Towards to More Flexible: The Reconstruction of Excalibur

JunYi Hu

inlmouse@glasssix.com

2020 年 5 月 21 日

What is Excalibur(for now)

Excalibur is a light weighted Kernel C++ library for some MATH(now move to Julius), IMAGE and CNN operations. This implementation, has been specifically optimized in response to Intel CPU, Nvidia GPU and ARM situation.

- Supports convolutional neural networks and most commonly used image processing operations.
- Supports multiple input and multi-branch structure, can calculate part of the branch.
- No third-party library dependencies in CPU-Only mode, does not rely on BLAS / NNPACK or any other computing framework.
- Pure C++ implementation, cross platform(Windows, x86-Linux, ARM-Linux, Android, iOS and MacOS) support.

What is Excalibur(for now)

- Multi-language support, except C/C++ native support, CSharp(.Net framework and .Net Core), Java/Koltin(Android) interfaces are also included.
- Sophisticated memory management and data structure design, very low cost in CPU and GPU interaction.
- Hard code models into executable file in binary to protect intellectual property.
- Can be registered with custom operations implementation and extended.
- Faster implementation on CPU for various types convolution operation.
- Support Convolution and Inner-product with sparse model.
- Half pricision(float16) support on x86(NVIDIA GPU only); fixed pricision(int8) support on x86 and ARM arch.

Why we need Excalibur

- We started this project in 2017/07 for RedEye3.
- For historical reason, it was designed very simple at first.
- Now, it is far from keeping up with the actual requirements.
- Avoid license conflict: infringement.
- Fully control: extend to everywhere.
- Build team competitiveness: technology reserve.
- Use latest tech in no latency: easy to drag racing.
- Protect our algorithm model: by hardcode.
- Source of invention patent.
- Our Company' s valuation support.
- Show the difference with other teams.

Other Frameworks

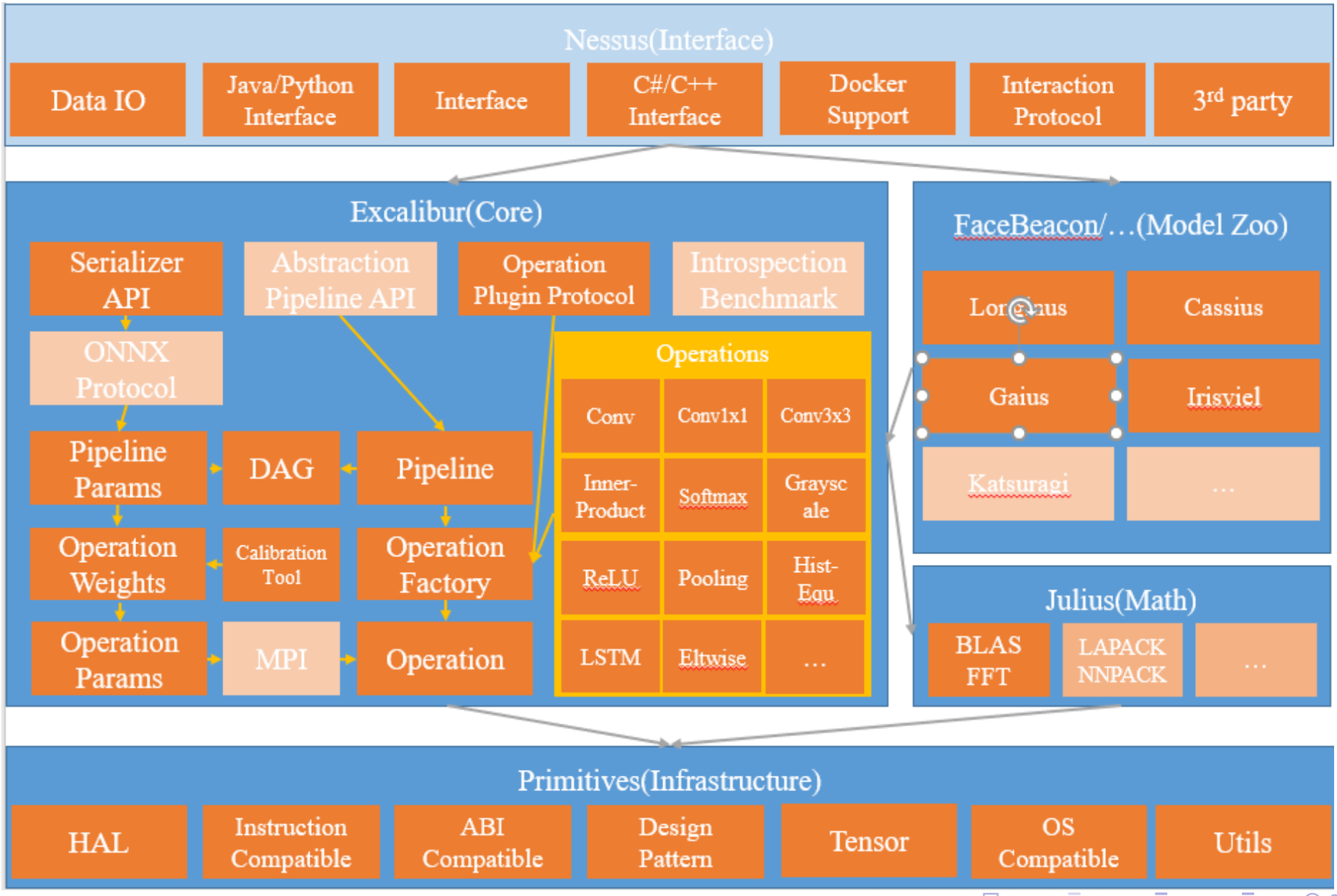
Framework	Leading Organization	Type	Main Author
Caffe	BVLC	Training& Inference	Yangqing Jia
TensorFlow	Google	Training& Inference	Unknown
Caffe2/PyTorch	Facebook	Training& Inference	Unknown
MxNet	Amazon	Training& Inference	Mu Li
CNTK	Microsoft	Training& Inference	MS Cognition Team
PaddlePaddle	Baidu	Training& Inference	Unknown
MindSpore	Huawei	Training& Inference	Unknown
MegEngine	Megvii	Training& Inference	Unknown
Mini-Caffe	–	Inference Only	luoyetx
NCNN	Tencent	Inference Only	nihui
MNN	Alibaba	Inference Only	Unknown
Mace	Xiaomi	Inference Only	Unknown
Tengine	Open AI Lab	Inference Only	圈圈虫
PPL	SenseTime	Inference Only	Yang Gao
Jittor	THU	Inference Only	H.D. Li

