SDR on GPU's: Practical OpenCL Blocks and Their Performance

Mike Piscopo, Director of Technical Consulting Services

DELTA RISK
A CHERTOFF GROUP COMPANY

GRCon 2017

Speaker Introduction – Who is this guy?

Mike Piscopo

- Delta Risk LLC - Director CyberSecurity Technical Consulting Services and Product Development
- ING U.S. Financial Services - Information Security Officer
- PeriNet Technologies - President and CTO, security solutions architecture and implementation, penetration testing
- Past "Lives"
  - BS Aerospace Engineering – Virginia Tech
  - Aerospace Contractor – Real-time distributed centrifuge team
  - Developer – embedded C++ and later application architect
  - Network Field Engineer and IT architect

Why Look to GPU’s and OpenCL?

- In cyber, we use GPU’s all the time for hash generation and cracking
  - Highly parallel computing with OpenCL and can leverage multiple cards
- Lots of talk about the benefits of GPU’s and signal processing (clfft libraries, etc.)
- On the surface it sounded like a reasonable question: Why don’t we have OpenCL SDR blocks?
  - NVIDIA 1070 has 1,920 cores and drives virtual reality systems
  - OpenCL Added bonus: cross-hardware capabilities with support for CPU’s, GPU’s, and OpenCL-enabled FPGA’s
  - Several parallelization modes: data parallel [“Single Instruction Multiple Data” (SIMD)] and task-parallel
- There is some great foundational research and proof-of-concept already available but no core open source GNURadio OOT modules covering all of the “basics”

CL-Enabled Project Goals

This gave birth to the cl-enabled project (github and pybombs). My "wish list":

- Implement blocks commonly used in digital demodulation to run on my GPU using OpenCL
  - Blocks you most frequently would use in a general flowgraph (signal source,multiply, filters)
  - Digital signal processing demod blocks for ASK, FSK, and PSK (2PSK/QPSK)
  - Opportunistically any other core blocks that looked like they could be accelerated
- Scalability
  - We use multiple cards in cyber, so I want to be able to use multiple GPU's in the same flowgraph simultaneously
  - Be able to choose what blocks run on which OpenCL devices
- Would the blocks support 60 Msps real-time processing? What about 250 Msps?
- Have to be able to measure performance
  - Know if the blocks actually ran better or worse on GPU's and why (need to add tools)
  - Categorize blocks into accelerated, offload, and enabled
- Extensibility - Custom kernel support and the ability to time them without re-compiling code

Which Modules Have Been Implemented?

✓ Basic Building Blocks

1. Signal Source (more “pure” as well)
2. Multiply
3. Add
4. Subtract
5. Multiply Constant
6. Add Constant
7. Filters (FIR and FFT)
   I. Low Pass
   II. High Pass
   III. Band Pass / Reject
   IV. Root-Raised Cosine
   V. Generic Tap-Based

✓ Custom OpenCL Kernels

8. 1 input, 1 output
9. 2 inputs, 1 output

✓ Common Math or Complex Data Functions

10. Complex Conjugate
11. Multiply Conjugate
12. Complex to Arg
13. Complex to Mag Phase
14. Mag Phase to Complex
15. Log10
16. SNR Helper (a custom block performing divide->log10->abs)
17. Forward FFT
18. Reverse FFT

✓ Digital Signal Processing

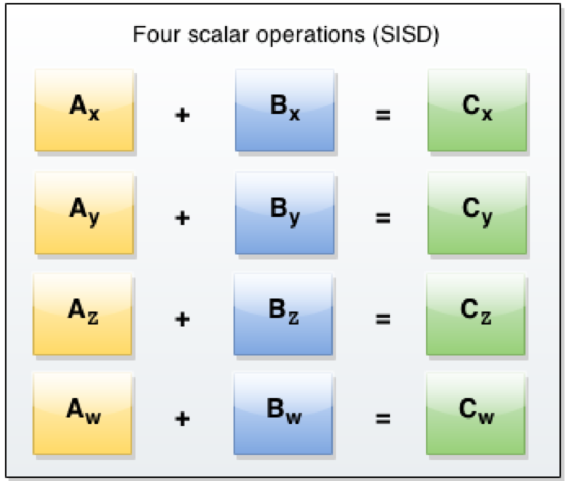
19. Complex to Mag (used for ASK/OOK)
20. Quadrature Demod (used for FSK)
21. Costas Loop (used for BPSK/QPSK)

