fluids

MDPI

Article

Complex-Geometry 3D Computational Fluid Dynamics with Automatic Load Balancing

József Bakosi [1,*], Mátyás Constans [1], Zoltán Horváth [1], Ákos Kovács [1], László Környei [1], Marc Charest [2], Aditya Pandare [2], Paula Rutherford [2] and Jacob Waltz [2]

[1] Széchenyi Egyetem, University of Győr, 9026 Győr, Hungary
[2] Los Alamos National Laboratory, Los Alamos, NM 87545, USA
* Correspondence: bakosi.jozsef@sze.hu

Abstract: We present an open-source code, Xyst, intended for the simulation of complex-geometry 3D compressible flows. The software implementation facilitates the effective use of the largest distributed-memory machines, combining data-, and task-parallelism on top of the Charm++ runtime system. Charm++'s execution model is asynchronous by default, allowing arbitrary overlap of computation and communication. Built-in automatic load balancing enables redistribution of arbitrarily heterogeneous computational load based on real-time CPU load measurement at negligible cost. The runtime system also features automatic checkpointing, fault tolerance, resilience against hardware failure, and supports power- and energy-aware computation. We verify and validate the numerical method and demonstrate the benefits of automatic load balancing for irregular workloads.

Keywords: computational fluid dynamics; finite element method; Charm++; automatic load balancing; unstructured grids; high-speed flows

Citation: Bakosi, J.; Constans, M.; Horváth, Z.; Kovács, Á.; Környei, L.; Charest, M.; Pandare, A.; Rutherford, P.; Waltz, J. Complex-Geometry 3D Computational Fluid Dynamics with Automatic Load Balancing. *Fluids* **2023**, *8*, 147. https://doi.org/10.3390/fluids8050147

Academic Editors: Federico Piscaglia, Jérôme Hélie and D. Andrew S. Rees

Received: 28 March 2023
Revised: 2 May 2023
Accepted: 4 May 2023
Published: 6 May 2023

1. Motivation and Significance

We are developing an open-source software tool, Xyst, (https://xyst.cc, accessed on 27 March 2023), for large-scale computational fluid dynamics on top of the adaptive runtime system, Charm++ (http://charmplusplus.org, accessed on 27 March 2023). Our target application areas include aerodynamics and flows around buildings with complex structures, e.g., in urban environments. Common features of these problems are flow around complex 3D geometries and large variations in length scales that must be accurately resolved. This requires computational meshes that can explicitly resolve complex-geometry features, yielding $\mathcal{O}(10^8)$–$\mathcal{O}(10^9)$ cells, demanding compute resources in $\mathcal{O}(10^4)$ CPUs as routine simulations. Complex geometries necessitate unstructured meshes; however, any unstructured algorithm requires storage of the mesh connectivity, which increases the use of indirect addressing compared to structured-grid codes, which limits performance. Many techniques have been developed in aerospace engineering to reduce indirect addressing, such as formulating the numerical method in terms of edges instead of elements [1,2] and derived data structures [3–5] to quickly access proximity information.

The above requirements drive method, algorithm, and software design decisions, such as the choice of the distributed-memory parallel computing paradigm, demanded by large problems that do not fit in the memory of a single computer. To enable fast and automatic generation of meshes around complex 3D geometries (and to simplify algorithm development), we use tetrahedra-only meshes and adopted an edge-based finite element formulation. Practicality also requires good scalability to thousands of CPUs and the effective use of large computing clusters. Historically, the desire to adequately resolve large variations in physical scales is frequently made more efficient by solution-adaptive mesh refinement, commonly used to increase mesh resolution only where and when necessary. However, mesh refinement in a distributed-memory environment is only practical with

an effective parallel load balancing strategy. In addition, there may exist many other algorithm features, desired for specific applications (coupled to fluid flow), which result in a priori unknown distributions of computational costs across a single parallel computation. Examples are the complex equation of state evaluations, combustion, radiation, atmospheric cloud physics, and particle clustering.

To fulfill the above requirements and to prepare for applications with unknown parallel load imbalances, instead of the message passing interface (MPI), we have built Xyst on top of Charm++ [6]. Charm++ is an open source runtime system built on the fundamental attributes of (1) overdecomposition, (2) asynchronous message-driven execution, and (3) migratability. Charm++ enables asynchronous communication and task-parallel execution and provides automatic load balancing based on real-time CPU load measurement. Charm++ is a general parallel programming framework in C++ supported by an adaptive runtime system and an ecosystem of libraries. It enables users to expose and express parallelism in their algorithms while automating many of the requirements for high performance and scalability. It permits writing parallel programs in units that are natural to the domain without having to deal with processors and threads, increasing productivity. Xyst is built from the ground up on top of Charm++ to enable exploiting all features of the runtime system, such as automatic overlap of computation and communication, automatic checkpointing and fault tolerance, energy-consumption-aware computation, and automatic load balancing, which is independent of the source of the load imbalance, e.g., due to CPU frequency scaling or mesh refinement.

The purpose of this paper is to provide a high-level discussion of the method and to briefly demonstrate Charm++'s capabilities in conjunction with a flow solver: we discuss basic verification and validation of the algorithm implemented and demonstrate the effectiveness of automatic load balancing via an example problem where we induce heterogeneous parallel load imbalance. Since the implemented method is not new, we do not discuss it in much detail. For details on the numerical method, the reader is referred to [1,2,5,7]. There are two novel contributions of this paper:

1. The software implementation in Xyst enables the exploitation of the advanced features of the Charm++ runtime system with a fluid solver;
2. The implementation is public and open source.

