Fachpraktikum Parallele Programmierung, Wintersemester 2024/25

1. Aufgabe: Bild-Filter

Norbert Baumstark, Johannes Becker, Stefan Butz

1 Aufgabenstellung

Es sollten zwei Filter zur Bearbeitung von JPEG-Bildern erstellt werden:

- ein Filter, der ein Farbbild in Graustufen umrechnet;
- ein Weichzeichner-Filter, der durch Durchschnittsbildung mit umliegenden Pixeln einen Unschärfeeffekt erzeugt. Der Filter soll sowohl Graustufen- als auch Farbbilder sowie verschiedene Randgrößen unterstützen.

Es sollten die Zeiten für die Berechnung sowie den Speichertransfer analysiert und dokumentiert werden.

2 Verzeichnisstruktur

Auf der obersten Verzeichnisebene befinden sich drei Dateien:

- ein `Makefile`,
- ein Python-Skript `download_images.py`, welches einige Testbilder aus dem Web lädt, welche ihrer Größe wegen nicht beigefügt sind,
- ein Python-Skript `perform_measurements.py`, welches alle Messungen ausführt.

Es gibt folgende Unterverzeichnisse:

- `src/` enthält den Quellcode der beiden Kernels sowie die beiden Header-Dateien `jpeg.h` (eine objektorientierte Schnittstelle zu `libjpeg`) und `util.h` (zwei `ASSERT`-Makros sowie die Funktion `cudaInit`, die durch einmaligen Aufruf von `cudaFree(0)` dafür sorgt, dass zum Zeitpunkt der Messungen der Treiber bereits initialisiert ist).
- `analysis/` enthält ein Jupyter-Notebook zur Auswertung der Messergebnisse. Dieses diente der Analyse sowie der Erstellung von Tabellen und Diagrammen für die vorliegende Dokumentation.
- `doc/` enthält die vorliegende Dokumentation.
- `images/` enthält Bilddateien, die für die Messungen verwendet werden, außerdem ein Python-Skript, mit dem die ebenfalls enthaltene Datei `info.json` erstellt wurde, welche Informationen über die Bilddateien enthält, insbesondere die Bildgrößen.
- `nvidia/` enthält das mit den CUDA-Entwicklungstools mitgelieferte Beispielprogramm `deviceQuery`, welches Geräteinformationen über die GPU ausgibt.
- `images_from_web/` wird von `./download_images.py` angelegt und dient der Ablage der Testbilder aus dem Web.
- `bin/` wird ggf. von `make` angelegt und dient als Zielverzeichnis für die Objektdateien sowie die ausführbaren Binärdateien.
- `images_out/` wird ggf. von `perform_measurements.py` angelegt und dient der Ablage der Output-Bilddateien.
- `measurements/` wird ggf. von `perform_measurements.py` angelegt und dient der Ablage der Geräteinformationen und Messergebnisse.

In den abgegebenen Dateien sind die Messergebnisse enthalten, die dieser Dokumentation zugrunde liegen.

3 Aufruf

Wenn gewünscht, können zunächst durch

```
./download_images.py
```

die zusätzlichen Testbilder aus dem Web nachgeladen werden. Nach Erstellen der ausführbaren Dateien durch

```
make
```

startet man durch

```
./perform_measurements.py
```

die Messungen. Gegebenenfalls sind die Parameter im `Makefile` auf die verwendete Architektur und im Python-Skript der Pfad zum Python-Interpreter anzupassen.

Das Python-Skript `perform_measurements.py` ermittelt zunächst die Geräteinformationen durch Aufruf von `bin/deviceQuery` und erstellt dann mittels des Tools `nsys` von NVIDIA Laufzeitprofile für `bin/01_grayscale` und `bin/02_blur`. Den beiden Binarys wird durch Kommandozeilenargumente Eingabe- und Ausgabedatei mitgeteilt; `02_blur` erhält als erstes Kommandozeilenargument zusätzlich die zu verwendende Margin. Es wird also für jedes der Testbilder aufgerufen:

- `bin/01_grayscale` sowie
- `bin/02_blur` jeweils einmal mit den Margins 1, 2 und 3.

In einem Unterverzeichnis von `measurements/`, das den aktuellen Zeitstempel als Namen trägt, werden abgelegt:

- die Geräteinformationen als Textdatei,
- jeweils der Output von `nsys` als Textdatei,
- die aus dem Output extrahierten für diese Aufgabe relevanten Messergebnisse als JSON-Datei.

Im Unterverzeichnis `measurements/[Zeitstempel]/temp/` legt `nsys` temporäre Dateien ab.

Die Binarys können wie folgt direkt aufgerufen werden:

```
bin/01_grayscale input.jpg output.jpg
```

```
bin/02_blur 2 input.jpg output.jpg
```
(„2“ spezifiziert hier eine Margin von 2).

Die Aufrufe von `nsys`, welche `perform_measurements.py` vornimmt, haben dementsprechend folgende Form:

```
nsys profile --stats=true \
  --output measurements /[Zeitstempel]/temp/report%n \
  bin/01_grayscale input.jpg output.jpg
```

bzw.

```
nsys profile --stats=true \
  --output measurements /[Zeitstempel]/temp/report%n \
  bin/02_blur 2 input.jpg output.jpg
```

4 Der Programmcode: Umwandlung in Graustufen – 01_grayscale.cu

Jeder Thread bearbeitet genau ein Pixel. Der Grauwert wird als Linearkombination von Rot-, Blau- und Gelbwert gebildet:

```
__global__ void RgbToGrayscale(
    unsigned char *inputImage, unsigned char *outputImage,
    int width,int height
) {
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;
    int idx = y * width + x;

    if (x < width && y < height) {
        int rgb_idx = idx * 3;

        unsigned char r = inputImage[rgb_idx];
        unsigned char g = inputImage[rgb_idx + 1];
        unsigned char b = inputImage[rgb_idx + 2];

        unsigned char gray = static_cast<unsigned char>(
           0.299f * r + 0.587f * g + 0.114f * b
        );

        outputImage[idx] = gray;
    }
}
```

Entsprechend wird die Anzahl der Blöcke auf Grundlage der Bildgröße berechnet; der Aufruf des Kernels geschieht wie folgt:

```
dim3 blockSize(16, 16);
dim3 gridSize(
    (width + blockSize.x - 1) / blockSize.x,
    (height + blockSize.y - 1) / blockSize.y
);

RgbToGrayscale<<<gridSize, blockSize>>>(
    dInputImage, dOutputImage, width, height
);
```

Um zufällige Messfehler zu reduzieren, ruft `main()` den Filter 100mal auf.

