I Projektüberblick

- /src/ enthält C++ und CUDA Quelltexte
- /csharp-src/ImageModifier/ beinhaltet C# ImageSharp Quelltext
- implementiert wurden Greyscale, Blur und HSV Operationen auf Bildern
- C++ Implementation enhält außerdem noch Emboss
- run_no_parallism.sh baut den C++ Quelltext und führt einmal alle Operationen aus
- 4 verschiedene Implementationen: OpenCV, CImg, lodepng, CUDA

II Programmieraufwand

Neuimplementation

• häufig wiederholendene Abläufe (Schleifen, Berechnung des Index, ...)

• Operatoren wie Gaussian Blur benötigen starke Vereinfachungen

• Farbräume sind häufig uneindeutig und implementationsabhängig

• 3 verschiedene Bibliotheken für Kompabilität ausprobiert

CUDA

• Standardbibliotheken müssen ersetzt werden (crosscompilation)

• viele Funktionen schon gegeben

High-Level-Bibliothek

• häufig genutzte Funktionen schnell umsetzbar

• exotische Funktionen aufwendig

III Implementation: Algorithmen

Gaussian Blur

```
float filter[filter_size][filter_size] = {
  {1.0f,  4.0f,  6.0f,  4.0f,  1.0f},
  {4.0f, 16.0f, 24.0f, 16.0f,  4.0f},
  {6.0f, 24.0f, 36.0f, 24.0f,  6.0f}, ...
```

- Annäherung an Gauss durch im Vorfeld bestimmte Werte
- Iteration über Filter
- Filterposition bestimmt Pixelnachbar

HSV

- Problem: HSV hat nur 3 Farbkanäle
- Problem: Konvertierung hängt vom Farbspektrum ab

```
if(diff != 0) {
   if(cmax == red)
      hue = 43 * ((green - blue) / diff);
   else if(cmax == green)
      hue = 85 + 43 * ((blue - red) / diff);
   else
      hue = 200 + 171 + 43 * ((red - green) / diff); /*
200 is temporary. It works for some reason. */ }
```

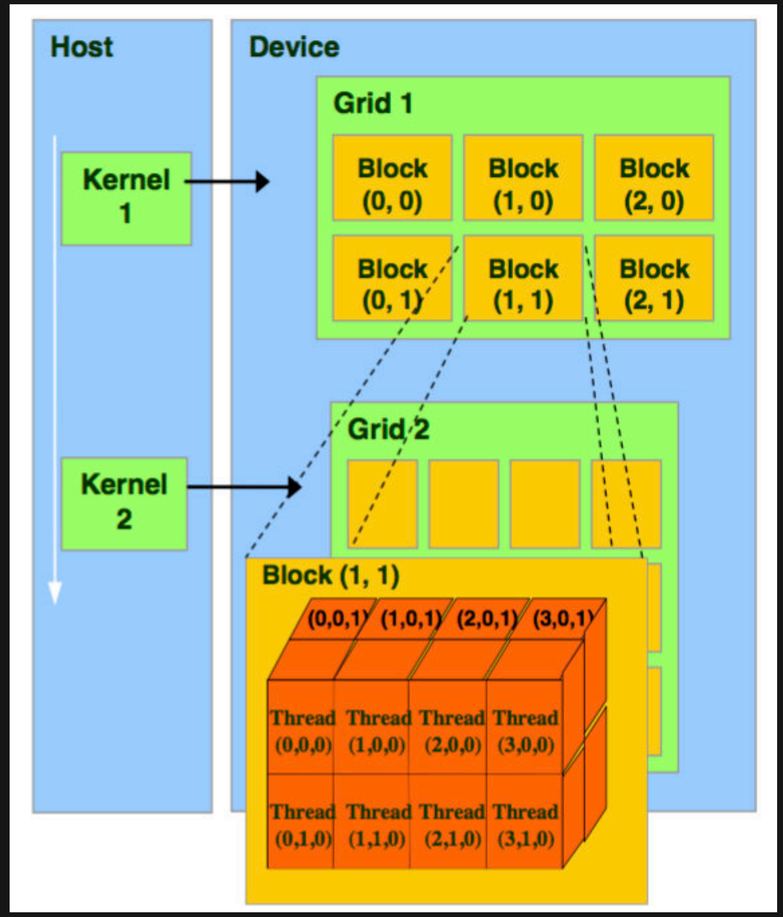
III Implementation: Parallelisierung

```
dim3 blocks((x / threads.x + 1), (y / threads.y + 1));
```

