Implementing a Garbage Collector in a High Level Language using the OMR GC Framework

Joannah Nanjekye

UNB/IBM

2019, November, 05

Outline

CAS–Atlantic

Outline

CAS-Atlantic

Garbage Collection

- A way of reclaiming unused memory automatically i.e., automatic memory management.
- Unused objects are destroyed to give room for new ones.
- In C/C++, use *free()* and *delete()* to free memory.
- In garbage collected languages, it is automatic.
- Examples include Java, Python, etc.

Reuse in Garbage Collection

- Need reuse because implementing GC is hard and time consuming.
- Need to abstract GC implementation to allow GC portability for any runtime.
- Reduces need for GC expertise about the nitty gritty and complex details.

Existing Solutions for Reuse in Garbage Collection

- GC-as-a-Service (GaS) Library by Wegiel and Krintz.
  - Provides a shared C library.
  - Moves GC logic into a modular library.
  - Supports a non-moving GC.
- Eclipse OMR GC Framework.

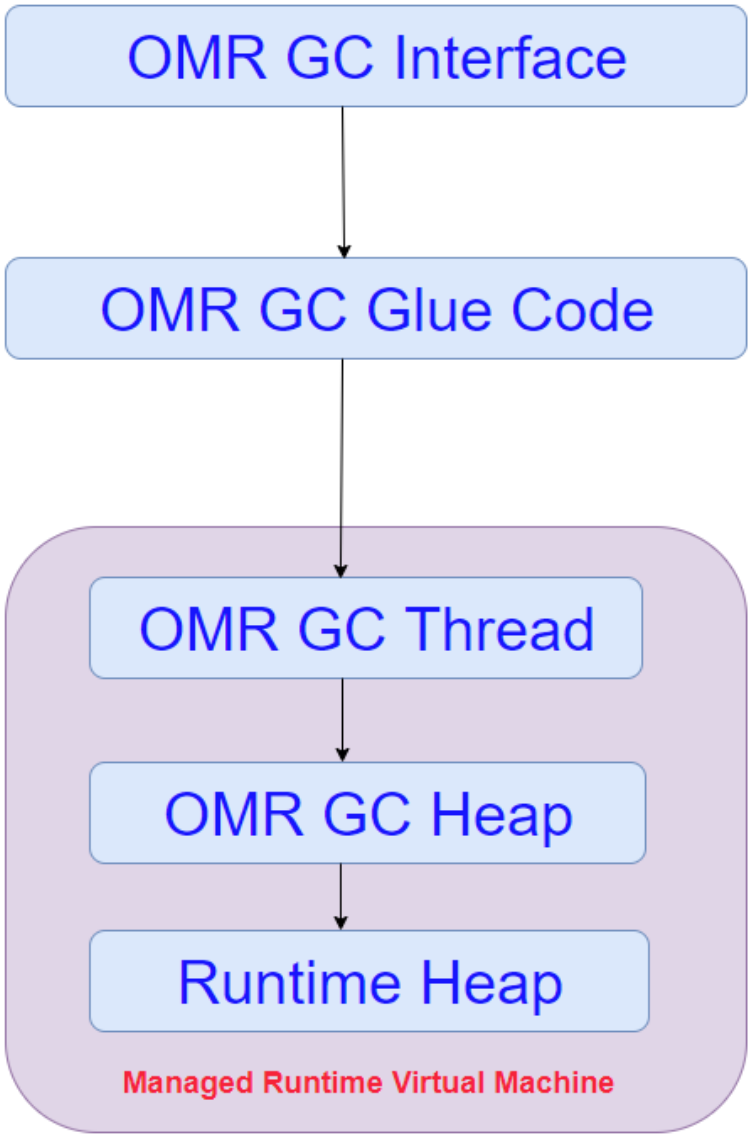
Outline

CAS-Atlantic

Eclipse OMR GC Framework

- A component of the Eclipse OMR project.
- The Eclipse OMR project is a set of open source C and C++ components for building robust language runtimes.
- The GC framework is a component for implementing automatic memory management for any runtime.
- It exposes a standard C interface that any runtime can call to integrate Garbage Collection.

