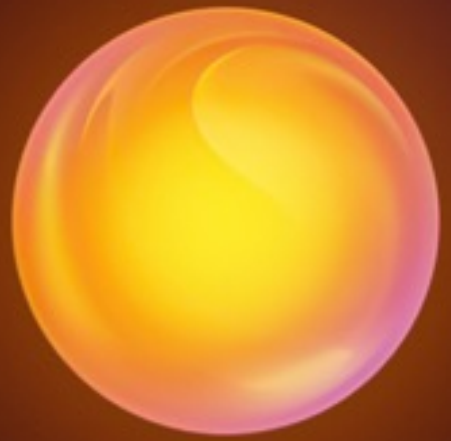
Ускоряем синтез:

от TensorRt до CUDA C++



Хорошев Артем, askhoroshev@sberbank.ru

О чем доклад

Технология TextToSpeech от SberDevices 

О чем доклад

Технология TextToSpeech от SberDevices

Зачем использовать GPU и какой выигрыш это даёт

О чем доклад

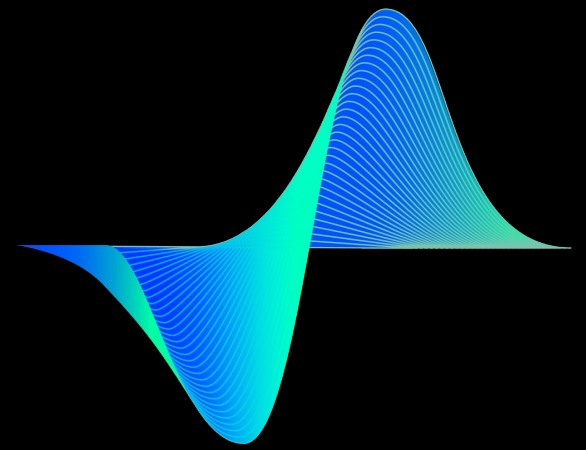
Технология TextToSpeech от SberDevices

Зачем использовать GPU и какой выигрыш это даёт

Как использовать GPU. От использования высокоуровневых фреймворков до CUDA C++

SaluteSpeech

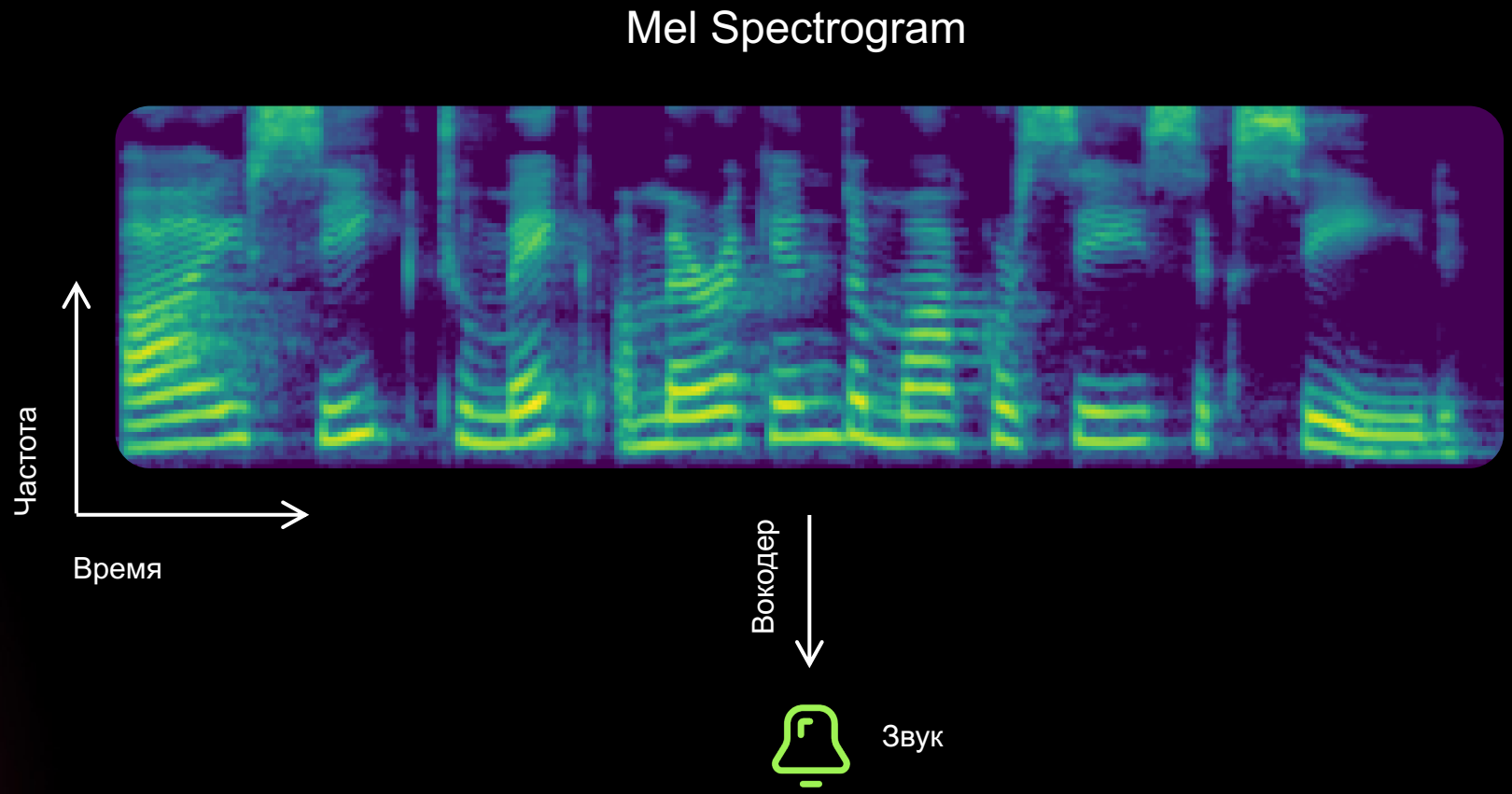
- Сервис для синтеза человеческой речи из текста
- Потоковое и асинхронное API
- Работает как на CPU так и на GPU
- Более 50 голосов



SBER DEVICES

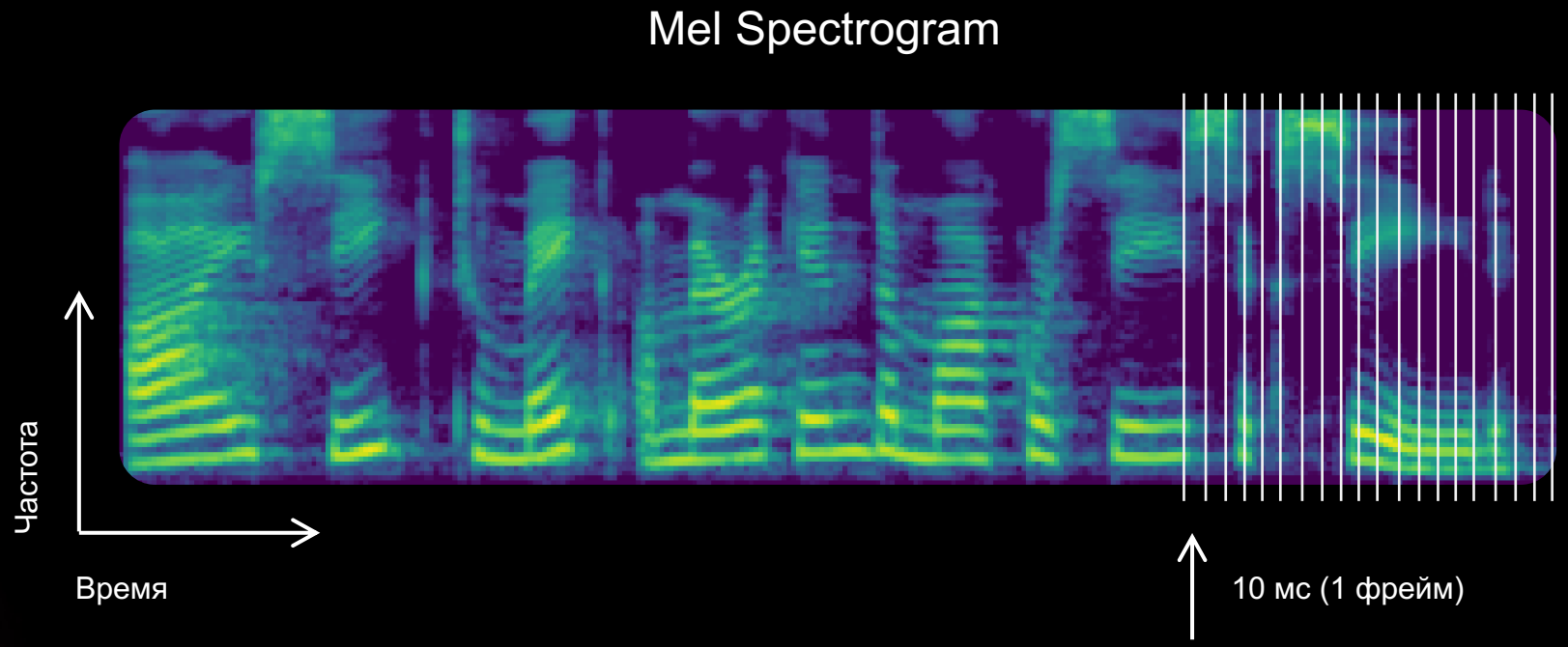
Генерация звука

- Генерируем спектрограмму
- Из спектрограммы генерируем звук



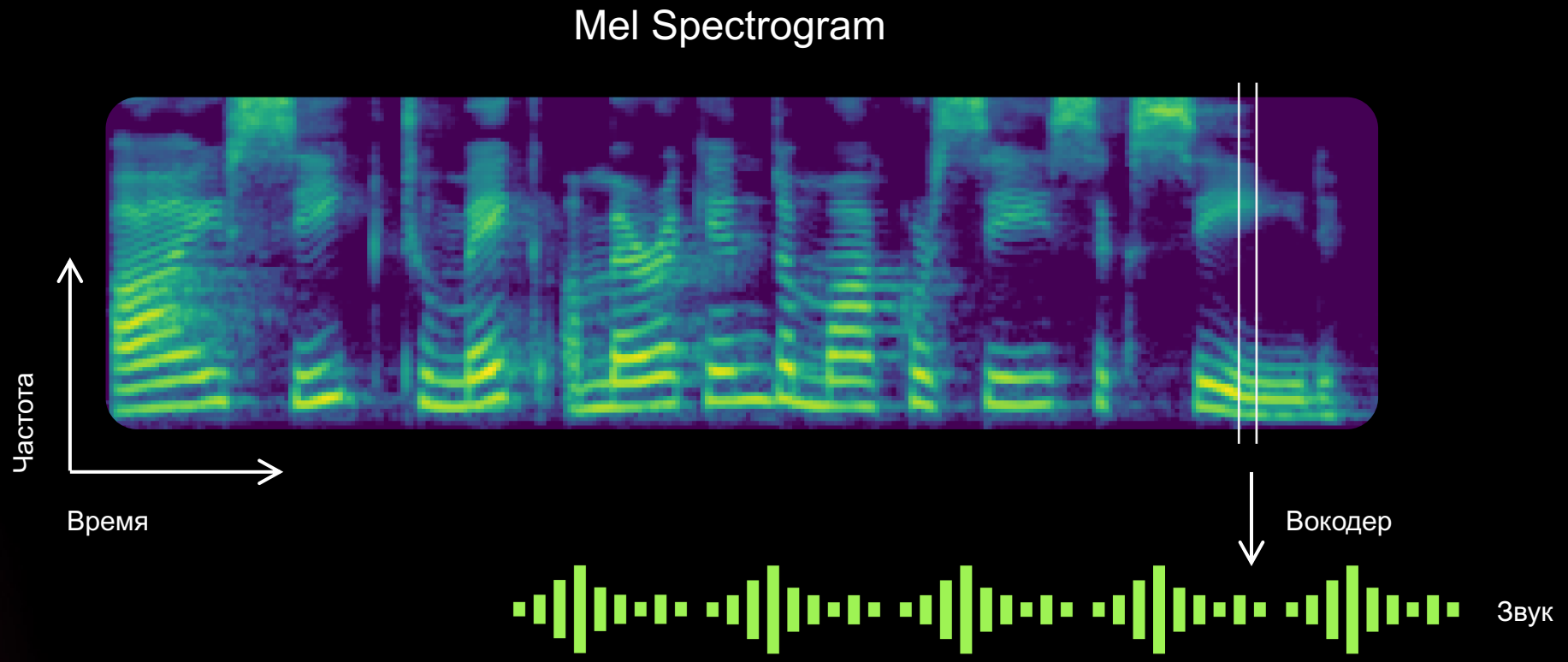
Генерация звука

Спектрограмма генерируется в потоковом режиме, фреймами по 10 мс



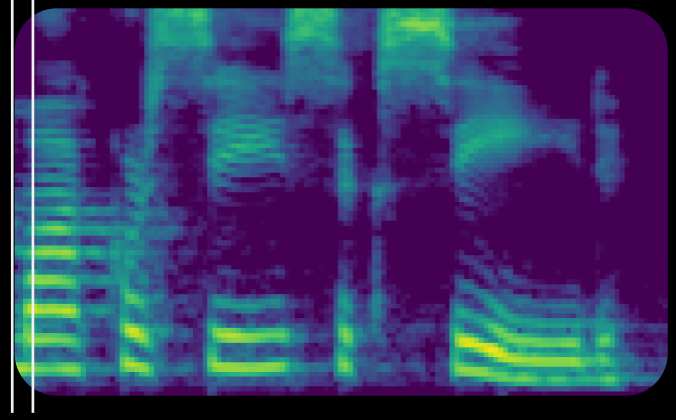
Генерация звука

Результирующий звук генерируется вокодером чанками по 10 мс

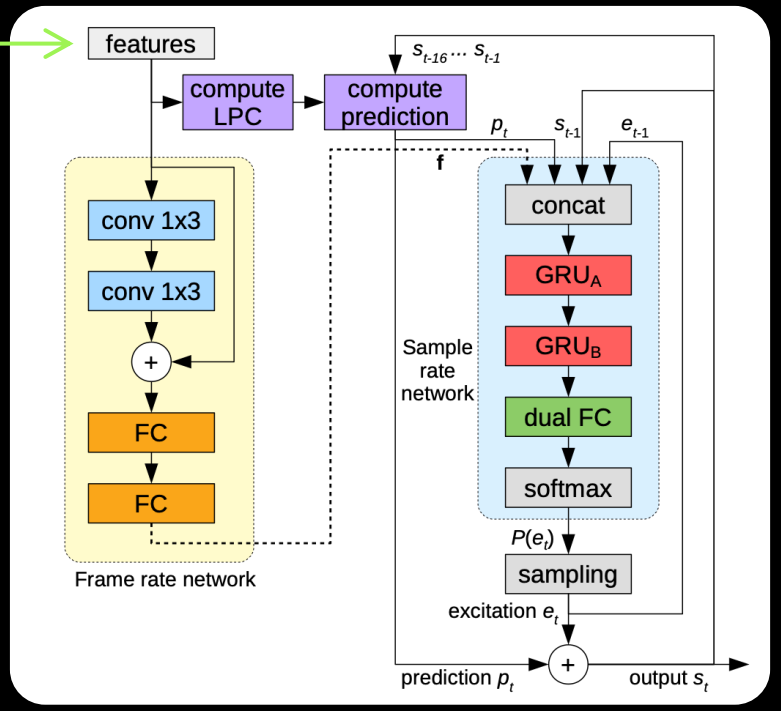


Вокодер

Мы используем вокодер на основе алгоритма LPCNet [1]



1 фрейм



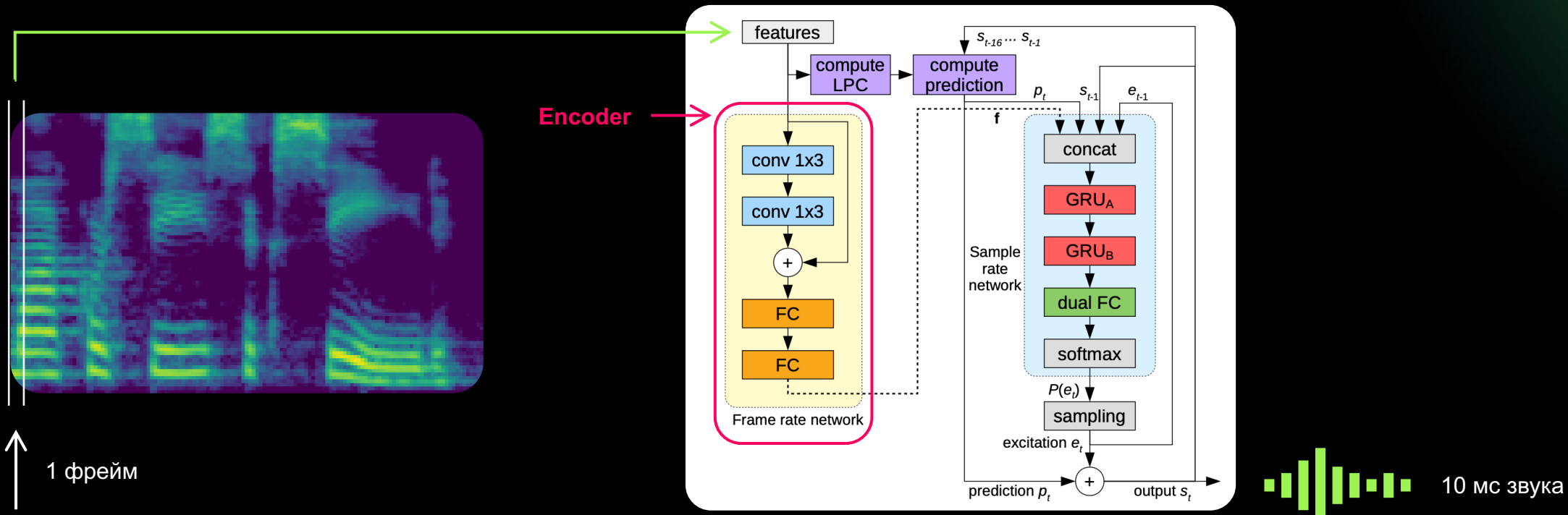
10 мс звука

[1] LPCNet: Improving Neural Speech Synthesis Through Linear Prediction https://arxiv.org/abs/1810.1184

SBER DEVICES

Вокодер

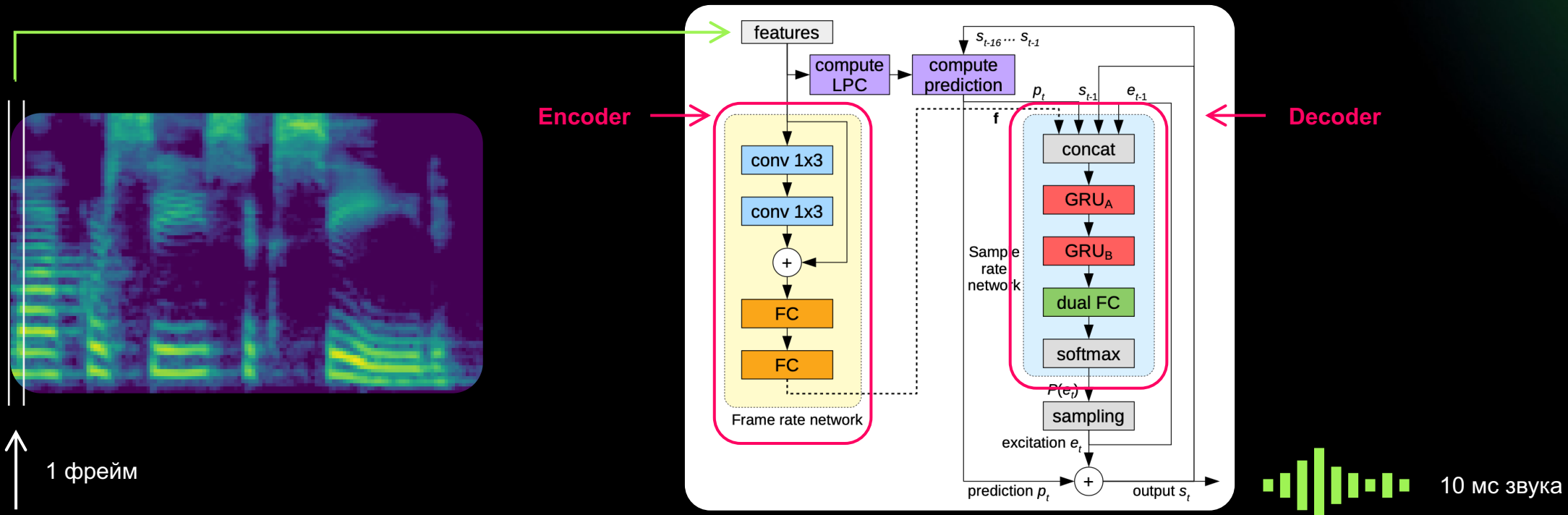
Мы используем вокодер на основе алгоритма LPCNet [1]



