View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Online Publikationen der Universität Stuttgart

Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Performanzanalyse und Optimierung einer verteilten Multi-Physik Simulationssoftware

Peter Vollmer

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Miriam Mehl
Betreuer/in:	Dipl.-Ing. Florian Lindner
Beginn am:	19. September 2016
Beendet am:	21. März 2017
CR-Nummer:	C.1.4,C.2.4

Kurzfassung

Für die Welt der Multi-Physik-Simulationen wurde die Simulationssoftware PreCICE entwickelt um dem Problem der steigenden Anforderungen und der steigenden Komplexität der Simulationen mit dem Teile-und-Herrsche-Prinzip zu begegnen. Das heißt es lassen sich zwei getrennt entwickelte und getrennt laufende Löser über PreCICE verbinden. Bei der Optimierung der Kommunikation zwischen Lösern sind im Bereich der Kommunikationserstellung Performanz-Probleme festgestellt worden.

Ausgehend vom Verbindungsaufbau zwischen Kopplungspartnern von PreCICE wurden weitergehende Untersuchungen bezüglich möglicher Probleme im Kommunikationsaufbau und der Kommunikation zwischen Prozessen angestellt. Als mögliches Problem wurde der Austausch der Portnamen über das Dateisystem ausgemacht.

Zur Durchführung einfacher Tests wurde die Referenzimplementierung in einem Testprogramm nachgebaut und Ansatz 0 genannt. Als alternative Austauschmöglichkeit wurde in das Testprogramm der Austausch der Portnamen über einen Nameserver als Ansatz 1 implementiert. Ansatz 2 war wegen zu hohem Aufwand nur theoretisch durchdacht worden, er würde den Austausch über Master-Prozesse vorsehen. Mit dem Ansatz 3 wurde der Verbindungsaufbau über nur noch einen Inter-Kommunikator für alle Verbindungen vorgesehen, anstatt für jede benötigte Verbindung einen separaten Inter-Kommunikator zu erstellen. Zum Vergleich der Ansätze wurden Testreihen auf beiden Referenzsystemen durchgeführt. Die Ergebnisse der Tests zeigen, dass anhand der Zahlen keinem der Ansätze ein Vorrang eingeräumt werden kann. Als vielversprechendster Ansatz galt die Ersetzung der einzelnen Inter-Kommunikatoren zwischen Prozessen durch einen einzigen für alle Prozesse (Ansatz 3).

Zu den Herausforderungen in der Zukunft zählt sicher die Entscheidung, ob und wenn ja wie der Ansatz 3 in PreCICE eingebaut werden soll. Dies erfordert dann wiederum weitere Untersuchungen über die Leistungsfähigkeit des Ansatzes. Um eventuelle Performanz-Probleme langfristig beseitigen zu können, sollten die entsprechenden Stellen der Referenzsysteme informiert werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ALPS Application Level Placement Scheduler. 27

CLE Cray Linux Environment. 27

GPFS General Parallel File System. 25

HLRS Höchstleistungsrechenzentrum Stuttgart. 13

LRZ Leibniz-Rechenzentrum. 13

MPI Message-Passing Interface. 15

PreCICE Precise Code Interaction Coupling Environment. 13

SLES Suse Linux Enterprise Server. 25

1. Einleitung

Simulationen stellen eines der zentralen Aufgabengebiete für die Weiterentwicklung von Datensystemen dar. Damit verbunden sind wachsende Anforderungen und zunehmende Problemstellungen beim Handling der Software, die trotz der steigenden Komplexität skalierbar und flexibel sein muss. Im Bereich Multi-Physik- Simulation wurde mit Precise Code Interaction Coupling Environment (PreCICE) eine Multi-Physik-Simulationssoftware entwickelt, mit der es möglich ist, Löser nach dem Teile-und-Herrsche-Prinzip zu verbinden. Das heißt es lassen sich zwei getrennt entwickelte und getrennt laufende Löser über PreCICE verbinden. Bei Weiterentwicklungen zur Optimierung der Kommunikation zwischen den Lösern wurden im Bereich des Kommunikationsaufbaus Performanz-Probleme festgestellt [Shu15].

Ziel dieser Arbeit ist es Untersuchungen durchzuführen, die das Problem eingrenzen und Lösungsansätze aufzeigen. Für die Untersuchungen standen die Supercomputer des Höchstleistungsrechenzentrum Stuttgart (HLRS) der Universität Stuttgart und des Leibniz-Rechenzentrum (LRZ) der Bayrischen Akademie der Wissenschaft in München zur Verfügung.

Nachdem der Problembereich in PreCICE lokalisiert ist (siehe Kapitel 2), wird er zusammen mit Ansätzen, die das Problem lösen könnten, in einem Testprogramm zusammengefasst (siehe Kapitel 3). Um die Ansätze bewerten zu können, werden sie mittels der Ergebnisse des Testprogramms mit der problembehafteten Implementierung verglichen (siehe Kapitel 4). Abschließend wird die Bewertung der Vergleiche Hinweise liefern für die Behandlung der Probleme (siehe Kapitel 5).

2. Hintergrund

In einer Welt, in der Simulationen immer wichtiger werden, werden diese auch immer umfangreicher, was letztendlich auch die Anforderungen an die Rechner erhöt. Hiermit stößt die Steigerung der Leistungsfähigkeit einzelner Prozessoren an die physikalischen Grenzen[1]. Auch die Steigerung der Kernzahl pro Prozessor ist nicht unendlich skalierbar, daher begegnet man diesem Problem in einem Rechenzentrum mit vielen Rechnern, die über Netzwerke verbunden sind. Um nun mit diesen Rechnern effizient komplexe Probleme lösen zu können, sollte die Kommunikation zwischen den verteilt laufenden Programmteilen effizient und einheitlich sein. Dies führte zur Entwicklung des Message-Passing Interface (MPI) (siehe Abschnitt 2.1), welche den Großteil der eingesetzten Programmiersprachen wie Fortran und C/C++ unterstützt. Hiermit lassen sich nun parallele Löser für verschiedene physikalische Komponenten wie Strömungsmechanik, -dynamik, Akustik oder Festkörpermechanik entwickeln, die diese simulieren. Da die reale Welt nicht aus einem isolierten physikalischen Phänomen, sondern einer Kombination verschiedenster Interaktionen besteht, erscheint es sinnvoll, verschiedene Löser zu kombinieren. Daher werden Programme entwickelt, in die diese Löser eingebaut werden. Will man nun verschiedene Löser kombinieren, so ist dies mit erheblichem Aufwand verbunden, vor allem, wenn sie von verschiedenen Entwicklern stammen.

