SOLVCON: A Python-Based CFD Software Framework for Hybrid Parallelization

Yung-Yu Chen,[*] David L. Bilyeu,[†] Lixiang Yang[‡] and Sheng-Tao John Yu[§]

Department of Mechanical and Aerospace Engineering, The Ohio State University, Columbus, Ohio, 43210

SOLVCON is a new, open-sourced software framework for high-fidelity solutions of linear and non-linear hyperbolic partial differential equations. SOLVCON emphasizes scalability, portability, and maintainability for supercomputing by using emerging multi-core architectures. The code development effort follows Extreme Programming practices, including version control, documentation, issue tracking, user support, and frequent code releases. In SOLVCON, the Python framework includes all supportive functionalities for the work flow. For pre-processing operations, the Python framework provides parallelized mesh data input and automatically sets up domain decomposition. In calculations, the Python framework provides light-weight memory management through extensive use of pointers. Computation-intensive operations are implemented by using C and FORTRAN for high performance. The default numerical algorithm employed is the space-time Conservation Element and Solution Element (CESE) method. The code uses general unstructured meshes with mixed elements, including tetrahedra, hexahedra, prisms, and pyramids for three-dimensional calculations. Hybrid parallelism includes shared- and distributed-memory parallelization. The temporal loop and the spatial loop in modern finite-volume methods are implemented in a two-layered structure in SOLVCON. Distributed-memory parallelization by domain decomposition and MPI is performed in the temporal loop. Shared-memory parallel computing by using accelerator technologies, e.g., General-Purpose Graphic Processor Unit (GPGPU), is performed in the spatial loop. More than 99% of the execution time of SOLVCON is used for number-crunching in the solver as a part of the space loop. Written in C or FORTRAN, a typical solver contains only 10% of the code statements in SOLVCON. To demonstrate the capabilities of newly developed SOLVCON, we performed CFD calculations by using 23 million elements. The code was run on a 512-core cluster. SOLVCON delivers calculations of flow variables in 11.29 million elements per second. The parallel efficiency is 70%. In the open-sourced SOLVCON, two solvers are available: (i) the Euler equations solver for compressible flows, and (ii) the velocity-stress equations solver for waves in anisotropic elastic solids. SOLVCON can be easily extended for other applications, including viscous flows, aero-acoustics, nonlinear solid mechanics, and electromagnetism. The Python framework allows fast adaption to new heterogeneous, multi-core hardware as well as further development of the code for peta-scale supercomputing.

I. Introduction

Hyperbolic partial differential equations (PDEs) are extensively used to model wave motions and advective transport of substances. Numerical methods for time-accurate solutions of hyperbolic PDEs is the foundation of modern Computational Fluid Dynamics (CFD) methods for compressible flows. Hyperbolic PDEs also arise in a wide range of non-fluid applications, including acoustics, solid mechanics, electromagnetism, optics, etc. In the past decades, mathematical theories and numerical methods for solving hyperbolic PDEs[1–3] have become mature. The modern approaches cast the model equations into a set of fully-coupled, first-order, hyperbolic PDEs, in which the physics are described by the Jacobian matrices of the equations.

[*]Ph.D. Candidate, AIAA Student Member, `chen.1352@osu.edu`.

[†]Ph.D. Student, AIAA Student Member, `bilyeu.4@osu.edu`.

[‡]Ph.D. Candidate, `yang.1130@osu.edu`.

[§]Associate Professor, AIAA Member, `yu.274@osu.edu`.

As such, strong commonality exists in the theoretical framework for a wide range of hyperbolic systems with drastically different applications.

For CFD, researchers have developed monolithic legacy codes to aid important scientific and engineering studies. Well-known aerodynamics codes developed by NASA include FUN3D,[4] WIND-US,[5] National Combustion Code (NCC),[6] etc. These legacy codes share the following common attributes: (i) These CFD codes are hard-wired for fluid mechanics only and cannot be easily adapted to model different physical processes. However, the numerical methods employed, in general, are fully capable of solving other hyperbolic processes. (ii) These codes use unstructured meshes with mixed elements,[7] and a large part of the code is to manage the data of unstructured mesh. The code statements directly related to physical models and numerical methods are much less than those of the supporting functionalities for manipulating data of unstructured meshes. However, few legacy codes clearly separate the solver from supporting functionalities. Without a unclear software structure, the process of adapting these legacy codes to achieve hybrid parallelism will be laborious. (iii) In the past, legacy codes have been renovated twice for adapting to new hardware. Decades ago, these codes were parallelized for shared-memory, vector machines, i.e., Cray computers. Later, these codes were migrated to Beowulf clusters by using MPI for distributed-memory parallel computing. At the present time, all legacy codes will need to be migrated to heterogeneous platforms based on multi-core technologies.

Recently, the accelerator-based, multi-core technology, especially General-Purpose Graphic Process Unit (GPGPU) computing, became a mainstream approach for supercomputing. According to the current Top 500 list (http://top500.org/), the fastest supercomputer in the world, Tianhe-1A, is a GPU cluster. Similarly to vector processors, GPGPU computing is based on shared-memory parallelization. Clustering, on the other hand, is based on distributed-memory parallelization. The use of a GPGPU cluster implies combination of shared- and distributed-memory parallelization, i.e., hybrid parallelism.

