An On-The-Fly Copying Garbage Collection Framework for Jikes RVM

Tomoharu Ugawa
Kochi University of Technology
Richard E. Jones
Carl G. Ritson
University of Kent Computing

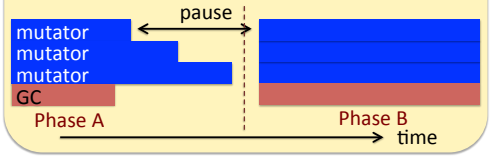
Motivation

GC pauses are undesirable for modern enterprise

➡ Eliminate GC pauses from multi-threaded applications

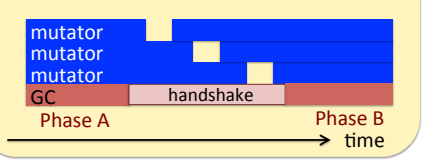
Mostly Concurrent GC (widely used)

need barrier sync. for phase change



On-The-Fly (OTF) GC (we develop)

never stop more than one thread at a time



Challenge on OTF GC: designing correct and efficient write barrier

Contributions

1. Implemented Sapphire OTF GC on widely-used Java VM (Jikes RVM)
2. Developed general framework for OTF, parallel GC
3. Identified a pattern of lagged phase change and fixed a bug in Sapphire
4. Developed efficient concurrent copying method using transactions
5. Support subtleties such as `Object.hashCode()` and weak references

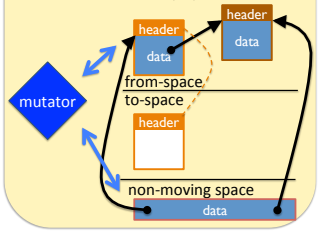
1. Sapphire [Hudson & Moss, 2001]

The only known on-the-fly copying GC, but no full-scale implementation exists

Replication: create semantically equivalent replica behind mutators

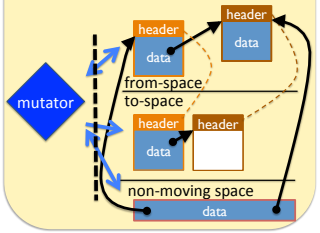
write barrier enforces invariant: no to-space ⟵ from-space pointer

Mark Phase
creates empty "shells"



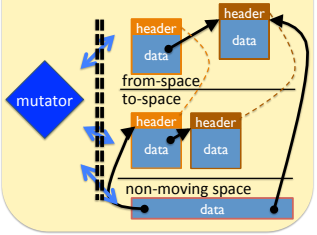
mutator works with from-space and non-moving space

Copy Phase
copies object bodies



write to to-space as well

Flip Phase
flips non-moving space



mutator works with all spaces
flip before writing from-space ptr

2. Lagged Phase Change

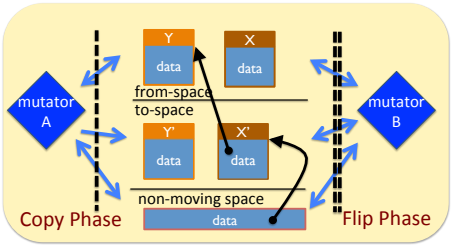
Different phases require different invariants

Sapphire's bug

Mutator A in copy phase
INV: no non-moving⟶ to-space
Mutator B in flip phase
INV: no new non-moving⟶ from-space

1. B stores pointer to to-space object X' to non-moving space
2. A loads X' from non-moving space
3. A stores pointer to from-space object Y to a slot of X'

We introduce intermediate states to prevent conflicts between invariants of adjacent phases



3. Concurrent Copy

Sapphire: compare-and-swap per word

Our solution: copy-fence-verify per object

```
copy-object(X, X')
  for(f : fields(X))
    { buf.f = X.f;  X'.f = forward(X.f); }
  fence;
  for (f : fields(X))
    if (X.f != buf.f) fallback;
```

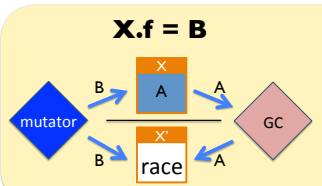
- fewer synchronisation
- sequential memory access

Can use HW transaction for race detection

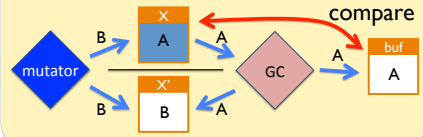
- transaction setup was heavier than fence

  ➡ similar throughput to SW transaction

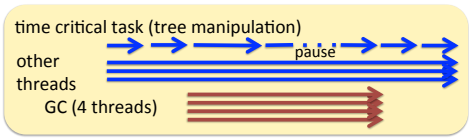
X.f = B



Race detection



Evaluation Result



- Long pauses were very rare (observed regardless of GC)
- Write barrier slowed down mutators to roughly half speed