



# 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](mailto:jonas.schenke@tu-dresden.de)

- 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

- 1 Hinweise
- 2 Evaluierung des Wellenfrontrekonstruktionsalgorithmus
- 3 Performance-Analyse der vorgegebenen Python-Implementierung
- 4 Parallelisierung der kritischen Abschnitte
- 5 Performance-Messungen der parallelen Implementation
- 6 Auswertung

- 1 Hinweise
- 2 Evaluierung des Wellenfrontrekonstruktionsalgorithmus
- 3 Performance-Analyse der vorgegebenen Python-Implementierung
- 4 Parallelisierung der kritischen Abschnitte
- 5 Performance-Messungen der parallelen Implementation
- 6 Auswertung

- Teil des **European Cluster of Advanced Laser Light Sources** (EUCALL)-Projektes → Überschneidung von **Ultrafast Data Acquisition** (UFDAC; WP5) und **Pulse Characterisation and Control** (PUCCA; WP7)
- Zusammenarbeit des **Helmholtz-Zentrum Dresden-Rossendorf e.V.** (HZDR) und **European Synchrotron Radiation Facility** (ESRF)
- **Algorithmus:** Sébastien Béruron
- **Code:** Elena-Ruxandra Cojocaru

- Teil des **European Cluster of Advanced Laser Light Sources** (EUCALL)-Projektes → Überschneidung von **Ultrafast Data Acquisition** (UFDAC; WP5) und **Pulse Characterisation and Control** (PUCCA; WP7)
- Zusammenarbeit des **Helmholtz-Zentrum Dresden-Rossendorf e.V.** (HZDR) und **European Synchrotron Radiation Facility** (ESRF)
- **Algorithmus:** Sébastien Béruron
- **Code:** Elena-Ruxandra Cojocaru

- Teil des **E**uropean **C**luster of **A**dvanced **L**aser **L**ight Sources (EUCALL)-Projektes → Überschneidung von **U**ltrafast **D**ata **A**cquisition (UFDAC; WP5) und **P**ulse **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**acility (ESRF)
- **Algorithmus:** Sébastien Béruron
- **Code:** Elena-Ruxandra Cojocaru

- Teil des **E**uropean **C**luster of **A**dvanced **L**aser **L**ight Sources (EUCALL)-Projektes → Überschneidung von **U**ltrafast **D**ata **A**cquisition (UFDAC; WP5) und **P**ulse **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**acility (ESRF)
- **Algorithmus:** Sébastien Béruron
- **Code:** Elena-Ruxandra Cojocaru



- Teil des **E**uropean **C**luster of **A**dvanced **L**aser **L**ight Sources (EUCALL)-Projektes → Überschneidung von **U**ltrafast **D**ata **A**cquisition (UFDAC; WP5) und **P**ulse **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**acility (ESRF)
- **Algorithmus:** Sébastien Béruron
- **Code:** Elena-Ruxandra Cojocaru

**Daten:** Beamline BM05, ESRF, Grenoble, Frankreich

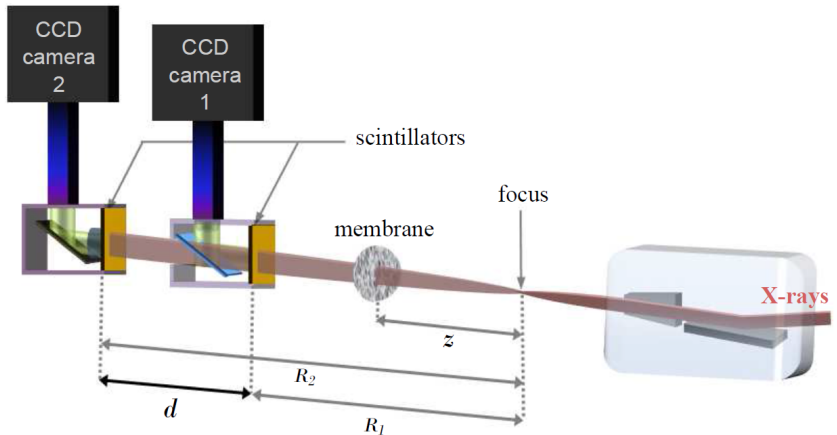
---

	<b>Experiment 6</b>	<b>Lenses</b>
Aufnahmedatum	Ruxandra Cojocaru, Sébastien Bérupon, Eric Ziegler	Thomas Rothand, Raymond Barrett, Sébastien Bérupon, Rafael Celeste
Aufgenommen von	24. September 2017	10. April 2017

---

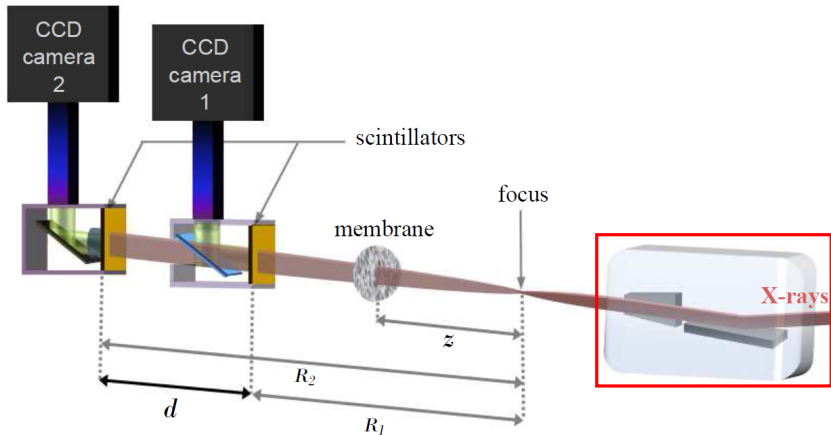
- 1 Hinweise
- 2 Evaluierung des Wellenfrontrekonstruktionsalgorithmus
- 3 Performance-Analyse der vorgegebenen Python-Implementierung
- 4 Parallelisierung der kritischen Abschnitte
- 5 Performance-Messungen der parallelen Implementation
- 6 Auswertung

