

CFL: A Domain-Specific Language for Simplifying Integration Kernels

Daniel Arndt Guido Kanschat

Heidelberg University

Fast high order DG methods for future architectures

Interdisciplinary Center for Scientific Computing



03. - 05. July 2017



Contents

- 1 Introduction
- 2 CFL
- 3 Code Examples
- 4 Summary

Assembly

- Major part in FE calculations:
Matrix-vector product

$$y = A \cdot x$$

defined by some bilinear form.

- Matrix-based or matrix-free

Example: Laplace problem

Find $u \in H^1(\Omega)$, s.t.

$$(\nabla u, \nabla v) = (f, v) \quad \forall v \in H_0^1(\Omega).$$

Assembly in deal.II

Assembly approaches in deal.II

- Manually write the assembly of the matrix

Manual Matrix Assembly - Step-6

```
1  typename DoFHandler<dim>::active_cell_iterator
   cell = dof_handler.begin_active(),
3  endc = dof_handler.end();
   for (; cell!=endc; ++cell)
5      {
       cell_matrix = 0;
7       fe_values.reinit (cell);
       for (unsigned int q_index=0; q_index<n_q_points; ++q_index)
9           for (unsigned int i=0; i<dofs_per_cell; ++i)
               for (unsigned int j=0; j<dofs_per_cell; ++j)
11                  cell_matrix(i,j) += (current_coefficient *
                                           fe_values.shape_grad(i, q_index) *
13                                           fe_values.shape_grad(j, q_index) *
                                           fe_values.JxW(q_index));
15
       cell->get_dof_indices (local_dof_indices);
17       constraints.distribute_local_to_global (cell_matrix,
                                               local_dof_indices,
19                                               system_matrix);
   }
21
```

Assembly in deal.II

Assembly approaches in deal.II

- Manually write the assembly of the matrix
- MatrixCreator

- `create_mass_matrix:` $m_{ij} = \int_{\Omega} a(x) \phi_i(x) \phi_j(x) dx$
- `create_laplace_matrix:` $m_{ij} = \int_{\Omega} a(x) \nabla \phi_i(x) \nabla \phi_j(x) dx$
- `create_boundary_matrix:` $m_{ij} = \int_{\partial\Omega} a(x) \phi_i(x) \phi_j(x) dx$

Assembly in deal.II

Assembly approaches in deal.II

- Manually write the assembly of the matrix
- MatrixCreator
- MeshWorker

MeshWorker

```
1  template <int dim>
2  class MatrixIntegrator : public MeshWorker::LocalIntegrator <dim>
3  {
4  public:
5      void cell (MeshWorker::DoFInfo<dim> &dinfo ,
6                typename MeshWorker::IntegrationInfo <dim> &info) const;
7
8      void boundary (MeshWorker::DoFInfo<dim> &dinfo ,
9                    typename MeshWorker::IntegrationInfo <dim> &info) const;
10
11     void face (MeshWorker::DoFInfo<dim> &dinfo1 ,
12              MeshWorker::DoFInfo<dim> &dinfo2 ,
13              typename MeshWorker::IntegrationInfo <dim> &info1 ,
14              typename MeshWorker::IntegrationInfo <dim> &info2) const;
15 };
16
```


Assembly in deal.II

Assembly approaches in deal.II

- Manually write the assembly of the matrix
- MatrixCreator
- MeshWorker
- MatrixFree

MatrixFree - Step-37

```
1  template <int dim, int fe_degree, typename number>
   void
3  LaplaceOperator<dim, fe_degree, number>::local_apply
   (const MatrixFree<dim, number> &data,
5   LinearAlgebra::distributed::Vector<number> &dst,
   const LinearAlgebra::distributed::Vector<number> &src,
7   const std::pair<unsigned int, unsigned int> &cell_range) const
   {
9   FEEvaluation<dim, fe_degree, fe_degree+1, 1, number> phi (data);
   for (unsigned int cell=cell_range.first; cell<cell_range.second; ++cell)
11  {
       phi.reinit (cell);
13     phi.read_dof_values(src);
       phi.evaluate (false, true);
15     for (unsigned int q=0; q<phi.n_q_points; ++q)
           phi.submit_gradient (coefficient(cell, q) *
17                               phi.get_gradient(q), q);
       phi.integrate (false, true);
19     phi.distribute_local_to_global (dst);
   }
21 }
```

Assembly in deal.II

Assembly approaches in deal.II

- Manually write the assembly of the matrix
- MatrixCreator
- MeshWorker
- MatrixFree
- MatrixFreeOperators
 - Base
 - Mass
 - Laplace

Idea:

- Only overload write (cell) operator (apply_add),
- MatrixFreeOperators takes care of the rest

Similar to MeshWorker.

