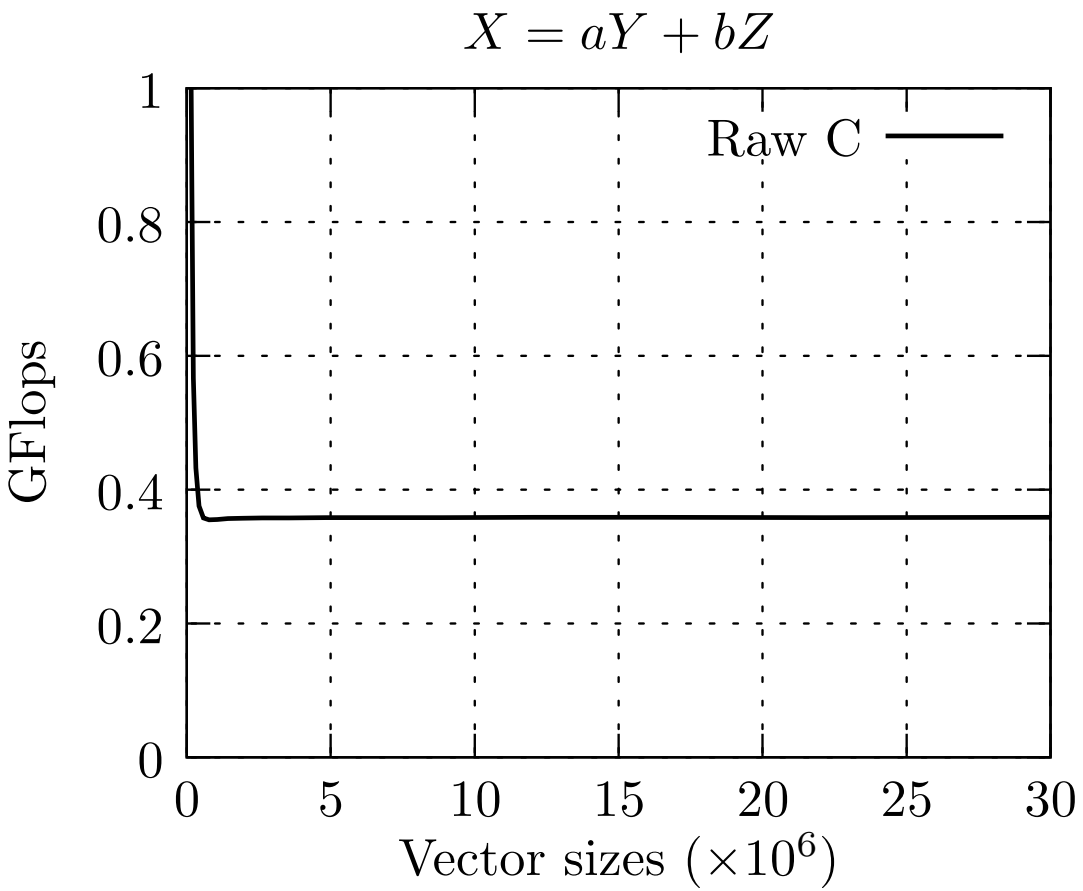

```
X=A*B;
```

Multithreaded dgemm dual processor/quad core Intel Xeon with 2.33GHz with Intel MKL version 10 and ATLAS version 3.8.0



