# ArrayFire,
# a tensor interface for oneAPI

Umar Arshad, ArrayFire Maintainer Chair

UXL Math SIG, July 31, 2024

# ArrayFire and UXLF

- Questions raised by this effort:
  - Should UXLF maintain a tensor-based spec/implementation of math functions? In Python, C/C++?
  - How can UXLF include and nurture ecosystem libraries?
  - Should ArrayFire join UXLF? What does that look like?


- High-level math programming, like MATLAB or NumPy/SciPy
- C/C++ Library with wrappers for Python, Rust, Julia, etc.
- Supports multiple platforms including CPU, CUDA, OpenCL, and now oneAPI
- Open Source based on BSD v3
- Performs kernel fusion based on a custom Just-In-Time implementation that generates kernels at runtime

# ArrayFire Design Goals

- Ease of use
  - Minimize the use of templates
  - Pick reasonable defaults
  - Manage library state, memory and caches
- Binary stability and portability
  - Semantic Versioning
  - Stable C interface
  - C++ API and other wrappers built on top of C API
- Focus on throughput instead of latency
  - Target larger problem sets that need to be calculated repeatedly

# Productivity, cross correlation

## oneAPI

```
// Initialize SYCL queue
sycl::queue Q(sycl::default_selector{});

// Allocate 2D image and correlation arrays
auto img1 = sycl::malloc_shared<float>(n_rows*n_cols*2+2, Q);
auto img2 = sycl::malloc_shared<float>(n_rows*n_cols*2+2, Q);
auto corr = sycl::malloc_shared<float>(n_rows*n_cols*2+2, Q);

// Set generalized strides for row-major addressing
int r_stride = 1;
int c_stride = (n_cols / 2 + 1) * 2;
int c_stride_h = (n_cols / 2 + 1);

// Initialize input images with artificial data.
// Do initialization on the device.
Q.parallel_for<>(sycl::range<2>{n_rows, n_cols},
                 [=](sycl::id<2> idx)
{
   unsigned int r = idx[0];
   unsigned int c = idx[1];
   img1[r * c_stride + c * r_stride] = 0.0;
   img2[r * c_stride + c * r_stride] = 0.0;
   corr[r * c_stride + c * r_stride] = 0.0;
}).wait();

Q.single_task<>([=]()
{
   // Set elements in lower right of the first image
   img1[4 * c_stride + 5 * r_stride] = 1.0;
   img1[4 * c_stride + 6 * r_stride] = 1.0;
   img1[5 * c_stride + 5 * r_stride] = 1.0;
   img1[5 * c_stride + 6 * r_stride] = 1.0;

   // Set elements in upper left of the second image
   img2[1 * c_stride + 1 * r_stride] = 1.0;
   img2[1 * c_stride + 2 * r_stride] = 1.0;
   img2[2 * c_stride + 1 * r_stride] = 1.0;
   img2[2 * c_stride + 2 * r_stride] = 1.0;
}).wait();
```

```
// Initialize FFT descriptor
oneapi::mkl::dft::descriptor<oneapi::mkl::dft::precision::SINGLE,
                             oneapi::mkl::dft::domain::REAL>
                                      forward_plan({n_rows, n_cols});

// Data layout in real domain
std::int64_t real_layout[4] = {0, c_stride, 1};

// Data layout in conjugate-even domain
std::int64_t complex_layout[4] = {0, c_stride_h, 1};

forward_plan.set_value(oneapi::mkl::dft::config_param::INPUT_STRIDES,
                       real_layout);
forward_plan.set_value(oneapi::mkl::dft::config_param::OUTPUT_STRIDES,
                       complex_layout);
forward_plan.commit(Q);

auto evt1 = oneapi::mkl::dft::compute_forward(forward_plan, img1);
auto evt2 = oneapi::mkl::dft::compute_forward(forward_plan, img2);


oneapi::mkl::vm::mulbyconj(Q, n_rows * c_stride_h,
                           reinterpret_cast<std::complex<float>*>(img1),
                           reinterpret_cast<std::complex<float>*>(img2),
                           reinterpret_cast<std::complex<float>*>(corr),
                           {evt1, evt2}).wait();


oneapi::mkl::dft::descriptor<oneapi::mkl::dft::precision::SINGLE,
                             oneapi::mkl::dft::domain::REAL>
                                      backward_plan({n_rows, n_cols});

// Data layout in conjugate-even domain
backward_plan.set_value(oneapi::mkl::dft::config_param::INPUT_STRIDES,
                        complex_layout);

// Data layout in real domain
backward_plan.set_value(oneapi::mkl::dft::config_param::OUTPUT_STRIDES,
                        real_layout);
backward_plan.commit(Q);

auto bwd = oneapi::mkl::dft::compute_backward(backward_plan, corr);
bwd.wait();


auto policy = oneapi::dpl::execution::make_device_policy(Q);
auto maxloc = oneapi::dpl::max_element(policy,
                              corr,
                              corr + (n_rows * n_cols * 2 + 2));

auto s = oneapi::dpl::distance(corr, maxloc);
float max_corr = corr[s];
int x_shift = s % (n_cols + 2);
int y_shift = s / (n_rows + 2);
```

## ArrayFire

```
af::setDevice(0);

auto img1 = af::constant(0.0,
                         n_rows,
                         n_cols,
                         af::dtype::f32);
auto img2 = af::constant(0.0,
                         n_rows,
                         n_cols,
                         af::dtype::f32);
auto corr = af::constant(0.0,
                         n_rows,
                         n_cols,
                         af::dtype::f32);

img1(af::seq(4, 5), af::seq(5, 6)) = 1.0f;
img2(af::seq(1, 2), af::seq(1, 2)) = 1.0f;

img1 = af::fftR2C<2>(img1, 0.0);
img2 = af::fftR2C<2>(img2, 0.0);
corr = img1 * af::conjg(img2);
corr = af::fftC2R<2>(corr, 0.0);
af::array max_score, shift;
af::max(max_score, shift, af::flat(corr));

auto max_corr = max_score.scalar<float>();
auto s = shift.scalar<unsigned>();
int x_shift = s / n_cols;
int y_shift = s % n_rows;
```