# Versuchsaufbau



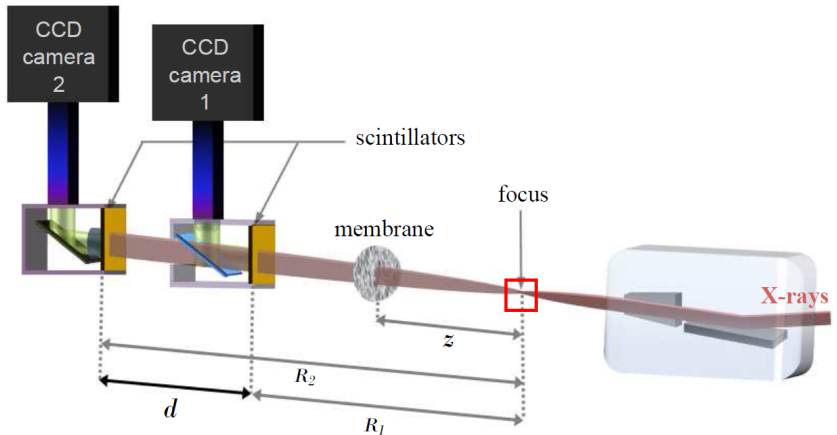
"X-ray pulse wavefront metrology using speckle tracking", Bérújon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

# Versuchsaufbau



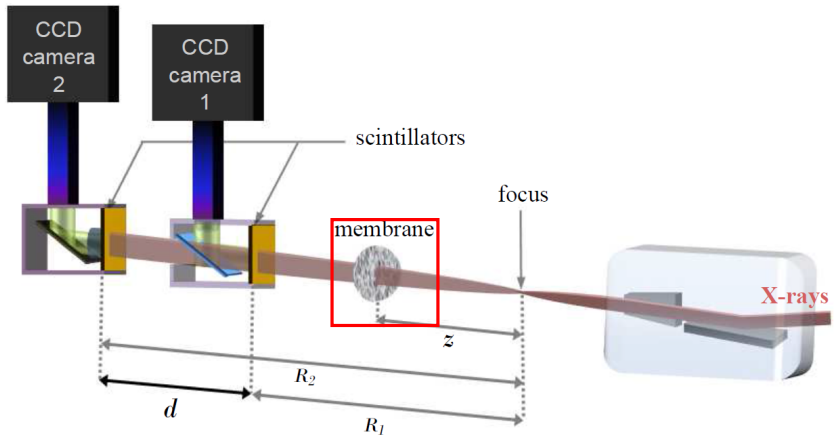
"X-ray pulse wavefront metrology using speckle tracking", Bérújon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

# Versuchsaufbau



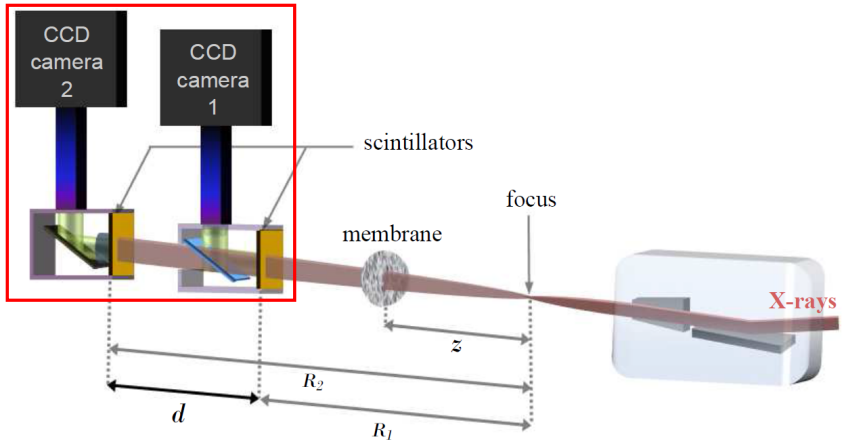
"X-ray pulse wavefront metrology using speckle tracking", Bérújon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

