Aquila-LCS: GPU/CPU-Accelerated Particle Advection Schemes for Large-Scale Simulations

Christian Lagares[a,b], Guillermo Araya[b]

[a] *Dept. of Mechanical Eng., University of Puerto Rico at Mayaguez, PO Box 9000, PR 00681, USA, christian.lagares@upr.edu*

[b] *Computational Turbulence and Visualization Lab, Dept. of Mechanical Eng., University of Texas at San Antonio, One UTSA Circle, TX 78249, USA, araya@mailaps.org*

Abstract

We introduce Aquila-LCS, GPU and CPU optimized object-oriented, in-house codes for volumetric particle advection and 3D Finite-Time Lyapunov Exponent (FTLE) and Finite-Size Lyapunov Exponent (FSLE) computations. The purpose is to analyze 3D Lagrangian Coherent Structures (LCS) in large Direct Numerical Simulation (DNS) data. Our technique uses advanced search strategies for quick cell identification and efficient storage techniques. This solver scales effectively on both GPUs (up to 62 Nvidia V100 GPUs) and multi-core CPUs (up to 32,768 CPU cores), tracking up to 8-billion particles. We apply our approach to four turbulent boundary layers at different flow regimes and Reynolds numbers.

Keywords: LCS, GPU-Accelerated, DNS, Distributed Memory Algorithms, FTLE, FSLE

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=4745205

Metadata

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v1.0
C2	Current code version	https://github.com/ChristianLagares/Aquila-LCS
C3	Permanent link to Reproducible Capsule	None
C4	Legal Code License	Modified BSD License with Non-Commercial Use Clause
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python
C7	Compilation requirements, operating environments & dependencies	Operating Systems: Linux, macOS
C8	If available Link to developer documentation/manual	https://github.com/ChristianLagares/Aquila-LCS
C9	Support email for questions	GitHub Issues, araya@mailaps.org

1. Motivation and significance

The quest to find order in seemingly chaotic velocity domains of fluid dynamics has long captivated a diverse group of researchers and professionals. This very essence defines the realm of turbulence research. While high-speed turbulence is vital in numerous applications, spanning both civilian and defense sectors, it introduces specific challenges across experimental and numerical domains. Yet, with advancing computational power, executing high-fidelity simulations within complex domains is now possible. When we have access to top-tier data, the next challenge is leveraging the right tools to distill wisdom from these massive datasets, which hinge on the simulation methods chosen for fluid flow analysis.

Software tools are undeniably pivotal in this domain. When diving into Computational Fluid Dynamics (CFD), we can broadly classify it into three primary segments: Reynolds-averaged Navier-Stokes (RANS), Large-Eddy Simulations (LES), and Direct Numerical Simulation (DNS). Of these, DNS stands out as it does not necessitate turbulence models, although one might still incorporate models such as fluid type or molecular viscosity models in compressible flows. With DNS data at hand, one can explore the coherent structure dynamics either through an Eulerian or a Lagrangian lens, each offering distinct and valuable perspectives.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=4745205

Eulerian techniques present computational efficiencies, largely due to their predictable memory access and consistent patterns, as underscored by [1] and [2]. In contrast, Lagrangian methodologies demand a profound comprehension of hardware specifics to attain satisfactory performance metrics. Crucially, optimal memory buffer utilization is paramount, as is the prevention of codebase fragmentation even as one harnesses algorithmic strengths. Recognizing the objectivity and widespread applicability of LCS ([3, 4, 5], a plethora of implementation strategies can be found in scholarly articles ([6, 7]).

