A. JAMES CLARK
SCHOOL OF ENGINEERING

Applying Graphics Processor Acceleration in a Software Defined Radio Prototyping Environment

GNU Radio with Graphics Processor Acceleration as a Standalone Package

Will Plishker, George F. Zaki,
Shuvra S. Bhattacharyya
University of Maryland

Charles Clancy, John Kuykendall
Laboratory for Telecommunication Sciences




UMIACS
University of Maryland Institute for Advanced Computer Studies

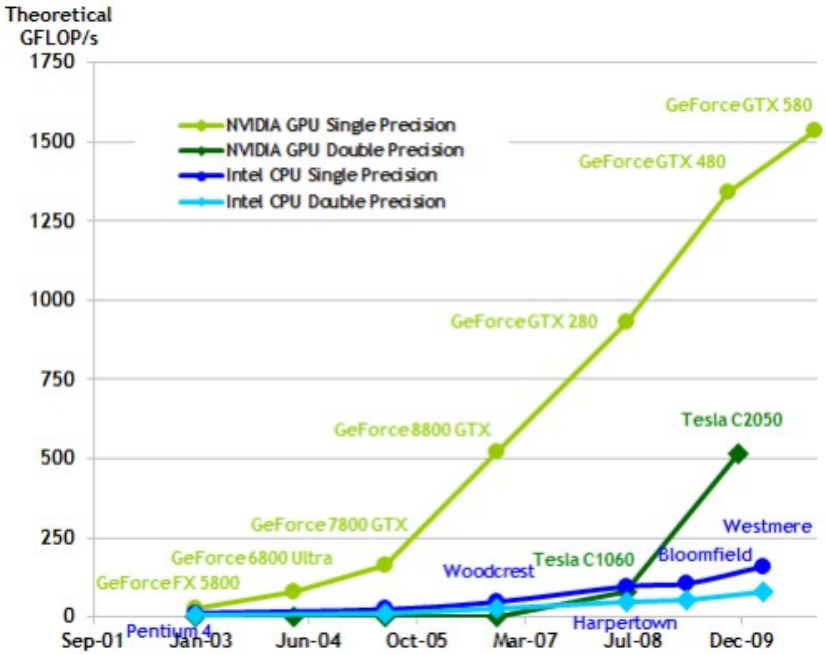
DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING

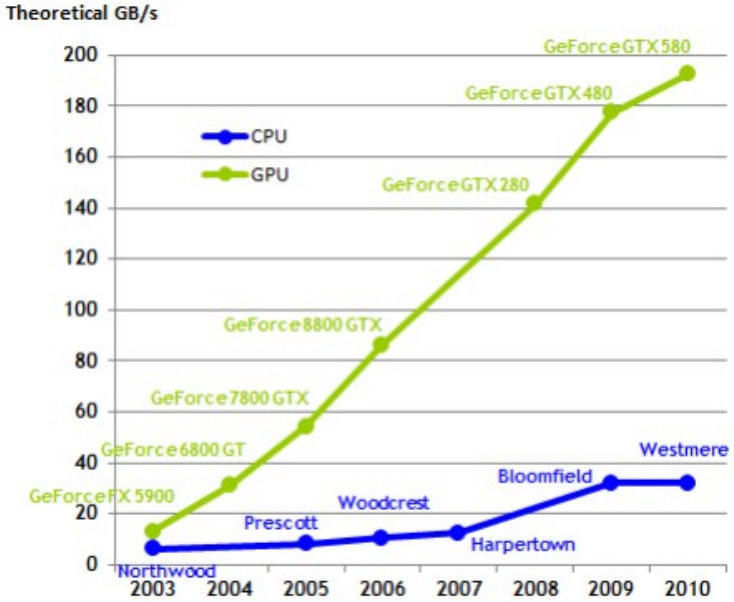
Outline

- Introduction
- Related Work
- GNU Radio Background
- Graphics Processor Integration with GNU Radio
- Evaluation
- Summary

Motivation

- High-Performance prototyping with GPUs is tempting…
  - *Potentially* fast design times
  - Flop/$
- …but hard.
  - Learn a new language
  - Expose or refactor parallelism





A brief history of GPU Programming

- OpenGL (1992)/DirectX(1995) + shader programming in assembly
- Cg, GLSL, HLSL (early 2000s)
- Sh, Brook (2004)
- CUDA (2006)
- OpenCL (2008)

Goals for a GPU Rapid Prototyping Tool

- Reusable
- Portable
- Transparent
- High-performance

Related Work

- GNU Radio Acceleration
  - Cell
  - FPGAs
  - Tilera
  - General Purpose Processor (GPP) Multicore
- CUDA Integration with interpreted languages
  - PyCUDA
  - MATLAB Multicore Toolbox

Related Work – PyCUDA

```
import pycuda.autoinit
import pycuda.driver as drv
import numpy

from pycuda.compiler import SourceModule
mod = SourceModule("""
__global__ void multiply_them(float *dest, float *a, float *b)
{
  const int i = threadIdx.x;
  dest[i] = a[i] * b[i];
}
""")

multiply_them = mod.get_function("multiply_them")

a = numpy.random.randn(400).astype(numpy.float32)
b = numpy.random.randn(400).astype(numpy.float32)

dest = numpy.zeros_like(a)
multiply_them(
        drv.Out(dest), drv.In(a), drv.In(b),
        block=(400,1,1), grid=(1,1))

print dest-a*b
```

Related Work – MATLAB

MATLAB 7.11.0 (R2010b)

File Edit Debug Parallel Desktop Window Help

Workspace

Select da...

