Multiplication matrice creuse–vecteur dense exacte et efficace dans Fflas-Ffpack.

Brice Boyer
LIP6, UPMC, France

Joint work with Pascal Giorgi (LIRMM), Clément Pernet (LIG, ENSL), Bastien Vialla (LIRMM)

11 Mars 2015

Séminaire Performance et Généricité LRDE

Motivations/Goals

Facts

– FFLAS-FFPACK is a C++ exact linear algebra library based on numerical BLAS and fast (sub-cubic) algorithms.

$\leadsto$ Its performance mainly depends on `FFLAS::fgemm` operation.

Motivations/Goals

Facts

- FFLAS-FFPACK is a C++ exact linear algebra library based on numerical BLAS and fast (sub-cubic) algorithms.

$\rightsquigarrow$ Its performance mainly depends on `FFLAS::fgemm` operation.

- LinBox is a generic C++ exact linear algebra library with emphasis on dense (FFLAS-FFPACK) and *black-box* algorithms.

$\rightsquigarrow$ Create a building block `FFLAS::fspmv` for *black-box* algorithms.

Motivations/Goals

Problems

⚠ Our matrices can be several Gb large (`http://hpac.imag.fr/gbla`).

Motivations/Goals

Problems

⚠ Our matrices can be several Gb large (`http://hpac.imag.fr/gbla`).

– SPMV has a huge literature, many available numerical implementations

Motivations/Goals

Problems

⚠ Our matrices can be several Gb large (`http://hpac.imag.fr/gbla`).

- SPMV has a huge literature, many available numerical implementations
- SPMV efficiency depends primarily on storage

Motivations/Goals

Problems

- ⚠ Our matrices can be several Gb large (`http://hpac.imag.fr/gbla`).
- – SPMV has a huge literature, many available numerical implementations
- – SPMV efficiency depends primarily on storage
- ⚠ sparse-BLAS standard is usually not implemented

Motivations/Goals

Problems

⚠ Our matrices can be several Gb large (`http://hpac.imag.fr/gbla`).

– SPMV has a huge literature, many available numerical implementations

– SPMV efficiency depends primarily on storage

⚠ sparse-BLAS standard is usually not implemented

⇝ implementation in libraries like Nvidia cuSPARSE (gpu) or Intel MLK (multicore cpu)

Motivations/Goals

Problems

⚠ Our matrices can be several Gb large (`http://hpac.imag.fr/gbla`).

– SPMV has a huge literature, many available numerical implementations

– SPMV efficiency depends primarily on storage

⚠ sparse-BLAS standard is usually not implemented

⇝ implementation in libraries like Nvidia cuSPARSE (gpu) or Intel MLK (multicore cpu)

– MKL proposes a new sparse API with auto-tuning

Motivations/Goals

Problems

⚠ Our matrices can be several Gb large (`http://hpac.imag.fr/gbla`).

– SPMV has a huge literature, many available numerical implementations

– SPMV efficiency depends primarily on storage

⚠ sparse-BLAS standard is usually not implemented

⤳ implementation in libraries like Nvidia cuSPARSE (gpu) or Intel MLK (multicore cpu)

– MKL proposes a new sparse API with auto-tuning

⤳ provide an *efficient* implementation in FFLAS-FFPACK

Motivations/Goals

B. Boyer, J.-G. Dumas, and P. Giorgi.
Exact sparse matrix-vector multiplication on GPU's and multicore architectures.
PASCO 2010.

B. Boyer, J.-G. Dumas, P. Giorgi, C. Pernet, and B.D. Saunders.
Elements of design for containers and solutions in the LinBox library.
In ICMS 2014.

P. Giorgi and B. Vialla.
Generating optimized sparse matrix vector product over finite fields.
In ICMS 2014.

Motivations/Goals

Problems

- ⚠ Our matrices can be several Gb large (`http://hpac.imag.fr/gbla`).
- – SPMV has a huge literature, many available numerical implementations
- – SPMV efficiency depends primarily on storage
- ⚠ sparse-BLAS standard is usually not implemented
- ⇝ implementation in libraries like Nvidia cuSPARSE (gpu) or Intel MLK (multicore cpu)
- – MKL proposes a new sparse API with auto-tuning
- ⇝ provide an *efficient* implementation in FFLAS-FFPACK

Basics: Delaying reduction

Consider :

