Parallelisierung der Leibniz-Reihe zur Approximation von PI

Alexander Herrmann  
*9859538*  
*DHBW Ravensburg*Friedrichshafen, Deutschland  
herrmann.alexa-tfe18@it.dhbw-ravensburg

Johannes Ruffer  
*1011921  
DHBW Ravensburg*Friedrichshafen, Deutschland  
ruffer.johann-tfe18@it.dhbw-ravensburg.de

Serkant Soylu  
*9964027*  
*DHBW Ravensburg*Friedrichshafen, Deutschland  
soylu.serkant-tfe18@it.dhbw-ravensburg.de

*Abstract*—Das hier ist der Abstract. BLABLABLA (nicht zwingend notwendig)

Keywords—informatics, parallel machines, parallel processing, parallel algorithms, approximation algorithms, approximation methods

# Einleitung

In der heutigen Software gibt es die verschiedensten Methoden Programme zu optimieren und deren Laufzeiten zu verkürzen bzw. zu optimieren. Das Ziel der Vorlesung Informatik IV ist es die Methode der Parallelisierung von Programmen kennenzulernen. Dafür sollen die Vor- und Nachteile in einem eigenen Programmentwurf untersucht und dokumentiert werden.

Auf der BW Cloud ist hierzu ein Netzwerk anzulegen, welches mehrere Instanzen verknüpft. Mithilfe von PUTTY sollen SSH-Keys erstellt werden, mithilfe derer auf die verschiedenen Instanzen zugegriffen werden kann.

In der vorliegenden Arbeit wird die Methode der Parallelisierung anhand der PI-Approximation beschrieben.

# MPI

Message Passing Interface (MPI) ist ein Standard für die Implementierung von parallelen Nachrichtenweitergaben und Hochleistugsrechnungen (HPC). Das Message Passing ist also ein Kommunikationsansatz, welcher einen Datenaustausch zwischen Prozessen ermöglicht, welche auf einem oder mehreren Rechenknoten ausgeführt werden. Dazu wird die Aufgabe in mehrere Teilaufgaben zerlegt. Der Vorteil an dieser Herangehensweise ist, dass die Verarbeitung parallelisiert, also auf verschiedene Rechner aufgeteilt, vorgenommen werden kann. (Shekhar S., 2017, S. 1390)

Zur Realisierung der oben genannten Parallelisierungen in den Programmiersprachen C, C++ und Fortran wird OpenMP als offene Programmierschnittstelle verwendet. Der Vorteil der Programmierschnittstelle ist, dass der ursprüngliche sequenzielle Quellcode, welcher auf einer Maschine ausgeführt wurde, im Vergleich zu konkurrierenden Ansätzen oft nur geringfügig geändert werden muss. (S. Hoffmann, 2008, S. 1f.)

# Varianten der PI-Approximation

Für die Approximation von PI können verschiedene Methoden herangezogen werden. Innerhalb dieser Arbeit wird dazu auf die Monte Carlo Methode und die Leibniz-Reihe eingegangen.

## Monte Carlo Methode

Die Monte Carlo Methode basiert im Unterschied zu anderen Verfahren der PI-Approximation nicht auf einer Integration oder einer Reihenbildung. Viel mehr wird dieses mathematische Problem mit Hilfe der Generierung von Zufallszahlen gelöst. [3]

Im ersten Schritt wird hierfür ein Quadrat mit einer festen Kantenlänge definiert. Anschließend wird ein Viertelkreisbogen mit dem Radius der genannten Kantenlänge in das Quadrat eingezeichnet. Mithilfe eines Zufallsgenerators werden nun, wie in Abbildung 1 veranschaulicht, Punkte in die Fläche eingezeichnet.