5 Der Programmcode: Weichzeichnen – 02_blur.cu

Es bearbeitet jeder Thread genau einen Farbkanal genau eines (Output-)Pixels. Die Anzahl der Threads entspricht also der Anzahl der Kanäle (3 für Farbbilder, 1 für Graustufenbilder) mal der Anzahl der Pixel. Die verwendeten Blöcke sind dreidimensional, wobei die z-Komponente den zu bearbeitenden Farbkanal kodiert. Zur Weichzeichnung wird der Durchschnittswert aller Pixel berechnet, die in x- oder in y-Richtung höchstens um die Margin vom Outputpixel entfernt sind.

```
__global__ void Blur(
    unsigned char *inputImage, unsigned char *outputImage,
    int width, int height,
    int channels, int margin
) {
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;
    int channel = threadIdx.z;

    if (x < width && y < height) {
        int startX = max(x - margin, 0);
```

```
        int endX = min(x + margin, width);
        int startY = max(y - margin, 0);
        int endY = min(y + margin, height);

        float v = 0;
        for (int i = startY; i < endY; i++) {
            for (int j = startX; j < endX; j++) {
                v += inputImage[(i * width + j) * channels + channel];
            }
        }
        float n = (endX - startX) * (endY - startY);
        outputImage[(y * width + x) * channels + channel] = v / n;
    }
}
```

Der Aufruf des Kernels erfolgt entsprechend folgendermaßen:

```
dim3 blockSize(16, 16, channels);
dim3 gridSize(
    (width + blockSize.x - 1) / blockSize.x,
    (height + blockSize.y - 1) / blockSize.y
);

Blur<<<gridSize, blockSize>>>(
    dInputImage, dOutputImage, width, height, channels, margin
);
```

Auch hier ruft `main()` den Filter 100mal auf, um zufällige Messfehler zu reduzieren.

6 Vorgehen bei den Messungen

Wir haben bei dieser Aufgabe Zeitmessungen sowohl mittels CUDA-Events als auch mit dem Tool `nsys` von NVIDIA ausprobiert. Die Ergebnisse stimmten weitgehend überein. Für die Einsendung haben wir uns dazu entschlossen, die Ergebnisse von `nsys` zu dokumentieren. (In anderen Aufgaben verwenden wir CUDA-Ereignisse.)

Aus den von `nsys` erstellten Reports extrahieren wir die Laufzeitinformationen zu

- der Startdauer des Kernels („cudaLaunchKernel“),
- der Zeit für den Transfer vom Host zum Device („CUDA memcpy Host-to-Device“),
- der Ausführungszeit des Kernels,
- der Zeit für den Transfer vom Device zum Host („CUDA memcpy Device-to-Host“).

7 Eingabedaten

Als Eingabedaten verwenden wir farbige Testbilder verschiedener Größe. Tabelle 1 gibt einen Überblick.

Bild	Breite	Höhe	Pixel
01_small.jpg	100	100	10000
toledo01.jpg	320	214	68480
toledo02.jpg	640	428	273920
toledo03.jpg	800	535	428000
toledo04.jpg	1024	685	701440
02_medium.jpg	1200	675	810000
toledo05.jpg	1280	857	1096960
toledo06.jpg	2560	1713	4385280
03_large.jpg	5616	3744	21026304
toledo07.jpg	13226	8852	117076552

Tabelle 1. Die für die Messungen verwendeten Bilddateien.

8 Hardware

Die Messungen wurden auf folgenden Geräten durchgeführt:

- NVIDIA Jetson Xavier NX 16 GB in einem Seeed Studio reComputer J2022;
  GPU gemäss Datenblatt: 384-core NVIDIA Volta GPU with 48 Tensor Cores,
  CPU gemäss Datenblatt: 6-core NVIDIA Carmel ARM v8.2 64-bit CPU 6MB L2 + 4MB L3.
- NVIDIA Tesla V100-SXM2-32GB,
  funkel.fernuni-hagen.de.

Die Ausgaben von `deviceQuery` für die beiden Geräte sind in Anhang A wiedergegeben.

9 Messergebnisse

Die Werte finden sich in tabellarischer Form in Anhang B. Wir nehmen hier eine graphische Auswertung vor. blur-01, blur-02 und blur-03 bezeichnen den Weichzeichnungsfilter mit einer Margin von 1, 2 bzw. 3, grayscale bezeichnet die Graustufenumwandlung

Abbildung 1 zeigt die Zeiten für den Transfer der Daten vom Host zum Device in Abhängigkeit von der Bildgröße, d.h. der Anzahl Pixels. Wie zu erwarten, ist die Transferzeit einigermaßen linear in der Anzahl der Pixel. Abbildung 2 zeigt die Transferzeiten in umgekehrter Richtung, d.h. vom Device zum Host. Auch hier zeigt sich eine im Wesentlichen lineare Abhängigkeit. Bei der Umwandlung in Grauwerte sind die Transferzeiten nun wesentlich kürzer als bei den Weichzeichnungsfiltern. Dies überrascht nicht, da jeweils nur ein Drittel der Datenmenge zu übertragen ist.

In Abbildung 3 sind die Ausführungszeiten in Abhängigkeit von der Bildgröße dargestellt. Für die Graustufenumwandlung sowie für jede Weichzeichnung mit jeder festen Margin zeigt sich eine im Wesentlichen lineare Abhängigkeit, wobei die Steigungen aufgrund des unterschiedlichen Aufwands je Outputpixel nun unterschiedlich sind. Es fällt auf, dass kleinere Bilder eher überproportional viel Aufwand verursachen. Der Grund dafür könnte – aber das ist Spekulation – dadurch bedingt sein, dass bei kleineren Bildern aufgrund der festen Blockgröße der Anteil an „Randblöcken“ in Relation zur gesamten Anzahl Blöcke größer ist und somit der Rechenaufwand überproportional zur Anzahl Bildpixel ist.