The emerging multi-core technologies motivate us to pursue a futuristic architecture of supercomputing CFD. To date, the state-of-the-art approach is running CUDA codes on clusters equipped with NVIDIA GPUs. However, the multi-core technologies are evolving very rapidly. There are many advanced multi-core technologies in the pipeline. Undoubtedly, there will be other viable options for hybrid parallelism. Next-generation CFD codes must be adaptive to new multi-core hardware. To this end, SOLVCON has been developed based on a clear organization of the software structure, such that the coding effort to achieve shared-memory parallel computing is confined to the solver kernel, i.e., a very small part of the code. As a result, SOLVCON is portable for various platforms and very easy to be maintained.

SOLVCON is a software framework developed by using the Python programming language.[8] Segregation between the Python framework and the solver kernel allows us to use SOLVCON to solve various non-fluid physical processes as long as the model equations can be cast into the first-order hyperbolic PDEs, including all wave motions and convective transport of substances. In recent years, the modern numerical methods originally developed for fluid mechanics have been widely adopted to solve non-fluid problems, including stress waves in solids[9–12] and electromagnetic waves.[13, 14] However, few CFD codes were developed with a clear software structure such that it can be easily extended for multi-physics.

SOLVCON is designed for multi-physics. The software framework of SOLVCON is based on the theoretical framework of the first-order hyperbolic PDEs and modern finite-volume methods. All supportive functionalities, a major part of SOLVCON, is segregated from the PDE solvers. The code for those functionalities is written by using Python, and designed to be reused for the hyperbolic PDE solvers. In particular, the Python framework manages unstructured mesh, parallel processing, and input and output. The solver kernels in SOLVCON, on the other hand, are tailored to solve specific physical models of interest. The solver kernels are made pluggable to the Python framework. In other words, the Python framework provides an environment with all supporting functionalities, mainly for manipulating the mesh data, to support the number-crunching tasks performed by the solver.

Another concern for the next-generation CFD codes is the extensive man power required for a supercomputing task. For most of modern parallelized CFD codes, when using more than 50 million cells in a task, many difficulties were encountered, including lower parallelization efficiency, very slow data input and output, and almost insurmountable difficulties in post-processing the results when using commercial post-processing tools. Tremendous man power must be spent in setting up parallel computing, transmitting and post-processing the solutions of unsteady flows, and graphics or animation rendering. In addition to the required runtime of computational tasks, laborious work flow and required man power are indeed the bottleneck of a CFD supercomputing task. The Python framework of SOLVCON is designed to streamline the work flow by organizing all supportive functionalities, including parallel input and output, automatically setting up domain decomposition. In the future, we will develop in situ animation for processing the results. The Python software structure is designed to streamline the work flow of a supercomputing task. The use of SOLVCON greatly increase the productivity for performing large-scale simulations.

To recapitulate, SOLVCON is a new software framework for time-accurate solutions of hyperbolic PDEs and conservation laws. Scalability for running on supercomputers is emphasized. The software framework is developed by using Python, a dynamic, object-oriented programming language for high flexibility. The Python framework provides the necessary supportive functionalities for PDE solvers and parallel computing, so that the developers can focus on the physical models and numerical methods. Because of a clear software structure, solver kernels are made pluggable to the Python framework. The computational efficiency is not impaired. Three specific goals in the design of SOLVCON are: (i) All supportive functionalities are *completely segregated from* the numerical algorithms and the model equations, such that solver kernels are made pluggable to the Python framework. (ii) The Python code provides *automatic distributed-memory parallelization* through setting up domain-decomposition and message-passing. The goal here is to enable highly productive work flows. (iii) To achieve *hybrid parallelism* systematically for advanced computing technologies, e.g., GPGPU clusters. Essentially, only the solver kernels of SOLVCON need to be rewritten for the new hardware employed.

The rest of the paper is organized in the following. Section II illustrates the model equations and the CESE method used in SOLVCON. Section III describes the software structure of SOLVCON and its object-oriented approach. Section IV explains how automatic distributed-memory parallelization and hybrid parallelism are achieved. Section V provides the benchmark results. Section VI provides concluding remarks.

II. Model Equations and Numerical Methods

This section provides a brief review of the first-order, hyperbolic PDEs and the multi-physics CESE method[15] for numerical solutions.

II.A. Hyperbolic PDEs

A set of coupled, first-order, three-dimensional hyperbolic PDEs can be formulated in a vector form:

$$\frac{\partial \mathbf{u}}{\partial t} + \sum_{i=1}^{3} \frac{\partial \mathbf{f}^i(\mathbf{u})}{\partial x_i} = 0, \tag{1}$$

where t is time, $\mathbf{u} = (u_1, \dots, u_N)^T$ is a column vector of N unknowns, x_1, x_2, and x_3 are the three spatial Cartesian coordinates, and $\mathbf{f}^1, \mathbf{f}^2$, and $\mathbf{f}^3$ are the flux functions. Aided by the chain rule and formulated in a component form, Eq. (1) becomes

$$u_{mt} + \sum_{i=1}^{3}\sum_{l=1}^{N} a^i_{m,l} u_{l,i} = 0, \quad m = 1, \dots, N, \tag{2}$$

where

$$u_{mt} \overset{\text{def}}{=} \frac{\partial u_m}{\partial t}, \quad u_{l,i} \overset{\text{def}}{=} \frac{\partial u_l}{\partial x_i}, \quad a^i_{ml} \overset{\text{def}}{=} \frac{\partial f^i_m}{\partial u_l},$$

$i = 1, 2, 3$, and $m, l = 1, \dots, N$. Eq. (2) can be further rewritten in the following vector-matrix form:

$$\frac{\partial \mathbf{u}}{\partial t} + \sum_{i=1}^{3} A^i(\mathbf{u}) \frac{\partial \mathbf{u}}{\partial x_i} = 0, \tag{3}$$

where the component at the m-th row and the l-th column of the matrix A^i is a^i_{ml}, for $i = 1, 2, 3$. A^1, A^2, and A^3, are the three Jacobian matrices associated with the x_1, x_2, and x_3 coordinates, respectively. If A^1, A^2, and A^3 are functions of $\mathbf{u}$, the PDEs are non-linear. Otherwise, the PDEs are linear.

