3. Aufgabe: Histogramm

Johannes Becker, Norbert Baumstark, Stefan Butz

1 Aufgabenstellung

Es sollten Routinen zur Erstellung eines Histogramms auf einer GPU implementiert und hinsichtlich ihrer Laufzeit untersucht werden. Dabei sollte gezählt werden, wie oft einzelne Bytewerte in einem Array von Bytes vorkommen. Das Hochzählen der Anzahlen in den Bins des Histogramms sollte durch atomare Operationen erfolgen. Es waren zwei Varianten zu untersuchen: zum einen das Hochzählen direkt im globalen Speicher, zum anderen sollten zunächst mehrere Instanzen der Bins im Shared Memory angelegt und diese erst abschließend zu einem einzigen Histogramm akkumuliert werden.

Die Eingabedaten waren nach Annahme Bytewerte zwischen 1 und 128. Es waren zwei Varianten des Histogramms zu erstellen, eine mit 128 Bins für die einzelnen Bytewerte sowie eine weitere mit jeweils einem Bin für die Buchstaben ‚A' bzw. ‚a' bis ‚Z' bzw. ‚z' und einem zusätzlichen Bin für alle anderen Zeichen, d.h. insgesamt 27 Bins.

2 Verzeichnisstruktur

Auf der obersten Verzeichnisebene befinden sich zwei Dateien, nämlich ein `Makefile` sowie ein Python-Skript `perform_measurements.py`, das alle Messungen ausführt.

Es gibt folgende Unterverzeichnisse:

- `src/` enthält den Quellcode.
- `analysis/`: enthält ein Jupyter-Notebook zur Auswertung der Messergebnisse. Das Notebook ist nicht zur Präsentation der Ergebnisse gedacht; dazu dient die vorliegende Dokumentation.
- `doc/` enthält diese Dokumentation.
- `input_data/` enthält die Testdaten, die der Aufgabenstellung beigefügt waren.
- `nvidia/` enthält das mit den CUDA-Entwicklungstools mitgelieferte Beispielprogramm `deviceQuery`, welches Geräteinformationen über die GPU ausgibt.
- `bin/` wird ggf. von `make` angelegt und dient als Zielverzeichnis für die Objektdateien sowie die ausführbaren Dateien `deviceQuery` und `histogram`.
- `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

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 ermittelt zunächst die Geräteinformationen durch Aufruf von `bin/deviceQuery` und ruft dann `bin/histogram` mit verschiedenen Kommandozeilenargumenten auf, um verschiedene Szenarien zu messen. Insbesondere wird `bin/histogram` angewiesen, pro Szenario und Kernel 100 Messungen vorzunehmen. Neben den der Aufgabenstellung beiliegenden Testdaten umfassen die gemessenen Datengrößen den Bereich von 2^3 = 8 Bytes bis 2^{32} = 4 GiB in Zweierpotenzen. Die Ausführungsdauer des Python-Skripts ist also erheblich. Sollen aus Gründen der Zeitersparnis weniger als 100 Durchläufe je Szenario erfolgen, kann die Konstante `N_RUNS` im Python-Skript entsprechend geändert werden.

Die Geräteinformationen werden als Textdatei, die Messergebnisse im JSON-Format in einem Unterverzeichnis von `measurements/` gespeichert. Das angelegte Unterverzeichnis trägt einen Zeitstempel als Namen.

Einzelne (wiederholte) Messungen können direkt durch Aufruf des Binarys `bin/histogram` mit entsprechenden Kommandozeilenargumenten vorgenommen werden.

Beispiele:

```
bin/histogram ./input_data/test.txt
```

führt mit jedem der vier Kernels eine Messung für die Beispieldatei aus, wobei 128 Bins verwendet werden.

```
bin/histogram -- 10500 asl 7
```

führt für pseudozufällige Daten der Grösse 10500 Bytes mit den Kernels histogram_atomic_private (‚`a`') und histogram_atomic_private_stride (‚`s`') jeweils 7 Messungen aus, wobei 27 Bins (‚`l`') verwendet werden.

Die Usage-Information, die nach Aufruf von `bin/histogram` ohne Argumente erscheint (sie findet sich auch am Anfang von `src/histogram.cu`), erklärt die Bedeutung der möglichen Kommandozeilenargumente.

4 Der Programmcode – histogram.cu

Kernels

Die Kernels sind als Template-Funktionen ausgestaltet. Für den Template-Parameter `Mapping` kann eine `struct` übergeben werden, welche die Zuordnung von Zeichencodes zu Bins definiert. Wir verwenden zwei Mappings, entsprechend den beiden Aufgabenteilen.

Für Aufgabenteil a werden die Zeichencodes 1 bis 128 den Bins 0 bis 127 zugeordnet:

```
struct Mapping128 {
    constexpr static size_t numBins = 128;
    constexpr static __host__ __device__ unsigned char map(
        unsigned char c
    ) {
        return (c - 1u) & 0x7f;
    }
};
```

Für Aufgabenteil b werden die Buchstaben ‚A'/‚a' bis ‚Z'/‚z' (Zeichencodes 65 bzw. 90 bis 97 bzw. 122) den Bins 1 bis 26 zugeordnet; alle übrigen Zeichencodes dem Bin 0:

```
struct MappingLetter {
    constexpr static size_t numBins = 27;
    constexpr static __host__ __device__ unsigned char map(
        unsigned char c
    ) {
        c = (c & 0xdf) - 64u;
```

```
        return c & (0u - (c <= 26u));
    }
};
```

Es werden vier Kernels untersucht.

histogram_kernel_atomic_global („global“)

Die „Baseline“ ist ein einfacher Kernel, bei dem jeder Thread genau ein Zeichen bearbeitet und dann den entsprechenden Bin im globalen Speicher hochzählt:

```
template<typename Mapping>
__global__ void histogram_kernel_atomic_global(
    unsigned char * input, BinType * bins, size_t numElements
) {
    unsigned int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx >= numElements) return;
    unsigned char c = input[idx];
    atomicAdd(&bins[Mapping::map(c)], 1);
}
```

Beim Aufruf des Kernels wird berechnet, wie viele Blöcke benötigt werden, in Abhängigkeit von der Eingabegröße:

```
constexpr size_t nThreadsPerBlock = 256;
// …
dim3 dimGrid(
    (numElements + nThreadsPerBlock - 1) / nThreadsPerBlock, 1, 1
);
dim3 dimBlock(nThreadsPerBlock, 1, 1);

histogram_kernel_atomic_global<Mapping> <<<dimGrid, dimBlock>>> (
    input, bins, numElements
);
```

histogram_kernel_atomic_private („private“)

Der zweite Kernel unterscheidet sich vom ersten nur in der Verwendung von Shared Memory. Jeder Block erhält eine private Instanz des Bin-Arrays im Shared Memory. Diese Instanzen werden abschließend im globalen Speicher aggregiert.

```
template <typename Mapping>
__global__ void histogram_kernel_atomic_private(
    unsigned char * input, BinType * bins, size_t numElements
) {
    unsigned int idx = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ BinType sBins[Mapping::numBins * sizeof(BinType)];
    for (
        unsigned int t = threadIdx.x; t < Mapping::numBins;
        t += blockDim.x
    ) {
        sBins[t] = 0;
    }
    __syncthreads();

    if (idx < numElements) {
        unsigned char c = input[idx];
        atomicAdd(&sBins[Mapping::map(c)], 1);
    }
```

```
    __syncthreads();

    for (
        unsigned int t = threadIdx.x; t < Mapping::numBins;
        t += blockDim.x
    ) {
        atomicAdd(&bins[t], sBins[t]);
    }
}
```

Der Aufruf erfolgt entsprechend wie beim ersten Kernel – es werden so viele Blöcke angefordert, wie für die Eingabe benötigt werden.

histogram_kernel_atomic_private_stride („private_stride“)

Beim dritten Kernel ist die Anzahl der Threads fest, unabhängig von der Eingabegröße. Pro Schritt („stride“) wird ein Block nebeneinanderliegender Zeichen bearbeitet, bis die Eingabe abgearbeitet ist. Es wird wieder mit privaten Instanzen des Bin-Arrays für die einzelnen Blöcke gearbeitet.

```
template <typename Mapping>
__global__ void histogram_kernel_atomic_private_stride(
    unsigned char * input, BinType * bins, size_t numElements
) {
    unsigned int idx = blockIdx.x * blockDim.x + threadIdx.x;

    // initialisiere Array im shared memory mit 0
    __shared__ BinType sBins[Mapping::numBins * sizeof(BinType)];
    for (
        unsigned int t = threadIdx.x; t < Mapping::numBins;
        t += blockDim.x
    ) {
        sBins[t] = 0;
    }
    __syncthreads();

    {
        // baseLimit ist die kleinste Zeichenposition, ab der ein
        // ab baseLimit beginnender Stride genau am letzten Zeichen
        // des Inputs endet oder über den Input hinausragt.
        // Im Prinzip wäre das die letzte Iteration der Schleife. Da
        // allerdings hier durch eine if-Abfrage geprüft werden
        // müsste, ob idx noch innerhalb des Inputs liegt, spart es
        // etwas Zeit, den letzten Stride separat zu behandeln.
        int stride = blockDim.x * gridDim.x;
        size_t baseLimit = numElements >= stride ?
             numElements - stride : 0;
        size_t base = 0;
        for (; base < baseLimit; base += stride) {
             unsigned char c = input[base + idx];
             atomicAdd(&sBins[Mapping::map(c)], 1);
        }
        if (base + idx < numElements) {
             unsigned char c = input[base + idx];
             atomicAdd(&sBins[Mapping::map(c)], 1);
        }
    }
    __syncthreads();
```

```
    for (
        unsigned int t = threadIdx.x; t < Mapping::numBins;
        t += blockDim.x
    ) {
        atomicAdd(&bins[t], sBins[t]);
    }
}
```

Die Anzahl Threads ist nun unabhängig von der Eingabegröße:

```
constexpr size_t nThreadsPerBlock = 256;
// …
dim3 dimGrid(1024, 1, 1);
dim3 dimBlock(nThreadsPerBlock, 1, 1);
histogram_kernel_atomic_private_stride<Mapping> <<<dimGrid, dimBlock>>> (
    input, bins, numElements
);
```

Anmerkung: Die Anzahl Blöcke wurde mittels „Trial and Error“ und aufgrund allgemeiner Empfehlungen aus dem Web als 1024 festgelegt. Die gemessenen Ausführungszeiten scheinen nicht wesentlich von der Anzahl der Blöcke abzuhängen, sofern diese nicht zu klein wird. Bei einer großen Anzahl Blöcke wird der Kernel wohl äquivalent zum Kernel ohne Stride. Die optimalen Parameterwerte sind wahrscheinlich geräteabhängig und nur mittels eines Skripts sinnvoll zu ermitteln.

histogram_kernel_atomic_global_stride („global_stride“)

Der vierte Kernel behält das Stride-Konzept bei, verzichtet jedoch auf die Verwendung von Shared Memory – die Bins werden direkt im globalen Speicher hochgezählt:

```
template <typename Mapping>
__global__ void histogram_kernel_atomic_global_stride(
    unsigned char * input, BinType * bins, size_t numElements
) {
    unsigned int idx = blockIdx.x * blockDim.x + threadIdx.x;

    {
        int stride = blockDim.x * gridDim.x;
        size_t baseLimit = numElements >= stride ?
             numElements - stride : 0;
        size_t base = 0;
        for (; base < baseLimit; base += stride) {
             unsigned char c = input[base + idx];
             atomicAdd(&bins[Mapping::map(c)], 1);
        }
        if (base + idx < numElements) {
             unsigned char c = input[base + idx];
             atomicAdd(&bins[Mapping::map(c)], 1);
        }
    }
}
```

Der Aufruf erfolgt wie beim dritten Kernel – die Anzahl der Threads ist unabhängig von der Eingabegröße.

Der Kernel „global_stride“ wird in diesem Dokument im Folgenden jeweils an zweiter Stelle, direkt nach „global“, aufgeführt, da sich die Messergebnisse von „global“ und „global_stride“ meist kaum unterscheiden.

