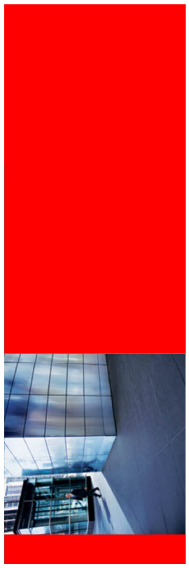


ORACLE

Bengt Rutisson

Garbage Collection in Oracle's JVMs

Who am I?

Computer Science at KTH
Worked at Oberon with BlackBox Component Builder
Worked with Java development at Appear Networks
Last 4.5 years at Oracle
- 4 years with JRockit
- Half a year with HotSpot
- GC/Memory management

Oracle

One of the largest software companies in the world
370 000 customers in 145 countries
All of the 'Furtune top 100' companies use Oracle products
104 500 employees world wide, 29 000 programers

The Stockholm Office

Oracle's largest development office outside USA
Central location on “Söder”
~80 persons
- 500 in Sweden

1 - 2 Master Thesis interns every year
We control Java!

HotSpot vs. JRockit

HotSpot
- The mainstream JVM – reference implementation
- Classic VM → Exact VM → HotSpot
- Client/Server
- Interpreter

JRockit
- I know it the best
- Mostly a server VM
- No interpreter
- Deterministic GC

Competition

Other JVMs
- IBM J9
- Azul (Based on the Open Source code from JRockit)
- Embedded VMs (JBed)
- Vendor specific HotSpot based JVMs
  - HP, Apple, …

Other Platforms
- .Net
- Scala, Ruby, ...

Uniprocessor vs. Multi-core

The trend is definitely towards multi-core

Parallel algorithms are required

Collection Algorithms

Mark and sweep
- Good for large live set

Stop and copy
- Good for a small live set

Reference counting
- Good for a-cyclic structures with infrequent use

We use:
- Mark and sweep for old gen
- Stop and copy for young gen
- Reference counting for meta data (interned strings, classes)

Throughput vs. Pause Times

Throughput “easy”
- Pick an algorithm and optimize it. Parallelize it.
- Only overhead for java threads in generational mode
- Start the collection when an allocation fails

Pausetime more difficult
- Concurrent
- Balance between java execution time and GC exection time
- Overhead for the java threads
- When to start a collection?

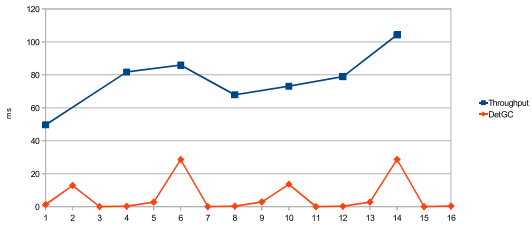
SPECjbb 2005

Nahalem, Linux x64

Throughput	Concurrent
310673	245391

~26% overhead

Turtle



Generations

HotSpot

JRockit

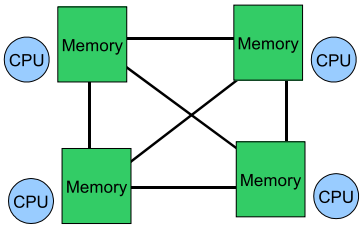
Leaving the Simple Algorithm

- Load balancing
  - Task stealing
  - Task pushing
  - Termination
- Thread local allocations
- Pinned objects
  - JNI
  - Large objects
- Heap holes
  - Non contiguous heap
- Prefetching

Meta Data

- Where to store it?
  - As Java objects in the heap?
    - PermGen
  - As structures on the native heap?
- Reducing memory footprint
- Reducing memory reads
  - Alignment of ClassBlocks

NUMA



Finalizers

- When is a finalizer run?
- Infers overhead
  - Memory footprint
  - GC times
  - Java execution times
- Run only once
- Possible to resurrect dead objects
- Use PhantomReferences instead
- HotSpot: FinalReferences
- JRockit: FinalHandles

Reference Objects

- WeakReference
- SoftReference
- PhantomReference
- FinalReference

Requires special treatment by the GC

Pre-Cleaning

Dirty cards
Meta data
Reference objects

Fragmentation

Free lists
Free memory caches
TLA caches
Dark matter

Compaction

Compaction vs. Evacuation
- Depth first

Partial vs. full compaction
- Requires pointer set

Abortable compactions
- Requires pointer matrix

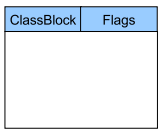
Parallel compaction
Concurrent compaction/evacuation

JRockit DetGC

Cuncurrent collections
No young collections
Aggressive pre-cleaning
Partial and abortable compaction

10ms pauses on 20 GB heap (extreme case)

Object Layout



ClassBlock, 4 bytes

Flag word, 4 bytes

Object alignment, 8 bytes

Only every 8th memory address can contain an object.

Compressed References

Using a heap < 4 GB in a 64 bit address space actually only requires 32 bit pointers
Make sure the heap starts at an even 4GB address
Compressing a reference: take the last 32 bits
Uncompressing a reference: or in the heap base

Due to object alignment the last 3 bits are always 0
Can actually address up to 32 GB heap using 32 bit pointers by shifting in the last three 0:s as needed.

Compressed References

- Playing with the object alignment allows even larger heaps with 32 bit pointers
  - 16 byte alignment → 64 GB heaps
- Not just saving memory
- Compressed references are primarily used for performance.
  - Most instructions are faster on 32 bit words

Hash Codes

- Use the object address as hash code
- When an object moves it needs to have the same hash code
- Three cases
  - o.hashCode() has never been called
  - o.hashCode() called but object not moved
  - o.hashCode() called and object has been moved
- Hash bits in the flag word
- Hash tag before the object

Testing

- Not possible to test all possible inputs
- Long running tests – stability
- Corner cases
  - Races
  - Objects at different borders
  - Compressed references
  - Concurrent modes
- Regressions
  - Throughput vs. pausetime

Supported Platform

- Lots of ports
- Main platforms: Linux, Windows, Solaris
- A lot of combinations to test
- Different compilers: gcc, cl, SunStudio
- Different architectures
  - Different object layout on Sparc
  - Niagara
- Differences in OS
  - Large pages
- Virtual Edition (VE)

Balancing

- Interpreter, quick compile, advanced compile
- OutOfMemoryError
  - Reclaim the last byte or throw OOME fast?
- Heap sizing
  - Allow aggressive heap sizing but don't take over the machine
  - Avoid swaping
- Perfect data expensive to collect
  - Where to compact
  - What collector to use
- Priority and number of concurrent threads
  - Finalizer thread, compiler thread

Benchmarking

- What to measure?
  - Throughput
  - Pausetimes
  - Throughput and pause times
  - Transaction times, average, longest, ...
  - Number of transactions
- Warm up – let the optimizer run
- Variation
- Real world check – always possible to write a benchmark to prove a point

Debugging

- GC crashes due to corrupt memory
  - Often not (this) GC's fault
- asserts
- Verifications before, during and after
- Reproducers

Serviceability

- Mission Control
- Logging
- Jrcmd
- Core dumps and dump files

HotSpot Open Source

- Free binary download:
  http://www.java.com/en/download
- Source available as a Mercurial repository:
  http://hg.openjdk.java.net/jdk7/hotspot/hotspot
- How to download and build:
  http://openjdk.java.net/guide/
- Try it out! Let me know if you want to contribute.

Thanks for Listening!

bengt.rutisson@oracle.com