Auffällig ist, dass manche Punkte innerhalb und andere außerhalb des Viertelkreisbogens liegen. Nun muss das Verhältnis zwischen der Anzahl von Punkten im inneren zur Anzahl von Punkten im äußeren Teil des Viertelkreises betrachtet werden.

|  |  |
| --- | --- |
|  | (1) |

Für die Kreisfläche gilt auch:

|  |  |
| --- | --- |
|  | (2) |

|  |  |
| --- | --- |
|  | (3) |

Umgeformt nach PI ergibt es:

Das Ergebnis ist ein Maß für die Fläche des Viertelkreises und somit ein Viertel der Kreiszahl PI. Je mehr Punkte generiert werden, desto gleichmäßiger ist die Verteilung der Punkte und die Genauigkeit der genäherten Zahl von PI. [3]

Das Problem an dieser Methode ist die Aufteilung auf mehrere Instanzen, da durch die zufällig genierten Punkte keine genaue Unterteilung möglich bzw. nur mit einem sehr hohen Aufwand erreichbar ist. (näher beschreiben & erklären)

## Leibniz-Reihe

Ein weiteres Verfahren zur Approximation von PI ist die in Formel (4) beschriebene Leibniz-Reihe. Wie bei der Monte Carlo Methode wird auch hier ein Viertel von PI berechnet. Dies wird mithilfe einer Summenreihe realisiert. Das Indizes *k* wird von *k* = 0 bis *k* = hochgezählt und ist somit die Anzahl an Iterationsschritten für das Programm. Hierbei gilt: Je höher die Anzahl von Schleifendurchläufen, desto genauer ist das Ergebnis.

|  |  |
| --- | --- |
|  | (4) |

Auffällig ist dabei, dass der Zähler durch die Potenzierung jede ungerade Potenz ein negatives Vorzeichen bekommt. Es wird sich somit von beiden Seiten an PI angenähert.

|  |  |
| --- | --- |
|  | (5) |

Die Methode PI mithilfe der Leiniz-Reihe zu berechnen ist daher gut für eine Parallelisierung geeignet, da es sich um eine Reihendarstellung bzw. Reihenform handelt. Für die Parallelisierung des Verfahrens auf mehrere Instanzen können zwei verschiedene Strategien verfolgt werden:

### Autfeilung der Leibniz-Reihe in gleich große Abschnitte

In den ersten Überlegungen, zur Umsetzung der parallelisierten Approximation von PI, kam man zu der Idee den Zahlenbereich der Iterationen auf die jeweiligen Instanzen innerhalb des Netzwerkes aufzuteilen. Nach der Initialisierung der Nodes, würde der Master die Anzahl der zu durchlaufenden Schleifen, in Abhängigkeit der vorhandenen Teilnehmern, aufteilen. Jeder Bereich wird hierbei mittels einer MPI Nachricht gesendet. Dafür ist es notwendig auf Senderseite die MPI\_Send Funktion aufzurufen und auf Empfängerseite die dazugehörige Empfangsfunktion MPI\_Recv. Sind die Nachrichten ausgesendet beginnen die Instanzen mit ihren Berechnungen, sind diese abgeschlossen wird auch, für das Zusammenführen der Ergebnisse, ein Nachrichtenprotokoll implementiert. Problematisch hierbei ist, dass durch die Aufteilung des Zahlenbereiches mit der beschriebenen Strategie die Programmlaufzeiten wohlmöglich sehr unterschiedlich ausfallen werden. Denn die Instanz mit dem ersten Zahlenpaket hat die kleinsten Zahlen und wird somit schneller mit der Berechnung fertig, wohingegen der nächste Zahlenbereich schon erheblich größer ist und somit auch mehr Rechenzeit in Anspruch nimmt. Dazu kommt noch die Implementierung von zwei Nachrichtenprotokollen, die das Programm verlangsamen, da es sich um blockierende Nachrichten handelt. In dieser Art der Kommunikation pausiert das Programm an der Stelle des Sendens seiner eigenen Informationen, bis die dafür vorgesehene Empfangsfunktion aufgerufen wird und die Kommunikation abgeschlossen wurde.