5 Vorgehen bei den Messungen

Das Programm alloziert zunächst (einmalig) Host- und Device-Speicher. Danach wird für *jeden* der zu messenden Kernels Folgendes durchgeführt:

- Um Verfälschungen der Messungen durch niedrigeren Takt der GPU im Idle-Zustand auszuschließen, werden zunächst 200 Warmup-Runs mit einer Datengröße von 100 MiB durchgeführt, d.h. der jeweilige Kernel wird 200mal aufgerufen.
- Danach wird die mittels Kommandozeile spezifizierte Anzahl von Malen
  - der zu bearbeitende Datenbestand vom Host- in den Device-Speicher transferiert,
  - der Kernel ausgeführt, d.h. das Histogramm erstellt,
  - das Histogramm vom Device- in den Host-Speicher übertragen.

  Die Zeit für jeden dieser drei Schritte wird mittels CUDA-Events gemessen.
- Das Histogramm, das beim letzten Durchlauf des Kernels generiert wurde, wird durch Vergleich mit einem auf der CPU erstellten Histogramm auf Korrektheit geprüft.

Das Programm gibt die Aufrufparameter sowie die Messergebnisse im JSON-Format auf `stdout` aus.

6 Eingabedaten

Das Programm erlaubt sowohl das Einlesen einer Textdatei als auch die Generierung synthetischer Daten beliebiger Größe. Da zu erwarten ist, dass die Laufzeit von der Anzahl auftretender Konflikte beim atomaren Zugriff auf den Speicher abhängt (was sich durch die Messwerte bestätigt), verwenden wir zweierlei synthetische Daten:

- Pseudozufällige Daten (Werte zwischen 1 und 128), die durch einen Lehmer-Zufallszahlengenerator erzeugt werden. Der Seed des Zufallszahlengenerators ist fest, d.h. jeder Durchlauf arbeitet mit den gleichen Daten. Die Implementierung des Zufallszahlengenerators entstammt der Wikipedia.[1] Es ist nicht anzunehmen, dass die Zufallszahlen von guter Qualität sind – allerdings sind sie für den vorliegenden Zweck wohl ausreichend, da es nur darum geht, sehr kurze systematische Muster zu vermeiden. Die Zahlenwerte von 1 bis 128 sind in den Daten annähernd gleichverteilt. Bei 2^{30} generierten Zahlenwerten hat jeder der 128 Zahlenwerte eine relative Häufigkeit von 0.7817 %, und der Unterschied in den relativen Häufigkeiten des seltensten und des häufigsten Zahlenwerts beträgt $9.3 \cdot 10^{-4}$ Prozentpunkte.
- Gleichförmige, d.h. konstante Daten, die nur aus dem Zeichen ‚a' bestehen.

7 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.

Da es einfach durchzuführen war, wurde der CUDA-Code mit dem Tool `hipify-perl` von AMD nach HIP konvertiert und mittels `hipcc` für AMD-GPUs kompiliert. Es waren keine wesentlichen Anpassungen am konvertierten Code erforderlich. Zum Vergleich mit den GPUs von NVIDIA wurden die Messungen dann auch auf folgender Grafikkarte durchgeführt:

- AMD Radeon RX 6800 XT, 16 GB, auf einem AMD Ryzen 9 7950X mit 64 GB Hauptspeicher.

[1] https://en.wikipedia.org/w/index.php?title=Lehmer_random_number_generator&oldid=1260951580#Sample_C99_code

Der Code für AMD ist der Abgabe *nicht* beigefügt, findet sich aber auf Github unter

https://github.com/sbutz/parallel-programming/tree/85bc3324121173e7cfbfa9061a8a3e4b14d53b18/000_johannes/03_amd .

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

8 Messergebnisse für die Eingabedaten aus der Beispieldatei

Die Beispieldatei `./input_data/test.txt` enthält 10532866 Zeichen, ist also rund 10 MiB groß. Abbildung 1 (sowie die Tabelle in Anhang B) zeigen die Häufigkeitsverteilung der einzelnen Zeichencodes.

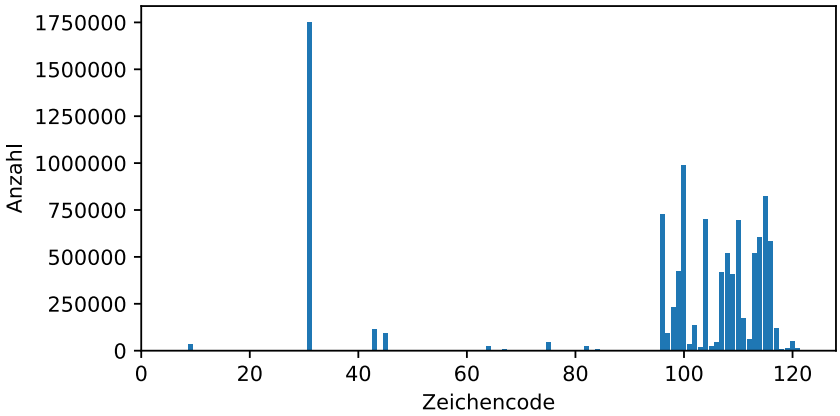

Abbildung 1. Häufigkeiten der einzelnen Zeichencodes in der Beispieldatei.

Die Tabellen in Anhang C enthalten eine Zusammenfassung der Messergebnisse, nämlich:

- die Zeit für den Datentransfer vom Host zum Device,
- die Ausführungszeit auf dem Device,
- die Zeit für den Datentransfer vom Device zum Host,
- die Summe dieser drei Zeiten („Gesamtzeit“).

Es sind jeweils die minimale Zeit, das 10%-Quantil, der Median, das 90%-Quantil und die maximale Zeit angegeben.

Es fällt auf, dass die Transferzeiten teilweise eine recht große Variation aufweisen. Abbildung 2 zeigt den zeitlichen Verlauf der Zeiten für den Transfer vom Host zum Device über die jeweils 100 Wiederholungen.

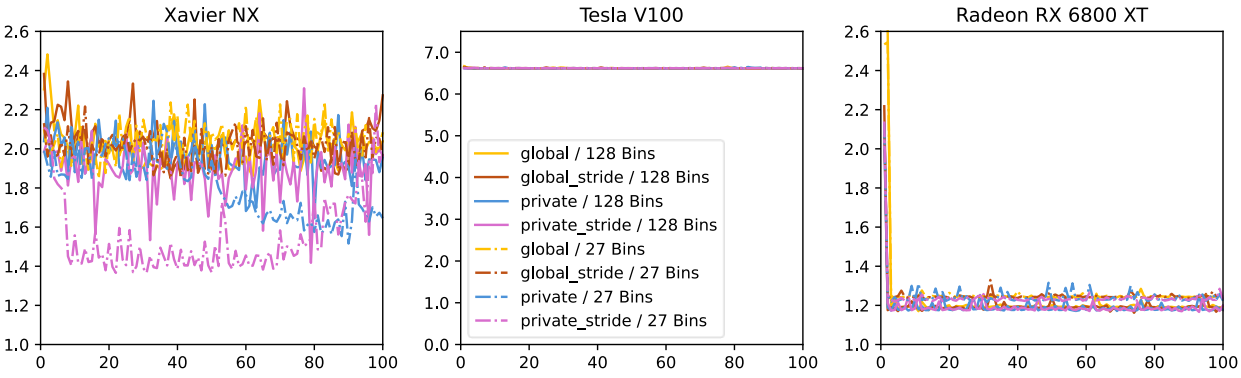

Abbildung 2. Zeiten für den Transfer der Eingabedaten aus der Beispieldatei vom Host zum Device. Die horizontale Achse zeigt den zeitlichen Verlauf über die jeweils 100 Wiederholungen. Auf der vertikalen Achse ist jeweils die gemessene Transferzeit in Millisekunden (ms) abgetragen. Man beachte die unterschiedliche Skalierung der vertikalen Achsen in den drei Diagrammen.

Während die Transferzeiten auf das Tesla-Gerät sehr konstant sind, erkennt man bei den anderen beiden Geräten erhebliche Fluktuationen. Beim Radeon-Gerät fällt auf, dass der jeweils erste Transfer weitaus länger dauert als die folgenden. Die Transfers dauern generell bei den Varianten mit 27 Bins (oberes Liniencluster im Diagramm) länger als bei den Varianten mit 128 Bins (unteres Liniencluster im Diagramm), obwohl die gleichen Daten übertragen werden und die Bins an dieser Stelle des Programms noch keine Rolle spielen sollten. Generell scheint sowohl beim Xavier-Gerät als auch beim Radeon-Gerät der Transfer für einige Kernels länger zu dauern als für andere, wenngleich dies bei beiden Geräten nicht für die gleichen Kernels der Fall ist. Die Fluktuationen scheinen aber nicht rein zufällig zu sein. Wir haben für dieses Verhalten keine plausible Erklärung. Möglicherweise hängt es mit Compileroptimierungen oder Cache-Effekten zusammen. Es ist jedenfalls compiler- oder hardwareabhängig. Es wurde *nicht* geprüft, ob sich das Verhalten mit Änderungen am Programmcode, insbesondere einer anderen Ausführungsreihenfolge der Kernels, ändert.

In Abbildung 3 sind der Vollständigkeit halber die Zeiten für den Transfer vom Device zum Host dargestellt. Aufgrund der geringen Datenmenge (128 bzw. 27 Werte vom Typ `unsigned int`) sind die Messwerte wohl wenig aussagekräftig und die Schwankungen als „Noise“ zu betrachten.

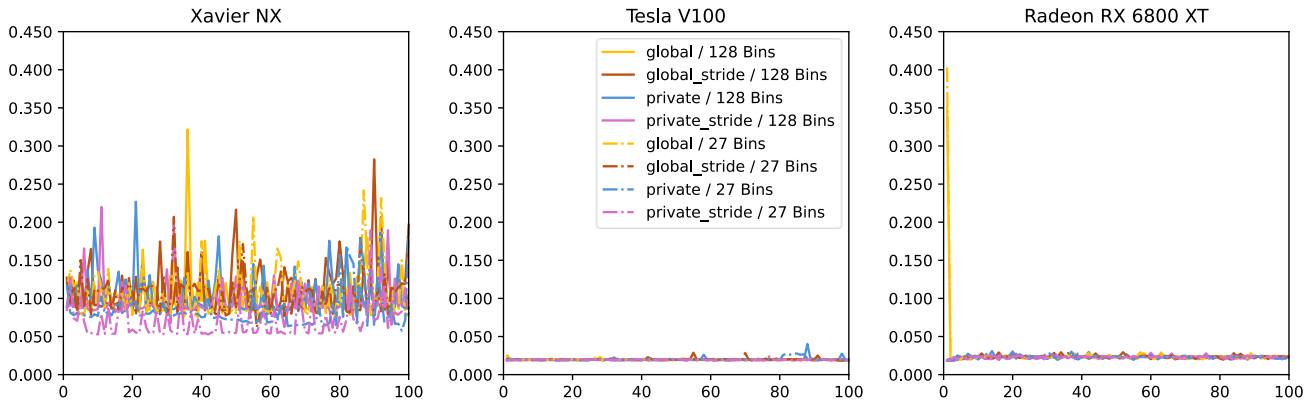

Abbildung 3. Zeiten für den Transfer der Resultate vom Device zum Host. Die horizontale Achse zeigt den zeitlichen Verlauf über die jeweils 100 Wiederholungen. Auf der vertikalen Achse ist jeweils die gemessene Transferzeit in Millisekunden (ms) abgetragen.

Wir kommen nun zur Analyse der Ausführungszeiten. Abbildung 4 zeigt, dass die Ausführungszeiten im Laufe der 100 Wiederholungen jeweils sehr konstant sind; wir können also beim Vergleich der Kernels und Mappings mit den Medianwerten arbeiten.

