Лекция 5

Прикладные CUDA библиотеки

Библиотеки NVIDIA

http://developer.nvidia.com/gpu-accelerated-libraries

CUDA Toolkit

- CUBLAS – линейная алгебра с плотными матрицами
- CUSPARSE – линейная алгебра с разреженными матрицами
- CUFFT – преобразование Фурье
- CURAND – генерация случайных чисел

А также:

- Thrust – библиотека шаблонов (~STL)
- NPP – обработка сигналов, изображений

Сторонние библиотеки

MAGMA – линейная алгебра с плотными матрицами, открытый аналог LAPACK с расширениями

CUSP – линейная алгебра с разреженными матрицами и итеративные решатели систем уравнений

OpenCurrent – решатели уравнений в частных производных на регулярных сетках

Accelereyes ArrayFire

CURAND – общие положения

Библиотека для генерации псевдослучайных и квазислучайных чисел.

Работает как на стороне GPU так и на стороне центрального процессора

При одних и тех же начальных условиях (тип генератора, seed и пр.) последовательности чисел будут одинаковые. Как на GPU так и на CPU.

CURAND - использование

1. Создать генератор curandCreateGenerator(), curandCreateGeneratorHost().
2. Установить параметры.
3. Выделить память под результат cudaMalloc(), cudaMallocHost().
4. Сгенерировать последовательность.
5. Использовать результат.
6. При необходимости вернутся к пункту 4.
7. Освободить память.
8. Удалить генератор curandDestroyGenerator().

Параметры CURAND

Тип

- CURAND_RNG_PSEUDO_XORWOW
- CURAND_RNG_PSEUDO_MRG32K3A
- CURAND_RNG_PSEUDO_MTGP32
- CURAND_RNG_QUASI_SOBOL32
- CURAND_RNG_QUASI_SCRAMBLED_SOBOL32
- CURAND_RNG_QUASI_SOBOL64
- CURAND_RNG_QUASI_SCRAMBLED_SOBOL64

Опции

- Seed (инициализация, начальное значение)
- Offset (смещение от начала исходной последовательности)
- Ordering (расположение чисел в памяти)
  - CURAND_ORDERING_PSEUDO_DEFAULT
  - CURAND_ORDERING_PSEUDO_BEST
  - CURAND_ORDERING_QUASI_DEFAULT

Функции генерации CURAND

curandGenerate — 32-bit unsigned int

curandGenerateUniform — равномерное распределение

float (0.0, 1.0]

curandGenerateNormal — нормальное распределение с заданными средним значением и стандартным отклонением

float

curandGenerateUniformDouble - равномерное распределение

double (0.0, 1.0]

curandGenerateNormalDouble — нормальное распределение с заданными средним значением и стандартным отклонением

double

Функции генерации CURAND

```
#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;
}
```

Пример CURAND (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;
}
```

Пример CURAND (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;
}
```

Пример CURAND (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 rng 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;
}
```

CUBLAS

Реализация интерфейса BLAS

Многомерные массивы выделяются по столбцам (Fortran)

Документация входит в состав CUDA Toolkit

Уровень	Сложность	Примеры функций
1 (операции с векторами)	$O(n)$	AXPY: $y = ax + y$ DOT: $s = (x, y)$
2 (матрица-вектор)	$O(n^2)$	GEMV – произведение матрицы и вектора
3 (матрица-матрица)	$O(n^3)$	GEMM – произведение двух матриц

CUBLAS

- Именование функций: cublas<T><func>
  - <T> - тип данных (S, D, C, Z) соответствует вещественным числам с одинарной и двойной точностью и комплексным числам с одинарной и двойной точностью
  - <func> - 3x-4x буквенное название функции BLAS
  - Пример: cublasDgemm

В API v.2 (CUDA 4.0+) используются дескрипторы для того, чтобы библиотека была потокобезопасной.

CUBLAS – сценарий использования

Инициализировать дескриптор (`cublasCreate()`)

Выделить память и загрузить данные

Вызвать последовательность функций CUBLAS

Скопировать результат в оперативную память

Освободить дескриптор (`cublasDestroy()`)

Пример использования CUBLAS

(перемножение матриц)

```
#include <stdlib.h>
#include <stdio.h>
#include "cublas.h"
main ()
{
    float *a, *b, *c;
    float *d_a, *d_b, *d_c;
    int lda, ldb, ldc;
    int i, j, n;
    struct timeval t1, t2, t3, t4;
    double dt1, dt2, flops;
    /* CUBLAS initialization */
    cublasInit();
    printf(" n t1 t2 GF/s GF/s\n");

    for (n=512; n<5120; n*=512) {
        lda = ldb = ldc = 2*n;
```

```
/* Host page-locked memory allocation*/
cudaMallocHost( (void**)&a, n * lda * sizeof(float) );
cudaMallocHost( (void**|&b, n * ldb * sizeof(float) );
cudaMallocHost( (void**)&c, n * ldc * sizeof(float) );


