SFC - the SyFi Form Compiler

Martin Sandve Alnæs and Kent-André Mardal

Workshop on Automating the Development of Scientific Computing Software

LSU, March 6th 2008

[simula . research laboratory]

Outline

- How SFC fits into FEniCS
- Form definition examples
- Efficiency tests

SFC is a *form compiler* in the FEniCS software framework

```
from sfc import *
<define a, L using sfc>

from dolfin import *
mesh = Mesh(filename)
A = assemble(a, mesh)
b = assemble(L, mesh)
u = Vector()
solve(A, u, b)
```

- a, L are implementations of the UFC interface

Weak Form -> SFC Code

- The integrand of a weak form is computed using the symbolic tools of SyFi

$$a(v, u) = \int_\Omega uv \, dx \longrightarrow$$

```
# user code:
def mass(v, u, itg):
    return inner(u, v)
```

SFC Code -> UFC C++ Code

- SFC compiles a symbolic description of a weak form into UFC-compatible C++ code

```
// generated code:
void ...::tabulate_tensor(...) const
{
  ...
  A[3*0 + 0] = detG*( (Ginv00*Ginv00)/2.0 + ... )
  A[3*0 + 1] = detG*(-(Ginv00*Ginv00)/2.0 - ... )
  A[3*0 + 2] = detG*(-(Ginv01*Ginv01)/2.0 - ... )
  ...
}
```

```
# user code:
def stiffness(v, u, itg):
    GinvT = itg.GinvT()
    Du = grad(u, GinvT)
    Dv = grad(v, GinvT)
    return inner(Du, Dv)
```

```
form = Form(<...>)
a = compile_form(form)
```

Declaring Elements

- Element declaration is similar to FFC

```
polygon = "tetrahedron"
element = FiniteElement("Lagrange", polygon, 1)
element = VectorElement("Lagrange", polygon, 1)
element = TensorElement("Lagrange", polygon, 1)
```

- Polygon can be one of "interval", "triangle", "tetrahedron", "quadrilateral", and "hexahedron" (defined by UFC)

Declaring Form Arguments

$$a(v, u; f) = \int_\Omega f u v \, dx$$

```
v = TestFunction(element)
u = TrialFunction(element)

f = Function(element)

a = Form(basisfunctions = [v, u],
             coefficients     = [f])

<define integrands>
```

- Argument declaration similar to FFC

Defining Integrands (1)

- Alternative 1:
  Loop over symbolic expressions
  for basis functions manually

```
a = Form(basisfunctions = [v, u],
               coefficients = [c])
itg = a.add_cell_integral()

c = itg.coefficient(0)
for j, u in enumerate(itg.v_basis(1)):
    for i, v in enumerate(itg.v_basis(0)):
        integrand = inner(v, u)
        itg.set_A((i, j), integrand)
```

Defining Integrands (2)

- Alternative 2:
  Define integrand in a callback function

```
def mass(v, u, c, itg):
    return c*inner(v, u)

a = Form(basisfunctions = [v, u],
            coefficients = [c],
            cell_integrands = [mass])
# OR:
a.add_cell_integral(mass)
```

Symbolic Tools

- Based on the symbolic library GiNaC (swiginac)
- Can use f.ex. symbolic differentiation
- Computing the Jacobi of a nonlinear form is automated:

```
F = Form(basisfunctions = [v],
         coefficients = [w])
<define integrands of F>
J = Jacobi(F)
```

Hyperelasticity: Tedious differentiation of the stress tensor

$$S^p = \frac{\partial \Psi}{\partial E},$$
$$\Psi = \frac{1}{2}K(e^W - 1) + C_{compr}(J \log J - J + 1)$$
$$W = b_{ff}E_{ff}^2 + b_{xx}(E_{nn}^2 + E_{ss}^2 + E_{ns}^2) + b_{fx}(E_{fn}^2 + E_{nf}^2 + E_{fs}^2 + E_{fs}^2).$$

```
# Fung type strain energy function
W = bff*E[0,0]**2 + bxx*(E[2,2]**2 + ...) + bfx*(E[0,1]**2 + ...)
psi = K * (exp(W) - 1) / 2 + C_compr*(J*ln(J) – J + 1)
S = diff(psi, E)
```

