Uni Hamburg
WR - Informatik
04.04.2012

# Simulation der MPI collective operations im PIOsimHD

Artur Thiessen
Informatik Bachelor
Matrikelnummer: 5944906
7thiesse@informatik.uni-hamburg.de

# Inhaltsverzeichnis

# 1. Einleitung

Viele Probleme aus unserem Alltag stellen uns vor mathematische Probleme, die nicht mit dem Heimrechner in annehmbarer Zeit zu meistern sind. Somit wird die Bedeutung von Hochleistungsrechnern immer groser und damit auch ihre Geschwindigkeit.
Eine Unterart der Hochleistungsrechner ist der Cluster. Dieser besteht aus vielen Rechnern, die ihre Arbeit dezentral bearbeiten. Dazu wird das zu losende Problem in Einzelteile zerlegt und auf die Rechner verteilt. Nach den Berechnungen werden die Daten wieder zusammengefugt. Die Kommunikation zwischen den Rechner wird haufig uber MPI (Message Passing Interface) abgewickelt, speziell von den kollektiven Operationen. Diese sind allein dafur zustandig Daten zu verteilen und zusammenzutragen. Es ist von grosem Interesse die Laufzeit von solchen Operationen so gering wie moglich zu halten, um die beste Performance zu erreichen.

# 2. Aufgabenstellung

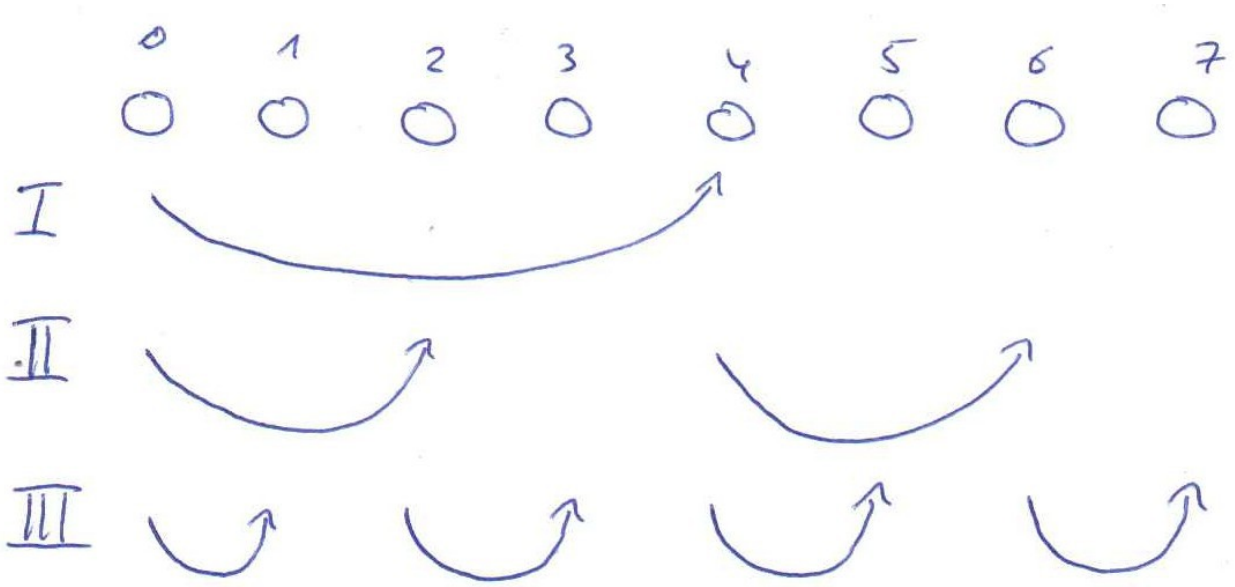
Das Ziel dieses Projektes ist es den bestehenden PIOsimHDSimulator um die kollektiven Funktionen aus dem MPI Standart(Broadcast, Gather, Reduce, Scatter, Allgather, Allreduce) zu erweitern und diese so zu implementieren, dass die Funktionsweise der von MPICH2 gleicht. Daruber hinaus sollen die Durchlaufzeiten der Einzelnen Funktionen auf dem Simulator mit denen des realen Clusters verglichen werden.

Hauptziel dieser Arbeit ist es den Grundstein dafur zu legen, dass die bestehenden MPI Funktionen bequem getestet und optimiert werden konnen, ohne einen realen Cluster dafur zu brauchen.

## 3.1.1 Bcast:

MPI_BCAST( buffer, count, datatype, root, comm )
INOUT buffer starting address of buffer (choice)
IN count number of entries in buffer (non-negative integer)
IN datatype data type of buffer (handle)
IN root rank of broadcast root (integer)
IN comm communicator (handle)

## 3.1.2 Kommunikationsmuster:




(Binary tree algorithm)

## 3.1.3 Bemerkung

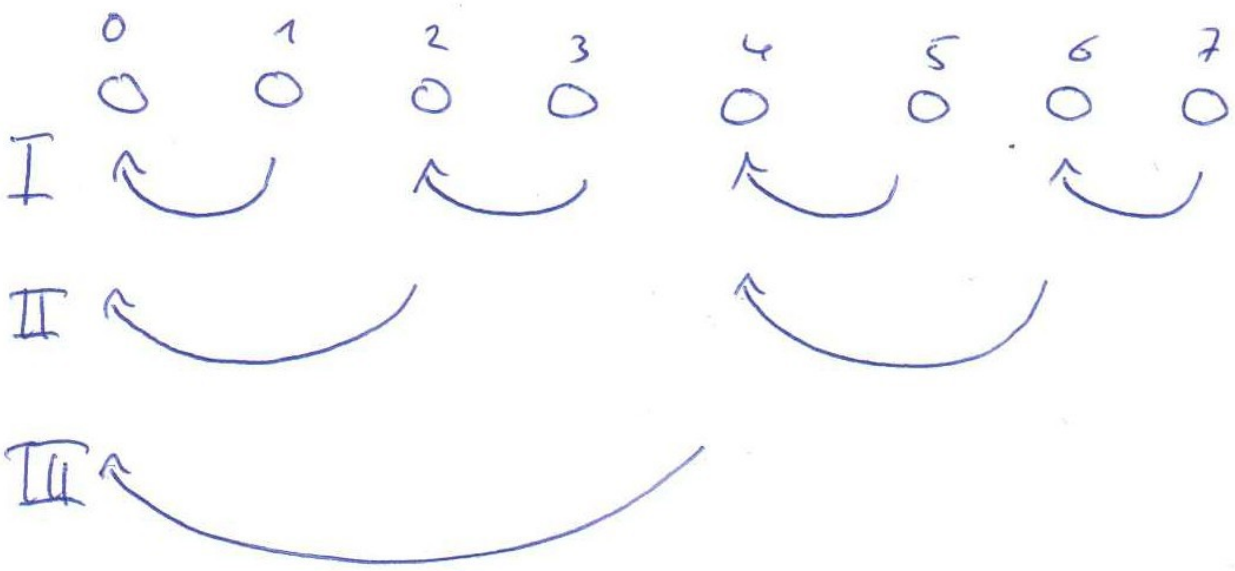
Beim MPI_BCast werden, im Gegensatz zu MPI_Scatter, die gesammten Daten weitergeschickt. Ziel ist es nämlich zu erreichen, dass jeder Knoten am Ende der Operation über die selben Daten verfügt.

Der MPI_BCast Algorithmus läuft im besten Fall in einer Zeit von log(n).

## 3.2.1 Gather:

MPI_GATHER( sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)
IN sendbuf starting address of send buffer (choice)
IN sendcount number of elements in send buffer (non-negative integer)
IN sendtype data type of send buffer elements (handle)
OUT recvbuf address of receive buffer (choice, significant only at root)
IN recvcount number of elements for any single receive (non-negative integer, significant only at
root)
IN recvtype data type of recv buffer elements (significant only at root) (handle)
IN root rank of receiving process (integer)
IN comm communicator (handle)

## 3.2.2 Kommunikationsmuster:




(Reversed binary tree algorithm)

## 3.2.3 Bemerkung

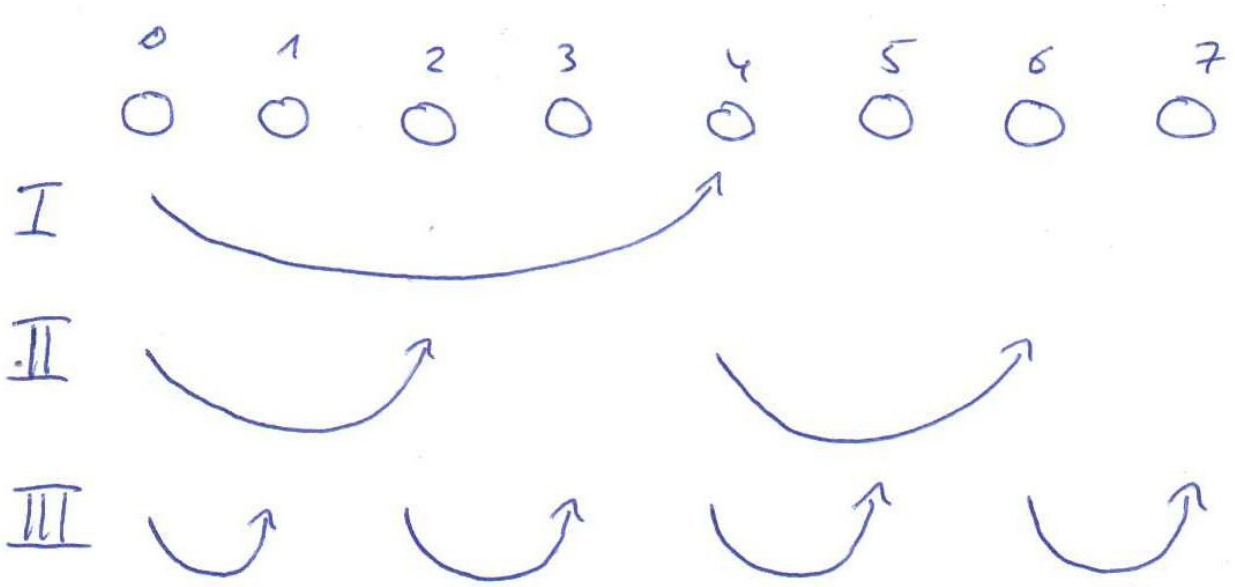
Die Ausführung von MPI_Gather führt dazu, dass der sog. "root rank" (hier die 0) am Ende der Operation über alle Daten aller Knoten verfügt. Diese werden zu einem Gesammtpaket akkumuliert.

Der MPI_Gather Algorithmus läuft im besten Fall in einer Zeit von log(n).

