JITWatch by Example

https://github.com/AdoptOpenJDK/jitwatch

Chris Newland, JCrete 2015

Aim of this session

VM does a good job of optimising most code

Lets you write idiomatic Java

There are some failure modes

Ask Questions! Bring code!

Play along at home!

```
git clone https://github.com/AdoptOpenJDK/jitwatch.git

mvn clean install exec:java


java -XX:+UnlockDiagnosticVMOptions
     -XX:+TraceClassLoading
     -XX:+LogCompilation
     -XX:+PrintAssembly
     MyClass
```

What we could look at

Escape Analysis - heap elimination, lock elision

Lock coarsening

Inlining - {mono|bi|mega}morphic

Intrinsics

Traps and De-optimisations

Other VM languages (Nashorn)

Heap Elimination

NoEscape

Object does not escape the method in which it was created.

“Allocated” somewhere other than the Java heap.

No garbage collection :)

Disappears when method's stack frame is popped.

Reduce overhead of high-churn objects.

Heap Elimination

ArgEscape

Object is passed to another method.

Does not escape current thread.

Cannot eliminate heap allocation.

Locks on this object may be elided.

Heap Elimination

GlobalEscape

Object is accessible by other methods and threads.

Not available for heap elimination or lock elision.

Garbage collected at end of life.

Heap Elimination

Vladimir Ivanov wrote: (paraphrased)

- HotSpot can eliminate allocations in Java heap
  - **Does not "allocate" Java objects on stack**.
    - Would complicate GC logic.

- Allocation elimination by "**object explosion**"
  - Fields of eliminated objects are treated as locals
  - **Register allocator** decides where they are stored:
    - prefers registers, spills to stack if necessary
    - stack slots aggressively reused

Heap Elimination

Examples!

What are the limits of heap elimination?

What kind of objects can’t be heap-eliminated?

Collections? Deep graphs?

What is the cost of register allocator vs heap?

Lock Elision

Eliminate locks on object if determined thread local

Part of Escape Analysis

StringBuffer and Vector

In many scenarios they are used in a thread local manner.

Lock Coarsening

Lock is released and then reacquired where no ***meaningful (?)*** operations occur in between.

Reduces the amount of synchronization work by enlarging an existing synchronized region.

Only used on non-looping control flow.

Examples!

Further work

LogCompilation predicts future performance fragility?

- Bimorphic traps?
- Near inlining limits?
- Near EA register allocator (stack spill) limits?

What additional LogCompilation info would be useful?

Loop unrolls? Register allocator?