Plan

Différentes implémentations de X=a*Y+b*Z

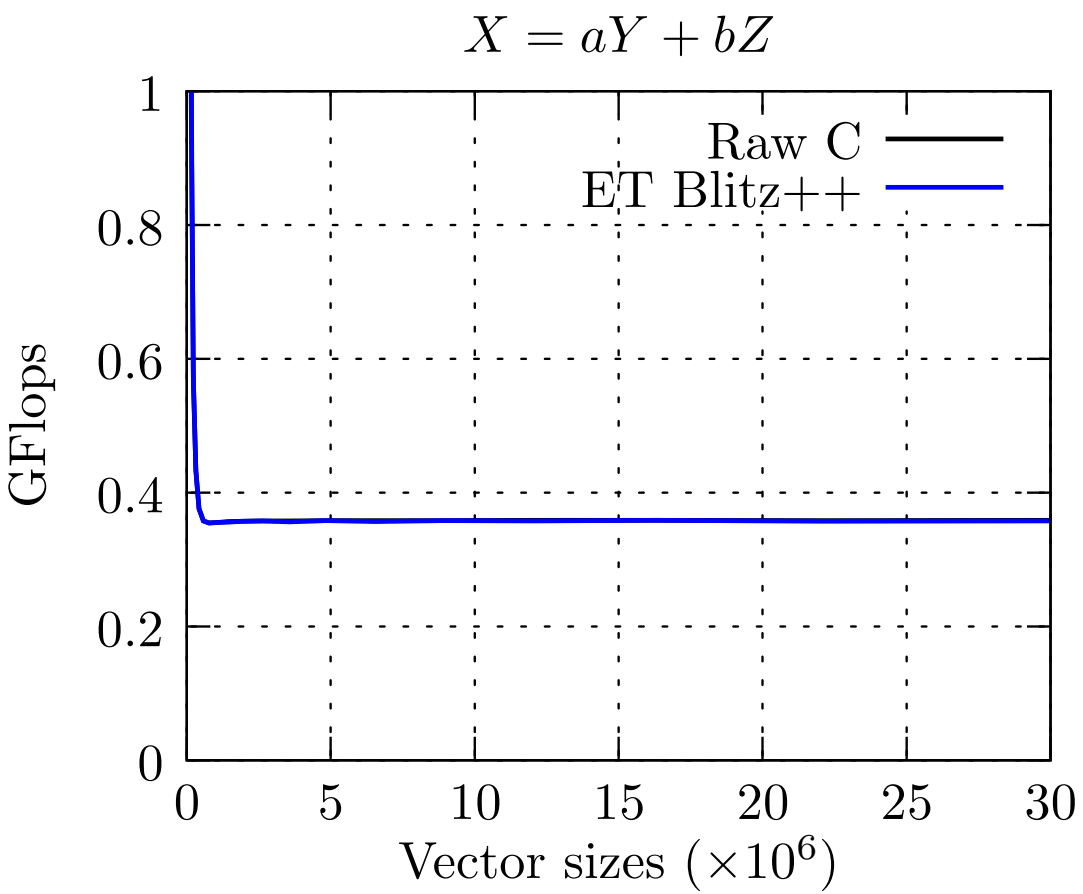


Intel Bi-Quad Core 2.33GHz

Boucle C

```
for (int i=0;i<N;++i){
  X[i]=a*Y[i]+b*Z[i];
}
```

Blitz++ Expression Template

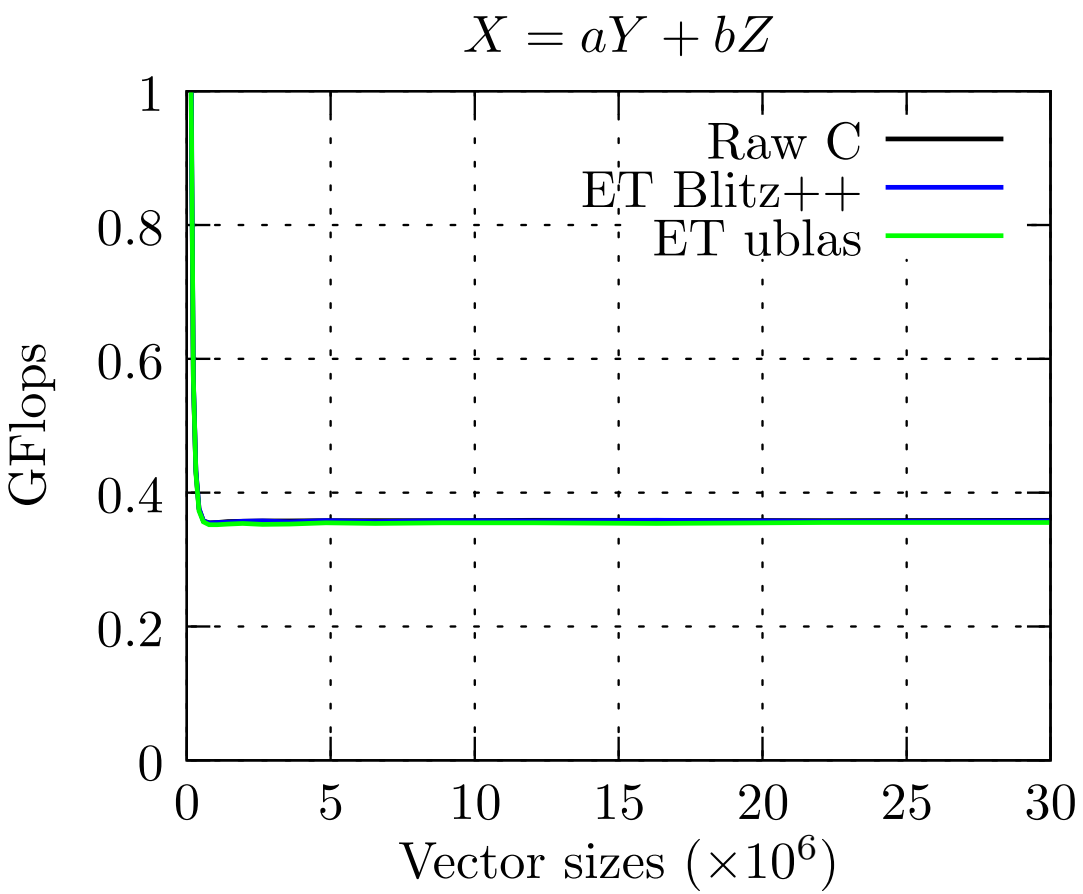


Intel Bi-Quad Core 2.33GHz

Blitz++

```
X=a*Y+b*Z;
```

uBLAS

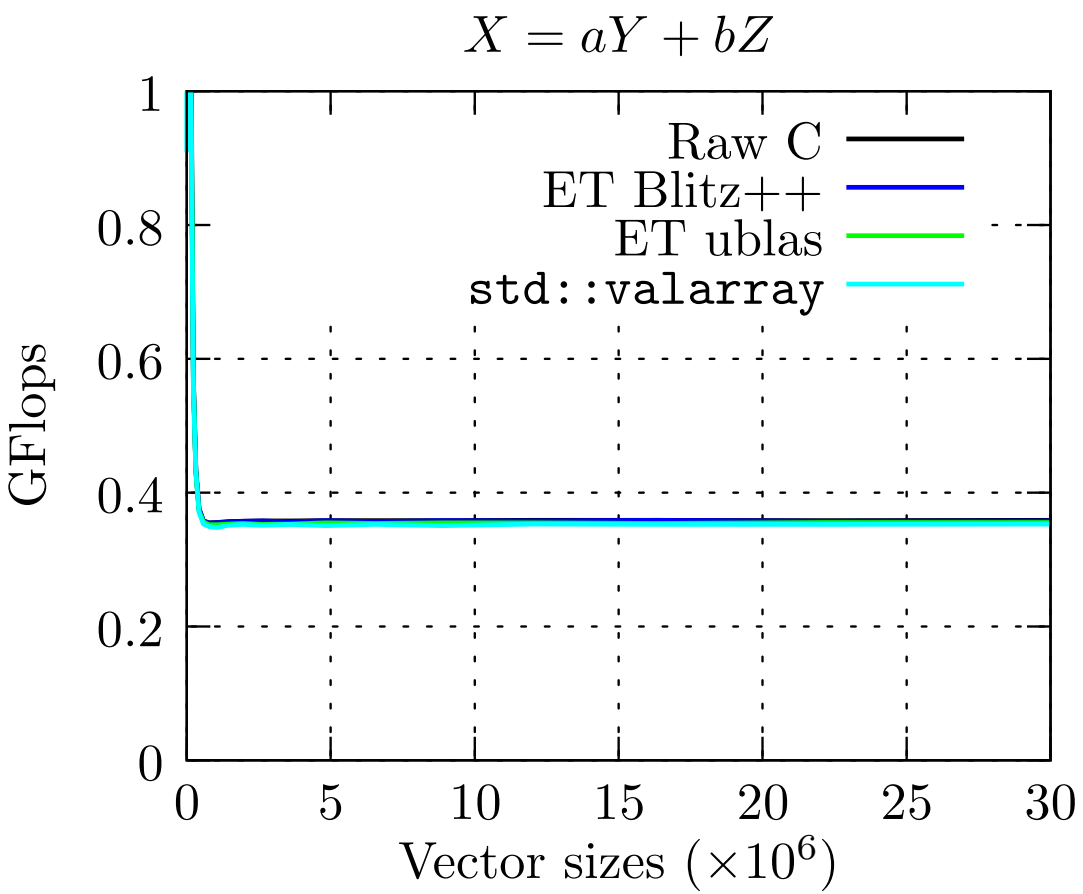


Intel Bi-Quad Core 2.33GHz

uBLAS

```
X.assign(a*Y+b*Z);
```

std::valarray

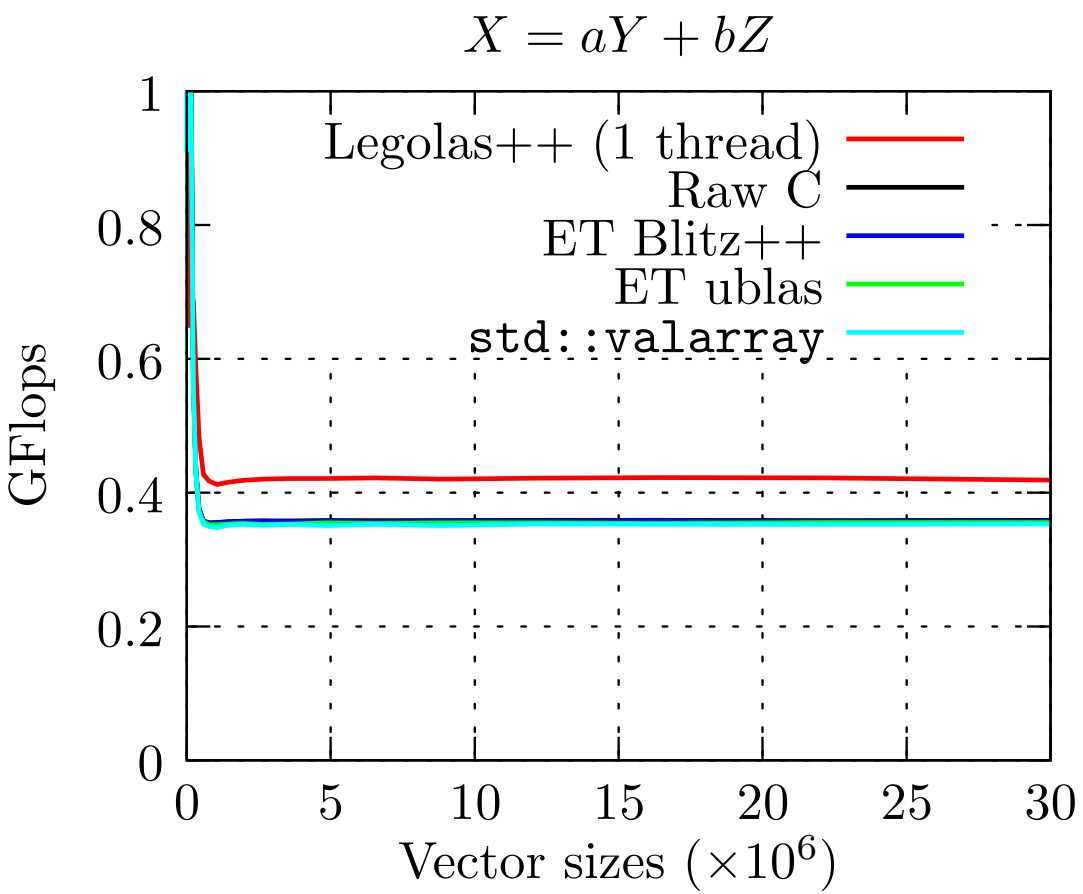


Intel Bi-Quad Core 2.33GHz

std::valarray

```
X=a*Y+b*Z;
```

Legolas++ (movntq)



