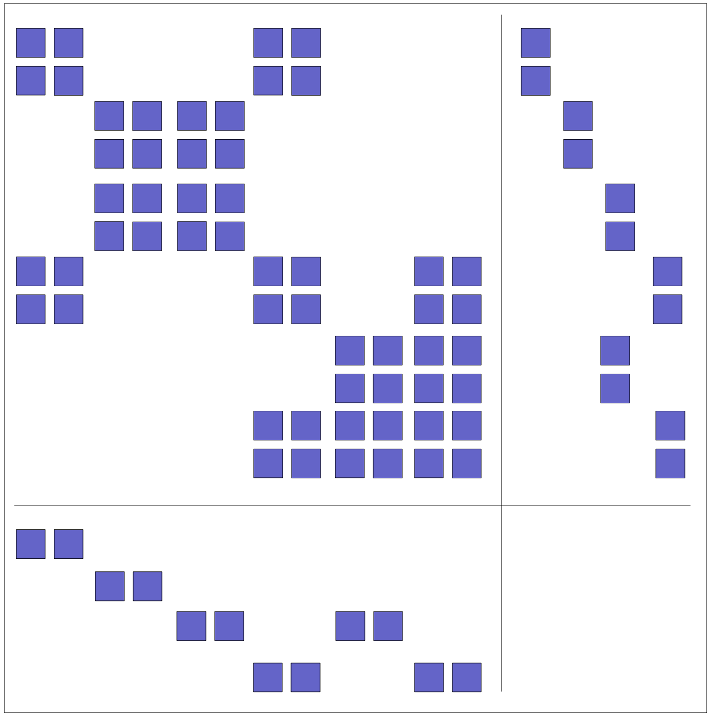
Template Metaprogramming in Finite Element Computations

Martin Weiser

Zuse Institute Berlin

DFG Research Center MATHEON



Workshop on Data Structures for Finite Element and Finite Volume Computations, 2008-02-29, Berlin

Contents

Template metaprogramming: why and what

Advantages of template metaprogramming

Heterogeneous data structures in KASKADE

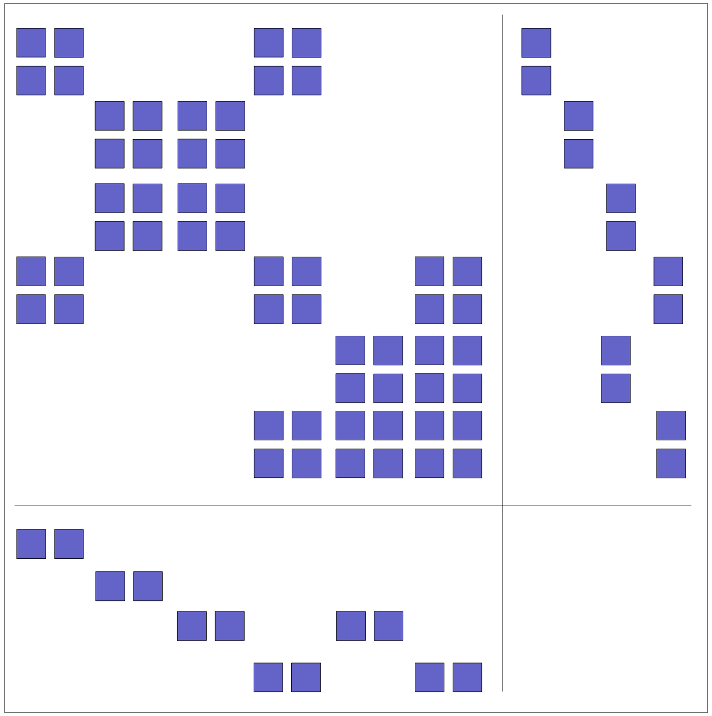
Warning: Whole talk is very C++ specific.

Why Template Metaprogramming?

MATHEON ZIB

Stokes problem

$$-\Delta u + \nabla p = f$$
$$\nabla \cdot u = 0$$



- heterogeneous block structure (dxd, 1xd, dx1)
- different possibilities of organizing storage
- either explicitly stored and checked metainformation or implicit assumptions throughout the code

Why Template Metaprogramming?

Storing FE coefficient vectors

- u and p live on different grid entities → no interleaving

  [[u],[p]]

- components of u have physical meaning in every point → grouping

  [[[u11,u12],[u21,u22],[u31,u32],...],[p1,p2,p3,...]]