## 3.3.1 Scatter:

MPI_SCATTER( sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)
IN sendbuf address of send buffer (choice, significant only at root)
IN sendcount number of elements sent to each process (non-negative integer, significant only at
root)
IN sendtype data type of send buffer elements (significant only at root) (handle)
OUT recvbuf address of receive buffer (choice)
IN recvcount number of elements in receive buffer (non-negative integer)
IN recvtype data type of receive buffer elements (handle)
IN root rank of sending process (integer)
IN comm communicator (handle)

## 3.3.2 Kommunikationsmuster:




(Binary tree algorithm)

## 3.3.3 Bemerkung

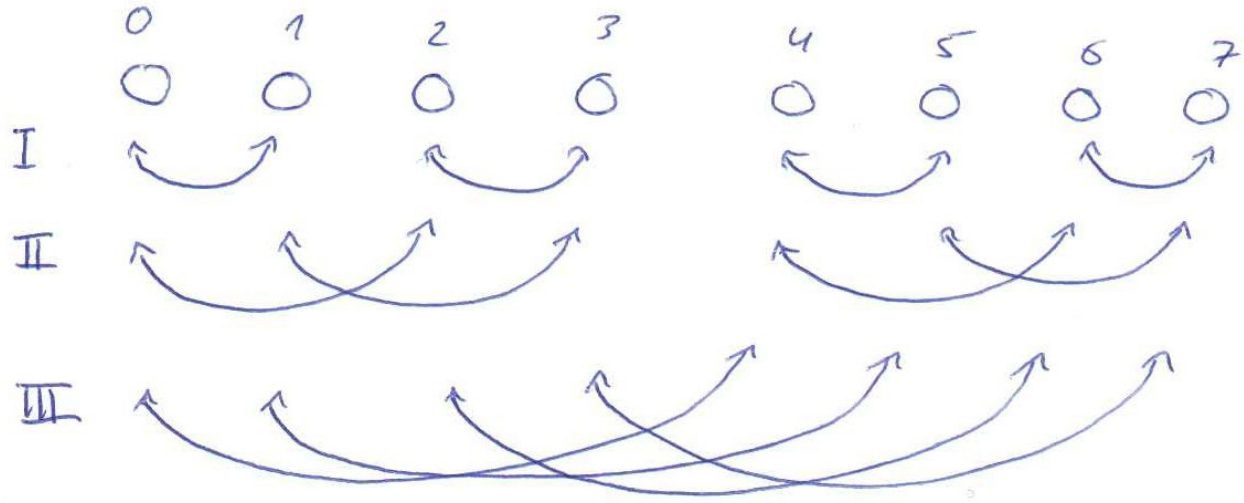
MPI_Scatter verteilt die dem "root rank" zur Verfügung stehenden Daten so, dass jeder Knoten nach möglichkeit die gleiche Menge Daten erhält. Wichtig hierbei ist, dass die Daten, die jeder einzelne Knoten bekommt, unterschiedlich sind.

Der MPI_Scatter Algorithmus läuft im besten Fall in einer Zeit von log(n).

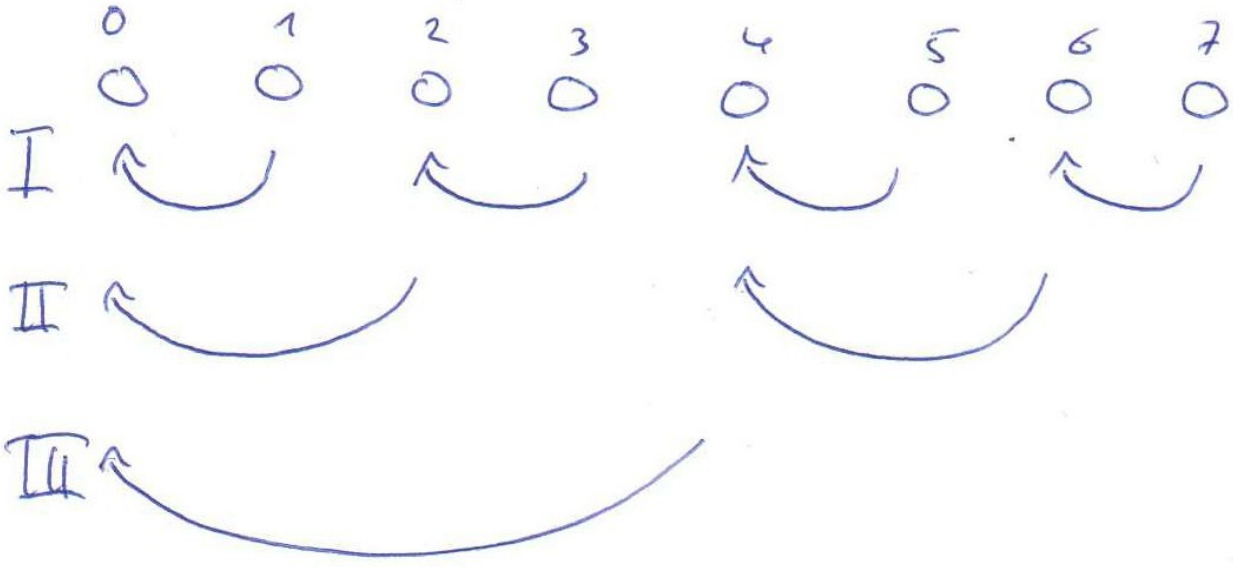
## 3.4.1 Reduce:

MPI_REDUCE( sendbuf, recvbuf, count, datatype, op, root, comm)
IN sendbuf address of send buffer (choice)
OUT recvbuf address of receive buffer (choice, significant only at root)
IN count number of elements in send buffer (non-negative integer)
IN datatype data type of elements of send buffer (handle)
IN op reduce operation (handle)
IN root rank of root process (integer)
IN comm communicator (handle)

## 3.4.2 Kommunikationsmuster (ReduceScatter → Gather):




(ReduceScatter – recursive halfing algorithm)




(Gather – reversed binary tree algorithm)

### 3.4.3 Bemerkung

MPI_Reduce sorgt dafür, dass eine Reduce-Operation von allen Knoten ausgeführt wird. Zuvor werden die zu verrechnenden Daten verstreut und anschließend wieder beim "root rank" zusammen geführt.

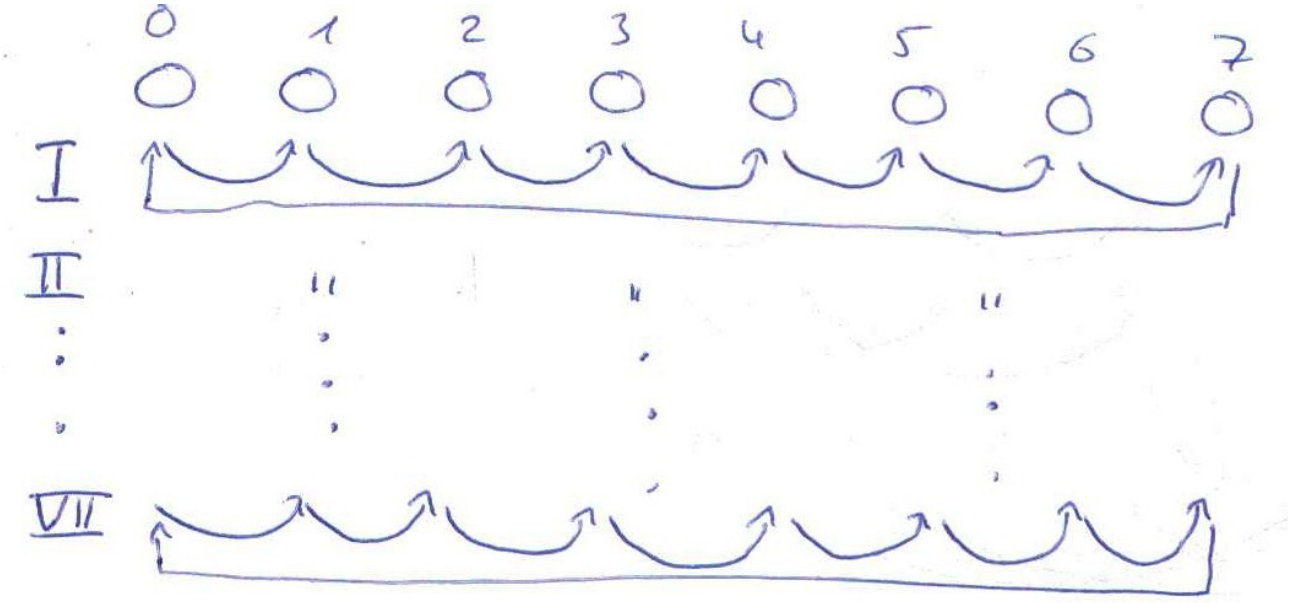
Als Beispiel kann hier eine Summenbildung betrachtet werden, wobei die einzelnen Summanden auf verschiedene Knoten übertragen werden, dort verrechnet und am Ende wieder zum Hauptknoten zusammenfließen. Diese Operation kann eine enorme Zeitersparnis bei der Berechnung von Kommutativen aufgaben mit sich bringen.

Der MPI_Reduce Algorithmus läuft im besten Fall in einer Zeit von $\log(n)^2$.

## 3.5.1 AllGather:

MPI_ALLGATHER( sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)
IN sendbuf starting address of send buffer (choice)
IN sendcount number of elements in send buffer (non-negative integer)
IN sendtype data type of send buffer elements (handle)
OUT recvbuf address of receive buffer (choice)
IN recvcount number of elements received from any process (non-negative integer)
IN recvtype data type of receive buffer elements (handle)
IN comm communicator (handle)

## 3.5.2 Kommunikationsmuster:




(Ring algorithm)

## 3.5.3 Bemerkung

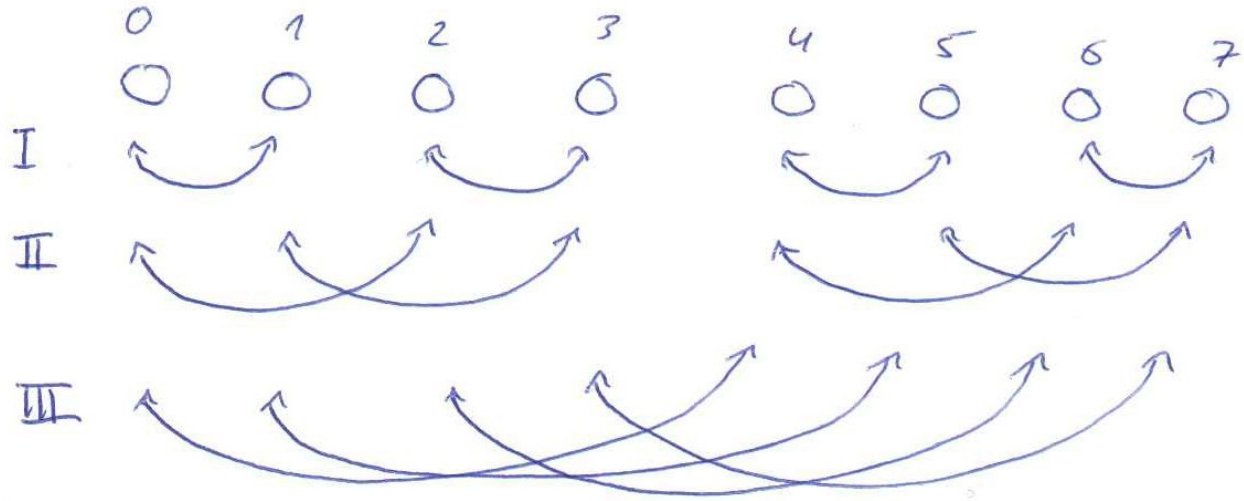
MPI_Allgather funktioniert analog zu MPI_Gather, mit der Ausnahme, dass die gesammelten Daten an alle Knoten übermittelt werden. Hierbei wird ein einfacher Ringalgorithmus benutzt.

Der MPI_Allgather Algorithmus läuft im besten Fall in einer Zeit von n-1.