UFL

- The symbolic expressions SFC uses are too explicit for some high level optimizations
- More interoperability between SFC and FFC would be nice
- A declarative form language to be shared by FFC and SFC is under construction, called UFL

Timing setup

- In the following we have run the element tensor computation in a loop without the mesh iteration and matrix insertion overhead
- Times are per element tensor computation
- The best achievable actual speedup of the assembly is limited by sparse matrix insertion

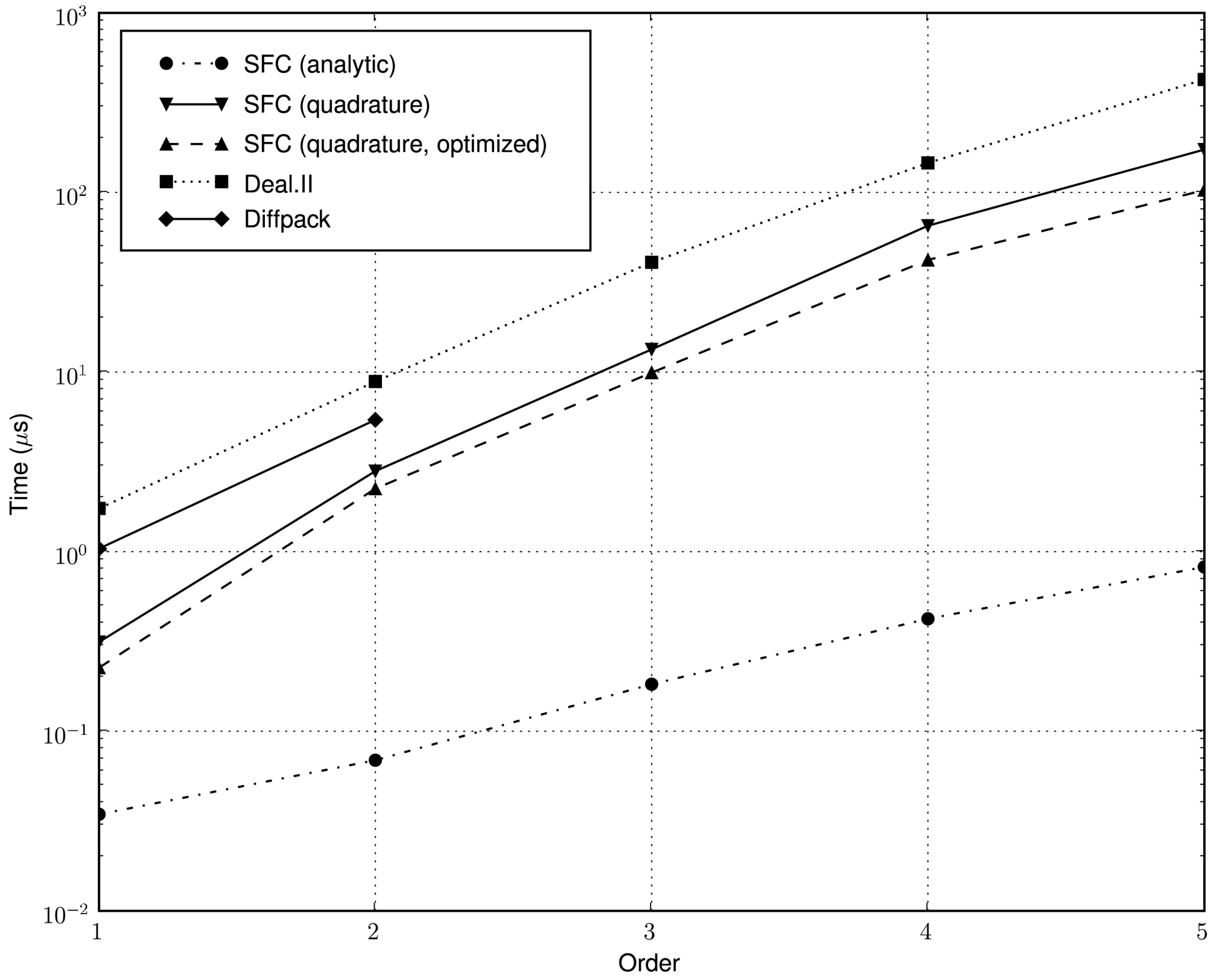
Example: mass matrix

$$a(v, u) = \int_\Omega vu \, dx$$

```
def mass(v, u, itg):
    return inner(v, u)
```

