Towards Efficient Python Interpreter for Tiered Memory Systems

Yuze Li[1] Shunyu Yao [1] Jaiaid Mobin [2] M. Mustafa Rafique [2] Dimitrios Nikolopoulos [1] Kirshanthan Sundararajah [1] Huaicheng Li [1] Ali R. Butt [1]

[1]Virginia Tech [2]Rochester Institute of Technology

Tiered Memory Systems

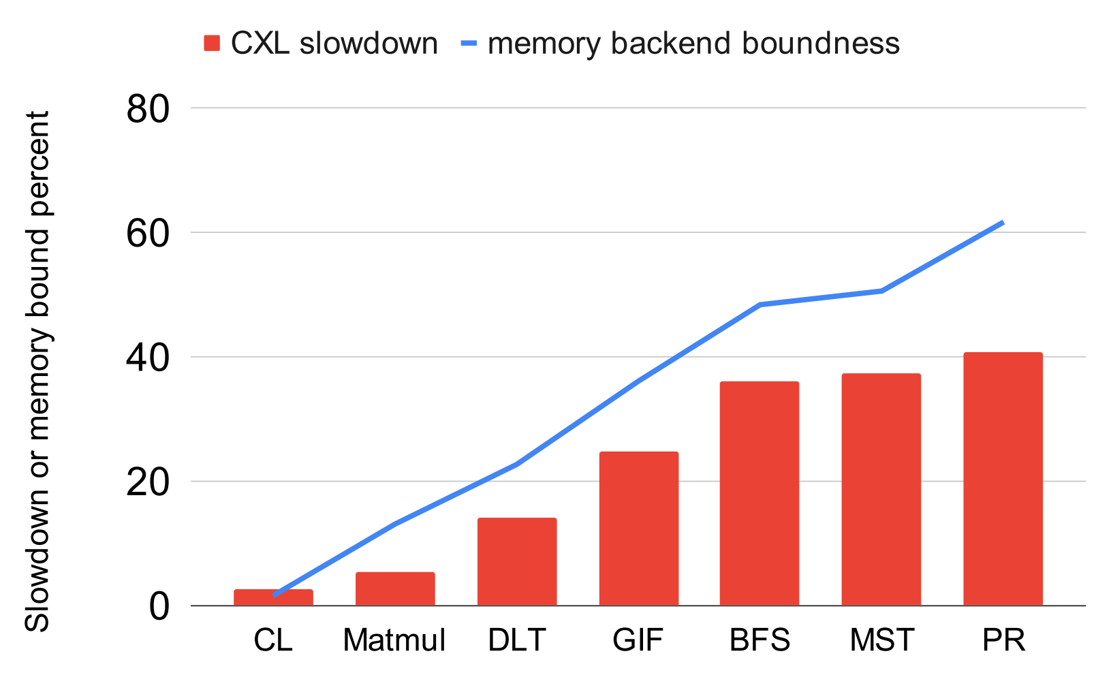

Figure 1. Slowdown percent of different workloads in CXL.