Name	Class
A	parallel.gpu.GPUArray
b	parallel.gpu.GPUArray
f	double (complex)
f_gpu	parallel.gpu.GPUArray
x	double
x_gpu	parallel.gpu.GPUArray

Command Window

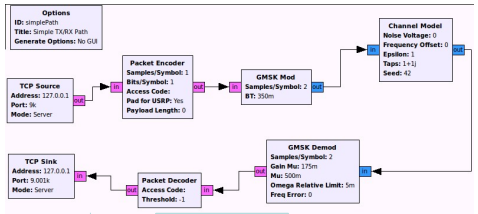
```
>>
>> % Create arrays that reside on the GPU
>> A = gpuArray(rand(1000, 1000));
>> b = gpuArray(rand(1000, 1));
>>
>> % Use GPU-enabled MATLAB functions
>> x_gpu = A \ b; % "\" is GPU-enabled
>>
>> f_gpu = fft(A);
>>
>> % Bring data back from GPU memory into MATLAB workspace
>> x = gather(x_gpu);
>> f = gather(f_gpu);
>>
```

Start

OVR

Background – GNU Radio Design Flow

GNU Radio Companion (GRC)

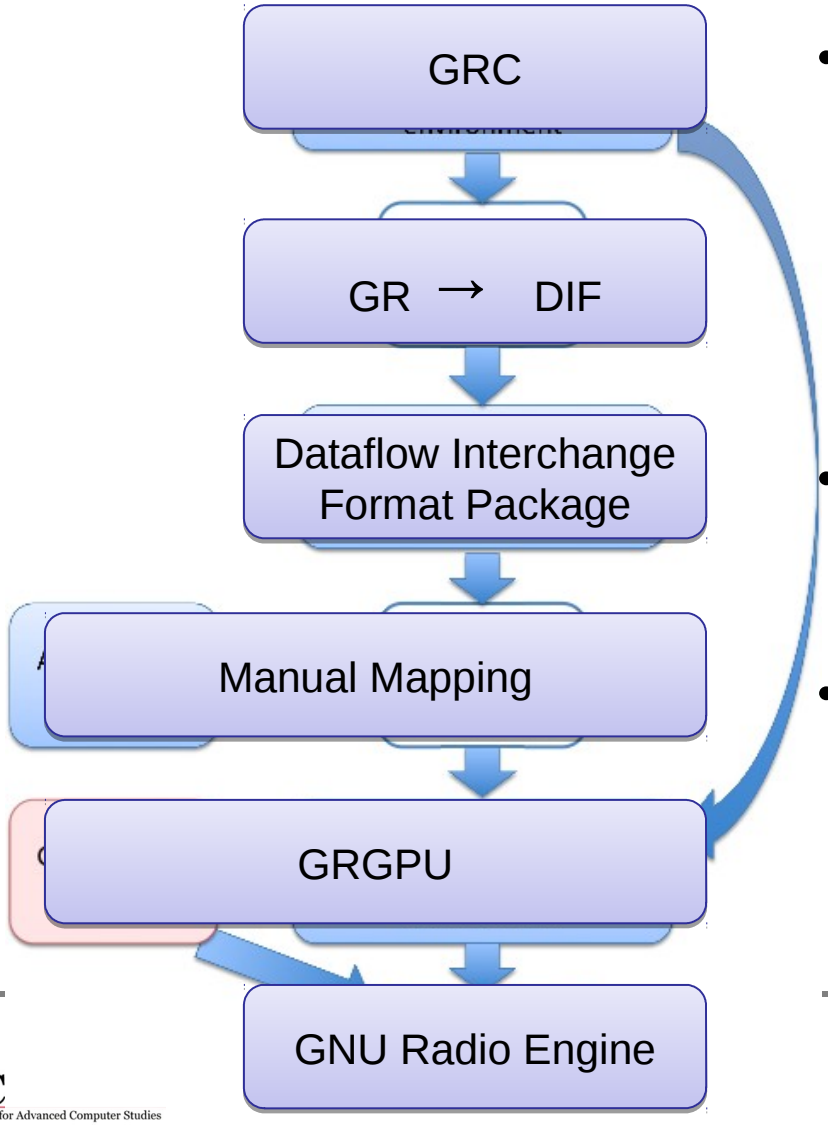


GNU Radio Python Flowgraph

```
class top(gr.top_block):
    def __init__(self):
        gr.top_block.__init__(self)
        ntaps = 256

        # Something vaguely like floating point ops
        self.flop = 2 * ntaps * options.npipelines

        src = gr.null_source(gr.sizeof_float)
        head = gr.head(gr.sizeof_float, int(options.nsamples))
        self.connect(src, head)

        for n in range(options.npipelines):
            self.connect(head, pipeline(options.nstages, ntaps))
```

