Java Fork-Join Framework Internals: Work Stealing

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

• Understand how the Java fork-join framework implements worker threads

• Understand how the Java fork-join framework implements work stealing
Working Stealing in a Java Fork-Join Pool
Work Stealing in a Java Fork-Join Pool

- Worker threads only block if there are no tasks available to run.
Worker threads only block if there are no tasks available to run

Blocking threads & cores is costly on modern processors

See Doug Lea’s talk at www.youtube.com/watch?v=sq0MX3fHkro
Worker threads only block if there are no tasks available to run.

Blocking threads & cores is costly on modern processors.

Each worker thread therefore checks other deques in the pool to find other tasks to run.
To maximize core utilization, idle worker threads “steal” work from the tail of busy threads’ deques.

See [docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html](docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html)
Work Stealing in a Java Fork-Join Pool

- To maximize core utilization, idle worker threads “steal” work from the tail of busy threads’ deques.

The worker thread deque to steal from is selected randomly to lower contention.
• Tasks are stolen in FIFO order

See en.wikipedia.org/wiki/FIFO_(computing_and_electronics)
Work Stealing in a Java Fork-Join Pool

- Tasks are stolen in FIFO order
- Minimizes contention w/worker thread owning the deque

Work Stealing in a Java Fork-Join Pool

- Tasks are stolen in FIFO order
  - Minimizes contention w/worker thread owning the deque
  - An older stolen task may provide a larger unit of work

This behavior arises from “divide & conquer” nature of fork-join tasks that split evenly.
Work Stealing in a Java Fork-Join Pool

- Tasks are stolen in FIFO order
- Minimizes contention w/worker thread owning the deque
- An older stolen task may provide a larger unit of work

Larger chunks are pushed onto the deque before smaller chunks.
Tasks are stolen in FIFO order
- Minimizes contention w/worker thread owning the deque
- An older stolen task may provide a larger unit of work

```
List<String> trySplit()
```

```
List<String>1 List<String>2
```

```
WorkQueue

| Sub-Task1,1 |
| Sub-Task1,2 |
| Sub-Task1,3 |
```

```
WorkQueue

| Sub-Task3,3 |
| Sub-Task3,4 |
```

```
poll()
```

A pool of worker threads

T1 T2 T3
Work Stealing in a Java Fork-Join Pool

- Tasks are stolen in FIFO order
  - Minimizes contention w/worker thread owning the deque
  - An older stolen task may provide a larger unit of work
Work Stealing in a Java Fork-Join Pool

- Tasks are stolen in FIFO order
- Minimizes contention with worker thread owning the deque
- An older stolen task may provide a larger unit of work

Thread $T_2$ steals a larger (sub-)task from the end of the deque.
Work Stealing in a Java Fork-Join Pool

- Tasks are stolen in FIFO order
  - Minimizes contention with worker thread owning the deque
- An older stolen task may provide a larger unit of work
  - Enables further recursive decompositions by the stealing thread
• Tasks are stolen in FIFO order
  • Minimizes contention w/worker thread owning the deque
• An older stolen task may provide a larger unit of work
  • Enables further recursive decompositions by the stealing thread

Again, larger chunks are pushed onto the deque before smaller chunks
The WorkQueue deque that implements work-stealing minimizes locking contention
The WorkQueue deque that implements work-stealing minimizes locking contention.

- push() & pop() are only called by the owning worker thread.
Work Stealing in a Java Fork-Join Pool

- The WorkQueue deque that implements work-stealing minimizes locking contention
- `push()` & `pop()` are only called by the owning worker thread
- These methods use wait-free “compare-and-swap” (CAS) operations

• The WorkQueue deque that implements work-stealing minimizes locking contention
  • push() & pop() are only called by the owning worker thread
  • poll() may be called from another worker thread to “steal” a (sub-)task
Work Stealing in a Java Fork-Join Pool

- The WorkQueue deque that implements work-stealing minimizes locking contention
  - `push()` & `pop()` are only called by the owning worker thread
  - `poll()` may be called from another worker thread to “steal” a (sub-)task
  - May not always be wait-free

See [gee.cs.oswego.edu/dl/papers/fj.pdf](gee.cs.oswego.edu/dl/papers/fj.pdf)
The WorkQueue deque that implements work-stealing minimizes locking contention

- `push()` & `pop()` are only called by the owning worker thread
- `poll()` may be called from another worker thread to “steal” a (sub-)task
- May not always be wait-free
  - See “Implementation Overview” comments in the ForkJoinPool source code for details.

See [java8/util/concurrent/ForkJoinPool.java](http://java8/util/concurrent/ForkJoinPool.java)
End of Java Fork-Join Framework Internals: Work Stealing