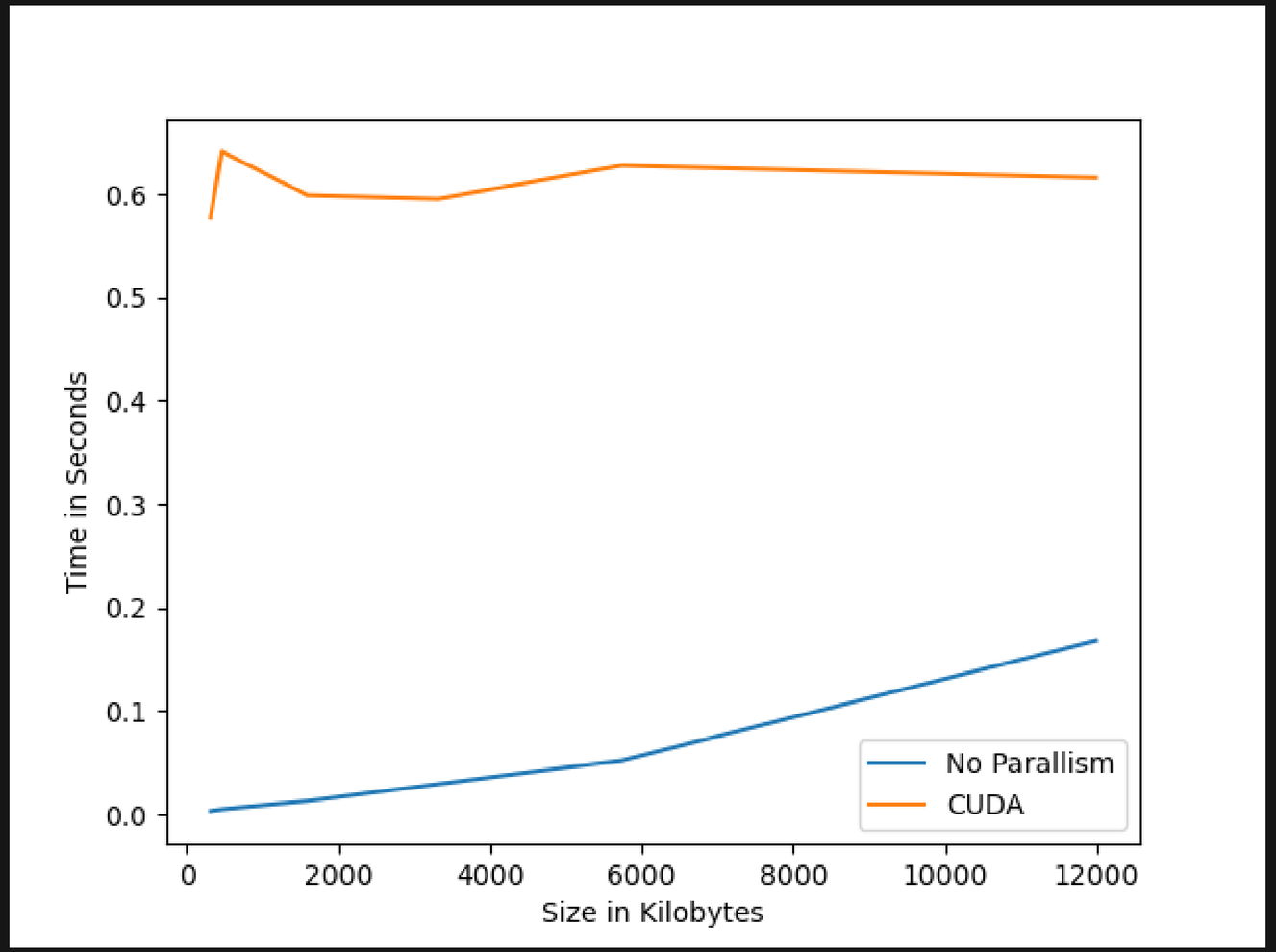
- Verteilung mittels GPU auf Threads und Blöcke

- Parallelisierung der Operationen erfolgt mittels Kernels automatisch über CUDA

- Operationen verarbeiten Daten vom Inputbuffer und schreiben in den Outputbuffer

