Automatic code generation for Finite Element Methods applied to the Shallow-Water equations

Van Thieu Vu
Leiden Institute of
Advanced Computer Science
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands
Email: vtvu@liacs.nl

Gerard Cats
Royal Netherlands
Meteorological Institute
Wilhelminalaan 10
3730 AE De Bilt
The Netherlands
Email: cats@knmi.nl

Lex Wolters
Leiden Institute of
Advanced Computer Science
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands
Email: lexx@liacs.nl

Abstract

CTADEL, a code generation tool, has proven that it can generate highly optimized Fortran code for several schemes in weather forecast models such as $dynamics$, $cloud$ and $turbulence$ schemes. All these codes use finite difference methods as discretization. In this paper we describe how to extend CTADEL to generate code for the energy conserving Galerkin finite element schemes. We apply this technique to solve the Shallow-Water equations. We use the hand code as the reference to assess the efficiency of the generated code. The experiments show that the generated code has a performance gain upto two-thirds over the hand code.

1 Introduction

Coding for computationally intensive programs usually faces a number of issues. The main issue is the task to correctly translate the problem from the underlying mathematical theory into an efficient algorithm written in some programming language, and how to maintain the program when the problem specification is modified. Furthermore, making these programs efficient on different types of computer architectures is difficult, because each computer system requires different algorithms for optimization. A Problem Solving Environment (PSE) or code generator can assist these problems.

CTADEL (Code generation Tool for Applications based on Differential Equation using high-level Language specification), developed by van Engelen [2], is able to produce efficient code for sequential, vector, shared memory virtual, and distributed memory parallel computers. The problem, specified in a high-level language, is transformed into an abstract syntax tree (AST) using a transformation-database. On the AST global optimizations, common subexpression elimination and architecture dependent optimizations are performed before the result code is generated. These transformations ensure the generation of correct and efficient codes, which can compete with hand optimized codes.

CTADEL can generate code for several schemes in weather forecast models such as $dynamics$, $cloud$ and $turbulence$ schemes. Till now, all codes which have been generated by CTADEL use finite difference discretization techniques. The aim of this work is to extend CTADEL to handle finite element methods. In this paper we present how to extend CTADEL to generate code for the energy conserving Galerkin finite element schemes. The Shallow-Water equations are chosen as a test-case. These equations describe a single homogenous layer of fluid. In general they are applied to various problems, such as tidal flow, earthquake waves in oceans, etc. In meteorology they are used to forecast the geopotential.

This paper is organized as follows: Section 2 introduces the CTADEL system. A short description of the Galerkin finite element schemes for the Shallow-Water equations given in Section 3. Section 4 presents templates for Galerkin finite element schemes and the specification of the Shallow-Water equations. In Section 5 we describe the code generation process in CTADEL. The results and an assessment of the generated code are shown in Section 6. Section 7 is reserved for conclusions and future work.

2 CTADEL

Many attempts have been made to solve scientific or physical models numerically using computers. Today, a large collection of libraries, tools, and Problem Solving Environments (PSEs) have been developed for these problems. The computational kernels of most PSEs consist of a large library containing routines for several numerical solution methods. These routines form the templates for the resulting code. The power of this library determines the numerical knowledge of such a system.

A different approach to those so-called library-based PSEs consist of a collection of tools that generate code based on a problem specification without the use of a library. The numerical knowledge is determined by the expressiveness of the problem specification language and the underlying translation techniques. CTADEL belongs to this class of PSEs. Furthermore, a hardware description of the target platform is included in CTADEL's problem specification. This makes it possible to produce efficient codes for different types of architectures.

3 Galerkin finite element methods for the Shallow-Water equations

This section gives a short description of Galerkin finite element methods for the Shallow-Water equations. For more detail, the reader is referred to [13, 14].

3.1 Galerkin projection and finite element methods

For an arbitrary field ϕ, we assume the following representation

$$\tilde{\phi}\left(r\right) = \sum_{\nu=1}^{N} \phi_\nu e_\nu \left(r\right). \tag{1}$$

Here r denotes the x or y horizontal coordinates and e_ν are basis functions.

In order to formulate the finite element methods, it is necessary to approximate a rather general field $\phi\left(r\right)$ by a function $\tilde{\phi}\left(r\right)$ in the form of the right hand side of Equation (1). This is achieved by the Galerkin projection, where the coefficient ϕ_ν in Equation (1) are defined by

$$\left(\tilde{\phi}, e_\nu\right) = \left(\phi_\nu, e_\nu\right), \quad \nu \in \{1, \cdots, N\}, \tag{2}$$

with the scalar product (a, b) is computed as

$$(a, b) = \int a\left(r\right) b\left(r\right) d\mu, \tag{3}$$

where

$$d\mu = w\left(r\right) dxdy \tag{4}$$

with the positive continuous function w is called the $weight$ of the Galerkin projection. The relation between the fields ϕ and $\tilde{\phi}$, given by (2), can be written as

$$\tilde{\phi} = G\phi, \tag{5}$$

where G is called the Galerkin projection operator.

For equations of the form

$$\phi_t = f\left(\phi\right) \tag{6}$$

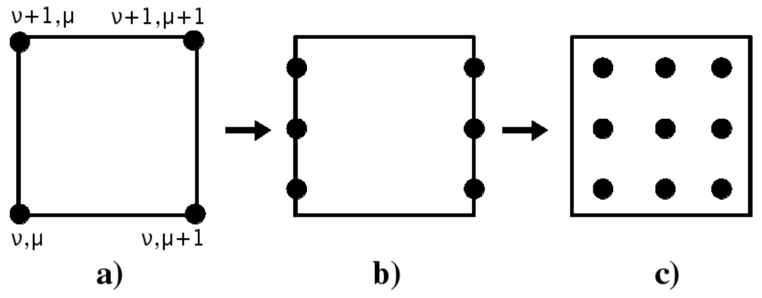

Figure 1. Grids: a) Nodepoint grid; b) Y-collocation grid; c) Collocation grid.

where the subscript t denotes the differentiation in time, the approximation

$$\tilde{\phi}_t = Gf\left(\tilde{\phi}\right) \tag{7}$$

is called the standard finite element scheme.