The formulation of the Jacobian matrices A^1, A^2, and A^3 in the first-order hyperbolic PDEs, Eq. (3), determines the physical processes. Physical significance of the governing equations are represented by the three Jacobian matrices. SOLVCON takes advantage of this mathematical structure. The PDE solvers of multiple physical processes can be implemented with the governing equations provided by the users of SOLVCON.

To solve Eq. (3), modern finite-volume solvers perform explicit time-marching calculations. For each time step, solvers enforce flux conservation over each cell in the spatial domain. The overall numerical algorithms consist of two mandatory loops: (i) the temporal loop for time-marching and (ii) the spatial loops for flux calculation in the spatial domain. We entitle this as the intrinsic *two-loop structure* for hyperbolic PDE solvers. "Cells" are the fundamental unit for the discretized space, i.e., the mesh, for the numerical method to impose the constraint specified by the PDEs. It can be expected that almost all the execution time of the solvers is used in the spatial loop.

II.B. The CESE Method

The CESE method solves first-order, non-linear or linear, hyperbolic PDEs by using the unstructured meshes in two- and three-dimensional space.[16, 17] Details of the CESE method such as the numerical algorithm and the stability criteria can be found in the literature.[15, 18–20]

The basic idea of the CESE method is to enforce flux conservation in the space-time domain. In particular, the CESE method treats space and time as one entity. For conciseness, we only briefly review the one-dimensional version of the CESE method. The one-dimensional model equations take the form of Eq. (1) but have only one spatial coordinate:

$$\frac{\partial u_m}{\partial t} + \frac{\partial f_m}{\partial x} = 0, \quad m = 1, \ldots, N, \tag{4}$$

where t is time, x is the spatial coordinate, and f_m is the flux function. Let $\xi_1 = x$ and $\xi_2 = t$ be the coordinates of the Euclidean space E^2. By defining a vector $\mathbf{h}_m = (f_m, u_m)$ in E^2, it can be shown that Eq. (4) is equivalent to:

$$\oint_{S(V)} \mathbf{h}_m \cdot d\mathbf{s} = 0, \quad m = 1, \ldots, N, \tag{5}$$

where V is an arbitrary space-time area in E^2 and $S(V)$ is the boundary of V.

The CESE method defines discrete conservation elements (CEs) to cover the space-time domain and enforces flux conservation over each CE by using Eq. (5). The integration of fluxes passing the boundary of each CE is facilitated by the definitions of solution elements (SEs), in which u_m and f_m, $m = 1, \ldots, N$ are assumed to have linear distribution. Discontinuity is allowed across SEs. With the given initial condition, the hyperbolic PDEs are solved through the explicit time-marching scheme. The unknown variables at the new time step are calculated based on the flux conservation over the CEs.

Both non-linear and linear equations are accommodated in the CESE method. In general, the functions f_m, $m = 1, \ldots, N$ are non-linear. Compared to other finite-volume-based methods, the CESE method does not use Riemann solvers which are specific to physical models to capture shocks. The shocks in non-linear equations are captured by the weighting functions in the CESE method.[15] As such, the CESE method fully utilizes the abstraction of physical models provided by the Jacobian matrices in the first-order formulation of hyperbolic PDEs.

II.C. Mixed-Language Approach

To aid the construction of multi-physics PDE solvers for various applications, a flexible implementation of SOLVCON is important. The demanded flexibility can be hardly achieved by using a high-performance programming language, e.g., C and FORTRAN. The framework of SOLVCON is written by using Python.[8] Python has been used in many HPC applications.[21–24] However, plain Python is not appropriate to computation-intensive, number-crunching tasks, because it uses a virtual machine for high-level language features, e.g., dynamic typing systems and metaclassing. In order to extract the highest-possible performance from the hardware, code for number-crunching tasks needs to be written in C or FORTRAN. Over 70% of statements in SOLVCON are written in Python.

The *mixed-language* approach in SOLVCON resolves the performance issue. The resultant code achieves high flexibility by Python framework and high performance by C or FORTRAN code. Within the overall software architecture defined by Python, the high-performance C or FORTRAN code provide optimal

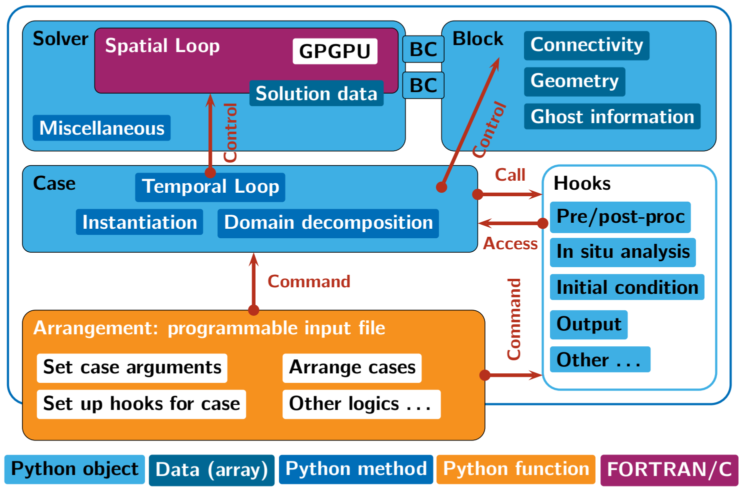

Fig. 1. The structure of the framework materialized in SOLVCON. The schematic is from bottom up.

computation-intensive operations. This mixed-language approach is achieved by taking advantage of (i) the two-loop structure of modern finite volume methods for solving hyperbolic PDEs, and (ii) the easy-to-use foreign function interface (FFI) and other mixed-language functionalities provided by Python,[25, 26] e.g., the `ctypes` library.