Assembly in deal.II

Assembly approaches in deal.II

- Manually write the assembly of the matrix
- `MatrixCreator`
- `MeshWorker`
- `MatrixFree`
- `MatrixFreeOperators`

Summary

- Each good for specific purposes, but not generic.
- Define bilinear form on low level rather than high level.

⇒ CFL

Contents

- 1 Introduction
- 2 CFL**
- 3 Code Examples
- 4 Summary

CFL

Idea:

- Have a generic language to express bilinear form
- Address multiple back ends (in deal.II or other PDE frameworks)
- Write bilinear form in an intuitive way as you learn them in a PDE class.
- Provide compile-time checks for compatibility of the used objects.
- Address the back ends as efficient as possible; minimal overhead.

CFL - Ingredients

Ingredients:

- `FEData`: representation of the used finite elements
- `FEDatas`: container for `FEData` objects
- `Form`: representation of a bilinear form
- `Forms`: container for `Form` objects

Fed into an object `MatrixFreeIntegrator` that implements a `vmult`.

CFL - Ingredients - FEData

```
1  template <template <int , int> class FiniteElementType , int fe_degree ,
      int n_components , int dim , unsigned int fe_no ,
3      unsigned int max_fe_degree , typename Number = double>
      class FEData final
5  {
      public:
7      using FEEvaluationType =
          typename dealii::FEEvaluation<dim , fe_degree , max_fe_degree + 1 ,
9              n_components , Number>;
          // store template arguments
11     [...]

13     const FiniteElementType<dim , dim> &fe ;
        std::shared_ptr<FEEvaluationType> fe_evaluation = nullptr ;

15     explicit FEData(const FiniteElementType<dim , dim>& fe_) : fe(fe_)
17     {
          // check compatibility
19     [...]
        }
21 };
```


CFL - Ingredients - FEDatas

Use variadic templates to define a storage container

```
1  template <class FEData>
   class FEDatas<FEData>;
3
   template <class FEData, typename... Types>
5  class FEDatas<FEData, Types...> : public FEDatas<Types...>
   {
7    public:
       // operators to access the top-level FEData object
9     // and the FEEvaluation object
       [...]
11
       // store integration and evaluation flags
13     [...]
15
       FEData fe_data;
   }
```

CFL - Ingredients - Form

```
1  template <class Test, class Expr, FormKind kind_of_form, typename NumberType>
2  class Form final
3  {
4  public:
5      // access to the template parameters
6      // and integration flags from the Test object
7      [... ]
8
9      template <class FEEvaluation>
10     void evaluate(FEEvaluation& phi, unsigned int q) const;
11
12     template <class FEEvaluation>
13     static void integrate(FEEvaluation& phi);
14
15     template <class FEEvaluation>
16     auto value(FEEvaluation& phi, unsigned int q) const;
17
18     template <class FEEvaluation, typename ValueType>
19     static void submit(FEEvaluation& phi, unsigned int q,
20                       const ValueType& value);
21 }
```

CFL - Ingredients - Forms

Use variadic templates to define a storage container

```
1  template <typename FormType>
   class Forms<FormType>;
3
   template <typename FormType, typename... Types>
5  class Forms<FormType, Types...> : public Forms<Types...>
   {
7  public:
   // similar interface as the Form class recursively defined, e.g.
9  template <class FEEvaluation>
   static void
11  integrate(FEEvaluation& phi)
   {
13     phi.template integrate<fe_number>(integrate_value, integrate_gradient);
     Forms<Types...>::integrate(phi);
15  }
   }
```

CFL - Ingredients - Test Functions

Use CRTP-style declarations

```
1  template <int rank, int dim, unsigned int idx>
2  class TestFunction final
3      : public TestFunctionBase<TestFunction<rank, dim, idx>>
4  {
5      public:
6          using Base = TestFunctionBase<TestFunction<rank, dim, idx>>;
7          static constexpr IntegrationFlags integration_flags{ true, false,
8                                                                false, false };
9
10         template <class FEEvaluation, typename ValueType>
11             static void
12             submit(FEEvaluation& phi, unsigned int q, const ValueType& value);
13     };
```

CFL - Ingredients - Ansatz Functions

Use CRTP-style declarations

```
1 template <int rank, int dim, unsigned int idx>
2 class FEFunctor final : public FEFunctorBase<FEFunction<rank, dim, idx>>
3 {
4     public:
5         using Base = FEFunctorBase<FEFunction<rank, dim, idx>>;
6         // inherit constructors
7         using Base::Base;
8
9         template <class FEDatas>
10        auto value(const FEDatas& phi, unsigned int q) const;
11
12        template <class FEEvaluation>
13        static void set_evaluation_flags(FEEvaluation& phi);
14 };
```

Linear combinations and products allowed.