Um dieses Problem zu vereinfachen, wurde PreCICE entwickelt (siehe Abschnitt 2.3). Es ermöglicht Entwicklern ihre Löser getrennt zu entwickeln und nur über eine einheitliche Schnittstelle mit einem anderen Löser verbunden zu verbinden. Nach dem Teile-und-Herrsche-Prinzip lassen sich damit aus vorhandenen Lösern komplexe Multi-Physik Szenarien erstellen.

2.1. MPI

Der MPI Standard [15] spezifiziert eine Schnittstelle, die dazu dienen soll, eine effiziente und einheitlich Kommunikation für parallele und verteilte Programmierung bereit zu stellen. Während sie dabei portabel und sprachunabhängig sein soll, besteht dennoch eine klare Präferenz für Fortran und C. Im Besonderen sollte Software, die MPI nutzt, durch die selbe Aufrufsyntax und Semantik plattformunabhängig sein. Implementierungen sollten in heterogenen Umgebungen benutzt werden können, daher soll sie für Hersteller ohne große Änderungen an deren unterliegenden Kommunikationskonzepten umgesetzt werden können. MPI bietet daher

[1] https://en.wikipedia.org/wiki/Moore's_law

verschiedene Kommunikationskonzepte, unter anderem Punkt-zu-Punkt Kommunikation oder Verbandsoperationen, sowie Definitionen von Datentypen.

Das Grundkonzept für die Parallelisierung in MPI ist das Instanziieren einer Applikation als Gruppe von Prozessen.

2.1.1. Kommunikatoren

Bevor man MPI innerhalb eines solchen Prozesses nutzen kann, werden mit `MPI_Init(...);` die Grundstrukturen initialisiert. Dazu gehört auch die Erstellung der Standard Kommunikatoren `MPI_COMM_WORLD` und `MPI_COMM_SELF`.

Ein Kommunikator ist eine Gruppe von Prozessen, die immer eine eindeutige Adresse besitzen. Dieser Rang ist ein Integer-Wert, der aufsteigend von 0 vergeben wird. Aus Konvention bekommt der Prozess mit dem Rang 0 Zusatzaufgaben, weshalb wir ihn der Einfachheit halber Master-Prozess nennen. Alle anderen Prozesse sind sogenannte Slave-Prozesse.

Den eigenen Rang in einem Kommunikator erfährt jeder Prozess mit `MPI_Comm_rank(...)` vom entsprechenden Kommunikator. Da in `MPI_COMM_SELF` nur der Prozess selbst enthalten ist, ist sein Rang in diesem Kommunikator immer 0.

Die Anzahl der Prozesse in einem Kommunikator erfährt der Prozess durch Ausführen des Befehls `MPI_Comm_size(...)`.

Intra-Kommunikator

`MPI_COMM_WORLD` und `MPI_COMM_SELF` zählen zu den Intra-Kommunikatoren. Diese enthalten immer eine Gruppe an teilnehmenden Prozessen und werden zur Kommunikation innerhalb ihrer Teilnehmer benutzt. Die Prozesse adressieren sich anhand ihres Ranges. `MPI_COMM_WORLD` enthält hierzu alle gemeinsam gestarteten Prozesse, wohingegen `MPI_COMM_SELF` nur den eigenen Prozess enthält.

Inter-Kommunikator

Die andere Art von Kommunikatoren ist die des Inter-Kommunikators. Er verbindet zwei Kopien von nicht überlappenden Intra-Kommunikatoren und deren Gruppen an Prozessen. Die Kommunikation kann dabei immer nur zwischen den Gruppen stattfinden und nicht, wie beim Intra-Kommunikator, innerhalb einer Gruppe. Die Adressierung der entfernten Prozesse findet über deren lokale Ränge der zugrunde liegenden Intra-Kommunikatoren statt.

Um die Größe des entfernten Intra-Kommunikators zu erhalten, lässt sich hier der Befehl `MPI_COMM_remote_size(...)` verwenden.

2.1.2. Domäne

Die Domäne definiert einen Bereich, in dem etwas gilt. Für Portnamen definiert eine Domäne den Eindeutigkeitsbereich und damit die Gültigkeit.

2.1.3. Portname

Ein Portname ist eine eindeutige Adresse eines Prozesses in einer Domäne. Sie kann zum Beispiel aus einem klassischen Socket und damit aus einer IP-Adresse:Port bestehen. Um in MPI einen solchen Port zu erstellen, verwendet man den Befehl `MPI_Open_port(...)`. In Tabelle 2.1 ist ein Beispielaufruf aufgeführt, der einen Portnamen in die Variable *portname* schreibt.

ausführender Prozess	Befehl
Port-Ersteller	`MPI_Open_port(portname)`

Tabelle 2.1.: Beispielaufruf zum Erstellen eines Portnamen in der Variable *portname*

2.1.4. Austausch von Portnamen

Da Portnamen, die in Abschnitt 2.1.3 erstellt wurden, immer beiden Parteien, die ihn nutzen wollen, bekannt sein müssen, müssen sie veröffentlicht und aufgefunden werden. Eine unpraktische Varianten wäre z.B. die händische Übertragung über die Kommandozeile. Ein Austausch über ein gemeinsam erreichbares Dateisystem wäre jedoch praktischer, wenn MPI nicht eine interne Möglichkeit anbieten würde. Diese Möglichkeit ist die Veröffentlichung und das anschließende Auffinden eines Portnamens unter einem wählbaren Service-Namen auf einem Nameserver. Dazu führt der Port-Ersteller den Befehl `MPI_PUBLISH_NAME(...)` aus. Anschließend können alle interessierten Parteien den Portnamen mit `MPI_LOOKUP_NAME(...)` und dem Service-Namen auffinden. Damit ein solches Auffinden gelingt, müssen beide Parteien in der selben Domäne sein. In Tabelle 2.2 sind Beispielaufrufe für die Parteien des Port-Erstellers und Port-Interessierten aufgeführt, die einen Portnamen *portname* unter dem Service-Namen *service_name* veröffentlichen bzw. auffinden wollen. Der Portname *portname* wurde wie in Abschnitt 2.1.3 beschrieben vom Port-Ersteller erstellt.

Um eine einmal getätigte Veröffentlichung zurückzuziehen, führt der Port-Ersteller, in Verbindung mit dem Service-Namen, den Befehl `MPI_UNPUBLISH_NAME)` aus. Anschließend kann der Portnamen im Nameserver nicht mehr aufgefunden werden.

ausführender Prozess	Befehl
Port-Ersteller	`MPI_PUBLISH_NAME(service_name,portname)`
Port-Interessierter	`MPI_LOOKUP_NAME(service_name,portname)`

Tabelle 2.2.: Beispielaufruf für das Veröffentlichen und Auffinden eines Portnamen *portname* unter dem Service-Namen *service_name*.

ausführender Prozess	Befehl
Port-Ersteller	`MPI_UNPUBLISH_NAME(service_name)`

Tabelle 2.3.: Beispielaufruf für den Rückzug einer Veröffentlichung unter dem Service-Namen *service_name*.