From Intel's Parallel Universe

Source:
https://www.intel.com/content/www/us/en/developer/articles/technical/accelerate-2d-fourier-correlation-algorithm.html

# Library Components

- Hand-optimized oneAPI/OpenCL/CUDA kernels
- External libraries(BLAS, FFT, Sparse, etc.)
- ArrayFire’s JIT Engine
- Memory Manager
- Device/Library State management
- Interop Functions

# Core data structure - **af::array**

- Multi-dimensional data array(up to 4D)
- Supports all standard data types
  - complex and real 32, 64; integers 8, 16, 32, 64
- Dense and sparse support

# ArrayFire Functions

reductions
- sum, min, max, count, prod
- vectors, columns, rows, etc

dense linear algebra
- LU, QR, Cholesky, SVD, Eigenvalues, Inversion, Solvers, Determinant, Matrix Power

convolutions
- 2D, 3D, ND

FFTs
- 2D, 3D, ND

image processing
- filter, rotate, erode, dilate, morph, resize, rgb2gray, histograms

interpolate & scale
- vectors, matrices
- rescaling

sorting
- along any dimension
- sort detection

and many more…

# Function calls

- All function calls are asynchronous
- Calls to complex functions are queued immediately
- Simple arithmetic calls are evaluated, and an AST node is created

# Basic usage

```
float b_ptr[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
array b(1, 10, b_ptr);
b(seq(3));          //  {0,1,2}
b(seq(1, 7));       //  {1,2,3,4,5,6,7}
b(seq(1, 2, 7));    //  {1,3,5,7}
b(seq(0, 2, end)); //  {0,2,4,6,8}
```

# Assignment

```
// setting entries to a constant
A(span) = 4;          // fill entire array
A.row(0) = -1;        // first row
A(seq(3)) = 3.1415; // first three elements
```

# Arithmetic

```
array R = randu(3,3);
array C = constant(1,3,3) + complex(sin(R));  // C is c32

// rescale complex values to unit circle
array a = randn(5,c32);
af_print(a / abs(a));
```

# Allocations

- Data and random number arrays are allocated immediately
- Constant, and results from element-wise operations are not allocated until necessary
- Tiling operations are also not allocated

# ArrayFire JIT

- Basic Arithmetic functions are evaluated but not executed
- ArrayFire builds a operation dependency tree based on the function calls
- Creates, compiles and links a kernel and executes them at runtime
- Kernels are cached in memory and on disk

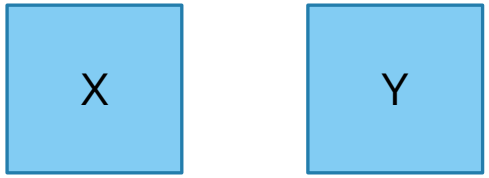
# Why Just-In-Time Kernel Generation?

- There is driver overhead associated with kernel calls
- Access to global memory is expensive
- Temporary values require unnecessary allocation

# Example

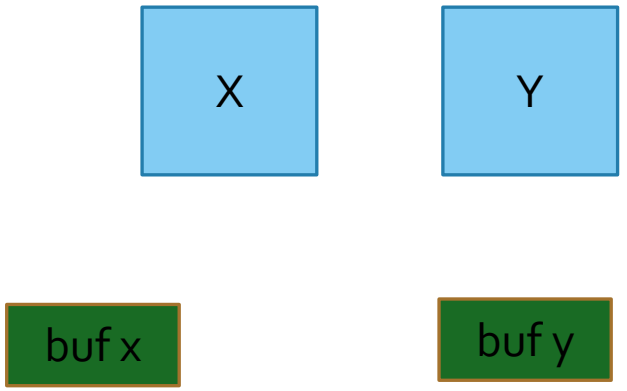
```
int main() {
    // 20 million random samples
    int n = 20e6;
    array x = randu(n,1), y = randu(n,1);
    // how many fell inside unit circle?
    float pi = 4 * sum<float>(x*x + y*y < 1) / n;
    printf("pi = %g\n", pi);
    return 0;
}
```

# Example




```
int main() {
    // 20 million random samples
    int n = 20e6;
    array x = randu(n,1), y = randu(n,1);
    // how many fell inside unit circle?
    float pi = 4 * sum<float>(x*x + y*y < 1) / n;
    printf("pi = %g\n", pi);
    return 0;
}
```

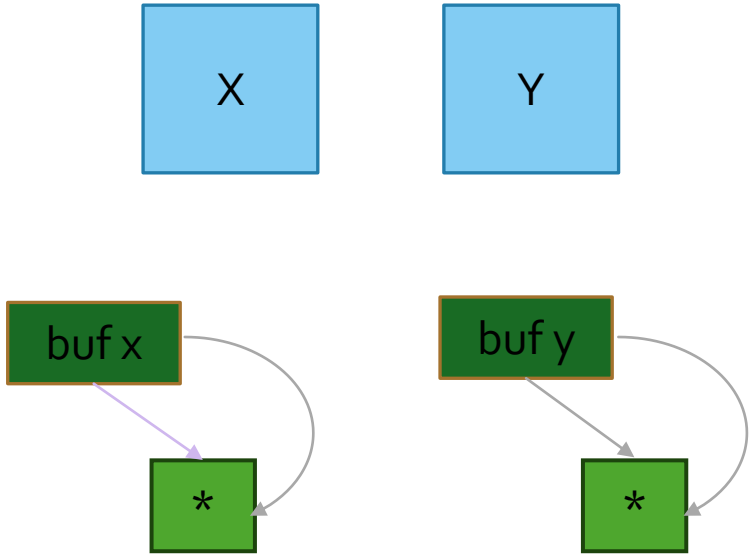
# Example




```
int main() {
    // 20 million random samples
    int n = 20e6;
    array x = randu(n,1), y = randu(n,1);
    // how many fell inside unit circle?
    float pi = 4 * sum<float>(x*x + y*y < 1) / n;
    printf("pi = %g\n", pi);
    return 0;
}
```