CFL - Ingredients - MatrixFreeIntegrator - I

```
template <int dim, typename VectorType, class FORM, class FEDatas>
2 MatrixFreeIntegrator <[ ... ]>::local_apply
  (const dealii::MatrixFree<dim, Number>& data_, VectorType& dst,
4   const VectorType& src,
   const std::pair<unsigned int, unsigned int>& cell_range) const
6 {
   for (unsigned int cell = cell_range.first; cell < cell_range.second; ++cell)
8     {
       fe_datas->reinit(cell);
10      fe_datas->read_dof_values(src);
       do_operation_on_cell(*fe_datas, cell);
12      fe_datas->distribute_local_to_global(dst);
     }
14 }
```

CFL - Ingredients - MatrixFreeIntegrator - II

```
1  template <int dim, typename VectorType, class FORM, class FEDatas>
2  template <class FEEvaluation>
3  void MatrixFreeIntegrator<[...]> do_operation_on_cell
4  (FEEvaluation& phi, const unsigned int cell) const
5  {
6  phi.evaluate();
7  constexpr unsigned int n_q_points = FEEvaluation::get_n_q_points();
8
9  for (unsigned int q = 0; q < n_q_points; ++q)
10     form->evaluate(phi, q);
11
12 phi.integrate();
13 }
14
```

Contents

- 1 Introduction
- 2 CFL
- 3 Code Examples**
- 4 Summary

Code Examples - Laplace

Laplace problem

Find $u \in H^1(\Omega)$, s.t.

$$(\nabla u, \nabla v) = (f, v) \quad \forall v \in H_0^1(\Omega)$$

Solve using a

- CG solver
- with a Multigrid preconditioner
- with a Cheyshev smoother.

Code Examples - Laplace

```
1   FE_Q<dim> fe_u(degree);

3   FEData<FE_Q, degree, 1, dim, 0, degree, double> fedata_system(fe_u);
   FEData<FE_Q, degree, 1, dim, 0, degree, float> fedata_level(fe_u);

5

7   CFL::dealii::MatrixFree::TestFunction<0, dim, 0> v;
   auto Dv = grad(v);
   CFL::dealii::MatrixFree::FEFunction<0, dim, 0> u("u");
9   auto Du = grad(u);
   auto f_system = CFL::form(Du, Dv);

11

13  LaplaceProblem<dim,
        decltype(fe_data_system),
        decltype(fe_data_level),
15  decltype(f_system)>
    laplace_problem(fe_datas_system, fe_datas_level, f_system);
17  laplace_problem.run();
```

Code Examples - Schloegl

Schloegl problem

Find $u \in H^1(\Omega)$, s.t.

$$(\nabla u, \nabla v) + (3u^3 + \alpha u, v) = 0 \quad \forall v \in H_0^1(\Omega)$$

Newton iteration

Find $e \in H_0^1(\Omega)$, s.t.

$$(\nabla e, \nabla v) + (3u_{old}^2 e + \alpha e, v) = -(\nabla u, \nabla v) - (3u_{old}^3 + \alpha u_{old}, v)$$

for all $v \in H_0^1(\Omega)$ and $u_{new} = e + u_{old}$.

Features:

- nonlinear
- Multigrid

Code Examples - Schloegl

```
1 FE_Q<dim> fe_u(degree);

3 FEData<FE_Q, degree, 1, dim, 0, degree, double> fedata_e_system(fe_u);
  FEData<FE_Q, degree, 1, dim, 1, degree, double> fedata_u(fe_u);
5 auto fe_datas_system = (fedata_e_system, fedata_u);
  FEData<FE_Q, degree, 1, dim, 0, degree, float> fedata_e_level(fe_u);
7 FEData<FE_Q, degree, 1, dim, 1, degree, float> fedata_u_level(fe_u);
  auto fe_datas_level = (fedata_e_level, fedata_u_level);
9
  CFL::dealii::MatrixFree::TestFunction<0, dim, 0> v;
11 auto Dv = grad(v);
  CFL::dealii::MatrixFree::FEFunction<0, dim, 0> e("e");
13 auto De = grad(e);
  CFL::dealii::MatrixFree::FEFunction<0, dim, 1> u("u");
15 auto Du = grad(u);

17 auto f1 = CFL::form(De, Dv);
  auto f2 = CFL::form(3 * u * u * e - alpha * e, v);
19 auto f = f1 + f2;

21 auto rhs = CFL::form(-Du, Dv) + CFL::form(-u * u * u + alpha * u, v);
```

Code Examples - Stokes

Stokes problem

Find $(\mathbf{u}, p) \in \mathbf{V} \times Q$, s.t.

