MPI-based Remote OpenMP Offloading

Präsentation von Mark Haube

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

informatik
die zukunft

Gliederung

- Definitionen
- Was ist OpenMP und wie wird es genutzt?
- UCX-basierte Version des OpenMP Plugins
- MPI-basierte Version des OpenMP Plugins
- Verbesserungen des MPI-basierten Plugins
- Ortsbezogenes Abladen
- Benutzerfreundlichkeit
- Testen des MPI-basierten Plugins

Definitionen (1)

Definitionen (1)

- HPC: High Performance Computing

Definitionen (1)

- HPC: High Performance Computing
- OpenMP: Open Multi-Processing

Definitionen (1)

- HPC: High Performance Computing
- OpenMP: Open Multi-Processing
  - API für die Programmierung mit geteiltem Speicher und Platformunabhängigkeit

Definitionen (1)

- HPC: High Performance Computing
- OpenMP: Open Multi-Processing
  - API für die Programmierung mit geteiltem Speicher und Platformunabhängigkeit
- MPI: Message Passing Interface

Definitionen (2)

Definitionen (2)

- UCX: Unified Communication X

Definitionen (2)

- UCX: Unified Communication X
- RPC: Remote Procedure Call

Definitionen (2)

- UCX: Unified Communication X
- RPC: Remote Procedure Call
- Slurm: Simple Linux Utility for Resource Management

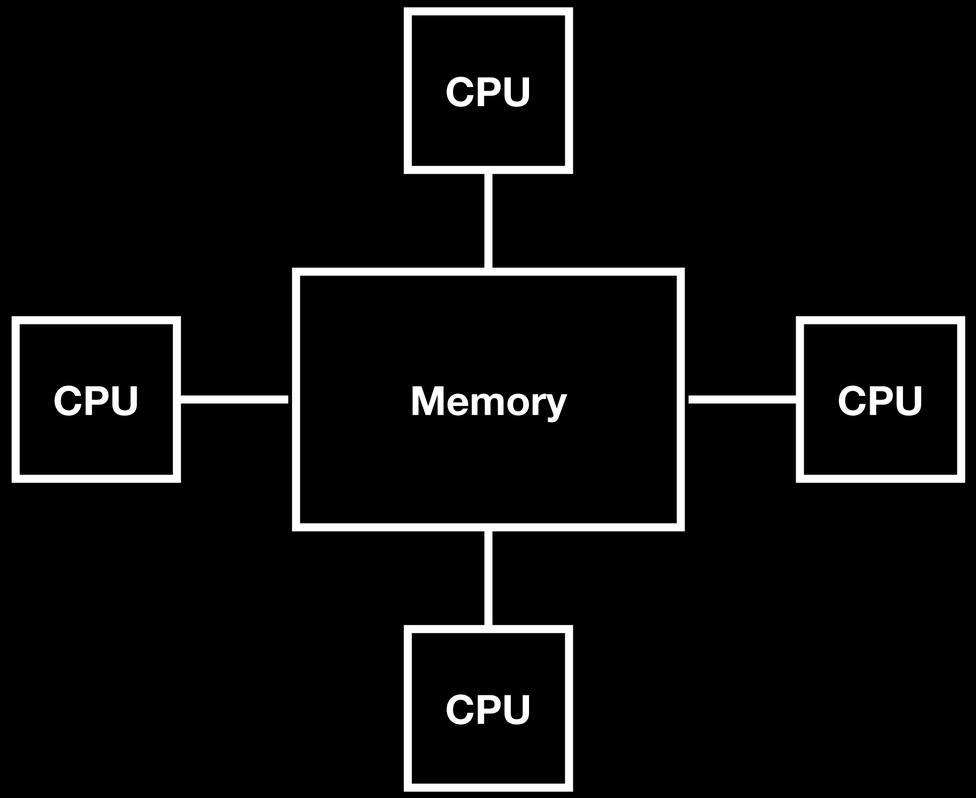
Definitionen (2)

- UCX: Unified Communication X
- RPC: Remote Procedure Call
- Slurm: Simple Linux Utility for Resource Management
- Heterogenes Rechnen: Systeme die verschiedene Hardware verwenden arbeiten zusammen

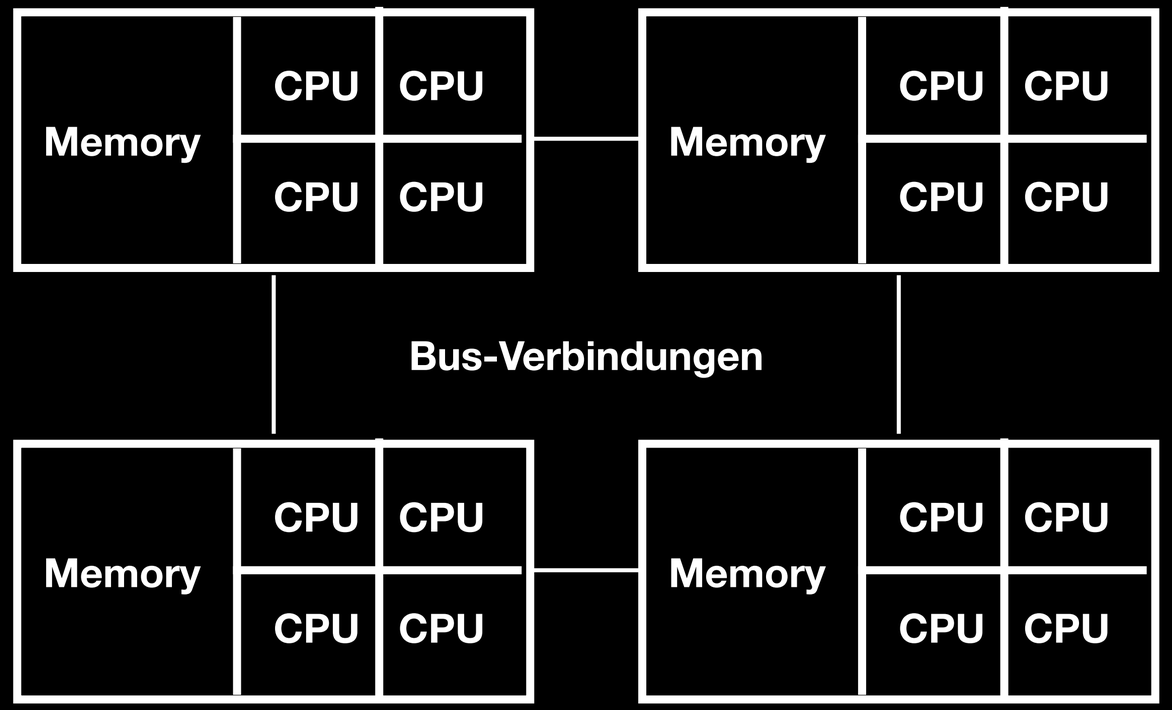
Motivation

- Ein Rechner mit vielen Prozessoren und Beschleunigern
- Zusammenarbeit heterogener Systeme
  - Geteilter Speicher
  - Geteilte Arbeit
  - Generelle Kommunikation
- Wir wollen wenig Aufwand in der Implementation

Unified Memory Access (UMA)



Non-Uniform Memory Access (NUMA)



Was ist OpenMP und wie wird es genutzt?

Was ist OpenMP und wie wird es genutzt?

- OpenMP ist das de facto Standard-Programmiermodell für HPCs

Was ist OpenMP und wie wird es genutzt?

- OpenMP ist das de facto Standard-Programmiermodell für HPCs
  - Einfaches Teilen des Speichers von mehreren heterogenen Systemen

Was ist OpenMP und wie wird es genutzt?

- OpenMP ist das de facto Standard-Programmiermodell für HPCs
  - Einfaches Teilen des Speichers von mehreren heterogenen Systemen
  - Unterstützt: ARM, AMDGCN, CUDA, PPC, Remote, VE, X86

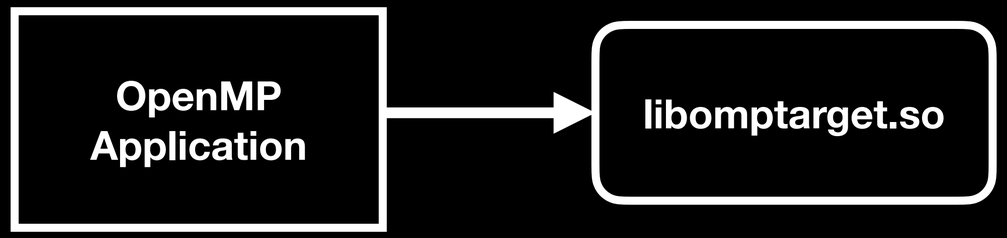
Was ist OpenMP und wie wird es genutzt?

- OpenMP ist das de facto Standard-Programmiermodell für HPCs
  - Einfaches Teilen des Speichers von mehreren heterogenen Systemen
  - Unterstützt: ARM, AMDGCN, CUDA, PPC, Remote, VE, X86
- Programmierer muss sich nicht mit den Hardwareschnittstellen beschäftigen (bspw. CUDA)

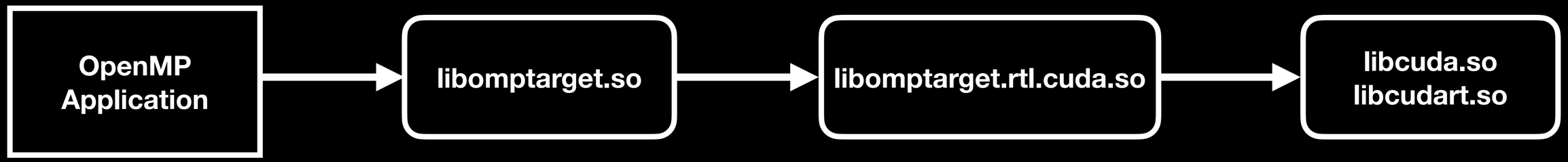
Was ist OpenMP und wie wird es genutzt?