3.2 The Galerkin finite element method for the Shallow-Water equations

The Shallow-Water equations have the following form

$$\begin{aligned} \frac{\partial U}{\partial t} &= fV - UU_x - VU_y - gH_x, \\ \frac{\partial V}{\partial t} &= -fU - UV_x - VV_y - gH_y, \\ \frac{\partial H}{\partial t} &= -\left(UH\right)_x - \left(VH\right)_y, \end{aligned} \tag{8}$$

Here, the subscripts denote the differentiation with respect to the specified variable. U and V are the velocity components in x and y direction, respectively, H denotes the height field, g is the gravity constant, and f represents the Coriolis parameter.

The Galerkin finite element method for the Shallow-Water equations, as proposed by Steppler [13, 15], reads as follows

$$\begin{aligned} \frac{\partial U_{\nu,\mu}}{\partial t} &= G_1 \left\langle \eta V - \left(G_2 \left\langle \tfrac{1}{2}\left(U^2+V^2\right) + gH\right\rangle\right)_x \right\rangle_{\nu,\mu}, \\ \frac{\partial V_{\nu,\mu}}{\partial t} &= G_1 \left\langle -\eta U - \left(G_2 \left\langle \tfrac{1}{2}\left(U^2+V^2\right) + gH\right\rangle\right)_y \right\rangle_{\nu,\mu}, \\ \frac{\partial H_{\nu,\mu}}{\partial t} &= -G_2 \left\langle \left(UH\right)_x + \left(VH\right)_y \right\rangle_{\nu,\mu}, \\ \eta &= V_x - U_y + f, \end{aligned} \tag{9}$$

where η represents the absolute vorticity, ν and μ are the coordinates of the nodepoint grid (Figure 1a). G_1 and G_2 are the Galerkin operators with weights H and 1, respectively. The angle brackets denote the Galerkin projection (see [14]).

In order to approximate Equation (9) by the Galerkin finite element method, we interpolate the fields from the nodepoint to the y-collocation grid (Figure 1b) and then from the y-collocation to the collocation grid (Figure 1c). The purpose of interpolation is to create a finer resolution grid. Next, we compute the right hand side of Equation (9) on the collocation grid. Finally, we perform the Galerkin projection in the x direction and then in the y direction to transform from collocation to nodepoint grid. In the next paragraphs, we will describe these interpolations and projections.

Interpolation

Let ξ be either x or y and $\psi(\xi)$ be a function obtained from two dimensional function $\psi(x, y)$ by keeping the other variable fixed, then the interpolation formula for an interval (a, b) can be written as

$$\psi(\xi) = \frac{\psi(a)\,(b-\xi) + \psi(b)\,(\xi - a)}{b-a}, \tag{10}$$

where ψ stands for U, V, or H. Within each interval, we always interpolate to the three Gauss-points ξ_ρ, $\rho = 1, 2, 3$ [1], defined as

$$\xi_\rho = \frac{b-a}{2}\xi_\rho^{'} + \frac{b+a}{2}, \tag{11}$$

with

$$\xi_1^{'} = -0.774596669241483,\ \ \xi_2^{'} = 0,\ \ \xi_3^{'} = +0.774596669241483. \tag{12}$$

Choosing the interval $(a, b) \equiv (\xi, \xi + 1)$ and substitute Equation (11) in Equation (10), we have

$$\psi\left(\xi_\rho\right) = \frac{\psi(\xi)\left(1 - \xi_\rho^{'}\right) + \psi(\xi+1)\left(1 + \xi_\rho^{'}\right)}{2}. \tag{13}$$

When $\overline{\psi(\xi)}^{+} = \frac{\psi(\xi)+\psi(\xi+1)}{2}$ and $\triangle^{+}\psi(\xi) = \psi(\xi+1) - \psi(\xi)$ Equation (13) can be rewritten as

$$\psi\left(\xi_\rho\right) = \overline{\psi(\xi)}^{+} + \frac{1}{2}\xi_\rho^{'}\triangle^{+}\psi(\xi). \tag{14}$$

This equation is used for the interpolation. With $\xi = y$ and taking $(\nu, \nu + 1)$ as interval, the interpolation will lead from the nodepoint to the y-collocation grid. Similarity, interpolate with $\xi = x$ in the interval $(\mu, \mu + 1)$ will transfer fields from the y-collocation to the collocation grid (see Figure 1).

Projection

Let $RS(\xi)$ denotes the right hand side of Equation (9) where ξ is either x or y, then the Galerkin projection for $RS(\xi)$ is defined as

$$G\left[RS(\xi)\right] = \int e_\nu\left(\xi_\mu\right) RS(\xi) w\left(\xi\right) d\xi, \tag{15}$$

with basis functions $e_\nu\left(\xi\right)$ defined by the properties

$$e_\nu\left(\xi_\mu\right) = \begin{cases} 0, & \text{if } \mu \neq \nu, \\ 1, & \end{cases} \tag{16}$$

and $w\left(\xi\right)$ is a weight function which will be either H or 1 for the $G1$ and $G2$ Galerkin operator, respectively.

The integration of any function $\psi(\xi)$ in the interval (a, b) can be computed as follows

$$\int_a^b \psi(\xi) d\xi = \sum_{\rho=1}^{3} \psi(\xi_\rho) g_\rho \left(b - a\right), \tag{17}$$

with ξ_ρ are calculated according to Equation (11) and g_ρ are the Gaussian-weights, defined as

$$g_1 = g_3 = 0.555555555555555,\ \ g_2 = 0.888888888888888. \tag{18}$$

Choosing the interval $(a, b) \equiv (\xi, \xi + 1)$ and substitute Equations (16) and (17) in Equation (15), we have

$$\psi(\xi) = \sum_{\rho=1}^{3}\left(\frac{\triangle\xi}{2}\frac{\psi\left(\xi_\rho\right) w\left(\xi_\rho\right) g_\rho\left(1 - \xi_\rho^{'}\right) + \psi\left(\xi_\rho - 1\right) w\left(\xi_\rho - 1\right) g_\rho\left(1 + \xi_\rho^{'}\right)}{2}\right). \tag{19}$$