A significant portion of these strategies is inclined toward 2D LCS [8, 7]. As an illustration, [9] unveiled a tool within the widely-adopted Matlab framework, aptly christened "LCS Tool." This tool has garnered acknowledgment in various studies as a ready-to-use solution, as exemplified by [10]. Being rooted in Matlab, LCS Tool's capabilities are confined to singular node operations, heavily leaning on Matlab's inherent parallel functions, which come with significant memory overheads. Concurrently, alternative strategies grounded in the finite-element method emerged, including the innovative work of [11], where they leveraged a discontinuous Galerkin approach to compute the FTLE. Given the advantages of Finite Element Method (FEM) with its adaptive mesh refinement capabilities, [12] proposed a method for adeptly refining intricate meshes for LCS evaluations. [13] introduced a GPU-centric approach to compute LCS for smooth-particle hydrodynamics (SPH) datasets. Although laudable in its efficiency, it remained restricted to singular node operations and did not present an optimized CPU-based counterpart. Their findings showcased GPU implementation speedups ranging from 33× to 67×. The hardware as described in [13] suggests that an optimized CPU version could have reduced this discrepancy, pointing toward the potential for further enhancement in hybridized computational strategies.

1.1. Numerical Methods

Numerical delineation of LCS, representing the manifolds created by particle pathways in fluidic movement, can be proficiently done using techniques such as the FTLE or its alternative, the FSLE. Even though their operational frameworks vary, both techniques assess the distortion of a particulate field regarding its initial state. Specifically, FSLE measures the duration required for two particles to achieve a defined finite separation, whereas FTLE conducts integration across a consistent, finite duration, independent of the spacing of adjacent particles [14, 15]. Within the scope of this research, we evolve the 2D FTLE technique delineated in [10] into a more comprehensive 3D format, considering computational intricacies as flagged by [5]. The trajectory of a particle, originating at a time t_0 and position x_0, across a set

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=4745205

interval can be articulated through the flow map, considering the velocity domain. The FTLE can be expressed as:

$$FTLE_{\tau}(\mathbf{x},t) = \frac{1}{|\tau|}\log\left(\sqrt{\lambda_{max}(C^{t}_{t_o}(\mathbf{x}))}\right) \tag{1}$$

Here, λ_{max} symbolizes the dominant eigenvalue and $C^{t}_{t_o}(\mathbf{x})$ is the right Cauchy-Green (CG) deformation tensor at a particular spatial vector $\mathbf{x}$ after particle integration to $t = t_o+\tau$. This right CG tensor, characterizing the deformation dynamics of a continuum, is defined as $C_{i,j} = \frac{\partial x^{t}_{k}}{\partial x^{t_o}_{i}}\frac{\partial x^{t}_{k}}{\partial x^{t_o}_{j}}$. Elevated FTLE metrics spotlight potential attracting or repelling manifold points. A forward-in-time trajectory integration of the particle indicates a repelling structure. In contrast, tracing the pathway backwards in time reveals attracting boundaries [4]. This computational methodology delivers a resilient and adaptable blueprint for dissecting intricate fluid motion patterns and identifying high shear zones. Similarly, the FSLE is described as $\frac{1}{|\tau|}\log\left[\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\delta_{f,i}}{\delta_{o,i}}\right)\right]$, where $\delta_{o,i}$ is the original distance between the i-th particle and a reference particle; whereas, $\delta_{f,i}$ is the final distance between the i-th particle and a reference particle.

2. Software description

In the present manuscript, our contribution is a highly scalable software for dynamic 3D FTLE/FSLE calculations suitable for extensive DNS datasets [16] in structured meshes (nonetheless, other unsteady three-dimensional datasets can be employed such as those from LES), detailing unique specifications for both GPU and CPU environments. We emphasize the shared efficiencies and highlight performance-driven deviations. A core feature we introduce is an innovative cell locator mechanism optimized for constant-time cell searches.