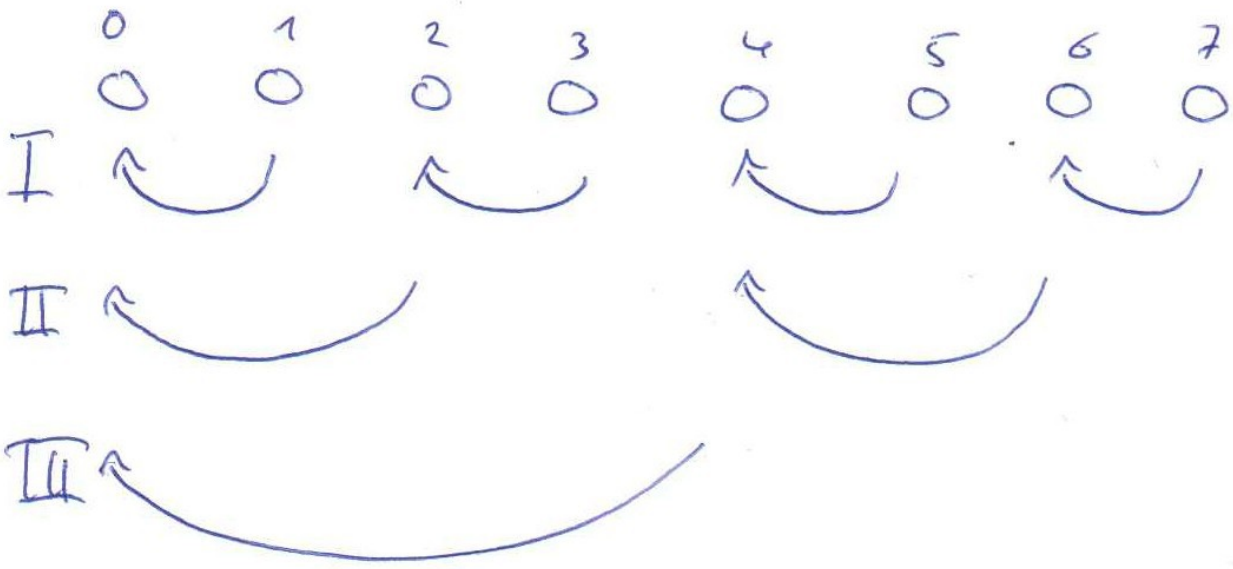
### 3.6.1 AllReduce:

MPI_ALLREDUCE( sendbuf, recvbuf, count, datatype, op, comm)
IN sendbuf starting address of send buffer (choice)
OUT recvbuf starting address of receive buffer (choice)
IN count number of elements in send buffer (non-negative integer)
IN datatype data type of elements of send buffer (handle)
IN op operation (handle)
IN comm communicator (handle)

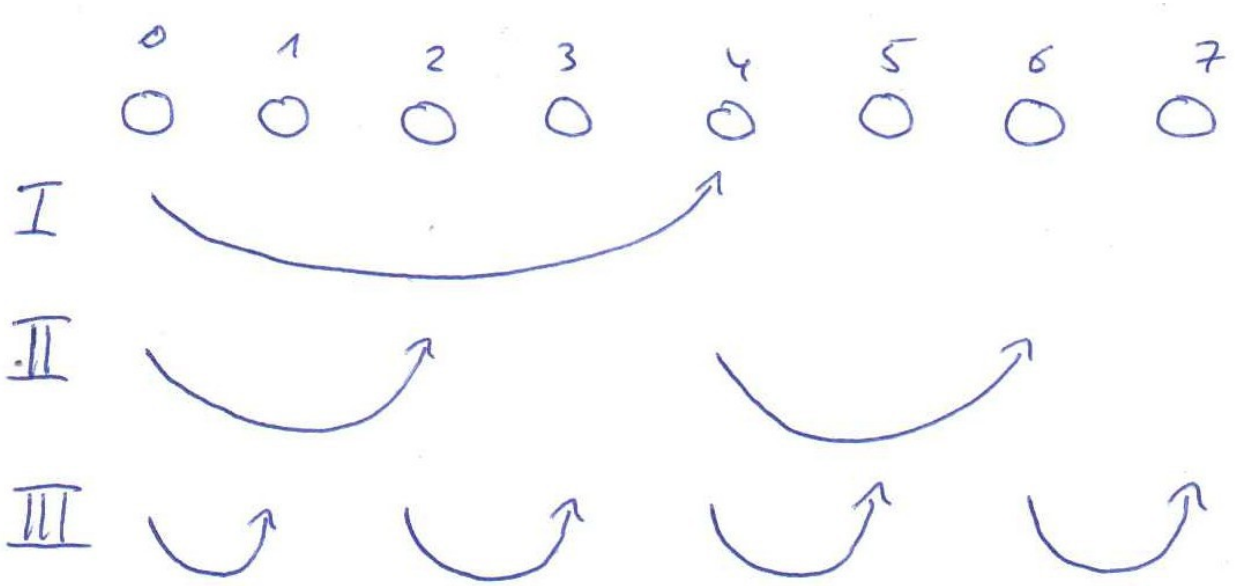
### 3.6.2 Kommunikationsmuster (ReduceScatter → Gather → Bcast):




(ReduceScatter)



(Gather)

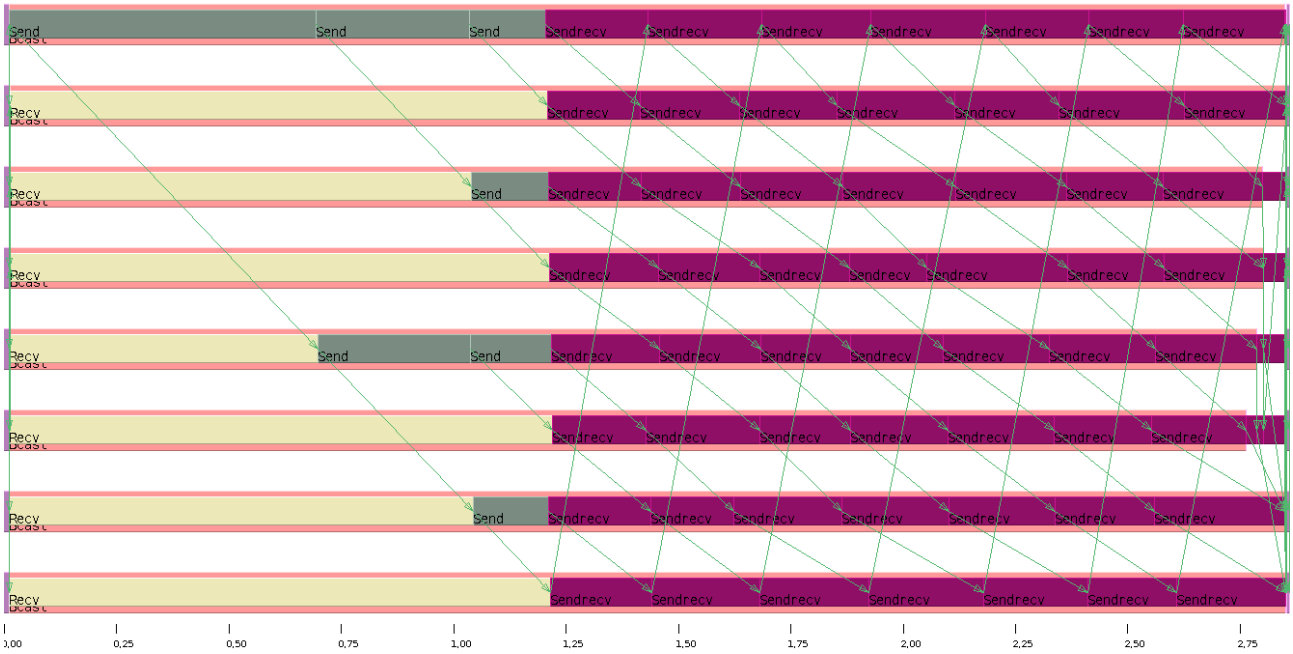



(Bcast)

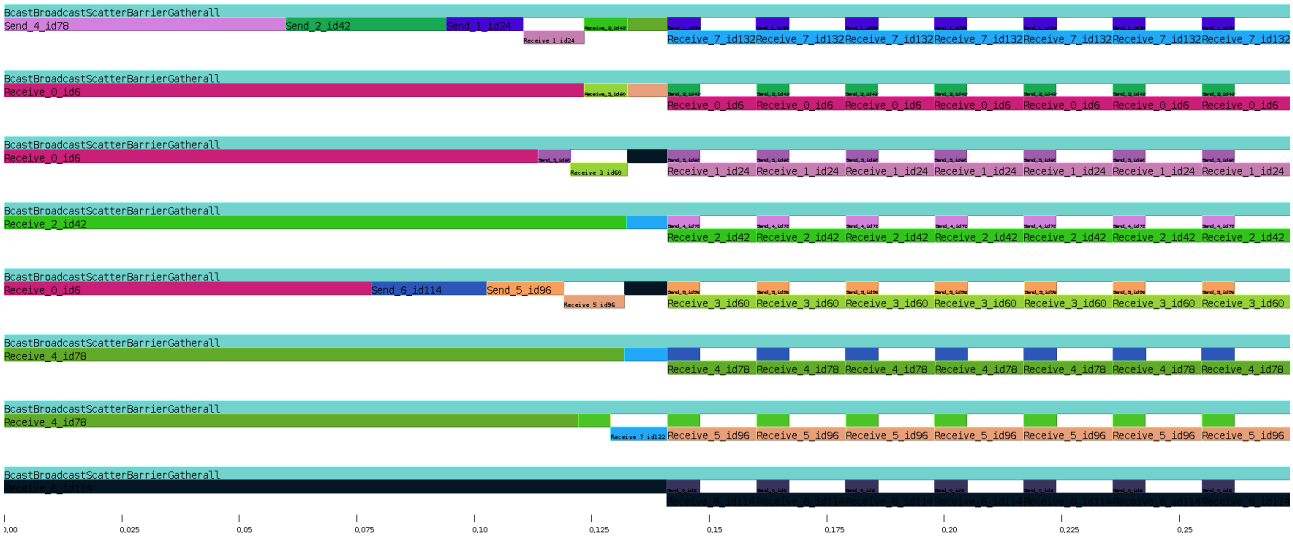
### 3.6.3 Bemerkung

MPI_Allreduce funktioniert analog zu MPI_Reduce, mit der Ausnahme, dass am Ende der Reduce Operation das Ergebnis per MPI_Broadcast verteilt wird.

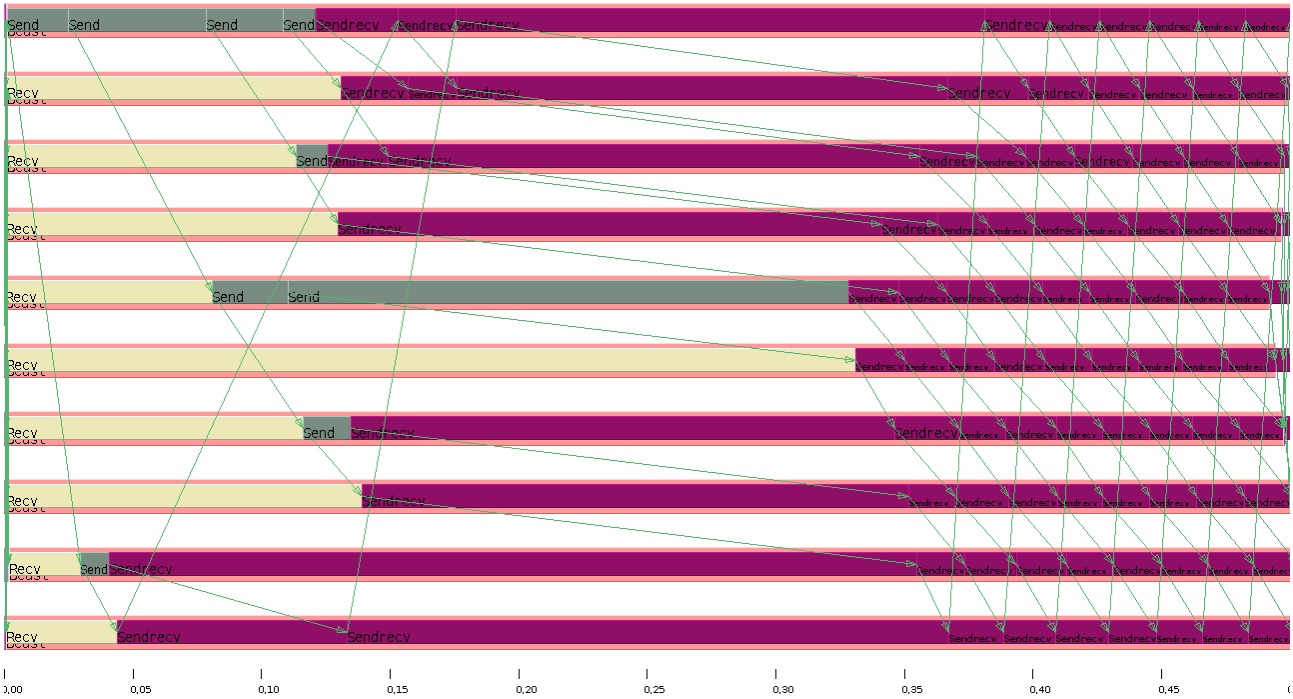
## 4.1.1 Cluster/Simulator 8/10 Screenshots



