Optimization of the Finite Volume Method Source Code by using Polymorphism

R. Leithner, H. Zindler and A. Hauschke

***Abstract*—**Often CFD programs are used for solving flow problems that are based on finite volume methods (FVM). The FVM solves the balance equations in an iterative process. Since the single balances are coupled, different coupling methods like the SIMPLER (Semi Implicit Method for Pressure Linked Equations Revised) are used. Since the solving algorithm is passed through several times during the iteration, all time critical branching like *if* statements should be avoided. But branching appears several times, because of the different handling of volume elements in the middle and volume elements with boundary conditions. This differentiation can be done once during the initialization of the algorithm and it is not necessary to repeat the differentiation several times during the iteration. For example the sorting of the calculation functions can be done by the polymorphism of object orientated program languages like C++.

***Keywords*—** C++, Optimization, Polymorphism, SIMPLER

I. Introduction

FOR solving flow problems like in heat exchangers partial differential equations for momentum, mass and energy appear. These balances are solved by using the finite volume method that discretizes the balance equations over location and time. The single balance equations are coupled by velocity, pressure and density. For solving this coupling several algorithms for solving the momentum and mass balance equations at the same time are developed. The algorithm that is used as an example in this article is the SIMPLER (Semi Implicit Method for Pressure Linked Equations Revised). All solving algorithms work in an iterative loop that solves the single balance equations one after the other.

To set up the balance equations for each volume element several coefficients have to be calculated (see Fig. 2) that differ from each other depending on the time period and the location of the volume element. Fig. 1 shows the array of the volume elements for the case of an one-dimensional transient flow through a pipe. Two boundary condition types are regarded:

1. constant pressure at both sides of the pipe
2. constant velocity at the inlet and constant pressure at the outlet of the pipe

If the position of a volume element is checked during the iteration to find the correct calculation instructions for the coefficients of the balance equations, the algorithms would be slow and difficult to understand. Therefore a solution, using modern object orientated program languages, needs to be found, that orders the calculation instructions of the coefficients of the balance equations in the initial phase before the iteration starts. During the iteration no further checking is required. An efficient kernel can be achieved by an analysis of the SIMPLER and of the virtual inheritance of object orientated program languages.

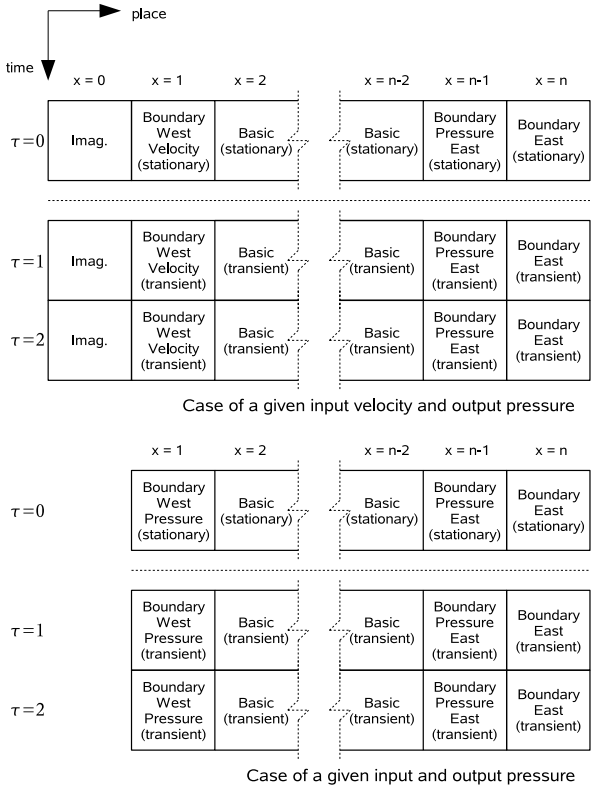

Fig. 1 Arrays of the volume elements with different boundary conditions; Source: [8]

Manuscript submitted November 27,2006; Revised April 15, 2007

Zindler, H. Dipl. Ing. is with Institute of Heat- and Fuel Technology, Braunschweig, CO 38106 Germany (corresponding author to provide phone: ++49-531-3913032; fax: ++49-531-391-5932; e-mail: h.zindler@ tu-bs.de).