[1] LPCNet: Improving Neural Speech Synthesis Through Linear Prediction https://arxiv.org/abs/1810.1184

Вокодер

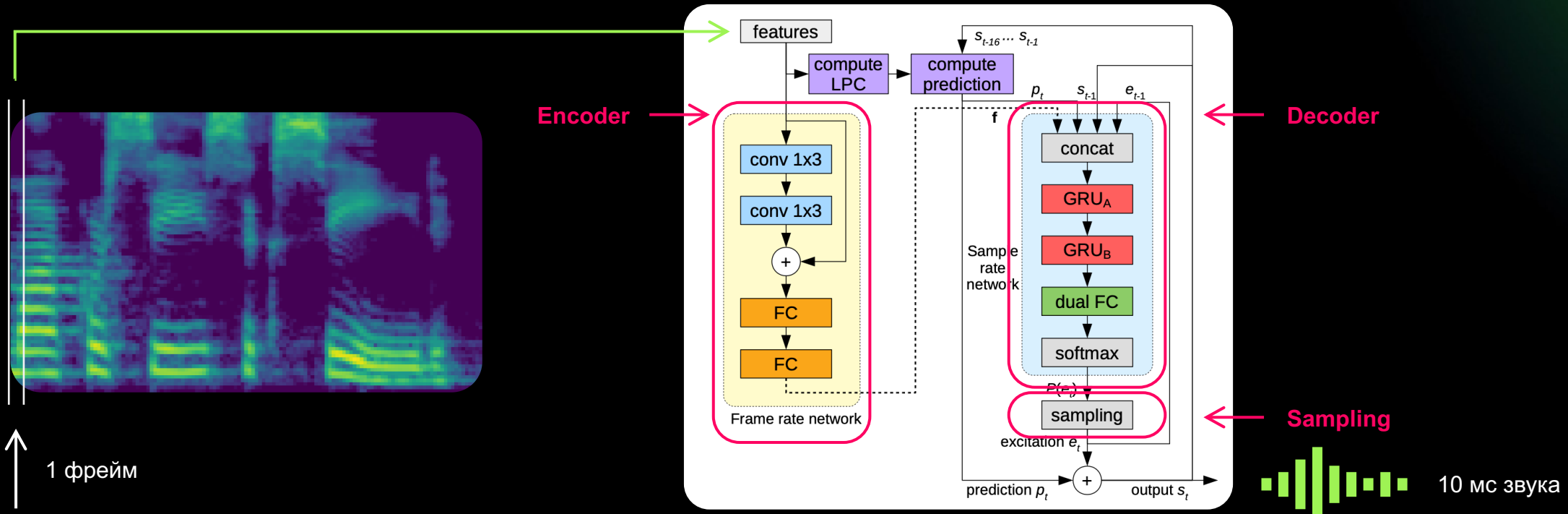
Мы используем вокодер на основе алгоритма LPCNet [1]



[1] LPCNet: Improving Neural Speech Synthesis Through Linear Prediction https://arxiv.org/abs/1810.1184

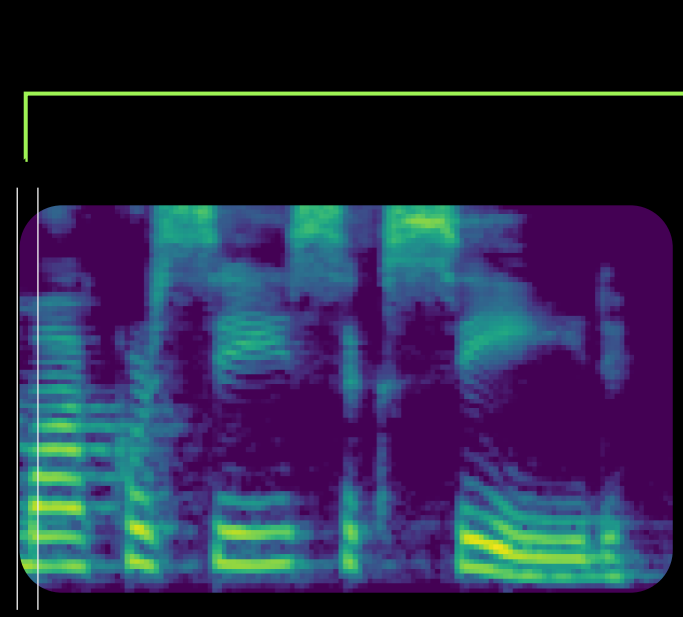
Вокодер

Мы используем вокодер на основе алгоритма LPCNet [1]



[1] LPCNet: Improving Neural Speech Synthesis Through Linear Prediction https://arxiv.org/abs/1810.1184

Вокодер



1 фрейм

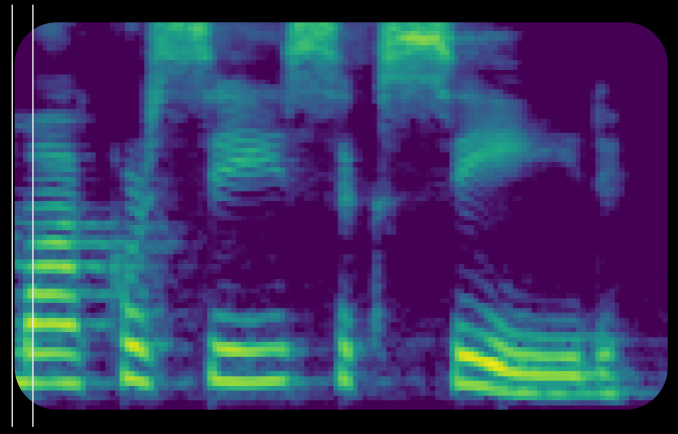
Encoder

Decoder

Sampling

10 мс звука

Вокодер



1 фрейм

Encoder

При частоте дискретизации 24 KHz цикл decoder-sampling выполняется 240 раз

Decoder

Sampling

10 мс звука

Метрики производительности

Хотим обслуживать наибольшее число клиентов при наименьшем RTF

RTF должен быть меньше 1, иначе синтезируемый звук будет «заикаться»

RTF (real time factor) – если синтезируем 10 секунд звука за 1 секунду, то RTF 0.1

Производительность на CPU

Написан на C

Хорошо оптимизирован, использует векторные инструкции под капотом

Вокодер на Xeon Gold 5218 x2 (32c/64t) выдерживает до 64 пользователей

Зачем оптимизировать вокодер под GPU

GPU позволяет использовать более качественные модели

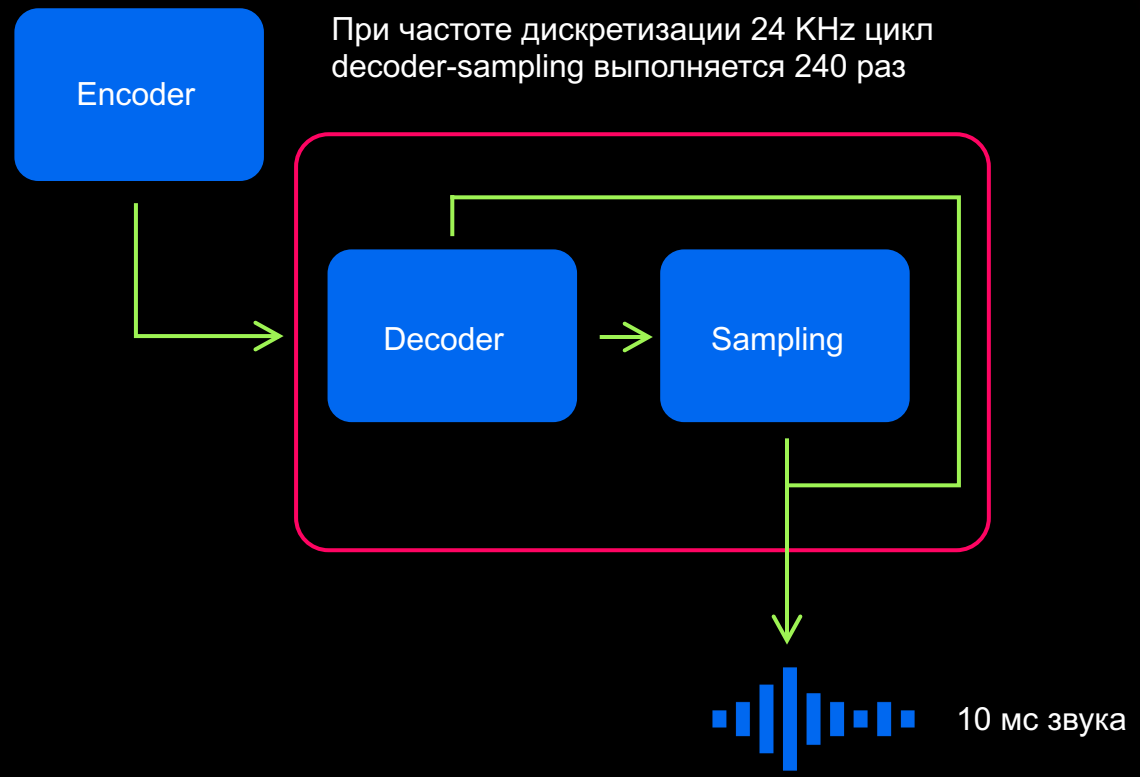
Стоимость обслуживания в пересчете на 1 клиента у GPU ниже

Вокодер стал узким местом после того, как другие компоненты были перенесены на GPU

SBER DEVICES

Что хотим получить

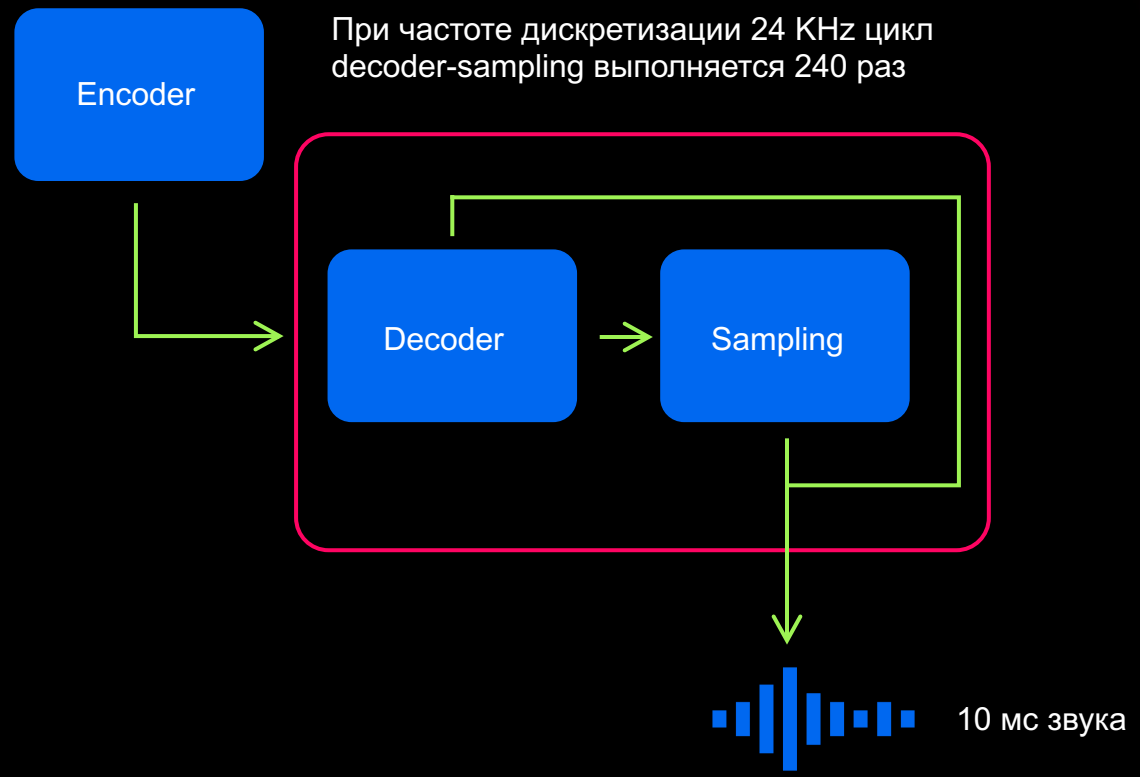
Как можно больше потоков с RTF < 1



Что хотим получить

Как можно больше потоков с RTF < 1

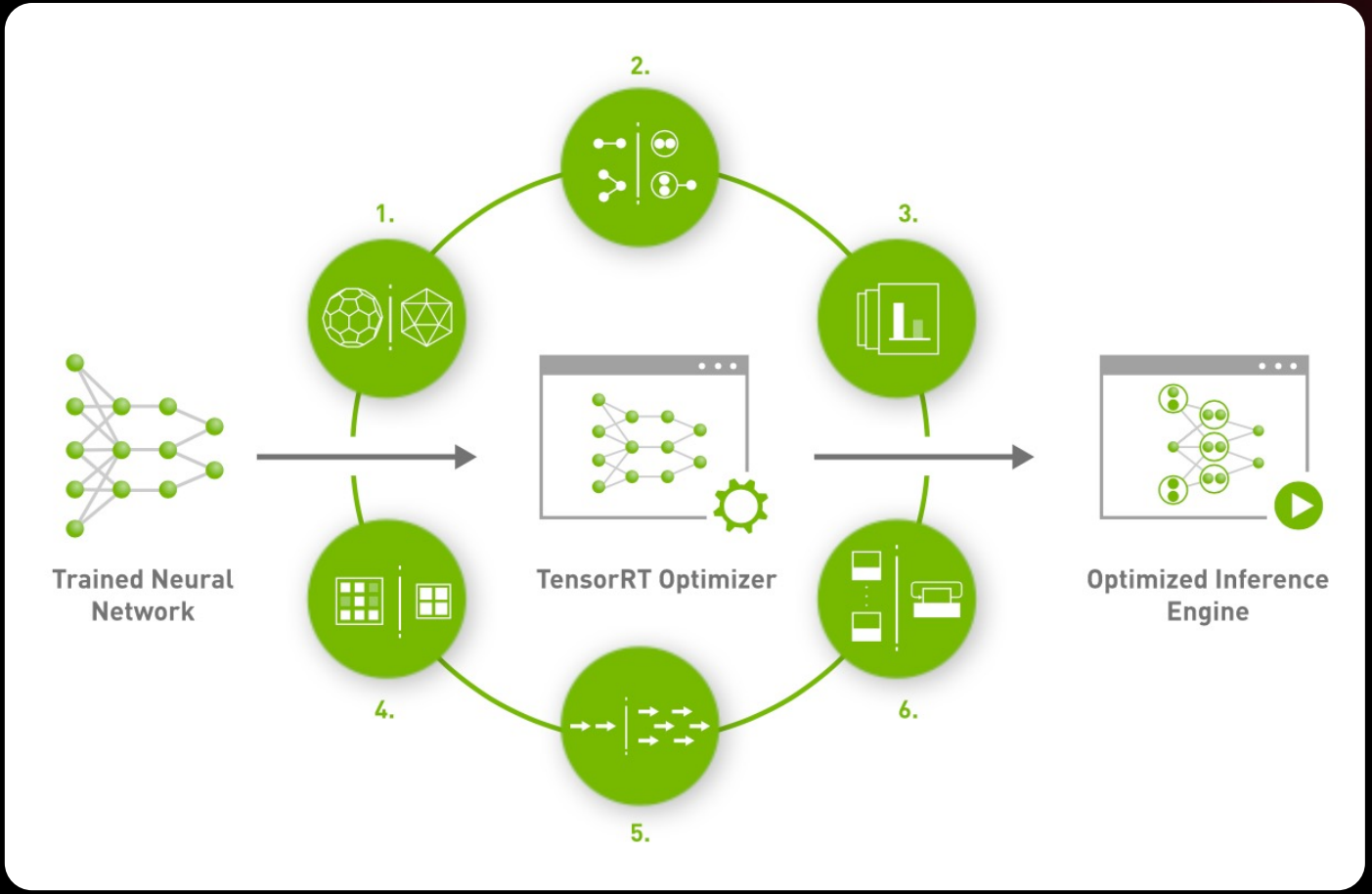
В среднем один цикл decoder-sampling должен умещаться в 10 мс / 240 ~ 42 мкс



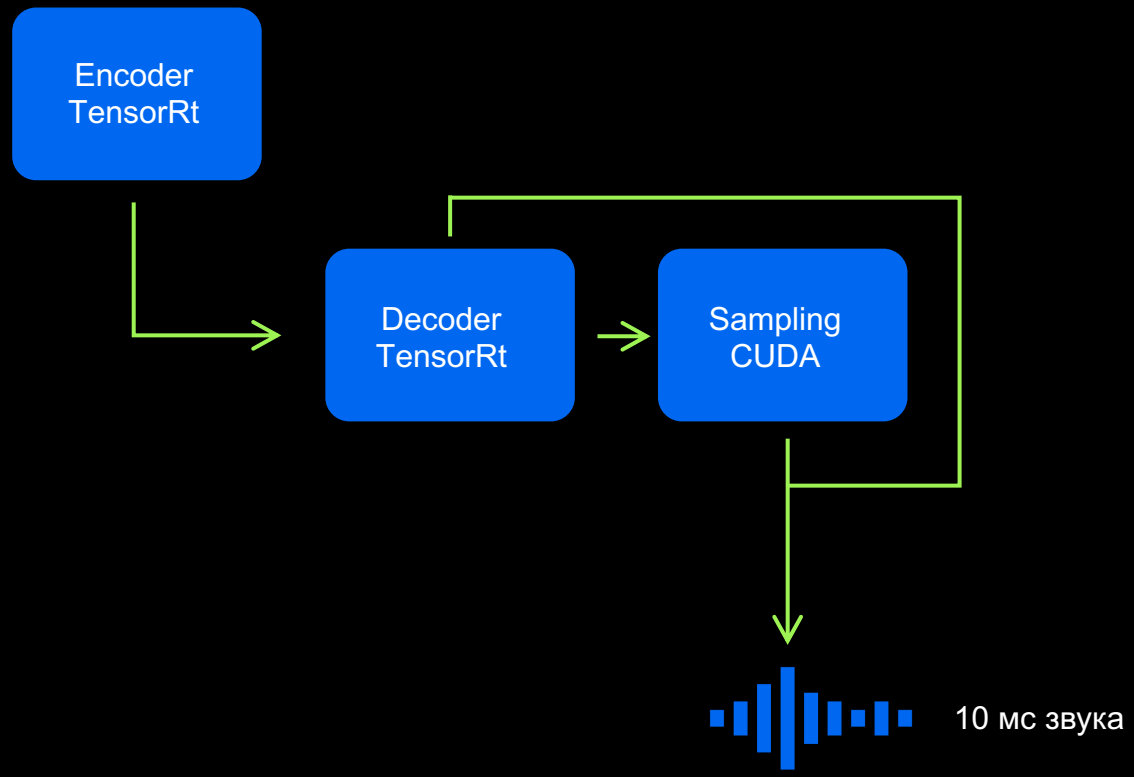
TensorRt

Фреймворк от NVIDIA для оптимизации инференса нейросетей

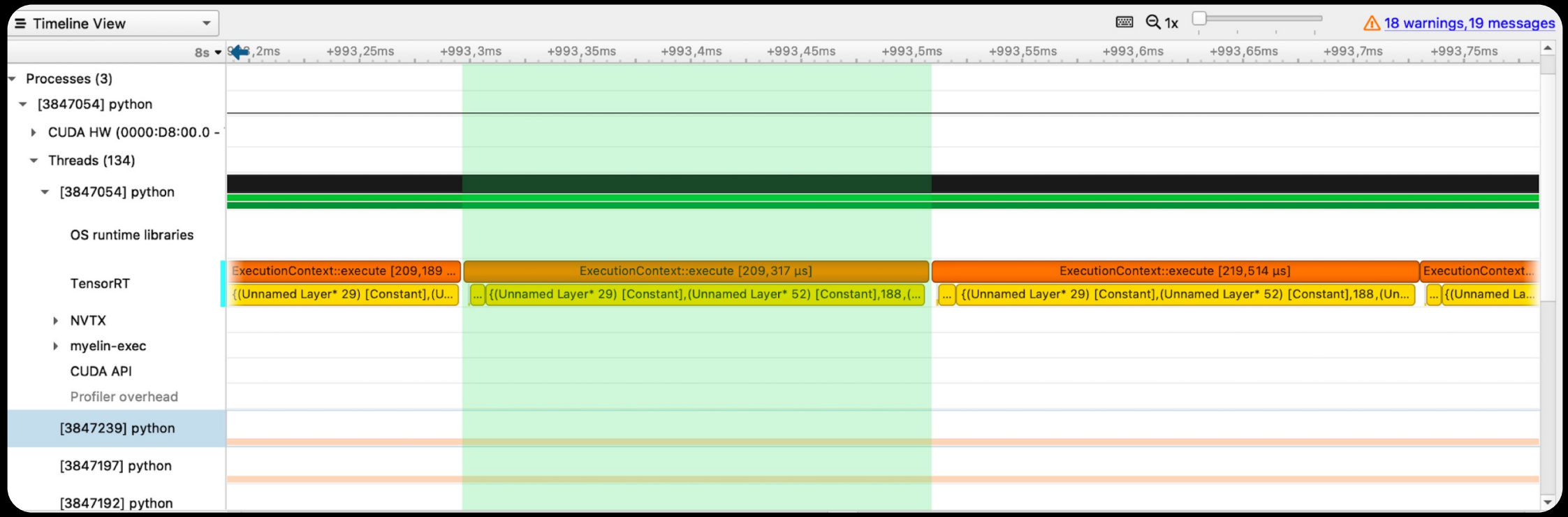
Достаточно прост, работает по принципу черного ящика (почти всегда)