Let $\overline{\psi\left(\xi_{\rho}\right) w\left(\xi_{\rho}\right)}^{-}=\frac{\psi(\xi_{\rho}-1) w(\xi_{\rho}-1)+\psi(\xi_{\rho}) w(\xi_{\rho})}{2}$ and $\triangle^{-}\psi\left(\xi_{\rho}-1\right) w\left(\xi_{\rho}-1\right)=$ $\psi\left(\xi_{\rho}\right) w(\xi_{\rho})-\psi\left(\xi_{\rho}-1\right) w\left(\xi_{\rho}-1\right)$, we have

$$\psi(\xi)=\sum_{\rho=1}^{3}\left(\frac{\triangle\xi}{2} g_{\rho}\left(\overline{\psi\left(\xi_{\rho}\right) w\left(\xi_{\rho}\right)}^{-}-\frac{\xi_{\rho}^{'}}{2}\triangle^{-}\psi\left(\xi_{\rho}-1\right) w\left(\xi_{\rho}-1\right)\right)\right). \tag{20}$$

Equation (20) is used for the projection. If $\xi = x$ and RS is given on the collocation grid, by keeping y fixed, the Galerkin projection will transform from the collocation to the y-collocation grid. Similarity, if $\xi = y$ and RS is given on the y-collocation grid, by keeping x fixed, the Galerkin projection will lead from the y-collocation to the nodepoint grid.

4 Specification

As discussed in Section 3, the Galerkin finite element methods consist of two main steps: interpolation and projection. In this section we will present how to construct templates for these steps in CTADEL. Next, we show the specification of the Shallow-Water equations applying the Galerkin finite element method.

4.1 Templates

Interpolation template

The interpolation of a field is performed following Equation (14). In CTADEL we define the template $interpolate$ which has as input ψ, the field to be interpolated in ξ direction, and as output the corresponding interpolated field. The code of $interpolate$ reads

```
average_f(_::field Xi(grid) ,Xi)::field Xi(half).
average_f(Psi::float dim Unit,Xi)::float dim Unit:=1/2*(Psi+Psi@(Xi=Xi+1)).
delta_f(_::field Xi(grid) ,Xi)::field Xi(half).
delta_f(Psi::float dim Unit,Xi)::float dim Unit:=Psi@(Xi=Xi+1)-Psi.
interpolate(_::field Xi(grid) ,_::field Xi(grid) ,Xi)::field Xi(grid).
interpolate(Psi::float dim Unit,Gp::float dim Unit,Xi)::float dim Unit
                              := average_f(Psi,Xi)+1/2*Gp*delta_f(Psi,Xi).
```

In the template $interpolate$ we denote Psi, Gp, and Xi for the notations ψ, $\xi^{'}$, and ξ as in Equation (14), respectively. The LaTeX output for this template produced by CTADEL has the following form

$$interpolate\left(\psi,\xi_{\rho}^{'},\xi\right)=\overline{\psi(\xi)}^{+}+\frac{1}{2}\xi_{\rho}^{'}\triangle^{+}\psi(\xi). \tag{21}$$

We see that the right hand sides of Equations (14) and (21) are exactly the same. This proves that the template $interpolate$ definition is correct.

Projection template

Based on Equation (20) we construct the template $project$ for Galerkin projection. This template has as input the field $\psi\left(\xi_{\rho}\right)$ to be projected in the ξ direction, the weight function $w\left(\xi_{\rho}\right)$, the Gauss-points and Gauss-weights parameters. The output of $project$ is the corresponding projected field. The template $project$ has the following form

```
average_b(_::field Xi(grid) ,Xi)::field Xi(half).
average_b(Psi::float dim Unit,Xi)::float dim Unit:=1/2*(Psi@(Xi=Xi-1)+Psi).
delta_b(_::field Xi(grid) ,Xi)::field Xi(half).
delta_b(Psi::float dim Unit,Xi)::float dim Unit:=Psi-Psi@(Xi=Xi-1).
project(_::field Xi(grid),_::field Xi(grid),_::field Xi(grid),_::field Xi(grid),Xi)
     ::field Xi(gird).
project(Psi::float dim Unit,W::float dim Unit,Gp::float dim Unit,Gw::float dim Unit,Xi)
       ::float dim Unit
      :=sum(1/2*delta(Xi)*Gw*(average_b((PsiW),Xi)-1/2*Gp*delta_b((PsiW),Xi)),rho=1..3).
```

In the template $project$ Psi denotes the input field $\psi\left(\xi_\rho\right)$, Xi is direction of projection ξ, W is the weight function w, and Gp and Gw denote the Gauss-points $\xi_\rho^{'}$ and the Gauss-weights g_ρ, respectively. CTADEL produces the LaTex output for this template as

$$project\left(\psi, w, \xi_\rho^{'}, g_\rho, \xi\right) = \sum_{\rho=1}^{3}\left(\frac{\triangle\xi}{2} g_\rho \left(\overline{\psi\left(\xi_\rho\right) w\left(\xi_\rho\right)}^{-} - \frac{\xi_\rho^{'}}{2}\triangle^{-}\psi\left(\xi_\rho - 1\right) w\left(\xi_\rho - 1\right)\right)\right). \tag{22}$$

The right hand side of Equation (22) is exactly the same as it in Equation (20). This proves the correctness of $project$ template definition.

