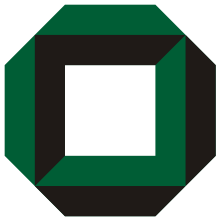
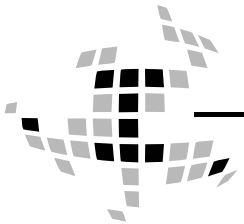


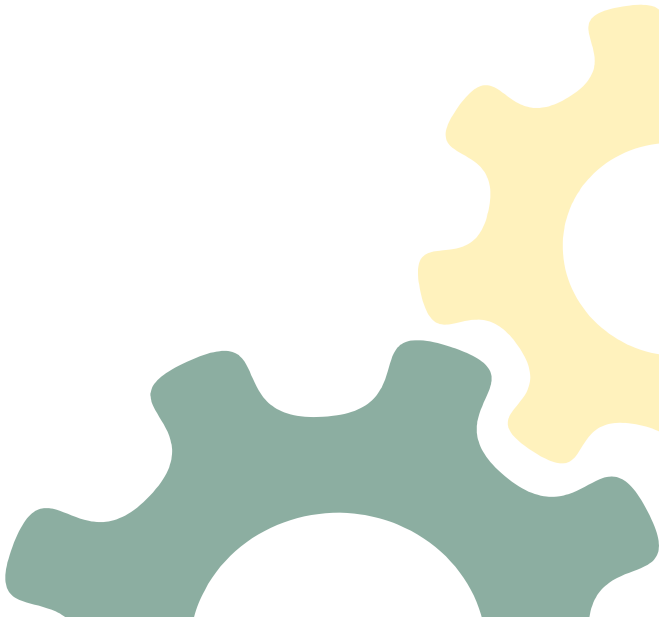
Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

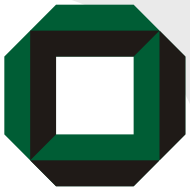
Softwaretechnik 1
Übung 5

02.07.2009



Fakultät für **Informatik**
Lehrstuhl für Programmiersysteme





Aufgabe 1a)

Zeichnen Sie die komplette Vererbungshierarchie der Klasse **`BufferedOutputStream`** als UML-Klassendiagramm. Die Klasse **`Object`** können Sie hierbei weglassen. Geben Sie in Ihrem UML-Klassendiagramm öffentliche Methoden immer dann an, wenn sie zum ersten Mal deklariert werden oder (neu) implementiert werden. Eventuell geworfene Ausnahmen können Sie vernachlässigen. Attribute (Fields), Konstruktoren und nicht öffentliche Methoden brauchen Sie ebenfalls nicht einzuzeichnen.

Aufgabe 1a)

Overview Package **Class** Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Java™ Platform Standard Ed. 6

java.io

Class BufferedOutputStream

```
java.lang.Object
  └ java.io.OutputStream
      └ java.io.FilterOutputStream
          └ java.io.BufferedOutputStream
```

All Implemented Interfaces:

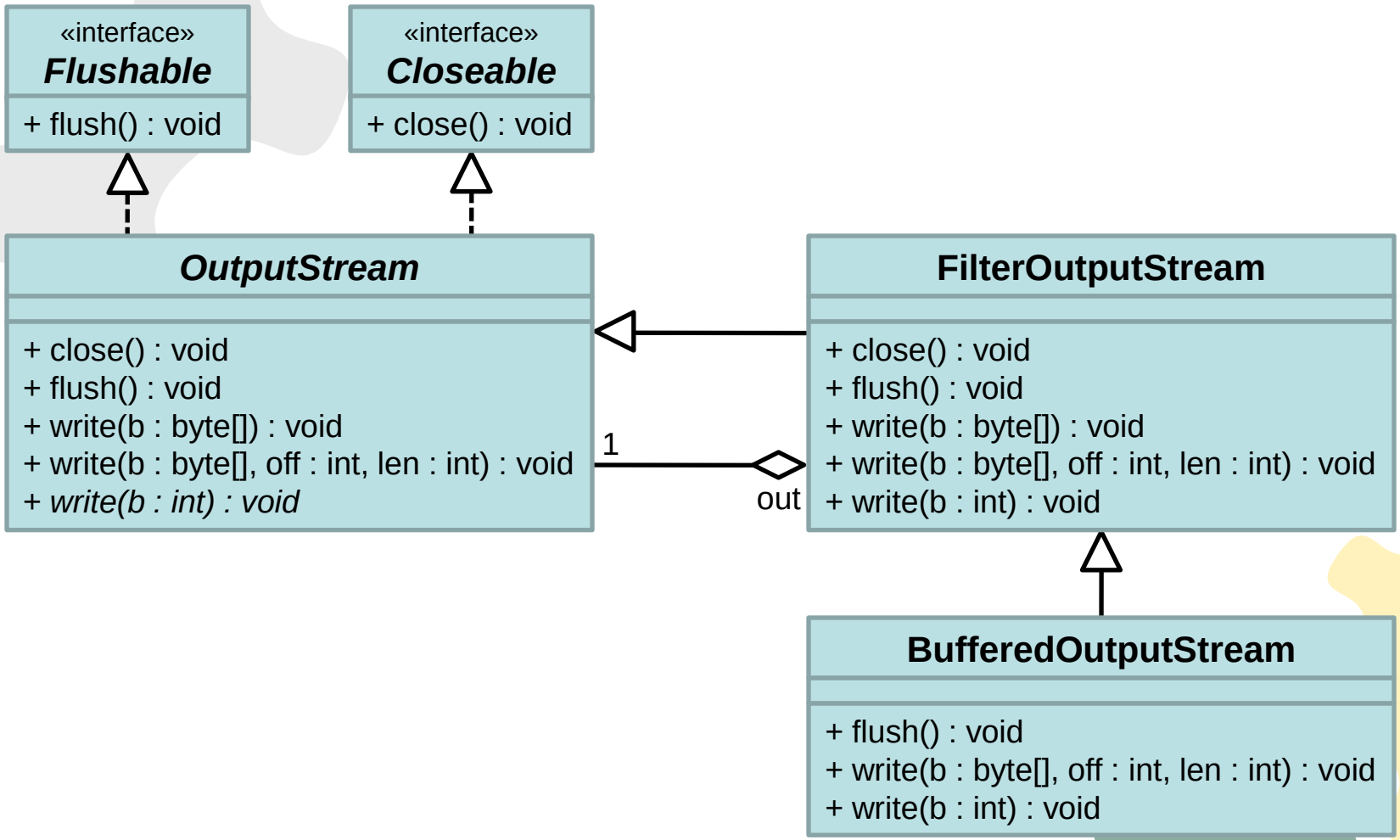
Closeable, Flushable

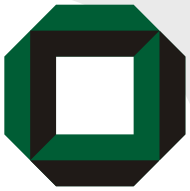
Aufgabe 1a)

Method Summary	
void	**flush**() Flushes this buffered output stream.
void	**write**(byte[] b, int off, int len) Writes `len` bytes from the specified byte array starting at offset `off` to this buffered output stream.
void	**write**(int b) Writes the specified byte to this buffered output stream.

Methods inherited from class java.io.FilterOutputStream
close, write

Aufgabe 1a)

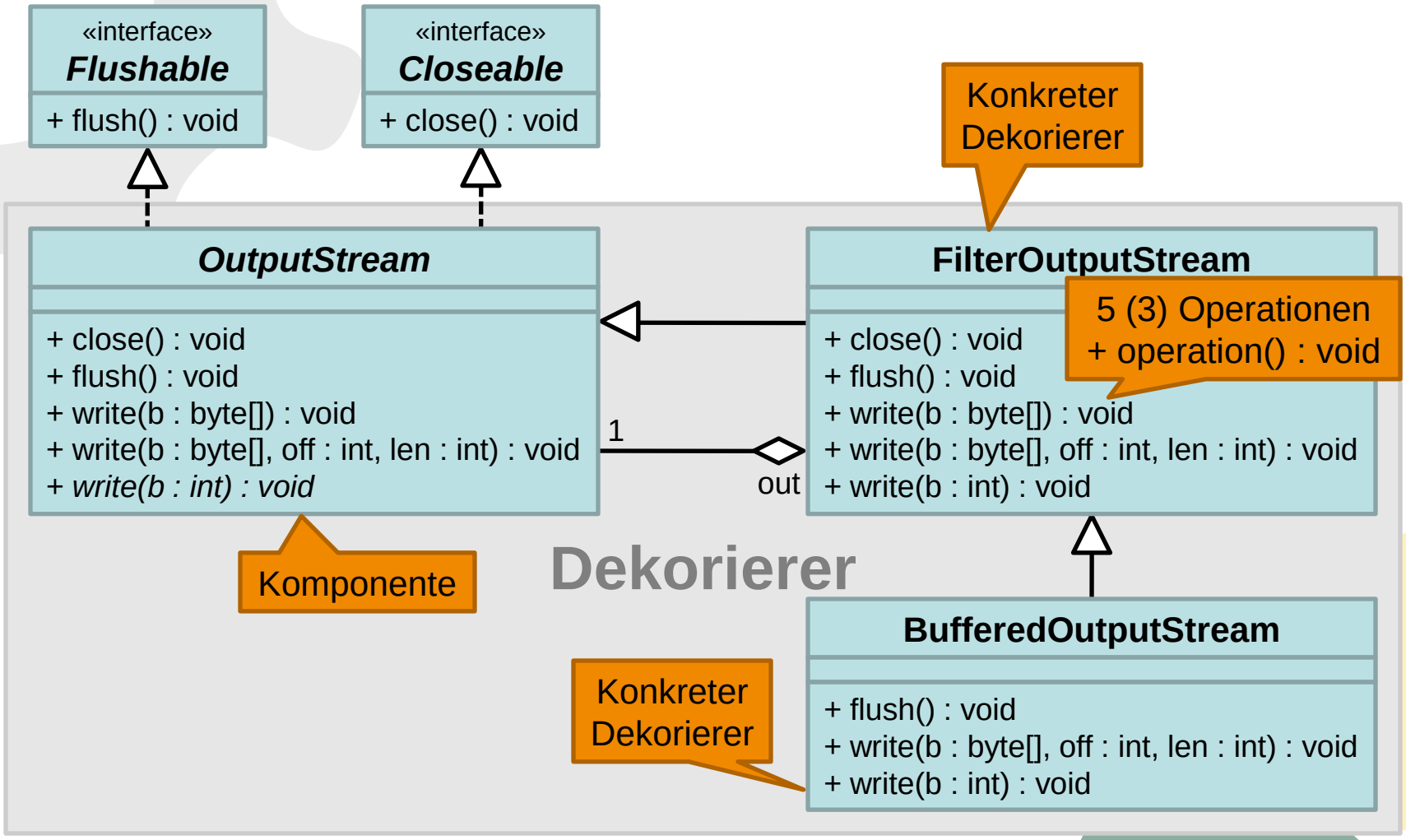

«interface»
Flushable
+ flush() : void
«interface»
Closeable
+ close() : void
OutputStream
+ close() : void
+ flush() : void
+ write(b : byte[]) : void
+ write(b : byte[], off : int, len : int) : void
+ write(b : int) : void
1
out
FilterOutputStream
+ close() : void
+ flush() : void
+ write(b : byte[]) : void
+ write(b : byte[], off : int, len : int) : void
+ write(b : int) : void
BufferedOutputStream
+ flush() : void
+ write(b : byte[], off : int, len : int) : void
+ write(b : int) : void

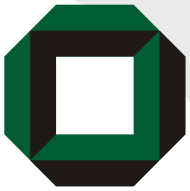


Aufgabe 1b)

Welches Entwurfsmuster ist in dem UML-Klassendiagramm zu finden? Kennzeichnen alle zum Entwurfsmuster gehörenden Klassen/Schnitt-stellen im UML-Klassendiagramm. Weisen Sie diesen die entsprechenden Rollen aus dem Entwurfsmuster zu. Ordnen Sie den Methoden im UML-Klassendiagramm – soweit möglich – ihre Entsprechungen im Entwurfsmuster zu.

Aufgabe 1b)

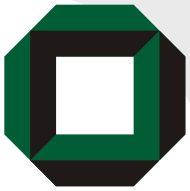

«interface»
Flushable
+ flush() : void
«interface»
Closeable
+ close() : void
Konkreter Dekorierer
OutputStream
+ close() : void
+ flush() : void
+ write(b : byte[]) : void
+ write(b : byte[], off : int, len : int) : void
+ write(b : int) : void
FilterOutputStream
5 (3) Operationen
+ operation() : void
+ close() : void
+ flush() : void
+ write(b : byte[]) : void
+ write(b : byte[], off : int, len : int) : void
+ write(b : int) : void
1
out
Komponente
Dekorierer
BufferedOutputStream
Konkreter Dekorierer
+ flush() : void
+ write(b : byte[], off : int, len : int) : void
+ write(b : int) : void



Aufgabe 2a)

- Gegeben:
  - Matrix `a`, Vektor `b`, `t` Fäden
  - Folgender Quelltext wird zur Aufteilung der Matrix verwendet:

```
int s = a.length / t;
int r = a.length % t; // Rest bestimmen
for (int i = 0; i < t; i++) {
  start = i * s;
  end = (i + 1) * s;
  if (i == t - 1) {
    // Letzter Faden übernimmt
    // zusätzlich die übrigen Elemente
    end += r;
  }
  // ... Arbeiter erstellen und Threads starten
}
```

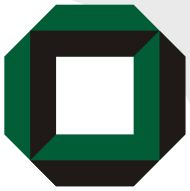


Aufgabe 2a)

- Wieso ist diese alternative Implementierung zur Berechnung der Blockgrößen nicht geeignet?
  - Annahme: Es gibt `N` Fäden und `a` hat `N-1` Zeilen:
    - `N-1` Fäden bekommen keine Elemente zugewiesen.
    - Der `N`-te Faden führt Berechnung auf allen Elementen durch.
  - Beispielrechnung:
    - `N = 51`, `a` hat `50` Zeilen
    - `r = 50 % 51 = 50`
    - Die Fäden 1 bis 50 erhalten `50 / 51 = 0` Elemente
    - Der Faden 51 erhält die Elemente von `start = 50 * 0 = 0` bis `end = (51 * 0) + 50 = 50`

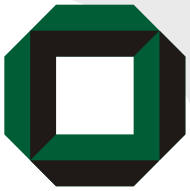
Ganzzahldivision!

Beachte: Index beginnt in Java bei 0!



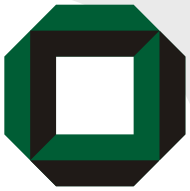
Aufgabe 2a)

- Beispielrechung (Algorithmus aus der VL):
  - `N = 51`, `a` hat `50` Zeilen
  - Die Fäden 1 bis 50 erhalten `50 / 51 = 1` Element
  - Der Faden 51 erhält die Elemente von `start = 50 * 1 = 50` bis `end = (51 * 1) = 51`
  - Der Faden 51 berechnet in diesem Fall nichts.



