Using GPUs to Track Changes in the Sun

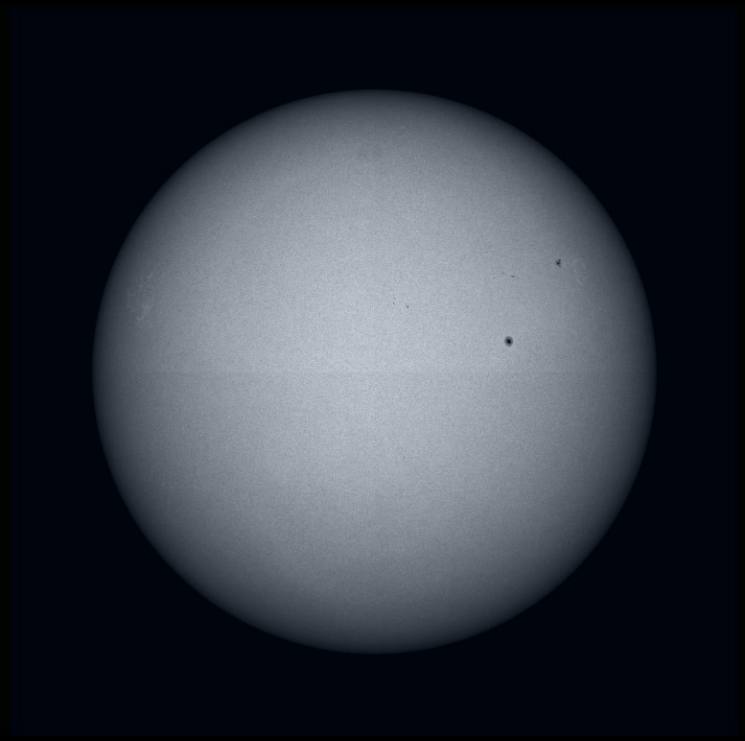
Mark Cheung[1] with contributions by
Ralph Seguin[1] & Brian Harker-Lundberg[2]

1. Lockheed Martin Solar & Astrophysics Observatory, Palo Alto, CA
2. National Solar Observatory, Tuscon, AZ

Outline

- Introducing the Sun as a “living” astrophysical object
- NASA’s Solar Dynamics Observatory
  - Science goals and instruments
  - A Visual Tour of the Sun presented by Ralph Seguin, LMSAL
- GPUs techniques for astrophysical studies of the Sun
  - Using GPUs to Measure the Sun’s Magnetic Field by Brian Harker-Lundberg, National Solar Observatory
  - Ray-tracing through Magnetohydrodynamics (MHD) models of the Solar Atmosphere by Mark Cheung, LMSAL
  - Tracking Plasma Flows by Mark Cheung, LMSAL

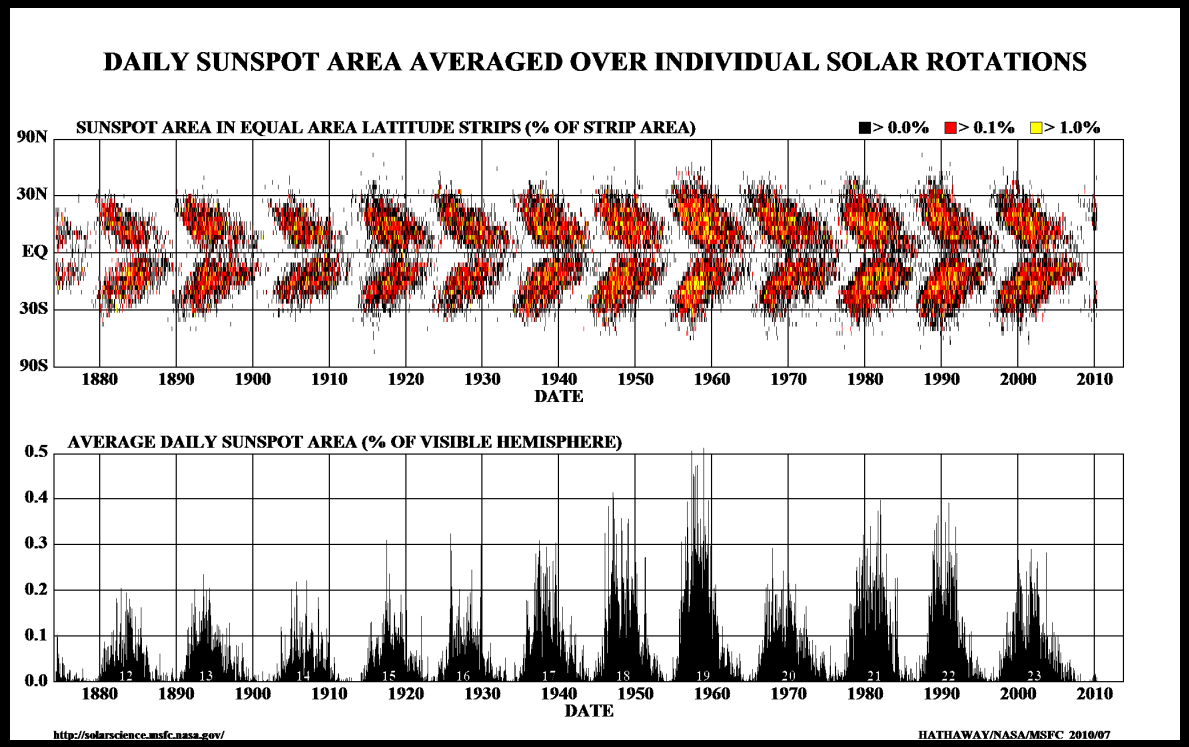
Just a Sphere of Gas?



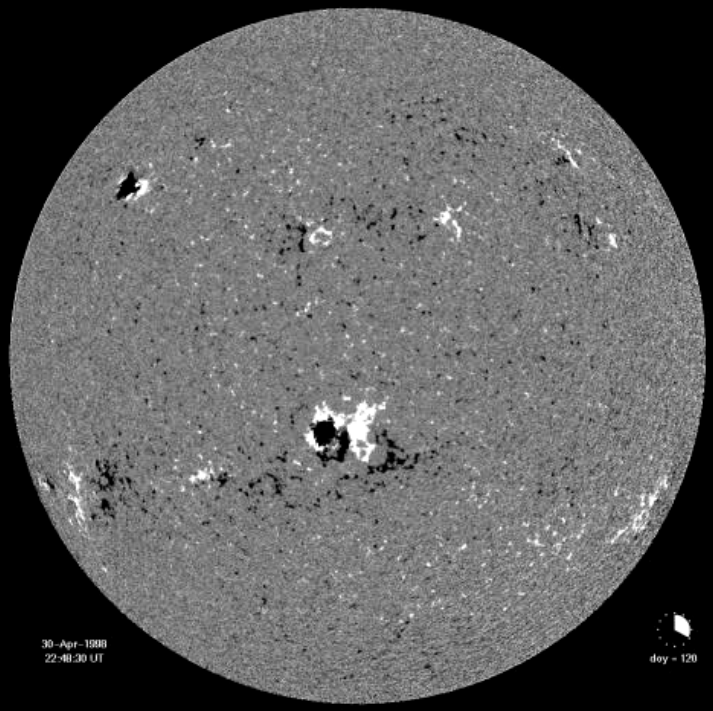
Fun facts about the Sun

- The Sun evolves on a wide range of time-scales
  - 11-year sunspot cycle / 22-year magnetic cycle
  - The Sun rotates once approximately every 27 days
  - 5-10 minute lifetime of convective granules on the solar surface
- The Sun's atmosphere and the heliosphere are pervaded by magnetic fields.
- Solar eruptions from Active Regions (usually consisting of sunspots) can affect the conditions in the near-terrestrial interplanetary medium.
- NASA's Heliophysics Observatory is a fleet of satellites that monitor the Sun's latest moves and its effects on the Earth's atmosphere.

