Numba: Expanding Capabilities to Power a New Generation of Python Libraries

Siu Kwan Lam, Sr Software Engineer, Anaconda

What is Numba?

Numba is a just-in-time type-specializing function compiler

Demo

“What is Numba” demo key points

- **@jit** and **@njit** decorator
- **nopython=True** enforces native code compilation
- **type specialization** optimizes based on types
- **SIMD-vector** instructions

Slow to fast loops

- Does not support all python syntax
- Optimizes loops and array ops best
- “FORTRAN-style python”

Demo

“Slow to fast loop” demo key points

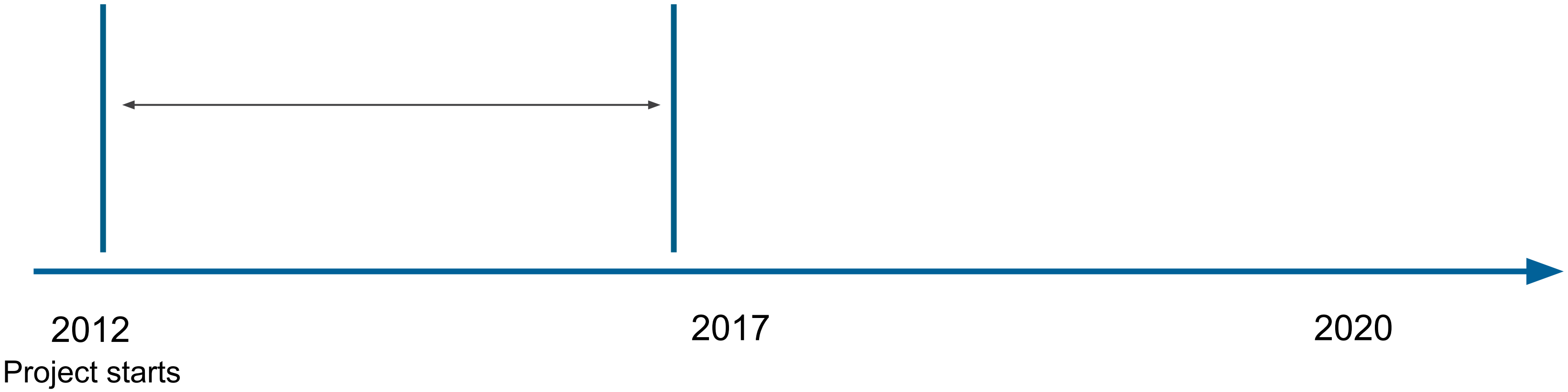
- Pure-python loop is inefficient
- LLVM enables loop-unroll, loop-vectorization
- Intel SVML
  - LLVM patch by Intel
  - `conda install -c numba icc_rt`
- **parallel=True** for auto-parallelization
  - Contributed by Intel

Expanding capabilities

Array computing era

Key features:

- @jit, @njit
- @vectorize, @guvectorize
  - cpu/gpu portable ufunc
- @cuda.jit, @rocm.jit
  - low level GPU kernel



Auto-parallelization era

In 2017 IntelLab contributed

ParallelAccelerator

@jit(parallel=True)

@stencil

prange

2012
Project starts

2017

2020

Library extension era

Since 2018

New extension libraries

- defines new containers

General purpose features

- list, dictionary, unicode
- try-except

Extensible compiler

2012
Project starts

2017

2020

OSS Projects that extend Numba

Awkward 1.0

https://github.com/scikit-hep/awkward-1.0

“one of the most widely pip-installed packages for particle physics”

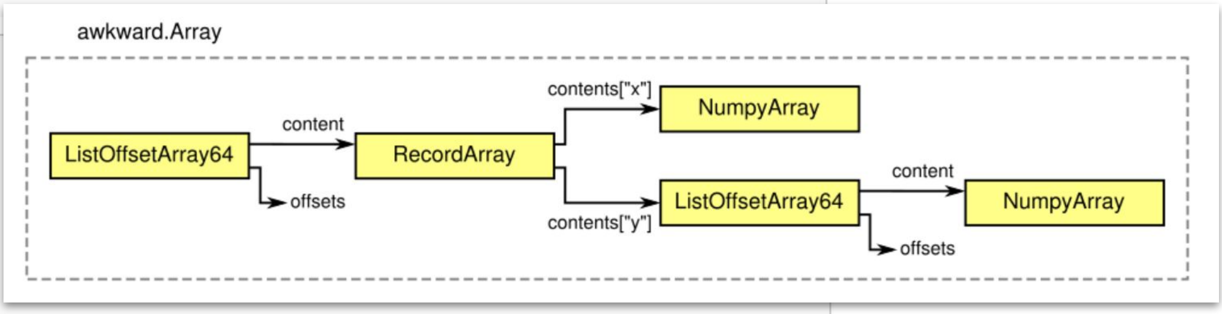
```python
import awkward1 as ak
```

```python
import numba as nb

@nb.jit(nopython=True)
def run(array):
    out = np.empty(len(array), np.float64)
    for i in range(len(array)):
        out[i] = array[i]["x"]
        for y in array[i]["y"]:
            out[i] += y
    return out

akarray = ak.Array([{"x": 100, "y": [1.1, 2.2]}, {"x": 200, "y": []}, {"x": 300, "y": [3.3]}])

# Works for the layout nodes, but not the high-level ak.Array wrapper yet.
run(akarray.layout)
```