Eclipse OMR GC Architecture



CAS–Atlantic

Outline

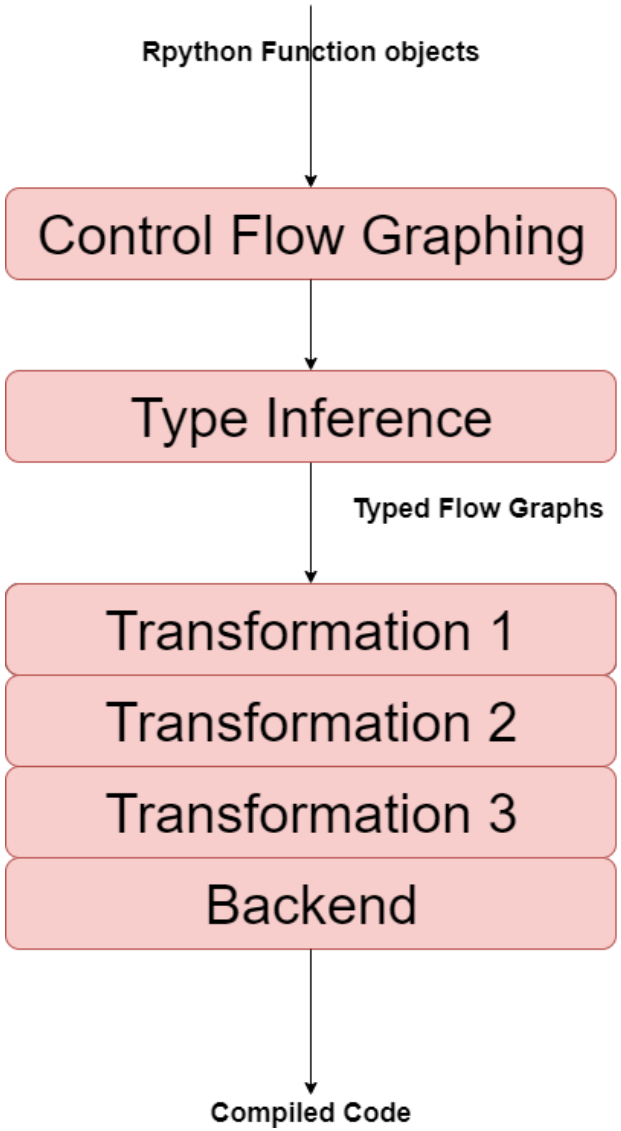
About Pypy

- Framework for writing interpreters in Python.
- To a layman, it is a *Python interpreter implemented in Python.*
- Pypy tries to be compatible with CPython as its reference implementation.

Pypy Architecture

- Pypy's Architecture has two components:
  - ***Standard Interpreter***. An implementation of the Python programming language mostly compliant with the current version of the language.
  - ***Translation framework***. A translation tool-suite whose goal is to compile subsets of Python(Rpython) to various environments. Rpython is a subset of the Python language called "restricted Python".

The Translation Framework



CAS–Atlantic

Outline

GCs in Pypy

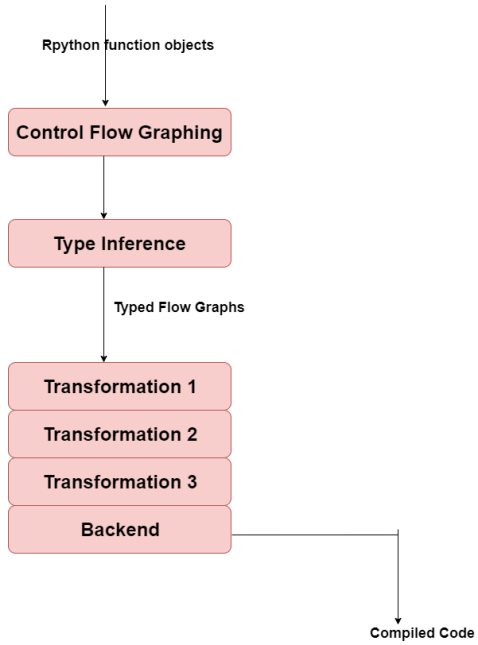
- Each allocation call in the control flow graphs is replaced with a call to the GC.
- There are multiple GCs.
- Significant is the *minimark* which is a generational GC using mark-sweep for the old generation.
- The default is the *incminimark*.
- An incremental and improved version of minimark with reduced *pause times*.

