1. **Для чего используются функции, предоставляемые библиотекой cuRAND?**

**cuRAND** – оптимизированная библиотека для генерирования псевдо - и квазислучайных чисел на хосте и GPU.

Функции используются для:

1. Генерация псевдослучайных чисел;
2. Возможность API вызов на хосте для генерации случайных чисел;
3. Возможность использовать внутри GPU функций/ядер;
4. Равномерные, нормальные и логнормальные распределения одинарной и двойной точности;
5. **Из каких частей состоит библиотека cuRAND?**

Библиотека cuRAND состоит из 2 частей:

1. Host API: библиотека функций, вызываемых со стороны хоста, генерация может осуществляться как на хосте, так и на GPU;
2. Device API: набор device-функций для вызовов из ядер;
3. **Какие функции имплементированы в библиотеке cuBLAS?**

Базовые функции линейной алгебры, состоящие из трёх уровней:

1 уровень: Векторные операции;

2 уровень: Векторно-матричные операции;

3 уровень: Матричные операции.

1. **Какие подмножества API реализованы в библиотеке cuBLAS? В чём их отличия друг от друга?**

* The **cuBLAS API**
* The **CUBLASXT API** (starting with CUDA 6.0)
* The **cuBLASLt API** (starting with CUDA 10.1)

Для использования **cuBLAS API**, программа должна хранить матрицы и вектора в памяти GPU, заполнять их данными, вызывать последовательность cuBLAS функций, и загружать результат из памяти GPU в память хоста. cuBLAS API также предоставляет функции для записи и чтения данных с GPU.

Для использования **CUBLASXT API**, программа должна хранить данные на хосте, и библиотека позаботится о распределении операций на один или несколько GPU, присутствующих в системе, в соответствии с требованиями пользователя.

**cuBLASLt** это легковесная библиотека, предназначенная для GEneral Matrix-to-matrix Multiply (GEMM) операций с новым вариативным API. Эта библиотека добавляет вариативности в расположении матричных данных, типах ввода, вычислительных типах, и также в выборе алгоритмических реализаций.

1. **Что представляет собой библиотека Thrust**

*Библиотека Thrust* - готовая библиотека шаблонов для CUDA, реализованная на основе стандартной библиотеки шаблонов (STL). Библиотека Thrust позволяет реализовывать высоко производительные параллельные приложения с минимальными усилиями программирования.

Библиотека *Thrust* предоставляет богатую коллекцию параллельных примитивов, таких как *scan*, *sort* и *reduce*, которые можно всячески комбинировать, чтобы реализовать сложные алгоритмы с кратким, читаемым исходным кодом.

Библиотеку *Thrust* можно разделить на 3 раздела:

* [Операции над векторами](http://it.kgsu.ru/C_CUDA/c_cu022.html)
* [Алгоритмы](http://it.kgsu.ru/C_CUDA/c_cu023.html)
* [Итераторы](http://it.kgsu.ru/C_CUDA/c_cu024.html)

Рассмотрим каждую из них.

1. [**Операции над векторами**](http://it.kgsu.ru/C_CUDA/c_cu022.html).

В *библиотеке Thrust* содержатся два вектора *host\_vector* и *device\_vector*.

*host\_vector* хранится в памяти компьютера, a *device\_vector* хранится в памяти GPU устройства. Данные векторы могут хранить данные любого типа и быть изменены динамически.

1. **Алгоритмы.**

В *библиотеке Thrust* реализованы следующие алгоритмы:

* Трансформация (Transformations);
* Редукция (Reductions);
* Нахождение префикс-сумм массива (Prefix-Sums);
* Алгоритмы изменения порядка (Reordering);
* Алгоритмы сортировки (Sorting).

1. **Итераторы**.

В *библиотеке Thrust* есть итераторы для разных целей. Их использование со стандартными алгоритмами позволяет охватить многие классы задач. Можно выделить 5 видов итераторов:

*constant\_iterator* – данный итератор представляет собой указатель в диапазоне постоянных значений. Он удобен для создания промежутка, заполненного одним и тем же значением, при этом не нужно явно создавать его в памяти. Такой итератор позволяет экономить объем памяти и пропускной способности;

*counting\_iterator* – данный итератор похож на *constant\_iterator*. Он отличается от *constant\_iterator* тем, что представляет собой указатель на участок последовательно меняющихся значений. По умолчанию используется шаг, равный 1;

*transform\_iterator* – данный итератор используется для группового изменения элементов векторов. При этом сами элементы не изменятся;

*permutation\_iterator* – данный итератор служит для перестановки элементов;

*zip\_iterator* – данный итератор представляет собой указатель в диапазоне кортежей. Удобен для создания массивов структур. Также можно использовать для групповых операций.