- After analytic integration there's only one multiplication per element tensor entry

Example: mass matrix (on quadrilaterals)



Example: mass matrix

	Triangle					Tetrahedron			
Order	1	2	3	4	5	1	2	3	4
Timescales (μs)	0.024	0.039	0.083	0.16	0.29	0.051	0.095	0.28	0.82
SFC	1.0	1.0	1.0	1.0	1.0	1.00	1.0	1.0	1.0
SFC (quad.)	7.3	26.8	61.8	114.2	177.9	6.6	48.1	161.8	333.2
SFC (quad., opt.)	5.4	18.7	46.0	71.4	111.0	5.3	36.6	104.1	198.5
FFC	1.0	1.1	1.0	1.1	1.1	0.9	1.0	1.0	1.1
Diffpack	19.0	56.4	–	–	–	15.1	–	–	–

Table I. Time to compute the element tensor of the mass form, relative to a symbolic integration for each order.

	Quadrilateral					Hexahedron			
Order	1	2	3	4	5	1	2	3	4
Timescales (μs)	0.035	0.069	0.18	0.42	0.82	0.072	0.49	5.45	21.8
SFC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
SFC (quad.)	9.1	40.7	73.2	154.4	210.8	32.6	204.9	264.2	–
SFC (quad., opt.)	6.5	32.5	54.3	99.5	125.4	25.0	120.1	201.5	379.5
Deal.II	50.4	128.4	223.5	345.2	518.3	160.8	404.5	453.2	811.2
Diffpack	30.1	78.6	–	–	–	88.3	228.2	–	–

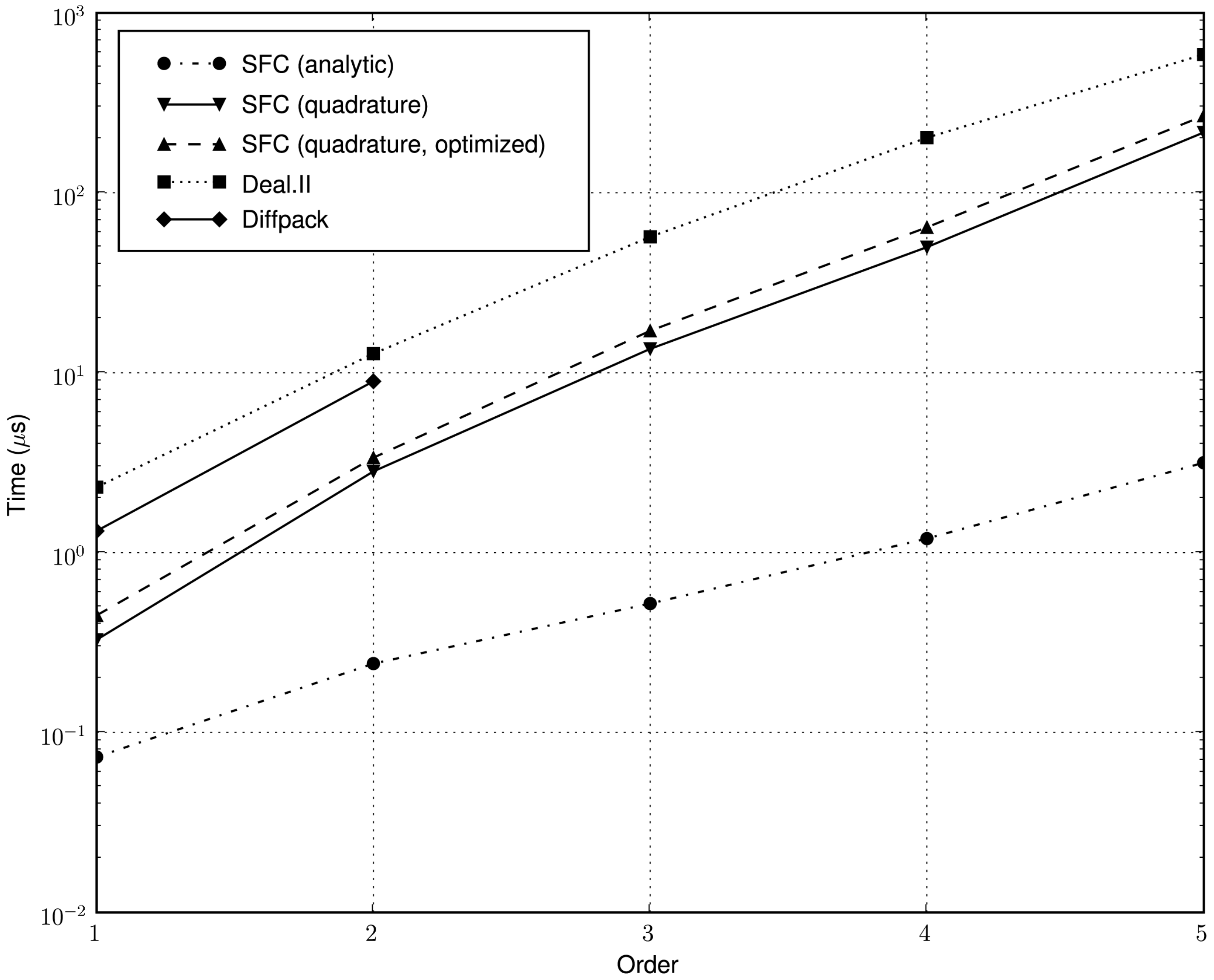
Table II. Time to compute the element tensor of the mass form, relative to a symbolic integration for each order.

