CFL: A Domain-Specific Language for Simplifying Integration Kernels

Daniel Arndt Guido Kanschat

Heidelberg University

Sixth deal.II Users and Developers Workshop

Interdisciplinary Center for Scientific Computing

July 24, 2018

Contents

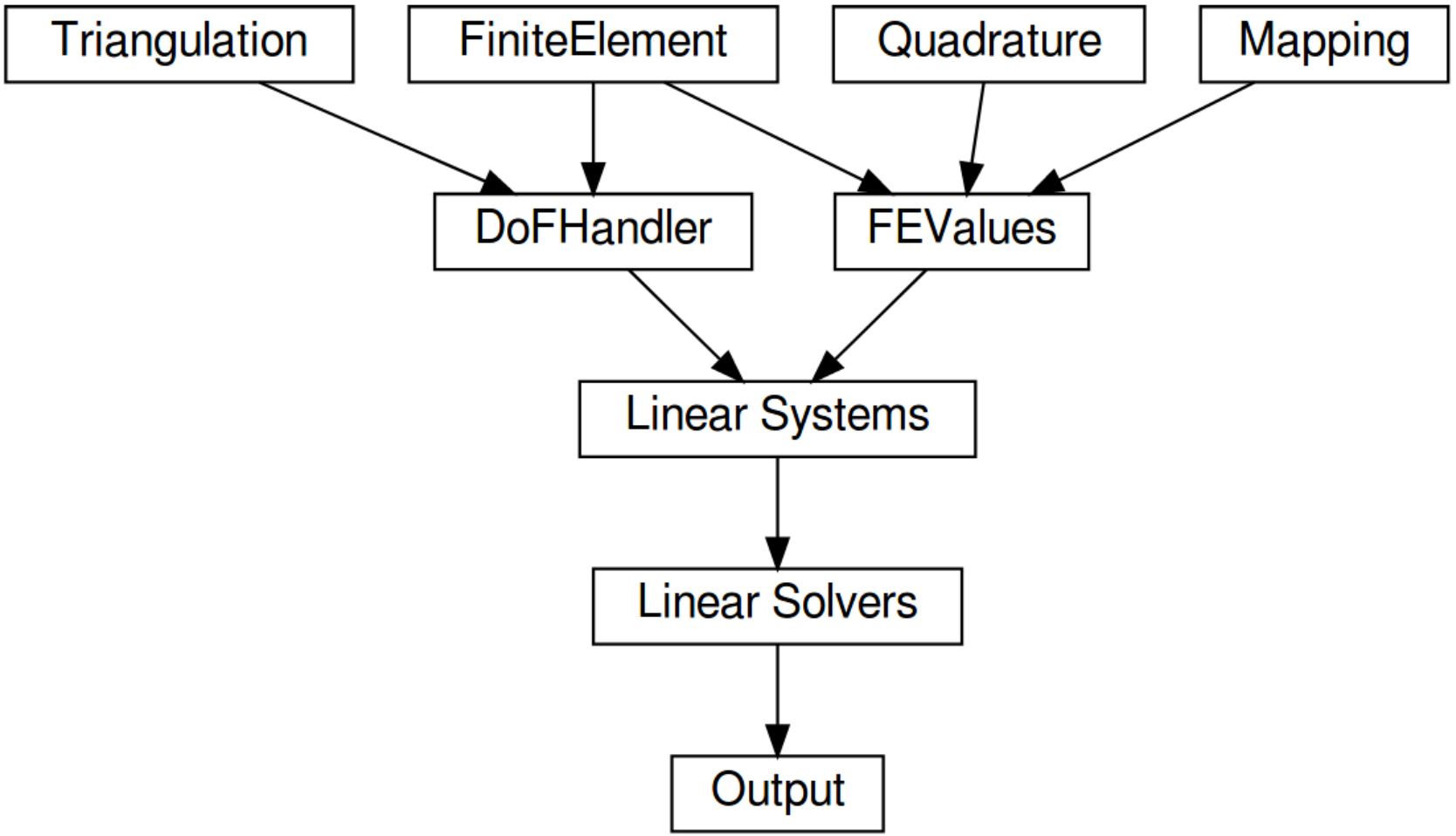
1 Introduction

2 CFL

3 Code Examples

4 Summary

Assembly


Triangulation
FiniteElement
Quadrature
Mapping
DoFHandler
FEValues
Linear Systems
Linear Solvers
Output

