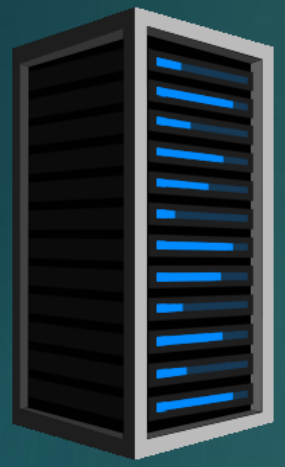
Разработка системы балансировки вычислительной нагрузки между CPU и GPU

ЗИМНЯЯ ШКОЛА ПО ПАРАЛЛЕЛЬНОМУ ПРОГРАММИРОВАНИЮ

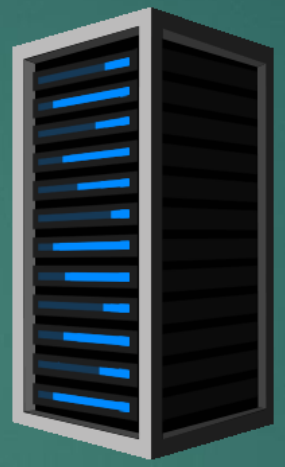
30.01.2017 - 03.02.2017

ПРОВОТОРОВ НИКИТА, ФИТ НГУ, 3 КУРС

Введение

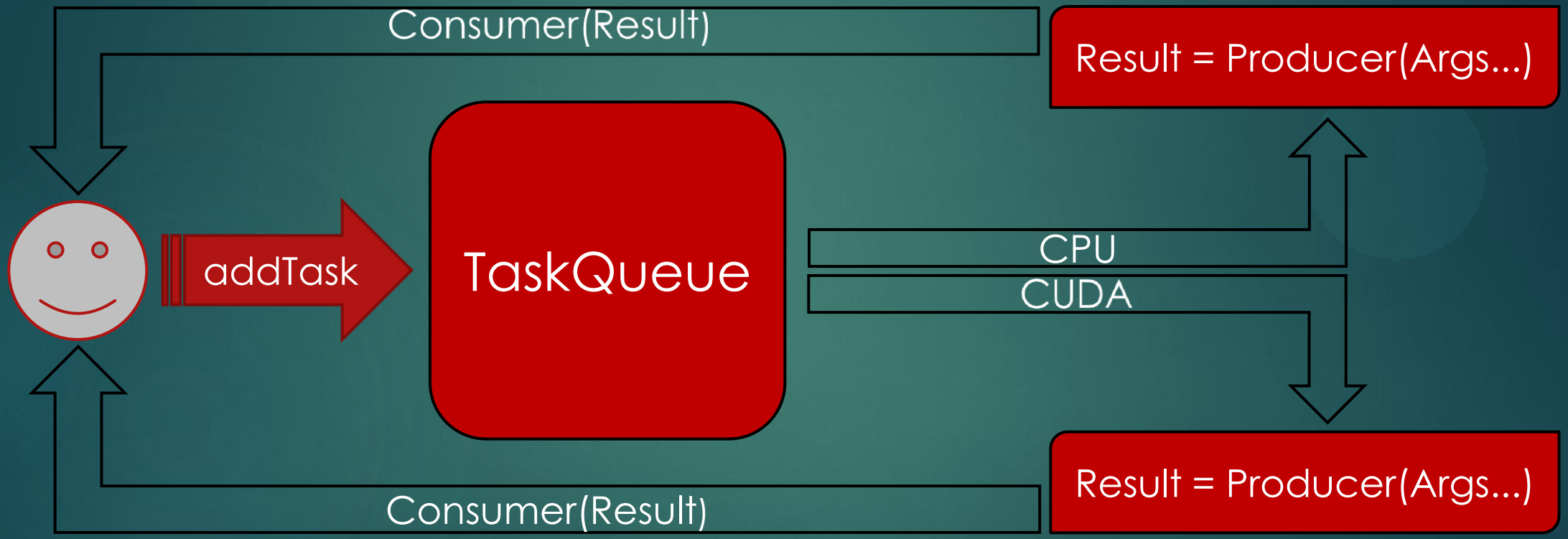


НАГРУЗКА



CPUs <-> GPUs <-> Coprocessors

Общая схема работы



Типы данных

- SingleValue<T> - любой объект
  - T не может быть ссылкой / указателем
- Reference<T> - ссылка на объект
- ConstReference<T> - ссылка на неизменяемый объект
- Range<T> - диапазон объектов
  - { T begin, T end }
  - T обязан удовлетворять концепциям C++ Iterator, EqualityComparable
- ConstRange<T> - диапазон неизменяемых объектов