Facts about GCs in Pypy

- Written in Rpython which assumes automatic memory management.
- The GC is analyzed like any other program during translation.
- Low level calls do not assume automatic memory management e.g., C code.
- The GC needs to support allocation for both Python visible objects and internal interpreter objects e.g., list, instances etc.

How to integrate a new GC in Pypy

- There are two options:
  - *Link an external GC to the C code produced from the first transformation.*
  - *Write an actual GC coupled with a transformation.*
- The GC transformer inserts a GC in a program being translated.



CAS–Atlantic

Outline

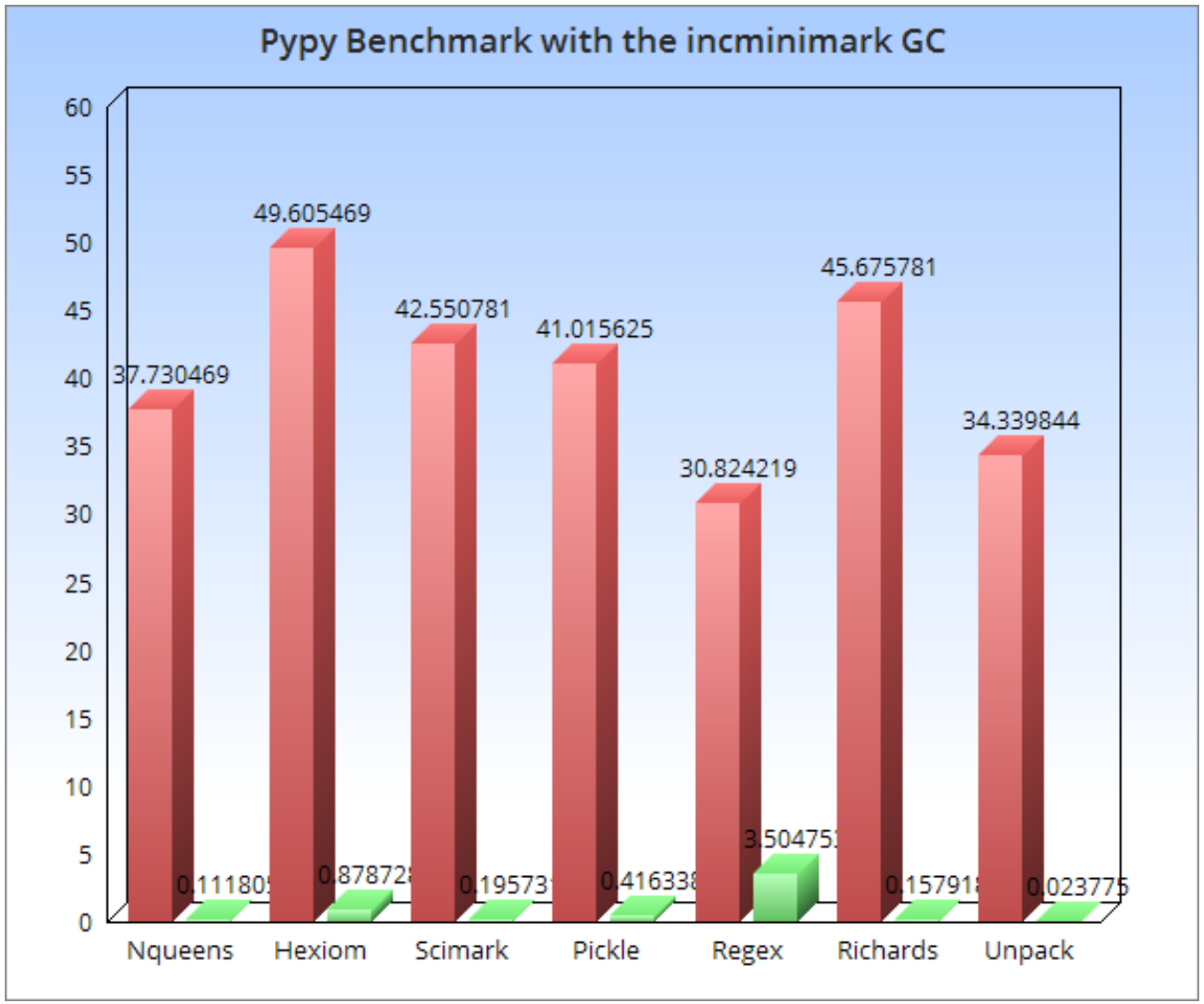
OMR GC Integration in Pypy

- We used the second option above i.e., implemented an actual GC.
- The OMR GC is written in C/C++.
- Rpython is like Python, there was need to wrap the C calls.
- RFFI was used for the wrapping.

```
# Extracted from omrmarkandsweep.py

eci = ExternalCompilationInfo(
includes = ['malloc.h'],
post_include_bits = [
    "PypyAllocateObject(uintptr_t allocSize,
    uintptr_t allocateFlags);\n"
],
)

pypy_AllocateObject = rffi.llexternal(
    'PypyAllocateObject',
    [rffi.INT, rffi.INT],
    rffi.VOIDP, compilation_info=eci)

def omr_allocate(self, size, allocateFlags):
    return pypy_AllocateObject(size, allocateFlags)
```