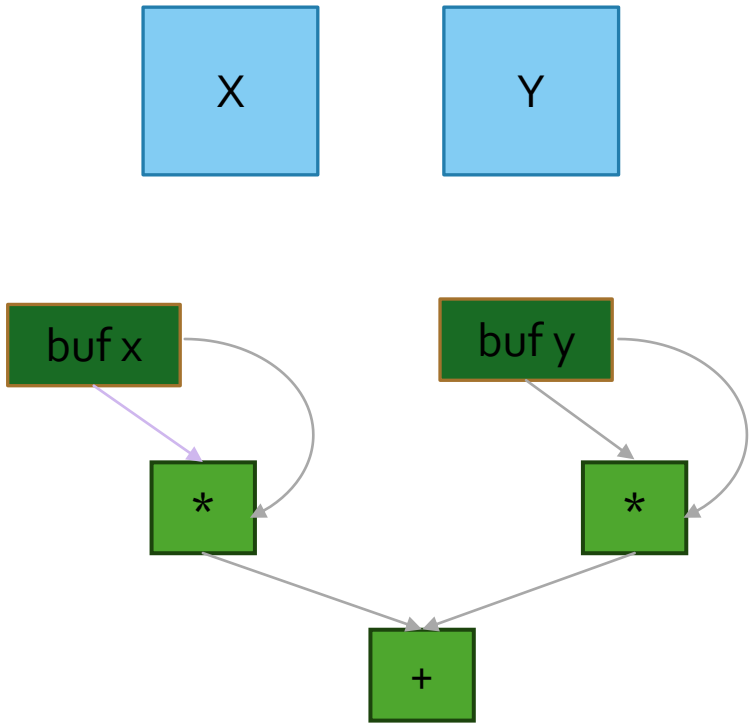
# Example

```
int main() {
    // 20 million random samples
    int n = 20e6;
    array x = randu(n,1), y = randu(n,1);
    // how many fell inside unit circle?
    float pi = 4 * sum<float>(x*x + y*y < 1) / n;
    printf("pi = %g\n", pi);
    return 0;
}
```



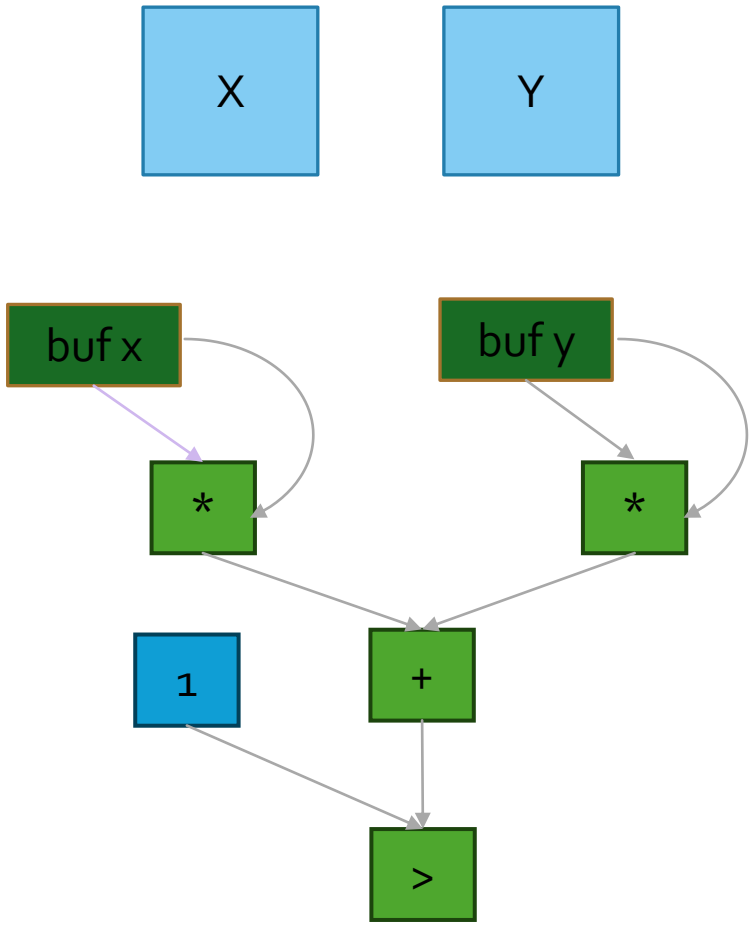
# Example

```
int main() {
    // 20 million random samples
    int n = 20e6;
    array x = randu(n,1), y = randu(n,1);
    // how many fell inside unit circle?
    float pi = 4 * sum<float>(x*x + y*y < 1) / n;
    printf("pi = %g\n", pi);
    return 0;
}
```