Abbildung 4 zeigt die Summe aus Transfer- und Ausführungszeiten. Wegen des im Wesentlichen linearen Zusammenhangs für jede der Komponenten ergibt sich für deren Summe hier jeweils ein einigermaßen linearer Verlauf, insbesondere auf dem leistungsstärkeren Tesla-Gerät. Abbildung 5 zeigt die Zusammensetzung der Gesamtlaufzeit aus den Zeiten für den Start des Kernels, die Transfers und die Ausführung. Man sieht, dass die Zeit für den Start des Kernels jeweils zu vernachlässigen ist. Man sieht ebenfalls, dass für die betrachteten Eingabegrößen bei dem Xavier-Gerät (insbesondere beim Weichzeichnungsfilter) die Ausführungszeit die wesentliche Komponente ist, während beim Tesla-Gerät die Transferzeiten dominieren. Dies ist wohl durch die jeweilige Art der Einbindung der GPU in das Gesamtsystem sowie die deutlich geringere Anzahl Cores beim Xavier-Gerät zu erklären.

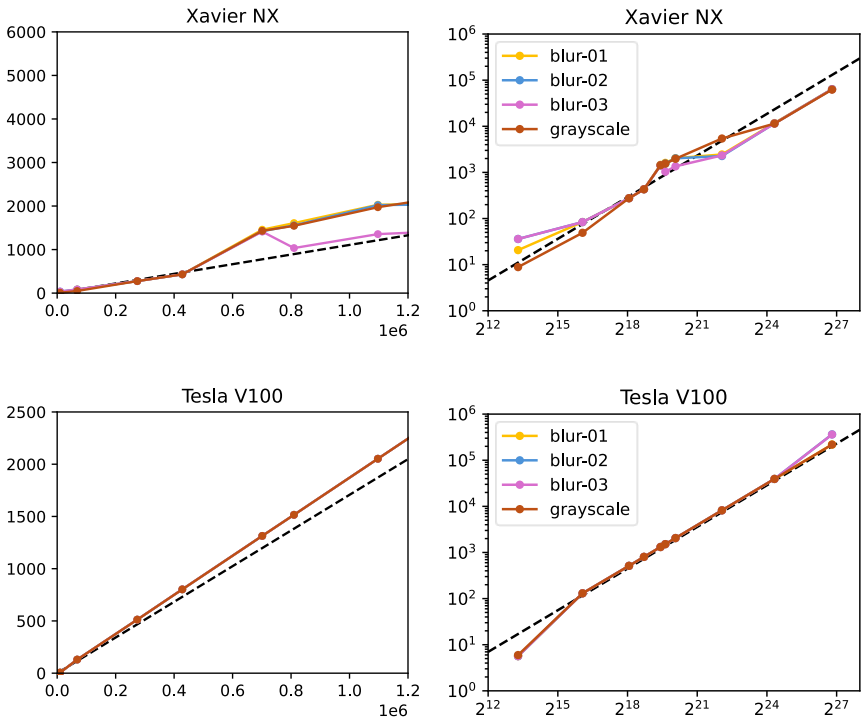

Abbildung 1. Arithmetische Mittel der Zeiten für den Transfer der Daten vom Host zum Device in Abhängigkeit von der Bildgröße. Die horizontale Achse stellt die Anzahl Pixel dar. Auf der vertikalen Achse ist jeweils die mittlere Transferzeit in Mikrosekunden (µs) abgetragen. Das linke Diagramm zeigt die Werte für die Bilder mit bis zu 1.2 Millionen Pixeln, mit linear skalierten Achsen. Das rechte Diagramm zeigt alle Werte, mit logarithmisch skalierten Achsen. Die gestrichelte schwarze Gerade zeigt eine (mit logarithmischen Kosten) „gefittete" Ursprungsgerade, d.h. eine ideale lineare Laufzeit.

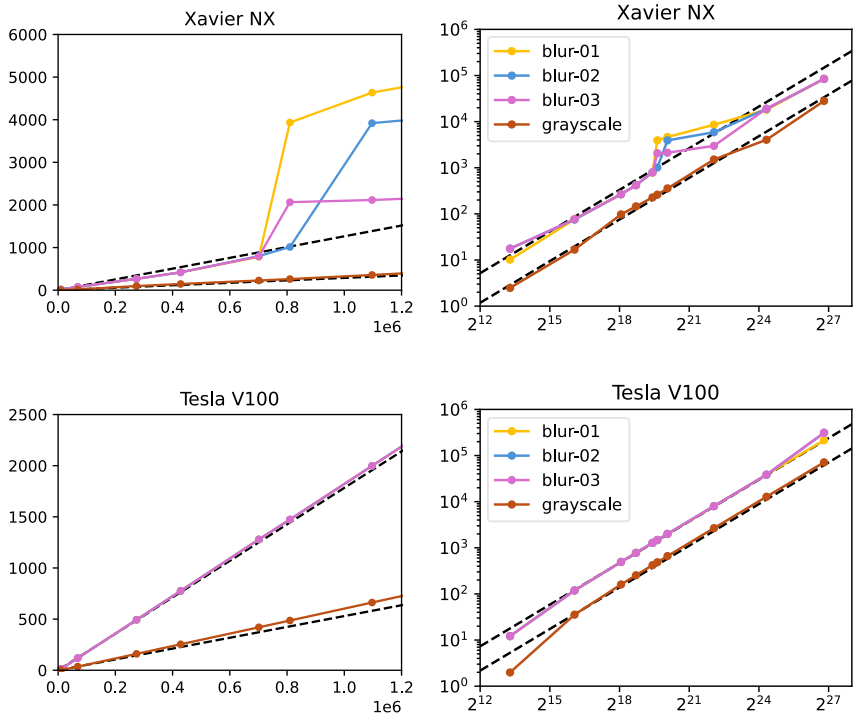

Abbildung 2. Arithmetische Mittel der Zeiten für den Transfer der Daten vom Device zum Host (vertikale Achse, in µs) in Abhängigkeit von der Bildgröße (horizontale Achse, in Pixeln). Aufgrund der unterschiedlichen Datenmengen wurden nun für die Umwandlung in Graustufen und den Weichzeichnungsfilter zwei unterschiedliche Ursprungsgeraden gefittet.