Leithner, R. Prof. Dr. techn. is with Institute of Heat- and Fuel Technology, Braunschweig, CO 38106 Germany (e-mail: r.leithner@tu-bs.de).

Hauschke, A. Dipl. Ing. is with Institute of Heat- and Fuel Technology, Braunschweig, CO 38106 Germany (e-mail: a.hauschke@ tu-bs.de).

Momentum balance / pseudo velocity								
coefficient	transient / standard	stationary	imag.	b. west	b. west p	b. west w	b. east	b. east p
a^0_{ei}	$\frac{\rho^0_{i+\frac{1}{2}} A_{i+\frac{1}{2}} \Delta x_{i+\frac{1}{2}}}{\Delta t}$	0	-	standard	standard	standard	standard	standard
b^0_{ei}	$a^0_{ei} w^0_{i+\frac{1}{2}}$	0	-	standard	standard	standard	standard	standard
b_{ei}	$\Delta p_{H,i+\frac{1}{2}} A_{i+\frac{1}{2}} + b^0_{ei} + dp_{pi}$	standard	-	standard	standard	standard	standard	standard
dp_{pi}	$(p_i - p_{i+1}) A_{i+\frac{1}{2}}$	standard	-	standard	standard	standard	standard	standard
a_{wi}	$\max\left[(\rho\tilde{w})_{i-\frac{1}{2}}, 0\right] A_{i-\frac{1}{2}}$	standard	-	standard	0	standard	standard	standard
a_{eei}	$\max\left[-(\rho\tilde{w})_{i+\frac{3}{2}}, 0\right] A_{i+\frac{3}{2}}$	standard	-	standard	standard	0	standard	0
a_{ei}	$a_{wi} + a_{ei} + a^0_{ei} + \frac{\lvert\Delta p_{R,i+\frac{1}{2}}\rvert}{\lvert w_{i+\frac{1}{2}}\rvert} A_{i+\frac{1}{2}}$	standard	-	standard	standard	standard	standard	standard
pressure correction / calculation of pressure feld								
coefficient	transient / standard	stationary	imag.	b. west	b. west p	b. west w	b. east	b. east p
b^0_{mi}	$(\rho^0_i - \rho_i)\frac{A_i \Delta x_i}{\Delta t}$	0	-	standard	standard	standard	standard	standard
b_{mi}	$b^0_{mi} + (\rho w^* A)_{i-\frac{1}{2}} - (\rho w^* A)_{i+\frac{1}{2}}$	standard	-	standard	0	standard	0	standard
$b_{mi,pseudo}$	$b^0_{mi} + (\rho\tilde{w} A)_{i-\frac{1}{2}} - (\rho\tilde{w} A)_{i+\frac{1}{2}}$	standard	-	standard	p_1	standard	p_n	standard
a_{mWi}	$(\rho A)_{i-\frac{1}{2}} \frac{A_{i-\frac{1}{2}}}{a_{ei-1}}$	standard	-	standard	0	0	0	standard
a_{mEi}	$(\rho A)_{i+\frac{1}{2}} \frac{A_{i+\frac{1}{2}}}{a_{ei}}$	standard	-	standard	0	standard	0	standard
a_{mPi}	$a_{mWi} + a_{mEi}$	standard	-	standard	1	standard	1	standard
energy balance								
coefficient	transient / standard	stationary	imag.	b. west	b. west p	b. west w	b. east	b. east p
a^0_{hPi}	$\frac{\rho^0_i A_i \Delta x_i}{\Delta t}$	0	-	standard	standard	standard	standard	standard
b^0_{hi}	$a^0_{hPi} h^0_i$	0	-	standard	standard	standard	standard	standard
b_{hi}	$\max\left[\dot{Q}, 0\right] + b^0_{hi}$	standard	-	$(w_{\frac{3}{2}} > 0)?h_1$:standard	standard	standard	$(w_{n-\frac{1}{2}} < 0)?h_n$:standard	standard
a_{hWi}	$\max\left[(\rho w)_{i-\frac{1}{2}}, 0\right] A_{i-\frac{1}{2}}$	standard	-	0	standard	standard	$(w_{n-\frac{1}{2}} < 0)?0$:standard	standard
a_{hEi}	$\max\left[-(\rho w)_{i+\frac{1}{2}}, 0\right] A_{i+\frac{1}{2}}$	standard	-	$(w_{\frac{3}{2}} > 0)?0$:standard	standard	standard	0	standard
a_{hPi}	$a_{hWi} + a_{hEi} + a^0_{hPi} + \frac{\max[-\dot{Q},0]}{h_i}$	standard	-	$(w_{\frac{3}{2}} > 0)?1$:standard	standard	standard	$(w_{n-\frac{1}{2}} < 0)?1$:standard	standard