What makes an algorithm a good choice?

ATOMIC Calculations

- SIMD wants to run the same line of code on multiple data points simultaneously
- Branching impacts performance (threads need to wait so they're on the same instruction)

Maps Really Well



Suboptimal Performance
Branching and different instructions

```
For (int i=0;i<max;i++) {
        if (in[i] <= TWO_PI)
                out[i] = sin(in[i]);
        else
                out[i] = cos(in[i]);
}
```

Potentially unimplementable in SIMD:
Sequential calculations

```
For (int i=0;i<max;i++) {
        alpha=alpha + sqrt(in[i]);
        out[i] = beta + alpha;
}
```

Real Block: Quad Demod Maps Well

```
volk_32fc_x2_multiply_conjugate_32fc(&tmp[0], &in[1], &in[0], noutput_items);
for(int i = 0; i < noutput_items; i++) {
    out[i] = f_gain * fast_atan2f(imag(tmp[i]), real(tmp[i]));
}
```

SIMD-UnFriendly Calculations

Flattened Costas Loop

```
for(i = 0; i < noutput_items; i++) {
        nco_out = gr_expj(-d_phase);
        optr[i] = iptr[i] * nco_out;

        d_error = (*this.*d_phase_detector)(optr[i]);
        d_error = 0.5 * (std::abs(d_error+1) - std::abs(d_error-1));

        //advance_loop(d_error);
        d_freq = d_beta * d_error + d_freq;
        d_phase = d_phase + d_alpha * d_error + d_freq;

        //phase_wrap();
        if (d_phase > CL_TWO_PI) {
                while(d_phase>CL_TWO_PI) {
                        d_phase -= CL_TWO_PI;
                }
        }
}
```

Problems!
(715 Ksps task-parallel)

Implemented as a single task on 1 core:

```
Testing Costas Loop performance with 8192 items...
OpenCL: using NVIDIA CUDA
OpenCL Run Time:      0.011451 s  (715,387.6875 sps)
CPU-only Run Time:     0.000462 s  (17,719,240.00 sps)
```

Performance Killer #1: Data Copy

- Have to move the data to the card then retrieve the results
- This incurs a transfer cost
- Total GPU execution time = Mem In + function execution + Mem Out
- Total CPU execution time = function execution (volk/SIMD-4 makes it even faster)
- Block sizes – Processing more data in a single transaction can offset the memory transfer cost
- Computational Complexity - Some functions take longer to run on a CPU (sin,cos,atan2,log10) and can be good candidates
- Generic equation for when OpenCL Performs better:

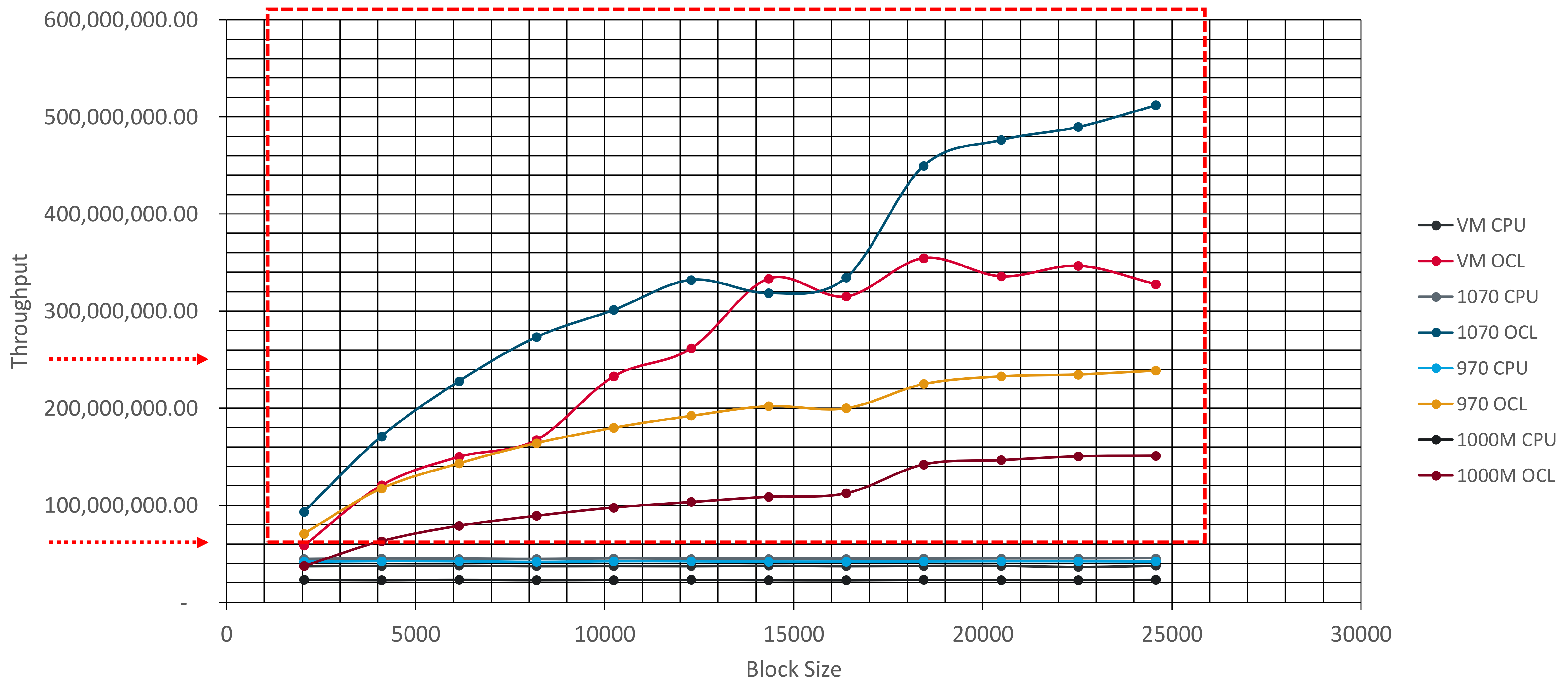
$$T_{MemIn}[\text{block size}] + T_{exec}[\text{block size}] + T_{MemOut}[\text{block size}] < T_{CPU}[\text{block size}]$$

That's the "magic" spot for OpenCL SIMD/GPU acceleration in GNURadio!