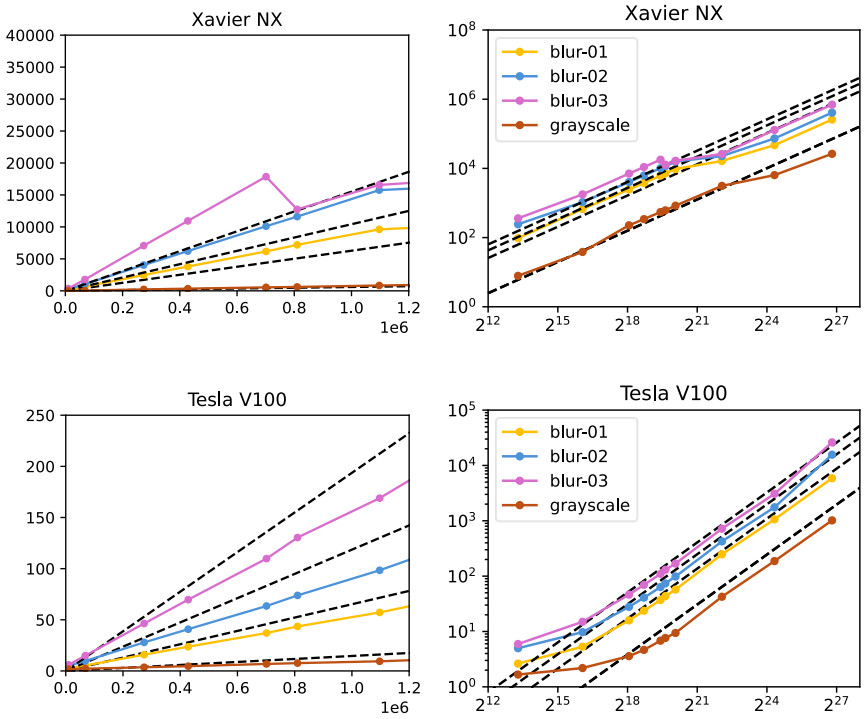

Abbildung 3. Arithmetische Mittel der Ausführungszeiten (vertikale Achse, in µs) in Abhängigkeit von der Bildgröße (horizontale Achse, in Pixeln). Wegen des prinzipiell unterschiedlichen Rechenaufwands wurde für jede Variante eine separate Ursprungsgerade gefittet.

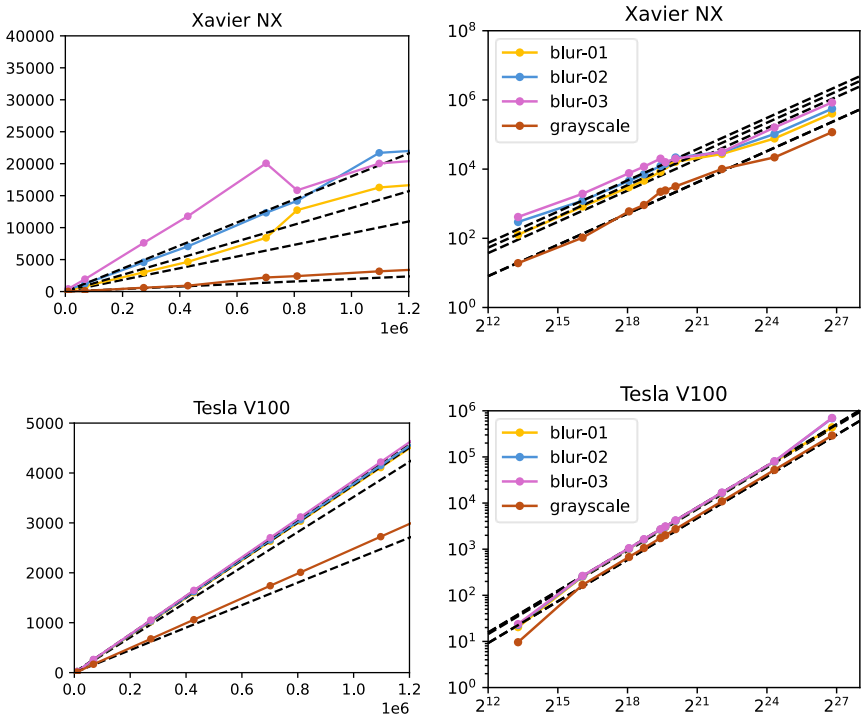

Abbildung 4. Summen der arithmetischen Mittel aus Transferzeiten und Ausführungszeit (vertikale Achse, in µs) in Abhängigkeit von der Bildgröße (horizontale Achse, in Pixeln).