Aufgabe 2a)

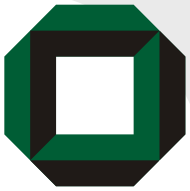
- Unter welchen Annahmen führt auch dieser Algorithmus zu einer ungünstigen Aufteilung?
  - Es gibt weniger „Blöcke“ als Fäden.
    Annahme: **a** hat **N** Zeilen und es gibt **2*N** Fäden
    - Die ersten **N** Fäden bekommen genau ein Element zugewiesen.
    - Die letzten **N** Fäden haben hingegen nichts zu tun.
  - Start eines neuen Fadens ist teurer als der Gewinn durch den höheren Grad der Parallelität der Operation **o** auf der Matrix **a**.
  - Hängt ab von…
    - Kosten für Start eines Fadens
    - Minimaler Blockgröße
    - Kosten für Operation auf minimaler Blockgröße
    - Cacheeffekten



Aufgabe 2b)

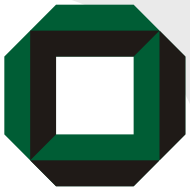
- Betrachten Sie die folgende Implementierung einer Barriere:

```
public class Barrier {
  final int cnt;   // Anzahl teilnehmender Fäden
  int arrived = 0; // Anzahl der angekommenen Fäden

  public synchronized void sync()
      throws InterruptedException {
    arrived++;
    if (arrived == cnt) {
      arrived = 0;
      notifyAll();
    } else {
      while (arrived > 0) {
        wait();
      }
    }
  }
}
```



Aufgabe 2b)

- Erklären Sie in wenigen Sätzen, warum diese Barriere nicht korrekt funktioniert. Geben Sie dazu ein möglichst einfaches Beispiel an, welches das Problem aufdeckt.
- Annahme:
  - N Fäden synchronisieren sich an der Barriere.
  - Fäden 1 − N-1 legen sich schlafen.
  - Faden N betritt Rücksetzen-Block.
  - Fäden 1 − N-1 bekommen notifyAll() zu spät mit.
  - Faden N überholt die anderen → Deadlock.



Aufgabe 2b)

Monitor	T1	T2
T1 acquire	`sync() {`	
	`  arrived++; // arrived = 1`	
	`  } else {`	
	`  while (arrived > 0) {`	
T1 release	`  wait();`	
T2 acquire		`sync() {`
		`  arrived++; // arrived = 2`
		`  if (arrived == cnt) {`
		`  arrived = 0;`
		`  notifyAll();`
T2 release		`}}`
T2 acquire		`sync() {`
		`  arrived++; // arrived = 1`
		`  } else {`
		`  while (arrived > 0) {`
T2 release		`  wait();`
T1 acquire	`  wait();`	
	`while (arrived > 0) {`	
T1 release	`  wait();`	

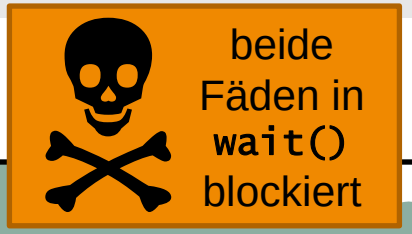
Blockiert in 1. Generation

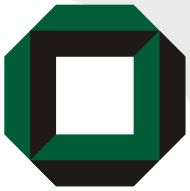
Verlässt 1. Generation

Betritt 2. Generation

innerhalb von `wait()`

Immer noch in 1. Generation





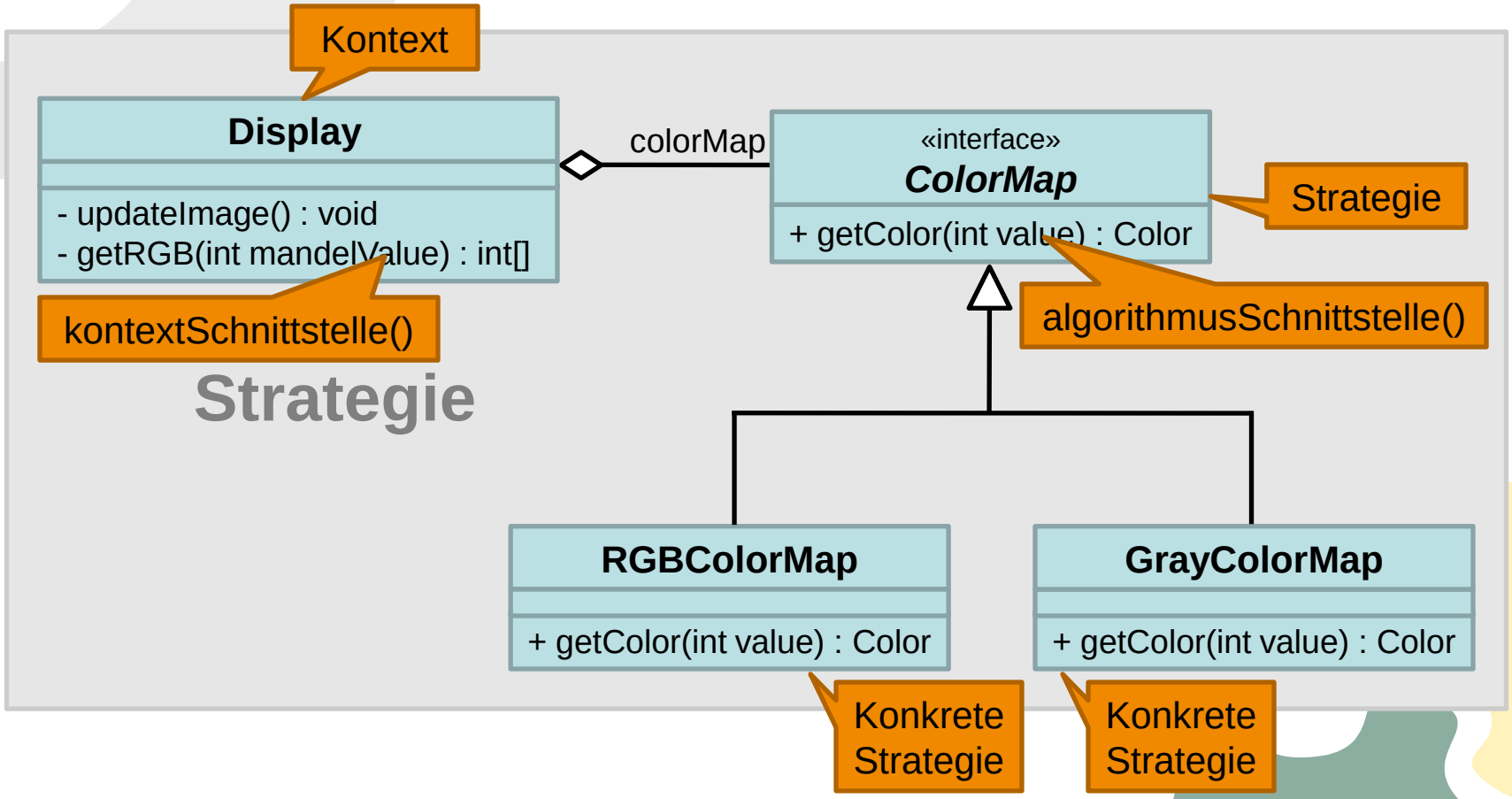
Aufgabe 2b)

```
public class Barrier {
    private final int cnt;
    private int arrived = 0;
    private boolean state = true;

    public Barrier(int cnt) {
        this.cnt = cnt;
    }

    public synchronized void sync() throws InterruptedException {
        arrived++;
        if (arrived == cnt) {
            arrived = 0;
            state = !state; notifyAll();
        } else {
            boolean stateBeforeWait = state;
            while (stateBeforeWait == state)
                wait();
        }
    }
}
```

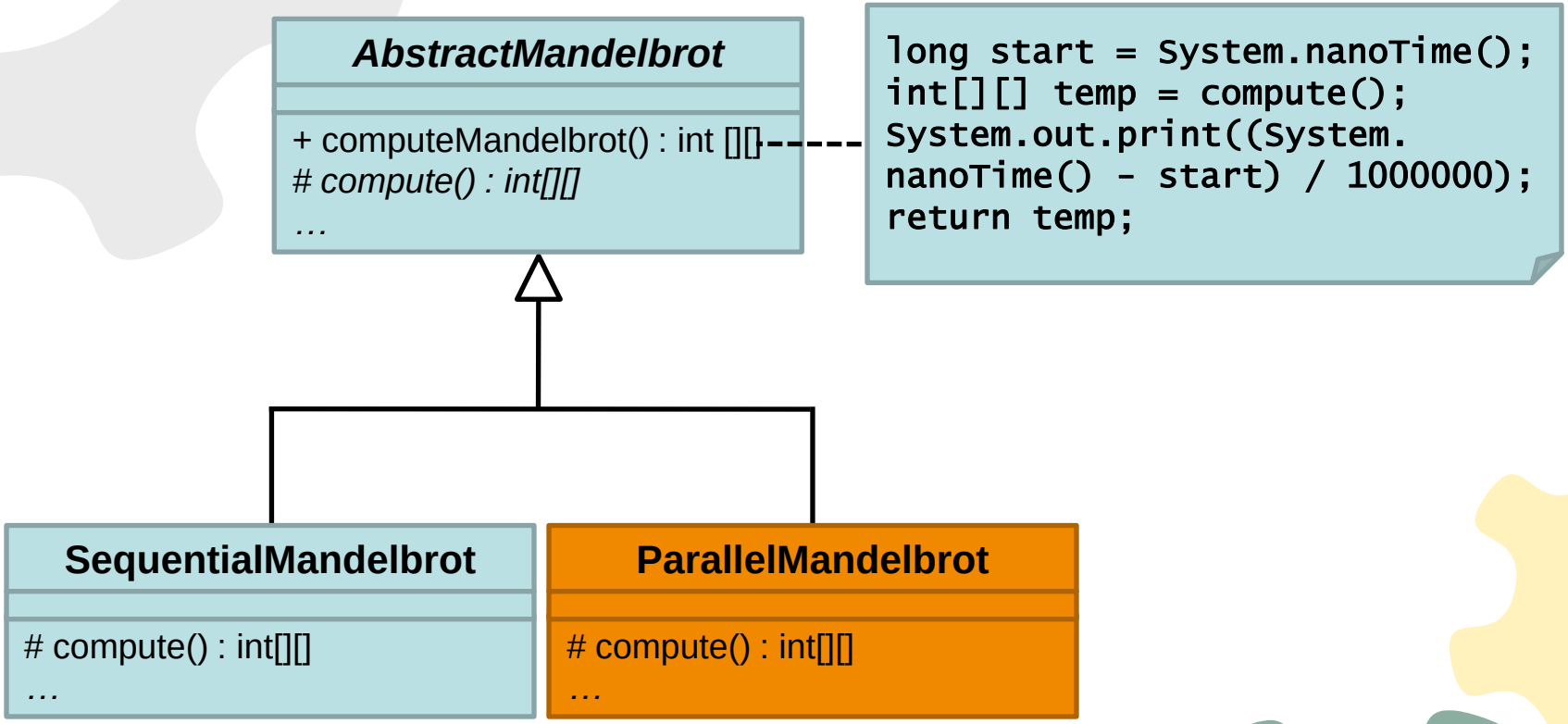
Idee: Markiere „Generationen“.
Der letzte Faden stellt ein Fähnchen um und initiiert neue „true“ oder „false – Generation“.

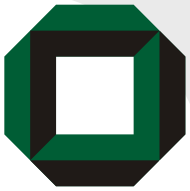
Aufgeweckte Fäden können nicht mehr in der While-Schleife blockiert bleiben. Prüfe „Generationswechsel“.

Aufgabe 3a)


Kontext
Display
- updateImage() : void
- getRGB(int mandelValue) : int[]
kontextSchnittstelle()
colorMap
«interface»
ColorMap
+ getColor(int value) : Color
Strategie
algorithmusSchnittstelle()
Strategie
RGBColorMap
+ getColor(int value) : Color
GrayColorMap
+ getColor(int value) : Color
Konkrete Strategie
Konkrete Strategie

Aufgabe 3b)


AbstractMandelbrot
+ computeMandelbrot() : int [][]
# compute() : int[][]
…
long start = System.nanoTime();
int[][] temp = compute();
System.out.print((System.
nanoTime() - start) / 1000000);
return temp;
SequentialMandelbrot
compute() : int[][]
…
ParallelMandelbrot
compute() : int[][]
…



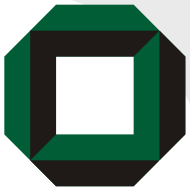
Aufgabe 3b)

```
protected int[][] compute() {
  int w = (int) Math.ceil((double) width / numberOfThreads);
  CyclicBarrier barrier = new CyclicBarrier(numberOfThreads + 1);
  Runnable r;
  Thread[] threads = new Thread[numberOfThreads];
  int[][] result = new int[width][height];
  for (int i = 0; i < numberOfThreads; i++) {
    int fromW = i * w;
    int toW = Math.min((i + 1) * w, width);
    r = new RunnableMandelbrot(barrier, fromW, toW, 0, height, result);
    threads[i] = new Thread(r);
    threads[i].start();
  }
  try {
    barrier.await();
  } catch (InterruptedException e) { e.printStackTrace();
  } catch (BrokenBarrierException e) { e.printStackTrace(); }
  return result;
}
```

Synchronisation mittels Barriere

Fäden abspalten

Auf abgespaltene Fäden warten

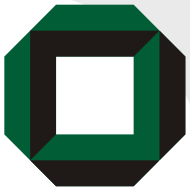


Aufgabe 3b)

```
private class RunnableMandelbrot implements Runnable { […]

  public RunnableMandelbrot(CyclicBarrier barrier, int fromW, int toW,
    int fromH, int toH, int[][] target) { […]
  }

  public void run() {
    double cReal, cImag;
    for (int x = fromW; x < toW; x++) {
      for (int y = fromH; y < toH; y++) {
        cReal = translateReal(x);
        cImag = translateImag(y);
        target[x][y] = computeMandelbrotPoint(cReal, cImag);
      }
    }
    try { barrier.await();
    } catch (InterruptedException e) { e.printStackTrace();
    } catch (BrokenBarrierException e) { e.printStackTrace(); }
  }
}
```

Barriere wird übergeben

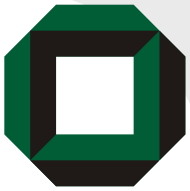
Auf alle anderen Fäden warten



Aufgabe 3b)

```
public final class Start {
  […]

  private static AbstractMande          createMandelbrot() {
    AbstractMandelbrot man      brot;
    int s = Math.abs(s      ze);
    if (parallel && threads > 1) {
      mandelbrot = new ParallelMandelbrot(s, s, threads);
    } else {
      mandelbrot = new SequentialMandelbrot(s, s);
    }
    return mandelbrot;
  }
  […]
}
```

