Gravitational Wave Data Analysis Using Naked OpenCL

Máté Ferenc Nagy-Egri
Wigner RCP of the HAS
GPU-Day 2019 – Budapest

In this talk...

- Physical overview of gravitational waves
- Testing a hypothesis
  - Motivation for OpenCL acceleration
  - Challanges
  - Results
- Future directions
  - SYCL
  - Machine learning

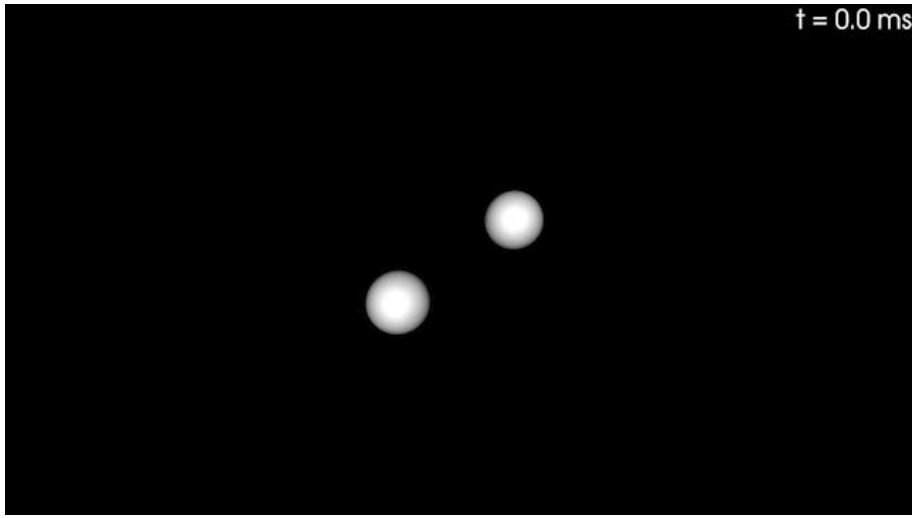
What and how?

GRAVITATIONAL WAVES

Sources of waves

Mergers

- **Black holes and neutron stars**



Credits: ESO/L. Calçada (www.eso.org)

Continous

- **Wide binary systems**
- **Non-axisymmetric objects**

Credits: W. Kastaun/T. Kawamura/B. Giacomazzo/R. Ciolfi/A. Endrizz (www.eso.org)

Have we gone overboard?

TESTING A HYPOTHESIS

Precise enough?

- Scientific computations default to double-precision floating-point (IEEE 754)
  - Hardware accelerated
  - Seldom have to fear rounding errors
- What do we sacrifice?
  - Double the bandwidth
  - Low-power architectures
    - Low/mid-range dedicated/integrated GPUs
    - ULV/mobile

Nail & hammer

Source: Marco Verch (https://flic.kr/p/CTUznS)