2.1. Software architecture

The selected implementation language for Aquila-LCS is Python, as an object-oriented and high-level implementation language. With the purpose of achieving efficiency, good scalibility and targeting multi-core CPUs and GPUs in plain Python; additional libraries are employed. NumPy ([17]) supplies a multi-dimensional array and additional functionality aiming near-native performance by targeting pre-compiled loops at the array level. We are also utilizing Numba JIT compiler [18]. Numba enables compiling Python to machine code using LLVM and achieve speeds typically only attainable by

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=4745205

lower-level, compiled languages such as C, C++ or FORTRAN. LLVM was proposed by [19] as a high-performance compiler and front-end, presenting both CPU and GPU programming capabilities. Furthermore, Numba's parallel CPU backend also provides a higher level of abstraction over multiple threading backends. In particular, it allows for specific selection between OpenMP ([20]) and TBB ([21]). Due to dissimilarity in transient Lagrangian workloads, TBB's ability to dynamically balance workloads, while accounting for locality and affinity, enhances the code performance in our particular case. The implementation can just be changed by setting a command line argument at program launch. More importantly, in designs with exotic architectures featuring hybrid core or in certain unbalanced execution resources; TBB mitigates two limiting scaling issues in heterogeneous systems, as follows: (i) since TBB does not have a central queue thus removing that bottleneck, and (ii) TBB enables workers to steal work from the back of other workers' queue.

Both the CPU and GPU implementations are self-contained to a single script each. Whereas this limits modularity to some extent, it greatly simplifies usage and deployment since a single file is required to execute any given portion of the functionality. Both implementations (code and software metadata), documentation, DNS dataset as well as an example bash script for executing it are provided at the Metada section. The overall design can be seen in figure 1.Addressing efficiently particle advection's numerical complexities is achieved by breaking it into five core components: Data I/O, flow field interpolation, cell locator, velocity derivation, and particle dynamics. Different computational components have varied platform sensitivities. For example, I/O is network-centric, while velocity derivation demands both memory and computational capacity. The cell locator has posed challenges historically, prompting our innovative method influenced by numerical linear algebra and tree-traversal techniques. Our method, termed Queue-Less, Multi-Level, Best-First Search (QL-MLBFS), leverages coarser meshes for efficient particle location determination. Our enhanced cell search mechanism transitions from a $\mathcal{O}(N_p N_c^3)$ complexity to a scalable $\mathcal{O}(N_p \log_8(N_c))$, enabling linear scaling with particle count. Particle velocity is deduced using a trilinear interpolation based on its owning cell's natural coordinates. More details about the QL-MLBFS methodology and spatial interpolation scheme can be found in section 2.1.2 at [22]. This process varies in arithmetic intensity (0.78-3.67 FLOPs per Byte) depending on flow dynamics and particle distribution. For precision, cells are projected to a uniform coordinate system with stability maintained by velocity bounds. Our interpolation approach ensures stability even in highly chaotic flow situations.

To optimize memory use, our algorithm integrates right Cauchy-Green (CG)

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=4745205

tensor calculation with the FTLE eigenvalue phase. This design necessitates minimal memory per execution thread, accommodating numerous threads on GPUs and modern CPUs. Interestingly, a V100 GPU would need a fraction of memory compared to large datasets. Following CG tensor derivation, the algorithm computes the maximum eigenvalue of the tensor. Here, our CPU and GPU strategies differ. For the GPU version, we employ a power iteration method, tailored for symmetric positive definite matrices, ensuring convergence to a single dominant eigenvalue. On the CPU, we call an optimized LAPACK routine. For the temporal interpolation step, we have implemented a GPU technique analogous to game development's "triple buffering". This method involves pointer swapping to limit memory copies, ensuring only one read per time step overwrites the oldest buffer with the newest content. The interested reader is referred to section 2.1.3 [22] for detailed description of eigenvalue calculation. While our present methodology serves our computational constraints, we recognize the potential of advanced techniques like the QR algorithm for future exploration.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=4745205