TensorRt



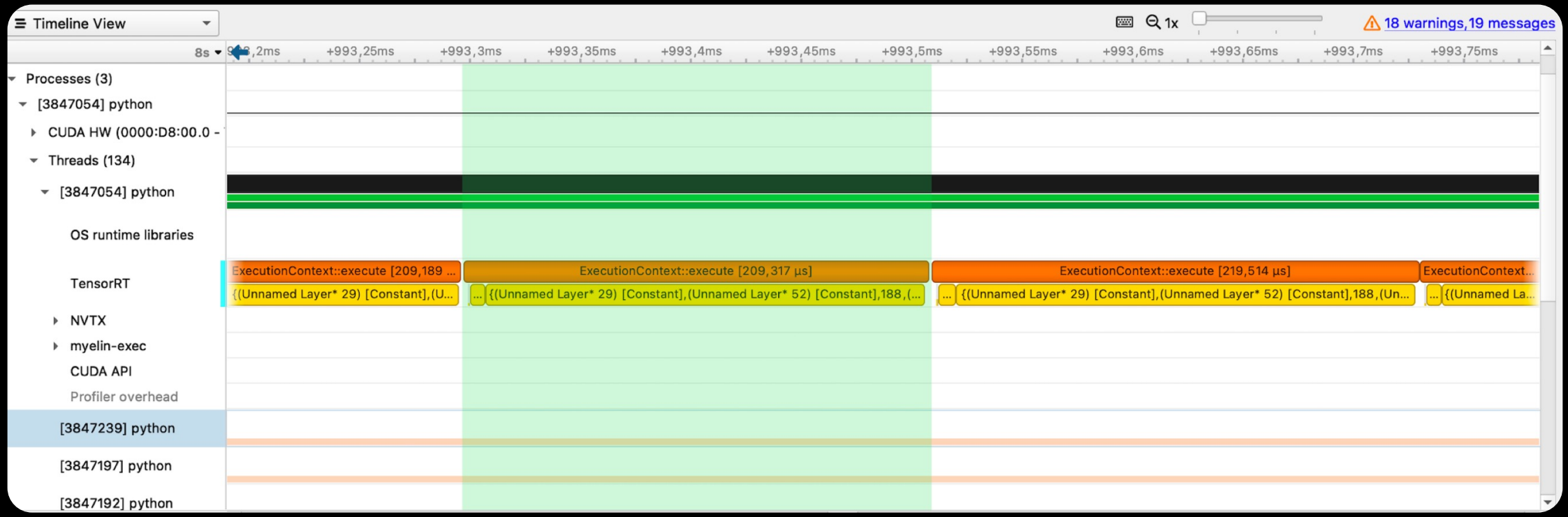
Время работы decoder используя TensorRt



TensorRt сделал хороший fuse слоев*

* fuse слоев – оптимизация, объединяющая несколько операций в один слой

Время работы decoder используя TensorRt

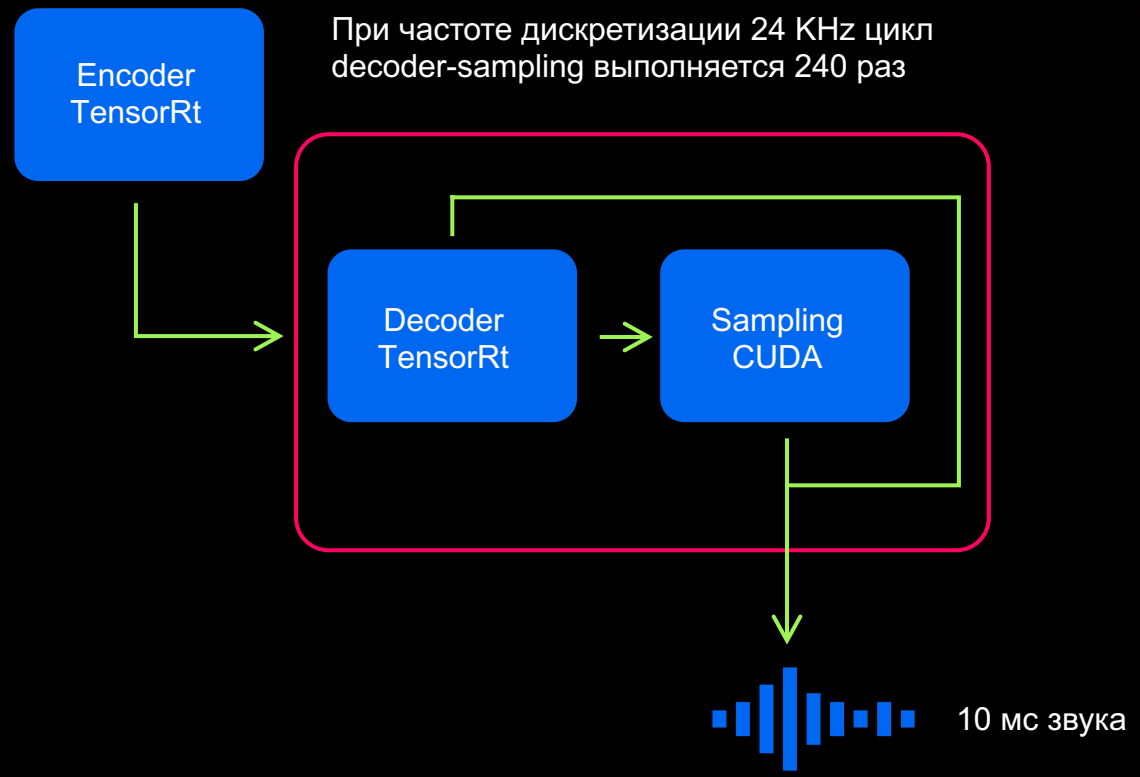


210 мкс на decoder в профилировщике
120 мкс без профилировщика

* fuse слоев – оптимизация, объединяющая несколько операций в один слой

Время работы decoder используя TensorRt

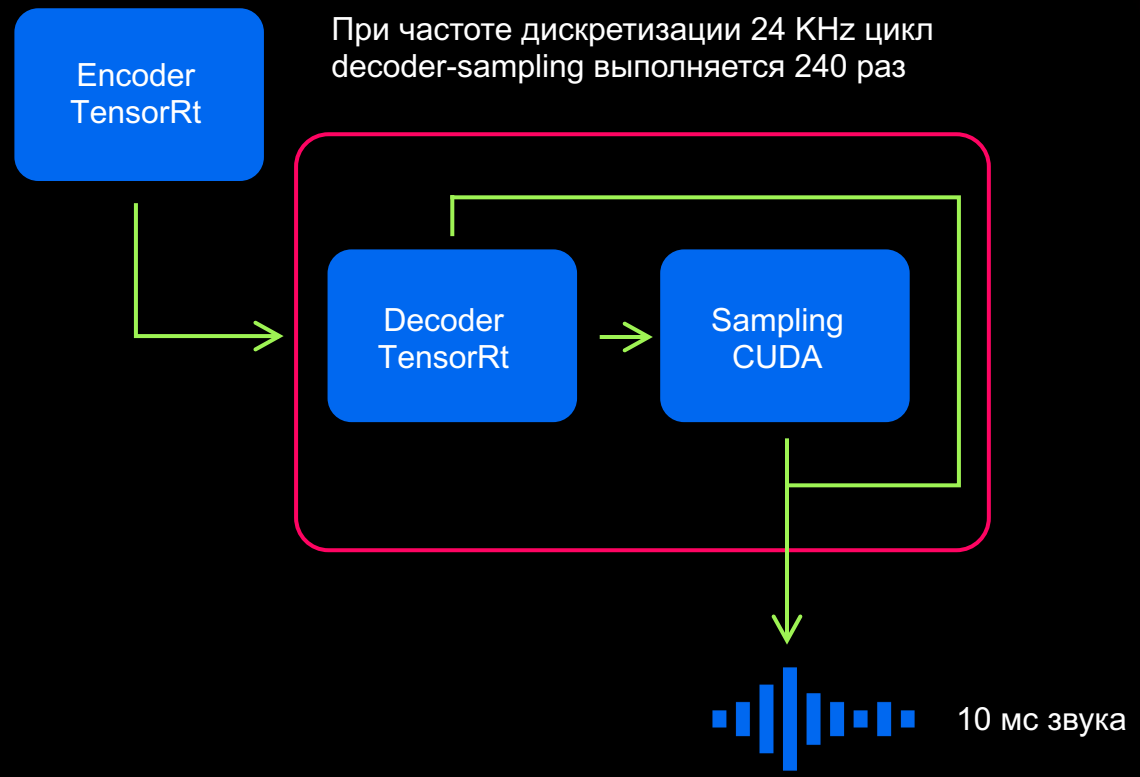
В среднем один цикл decoder-sampling должен умещаться в 10 мс / 240 ~ 42 мкс