To our knowledge, no open-source software exist with the above features. Details on the advanced features of the Charm++ runtime system and their specific use in Xyst is out of scope here and will be discussed elsewhere. Our specific goal here is to communicate the message that a documented, tested, open-source, edge-based finite element solver, well-suited for the simulation of high-speed compressible flows targeting large problems at large scales, in complex 3D geometries is publicly available. Our hope is to provide an example open-source partial differential equations solver for complex-geometry problems using Charm++ that can then be taken in various directions of computational physics.

2. Software Description

The algorithm in Xyst is an edge-based finite element method for unstructured tetrahedra meshes. The method and its implementation are well-suited for high-speed compressible flows [5] in complex engineering calculations targeting the largest supercomputing clusters available. The method implemented extends that of developed by Waltz [7,8], to use the asynchronous-by-default distributed-memory task-parallel programming paradigm, enabled by the Charm++ runtime system. Below, we give the equations solved and a brief overview of the numerical method. For more details, see [7].

2.1. The Equations of Compressible Flow

The equations solved are the 3D unsteady Euler equations, governing inviscid compressible flow:

$$\frac{\partial U}{\partial t} + \frac{\partial F_j}{\partial x_j} = S, \qquad U = \begin{Bmatrix} \rho \\ \rho u_i \\ \rho E \end{Bmatrix}, \quad F_j = \begin{Bmatrix} \rho u_j \\ \rho u_i u_j + p\delta_{ij} \\ u_j(\rho E + p) \end{Bmatrix}, \quad S = \begin{Bmatrix} S_\rho \\ S_{u,i} \\ S_E \end{Bmatrix}, \tag{1}$$

where the summation convention on repeated indices has been applied, and ρ is the density, u_i is the velocity vector, $E = u_i u_i/2 + e$ is the specific total energy, e is the specific internal energy, and p is the pressure. S_ρ, $S_{u,i}$, and S_E are source terms that arise from the application of the method of manufactured solutions, used for verification; these source terms are zero when computing practical problems. The system is closed with the ideal gas law equation of state $p = \rho e(\gamma - 1)$, where γ is the ratio of specific heats. Though the ideal gas law is used here, any arbitrary (i.e., analytic or tabulated) equation of state can be used to relate density and internal energy to pressure.

2.2. The Numerical Method

The system (1) is solved using an edge-based finite-element method for linear tetrahedra [1,2,5,7,8]. The unknown solution quantities are the conserved variables U, located at the nodes of the mesh. The solution at any point within the computational domain is represented as:

$$U(\vec{x}) = \sum_{v \in \Omega_h} N^v(\vec{x}) U^v \tag{2}$$

where Ω_h is the tetrahedron containing the point $\vec{x}$, U^v is the solution at vertex v, and $N^v(\vec{x})$ is a linear basis function associated with vertex v and element Ω_h.

Applying a continuous Galerkin lumped-mass approximation, see, e.g., [5], to Equation (1) yields the following semidiscrete form of the governing equations for a vertex v:

$$\frac{\mathrm{d}U^v}{\mathrm{d}t} = -\frac{1}{V^v} \sum_j \left[\sum_{vw \in v} D_j^{vw} F_j^{vw} + \sum_{vw \in v} B_j^{vw} \left(F_j^v + F_j^w \right) + B_j^v F_j^v \right] \tag{3}$$

where vw represent all edges connected to v, V^v is the volume surrounding v, F_j^{vw} is the numerical flux between v and w, and F^v is a boundary flux. The flux coefficients on the right of Equation (3) are defined as:

$$D_j^{vw} = \frac{1}{2} \sum_{\Omega_h \in vw} \int_{\Omega_h} \left(N^v \frac{\partial N^w}{\partial x_j} - N^w \frac{\partial N^v}{\partial x_j} \right) \mathrm{d}\Omega, \tag{4}$$

$$B_j^{vw} = \frac{1}{2} \sum_{\Gamma_h \in vw} \int_{\Gamma_h} N^v N^w n_j \, \mathrm{d}\Gamma, \qquad B_j^v = \sum_{\Gamma_h \in v} \int_{\Gamma_h} N^v N^v n_j \, \mathrm{d}\Gamma \tag{5}$$

where D_j^{vw} is the area-weighted normal of the dual face separating v and w, $\Omega_h \in vw$ represents all elements containing edge vw, and Γ_h denotes a boundary surface (triangle) element. For a detailed derivation of Equation (3), see [7]. The first integral in Equation (4) is evaluated for all edges vw within the domain, while the other two in Equation (5) apply to boundary edges vw and boundary vertices v. This scheme is guaranteed to be locally conservative, as D_j^{vw} is anti-symmetric. The numerical flux F_j^{vw} is evaluated at the midpoint between v and w. Note that the above formulation can be viewed as either an edge-based finite element method on tetrahedra elements or a node-centered finite volume method on the dual mesh of arbitrary polyhedra. In Equation (3), the fluxes are computed using the Rusanov flux, which approximates the flux between v and w as [9]

$$D_j^{vw} F_j^{vw} = D_j^{vw} \left(F_j^v + F_j^w \right) + \left| D_j^{vw} \right| \lambda^{vw} (U^w - U^v) \tag{6}$$

