ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Parallel Sorting

Parallel Programming in Scala

Viktor Kuncak

Merge Sort

We will implement a parallel merge sort algorithm.

1. recursively sort the two halves of the array in parallel
2. sequentially merge the two array halves by copying into a temporary array
3. copy the temporary array back into the original array

The `parMergeSort` method takes an array, and a maximum depth:

Merge Sort

We will implement a parallel merge sort algorithm.

1. recursively sort the two halves of the array in parallel
2. sequentially merge the two array halves by copying into a temporary array
3. copy the temporary array back into the original array

The `parMergeSort` method takes an array, and a maximum depth:

```
def parMergeSort(xs: Array[Int], maxDepth: Int): Unit = {
```

Allocating an Intermediate Array

We start by allocating an intermediate array:

```
val ys = new Array[Int](xs.length)
```

At each level of the merge sort, we will alternate between the source array `xs` and the intermediate array `ys`.

Sorting the Array

```
def sort(from: Int, until: Int, depth: Int): Unit = {
  if (depth == maxDepth) {
    quickSort(xs, from, until - from)
  } else {
    val mid = (from + until) / 2
    parallel(sort(mid, until, depth + 1), sort(from, mid, depth + 1))
```

Sorting the Array

```scala
def sort(from: Int, until: Int, depth: Int): Unit = {
  if (depth == maxDepth) {
    quickSort(xs, from, until - from)
  } else {
    val mid = (from + until) / 2
    parallel(sort(mid, until, depth + 1), sort(from, mid, depth + 1))

    val flip = (maxDepth - depth) % 2 == 0
    val src = if (flip) ys else xs
    val dst = if (flip) xs else ys
    merge(src, dst, from, mid, until)
  }
}
sort(0, xs.length, 0)
```

Merging the Array

Given an array `src` consisting of two sorted intervals, merge those interval into the `dst` array:

```
def merge(src: Array[Int], dst: Array[Int],
  from: Int, mid: Int, until: Int): Unit
```

The `merge` implementation is sequential, so we will not go through it.

Merging the Array

Given an array `src` consisting of two sorted intervals, merge those interval into the `dst` array:

```
def merge(src: Array[Int], dst: Array[Int],
  from: Int, mid: Int, until: Int): Unit
```

The `merge` implementation is sequential, so we will not go through it.

How would you implement `merge` in parallel?

Copying the Array

```scala
def copy(src: Array[Int], target: Array[Int],
  from: Int, until: Int, depth: Int): Unit = {
  if (depth == maxDepth) {
    Array.copy(src, from, target, from, until - from)
  } else {
    val mid = (from + until) / 2
    val right = parallel(
      copy(src, target, mid, until, depth + 1),
      copy(src, target, from, mid, depth + 1)
    )
  }
}
if (maxDepth % 2 == 0) copy(ys, xs, 0, xs.length, 0)
```

Demo

Let's compare the performance of `parMergeSort` against the Scala `quicksort` implementation.