

Assessment of Edge-Based Viscous Method for Corner-Flow Solutions on Graphics Processing Units

Yi Liu¹, Boris Diskin², and Hiroaki Nishikawa³

National Institute of Aerospace, Hampton, Virginia 23666, USA

William K. Anderson⁴, Gabriel C. Nastac⁵, Eric J. Nielsen⁶, Aaron Walden⁷, and Li Wang⁸

NASA Langley Research Center, Hampton, Virginia 23681, USA

A highly efficient, edge-based viscous (EBV) discretization method has been recently implemented in a practical, unstructured-grid, node-centered, finite-volume flow solver and evaluated for Reynolds-averaged Navier-Stokes formulations. In comparison to a well-established cell-based viscous (CBV) method, the EBV method has demonstrated multifold acceleration of all viscous-kernel computations on general unstructured mixed-element grids. The viscous kernels include evaluation of viscous fluxes, diffusion terms in turbulence models, and the corresponding Jacobian terms. In this paper, an EBV implementation of a nonlinear extension of the Spalart-Allmaras turbulence model, SA-neg-QCR2000, is presented and verified. The SA-neg-QCR2000 model is used for simulating turbulent corner flows. Previously reported EBV computations have been conducted on traditional computing architectures based on central processing units (CPU). This paper assesses benefits of the EBV method on modern high-performance computing architectures based on graphics processing units (GPU). The GPU implementations of the CBV and EBV methods are verified by comparing solutions and iterative convergence with those observed in CPU computations on the same grids. A comprehensive assessment of the EBV speedup on CPU and GPU architectures is presented for established benchmark corner flows, namely, a supersonic flow through a long square duct and a subsonic flow around a NASA juncture flow model.

I. Introduction

This paper assesses a recently developed edge-based viscous (EBV) method on modern computing architectures based on graphics processing units (GPU). The EBV method is used to simulate viscous effects in aerodynamic flows governed by the Reynolds-averaged Navier-Stokes (RANS) equations. The EBV formulation follows the approach introduced in Ref. [1] and employs an efficient edge-based data structure. The edge-based data structure minimizes the operation count and provides efficient memory access, which is beneficial for the GPU implementation. The EBV method has been developed for node-centered, finite-volume discretizations on unstructured tetrahedral [2, 3] and mixed-element [4] grids. The second-order accuracy of EBV solutions has been demonstrated in Ref. [5] using a formal analysis and the method of manufactured solutions. In comparison with a well-established cell-based viscous (CBV) method, the EBV method demonstrated multifold speedup for all viscous-kernel computations [2-5]. The viscous kernels of RANS equations include viscous fluxes, diffusion terms in turbulence and chemistry models, and the corresponding Jacobian terms.

¹Associate Principal Research Engineer, Senior Member AIAA

²Senior Research Fellow, Associate Fellow AIAA

³Research Fellow, Associate Fellow AIAA

⁴Senior Research Scientist, Computational Aerosciences Branch, Associate Fellow AIAA

⁵Research Scientist, Computational Aerosciences Branch, Member AIAA

⁶Research Scientist, Computational Aerosciences Branch, Associate Fellow AIAA

⁷Research Scientist, Computational Aerosciences Branch

⁸Research Aerospace Engineer, Computational Aerosciences Branch, Associate Fellow AIAA

The EBV method has been implemented in a large-scale flow solver, FUN3D [6], which is developed and maintained at the NASA Langley Research Center (LaRC). FUN3D is widely used for high-fidelity analysis of complex turbulent flows across the speed range from incompressible to hypersonic regimes [7-11]. The core functions of FUN3D have recently been adapted for GPU architectures [12, 13] and demonstrated breakthrough GPU performance for large-scale aerodynamic applications [14, 15]. Further improvements in GPU performance of FUN3D solutions have been reported in Ref. [16]. Recently, wall-modeled large-eddy simulation (WMLES) capabilities have been implemented and assessed for several benchmark flows [17].

The baseline finite-volume discretization scheme of FUN3D balances fluxes at median-dual control volumes that are centered at grid points. Inviscid fluxes are evaluated at edge midpoints in an efficient edge-based loop. Viscous fluxes use the Green-Gauss theorem to compute gradients at grid cells. For nontetrahedral cells, the cell gradients at each edge within the cell are augmented with an edge derivative. Because this approach relies on cell-based gradients, the viscous fluxes are computed in a separate cell-based loop. In this paper, the baseline viscous-flux implementation is referred to as the CBV method. The CBV method provides a compact nearest-neighbor stencil and is equivalent to the P1 continuous Galerkin finite-element method on tetrahedra [1]. The CBV solutions have been extensively verified and validated through formal analysis and applications [18-24].

The EBV method essentially mimics the CBV method on tetrahedra but uses an efficient edge-based data structure. The EBV method has been initially implemented on tetrahedra [2, 3] and demonstrated significant reduction of time to solution on tetrahedral grids. The EBV coefficients are stored at grid edges and represent grid metrics that are independent of the solution and fixed for stationary and rigidly moving grids. The viscous fluxes are computed in a loop over edges. The EBV source terms that are required for second-order accuracy are computed in a loop over grid points [3, 5]. The EBV implementation on mixed-element grids reported in Ref. [4] uses an extended set of EBV edges. The EBV method requires any two vertices of a cell to be connected by an edge. The primal edges of a nontetrahedral cell do not connect some vertices of the cell. To satisfy the vertex connectivity requirement, nontetrahedral cells of a general mixed-element grid are consistently divided into tetrahedra using an algorithm based on global numbering of grid points [25]. The EBV method is implemented on the derived tetrahedral grid. The EBV edges are all edges of the derived tetrahedral grid including all primal edges of the mixed-element grid. The EBV coefficients are stored for all EBV edges. The viscous-kernel computations are conducted in a loop over EBV edges. Recently, the EBV method has been ported to GPUs using a thin abstraction layer over CUDA C++ [26].

This paper also reports on the EBV implementation of a nonlinear turbulence model that is well suited for simulation of turbulent corner flows. A linear eddy viscosity model that makes use of the Boussinesq assumption cannot express anisotropy of normal stresses and fails to represent a secondary flow near a corner. In Ref. [27], Spalart developed a quadratic constitutive relation (QCR) that accurately models the secondary flow in a corner. The negative variant [28] of the standard Spalart-Allmaras (SA) model [29] that uses the QCR extension is referred to as the SA-neg-QCR2000 turbulence model. Recently, the CBV implementation of the SA-neg-QCR2000 model has been extensively verified for several relevant benchmark test cases [18]. In this paper, the EBV implementation of the SA-neg-QCR2000 model on CPU and GPU architectures is verified and assessed using two of these cases, namely, a supersonic flow through a long square duct and a subsonic flow around a NASA juncture-flow model (JFM).

The material in this paper is organized as follows. Section II outlines the baseline discretization scheme and iterative solver of FUN3D and the GPU implementation. Section III describes the SA-neg-QCR2000 model. Section IV details the EBV implementation in FUN3D, including the SA-neg-QCR2000 model. Section V compares the CBV and EBV RANS solutions on CPU and GPU architectures and analyzes the EBV efficiency benefits. Section VI provides concluding remarks.

II. Baseline Discretization and GPU Implementation

This section provides an overview the baseline discretization scheme and iterative solver of FUN3D. The basic approach to porting FUN3D kernels on GPU is also outlined.

A. Baseline Discretization Scheme

FUN3D solves the RANS equations discretized on unstructured mixed-element grids that may contain tetrahedra, pyramids, prisms, and hexahedra. The residuals are evaluated on a set of median-dual control volumes centered at grid points. Edge-based inviscid fluxes are computed at primal edge medians using an approximate Riemann solver. In the current study, Roe's flux-difference splitting [30] is used. For second-order accuracy, density, pressure, and velocity are reconstructed by a UMUSCL (Unstructured Monotonic Upstream-centered Scheme for Conservation Laws) scheme [32]. The spatial discretization of the SA-neg turbulence model uses a first-order accurate convection scheme. For the discretization of viscous fluxes, the Green-Gauss theorem is used to compute cell-based gradients resulting in

the CBV formulation. On tetrahedral grids, this CBV approach is equivalent to a Galerkin approximation [1]. For nontetrahedral grids, cell-based Green-Gauss gradients are combined with edge-based gradients [33, 34] to improve the stability of viscous operators and prevent odd-even decoupling. In this paper, the edge-normal augmentation [34] is used. The diffusion term in the turbulence model is handled similarly. In the CBV implementation of the QCR correction, the antisymmetric normalized rotation tensor is computed using the cell-based gradients. The QCR fluxes are added in a cell loop. The vorticity-based source term for the turbulence model is computed using velocity gradients evaluated by the Green-Gauss method on dual control volumes. The boundary conditions involved in the present study include subsonic pressure-based outflow, farfield Roe-based, symmetry, and viscous-wall boundary conditions [35].

B. Iterative Solvers

FUN3D has two iterative solvers, a baseline solver and a strong nonlinear solver. Currently, only the baseline iterative solver has been fully ported to GPU and verified. The baseline nonlinear iterations use a defect-correction method. Nonlinear residuals represent the target operator, and a linear system that is based on an approximate Jacobian represents the driver operator. The iterations for the meanflow and turbulence-model equations are loosely coupled. The approximate Jacobian for the meanflow equations is formed using the linearization of the first-order flux-vector splitting inviscid fluxes [36] and the second-order viscous fluxes. The Jacobian for the SA-neg equation includes contributions from the advection, diffusion, and source terms. The exact linearization is used for the advection and diffusion terms. The entire contribution from the linearized source term is added to the diagonal. The pseudotime term is controlled by a Courant-Friedrichs-Lewy (CFL) number that can be predefined or ramped linearly within a specified number of nonlinear iterations. At each nonlinear iteration, the linear system is solved using a specified number of point-implicit Gauss-Seidel (GS) sweeps with multicolor ordering. The solution of the linear system is added to the nonlinear solution. Nonlinear iterations can reuse the Jacobian computed at a previous iteration; Jacobian updates are scheduled according to the convergence of the nonlinear residual. Every tenth nonlinear iteration always performs the Jacobian update; other nonlinear iterations may skip the update if the residual reduction in the previous nonlinear iteration exceeded a specified target. This option is referred to as a “smart” Jacobian update. The smart Jacobian update is performed separately for the meanflow and SA-neg Jacobians.

Hierarchical adaptive nonlinear iteration method (HANIM) is a strong nonlinear solver that enhances the baseline iterative solver with a mechanism to increase convergence rate and overcome instabilities occurring in transient solutions [37-41]. HANIM has been ported to GPUs for the laminar Navier-Stokes equations. Porting of HANIM kernels to GPU for RANS models has not been completed yet, thus the results reported in this paper do not include HANIM computations. Previous CPU-based studies [3, 4] have demonstrated that the EBV speedup of HANIM solutions is comparable with the EBV speedup of the baseline iterative solver.