/*  Filling the matrices  */
for( j = 0; j < n;  j++) {
    for( i = 0; i < n; i++) {
        a[i + j * lda] = (float)rand() / (float)RAND_MAX;
        b[i + j * ldb] = (float)rand() / (float)RAND_MAX;
        c[i + j * ldc] = (float)rand() / (float)RAND_MAX;

    }
}
/* Allocating device memory */
cublasAlloc( n * lda, sizeof(float), (void**)&d_a );
cublasAlloc( n * ldb, sizeof(float), (void**)&d_b );
cublasAlloc( n * ldc, sizeof(float), (void**)&d_c );
```

Пример использования CUBLAS

(перемножение матриц)

```
/* Copying data from host to device */
gettimeofday (&t1, NULL);
cublasSetMatrix( n, n, sizeof(float), a, lda, d_a, lda);
cublasSetMatrix( n, n, sizeof(float), b, ldb, d_b, ldb);
gettimeofday (&t2, NULL);
/* Performing matrix multiplication */
cublasSgemm('N', 'N', n, n, n, 1.0, d_a, lda, d_b, ldb, 0.0, d_c, ldc);
/* Waiting for multiplication finish */
cudaDeviceSynchronize();
/* Copying data back to host */
gettimeofday (&t3, NULL);
cublasGetMatrix( n, n, sizeof(float), d_c, ldc, c, ldc);
gettimeofday (&t4, NULL);
/* Clean up */
cublasFree( d_a);
cublasFree( d_b);
cublasFree( d_c);
```

```
    cudaFreeHost(a);
    cudaFreeHost(b);
    cudaFreeHost(c);

    tdiff1 = t4.tv_sec - t1.tv_sec + 1.0e-6 * (t4.tv_usec – t1.tv_usec);
    tdiff2 = t3.tv_sec - t2.tv_sec + 1.0e-6 * (t3.tv_usec - t2.tv_usec);
    flops = 2.0 * (double)n * (double)n * (double)n;
    /* Printing execution time */
    printf( "%4d %8.5f %8.5f %5.0f %5.0f\n", n, dt1, dt2,
            1.0e-9 * flops / tdiff1,1.0e-9 * flops/tdiff2);
  }

  cublasShutdown();
  return 0;
}
```

CUSPARSE

Работа с разреженными матрицами и векторами

«Разреженные» = большое количество нулевых элементов

- Стандартное «плотное» представление неэффективно
- Используются индексные форматы
  - Разреженный вектор: массив индексов и массив значений
  - Разреженная матрица: координатный формат (COO), сжатые форматы (CSR, CSC)

CUSPARSE - особенности

4 группы функций: cusparse<T><func>

- Над разреженными и плотными векторами
- Над разреженными матрицами и векторами
- Над разреженными матрицами и несколькими векторами
- Смена форматов

Поддержка вещественных и комплексных чисел одинарной и двойной точности.

CUSPARSE – сценарий использования

Инициализировать дескриптор (`cusparseCreate()`)

Выделить память и загрузить данные

Вызвать последовательность функций CUSPARSE

Скопировать результат в оперативную память

Освободить дескриптор (`cusparseDestroy()`)

CUFFT

Интерфейс аналогичен FFTW, поддерживает режим совместимости с FFTW

1d, 2d и 3d вещественные и комплексные преобразования одинарной и двойной точности

Поддерживаются асинхронные преобразования с использованием CUDA Streams

CUFFT - пример

(прямое и обратное преобразование Фурье над комплексными данными)

```
#include <stdlib.h>
#include <stdio.h>
#include "cufft.h"

#define NX 256
#define NY 128

main()
{
        cufftHandle plan;
        cufftComplex *idata, *odata;
        int i;

        cudaMalloc((void**)&idata, sizeof(cufftComplex)*NX*NY);
        cudaMalloc((void**)&odata)sizeof(cufftComplex)*NX*NY);
        for( i=0; i<NX*NY;i++){
              idata[i].x = (float)rand() / (float)RAND_MAX;
              idata[i].y = (float)rand() / (float)RAND_MAX;
        }

        /* Create a 2D FFT plan*/
        cufftPlan2d(&plan, NX, NY, CUFFT_C2C);