Example: stiffness matrix

$$a(v, u) = \int_\Omega \nabla v \cdot \nabla u \, dx$$

```
def stiffness(v, u, itg):
    GinvT = itg.GinvT()
    Dv = grad(v, GinvT)
    Du = grad(u, GinvT)
    return inner(Du, Dv)
```

Example: stiffness matrix (on quadrilaterals)



Example: stiffness matrix

	Triangle					Tetrahedron			
Order	1	2	3	4	5	1	2	3	4
Timescale (in μs)	0.057	0.10	0.28	0.8	1.5	0.14	0.7	2.6	11.3
SFC (analytic)	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
SFC (quadrature)	2.7	9.6	17.0	17.3	24.9	2.1	8.6	20.4	36.5
FFC	0.8	0.7	0.7	0.6	0.7	0.7	0.4	0.5	0.8
Diffpack	10.5	31.0	–	–	–	8.2	–	–	–

Table III. Time to compute the element tensor of the stiffness form for each order respectively on triangle and tetrahedron elements.

	Quadrilateral					Hexahedron			
Order	1	2	3	4	5	1	2	3	4
Timescale (in μs)	0.073	0.24	0.52	1.2	3.2	0.36	2.3	20.6	75.8
SFC (analytic)	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
SFC (quadrature)	4.5	11.7	26.1	41.7	68.5	7.8	52.2	83.8	209.6
Deal.II	31.6	52.8	109.3	169.2	186.1	42.3	123.0	166.9	332.4
Diffpack	18.1	37.1	–	–	–	27.5	105.4	–	–