C. GPU Implementation

FUN3D originated approximately 30 years ago as a CPU-based code written in Fortran. Coarse-grained message passing interface (MPI)[†] with conventional domain decomposition techniques are used for parallelism. Recently, FUN3D has been adapted for NVIDIA GPU architectures [12, 13] and has been used to demonstrate breakthrough performance for perfect gas and thermochemical nonequilibrium flows [14, 15]. FUN3D calls the FUN3D library for universal device acceleration (FLUDA) to compute on GPU-based architectures. FLUDA was written in CUDA C++ [42] by porting the original FUN3D Fortran compute kernels and has undergone extensive optimization [12, 15, 16, 43, 44]. Kernel performance optimizations include, but are not limited to, memory coalescing, reduction of state, reduction of thread divergence, and utilization of hierarchical parallelism. Support for AMD and Intel GPUs and multicore CPUs has been added to FLUDA using a thin abstraction layer above CUDA C++ [26]. Both CBV and EBV methods are available for perfect- and generic-gas flow simulations in FLUDA.

III. SA-neg-QCR2000 Turbulence Model

Following the formulation presented at the NASA Turbulence Modeling Resource (TMR) website*, the standard SA model [29] is given by the following nonconservative equation:

$$\partial_t \hat{v} + \mathbf{u} \cdot \nabla \hat{v} - c_{b1}(1 - f_{t2})\hat{S}\hat{v} + \left[c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2} \right] \left(\frac{\hat{v}}{d} \right)^2 - \frac{1}{\sigma} [\nabla \cdot ((\nu + \hat{v})\nabla \hat{v}) + c_{b2}(\nabla \hat{v} \cdot \nabla \hat{v})] = 0. \quad (1)$$

[†] <https://www.mpi-forum.org/>; accessed October, 2022

* <https://turbmodels.larc.nasa.gov/>; accessed October, 2022

Here, $\nabla \equiv (\partial_x, \partial_y, \partial_z)^T$ denotes a formal vector of spatial derivatives, $\hat{v}$ is the turbulence variable, $\mathbf{u} = (u, v, w)$ is the velocity vector, d is the distance to the nearest wall, $\nu = \mu/\rho$ is the kinematic viscosity, ρ is the density, and μ is the dynamic laminar viscosity computed by Sutherland's law

$$\frac{\mu}{\mu_0} = \left(\frac{T}{T_0}\right)^{3/2} \left(\frac{T_0 + S}{T + S}\right). \quad (2)$$

T is the temperature, $\mu_0 = 1.716 \times 10^{-5} \text{ kg/(ms)}$, $T_0 = 491.6 \text{ }^\circ\text{R}$, and $S = 198.6 \text{ }^\circ\text{R}$, $\hat{S}$ in Eq. 1 is defined as

$$\hat{S} = \Omega + \frac{\hat{v}}{\kappa^2 d^2} f_{v2}, \quad (3)$$

where Ω is the magnitude of vorticity,

$$\Omega = \sqrt{(\partial_y w - \partial_z v)^2 + (\partial_z u - \partial_x w)^2 + (\partial_x v - \partial_y u)^2}, \quad (4)$$

$$f_{v1} = \frac{\chi^3}{c_{v1}^3 - \chi^3}, \quad \chi = \frac{\hat{v}}{\nu}, \quad f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}}, \quad f_w = g \left[\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6} \right]^{\frac{1}{6}}, \quad g = r + c_{w2}(r^6 - r), \quad (5)$$

$$f_{t2} = c_{t3} \exp(-c_{t4} \chi^2), \quad r = \min \left[\frac{\hat{v}}{\hat{S} \kappa^2 d^2}, 10 \right], \quad (6)$$

and the constants are $\kappa = 0.41$, $\sigma = \frac{2}{3}$, $c_{b1} = 0.1355$, $c_{b2} = 0.622$, $c_{w1} = \frac{c_{b1}}{\kappa} + \frac{1+c_{b2}}{\sigma}$, $c_{w2} = 0.3$, $c_{w3} = 2$, $c_{v1} = 7.1$, $c_{t3} = 1.2$, and $c_{t4} = 0.5$.

The standard SA model equation is solved for $\hat{v} > 0$. For negative $\hat{v}$, the SA-neg equation [28] is solved

$$\partial_t \cdot \nabla \hat{v} - c_{b1}(1 - c_{t3})\Omega \hat{v} - c_{w1} \left(\frac{\hat{v}}{d} \right)^2 - \frac{1}{\sigma} [\nabla \cdot ((v + \hat{v} f_n) \nabla \hat{v}) + c_{b2}(\nabla \hat{v} \cdot \nabla \hat{v})] = 0, \quad (7)$$

with

$$f_n = \frac{c_{n1} + \chi^3}{c_{n1} - \chi^3}, \quad c_{n1} = 16. \quad (8)$$

The turbulent eddy viscosity is computed as

$$\mu_t = \max(\rho \hat{v} f_{v1}, 0). \quad (9)$$

The tensor of turbulent stresses is defined as follows.

$$\mathbf{T} = \begin{bmatrix} \tau_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \tau_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \tau_{zz} \end{bmatrix}, \quad (10)$$

$$\tau_{xx} = \frac{2}{3} \mu_t (2\partial_x u - \partial_y v - \partial_z w), \quad (11)$$

$$\tau_{yy} = \frac{2}{3} \mu_t (2\partial_y v - \partial_x u - \partial_z w), \quad (12)$$

$$\tau_{zz} = \frac{2}{3} \mu_t (2\partial_z w - \partial_x u - \partial_y v), \quad (13)$$

$$\tau_{yx} = \tau_{xy} = \mu_t (\partial_x v + \partial_y u), \quad (14)$$

$$\tau_{zx} = \tau_{xz} = \mu_t (\partial_x w + \partial_z u), \quad (15)$$

$$\tau_{zy} = \tau_{yz} = \mu_t (\partial_z v + \partial_y w). \quad (16)$$

In the QCR formulation [27], the turbulent stress tensor is corrected using the normalized antisymmetric rotation tensor that is defined as

$$O_{xx} = O_{yy} = O_{zz} = 0, \quad O_{xy} = -O_{yx} = \frac{\partial_y u - \partial_x v}{b}, \quad O_{xz} = -O_{zx} = \frac{\partial_z u - \partial_x w}{b}, \quad O_{yz} = -O_{zy} = \frac{\partial_z v - \partial_y w}{b}. \quad (17)$$

Here, b is the l_2 -norm of velocity gradient tensor defined as

$$b = \sqrt{(\partial_x u)^2 + (\partial_y u)^2 + (\partial_z u)^2 + (\partial_x v)^2 + (\partial_y v)^2 + (\partial_z v)^2 + (\partial_x w)^2 + (\partial_y w)^2 + (\partial_z w)^2}. \quad (18)$$

The rotation tensor is set to be zero when $b = 0$. The QCR turbulent stress tensor is computed as follows

$$\tau_{xx, QCR} = \tau_{xx} - 2c_{cr1}(O_{xy}\tau_{xy} + O_{xz}\tau_{xz}), \quad (19)$$

$$\tau_{yy,QCR} = \tau_{yy} - 2c_{cr1}(O_{yx}\tau_{yx} + O_{yz}\tau_{yz}), \quad (20)$$

$$\tau_{zz,QCR} = \tau_{zz} - 2c_{cr1}(O_{zx}\tau_{zx} + O_{zy}\tau_{zy}), \quad (21)$$

$$\tau_{xy,QCR} = \tau_{xy} - c_{cr1}(O_{xy}(\tau_{yy} - \tau_{xx}) + O_{yz}\tau_{xz} + O_{zx}\tau_{yz}), \quad (22)$$

$$\tau_{xz,QCR} = \tau_{xz} - c_{cr1}(O_{xz}(\tau_{zz} - \tau_{xx}) + O_{xy}\tau_{zy} + O_{zy}\tau_{xy}), \quad (23)$$

$$\tau_{yz,QCR} = \tau_{yz} - c_{cr1}(O_{yz}(\tau_{zz} - \tau_{yy}) + O_{yx}\tau_{zx} + O_{zx}\tau_{yx}), \quad (24)$$

$$c_{cr1} = 0.3.$$

IV. EBV Method

This section provides a brief description of the EBV method and the EBV implementation of the QCR correction.

A. Precomputing EBV coefficients

The EBV preprocessing step is conducted following the standard startup procedure of loading and partitioning a mixed-element grid and creating all necessary data structures for grid points, edges, faces, and cells. Within each local partition, nontetrahedral cells are divided into tetrahedra. The division based on the global grid-point numbering [25] is unique and ensures consistency of the derived tetrahedral grid across the partitions. Note that partitioning of cells with highly non-planar quadrilateral faces may result in tetrahedra with negative volumes. The EBV edges include all edges of the derived tetrahedral grid. An EBV edge pointer array with dimensions (2, number of EBV edges) is created. The set of primal edges of the mixed-element grid is used to compute inviscid fluxes, and the set of EBV edges is used to compute viscous fluxes.

An auxiliary edge-based array is created for pointers to the compressed-row data structure [45] for the Jacobian off-diagonal terms. This auxiliary array has dimensions of (2, number of EBV edges) and facilitates access to the off-diagonal terms of the Jacobian at each edge. Another auxiliary array is created to mark all boundary grid points. This marking distinguishes the boundary EBV edges from the interior EBV edges. The EBV coefficients are precomputed in a loop over cells of the derived tetrahedral grid. See Refs. [2, 3] for more details. The EBV data structures include two two-dimensional 32-bit integer arrays for local cell-to-node and cell-to-edge data structures of the derived tetrahedral grid; two 32-bit integer edge-pointer (2, number of EBV edges) arrays to point to the edge endpoints and to the Jacobian off-diagonal terms; a one-dimensional 32-bit integer array for boundary-point indicators; two two-dimensional 64-bit floating-point edge-based arrays, one array to store the six EBV coefficients per edge for a symmetric EBV matrix at interior EBV edges and another array to store additional three EBV coefficients per edge for a non-symmetric EBV matrix at boundary EBV edges. For static and rigidly moving grids, the EBV cell-to-node and cell-to-edge data structures are needed only for the preprocessing stage and can be deallocated after the EBV coefficients have been precomputed. All other EBV data structures are used in the solver execution and carried through the entire simulation.