2.1.5. Punkt-zu-Punkt Kommunikation

Möchte ein Prozess mit nur einem anderen Prozess kommunizieren, so ist das eine Punkt-zu-Punkt Verbindung. Für diese wird in MPI ein gemeinsamer Kommunikator benötigt und der Rang des jeweils anderen. Für ein blockierendes Senden ruft der sendende Prozess dazu `MPI_Send(...)` auf, damit der empfangende Prozess, blockierend mit dem Aufruf von `MPI_Recv(...)`, die Daten empfangen kann. Eine Punkt-zu-Punkt Verbindung über einen Inter-Kommunikator lässt nur Prozesse aus den verschiedenen, verbundenen Intra-Kommunikatoren miteinander kommunizieren. Bei der Verwendung eines Intra-Kommunikators können die Prozesse untereinander kommunizieren.

In Tabelle 2.4 und Abbildung 2.1 *1.)* sind Beispielaufrufe bei der Verwendung des Intra-Kommunikators *MPI_COMM_WORLD*, einem sendenden Prozess mit Rang 0 und einem empfangenden Prozess mit Rang 1 illustriert. In Tabelle 2.5 und Abbildung 2.1 *2.)* sind Beispielaufrufe bei der Verwendung des Inter-Kommunikators *a_b_inter_comm*, einem sendenden Prozess mit Rang 0 des Intra-Kommunikators *intra_comm_a* und einem empfangenden Prozess mit Rang 1 aufgeführt des Intra-Kommuminikators *intra_comm_b* illustriert.

ausführender Prozess	Befehl
0	`MPI_Send(...,1,MPI_COMM_WORLD)`
1	`MPI_MPI_Recv(...,0,MPI_COMM_WORLD)`

Tabelle 2.4.: Beispielaufruf für das Senden von Daten von Prozess mit Rang 0 an Prozess mit Rang 1 über den Intra-Kommunikator `MPI_COMM_WORLD`.

ausführender Prozess		Befehl
Gruppe A	0	`MPI_Send(...,1,a_b_inter_comm)`
Gruppe B	1	`MPI_Recv(...,0,a_b_inter_comm)`

Tabelle 2.5.: Beispielaufruf für das Senden von Daten von Prozess mit Rang 0 eines Intra-Kommunikators *intra_comm_a* an Prozess mit Rang 1 eines Intra-Kommunikators *intra_comm_b* über den Inter-Kommunikator *a_b_inter_comm*.

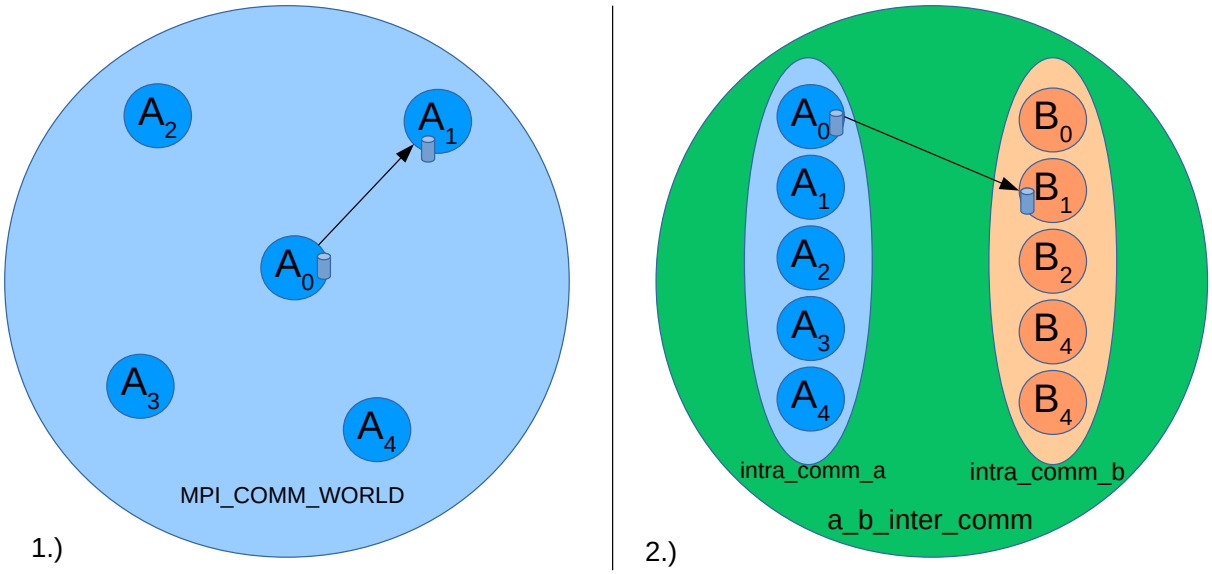

Abbildung 2.1.: 1. Senden von Daten von Prozess mit Rang 0 zu Prozess mit Rang 1 des Intra-Kommunikators *MPI_COMM_WOLRD*. 2.) Senden als Prozess des Intra-Kommunikators *intra_comm_a* zu Prozess mit Rang 1 des Intra-Kommunikators *intra_comm_b* des Inter-Kommunikators *a_b_inter_comm*.

2.1.6. Verbandsoperationen

Verbandsoperationen sind Operationen, bei denen alle Prozesse eines Kommunikators beteiligt sind.

Wichtige Operationen sind dabei Barrieren, die zur Synchronisation verwendet werden, Broadcasts, die zur Verteilung von Daten verwendet werden, Sammeln von verteilten Daten oder Operationen, die zur Erstellung von Inter-Kommunikatoren verwendet werden.

Barriere

Barrieren werden dazu verwendet, alle Prozesse eines Kommunikators an einer bestimmten Stelle zu synchronisieren und sie erst weiter laufen zu lassen, wenn alle an dieser Stelle sind. `MPI_Barrier(...)` ist der von MPI bereitgestellte Befehl, den alle Prozesse eines Kommunikators ausführen müssen. Bei einem Inter-Kommunikator müssen dies alle Prozesse beider Gruppen.

In Tabelle 2.6 sind Beispielaufrufe bei der Verwendung eines beliebigen Kommunikators aufgeführt.

ausführender Prozess	Befehl
alle	`MPI_Barrier, comm)`

Tabelle 2.6.: Beispielaufruf für eine Barriere mit einem beliebigen Kommunikator *comm*.

