An object-oriented parallel finite-volume CFD code

Dulceneia Becker

Innovative Computing Laboratory
July 9, 2010
Summary

1. Background
2. Introduction
3. Conceptual Model
4. Code Overview
5. Mixing Plane Model
6. Numerical Results
Background
Education & Experience

1991

Unisc
Optimization

1994

UFRGS
DBG MRES

1999

Cranfield
DCG

2002

ITA
CFD Turbomachinery

2006

ICL/UTK

2010

Gráfica e Adesivos Lajeado Ltda
Several Positions

National Supercomputing Centre / UFRGS
PRRQR

Cranfield University and AspenTech Group
32 to 64-bit

Universidade Regional Integrada
Lecturer

UFU
AMR
Multiphase

iTSix Tecnologia
FEA
Introduction
Goals

- Parallelize a three-dimensional unstructured finite-volume computational fluid dynamics (CFD) code developed at the Technological Institute of Aeronautics - ITA, hereafter called GT77
- Fully port the CFD code from a Fortran 77/90 to a Fortran 90 version, making use of the new features, especially dynamic memory allocation
- Introduce the object-oriented programming paradigm aiming to enhance code reuse and increase efficiency during application development
- Develop a model to be used by other researchers of the group
GT77 & GT90

- GT77 was conceived to solve the Navier-Stokes equations for compressible flows using unstructured meshes and finite volume methods.
- The primary goal is to simulate axial compressors using CFD techniques.
- The original serial code is written in a Fortran 77 fashion though recent added subroutines include Fortran 90 features.
- GT90, the re-implemented code, uses a message-passing programming model and an object-oriented paradigm and is written in Fortran 90 and MPI.
- Some routines needed minor changes and others major redesign.
Conceptual Model
The conceptual model considers the implementation of several types of elements (hexahedron, tetrahedron, etc), meshes and methods, including adaptive mesh refinement.

The mesh type delineates the behaviour of the entire simulation.

Depending on the chosen mesh, each method performs in a specific manner.

Ideally, this would be done by using dynamic dispatching or run-time polymorphism. However, this is one of the features not directly implemented in Fortran 90 and its emulation, though possible through a polymorphic class, can be costly and add an excessive overhead.
Several modules with the same name mtype are implemented for every mesh type and a library is created for each mesh type at compilation.

Considering that two mesh types are implemented and written in the files hexa8.f90 and tet4.f90

- Although the file names are different, both modules are called mtype.
- At compilation level, a library is generated for each mesh-type file, namely libhexa8.a and libtet4.a.
- The appropriate library must be linked to the main program.

The overhead caused by a polymorphic class is completely eliminated but the polymorphism is kept to a level.

Every class that depends on the mesh type inherits the mesh object through a use statement to mtype.
Class mtype

! HEXA8.F90 !

module mtype
use geometry_datatypes
implicit none

type(geom_hexa8_def) :: edef

type(geom_hexa8_mesh), pointer :: mesh
type(geom_edesc), parameter :: edesc=hexa8_desc

end module mtype
Code Overview
Current implementation of GT90

- Solves the Euler and Navier-Stokes equations and the one-equation Spalart-Allmaras turbulence model
- The governing equations are solved with the explicit Runge-Kutta scheme
- The Maccomark predictor-corrector scheme is also implemented but has not been used
- The Spalart-Almaras turbulence model is implemented with the Bi-conjugate Gradient method
- Each scheme is allocated to a class accordingly to its purpose
Parallelization

- Single Program Multiple Data (SPMD) programming model
- MPI (Message Passing Interface)
- Domain decomposition

The term domain decomposition, has slightly different meanings. In this context, it means the separation of the physical domain into regions, i.e. the partitioning of the mesh into parts.
Mesh Partition

- It is relatively simple to partition a structured grid. However, the partition of unstructured meshes is not trivial and is an area of research in itself.
- A large number of efficient partitioning heuristics have been developed during recent years and there are several mesh partition programs available.
- METIS was chosen for the present work mainly due to its availability and approval in the research community.
- The mesh partition in GT90 can be divided into two steps:
  1. Using METIS the domain $\Omega$ is partitioned into $n_p$ (number of processors) non-overlapping subdomains $\Omega_p$.
  2. An overlapping is added to each subdomain, according to the amount specified.
Mesh Partition Sketch

- Local nodes
- Halo nodes
Communication Among Adjacent Subdomains

- Types of communication:
  - Between two subdomains (point-to-point)
  - Among all subdomains (collective)
  - Among adjacent subdomains

- The two first are handled efficiently by the message-passing library
- The latter is not directly supported
- To date, only one communication scheme was implemented in GT90
- Another scheme, used in a similar topology, reduces the number of neighbours but not necessarily reduces the communication time
Communication Schemes

Figure: Squared domain divided into four parts.
Figure: Processors connectivity for two communication schemes.

- Scheme 2 reduces communication but the execution time is not consistently lower
- Synchronous mode imposed at scheme 2
The solver deals with three different data types: scalar values, vectors and matrices. Only the latter two are liable to data partitioning.

The data partition adopted renders the spatial discretization step to be altogether local, i.e. it does not involve data transfer among subdomains.

The temporal discretization needs the updating of the halo nodes and hence communication among processes.