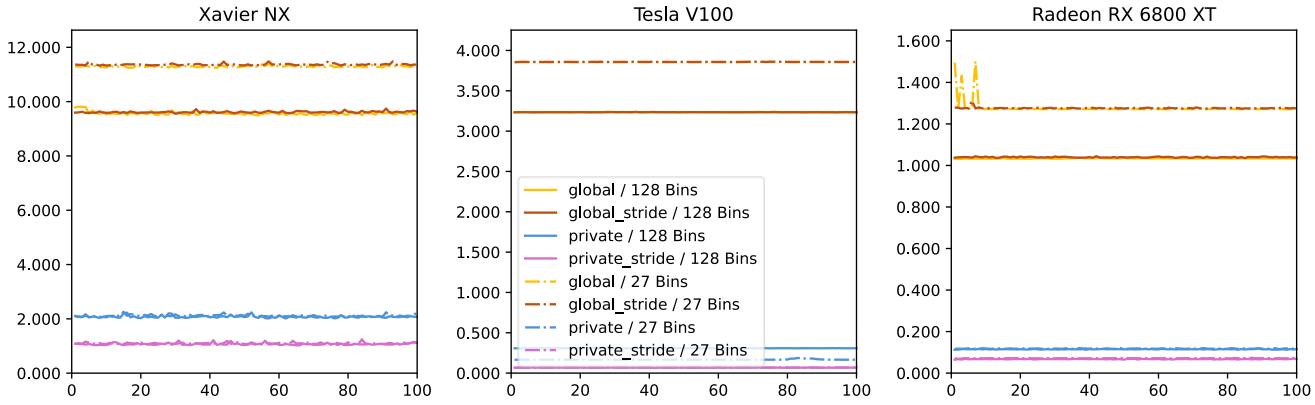

Abbildung 4. Ausführungszeiten auf dem Device für die Eingabedaten aus der Beispieldatei. Die horizontale Achse zeigt den zeitlichen Verlauf über die jeweils 100 Wiederholungen. Auf der vertikalen Achse ist jeweils die gemessene Transferzeit in Millisekunden (ms) abgetragen. Man beachte die unterschiedliche Skalierung der vertikalen Achsen.

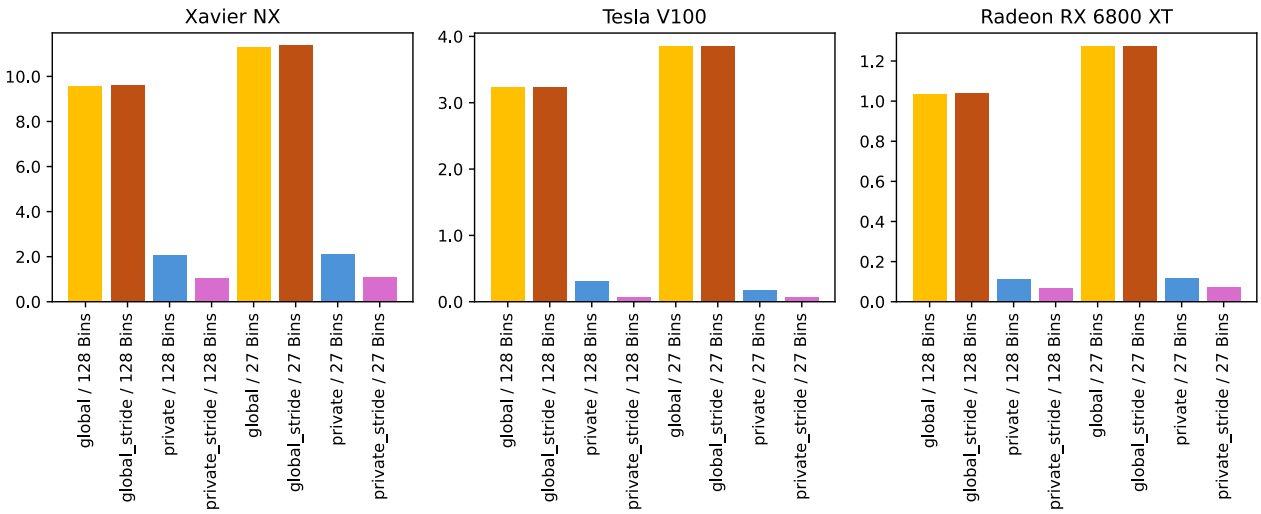

Abbildung 5. Median der Ausführungszeit (in ms). Man beachte die unterschiedliche Skalierung der vertikalen Achsen.

Abbildung 5 zeigt die Mediane der Ausführungszeiten. Es fällt zweierlei auf.

Zum einen: Während bei ausschließlicher Verwendung des globalen Speichers (Kernels „global“ und „global_stride“) der Stride keinen nennenswerten Unterschied macht, führt bei Verwendung von Shared Memory (Kernels „private“ und „private_stride“) der Stride zu einem deutlichen Zeitvorteil. Die Tabellen Tabelle 1 und Tabelle 2 zeigen den Speedup im Vergleich zum Kernel „global“ für 128 bzw. 27 Bins. Die uns am wahrscheinlichsten scheinende Erklärung für die Überlegenheit des Stride bei Verwendung von Shared Memory ist der verringerte Zugriff auf den globalen Speicher bei der abschließenden Aggregation durch die niedrigere Anzahl an Blöcken ((10532866 + 255) / 256 = 41145 Blöcke ohne Stride vs. 1024 Blöcke mit Stride).

Zum anderen: Bei ausschließlicher Verwendung globalen Speichers führt eine höhere Anzahl von Bins zu einem Zeitvorteil. Dieser Effekt ist bei Verwendung von Shared Memory nicht zu sehen. Tabelle 3 zeigt den Faktor, um den die Ausführungszeit bei 27 Bins gegenüber 128 Bins höher ist. Eine mögliche Erklärung für dieses Laufzeitverhalten wäre, dass bei einer höheren Anzahl von Bins seltener Kollisionen auftreten, d.h. verschiedene Threads gleichzeitig denselben Bin hochzählen möchten. Bei Verwendung von Shared Memory treten solche Kollisionen prinzipbedingt seltener auf. Auffällig ist allerdings der Wert von 0.54 für den Kernel „private“ auf dem Tesla-Gerät, d.h. hier ist die Ausführungszeit bei 27 Bins trotz Shared Memory *wesentlich höher* als bei 128 Bins. Die Ursache ist unklar. Bei der Suche nach einer Erklärung wäre vielleicht zu fragen, inwiefern die Tatsache eine Rolle spielt, dass die Anzahl Bins im einen Fall mit 128 eine Zweierpotenz ist und im anderen Fall mit 27 nicht. Es wäre denkbar, dass die Zahl 128 bei der Aggregation vom Shared Memory in den globalen Speicher zu ungünstigen Zugriffsmustern führt und diese aus einem unbekannten Grund gerade auf dem Tesla-Gerät einen großen Einfluss auf die Laufzeit haben. Das ist aber Spekulation.

	Speedup gegenüber „global“, 128 Bins			
Device	**global**	**global_stride**	**private**	**private_stride**
Xavier NX	1.00	1.00	4.62	9.04
Tesla V100	1.00	1.00	10.50	47.81
Radeon RX 6800 XT	1.00	0.99	9.02	15.38

Tabelle 1. Speedup der untersuchten Kernels gegenüber dem Kernel „global“ für die Eingabedaten aus der Beispieldatei bei Verwendung von 128 Bins.

	Speedup gegenüber „global“, 27 Bins			
Device	**global**	**global_stride**	**private**	**private_stride**
Xavier NX	1.00	0.99	5.35	10.35
Tesla V100	1.00	1.00	23.19	56.35
Radeon RX 6800 XT	1.00	1.00	10.80	17.77

Tabelle 2. Speedup der untersuchten Kernels gegenüber dem Kernel „global“ für die Eingabedaten aus der Beispieldatei bei Verwendung von 27 Bins.

	Ausführungszeit bei 27 Bins im Verhältnis zur Ausführungszeit bei 128 Bins			
Device	**global**	**global_stride**	**private**	**private_stride**
Xavier NX	1.18	1.18	1.02	1.03
Tesla V100	1.19	1.19	0.54	1.01
Radeon RX 6800 XT	1.23	1.23	1.03	1.06

Tabelle 3. Ausführungszeit bei 27 Bins im Verhältnis zur Ausführungszeit bei 128 Bins für die verschiedenen Kernels und Geräte.

9 Messergebnisse für unterschiedliche Eingabegrößen

Um zu untersuchen, wie die Ausführungszeiten von der Größe der Eingabe abhängen, haben wir die Laufzeiten auch mit generierten Daten gemessen. Da zu vermuten ist, dass der Umfang der Kollisionen (d.h. wie viele Threads auf den gleichen Bin zu schreiben versuchen) einen Einfluss auf die Ausführungszeit hat, haben wir, wie oben beschrieben, zum einen pseudozufällige Daten verwendet, bei denen jeder Zeichencode zwischen 1 und 128 annähernd gleich häufig vorkommt, und zum anderen konstante Daten, welche nur aus dem Buchstaben ‚a‘ bestehen.

Es wurden als Eingabegrößen alle Zweierpotenzen zwischen 2^3 = 8 Bytes und 2^{32} = 4 GiB untersucht. Bei 2^{32} Bytes überschreitet bei den Kernels ohne Stride die Anzahl der nötigen Threads die Anzahl möglicher Threads, so dass für diese beiden Kernels und diese Eingabegröße keine Messwerte existieren. Anhang D enthält eine Auswahl an Zahlenwerten. Wir beschränken uns hier auf die graphische Darstellung.

Abbildung 6 zeigt die Ausführungszeiten in Abhängigkeit von der Eingabegröße. Die durchgezogenen Kurven repräsentieren jeweils den Median, die (aufgrund der meist geringen Schwankungen kaum sichtbaren) gestrichelten Kurven das erste und letzte Dezil. Abbildung 7 zeigt den Speedup der einzelnen Kernels im Vergleich zum Kernel „global“, wieder in Abhängigkeit von der Eingabegröße.

Es fällt auf:

- Für pseudozufällige Eingabedaten ergeben sich meist geringere Laufzeiten als für konstante Eingabedaten. Dieser Effekt ist bei 128 Bins stärker ausgeprägt als bei 27 Bins. Eine plausible Erklärung ist wieder, dass Kollisionen einen negativen Effekt auf die Laufzeit haben.
- Für große Eingaben ist der Kernel „private_stride“ den anderen Kernels überlegen. Als Grund vermuten wir wieder, dass dieser Kernel aufgrund der konstanten Anzahl an Blöcken die Zugriffe auf den globalen Speicher minimiert.
- Für kleine Eingaben ist die Laufzeit bei allen Kernels zunächst über einen gewissen Bereich nahezu konstant. Dies ist wohl so lange der Fall, wie die Hardware die Ausführung noch perfekt parallelisieren kann.
- Der Kernel „atomic_private_stride“ ist für kleine Eingaben den übrigen Kernels unterlegen, wobei der Effekt auf unterschiedlichen Geräten unterschiedlich ausgeprägt ist. Verantwortlich für die Unterlegenheit ist wohl die konstante Anzahl an Blöcken, welche für kleine Eingaben größer als optimal ist. Dieser Effekt tritt jedoch beim Kernel „global_stride“ nicht auf. Betrachtet man den Programmcode, fällt auf, dass beim Kernel „private_stride“ auch die „überflüssigen“ Blöcke an der abschließenden Aggregation teilnehmen. Es wäre zu untersuchen, ob sich eine Laufzeitverbesserung ergäbe, wenn man explizit prüft, ob Wert in der privaten Instanz im Shared Memory nicht 0 ist, bevor man die atomare Addition in den globalen Speicher ausführt. Andererseits

erscheint die Tatsache merkwürdig, dass der Compiler eine solche Optimierung nicht selbst vornähme, wenn sie sinnvoll wäre.