### Schrittweise Aufteilung der Leibniz-Reihe

Anhand der in der vorherigen Methode aufgetretenen Bedenken bezüglich der Laufzeit galt es eine neue Strategie zu entwickeln. Einerseits müsste der Zahlenbereich gleichmäßiger verteilt werden, andererseits sollten die benötigten Nachrichten zum Informationsaustausch möglichst geringgehalten werden.

Im ersten Schritt wird die Aufteilung der Iterationen verbessert. Dabei helfen die Eigenschaften des MPI Netzwerkes. Denn mit dem Start des Programmcodes wird jeder einzelne Teilnehmer initialisiert und einem Rang zugewiesen. Nun lässt man die *for*-Schleife auf jeder Instanz in Abhängigkeit der Größe des Netzwerkes hochzählen und nicht mehr wie in der vorherigen Methode um eins. Den Startpunkt zur Berechnung definiert der Rang einer Node. Dadurch wird der Zahlenbereich gleichmäßiger aufgeteilt, was zu ähnlichen Laufzeiten der Programme führt. Auch die gesamtheitliche Berechnungszeit verringert sich, denn das Programm kann maximal so schnell sein wie die langsamste Instanz in einem Netzwerk. Eine zusätzliche Verbesserung der Laufzeit ergibt sich durch die Ersparnis eines Nachrichtenprotokolls. Die Definition der Iterationsschleifen durch Rang und Netzwerkgröße erübrigt die Aufteilung des Zahlenbereiches durch den Master. Somit kann die Anzahl der zu sendenden Nachrichten pro Programm, im Vergleich zur vorherigen Parallelisierungsmethode, auf eine reduziert werden.

# Programm-Ablauf

Bei der Implementierung zur Parallelisierung muss beachtet werden, dass auf jeder Instanz die gleiche .cpp-Datei ausgeführt wird. Daher ist es nötig eine Abhängigkeit zwischen dem Programmcode und der durch MPI entstehenden Netzwerktopologie herzustellen.

Zu Beginn wird eine Funktion für die Berechnung von PI definiert. Übergabeparameter für die Funktion sind *step\_size* und *start\_point*, diese beschreiben Schrittweite und Startpunkt einer *for*-Schleife. Innerhalb der Iterationsschleife ist die Leibniz-Reihe in Codezeilen übersetzt worden. Mit jedem Durchlauf der Schleife wird ein weiterer Summand an die Reihe angefügt, bis die Abbruchbedingung erfüllt wird. Nach Abbruch der Schleife wird *sum*, die Summe der einzelnen Summanden, als Rückgabewert hergenommen.

In der *main-*Funktion werden für jeden Worker die Variablen *size* und *rank* definiert, diese werden hergenommen, um den Programmablauf individuell zu steuern. Zugewiesen werden die Werte mittels Funktionen aus der *mpi.h*-Bibliothek. Mit einer *if*-Abfrage, soll evaluiert werden, ob es sich bei der jeweiligen Instanz um einen Master oder Slave handelt.

Im Falle eines Slaves wird die Funktion zur Berechnung der Leibniz-Reihe aufgerufen. Das Ergebnis wird dann mittels *MPI\_Send* an den Master zurückgesendet.

Zur Nachrichtendefinition benötigt man einen Pointer, der auf die zu sendenden Daten zeigt und die benötigte Elementen-Anzahl für die Nachricht. Außerdem muss der Datentyp, der Rang der Zielinstanz, sowie eine Identifikationsnummer der Nachricht und der Bezeichner für das Netzwerk angegeben werden.

