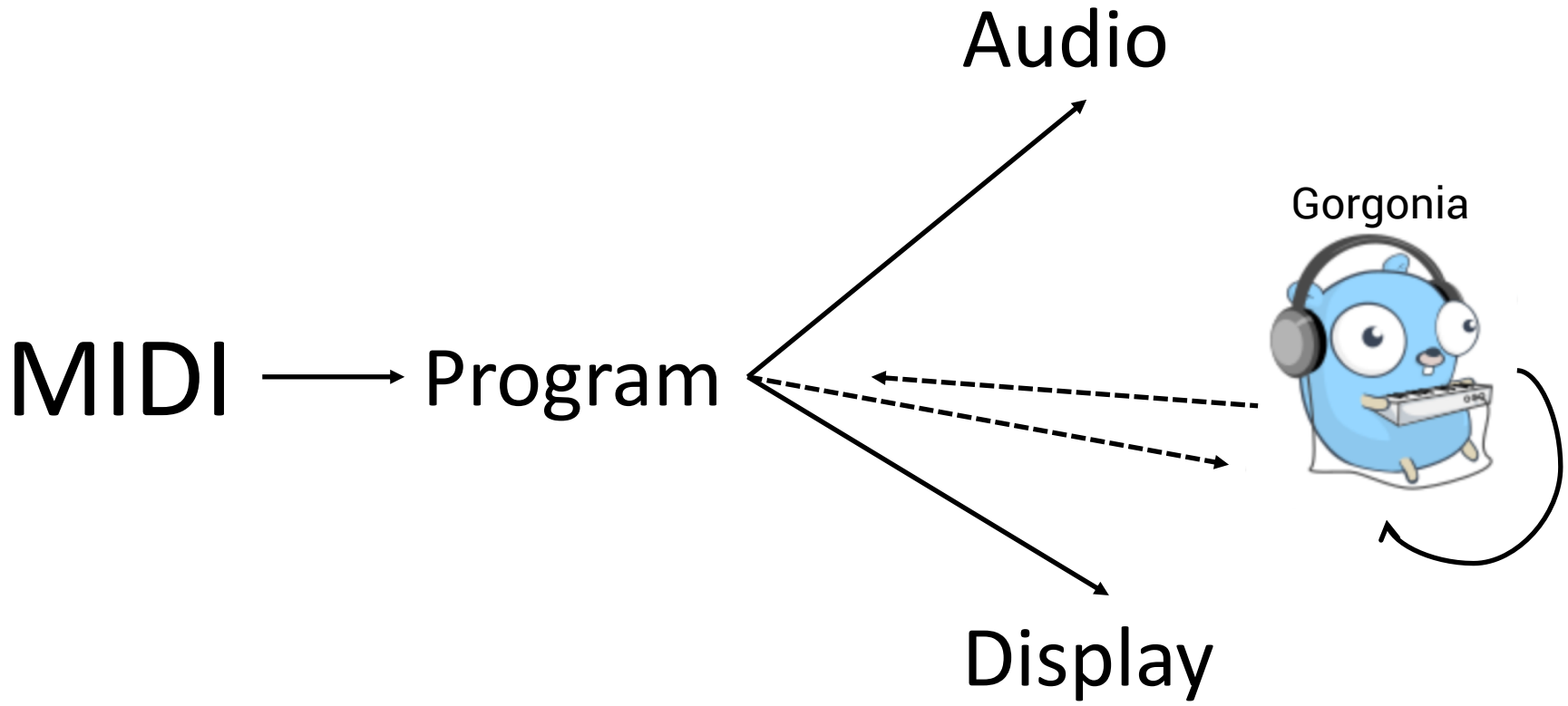
The Art of Bondage

Xuanyi Chew

GopherCon Singapore 2018

The Program



Download cuDNN v7.1.3 (April 17, 2018), for CUDA 9.1

cuDNN v7.1.3 Library for Linux

cuDNN v7.1.3 Library for Linux (Power8/Power9)

cuDNN v7.1.3 Library for Windows 7

cuDNN v7.1.3 Library for Windows 10

cuDNN v7.1.3 Runtime Library for Ubuntu16.04 (Deb)

cuDNN v7.1.3 Developer Library for Ubuntu16.04 (Deb)

cuDNN v7.1.3 Code Samples and User Guide for Ubuntu16.04 (Deb)

cuDNN v7.1.3 Runtime Library for Ubuntu16.04 & Power8 (Deb)

cuDNN v7.1.3Developer Library for Ubuntu16.04 & Power8 (Deb)

cuDNN v7.1.3 Code Samples and User Guide for Ubuntu16.04 & Power8 (Deb)

cuDNN v7.1.3 Runtime Library for Ubuntu14.04 (Deb)

cuDNN v7.1.3 Developer Library for Ubuntu14.04 (Deb)

cuDNN v7.1.3 Code Samples and User Guide for Ubuntu14.04 (Deb)

Macs are friends too!

