Parallel Programming with Java 7

Wolf Schlegel
March 2012
Who is ThoughtWorks?

- 20+ offices
- 8 countries
- 1800+ people
- On & off shore
- US$200m revenue
Interested in something completely different?
Visit join.thoughtworks.com
Some theory
New kids on the block
A lot of practice
Wrap up
Some theory 13:30 – 14:00
New kids on the block 14:00 – 14:30
All hands on deck 14:30 – 16:30
Wrap up 16:30 – 17:00
Part 1
Some theory
Why parallel programming?

**Grand challenge problems**
- Modelling DNA structures
- Weather forecasting

**Parallel programming splits problems into parts**
- Parts are solved in parallel by multiple processors
No more free lunch

* Source: http://de.wikipedia.org/wiki/Mooresches_Gesetz
Partitioning & divide-and-conquer

Data partitioning is applied to data (aka domain decomposition)
Functional decomposition concurrently executes independent functions
Less common than data partitioning

Divide-and-conquer divides a problem into similar sub-problems
Sub-problems have the same form as the original problem
Can be repeated recursively
Which way is faster?

Solve the problem

or

Split the problem

Solve partial problem

Merge results
Amdahl's law (1/3)

\[ S = \text{speedup factor} \quad S = \text{function}(n,f) \quad \text{where} \]

\[ n = \text{number of processors/cores} \]

\[ f = \text{fraction of the computation that cannot be parallelised} \]
Amdahl’s law (2/3)

S = speedup factor = function(n, f) where

n = number of processors/cores

f = fraction of the computation that cannot be parallelised

\[ S(n) = \frac{\text{single processor execution time}}{n \text{ processor execution time}} = \frac{n}{1 + (n - 1)f} \]

\[ \lim_{n \to \infty} S(n) = \frac{1}{f} \]
Amdahl's law (3/3)
How to benefit from multiple CPUs/cores in Java

Implicitly — magic done under the hood by the JVM
Explicitly — using Java 7’s fork/join framework

This session focuses on the fork/join framework
Testing fork/join

Assert that...

... the core computation works as expected

... the complete task works as expected

... splitting the task does not corrupt input data

... merging results behaves as expected
Part 2
New kids on the block
New classes on the block

Thread
  └── ForkJoinWorkerThread

Callable<V>

Executor
  └── ExecutorService
    └── ForkJoinPool

Future<V>
  └── ForkJoinTask<V>
    └── RecursiveAction
    └── RecursiveTask<V>
Anatomy of the fork/join interaction (1/2)

ForkJoinTask

RecursiveAction RA1

Fork

RA2

RA4

RA2

RA5

RA6

RA3

RA7

Join

RA1

Result
Part 3
All hands on deck
How to implement a ForkJoinTask generally

if (my portion of the work is small enough)
   do the work **directly**
else
   **split** my work into two pieces and **fork** them
   **merge/join** the results
A guided example: Calculating checksums
Setting up your environment

- Sources and slides are on the stick
- Follow the instructions in the README file
Exercise 1: Bucket sort

- Implement a sequential and a parallel version of bucket sort
- Put elements into buckets that represent ranges
- Sort and concatenate buckets

Exercise 1: Bucket sort as pseudo code

```plaintext
function bucketSort(array, n) is
    buckets ← new array of n empty lists

    for i = 0 to (length(array)-1) do
        insert array[i] into buckets[whichBucket(array[i])]

    for i = 0 to n - 1 do
        nextSort(buckets[i])

    return the concatenation of buckets[0], ..., buckets[n-1]
```

Source: http://en.wikipedia.org/wiki/Bucket_sort
Exercise 1: One way to realise bucket sort

**ISorter**
public List<T> sort(List<T> list)

**IBucketSortHelper**
public void sortBuckets(SortedMap<T, List<T>> buckets)

**BucketSorter**

**SequentialBucketSortHelper**

**ActionBasedBucketSortHelper**

**TaskBasedBucketSortHelper**

**ParallelBucketSortAction**

**ParallelBucketSortTask**
Exercise 2: Tuning the parallel bucket sort

- Buckets of equal width work well when the list to sort is evenly populated
- When the distribution of elements is skewed, the parallel bucket sort may degrade (why?)
- Run your parallel bucket sort with different kinds of lists (i.e. evenly populated ones and skewed ones)
- Improve the performance of your parallel bucket sort if required
Exercise 3: Merge sort

- Implement a sequential and a parallel version of merge sort
  - If the list is of length 0 or 1 the list is already sorted
  - Divide the unsorted list into two sublists of about half the size
  - Sort each sublist recursively by re-applying the merge sort
  - Merge the two sublists back into one sorted list

Source: http://en.wikipedia.org/wiki/Merge_sort
Exercise 3: Merge sort as pseudo code (1/2)

function merge_sort(m)
    if length(m) ≤ 1
        return m
    var list left, right, result
    var integer middle = length(m) / 2
    for each x in m up to middle
        add x to left
    for each x in m after or equal middle
        add x to right
    left = merge_sort(left)
    right = merge_sort(right)
    result = merge(left, right)
    return result
Exercise 3: Merge sort as pseudo code (2/2)

function merge(left,right)
    var list result
    while length(left) > 0 or length(right) > 0
        if length(left) > 0 and length(right) > 0
            if first(left) ≤ first(right)
                append first(left) to result
                left = rest(left)
            else
                append first(right) to result
                right = rest(right)
        else if length(left) > 0
            append first(left) to result
            left = rest(left)
        else if length(right) > 0
            append first(right) to result
            right = rest(right)
    end while
    return result

Source: http://en.wikipedia.org/wiki/Merge_sort
Exercise 4: Searching strings in text files

* Implement a parallel search for a string in a text file
* The search string can be assumed to contain no whitespace (i.e. no spaces or tabs)
* Have a look at the following class:

```
com.thoughtworks.fjw.search.SimpleStringSearchTest
```

- The class contains examples for string based searching and reading from a text file
Exercise 5: Searching strings revisited

- Revisit your solution for the previous exercise
- Drop the assumption so that search strings can contain whitespace
- Consider how dropping the assumption affects the way you partition the text to search
Thanks to Fabian for co-authoring the workshop

Thank you
How can we help?

ThoughtWorks is a global custom software solutions consultancy trusted by many of the world’s leading businesses with their most complex and critical systems.

We deliver consulting grounded in delivery expertise, build custom applications and help organisations across all market sectors to drive IT efficiency – working to an exceptionally high standard.

Contact us

Wolf Schlegel
+49 173 543 7465
wolf.schlegel@thoughtworks.com
www.thoughtworks.com
Backup slides
Amdahl’s law (3/3)
Amdahl's law (3/3)
Thread state diagram