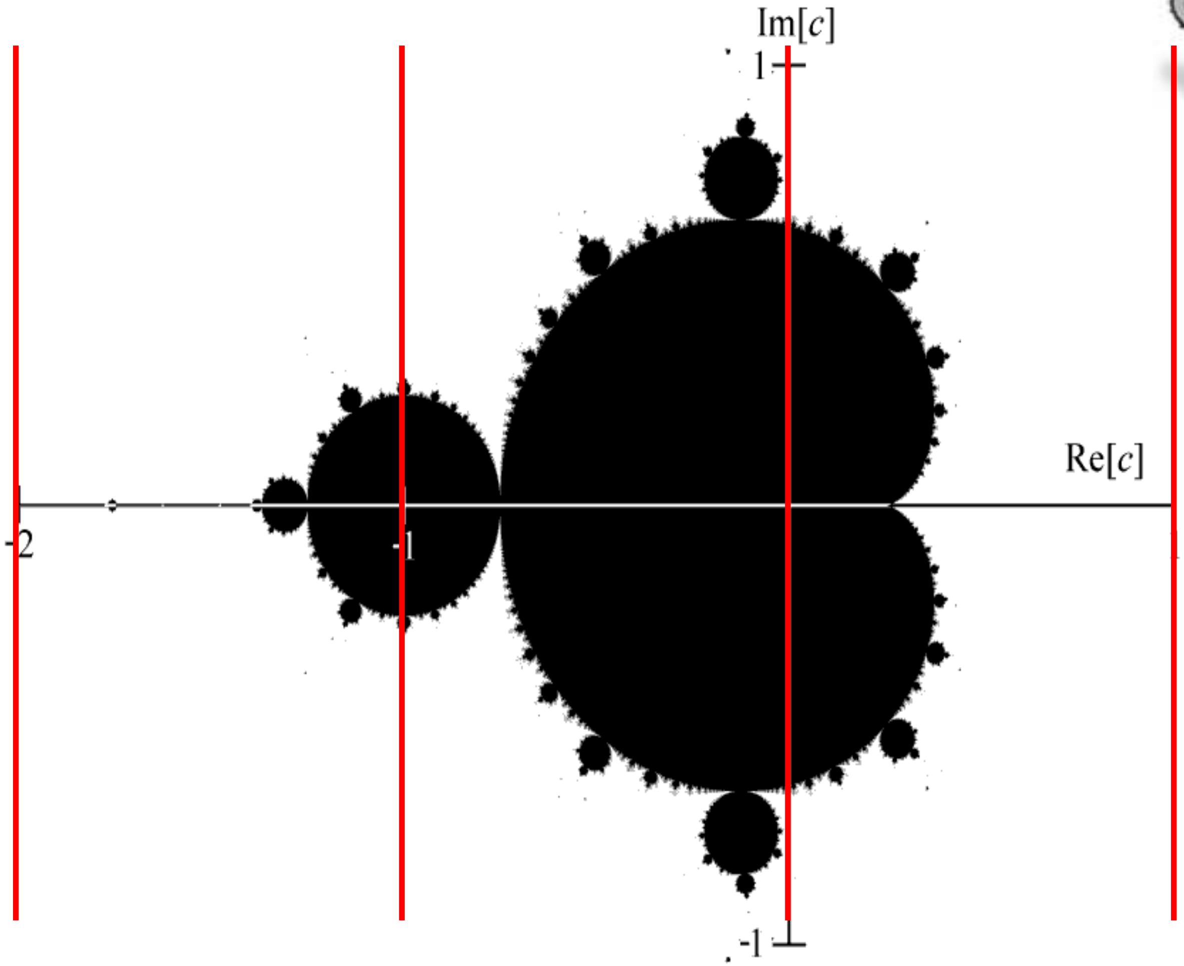
Fachhochschule
Nordwestschweiz

Idea: Cut Problem into Slices



MandelSlice

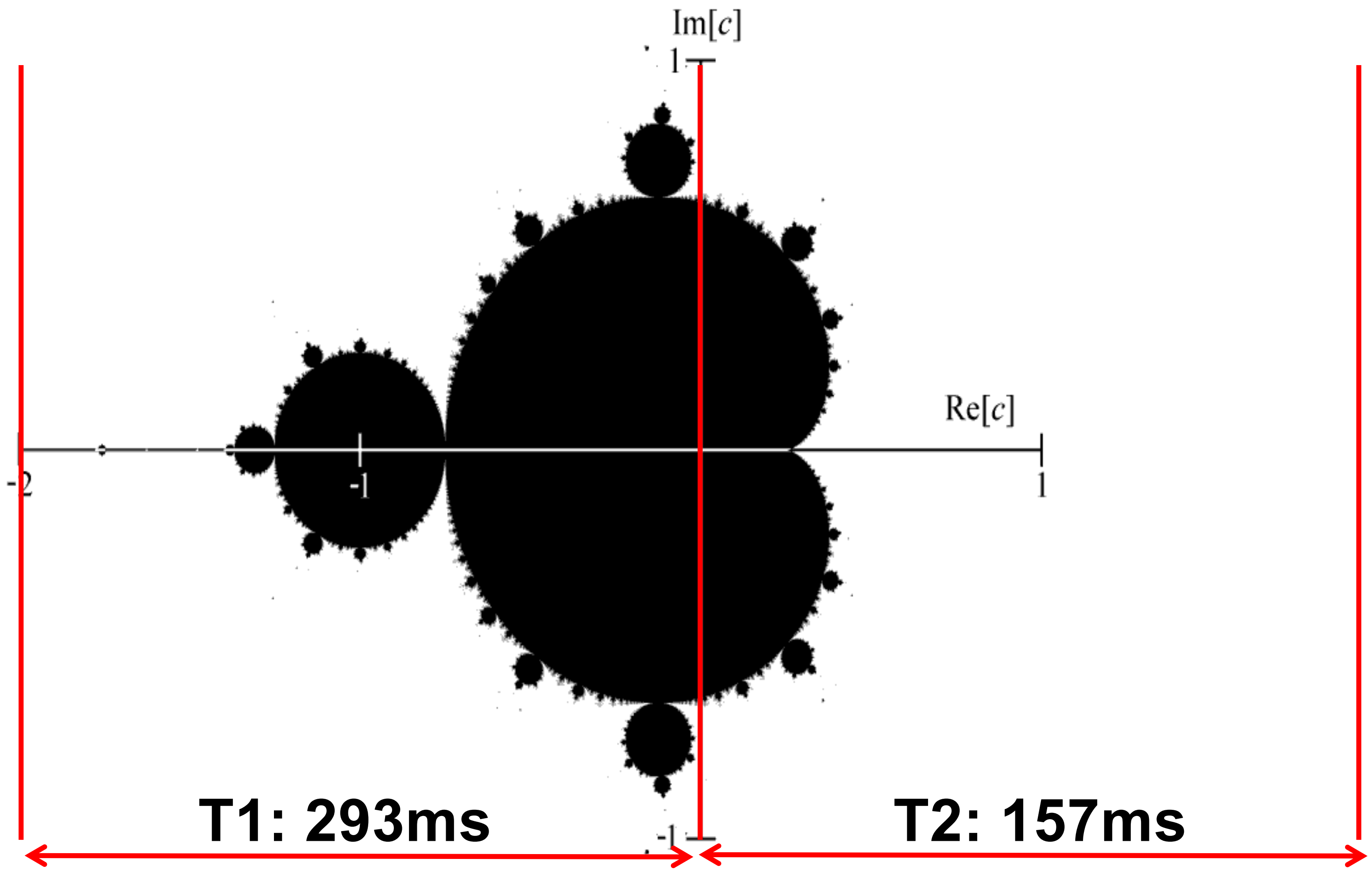
```
class MandelSlice implements Runnable {
   private final int startX, endX;          private final Plane plane;
   private final PixelPainter painter;      private CancelSupport cancel;

   private MandelSlice(int startX, int endX, PixelPainter pp, Plane p, CancelSupport cs) {
     this.startX = startX; this.endX = endX; this.painter = pp; this.plane = p; this.cancel = cs;
   }

   @Override public void run() {
      double half = plane.length / 2; double reMin = plane.center.r - half;
      double imMax = plane.center.i + half; double step = plane.length / IMAGE_LENGTH;

      for (int x = startX; x < endX && !cancel.isCancelled(); x++) {
          double re = reMin + x * step;
          for (int y = 0; y < IMAGE_LENGTH; y++) {
             double im = imMax - y * step;
             int iterations = mandel(re, im);
             painter.paint(x, y, getColor(iterations));
          }
      }
   }
}
```

Fachhochschule
Nordwestschweiz

One Thread per Slice

```
public static void computeParallel(PixelPainter painter, Plane plane, CancelSupport cancel) {
   final int N = Runtime.getRuntime().availableProcessors();
   final int heightPerThread = IMAGE_LENGTH / N;

   final List<Thread> threads = new ArrayList<Thread>(N);