- structured representation in std::vector's is highly inefficient (and looses structure)

```
vector<vector<vector<double>>,vector<vector<double>>>
```

components

variable's values

- 'flat' representation contains no structure

```
vector<double>
```

Why Template Metaprogramming?

MATHEON ZIB

Type safety and structured data

Compilers are much better at tedious error checking than programmers.
- array access checking
- array blocking

Aggressive compiler optimizations

Compilers are much better at low level optimizations than programmers.
- loop unrolling
- inlining
- constant folding
- dead code elimination

Code transformations

Computers (e.g. compilers) are much better at repetitive tasks than programmers.
- expression templates

What is Template Metaprogramming?

template metaprogramming ⟺ provide a maximum of static information to the compiler

Which information

- elementary types (float/double/complex)
- problem specific types
- options (bool)
- indices, ranges (int)
- ... collections and combinations thereof
- everything (!)

... everything (!)

```
template <int p, int i> struct isPrime {
  static int const value = (p==2) || (p%i) && isPrime<p,i-1>::value;
};
template <int p> struct isPrime<p,1> {
  static int const value = true;
};

template <int p> struct previous {
  static int const value = isPrime<p-1,p-2>::value? p-1:
                                                     previous<p-1>::value;
};
template <> struct previous<2> {
  static int const value = 1;
};

template <int p> struct is {
  static int const prime = is<previous<p>::value>::prime;
};
template <> struct is<1> {};

int main() { return is<23>::prime; }
```

[Unruh '95]

... everything (!)

```
prime.cc: In instantiation of ‘const int is<2>::prime’:
prime.cc:18:   instantiated from ‘const int is<3>::prime’
prime.cc:18:   instantiated from ‘const int is<5>::prime’
prime.cc:18:   instantiated from ‘const int is<7>::prime’
prime.cc:18:   instantiated from ‘const int is<11>::prime’
prime.cc:18:   instantiated from ‘const int is<13>::prime’
prime.cc:18:   instantiated from ‘const int is<17>::prime’
prime.cc:18:   instantiated from ‘const int is<19>::prime’
prime.cc:18:   instantiated from ‘const int is<23>::prime’
prime.cc:26:   instantiated from here
prime.cc:18: error: ‘prime’ is not a member of ‘is<1>’
prime.cc: In instantiation of ‘const int is<3>::prime’:
prime.cc:18:   instantiated from ‘const int is<5>::prime’
prime.cc:18:   instantiated from ‘const int is<7>::prime’
prime.cc:18:   instantiated from ‘const int is<11>::prime’
prime.cc:18:   instantiated from ‘const int is<13>::prime’
...
```

[GCC 4.2.1]

Advantages of Template Metaprogramming

Type Safety and Structured Data

MATHEON ZIB

FE coefficient vectors

Stokes: [[[u11,u12],[u21,u22],[u31,u32],...], [p1,p2,p3,...]]

```
vector<vector<vector<double>>,vector<vector<double>>>
```

components

variable's values

- coefficient arrays have the same type
- coefficients scattered throughout the memory
- shape constraints (same number of components for each value) is not enforced

Type Safety and Structured Data

FE coefficient vectors

Stokes: [[[u11,u12],[u21,u22],[u31,u32],...], [p1,p2,p3,...]]

Fixed-size vectors

```
template <class Scalar, int n>
class FixedVector {
  Scalar data[n];
public:
  Scalar& operator[](int i) { return data[i]; }
  void operator+=(FixedVector& v) {
    for (int i=0; i<n; ++i) data[i] += v[i];
  }
  ...
};
```

http://dune-project.org/

`Dune::FieldVector<Scalar,n>`

Type Safety and Structured Data

MATHEON ZIB

FE coefficient vectors

Stokes: [[[u11,u12],[u21,u22],[u31,u32],...], [p1,p2,p3,...]]

```
[ std::vector<FixedVector<double,n>>, std::vector<double> ]
```

- compiler knows and enforces that all u's have the same number of components
- coefficients of u and p are contiguous in memory
- different type of coefficient array for each variable

Type Safety and Structured Data

Heterogeneous Containers

```
template <class T1, class T2>
struct HeterogeneousVector {
  T1 data1;
  T2 data2;
};

template <class T1, class T2, int n>
struct VectorAccess {};

template <class T1, class T2>
struct VectorAccess<T1,T2,1> {
  typedef T1 type;
  type& at(HeterogeneousVector<T1,T2>& v) { return v.data1; }
};

template <class T1, class T2, int n>
VectorAccess<T1,T2,n>::type& at(HeterogeneousVector<T1,T2>& v) {
  return VectorAccess<T1,T2,n>::at(v);
}
```