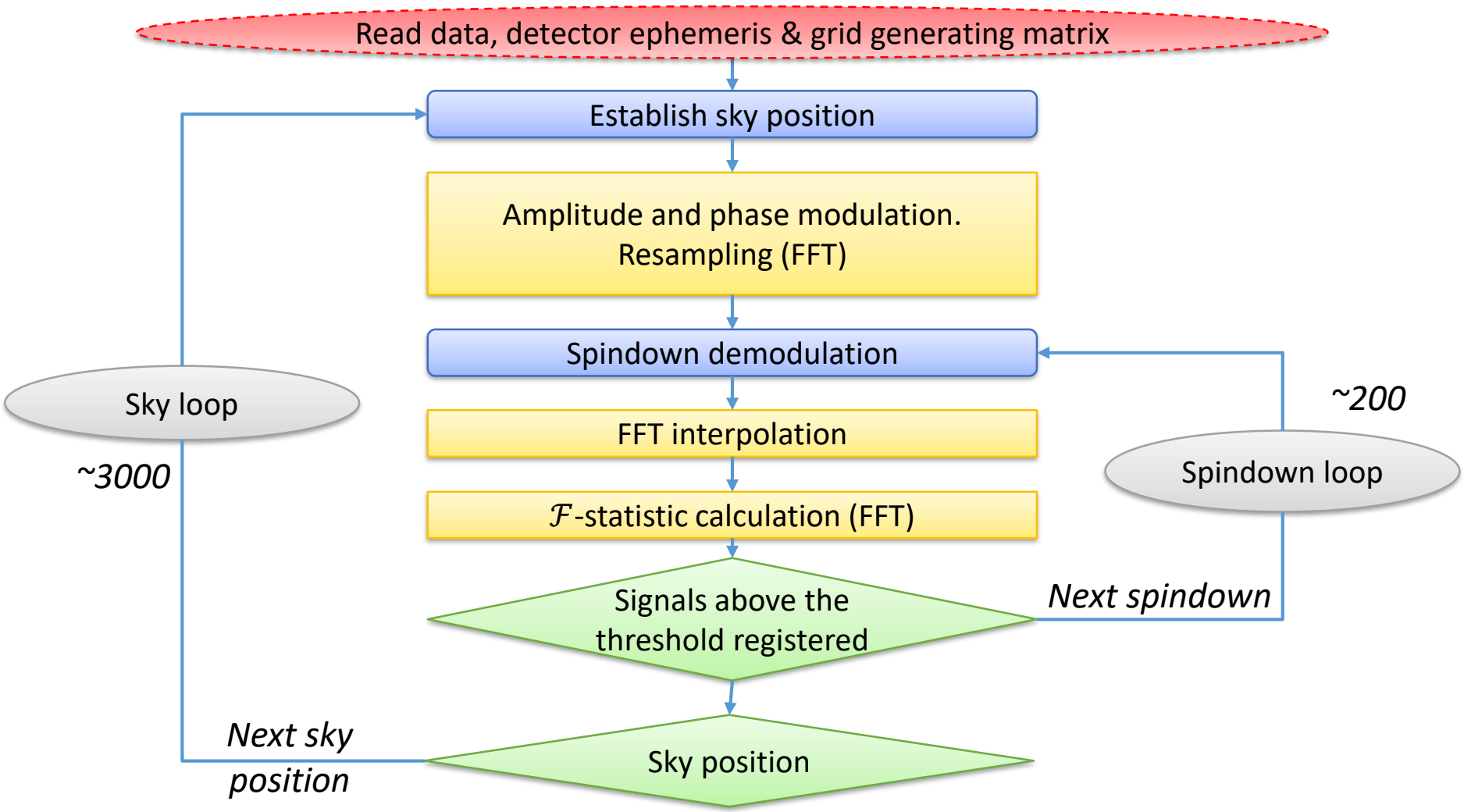
Nail & minigun

Source: Marcin Wichary (https://flic.kr/p/4MEakL)

Why OpenCL?

- OpenCL is a Khronos standard for data parallel computations
- It's portable
  - Not 100% performance portable (see later)
  - Cross-platform, cross-vendor
  - Target non-existing architectures
    - ~Future proof? (Intel 2020)
- Generic GPGPU optimizations result in decent multi-core CPU code

Algorithmic overview



Challanges

- GNU C to ISO C11
  - Portable warning free compilation is a nightmare
  - The secure extensions to the C runtime isn’t a success story
  - POSIX extensions to the CRT are ubiquitous
  - Complex numbers with MSVC still remain a dream

Challanges

- Isolating state
  - Remove all global state
  - Remove c-style stateful functions
  - Const correctness
- Adding parallelism
  - Coarse grain
    - Need an efficient thread-pool implementation
      - OpenMP to the rescue!
  - Fine grain
    - OpenCL

Challanges

- OpenCL
  - Buggy runtimes
    - Nvidia kernel cache bug (4 years old!)
  - Disenchantingly neglected on some platforms
    - AMD + Windows (3-4 years without update)
    - Apple (deprecated)
  - Most common libraries are on life-support
    - Bug fixes only, no new features
    - clBLAS and clFFT share non-standard API usage in their kernel cache mechanism ☹☹

Challanges

```
CLFFTAPI clfftStatus
clfftEnqueueTransform(
        clfftPlanHandle plHandle,
        clfftDirection dir,
        cl_uint numQueuesAndEvents,
        cl_command_queue* commQueues,
        cl_uint numWaitEvents,
        const cl_event* waitEvents,
        cl_event* outEvents,
        cl_mem* inputBuffers,
        cl_mem* outputBuffers,
        cl_mem tmpBuffer)
```

- When numQueuesAndEvents != 0
  - CLFFT_NOTIMPLEMENTED
- Documentation reads:

„Currently, you must manage the multi-device operation. You can create OpenCL contexts that are associated with multiple devices, but clFFT only uses a single device from that context to transform the data. You can manage a multi-device operation by creating multiple contexts, in which each context contains a different device; you are responsible for scheduling and partitioning the work across multiple devices and contexts."

