Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Reproduzierbarkeit bitgenauer Anwendungen

von Laura Wenderoth

Proseminar 2020 Softwareentwicklung in der Wissenschaft

Betreuerin: Petra Nerge

Abgabedatum: 31.08.2020

Inhaltsverzeichnis

1. Einleitung

Durch Fortschritt und Ausbau wurde in den letzten Jahrzehnten seit der Erfindung von Mikroprozessoren im Jahr 1971 die Leistung von Computern weiter vervielfacht. [1] Der Übergang von Skalarsystemen zu Vektorsupercomputern und die Innovation der Hardware hat es ermöglicht mit Hochleistungsrechnern komplexe nicht lineare Systeme zu simulieren.

Bei diesem Prozess hat die Parallelisierung der Berechnungen eine maßgebliche Rolle gespielt. Es kann aufgrund von parallelen Berechnungen Zeit eingespart werden, sodass die Kapazitäten vollständig genutzt werden können.

Neben einer guten Performance ist auch das Ergebnis von besonderer Bedeutung. Dieses sollte bei gleicher Eingabe über dem gleichen Algorithmus immer zum gleichen Ergebnis führen. Es kann jedoch zu Problemen der Reproduzierbarkeit kommen, die im Folgenden erläutert werden. Zunächst werden High performance computing und bitgenaue Reproduzierbarkeit definiert, anschließend wird eine Lösungsmöglichkeit – die deterministische Summenreduktion – vorgestellt.

2. High performance computing

In unserer digitalisierten Welt entstehen täglich unvorstellbar viele Daten. Um diese nutzen und verarbeiten zu können, bedarf es einer enormen Rechenleistung[1]. High performance computing (*zu Deutsch* Hochleistungsrechnen, *Abkürzung*: HPC) bietet die Möglichkeit sehr große Daten in einer adäquaten Zeit zu analysieren bzw. „Berechnungen, deren Komplexität oder Umfang eine Berechnung auf einfachen Arbeitsplatzrechnern unmöglich oder zumindest unsinnig macht“ [2] in einer sinnvollen Zeit zu berechnen.

Das Prinzip hinter high performance computing ist die Nutzung verteilter Rechenressourcen. So ist es möglich komplexe Probleme mit großen Datenmengen zu lösen. Das Schlüsselwort ist die Parallelität. Durch paralleles Berechnen können Berechnungen um ein Vielfaches beschleunigt werden. Sogenannte Supercomputer (*auch* Hochleistungsrechner) finden meistens Einsatz in der Wissenschaft und Forschung.

3. Reproduzierbarkeit bitgenauer Anwendungen

In der Wissenschaft gibt es viele Probleme, die durch Modellbeschreibungen näherungsweise gelöst oder simuliert werden. Ein Beispiel sind die Klimaberechnungen. Damit das Ergebnis einer solchen Berechnung einer Simulation für die Wissenschaft eine Bedeutung hat, muss immer das gleiche Ergebnis bei gleicher Eingabe herauskommen. Die Ergebnisse müssen reproduzierbar sein. Bei der Berechnung eines Problems spielt somit die Reproduzierbarkeit eine zentrale Rolle. Nur wenn ein Ergebnis der Berechnung bei gleicher Eingabe immer dasselbe ist, ist das es auch aussagekräftig.

3.1. Definition

Ahrens, Nguyen und Demmel definieren die Reproduzierbarkeit bitgenauer Anwendungen als das Erhalten von bitweise identischen Gleitkomma-Ergebnissen auch bei mehreren Durchläufen desselben Programms bei gleicher Eingabe. [3]

3.2. Wichtige Faktoren

Es gibt unterschiedliche Stellschrauben an denen gedreht werden kann, um Ergebnisse bitgenau reproduzierbar zu berechnen. Zunächst muss eine solide Hardware vorliegen, die immer die gleichen

[1] Rechenleistung: Anzahl der Berechnungen pro Zeiteinheit

Ergebnisse liefert. Es muss also sichergestellt sein, dass bei den grundlegenden logischen Verknüpfungen wie OR, XOR, NOT und AND bei gleicher Eingabe immer das gleiche Ergebnis herauskommt. Ebenso muss sichergestellt sein, dass es bei Datenübertragen nicht zu Fehlern kommt, bzw. diese gefunden und korrigiert werden.