- OpenMP ist das de facto Standard-Programmiermodell für HPCs
  - Einfaches Teilen des Speichers von mehreren heterogenen Systemen
  - Unterstützt: ARM, AMDGCN, CUDA, PPC, Remote, VE, X86
- Programmierer muss sich nicht mit den Hardwareschnittstellen beschäftigen (bspw. CUDA)
- LLVM/OpenMP: Integrierung in Compiler-Sammlung


OpenMP
Application


OpenMP
Application
libomptarget.so


OpenMP
Application
libomptarget.so
libomptarget.rtl.cuda.so


OpenMP
Application
libomptarget.so
libomptarget.rtl.cuda.so
libcuda.so
libcudart.so

UCX-basierte Version des OpenMP Plugins (1)

UCX-basierte Version des OpenMP Plugins (1)

- Verwendet RPC um Prozesse fernzusteuern

UCX-basierte Version des OpenMP Plugins (1)

- Verwendet RPC um Prozesse fernzusteuern
- Ein Klient, der das eigentliche OpenMP Programm repräsentiert

UCX-basierte Version des OpenMP Plugins (1)

- Verwendet RPC um Prozesse fernzusteuern
- Ein Klient, der das eigentliche OpenMP Programm repräsentiert
- Beliebig viele Rechenknoten, auf denen pro Beschleuniger (bspw. GPU) eine Instanz des Servers läuft

UCX-basierte Version des OpenMP Plugins (1)

- Verwendet RPC um Prozesse fernzusteuern
- Ein Klient, der das eigentliche OpenMP Programm repräsentiert
- Beliebig viele Rechenknoten, auf denen pro Beschleuniger (bspw. GPU) eine Instanz des Servers läuft
- Parameter werden serialisiert an Server per RPC geschickt

UCX-basierte Version des OpenMP Plugins (1)

- Verwendet RPC um Prozesse fernzusteuern
- Ein Klient, der das eigentliche OpenMP Programm repräsentiert
- Beliebig viele Rechenknoten, auf denen pro Beschleuniger (bspw. GPU) eine Instanz des Servers läuft
- Parameter werden serialisiert an Server per RPC geschickt
  - Dort de-serialisiert und die Prozedur mit Parametern ausgeführt

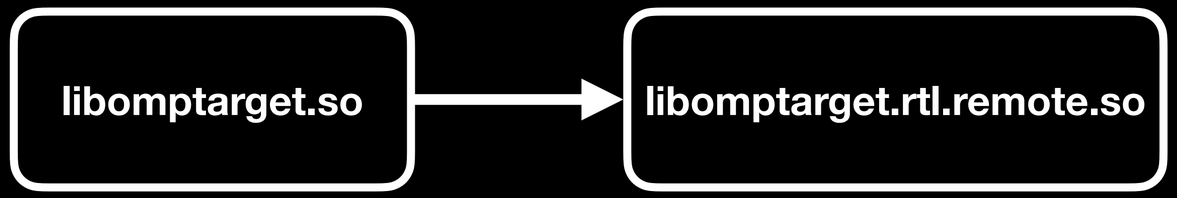
Client

Server

libomptarget.so

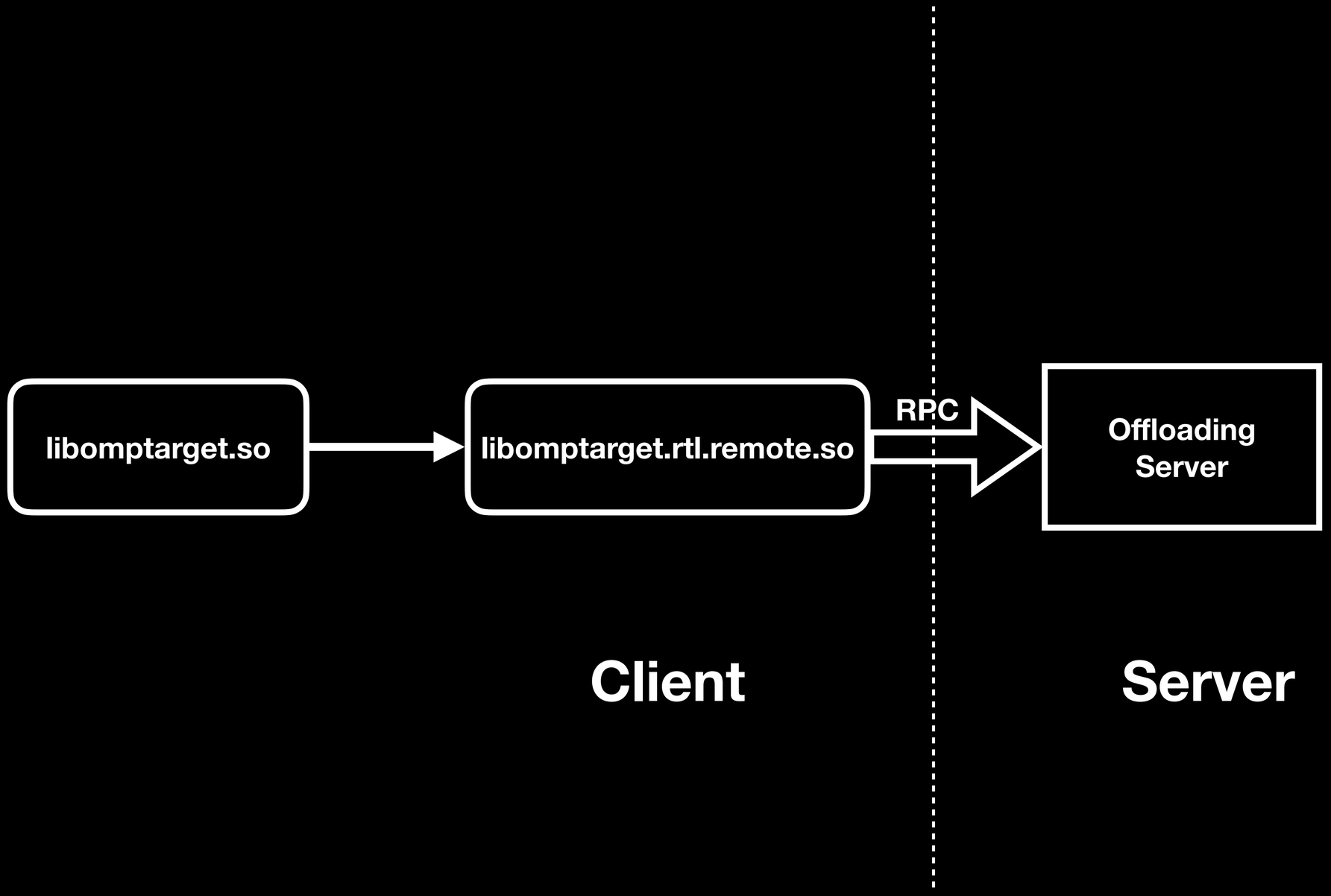
Client

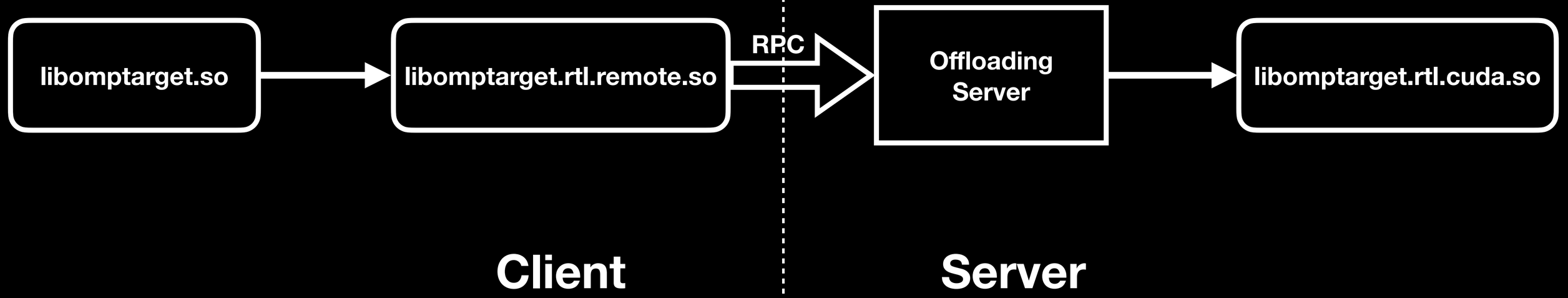
Server



Client

Server


libomptarget.so
libomptarget.rtl.remote.so
RPC
Offloading
Server
Client
Server


libomptarget.so
libomptarget.rtl.remote.so
RPC
Offloading
Server
libomptarget.rtl.cuda.so
Client
Server

UCX-basierte Version des OpenMP Plugins (2)

UCX-basierte Version des OpenMP Plugins (2)

- Ursprünglich nutzte das remote offloading Plugin gRPC als das RPC Backend

UCX-basierte Version des OpenMP Plugins (2)

- Ursprünglich nutzte das remote offloading Plugin gRPC als das RPC Backend
- Nun wird das UCX Backend verwendet

UCX-basierte Version des OpenMP Plugins (2)

- Ursprünglich nutzte das remote offloading Plugin gRPC als das RPC Backend
- Nun wird das UCX Backend verwendet
  - Da es Hochleistungsfähige Verbindungsleitungen wie Infiniband nutzen kann

MPI-basierte Version des OpenMP Plugins (1)

MPI-basierte Version des OpenMP Plugins (1)

- Nutzt MPI zur Kommunikation zwischen den Rechenknoten

MPI-basierte Version des OpenMP Plugins (1)

- Nutzt MPI zur Kommunikation zwischen den Rechenknoten
- Dezentrale Netzwerk-Architektur

MPI-basierte Version des OpenMP Plugins (1)

- Nutzt MPI zur Kommunikation zwischen den Rechenknoten
- Dezentrale Netzwerk-Architektur
  - D.h. kein zentraler Rechenknoten durch den alle Daten fließen müssen

MPI-basierte Version des OpenMP Plugins (1)

- Nutzt MPI zur Kommunikation zwischen den Rechenknoten
- Dezentrale Netzwerk-Architektur
  - D.h. kein zentraler Rechenknoten durch den alle Daten fließen müssen
- „Node 0“ wird als Host-Knoten gesetzt

MPI-basierte Version des OpenMP Plugins (1)

- Nutzt MPI zur Kommunikation zwischen den Rechenknoten
- Dezentrale Netzwerk-Architektur
  - D.h. kein zentraler Rechenknoten durch den alle Daten fließen müssen
- „Node 0“ wird als Host-Knoten gesetzt
  - Von hieraus wird das Programm unterteilt und zum parallelen abarbeiten auf die anderen Rechenknoten verteilt

MPI-basierte Version des OpenMP Plugins (2)

MPI-basierte Version des OpenMP Plugins (2)

- „Manager“-Klasse:

MPI-basierte Version des OpenMP Plugins (2)

- „Manager“-Klasse:
  - Findet alle lokalen Beschleuniger

MPI-basierte Version des OpenMP Plugins (2)

- „Manager“-Klasse:
  - Findet alle lokalen Beschleuniger
  - Sucht dann mithilfe von MPI Kommunikation andere Beschleuniger auf anderen Rechenknoten

MPI-basierte Version des OpenMP Plugins (2)

- „Manager“-Klasse:
  - Findet alle lokalen Beschleuniger
  - Sucht dann mithilfe von MPI Kommunikation andere Beschleuniger auf anderen Rechenknoten
  - Baut aus den gefunden Beschleunigern eine Tabelle aus allen bereitstehenden Beschleunigern

MPI-basierte Version des OpenMP Plugins (2)

- „Manager“-Klasse:
  - Findet alle lokalen Beschleuniger
  - Sucht dann mithilfe von MPI Kommunikation andere Beschleuniger auf anderen Rechenknoten
  - Baut aus den gefunden Beschleunigern eine Tabelle aus allen bereitstehenden Beschleunigern
- => Mit dieser Tabelle werden dann alle erreichbaren Beschleuniger verwaltet

Verbesserungen des MPI-basierten Plugins (1)

Verbesserungen des MPI-basierten Plugins (1)

- Vergleich gegenüber der UCX-basierten Version des Plugins

Verbesserungen des MPI-basierten Plugins (1)

- Vergleich gegenüber der UCX-basierten Version des Plugins
- Die UCX-basierte Version ist zentralisiert:

Verbesserungen des MPI-basierten Plugins (1)