# Versuchsaufbau



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

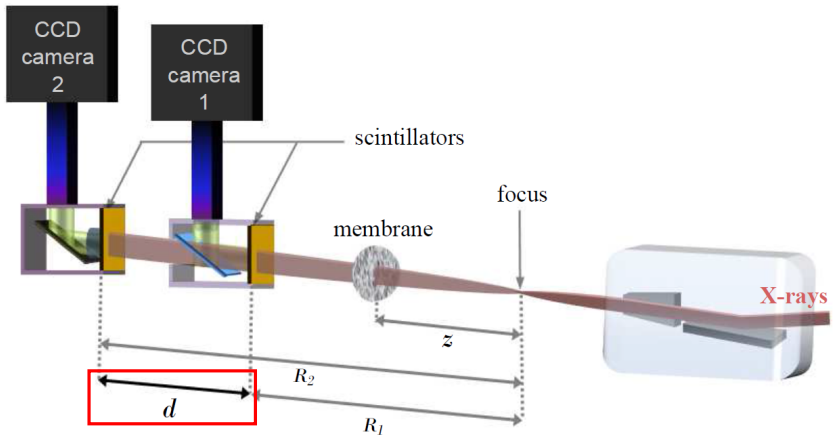
# Versuchsaufbau



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

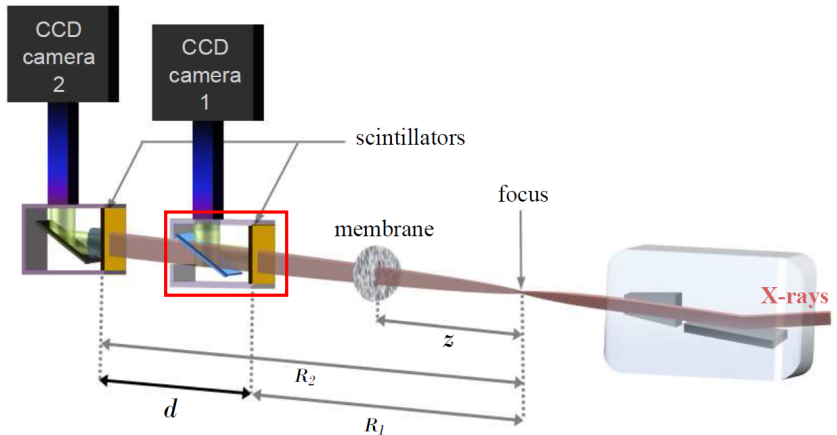


# Versuchsaufbau



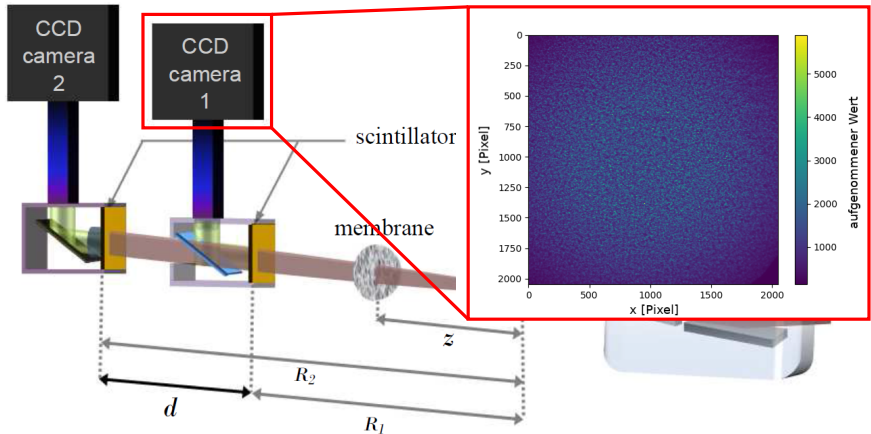
"X-ray pulse wavefront metrology using speckle tracking", Bérújon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

# Versuchsaufbau



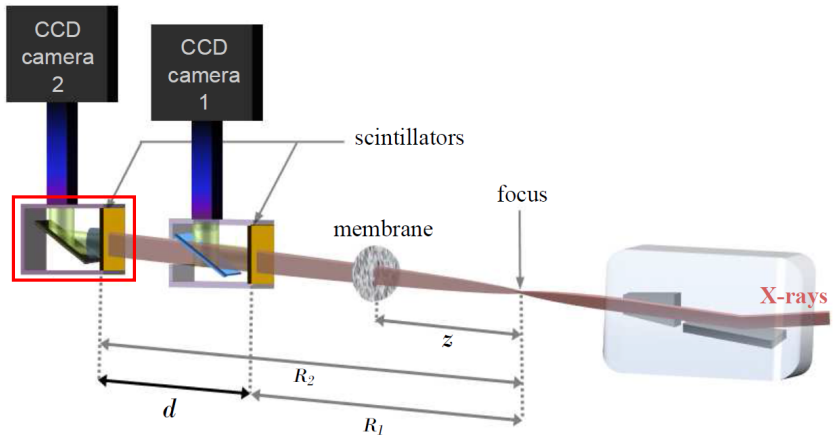
"X-ray pulse wavefront metrology using speckle tracking", Bérújon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

# Versuchsaufbau



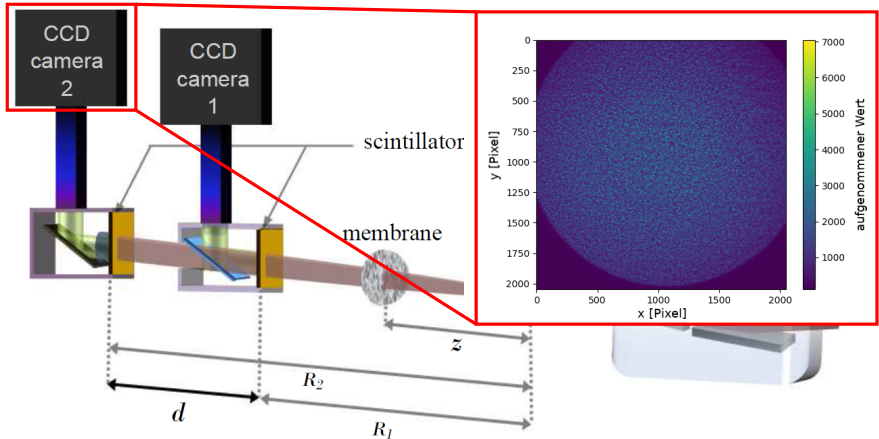
"X-ray pulse wavefront metrology using speckle tracking", Bérújon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

# Versuchsaufbau



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

# Versuchsaufbau

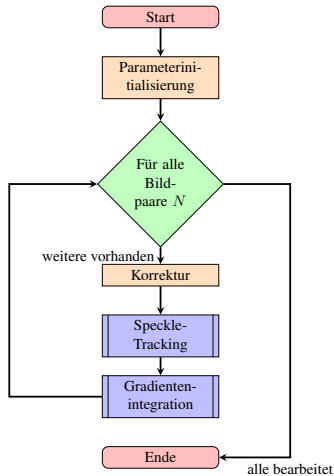


"X-ray pulse wavefront metrology using speckle tracking", Bérújon, Ziegler und Cloetens 2015, *Journal of Synchrotron Radiation*

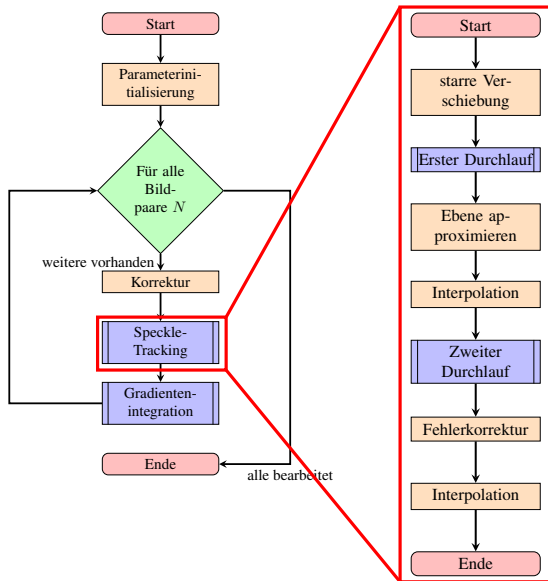
## Hauptroutine:

- Korrigieren von Kamerafehlern
- Ablenkung nachverfolgen
- Wellenfront rekonstruieren