where $\lambda^{vw} = \max(\lambda_v, \lambda_w)$ is the maximum wave speed, defined as:

$$\lambda_v = \left| u_j \frac{D_j^{vw}}{\left| D_j^{vw} \right|} \right| + c^v \tag{7}$$

where c^v is the speed of sound at vertex v. Up to this point, the method is numerically stable and first-order accurate [5]. To achieve higher-order accuracy, a piecewise limited reconstruction of the primitive variables [10] is employed on each edge vw as in a 1D finite volume method:

$$\hat{u}^v = u^v + \frac{1}{4}\left[(1-k)\phi(r^v)\delta_1 + (1+k)\phi\left(\frac{1}{r^v}\right)\delta_2\right] \tag{8}$$

$$\hat{u}^w = u^w - \frac{1}{4}\left[(1-k)\phi(r^w)\delta_3 + (1+k)\phi\left(\frac{1}{r^w}\right)\delta_2\right] \tag{9}$$

with

$$\delta_1 = 2x_i^{vw}\frac{\partial u^v}{\partial x_i} - \delta_2, \qquad \delta_2 = u^w - u^v, \qquad \delta_3 = 2x_i^{vw}\frac{\partial u^w}{\partial x_i} - \delta_2 \tag{10}$$

where $x_i^{vw} = x_i^w - x_i^v$, $r^v = \delta_2/\delta_1$, and $r^w = \delta_2/\delta_3$. This scheme corresponds to a piecewise linear reconstruction for $k = -1$ and a piecewise parabolic reconstruction for $k = 1/3$. The function ϕ can be any total variation diminishing limiter function, and we use that of van Leer [10]. The solution is advanced using a multi-stage explicit scheme of the form:

$$U^{v,k} = U^{v,0} - \alpha^k \Delta t \cdot \left.\frac{\mathrm{d}U^v}{\mathrm{d}t}\right|^{k-1} \tag{11}$$

where k is the current stage, m is the total number of stages, Δt is the time step size, and $\alpha^k = 1/(1+m-k)$. Explicit Euler time stepping is obtained for $m = 1$ and the classical second-order Runge–Kutta method with $m = 2$. We use $m = 3$, which is third-order accurate for smooth problems and linear advection but not necessarily for the nonlinear Euler equations; nevertheless, it reduces the absolute error compared to $m = 2$. The time step size is adaptive and obtained by finding the minimum value for all mesh points at every time step as:

$$\Delta t = \mathrm{C} \min_{\forall v} \frac{(V^v)^{1/3}}{\lambda^v} \tag{12}$$

where $\mathrm{C} \leq 1$ is the Courant number, and V^v is the nodal volume of the dual-mesh associated with point v.

3. Illustrative Examples

This section presents three examples of the method and software capabilities via verification, validation, and automatic load balancing.

3.1. Verification

We verify the accuracy of the numerical method and the correctness of its implementation by computing a problem whose analytical solution is available. The method is second-order accurate for smooth solutions and first-order accurate for discontinuous problems. Of the several solutions derived by Waltz and co-workers [11], we compute the vortical flow case, which is useful to test velocity errors generated by spatial operators in the presence of vorticity. The source term S and the exact solution for each flow variable are given in [11], repeated here for completeness:

$$\begin{aligned} \rho &= 1 \\ u_i(x_i) &= \begin{pmatrix} \alpha x_1 - \beta x_2 \\ \beta x_1 + \alpha x_2 \\ -2\alpha x_3 \end{pmatrix} \\ p(x_3) &= p_0 - 2\rho\alpha^2 x_3^2 \\ e &= p(x_3)\rho(\gamma - 1), \end{aligned} \tag{13}$$

and the source terms to be used in Equation (1) are defined as:

$$\begin{aligned} S_\rho &= 0 \\ S_{u,i} &= \begin{pmatrix} \rho\left(\alpha^2 - \beta^2\right)x_1 - 2\rho\alpha\beta x_2 \\ \rho\left(\alpha^2 - \beta^2\right)x_2 + 2\rho\alpha\beta x_1 \\ 0 \end{pmatrix} \\ S_E &= u_i S_{u,i} + \frac{8\rho\alpha^3 x_3^2}{\gamma - 1}. \end{aligned} \tag{14}$$

The simulation domain is a cube centered around the point $x_i = (0, 0, 0)$. The initial conditions are sampled from the analytic solution. We set Dirichlet boundary conditions on the sides of the cube, sampling the analytic solution. The numerical solution does not depend on time and approaches steady state due to the source term, which ensures equilibrium in time. As the numerical solution approaches a stationary state, the numerical errors in the flow variables converge to stationary values, determined by the combination of spatial and temporal errors, which are measured and assessed. We computed the solution with $\alpha = \beta = 1$ on three different meshes, listed in Table 1. Time integration was carried out in the range $t = 0 \ldots 1$ with a fixed time step of $\Delta t = 0.002$ for the coarsest mesh, which was successively halved for the finer ones.

Table 1. Meshes for the vortical flow case, where h is the average edge length.