Broadcast

Broadcasts werden zur Verteilung von Daten eines Prozesses an andere Prozesse genutzt. Je nach Typ des Kommunikators können diese unterschiedlich sein. Bei der Verwendung eines Intra-Kommunikators sind das alle anderen Prozesse des Kommunikators. Verwendet man einen Inter-Kommunikator, dann kann ein Prozess aus Gruppe A, nur Daten an alle Prozesse aus Gruppe B verteilen. Als Befehl kommt bei beiden Kommunikator-Typen `MPI_Bcast(...)` zum Einsatz. Dieser muss von allen Prozessen ausgeführt werden, auch von den vermeintlich nicht beteiligten Prozessen aus Gruppe A des Inter-Kommunikators.

Bei der Verwendung eines Intra-Kommunikators wird der Rang des sendenden Prozesses für alle Prozesse als Adresse verwendet, wohingegen bei der Verwendung eines Inter-Kommunikators nur die Teilnehmer aus Gruppe B diese Adresse verwenden. Alle Prozesse aus Gruppe A müssen Spezial-Ränge verwenden. Der sendende Prozess muss `MPI_ROOT` verwenden und alle anderen Prozesse aus Gruppe A müssen `MPI_PROC_NULL` verwenden.

In Tabelle 2.7 und Abbildung 2.2 *1.)* sind Beispielaufrufe bei der Verwendung des Intra-Kommunikators `MPI_COMM_WOLRD`, dem verteilenden Prozess mit Rang 0 an alle anderen Prozesse illustriert. In Tabelle 2.8 und Abbildung 2.2 *2.)* sind Beispielaufrufe bei der Verwendung des Inter-Kommunikators *a_b_inter_comm*, dem verteilenden Prozess mit Rang 0 eines Intra-Kommunikators *intra_comm_a*, alle Prozesse eines Intra-Kommunikators *intra_comm_b* illustriert.

ausführender Prozess	Befehl
alle	`MPI_Bcast(...,0,MPI_COMM_WORLD)`

Tabelle 2.7.: Beispielaufruf für Broadcast von Prozess mit Rang 0 über den Intra-Kommunikator `MPI_COMM_WORLD` an alle anderen Prozesse des Kommunikators.

ausführender Prozess		Befehl
intra_comm_a	0	`MPI_Bcast(...,MPI_ROOT,a_b_inter_comm)`
	1 bis N-1	`MPI_Bcast(...,MPI_PROC_NULL,a_b_inter_comm)`
intra_comm_b	alle	`MPI_Bcast(...,0,a_b_inter_comm)`

Tabelle 2.8.: Beispielaufruf für Broadcast von Prozess mit Rang 0 eines Intra-Kommunikators *intra_comm_a*, über den verbindenden Inter-Kommunikator `a_b_inter_comm` an alle Prozesse eines Intra-Kommunikators *intra_comm_b*.

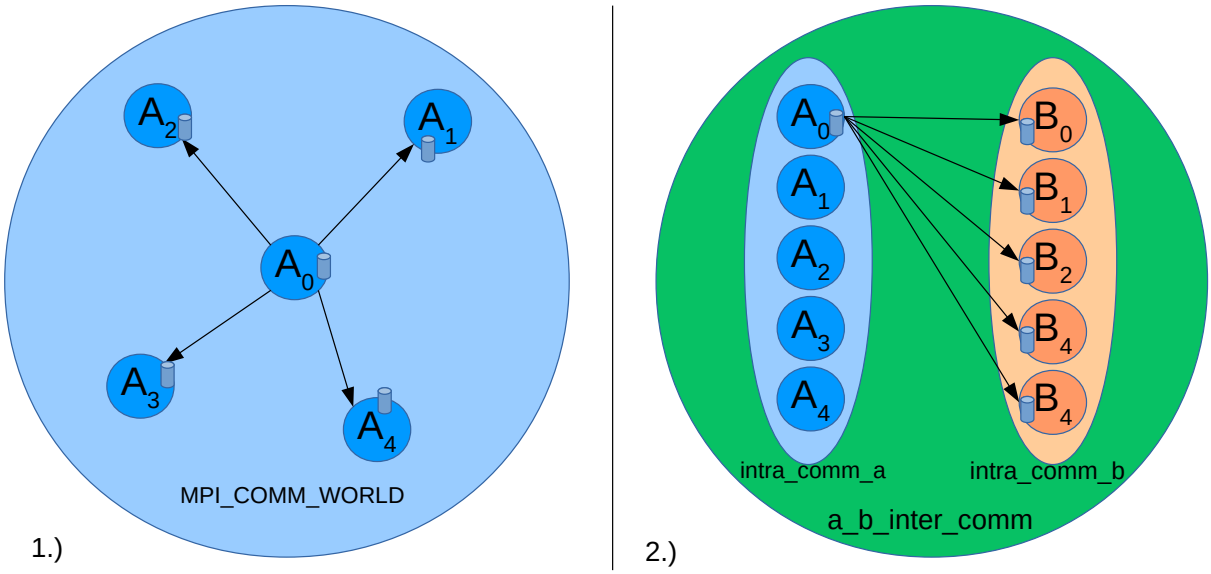

Abbildung 2.2.: 1. Broadcast von Prozess mit Rang 0 zu allen anderen Prozessen des Intra-Kommuniktors *MPI_COMM_WOLRD*. 2.) Broadcast als Prozess eines Intra-Kommunikators *intra_comm_a* zu allen Prozessen eines Intra-Kommunikators *intra_comm_b* des Inter-Kommunikators *a_b_inter_comm*.

Sammeln

Zum Sammeln von Daten auf einem Prozess, wie die verteilten Einträge eines Vektors, wird der Befehl `MPI_Gather(...)` eingesetzt. Das Verhalten der Kommunikatoren und der Adressierung ist gleich wie bei Abschnitt 2.1.6. Der Unterschied liegt in der Flussrichtung der Daten und dass der Empfangende keine eigene Daten besitzt.

In Tabelle 2.9 sind Beispielaufrufe zum Sammeln von Daten in Prozess mit Rang 0 des Intra-Kommunikators *MPI_COMM_WORLD* aufgeführt und in Abbildung 2.3 *1.)* illustriert. In Tabelle 2.10 und Abbildung 2.3 *2.)*sind Beispielaufrufe zum Sammeln von Daten in Prozess mit Rang 0 eines Intra-Kommunikators *intra_comm_a* von allen Prozessen eines Intra-Kommunikators *intra_comm_b*, die über den Inter-Kommunikator *a_b_inter_comm* verbunden sind, aufgeführt illustriert.

ausführender Prozess	Befehl
alle	`MPI_Gather(...,0,MPI_COMM_WORLD)`

Tabelle 2.9.: Beispielaufruf für das Sammeln auf Prozess mit Rang 0 über Intra_Kommunikator `MPI_COMM_WORLD`

ausführender Prozess		Befehl
Gruppe A	0	`MPI_Gather(...,MPI_ROOT,a_b_inter_comm)`
	1 bis N-1	`MPI_Gather(...,MPI_PROC_NULL,a_b_inter_comm)`
Gruppe B	alle	`MPI_Gather(...,0,a_b_inter_comm)`

Tabelle 2.10.: Beispielaufruf für das Sammeln von Daten auf Prozess mit Rang 0 eines Intra-Kommomunikators *intra_comm_a* über den Inter-Kommunikator *a_b_inter_comm* von allen Prozessen eines Intra-Kommunikators *intra_comm_b*.