Wenn das sichergestellt ist, ist die nächste Hürde die Kompilierung. Je nachdem, wie eine Berechnung interpretiert wird, also beispielsweise als (a+b)+c oder als a+(b+c), kann es zu Unterschieden im Ergebnis kommen.

An der Reproduzierbarkeit einer Berechnung sind die Hardware, die Laufzeitumgebung, die Kompilierung, die Programmiersprache, die Implementierung und der genaue Entwurf des Algorithmus beteiligt. [5]

3.3. Nutzen und Ziele

Ein Nutzen von reproduzierbaren Anwendungen liegt im Debuggen. Wenn man davon ausgehen kann, dass immer dasselbe Ergebnis bei gleicher Eingabe berechnet wird, können durch Vergleich leicht Fehler gefunden werden. Wenn das Ergebnis leicht variiert, erschwert das die Fehlerquellenfindung. [5]

Wenn wir reproduzierbar berechnen können, haben wir damit Möglichkeit nicht lineare Systeme genauer zu simulieren. Die Simulation von nicht linearen Systemen wird z.B. bei der Klima Forschung, bei der Wettervorhersage oder bei der molekulargenetischen Forschung genutzt.

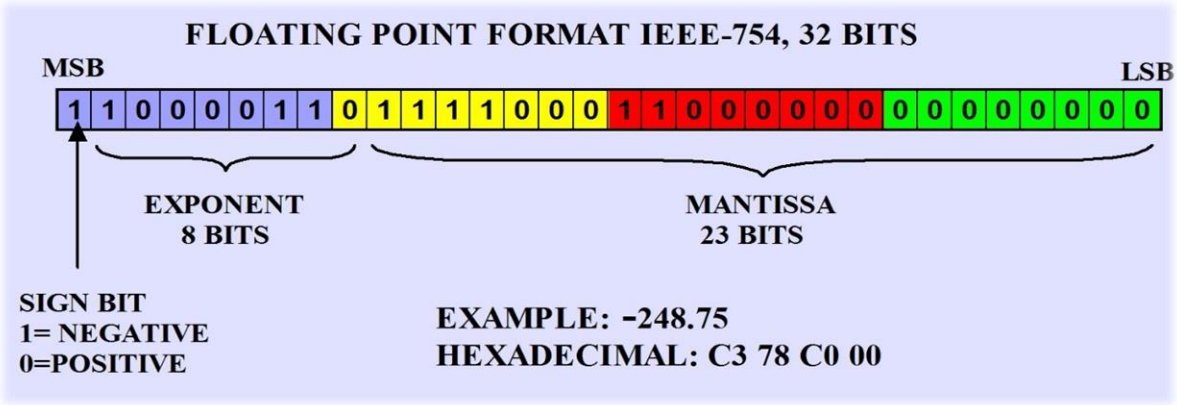
4. Problemdarstellung der Reproduzierbarkeit von Gleitkommazahlendarstellung bei HPC

Exkurs: IEEE-754 Standard und Gleitkommazahlen

Der IEEE-Standard ist eine Definition von Standarddarstellungen für binäre Gleichkommazahlen. Dieser wird bei vielen Berechnungen verwendet. Der große Vorteil ist, dass alle Rechenoperationen einschließlich Rundungsverfahren vollständig definiert und dokumentiert sind. [4]

Es gibt verschiedene Grundformate: 16 Bit (minifloat) – 32 Bit (single precision) – 64 Bit (double precision). Diese unterscheiden sich nur in der Größe. [4]

Eine Gleitkommazahldarstellung besteht aus dem Vorzeichen 1 Bit (0 = positiv; 1 = negativ), der Basis *b* (beim IEEE 754 Standard immer b =2), dem Exponenten *e* (*r* Bits) und der Mantisse *m* (*p* Bits). Da es sich um eine normalisierte Darstellung handelt, ist das erste Bit immer eine 1 und kann deswegen weggelassen werden.



[12]

Darüber hinaus gibt es noch die Darstellung *NaN: not a number*, um undefinierte Werte darzustellen. Dieser Wert wird verwendet, wenn beispielsweise durch null geteilt wird.

Normalerweise ist bei einer Addition Assoziativität[2] gegeben. Durch die Rundungen beim Rechnen mit Gleitkommazahlen kann diese jedoch nicht immer vorausgesetzt werden.