- What happens if you use only 1 device from a multi-device context?
  - AMD: It works as expected
  - Others: **Stack corruption**

Challanges

- Type generic code?
  - #ifdef around actual types
  - Buffers erase stored type

Challanges

- Type generic code?
  - #ifdef around actual types
  - Buffers erase stored type
- Host code involving complex becomes nasty
  - This means OpenCL device code too

```
void spline(const fft_complex* y,
            int n,
            spline_complex* y2,
            spline_complex* u) {
#ifndef _WIN32
// Linux code
#else
#if SPLINE_DOUBLE
#if FFT_DOUBLE
#else
#endif
#else
#if FFT_DOUBLE
#else
#endif
#endif
}
```

Challanges

- Type generic code?
  - #ifdef around actual types
  - Buffers erase stored type
- Host code involving complex becomes nasty
  - This means OpenCL device code too
- Complex numbers?
  - OpenCL C still only reserves the complex keyword, but does not define it
    - OpenCL C 2.2 spec, p.11

„6.1.4. Reserved Data Types

The data type names described in the following table are reserved and cannot be used byapplications as type names. The vector data type names defined in Built-in Vector Data Types, butwhere n is any value other than 2, 3, 4, 8 and 16, are also reserved.

booln
halfn
quad, quadn
complex half, complex halfn
imaginary half, imaginary halfn
complex float, complex floatn
…
floatnxm
…”

Pipelining

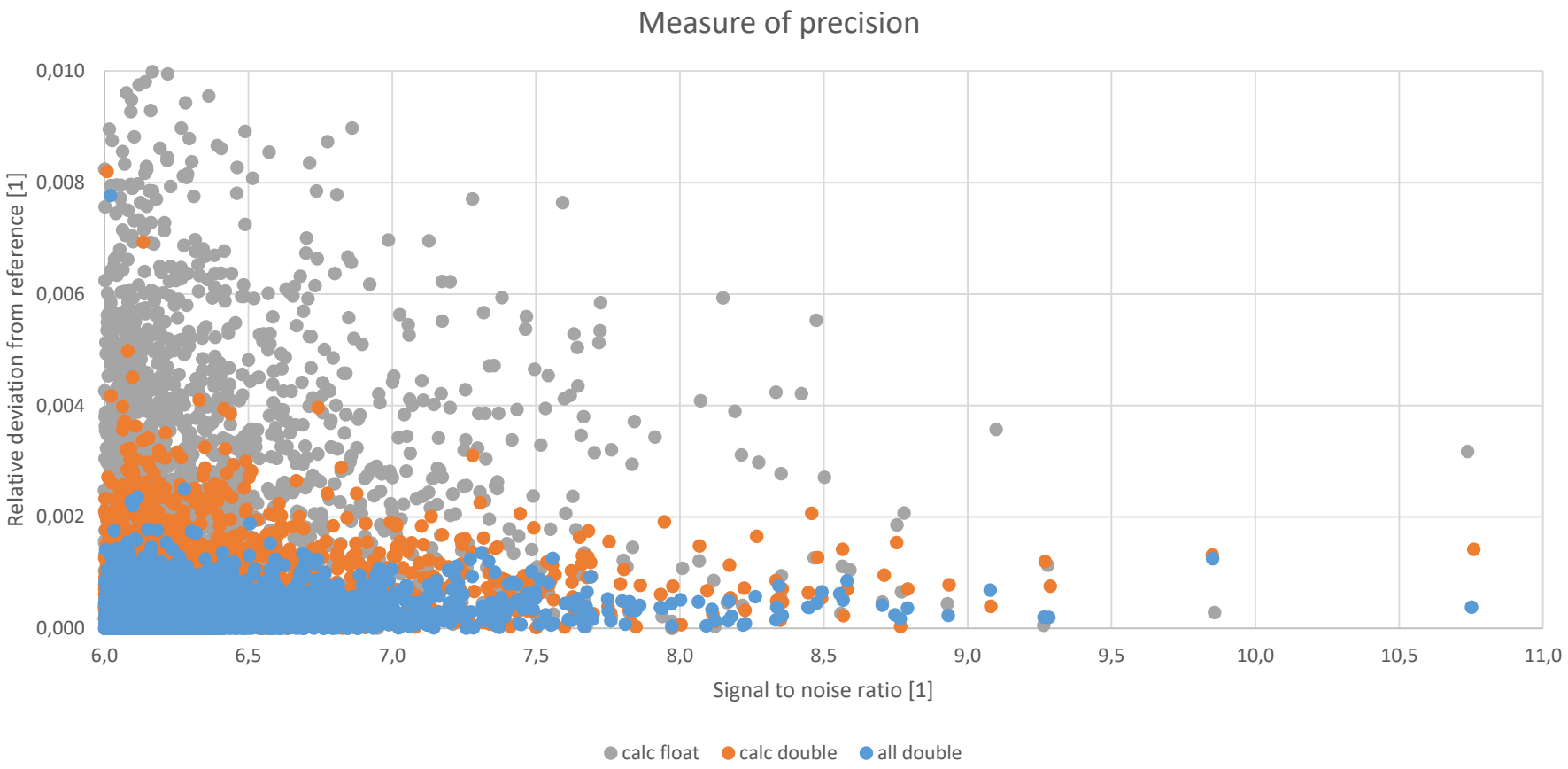
```
typedef struct _pipeline
{
  cl_event *modvir_events,
           *tshift_pmod_events;
  cl_event **fft_interpolate_fw_fft_events,
           **fft_interpolate_resample_copy_events,
           **fft_interpolate_resample_fill_events,
           **fft_interpolate_inv_fft_events,
           **spline_map_events,
           **spline_unmap_events,
           **spline_blas_events,
           **blas_dot_events;
  cl_event *mxx_fill_events,
           *axpy_events,
           *phase_mod_events,
           *zero_pad_events,
           *fw2_fft_events;
  cl_event compute_Fstat_event,
           normalize_Fstat_event,
           peak_map_event,
           peak_unmap_event;

} Pipeline;
```

