Que peut-on faire *exactement*, avec l’algèbre linéaire flottante

Clément PERNET

Université de Grenoble, LIG INRIA-MOAIS

M2RMA Calcul Exact
Lundi 15 nov, 2010
Outline

1. Numerical linear algebra: the BLAS
   - Why ?
   - BLAS
   - Optimizations

2. FFLAS: a BLAS for finite fields
   - Delayed reductions
   - Cache tuning
   - Sub-cubic algorithm
   - Memory efficiency
Huge range of applications in numerical computations

- All PDE based computations: Weather forecasts, mechanical designs, computational chemistry, ...
- ODE, Control, ...

boil down to linear algebra efficiency.
Huge range of applications in numerical computations
- All PDE based computations: Weather forecasts, mechanical designs, computational chemistry, ...
- ODE, Control, ...

boil down to linear algebra efficiency.

But
- many algorithms
- many architectures
Huge range of applications in numerical computations

- All PDE based computations: Weather forecasts, mechanical designs, computational chemistry, ...

- ODE, Control, ...

boil down to linear algebra efficiency.

But

- many algorithms
- many architectures

⇒ design for long term optimizations and portability?
BLAS : Basic Linear Algebra Subroutines

1979 [Lawson & Al.], first set of Fortran subroutines
1988 [Dongarra & Al], level 2 (MatVect)
1990 [Dongara & Al], level 3 (MatMul)
BLAS : Basic Linear Algebra Subroutines

1979  [Lawson & Al.], first set of Fortran subroutines
1988  [Dongarra & Al], level 2 (MatVect)
1990  [Dongara & Al], level 3 (MatMul)

Provide:
- an standard interface (Fortran77 or C)
- a reference, portable implementation
BLAS : Basic Linear Algebra Subroutines

1979 [Lawson & Al.], first set of Fortran subroutines
1988 [Dongarra & Al], level 2 (MatVect)
1990 [Dongara & Al], level 3 (MatMul)

Provide:
- an standard interface (Fortran77 or C)
- a reference, portable implementation

Optimized implementations :
- machine specific by computer vendors (Intel, SGI, IBM, ...)
- architecture independent: ATLAS, GOTO.
Features

4 data types: float (s), double (d), complex (c), double cpx (z)

3 levels:

level 1 Vector ops (rotation, dot-prod, add, scal axpy,...)

level 2 Matrix-Vector ops (MatVect prod, triangular system solve, tensor product,...)

level 3 Matrix-Matrix ops (MatMul, multi triangular system solve,...)
Features

4 data types: float (s), double (d), complex (c), double cpx (z)

3 levels:

level 1 Vector ops (rotation, dot-prod, add, scal axpy,...)

level 2 Matrix-Vector ops (MatVect prod, triangular system solve, tensor product,...)

level 3 Matrix-Matrix ops (MatMul, multi triangular system solve,...)
Optimizing data locality

Memory considerations:
- **CPU-Memory communication**: bandwidth gap
  ⇒ Hierarchy of several cache memory levels
Optimizing data locality

Memory considerations:

- **CPU-Memory communication**: bandwidth gap
  - ⇒ Hierarchy of several cache memory levels
- Row major representation of matrices

Clément Pernet
Optimizing data locality

Memory considerations:

- **CPU-Memory communication:** bandwidth gap
  - Hierarchy of several cache memory levels
- Row major representation of matrices
- a RAM memory access can fetch a bunch of contiguous elements
Optimizing data locality

Comparing

\[
\begin{align*}
\text{for } i=1 \text{ to } n \text{ do} \\
& \quad \text{for } j=1 \text{ to } n \text{ do} \\
& \quad \quad \text{for } k=1 \text{ to } n \text{ do} \\
& \quad \quad \quad C_{i,j} \leftarrow C_{i,j} + A_{i,k} B_{k,j} \\
& \quad \end{align*}
\]

end for
end for
Comparing

\begin{verbatim}
for i=1 to n do
  for j=1 to n do
    for k=1 to n do
      C_{i,j} \leftarrow C_{i,j} + A_{i,k} B_{k,j}
    end for
  end for
end for
\end{verbatim}

VS

\begin{verbatim}
for i=1 to n do
  for k=1 to n do
    for j=1 to n do
      C_{i,j} \leftarrow C_{i,j} + A_{i,k} B_{k,j}
    end for
  end for
end for
\end{verbatim}
Further memory optimizations

Larger dimensions: cache blocking.