:(

WHY?

Why?

- Existing libraries written in C/Fortran

Why?

- Existing libraries written in C/Fortran
  - Difficult to port over to Go

Why?

- Existing libraries written in C/Fortran
  - Difficult to port over to Go
- Our motivating example: CUDA libraries
  - `import "gorgonia.org/cu"`
  - `import "gorgonia.org/cu/blas"`
  - `import "gorgonia.org/cu/dnn"`

Some Definitions

C API

```
cudnnStatus_t cudnnCreate(cudnnHandle_t *handle)
```

Some Definitions

C API

```
cudnnStatus_t cudnnCreate(cudnnHandle_t *handle)
```

cgo Call

```
C.cudnnCreate(h *C.cudnnHandle_t) C.cudnnStatus_t
```

Some Definitions

C API

```
cudnnStatus_t cudnnCreate(cudnnHandle_t *handle)
```

cgo Call

```
C.cudnnCreate(h *C.cudnnHandle_t) C.cudnnStatus_t
```

Go API

```
func New(h *C.cudnnHandle_t) C.cudnnStatus_t
func New(h *C.cudnnHandle_t) error
func New() (*Handle, error)
func New() *Handle
```

Some Definitions

C API

```
cudnnStatus_t cudnnCreate(cudnnHandle_t *handle)
```

---

cgo Call

```
C.cudnnCreate(h *C.cudnnHandle_t) C.cudnnStatus_t
```

Go API

```
func New(h *C.cudnnHandle_t) C.cudnnStatus_t
func New(h *C.cudnnHandle_t) error
func New() (*Handle, error)
func New() *Handle
```

Why Write an API Library?

Why Write an API Library?

- A library that exposes a Go API reduces cognitive overhead.

Why Write an API Library?

- A library that exposes a Go API reduces cognitive overhead.
  - You don't have to know C anymore.

Why Write an API Library?

- A library that exposes a Go API reduces cognitive overhead.
  - You don't have to know C anymore.
- Abstract away bookkeeping activities.

Why Write an API Library?

- A library that exposes a Go API reduces cognitive overhead.
  - You don't have to know C anymore.
- Abstract away bookkeeping activities.
- More user friendly.

Why Write an API Library?

- A library that exposes a Go API reduces cognitive overhead.
  - You don't have to know C anymore.
- Abstract away bookkeeping activities.
- More user friendly.
- Provide performance improvements for common use cases.

Why Write an API Library?

In short, provide a great user experience.

This Talk

- Partly about strategies to write bindings.
- Partly about API design.

THE PROCESS: MANUAL WRITING

The Process: Transform from A to B

cgo Call

```
C.cudnnCreate(h *C.cudnnHandle_t) C.cudnnStatus_t
```

The Process: Transform from A to B

cgo Call

```
C.cudnnCreate(h *C.cudnnHandle_t) C.cudnnStatus_t
```

Wrap in one of these

Go API

```
func New(h *C.cudnnHandle_t) C.cudnnStatus_t
func New(h *C.cudnnHandle_t) error
func New() (*Handle, error)
func New() *Handle
```

Simplest Example

```
cudnnStatus_t cudnnCreate(cudnnHandle_t *handle)
```



```
func New(handle *C.cudnnHandle_t) C.cudnnStatus_t
```

Simplest Example

```
func New(handle *C.cudnnHandle_t) C.cudnnStatus_t {
   return C.cudnnCreate(handle)
}
```

Simplest Example

```
func New(handle *C.cudnnHandle_t) C.cudnnStatus_t {
    return C.cudnnCreate(handle)
}
```

The Process

```
type Handle struct{
    internal C.cudnnHandle_t
}

func New() *Handle {
    var h C.cudnnHandle_t
    if st := cudnnCreate(&h); st != C.CUDNN_STATUS_SUCCESS {
        panic(st)
    }
    return &Handle{h}
}
```

The Process

```
type Handle C.cudnnHandle_t

func New() Handle {
    var h C.cudnnHandle_t
    if st := cudnnCreate(&h); st != C.CUDNN_STATUS_SUCCESS {
        panic(st)
    }
    return Handle(h)
}
```

The Process: Go has Methods

- Methods make for nice APIs.

