GPU Implementation of Vortex Method for Simulating Unsteady Flows

Danny Sale
University of Washington
ME 599 – Texel Modeling
Mar. 16, 2013

Outline

- Background on Vortex Methods
  - Theory + Application to Vortex Rings
- Benchmarking
  - Speedups Using GPU
- Preliminary Results
  - Leapfrogging Vortex Rings
- Using CUDA on Linux
  - Lessons Learned

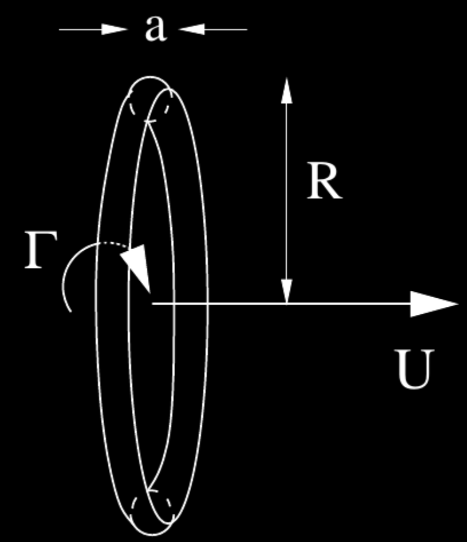
Background

Why vortex methods? why GPU?

- vortex methods have potential for efficiency due to compact vorticity support & parallelism
- “mesh-free” particle based methods suitable for fluid-structure interaction
- Significant speedups when combining advanced hardware & algorithms

Project Introduction

- ***Vortex Ring Dynamics***
  - Classical problem, simple
  - Plenty of literature for comparisons
  - Fun to watch!



Vortex-in-Cell Algorithm

- Governing eqns.

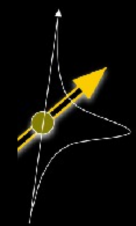
$$\frac{\partial \boldsymbol{\omega}}{\partial t} + (\mathbf{u} \cdot \nabla)\boldsymbol{\omega} = (\boldsymbol{\omega} \cdot \nabla)\mathbf{u} + \nu \nabla^2 \boldsymbol{\omega}$$

$$\boldsymbol{\omega} = \nabla \times \mathbf{u}$$

- Discretized w/ particles that transport vorticity

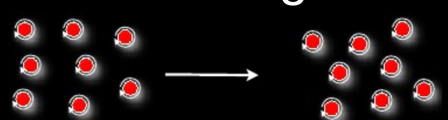
$$\boldsymbol{\alpha}_p = \int_{V_p} \boldsymbol{\omega} d\mathbf{x}$$

$$\boldsymbol{\omega}(\boldsymbol{x}, t) \approx \sum_p \boldsymbol{\alpha}_p(t) \zeta^h \left(\mathbf{x} - \mathbf{x}_p(t)\right)$$



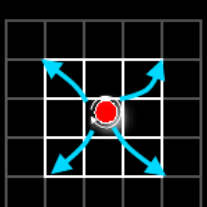
- Particle trajectories & strengths

$$\frac{d\mathbf{x}_p}{dt} = \mathbf{u}(\mathbf{x}_p)$$

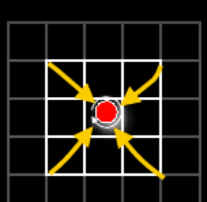


$$\frac{d\boldsymbol{\alpha}_p}{dt} = \int_{V_p} (\boldsymbol{\omega} \cdot \nabla)\mathbf{u} + \nu \nabla^2 \boldsymbol{\omega} d\mathbf{x}\,, \quad \simeq \left((\boldsymbol{\omega} \cdot \nabla)\mathbf{u}(\mathbf{x}_p) + \nu \nabla^2 \boldsymbol{\omega}(\mathbf{x}_p)\right) V_p$$

