TECHNISCHE UNIVERSITÄT DRESDEN

Institut für Technische Informatik, Professur für Rechnerarchitektur – TU Dresden

# Parallelisierung des Wellenfrontrekonstruktionsalgorithmus auf Multicore-Prozessoren

Jonas Schenke

16. April 2017

Betreuender HSL: Prof. Dr. Wolfgang E. Nagel
Betreuer: Dr. Elena-Ruxandra Cojocaru, Dr. Michael Bussmann, Matthias Werner
E-Mail: jonas.schenke@tu-dresden.de

ZIH
Center for Information Services & High Performance Computing

# Aufgabenstellung

- Evaluierung und Performance-Analyse des derzeit fast durchgängig seriellen Wellenfrontrekonstruktionsalgorithmus
- Parallelisierung der kritischen Pfade für Mehrkernarchitekturen
- Performance-Messungen der parallelen Implementation
- Auswertung sowie Validierung der Ergebnisse

# Gliederung

# Gliederung

# Hinweise

- Teil des **Eu**ropean **C**luster of **A**dvanced **L**aser **L**ight Sources (EUCALL)-Projektes →Überschneidung von **U**ltrafast **D**ata **Ac**quisition (UFDAC; WP5) und **Pu**lse **C**haracterisation and **C**ontrol (PUCCA; WP7)
- Zusammenarbeit des **H**elmholtz-**Z**entrum **D**resden-**R**ossendorf e.V. (HZDR) und **E**uropean **S**ynchrotron **R**adiation **F**acillity (ESRF)
- **Algorithmus:** Sébastien Bérujon
- **Code:** Elena-Ruxandra Cojocaru
- **Daten:** Beamline BM05, ESRF, Grenoble, Frankreich

# Hinweise

- Teil des **Eu**ropean **C**luster of **A**dvanced **L**aser **L**ight Sources (EUCALL)-Projektes →Überschneidung von **U**ltra**f**ast **D**ata **Ac**quisition (UFDAC; WP5) und **Pu**lse **C**haracterisation and **C**ontrol (PUCCA; WP7)
- Zusammenarbeit des **H**elmholtz-**Z**entrum **D**resden-**R**ossendorf e.V. (HZDR) und **E**uropean **S**ynchrotron **R**adiation **F**acillity (ESRF)
- **Algorithmus:** Sébastien Bérujon
- **Code:** Elena-Ruxandra Cojocaru
- **Daten:** Beamline BM05, ESRF, Grenoble, Frankreich

# Hinweise

- Teil des **Eu**ropean **C**luster of **A**dvanced **L**aser **L**ight Sources (EUCALL)-Projektes →Überschneidung von **U**ltra**f**ast **D**ata **Ac**quisition (UFDAC; WP5) und **Pu**lse **C**haracterisation and **C**ontrol (PUCCA; WP7)
- Zusammenarbeit des **H**elmholtz-**Z**entrum **D**resden-**R**ossendorf e.V. (HZDR) und **E**uropean **S**ynchrotron **R**adiation **F**acillity (ESRF)
- **Algorithmus:** Sébastien Bérujon
- **Code:** Elena-Ruxandra Cojocaru
- **Daten:** Beamline BM05, ESRF, Grenoble, Frankreich

# Hinweise

- Teil des **Eu**ropean **C**luster of **A**dvanced **L**aser **L**ight Sources (EUCALL)-Projektes →Überschneidung von **U**ltra**f**ast **D**ata **Ac**quisition (UFDAC; WP5) und **Pu**lse **C**haracterisation and **C**ontrol (PUCCA; WP7)
- Zusammenarbeit des **H**elmholtz-**Z**entrum **D**resden-**R**ossendorf e.V. (HZDR) und **E**uropean **S**ynchrotron **R**adiation **F**acillity (ESRF)
- **Algorithmus:** Sébastien Bérujon
- **Code:** Elena-Ruxandra Cojocaru
- **Daten:** Beamline BM05, ESRF, Grenoble, Frankreich

# Hinweise

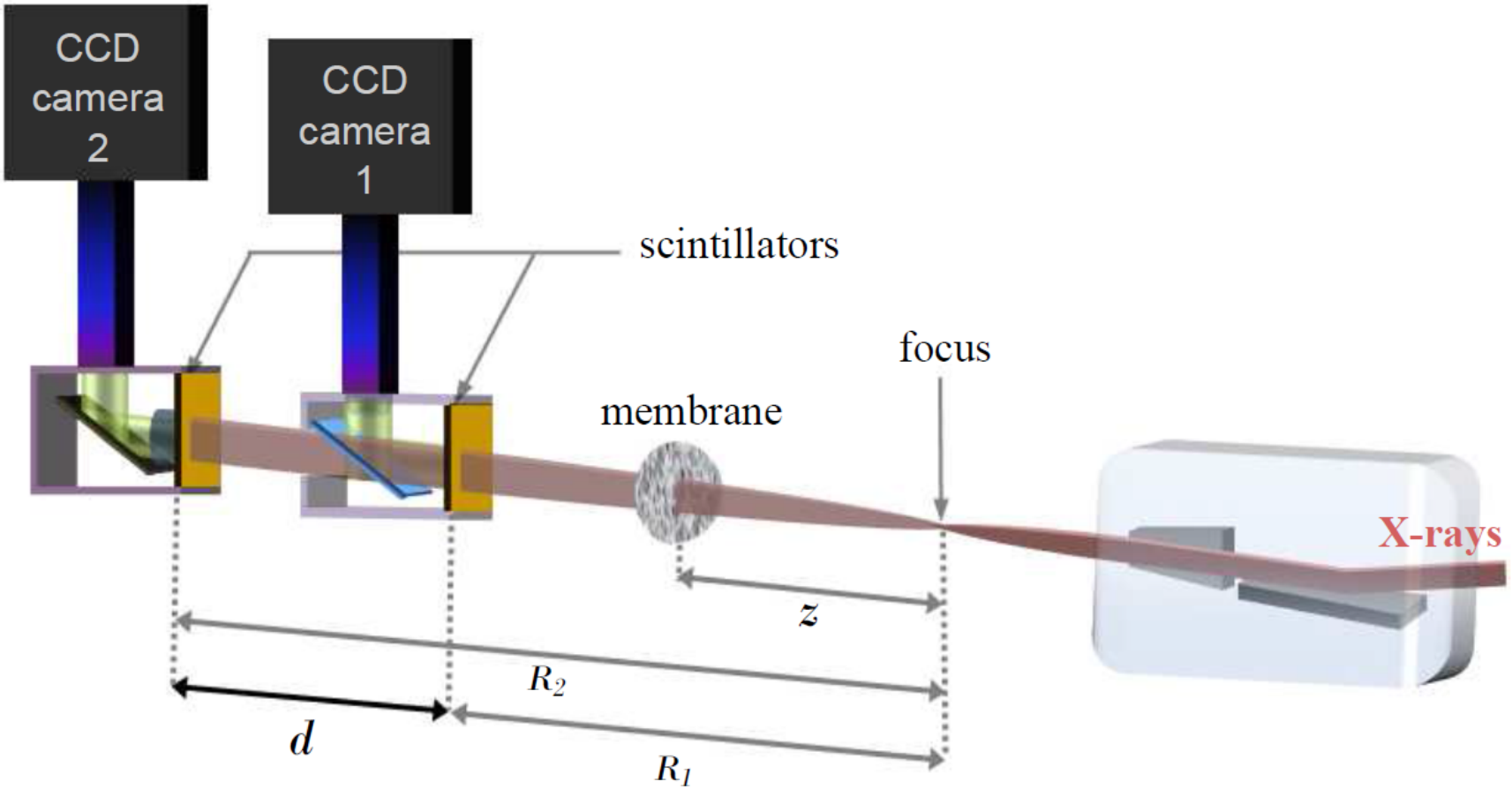
- Teil des **Eu**ropean **C**luster of **A**dvanced **L**aser **L**ight Sources (EUCALL)-Projektes →Überschneidung von **U**ltra**f**ast **D**ata **Ac**quisition (UFDAC; WP5) und **Pu**lse **C**haracterisation and **C**ontrol (PUCCA; WP7)
- Zusammenarbeit des **H**elmholtz-**Z**entrum **D**resden-**R**ossendorf e.V. (HZDR) und **E**uropean **S**ynchrotron **R**adiation **F**acillity (ESRF)
- **Algorithmus:** Sébastien Bérujon
- **Code:** Elena-Ruxandra Cojocaru
- **Daten:** Beamline BM05, ESRF, Grenoble, Frankreich

# Hinweise

- Teil des **Eu**ropean **C**luster of **A**dvanced **L**aser **L**ight Sources (EUCALL)-Projektes →Überschneidung von **U**ltra**f**ast **D**ata **Ac**quisition (UFDAC; WP5) und **Pu**lse **C**haracterisation and **C**ontrol (PUCCA; WP7)
- Zusammenarbeit des **H**elmholtz-**Z**entrum **D**resden-**R**ossendorf e.V. (HZDR) und **E**uropean **S**ynchrotron **R**adiation **F**acillity (ESRF)
- **Algorithmus:** Sébastien Bérujon
- **Code:** Elena-Ruxandra Cojocaru
- **Daten:** Beamline BM05, ESRF, Grenoble, Frankreich

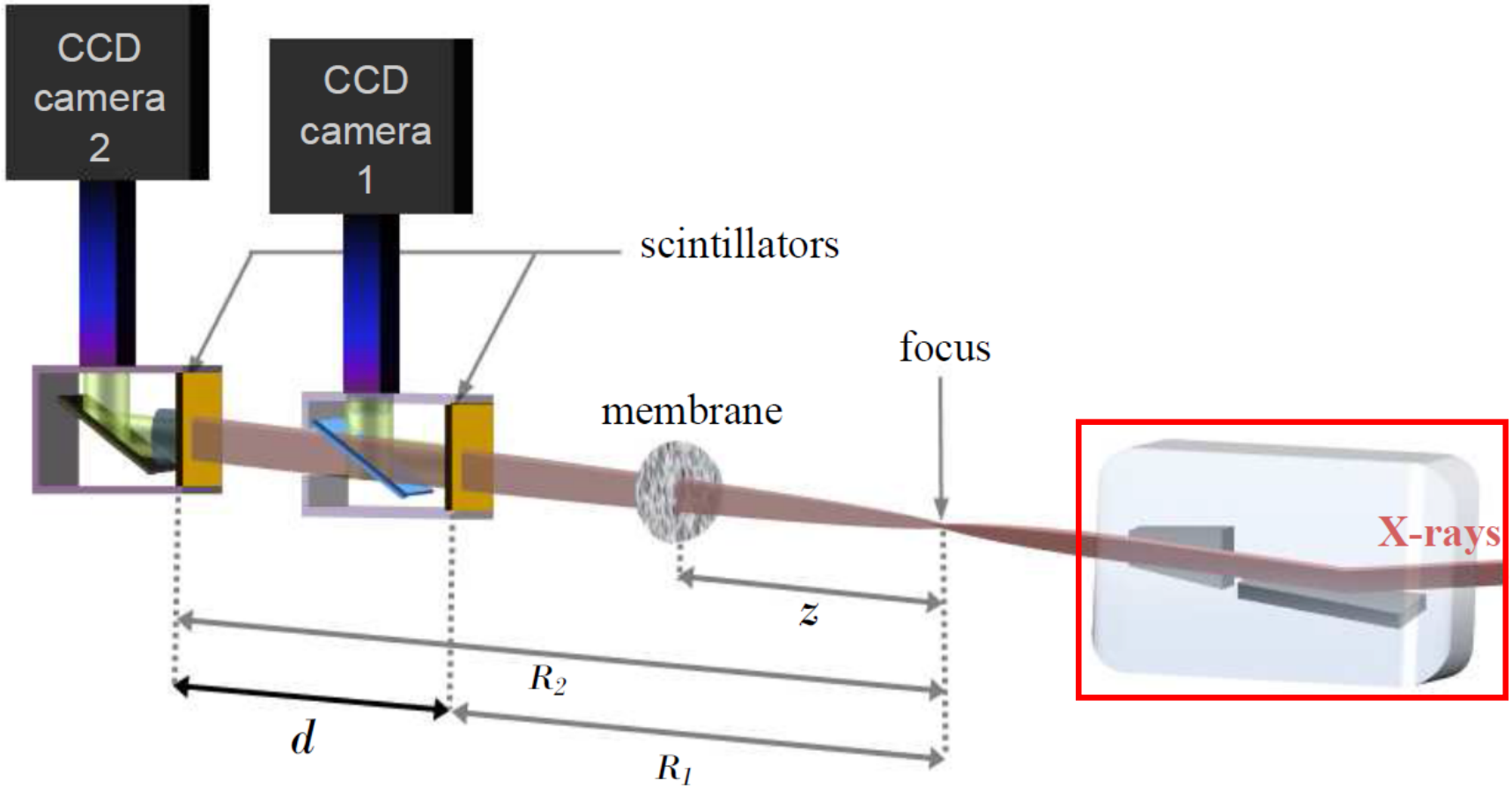
# Gliederung

# Versuchsaufbau



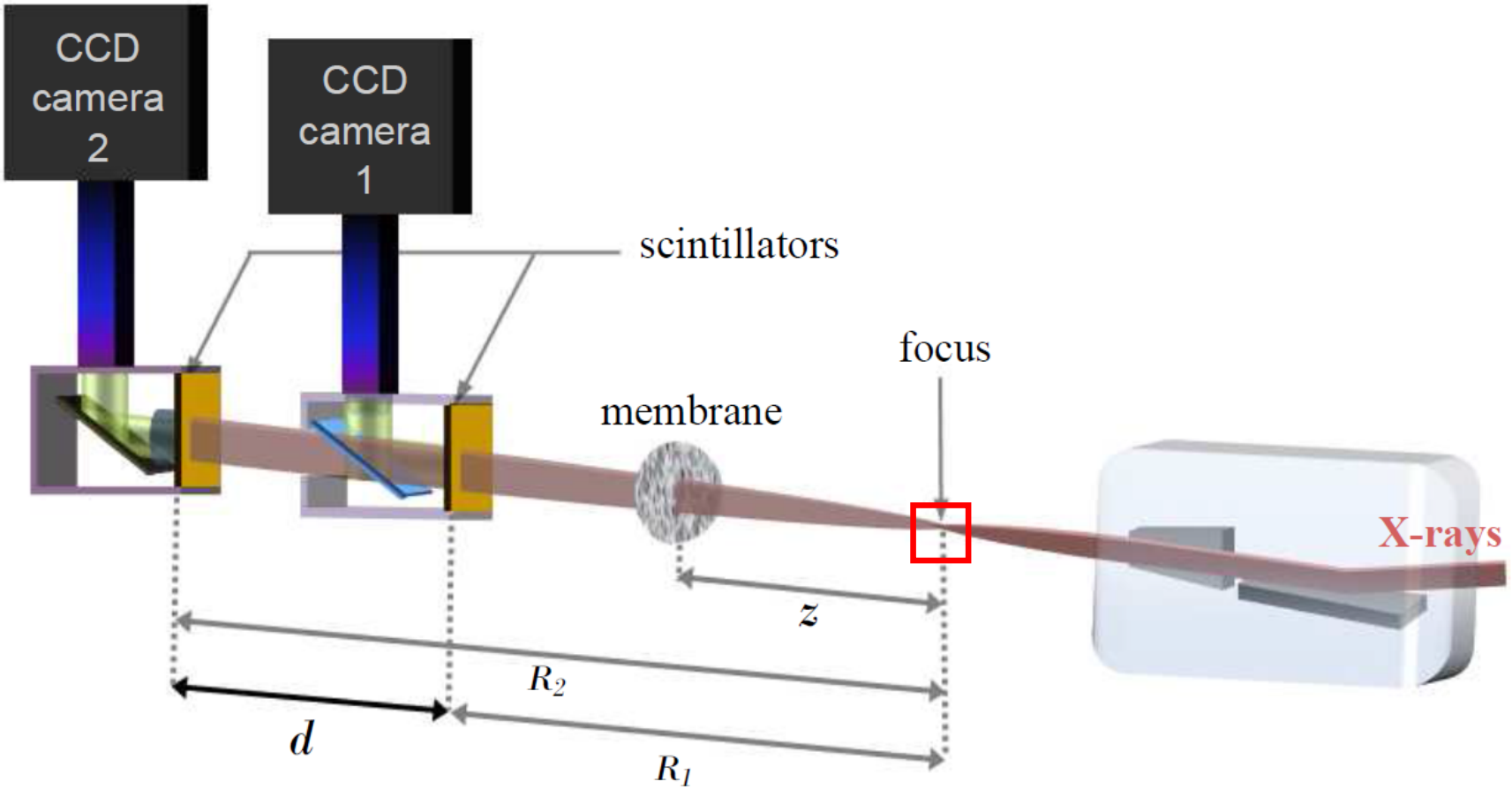

"X-ray pulse wavefront metrology using speckle tracking", Bérujon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

# Versuchsaufbau



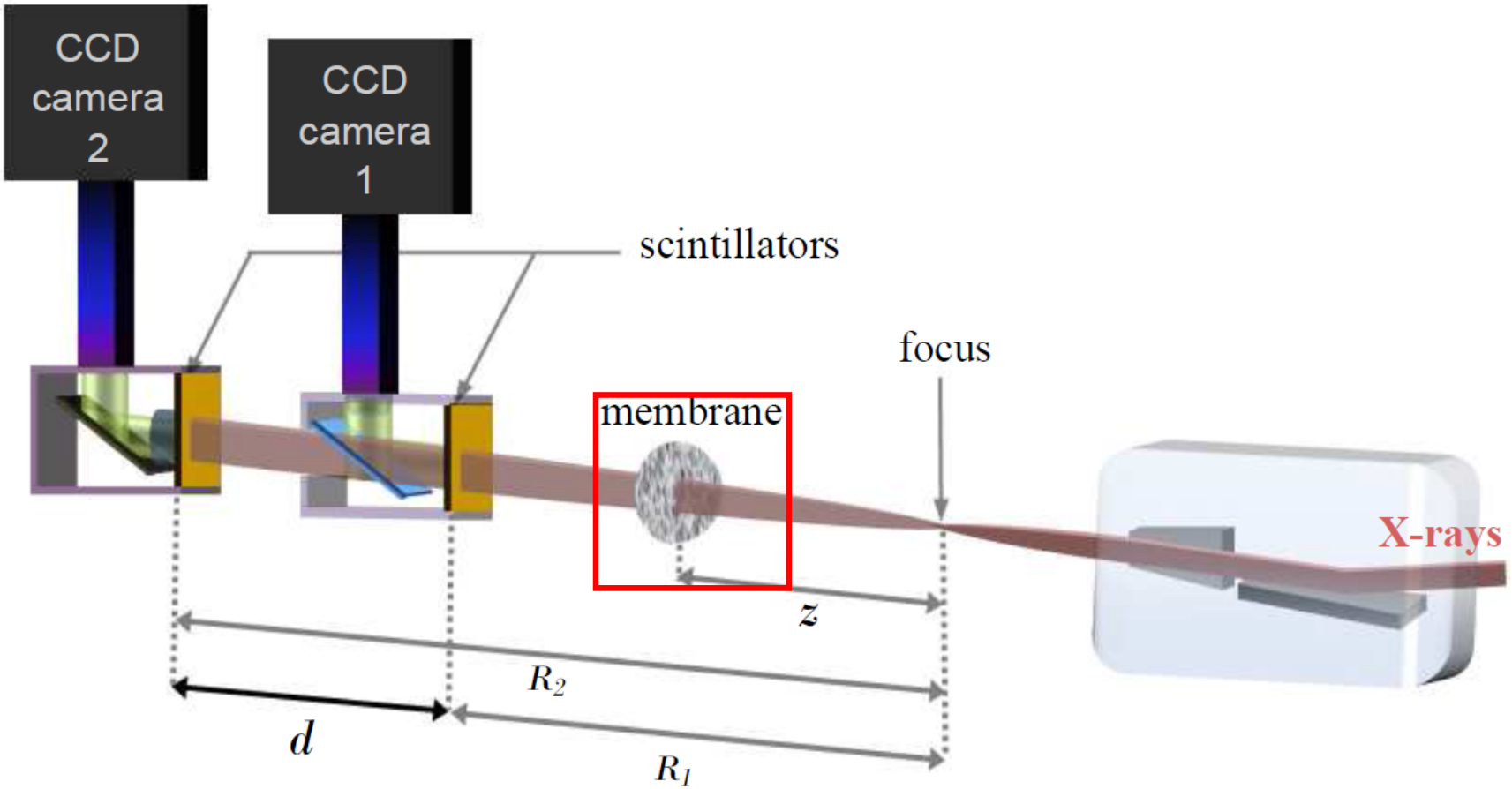

"X-ray pulse wavefront metrology using speckle tracking", Bérujon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