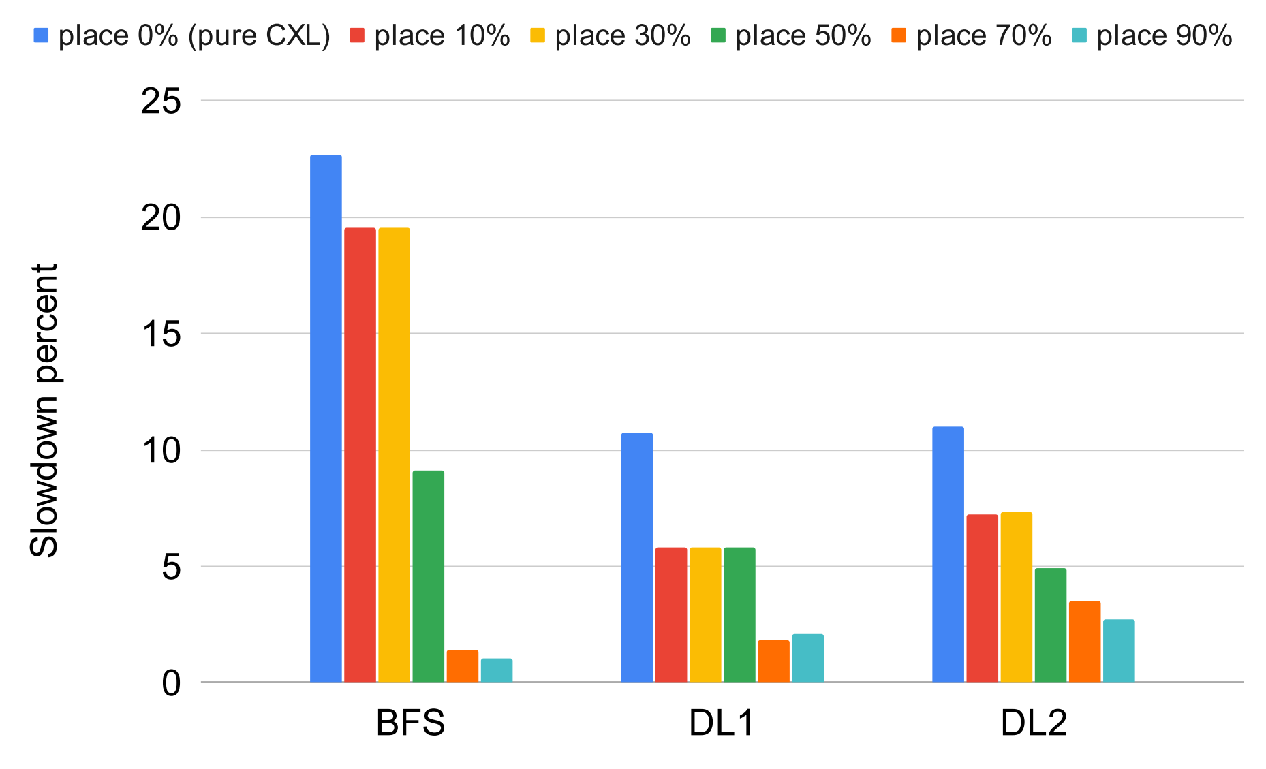

Figure 2. Workloads slowdown by static placing different percent of hottest memory pages to DRAM, the rest to CXL.

The emergence of low-latency non-DDR technologies offers cheaper $/GB memory cost. Running modern data-intensive applications in tiered memory systems experiences different percentage of slowdown. The principle is to track data access frequencies and automatically migrates them among tiered memory resources. Thus, a good solution must be:

- **Accuracy**: Be precise about the memory boundaries to be hot or cold.
- **Low overhead**: Solutions should not interfere with applications that much.
- **Portability**: Can readily be deployed to today's cloud.
- **Transparency**: No need for program re-writing, static analysis.

Problem of Existing Solutions

OS level: Page table entry checking, hardware event sampling, LRU, AutoNUMA, etc.

- **Coarse-grained observation point**: Sub-page information, and application semantics cannot be extracted.
- **Unbalanced accuracy and overhead**: By increasing the accuracy, overhead will increase

Runtime level: Defines new programming models through APIs, source code static analysis, and profiling.

- **lack of transparency**: Involves non-trivial programmer efforts, or exhaustive profiling.

None of the existing methods can be directly ported to the popular language, Python, considering Python's top-ranking position in 2023.

Challenges of Tracking Python Object Temperatures

Challenge 1: Method of Tracing

- Unlike C++, CPython does not offer smart pointer and operator overloading.
- Unlike JVM-based runtime, CPython does not have read-write barriers to instrument.

Challenge 2: Tracing Overhead

- CPython only maintains the references of **container** PyObjects, obtaining **all** PyObjects references requires the **GIL held** (application paused).