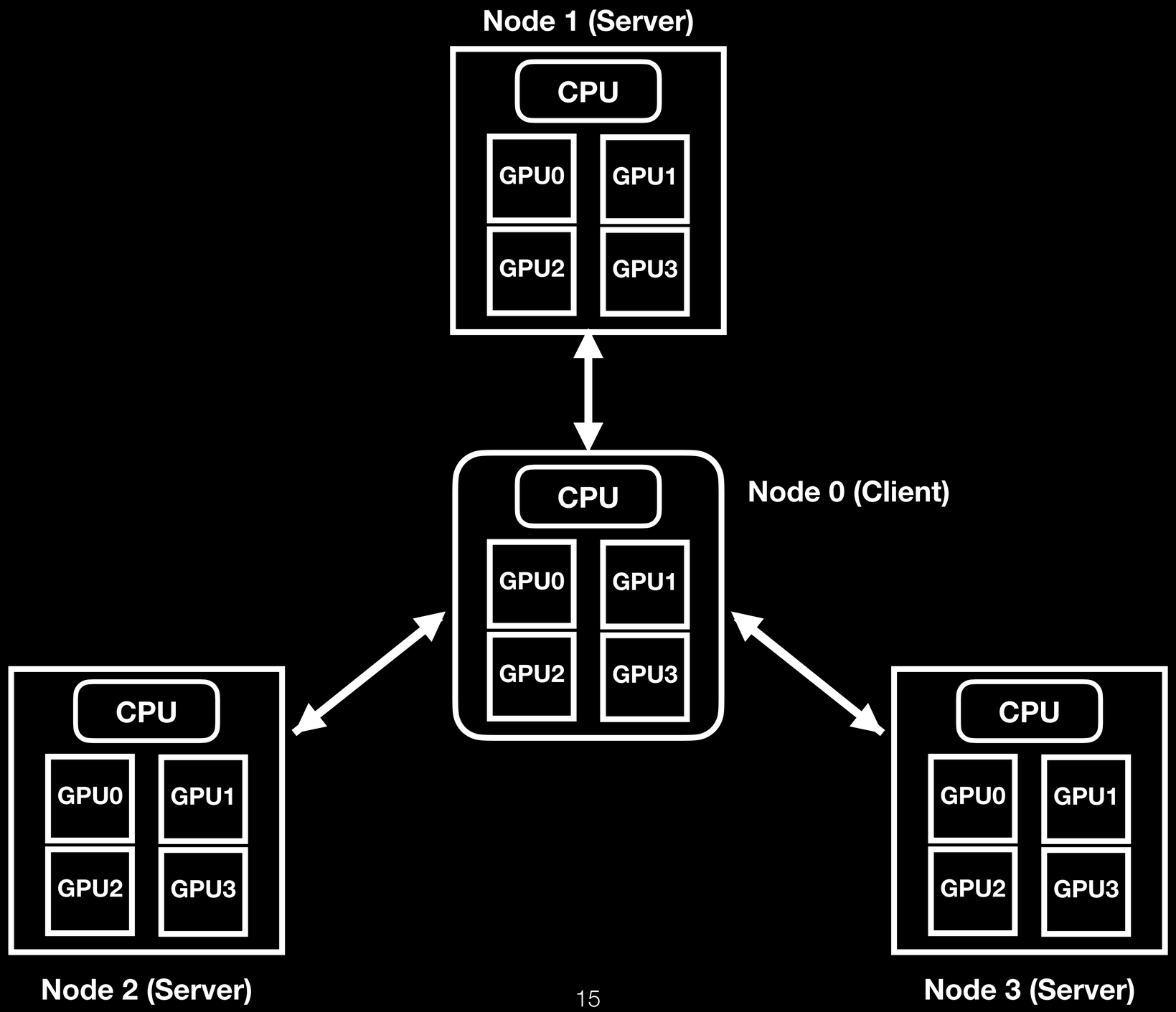
- Vergleich gegenüber der UCX-basierten Version des Plugins
- Die UCX-basierte Version ist zentralisiert:
  - D.h. alle Datenübertragungen zwischen Beschleunigern müssen durch den zentralen Rechenknoten fließen

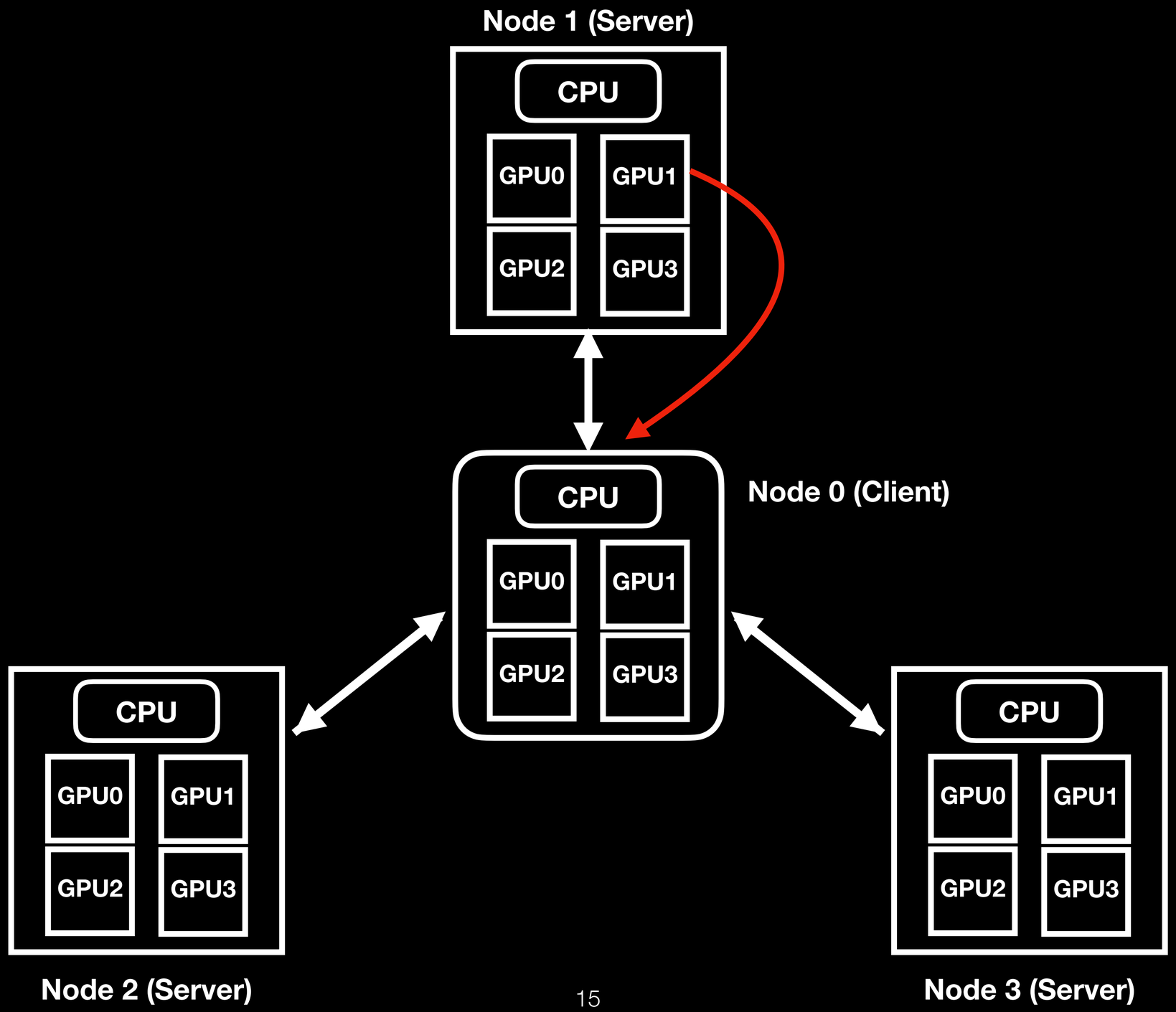
Verbesserungen des MPI-basierten Plugins (1)

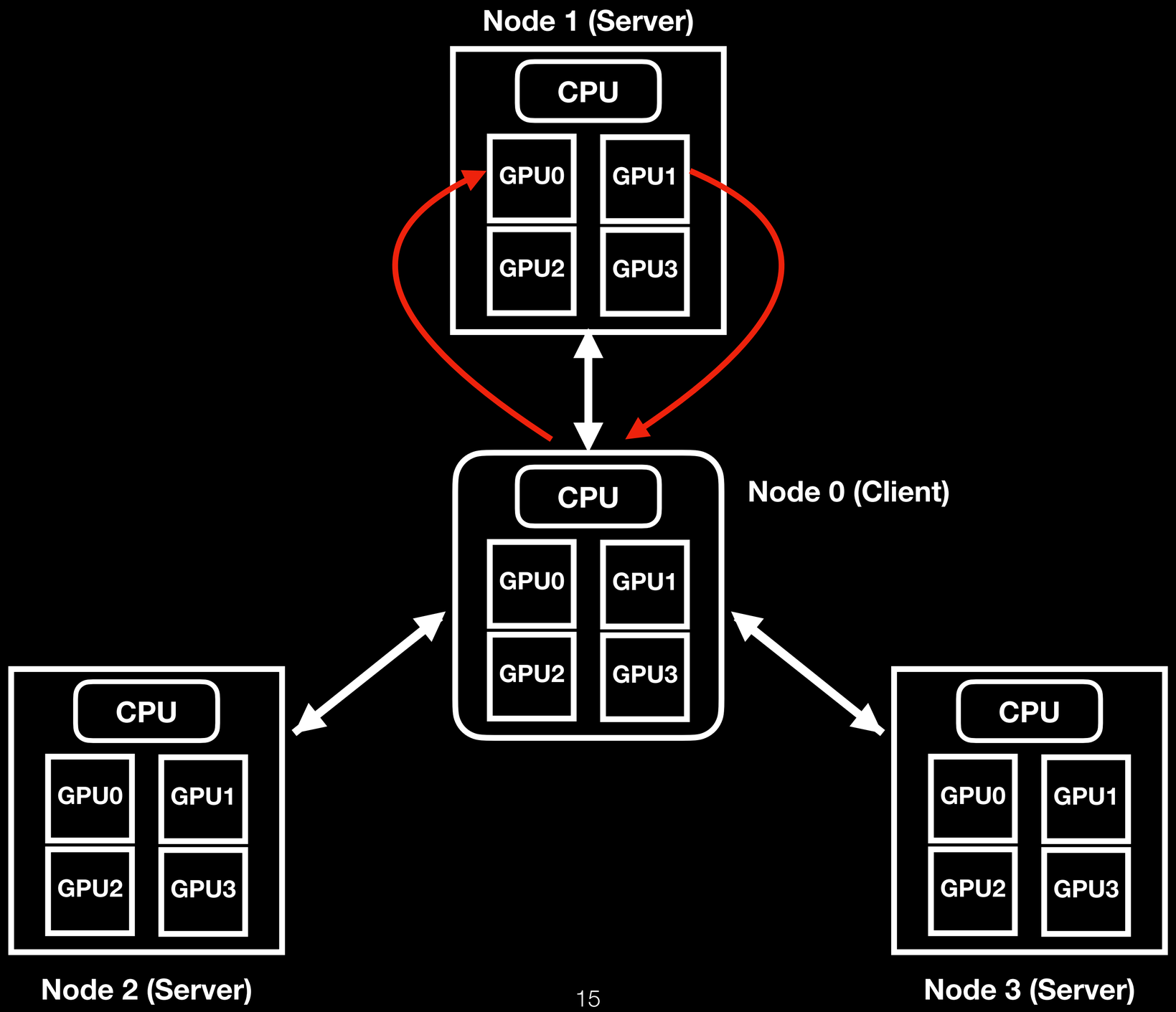
- Vergleich gegenüber der UCX-basierten Version des Plugins
- Die UCX-basierte Version ist zentralisiert:
  - D.h. alle Datenübertragungen zwischen Beschleunigern müssen durch den zentralen Rechenknoten fließen
  - Führt zu einem Engpass bei der Befehlsübermittlung bei vielen Befehlen