# Hauptroutine

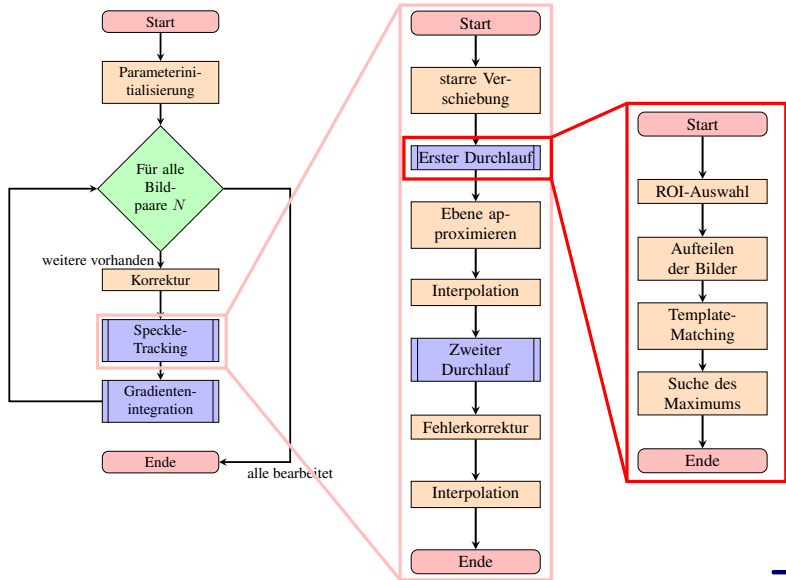


# Hauptroutine

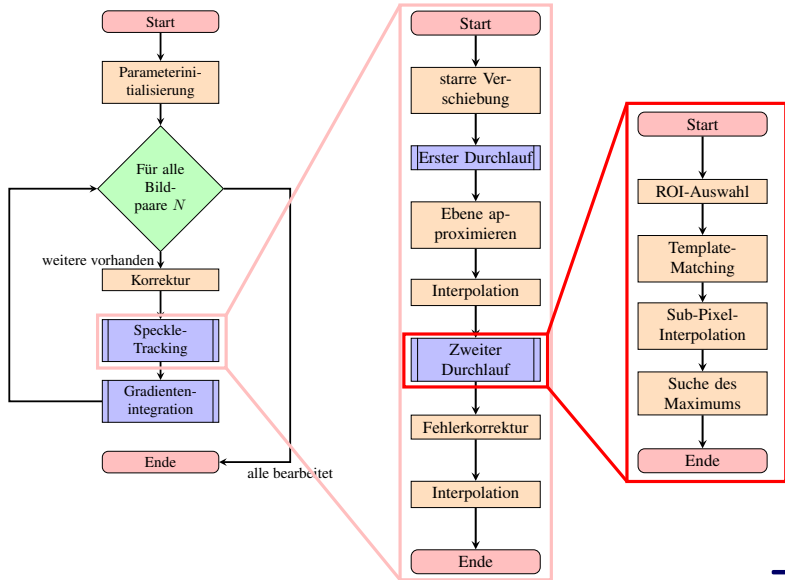




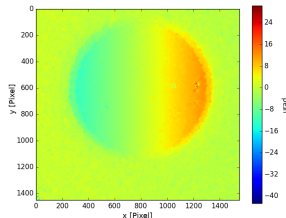
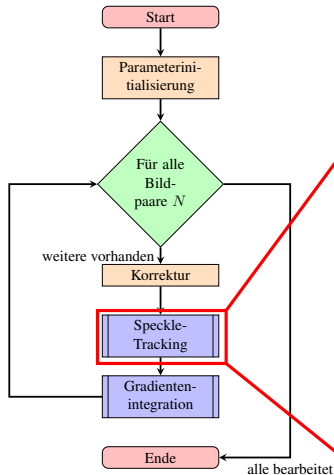
# Hauptroutine



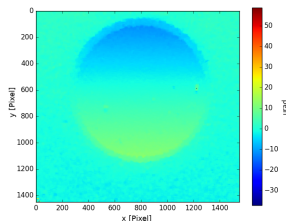
# Hauptroutine



# Hauptroutine

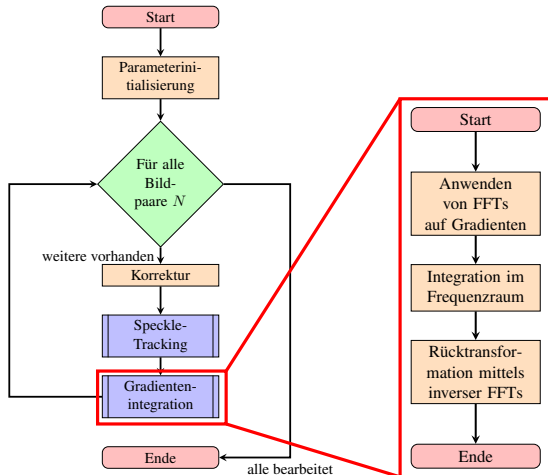


Horizontaler Gradient



Vertikaler Gradient

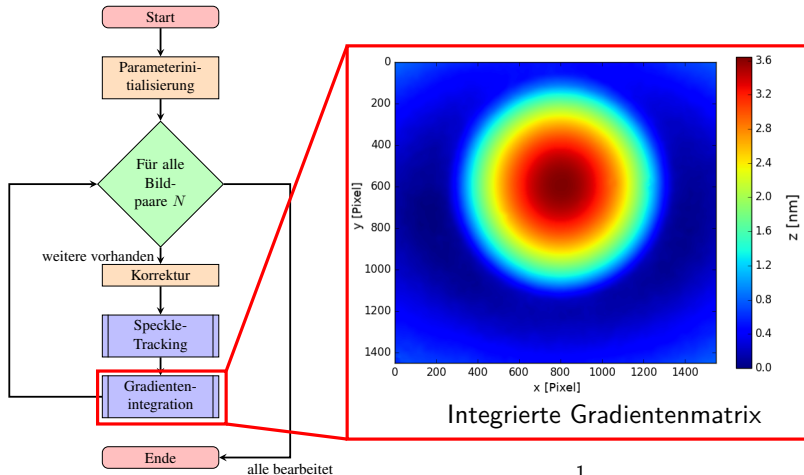
# Hauptroutine



1

<sup>1</sup>“A method for enforcing integrability in shape from shading algorithms”, Frankot und Chellappa 1988, *IEEE Transactions on Pattern Analysis and Machine Intelligence*

# Hauptroutine



1

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

- 1 Hinweise
- 2 Evaluierung des Wellenfrontrekonstruktionsalgorithmus
- 3 Performance-Analyse der vorgegebenen Python-Implementierung**
- 4 Parallelisierung der kritischen Abschnitte
- 5 Performance-Messungen der parallelen Implementation
- 6 Auswertung

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

## Hauptroutine:

Bild-Präprozessierschritte	$\mathcal{O}(R)$
Speckle-Tracking	$\mathcal{O}\left(\frac{R \cdot R_{corr} \cdot \log(R_{corr}) \cdot N_{corr}}{P_u^2 \cdot R_{grid}^2}\right) \approx$ $\mathcal{O}(R \cdot R_{corr} \cdot \log(R_{corr}))$
Rekonstruktion der Wellenfront	$\mathcal{O}(R * \log(R))$