Время работы decoder используя TensorRt

В среднем один цикл decoder-sampling должен умещаться в 10 мс / 240 ~ 42 мкс

С decoder TensorRt один цикл занимает минимум 120 мкс, т.е. RTF > 3

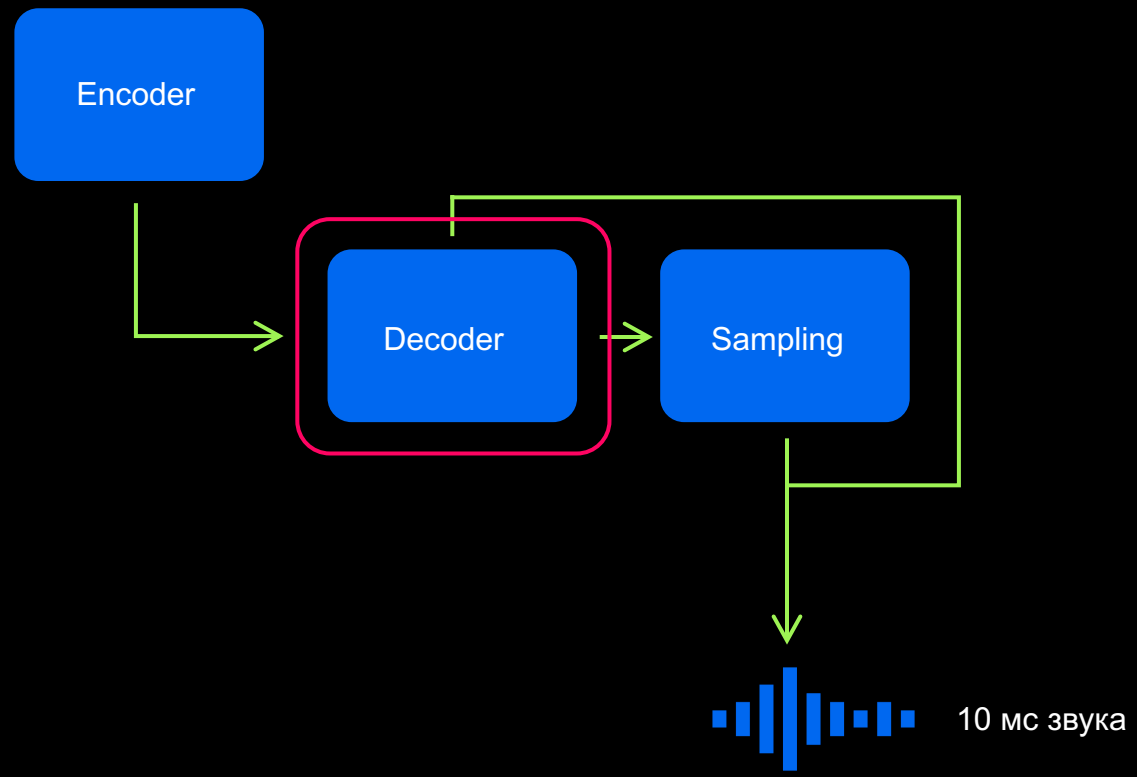


Decoder детальнее

Decoder это нейросеть

На вход и выход тензоры

По простому – граф вычислений

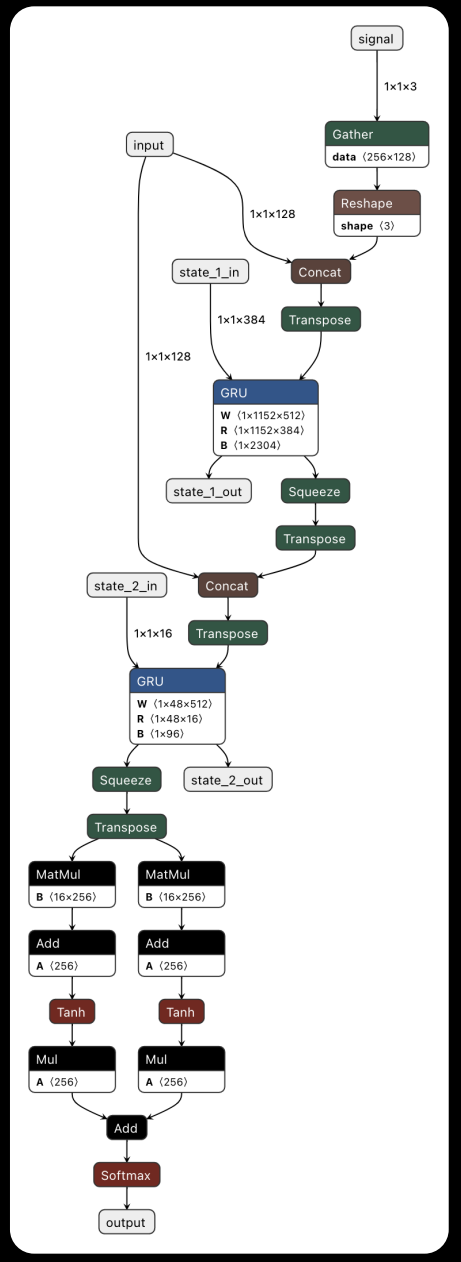


Decoder детальнее

Decoder это нейросеть

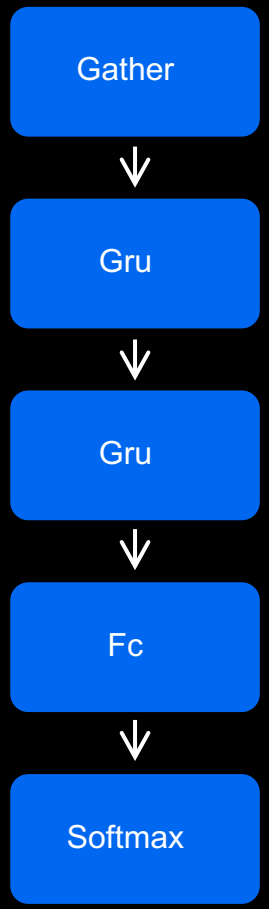
На вход и выход тензоры

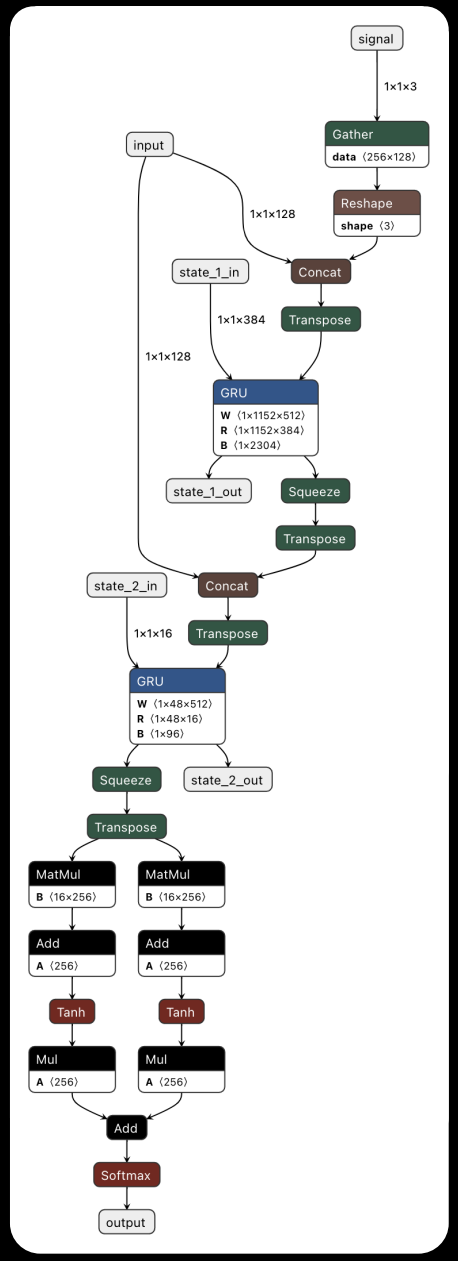
По простому – граф вычислений



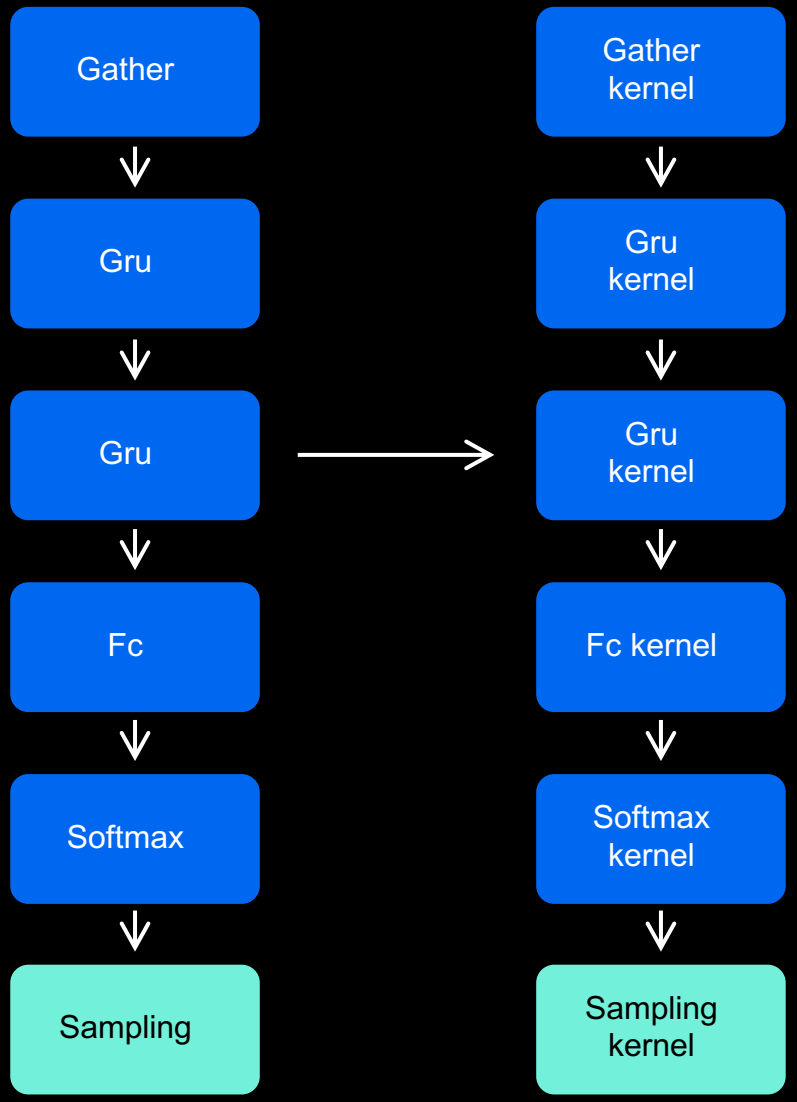
SBER DEVICES

Decoder детальнее

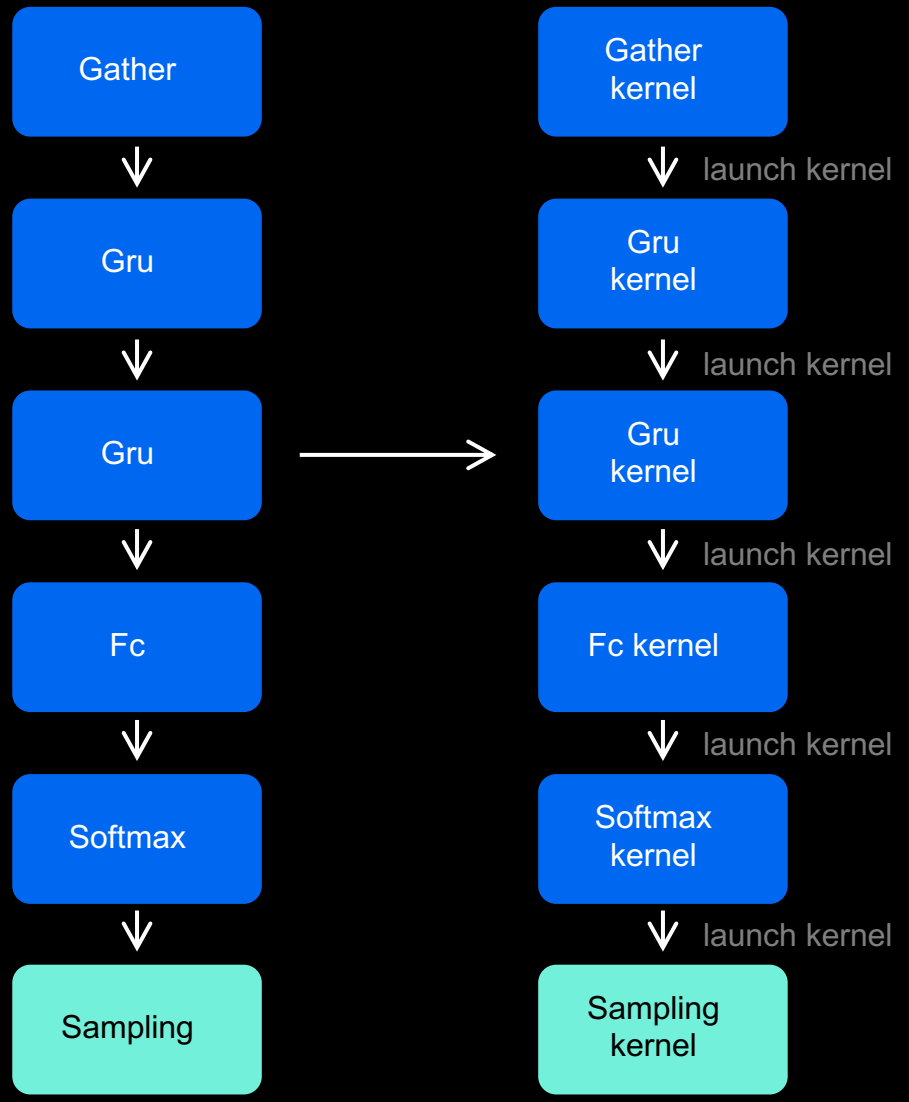




Decoder-sampling через Cublas и CUDA



Decoder-sampling через Cublas и CUDA

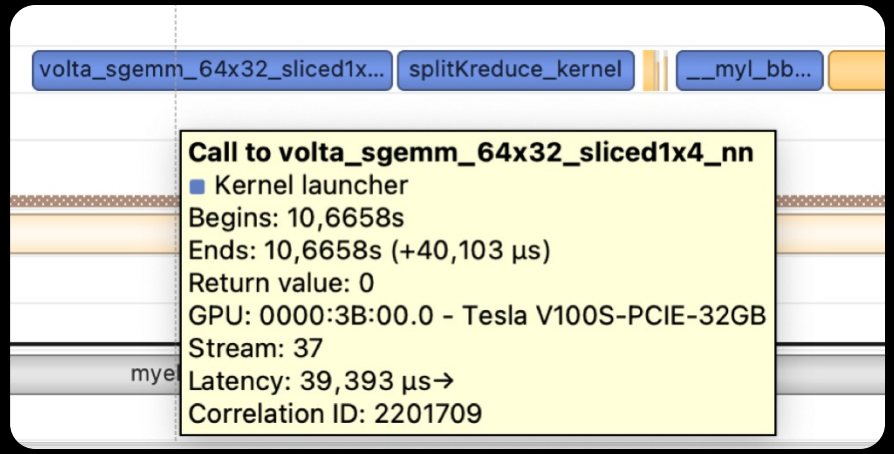


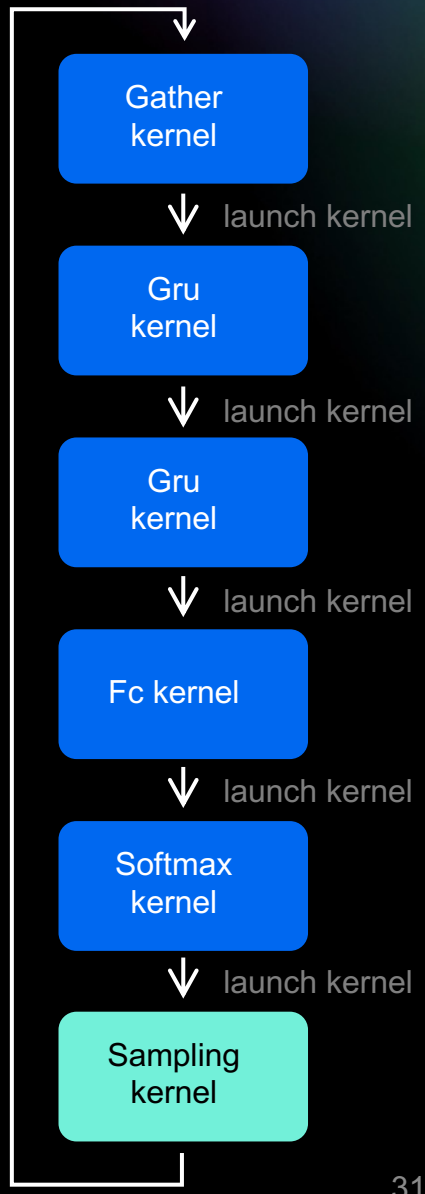
Время работы decoder используя Cublas и CUDA

В среднем один цикл decoder-sampling должен умещаться в 10 мс / 240 ~ 42 мкс

Операция launch kernel занимает 5-10 мкс без профилировщика

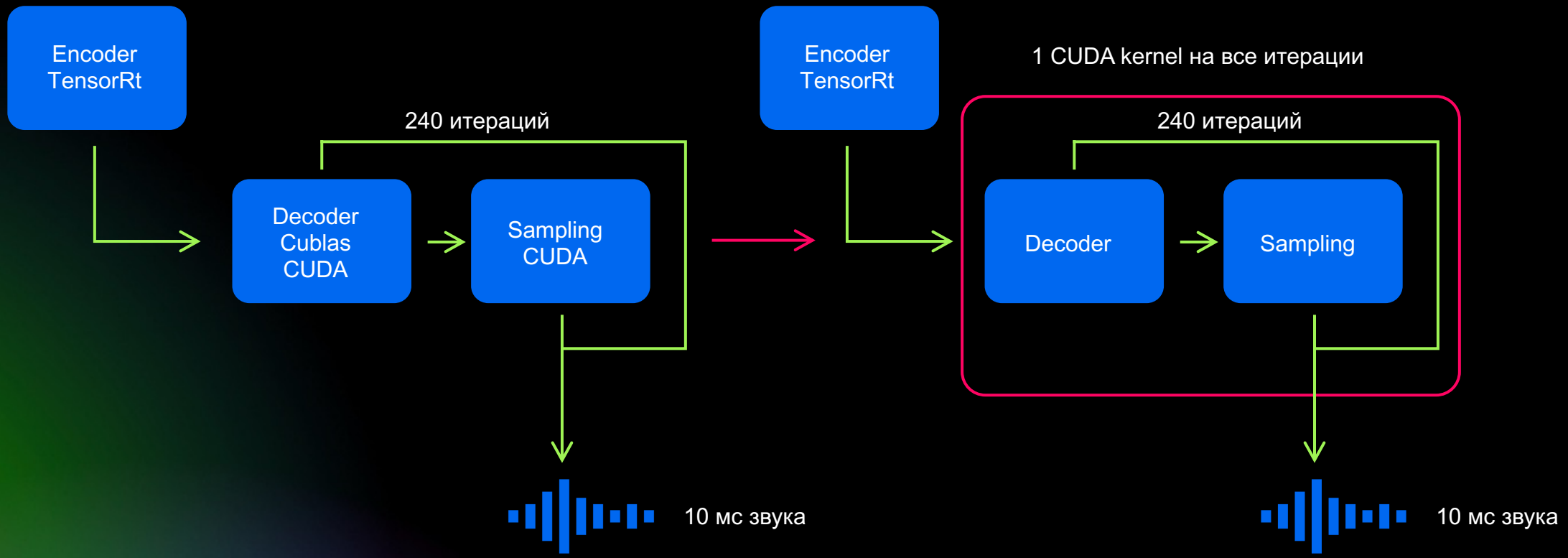
Оверхед на частый вызов небольших kernel’ов становится определяющим





SBER DEVICES

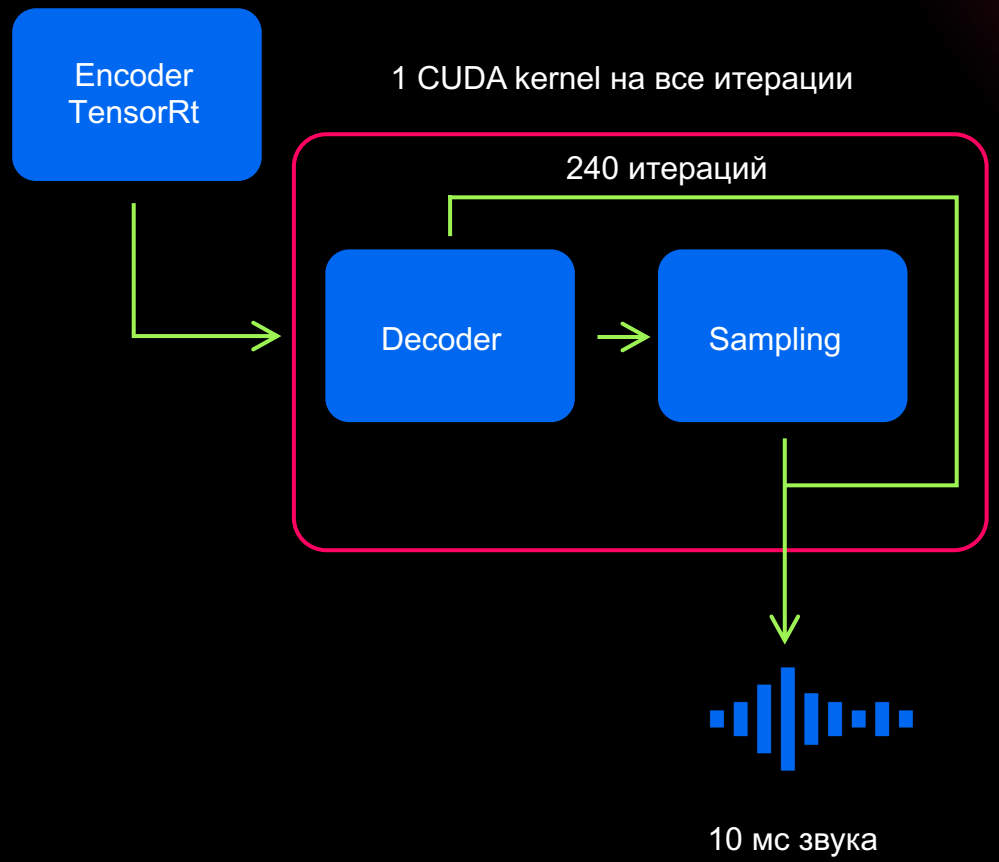
Decoder-sampling через один kernel



Decoder-sampling через один kernel

Обойдемся одним запуском kernel’а, тем самым скроем latency которое тратилось на частый запуск

Получим идеальный fuse, так как теперь у нас один kernel

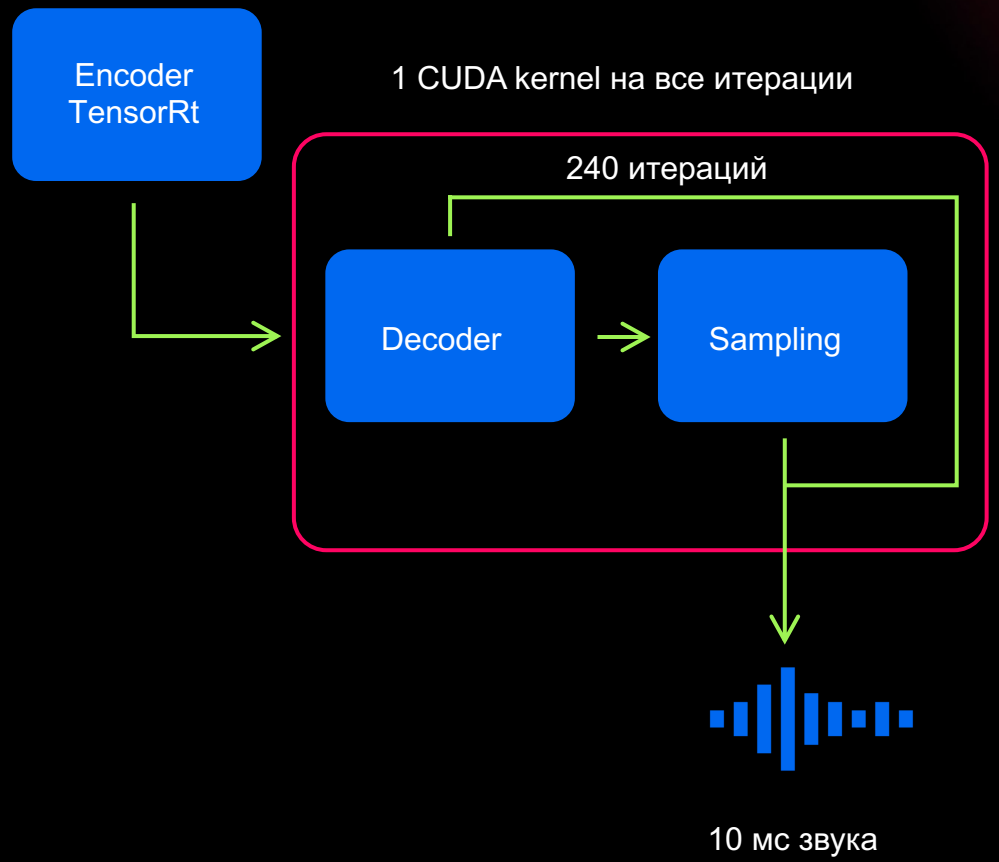


Decoder-sampling через один kernel

Обойдемся одним запуском kernel’а, тем самым скроем latency которое тратилось на частый запуск

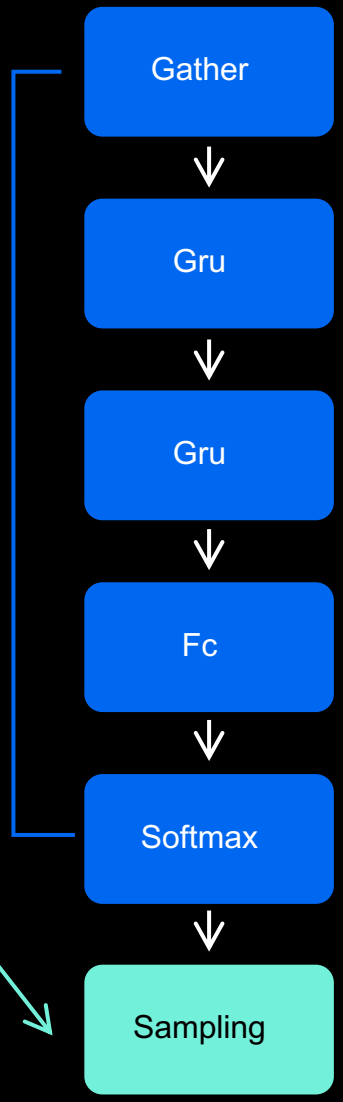
Получим идеальный fuse, так как теперь у нас один kernel

НО: внутри кастомных CUDA kernel’ов не выйдет использовать сторонние библиотеки по типу Cublas, все придется писать самим



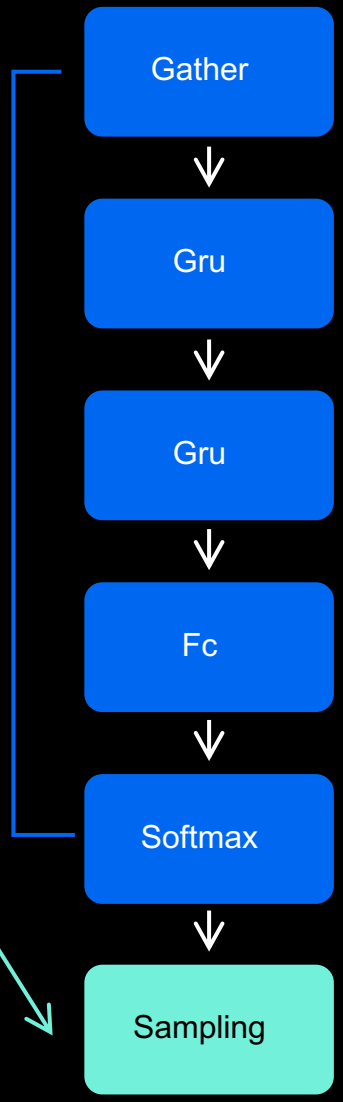
Реализация концептуально

```
1       __device__ void ComputeDecoder(State);
2       __device__ void ComputeSampling(State);
3
4       __global__ void Loop(State state) {
5
6         for (auto i = 0; i < 240; i++) {
7           ComputeDecoder(state);
8           ComputeSampling(state);
9         }
10
11      }
```