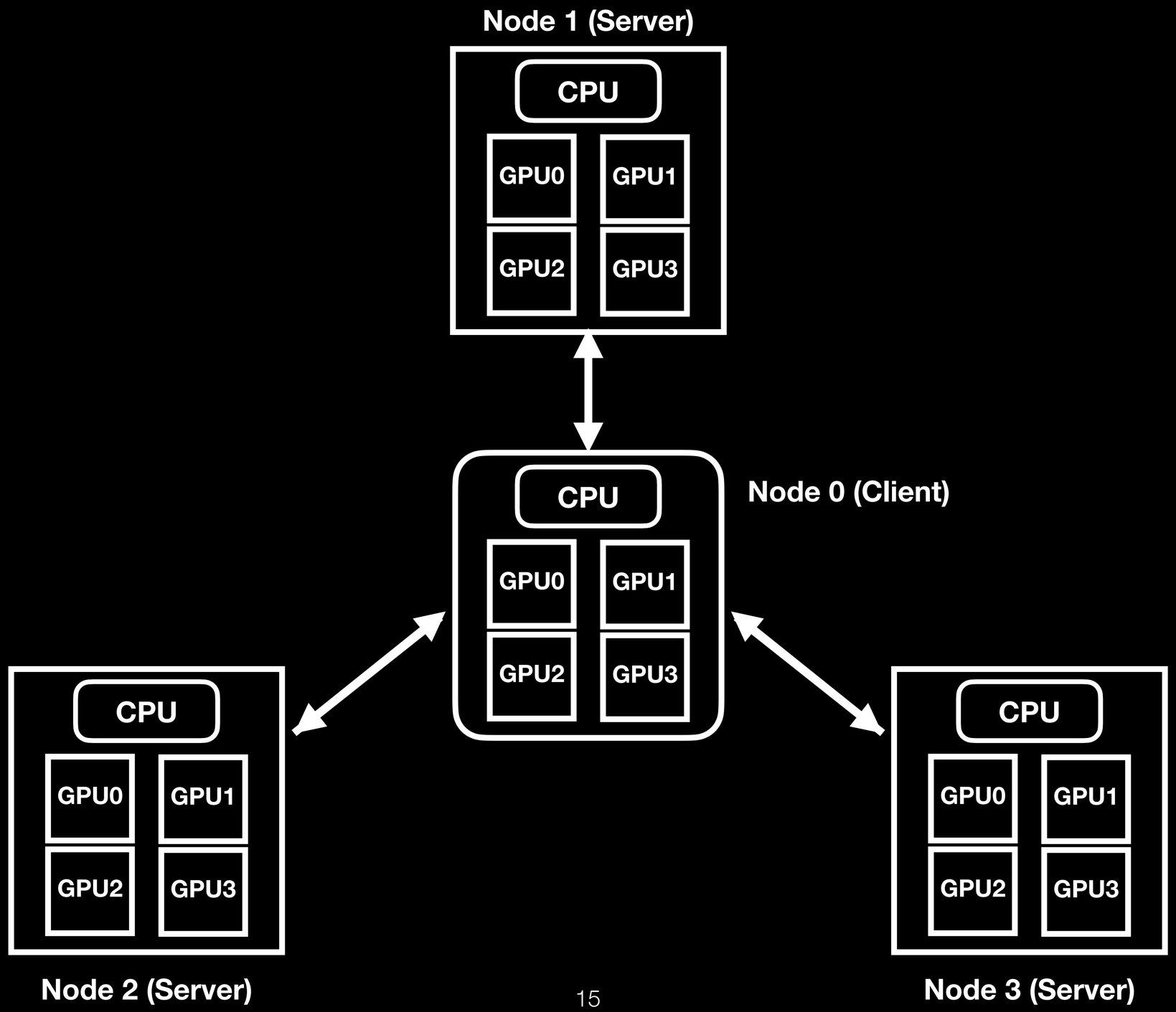
Verbesserungen des MPI-basierten Plugins (1)

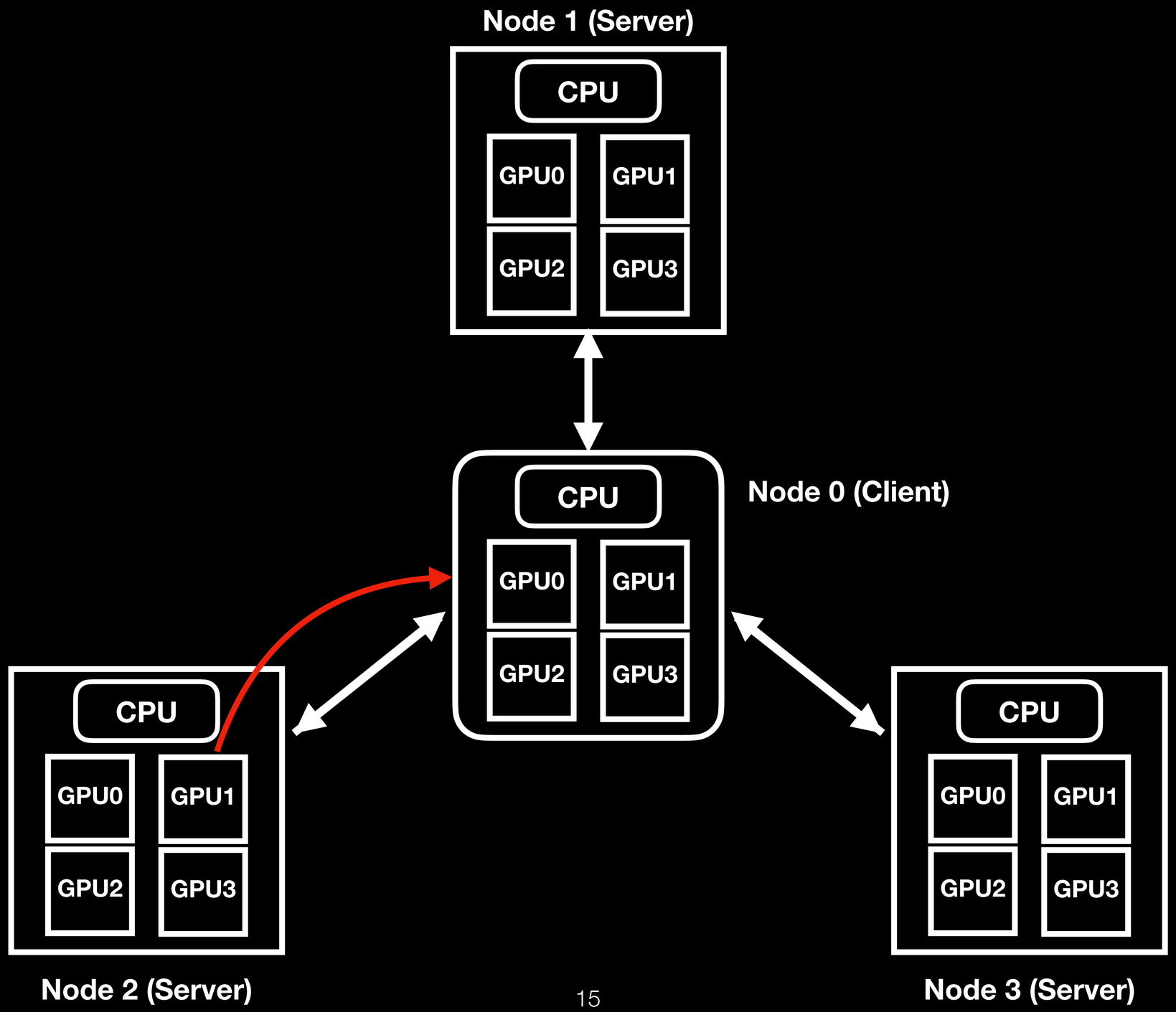
- Vergleich gegenüber der UCX-basierten Version des Plugins
- Die UCX-basierte Version ist zentralisiert:
  - D.h. alle Datenübertragungen zwischen Beschleunigern müssen durch den zentralen Rechenknoten fließen
  - Führt zu einem Engpass bei der Befehlsübermittlung bei vielen Befehlen
  - Führt zu Speichermangel im zentralen Rechenknoten, da alle Daten dort zwischengespeichert werden müssen

