A parallel compact-TVD method for compressible fluid dynamics employing shared and distributed-memory paradigms

Vincenzo Fico[a,b,1,*], David R. Emerson[a,2], Jason M. Reese[b,3]

[a] *STFC Daresbury Laboratory, Daresbury Science and Innovation Campus, Warrington WA4 4AD, UK*

[b] *University of Strathclyde, Glasgow G1 1XQ, UK*

Abstract

A novel multi-block compact-TVD finite difference method for the simulation of compressible flows is presented. The method combines distributed and shared-memory paradigms to take advantage of the configuration of modern supercomputers that host many cores per shared-memory node. In our approach a domain decomposition technique is applied to a compact scheme using explicit flux formulas at block interfaces. This method offers great improvement in performance over earlier parallel compact methods that rely on the parallel solution of a linear system. A test case is presented to assess the accuracy and parallel performance of the new method.

Keywords: compact methods, TVD schemes, shock-capturing schemes, parallel computing

[*]Corresponding author

Email address: `vincenzo.fico@stfc.ac.uk` (Vincenzo Fico)

[1]Ph.D. Student

[2]Professor, Computational Science and Engineering Department

[3]Weir Professor of Thermodynamics and Fluid Mechanics, Department of Mechanical Engineering

1. Introduction

Large Eddy Simulation (LES) and Direct Numerical Simulation (DNS) of fluid flows require high accuracy numerical schemes which must be capable of resolving a very broad range of length scales that are often orders of magnitude apart. For this purpose, compact methods have been developed that guarantee spectral-like resolution. However, compact schemes have a semi-global nature, as a linear system must be solved to compute spatial derivatives along grid lines. Efficient parallelisation of these schemes therefore represents a significant challenge. Previously, Sun and Moitra [1] and Povitsky [2] devised parallel algorithms for the solution of the underlying linear system, but as highlighted by Ladeinde *et al.* [3], these algorithms scale poorly at high processor counts. Improved scalability can be achieved if a structured-block domain decomposition technique is employed: Gaitonde and Visbal [4] and Sengupta *et al.* [5] have employed compact schemes to advance independently the solution in overlapping subdomains. Laizet *et al.* [6] have proposed a dual-domain decomposition, namely *slab* decomposition, which changes during a single time step depending on the spatial direction processed. Their incompressible DNS code showed good parallel performance, even though in some tests communication took up to 40% of the total simulation time because of the global data transposition implied by the dual decomposition.

The class of compact upwind methods developed by Pirozzoli [7] have opened new possibilities for the application of a classical and efficient parallel multi-block strategy to compact schemes. These compact schemes compute a numerical flux function, unlike other compact schemes which compute derivatives, and so a multi-block parallelisation approach can be implemented by simply enforcing the continuity of the numerical flux function across block-interfaces. Chao *et al.* [8] have exploited this possibility by developing a multi-block compact method. Their parallel algorithm is based on a compact-WENO scheme employing domain decomposition. We have extended the compact-TVD method developed by Tu and Yuan [9] for inviscid flows to viscous flows using the Navier-Stokes Kinetic Flux Vector Splitting [10] technique. In this paper, we present a parallel algorithm based on domain decomposition employing our compact-TVD method [11].

The parallelisation strategies discussed are task-based and rely on the SPMD (Single Program Multiple Data) programming model. The modern trend in High Performance Computing is to increase the number of cores sharing the memory on a compute node. In a shared memory environment thread-based parallelism is possible, based on the SMP (Symmetric Multi-Processing) programming model. For large-scale fluid dynamics calculations employing compact schemes, this is particularly interesting, as efficient parallelisation in a distributed memory environment is possible by weakly relaxing the global dependency of the scheme (this point is clarified in Section 3.1). As noted by Chao *et al.* [8], this will affect the global resolution properties of the scheme as the number of cores is increased [8]. In a shared memory environment, parallelisation is in principle possible without modifying the numerical method (and so preserving the resolution properties), even though attention must be payed in order to achieve good parallel performance. Many investigations into dual-level parallelism have been pubblished. For example, Huan and Tafti [12] combined multi-threading and multi-tasking to achieve a good load balance when the governing equations are solved using Adaptive Mesh Refinement. Gropp *et al.* [13] have shown that decreasing the number of tasks and increasing the number of threads can speed up explicit flux computation as the mesh size increases. Recently, Rabenseifner *et al.* [14] have shown that a dual-level parallelism mapped onto the machine topology can significantly speed up a parallel tridiagonal block CFD solver. We have developed a code exploiting both distributed and shared memory models. Two different parallelisation strategies are employed, depending on the memory model, and are nested in a funnelled mode as detailed in Section 3. The paper is structured as follows: in Section 2 we present the governing equations; in Section 3 we describe the parallelisation procedures for both distributed and shared-memory paradigms; in Section 4 we present a test case and analyse the difference between the single-block and the multi-block solutions, along with the parallel performance.

