Armadillo: Template Metaprogramming for Linear Algebra

Leo Antunes and Markus Kruber

Seminar on Automation, Compilers, and Code-Generation
HPAC

June 2014
What is Armadillo?

- C++ Library for Linear Algebra
What is Armadillo?

- C++ Library for Linear Algebra
- Aims for ease of use
- Still wants performance
What is Armadillo?

- C++ Library for Linear Algebra
- Aims for ease of use
- Still wants performance
- Abstract interface to BLAS and LAPACK
What is Armadillo?

- C++ Library for Linear Algebra
- Aims for ease of use
- Still wants performance
- Abstract interface to BLAS and LAPACK
- Pure template library (only headers)
Calculating $A^{-1}Bvv^T$:

```cpp
mat A = << 3.0 << 2.5 << endr
    << 2.0 << 1.0 << endr
mat B = randu(2,5);
colvec v = randu(5);
mat result = inv(A)*B*v*trans(v);
cout << "Result:" << endl << result.row(0) << endl;
```
Refresher: C++ Templates

- Allow type-agnostic development (e.g.: class Array<type>)
  "generics" in Java, C#; "parametric polymorphism" in Haskell, Scala
Refresher: C++ Templates

- Allow type-agnostic development (e.g.: class Array<type>)
  "generics" in Java, C#; "parametric polymorphism" in Haskell, Scala
- Realized at compile-time $\rightarrow$ code generation
Refresher: C++ Templates

- Allow type-agnostic development (e.g. class Array<type>)
  "generics" in Java, C#; "parametric polymorphism" in Haskell, Scala
- Realized at compile-time → code generation
- Generate different class for each type used in code
  (thus: "template")
Refresher: C++ Templates

- Allow type-agnostic development (e.g.: class Array<type>)
  "generics" in Java, C#; "parametric polymorphism" in Haskell, Scala

- Realized at compile-time → code generation

- Generate different class for each type used in code
  (thus: "template")

- Therefore: no dynamic typing
Refresher: C++ Templates

- Allow type-agnostic development (e.g.: class Array<type>)
  "generics" in Java, C#; "parametric polymorphism" in Haskell, Scala
- Realized at compile-time → code generation
- Generate different class for each type used in code
  (thus: "template")
- Therefore: no dynamic typing
- Allows for further optimizations by compiler
C++ Template example

Type-dependent:

```cpp
class Adder{
    int base;
    public:
    Adder(int x): base(x){}
    int doit(int y){
        return base+y;
    }
};
```

Type-agnostic:

```cpp
template<typename TYPE>
class Adder{
    TYPE base;
    public:
    Adder(TYPE x): base(x){}
    TYPE doit(TYPE y){
        return base+y;
    }
};
```
Template Metaprogramming

▷ Use C++ templates as a language in itself
Template Metaprogramming

- Use C++ templates as a language in itself
- Type specification as pattern matching → Branching (independently of C++’s control structures)
Template Metaprogramming

▷ Use C++ templates as a language in itself
▷ Type specification as pattern matching → Branching (independently of C++’s control structures)
▷ Template used inside template → Recursion
Template Metaprogramming

- Use C++ templates as a language in itself
- Type specification as pattern matching → Branching (independently of C++’s control structures)
- Template used inside template → Recursion
- Branching and Recursion → Turing complete
Template Metaprogramming

▷ Use C++ templates as a language in itself
▷ Type specification as pattern matching → Branching (independently of C++’s control structures)
▷ Template used inside template → Recursion
▷ Branching and Recursion → Turing complete
▷ Allows offloading of computation to compile-time
Factorial: normal C++

```c++
int factorial(int n) {
    if(n < 1){
        return 1;
    } else {
        return n*factorial(n-1);
    }
}
```
Factorial: Template metaprogramming

```cpp
#include <iostream>

template <int n>
struct factorial {
  const static int result =
    n * factorial<n - 1>::result;
};

template <>
struct factorial<0> {
  const static int result = 1;
};

int main() {
  std::cout << factorial<5>::result << std::endl;
  return 0;
}
```

▷ Types used for pattern matching.
▷ Structs used as functions.
▷ Most ”specific” template chosen first. (cf. Haskell)
TMP in Armadillo

- Classes for common LA objects: matrix, vector, etc
TMP in Armadillo

- Classes for common LA objects: matrix, vector, etc
- Literal expression in code converted to template tree, e.g.:
  $A \times \text{inv}(B) \rightarrow \text{Glue}<A, \text{Op}<B, \text{op\_inv}>, \text{op\_times}>$
TMP in Armadillo

