Multi-site dynamic computational ensembles with libEnsemble + funcX

John-Luke Navarro (jnavarro@anl.gov)
Argonne National Laboratory

Parsl & funcX Fest 2022
What is **libEnsemble**?

- A toolkit for coordinating workflows of concurrent asynchronous and dynamic computations

- Aims for:
  - Extreme scaling
  - Dynamic ensembles (concurrent evals)
  - Dynamic Resource Management
  - Monitoring/cancelling apps
  - Resilience/fault tolerance
  - Portability and flexibility
  - Exploitation of persistent data/control
  - Low start-up cost
The *libEnsemble* Paradigm

- **Worker** processes run either *simulator* or *generator* functions.

- The **Manager** distributes output from a worker’s *generator* function to workers to evaluate via *simulator* functions.

- How/when output is distributed is customizable via an optional *allocation* function.
Example functions

• Generators:
  • Asynchronously Parallel Optimizer for Solving Multiple Minima (AP0SMM, included with libEnsemble)
  • Surrogate model calibration/inference (via Surmise)
  • Sparse Grids sampling (via Tasmanian)

• Simulators:
  • Accelerator structure and beam line analysis (via 0PAL)
  • Particle-in-cell evaluations (via WarpX)
  • Ice sheet modeling
  • RNN-training
Workers call Simulator and Generator functions

Calls

.worker_process  simulator()  ./my_app.ex

or:

Calls

.worker_process  simulator()  ./my_app.ex
libEnsemble and funcX integrate nicely!

worker_process submits to funcX and simulator schedules.

Easy cross-system, heterogenous ensembles!
Multi-site dynamic ensembles

generator() → libEnsemble

func

simulator() → ... → simulator()
Configuring a libEnsemble function for submission to funcX

```python
from simulation_module import my_sim_func

sim_specs = {
    'sim_f': my_sim_func,
    'in': ['input1','input2'],
    'out': [('output1', float)],
    'funcx_endpoint': '3af6dc24-3f27-4c49-8d11-e301ade15353'
}

def my_sim_func(Input, persis_info, sim_specs, _):
    # All imports need to be within function
    from libensemble.executors import MPIExecutor

    # Instantiate an MPI Executor instance
    excr = MPIExecutor()

    # Register our app
    excr.register_app(full_path='/local/path/forces.ex', app_name='forces')

    # Submit our app using the autodetected MPI runner
    task = excr.submit(app_name='forces')

    # Simply wait for the app to complete
    flag = excr.polling_loop(task)
```

Configuration script

**Function**
Future Work

• Launch *persistent* functions to remote machines?
• How to send intermediate results back to Manager?
• How to send/receive manager-kill signals for running apps?
• Experiment with dynamically choosing endpoints?
• …Find interested users!
Questions?

- [https://libensemble.readthedocs.io](https://libensemble.readthedocs.io)
- [https://github.com/Libensemble/libensemble](https://github.com/Libensemble/libensemble)

libEnsemble@lists.mcs.anl.gov

- Thank you very much!