

LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Grundlagenpraktikum: Rechnerarchitektur

Berechnung der Konstante π (A329)
Projektaufgabe – Aufgabenbereich Algorithmik

1 Organisatorisches

Auf den folgenden Seiten finden Sie die Aufgabenstellung zu Ihrer Projektaufgabe für das Praktikum. Die Rahmenbedingungen für die Bearbeitung werden in der Praktikumsordnung festgesetzt, die Sie über die Praktikumshomepage¹ aufrufen können.

Wie in der Praktikumsordnung beschrieben, sind die Aufgaben relativ offen gestellt. Besprechen Sie diese innerhalb Ihrer Gruppe und konkretisieren Sie die Aufgabenstellung. Die Teile der Aufgabe, in denen C-Code anzufertigen ist, sind in C nach dem C17-Standard zu schreiben.

Der **Abgabetermin** ist **Sonntag 05. Februar 2023, 23:59 Uhr (CEST)**. Die Abgabe erfolgt per Git in das für Ihre Gruppe eingerichtete Projektrepository. Bitte beachten Sie die in der README.md angegebene Liste von abzugebenden Dateien.

Die **Abschlusspräsentationen** finden in der Zeit vom **14.03.2023 – 24.03.2023** statt. Weitere Informationen werden noch bekannt gegeben. Beachten Sie, dass die Folien für die Präsentation am obigen Abgabetermin im PDF-Format abzugeben sind und keine nachträglichen Änderungen akzeptiert werden können.

Bei Fragen/Unklarheiten in Bezug auf den Ablauf und die Aufgabenstellung wenden Sie sich bitte an Ihren Tutor.

Wir wünschen Ihnen viel Erfolg und Freude bei der Bearbeitung Ihrer Aufgabe!

Mit freundlichen Grüßen
Die Praktikumsleitung

¹<https://gra.caps.in.tum.de>

2 Berechnung der Konstante π

2.1 Überblick

Die Algorithmik ist ein Teilgebiet der Theoretischen Informatik, welches wir hier unter verschiedenen praktischen Aspekten beleuchten: Meist geht es um eine konkrete Frage- oder Problemstellung, welche durch mathematische Methoden beantwortet oder gelöst werden kann. Sie werden im Zuge Ihrer Projektaufgabe ein Problem lösen und die Güte Ihrer Lösung wissenschaftlich bewerten.

2.2 Karazuba-Multiplikation

Wenngleich die Multiplikation zweier 64-Bit Zahlen auf einem 64-Bit Prozessor einfach und effizient in Hardware realisierbar ist, so ist dies bei größeren Zahlen nicht der trivialerweise der Fall. Eine effiziente Berechnung bei größeren Zahlen erfordert daher zusätzlichen Aufwand.

Die naive Multiplikation zweier Zahlen mit je n Bits hat eine Laufzeitkomplexität von $\Theta(n^2)$, diese lässt sich mittels Divide-and-Conquer allerdings weiter reduzieren; beispielsweise mittels des Karazuba-Algorithmus [3] auf $\mathcal{O}(n^{1.59})$. Hierbei werden die Faktoren x und y in zwei gleich große Teile von je m Bits aufgeteilt, $x = x_0 + 2^m x_1$ und $y = y_0 + 2^m y_1$. Die Berechnung des Produkts sieht dann wie folgt aus:

$$\begin{aligned} x \cdot y &= (x_0 + 2^m x_1)(y_0 + 2^m y_1) \\ &= x_0 y_0 + 2^m (x_0 y_1 + x_1 y_0) + 2^{2m} x_1 y_1 \\ &= x_0 y_0 + 2^m ((x_0 + x_1)(y_0 + y_1) - x_0 y_0 - x_1 y_1) + 2^{2m} x_1 y_1 \end{aligned}$$

Auf diese Weise wird die erforderliche Anzahl der Teil-Multiplikation von vier auf drei reduziert – es müssen lediglich die Multiplikationen $x_0 y_0$, $x_1 y_1$ und $(x_0 + x_1)(y_0 + y_1)$ berechnet werden. Durch eine rekursive Anwendung des Algorithmus bis m effizient durch vorhandene CPU-Instruktionen berechenbar ist, lassen sich auch große Zahlen effizienter² multiplizieren.

