The Saaz Framework for Turbulent Flow Queries

Alden King * Eric Arobone † Scott B. Baden * Sutanu Sarkar †
* Department of Computer Science and Engineering University of California, San Diego
† Department of Mechanical and Aerospace Engineering University of California, San Diego

***Abstract*—In many respects, numerical simulations involving solutions to partial differential equations have replaced physical experimentation. However, few tools are available to sift through the deluge of data. We present *Saaz*, a query framework to analyze the simulation results of multi-scale physical phenomena which admit mathematical rules for characterizing features of interest. *Saaz* provides high-level primitives that free the domain-scientist to concentrate more on scientific discovery and less on code implementation and maintenance. It supports user-defined domain-specific query operations which may be subsequently composed into more complex queries. While *Saaz* supports offline processing of queries, we explore here the online capabilities by attaching *Saaz* to a running simulation, improving the simulation's effective temporal resolution. We discuss analysis for a computational fluid dynamics simulation of turbulent flow running on a cluster.**

I. Introduction

In some application domains, simulation has only begun to realize its potential as the "third leg" of science: the ability to understand scientifically important phenomena is limited by the paucity of available tools for discovering knowledge hidden within simulation data [1]. Indeed, the CRA report *Workshop on the Roadmap for the Revitalization of High End Computing* notes that "developing... new implementations [of data analysis codes] is often perceived as a poor use of human resources." [2, p. 73].

In turbulent flow studies, sophisticated algorithms and adaptive data structures are needed to sift through data, and the definition of what is interesting often changes as the user engages in scientific discovery. In the hope of discovering the elusive needle in an ever growing haystack, domain scientists use ad-hoc methods of data discovery, building custom tools tailored to specific problems. As requirements change, the tools must be recoded, sometimes at great cost.

Due to their well known inability to preserve memory locality in tightly bound loop nests, relational databases have failed to gain general acceptance by the CFD community in analyzing flow data [3]. Some publicly-available tools exist, but provide a fixed set of primitives to the user. When the discovery process motivates new forms of queries, a custom solution is the natural recourse, leading to maintainability problems for analysis software that also limits the ability to compare and share experimental data.

We discuss *Saaz*, a *computational query framework* for studying turbulent flows. The goal of *Saaz* is not only to remove obstacles to effective data discovery, but to change the way that scientists engage in such discovery. *Saaz* preserves locality inherent to flow analysis queries, thus permitting users to compose efficient custom queries. As a result, users may extend existing analysis tools without entailing costly re-coding, and may share flow analysis libraries with others. *Saaz* supports dynamic representations of irregular time-dependent structures arising in turbulent flow, a capability crucial in applying queries conditionally, e.g. to features of interest.

By providing a framework which scientists can customize to their own situations, we ease the development of new scientific models. *Saaz* not only provides certain primitives needed in studying turbulent flow, but, more importantly, the ability for users to devise their own primitives as their models grow in complexity or their sense of what is "interesting" changes. We show that *Saaz* meets this expectation in an actual case study.

To examine the usefulness of the *Saaz* framework, Sec. II presents a case study of turbulent flow in computational fluid dynamics carried out by two of us (Arobone and Sarkar) in the Computational Fluid Dynamics Lab in the Mechanical and Aerospace Engineering Department at UCSD. Since the lab implements its own custom simulators, different members have the responsibility of maintaining different simulators and post-processing code. The need to maintain simulators is unavoidable due to the deficiencies of commercial and open-source flow solvers; *Saaz* has the potential to replace all post-processing code currently used by the lab. One of the primary problems with integrating post-processing code with a simulator is that some queries lend themselves to different data decompositions than the simulator may use. Having a stand-alone post-processing tool like *Saaz*, which can also process code at runtime, is very valuable. While minor changes in simulators or problems under investigation can necessitate extensive revision of existing post-processing code, *Saaz* is flexible enough to require only minor modifications of inputs without the need to rewrite large amounts of code. The ability to concentrate on scientific discovery rather than coding will be valuable to members of the lab who find that much of their time is spent coding.

The plan for the rest of this paper is as follows. Sec. II describes our case study. Sec. III presents the *Saaz* API as well as a duscussion of how we implement in *Saaz* the turbulent flow queries from our case study. Sec. IV presents our results. We conclude with related work and a discussion.

II. Case Study: Simulation of Turbulent Flow

Formally defining turbulence is not trivial, but several features are universal in turbulent flows. Turbulence is dissipative, meaning that significant amounts of kinetic energy are converted continuously into internal energy. Therefore, turbulent flows tend to decay rather rapidly when no energy source is present. Turbulent flows are also highly nonlinear, resulting in flow that appears random and is highly sensitive to initial conditions. Turbulence, however, is not truly random and has statistics that are highly reproducible when appropriate averaging (ensemble, spatial, or temporal) is implemented. Efficient mixing is another feature of turbulent flows. For example, efficient mixing of momentum leads to increased drag on solid bodies such as airplane wings. The adverse health effects of tail-pipe emissions on pedestrians are minimized through efficient dispersion of particulates. Turbulence is three-dimensional with velocity and vorticity fluctuations of comparable magnitude in all three directions [4].