2. The Governing Equations

The method we have developed [11] solves the compressible Navier-Stokes equations in general coordinates. Here we present a two-dimensional inviscid test, so the governing equations are the two-dimensional Euler equations, whose strong conservation form in general coordinates (ξ, η) is:

$$\partial_t \tilde{Q} + \partial_\xi \tilde{E} + \partial_\eta \tilde{F} = 0, \tag{1}$$

where

$$\tilde{Q} = 1/J \left\{ \rho, \rho u, \rho v, \rho E \right\}^T, \tag{2}$$

$$\tilde{E} = 1/J \left\{ \rho V_\xi, \rho V_\xi u + p\xi_x, \rho V_\xi v + p\xi_y, \rho V_\xi H \right\}^T, \tag{3}$$

$$\tilde{F} = 1/J \left\{ \rho V_\eta, \rho V_\eta u + p\eta_x, \rho V_\eta v + p\eta_y, \rho V_\eta H \right\}^T . \tag{4}$$

In the above relations t is time, $\xi_x, \xi_y, \eta_x, \eta_y$ are the metrics and J the Jacobian of the transformation, ρ is the density, p the pressure, u and v are, respectively, the components of the velocity vector along cartesian coordinates x and y, $E = e + (u^2 + v^2)/2$ is the total energy, $H = E + p/\rho$ is the total enthalpy, $e = p/[(\gamma - 1)\rho]$ is the internal energy, $V_\xi = u\xi_x + v\xi_y$ and $V_\eta = u\eta_x + v\eta_y$ are, respectively, the ξ and η components of the velocity vector in the transformed space.

3. Numerical method

Parallelisation of the scheme developed by the authors [11] in a distributed-memory environment is achieved via the classical and efficient *structured-block* domain decomposition procedure: the computational domain is partitioned into several structured blocks and each block is assigned to a different process. The algorithm, discussed in Section 3.1, employs the *Message Passing Interface*, MPI. In a distributed-memory environment this technique is preferred to slab decomposition [6] because it avoids the data transposition step and requires less communication which leads to a better parallel performance. However, we use the slab decomposition approach to parallelise our compact-TVD method [11] over a compute-node because, in a shared-memory environment, it can be implemented easily and without the overhead of data transposition and communication. Our implementation is based on OpenMP and the algorithm is described in Section 3.2. The shared-memory parallel algorithm is nested into the distributed-memory parallel algorithm in a *funnelled* mode. The computational domain is partitioned into several blocks, each of them assigned to a different MPI process. Each MPI process is the master thread in a team of OpenMP threads, which process the block in parallel according to the algorithm described in Section 3.2. Master threads processing different blocks communicate using MPI according to the algorithm described in Section 3.1. Slave threads processing the same block communicate using shared memory, while slave threads processing different blocks do not communicate.

3.1. Distributed-memory parallel algorithm

The communication between different processes is achieved by providing each block with ghost cells, where the solution is not computed by the MPI process the current block is assigned to, but received from the MPI process assigned to the adjacent block. Domain decomposition cannot be applied in principle to our compact-TVD method [11], as the values of the high order numerical flux function at cell faces along a grid line are coupled. We follow the same approach as Chao *et al.* [8] and use fifth-order explicit upwind formulas at inter-block boundaries. By employing such formulas, consistency with the compact upwind formulas employed for the internal faces is retained, and no loss in formal accuracy is implied by the decomposition. While Chao *et al.* [8] use vertex-centred grids, we employ cell-centered grids. This allow us to enforce just the continuity of the numerical flux function, while they must enforce the continuity of the flux divergence across blocks, which in turn implies the continuity of the numerical flux function. In our case, explicit formulas are employed only once per inter-block boundary per grid line, whereas in their case they are employed twice.

