Introduction to GPGPUs and to CUDA programming model: CUDA Libraries

Marzia Rivi
m.rivi@cineca.it

www.cineca.it

NVIDIA CUDA Libraries

http://developer.nvidia.com/technologies/Libraries

CUDA Toolkit includes several libraries:

- **CUFFT**: Fast Fourier Transforms
- **CUBLAS**: Dense Linear Algebra
- **CUSPARSE** : Sparse Linear Algebra
- **LIBM**: Standard C Math library
- **CURAND**: Pseudo-random and Quasi-random numbers
- **NPP**: Image and Signal Processing
- **Thrust** : C++ Template Library

Several open source and commercial* libraries:

- **MAGMA**: Linear Algebra
- **CULA Tools***: Linear Algebra
- **CUSP**: Sparse Linear Solvers ……..
- **NAG***: Computational Finance

CUFFT Library

GPU based Fast Fourier Transform library

- Simple interface similar to FFTW
- 1D, 2D and 3D transforms of complex and real data
- Row-major order (C-order) for 2D and 3D data
- Single precision (SP) and Double precision (DP) transforms
- In-place and out-of-place transforms
- 1D transform sizes up to 128 million elements
- Batch execution for doing multiple transforms
- Streamed asynchronous execution
- Non normalized output: IFFT(FFT(A))=len(A)*A

CUFFT in 4 easy steps

Step 1 – Allocate space on GPU memory

Step 2 – Create plan specifying transform configuration like the size and type (real, complex, 1D, 2D and so on).

Step 3 – Execute the plan as many times as required, providing the pointer to the GPU data created in Step 1.

