CUDA Libraries & Tools

CUDA-Accelerated Libraries

- **CUDA C**
  - Over 60,000 developers
  - Released 2008
  - SDK
  - Libraries
  - Visual Profiler
  - Debugger
  - Nexus

- **OpenCL**
  - Shipped 1st OpenCL Conformant Driver

- **Direct Compute**
  - Microsoft's GPU Computing API
  - Supports all CUDA-Architecture GPUs since G80 (DX10 and future DX11 GPUs)

- **Fortran**
  - CUDA Fortran
  - PGI Accelerator
  - NOAA Fortran bindings (ftoc)
  - FLAGON

- **Python, Java .NET, ...**
  - PyCUDA
  - JaCUDA
  - CUDA.NET
  - BSGP
  - And more...

NVIDIA GPU with the CUDA Parallel Computing Architecture
Libraries
- CUBLAS
- CUFFT
- MAGMA
- CULA
- Thrust
- ...

Tools
- CUDA-gdb
- CUDA-Memcheck
- CUDA Visual Profiler
- Nexus
- ...

© NVIDIA Corporation 2009
CUDA Libraries
CUDA accelerated BLAS (Basic Linear Algebra Subprograms)

- Create matrix and vector objects in GPU memory space
- Fill objects with data
- Call sequence of CUBLAS functions
- Retrieve data from GPU (optionally)

```c
while( i++ < max_iter && deltanew > stop_tol )
{
    cublasSgemv('n', N, N, 1.0, d_A, N, d_d, 1, 0, d_y, 1);
    float alpha = deltanew / cublasSdot(N,d_d,1,d_y,1);
    cublasSaxpy(N, alpha,d_d,1,d_x,1);
    // every 50 iterations, restart residual
    if (i % 50 == 0) {
        cublasSgemv('n', N, N, 1.0, d_A, N, d_x, 1, 0, d_y, 1);
        cublasScopy(N, d_b, 1, d_r, 1);
        cublasSaxpy(N, -1.0, d_y, 1, d_r, 1);
    } else
        cublasSaxpy(N,-alpha,d_y,1,d_r,1);
...
CUBLAS Features

- 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
CUBLAS Performance: CPU vs GPU

CUBLAS: CUDA 2.3, Tesla C1060
MKL 10.0.3: Intel Core2 Extreme, 3.00GHz
CUFFT

- CUFFT is the CUDA FFT library
- Computes parallel FFT on an NVIDIA GPU
- Uses ‘Plans’ like FFTW
  - Plan contains information about optimal configuration for a given transform.
  - Plans can be persisted to prevent recalculation.
  - Good fit for CUFFT because different kinds of FFTs require different thread/block/grid configurations.
CUFFT 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.
```c
#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);

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

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

Complex 2D transform
CUFFT Performance: CPU vs GPU

**Single Precision FFT**

- cuFFT 2.3
- MKL 4 Threads
- FFTW 1 Thread

**Double Precision FFT**

- cuFFT 2.3
- MKL 4 Threads

**cuFFT 2.3:** NVIDIA Tesla C1060 GPU

**MKL 10.1r1:** Quad-Core Intel Core i7 (Nehalem) 3.2GHz
MAGMA: Matrix Algebra on GPU and Multicore Architectures

MAGMA and LAPACK
- MAGMA - based on LAPACK, extended for heterogeneous systems
- MAGMA - similar to LAPACK in functionality, data storage, interface

Features
- **Goal**: easy porting from LAPACK to take advantage of the new GPU + multicore architectures
- **Leverage**: experience developing open source Linear Algebra software (LAPACK, ScaLAPACK, BLAS, ATLAS)
- **Incorporate**: newest numerical developments (e.g. communication avoiding algorithms) and experiences on homogeneous multicores (e.g. PLASMA)

MAGMA Developers
- University of Tennessee, Knoxville
- University of California, Berkeley
- University of Colorado, Denver
- Number of contributors from the LA community

© NVIDIA Corporation 2009

Portions of this slide courtesy Stan Tomov
MAGMA version 0.1 (08/04/09)

- One-sided factorizations [for linear solvers] in single and double precision arithmetic
- Hardware target: 1 core + 1 GPU (CUDA enabled)