⇒ split matrices into blocks, s.t. their product can be computed within the cache.
Further memory optimizations

Larger dimensions: cache blocking.
⇒ split matrices into blocks, s.t. their product can be computed within the cache.

Reuse of the data
- if Work ≫ Data: memory fetch is amortized
  ⇒ reach the peak performance of the CPU
- Matrix multiplication: $n^3 ≫ n^2$
  ⇒ well suited for block design
Arithmetic optimizations

- fma (fused multiply and accumulate) $z \leftarrow z + x \times y$
- SSE: 128 bits registers
- pipeline
- ...

Que peut-on faire exactement, avec l’algèbre linéaire flottante
Arithmetic optimizations

- fma (fused multiply and accumulate) $z \leftarrow z + x \times y$
- SSE: 128 bits registers
- pipeline
- ...

Tends to give advantage to floating point arithmetic up to now.
Outline

1. Numerical linear algebra: the BLAS
   - Why?
   - BLAS
   - Optimizations

2. FFLAS: a BLAS for finite fields
   - Delayed reductions
   - Cache tuning
   - Sub-cubic algorithm
   - Memory efficiency
Overview

- word sized finite fields: elements can be represented on 16, 23, 32, 53 or 64 bits
- Delayed modular reductions: avoid unnecessary field arithmetic by computing over $\mathbb{Z}$ as much as possible.
- Cache tuning
- Fast sub-cubic algorithm
Delayed reductions

Existence of 2 ring homomorphisms:

- $\Phi : GF(q) \rightarrow \mathbb{Z}$
- $\Psi : \mathbb{Z} \rightarrow GF(q)$

$GF(q) \xrightarrow{\Phi} \mathbb{Z}$

s.t. $+_{GF(q)}, \times_{GF(q)}$ commute with $+_{\mathbb{Z}}, \times_{\mathbb{Z}}$

$GF(q) \xleftarrow{\Psi} \mathbb{Z}$
Existence of 2 ring homomorphisms:

- $\Phi : GF(q) \rightarrow \mathbb{Z}$
- $\Psi : \mathbb{Z} \rightarrow GF(q)$

$GF(q) \xrightarrow{\Phi} \mathbb{Z}$

s.t. $GF(q) \xrightarrow{+_{GF(q)}, \times_{GF(q)}}, \mathbb{Z} \xrightarrow{+_{\mathbb{Z}}, \times_{\mathbb{Z}}}$ commutes

$GF(q) \xleftarrow{\Psi} \mathbb{Z}$

$\mathbb{Z}_p : \Phi = Id, \Psi : x \mapsto x \mod p$

$GF(p^k) : \Phi : P(X) \rightarrow P(\gamma)$ with $\gamma > nk(p - 1)$. ($\gamma$-adic reconstruction).
Delayed reductions

compute over \( \mathbb{Z} \) with word size elements (int, long, float double)
perform the necessary back conversion (\( \Psi \)) only when necessary.

Conditions of validity:

\[
\mathbb{Z}_p : \quad n(p - 1)^2 < 2^m
\]

\[
GF(p^k) : \quad q^{(2k - 1)} < 2^m \text{ and } \gamma > nk(p - 1).
\]
Cache tuning

Could mimic the numerical BLAS.