Butterfly Diagram

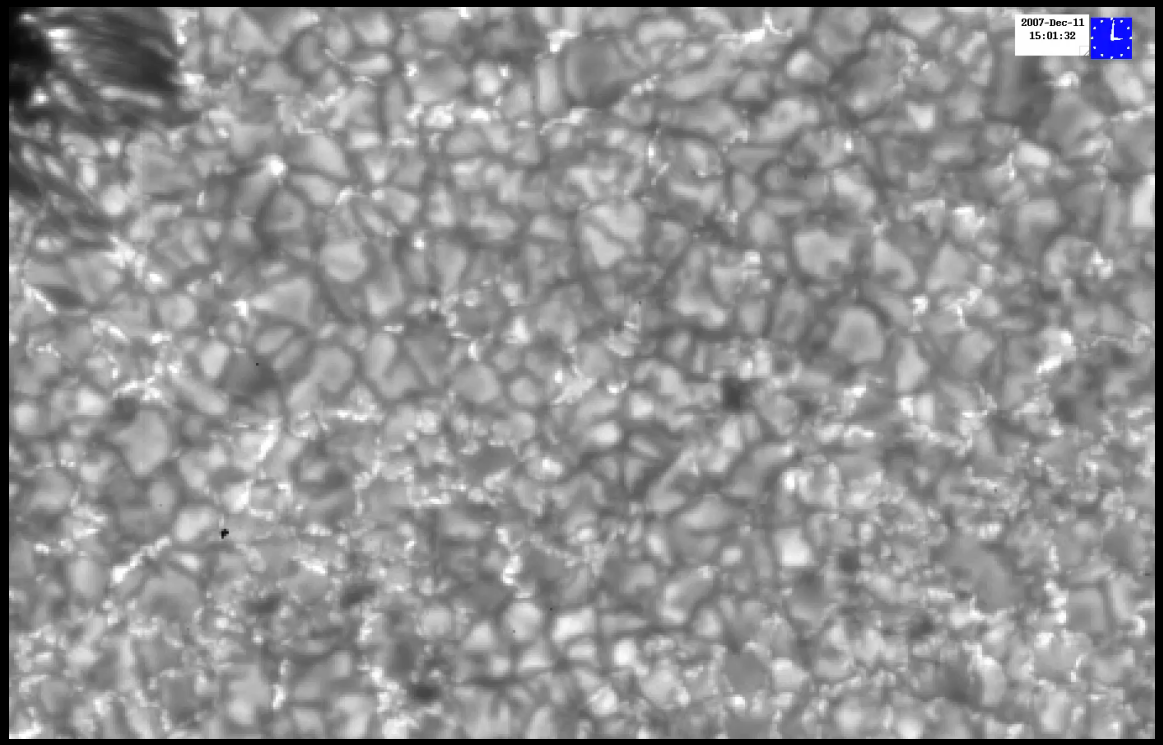


Magnetic Surface of the Sun



The Sun's Tumultuous Terrain

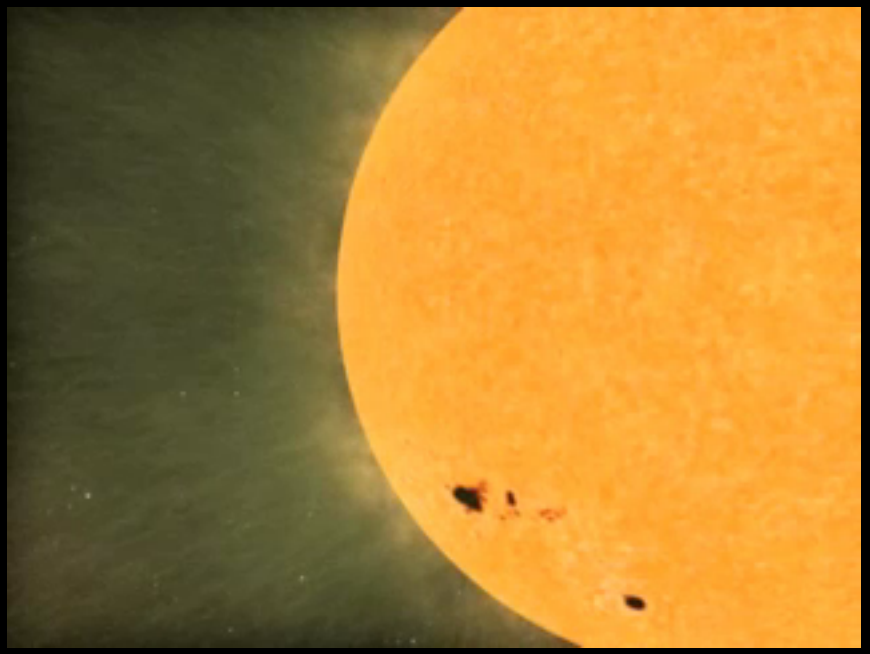
Hinode observations at 430 nm



Fun facts about the Sun

- The Sun evolves on a wide range of time-scales
  - 11-year sunspot cycle / 22-year magnetic cycle
  - The Sun rotates once approximately every 27 days
  - 5-10 minute lifetime of convective granules on the solar surface
- The Sun's atmosphere and the heliosphere are pervaded by magnetic fields.
- Solar eruptions from Active Regions (usually consisting of sunspots) can affect the conditions in the near-terrestrial interplanetary medium.
- NASA's Heliophysics Observatory is a fleet of satellites that monitor the Sun's latest moves and its effects on the Earth's atmosphere.

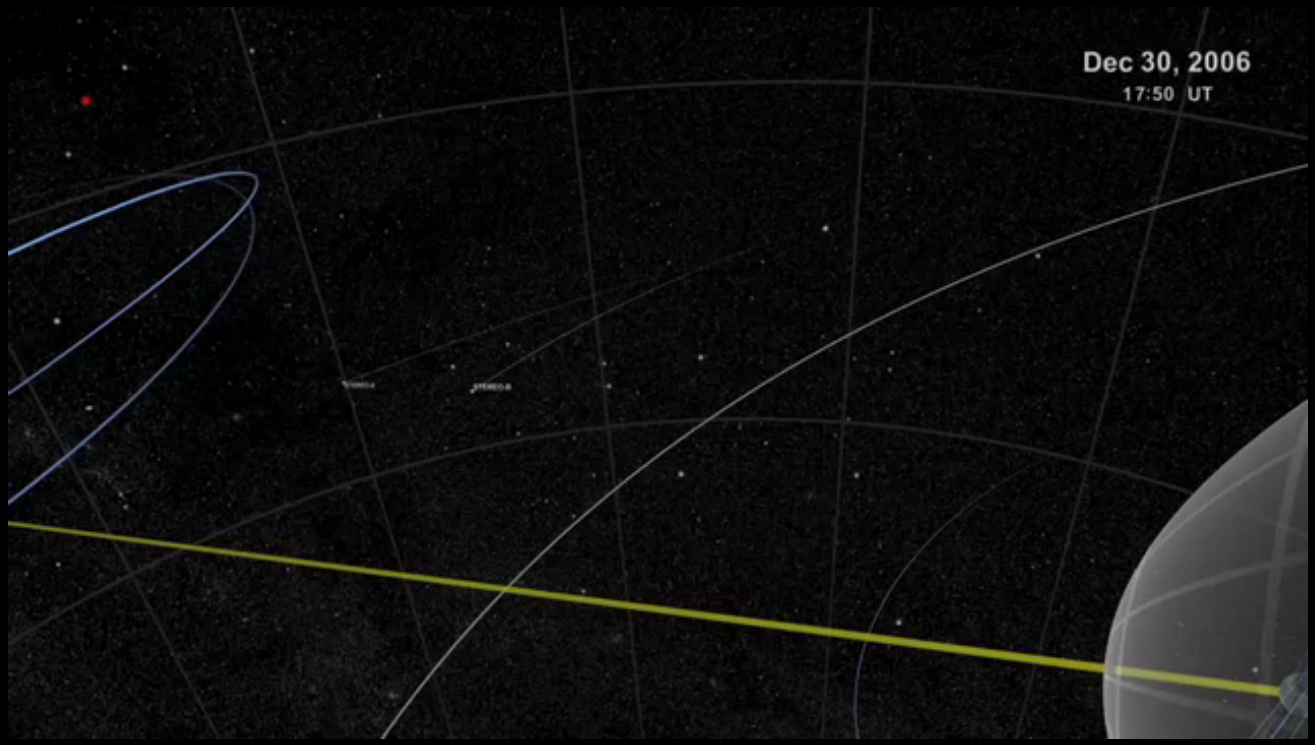
Solar Eruption Impacting Earth



Fun facts about the Sun

- The Sun evolves on a wide range of time-scales
  - 11-year sunspot cycle / 22-year magnetic cycle
  - The Sun rotates once approximately every 27 days
  - 5-10 minute lifetime of convective granules on the solar surface
- The Sun's atmosphere and the heliosphere are pervaded by magnetic fields.
- Solar eruptions from Active Regions (usually consisting of sunspots) can affect the conditions in the near-terrestrial interplanetary medium.
- NASA's Heliophysics Observatory is a fleet of satellites that monitor the Sun's latest moves and its effects on the Earth's atmosphere.