The following important benefits are obtained by using Python for the overall software framework: (i) Legacy input files for PDE solvers are no longer needed. As an interpreted language, Python code is automatically compiled before execution. Python scripts can be used as the entry points of PDE solver programs, and replace the role of legacy input files. (ii) The dynamic typing system of Python enables high flexibility. Reflective programming is easy. Some functionalities, e.g., restart, can be implemented more freely. (iii) The metaclassing facilities in Python increase the degree of modularity of SOLVCON. (iv) The network communication functionalities in the standard library can be readily used for fast prototyping of message-passing.

III. Software Model

SOLVCON uses object-oriented programming to model the hyperbolic PDE solvers. SOLVCON has four mandatory constructs: (i) unstructured mesh, (ii) two-loop structure, (iii) supportive functionalities, and (iv) invocation of the programs. Figure 1 illustrates the main structure of SOLVCON.

The `Block` class implements the cell-centered data structure of unstructured meshes with mixed elements,[7] which is the most fundamental construct in SOLVCON. It also implements the related manipulations. The data structure closely regulates the implementation of the spatial loops. Three primary entities should be defined for the mesh: (i) node, (ii) face, and (iii) cell. A node represents a location in the space. A face consists of several nodes, and a cell in turn consists of several faces. The overall spatial domain is covered by a set of non-overlapping cells. The mesh can have cells of different shapes. Conventionally, triangles and quadrilaterals are used in two-dimensional space, and tetrahedra, prisms, pyramids, and hexahedra are used in three-dimensional space. These entities are stored in arrays and assigned unique indices for reference. The `Block` class also incorporates the data structure for the ghost cells, which is used for boundary-condition treatment.[3]

The `Case` and `Solver` classes implement the temporal and spatial loops, respectively, to build up the two-loop structure. Various supportive functionalities are wrapped around the `Case` objects, i.e., the temporal loop, including execution control, input and output, in situ data processing, etc. The `Solver` class is responsible for materializing the numerical algorithm in the spatial loop to form a *solving kernel.* As required by the two-loop structure, the `Case` and `Solver` objects work together closely. The `Solver` object will be manipulated by the `Case` object per each time step.

The various supportive functionalities are mostly implemented with a hierarchy of `Hook` classes. The `Hook` classes provide a systematic approach to organize the highly diverse supportive functionalities. A `Hook` class implements a specific functionality, e.g., outputting the current time step. Unrelated functionalities can be implemented in different `Hook` classes. Any set of `Hook` objects can be *installed* on a `Case` object to meet the demand from a simulation. During the simulation, the `Case` object calls the callback methods pre-defined in the installed `Hook` objects at appropriate positions. As such, aided by an optionally installable `Hook` object, the computer codes to determine to use the functionality or not, can be moved to be very far from the numerical algorithms, even outside the two-loop structure.

To provide a mechanism to flexibly invoke the software constructed by the foregoing `Block`, `Case`, `Solver`, and `Hook` classes, a construct called `Arrangement` is designed in SOLVCON. The `Arrangement` needs to collect parameters to run a simulation, instantiate the `Case` object with the parameters, and install desired `Hook` objects on the `Case` objects. The behavior of the `Arrangement` shows no object-oriented features. Therefore, rather than the object-oriented paradigm, an `Arrangement` is implemented with the *imperative paradigm* as a Python `callable`. It should be noted that, although object-oriented programming has been proven as a powerful concept for software modelling, and especially suitable for building software frameworks, blind application of the object-oriented paradigm is harmful. The `Arrangement` is not implemented with object-oriented programming to avoid unnecessary complexity.

The section completes the outline of the software structure of SOLVCON. One important objective of the current design is to enable the infrastructure for *solving packages* that consist of modularized solving kernels and coupled supportive functionalities. Aided by the capability of inheritance provided in Python, concrete numerical algorithms for various applications can be implemented using subclasses of the `Solver` class. Customized supportive functionalities can be materialized by subclassing the `Hook` classes. The combination of these subclasses can form multiple solving packages that share a large portion of reusable codes reside in SOLVCON. The codes in SOLVCON are responsible for complex operations for data manipulation and parallelization.

