f\_par\_pro\_WiSe 2024/25;

4. Aufgabe: Reduktion;

Ersteller: Stefan Butz;

Verwendete Hardware:

* Google Cloud Compute Engine (g2-standard-4)
* NVIDIA L4

Verwendete Programmierung-Umgebung: Visual Studio Code (Linux)

Inhalt

[1. Problemlösungsansatz 2](#_Toc186982719)

[2. Messergebnisse 3](#_Toc186982720)

[3. Vergleiche und Schlussfolgerungen 7](#_Toc186982721)

# Problemlösungsansatz

Die vorstellten Optimierungen wurden inkrementell umgesetzt. Es wurden insgesamt sieben Kernel erstellt.

**Separate Kernel (01\_separate\_kernel.cu)**

Das Programm besteht aus drei Kernel. Der erste Kernel (Init) initialisiert die zu verarbeitenden Daten. Der zweite Kernel (Sum) berechnet jeweils die Summe eines Blocks. Das Ergebnis wird als Eingabe für den gleichen Kernel benutzt. Dieser wird so lange aufgerufen, bis die Reduktion abgeschlossen ist.

**Atomare Operationen (02\_atomic\_kernel.cu)**

Der mehrfachen Kernelaufrufe, welche für die Inter-Block-Reduktion notwendig waren, werden durch atomare Operationen ersetzt.

**Kaskadierung (03\_atomic\_kernel\_cascade.cu)**

Jeder Thread verarbeitet nun mehr als einen Wert. Dadurch werden die Ressourcen besser genutzt, da nicht die Hälfte der Threads nach dem Laden leerlaufen. Die Anzahl der Rechenoperationen pro Thread wurden variert.

**Vermeidung des Modulo Operator (04\_atomic\_kernel\_opt.cu)**

Der Modulo Operator wurde durch eine Multiplikation ersetzt.

**Sequenzielle Adressierung (05\_atomic\_kernel\_seq\_address.cu)**

Der Zugriffsreihenfolge auf den geteilten Speicher wurde optimiert, um Coalescing Effekte auszunutzen.

**Loop Unrolling (06\_atomic\_kernel\_loop\_unroll.cu)**

Die Anzahl der Sprung- und Vergleichsoperationen wurde durch Schleifenabwicklung reduziert.

**Shuffle Operationen (07\_atomic\_kernel\_shuffle.cu)**

Statt des geteilten Speichers wurde Shuffle-Operationen genutzt, um Daten innerhalb eines Warps auszutauschen.

# Messergebnisse

Die Messwerte wurden mittels Nvidia Nsight, dem Nachfolger von nvprof) bestimmt. Die genauen Befehle können dem beiliegendem Messskript entnommen werden.

Für jeden Kernel wurden pro Problemgröße 1024 gemessen.