B. EBV method implementation and execution

First, the temperature and laminar viscosity are precomputed in a loop over grid points. This preliminary loop is needed to avoid repetitive computations. For formal second-order accuracy, the EBV method for the meanflow equations requires correction terms [5] that depend on the gradient of viscosity and the gradients of the velocity components. The specific formulation of the correction terms for each conservation equation is the following.

$$x - \text{momentum} : \frac{5}{6} [\partial_x(\mu + \mu_t)\partial_y v - \partial_y(\mu + \mu_t)\partial_x v + \partial_x(\mu + \mu_t)\partial_z w - \partial_z(\mu + \mu_t)\partial_x w], \quad (25)$$

$$y - \text{momentum} : \frac{5}{6} [\partial_y(\mu + \mu_t)\partial_x u - \partial_x(\mu + \mu_t)\partial_y u + \partial_y(\mu + \mu_t)\partial_z w - \partial_z(\mu + \mu_t)\partial_y w], \quad (26)$$

$$z - \text{momentum} : \frac{5}{6} [\partial_z(\mu + \mu_t)\partial_x u - \partial_x(\mu + \mu_t)\partial_z u + \partial_z(\mu + \mu_t)\partial_y v - \partial_y(\mu + \mu_t)\partial_z v], \quad (27)$$

$$\text{energy} : \frac{5}{6} \{ 2(\mu + \mu_t) [(\partial_x u \partial_y v + \partial_x u \partial_z w + \partial_y v \partial_z w) - (\partial_y u \partial_x v + \partial_z u \partial_x w + \partial_z v \partial_y w)] \quad (28)$$

$$+ \partial_x(\mu + \mu_t) [u(\partial_y v + \partial_z w) - v\partial_y u - w\partial_z u] \\ + \partial_y(\mu + \mu_t) [v(\partial_x u + \partial_z w) - u\partial_x v - w\partial_z v] \\ + \partial_z(\mu + \mu_t) [w(\partial_z w + \partial_y v) - u\partial_x w - v\partial_y v] \}.$$

The correction terms use velocity and viscosity gradients evaluated at grid points. The gradients of velocity components computed by an unweighted least-squares (ULSQ) method are available at grid points. These velocity gradients are used for inviscid-flux UMUSCL reconstruction. ULSQ gradients of the total viscosity, which combines the laminar and eddy viscosities, are computed and stored at grid points. A viable alternative to the ULSQ method is the Green-Gauss method on dual volumes that is used by FUN3D to compute vorticity for the turbulence model source term. The EBV correction terms are computed as source terms for conservation equations in a separate loop over grid points. In the current EBV implementation, the meanflow Jacobian does not include linearization of the EBV correction source terms.

The EBV method for meanflow viscous fluxes is implemented in an edge loop in which each EBV edge provides contributions to the meanflow residuals at the edge endpoints. The residual contributions are products of the (six or nine) EBV coefficients at the edge, solution edge differences, and edge averages of viscosity and velocity components [2-5]. The point values of temperature are used to compute the temperature edge difference that is needed for the evaluation of the heat flux. The point values of viscosity and velocity are used to compute the corresponding edge averages.

The residual contributions from the QCR correction fluxes (the terms in Eqs. 19-24 that are multiplied by c_{cr1}) are computed in a loop over primal edges of the mixed-element grid. First, a point-based loop is performed to compute the shear-stress tensor (Eqs. 10-16) and the rotation tensor (Eq. 17) and to evaluate and store the correction terms of the QCR stress tensor at grid points. These terms are computed within the same point-based loop where EBV correction terms are calculated. Since the 3×3 matrix of correction terms is antisymmetric, six 64-bit floating-point numbers are stored at each grid point. The correction terms use velocity gradients that are readily available at grid points. The QCR correction stress tensor at a primal edge is averaged from the QCR correction stress tensors stored at the edge endpoints. The edge-based viscosity and velocity are averaged from the edge endpoints as well. The resulting QCR correction fluxes are multiplied by the directed area associated with the primal edge and added with an appropriate sign to the residuals at the edge endpoints. Currently, the meanflow Jacobian does not include linearization of the QCR correction fluxes.

The EBV method for the SA-neg diffusion term and its Jacobian is implemented in a separate edge loop. Only one EBV coefficient is needed for evaluation of the SA-neg diffusion term. This EBV coefficient is the trace (sum of diagonal elements) of the EBV matrix. No additional storage is needed for the EBV implementation of the SA-neg diffusion term.

The array to store the off-diagonal terms of the Jacobian matrix is implemented to include only terms that correspond to grid points connected by the EBV edges. For tetrahedral grids, the off-diagonal Jacobian array has the same size for the CBV and EBV methods. For mixed-element grids, the EBV discretization stencil is smaller than the CBV discretization stencil, resulting in significant reduction in the size of the off-diagonal Jacobian array. A smaller Jacobian implies a smaller memory footprint and faster matrix-vector multiplication within the linear solver.

Some boundary conditions implemented in the baseline solver use cell-based gradient evaluation. In the presence of such boundary conditions, the rows of the EBV off-diagonal Jacobian matrix corresponding to boundary grid points are extended to include off-diagonal terms for all vertices of the cells that share the boundary grid point and have a face at a boundary surface.

V. EBV Solutions for Benchmark Flows

This section compares the CBV and EBV SA-neg-QCR2000 solutions for benchmark flows described in Ref. [18]. Fully converged RANS solutions are computed on the CPU and GPU architectures. Comprehensive analysis of the accuracy and efficiency comparisons between CBV and EBV methods are reported.

A. Supersonic Flow through Square Duct

The test case studies a supersonic flow through a long duct with a square cross-section of height and width $D = 25.4$ mm. This benchmark flow is described at the NASA Turbulence Modeling Resource (TMR) website[§]. The case is designated to verify RANS capabilities for simulating corner flows that exhibit secondary recirculation. The important characteristics to be observed are the resolution of secondary vortices and accurate assessment of the centerline velocity. The configuration geometry and boundary conditions are illustrated in Fig. 1. The experimental measurements were conducted at the stations located $40D$ and $50D$ downstream of the duct inlet. The origin of the coordinate system is set at the lower-left corner of the front section of the duct. The positive direction of the x-axis is the streamwise direction along the duct. The duct dimensions are $0 \leq x \leq 52D$; $0 \leq y \leq 0.5D$; $0 \leq z \leq 0.5D$.

[§] https://turbmodels.larc.nasa.gov/3dsqduct_mod_numerics_val.html. Accessed October, 2022

Adiabatic-wall conditions are set at the duct boundaries corresponding to $0 \leq x \leq 52D; y = 0; 0 \leq z \leq 0.5D$ and $0 \leq x \leq 52D; 0 \leq y \leq 0.5D; z = 0$. The computational domain is extended upstream with a small segment of symmetry boundaries $-1.26829D \leq x \leq 0$. The flow corresponds to the reference freestream Mach number $M_\infty = 3.9$, the Reynolds number $Re_D = 5.08 \times 10^5$, the angle of attack $\alpha = 0^\circ$, and the reference freestream temperature $T_\infty = 520^\circ\text{R}$. Supersonic inflow conditions are set at $x = -1.26829D$ corresponding to the reference conditions. Supersonic outflow conditions are set at $x = 52D$. Symmetry conditions are specified at the boundaries corresponding to $y = 0.5D$, and $z = 0.5D$, thus, only one quarter of the duct is computed. The reference area is unity.

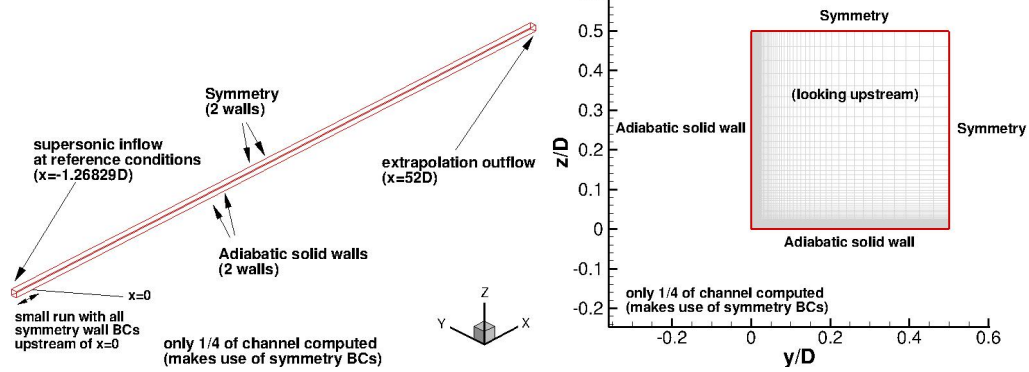


Figure 1. Supersonic flow through square duct: geometry and boundary conditions.

A family of hexahedral grids has been generated using FORTRAN programs available at the TMR website. Due to the use of symmetry boundary conditions, only one quarter of the duct is gridded. There are four grids available at the TMR website and in Ref. [18], named as G1 to G4 from the finest of 1.59 billion nodes to the coarsest of 3.17 million nodes. The reference solutions for the SA-neg-QCR2000 model presented in Ref. [18] and the TMR website are computed using established NASA RANS solvers, FUN3D and USM3D. Reference FUN3D solutions use the CBV method. Figures 2 and 3 compare EBV solutions on grids G2, G3 and G4 with the reference solutions. Figure 2 shows grid convergence of the drag coefficient and the maximum eddy viscosity versus the nominal mesh size computed as $h = N^{-1/3}$, where N is the degrees of freedom, i.e., grid nodes for FUN3D. The drag coefficient and the maximum eddy viscosity coefficient computed by the EBV method are in good agreement with the TMR results, approaching the coefficients computed by the CBV method in grid refinement. The variation between the coefficients computed on the finer grids is a small fraction of one percent. Figure 3 shows grid convergence of centerline velocity normalized by the freestream speed of sound at two locations, $x/D = 40$ and $x/D = 50$. The centerline velocity from the EBV solution is between the corresponding velocity values extracted from the reference solutions on grids G2 and G3 and approaches the velocity from the CBV solution on finer grids.

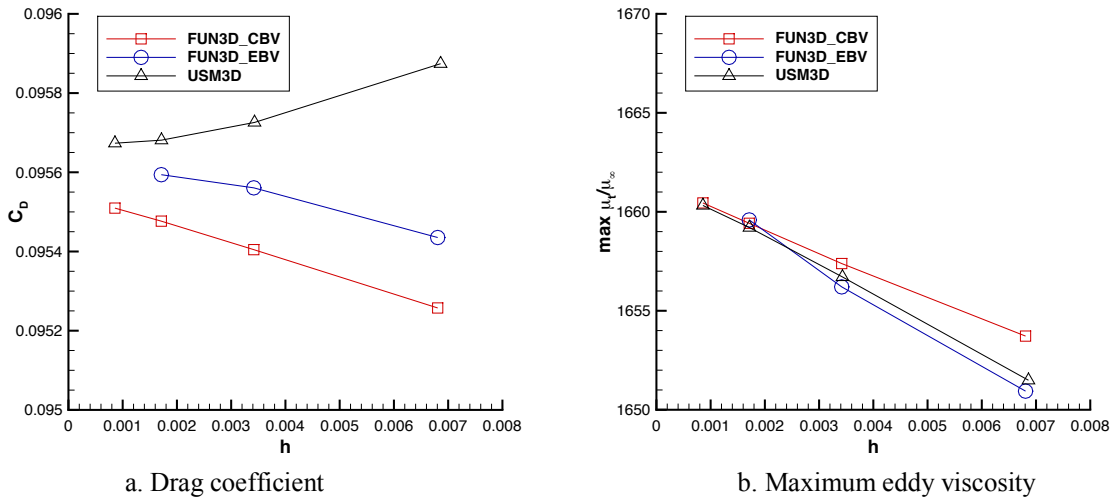


Figure 2. Supersonic flow through square duct: grid convergence of drag and maximum eddy viscosity.

