

# Weekly Progress Report

Camille Scott

February 28, 2015

*Lamprey*

## **Finished**

- Finished the abundance estimation pipeline. Uses the same metadata as the rest of the preprocessing pipeline, along with the pydoit task API. Steps are:
  1. Build bowtie2 index for assembly
  2. Extract paired reads and split
  3. Run bowtie2-align with loose parameters
  4. Run eXpress
  5. Aggregate results into final abundance matrix
- Running new abundance estimation pipeline on lamp3
  1. Many delays this week from HPC scheduler issues
  2. Still waiting on the paired-end samples to finish (these took longest and were cut short)

## **TODO next**

- Use new abundance info to build gene models from existing transcripts
- Get detailed comparison metrics between existing gene models, transcript alignments, and new transcript-based gene models
- Finalize reproducible pipeline
  1. Finish data management plan (git-annex? where to host?)
  2. Finish unifying metadata, task dependency between pre-processing, Trinity, and post-processing
  3. Test entire pipeline on EC2

*khmer multiprocessing***Finished**

- Introduced new state management system for all Async subclasses
  1. 4 states: START, DORMANT, RUNNING, WAIT
    1. START: state prior to constructor call
    2. DORMANT: State during construction, before call to `start()`, after calling `stop()`
    3. RUNNING: Reset all counters, spawn threads for `consume()` function (ie, begin processing reads)
    4. WAIT: Reached once the current parser is empty, or generally, once all processing is complete; `start()` can reset and return to RUNNING, `stop()` transitions to DORMANT
  2. State is shared between all threads of any Async object
  3. Externally accessible, thread-safe functions for setting and retrieving state
  4. Fixed several deadlocking issues that were difficult to diagnose with previous non-generalized state system
- AsyncParser is now it's own Async subclass, instead of being built into AsyncSequenceProcessor, allowing easier state management with the new system and a more extensible system for parsing. Also allows the AsyncSequenceProcessor more flexibility in where it receives reads from, and easier testing.
- Removed all AsyncWriter classes, as hashtables are now entirely threadsafe and all writing is handled directly by Hashtable
- Added more robust / safe initialization of Hashtable thread safety

**TODO Next**

- Fix remaining deadlock in AsyncDiginorm
- Flesh out tests
- Add output callback system previously discussed