|  |  |  |  |
| --- | --- | --- | --- |
| Kernel | Problem Größe | Ausführungszeit des Kernels (in us) | Startzeit des Kernels (in us) |
| 01\_seperate\_kernels | 65536 | 11.5146 | 13.2044 |
| 01\_seperate\_kernels | 131072 | 15.0684 | 12.8376 |
| 01\_seperate\_kernels | 262144 | 22.0914 | 12.6946 |
| 01\_seperate\_kernels | 524288 | 35.8266 | 14.3840 |
| 01\_seperate\_kernels | 1048576 | 66.2976 | 19.5714 |
| 01\_seperate\_kernels | 2097152 | 131.0946 | 19.9809 |
| 02\_atomic\_kernel | 65536 | 7.7364 | 7.4588 |
| 02\_atomic\_kernel | 131072 | 11.3522 | 7.2189 |
| 02\_atomic\_kernel | 262144 | 18.6092 | 7.4962 |
| 02\_atomic\_kernel | 524288 | 32.8542 | 8.4472 |
| 02\_atomic\_kernel | 1048576 | 64.3538 | 8.9075 |
| 02\_atomic\_kernel | 2097152 | 127.1948 | 9.2633 |
| 03\_atomic\_kernel\_cascade | 65536 | 8.7317 | 7.2424 |
| 03\_atomic\_kernel\_cascade | 131072 | 8.9582 | 7.4323 |
| 03\_atomic\_kernel\_cascade | 262144 | 8.9646 | 7.1189 |
| 03\_atomic\_kernel\_cascade | 524288 | 8.9892 | 8.1875 |
| 03\_atomic\_kernel\_cascade | 1048576 | 9.0402 | 8.5692 |
| 03\_atomic\_kernel\_cascade | 2097152 | 16.3431 | 8.7488 |
| 04\_atomic\_kernel\_opt | 65536 | 7.5753 | 7.2932 |
| 04\_atomic\_kernel\_opt | 131072 | 7.8054 | 7.2379 |
| 04\_atomic\_kernel\_opt | 262144 | 7.8153 | 7.2156 |
| 04\_atomic\_kernel\_opt | 524288 | 7.8276 | 8.3312 |
| 04\_atomic\_kernel\_opt | 1048576 | 7.8837 | 9.0634 |
| 04\_atomic\_kernel\_opt | 2097152 | 13.9925 | 8.7350 |
| 05\_atomic\_kernel\_seq\_address | 65536 | 7.2283 | 7.2764 |
| 05\_atomic\_kernel\_seq\_address | 131072 | 7.4471 | 7.3772 |
| 05\_atomic\_kernel\_seq\_address | 262144 | 7.4521 | 7.3321 |
| 05\_atomic\_kernel\_seq\_address | 524288 | 7.4768 | 7.8958 |
| 05\_atomic\_kernel\_seq\_address | 1048576 | 7.5221 | 8.5797 |
| 05\_atomic\_kernel\_seq\_address | 2097152 | 13.2812 | 8.7220 |
| 06\_atomic\_kernel\_loop\_unroll | 65536 | 6.5488 | 7.2566 |
| 06\_atomic\_kernel\_loop\_unroll | 131072 | 6.7746 | 7.3096 |
| 06\_atomic\_kernel\_loop\_unroll | 262144 | 6.7805 | 7.3185 |
| 06\_atomic\_kernel\_loop\_unroll | 524288 | 6.8021 | 8.2295 |
| 06\_atomic\_kernel\_loop\_unroll | 1048576 | 6.8428 | 7.7973 |
| 06\_atomic\_kernel\_loop\_unroll | 2097152 | 11.9413 | 8.6792 |
| 07\_atomic\_kernel\_shuffle | 65536 | 6.1897 | 7.2973 |
| 07\_atomic\_kernel\_shuffle | 131072 | 6.4006 | 7.1836 |
| 07\_atomic\_kernel\_shuffle | 262144 | 6.4098 | 7.5451 |
| 07\_atomic\_kernel\_shuffle | 524288 | 6.507 | 8.2465 |
| 07\_atomic\_kernel\_shuffle | 1048576 | 7.3400 | 8.0245 |
| 07\_atomic\_kernel\_shuffle | 2097152 | 12.0462 | 8.6830 |