⇒ huge amount of work
Cache tuning

Could mimic the numerical BLAS.
⇒ huge amount of work

Instead:

Reuse the existing technology: compute with floating points and use BLAS.
Cache tuning

Could mimic the numerical BLAS.
⇒ huge amount of work
Instead:

Reuse the existing technology: compute with floating points and use BLAS.

Pros:

Cons:

- exponent is useless
Cache tuning

Could mimic the numerical BLAS.
⇒ huge amount of work
Instead:

Reuse the existing technology: compute with floating points and use BLAS.

Pros:
- floating point arithmetic is better optimized

Cons:
- exponent is useless
Could mimic the numerical BLAS.  
⇒ huge amount of work  
Instead:

Reuse the existing technology: compute with floating points and use BLAS.

Pros:
- floating point arithmetic is better optimized

Cons:
- exponent is useless
- integer arithmetic may become as efficient
Cache tuning

Could mimic the numerical BLAS.
⇒ huge amount of work
Instead:

Reuse the existing technology: compute with floating points and use BLAS.

Pros:
- floating point arithmetic is better optimized
- long term efficiency: rely on the numerical community

Cons:
- exponent is useless
- integer arithmetic may become as efficient
Strassen-Winograd algorithm

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix} =
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix},
\]

- 8 additions:

\[
\begin{align*}
S_1 & \leftarrow A_{21} + A_{22} \\
S_2 & \leftarrow S_1 - A_{11} \\
S_3 & \leftarrow A_{11} - A_{21} \\
S_4 & \leftarrow A_{12} - S_2
\end{align*}
\]

\[
\begin{align*}
T_1 & \leftarrow B_{12} - B_{11} \\
T_2 & \leftarrow B_{22} - T_1 \\
T_3 & \leftarrow B_{22} - B_{12} \\
T_4 & \leftarrow T_2 - B_{21}
\end{align*}
\]

- 7 recursive multiplications:

\[
\begin{align*}
P_1 & \leftarrow A_{11} \times B_{11} \\
P_2 & \leftarrow A_{12} \times B_{21} \\
P_3 & \leftarrow S_4 \times B_{22} \\
P_4 & \leftarrow A_{22} \times T_4
\end{align*}
\]

\[
\begin{align*}
P_5 & \leftarrow S_1 \times T_1 \\
P_6 & \leftarrow S_2 \times T_2 \\
P_7 & \leftarrow S_3 \times T_3
\end{align*}
\]

- 7 final additions:

\[
\begin{align*}
U_1 & \leftarrow P_1 + P_2 \\
U_2 & \leftarrow P_1 + P_6 \\
U_3 & \leftarrow U_2 + P_7 \\
U_4 & \leftarrow U_2 + P_5 \\
U_5 & \leftarrow U_4 + P_3 \\
U_6 & \leftarrow U_3 - P_4 \\
U_7 & \leftarrow U_3 + P_5
\end{align*}
\]

The result is the matrix:

\[
C = \begin{bmatrix}
U_1 & U_5 \\
U_6 & U_7
\end{bmatrix}
\]
Strassen-Winograd algorithm

Used to be considered as not practicable:

- threshold too high
- numerical stability
Strassen-Winograd algorithm

Used to be considered as not practicable:
- threshold too high
- numerical stability

Over finite fields: not problem
- update the validity condition for delayed reductions from
  \[ k(p - 1)^2 < 2^{53} \]
  to
  \[ \left( \frac{1+3^l}{2^l} \right)^2 \left\lfloor \frac{k}{2^l} \right\rfloor (p - 1)^2 < 2^{53} \]
  for \( l \) recursive levels.
Strassen-Winograd algorithm

Used to be considered as not practicable:
- threshold too high
- numerical stability

Over finite fields: not problem
- update the validity condition for delayed reductions from
  \[ k(p - 1)^2 < 2^{53} \]
  to
  \[ \left( \frac{1+3^l}{2} \right)^2 \left\lceil \frac{k}{2^l} \right\rceil (p - 1)^2 < 2^{53} \]
  for \( l \) recursive levels.

