# Dokumentation zu den Projektaufgaben 1 und 2

**Message Passing Programmierung**

Matthias Zober und Michael Horn

20. Januar 2017

## Inhaltsverzeichnis

# 1 Einleitung

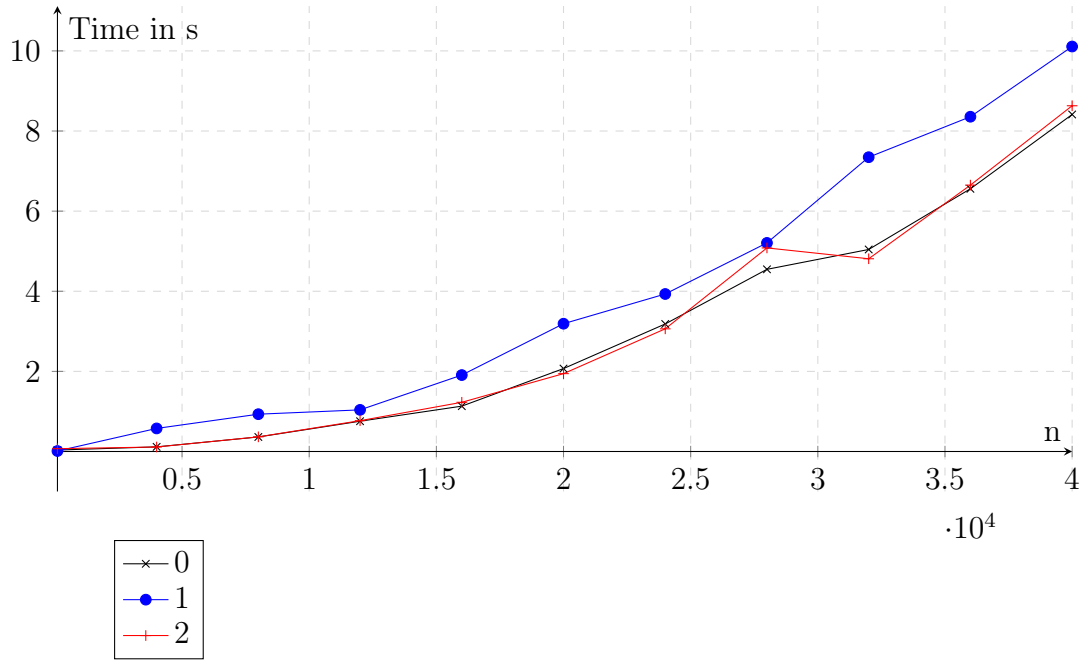
In dieser Dokumentation werden die Aufgaben „Rechteckmustererkennung“ sowie „Numerische Integration mittels Parabelformel“ betrachtet und hinsichtlich ihrer Parallelität untersucht. Der entstandene Quellcode kann unter: https://github.com/MZober1993/MessagePassingProjects[1] eingesehen werden.