Can we do it well?

Of course! We can do it better!
The execution time(ms) of 2 framework on ARM A72

Model	Excalibur	NCNN	Mini-Caffe
Cassius	365	432	1038
Gaius	65	56	213
Longinus-mobile	25	42	N/A
Longinus	164	149	N/A

Overview of Status Quo



syncedmem

```
1  template <typename Dtype>
2  class syncedmem
3  {
4  public:
5  syncedmem();
6  explicit syncedmem(size_t size, int device = -1);
7  ~syncedmem();
8  enum SyncedHead { UNINITIALIZED, HEAD_AT_CPU, HEAD_AT_GPU,
       SYNCED };
9  SyncedHead head() { return head_; }
10 void set_allocator(pool_allocator<Dtype>* allocator);
11 size_t size() { return size_; }
12 const Dtype* cpu_data();
13 const Dtype* gpu_data();
14 Dtype* mutable_cpu_data();
15 Dtype* mutable_gpu_data();
16 #ifdef USE_CUDA
17 void async_gpu_push(const cudaStream_t& stream);
```

syncedmem

```
#endif
private:
Dtype* cpu_ptr_;
Dtype* gpu_ptr_;
#ifdef VULKAN
VkBufferMemory* vkgpu_ptr_;
#endif
pool_allocator<Dtype>* allocator_;
size_t size_;
SyncedHead head_;
bool own_cpu_data_;
bool own_gpu_data_;
int device_;
void check_device();
void to_cpu();
void to_gpu();
};
```

Temsor

```
template <typename Dtype>
class EXPORT_EXCALIBUR_PRIMITIVES tensor : public tensor_
{
// Data pointer
std::shared_ptr<syncedmem<Dtype>> data_;
// Allocator from outside
pool_allocator<Dtype>* allocator_;
// The 4-dim shape of the tensor in" NCHW/NHWC
std::vector<int> shape_;
// For NCHW: n * c * step_
// For NHWC: n * h * step_
size_t count_;

// Pick CUDA/VULKAN support device
int device_;
// Data arrange order
orderType order_;
```

Temsor

```
1  // Size of the data_offset(step_ * sizeof(Dtype) is
       aligned to 16):
2  // h * w in NCHW order
3  // w * c in NHWC order
4  size_t step_;
5
6  public:
7  // empty tensor
8  explicit tensor(orderType order = NCHW, pool_allocator<
       Dtype>* allocator = nullptr);
9  ......
10 };
```

Operation

```
1  template<typename Dtype>
2  class operation
3  {
4  public:
5  explicit operation(const operation_param& param) : params_
       (param) {}
6
7  // inference on cpu
8  void forward_cpu(const std::vector<std::shared_ptr<memory
       ::tensor<Dtype>>>& ^^Ibottoms, std::vector<std::
       shared_ptr<memory::tensor<Dtype>>>& tops);
9
10 // inference on gpu
11 void forward_gpu(
12 #ifdef USE_CUDA
13 cublasHandle_t &cublas_handle_,
14 #ifdef USE_CUDNN
15 cudnnHandle_t cudnn_handle,
```