Типы данных

- Cpu::Task
  - Producer, Consumer, Args...
  - Producer, Consumer обязаны удовлетворять концепции C++ Callable
  - Args... обязаны быть одним из :
    - SingleValue<T>
    - Reference<T>, ConstReference<T>
    - Range<T>, ConstRange<T>
- Cuda::Task
  - Producer - строго R (*) (Args...)

Типы данных

- Производитель - потребитель
  1. Result = Producer(Args...)
  2. Consumer(Result)

Типы данных

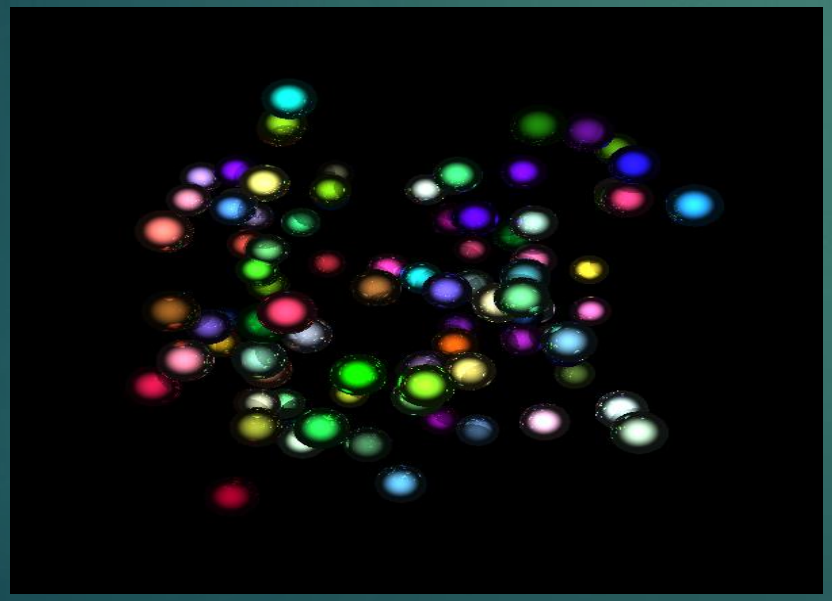
- TaskQueue - очередь задач
  - addTask(Cpu::Task)
  - addTask(Cuda::Task)