The Heliophysics Observatory

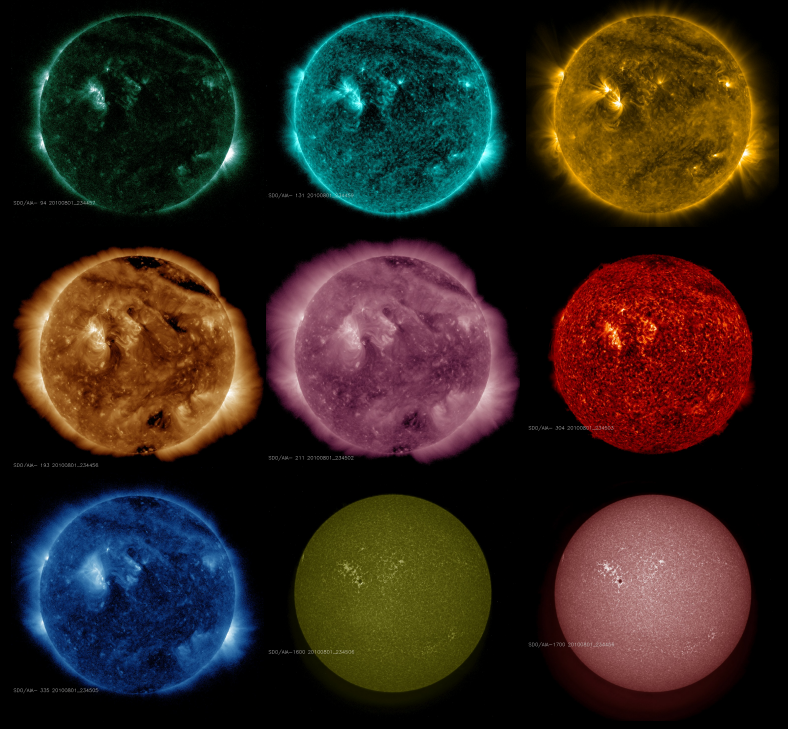


Solar Dynamics Observatory

- First mission of NASA's Living With a Star program
- Science goal
  - Understand the solar dynamo
  - Understand the physical drivers on the Sun that affect space weather
- Science Instruments
  - Atmospheric Imaging Assembly (AIA) – next slide
  - Helioseismic Magnetic Imager (HMI) – Principal Investigator: Prof. Phillip Scherrer at Stanford University but the instrument was built at our labs in Palo Alto, CA.
  - EUV Variability Experiment (EVE) – Principal Investigator: Dr. Tom Woods at Laboratory for Atmospheric and Space Physics, Boulder, CO.

Atmospheric Imaging Assembly

- Four telescopes, each delivering
  - Full-disk images of the Sun at 4096 x 4096 resolution
  - Cadence of 1 image / channel / 12 s
  - 7 EUV channels
  - 2 UV channels
  - 1 visible channel at 4500 Å
  - Temperature coverage from 6000 K to 10 million K.
- Instrument designed and built at our labs in Palo Alto, CA.



A Visual Tour of the Sun

Ralph Seguin
Lockheed Martin Solar & Astrophysics Laboratory, Palo Alto, CA

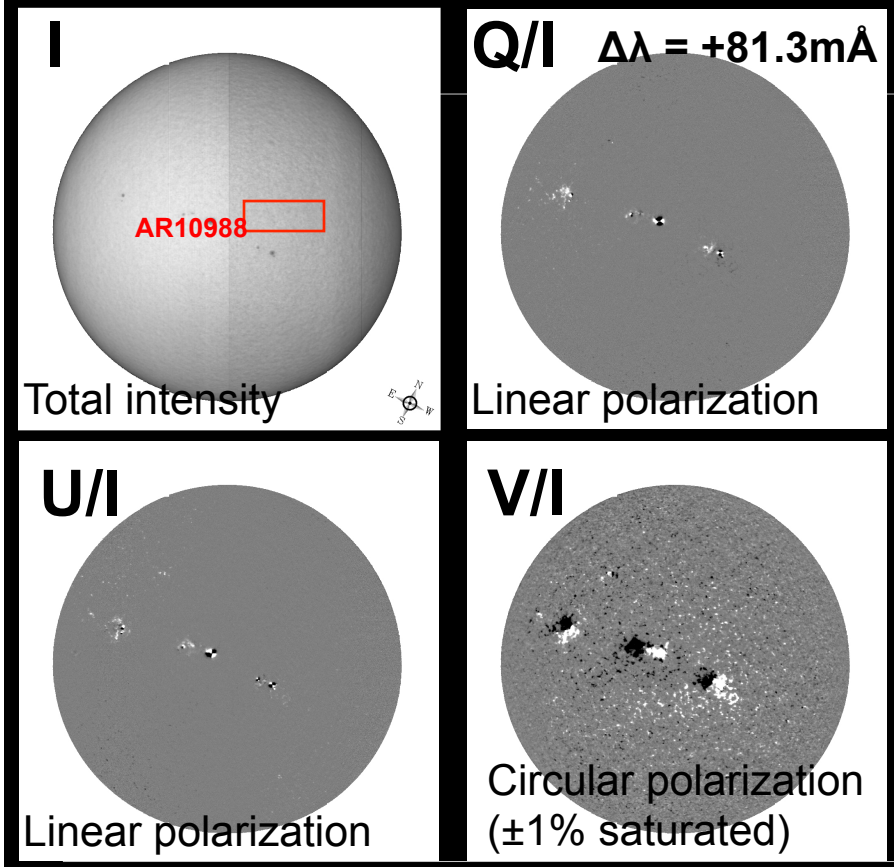
A Visual Tour of the Sun

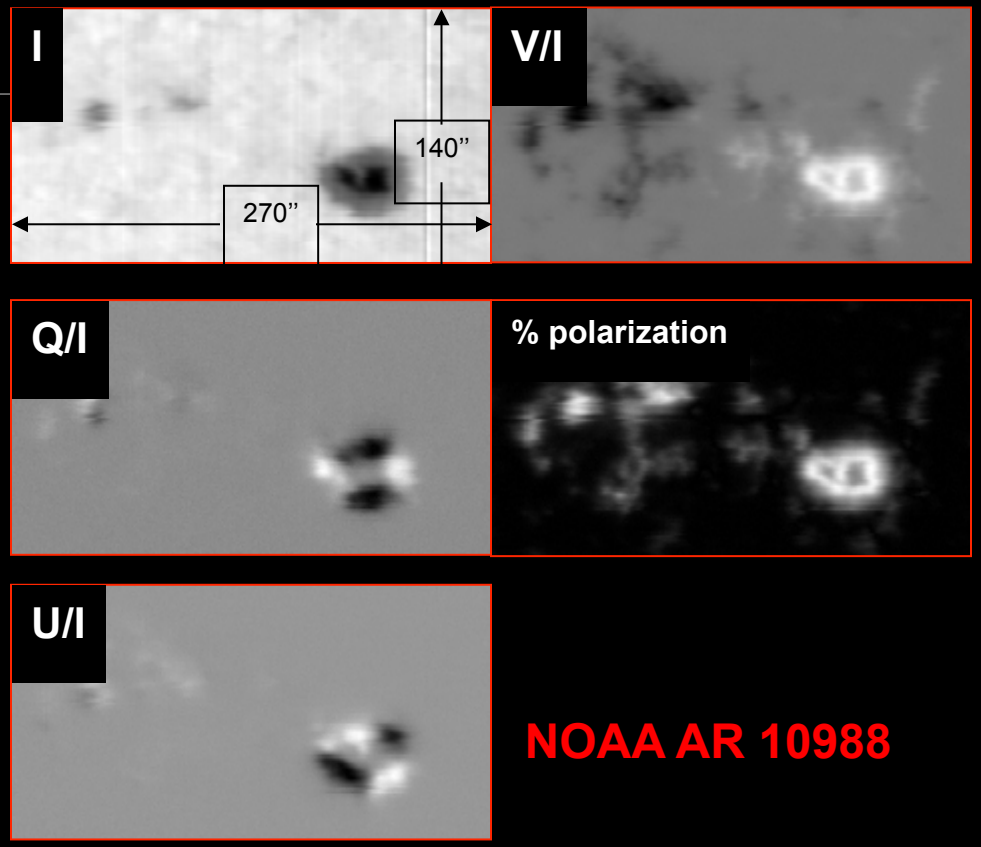
- Panorama is a data exploration and visualization tool
  - Written in OpenGL and GLSL
  - Uses fragment shaders to enable novel visualization of high resolution data sets.
    - Blending and tiled views of various AIA channels
    - Running difference movies
  - Lessons learned
    - Working with different with disparate data formats
    - Naming conventions amongst shader variables
- Future plans
  - Start using OpenCL for computations, e.g. inversions from AIA signals to plasma temperatures.