# Versuchsaufbau



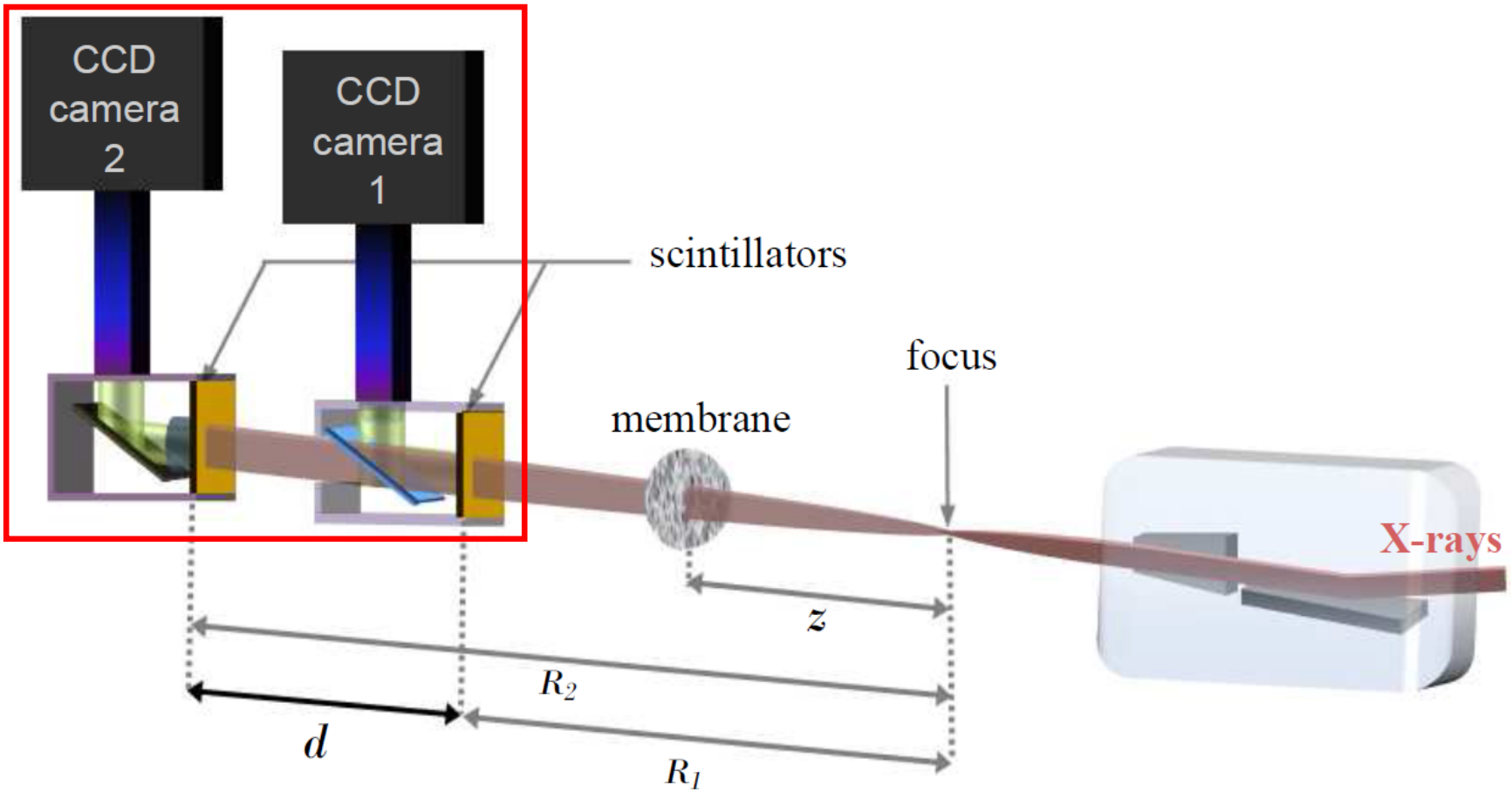

"X-ray pulse wavefront metrology using speckle tracking", Bérujon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

# Versuchsaufbau



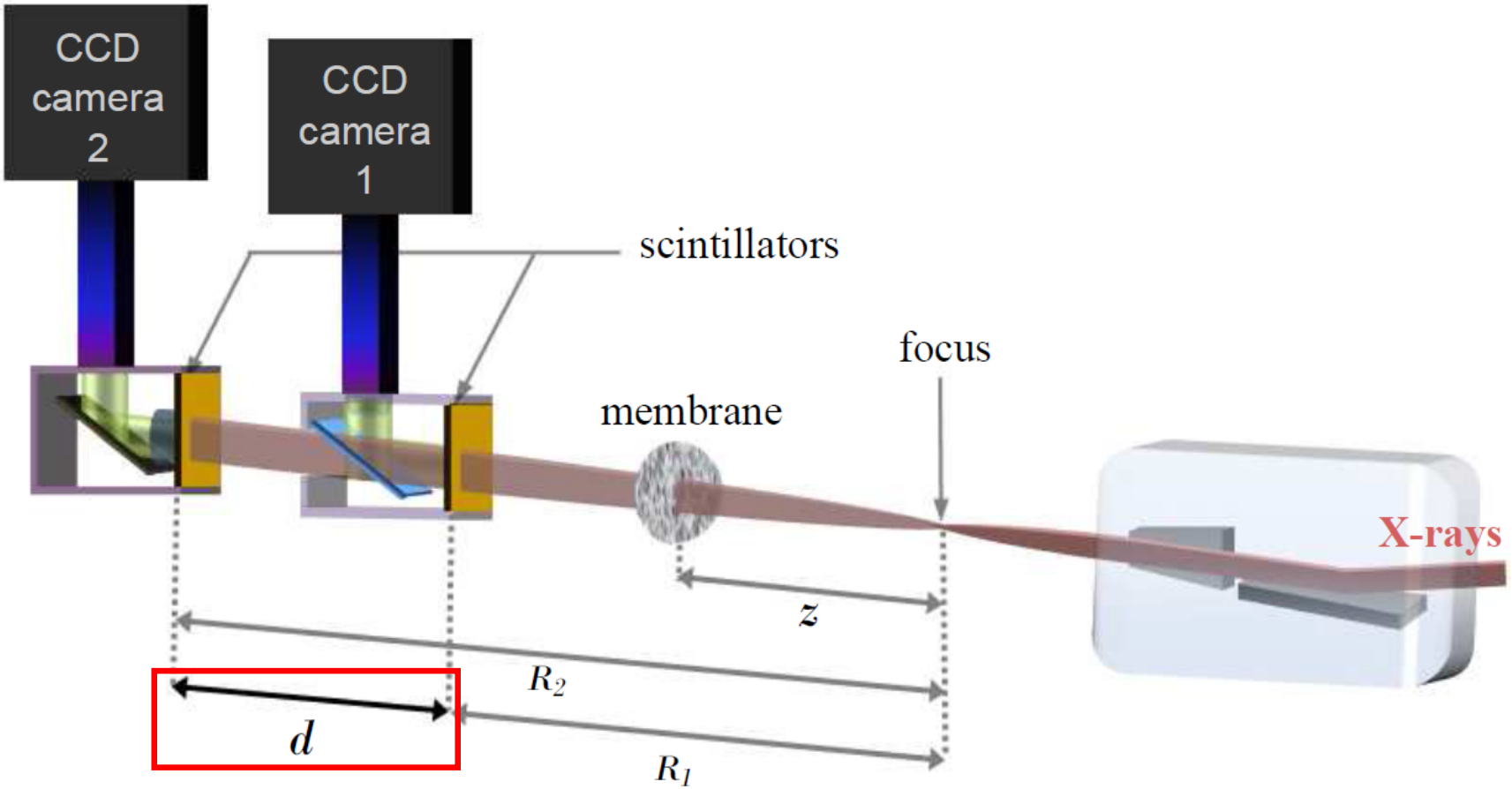

"X-ray pulse wavefront metrology using speckle tracking", Bérujon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

# Versuchsaufbau



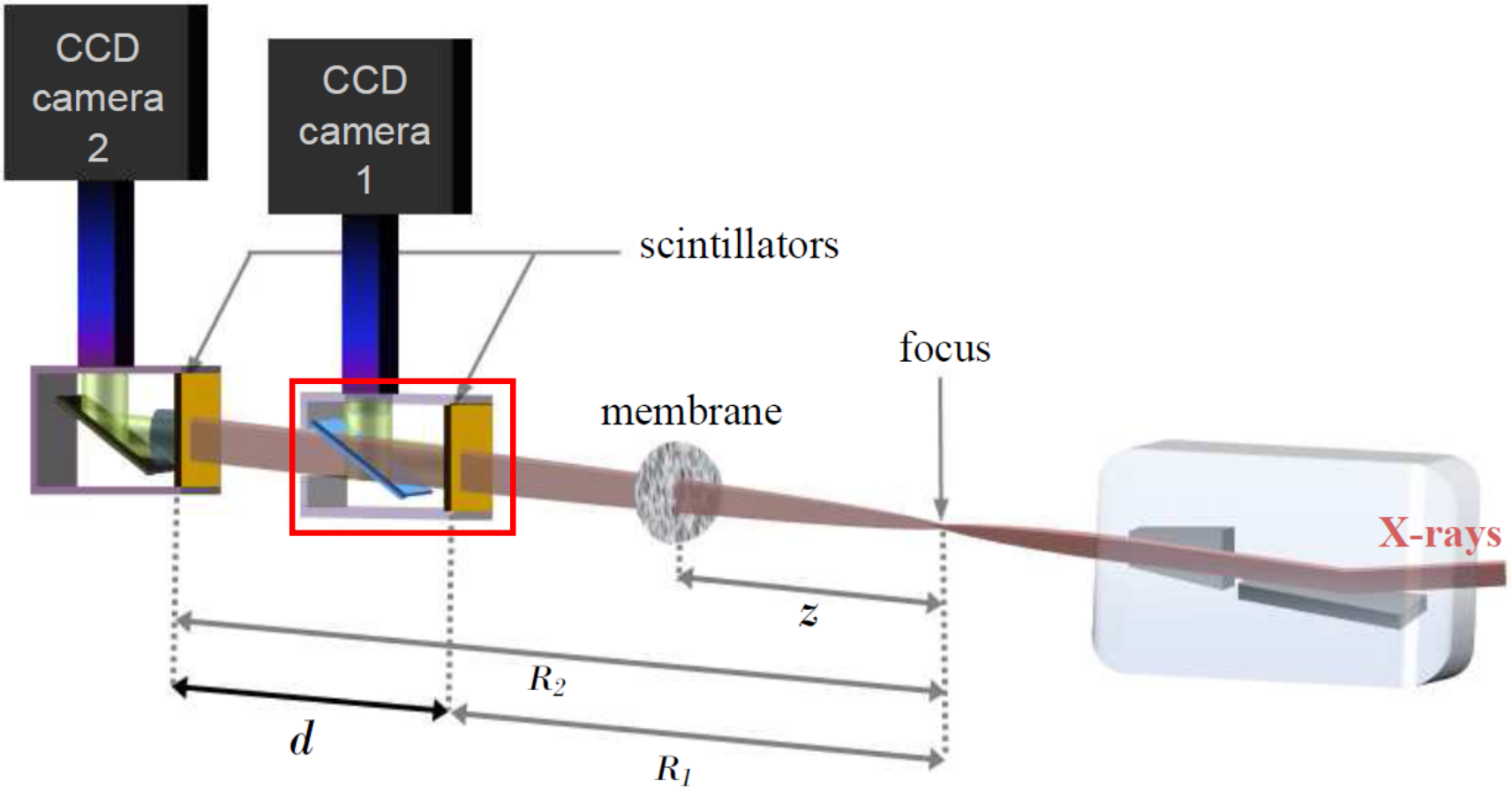

"X-ray pulse wavefront metrology using speckle tracking", Bérujon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

# Versuchsaufbau



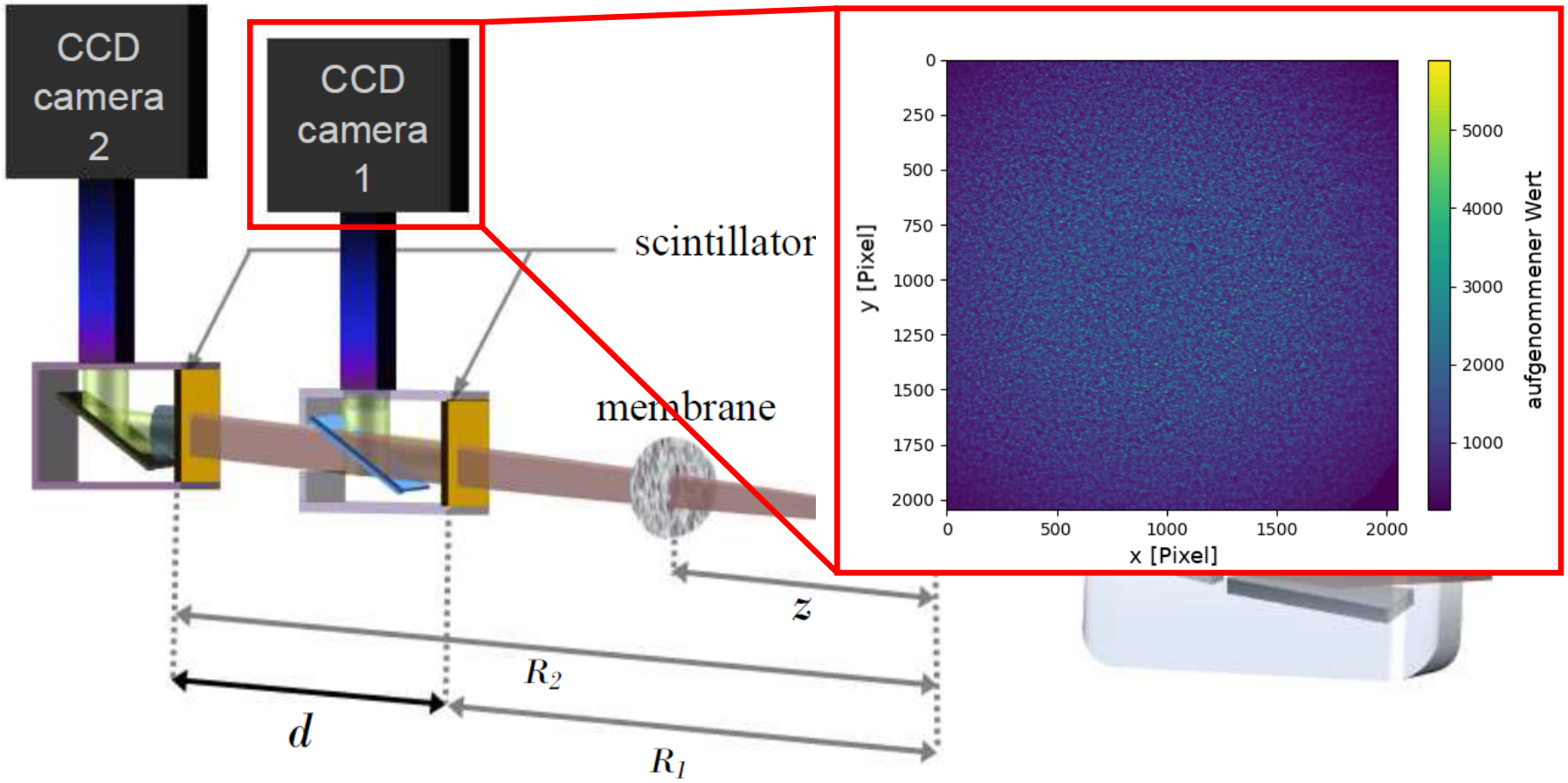

"X-ray pulse wavefront metrology using speckle tracking", Bérujon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

# Versuchsaufbau



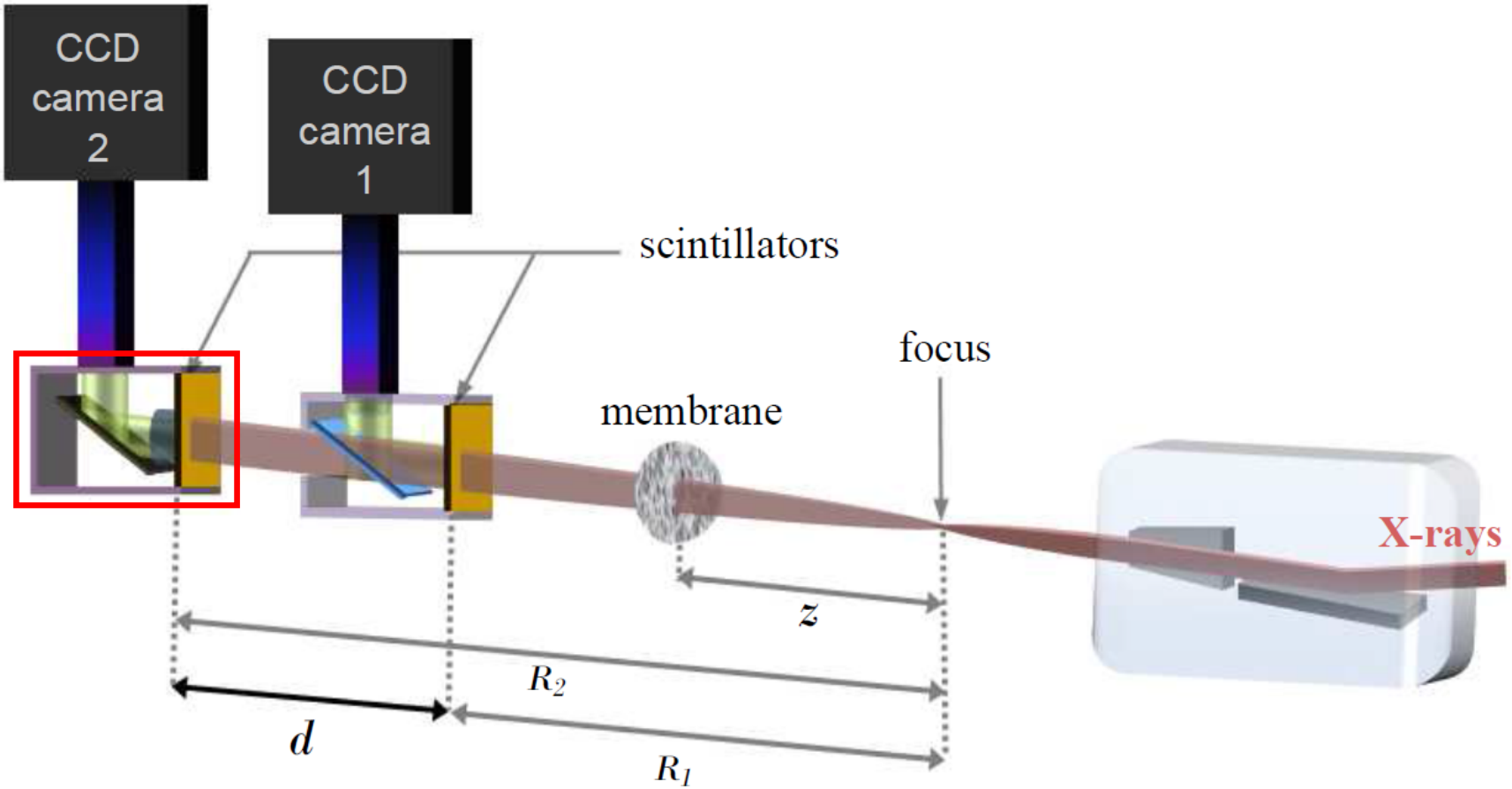

"X-ray pulse wavefront metrology using speckle tracking", Bérujon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

