LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Grundlagenpraktikum: Rechnerarchitektur

Gruppe 233 – Abgabe zu Aufgabe A316
Sommersemester 2023

Ludwig Gröber Julian Pins Daniel Safyan

1 Einleitung

Die gegebene Aufgabenstellung A316 verlangt die Implementierung der Funktion f(x)=arsinh(x) im C17 Standard von C. Wie die angewandte Methodik und der mathematische Lösungsansatz aussieht, wird im Folgenden beschrieben.

1.1 Einführung 1/2Seite

Im Folgenden wird ein Problem aus dem Bereich der optimierung von Programmen behandelt. Diese Optimierung kann unter verschiedenen im Folgenden erläuterten Kriterien erfolgen. Die Optimierung vom Programmen ist im Allgemeinen immer ein Abwägen zwischen Implementierungsaufwand/Laufzeit in Zyklen/Laufzeit in Zeit/Energieaufwand/Portabilität/Speicherverbrauch und weiteren Anwendungsspezifischen Faktoren. Diese Abwägungen werden im Folgenden auch bei unserer Implementierung getroffen und und es wir versucht ein 'optimales' Ergebnis in verschiedenen Dimensionen zu erzielen, zu messen und zu bewerten. Hierbei dient uns der Vergleich als Bewertungskriterium und die bewerteten Dimensionen werden in den weiteren Kapiteln noch genauer spezifiziert. Eine Bewertung der Ergebnisse erfolgt am Ende des Kapitels Performanzanalyse.

Die Gruppe der hyperbolischen Funktionen wird für die Hankel-Transformation [**?**] , bei der Lösung bestimmter Differenzialgleichungen [**?**], bei der Beschreibung von Katenoiden [**?**] sowie in den Friedmann-Gleichungen für die zeitliche Entwicklung der Ausdehnung des Universums [**?**, **?**] benötigt. Für diese üblicherweise rechenintensiven Anwendungen ist eine effiziente und genaue Implementierung aller dafür verwendeten Funktionen essenziell um Gesamtergebnisse effizient erzielen zu können. Deshalb widmed sich diese Ausarbeitung im folgenden der Implementierung einer hyperbolischen Funktion, dem $arsinh$.

1.2 Aufgabenstellung analysiert und spezifiziert 1/2Seite

Die gestellte Aufgabe verlangt die numerische Berechnung der Funktion f(x) = arsinh(x) im C17 Standard von C.

Über den komplexen Zahlen $\mathbb{C}$ bildet die Funktion $arsinh$ mehrwertig ab, weshalb die Einschränkung auf die reellen Zahlen $\mathbb{R}$ getroffen wird. Die Funktion area sinus hyperbolicus bildet über $\mathbb{R}$ nach $\mathbb{R}$ ab und ist im Bereich $-\infty < x < +\infty$ definiert. Der

area sinus hyperbolicus kann über den Logarithmus, über ein Integral oder über den $sinh$ definiert werden.

$$(1)\,\operatorname{arsinh}(x) = \ln\left(x + \sqrt{x^2+1}\right)\ mit\, x \in \mathbb{R}$$

$$(2)\,\operatorname{arsinh}(x) = \int\_0^1 \frac{x}{\sqrt{x^2y^2+1}}\,\mathrm{d}y\, mit\, x \in \mathbb{R}$$

$$(3)\,\operatorname{arsinh}(x) = \frac{1}{sinh(x)}$$

Im Limit kann er durch die Funktion $f(x) \to \pm\ln(2|x|)$ angenähert werden. Als für die Implementierung relevante mathematische Eigenschaft die Punktsymmetrie zum Ursprung $(0,0)$ bereits hier zu erwähnen.

Die Familie der hyperbolischen Funktionen sind quadratische rationale Funktionen von der Exponentialfunktion $\exp$, die über die Mitternachtsformel gelöst werden könnten und mit dem natürlichen Logarithmus $\ln$ ausgedrückt werden.

Bei der Abbildung einer stetigen Funktion in ein endliches System müssen zudem Einschränkungen im Wertebereich sowie bei der Genauigkeit getroffen werden.