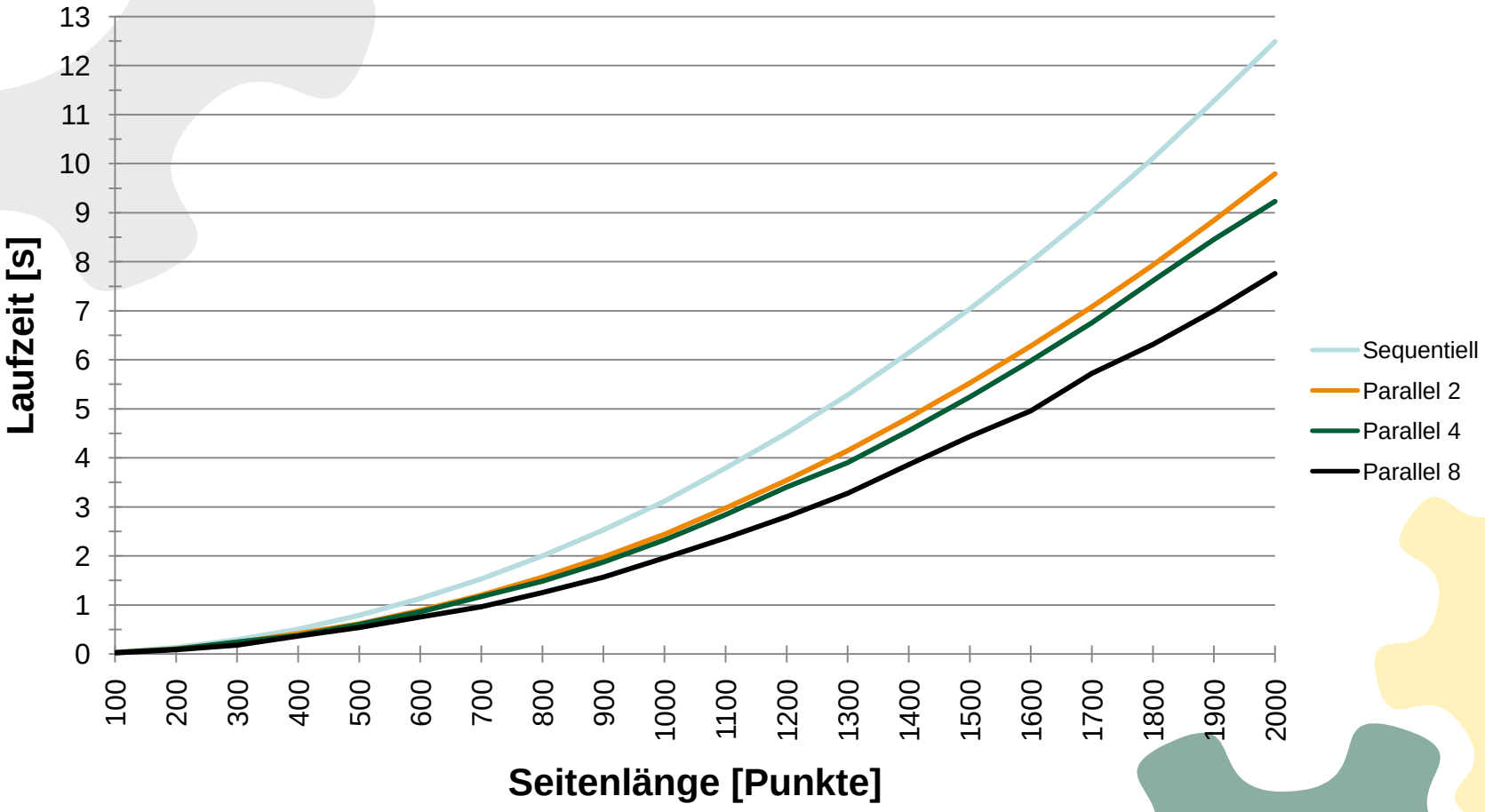
Die parallele Berechnung der Mandelbrot-Menge soll ausgeführt werden, wenn das Programm mit dem Parameter --parallel gestartet wird und die Anzahl der Threads (--threads) größer eins ist.



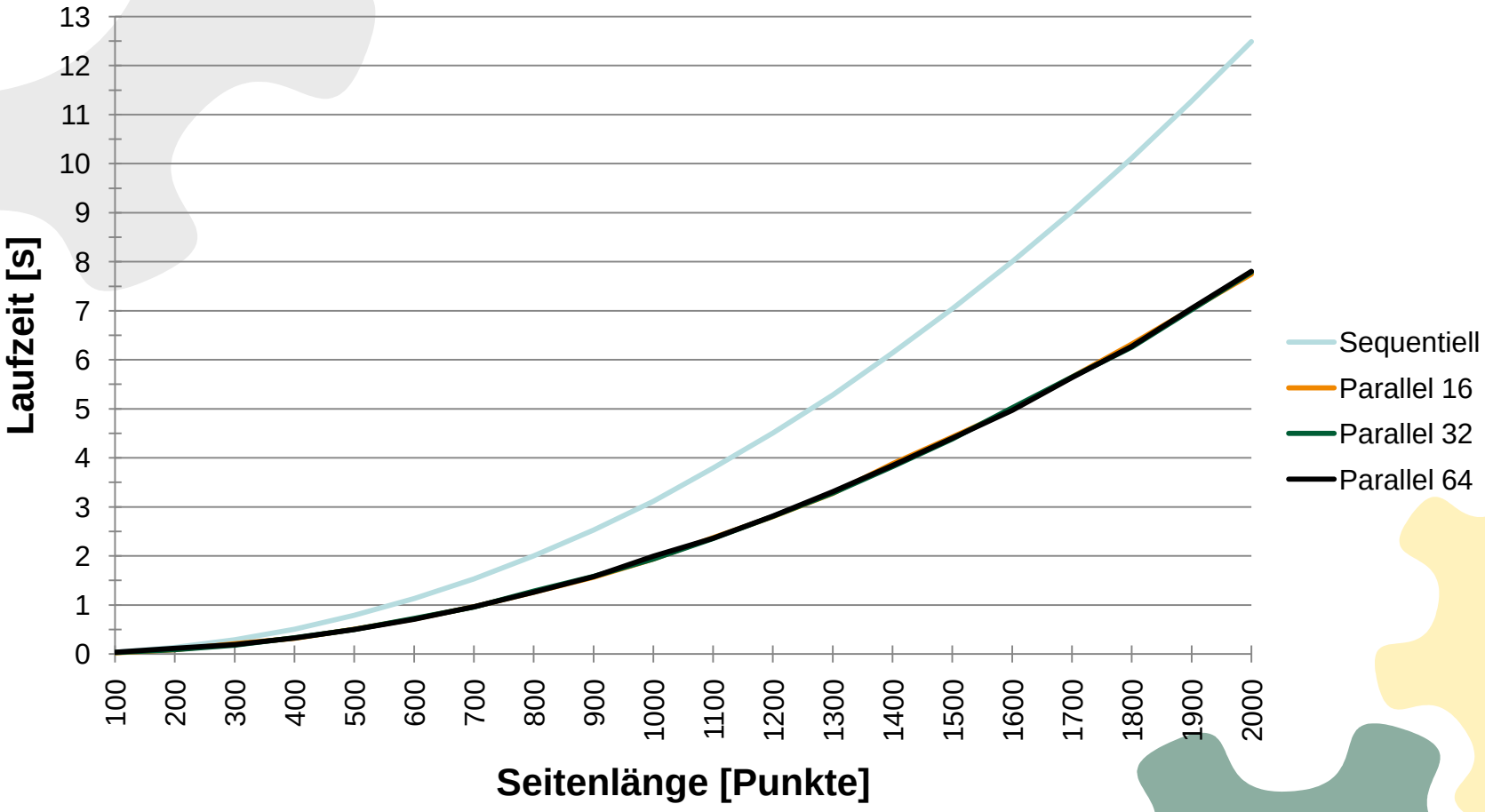
Aufgabe 3c)

Prozessor	Intel P4 531	Intel Core 2 Duo E6420	Intel Core 2 Quad Q6600	Sun UltraSPARC T2
Kerne	1 (× 2 Fäden)	2	4	8 (× 8 Fäden)
Takt [GHz]	3,0	2,133	2,4	0,9 - 1,4
L1 Cache (D + B) [KB]	16	(32 + 32) × 2	(32 + 32) × 4	(8 + 16) × 8
L2 Cache [MB]	1	4	4 × 2	4
Hauptspeicher [GB]	2	2	4	16