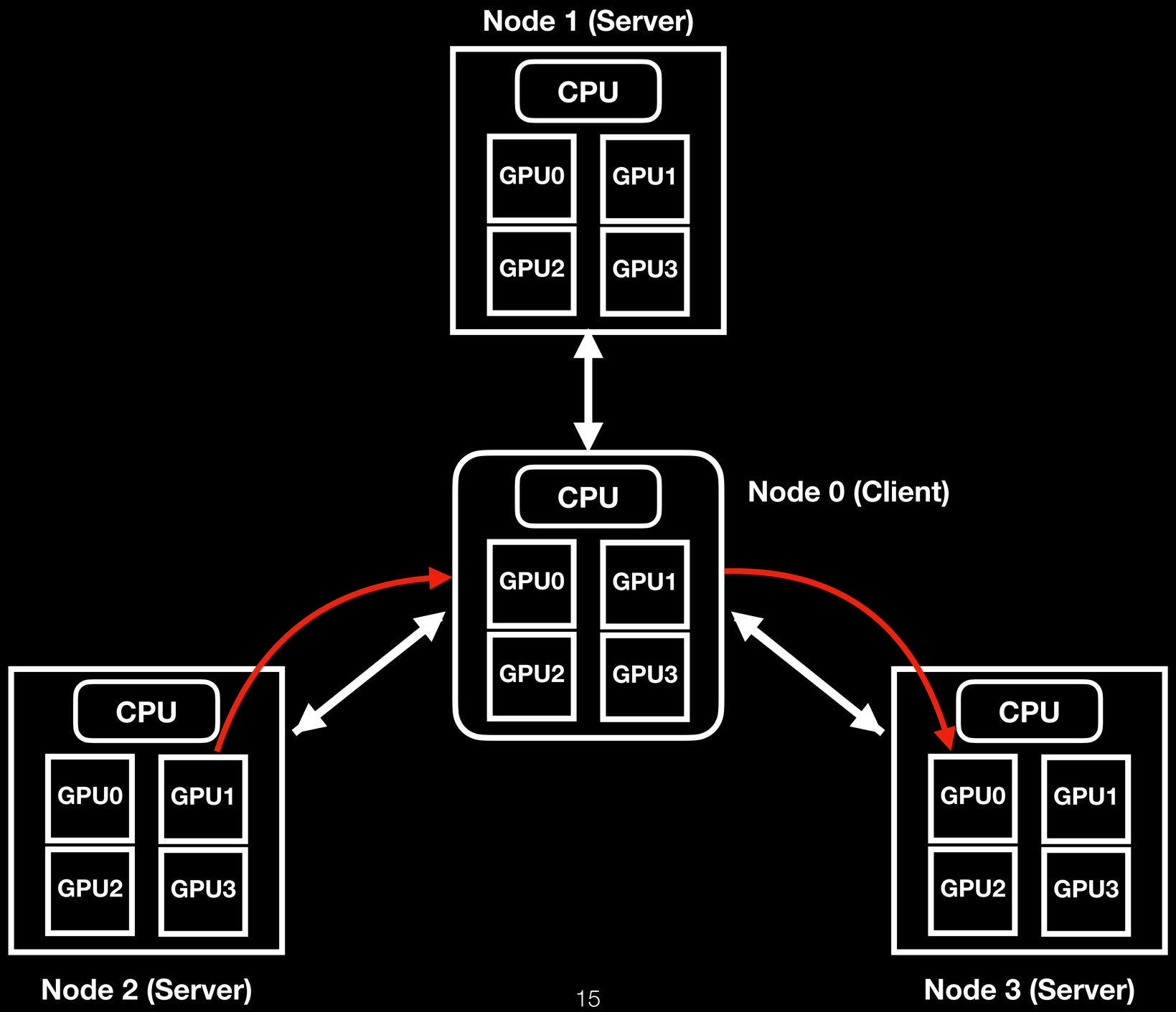

Node 1 (Server)
CPU
GPU0
GPU1
GPU2
GPU3
CPU
GPU0
GPU1
GPU2
GPU3
Node 0 (Client)
CPU
GPU0
GPU1
GPU2
GPU3
Node 2 (Server)
CPU
GPU0
GPU1
GPU2
GPU3
Node 3 (Server)


Node 1 (Server)
CPU
GPU0
GPU1
GPU2
GPU3
CPU
GPU0
GPU1
GPU2
GPU3
Node 0 (Client)
CPU
GPU0
GPU1
GPU2
GPU3
Node 2 (Server)
CPU
GPU0
GPU1
GPU2
GPU3
Node 3 (Server)


Node 1 (Server)
CPU
GPU0
GPU1
GPU2
GPU3
CPU
GPU0
GPU1
GPU2
GPU3
Node 0 (Client)
CPU
GPU0
GPU1
GPU2
GPU3
Node 2 (Server)
CPU
GPU0
GPU1
GPU2
GPU3
Node 3 (Server)


Node 1 (Server)
CPU
GPU0
GPU1
GPU2
GPU3
CPU
GPU0
GPU1
GPU2
GPU3
Node 0 (Client)
CPU
GPU0
GPU1
GPU2
GPU3
Node 2 (Server)
CPU
GPU0
GPU1
GPU2
GPU3
Node 3 (Server)


Node 1 (Server)
CPU
GPU0
GPU1
GPU2
GPU3
CPU
GPU0
GPU1
GPU2
GPU3
Node 0 (Client)
CPU
GPU0
GPU1
GPU2
GPU3
Node 2 (Server)
CPU
GPU0
GPU1
GPU2
GPU3
Node 3 (Server)


Node 1 (Server)
CPU
GPU0
GPU1
GPU2
GPU3
CPU
GPU0
GPU1
GPU2
GPU3
Node 0 (Client)
CPU
GPU0
GPU1
GPU2
GPU3
Node 2 (Server)
CPU
GPU0
GPU1
GPU2
GPU3
Node 3 (Server)

Verbesserungen des MPI-basierten Plugins (2)

Verbesserungen des MPI-basierten Plugins (2)

- Die MPI-basierte Version ist dezentralisiert:

Verbesserungen des MPI-basierten Plugins (2)

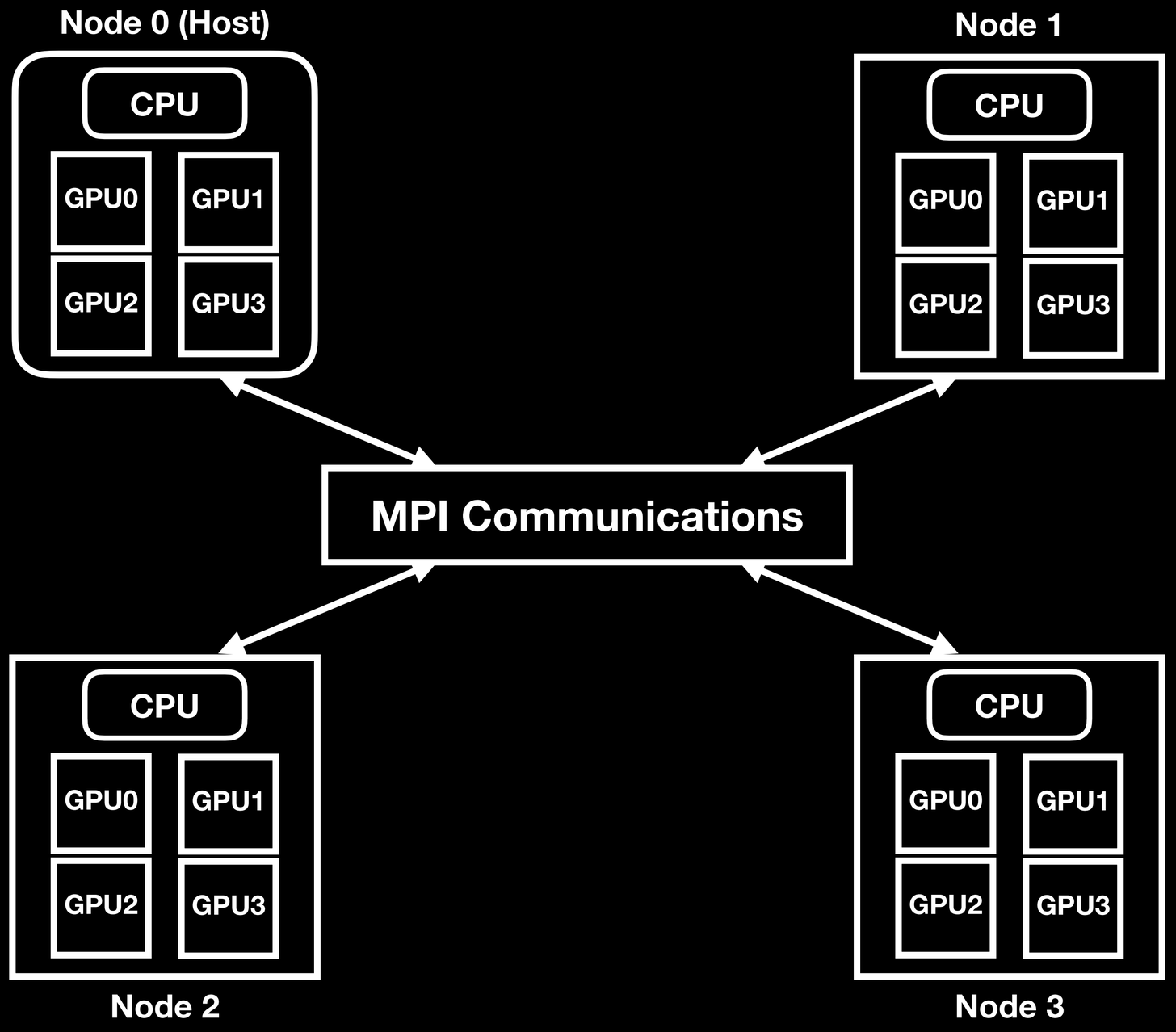
- Die MPI-basierte Version ist dezentralisiert:
  - D.h. die Probleme der UCX-basierten Version treten hier nicht auf

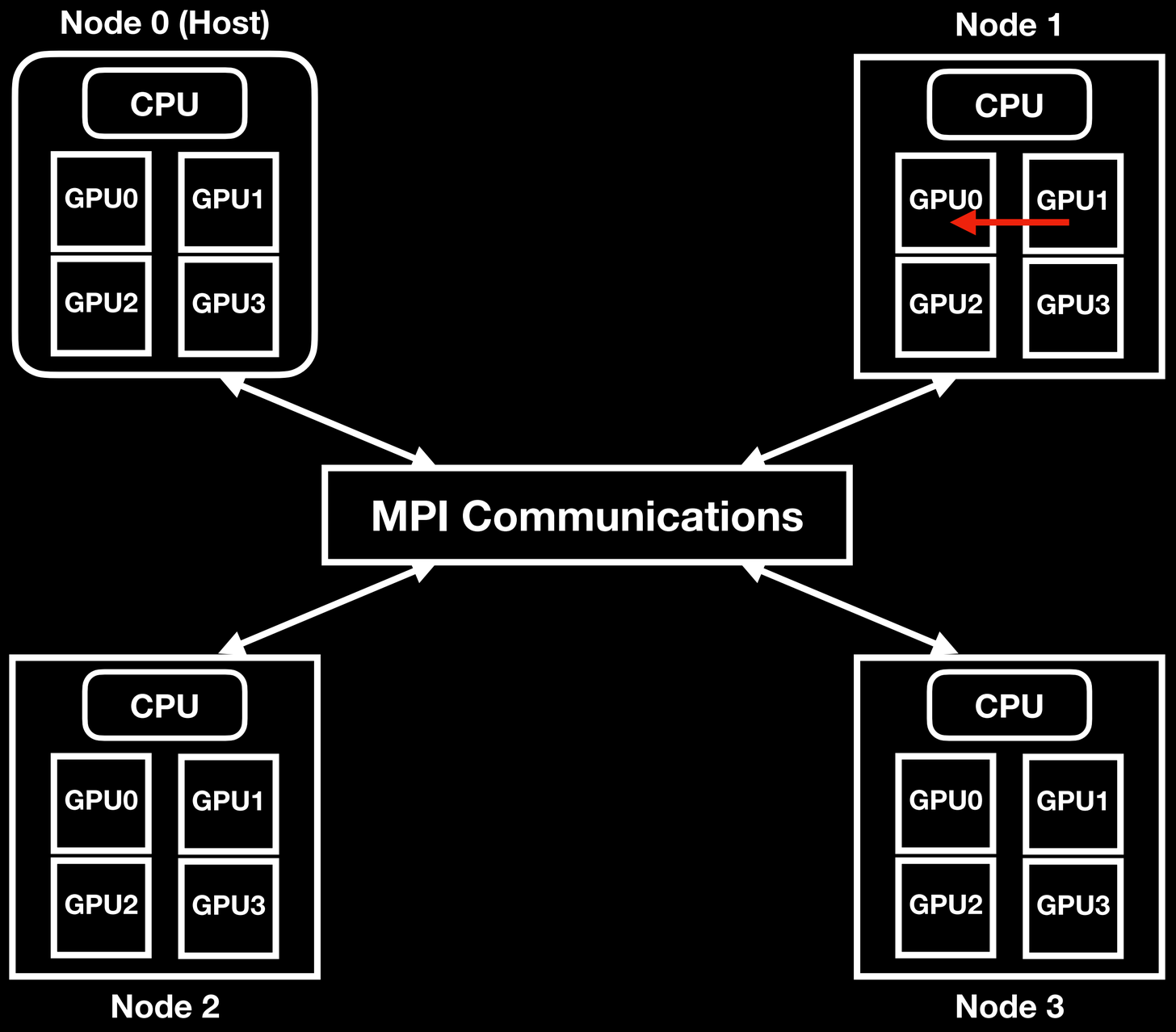
Verbesserungen des MPI-basierten Plugins (2)