# Versuchsaufbau



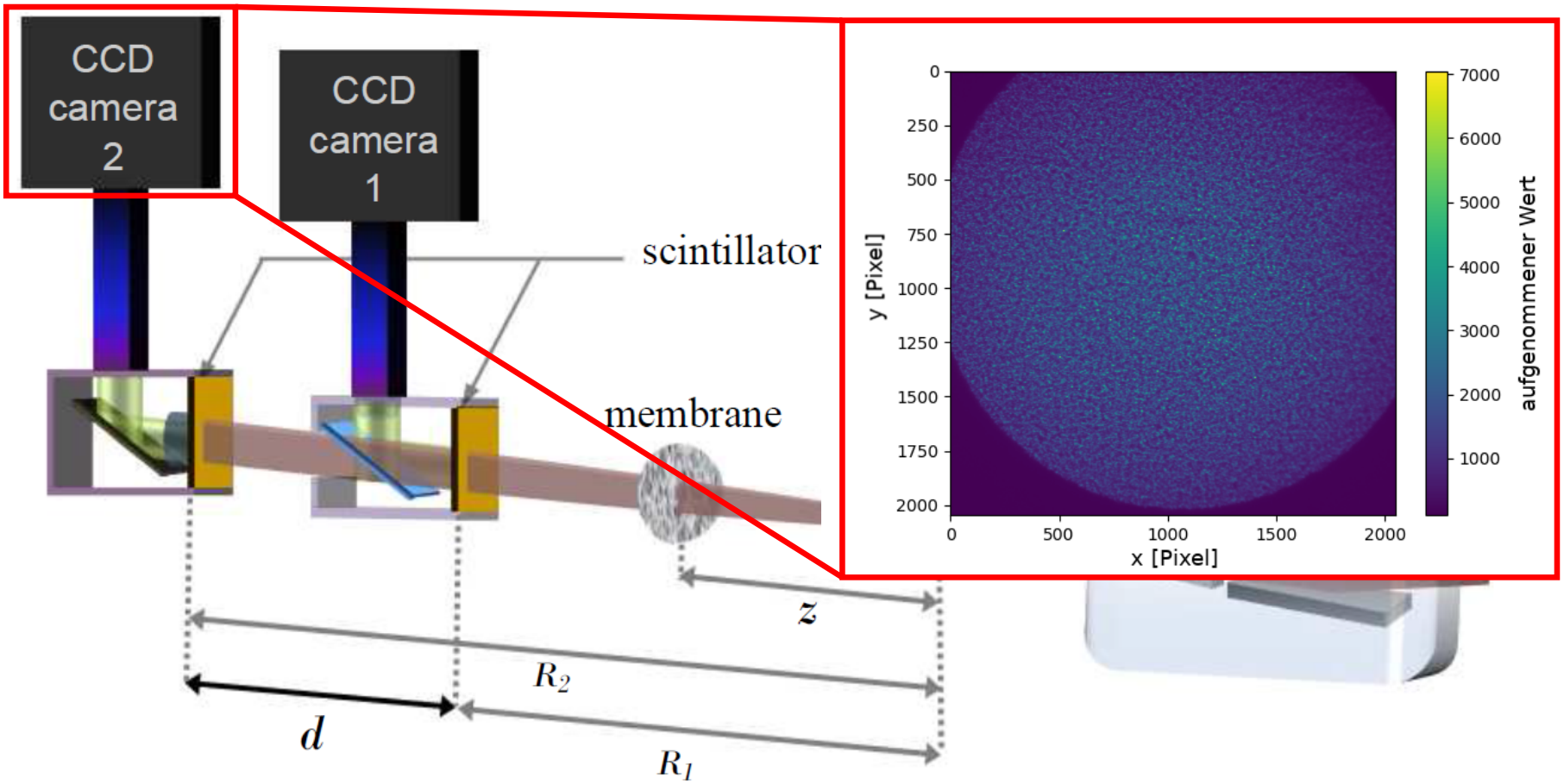

"X-ray pulse wavefront metrology using speckle tracking", Bérujon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

# Versuchsaufbau




"X-ray pulse wavefront metrology using speckle tracking", Bérujon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

# Versuchsaufbau



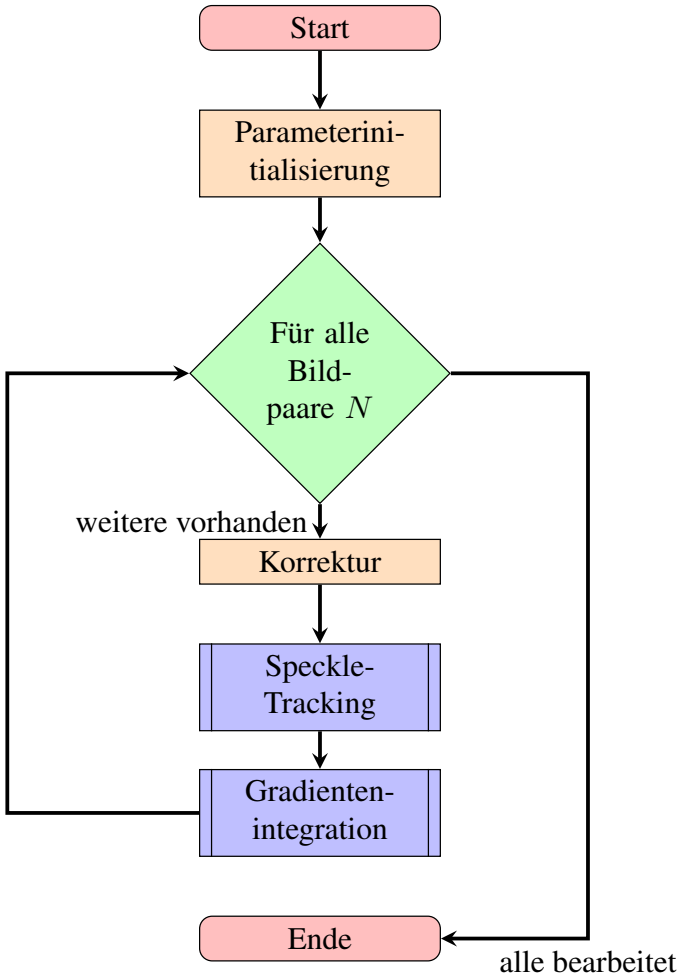

"X-ray pulse wavefront metrology using speckle tracking", Bérujon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

# Überblick

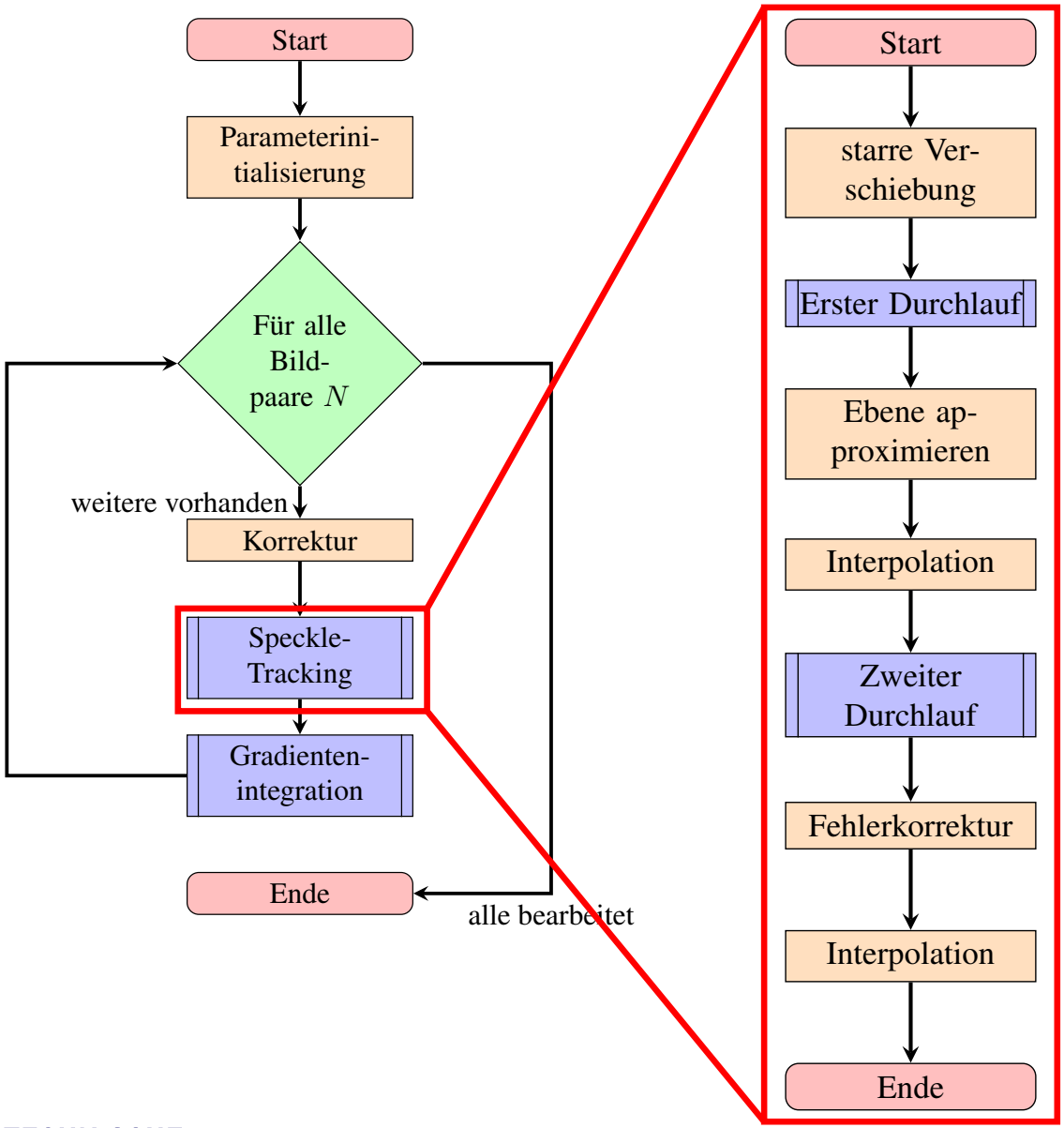
**Hauptroutine:**

- Korrigieren von Kamerafehlern
- Ablenkung nachverfolgen
- Wellenfront rekonstruieren

