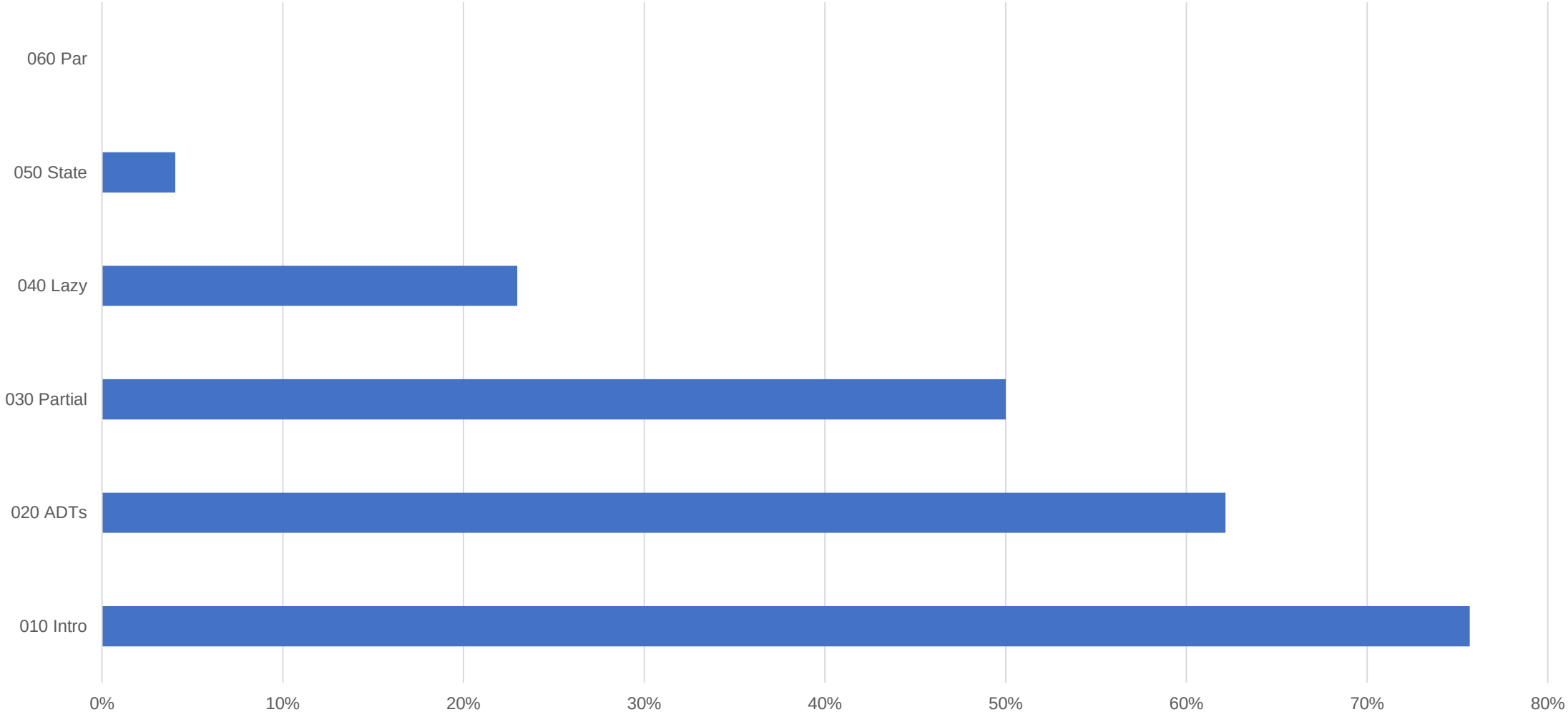
Advanced Programming 060 Par

Code Excerpts

Homework Pulse (% submitted)



Java's Runnable Interface

```
trait Runnable {
    def run: Unit
}

class Thread (r :Runnable) {
    def start: Unit
    def join: Unit
}
```

Java's Future Interface

```
class ExecutorService {
    def submit[A] (a: Callable[A]) : Future[A]
}

trait Future[A] {
    def get: A
}
```

Sum a sequence with Par

```
def sum (ints: IndexedSeq[Int]) : Par[Int] =
      if (ints.size <= 1)
            Par.unit(ints.headOption.getOrElse (0))
      else {
            val (l,r) = ints.splitAt (ints.length/2)
            map2 (fork(sum(l)), fork(sum(r))) (_ + _)
      }
```

A Par Implementation

```
type Par[A] = ExecutorService => Future[A]
def run[A] (s: ExecutorService => Future[A]) :A
// executor service is a Java class, and so is Future.
// Future has a property get that extracts the value (blocking)

def unit[A] (a: A): Par[A] = (es :ExecutorService) => UnitFuture(a)

private case class UnitFuture[A] (get: A) extends Future[A] {
  def get (timeout: Long, units: [[TimeUnit]]) = get
       // arguments are ignored in the simple implementation
}

def map2[A,B,C] (a: Par[A], b: Par[B]) (f: (A,B) => C): Par[C] =
  (es: ExecutorService) => { val af = a(es)
                             val bf = b(es)
                             UnitFuture (f(af.get, bf.get))
                           }
```

Par implemention (ctd.)

```
def fork[A] (a: => Par[A]) :Par [A] =
     es => es.submit (
          new Callable[A] { def call = a(es).get }
     )
```

parMap

```
def parMap[A.B] (ps: List[A]) (f: A => B): Par[List[B]] =
fork {
      val fbs: List[Par[B]] = ps.map(asyncF(f))
      sequence(fbs)
}
```