Mesh	Points	Tetrahedra	h
0	132,651	750,000	0.02
1	1,030,301	6,000,000	0.01
2	8,120,601	48,000,000	0.005

Figure 1 shows the initial and final velocity fields on center planes through the origin, which confirms the steady-state nature of the problem. Shown also are the steady state pressure and energy fields. Table 2 lists and Figure 2 depicts the L_1 errors in the computed flow variables along with the measured convergence rates. The L_1 error in each field variable is computed as:

$$L_1 = ||\varepsilon||_1 = \frac{\sum_{v=1}^{n} V^v \left|\hat{U}^v - U^v\right|}{\sum_{v=1}^{n} V^v} \tag{15}$$

where n is the number of points and $\hat{U}^v$ and U^v are the computed and exact solutions at mesh point v. The convergence rate $\mathbf{p}$ is then calculated as:

$$\mathbf{p} = \frac{\log||\varepsilon||_1^{m+1} - \log||\varepsilon||_1^{m}}{\log h^{m+1} - \log h^{m}} \tag{16}$$

where m is a mesh index. The measured convergence rate in all variables approaches 2.0, as expected.

Table 2. L_1 errors and convergence rates for the vortical flow problem. The convergence rates in the last line are computed from Equation (16) based on the errors from the 48 M and 6 M meshes.

Mesh	$L_1(\rho)$	$L_1(u_1)$	$L_1(u_2)$	$L_1(u_3)$	$L_1(e)$
750 K	6.07×10^{-5}	6.68×10^{-5}	4.16×10^{-5}	7.98×10^{-5}	1.24×10^{-3}
6 M	1.85×10^{-5}	1.63×10^{-5}	9.71×10^{-6}	2.00×10^{-5}	3.57×10^{-4}
48 M	5.08×10^{-6}	3.93×10^{-6}	2.25×10^{-6}	4.89×10^{-6}	9.84×10^{-5}
p	1.86	2.05	2.11	2.03	1.86

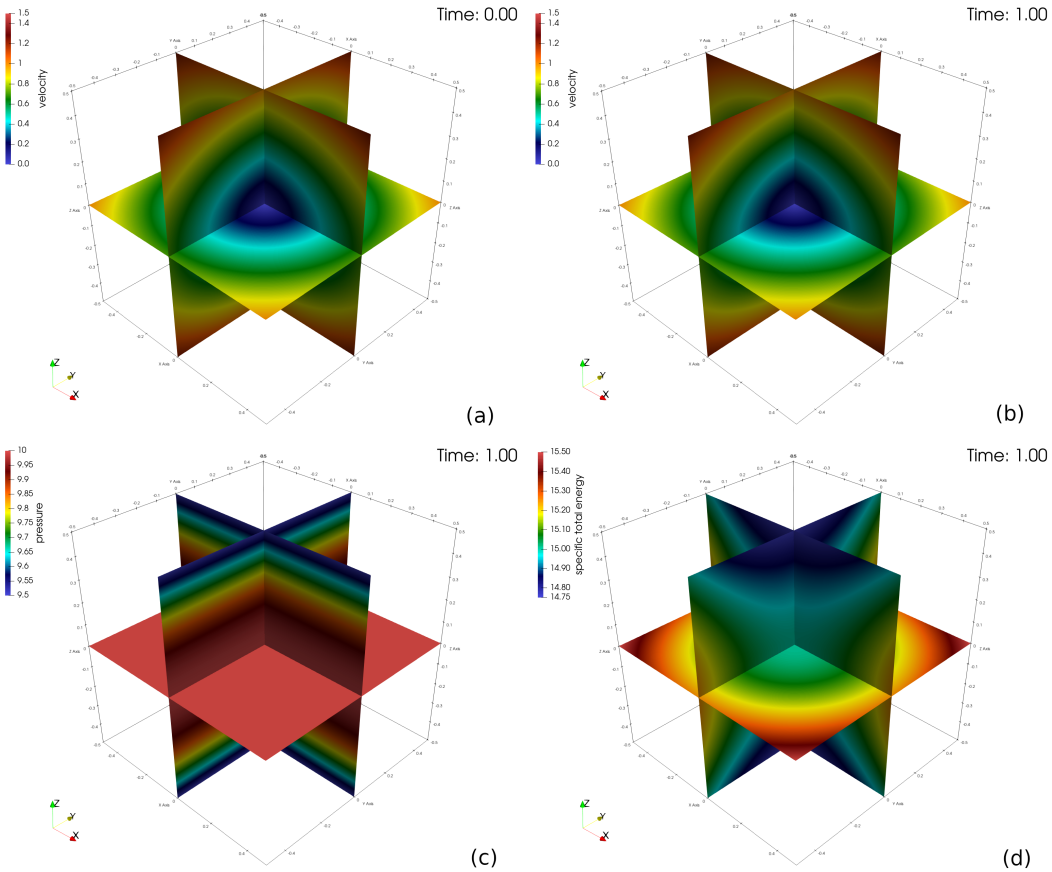

Figure 1. Initial and final velocity (**a**,**b**), pressure (**c**), and total energy (**d**) for the vortical flow problem.

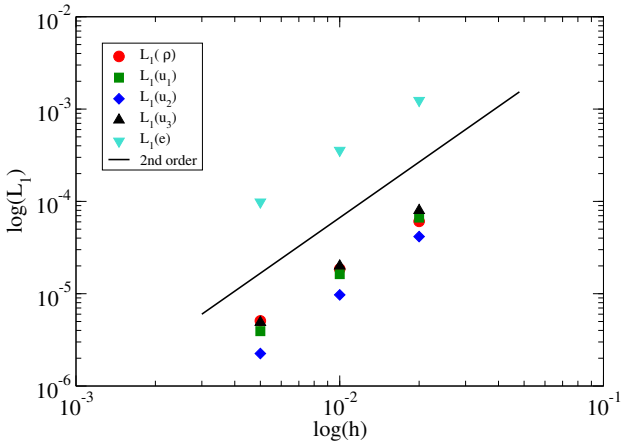

Figure 2. L_1 errors of the density, velocity components, and internal energy in the vortical flow problem.