$$(\boldsymbol{\varepsilon}(\mathbf{u}), \boldsymbol{\varepsilon}(\mathbf{v})) - (p, \nabla \cdot \mathbf{v}) = (\mathbf{f}, \mathbf{v}) \quad \forall \mathbf{v} \in \mathbf{V}$$

$$(\nabla \cdot \mathbf{u}, q) = 0 \quad \forall q \in Q$$

where

$$\boldsymbol{\varepsilon}(\mathbf{u}) := \frac{\nabla \mathbf{u} + (\nabla \mathbf{u})^T}{2}.$$

Code Examples - Stokes

```
1 FESystem<dim> fe_u (FE_Q<dim>(degree + 1), dim);
  FE_Q<dim> fe_p (degree);
3
  FEData<FESystem, degree, dim, dim, 0, degree> fedata1(fe_u);
5  FEData<FE_Q, degree, 1, dim, 1, degree> fedata2(fe_p);
  auto fe_datas = (fedata1, fedata2);
7
  FEFunctor<1, dim, 0> u("u");
9  auto Divu = div(u);
  FESymmetricGradient<2, dim, 0> Du(R"((\nabla+\nabla^T)u)");
11 FEFunctor<0, dim, 1> p("p");
  FELiftDivergence<decltype(p)> Liftp(p);
13 TestFunction<1, dim, 0> v;
  TestFunction<0, dim, 1> q;
15 auto Dv = grad(v);

17 auto f1 = form(Du + Liftp, Dv);
  auto f2 = form(Divu, q);
19 auto f = f1 + f2;
```

Code Examples - Laplace (DG)

Find $u \in V_h$, s.t.

$$\begin{aligned}
 & \sum_{K \in \mathbb{T}_h} (\nabla u, \nabla v)_K \\
 & + \sum_{F \in F_h^i} \left\{ -2(\{\{\nabla u\}\}, \{\{vn\}\})_F - 2(\{\{\nabla v\}\}, \{\{un\}\})_F \right. \\
 & \quad \left. + 4\sigma_F(\{\{un\}\}, \{\{vn\}\})_F \right\} \\
 & + \sum_{F \in F_h^b} \left\{ 2\sigma_F(u, v)_F - (\partial_n u, v)_F - (\partial_n v, u)_F \right\} \\
 & = (f, v)_\Omega + \sum_{F \in F_h^b} \left\{ 2\sigma_F(u^D, v)_F - (\partial_n v, u^D)_F \right\} \quad \forall v \in V_h.
 \end{aligned}$$

Code Examples - Laplace (DG) - I

```
1  FE_DGQ<dim> fe_u(degree);

3  FEData<FE_DGQ, degree, 1, dim, 0, degree> fedata1(fe_u);
   FEDataFace<FE_DGQ, degree, 1, dim, 0, degree> fedata_face1(fe_u);
5  auto fe_datas = (fedata1, fedata_face1);

7  TestFunction<0, 1, 0> v;
   auto Dv = grad(v);
9  TestFunctionInteriorFace<0, 1, 0> v_p;
   TestFunctionExteriorFace<0, 1, 0> v_m;
11 TestNormalGradientInteriorFace<0, 1, 0> Dnv_p;
    TestNormalGradientExteriorFace<0, 1, 0> Dnv_m;

13
   FEFunction<0, 1, 0> u("u");
15 auto Du = grad(u);
   FEFunctionInteriorFace<0, 1, 0> u_p("u+");
17 FEFunctionExteriorFace<0, 1, 0> u_m("u-");
   FENormalGradientInteriorFace<0, 1, 0> Dnu_p("Dn_u+");
19 FENormalGradientExteriorFace<0, 1, 0> Dnu_m("Dn_u-");
```


Code Examples - Laplace (DG) - II

```
20  auto cell = form(Du, Dv);

22  auto jump = u_p - u_m;
    auto Dn_jump = Dnu_p - Dnu_m;

24

    auto face = -face_form(.5 * jump, Dnv_p)
                +face_form(.5 * jump, Dnv_m)
                -face_form(-jump + .5 * Dn_jump, v_p)
28                +face_form(-jump + .5 * Dn_jump, v_m);

30  auto boundary = boundary_form(2. * u_p - Dnu_p, v_p);
                -boundary_form(u_p, Dnv_p);

32

    auto f = cell + face + boundary;
```

Contents

- 1 Introduction
- 2 CFL
- 3 Code Examples
- 4 Summary**

Summary

Features:

- Supports deal.II
- Write local operators like bilinear forms.
- Compile-time checks
- Get the same performance as code written using the (MatrixFree) back end; negligible overhead.

Future:

- Support more back ends.
- Provide more flexibility in the operators defined.
- Use with *Neo* (Joint work with Pablo Lucero, Julius Witte; currently mostly MeshWorker based)