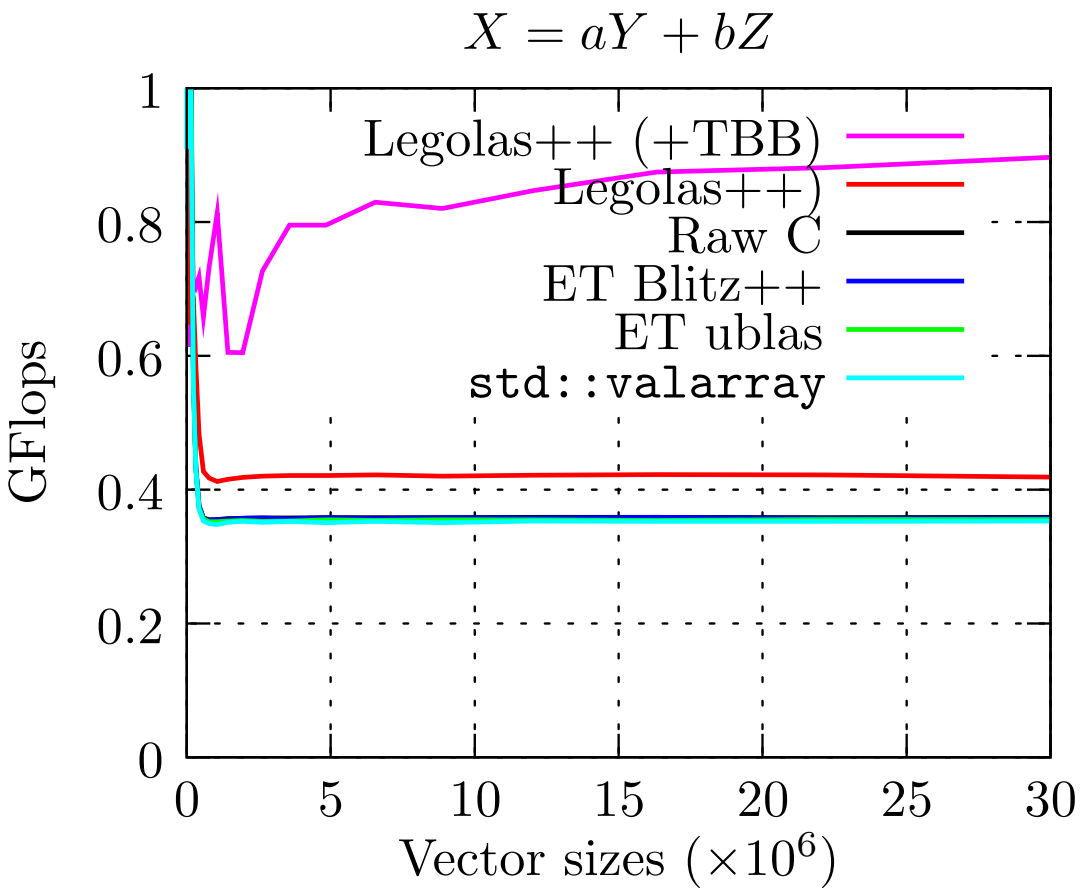
Intel Bi-Quad Core 2.33GHz

Legolas++ (movntq)

```
X=a*Y+b*Z;
```

Plagne and Hülsemann (MPOOL07)

Legolas++ (TBB)



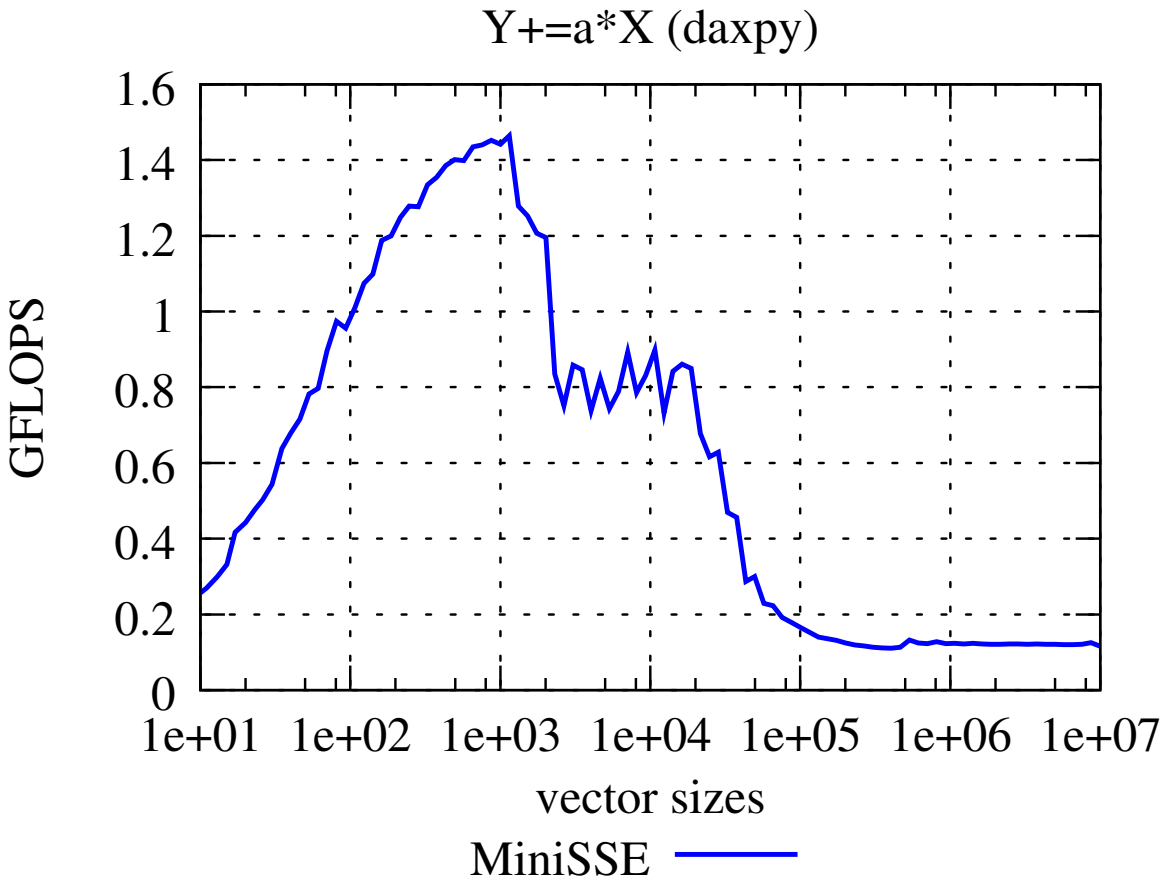
Intel Bi-Quad Core 2.33GHz

Legolas++ (TBB)