- Being „chatty” over PCI-E is not good
- Issue as many commands up front as possible
- Allow the runtime to schedule as it sees fit

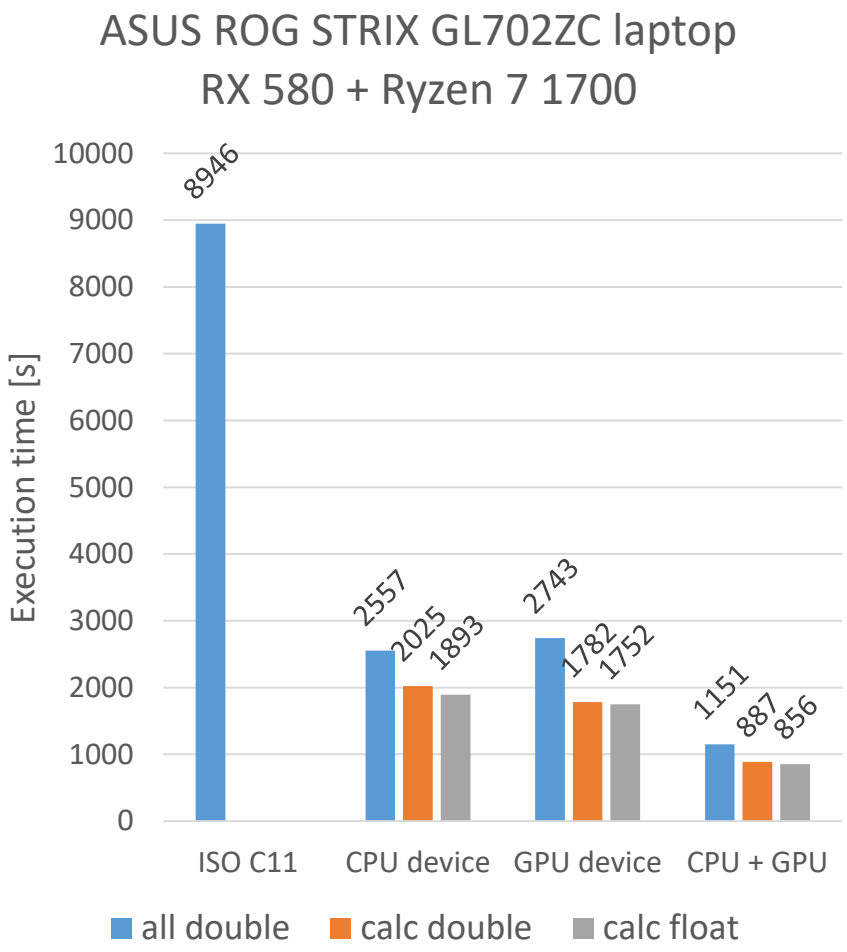
Results

Results



Preliminary results on the effect of floating-point precision on the search pipeline