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

```
for (const auto &cell : dof_handler.active_cell_iterators())
  {
    cell_matrix = 0.;
    fe_values.reinit (cell);
    for (unsigned int q_index=0; q_index<n_q_points; ++q_index)
      for (unsigned int i=0; i<dofs_per_cell; ++i)
        for (unsigned int j=0; j<dofs_per_cell; ++j)
          cell_matrix(i,j) += (current_coefficient *
                               fe_values.shape_grad(i,q_index) *
                               fe_values.shape_grad(j,q_index) *
                               fe_values.JxW(q_index));

    cell->get_dof_indices (local_dof_indices);
    constraints.distribute_local_to_global (cell_matrix,
                                            local_dof_indices,
                                            system_matrix);
  }
```

Assembly in deal.II

Assembly approaches in `deal.II`

- Manually write the assembly of the matrix
- `MatrixCreator`
  - `create_mass_matrix`: $m_{ij} = \int_\Omega a(x)\phi_i(x)\phi_j(x)\,\mathrm{d}x$
  - `create_laplace_matrix`: $m_{ij} = \int_\Omega a(x)\nabla\phi_i(x)\nabla\phi_j(x)\,\mathrm{d}x$
  - `create_boundary_matrix`: $m_{ij} = \int_{\partial\Omega} a(x)\phi_i(x)\phi_j(x)\,\mathrm{d}x$

Assembly in deal.II

Assembly approaches in `deal.II`

- Manually write the assembly of the matrix
- `MatrixCreator`
- `MeshWorker/MeshLoop`

MeshWorker

```
template <int dim>
class MatrixIntegrator : public MeshWorker::LocalIntegrator<dim>
{
public:
  void cell(MeshWorker::DoFInfo<dim> &dinfo,
            typename MeshWorker::IntegrationInfo<dim> &info) const;

  void boundary(MeshWorker::DoFInfo<dim> &dinfo,
                typename MeshWorker::IntegrationInfo<dim> &info) const;

  void face(MeshWorker::DoFInfo<dim> &dinfo1,
            MeshWorker::DoFInfo<dim> &dinfo2,
            typename MeshWorker::IntegrationInfo<dim> &info1,
            typename MeshWorker::IntegrationInfo<dim> &info2) const;
};
```

Assembly in deal.II

Assembly approaches in `deal.II`

- Manually write the assembly of the matrix
- `MatrixCreator`
- `MeshWorker/MeshLoop`
- `MatrixFree`

MatrixFree - Step-37

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

Assembly in deal.II

Assembly approaches in `deal.II`

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

Summary

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

$\Rightarrow$ 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

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

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

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

CFL - Ingredients - FEDatas

Use variadic templates to define a storage container

```
template <class FEData>
class FEDatas<FEData>;

template <class FEData, typename... Types>
class FEDatas<FEData, Types...> : public FEDatas<Types...>
{
  public:
  // operators to access the top-level FEData object
  // and the FEEvaluation object
  [...]

  // store integration and evaluation flags
  [...]

  FEData fe_data;
}
```

CFL - Ingredients - Form

```
template <class Test, class Expr, FormKind kind_of_form, typename NumberType>
class Form final
{
  public:
  // access to the template parameters
  // and integration flags from the Test object
  [...]

  template <class FEEvaluation>
  void evaluate(FEEvaluation& phi, unsigned int q) const;

  template <class FEEvaluation>
  static void integrate(FEEvaluation& phi);

  template <class FEEvaluation>
  auto value(FEEvaluation& phi, unsigned int q) const;

  template <class FEEvaluation, typename ValueType>
  static void submit(FEEvaluation& phi, unsigned int q,
                     const ValueType& value);
}
```

CFL - Ingredients - Forms

Use variadic templates to define a storage container

```
template <typename FormType>
class Forms<FormType>;

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

CFL - Ingredients - Test Functions

Use CRTP-style declarations