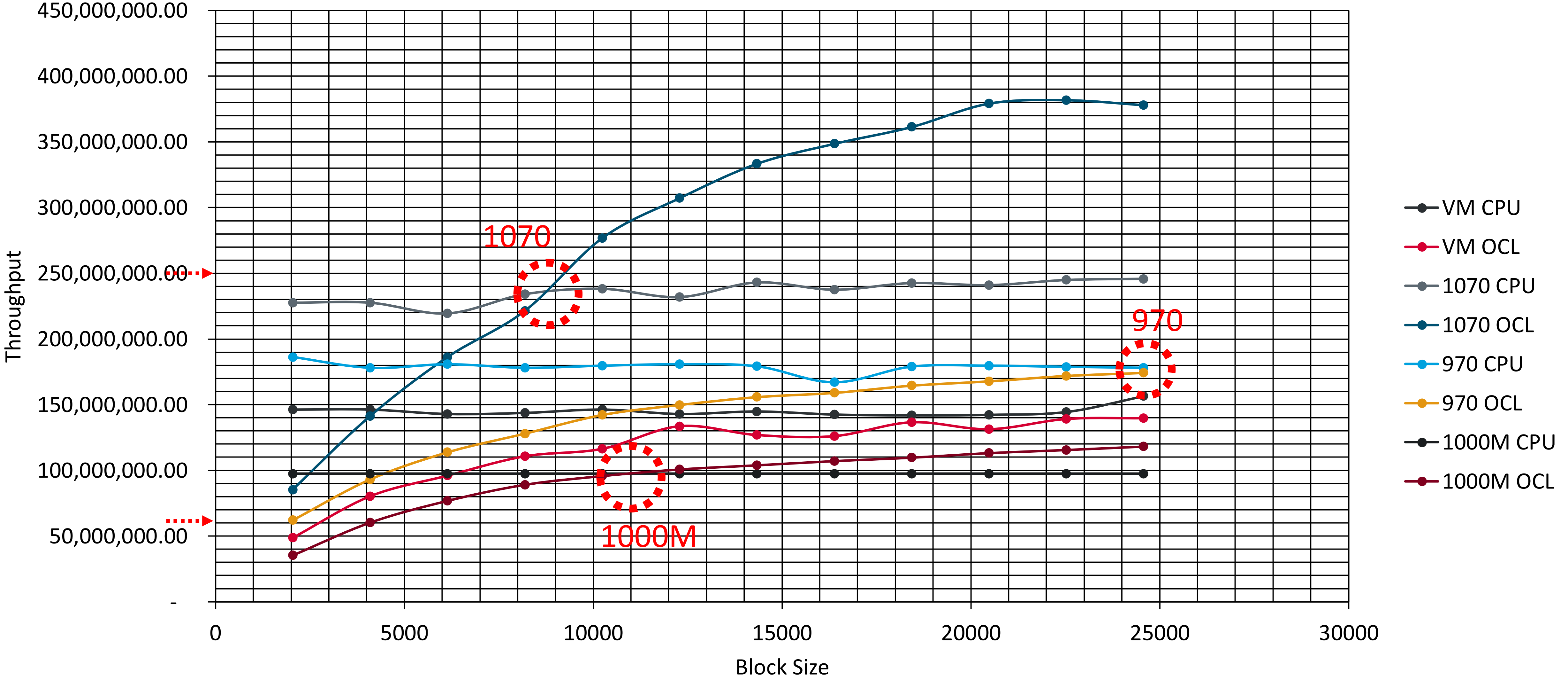
Testing Notes

- Tested on 4 platforms: a new NVIDIA 1070, older 970, laptop 1000M, and OpenCL-CPU
- Block sizes were the actual passed block sizes. Remember GNURadio's scheduling engine may be passing around half the default block size.
- To get the block sizes in the charts, you may have to double it in your flowgraph
- Timing is isolated testing – Straight function calls
- Used included tools for timing (test-clenabled, test-clfilter, test-clkernel)
- Where I could the project even takes advantage of Fused Multiply Add (FMA) for performance