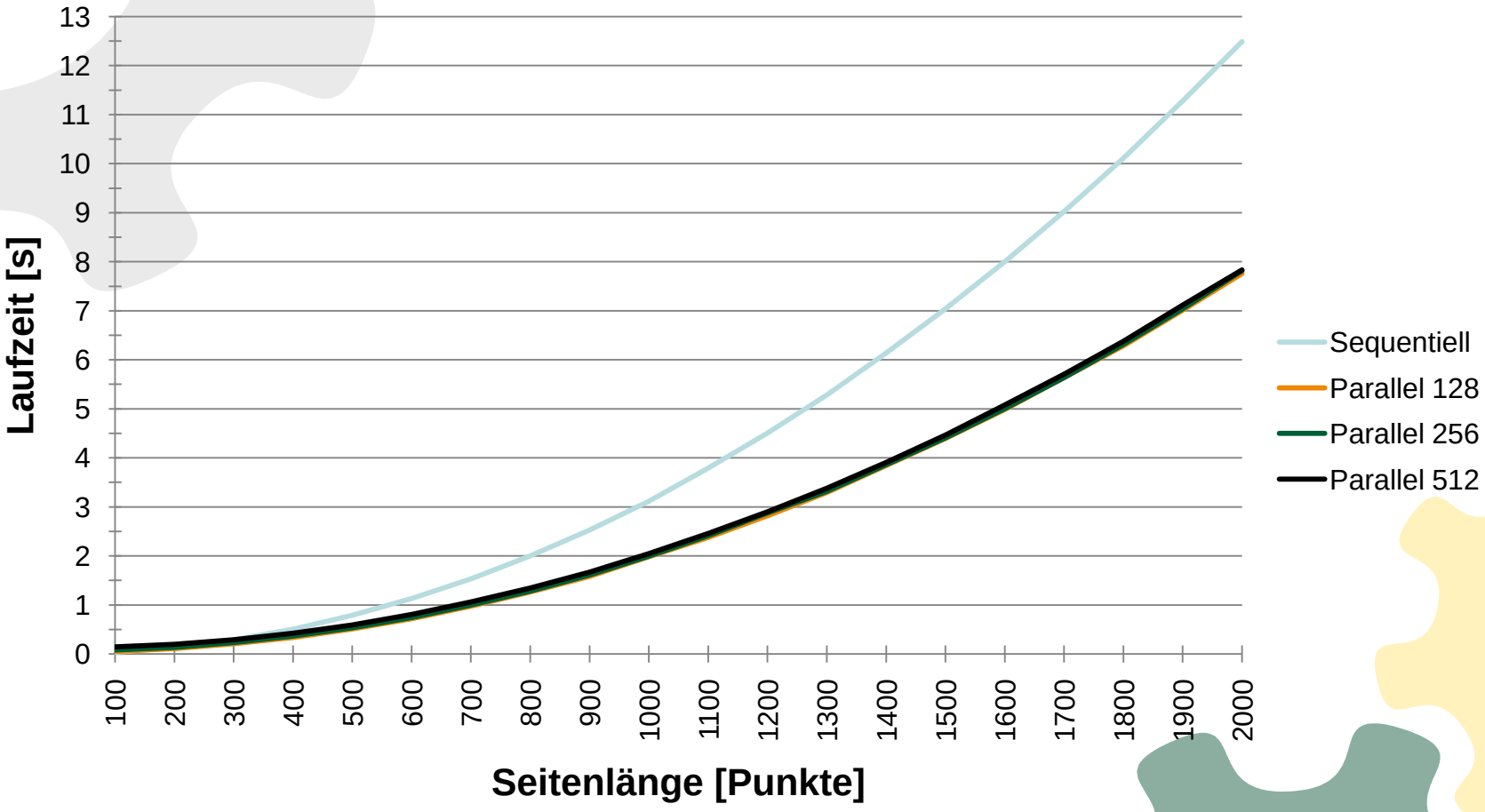
Laufzeit auf Intel Pentium 4 531



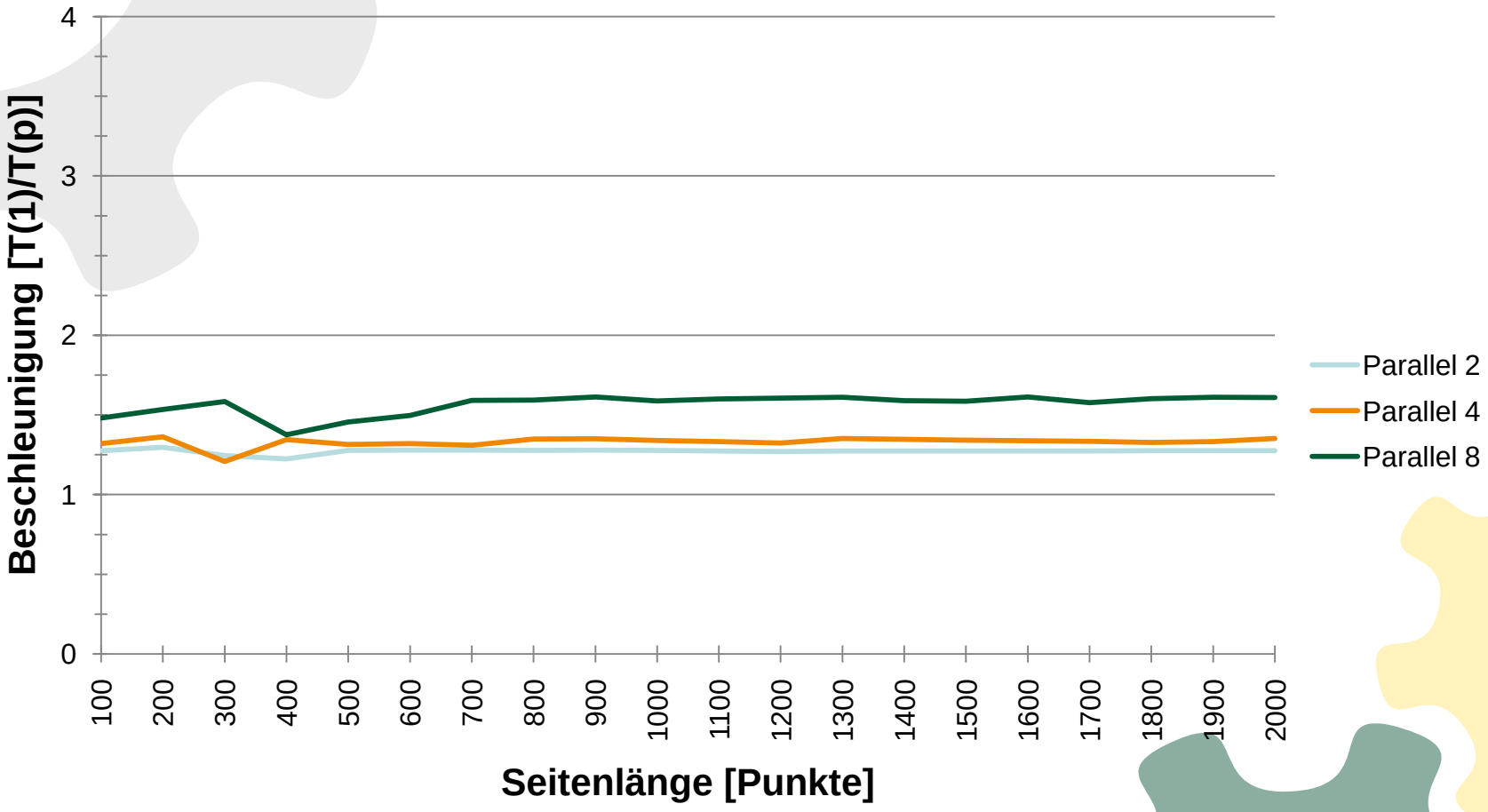
Laufzeit auf Intel Pentium 4 531



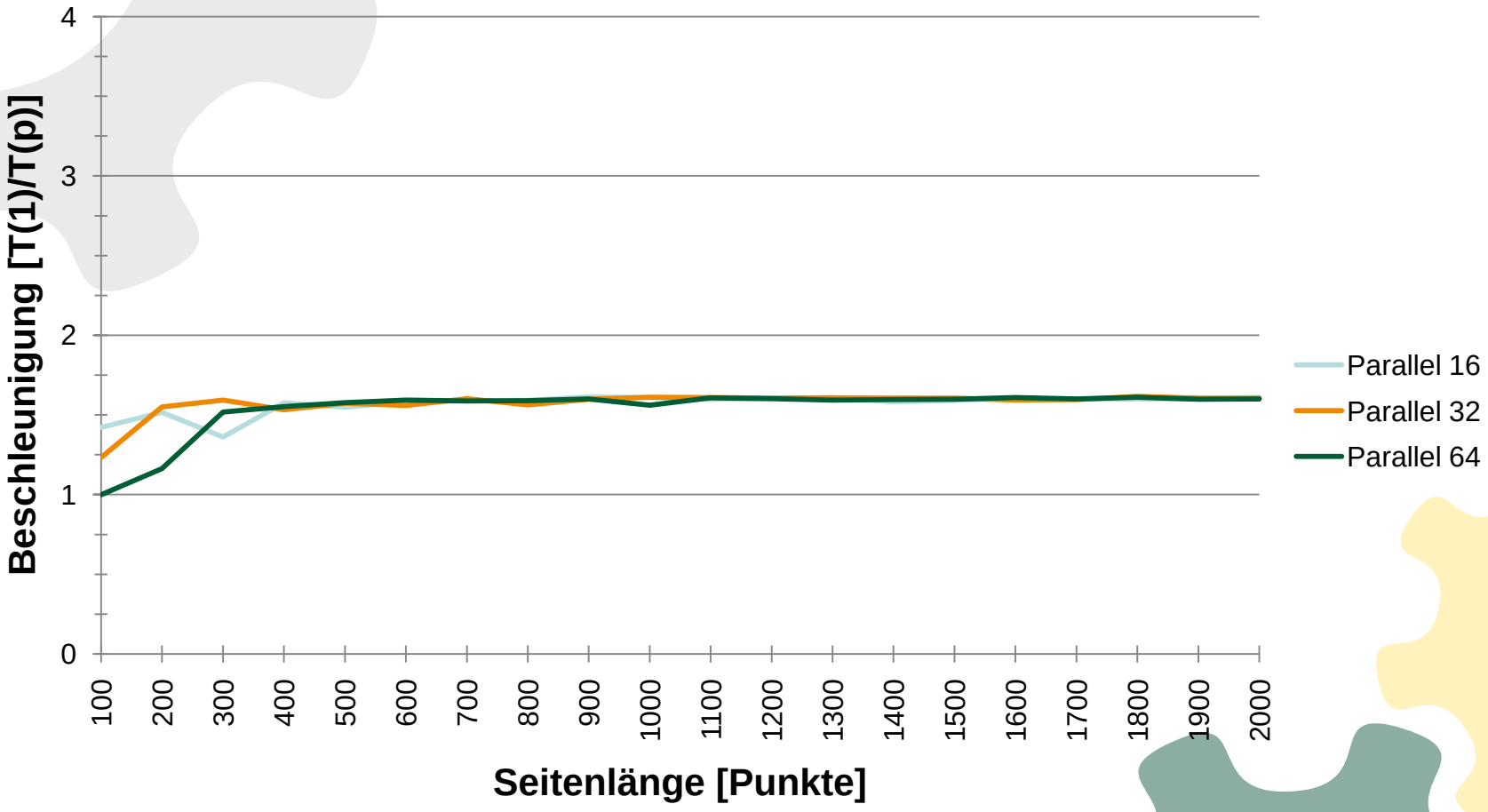
Laufzeit auf Intel Pentium 4 531



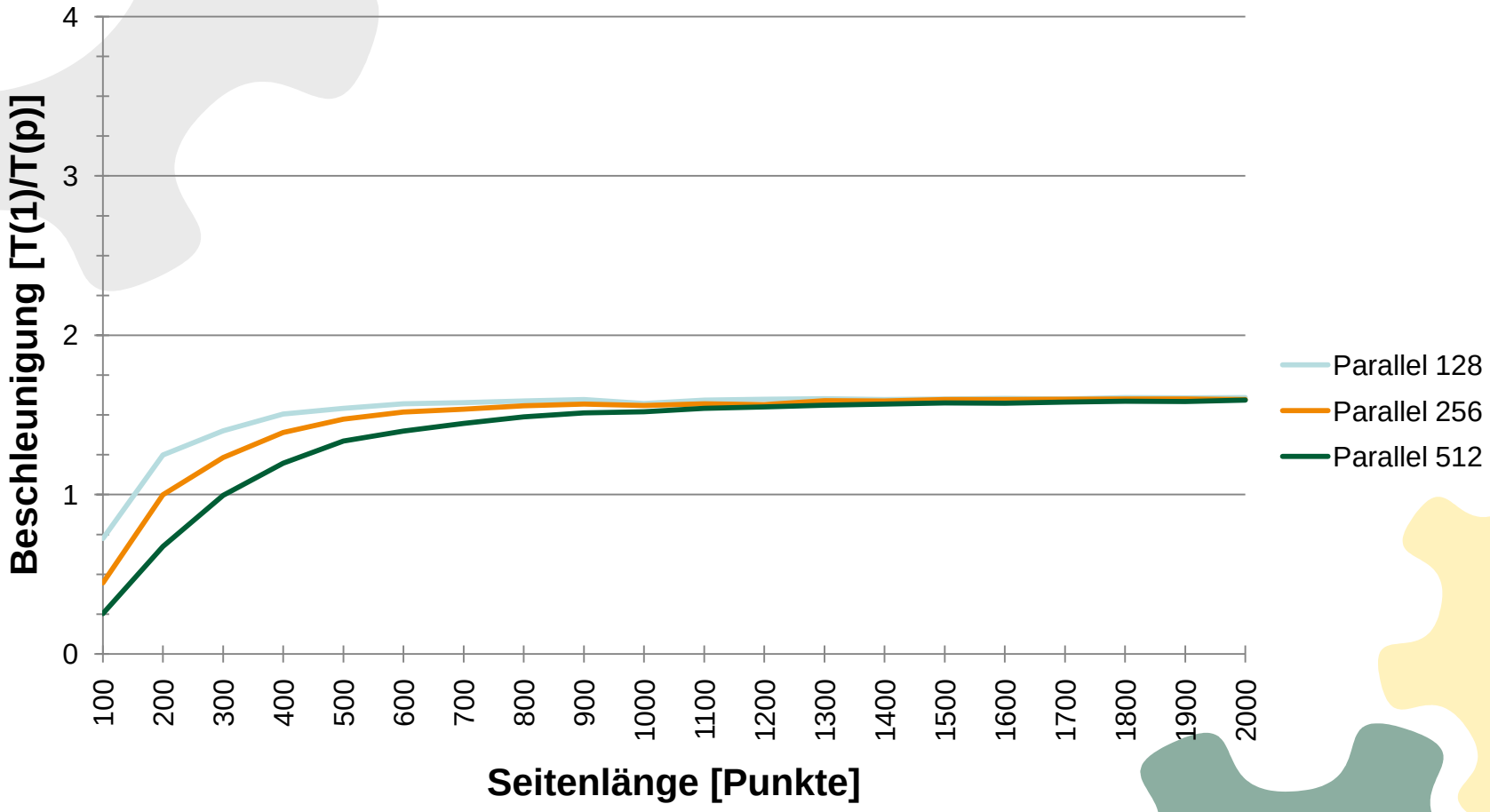
Beschleunigung auf Intel Pentium 4 531



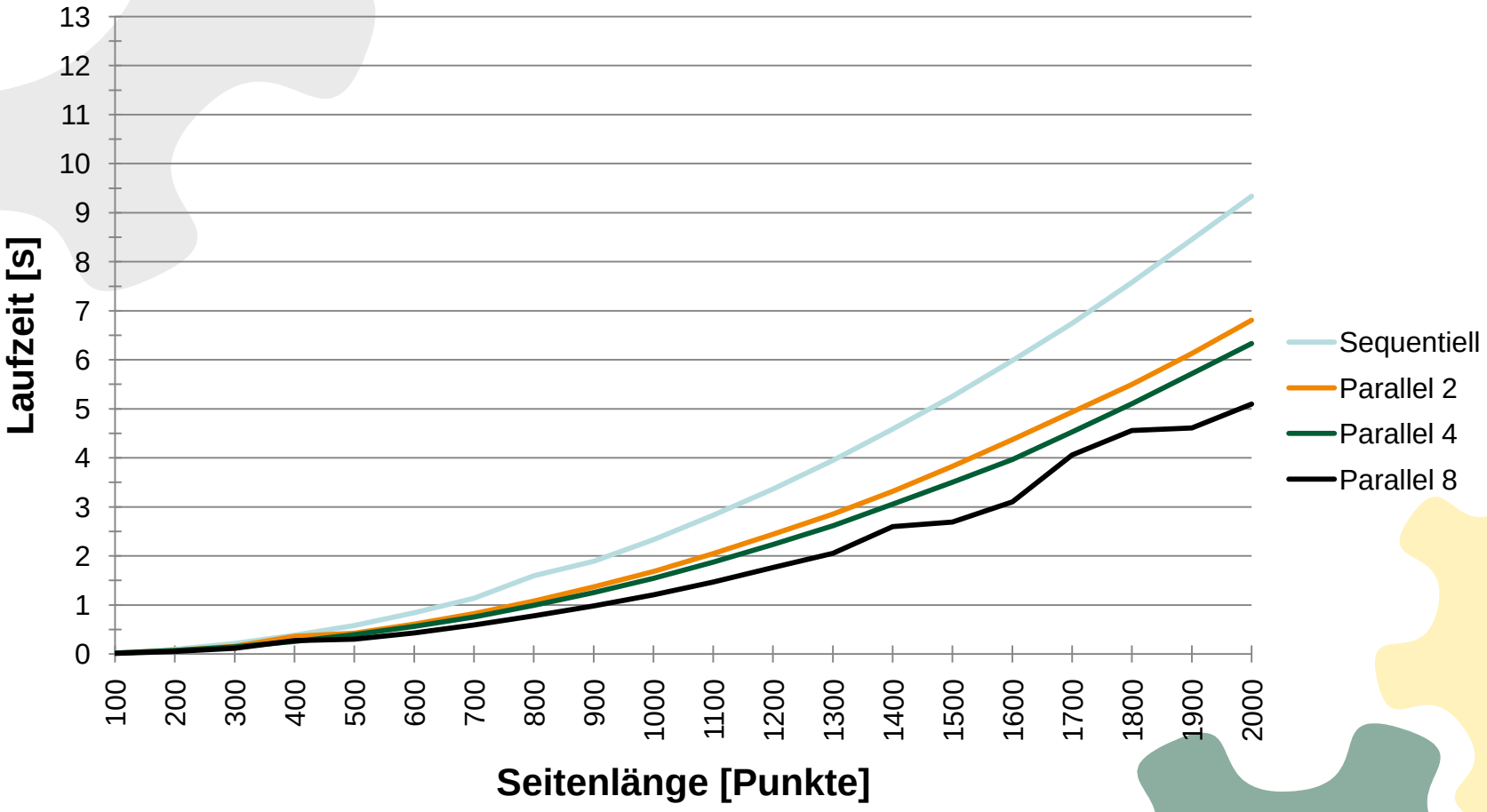
Beschleunigung auf Intel Pentium 4 531