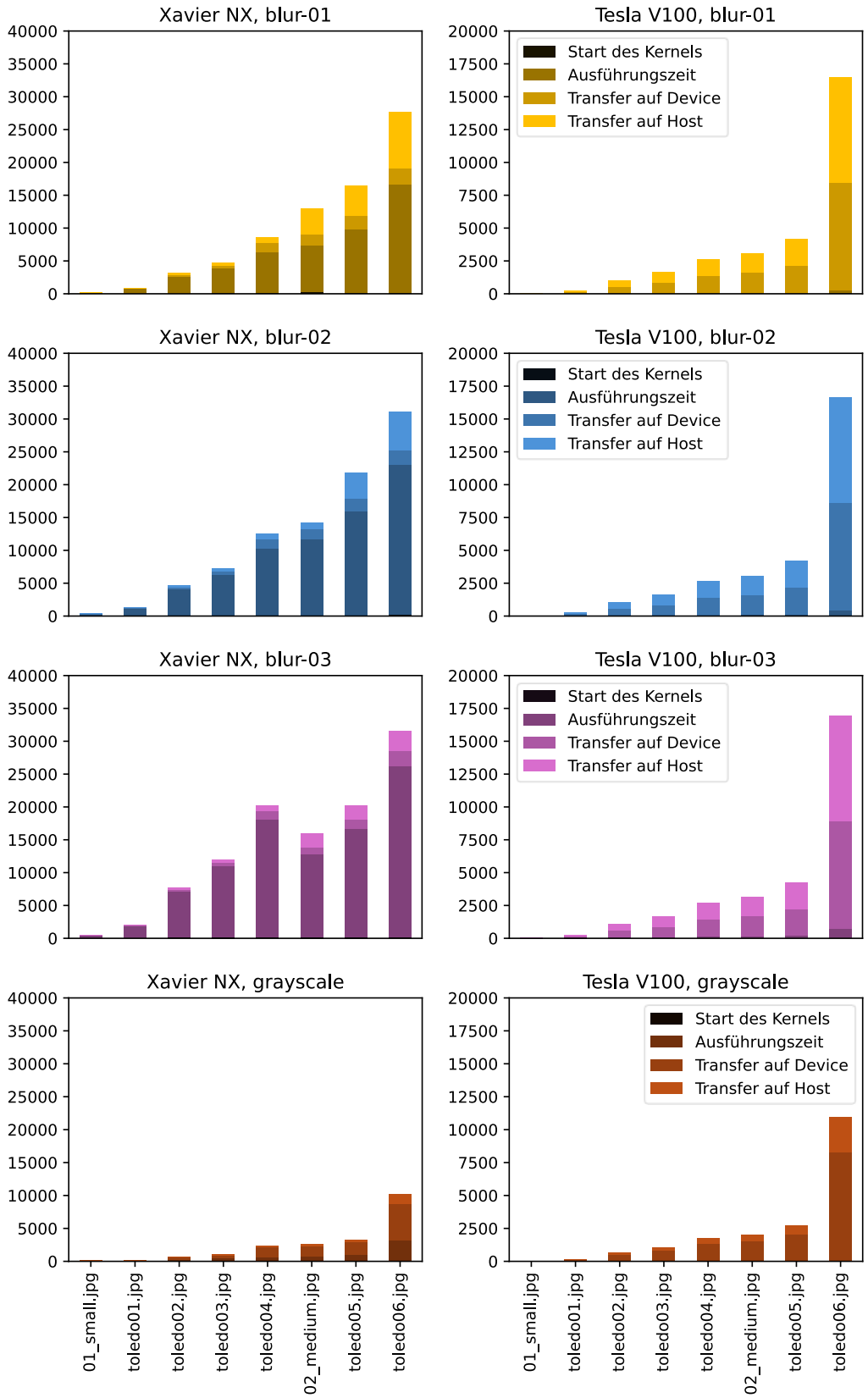

Abbildung 5. Mittlere Zeiten für Kernelstart, Transfers und Ausführung (in µs) im Vergleich. Die beiden größten Eingabebilder (03_large.jpg und toledo07.jpg) wurden aus Darstellungsgründen hier nicht berücksichtigt.

Wir untersuchen abschließend noch die Abhängigkeit der *Ausführungszeit* von der Größe der Margin im Weichzeichnungsfilter (Abbildung 6). Theoretisch zu erwarten ist ein in der Margin asymptotisch quadratischer Aufwand. Für die meisten Testbilder zeigt sich im Diagramm das erwartete Verhalten: Wenngleich natürlich das asymptotische Verhalten aufgrund der kleinen Marginwerte nicht ablesbar ist, ist die Laufzeit jedenfalls in der Regel überlinear. Zu bemerken ist, dass wir auf dem Xavier-Gerät für drei Testbilder mittlerer Größe genau das umgekehrte, d.h. ein unterlineares Wachstum und auf dem Tesla-Gerät für das größte Testbild einen nahezu linearen Verlauf sehen – warum, ist unklar. Ein systematischer Effekt für Bilder „mittlerer“ Größe ist nicht ganz auszuschließen, denn es ist zu bedenken, dass für stärkere Hardware eine größere Eingabegröße als „mittel“ zu betrachten wäre. Das ist aber Spekulation.