Батчинг

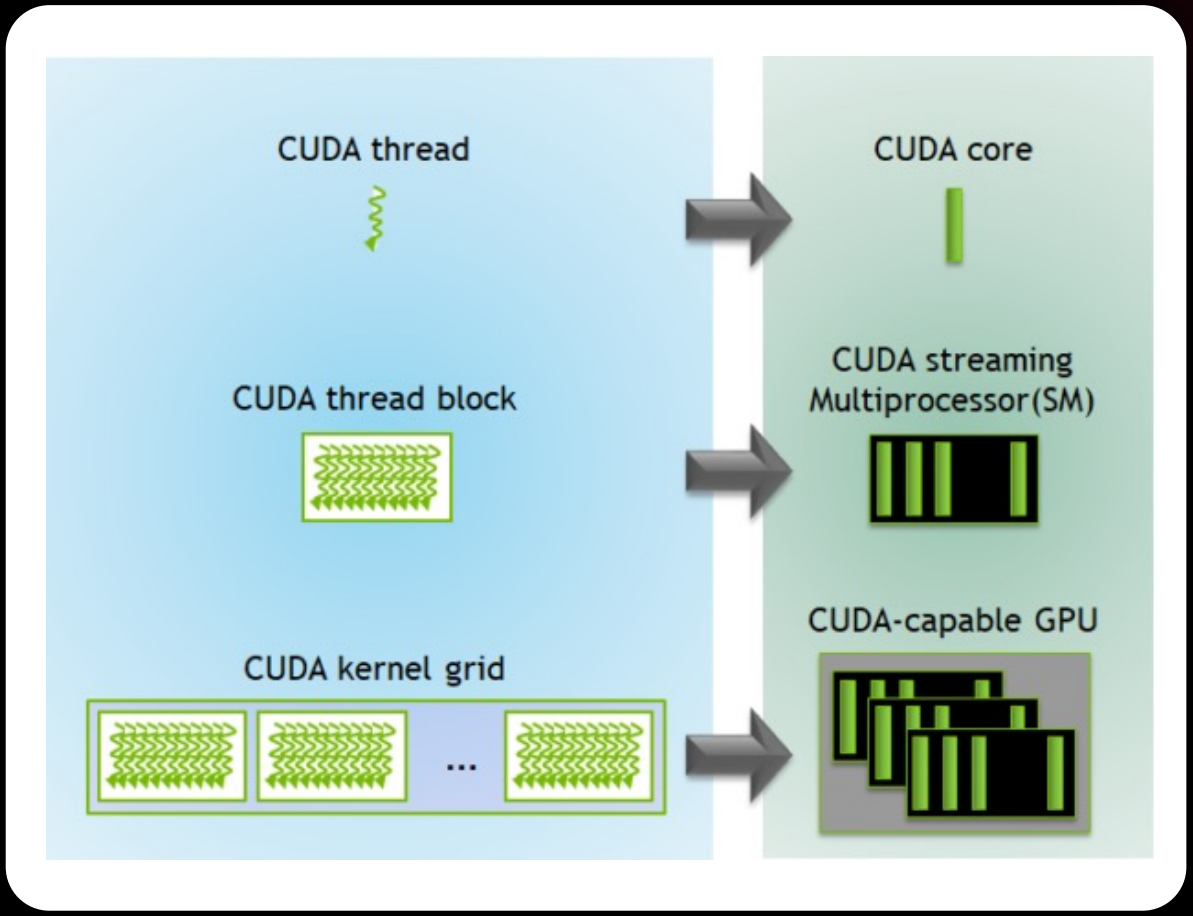
```
1       __device__ void ComputeDecoder(BatchedState);
2       __device__ void ComputeSampling(BatchedState);
3
4       __global__ void Loop(BatchedState state) {
5
6         for (auto i = 0; i < 240; i++) {
7           ComputeDecoderBatch(state);
8           ComputeSamplingBatch(state);
9         }
10
11      }
```



Конфигурация сетки

Существует концепция потоков и блоков

Выбор количества потоков внутри одного блока а так же количества блоков определяет размер всей сетки

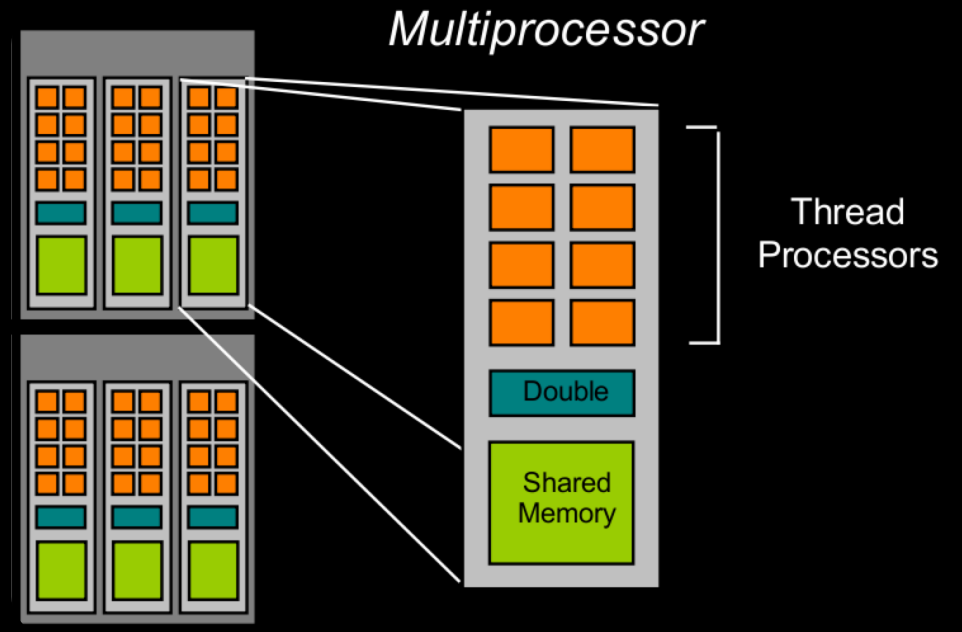


Decoder-sampling через один kernel

Каждый отдельный мультипроцессор (блок потоков) будет обрабатывать несколько элементов батча

Необходимые данные для обработки одного запроса могут быть кешированы в shared memory

Нет необходимости синхронизации всей сетки, достаточно лишь внутри мультипроцессора (блока)



Структура для хранения весов

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

Структура для хранения весов

```
1   template <typename T, typename VoiceWeightsConfig>
2   struct VoiceWeights {
3       static_assert(!std::is_const_v<T>);
4
5       using Config = VoiceWeightsConfig;
6       using Type = T;
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

Структура для хранения весов

```
1   template <typename T, typename VoiceWeightsConfig>
2   struct VoiceWeights {
3       static_assert(!std::is_const_v<T>);
4
5       using Config = VoiceWeightsConfig;
6       using Type = T;
7
8       weights::PrecomputedEmbed<Type, typename VoiceWeightsConfig::PrecomputedEmbed> precomputed_embed;
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

Структура для хранения весов

```
template <typename T, typename VoiceWeightsConfig>
struct VoiceWeights {
        static_assert(!std::is_const_v<T>);

        using Config = VoiceWeightsConfig;
        using Type = T;

        weights::PrecomputedEmbed<Type, typename VoiceWeightsConfig::PrecomputedEmbed> precomputed_embed;
        weights::Gru<Type, typename VoiceWeightsConfig::GruA> gru_a;
        weights::Gru<Type, typename VoiceWeightsConfig::GruB> gru_b;
```

Структура для хранения весов

```
1   template <typename T, typename VoiceWeightsConfig>
2   struct VoiceWeights {
3       static_assert(!std::is_const_v<T>);
4
5       using Config = VoiceWeightsConfig;
6       using Type = T;
7
8       weights::PrecomputedEmbed<Type, typename VoiceWeightsConfig::PrecomputedEmbed> precomputed_embed;
9       weights::Gru<Type, typename VoiceWeightsConfig::GruA> gru_a;
10      weights::Gru<Type, typename VoiceWeightsConfig::GruB> gru_b;
11      weights::FullConnected<Type, typename VoiceWeightsConfig::FullConnected> fc_fst;
12      weights::FullConnected<Type, typename VoiceWeightsConfig::FullConnected> fc_snd;
13  };
14
15
16
17
18
19
20
21
22
23
24
25
26
```

Структура для хранения весов

```
1   template <typename T, typename VoiceWeightsConfig>
2   struct VoiceWeights {
3           static_assert(!std::is_const_v<T>);
4
5           using Config = VoiceWeightsConfig;
6           using Type = T;
7
8           weights::PrecomputedEmbed<Type, typename VoiceWeightsConfig::PrecomputedEmbed> precomputed_embed;
9           weights::Gru<Type, typename VoiceWeightsConfig::GruA> gru_a;
10          weights::Gru<Type, typename VoiceWeightsConfig::GruB> gru_b;
11          weights::FullConnected<Type, typename VoiceWeightsConfig::FullConnected> fc_fst;
12          weights::FullConnected<Type, typename VoiceWeightsConfig::FullConnected> fc_snd;
13  };
14
15  template <typename T, typename FullConnectedConfig>
16  struct FullConnected {
17          static_assert(!std::is_const_v<T>);
18
19          using Type = T;
20          static constexpr index_t kInputSize = FullConnectedConfig::kInputSize;
21          static constexpr index_t kOutputSize = FullConnectedConfig::kOutputSize;
22
23          container::Matrix<Type, kOutputSize, kInputSize> matmul;
24          container::Vector<Type, kOutputSize> bias;
25          container::Vector<Type, kOutputSize> scale;
26  };
```

Структура для хранения состояния нейросети

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

Структура для хранения состояния нейросети

```
template <typename T, index_t BatchSize, typename LPCNetConfig>
struct LPCNetBatched {
        static_assert(!std::is_const_v<T>);

        using Type = T;
        using StateConfig = typename LPCNetConfig::State;
        using DecoderConfig = typename LPCNetConfig::Decoder;

        static constexpr auto kBatchSize = BatchSize;
```

Структура для хранения состояния нейросети

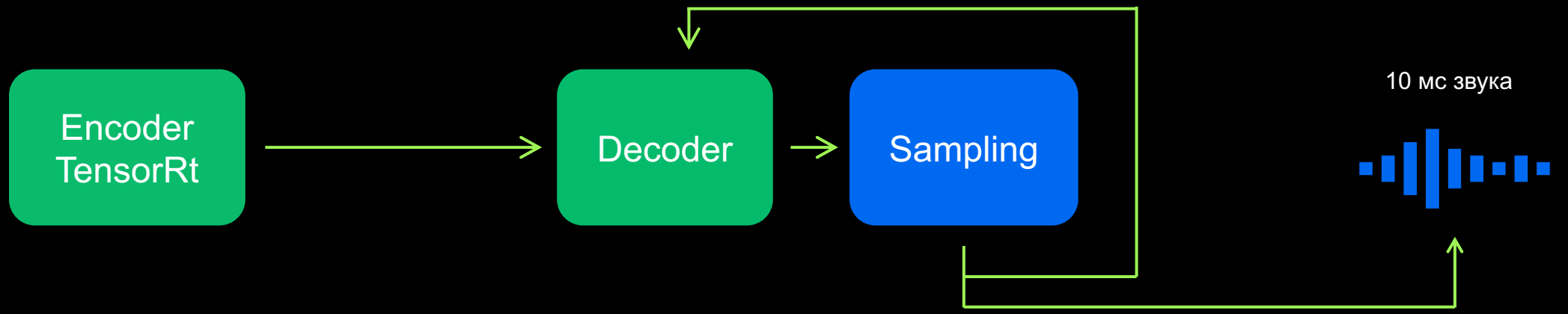
```
1       template <typename T, index_t BatchSize, typename LPCNetConfig>
2       struct LPCNetBatched {
3               static_assert(!std::is_const_v<T>);
4
5               using Type = T;
6               using StateConfig = typename LPCNetConfig::State;
7               using DecoderConfig = typename LPCNetConfig::Decoder;
8
9               static constexpr auto kBatchSize = BatchSize;
10
11              StateBatched<Type, kBatchSize, StateConfig> state;
12
13
```

Структура для хранения состояния нейросети

```
1   template <typename T, index_t BatchSize, typename LPCNetConfig>
2   struct LPCNetBatched {
3           static_assert(!std::is_const_v<T>);
4
5           using Type = T;
6           using StateConfig = typename LPCNetConfig::State;
7           using DecoderConfig = typename LPCNetConfig::Decoder;
8
9           static constexpr auto kBatchSize = BatchSize;
10
11          StateBatched<Type, kBatchSize, StateConfig> state;
12          DecoderBatched<Type, kBatchSize, DecoderConfig> decoder;
13  };
```

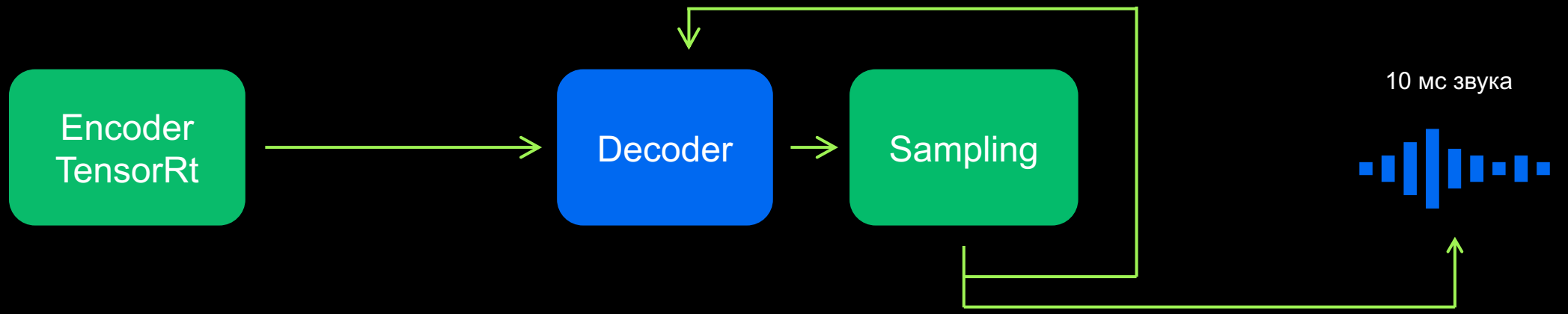
Структура для хранения состояния нейросети

```
1   template <typename T, index_t BatchSize, typename StateConfig>
2   struct StateBatched {
3           static_assert(!std::is_const_v<T>);
4
5           using Type = T;
6
7           static constexpr auto kBatchSize = BatchSize;
8
9           container::VectorBatched<Type, 1, kBatchSize> mem;
10          container::VectorBatched<index_t, 1, kBatchSize> last_exc;
11          container::VectorBatched<Type, StateConfig::kOrder, kBatchSize> lpc;
12          container::VectorBatched<Type, StateConfig::kOrder, kBatchSize> last_sig;
13          container::VectorBatched<int16_t, StateConfig::kFrameSize, kBatchSize> wav;
14  };
```



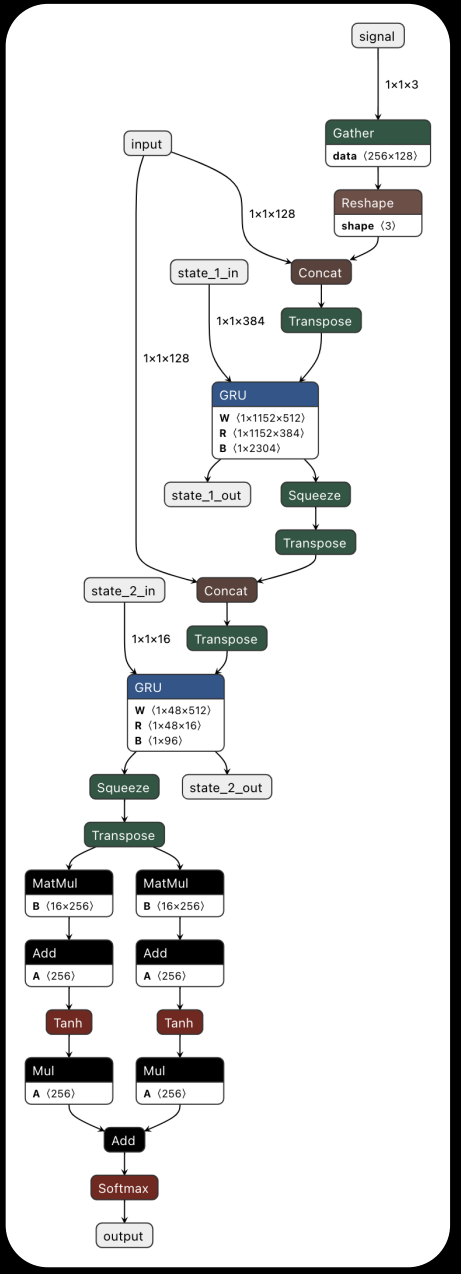
Структура для хранения состояния нейросети

```cpp
template <typename T, index_t BatchSize, typename DecoderConfig>
struct DecoderBatched {
        static_assert(!std::is_const_v<T>);

        using Type = T;

        static constexpr auto kBatchSize = BatchSize;

        container::VectorBatched<Type, DecoderConfig::kInputSize, kBatchSize> input;
        container::VectorBatched<Type, DecoderConfig::kOutputSize, kBatchSize> output;
        container::VectorBatched<Type, DecoderConfig::kGruAHiddenSize, kBatchSize> gru_a_state;
        container::VectorBatched<Type, DecoderConfig::kGruBHiddenSize, kBatchSize> gru_b_state;

        container::VectorBatched<Type, DecoderConfig::kGruAHiddenSize * 3, kBatchSize> precomputed_input_gru_a;
        container::VectorBatched<Type, DecoderConfig::kGruBHiddenSize * 3, kBatchSize> precomputed_input_gru_b;
};
```



Конфигурация во время компиляции

```
using DefaultVoiceConfig = VoiceWeightsConfig<
        /*EmbedConfig=*/EmbedConfig</*InputSize=*/256, /*OutputSize=*/128>,
        /*PrecomputedEmbedConfig=*/PrecomputedEmbedConfig</*InputSize=*/256, /*OutputSize=*/1152>,
        /*GruAConfig=*/GruConfig</*HiddenSize=*/384, /*InputSize=*/512>,
        /*GruBConfig=*/GruConfig</*HiddenSize=*/16, /*InputSize=*/512>,
        /*FullConnectedConfig=*/FullConnectedConfig</*InputSize=*/16, /*OutputSize=*/256>>;



using DefaultLPCNetConfig = LPCNetConfig<
       /*StateConfig=*/StateConfig</*Order=*/22, /*FrameSize=*/240>,
        /*DecoderConfig=*/DecoderConfig</*InputSize=*/128, /*OutputSize=*/256, /*GruAHiddenSize=*/384,
        /*GruBHiddenSize=*/16>>;

```



Главный цикл. Сигнатура

```
1       template <index_t kBlockSize, index_t kBatchSize, index_t kSharedMemory, index_t kIterations, typename T,
2       typename LPCNetConfig, typename VoiceWeightsConfig>
3       __global__ void __launch_bounds__(kBlockSize) LoopDecoderSampling(
```

Главный цикл. Launch bounds

```
1   template <index_t kBlockSize, index_t kBatchSize, index_t kSharedMemory, index_t kIterations, typename T,
2   typename LPCNetConfig, typename VoiceWeightsConfig>
3   __global__ void __launch_bounds__(kBlockSize) LoopDecoderSampling(
```

- В общем случае kernel может быть вызван с любой размерностью сетки
- Это определяется в момент вызова

```
1   inference::LoopDecoderSampling<<<16, 512>>>(...)
2   inference::LoopDecoderSampling<<<32, 1024>>>(...)
3   inference::LoopDecoderSampling<<<128, 768>>>(...)
```

- В зависимости от того, сколько будет потоков в блоке, зависит потребление регистров
- Количество доступных регистров влияет на оптимизации

Главный цикл. Launch bounds

```
1  template <index_t kBlockSize, index_t kBatchSize, index_t kSharedMemory, index_t kIterations, typename T,
2  typename LPCNetConfig, typename VoiceWeightsConfig>
3  __global__ void __launch_bounds__(kBlockSize) LoopDecoderSampling(
```

```
1  __launch_bounds__(128)
```

```
ptxas info    : Compiling entry function
'_ZN8culpcnet9inference22LoopDecoderSampling' for 'sm_70'
ptxas info    : Function properties for _ZN8culpcnet9inference22LoopDecoderSampling
    0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Used 80 registers, 384 bytes cmem[0]
```

```
1  __launch_bounds__(1024)
```

```
ptxas info    : Compiling entry function
'_ZN8culpcnet9inference22LoopDecoderSampling' for 'sm_70'
ptxas info    : Function properties for _ZN8culpcnet9inference22LoopDecoderSampling
    0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Used 64 registers, 384 bytes cmem[0]
```

Главный цикл.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

Главный цикл. Аргументы

```
template <index_t kBlockSize, index_t kBatchSize, index_t kSharedMemory, index_t kIterations, typename T,
typename LPCNetConfig, typename VoiceWeightsConfig>
__global__ void __launch_bounds__(kBlockSize) LoopDecoderSampling(
const typename weights::VoiceWeights<T, VoiceWeightsConfig>::ConstView *voice_weights,
```

Главный цикл. Аргументы

```
template <index_t kBlockSize, index_t kBatchSize, index_t kSharedMemory, index_t kIterations, typename T,
typename LPCNetConfig, typename VoiceWeightsConfig>
__global__ void __launch_bounds__(kBlockSize) LoopDecoderSampling(
const typename weights::VoiceWeights<T, VoiceWeightsConfig>::ConstView *voice_weights,
const index_t *active_voices,
```

Главный цикл. Аргументы

```
template <index_t kBlockSize, index_t kBatchSize, index_t kSharedMemory, index_t kIterations, typename T,
typename LPCNetConfig, typename VoiceWeightsConfig>
__global__ void __launch_bounds__(kBlockSize) LoopDecoderSampling(
const typename weights::VoiceWeights<T, VoiceWeightsConfig>::ConstView *voice_weights,
const index_t *active_voices,
const typename context::LPCNetBatched<T, kBatchSize, LPCNetConfig>::View *lpcnets,
```