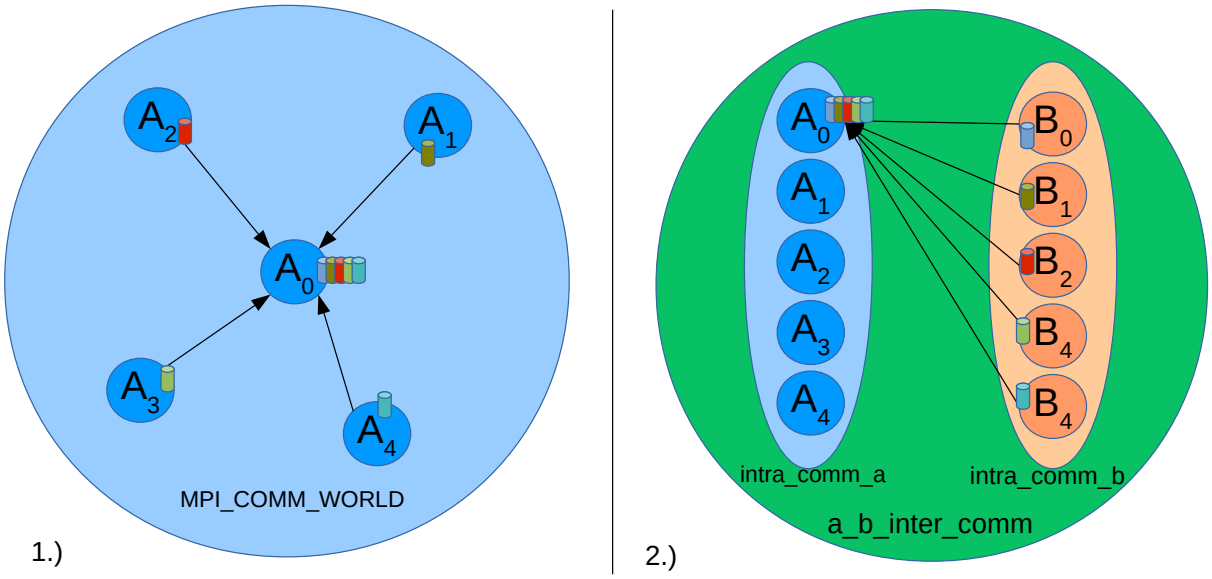

Abbildung 2.3.: 1. Sammeln von Daten auf Prozess mit Rang 0 von allen anderen Prozessen des Intra-Kommunikators *MPI_COMM_WOLRD*. 2.) Sammeln als Prozess eines Intra-Kommunikators *intra_comm_a* von allen Prozessen eines Intra-Kommunikators *intra_comm_b* des Inter-Kommunikators *a_b_inter_comm*.

Inter-Kommunikatoren erstellen

Wenn zwei nicht überlappende Gruppen von Prozessen A und B und ihre Intra-Kommunikatoren verbinden werden sollen, so übernimmt eine Gruppe den Server-Teil und eine andere Gruppe den Client-Teil. Der Server-Teil besteht daraus, dass der Master-Prozess der Gruppe einen Portnamen erstellt, diesen mit einer geeigneten Methode veröffentlicht und sich anschließend mit `MPI_Comm_accept(...)` verfügbar macht. Die alleinige Verwendung des Portnamens auf dem Master-Prozess genügt, da alle beteiligten Prozesse der Server-Gruppe den dazugehörigen Intra-Kommunikator und den Rang des Prozesses, der den Portnamen kennt, angeben. Die Slave-Prozesse benötigen für die Signatur des Befehls nur einen beliebigen Platzhalter. Der Client-Teil funktioniert nach dem gleichen Prinzip, der Master-Prozess der Gruppe benötigt den zuvor von dem Server-Teil veröffentlichten Portnamen und verbindet alle mit `MPI_Comm_connect(...)` zur Server-Gruppe.

Um Intra-Kommunikatoren, die z.B. aus unterschiedlich gestarteten MPI-Programmen stammen, verbinden zu können, kommt es darauf an, dass sich die Domäne der Portnamen über diese beiden Programme erstreckt. Nur wenn die selbe Domäne gilt, können sie sich miteinander verbinden, ansonsten nicht.

In Tabelle 2.11 sind Beispielaufrufe für eine Gruppe A mit ihrem Intra-Kommunikator *intra_comm_a* als Server-Teil und eine Gruppe B mit ihrem Intra-Kommunikator *intra_comm_b*

als Client-Teil aufgeführt. Der Portname *portname* wurde vorher schon erstellt, wie in Abschnitt 2.1.3 beschrieben, und anschließend, mit einer Varianten, in Abschnitt 2.1.4 beschrieben, ausgetauscht.

ausführender Prozess		Befehl
Gruppe A	0	`MPI_Comm_accept(portname,...,0,intra_comm_a)`
	1 bis N-1	`MPI_Comm_accept(empty_portname,...,0,intra_comm_a)`
Gruppe B	0	`MPI_Comm_connect(portname,...,0,intra_comm_b)`
	1 bis N-1	`MPI_Comm_connect(empty_portname,...,0,intra_comm_b)`

Tabelle 2.11.: Beispielaufruf für das Erstellen eines Inter-Kommunikators zwischen den Intra-Kommunikatoren *intra_comm_a* und *intra_comm_b* mit jeweils den Portnamen *portname* auf den Master-Prozessen und einen Platzhalter-Portnamen *empty_portname* auf den Slaves

2.1.7. Implementierungen

Zu den bekanntesten Implementierungen gehören OpenMPI [GFB+04] und MPICH[2] und beide unterstützen den MPI Standard 3.1 vollständig. Weil beide Open Source und kommerzialisierbar sind, werden sie gerne auch als Basis für kommerzielle Produkte verwendet.