Um das zu verdeutlichen, betrachten wir dieses Beispiel: $2^{57} + 1 = 2^{57}$. Die Eins ist im Vergleich zu der 2^{57} vernachlässigbar klein und wird deswegen bei der Rundung nicht berücksichtigt. Das ist zunächst unproblematisch, da es keinen großen Unterschied zwischen $2^{57} + 1$ und 2^{57} gibt. Betrachten wir nun folgen Fall:

- $2^{57} + 1 - 2^{57} = 0$
- $2^{57} - 2^{57} + 1 = 1$

An diesem Beispiel wird deutlich, dass es einen Unterschied macht, ob erst addiert und dann gerundet wird oder erst subtrahiert wird. In diesem konkreten Fall erzeugen wir durch Umstellen andere Ergebnisse. Deswegen muss sich an Assoziativität gehalten werden, um immer bitgenau das gleiche Ergebnis zu erhalten.

Da keine Assoziativität angenommen werden kann, müssen alle Summanden nacheinander verrechnet und am Ende gerundet werden. Das hat zur Folge, dass die Berechnungen nicht parallelisiert werden können – zu Lasten einer verschlechterten Performance. Die Kernidee von HPC ist das Parallelisieren, sodass Berechnungen effizienter und schneller berechnet werden können.

Aufgrund der fehlenden Assoziativität bei dem Rechnen mit Gleitkommazahlen, können die Berechnungen entweder nur unter dem Verlust von Performance mit einer hohen Bitgenauigkeit (und damit Reproduzierbarkeit) oder mit guter Performance und einer geringeren Bitgenauigkeit (und damit verminderter Reproduzierbarkeit) dargestellt werden.

5. Lösungsansätze für Reproduzierbarkeit beim Rechnen mit Gleitkommazahlen

Es gibt zur Zeit gute Lösungsansätze um diesem Problem zu begegnen. Im Folgenden sind vier sehr wichtige Komponenten aufgezählt, die die Reproduzierbarkeit von Berechnungen bei guter Performance verbessern. [5]

- **Konsequente, vollständige Definition**: eine vollständige Definition ist wichtig für die Reproduzierbarkeit, da nur bei guter Definition von Rechnungen diese immer nach demselben Muster durchgeführt werden können. Wenn das gelingt, können die Ergebnisse auch bitgenau reproduzierbar sein.
- **Vermeiden von Rundungen**: Durch das Vermeiden von Rundungen werden Ungenauigkeiten vermieden. So können Fehler, wie am Beispiel oben beschrieben, nicht auftreten und erhöhen somit die Reproduzierbarkeit.
- **Lookup Tabellen (Bibliothek)**: In diesen sind für bestimmte Berechnungen Werte gespeichert, auf die nur zurückgegriffen werden muss. Aus diesem Grund sind diese dann nicht mehr anfällig für eventuelle Unterschiede nach Rundungen. Ein gutes Beispiel sind Kosinus- und Sinus-Funktionen. Für jeden Wert ist der entsprechende Funktionswert hinterlegt und muss nicht ausgerechnet werden.

[2] Definition Assoziativität: Einer Rechenoperation x: K x K → K heißt assoziativ, wenn für a,b,c ∈ K gilt:
a x (b x c) = (a x b) x c

- **Genaue Dokumentation** von mathematischen Funktionen der verschiedenen Programmiersprachen: Der erste Schritt ist die vollständige Definition von Rechenoperationen. Genauso wichtig ist die vollständige und genaue Dokumentation, damit ProgrammiererInnen wissen, wie die einzelnen Komponenten berechnet werden und welche Rechengesetze gelten.

Diese vier Komponenten verbessern die Reproduzierbarkeit. Das Vermeiden von Rundungen und auch Lookup Tabellen verschlechtern jedoch die Performance. Durch das Vermeiden von Rundungen wird in der Regel die Parallelisierung von Berechnungen erschwert. Bei dem Verwenden von Lookup Tabellen muss immer wieder auf den Speicher zugegriffen werden. Das wirkt sich auf die Performance aus. Je nachdem wie die Suche nach dem richtigen Funktionswert gestaltet ist, hat auch diese negativen Einflüsse auf die Performance einer Berechnung.