IV. Domain Decomposition and Message-Passing

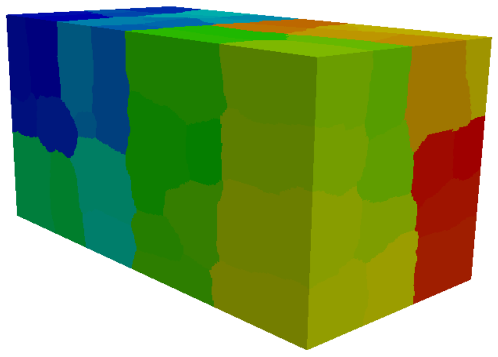

Fig. 2. Decomposed three-dimensional computation domain. The decomposition is done in SOLVCON by directly calling the METIS library.[27]

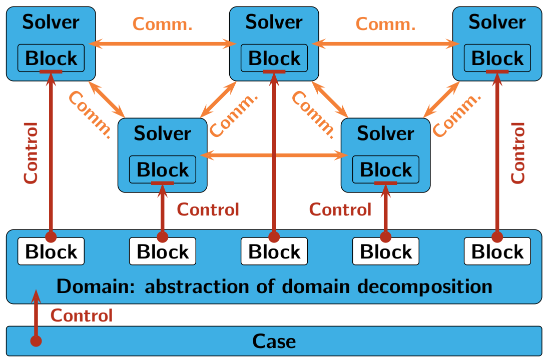

Fig. 3. Automatic domain decomposition and message-passing parallelization handled by the framework.

Distributed-memory parallelization is made automatic by SOLVCON. Once a serial code is written with SOLVCON, *domain decomposition* and *message-passing* functionalities are readily usable. The domain decomposition is based on the data structure of the unstructured meshes with mixed elements. The connectivity of the unstructured mesh is used to build a graph of cells, and the graph is partitioned by using the METIS library.[27] The mesh can be further decomposed into multiple sub-domains according to the partitioned graph. The decomposed sub-domains are the basic unit to distribute computational loads to a set of networked workstations for parallel processing. Then, the workstations perform the distributed-memory parallel computing through message-passing across the network. Figure 2 demonstrates a decomposed mesh.

SOLVCON is responsible for both domain decomposition and message-passing. Although these two functionalities are purely supportive, they are too fundamental to be implemented as `Hook` classes. The domain decomposition is implemented in a standalone module named `domain` and managed by the `Case` class. The message-passing is more complex and needs to have codes in both `Case` and `Solver` classes, i.e., the temporal and spatial loops, respectively. A network communication layer is implemented in SOLVCON to:

(i) transfer objects among the controlling workstation and slave workstations, and (ii) exchange information on the interface between adjacent sub-domains. SOLVCON provides two switchable communication layer. One directly uses TCP/IP socket, and the other calls Message-Passing Interface (MPI) library. As such, SOLVCON can operate in unconventional parallel environment which does not have MPI installed. Figure 3 depicts the distributed-memory parallelization materialized in SOLVCON.

Hybrid parallelism joins both distributed-memory and shared-memory parallelization, while the latter is the fundamental parallel paradigm for modern multi-core computer architecture and advanced many-core GPGPU computing. Although hybrid parallelism is important for further scale-up in the modern HPC architecture, the accompanying complexity needs to be mitigated. The software model shown in Section III prevents obfuscation in the hybrid parallel implementation in SOLVCON. Aided by the distributed-memory parallelization handled automatically in SOLVCON, shared-memory parallelization can be insulated in the solving kernels. In practice, the functionalities for distributed-memory parallelization reside in the `Case` class and the base `Solver` class defined in SOLVCON, while the shared-memory parallelization is put in the customized subclass of `Solver` class defined in the solving kernels. Undesired interference is eliminated by the strict isolation.

V. Performance Results

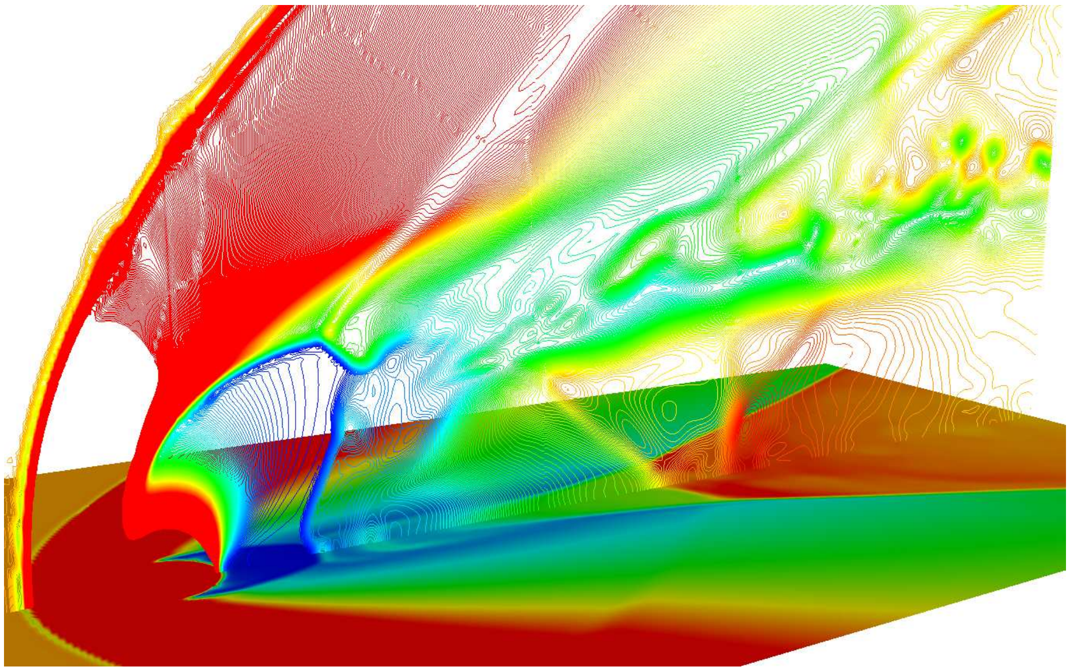

Fig. 4. Pressure contours of simulated jet in cross flow. The free stream conditions are $\rho = 0.86\,\mathrm{kg/m^3}$, $p = 41.8\,\mathrm{kPa}$, and $M = 1.98$. The jet conditions are $\rho = 6.64\,\mathrm{kg/m^3}$, $p = 476\,\mathrm{kPa}$, and $M = 1.02$. The simulation uses 23 millions cells and computation is completed within one day.