Step 4 – Destroy plan, free GPU memory

CUFFT: code example

```
#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.
 Different pointers to input and output arrays implies out of place transformation  */
 cufftExecC2C(plan, odata, odata, CUFFT_INVERSE);
....
 /* Destroy the CUFFT plan. */
   cufftDestroy(plan);

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

CUBLAS Library

Implementation of BLAS (Basic Linear Algebra Subprograms)

- Self-contained at the API level

Supports all the BLAS functions

- Level1 (vector,vector): O(N)
  - AXPY : y = alpha.x + y
  - DOT : dot = x.y
- Level 2(matrix,vector): O(N2)
  - Vector multiplication by a General Matrix : GEMV
  - Triangular solver : TRSV
- Level3(matrix,matrix): O(N3)
  - General Matrix Multiplication : GEMM
  - Triangular Solver : TRSM

Following BLAS convention, CUBLAS uses column-major storage

Interface to CUBLAS library is in cublas.h
Function naming convention: cublas + BLAS name
Eg. cublasSGEMM

Lapack for CUDA: CULA Library

http://www.culatools.com

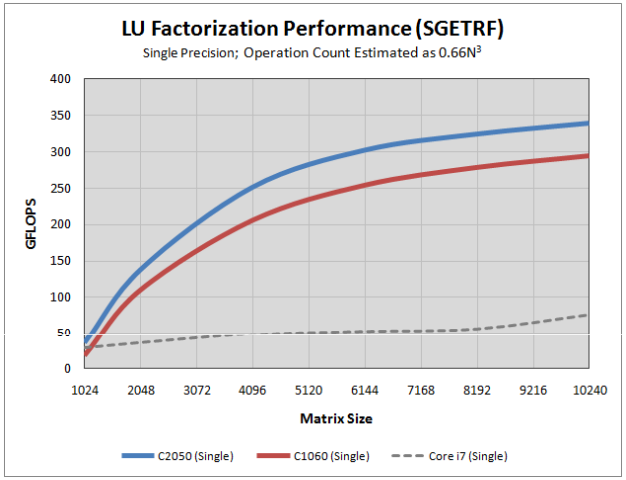
Proprietary library that implements the LAPACK in CUDA, which is available in several versions.

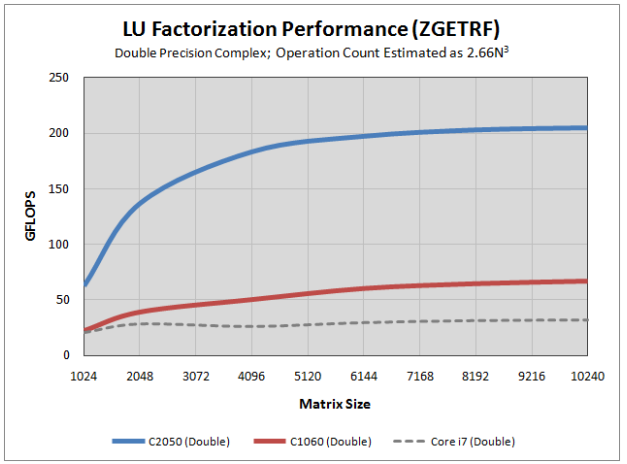
The speed-up of the picture on the right refers to:

CPU: *Quad-core Intel Core i7 930 @ 2.8 GHZ CPU*

GPU: *NVIDIA Tesla C1060*

OS: *Windows 7 (64-bit)*





CUSPARSE Library

- New library for sparse basic linear algebra
- Conversion routines for dense, COO, CSR and CSC formats
- Optimized sparse matrix-vector multiplication
- Building block for sparse linear solvers

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \alpha \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix} \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{bmatrix} + \beta \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

CUDA Libm

High performance and high accuracy implementation:

- C99 compatible math library, plus extras
- Basic ops: x+y, x*y, x/y, 1/x, sqrt(x), FMA (IEEE-754 accurate in single, double)
- Exponentials: exp, exp2, log, log2, log10, ...
- Trigonometry: sin, cos, tan, asin, acos, atan2, sinh, cosh, asinh, acosh, ...
- Special functions: lgamma, tgamma, erf, erfc
- Utility: fmod, remquo, modf, trunc, round, ceil, floor, fabs, ...
- Extras: rsqrt, rcbrt, exp10, sinpi, sincos, erfinv, erfcinv, ...

CURAND Library

Library for generating random numbers

Features:

- XORWOW pseudo-random generator
- Sobol' quasi-random number generators
- Single- and double-precision, uniform, normal and log-normal distributions
- Host API: call kernel from host, generate numbers on GPU, consume numbers on host or on GPU.
- GPU API: generate and consume numbers during kernel execution.

CURAND use

1. **Create a generator:**
   curandCreateGenerator()
2. **Set a seed:**
   curandSetPseudoRandomGeneratorSeed()
3. **Generate the data from a distribution:**
   curandGenerateUniform()/curandGenerateUniformDouble() : Uniform
   curandGenerateNormal()/cuRandGenerateNormalDouble() : Gaussian
   curandGenerateLogNormal/curandGenerateLogNormalDouble() : Log-Normal
4. **Destroy the generator:**
   curandDestroyGenerator()

CURAND example: host API

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include <curand.h>

main()
{
    int i, n = 100;
    curandGenerator_t gen;
    float *devData, *hostData;

    /* Allocate n floats on host */
    hostData = (float *)calloc(n, sizeof(float));

    /* Allocate n floats on device */
    cudaMalloc((void **)&devData, n * sizeof(float));

    /* Create pseudo-random number generator */
    curandCreateGenerator(&gen, CURAND_RNG_PSEUDO_DEFAULT);

    /* Set seed */
    curandSetPseudoRandomGeneratorSeed(gen, 1234ULL);
    /* Generate n floats on device */
    curandGenerateUniform(gen, devData, n);
    /* Copy device memory to host */
    cudaMemcpy(hostData, devData, n * sizeof(float),
                    cudaMemcpyDeviceToHost);

    /* Show result */
    for(i = 0; i < n; i++) {
        printf("%1.4f ", hostData[i]);
    }
    printf("\n");

    /* Cleanup */
    curandDestroyGenerator(gen);
    cudaFree(devData);
    free(hostData);

    return 0;
}
```

CURAND example: run on CPU

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include <curand.h>

main()
{
    int i, n = 100;
    curandGenerator_t gen;
    float *hostData;

    /* Allocate n floats on host */
    hostData = (float *)calloc(n, sizeof(float));

    /* Create pseudo-random number generator */
    curandCreateGeneratorHost(&gen,
        CURAND_RNG_PSEUDO_DEFAULT);

    /* Set seed */
    curandSetPseudoRandomGeneratorSeed(gen, 1234ULL);
    /* Generate n floats on host */
    curandGenerateUniform(gen, hostData, n);

    /* Show result */
    for(i = 0; i < n; i++) {
        printf("%1.4f ", hostData[i]);
    }
    printf("\n");

    /* Cleanup */
    curandDestroyGenerator(gen);
    free(hostData);