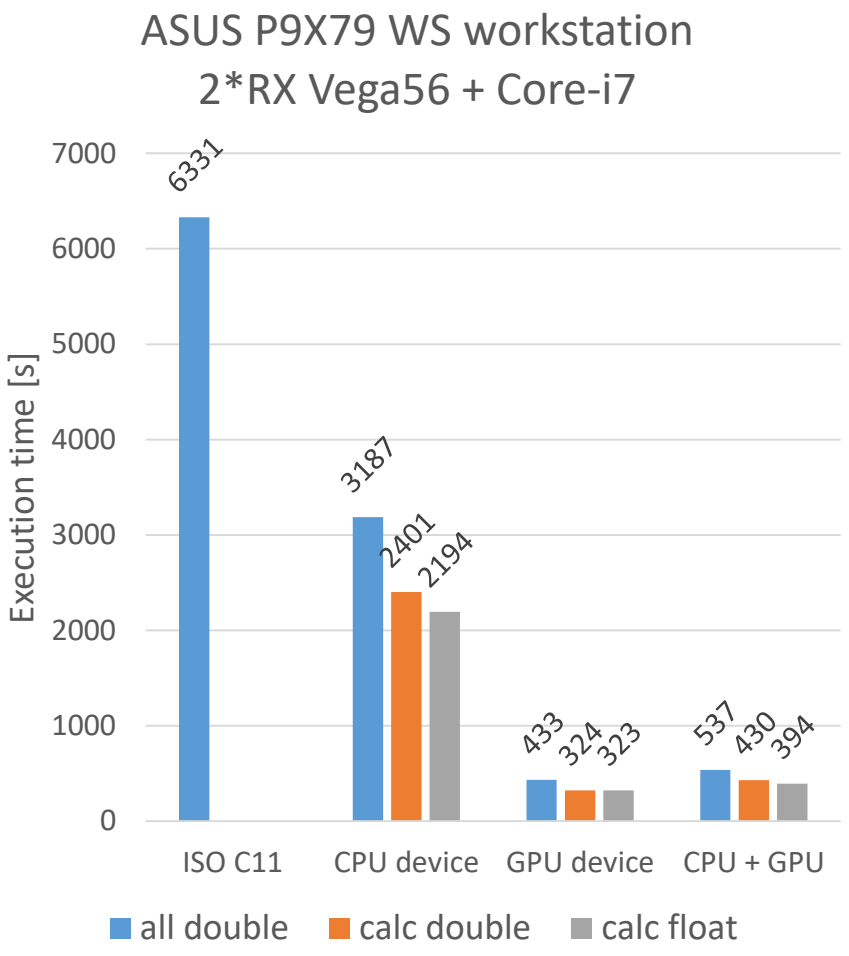
Results



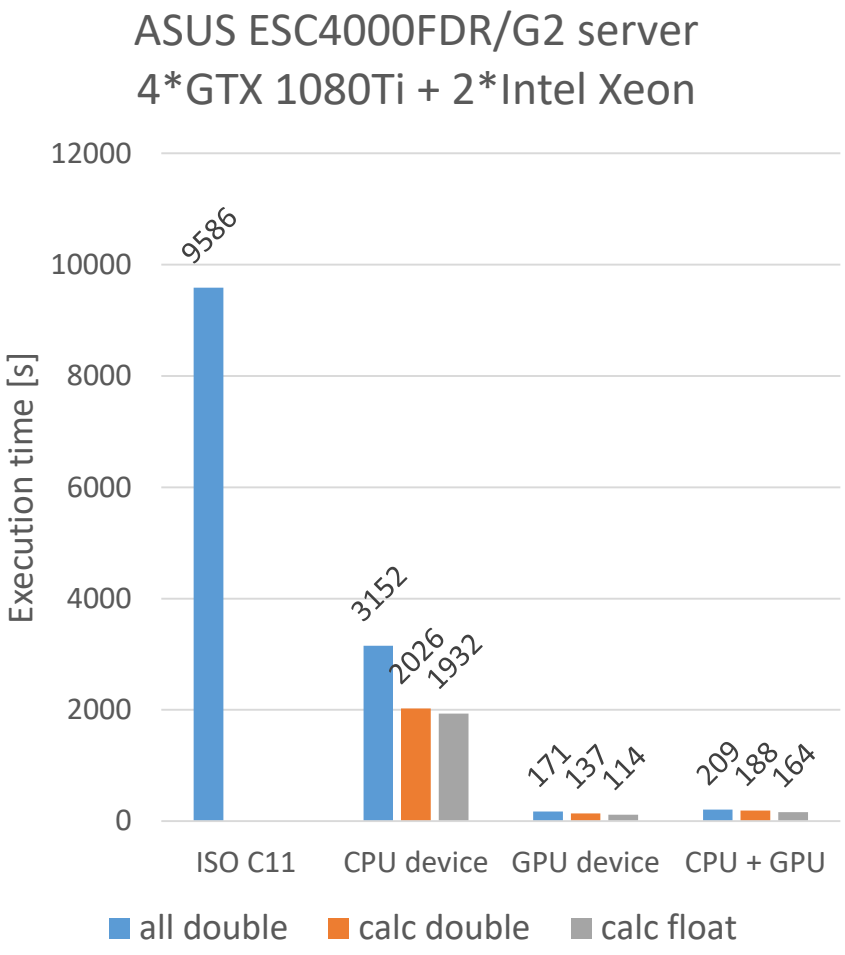
- Code runs mostly as expected
  - AMD runtime on Windows is terribly slow
    - No linux run ☹
  - A great deal of perf is lost on data movement
- Using both the CPU & GPU is a win