Abbildung 8 zeigt das Verhältnis der Ausführungsdauer für 27 Bins zur Ausführungsdauer bei 128 Bins. Man beobachtet bei kleinen Datenmengen für alle Kernels ungefähr ein Verhältnis von 1. Für große Datenmengen sieht man Folgendes:

- Bei ausschließlicher Verwendung globalen Speichers *und pseudozufälligen Daten* führt eine höhere Anzahl Bins zu einem Zeitvorteil, jedoch *nicht bei konstanten Daten*.
- Bei Verwendung von Shared Memory *und pseudozufälligen Daten* führt eine höhere Anzahl Bins (mit einer Ausnahme, siehe nächster Punkt) ebenfalls zu einem Zeitvorteil, jedoch *nicht bei konstanten Daten*.
- Auf dem Tesla-Gerät sind mit dem Kernel „private“ 27 Bins *günstiger* als 128 Bins, und zwar sowohl für pseudozufällige als auch für konstante Daten.

Diese Beobachtungen sind ist aus unserer Sicht konsistent mit denjenigen, die wir für die Eingabedaten aus der Beispieldatei gemacht haben. Als vorsichtiges Fazit können wir festhalten, dass sich Kollisionen sowohl im globalen Speicher (das zeigt der erste Punkt) als auch im geteilten Speicher (das zeigt der zweite Punkt) negativ auf das Laufzeitverhalten auswirken.

Die Ursache für die Beobachtung beim dritten Punkt kann nicht an Kollisionen beim Zugriff auf den Shared Memory liegen, denn wir bekommen für pseudozufällige und für konstante Daten das gleiche Ergebnis. Die Beobachtung muss also höchstwahrscheinlich durch die abschließende Aggregation verursacht werden. Natürlich sind 27 Bins günstiger zu aggregieren als 128, aber es überrascht, dass der Effekt nur auf dem Tesla-Gerät auftritt. Allenfalls sind, wie oben bereits spekuliert wurde, auch die Zugriffsmuster auf den globalen Speicher unterschiedlich. Eine überzeugende Erklärung fehlt.

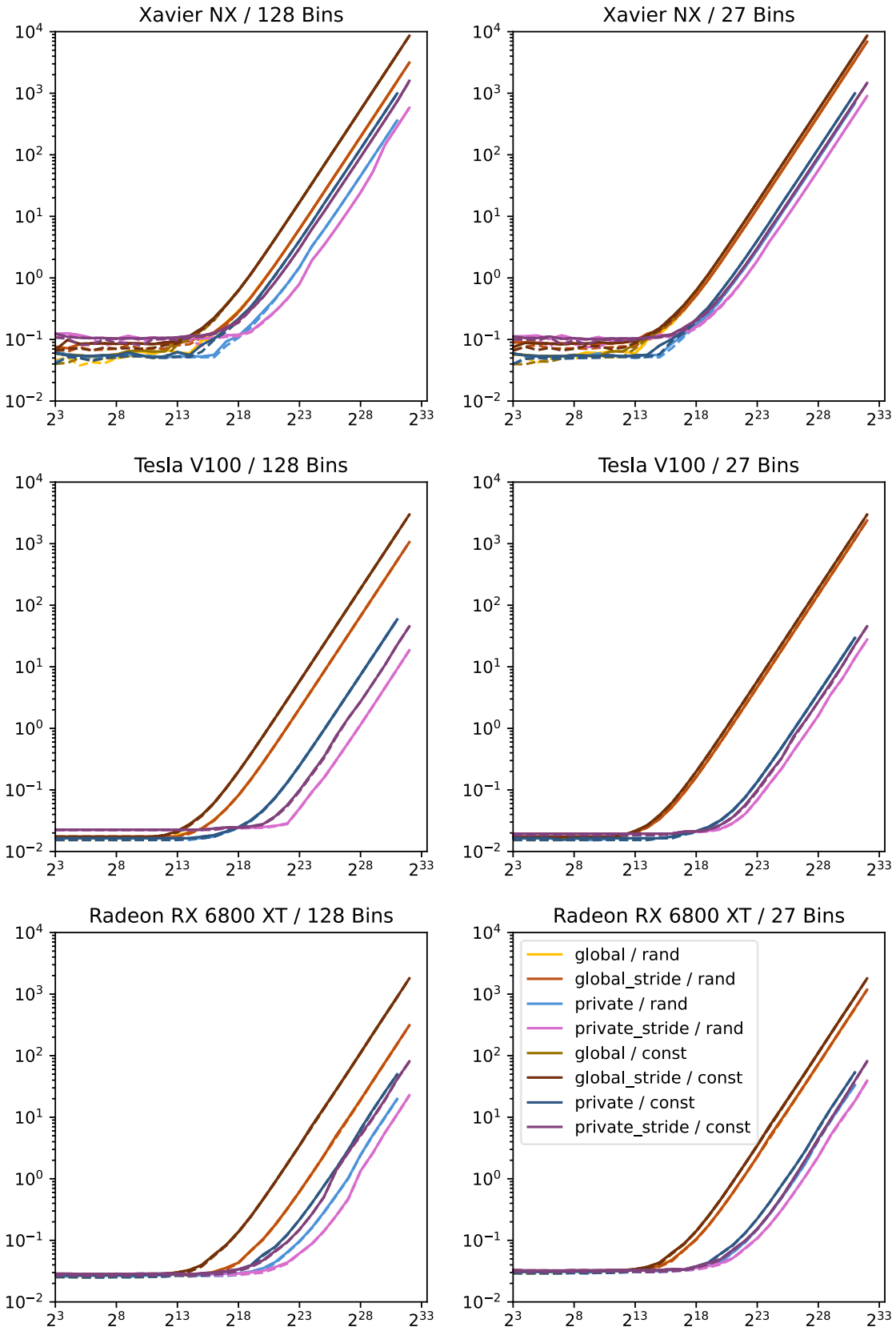

Abbildung 6. Ausführungszeiten (in ms, vertikale Achse) in Abhängigkeit von der Eingabegröße (horizontale Achse). Die helleren Kurven („rand“) repräsentieren die Zeiten bei pseudozufälligen Eingabedaten, die dunkleren Kurven („const“) bei konstanten Eingabedaten. Die Kurven für den Kernel „global“ werden weitestgehend durch die Kurven für „global_stride“ verdeckt. Die durchgezogenen Kurven repräsentieren jeweils den Median. Das erste und letzte Dezil sind jeweils gestrichelt eingezeichnet, aber auch diese Kurven werden weitestgehend verdeckt.

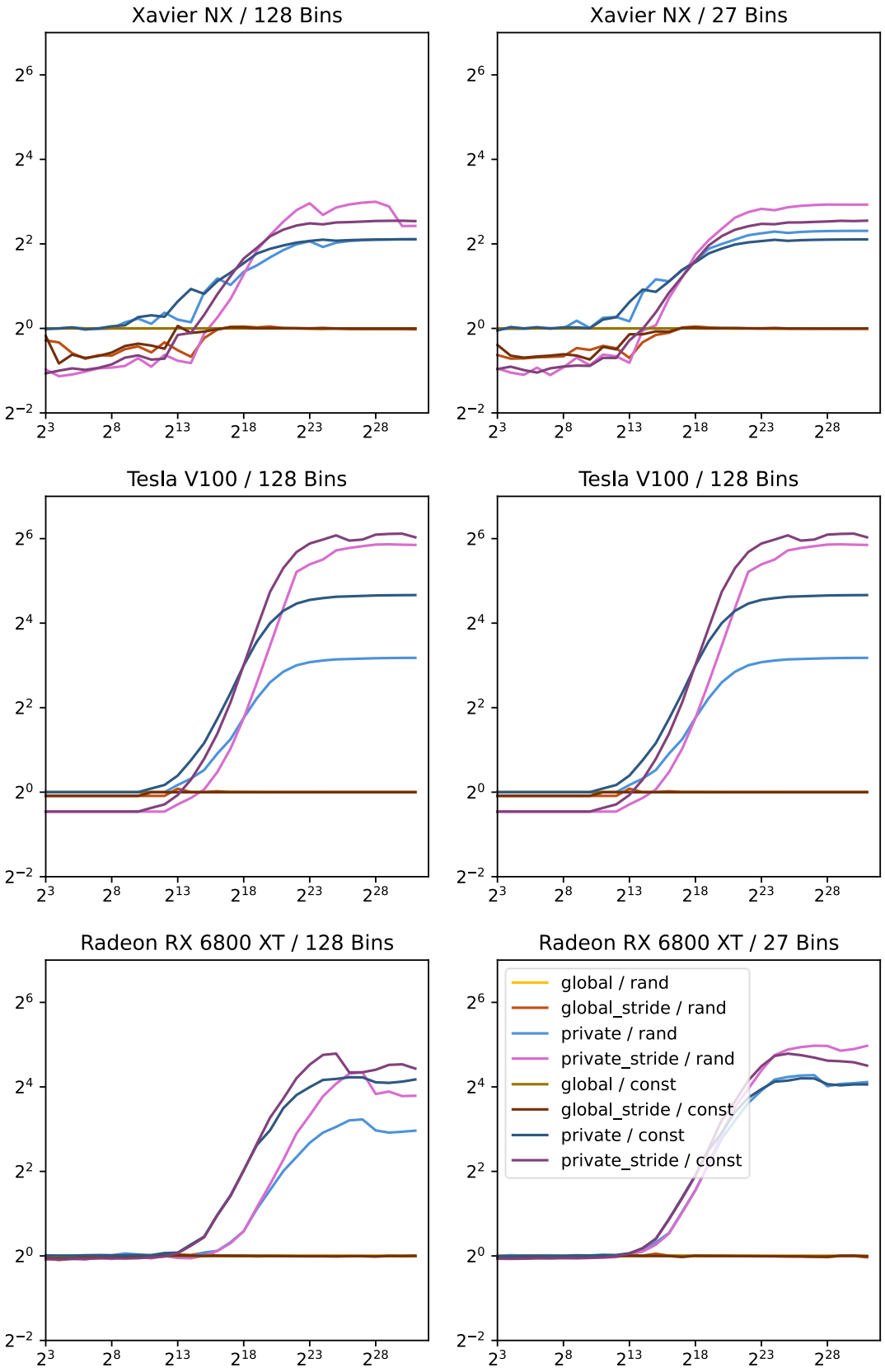

Abbildung 7. Speedup (vertikale Achse) gegenüber dem Kernel „global“ in Abhängigkeit von der Eingabegröße (horizontale Achse). Die helleren Kurven („rand“) repräsentieren die Ergebnisse bei pseudozufälligen Eingabedaten, die dunkleren Kurven („const“) bei konstanten Eingabedaten.