The current implementation updates the halo nodes every time they are changed. However, preliminary tests have shown that this updating can be done at every few steps for the cases tested. Further investigation must be carried out in order to extent these results and also to find an optimum value.
Mixing Plane Model
Averages the flow properties circumferentially between the rotating blade passage and the steady vane passage.
Mixing Plane Interface

Inlet

R1

S1

R2

S2

Outlet

C2

C3

C4

C5

C6

Axis of Rotation
How does it work

The interface is divided in parts as concentric circumferences with radius $r + c\Delta$, where

$$\Delta = \frac{R - r}{nd}$$

- $c$ is the circumference number
- $r$ and $R$ are the smallest and biggest radii
- $nd$ is the number of divisions or parts
Mixing Plane Divisions
Two main parts:

• Setup: set divisions and the division number of each element in the mixing plane
• Calculate Q - variables vector

The setup is executed only once at initialization
Q is recalculated every time boundary conditions are applied
Setup

- Setup of mixing plane is relatively simple in the serial code
- In parallel there may be the need of exchanging data among processes
- Current mesh partitioning does not take into account geometric features and hence each mixing plane interface may be distributed over several processes
- A communicator is created during setup to be used on updating $Q$

Load balance can be quite dramatically affected by the mixing plane.
\( Q \) Calculation

- \( Q \) is initially calculated per division per process, i.e. each process calculates a partial \( Q_p \)
- All \( Q_p \) are added using \texttt{mpi\_allreduce} and the mixing plane communicator created at setup:
  \[
  Q^{MP} = \sum_{p=1}^{np} Q_p
  \]
- \( Q^{MP} \) is used to apply the appropriate boundary conditions at mixing plane
Numerical Results
Convergence

- In order to verify and validate the code, several tests were carried out using problems that were already used to validate the original version of the code.
- Both serial and parallel versions of the new code produced convergence curves identical or very similar to the original code.
- Notice that these test cases aimed to validate GT90 by comparison to GT77, assuming that the results produced by GT77 are correct.
Figure: GT90 convergence of laval nozzle test case with inlet velocities: 10, 200 and 500 m/s
In terms of CPU time, GT90 may be considerably faster than GT77. In some cases, a reduction of about 90% in the execution time was observed. This only happens for specific cases due to the improved data structure and the execution of only the strictly necessary procedures. In complex cases, a reduction in time is also observed but not at the same level.
The setup and post-processing routines of GT77 represent a considerable fraction of the total time.

The setup and post-processing time increases dramatically when the mesh size increases.

<table>
<thead>
<tr>
<th>Event</th>
<th>mesh 1</th>
<th>mesh 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>elements</td>
<td>3078</td>
<td>303750</td>
</tr>
<tr>
<td>nodes</td>
<td>3900</td>
<td>319056</td>
</tr>
<tr>
<td>iteration time</td>
<td>0.44</td>
<td>13.51</td>
</tr>
<tr>
<td>setup and post-processing time</td>
<td>2.28</td>
<td>15015.31</td>
</tr>
</tbody>
</table>
Some procedures perform operations among every facet of a given element and every facet of all other elements, i.e. number of facets times the squared of the number of elements loops are performed.

In some cases it was observed that the setup and post-processing steps took up to 10% of the total time.
<table>
<thead>
<tr>
<th>Procedure</th>
<th>GT77</th>
<th>GT90</th>
<th>rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>setup</td>
<td>6411.28</td>
<td>574.26</td>
<td>0.09</td>
</tr>
<tr>
<td>connectivity table</td>
<td>911.62</td>
<td>0.39</td>
<td>0.00</td>
</tr>
<tr>
<td>wall distance</td>
<td>4036.58</td>
<td>565.05</td>
<td>0.14</td>
</tr>
<tr>
<td>others</td>
<td>1463.08</td>
<td>8.82</td>
<td>0.01</td>
</tr>
<tr>
<td>* setup - wall distance</td>
<td>2374.70</td>
<td>9.21</td>
<td>0.00</td>
</tr>
<tr>
<td>loop (4 iterations)</td>
<td>54.05</td>
<td>44.24</td>
<td>0.82</td>
</tr>
<tr>
<td>time integration</td>
<td>37.57</td>
<td>35.68</td>
<td>0.95</td>
</tr>
<tr>
<td>turbulence</td>
<td>16.14</td>
<td>8.08</td>
<td>0.50</td>
</tr>
<tr>
<td>boundary conditions</td>
<td>0.16</td>
<td>0.03</td>
<td>0.19</td>
</tr>
<tr>
<td>others</td>
<td>0.18</td>
<td>0.45</td>
<td>2.50</td>
</tr>
<tr>
<td>post-processing</td>
<td>8604.03</td>
<td>5.70</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Figure: Efficiency - test case nozzle, 303750 elements.
Figure: Speedup - test case nozzle, 303750 elements.
Final Remarks

- The new implementation has shown considerable improvement both in terms of CPU time and memory usage.
- The object-oriented programming approach used has enhanced code reuse and the current implementation is ready to be extended.
- The results obtained by the new code are in agreement with those of the original code, both for the serial and parallel versions of the new code.
In terms of parallel performance, the first results achieved satisfactory efficiency although improvement is needed on some procedures.

Almost all object-oriented features could directly be translated into Fortran 90 without great effort.

The lack of run-time polymorphism was overcome by adding polymorphism at an external level.
Future Work

- Improvement of some existing routines
- Addition of new mesh types and methods
- Replacement of the mesh partitioner
The authors give thanks to FAPESP and to the Center for Reference on Gas Turbine (ITA) for their support to this research.