5.1. CUDA (Compute Unified Device Architecture) von NVIDIA

Ein Beispiel für eine High performance computing (HPC) Anwendung, mit der auch bitgenau reproduzierbare Berechnungen verbessert werden, ist CUDA. CUDA steht für Compute Unified Device Architecture. CUDA kann jedoch nicht nur bei HPC angewendet werden, sondern führt auch zu einer verbesserten Performance auf normalen Hauscomputern. [6]

Bei CUDA handelt es sich um eine parallele Computerplattform und ein Programmiermodell, das für GPU[3]-beschleunigte Anwendungen große Flexibilität und Performance bietet. Dabei wird in C für CUDA programmiert. Darüber hinaus gibt es auch Erweiterungen für C++ und Wrapper für weitere Programmiersprachen wie Java. [7] Bei CUDA sind alle Berechnungen vollständig definiert und auch dokumentiert, sodass beim Programmieren genau nachvollzogen werden kann, wie die einzelnen Komponenten der Berechnung durch CUDA verarbeitet/ berechnet werden.

CUDA macht sich den Umstand zu Nutze, dass normalerweise für Berechnungen nur CPUs[4] verwendet werden. Für die Parallelisierung stehen somit bei einem einfachen Hausrechner mit vier Kernen vier CPUs zur Verfügung. Im Vergleich zu diesem Vorgehen nutzt CUDA die zusätzliche Rechenkapazität der GPUs. Diese werden als Coprozessoren eingesetzt.

Somit stehen mehr Prozessoren für das gleichzeitige Ausführen von Berechnungen zur Verfügung. Weiteres Parallelisieren ist möglich, welches zu einer besseren Performance führt. CUDA wird zum einen in der Forschung angewendet, kann aber auch im privaten Rahmen nützlich sein bei der Verarbeitung großer Datenmengen, wie beispielsweise dem Schneiden und Verarbeiten von Videos. Viele Anwendungen wie beispielsweise Adobe, Microsoft, Autodest oder Dassault Systems benutzen CUDA. [8]

Durch das Verwenden der GPUs als Coprozessoren, kann durch Parallelisierung die Performance gesteigert werden. Darüber hinaus besitzt CUDA eine vollständige Definition mit Dokumentation, sowie Bibliotheken, die einfach importiert werden können. Dazu gehören auch Lookup Tabellen, die die Wahrscheinlichkeit der Reproduzierbarkeit einer Berechnung erhöhen. CUDA hat somit die meisten der oben vorgestellten möglichen Lösungsansätze umgesetzt.

[3] GPU (engl. *graphics processing unit*): Grafikprozessor
[4] CPU (engl. *central processing unit*) Hauptprozessor oder auch zentrale Verarbeitungseinheit

6. Problemlösung durch reproduzierbare Reduktion

Es gibt auf allen Ebenen von der Hardware bis zum Algorithmus Lösungsansätze. Die reproduzierbare Reduktion stellt einen möglichen Lösungsansatz auf Softwareebene dar. Im Folgenden wird der Algorithmus, der die reproduzierbare Reduktion beschreibt, vorgestellt. Dabei steht die Sicherstellung der Assoziativität beim Rechnen mit Gleitkommazahlen im Vordergrund. Der Algorithmus soll bei guter Performance bitgenaue Ergebnisse ermöglichen. Es handelt sich dabei um aktuelle Forschung, die im Gegensatz zu CUDA noch keinen Einzug in Anwendungen gefunden hat.

6.1. Definition Summenreduktion

Unter der Summenreduktion versteht man einen Algorithmus, der eine schnelle, nicht Ressourcen aufwendige Multiplikation von nicht vorzeichenbehafteten binären Zahlen beschreibt. Dieser besteht aus drei Schritten: 1) der Produktion von Summanden, 2) der Reduktion von Summanden und 3) der Addition der Summanden. Das Prinzip wird im Folgenden an einem Beispiel erklärt:

Die Binärzahlen a = 1100 und b = 1100 sollen miteinander multipliziert werden (a x b). Als erstes werden die Summanden berechnetet. Für jedes Bit von b mit dem Wert eins, wird ein neuer Summand kreiert, indem nach der letzten Stelle von a so viele Nullen angehangen werden, wie b nach der 1 weitere Bits besitzt (siehe Beispiel).