⇒ **Gesamtkomplexität:**  $\approx \mathcal{O}(n \cdot R \cdot R_{corr} \cdot \log(R_{corr}))$

## Hauptroutine:

- Experiment 6
  - ROI Größe: 1450x1450 (Bild), 550x550 (Template)
  - Gitterauflösung: 1
  - Korrelationsgröße: 91
  - unterschiedliche Pixelgröße
- Lenses
  - ROI Größe: 1450x1550 (Bild), 1450x1550 (Template)
  - Gitterauflösung: 1
  - Korrelationsgröße: 41
  - gleiche Pixelgröße



## Hauptroutine:

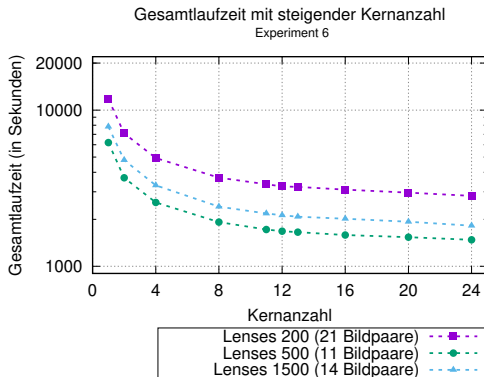
- Experiment 6
  - ROI Größe: 1450x1450 (Bild), 550x550 (Template)
  - Gitterauflösung: 1
  - Korrelationsgröße: 91
  - unterschiedliche Pixelgröße
- Lenses
  - ROI Größe: 1450x1550 (Bild), 1450x1550 (Template)
  - Gitterauflösung: 1
  - Korrelationsgröße: 41
  - gleiche Pixelgröße

## Hauptroutine:

- Experiment 6
  - ROI Größe: 1450x1450 (Bild), 550x550 (Template)
  - Gitterauflösung: 1
  - Korrelationsgröße: 91
  - unterschiedliche Pixelgröße
- Lenses
  - ROI Größe: 1450x1550 (Bild), 1450x1550 (Template)
  - Gitterauflösung: 1
  - Korrelationsgröße: 41
  - gleiche Pixelgröße

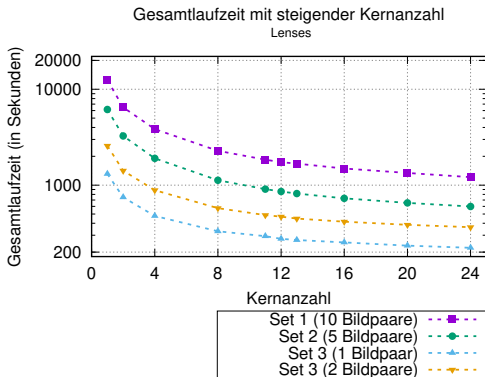
## Hauptroutine:

- Experiment 6
  - ROI Größe: 1450x1450 (Bild), 550x550 (Template)
  - Gitterauflösung: 1
  - Korrelationsgröße: 91
  - unterschiedliche Pixelgröße
- Lenses
  - ROI Größe: 1450x1550 (Bild), 1450x1550 (Template)
  - Gitterauflösung: 1
  - Korrelationsgröße: 41
  - gleiche Pixelgröße



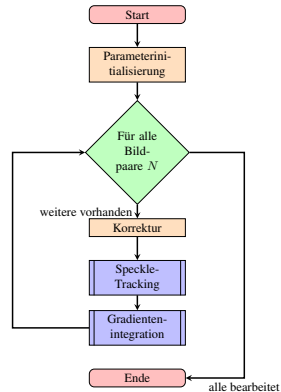
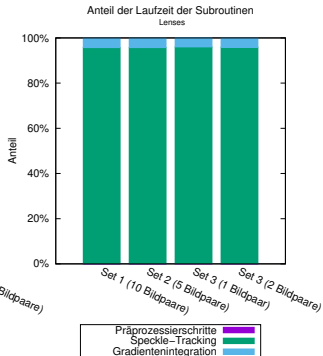
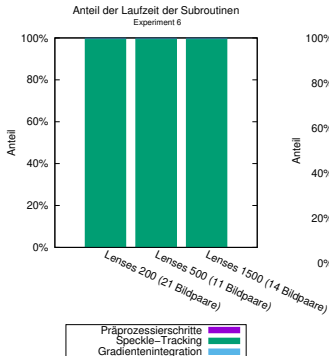
Python 2.7, 2x Intel(R) Xeon(R) E5-2680 v3 (12 Kerne) @ 2.50GHz, kein MultiThreading

# Gesamtlaufzeit II



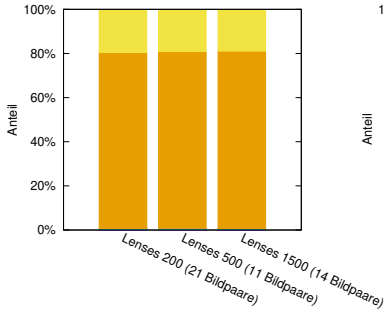
Python 2.7, 2x Intel(R) Xeon(R) E5-2680 v3 (12 Kerne) @ 2.50GHz, kein MultiThreading

# Profiling I

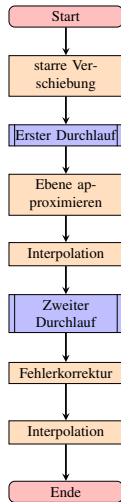
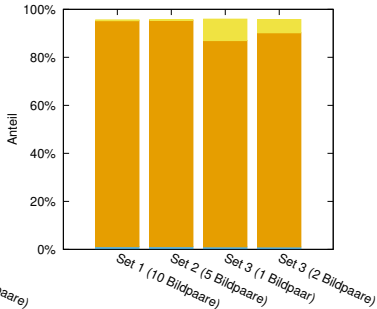


# Profiling II

Anteil der Laufzeit der Speckle-Tracking-Subroutinen  
Experiment 6



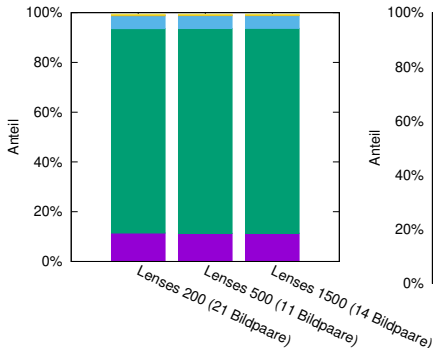
Anteil der Laufzeit der Speckle-Tracking-Subroutinen  
Lenses