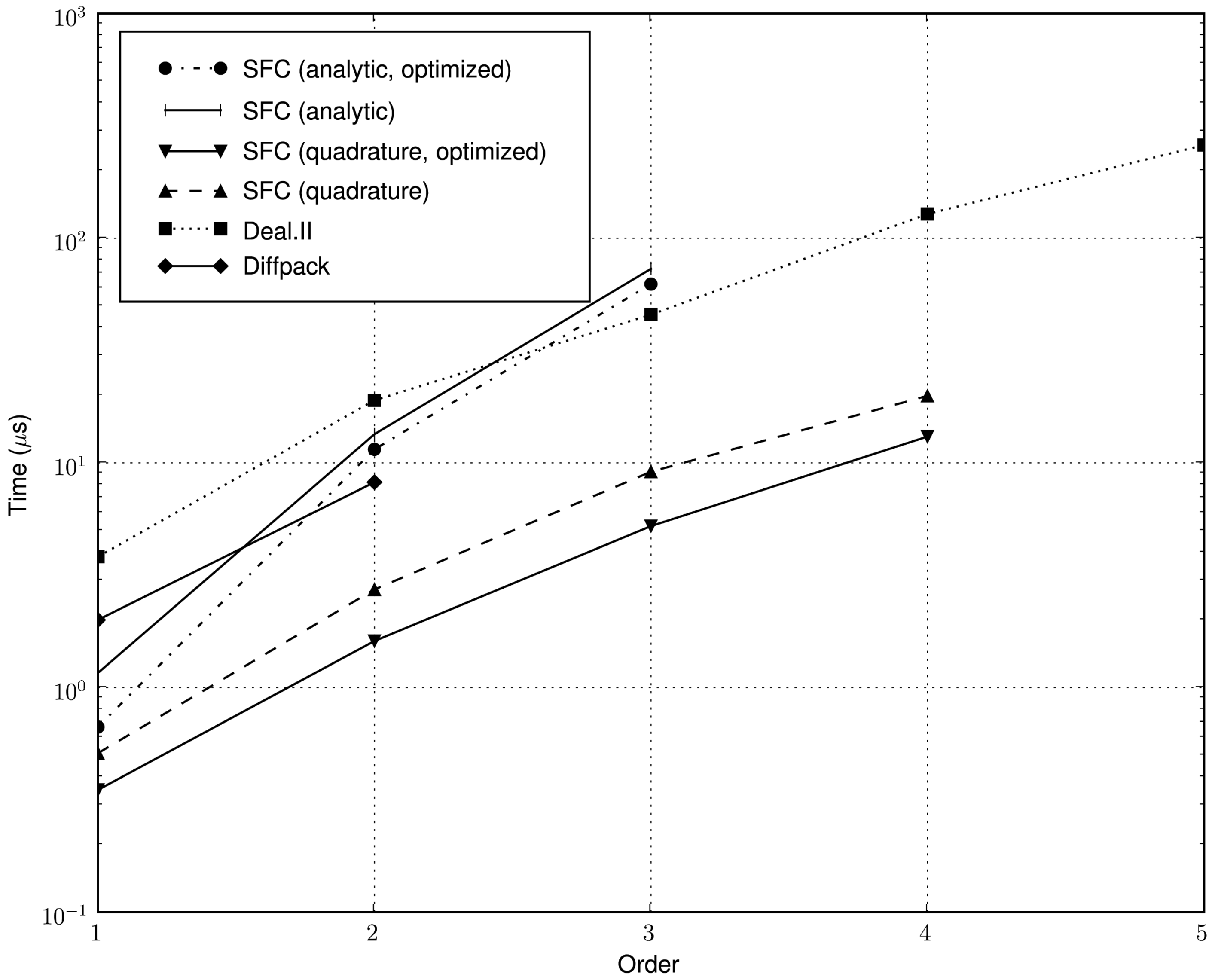
Table IV. Time to compute the element tensor of the stiffness form for each order respectively on quadrilateral and hexahedron elements.

Example: nonlinear convection vector (on quadrilaterals)

$$a(v; w) = \int_\Omega w \cdot \nabla w \cdot v \, dx$$

```
def convection_vector(v, w, itg):
    GinvT = itg.GinvT()
    Dw = grad(w, GinvT)
    return dot(dot(w, Dw), v)
```

Example: nonlinear convection vector (on quadrilaterals)