2.3 Binary Splitting

Viele mathematische Folgen lassen sich in einer Weise formulieren, die sich mittels *Binary Splitting* [2] effizienter berechnen lassen. Sei S_{n_1, n_2} für $n_1, n_2 \in \mathbb{N}$ mit $n_1 < n_2$ eine Folge der folgenden Form, und seien $a(n)$, $b(n)$, $p(n)$ und $q(n)$ Polynome mit ganzzahligen Koeffizienten:

$$S_{n_1, n_2} = \sum_{n_1 \leq n < n_2} \frac{a(n)}{b(n)} \cdot \frac{p(n_1) \cdots p(n)}{q(n_1) \cdots q(n)} = \sum_{n=n_1}^{n_2-1} \frac{a(n)}{b(n)} \prod_{k=n_1}^n \frac{p(k)}{q(k)}$$

Zur Berechnung werden folgende Hilfst Terme mit $n_m = \lfloor \frac{n_2+n_1}{2} \rfloor$ rekursiv definiert:

$$P_{n_1, n_2} = p(n_1) \cdots p(n_2 - 1) = \begin{cases} p(n_1) & \text{if } n_1 = n_2 - 1 \\ P_{n_1, n_m} P_{n_m, n_2} & \text{otherwise} \end{cases}$$

²Es gibt auch effizientere Verfahren mit geringerer asymptotischer Laufzeit, siehe z.B. [4].

$$Q_{n_1, n_2} = q(n_1) \cdots q(n_2 - 1) = \begin{cases} q(n_1) & \text{if } n_1 = n_2 - 1 \\ Q_{n_1, n_m} Q_{n_m, n_2} & \text{otherwise} \end{cases}$$

$$B_{n_1, n_2} = b(n_1) \cdots b(n_2 - 1) = \begin{cases} b(n_1) & \text{if } n_1 = n_2 - 1 \\ B_{n_1, n_m} B_{n_m, n_2} & \text{otherwise} \end{cases}$$

$$T_{n_1, n_2} = B_{n_1, n_2} Q_{n_1, n_2} S_{n_1, n_2} = \begin{cases} a(n_1) p(n_1) & \text{if } n_1 = n_2 - 1 \\ B_{n_m, n_2} Q_{n_m, n_2} T_{n_1, n_m} + B_{n_1, n_m} P_{n_1, n_m} T_{n_m, n_2} & \text{otherwise} \end{cases}$$

Damit ergibt sich:

$$S_{n_1, n_2} = \frac{T_{n_1, n_2}}{B_{n_1, n_2} Q_{n_1, n_2}}$$

Für die Berechnung einer Folge wählt man $n_1 = 1$ und $n_2 \rightarrow \infty$. Man beachte, dass nahezu die gesamte Berechnung nur mit Ganzzahlarithmetik und ohne Divisionsoperationen ausgeführt wird; lediglich im letzten Schritt wird eine Fließkommadivision durchgeführt. Als Optimierung der Berechnung lassen sich die Terme auch bei kleinen Differenzen $n_2 - n_1$ direkt ohne Rekursion berechnen.