   // Create N slices
   for (int i = 0; i < N; i++) {
      final int startX = i * heightPerThread;
      final int endX = (i < N - 1) ? startX + heightPerThread : IMAGE_LENGTH;
      Thread thread = new Thread(new MandelSlice(startX, endX, painter, plane, cancel));
      threads.add(thread);
      thread.start(); // Start all Threads
   }

   for (Thread thread : threads) {
      try {
         thread.join(); // Wait for all Threads
      } catch (InterruptedException e) {
         /* Ignored */
      }
   }
}
```

Problem: Not all Slices are Equal



Non Ideal Solution: Many Threads

```
public static void computeParallel(PixelPainter painter, Plane plane, CancelSupport cancel) {
   final int N = 256;
   final int heightPerThread = IMAGE_LENGTH / N;

   final List<Thread> threads = new ArrayList<Thread>(N);

   // Create N slices
   for (int i = 0; i < N; i++) {
      final int startX = i * heightPerThread;
      final int endX = (i < N - 1) ? startX + heightPerThread : IMAGE_LENGTH;
      Thread thread = new Thread(new MandelSlice(startX, endX, painter, plane, cancel));
      threads.add(thread );
      thread.start(); // Start all Threads
   }

   for (Thread thread : threads) {
      try {
         thread.join(); // Wait for all Threads
      } catch (InterruptedException e) {
         /* Ignored */
      }
   }
}
```

Scheduling Overhead!