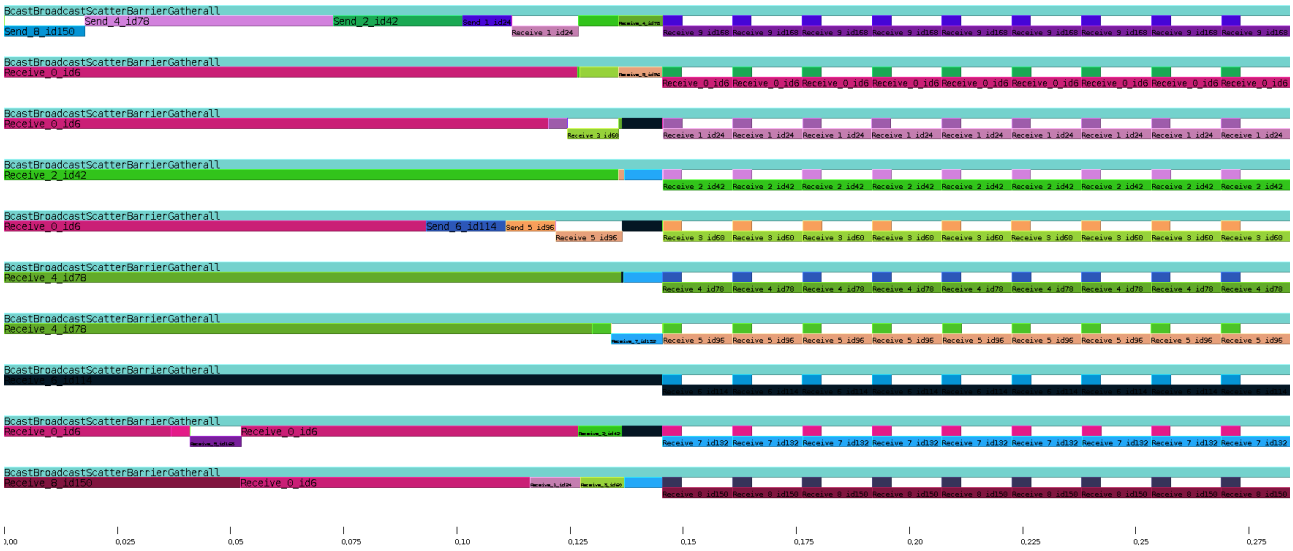

(Bcast 8-8 Cluster)




(Bcast 8-8 Simulator)




(Bcast 10-10 Cluster)

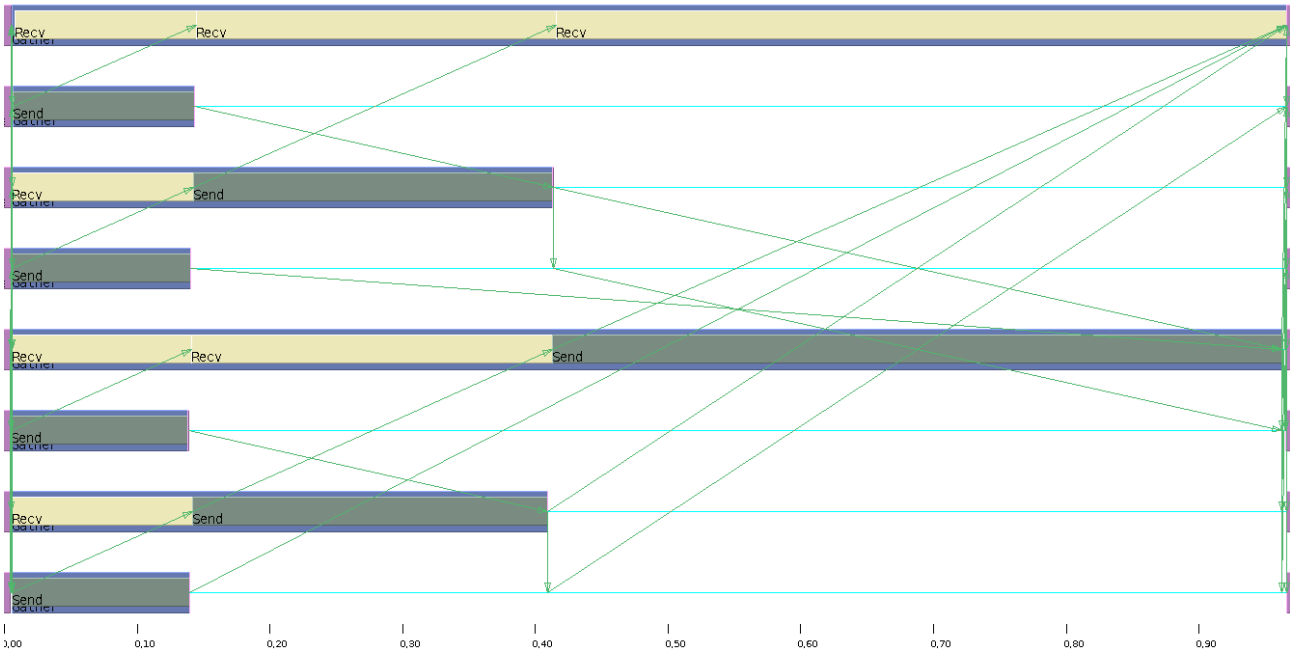



(Bcast 10-10 Simulator)

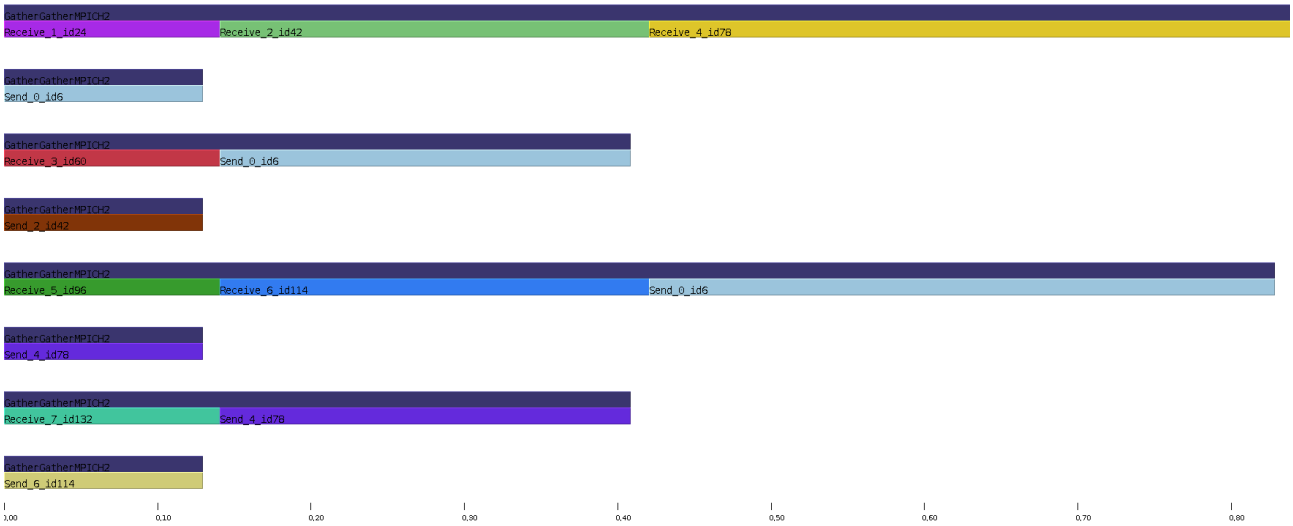
## 4.1.2 Beobachtungen:

Offensichtlich ist die Implementierung von Broadcast korrekt. Die Kommunikationsmuster stimmen mit dem Original uberein. Was jedoch auffallt ist der Laufzeitunterschied. Die Simulation lauft knapp 50% schneller durch als MPI_Bcast auf dem Cluster.

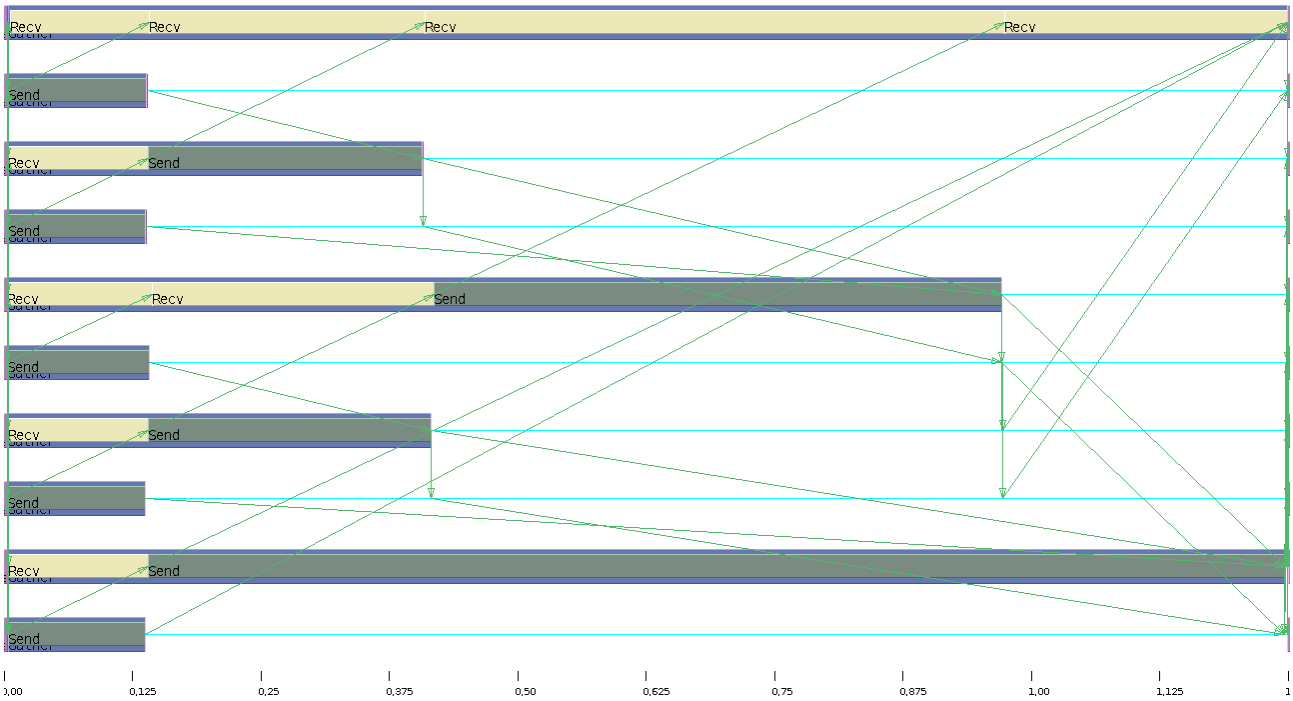
## 4.2.1 Cluster/Simulator 8/10 Screenshots