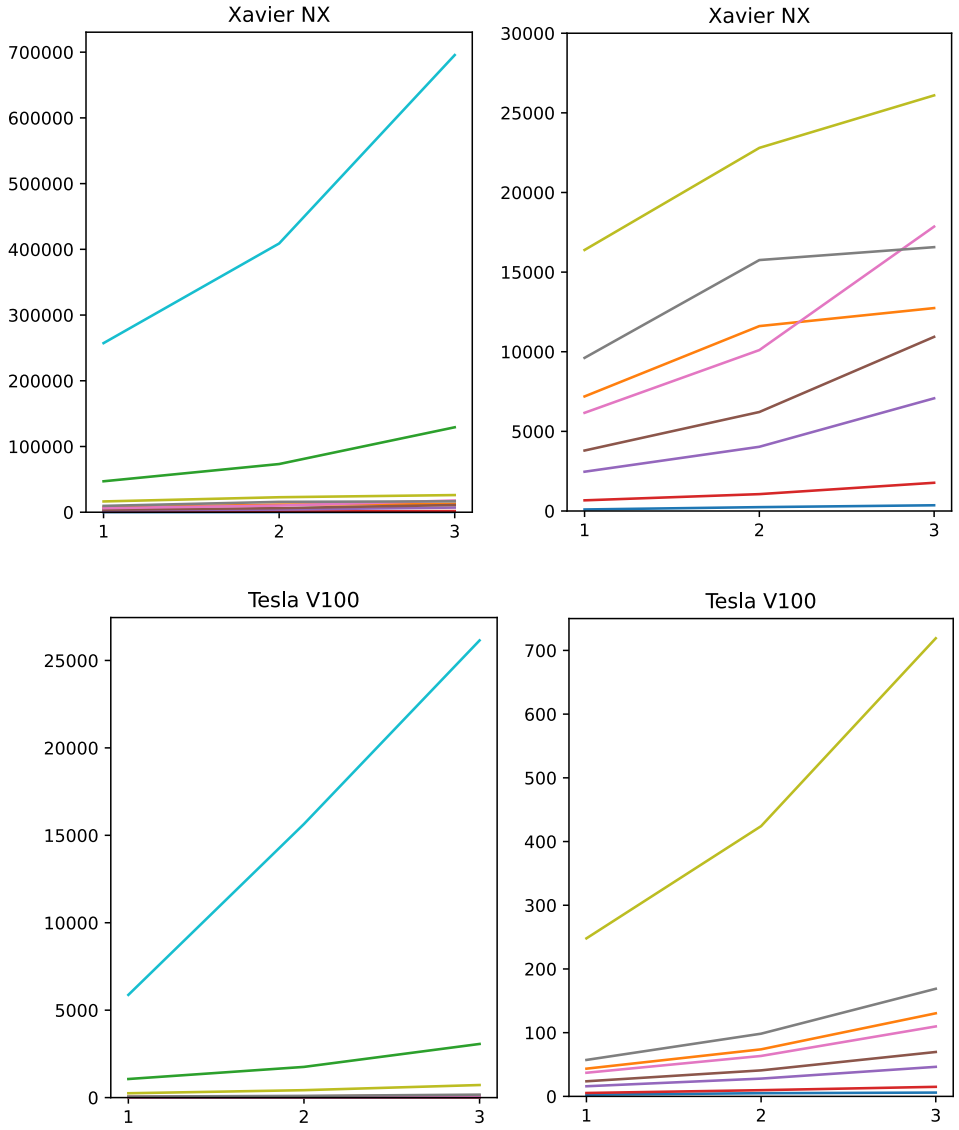

Abbildung 6. Arithmetische Mittel der Ausführungszeiten (vertikale Achse, in µs) in Abhängigkeit von der Margin (horizontale Achse, in Pixeln). Die beiden Diagramme in jeweils derselben Zeile unterscheiden sich lediglich in der Skalierung der vertikalen Achse. Jede Kurve repräsentiert eines der Testbilder.

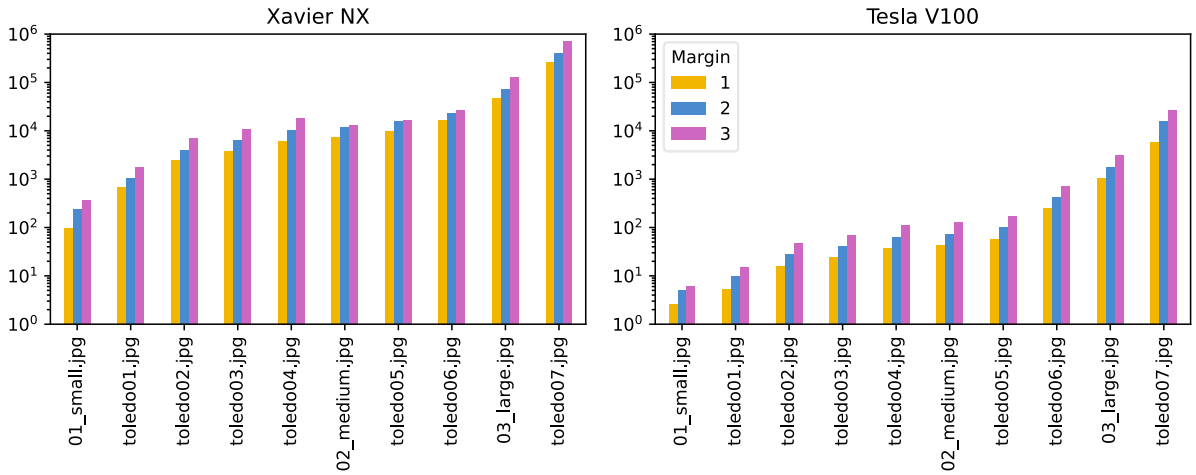

Abbildung 7. Ausführungszeiten (in µs) für den Weichzeichnungsfilter für die verschiedenen Eingabebilder in Abhängigkeit von der Margin.

Abbildung 7 stellt die Ausführungszeiten auf den beiden verwendeten Geräten gegenüber. Trotz der Unterschiede im Detail sind die oberen Konturen beider Diagramme von der Form her sehr ähnlich. Wir können also – mit aller Vorsicht – sagen, dass wir auf beiden Diagrammen das gleiche Laufzeitverhalten beobachten, wobei die Ausführungsgeschwindigkeit auf dem Tesla-Gerät natürlich generell höher ist.

Anhang A. Ausgabe von deviceQuery für die verwendete Hardware.

Xavier NX

[...]

```
Device 0: "Xavier"
  CUDA Driver Version / Runtime Version          10.2 / 10.2
  CUDA Capability Major/Minor version number:    7.2
  Total amount of global memory:                 15827 MBytes (16596041728
bytes)
  ( 6) Multiprocessors, ( 64) CUDA Cores/MP:     384 CUDA Cores
  GPU Max Clock rate:                            1109 MHz (1.11 GHz)
  Memory Clock rate:                             1109 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 524288 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072,
65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048
layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            Yes
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 0 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device
simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 10.2, CUDA Runtime
Version = 10.2, NumDevs = 1
Result = PASS
```

Tesla V100

[...]

```
Device 0: "Tesla V100-SXM2-32GB"
  CUDA Driver Version / Runtime Version          12.4 / 12.3
  CUDA Capability Major/Minor version number:    7.0
  Total amount of global memory:                 32494 MBytes (34072559616
bytes)
  (80) Multiprocessors, ( 64) CUDA Cores/MP:     5120 CUDA Cores
  GPU Max Clock rate:                            1530 MHz (1.53 GHz)
  Memory Clock rate:                             877 Mhz
  Memory Bus Width:                              4096-bit
  L2 Cache Size:                                 6291456 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072,
65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048
layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 4 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   4 / 4 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device
simultaneously) >
```

[...]

```
deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 12.4, CUDA Runtime
Version = 12.3, NumDevs = 4
Result = PASS
```

Anhang B. Messwerte

SD bezeichnet die Standardabweichung.