        /* Use the CUFFT plan to transform the signal out of
        place.
        * Note: idata != odata indicates an out of place
        * transformation to CUFFT at execution time. */
        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);
        return 0;
}
```

NPP: Image & Signal Processing

-идентично Intel® Integrated Performance Primitives (Intel® IPP);
- арифметические и логические операции;
- конвертация в другие цветовые модели;
- сжатие;
- фильтры;
- геометрические преобразования;
- статистика.

ArrayFire

A comprehensive GPU matrix library:

- Linear Algebra
- Signal&image processing
- Statistics
- Code timing
- Graphics

Unified array container type:

- Single/Double Real/Complex [Un]signed + Boolean
- Easy index manipulation (Matlab-like)
- Parallel for loops and multi-gpu scaling

ArrayFire

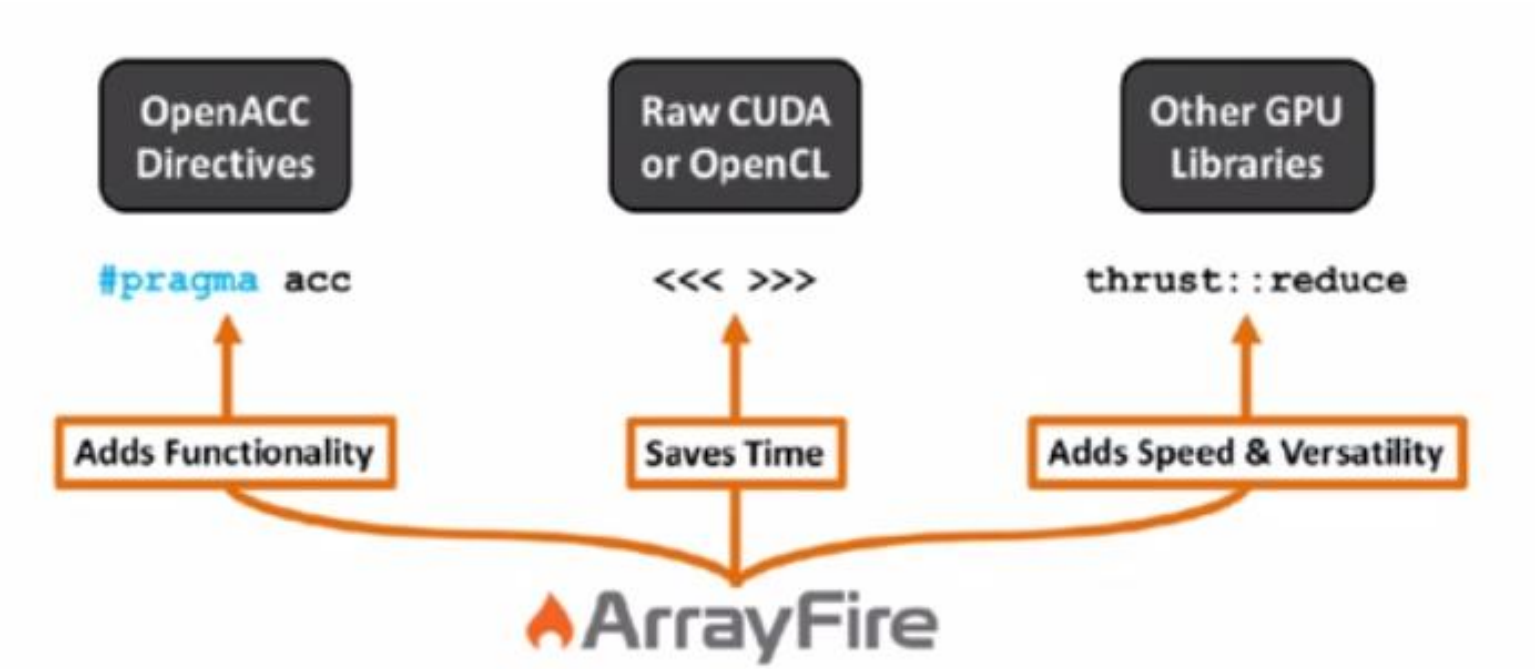
(подсчет числа Пи методом Монте-Карло)

Example: Pi value

```
#include <stdio.h>
#include <arrayfire.h>
using namespace af;
int main()
{
   // 20 million random samples
   int n = 20e6;
   array x = randu(n,1), y = randu(n,1);
   // how many fell inside unit circle?
   float pi = 4 * sum<float>(sqrt(mul(x,x)+mul(y,y))<1) / n;
   printf("pi = %g\n", pi);
   return 0;
}
```

24 AI

ArrayFire


OpenACC Directives
Raw CUDA or OpenCL
Other GPU Libraries
#pragma acc
<<< >>>
thrust::reduce
Adds Functionality
Saves Time
Adds Speed & Versatility
ArrayFire

Выводы

– Если вы не специалист в какой-то области – используйте библиотеки;

– Если вы специалист в какой-то области – сначала используйте библиотеки;

– Не расстраивайтесь, если вы не можете написать код какой-либо функции эффективнее, чем она представлена в библиотеке