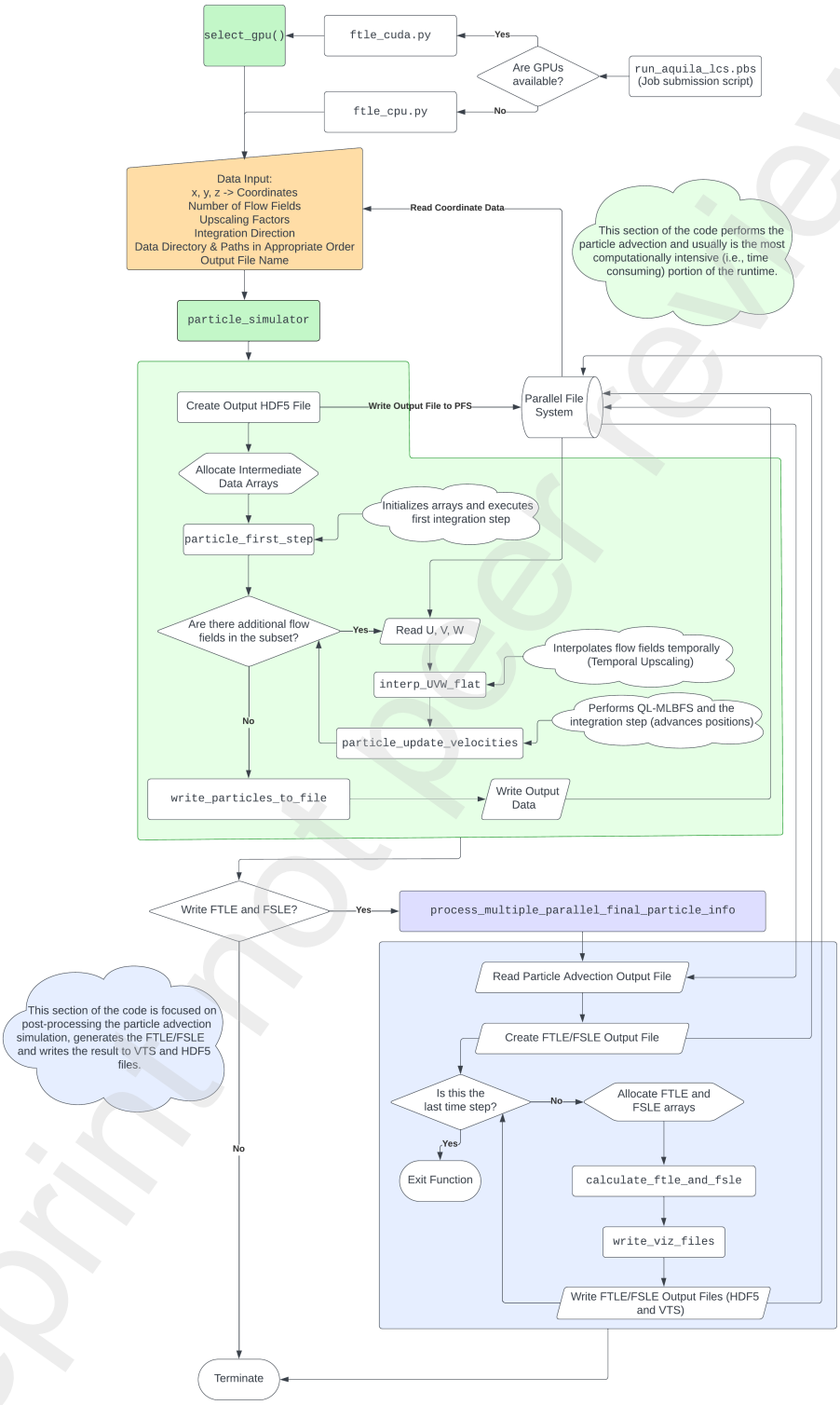

Figure 1: Aquila-LCS flow chart. Note that green segments are related to the particle advection workflow and blue segments are related to the FTLE/FSLE calculations and output file generation.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=4745205

