# Speed Up Your Language

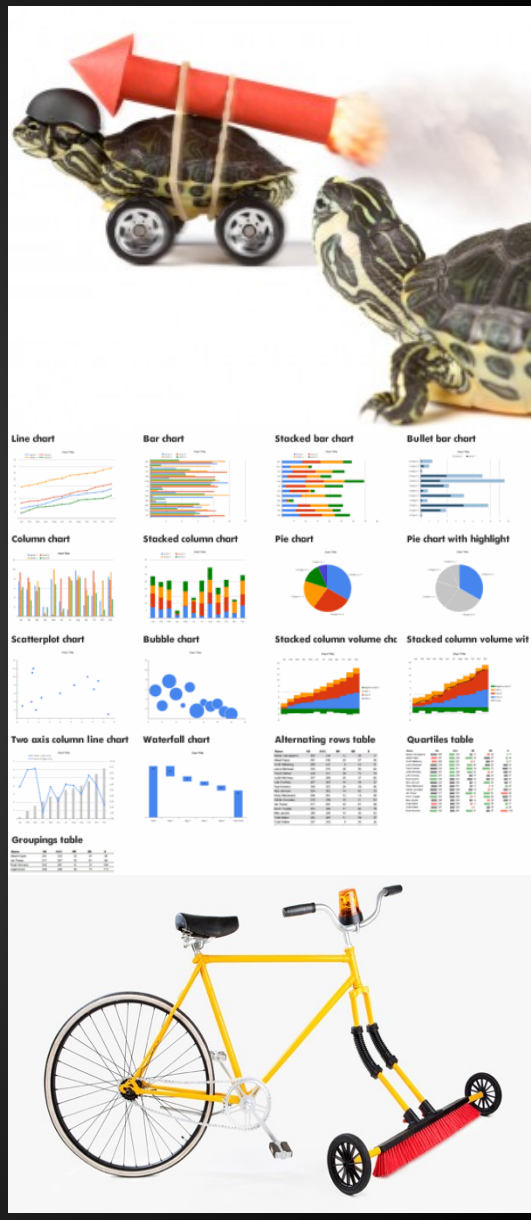
with **CUDA**

# Agenda

- Motivation
- C++
- Name manglings
- PyCUDA
- managedCUDA
- CUDAfy
- JCUDA
- Etc.
- Q

# Motivation

- Old projects
- Research calculations
- Prototypes
- Etc.



# C++

Nice to have NVidia samples :)
MatrixMul and MatrixMulCUBLAS

Goal:
Check ability to use kernels and libraries

# Name Mangling

## Function aliasing in some kind

Microsoft mangling scheme for simple cases

| | | |
|---|---|---|
| int _cdecl f(int); | _name | _f |
| int _stdcall g(int); | _name@X | _g@4 |
| int _fastcall h(int); | @name@X | @h@4 |

GNU GCC 3.x complex example

```
namespace wikipedia
{
  class article
  {
  public:
    std::string format (void);
  }
}
```

_ZN9wikipedia7article6formatEv

_Z + N + <length, id>... + E + type id

# Name Mangling

Spare table from Wikipedia

| Compiler | void h(int) | void h(int, char) | void h(void) |
|---|---|---|---|
| Intel C++ 8.0 for Linux | _Z1hi | _Z1hic | _Z1hv |
| HP aC++ A.05.55 IA-64 | _Z1hi | _Z1hic | _Z1hv |
| IAR EWARM C++ 5.4 ARM | _Z1hi | _Z1hic | _Z1hv |
| GCC 3.*x* and 4.*x* | _Z1hi | _Z1hic | _Z1hv |
| GCC 2.9*x* | h__Fi | h__Fic | h__Fv |
| HP aC++ A.03.45 PA-RISC | h__Fi | h__Fic | h__Fv |
| Microsoft Visual C++ v6-v10 (mangling details) | ?h@@YAXH@Z | ?h@@YAXHD@Z | ?h@@YAXXZ |
| Digital Mars C++ | ?h@@YAXH@Z | ?h@@YAXHD@Z | ?h@@YAXXZ |
| Borland C++ v3.1 | @h$qi | @h$qizc | @h$qv |
| OpenVMS C++ V6.5 (ARM mode) | H__XI | H__XIC | H__XV |
| OpenVMS C++ V6.5 (ANSI mode) | CXX$__7H__FI0ARG51T | CXX$__7H__FIC26CDH77 | CXX$__7H__FV2CB06E8 |
| OpenVMS C++ X7.1 IA-64 | CXX$_Z1HI2DSQ26A | CXX$_Z1HIC2NP3LI4 | CXX$_Z1HV0BCA19V |
| SunPro CC | __1cBh6Fi_v_ | __1cBh6Fic_v_ | __1cBh6F_v_ |
| Tru64 C++ V6.5 (ARM mode) | h__Xi | h__Xic | h__Xv |
| Tru64 C++ V6.5 (ANSI mode) | __7h__Fi | __7h__Fic | __7h__Fv |
| Watcom C++ 10.6 | W?h$n(i)v | W?h$n(ia)v | W?h$n()v |