4.2 Specification of the Shallow-Water equations

Next we show the specification of the Shallow-Water equations. For convenience, we only give the specification related to the field H. The fields U and V are specified in a similar way.

```
% Gauss-points(Gp) and Gauss-weights(Gw) parameters:
Gp :: float field (xc(grid)) on long_collocation.
Gw :: float field (xc(grid)) on long_collocation.

% Field on nodepoint(H), y-collocation(Hycol), and collocation(Hcol) grid:
H    :: float  dim  Height field (x(grid),y(grid),t) on horizontal.
Hycol:: float  dim  Height field (x(grid),y(grid),yc(grid),t) on y_collocation.
Hcol :: float  dim  Height field (x(grid),xc(grid),y(grid),yc(grid),t) on collocation.

% Interpolate field from nodepoint to collocation grid.
Hycol = interpolate(H,    Gp,y).
Hcol  = interpolate(Hycol,Gp,x).

% Compute the right hand side of Equation(9) in collocation grid:
RsHcol :: float  dim  Height field (x(grid),xc(grid),y(grid),yc(grid),t) on collocation.
RsHcol = - d (Ucol*Hcol) / d x - d (Vcol*Hcol) / d y.

% Project right hand size from collocation to nodepoint grid:
RsHycol:: float  dim  Velocity field (x(grid), y(grid), yc(grid), t) on y_collocation.
RsH    :: float  dim  Velocity field (x(grid), y(grid), t) on horizontal.
RsHycol= project_bound(RsHcol,1,Gp,Gw,x) if on boundary\\
                project(RsHcol,1,Gp,Gw,x) otherwise.
RsH    = project_bound(RsHycol,1,Gp,Gw,y) if on boundary\\
                project(RsHycol,1,Gp,Gw,y) otherwise.
```

5 Generating code

This section presents the code generation process in CTADEL. We describe the steps which CTADEL performs in order to translate from the abstract specification into optimized Fortran code. We give the LaTeX output of each step for the field H which is the result of the specification given in Subsection 4.2. These LaTeX outputs are produced by CTADEL to compare the specification with the mathematical description and to check the different steps. The outputs of the fields U and V are similar.

In CTADEL, the code is generated through the following steps:

1. Get scalar values and Dimensional analysis: All scalar values and equations in the specification are retrieved and sorted in order of computation. Then CTADEL analyzes all units and dimensions declared in the specification. Scalar form as follows

$$\begin{aligned}\text{RsHcol} := \text{-}&\left(\overline{\overline{U} + \tfrac{1}{2}\xi_\rho^{'}\triangle_y\left[U\right]} + \tfrac{1}{2}\xi_\rho^{'}\triangle_x\left[\overline{U} + \tfrac{1}{2}\xi_\rho^{'}\triangle_y\left[U\right]\right]\right) * \left(\frac{\partial\left(\overline{\overline{H}+\frac{1}{2}\xi_\rho^{'}\triangle_y[H]}+\frac{1}{2}\xi_\rho^{'}\triangle_x\left[\overline{H}+\frac{1}{2}\xi_\rho^{'}\triangle_y[H]\right]\right)}{\partial x}\right)\\ -&\left(\overline{\overline{V} + \tfrac{1}{2}\xi_\rho^{'}\triangle_y\left[V\right]} + \tfrac{1}{2}\xi_\rho^{'}\triangle_x\left[\overline{V} + \tfrac{1}{2}\xi_\rho^{'}\triangle_y\left[V\right]\right]\right) * \left(\overline{\frac{\partial\left(\overline{H}+\frac{1}{2}\xi_\rho^{'}\triangle_y[H]\right)}{\partial y}} + \tfrac{1}{2}\xi_\rho^{'}\triangle_x\left[\frac{\partial\left(\overline{H}+\frac{1}{2}\xi_\rho^{'}\triangle_y[H]\right)}{\partial y}\right]\right)\end{aligned}$$

$$-\left(\overline{\overline{H}+\tfrac{1}{2}\xi'_\rho \triangle_y [H]} + \tfrac{1}{2}\xi'_\rho \triangle_x \left[\overline{H}+\tfrac{1}{2}\xi'_\rho \triangle_y [H]\right] \right) * \left(\begin{array}{c} \left(\frac{\partial\left(\overline{\overline{U}+\frac{1}{2}\xi'_\rho\triangle_y[U]}+\frac{1}{2}\xi'_\rho\triangle_x\left[\overline{U}+\frac{1}{2}\xi'_\rho\triangle_y[U]\right]\right)}{\partial x} \right) + \\ \left(\frac{\partial\left(\overline{V}+\frac{1}{2}\xi'_\rho\triangle_y[V]\right)}{\partial y} + \frac{1}{2}\xi'_\rho\triangle_x\left[\frac{\partial\left(\overline{V}+\frac{1}{2}\xi'_\rho\triangle_y[V]\right)}{\partial y}\right] \right) \end{array} \right)$$

$$\text{RsH} := \sum_{\rho=1}^{3} \left(\begin{array}{l} \frac{\triangle Y}{2} g_\rho \overline{\left(\sum_{\rho=1}^{3}\left(\frac{\triangle X}{2} g_\rho \left(\overline{(RsHcol)} - \frac{\xi'_\rho}{2}\triangle_x RsHcol\right)\right)\right)} \\ -\frac{\xi'_\rho}{2}\triangle_y \sum_{\rho=1}^{3}\left(\frac{\triangle X}{2} g_\rho \left(\overline{(RsHcol)} - \frac{\xi'_\rho}{2}\triangle_x RsHcol\right)\right) \end{array} \right)$$

2. Discretization: Discretization of the continuous equations by finite element method. Interpolation and Galerkin projection are used to discretize all spatial variables and its derivatives. Since extra information is needed for discretizing at boundary, the boundary condition is applied in this process. Output of this step are spatial discrete equations as