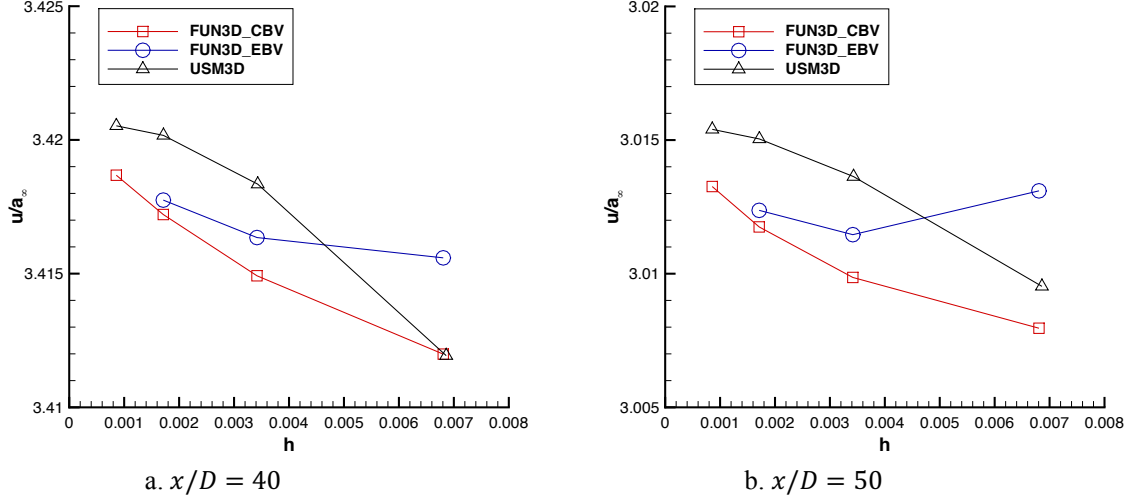


Figure 3. Supersonic flow through square duct: grid convergence of centerline velocity.

Figures 4 and 5 show the contours of the crossflow velocity for solutions computed on grid G4 with SA-neg-QCR2000 and SA-neg models, respectively. The crossflow velocity is normalized by the reference speed of sound ($\sqrt{v^2 + w^2}/a_\infty$) and computed at $x/D = 50$. The CBV and EBV solutions computed with the same turbulence model are nearly identical in this view. The SA-neg-QCR2000 solutions shown in Figure 4 clearly exhibit the secondary circulation that cannot be captured with linear turbulence models as shown in Figure 5.

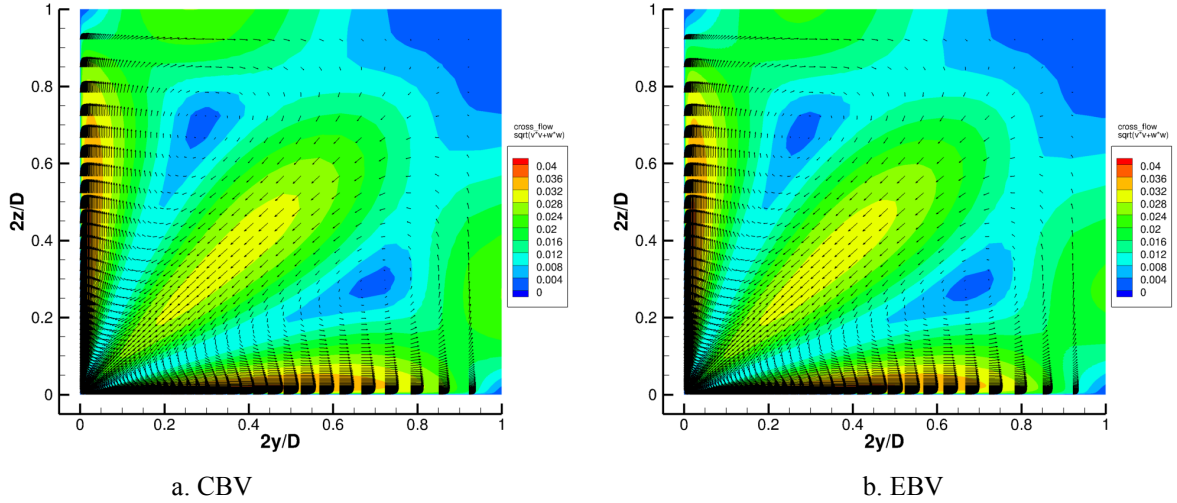


Figure 4. Crossflow velocity at $x/D = 50$ computed with SA-neg-QCR2000 turbulence model.

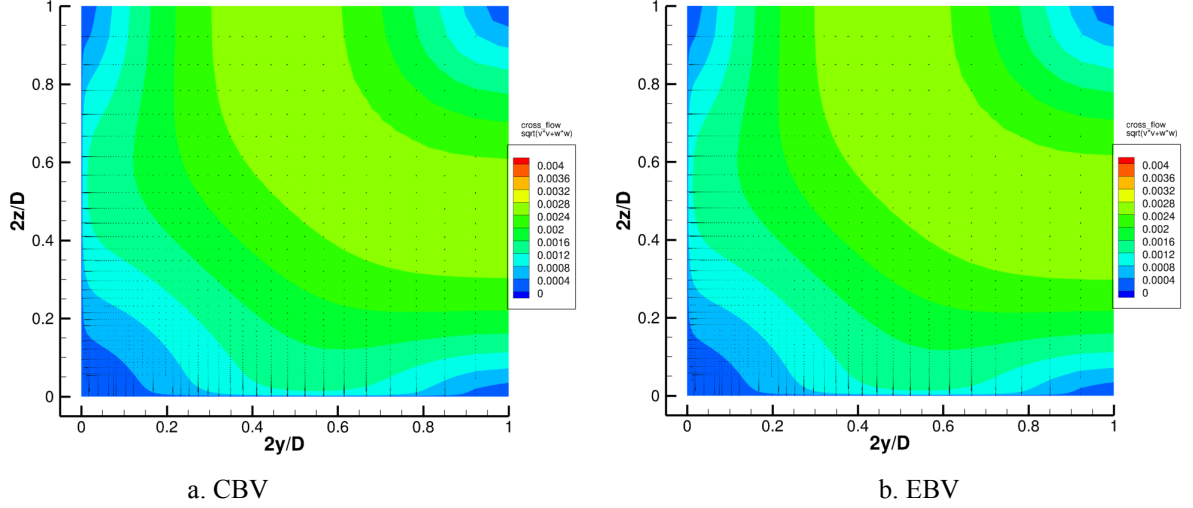


Figure 5. Crossflow velocity at $x/D = 50$ computed with SA-neg turbulence model.

The simulations on grid G4 are profiled on CPU and GPU architectures. Grid G4 has approximately 3 million grid points and fits well in one GPU card. The CBV and EBV solutions are computed on the NASA LaRC K4 cluster. CPU solutions are obtained using one and five Skylake Xeon Gold 6148 nodes, each Skylake node has 40 CPU cores. GPU solutions are obtained using one NVIDIA V100 card.

Figure 6 shows the residual convergence history of the CBV and EBV solutions. In these CPU computations, one Skylake node is used to provide node-to-node comparison. Figures 6a and 6b show CPU and GPU residual convergence versus iterations. Figures 6c and 6d show the same convergence history versus wall time. The subscripts $_R2$ and $_R6$ denote the root-mean-square (rms) norms of the x -momentum and SA-neg-QCR2000 residuals, respectively. The CBV and EBV convergence per iteration is hardly distinguishable on CPU and GPU. The EBV method takes less time per iteration than the CBV method and significantly reduces the time to convergence. Comparing Figures 6c and 6d, the GPU simulations are five times faster than the CPU simulations. The drag coefficients computed from the CBV and EBV solutions on grid G4 differ by less than 0.2%. The drag coefficients computed by the same viscous method on CPU and GPU match for at least 10 digits.

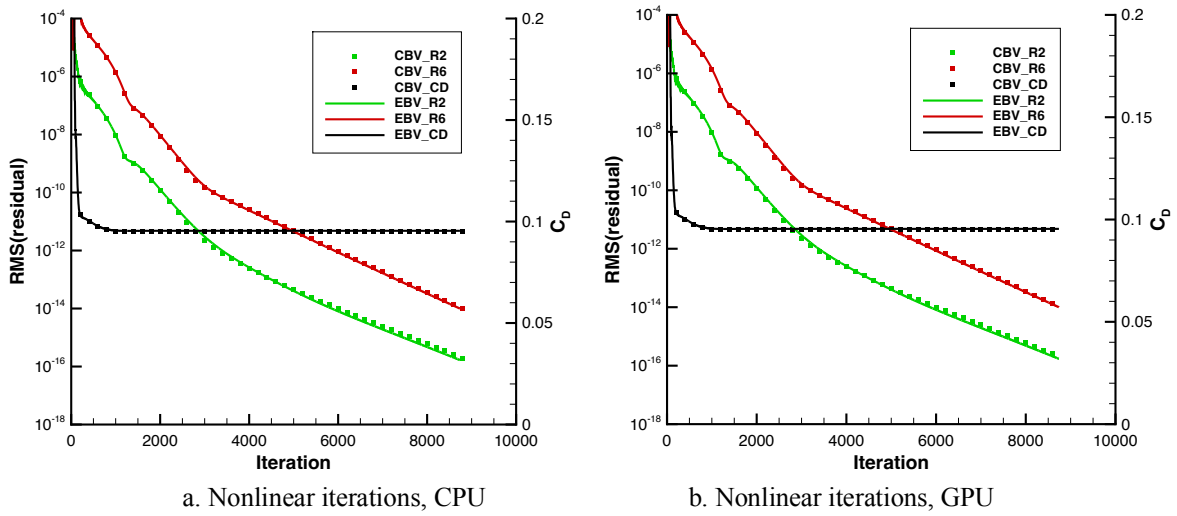


Figure 6. Supersonic flow through square duct: residual convergence on grid G4.

