The Saaz Framework for Turbulent Flow Queries

2011.12.08 eScience Stockholm, Sweden

Alden King¹, Eric Arobone²,
Scott Baden³, Sutanu Sarkar²

University of California San Diego
¹Computer Science and Engineering
²Mechanical and Aerospace Engineering
Numerical Simulation

- Simulate low-level equations
  - Try to find higher-level patterns
- Produces large datasets
  - Data size reduces frequency
  - Analyze on the spot
    - Too much to analyze by hand
    - Slow down simulation
    - Complicates code
  - Save lots of data for later analysis
Analyzing Data

- Lots of data
- Few analysis tools
  - Inappropriate assumptions
  - Cannot customize
- Existing analysis tools are ad-hoc
  - Small changes to an experiment require extensive reprogramming
Data Analysis

- Relational databases
  - Useful for data retrieval and simple queries
  - Do not preserve locality
  - Declarative queries (what to retrieve)

- Saaz
  - More familiar to domain scientists
  - Preserve locality
  - Imperative queries (sequence of action)

```java
double sum = 0;
for (int i = 0; i < u.size(); ++i)
    sum += u[i];
```

```sql
SELECT SUM(value) FROM u
```
Outline

1. Case Study - Simulation of Turbulent Flow
2. The Saaz Query Framework
3. Example Queries
4. Results
Turbulent Flow

- Solve incompressible Navier-Stokes equations in Boussinesq form
- 4 Variables \((u, v, w, \rho)\): velocity and density
Coherent Structures in Turbulent Flow

- Disorganized and organized structures
  - Organized – inside $\lambda_2$ structures
  - Disorganized – outside $\lambda_2$ structures
- Statistics in organized or disorganized regions

Saaz

- C++ Array Library
  - Obtain statistics conditional on structures
  - Discover the dynamical importance of organized structures
- Objects
  - Points in $\mathbb{Z}^n$
  - Domains
    - Cross-product of points – rectangle in $\mathbb{Z}^n$
    - Iteration space
  - Arrays
    - Point $\rightarrow$ Value map
    - One of several layouts
Saaz

Array A

Domain2 D

shrink

Domain2 d

iterate

foreach p in d

query(A[p])

Handling Boundary Conditions with Padding
We:

\[
\text{iterate}
\]

\[
\text{foreach } p \text{ in } d
\]

\[
\text{query}(A[p])
\]

Sparse Domains / Conditional Execution

\[
\text{where } B[p] < 1
\]
Saaz

- Iterators simplify traversing domains
- Can easily aggregate across multiple dimensions at a time
- Easy to change properties of the grid/problem
- Easy to restrict operations

\[
\forall \text{ Index1 } \ p_1 \in \delta(A,YDIM) \\
\text{Domain2 } P\ln := \text{Slice}(\delta(A),p_1,YDIM) \\
M[p_1] := 0 \\
\forall \text{ Index2 } \ p_2 \in P\ln \\
M[p_1] := M[p_1]+A[p_1\oplus p_2] \\
M[p_1] := M[p_1] / \text{size}(P\ln)
\]
∀ Index1 p1 ∈ δ(A, YDIM)
2a. Domain2 Pln :=
2b. Slice(δ(A), p1, YDIM)
3. M[p1] := 0
4. ∀ Index2 p2 ∈ Pln
5b. A[p1, p2]

1. Declare and iterate over a 1-D point, p1, to along all the points in the domain of A along the YDIM axis.
2a. Declare a 2D plane, Pln.
2b. and assign to it a slice of the domain A, taken at the point p1 along the YDIM axis.
3. Zero the array

for j, 1:NY
M(j) = sum(sum(A(:,j,:)))/(NX*NZ)

Matlab
for j, 1:NY
    M(j) = sum(sum(A(:,j,:))//(NX*NZ)

Matlab

\begin{align*}
\forall \text{ Index1 } p1 \in \delta(A,YDIM) \\
2a & \quad \text{Domain2 } P\text{ln} := \text{Slice}(\delta(A),p1,YDIM) \\
2b & \quad M[p1] := 0 \\
3 & \quad \forall \text{ Index2 } p2 \in P\text{ln} \\
5a & \quad M[p1] := M[p1] / \text{size}(P\text{ln})
\end{align*}

Pseudo-code

4. Iterate with \(p2\) in two dimensions over the points in \(P\text{ln}\).

5a. Increment the value in 1D array \(M\) at point \(p1\)

5b. by the value in array \(A\) at the point represented by combining \(p1\) and \(p2\).