Главный цикл. Аргументы

```
template <index_t kBlockSize, index_t kBatchSize, index_t kSharedMemory, index_t kIterations, typename T,
typename LPCNetConfig, typename VoiceWeightsConfig>
__global__ void __launch_bounds__(kBlockSize) LoopDecoderSampling(
const typename weights::VoiceWeights<T, VoiceWeightsConfig>::ConstView *voice_weights,
const index_t *active_voices,
const typename context::LPCNetBatched<T, kBatchSize, LPCNetConfig>::View *lpcnets,
unsigned long long seed) {
```

Главный цикл. Shared memory

```
1   template <index_t kBlockSize, index_t kBatchSize, index_t kSharedMemory, index_t kIterations, typename T,
2   typename LPCNetConfig, typename VoiceWeightsConfig>
3   __global__ void __launch_bounds__(kBlockSize) LoopDecoderSampling(
4   const typename weights::VoiceWeights<T, VoiceWeightsConfig>::ConstView *voice_weights,
5   const index_t *active_voices,
6   const typename context::LPCNetBatched<T, kBatchSize, LPCNetConfig>::View *lpcnets,
7   unsigned long long seed) {
8           extern __shared__ char buffer[]; // аллокация буфера shared memory для использования в дальнейшем
9           char *curr = buffer;
10          char *const end = curr + kSharedMemory;
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

Главный цикл. Загрузка весов и состояний

```
template <index_t kBlockSize, index_t kBatchSize, index_t kSharedMemory, index_t kIterations, typename T,
typename LPCNetConfig, typename VoiceWeightsConfig>
__global__ void __launch_bounds__(kBlockSize) LoopDecoderSampling(
const typename weights::VoiceWeights<T, VoiceWeightsConfig>::ConstView *voice_weights,
const index_t *active_voices,
const typename context::LPCNetBatched<T, kBatchSize, LPCNetConfig>::View *lpcnets,
unsigned long long seed) {
        extern __shared__ char buffer[]; // аллокация буфера shared memory для использования в дальнейшем
        char *curr = buffer;
        char *const end = curr + kSharedMemory;

        const auto &this_lpcnet = lpcnets[blockIdx.x]; // определение элемента для обработки в текущем блоке
        const auto &this_voice = voice_weights[active_voices[blockIdx.x]]; // определение весов для обработки в текущем блоке
```

Главный цикл. Кеширование весов и состояний

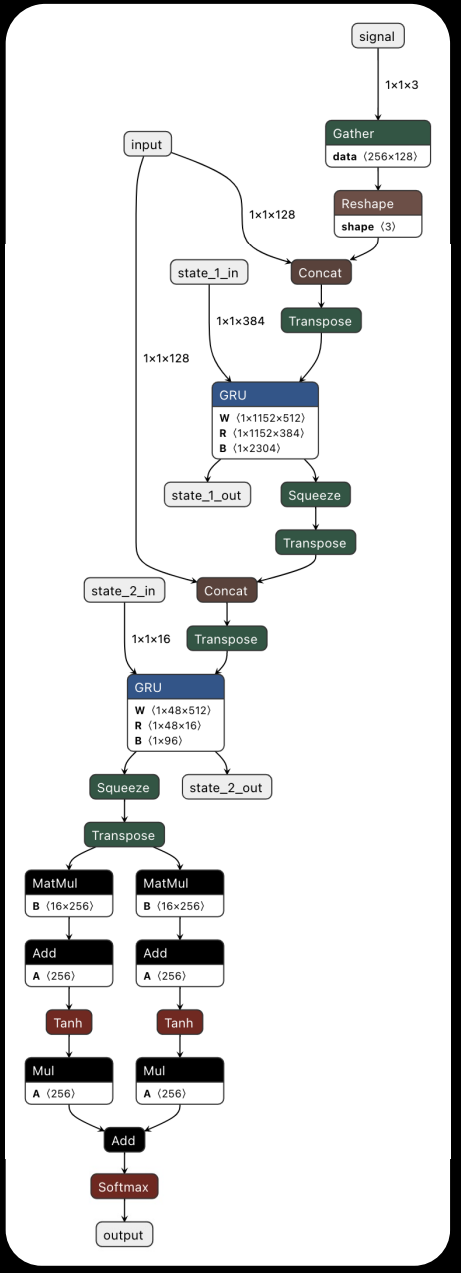
```
1   template <index_t kBlockSize, index_t kBatchSize, index_t kSharedMemory, index_t kIterations, typename T,
2   typename LPCNetConfig, typename VoiceWeightsConfig>
3   __global__ void __launch_bounds__(kBlockSize) LoopDecoderSampling(
4   const typename weights::VoiceWeights<T, VoiceWeightsConfig>::ConstView *voice_weights,
5   const index_t *active_voices,
6   const typename context::LPCNetBatched<T, kBatchSize, LPCNetConfig>::View *lpcnets,
7   unsigned long long seed) {
8           extern __shared__ char buffer[]; // аллокация буфера shared memory для использования в дальнейшем
9           char *curr = buffer;
10          char *const end = curr + kSharedMemory;
11
12          const auto &this_lpcnet = lpcnets[blockIdx.x]; // определение элемента для обработки в текущем блоке
13          const auto &this_voice = voice_weights[active_voices[blockIdx.x]]; // определение весов для обработки в текущем блоке
14
15          auto voice_cache = AllocVoiceCacheNew<T, VoiceWeightsConfig>(curr, end, this_voice); // инициализация кешей в shared memory
16          auto decoder_cache = AllocDecoderCacheBatched<T, kBatchSize, LPCNetConfig::Decoder, VoiceWeightsConfig>(curr, end, this_lpcnet.decoder);
17          auto state_cache = AllocStateCacheBatched<T, kBatchSize, LPCNetConfig::State>(curr, end, this_lpcnet.state);
18
19
20
21
22
23
24
25
26
27
28
29
30
```

Главный цикл. Декодер и семплирование

```
1   template <index_t kBlockSize, index_t kBatchSize, index_t kSharedMemory, index_t kIterations, typename T,
2   typename LPCNetConfig, typename VoiceWeightsConfig>
3   __global__ void __launch_bounds__(kBlockSize) LoopDecoderSampling(
4   const typename weights::VoiceWeights<T, VoiceWeightsConfig>::ConstView *voice_weights,
5   const index_t *active_voices,
6   const typename context::LPCNetBatched<T, kBatchSize, LPCNetConfig>::View *lpcnets,
7   unsigned long long seed) {
8           extern __shared__ char buffer[]; // аллокация буфера shared memory для использования в дальнейшем
9           char *curr = buffer;
10          char *const end = curr + kSharedMemory;
11
12          const auto &this_lpcnet = lpcnets[blockIdx.x]; // определение элемента для обработки в текущем блоке
13          const auto &this_voice = voice_weights[active_voices[blockIdx.x]]; // определение весов для обработки в текущем блоке
14
15          auto voice_cache = AllocVoiceCacheNew<T, VoiceWeightsConfig>(curr, end, this_voice); // инициализация кешей в shared memory
16          auto decoder_cache = AllocDecoderCacheBatched<T, kBatchSize, LPCNetConfig::Decoder, VoiceWeightsConfig>(curr, end, this_lpcnet.decoder);
17          auto state_cache = AllocStateCacheBatched<T, kBatchSize, LPCNetConfig::State>(curr, end, this_lpcnet.state);
18
19          for (index_t iter = 0; iter < kIterations; iter++) {
20                  // decoder
21                  ProcessDecoderIteration<kBlockSize, kBatchSize>(decoder_cache, voice_cache, this_voice);
22
23                  // sampling
24                  PostIterationWarpSync(state_cache, decoder_cache, iter, &curand_state);
25          }
26
27          // копирование результата обратно в global memory
28          CopyResults(state_cache, this_lpcnet.state);
29          CopyResults(decoder_cache, this_lpcnet.decoder);
30  }
```

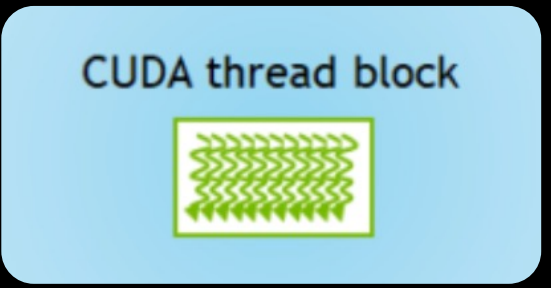
Декодер

```
1   template <index_t kBlockSize, index_t kBatchSize, typename T, typename DecoderConfig, typename VoiceWeightsConfig>
2   __device__ void ProcessDecoderIteration(
3   const DecoderCacheBatched<T, kBatchSize, DecoderConfig, VoiceWeightsConfig> &decoder_cache,
4   const VoiceCacheNew<T, VoiceWeightsConfig> &voice_cache,
5   const typename weights::VoiceWeights<T, VoiceWeightsConfig>::ConstView &voice) {
6
7           Gather(decoder_cache, voice)
8
9           GruA(decoder_cache, voice_cache, voice);
10
11          GruB(decoder_cache, voice_cache, voice);
12
13          FullyConnected(decoder_cache, voice_cache);
14
15          Softmax(decoder_cache);
16  }
```



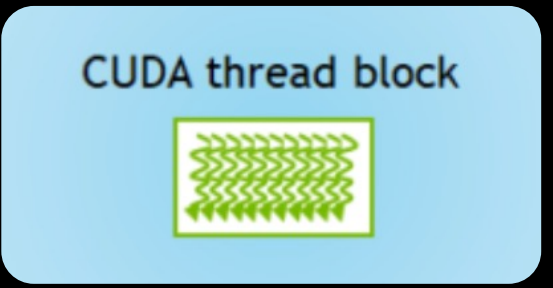
Декодер. Синхронизация

```
1   template <index_t kBlockSize, index_t kBatchSize, typename T, typename DecoderConfig, typename VoiceWeightsConfig>
2   __device__ void ProcessDecoderIteration(
3   const DecoderCacheBatched<T, kBatchSize, DecoderConfig, VoiceWeightsConfig> &decoder_cache,
4   const VoiceCacheNew<T, VoiceWeightsConfig> &voice_cache,
5   const typename weights::VoiceWeights<T, VoiceWeightsConfig>::ConstView &voice) {
6
7           Gather(decoder_cache, voice);
8
9           GruA(decoder_cache, voice_cache, voice);
10
11          GruB(decoder_cache, voice_cache, voice);
12
13          FullyConnected(decoder_cache, voice_cache);
14
15          Softmax(decoder_cache);
16  }
```



Декодер. Синхронизация

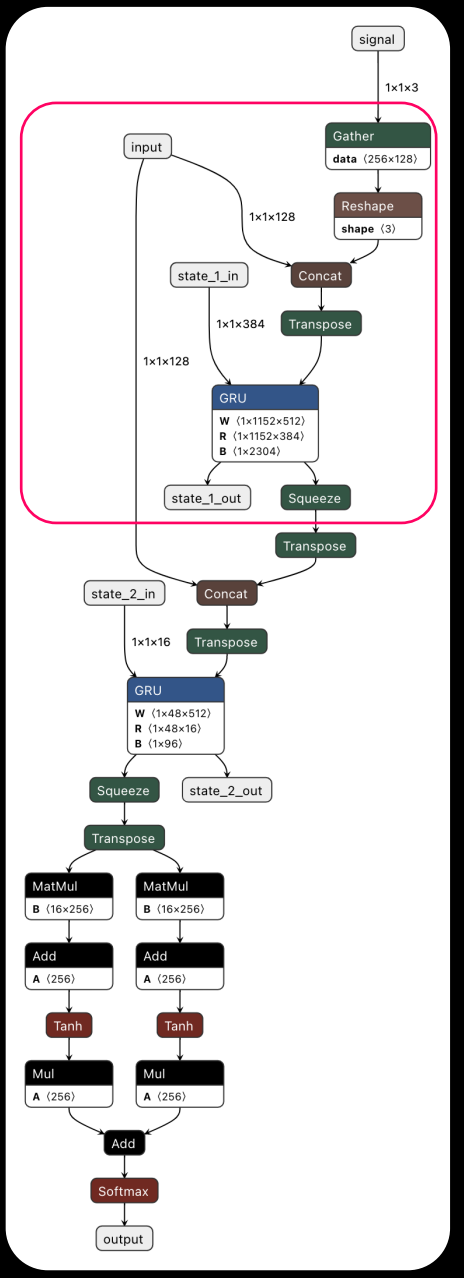
```
1   template <index_t kBlockSize, index_t kBatchSize, typename T, typename DecoderConfig, typename VoiceWeightsConfig>
2   __device__ void ProcessDecoderIteration(
3   const DecoderCacheBatched<T, kBatchSize, DecoderConfig, VoiceWeightsConfig> &decoder_cache,
4   const VoiceCacheNew<T, VoiceWeightsConfig> &voice_cache,
5   const typename weights::VoiceWeights<T, VoiceWeightsConfig>::ConstView &voice) {
6
7           Gather(decoder_cache, voice);
8           cooperative_groups::thread_block::sync();
9           GruA(decoder_cache, voice_cache, voice);
10          cooperative_groups::thread_block::sync();
11          GruB(decoder_cache, voice_cache, voice);
12          cooperative_groups::thread_block::sync();
13          FullyConnected(decoder_cache, voice_cache);
14          cooperative_groups::thread_block::sync();
15          Softmax(decoder_cache);
16  }
```



Декодер. Узкие места

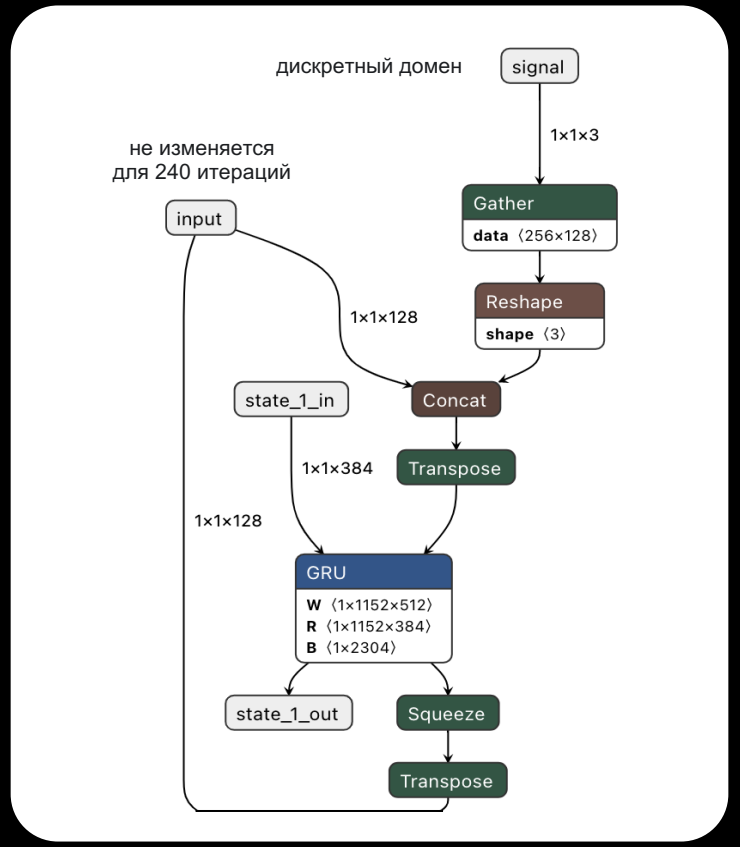
```
1   template <index_t kBlockSize, index_t kBatchSize, typename T, typename DecoderConfig, typename VoiceWeightsConfig>
2   __device__ void ProcessDecoderIteration(
3   const DecoderCacheBatched<T, kBatchSize, DecoderConfig, VoiceWeightsConfig> &decoder_cache,
4   const VoiceCacheNew<T, VoiceWeightsConfig> &voice_cache,
5   const typename weights::VoiceWeights<T, VoiceWeightsConfig>::ConstView &voice) {
6
7           Gather(decoder_cache, voice);
8           cooperative_groups::thread_block::sync();
9           GruA(decoder_cache, voice_cache, voice);
10          cooperative_groups::thread_block::sync();
11          GruB(decoder_cache, voice_cache, voice);
12          cooperative_groups::thread_block::sync();
13          FullyConnected(decoder_cache, voice_cache);
14          cooperative_groups::thread_block::sync();
15          Softmax(decoder_cache);
16  }
```

Более 90% весов



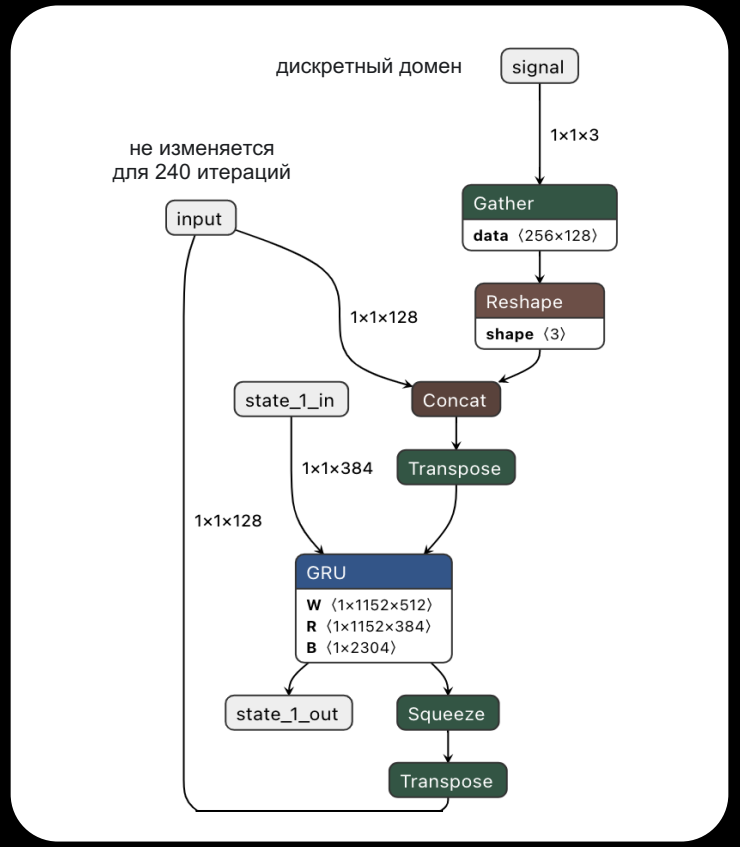
Декодер. Узкие места

- Произведение concat_out на матрицу W: (1x1152x512)x(1x1x512)
- Произведение state_1_in на матрицу R: (1x1152x384)x(1x1x384)



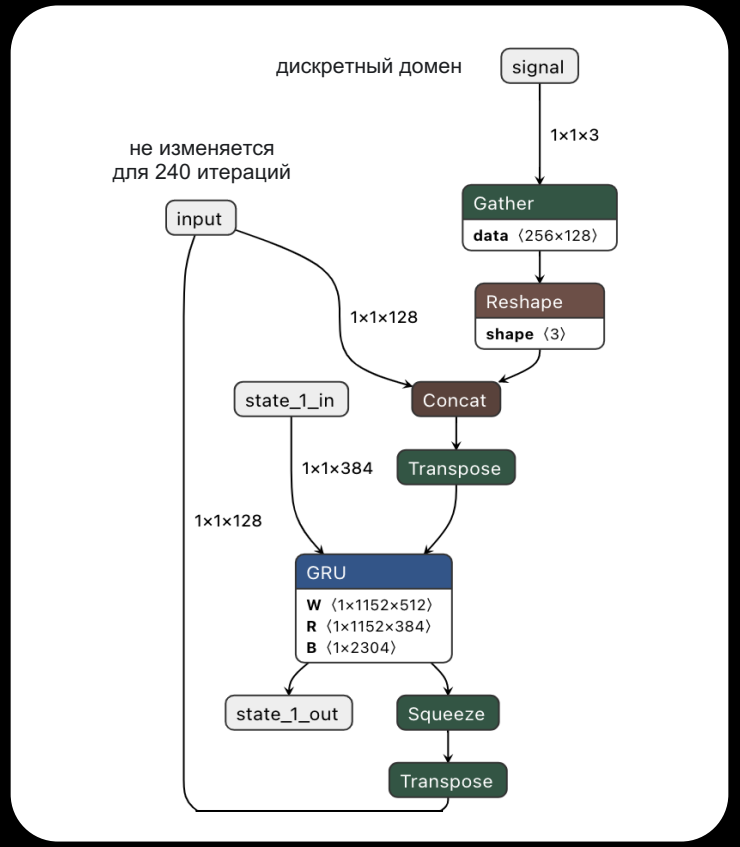
Декодер. Узкие места

- Произведение concat_out на матрицу W: (1x1152x512)x(1x1x512) – можем предподсчитать
- Произведение state_1_in на матрицу R: (1x1152x384)x(1x1x384) – на самом деле R разреженная матрица.

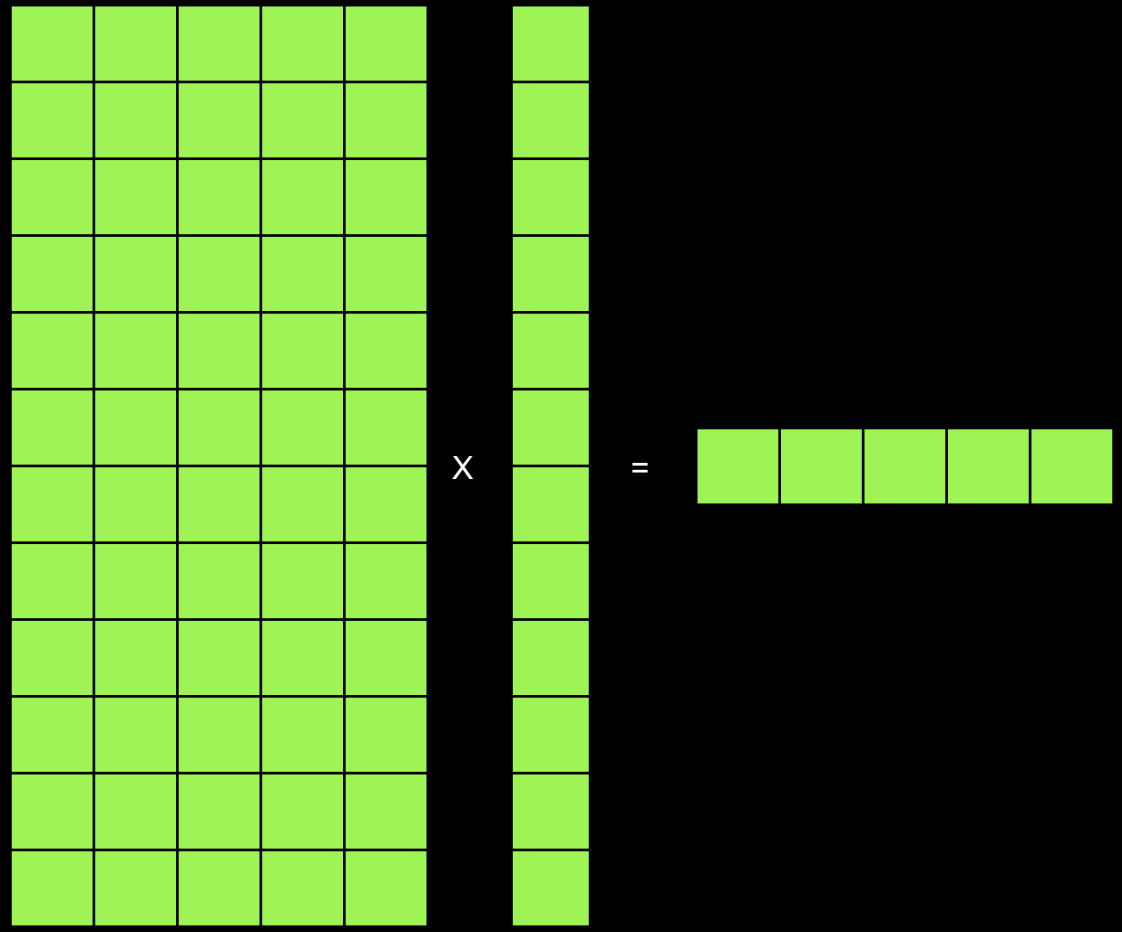


Декодер. Узкие места

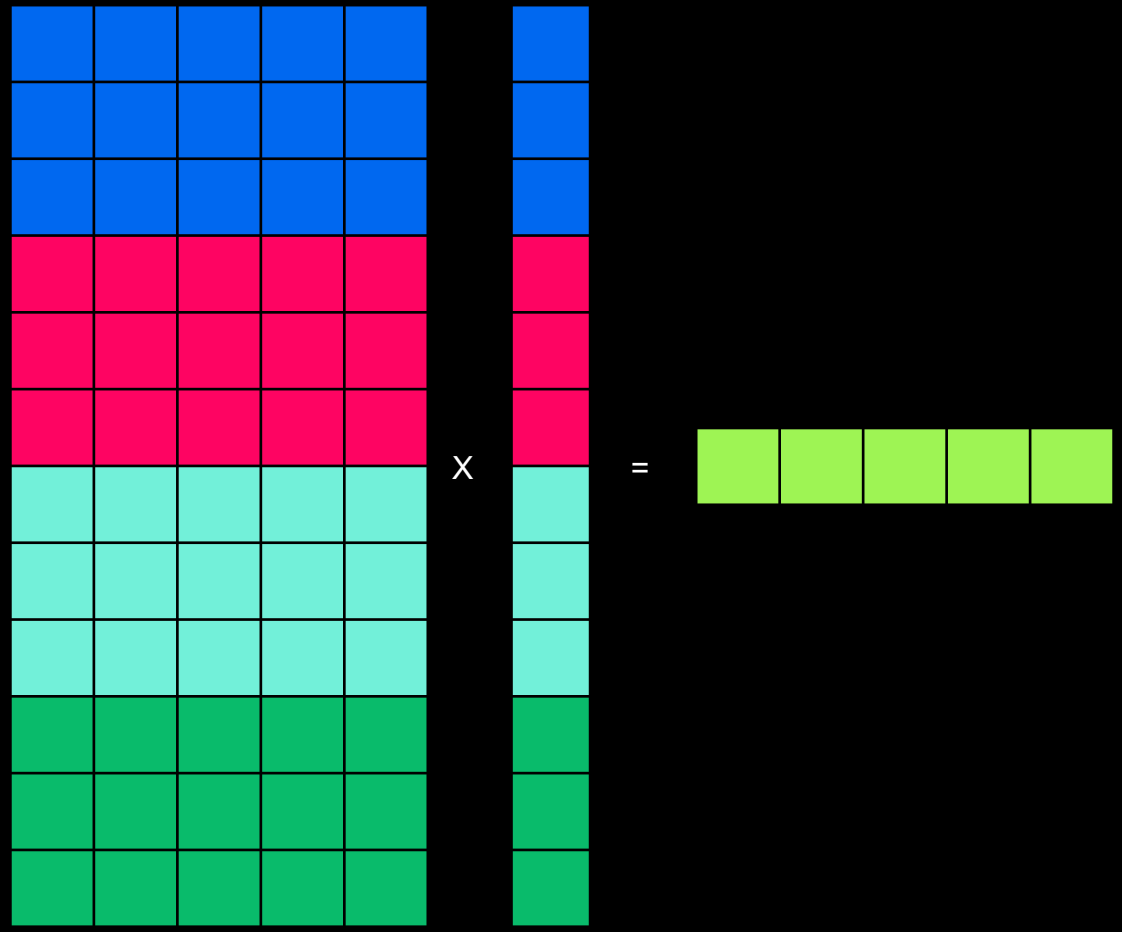
- Произведение concat_out на матрицу W: (1x1152x512)x(1x1x512) – можем предподсчитать
- Произведение state_1_in на матрицу R: (1x1152x384)x(1x1x384) – на самом деле R разреженная матрица. Бонус: она блочная, блоки размером (16x1)



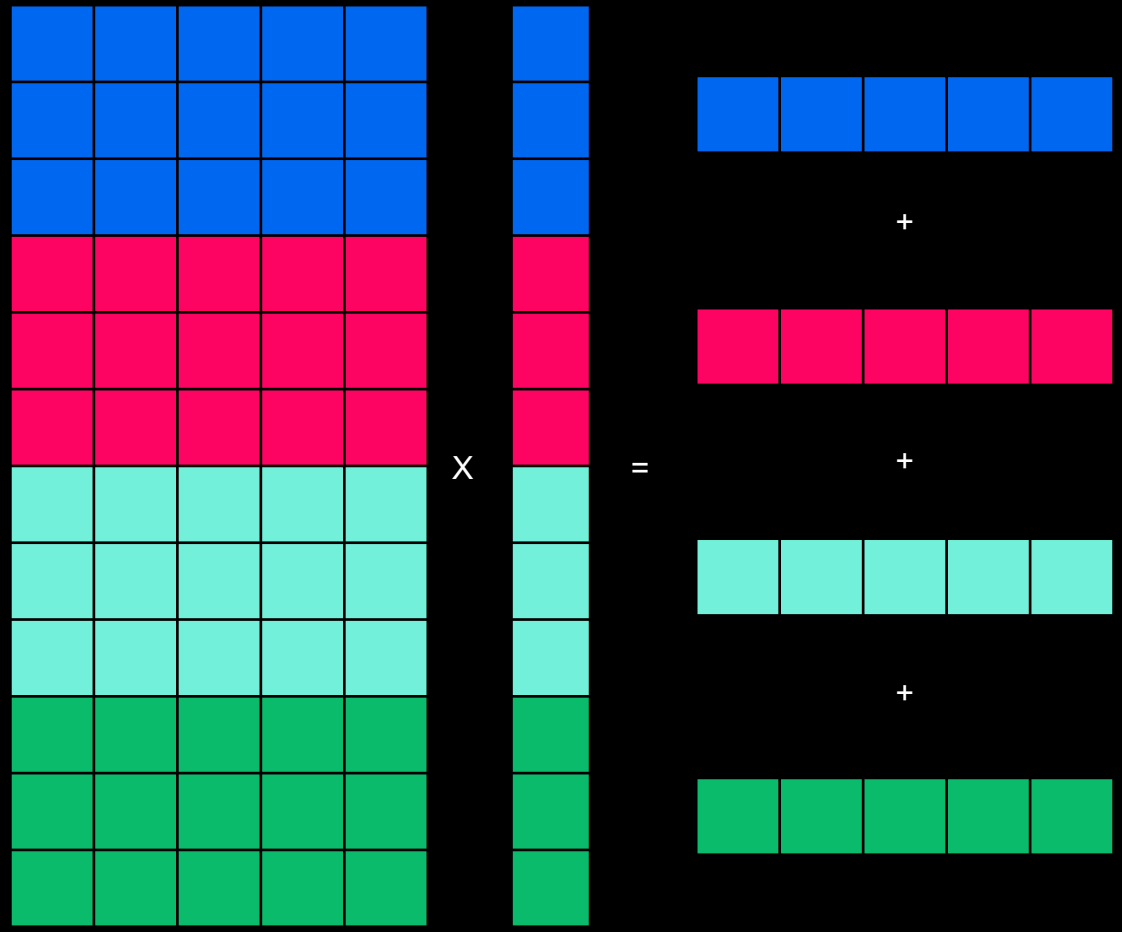
Декодер. Трюк с предподсчетом



Декодер. Трюк с предподсчетом



Декодер. Трюк с предподсчетом



Декодер. Трюк с предподсчетом

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

Декодер. Трюк с предподсчетом