Using GPUs to Measure the Sun's Magnetic Field

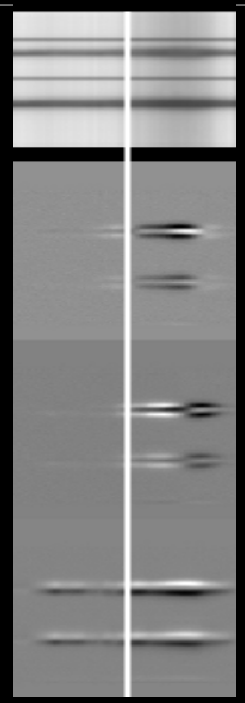
Brian Harker-Lundberg
National Solar Observatory, Tuscon, AZ

SOLIS Vector Spectromagnetograph (VSM) observations on 28 March 2008 15:45 UTC



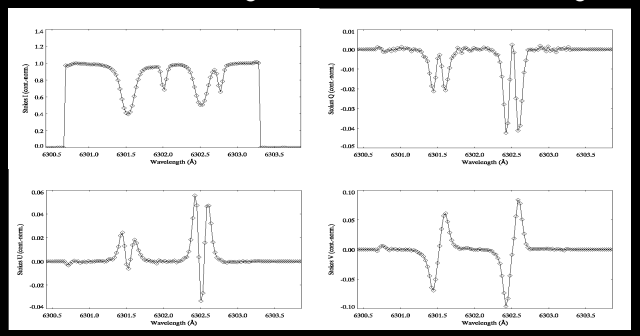


SOLIS Vector Spectromagnetograph (VSM) observations on 28 March 2008 15:45 UTC



- *(Left)* Excerpts from SOLIS VSM scanline spectra. The full spectra contain 2048 spatial elements horizontally by 128 spectral elements vertically.
  - Spectra obtained in ~ 3Å bandpass around multiplet 816 (Fe I λλ6301.5, 6302.5)
- *(Below)* Stokes profiles obtained by the vertical slice across the spectra to the left.

Stokes I/I_c **Stokes Q/I_c**



Stokes U/I_c **Stokes V/I_c**

- We infer properties of vector magnetic fields by fitting to these profiles functions of the Voigt and Faraday-Voigt profiles
- "Stokes Inversion"

Inversion Method

- Voigt/Faraday-Voigt- and subsequent Stokes-synthesis routines have high arithmetic intensity (rational function and polynomial approximations, straightforward analytic expressions)

$$V(a,v) = \frac{a}{2\pi}\int_{-\infty}^{\infty}\frac{e^{-y^2}}{(v-y)^2+a^2}dy$$

$$F(a,v) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\frac{(v-y)e^{-y^2}}{(v-y)^2+a^2}dy$$

- Efficient techniques exist for calculating these elementary profiles (e.g. Hui et al. (1978))

- Eschew traditional, iterative, gradient-based optimization for more global approach…
  - Genetic Algorithms (GA): population-based stochastic search and optimization algorithms which mimic biological evolution to evolve an optimal solution to the problem from a pool of *potential* solutions, encoded as binary strings.
  - Implementation: each Block dedicated to evaluation of single population-member, each Thread dedicated to computation of synthetic Stokes profiles at a single wavelength.
  - Benefits: insensitive to initial conditions, insensitive to local (false) optimizers, embarassingly parallel, uniquely-suited to take advantage of parallel-processing on CUDA-capable GPUs.

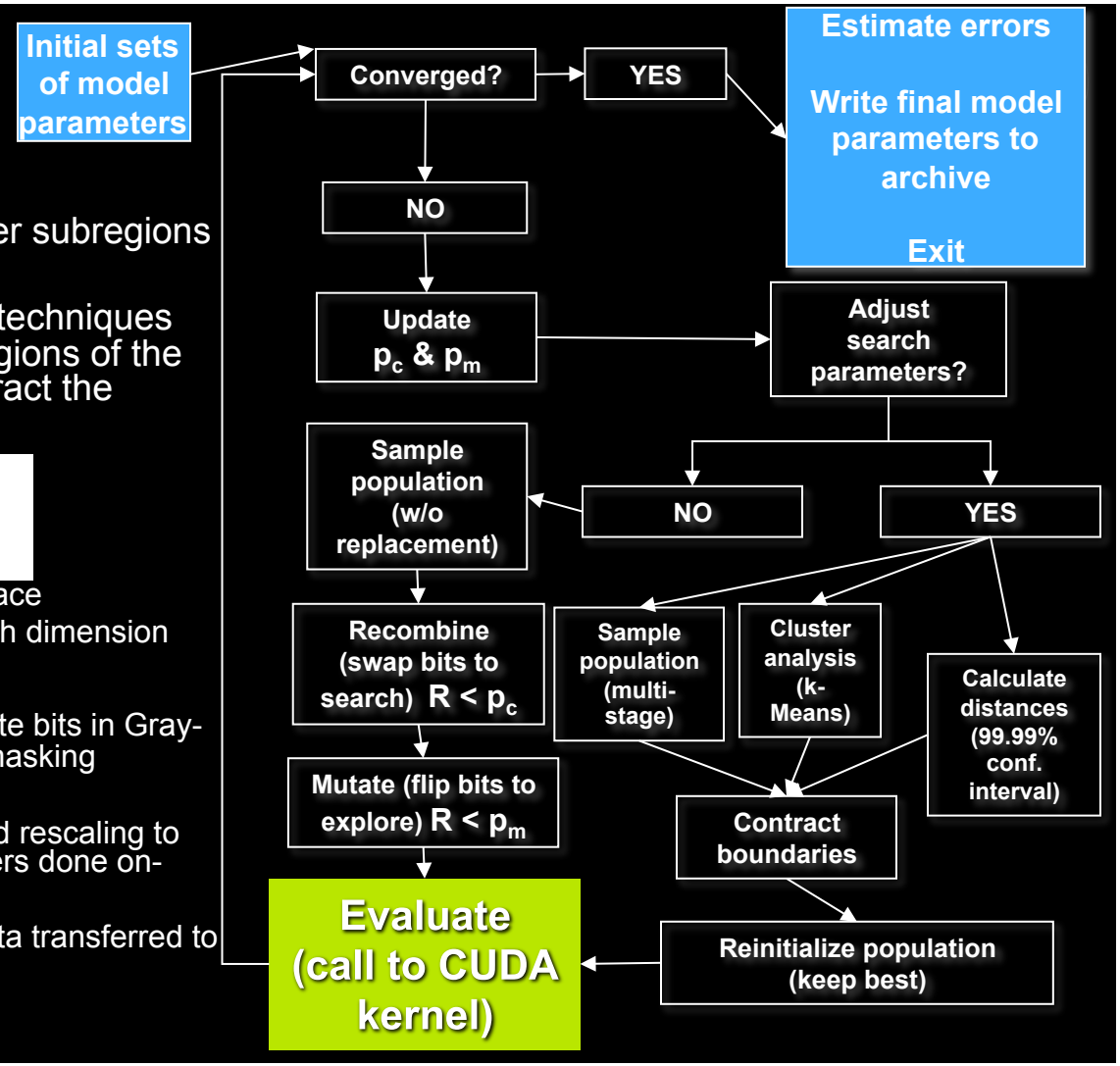
- Fractal-like GA:
  - Searches in successively-smaller subregions of the parameter space.
  - Intelligent; population sampling techniques ensure we're not losing good regions of the parameter space when we contract the feasible region.

$$\overbrace{\underbrace{1000010101001001}_{N\ bits}\cdots\underbrace{1010010100100101}_{N\ bits}}^{M\times N\ bits},$$

M = dimension of parameter space

N = number of bits used to represent each dimension

  - Integer/binary encoding used:
    - Genetic operators manipulate bits in Gray-coded integers via fast bit-masking techniques.
    - Gray-to-binary decoding and rescaling to real-valued model parameters done on-device.
    - Minimizes the amount of data transferred to device