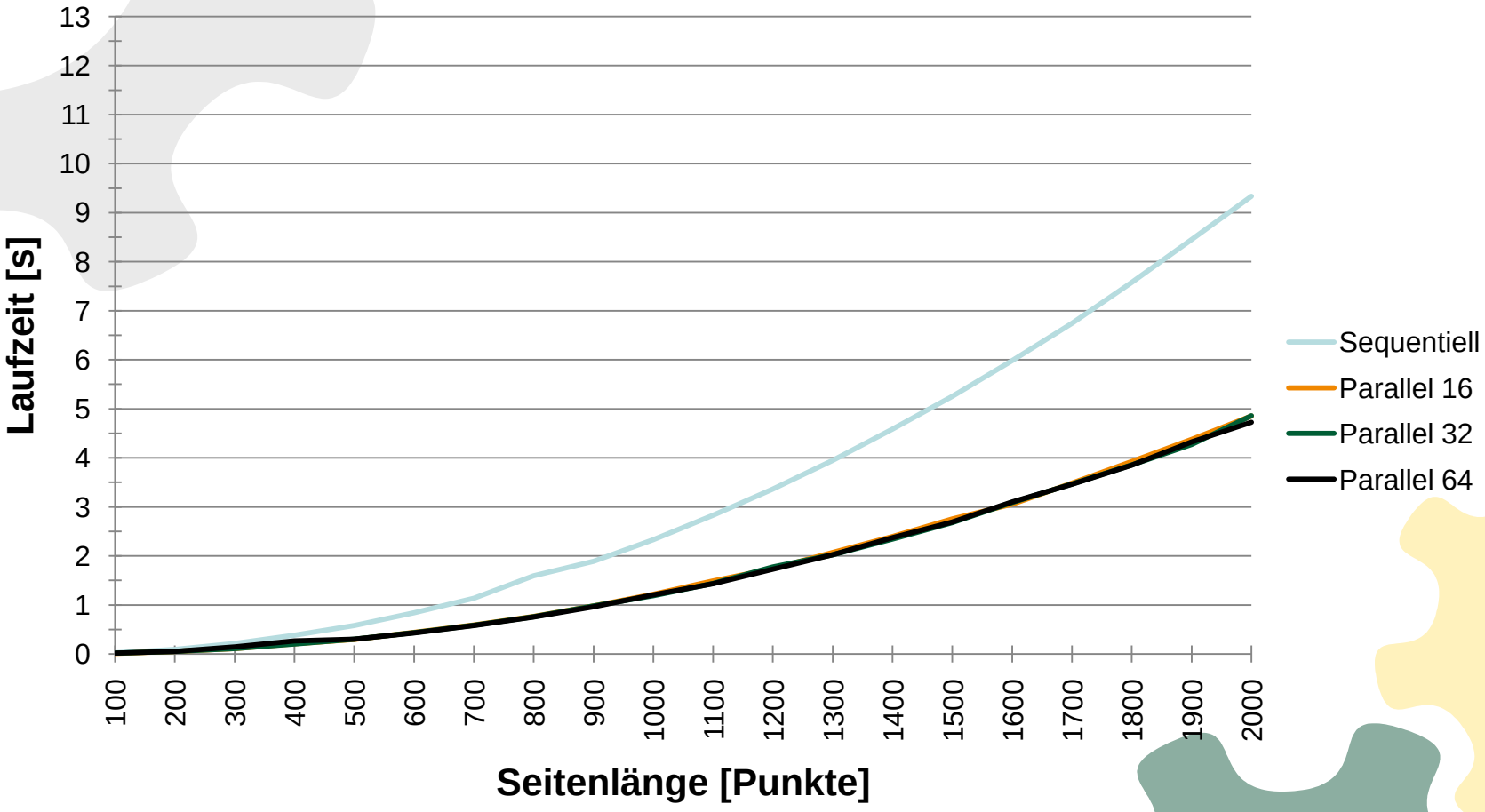
Beschleunigung auf Intel Pentium 4 531



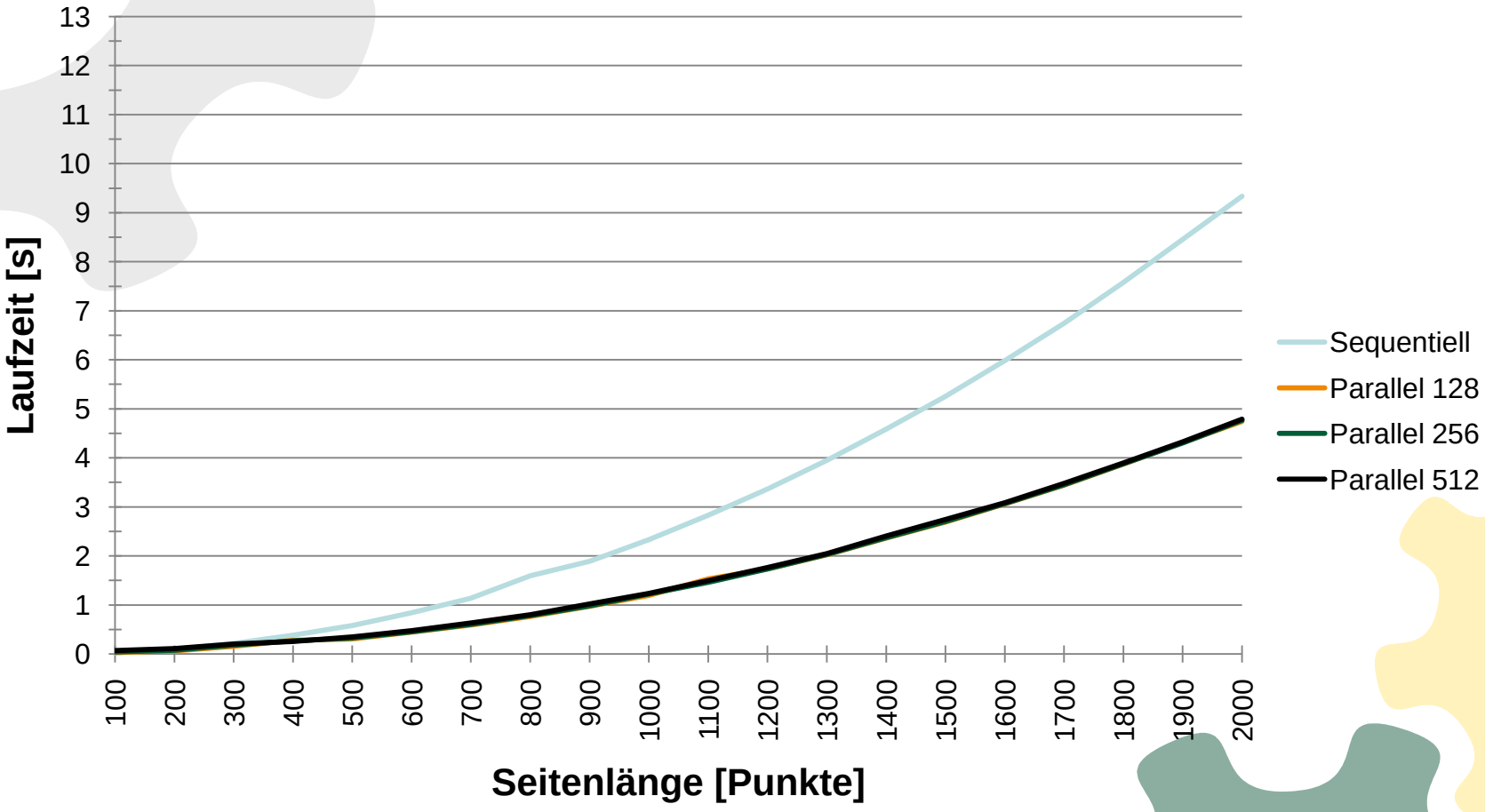
Laufzeit auf Intel Core 2 Duo E6420



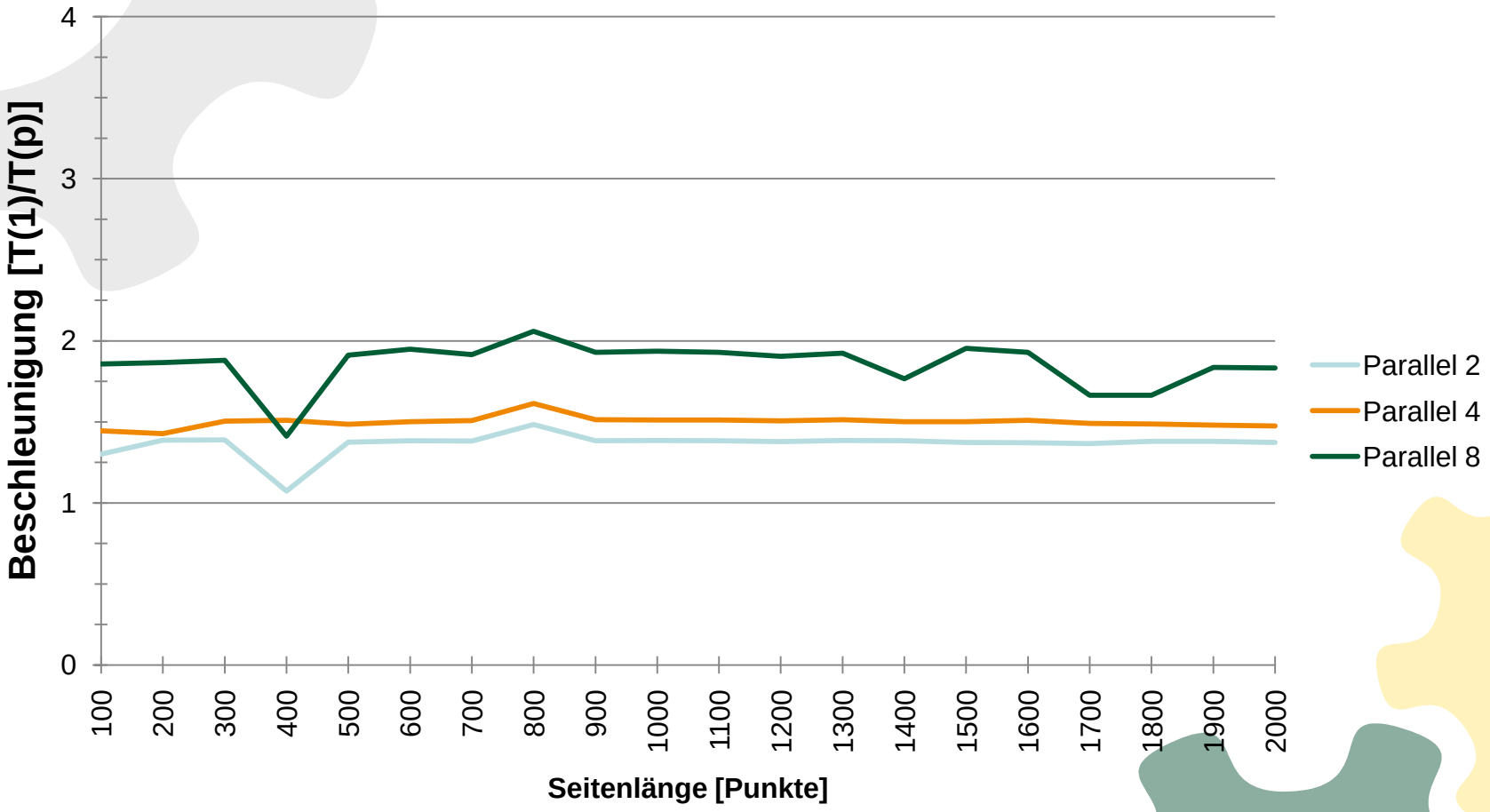
Laufzeit auf Intel Core 2 Duo E6420



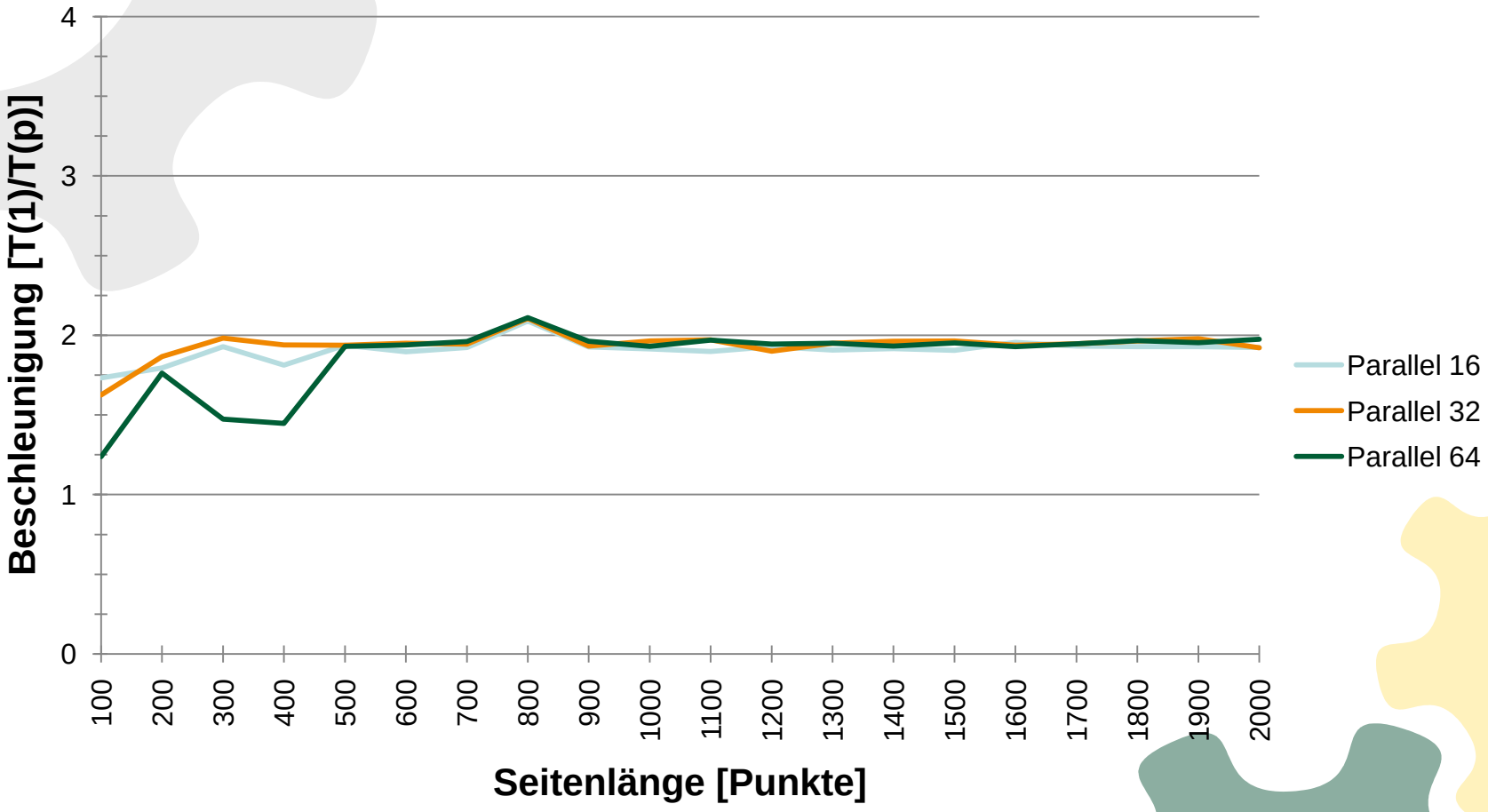
Laufzeit auf Intel Core 2 Duo E6420



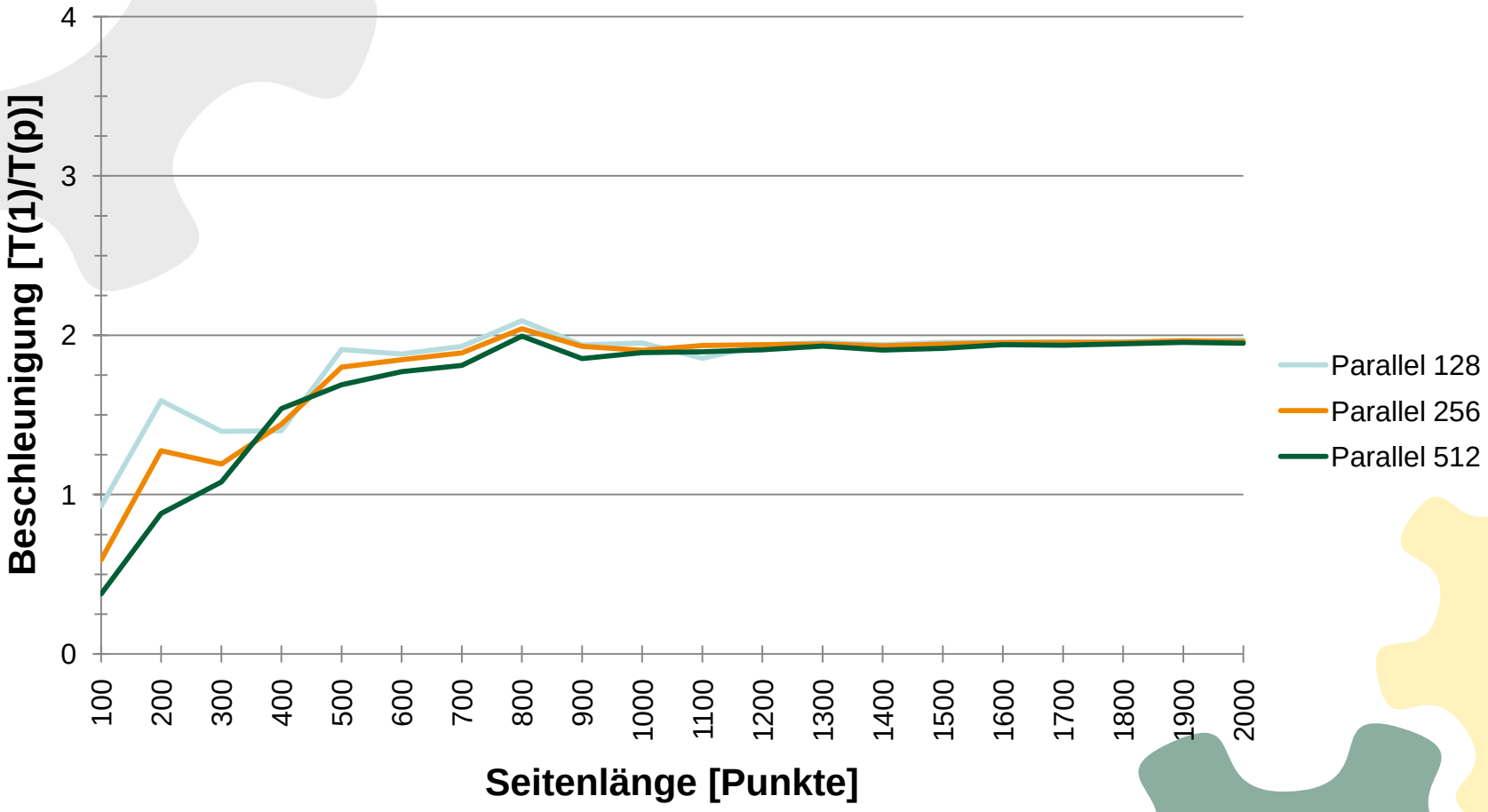
Beschleunigung auf Intel Core 2 Duo E6420