- $\mathbb{F}_p$ represented on a machine type (= `double`, `uint64_t`,...)
- `fdot(F,x,n,y)` computing the dot product:

```
for ( ; i < n ; ++i) F.axpyin(res,x[i],y[i])
```

Basics: Delaying reduction

Consider :

- $\mathbb{F}_p$ represented on a machine type ($=$ `double`, `uint64_t`,...)
- `fdot(F,x,n,y)` computing the dot product:
  `for ( ; i < n ; ++i) F.axpyin(res,x[i],y[i])`

Instead of doing modular reduction at each `F.axpyin` call :

- one can delay the `mod` operation.
- delay `mod` as long as *e.g.* $res + (p-1)^2 < 2^{53}$ on `double`.
- do the accumulation numerically using `cblas_?dot`

Basics: Delaying reduction

Consider :

- $\mathbb{F}_p$ represented on a machine type (= `double`, `uint64_t`,...)
- `fdot(F,x,n,y)` computing the dot product:
  `for ( ; i < n ; ++i) F.axpyin(res,x[i],y[i])`

Inste

- or
- de

J-G. Dumas, P. Giorgi, and C. Pernet.
Dense linear algebra over word-size prime fields: the FFLAS and FFPACK packages.
ACM Trans. Math. Softw., 2008.

- do the accumulation **numerically** using `cblas_?dot`

Outline

1. SIMD tools

Outline

1. SIMD tools
2. Helpers and Strategies
   - Controller/Module Design
   - Helper structures

Outline

Outline

Why SIMD?

- data parallelism available on many CPUs
- support from SSE4.1 up to AVX2, MIC to come, fall back provided
- little/no overhead

Why SIMD?

- data parallelism available on many CPUs
- support from SSE4.1 up to AVX2, MIC to come, fall back provided
- little/no overhead
- minimal knowledge writing, automatic implementation choosing

Why SIMD?

- data parallelism available on many CPUs
- support from SSE4.1 up to AVX2, MIC to come, fall back provided
- little/no overhead
- minimal knowledge writing, automatic implementation choosing
- no need to depend on an external library (boost,...)
- ad-hoc optimised functions (modp, gcd) other than BLAS-provided
- specialised integer functions usually missing

Writing SIMD

Example

```
/* C = A + B mod P (positive or balanced representation) */
using simd = Simd<Element>;
C = simd::add(A, B);
Q = simd::vand(simd::greater(C, MAX),NEGP);
if (!positive) {
    T = simd::vand(simd::lesser(C, MIN),P);
    Q = simd::vor(Q, T);
}
C = simd::add(C, Q);
```

SIMD implementation

3 layer implementation :

- `template <class T> using Simd = typename SimdChooser<T>::value;`

SIMD implementation

3 layer implementation :

- `template <class T> using Simd = typename SimdChooser<T>::value;`
- `SimdChooser` chooses among `Simd128<T>` and `Simd256<T>`

SIMD implementation

3 layer implementation :

- `template <class T> using Simd = typename SimdChooser<T>::value;`
- `SimdChooser` chooses among `Simd128<T>` and `Simd256<T>`
- template <class T> using Simd128 =

```
template <class T> using Simd128 =
Simd128_impl<std::is_arithmetic<T>::value,
              std::is_integral<T>::value,
              std::is_signed<T>::value,
              sizeof(T)
             >;
```

SIMD implementation

3 layer implementation :

- `template <class T> using Simd = typename SimdChooser<T>::value;`
- `SimdChooser` chooses among `Simd128<T>` and `Simd256<T>`
-
```
template <class T> using Simd128 =
Simd128_impl<std::is_arithmetic<T>::value,
             std::is_integral<T>::value,
             std::is_signed<T>::value,
             sizeof(T)
            >;
```
- covers SIMD 128/256 and `float`/`double` or `(u)intXX_t`

SIMD performance

- allocators provide automatically aligned data
- $\approx$ 8–10$\times$ faster for AVX2 compared to `fmod` on `double`
- $\approx$ 4$\times$ faster for AVX compared to `%` on `int32_t`
- BLAS-like implementation of `igemm` on `int64_t` is only 50% slower than OpenBLAS `fgemm` on `double` but we can reach higher modulo.