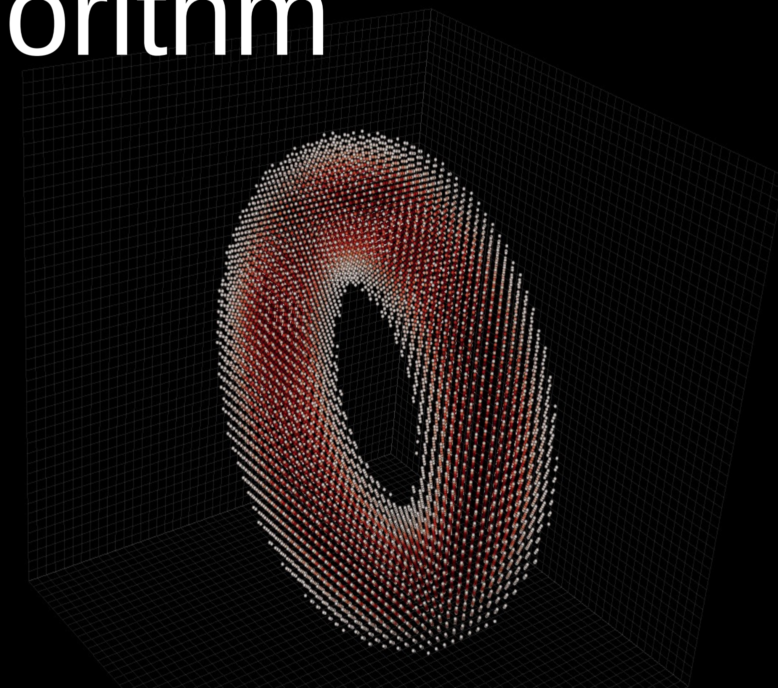
- Particles & mesh communicate w/ interpolation

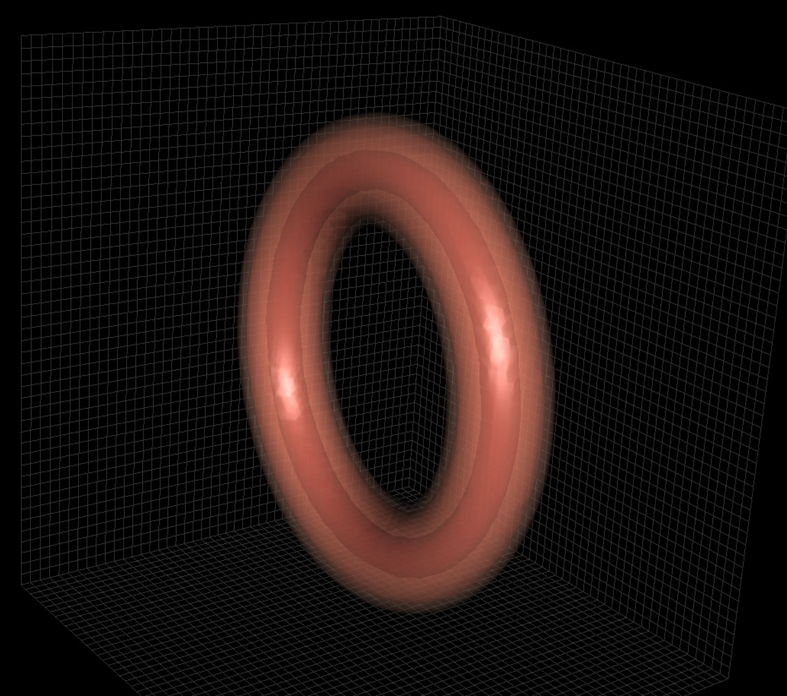


$$\boldsymbol{\alpha}_p \rightarrow \boldsymbol{\omega}_{ij}$$



$$\begin{matrix} \mathbf{u}_{ij} \\ (\boldsymbol{\omega} \cdot \nabla)\mathbf{u}_{ij} \\ \Delta \boldsymbol{\omega}_{ij} \end{matrix} \rightarrow \begin{matrix} \mathbf{u}_p \\ (\boldsymbol{\omega} \cdot \nabla)\mathbf{u}_p \\ \Delta \boldsymbol{\omega}_p \end{matrix}$$

