A Sampling of CUDA Libraries

Michael Garland

NVIDIA Research

CUBLAS

- Implementation of BLAS (Basic Linear Algebra Subprograms) on top of CUDA driver
  - Self-contained at the API level, no direct interaction with CUDA driver
- Basic model for use
  - Create matrix and vector objects in GPU memory space
  - Fill objects with data
  - Call sequence of CUBLAS functions
  - Retrieve data from GPU
- CUBLAS library contains helper functions
  - Creating and destroying objects in GPU space
  - Writing data to and retrieving data from objects

Supported Features

- BLAS functions
  - Single precision data:
    - Level 1 (vector-vector O(N))
    - Level 2 (matrix-vector O(N^2))
    - Level 3 (matrix-matrix O(N^3))
  - Complex single precision data:
    - Level 1
    - CGEMM
  - Double precision data:
    - Level 1: DASUM, DAXPY, DCOPY, DDOT, DNRM2, DROT, DROTM, DSCAL, DSWAP, ISAMAX, IDAMIN
    - Level 2: DGEMV, DGER, DSYR, DTRSV
    - Level 3: ZGEMM, DGEMM, DTRSM, DTRMM, DSYMM, DSYRK, DSYR2K
- Following BLAS convention, CUBLAS uses column-major storage

CUFFT

- The Fast Fourier Transform (FFT) is a divide-and-conquer algorithm for efficiently computing discrete Fourier transform of complex or real-valued data sets.
- CUFFT is the CUDA FFT library
  - Provides a simple interface for computing parallel FFT on an NVIDIA GPU
  - Allows users to leverage the floating-point power and parallelism of the GPU without having to develop a custom, GPU-based FFT implementation