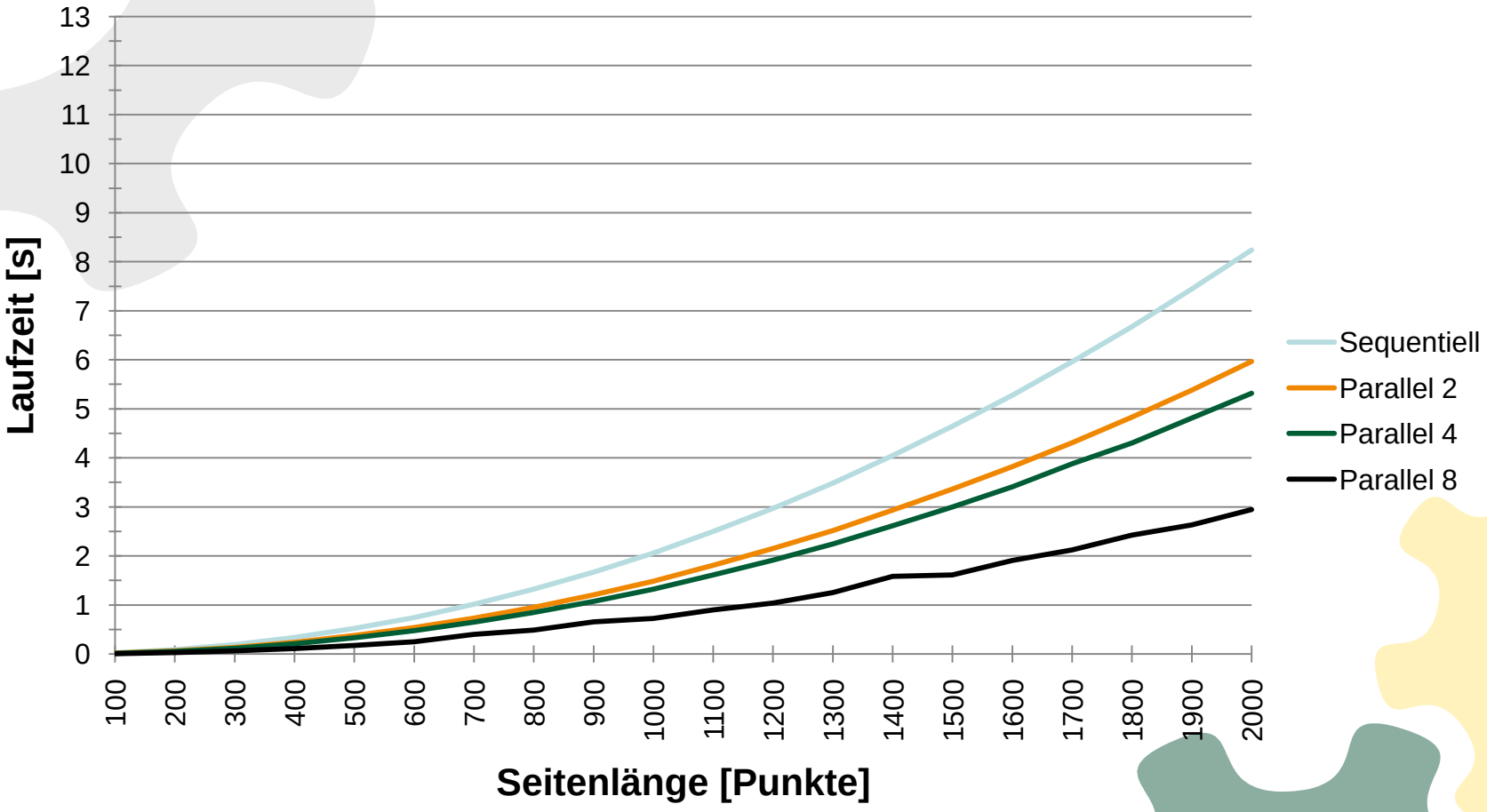
Beschleunigung auf Intel Core 2 Duo E6420



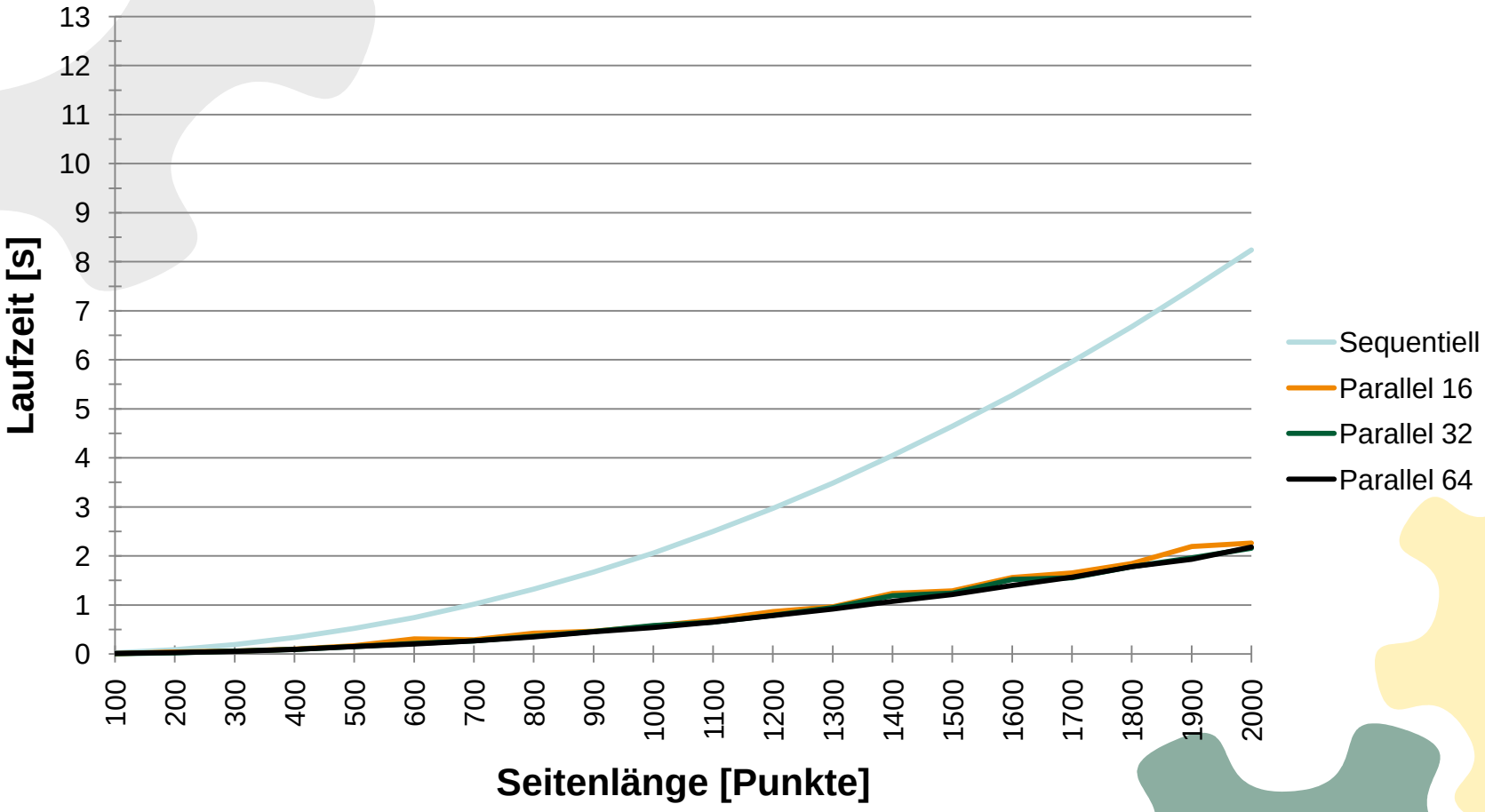
Beschleunigung auf Intel Core 2 Duo E6420