Für den Kernel 03\_atomic\_kernel\_cascade wurde pro Problemgröße Messungen mit verschieden Kaskadierungsstufen, genauer wie viele Werte jeder Thread lädt und aufsummiert bevor der Reduktionsbaum durchschritten wird, durchgeführt.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Kernel | Problemgröße | Kaskadierungsstufe | Ausführungszeit des Kernels (in us) | Startzeit des Kernels (in us) |
| 03\_atomic\_kernel\_cascade | 65536 | 2 | 4.4434 | 7.4216 |
| 03\_atomic\_kernel\_cascade | 65536 | 4 | 4.7307 | 7.4926 |
| 03\_atomic\_kernel\_cascade | 65536 | 8 | 5.3256 | 7.3685 |
| 03\_atomic\_kernel\_cascade | 65536 | 16 | 6.5419 | 7.3708 |
| 03\_atomic\_kernel\_cascade | 65537 | 32 | 8.7335 | 7.4620 |
| 03\_atomic\_kernel\_cascade | 65536 | 64 | 13.3604 | 7.2411 |
| 03\_atomic\_kernel\_cascade | 65536 | 128 | 13.3621 | 7.1484 |
| 03\_atomic\_kernel\_cascade | 65536 | 256 | 13.3713 | 7.8037 |
| 03\_atomic\_kernel\_cascade | 65536 | 512 | 13.3613 | 7.3834 |
| 03\_atomic\_kernel\_cascade | 65536 | 1024 | 13.3608 | 7.4660 |
| 03\_atomic\_kernel\_cascade | 131072 | 2 | 8.0893 | 7.6224 |
| 03\_atomic\_kernel\_cascade | 131072 | 4 | 4.7539 | 7.4678 |
| 03\_atomic\_kernel\_cascade | 131072 | 8 | 5.3363 | 7.2517 |
| 03\_atomic\_kernel\_cascade | 131072 | 16 | 6.5322 | 7.6826 |
| 03\_atomic\_kernel\_cascade | 131072 | 32 | 8.9562 | 7.2523 |
| 03\_atomic\_kernel\_cascade | 131072 | 64 | 13.3497 | 7.3498 |
| 03\_atomic\_kernel\_cascade | 131072 | 128 | 22.6063 | 7.3087 |
| 03\_atomic\_kernel\_cascade | 131072 | 256 | 22.6055 | 7.0913 |
| 03\_atomic\_kernel\_cascade | 131072 | 512 | 22.6084 | 7.9121 |
| 03\_atomic\_kernel\_cascade | 131072 | 1024 | 22.6081 | 7.4438 |
| 03\_atomic\_kernel\_cascade | 262144 | 2 | 11.9347 | 7.5234 |
| 03\_atomic\_kernel\_cascade | 262144 | 4 | 8.67429 | 6.9700 |
| 03\_atomic\_kernel\_cascade | 262144 | 8 | 5.3754 | 7.2815 |
| 03\_atomic\_kernel\_cascade | 262144 | 16 | 6.5667 | 7.9498 |
| 03\_atomic\_kernel\_cascade | 262144 | 32 | 8.9652 | 7.3232 |
| 03\_atomic\_kernel\_cascade | 262144 | 64 | 13.7922 | 7.5527 |
| 03\_atomic\_kernel\_cascade | 262144 | 128 | 22.6008 | 7.2287 |
| 03\_atomic\_kernel\_cascade | 262144 | 256 | 41.0954 | 7.0823 |
| 03\_atomic\_kernel\_cascade | 262144 | 512 | 41.092 | 7.4748 |
| 03\_atomic\_kernel\_cascade | 262144 | 1024 | 41.1024 | 7.4992 |
| 03\_atomic\_kernel\_cascade | 524288 | 2 | 19.4621 | 8.1205 |
| 03\_atomic\_kernel\_cascade | 524288 | 4 | 12.7627 | 8.0816 |
| 03\_atomic\_kernel\_cascade | 524288 | 8 | 9.7904 | 8.5553 |
| 03\_atomic\_kernel\_cascade | 524288 | 16 | 6.5949 | 7.9864 |
| 03\_atomic\_kernel\_cascade | 524288 | 32 | 8.9887 | 8.1596 |
| 03\_atomic\_kernel\_cascade | 524288 | 64 | 13.7977 | 9.4967 |
| 03\_atomic\_kernel\_cascade | 524288 | 128 | 23.4499 | 8.3531 |
| 03\_atomic\_kernel\_cascade | 524288 | 256 | 41.0856 | 8.0595 |
| 03\_atomic\_kernel\_cascade | 524288 | 512 | 78.0716 | 8.9596 |
| 03\_atomic\_kernel\_cascade | 524288 | 1024 | 78.0686 | 8.4697 |
| 03\_atomic\_kernel\_cascade | 1048576 | 2 | 34.4987 | 8.5801 |
| 03\_atomic\_kernel\_cascade | 1048576 | 4 | 20.9382 | 8.7529 |
| 03\_atomic\_kernel\_cascade | 1048576 | 8 | 14.5483 | 8.7291 |
| 03\_atomic\_kernel\_cascade | 1048576 | 16 | 12.0168 | 8.4225 |
| 03\_atomic\_kernel\_cascade | 1048576 | 32 | 9.0413 | 8.3868 |
| 03\_atomic\_kernel\_cascade | 1048576 | 64 | 13.827 | 8.7358 |
| 03\_atomic\_kernel\_cascade | 1048576 | 128 | 23.4473 | 8.9173 |
| 03\_atomic\_kernel\_cascade | 1048576 | 256 | 42.7378 | 8.6594 |
| 03\_atomic\_kernel\_cascade | 1048576 | 512 | 78.0381 | 8.4631 |
| 03\_atomic\_kernel\_cascade | 1048576 | 1024 | 152.0097 | 8.6795 |
| 03\_atomic\_kernel\_cascade | 2097152 | 2 | 67.6087 | 8.8306 |
| 03\_atomic\_kernel\_cascade | 2097152 | 4 | 37.0965 | 9.2235 |
| 03\_atomic\_kernel\_cascade | 2097152 | 8 | 23.8682 | 8.7187 |
| 03\_atomic\_kernel\_cascade | 2097152 | 16 | 18.021 | 9.1297 |
| 03\_atomic\_kernel\_cascade | 2097152 | 32 | 16.3355 | 9.1127 |
| 03\_atomic\_kernel\_cascade | 2097152 | 64 | 13.9077 | 8.5791 |
| 03\_atomic\_kernel\_cascade | 2097152 | 128 | 23.5108 | 9.0050 |
| 03\_atomic\_kernel\_cascade | 2097152 | 256 | 42.7378 | 8.9721 |
| 03\_atomic\_kernel\_cascade | 2097152 | 512 | 81.2978 | 8.8363 |
| 03\_atomic\_kernel\_cascade | 2097152 | 1024 | 151.9416 | 8.9289 |