```
template <index_t kBlockSize, index_t kBatchSize, typename T, typename DecoderConfig, typename VoiceWeightsConfig>
__device__ void Gather(
const DecoderCacheBatched<T, kBatchSize, DecoderConfig, VoiceWeightsConfig> &decoder_cache,
const typename weights::VoiceWeights<T, VoiceWeightsConfig>::ConstView &voice) {
```

Декодер. Трюк с предподсчетом

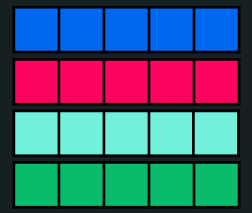
```
template <index_t kBlockSize, index_t kBatchSize, typename T, typename DecoderConfig, typename VoiceWeightsConfig>
__device__ void Gather(
const DecoderCacheBatched<T, kBatchSize, DecoderConfig, VoiceWeightsConfig> &decoder_cache,
const typename weights::VoiceWeights<T, VoiceWeightsConfig>::ConstView &voice) {
        constexpr auto kThredsPerBatch = kBlockSize / kBatchSize;
        static_assert(kBlockSize % kBatchSize == 0);

        const auto batchItem = threadIdx.x / kThredsPerBatch;
        const auto threadIdInsideElement = threadIdx.x % kThredsPerBatch;
```

Декодер. Трюк с предподсчетом

```
1   template <index_t kBlockSize, index_t kBatchSize, typename T, typename DecoderConfig, typename VoiceWeightsConfig>
2   __device__ void Gather(
3   const DecoderCacheBatched<T, kBatchSize, DecoderConfig, VoiceWeightsConfig> &decoder_cache,
4   const typename weights::VoiceWeights<T, VoiceWeightsConfig>::ConstView &voice) {
5           constexpr auto kThredsPerBatch = kBlockSize / kBatchSize;
6           static_assert(kBlockSize % kBatchSize == 0);
7
8           const auto batchItem = threadIdx.x / kThredsPerBatch;
9           const auto threadIdInsideElement = threadIdx.x % kThredsPerBatch;
10
11          const auto signal = decoder_cache.signal.BatchItem(batchItem);
12          const auto precomputed_input_gru_a = decoder_cache.precomputed_input_gru_a.BatchItem(batchItem).Vectorize2();
13          const auto gru_a_x_term = decoder_cache.gru_a_x_term.BatchItem(batchItem).Vectorize2();
14
15
16
17
18
19
20
21
22
23
24
25
26
```

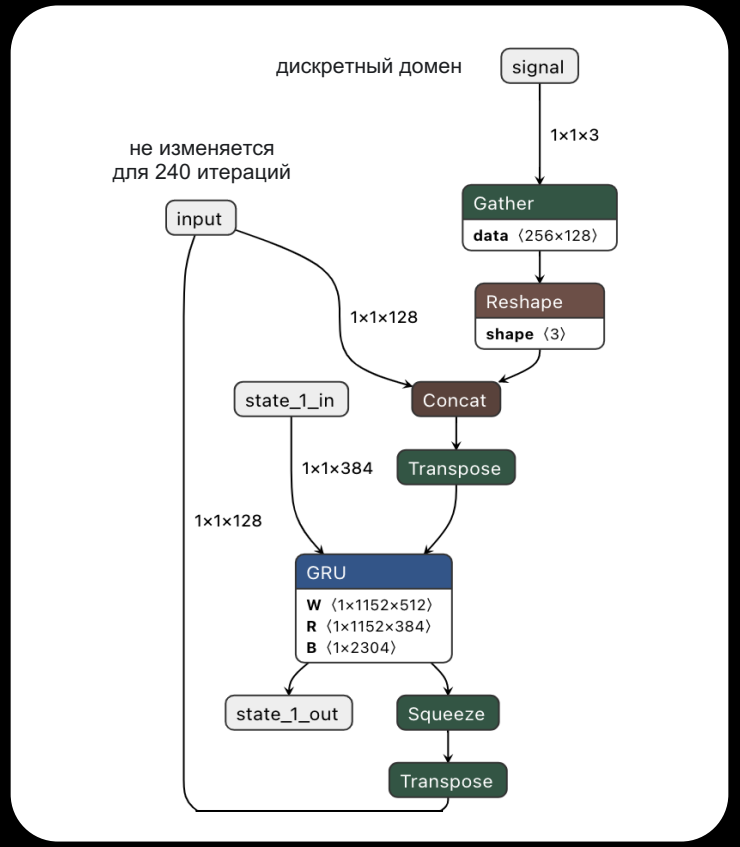
Декодер. Трюк с предподсчетом

```
1   template <index_t kBlockSize, index_t kBatchSize, typename T, typename DecoderConfig, typename VoiceWeightsConfig>
2   __device__ void Gather(
3   const DecoderCacheBatched<T, kBatchSize, DecoderConfig, VoiceWeightsConfig> &decoder_cache,
4   const typename weights::VoiceWeights<T, VoiceWeightsConfig>::ConstView &voice) {
5           constexpr auto kThredsPerBatch = kBlockSize / kBatchSize;
6           static_assert(kBlockSize % kBatchSize == 0);
7
8           const auto batchItem = threadIdx.x / kThredsPerBatch;
9           const auto threadIdInsideElement = threadIdx.x % kThredsPerBatch;
10
11          const auto signal = decoder_cache.signal.BatchItem(batchItem);
12          const auto precomputed_input_gru_a = decoder_cache.precomputed_input_gru_a.BatchItem(batchItem).Vectorize2();
13          const auto gru_a_x_term = decoder_cache.gru_a_x_term.BatchItem(batchItem).Vectorize2();
14
15          for (auto tid = threadIdInsideElement; tid < gru_a_x_term.Size; tid += kThredsPerBatch) {
16                  auto [s_0, s_1, s_2] = signal.data[0];
17
18                  auto a = precomputed_input_gru_a.data[tid];
19
20                  auto b = voice.precomputed_embed.precomputed_signal_0.Col(s_0).Vectorize2().data[tid];
21                  auto c = voice.precomputed_embed.precomputed_signal_1.Col(s_1).Vectorize2().data[tid];
22                  auto d = voice.precomputed_embed.precomputed_signal_2.Col(s_2).Vectorize2().data[tid];
23
24                  gru_a_x_term.data[tid] = a + b + c + d;
25          }
26  }
```



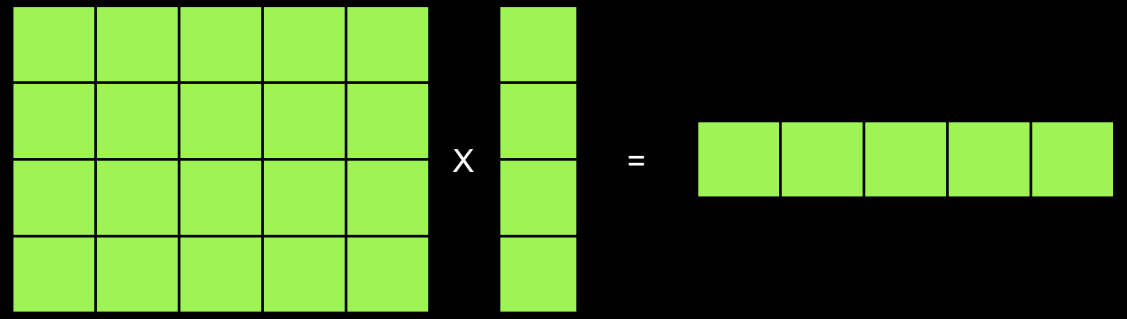
Декодер. Узкие места

- Произведение concat_out на матрицу W: (1x1152x512)x(1x1x512) – можем предподсчитать
- Произведение state_1_in на матрицу R: (1x1152x384)x(1x1x384) – на самом деле R разреженная матрица. Бонус: она блочная, блоки размером (16x1)



Декодер. Dense GRU

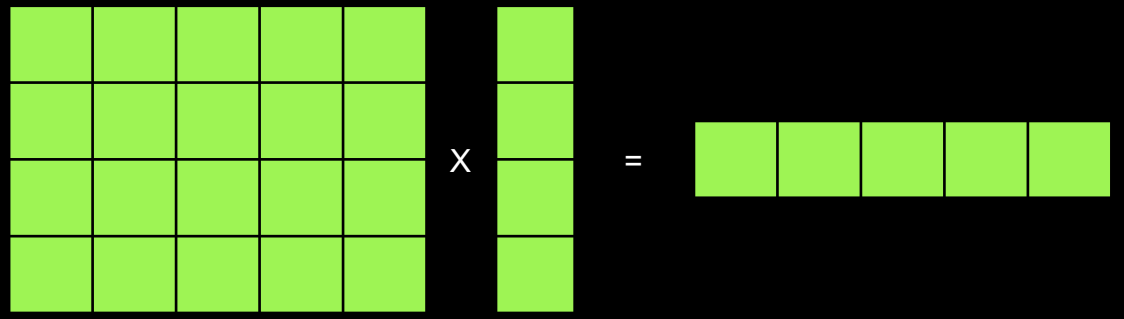
```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```