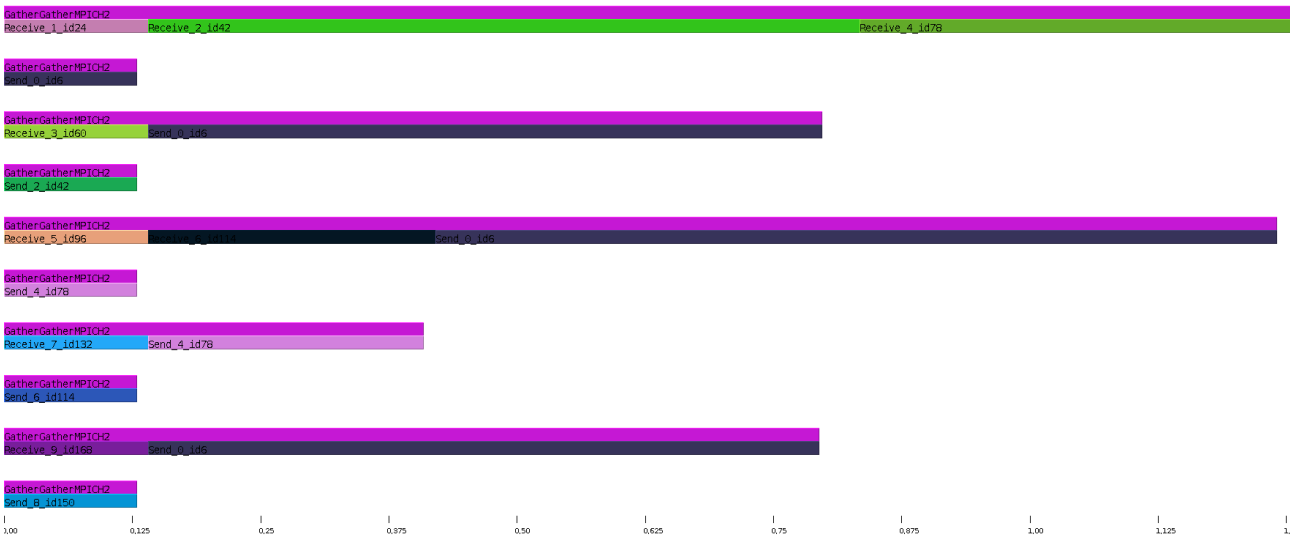

(Gather 8-8 Cluster)




(Gather 8-8 Simulator)




(Gather 10-10 Cluster)

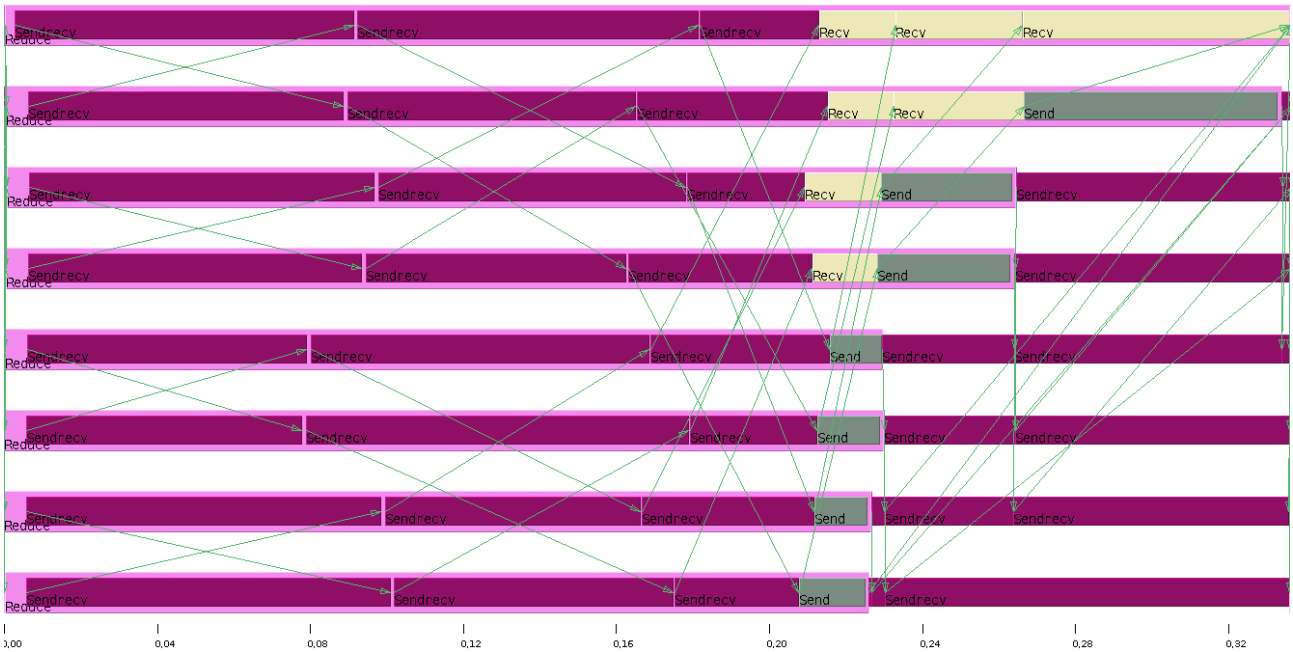



(Gather 10-10 Simulator)

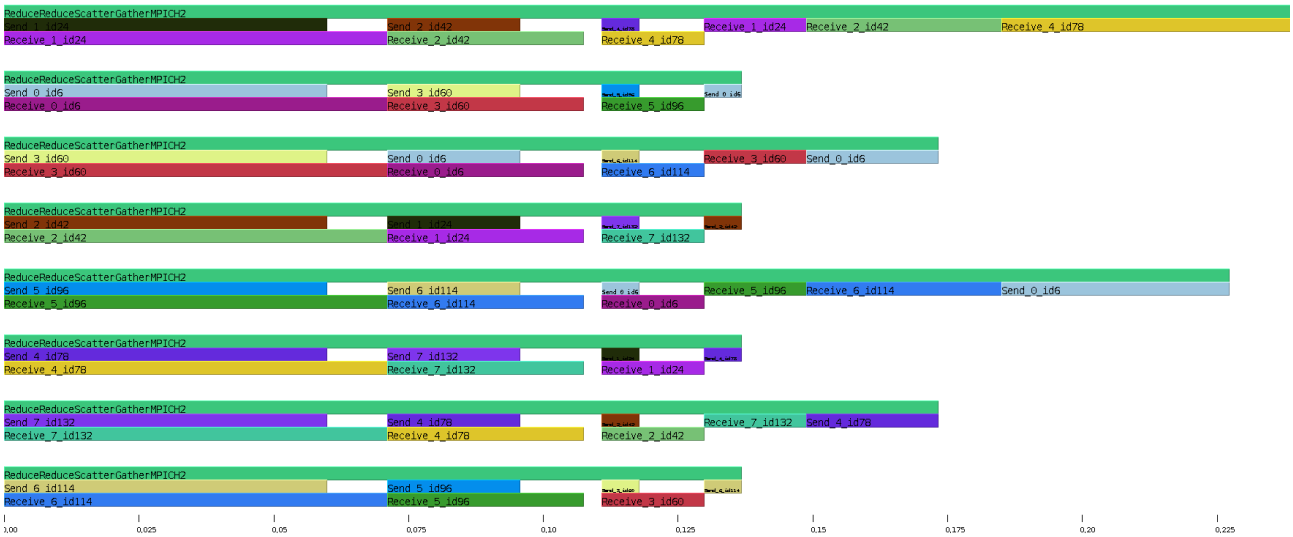
## 4.2.2 Beobachtungen:

Die Laufzeit im Vergleich sieht sehr gut aus. Die Abweichung von ca. 10% ist zu vernachlassigen. Die Implementierung scheint jedoch kleine Macken aufzuweisen: Die Prozessreihenfolge ist vertauscht.

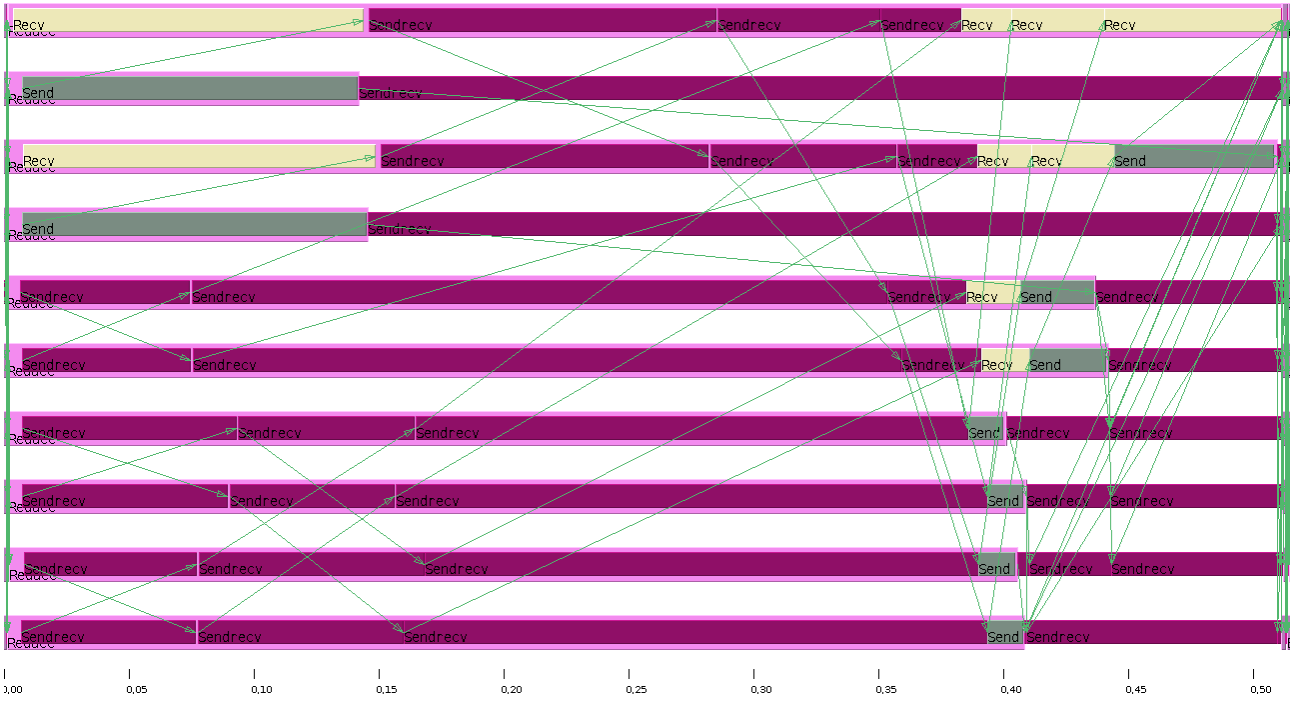
## 4.3.1 Cluster/Simulator 8/10 Screenshots