The Process: Go has Methods

- Methods make for nice APIs.
- Methods allow for interfaces to be implemented.

Manual Writing

- Forces one to actually think about the API.

Manual Writing

- Forces one to actually think about the API.
  - Result: nicer API based on use case

Manual Writing

- Forces one to actually think about the API.
- Requires one to be deeply familiar with the library.

Good Examples

- github.com/rakyll/portmidi
- github.com/mattn/go-gtk

Manual Writing: Downsides

- A lot of time and effort required.

Manual Writing: Downsides

- A lot of time and effort required.
- Not consistently replicable.

Manual Writing: Downsides

- A lot of time and effort required.
- Not consistently replicable.
- Coverage issues.

Manual Writing: Downsides

- A lot of time and effort required.
- Not consistently replicable.
- Coverage issues.
- Inability to keep up with new versions.


Wait a minute...
We are programmers!
Follow @chewxy on Twitter

AUTOMATIC GENERATION

Automatic Generation

- Repeatable results.

Automatic Generation

- Repeatable results.
- Generate multiple API levels.

Automatic Generation

Automatic generation steps:

1. Parse the C-header.
2. Generate the Go source code.

Good Examples

- github.com/go-gl/glfw
- github.com/go-gl/gl/vX.X-core/gl
  - Uses GLOW: github.com/go-gl/glow

- github.com/martine/gocairo
  - Uses rsc.io/c2go

Immediately Obvious Solution

github.com/xlab/c-for-go

The Devil Is In the Details

- Enums
- Types
- Functions
- Procedures

The Devil Is In the Details

- Enums → Type + Const
- Types
- Functions
- Procedures

The Devil Is In the Details

- Enums → Type + Const
- Types → Type + Getter functions
- Functions
- Procedures

The Devil Is In the Details

- Enums → Type + Const
- Types → Type + Getter functions
- Functions → Functions/Methods
- Procedures

The Devil Is In the Details

- Enums → Type + Const
- Types → Type + Getter functions
- Functions → Functions/Methods.
- Procedures → Functions/Methods

Note from xlab

*"The **resulting bindings are as low-level as C code**, i.e. it would require knowledge of memory management to carefully use the resulting code, however no more C code is needed to make things done. ... Also **usually a high-level wrapper is created by hand**"*

Solution: Manual Annotation

Annotate the things from the header – what should be a function, what should be a method, etc.

A Balance

Time spent manually annotating functions vs time spent generating code.

Solution: Multi-Stage Generation

1. Generate annotations that can be deduced by rules.
2. Manually check/edit select annotations.
3. Generate library.

CUDA Libraries

- Generated with:
  - github.com/cznic/cc
  - github.com/gorgonia/bindgen
  - go/token
  - go/parser
  - text/template
  - Specific programs to generate each library:
    - gorgonia.org/cu/cmd/genlib
    - gorgonia.org/cu/cmd/gencublas
    - gorgonia.org/cu/cmd/gencudnn

TACTICAL ISSUES WITH AUTOMATIC GENERATION

Issues with Automatic Generation

- Unclear type

Unclear Types

What is the type of `labels`?

```
cudnnStatus_t cudnnGetCTCLossWorkspaceSize(
    cudnnHandle_t                      handle,
    const   cudnnTensorDescriptor_t    probsDesc,
    const   cudnnTensorDescriptor_t    gradientsDesc,
    const   int                       *labels,
    const   int                       *labelLengths,
    const   int                       *inputLengths,
    cudnnCTCLossAlgo_t                 algo,
    const   cudnnCTCLossDescriptor_t   ctcLossDesc,
    size_t                            *sizeInBytes)
```

Unclear Types

What is the type of `labels`?

```
func (ctx *Handle) CTCLossWorkspaceSize(probDesc, gradientsDesc Tensor,
                                        labels ???,
                                        labelLengths ???,
                                        inputLengths ???,
                                        algo CTCLossAlgo,
                                        ctcLossDesc Tensor) (size uintptr)
```