MAGMA version 0.2 (11/14/09)

- One-sided factorizations in complex arithmetic
- Two-sided factorizations for eigenvalue solvers
- Linear solvers, including least squares and mixed precision iterative solvers
- MAGMA BLAS (gemm optimized for rectangular matrices, triangular solvers, gemv, etc)
- Hardware target:
  - 1 core + 1 GPU (all)
  - multicore + multi-GPU (one-sided factorizations)

Portions of this slide courtesy Stan Tomov
MAGMA Version 0.1 Performance

QR factorization in single precision arithmetic, CPU interface
Performance of MAGMA vs MKL

For more performance data, see http://icl.cs.utk.edu/magma

Portions of this slide courtesy Stan Tomov
MAGMA Version 0.2 Performance

Linear Solvers
[e.g. $A x = b$ using LU Factorization]

Hessenberg factorization
[e.g. double precision, CPU interface]

GPU : NVIDIA GeForce GTX 280 (240 cores @ 1.30GHz)
CPU : Intel Xeon dual socket quad-core (8 cores @2.33 GHz)

GPU BLAS : CUBLAS 2.2, dgemm peak: 75 GFlop/s
CPU BLAS : MKL 10.0, dgemm peak: 65 GFlop/s

For more performance data, see http://icl.cs.utk.edu/magma

 Portions of this slide courtesy Stan Tomov
MAGMA Multi-GPU Performance

Cholesky factorization in single precision arithmetic
Performance and scalability on 4 GPUs

<table>
<thead>
<tr>
<th>Matrix size x 1000</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>18</th>
<th>20</th>
<th>22</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>GFlop/s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 CPU + 4 GPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 CPU + 3 GPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 CPU + 2 GPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 CPU + 1 GPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MAGNUM-tiles approach
- Communication-avoiding/tile type algorithms on large (magnum) tiles
- A magnum tile/task is defined for hybrid 1 CPU + 1 GPU computing
- PLASMA scheduling on the magnum tiles/tasks

GPU : NVIDIA Tesla C1070 (4 GPUs @1.44GHz)
CPU : AMD Opteron dual socket dual-core (4 cores @1.8 GHz)

For more performance data, see http://icl.cs.utk.edu/magma

Portions of this slide courtesy Stan Tomov
3rd Party Implementation of LAPACK interface from EM Photonics (www.culatools.com)

**CULA basic**
- Six popular single/complex-single LAPACK functions
- Free!

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getrf</td>
<td>LU decomposition</td>
</tr>
<tr>
<td>gesv</td>
<td>System solve</td>
</tr>
<tr>
<td>geqrf</td>
<td>QR factorization</td>
</tr>
<tr>
<td>gesvd</td>
<td>Singular value decomposition</td>
</tr>
<tr>
<td>gels</td>
<td>Least squares</td>
</tr>
<tr>
<td>gglse</td>
<td>Constrained least squares</td>
</tr>
</tbody>
</table>

**CULA premium**
- Available for purchase
- Adds 18 more routines (and growing)
- Adds Double (D) / Double Complex (Z)

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>potrf</td>
<td>Cholesky factorization</td>
</tr>
<tr>
<td>gebrd</td>
<td>Bidiagonalization</td>
</tr>
<tr>
<td>getri</td>
<td>Matrix inversion</td>
</tr>
<tr>
<td>gets</td>
<td>LU Backsolve</td>
</tr>
<tr>
<td>trtrs</td>
<td>Triangular solve</td>
</tr>
<tr>
<td>gelqf</td>
<td>LQ factorization</td>
</tr>
<tr>
<td>posv</td>
<td>Positive-definite system solve</td>
</tr>
</tbody>
</table>
CULA 1.0 vs Intel MKL 10.2

- gesvd: 7.2x
- geqrf: 5.3x
- gesv: 3.5x
- getrf: 4.2x
- gglse: 4.3x
- gels: 3.9x

Speed Up vs MKL

CULA 1.0 vs Netlib Reference LAPACK

- gesvd: 87.7x
- geqrf: 56.8x
- gesv: 77.1x
- getrf: 92.1x
- gglse: 77.5x
- gels: 63.6x