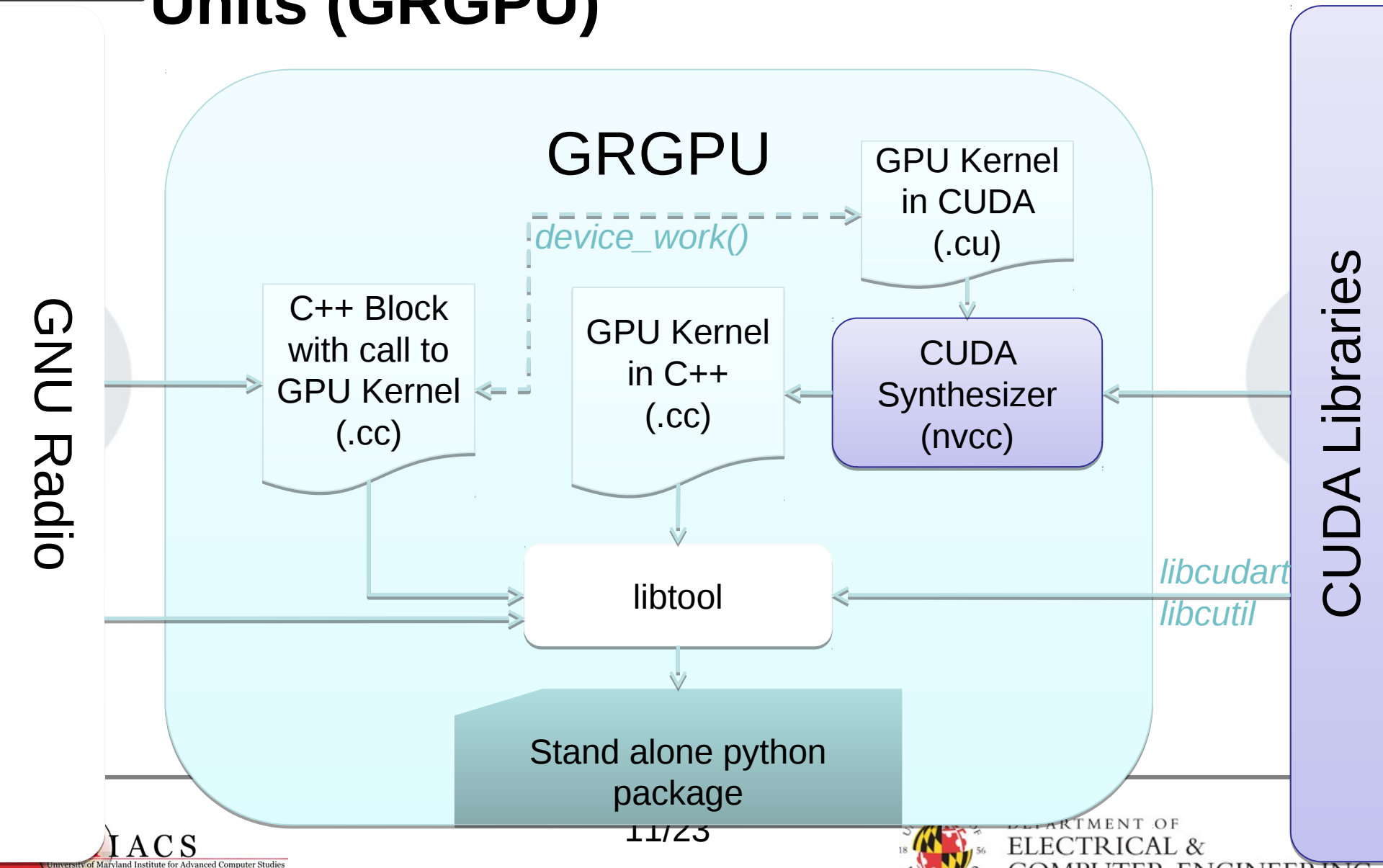
Universal Software Radio Peripheral (USRP)

Proposed Design Flow



- Start from an unambiguous application description based on the application area itself
  - Dataflow
- Use tools to optimize scheduling, assignment
- Generate an accelerated functional description

GNU Radio for Graphics Processing Units (GRGPU)



Build Process

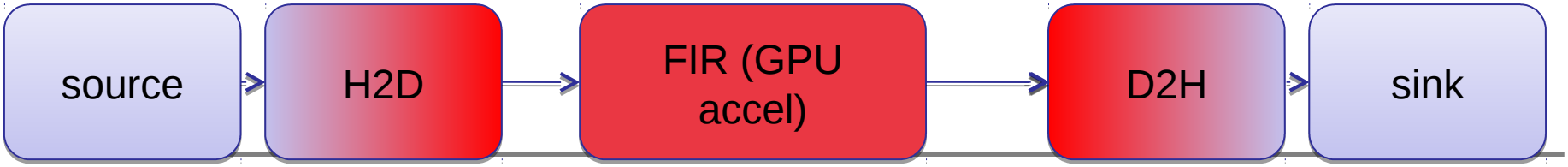
- make-cuda : .cu → .cc
  - nvcc as a software synthesizer
  - transform cuda code into target specific GPU code
  - wrap functions in extern “C”
- make
  - Resulting cc’s are just more inputs to existing build structure
  - link against CUDA runtime (cudart) and CUDA utils (cutil)