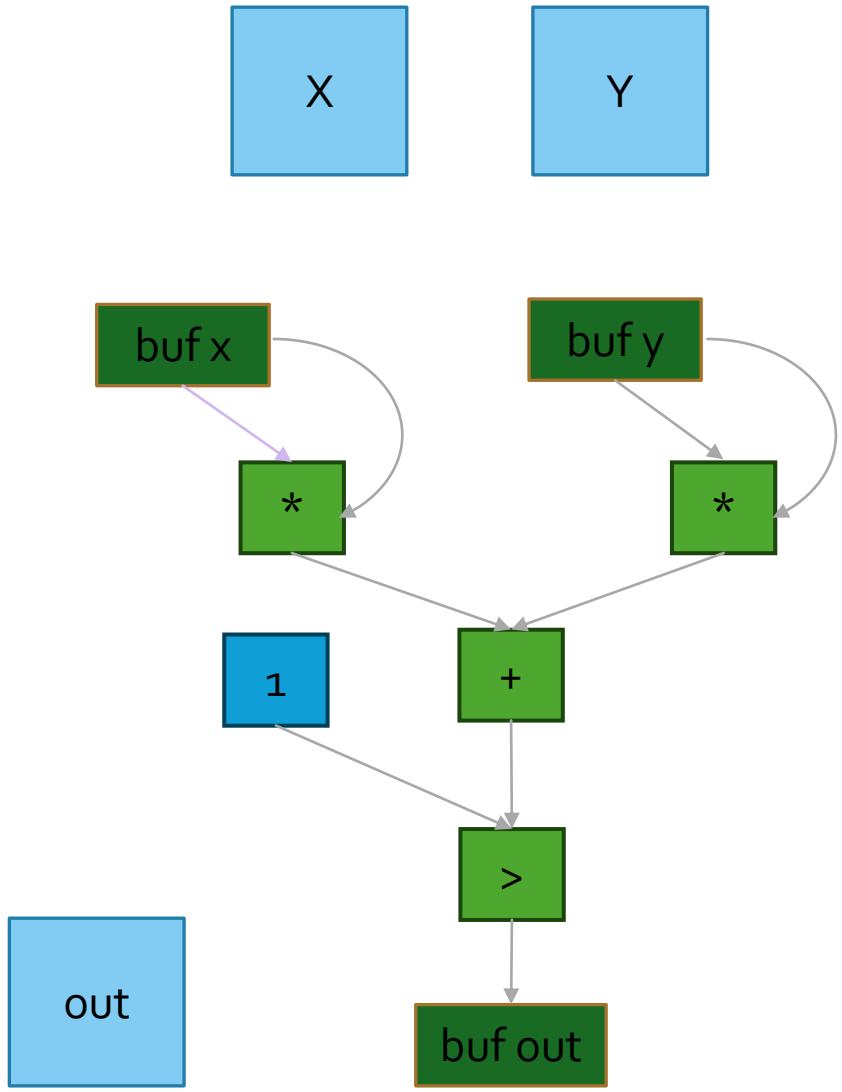
# Example

```
int main() {
    // 20 million random samples
    int n = 20e6;
    array x = randu(n,1), y = randu(n,1);
    // how many fell inside unit circle?
    float pi = 4 * sum<float>(x*x + y*y < 1) / n;
    printf("pi = %g\n", pi);
    return 0;
}
```



# Example

```
int main() {
    // 20 million random samples
    int n = 20e6;
    array x = randu(n,1), y = randu(n,1);
    // how many fell inside unit circle?
    float pi = 4 * sum<float>(x*x + y*y < 1) / n;
    printf("pi = %g\n", pi);
    return 0;
}
```



# Example

```
__kernel void KER2482264519866675467(
__global float *in0, dim_t iInfo0_offset,
__global float *in2, dim_t iInfo2_offset,
float scalar5,
__global char *out0, int offset0,
KParam oInfo){
   int idx = get_global_id(0);
   const int idxEnd = oInfo.dims[0];
   if (idx < idxEnd) {
in0 += iInfo0_offset;
#define idx0 idx
in2 += iInfo2_offset;
#define idx2 idx

out0 += offset0;


float val0 = in0[idx0];
float val1 = __mul(val0, val0);
float val2 = in2[idx2];
float val3 = __mul(val2, val2);
float val4 = __add(val1, val3);
float val5 = scalar5;
char val6 = __lt(val4, val5);
out0[idx] = val6;

   }
}
```

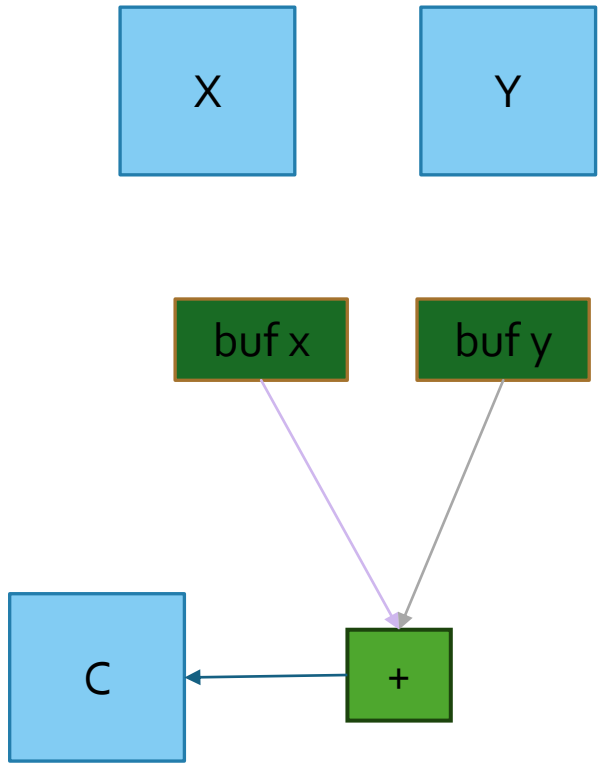
# JIT evaluation criteria

- JIT kernels are generated whenever the result is necessary
- Kernel parameter size limitations
- Memory pressure is also considered to evaluate an array

# JIT evaluation criteria

- JIT kernels are generated whenever the result is necessary
- Kernel parameter size limitations
- Memory pressure is also considered to evaluate an array

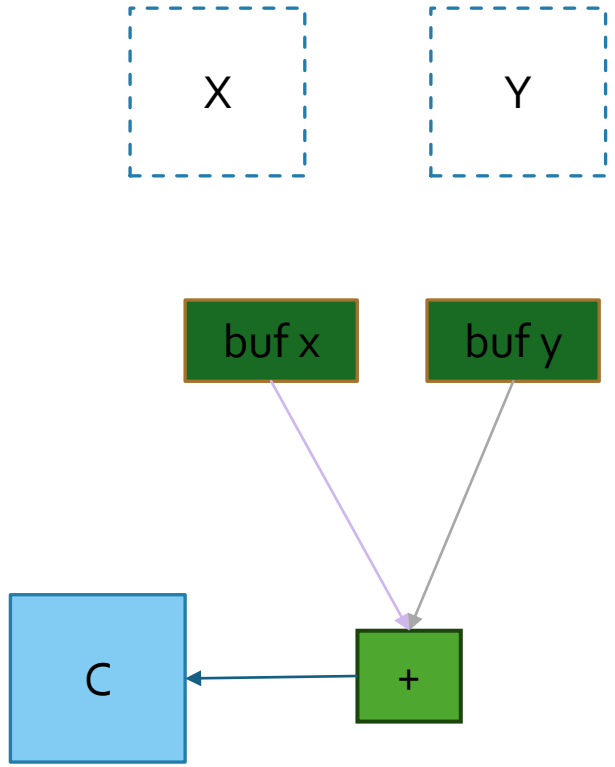
```
array C;
{
   array X = randu(100, 100);
   array Y = randu(100, 100);
  C = X + Y;
}
???
```



