Another example is to look for Mersenne prime numbers. This is done with the following, again wrapping the computation in `Parallelize`.

```
In[1]:=  Parallelize[Select[ Range[2000], PrimeQ[2^# - 1] &]]
Out[1]=  {2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279}
```

This shows that the first 15 Mersenne prime numbers have been found.

When you get to this stage you should be ready to start carrying out parallel computation in *Mathematica*.

Using Your Own Functions in Parallel Computations

The previous example worked by simply wrapping a parallelizable expression in `Parallelize[…]`. If the expressions involves not only built-in functions, but functions you defined yourself, some preparatory work is necessary.

Definitions for symbols to be evaluated on the parallel kernels, other than built-in ones, need to be *distributed* to all kernels before they can be used.

We define a predicate that tests whether $2^n - 1$ is prime.

```
In[14]:=  mersenneQ[n_] := PrimeQ[2^n - 1]
```

We distribute the definition to all parallel kernels.

```
In[15]:=  DistributeDefinitions[mersenneQ]
```

Now we can use it as part of a parallel computation.

```
In[16]:=  Parallelize[Select[Range[1000], mersenneQ]]
Out[16]=  {2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607}
```

What happens if we forget to distribute definitions for a parallel computation?

This definition is the same as `mersenneQ` above.

```
In[17]:=  m2[n_] := PrimeQ[2^n - 1]
```

The parallel kernels do not know the definition, so it never returns `True`.

```
In[18]:=  Parallelize[Select[Range[1000], m2]]
Out[18]=  {}
```

In many cases the computation seems to work anyway, but if you analyze its performance, you should see that it was not in fact evaluated as fast as it should.

This computation gives the right result, but it is not faster than a normal `Table` would be.

```
In[22]:=  ParallelTable[m2[i], {i, 10 000, 10 020}]
Out[22]=  {False, False, False, False, False, False, False, False, False, False,
           False, False, False, False, False, False, False, False, False, False, False}
```

The reason it seems to work is that the unknown function m2 does not evaluate on the parallel kernels, so the expressions *m2*[10 000], *m2*[10 001], … are sent back, and they then evaluate on the master kernel, where the definition is known.