Pros:
- faster

Cons:

Clément Pernet
Que peut-on faire *exactement*, avec l’algèbre linéaire flottante
Strassen-Winograd algorithm

Used to be considered as not practicable:

- threshold too high
- numerical stability

Over finite fields: not problem

- update the validity condition for delayed reductions from
  \[ k(p - 1)^2 < 2^{53} \]
  
  to
  \[ \left( \frac{1+3^l}{2} \right)^2 \left\lfloor \frac{k}{2^l} \right\rfloor (p - 1)^2 < 2^{53} \]

for \( l \) recursive levels.

Pros:
- faster

Cons:
- more reductions if \( q \) or \( n \) is big
Strassen-Winograd algorithm

Used to be considered as not practicable:
- threshold too high
- numerical stability

Over finite fields: not a problem

- update the validity condition for delayed reductions from
  \[ k(p - 1)^2 < 2^{53} \]
  to
  \[ \left( \frac{1+3^l}{2} \right)^2 \left\lceil \frac{k}{2^l} \right\rceil (p - 1)^2 < 2^{53} \]
  for \( l \) recursive levels.

Pros:
- faster

Cons:
- more reductions if \( q \) or \( n \) is big
- temporary memory allocations
Memory requirements of Winograd’s algorithm

\[
\begin{align*}
C & \leftarrow A \times B + C \Rightarrow \text{from 3 to 2 temp. (3 pre-adds)} \\
C & \leftarrow A \times B + C \Rightarrow \text{from 3 to 2 temp. (2 pre-adds, overwriting inputs)} \\
C & \leftarrow A \times B \text{ fully in-place (overwriting inputs)}
\end{align*}
\]

Question: Is there an in-place $O(n^2) \cdot 807$ algorithm with constant inputs? \Rightarrow yes

7.2n^2 \cdot 807 \text{ instead of } 6n^2 \cdot 807
Memory requirements of Winograd’s algorithm

- $C \leftarrow A \times B + C \Rightarrow$ from 3 to 2 temp. (3 pre-adds)
- $C \leftarrow A \times B + C \Rightarrow$ from 3 to 2 temp. (2 pre-adds, overwriting inputs)
- $C \leftarrow A \times B$ fully in-place (overwriting inputs)
Memory requirements of Winograd’s algorithm

- $C \leftarrow A \times B + C \Rightarrow$ from 3 to 2 temp. (3 pre-adds)
- $C \leftarrow A \times B + C \Rightarrow$ from 3 to 2 temp. (2 pre-adds, overwriting inputs)
- $C \leftarrow A \times B$ fully in-place (overwriting inputs)

**Question:**

Is there an in-place $O(n^{2.807})$ algorithm with constant inputs?
Memory requirements of Winograd’s algorithm

- $C \leftarrow A \times B + C \Rightarrow$ from 3 to 2 temp. (3 pre-adds)
- $C \leftarrow A \times B + C \Rightarrow$ from 3 to 2 temp. (2 pre-adds, overwriting inputs)
- $C \leftarrow A \times B$ fully in-place (overwriting inputs)

**Question:**

Is there an in-place $O(n^{2.807})$ algorithm with constant inputs?  

⇒ yes
Memory requirements of Winograd’s algorithm

- \( C \leftarrow A \times B + C \) \(\Rightarrow\) from 3 to 2 temp. (3 pre-adds)
- \( C \leftarrow A \times B + C \) \(\Rightarrow\) from 3 to 2 temp. (2 pre-adds, overwriting inputs)
- \( C \leftarrow A \times B \) fully in-place (overwriting inputs)

Question:
Is there an in-place \( O(n^{2.807}) \) algorithm with constant inputs?

\(\Rightarrow yes \ 7.2n^{2.807} \) instead of \(6n^{2.807}\)