$$\text{RsHcol} := - \left(\begin{array}{c} 7.9545454545454545454549E-7 \\ * \left(\begin{array}{c} 1/2 * \left(H_{i+1,j} * (1-Gp_{jc}) + H_{i+1,j+1} * (Gp_{jc}+1) \right) \\ -1/2 * \left(H_{i,j} * (1-Gp_{jc}) + H_{i,j+1} * (Gp_{jc}+1) \right) \end{array} \right) \\ * \left(\begin{array}{c} 1/2 * (1-Gp_{ic}) * (U_{i,j} * (1-Gp_{jc}) + U_{i,j+1} * (Gp_{jc}+1)) \\ +1/2 * (Gp_{ic}+1) * (U_{i+1,j} * (1-Gp_{jc}) + U_{i+1,j+1} * (Gp_{jc}+1)) \end{array} \right) \end{array} \right)$$
$$- \left(\begin{array}{c} 1/4 * \left(\begin{array}{c} 1.5909090909090909091E-6 * (1-Gp_{ic}) * (H_{i,j+1} - H_{i,j}) \\ +1.5909090909090909091E-6 * (Gp_{ic}+1) * (H_{i+1,j+1} - H_{i+1,j}) \end{array} \right) \\ * \left(\begin{array}{c} 1/2 * (1-Gp_{ic}) * (V_{i,j} * (1-Gp_{jc}) + V_{i,j+1} * (Gp_{jc}+1)) \\ +1/2 * (Gp_{ic}+1) * (V_{i+1,j} * (1-Gp_{jc}) + V_{i+1,j+1} * (Gp_{jc}+1)) \end{array} \right) \end{array} \right)$$
$$- \left(\begin{array}{c} 1/2 * \left(\begin{array}{c} 1/2 * \left(\begin{array}{c} 1.5909090909090909091E-6* \\ \left(\begin{array}{c} (U_{i+1,j} * (1-Gp_{jc}) + U_{i+1,j+1} * (Gp_{jc}+1)) \\ -(U_{i,j} * (1-Gp_{jc}) + U_{i,j+1} * (Gp_{jc}+1)) \end{array} \right) \end{array} \right) \\ +1/2 * \left(\begin{array}{c} 1.5909090909090909091E-6* \\ \left(\begin{array}{c} (1-Gp_{ic}) * (V_{i,j+1} - V_{i,j}) \\ +(Gp_{ic}+1) * (V_{i+1,j+1} - V_{i+1,j}) \end{array} \right) \end{array} \right) \end{array} \right) \\ * \left(\begin{array}{c} 1/2 * (1-Gp_{ic}) * (H_{i,j} * (1-Gp_{jc}) + H_{i,j+1} * (Gp_{jc}+1)) \\ +1/2 * (Gp_{ic}+1) * (H_{i+1,j} * (1-Gp_{jc}) + H_{i+1,j+1} * (Gp_{jc}+1)) \end{array} \right) \end{array} \right)$$

$$\text{RsH}_{i,j} := \begin{cases} \left(\begin{array}{l} 1.5714285714285714285 6E5 * Gw_{jc} * (1-Gp_{jc})* \\ \begin{cases} \left(\begin{array}{l} 1.57142857142857142856E5 * (RsHcol_{i,ic,j,jc} * Gw_{ic} * (1-Gp_{ic}) \\ +RsHcol_{L-1,ic,j,jc} * Gw_{ic} * (Gp_{ic}+1)) \end{array} \right) & \textbf{if } i=1 \\ \left(\begin{array}{l} 1.57142857142857142856E5 * (RsHcol_{i,ic,j,jc} * Gw_{ic} * (1-Gp_{ic}) \\ +RsHcol_{i-1,ic,j,jc} * Gw_{ic} * (Gp_{ic}+1)) \end{array} \right) \end{cases} \end{array} \right) & \textbf{if } j=1 \\ \left(\begin{array}{l} 1.57142857142857142856E5 * Gw_{jc} * (1-Gp_{jc})* \\ \begin{cases} \left(\begin{array}{l} 1.57142857142857142856E5 * (RsHcol_{i,ic,j,jc} * Gw_{ic} * (1-Gp_{ic}) \\ +RsHcol_{L-1,ic,j,jc} * Gw_1 * (Gp_{ic}+1)) \end{array} \right) & \textbf{if } i=1 \\ \left(\begin{array}{l} 1.57142857142857142856E5 * (RsHcol_{i,ic,j,jc} * Gw_{ic} * (1-Gp_{ic}) \\ +RsHcol_{i-1,ic,j,jc} * Gw_{ic} * (Gp_{ic}+1)) \end{array} \right) \end{cases} \\ +1.57142857142857142856E5 * Gw_{jc} * (Gp_{jc}+1)* \\ \begin{cases} \left(\begin{array}{l} 1.57142857142857142856E5* \\ (RsHcol_{i,ic,j-1,jc} * Gw_{ic} * (1-Gp_{ic}) \\ +RsHcol_{L-1,ic,j-1,jc} * Gw_{ic} * (Gp_{ic}+1)) \end{array} \right) & \textbf{if } i=1 \\ \left(\begin{array}{l} (RsHcol_{i,ic,j-1,jc} * Gw_{ic} * (1-Gp_{ic}) \\ +RsHcol_{i-1,ic,j-1,jc} * Gw_{ic} * (Gp_{ic}+1)) \end{array} \right) \end{cases} \end{array} \right) \end{cases}$$

3. Common-subexpression elimination (CSE): The global common-subexpression eliminator constructs a dataflow graph for the intermediate code which is iteratively refined using a hardware cost model of the target computer architecture. The result of CSE is an optimized intermediate code in which the subexpressions which are repeatly used are computed and stored in temporary variables for later uses. After CSE, CTADEL gives the following output

$\text{t0}_{jc} := 1 - Gp_{jc}$
$t2_{jc} := Gp_{jc} + 1$
$t7_{ic} := 1 - Gp_{ic}$

$t12_{ic} := Gp_{ic} + 1$
$t13_{i,j,jc} := 1/2 * (U_{i,j} * t0_{jc} + U_{i,j+1} * t2_{jc})$
$t19_{i,j,jc} := 1.5909090909090909091E - 6 * (H_{i,j+1} - H_{i,j})$
$t26_{i,j,jc} := 1/2 * (V_{i,j} * t0_{jc} + V_{i,j+1} * t2_{jc})$
$t34_{i,j,jc} := 1.5909090909090909091E - 6 * (V_{i,j+1} - V_{i,j})$
$t5_{i,j,jc} := 1/2 * (H_{i,j} * t0_{jc} + H_{i,j+1} * t2_{jc})$

$$\begin{aligned} RsHcol_{i,ic,j,jc} := & -\left(\ 7.9545454545454545454549E - 7 * (t5_{i+1,j,jc} - t5_{i,j,jc}) * (t12_{ic} * t13_{i+1,j,jc} + t13_{i,j,jc} * t7_{ic})\ \right) \\ & - \left(\ 1/4 * (t12_{ic} * t19_{i+1,j,jc} + t19_{i,j,jc} * t7_{ic}) * (t12_{ic} * t26_{i+1,j,jc} + t26_{i,j,jc} * t7_{ic})\ \right) \\ & - \begin{pmatrix} 1/2 * \left(\ 1.5909090909090909091E - 6 * (t13_{i+1,j,jc} - t13_{i,j,jc})\ \right) \\ +1/2 * \left(\ t12_{ic} * t34_{i+1,j,jc} + t34_{i,j,jc} * t7_{ic} * (t12_{ic} * t5_{i+1,j,jc} + t5_{i,j,jc} * t7_{ic})\ \right) \end{pmatrix} \end{aligned}$$