Fig. 2 Arrays of the volume elements with different boundary conditions; Source: [8]

II. Analysis of the SIMPLER-Algorithm

The SIMPLER-Algorithm was published by Patankar in [2]. Walter has found in [3] a very good representation of the SIMPLER-Algorithm for one-dimensional transient pipe flows that is used in this article. There is just one difference in the calculation of the source term. The transient summand of the source term is calculated in a separate operation. All calculation instructions of all coefficients are listed in Fig. 2.

The iteration of the SIMPLER-Algorithm for a transient one-dimensional flow through a pipe can be explained as follows:

1) With the help of an estimated velocity array $w^*_{i+0.5}$, an estimated pressure array p^*_i and an estimated temperature array T^*_i the array of the pseudo velocity $\tilde{w}_{i+0.5}$ is calculated.

$$\tilde{w}_{i+\frac{1}{2}} = \frac{a_{eei} w^*_{i+\frac{3}{2}} + a_{wi} w^*_{i-\frac{1}{2}} + b_{ei}}{a_{ei}} \tag{1}$$

2) The pressure array p_i is calculated by using the array of the pseudo velocity $\tilde{w}_i$.

$$a_{mPi} p_i = a_{mWi} p_{i-1} + a_{mEi} p_{i+1} + b_{ei} \tag{2}$$

3) The estimated velocity array $w^*_{i+0.5}$ is calculated.

$$a_{ei} w^*_{i+0.5} = a_{wi} w^*_{i-0.5} + a_{eei} w^*_{i+1.5} + b_{ei} \tag{3}$$

4) With the improved velocity array $w^*_{i+0.5}$ the pressure correction array $\hat{p}_i$ is calculated.

$$a_{mPi} \hat{p}_i = a_{mWi} \hat{p}_{i-1} + a_{mEi} \hat{p}_{i+1} + b_{ei} \tag{4}$$

5) The pressure correction array $\hat{p}_i$ is used to calculate the velocity array $w_{i+0.5}$.

$$w_{i+\frac{1}{2}} = w^*_{i+\frac{1}{2}} + \frac{A_{i+\frac{1}{2}}}{a_{ei}} (\hat{p}_i - \hat{p}_{i+1}) \tag{5}$$

6) The energy balance is solved.

$$a_{mPi} h_i = a_{hWi} h_{i-1} + a_{hEi} h_{i+1} + b_{hi} \tag{6}$$

7) All other physical state variables like density or viscosity are recalculated.

8) Return to the step 1) and repeat the entire procedure until a converged solution is obtained.

There are temporal boundary conditions (stationary, transient) and local boundary conditions (pipe inlet, pipe outlet) that holds different constant physical values like pressure, velocity or enthalpy. Fig. 1 shows that a coefficient holds a local or temporal boundary condition, but never a local and temporal boundary condition at the same time. This fact can be used to implement efficiently volume elements with combined boundary conditions (temporal and local) at the same time by using the virtual inheritance (see [4]).

III. Polymorphism and Virtual Inheritance

The polymorphism is beside the encapsulation and the inheritance one of the three most important parts of object orientated programming. It is a mechanism to implement software interfaces. That means, that an object has the same syntax but a different semantic is hidden behind this syntax. This mechanism is useful during iterations, when the calculation of coefficients looks always the same, but the coefficients are calculated internally in a different way depending on the boundary conditions. The C++ programming language uses the mechanism of the inheritance and the key word *virtual*.