# Vergleiche und Schlussfolgerungen

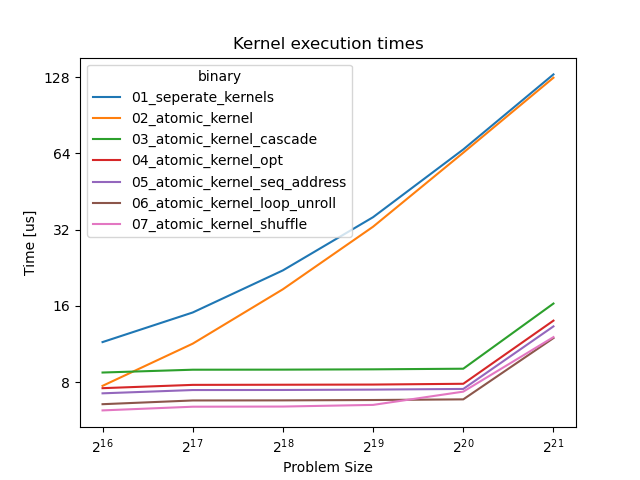


Abbildung : Vergleich von Ausführungszeiten

In der gleichen Reihenfolge wie in Kapitel 1 folgt hier der Vergleich und die Bewertung der verschiedenen Optimierungsmaßnahmen.

Abbildung 1 zeigt die Ausführungszeit der verschiedenen Kernel mit steigender Problemgröße. Abbildung 2 zeigt für jeden Kernel die Ausführungszeit des Kernels auf der GPU sowie die Startzeit des Kernels. Bei Zeitmessungen mit CUDA Events ist die gemessene Zeit für einen Kernel jeweils die Summe aus Start- und Ausführungszeit und die Analyse folglich grobgranularer als dies mit Nsight möglich ist.

Durch die Einführung atomarer Operationen wurde die Laufzeit, insbesondere für geringe Problemgrößen, deutlich verbessert werden. Pro Kernelstart gibt es einen annähernd konstanten Overhead. Besonders bei geringer Problemgröße zeigt sich hier der Nachteil der mehrstufigen Lösung (01\_seperate\_kernels).