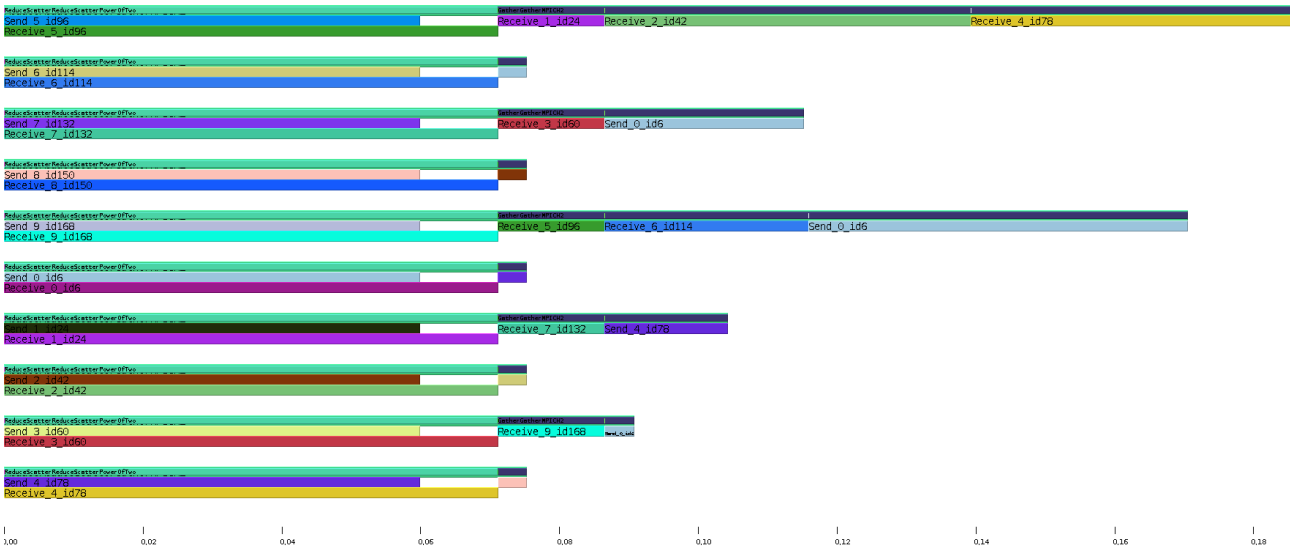

(Reduce 8-8 Cluster)




(Reduce 8-8 Simulator)




(Reduce 10-10 Cluster)



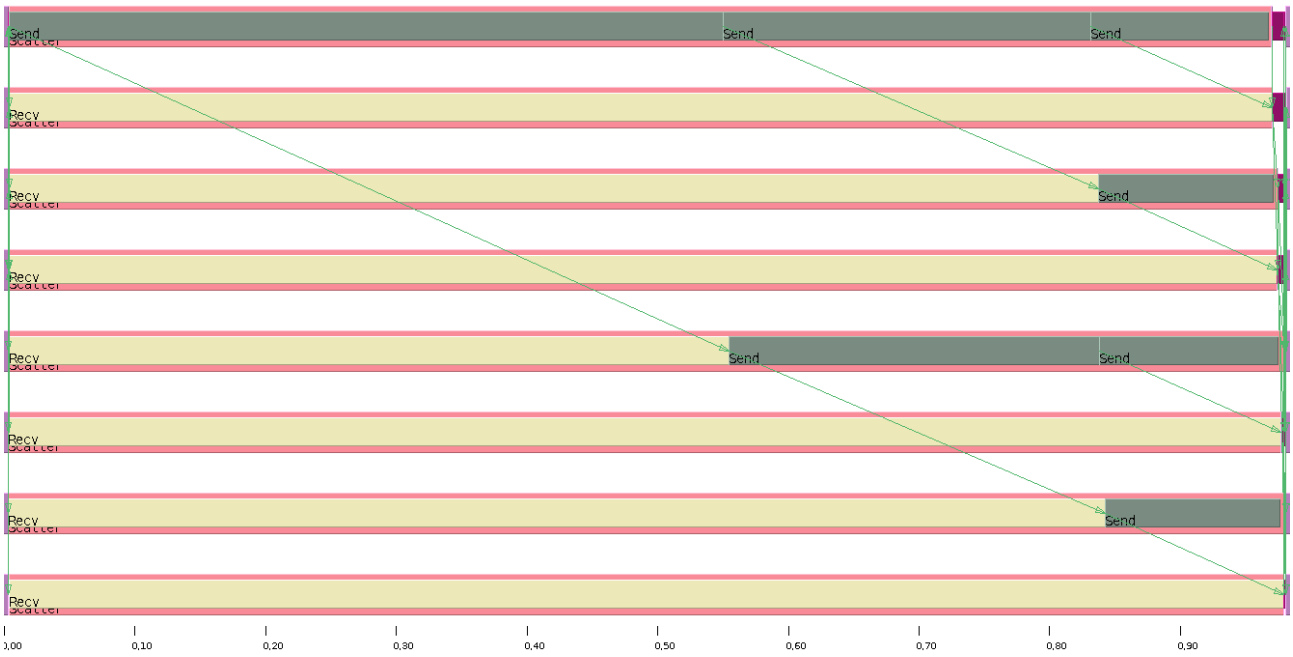

(Reduce 10-10 Simulator)

## 4.3.2 Beobachtungen:

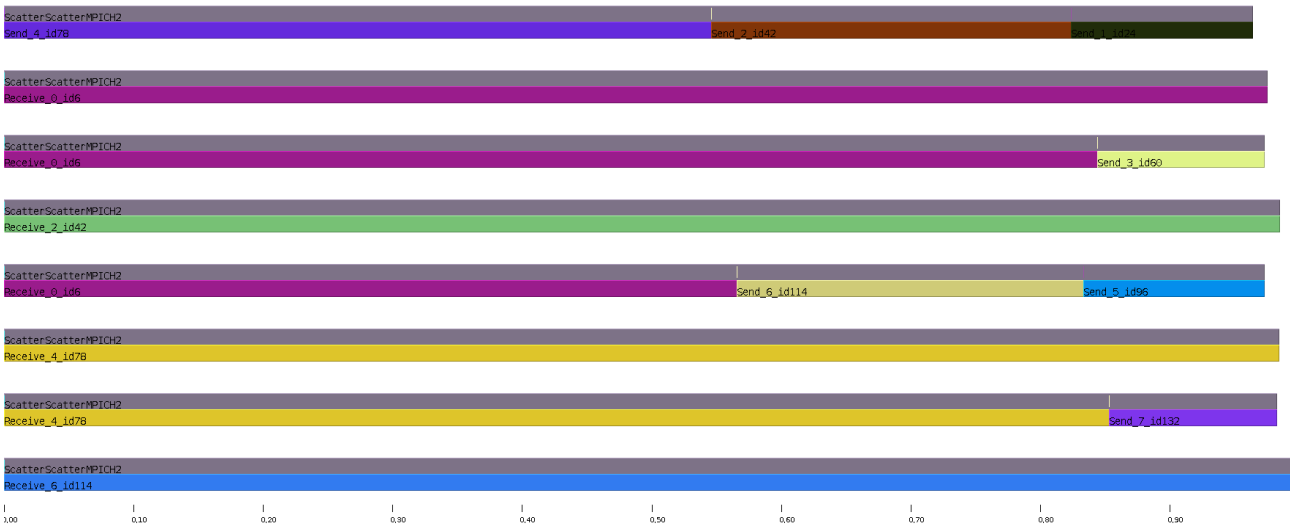
Hier scheint es Probleme zu geben. Zum einen scheint die Gather Methode auf dem Cluster anders abzulaufen, als auf dem Simulator. Zum anderen unterscheiden sich die Laufzeiten bei „non-power-oftwo“

Fallen enorm (mehr als 50%)

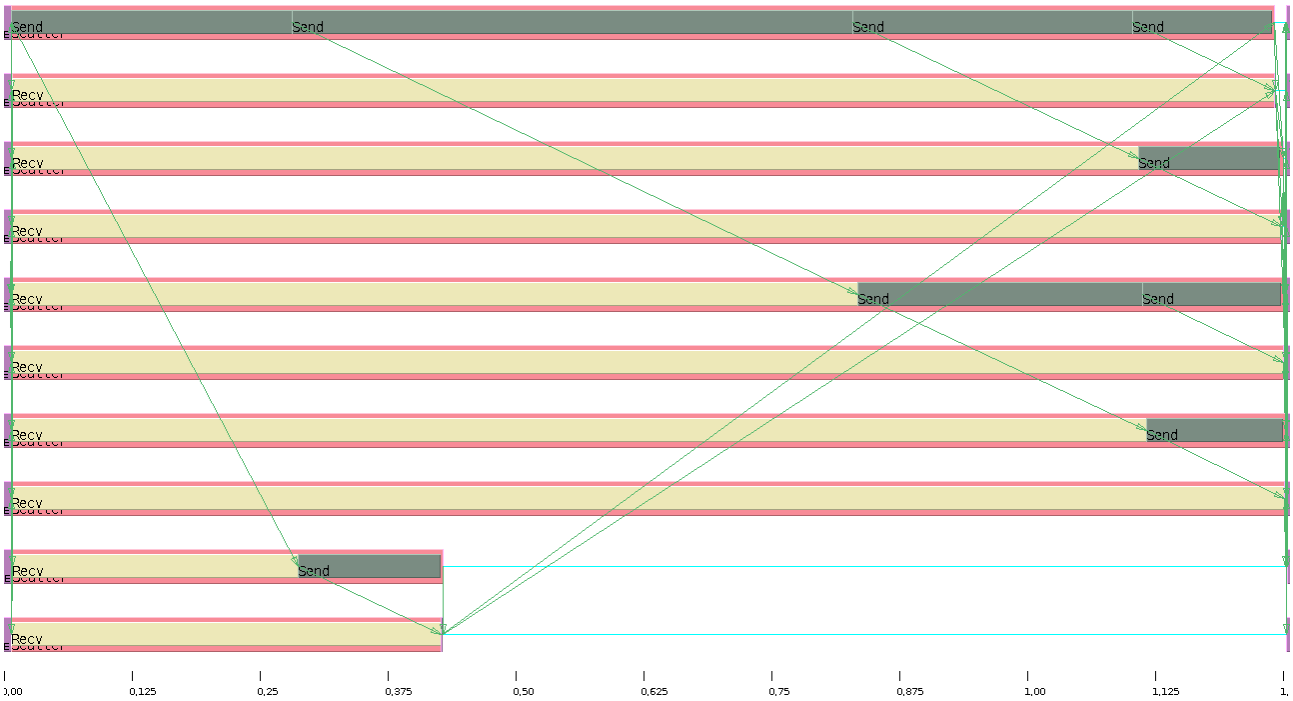
### 4.4.1 Cluster/Simulator 8/10 Screenshots



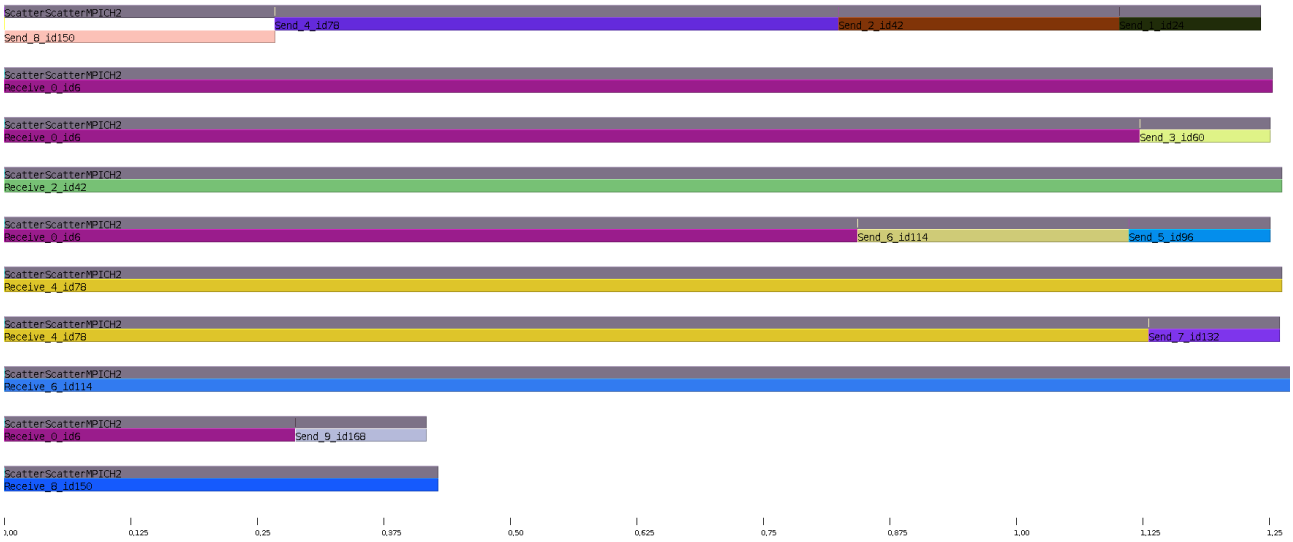

(Scatter 8-8 Cluster)