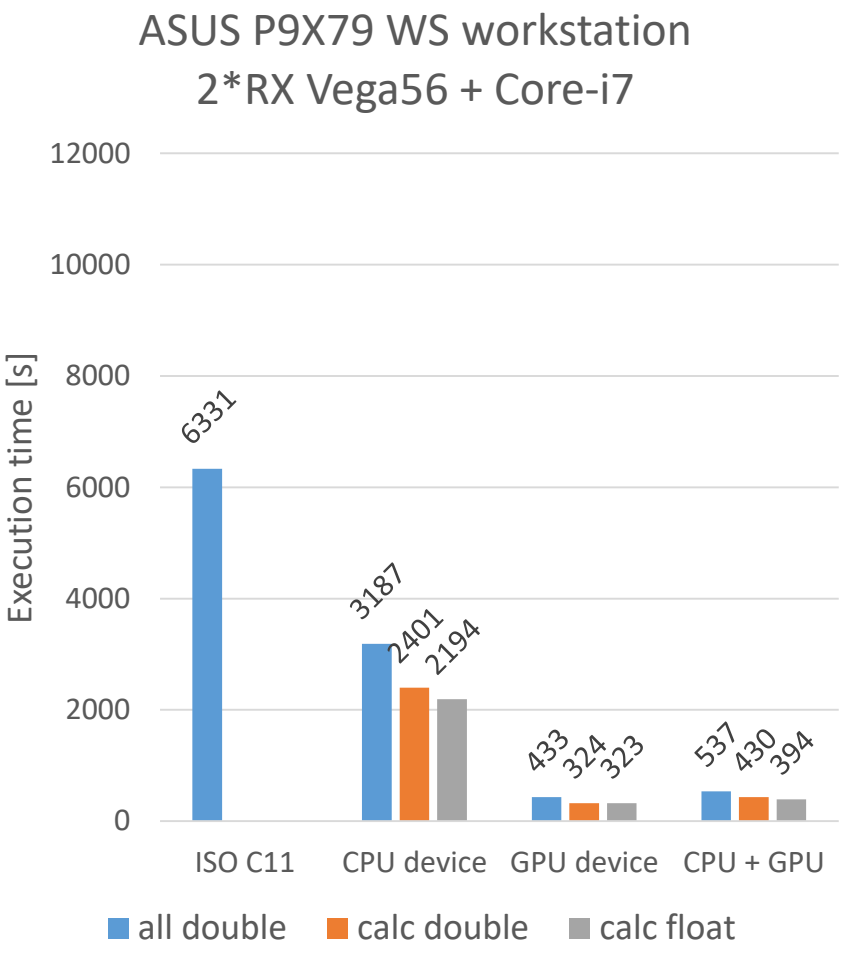
Results

- On a beefier linux node
  - 2X Radeon Vega56
  - Higher quality runtime
- Using the GPU device is a clear win
- Using both CPU & GPU is a net loss
  - The idle time of the GPUs are not compensated