2.4 Funktionsweise

Die mathematische Konstante π lässt sich wie folgt berechnen [1]:

$$\pi = 2 + \sum_{k=1}^{\infty} \frac{k!^2 2^{k+1}}{(2k+1)!}$$

Ihre Projektaufgabe ist die Approximation der mathematischen Konstante π mit beliebiger, konfigurierbarer Genauigkeit effizient zu berechnen. Verwenden Sie zur Berechnung der Summe das oben beschriebene *Binary Splitting* und wählen Sie hierzu die Polynome wie folgt:

$$\begin{aligned} a(n) &= 2 \\ b(n) &= 1 \\ p(n) &= n \\ q(n) &= 2n + 1 \end{aligned}$$

2.5 Aufgabenstellungen

Ihre Aufgaben lassen sich in die Bereiche Konzeption (theoretisch) und Implementierung (praktisch) aufteilen. Sie können (müssen aber nicht) dies bei der Verteilung der Aufgaben innerhalb Ihrer Arbeitsgruppe ausnutzen. Antworten auf konzeptionelle Fragen sollten an den passenden Stellen in Ihrer Ausarbeitung in angemessenem Umfang erscheinen. Entscheiden Sie nach eigenem Ermessen, ob Sie im Rahmen Ihres Abschlussvortrags auch auf konzeptionelle Fragen eingehen. Die Antworten auf die Implementierungsaufgaben werden durch Ihren Code reflektiert.

2.5.1 Theoretischer Teil

- Zeigen Sie, warum die Formulierung auf Basis der Polynome a , b , p und q identisch zur ursprünglichen Formulierung ist.
- Überlegen Sie, wie Sie die benötigten Operationen der Integer-Arithmetik (Addition und Multiplikation) mit beliebiger Genauigkeit effizient realisieren können.
- Leiten Sie ein effizientes Verfahren für die Division her, beispielsweise das *Newton-Raphson*-Verfahren. Warum sind Fließkommazahlen nach IEEE-754 *nicht* geeignet?
- Definieren Sie eine Datenstruktur `struct bignum`, um Ganzzahlen beliebiger Größe zu speichern, sowie ein Format, um das Berechnungsergebnis als Fixkommazahl zu speichern.
- Wie können Sie die Berechnung mittels SIMD optimieren?
- Wie lange braucht Ihr Programm, um die ersten 10^6 Nachkommastellen zu berechnen?

2.5.2 Praktischer Teil

- Implementieren Sie im Rahmenprogramm I/O-Operationen in C, mit welchen der Benutzer die Konstante in wählbarer Genauigkeit berechnen und wahlweise in dezimaler oder hexadezimaler Darstellung ausgeben lassen kann.
- Implementieren Sie in der Datei mit Ihrem C-Code die Funktion:

```
struct bignum comp_pi(size_t s)
```

Diese soll $\frac{T(0,n)}{B(0,n)Q(0,n)} \approx \pi$ als Fixpunkt-Zahl mit s binären Nachkommastellen berechnen. Dabei soll n so gewählt werden, dass das Ergebnis auf s binäre Nachkommastellen genau ist.

2.5.3 Rahmenprogramm

Ihr Rahmenprogramm muss bei einem Aufruf die folgenden Optionen entgegennehmen und verarbeiten können. Wenn möglich soll das Programm sinnvolle Standardwerte definieren, sodass nicht immer alle Optionen gesetzt werden müssen.

- `-V <Zahl>` — Die Implementierung, die verwendet werden soll. Hierbei soll mit `-V 0` Ihre Hauptimplementierung verwendet werden. Wenn diese Option nicht gesetzt wird, soll ebenfalls die Hauptimplementierung ausgeführt werden.
 - `-B <Zahl>` — Falls gesetzt, wird die Laufzeit der angegebenen Implementierung gemessen und ausgegeben. Das *optionale* Argument dieser Option gibt die Anzahl an Wiederholungen des Funktionsaufrufs an.
-

- `-d <Zahl>` — Ausgabe von n dezimalen Nachkommastellen. Hat Vorrang vor `-h`
- `-h <Zahl>` — Ausgabe von n hexadezimalen Nachkommastellen
- `-h` — Eine Beschreibung aller Optionen des Programms und Verwendungsbeispiele werden ausgegeben und das Programm danach beendet.
- `--help` — Eine Beschreibung aller Optionen des Programms und Verwendungsbeispiele werden ausgegeben und das Programm danach beendet.