Beispiel:

1)
```
    1 1 0 0
  x 1 1 0 0
  ---------
    0 0 0 0
  0 0 0 0
1 1 0 0
1 1 0 0
```

2)
```
    1 1 0 0
  x 1 1 0 0
  ---------
1 1 1 0 0 0 0
  1 0 0 0 0 0
```

3)
```
  1 1 1 0 0 0 0
+   1 0 0 0 0 0
  ------------
    1
  1 0 0 1 0 0 0 0
```

Anschließend werden die Summanden reduziert: Alle von Null verschiedenen Bits werden in eine Zeile geschrieben, die Spalte der Eins bleibt dabei erhalten. Sollten sich in einer Spalte mehr als eine Eins befinden, werden diese in der gleichen Spalte in der darunterliegenden Zeile gespeichert. (Siehe 2) des Beispiels).

Der letzte Schritt ist die Addition der reduzierten Summanden. Das Ergebnis ist das Ergebnis der Multiplikation. In unserem Beispiel werden 1110000 mit 100000 addiert. Das Ergebnis der Beispielberechnung ist a x b = 10010000. [9]

6.2. Deterministische Summenreduktion: Double-Sweep-Algorithmus

Wir haben herausgearbeitet, dass Rundungen vermieden werden müssen, um bitgenau-Ergebnisse zu erhalten. Somit gilt beim Rechnen mit Gleitkommazahlen keine Assoziativität.

Das Prinzip des Double-Sweep-Algorithmus ist das abschließende Runden vor die Addition zu ziehen, sodass im Anschluss nach der Addition nicht mehr gerundet werden muss. Das Verfahren wurde durch Rump, Demmel and Nguyen [10], [11] entwickelt.

Für das Runden jedes einzelnen Summanden wird ein Extraktor bestimmt, mit dem der Summand vorgerundet wird. Um den Extraktor zu bestimmen wurde ein Algorithmus entwickelt:

$$M \geq n \cdot |v_i| \,/\, (1 - 2n\epsilon) \quad \forall\ 1 \leq i \leq n$$ [5]

- Extraktor *M*: größer als jede Partialsumme aller Summanden
- Genauigkeit ϵ
- *n* Summanden v_i

Nach dem der Extraktor bestimmt wurde, werden die Summanden nach folgendem Schema vorgerundet und anschließend addiert:

- Berechnung der vorgerundeten Werte (constributions)
  $q_i := (v_i \oplus M) \ominus M$
- Berechnung der Summe:
  $T := \sum_{i=1}^{n} q_i$

Das Verfahren wird im Folgenden an einem Beispiel erklärt:

$$M = 16 = 1 \cdot 2^4$$
$$v_1 = fl(3{,}16) = 1{,}1001010010 \cdot 2^1$$
$$v_2 = fl(1{,}73) = 1{,}1011101011 \cdot 2^0$$

Der Extraktor *M* ist groß genug gewählt für die beiden Summanden v_1, v_2 . Die Summanden sind hier normalisiert dargestellt. Als Gleitkommazahl würde in der Mantisse die erste Eins vor dem Komma nicht gespeichert sein.

Zur besseren Visualisierung sind die beiden Summanden graphisch dargestellt (1= dunkel, 0 =hell)

v_1 = 

v_2= 

Zur Veranschaulichung wird das Zahlenbeispiel an Dezimalzahlen vorgerechnet:

$$3{,}16 + 1{,}73 = 4{,}89$$
$$4{,}89_{10} = 100.111000\ldots_2$$

Nun werden die Summanden vorgerundet:

$v_1 + M = 1{,}0011001010 \cdot 2^4$
$q_1 = (v_1 + M) - M = 1{,}1001010000 \cdot 2^1$

$v_2 + M = 1{,}0001101110 \cdot 2^4$
$q_2 = (v_1 + M) - M = 1{,}1011100000 \cdot 2^0$

$v_1 + M$ = 

$v_2 + M$ = 

$q_1 = (v_1 + M) - M$ = 

$q_2 = (v_1 + M) - M$ = 

An der graphischen Darstellung wird ersichtlich, dass durch den ausreichend großen Extraktor die letzten Bits aus dem Speicherbereich geschoben werden, wenn der Extraktor zu dem Summanden addiert wird. Anschließend wird der Extraktor abgezogen und die Bits, die aus dem Speicherbereich geschoben worden sind, werden durch Nullen ersetzt. So werden die Summanden vorgerundet. An dieser Stelle kommt es durch das Runden zu einem Datenverlust. Es wird dadurch jedoch sichergestellt, dass nach der Addition nicht mehr gerundet werden muss.