Outline

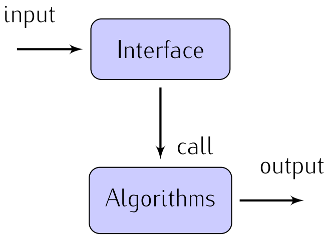
Generic design in FFLAS-FFPACK

- many algorithms for one problem (*e.g.* `fgemm`)
- redundant code
- difficult to generalise for new fields and new algorithms: ad-hoc specialisations

Outline

Algorithm Design

Strategy Design Pattern

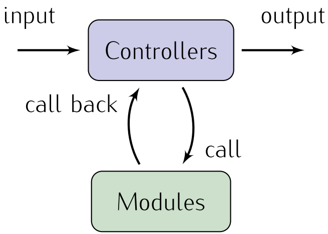


Strategy Design Pattern

E. Gamma.
Design Patterns: Elements of Reusable Object-Oriented Software.
Addison-Wesley Professional Computing Series.
Addison-Wesley, 1995.

Algorithm Design

Controller/Module Design Pattern



Algorithm Design

Controller/Module Design Pattern

input output

Van-Dat Cung, Vincent Danjean, Jean-Guillaume Dumas, Thierry Gautier, Guillaume Huard, Bruno Raffin, Christophe Rapine, Jean-Louis Roch, and Denis Trystram.

Adaptive and hybrid algorithms: classification and illustration on triangular system solving.
In Jean-Guillaume Dumas, editor, Proceedings ofTransgressive Computing2006, Granada, España, April 2006.

Example: Cascading (dense)

Controller/Module algorithm

Algorithm 1: `MatMul` controller

Input: A and B resp. $n \times k$ and $k \times n$.
Input: H General Helper
Output: $C = A \times B$
if $\min(m, k, n) <$ H.`threshold`() then
  `MatMul`(C,A,B,`Base`()) ;
else
  `MatMul`(C,A,B,`Recursive`())
end

Algorithm 2: `MatMul` module

Input: A, B, C as in controller.
Input: `Recursive` Helper
Output: $C = A \times B$
Cut A,B,C in S_i, T_i
...
`MatMul`(P_i,S_i,T_i,H)
...

Outline

Helpers

Structures

- light-weight
- selects a (class of) algorithm
- stores information (largest entry, hints, other representation,...)
- partial specialisation

Code Example

```
template<class Field,
        typename AlgoTrait   = MMHelperAlgo::Auto,
        typename ModeTrait   = typename ModeTraits<Field>::value,
        typename ParSeqTrait = ParSeqHelper::Sequential >
struct MMHelper {
    ...
};
```

Code Example

```
void fgemm (const Givaro::DoubleDomain& F,
            const FFLAS_TRANSPOSE ta,
            const FFLAS_TRANSPOSE tb,
            const size_t m, const size_t n,const size_t k,
            const Givaro::DoubleDomain::Element alpha,
            Givaro::DoubleDomain::ConstElement_ptr Ad,
            const size_t lda,
            ...
            MMHelper<Givaro::DoubleDomain, MMHelperAlgo::Classic>
                 &H)
{
    H.setOutBounds(k, alpha, beta);

    cblas_dgemm (CblasRowMajor, (CBLAS_TRANSPOSE) ta,
                 (CBLAS_TRANSPOSE) tb, (int)m, (int)n, (int)k,
                 (Givaro::DoubleDomain::Element) alpha,
                 ...
                 Cd, (int)ldc);
}
```

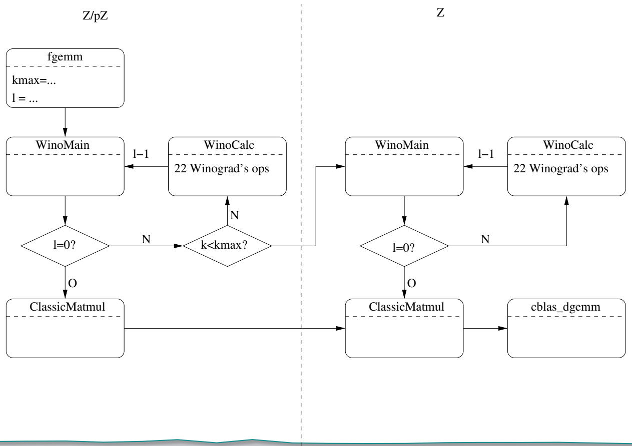
Helpers (Cont'ed)

Benefits

- fewer routine names, more entry points
- simplified/more generic structure of the algorithm

Helpers (Cont'ed)

former `fgemm` structure in FFLAS



Helpers (Cont'ed)