Consider, for instance, the computation of $\partial_\xi \tilde{E}$, and suppose boundary

$i = 1/2$ is an inter-block boundary. If the fluid variables are cell-centred, the high-order numerical flux function $\hat{E} = \hat{E}^{+} + \hat{E}^{-}$ is computed by:

$$\hat{E}^{+}_{1/2} = 1/60 \left(2\tilde{E}^{+}_{-2} - 13\tilde{E}^{+}_{-1} + 47\tilde{E}^{+}_{0} + 27\tilde{E}^{+}_{1} - 3\tilde{E}^{+}_{2}\right), \tag{5}$$

$$\hat{E}^{-}_{1/2} = 1/60 \left(-3\tilde{E}^{-}_{-1} + 27\tilde{E}^{-}_{0} + 47\tilde{E}^{-}_{1} - 13\tilde{E}^{-}_{2} + 2\tilde{E}^{-}_{3}\right). \tag{6}$$

Provided the fluid dynamic variables in the cells with indices from -2 to 3 in the current block indexing are consistent across the two neighbouring blocks, the same flux at the shared face is independently computed by the processes handling the blocks. The resulting multi-block scheme is conservative over the whole domain with minimum modification of the original method. If fluid dynamic variables are computed at cell vertices instead, as vertex $i = 1$ is shared among the blocks, the two processes must compute independently fluxes $\hat{E}^{\pm}_{1/2}$ and $\hat{E}^{\pm}_{3/2}$ in order to compute the same value of $\left(\partial_\xi \tilde{E}\right)_1$. Explicit formulas must then be used for both $\hat{E}^{\pm}_{1/2}$ and $\hat{E}^{\pm}_{3/2}$. In both cases, at each block boundary, three layers of ghost cells are required to retain the upwind bias and the formal accuracy of the inner compact scheme.

3.2. Shared-memory parallel algorithm

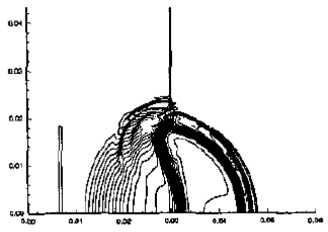
A slab decomposition approach has been implemented over a compute-node, as follows. Consider a two-dimensional structured grid made of $N_\xi \times N_\eta$ cells. For each coordinate η_j, a one-dimensional compact-TVD reconstruction along ξ of the numerical flux function is performed in order to compute $\partial_\xi \tilde{E}$. As the reconstructions are independent, the domain is partitioned in slabs along the η direction, so that each thread performs reconstructions along ξ for the grid lines belonging to the slab it is assigned with. In a similar fashion, the computation of $\partial_\eta \tilde{F}$ is divided among the threads partitioning the domain in slabs along the ξ-direction, so that each thread performs reconstructions along η. As the memory is shared among the threads, no explicit communication is implied by the overlapping between the domains assigned to the threads in the ξ and η partitions. Unlike MPI-based parallelisation, there is no approximation involved with the OpenMP approach, which makes it attractive in Direct Numerical Simulation of turbulence, where breaking the global dependence could have a significant impact on the results.

4. Numerical test

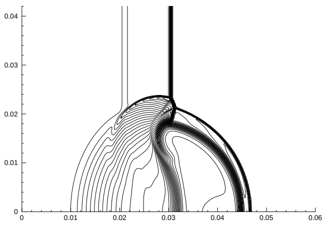
In this section we present a numerical test. Our main goals are: (i) to assess the accuracy of our compact-TVD method through direct comparison with numerical schemes of established accuracy, (ii) to study the effect of the structured-block domain decomposition on the computed results, and (iii) to assess the parallel performance of the method when distributed and shared-memory algorithms are combined. The first and second points are covered in Section 4.1, while the third point is addressed in Section 4.2.

4.1. Computation of shock-jet interaction

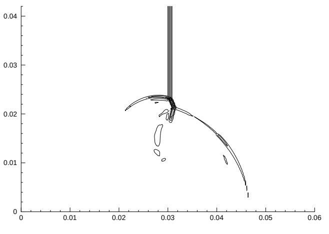
In this section we present the simulation of a Mach 2 planar shock wave propagating in air and interacting with a cylindrical jet. The test case was designed by Don and Quillen [15], who investigated the ability of finite difference ENO and spectral schemes to predict a complex process to enhance the performance of combustion engines. The set up is the same as described by Don and Quillen [15], but, while they model the presence of different species, having a hydrogen jet surrounded by air, we consider a single species with density variations. As the test case is inviscid, there is no species diffusion;