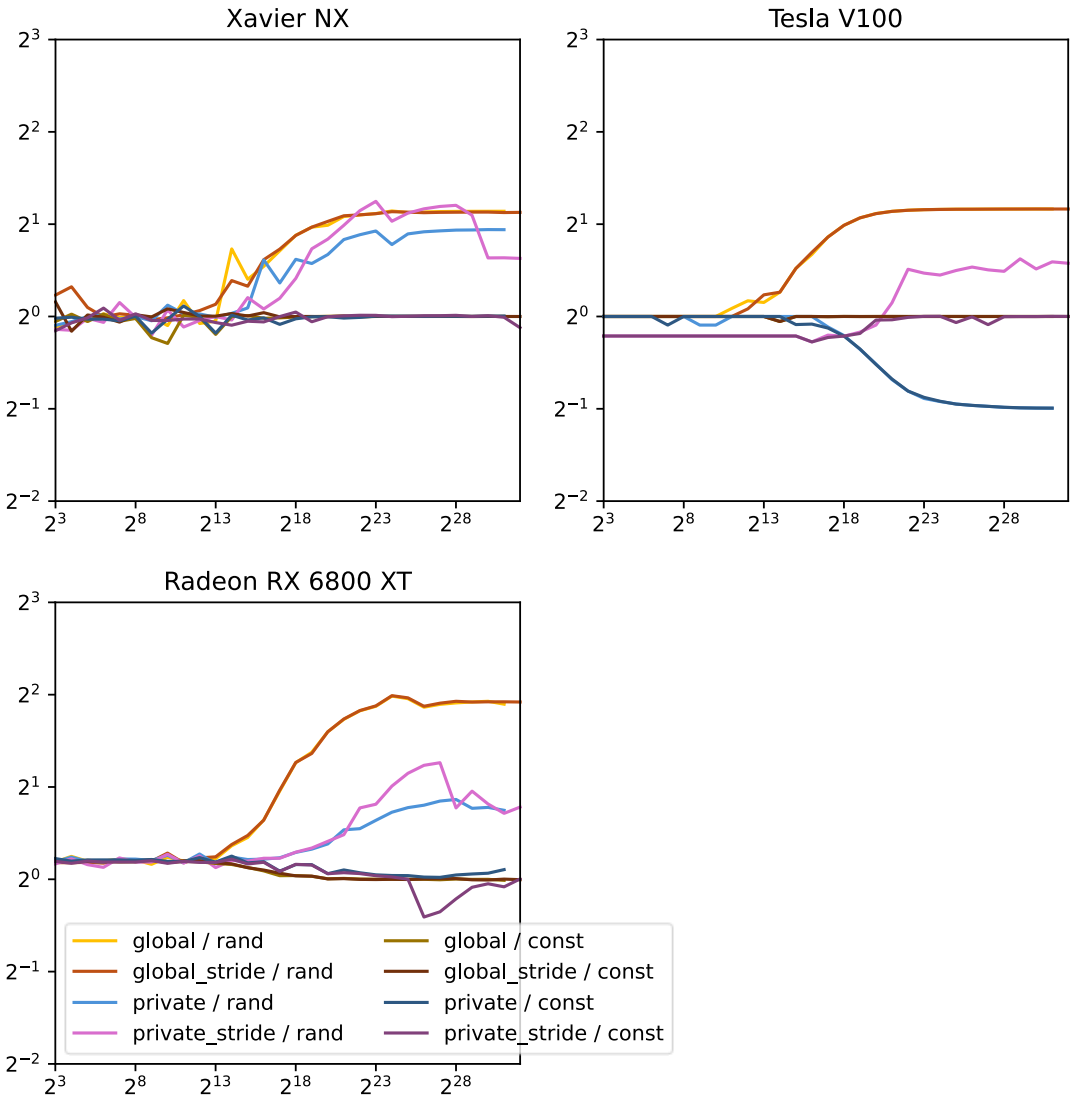

Abbildung 8. Verhältnis der Ausführungsdauer bei 27 Bins zur Ausführungsdauer bei 128 Bins in Abhängigkeit von der Eingabegröße. Der Quotient ist auf der vertikalen Achse aufgetragen, die Eingabegröße auf der horizontalen Achse. Die helleren Kurven („rand“) repräsentieren die Ergebnisse bei pseudozufälligen Eingabedaten, die dunkleren Kurven („const“) bei konstanten Eingabedaten.

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
```

AMD Radeon RX 6800 XT

[…]

```
Device 0: "AMD Radeon RX 6800 XT"
  HIP Driver Version / Runtime Version          50422.0 / 50422.0
  HIP Capability Major/Minor version number:    10.3
  Total amount of global memory:                 16368 MBytes (17163091968
bytes)
MapSMtoCores for SM 10.3 is undefined.  Default to use 64 Cores/SM
MapSMtoCores for SM 10.3 is undefined.  Default to use 64 Cores/SM
  (36) Multiprocessors, ( 64) HIP Cores/MP:     2304 HIP Cores
  GPU Max Clock rate:                            2575 MHz (2.58 GHz)
  Memory Clock rate:                             1000 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 4194304 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(16384), 2D=(16384,
16384), 3D=(16384, 16384, 8192)
  Total amount of constant memory:               2147483647 bytes
  Total amount of shared memory per block:       65536 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, 1024)
  Max dimension size of a grid size    (x,y,z): (2147483647, 2147483647,
2147483647)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             256 bytes
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Device has ECC support:                        Disabled
  Supports Cooperative Kernel Launch:            No
  Supports MultiDevice Co-op Kernel Launch:      No
  Device PCI Domain ID / Bus ID / location ID:   0 / 3 / 0
  Compute Mode:
     < Default (multiple host threads can use ::hipSetDevice() with device
simultaneously) >
```

[…]