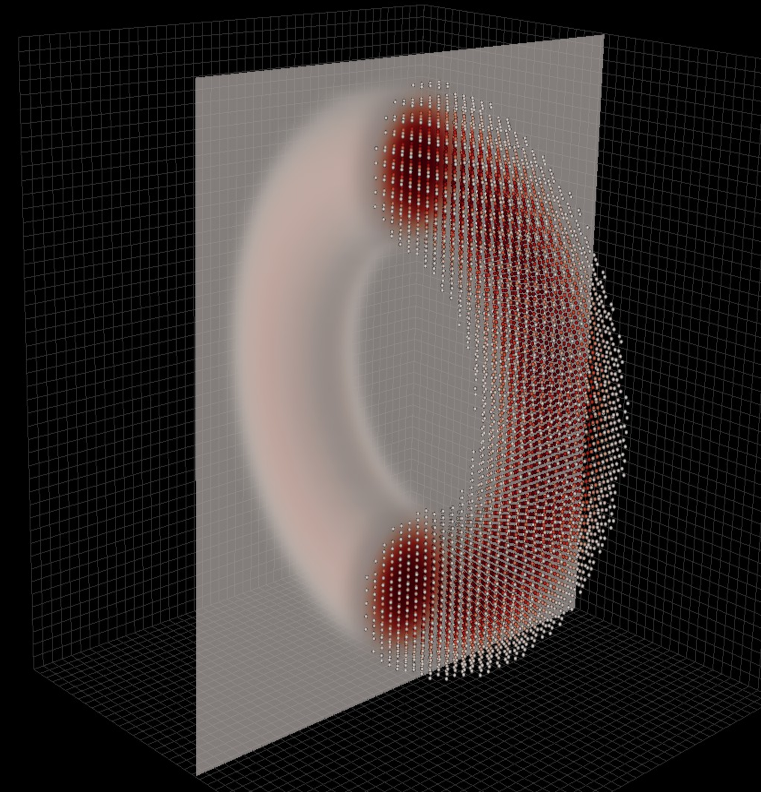


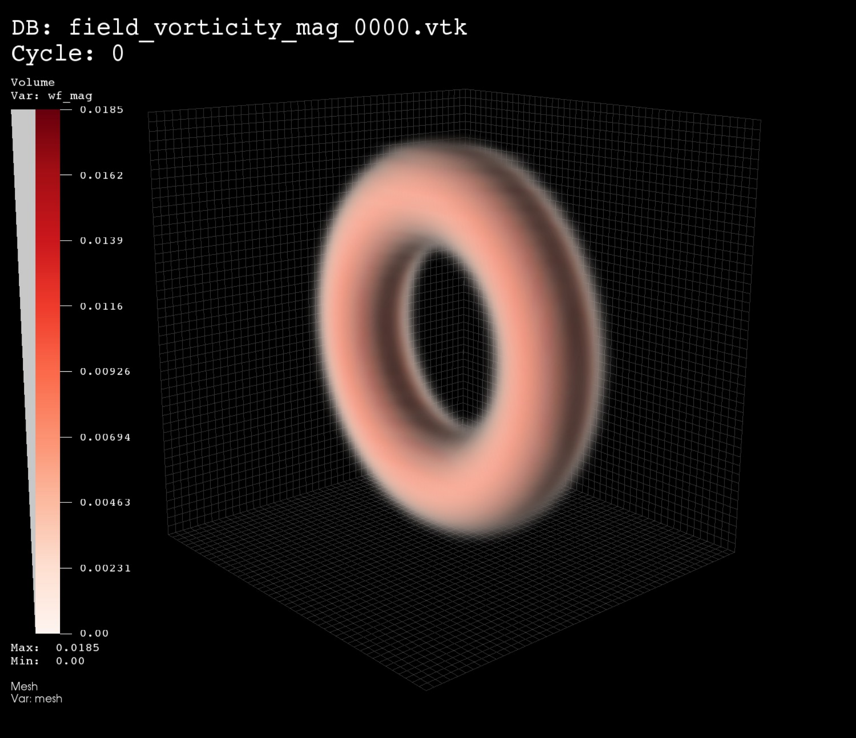


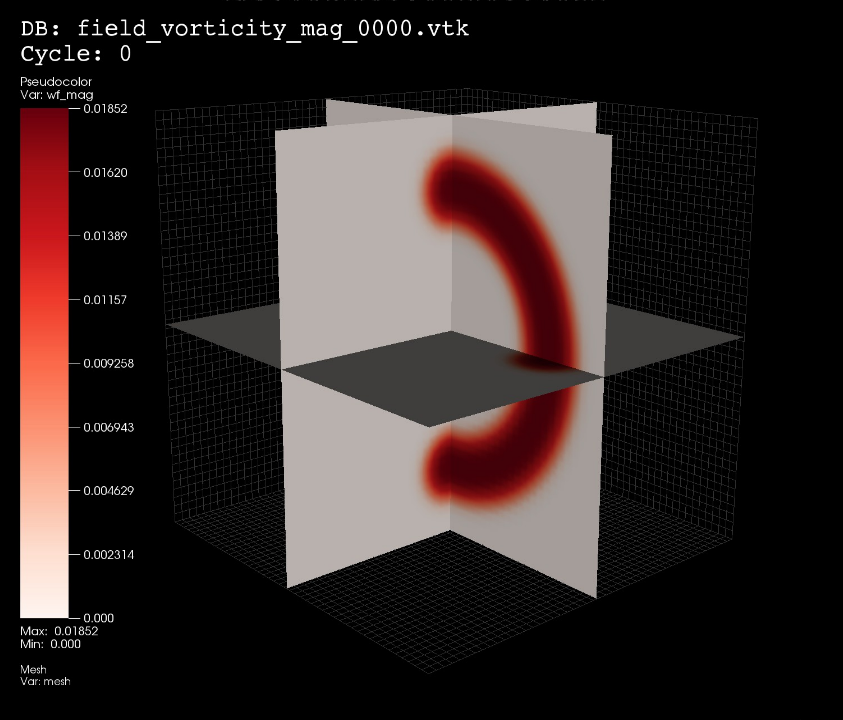
Preliminary results

VIC code in ***Matlab w/ Parallel Toolbox***

- initiates vortex particles & field
- Biot-Savart velocity solver
- writes *.vtk files for post-processing
- used for algorithm prototyping & benchmarking







Using the GPU w/ Matlab

arrayfun

Apply function to each element of array on GPU

Syntax

```
A = arrayfun(FUN, B)
A = arrayfun(FUN, B, C, ...)
[A, B, ...] = arrayfun(FUN, C, ...)
```

`A = arrayfun(FUN, B)` applies the function specified by `FUN` to each element of the gpuArray `B`, and returns the results in gpuArray `A`. `A` is the same size as `B`, and `A(i,j,...)` is equal to `FUN(B(i,j,...))`. `FUN` is a function handle to a function that takes one input argument and returns a scalar value. `FUN` must return values of the same class each time it is called. The input data must be an array of one of the following types: numeric, logical, or gpuArray. The order in which `arrayfun` computes elements of `A` is not specified and should not be relied on.

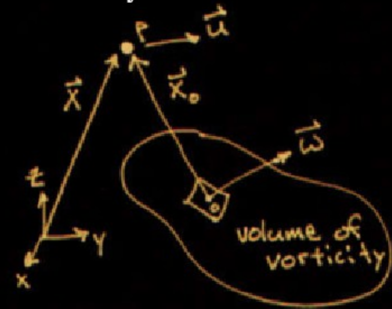
PC Hardware

Host:	Intel Core i5-3570K 4 cores @ 3.4GHz 16 GiB DDR3
Device:	GeForce GTX 680 1536 CUDA cores @ 1006 MHz 4 GiB GDDR5 CUDA Compute 3.0