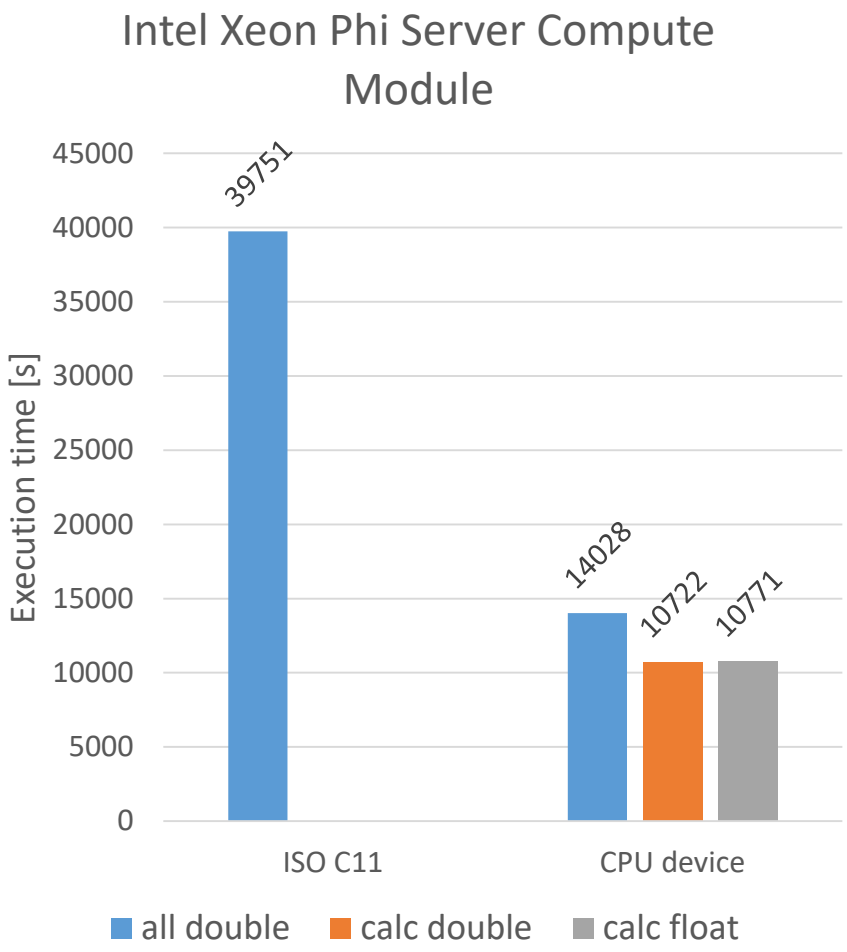


Results


ASUS ESC4000FDR/G2 server
4*GTX 1080Ti + 2*Intel Xeon
Execution time [s]
12000
10000
8000
6000
4000
2000
0
9586
3152
2026
1932
171
137
114
209
188
164
ISO C11
CPU device
GPU device
CPU + GPU
all double
calc double
calc float


ASUS P9X79 WS workstation
2*RX Vega56 + Core-i7
Execution time [s]
12000
10000
8000
6000
4000
2000
0
6331
3187
2401
2194
433
324
323
537
430
394
ISO C11
CPU device
GPU device
CPU + GPU
all double
calc double
calc float

Results

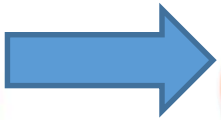


- Running serial CPU code on a MIC is a terrible idea
  - As expected
- OpenCL code performs slower than anticipated
  - Requires a deep dive to find out why

„Et maintenant, on va ou?”

FUTURE DIRECTIONS

OpenCL to SYCL



OpenCL

What else do we gain?

- Say no to macros, say hello to templates
- Two high-quality libraries ML libraries
  - SYCL-ML, SYCL-DNN
- Not limited to built-in FFT types
  - half? double_float?
- Portable complex numbers
  - std::complex FTW
- More robust, readable, maintainable code

What else do we gain?