Der letzte Schritt ist die Addition der vorgerundeten Summanden:

$$\begin{array}{rr} & 11,001010000 \\ + & 1,101110000 \\ \hline & 100,111000000 \end{array}$$

$$q_1 + q_2 = 1{,}001110000 \cdot 2^2$$

Wenn wir das normalisierte Ergebnis mit dem Ergebnis aus der Addition der gleichen Dezimalzahlen (*100.111000…*$_2$) vergleichen, zeigt sich, dass es identisch ist. Zudem zeigt sich, dass die 10 Bit der Mantisse auch nach der Normalisierung nicht überschritten werden, da es sich bei den Bit, die außerhalb des Speicherplatzes liegen, nur um Nullen handelt.

Der große Nachteil bei diesem Verfahren ist der relativ große Informationsverlust durch das Runden der Summanden vor der Addition. Um diesen auszugleichen, kann der Algorithmus wiederholt werden. Dazu werden die Reste aus den Summanden und den vorgeredeten Summanden berechnet.
$r_i = v_i - q_i$
Nun wir mit den Resten r_i der Double-Sweep-Algorithmus wiederholt. Das Ergebnis dieser Summe mit den Resten wird zu der ersten Summe hinzuaddiert.

Die Summe der originalen Gleitkommazahlen wird first-level extraction (Extraktion der ersten Ebene) genannt. Das Ergebnis der Rechnung mit den Resten second-level extraction (Extraktion der zweiten Ebene).

Durch den Double-Sweep-Algorithmus werden die Summanden einer Summe, dargestellt als Gleitkommazahl, mit einem geeignet großen Extraktor vorgerundet. Diese vorgerundeten Zahlen können dann in beliebiger Reihenfolge summiert werden, da durch das Vorrunden Assoziativität beim Rechnen mit Gleitkommazahlen gegeben ist. Durch das Vorrunden sind die Summanden semantisch äquivalent zu Integer-Werten, die fehlerfrei summiert werden können.

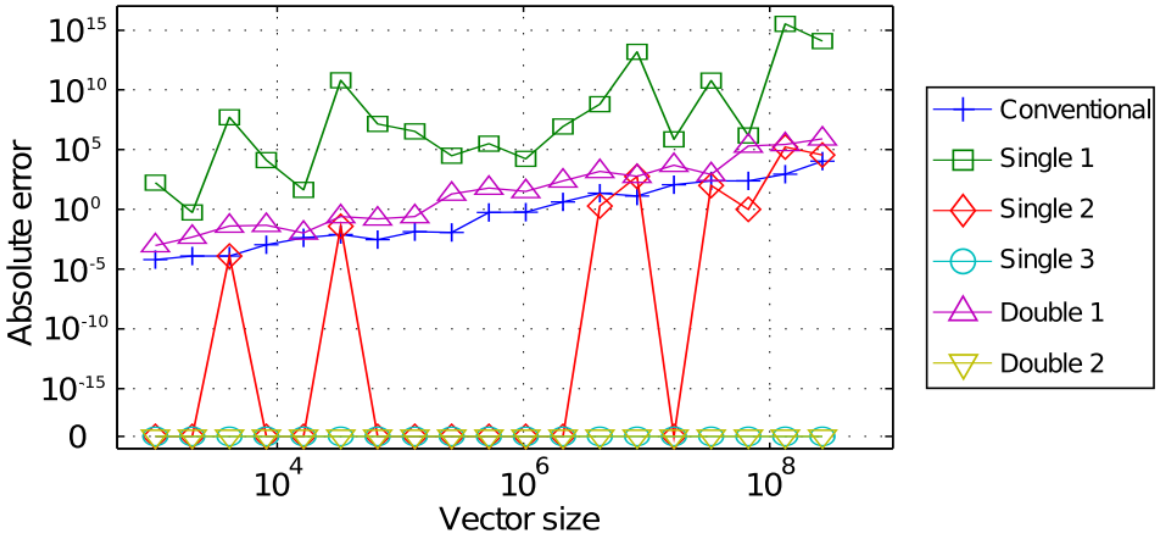
Es kommt zu einem Datenverlust durch das Vorrunden der Summanden, der durch Wiederholung des Algorithmus mit den Resten ausgeglichen werden kann. Es werden jedoch zwei Kommunikationsschritte benötigt. D.h. an zwei Stellen muss auf alle Summanden nacheinander zugegriffen werden können: das erste Mal zum Berechnen das Extraktors, das zweite Mal bei der Addition aller Summanden. Diese Kommunikationsschritte zwischen dem Prozessor und dem Speicher lassen sich nicht parallelisieren, sodass sie schlecht für die Performance der Berechnung sind.