# Profiling III

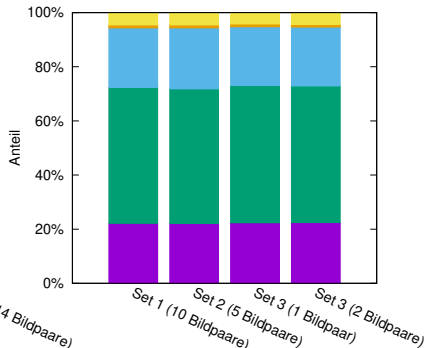
Anteil der Laufzeit der rechenaufwendigsten Funktionen

Experiment 6



Anteil der Laufzeit der rechenaufwendigsten Funktionen

Lenses

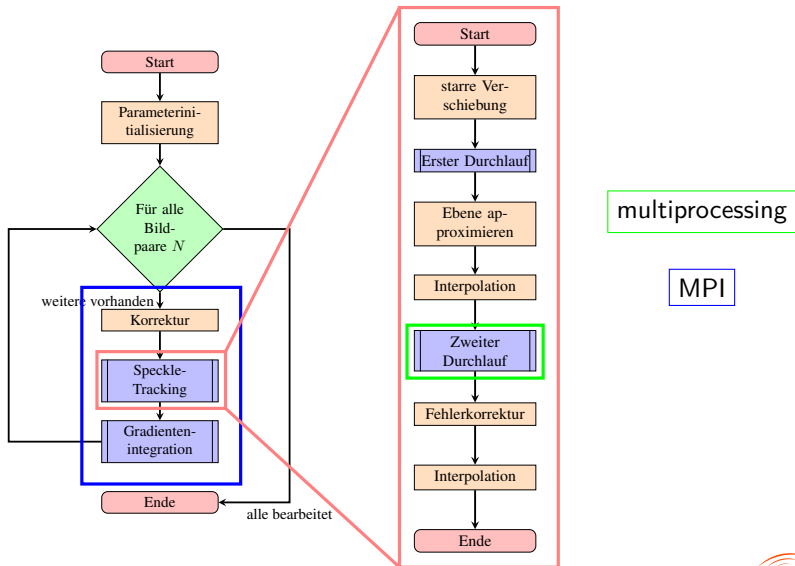


⇒ über 95%



- 1 Hinweise
- 2 Evaluierung des Wellenfrontrekonstruktionsalgorithmus
- 3 Performance-Analyse der vorgegebenen Python-Implementierung
- 4 Parallelisierung der kritischen Abschnitte**
- 5 Performance-Messungen der parallelen Implementation
- 6 Auswertung

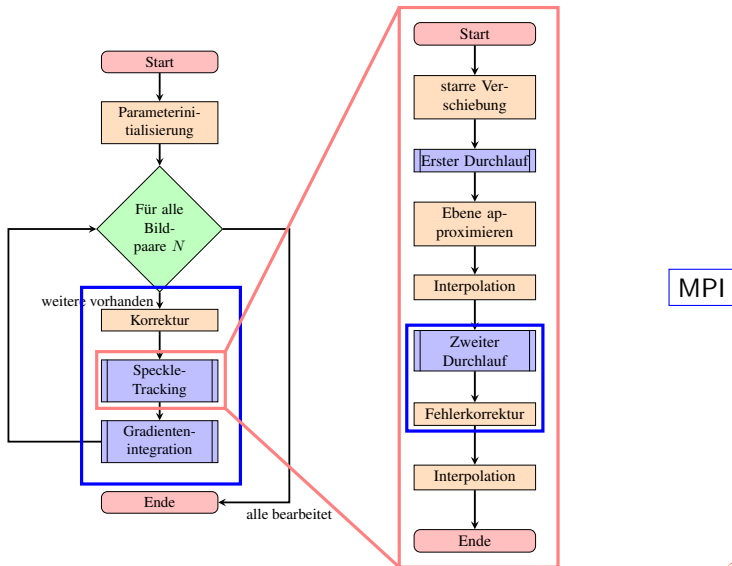
# Parallele Verarbeitung einzelner Bildpaare



## 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



# Parallele Verarbeitung innerhalb einzelner Bildpaare I

**Benennung:** *mpi-advanced*

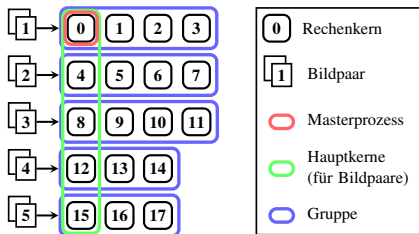


Abbildung: Verteilung von fünf Bildpaaren auf 18 Rechenknoten

## 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)
```

## 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
15                       k in range(n_jobs))
16 print("The result is", result)
```

# Parallele Verarbeitung innerhalb einzelner Bildpaare IV

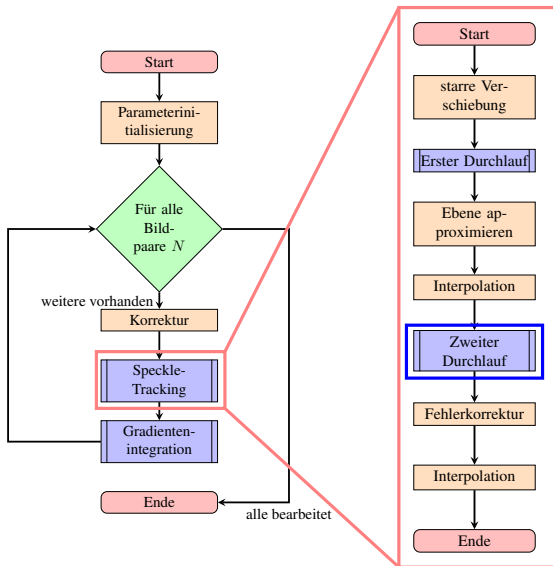
---

## Ausgabe:

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



# Nutzen bereits optimierter Funktionen



**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
```

## 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
```