Col major

Декодер. Dense GRU

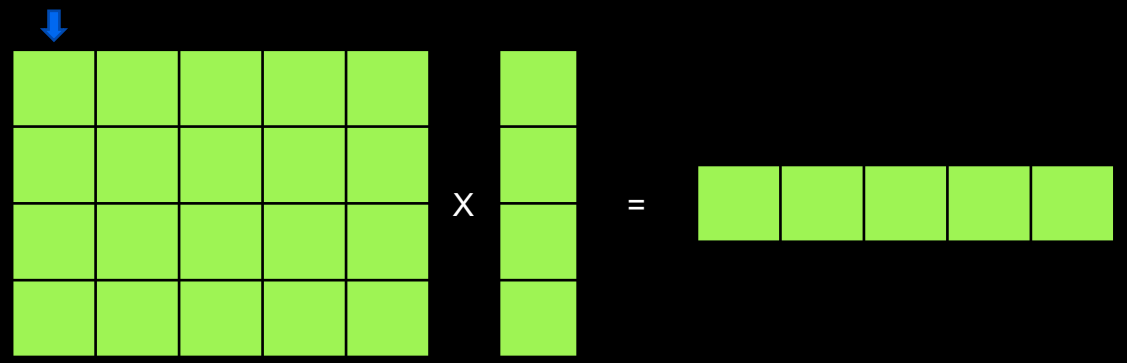
```
1       template <index_t kThreadPerTile, typename T1, typename T2, typename T3, index_t Rows, index_t Cols>
2       __device__ void MatrixTransByVector(
3       container::MatrixView<T1, Rows, Cols> a,
4       container::VectorView<T2, Rows> x,
5       container::VectorView<T3, Cols> y) {
6               const auto this_tile = cooperative_groups::tiled_partition<kThreadPerTile>(cooperative_groups::this_thread_block());
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```



Col major

Декодер. Dense GRU

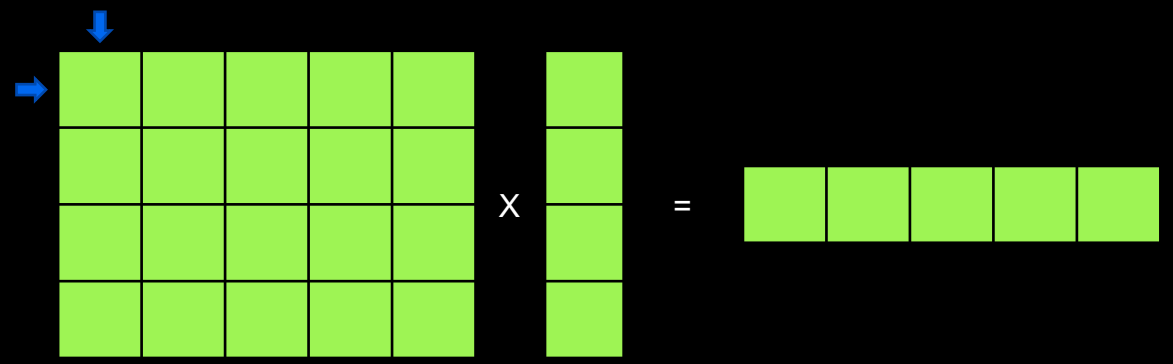
```
1   template <index_t kThreadPerTile, typename T1, typename T2, typename T3, index_t Rows, index_t Cols>
2   __device__ void MatrixTransByVector(
3   container::MatrixView<T1, Rows, Cols> a,
4   container::VectorView<T2, Rows> x,
5   container::VectorView<T3, Cols> y) {
6           const auto this_tile = cooperative_groups::tiled_partition<kThreadPerTile>(cooperative_groups::this_thread_block());
7           for (index_t col = this_tile.meta_group_rank(); col < Cols; col += this_tile.meta_group_size()) {
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```



Col major

Декодер. Dense GRU

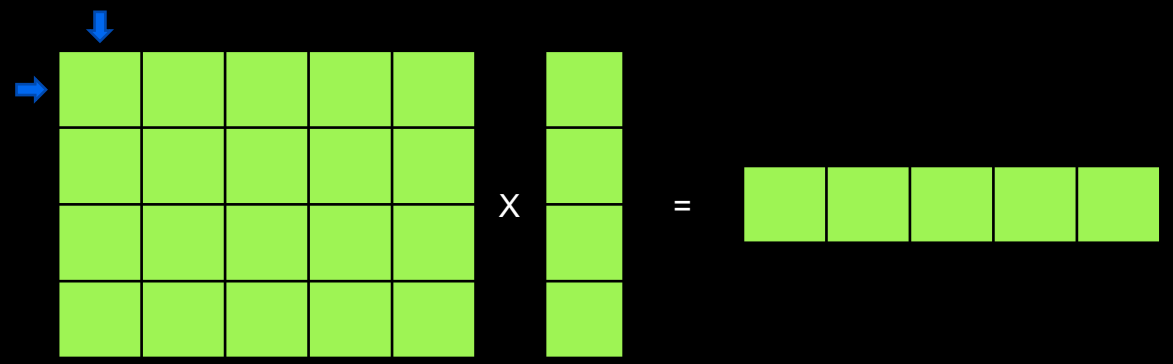
```
1   template <index_t kThreadPerTile, typename T1, typename T2, typename T3, index_t Rows, index_t Cols>
2   __device__ void MatrixTransByVector(
3   container::MatrixView<T1, Rows, Cols> a,
4   container::VectorView<T2, Rows> x,
5   container::VectorView<T3, Cols> y) {
6           const auto this_tile = cooperative_groups::tiled_partition<kThreadPerTile>(cooperative_groups::this_thread_block());
7           for (index_t col = this_tile.meta_group_rank(); col < Cols; col += this_tile.meta_group_size()) {
8                   T3 sum{};
9                   const auto a_col = a.data + col * Rows;
10
11                  for (index_t row = this_tile.thread_rank(); row < Rows; row += this_tile.size()) {
12                          auto a_value = a_col[row];
13                          auto x_value = x.data[row];
14                          sum += a_value * x_value;
15                  }
16
17
18
19
20
21
```



Col major

Декодер. Dense GRU

```
1   template <index_t kThreadPerTile, typename T1, typename T2, typename T3, index_t Rows, index_t Cols>
2   __device__ void MatrixTransByVector(
3   container::MatrixView<T1, Rows, Cols> a,
4   container::VectorView<T2, Rows> x,
5   container::VectorView<T3, Cols> y) {
6           const auto this_tile = cooperative_groups::tiled_partition<kThreadPerTile>(cooperative_groups::this_thread_block());
7           for (index_t col = this_tile.meta_group_rank(); col < Cols; col += this_tile.meta_group_size()) {
8                   T3 sum{};
9                   const auto a_col = a.data + col * Rows;
10
11                  for (index_t row = this_tile.thread_rank(); row < Rows; row += this_tile.size()) {
12                          auto a_value = a_col[row];
13                          auto x_value = x.data[row];
14                          sum += a_value * x_value;
15                  }
16                  sum = GroupReduceSum(this_tile, sum);
17                  if (this_tile.thread_rank() == 0) {
18                          y.data[col] = sum;
19                  }
20          }
21  }
```

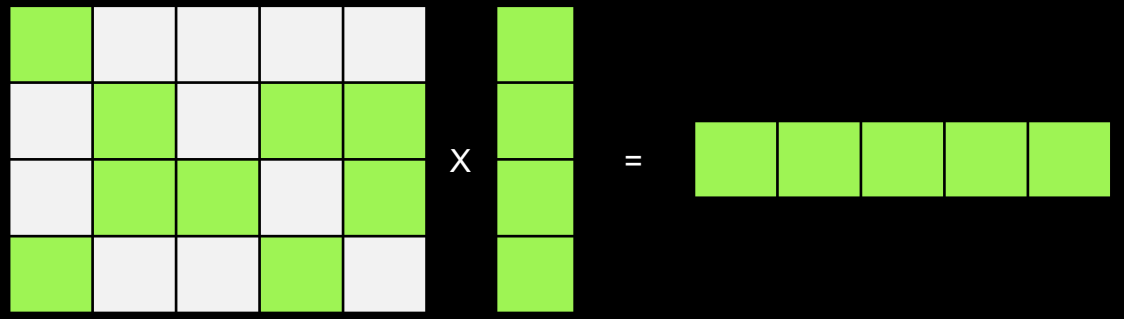


Col major

SBER DEVICES

Декодер. Sparse GRU

```
template <index_t kThreadPerTile, typename T1, typename T2, typename T3, index_t Rows, index_t Cols>
__device__ void SparseMatrixTransByVector(
container::SparseMatrixView<T1, Rows, Cols> a,
container::VectorView<T2, Rows> x,
container::VectorView<T3, Cols> y) {
        const auto this_tile = cooperative_groups::tiled_partition<kThreadPerTile>(cooperative_groups::this_thread_block());
        for (index_t col = this_tile.meta_group_rank(); col < Cols; col += this_tile.meta_group_size()) {
                T3 sum{};
                const auto col_begin = a.col_index[col];
                const auto col_end = a.col_index[col + 1];
                for (uint32_t row = this_tile.thread_rank() + col_begin; row < col_end; row += this_tile.size()) {
                        auto [a_data, row_idx] = a.data_row_index[row];
                        T2 x_data = x.data[row_idx];
                        sum += a_data * x_data;
                }
                sum = GroupReduceSum(this_tile, sum);
                if (this_tile.thread_rank() == 0) {
                        y.data[col] = sum;
                }
        }
}
```



Compressed
sparse column

Декодер. Sparse GRU blocked

Compressed sparse column

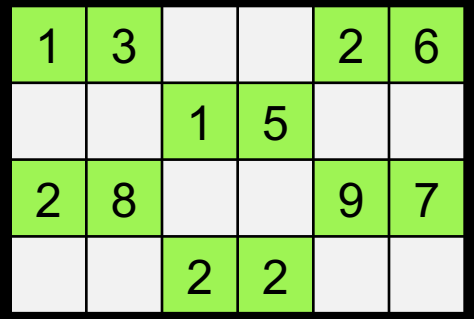
1	3			2	6
		1	5		
2	8			9	7
		2	2		

Compressed sparse column blocked

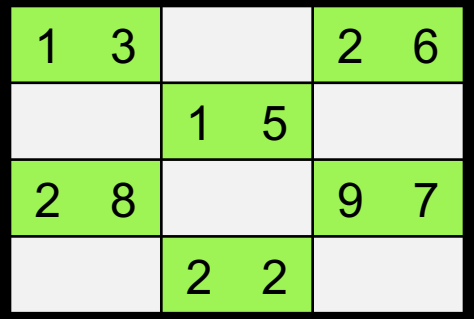
1	3			2	6
		1	5		
2	8			9	7
		2	2		

Декодер. Sparse GRU blocked

Compressed sparse column



Compressed sparse column blocked



data

Декодер. Sparse GRU blocked

Compressed sparse column

1	3			2	6
		1	5		
2	8			9	7
		2	2		

Compressed sparse column blocked

1	3			2	6
		1	5		
2	8			9	7
		2	2		

data	1	2	3	8	1	2	5	2	2	9	6	7

col_index	0	2	4	6	8	10	12

Декодер. Sparse GRU blocked

Compressed sparse column

1	3			2	6
		1	5		
2	8			9	7
		2	2		

Compressed sparse column blocked

1	3			2	6
		1	5		
2	8			9	7
		2	2		

data	1	2	3	8	1	2	5	2	2	9	6	7

col_index	0	2	4	6	8	10	12

row_index	0	2	0	2	1	3	1	3	0	2	0	2

Декодер. Sparse GRU blocked

Compressed sparse column

1	3			2	6
		1	5		
2	8			9	7
		2	2		

data	1	2	3	8	1	2	5	2	2	9	6	7

col_index	0	2	4	6	8	10	12

row_index	0	2	0	2	1	3	1	3	0	2	0	2

Compressed sparse column blocked

1	3			2	6
		1	5		
2	8			9	7
		2	2		

data	1	3	2	8	1	5	2	2	2	6	9	7

Декодер. Sparse GRU blocked

Compressed sparse column

1	3			2	6
		1	5		
2	8			9	7
		2	2		

data	1	2	3	8	1	2	5	2	2	9	6	7
col_index	0	2	4	6	8	10	12					
row_index	0	2	0	2	1	3	1	3	0	2	0	2

Compressed sparse column blocked

1	3			2	6
		1	5		
2	8			9	7
		2	2		

data	1	3	2	8	1	5	2	2	2	6	9	7
col_index	0		2		4		6					

Декодер. Sparse GRU blocked

Compressed sparse column

1	3			2	6
		1	5		
2	8			9	7
		2	2		

data	1	2	3	8	1	2	5	2	2	9	6	7
col_index	0	2	4	6	8	10	12					
row_index	0	2	0	2	1	3	1	3	0	2	0	2

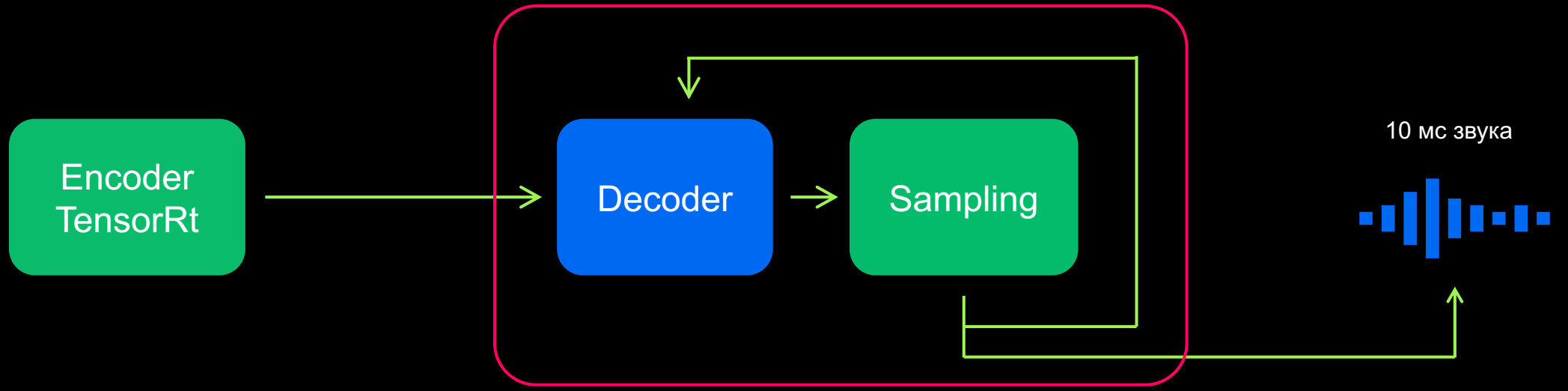
Compressed sparse column blocked

1 3				2 6	
		1 5			
2 8				9 7	
		2 2			

data	1	3	2	8	1	5	2	2	2	6	9	7
col_index	0		2		4		6					
row_index	0	2	1	3	0	2						

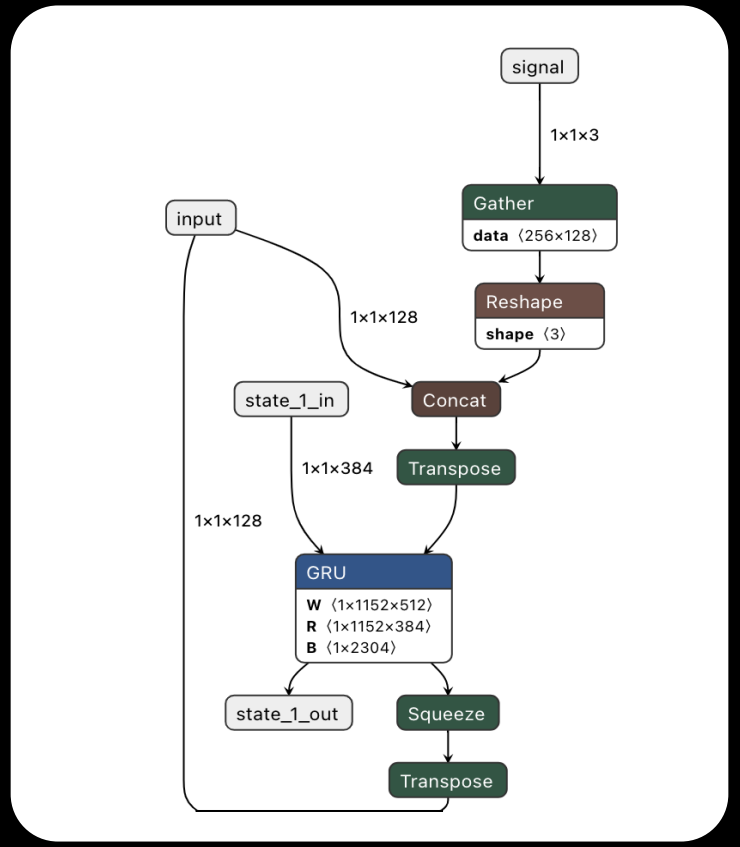
Время работы dense vs sparse vs sparse blocked

- Batch size: 1
- Dense: 7.2 мс
- Sparse: 5 мс 🚀
- Sparse blocked: 2.8 мс 🚀 🚀



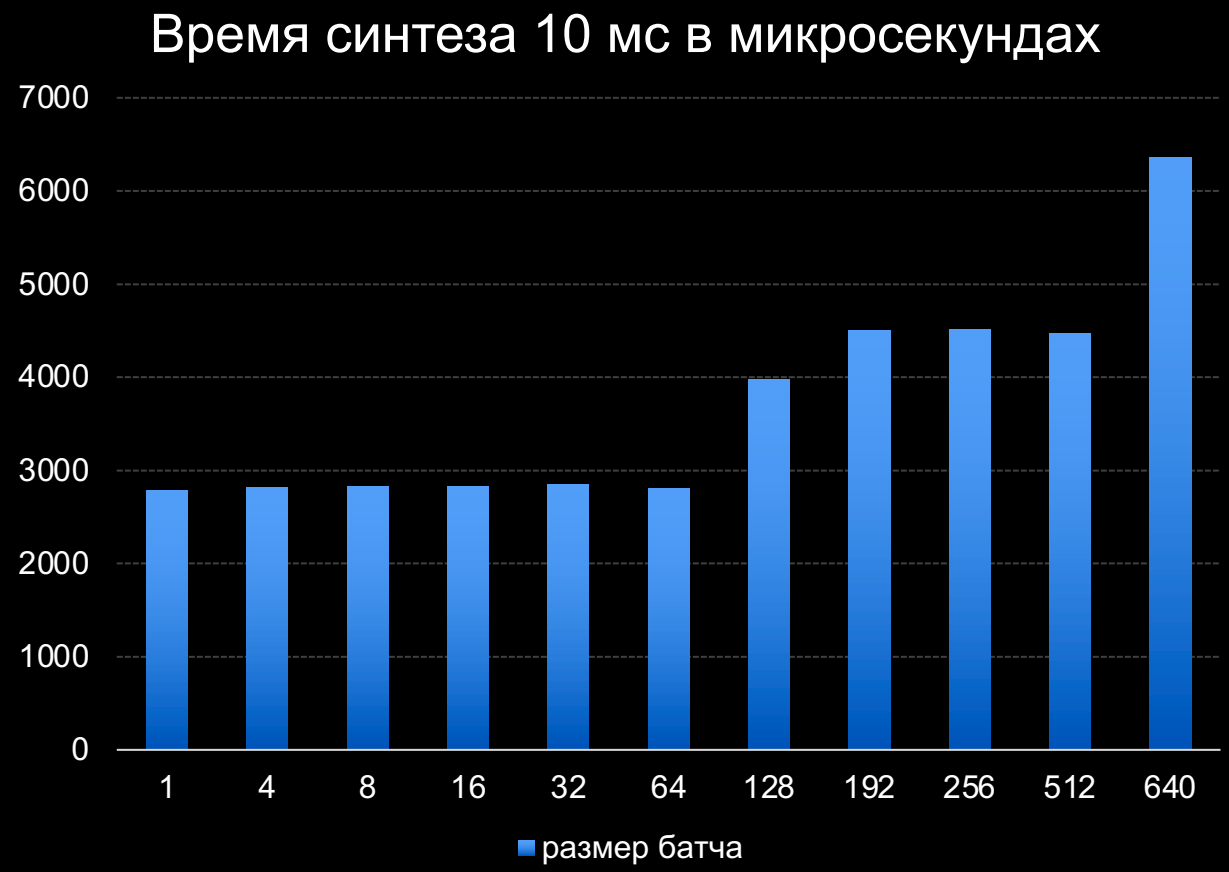
Декодер. Узкие места

- Произведение concat_out на матрицу W: (1x1152x512)x(1x1x512) – можем предподсчитать
- Произведение state_1_in на матрицу R: (1x1152x384)x(1x1x384) – на самом деле R разреженная матрица. Бонус: она блочная, блоки размером (16x1)



Итоговая производительность

- Используя TensorRt получали более 30 мс для батча 1, теперь получили менее 3 мс
- GPU вокодер способен обработать 640 пользователей с RTF < 1



Подведём итог

Существенно улучшили пропускную способность одной из компонент

Стоимость обслуживания уменьшилась от 2 до 3 раз

Использование CUDA C++ позволило получить 10 кратное ускорение относительно TensorRt

SBER DEVICES