```
deviceQuery, HIP Driver = HIPRT, HIP Driver Version = 50422.0, HIP Runtime
Version = 50422.0, NumDevs = 2
Result = PASS
```

Anhang B. Häufigkeit der einzelnen Zeichencodes in der Beispieldatei

Code	Anzahl	relative Häufigkeit
10	37022	0.351 %
32 (, ‘)	1749664	16.611 %
44 (,,‘)	114062	1.083 %
46 (,.‘)	92662	0.880 %
65 (‚A‘)	23754	0.226 %
67 (‚C‘)	2568	0.024 %
68 (‚D‘)	8132	0.077 %
76 (‚L‘)	47722	0.453 %
78 (‚N‘)	2568	0.024 %
83 (‚S‘)	23754	0.226 %
85 (‚U‘)	5350	0.051 %
97 (‚a‘)	725674	6.890 %
98 (‚b‘)	92662	0.880 %
99 (‚c‘)	231334	2.196 %
100 (‚d‘)	422650	4.013 %
101 (‚e‘)	987396	9.374 %
102 (‚f‘)	32528	0.309 %
103 (‚g‘)	135676	1.288 %
104 (‚h‘)	16050	0.152 %
105 (‚i‘)	702348	6.668 %
106 (‚j‘)	23754	0.226 %
107 (‚k‘)	47508	0.451 %
108 (‚l‘)	419868	3.986 %
109 (‚m‘)	520020	4.937 %
110 (‚n‘)	407242	3.866 %
111 (‚o‘)	693360	6.583 %
112 (‚p‘)	175266	1.664 %
113 (‚q‘)	61418	0.583 %
114 (‚r‘)	518094	4.919 %
115 (‚s‘)	603908	5.734 %
116 (‚t‘)	824542	7.828 %
117 (‚u‘)	584220	5.547 %
118 (‚v‘)	117486	1.115 %
119 (‚w‘)	5350	0.051 %
120 (‚x‘)	10700	0.102 %
121 (‚y‘)	52858	0.502 %
122 (‚z‘)	13696	0.130 %
Summe	**10532866**	**100.000 %**

Anhang C. Gemessene Zeiten für die Eingabedaten aus der Beispieldatei

		128 Bins																			
		Transferzeit Host zu Device (ms)					Ausführungszeit (ms)					Transferzeit Device zu Host (ms)					Gesamtzeit (ms)				
Device	**Kernel**	**Min**	**Q10**	**Med**	**Q90**	**Max**	**Min**	**Q10**	**Med**	**Q90**	**Max**	**Min**	**Q10**	**Med**	**Q90**	**Max**	**Min**	**Q10**	**Med**	**Q90**	**Max**
Xavier NX	global	1.860	1.904	2.016	2.016	2.483	9.493	9.516	9.561	9.561	9.814	0.068	0.085	0.092	0.092	0.322	11.452	11.558	11.674	11.674	12.389
	global_stride	1.849	1.895	2.044	2.044	2.383	9.555	9.586	9.606	9.606	9.745	0.081	0.084	0.096	0.096	0.283	11.562	11.604	11.756	11.756	12.148
	private	1.644	1.867	1.934	1.934	2.245	2.012	2.037	2.069	2.069	2.240	0.075	0.085	0.091	0.091	0.227	3.794	4.010	4.129	4.129	4.400
	private_stride	1.417	1.737	1.916	1.916	2.309	1.023	1.038	1.058	1.058	1.217	0.061	0.085	0.090	0.090	0.220	2.536	2.889	3.075	3.075	3.434
Tesla V100	global	6.615	6.615	6.616	6.616	6.666	3.232	3.233	3.234	3.234	3.242	0.019	0.019	0.019	0.019	0.024	9.867	9.868	9.870	9.870	9.922
	global_stride	6.615	6.616	6.617	6.617	6.641	3.232	3.234	3.234	3.234	3.239	0.018	0.019	0.020	0.020	0.029	9.868	9.869	9.871	9.871	9.894
	private	6.615	6.615	6.616	6.616	6.640	0.306	0.307	0.308	0.308	0.310	0.019	0.019	0.019	0.019	0.040	6.940	6.942	6.944	6.944	6.968
	private_stride	6.615	6.616	6.616	6.616	6.620	0.065	0.066	0.068	0.068	0.075	0.018	0.019	0.019	0.019	0.021	6.701	6.702	6.703	6.703	6.710
Radeon	global	1.170	1.184	1.194	1.194	11.573	1.032	1.033	1.034	1.034	1.039	0.019	0.021	0.023	0.023	0.351	2.226	2.242	2.251	2.251	12.957
RX 6800 XT	global_stride	1.162	1.178	1.190	1.190	2.217	1.035	1.037	1.040	1.040	1.045	0.018	0.020	0.023	0.023	0.029	2.225	2.239	2.252	2.252	3.273
	private	1.171	1.175	1.179	1.179	2.088	0.112	0.114	0.115	0.115	0.118	0.018	0.020	0.023	0.023	0.030	1.307	1.312	1.317	1.317	2.219
	private_stride	1.166	1.174	1.181	1.181	2.060	0.065	0.066	0.067	0.067	0.070	0.018	0.020	0.023	0.023	0.029	1.255	1.262	1.271	1.271	2.143

		27 Bins																			
		Transferzeit Host zu Device (ms)					Ausführungszeit (ms)					Transferzeit Device zu Host (ms)					Gesamtzeit (ms)				
Device	**Kernel**	**Min**	**Q10**	**Med**	**Q90**	**Max**	**Min**	**Q10**	**Med**	**Q90**	**Max**	**Min**	**Q10**	**Med**	**Q90**	**Max**	**Min**	**Q10**	**Med**	**Q90**	**Max**
Xavier NX	global	1.850	1.937	2.045	2.045	2.240	11.243	11.266	11.289	11.289	11.403	0.076	0.078	0.093	0.093	0.242	13.200	13.323	13.440	13.440	13.684
	global_stride	1.851	1.897	2.006	2.006	2.216	11.324	11.338	11.362	11.362	11.485	0.062	0.079	0.111	0.111	0.207	13.283	13.361	13.477	13.477	13.701
	private	1.516	1.623	1.840	1.840	2.128	2.059	2.076	2.110	2.110	2.268	0.058	0.065	0.078	0.078	0.190	3.650	3.795	4.010	4.010	4.326
	private_stride	1.366	1.390	1.478	1.478	2.230	1.055	1.071	1.091	1.091	1.247	0.053	0.054	0.057	0.057	0.190	2.489	2.538	2.640	2.640	3.421
Tesla V100	global	6.611	6.617	6.619	6.619	6.667	3.854	3.855	3.856	3.856	3.863	0.018	0.019	0.020	0.020	0.026	10.486	10.493	10.494	10.494	10.548
	global_stride	6.616	6.617	6.619	6.619	6.648	3.855	3.857	3.858	3.858	3.866	0.018	0.019	0.020	0.020	0.028	10.493	10.494	10.497	10.497	10.525
	private	6.617	6.617	6.619	6.619	6.648	0.165	0.166	0.166	0.166	0.188	0.018	0.019	0.020	0.020	0.028	6.801	6.803	6.805	6.805	6.857
	private_stride	6.616	6.617	6.619	6.619	6.632	0.066	0.067	0.068	0.068	0.078	0.018	0.019	0.020	0.020	0.024	6.703	6.705	6.706	6.706	6.722
Radeon	global	1.237	1.240	1.245	1.245	2.544	1.270	1.271	1.272	1.272	1.502	0.019	0.021	0.024	0.024	0.403	2.531	2.535	2.541	2.541	4.432
RX 6800 XT	global_stride	1.227	1.239	1.244	1.244	2.128	1.273	1.274	1.275	1.275	1.312	0.019	0.022	0.024	0.024	0.029	2.525	2.539	2.544	2.544	3.426
	private	1.216	1.224	1.230	1.230	2.132	0.115	0.117	0.118	0.118	0.121	0.020	0.021	0.023	0.023	0.031	1.355	1.366	1.372	1.372	2.268
	private_stride	1.219	1.225	1.231	1.231	2.140	0.069	0.071	0.072	0.072	0.074	0.019	0.021	0.023	0.023	0.029	1.312	1.319	1.326	1.326	2.230

Anhang D. Gemessene Zeiten für die generierten Eingabedaten

Es ist n := $\log_2$(Eingabegröße). Dargestellt sind nur die Messergebnisse für n = 8, 16, 24, 32. Auf die Darstellung der Zeiten für den Transfer vom Device zum Host wurde verzichtet.

Xavier NX, pseudozufällige Daten, 128 Bins

		Transferzeit Host zu Device (ms)					Ausführungszeit (ms)					Gesamtzeit (ms)				
n	**Kernel**	**Min**	**Q10**	**Med**	**Q90**	**Max**	**Min**	**Q10**	**Med**	**Q90**	**Max**	**Min**	**Q10**	**Med**	**Q90**	**Max**
19	atomic_global	0.129	0.134	0.138	0.138	0.242	0.473	0.479	0.486	0.486	0.566	0.664	0.675	0.701	0.701	0.855
	atomic_global_stride	0.129	0.138	0.141	0.141	0.208	0.467	0.475	0.479	0.479	0.548	0.651	0.677	0.686	0.686	0.795
	atomic_private	0.102	0.104	0.128	0.128	0.226	0.150	0.155	0.173	0.173	0.269	0.293	0.300	0.353	0.353	0.517
	atomic_private_stride	0.101	0.104	0.106	0.106	0.208	0.122	0.131	0.135	0.135	0.199	0.274	0.277	0.299	0.299	0.470
23	atomic_global	1.444	1.462	1.542	1.542	1.907	6.192	6.208	6.237	6.237	6.313	7.733	7.784	7.857	7.857	8.312
	atomic_global_stride	1.419	1.440	1.483	1.483	1.707	6.243	6.249	6.264	6.264	6.419	7.738	7.769	7.843	7.843	8.069
	atomic_private	1.381	1.441	1.484	1.484	1.741	1.465	1.475	1.494	1.494	1.662	2.943	3.002	3.063	3.063	3.393
	atomic_private_stride	1.062	1.078	1.156	1.156	1.636	0.777	0.794	0.801	0.801	0.882	1.906	1.925	2.010	2.010	2.504
27	atomic_global	19.748	19.941	20.273	20.273	21.811	97.155	97.362	97.382	97.382	97.473	117.203	117.445	117.768	117.768	119.167
	atomic_global_stride	19.120	19.850	20.152	20.152	21.033	98.090	98.305	98.472	98.472	98.778	117.939	118.339	118.700	118.700	119.500
	atomic_private	19.356	19.512	19.814	19.814	20.593	22.875	22.887	22.910	22.910	22.984	42.328	42.511	42.836	42.836	43.596
	atomic_private_stride	18.230	18.459	19.033	19.033	20.371	12.334	12.360	12.383	12.383	12.429	30.688	30.910	31.515	31.515	32.838
31	atomic_global	280.124	283.486	284.034	284.034	312.136	1551.195	1551.202	1551.224	1551.224	1551.333	1831.579	1834.810	1835.375	1835.375	1863.532
	atomic_global_stride	282.634	282.966	283.765	283.765	288.764	1568.555	1572.247	1575.216	1575.216	1577.995	1852.581	1856.092	1858.882	1858.882	1865.329
	atomic_private	279.885	281.492	282.022	282.022	288.330	359.424	359.514	359.563	359.563	359.664	639.571	641.155	641.704	641.704	648.006
	atomic_private_stride	280.958	281.334	281.864	281.864	286.805	288.582	288.768	288.835	288.835	288.935	569.872	570.233	570.821	570.821	575.822

Xavier NX, konstante Daten, 128 Bins

		Transferzeit Host zu Device (ms)					Ausführungszeit (ms)					Gesamtzeit (ms)				
n	**Kernel**	**Min**	**Q10**	**Med**	**Q90**	**Max**	**Min**	**Q10**	**Med**	**Q90**	**Max**	**Min**	**Q10**	**Med**	**Q90**	**Max**
19	atomic_global	0.129	0.131	0.133	0.133	0.213	1.141	1.144	1.150	1.150	1.228	1.337	1.341	1.361	1.361	1.472
	atomic_global_stride	0.130	0.132	0.134	0.134	0.208	1.139	1.143	1.146	1.146	1.264	1.336	1.339	1.346	1.346	1.463
	atomic_private	0.104	0.111	0.131	0.131	0.171	0.313	0.322	0.337	0.337	0.401	0.473	0.492	0.529	0.529	0.603
	atomic_private_stride	0.102	0.104	0.115	0.115	0.202	0.292	0.298	0.307	0.307	0.390	0.436	0.446	0.474	0.474	0.646
23	atomic_global	1.464	1.481	1.519	1.519	1.875	16.863	16.874	16.907	16.907	17.186	18.412	18.439	18.528	18.528	19.229
	atomic_global_stride	1.476	1.519	1.568	1.568	1.823	16.874	16.885	16.896	16.896	16.999	18.448	18.498	18.556	18.556	18.797
	atomic_private	1.450	1.484	1.560	1.560	1.860	3.999	4.015	4.027	4.027	4.143	5.533	5.585	5.673	5.673	5.952
	atomic_private_stride	1.446	1.492	1.558	1.558	1.809	2.995	3.006	3.018	3.018	3.145	4.519	4.574	4.646	4.646	4.940
27	atomic_global	18.352	20.004	20.363	20.363	22.275	267.749	268.000	268.016	268.016	268.089	286.431	288.109	288.481	288.481	290.387
	atomic_global_stride	19.689	19.884	20.131	20.131	20.597	268.170	268.467	268.581	268.581	268.781	288.226	288.514	288.744	288.744	289.324
	atomic_private	18.215	19.937	20.188	20.188	20.735	62.350	62.613	62.635	62.635	62.718	80.912	82.622	82.910	82.910	83.458
	atomic_private_stride	19.679	19.892	20.165	20.165	21.222	45.976	46.221	46.479	46.479	46.838	66.090	66.422	66.704	66.704	68.005
31	atomic_global	283.048	284.064	284.833	284.833	312.371	4281.418	4281.486	4281.554	4281.554	4281.728	4564.745	4565.726	4566.518	4566.518	4594.175
	atomic_global_stride	281.338	283.702	297.063	297.063	304.299	4290.564	4290.617	4291.023	4291.023	4296.653	4573.851	4574.899	4587.997	4587.997	4598.779
	atomic_private	283.771	284.213	285.105	285.105	301.789	990.382	990.615	992.017	992.017	995.181	1274.816	1275.398	1277.652	1277.652	1294.727
	atomic_private_stride	283.249	283.863	284.686	284.686	301.464	728.872	732.073	736.461	736.461	746.007	1012.874	1016.354	1025.308	1025.308	1043.373

Tesla V100, pseudozufällige Daten, 128 Bins

		Transferzeit Host zu Device (ms)					Ausführungszeit (ms)					Gesamtzeit (ms)				
n	Kernel	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max
19	atomic_global	0.353	0.353	0.353	0.353	0.398	0.146	0.146	0.147	0.147	0.148	0.515	0.516	0.517	0.517	0.572
	atomic_global_stride	0.353	0.353	0.353	0.353	0.367	0.146	0.146	0.147	0.147	0.149	0.516	0.516	0.517	0.517	0.537
	atomic_private	0.353	0.353	0.353	0.353	0.363	0.030	0.031	0.031	0.031	0.040	0.400	0.401	0.402	0.402	0.427
	atomic_private_stride	0.353	0.353	0.353	0.353	0.356	0.024	0.024	0.024	0.024	0.025	0.394	0.394	0.395	0.395	0.397