3.2. Validation

To demonstrate the solver in a 3D complex-geometry case, we computed the well-documented aerodynamic flow over the ONERA M6 wing, see, e.g., [12]. The available experimental data of the pressure distribution along the wing allow validation of the numerical method. The initial conditions prescribed a Mach number of 0.84 and the velocity with an angle of attack of 3.06°. At the outer surface of the domain, characteristic far-field boundary conditions are applied, while along the wing surface, symmetry (free-slip) conditions are set.

Since we know that the solution of this problem does not depend on time, the most efficient ways to compute it are to use implicit time marching or local time stepping [5], of which we implemented the latter, allowing to take larger time steps for larger computational cells. Local time stepping is implemented by defining a separate timestep for each gridpoint v:

$$\Delta t^v = \mathrm{C}\frac{(V^v)^{1/3}}{\lambda^v} \tag{17}$$

and advancing the solution in every point using its own time step size Δt^v. The numerical solution is marched until a steady solution reached. The stopping criterion is determined by a sufficiently small value of the 2–norm of the density residual as

$$||\Delta\rho||_2 = ||\rho^{n+1} - \rho^n||_2 < 10^{-7} \tag{18}$$

The computational mesh consists 710,971 tetrahedra and 131,068 points. The surface mesh is shown in Figure 3 together with the converged computed pressure contours on the upper and lower surfaces of the wing. Figure 4 shows how the density residual converges to steady state. The wall-clock time for this computation was about 2 min using 32 CPUs. The distribution of the computed pressure coefficient, $C_p = (p - p_\infty)/(\rho_\infty u_\infty^2/2)$, is compared to experimental data [13] at various semispans in Figure 5. Here, the ∞ subscript denotes the farfield conditions and u is the length of the velocity vector.

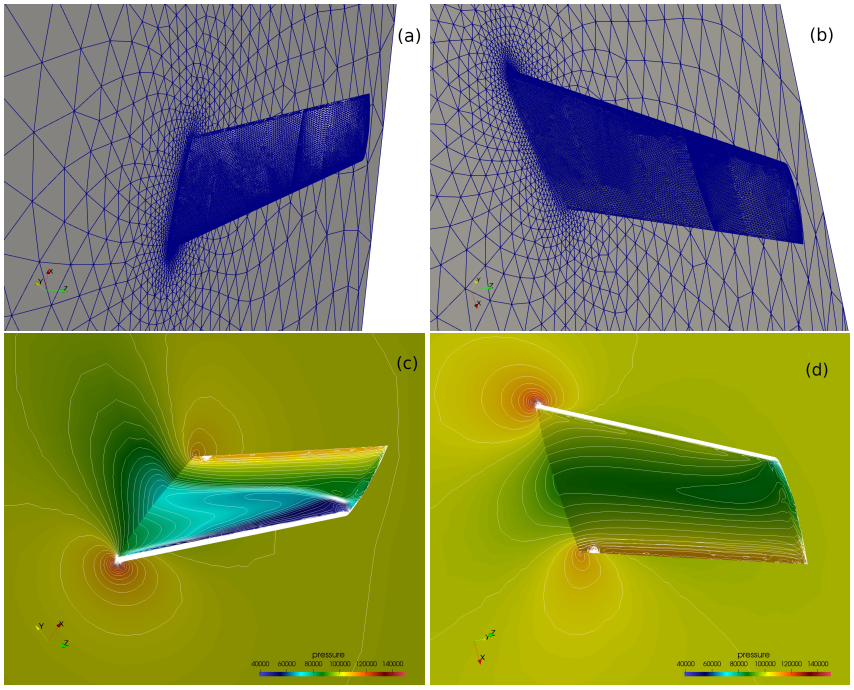

Figure 3. Upper (**a**) and lower (**b**) sides of the surface mesh used for the ONERA M6 wing calculation. Computed pressure contours on the upper (**c**) and lower (**d**) surface.

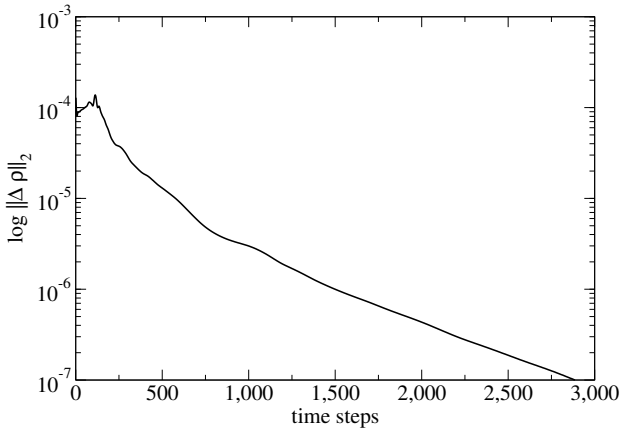

Figure 4. Convergence history of the ONERA wing computation.