Type Safety and Structured Data

MATHEON

FE coefficient vectors

Stokes: [[[u11,u12],[u21,u22],[u31,u32],...], [p1,p2,p3,...]]

```
HeterogeneousVector< std::vector<FixedVector<double,2> >,
                     std::vector<double> >
```

- compiler knows and enforces that all u's have the same number of components
- coefficients of u and p are contiguous in memory
- different type of coefficient array for each variable

boost C++ LIBRARIES

```
boost::fusion::vector<T1,T2>
```

Metaprogramming: Heterogeneous Loops

Dynamic polymorphism

```
struct Base {
  virtual void doSomething() = 0;
};

std::vector<Base*> container;

std::for_each(container.begin(),container.end(),
              std::mem_fun(&Base::doSomething));
```

```
template <class Base>
struct mem_fun_t {
  mem_fun_t( (Base::*p_)() ): p(p_) {}
  void operator()(Base* base) { (base->*p)(); }
  (Base::*p)();
};
template <class Base>
mem_fun_t<Base> mem_fun( (Base::*p)() ) {
  return mem_fun_t<Base>(p);
}
```

Metaprogramming: Heterogeneous Loops

Static polymorphism

```
struct T1 {
  void doSomething();
};

struct T2 {
  void doSomething();
};

boost::fusion::vector<T1,T2> container;

boost::fusion::for_each(container,Functor());


struct Functor {
  template <class T>
  void operator()(T& data) {
    data.doSomething();
  }
};
```

Loop Unrolling

```
double dynamic_sum(double* x,
                   int n) {
  double sum = 0;
  for (int i=0; i<n; ++i)
    sum += x[i];
  return sum;
}
```

```
template <int n>
double static_sum(double* x) {
  double sum = 0;
  for (int i=0; i<n; ++i)
    sum += x[i];
  return sum;
}
```

Loop Unrolling

```
.LFB508:
        pushl   %ebp
.LCFI0:
        movl    %esp, %ebp
.LCFI1:
        movl    12(%ebp), %edx
        movl    8(%ebp), %ecx
        testl   %edx, %edx
        jle     .L9
        fldz
        xorl    %eax, %eax
        .p2align 4,,7
.L5:
        faddl   (%ecx,%eax,8)
        addl    $1, %eax
        cmpl    %edx, %eax
        jne     .L5
        popl    %ebp
        ret
.L9:
        popl    %ebp
        fldz
        ret
```

static n=3

```
.LFB513:
        pushl   %ebp
.LCFI2:
        movl    %esp, %ebp
.LCFI3:
        movl    8(%ebp), %eax
        fldz
        popl    %ebp
        faddl   (%eax)
        faddl   8(%eax)
        faddl   16(%eax)
        ret
```

[g++ -O3 -S]

Inlining & Dead Code Elimination

MATHEON ZIB

```
struct Integrand {
  virtual double f(double x)=0;
};

struct Constant: public Integrand {
  virtual double f(double x) {
    return 1; }
};

double integral(Integrand& f,
                int n) {
  double sum = 0;

  for (int i=0; i<n; ++i)
    sum += f.f((i+0.5)/n);

  return sum/n;
}

...
Constant f;
integral(f,1000000000);
...
```

```
struct Constant {
  double f(double x) {
    return 1; }
};

template <class Integrand>
double Integral(Integrand& f,
                int n) {
  double sum = 0;

  for (int i=0; i<n; ++i)
    sum += f.f((i+0.5)/n);

  return sum/n;
}

...
Constant f;
integral(f,1000000000);
...
```