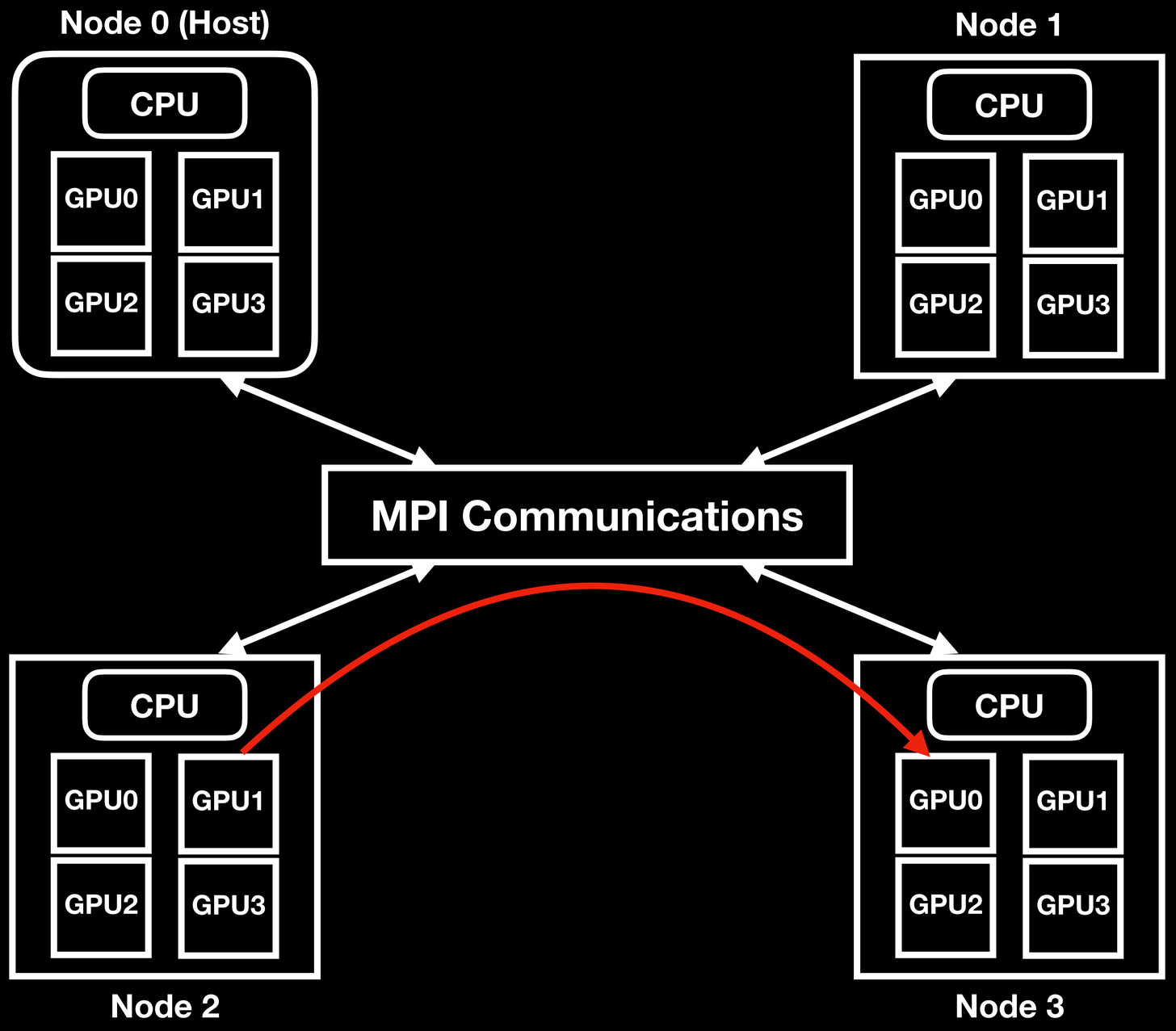
- Die MPI-basierte Version ist dezentralisiert:
  - D.h. die Probleme der UCX-basierten Version treten hier nicht auf
  - MPI erlaubt für Kommunikation zwischen zwei beliebigen Knoten

Verbesserungen des MPI-basierten Plugins (2)

- Die MPI-basierte Version ist dezentralisiert:
  - D.h. die Probleme der UCX-basierten Version treten hier nicht auf
  - MPI erlaubt für Kommunikation zwischen zwei beliebigen Knoten
  - Daher können Rechenknoten direkt auf Daten eines anderen Beschleunigers zugreifen


Node 0 (Host)
CPU
GPU0
GPU1
GPU2
GPU3
Node 1
CPU
GPU0
GPU1
GPU2
GPU3
MPI Communications
CPU
GPU0
GPU1
GPU2
GPU3
Node 2
CPU
GPU0
GPU1
GPU2
GPU3
Node 3


Node 0 (Host)
CPU
GPU0
GPU1
GPU2
GPU3
Node 1
CPU
GPU0
GPU1
GPU2
GPU3
MPI Communications
CPU
GPU0
GPU1
GPU2
GPU3
Node 2
CPU
GPU0
GPU1
GPU2
GPU3
Node 3


Node 0 (Host)
CPU
GPU0
GPU1
GPU2
GPU3
Node 1
CPU
GPU0
GPU1
GPU2
GPU3
MPI Communications
CPU
GPU0
GPU1
GPU2
GPU3
Node 2
CPU
GPU0
GPU1
GPU2
GPU3
Node 3

Ortsbezogenes Abladen (1)

Ortsbezogenes Abladen (1)

- Verwendung von verschiedenen Vorgehen bei Datenübertragungen zu anderen Beschleunigern

Ortsbezogenes Abladen (1)

- Verwendung von verschiedenen Vorgehen bei Datenübertragungen zu anderen Beschleunigern
  - OpenMP Entwickler müssen nichts ändern, sondern das Plugin unterscheidet zwischen „Inter-Node“- und „Intra-Node“-Kommunikation

Ortsbezogenes Abladen (1)

- Verwendung von verschiedenen Vorgehen bei Datenübertragungen zu anderen Beschleunigern
  - OpenMP Entwickler müssen nichts ändern, sondern das Plugin unterscheidet zwischen „Inter-Node“- und „Intra-Node“-Kommunikation
- Vorgehen des MPI-basierten Plugins:

Ortsbezogenes Abladen (1)

- Verwendung von verschiedenen Vorgehen bei Datenübertragungen zu anderen Beschleunigern
  - OpenMP Entwickler müssen nichts ändern, sondern das Plugin unterscheidet zwischen „Inter-Node“- und „Intra-Node“-Kommunikation
- Vorgehen des MPI-basierten Plugins:
  - Intra-Node: Daten werden per P2P direkt über PCIe oder NVLink übertragen

Ortsbezogenes Abladen (1)

- Verwendung von verschiedenen Vorgehen bei Datenübertragungen zu anderen Beschleunigern
  - OpenMP Entwickler müssen nichts ändern, sondern das Plugin unterscheidet zwischen „Inter-Node“- und „Intra-Node“-Kommunikation
- Vorgehen des MPI-basierten Plugins:
  - Intra-Node: Daten werden per P2P direkt über PCIe oder NVLink übertragen
  - Inter-Node: Plugin verwendet MPI zur Datenübertragung

Ortsbezogenes Abladen (1)

- Verwendung von verschiedenen Vorgehen bei Datenübertragungen zu anderen Beschleunigern
  - OpenMP Entwickler müssen nichts ändern, sondern das Plugin unterscheidet zwischen „Inter-Node“- und „Intra-Node“-Kommunikation
- Vorgehen des MPI-basierten Plugins:
  - Intra-Node: Daten werden per P2P direkt über PCIe oder NVLink übertragen
  - Inter-Node: Plugin verwendet MPI zur Datenübertragung
- => Netzwerkbelastung wird verringert (weniger Datenübertragungen über das Netzwerk)

Ortsbezogenes Abladen (2)

Ortsbezogenes Abladen (2)

- Zukünftige Pläne für das MPI-basierte Plugin:

Ortsbezogenes Abladen (2)

- Zukünftige Pläne für das MPI-basierte Plugin:
  - Bei Datenübertragung von „Host“-Knoten zu Rechenknoten werden Daten meist mehrmals übertragen (für jeden Beschleuniger des Rechenknotens)

Ortsbezogenes Abladen (2)

- Zukünftige Pläne für das MPI-basierte Plugin:
  - Bei Datenübertragung von „Host“-Knoten zu Rechenknoten werden Daten meist mehrmals übertragen (für jeden Beschleuniger des Rechenknotens)
  - Daten sollen also zukünftig nur einmal in den CPU-Speicher übertragen werden und dann in die einzelnen Beschleuniger

Ortsbezogenes Abladen (2)

- Zukünftige Pläne für das MPI-basierte Plugin:
  - Bei Datenübertragung von „Host“-Knoten zu Rechenknoten werden Daten meist mehrmals übertragen (für jeden Beschleuniger des Rechenknotens)
  - Daten sollen also zukünftig nur einmal in den CPU-Speicher übertragen werden und dann in die einzelnen Beschleuniger
  - => geringere Netzwerkbelastung

Benutzerfreundlichkeit (1)

Benutzerfreundlichkeit (1)

- OpenMP für einfachere Entwicklung

Benutzerfreundlichkeit (1)

- OpenMP für einfachere Entwicklung
  - Anstatt die Hardware-Schnittstellen wie bspw. CUDA lernen zu müssen

Benutzerfreundlichkeit (1)

- OpenMP für einfachere Entwicklung
  - Anstatt die Hardware-Schnittstellen wie bspw. CUDA lernen zu müssen
- UCX-basierte Version des Plugins ist kompliziert aufzusetzen und zu verwenden

Benutzerfreundlichkeit (1)

- OpenMP für einfachere Entwicklung
  - Anstatt die Hardware-Schnittstellen wie bspw. CUDA lernen zu müssen
- UCX-basierte Version des Plugins ist kompliziert aufzusetzen und zu verwenden
  - Es muss eine entsprechende Entwicklungsumgebung aufgesetzt werden und extra Software installiert werden

Benutzerfreundlichkeit (1)

- OpenMP für einfachere Entwicklung
  - Anstatt die Hardware-Schnittstellen wie bspw. CUDA lernen zu müssen
- UCX-basierte Version des Plugins ist kompliziert aufzusetzen und zu verwenden
  - Es muss eine entsprechende Entwicklungsumgebung aufgesetzt werden und extra Software installiert werden
  - Klient und Server IPs und ports müssen manuell vergeben werden

Benutzerfreundlichkeit (2)