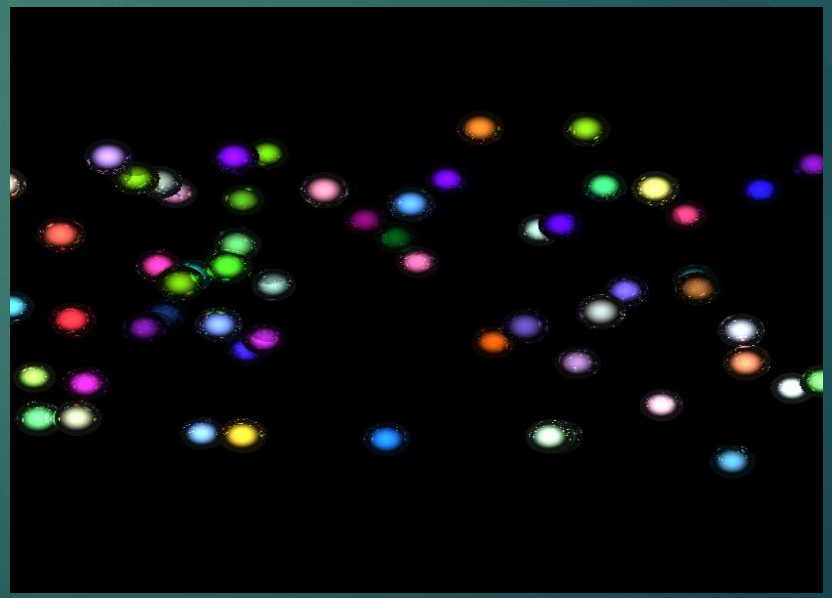
CPU Queue
Task 1
Task 2
…
Task N

CUDA Queue
Task 1
Task 2
…
Task M

Результаты

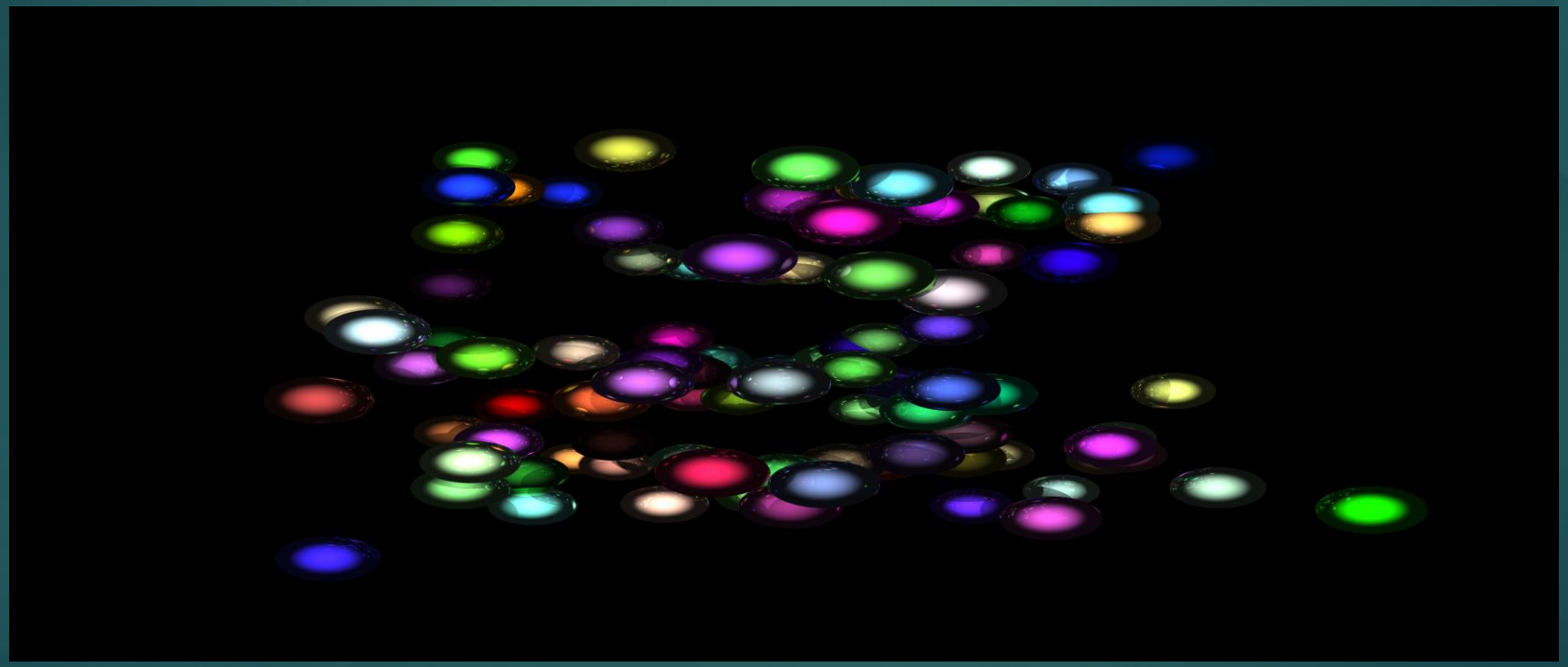
- Raytracing (CPU only, Intel Core i5 – 6200U, VC++ 2015)
  - 28.27с (последовательный код)
  - 13.14с (параллельный код)





Результаты

- Общий тест : 3.74с.



Следующие шаги

- Реализация Cuda::Task
- Введение зависимостей
- Реорганизация TaskQueue в ROB

Спасибо за внимание!