$t46_{i,ic,j,jc} := RsHcol_{i,ic,j,jc} * Gw_{ic}$

$$t102_{i,ic,j,jc} := Gw_{jc} * 1.5714285714285714285 6E5 * \begin{cases} (t12_{ic} * t46_{L-1,ic,j,jc} + t46_{1,ic,j,jc} * t7_{ic}) \textbf{ if } i = 1 \\ (t12_{ic} * t46_{i-1,ic,j,jc} + t46_{i,ic,j,jc} * t7_{ic}) \end{cases}$$

$$RsH_{i,j} := \begin{cases} 1.571428571428571428 56E5 * (t0_{jc} * t102_{i,ic,1,jc}) \textbf{ if } j = 1 \\ 1.571428571428571428 56E5 * (t0_{jc} * t102_{i,ic,j,jc} + t2_{jc} * t102_{i,ic,j-1,jc}) \end{cases}$$

4. Code generation: From the optimized intermediate code, CTADEL produces the following Fortran code based on predefined templates and rulebases. A part of code generated by CTADEL as follows

```
      DO 119 jc=1,LCOL
       DO 120 j=0,L
        DO 121 ic=1,LCOL
         DO 122 i=0,L
          RsHcol(i,ic,j,jc)=(-7.9545454545454545454549E-7
     &          *(t5(i+1,j,jc)-t5(i,j,jc))*(t12(ic)*t13(i+1,j,jc)
     &          +t13(i,j,jc)*t7(ic)))-2.5E-1*(t12(ic)*t19(i+1,j,jc)+t19(i,j,jc)*t7(ic))
     &          *(t12(ic)*t26(i+1,j,jc)+t26(i,j,jc)*t7(ic))-5.0E-1*(1.5909090909090909091E-6
     &          *(t13(i+1,j,jc)-t13(i,j,jc))+5.0E-1*(t12(ic)*t34(i+1,j,jc)+t34(i,j,jc)
     &          *t7(ic)))*(t12(ic)*t5(i+1,j,jc)+t5(i,j,jc)*t7(ic))
122       CONTINUE
*        ENDDO
121      CONTINUE
*       ENDDO
120     CONTINUE
*      ENDDO
119    CONTINUE
*     ENDDO

      DO 147 j=1,L
       DO 148 i=1,L
        IF(0.EQ.1-j)THEN
         RsH(i,j)=1.5714285714285714285 6E5*(t0(jc)*t102(i,ic,1,jc))
         ELSE
         RsH(i,j)=1.5714285714285714285 6E5*(t0(jc)*t102(i,ic,j,jc)+t102(i,ic,j-1,jc)*t2(jc))
        ENDIF
148     CONTINUE
*      ENDDO
147    CONTINUE
*     ENDDO
```

6 Experiments

This section presents experiments with the code generated by CTADEL for the Shallow-Water equations based on Galerkin finite element methods. The original hand code [14] is considered as a reference. Firstly, we compare the output of the both codes to prove the correctness of the generated code. Next we assess the performance of the generated code by comparing execution times of the two codes.

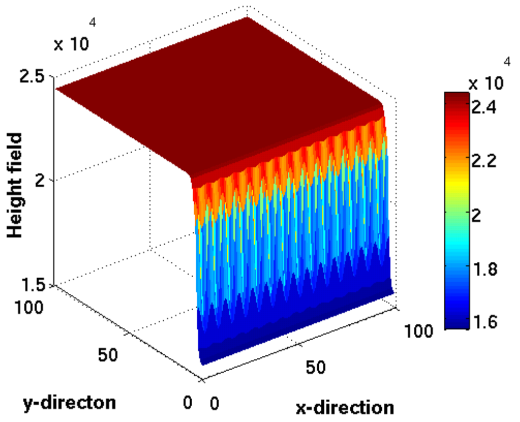

Figure 2. Height field value after 100 time steps.

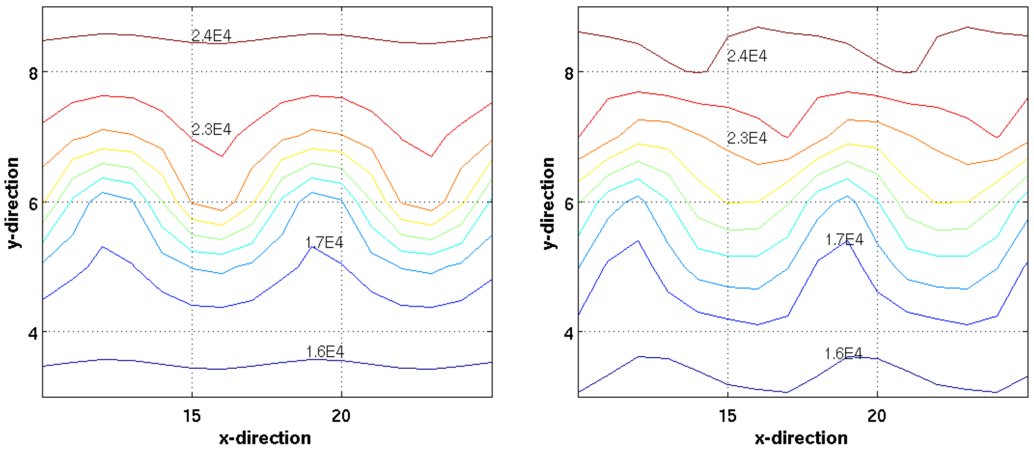

Figure 3. Height field: Initial value (left); Forecast after 100 time steps (right).