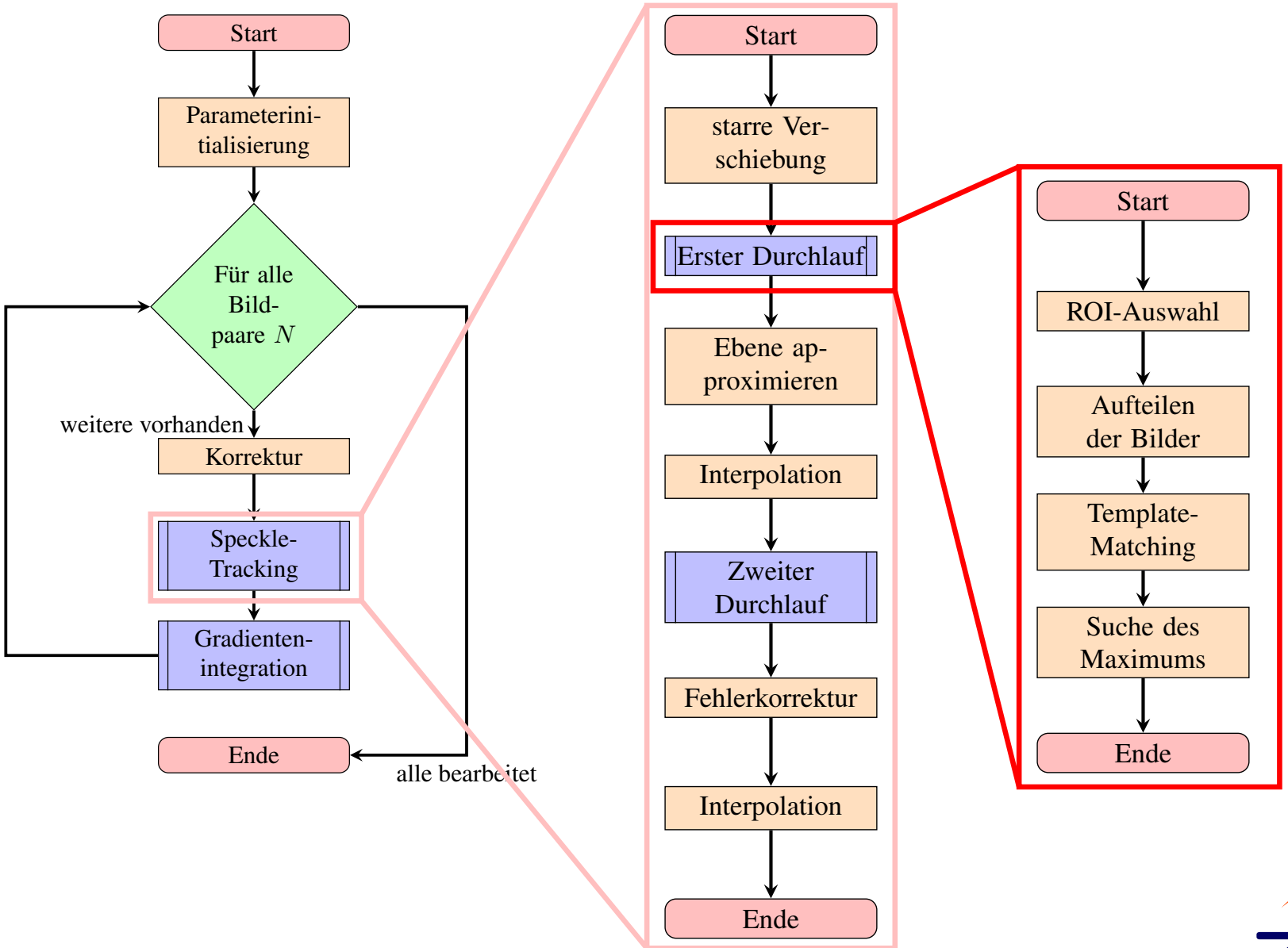
# Hauptroutine



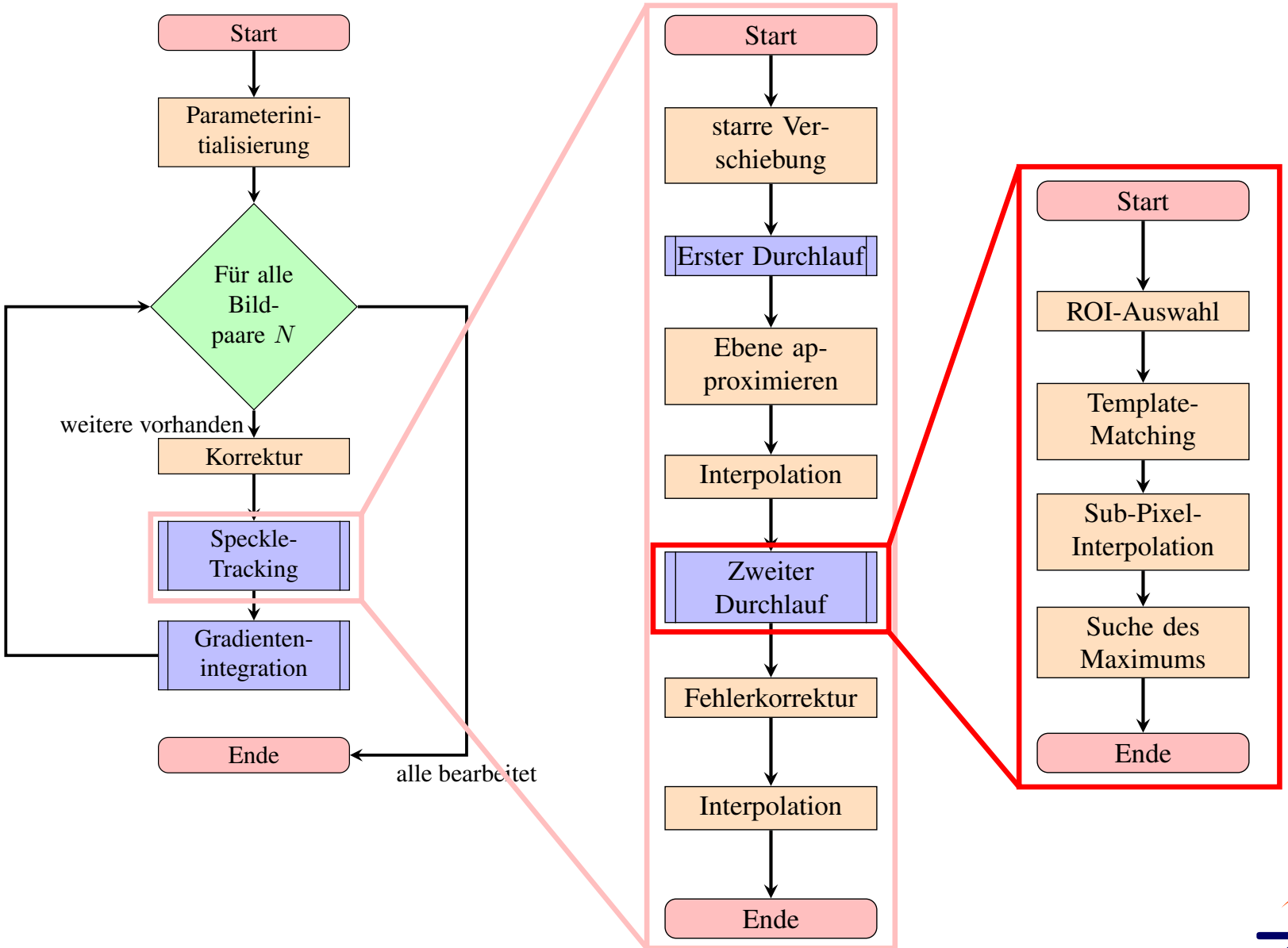
# Hauptroutine



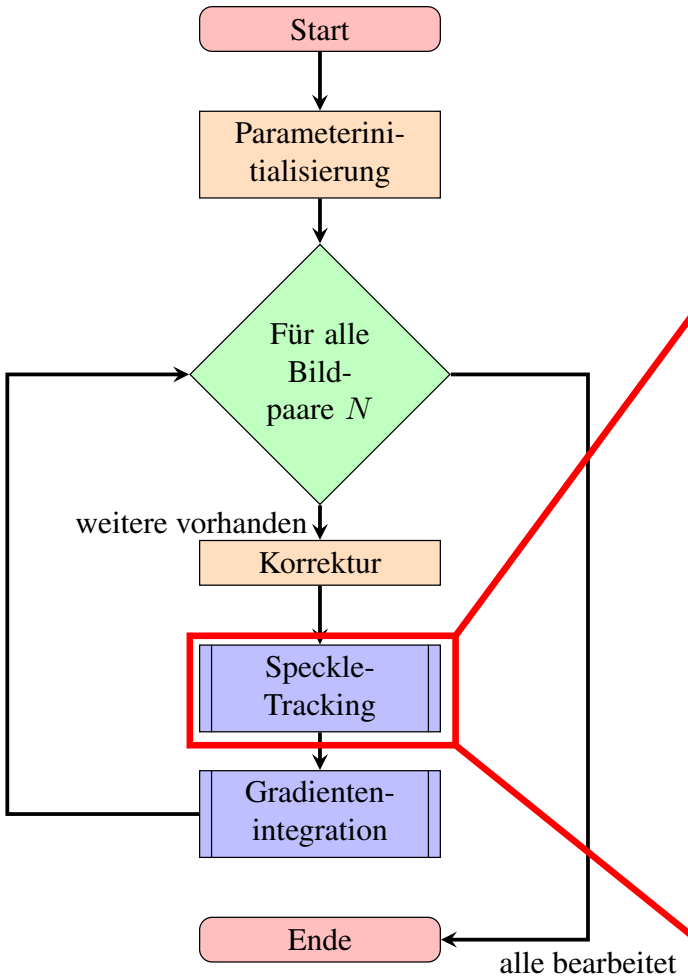
# Hauptroutine

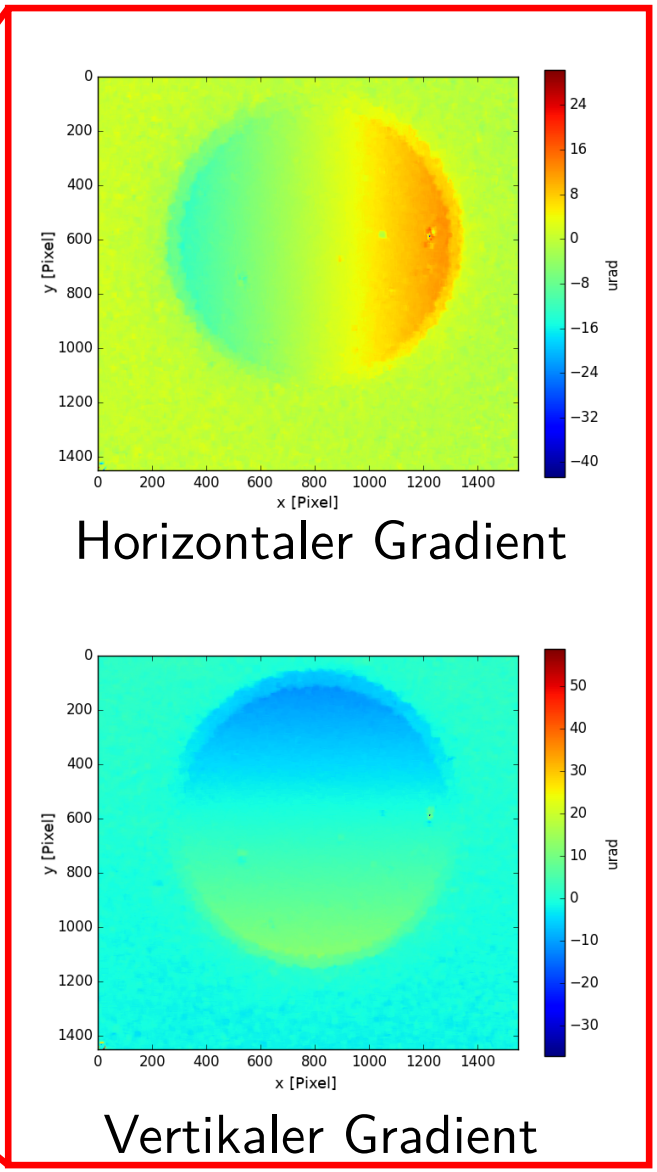


# Hauptroutine

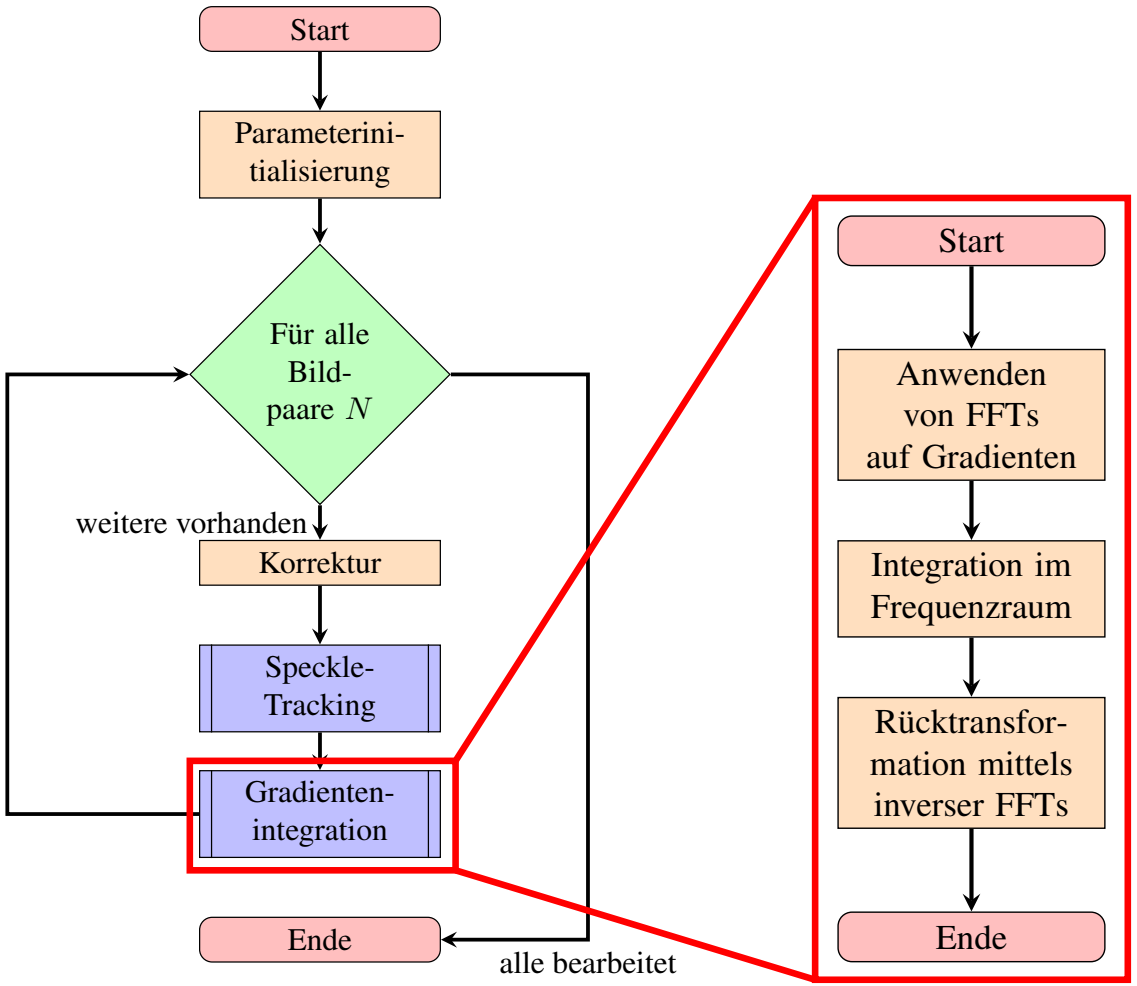


# Hauptroutine



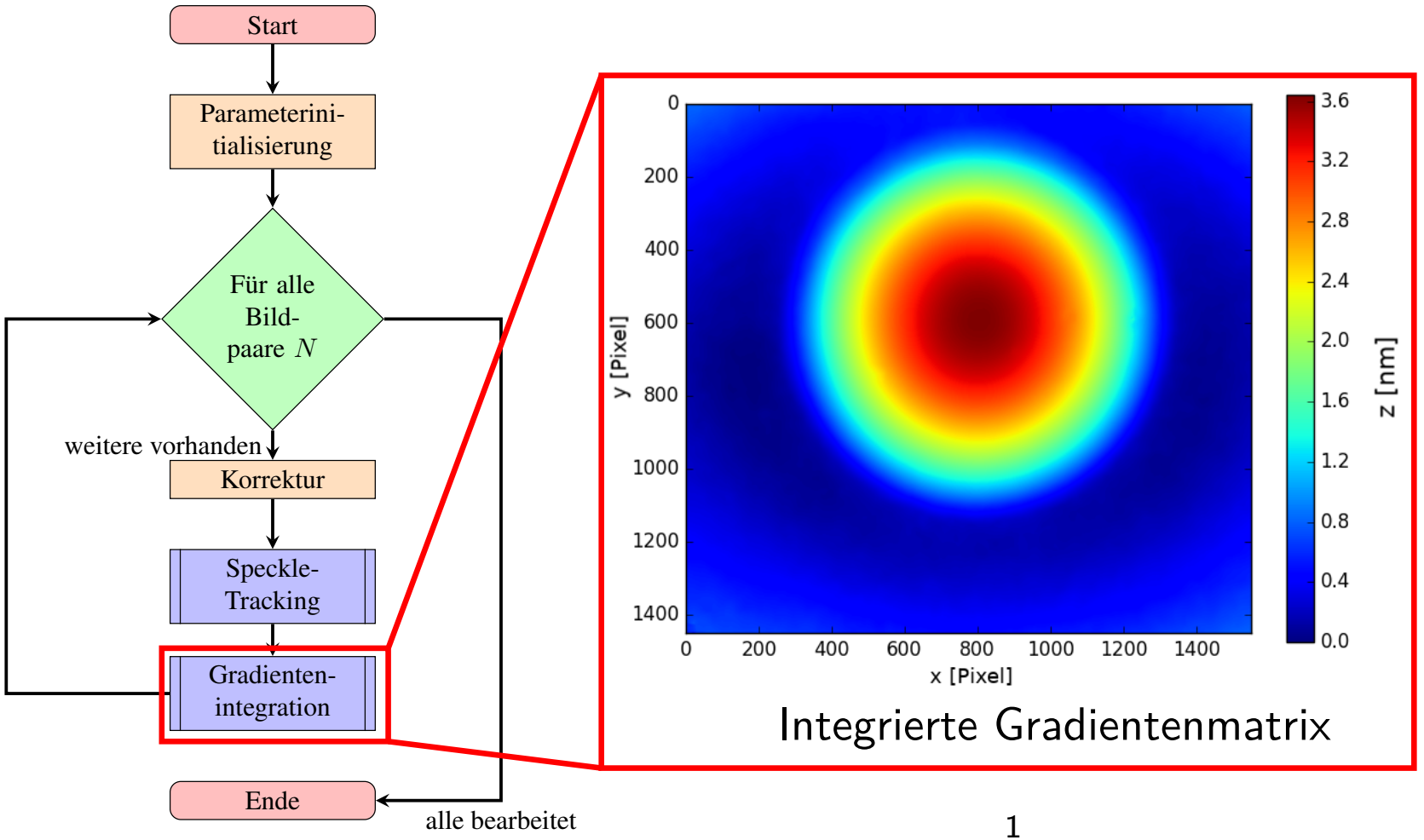
# Hauptroutine




[1]

[1] "A method for enforcing integrability in shape from shading algorithms", Frankot und Chellappa 1988, *IEEE Transactions on Pattern Analysis and Machine Intelligence*

# Hauptroutine




[1]

[1] "A method for enforcing integrability in shape from shading algorithms", Frankot und Chellappa 1988, *IEEE Transactions on Pattern Analysis and Machine Intelligence*

# Gliederung

# Komplexität

$R$ – Anzahl der Pixel; $R_{corr}$ – Korrelationsgröße; $n$ – Anzahl der Bilder

**Hauptroutine:**

| | |
|---|---|
| Bild-Präprozessierschritte | $\mathcal{O}(R)$ |
| Speckle-Tracking | $\mathcal{O}\left(R \cdot R_{corr} \cdot log\left(R_{corr}\right)\right)$ |
| Rekonstruktion der Wellenfront | $\mathcal{O}(R \cdot log(R))$ |

$\Rightarrow$ **Gesamtkomplexität:** $\mathcal{O}\left(n \cdot R \cdot R_{corr} \cdot log\left(R_{corr}\right)\right)$

# Laufzeitumstände

**Datensätze:**

- Experiment 6
  - ROI Größe: 1450x1450 (Bild), 550x550 (Template)
  - Korrelationsgröße: 91
  - unterschiedliche Pixelgröße
- Lenses
  - ROI Größe: 1450x1550 (Bild), 1450x1550 (Template)
  - Korrelationsgröße: 41
  - gleiche Pixelgröße

**Testsystem:**

- Taurus Knoten
- 2x Intel®Xeon®E5-2680 v3 (12 Kerne) @ 2.50GHz, kein MultiThreading
- Python 2.7

# Laufzeitumstände

**Datensätze:**

- Experiment 6
  - ROI Größe: 1450x1450 (Bild), 550x550 (Template)
  - Korrelationsgröße: 91
  - unterschiedliche Pixelgröße
- Lenses
  - ROI Größe: 1450x1550 (Bild), 1450x1550 (Template)
  - Korrelationsgröße: 41
  - gleiche Pixelgröße

**Testsystem:**

- Taurus Knoten
- 2x Intel®Xeon®E5-2680 v3 (12 Kerne) @ 2.50GHz, kein MultiThreading
- Python 2.7

# Laufzeitumstände

**Datensätze:**

- Experiment 6
  - ROI Größe: 1450x1450 (Bild), 550x550 (Template)
  - Korrelationsgröße: 91
  - unterschiedliche Pixelgröße
- Lenses
  - ROI Größe: 1450x1550 (Bild), 1450x1550 (Template)
  - Korrelationsgröße: 41
  - gleiche Pixelgröße

**Testsystem:**

- Taurus Knoten
- 2x Intel®Xeon®E5-2680 v3 (12 Kerne) @ 2.50GHz, kein MultiThreading
- Python 2.7

# Laufzeitumstände

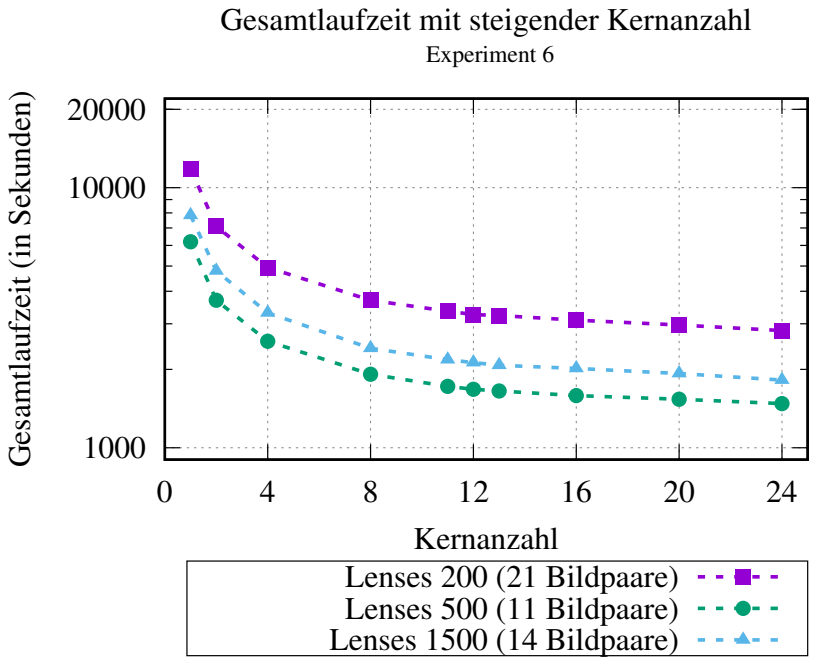
**Datensätze:**

- Experiment 6
  - ROI Größe: 1450x1450 (Bild), 550x550 (Template)
  - Korrelationsgröße: 91
  - unterschiedliche Pixelgröße
- Lenses
  - ROI Größe: 1450x1550 (Bild), 1450x1550 (Template)
  - Korrelationsgröße: 41
  - gleiche Pixelgröße

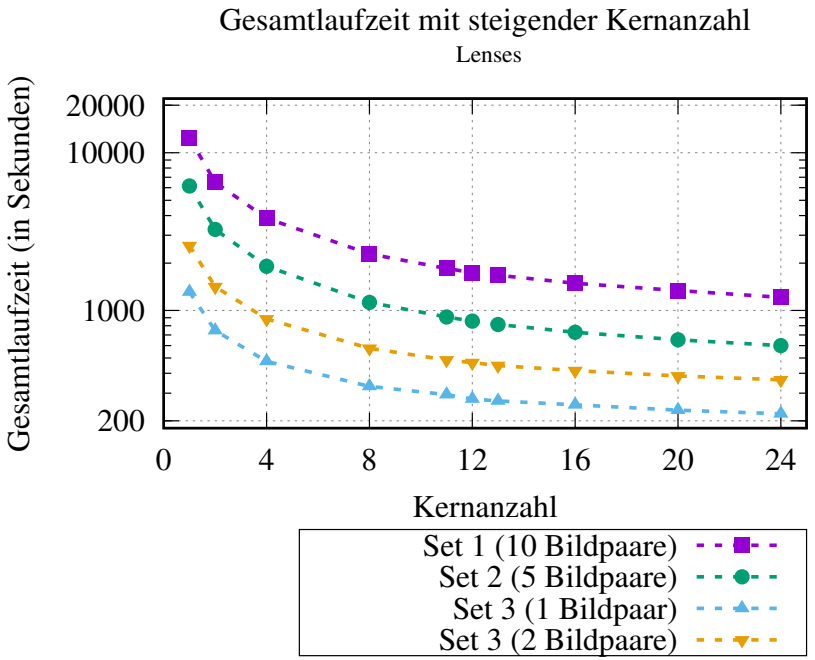
**Testsystem:**

- Taurus Knoten
- 2x Intel®Xeon®E5-2680 v3 (12 Kerne) @ 2.50GHz, kein MultiThreading
- Python 2.7