    return 0;
}
```

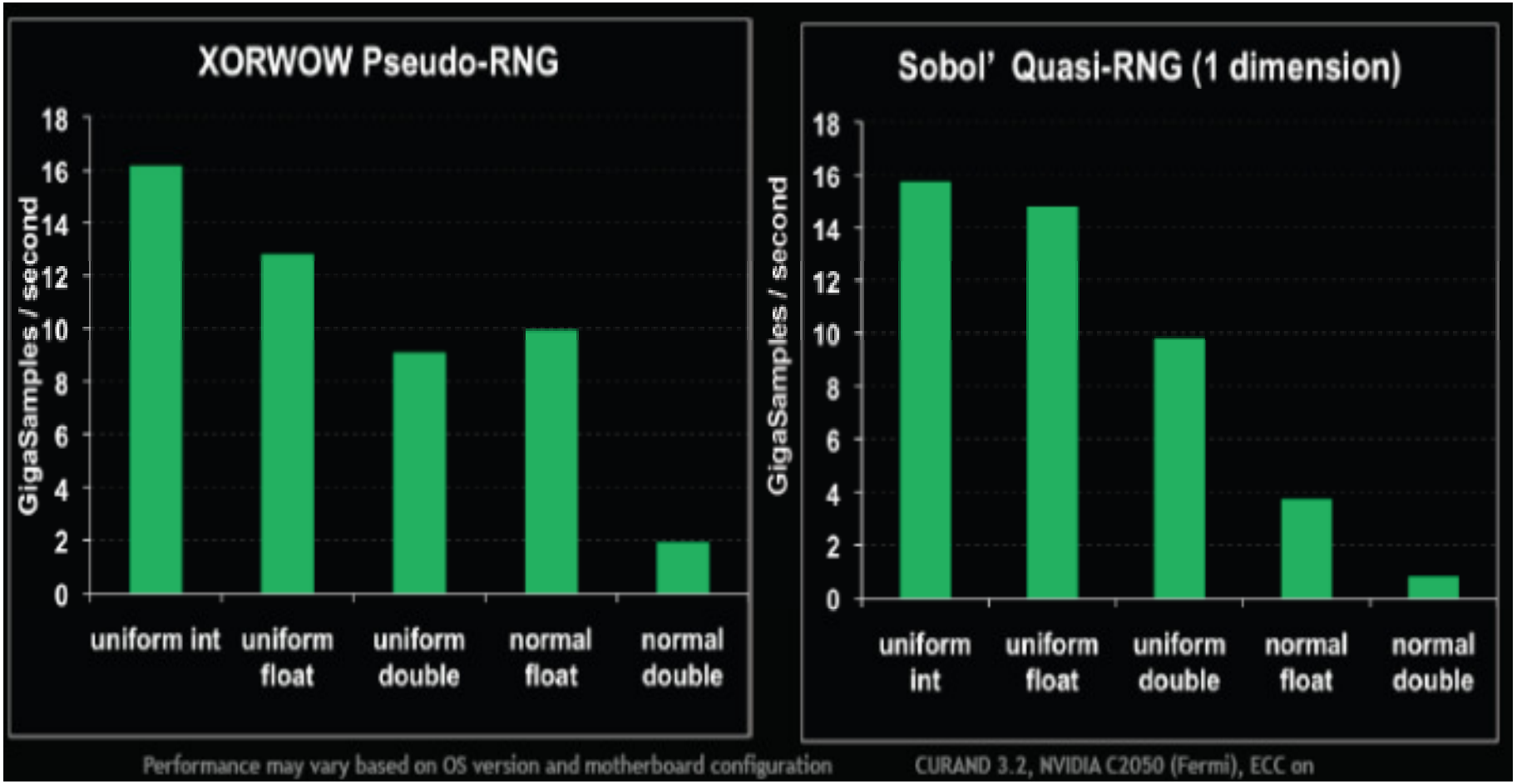
CURAND example: device API

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include <curand_kernel.h>

__global__ void setup_kernel(curandState *state)
{
    int id = threadIdx.x + blockIdx.x * 64;
    /* Each thread gets same seed,
       a different sequence number, no offset */
    curand_init(1234, id, 0, &state[id]);
}

__global__ void generate_kernel(curandState *state, int *result)
{
    int id = threadIdx.x + blockIdx.x * 64;
    int count = 0;
    unsigned int x;
    /* Copy state to local memory for efficiency */
    curandState localState = state[id];

    /* Generate pseudo-random unsigned ints */
    for(int n = 0; n < 100000; n++) {
        x = curand(&localState);
        /* Check if low bit set */
        if(x & 1) count++;
    }

    /* Copy state back to global memory */
    state[id] = localState;

    /* Store results */
    result[id] += count;
}
```

