Overview of the Java Fork-Join Framework

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 processes tasks in parallel
Overview of the Java Fork-Join Pool Computation Model
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool provides a high performance, fine-grained task execution framework for Java data parallelism

```java
public class ForkJoinPool
extends AbstractExecutorService

An ExecutorService for running ForkJoinTasks. A ForkJoinPool provides the entry point for submissions from non-ForkJoinTask clients, as well as management and monitoring operations.

A ForkJoinPool differs from other kinds of ExecutorService mainly by virtue of employing work-stealing: all threads in the pool attempt to find and execute tasks submitted to the pool and/or created by other active tasks (eventually blocking waiting for work if none exist). This enables efficient processing when most tasks spawn other subtasks (as do most ForkJoinTasks), as well as when many small tasks are submitted to the pool from external clients. Especially when setting asyncMode to true in constructors, ForkJoinPools may also be appropriate for use with event-style tasks that are never joined.

A static commonPool() is available and appropriate for most applications. The common pool is used by any ForkJoinTask that is not explicitly submitted to a specified pool. Using the common pool normally reduces resource usage (its threads are slowly reclaimed during periods of non-use, and reinstated upon subsequent use).

For applications that require separate or custom pools, a ForkJoinPool may be constructed with a given target parallelism level; by default, equal to the number of available processors. The pool attempts to maintain enough active (or available) threads by dynamically adding, suspending, or resuming internal worker threads, even if some tasks are stalled waiting to join others. However, no such adjustments are guaranteed in the face of blocked I/O or other unmanaged synchronization. The nested ForkJoinPool.ManagedBlocker interface enables extension of the kinds of synchronization accommodated.
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html)
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool provides a high performance, fine-grained task execution framework for Java data parallelism
- Its parallel computing engine is used by many higher-level frameworks

See [www.infoq.com/interviews/doug-lea-fork-join](http://www.infoq.com/interviews/doug-lea-fork-join)
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by “divide & conquer”

\[
\text{Solve(} \text{problem} \text{)}
\]

\[
\begin{align*}
\text{if} & \quad (\text{problem is small enough}) \\
& \quad \text{solve problem directly} \\
& \quad \text{(sequential algorithm)} \\
\text{else} & \\
& \quad \text{split problem into independent parts} \\
& \quad \text{fork new sub-tasks to solve each part} \\
& \quad \text{join all sub-tasks} \\
& \quad \text{compose result from sub-results}
\end{align*}
\]

Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide & conquer"

\[
\text{Solve(\text{problem})}
\]

\[
\text{if (problem is small enough)}
\]

\[
\text{solve problem directly}
\]

\[
\text{(sequential algorithm)}
\]

\[
\text{else}
\]

\[
\text{split problem into independent parts}
\]

\[
\text{fork new sub-tasks to solve each part}
\]

\[
\text{join all sub-tasks}
\]

\[
\text{compose result from sub-results}
\]
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by “divide & conquer”

\[
\text{Solve}(\text{problem})
\]

\[
\begin{align*}
\text{if} & \quad (\text{problem is small enough}) \\
& \quad \text{solve problem directly} \\
& \quad \text{(sequential algorithm)} \\
\text{else} & \\
& \quad \text{split problem into independent parts} \\
& \quad \text{fork new sub-tasks to solve each part} \\
& \quad \text{join all sub-tasks} \\
& \quad \text{compose result from sub-results}
\end{align*}
\]
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by “divide & conquer”, e.g.
  - Splitting a task into sub-tasks

See [en.wikipedia.org/wiki/Fork-join_model](en.wikipedia.org/wiki/Fork-join_model)
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide & conquer", e.g.
- Splitting a task into sub-tasks
- A task creates sub-tasks by fork()’ing

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html#fork
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide & conquer", e.g.
  - Splitting a task into sub-tasks
    - A task creates sub-tasks by fork()’ing

A (sub-)task only splits itself into (more) sub-tasks if the work is sufficiently large at that level
The fork-join pool supports a style of parallel programming that solves problems by “divide & conquer”, e.g.

- Splitting a task into sub-tasks
- Applying sub-tasks in parallel
The fork-join pool supports a style of parallel programming that solves problems by “divide & conquer”, e.g.

- Splitting a task into sub-tasks
- Applying sub-tasks in parallel

Overview of the Java Fork-Join Pool Computation Model

Implemented by fork-join framework, Java execution environment, OS, & hardware
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by “divide & conquer”, e.g.
  - Splitting a task into sub-tasks
  - Applying sub-tasks in parallel
  - Sub-tasks run in parallel on different cores
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by “divide & conquer”, e.g.
  - Splitting a task into sub-tasks
  - Applying sub-tasks in parallel
  - Sub-tasks run in parallel on different cores

Performance typically increases as the # of cores increases
The fork-join pool supports a style of parallel programming that solves problems by “divide & conquer”, e.g.

- Splitting a task into sub-tasks
- Applying sub-tasks in parallel
  - Sub-tasks run in parallel on different cores
  - Sub-tasks can also run concurrently in different threads on a single core

This configuration may not enhance performance unless sub-tasks are I/O bound.
The fork-join pool supports a style of parallel programming that solves problems by “divide & conquer”, e.g.

- Splitting a task into sub-tasks
- Applying sub-tasks in parallel
- Combining sub-task results
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by “divide & conquer”, e.g.
  - Splitting a task into sub-tasks
  - Applying sub-tasks in parallel
  - Combining sub-task results
- `join()` waits for a sub-task to finish

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html#join](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html#join)
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by “divide & conquer”, e.g.
  - Splitting a task into sub-tasks
  - Applying sub-tasks in parallel
  - Combining sub-task results
    - `join()` waits for a sub-task to finish

See upcoming lesson on “The Java Fork-Join Pool: Key Methods in ForkJoinTask”
The fork-join pool supports a style of parallel programming that solves problems by “divide & conquer”, e.g.

- Splitting a task into sub-tasks
- Applying sub-tasks in parallel
- Combining sub-task results
  - `join()` waits for a sub-task to finish
  - & merges the results
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by “divide & conquer”, e.g.
  - Splitting a task into sub-tasks
  - Applying sub-tasks in parallel
  - Combining sub-task results
    - `join()` waits for a sub-task to finish
    - & merges the results

Partial (sub-)results are merged into a final result
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by “divide & conquer”, e.g.
  - Splitting a task into sub-tasks
  - Applying sub-tasks in parallel
  - Combining sub-task results
    - `join()` waits for a sub-task to finish
    - `join()` occurs in a single thread at each level

As a result, there’s typically no need for synchronizers during the joining phase
End of Overview of the Java Fork-Join Framework