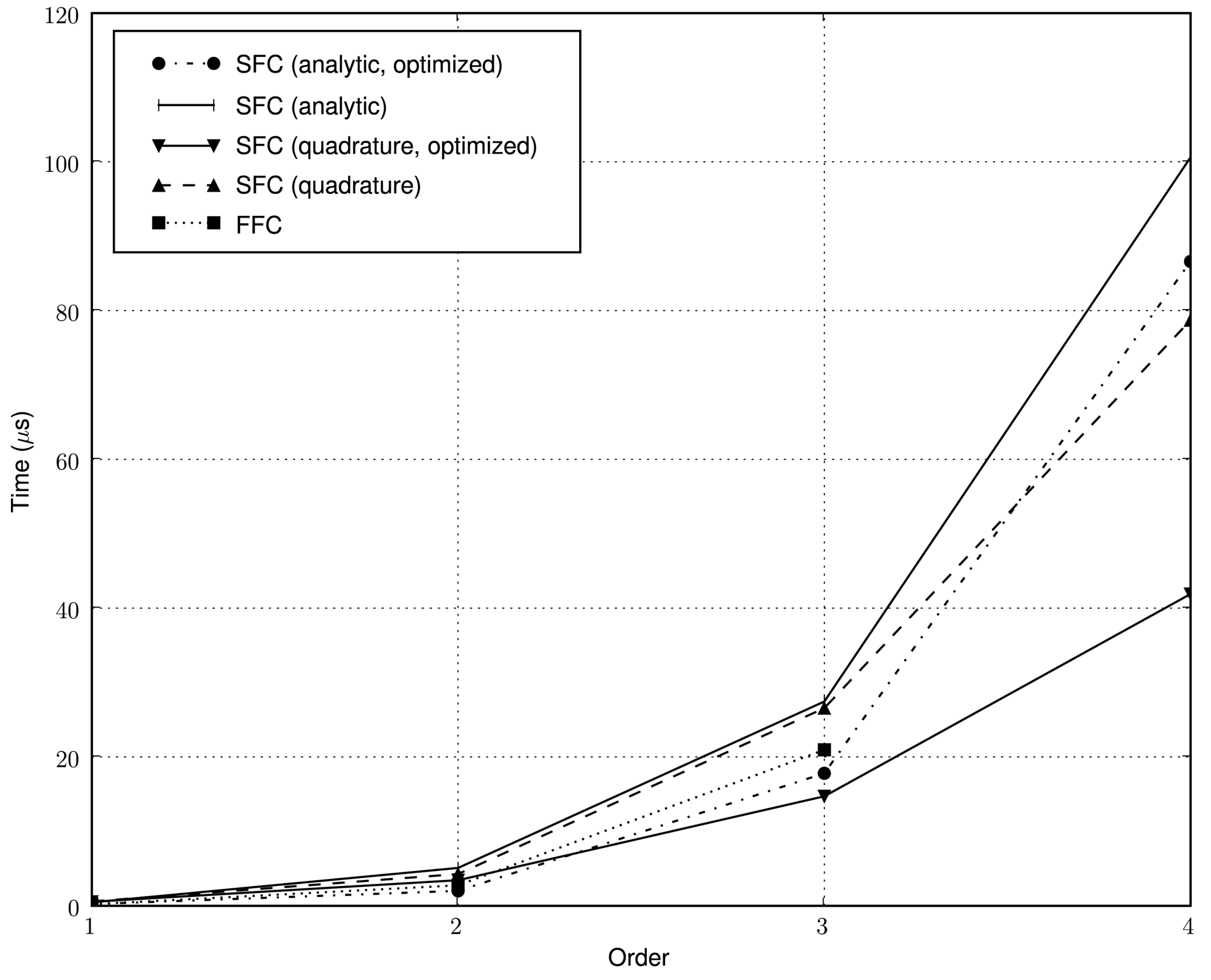


Example: Jacobi of nonlinear convection vector

$$a(v, u; w) = \int_\Omega u \cdot \nabla w \cdot v + w \cdot \nabla u \cdot v \, dx$$

```
def convection_jacobi(v, u, w, itg):
    GinvT = itg.GinvT()
    Dw = grad(w, GinvT)
    Du = grad(u, GinvT)
    return dot(dot(u, Dw) + dot(w, Du), v)
```

Example: Jacobi of nonlinear convection vector (on triangles)



Questions?