Unclear Types

What is the type of `labels`?

```
cudnnStatus_t cudnnGetCTCLossWorkspaceSize(
    cudnnHandle_t                      handle,
    const   cudnnTensorDescriptor_t    probsDesc,
    const   cudnnTensorDescriptor_t    gradientsDesc,
    const   int                       *labels,
    const   int                       *labelLengths,
    const   int                       *inputLengths,
    cudnnCTCLossAlgo_t                 algo,
    const   cudnnCTCLossDescriptor_t   ctcLossDesc,
    size_t                            *sizeInBytes)
```

Unclear Types

Compare: What is the type of u, v?

```
cudnnStatus_t cudnnGetConvolution2dDescriptor(
    const cudnnConvolutionDescriptor_t  convDesc,
    int                                 *pad_h,
    int                                 *pad_w,
    int                                 *u,
    int                                 *v,
    int                                 *dilation_h,
    int                                 *dilation_w,
    cudnnConvolutionMode_t              *mode,
    cudnnDataType_t                     *computeType)
```

Unclear Types

What is the type of `labels`?

```
cudnnStatus_t cudnnGetCTCLossWorkspaceSize(
    cudnnHandle_t                      handle,
    const   cudnnTensorDescriptor_t    probsDesc,
    const   cudnnTensorDescriptor_t    gradientsDesc,
    const   int                       *labels,
    const   int                       *labelLengths,
    const   int                       *inputLengths,
    cudnnCTCLossAlgo_t                 algo,
    const   cudnnCTCLossDescriptor_t   ctcLossDesc,
    size_t                            *sizeInBytes)
```

Unclear Types

What is the type of `labels`?

```
cudnnStatus_t cudnnGetCTCLossWorkspaceSize(
    cudnnHandle_t                     handle,
    const   cudnnTensorDescriptor_t   probsDesc,
    const   cudnnTensorDescriptor_t   gradientsDesc,
    const   size_t                    labelsSize,
    const   int                       labels[],
    const   int                       labelLengths[],
    const   int                       inputLengths[],
    cudnnCTCLossAlgo_t                algo,
    const   cudnnCTCLossDescriptor_t  ctcLossDesc,
    size_t                           *sizeInBytes)
```

Unclear Types

What is the type of `labels`?

```
cudnnStatus_t cudnnGetCTCLossWorkspaceSize(
    cudnnHandle_t                       handle,
    const   cudnnTensorDescriptor_t     probsDesc,
    const   cudnnTensorDescriptor_t     gradientsDesc,
    const   size_t                      labelsSize,
    const   int                         labels[],
    const   int                         labelLengths[],
    const   int                         inputLengths[],
    cudnnCTCLossAlgo_t                  algo,
    const   cudnnCTCLossDescriptor_t    ctcLossDesc,
    size_t                             *sizeInBytes)
```

To a C compiler, there is no difference between `int*` and `int[]`

Issues with Automatic Generation

- Unclear type
- Unclear use cases

Unclear Use Case

What is the use case of `algo`?

```
cudnnStatus_t cudnnGetCTCLossWorkspaceSize(
    cudnnHandle_t                     handle,
    const   cudnnTensorDescriptor_t   probsDesc,
    const   cudnnTensorDescriptor_t   gradientsDesc,
    const   int                      *labels,
    const   int                      *labelLengths,
    const   int                      *inputLengths,
    cudnnCTCLossAlgo_t                algo,
    const   cudnnCTCLossDescriptor_t  ctcLossDesc,
    size_t                           *sizeInBytes)
```

Unclear Use Case

What is the use case of `algo`?

```
cudnnStatus_t cudnnGetCTCLossWorkspaceSize(
    cudnnHandle_t                      handle,
    const   cudnnTensorDescriptor_t    probsDesc,
    const   cudnnTensorDescriptor_t    gradientsDesc,
    const   int                       *labels,
    const   int                       *labelLengths,
    const   int                       *inputLengths,
    const   cudnnCTCLossAlgo_t         algo,
    const   cudnnCTCLossDescriptor_t   ctcLossDesc,
    size_t                            *sizeInBytes)
```

Hybrid Solution

1. Parse the docs. Add them as constants.
2. Manually annotate specific parameters.
3. Automatic generation as an aid to manual generation.

