GPULib: GPU Computing in IDL (An Update)

Peter Messmer, Paul Mullowney, Mike Galloy, Brian Granger, Dan Karipides, David Fillmore, Nate Sizemore, Keegan Amyx, Dave Wade-Stein, Seth Veitzer

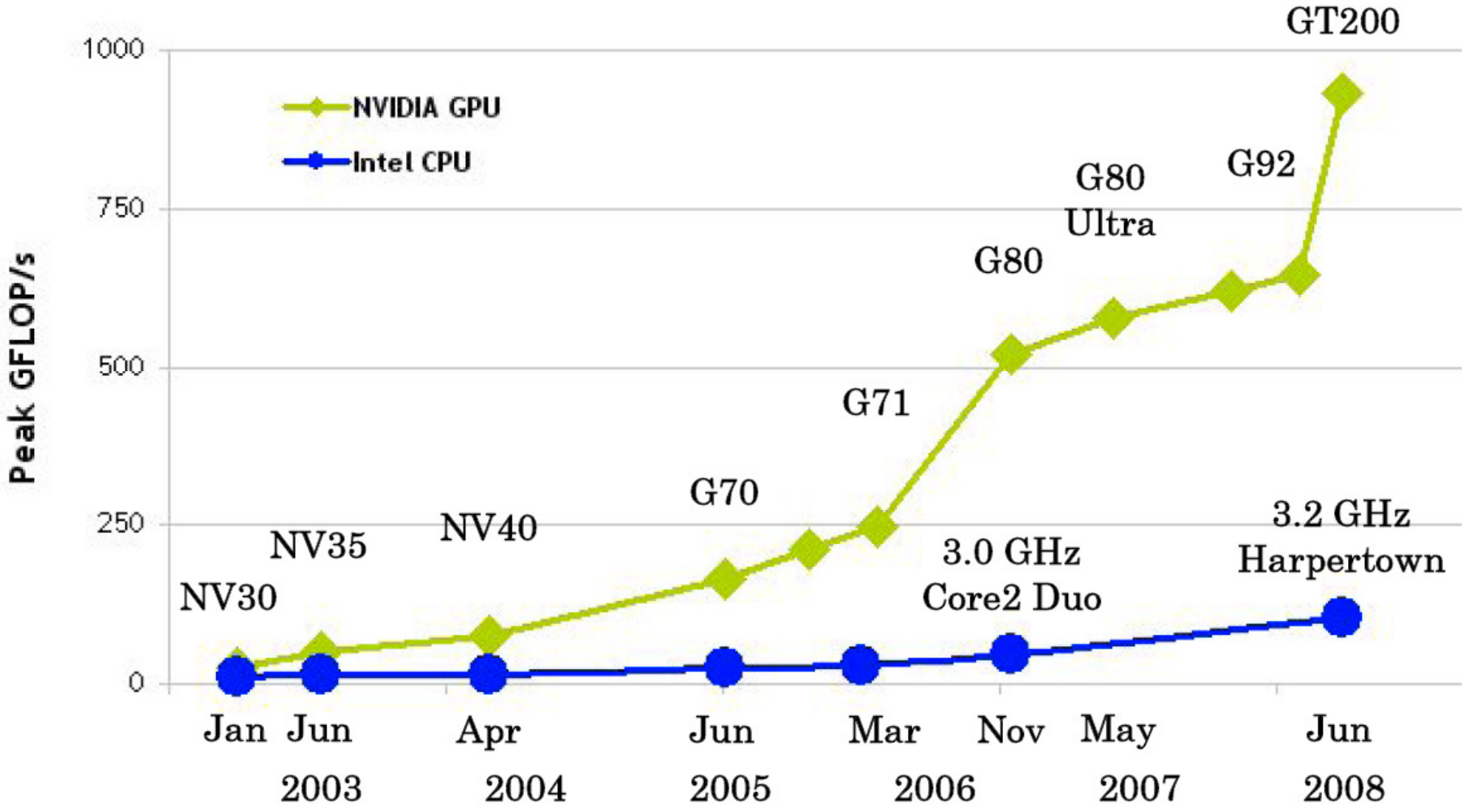
messmer@txcorp.com

Tech-X Corporation
5621 Arapahoe Ave., Boulder, CO 80303
www.txcorp.com

This work is supported by
NASA SBIR Phase-II Grant #NNG06CA13C

IDL User Group Meeting, LASP, Boulder CO, October 16, 2008

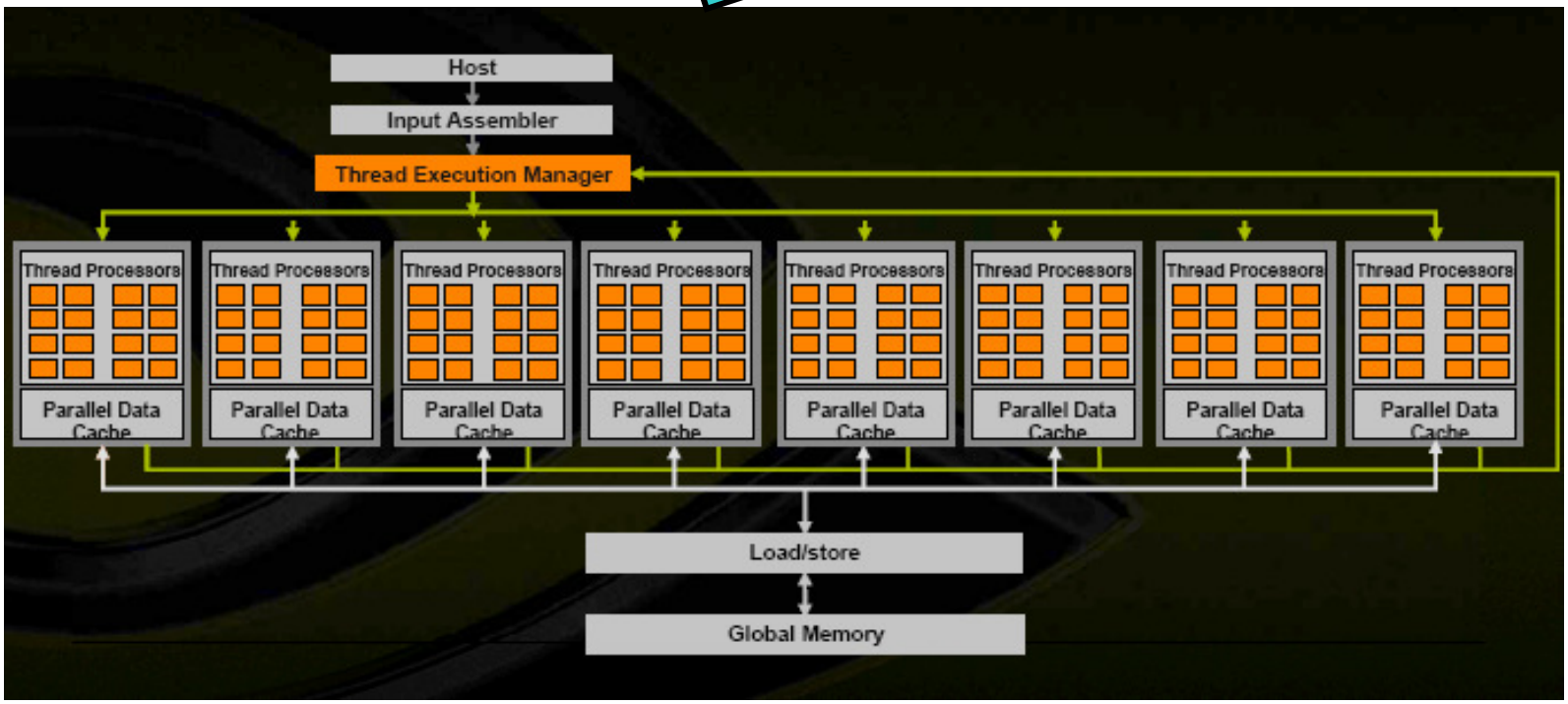
Why IDL on a GPU?



GPUs: Floating-Point Co-Processors

- **NVIDIA's CUDA (Compute Unified Device Architecture)**
  - Architecture/Programming model no longer focussed [illegible]ics
  - 128 processing elements, grouped into 16 [illegible]ultiprocessor')
  - Processors have access to entire [illegible]w (no conv.cache)
  - 2 SIMD processors share a co[illegible]ared memory")
  - Stream processor: Scalar proce[illegible]str/cycle, branching, etc.