Challenge 3: Handling Native Calls

- CPython does not capture runtime semantics in native executions (C/C++).

Major Insights

Insight 1: **Reference counting** can be a potential indicator to infer PyObjects accesses (challenge 1).

Insight 2: The set of live PyObjects is **not likely to change** until a cyclic-GC is triggered; selectively tracing based on object semantics (challenge 2).

Pypper Overview

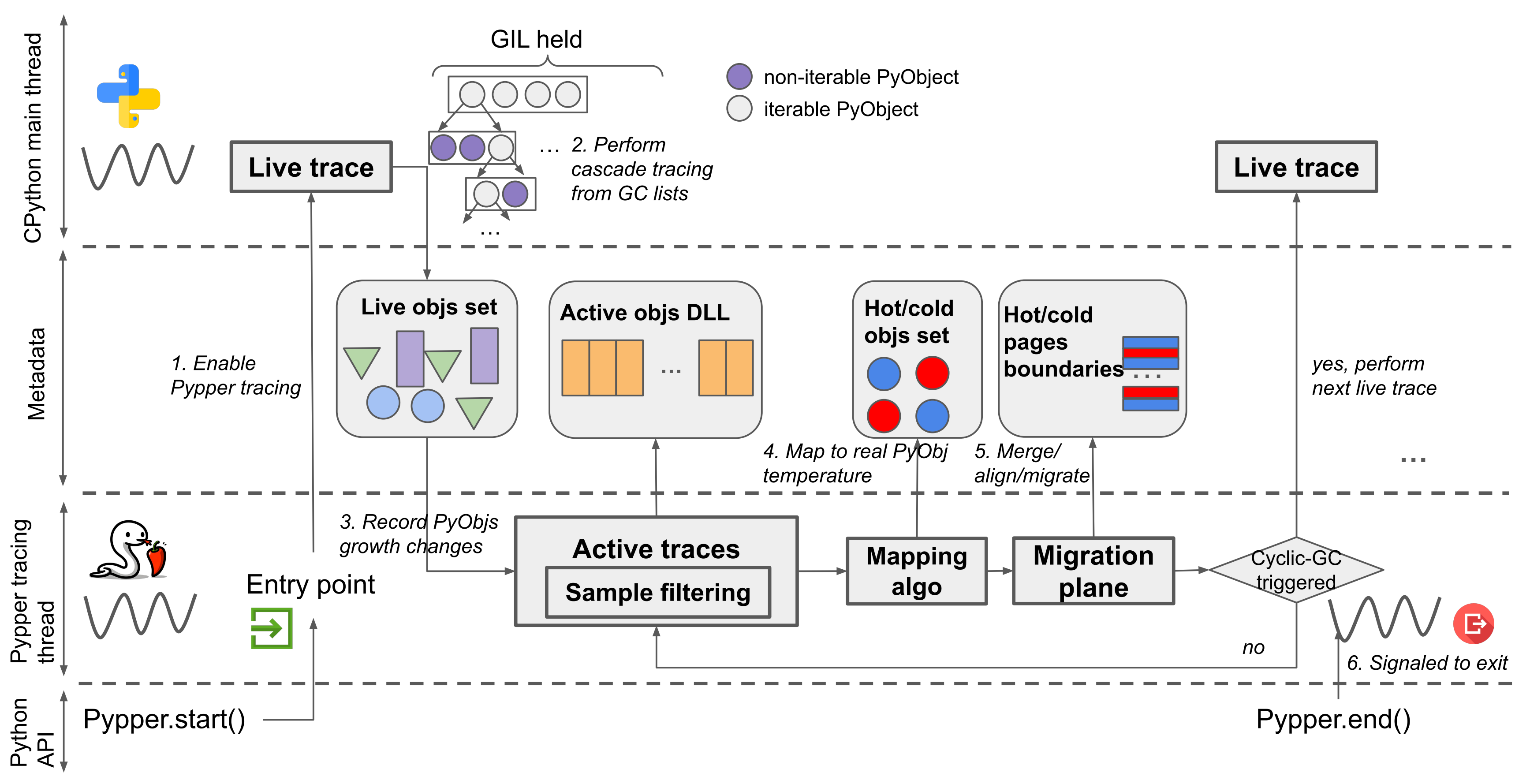

Figure 3. Pypper's workflow.