# JIT evaluation criteria

- JIT kernels are generated whenever the result is necessary
- Kernel parameter size limitations
- Memory pressure is also considered to evaluate an array

```
array C;
{
   array X = randu(100, 100);
   array Y = randu(100, 100);
  C = X + Y;
}
???
```



# Memory manager

- Allocation and deallocation events typically require synchronization
- Synchronization events increase driver/runtime overhead
- Memory is allocated using bins of 1KB sizes
- Multiple iterations can reuse the same buffer as long as they are in the same device/queue. Even if the previous operation have not completed.

# Case Studies

FLASHLIGHT: ENABLING INNOVATION IN TOOLS FOR MACHINE LEARNING

A PREPRINT


**Jacob Kahn**
Facebook AI Research
Menlo Park, CA
jacobkahn@fb.com

**Vineel Pratap**
Facebook AI Research
Menlo Park, CA
vineelkpratap@fb.com

**Tatiana Likhomanenko**
Facebook AI Research*
Menlo Park, CA
antares@fb.com

**Qiantong Xu**
Facebook AI Research†
Menlo Park, CA
qiantong@fb.com

**Awni Hannun**
Zoom AI
San Jose, CA
awni.hannun@zoom.us

**Jeff Cai**
Facebook AI Research‡
Menlo Park, CA
jcai@fb.com

**Paden Tomasello**
Facebook AI Research
Menlo Park, CA
padentomasello@fb.com

**Ann Lee**
Facebook AI Research
New York, NY
annl@fb.com

**Edouard Grave**
Facebook AI Research
Paris
egrabve@fb.com

**Gilad Avidov**
Facebook
Menlo Park, CA
avidov@fb.com

**Benoit Steiner**
Facebook AI Research
Menlo Park, CA
benoitsteiner@fb.com

**Vitaliy Liptchinsky**
Facebook AI Research
Menlo Park, CA
vitaliy888@fb.com

**Gabriel Synnaeve**
Facebook AI Research
Paris
gab@fb.com

**Ronan Collobert**
Facebook AI Research§
Menlo Park, CA
locronan@fb.com


| METRIC | PYTORCH | TENSORFLOW | (OURS) FLASHLIGHT |
|---|---|---|---|
| BINARY SIZE (MB) | 527 | 768 | 10 |
| LINES OF CODE | 1,798,292 | 1,306,159 | 27,173 |
| NUMBER OF OPERATORS | 2,166 | 1,423 | 60 |
| **APPROX NUM. OPS. THAT PERFORM:** | | | |
| ADD | 55 | 20 | 1 |
| CONV | 85 | 30 | 2 |
| SUM | 25 | 10 | 1 |

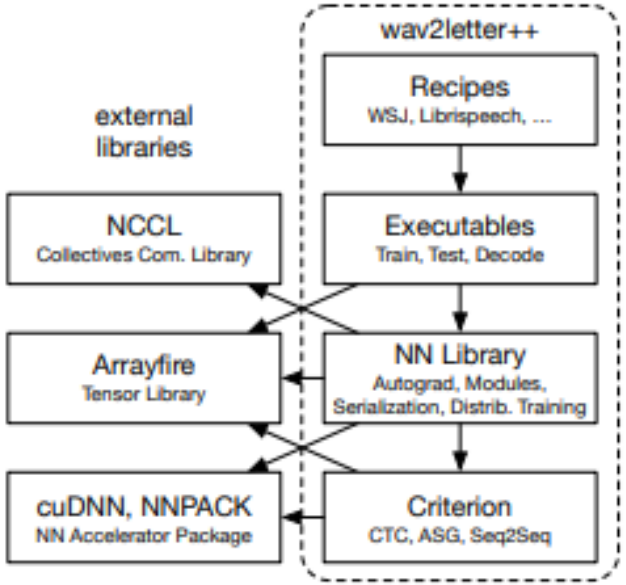
| MODEL | NUM. PARAMS (M) | BATCH SIZE | 1 GPU | | | 8 GPUS | | |
|---|---|---|---|---|---|---|---|---|
| | | | PT | TF | FL | PT | TF | FL |
| ALEXNET | 61 | 32 | 2.0 | 4.0 | **1.4** | 6.0 | 6.5 | **2.1** |
| VGG16 | 138 | 32 | 14.8 | **12.6** | 13.2 | 16.3 | 17.9 | **14.9** |
| RESNET-50 | 25 | 32 | 11.1 | 12.4 | **10.3** | 12.3 | 15.9 | **11.9** |
| BERT-LIKE | 406 | 128 | 19.6 | 19.8 | **17.5** | 22.7 | 23.6 | **19.2** |
| ASR TR. | 263 | 10 | 58.5 | 63.7 | **53.6** | 63.7 | 69.7 | **57.5** |
| VIT | 87 | 128 | 137.8 | 140.3 | **129.3** | 143.1 | 169.6 | **141.0** |

