Colmena: Steering Ensemble Simulations on HPC

Logan Ward ...and a big team at Argonne, UC, ...

Data Science and Learning Division
Argonne National Laboratory
1 November 2021
Acknowledgements: The (growing!) team

**Argonne: ExaLearn** – Using AI with HPC
Yadu Babuji, Ben Blaiszik, Ryan Chard, Kyle Chard, Ian Foster, Greg Pauloski, Ganesh Sivaraman, Rajeev Thakur

**Argonne: JCESR** – Molecular modeling for batteries
Rajeev Assary, Larry Curtiss, Naveen Dandu, Paul Redfern

**MolSSI** – Workflows for quantum chemistry
Lori A. Burns, Daniel Smith, Matt Welborn, *many other open-source contributors*

**PNNL: ExaLearn** – Graph algorithms for learning
Sutanay Choudhury, Jenna Pope

**BNL: ExaLearn** – Optimal experimental design
Frank Alexander, Shantenu Jha, Kris Reyes, Li Tan, Byung Jun, *and more*

**Argonne ALCF** – AI, Data and Simulation on HPC
Murali Emani, Alvaro Vazquez-Mayagoitia, Venkat Vishnawath
Big Picture: Expanding Computational Campaigns to the ExaScale

**Current Model:** Humans steer HPC, HPC performs simulations

**Current Model Won't Scale.** Humans are slow and not getting any faster

Our goal: HPC steering itself!
Parallelism makes steering on HPC difficult

**Root Problem:** Sequential search is impractical, we must run >1 simulation at once

Consider a few parallel strategies...

![Graph 1: Wait for N tasks to complete, then pick next batch](image1)

- **Wait for N tasks to complete, then pick next batch**
  - Most information per decision
  - Least utilization

![Graph 2: Pick new tasks as soon as one completes](image2)

- **Pick new tasks as soon as one completes**
  - Least information per decision
  - Greatest utilization

![Graph 3: Maintain a task queue](image3)

- **Maintain a task queue**

**Bottom Line:** Active learning on HPC requires intelligent policies

**Today’s Talk:** You can build complicated steering with Colmena

…and that lets you do cool things.
Colmena: An overview
What kind of “intelligence” goes into steering applications

**Observation:** We have many policy ideas…

- Submit a new simulation *once another completes*
- Retrain a model *after 8 successful computations*
- *Allocate more nodes to inference* after models finish training

...and others are possible.

**Solution:** We need a way of **programming** agents to encode such policies
Colmena is a wrapper over Exascale Workflow tools (e.g. Parsl!)

**Programming Model: Task Queues**

```python
# Primitive Units
queue.send_inputs(1)
result = queue.get_result()
```

**Programming Model: Agents**

```python
class Thinker(BaseThinker):
    @agent
def make_work(self):
        self.queue.send_inputs(1)
```

**Task Server:**
- Dispatches work requests to compute
- Communicates results back to thinker

**Backend:** Parsl
- Supports most HPC and cloud services
- Easily configure multiple worker types, multi-site workflows
- Limited support for ensembles of MPI applications
- **Future:** Balsam, FuncX, RCT
Building a Colmena app: Defining the “tasks” and “thinker”

Key points:
1. Subclass the “BaseThinker” abstract class
2. Mark “agent” operations form the policy
3. Communicate with method server via queues
4. Communicate with other via Threading primitives

How does it work:
- “.run()” launches all agents

```python
class Thinker(BaseThinker):
    def __init__(self, queue):
        super().__init__(queue)
        self.remaining_guesses = 10
        self.best_guess = None
        self.best_result = inf

@result_processor(topic='simulate')
def consumer(self, result):
    # Update the best result, check for termination
    if result.value < self.best_result:
        self.best_result = result.value
        self.best_guess = result.args[0]
        self.remaining_guesses -= 1
    if self.remaining_guesses == 0:
        self.done.set()

@agent
def producer(self):
    while not self.done.is_set():
        # Make a new guess
        self.queues.send_inputs(self.best_guess, method='task_generator', topic='generate')
        # Get the result, push new task to queue
        result = self.queues.get_result(topic='generate')
```

```python
@agent
def producer(self):
    while not self.done.is_set():
        # Make a new guess
        self.queues.send_inputs(self.best_guess, method='task_generator', topic='generate')
        # Get the result, push new task to queue
        result = self.queues.get_result(topic='generate')
```
Main effort: Defining the “tasks” and “thinker”

• Main steps:
  1. Write methods as Python functions
  2. Specify computational sources
  3. Instantiate method server

• Launching the server:
  – “.run()” launches server as a second process
  – Main thread reads from queue, launches workflows
  – Workflows end by writing results to queue
  – Parsl distributes work, collects results

```python
def target_function(x: float) -> float: return x ** 2
def task_generator(best_to_date: float) -> float:
    from random import random
    return best_to_date + random() - 0.5

cfg = Config(executors=[
    HighThroughputExecutor(max_workers=4)])
doer = ParslTaskServer([target_function, task_generator],
                       server_queues, cfg)
doer.start()
```
Colmena and Molecular Design
What does our “active learning application” look like
What is the application behavior?

1. Start by running **inference** on all nodes.
2. Run **simulations** on all nodes.
3. After 8 complete, switch nodes to **training** models.
4. After training, re-task available nodes to **inference**.
5. After inference, reallocate all nodes back to **simulation**.
Did the application have good scientific performance? [Yes]

![Graph showing performance](image)

- **No-retrain**
- **Random**
- **Update-8**

- Found 10% more high-performing molecules with same allocation size
Where are the sources of underutilization in ours run?

Poor utilization due to long-tail, trailing NWChem tasks

NWChem are long compared to allocation walltime

Starting a new TF node takes minutes

*Comparable in cost to inference tasks*
Summary: Colmena is for deploying AI+Simulation HPC

Key points:

- AI will play an increasing role in controlling campaigns of simulations
- Success will require deploying AI on HPC
- Colmena provides a Python library for building applications to interleave simulation and AI workflows
  - Simple, agent-based programming model
  - Backed by performant workflow engines (Parsl!)

See also: [https://colmena.rtfd.io/](https://colmena.rtfd.io/), [https://github.com/exalearn/colmena](https://github.com/exalearn/colmena)