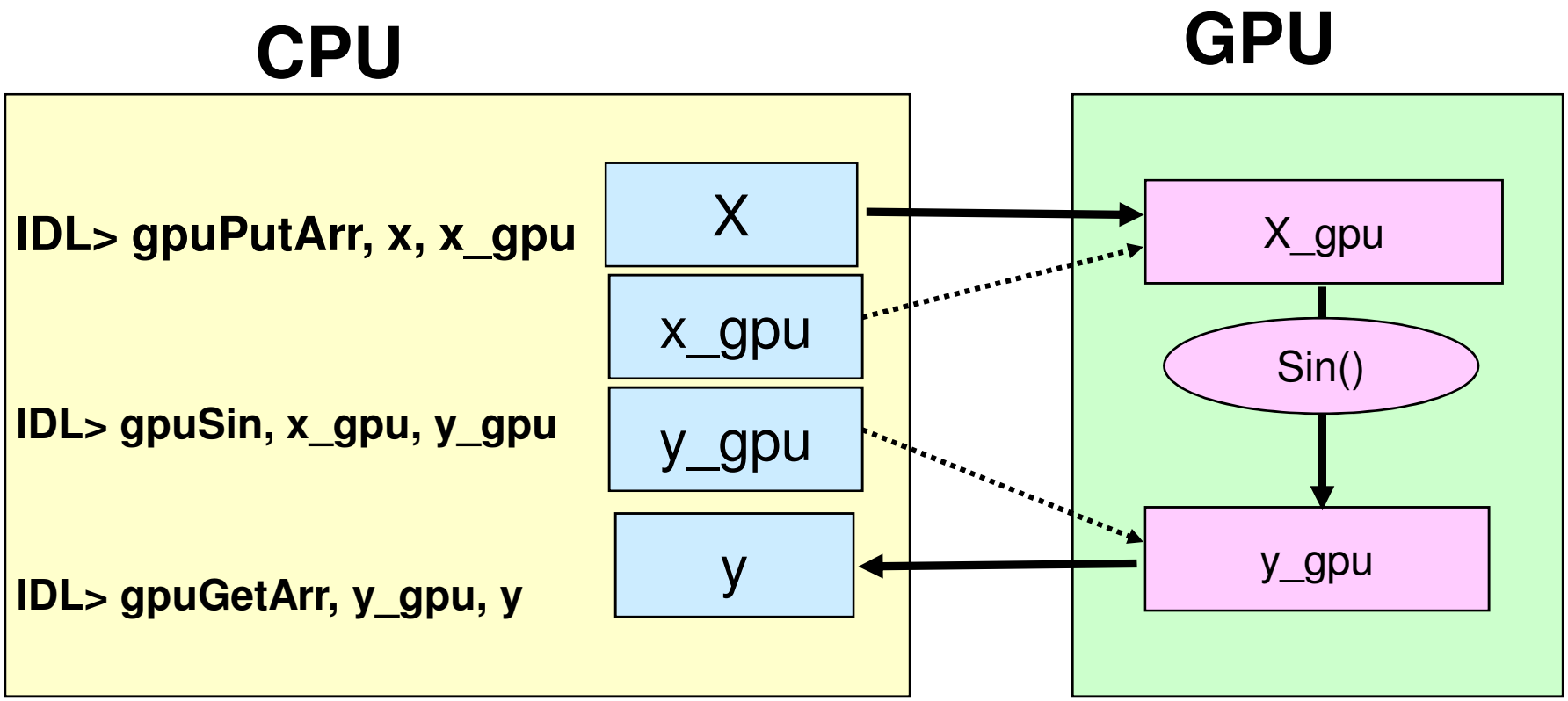
Latest Generatin (GTX280):
240 processors
Support for Double Precision



GPUlib: One way to simplify GPU development

- **Data objects on GPU represented as structure/object on CPU**
  - Contains size information, dimensionality and pointer to GPU memory
- **GPULib provides a large set of vector operations**
  - Data transfer GPU/CPU, memory management
  - Arithmetic, transcendental, logical functions
  - **Support for different types (float, double, complex, dcomplex)**
  - Data parallel primitives, reduction, masking (**total, where**)
  - Array operations (reshaping, interpolation, range selection, **type casting**)
  - NVIDIA's cuBLAS, **cuFFT**
- **Download technology preview http://gpulib.txcorp.com**
  (free for non-commercial use)
- **Release at SC'08 (Mid November)**