# Case Studies

**WAV2LETTER++: THE FASTEST OPEN-SOURCE SPEECH RECOGNITION SYSTEM**

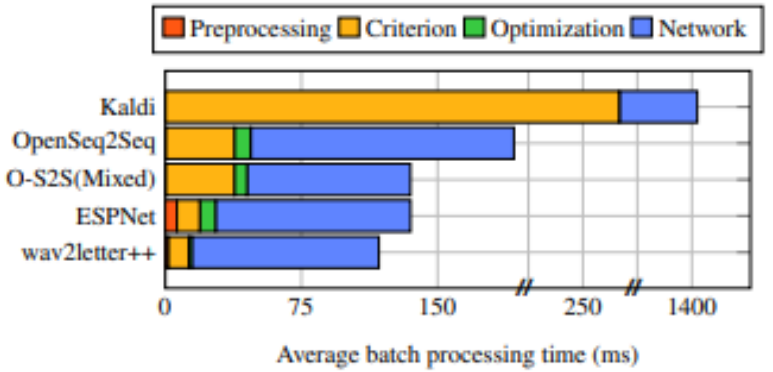
*Vineel Pratap, Awni Hannun, Qiantong Xu, Jeff Cai, Jacob Kahn, Gabriel Synnaeve, Vitaliy Liptchinsky, Ronan Collobert*

Facebook AI Research



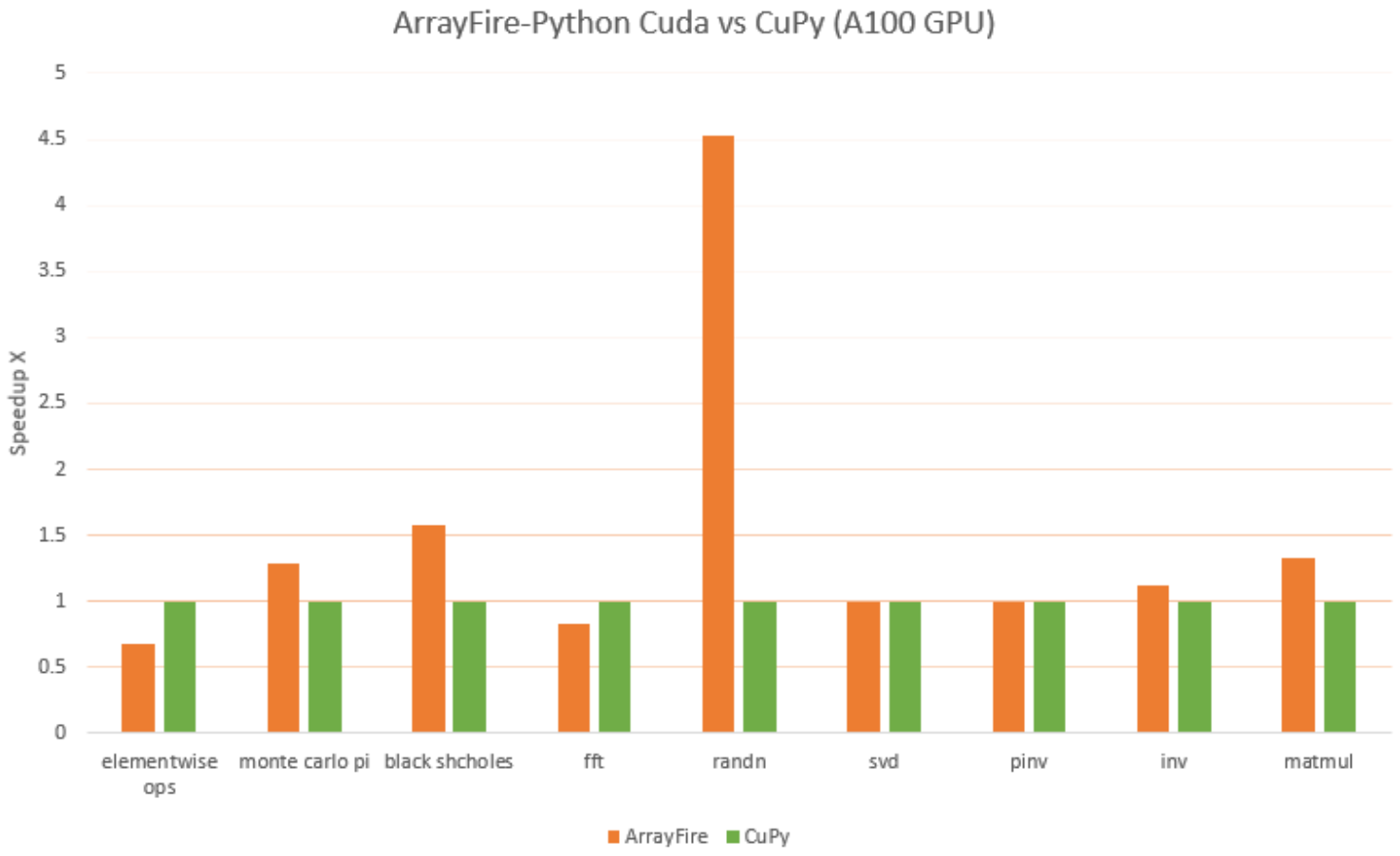

| Name | Language | Model(s) | ML Syst. |
|---|---|---|---|
| Kaldi | C++, Bash | HMM/GMM | - |
| | | DNN/LF-MMI | - |
| ESPNet | Python, Bash | CTC, seq2seq, hybrid | PyTorch, Chainer |
| OpenSeq2Seq | Python, C++ | CTC, seq2seq | TensorFlow |
| wav2letter++ | C++ | CTC, seq2seq, ASG | ArrayFire |

