Parallel Programming with Parsl

Douglas Friedel (NCSA)

Complex Workflows on HPC Systems

- Can be challenging to manage
- May need to be partly re-written depending on what system you are on
- May not scale well depending on need
- Synchronization between different task is difficult
-
-
- Parsl aims to address these issues

  (https://parsl-project.org)

What is Parsl?

- Flexible and scalable parallelization framework
- Purely written in Python
- Simple to use:
  - `@python_app` decorator for python functions
  - `@bash_app` for anything else
- Extensible from 1 to 8192 compute nodes, up to 0.25 M workers
- Built in interfaces for many systems (AWS, Slurm, Kubernetes, etc.)
- Can be used to implement many different parallel paradigms

What Can Parsl Do?

- Manage large number of parallel and serial tasks
- Synchronize input and output files for each task
  - Tasks with input files are held until all files are available
- Run your code on any system (laptop or HPC) with no changes
  - Just need to change your configuration

```
@python_app
def hello ():
    return 'Hello World!'

print(hello().result())
```

```
Hello World!
```

```
@bash_app
def echo_hello(stdout='echo-hello.stdout'):
    return 'echo "Hello World!"'

echo_hello().result()

with open('echo-hello.stdout', 'r') as f:
    print(f.read())
```

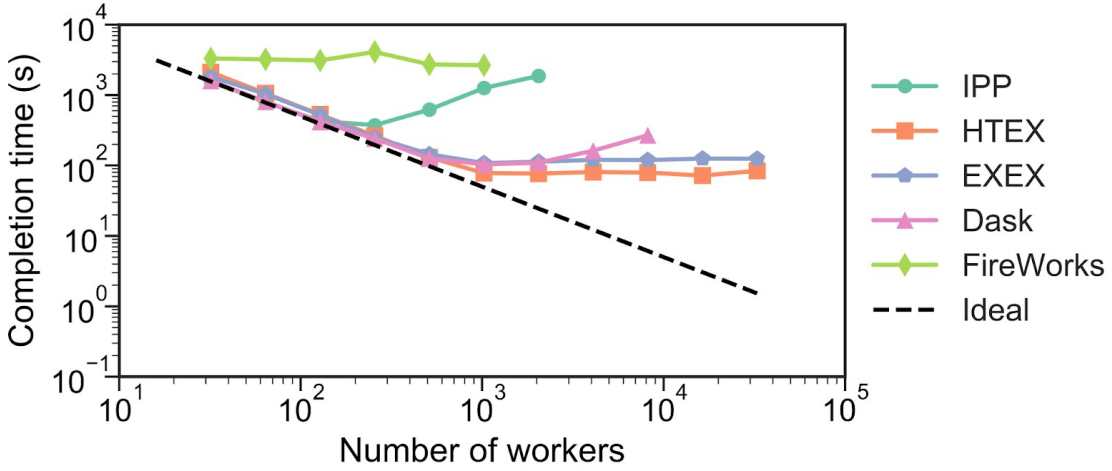
```
Hello World!
```

Parsl Performance

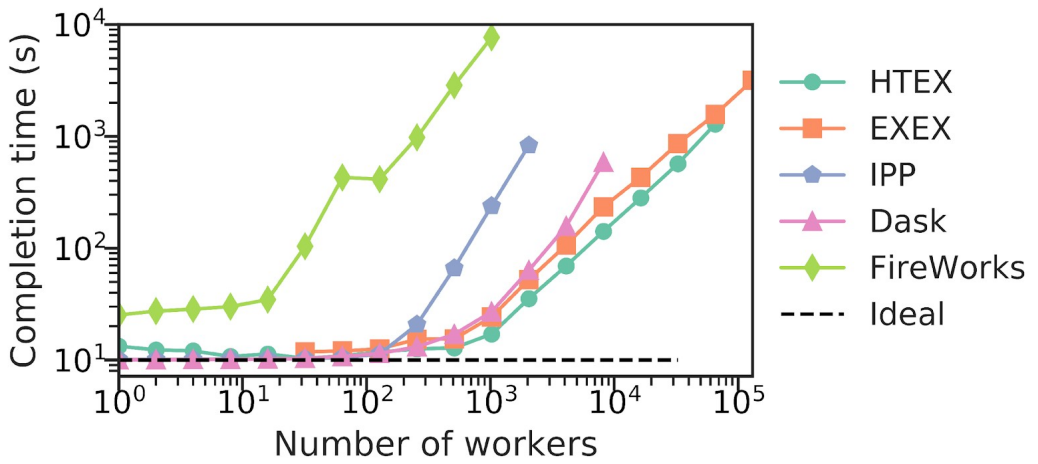
HTEX and EXEX outperform other Python-based approaches

Framework	Maximum # of workers†	Maximum # of nodes†	Maximum tasks/second‡
Parsl-IPP	2048	64	330
Parsl-HTEX	65 536	2048*	1181
Parsl-EXEX	262 144	8192*	1176
FireWorks	1024	32	4
Dask distributed	4096	128	2617

Strong scaling (50K 1s tasks)



Weak scaling (10 1s tasks per worker)



Babuji et.al. DOI:10.1145/3307681.3325400

How Do You Use Parsl?