In the ocean, spatial density variation (stratification) results from gradients of temperature and salinity. The atmosphere is also a stratified system, where density variations result primarily from temperature and water vapor gradients. When stratification is such that density is a function of elevation alone, with lighter fluid positioned above heavier fluid, then stratification acts to stabilize turbulence, in turn inhibiting vertical motion. If stratification differs from this base state then energy can be extracted from the density field, providing an energy source for turbulent mixing and internal wave emission.

Oceanic flow with scales much larger than those at which energy is dissipated and much smaller than geostrophic scales are poorly understood. These scales are influenced by stable stratification and moderate effects of the Earth's rotation [5]. Currently we are investigating a model scale flow configuration representing a particular class of shear instabilities in stratified flow called barotropic instabilities. These instabilities are known to form coherent vortical structures, which are important in geophysical flows. It is our objective to increase physical understanding of the role these structures play in environmental mixing and transport by performing statistic averaging conditioned upon the vortical structures. Ocean models require better parameterization of physical processes to become an accurate predictive tool.

The model problem (Fig. 1) corresponding to a temporally evolving shear layer representing two streams with velocity difference ΔU oriented horizontally, is subjected to uniform vertical stratification. The vertical and streamwise directions are assumed to be infinite and homogeneous, and are thus appropriate directions over which to perform statistical averaging. Broadband fluctuations are applied in the region between the two streams to accelerate transition to turbulence.

The approach used to simulate the flow is called Direct Numerical Simulation (DNS). DNS consists of solving the Navier-Stokes equations on a computational domain using no empirical models, such as the use of eddy viscosity or a Smagorinsky model. DNS is not tractable for many engineering problems due to the wide range of spatial and temporal scales that must be resolved, but it has served as a successful research tool owing to its lack of empiricism. We solve the Navier-Stokes equations using a pseudospectral method with staggered second order finite difference computation of derivatives in the inhomogeneous direction and spectral collocation in the homogeneous directions. Conservation of mass is enforced using a projection method, which is solved using a parallel Thomas algorithm. The pressure solve reduces to a tridiagonal system of equations which is also solved using a parallel Thomas algorithm. Density advances in time via an advection-diffusion equation. Time marching is accomplished using a low storage third order mixed Runge-Kutta/Crank-Nicholson scheme with viscous terms treated implicitly [6].

III. Saaz

Relational queries have proven useful in certain scientific application domains, notably, the Sloan Digital Sky Survey [7]. However, our queries tend to be computationally expensive, and they exhibit extreme locality in loop nests. Relational query models can disrupt this locality, leading to severe performance penalties (although in some cases, compilation strategies can mitigate this effect). *Saaz* instead supports an imperative query model, allowing scientists to naturally express equations in a manner they are accustomed to. It offers higher-level semantic concepts that hide tedious details of implementation, saving valuable time.

A. The Imperative Query Model

Saaz presents a concise model with a few fundamental datatypes. A *Point* is a tuple in $\mathbb{Z}^n$, and may be optionally qualified with the number of spatial dimensions as in Point1, Point2, and so on. We may access a single component of a Point as a pre-defined member, e.g. $p.x$, $p.y$, $p.z$. We may also sum points as a vector sum. This capability is useful in accessing nearest neighbors on a Cartesian mesh. For example $p + \vec{Y}$, where $p = (p_x, p_y, p_z)$ and $Y = (0, 1, 0)$ corresponds to the point $(p_x, p_y + 1, p_z)$ and so on.

A *Domain* represents a subset of $\mathbb{Z}^n$ and can be either dense or sparse. A dense *Domain* is the cross-product of integer ranges, $[i_0 : i_1] \times [j_0 : j_1]$. A sparse *Domain* is formed using *Predicate*s, which will be described shortly.

An *Array* is a mapping of values, all of the same type, from an index space defined by a *Domain* object. We can query the domain of an *Array* $\mathcal{A}$ using the δ operator, i.e. $\delta(\mathcal{A})$. We may index an *Array* with a *Point* rather than using a tuple as in traditional programming languages. This capability was pioneered in the Fidil programming language [8] and later employed in the KeLP system [9], [10] and the Titanium programming language [11].

A Parallel `forall` iteration is provided, in which the Point-valued iterator takes on all values within a specified *Domain* object, which may be an expression. For example, we can linearly combine two arrays $\mathcal{B}$ and $\mathcal{C}$ as follows, assuming that all arrays are defined on the domain of $\mathcal{A}$.