Xavier NX, blur-01

		Start Kernel (µs)					Transfer Host to Device (µs)					Ausführung (µs)					Transfer Device To Host (µs)				
Image	**Pixels**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**
01_small.jpg	10000	40	46	117	54	14	20	20	22	21	0	94	96	101	96	1	10	10	10	10	0
toledo01.jpg	68480	45	61	117	70	18	81	84	87	84	1	641	669	694	670	10	75	76	77	76	0
toledo02.jpg	273920	65	90	1445	107	137	271	275	286	275	2	2424	2463	2526	2467	21	266	267	274	267	1
toledo03.jpg	428000	73	102	1446	115	137	425	430	446	431	3	3742	3792	3876	3797	30	418	419	428	419	1
toledo04.jpg	701440	75	161	681	183	109	1365	1458	1668	1457	66	6085	6163	6270	6165	38	677	678	6390	782	610
02_medium.jpg	810000	82	149	1952	200	199	819	1616	1793	1608	133	7124	7192	7274	7194	31	3369	3588	6197	3934	688
toledo05.jpg	1096960	83	118	1851	178	292	1868	1957	2408	2030	131	9512	9618	9716	9616	42	3893	4015	13232	4636	1182
toledo06.jpg	4385280	89	106	1107	203	232	2113	2234	5624	2448	652	14992	15078	38856	16391	3959	7881	8196	13855	8566	856
03_large.jpg	21026304	92	175	1070	392	303	11116	11535	25594	11731	1421	46040	46108	92041	47073	5064	16523	18015	24938	18105	846
toledo07.jpg	117076552	334	682	1241	684	125	62662	63225	112273	63767	4912	256561	256759	310539	257293	5379	80443	84499	89514	84420	1401

Tesla V100, blur-01

		Start Kernel (µs)					Transfer Host to Device (µs)					Ausführung (µs)					Transfer Device To Host (µs)				
Image	**Pixels**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**
01_small.jpg	10000	9	9	145	11	14	5	6	7	6	0	3	3	4	3	0	12	12	13	12	0
toledo01.jpg	68480	9	10	225	12	22	130	130	130	130	0	5	5	6	5	0	119	119	119	119	0
toledo02.jpg	273920	9	10	615	16	60	512	513	513	513	0	16	16	16	16	0	493	493	494	493	0
toledo03.jpg	428000	9	10	885	19	88	802	802	803	802	0	23	24	24	24	0	776	776	776	776	0
toledo04.jpg	701440	12	12	736	20	72	1313	1314	1315	1314	0	37	37	38	37	0	1279	1279	1280	1279	0
02_medium.jpg	810000	13	14	932	23	92	1515	1516	1516	1516	0	43	43	44	43	0	1473	1474	1474	1474	0
toledo05.jpg	1096960	15	16	823	26	81	2051	2052	2053	2052	0	57	57	59	57	0	1998	1998	1999	1998	0
toledo06.jpg	4385280	18	22	1071	34	105	8192	8193	8207	8194	2	246	248	250	248	1	8006	8007	8008	8007	0
03_large.jpg	21026304	53	57	854	66	80	39261	39263	40908	39280	164	1057	1061	1207	1063	15	38406	38408	38562	38410	16
toledo07.jpg	117076552	56	60	1337	74	128	218629	218636	220564	218679	262	5839	5850	6555	5874	103	213877	213883	213977	213886	12

Xavier NX, blur-02

		Start Kernel (µs)					Transfer Host to Device (µs)					Ausführung (µs)					Transfer Device To Host (µs)				
Image	**Pixels**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**
01_small.jpg	10000	39	47	153	55	17	36	36	37	36	0	234	242	256	243	5	17	18	18	18	0
toledo01.jpg	68480	49	56	127	67	18	82	84	86	84	1	1046	1059	1074	1059	5	75	75	76	75	0
toledo02.jpg	273920	68	99	1394	110	133	271	276	281	275	2	4001	4031	4067	4033	14	265	266	267	266	0
toledo03.jpg	428000	74	87	1466	112	139	426	431	448	431	3	6154	6221	6263	6216	27	417	418	429	419	2
toledo04.jpg	701440	74	142	798	164	107	748	1404	1592	1421	87	10043	10101	10156	10104	29	677	678	8309	801	792
02_medium.jpg	810000	82	115	589	126	72	836	1548	1702	1555	82	11577	11608	11642	11610	14	824	869	7679	1013	737
toledo05.jpg	1096960	85	117	1704	167	230	1901	1979	2335	2021	101	15662	15760	15882	15757	45	3230	3370	7168	3920	777
toledo06.jpg	4385280	94	104	1072	212	252	2092	2243	4271	2264	215	18564	24270	35753	22801	2876	5471	5705	11357	5888	742
03_large.jpg	21026304	91	178	936	395	297	11274	11481	12069	11520	157	73233	73325	73409	73325	37	16535	18930	24833	18908	1100
toledo07.jpg	117076552	376	672	1017	675	91	62463	63105	125011	63786	6199	406169	406339	664158	408917	25782	80839	84361	95181	84423	1701

Tesla V100, blur-02

		Start Kernel (µs)					Transfer Host to Device (µs)					Ausführung (µs)					Transfer Device To Host (µs)				
Image	**Pixels**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**
01_small.jpg	10000	9	10	158	11	15	5	6	7	6	0	5	5	6	5	0	12	12	13	12	0
toledo01.jpg	68480	9	9	227	12	22	130	130	130	130	0	9	10	11	10	0	119	119	119	119	0
toledo02.jpg	273920	9	10	614	16	60	512	513	513	513	0	27	28	29	28	0	493	493	494	493	0
toledo03.jpg	428000	10	10	885	19	87	801	802	804	802	0	40	41	42	41	0	776	776	777	776	0
toledo04.jpg	701440	12	14	739	21	73	1313	1314	1315	1314	0	63	63	64	63	0	1279	1279	1280	1279	0
02_medium.jpg	810000	14	15	933	24	92	1515	1516	1516	1516	0	73	74	75	74	0	1473	1474	1474	1474	0
toledo05.jpg	1096960	14	15	821	24	81	2051	2052	2053	2052	0	98	98	101	98	0	1998	1998	1999	1998	0
toledo06.jpg	4385280	17	22	1075	33	105	8191	8193	8204	8193	1	421	424	426	424	1	8006	8007	8389	8011	38
03_large.jpg	21026304	54	58	852	66	79	39266	39269	39286	39269	3	1749	1753	2034	1755	28	38405	38407	38421	38408	2
toledo07.jpg	117076552	45	60	1335	74	128	218627	218635	878059	363271	245277	9647	9660	38370	15646	10100	213878	213887	569849	310845	156315