# Gesamtlaufzeit



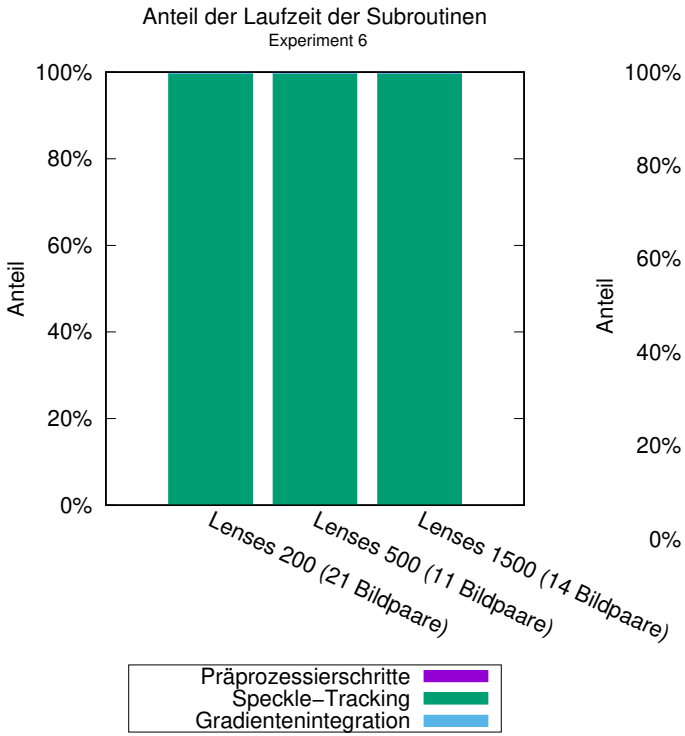

(a) Experiment 6



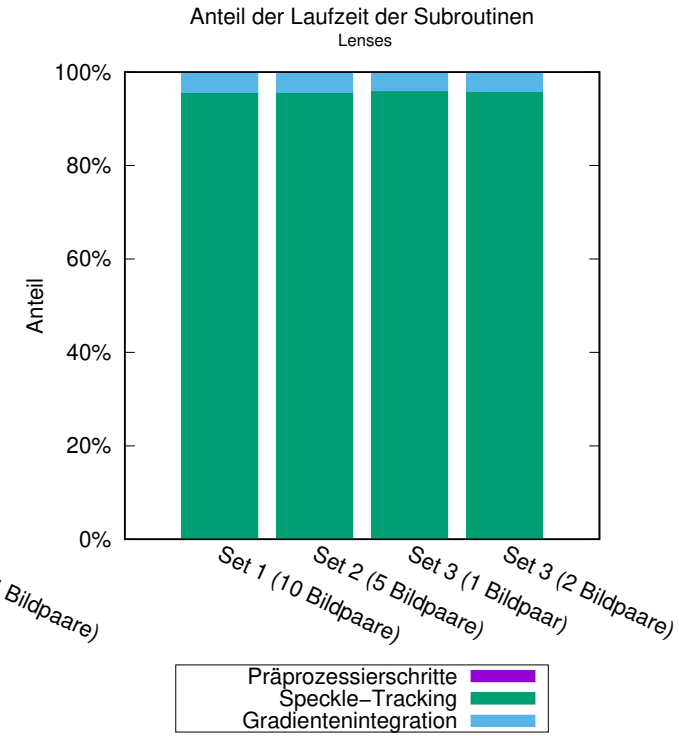

(b) Lenses

Abbildung: Gesamtlaufzeiten

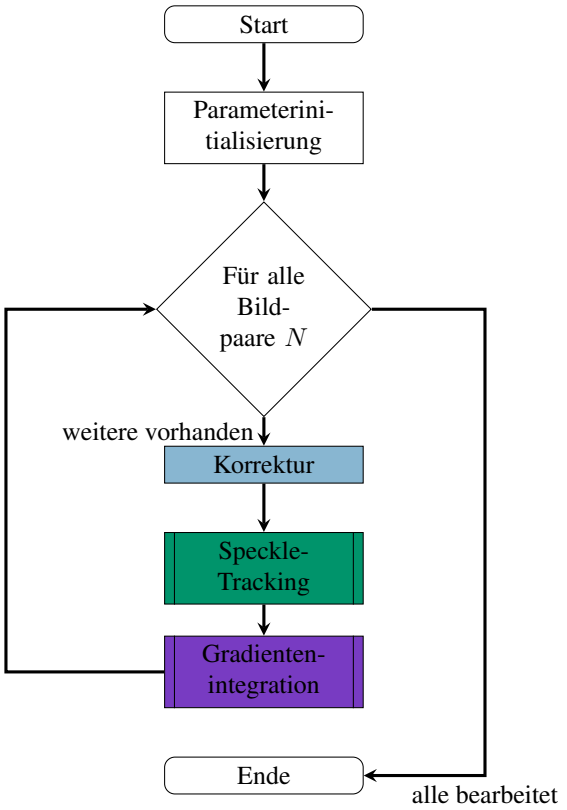
# Profiling I




(a) Experiment 6



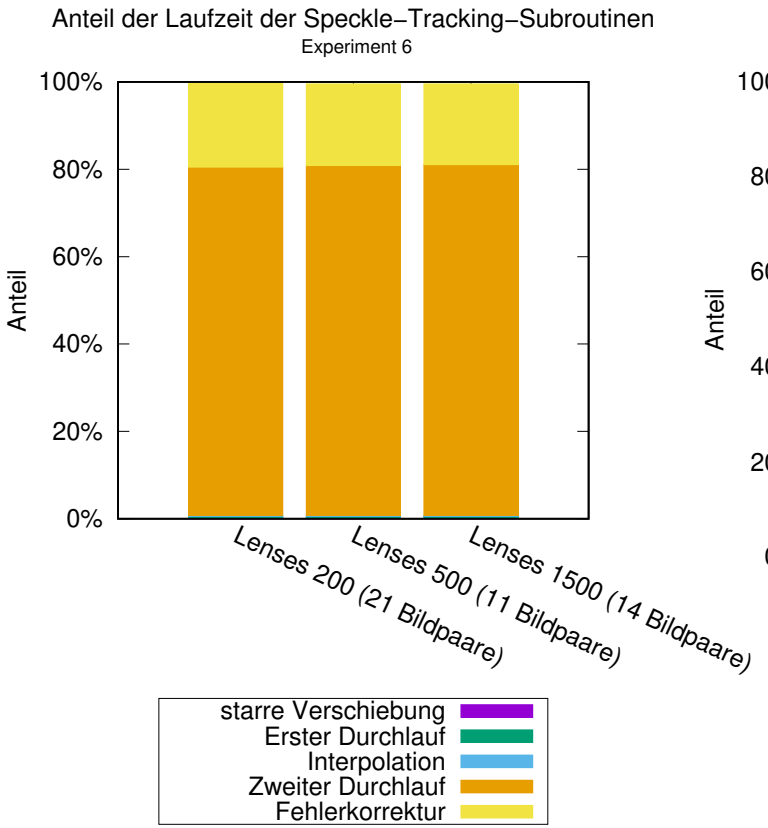

(b) Lenses



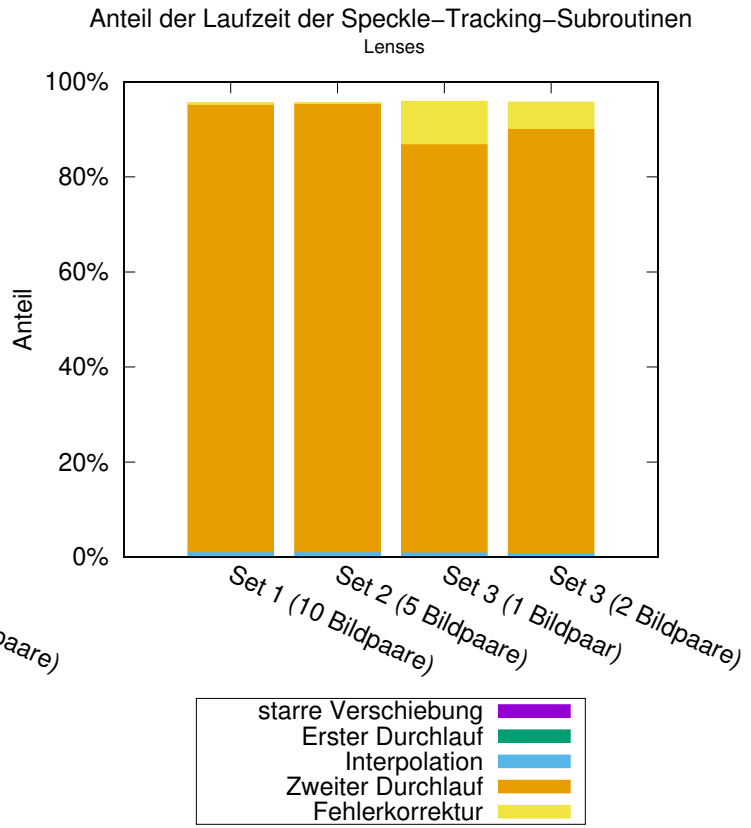

(c) Algorithmus

Abbildung: Laufzeitanteile der Subroutinen

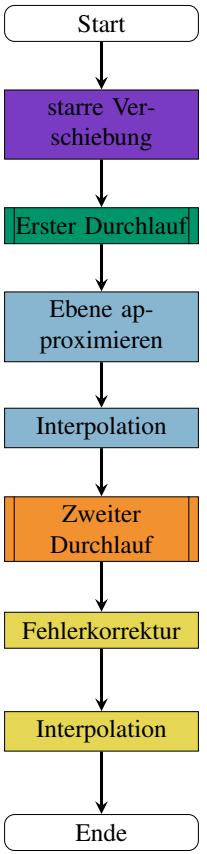
# Profiling II




(a) Experiment 6



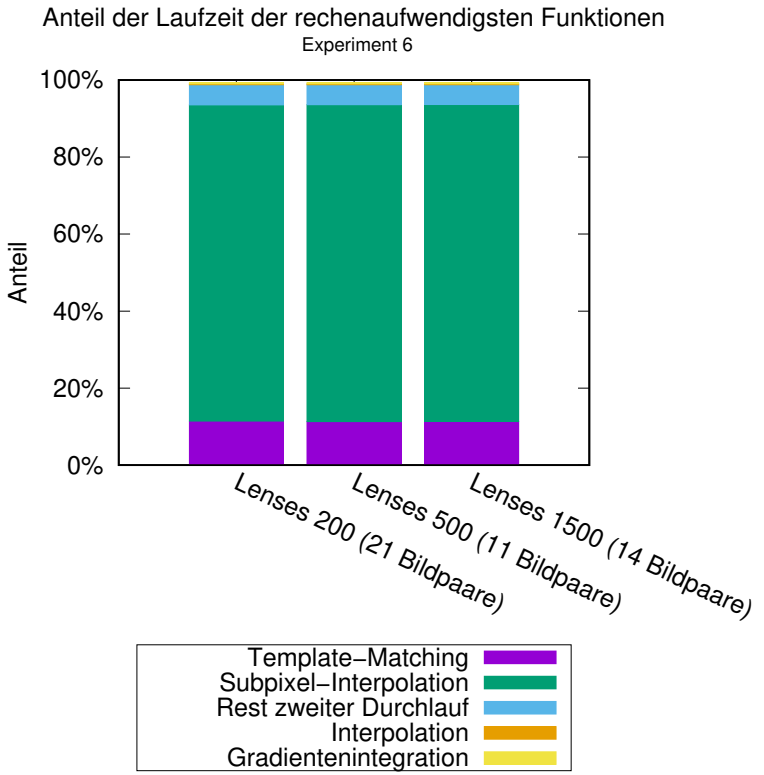

(b) Lenses



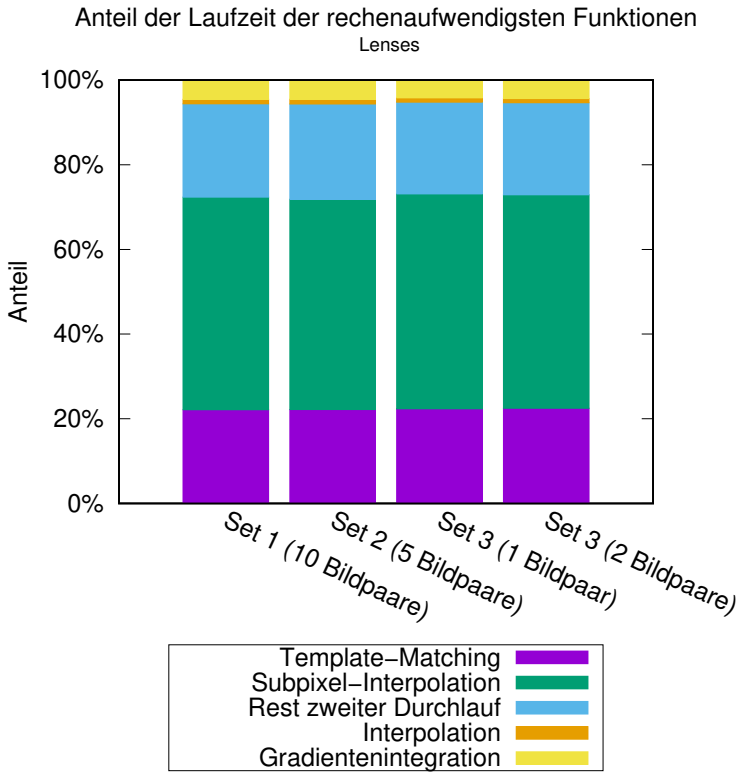

(c) Algorithmus

Abbildung: Laufzeitanteile der Speckle-Tracking-Routinen

# Profiling III




(a) Experiment 6



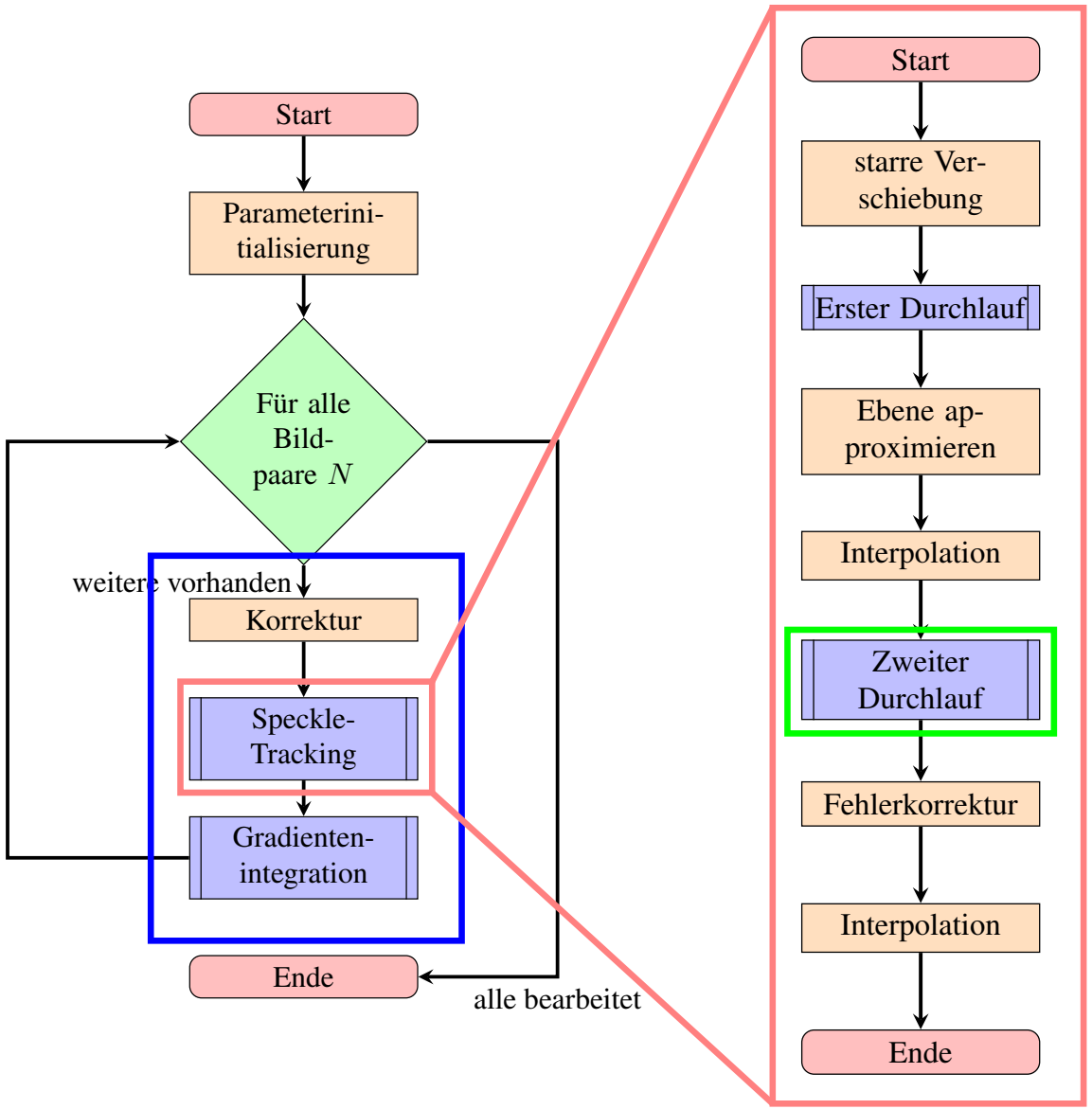

(b) Lenses

Abbildung: Laufzeitanteile der rechenaufwendigsten Funktionen

$\Rightarrow$ **über 95%**

# Gliederung

# Parallele Verarbeitung einzelner Bildpaare




multiprocessing

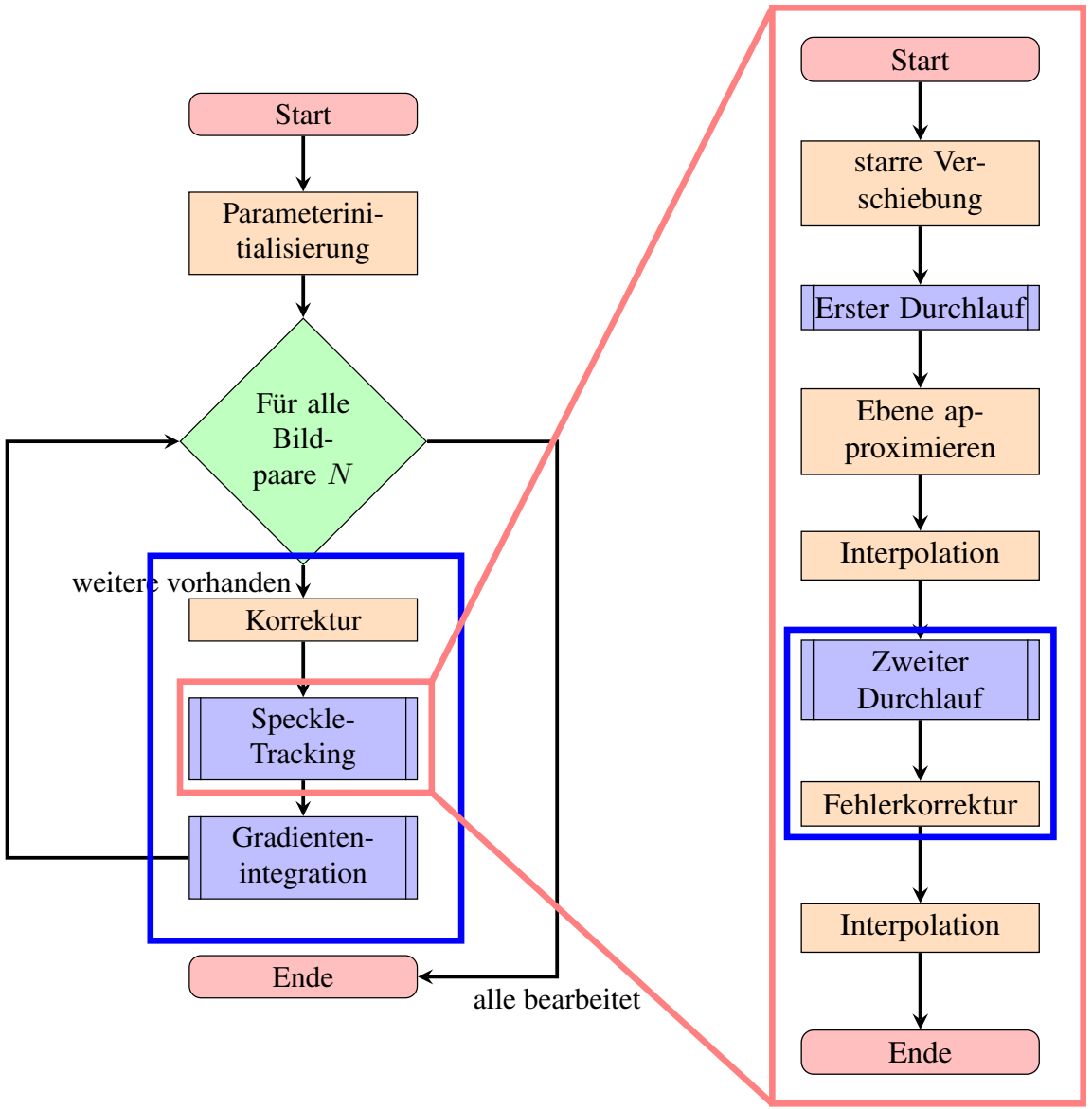
MPI

# Parallele Verarbeitung einzelner Bildpaare

**Benennung:** *mpi*

1. Verteilung einzelner Bildpaare auf Rechenkerne
2. parallele Verarbeitung mittels MPI und multiprocessing
3. Sammeln der Daten auf dem Masterkern