Using GRGPU

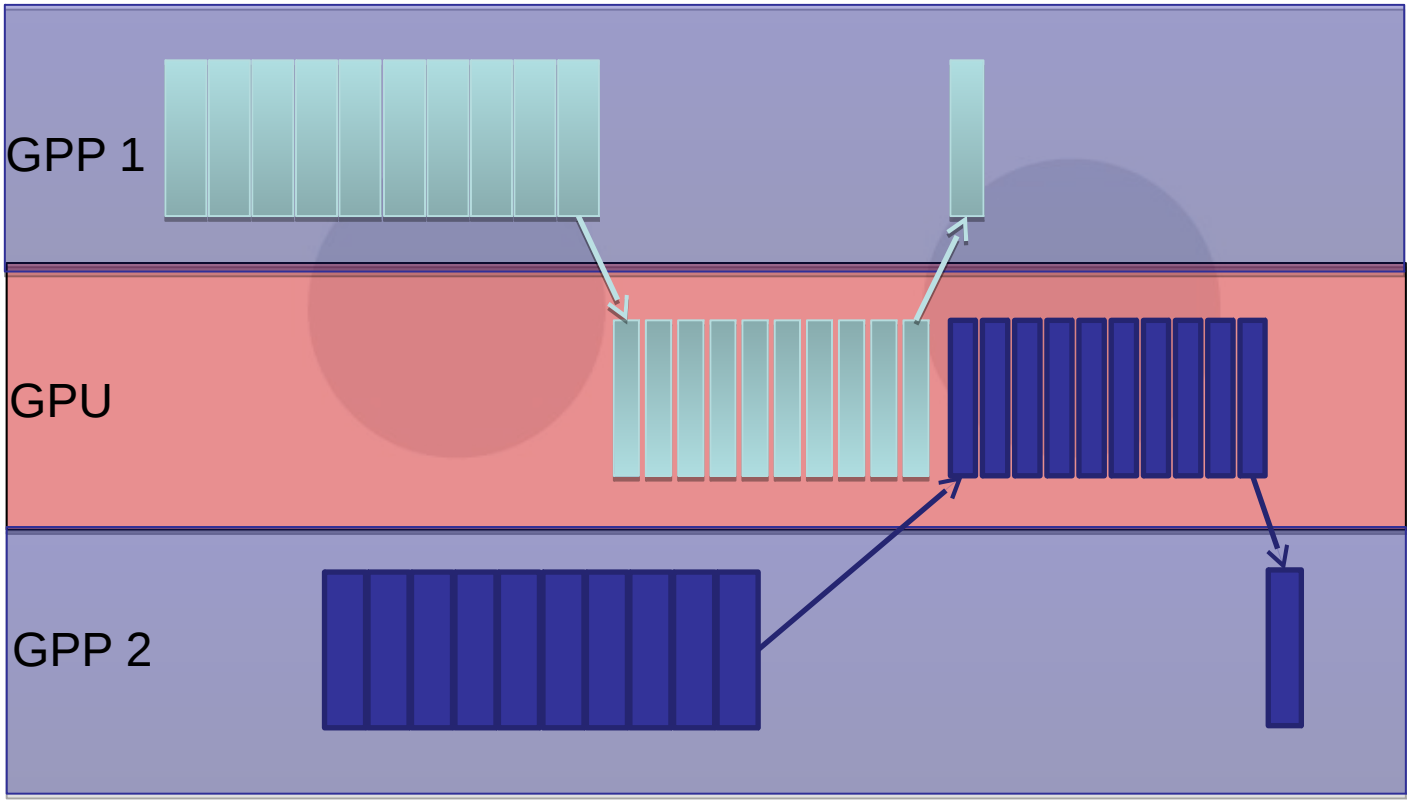
- Used just like gr in GNU Radio flowgraphs
  - import grgpu
  - op = grgpu.op_cuda()
- Dependent blocks
  - Host/device data movement handled by other blocks
  - Has device pointers as interface

GRGPU Example

```
class GPU_Thread(threading.Thread):
    """GPU Thread"""
    def init(self):
        self.tb = gr.top_block ()
        src_data = list(math.sin(x) for x in range(TOTAL_LENGTH))
        src = gr.vector_source_f (src_data)

        h2d = grgpu.h2d_cuda()
        taps = [0.1]*60
        op  = grgpu.fir_filter_fff_cuda(taps)
        d2h = grgpu.d2h_cuda()

        self.dst = gr.vector_sink_f ()
        self.tb.connect(src, h2d)
        self.tb.connect(h2d, op)
        self.tb.connect(op, d2h)
        self.tb.connect(d2h, self.dst)
```