(Scatter 8-8 Simulator)




(Scatter 10-10 Cluster)

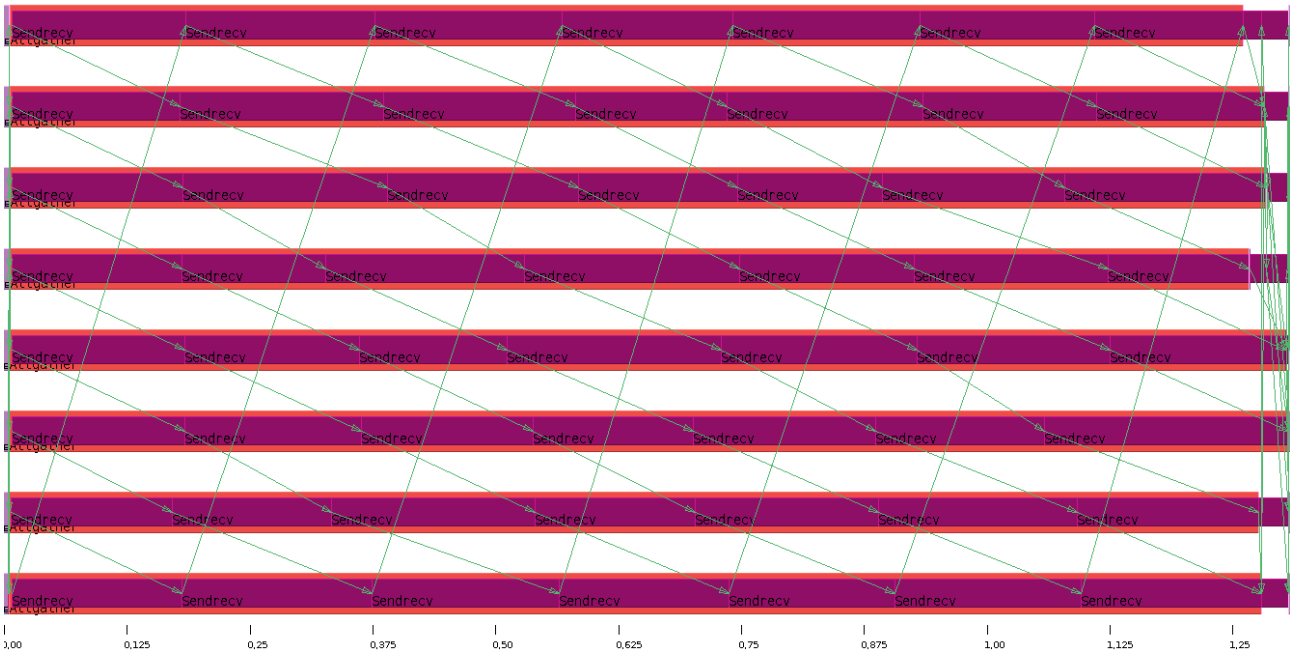



(Scatter 10-10 Simulator)

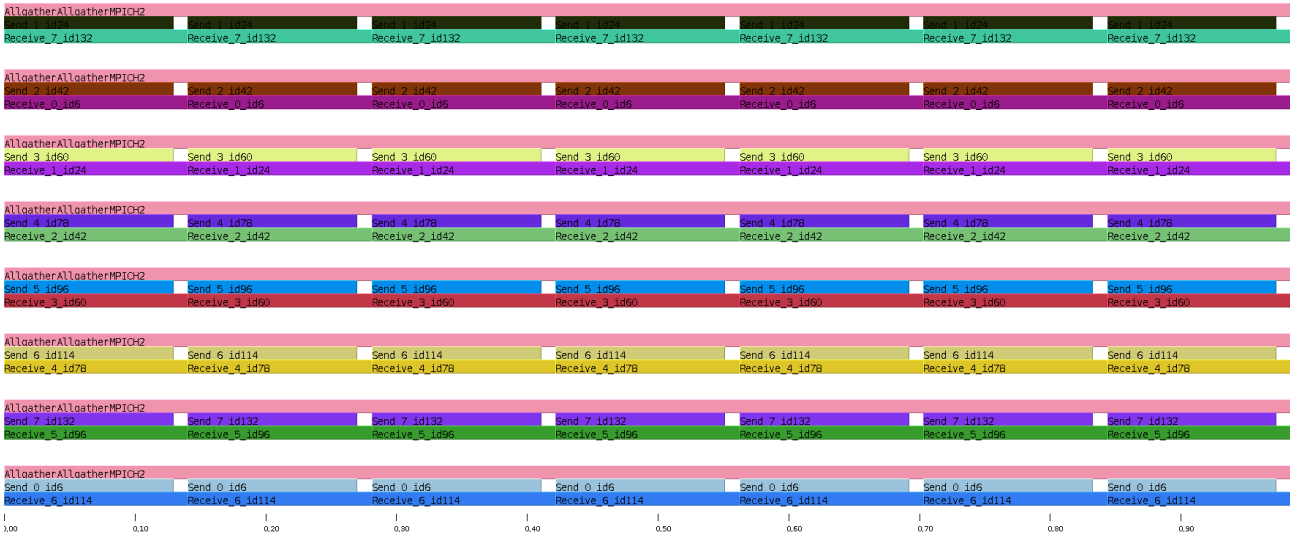
## 4.4.2 Beobachtungen:

Sowohl Laufzeiten als auch die Muster sehen sehr gut aus. Der Simulator ist also in der Lage MPI_Scatter sehr genau zu simulieren.

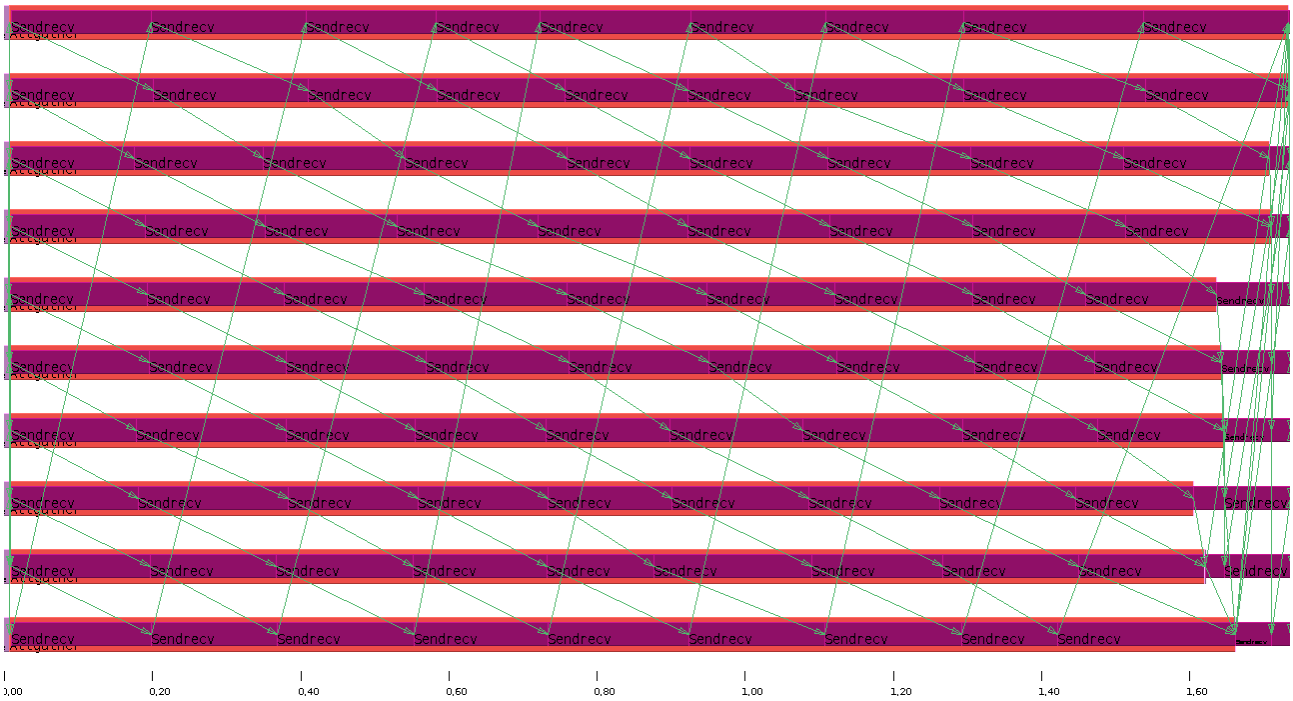
## 4.5.1 Cluster/Simulator 8/10 Screenshots



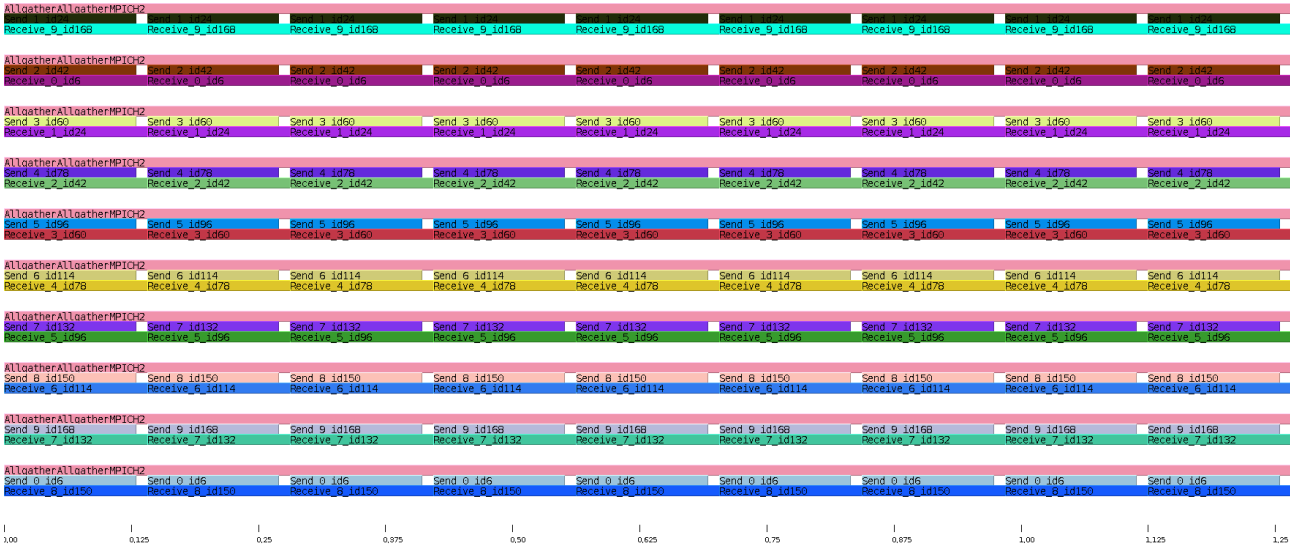

(Allgather 8-8 Cluster)




(Allgather 8-8 Simulator)




(Allgather 10-10 Cluster)