Benefits

- fewer routine names, more entry points
- simplified/more generic structure of the algorithm

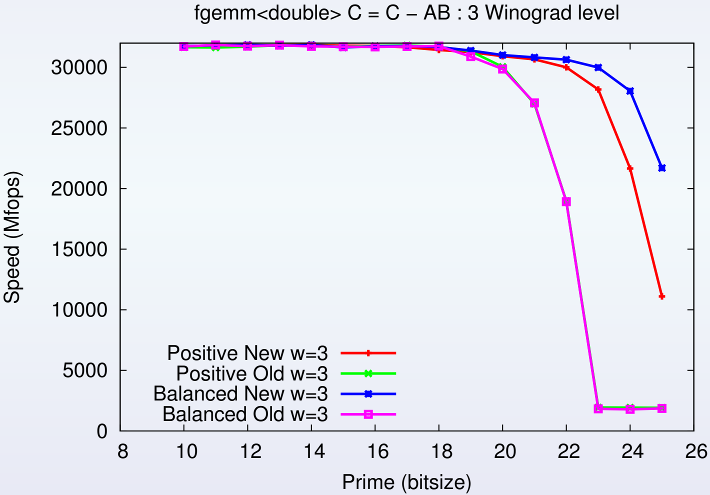
Helpers (Cont'ed)

Benefits

- fewer routine names, more entry points
- simplified/more generic structure of the algorithm
- discovers the best strategy without (inaccurate) pre-computations
- improved delayed reductions (larger primes reachable, fewer modulo)
- faster overall algorithm in all cases