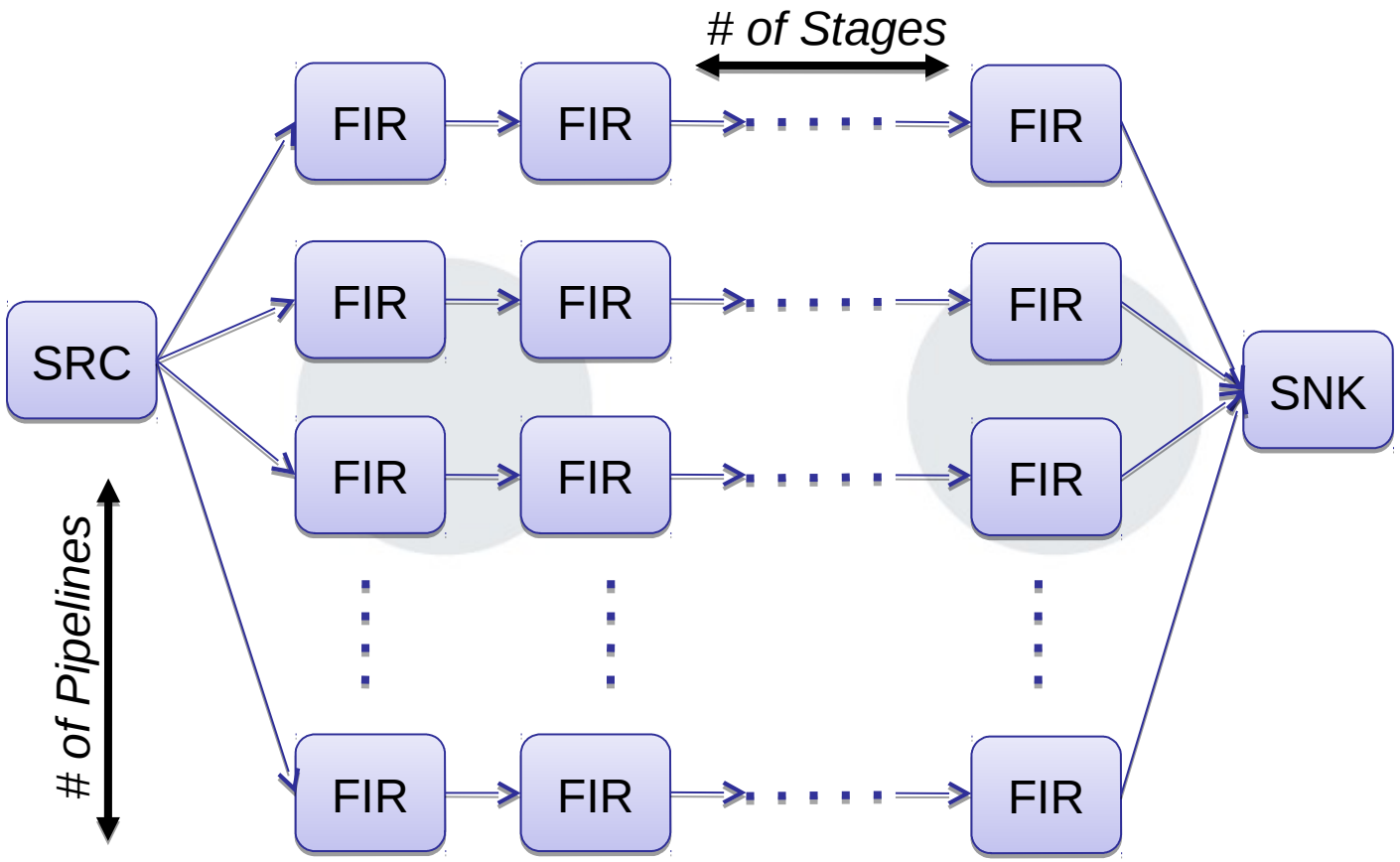
Multicore with GRGPU



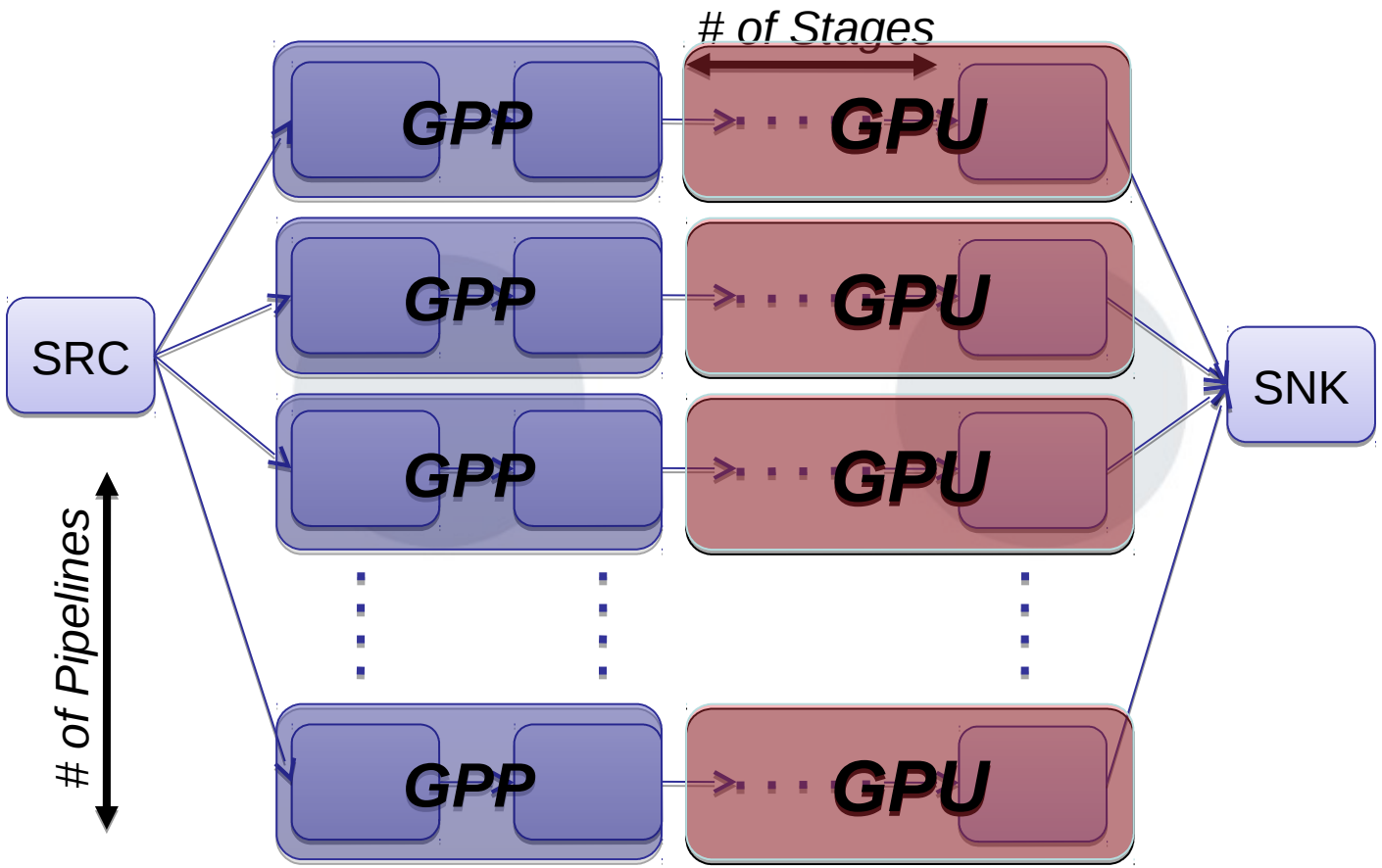
Evaluation

- Benchmark GPU options with GNU Radio multicore benchmark (mp-sched)
  - 2 Intel Xeons (3GHz) and 1 GTX 260
- GPU and GPP accelerated FIR
  - CUDA and SSE acceleration
- Vary: size and assignment
- Measure: total execution time