VARIOUS CONSIDERATIONS

Various Considerations

- Performance
- Contexts and multi-level API
- Error detection
- Reporting

Various Considerations

- Performance
- Contexts and multi-level API
- Error detection
- Reporting

Performance Tip: Batching

```
func NewConvolution(mathType MathType, groupCount int, padding, filterStride, dilation []int, convolutionMode ConvolutionMode, datatype
DataType) (*Convolution, error) {
	var internal C.cudnnConvolutionDescriptor_t
	if err := result(C.cudnnCreateConvolutionDescriptor(&internal)); err != nil { ... }
	if err := result(C.cudnnSetConvolutionMathType(internal, mathType.C())); err != nil { ... }
	if err := result(C.cudnnSetConvolutionGroupCount(internal, C.int(groupCount))); err != nil { ... }
	switch len(padding) {
	case 0, 1:
		return nil, errors.Errorf("Only 2+ dims are allowed")
	case 2:
		// snipped: destructuring of padH, padW, u, v, etc
		if err := result(C.cudnnSetConvolution2dDescriptor(internal, C.int(padH), C.int(padW), C.int(u), C.int(v), C.int(dilationH),
C.int(dilationW), convolutionMode.C(), datatype.C())); err != nil {
			return nil, err
		}
	default:
		// snipped: converting padding, filterStride, dilation to a pointer
		if err := result(C.cudnnSetConvolutionNdDescriptor(internal, C.int(len(padding)), padA, strideA, dilationA,
convolutionMode.C(), datatype.C())); err != nil {
			return nil, err
		}
	}
	retVal := &Convolution{...}
	runtime.SetFinalizer(retVal, destroyConvolution)
	return retVal, nil
}
```

Performance Tip: Batching

```
func NewConvolution(mathType MathType, groupCount int, padding, filterStride, dilation []int, convolutionMode ConvolutionMode, datatype
DataType) (*Convolution, error) {
	var internal C.cudnnConvolutionDescriptor_t
	if err := result(C.cudnnCreateConvolutionDescriptor(&internal)); err != nil { ... }
	if err := result(C.cudnnSetConvolutionMathType(internal, mathType.C())); err != nil { ... }
	if err := result(C.cudnnSetConvolutionGroupCount(internal, C.int(groupCount))); err != nil { ... }
	switch len(padding) {
	case 0, 1:
		return nil, errors.Errorf("Only 2+ dims are allowed")
	case 2:
		// snipped: destructuring of padH, padW, u, v, etc
		if err := result(C.cudnnSetConvolution2dDescriptor(internal, C.int(padH), C.int(padW), C.int(u), C.int(v), C.int(dilationH),
C.int(dilationW), convolutionMode.C(), datatype.C())); err != nil {
			return nil, err
		}
	default:
		// snipped: converting padding, filterStride, dilation to a pointer
		if err := result(C.cudnnSetConvolutionNdDescriptor(internal, C.int(len(padding)), padA, strideA, dilationA,
convolutionMode.C(), datatype.C())); err != nil {
			return nil, err
		}
	}
	retVal := &Convolution{...}
	runtime.SetFinalizer(retVal, destroyConvolution)
	return retVal, nil
}
```

Performance Tip: Batching

```
cudnnStatus_t gocudnnNewConvolution(cudnnConvolutionDescriptor_t *retVal, cudnnMathType_t mathType, const int groupCount,
      const int size, const int* padding, const int* filterStrides, const int* dilation, cudnnConvolutionMode_t convolutionMode,
cudnnDataType_t dataType) {
      cudnnStatus_t status ;
      status = cudnnCreateConvolutionDescriptor(retVal);
      if (status != CUDNN_STATUS_SUCCESS) { return status;  }

      status = cudnnSetConvolutionMathType(*retVal, mathType);
      if (status != CUDNN_STATUS_SUCCESS) { return status;  }

      status = cudnnSetConvolutionGroupCount(*retVal, groupCount);
      if (status != CUDNN_STATUS_SUCCESS) { return status;  }

      switch (size) {
      case 0, 1 :
            return CUDNN_STATUS_BAD_PARAM;
      case 2:
            // snipped : unpacking the params
            status = cudnnSetConvolution2dDescriptor(*retVal, padH, padW, u, v, dilationH, dilationW, convolutionMode, dataType);
            break;
      default:
            status = cudnnSetConvolutionNdDescriptor(*retVal, paddingSize, padding, filterStrides, dilation, convolutionMode,
dataType);
            break;
      }
      return status;
}
```