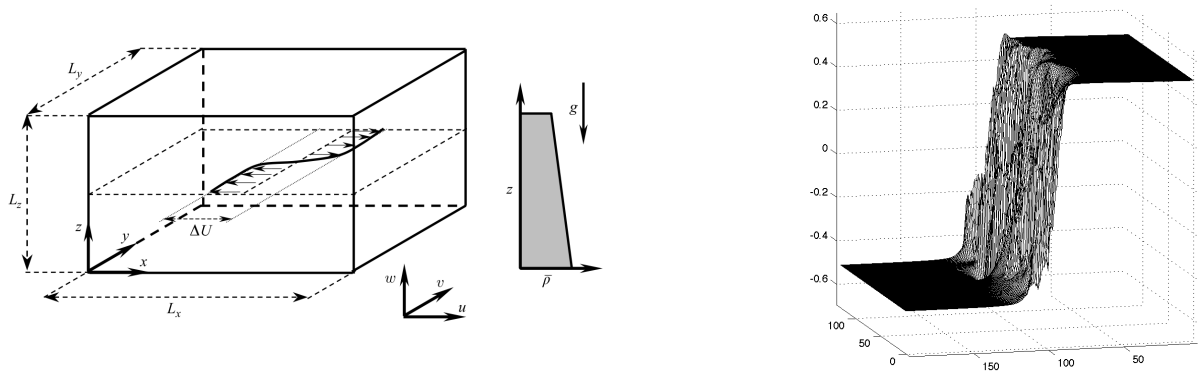

Fig. 1. (Left) Configuration of the temporally evolving shear layer. Two streams of fluid are shown, one moving in the positive x direction and the other in the negative x direction. The streams are separated by the $y = 256$ mid-plane. Stratification (mean density variation) is in the z direction, which is shown by the constant density gradient, $d\overline{\rho}/dz$. (Right) Simulation data for the u velocity in the $y - z$ plane.

```
1  ∀ Point1 p ∈ δ(A, j)
2     Domain2 pl := Slice(δ(A), p, j)
3     M[p] := 0
4     ∀ Point2 p2 ∈ pl
5        M[p] := M[p] + A[p ⊕ p2]
6     M[p] := M[p] / size(pl)
```

Fig. 2. Pseudo-code for the planar average query along the j axis. The $\oplus$ operator creates an appropriate 3D point from its 1D and 2D arguments.

$$\forall \text{ Point } p \in (\delta(\mathcal{A}))$$
$$\mathcal{A}[p] \;:=\; \mathcal{B}[p] * x + \mathcal{C}[p]$$

A *Predicate* constrains execution over a subset of a domain for arbitrary conditions. Here we modify the previous loop to perform sums where $\mathcal{C} < \epsilon$. This is similar to the *where* clause in CM-Fortran and HPF [12], [13].

$$\forall \text{ Point } p \in (\delta(\mathcal{A}) \text{ where } \mathcal{C}[p] < \epsilon)$$
$$\mathcal{A}[p] \;:=\; \mathcal{B}[p] * x + \mathcal{C}[p]$$

*Predicate*s can also be used to construct irregular domains. We can think of an irregular domain as a bit mask that specifies which points are present in the domain. Irregular domains first appeared in the Fidil language as well as LPAR [14] and the Titanium programming language.

Saaz provides the *Slice* primitive to select a subdomain along one or more axes. This operation achieves the effect of the MATLAB colon notation, but encodes the axes (and bounds) to be encoded as an object rather than as an index expression. Domain slicing is useful in our turbulent flow queries. For example, many queries require an array of the average value of each plane of an array along a specified direction. The code for this useful primitive is show in Fig. 2.

By keeping the concepts in the model similar to those in common scientific languages such as MATLAB and Fortran, we keep our model more intuitive for scientists. The domain concept is an extremely powerful one and serves as meta-data that may be subsequently optimized.

`PlanarAverage` is a simple *query*; we often combine it with other operations to create more complex queries. In our analyses of turbulent flow, for example, normalization of a value requires that we subtract, from each value in an array, the planar average of an appropriately oriented plane enclosing the point. Normalization of $\mathcal{A}$ for each point p could be written as: $\mathcal{A}[p] \;:=\; \mathcal{A}[p] - \texttt{PlanarAverage}(\mathcal{A}, \delta(\mathcal{A}), p)$

`PlanarAverage` and `Normalization` are our simplest queries, and are built atop *Saaz* query primitives. From these two basic queries we build up more complicated ones, such as *buoyancy flux* and *energy transport*, described next.

B. Sample Queries

To show how *Saaz* query primitives work, we examine four sample queries. The first query, buoyancy flux, quantifies transfer of energy from the turbulent velocity to the turbulent scalar fields. The second and third queries, the energy transport terms, quantify the motion of energy within the turbulent velocity and turbulent scalar fields. These three queries help provide a detailed picture of the behavior of energetics in stratified turbulence. The fourth query, λ_2, helps split the domain into vortical and non-vortical subdomains for additional physical insight. We refer to Tab. I which describes the queries mathematically and in pseudo-code. These cases operate on the full flow field, that is, every point of the simulation domain, and then provide summary values for individual planes.