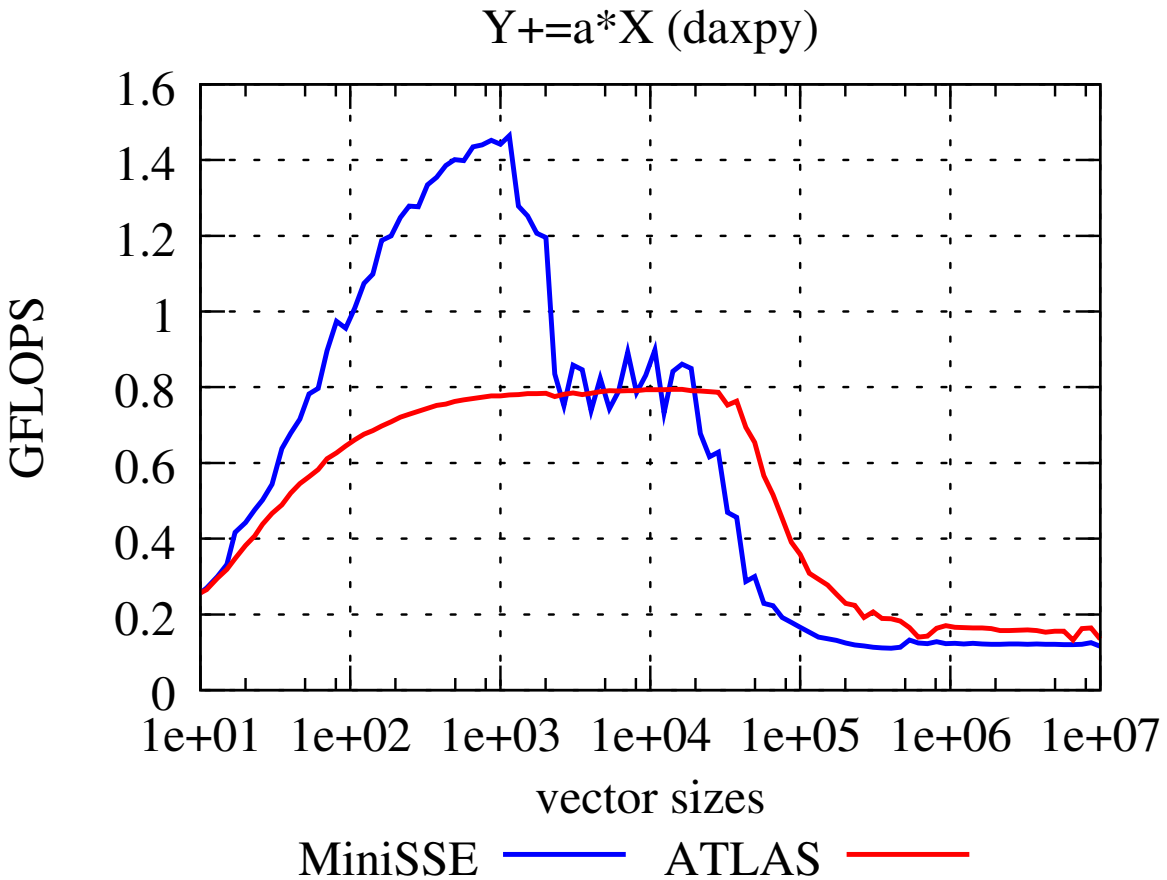
```
X=a*Y+b*Z;
```

Plan

Synthèse automatique de bibliothèques optimales



Synthèse automatique de bibliothèques optimales



Synthèse automatique de bibliothèques optimales

Trouver L_i,L_j et t qui maximisent

$$S(L_i, L_j, t) = \sum_{s \leq t} \text{perf}(\texttt{f}, L_1, s) + \sum_{s > t} \text{perf}(\texttt{f}, L_2, s) \ . \qquad (1)$$

- $\{L_i\}$: l'ensemble des bibliothèques qui implémentent une subroutine f donnée (API BLAS).
- $\text{perf}(\texttt{f}, L_i, s)$: la performance mesurée pour une bibliothèque L_i et une taille s.
- t : la taille seuil.

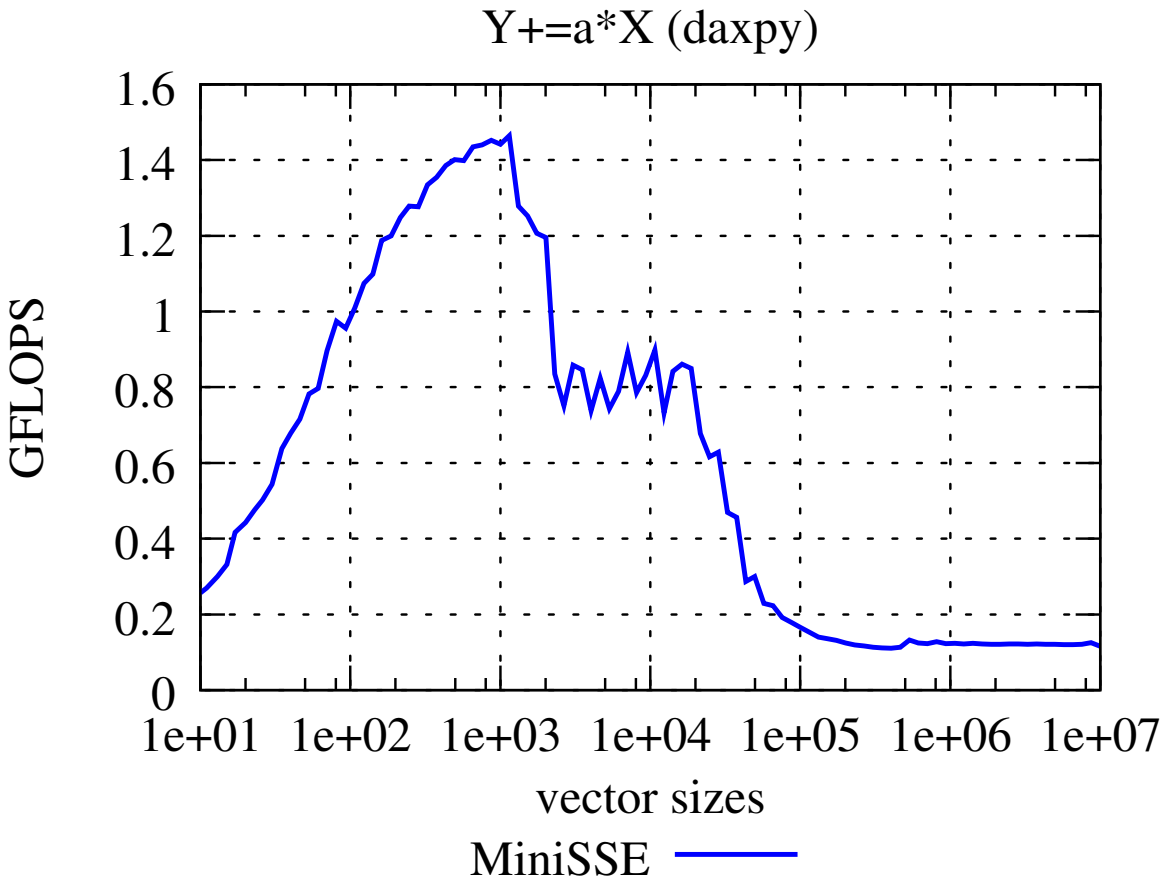
Table: Résultats (Pentium M 1.6 GHz)

BLAS routine	Best Library for small problem sizes	Small/large threshold	Best Library for large problem sizes
daxpy	MiniSSE	18738	ATLAS
dcopy	Netlib	86974	ATLAS
ddot	MiniSSE	18738	ATLAS
dgemm	ATLAS		ATLAS
dgemv	MKL		MKL

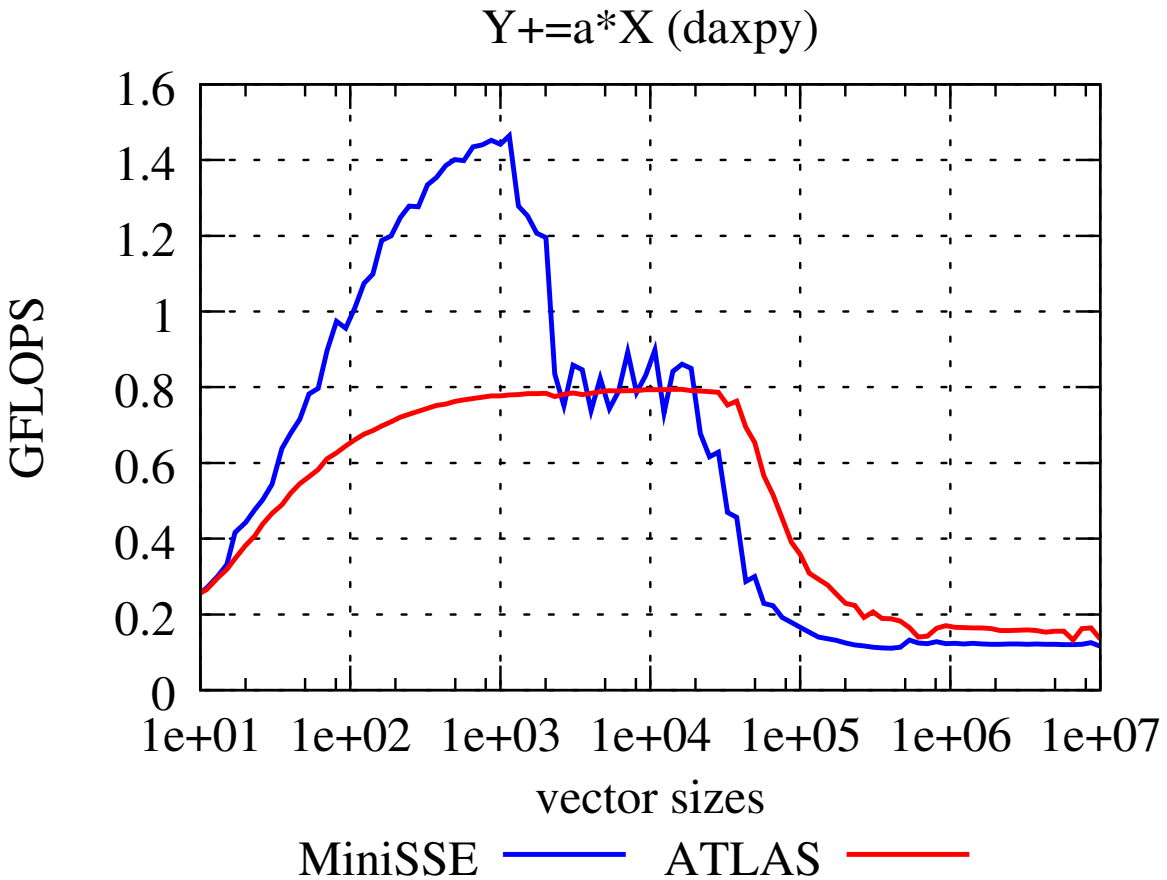
Synthèse automatique de bibliothèques optimales : daxpy

```
void cblas_daxpy(const int N, const double alpha,
                 const double *X, const int incX,
                 double *Y, const int incY){
  if (N<18738){
    MiniSSE1_cblas_daxpy(N,alpha,X,incX,Y,incY);
  }
  else{
    ATLAS_cblas_daxpy(N,alpha,X,incX,Y,incY);
  }
}
```

