A Concurrent Hash Table for CPython

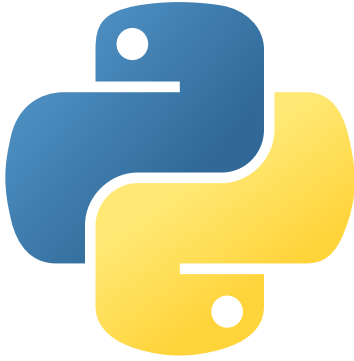
MSc Thesis Dissertation

Daniele Parmeggiani

Università degli Studi di Trento, DISI

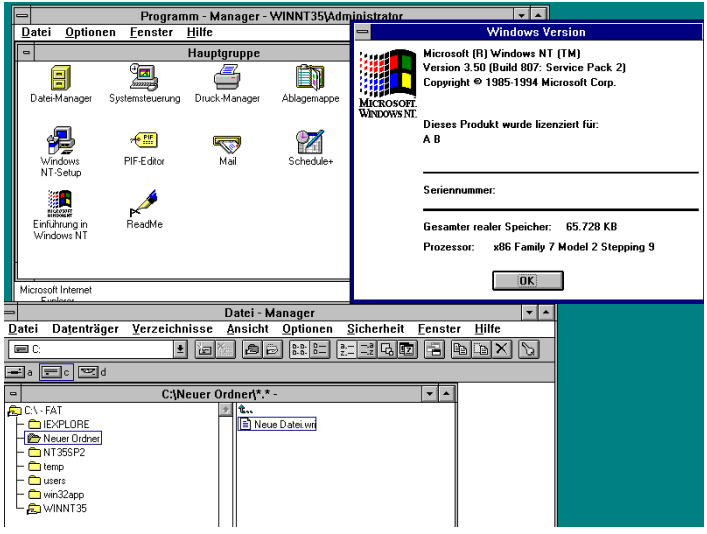
2024-10-16

Python and CPython



Back to ~~the future~~ 1994

Back to ~~the future~~ 1994



CPython and its GIL

CPython and its GIL

3.13

Free-threading

java.util.concurrent

* Semaphore ✓
* CyclicBarrier ✓
* Lock ✓
* Concurrent**HashMap**
* ConcurrentLinked**Queue**
* CopyOnWriteArray**List**
* Atomic**Boolean**
* Atomic**Integer**
* Atomic**Reference**
* ...

Thread synchronization utilities for Python.

The *Cereus Greggii* is a flower native to Arizona, New Mexico, Texas, and some parts of northern Mexico.
This flower blooms just one summer night every year and in any given area, all these flowers bloom in synchrony.

https://github.com/dpdani/cereggii

Exposes a so-called *natural parallelism*.
High-level design:

* linear probing
* resizable
* double-hashing
* split index and data tables
* lock-free*

Mostly inspired by:

* T. Maier's Growt concurrent hash table, and
* CPython's built-in sequential hash table (dict).

Concurrent Resizing

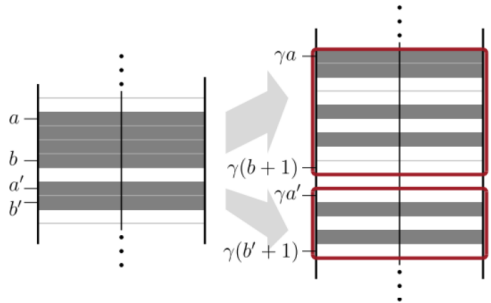


Figure: Growt's migration process.[1]

[1] T. Maier et al., Concurrent Hash Tables: Fast and General(?)!, Fig. 1.

Double-Hashing

* Growt uses most-significant bits as position
* dict uses least-significant bits as position

Double-Hashing

* Growt uses most-significant bits as position
* dict uses least-significant bits as position
* AtomicDict:

Double-Hashing

* Growt uses most-significant bits as position
* dict uses least-significant bits as position
* AtomicDict:
    1. get an object's hash (as generated by CPython)

Double-Hashing

* Growt uses most-significant bits as position
* dict uses least-significant bits as position
* AtomicDict:
  1. get an object's hash (as generated by CPython)
  2. re-hash with CRC32

Double-Hashing

* Growt uses most-significant bits as position
* dict uses least-significant bits as position
* AtomicDict:
  1. get an object's hash (as generated by CPython)
  2. re-hash with CRC32
  3. use most-significant bits as position

Split Index and Data Tables

- put keys and values into a separate *data* table
- actual hash table is an *index* over the data table

Split Index and Data Tables

* put keys and values into a separate *data* table
* actual hash table is an *index* over the data table

* size of an index slot: 1 to 8 bytes
  * dependent on capacity
* size of a data entry: 32 bytes

Split Index and Data Tables

* put keys and values into a separate *data* table
* actual hash table is an *index* over the data table
* so that:
  * the index stays sparse to better handle collisions
  * the large entry size decreases false sharing

* size of an index slot: 1 to 8 bytes
  * dependent on capacity
* size of a data entry: 32 bytes

Lock-freedom

Definition

At least one thread can always make progress. (No locks $\Rightarrow$ no deadlocks.)