Ideal Solution: Separate Tasks from Workers

```
public static void computeParallelPool(PixelPainter painter, Plane plane,
                            CancelSupport cancel) {
   final int N_THREADS = Runtime.getRuntime().availableProcessors();
   final int N_SLICES = 64;

   final int widthPerThread = IMAGE_LENGTH / N_SLICES;
   final List<MandelSlice> tasks = new LinkedList<MandelSlice>();

   for (int i = 0; i < N_SLICES; i++) {
      final int startX = i * widthPerThread;
      final int endX = (i < N_SLICES - 1) ? startX + widthPerThread : IMAGE_LENGTH;
      tasks.add(new MandelSlice(startX, endX, painter, plane, cancel));
    }
   ...
```

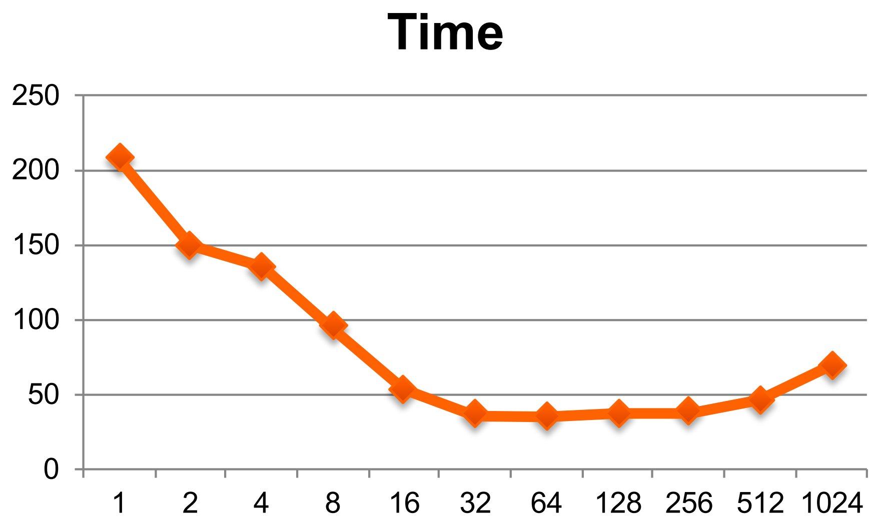
Fachhochschule
Nordwestschweiz

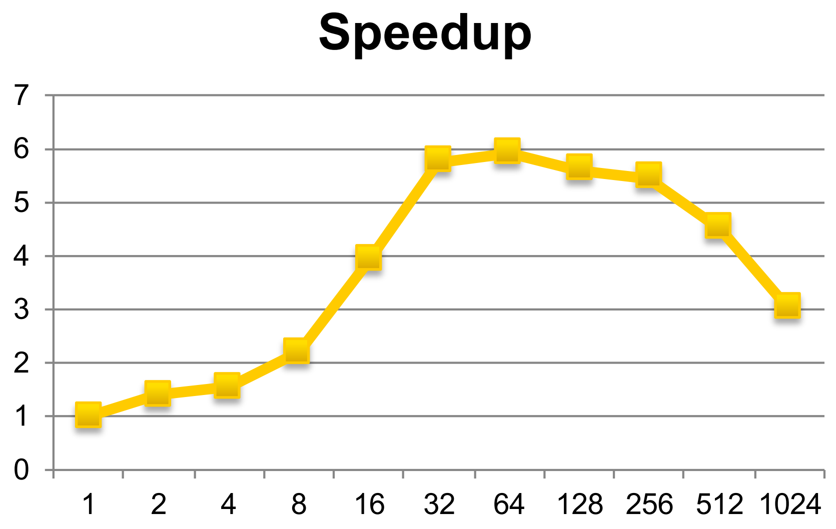
Ideal Solution: Separate Tasks from Workers

```
    final List<Thread> threads = new ArrayList<Thread>(N_THREADS);
    for (int i = 0; i < N_THREADS; i++) {
       Thread thread = new Thread(() -> {
          boolean running = true;
          while (running) {
             MandelSlice slice = null;
             synchronized (tasks) {
                if (!tasks.isEmpty()) {
                   slice = tasks.remove(0);
                } else {
                   running = false;
                }
             }
             if (slice != null) { slice.run(); }
       });

       threads.add(thread);
       thread.start();
    }
    for (Thread thread: threads) {
       try {thread.join(); } catch (InterruptedException e) {}
    }

 }
```

Execution Times

- Sequential (1 Thread, 1 Slice) : 471 ms
- Parallel (8 Threads, 8 Slices) : 198 ms
- Parallel (256 Threads, 256 Slices) : 151 ms
- Parallel (8 Threads, 256 Slices): 137 ms


Time


Speedup