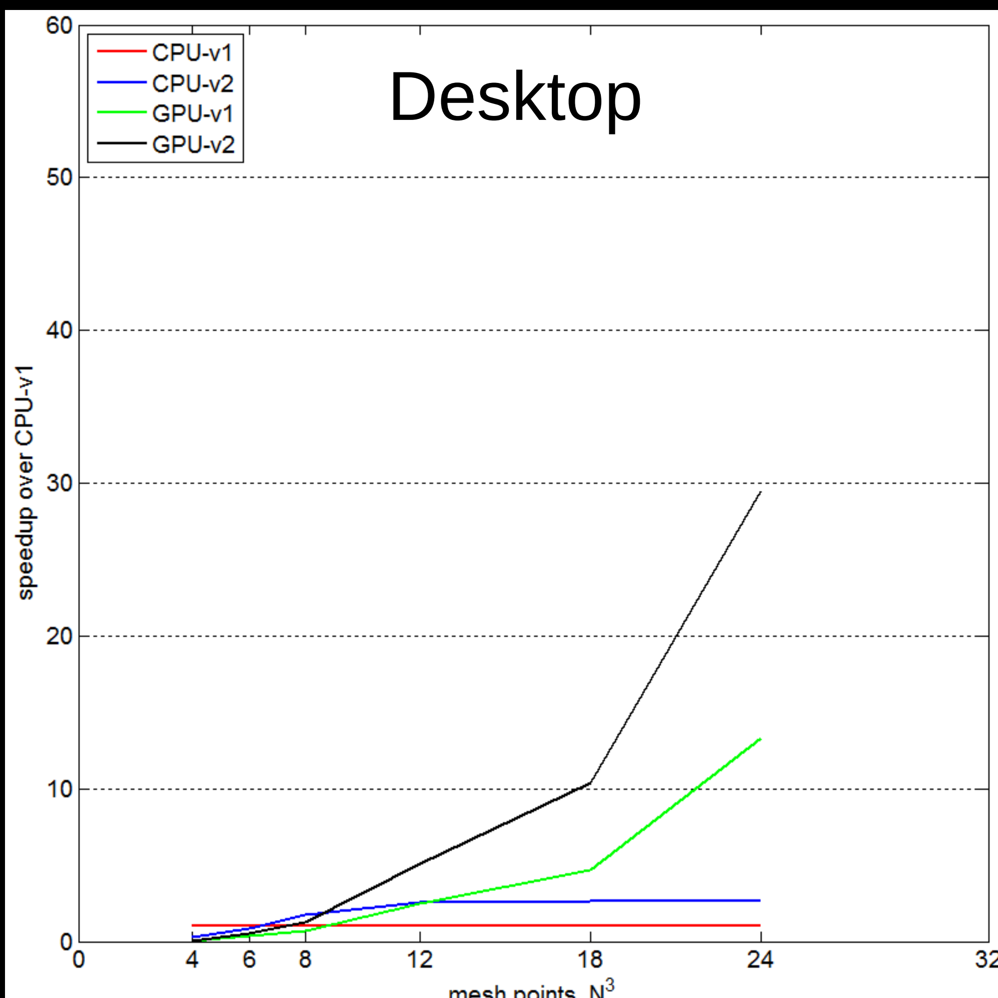
Host:	Intel Core i7-3612QM 4 cores @ 2.1GHz 8 GiB DDR3
Device:	GeForce GTX 650M 384 CUDA cores @ 900 MHz 2 GiB GDDR5 CUDA Compute 3.0