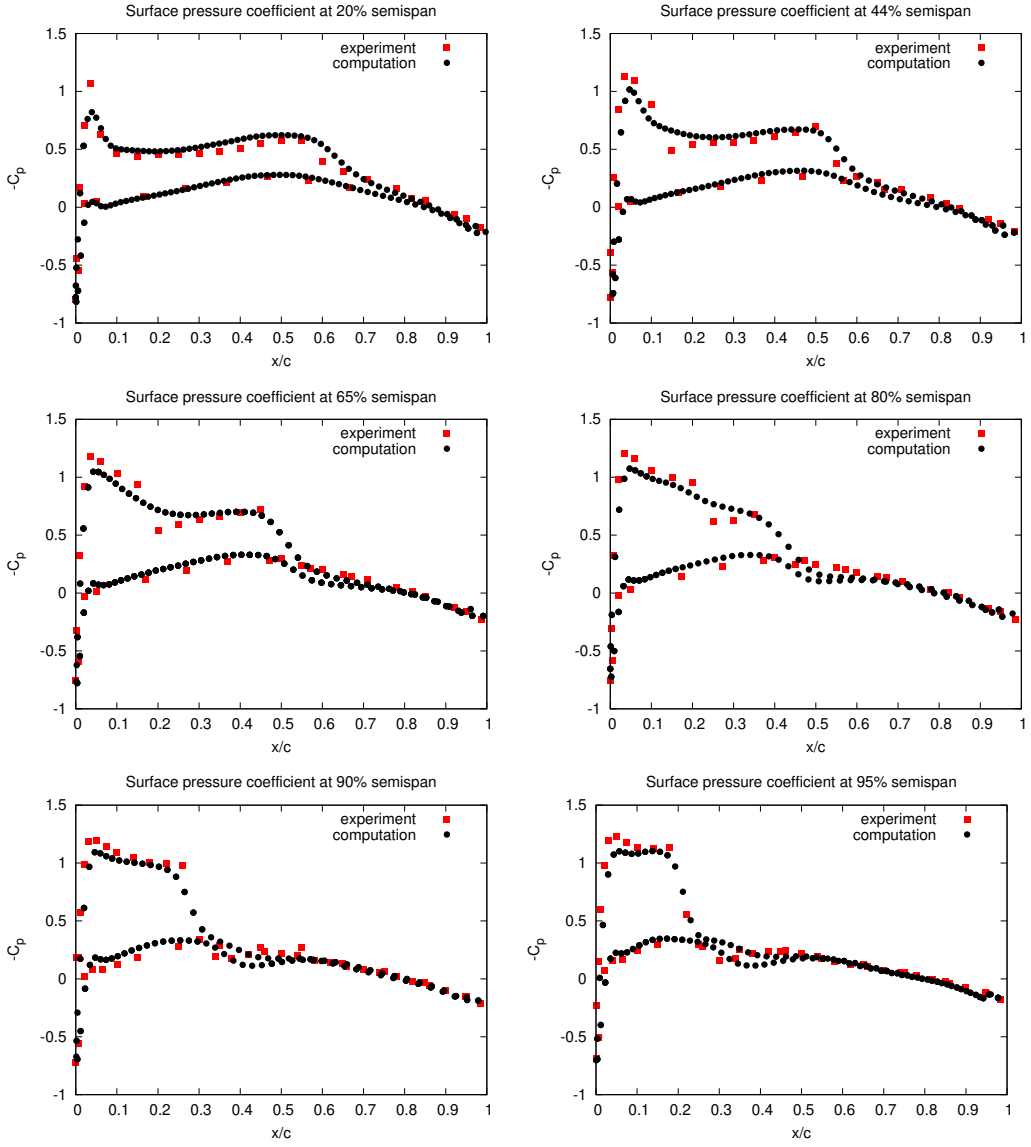

Figure 5. Computed and experimental surface pressure coefficient at different semispans of the ONERA wing. Here, c denotes the length in $x = x_1$ of the wing cross section at the semispan location.

3.3. Automatic Load Balancing

With respect to solving partial differential equations on meshes, there is no magic using Charm++ compared to MPI: the computational domain is still decomposed and communication of partition-boundary data is still explicitly coded and communicated. However, an important difference compared to the usual divide-and-conquer strategy of domain decomposition is that Charm++ also allows combining this data-parallelism with task parallelism. As a result, various tasks (e.g., computation and communication) are performed independently, in arbitrary order, and are overlapped due to Charm++'s asynchronous-by-default paradigm. For an example of how task-parallelism can be specified in Charm++ with a different Euler solver, see [14].

Another unique feature of Charm++ compared to MPI is built-in automatic load-balancing. Charm++ can perform real-time CPU load measurement and, if necessary, can migrate data to under-loaded processors to homogenize computational load. This can be beneficial independent of its origin, i.e., adaptive mesh refinement, complex local equations of state, and CPU frequency scaling, or simply if some work units have a different number of boundary conditions to apply compared to others. Multiple load-balancing strategies are available in Charm++ and using them requires no extra programming effort: the feature is turned on by a command-line switch. Load balancing costs are negligible (compared to the physics operators) and can be beneficial for irregular work loads at any problem size, from laptop [15] to cluster [14].