```
template <int rank, int dim, unsigned int idx>
class TestFunction final
  : public TestFunctionBase<TestFunction<rank, dim, idx>>
{
  public:
    using Base = TestFunctionBase<TestFunction<rank, dim, idx>>;
    static constexpr IntegrationFlags integration_flags{ true, false,
                                                         false, false };

    template <class FEEvaluation, typename ValueType>
    static void
    submit(FEEvaluation& phi, unsigned int q, const ValueType& value);
};
```

CFL - Ingredients - Ansatz Functions

Use CRTP-style declarations

```
template <int rank, int dim, unsigned int idx>
class FEFunction final : public FEFunctionBase<FEFunction<rank, dim, idx>>
{
  public:
    using Base = FEFunctionBase<FEFunction<rank, dim, idx>>;
    // inherit constructors
    using Base::Base;

    template <class FEDatas>
    auto value(const FEDatas& phi, unsigned int q) const;

    template <class FEEvaluation>
    static void set_evaluation_flags(FEEvaluation& phi);
};
```

Linear combinations and products allowed.

CFL - Ingredients - MatrixFreeIntegrator - I

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

CFL - Ingredients - MatrixFreeIntegrator - II

```
template <int dim, typename VectorType, class FORM, class FEDatas>
template <class FEEvaluation>
void MatrixFreeIntegrator<[...]> do_operation_on_cell
  (FEEvaluation& phi, const unsigned int cell) const
{
  phi.evaluate();
  constexpr unsigned int n_q_points = FEEvaluation::get_n_q_points();

  for (unsigned int q = 0; q < n_q_points; ++q)
    form->evaluate(phi, q);

  phi.integrate();
}

```

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

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

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);

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

LaplaceProblem<dim,
               decltype(fe_data_system),
               decltype(fe_data_level),
               decltype(f_system)>
  laplace_problem(fe_datas_system, fe_datas_level, f_system);
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

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

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);
auto fe_datas_system = (fedata_e_system, fedata_u);
FEData<FE_Q, degree, 1, dim, 0, degree, float> fedata_e_level(fe_u);
FEData<FE_Q, degree, 1, dim, 1, degree, float> fedata_u_level(fe_u);
auto fe_datas_level = (fedata_e_level, fedata_u_level);

CFL::dealii::MatrixFree::TestFunction<0, dim, 0> v;
auto Dv = grad(v);
CFL::dealii::MatrixFree::FEFunction<0, dim, 0> e("e");
auto De = grad(e);
CFL::dealii::MatrixFree::FEFunction<0, dim, 1> u("u");
auto Du = grad(u);

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

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

Code Examples - Stokes

Stokes problem

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

$$\begin{aligned} (\varepsilon(\boldsymbol{u}), \varepsilon(\boldsymbol{v})) - (p, \nabla \cdot \boldsymbol{v}) &= (\boldsymbol{f}, \boldsymbol{v}) && \forall \boldsymbol{v} \in \boldsymbol{V} \\ (\nabla \cdot \boldsymbol{u}, q) &= 0 && \forall q \in Q \end{aligned}$$

where

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

Code Examples - Stokes

```
FESystem<dim> fe_u (FE_Q<dim>(degree + 1), dim);
FE_Q<dim> fe_p (degree);

FEData<FESystem, degree, dim, dim, 0, degree> fedata1(fe_u);
FEData<FE_Q, degree, 1, dim, 1, degree> fedata2(fe_p);
auto fe_datas = (fedata1, fedata2);

FEFunction<1, dim, 0> u("u");
auto Divu = div(u);
FESymmetricGradient<2, dim, 0> Du(R"((\nabla+\nabla^T)u)");
FEFunction<0, dim, 1> p("p");
FELiftDivergence<decltype(p)> Liftp(p);
TestFunction<1, dim, 0> v;
TestFunction<0, dim, 1> q;
auto Dv = grad(v);

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

Code Examples - Laplace (DG)

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

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

Code Examples - Laplace (DG) - I

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

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

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

FEFunction<0, 1, 0> u("u");
auto Du = grad(u);
FEFunctionInteriorFace<0, 1, 0> u_p("u+");
FEFunctionExteriorFace<0, 1, 0> u_m("u-");
FENormalGradientInteriorFace<0, 1, 0> Dnu_p("Dn_u+");
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)
26              +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:

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

Future:

- Support more back ends.
- Provide more flexibility in the operators defined.