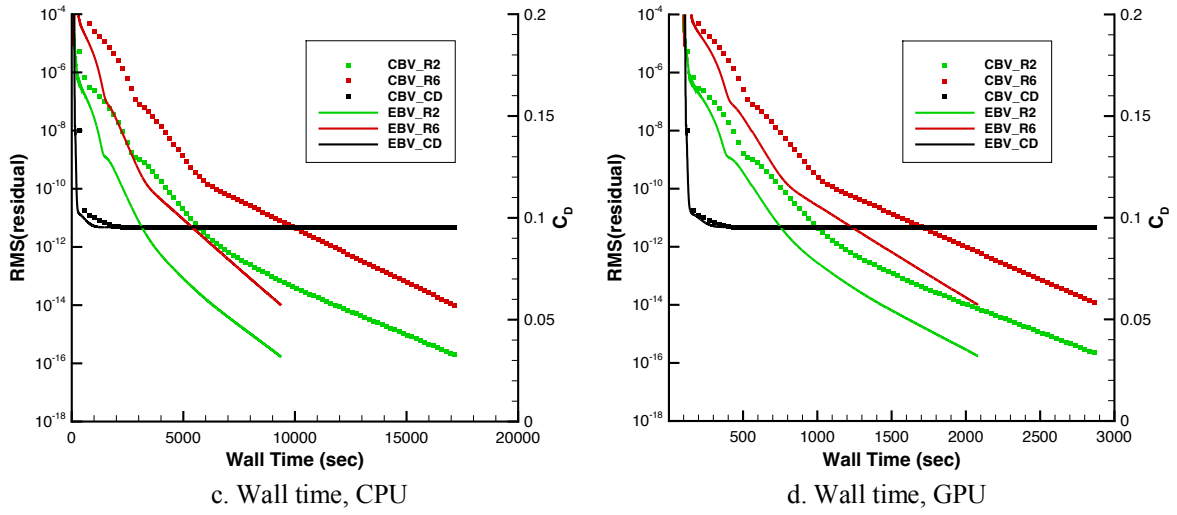


Figure 6. Concluded.

Table 1 shows the wall time of the CBV and EBV solutions computed using CPU and GPU and the speedup obtained due to the EBV method and the GPU acceleration. The speedup is computed as the ratio of the baseline time to the reduced time. The GPU speedup is $6.0\times$ and $4.7\times$ for the CBV and EBV methods, respectively. The EBV speedup is $1.8\times$ on CPU, and $1.4\times$ on GPU.

Table 1. Time-to-solution statistics.

	CPU (40 cores)		CPU (200 cores)		GPU		GPU speedup
	Iterations	Wall time (sec.)	Iterations	Wall time (sec.)	Iterations	Wall time (sec.)	
CBV	8,805	17,148	8,805	3,561	8,799	2,817	6.09
EBV	8,737	9,375	8,737	1,887	8,739	1,996	4.69
EBV speedup		1.83		1.89		1.41	

The GPU implementation of the CBV SA-neg-QCR2000 model is fully optimized for NVIDIA GPUs [46], while the corresponding CPU implementation is not optimized. The EBV implementation is optimized for both CPU and GPU computations. This is the reason for different EBV speedup factors observed on CPU and GPU architectures and for different GPU speedup factors observed for the CBV and EBV methods. Considering the GPU speedup as representative, the performance of one NVIDIA V100 card is roughly similar to the performance 188 Skylake CPU cores. The observed GPU speedup of $4.7\times$ provided for EBV solutions matches well the theoretical speedup reported in Ref. [26]. Table 1 includes the timing data for solutions computed using five Skylake nodes with 200 CPU cores. The time to convergence observed in the latter computations is comparable with the time of the solutions computed using one GPU card.

To illustrate the effect of optimization of the CBV implementation on code performance, Tables 2 and 3 show timing of viscous-kernel and linear-solver CPU computations conducted with the CBV and EBV methods and the corresponding EBV speedup. The timing for such computations has been averaged over all iterations for residual evaluations and linear solvers and over the number of Jacobian updates for Jacobian evaluations. Table 2 shows computations for an optimized SA-neg model and Table 3 shows similar computations for a non-optimized SA-neg-QCR2000 model. The first observation is that the EBV speedup factors for viscous fluxes and Jacobian computations dramatically increase for the SA-neg-QCR2000 model. At the same time, the EBV speedup for all other computations somewhat decreases. The second observation is that the EBV timing remains approximately the same for all computations with the exception of the viscous flux. A modest increase in the viscous flux timing is expected because viscous fluxes in the SA-neg-QCR2000 model include the QCR correction terms. In contrast to the EBV method, the CBV timing of meanflow viscous fluxes and Jacobian computations dramatically increases for the SA-neg-QCR2000 model; the viscous flux time is more than tripled, the Jacobian time is more than doubled. These increases represent the evidence of a suboptimal CPU implementation of meanflow viscous fluxes in the SA-neg-QCR2000 model.

Table 2. SA-neg model: time of viscous-kernel and linear-solver computations in one iteration.

Viscous method	Meanflow time per iteration (sec.)			SA-neg time per iteration (sec.)		
	Viscous flux	Jacobian	Linear solver	Diffusion	Jacobian	Linear solver
CBV	0.0978	0.6519	0.9556	0.0886	0.2817	0.0523
EBV	0.0310	0.1987	0.5945	0.0094	0.0192	0.0385
EBV Speed up	3.15	3.28	1.61	9.42	14.67	1.36

Table 3. SA-neg-QCR2000 model: time of viscous-kernel and linear-solver computations in one iteration.

Viscous method	Meanflow time per iteration (sec.)			SA-neg time per iteration (sec.)		
	Viscous flux	Jacobian	Linear solver	Diffusion	Jacobian	Linear solver
CBV	0.3244	1.4896	0.9404	0.0836	0.2805	0.0513
EBV	0.0411	0.1990	0.5921	0.0099	0.0207	0.0393
EBV Speed up	7.89	7.48	1.58	8.44	13.55	1.31

Another view on the same effect of a suboptimal implementation of the SA-neg-QCR2000 model is presented in Table 4. Table 4 shows the fraction of time spent by the CBV and EBV methods in the viscous-kernel and linear-solver computations. The fraction is computed from the wall time that is required to reach the converged solution from the freestream setup.

Table 4. Fractions of meanflow viscous-kernel and linear-solver computations.

Turbulence model	Viscous method	Iterations	Viscous flux fraction	Jacobian		Linear solver fraction
				Updates	Fraction	
SA-neg-QCR2000	CBV	8,805	16.7%	1,046	9.1%	48.3%
	EBV	8,737	3.8%	1,042	2.2%	55.2%
SA-neg	CBV	10,677	5.9%	1,236	4.6%	58.0%
	EBV	10,677	2.9%	1,235	2.2%	56.0%

All EBV fractions are approximately the same for both turbulence models. Only the EBV fraction for meanflow viscous fluxes has increased from 2.9% for the SA-neg model to 3.8% for the SA-neg-QCR2000 model. Note that these fractions represent an increase from the meanflow viscous-flux fraction of 1.7% reported in Ref. [4] for SA-neg solutions on prismatic-hexahedral grids for a hemisphere-cylinder configuration. The reason for this increase is not clear and may represent a case-to-case variation. The CBV fractions of meanflow viscous fluxes and Jacobian dramatically increase for the SA-neg-QCR2000 model; the viscous flux fraction increases from 5.9% to 16.7% and the Jacobian fraction increases from 4.6% to 9.1%. These increases represent the evidence of a suboptimal CPU implementation of meanflow viscous fluxes in the SA-neg-QCR2000 model. The increase of the fractions associated with the fluxes and Jacobian is so significant that it actually leads to a noticeable reduction in the fraction associated with the linear solver and (not shown) fractions associated with the turbulence model equation.

B. Juncture Flow Model (JFM)

Reference [4] reported EBV solutions on a mixed-element grid for a flow around a NASA JFM configuration. The SA-neg model used in that study is considered lacking for simulating flow separation for juncture flows; the SA-neg-QCR2000 model is recognized as a more adequate choice. Nevertheless, the SA-neg model was suitable for assessment of EBV performance on practical mixed-element grids. In this paper, the CBV and EBV solutions with the SA-neg-QCR2000 model are computed for the same grid and flow conditions using CPU and GPU architectures.

The flow conditions for this test are the following: the freestream Mach number is 0.189, the Reynolds number is 2,400,000 based on the crank chord of 557.17 mm, the reference temperature is 288.84 K, and the angle of attack is 5°. A family of grids have been generated by Pointwise® using the Glyph script package GeomToMesh [47]. The specific mixed-element grid chosen for the study is Grid I [48] that has 13,036,210 grid points, 16,645,072 tetrahedra, 20,368,758 prisms, and 112,989 pyramids. The volume and surface mesh are shown in Figure 7.

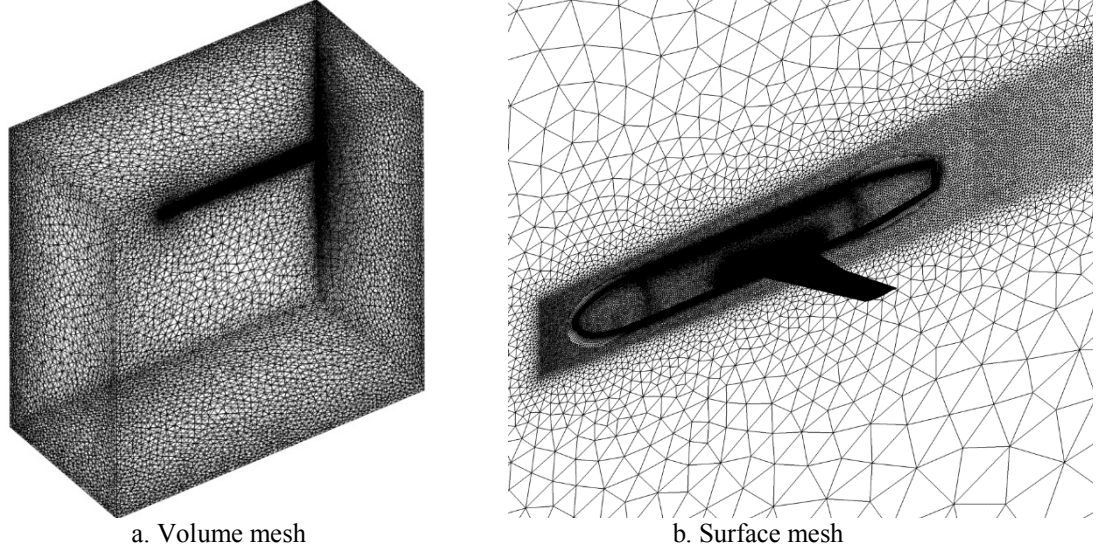


Figure 7. Volume and surface mesh for Grid I of JFM.