Lock-freedom

Definition

At least one thread can always make progress. (No locks ⇒ no deadlocks.)

- For AtomicDict:
  - no deadlocks!
  - (unless a resizing is in progress)
  - lock-free resizing is not practical
  - if you correctly set `min_size`, no resizing ever happens

Small Contributions

* Partitioned iterations

```
d = AtomicDict(...)

def my_thread(thread_id, threads_count):
    for key, value in d.fast_iter(partitions=threads_count,
                                  this_partition=thread_id)
        do_some_stuff(key, value)
```

Small Contributions (cont.d)

* Reduce

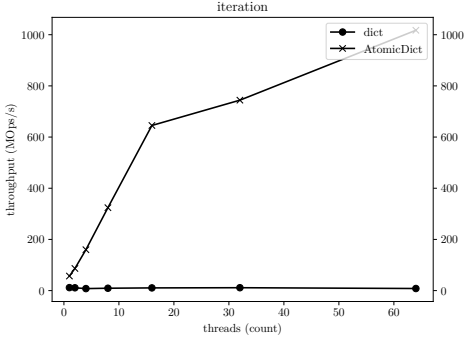
```
d = AtomicDict(...)

data = [
    ("red", 1),
    ("green", 42),
    ("blue", 3),
    ("red", 5),
]

def count(key, current, new):
    if current is NOT_FOUND:
        return new
    return current + new

d.reduce(data, count)
```

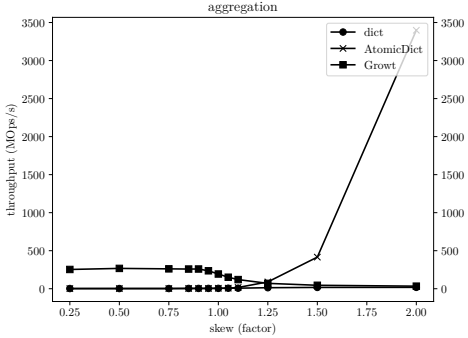
Small Contributions (cont.d)

* Batch Lookup

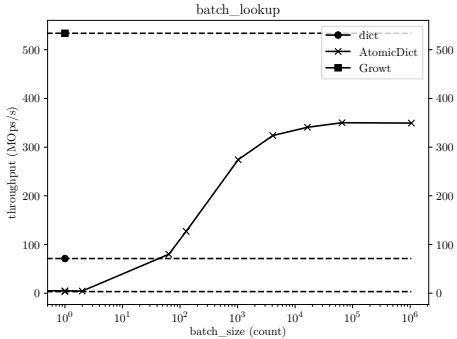
```
foo = AtomicDict({'a': 1, 'b': 2, 'c': 3})

result = foo.batch_getitem({
  'a': None,
  'b': None,
  'f': None,
})

assert result == {
  'a': 1,
  'b': 2,
  'f': cereggii.NOT_FOUND,
}
```

Measurements

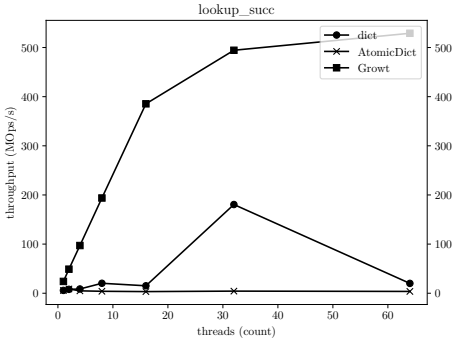


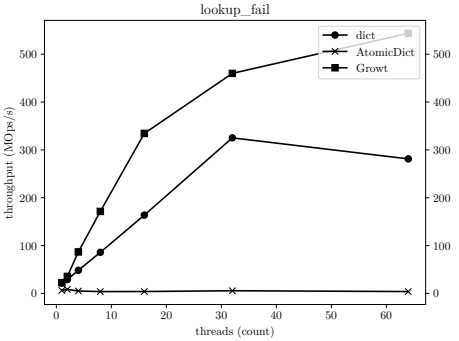


Measurements (cont.d)



Measurements (cont.d)





Pittsburgh (USA), May 2024.

Summary

1. Python = language / CPython = interpreter
2. CPython and its GIL
   - historical rationale
3. Free-threading in CPython 3.13
   - per-object locks
4. Java has atomic data structures, Python doesn't
5. cereggii
   - atomic data structures for Python (currently: AtomicRef, AtomicInt, AtomicDict)
6. AtomicDict
   - natural parallelism
   - inspired by:
     - Maier's Growt (concurrent)
     - CPython's dict (sequential)
   - linear probing
   - resizable
   - double-hashing
   - split index and data tables
   - lock-free*
   - partitioned iterations
   - reduce
   - batch lookup
7. Python Language Summit