# Parallele Verarbeitung innerhalb einzelner Bildpaare




MPI

# Parallele Verarbeitung innerhalb einzelner Bildpaare I

**Benennung:** *mpi-advanced*

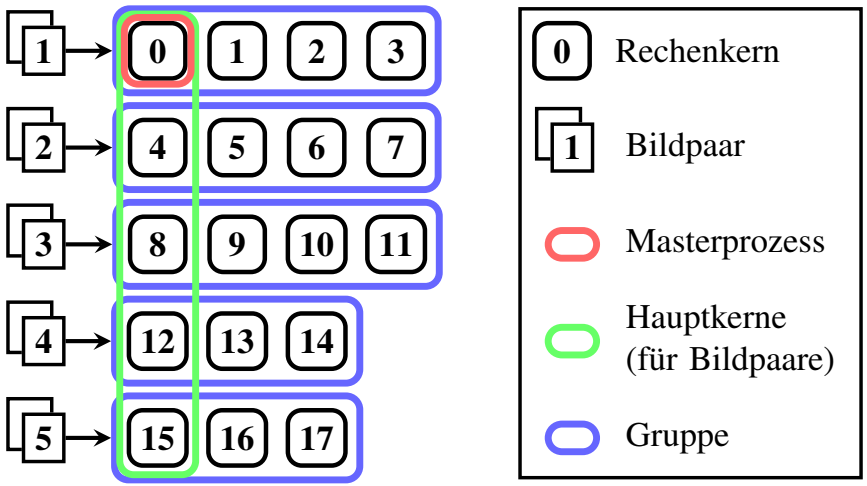



Abbildung: Verteilung von fünf Bildpaaren auf 18 Rechenknoten

# Parallele Verarbeitung innerhalb einzelner Bildpaare II

**Parallelisierung mittels MPI:**

```
1  from distributor import Distributor
2  def initFn():
3      global_args = (1, 2)
4      local_args = [[1, 5], [2, 6], [3, 7], [4, 8]]
5      return (global_args, local_args)
6
7  def addFn(global_args, local_args):
8      (multiply, add) = global_args
9      return local_args * multiply + add
10
11 def mainFn(global_args, local_args, dist):
12     return dist.parallel(addFn, global_args, local_args)
13
14 def exitFn(global_args, result):
15     print("The result is", result)
16     return 0
17
18 dist = Distributor(initFn, mainFn, exitFn)
```

# Parallele Verarbeitung innerhalb einzelner Bildpaare III

**Parallelisierung mittels joblib:**

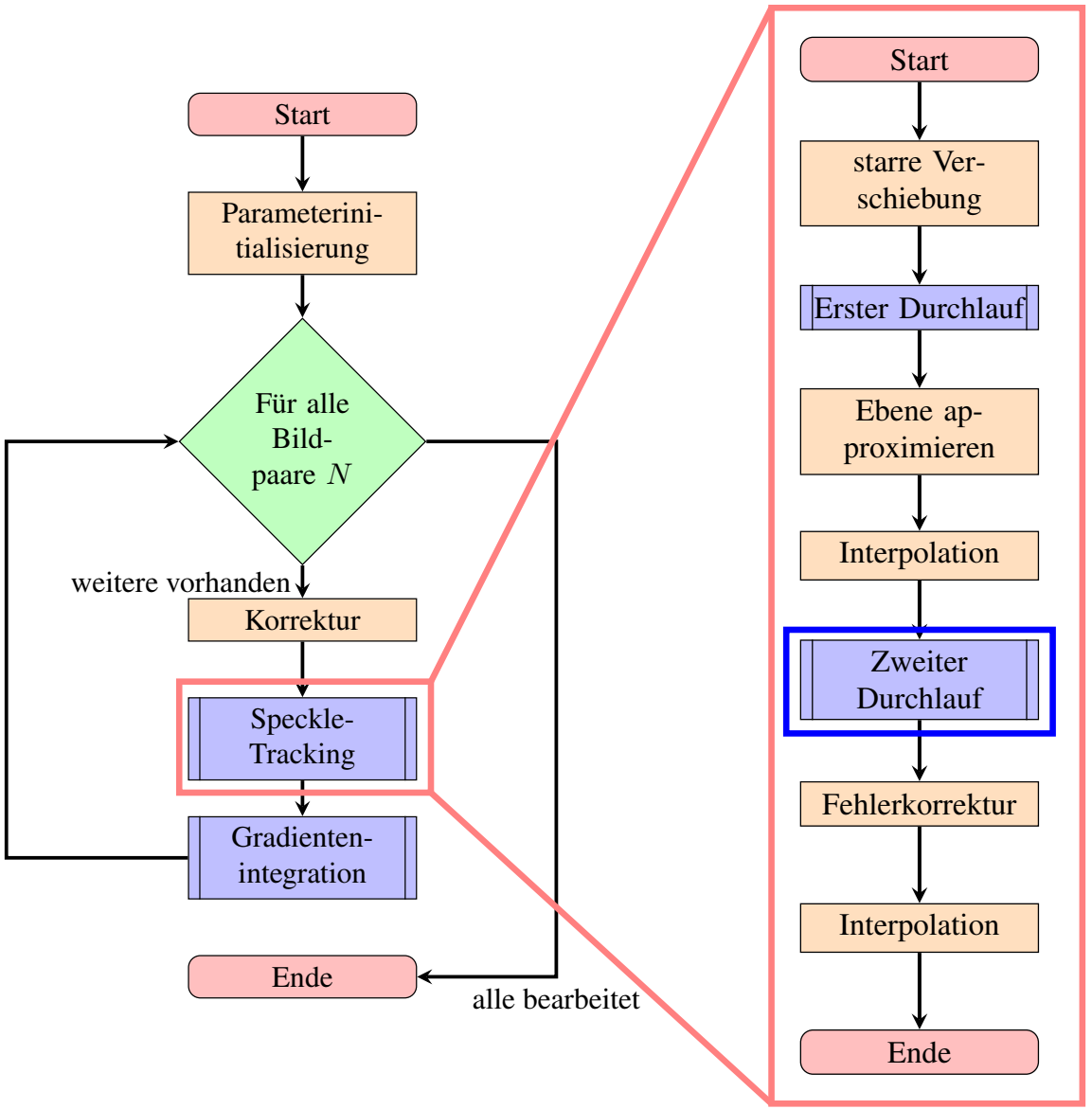
```
1  from joblib import Parallel, delayed
2  import multiprocessing
3
4  def addFn(array, multiply, add):
5      result = []
6      for element in array:
7          result += [element * multiply + add]
8      return result
9
10 matrix = [[1, 5], [2, 6], [3, 7], [4, 8]]
11 multiply = 1
12 add = 2
13 n_jobs = len(matrix)
14 result = Parallel()(delayed(addFn)(matrix[k], multiply, add) for
       k in range(n_jobs))
15 print("The result is", result)
```

# Parallele Verarbeitung innerhalb einzelner Bildpaare IV

**Ausgabe**:

```
1 The result is [[3, 7], [4, 8], [5, 9], [6, 10]]
```

# Nutzen bereits optimierter Funktionen



# Nutzen bereits optimierter Funktionen I

**Benennung:** *intrinsics*

**Finden des Maximums einer Matrix:**

```
1 for i in range(1, lengthY - 1):
2     for j in range(1, lengthX - 1):
3         if(nxcorr[i, j] > maxValue):
4             maxValue = nxcorr[i, j]
5             maxI = i
6             maxJ = j
```

# Nutzen bereits optimierter Funktionen II

**Finden des Maximums einer Matrix:**

```
1 nxcorr_small = nxcorr[1:-1, 1:-1]
2 (_, maxValue, _, (maxJ, maxI)) = cv2.minMaxLoc(nxcorr_small)
3 maxI += 1
4 maxJ += 1
```

# Nutzen bereits optimierter Funktionen III

**Berechnung des Signal-Rausch-Verhältnisses:**

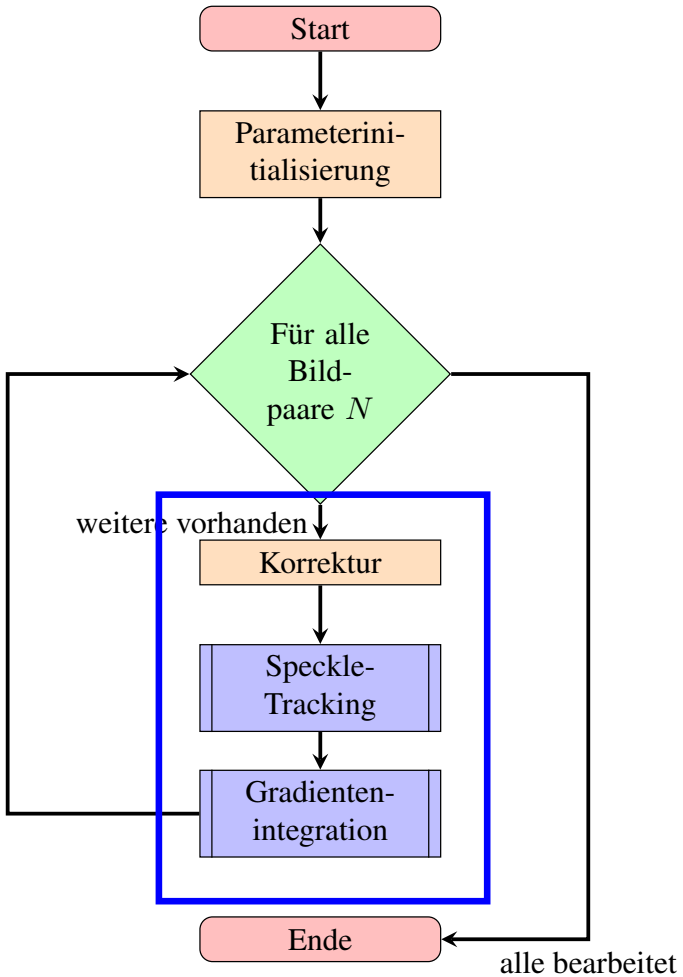
```
1 avg = 0.0
2 count = 0
3 for i in range(lengthY):
4     for j in range(lengthX):
5         if((i is not maxI) and (j is not maxJ)):
6             avg = avg + abs(nxcorr[i,j])
7             count = count + 1
8             avg = avg / float()count)
9             SNr = maxValue / avg
```

# Kompilieren - Gesamtes Programm

**Benennung:** *compiled*

- Übersetzen großer Teile des Python-Codes in C mittels Cython
- Übersetzen des C-Codes in nativen Bytecode

# Kompilieren - Gesamtes Programm

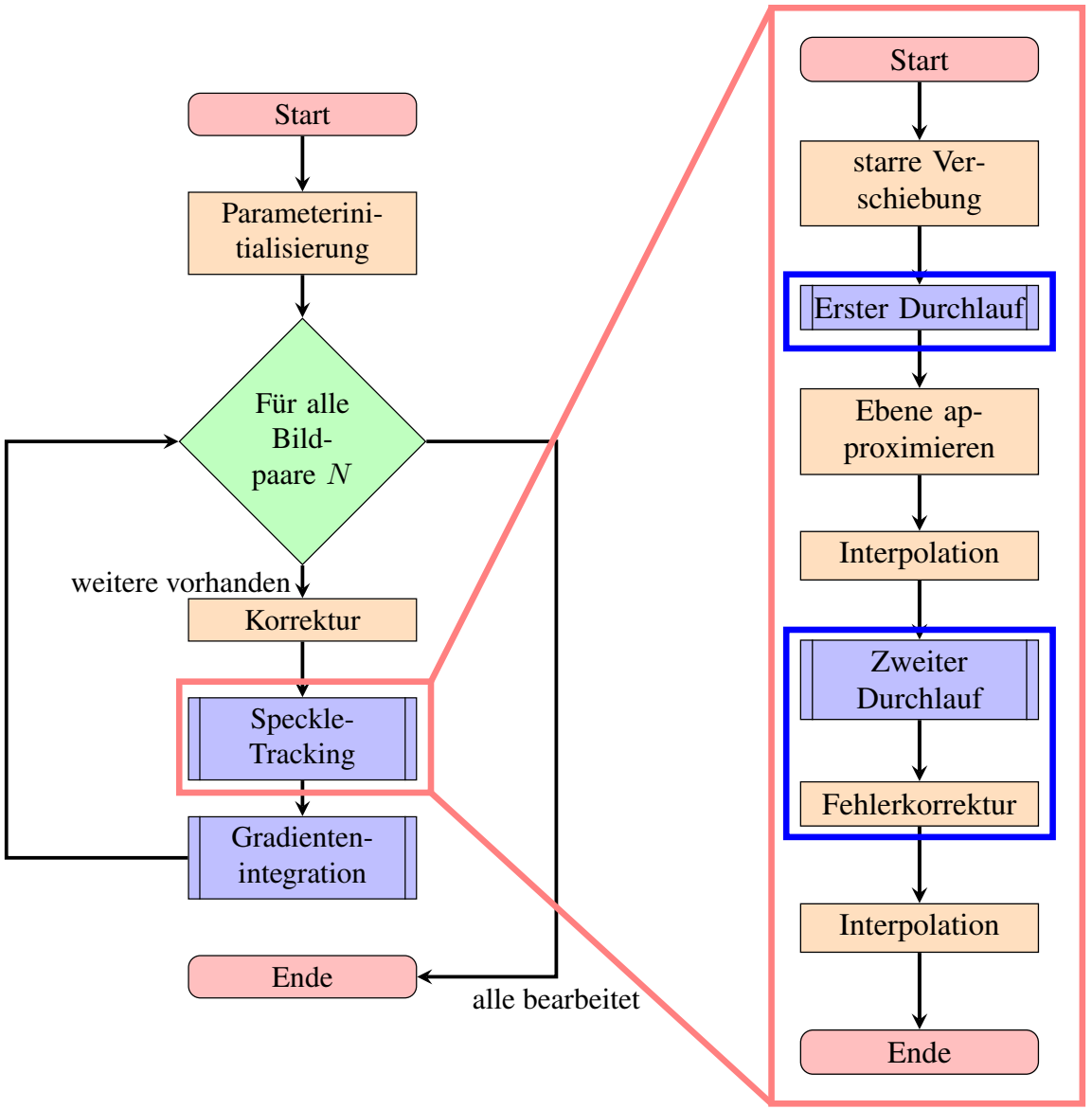


# Kompilieren - numba

**Benennung:** *numba*

- Annotieren der rechenaufwendigsten Funktionen mittels `@jit(cached = True)`
- just-in-time Übersetzung durch numba beim ersten Aufruf
- anschließende Nutzung der Übersetzten Funktionen

# Kompilieren - numba



# Kompilieren - Cython

**Benennung:** *cython*

- Typisierung der rechenaufwendigsten Funktionen
- Übersetzen dieser Funktionen

**Benennung:** *compiled-advanced*

- Typisierung der rechenaufwendigsten Funktionen
- Übersetzen großer Teile des Python-Codes

# Kompilieren - Cython

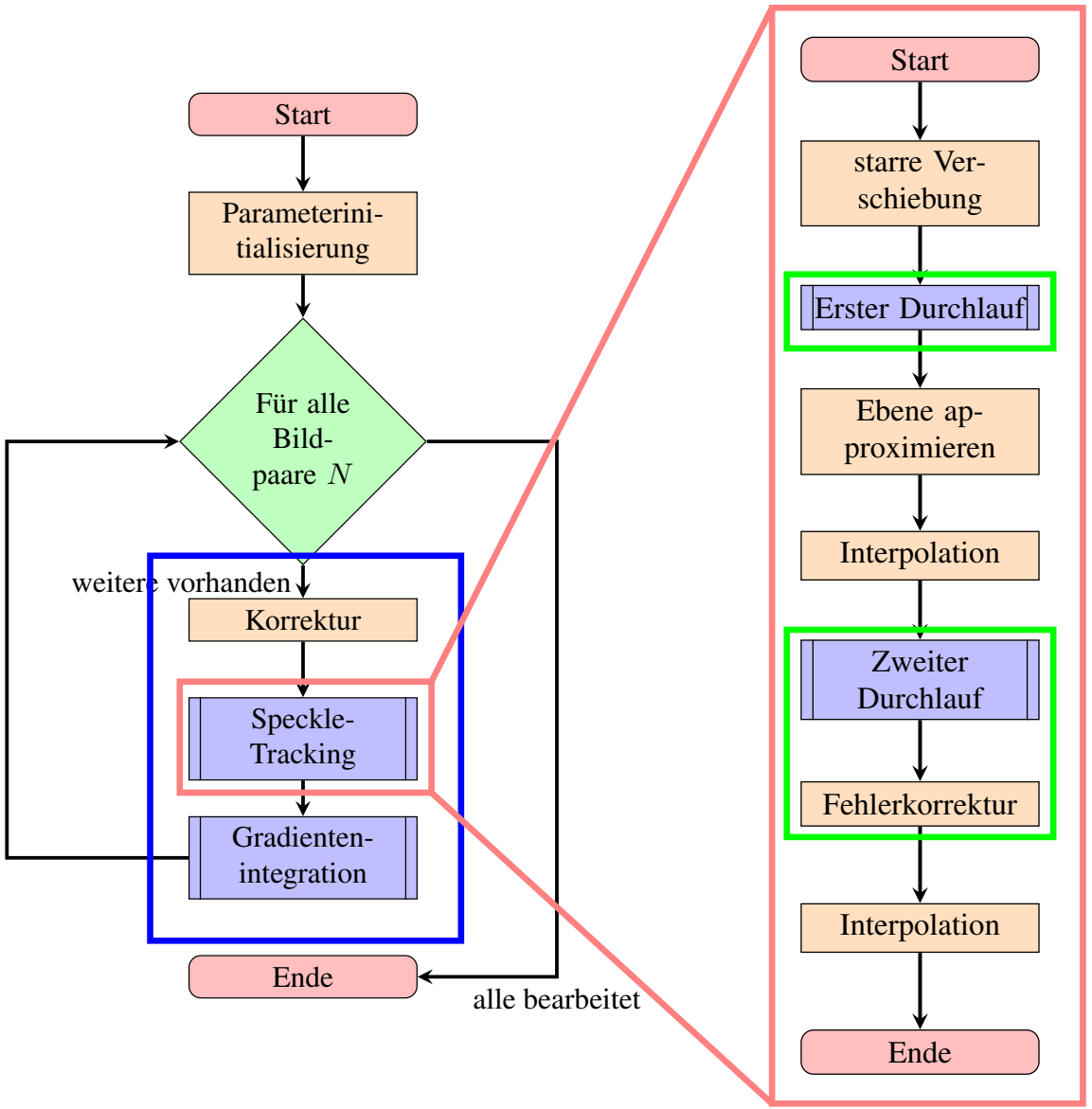
**Benennung:** *cython*

- Typisierung der rechenaufwendigsten Funktionen
- Übersetzen dieser Funktionen

**Benennung:** *compiled-advanced*

- Typisierung der rechenaufwendigsten Funktionen
- Übersetzen großer Teile des Python-Codes

