Computers & Fluids 110 (2015) 27–35

Contents lists available at ScienceDirect

Computers & Fluids

journal homepage: www.elsevier.com/locate/compfluid

# Parallel Navier–Stokes simulations for high speed compressible flow past arbitrary geometries using FLASH

Benzi John *, David R. Emerson, Xiao-Jun Gu

*Scientific Computing Department, STFC Daresbury Laboratory, Warrington WA4 4AD, United Kingdom*

ARTICLE INFO

*Article history:*
Received 15 November 2013
Received in revised form 4 December 2014
Accepted 7 December 2014
Available online 16 December 2014

*Keywords:*
FLASH code
Navier–Stokes
Hypersonic flow
Parallel

ABSTRACT

We report extensions to the FLASH code to enable high-speed compressible viscous flow simulation past arbitrary two- and three-dimensional stationary bodies. The body shape is embedded in a block-structured Cartesian adaptive mesh refinement grid by implementing appropriate computer graphics algorithms. A high mesh refinement level is required for an accurate body shape representation which results in large grid sizes especially for three-dimensional simulations. Simulations are done in parallel on IBM Blue Gene/Q computing system on which the code performance has been assessed in both pure MPI and hybrid MPI-OpenMP modes. We also implement appropriate wall boundary conditions in FLASH to model viscous-wall effects. Navier–Stokes (NS) solutions for various two-dimensional test cases like a shock–boundary layer interaction problem as well as for hypersonic flow past blunted cone–cylinder–flare and double-cone geometries are shown. Three dimensional NS simulations of micro vortex generators employed in hypersonic flow control have also been carried out and the computed results have been found to be consistent with experimental results.



## 1. Introduction

The FLASH code [1] developed at the Flash center at the University of Chicago is an open-source code with a wide user base. It was originally designed for application to astrophysics problems like super novae, galaxy clusters and stellar structure. More recently the code has also been used to study High Energy Density Physics (HEDP) problems like laser-driven fusion experiments. The FLASH software system [2,3] follows a modular structure consisting of several inter-operable modules like hydrodynamic, material property and nuclear physics solvers that can be combined to solve various problems in cosmology, high energy density physics, etc. The compressible hydrodynamics code has already been validated for typical benchmark problems like the Sod shock tube test case and the classic wind tunnel step problem [4] by solving the inviscid hydrodynamic (Euler) equations. However, it has not yet been applied to practical high-speed CFD applications mainly due to the fact that it relies on a block-structured adaptive mesh refinement scheme using Cartesian cells to generate the grid. The inherent Cartesian grid structure means that special schemes need to be devised to embed geometries of arbitrary shape in the flow domain. Also, FLASH originally being designed as an astrophysics code is not designed to simulate any viscous wall effects. It has not yet been applied for any compressible Navier–Stokes CFD simulations for flow past arbitrary body shapes, to the best of our knowledge. The FLASH code is modular and extensible which enables users to extend its functionality for their own applications. It has one of the best adaptive mesh compressible hydrodynamics solvers among various open source CFD codes and methodologies to extend its capability for hypersonic flow simulation past arbitrary geometries will be beneficial and of interest to the hypersonic flow community. FLASH is also a scalable parallel code currently featuring both MPI and hybrid MPI-OpenMP modes and stands in good stead in comparison with other codes and parallelization methods [5–8].

The latest version of FLASH (FLASH 4) [1] includes a strategy to incorporate stationary rigid bodies in a computational domain. The solid body is essentially treated as part of the fluid domain and a reflecting boundary condition is applied at the solid/fluid interface. The surface of the rigid body is represented by stair steps due to the regular Cartesian grid structure in FLASH. This scheme, however, can be done only for simple rectangular, spherical or any other shape that can be represented by an analytical expression, which implies that arbitrary complex geometries for real CFD applications cannot be handled currently. In this work, we extend this scheme to generate grids around arbitrary two-dimensional (2D) or three-dimensional (3D) geometric shape by implementing

* Corresponding author.
*E-mail address:* benzi.john@stfc.ac.uk (B. John).

http://dx.doi.org/10.1016/j.compfluid.2014.12.008
0045-7930/

appropriate computer graphics algorithms in FLASH. We also invoke the material property-viscosity module in FLASH to solve the Navier–Stokes equations. Additionally, we implement the no-slip wall boundary conditions and Sutherland's law of viscosity to accurately model viscous-wall effects. A brief discussion of this implementation and preliminary flow results were reported by the authors in [9]. In this work, we elaborate on this and carry out additional Navier–Stokes simulations for several two-dimensional test cases as well as a three-dimensional viscous simulation of micro vortex generators employed in hypersonic flow control.

## 2. Numerical method

The FLASH code has two compressible gas hydrodynamic solvers [1] based on the Finite Volume Method (FVM), built around different operator splitting methods, viz. directionally split and unsplit solvers. They solve the standard compressible Navier–Stokes equations for continuity, momentum and energy, with the pressure field determined from the equation of state [1]. The split solver is based on the piecewise parabolic method (PPM) [10], which is essentially a higher order version of the Godunov scheme. The directionally unsplit solver [11–13] is based on a Monotone Upstream-centered Scheme for Conservation Laws (MUSCL) Hancock type second-order scheme. The unsplit hydro implementation can solve 1D, 2D and 3D problems with added capabilities of exploring various numerical implementations: different types of Riemann solvers; slope limiters; first, second, third and fifth order reconstruction methods as well as a strong shock/rarefaction detection algorithm. One of the notable features of the unsplit hydro scheme is that it particularly improves the preservation of flow symmetries as compared to the splitting formulation. Also, the scheme used in this unsplit algorithm can take a wide range of CFL stability limits for all three dimensions when compared to the directionally split algorithm [11–13].