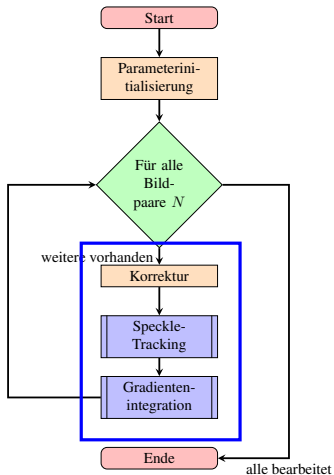
## Berechnung des Signal-Rausch-Verhältnisses:

```
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
```

**Benennung:** *compiled*

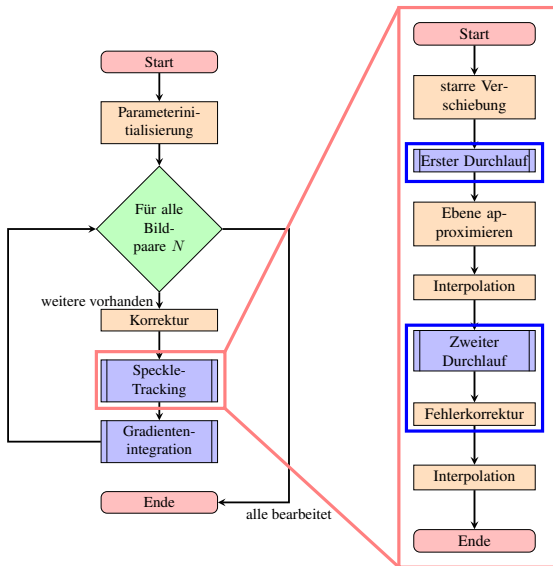
- Übersetzen großer Teile des Python-Codes in C mittels Cython
- Übersetzen des C-Codes in nativen Bytecode

# Kompilieren - Gesamtes Programm



## Benennung: *numba*

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





## **Benennung:** *cython*

- Typisierung der rechenaufwendigsten Funktionen
- Übersetzen dieser Funktionen

## **Benennung:** *compiled-advanced*

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

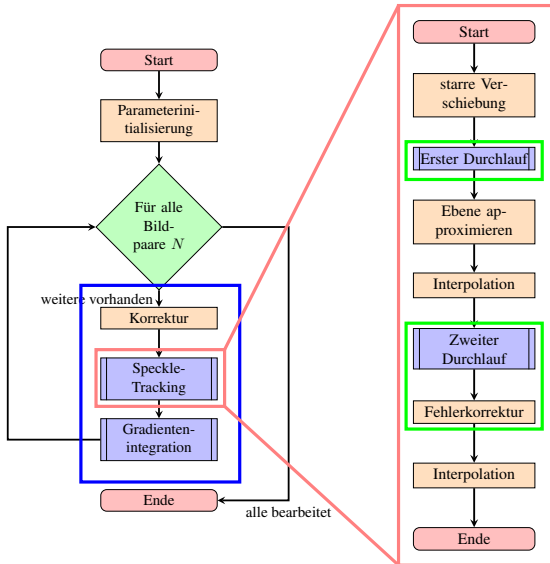
## **Benennung:** *cython*

- Typisierung der rechenaufwendigsten Funktionen
- Übersetzen dieser Funktionen

## **Benennung:** *compiled-advanced*

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

# Kompilieren - Cython

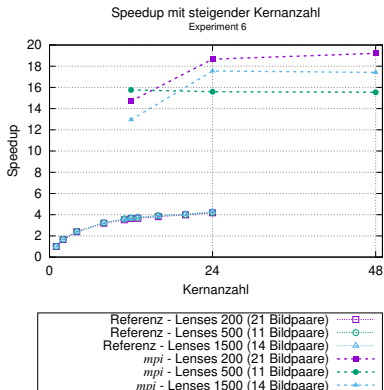


zusätzlich übersetzt

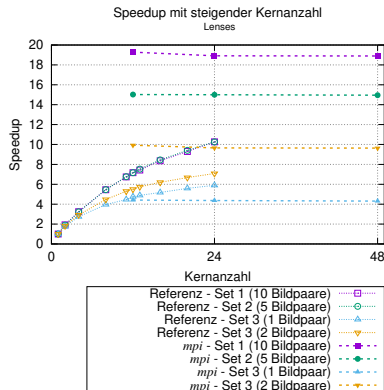
typisiert

- 1 Hinweise
- 2 Evaluierung des Wellenfrontrekonstruktionsalgorithmus
- 3 Performance-Analyse der vorgegebenen Python-Implementierung
- 4 Parallelisierung der kritischen Abschnitte
- 5 Performance-Messungen der parallelen Implementation**
- 6 Auswertung

# Parallele Verarbeitung einzelner Bildpaare I



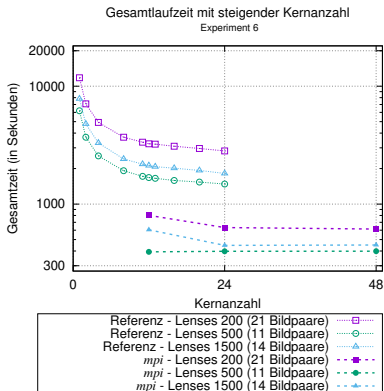
(a) Experiment 6



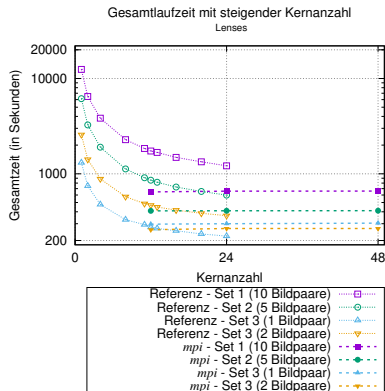
(b) Lenses

Abbildung: Speed-Up *mpi* gegenüber Referenz

# Parallele Verarbeitung einzelner Bildpaare II



(a) Experiment 6

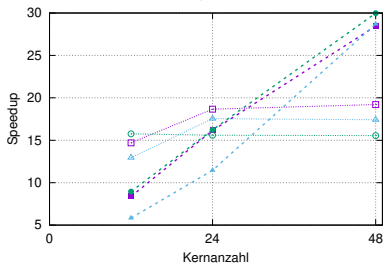


(b) Lenses

Abbildung: Speed-Up *mpi* gegenüber Referenz

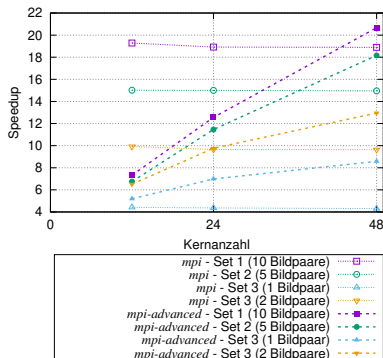
# Parallele Verarbeitung innerhalb einzelner Bildpaare I