In this section, we provide benchmark results for the parallel performance of SOLVCON. To validate the modularized structure of the software, we have developed two PDE solvers for: (i) the Euler equations, and (ii) the velocity-stress equations.[28] In this paper, we focuses on the Euler solver. The performance is analyzed against a three-dimensional model problem of supersonic jet in cross flow, as shown in Fig. 4. Three different sizes of mesh are used: 3 million, 16 million, and 23 million elements. The benchmark is run on the two major partitions of the Glenn cluster in Ohio Supercomputer Center. One partition contains dual-socket, dual-core (4-core) 2.6 GHz Opteron with 8 GB ram. The other partition contains dual-socket, quad-core (8-core) 2.5 GHz Opteron with 24 GB ram. The interconnect is 10 Gbps or 20 Gbps Infiniband. The benchmark used up to 64 computer nodes, or 512 cores. The highest speed measured is 11.29 million elements per second by using 64 8-core nodes with 23 million elements, as shown in Fig. 5. The measured performance is in the unit of million elements per second (Meps).

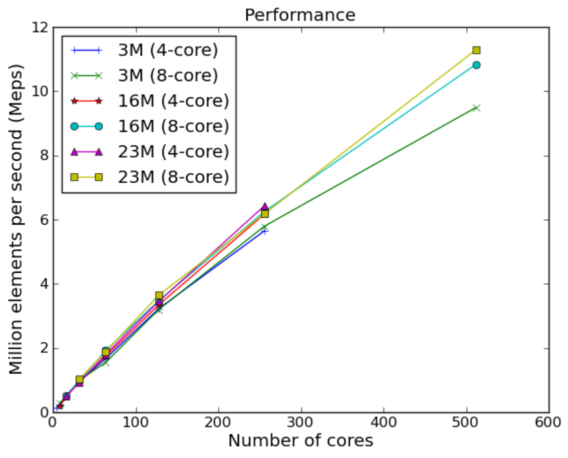

Fig. 5. Performance of the simulations using meshes with 3, 16, and 23 million elements, and on 4-core and 8-core nodes. The highest performance achieved is 11.29 million elements per second.

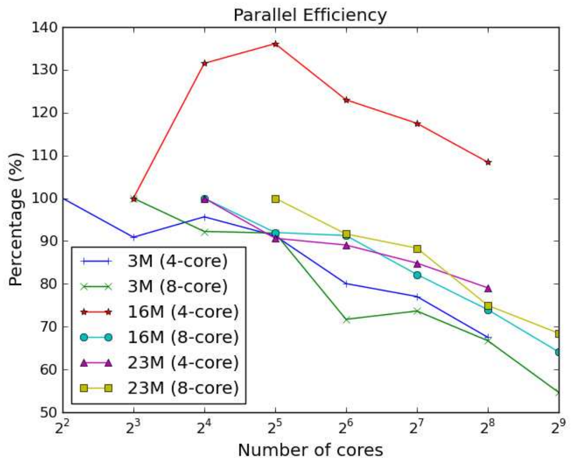

Fig. 6. The parallel efficiency. For the largest simulation using 23 million elements with 512 cores, the parallel execution maintains a good efficiency of 70%.

We further calculate the parallel efficiency for SOLVCON. We define the parallel efficiency:

$$\eta(\phi) = \frac{p(\phi)}{p(\phi_r)}\frac{\phi_r}{\phi}, \tag{6}$$

where ϕ is the number of nodes used in the simulation, $p(\phi)$ is the performance measured by using ϕ cores, and ϕ_r is the referential node number. Constrained by memory size, for larger runs using 16 and 23 million elements, the minimum nodes for the simulations are 2 and 4, respectively. Thus the referential number ϕ_r for 3, 16, and 23 million elements are 1, 2, and 4, respectively. The calculated parallel efficiency is plotted in Fig. 6. The efficiency of the runs of 16 million elements on 4-core partition is more than 100%, because the swapping occurred in the reference case slowed it down. The case used only 2 nodes, in which the memory is not sufficient to calculate without swapping. For the largest simulation which uses 23 million elements with 512 cores on the 8-core partition, the parallel efficiency is 70%. Since the CESE method uses explicit time-marching, data transfer must be performed for each time step. The overhead in communication is not negligible. Thus, 70% of parallel efficiency for 512 core is a good achievement.

We provide more detailed analysis by the weak scaling of SOLVCON. To measure the weak scaling, the same amount of elements should be used on each computer node. Because SOLVCON uses arbitrarily generated unstructured mesh, number of elements on each computer node cannot be made even for different mesh size. Instead, we configure the simulations to have approximately 1 million elements on each computer node. The weak scaling of SOLVCON is presented in Fig. 7. The y-axis is the performance of each computer node measured in Meps. The two curves in Fig. 7 are measured for the 4-core and 8-core hardware partitions. The performance per node of the 8-core partition is twice of that of the 4-core partition. For benchmark on the 4-core partition, the scaling curve is almost flat, which means perfect weak scaling. For the 8-core partition, the curve is just slightly inclined. Based on the good result in weak scaling, we show that SOLVCON is capable to simulate much larger problems.