Benutzerfreundlichkeit (2)

- UCX-basierte Version des Plugins ist am effizientesten wenn ein Arbeiter-Programm pro Beschleuniger läuft

Benutzerfreundlichkeit (2)

- UCX-basierte Version des Plugins ist am effizientesten wenn ein Arbeiter-Programm pro Beschleuniger läuft
  - Dies ist viel Arbeitsaufwand für den Entwickler des HPC-Programms

Benutzerfreundlichkeit (2)

- UCX-basierte Version des Plugins ist am effizientesten wenn ein Arbeiter-Programm pro Beschleuniger läuft
  - Dies ist viel Arbeitsaufwand für den Entwickler des HPC-Programms
  - Man könnte einfach einen Beschleuniger pro Rechenknoten haben, jedoch ist dies platztechnisch für Rechenzentren schlecht

Benutzerfreundlichkeit (3)

Benutzerfreundlichkeit (3)

- MPI-basierte Version des Plugins:

Benutzerfreundlichkeit (3)

- MPI-basierte Version des Plugins:
  - Benötigt keine zusätzliche Einrichtung

Benutzerfreundlichkeit (3)

- MPI-basierte Version des Plugins:
  - Benötigt keine zusätzliche Einrichtung
  - Einfaches Ausführen des HPC-Programms mittels Slurm

Benutzerfreundlichkeit (3)

- MPI-basierte Version des Plugins:
  - Benötigt keine zusätzliche Einrichtung
  - Einfaches Ausführen des HPC-Programms mittels Slurm

```
. srun --pty --exclusive -N 4 ./XSBench -m event
```

Benutzerfreundlichkeit (3)

- MPI-basierte Version des Plugins:
  - Benötigt keine zusätzliche Einrichtung
  - Einfaches Ausführen des HPC-Programms mittels Slurm

```
. srun --pty --exclusive -N 4 ./XSBench -m event
```

  - Ohne Slurm kann das Plugin einfach als normales MPI-Programm ausgeführt werden

Benutzerfreundlichkeit (3)

- MPI-basierte Version des Plugins:
  - Benötigt keine zusätzliche Einrichtung
  - Einfaches Ausführen des HPC-Programms mittels Slurm

```
. srun --pty --exclusive -N 4 ./XSBench -m event
```

  - Ohne Slurm kann das Plugin einfach als normales MPI-Programm ausgeführt werden
  - => sehr viel weniger Entwicklungs- und Betriebsaufwand

Testen des MPI-basierten Plugins

Testen des MPI-basierten Plugins

- Baseline: Originales UCX-basiertes Plugin

Testen des MPI-basierten Plugins

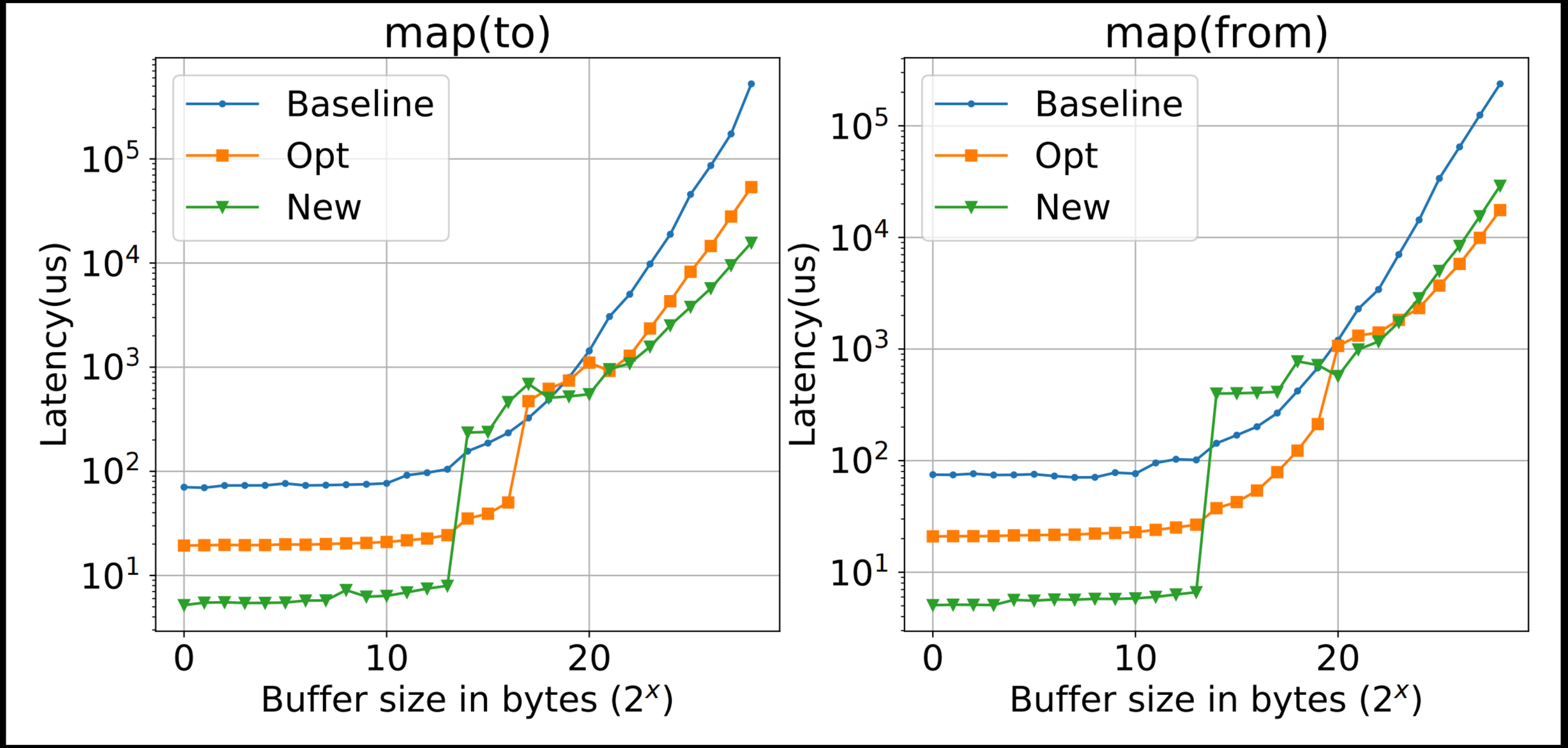
- Baseline: Originales UCX-basiertes Plugin
- Opt: UCX-basiertes Plugin mit Optimierungen

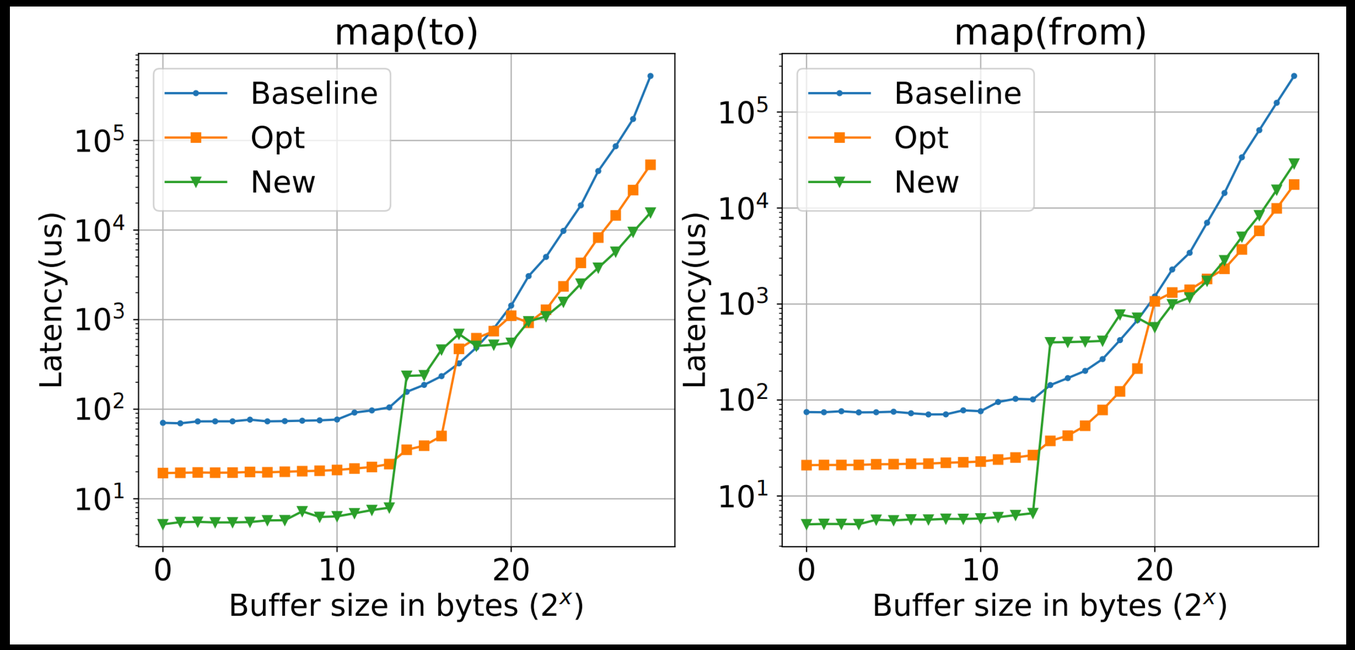
Testen des MPI-basierten Plugins

- Baseline: Originales UCX-basiertes Plugin
- Opt: UCX-basiertes Plugin mit Optimierungen
- Opt-L: UCX-basiertes Plugin mit Klientenseitigen Abladen

Testen des MPI-basierten Plugins

- Baseline: Originales UCX-basiertes Plugin
- Opt: UCX-basiertes Plugin mit Optimierungen
- Opt-L: UCX-basiertes Plugin mit Klientenseitigen Abladen
- New: Das neue MPI-basierte Plugin