We also consider three of these queries over a subdomain, as defined by a predicate on the fourth query ($\lambda_2 < \epsilon$). In this case we use $\epsilon = -1 \times 10^{-2}$. Only a small fraction of points in the domain satisfy this condition, approximately 7% for the data we are working with, although the exact fraction is time-dependent and varies from simulation to simulation.

1) Buoyancy Flux: The buoyancy flux quantifies the rate of exchange between turbulent potential and turbulent kinetic energies due to small scale motions. A smoke plume generates buoyancy flux as potential energy, in the form of thermal energy, is converted into kinetic energy, resulting in motion against gravity. Overall, buoyancy flux is the mechanism

```
IN:   ρ, w
OUT:  bfx
1     ρ_a := PlanarAverage(ρ, δ(ρ))
2     w_a := PlanarAverage(w, δ(w))
3     ∀ Point1 p ∈ δ(w, j)
4       Domain2 pl := Slice(δ(w), p, j)
5       ∀ Point p2 ∈ pl
6         bfx[p] += ((ρ[p ⊕ p2] − ρ_a[p]) *
                     (w[p ⊕ p2] − w_a[p]))
7       bfx[p] := bfx[p]/size(pl)
```

Fig. 3. Pseudo-code for the buoyancy flux query of $\langle \rho' w' \rangle$ with normalization inlined. $\langle \cdot \rangle$ denotes the planar average operator, and $\cdot'$ denotes normalization.

```
IN :  domain,   ρ, v
OUT:  pet
1     ρ_a := PlanarAverage(ρ, domain)
2     v_a := PlanarAverage(v, domain)
3     ∀ Point p ∈ domain
4       tmp[p] := (ρ[p] − ρ_a[p.y])^2 * (v[p] − v_a[p.y])
5     ddy_tmp := ddy(tmp)
6     pet := PlanarAverage(ddy_tmp, domain)
```

Fig. 4. Pseudo-code for the potential energy transport query of $\langle \rho' \rho' v' \rangle$ with normalization inlined. $\langle \cdot \rangle$ denotes the planar average operator, and $\cdot'$ denotes normalization. $p.y$ signifies the y-component of point p. `ddy` performs the numerical approximation of the derivative in the y direction.

through which stratification directly influences turbulent energetics. The overall effect of buoyancy flux is the suppression of vertical motion, resulting in primarily horizontal motion in highly stratified flows. However, in localized regions or small periods of time, buoyancy flux can drive vertical motion as the density field relaxes to a state of lesser potential energy.

The buoyancy flux calculation is a relatively simple query. It consists of a two-variable correlation. Correlations are pointwise products which are averaged over their plane. This can be done with a two-level loop, the outer loop iterating over each plane, and the inner loop computing the sums of the products within that plane. This is an extension of the `PlanarAverage` query described above. Instead of just averaging a single array, we average over the evaluation of two 3D arrays (ρ' and w') at each point. Fig. 3 shows the buoyancy flux computation using the *Saaz* framework.

2) Energy Transport: The energy transport terms quantify the net lateral transport of velocity and density fluctuations due to small scale motions. Generally speaking, turbulent fluctuations act to accelerate the spread, or transport, of turbulent energetics.

Energy transport consists of two separate queries: kinetic energy transport and potential energy transport. We are primarily interested in transport through planes intersecting the inhomogeneous axis. Potential energy is a similar calculation to buoyancy flux, in that it also involves a correlation. Kinetic energy is a sum of correlations and a derivative is applied before averaging the correlation. When operating on the full domain, the derivative and planar average are associative as shown in Fig. 4

3) λ_2 (Lambda 2): The definition of a vortex is not trivial in turbulent flows. While no universal vortex eduction criteria exists, there are several popular methods based upon manipulation of the rate of deformation tensor. A common criterion is the λ_2-criteria [15] defined as the median eigenvalue of the symmetric tensor $S_{ik}S_{kj} + \Omega_{ik}\Omega_{kj}$ where S and Ω are the rate of strain and rate of rotation tensors, respectively. λ_2 identifies pressure minima in a flow field neglecting the effects of viscosity and unsteady straining, which are known to make the simple pressure minima criterion incorrectly identify regions of the flow as vortical.

λ_2 is a costly computation. While it does operate independently at each point using a 7-point stencil, it has many arithmetic, and some trigonometric computations. This leads to a predictable memory access pattern and a CPU-bound computation. We also query both buoyancy flux and energy transport restricted to those locations where $\lambda_2 < \epsilon$, for small parameter ϵ.