23	atomic_global	5.274	5.275	5.275	5.275	5.330	2.091	2.091	2.092	2.092	2.094	7.384	7.385	7.386	7.386	7.446
	atomic_global_stride	5.274	5.275	5.276	5.276	5.320	2.091	2.092	2.093	2.093	2.097	7.384	7.386	7.387	7.387	7.432
	atomic_private	5.274	5.274	5.275	5.275	5.290	0.247	0.247	0.248	0.248	0.255	5.540	5.541	5.542	5.542	5.558
	atomic_private_stride	5.274	5.275	5.275	5.275	5.280	0.048	0.049	0.050	0.050	0.052	5.341	5.342	5.344	5.344	5.351
27	atomic_global	83.639	83.641	83.644	83.644	83.665	33.214	33.215	33.218	33.218	33.261	116.884	116.888	116.894	116.894	116.966
	atomic_global_stride	83.640	83.641	83.643	83.643	83.660	33.223	33.226	33.227	33.227	33.236	116.896	116.898	116.901	116.901	116.920
	atomic_private	83.638	83.641	83.643	83.643	83.659	3.711	3.712	3.714	3.714	3.722	87.380	87.385	87.388	87.388	87.403
	atomic_private_stride	83.640	83.641	83.643	83.643	83.659	0.579	0.584	0.588	0.588	0.602	84.252	84.257	84.261	84.261	84.287
31	atomic_global	1337.006	1337.014	1337.019	1337.019	1339.197	530.681	530.685	530.694	530.694	530.852	1867.719	1867.737	1867.750	1867.750	1869.940
	atomic_global_stride	1337.010	1337.013	1337.020	1337.020	1339.190	530.796	530.802	530.810	530.810	530.850	1867.841	1867.853	1867.863	1867.863	1870.043
	atomic_private	1337.006	1337.012	1337.018	1337.018	1339.091	58.623	58.626	58.629	58.629	58.646	1395.666	1395.673	1395.680	1395.680	1397.770
	atomic_private_stride	1337.008	1337.012	1337.019	1337.019	1339.097	9.169	9.178	9.199	9.199	9.263	1346.215	1346.227	1346.247	1346.247	1348.333

Tesla V100, konstante Daten, 128 Bins

		Transferzeit Host zu Device (ms)					Ausführungszeit (ms)					Gesamtzeit (ms)				
n	Kernel	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max
19	atomic_global	0.353	0.353	0.353	0.353	0.394	0.379	0.379	0.380	0.380	0.381	0.749	0.749	0.750	0.750	0.797
	atomic_global_stride	0.353	0.353	0.353	0.353	0.365	0.379	0.379	0.380	0.380	0.382	0.749	0.749	0.750	0.750	0.768
	atomic_private	0.353	0.353	0.353	0.353	0.360	0.031	0.032	0.032	0.032	0.033	0.401	0.401	0.402	0.402	0.410
	atomic_private_stride	0.353	0.353	0.353	0.353	0.359	0.025	0.025	0.026	0.026	0.028	0.394	0.395	0.396	0.396	0.402
23	atomic_global	5.275	5.276	5.276	5.276	5.328	5.820	5.821	5.821	5.821	5.826	11.115	11.115	11.117	11.117	11.175
	atomic_global_stride	5.275	5.276	5.276	5.276	5.298	5.819	5.820	5.821	5.821	5.824	11.114	11.115	11.116	11.116	11.138
	atomic_private	5.275	5.276	5.276	5.276	5.297	0.247	0.248	0.248	0.248	0.257	5.541	5.542	5.544	5.544	5.564
	atomic_private_stride	5.274	5.276	5.276	5.276	5.283	0.093	0.095	0.099	0.099	0.106	5.388	5.390	5.394	5.394	5.401
27	atomic_global	83.642	83.643	83.645	83.645	83.676	92.869	92.871	92.873	92.873	92.883	176.544	176.546	176.549	176.549	176.584
	atomic_global_stride	83.640	83.643	83.644	83.644	83.667	92.871	92.873	92.875	92.875	92.915	176.546	176.548	176.551	176.551	176.592
	atomic_private	83.641	83.642	83.644	83.644	85.762	3.710	3.712	3.714	3.714	3.775	87.383	87.386	87.389	87.389	89.528
	atomic_private_stride	83.641	83.642	83.644	83.644	83.653	1.386	1.460	1.473	1.473	1.478	85.059	85.135	85.148	85.148	85.157
31	atomic_global	1336.964	1336.971	1336.990	1336.990	1341.759	1485.092	1485.094	1485.104	1485.104	1493.056	2822.089	2822.099	2822.125	2822.125	2830.083
	atomic_global_stride	1336.959	1336.967	1336.983	1336.983	1337.055	1485.095	1485.096	1485.100	1485.100	1494.072	2822.086	2822.096	2822.113	2822.113	2831.084
	atomic_private	1336.958	1336.966	1336.979	1336.979	1339.033	58.581	58.584	58.586	58.586	58.616	1395.568	1395.578	1395.592	1395.592	1397.675
	atomic_private_stride	1336.955	1336.966	1336.978	1336.978	1339.013	22.692	22.697	22.703	22.703	22.733	1359.684	1359.690	1359.709	1359.709	1361.763

Radeon RX 6800 XT, pseudozufällige Daten, 128 Bins

		Transferzeit Host zu Device (ms)					Ausführungszeit (ms)					Gesamtzeit (ms)				
n	Kernel	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max
19	atomic_global	0.048	0.049	0.049	0.049	0.480	0.064	0.067	0.067	0.067	0.211	0.133	0.135	0.136	0.136	0.933
	atomic_global_stride	0.047	0.049	0.050	0.050	0.133	0.065	0.067	0.068	0.068	0.150	0.133	0.137	0.141	0.141	0.224
	atomic_private	0.045	0.048	0.049	0.049	0.053	0.029	0.030	0.031	0.031	0.069	0.093	0.098	0.100	0.100	0.137
	atomic_private_stride	0.045	0.048	0.049	0.049	0.055	0.025	0.029	0.030	0.030	0.034	0.086	0.097	0.099	0.099	0.106
23	atomic_global	0.683	0.692	0.696	0.696	1.961	0.616	0.618	0.618	0.618	0.725	1.325	1.333	1.337	1.337	3.125
	atomic_global_stride	0.689	0.691	0.695	0.695	1.592	0.618	0.619	0.619	0.619	0.732	1.331	1.334	1.339	1.339	2.343
	atomic_private	0.688	0.691	0.694	0.694	1.581	0.094	0.095	0.097	0.097	0.101	0.808	0.810	0.815	0.815	1.694
	atomic_private_stride	0.686	0.691	0.694	0.694	1.590	0.059	0.061	0.062	0.062	0.068	0.772	0.775	0.779	0.779	1.669
27	atomic_global	11.276	11.289	11.310	11.310	13.979	9.407	9.410	9.678	9.678	9.688	20.706	20.732	21.002	21.002	24.017
	atomic_global_stride	11.278	11.294	11.315	11.315	11.965	9.445	9.457	9.722	9.722	9.761	20.754	20.788	21.054	21.054	21.726
	atomic_private	10.496	10.521	11.326	11.326	11.539	1.006	1.012	1.031	1.031	1.055	11.523	11.574	12.375	12.375	12.576
	atomic_private_stride	10.517	11.297	11.331	11.331	11.523	0.456	0.463	0.477	0.477	0.484	11.002	11.788	11.823	11.823	12.009
31	atomic_global	110.288	112.201	113.354	113.354	114.239	154.439	154.542	154.772	154.772	155.143	264.849	266.975	268.202	268.202	269.274
	atomic_global_stride	110.408	111.807	113.260	113.260	113.910	153.716	154.633	155.511	155.511	156.477	265.680	267.303	268.644	268.644	270.065
	atomic_private	111.125	111.690	112.990	112.990	114.239	19.446	19.623	19.840	19.840	20.060	130.845	131.464	132.828	132.828	133.902
	atomic_private_stride	110.840	111.510	112.887	112.887	113.796	10.906	11.040	11.193	11.193	13.098	122.092	122.852	124.095	124.095	124.950

Radeon RX 6800 XT, konstante Daten, 128 Bins

		Transferzeit Host zu Device (ms)					Ausführungszeit (ms)					Gesamtzeit (ms)				
n	Kernel	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max
19	atomic_global	0.048	0.048	0.049	0.049	0.458	0.241	0.242	0.243	0.243	0.460	0.310	0.311	0.312	0.312	1.095
	atomic_global_stride	0.047	0.048	0.049	0.049	0.130	0.241	0.242	0.242	0.242	0.472	0.307	0.310	0.311	0.311	0.541
	atomic_private	0.045	0.048	0.049	0.049	0.053	0.037	0.038	0.039	0.039	0.081	0.099	0.106	0.108	0.108	0.150
	atomic_private_stride	0.044	0.048	0.049	0.049	0.053	0.033	0.037	0.038	0.038	0.041	0.095	0.104	0.107	0.107	0.112
23	atomic_global	1.427	1.454	1.498	1.498	1.934	3.453	3.454	3.455	3.455	3.639	4.905	4.932	4.975	4.975	5.998
	atomic_global_stride	1.418	1.434	1.460	1.460	1.586	3.466	3.474	3.478	3.478	3.655	4.912	4.934	4.960	4.960	5.124
	atomic_private	0.550	0.551	0.554	0.554	1.449	0.214	0.216	0.217	0.217	0.222	0.788	0.791	0.795	0.795	1.683
	atomic_private_stride	0.540	0.544	0.548	0.548	1.436	0.145	0.149	0.150	0.150	0.159	0.710	0.716	0.721	0.721	1.607
27	atomic_global	12.140	13.277	13.315	13.315	15.814	54.818	54.824	55.090	55.090	55.101	66.982	68.138	68.413	68.413	71.258
	atomic_global_stride	12.109	13.273	13.331	13.331	13.924	55.030	55.096	55.344	55.344	55.614	67.183	68.428	68.696	68.696	68.976
	atomic_private	12.072	12.136	13.316	13.316	13.686	2.934	2.937	2.945	2.945	2.952	15.033	15.101	16.279	16.279	16.648
	atomic_private_stride	12.084	13.230	13.256	13.256	13.647	2.649	2.680	2.709	2.709	2.769	14.816	15.938	15.982	15.982	16.407
31	atomic_global	116.963	118.640	120.158	120.158	122.871	884.648	885.157	898.592	898.592	899.968	1001.633	1004.253	1018.999	1018.999	1021.991
	atomic_global_stride	116.945	117.512	118.237	118.237	120.749	892.669	893.865	896.126	896.126	900.364	1010.924	1012.119	1014.356	1014.356	1018.873
	atomic_private	116.900	118.119	118.286	118.286	119.819	49.180	49.497	49.760	49.760	50.460	166.336	167.504	168.116	168.116	169.526
	atomic_private_stride	116.880	117.207	118.216	118.216	118.701	40.961	41.182	41.596	41.596	42.643	158.107	159.054	159.823	159.823	160.758

Xavier NX, pseudozufällige Daten, 27 Bins

		Transferzeit Host zu Device (ms)					Ausführungszeit (ms)					Gesamtzeit (ms)				
n	Kernel	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max
19	atomic_global	0.131	0.132	0.134	0.134	0.213	0.930	0.934	0.951	0.951	1.028	1.125	1.132	1.164	1.164	1.260
	atomic_global_stride	0.131	0.131	0.134	0.134	0.220	0.931	0.933	0.936	0.936	0.991	1.125	1.128	1.145	1.145	1.291
	atomic_private	0.102	0.103	0.109	0.109	0.204	0.242	0.245	0.257	0.257	0.351	0.387	0.391	0.416	0.416	0.593
	atomic_private_stride	0.103	0.104	0.108	0.108	0.184	0.212	0.217	0.224	0.224	0.317	0.360	0.364	0.380	0.380	0.556
23	atomic_global	1.442	1.496	1.584	1.584	1.756	13.487	13.497	13.509	13.509	13.603	15.014	15.079	15.185	15.185	15.360
	atomic_global_stride	1.439	1.483	1.558	1.558	1.747	13.534	13.545	13.556	13.556	13.653	15.088	15.111	15.212	15.212	15.370
	atomic_private	1.436	1.452	1.541	1.541	1.865	2.823	2.830	2.840	2.840	2.915	4.341	4.367	4.464	4.464	4.806
	atomic_private_stride	1.296	1.390	1.503	1.503	1.705	1.865	1.879	1.902	1.902	2.055	3.245	3.363	3.497	3.497	3.815
27	atomic_global	19.745	19.978	20.390	20.390	22.235	214.060	214.066	214.077	214.077	214.131	233.922	234.133	234.560	234.560	236.488
	atomic_global_stride	19.523	19.796	20.081	20.081	22.342	214.904	215.164	215.234	215.234	215.330	234.892	235.101	235.396	235.396	237.671
	atomic_private	19.834	19.956	20.128	20.128	20.602	43.502	43.524	43.575	43.575	43.780	63.471	63.604	63.805	63.805	64.309
	atomic_private_stride	18.326	19.838	20.143	20.143	21.097	28.127	28.254	28.290	28.290	28.606	46.684	48.245	48.546	48.546	49.458
31	atomic_global	282.521	283.191	284.034	284.034	311.395	3418.587	3418.596	3418.617	3418.617	3418.676	3701.257	3701.896	3702.765	3702.765	3730.183
	atomic_global_stride	282.182	282.901	283.808	283.808	296.675	3437.483	3437.536	3437.603	3437.603	3442.019	3719.768	3720.760	3722.688	3722.688	3736.744
	atomic_private	282.186	282.975	283.548	283.548	288.860	689.708	689.869	690.048	690.048	690.682	972.526	973.144	973.820	973.820	979.645
	atomic_private_stride	279.076	281.288	282.014	282.014	286.487	448.587	448.701	448.768	448.768	449.395	727.962	730.154	730.843	730.843	735.315

Xavier NX, konstante Daten, 27 Bins

		Transferzeit Host zu Device (ms)					Ausführungszeit (ms)					Gesamtzeit (ms)				
n	Kernel	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max
19	atomic_global	0.132	0.132	0.135	0.135	0.229	1.135	1.140	1.149	1.149	1.216	1.314	1.337	1.358	1.358	1.501