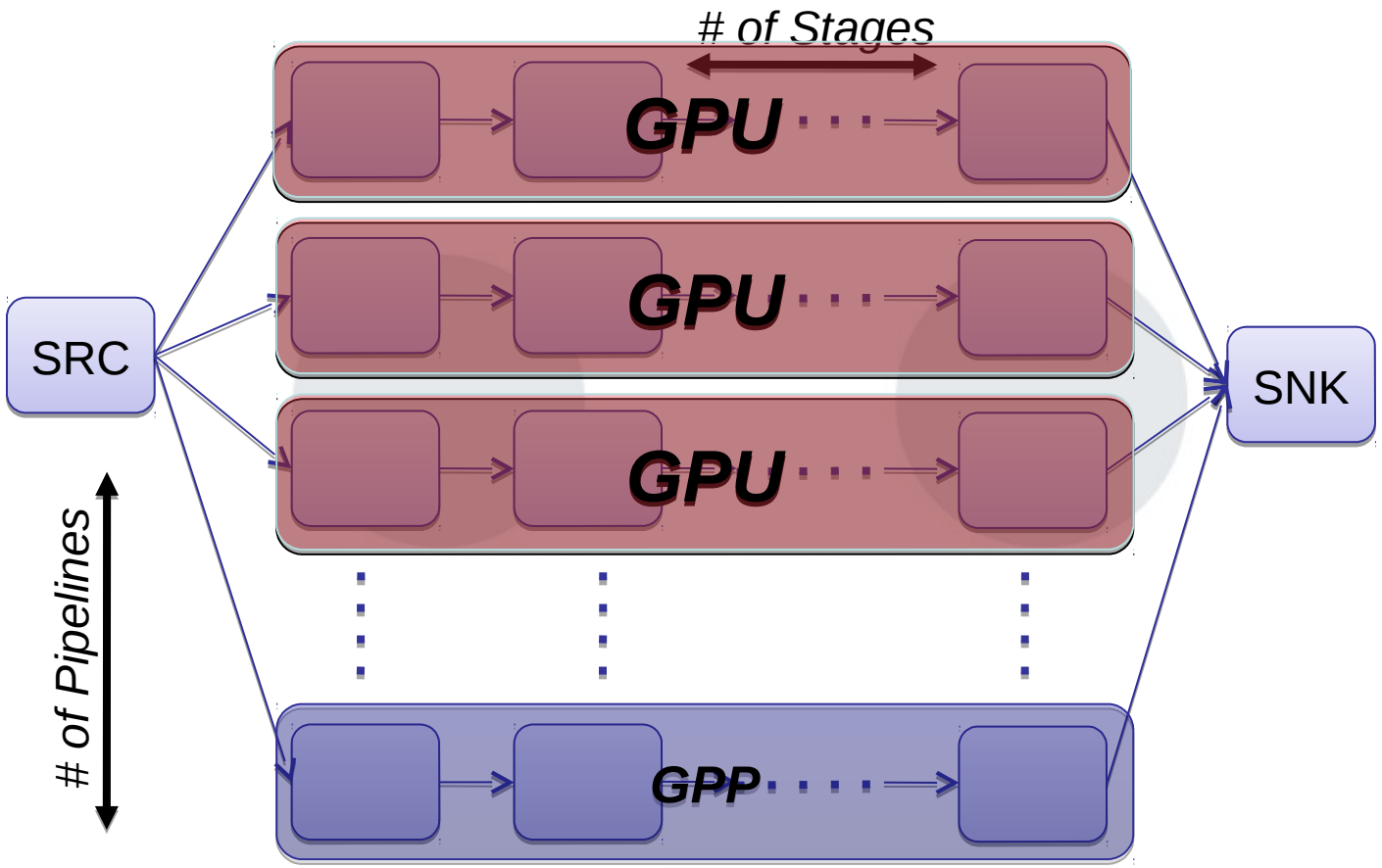
Evaluation Application



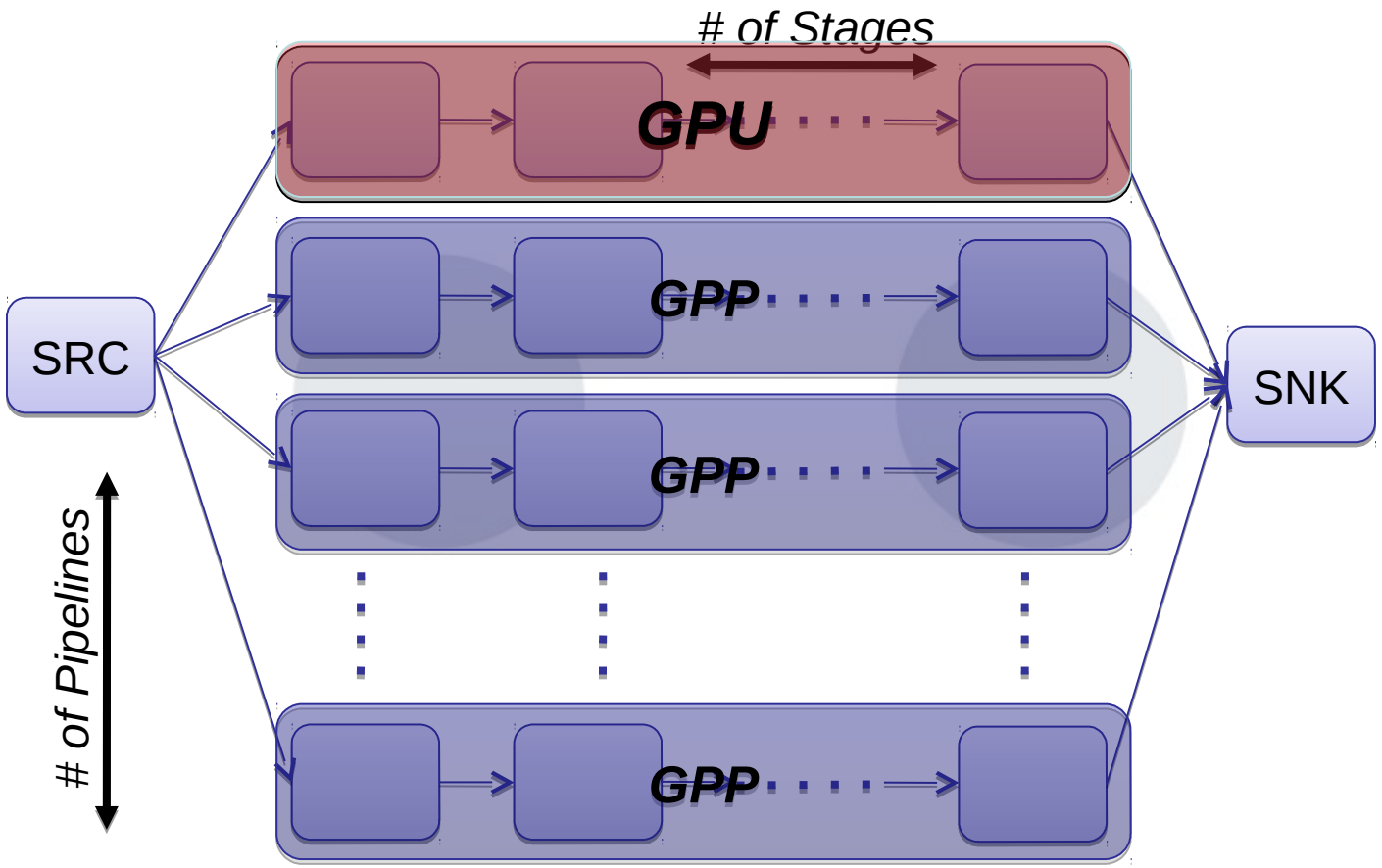
Evaluation Application – "Half CPU"