Ten Skylake compute nodes (400 cores) are used for CPU simulations and four NVIDIA V100 cards are used for GPU simulations to demonstrate the parallel performances of GPUs. All solutions converged the rms norm of residuals of the meanflow and turbulence-model equations below 10^{-9} . Table 5 shows the aerodynamic coefficients computed from the converged CBV and EBV solutions. The lift and drag coefficients computed on CPU and GPU with the same viscous method are matching at least 10 digits. The matching digits of the CBV and EBV aerodynamic coefficients are highlighted by the bold font; the difference between the CBV and EBV coefficients is less than 0.05%.

Table 5. Aerodynamic coefficients for JFM flow.

	Lift Coefficient		Drag Coefficient	
	CBV	EBV	CBV	EBV
CPU	0.8459285	0.8459726	0.0726731	0.0727031
GPU	0.8459285	0.8459726	0.0726731	0.0727031

To further verify the CBV and EBV solutions, variations of surface pressure coefficients (C_p) are considered in four x - z planes corresponding to $y = -254 \text{ mm}$, $y = -685.8 \text{ mm}$, $y = -1295.4 \text{ mm}$ and $y = -1663.7 \text{ mm}$. The locations are selected from the experimental measurement locations shown in Figure 8.

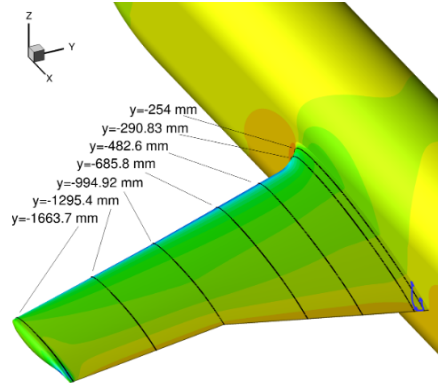
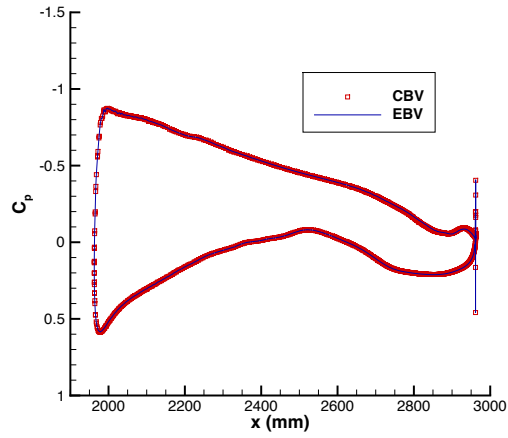
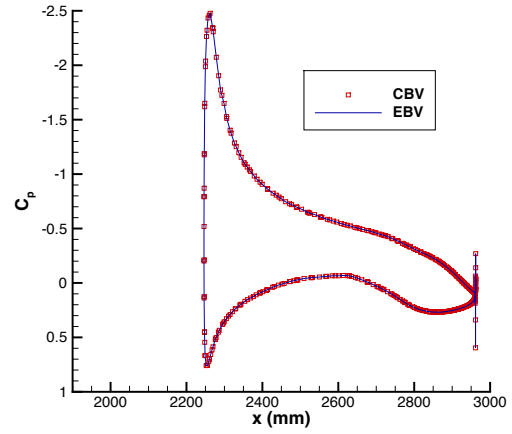


Figure 8. Locations of wing pressure measurements.

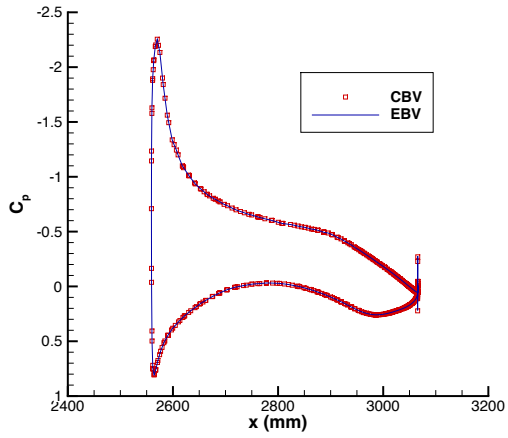
Figure 9 shows the pressure coefficients computed from the CBV and EBV solutions; the CBV solution is marked as FUN3D-FV solution in Ref. [18]. The CBV and EBV pressure plots are nearly indistinguishable in global views. Although not shown, in a zoomed-in view, there is a small difference on the airfoil upper surface at the wing tip region.



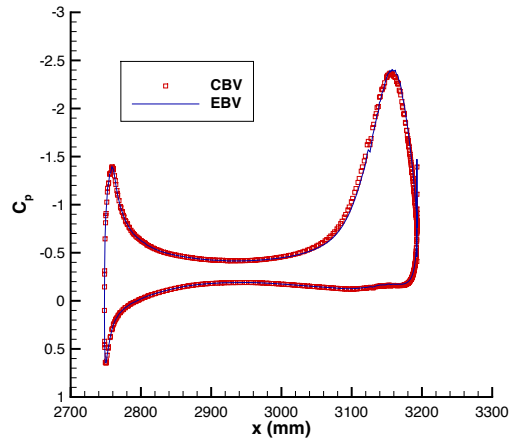
a. $y = -254 \text{ mm}$



b. $y = -685.8 \text{ mm}$



c. $y = -1295.4 \text{ mm}$



b. $y = -1663.7 \text{ mm}$

Figure. 9 Pressure coefficients distributions along wing spanwise locations

Figure 10 shows the profile of the x-component of the skin-friction coefficient computed in the plane corresponding to $y = -254 \text{ mm}$ and $y = -1663.7 \text{ mm}$. The CBV and EBV skin-friction profiles match well on the low surface of the airfoil and show a small variation on the upper surface.

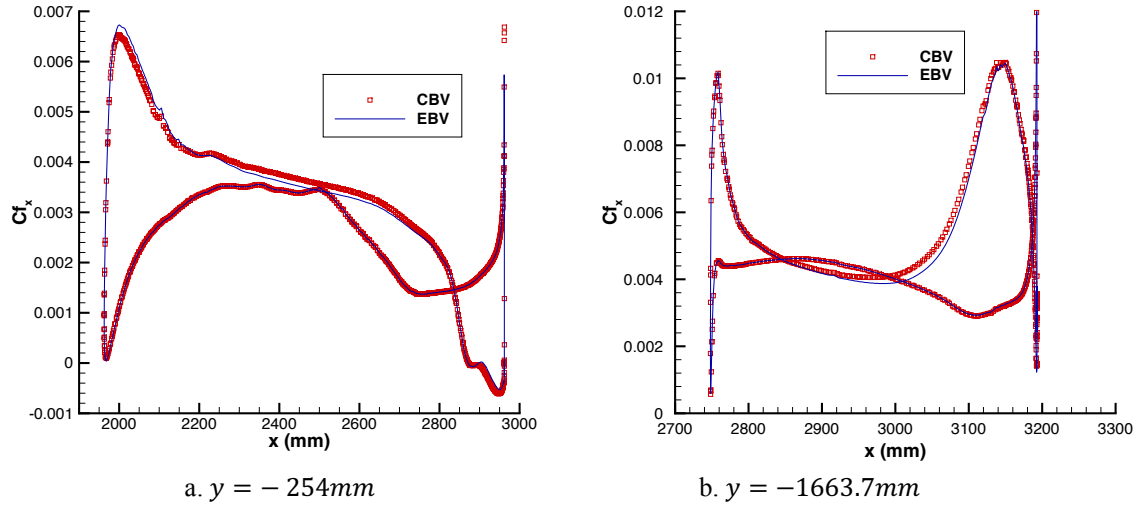


Figure 10. Skin-friction coefficients at two wing spanwise locations

Figure 11 shows residual convergence history of the CBV and EBV solutions. Figures 11a and 11b show residual convergence versus iterations on CPU and GPU, respectively. Figures 11c and 11d show the same convergence history versus wall time. The CBV and EBV convergence per iteration is indistinguishable on both CPU and GPU architectures. Similar to the results reported in Ref. [4], the EBV solutions exhibit faster asymptotic convergence rate per nonlinear iteration than the CBV solutions. Consequently, the EBV method takes significantly fewer iterations and less wall time to converged solutions with EBV method than the CBV method. Some differences in iterative convergence between EBV and CBV solvers are expected, but a significant disparity is not a general phenomenon.

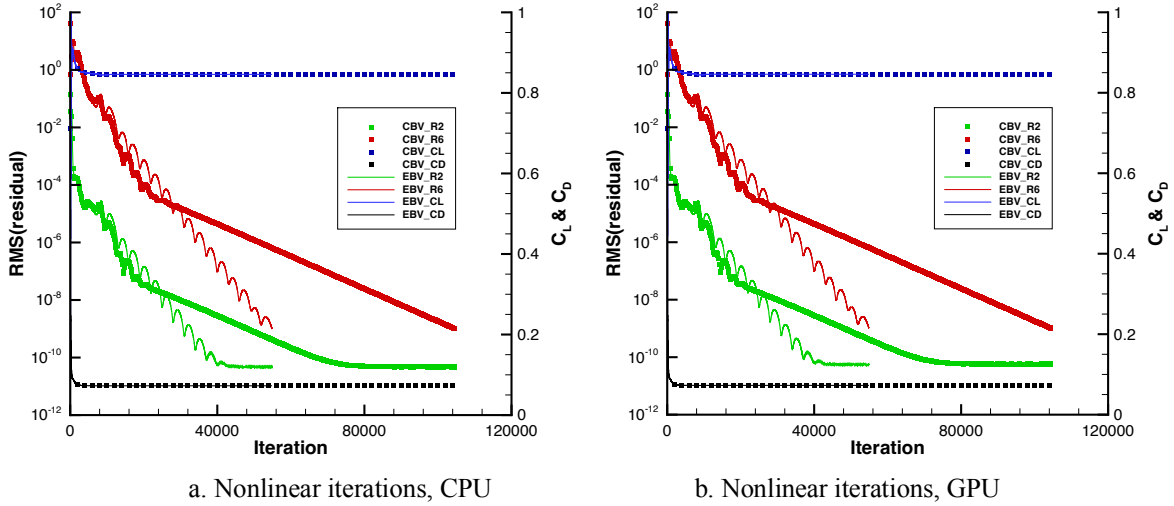


Figure 11. Iterations on JFM I grid using CPU and GPU.