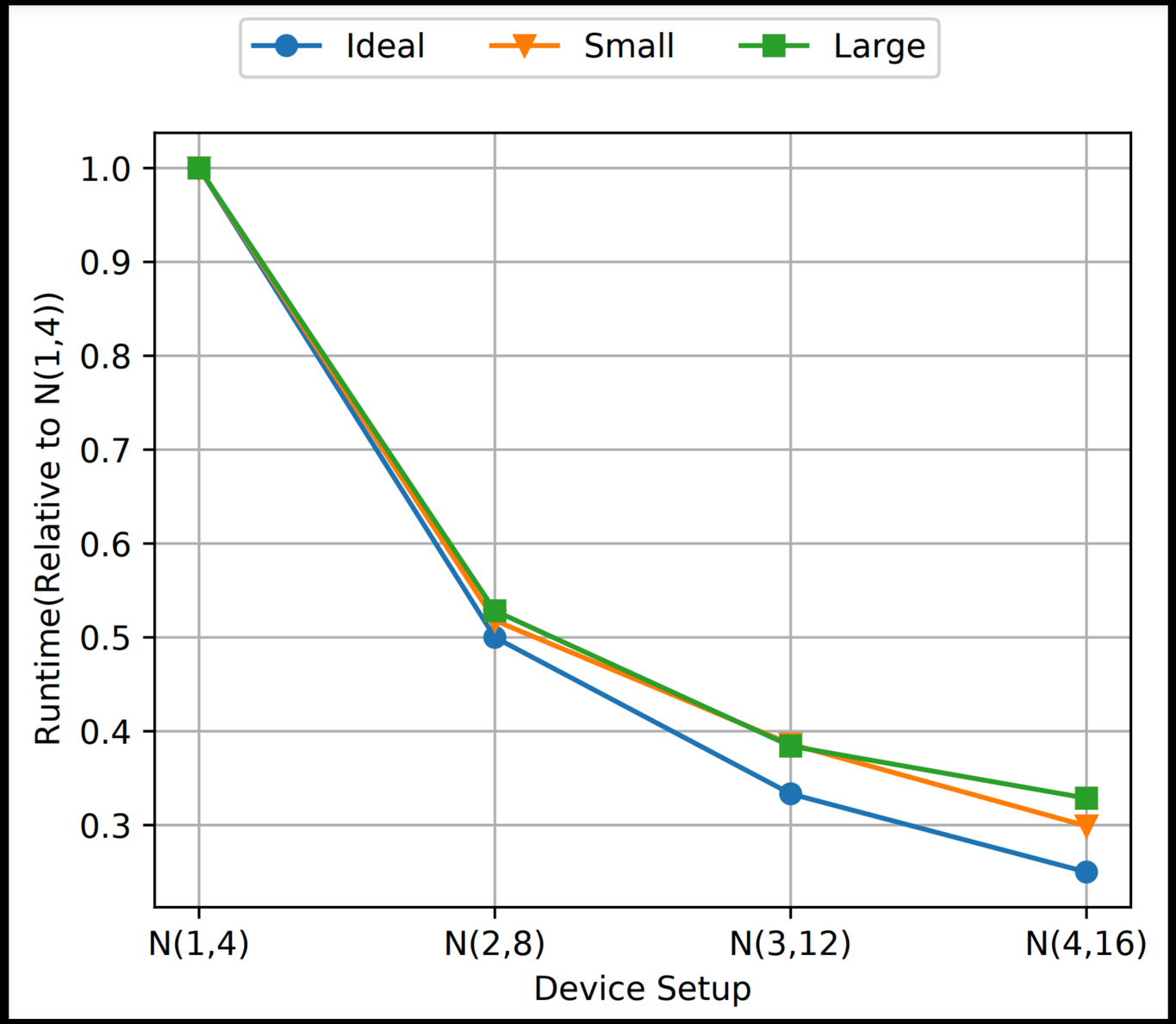

map(to)
Baseline
Opt
New
Latency(us)
10^5
10^4
10^3
10^2
10^1
0
10
20
Buffer size in bytes (2^x)
map(from)
Baseline
Opt
New
Latency(us)
10^5
10^4
10^3
10^2
10^1
0
10
20
Buffer size in bytes (2^x)



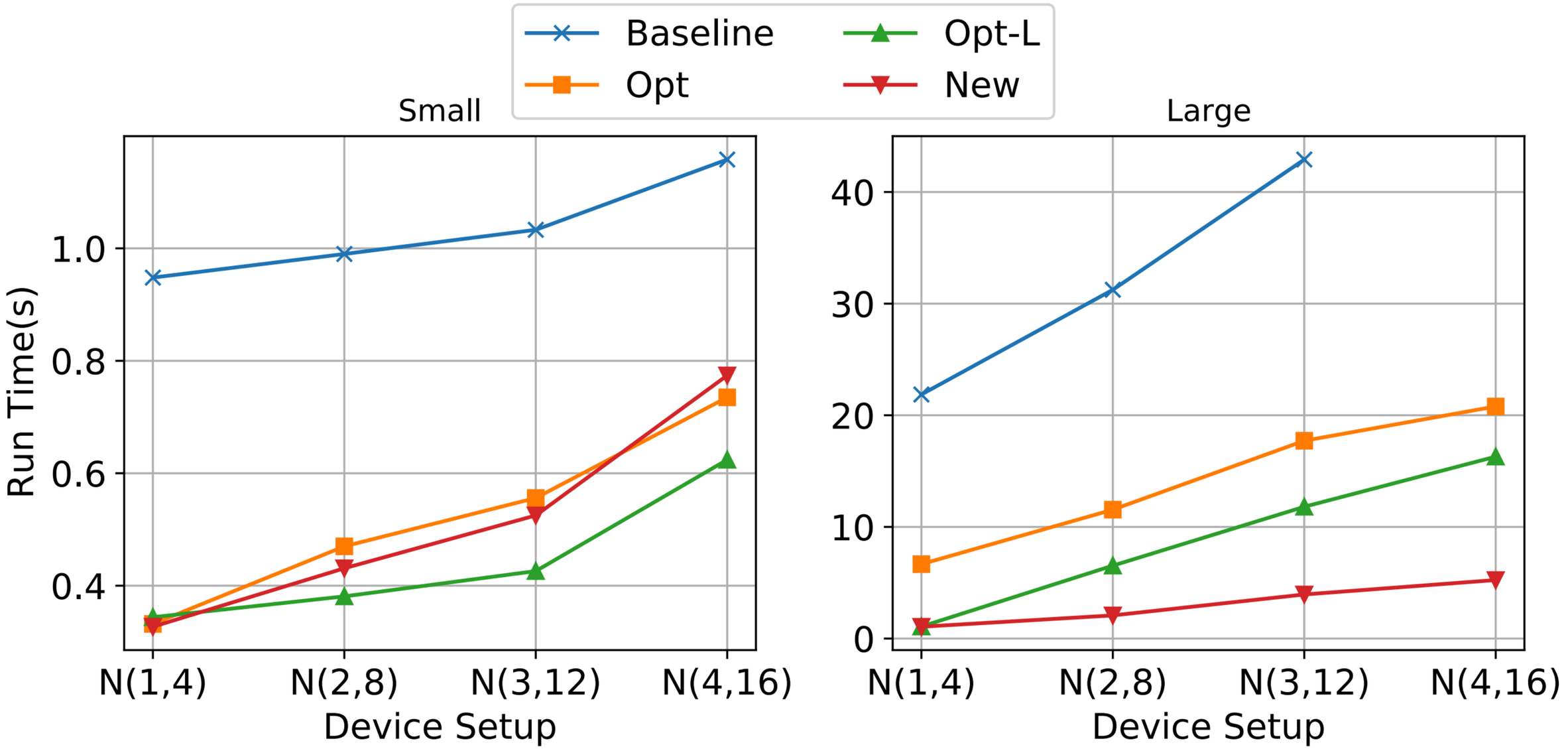
Verringerung der Nachrichtenlatenz:

- **New vs. Baseline:**
  - **~92% (kleine Datenpuffer)**
  - **~97% (große Datenpuffer)**
- **New vs. Opt:**
  - **~73% für kleine Datenpuffer**
  - **Große Datenpuffer:**
    - **To-Mapping: ~70% Verbesserung**
    - **From-Mapping: akzeptable Verbesserung**

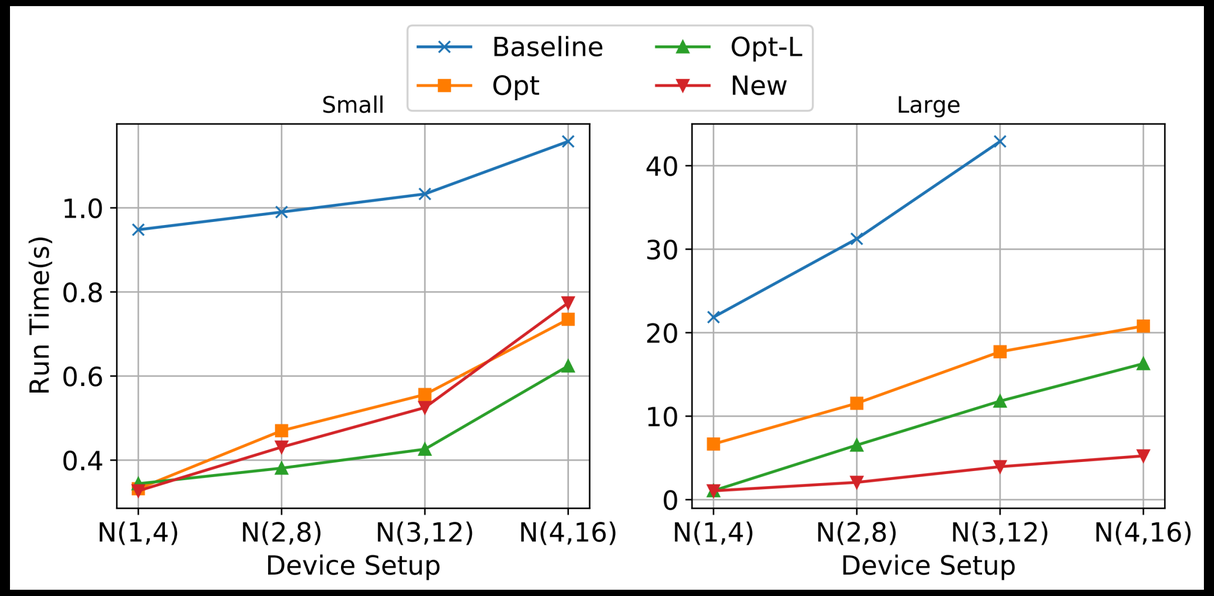
XSBench-Test mit starker Skalierung von Laufzeit ~ Rechenressourcen



XSBench-Test mit schwacher Skalierung von Laufzeit ~ Rechenressourcen



XSBench-Test mit schwacher Skalierung von Laufzeit ~ Rechenressourcen



Verringerung der Laufzeit:

- New vs. Baseline:
  - Baseline stürzt ab bei großen Datenmengen
- New vs. Opt und Opt-L:
  - Kleine Datengröße:
    - begrenzte Verbesserung aufgrund von begrenztem Overhead
    - (Verschlechterung zu Opt-L)
  - Große Datengröße: Verbesserung von ~68%

Zusammenfassung

Zusammenfassung

- Die MPI-basierte Erweiterung zu OpenMP ist ein großer und wichtiger Schritt in der Entwicklung von OpenMP

Zusammenfassung

- Die MPI-basierte Erweiterung zu OpenMP ist ein großer und wichtiger Schritt in der Entwicklung von OpenMP
- => Bessere Leistung

Zusammenfassung

- Die MPI-basierte Erweiterung zu OpenMP ist ein großer und wichtiger Schritt in der Entwicklung von OpenMP
- => Bessere Leistung
- => Einfachheit der Nutzung (Benutzerfreundlichkeit)

Zusammenfassung

- Die MPI-basierte Erweiterung zu OpenMP ist ein großer und wichtiger Schritt in der Entwicklung von OpenMP
- => Bessere Leistung
- => Einfachheit der Nutzung (Benutzerfreundlichkeit)
- => Bessere Skalierung und Portabilität

Literatur

- **„MPI-based Remote OpenMP Offloading: A More Efficient and Easy-to-use Implementation“** *von Baodi Shan, Mauricio Araya-Polo, Abid M. Malik, Barbara Chapman*

- https://hpc-tutorials.llnl.gov/openmp/programming_model/ *(Stand 10.12.2023)*