Operation

```
1  #endif //!USE_CUDNN
2  #endif //!USE_CUDA
3  const std::vector<std::shared_ptr<memory::tensor<Dtype>>>&
       bottoms, ^^Istd::vector<std::shared_ptr<memory::
       tensor<Dtype>>>& tops);
4
5  protected:
6  virtual void forward_cpu_f32(const ^^Istd::vector<std::
       shared_ptr<memory::tensor<float>>>& bottoms, ^^Istd::
       vector<std::shared_ptr<memory::tensor<float>>>& tops)
7  {
8  NOT_IMPLEMENTED;
9  };
10 };
```

Operation Reflector

```
1  template<typename Dtype>
2  class operation_reflector : public singleton<
       operation_reflector<Dtype>>
3  {
4  private:
5  std::map<std::string,
6  std::function<std::shared_ptr<operation<Dtype>>(const
       operation_param&)>> object_map_;
7
8  public:
9  std::shared_ptr<operation<Dtype>> create_object(const
       operation_param& param)
10 {
11 for (auto & x : object_map_)
12 {
```

Operation Reflector

```
1  std::string type = param.type_;
2  std::transform(type.begin(), type.end(), type.begin(), ::
       tolower);
3  if (x.first == std::string("operation_") + type)
4  return x.second(param);
5  }
6  return nullptr;
7  }
8
9  void registe(const std::string &class_name,
10 std::function<std::shared_ptr<operation<Dtype>>(const
       operation_param&)> && generator)
11 {
12 object_map_[class_name] = generator;
13 }
14 };
```

Pipeline

```
1 class pipeline
2 {
3 public:
4 explicit pipeline(const pipeline_param& param)
5 {
6 }
7
8 explicit pipeline(std::string param_file, std::string
      model_file);
9 ~pipeline() {};
```

Pipeline

```
1  void init_weights(std::string model_file);
2
3  private:
4  pipeline_param param_;
5  //  Individual operations in the pipeline
6  std::vector<std::shared_ptr<operation<float>> >
       operations_;
7
8  std::vector<std::vector<std::string>> net_param_str_;
9  std::vector<operation_param> op_params_;
10 std::vector<std::shared_ptr<memory::tensor<float>>>
       featmaps_;
11 DISABLE_COPY_AND_ASSIGN(pipeline);
12 };
```

Pipeline Optimize

Operator Fusion:

- batchnorm - scale
- convolution - batchnorm
- convolutiondepthwise - batchnorm
- deconvolution - batchnorm
- deconvolutiondepthwise - batchnorm
- innerproduct - batchnorm
- convolution - relu
- convolutiondepthwise - relu
- deconvolution - relu
- deconvolutiondepthwise - relu
- innerproduct –relu

Pipeline Optimize

Eliminate No-op Operation:

- innerproduct - dropout
- flatten after global pooling

Prefer Better Operation:

- replace convolution with innerproduct after global pooling

6 Pipeline Infer Type

Type	fmadd type	Feature map type
double64	double64	double64
float32	float32	float32
float16	float16	float16
brain-float16	float32	brain-float16
int8	int8	float32
byte2	byte2	float?/byte2
tfloat32*	?	?

The rise of sparse

- Since NVIDIA launched its new GPU PASCAL A100 last weak, we have an actually sparse matrix dedicated computing hardware: TENSOR CORE3.
- ARM V8.2 enhanced the FMA instruction for float16 and SDOT instruction of int8.
- The success of Int8 and BNN training algorithm by SenseTime and JD coupled with TENSOR CORE3 will burst out new CNN forms.

Introspection Benchmark

As we know in the kinder-garden, there is no single method which is able to handle convolution optimization. Each method(im2col/ MEC/ FFT/ direct/ winograd) in different situations (size, memory, number of cores, energy consumption, etc.) has its own speed advantages.

As for which is the fastest, you will know if you actually run it once. To deal with black boxes with black boxes, there is no need to use a bunch of judgment conditions.

The rest operation is the same.

Why not eager?

We designed Excalibur executor on eager mode at first in 2017.

BUT!

If concentrated on compiler optimization, it becomes complex...

The Achilles' Heel on GPU

Actually, our GPU optimization is a piece of shit.

In some case, the GPU runs the same speed as CPU.

The TENSOR CORE is not enabled at all.

The Mobile GPU is not enabled at all.

We need a heterogeneous computing engineer.

GLASSSIX

Thank you!