In Fig.3 is represented an example of a polymorphic allocation. The basic class *CalcKoeff* defines a virtual method *calc*. A class *CalcKoeffA* is derived from the class *CalcKoeff* and overwrites the virtual methods. A pointer to the object of the classes *CalcKoeff* and *CalcKoeffA* would always call the own method of the object.

```
CCalcKoeff *z_K = new CCalcKeoff;
CCalcKoeffA *z_KA = new CCalcKoeffA;

std::cout << z_K->calc(...);
// Output:  0.0
std::cout << z_KA->calc(...);
// Output: -3.0
```

The polymorphism respectively the polymorphic allocation is defined as the setting of a pointer of the type *CCalcKoeff* to an object of the type of the derived class *CCalcKoeffA*. Whereas accessing a method that is defined by the key word *virtual*, the pointer accesses the overwritten method of the derived class.

```
CCalcKoeffA *z_KA = new CalcKoeffA;
// polymorphic allocation
CCalcKoeff  *z_K  = (CalcKoeff*) z_KA;
z_K->calc(...);
// output: -3.0
```

Using this mechanism it is possible to define several classes that are derived from the basic class that implements their own specific methods to calculate the coefficients depending on the position of the volume element. The instances of the derived classes can be collected in an array of the type of the basis class. In this way the calculation methods can be evaluated in a loop.

```
// instances of the derived classes
CCalcKoeffA *z_KA = new CCalcKoeffA;
CCalcKoeffB *z_KB = new CCalcKoeffB;
...

// array of the basic class
CCalcKoeff **z_K_array = new CCalcKoeff*[n];

// polymorphic allocation
 z_K_array[0] = (CCalcKoeff*) z_KA;
 z_K_array[1] = (CCalcKoeff*) z_KB;
...

// calculation of the coefficient in a loop
for (int i = 0; i < n; i++)
  cout << z_K_array[i]->calc(...) << endl;
```

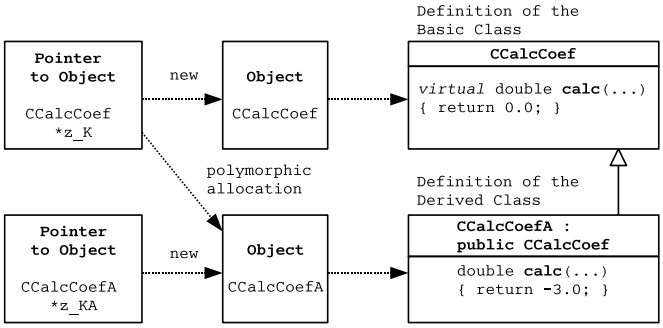

Fig. 3 Example of a polymorphic allocation; Source: [8]

IV. Solving the Problem by Virtual Inheritance

Using the polymorphism it is possible to overwrite single function of the basic class. This functionality would be sufficient if just a temporal or a local boundary condition exists. But there are volume elements that hold both boundary conditions at the same time. This difficulty can be overcome by using the virtual inheritance. The polymorphic allocation uses the key word *virtual* with functions of the basic class. The virtual inheritance uses the key word *virtual* with classes.

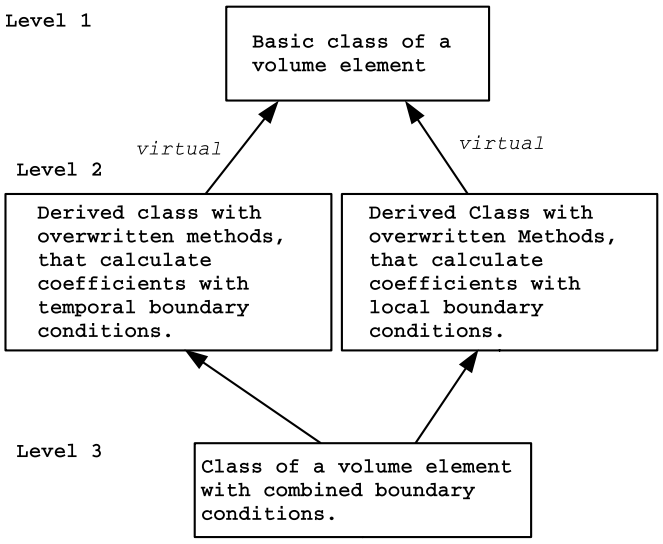

Fig. 4 Principle of the virtual inheritance; Source: [8]

A basic class is constructed, that holds all methods that calculate the coefficients of the general case of a middle transient volume element and holds all physical state variables (level 1). Virtual classes are derived from the basic class, that overwrite methods for calculating coefficients for implementing boundary conditions that are described in Fig. 2 (level 2).