Letzteres wird im eigens dafür angelegten Kapitel "Genauigkeit"diskutiert, allgemein kann jedoch bereits zum Datentyp $double$ erwähnt werden, dass bei 52 Fragment-bits eine dezimale Genauigkeit von $53 * \log 10(2) \approx 15,96$ die Grenze des Datentyps darstellt. [**?**] Erstere Grenze gibt die Aufgabenstellung indirekt durch die doppelte Genauigkeit der Fließkommaarithmetik vor. Der Wertebereich der Implementierung ist ebenso durch den Datentypen $double$ beschränkt $\min(double) == 2,2250738585072014 * 10^308$ und $\max(double) == 1,7976931348623157 * 10^308$

Die Werte $\pm\infty$ und $\pm NaN$ sind bei $double$ besonders zu beachten. Da die Funktion streng monoton steigend ist, definieren wir:

$$(4)\,\operatorname{arsinh}(x) = \begin{cases} arsinh(x) & \text{falls } x < \infty \wedge x > -\infty \\ +NaN & \text{falls } x = +NaN \\ -NaN & \text{falls } x = -NaN \\ +\infty & \text{falls } x = +\inf \\ -\infty & \text{falls } x = -\inf \\ +NaN & \text{sonst} \end{cases}$$

Ein Problem bei $double$ ist zudem die Endianness, die nicht in IEEE 754 [**?**] spezifiziert ist. Es könnte also zu Fehlern zwischen Systemen kommen, die Behandlung dessen übersteigt den Umfang diese Ausarbeitung und wird deshalb aus dem Themenbereich ausgeschlossen.

Neben den sich aus dem Datentyp ergebenden Einschränkungen stellt die Aufgabenstellung ebenso Einschränkungen an arithmetischen Operatoren. In der ersten naiven Ausarbeitung dürfen nur grundlegende Berechnungen, also die vier Grundrechenarten

sowie shifts verwendet werden. In der weiterführenden Ausarbeitung sind komplexe Berechnungen wie exp oder log sowie SIMD, SSE zugelassen und weitere Optimierungen wie Loop-unrolling, Endrekursion, die Wahl anderer Algorithmen und Datenstrukturen erlaubt und erwünscht.

Von den beiden geforderten Vergleichsimplementierungen als Reihenentwicklung und als Tabellen-Lookup ist letztere bereits als eine Form der Optimierung zu verstehen. Und kann deshalb bereits mit der Reihe, ceteris paribus, vergleichen werden. Tabellen-Lookup scheint für die gegebene Aufgabe eine gute Optimierung zu sein, da für eine bekannte Menge an Funktionswerten häufige und identische Berechnungen durchgeführt werden. Die Wahl der größe eines Tabellen-Lookups ist eine Abwägung zwischen Laufzeit und Speicherplatz, dies wird im Folgendem Kapitel Öptimierungen"weiter erläutert.

Gemäß Aufgabenstellung sollen also drei C-Implementierungen erarbeitet und verglichen werden.

(A) Eine Implementierung als reine Reihendarstellung mit grundlegenden Berechnungen

(B) Eine Implementierung als Tabellen-Lookup mit grundlegenden Berechnungen

(C) Hauptimplementierung: Eine Implementierung nach freier Wahl mit komplexen Instruktionen

Im folgenden werden die Implementierungen mit den Buchstaben (A), (B) und (C) referenziert, die Formeln analog mit den Zahlen. A wird im Folgenden als naive Implementierung und als Benchmark für die Performanz verwendet.

Das Rahmenprogramm unterstützt die Funktionen $-V < Zahl >$ für die Wahl der Implementierung, $-B < Zahl >$ für die Wiederholungen des Funktionsaufrufs, $< FloatingPointZahl >$ für den Wert x, $-h$ oder $--help$ für die Beschreibung der Optionen des Programms und Verwendungsbeispiele hierfür.

Eine Implementierung in Assembly ist nicht gefordert und es wird in dieser Ausarbeitung auch darauf verzichtet, da der Fokus im Folgenden auf Genauigkeit und nicht auf Performanz liegen wird. Ebenso verzichten wir auf Mikro-Optimierungen, da es allgemein schwer ist moderne Compiler in Optimierungsaufgaben zu schlagen und der Code hierdurch schwerer zu lesen, fehleranfälliger und schwerer zu warten wird. Dies steht allgemein best-practices im Software-Engineering entgegen.

Intervalle werden größer an den Rändern, weil double precision nicht so genau ist. Relativer Fehler bleibt gleich an jeder Stelle.

2 Lösungsansatz

Leiten Sie mathematisch zwei Moglichkeiten her, die Funktion zu berechnen. Verwenden Sie dazu sowohl eine reine Reihendarstellung als auch eine Methodik, die einen Tabellen-Lookup benutzt.