Speed Up vs Netlib

Tesla C1060 vs Intel Core i7, matrix size ~10,000x10,000

Courtesy EM Photonics
PyCUDA

- 3rd party open source, written by Andreas Klöckner
- Exposes all of CUDA via Python bindings
- Compiles CUDA on the fly
  - presents CUDA as an interpreted language
- Integration with numpy
- Handles memory management, resource allocation
- CUDA programs are Python strings
  - Metaprogramming – modify source code on-the-fly
  - Like a really complex pre-processor

http://mathema.tician.de/software/pycuda
import pycuda.driver as cuda
import pycuda.autoinit
import numpy

a = numpy.random.randn(4,4).astype(numpy.float32)
a_gpu = cuda.mem Alloc(a.size, a.dtype.itemsize)
cuda.memcpy_htod(a_gpu, a)

mod = cuda.SourceModule(""
__global__ void doublify(float *a)
{
    int idx = threadIdx.x + threadIdx.y*4;
    a[idx] *= 2.0f;
}
""
func = mod.get_function("doublify")
func(a_gpu, block=(4,4,1))

a_doubled = numpy.empty_like(a)
cuda.memcpy_dtoh(a_doubled, a_gpu)
print a_doubled
print a
Thrust – Deep Dive

“Standard Template Library for CUDA”
- From Nathan Bell and Jared Hoberock, NVIDIA Research
- Open source project (Apache 2.0 license)

Development
- > 1000 downloads (as of September)
- 460+ unit tests
- 35k lines of code

Uses CUDA Runtime API
- Heavy use of C++ templates for efficiency
What is Thrust?

- C++ template library for CUDA
  - Mimics Standard Template Library (STL)
- Containers
  - thrust::host_vector<T>
  - thrust::device_vector<T>
- Algorithms
  - thrust::sort()
  - thrust::reduce()
  - thrust::inclusive_scan()
  - thrust::segmented_inclusive_scan()
- Etc.
Containers

Make common operations concise and readable
- Hides cudaMalloc & cudaMemcpy

```cpp
// allocate host vector with two elements
thrust::host_vector<int> h_vec(2);

// copy host vector to device
thrust::device_vector<int> d_vec = h_vec;

// manipulate device values from the host
d_vec[0] = 13;
d_vec[1] = 27;

std::cout << "sum: " << d_vec[0] + d_vec[1] << std::endl;
```

© NVIDIA Corporation 2009
Containers

- Compatible with STL containers
  - Eases integration
  - vector, list, map, ...

```cpp
// list container on host
std::list<int> h_list;
h_list.push_back(13);
h_list.push_back(27);

// copy list to device vector
thrust::device_vector<int> d_vec(h_list.size());
thrust::copy(h_list.begin(), h_list.end(), d_vec.begin());

// alternative method
thrust::device_vector<int> d_vec(h_list.begin(), h_list.end());
```
Iterators

- Track memory space (host/device)
- Guides algorithm dispatch

```cpp
// initialize random values on host
thrust::host_vector<int> h_vec(1000);
thrust::generate(h_vec.begin(), h_vec.end(), rand);

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

// compute sum on host
int h_sum = thrust::reduce(h_vec.begin(), h_vec.end());

// compute sum on device
int d_sum = thrust::reduce(d_vec.begin(), d_vec.end());
```
Algorithms

- Thrust provides ~50 algorithms
  - Reduction
  - Prefix Sums
  - Sorting
  - Segmented Scan
  - Binary search, Lower/Upper bound
  - Parallel PRNG

- Generic meta programming
  - Support built-in or user-defined types
  - Customize algorithms with functors
    - Redefine “plus” for reduction / scan
    - Provide custom comparison for sort
Features & Optimizations

- **gather & scatter**
  - Works between host and device

- **fill & reduce**
  - Avoids G8x coalescing rules for char, short, etc.