We computed the Sedov problem [16], widely used in shock hydrodynamics, to test the ability of numerical methods to maintain spherical symmetry. In this problem, a source of energy is defined to produce a shock in a single computational cell at the origin at $t = 0$ as:

$$u_i(x_i) = 0 \tag{19}$$

$$\rho(x_i) = 1 \tag{20}$$

$$e(x_i) = \begin{cases} 1.0 \times 10^{-4} & x_i \neq 0 \\ 2.78 \times 10^{6} & x_i = 0 \end{cases} \tag{21}$$

$$p(x_i) = \begin{cases} 0.67 \times 10^{-4} & x_i \neq 0 \\ 1.85 \times 10^{6} & x_i = 0 \end{cases} \tag{22}$$

In this smulation we use $\gamma = 5/3$. These initial conditions correspond to pressure ratios of $\mathcal{O}(10^8)$–$\mathcal{O}(10^{10})$ across a single element at the origin and are specified so that the source of energy produces a shock location of $x = 1$ at $t = 1$. Thus, the 1D solution in time is a spherically spreading wave starting from a single point. We used a 3D computational domain that is an octant of a sphere, with the mesh consisting of 23,191,232 tetrahedra and 3,956,135 points. The computed density is compared to the semi-analytic solution at $t = 1$ in Figure 6.

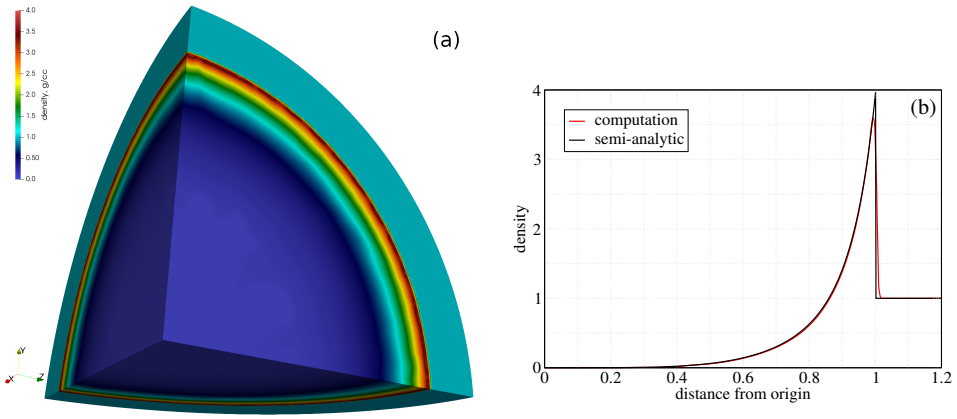

Figure 6. Surface density (**a**) and its radial distribution (**b**) at $t = 1$ from the Sedov calculation.

To exercise Charm++'s built-in load balancing, we modified the Sedov problem by adding extra computational load to the function that computes the pressure based on density and internal energy: `if fluid density > 2.0 then sleep (1 ms)`. This increases the cost of the equation of state evaluation, whose location propagates in space and time, which induces load imbalance across multiple mesh partitions in parallel. Figure 7 and Table 3 show the effect of load balancing on wall-clock time: in this particular case, the extra load would make the simulation about 36× more expensive. This is made 6.1× faster using load balancing.

We emphasize that we wrote no load-balancing code: we simply ensure overdecomposition (in this case, we used 6399 mesh partitions on 32 CPUs) and turn on load balancing; the runtime system measures real-time CPU load and automatically performs object migration to homogenize the load. While this example was run on a single computer, experience shows that the benefits of such load balancing can be even greater on large distributed-memory machines [14].

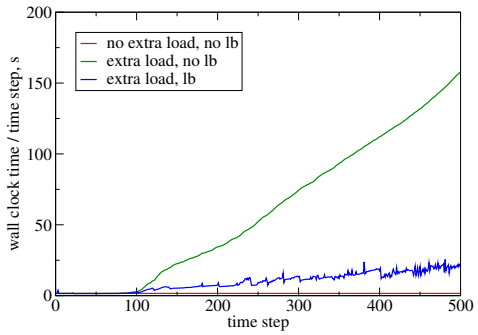

Figure 7. Measured wall-clock time of each time step during the Sedov calculation without extra load, as well as with extra load with and without load balancing. The area below each curve is proportional to the total computational cost.

Table 3. Timings for the Sedov problem with and without load balancing.

Case	Extra Load	Total Time, s	Speed-Up
0	no	825	-
1	yes	30,276	-
2	yes	4958	6.11×

4. Impact and Conclusions

We presented an open-source code, Xyst (https://xyst.cc, accessed on 27 March 2023), for the simulation of compressible flows. The software implementation with documentation, testing, code quality assessment, and reproducible examples is made publicly available. Xyst enables the use of advanced runtime system features not directly available for other flow solvers using MPI, which would often require advanced computer science expertise and significant ongoing nontrivial software maintenance effort in production. Xyst can be used to solve complex-geometry engineering problems involving 3D compressible flows at large scales. Building on the publicly available source code can also be used as a basis of multiphysics simulations.

Using Charm++ (http://charmplusplus.org, accessed on 27 March 2023) as its parallel programming paradigm, the implementation enables combining data-, and task-parallelism in an asynchronous-by-default execution model. Using a single software abstraction on both shared-, and distributed-memory machines enables performance and scalability. Automatic load balancing, built into Charm++, enables redistribution of heterogeneous computational load based on real-time CPU load measurement. The runtime system also features automatic checkpointing, fault tolerance, resilience against hardware failure, and supports power- and energy-aware computation without extra effort required from the application programmer.