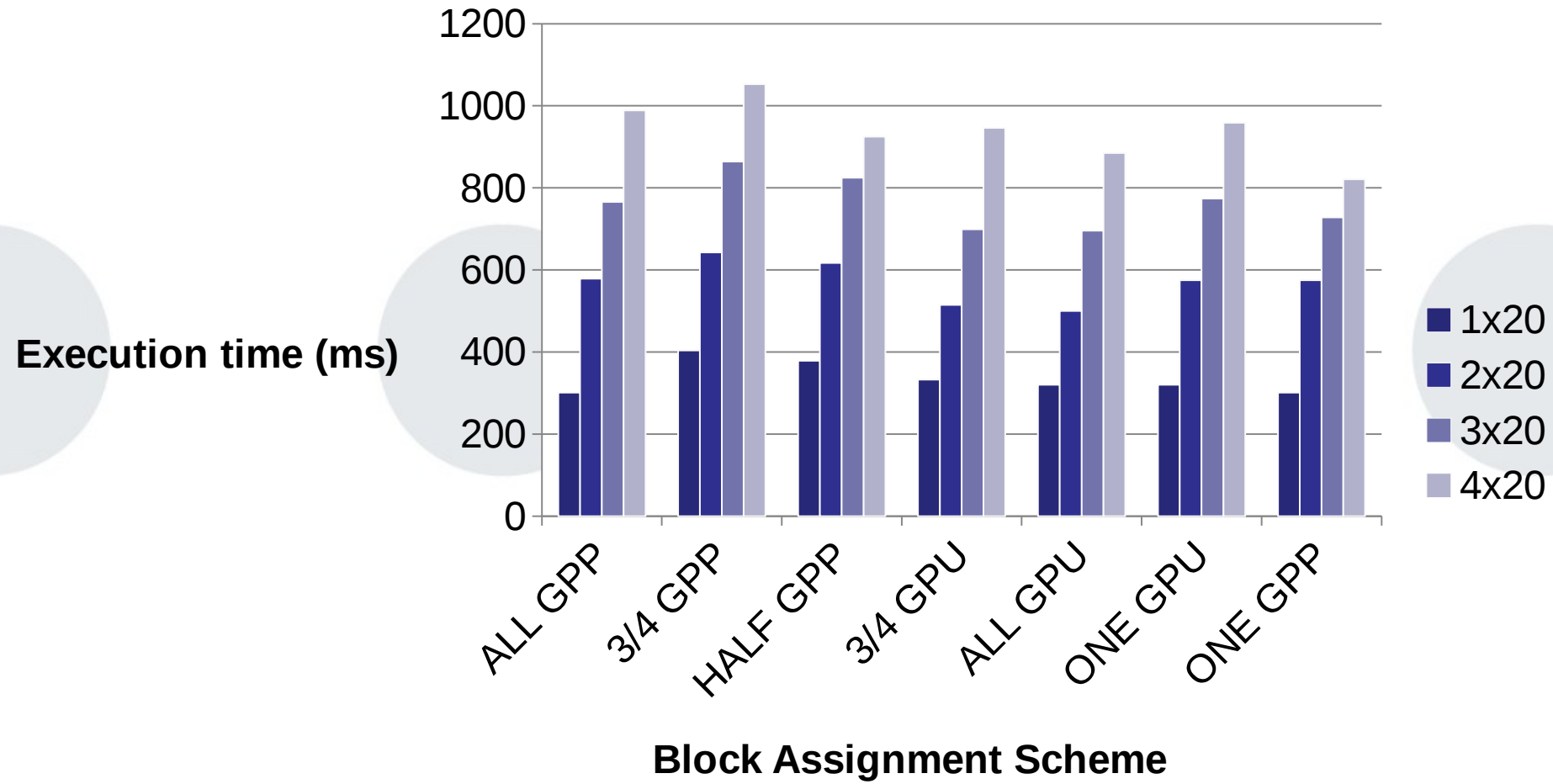
Evaluation Application – "One GPP"



Evaluation Application – "One GPU"



Results



Conclusions

- GRGPU provides GPU acceleration to GNU Radio as a stand alone library
  - Facilitates design space exploration
- Status
  - Planned to be contributed back
  - Few but growing number of CUDA accelerated blocks
- Future Work
  - Automated Mapping
  - MultiGPU

Questions?

Create GRGPU Block

- python script
  - ./create_new_cuda_block.py <new_block_cuda>
- Creates and internally renames:
  - ./grc/grgpu_new_block_cuda.xml
  - ./swig/grgpu_new_block_cuda.i
  - ./lib/grgpu_new_block_cuda_kernel.cu
  - ./lib/qa_grgpu_new_block_cuda.cc
  - ./lib/grgpu_new_block_cuda.cc
  - ./lib/qa_grgpu_new_block_cuda.h
  - ./lib/grgpu_new_block_cuda.h
- Rewrites:
  - ./lib/Makefile.am
  - ./lib/make-cuda
  - ./swig/grgpu.i
  - ./swig/Makefile.am