Xavier NX, blur-03

		Start Kernel (µs)					Transfer Host to Device (µs)					Ausführung (µs)					Transfer Device To Host (µs)				
Image	**Pixels**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**
01_small.jpg	10000	41	47	117	56	16	36	36	37	36	0	346	359	381	360	7	17	18	18	18	0
toledo01.jpg	68480	50	60	122	69	18	82	84	85	84	1	1764	1774	1782	1773	4	75	77	78	76	1
toledo02.jpg	273920	67	97	1427	108	136	271	275	291	275	2	7044	7077	7095	7076	8	266	267	276	267	1
toledo03.jpg	428000	74	105	1697	135	224	426	431	457	431	3	10917	10935	10950	10935	7	417	418	432	419	2
toledo04.jpg	701440	79	155	783	170	99	693	1440	1537	1413	131	17831	17860	17880	17859	10	677	678	5555	792	561
02_medium.jpg	810000	81	115	643	144	100	447	1021	1624	1035	249	11911	11929	20984	12742	2602	1556	1671	6376	2066	737
toledo05.jpg	1096960	87	121	540	139	85	617	1298	2032	1355	240	15806	15846	27940	16567	2879	1698	1953	9972	2116	828
toledo06.jpg	4385280	92	103	735	178	181	2164	2327	2841	2335	90	25901	25911	32104	26097	1060	2666	2859	9653	2981	709
03_large.jpg	21026304	92	187	1099	403	305	11225	11480	12063	11504	162	129175	129235	129317	129239	34	16738	19112	25127	19079	935
toledo07.jpg	117076552	342	694	1211	708	145	62595	63165	84508	63431	2173	694034	694277	827898	695613	13362	80404	84712	91555	84696	1594

Tesla V100, blur-03

		Start Kernel (µs)					Transfer Host to Device (µs)					Ausführung (µs)					Transfer Device To Host (µs)				
Image	**Pixels**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**
01_small.jpg	10000	9	10	167	11	16	5	6	7	6	0	6	6	7	6	0	12	12	13	12	0
toledo01.jpg	68480	9	10	229	12	22	130	130	130	130	0	15	15	16	15	0	119	119	119	119	0
toledo02.jpg	273920	9	10	614	16	60	512	513	513	513	0	46	46	47	46	0	493	493	494	493	0
toledo03.jpg	428000	10	10	890	19	88	802	802	803	802	0	69	70	70	70	0	776	776	776	776	0
toledo04.jpg	701440	12	13	744	21	73	1313	1314	1317	1314	0	109	110	112	110	0	1279	1279	1281	1279	0
02_medium.jpg	810000	13	14	933	24	92	1515	1516	1516	1516	0	130	130	131	130	0	1473	1474	1474	1474	0
toledo05.jpg	1096960	15	16	825	24	81	2051	2052	2053	2052	0	168	169	171	169	0	1998	1998	2001	1998	0
toledo06.jpg	4385280	17	22	1068	34	105	8192	8193	8195	8193	0	718	719	721	719	1	8006	8006	8012	8007	1
03_large.jpg	21026304	53	57	860	66	80	39267	39269	41207	39289	194	3057	3063	3558	3068	50	38406	38408	38416	38408	2
toledo07.jpg	117076552	46	61	1336	74	128	218626	218632	815560	361234	235831	16391	16412	55732	26146	15807	213879	213887	566920	311671	154300

Xavier NX, grayscale

		Start Kernel (µs)					Transfer Host to Device (µs)					Ausführung (µs)					Transfer Device To Host (µs)				
Image	**Pixels**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**
01_small.jpg	10000	39	65	148	63	18	8	9	9	9	0	7	8	8	8	1	2	2	3	2	0
toledo01.jpg	68480	48	61	224	71	27	48	49	50	49	0	37	39	39	38	1	17	17	17	17	0
toledo02.jpg	273920	60	74	161	87	24	271	276	286	276	2	224	226	230	226	1	97	97	107	97	1
toledo03.jpg	428000	63	84	1370	104	130	424	430	440	430	3	338	341	345	341	1	145	146	146	145	0
toledo04.jpg	701440	69	143	854	160	115	1357	1429	1602	1430	52	544	547	551	547	1	226	227	228	227	0
02_medium.jpg	810000	79	90	762	110	72	1510	1537	1701	1546	35	608	613	618	613	2	260	261	262	261	0
toledo05.jpg	1096960	79	109	1837	146	242	1165	1934	2248	1973	121	830	838	843	837	2	355	357	358	357	1
toledo06.jpg	4385280	82	99	686	146	125	3101	5080	10874	5426	1199	1867	3216	3225	3122	346	871	1398	10096	1510	947
03_large.jpg	21026304	92	130	1469	348	352	10526	10733	49497	11420	3943	5993	6003	15115	6391	1225	3551	3719	24269	4041	2063
toledo07.jpg	117076552	353	717	1183	717	113	58499	61523	89512	62366	3601	22589	27097	51424	26396	4067	27197	27641	53782	28079	2795

Tesla V100, grayscale

		Start Kernel (µs)					Transfer Host to Device (µs)					Ausführung (µs)					Transfer Device To Host (µs)				
Image	**Pixels**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**	**Min**	**Med**	**Max**	**Avg**	**SD**
01_small.jpg	10000	9	9	136	11	13	5	6	7	6	0	2	2	3	2	0	2	2	3	2	0
toledo01.jpg	68480	9	10	211	12	20	130	130	130	130	0	2	2	3	2	0	35	35	36	36	0
toledo02.jpg	273920	9	9	596	15	59	512	513	513	512	0	4	4	5	4	0	160	160	161	160	0
toledo03.jpg	428000	9	10	865	18	86	801	802	802	802	0	5	5	6	5	0	254	254	254	254	0
toledo04.jpg	701440	10	11	727	18	72	1313	1314	1316	1314	0	7	7	7	7	0	420	420	421	420	0
02_medium.jpg	810000	11	11	912	20	90	1515	1516	1516	1516	0	8	8	8	8	0	486	486	486	486	0
toledo05.jpg	1096960	12	12	811	20	80	2051	2052	2057	2052	1	9	9	10	9	0	662	662	663	662	0
toledo06.jpg	4385280	16	18	1051	30	103	8192	8193	8197	8193	1	41	42	44	42	1	2665	2665	2669	2665	1
03_large.jpg	21026304	53	57	845	66	79	39273	39275	39291	39275	2	185	187	190	187	1	12801	12801	12807	12802	1
toledo07.jpg	117076552	46	60	1315	76	125	218625	218631	220601	218657	197	1007	1016	1072	1018	8	71287	71290	71320	71291	5