6.3. Deterministische Summenreduktion: Single-Sweep-Algorithmus

Der Single-Sweep-Algorithmus [5] ist eine Weiterentwicklung des Double-Sweep-Algorithmus: Zur Performanceverbesserung soll es nur noch einen Kommunikationsschritt geben, indem der Extraktor nicht mehr durch Iterieren über alle Summanden bestimmt wird. Es wird somit kein gemeinsamer Extraktor am Anfang gebildet, sondern mit dem größtmöglichen Extraktor begonnen. Dadurch kommt es zu einem größerem Informationsverlust. Aufgrund des Formates ist die maximale Größe des Extraktor bestimmt. So kann der erste Kommunikationsschritt wegfallen.

Nun ist zu klären, ob der Single-Sweep-Algorithmus auch eine ähnliche Genauigkeit besitzt wie der Double-Sweep-Algorithmus. Durch den sehr groß gewählten Extraktor kommt es nämlich zu einem höheren Datenverlust.

In dem Paper „Designing Bit-Reproducible Portable High-Performance Applications“ [5] wurde der Single-Sweep-Algorithmus vorgestellt und mit dem Double-Sweep-Algorithmus verglichen. Das folgende Diagramm wurde aus dieser Publikation entnommen. Auf der y-Achse ist die Anzahl der Fehler dargestellt und auf der x-Achse die Größe der Vektoren.



[13]

Für konventionelle Verfahren (hier in blau dargestellt) zeigt sich, dass mit steigender Vektorgröße auch die Fehleranzahl erhöht. Die Fehlerrate des Double-Sweep-Algorithmus mit einer first-level-extraktion (pink) liegt leicht über der Fehlerrate konventioneller Verfahren. Deutlich zu erkennen ist, dass bei zweifachem Anwenden des Algorithmus (einmal mit den Summanden und einmal mit den Resten) die Fehlerrate näherungsweise bei Null liegt. Darin zeigt sich deutlich der Vorteil des Double-Sweep-Algorithmus gegenüber konventionellen Verfahren. Er ermöglicht bitgenaue reproduzierbare Ergebnisse.

Die Fehlerrate des Single-Sweep-Algorithmus mit einer first-level-extraktion (dunkelgrün) liegt stark über der Fehlerrate konventioneller Verfahren und der Fehlerrate des Double-Sweep-Algorithmus. Auch die second-level-extraction des Single-Sweep- Algorithmus ist bezogen auf die Fehlerrate nicht so gut wie die second-level-extraction des Double-Sweep-Algorithmus. Erst bei erneuter Anwendung des Algorithmus mit den Resten der Reste zeigt sich eine gleich geringe Fehlerrate.

Die Anwendung der Deterministischen Summenreduktion ermöglicht bitgenau reproduzierbare Ergebnisse zu berechnen – unabhängig ob es sich dabei um die Anwendung der die second-level-

extraction des Double-Sweep-Algorithmus handelt oder um die third-level-extraction des Single-Sweep-Algorithmus. Dabei können die Gleitkommazahlen nach dem Vorrunden in beliebiger Reihenfolge addiert werden, sodass diese auch parallel verrechnet werden können, ohne dass die Bitgenauigkeit des Ergebnisses gestört wird.

Die deterministische Summenreduktion hat jedoch noch keinen Einzug in bekannte Anwendungen gefunden und ist daher Gegenstand der Forschung im Bereich Algorithmus-Entwicklung für bitgenau reproduzierbare Ergebnisse bei HPC.

7. Ausblick der Anwendung und Forschung

Seit Jahrzehnten besteht das Bestreben bitgenaue Berechnungen auf Hochleistungsrechnern durchzuführen. CUDA ist ein Beispiel für eine Technik, die Einzug in verschiedene Anwendungen und Forschungsgebiete erhalten hat. Gerade im Bereich der Klima-Forschung und der Wettervorhersage müssen die Ergebnisse reproduzierbar sein, damit sie belastbar sind.

Aber auch in anderen Bereichen der Forschung, wie der Molekulardynamik oder Bioinformatik zum Berechnen von Proteinstrukturen und Interaktionen, der Astrophysik oder der Cyper Security. Auch im Finanzwesen wird auf bitgenaue Reproduzierbarkeit bei nicht-linearer Anwendungen zurückgegriffen, um beispielsweise Vorhersagen über den Finanzmarkt zu treffen.