**Table 1**. Major open-source speech recognition systems.



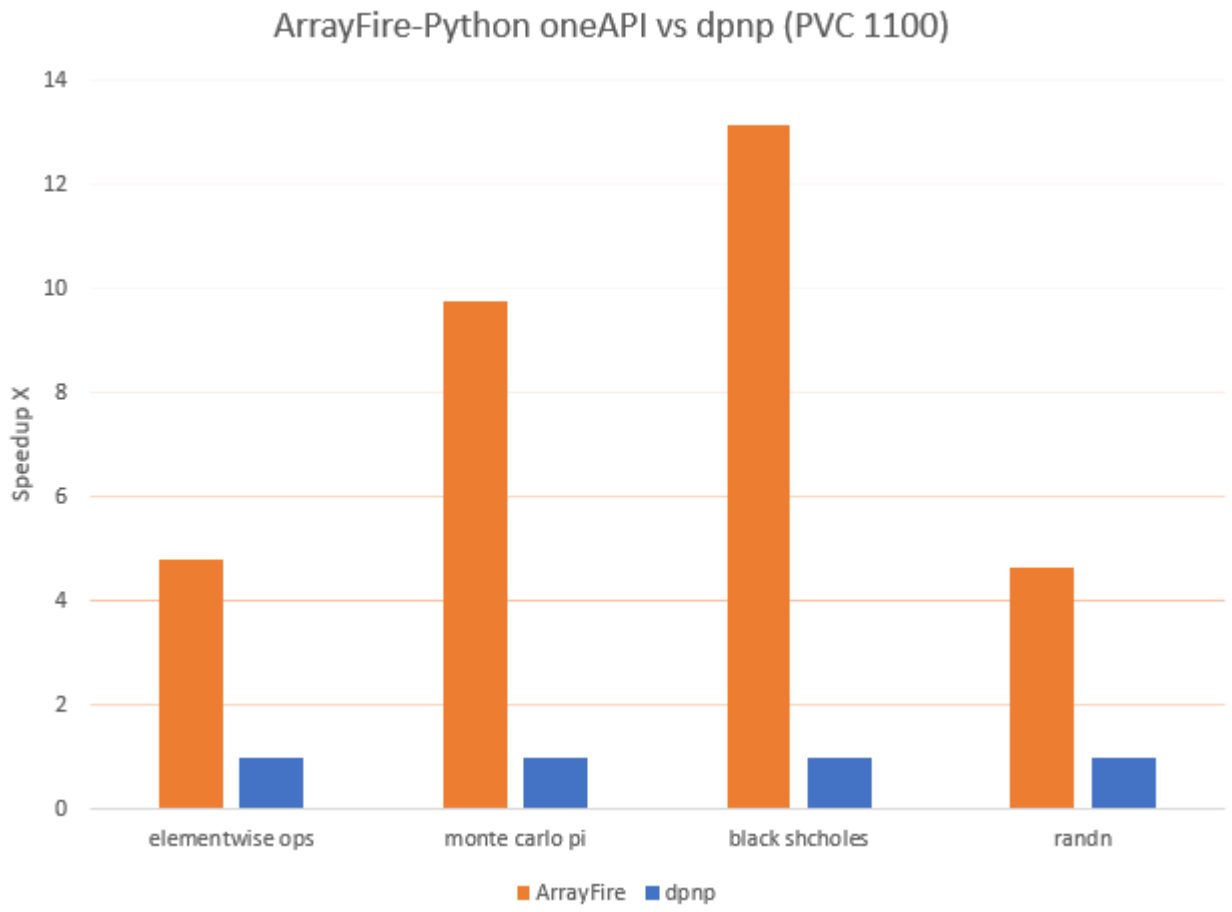
# Python Performance

- Same ArrayFire through python
- Exposes OneAPI, OpenCL, CUDA through python
- Competitive with CuPy
- Opts into new **array_api** standard (partial support)



# JIT + Python Performance

- ArrayFire JIT implementation in oneAPI offers competitive advantage
- JIT for level 0 possible, but WIP
- CUDA JIT in OneAPI possible, would replace CUDA backend!
- native sycl JIT enabled by [CMPLRLLVM-32537]



# Two Example Projects from 2023

## Utilising LSTM Based Network for Image Forgery Detection

[1]Arunima Jaiswal, [2]Aradhana Parmar, and [3]Nitin Sachdeva

[1,2]Department of Computer Science and Engineering; Indira Gandhi Delhi Technical University for Women, India
[3]IT Department, Galgotias College of Engineering and Technology, Greater Noida, India

E-mail : arunimajaiswal@igdtuw.ac.in, aradhana004mtcse21@igdtuw.ac.in, nitin.sachdeva@galgotiacollege.edu

**Abstract- Image Forgery refers to doing various kinds of modifications to the original image such that the actual meaning of it gets changed, due to which false information is spread everywhere. Moreover in this era where social media is at huge heights and images have become one of the most integral part of everyones life ,people tend to believe what they see and hence it's a very serous concern which needs a solution. There are many techniques present to detect image forgeries .In our research we have shown two major techniques to detect forgeries using resampling techniques. As Resampling is a key indicator of altered images, two approaches for detecting and localising picture modifications, employing resampling (by forming different samples) varied integral mechanisms. First approach, the Radon convert of these features is computed on all those image parts which are overlapping to each other, then a heatmap generated using classifiers of deep learning appproach and also gaussian random variable ,which is model for conditional probability values . To locate tampered sections, the Random Walker segmentation approach is utilised. For categorisation and identification of local tampered regions, the second method includes transmitting resampling characteristics computed on intersecting picture**
**network. We hav**
**detection/localizatio**
**both strategies are e**
**image forgeries of a**
**according to the resu**

***Keywords — Machi***
***Networks, Image***
***Forgery***

functionalities, smartphones with highly enhanced features, and tablets.Social media networks like Instagram, Twitter, Snapchat and Facebook have also helped to spread them [1]. Similarly, methods for digitally editing these photographs have advanced dramatically, with software like Photoshop and Gimp, as well as smartphone applications like Snapseed and Pixlr, making image modification extremely simple for users [2].

Several approaches for identifying digital picture alterations have been proposed, including resampling artefacts, colour filter arrays, lighting, camera forensics, JPEG compression, and many others.All these approaches follow a framework which begins with patch identification ,then pixel identification and then other types of changes . Artificial Intelligence gave a huge set of algorithms to test and work upon the mechanism and detect the final output.Many type of forgeries can be detected using this approach.