The experiments were performed on DAS3 [16]. We run program on one node with has 2.6GHz CPU and 4GB memory. The operation system is Scientific Linux. For compiling the program we used the g77 version 3.4.6 and the Pathscale version 3.0 compiler. In both cases we turned on the optimization option (-O2).

The solid boundary conditions were implemented in the y direction. In the x direction we used periodic boundary conditions. This presents physically a channel around the Earth at midlatitudes [10] (see Figure 2). The initial values as follows

$$\begin{aligned} &H\left(X,Y\right) = H_0 + H_1 tanh\frac{9(Y-Y_0)}{2D} + H_2\frac{1}{cosh^2(\frac{9Y-Y_0}{D})} sin 2\pi x, \\ &U = -\frac{1}{f}\frac{\partial H}{\partial Y}, \\ &V = -\frac{1}{f}\frac{\partial H}{\partial X}. \end{aligned} \tag{23}$$

The parameters were chosen same to [4, 5, 14]. Figures 3 and 4 (left) show the initial values of the height and the velocity field, respectively. The predicted values after 100 time steps (20.000 second) are shown in Figures 3 and 4 (right).

In order to check the correctness of the generated code, we compare the output of the generated code with the one of the hand code. Figure 5 presents the differences between these outputs. These results were obtained by subtracting the H field values returned by the hand code and those returned by the generated code. We tested with the domain

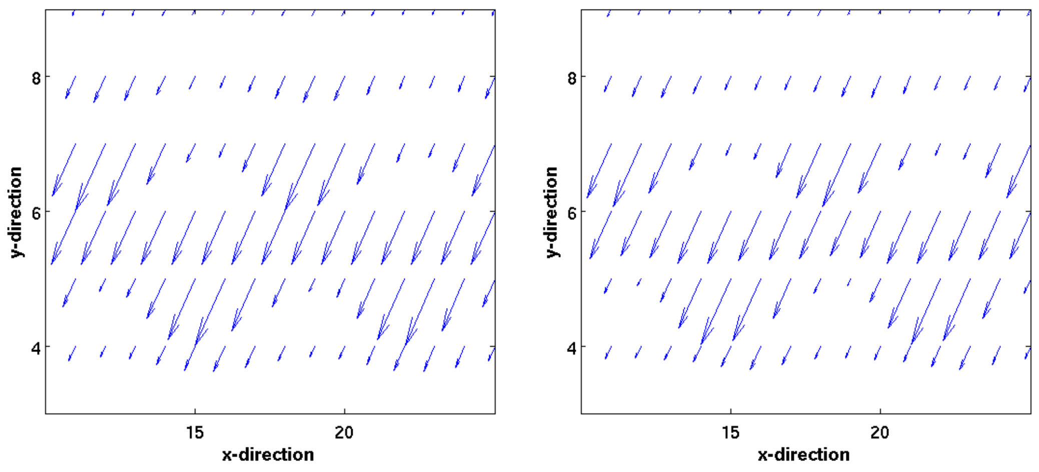

Figure 4. Velocity field: Initial value (left); Forecast after 100 time steps (right).

of 100x100 grid points and 100 time steps. From Figure 5 we can conclude that the differences between the outputs of the hand code and the generated code are very small. Figure 6 shows the total execution times of the generated

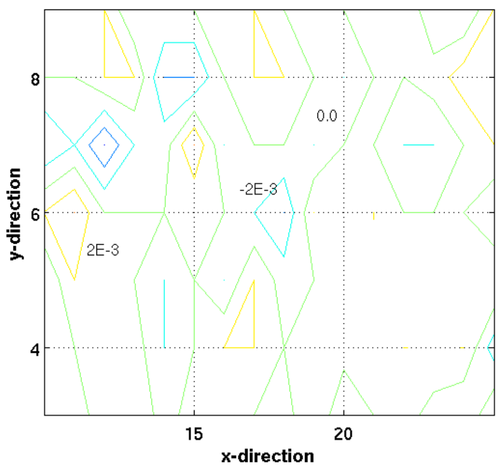

Figure 5. Differences between the hand code and generated code.

and the hand code compiled by the g77 compiler (left) and the Pathscale compiler (right), where the domain varies from 10000 to 90000 grid points. We see that, with both compilers, the generated code is much faster than the hand code. With 90000 grid points, the generated code has a performance gain upto two-thirds over the hand code. These results are the consequence of the fact that the hand code was not well optimized, whereas CTADEL which performs optimization like common subexpression elimination can generate a more efficient code.

7 Conclusions and Future work

Previous studies such as [2, 3, 8, 7] have shown that CTADEL can generate code for several schemes in weather forecast models using finite difference methods. In this paper we have proven that with some extensions, CTADEL is able to generate code for Galerkin finite element schemes applied to the Shallow-Water equations. Furthermore, the code generated by CTADEL is more efficient than the hand code. This is due to the hand code was not well optimized,

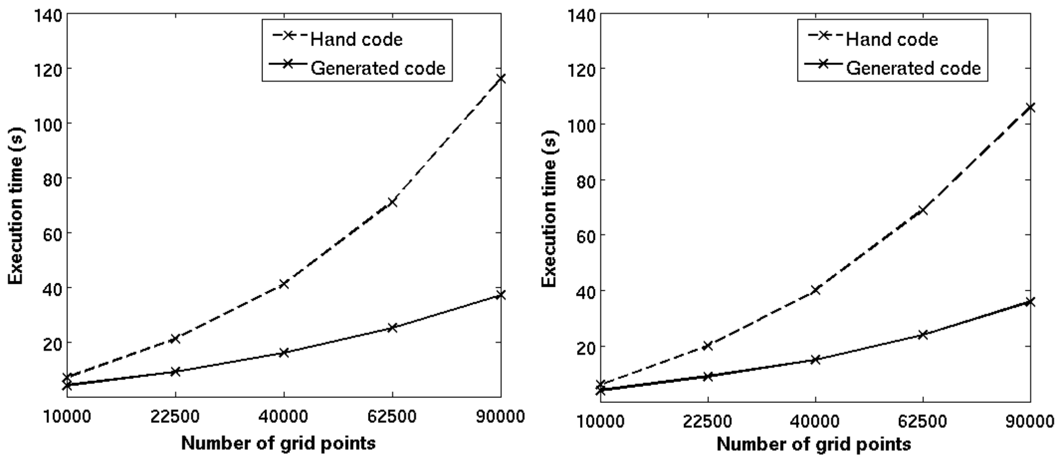

Figure 6. Execution times of the hand code and generated code with the g77 compiler (left) and the Pathscale compiler (right) using -O2 optimization option.