CURAND example: device API

```
main()
{
    int i, total;
    curandState *devStates;
    int *devResults, *hostResults;

    /* Allocate space for results on host */
    hostResults = (int *)calloc(64 * 64, sizeof(int));

    /* Allocate space for results on device */
    cudaMalloc((void **)&devResults, 64 * 64 * sizeof(int));

    /* Set results to 0 */
    cudaMemset(devResults, 0, 64 * 64 * sizeof(int));

    /* Allocate space for prng states on device */
    cudaMalloc((void **)&devStates, 64 * 64 * sizeof(curandState));

    /* Setup prng states */
    setup_kernel<<<64, 64>>>(devStates);

    /* Generate and use pseudo-random */
    for(i = 0; i < 10; i++) {
        generate_kernel<<<64, 64>>>(devStates, devResults);
    }

    /* Copy device memory to host */
    cudaMemcpy(hostResults, devResults, 64 * 64 * sizeof(int),
                cudaMemcpyDeviceToHost);

    /* Show result */
    total = 0;
    for(i = 0; i < 64 * 64; i++) {
        total += hostResults[i];
    }
    printf("Fraction with low bit set was %10.13f\n",
    (float)total / (64.0f * 64.0f * 100000.0f * 10.0f));

    /* Cleanup */
    cudaFree(devStates);
    cudaFree(devResults);
    free(hostResults);

    return 0;
}
```

CURAND performance



Thrust library

A template library for CUDA
- Mimics the C++ STL

- Containers
  - Manage memory on host and device:

    thrust::host_vector<T>
    thrust::device_vector<T>

- Algorithms
  - Sorting, reduction, scan, etc:

    thrust::sort()
    thrust::reduce()
    thrust::inclusive_scan()
  - act on ranges of the container data

Thrust example

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/sort.h>
#include <cstdlib.h>

int main(void)
{
    // generate 32M random numbers on the host
    thrust::host_vector<int> h_vec(32 << 20);
    thrust::generate(h_vec.begin(), h_vec.end(), rand);

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

    // sort data on the device (846M keys per sec on GeForce GTX 480)
    thrust::sort(d_vec.begin(), d_vec.end());

    // transfer data back to host
    thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());

    return 0;
}
```

Interoperability (from Thrust to C/CUDA)

Convert iterators to raw pointers

```
// allocate device vector
thrust::device_vector<int> d_vec(4);

// obtain raw pointer to device vector's memory
int * ptr = thrust::raw_pointer_cast(&d_vec[0]);

// use ptr in a CUDA C kernel
my_kernel<<< N / 256, 256 >>>(N, ptr);

// Note: ptr cannot be dereferenced on the host!
```

Interoperability (from Thrust to C/CUDA)

Wrap raw pointers with device_ptr

```
// raw pointer to device memory
int * raw_ptr;
cudaMalloc((void **) &raw_ptr, N * sizeof int));

// wrap raw pointer with a device_ptr
device_ptr<int> dev_ptr(raw_ptr);

// use device_ptr in thrust algorithms
fill(dev_ptr, dev_ptr + N, (int) 0);

// access device memory through device_ptr
dev_ptr[0] = 1;

// free memory
cudaFree(raw_ptr);
```

code.google.com/p/thrust

Quick Start Guide

Examples

Documentation

Mailing List (thrust-users)

thrust
Code at the speed of light

Search projects

Project Home | Downloads | Wiki | Issues | Source | Administer

Summary | Updates | People

What is Thrust?

Thrust is a CUDA library of parallel algorithms with an interface resembling the C++ Standard Template Library (STL). Thrust provides a flexible **high-level** interface for GPU programming that greatly enhances developer **productivity**. Develop **high-performance** applications rapidly with Thrust!

News

- **Thrust v1.2.1 has been released!** v1.2.1 contains compatibility fixes for CUDA 3.1.
- Posted An Introduction to Thrust presentation.
- Thrust v1.2 has been released! Refer to the CHANGELOG for changes since v1.1.
- A video recording of the Thrust presentation at the GPU Technology Conference has been posted.
- Thrust v1.1 has been released! Refer to the CHANGELOG for changes since v1.0.
- Started Thrust Developer Blog

Examples

Thrust is best explained through examples. The following source code generates random numbers on the host and transfers them to the device where they are sorted.

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/sort.h>
#include <thrust/copy.h>
#include <cstdlib>

int main(void)
{
    // generate 32M random numbers on the host
    thrust::host_vector<int> h_vec(1 << 24);
    thrust::generate(h_vec.begin(), h_vec.end(), rand);

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

    // sort data on the device (805M keys per second on a GTX 480)
    thrust::sort(d_vec.begin(), d_vec.end());

    // transfer data back to host
    thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());

    return 0;
}
```

Starred (view starred projects)

Activity: High

Code license:
Apache License 2.0

Labels:
CUDA, GPU, Parallel, Template, nvcc, CPlusPlus, STL, Library, GPGPU, Sorting, Reduction, Scan

Featured downloads:
An Introduction To Thrust.pdf
thrust-v1.2.1.zip
Show all »

Featured wiki pages:
Documentation
FrequentlyAskedQuestions
QuickStartGuide
Show all »

Blogs:
MegaNewtons

External links:
CUDA

Feeds:
Project feeds

Groups:
Thrust User Discussion List

Owners:
wnbell, jaredhoberock

Committers:
duane.merrill

People details »