In this research, we look at artefacts generated by typical when doing Both parameters ), mage splicing. We be detected and pproaches [3]. We y and localise these oose a end-to-end nsform and Deep And for our second atches, we combine m probability maps

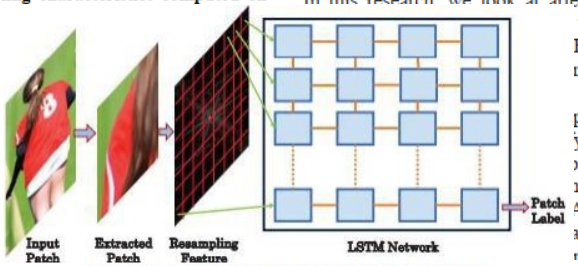



**Figure 8**: LSTM Framework for Patch Classification

## A GPU-based real-time processing system for frequency division multiple-input-multiple-output radar

Gaogao Liu | Yuqian Bao | Ning Yue | Sitian Wang | Hui Wu | Qiang Liu | Wenbo Yang

School of Electronic and Engineering, Xidian University, Xi'an, China

Correspondence
Gaogao Liu.
Email: ggliu@xidian.edu.cn

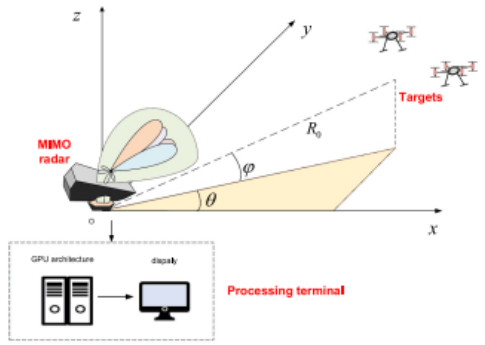



**FIGURE 1** Frequency division MIMO radar detection scenario.

**Abstract**

Multiple-Input-Multiple-Output (MIMO) radar has the characteristic of multiple antenna channels, bringing on a big data volume for signal processing. Therefore, the trade-off between detection ability and computational efficiency is always considered. In this article, an optimisation system is proposed to enhance the real-time performance of frequency division MIMO radar without compromising accuracy. For the scenario of low-altitude small target detection, a signal processing acceleration method is proposed and a MIMO radar optimisation system based on graphics processing unit (GPU) architecture is built. The signal model of frequency division MIMO radar is first established, improving the classical signal processing flow efficiency from the perspective of reducing fast Fourier transform (FFT) times and windowing operation times. To achieve an advanced acceleration, the parallel architecture of ArrayFire-library in GPU is then employed. Distinct minimum parallel units are extracted by analysing the principle of digital beamforming (DBF), pulse compression, moving target detection (MTD), and constant false-alarm rate (CFAR) algorithms. And the corresponding parallel algorithms are designed to constitute a parallel acceleration system of frequency division MIMO. Simulation results indicate that the proposed method significantly improves the efficiency of MIMO system with a maximum acceleration ratio of 65 times, meeting the real-time processing requirements.

**KEYWORDS**
array signal processing, frequency-domain analysis, MIMO radar, optimisation, parallel algorithms

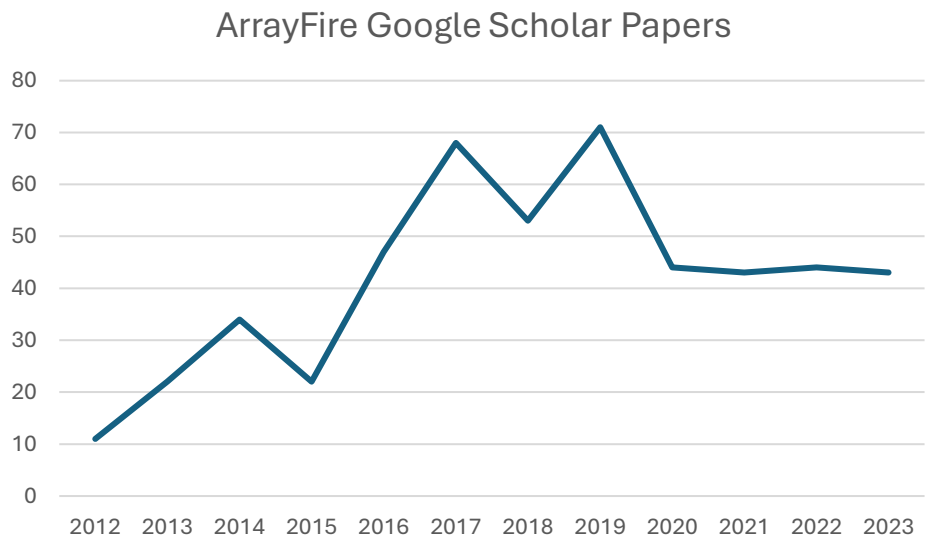
**TABLE 8** Runtime with different data structure.

| Data structure | Matlab/s | Matlab (improved)/s | ArrayFire/s | Acceleration ratio[a] |
|---|---|---|---|---|
| 8000 × 32 × 4 × 64 | 19.7226 | 15.0831 | 0.3719 | 53.03 |
| 8000 × 64 × 4 × 64 | 37.6517 | 22.3699 | 0.7101 | 53.02 |
| 8000 × 128 × 4 × 64 | 74.8637 | 40.1178 | 1.2966 | 57.74 |
| 8000 × 256 × 4 × 64 | 152.9341 | 78.1676 | 2.3407 | 65.34 |

[a]Acceleration ratio equals the running time on Matlab divided by the running time on ArrayFire.

# Google Scholar Hits

- Google Scholar hits in 2023
  - 141 for oneMKL
  - 110 for oneDNN
  - 43 for ArrayFire
  - 11 for oneDAL



# Conclusion

- Flexible library with a hybrid approach to GPU compute
- Allows lazy evaluation and allocation of data
- Hundreds of functions
- Free and Open Source under the BSD3 License

# Backup Slides

# Ongoing oneAPI Tasks in ArrayFire to Stay Competitive

- Architecture-specific optimizations as new devices emerge
- Support for new oneAPI releases and new library functions
- Language wrapper maintenance and improvements
- Binary distribution and package manager maintenance
- Benchmarking suite maintenance (used by Phoronix)

# On-deck Roadmap Tasks

- Upgrade CPU-backend performance, including LLVM/MLIR
- Optimize data transfers and concurrent queue execution
- Better integration between the CPU-backend and Device-backends, improving the performance of smaller datasets
- Integration of distributed workloads