IV. Results

Our results were obtained on the Triton Compute Cluster, located at the San Diego Supercomputer Center. Triton provides 256 Appro gB222X Blade Servers, each with dual-socket quad-core Intel Xeon E5530 (Nehalem, 2.40 GHz) processors and 24 GBytes of memory, for a total of 2048 cores. C++ code was compiled with `g++ v4.1`, using compiler options `-g3 -m64 -O3 -lmpi -lmpi_cxx -lpthread`; Fortran code was compiled with `ifort 11.1`, using compiler options `-i-dynamic -mcmodel=medium -shared-intel -g3 -m64 -O3`. We used openmpi 1.3, an open-source implementation of MPI2.

We conducted Direct Numerical Simulations using a code developed in the UCSD CFD Lab. This code solves the Navier Stokes equations and simulates mixing of density in stratified flows with multiscale dynamics involving coupled evolution of coherent structures, turbulence and internal waves [16]. The simulations used here were carried out on a mesh of size $641 \times 512 \times 192$ and ran for 20 timesteps, starting from an advanced stage of the simulation when the physics become "interesting." A full scale simulation would run for considerably longer–for thousands of timesteps– and on a larger mesh.

We periodically query the simulation state, consisting of 4 double precision values at each point in the mesh: 3D vector-valued velocity and scalar pressure. Due to timestep constraints, simulations evolve gradually, and thus it is unnecessary to conduct queries every timestep. Turbulence simulation state generally contains a considerable amount of uninteresting background data, and this is what *Saaz* filters out. In our simulations, only 7% of the data was retained, though this quantity is simulation dependent. Constraints in the storage of generated data restrict scaling more than running time (64 cores is sufficient), so data reduction is more important than computational duration. The conventional way of conducting queries is to do so off-line. This approach does not take advantage of filtering; thus *Saaz* effectively raises the time resolution of queries for a given storage budget.

Our strategy provides considerable flexibility. Some queries require a different method of data decomposition than used

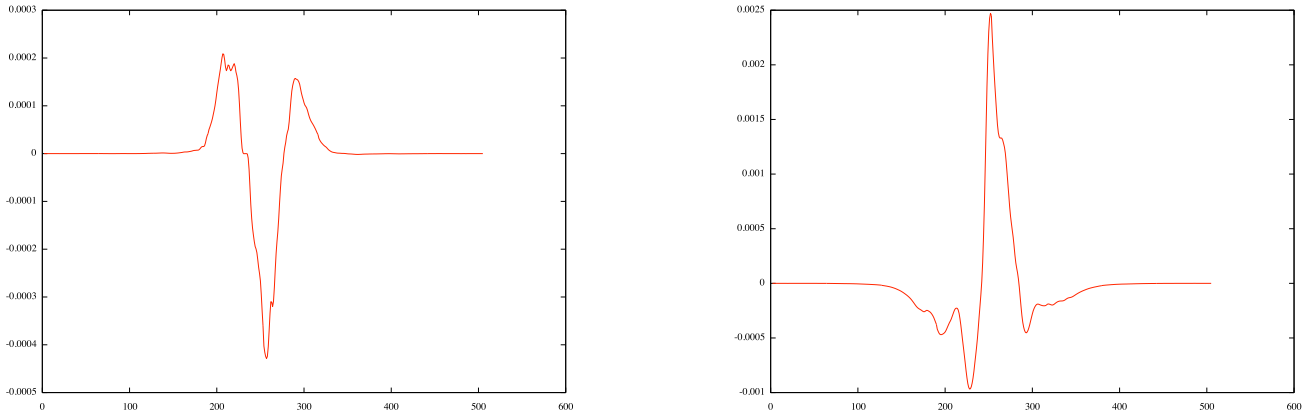

Fig. 5. Turbulent potential energy transport (Left) and turbulent kinetic energy transport (Right), at a time-step in the middle of the simulation. Net inward transport of turbulent potential energy and net outward transport of turbulent kinetic energy are observed. The inward transport of potential energy drives the collapse of vortical structures and the coherent nature of the structures enables pronounced lateral transport of kinetic energy.

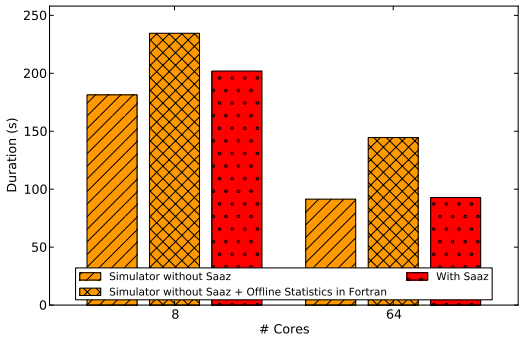

Fig. 6. Running times for the simulator with and without offline query analysis in Fortran are shown on the hashed orange bars. Running times for the simulator with *Saaz* are shown in the dotted red bars. Times are wall-clock, not node-seconds.

by the simulator. Autocorrelations, for instance, do a point-to-point comparison of all the points in the domain, requiring many data gathers. Because *Saaz* is decoupled from the simulator, however, it can perform these analyses online using a data decomposition separate from that of the simulator.