Sie dürfen weitere Optionen implementieren, beispielsweise um vordefinierte Testfälle zu verwenden. Ihr Programm muss jedoch nur unter Verwendung der oben genannten Optionen verwendbar sein. Beachten Sie ebenfalls, dass Ihr Rahmenprogramm etwaige Randfälle korrekt abfangen muss und im Falle eines Fehlers mit einer aussagekräftigen Fehlermeldung auf `stderr` und einer kurzen Erläuterung zur Benutzung terminieren sollte.

2.6 Allgemeine Bewertungshinweise

Beachten Sie grundsätzlich alle in der Praktikumsordnung angegebenen Hinweise. Die folgende Liste konkretisiert einige der Bewertungspunkte:

- Stellen Sie unbedingt sicher, dass *sowohl* Ihre Implementierung *als auch* Ihre Ausarbeitung auf der Referenzplattform des Praktikums (1xhalle) kompilieren und vollständig korrekt bzw. funktionsfähig sind.
- Die Implementierung soll mit GCC/GNU as kompilieren. Verwenden Sie keinen Inline-Assembler. Achten Sie darauf, dass Ihr Programm keine x87-FPU- oder MMX-Instruktionen und SSE-Erweiterungen nur bis SSE4.2 verwendet. Andere ISA-Erweiterungen (z.B. AVX, BMI1) dürfen Sie nur benutzen, sofern Ihre Implementierung auch auf Prozessoren ohne derartige Erweiterungen lauffähig ist.
- Sie dürfen die angegebenen Funktionssignaturen (nur dann) ändern, wenn Sie dies (in Ihrer Ausarbeitung) begründen.
- Verwenden Sie die angegebenen Funktionsnamen für Ihre Hauptimplementierung. Falls Sie mehrere Implementierungen schreiben, legen wir Ihnen nahe, für die Benennung der alternativen Implementierungen mit dem Suffix „_V1“, „_V2“ etc. zu arbeiten.
- Denken Sie daran, das Laufzeitverhalten Ihres Codes zu testen (Sichere Programmierung, Performanz) und behandeln Sie *alle möglichen Eingaben*, auch Randfälle. Ziehen Sie ggf. alternative Implementierungen als Vergleich heran.
- Eingabedateien, welche Sie generieren, um Ihre Implementierungen zu testen, sollten mit abgegeben werden; größere Eingaben sollten stattdessen stark komprimiert oder (bevorzugt) über ein abgegebenes Skript generierbar sein.

- Stellen Sie Performanz-Ergebnisse nach Möglichkeit grafisch dar.
- Vermeiden Sie unscharfe Grafiken und Screenshots von Code.
- Geben Sie die Folien für Ihre Abschlusspräsentation im PDF-Format ab. Achten Sie auf hinreichenden Kontrast (schwarzer Text auf weißem Grund!) und eine angemessene Schriftgröße. Verwenden Sie 16:9 als Folien-Format.

Literatur

- [1] M. Beeler, R. W. Gaspel und R. Schroepel. *HAKMEM*. <https://dspace.mit.edu/bitstream/handle/1721.1/6086/AIM-239.pdf>. Feb. 1972.
 - [2] Bruno Haible und Thomas Papanikolaou. „Fast multiprecision evaluation of series of rational numbers“. In: *International Algorithmic Number Theory Symposium*. <https://www.ginac.de/CLN/binsplit.pdf>. 1998, S. 338–350.
 - [3] Anatolii Alekseevich Karatsuba und Yu P. Ofman. „Multiplication of many-digital numbers by automatic computers“. In: *Doklady Akademii Nauk*. Bd. 145. 2. Russian Academy of Sciences. 1962, S. 293–294.
 - [4] Donald E. Knuth. *Art of Computer Programming, Volume 2: Seminumerical Algorithms*. 3. Aufl. Addison-Wesley Professional, 2014.
-