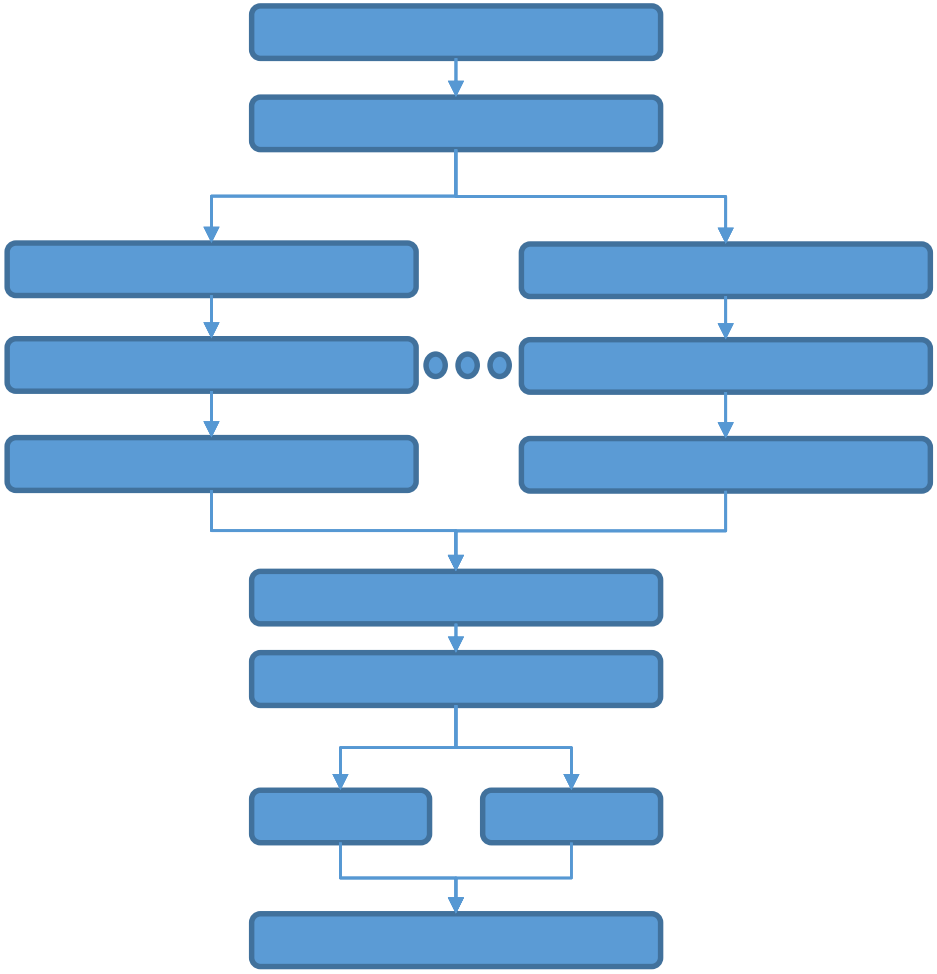
```
typedef struct _pipeline
{
  cl_event *modvir_events,
           *tshift_pmod_events;
  cl_event **fft_interpolate_fw_fft_events,
           **fft_interpolate_resample_copy_events,
           **fft_interpolate_resample_fill_events,
           **fft_interpolate_inv_fft_events,
           **spline_map_events,
           **spline_unmap_events,
           **spline_blas_events,
           **blas_dot_events;
  cl_event *mxx_fill_events,
           *axpy_events,
           *phase_mod_events,
           *zero_pad_events,
           *fw2_fft_events;
  cl_event compute_Fstat_event,
           normalize_Fstat_event,
           peak_map_event,
           peak_unmap_event;

} Pipeline;
```

What else do we gain?

```
typedef struct _pipeline
{
  cl_event *modvir_events,
           *tshift_pmod_events;
  cl_event **fft_interpolate_fw_fft_events,
           **fft_interpolate_resample_copy_events,
           **fft_interpolate_resample_fill_events,
           **fft_interpolate_inv_fft_events,
           **spline_map_events,
           **spline_unmap_events,
           **spline_blas_events,
           **blas_dot_events;
  cl_event *mxx_fill_events,
           *axpy_events,
           *phase_mod_events,
           *zero_pad_events,
           *fw2_fft_events;
  cl_event compute_Fstat_event,
           normalize_Fstat_event,
           peak_map_event,
           peak_unmap_event;

} Pipeline;
```



What else do we gain?

```
typedef struct _pipeline
{
  cl_event *modvir_events,
           *tshift_pmod_events;
  cl_event **fft_interpolate_fw_fft_ev
           **fft_interpolate_resampl
           **fft_interpolate_resam
           **fft_interpolate_inv_
           **spline_map_events,
           **spline_unmap_event
           **spline_blas_events
           **blas_dot_events;
  cl_event *mxx_fill_events,
           *axpy_events,
           *phase_mod_events,
           *zero_pad_events,
           *fw2_fft_events;
  cl_event compute_Fstat_event,
           normalize_Fstat_event,
           peak_map_event,
           peak_unmap_event;

} Pipeline;
```

Machine Learning

- The search pipeline outputs too many candidates to investigate individually.
- Filter output by a classifier
  - Teach on synthetic data (?)
  - Data can be generated ON the GPU
  - Feed immediately to a training algorithm

To those whom it may concern

AMMENDMENT

SYCL-PRNG

- Open-source
- Initially supports two PRNGs
  - MWC64X (MultiplyWithCarry), Tiny-Mersenne Twister
- STL compatible
  - Satisfies the UniformRandomBitGenerator concept
- SYCL compatible
  - Satisfies StandardLayoutType concept
- It is in alpha-beta stage
  - Should you wish to help push it through the finish line, don't hesitate to submit a PR

THANK YOU FOR YOUR ATTENTION