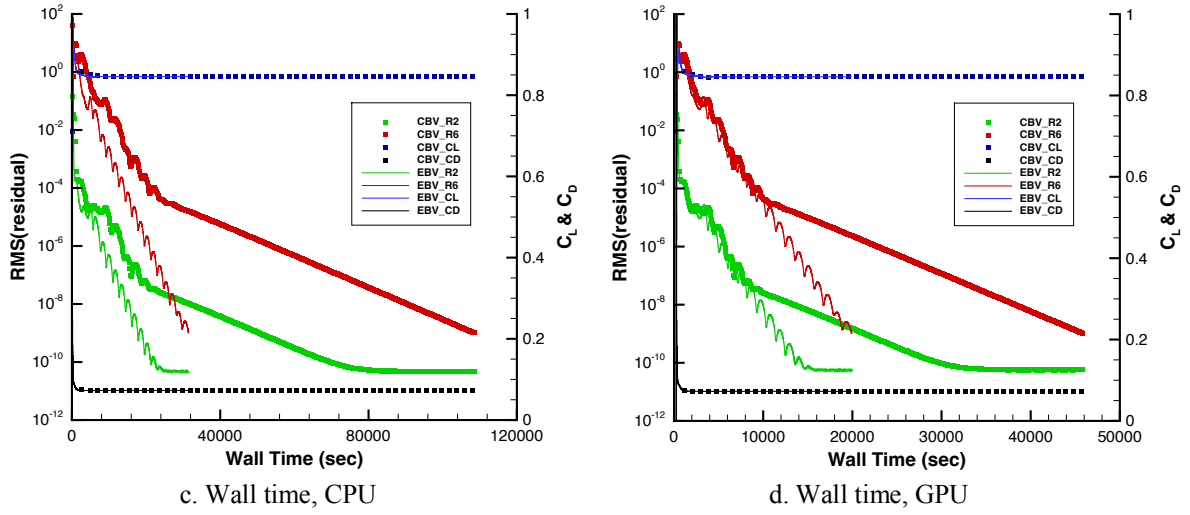


Figure 11. Concluded.

Table 6 provides statistics about the nonlinear iterations and time required to reduce the rms norm of the residual below 10^{-9} . EBV speedup of $3.5\times$ is observed for the CPU solutions and speedup of $2.3\times$ is observed for the GPU solutions. The GPU speedup factor for the CBV and EBV solutions is $2.4\times$ and $1.6\times$, respectively. A suboptimal CPU implementation of the CBV SA-neg-QCR2000 turbulence model is the reason for increased speedup for comparisons that involve this particular solution. Note that the GPU speedup is observed for four NVIDIA V100 cards over 400 Skylake CPU cores. Taking the GPU speedup for the EBV solutions and the EBV speedup for the GPU solutions as representative, the EBV method implemented on GPU reduces the time to solution by a factor of 3.7. In EBV computations that involve multiple GPU cards, the performance of a single NVIDIA V100 card is roughly similar to the performance of four Skylake nodes with 160 CPU cores.

Table 6. Time-to-solution statistics for JFM flow.

	CPU		GPU		GPU speedup
	Iterations	Wall time (sec)	Iterations	Wall time (sec.)	
CBV	104,626	108,504	104,574	45,773	2.4
EBV	54,893	31,397	54,923	19,713	1.6
EBV speedup		3.5		2.3	

VI. Concluding Remarks

An efficient edge-based viscous (EBV) method has been recently implemented in FUN3D, a practical, node-centered, unstructured-grid, finite-volume flow solver developed and supported at NASA Langley. The EBV method has been compared with an established cell-based viscous (CBV) method and demonstrated multifold speedup of all viscous-kernel computations for Reynolds-averaged Navier-Stokes (RANS) simulations using a negative variant of a linear one-equation Spalart-Allmaras (SA-neg) turbulence model. In this paper, the EBV method has been applied to a nonlinear extension of the SA-neg model that uses a quadratic constitutive relation (SA-neg-QCR2000). This RANS model is essential for accurate resolution of secondary circulation in turbulent corner flows.

The previous EBV demonstrations were performed on traditional central-processing-unit (CPU) computing systems. Recently, the EBV kernels have been ported to graphics-processing-unit (GPU) computing systems using CUDA C++ based approach. This paper has reported about verification and assessment of the EBV GPU implementation by comparing accuracy and efficiency of the GPU solutions with the corresponding CPU solutions. The solutions have been computed for two benchmark corner flows, namely, a supersonic flow through a long square duct and a subsonic flow around a NASA juncture-flow model (JFM).

The EBV SA-neg-QCR2000 solutions for the duct flow have been computed on a family of hexahedral grids. All solutions converged to machine-zero residuals. The EBV solutions compare well with the reference solutions reported in the literature and at the NASA Turbulence Modeling Resource website. The performance of CBV and EBV solvers has been assessed on a particular grid with 3.17 million grid points. The CPU solutions have been computed using

one Skylake compute node with 40 CPU cores. The GPU solutions have been computed using one NVIDIA V100 card. The CPU and GPU solutions computed with the same viscous method are indistinguishable; for example, the drag coefficients computed on CPU and GPU match up to ten digits. The CBV and EBV solutions are very similar on both CPU and GPU computing systems, as expected; the difference between CBV and EBV drag coefficients is less than 0.2%. The EBV speedup factor over the CBV solution on GPU is 1.4 $\times$. The EBV speedup factor on CPU is higher, but it is not representative because the CBV SA-neg-QCR2000 CPU implementation is suboptimal. The EBV SA-neg-QCR2000 implementation on CPU and GPU and the CBV SA-neg-QCR2000 implementation on GPU are highly optimized. The GPU speedup for the EBV SA-neg-QCR2000 implementation is 4.7 $\times$. In other words, one NVIDIA V100 card has performance that is similar to the performance of 188 Skylake CPU cores.

The JFM solutions have been computed on a relatively coarse mixed-element grid with 13 million grid points. Ten Skylake nodes (400 CPU cores) are used for CPU simulations and four NVIDIA V100 cards are used for GPU simulations. The CPU and GPU solutions computed for the same viscous method are identical. The aerodynamic coefficients computed from the converged CBV and EBV solutions differ by less than 0.05%, and the pressure coefficients and skin-friction coefficients at different spanwise locations are in good agreement. Similar to the computations with the SA-neg model reported in a previous study, the EBV SA-neg-QCR2000 solutions require significantly fewer nonlinear iterations to reduce the root-mean-square norm of the residuals below the target level of 10^{-9} . Overall, the EBV method has a speedup factor of 2.3 $\times$ on GPU. The GPU speedup factor for the EBV solutions is 1.6 $\times$. In other words, for the JFM benchmark case, one NVIDIA V100 card performs similarly to 160 Skylake CPU cores; this GPU performance is similar to the GPU performance shown for the duct flow.

In the past two years, the EBV method has been verified and provided significant efficiency benefits for multiple practical applications. However, there are still developments that are required to improve applicability and robustness of the EBV method. Robust partitioning of general cells into tetrahedra remains a challenge. Practical mixed-element grids often include nontetrahedral cells that feature highly non-planar quadrilateral faces. Formal partitioning of cells with highly non-planar quadrilateral faces may result in tetrahedra with negative volumes and complicate generation of virtual edges for the EBV method. Hierarchical adaptive nonlinear iteration method (HANIM) is a strong nonlinear solver in FUN3D that is essential for efficiency and robustness of practical solutions. In previous studies, HANIM has proved to work well with the EBV method on CPU. Currently, HANIM is being ported to GPU. Demonstration of combined HANIM-EBV performance on GPU for practical simulations is another important step. Wall-modeled large-eddy simulation (WMLES) capability has recently been implemented in FUN3D and ported to GPU. Assessment of the HANIM-EBV method for WMLES on GPU architectures is another important research topic.

Acknowledgments

The Transformative Tools and Technologies (TTT) project of the Transformative Aeronautics Concepts Program and the NASA Revolutionary Vertical Lift Technology (RVLT) project within the NASA Aeronautics Research Mission Directorate partially funded the work reported here. The first three authors were supported by the NASA Langley Research Center under cooperative agreement 80LARC17C0004 with the National Institute of Aerospace.

References

- [1] Barth T. J., "Numerical Aspects of Computing Viscous High Reynolds Number Flows on Unstructured Meshes," AIAA 91-0721, 1991. <https://doi.org/10.2514/6.1991-721>
- [2] Liu Y., Diskin B., Anderson W. K., Nielsen E. J., and Wang L., "Edge-Based Viscous Method for Node-Centered Formulations," AIAA 2021-2728, 2021. <https://doi.org/10.2514/6.2021-2728>
- [3] Liu Y., Diskin B., Nishikawa, H., Anderson W. K., Nielsen E. J., and Wang L., "Edge-Based Viscous Method for Node-Centered Finite-Volume Formulation on Tetrahedra," *AIAA Journal*, published online October 2022. <https://doi.org/10.2514/6.2022-4083>
- [4] Liu Y., Diskin B., Nishikawa H., Anderson W. K., Nastac G., Nielsen E. J., and Wang L., "Edge-Based Viscous Method for Mixed-Element Node-Centered Finite-Volume Solvers," AIAA 2022-4083, 2022. <https://doi.org/10.2514/6.2022-4083>
- [5] Diskin B., Nishikawa H., and Liu Y., "Analysis of Edge-Based Method for Diffusion," Paper ICCFD11-1043, *Eleventh International Conference on Computational Fluid Dynamics (ICCFD-11)*, Maui, HI, 2022. https://www.iccfd.org/iccfd11/assets/pdf/papers/ICCFD11_paper-1403.pdf
- [6] Biedron R. T., Carlson J. R., Derlaga J. M., Gnoffo P. A., Hammond D. P., Jones W. T., Kleb B., Lee-Rausch E. M., Nielsen E. J., Park M. A., Rumsey C. L., Thomas J. L., Thompson K. B., Walden, A. C., Wang, L., and Wood W. A., "FUN3D Manual: 13.7," NASA TM 2020-5010139, 2020.