2.2. Software functionalities

Our specialized Python software has been designed to handle large CFD data, specifically on rectilinear grids. These grids maintain a consistent node count across each dimension throughout the entire domain. One of the key features of our software is its ability to compute both FTLEs and FSLEs, which represent an effective methodology to analyze time-varying analogs of invariant manifolds (attracting and repelling) in unsteady fluid flows.
Aquila-LCS has demonstrated almost ideal linear scaling. It can manage particle advection data efficiently, accommodating billions of particles ($\approx$ 8) without sacrificing speed or accuracy. This is made possible through its optimized architecture that can leverage the parallel processing capabilities of multi-core Central Processing Units (CPUs) as well as the raw power of Nvidia Graphics Processing Units (GPUs). In terms of data handling and visualization, the software produces VTS files, which are instrumental for high-quality visualization of the computed data, aiding in the analytical process and providing visual context to complex fluid behaviors. The code also stores all processed data in HDF5 files (volumetric and temporal FTLEs/FSLEs), ensuring a streamlined storage solution that is both compact and easy to access for further analysis or sharing.

3. Illustrative examples

Onyx, a Cray XC40/50 machine, employs Intel Broadwell and Knights Landing CPUs, and Nvidia P100 GPUs. It features dual socket compute nodes with 22 cores each, simultaneous multithreading, and 128 GB of accessible RAM. Narwhal, an HPE Cray EX, can process up to 12.8 petaflops. Each node contains two AMD EPYC 7H12 CPUs with 128 cores and 256 threads, 256 GB of DDR4 memory, and some nodes include V100 GPUs. The nodes are interconnected using an HPE Slingshot 200 Gbit/s network. Anvil, a Dell EMC PowerEdge C6420-based supercomputer at Purdue University, uses second-generation Intel Xeon Scalable processors. It has 1,008 compute nodes, each with 48 cores and 192 GB of memory, and is equipped with 56 Nvidia V100 GPUs. Chameleon's A100 node, which we also tested, features 2 Intel Xeon Platinum 8380 CPUs with a total of 80 cores and 160 threads, and 4 A100 GPUs with 80 GBs of HBM 2e memory. The node also has 512 GBs of DDR4 memory and 1.92 TBs of local NVMe storage. In comparing CPUs and GPUs, it is important to consider architectural differences (see Table 2 in [22]). CPUs prioritize low latency of an individual operation thread, while GPUs favor high-throughput by processing a larger number of work items. Despite similar memory bandwidth when accounting for vector widths, modern GPUs have the advantage of zero-overhead context

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=4745205

switching between multiple threads, contributing to their resilience to high latency memory operations and often resulting in simpler, yet faster, code. Figure 2 depicts the Aquila-LCS's strong scaling performance, which was tested up to 32,768 CPU cores and up to 62 Nvidia V100 GPUs (4960 GPU SM cores or an equivalent 317,440 CUDA cores). Therefore, the total number of CPU threads evaluated was roughly 32K, whereas the peak number of GPU threads reached roughly 10M (GPU) threads (2048 threads per Volta SM core arranged in blocks of 64 threads for a total of 32 thread blocks per SM), or 310× more threads on a GPU (GPU threads are lightweight threads compared to the heavier OS-managed threads on CPUs). The horizontal axis represents nodes in the computing system. It is observed an almost "ideal linear scaling" up to 30 nodes in some systems. Additionally, for larger numbers of nodes some inefficiencies in the software stack or hardware resources below the application can deteriorate the code performance with deviation from the ideal scaling, which is very obvious in Onyx (Broadwell processors) and Narwhal (Rome processors). Here, integration time is the time of the particle advection loop only; whereas, the system time is the total running time, including MPI overhead, Python runtime and writing files. In addition, in figure 3 the number of nodes was fixed (16 compute nodes), while increasing the number of particles (from 50M to over 8B particles). The most relevant aspect to describe is the dominance of network latency in large-scale parallel file systems (PFS): note how a powerful GPU is essentially idle when waiting for data from the PFS, particularly for both the P100 and V100 results up to 1-2 $\times 10^8$ particles, with a following linear trend. Readers are referred to pages 16-20 in [22] for an in-depth scaling performance discussion and computational architecture assessment.

