COMP 322: Fundamentals of Parallel Programming

Lecture 20: Java Concurrent Collections

Vivek Sarkar
Department of Computer Science
Rice University
vsarkar@rice.edu
Announcements

• Graded midterm exams can be picked up from Amanda Nokleby in Duncan Hall room 3137

• Homework 5 will be sent out by tomorrow
  – Homework 6 dates will be adjusted accordingly
Acknowledgments for Today’s Lecture

• Lecture 20 handout
• “Java’s Collection Framework” slides by Rick Mercer
• “Introduction to Concurrent Programming in Java”, Joe Bowbeer, David Holmes, OOPSLA 2007 tutorial slides
  — Contributing authors: Doug Lea, Brian Goetz
• “Java Concurrency Utilities in Practice”, Joe Bowbeer, David Holmes, OOPSLA 2007 tutorial slides
  — Contributing authors: Doug Lea, Tim Peierls, Brian Goetz
Table 2: Examples of common isolated statement idioms and their equivalent AtomicInteger implementations (Corrected version)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Rank computation:</td>
<td>AtomicInteger rank = new AtomicInteger();</td>
<td>AtomicInteger rank = new AtomicInteger();</td>
</tr>
<tr>
<td>rank = new ...; rank.count = 0;</td>
<td>. . .</td>
<td>. . .</td>
</tr>
<tr>
<td>. . .</td>
<td>r = rank.incrementAndGet();</td>
<td>r = rank.incrementAndGet();</td>
</tr>
<tr>
<td>isolated r = ++rank.count;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2) Work assignment:</td>
<td>AtomicInteger rem = new AtomicInteger(n);</td>
<td>AtomicInteger rem = new AtomicInteger(n);</td>
</tr>
<tr>
<td>rem = new ...; rem.count = n;</td>
<td>. . .</td>
<td>. . .</td>
</tr>
<tr>
<td>. . .</td>
<td>r = rem.getAndDecrement();</td>
<td>r = rem.getAndDecrement();</td>
</tr>
<tr>
<td>isolated r = rem.count--;</td>
<td>if ( r &gt; 0 ) . . .</td>
<td>if ( r &gt; 0 ) . . .</td>
</tr>
<tr>
<td>3) Counting semaphore:</td>
<td>AtomicInteger sem = new AtomicInteger();</td>
<td>AtomicInteger sem = new AtomicInteger();</td>
</tr>
<tr>
<td>sem = new ...; sem.count = 0;</td>
<td>. . .</td>
<td>. . .</td>
</tr>
<tr>
<td>. . .</td>
<td>r = sem.incrementAndGet();</td>
<td>r = sem.incrementAndGet();</td>
</tr>
<tr>
<td>isolated r = ++sem.count;</td>
<td>. . .</td>
<td>. . .</td>
</tr>
<tr>
<td>. . .</td>
<td>r = sem.decrementAndGet();</td>
<td>r = sem.decrementAndGet();</td>
</tr>
<tr>
<td>isolated r = --sem.count;</td>
<td>. . .</td>
<td>. . .</td>
</tr>
<tr>
<td>. . .</td>
<td>isolated s = sem.count; isZero = (s==0);</td>
<td>isolated s = sem.count; isZero = (s==0);</td>
</tr>
<tr>
<td>4) Sum reduction:</td>
<td>AtomicInteger sum = new AtomicInteger();</td>
<td>AtomicInteger sum = new AtomicInteger();</td>
</tr>
<tr>
<td>sum = new ...; sum.val = 0;</td>
<td>. . .</td>
<td>. . .</td>
</tr>
<tr>
<td>. . .</td>
<td>sum.addAndGet(x);</td>
<td>sum.addAndGet(x);</td>
</tr>
</tbody>
</table>
Java Collection Framework

— The *Java Collections Framework* is a unified architecture for representing and manipulating collections. It has:

- **Interfaces**: abstract data types (ADTs) representing collections of objects
- **Implementations**: concrete implementations of the collection interfaces
- **Algorithms**: methods that perform useful computations, such as searching and sorting

These algorithms are said to be *polymorphic*: the same method can be used on different implementations.
Java Collection interfaces
Implementations of Collection Interfaces

- A collection class
  - implements an ADT as a Java class
  - implements all methods of the interface
  - selects appropriate instance variables
  - can be instantiated

- Some well-known collection classes used in sequential Java programs
  - List: ArrayList, LinkedList, Vector
  - Map: HashMap, TreeMap
  - Set: TreeSet, HashSet
Working with Collections in a Parallel Program

Different approaches:

1. Restrict access to a single task \(\Rightarrow\) no modification needed

2. Ensure that each call to a public method is “synchronized” (isolated) with respect to other calls \(\Rightarrow\) excessive synchronization

3. Use specialized implementations that minimize serialization across public methods \(\Rightarrow\) Java Concurrent Collections

- We will focus on three java.util.concurrent classes that can be used freely in HJ programs, analogous to Java Atomic Variables
  - ConcurrentHashMap, ConcurrentLinkedQueue, CopyOnWriteArraySet

- Other j.u.c. classes can be used in standard Java, but not in HJ
  - ArrayBlockingQueue, CountDownLatch, CyclicBarrier, DelayQueue, Exchanger, FutureTask, LinkedBlockingQueue, Phaser, PriorityBlockingQueue, Semaphore, SynchronousQueue
Java Collection interfaces

![Diagram showing the hierarchy of collection interfaces in Java.](image-url)
The Java Map Interface

- Map describes a type that stores a collection of key-value pairs
- A Map associates (maps) a key the it's value
- The keys must be unique
  - the values need not be unique
- Useful for implementing software caches (where a program stores key-value maps obtained from an external source such as a database), dictionaries, sparse arrays, ...