- Classes for common LA objects: matrix, vector, etc
- Literal expression in code converted to template tree, e.g.:
  $A*\text{inv}(B) \rightarrow \text{Glue}<A, \text{Op}<B, \text{op}\_\text{inv}>, \text{op}\_\text{times}>
- Infers information on arguments via pattern matching
TMP in Armadillo

- Classes for common LA objects: matrix, vector, etc
- Literal expression in code converted to template tree, e.g.:
  \[ A \times \text{inv}(B) \rightarrow \text{Glue}<A, \text{Op}<B, \text{op}_\text{inv}>, \text{op}_\text{times}> \]
- Infers information on arguments via pattern matching
- User can provide information about objects. e.g.: diagmat(A) or trimatu(A)
TMP in Armadillo

- Classes for common LA objects: matrix, vector, etc
- Literal expression in code converted to template tree, e.g.:
  \[ A*\text{inv}(B) \rightarrow \text{Glue}<A, \text{Op}<B, \text{op}\_\text{inv}>, \text{op}\_\text{times}> \]
- Infers information on arguments via pattern matching
- User can provide information about objects.
  e.g.: \text{diagmat}(A) or \text{trimatu}(A)
- Actual calculation delayed until runtime request for results
Simple example: \texttt{inv(inv(A))}

- **Rationale:** should not compute anything; just return \(A\)
- Inner call to \texttt{inv()} returns a \texttt{Op<A, op_inv>} object.
- Outer call matches the definition of \texttt{inv()} with \texttt{Op<A, op_inv>} argument. Returns \(A\).
Simple example: \texttt{inv(inv(A))}

- Rationale: should not compute anything; just return A
- Inner call to \texttt{inv()} returns a \texttt{Op<A, \texttt{op_inv}>} object.
- Outer call matches the definition of \texttt{inv()} with \texttt{Op<A, \texttt{op_inv}>} argument. Returns A.
- Advantages:
  - Obvious: no actual inversion.
  - Not obvious: multiple-dispatch without the runtime overhead.
  - Allows compiler to get rid of unneeded function calls.
Matrix Chain Multiplication

- Matrix multiplication is associative.
  \[(ABC)D = A(BCD) = \ldots\]
- Order affects the number of operations.
Matrix Chain Multiplication

- Matrix multiplication is associative.
  \[(ABC)D = A(BCD) = \ldots\]
- Order affects the number of operations.
- \(A \in \mathbb{R}^{60 \times 5}, B \in \mathbb{R}^{5 \times 30}, C \in \mathbb{R}^{30 \times 10}\)
  - \((AB)C = (60 \cdot 5 \cdot 30) + (60 \cdot 30 \cdot 10) = 27,000\)
  - \(A(BC) = (5 \cdot 30 \cdot 10) + (60 \cdot 5 \cdot 10) = 4,500\)
Matrix Chain Multiplication

- Matrix multiplication is associative.
  \[(ABC)D = A(BCD) = \ldots\]

- Order affects the number of operations.

- \(A \in \mathbb{R}^{60 \times 5}, B \in \mathbb{R}^{5 \times 30}, C \in \mathbb{R}^{30 \times 10}\)
  - \((AB)C = (60 \cdot 5 \cdot 30) + (60 \cdot 30 \cdot 10) = 27,000\)
  - \(A(BC) = (5 \cdot 30 \cdot 10) + (60 \cdot 5 \cdot 10) = 4,500\)

- E.g. Matlab: naive left to right ordering

- Complexity: \(O(n^3)\) with dynamic programming
  Smarter approach: \(O(n \cdot \log(n))\)
Complex example: \( A \times B \times C \times D \)

- Instantiated as:
  \[
  \text{Glue}<\text{Glue}<\text{Glue}<A,B,\text{op\_times}>,C,\text{op\_times}>,D,\text{op\_times}>
  \]

- Determines ”depth” of expression: \( N = 3 \)
  (amount of operators with same precedence)
Complex example: $\mathbf{A} \ast \mathbf{B} \ast \mathbf{C} \ast \mathbf{D}$

- Instantiated as:
  $\text{Glue} < \text{Glue} < \text{Glue} < \mathbf{A}, \mathbf{B}, \text{op\_times} > , \mathbf{C}, \text{op\_times} > , \mathbf{D}, \text{op\_times} >$

- Determines "depth" of expression: $N = 3$
  (amount of operators with same precedence)

- Pattern matches specialized function for $N = 3$ which:
  - Decide $(\mathbf{A}\mathbf{B}\mathbf{C})\mathbf{D}$ vs. $\mathbf{A}(\mathbf{B}\mathbf{C}\mathbf{D})$
  - Heuristic: $\text{cost}(\mathbf{A}\mathbf{B}\mathbf{C}) := \text{columns}(\mathbf{A}) \cdot \text{rows}(\mathbf{C})$
  - Recursively calls reordering function with $N = 2$
  - Thus: linear (but sub-optimal) reordering of operations
Complex example: $A \times B \times C \times D$