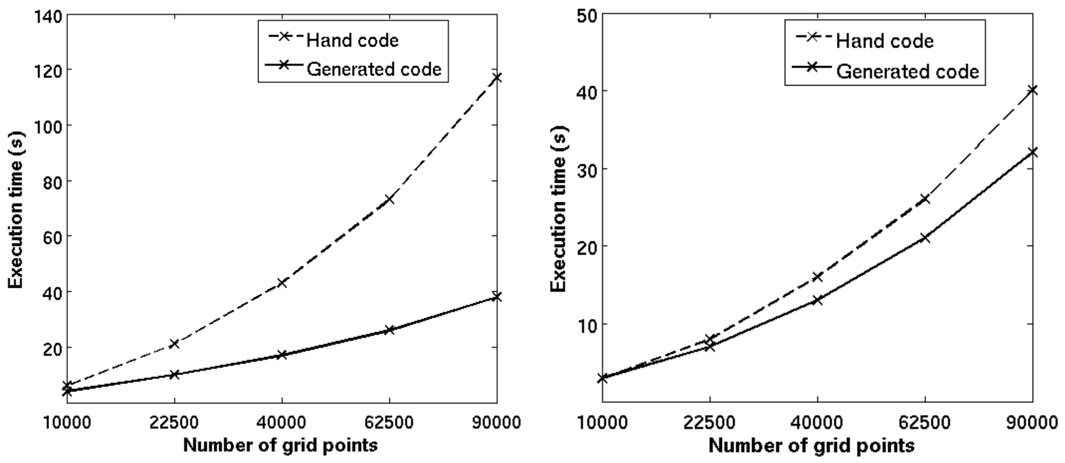

Figure 7. Execution times of the hand code and generated code with the g77 compiler (left) and the Pathscale compiler (right) using -O3 optimization option.

whereas CTADEL, which takes optimization such as common subexpression elimination into account, can generate an efficient code that can compete with optimized hand code.

In the generated code, a set of temporary variables were used which increase the use of memory. In fact, the memory used in the generated code is 1.3 time larger than the memory which the hand code uses. With very large domain problem, this may become a problem since memory is usually not an unlimited resource in most computers. Further research on CTADEL may focus on how to reuse these variables.

Mesh generation is an important step of the finite element methods for numerical computation. Currently CTADEL uses regular rectangular grid. In the next step, we will extend CTADEL to generate complex mesh such as structured or unstructured meshes.

Our experiments were performed on one processor. In the future we will generate parallel code and run program on other environments like clusters and grids in order to confirm that CTADEL can generate efficient code for different computer architectures.

References

[1] Abramowitz, M., and Stegun, I. A., 1970, Handbook of mathematical functions: Dover Publ., New York, 1046p.

[2] R.A. van Engelen. Ctadel: A Generator of Effcient Numerical Codes. PhD thesis, Universiteit Leiden,1998.

[3] R.A. van Engelen, Lex Wolters, and Gerard Cats. Ctadel: A generator of multiplatform high performance codes for pde-based scientific applications. In Proceedings of the 10th International Conference on Supercomputing, pages 8693, Philadelphia, USA, May 1996. ACM.

[4] Grammeltvedt, A., 1969, A survey of finite-difference schemes for the primitive equations for a barotropic fluid: Mon. Wea. Rev., v. 97, no. 5, p. 384-404.

[5] Gustafsson, B., 1971, An alternative direction implicit method for solving the shallow-water equations: Jour. Comput. Phys., v. 7, no. 2, p. 239-251.

[6] HIRLAM home page. http://hirlam.org/

[7] P. van der Mark, R. van Engelen, K. Gallivan, and W. Dewar. A Case Study for Automatic Code Generation on a Coupled OceanAtmosphere Model. In Proceedings of the 2002 International Conference on Computational Science (ICCS02), pp 419-419.

[8] P. van der Mark, Gerard Cats, and Lex Wolters. Automatic code generation for a turbulence scheme. In Proceedings of the 15th International Conference of Supercomputing, pages 252259, Sorrento, Italy, June 2001. ACM.

[9] Navon, I. M., 1979, Finite-element simulation of the shallow-water equations model on a limited area domain: Appl. Math. Modell., v. 3, no. 1, p. 337-348.

[10] Navon, I. M., 1987, FEUDX: A two-stage, high-accuracy, finite-element FORTRAN program for solving shallowwater equations: Computers Geosciences, v. 13, no. 3, p. 255-285.

[11] Sadourny, R., 1975, The dynamics of finite-difference models of the shallow-water equations: Jour. Atm. Sci., v. 32, no. 4, p. 680 689.

[12] Staniforth, A. N., and Beaudoin, C., 1986, On the efficient evaluation of certainintegrals in the Galerkin F. E. Method: Intern. Jour: Num. Meth. Fluid, v. 6, no. 2, p. 317-324.

[13] Steppeler, J., Energy conserving Galerkin finite element schemes, for the primitive equations of numerical weather prediction. Journal of Computational Physics, Vol. 69, No. l, p. 258-264, 1987.

[14] Steppeler, J., FE2DY: A finite element Fortran program for the solution of the Shallow-Water equations with energy conservation. Computers & Geoseiences Vol. 16, No. 5, pp. 645 67, 1990.

[15] Steppeler, J., 1988, A Galerkin finite-element-spectral weather forecast model in hybrid coordinates: Comput. Math. Applics., v. 16, no. 1/’2, p. 23-30.

[16] The Distributed ASCI Supercomputer 3 (DAS-3) :http://www.cs.vu.nl/das3/