Grid generation in FLASH is mainly based on a block-structured adaptive mesh refinement (AMR) scheme using PARAMESH [14]. In block-structured AMR, the fundamental data structure is a block of cells arranged in a logically Cartesian fashion, which implies that each cell can be specified using a block identifier (processor number and local block number) and a coordinate triple $(i, j, k)$, where $i$, $j$ and $k$ refer to cell number in the $x$-, $y$-, and $z$-directions, respectively. PARAMESH handles the filling of guard cells that surrounds each block with information from other blocks or, at the boundaries of the physical domain from an external boundary routine. If the neighbor block has a different level of refinement, the data from the neighbor's cells is adjusted by either prolongation (interpolation from a coarse to finer level of resolution) or restriction (averaging from a fine to a coarser level). PARAMESH also enforces flux conservation at jumps in refinement across block boundaries, as described by Berger and Colella [15].

In this work, we implement a computer graphics algorithm in FLASH to enable the generation of random complex three dimensional shapes that represents a rigid body. The algorithm is based on a point-in-polyhedron test using spherical polygons proposed by Carvalho and Cavalcanti [16] coded by John Burkardt. The algorithm determines if a given point is inside or outside a three-dimensional polyhedron based on a method using spherical polygons. The user needs to define all faces of the 3D geometry by specifying the co-ordinate points of each face listed according to the orientation with respect to the outward normal at that face as input to the algorithm. Each face of the polyhedron is then projected onto a unit sphere and the resulting signed area of the spherical polygon thus formed, determines whether the point is inside or outside the polyhedron. A new variable called *bdry_var* is defined and is specified as an adaptive mesh refinement (AMR) variable for all the grid points. The grid points of the computational domain computed to be outside the solid body represents the fluid domain, while those inside represents the solid domain. The values of *bdry_var* for all grid points within the fluid domain are assigned negative values, while those falling within the solid domain are assigned positive values. An illustration of the demarcation between fluid and solid regions in an AMR Cartesian grid is shown in Fig. 1. The fluid/solid interface represents the wall at which appropriate wall boundary conditions should be imposed. An advantage of this scheme is that mesh generation is fairly simple and extremely quick as there is no time consuming phase associated with surface mesh generation and the associated volume mesh. A limitation of this scheme is that, as the grid is non-body-fitted, the body shape is represented by stair steps. A high level of refinement is needed to accurately represent a shape which can result in very large grid sizes for 3D simulations. This can be resolved to a great extent by resorting to high performance parallel computing.

We have also implemented the no-slip boundary condition at the wall (fluid/solid interface) in FLASH to model viscous-wall effects [17]. Boundary conditions in FLASH need to be handled with the aid of guard cells in each coordinate direction which surrounds each block of local data. The data need to be carefully set such that fluxes at the boundary are physically correct. The ghost cell adjacent to the wall is denoted by $g$ whereas the cell inside the computational domain is denoted by 1. The velocity components $(u, v, w)$ and pressure $p$ in the ghost cells are set as:

$$\begin{aligned} u^g &= 2V_w - u^1 \\ v^g &= -v^1 \\ w^g &= -w^1 \\ p^g &= +p^1 \end{aligned} \tag{1}$$

where $V_w$ is the wall velocity. For adiabatic wall boundary conditions, the density gradient in the normal direction vanishes and hence density, $\rho$ can be specified as

$$\rho^g = +\rho^1 \tag{2}$$

For an isothermal wall, density and energy $\varepsilon$ can be specified as

$$\begin{aligned} \rho^g &= 2\rho_w - \rho^1 \\ \varepsilon^g &= 2\varepsilon_w - \varepsilon^1 \end{aligned} \tag{3}$$

The Sutherland's law of viscosity [11] has been implemented in FLASH to model the variation of absolute viscosity with temperature.

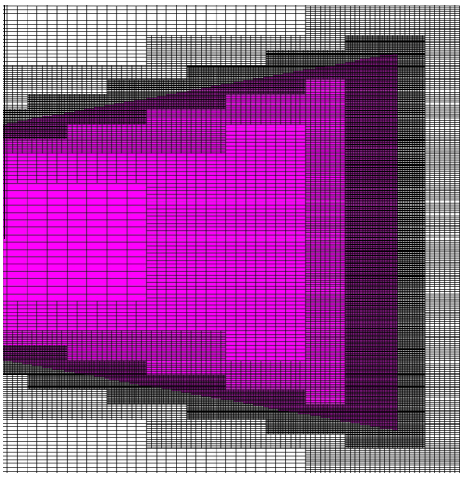

**Fig. 1.** AMR Cartesian grid demarcating the solid domain (denoted in red color) and fluid domain. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)