Fig. 6 shows that *Saaz* adds little overhead. The right-hatched bar shows the duration of the simulator when operating by itself on either one or two nodes. The double-hatched bar includes the cost of offline analysis. Offline analysis is performed on a single node after the run is complete, and here does not include the cost of data motion to that node. The dotted bars indicate the duration of the simulator attached to a *Saaz*-based analyzer which runs concurrently. Utilizing *Saaz* increases this to only about 11% in the single-node case, dropping to less than 2% on two nodes. As a short-running task, CPU contention slows down *Saaz* more than the simulator. In both cases, *Saaz* effectively reduces the running time for the full analysis by significant margins.

V. Related Work

SciDB [17] is a current project to build a database management system for the scientific community at large. SciDB plans to provide an array-based data model and is designed to scale to much larger sizes than *Saaz* (petabytes). SciDB focuses on such concepts as uncertainty and provenance, which are currently outside the scope of *Saaz*. AQL [18] and AML [19] provide frameworks for manipulating and representing arrays. Both present arrays as functions and define common operations over them, constructing a compiler for their queries. AML focuses heavily on preserving locality in its operations, but is geared mostly towards two-dimensional image data. Predication is absent from AML, but AQL does support a similar operation via *filter*.

Map Reduce [20] and Dryad [21] provide large data processing frameworks designed to extract information from large datasets and assist model-building.

Relational databases have not gained general acceptance by the CFD community [3] and there have been few inroads. Notably, the JHU Turbulence Database Cluster stores DNS datasets, and provides an interface enabling users to author analysis codes that run locally using data stored remotely on a SQL server [22]. The Cluster provides a set of built-in primitives that are generally useful for performing queries on the DNS data. iCFDdatabase is a repository for simulation datasets [23].

The main impediment to employing relational database technology in CFD is that the relational model disrupts locality in *imperative* computations consisting of sequences of tight loop nests. These loop nests are ubiquitous in flow simulations as well as algorithms that analyze the data. Retrieval is an inefficient way to access data represented as multidimensional arrays.

The most common approaches to studying flow data do not involve the use of a formal relational database at all, and employ computational steering environments [24], [25] or application specific tools [26]–[28]. In many cases the tools are ad-hoc and are perceived as a means rather than an end.

This state of affairs inhibits the dissemination and sharing of ideas and tools.

VI. Discussions and Conclusions

We have presented the *Saaz* framework for conducting queries on turbulent flow simulations. *Saaz* provides appropriate abstraction for expressing fluid dynamics queries at a high level of detail. The domain abstraction provides a crucial capability that enables retrieved data to be accessed efficiently, conserving locality within computational loop nests. This is currently a challenge in the (object) relational database world. We use predication to construct more complicated queries involving sparse subdomains.

Online analysis enables *Saaz* to increase temporal resolution. We ran the queries concurrently with simulations and produced results that are consistent with those obtained with hand coded analysis tools written in Fortran. While we were initially developing *Saaz*, we verified it against the existing flow analysis tools. Verification works both ways, however, and *Saaz* revealed several bugs in the hand coded Fortran tools.

Saaz is a framework, and it enabled us to build new query primitives in far less time and effort than in Fortran. Soon it will replace hand coded tools, which are difficult to re-target to related turbulent flows, which have similar notions of locality, but exhibit different physical behavior.

We have designed *Saaz* to be easy to use and to operate at a high level of abstraction. We believe we have achieved this goal. *Saaz* enables its users to conduct queries without becoming entangled with the implementation details. *Saaz* is designed for Cartesian meshes. The domain abstraction we introduced may or may not apply to irregular meshes such as finite elements, though it should apply to Structured Adaptive Mesh Refinement (SAMR) methods [29], since earlier tools that employ the domain abstraction were designed with SAMR in mind [8], [10], [11], [30].

While our queries perform quickly enough for our current simulation, we anticipate more complicated queries in the future. For example, we would like to be able to execute queries fast enough to enable immediate visualization of results or animation of the query results along the temporal axis, or to conduct queries off-line on stored data sets. We are implementing a source-to-source translator and optimizer to address these requirements, and will describe them elsewhere.

Acknowledgment

This work was supported by NSF contract OCE-0835839. Computer time on the Triton system was provided by the San Diego Supercomputer Center and the University of California, San Diego.