- GPU-accelerated χ^2 calculation
  - Decoding
  - Rescaling
  - Synthesis
  - Summation
  - Return

```
const unsigned int bid = blockIdx.x;
const unsigned int tid = threadIdx.x;

if ( tid < NDIM ) {

        // Load the population of model vectors from
        // device to shared memory for this block.
        *(pop_s+ bid*NDIM + tid) = *(pop_d+ bid*NDIM + tid);

        // Decode the model vectors from the reflected Gray binary
        // representation to standard binary.
        *(pop_s+ bid*NDIM + tid) ^= *(pop_s+ bid*NDIM + tid) >> 1;
        *(pop_s+ bid*NDIM + tid) ^= *(pop_s+ bid*NDIM + tid) >> 2;
        *(pop_s+ bid*NDIM + tid) ^= *(pop_s+ bid*NDIM + tid) >> 4;
        *(pop_s+ bid*NDIM + tid) ^= *(pop_s+ bid*NDIM + tid) >> 8;

}
__syncthreads();

// Rescale to real-valued synthetic model parameters for each thread.
for ( unsigned int i=0; i<NDIM; i++ ) {
        *(synModel+ tid*NDIM + i) = min[i] + *(pop_s+ bid*NDIM + i)*(max[i]-min[i])/norm;
}

// Calculate synthetic Stokes profiles at unique wavelength.
CudaStokes( &synModel[tid*NDIM], &synStokes[tid*NSTOKES] );

// Sum deviations from observed Stokes profiles (in constant memory).
chisq[tid] = 0.0;
for ( unsigned int i=0; i<NSTOKES; i++ ) {
        dev = weight[i]*(obsData[tid*NSTOKES + i] - synStokes[tid*NSTOKES + i]);
        chisq[tid] += dev*dev;
}
__syncthreads();

// Do parallel reduction in shared memory to get total
// chi-squared value.
for ( unsigned int i=blockDim.x/2; i>0; i>>=1 ) {
        if ( tid < i ) {
                chisq[tid] += chisq[tid + i];
        }
        __syncthreads();
}

// thread-0 loads result from shared to global memory
if ( tid == 0 ) *(f_d+ bid) = chisq[0];
__syncthreads();
return;
```

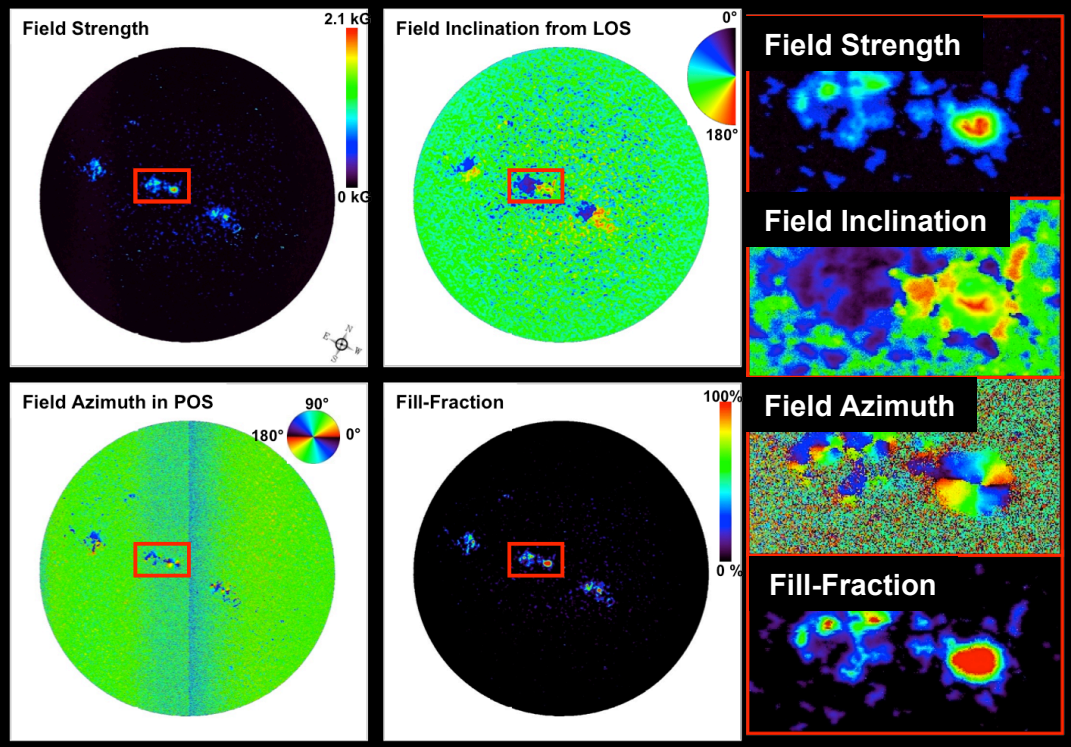
- Excerpt from Stokes synthesis routine (CudaStokes)
  - Straightforward forward modeling done in shared memory
    - No divergent threads outside Voigt synthesis
  - Thread-indexed variables aligned to avoid bank conflicts:
    - At least 32 wavelengths span spectral lines of interest, can use 64 or all 128 observed wavelengths.
    - “Nice numbers” for CUDA

```
/**
 * MORE COMMON TERMS
**/
eta_sum[tid] = eta_left[tid] + eta_right[tid];
rho_sum[tid] = rho_left[tid] + rho_right[tid];

factor2[tid] = (eta_pi[tid] - 0.5f*eta_sum[tid])*factor1[tid];
factor3[tid] = (rho_pi[tid] - 0.5f*rho_sum[tid])*factor1[tid];

/**
 * ELEMENTS OF ATTENUATION MATRIX...[ABSORPTION]...
**/
Eta_I[tid] = 1.f + 0.25f*eta_sum[tid]*(1.f + cos_inc[tid]*cos_inc[tid]) +
                    eta_pi[tid]*factor1[tid];
Eta_Q[tid] = factor2[tid]*cos_twoaz[tid];
Eta_U[tid] = factor2[tid]*sin_twoaz[tid];
Eta_V[tid] = (eta_left[tid] - eta_right[tid])*factor0[tid];

Eta_I_squared[tid] = Eta_I[tid]*Eta_I[tid];
Eta_Q_squared[tid] = Eta_Q[tid]*Eta_Q[tid];
Eta_U_squared[tid] = Eta_U[tid]*Eta_U[tid];
Eta_V_squared[tid] = Eta_V[tid]*Eta_V[tid];

/**
 * ELEMENTS OF ATTENUATION MATRIX...[ANOMALOUS DISPERSION]...
**/
Rho_Q[tid] = factor3[tid]*cos_twoaz[tid];
Rho_U[tid] = factor3[tid]*sin_twoaz[tid];
Rho_V[tid] = (rho_left[tid] - rho_right[tid])*factor0[tid];

Rho_Q_squared[tid] = Rho_Q[tid]*Rho_Q[tid];
Rho_U_squared[tid] = Rho_U[tid]*Rho_U[tid];
Rho_V_squared[tid] = Rho_V[tid]*Rho_V[tid];

/**
 * MORE COMMON TERMS
**/
RR[tid]      = Eta_Q[tid]*Rho_Q[tid] +
               Eta_U[tid]*Rho_U[tid] +
               Eta_V[tid]*Rho_V[tid];
factor4[tid] = Rho_Q_squared[tid] +
               Rho_U_squared[tid] +
               Rho_V_squared[tid];
Delta[tid]   = Eta_I_squared[tid]*(Eta_I_squared[tid] - Eta_Q_squared[tid] -
                                   Eta_U_squared[tid] - Eta_V_squared[tid] +
                                   factor4[tid]) - RR[tid]*RR[tid];
factor5[tid] = pvec[4]/Delta[tid];
```