The strong scaling is shown in Fig. 8. To measure the strong scaling, the mesh size is fixed for each test run, and only the number of computer nodes is changed. Figure 8 shows the strong scaling of 6 benchmark sets, for the three meshes on the two hardware partitions. The mesh size is positively related to the speed-up, because communication overhead takes smaller portion of the overall execution time when larger mesh is used. SOLVCON also scales well in terms of the strong scaling. For problems of moderate size, such as those of 16 and 23 million elements, SOLVCON allows quick turnaround time by adding more computer nodes.

VI. Conclusions

We have successfully developed SOLVCON, a new software framework for high-fidelity numerical solution of hyperbolic PDEs and conservation laws. As a multi-physics code, SOLVCON can be used for high-fidelity

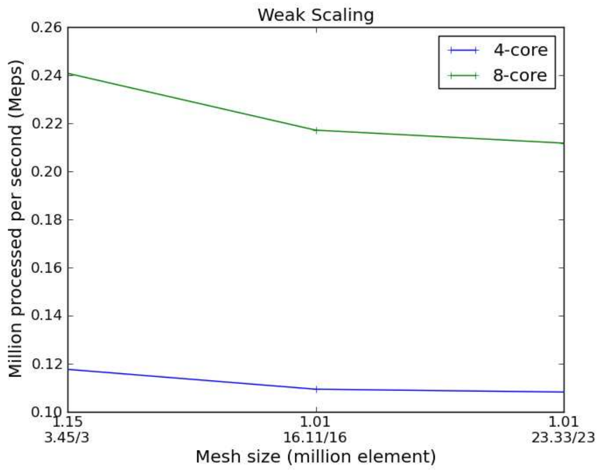

Fig. 7. The weak scaling for 3, 16, and 23 computer nodes on the 4-core and 8-core hardware partitions. Approximately 1 million elements are maintained on each computer node. The *y*-axis is the performance per computer node. SOLVCON scales well on both partitions. For the 4-core partition, the utilized cores are 12, 48, 92. For the 8-core partition, the core counts are 24, 96, 184.

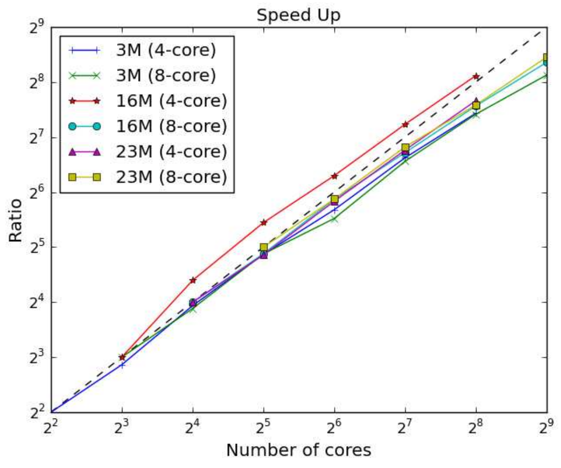

Fig. 8. The strong scaling for meshes of 3 million, 16 million, and 23 million elements. The dashed line is the reference of linear scaling. SOLVCON scales close to linear. Up to 512 (2^9) cores are used. Each line in the plot uses a single mesh, but different number of computer nodes. Due to the memory constraint, starting point of each line is different.

solutions of all wave motions as well as convective transport of substances as long as the model equations can be cast into a set of first-order, hyperbolic PDEs or in the form of coupled convection-diffusion equations. SOLVCON can use any density-based finite-volume method. The default method employed in SOLVCON is the space-time CESE method. The software structure of SOLVCON is designed based on the space-and-time, two-loop structure of the modern finite-volume methods in solving the hyperbolic PDEs. As such, the solver kernel, which is embedded in the spatial loop, is completely segregated from the overarching software framework, which manages the work flow in the temporal loop. The framework is written by using Python, while the solver kernels are written by high-performance languages, e.g., C and FORTRAN. As such, we achieves pluggable multi-physics without sacrificing computing performance. Another goal of SOLVCON is to improve productivity of the work flow of supercomputing tasks. As a part of SOLVCON, MPI and domain decomposition are automated, such that a user of SOLVCON can focus his/her effort on developing numerical algorithms and physical models. All tasks about setting up parallel computing on supercomputers are taken care of by SOLVCON.

Aided by the organized software structure, hybrid parallelism can be systematically achieved with limited coding efforts for a wide range of multi-core hardware. In the open-sourced SOLVCON, we simultaneously use pthread and MPI for hybrid parallel computing. When using SOLVCON to solve the three-dimensional Euler equations, SOLVCON delivers CESE calculations of flow variables of 11.29 million cells per second by using 512 CPU cores. In SOLVCON, we use an elaborate Python environment to streamline mesh input, domain decomposition, solution analysis, and data output. The Python framework provides a robust foundation for in situ visualization in the future works. Developed from ground up, SOLVCON is indeed a futuristic software structure for advanced accelerator-based supercomputing by using multi-core, heterogeneous platforms.

Acknowledgement

The authors are indebted to the Ohio Supercomputer Center for technical support and supercomputing resource.

References

[1]Godlewski, E. and Raviart, P., *Numerical Approximation of Hyperbolic Systems of Conservation Laws*, Springer, New York, 1996.

[2]Kulikovskii, A. G., Pogorelov, N. V., and Semenov, A. Y., *Mathematical Aspects of Numerical Solution of Hyperbolic Systems*, No. 118 in Chapman & Hall/CRC monographs and surveys in pure and applied mathematics, Chapman & Hall/CRC, Boca Raton, 2001.