References

[1] C. J. et al., "Visualization and knowledge discovery: Report from the doe/ascr workshop on visual analysis and data exploration at extreme scale," Tech. Rep., October 2007. [Online]. Available: http://science.doe.gov/ascr/ProgramDocuments/Docs/DOE-Visualization-Report-2007.pdf

[2] D. R. at al., "Roadmap for the revitalization of high-end computing," Tech. Rep., 2003. [Online]. Available: http://usms.nist.gov/roadmaps/object.cfm?ObjectID=21

[3] Y. Li, E. Perlman, M. Wan, Y. Yang, C. Meneveau, R. Burns, S. Chen, A. Szalay, and G. Eyink, "A public turbulence database cluster and applications to study lagrangian evolution of velocity increments in turbulence," *arXiv.org*, pp. 1–31, 2008. [Online]. Available: http://arxiv.org/abs/0804.1703v1

[4] H. Tennekes and J. L. Lumley, *A first course in turbulence*. The MIT press, 1972.

[5] J. J. Riley and E. Lindborg, "Stratified turbulence: A possible interpretation of some geophysical turbulence measurements," *Journal of the Atmospheric Sciences*, vol. 65, no. 7, pp. 2416–2424, 2008. [Online]. Available: http://journals.ametsoc.org/doi/abs/10.1175/2007JAS2455.1

[6] T. Bewley, *Numerical Renaissance: simulation, optimization & control*. San Diego: Renaissance Press, 2010.

[7] J. Gray, D. Slutz, A. Szalay, A. Thakar, J. vandenBerg, P. Kunszt, and C. Stoughton, "Data mining the sdss skyserver database," Microsoft Research, Tech. Rep. MSR-TR-2002-01, January 2002. [Online]. Available: http://research.microsoft.com/research/pubs/view.aspx?msr_tr_id=MSR-TR-2002-01

[8] P. N. Hilfinger and P. Colella, "Fidil: A language for scientific programming," Lawrence Livermore National Laboratory, Tech. Rep. UCRL-98057, January 1988.

[9] S. J. Fink, "Hierarchical programming for block–structured scientific calculations," Ph.D. dissertation, Department of Computer Science and Engineering, University of California, San Diego, 1998.

[10] S. J. Fink, S. B. Baden, and S. R. Kohn, "Efficient run-time support for irregular block-structured applications," *J. Parallel Distrib. Comput.*, vol. 50, no. 1-2, pp. 61–82, April-May 1998.

[11] K. Yelick, L. Semenzato, G. Pike, C. Miyamoto, B. Liblit, A. Krishnamurthy, P. Hilfinger, S. Graham, D. Gay, P. Colella, and A. Aiken, "Titanium: A high-performance Java dialect," *Concurrency—Practice and Experience, Java Special Issue*, 1998.

[12] G. W. Sabot, "Optimizing cm fortran compiler for connection machine computers," *J. Parallel Distrib. Comput.*, vol. 23, no. 2, pp. 224–238, 1994.

[13] High Performance Fortran Forum, "High Performance Fortran language specification, version 1.0," Center for Research on Parallel Computation, Rice University, Houston, TX, Tech. Rep. CRPC-TR92225, 1993.

[14] S. B. Baden and S. R. Kohn, "Portable parallel programming of numerical problems under the lpar system," *J. Parallel Distrib. Comput.*, pp. 38–55, May 1995.

[15] J. Jeong and F. Hussain, "On the identification of a vortex," *J. Fluid Mech.*, vol. 285, pp. 69–94, 1995.

[16] S. Basak and S. Sarkar, "Dynamics of a stratified shear layer with horizontal shear," *J. Fluid Mech.*, vol. 568, pp. 19–54, 2006.

[17] M. Stonebraker, J. Becla, D. Dewitt, K.-T. Lim, D. Maier, O. Ratzesberger, and S. Zdonik, "Requirements for science data bases and SciDB," *Conference on Innovative Data Systems Research 2009*, 2009. [Online]. Available: http://www-db.cs.wisc.edu/cidr/cidr2009/Paper_26.pdf

[18] L. Libkin, R. Machlin, and L. Wong, "A Query Language for Multidimensional Arrays: Design, Implementation, and Optimization Techniques," in *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*. ACM, 1996, pp. 228–239.

[19] A. P. Marathe and K. Salem, "Query processing techniques for arrays," *The VLDB Journal*, vol. 11, no. 1, pp. 68–91, 2002.

[20] J. Dean and S. Ghemawat, "Map Reduce: Simplified data processing on large clusters," *Communications of the ACM-Association for Computing Machinery-CACM*, vol. 51, no. 1, pp. 107–114, 2008.

[21] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. ACM, 2007, p. 72.

[22] E. Perlman, R. Burns, Y. Li, and C. Meneveau, "Data exploration of turbulence simulations using a database cluster," in *Proceedings of the ACM/IEEE SC2007 Conference*, Reno, NV, November 2007.