- Basic use is to just add a decorator to your python functions

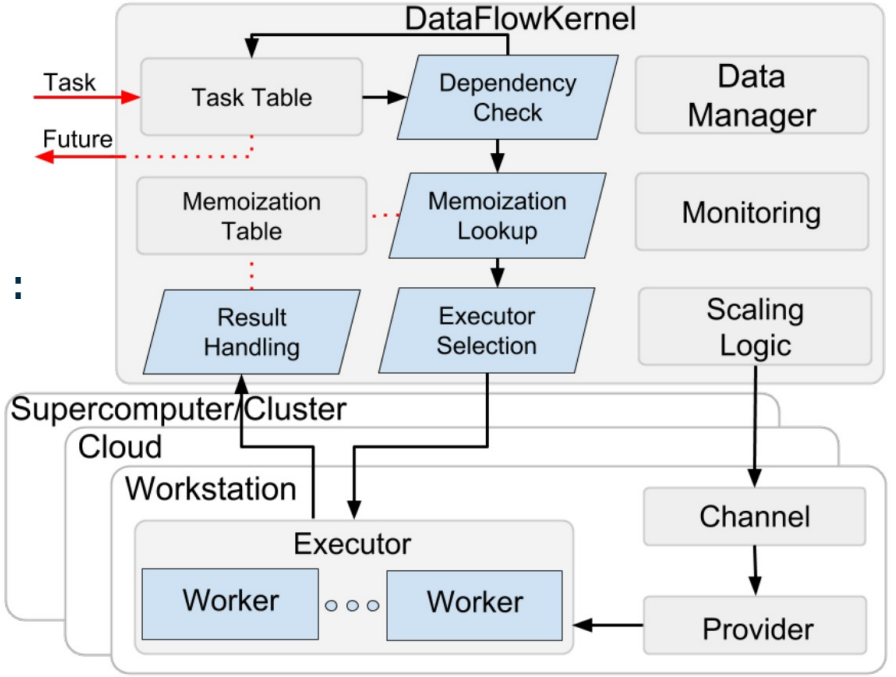
```
@python_app
def random_multiply(x):
    import random
    return x * random.random()

@bash_app
def echo(word, stderr='std.err', stdout='std.out'):
    return f'echo "{word}"'
```

- Set up your configuration:

  defines resources and

  requirements



Babuji et.al. DOI:10.1145/3307681.3325400

How Do You Use Parsl?

- Call the decorated functions as normal

```
In [1]: value = random_multiply(5)
In [2]: value.result()
Out[2]: 4.934032990642622
```

```
In [1]: print(f"Return value is {echo('Hello world').result()}")
Out[1]: 0
In [2]: for line in open('std.out', 'r').readlines():
   ...:     print(line)
Out[2]: Hello world
```

Parsl and Files

- Any file passed as input or given as output needs to be wrapped in a Parsl File object
- Can handle local and remote files via
  - ftp/http/https
  - rsync
  - Globus file transfer

```
File(“/local/scratch/myinput.txt”)
File(“https://github.com/Parsl/parsl/blob/master/README.rst”)
File(“globus://037f054a-15cf-11e8-b611-0ac6873fc732/unsorted.txt”)
```

Parsl Structure

- Parsl Apps and Files are *futures*, a placeholder for an object which may not be available yet
- Any Python object (primitives, classes, etc.) can be passed as arguments
  - Must be serializable
- All `import` statements must be inside the function body
- stdout/stderr from remote machines may not be captured
-

Parsl Tutorial

- Fully hands on using Jupyter Notebooks (all you need is a browser and wifi connection)
- Parsl basics
  - Writing Apps
  - Parsl Blocks
- Configuring Parsl
- Handling files in Parsl
- Writing workflows
- Basic debugging

What is Parsl?

- Flexible and scalable parallelization framework
- Purely written in Python
- Simple to use:
  - `@python_app` decorator for python functions
  - `@bash_app` for anything else
- Each task can be run on a different resource
  -
    - o

Questions?

This material is based in part upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number *DE-NA0003963*.