2.2. Rechenzentren

Um nun massiv parallele Löser ausführen zu können, werden eine große Anzahl von Computern, den so genannten Knoten, benötigt, welche effizient vernetzt sein müssen. Für diese Arbeit standen uns im LRZ der Bayrischen Akademie der Wissenschaften in München der Rechner mit dem Namen SuperMUC und im HLRS der Rechner mit dem Namen Hazel Hen zur Verfügung und dienen dieser Arbeit als Referenzsysteme. Beide Rechner zählen zu den aktuellsten Supercomputern und finden sich auf der aktuellen Liste (November 2016) Top 500[3] der schnellsten Rechner. In diesem Ranking belegt Hazel Hen aktuell Platz 14 und SuperMuc Platz 36. Die Zahlen, die für dieses Ranking verwendet wurden, beziehen sich immer auf die Verwendung des gesamten Supercomputers. Um die Auslastung zu erhöhen, stehen im Normalbetrieb die Rechner nur mit einer geringeren Maximalzahl an Rechenknoten zur Verfügung.

[2] http://www.mpich.org
[3] https://www.top500.org/lists/2016/11/

2.2.1. SuperMUC

Der Name SuperMUC[4] steht für drei Supercomputer. Sie sind als Einzelrechner durch wandelnde Anforderungen und Anpassung an den Stand der Technik entstanden. Diese Arbeit bezieht sich immer auf die Thin Nodes, einem *IBM System x iDataPlex dx360M4*[5] System, das im Jahre 2012 gebaut wurde und die meisten Knoten zur Verfügung stellen kann.

Hardware

Die hierarchische Struktur des SuperMUCs besteht aus 18 Inseln, die jeweils 512 Knoten beherbergen. Jeder Knoten enthält 2 Sandy Bridge-EP Xeon E5-2680 8C[6] Prozessoren mit jeweils 8 Kernen, womit das Gesamtsystem auf 147.456 Kerne kommt.

Als Netzwerktechnologie zur Vernetzung aller Systemteile wird Infiniband FDR10 eingesetzt. Als Hochleistungs-Netzwerkspeicher wird das parallele Dateisystem IBM General Parallel File System (GPFS)[7] von IBM eingesetzt.

Software

Suse Linux Enterprise Server (SLES) kommt als Betriebssystem auf dem SuperMUC zum Einsatz.

Zur Softwareumgebungsverwaltung kommt modules[8] zum Einsatz. Modules ermöglicht es Systembetreuern, Module vorzukonfigurieren, welche Benutzer nur laden müssen, um einen bestimmten Compiler, eine bestimmte Bibliothek oder andere Software nutzen zu können, ohne viele Konfigurationen machen zu müssen.

Zur Aufgabenverwaltung liefert IBM den IBM Loadleveler[9] und ermöglicht die Steuerung der Aufgaben. Für eine hohe Auslastung verwaltet die Aufgabenverwaltung drei Warteschlangen[10]: *test*, *general* und *large*. Der Benutzer teilt seine Aufgaben selbständig anhand von Parametern der entsprechenden Warteschlange zu. Die Warteschlange *test* ist für kleine kurze Aufgaben vorgesehen, die auf maximal 30 Minuten und 32 Knoten beschränkt ist. Außerdem kann ein Nutzer immer nur eine Aufgabe gleichzeitig auf *test* ausführen lassen. Durch eine höhere

[4] https://www.lrz.de/services/compute/supermuc/systemdescription/
[5] https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=AN&subtype=CA&htmlfid=897/ENUS113-005
[6] http://ark.intel.com/products/64583/Intel-Xeon-Processor-E5-2680-(20M-Cache-2_70-GHz-8_00-GTs-Intel-QPI)
[7] http://www-03.ibm.com/systems/software/gpfs/
[8] http://modules.sourceforge.net/
[9] https://www.ibm.com/support/knowledgecenter/SSFJTW/loadl_welcome.html
[10] https://www.lrz.de/services/compute/supermuc/loadleveler/

Priorität in der Rechenzeitvergabe eignet sich *test* zum Entwickeln und Testen. Werden längere Rechenzeiten oder mehr Knoten benötigt, verwendet man *general* für mittlere Aufgabengrößen oder *large* für große Aufgaben. Für beide Warteschlangen gilt ein Zeitlimit von 48 Stunden und es können maximal 8 Aufgaben parallel laufen. Das Knotenlimit liegt für *general* zwischen 33 und 512 Knoten und für *large* bei 513 bis 4.096. 4.096 Knoten bzw. 65.536 Kerne sind auf dem SuperMUC, wie im Abschnitt 2.2 erwähnten Normalfall, die maximale Anzahl an verfügbaren Kernen. Für eventuell verfügbare Sonderregelungen für Aufgaben außerhalb der Spezifikation sollte Kontakt mit dem LRZ aufgenommen werden.

Unter den verfügbaren MPI Implementierungen wurde das bis dahin aktuellste und für den SuperMUC angepasste Release von Intels MPI Library Version 5.1 verwendet. Es basiert, wie in Abschnitt 2.1.7 erwähnt, auf MPICH, unterstützt allerdings nicht den gesamten MPI Standard 3.1. Die fehlende Funktionalität[11] ist jedoch nicht relevant. Fehlende Funktionalität lässt sich nur im Vergleich zwischen MPI Standard[15] und einer Referenzkarte[12] der verfügbaren Funktionalität herausfinden. Bei der Compiler-Wahl wurde das aktuellste verfügbare Release des Intel C++ Compilers mit der Version 17.0 gewählt.

2.2.2. Hazel Hen

Hazel Hen ist ein Cray Cascade XC40 Supercomputer[13] und wurde 2015 in Betrieb genommen.

Hardware

Die hierarchische Struktur[14] des Hazel Hen besteht aus 41 Schränken[15], wobei zwei zu einer Schrankgruppe zusammengefasst werden. Jeder Schrank[16] enthält 3 Gruppen zu je 16 Blades in einem Chassis. In jedem Blade befinden sich 4 Knoten, wobei jeder 2 Intel® Xeon® CPU E5-2680 v3 (30M Cache, 2.50 GHz)[17] Prozessoren mit jeweils 12 Kernen enthält. Insgesamt sind es 7.712 Knoten und damit 185.088 Kerne.

Als Netzwerktechnologie zur Vernetzung aller Systemteile wird ein Cray® XC™ Series Network eingesetzt.

[11] https://www.lrz.de/services/software/parallel/mpi/#Doc
[12] https://www.lrz.de/services/software/parallel/mpi/functions.ps
[13] http://www.cray.com/Products/Computing/XC.aspx
[14] https://wickie.hlrs.de/platforms/index.php/Communication_on_Cray_XC40_Aries_network#XC40_design
[15] http://www.hlrs.de/systems/cray-xc40-hazel-hen/
[16] https://wickie.hlrs.de/platforms/index.php/Communication_on_Cray_XC40_Aries_network
[17] https://ark.intel.com/products/81908/Intel-Xeon-Processor-E5-2680-v3-30M-Cache-2_50-GHz

Als Hochleistungs-Netzwerkspeicher wird das Cray® Sonexion® Scale-out Lustre® Storage System[18] in der Version 1600 eingesetzt.