- Instantiated as:
  \[ \text{Glue}<\text{Glue}<\text{Glue}<A,B,\text{op\_times}>,C,\text{op\_times}>,D,\text{op\_times}> \]

- Determines ”depth” of expression: $N = 3$
  (amount of operators with same precedence)

- Pattern matches specialized function for $N = 3$ which:
  - Decide $(ABC)D$ vs. $A(BCD)$
  - Heuristic: $\text{cost}(ABC) := \text{columns}(A) \times \text{rows}(C)$
  - Recursively calls reordering function with $N = 2$

- Thus: linear (but sub-optimal) reordering of operations

- Remember: all this at **compile-time**!
Complex example: $A*B*C*D$ (cont.)

- $(AB)C)D$ ✔
- $(A(BC))D$ ✔
- $A((BC)D)$ ✔
- $A(B(CD))$ ✔
- $(AB)(CD)$ ✗
Complex example: $A*B*C*D$ (cont.)

- $(AB)C)D$ ✓
- $(A(BC))D$ ✓
- $A((BC)D)$ ✓
- $A(B(CD))$ ✓
- $(AB)(CD)$ ✗
Complex example: $A*B*C*D$ (cont.)

- $((AB)C)D$ ✔ $\Rightarrow \mathcal{O}(n^3)$
- $(A(BC))D$ ✔ $\Rightarrow \mathcal{O}(n^3)$
- $A((BC)D)$ ✔ $\Rightarrow \mathcal{O}(n^3)$
- $A(B(CD))$ ✔ $\Rightarrow \mathcal{O}(n^3)$
- $(AB)(CD)$ ✗ $\Rightarrow \mathcal{O}(n^2)$
Complex example: $A*B*C*D$ (cont.)

- $((AB)C)D$  ✔  $\Rightarrow O(n^3)$
- $(A(BC))D$  ✔  $\Rightarrow O(n^3)$
- $A((BC)D)$  ✔  $\Rightarrow O(n^3)$
- $A(B(CD))$  ✔  $\Rightarrow O(n^3)$
- $(AB)(CD)$  ✗  $\Rightarrow O(n^2)$

<table>
<thead>
<tr>
<th>$n$</th>
<th>Armadillo</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>42</td>
</tr>
<tr>
<td>7</td>
<td>32</td>
<td>132</td>
</tr>
</tbody>
</table>
Other improvements

- \( A = A_1 + A_2 + \ldots + A_n \)

No temporary matrix besides output
Other improvements

- $A = A_1 + A_2 + \ldots + A_n$
  No temporary matrix besides output

- $y = A^{-1}x \rightarrow y = \text{solve}(A, x)$
  Faster and more accurate
Other improvements

- \( A = A_1 + A_2 + \ldots + A_n \)
  No temporary matrix besides output

- \( y = A^{-1}x \rightarrow y = \text{solve}(A, x) \)
  Faster and more accurate

- \( \text{inv(diag}(A)) \)
  Elementwise inverse
Other improvements

辽宁省 A = A₁ + A₂ + ... + Aₙ
No temporary matrix besides output

辽宁省 y = A⁻¹x → y = solve(A, x)
Faster and more accurate

辽宁省 inv(diag(A))
Elementwise inverse

辽宁省 as_scalar(r * X * q) with r ∈ ℝ¹×n, X ∈ ℝⁿ×ⁿ, q ∈ ℝⁿ×₁
Result of computation is 1 × 1 matrix
Other improvements

- $A = A_1 + A_2 + \ldots + A_n$
  No temporary matrix besides output

- $y = A^{-1}x \rightarrow y = \text{solve}(A, x)$
  Faster and more accurate

- `inv(diag(A))`
  Elementwise inverse

- `as_scalar(r \ast X \ast q)` with $r \in \mathbb{R}^{1\times n}$, $X \in \mathbb{R}^{n\times n}$, $q \in \mathbb{R}^{n\times 1}$
  Result of computation is $1 \times 1$ matrix

- Lots more!
Conclusion

✓ Easy to use for non-LA-experts
✓ Some optimization comes built-in
Conclusion

✓ Easy to use for non-LA-experts
✓ Some optimization comes built-in
✗ Cannot optimize all cases
✗ Debugging for more complex code can be a problem (C++)
Conclusion

✔ Easy to use for non-LA-experts
✔ Some optimization comes built-in
✘ Cannot optimize all cases
✘ Debugging for more complex code can be a problem (C++)

▶ Our recommendation: Use it! Don’t try to understand it.
Thanks! Questions?

Sources:

- Armadillo source code.
- *C++ Programming*, chapter Template Meta-Programming, Wikibooks