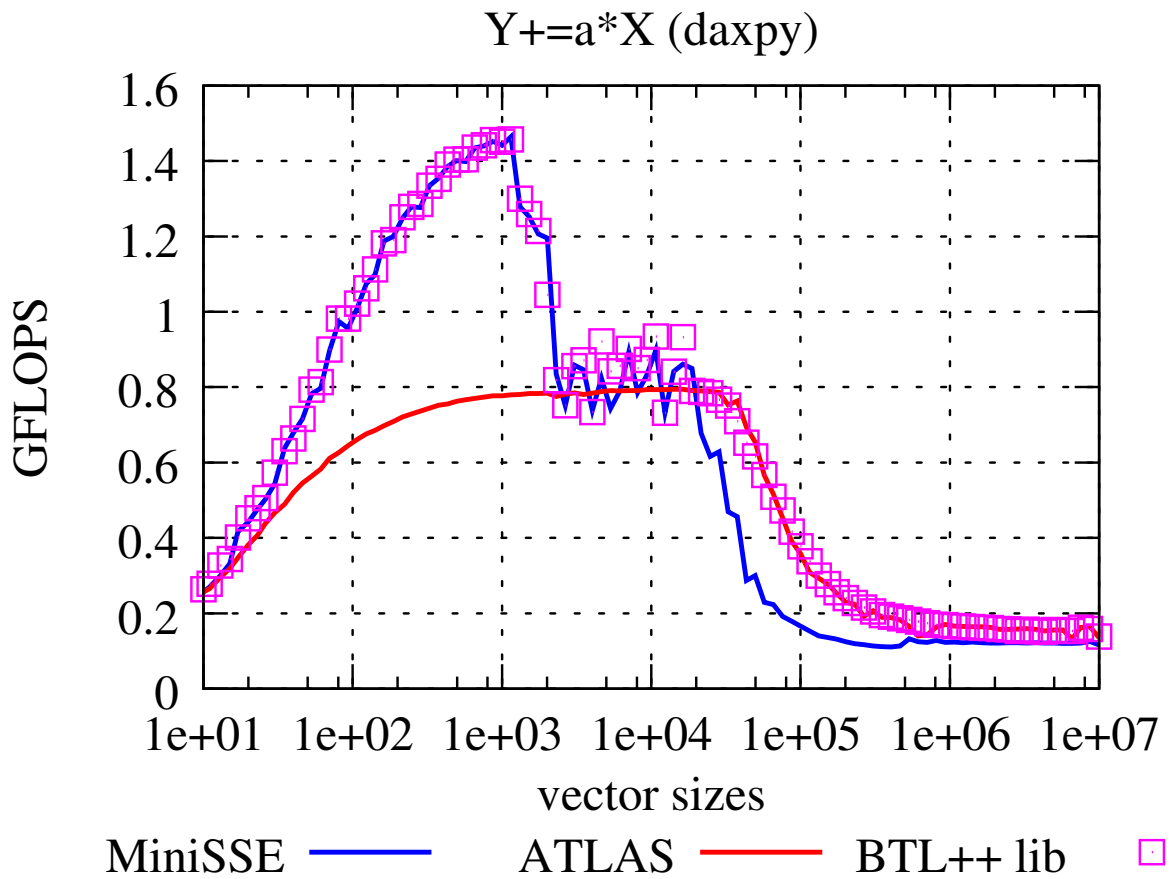
Synthèse automatique de bibliothèques optimales