Inlining & Dead Code Elimination

```
_ZN8ConstantclEd:
.LFB2:
        movsd  .LC0(%rip), %xmm0
        ret

main:
.LFB4:
        pushq     %rbp
.LCFI4:
        pushq     %rbx
.LCFI5:
        movl      $1, %ebx
        subq      $40, %rsp
.LCFI6:
        leaq      16(%rsp), %rbp
        movq      $_ZTV8Constant+16, 16(%rsp)
        movsd     .LC3(%rip), %xmm0
        movq      %rbp, %rdi
        call      *_ZTV8Constant+16(%rip)
        movapd    %xmm0, %xmm1
        addsd     .LC1(%rip), %xmm1
        .p2align 4,,7
.L13:
        cvtsi2sd %ebx, %xmm0
        movq 16(%rsp), %rax
        movsd     %xmm1, (%rsp)
        movq %rbp, %rdi
        addl $1, %ebx
        addsd     .LC2(%rip), %xmm0
        divsd     .LC4(%rip), %xmm0
        call *(%rax)
        movsd     (%rsp), %xmm1
        cmpl $1000000000, %ebx
        addsd     %xmm0, %xmm1
        jne  .L13
        divsd     .LC4(%rip), %xmm1
        addq      $40, %rsp
        popq      %rbx
        popq      %rbp
        cvttsd2si %xmm1, %eax
        ret
```

```
main:
.LFB4:
    movsd   .LC0(%rip), %xmm0
    movl    $1, %eax
    movapd  %xmm0, %xmm1
    .p2align 4,,7
.L2:
    addl    $1, %eax
    addsd   %xmm1, %xmm0
    cmpl    $1000000000, %eax
    jne .L2
    divsd   .LC1(%rip), %xmm0
    cvttsd2si  %xmm0, %eax
    ret
```

[g++ -O3 -S]

Timings

	dynamic	11.85s
	static	1.48s

Expression Templates: Operator Overloading

MATHEON ZIB

Vector addition

$$c := a + b$$

```
vector a,b,c;
c = a;
daxpy(a.size,1.0,b,1,c,1);
```

Classic operator overloading

```
vector operator+(vector& a, vector& b) {
  vector tmp = a;
  for (int i=0; i<a.size; ++i)
    tmp[i] += b[i];
  return tmp;
}
```

$$c := a + b$$

```
vector a, b, c;
c = a+b;
```

⇒

```
vector a,b,c;
vector tmp = a;
tmp += b;
c = tmp;
```

Expression Templates: Operator Overloading

MATHEON ZIB

```
template <class Op1, class Op2>
struct OpAdd {
  OpAdd(Op1& op1_, Op2& op2_): op1(op1_), op2(op2_) {}
  double operator[](int i) { return op1[i]+op2[i]; }
  Op1& op1;
  Op2& op2;
};

template<class Op1, class Op2>
struct OpAdd<Op1,Op2> operator+(Op1& op1, Op2& op2) {
  return OpAdd<Op1,Op2>(op1,op2);
}

template <class Expression>
void vector::operator=(Expression& ex) {
  for (int i=0; i<size; ++i)
    (*this)[i] = ex[i];
}
```

[Veldhuizen '95]

Expression Templates: Operator Overloading

```
vector a, b, c;
c = a+b;
```

inlining

```
vector a,b,c;
for (int i=0; i<c.size; ++i)
  c[i] = a[i]+b[i];
```

Intermediate expression type: `OpAdd<vector,vector>`

```
vector a,b,c,d;
d = a+b-c;
```

inlining

```
vector a,b,c,d;
for (int i=0; i<c.size; ++i)
  d[i] = (a[i]+b[i])-c[i];
```

Intermediate expression type: `OpSub<OpAdd<vector,vector>,vector>`

Expression Templates: Algorithmic Differentiation

MATHEON ZIB

```
template <int n>
struct Var {
  double value() { return val; }
  template <int i>
  double diff(Var<i>&) { return i==n? 1: 0; }
  double val;
};

template <class Op1, class Op2>
struct OpAdd {
  OpAdd(Op1& op1_, Op2& op2_): op1(op1_), op2(op2_) {}
  double value() { return op1.value()+op2.value(); }
  template <int i>
  double diff(Var<i>& x) { return op1.diff(x)+op2.diff(x); }
  Op1& op1;
  Op2& op2;
};
```

Expression Templates: Algorithmic Differentiation

MATHEON ZIB

Code

```
Var<0> x;
Var<1> y;
Var<2> z;

std::cout << "d(x+y)/dx: " << (x+y).diff(x) << '\n'
          << "d(x+y)/dz: " << (x+y).diff(z) << '\n';
```

Output

```
d(x+y)/dx: 1
d(x+y)/dz: 0
```

Heterogeneous Data Structures in KASKADE

FEM Systems in KASKADE

MATHEON

Stokes problem

$$-\Delta u + \nabla p = f$$
$$\nabla \cdot u = 0$$

Weak formulation

$$u \in H_0^1(\Omega)^2,\ p \in L^2(\Omega):$$

$$\begin{array}{lclcll} \langle \nabla u, \nabla v\rangle & + & \langle \nabla p, v\rangle & = & \langle f, v\rangle & \forall v \in H_0^1(\Omega)^2 \\ \langle \nabla \cdot u, w\rangle & & & = & 0 & \forall w \in L^2(\Omega) \end{array}$$