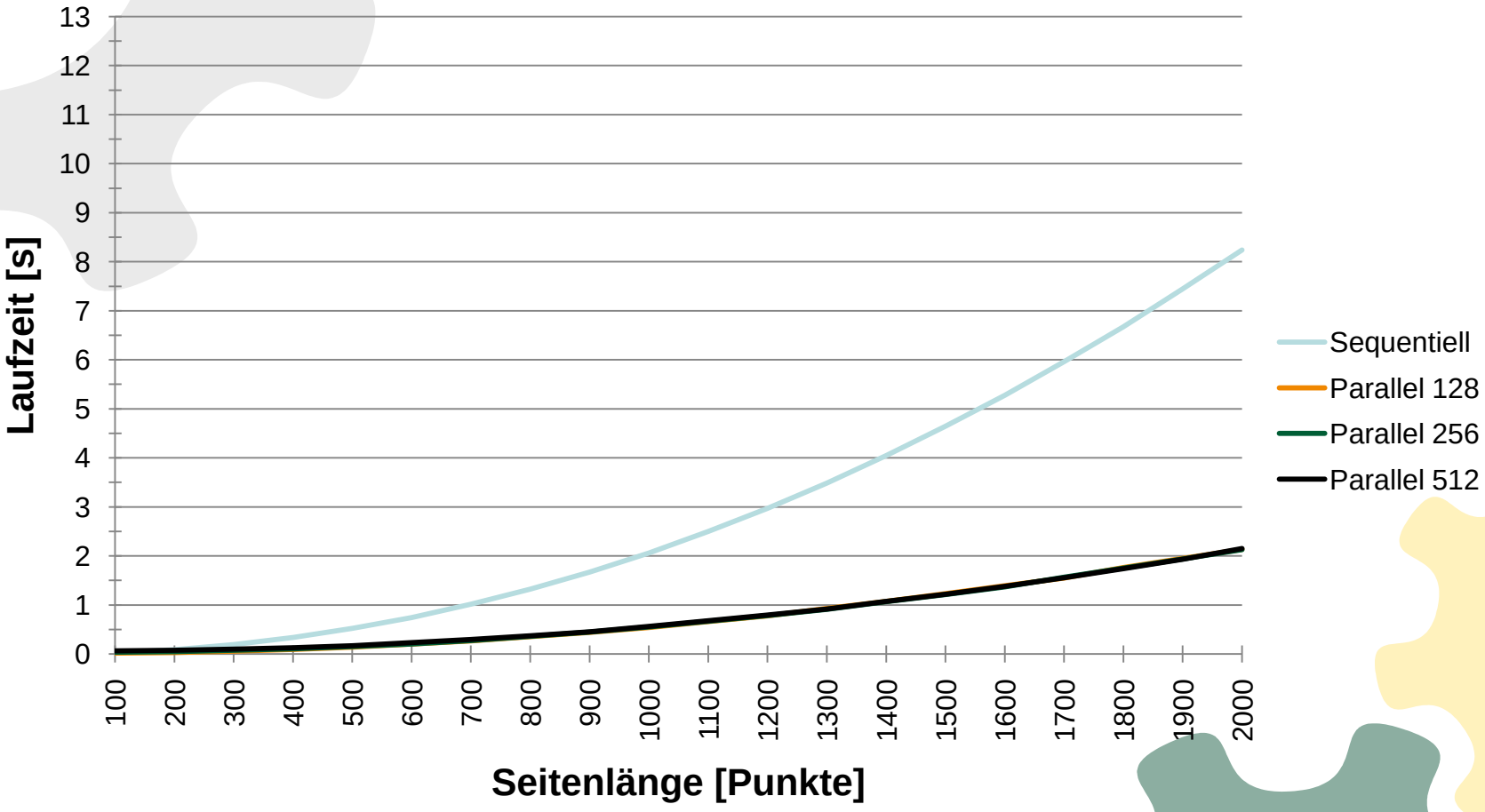
Laufzeit auf Intel Core 2 Quad Q6600



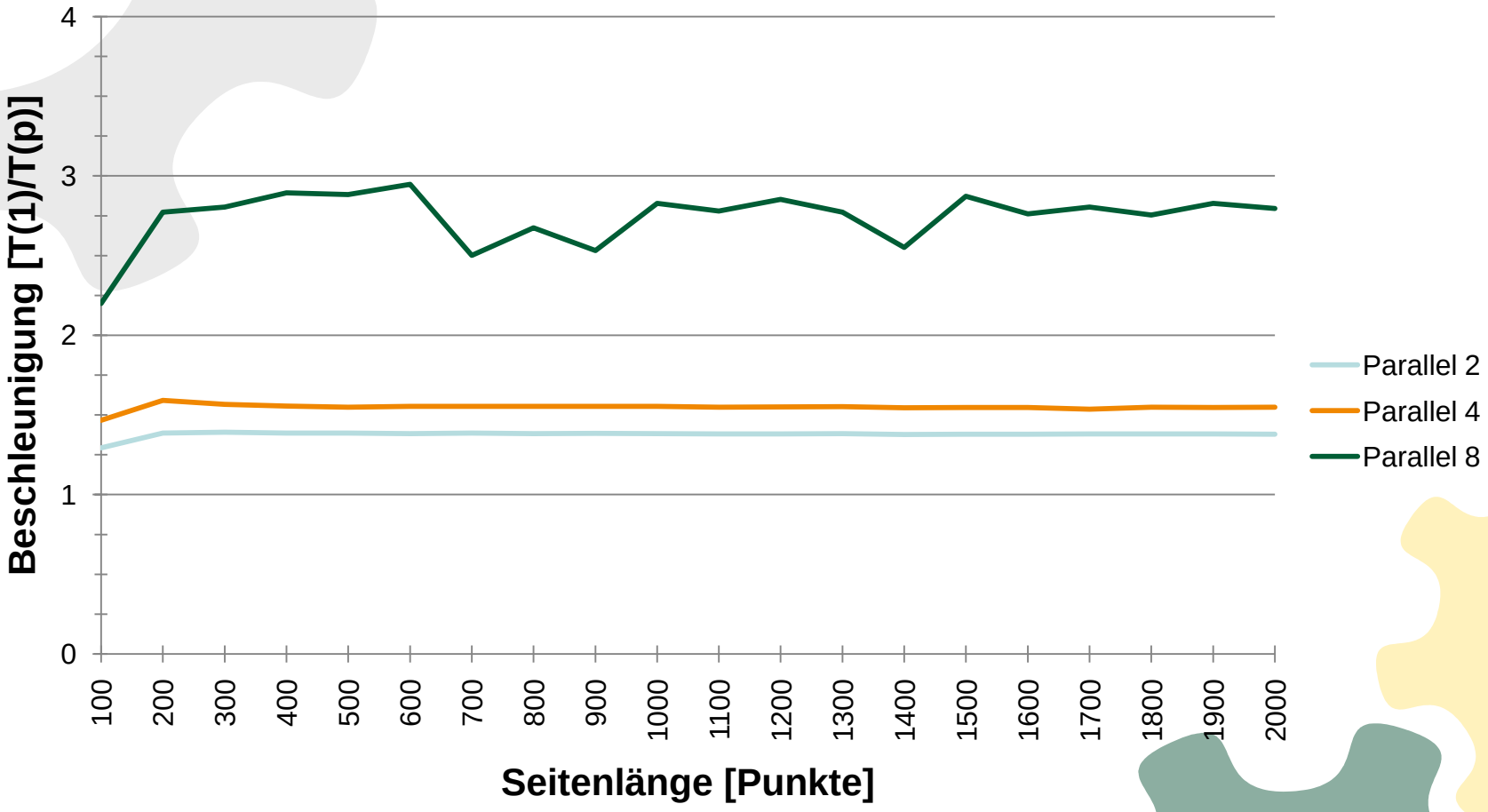
Laufzeit auf Intel Core 2 Quad Q6600



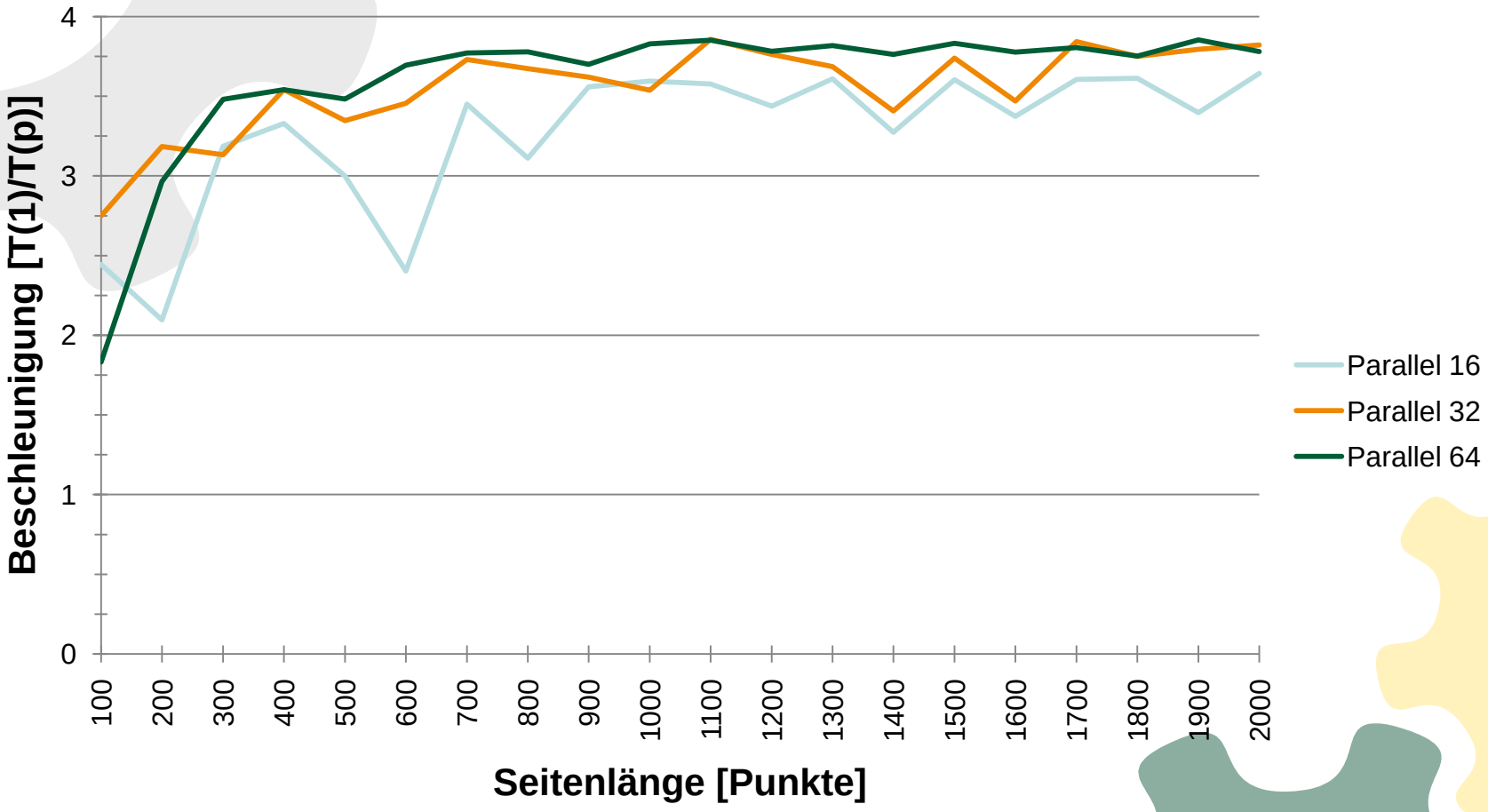
Laufzeit auf Intel Core 2 Quad Q6600



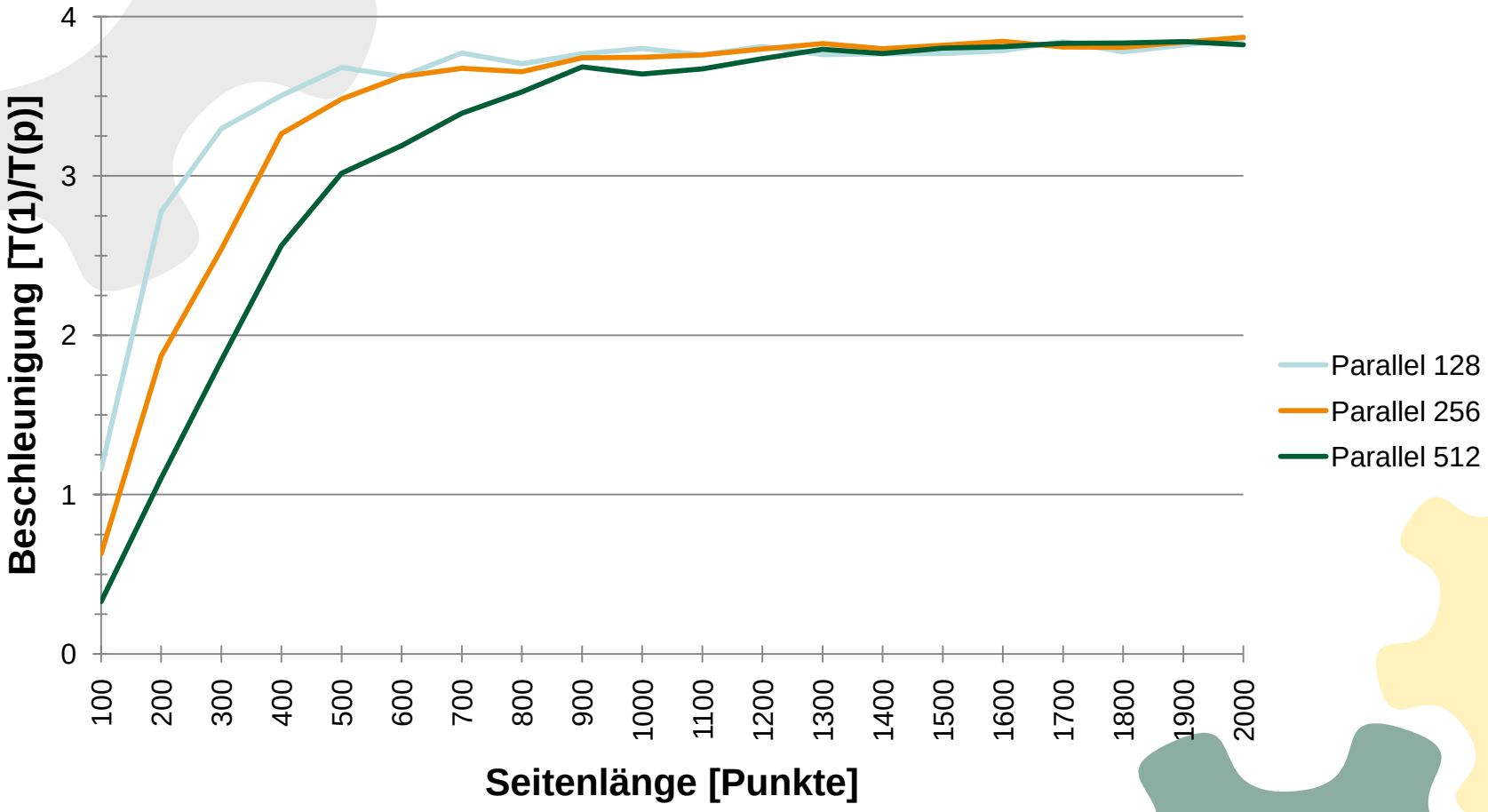
Beschleunigung auf Intel Core 2 Quad Q6600



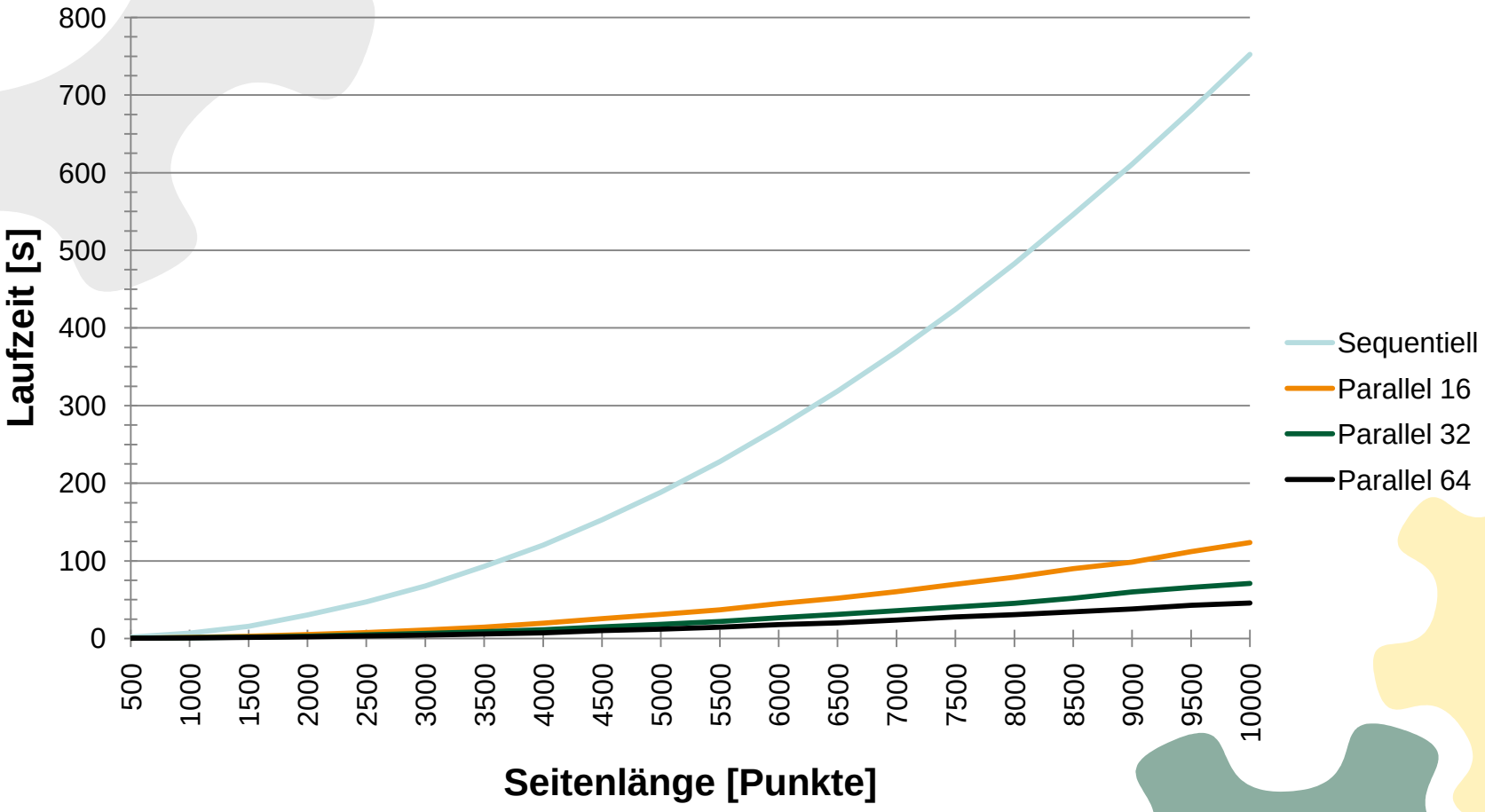
Beschleunigung auf Intel Core 2 Quad Q6600