Pypper comprises a control layer and a metadata layer. The control layer populates and analyzes the metadata.

1. Invoked from Python API (`Pypper.start()`), tracing enabled within CPython main thread.
2. **Live trace** cascade traverses the cyclic-GC list to get all PyObjects references.
3. Pypper triggers a separate CPython thread for **consecutive active traces**, and records refcnt changes for each observed PyObject.
4. **Mapping algorithm** inspects the captured refcnt changes to infer the real PyObject temperatures.
5. **Migration plane** merges hot/cold objects into compact segregated memory ranges, aligns them to page boundaries, before migrating to designated areas.
6. Upon receiving stop signal (`Pypper.end()`), Pypper frees metadata, resets states, stops tracing.

Preliminary Results

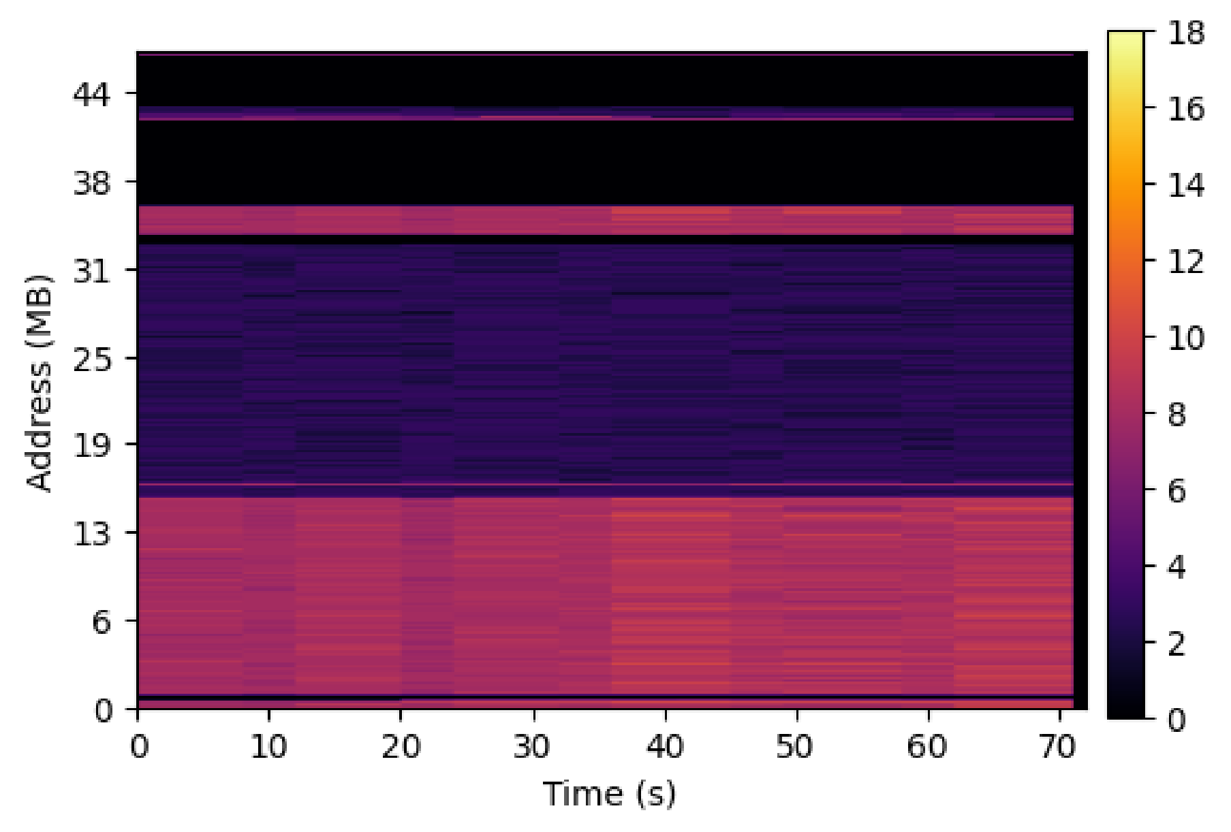

Figure 4. Inferred PyObj temperatures based on refcnt changes.

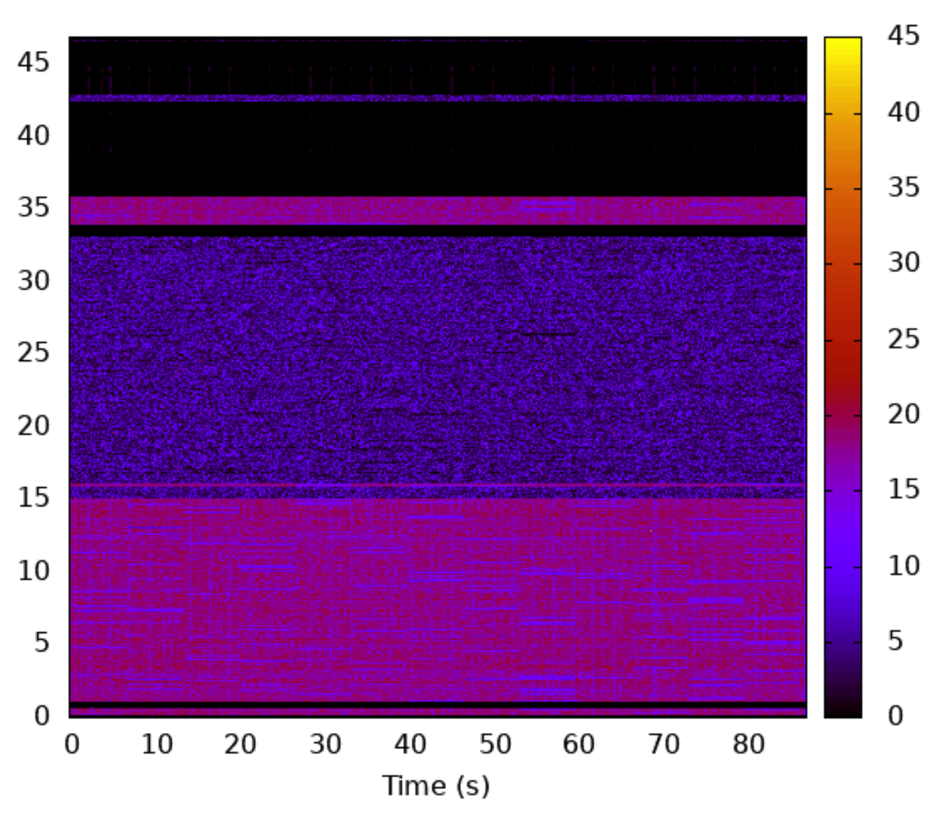

Figure 5. Real heatmap from OS-based profiling.

Takeaway: The reference counting in the GC scheme can also be used to infer object temperatures by defining a mapping model.

WiP and Future Work

Live Trace Overhead Mitigation (WiP)

- Make the best use of CPython's cyclic-GC module by only traversing **newly survived** container PyObjs.
- Filter live PyObjs by observing their semantics, e.g., length, depths.

Mapping Algorithm (WiP)

- A fine-grained mapping module from refcnt-changing to real object temperatures is yet to be defined.

Handling Native Executions (future work)

- Pypper should distinguish and handle native execution that is not based on refcnt changes.