Project Loom

October 18, 2018

Agenda

- Introductions
- Current status
- Areas of exploration
- Areas to collaborate

Current status

Current status

- Early exploration/prototype in loom/loom repository
- *cont* branch
- *fibers* branch
- Nothing on tail-calls for now

Continuations prototype

- Continuation API
  - run (to start or continue), yield, test if done, …
  - Nested continuations
- Current implementation/approach
  - Freezing and thawing of stacks
  - Lazy copy
  - Rationale for current approach
  - Performance

Fibers prototype

- Minimal API
  - create, schedule, await termination, …
- LockSupport can park/unpark fibers
- Socket and pipe APIs park fibers rather than block threads in syscalls
- Attempting to park fiber with native frame or synchronized method/block on stack will park carrier thread

I/O APIs

- Mostly focused on network APIs so far
  - java.nio.channels API
  - java.net.Socket
    - Blocking operations park fiber when socket not ready for I/O
    - May advance some refactoring in jdk/jdk to make this easier
- Prototyped console and a few other APIs
- File I/O ignored for now (as file I/O is mostly buffered I/O)

Current limitations

- Can’t yield with native frames on continuation stack
- Can’t yield while holding a monitor
- For fibers, parking may pin the carrier thread
- monitorenter/Object.wait parks carrier thread until unlocked/notified

Ports, testing, …

- macOS and Linux x64 only for now
  - Trying to keep product and fastdebug builds stable
  - Ignoring Windows, AArch64, SPARC at this time
- Testing
  - Basic tests for continuations and fibers
  - Some microbenchmarks

Work in progress

- Performance
- JVM TI support - first step towards debugger support
- Eliminating native frame from Method.invoke and AccessController.doPrivileged
- Exploration …

Areas of exploration

Can Fibers run existing code?

- Big question: Will fibers be able to run *all existing code*?
- Do we completely re-imagine threads?
- We expect to wrestle with this topic for a long time
- Current prototype can run a lot of existing code by emulating currentThread and Thread API

Re-imagine threads

- Project Loom provides big opportunity to re-examine threads
- What is wrong with java.lang.Thread
    - ThreadGroup
    - Context ClassLoader
    - Inheritance: InheritedThreadLocal, TCCL, ACC
    - Deprecated for 20+ years: suspend/resume/stop
    - java.lang.Thread has 22 fields in JDK 11, at least 11 are not interesting to fibers
- and thread pools …
    - ThreadLocals do not work well with thread pools
    - Thread interruption does not work well with thread pools

Strand as superclass for Thread and Fiber

- java.util.concurrent and I/O area have many places that need the *current strand*
  - Waiter queues
  - Checking and re-asserting interrupt status
  - …
- Needs further analysis

Locals

- Assuming *-Djdk.emulateCurrentThread=false*
  - No fiber locals in API at this time
  - No equivalent of TCCL or other locals
- Exploring usages of ThreadLocal
  - Many cases are candidates for *frame locals* (dynamic bind, special variables, …)
  - Some cases use thread locals as approximation to *processor local*

Locals

- Planning to prototype frame locals
  - Lisp has special variables, Clojure has dynamic binding
  - Semantics, API, to be explored
- Also thinking about processor locals
  - Locals keyed on cpu id rather than currentThread
  - Would be nice to drop threadLocalRandomXXX and other fields from Thread

Debugging and serviceability

- Exploring debugging support
  - Basic support in JVM TI to track fiber scheduling, mount and unmount
  - Debugger agent support will take time
- No investigation yet on JMX/java.lang.management and other tool APIs

Structured Concurrency

- Background reading and motivations:
  - Nathaniel J Smith blogs:
    - Notes on structured concurrency, or: Go statement considered harmful
    - Timeouts and cancellation for humans
- Martin Sustrik blogs:
  - Getting rid of state machines (II)
  - Structured Concurrency in High-Level Languages

Structured Concurrency

- Many interesting concepts
  - *block* doesn’t exit until all fibers created in the block have terminated
  - level-triggered and cooperative cancellation
  - composable timeouts and deadlines
  - cancel scopes
- May create a branch in loom repo for prototyping
- Not sure if concepts are mature enough for Java SE

Channels

- Existing APIs
  - SynchronousQueue = non-buffered channel. Designed for high contention rather than fibers
  - ArrayBlockingQueue = buffered channel
- Need to decide if we want to introduce new IPC mechanism for fibers.
  - What is the scope?
  - Select-like features?
  - Specialized for primitive elements?

Future work

- Cloning
- Serialization
- Forced preemption
- Tail calls

Potential areas to collaborate or widen involvement

- Adapt existing code to use Fibers API and provide feedback
- Feedback/input on structured concurrency concepts
- Full-time involvement?
    - Monitors
    - JFR
    - Performance
- More tests would be useful. Benchmarks would be useful at a later time.
- Channels
- Interest in pushing out porting to other architectures until project is further along

Future meeting?

Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.