- **sort**
  - Dispatches radix_sort for all primitive types
  - Uses optimal number of radix_sort iterations
  - Dispatches merge_sort for all other types
Example: 2D Bucket Sort

**Procedure:**

[Step 1] create random points
[Step 2] compute bucket index for each point
[Step 3] sort points by bucket index
[Step 4] compute bounds for each bucket
[Step 1] create random points

// number of points
const size_t N = 100000;

// return a random float2 in [0,1)^2
float2 make_random_float2(void)
{
    return make_float2( rand() / (RAND_MAX + 1.0f),
                        rand() / (RAND_MAX + 1.0f) );
}

// allocate some random points in the unit square on the host
host_vector<float2> h_points(N);
generate(h_points.begin(), h_points.end(), make_random_float2);

// transfer to device
device_vector<float2> points = h_points;
Example: 2D Bucket Sort

[Step 2] compute bucket index for each point

```c
struct point_to_bucket_index
{
    unsigned int w, h;

    __host__ __device__
    point_to_bucket_index(unsigned int width, unsigned int height)
    :w(width), h(height){}

    __host__ __device__
    unsigned int operator()(float2 p) const
    {
        // coordinates of the grid cell containing point p
        unsigned int x = p.x * w;
        unsigned int y = p.y * h;

        // return the bucket's linear index
        return y * w + x;
    }
};
```
Example: 2D Bucket Sort

[Step 2] compute bucket index for each point

// resolution of the 2D grid
unsigned int w = 200;
unsigned int h = 100;

// allocate storage for each point's bucket index
device_vector<unsigned int> bucket_indices(N);

// transform the points to their bucket indices
transform(points.begin(),
          points.end(),
          bucket_indices.begin(),
          point_to_bucket_index(w,h));
Example: 2D Bucket Sort

[Step 3] sort points by bucket index

// sort the points by their bucket index
sort_by_key(bucket_indices.begin(),
bucket_indices.end(),
points.begin());
Digression: counter iterators

**counting_iterator**
- “Pointer” to an infinite array with sequential values
- Lazily create array of values, without allocating them

```cpp
// create iterators
counting_iterator<int> first(1);
counting_iterator<int> last = first + 3;

first[0]  // returns 1
first[1]  // returns 2
first[100] // returns 101

// sum of [first, last)
reduce(first, last);  // returns 6 (i.e. 1 + 2 + 3)
```
Example: 2D Bucket Sort

[Step 4] compute bounds for each bucket

// bucket_begin[i] indexes the first element of bucket i
// bucket_end[i] indexes one past the last element of bucket i
device_vector<unsigned int> bucket_begin(w*h);
device_vector<unsigned int> bucket_end(w*h);

// used to produce integers in the range [0, w*h]
counting_iterator<unsigned int> search_begin(0);

// find the beginning of each bucket's list of points
lower_bound(bucket_indices.begin(), bucket_indices.end(),
            search_begin, search_begin + w*h, bucket_begin.begin());

// find the end of each bucket's list of points
upper_bound(bucket_indices.begin(), bucket_indices.end(),
            search_begin, search_begin + w*h, bucket_end.begin());
Example: 2D Bucket Sort

[Result]

// print all points in bucket 10,25
int i = 10, j=25;
int linear_index = i*w + j;

// implicitly transfers ints from device to host:
int first_id_in_bucket = bucket_begin[linear_index];
int last_id_in_bucket = bucket_end[linear_index];

for (int id = first_id_in_bucket; id < last_id_in_bucket; id++)
{
    // implicitly transfers float2 from device to host
    float2 pt = points[id];
    printf("%f, %f\n", pt.x, pt.y);
}
CUDA Development Tools
CUDA-gdb

- Integrated into gdb
- Supports CUDA C
- Seamless CPU+GPU development experience
- Enabled on all CUDA supported 32/64bit Linux distros
- Set breakpoint and single step any source line
- Access and print all CUDA memory allocs, local, global, constant and shared vars.
```c
} else {
  acos_noftz_main<<<ACOS_CTA_CNT,ACOS_THREAD_CNT>>>(funcParams);
}
} else {
  acos_ieee3_ftz_main<<<ACOS_CTA_CNT,ACOS_THREAD_CNT>>>(funcParams);
} else if (opts.ieee == 2) {
  acos_ieee2_ftz_main<<<ACOS_CTA_CNT,ACOS_THREAD_CNT>>>(funcParams);
} else if (opts.ieee == 1) {
  acos_ieee1_ftz_main<<<ACOS_CTA_CNT,ACOS_THREAD_CNT>>>(funcParams);
} else {
  acos_main<<<ACOS_CTA_CNT,ACOS_THREAD_CNT>>>(funcParams);
}

#endif

__device_func__ (float __cuda_acosf (float a))
{
  float t0, t1, t2;

  t0 = __cuda_fabsf (a);
  t2 = 1.0f - t0;
  t2 = 0.5f * t2;
  t2 = __cuda_sqrtf (t2);
  t1 = t0 > 0.50f ? 2.0f * t1 : 1.0f;
  if (__cuda_signbitf (a))
    t1 = __internal_asinf_device (t1);
  if (!defined (__CUDA_BE__))
    if (__cuda_isnanf (a))
      t1 = a + a;

  return t1;
}
```
/* ------------------------- target code -------------------------*/