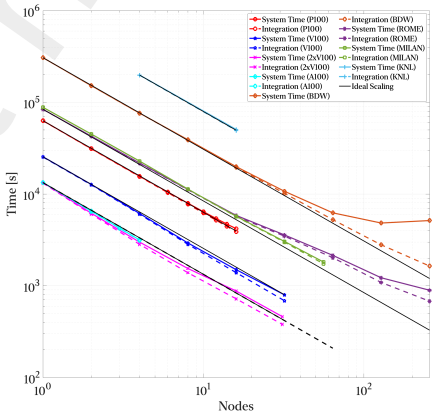

Figure 2: Strong scaling evaluation for the proposed particle advection scheme.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=4745205

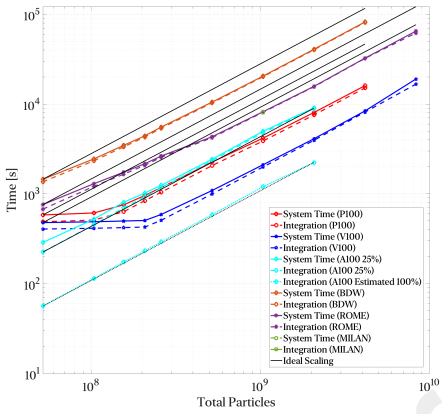

Figure 3: Particle scaling evaluation for the proposed particle advection scheme.

Aquila-LCS has been recently tested and validated (by varying the number of advecting particles and temporal frequency sampling) in [22] via four DNS scenarios over spatially-developing turbulent boundary layers (SDTBL) conducted on adiabatic/isothermal flat surfaces across several flow regimes [23, 24, 25, 16], to draw insights on flow compressibility and Reynolds number interplaying on LCS. Major outcomes in [22] were three-fold, namely: (i) isotropy enhancement of attracting and repelling manifolds at higher Mach numbers, (ii) similarly, observed increasing isotropy and FTLE disorganization at larger Reynolds numbers, and (iii) attracting FTLE manifolds describing inclined quasi-streamwise vortices or hairpin legs with heads of the spanwise vortex tube located in the outer region of the boundary layer. While a comprehensive dissection of the underlying physics is outside the purview of this paper, a more detailed exploration focusing on Lagrangian coherent structures in SDTBL is earmarked for an upcoming publication.
Figure 4 displays attracting FTLE contours in incompressible SDTBL at low Reynolds numbers. Note that FTLE manifolds represent inclined quasi-streamwise vortices or hairpin legs indicating the presence of high shear.

4. Impact

The implementation introduced in this article sets a new benchmark in the domain of GPU-optimized particle movement computations, particularly in the context of Finite-Time and Finite-Size Lyapunov Exponents. By melding advanced search strategies with computational efficiency, our approach enables the nuanced analysis of LCS within expansive numerical datasets. Its scalability, spanning both GPU and CPU architectures, promises to democ-

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=4745205

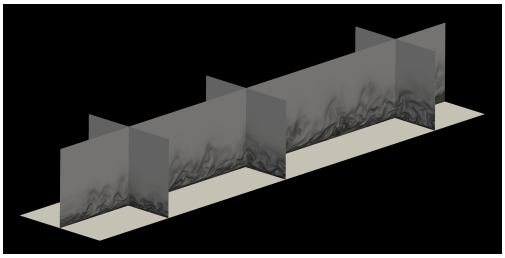

Figure 4: Attracting material lines (FTLE contours) in low-Reynolds number boundary layers (flow from left to right).

ratize access to such tools across diverse computational platforms. Moreover, the technique's application to turbulent boundary layer studies not only offers deeper insights into coherent patterns but also demonstrates remarkable storage efficiencies. As a pivot, this work will undeniably shape future endeavors in visualizing and understanding LCS, catalyzing advancements in fluid dynamics research.
It is worth noting that [5] stated that calculations required for FTLE could be prohibitive in three-dimensions. Nonetheless, we have demonstrated the feasibility and scalability of our approach for large-scale 3D cases.