Handelt es sich um einen Master prüft dieser, ob sich noch andere Instanzen in diesem Netzwerk befinden. Ist dem nicht so führt er die Berechnung der Leibniz-Reihe selbstständig ohne Parallelisierung durch. Sind andere Teilnehmer vorhanden, wird die *calc\_PI*-Funktion mit den entsprechenden Argumenten aufgerufen. Nach der eigenen Berechnung müssen an dieser Stelle *MPI\_Recv*-Funktionen definiert werden, damit gesendete Daten der Worker auch empfangen werden können. Die Übergabeparameter sind vergleichbar zur Sendefunktion. Die einzigen Unterschiede sind, dass an der Stelle der Empfängerdefinition hier der Sender der Nachricht angegeben werden muss und zusätzlich noch eine Statusvariable übergeben wird. Um jede gesendete Nachricht zu empfangen wird eine *for*-Schleife implementiert, in der die Ränge der Instanzen durchlaufen werden. Der Master addiert die erhaltenen Daten und schließt die Berechnung zur Approximation von PI ab.

# Laufzeitanalyse

Innerhalb dieses Projekts wurde eine Laufzeitanalyse durchgeführt, um mögliche Vor- und/oder Nachteile der Parallelisierung der PI-Approximation aufzeigen zu können.

Hier werden noch die beiden Varianten erklärt

## Schwankung unterschiedlicher Laufzeiten

Die Schwankungen der Laufzeitmessungen sind mit den unterschiedlichen Latenzzeiten zu begründen, welche durch die interne Kommunikation entsteht. Dazu kann es zu leicht unterschiedlichen Ergebnissen kommen. Deutlich wird dies in Abbildung 2. Die Einzelmessungen sind statistisch um die durchschnittlich errechnete Laufzeit verteilt. Deutliche Ausreißer sind nicht erkennbar, sodass die Einzelmessungen plausibel sind.

## Laufzeitverkürzung

Beide Parallelisierungs-Varianten zeigen eine Laufzeitverkürzung, sodass ein Mehrgewinn eindeutig erkennbar ist. In der Testkonfiguration, in welcher die Rechenkapazität des Masters (login-node) genutzt wird, nimmt die Laufzeit, wie in Abbilung … zu sehen, mit der Anzahl der Worker exponentiell ab. Während Netzwerk in der Berechnung mir nur einer Instanz 18,83s benötigt, beläuft sich die Rechendauer bei der Nutzung von 5 virtuellen Maschinen auf 4,512s. Das ist eine Zeiteinsparung von 76,04%.

\*\*\*\*\* kannst du nochmal ein zwei sätze zu dem fall ohne Master schreiben ich bin mir nicht ganz sicher woran das mit der längeren Laufzeit bei 2 instanzen liegt\*\*\*

## Vergleich Approximation mit bzw. ohne Master

Die zentralisierte Kommunikation über den Master bring Vorteile in der Laufzeit mit, wenn die Anwendung auf bis zu drei Intanzen verteilt wird. Ab der Nutzung von mehr als drei Instanzen ist eine signifikante Verkürzung der Laufzeit nicht mehr zu erkennen. Grund dafür ist die MPI-Kommunikation, welche ab diesem Zeitpunkt prozentual gesehen einen größeren Einfluss auf die Laufzeit hat, als die eigentliche PI-Approximation.

# Fazit

ABCDXYZ (Wer hat was geschrieben?)

Auffälligkeiten (jeder eine Seite Reflexion) Länge Abhängig von Verzeichnis

Bis Sonntag 18/19 Uhr für Feedback

# Ausblick

ABCDXYZ

##### References

S. Hoffmann, R. L. (2008). *OpenMP: Eine Einführung in die parallele Programmierung mit C/C++.* (P. D. Prof. Dr. O. Günther, Hrsg.) Heidelberg: Springer-Verlag Berlin Heidelberg. doi:10.1007/978-3-540-73123-8

Shekhar S., X. H. (2017). *Message Passing Interface.* Cham: Springer International Publishing AG. doi:10.1007/978-3-319-17885-1\_100765

1. Kardinal Michael Faulhaber, “Monte-Carlo-Methode,” Mathepedia, https://mathepedia.de/Monte-Carlo-Methode.html
2. Jörg Christmann, Mathefritz Verlag, “Bestimmung von PI mit Monte-Carlo Methode,”., <https://www.mathestunde.com/kreiszahl-pi-bestimmen-monte-carlo>