	atomic_global_stride	0.135	0.137	0.140	0.140	0.208	1.140	1.142	1.146	1.146	1.194	1.329	1.341	1.352	1.352	1.461
	atomic_private	0.104	0.108	0.118	0.118	0.184	0.313	0.319	0.336	0.336	0.414	0.463	0.475	0.515	0.515	0.639
	atomic_private_stride	0.102	0.103	0.120	0.120	0.243	0.277	0.283	0.295	0.295	0.331	0.423	0.429	0.477	0.477	0.623
23	atomic_global	1.450	1.478	1.514	1.514	1.874	16.877	16.887	16.917	16.917	17.017	18.418	18.447	18.522	18.522	18.973
	atomic_global_stride	1.460	1.485	1.556	1.556	1.830	16.879	16.889	16.898	16.898	17.028	18.432	18.460	18.544	18.544	18.805
	atomic_private	1.092	1.439	1.480	1.480	1.744	4.014	4.025	4.039	4.039	4.126	5.158	5.534	5.592	5.592	5.878
	atomic_private_stride	1.438	1.505	1.569	1.569	1.811	2.994	3.026	3.044	3.044	3.261	4.532	4.617	4.710	4.710	5.137
27	atomic_global	19.758	19.914	20.214	20.214	21.769	267.967	268.209	268.226	268.226	268.305	288.053	288.240	288.551	288.551	289.980
	atomic_global_stride	19.612	19.791	20.111	20.111	22.133	268.211	268.496	268.629	268.629	268.777	288.217	288.424	288.779	288.779	290.732
	atomic_private	19.701	19.869	20.109	20.109	20.873	62.667	62.705	62.749	62.749	62.873	82.550	82.709	82.953	82.953	83.651
	atomic_private_stride	18.482	19.841	20.100	20.100	20.933	46.202	46.488	46.757	46.757	47.077	65.377	66.542	66.987	66.987	68.040
31	atomic_global	279.969	292.774	296.445	296.445	312.967	4284.958	4285.038	4285.132	4285.132	4285.381	4565.387	4577.985	4581.756	4581.756	4598.491
	atomic_global_stride	295.589	296.387	297.186	297.186	302.839	4290.977	4291.047	4291.282	4291.282	4294.849	4586.784	4587.838	4590.311	4590.311	4597.164
	atomic_private	293.783	296.281	296.855	296.855	301.968	994.738	994.813	995.226	995.226	996.651	1289.407	1291.514	1292.313	1292.313	1297.050
	atomic_private_stride	295.793	296.192	297.008	297.008	302.650	729.459	732.205	732.239	732.239	740.719	1025.987	1028.304	1029.477	1029.477	1038.514

Tesla V100, pseudozufällige Daten, 27 Bins

		Transferzeit Host zu Device (ms)					Ausführungszeit (ms)					Gesamtzeit (ms)				
n	Kernel	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max
19	atomic_global	0.353	0.353	0.353	0.353	0.393	0.307	0.307	0.308	0.308	0.310	0.676	0.677	0.678	0.678	0.724
	atomic_global_stride	0.353	0.353	0.353	0.353	0.364	0.307	0.307	0.308	0.308	0.313	0.676	0.677	0.678	0.678	0.695
	atomic_private	0.353	0.353	0.353	0.353	0.355	0.024	0.024	0.025	0.025	0.026	0.393	0.394	0.395	0.395	0.400
	atomic_private_stride	0.353	0.353	0.353	0.353	0.358	0.021	0.021	0.022	0.022	0.023	0.390	0.391	0.392	0.392	0.397
23	atomic_global	5.272	5.272	5.273	5.273	5.329	4.664	4.664	4.665	4.665	4.670	9.955	9.956	9.957	9.957	10.017
	atomic_global_stride	5.272	5.272	5.273	5.273	5.298	4.664	4.664	4.665	4.665	4.675	9.955	9.956	9.958	9.958	9.986
	atomic_private	5.271	5.272	5.273	5.273	5.287	0.133	0.133	0.134	0.134	0.142	5.423	5.424	5.425	5.425	5.441
	atomic_private_stride	5.271	5.272	5.272	5.272	5.279	0.067	0.068	0.069	0.069	0.072	5.357	5.359	5.360	5.360	5.367
27	atomic_global	83.640	83.642	83.645	83.645	83.679	74.397	74.399	74.402	74.402	74.428	158.070	158.074	158.079	158.079	158.132
	atomic_global_stride	83.641	83.642	83.644	83.644	83.658	74.400	74.401	74.403	74.403	74.442	158.073	158.076	158.079	158.079	158.122
	atomic_private	83.640	83.642	83.645	83.645	83.661	1.887	1.890	1.891	1.891	1.936	85.558	85.564	85.567	85.567	85.624
	atomic_private_stride	83.641	83.642	83.644	83.644	83.662	0.794	0.830	0.834	0.834	0.843	84.470	84.504	84.508	84.508	84.529
31	atomic_global	1336.987	1336.996	1337.005	1337.005	1339.125	1189.591	1189.596	1189.604	1189.604	1197.395	2526.616	2526.627	2526.647	2526.647	2534.447
	atomic_global_stride	1336.984	1336.991	1336.996	1336.996	1339.042	1189.652	1189.655	1189.661	1189.661	1198.523	2526.678	2526.684	2526.693	2526.693	2535.568
	atomic_private	1336.990	1336.995	1337.001	1337.001	1339.081	29.433	29.435	29.439	29.439	29.453	1366.457	1366.463	1366.472	1366.472	1368.568
	atomic_private_stride	1336.987	1336.994	1337.000	1337.000	1339.098	13.838	13.853	13.864	13.864	13.885	1350.869	1350.883	1350.895	1350.895	1353.011

Tesla V100, konstante Daten, 27 Bins

		Transferzeit Host zu Device (ms)					Ausführungszeit (ms)					Gesamtzeit (ms)				
n	Kernel	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max
19	atomic_global	0.353	0.353	0.353	0.353	0.394	0.379	0.379	0.380	0.380	0.381	0.748	0.749	0.750	0.750	0.802
	atomic_global_stride	0.353	0.353	0.353	0.353	0.362	0.379	0.379	0.379	0.379	0.383	0.748	0.749	0.750	0.750	0.764
	atomic_private	0.353	0.353	0.353	0.353	0.353	0.024	0.025	0.025	0.025	0.026	0.394	0.394	0.395	0.395	0.397
	atomic_private_stride	0.352	0.353	0.353	0.353	0.359	0.022	0.022	0.023	0.023	0.024	0.391	0.392	0.393	0.393	0.399
23	atomic_global	5.276	5.276	5.277	5.277	5.328	5.820	5.821	5.821	5.821	5.827	11.115	11.116	11.118	11.118	11.173
	atomic_global_stride	5.276	5.276	5.277	5.277	5.292	5.820	5.820	5.821	5.821	5.826	11.115	11.116	11.117	11.117	11.134
	atomic_private	5.275	5.276	5.276	5.276	5.290	0.134	0.135	0.135	0.135	0.141	5.427	5.429	5.431	5.431	5.446
	atomic_private_stride	5.275	5.276	5.276	5.276	5.287	0.093	0.094	0.099	0.099	0.106	5.388	5.390	5.394	5.394	5.406
27	atomic_global	83.639	83.641	83.643	83.643	83.662	92.868	92.871	92.874	92.874	92.885	176.538	176.544	176.549	176.549	176.575
	atomic_global_stride	83.639	83.641	83.643	83.643	83.662	92.869	92.871	92.873	92.873	92.881	176.540	176.544	176.548	176.548	176.572
	atomic_private	83.639	83.641	83.643	83.643	83.659	1.888	1.890	1.891	1.891	1.899	85.560	85.562	85.564	85.564	85.584
	atomic_private_stride	83.639	83.641	83.643	83.643	85.084	1.377	1.381	1.386	1.386	3.427	85.049	85.054	85.060	85.060	88.545
31	atomic_global	1336.983	1336.993	1336.999	1336.999	1339.232	1485.124	1485.128	1485.139	1485.139	1485.589	2822.154	2822.160	2822.178	2822.178	2824.416
	atomic_global_stride	1336.989	1336.995	1337.000	1337.000	1339.087	1485.126	1485.132	1485.139	1485.139	1494.147	2822.152	2822.165	2822.176	2822.176	2831.205
	atomic_private	1336.983	1336.988	1336.996	1336.996	1339.085	29.438	29.442	29.444	29.444	29.457	1366.462	1366.466	1366.474	1366.474	1368.574
	atomic_private_stride	1336.988	1336.992	1336.998	1336.998	1339.108	22.712	22.719	22.727	22.727	22.758	1359.738	1359.747	1359.758	1359.758	1361.891

Radeon RX 6800 XT, pseudozufällige Daten, 27 Bins

		Transferzeit Host zu Device (ms)					Ausführungszeit (ms)					Gesamtzeit (ms)				
n	Kernel	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max
19	atomic_global	0.048	0.049	0.050	0.050	0.504	0.172	0.172	0.174	0.174	0.176	0.242	0.244	0.247	0.247	1.131
	atomic_global_stride	0.048	0.049	0.050	0.050	0.149	0.170	0.172	0.174	0.174	0.372	0.238	0.243	0.247	0.247	0.444
	atomic_private	0.046	0.048	0.049	0.049	0.064	0.034	0.037	0.039	0.039	0.077	0.097	0.108	0.112	0.112	0.153
	atomic_private_stride	0.046	0.048	0.049	0.049	0.056	0.033	0.036	0.038	0.038	0.040	0.095	0.107	0.111	0.111	0.120
23	atomic_global	0.892	2.004	2.028	2.028	2.365	2.263	2.265	2.265	2.265	2.271	3.179	4.292	4.315	4.315	5.039
	atomic_global_stride	0.893	1.819	2.017	2.017	2.312	2.268	2.270	2.278	2.278	2.510	3.192	4.237	4.319	4.319	4.604
	atomic_private	0.871	0.878	0.882	0.882	1.815	0.148	0.149	0.150	0.150	0.155	1.043	1.052	1.056	1.056	1.982
	atomic_private_stride	0.877	0.878	0.882	0.882	1.793	0.105	0.107	0.108	0.108	0.110	1.004	1.009	1.013	1.013	1.920
27	atomic_global	12.006	12.025	12.047	12.047	14.534	35.723	35.731	35.995	35.995	36.004	47.755	47.795	48.055	48.055	50.876
	atomic_global_stride	11.996	12.036	12.060	12.060	13.912	36.016	36.245	36.481	36.481	36.654	48.119	48.353	48.565	48.565	50.221
	atomic_private	11.978	12.085	12.122	12.122	12.310	1.851	1.854	1.856	1.856	1.864	13.851	13.957	13.995	13.995	14.189
	atomic_private_stride	11.959	11.985	12.015	12.015	12.749	1.134	1.139	1.145	1.145	1.175	13.117	13.147	13.183	13.183	13.930
31	atomic_global	111.409	111.641	113.064	113.064	113.855	576.018	576.098	576.343	576.343	576.500	687.708	687.973	689.378	689.378	690.284
	atomic_global_stride	111.449	111.889	113.207	113.207	114.220	584.060	587.720	589.601	589.601	591.647	697.249	700.353	702.689	702.689	705.058
	atomic_private	111.373	111.724	113.135	113.135	113.738	33.002	33.051	33.307	33.307	33.537	144.560	145.039	146.476	146.476	147.020
	atomic_private_stride	111.422	111.528	113.125	113.125	113.827	18.011	18.108	18.364	18.364	18.855	129.574	130.010	131.506	131.506	132.447

Radeon 6800 XT, konstante Daten, 27 Bins

		Transferzeit Host zu Device (ms)					Ausführungszeit (ms)					Gesamtzeit (ms)				
n	Kernel	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max	Min	Q10	Med	Q90	Max
19	atomic_global	0.048	0.048	0.049	0.049	0.461	0.246	0.248	0.248	0.248	0.251	0.316	0.318	0.321	0.321	1.099
	atomic_global_stride	0.047	0.048	0.049	0.049	0.128	0.246	0.248	0.248	0.248	0.490	0.311	0.318	0.320	0.320	0.566
	atomic_private	0.045	0.048	0.049	0.049	0.055	0.039	0.042	0.044	0.044	0.068	0.101	0.112	0.116	0.116	0.140
	atomic_private_stride	0.045	0.048	0.049	0.049	0.054	0.037	0.041	0.043	0.043	0.049	0.100	0.112	0.115	0.115	0.120
23	atomic_global	0.885	0.893	1.749	1.749	2.220	3.457	3.457	3.458	3.458	3.601	4.364	4.373	5.229	5.229	6.229
	atomic_global_stride	0.881	0.887	1.745	1.745	1.943	3.459	3.464	3.472	3.472	3.782	4.361	4.380	5.237	5.237	5.580
	atomic_private	0.857	0.868	0.872	0.872	1.877	0.222	0.223	0.224	0.224	0.240	1.101	1.114	1.120	1.120	2.119
	atomic_private_stride	0.863	0.866	0.871	0.871	1.782	0.151	0.152	0.154	0.154	0.159	1.035	1.041	1.047	1.047	1.953
27	atomic_global	11.917	11.935	11.972	11.972	12.970	54.815	54.821	54.897	54.897	55.103	66.761	66.781	66.991	66.991	68.418
	atomic_global_stride	11.903	11.930	11.959	11.959	14.009	55.144	55.333	55.582	55.582	55.756	67.131	67.302	67.579	67.579	69.723
	atomic_private	11.861	11.893	11.986	11.986	12.243	2.980	2.984	2.989	2.989	2.994	14.860	14.899	14.992	14.992	15.252
	atomic_private_stride	11.855	11.878	11.895	11.895	12.382	2.077	2.094	2.122	2.122	2.207	13.975	13.999	14.055	14.055	14.541
31	atomic_global	119.242	119.935	121.555	121.555	122.460	891.107	891.261	891.591	891.591	892.110	1010.586	1011.349	1013.049	1013.049	1014.259
	atomic_global_stride	119.001	119.986	121.725	121.725	122.940	887.748	894.824	897.364	897.364	902.393	1008.997	1015.545	1018.975	1018.975	1024.171
	atomic_private	117.959	119.352	119.812	119.812	122.460	52.394	53.016	53.510	53.510	54.368	171.043	172.540	173.424	173.424	176.276
	atomic_private_stride	118.420	119.743	121.580	121.580	122.736	38.453	38.934	39.307	39.307	40.918	157.319	159.098	160.799	160.799	162.396