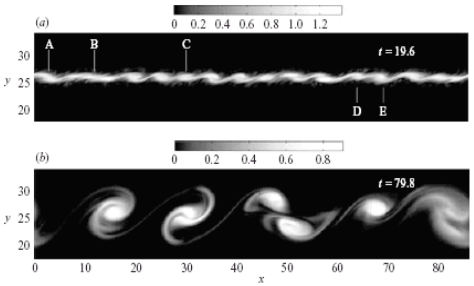
A Query Language for a Computational Database

Alden King; Scott Baden, Advisor



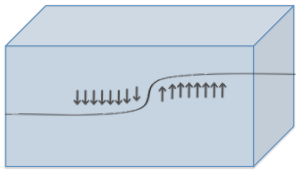
Horizontal spanwise vorticity (mag) using DNS, [Basak & Sarkar, JFM 2006]

What is a computational database?

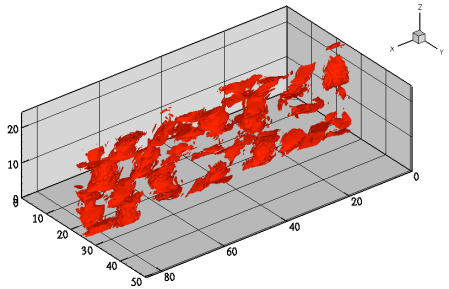
- Computational science derives knowledge from large datasets.
- Relational databases are not appropriate for scientific data. Particularly, the relational database model doesn't easily:
  - Express arbitrary aggregation functions
  - Preserve locality (stencil operators require this)
- Computational database for data analysis
  - Non-relational query language which combines queries with computation
  - Generate source code for optimized programs to do the analysis. Target the appropriate architecture, such as host CPUs, or compute cards like GPUs.

Identifying Vortices in Horizontal Sheers

With Sutanu Sarkar,
Hieu Pham, and Eric Arobone



Adjacent regions of water slide past each other at different velocities



Vortices appear where delta is larger than a threshold value, ε

Identification of vortices is done via the "delta-criterion": `delta > ε`.
We then want to evaluate a function over these points. This requires conditional execution:

```
[ delta(0,0,0) = … ] nx,ny,nz;
[ /* compute f */ ] nx,ny,nz where delta > ε
```

We can use this to optimize for `delta` being sparse and for caching based on epsilon.
This is equivalent to the C version:

```
for (int i = 0; i < nx; ++i)
    for (int j = 0; j < ny; ++j)
        for (int k = 0; k < nz; ++k)
            if ( f(i, j, k) > delta )
                /* compute f */
```

or the CUDA version:

```
__global__ void ComputeF_kernel( Mat* f, Mat* delta, float epsilon, … ) {
    if ( f(threadIdx.x, threadIdx.y, threadIdx.z) > delta )
        /* compute f */
}

/* in main */
    ComputeDelta_kernel<<< 1, dim3(nx,ny,nz) >>>(delta, …);
    ComputeF_kernel<<< 1, dim3(nx,ny,nz) >>>(f, delta, epsilon, …);
```

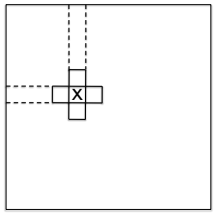
Computational Query Language

Finite-Difference method for the Laplace equation

```
[ u'(0,0,0) =  c * ( u(-1,0,0) + u(+1,0,0)
                   + u(0,-1,0) + u(0,+1,0)
                   + u(0,0,-1) + u(0,0,+1) )
] nx,ny,nz
```

We want to be able to express common computational operations (like stencils, right) easily and simply. So we have built-in primitives like matrices, ranges, and foreach loops.

We also want to be able to operate on a subset of data, so we allow conditional execution (see box at left).



Why a new language?

We use generate source code for different targets, and they have different models for which we want to optimize. We can choose the appropriate representations of primitives automatically according to the target and the data itself. Generating source code lets us take advantage of existing compiler optimizations, as well as implement hand-tuning if necessary.

	Matrix	Loop
C	Flattened array (dense); Sparse-matrix	For-loop
CUDA	Flattened array (small, dense); Blocked array (dense); Sparse-matrix	Kernel function and method call

Support: NSF (OCE-0835839)