Ideen: Reihenentwicklung nicht zur Runtime, sondern zur Compiletime berechnen. Runge Effekt: zwischen Integralgrenzen gut angenähert, wenn mit Polynom genähert wird. -> Deshalb funktioniert die Reihenentwicklung nicht. Lösung: Splines über großen Bereich mit x$^3 Polynomannhern. Intervallgleichverteilt. alphax^3 + betax^2 +$

$gammax + deltaAnnherungfrgroeWerte : x^2 + 1 = x^2 AbleitungperTailor - ReiheIdee :$
$1.ChilligerLookupTablemitInterpolation 2.SplinesLookupTableEntwicklungalsechteReihendarstellungn$

Grunstätzlich möglichst wenig zur Runtime berechnen. Reihe: Basisfunktionen + Lineares Gleichungssystem Lagrange Polynome Tailor-Reihe / Tailor Entweicklung

2.1 Naive Implementierung Reihenentwicklung]-1/1[

2.2 Naive Implementierung Tabellen-Lookup]-Inf/+Inf[

2.3 Vergleich der beiden Ansätze

2.4 Optimierte Implementierung Reihenentwicklung]-1/1[

Complexe Instruktionen, Wurzel, Log, Exponent, SIMD?

2.5 Optimierte Implementierung Tabellen-Lookup]-Inf/+Inf[

Idee: Hash-Map

2.6 Implementierung mit komplexen Instruktionen

3 Genauigkeit

Im gegebenen Kontext ist die Genauigkeit der Lösung als die Abweichung der Implementierung (4) vom Funktionswert der mathematisch definierten Funktion (1) zu verstehen.

4 Performanzanalyse

Die Performanz der Implementierungen wird anhand der Laufzeit gemessen. Die Laufzeit ist mit der Libaray $time.h$ gemessen worden. Hierbei sind einige Dinge vorab zu definieren.

4.1 Methodik und Annahmen 0,5Seite

Um optimale Vergleichbarkeit der Ergebnisse sicher zu stellen, versuchen wir eine möglichst exklusive nutzung der Maschine sicher zu stellen. Hierfür wurden alle nicht für das jeweilige Betriebssystem nötigen Prozesse beendet, alle Programme geschlossen, die Maschine an das Stromnetz angeschlossen und die Kühlung manuell auf die höchste Stufe gesetzt. Dies soll Performanceänderungen aufgrund von Überhitzung bei mehrfacher Ausführung vermeiden. Um einen stabilen Mittelwert zu erzielen, wird jede Implementierung mindestens 10 mal ausgeführt, nähere Angaben dazu sind der jeweiligen Messung zu entnehmen.

Alle Performaztests werden auf folgender Maschine ausgeführt: Name: $_{Prozessor:}$ $_{Takt:}$ $_{Multicore/SingleCoreSpeiche}$

Die Tests werden mit den Eingaben $_{ausgefhrtund}$ $_{malwiederholt.}$

Da es sich in der Aufgabe um eine numerische Berechnung handelt, erwarten wir bei den Performanztests lediglich Algorithmisch bedingte Abweichungen, welche durch die Laufzeitklassen beschrieben werden könnten. Implementierung A: $O(7)$ Implementierung B: $O(1)$ Implementierung C: $O(1)$

4.1.1 Zeitmesung der naiven Reihenentwicklung

4.1.2 Zeitmesung des naiven Tabellen-Lookup

4.1.3 Zeitmesung des optimierten Tabellen-Lookup

ziehen Sie als weiteren Vergleich eine C-Implementierung unter Nutzung von komplexeren Instruktionen, die beispielsweise eine Wurzelberechnung durchführen, heran.

4.2 Bewertung, Einordnung und Erklärung der Ergebnisse

5 Zusammenfassung und Ausblick

Die Wahl des Datentyp *double* bietet eine einfache Möglichkeit zur Erhöhung der Genauigkeit oder zur Verkleinerung des Speicherverbrauchs. Gemäß IEEE 754 [**?**] könnte auch die einfache oder vierfache Genauigkeit der Fließkommaarithmetik gewählt werden, wodurch sich die Genauigkeit auf $23 * \log 10(2) \approx 6,92$ dezimale Nachkommastellen bzw. $112 * \log 10(2) \approx 33,72$ dezimale Nachkommastellen respektive verändern würde. Der Speicherverbrauch würde sich bei einfacher Genauigkeit von 64 bit auf 32 bit halbieren, bei vierfacher Genauigkeit von 64bit auf 128bit verdoppeln. Hierbei ist je nach Anwendung die entsprechende Abwägung zu treffen.