global void acos_main (struct acosParams params)
{
    int i;
    int totalThreads = gridDim.x * blockDim.x;
    int ctaStart = blockDim.x * blockIdx.x;
    for (i = ctaStart + threadIdx.x; i < params.n; i += totalThreads) {
        params.res[i] = acosl(params.arg[i]);
    }
}

Breakpoint 2 at 0x8073b40: file acos.cu, line 390.
[Switching to Thread -1211672896 (LWP 28236)]
[Current CUDA Thread <<(0,0),(0,0,0)>>]

Breakpoint 1, acos_main () at acos.cu:389
(gdb) step
[Current CUDA Thread <<(0,0),(0,0,0)>>]

Breakpoint 2, acos_main () at acos.cu:390
(gdb) graph display totalThreads
(gdb) graph display blockDim
(gdb) graph display threadIdx
(gdb)
CUDA-MemCheck

- Coming with CUDA 3.0 Release
- Track out of bounds and misaligned accesses
- Supports CUDA C
- Integrated into the CUDA-GDB debugger
- Available as standalone tool on all OS platforms.
Parallel Source Memory Checker
CUDA-MemCheck

unspecified launch failure: 125
Invalid read of size 4
    at 0x000000f0 in kernel2 (/src/gpgpu/cudamemcheck/test/ptrchecktest.cu:27)
    by thread 5 in block 3
Address 0x00101015 is misaligned

Invalid read of size 4
    at 0x000000f0 in kernel1 (/src/gpgpu/cudamemcheck/test/ptrchecktest.cu:18)
    by thread 3 in block 5
Address 0x00101028 is out of bounds

Invalid write of size 8
    at 0x000000170 in kernel3 (/src/gpgpu/cudamemcheck/test/ptrchecktest.cu:38)
    by thread 1 in block 8
Address 0x00102004 is misaligned

Invalid write of size 4
    at 0x000000a0 in kernel4 (/src/gpgpu/cudamemcheck/test/ptrchecktest.cu:44)
    by thread 63 in block 22
Address 0x00000000 is out of bounds

ERROR SUMMARY: 4 errors
CUDA Visual Profiler
CUDA Visual Profiler Signals

Events are tracked with hardware counters on signals in the chip:

- **timestamp**
- **gld_incoherent**
- **gld_coherent**
- **gst_incoherent**
- **gst_coherent**
- **local_load**
- **local_store**
- **branch**
- **divergent_branch**
- **instructions** – instruction count
- **warp_serialize** – thread warps that serialize on address conflicts to shared or constant memory
- **cta_launched** – executed thread blocks

Global memory loads/stores are coalesced (coherent) or non-coalesced (incoherent) (Compute 1.0/1.1)

Local loads/stores

Total branches and divergent branches taken by threads
NVIDIA IDE: code name “Nexus”

The first development environment for massively parallel applications.

- **Hardware** GPU Source Debugging
- **Platform-wide** Analysis
- **Complete** Visual Studio-integration

Register for the Beta!

Releasing in Q1 2010
For more on Nexus

A Powerful IDE for GPU Computing on Windows: Code Named Nexus

At the NVIDIA Booth
- Tue 2:00 - 2:30
- Wed 1:30 - 2:00
- Thur 12:00 - 12:30
Thanks

- Stan Tomov & MAGMA development team
- EM Photonics
- Andreas Klöckner
- Jared Hoberock and Nathan Bell

http://www.nvidia.com/CUDA