6. Convert the sum into an average by dividing by the number of points in \(P\text{ln}\).
Sample Queries

- Building Blocks
  - PlanarAverage
  - Normalization

PlanarAverage
∀ Index1 $p_1 \in \delta(A, YDIM)$
  Domain2 $Pln := \text{Slice}(\delta(A), p_1, YDIM)$
  $M[p_1] := 0$
  ∀ Index2 $p_2 \in Pln$
    $M[p_1] := M[p] + A[p_1 \oplus p_2]$
    $M[p_1] := M[p_1] / \text{size}(Pln)$

Normalize
∀ Point3 $p \in \delta(A)$
  $M = \text{PlanarAverage}(A)$
Sample Queries

- **Buoyancy Flux**
  
- **Correlation between density and velocity**

Buoyancy Flux

Array1 $\rho_{avg} = \text{PlanarAverage}(\rho)$
Array1 $w_{avg} = \text{PlanarAverage}(w)$

\[ \forall \text{ Index3 } ijk \in \delta(A) \]

\[ bf[p1] += (\rho[ijk] - \rho_{avg}[ijk.y]) \times (w[ijk] - w_{avg}[ijk.y]) \]

return $\text{PlanarAverage}(bf)$
Sample Queries

- Potential Energy Transport

PETransport

Array1 pavg = PlanarAverage(p)
Array1 vavg = PlanarAverage(v)
∀ Point3 i ∈ δ(p)
    pet[i] += (p[i] - pavg[i.y])^2 * (w[i] - wavg[i.y])

pet = ∂y(pet)
return PlanarAverage(pet)
KETransport

Array1 uavg = PlanarAverage(u)
Array1 vavg = PlanarAverage(v)
Array1 wavg = PlanarAverage(w)
∀ Point3 i ∈ δ(u)
    un = u[i] - uavg[p.y]
    vn = v[i] - vavg[p.y]
    wn = w[i] - wavg[p.y]
ket[i] += (un*un + 
    vn*vn + wn*wn) * vn
ket = ∂y(ket)
return PlanarAverage(ket)
Physics Goal

- Study vortical structures
  - Identify with $\lambda_2$
- What properties differentiate them?
- Only a small part of the domain
Conditional Queries

\[ \forall \text{ Index3 } ijk \in \delta(u) \text{ where } l_2 < \epsilon \ldots / * \text{same as previously} */ \ldots \]

- Used to restrict queries
- Sparse domains
- Sparse iteration spaces

Conditional Queries

∀ Index3 ijk ∈ δ(u) where l2 ≥ ϵ

.../*same as previously*/...


• Used to restrict queries
• Sparse domains
• Sparse iteration spaces
Experiment

- Attach analysis to running simulation
- Identify relevant data
- Reduce storage costs per timestep
- Increased temporal granularity
Performance

Core Count

Duration (sec)

- Simulator without Saaz
- Simulator without Saaz + Offline Statistics in Fortran
- Simulator with Saaz

Triton at SDSC
Dual-socket quad-core
Intel E5530 (Nehalem, 2.40 GHz)
24 GB RAM
Offline Analysis

- Want interactive updates
- No simulation as the bottleneck
- Current Work
  - Building the Tettnanger compiler to optimize
- Have several users
  - First users at UCSD and KTH
  - Library not yet released yet
  - Contact: apking@cs.ucsd.edu
Acknowledgements

• Acknowledgements
  • Co-authors: Eric Arobone, Scott Baden, Sutanu Sarkar
  • NSF contract OCE-0835839
  • Triton system
    • San Diego Supercomputer Center
    • University of California, San Diego.

• Contact
  • http://cseweb.ucsd.edu/~apking/research
  • apking@cs.ucsd.edu
Fin