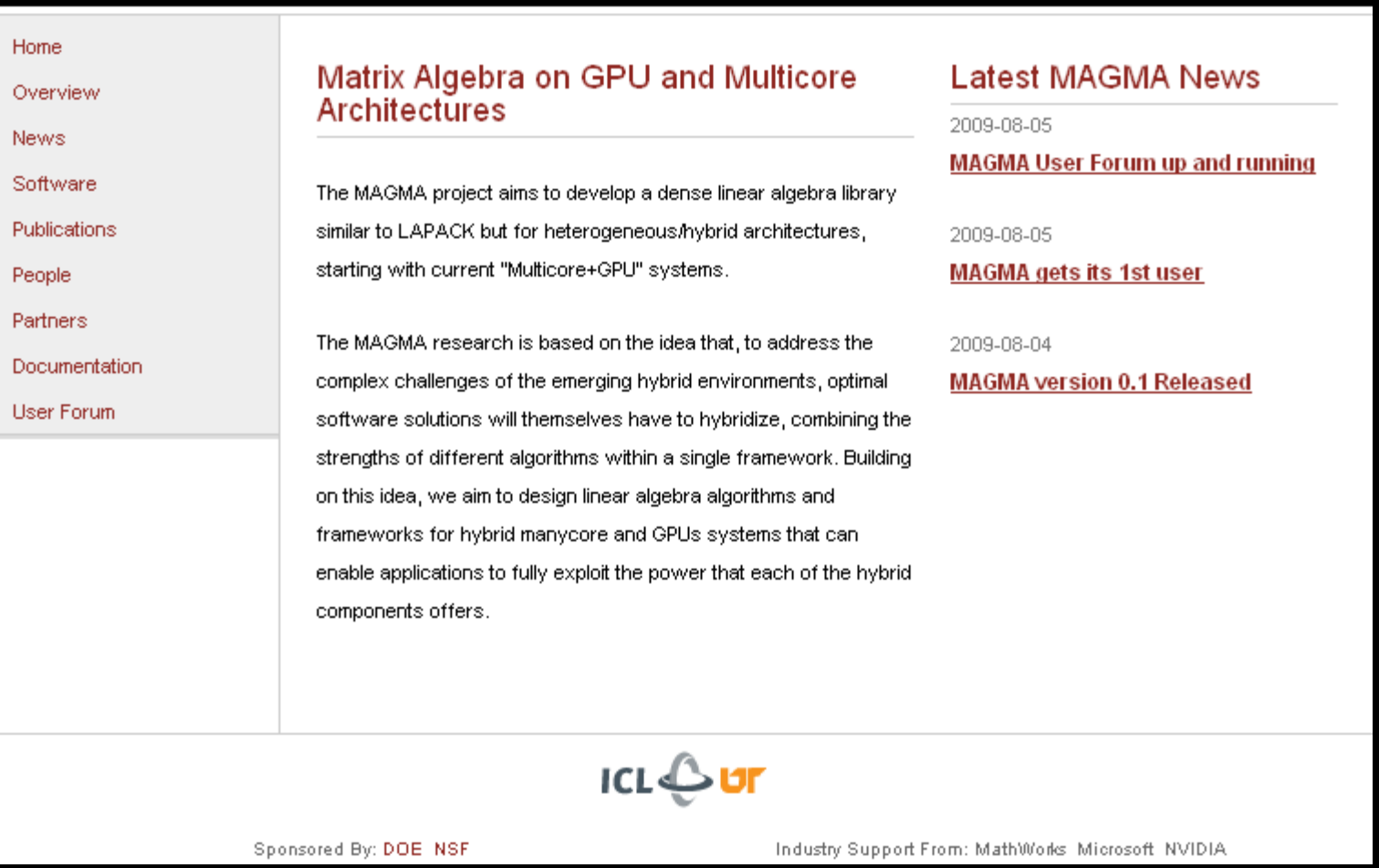
Supported Features

- 1D, 2D and 3D transforms of complex and real-valued data
- Batched execution for doing multiple 1D transforms in parallel
- 1D transform size up to 8M elements
- 2D and 3D transform sizes in the range [2,16384]
- In-place and out-of-place transforms for real and complex data.

Code example:
2D complex to complex transform

```
#define NX 256
#define NY 128

cufftHandle plan;
cufftComplex *idata, *odata;
cudaMalloc((void**)&idata, sizeof(cufftComplex)*NX*NY);
cudaMalloc((void**)&odata, sizeof(cufftComplex)*NX*NY);
…
/* Create a 2D FFT plan. */
 cufftPlan2d(&plan, NX,NY, CUFFT_C2C);

 /* Use the CUFFT plan to transform the signal out of place. */
 cufftExecC2C(plan, idata, odata, CUFFT_FORWARD);

/* Inverse transform the signal in place. */
 cufftExecC2C(plan, odata, odata, CUFFT_INVERSE);

/* Note:
    Different pointers to input and output arrays implies out of place transformation
 */

 /* Destroy the CUFFT plan. */
   cufftDestroy(plan);

   cudaFree(idata), cudaFree(odata);
```

MAGMA



LU, QR, & Cholesky factorization (`getrf`, `geqrf`, `potrf`)

http://icl.cs.utk.edu/magma

NVPP Image Processing Library

- **API identical to IPP** (Intel Integrated Performance Primitives)
- **Supported functionality:**
  - **Data exchange & initialization**
  - **Arithmetic & Logical Ops**
  - **Threshold & Compare Ops**
  - **Color Conversion**
  - **JPEG**
  - **Filter Functions**
  - **Geometry Transforms**
  - **Statistics**
  - **Computer Vision (ApplyHaarClassifier, Canny)**

http://www.nvidia.com/nvpp

Thrust: Algorithms + Containers

```
int main(void)
{
  // generate random data on the host
  thrust::host_vector<int> h_vec(1000000);
  thrust::generate(h_vec.begin(), h_vec.end(), rand);

  // transfer to device
  thrust::device_vector<int> d_vec = h_vec;

  // sort 140M 32b keys/sec on GT200
  thrust::sort(d_vec.begin(), d_vec.end());

  return 0;
}
```

automatic allocation & copy

http://thrust.googlecode.com

Example: Level 1 BLAS operations

- Define some function object types

```
struct scale_and_add;  struct square;  struct plus;
```

- Apply some Thrust algorithms

```
void saxpy(int n, float alpha, float *x, float *y){
    thrust::transform(x, x+n, y,
                      y, scale_and_add(alpha));
}

float snrm2(int n, float *x) {
    return sqrt(thrust::transform_reduce(x, x+n,
                square(), 0, plus()));
}
```

Templates for compile-time dispatch

- `thrust::sort` will select right algorithm
  - radix sort for built-in types (int, float, etc.)
  - merge sort where radix sort cannot be used
- `thrust::reduce` handles data sizes appropriately
  - e.g., efficient memory access of 8-bit chars vs. 32-bit floats
- Handle `host_vector` & `device_vector` transparently
- Provide user-defined function objects
  - `thrust::sort(begin, end, my_comparator());`

CUDPP:
CUDA Data Parallel Primitives

- Parallel primitives with best-in-class performance

- Collaboration between UC Davis and NVIDIA
  - John Owens (UC Davis)
    - Shubho Sengupta, Yao Zhang, Andrew Davidson, Stanley Tseng
  - Mark Harris (NVIDIA)

http://cudpp.googlecode.com

CUDPP Functionality

- Currently supported in CUDPP 1.1:
  - scan, segmented scan, stream compact
  - radix sort
  - sparse matrix-vector multiply
  - random number generation
- Work in progress:
  - parallel reduction
  - more sorts
  - graphs & trees
- Open Source under BSD License

Common Situations in Parallel Code

- Many threads need to generate a single result value
  - Reduce
- Many parallel threads that need to partition data
  - Split
- Many parallel threads and variable output per thread
  - Compact / Expand / Allocate

CUSP: Sparse Methods

- Generic data structures and algorithms
  - for sparse matrices
  - for sparse graphs
- Caution: very early in development
- Current functionality
  - several sparse matrix / graph storage formats
  - templated BLAS wrappers
  - high performance SpMV kernels (described in SC09 paper)
  - simple Conjugate Gradient solver
  - much more to come!

http://cusp-library.googlecode.com

Questions?

mgarland@nvidia.com

http://www.nvidia.com/research