- A Map is often implemented with a hash table (HashMap)
- Hash tables attempt to provide constant-time access to objects based on a key (String or Integer)
  - key could be your Student ID, your telephone number, social security number, account number, ...
- The direct access is made possible by converting the key to an array index using a hash function that returns values in the range 0 ... ARRAY_SIZE-1, typically by using a (mod ARRAY_SIZE) operation
Collisions

- A good hash method
  - executes quickly
  - distributes keys equitably
- But you still have to handle collisions when two keys have the same hash value
  - the hash method is not guaranteed to return a unique integer for each key
    example: simple hash method with "baab" and "abba"
- There are several ways to handle collisions
  - Consider separate chaining hashing
An Array of LinkedList Objects
(to support Collisions)

An array of linked lists

0

1

2

321

365
java.util.concurrent.concurrentHashMap

- Implements ConcurrentMap sub-interface of Map
- Allows read (traversal) and write (update) operations to overlap with each other
- Some operations are atomic with respect to each other e.g.,
  - get(), put(), putIfAbsent(), remove()
- Aggregate operations may not be viewed atomically by other operations e.g.,
  - putAll(), clear()
- Expected degree of parallelism can be specified in ConcurrentHashMap constructor
  - ConcurrentHashMap(initialCapacity, loadFactor, concurrencyLevel)
  - A larger value of concurrencyLevel results in less serialization, but a larger space overhead for storing the ConcurrentHashMap
Concurrent Collection Performance

Throughput in Thread-safe Maps

- **ConcurrentHashMap**
- **ConcurrentSkipListMap**
- **SynchronizedHashMap**
- **SynchronizedTreeMap**

**Java 6 B77**
- 8-Way System
- 40% Read Only
- 60% Insert
- 2% Removals
Example usage of ConcurrentHashMap in org.mirrorfinder.model.BaseDirectory

```
public abstract class BaseDirectory extends BaseItem implements Directory {
    Map files = new ConcurrentHashMap();

    public Map getFiles() {
        return files;
    }

    public boolean has(File item) {
        return getFiles().containsValue(item);
    }

    public Directory add(File file) {
        String key = file.getName();
        if (key == null) throw new Error(...);
        getFiles().put(key, file);
        return this;
    }

    public Directory remove(File item) throws NotFoundException {
        if (has(item)) {
            getFiles().remove(item.getName());
        }
        else throw new NotFoundException("can’t_remove_unrelated_item");
    }
}
```

Java Collection interfaces
java.util.concurrent.ConcurrentLinkedQueue

• Queue interface added to java.util
  - interface Queue extends Collection and includes
    boolean offer(E x); // same as add() in Collection
    E poll(); // remove head of queue if non-empty
    E remove(o) throws NoSuchElementException;
    E peek(); // examine head of queue without removing it

• Non-blocking operations
  — Return false when full
  — Return null when empty

• Fast thread-safe non-blocking implementation of Queue interface: ConcurrentLinkedQueue
Example usage of ConcurrentLinkedQueue in org.apache.catalina.tribes.io.BufferPool15Impl

```java
class BufferPool15Impl implements BufferPool.BufferPoolAPI {
    protected int maxSize;
    protected AtomicInteger size = new AtomicInteger(0);
    protected ConcurrentLinkedQueue queue = new ConcurrentLinkedQueue();

    public XByteBuffer getBuffer(int minSize, boolean discard) {
        XByteBuffer buffer = (XByteBuffer) queue.poll();
        if (buffer != null) size.addAndGet(-buffer.getCapacity());
        if (buffer == null) buffer = new XByteBuffer(minSize, discard);
        else if (buffer.getCapacity() <= minSize) buffer.expand(minSize);
        ...
        return buffer;
    }

    public void returnBuffer(XByteBuffer buffer) {
        if ((size.get() + buffer.getCapacity()) <= maxSize) {
            size.addAndGet(buffer.getCapacity());
            queue.offer(buffer);
        }
    }
}
```

Listing 2: Example usage of ConcurrentLinkedQueue in org.apache.catalina.tribes.io.BufferPool15Impl
Java Collection interfaces
java.util.concurrent.CopyOnWriteArraySet

- Set implementation optimized for case when sets are not large, and read operations dominate update operations in frequency.
- This is because update operations such as add() and remove() involve making copies of the array.
  - Functional approach to mutation.
- Iterators can traverse array “snapshots” efficiently without worrying about changes during the traversal.
Example usage of CopyOnWriteArraySet in org.norther.tammi.spray.freemarker.DefaultTemplateLoader

```java
public class DefaultTemplateLoader implements TemplateLoader, Serializable {
    private Set resolvers = new CopyOnWriteArraySet();
    public void addResolver(ResourceResolver res) {
        resolvers.add(res);
    }
    public boolean templateExists(String name) {
        for (Iterator i = resolvers.iterator(); i.hasNext();)
            if (((ResourceResolver) i.next()).resourceExists(name)) return true;
        return false;
    }
    public Object findTemplateSource(String name) throws IOException {
        for (Iterator i = resolvers.iterator(); i.hasNext();)
            CachedResource res = ((ResourceResolver) i.next()).getResource(name);
            if (res != null) return res;
        return null;
    }
}
```

Listing 3: Example usage of CopyOnWriteArraySet in org.norther.tammi.spray.freemarker.DefaultTemplateLoader