Im Allgemeinen spielt das bitgenaue Berechnen bei vielen nicht-linearen Systemen mit großen Daten eine sehr wichtige Rolle. Das Grundproblem ist, dass sich Parallelisierung und damit HPC und bitgenaue Berechnungen oft entgegenstehen. Aus diesem Grund gibt es vielfältige Forschung zur Verbesserung bitgenauer Berechnungen bei HPC.

8. Zusammenfassung

High Performance Computing ermöglicht durch Parallelisierung eine sehr schnelle Berechnung von großen Datenmengen.

Bei dem Rechnen mit Gleitkommazahlen, auch wenn sie im IEEE-Standard dargestellt werden, kann keine Assoziativität sichergestellt werden. Es muss immer die gleiche Reihenfolge bei Berechnungen eingehalten werden, damit ein Ergebnis bitgenau reproduzierbar ist. Dieser Umstand ist nicht mit der Parallelisierung bei HPC vereinbar.

Die bitgenaue Reproduzierbarkeit ist jedoch in vielen Berechnungen in der aktuellen Forschung sehr wichtig, um belastbare Ergebnisse zu produzieren. In der Vorhersage von Wetter und Klima werden reproduzierbare Reduktionen für Berechnungen in nicht-linearen Systemen verwendet. Für die Berechnung werden sehr große Datenmengen benötigt, sodass HPC unverzichtbar ist. Jedoch ist zugleich ein reproduzierbares Ergebnis notwendig.

Aus diesem Grund gibt es verschiedene Ansätze auf allen Ebenen von Hardware bis zum Algorithmus, um dieses Problem zu lösen. Ein interessanter Ansatz auf Ebene der Software wurde in dieser Hausarbeit vorgestellt: die deterministische Summenreduktion. Mit dieser ist es möglich bei einer Summe alle Summanden vor zu runden, sodass anschließend Assoziativität gilt, obwohl die Summanden als Gleitkommazahlen dargestellt sind.

Die Forschung mit nicht linearen Systemen zur Vorhersage von z.B. Klima, Molekulardynamit o.ä. ist sehr aktuell und vielfältig, sodass auf Seiten der Informatik immer neue Ansätze entwickelt werden, um Genauigkeit und Performance miteinander zu verbinden.

9. Literaturverzeichnis

[1] Pawel Gepner, Michaá F. Kowalik: “Multi-Core Processors: New Way to Achieve High System Performance”, Proceedings of the International Symposium on Parallel Computing in Electrical Engineerin, 2006

[2] https://de.wikipedia.org/wiki/Hochleistungsrechnen [letzter Zugriff 21.06.20]

[3] Peter Ahrens, Hong Diep Nguyen, James W. Demmel: “Efficient Reproducible Floating Point Summation and BLAS”; EECS Department, University of California, Berkeley: Technical Report No. UCB/EECS-2015-229, 2015

[4] https://en.wikipedia.org/wiki/IEEE_Standards_Association [letzter Zugriff 29.06.20]

[5] A. Arteaga, O. Fuhrer, T. Hoefler: “Designing Bit-Reproducible Portable High-Performance Applications”; IEEE 28th International Parallesl and Distributed Processing Symposium, 2014

[6] https://developer.nvidia.com/hpc [letzter Zugriff 23.06.20]

[7] https://de.wikipedia.org/wiki/CUDA [letzter Zugriff 23.06.20]

[8] https://developer.nvidia.com/cuda-zone [letzter Zugriff 29.06.20]

[9] https://en.wikipedia.org/wiki/Reduction_of_summands

[10] S. M. Rump: “Ultimately fast accurate summation”, SIAM Journal on Scientific Comp., vol. 31, no. 5, p. 3466, 2009.

[11] J. Demmel and H. D. Nguyen: “Fast Reproducible FloatingPoint Summation”, in 21st IEEE Symp. on Computer Arithmetic, ser. ARITH’13, pp. 163–172, 2013

10. Bildverzeichnis

[12] http://blog.ruofeidu.com/exploration-ieee-754-floating-point-standard/

[13] A. Arteaga, O. Fuhrer, T. Hoefler: “Designing Bit-Reproducible Portable High-Performance Applications”; IEEE 28th International Parallesl and Distributed Processing Symposium, p. 1241, 2014