# 2 Projektaufgabe 1: Rechteckmustererkennung

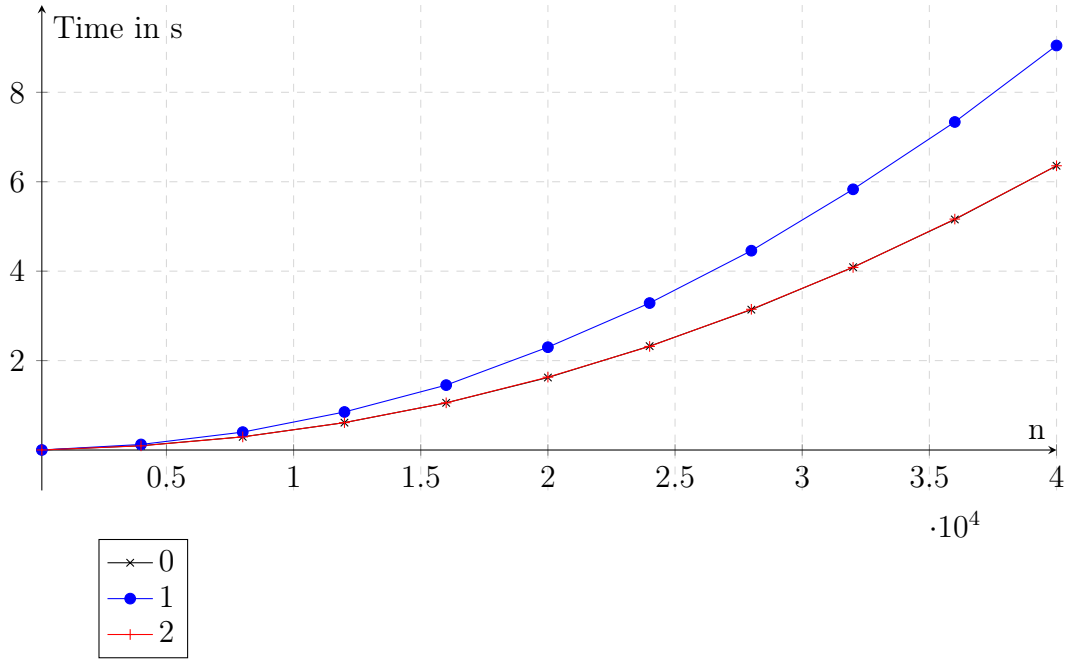
## 2.1 Realisierung

## 2.2 Laufzeitverhalten

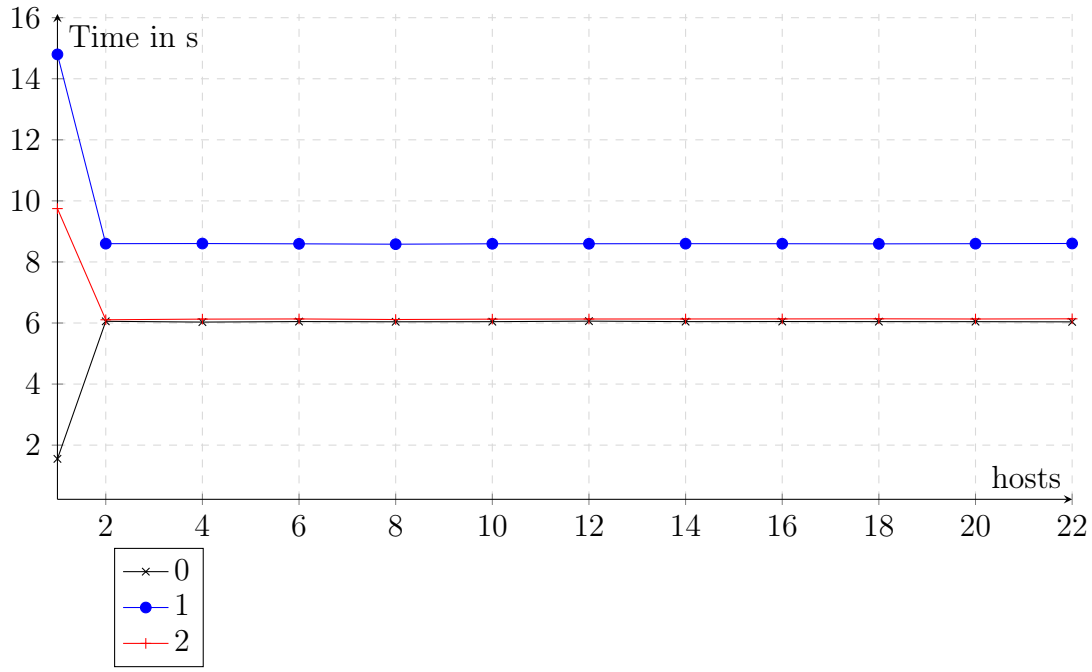
### 2.2.1 Festes n

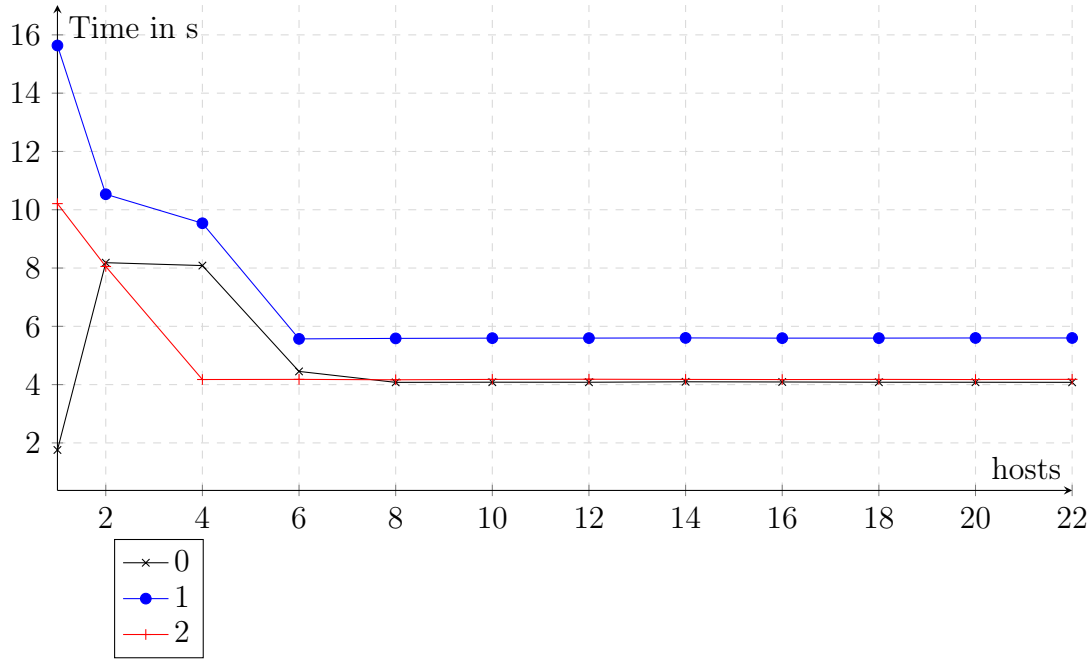



[1] Letzter Aufruf: 20. Januar 2017


Time in s
8
6
4
2
n
0.5
1
1.5
2
2.5
3
3.5
4
$\cdot 10^4$
0
1
2

### 2.2.2 Festes p


16
14
12
10
8
6
4
2
Time in s
hosts
2
4
6
8
10
12
14
16
18
20
22
0
1
2




## 2.3 Fazit

# 3 Projektaufgabe 2: Numerische Integration mittels Parabelformel

## 3.1 Verwendung

Der Quellcode kann mit Hilfe folgendes Aufrufes auf dem MC-3-System kompiliert werden:

- f77.px -o A2star.px A2star.f

Mit folgendem Befehl wird das Programm ausgeführt:

- run -f0 4 2 A2star.px **funktion n**

Für das Programm können Kommandozeilenargumente zur Auswahl der Testfunktion und die Größe der zu berechnenden Teilstücke des Integrals angegeben werden.

**funktion:** Das Programmargument **funktion** entscheidet darüber, welche der beiden gegebenen Testfunktionen verwendet wird. Als gültige Eingabe wird 1 für Funktion 1 und 2 für Funktion 2 erwartet.

**n:** Das zweite Programmargument **n** gibt die 2er Potenz der zu berechnenden Teilstücke des Integrals an. Je größer das **n**, umso genauer wird das berechnete PI. De höchstmöglichste zulässige Eingabe ist 20, das sind 1048576 zu berechnende Teilstücke. Diese Vorgehensweise soll zum Einen sicher stellen, dass **n** stets durch 2 Teilbar ist und zum Anderen das Laufzeitmessen vereinfachen, da ein exponentielles Wachstum vorliegt. Sollte kein Programmargument angeben werden, so wird intern die Funktion 1 mit $n = 2^{15} = 32768$ verwendet.

Weiterhin wird eine ungerade Prozessorzahl ausgeschlossen, damit stets gewährleistet ist, dass die zu berechnende Anzahl an Teilstücke pro Prozessor eine ganzzahlige Zahl ist.

## 3.2 Realisierung

Grundlegend wurde sich bei der Realisierung für eine Stern-Topologie entschieden. Gewählt wurde diese Topologie, da bei der Parallelisierung dieser Aufgabe ein Master existert, welche alle Teilergebnisse empfängt und summiert. Weiterhin wurde diese Topologie bereits in einem Seminar implementiert. Wie die Topologie erstellt wird zeigt Listing 1.