Performance Tip: Batching

```
func NewConvolution(mathType MathType, groupCount int, padding, filterStride, dilation []int, convolutionMode ConvolutionMode, datatype
DataType) (*Convolution, error) {
	var internal C.cudnnConvolutionDescriptor_t
	if err := result(C.cudnnCreateConvolutionDescriptor(&internal)); err != nil { ... }
	if err := result(C.cudnnSetConvolutionMathType(internal, mathType.C())); err != nil { ... }
	if err := result(C.cudnnSetConvolutionGroupCount(internal, C.int(groupCount))); err != nil { ... }
	switch len(padding) {
	case 0, 1:
		return nil, errors.Errorf("Only 2+ dims are allowed")
	case 2:
		// snipped: destructuring of padH, padW, u, v, etc
		if err := result(C.cudnnSetConvolution2dDescriptor(internal, C.int(padH), C.int(padW), C.int(u), C.int(v), C.int(dilationH),
C.int(dilationW), convolutionMode.C(), datatype.C())); err != nil {
			return nil, err
		}
	default:
		// snipped: converting padding, filterStride, dilation to a pointer
		if err := result(C.cudnnSetConvolutionNdDescriptor(internal, C.int(len(padding)), padA, strideA, dilationA,
convolutionMode.C(), datatype.C())); err != nil {
			return nil, err
		}
	}
	retVal := &Convolution{...}
	runtime.SetFinalizer(retVal, destroyConvolution)
	return retVal, nil
}
```

Performance Tip: Batching

```
func NewConvolution(mathType MathType, groupCount int, padding, filterStride, dilation []int, convolutionMode ConvolutionMode, datatype
DataType) (*Convolution, error) {
	// elided: checks

	var internal C.cudnnConvolutionDescriptor_t
	padA := (*C.int)(unsafe.Pointer(&padding[0]))
	strideA := (*C.int)(unsafe.Pointer(&filterStride[0]))
	dilationA := (*C.int)(unsafe.Pointer(&dilation[0]))
	if err = result(C.gocudnnNewConvolution(&internal, mathType.C(), C.int(groupCount), C.int(len(padding)), padA, filterStrideA,
dilationA, convolutionMode.C(), datatype.C())); err != nil {
		return nil, err
	}

	retVal := &Convolution{...}
	runtime.SetFinalizer(retVal, destroyConvolution)
	return retVal, nil
}
```

Performance: Batching Tips

- `astyle` is your friend.
-
```
astyle --style=google  \
       --lineend=linux \
       --indent=tab    \
       --indent-switches     \
       --align-pointer=type  \
       --align-reference=name \
       --delete-empty-lines
```
- http://astyle.sourceforge.net/

Performance Tip: Cache Fields

Opaque types:

```
struct cudnnTensorStruct* cudnnTensorDescriptor_t
```

Go version:

```
type TensorDescriptor struct {
   internal C.cudnnTensorDescriptor_t

   format TensorFormat
   dataType DataType
   shape []int
   strides []int
}
```

Performance Tip: Cache Fields

Opaque types:

```
struct cudnnTensorStruct* cudnnTensorDescriptor_t
```

Go version:

```
type TensorDescriptor struct {
   internal C.cudnnTensorDescriptor_t

   format TensorFormat
   dataType DataType
   shape []int
   strides []int
}
```

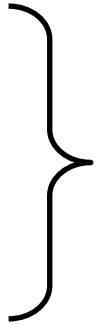
Where did these come from?

Performance Tip: Cache Fields

```
cudnnStatus_t cudnnSetTensor4dDescriptor(
                cudnnTensorDescriptor_t         tensorDesc,
                cudnnTensorFormat_t             format,
                cudnnDataType_t                 dataType,
                int                             n,
                int                             c,
                int                             h,
                int                             w);
```

Performance Tip: Cache Fields

```
cudnnStatus_t cudnnSetTensor4dDescriptor(
                    cudnnTensorDescriptor_t         tensorDesc,
                    cudnnTensorFormat_t             format,
                    cudnnDataType_t                 dataType,
                    int                             n,
                    int                             c,
                    int                             h,
                    int                             w);
```