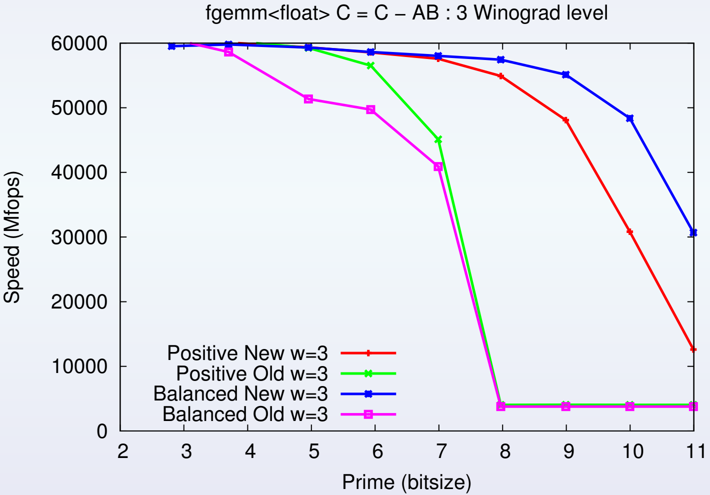
Helpers: `fgemm` timings

Perfs over `double` $n = 6000$



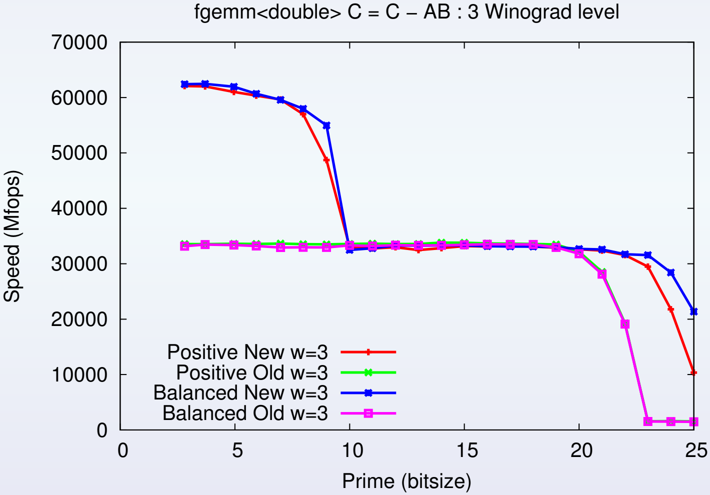
Helpers: `fgemm` timings

Perfs over `float` $n = 6000$



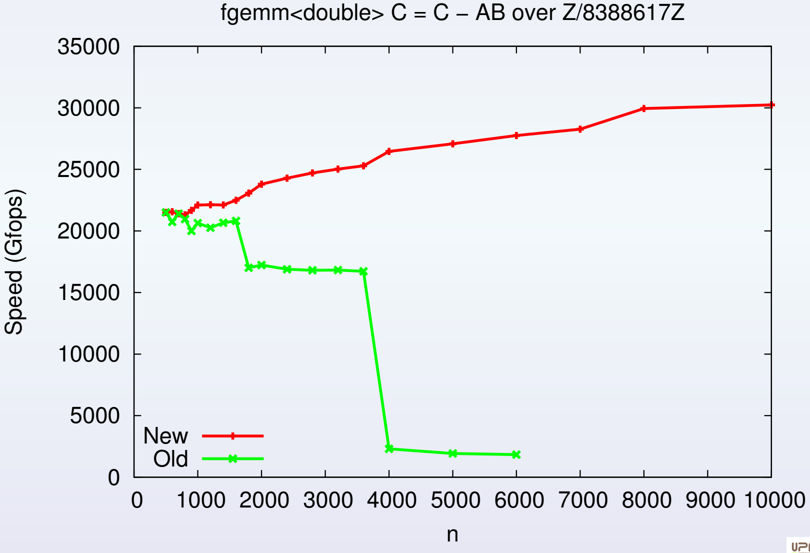
Helpers: `fgemm` timings

Automatic **`double`** $\rightarrow$**`float`** switching ($n = 10\,000$



Helpers: `fgemm` timings

Impact on asymptotic perfs $p = 8388617$



Outline

SPMV overview

Classical formats

- COO (coordinate, `row`/`col`/`data` triplets)
- CSR (compressed row, $\approx$ stores starting point of each new row in `col` or `data`)
- ELL (row major table of `col` indices and `data` value, row i in tables correspond to row i in matrix)

SPMV overview

Special formats

Matrices with constant entries (especially ± 1) can be stored:

- without `data` field
- delay reduction much further (no multiplication, just additions)
- implemented for COO/CSR/ELL

SPMV overview

Hybrid formats

- Write $A = A_0 + A_1 + \cdots + A_k$ with A_j in some dedicated format, such as ELL+CSR, or ELL+COO or CSR_ZO+CSR...
- hybrid CSR stores, for each line, the columns in `col` that first correspond data -1 then $+1$ and then the rest.

SPMV overview

SIMD friendly formats (AVX)

SELL :

- like ELL but rows are sorted by weight
- 4 consecutive rows in a table are stored column major

SPMV overview

SIMD friendly formats (AVX)

SELL :

- like ELL but rows are sorted by weight
- 4 consecutive rows in a table are stored column major

$\rightsquigarrow$ loads 4 rows at a time

SPMV overview

SIMD friendly formats (AVX)

SELL :

- like ELL but rows are sorted by weight
- 4 consecutive rows in a table are stored column major

$\leadsto$ loads 4 rows at a time

SCSR (new?) :

- data are blocks of 4 consecutive non all zero elements
- consecutive columns are stored as `j k`
  ie. j is the first column of k consecutive 4–blocks
- single columns are masked by 2^{32}

SPMV overview

SIMD friendly formats (AVX)

SELL :

- like ELL but rows are sorted by weight
- 4 consecutive rows in a table are stored column major

⇝ loads 4 rows at a time

SCSR (new?) :

- data are blocks of 4 consecutive non all zero elements
- consecutive columns are stored as `j k`
  ie. j is the first column of k consecutive 4-blocks
- single columns are masked by 2^{32}

⇝ `col` always smaller, `data` may be $4\times$ larger

⚠ not efficient if matrix not 'locally' dense

SPMM overview

SIMD parallelism happens on the dense matrix

SPMV/SPMM architecture

- Implementation for generic and delayed fields
- matrices are supposed to be cut so that no reduction is needed
- use of numerical sparse blas (intel) or self-made routines (other data type)

SPMV/SPMM performance (early results)

- for CSR our code is $\approx 20\%$ slower than MKL
  (but we are working on it and we support more data types)
- ± 1 formats or SCSR are competitive for special matrices
- SELL is generally faster than CSR.

Merci
pour votre attention !

Multiplication matrice creuse–vecteur dense exacte et efficace dans Fflas–Ffpack.

Brice Boyer
LIP6, UPMC, France

Joint work with Pascal Giorgi (LIRMM), Clément Pernet (LIG, ENSL), Bastien Vialla (LIRMM)
11 Mars 2015

Séminaire Performance et Généricité LRDE