- Excerpt from Voigt synthesis routine (CudaVoigt)
- Polynomial approx. to Voigt and Faraday-Voigt function
  - (e.g.) here applicable in spectral line wings
- But since different threads must use different approximations, some threads will necessarily diverge
  - Just have to live with this and use __syncthreads()

```
z0[tid] = 272.1014f + y[tid]*(1280.829f + y[tid]*(2802.870f + y[tid]*(3764.966f +
                      y[tid]*(3447.629f + y[tid]*(2256.981f + y[tid]*(1074.409f +
                      y[tid]*(369.1989f + y[tid]*(88.26741f + y[tid]*(13.39880f +
                      y[tid]))))))))));

z2[tid] = 211.678f + y[tid]*(902.3066f + y[tid]*(1758.336f + y[tid]*(2037.310f +
                     y[tid]*(1549.675f + y[tid]*(793.4273f + y[tid]*(266.2987f +
                     y[tid]*(53.59518f + y[tid]*5.0f)))))))));

z4[tid] = 78.86585f + y[tid]*(308.1852f + y[tid]*(497.3014f + y[tid]*(479.2576f +
                      y[tid]*(269.2916f + y[tid]*(80.39278f + y[tid]*10.f)))));

z6[tid] = 22.03523f + y[tid]*(55.02933f + y[tid]*(92.75679f + y[tid]*(53.59518f +
                      y[tid]*10.f)));

z8[tid] = 1.496460f + y[tid]*(13.39880f + y[tid]*5.f);

p0[tid] = 153.5168f + y[tid]*(549.3954f + y[tid]*(919.4955f + y[tid]*(946.8970f +
                      y[tid]*(662.8097f + y[tid]*(328.2151f + y[tid]*(115.3772f +
                      y[tid]*(27.93941f + y[tid]*(4.264678f + y[tid]*0.3183291f))))))))));

p2[tid] = -34.16955f + y[tid]*(-1.322256f + y[tid]*(124.5975f + y[tid]*(189.7730f +
                       y[tid]*(139.4665f + y[tid]*(56.81652f + y[tid]*(12.79458f +
                       y[tid]*1.2733163f))))));

p4[tid] = 2.584042f + y[tid]*(10.46332f + y[tid]*(24.01655f + y[tid]*(29.81482f +
                      y[tid]*(12.79568f + y[tid]*1.9099744f))));

p6[tid] = -0.07272979f + y[tid]*(0.9377051f + y[tid]*(4.266322f + y[tid]*1.273316f));

p8[tid] = 0.0005480304f + y[tid]*0.3183291f;

q1[tid] = 173.2355f + y[tid]*(508.2585f + y[tid]*(685.8378f + y[tid]*(557.5178f +
                      y[tid]*(301.3208f + y[tid]*(111.0528f + y[tid]*(27.62940f +
                      y[tid]*(4.264130f + y[tid]*0.3183291f)))))));

q3[tid] = 18.97431f + y[tid]*(100.7375f + y[tid]*(160.4013f + y[tid]*(130.8905f +
                      y[tid]*(55.88650f + y[tid]*(12.79239f + y[tid]*1.273316f)))));

q5[tid] = 7.985877f + y[tid]*(19.83766f + y[tid]*(28.88480f + y[tid]*(12.79239f +
                      y[tid]*1.909974f)));

q7[tid] = 0.6276985f + y[tid]*(4.264130f + y[tid]*1.273316f);

d[tid]  = 1.7724538f/(z0[tid] + xq[tid]*(z2[tid] + xq[tid]*(z4[tid] +
                      xq[tid]*(z6[tid] + xq[tid]*(z8[tid] + xq[tid])))));

*v = d[tid]*(p0[tid] + xq[tid]*(p2[tid] + xq[tid]*(p4[tid] +
             xq[tid]*(p6[tid] + xq[tid]*(p8[tid])))));

*f = d[tid]*x[tid]*(q1[tid] + xq[tid]*(q3[tid] + xq[tid]*(q5[tid] +
                    xq[tid]*(q7[tid] + xq[tid]*(0.3183291f)))));
```

SOLIS Vector Spectromagnetograph (VSM) Inferred vector magnetic field (28 March 2008 15:45 UTC)



Inversion results for NOAA AR10988

Harker & Mighell, in preparation

Table 6
Timing profile for GENESIS inversions of Fe I λ6302.5

	ΔT_{run} (min)	ΔT_{eval} (min)
SERIAL	84.56 ± 0.26	55.53 ± 0.17
GPU-ACC.	32.82 ± 0.36	2.34 ± 0.01

Table 7
Timing profile for GENESIS inversions of Fe I λ6301.5

	ΔT_{run} (min)	ΔT_{eval} (min)
SERIAL	187.83 ± 0.69	147.54 ± 0.68
GPU-ACC.	36.40 ± 0.15	4.43 ± 0.01

Left: Average & 1σ standard deviations (over 50 independent full-disc inversions) of total runtime and total evaluation time.

Normal Zeeman triplet: 3 Voigt, 3 Faraday-Voigt evaluations per wavelength/thread

- serial evaluation ~ 66% of total runtime
- gpu-accelerated ~ 7% of total runtime
- fitness evaluation speedup ~ **23.7x faster**

Anomalous Zeeman triplet: 12 Voigt, 12 Faraday-Voigt evaluations per wavelength/thread

- serial evaluation ~ 79% of total runtime
- gpu-accelerated ~ 12% of total runtime
- fitness evaluation speedup ~ **33.3x faster**

- Clearly, assessing the quality of the fit to the spectral lines is the most time-consuming step in "evolving" the genetic algorithm population.
  - The embarassingly-parallel GPU-accelerated evaluation relieves this burden.
  - "Task-parallel" implementation also utilizing MPI to process different raster scanlines shows promise…preliminary tests on Accelerator Cluster at NCSA show linear speedup with number of CPU-GPU pairs.

The Future

- We (NSO) are in the initial stages of implementing this method in the SOLIS VSM (http://solis.nso.edu) near real-time data reduction pipeline for solar vector magnetography.

- Our implementation is a bit simplistic. We exploit the embarassingly-parallel aspect of a GA paired with a GPU. Not much effort has been put into ensuring optimum memory accesses.
  - With speedups already ~ 30x, effort may be better used in accelerating other time-consuming parts of the inversion code…

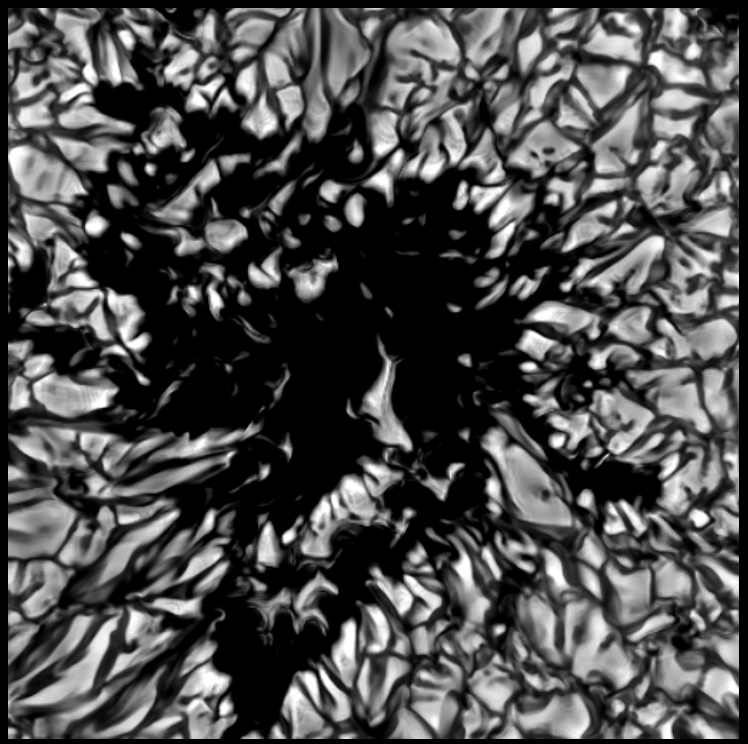
- Current computing environment: 6 dual-core machines in JBOC (Just a Bunch Of Computers) configuration…
  - Each JBOC machine runs a maximum of 4 "scanline processes" reducing a single raster of 2048 2D Stokes spectra with a traditional inversion algorithm based on the Levenberg-Marquardt optimization scheme.
    - Computing time ~ 1.5 hours (or greater, depending on solar activity)
    - For comparison, my GPU-accelerated method does the equivalent work *with a single CPU-GPU pair in ~ 30 minutes!*
    - This method is also uniquely suited to take advantage of SOLIS VSM fast-scan observations of the activity belt or individual active regions for flare pre-cursor monitoring

Ray-tracing Through Numerical Models

Numerical Radiation Magnetohydrodynamics simulations allow solar physicists to construct models that describe the solar atmosphere with a high degree of fidelity.

Such simulations allow us to study the internal structure of sunspots, the processes leading to their formation as well as the connection between the solar interior and the solar atmosphere, a connection largely mediated by magnetic fields.

Right: Model from Cheung et al (ApJ 2010). Ran on 2048 CPU procs on Pleiades (NASA Ames) for 4 weeks.