Here

Performance Tip: Cache Fields

```
cudnnStatus_t cudnnSetTensor4dDescriptor(
                cudnnTensorDescriptor_t            tensorDesc,
                cudnnTensorFormat_t                format,
                cudnnDataType_t                    dataType,
                int                                n,
                int                                c,
                int                                h,
                int                                w);

cudnnStatus_t cudnnGetTensor4dDescriptor(
                const cudnnTensorDescriptor_t      tensorDesc,
                cudnnDataType_t                   *dataType,
                int                               *n,
                int                               *c,
                int                               *h,
                int                               *w,
                int                               *nStride,
                int                               *cStride,
                int                               *hStride,
                int                               *wStride );
```

Performance Tip: Cache Fields

```
cudnnStatus_t cudnnSetTensor4dDescriptor(
                    cudnnTensorDescriptor_t             tensorDesc,
                    cudnnTensorFormat_t                 format,
                    cudnnDataType_t                     dataType,
                    int                                 n,
                    int                                 c,
                    int                                 h,
                    int                                 w);

cudnnStatus_t ~~cudnnGetTensor4dDescriptor~~(
                    const cudnnTensorDescriptor_t       tensorDesc,
                    cudnnDataType_t                    *dataType,
                    int                                *n,
                    int                                *c,
                    int                                *h,
                    int                                *w,
                    int                                *nStride,
                    int                                *cStride,
                    int                                *hStride,
                    int                                *wStride );
```

Guiding Principle

- Make your data structures useful.

Guiding Principle

- Make your data structures useful.
- Start from the use cases.

Guiding Principle

- Make your data structures useful.
- Start from the use cases.
- Eliminate binding calls where unnecessary.

Guiding Principle

- Make your data structures useful.
- Start from the use cases.
- Eliminate binding calls where unnecessary.
- Minimize errors – check them in Go before passing off to cgo.

Various Considerations

- Performance
- Contexts and multi-level API
- Error detection
- Reporting

Contextual Use

Most ~~graphics~~ heterogenous memory related libraries require a thread-lock.

Contextual Use: Generate Multi-Level API

- One that handles the common use case.
- One low level API, leaving it to the user to manage themselves.

CUDA Contexts

```
// CUContext is a CUDA context
type CUContext uintptr
```

CUDA Contexts

```
// CUContext is a CUDA context
type CUContext uintptr

// Ctx is a standalone CUDA Context that is threadlocked.
type Ctx struct{
    CUContext
    // contains filtered or unexported fields
}
```

CUDA Contexts

```
// CUContext is a CUDA context
type CUContext uintptr

// Ctx is a standalone CUDA Context that is threadlocked.
type Ctx struct{
    CUContext
    // contains filtered or unexported fields
}

// BatchedContext is a CUDA context where the CUDA calls are batched up.
type BatchedContext struct {
    Context
    Device
    // contains filtered or unexported fields
}
```

CUDA Context: Synchronize()

```
func Synchronize() (err error) {
   return result(C.cuCtxSynchronize())
}
```

Use in a main-threadlocked function

```
   ...
   handleErr(ctx.Lock())
   handleErr(cu.SetCurrentContext(ctx))
   handleErr(cu.Synchronize())
   handleErr(ctx.Unlock())
```

CUDA Context: Synchronize()

```
// Ctx is a standalone CUDA Context that is threadlocked.
type Ctx struct {
    work    chan (func() error)
    ...
}

func (ctx *Ctx) Do(fn func() error) error {
    ctx.work <- fn
    return <-ctx.errChan
}

func (ctx *Ctx) Synchronize() {
    f := func() error {
        return result(C.cuCtxSynchronize())
    }
    ctx.err = ctx.Do(f)
}
```

Various Considerations

- Performance
- Contexts and multi-level API
- Error detection
- Reporting

Error Detection

- Leverage existing Go source code analysis tools:
  - goimports
  - govet
  - go/token, go/parser, go/ast

Error Detection

- Potential Nil-checks
  - Alias analysis is undecidable[0].
  - Doesn't mean you can't try.
  - Easy to check when functions are small.

[0] G. Ramalingam (1994): The Undecidability of Aliasing.