5. Conclusions

This research introduces an optimized methodology to analyze volumetric particle motion and compute FTLEs and FSLEs, focusing on software design for examining Lagrangian Coherent Structures (LCS) within expansive Direct Numerical Simulation (DNS) datasets. Aquila-LCS is programmed under CPU and GPU architectures, showing strong scaling up to 32,768 CPU cores and up to 62 Nvidia V100 GPUs with almost linear scaling even when tracking 8B particles. By integrating traditional search techniques and contemporary multi-tier strategies, we have developed a unique cell-locating system that ensures swift and efficient cell identification across varied computational architectures. Our methodology has been effectively applied to examine turbulent boundary layer scenarios, unveiling significant patterns in coherent structures. The innovative cell locator mechanism, combined with our approach to efficient particle advection, offers a breakthrough in the domain of fluid dynamics, facilitating in-depth analyses of complex flow patterns with robust scalability. As we have touched upon the application of LCS in this study, a deeper exploration into the intricate physics and compressibility factors in potential LCS scenarios is reserved for subsequent research. In essence, our contribution lies in providing an advanced tool for

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=4745205

FTLE/FSLE calculations, underpinning the pivotal role of software in turbulence research and coherent structure investigations. The software's versatility and efficiency make it an indispensable resource for future fluid dynamics studies, especially when harnessing the capabilities of modern computational infrastructures.

Acknowledgment

This research was funded by the National Science Foundation (NSF) under grants #2314303 and #1847241. This material is based on research sponsored by the Air Force Office of Scientific Research (AFOSR) under agreement number FA9550-23-1-0241. This work was supported in part by a grant from the DoD High-Performance Computing Modernization Program (HPCMP).

References

[1] C. Lagares, W. Rivera, G. Araya, Aquila: A Distributed and Portable Post-Processing Library for Large-Scale Computational Fluid Dynamics, 2021. doi:10.2514/6.2021-1598.
URL https://arc.aiaa.org/doi/abs/10.2514/6.2021-1598

[2] C. Lagares, W. Rivera, G. Araya, Scalable post-processing of large-scale numerical simulations of turbulent fluid flows, Symmetry 14 (4) (2022). doi:10.3390/sym14040823.
URL https://www.mdpi.com/2073-8994/14/4/823

[3] G. Haller, G. Yuan, Lagrangian coherent structures and mixing in two-dimensional turbulence, Physica D 147 (2000) 352–370.

[4] G. Haller, Distinguished material surfaces and coherent structures in three-dimensional fluid flows, Physica D: Nonlinear Phenomena 149 (4) (2001) 248–277. doi:https://doi.org/10.1016/S0167-2789(00)00199-8.

[5] G. Haller, Lagrangian coherent structures, Ann. Rev. Fluid Mechanics 47 (2015) 137–162.

[6] M. Green, C. Rowley, G. Haller, Detection of Lagrangian Coherent Structures in 3D turbulence, Journal of Fluid Mechanics 572 (2007) 111–120.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=4745205

[7] Z. Wilson, M. Tutkun, R. Cal, Identification of lagrangian coherent structures in a turbulent boundary layer, Journal of Fluid Mechanics 728 (2013) 396–416.

[8] C. Pan, J. Wang, P. Zhang, L. Feng, Coherent structures in bypass transition induced by a cylinder wake, Journal of Fluid Mechanics 603 (2008) 367–389.

[9] K. Onu, F. Huhn, G. Haller, LCS Tool: A computational platform for Lagrangian coherent, Journal of Computational Science 7 (2015) 26–36.