Each virtual class implements just one temporal or local boundary condition. A third group of classes combines temporal and local boundary conditions by the derivation of two classes at the same time, that hold one temporal and one local boundary condition (level 3). To avoid ambiguity the classes of level 2 must be derived virtual from the basic class because if that classes would be derived in the normal way, the classes of the third level access different variables of the physical state. A simplified inheritance scheme is shown in Fig. 4 and the full inheritance scheme is shown in Fig. 5

To save the derived classes a two-dimensional matrix (time, location) of pointers of the basic class is used. The pointers of the basic class are allocated to objects of derived classes in the initial phase. In this way the coefficients could be calculated in a loop without if-statements.

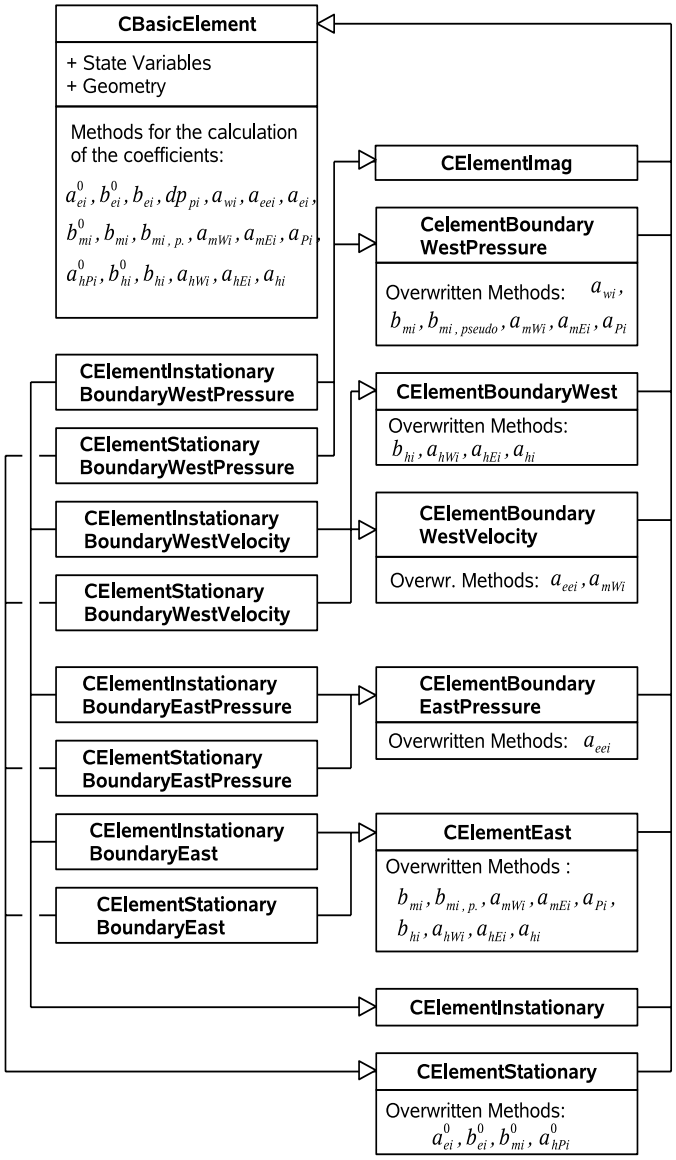

Fig. 5 Class diagram for the FVM; Source: [8]

V. Example

To check the efficiency of the new program design a comparison between the old and the new program design was made. The algorithm of the old program design has to check the type of each volume element in the loop before choosing the set of methods to calculate the coefficients.

The calculation example is a transient one-dimensional flow through a pipe of a super heater of a steam power plant like it is shown in Fig. 6.

The super heater is discretized by 10 volume elements. The flue gas is calculated quasi-stationary.

The convergence behavior of the algorithm is not affected by the new program design. Therefore the convergence

conditions are not checked any more and the algorithm runs just 1.000.000 times to compare the calculation speed.