Software

Als Betriebssystem kommt auf dem Hazel Hen Cray Linux Environment (CLE) Version 5.2[19] zum Einsatz, welches auf SLES 11 SP3 basiert.

Zur Softwareumgebungsverwaltung kommt dieselbe Sofware zum Einsatz wie beim SuperMUC (siehe Abschnitt 2.2.1).

Zur Aufgabenverwaltung kommt ein Application Level Placement Scheduler (ALPS) System zum Einsatz, welches die Steuerung der Aufgaben ermöglicht. Für eine hohe Auslastung verwaltet die Aufgabenverwaltung 4 Warteschlangen *test*[20], *single*, *multi* und *small*[21].

Die Warteschlange *test* ist die einzige Warteschlange, der Aufgaben explizit zugeordnet werden müssen, für alle anderen entscheidet dies das System anhand anderer Parameter. Die Warteschlange *test* ist für kleine kurze Aufgaben vorgesehen, die auf maximal 25 Minuten und 384 Knoten beschränkt sind. Ein Nutzer kann immer nur eine Aufgabe parallel ausführen lassen. Durch eine höhere Priorität in der Rechenzeitvergabe eignet sie sich zum Entwickeln und Testen. Für längere Rechenzeiten bis zu 24 Stunden und vom HLRS festgelegte Gruppen/Nutzer-Aufgabenlimits sollte man *single*, *multi* und *small* nutzen.

Das einzige Zuordnungsmerkmal, das *single*, *multi* und *small* unterscheidet, ist die Knotenanzahl. Aufgaben, die nur einen Knoten verwenden, werden *single*, ab 2 bis 48 Knoten *small* und alle darüber bis 4096 Knoten *multi* zugeordnet. 4096 Knoten bzw. 98.304 Kerne sind auf dem Hazel Hen, wie im Abschnitt 2.2 erwähnten Normalfall, die maximale Anzahl an verfügbaren Kernen. Für eventuell verfügbare Sonderregelungen für Aufgaben außerhalb der Spezifikation sollte Kontakt mit dem HLRS aufgenommen werden.

Bei der Wahl der MPI Implementierung wurde zunächst die aktuellste Version 7.5.3 des Cray MPICH gewählt, musste später allerdings ersetzt werden, da wichtige Funktionalitäten fehlten. Zu diesen fehlenden Funktionalitäten[22] gehören `MPI_OPEN_PORT(...)`, `MPI_COMM_ACCEPT(...)`, `MPI_COMM_CONNECT(...)`, `MPI_LOOKUP_NAME(...)`, `MPI_PUBLISH_NAME(...)`, `MPI_UNPUBLISH_NAME(...)` und die Verwendbarkeit eines Nameservers. Nach der Kontaktaufnahme mit dem HLRS wurde uns eine Beta Implementierung des Cray MPICH in der Version 7.5.0-pre zur Verfügung gestellt, die zumindest erlaubt, `MPI_OPEN_PORT()`,

[18] http://www.cray.com/products/storage/sonexion
[19] http://docs.cray.com/books/S-2425-52xx/
[20] https://wickie.hlrs.de/platforms/index.php/CRAY_XC40_Using_the_Batch_System#Test_jobs
[21] https://wickie.hlrs.de/platforms/index.php/CRAY_XC40_Batch_System_Layout_and_Limits
[22] http://docs.cray.com/cgi-bin/craydoc.cgi?mode=View;id=S-2529-116;right=/books/S-2529-116/html-S-2529-116//chapter-ck3x6qu8-brbethke.html%23section-9gqpmmm-brbethke

`MPI_COMM_ACCEPT(...)` und `MPI_COMM_CONNECT(...)` zu verwenden. Die restliche Funktionalität ist nicht unbedingt notwendig, da Alternativen zum Austausch von Portnamen existieren.

Als Compiler wurde ein gcc in der Version 6.3.0 gewählt.

2.3. PreCICE

PreCICE ist eine Kopplungsbibliothek, die ursprünglich von Bernhard Gatzhammer [Gat14] entwickelt wurde und an der Universität Stuttgart und der Technischen Universität München weiterentwickelt wird [BLG+16], mit dem Ziel Multi-Physik-Simulation flexibler und skalierbarer zu machen. Außerdem sollte die Komplexität der Kopplung verschiedener physikalischer Löser nach dem Prinzip Teilen und Herrschen, reduziert werden. Dies wurde erreicht, indem die Kommunikation zwischen den Lösern in einer Schnittstelle abgegrenzt und somit dem Entwickler abgenommen wurde. Die Entwicklung als Bibliothek ermöglichte es, eine hochsprachliche und generische Schnittstelle zu entwickeln, die leicht in bestehende und neue Lösern einzubauen ist.

Des Weiteren wird PreCICE unter der Open-Source Lizenz LGPL3 entwickelt und ist als Code auf github[23] verfügbar. Durch die Wahl dieser Lizenz kann die Schnittstelle auch in Closed-Source Lösern verwendet werden.

Mit PreCICE lassen sich somit unterschiedliche Löser, wie z.B. einen strömungs-mechanischen Löser, mit einem struktur-mechanischen Löser kombinieren und so eine Simulation von z.B. Strömung um flexible/verformbare Gegenstände erhalten.

Um eine solche Kopplung möglich zu machen, bietet PreCICE Methoden zur Gleichungskopplung und Interpolation um Datenpunkte in nicht-passenden Netzen abzubilden. Als Interpolations-Methoden stehen Nearest Neighbor, Nearest Projection, Radial Basis Functions [BW01] zur Verfügung. Außerdem bietet PreCICE die Möglichkeit die Kommunikation zwischen zwei ausführbaren Programmen bereitzustellen.

Als erste Struktur wurde eine zentralisierte Struktur gewählt, siehe Abbildung 2.4a, die die Kommunikation zwischen den beiden Lösern, Teilnehmer A und B, steuert. Dazu zählt auch die interne Kommunikation eines Lösers.

Durch Weiterentwicklungen von Benjamin Ueckermann [Uek16] und Alexander Shukaev [Shu15] wurde es möglich, eine parallele Peer-to-Peer Struktur (siehe Abbildung 2.4b) zu wählen und damit den Flaschenhals der zentralen Steuer- und Kommunikationseinheit zu umgehen.