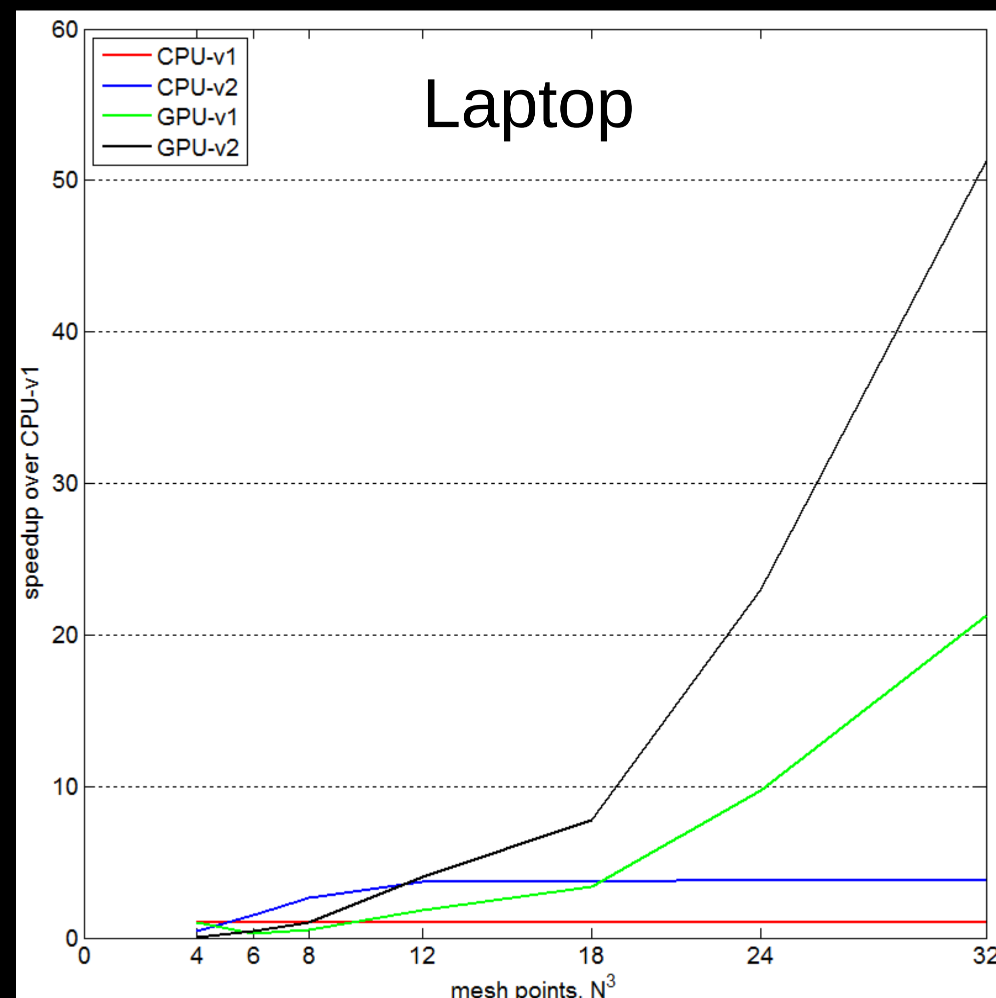
Acceleration of Biot-Savart law

$$\vec{u}_\omega(\vec{x},t) = \int_V -\frac{1}{4\pi}\frac{(\vec{x}-\vec{x}_o)}{|\vec{x}-\vec{x}_o|^3} \times \vec{\omega}(\vec{x}_o,t)\,d\vec{x}_o = \left(\vec{K}(\vec{x}-\vec{x}_o)\times\right) \star \vec{\omega}(\vec{x}_o,t)$$



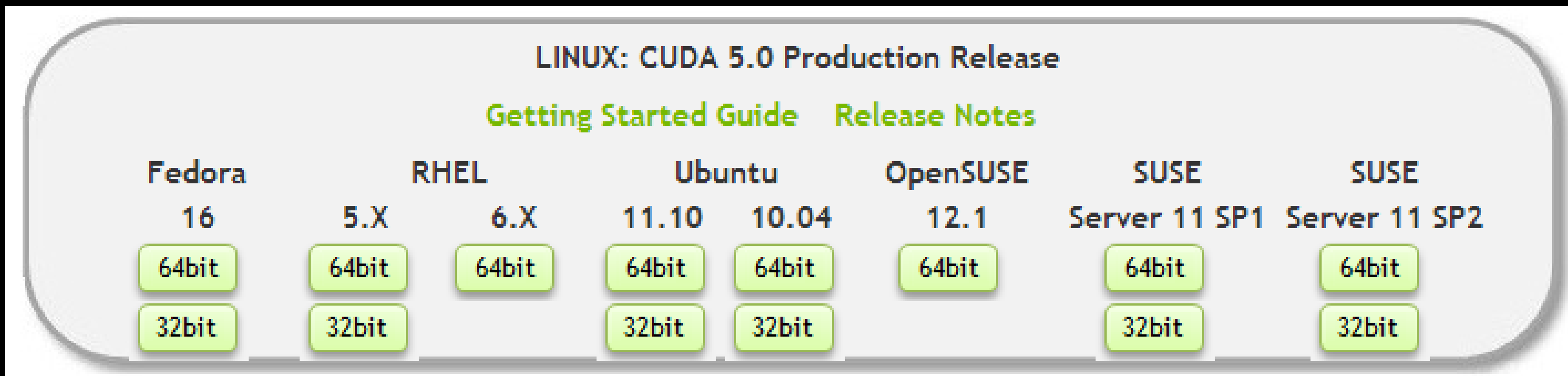
$$\vec{u}_\omega(\vec{x},t) = \sum_p \vec{K}\left(\vec{x}-\vec{x}_p(t)\right) \times \vec{\alpha}_p(t)$$





show VisIt simulation

CUDA on Linux



Possible to install CUDA 5 on unsupported distributions

- Tested on openSUSE 12.2 and 12.3 (comes with gcc 4.7+)
- CUDA depends on older gcc compiler (4.3 and 4.4)
- Pass compiler flag to nvcc to identify gcc version (see release notes)

Moving Forward

- **“stretch goals” - true VIC code**
  - Port Matlab code to C/C++ or Fortran
  - Interpolation between Eulerian & Lagrangian frames (interpP2M, interpM2P)
  - FFT-based Poisson Solver, $\nabla^2 \mathbf{u} = -\nabla \times \boldsymbol{\omega}$ (lib cuFFT, other?)
  - Other B.C. (no slip, inlet / outlet), synthetic inflow turbulence
    → model wind turbines?