Radiation MHD

MHD Induction Equation $$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{v} \times \mathbf{B}) - \nabla \times (\eta \nabla \times \mathbf{B})$$

Continuity Equation $$\frac{\partial \varrho}{\partial t} + \nabla \cdot (\varrho \mathbf{v}) = 0$$

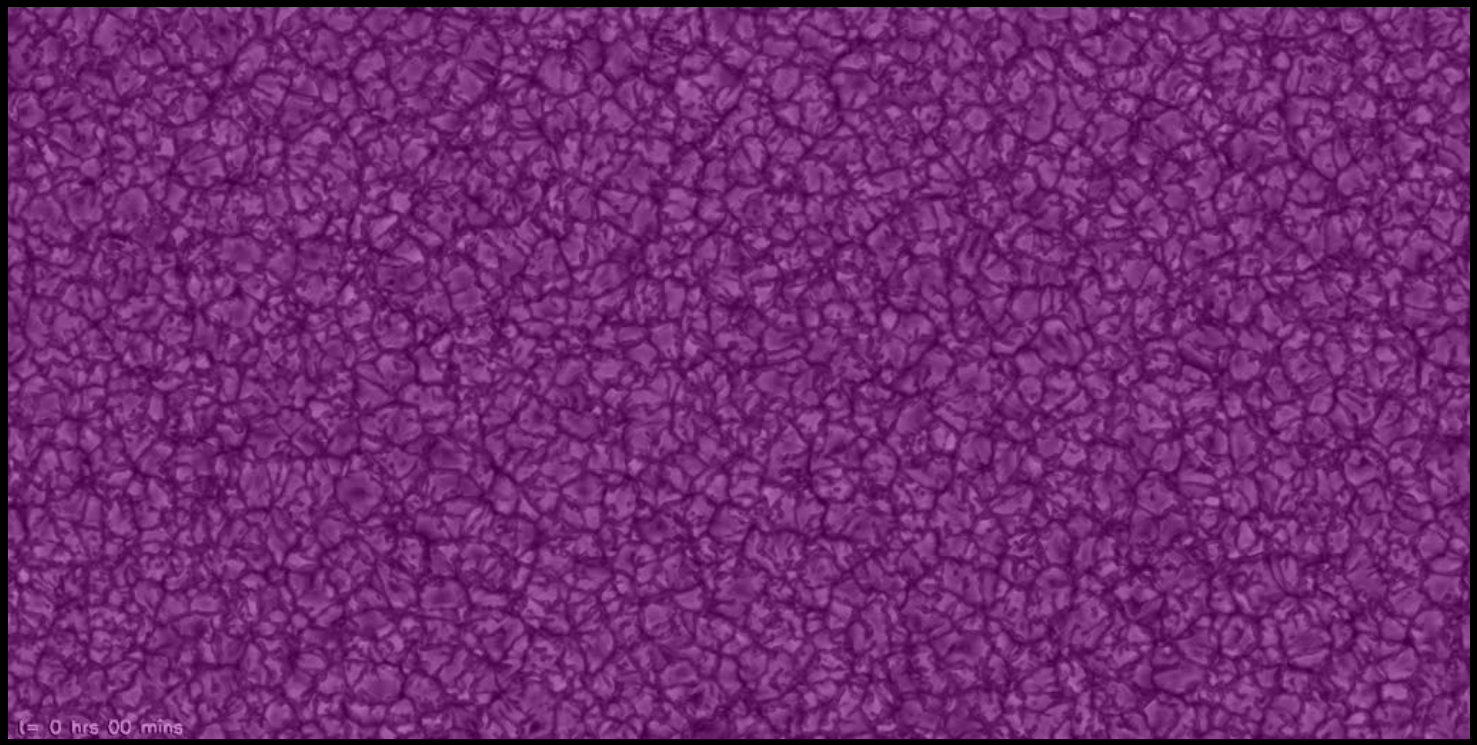
Momentum Equation $$\frac{\partial \varrho \mathbf{v}}{\partial t} + \nabla \cdot \left[\varrho \mathbf{v}\mathbf{v} + \left(p + \frac{|\mathbf{B}|^2}{8\pi} \right) \underline{\mathbf{1}} - \frac{\mathbf{B}\mathbf{B}}{4\pi} \right] = \varrho \mathbf{g} + \nabla \cdot \underline{\underline{\tau}}$$

Energy Equation $$\begin{aligned} \frac{\partial e}{\partial t} &+ \nabla \cdot \left[\mathbf{v} \left(e + p + \frac{|\mathbf{B}|^2}{8\pi} \right) - \frac{1}{4\pi} \mathbf{B}(\mathbf{v} \cdot \mathbf{B}) \right] \\ &= \frac{1}{4\pi} \nabla \cdot (\mathbf{B} \times \eta \nabla \times \mathbf{B}) + \nabla \cdot (\mathbf{v} \cdot \underline{\underline{\tau}}) + \nabla \cdot (K \nabla T) \\ &+ \varrho(\mathbf{g} \cdot \mathbf{v}) + Q_{\mathrm{rad}} \end{aligned}$$

Radiative Transfer Equation $$\boldsymbol{\mu} \cdot \nabla I_\nu = \kappa_\nu \varrho (S_\nu - I_\nu)$$

Equation of State $$T = T(\varrho, e_{\mathrm{int}}), \quad p = p(\varrho, e_{\mathrm{int}})$$

Simulation of Sunspot Formation



Ray-tracing code in CUDA

- Aim: To study the appearance of sunspots at different viewing angles
- Method: Solve the radiative transfer eqn.

$$\boldsymbol{\mu} \cdot \nabla I_\nu = \kappa_\nu \varrho (S_\nu - I_\nu)$$

- Implementation:
  - In Local Thermodynamic Equilibrium, the source function S is simply the Planck function and the opacity κ is dependent on the local plasma temperature T and gas density ρ, i.e. κ=κ(T,ρ).
  - The 3D distribution of T and ρ are stored as 3D textures and the look-up table κ(T,ρ) is stored as a 2D texture. On points along a ray, texture fetches (tex3D() and tex2D()) are used to obtain linearly interpolated values of ρ and T.
  - Each CUDA thread works on its own ray. First forward propagate to calculate optical depth, then back propagate to calculate intensity.

What the CUDA kernel does

```
for (int i=0; i<maxSteps; i++) {
    //Position along ray
    pos0 = eyeRay.o+eyeRay.d*(t);
      pos1 = eyeRay.o+eyeRay.d*(t+tstep);
      pos0 = (pos0-make_float3(xmin,ymin,zmin))/L;
      pos1 = (pos1-make_float3(xmin,ymin,zmin))/L;
      // Use texture fetches to get local values of thermodynamic quantities.
      t0 = (tex3D(ttex, pos0.x, pos0.y, pos0.z)); t1 = (tex3D(ttex, pos1.x, pos1.y, pos1.z));
      d0 = (tex3D(dtex, pos0.x, pos0.y, pos0.z)); d1 = (tex3D(dtex, pos1.x, pos1.y, pos1.z));
      //Use texture fetches to get opacity from look-up table.
      dd = (__logf(d0) - dmin)/drange;  tt = (__logf(t0) - tmin)/trange;
      kap0 = expf(tex3D(otex, tt, dd, ilam));
      dd = (__logf(d1) - dmin)/drange;  tt = (__logf(t1) - tmin)/trange;
      kap1 = expf(tex3D(otex, tt, dd, ilam));
      //Change in optical depth
      dtau = tstep*(0.3333333f*(kap0*d0+kap1*d1) + 0.1666667f*(kap0*d1+kap1*d0));

      tau += dtau;
      t += tstep;
      if ((tau > 100.0f) && ((t-tnear) < smalldist)) break;
      if ( (tfar-t) < smalldist) break;

      if (tau > 100.0f) {
        //If large optical depth, set intensity to source function attenuated by tau.
        b1 = expf(tex3D(atex, tt, dd, ilam));
        intensity = expf(-tau)*b1;
        break;
      }
}
```

What the CUDA kernel does

```
for(int i=0; i<maxSteps; i++) {
        if (t < tnear) break;
        pos0 = eyeRay.o+eyeRay.d*(t);
        pos1 = eyeRay.o+eyeRay.d*(t-tstep);
        pos0 = (pos0-make_float3(xmin,ymin,zmin))/L;
        pos1 = (pos1-make_float3(xmin,ymin,zmin))/L;
         // Use texture fetches to get local values of thermodynamic quantities.
        t0 = (tex3D(ttex, pos0.x, pos0.y, pos0.z)); t1 = (tex3D(ttex, pos1.x, pos1.y, pos1.z));
        d0 = (tex3D(dtex, pos0.x, pos0.y, pos0.z)); d1 = (tex3D(dtex, pos1.x, pos1.y, pos1.z));
      //Use texture fetches to get opacity and source function from look-up tables.
        dd = (__logf(d0) - dmin)/drange; tt = (__logf(t0) - tmin)/trange;
        kap0 = expf(tex3D(otex, tt, dd, ilam));
        b0 = expf(tex3D(atex, tt, dd, ilam));
        dd = (__logf(d1) - dmin)/drange; tt = (__logf(t1) - tmin)/trange;
        kap1 = expf(tex3D(otex, tt, dd, ilam));
        b1 = expf(tex3D(atex, tt, dd, ilam));
        dtau = tstep*(0.3333333f*(kap0*d0+kap1*d1) + 0.1666667f*(kap0*d1+kap1*d0));
        if (dtau < 5.0e-3) {
                edt = 1.0f - dtau + 0.5f*dtau*dtau;
                w0 = dtau - 0.5f*dtau*dtau;
            } else {
              edt= expf(-dtau);
                w0 = 1.0f-edt;
        }
        tau -= dtau; t -= tstep;
      //Update intensity
        intensity += (b1*(1.0f-w0/dtau) + b0*(w0/dtau - edt))*expf(-tau);
}
```