Synthèse automatique de bibliothèques optimales

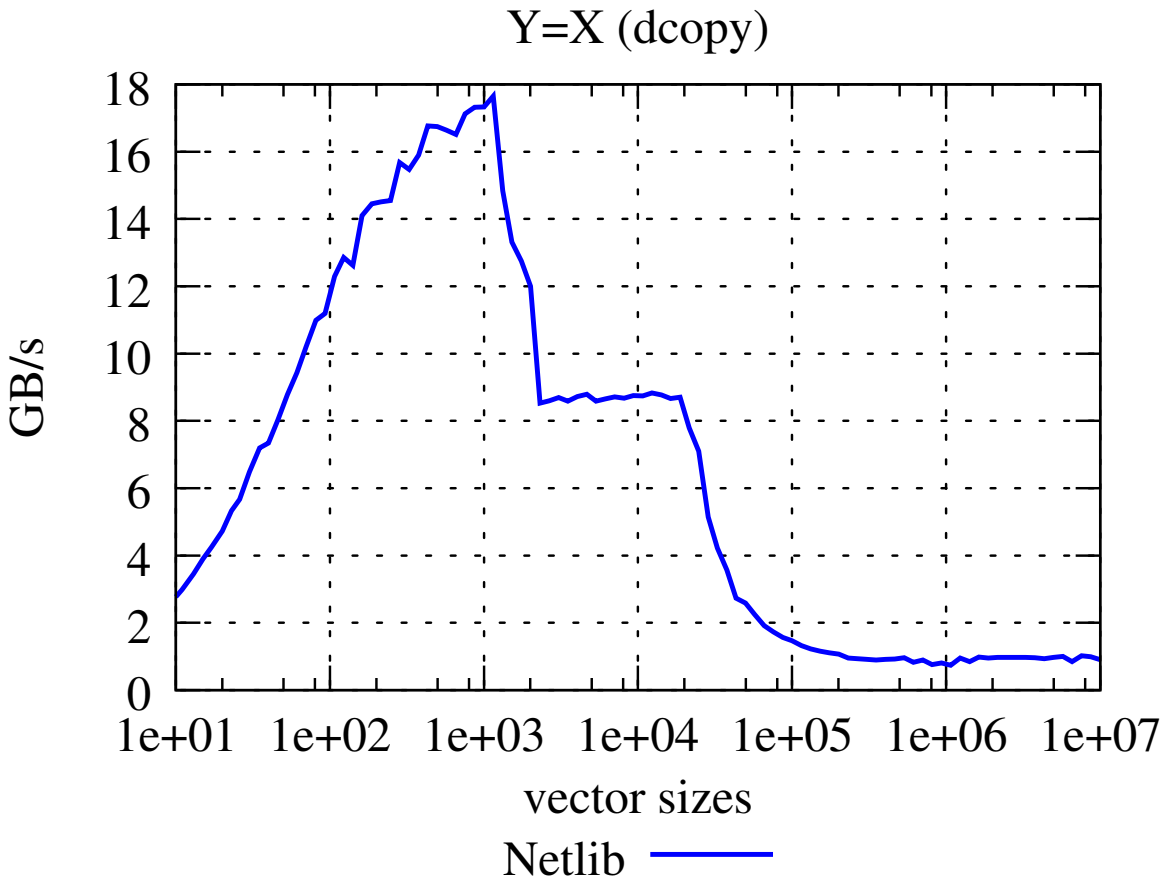


Synthèse automatique de bibliothèques optimales

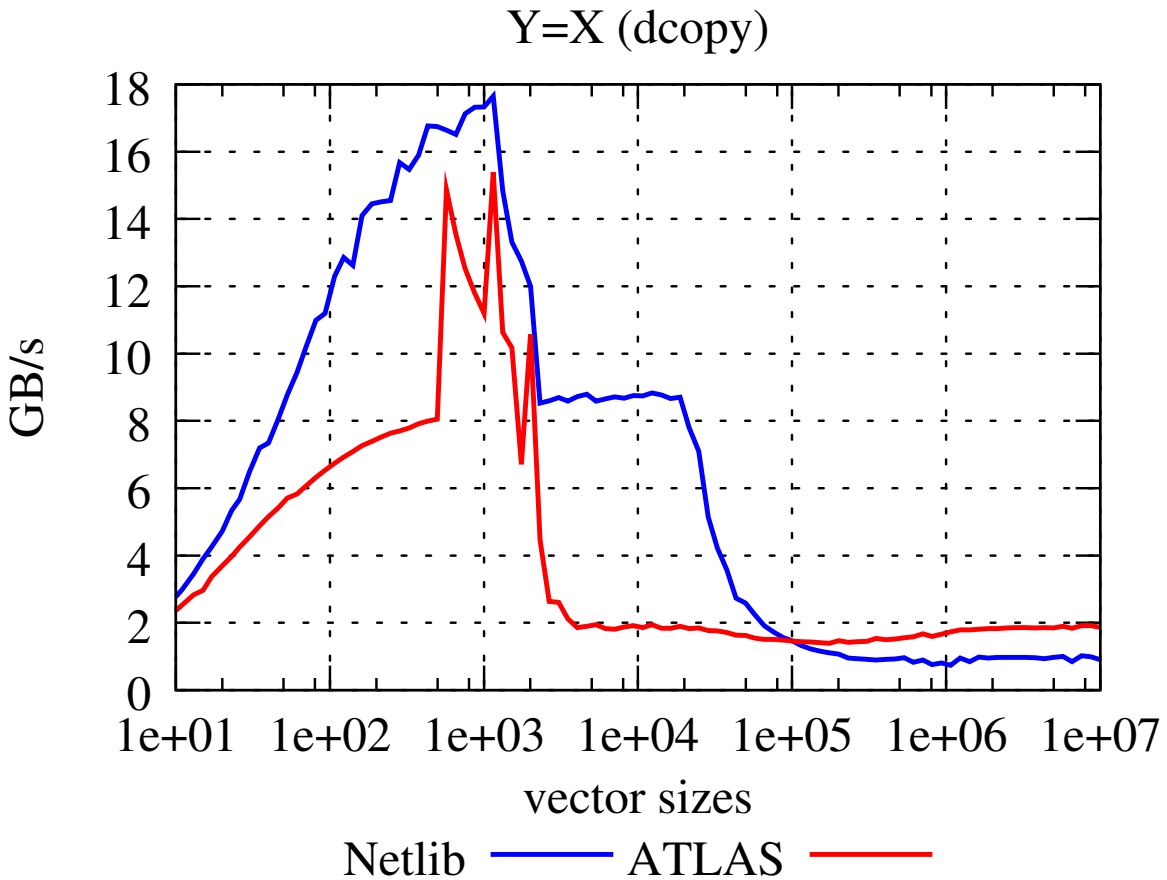


Plagne and Hülsemann (ICCS2008)

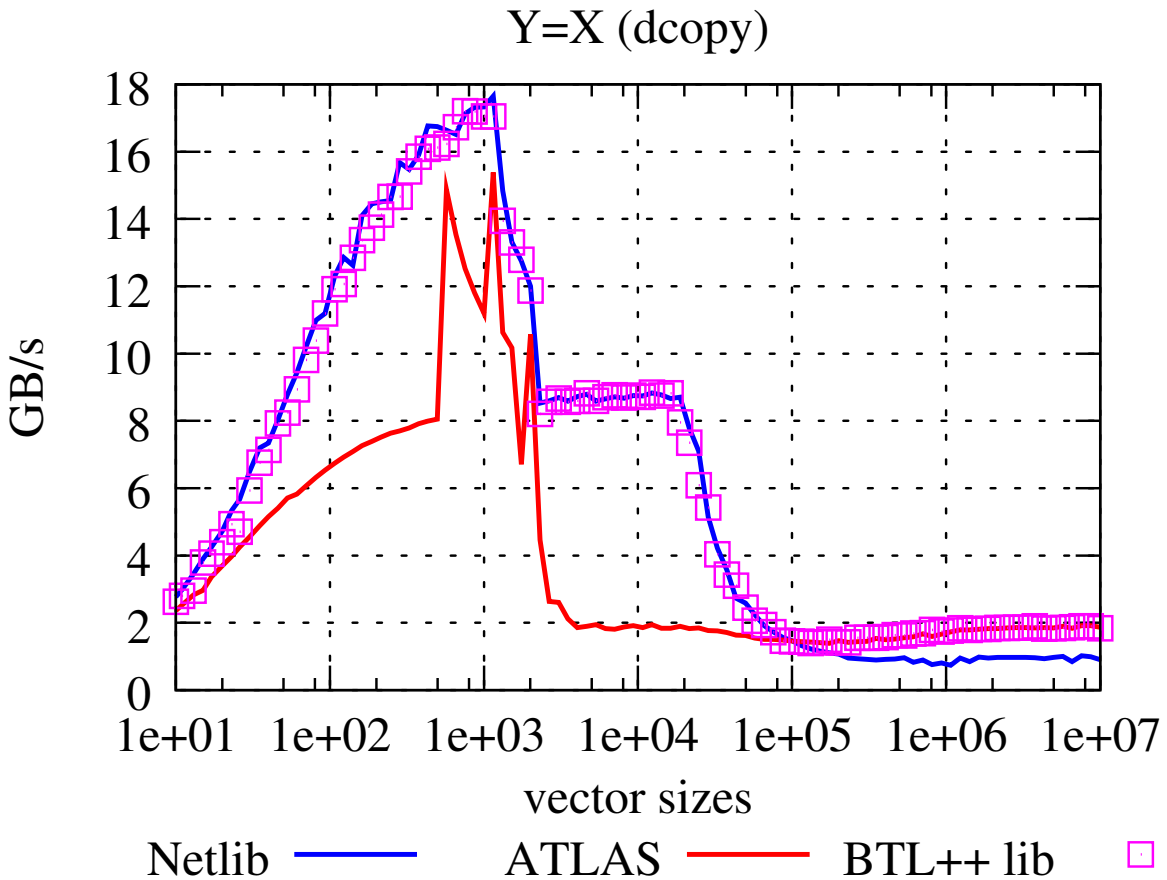
Synthèse automatique de bibliothèques optimales



Synthèse automatique de bibliothèques optimales



Synthèse automatique de bibliothèques optimales



Résumé et perspectives

Résumé et perspectives

1. Résumé
   - Extensibilité par l'utilisateur (*kernel* et *libraries*)

Résumé et perspectives

1. Résumé
   - Extensibilité par l'utilisateur (*kernel* et *libraries*)
   - Base de connaissance ouverte.

Résumé et perspectives

1. Résumé
   - Extensibilité par l'utilisateur (*kernel* et *libraries*)
   - Base de connaissance ouverte.
   - Utilisation quotidienne (formation et développements).

Résumé et perspectives

1. Résumé
   - Extensibilité par l'utilisateur (*kernel* et *libraries*)
   - Base de connaissance ouverte.
   - Utilisation quotidienne (formation et développements).
   - Synthèse de bibliothèques optimales.

Résumé et perspectives

1. Résumé
   - Extensibilité par l'utilisateur (*kernel* et *libraries*)
   - Base de connaissance ouverte.
   - Utilisation quotidienne (formation et développements).
   - Synthèse de bibliothèques optimales.

Résumé et perspectives

1. Résumé
   - Extensibilité par l'utilisateur (*kernel* et *libraries*)
   - Base de connaissance ouverte.
   - Utilisation quotidienne (formation et développements).
   - Synthèse de bibliothèques optimales.
2. Perspectives

Résumé et perspectives

1. Résumé
   - Extensibilité par l'utilisateur (*kernel* et *libraries*)
   - Base de connaissance ouverte.
   - Utilisation quotidienne (formation et développements).
   - Synthèse de bibliothèques optimales.
2. Perspectives
   - Toilettage et portabilité (auto-tools,. . .).