# Kompilieren - Cython

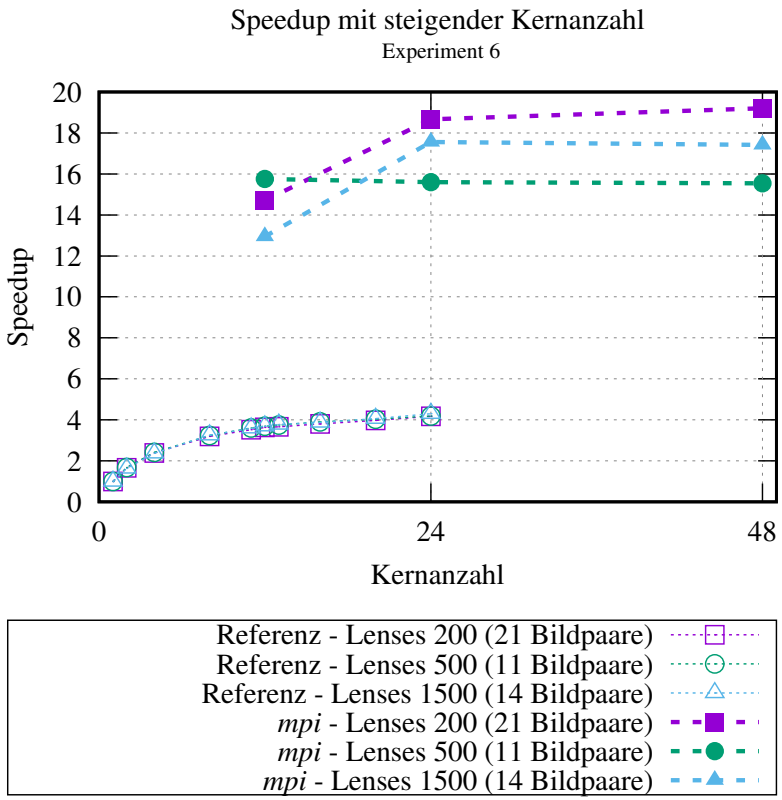



zusätzlich übersetzt

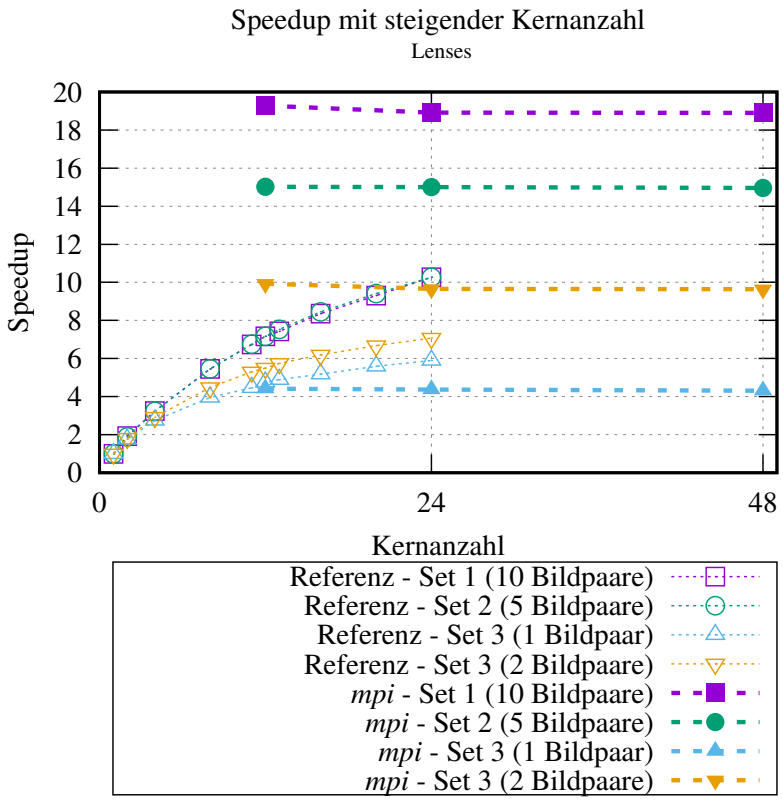
typisiert

# Gliederung

# Parallele Verarbeitung einzelner Bildpaare I



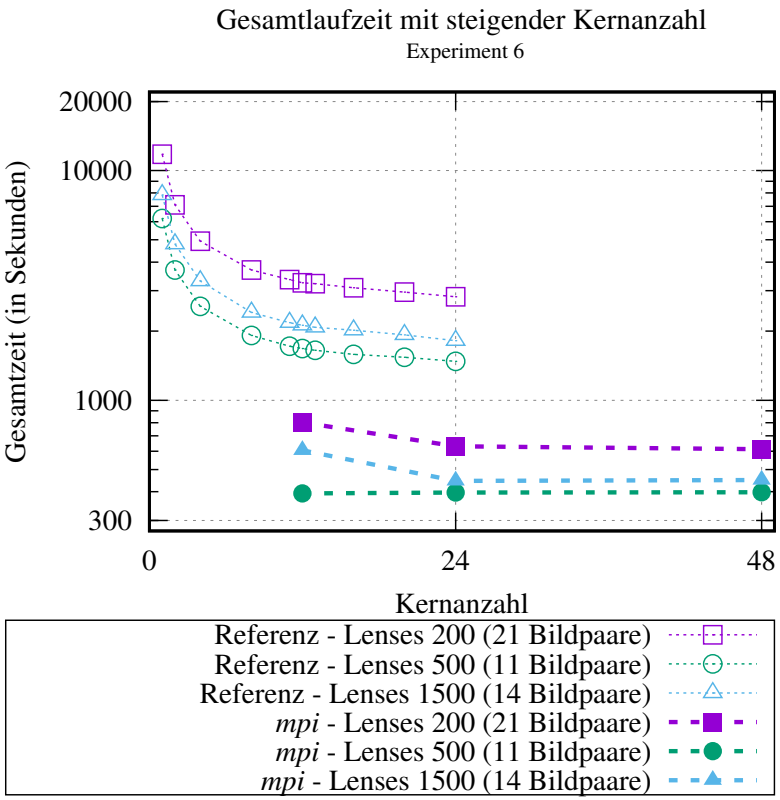

(a) Experiment 6



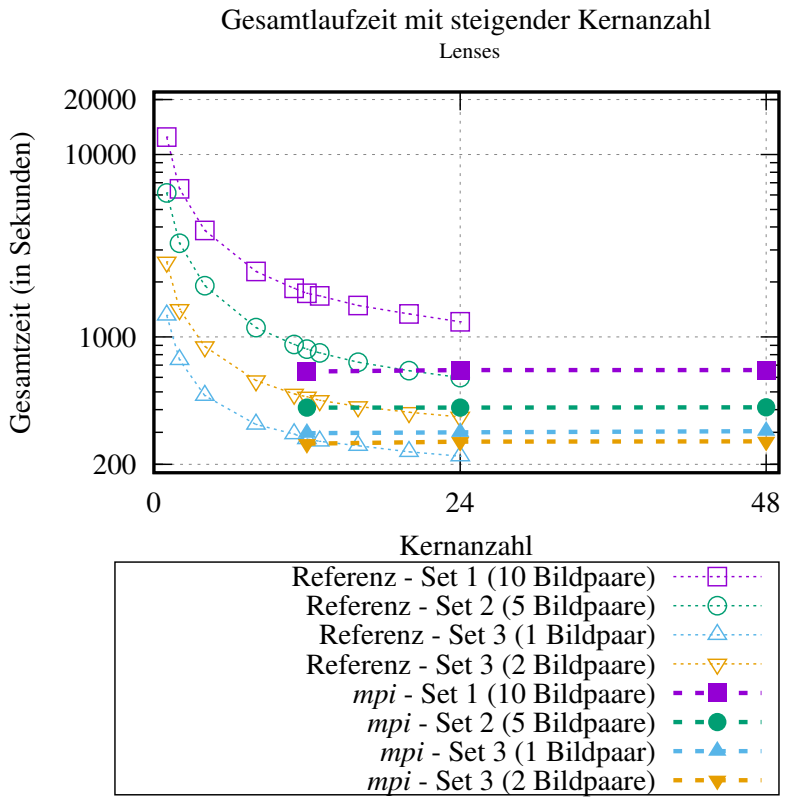

(b) Lenses

Abbildung: Speed-Up *mpi* gegenüber Referenz

# Parallele Verarbeitung einzelner Bildpaare II



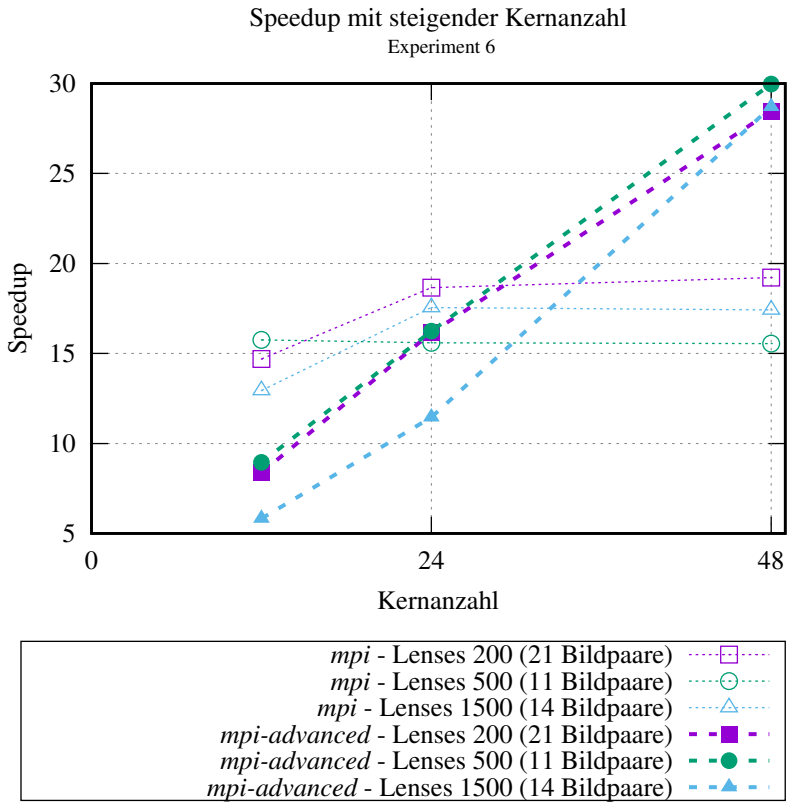

(a) Experiment 6



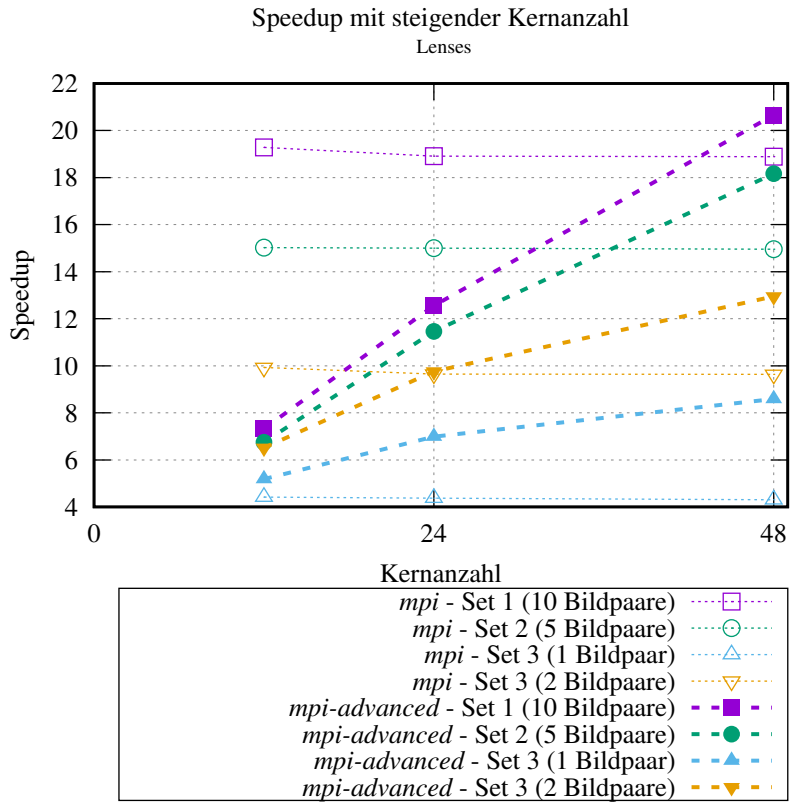

(b) Lenses

Abbildung: Speed-Up *mpi* gegenüber Referenz

# Parallele Verarbeitung innerhalb einzelner Bildpaare I



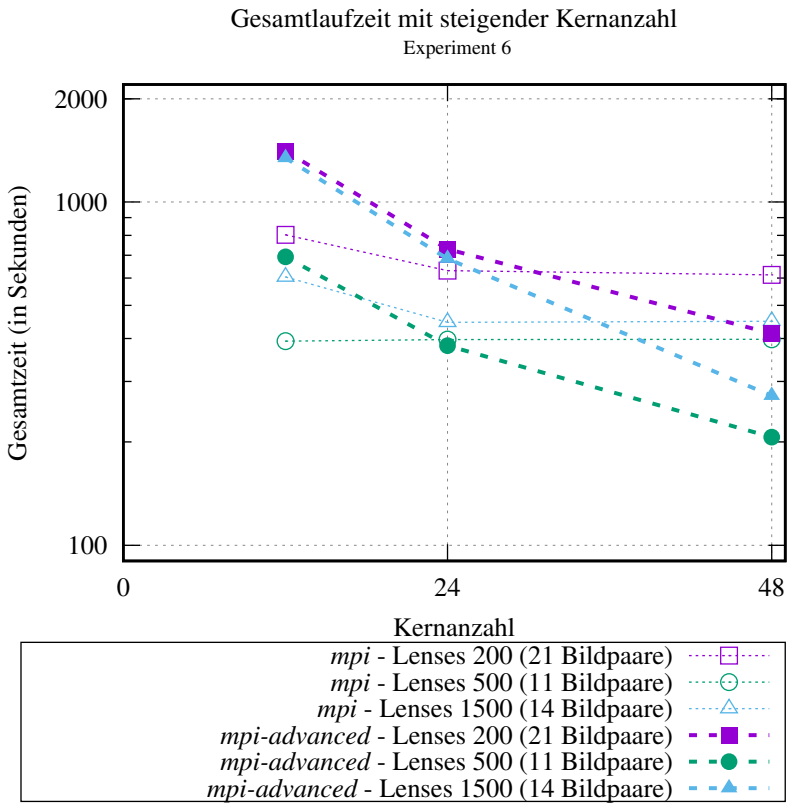

(a) Experiment 6



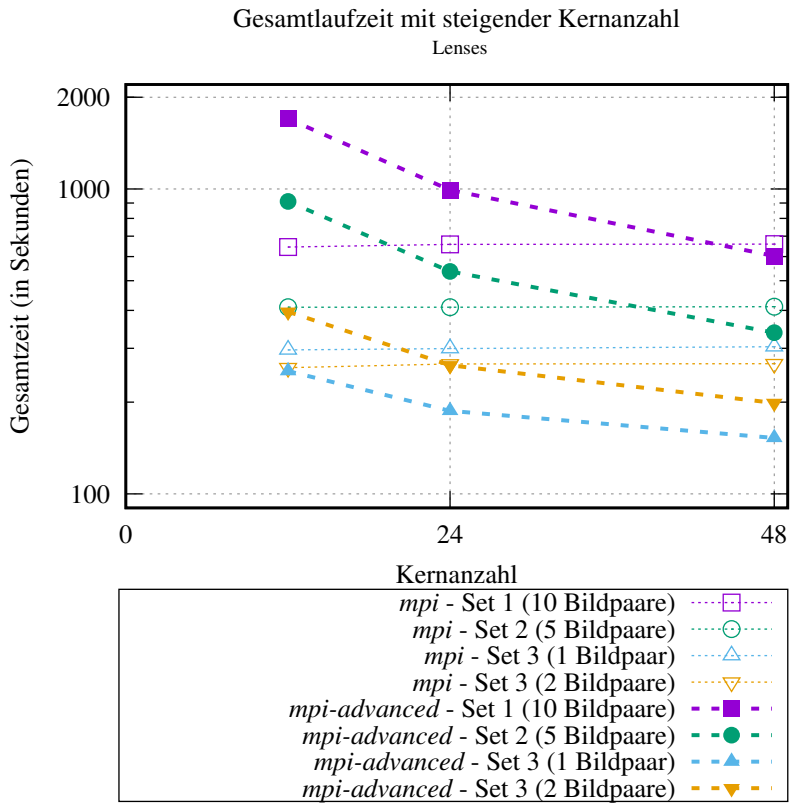

(b) Lenses

Abbildung: Speed-Up *mpi-advanced* gegenüber *mpi*

# Parallele Verarbeitung innerhalb einzelner Bildpaare II



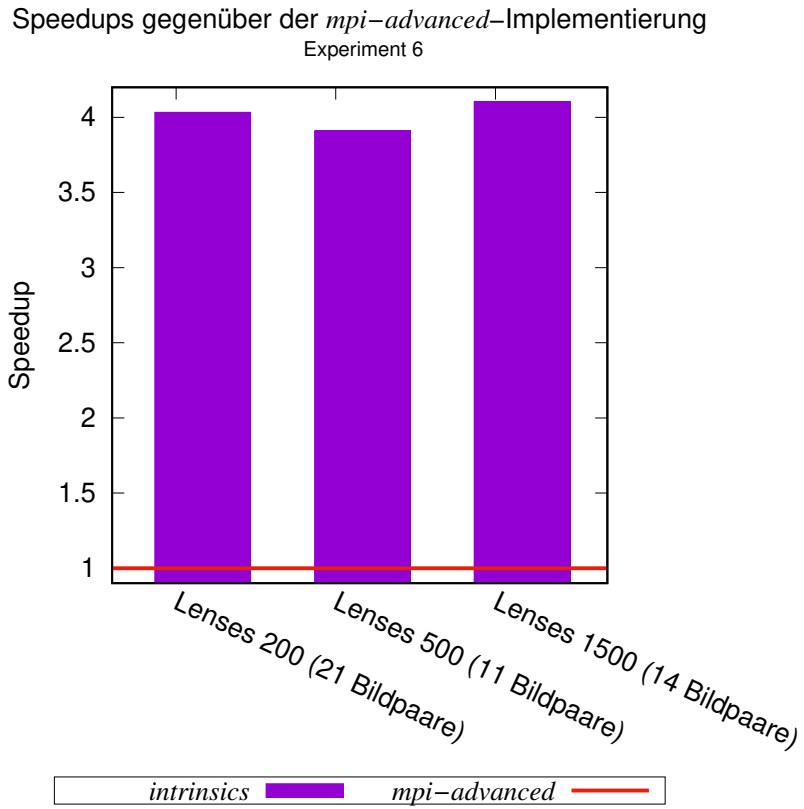

(a) Experiment 6



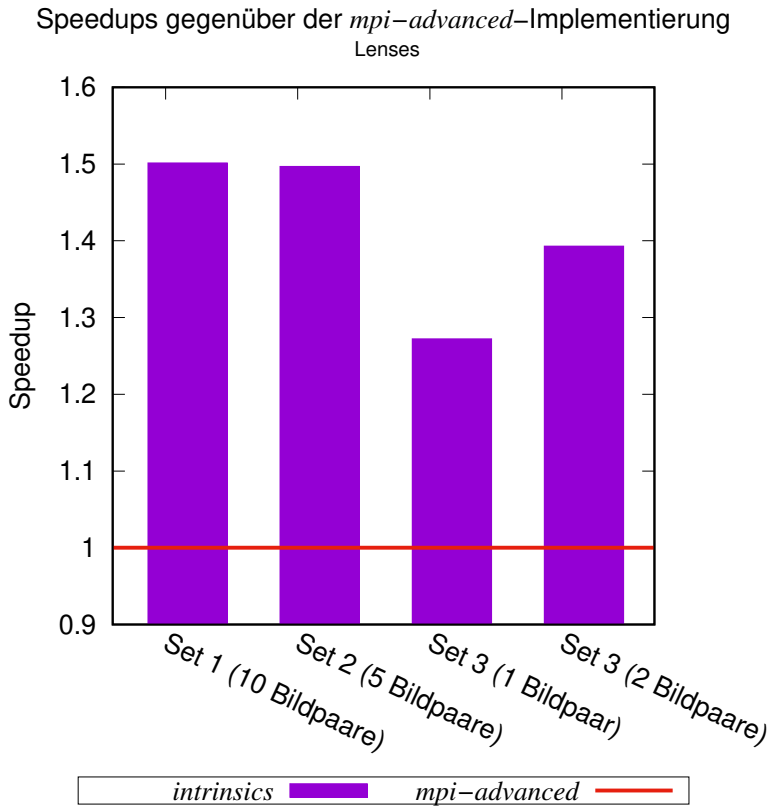

(b) Lenses

Abbildung: Speed-Up *mpi-advanced* gegenüber *mpi*

# Nutzen optimierter Bibliotheken



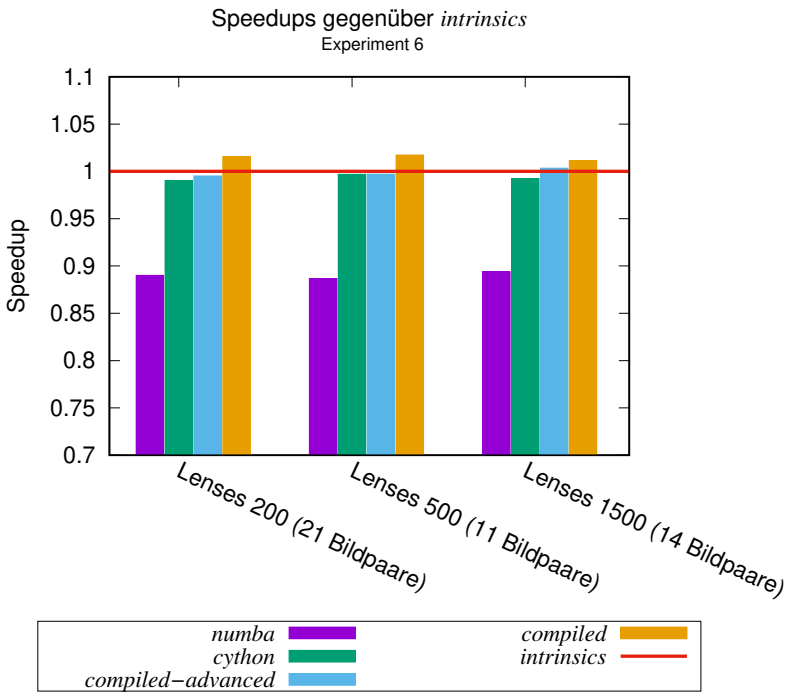

(a) Experiment 6



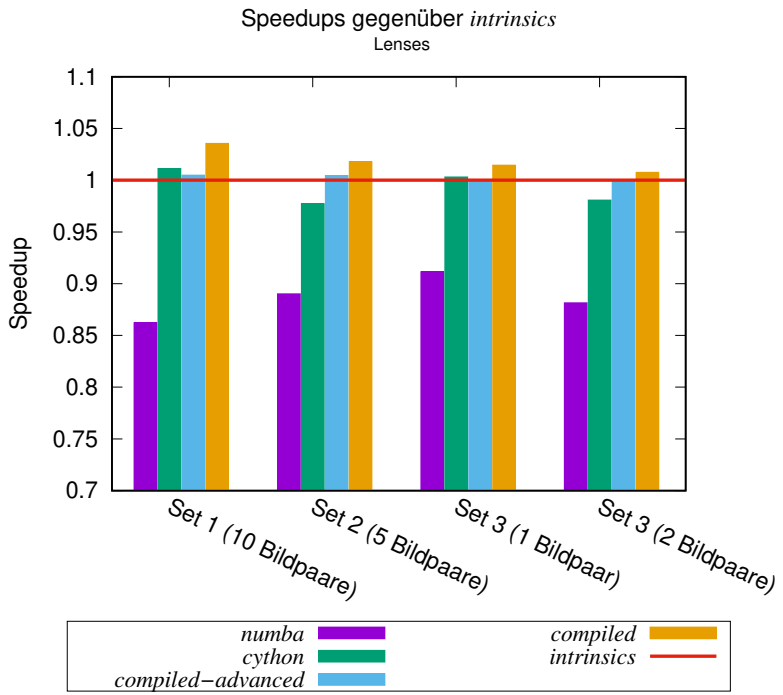

(b) Lenses

Abbildung: Speed-Up *intrinsics* gegenüber *mpi-advanced* (mit zwölf Kernen)

# Kompilieren



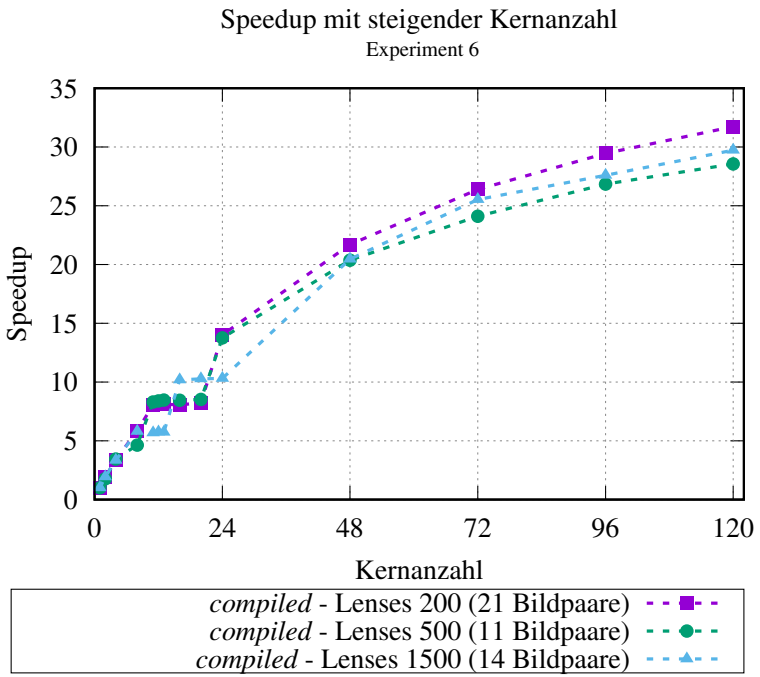

(a) Experiment 6



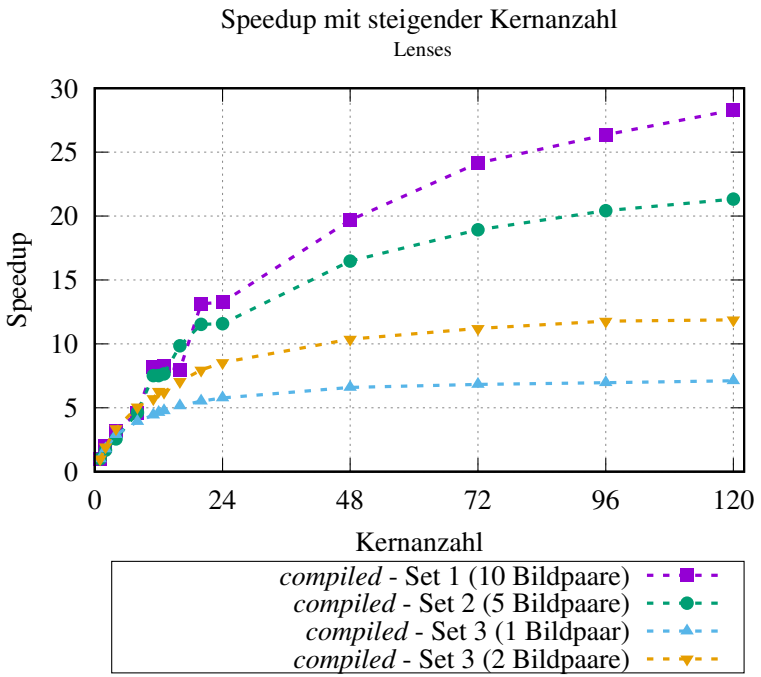

(b) Lenses

Abbildung: Speed-Ups gegenüber *intrinsics* (mit zwölf Kernen)

# Skalierung



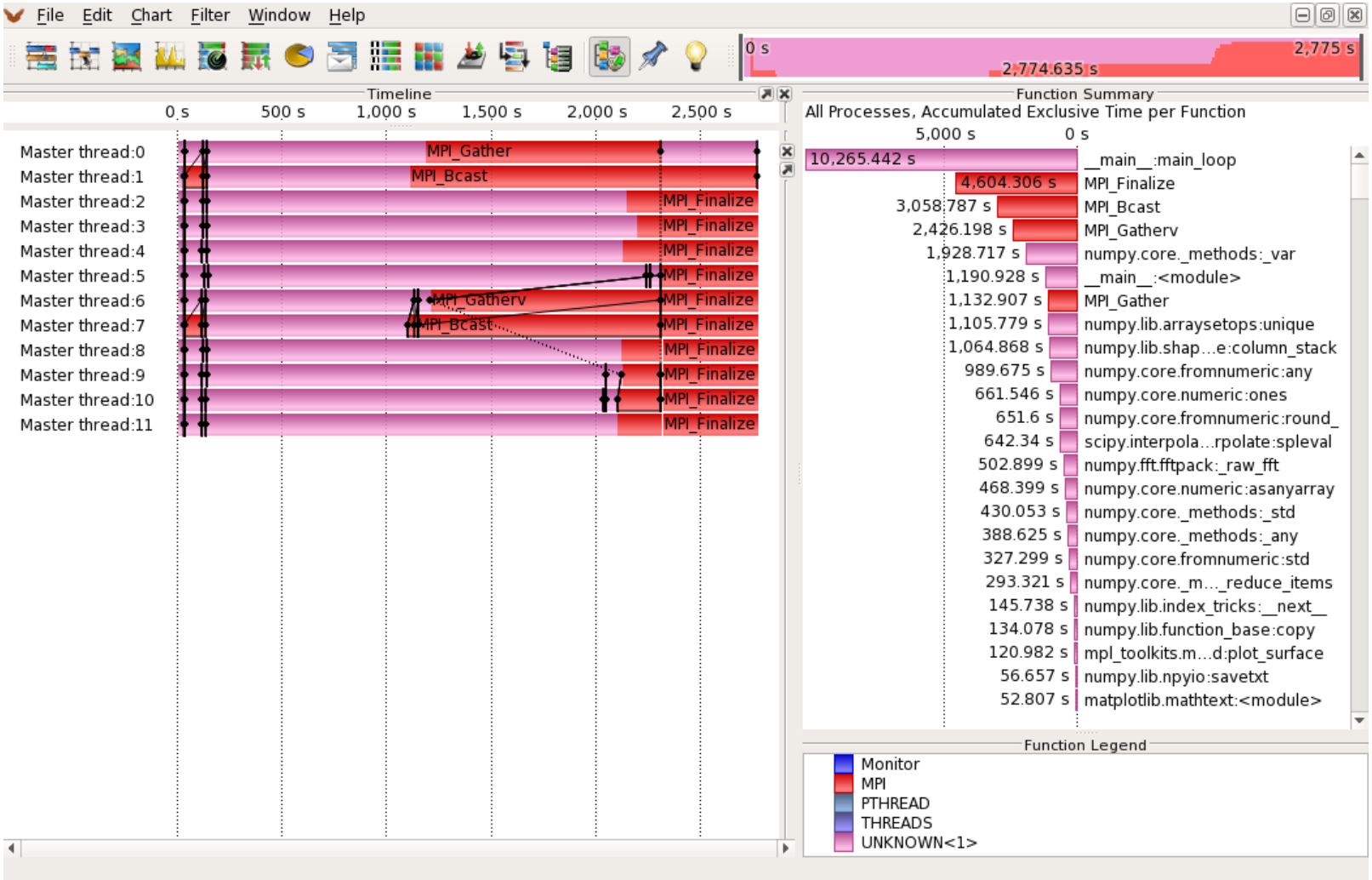

(a) Experiment 6




(b) Lenses

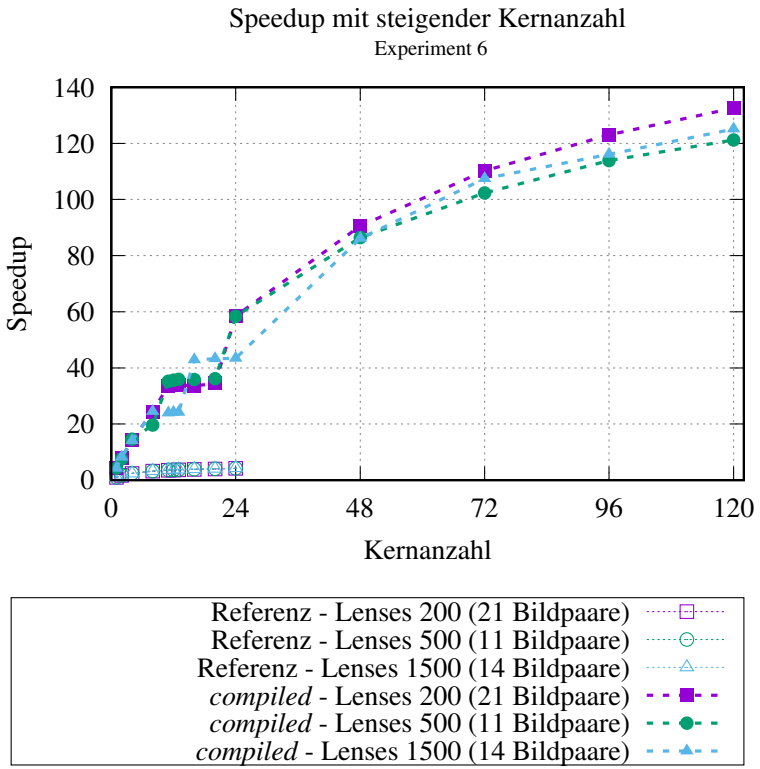
Abbildung: Speed-Up der *compiled* Implementierung

# Skalierung

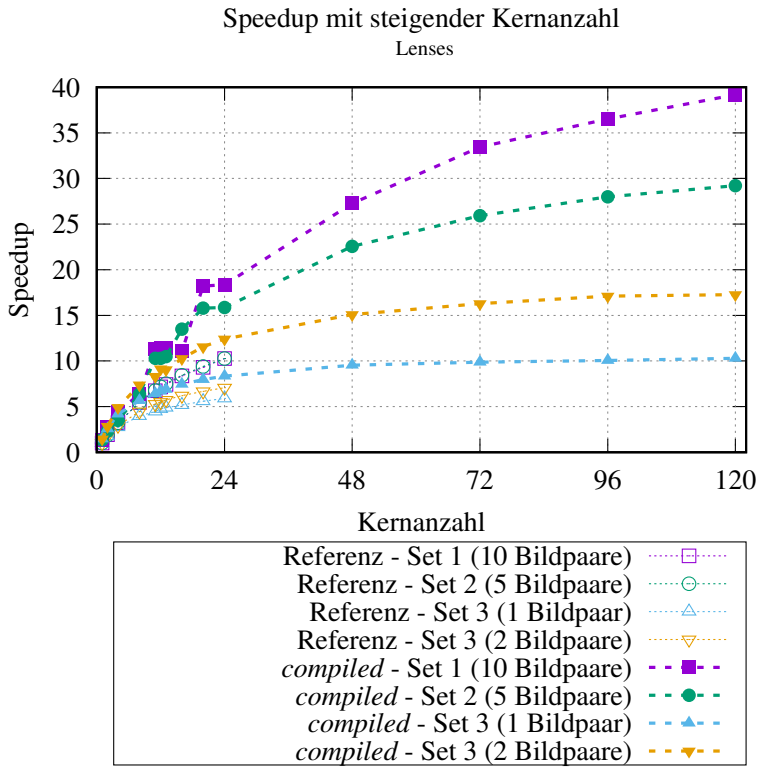

File Edit Chart Filter Window Help
0 s
2,775 s
2,774.635 s
Timeline
0 s
500 s
1,000 s
1,500 s
2,000 s
2,500 s
Master thread:0
Master thread:1
Master thread:2
Master thread:3
Master thread:4
Master thread:5
Master thread:6
Master thread:7
Master thread:8
Master thread:9
Master thread:10
Master thread:11
MPI_Gather
MPI_Bcast
MPI_Finalize
MPI_Gatherv
Function Summary
All Processes, Accumulated Exclusive Time per Function
5,000 s
0 s
10,265.442 s __main__:main_loop
4,604.306 s MPI_Finalize
3,058.787 s MPI_Bcast
2,426.198 s MPI_Gatherv
1,928.717 s numpy.core._methods:_var
1,190.928 s __main__:<module>
1,132.907 s MPI_Gather
1,105.779 s numpy.lib.arraysetops:unique