A GPULib example in IDL



GPUlib: Some Vector Operations on GPU

- **Memory allocation on GPU**

```
y_gpu = gpuFltarr(100, 100)
```

- **Data transfer**

```
gpuPutArr, x, x_gpu
```

- **Binary operators both plain and affine transform**

```
gpuAdd, x_gpu, y_gpu, z_gpu
gpuExp, a, b, x_gpu, c, d, z_gpu
```

- **IDL intrinsics**

```
gpuInterpolate, gpuTotal, gpuCongrid, gpuSubArr,
  gpuReform, gpuWhere, gpuRandomu, gpuFltarr, gpuComplex,
  gpuMatrix_multiply …
```

- **IDL structure contains all information about GPU object**

```
type, n_elements, n_dimensions, dimensions, handle
```

Example: Image Deconvolution

- Image is convolved with detector point-spread function:

$$I_{obs}(x,y) = \int I_{true}(x-u, y-v)P(u,v)dudv$$

- Clean image by (complex) division in Fourier space:

$$I_{true}(x,y) = FFT^{-1}(FFT(I_{obs}) / FFT(P))$$

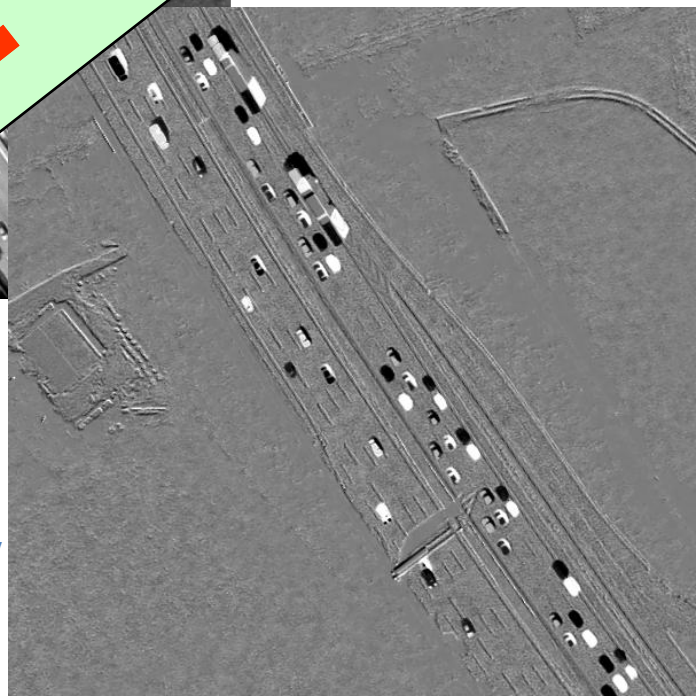
- Fairly large computational load per CPU-GPU data transfer
- Speedup ranging from 5x – 28x for 256x256 – 3kx3k images

```
gpuFFT, img, gpu_img_fft
gpuDiv, gpu_img_fft, gpu_psf, gpu_img_fft
gpuFFT, gpu_img_fft, gpu_clean, /INVERSE
gpuGetArr, gpu_clean, clean
```