Speedup mit steigender Kernanzahl  
Experiment 6



(a) Experiment 6

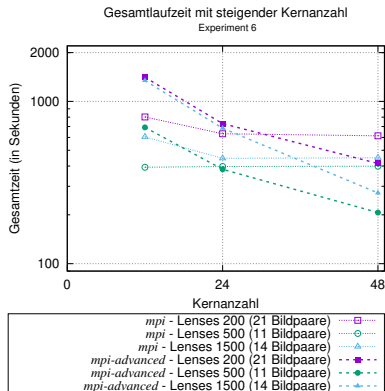
Speedup mit steigender Kernanzahl  
Lenses



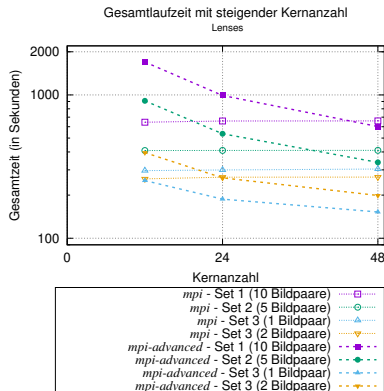
(b) Lenses

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

# Parallele Verarbeitung innerhalb einzelner Bildpaare II



(a) Experiment 6



(b) Lenses

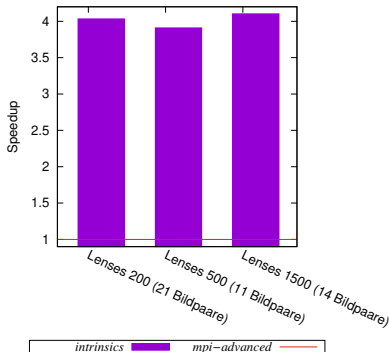
Abbildung: Speed-Up *mpi-advanced* gegenüber *mpi*



# Nutzen optimierter Bibliotheken

Speedups gegenüber der *mpi-advanced*-Implementierung

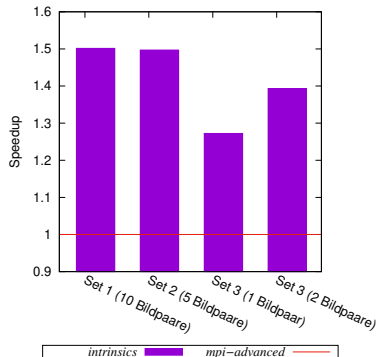
Experiment 6



(a) Experiment 6

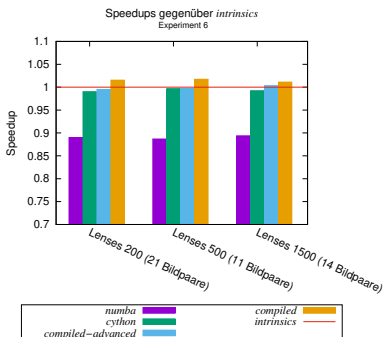
Speedups gegenüber der *mpi-advanced*-Implementierung

Lenses

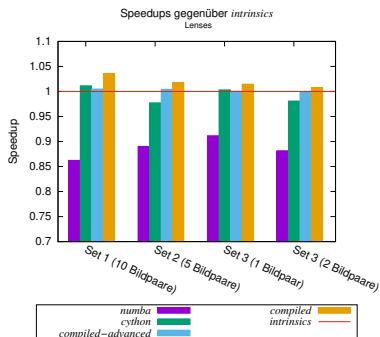


(b) Lenses

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

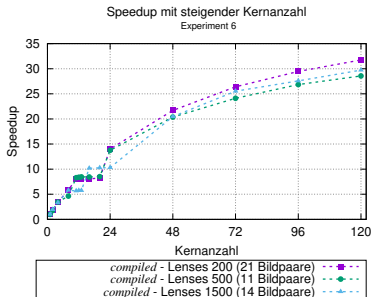


(a) Experiment 6

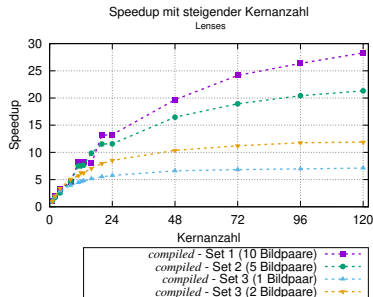


(b) Lenses

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



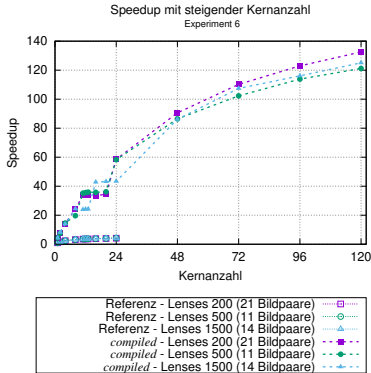
(a) Experiment 6



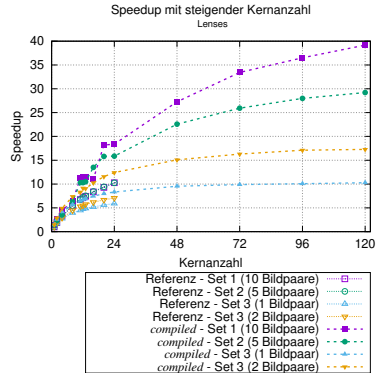
(b) Lenses

Abbildung: Speed-Up der *compiled* Implementierung

- 1 Hinweise
- 2 Evaluierung des Wellenfrontrekonstruktionsalgorithmus
- 3 Performance-Analyse der vorgegebenen Python-Implementierung
- 4 Parallelisierung der kritischen Abschnitte
- 5 Performance-Messungen der parallelen Implementation
- 6 Auswertung**



(a) Experiment 6



(b) Lenses

Abbildung: Speed-Up *compiled* gegenüber Referenz

## Speed-Up:

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

⇒ Echtzeitfähig mit ca. 100 Taurus-Knoten

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



Bérupon, 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>.



Bérupon, 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>.



Bérupon, 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>.





Cojocaru, Elena-Ruxandra und Sébastien Bérupon (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.