[23] F. Toschi, "icfd database," Tech. Rep., 2008. [Online]. Available: http://cfd.cineca.it/cfd/cfd-other-databases/pk-yeung-database

[24] C. Johnson, S. Parker, and D. Weinstein, "Large-scale computational science applications using the scirun problem solving environment," in *Proc. Supercomputing*, Nov. 2000.

TABLE I

QUERIES DISCUSSED IN THE TEXT. OUR QUERIES NEGLECT THE LEADING CONSTANTS IN THESE PHYSICAL TERMS. SUCH CONSTANTS MAY VARY ON A CASE-BY-CASE BASIS, AND MAY EVEN BE ZERO, RESULTING IN A LOSS OF INFORMATION (SUCH AS IN THE CASE OF POTENTIAL ENERGY TRANSPORT IN UNSTRATIFIED FLOWS). u, v, AND w REPRESENT THE x, y, AND z COMPONENTS OF VELOCITY, RESPECTIVELY; ρ IS TEMPERATURE. $\langle Expr \rangle$ IS THE PLANAR AVERAGE OPERATOR, REDUCING A 3D ARRAY TO A 1D ARRAY. "PRIMED" VARIABLES APPEARING IN COLUMN 2 (SUCH AS u') REPRESENT THE PLANAR NORMALIZATION OF THE VARIABLE; WE ADJUST THE VALUE AT EACH POINT BY SUBTRACTING THE AVERAGE VALUE OF ALL POINTS IN THE ENCLOSING PLANE ALONG WHAT IS KNOWN AS THE INHOMOGENEOUS DIRECTION (WHICH IS THE Y-DIRECTION FOR THE SIMULATIONS RUN IN THIS PAPER). THIS IS EQUIVALENT TO COMPUTING $u' = u - \langle u \rangle$. $\frac{\partial^h}{\partial y}$ IS THE DISCRETE APPROXIMATION TO THE PARTIAL DERIVATIVE WITH RESPECT TO y.

Description	Operation	Pseudo Code
Lambda 2	λ_2	
Buoyancy Flux	$\langle \rho' w' \rangle$	$\rho\_a := PlanarAverage(rho)$ $w\_a := PlanarAverage(w)$ $\forall$ Point $p \in domain$ $correl_p := (\rho_p - \rho\_a_{p.y}) * (w_p - w\_a_{p.y})$ $return(PlanarAverage(correl))$
Kinetic Energy Transport	$\langle \frac{\partial}{\partial y} \left(v' \lvert (u', v', w') \rvert^2 \right) \rangle$	$u\_a := PlanarAverage(u)$ $v\_a := PlanarAverage(v)$ $w\_a := PlanarAverage(w)$ $\forall$ Point $p \in domain$ $u' := (u_p - u\_a_{p.y})$ $v' := (v_p - v\_a_{p.y})$ $w' := (w_p - w\_a_{p.y})$ $correl_p := \lvert (u', v', w') \rvert^2 * v'$ $deriv := \frac{\partial^h}{\partial y} correl$ $return(PlanarAverage(deriv))$
Potential Energy Transport	$\langle \frac{\partial}{\partial y} (\rho' \rho' v') \rangle$	$\rho\_a := PlanarAverage(rho)$ $v\_a := PlanarAverage(v)$ $\forall$ Point $p \in domain$ $correl_p := (\rho_p - \rho\_a_{p.y})^2 * (v_p - v\_a_{p.y})$ $deriv := \frac{\partial^h}{\partial y}(correl)$ $return(PlanarAverage(deriv))$

[25] V. Mann, V. Matossian, R. Muralidhar, and M. Parashar, "Discover: An environment for web-based interaction and steering of high-performance scientific applications," *Concurrency and Computation: Practice and Experience*, vol. 13, no. 8-9, pp. 737–754, 2001.

[26] D. Silver and X. Wang, "Tracking and visualizing turbulent 3d features," *IEEE Transactions on Visualization and Computer Graphics*, vol. 3, no. 2, pp. 129–141, April-June 1997.

[27] F.-Y. Tzeng and K.-L. Ma, "Intelligent feature extraction and tracking for visualizing large-scale 4d flow simulations," in *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2005, p. 6.

[28] L. Finn, C. Kottke, and B. Boghosian, "Vortonics harnessing the power of grid computing to study the evolution and interaction of vortex cores in three-dimensional hydrodynamics," Tech. Rep., 2005. [Online]. Available: http://hilbert.math.tufts.edu/ bruceb/VORTONICS

[29] M. J. Berger and J. Oliger, "Adaptive mesh refinement for hyperbolic partial differential equations," *J. Comp. Phys.*, vol. 53, no. 3, pp. 484–512, Mar. 1984.

[30] P. Colella, D. T. Graves, D. Modiano, D. B. Serafini, , and B. van Straalen, "Chombo software package for amr applications," University of California, Lawrence Berkeley National Laboratory, Tech. Rep., 2000, also available as `http://seesar.lbl.gov/anag/chombo`.