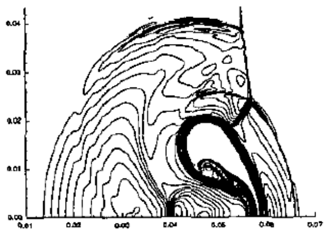
(a) Density at $t = 20\mu s$, spectral method



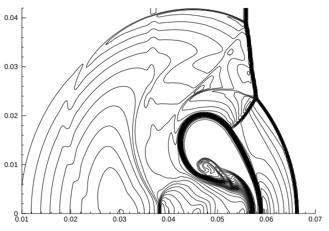
(b) Density at $t = 20\mu s$, CU5TVD-SB



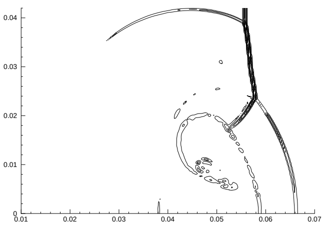
(c) Multi-block density difference at $t = 20\mu s$



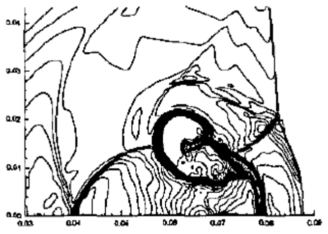
(d) Density at $t = 40\mu s$, spectral method



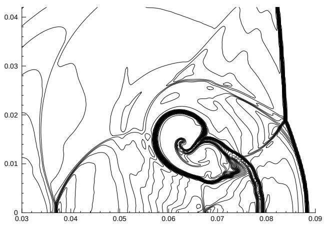
(e) Density at $t = 40\mu s$, CU5TVD-SB



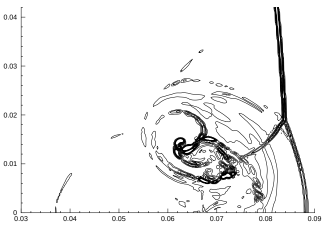
(f) Multi-block density difference at $t = 40\mu s$



(g) Density at $t = 60\mu s$, spectral method



(h) Density at $t = 60\mu s$, CU5TVD-SB



(i) Multi-block density difference at $t = 60\mu s$

Figure 1: Density contours for the shock-jet interaction at indicated times: left, spectral method of Don and Quillen [15]; middle, present single-block compact-TVD method; right, difference between the multi-block and single-block compact-TVD results.

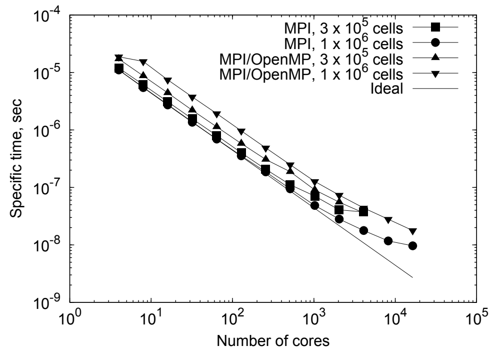
hydrogen and air are both modelled as diatomic gases, so the isentropic index is $\gamma = 7/5$ for both. So no major difference due to the different modelling is expected to appear in the computed fluid dynamic variables. In discussing the results we use the following acronyms: CU5TVD-SB refers to the inviscid compact-TVD method applied on a single-block structured grid; CU5TVD-MB refers to the inviscid compact-TVD method applied on a multi-block structured grid.

In Fig. 1 density contours computed by CU5TVD-SB on 752×376 cells are shown in the middle column. For comparison, we retain the resolution used by Don and Quillen [15] in their fifth-order ENO computation. The computed solution is compared to the results obtained by Don and Quillen using a spectral code, shown in the left column of Fig. 1. The spectral calculation has an equivalent resolution of about 800×512 grid points on the domain considered here [15], as it was carried out with fewer grid points on a smaller domain. For the density contour plots, we use the same levels as Don and Quillen i.e. 40 contour lines equally spaced between 0.05 and 1.2 kg/m^3, so comparison between the left and the middle columns in Fig. 1 is actually quantitative. The results of CU5TVD-SB clearly agree with those of the spectral code. It is not surprising that CU5TVD-SB gives a superior description of waves and shocklets at early times ($t = 20, 40\mu s$), as it was designed to compute shocks. However, it is surprising that the internal structure of the jet at time $t = 60\mu s$ predicted by CU5TVD-SB closely resembles the one predicted by the spectral code. This demonstrates that the TVD filter is very effective close to discontinuities and does not destroy the small scale structures. CU5TVD-SB resolves the feature of the jet better than the fifth order ENO scheme employed by Don and Quillen [15], whose results are not reported here, even though the ENO scheme is uniformly high accurate.