Galerkin discretization

$$u \in \mathbb{P}_2(\mathcal{T})^2,\ p \in \mathbb{P}_1(\mathcal{T}):$$

$$\begin{array}{lclcll} \langle \nabla u, \nabla v\rangle & + & \langle \nabla p, v\rangle & = & \langle f, v\rangle & \forall v \in \mathbb{P}_2(\mathcal{T})^2 \\ \langle \nabla \cdot u, w\rangle & & & = & 0 & \forall w \in \mathbb{P}_1(\mathcal{T}) \end{array}$$

FE Coefficient Vectors

Ansatz & test spaces

```
P2Space p2Space(...);
P1Space p1Space(...);
typedef boost::fusion::vector<P2Space*,P1Space*> Spaces;
Spaces spaces(&p2Space,&p1Space);
```

Variables

```
template <int id_, int components_, int space_>
struct Variable {
  static int const id = id_;
  static int const components = components_;
  static int const space = space_;
};

typedef boost::fusion::vector< Variable<0,2,0>,  // u in P2
                               Variable<1,1,1>   // p in P1
         > VariableList;
```

FE Coefficient Vectors

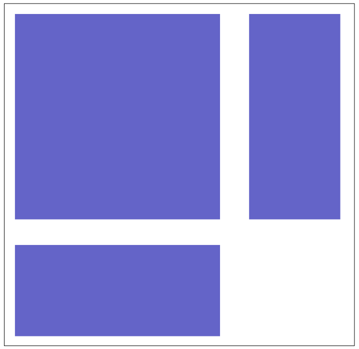
Coefficient Vectors

```
using namespace boost::fusion;

template <class Spaces>
struct CreateFEFunction {
  CreateFEFunction(Spaces& spaces_): spaces(spaces) {}

  template <class Variable>
  struct result {
    typedef typename result_of::at_c<Spaces,Variable::space>::type Space;
    typedef typename Space::template Element<Variable::components>::type type;
  };

  template <class Variable>
  typename result<Variable>::type operator()(Variable& v) {
    return typename result<Variable>::type(at_c<Variable::space>(spaces));
  }

  Spaces& spaces;
};

typename result_of::transform<VariableList,CreateFEFunction<Spaces> >::type
functions = transform(VariableList(),CreateFEFunction<Spaces>(spaces));
```

Block Stiffness Matrices

Heterogeneous block structure

$$\begin{aligned} \langle \nabla u, \nabla v \rangle \quad + \quad \langle \nabla p, v \rangle &= \langle f, v \rangle \\ \langle \nabla \cdot u, w \rangle &= 0 \end{aligned}$$



block (0,0): 2x2 entries
block (0,1): 2x1 entries
block (1,0): 1x2 entries
block (1,1): inexistent

Structure description

```
struct Stokes {
  template <int row, int col>
  struct D2 {
    static bool const present = row==0 || col==0;
  };
};
```

Block Stiffness Matrices

MATHEON

Filtering out inexistent blocks

```
using namespace boost::fusion;

template <class Problem>
struct IsPresent {
  template <class Block>
  struct apply {
    static int const row = at_c<0>(Block)::type::id;
    static int const col = at_c<1>(Block)::type::id;
    static bool const value = Problem::template D2<row,col>::present;
  };
};


typedef typename result_of::filter_if<
      typename result_of::outer_product<VariableList,VariableList>::type,
      IsPresent<Problem> >       PresentBlocks;
```

Block Stiffness Matrices

MATHEON

Creating sparse matrices

```
struct CreateBlock {
  template <class Block>
  struct result {
    static int const rComps = result_of::at_c<0,Block>::components;
    static int const cComps = result_of::at_c<1,Block>::components;
    typedef Dune::BCRSMatrix<Dune::FieldMatrix<Scalar,rComps,cComps> > type;
  };

  template <class Block>
  typename result<Block>::type operator()(Block b) {
    return typename result<Block>::type();
  }
};

ComplicatedType heterogeneousBlockMatrix
        = transform(PresentBlocks(),CreateBlock());
```

Conclusion

MATHEON

Template metaprogramming is a viable means to

- raise the abstraction level
- increase level of error checking
- allow more compiler optimizations

in FE computations.

Drawbacks are

- longer compile times
- cryptic error messages (may improve with coming C++ 0x standard)
- lengthy boilerplate code (will improve with coming C++ 0x standard)

Experience: Once the code compiles, it's semantically correct.