1,064.868 s numpy.lib.shap...e:column_stack
989.675 s numpy.core.fromnumeric:any
661.546 s numpy.core.numeric:ones
651.6 s numpy.core.fromnumeric:round_
642.34 s scipy.interpola...rpolate:spleval
502.899 s numpy.fft.fftpack:_raw_fft
468.399 s numpy.core.numeric:asanyarray
430.053 s numpy.core._methods:_std
388.625 s numpy.core._methods:_any
327.299 s numpy.core.fromnumeric:std
293.321 s numpy.core._m..._reduce_items
145.738 s numpy.lib.index_tricks:__next__
134.078 s numpy.lib.function_base:copy
120.982 s mpl_toolkits.m...d:plot_surface
56.657 s numpy.lib.npyio:savetxt
52.807 s matplotlib.mathtext:<module>
Function Legend
Monitor
MPI
PTHREAD
THREADS
UNKNOWN<1>

# Gliederung

# Auswertung




(a) Experiment 6




(b) Lenses

Abbildung: Speed-Up *compiled* gegenüber Referenz

# Auswertung

**Speed-Up:**

- Lenses Set 1: bis zu 40
- Experiment 6 Lenses 200: bis zu 130

$\Rightarrow$ Echtzeitfähigkeit mit $< 100$ Taurus-Knoten nicht möglich

# Verbesserungsmöglichkeiten

- Nutzen von FFTW
- Nutzen von GPGPUs
- Implementieren eines Belastungsausgleich
- Algorithmische Verbesserungen

# Weiterführende Literatur I


- Bérujon, Sébastien (2013). "Métrologie en ligne de faisceaux et d'optiques X de synchrotrons". Thèse de doctorat dirigée par Ziegler, Eric et Sawhney, Kawal Physique Grenoble 2013. Diss. Université de Grenoble. URL: http://www.theses.fr/2013GRENY010.
- Bérujon, Sébastien, Eric Ziegler, Roberto Cerbino u. a. (Apr. 2012). "Two-Dimensional X-Ray Beam Phase Sensing". In: *Phys. Rev. Lett.* 108 (15), S. 158102. DOI: 10.1103/PhysRevLett.108.158102. URL: https://link.aps.org/doi/10.1103/PhysRevLett.108.158102.
- Bérujon, Sébastien, Eric Ziegler und Peter Cloetens (2015). "X-ray pulse wavefront metrology using speckle tracking". In: *Journal of Synchrotron Radiation* 22.4, S. 886–894. ISSN: 1600-5775. DOI: 10.1107/S1600577515005433. URL: http://dx.doi.org/10.1107/S1600577515005433.

# Weiterführende Literatur II



Cojocaru, Elena-Ruxandra und Sébastien Bérujon (2017). *Wavefront-Sensor*. (Privat). URL: https://github.com/ComputationalRadiationPhysics/Wavefront-Sensor/tree/6f06a83babb66902fb83b7dfcdb9b0cf9f18acc0.


Frankot, R. T. und R. Chellappa (Juli 1988). "A method for enforcing integrability in shape from shading algorithms". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10.4, S. 439–451. ISSN: 0162-8828. DOI: 10.1109/34.3909.