Author Contributions: Conceptualization, A.P. and J.W.; Methodology, J.B., M.C. (Marc Charest), A.P. and J.W.; Software, J.B., M.C. (Marc Charest) and A.P.; Validation, J.B., M.C. (Marc Charest), P.R. and J.W.; Investigation, J.B.; Resources, M.C. (Mátyás Constans), Á.K., L.K., P.R. and J.W.; Data curation, J.W.; Project administration, J.B.; Funding acquisition, Z.H. All authors have read and agreed to the published version of the manuscript.

Funding: Co-funded by the European Union, this work has received funding from the European High Performance Computing Joint Undertaking (JU) and Poland, Germany, Spain, Hungary, and France under grant agreement number: 101093457. This work was also partially supported by the U.S. Department of Energy through the Los Alamos National Laboratory. Los Alamos National Laboratory is operated by Triad National Security, LLC, for the National Nuclear Security Administration of the U.S. Department of Energy (Contract No. 89233218CNA000001).

Data Availability Statement: The source code and data presented in this study are openly available at https://xyst.cc under "Examples".

Acknowledgments: We thank Hong Luo and Weizhao Li of North Carolina State University for the fruitful discussions regarding local time stepping, the ONERA wing meshes and the experimental data.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Luo, H.; Baum, J.D.; Löhner, R.; Cabello, J. Adaptive Edge-Based Finite Element Schemes for the Euler and Navier-Stokes Equations. In Proceedings of the 31st Aerospace Sciences Meeting, Reno, NV, USA, 1–14 January 1993.
2. Luo, H.; Baum, J.D.; Löhner, R. Edge-based finite element scheme for the Euler equations. *AIAA J.* **1994**, *32*, 1183–1190. [CrossRef]
3. Waltz, J. Derived data structure algorithms for unstructured finite element meshes. *Int. J. Numer. Methods Eng.* **2002**, *54*, 945–963. [CrossRef]
4. Löhner, R.; Galle, M. Minimization of indirect addressing for edge-based field solvers. *Commun. Numer. Methods Eng.* **2002**, *18*, 335–343.
5. Löhner, R. *Applied Computational Fluid Dynamics Techniques: An Introduction Based on Finite Element Methods*; Wiley and Sons Ltd.: Chichester, UK, 2008.
6. Acun, B.; Gupta, A.; Jain, N.; Langer, A.; Menon, H.; Mikida, E.; Ni, X.; Robson, M.; Sun, Y.; Totoni, E.; et al. Parallel Programming with Migratable Objects: Charm++ in Practice. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14), New Orleans, LA, USA, 16–21 November 2014; pp. 647–658. [CrossRef]
7. Waltz, J.; Morgan, N.; Canfield, T.; Charest, M.; Risinger, L.; Wohlbier, J. A three-dimensional finite element arbitrary Lagrangian-Eulerian method for shock hydrodynamics on unstructured grids. *Comput. Fluids* **2013**, *92*, 172–187. [CrossRef]
8. Waltz, J. Microfluidics simulation using adaptive unstructured grids. *Int. J. Numer. Methods Fluids* **2004**, *46*, 939–960. [CrossRef]
9. Davis, S.F. Simplified second-order Godunov-type methods. *SIAM J. Sci. Stat. Comput.* **1988**, *9*, 445–473. [CrossRef]
10. Hirsch, C. *Numerical Computation of Internal and External Flows: The Fundamentals of Computational Fluid Dynamics*; Wiley and Sons: Chichester, UK; New York, NY, USA; Brisbane, Australia; Toronto, ON, Canada; Singapore, 2007.
11. Waltz, J.; Canfield, T.; Morgan, N.; Risinger, L.; Wohlbier, J. Manufactured solutions for the three-dimensional Euler equations with relevance to Inertial Confinement Fusion. *J. Comput. Phys.* **2014**, *267*, 196–209. [CrossRef]
12. Luo, H.; Baum, J.; Löhner, R. A Fast, Matrix-Free Implicit Method for Compressible Flows on Unstructured Grids. *J. Comput. Phys.* **1998**, *146*, 664–690. [CrossRef]
13. Schmitt, V.; Charpin, F. *Pressure Distributions on the ONERA-M6-Wing at Transonic Mach Numbers, Experimental Data Base for Computer Program Assessment*; Report of the Fluid Dynamics Panel Working Group 04; Technical rep4ort, AGARD AR-138; 1979. Available online: https://www.sto.nato.int/publications/AGARD/AGARD-AR-138/AGARD-AR-138.pdf (accessed on 27 March 2023).
14. Bakosi, J.; Bird, R.; Gonzalez, F.; Junghans, C.; Li, W.; Luo, H.; Pandare, A.; Waltz, J. Asynchronous distributed-memory task-parallel algorithm for compressible flows on unstructured 3D Eulerian grids. *Adv. Eng. Softw.* **2021**, *160*, 102962. [CrossRef]
15. Li, W.; Luo, H.; Bakosi, J. A *p*-adaptive Discontinuous Galerkin Method for Compressible Flows using Charm++. In Proceedings of the AIAA Scitech 2020 Forum, Orlando, FL, USA, 6–10 January 2020.
16. Sedov, L. *Similarity and Dimensional Methods in Mechanics*, 10th ed.; Elsevier: Amsterdam, The Netherlands, 1993. [CrossRef]