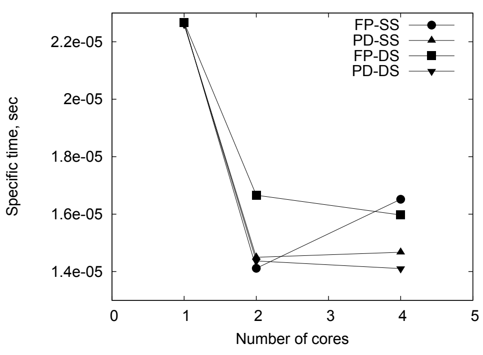
In the right column of Fig. 1 is shown the difference between the density computed by CU5TVD-SB and the one computed by CU5TVD-MB. For the multi-block calculation the domain was divided into 10×10 blocks along x and y, respectively. The density difference plots have 20 contour lines equispaced between -0.1 and 0.1 kg/m^3. At early times ($t = 20, 40\mu s$) the density difference contours map on shocks and contact discontinuities. The internal structure of discontinuities is not resolved and their thickness is purely a numerical artefact, so no error is introduced by the domain partitioning at this stage of the simulation. Discontinuities and waves are correctly propagated across blocks. As the simulation proceeds and vortical structures begin to form within the jet, small density differences are detected close to these structures. This confirms that breaking the global dependency of the compact scheme and using explicit formulas with lower resolution properties at block interfaces does affect the computation of vortical structures.

4.2. Parallel performance

We have made an assessment of the parallel performance of the multi-block compact-TVD method for the shock-jet interaction problem. The simulations have been run on a Cray XT4 system of the UK National Supercomputing Service, HECToR (`www.hector.ac.uk`). The XT4 comprises 1,416 compute blades, each of which has 4 quad-core processor sockets. This amounts to a total of 22,656 cores, each of which acts as a single CPU. The processor is an AMD 2.3 GHz Opteron. The point-to-point bandwidth is 2.17 GB/s and the latency between two processors is around 6 μs. Strong



(a) MPI vs MPI/OpenMP specific time



(b) OpenMP specific time

Figure 2: Strong scaling test. (a): MPI and MPI/OpenMP modes. (b): OpenMP mode.

scalability tests have been run for different problem sizes. A coarse grid of 752×376 cells and a fine grid of 1504×752 cells have been considered. The simulations have been run on an increasing number of cores, starting with 4 and doubling the number at each stage. For a fixed number of cores and grid resolution, each simulation has been run first in pure MPI mode and then in hybrid MPI/OpenMP mode. In pure MPI mode, the domain was partitioned into as many blocks as cores. In hybrid MPI/OpenMP mode, the domain was partitioned into as many blocks as shared-memory nodes, and each block was processed in parallel by the four cores belonging to the node, each of them assigned with an OpenMP thread. Fig. 2(a) shows the results in terms of *specific time*, defined as the time taken to simulate one time step, divided by the number of cells. A parallel efficiency can be defined as the ratio between the specific time and the ideal specific time, which scales linearly with the number of cores. In pure MPI mode, efficiencies of about 90% are achieved on 256 cores for the coarse grid and on 1024 cores for the fine grid, which is an excellent result. As the number of cores increases the work-load per core decreases, the time spent in core-to-core communication becomes comparable to the time each core spends in computation and eventually begins to dominate it. For the coarse grid this happens at about 4000 cores, whereas for the fine grid the code is still scaling at 16000 cores. The parallel efficiency in hybrid MPI/OpenMP mode is below 70% of that in pure MPI mode if the same number of cores is employed. At the highest core counts the communication time in pure MPI mode dominates the computing time and a better efficiency is achieved in hybrid mode because less communication is required. This is clearly shown for the coarse mesh, whereas for the fine mesh it would likely happen at a core count higher than the maximum number of cores currently available.