[3]LeVeque, R. J., *Finite-Volume Methods for Hyperbolic Problems*, Cambridge texts in applied mathematics, Cambridge University Press, Cambridge, 2002.

[4]FUN3D, "FUN3D Manual," http://fun3d.larc.nasa.gov/.

[5]Wind-US, "Wind-US Documentation," http://www.grc.nasa.gov/WWW/winddocs/index.html.

[6]NCC, "National Combustion Code Used To Study the Hydrogen Injector Design for Gas Turbines," http://www.grc.nasa.gov/WWW/RT/2004/RT/RTB-iannetti.html.

[7]Mavriplis, D. J., "Unstructured Grid Techniques," *Annual Review of Fluid Mechanics*, Vol. 29, Jan. 1997.

[8]van Rossum, G., "Python Programming Language," http://python.org/, 1991.

[9]de la Puente, J., Käser, M., Dumbser, M., and Igel, H., "An arbitrary high-order discontinuous Galerkin method for elastic waves on unstructured meshes; IV. Anisotropy," *Geophysical Journal International*, Vol. 169, No. 3, 2007, pp. 1210–1228.

[10]Dumbser, M., Käser, M., and de la Puente, J., "Arbitrary high-order finite volume schemes for seismic wave propagation on unstructured meshes in 2D and 3D," *Geophysical Journal International*, Vol. 171, No. 2, 2007, pp. 665–694.

[11]Yu, S. J., Yang, L., Lowe, R. L., and Bechtel, S. E., "Numerical simulation of linear and nonlinear waves in hypoelastic solids by the CESE method," *Wave Motion*, Vol. 47, No. 3, April 2010, pp. 168–182.

[12]Yang, L., Chen, Y., and Yu, S. J., "Velocity-Stress Equations for Waves in Solids of Hexagonal Symmetry Solved by the Space-Time CESE Method," *ASME Journal of Vibration and Acoustics*, Vol. in press, 2010.

[13]Shi, Y. and Liang, C., "The finite-volume time-domain algorithm using least square method in solving Maxwell's equations," *Journal of Computational Physics*, Vol. 226, No. 2, Oct. 2007, pp. 1444–1457.

[14]Hermeline, F., Layouni, S., and Omnes, P., "A finite volume method for the approximation of Maxwell's equations in two space dimensions on arbitrary meshes," *Journal of Computational Physics*, Vol. 227, No. 22, Nov. 2008, pp. 9365–9388.

[15]Chang, S., "The Method of Space-Time Conservation Element and Solution Element – A New Approach for Solving the Navier-Stokes and Euler Equations," *Journal of Computational Physics*, Vol. 119, No. 2, July 1995, pp. 295–324.

[16]Wang, X. and Chang, S., "A 2D Non-Splitting Unstructured Triangular Mesh Euler Solver Based on the Space-Time Conservation Element and Solution Element Method," *Computational Fluid Dynamics Journal*, Vol. 8, No. 2, 1999, pp. 309–325.

[17]Zhang, Z., Yu, S. T. J., and Chang, S., "A Space-Time Conservation Element and Solution Element Method for Solving the Two- and Three-Dimensional Unsteady Euler Equations Using Quadrilateral and Hexahedral Meshes," *Journal of Computational Physics*, Vol. 175, No. 1, Jan. 2002, pp. 168–199.

[18]Chang, S. and To, W., "A new numerical framework for solving conservation laws: The method of space-time conservation element and solution element," Tech. Rep. E-6403; NAS 1.15:104495; NASA-TM-104495, Aug. 1991.

[19]Chang, S., "On an origin of numerical diffusion: Violation of invariance under space-time inversion," E-7066; NAS 1.15:105776; NASA-TM-105776, July 1992.

[20]Chang, S., "On Space-Time Inversion Invariance and its Relation to Non-Dissipatedness of a CESE Core Scheme," Sacramento, CA, United States, July 2006, p. 35.

[21]Sanner, M. F., "Python: A Programming Language for Software Integration and Development," *Journal of Molecular Graphics and Modelling*, Vol. 17, No. 1, Feb. 1999, pp. 57–61.

[22]Cai, X., Langtangen, H. P., and Moe, H., "On the performance of the Python programming language for serial and parallel scientific computations," *Sci. Program.*, Vol. 13, No. 1, 2005, pp. 31–56.

[23]Langtangen, H., "A Case Study in High-Performance Mixed-Language Programming," *Applied Parallel Computing. State of the Art in Scientific Computing*, 2008, pp. 36–49.

[24]Langtangen, H. P. and Cai, X., "On the Efficiency of Python for High-Performance Computing: A Case Study Involving Stencil Updates for Partial Differential Equations," *Modeling, Simulation and Optimization of Complex Processes*, 2008, pp. 337–357.

[25]van Rossum, G. and Drake, F. L. J., "The Python Standard Library," http://docs.python.org/library/index.html, 2010.

[26]van Rossum, G. and Drake, F. L. J., "Extending and Embedding the Python Interpreter," http://docs.python.org/extending/index.html, 2010.

[27]Karypis, G. and Kumar, V., "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM Journal on Scientific Computing*, Vol. 20, No. 1, Jan. 1998, pp. 359–392.

[28]Chen, Y., Yang, L., and Yu, S. J., "Simulations of Waves in Elastic Solids of Cubic Symmetry by the Conservation Element and Solution Element Method," *Wave Motion*, Vol. 48, No. 1, Jan. 2011, pp. 39–61.