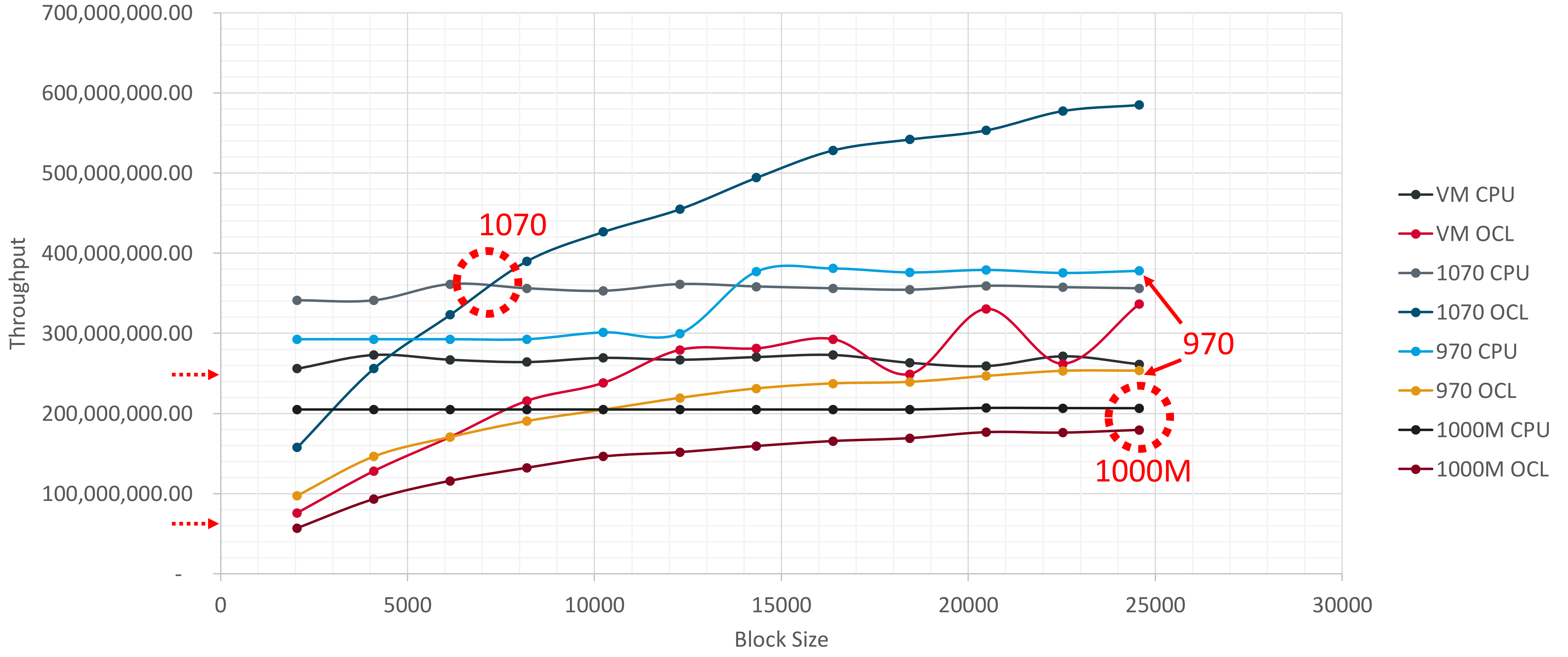
Performance – Log10



Performance – Quad Demod

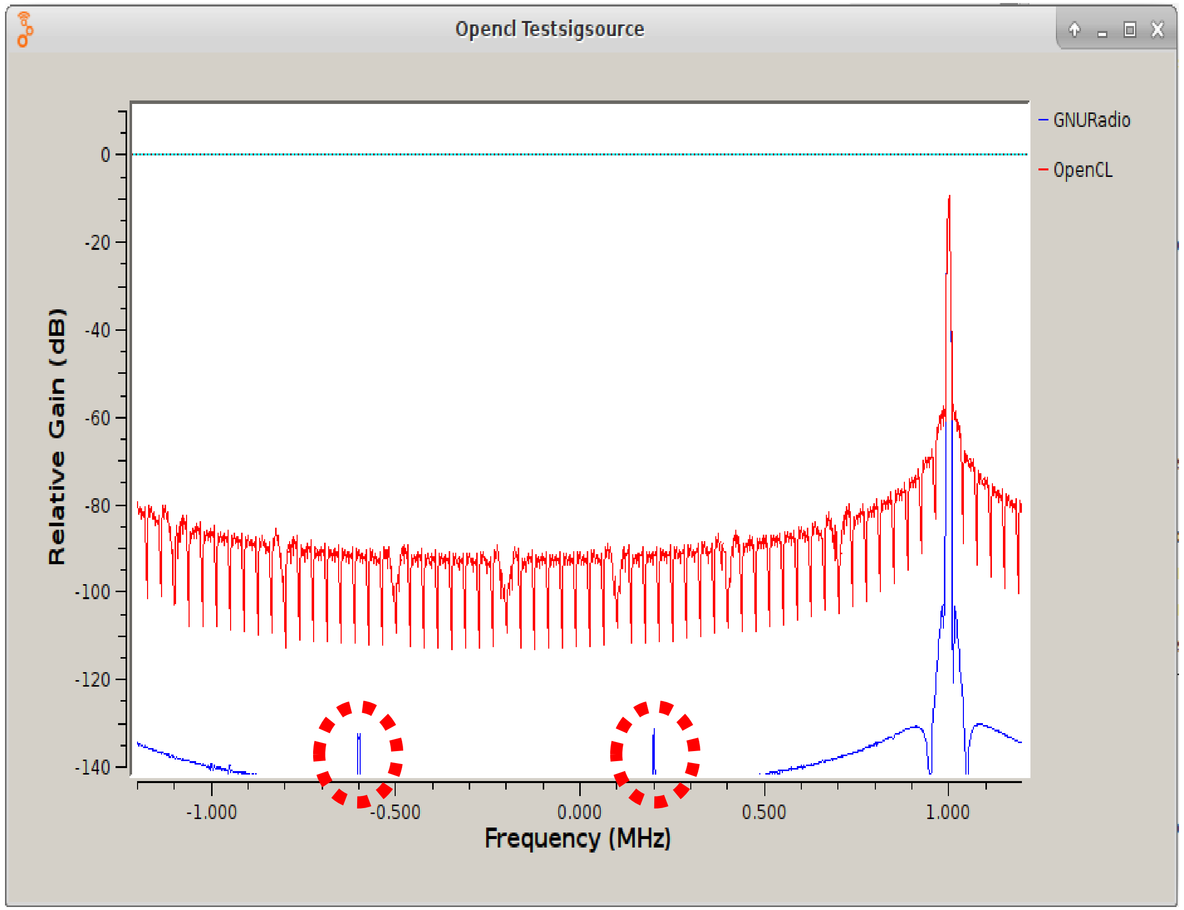


Performance – Signal Source

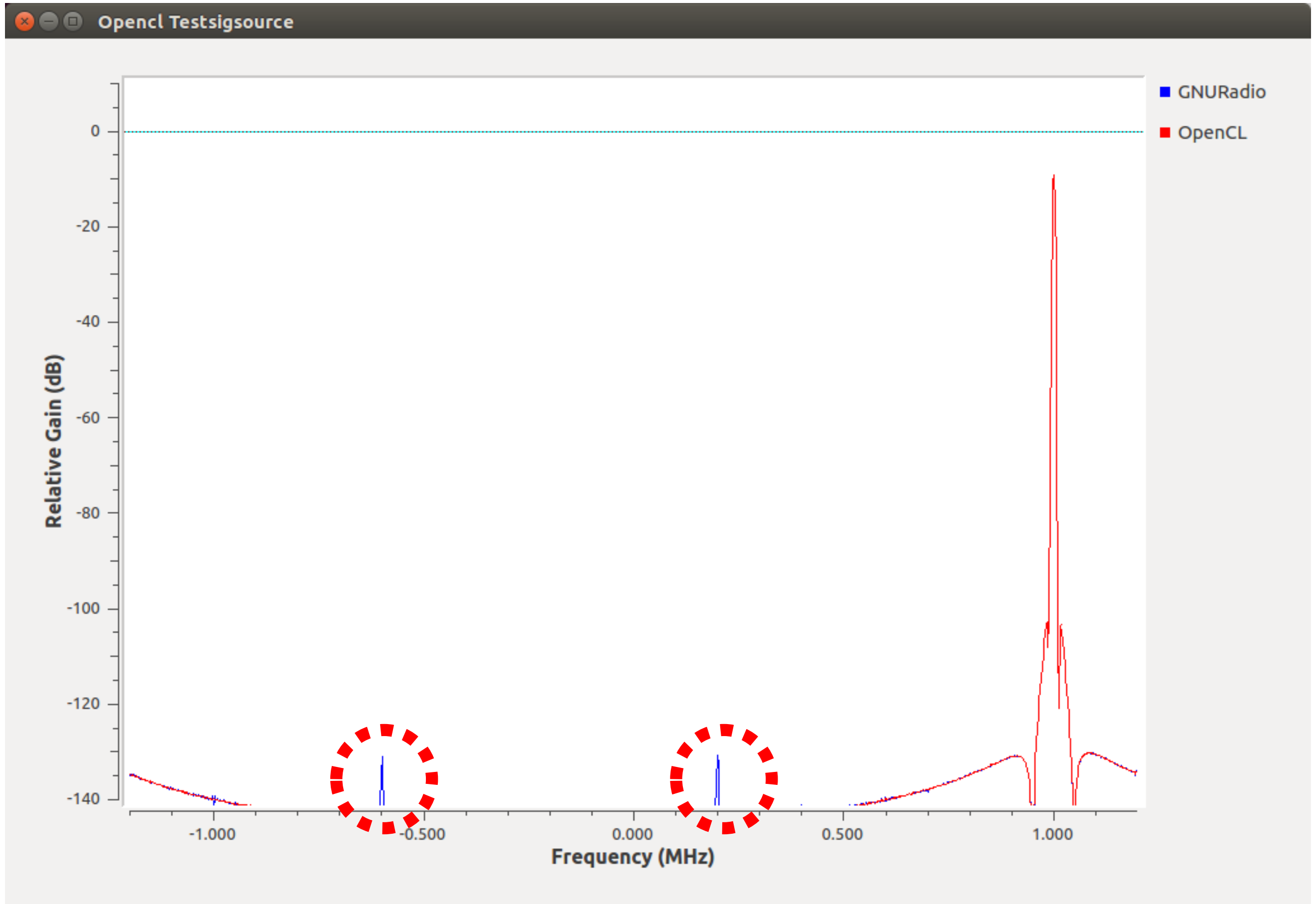


Signal Source – OpenCL and Trig

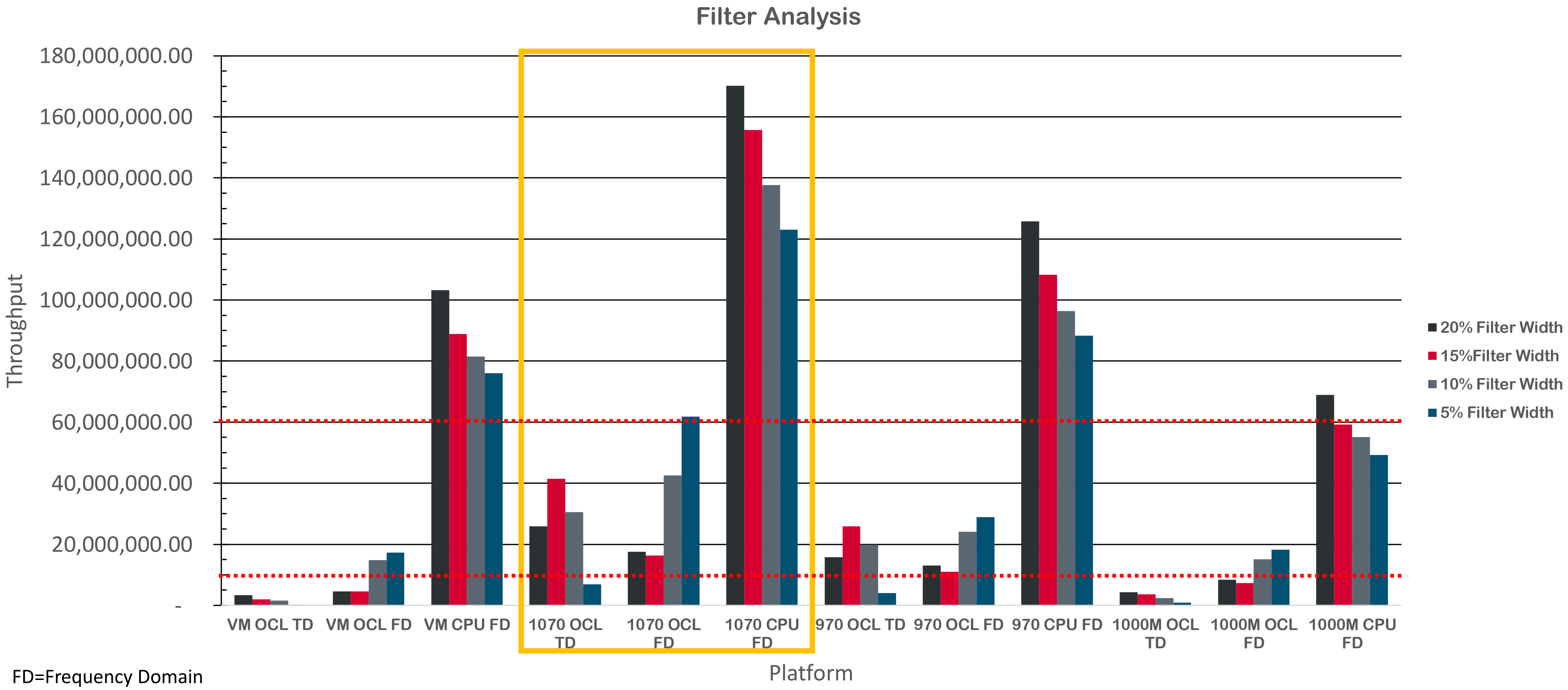
Float Calculations



Double Calculations



Performance – Filters



FD=Frequency Domain
TD=Time Domain

Project Take-Aways

- There are a good number of common OpenCL blocks that can improve performance, especially at high rates