LISTING 1: Erzeugen der Stern-Topologie

```
if(id.eq.0) then
        do i=1,(np-1)
                links(i)=addnewlink(topid,i,1)
        enddo
else
        link=addnewlink(topid,0,1)
endif
```

Da beim Starten des Programms alle Prozessoren **n** sowie die Testfunktion bereits kennen, aufgrund der Realisierung als Kommandozeilenparameter, entfällt der Kommunikationsoverhead für das Verteilen von **n** sowie das Übermitteln der Information, welche Funktion verwendet wird.

Die Prozessoren können explizit ihre eigenen Bereich für die Teilstücke bestimmen und somit die numerische Berechnung durchführen. Nach Abschluss der Berechnung ihres Ergebnisses, werden diese an den Master-Prozessor gesendet. Der Master-Prozessor addiert die Ergebnisse der anderen Prozessoren zu seinem eigenem Ergebnis auf und wendet die Multiplikation h/3 auf das Gesamtergebnis an. Im Anschluss werden die Ergebnisse wie Referenzwert-PI, Berechnetes-PI, Abweichung und Laufzeit (nicht im Listng dargestellt) ausgegeben. Das folgende Listing 2 zeigt dies.

LISTING 2: Empfangen und Auswerten

```
c       starte die Integration
        call startIntegration(f,n,integral,h)
        if(id.eq.0) then
                print*,"use f:",f," with n:",n
```

```
c       Integralteilstück vom Master
                summe = integral
                do i=1,(np-1)
c       Empfange alle Teilstücke und addiere Sie
                        call recv(topid,links(i),integral,8)
                        summe = summe + integral
                enddo
                summe = h/3 *summe
                derivation =  PI() - summe
                print*,"-----------------------------"
                print*,"PI Referece:",PI()
                print*,"PI Integral:",summe
                print*,"PI Derivat.:",derivation
        else
c       Sende Integralteilstücke zum Master
                call send(topid, link,integral,8)
        endif
```

Die numerische Berechnung des Integrals wird durch jeden Prozessor ausgeführt, aber jeder Prozessor führt nur einen Teil der kompletten Berechnung aus. Die eigentliche Berechnung ist in dem Listing 3 zu sehen. Zunächst berechnet jeder Prozessor die Anzahl der durchzuführenden Berechnungen, als lokales **n** (**nL**). Im Anschluss werden Startpunkt (**aL**) und Endpunkt (**bL**) ermittelt. Die Funktionswerte von **aL** und **bL** der Testfunktion werden addiert und in einer Summe gespeichert. Im Anschluss werden die Zwischenstücke mit entsprechendem Faktor berechnet und ebenfalls summiert. Die Variable **h** gibt dabei die allgemeine Schrittweite an und **step** die derzeitigen Schrittwert.

Nach erfolgter Berechnung sendet jeder Prozessor sein Ergebnis an den Master (falls der Prozessor nicht selbst der Master ist)

LISTING 3: Berechnen der relevanten Bereiche für jeden Prozessor

```
        id = myprocid()
        nL = n / nprocs()
        aL = a + id * nL * h
        bL = aL + nL * h
c       Randwerte, getVal liefert Funktionswert der Testfunktion
        summe = getVal(f,aL) + getVal(f,bL)
```

```
        step = aL + h
c       Berechnung zwischenstücke, factor alterniert mit 2 und 4
        do i=1,nL-1
                exponent = mod(i,2) + 1
                factor=2**exponent
                summe = summe + dble(getVal(f,step) * factor)
                step = step + h
        enddo
```

## 3.3 Ergebnisse

### 3.3.1 Testfunktionen

Wie in den Tabellen zu sehen ist liefert die Testfunktion 2 schneller bessere PI-Werte als Funktion 1. Ein Grund dafür das Testfunktion 1 schlechtere Ergebnisse als Funktion 2 liefert, könnte damit zusammenhängen, dass für die Funktion 1 die Oberegrenze PI selbst ist.

### 3.3.2 Laufzeitverhalten

#### Festes n

#### Festes p

## 3.4 Fazit

Kommunikationsoverhead spielt hier eine untergeordnete Rolle, da Minimum an Kommunikation zur Berechnung notwendig ist. Weiterhin lohnt sich die Parallelisierung,..