Synthetic AIA images by raytracing

MHD model courtesy of J. Martínez-Sykora

Tracking Flows in the Solar Atmosphere

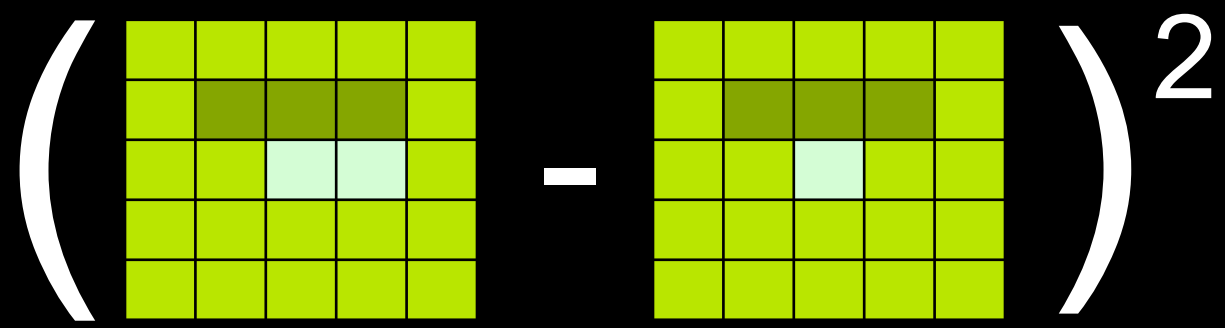
Beneath the solar photosphere is the convection zone, where convective flows can shuffle magnetic fields around the so-called “magnetic carpet” and drive plasma surges and jets into the chromosphere and corona.

To track the motion of plasma features in AIA images, we developed a Local Correlation Tracker to estimate the components of plasma velocity transverse to the line-of-sight.

NB: Intensity structures on the Sun do not change purely due to advection of plasma.

Local Correlation Tracker

- Implementation in OpenCL
  - Each work item estimates the velocity at a pixel (i,j). Consider tiles with 5x5 pixels:
    - Let $\mathbf{T_{i,j}^{n}}$ = Tile centered at (i,j) in the n-th frame of an image sequence. **G** is some apodizing function.
    - Define $\mathbf{S(\Delta i,\Delta j) = G(\ T_{i+\Delta i,j+\Delta j}^{n-1} - T_{i,j}^{n-1}\ )^2}$
    - Define $\mathbf{C(\Delta i,\Delta j) = G(\ T_{i+\Delta i,j+\Delta j}^{n+1} - T_{i,j}^{n-1}\ )^2}$



Local Correlation Tracker

- Implementation in OpenCL
  - Each work item estimates the velocity at a pixel (i,j).
  - For pixel shifts Δi, Δj between -5 and 5, calculate **S(Δi,Δj) and C(Δi,Δj)**. Since pixel values in an image are reused by multiple work items many times, the local portion of the images n-1 and n +1 are stored in local memory (OpenCL lingo; shared memory in CUDA lingo).
  - The estimate of the mean shifts (or velocity if divided by the time between frames) is

    $$\langle \Delta i \rangle = A^{-1} \Sigma\, \Delta i \exp(-kC/S),$$

    $$\langle \Delta j \rangle = A^{-1} \Sigma\, \Delta j \exp(-kC/S),$$

    $$A = \Sigma \exp(-kC/S)$$

    where **k** is some chosen “stiffness” coefficient.

OpenCL kernel code

```
//Loop over shifts
    for (di = -5; di<6; di++) {
        gX = GI + di;
        for (dj = -5; dj<6; dj++) {
          gY = GJ + dj;
          bind = gY*stride + gX;

          corr = 0.0f;
          sorr = 0.0f;

        //Calculate C. aTile and bTile both in local memory
          dorr = (aTile[aind-1-stride]-bTile[bind-1-stride]); corr+=dorr*dorr*2.0f;
          dorr = (aTile[aind  -stride]-bTile[bind  -stride]); corr+=dorr*dorr*4.0f;
          dorr = (aTile[aind+1-stride]-bTile[bind+1-stride]); corr+=dorr*dorr*2.0f;
          dorr = (aTile[aind-1       ]-bTile[bind-1       ]); corr+=dorr*dorr*4.0f;
           etc
        // Calculate S
          dorr = (aTile[aind-1-stride]-aTile[bind-1-stride]); sorr+=dorr*dorr*2.0f;
          dorr = (aTile[aind  -stride]-aTile[bind  -stride]); sorr+=dorr*dorr*4.0f;
          dorr = (aTile[aind+1-stride]-aTile[bind+1-stride]); sorr+=dorr*dorr*2.0f;
          dorr = (aTile[aind-1       ]-aTile[bind-1       ]); sorr+=dorr*dorr*4.0f;
          etc
```

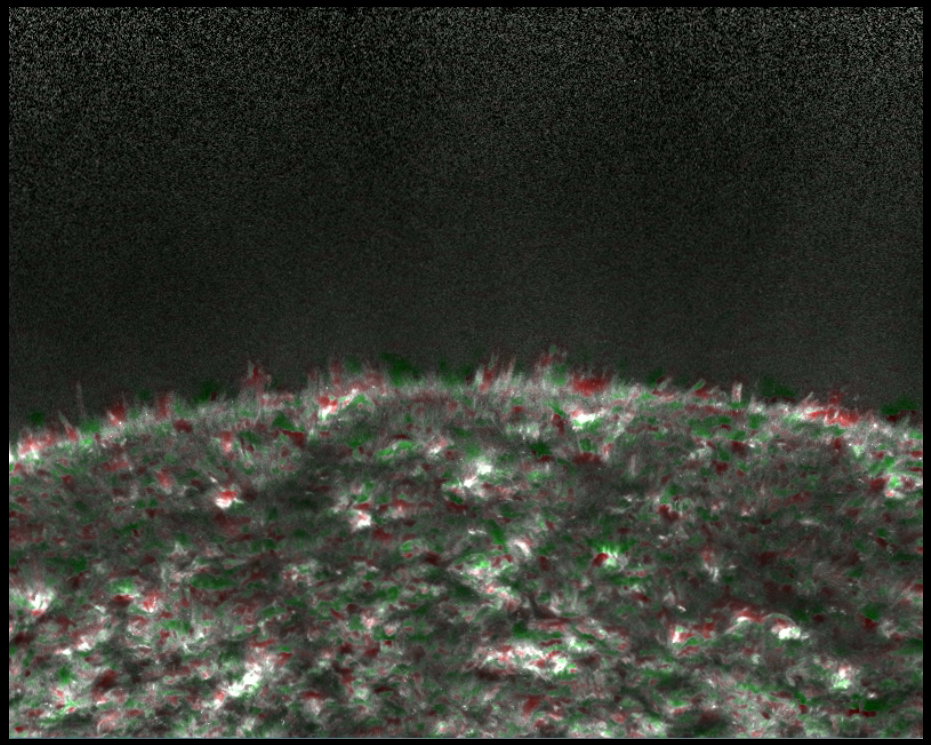
```
        //Calculate weight using C and S
          corr = native_exp(-corr/sorr*stiffness);
          corrmean += corr;
          vx      += corr*(float)di;
          vy      += corr*(float)dj;
        }
    }
    //Aveage
    corrmean = 1.0f/corrmean;
    velocity[(globalY*inWidth + globalX)*2] = vx*corrmean;
    velocity[(globalY*inWidth + globalX)*2+1] = vy*corrmean;
```

aTile and bTile stored on local memory

LCT performance

- LCT on an 800x640 image sequence at 8 frames / sec
- For calculation of C and S at each pixel for each shift: 5x5x5x2 FLOPs
- Parameter space for shifts: 11x11
- Total ~ 0.12 TFLOPs / sec
- Achieved on a GTX 285 card on a Mac Pro
  - Timing test includes display of image on the screen.

Movie Demonstrating LCT



Summary

- GPUs are being used in a variety of contexts in solar physics
  - Inversion of Stokes profiles to measure the Sun's magnetic field
  - Synthetic images from numerical magnetohydrodynamics simulations
  - Tracking of plasma flows
- Interesting possibilities for the future
  - Port MHD simulation code to run on GPUs
  - Incorporate OpenCL kernels into Panorama

Thank you for your attention

For more information on SDO

- http://sdo.gsfc.nasa.gov - NASA's SDO website
- http://sdowww.lmsal.com - Latest AIA images and movies

- Contacts
  - Mark Cheung, cheung@lmsal.com
  - Ralph Seguin, seguin@lmsal.com
  - Brian Harker-Lundberg, bhl@email.noao.edu