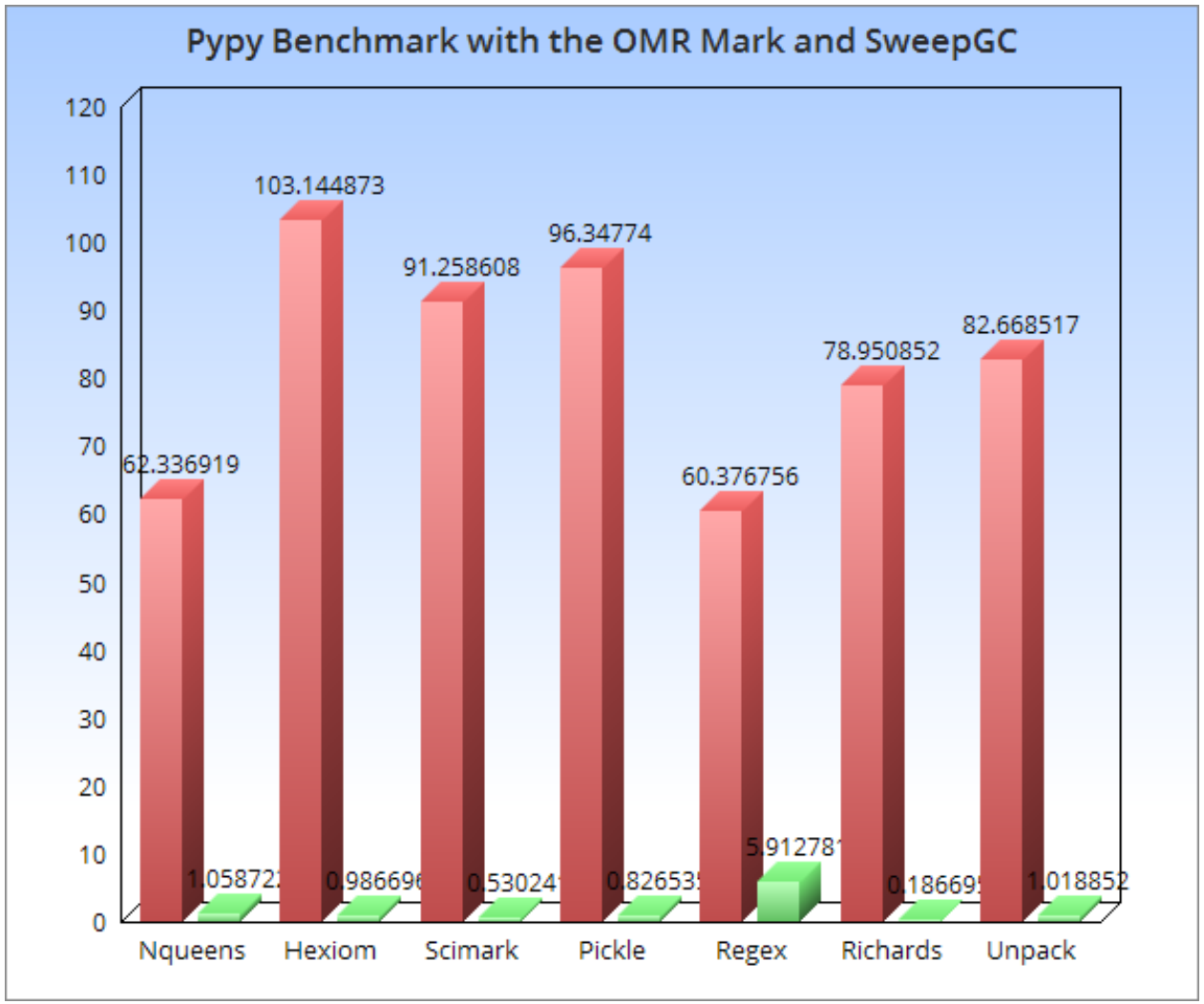
CAS-Atlantic

Results - Pypy with Incmiminimark GC



CAS–Atlantic

Results - Pypy with OMR GC



CAS–Atlantic

Summary of Results

- Not conclusive at all.
- Typical case of memory management not implemented so well, i.e bugs in the GC implementation.
- Runs 0.5x to 5x slower.
- Uses 1.5x to 25x more memory.

Results - Development Effort

- Started in Spring 2019.
- I only needed to know what the OMR GC functions did.
- Documentation would have even simplified this process.
- I spent more time figuring out Pypy than OMR.

Why the bad Performance

- Memory management bugs as this is still work in progress at the moment.
- The incminimark is powerful, its unfair to compare it with a Mark and Sweep Implementation.
- **Wrapper overhead**.
- **RFFI does allocations when making low level calls. This memory needs to be managed manually.**

Why did I use RFFI in the first place

- Be as Pythonic as possible.
- The Boehm GC integration is older than RFFI. We needed to know if a higher level integration was better than this.
- It had a promise of a cleaner integration. The authors of the Boehm GC hacked the translation framework -which is the usual meta-meta-meta Pypy programming.

CAS–Atlantic

Outline

CAS–Atlantic

Conclusion

- RFFI does allocations when making low level calls. This memory needs to be managed manually.
- To reduce wrapper overhead and having to manage memory manually, hack the translation tool.
- With this option:
  - Operate at a lower level than RFFI.
  - Let the GC transformer insert the low level OMR GC operations.
  - The code generator can turn these operations into direct C calls to OMR.

Conclusion

- We may be able to get some 5 percent improvements if we integrated a parallel or concurrent generational GC.
- That is if we are talking about the performance on a machine running i.e., one PyPy process and nothing else.
- Both a concurrent and parallel GC won't help so much on a machine that is running at 100 percent CPU for example running several PyPy processes.

Future work

- Integrate a generational OMR GC in Pypy.
- Ensure the integrated GC gives some benefits to Pypy.

Outline

Summary

- The OMR GC framework allows us to have reusability when implementing a new GC in a runtime.
- Since it is written in C/C++, there is need to use an efficient integration strategy in Pypy.
- Providing the OMR GC as an external GC after the first transformation in the Pypy translation tool will allow to reduce wrapper overhead and bugs due to manual memory management.

Thank - You