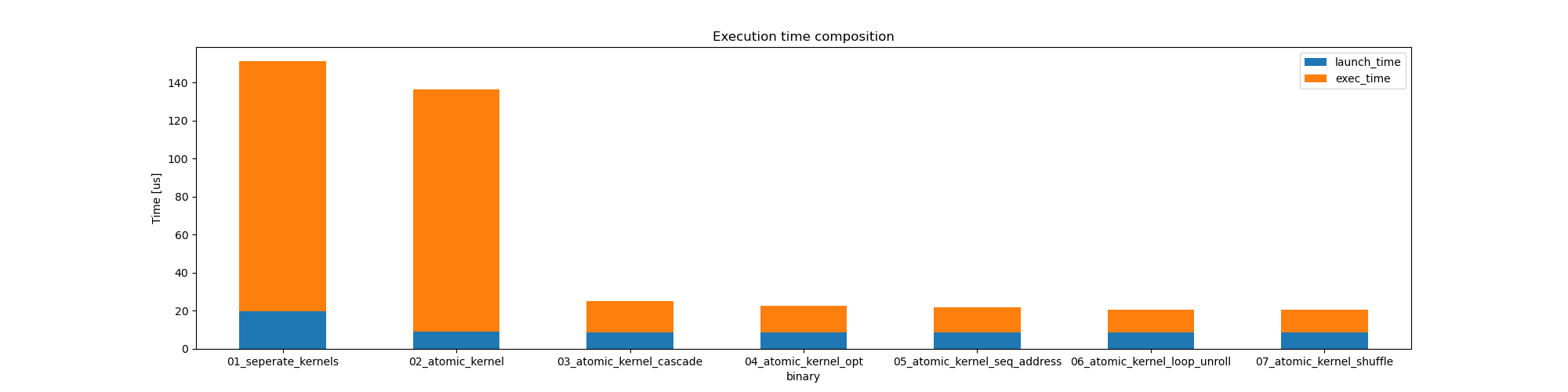


Abbildung : Zusammensetzung der Kernelzeit bei Problemgröße 2^21

Die dritte Performance-Maßnahme, Kaskadierung, zeigt den mit Abstand größten Effekt auf die Laufzeit. Auf die Auslastung der Threads bzw. die Vermeidung von schlafenden Threads sollte daher unbedingt geachtet werden. Die Anzahl Berechnungsvorgänge, die ein Thread vornimmt, bevor in den eigentlich Reduktionsbaum eingestiegen wird, wurde variiert. Die Ergebnisse sind in Abbildung 3 sichtbar. Auffällig ist hier das Treppenmuster bei Erhöhung der Problemgröße. Das anfängliche Plateau der blauen Linie, lässt sich einfach erklären. Jedem Block wurden immer 1024 Threads zugeteilt. Die Threads werden blockweise auf die Streaming-Multiprozessoren (SM) verteilt. Ein SM (der verwendeten Generation) kann 1536 Threads pro SM ausführen. Bei Problemgrößen kleiner gleich 2^16 werden maximal 32 Blöcke allokiert. Da die verwendete Grafikkarte 58 SMs besitzt, können alle Threads parallel laufen. Erst bei Problemgrößen größer als 2^16 werden mehr Blöcke allokiert als Ressourcen zur Verfügung stehen. Je nach zur Verfügung stehender GPU sollte die Block- und Gridgröße sowie die zu verrichtende Arbeit pro Thread mit Sorgfalt balanciert werden. Im Nachblick würde eine Blockgröße von 512 Threads für die hier verwendete Grafikkarte vermutlich eine bessere Performance liefern. Mit steigendem Kaskadierungslevel verschiebt sich der Zeitpunkt, an welchem nicht mehr genügend Hardware-Ressourcen zur parallelen Ausführung aller Threads bereitstehen. Der anfängliche Anstieg der Rechenzeit bei steigendem Kaskadierungslevel liegt in der While-Schleife (03\_atomic\_kernel\_cascade.cu, Zeile 24-28), welche für die Summierung vor Beginn des Reduktionsbaum, genutzt wird Die Anzahl der Schleifeniterationen ist nicht konstant, sondern steigt mit steigender Problemgröße bis das designierte Kaskadierungslevel (2, 4, …, 1024) erreicht ist.

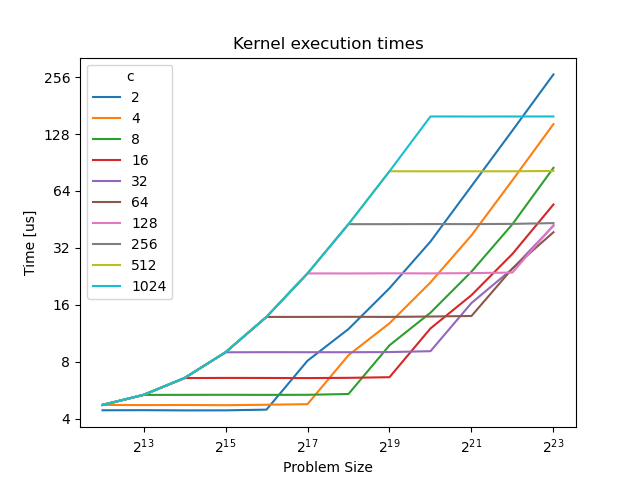


Abbildung : Vergleich der Ausführungszeit verschiedener Kaskadierungsstufen

Die weiteren Optimierungen darunter die Vermeidung des Modulo Operators, die sequenzielle Adressierung, das Loop-Unrolling sowie die Verwendung von Shuffle Operationen verbessern jeweils weiter die Performance. Dies zeigt, wie selbst kleine Änderungen deutlich messbare Verbesserungen bringen und der Kernelcode, welcher auf der Grafikkarte ausgeführt wird, ein besonders hohes Qualitätsniveau haben sollte. Nach Umsetzung aller Optimierungsmaßnahmen benötigt die Ausführung des Kernels nicht viel mehr Zeit als der Start des Kernels auf der Grafikkarte.