Résumé et perspectives

1. Résumé
   - Extensibilité par l'utilisateur (*kernel* et *libraries*)
   - Base de connaissance ouverte.
   - Utilisation quotidienne (formation et développements).
   - Synthèse de bibliothèques optimales.
2. Perspectives
   - Toilettage et portabilité (auto-tools,. . .).
   - Base de donnée et outil de navigation (data browser).

Résumé et perspectives

1. Résumé
   - Extensibilité par l'utilisateur (*kernel* et *libraries*)
   - Base de connaissance ouverte.
   - Utilisation quotidienne (formation et développements).
   - Synthèse de bibliothèques optimales.
2. Perspectives
   - Toilettage et portabilité (auto-tools,. . .).
   - Base de donnée et outil de navigation (data browser).
   - Matrices creuses.

Résumé et perspectives

1. Résumé
   - Extensibilité par l'utilisateur (*kernel* et *libraries*)
   - Base de connaissance ouverte.
   - Utilisation quotidienne (formation et développements).
   - Synthèse de bibliothèques optimales.
2. Perspectives
   - Toilettage et portabilité (auto-tools,. . .).
   - Base de donnée et outil de navigation (data browser).
   - Matrices creuses.
   - Calcul parallèle distribué.

Résumé et perspectives

1. Résumé
   - Extensibilité par l'utilisateur (*kernel* et *libraries*)
   - Base de connaissance ouverte.
   - Utilisation quotidienne (formation et développements).
   - Synthèse de bibliothèques optimales.
2. Perspectives
   - Toilettage et portabilité (auto-tools,. . .).
   - Base de donnée et outil de navigation (data browser).
   - Matrices creuses.
   - Calcul parallèle distribué.

- MERCI !!!

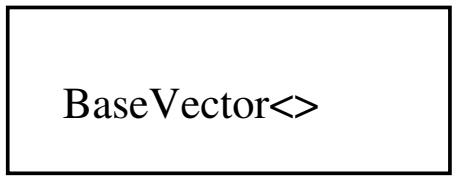
A Simple C++ Vector Class Implementation with Expression Template

Curiously Recurring Template Pattern

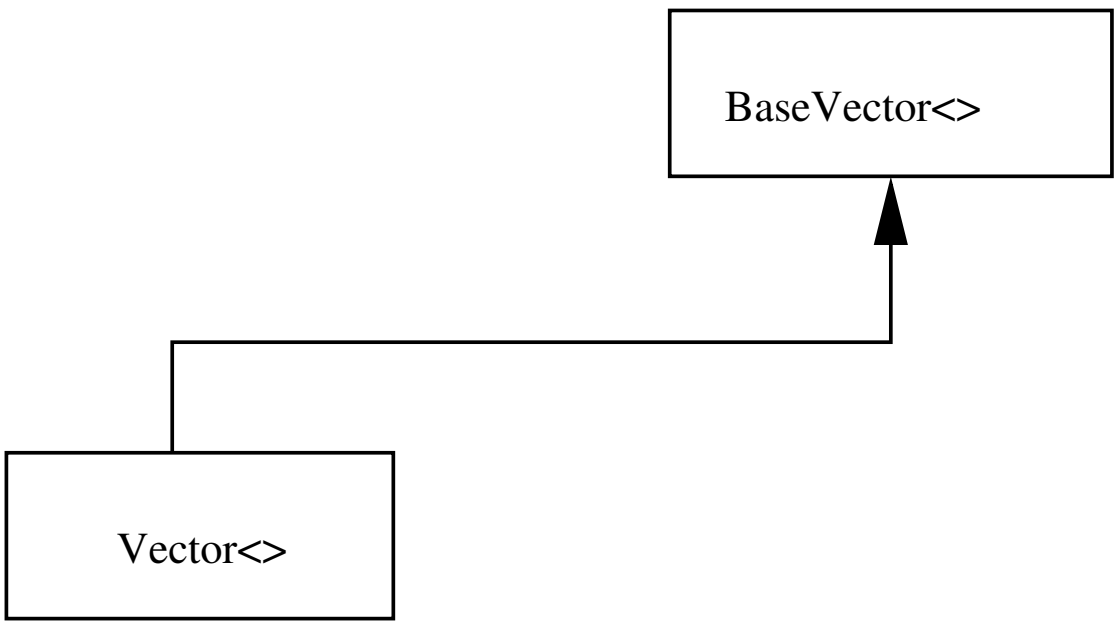
As Vandevoorde and Josuttis write, this pattern "*consists of passing a derived class as a template argument to one of its own base classes*":

```
template <class DERIVED>
class BaseVector{
public:
  typedef const DERIVED & CDR;
  CDR getCDR( void ) const {
    return static_cast<CDR>(*this);
  }
};
```