# Name Mangling

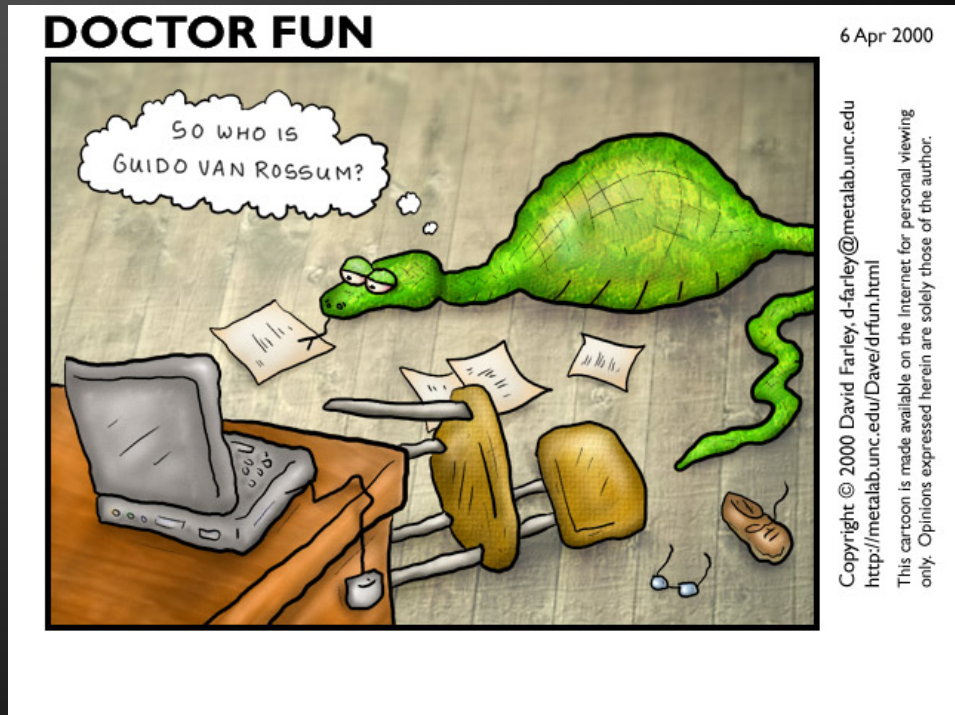
In case we don’t need mangling

```
#ifdef __cplusplus
extern "C" {
#endif

   int cudaKernelCall(int *, size_t);

#ifdef __cplusplus
}
#endif
```

# PyCUDA

- Installation
  - Setuptools, Pytools, Numpy, PyCUDA
- Configuration
  - Path += cl.exe, CUDA toolkit

Notes: only Python 2.x



# PyCUDA

**First workability check**

**Import all we need**

```
import pycuda.driver
import pycuda.autoinit
from pycuda.compiler import SourceModule
import pycuda.gpuarray
import numpy
```

# PyCUDA

**Second workability check**

**CUDA kernel compilation**

```
module = SourceModule("""
        __global__ void
        matrixMulCUDA
        (float *C, float *A, float *B, int wA, int wB)
        {...} """)

func = func_mod.get_function('matrixMulCUDA')
```

# PyCUDA


## CUDA Events

```
start = driver .Event()
start.record()
msecs = start.time_till(end)
```

## CUDA Kernel's Launch

```
func = func_mod.get_function('matrixMulCUDA')
func(<matrixMulCUDA args>,
        block=blockDim,grid=gridDim)
```

# PyCUDA

**C++ templates**

**vs**

**python jinja2 generator**

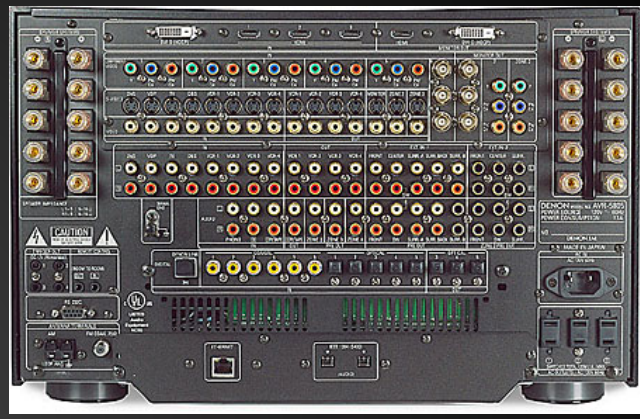

```
template = Template(kernelSource)
renderedKernelSource = template.render(
    blocks=(16, 32), realT="float")
module = SourceModule(renderedKernelSource)
```