Various Considerations

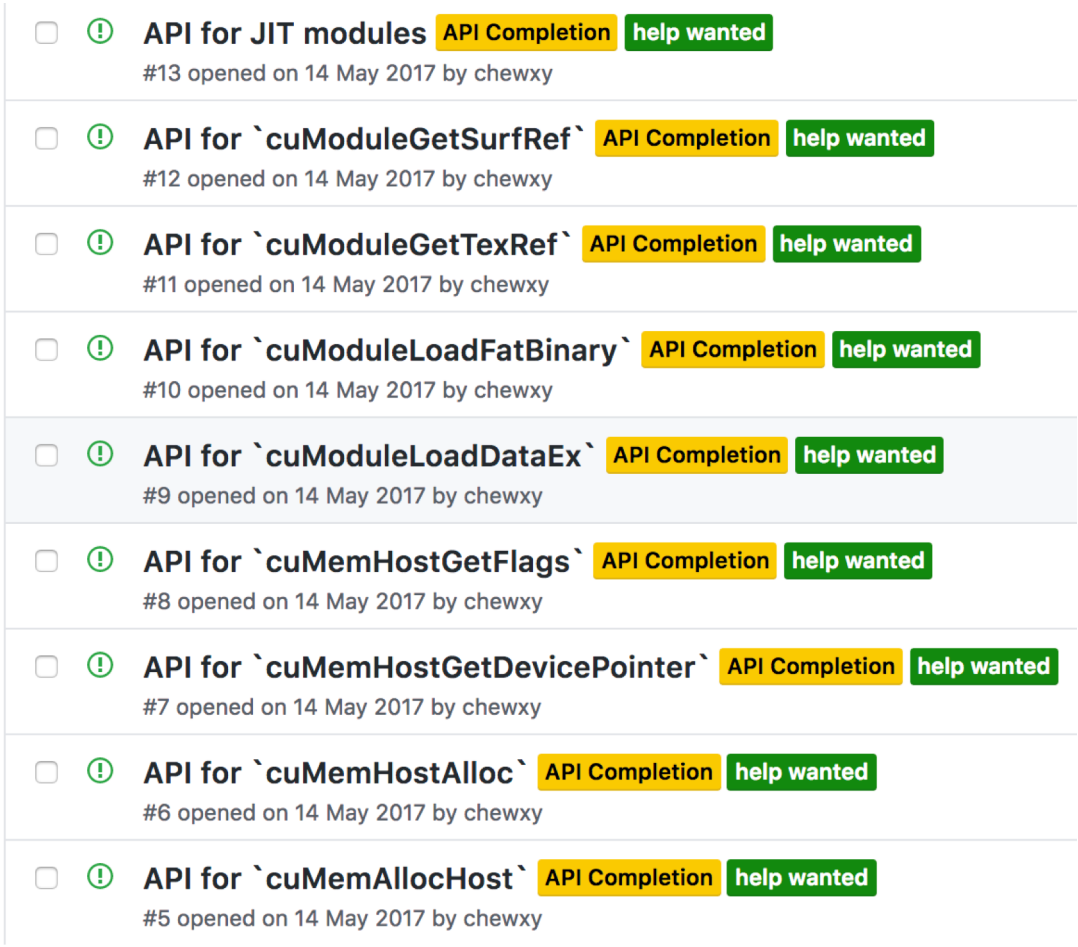
- Performance
- Contexts and multi-level API
- Documentation
- Error detection
- Reporting

Reporting

- In multistage generation programs – always report back what is not done yet.

Reporting

- In multistage generation programs – always report back what is not done yet.



FINALLY...

The source for the demo is online:

https://github.com/chewxy/gopherconsg2018

In This Talk...

- Introduced the *whys* of writing an API library for Go bindings.
- Showed *how* to think about writing an API library.
- Using `gorgonia.org/cu/dnn` and `gorgonia.org/cu` as a concrete examples.

The Ask

The Ask

- Through use and feedback, APIs can be improved.

The Ask

- Through use and feedback, APIs can be improved.
- Please use these, and provide feedback:
  - `gorgonia.org/gorgonia`
  - `gorgonia.org/tensor`
  - `gorgonia.org/cu/...`

The Ask

- Through use and feedback, APIs can be improved.
- Please use these, and provide feedback:
  - `gorgonia.org/gorgonia`
  - `gorgonia.org/tensor`
  - `gorgonia.org/cu/...`
- Help improve the generation programs:
  - `gorgonia.org/cu/cmd/genlib`
  - `gorgonia.org/cu/cmd/gencudnn`
  - `gorgonia.org/cu/cmd/gencublas`

Follow me on Twitter: @chewxy
Email me: chewxy@gmail.com

THANK YOU GOPHERCON SG!