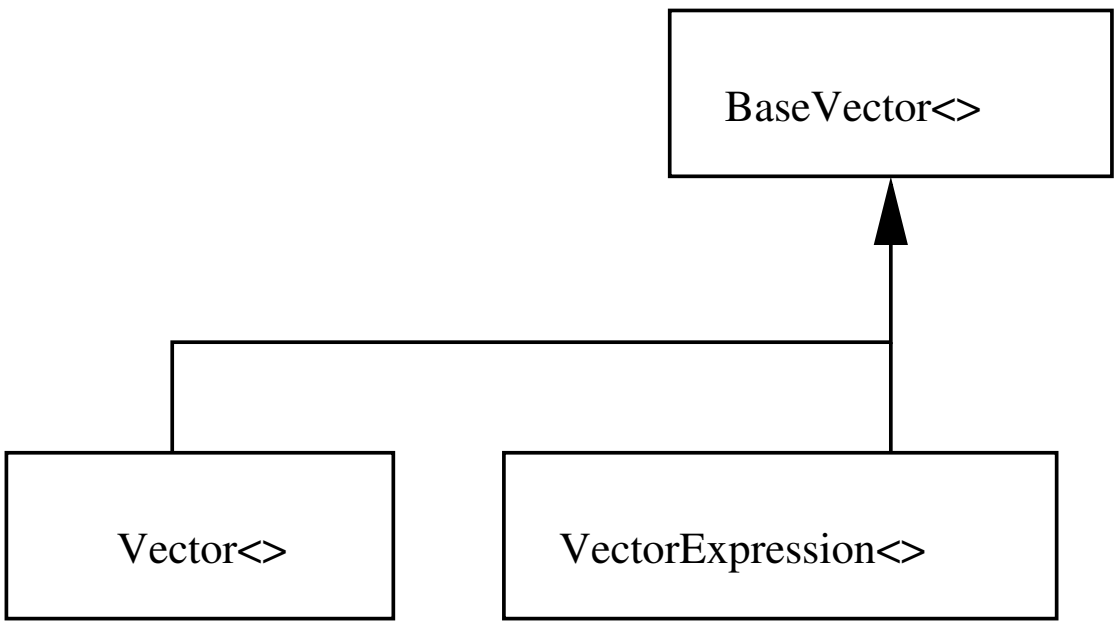
template class BaseVector<> Hierarchy



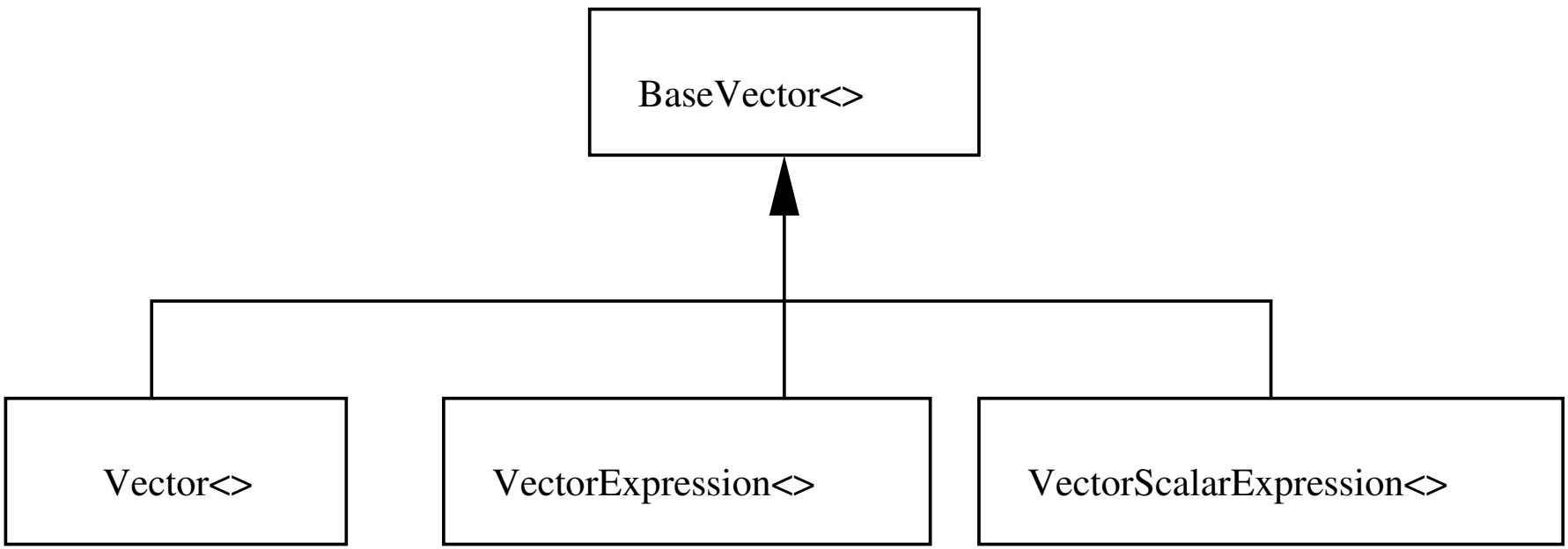
template class BaseVector<> Hierarchy



template class BaseVector<> Hierarchy



template class BaseVector<> Hierarchy



Expression Template Mechanism

Operators +/−:

- *BaseVector<L>+BaseVector<R>* $\longrightarrow$ *VectorExpression<L,Add,R>*

Expression Template Mechanism

Operators +/−:

- *BaseVector<L>+BaseVector<R>* → *VectorExpression<L,Add,R>*
- *BaseVector<L>-BaseVector<R>* → *VectorExpression<L,Minus,R>*

Expression Template Mechanism

Operators +/−:

- *BaseVector<L>+BaseVector<R>* → *VectorExpression<L,Add,R>*
- *BaseVector<L>-BaseVector<R>* → *VectorExpression<L,Minus,R>*
- *scalar*BaseVector<V>* → *VectorScalarExpression<V>*

Expression Template Mechanism

Operators +/−:

- *BaseVector<L>+BaseVector<R>* → *VectorExpression<L,Add,R>*
- *BaseVector<L>-BaseVector<R>* → *VectorExpression<L,Minus,R>*
- *scalar*BaseVector<V>* → *VectorScalarExpression<V>*
- *BaseVector<V>*scalar* → *VectorScalarExpression<V>*

Expression Template Mechanism

Operators +/−:

- *BaseVector<L>+BaseVector<R>* → *VectorExpression<L,Add,R>*
- *BaseVector<L>-BaseVector<R>* → *VectorExpression<L,Minus,R>*
- *scalar*BaseVector<V>* → *VectorScalarExpression<V>*
- *BaseVector<V>*scalar* → *VectorScalarExpression<V>*
- *Vector<T>=BaseVector<T>* → *BaseVector<>[i] evaluation*

Expression Template Mechanism

Operators +/-:

- *BaseVector<L>+BaseVector<R>* → *VectorExpression<L,Add,R>*
- *BaseVector<L>-BaseVector<R>* → *VectorExpression<L,Minus,R>*
- *scalar*BaseVector<V>* → *VectorScalarExpression<V>*
- *BaseVector<V>*scalar* → *VectorScalarExpression<V>*
- *Vector<T>=BaseVector<T>* → *BaseVector<>[i] evaluation*
- Then one can write vector expressions like :
  `X=Y*2.0+3.0*(Z-W);`
  that should perform as well as their loop based counterparts:
  `for (int i=0 ; i < N ; i++) X[i]=Y[i]*2.0+3.0*(Z[i]-W[i]);`

```
template <class ELEMENT_TYPE>
class Vector : public BaseVector< Vector<ELEMENT_TYPE> >
{
public:
  ...
  template <class DERIVED>
  Vector & operator = (const BaseVector<DERIVED> & right){
    const DERIVED & r=right.getCDR();
    for (SizeType i=0 ; i < size_ ; i++) data_[i]=r[i];
    return (*this);
  }
  ...
private:
  ELEMENT_TYPE * data_;
  SizeType       size_;
};
```

Blocked ET + ATLAS

Use the good ATLAS performance for the copy operation inside our ET Vector class.

Vector class modification

```
template <class ELEMENT_TYPE>
class Vector : public BaseVector< Vector<ELEMENT_TYPE> >
{
public:
  ...
  template <class DERIVED>
  Vector & operator = (const BaseVector<DERIVED> & right){
    const DERIVED & r=right.getCDR();

    for (SizeType i=0 ; i < size_ ; i++) data_[i]=r[i];

    return (*this);
  }
  ...
private:
  ELEMENT_TYPE * data_;
  SizeType       size_;
};
```

Vector class modification

```
template <class ELEMENT_TYPE>
class Vector : public BaseVector< Vector<ELEMENT_TYPE> >
{
public:
  ...
  template <class DERIVED>
  Vector & operator = (const BaseVector<DERIVED> & right){
    const DERIVED & r=right.getCDR();
    //for (SizeType i=0 ; i < size_ ; i++) data_[i]=r[i];
    BlockAssign<ELEMENT_TYPE>::apply(size_,r,data_);

    return (*this);
  }
  ...
private:
  ELEMENT_TYPE * data_;
  SizeType       size_;
};
```

```
template <>
struct BlockAssign<double>{
  static const int largeSize=50000, blockSize=1024;
  template <class DERIVED>
  static inline void apply(int N,
                           const DERIVED & source,
                           double * & target) {
    if (N<largeSize){
      for (int i=0 ; i < N ; i++) target[i]=source[i];
    }
    else{
      double * tempo = new double[blockSize];
      const int nblocks=N/blockSize;
      int offset=0;
      for (int i=0 ; i<nblocks ; i++){
        for (int j=0 ; j < blockSize ; j++)
           tempo[j]=source[j+offset];
        ATL_dcopy(blockSize,tempo,1,target+offset,1);
        offset+=blockSize;
      }
      for (int i=offset ; i< N ; i++) target[i]=source[i];
      delete[] tempo;
    }
  }
};
```

```
template <> struct BlockAssign<double>{
  static const int largeSize=50000, blockSize=1024;
  template <class VECTOR_EXPRESSION> class CopyFunctor{
  public:
    typedef double * RealPtr;
    CopyFunctor(const VECTOR_EXPRESSION & source,
                const RealPtr target,
                int blockSize):source_(source),
                               target_(target),
                               blockSize_(blockSize){}
    inline void operator ()(const tbb::blocked_range<int>
       & r) const {
      double * buffer = new double[blockSize];
      for (int i=r.begin() ; i!=r.end() ; i++){
        const int offset=i*blockSize_;
        for (int j=0 ; j < blockSize_ ; j++) buffer[j]=
            source_[j+offset];
        cblas_dcopy(blockSize_,buffer,1,target_+offset,1);
      }
      delete[] buffer;
    }
  private:
    const VECTOR_EXPRESSION & source_;
    const RealPtr target_;
    const int blockSize_;
  };
```

```
  template <class DERIVED>
  static inline void apply(int N,
                           const DERIVED & source,
                           Data & target) {
    if (N<largeSize){
      for (int i=0 ; i < N ; i++) target[i]=source[i];
    }
    else{

      const int nblocks=N/blockSize;
      tbb::task_scheduler_init init(-1);

      tbb::parallel_for(tbb::blocked_range<int>(0,nblocks)
          ,
        CopyFunctor<DERIVED>(source,target,blockSize));

      for (int i=blockSize*nblocks ; i < N ; i++)
        target[i]=source[i];
    }
  }
};
```