# PyCUDA

## Summary

| | |
|---|---|
| CUDA C kernels | Yes |
| C++ templates | Yes |
| CUDA Libraries | NO |
| CUDA Streams | Yes |

# Managed CUDA



- Installation
  - Binaries
- Configuration
  - NVidia project with kernel files
  - NVidia project type - Utility
  - NVidia project CUDA output - *.ptx
  - NVidia project delete post build event
  - C# project add ref to the Binaries
  - C# project add *.ptx as embedded resources

# Managed CUDA

## Include namespaces

```
using ManagedCuda;
```

## Create Cuda Context

```
ctx = new CudaContext(CudaContext.GetMaxGflopsDeviceId());
```

## LoadAssembly

```
Stream stream = Assembly.GetExecutingAssembly().
                                GetManifestResourceStream(resourceName);
CudaKernel matMulKernel = ctx.LoadKernelPTX
                                (stream, "matrixMulCUDA32");
```

## Configure Kernel

```
matMulKernel.BlockDimensions = new dim3(32, 32);
matMulKernel.GridDimensions =
    new dim3(matrixWidth/32, matrixWidth/32);
```

# Managed CUDA

## Prepare Data Pointers

```
static CudaDeviceVariable<float> d_A;
d_A = h_A;
```

## Run Kernel

```
float cudaTime_ms = matMulKernel.Run(
    d_C.DevicePointer, d_A.DevicePointer, d_B.DevicePointer
    , matrixWidth, matrixWidth);
```

# Managed CUDA

## CudaBlas - native dll wrapper only

Create CudaBlasHandle

Call native library function

Copy results

# Managed CUDA

## Summary

| | |
|---|---|
| CUDA C kernels | Yes |
| C++ templates | Partially |
| CUDA Libraries | Yes |
| CUDA Streams | Yes |

# CUDAfy


- Installation
  - Binaries
- Configuration
  - Run CUDAfy GUI
  - Check all available computation hardware
  - In C# project add ref to the CUDAfy.NET dll

# CUDAfy


Check compiled module

```
CudafyModule module = CudafyModule.TryDeserialize();
```

And/Or Compile module

```
module = CudafyTranslator.Cudafy
    (ePlatform.Auto, gpu.GetArchitecture(), typeof(MatrixMultClass));
```

Load module

```
gpu.LoadModule(module);
```

Prepare data in the device memory

```
float[] d_A = gpu.CopyToDevice(h_A);
```

# CUDAfy


## C# Kernel

```
[Cudafy]
private static void MatMultKernel
 (GThread thread, float[] C, float[] A, float[] B, int width)
```

## GThread

blockDim, threadIdx, etc
warpSize, All, Any, etc
AllocateShared
SyncThreads

## Kernel launch

```
gpu.Launch(grid, block,
   ((Action<GThread, float[], float[], float[], int>)MatMultKernel)
    , d_C, d_A, d_B, h_matrixWidth);
```

# CUDAfy


Shared memory

```
float[,] As = thread.AllocateShared<float>
    ("As", BLOCK_SIZE, BLOCK_SIZE);
```

Get results

```
gpu.CopyFromDevice(d_C, 0, h_C, 0, h_C.Length);
```

Release all memory

```
gpu.FreeAll();
```

NOTE: Cudafy generate .cu and .ptx from your C# kernel

# CUDAfy

## Summary

| | |
|---|---|
| C# kernels | Yes |
| C++ templates | NO |
| CUDA Libraries | Partially |
| CUDA Streams | Yes |

# JCUDA

- Installation
  - Binaries
- Configuration
  - Jar files should be present in CLASSPATH
  - PATH += path to native libraries
  - PATH += path to javac and java

# JCUDA

Basic check

```
import jcuda.*;
import jcuda.runtime.*;
Pointer pointer = new Pointer();
JCuda.cudaMalloc(pointer, 4);
```

CUDA kernel

Simple .cu file with extern “C” kernel definition

Compilation

generate nvcc compilation command string

execute in the system using exec

Next steps use CUDA Driver API

direct bindings

# JCUDA

## Summary

| | |
|---|---|
| C++ kernels | Yes |
| C++ templates | Partially |
| CUDA Libraries | Yes |
| CUDA Streams | Yes |

# Etc.

Haskell Accelerate-cuda package

Matlab GPU CUDA Computing

IDL GPULib

# Q.