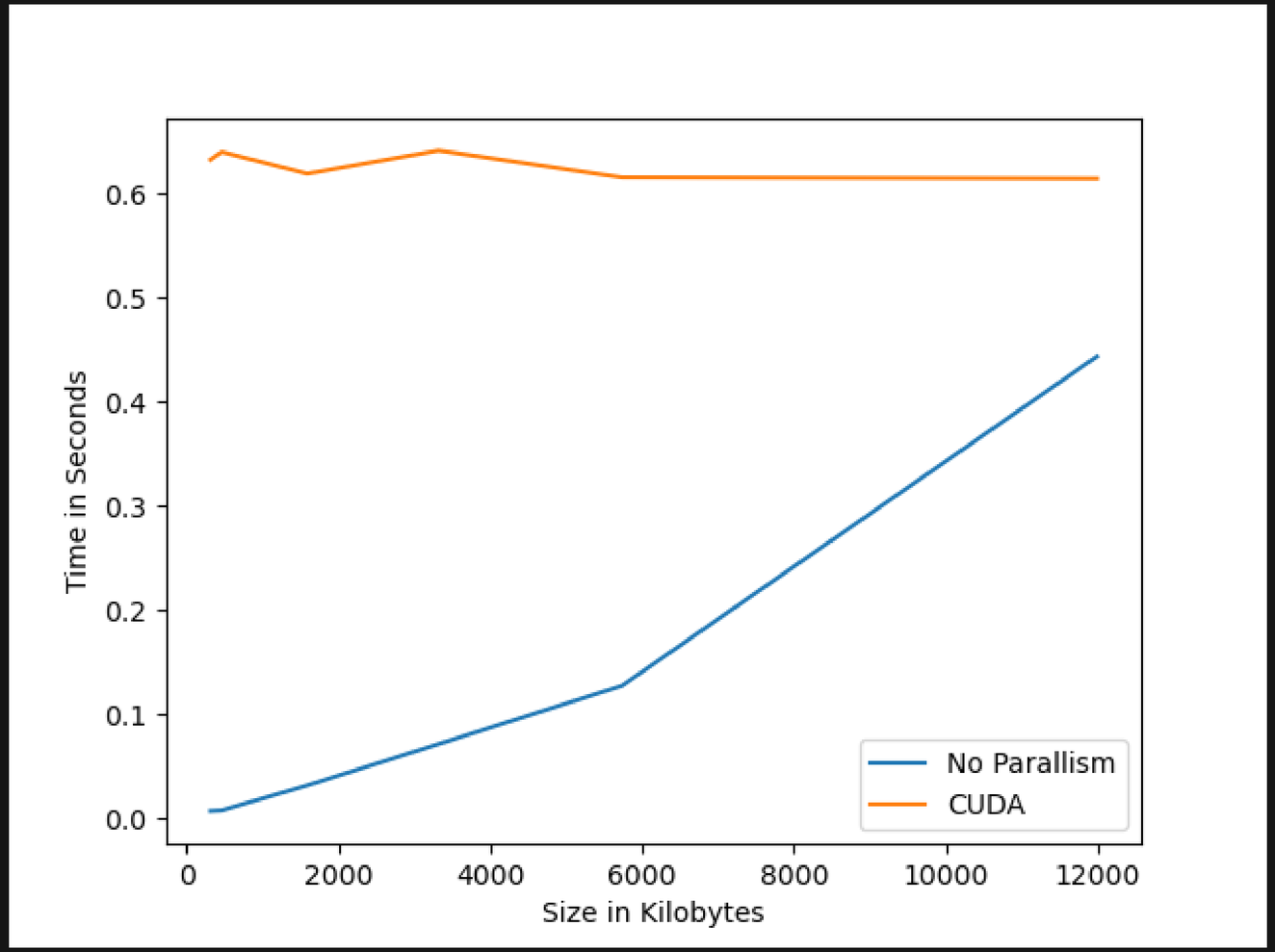


IV Vergleich Ausführungszeiten - Greyscale



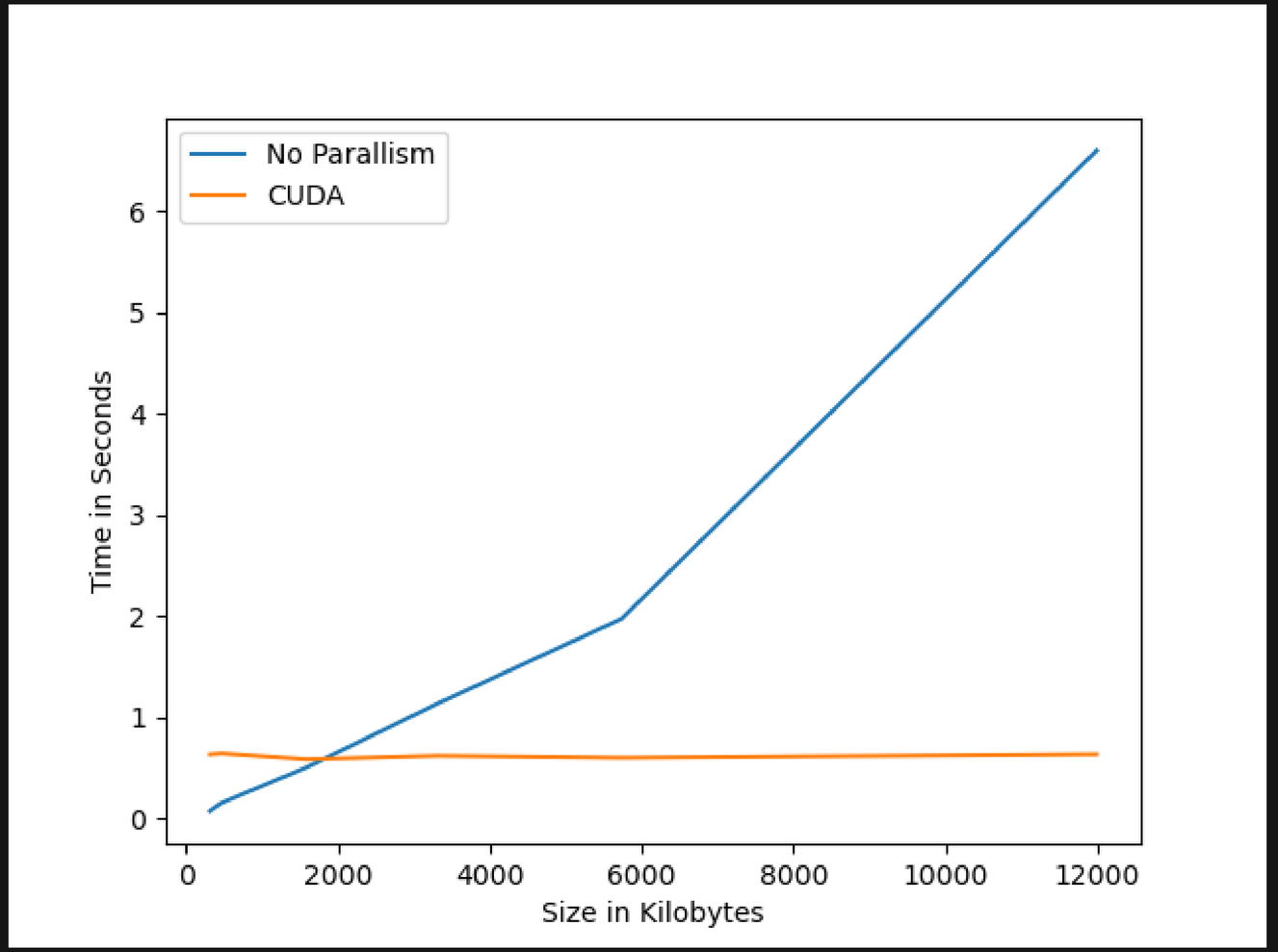
Getestet auf dem Nivida Cluster HTW Berlin

IV Vergleich Ausführungszeiten - HSV



Getestet auf dem Nivida Cluster HTW Berlin

IV Vergleich Ausführungszeiten - Blur



Getestet auf dem Nivida Cluster HTW Berlin

V Interpretation der Ergebnisse

1. für einfache Berechnungen:
   "CUDA benötigt länger die Grafikkarte anzusprechen,
   als der Prozessor für die Berechnung."

2. für komplexe parallelisierbare Berechnungen:
   "Durch multithreaded SIMD überholt CUDA bei großen Datenmengen
   den Prozessor über Parallelisierung der Berechnungen."

3. Beobachtung für CUDA:
   "Solange die Grafikkarte nicht vollständig ausgelastet ist,
   benötigen alle gemeinsamen Berechnungen ungefähr gleich viel Zeit."

VI Optimierungen

- datenorientiertes Layout (SOA etc)
- SIMD / MIMD Intrinsics implementieren
- CUDA Threads optimieren

VII Demo

DEMO

No Parallism

CUDA