[10] G. Saltar, C. Lagares, G. Araya, Compressibility and Reynolds number effect on Lagrangian Coherent Structures (LCS), AIAA AVIATION 2022 Forum (2022). doi:10.2514/6.2022-3627.
URL https://arc.aiaa.org/doi/abs/10.2514/6.2022-3627

[11] D. A. Nelson, G. B. Jacobs, DG-FTLE: Lagrangian coherent structures with high-order discontinuous-Galerkin methods, Journal of Computational Physics 295 (2015) 65–86. doi:https://doi.org/10.1016/j.jcp.2015.03.040.

[12] A. Fortin, T. Briffard, A. Garon, A more efficient anisotropic mesh adaptation for the computation of lagrangian coherent structures, Journal of Computational Physics 285 (2015) 100–110. doi:https://doi.org/10.1016/j.jcp.2015.01.010.
URL https://www.sciencedirect.com/science/article/pii/S0021999115000145

[13] T. Dauch, T. Rapp, G. Chaussonnet, S. Braun, M. Keller, J. Kaden, R. Koch, C. Dachsbacher, H.-J. Bauer, Highly efficient computation of Finite-Time Lyapunov Exponents (FTLE) on GPUs based on three-dimensional SPH datasets, Computers Fluids 175 (2018) 129–141. doi:https://doi.org/10.1016/j.compfluid.2018.07.015.

[14] D. Karrasch, G. Haller, Do finite-size lyapunov exponents detect coherent structures?, Chaos 23 (043126) (2013) 1–11.

[15] R. Peikert, A. Pobitzer, F. Sadlo, B. Schindler, A comparison of finite-time and finite-size lyapunov exponents, in: Topological Methods in Data Analysis and Visualization III, Springer International Publishing Switzerland, 2014.

[16] CTVLab, Computational Turbulence and Visualization Lab, https://ceid.utsa.edu/garaya/ (2023).

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=4745205

[17] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T. E. Oliphant, Array programming with NumPy, Nature 585 (7825) (2020) 357–362. doi:10.1038/s41586-020-2649-2.
URL https://doi.org/10.1038/s41586-020-2649-2

[18] S. K. Lam, A. Pitrou, S. Seibert, Numba: A llvm-based python jit compiler, in: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, LLVM '15, Association for Computing Machinery, New York, NY, USA, 2015. doi:10.1145/2833157.2833162.
URL https://doi.org/10.1145/2833157.2833162

[19] C. Lattner, V. Adve, Llvm: a compilation framework for lifelong program analysis transformation, in: International Symposium on Code Generation and Optimization, 2004. CGO 2004., 2004, pp. 75–86. doi:10.1109/CGO.2004.1281665.

[20] L. Dagum, R. Menon, OpenMP: an industry standard API for shared-memory programming, Computational Science & Engineering, IEEE 5 (1) (1998) 46–55.

[21] C. Pheatt, Intel® threading building blocks, J. Comput. Sci. Coll. 23 (4) (2008) 298.

[22] C. Lagares, G. Araya, A GPU-Accelerated Particle Advection Methodology for 3D Lagrangian Coherent Structures in High-Speed Turbulent Boundary Layers, Energies 16 (12) (2023). doi:10.3390/en16124800.
URL https://www.mdpi.com/1996-1073/16/12/4800

[23] C. Lagares, G. Araya, Power spectrum analysis in supersonic/hypersonic turbulent boundary layers, AIAA SCITECH 2022 Forum (2022). doi:10.2514/6.2022-0479.
URL https://arc.aiaa.org/doi/abs/10.2514/6.2022-0479

[24] G. Araya, C. Lagares, Implicit Subgrid-Scale Modeling of a Mach 2.5 Spatially Developing Turbulent Boundary Layer, Entropy 24 (4) (2022). doi:10.3390/e24040555.
URL https://www.mdpi.com/1099-4300/24/4/555

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=4745205

[25] G. Araya, C. Lagares, K. Jansen, Reynolds number dependency in supersonic spatially-developing turbulent boundary layers, 2020 AIAA SciTech Forum (AIAA 3247313) 6 - 10 January, Orlando, FL. (2020).

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=4745205