- [7] Biedron R. T. and Thomas J. L., "Recent Enhancements to the FUN3D Flow Solver for Moving-Mesh Applications," AIAA 2009-1360, 2009. <https://doi.org/10.2514/6.2009-1360>
- [8] Nielsen E. J., and Diskin B., "High-Performance Aerodynamic Computations for Aerospace Applications," *Parallel Computing*, Vol. 64, 2017, pp. 20–32. <https://doi.org/10.1016/j.parco.2017.02.004>
- [9] Gnoffo P., "Updates to Multi-Dimensional Flux Reconstruction for Hypersonic Simulations on Tetrahedral Grids," AIAA 2010-1271, 2010. <https://doi.org/10.2514/6.2010-1271>
- [10] Wang L., Diskin B., Lopes L., Nielsen E. J., Lee-Rausch E., and Biedron R. T., "High-Fidelity Aero-Acoustic Optimization Tool for Flexible Rotors," *Journal of the American Helicopter Society*, Vol. 66, No. 2, 2021. <https://doi.org/10.4050/JAHS.66.022004>
- [11] Kleb B., Park M. A., Wood W. A., Bibb K. L., Thompson K. B., Gomez R. J. III, and Tesch S. H., "Sketch-to-Solution: An Exploration of Viscous CFD with Automatic Grids," AIAA 2019-2948, 2019. <https://doi.org/10.2514/6.2019-2948>.
- [12] Walden A., Nielsen E. J., Zubair M., Linford J. C., Wohlbier J. G., Luitjens J. P., Orender J., Beekman I., Khuvis S., and Shende S. S., "Unstructured-Grid CFD Algorithms on Many-Core Architectures," Technical Research Posters of International Supercomputing Conference for High Performance Computing, Networking, Storage, and Analysis. p. 17, 2017. https://sc17.supercomputing.org/SC17%20Archive/tech_poster/poster_files/post134s2-file3.pdf
- [13] Walden A., Nielsen E. J., Diskin B., and Zubair M., "A Mixed Precision Multicolor Point-Implicit Solver for Unstructured Grids on GPUs," 2019 IEEE/ACM 9th Workshop on Irregular Applications: Architectures and Algorithms (IA3), Denver, CO, USA, 2019, pp. 23-30. <https://doi.org/10.1109/IA349570.2019.00010>
- [14] Nastac G., Walden A., Nielsen E. J., and Frendi K., "Implicit Thermochemical Nonequilibrium Flow Simulations on Unstructured Grids using GPUs," AIAA 2021-0159, 2021. <https://doi.org/10.2514/6.2021-0159>
- [15] Walden A., Nielsen E. J., Korzum A., et al., "Landing on Mars: Petascale Unstructured Grid Computational Fluid Dynamics on Summit," *International Workshop on OpenPOWER for HPC*, ISC, 2019. <https://indico-jsc.fz-juelich.de/event/87/material/slides/8.pdf>
- [16] Stone C. P., Walden A., Zubair M., and Nielsen E. J. "Accelerating unstructured-grid CFD algorithms on NVIDIA and AMD GPUs," 2021 IEEE/ACM 11th Workshop on Irregular Applications: Architectures and Algorithms (IA3), pp. 19-26, 2021. <https://doi.org/10.1109/IA354616.2021.00010>
- [17] Wang L., Anderson W. K., Nielsen E. J., Balakumar P., Park M., Carlson J-R., Iyer P. S., and Diskin B., "Wall-Modeled Large-Eddy Simulations for High-Lift Configurations using FUN3D," AIAA Paper 2022-1555, 2022. <https://doi.org/10.2514/6.2022-1555>
- [18] Diskin B., Ahmad N. N., Anderson W. K., Derlaga J. M., Pandya M. J., Rumsey C. L., Wang L., Wood S. L., Liu Y., Nishikawa H., and Galbraith M.C., "Verification Test Suite for Spalart-Allmaras QCR2000 Turbulence Model," AIAA Paper 2021-1552, 2021. <https://doi.org/10.2514/6.2021-1552>
- [19] Diskin B., Anderson W. K., Pandya M. J., Rumsey C. L., Thomas J.L., Liu Y., and Nishikawa H., "Grid Convergence for Three Dimensional Benchmark Turbulent Flows," AIAA Paper 2018-1102, 2018. <https://doi.org/10.2514/6.2018-1102>
- [20] Rumsey C. L., Carlson J.-R., and Ahmad N. N., "FUN3D Junction Flow Computations Compared with Experimental Data," AIAA Paper 2019-0079, 2019. <https://doi.org/10.2514/6.2019-0079>
- [21] Abdol-Hamid K.S., Carlson J.-R., Rumsey C. L., Lee-Rausch E. M., and Park M. A., "Sixth Drag Prediction Workshop Results Using FUN3D with k-kL-MEAH2015 Turbulence Model," *Journal of Aircraft*, Vol. 55, No. 6, pp. 1458-1468, 2018. <https://doi.org/10.2514/1.C034481>
- [22] Diskin B., Thomas J. L., Nielsen E. J., Nishikawa H., and White J. A., "Comparison of Node-Centered and Cell-Centered Unstructured Finite-Volume Discretizations: Viscous Fluxes," *AIAA Journal*, Vol. 48, No. 7, pp. 1326-1338. 2010. <https://doi.org/10.2514/1.44940>
- [23] Thomas J. L., Diskin B., and Ramsey C. L., "Towards Verification of Unstructured-Grid Solvers," *AIAA Journal*, Vol. 46, No. 12, pp. 3070-3079, 2008. <https://doi.org/10.2514/1.36655>
- [24] Diskin B. and Thomas J. L., "Accuracy Analysis for Mixed-Element Finite-Volume Discretization Schemes," NIA Report 2007-08. https://fun3d.larc.nasa.gov/papers/nia_2007_08.pdf
- [25] Dompierre J., Labb P., Vallet M.-G., and Camarero R., "How to Subdivide Pyramids, Prisms and Hexahedra into Tetrahedra," Ecole Polytechnique Montreal, Rapport CERCA R99-78, 1999.
- [26] Nastac G., Walden A., Wang L., Nielsen, E. J., Liu Y., Opgenorth M., Orender J., and Zubair M. "A Multi-Architecture Approach for Implicit Computational Fluid Dynamics on Unstructured Grids," Paper to be presented at AIAA SciTech 2023.
- [27] Spalart P. R., "Strategies for Turbulence Modeling and Simulation," *International Journal of Heat and Fluid Flow*, Vol. 21, 2000, pp 252-263. [https://doi.org/10.1016/S142-727X\(00\)00007-2](https://doi.org/10.1016/S142-727X(00)00007-2)

- [28] Allmaras S. R., Johnson F. T., and Spalart P. R., "Modifications and Clarifications for the Implementation of the Spalart-Allmaras Turbulence Model," ICCFD7-1902, 7th International Conference on Computational Fluid Dynamics, Big Island, Hawaii, 9-13 July 2012. https://www.iccfd.org/iccfd7/assets/pdf/papers/ICCFD7-1902_paper.pdf
- [29] Spalart P. R. and Allmaras S. R., "A One-Equation Turbulence Model for Aerodynamic Flows," *Recherche Aerospatiale*, No. 1, 1994, pp. 5-21. https://turbmodels.larc.nasa.gov/Papers/RechAerosp_1994_SpalartAllmaras.pdf
- [30] Roe P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 43, No. 2, 1981, pp. 357-372. [https://doi.org/10.1016/0021-9991\(81\)90128-5](https://doi.org/10.1016/0021-9991(81)90128-5).
- [31] Burg C. O. E., "Higher Order Variable Extrapolation for Unstructured Finite Volume RANS Flow Solvers," AIAA Paper 2005-4999, 2005. <https://doi.org/10.2514/6.2005-4999>
- [32] van Leer B., "Towards the Ultimate Conservative Difference Scheme, V. A Second Order Sequel to Godunov's Method," *Journal of Computational Physics*, Vol. 32, No. 1, 1979, pp. 101-136. [https://doi.org/10.1016/0021-9991\(79\)90145-1](https://doi.org/10.1016/0021-9991(79)90145-1)
- [33] Haselbacher A. C., *A Grid-Transparent Numerical Method for Compressible Viscous Flow on Mixed Unstructured Meshes*, Ph.D. thesis, Loughborough University, 1999. <https://hdl.handle.net/2134/7257>
- [34] Nishikawa H., "Beyond Interface Gradient: A General Principle for Constructing Diffusion Schemes", AIAA 2010-5093, 2010. <https://doi.org/10.2514/6.2010-5093>
- [35] Carlson J.-R., "Inflow/Outflow Boundary Conditions with Application to FUN3D," NASA/TM-2011-217181, 2011. <https://ntrs.nasa.gov/api/citations/20110022658/downloads/20110022658.pdf>
- [36] van Leer B. "Flux-Vector Splitting for the Euler Equations," ICASE Report 82-30, 1982.
- [37] Wang L., Diskin B., Nielsen E. J., and Liu Y., "Improvements in Iterative Convergence of FUN3D Solutions," AIAA 2021-0857, 2021. <https://doi.org/10.2514/6.2021-0857>
- [38] Pandya M., Diskin B., Thomas J., and Frink N., "Assessment of USM3D Hierarchical Adaptive Nonlinear Method Preconditioners for Three-Dimensional Cases," *AIAA Journal*, Vol. 55, No. 10, 2017, pp. 1-16. <https://doi.org/10.2514/1.J055823>
- [39] Pandya M. J., Diskin B., Thomas J. L., and Frink N. T., "Improved Convergence and Robustness of USM3D Solutions on Mix-Element Grids," *AIAA Journal*, Vol. 54, No. 9, 2016, pp. 2589-2596. <https://doi.org/10.2514/1.J054545>.
- [40] Pandya M. J., Jespersen D. C., Diskin B., Thomas J. L., and Frink N. T., "Efficiency of Mixed-Element USM3D for Benchmark Three-Dimensional Flows," *AIAA Journal*, Vol. 59, No. 8, 2021, published online June 21, 2021. <https://doi.org/10.2514/1.J059720>
- [41] Pandya M. J., Jespersen D. C., Diskin B., Thomas J. L., and Frink N. T., "Verification and Scalability of Mixed-Element USM3D for Benchmark Three-Dimensional Flows," *AIAA Journal*, published online August 30, 2021. <https://doi.org/10.2514/1.J060064>
- [42] NVIDIA, "CUDA C/C++ Basics," <https://www.nvidia.com/docs/IO/116711/sc11-cuda-c-basics.pdf>, 2011. Last Accessed May 5th, 2022.
- [43] Walden A., Zubair M., and Nielsen E. J., "Performance and Portability of a Linear Solver Across Emerging Architectures," International Workshop on Accelerator Programming Using Directives, pp. 61-79, 2020. https://doi.org/10.1007/978-3-030-74224-9_4
- [44] Walden A., Zubair M., Stone C. P., and Nielsen E. J., "Memory Optimizations for Sparse Linear Algebra on GPU Hardware," IEEE/ACM Workshop on Memory Centric High Performance Computing, pp. 25-32, 2021. <https://doi.ieeecomputersociety.org/10.1109/MCHPC54807.2021.00010>
- [45] Buluç A., Fineman J. T., Frigo M., Gilbert J. R., and Leiserson C. E., "Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks," ACM Symposium on Parallelism in Algorithms and Architectures, pp. 233-244, 2009. <https://doi.org/10.1145/1583991.1584053>
- [46] Nastac G., Nielsen, E. J., Walden A., Wang L., Stephens S., Diskin B., Zubair M., Luitjens J., and Tramel B., "Recent Developments for Computational Fluid Dynamics on Unstructured Meshes Using GPUs," https://ntrs.nasa.gov/api/citations/20205007834/downloads/nasa_langley_cfd_on_gpus_rev1.pdf
- [47] Karman S. and Wyman N., "Automatic Unstructured Mesh Generation with Geometry Attribution," AIAA Paper 2019-1721, 2019. <https://doi.org/10.2514/6.2019-1721>
- [48] Rumsey C. L., Lee H. C., and Pulliam T. H., "Reynolds-Averaged Navier-Stokes Computations of the NASA Juncture Flow Model Using FUN3D and OVERFLOW," AIAA Paper 2020-1304, 2020. <https://doi.org/10.2514/6.2020-1304>