- Time the OpenCL blocks on your hardware with test-clenabled, test-clfilter, and test-clkernel
- Pick the most “expensive” block with an OpenCL version in your flowgraph and use an OpenCL block for that
- Don’t run more than 1 block on a single card simultaneously (sounds obvious but incurs a 100x penalty). Consider multiple cards (blocks can be explicitly assigned to a card. Use clview for the #’s)
- OpenCL signal source is a more “pure” waveform (trig versus lookup table)
- Speed! Other flowgraph performance-boosters if speed is important:
  - CPU FFT Filters outperform FIR for increasing tap sizes (time with test-clfilter, gr-lfast FFT wrappers)
  - Look at gr-lfast for some CPU-optimized blocks where OpenCL doesn’t help (2nd/4th Costas and AGC)
  - If you’re doing signal sources and multiply blocks or an Xlating FIR Filter for offset tuning to get away from a DC spike, consider gr-correctiq or OpenCL signal source block / multiply to get rid of it and reduce CPU load
  - Spread out the work: Use a distributed model: Demod/Decode/Instrumentation. If you like visualization tools like gr-fosphor and you’re running into CPU constraints, consider a net block or gr-grnet to help you move data to other systems and split up your processing and visualization

Q & A

- Download code at https://github.com/ghostop14/gr-clenabled.git or via pybombs
- OpenCL install help on git in setup_help (card doesn’t need 1.2, just for compilation)
- Contact Info: ghostop14@gmail.com