reference: https://github.com/scikit-hep/awkward-1.0/blob/master/docs-jupyter/2020-01-22-numba-demo-EVALUATED.ipynb

Intel Scalable DataFrame Container (SDC)

```
import pandas as pd
from numba import njit, prange

# Dataset for analysis
FNAME = "employees.csv"


# This function gets compiled by Numba* and multi-threaded
@njit(parallel=True)
def get_analyzed_data():
    df = pd.read_csv(FNAME)
    s_bonus = pd.Series(df['Bonus %'])
    s_first_name = pd.Series(df['First Name'])

    # Use explicit loop to compute the mean. It will be compiled as parallel loop
    m = 0.0
    for i in prange(s_bonus.size):
        m += s_bonus.values[i]
    m /= s_bonus.size

    names = s_first_name.sort_values()
    return m, names


# Printing names and their average bonus percent
mean_bonus, sorted_first_names = get_analyzed_data()
print(sorted_first_names)
print('Average Bonus %:', mean_bonus)
```

```
$ python ./basic_workflow_parallel.py
7       ALEXANDER
4     CHRISTOPHER
0           EMILY
2           ISAAC
8          JOSEPH
9          JOSEPH
5             MIA
1            NOAH
6          OLIVIA
3             NaN
dtype: object
Average Bonus %: 11.204399999999998
```

Reference “Basic workflow in parallel” example:
https://intelpython.github.io/sdc-doc/latest/examples.html

Intel SDC Pipeline

- Extends Numba's pipeline
- Define custom passes

```
class SDCPipeline(numba.core.compiler.CompilerBase):
    """SDC compiler pipeline
    """

    def define_pipelines(self):
        name = 'sdc_extention_pipeline_distributed'
        pm = DefaultPassBuilder.define_nopython_pipeline(self.state)

        add_pass_before(pm, InlinePass, InlineClosureLikes)
        pm.add_pass_after(HiFramesPass, InlinePass)
        pm.add_pass_after(DataFramePass, AnnotateTypes)
        pm.add_pass_after(PostprocessorPass, AnnotateTypes)
        pm.add_pass_after(HiFramesTypedPass, DataFramePass)
        pm.add_pass_after(DistributedPass, ParforPass)
        pm.finalize()

        return [pm]


@register_pass(mutates_CFG=True, analysis_only=False)
class ParforSeqPass(FunctionPass):
    _name = "sdc_extention_parfor_seq_pass"

    def __init__(self):
        pass

    def run_pass(self, state):
        numba.parfors.parfor.lower_parfor_sequential(
            state.typingctx, state.func_ir, state.typemap, state.calltypes)

        return True
```

Extending with @numba.extending.overload

```
@overload(hq.heappush)
def heappush(heap, item):
    assert_heap_type(heap)
    assert_item_type_consistent_with_heap_type(heap, item)

    def hq_heappush_impl(heap, item):
        heap.append(item)
        _siftdown(heap, 0, len(heap) - 1)

    return hq_heappush_impl
```

Typing logic

Codegen logic

Auto-discoverable extensions

- Using setuptools entry points

```
setup(name = "awkward1",
...
        entry_points = {
          "numba_extensions": ["init = awkward1._connect._numba:register"]
        },
```

from https://github.com/scikit-hep/awkward-1.0/blob/master/setup.py

Conclusion

Why libraries are using Numba?

- Native code performance
    - Without rewriting in pre-compiled languages
- Auto-parallelization
    - Leverage multicores CPU with ease
- Runtime codegen → easy to distribute
    - No need to ship binaries for every platform
- From numerical computing to data-analytics

Supported Platforms

- X86: Windows, Linux, Mac (64-bit only)
- ARMv7, AARCH64, PPC64: Linux
- Python >= 3.6
- GPU: CUDA, ROCM
- Distribute wheels and conda packages

Resources and examples

Demo notebooks (per release since v0.41):

https://github.com/numba/numba-examples/tree/master/notebooks

Doc & mailing list: http://numba.pydata.org/

Github: https://github.com/numba/numba

Gitter: https://gitter.im/numba/numba

Feel free to ask general questions on mailing list or Gitter, and open Github issues on specific problems.

Thank You!

Siu Kwan Lam
Github: @sklam