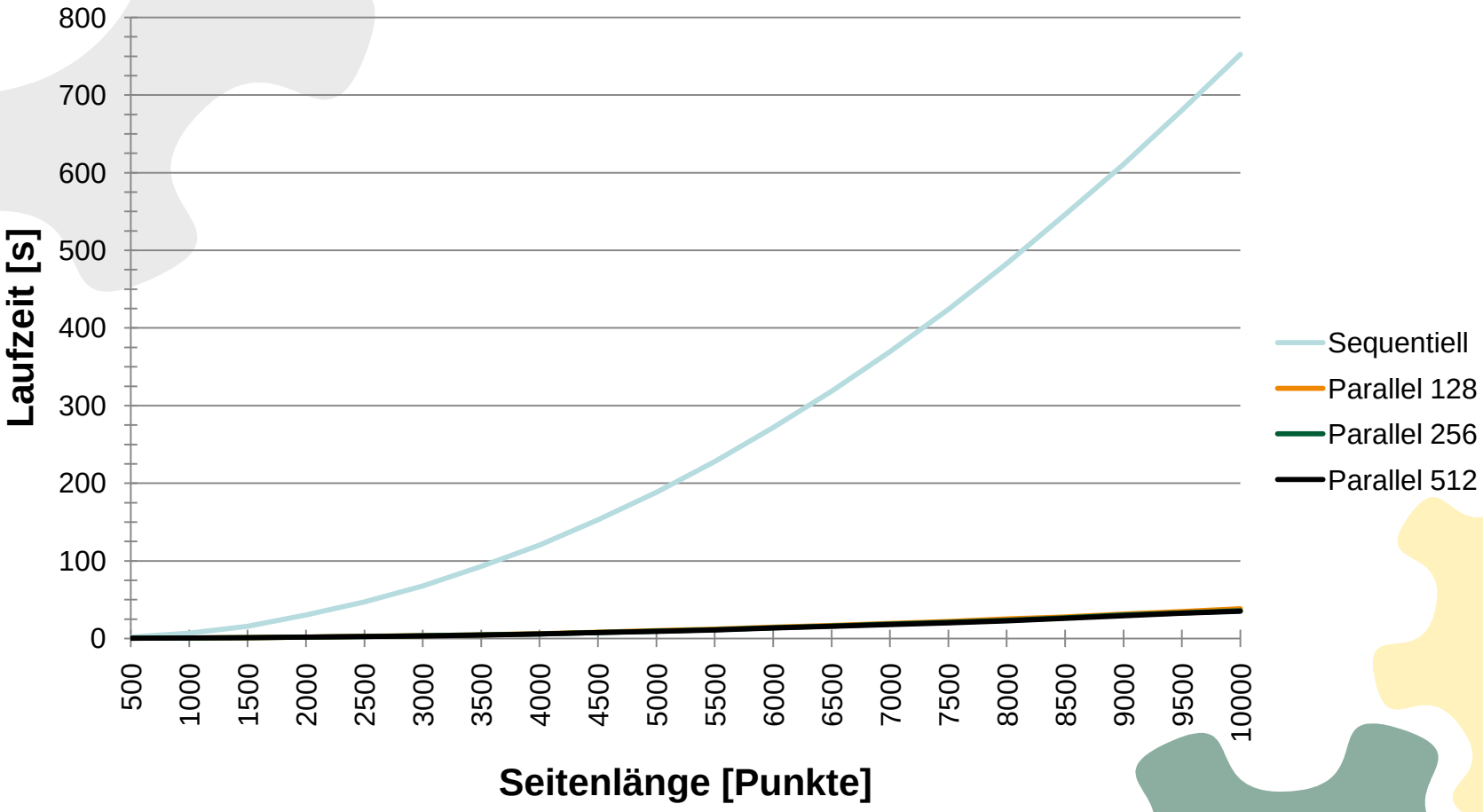
Beschleunigung auf Intel Core 2 Quad Q6600



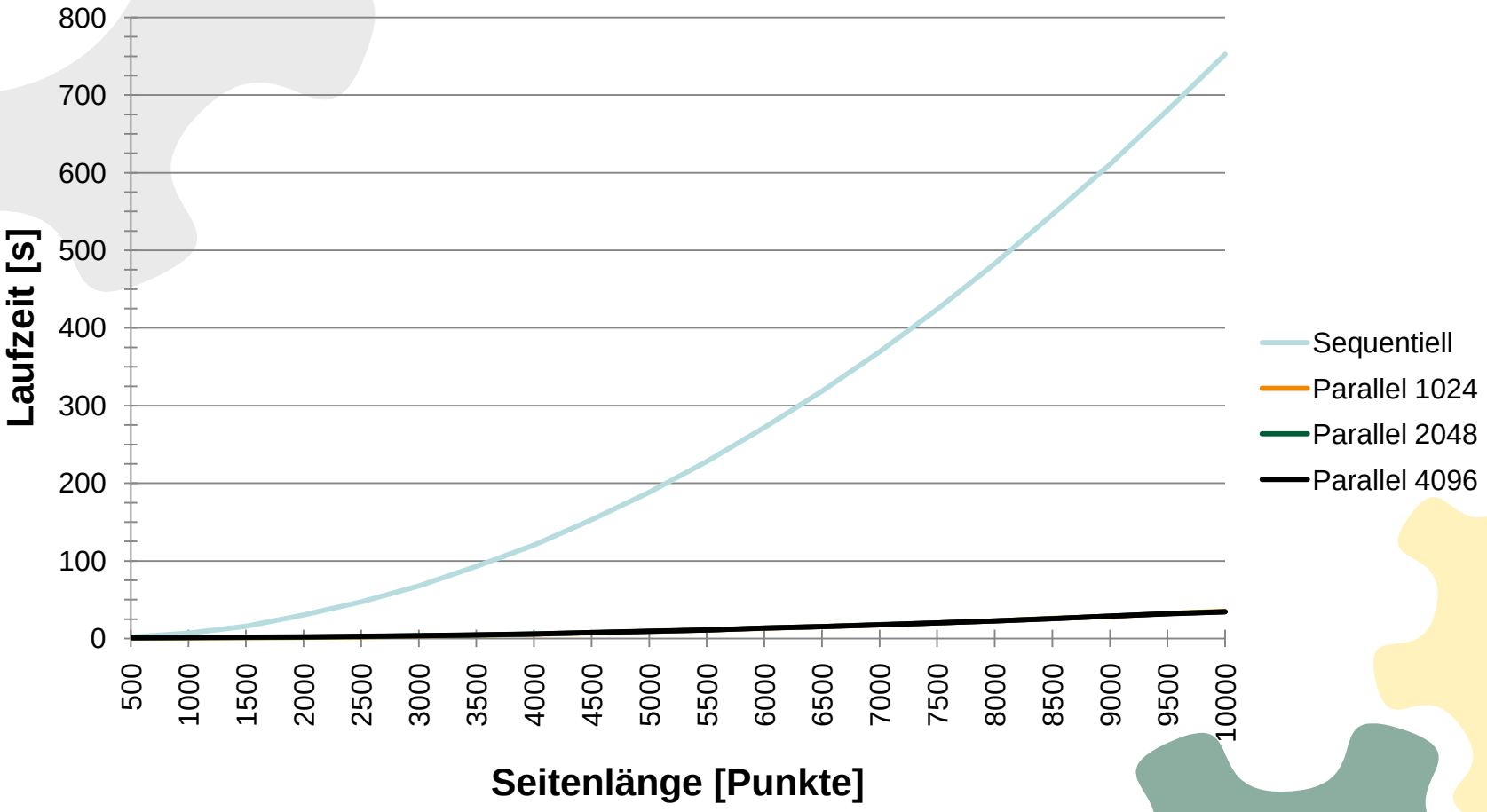
Laufzeit auf Sun UltraSPARC T2



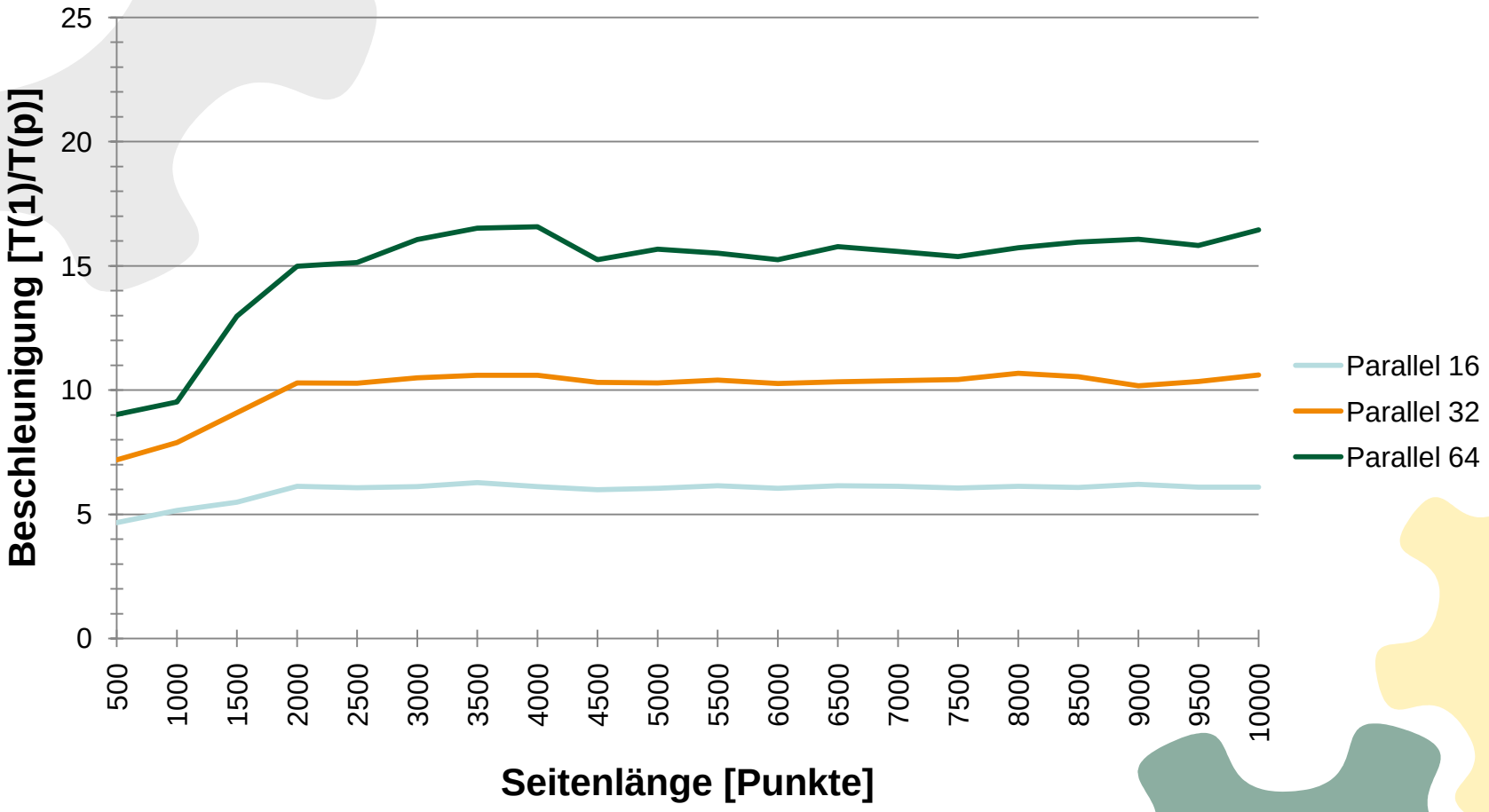
Laufzeit auf Sun UltraSPARC T2



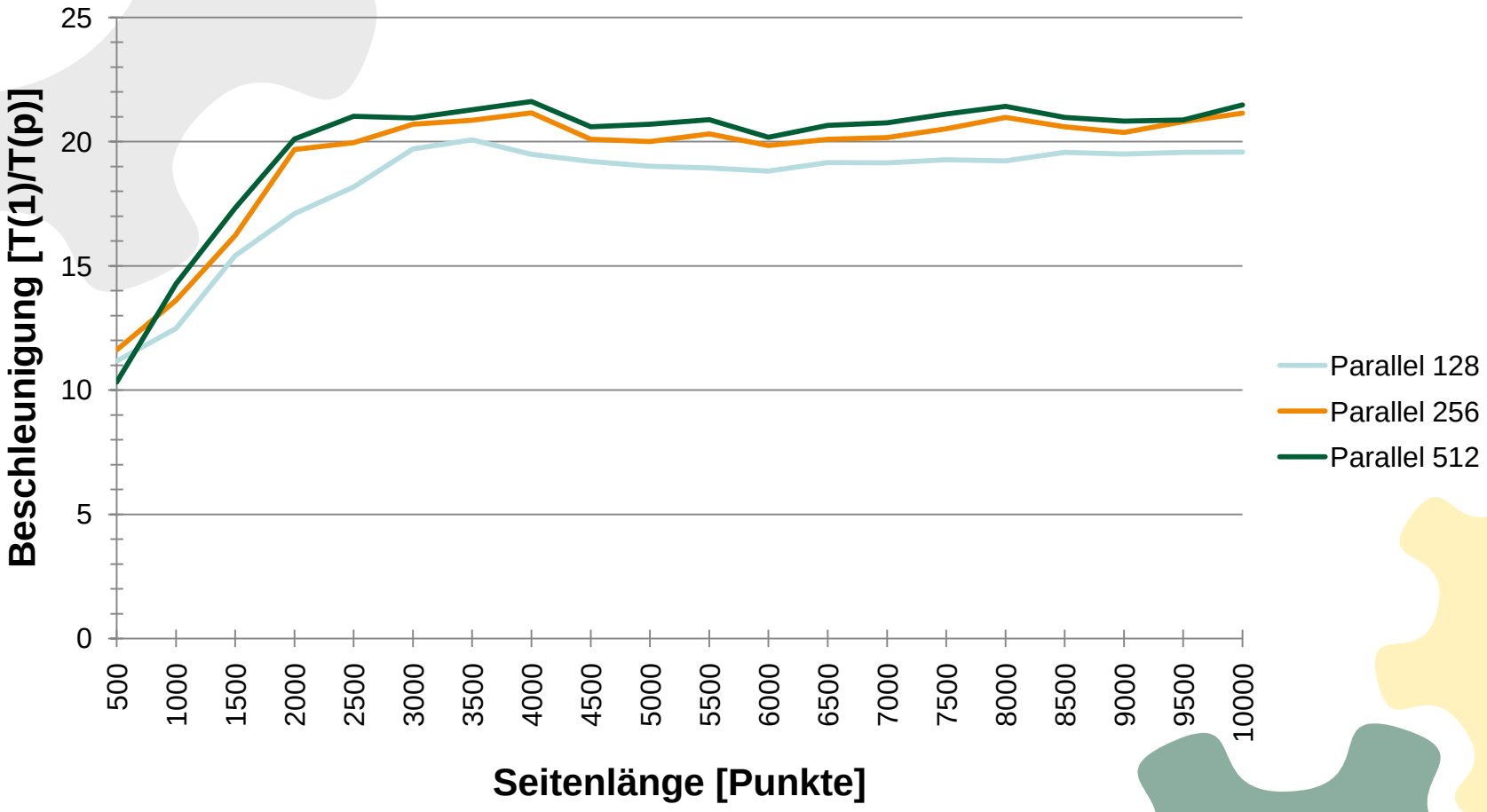
Laufzeit auf Sun UltraSPARC T2



Beschleunigung auf Sun UltraSPARC T2



Beschleunigung auf Sun UltraSPARC T2



Beschleunigung auf Sun UltraSPARC T2