We have examined the possibility of improving the performance of the shared-memory kernel. Specific times obtained with the different configurations are shown in Fig. 2(b). First, more tasks have been parallelised, such as boundary condition enforcement, variable initialisation and update. In Fig. 2(b), FP (full parallel) refers to this code configuration, whereas PD refers to the case in which just the flux divergence and the stable time step are computed in parallel. Second, as the OpenMP parallelism is mostly based on loop unrolling, we have considered different scheduling strategies, namely static and dynamic scheduling. In Fig. 2(b), the terms SS and DS refer to static and dynamic scheduling, respectively. The specific times shown in Fig. 2(a) for the hybrid MPI/OpenMP were based on runs employing PD-SS configuration for the shared-memory kernel.

An insight into the shared-memory algorithm behaviour is gained from analysis of Fig. 2(b). First, static scheduling is quicker for low core counts, whereas dynamic scheduling is quicker for high core counts. This is likely due to the fact that the overhead introduced by the dynamic scheduling is overwhelmed by the benefit of minimising the time threads lie idle, just if the number of threads is sufficiently high. Second, parallelising non-computationally-intensive operations degrades the performance at high core counts, but has a different effect for different scheduling strategies at low core counts. Finally, good scalability is achieved on two cores but this quickly degrades as the core count increases. This is likely to be related to the non-optimal data placement in Non-Uniform Memory Access architectures [16]. This issue will be addressed in the future.

5. Conclusions

We have developed a parallel algorithm based on a compact-TVD method using both distributed and shared-memory paradigms. The accuracy of the method has been assessed on a complex fluid dynamic test case. Our results agree with those obtained using a spectral code and the domain decomposition does not have a significant impact on the simulated field. Excellent scalability properties are demonstrated and employing a shared-memory kernel within a compute node improves the performance over a pure distributed-memory algorithm at high core counts. Data placement within a shared-memory node needs further attention in order to fully exploit the potential of modern multi-core supercomputers.

6. Acknowledgments

The authors would like to thank Steve Daley of the Defence Science and Technology Laboratory (Dstl), UK. This work is funded by the Engineering and Physical Sciences Research Council (EPSRC) and Dstl under grant no. GR/T05028/01. Additional support was provided by EPSRC through CCP12. Jason M. Reese would like to thank the Royal Academy of Engineering and the Leverhulme Trust for support through a Senior Research Fellowship.

References

[1] X. H. Sun, S. Moitra, A fast parallel tridiagonal algorithm for a class of CFD applications, Technical Paper 3585, NASA, 1996.

[2] A. Povitsky, Parallel directional split solver based on reformulation of pipelined Thomas algorithm, Technical Report CR-1998-208733, NASA, 1998.

[3] F. Ladeinde, X. Cai, M. Visbal, D. Gaitonde, International Journal of Computational Fluid Dynamics 17 (2003) 2886–2898.

[4] D. Gaitonde, M. Visbal, High-order schemes for Navier-Stokes equations: algorithm and implementation into FDL3D, Technical Report AFRL-VA-WP-RT-1998-3060, Air Force Research Laboratory, Write-Patterson Air Force Base, 1998.

[5] T. K. Sengupta, A. Dipankar, A. Kameswara Rao, Journal of Computational Physics 220 (2007) 654–677.

[6] S. Laizet, E. Lamballais, J. Vassilicos, Computers & Fluids 39 (2010) 471 – 484.

[7] S. Pirozzoli, Journal of Computational Physics 178 (2001) 81–117.

[8] J. Chao, A. Haselbacher, S. Balachandar, Journal of Computational Physics 228 (2009) 7473–7491.

[9] G. H. Tu, X. J. Yuan, Journal of Computational Physics 225 (2007) 2083–2097.

[10] S. Y. Chou, D. Baganoff, Journal of Computational Physics 130 (1997) 217–230.

[11] V. Fico, D. R. Emerson, J. M. Reese, in: Proceedings of 48th AIAA Aerospace Science Meeting, Orlando, FL, US (2010). AIAA-2010-0875.

[12] W. Huang, D. K. Tafti, in: Proceedings of Parallel CFD 1999, Williamsburg, VA, US (1999).

[13] W. D. Gropp, D. K. Kaushik, D. E. Keyes, B. F. Smith, Parallel Computing 27 (2001) 337 – 362.

[14] R. Rabenseifner, G. Hager, G. Jost, in: 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing, Weimar, Germany (2009).

[15] W. S. Don, C. B. Quillen, Journal of Computational Physics 122 (1995) 244–265.

[16] M. Nordén, H. Sverker, M. Thuné, Future Generation Computer Systems 22 (2006) 194–203.