GPULib example: Image processing

Speedup: 16x

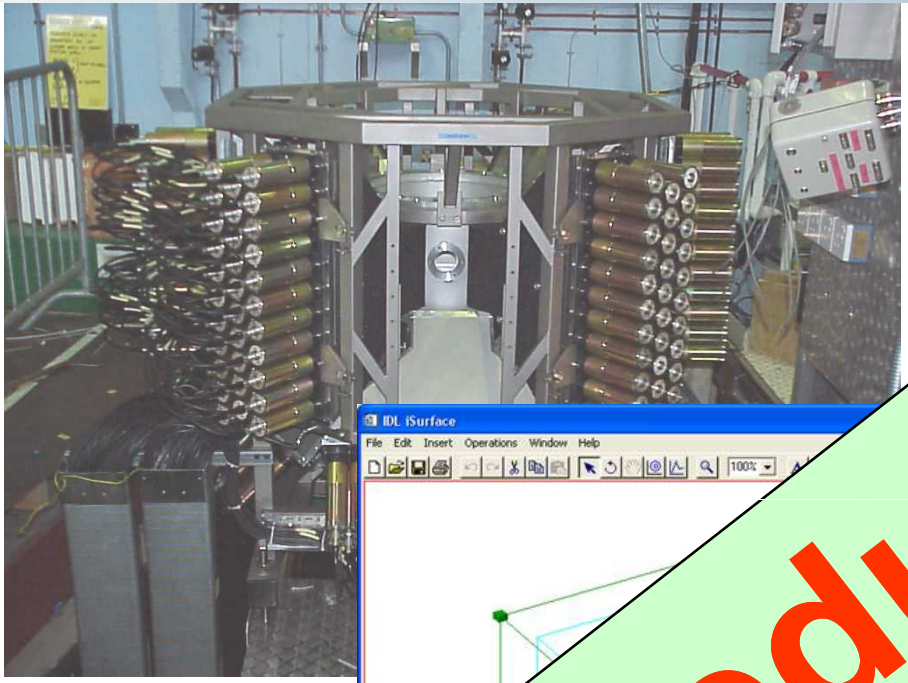
Principal Component Analysis (PCA)



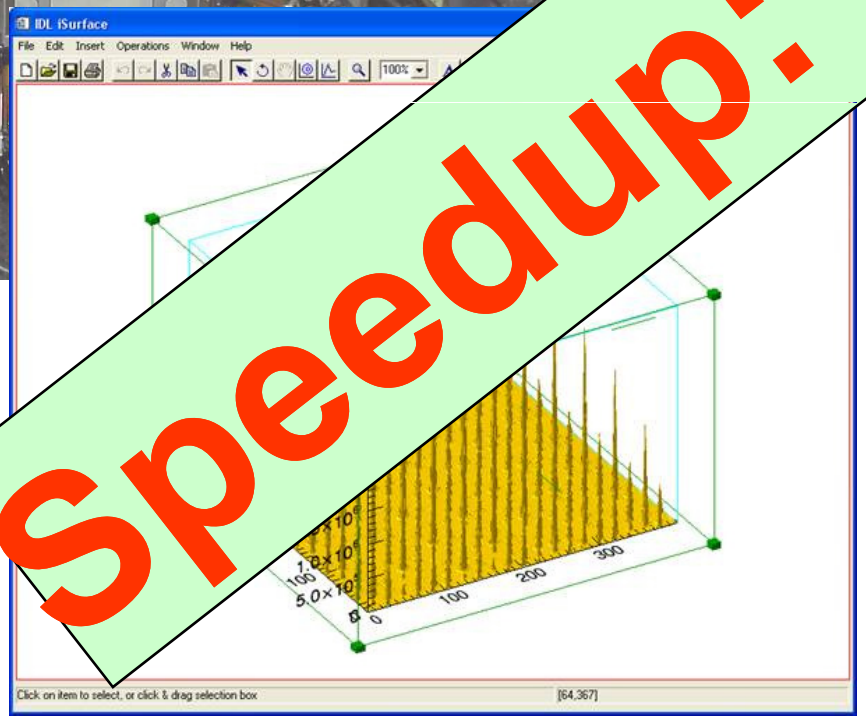
Data courtesy of
Dr. Mort Canty,
FZ Juelich, Germany

http://fwenvi-idl.blogspot.com/

GPULib example: Simulation



Neutron scattering experiment



Speedup: 10x

Use simulation written in IDL to compute location of scattering maxima (Bragg peaks)

Data courtesy of
Dr. Matthias Gutmann,
Rutherford Appleton
Research Lab, UK

How to write code for GPULib?

- **Develop/debug IDL code**
- **Optimize IDL code via vectorization**
  - A well tuned IDL code will benefit most from GPU
- **Time it**
  - Measure, don't guess
  - Use IDL's profiler
- **Successively start moving parts of computation onto GPU**
    - gpuPutArr, gpuGetArr for debugging
- **Redesign parts of the code**
  - reduce CPU<->GPU data transfer
  - increase vector length (> 10^4 elements)
- **Create custom kernels**
- **=> talk to us!**