The gas and steam properties are read from tables to reduce the influence of gas and steam properties to the calculation speed.

With the new program design the calculation velocity could be improved by 7 \%.

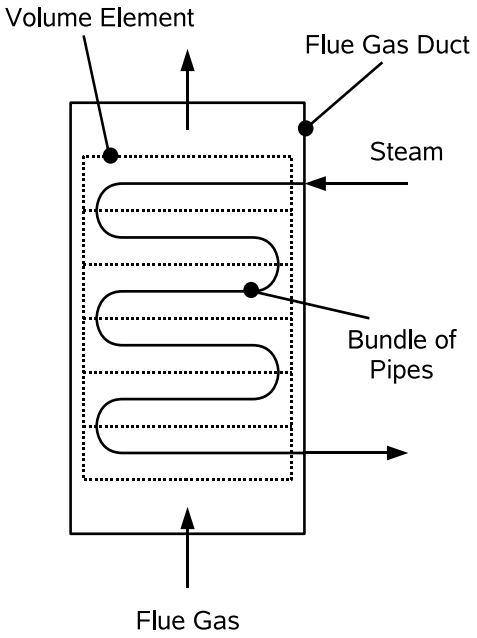

Fig. 6 Distribution of the volume elements over the heat exchanger; Source: [8]

CONCLUTION

The advantage of calculation velocity of the presented program design is poor, because most of the calculation time is used for calculating the properties of steam and gases and solving the linear equation systems. But with the slightly enhanced effort in program design, a very clear program code is generated, that concentrates all calculation methods of a special boundary condition in a specific class. The advantage in calculation time is low in one-dimensional cases like it is chosen here. But in cases of three-dimensional flow, difficult geometries and constant gas properties the improvement will increase.

Index of formulae

Symbol	Unit	Description
A	m^2	area
a	$var.$	coefficient
b	$var.$	source term
h	$\frac{J}{kg}$	enthalpy
p	Pa	pressure
t	s	time
w	$\frac{m}{s}$	velocity
x	m	location
Δ	$-$	difference
ρ	$\frac{kg}{m^3}$	density
x_i	$-$	index of location
x_e	$-$	eastern bound. of the vol. el.
x_w	$-$	western bound. of the vol. el.
x_E	$-$	eastern volume element
x_W	$-$	western volume element
x_P	$-$	central volume element
x_m	$-$	mass
x_h	$-$	enthalpy
x^*	$-$	estimated variable
$\hat{x}$	$-$	correction variable
x^0	$-$	from last time step
$\tilde{x}$	$-$	pseudo

References

[1] T.Kato, Non--stationary flows of viscous and ideal fluids in R3, J.Func.Anal. 9, 1972, pp. 296--305.

[2] S.Patankar, Numerical Heat Transfer and Fluid Flow, Hemisphere Publishing Corporation, 1980

[3] H. Walter, Modellbildung und numerische Simulation von Natur umlaufdampferzeugern, Fortschritt-Berichte VDI Reihe 6 Energie technik Nr.: 457, Wien 2001

[4] B. Stroustrup, Die C++-Programmiersprache, Addison-Wesley, 1998

[5] T.Löhr, Simulation stationaerer und instationaerer Betriebs zustaende kombinierter Gas- und Dampfturbinenanlagen, VDI-Fortschritt-Berichte Reihe 6 Nr.: 432, Braunschweig 1999

[6] H. Rohse, Untersuchung der Vorgänge beim Uebergang vom Umwaeltz- zum Zwangsdurchlaufbetrieb mit einer dynamischen Dampferzeugersimulation, VDI-Fortschritt-Berichte Reihe 6 Nr.: 327, Wien/Braunschweig 1995

[7] K. Ponweiser, Numerische Simulation von dynamischen Stroe mungsvorgaengen in netzwerkartigen Rohrkonstruktionen, VDI-Fortschritt-Berichte Reihe 6 Nr.: 378, Wiener Neustadt 1997

[8] H. Zindler, Dynamische Kraftwerkssimulation durch Kopplung von FVM und PECE Verfahren mit Hilfe von Adjungiertenverfahren, VDI Verlag