Parallel Computing using Numba: A High Performance Python Compiler

MTAT.08.020
Lecture - 11

Tek Raj Chhetri

tekrajchhetri@gmail.com,
tek.raj.Chhetri@ut.ee
Compiler

- User writes a program
- Uses compiler / interpreter based on language program is written to generate executable instructions.

Fig: Running a Program
Role of Compiler

• Translation of user program to machine understandable instructions
• Error checking and Syntax validation, code optimization
• It can also help boost performance - Numba

Fig: Running a Program
Numba

• An open source JIT compiler based on LLVM [2].
• Translates Python and NumPy code to almost speed of C or FORTRAN [2].

Figure: LLVM Design Overview [1]
LLVM – Low Level Virtual Machine

- Designed to support transparent, lifelong program analysis and transformation [3]
- Life long – throughout the duration of the program run
  - Inter-procedural optimizations in link time, machine-dependent optimizations at install time on each system, dynamic optimization at runtime, and profile-guided optimization between runs ("idle time") using profile information collected from the end-user [3].
- Low level code representation in Static Single Assignment (SSA)
  - language independent type system [3]
LLVM – Low Level Virtual Machine

Features:

• Persistent program information – preserve LLVM state throughout application lifetime [3].

• Offline code generation - possible to compile programs into efficient native machine code offline [3].

• User-based profiling and optimization – take both runtime and idle into account [3]

• Transparent runtime model – allows any language to compile with it.

• Uniform, whole-program compilation – due to language independent possible to compile and optimize all code in uniform manner [3]

Fig: LLVM system architecture diagram [3]
Why Numba?

• Python with almost speed of C [2].
• Based on open source technology – LLVM [4]
• Works within the standard Python interpreter and Integrates tightly with NumPy [6]
• Easy to use @decorator
Why Numba?

• Compatible with both multithreaded and distributed computing paradigms [5, 6]

• Vectorize array processing with @vectorize decorator – similar to ufuncs in numpy [5, 8] – example in Google colab notebook.

• Ability to extend [6, 7, 9].

• Non CPU hardware – GPU support[1,6,10]

• Numba's CPU detection will enable LLVM to autovectorize for appropriate SIMD instruction set [6,10]:
  • SSE, AVX, AVX2, AVX-512
Why Numba?

• Compatible with both multithreaded and distributed computing paradigms [5, 6]

• Vectorize array processing with @vectorize decorator – similar to ufuncs in numpy [5, 8] – example in Google colab notebook.

• Ability to extend [6, 7, 9].

• Non CPU hardware – GPU support [6,10] and caching[2]

• Numba's CPU detection will enable LLVM to autovectorize for appropriate SIMD instruction set [6,10]:
  • SSE, AVX, AVX2, AVX-512
Why Numba?

- Compile for CPU and GPU at the same time [5].
Using Numba Decorators

```python
@jit
def function():
    # normal task

# compile in non pythonic way
# fully compiled and python interpreter call is completely removed
@jit(nopython=True)
def function():
    # normal task

@jit(parallel=True)
def function():
    # execute in parallel mode

@cuda.jit
def matmul(A, B, C):
    row, col = cuda.grid(2)
    if row < C.shape[0] and col < C.shape[1]:
        tmp = 0.
        for k in range(A.shape[1]):
            tmp += A[row, k] * B[k, col]
        C[row, col] = tmp
```
How does Numba work?

```python
@jit
def do_math(a, b):
...

>>> do_math(x, y)
```

Fig: Numba executing instruction [6]
Numba Performance - Image Processing

Source: Google images
Numba Performance - Image Processing

• Parallel Approach (Numba) -71.8 milliseconds Vs 4.2 minutes (Non Parallel)[11]
Numba Performance - Sorting

- Both serial and parallel algorithms were run in Google Colaboratory environment
- `cat /proc/cpuinfo` – check CPU info
- GPU info issuing Nvidia SMI queries -- `nvidia-smi` [12]
Who uses Numba?

numba in XENON

Christopher Tunnell (1) working with Jelle Aalbers (2)
1. Kavli Institute for Cosmological Physics, U. Chicago
2. Nikhef
DIANA-HEP, April 2018

Accelerating a Spectral Algorithm for Plasma Physics with Python/Numba on GPU

FBPIC: A spectral, quasi-3D, GPU accelerated Particle-In-Cell code

What is XENON?
Liquid XENON dark matter detector instrumented with 248 photomultipliers and 10×10 flash ADCs. We make a world-leading new experiment every few years.

Dask is a flexible library for parallel computing in Python

Manuel Kirchen
Center for Free-Electron Laser Science
University of Hamburg, Germany
manuel.kirchen@desy.de

Rémi Lehe
BELLA Center & Center for Beam Physics,
LBNL, USA
rlehe@lbl.gov
Numba – is everything supported?

• Numba doesn’t support everything like python dictionary.

• Numba works with the following (see official doc for new updates):
  • OS: Windows (32 and 64 bit), OSX and Linux (32 and 64 bit)
  • Architecture: x86, x86_64, ppc64le. Experimental on armv7l, armv8l (aarch64).
  • GPUs: Nvidia CUDA. Experimental on AMD ROC.
  • CPython
  • NumPy 1.10 – latest

• For more see official documentation (some are listed below)
  • http://numba.pydata.org/numba-doc/latest/cuda/cudapysupported.html
  • http://numba.pydata.org/numba-doc/latest/user/parallel.html
References

8. [https://numba.pydata.org/numba-doc/dev/user/vectorize.html](https://numba.pydata.org/numba-doc/dev/user/vectorize.html)
11. Tek Raj Chhetri. Analysis of parallel computing, its necessity and the challenges, 2019
Thank You!