Advanced Geospatial Image Processing using Graphics Processing Units

Atle Borsholm
Ron Kneusel

Exelis Visual Information Solutions
Boulder, CO USA

Why?

- Common geospatial image processing algorithms are computationally demanding.
- Geospatial images are large and becoming larger. Sizes of 60,000 pixels on a side are not uncommon.
- GPU implementations of key geospatial processing algorithms can substantially offset this increase in calculation time.
- Users of our commercial products benefit from an environment that makes our tools easily extensible via the GPU.

AMPE – *Advanced Massively Parallel Execution*

Three tiers integrated with ENVI and IDL:

- Geospatial – GPU versions of key algorithms for common computationally intensive processing provided out of the box.
- Image Processing – GPU versions of common processing routines provided out of the box.
- User-Defined Kernels – A sophisticated integration layer allows advanced users the ability to define, compile, and link custom GPU kernels within ENVI and IDL.

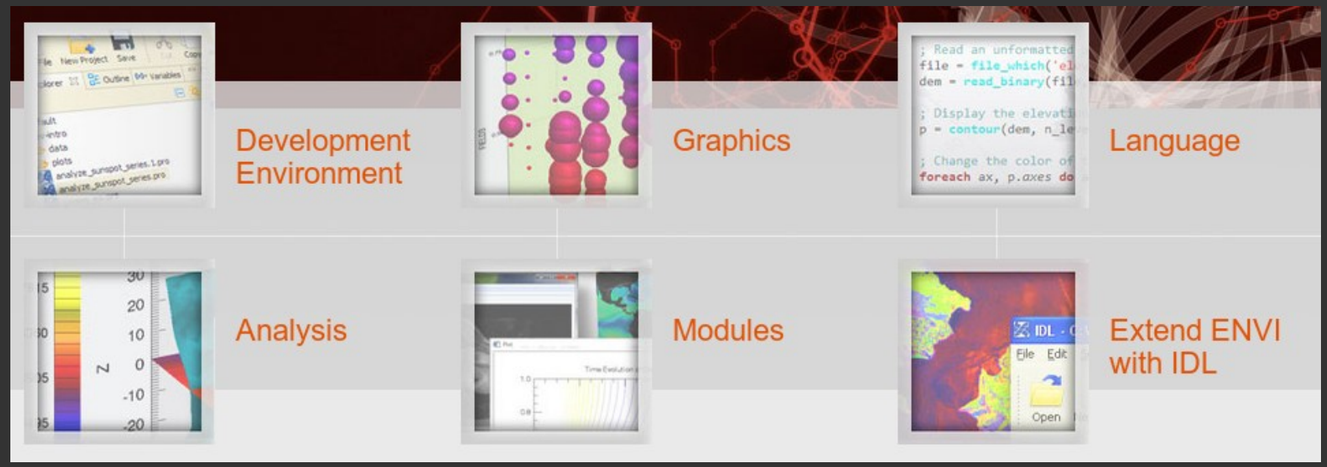
ENVI and IDL

ENVI
The image analysis software of choice for image scientists to GIS professionals

ENVI LiDAR
Fully customizable point cloud analysis software

ENVI SARscape
Specialized SAR analysis routines in the familiar ENVI interface

IDL®



AMPE is fully integrated with and extends ENVI and IDL

AMPE - Geospatial Algorithms

- Orthorectification
- Atmospheric Correction (using QUAC)
- Principal Component Analysis (PCA)
- Adaptive Coherence Estimator (ACE)

AMPE – Image Processing

- Interpolation – multiple algorithms from nearest-neighbor to Lanczos
- Array operations – min/max, mean, rescale
- Analysis - joint histogram, mutual information,cross-correlation, eigenvectors

AMPE – Testing Environment

All tests were performed on the following hardware:

- CPU: Dell, six cores, 3.47 GHz Xeon, 12 GB RAM, Ubuntu 14.04
- GPU: CUDA 6.5, NVIDIA Tesla K40

N.B. CPU tests used optimized code in ENVI and IDL which itself was often multithreaded.

AMPE – Performance (Geospatial)

	CPU	GPU	Improvement
Orthorectification	366.1	53.6	6.8x
Atmospheric Correction	138.72	5.16	26.9x
Principal Components	22.14	1.58	14.0x
Adaptive Coherence Estimator	70.28	4.10	17.1x

(time in seconds, average over multiple runs)

AMPE – Performance (Interpolation)

	CPU	GPU	Improvement
Nearest-Neighbor	0.0402	0.0033	12.3x
Bilinear	0.0855	0.0007	127.0x
Bicubic	0.2220	0.0015	143.6x
Lagrange	1.4495	0.0015	947.4x

(time in seconds, average over multiple runs)

AMPE – User-defined Kernels

Calculate the normalized difference vegetation index (NDVI):

$$NDVI = (NIR - VIS) / (NIR + VIS)$$

Where:

NIR = Near infrared band
VIS = Visible band (red, ~675 nm)

AMPE – User-defined Kernels

```
extern "C" {
__global__ void k_NDVI(short *NIR, short *VIS, float *result, int n)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    for ( ; i < n; i += blockDim.x * gridDim.x) {
        result[i] = ((float) NIR[i] - VIS[i]) / (NIR[i] + VIS[i]);
    }
}
}
```

- Kernel code in a separate .cu file (will change in future versions)

- Call `ampe_build_kernel()` from running ENVI/IDL session

- Will compile the kernel and build auto-generated IDL code (next slide)

AMPE – User-defined Kernels

```
function ampe::NDVI, NIR, VIS
    compile_opt idl2, logical_predicate

    ; parameter validation...

    ; allocate output device memory, float array
    output = ampe_makearray(NIR.dim, TYPE=4)

    ; define "C" entry point
    entry = 'k_NDVI'
    module = 'AMPE_NDVI' + self.bitString

    ; set up grid
    ng = NIR.n_elts / 256 + (NIR.n_elts mod nt gt 0) < 65000

    ; call kernel
    ampe_run, NIR, VIS, output, NIR.n_elts, THREAD_X=nt, GRID_X=ng, $
        FUNC=entry, MODULE=module, CALLBACK=self.oKernel

    return, output
end
```

AMPE – User-defined Kernels

```
IDL> d_nir = ampe_put(nir)
IDL> d_vis = ampe_put(vis)
IDL> d_ndvi = ampe.NDVI(d_nir, d_vis)
IDL> ndvi = ampe_get(d_ndvi)
```

- `IDL>` is the interactive IDL prompt
- `NIR` and `VIS` contain image bands on the CPU
- `ampe_put()` places the image bands on the GPU
- `ampe.NDVI()` calls a new method off the ampe object
- `ampe_get()` returns the NDVI array from the GPU

AMPE is...

Sophisticated geospatial analysis algorithms
and
common image processing routines
combined with
a framework for integrating user-defined kernels

in order to maximize performance using GPUs.

Thank you!

- Email
  - Atle Borsholm (atle.borsholm@exelisinc.com)
  - Ron Kneusel (ron.kneusel@exelisinc.com)

- Website
  - www.exelisvis.com

Please complete the Presenter Evaluation sent to you by email or through the GTC Mobile App. Your feedback is important!