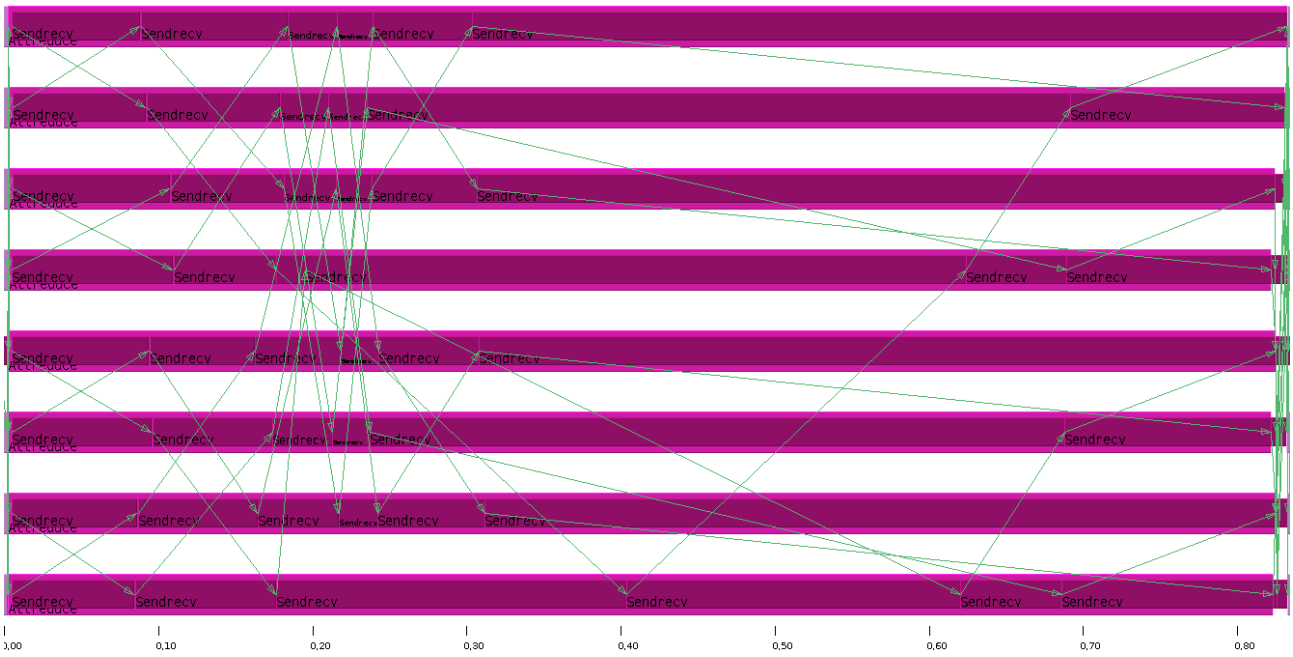



(Allgather 10-10 Simulator)

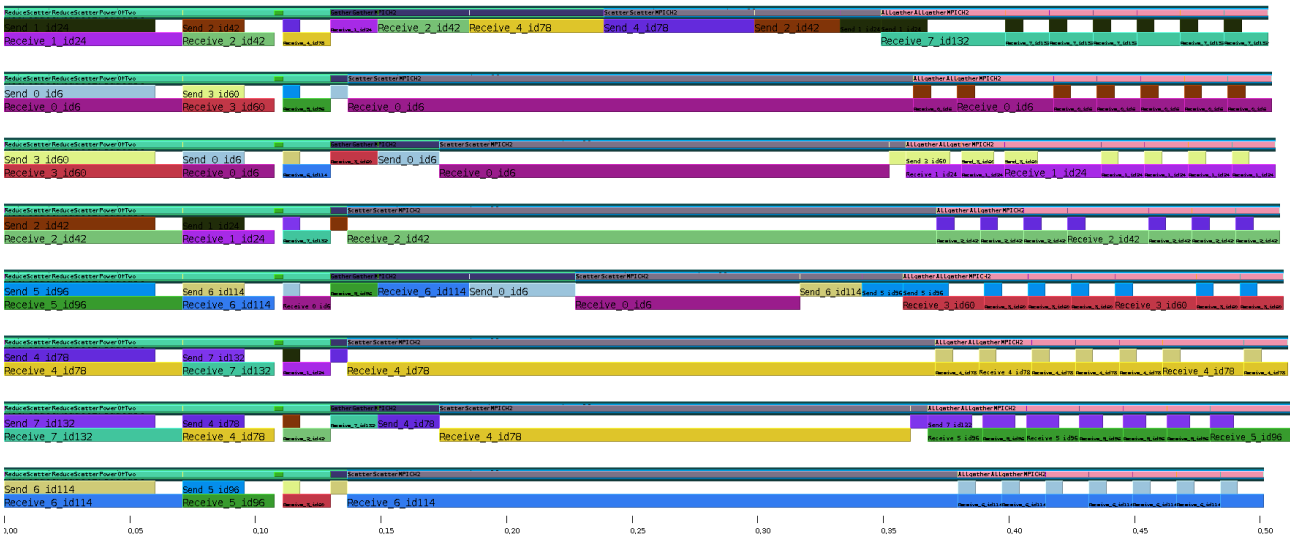
## 4.5.2 Beobachtungen:

Die Implementierung ist korrekt. Die Laufzeiten weichen jedoch stark ab. Eventuell ist dieses Problem mittels Anpassung der Hardwareparameter zu beheben.

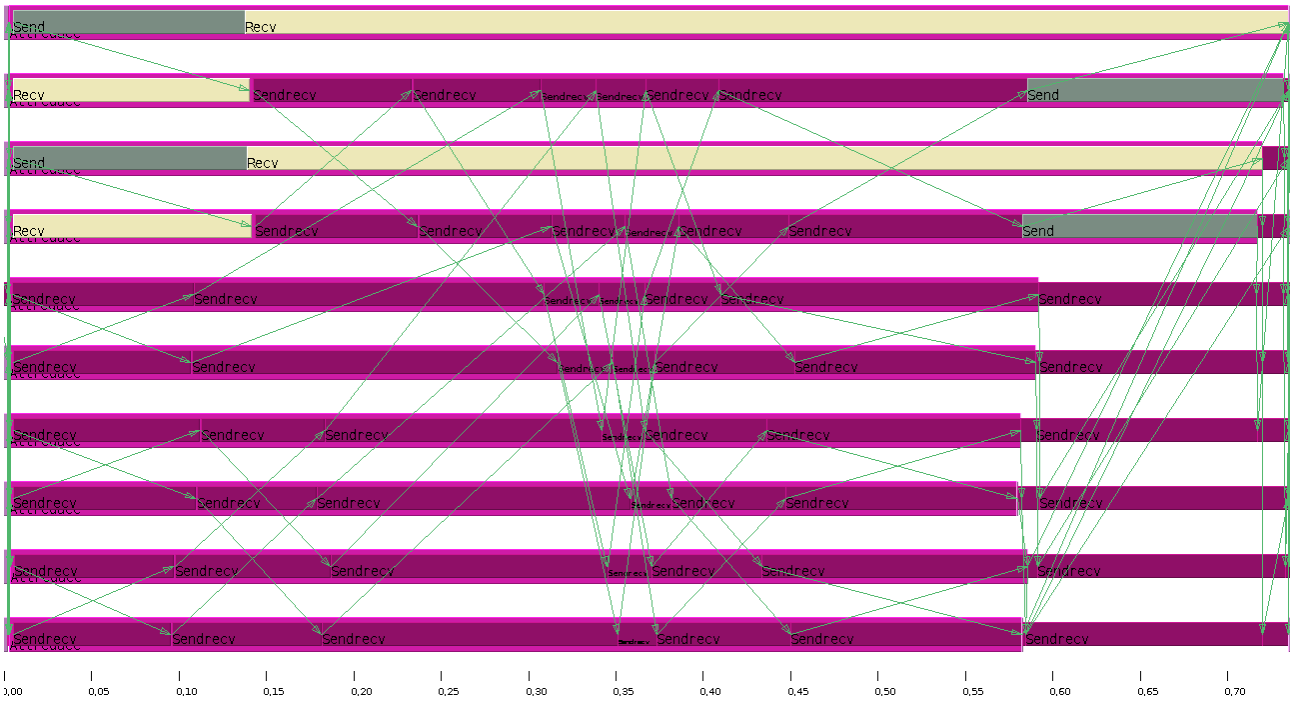
## 4.6.1 Cluster/Simulator 8/10 Screenshots



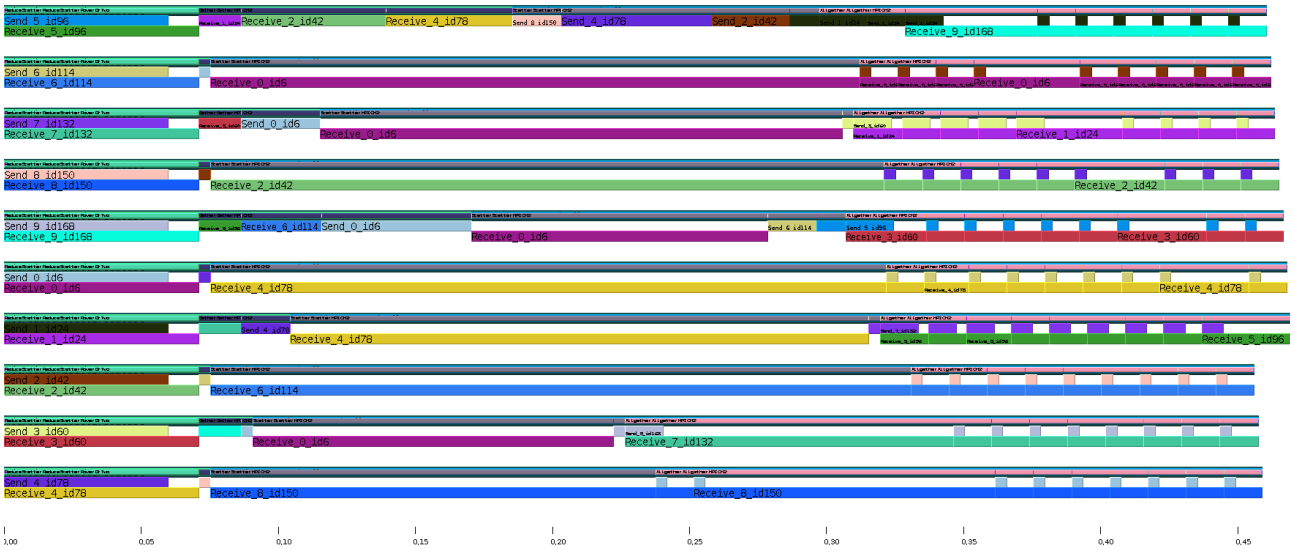

(Allreduce 8-8 Cluster)




(Allreduce 8-8 Simulator)




(Allreduce 10-10 Cluster)




(Allreduce 10-10 Simulator)

## 4.6.2 Beobachtungen:

Die Laufzeiten weichen ab (30% - 40%). Das Kommunikationsmuster sieht jedoch gut aus, was auf die korrekte Implementierung hinweist. Das Problem dürfte behoben sein, sobald MPI_Reduce optimiert wurde.

# 5. Fazit

Die Laufzeiten der einzelnen Simulationen schwankt teilweise sehr stark. Die Algorithmen sind aber korrekt implementiert. Das restliche Feintuning liegt nun in der Anpassung der Hardwareparameter des Simulators und dem Abfangen von Sonderfällen.

Das Ziel dieser Arbeit ist dennoch erreicht: Der Simulator verfügt nun über die nötigen Methoden, um alle benötigten kollektiven Operationen ausführen zu können.

# 6. Literaturangaben und Arbeitswerkzeug

Quellen:

- MPI Forum – Documentation (http://www.mpi-forum.org/)

Werkzeuge:

- PIOsimHD-Simulator
- Sunshot/Jumpshot zum Erstellen der Screenshots
- Eclipse zum Entwickeln der Funktionalitat
- OpenOffice zum Erstellen dieser Ausarbeitung