[23] https://github.com/precice/precice

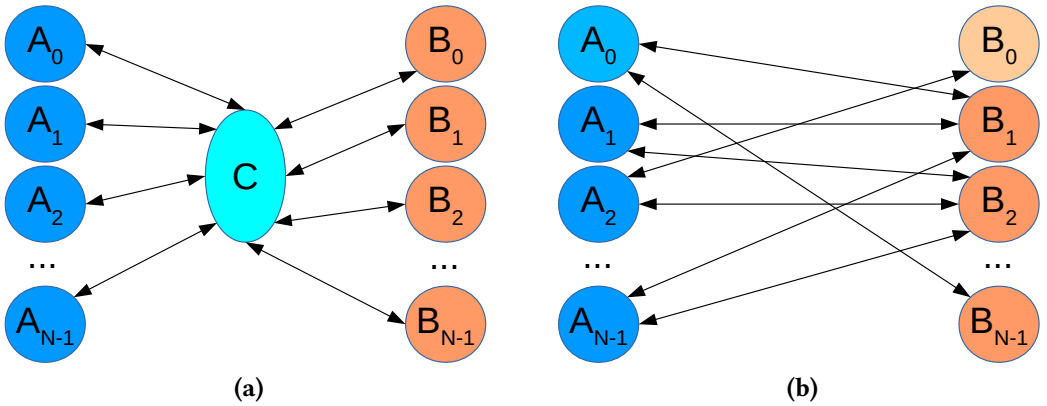

Abbildung 2.4.: (a) alter Kopplungsansatz mit zentraler Steuer und Kommunikationseinheit. (b) neuer Kopplungsansatz mit paralleler Peer-to-Peer Struktur zur Steuerung und Kommunikation [Uek16]

Um dies zu erreichen, wurde die Steuerung der Prozesse in die Master-Prozesse der Kopplungs-Partner integriert und die Haltung der Kopplungsdaten von der zentralen Einheit (siehe Abbildung 2.5) in die Prozesse verschoben (siehe Abbildung 2.6).

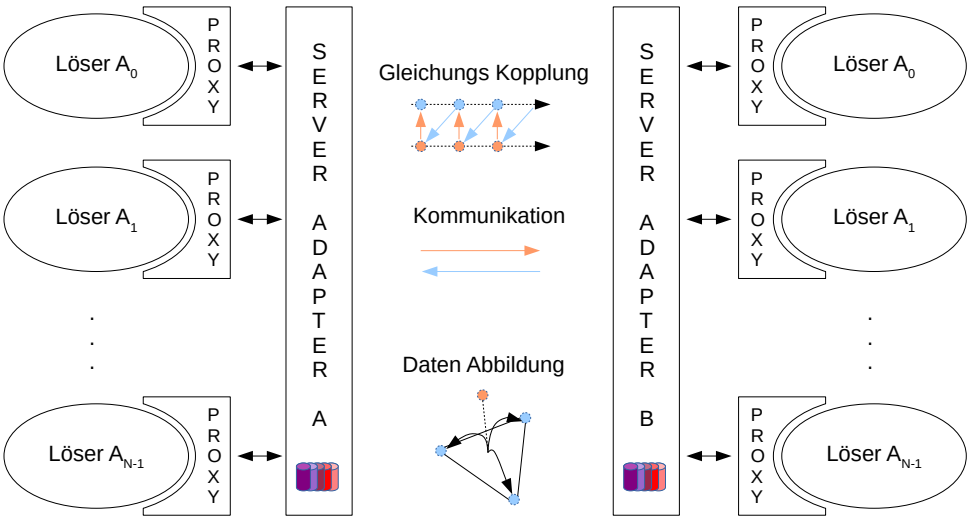

Abbildung 2.5.: alter Kopplungsansatz mit zentraler Steuer- und Kommunikationseinheit [Gat14]

Die Master-Prozesse steuern lediglich den Informationsfluss der Kopplungsdaten, die eigentliche Kopplung führen anschließend die Prozesse selbständig durch.

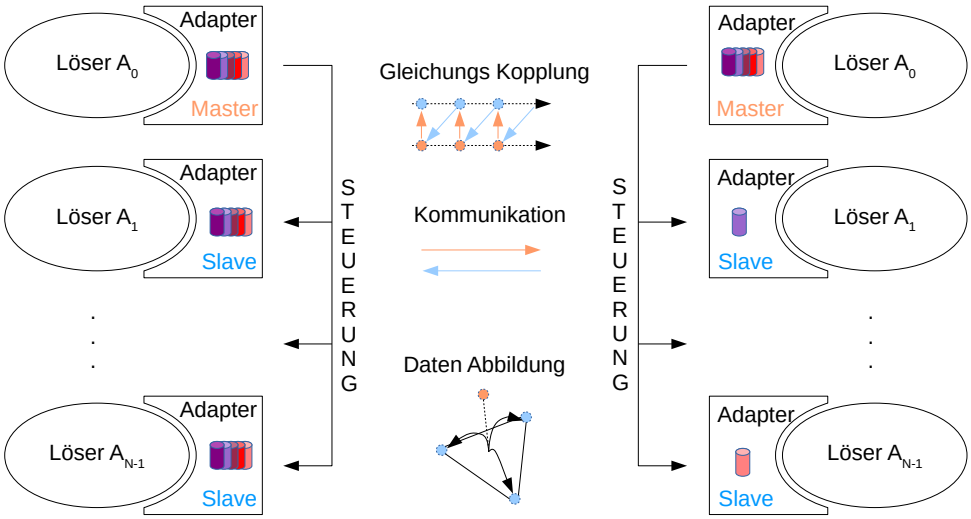

Abbildung 2.6.: Kopplungsansatz mit paralleler Peer-to-Peer Struktur zur Steuerung [Uek16]

2.3.1. Kopplungsdatenaustausch

Während des Kopplungsaustauschs werden Filterungen anhand der gewählten Interpolations-Methode ausgeführt. PreCICE bietet hierzu zwei Filter-Varianten[Uek16] *broadcast/filter* und *pre-filter/post-filter*. Da diese Arbeit sich auf eine Kommunikation mit MPI konzentriert, wird auch nur die dafür empfohlene Variante *broadcast/filter* im Folgenden näher ausgeführt.

Um einen Kopplungsdatenaustausch zwischen Kopplungspartner A und B durchzuführen, werden folgende Schritte durchgeführt:

1. Master-Prozess A sammelt alle Teilnetze seiner Slave-Prozesse ein und baut ein Gesamtnetz
2. Master-Prozess A überträgt Gesamtnetz an Master-Prozess B
3. Master-Prozess B verteilt Gesamtnetz an alle seine Slave-Prozesse, die die Filterung anhand der gewählten Interpolations-Methode durchführen. Dies hat zum Ergebnis eine Kommunikationszuordnung, die alle Kommunikationsbeziehungen enthält, an denen der Jeweilige beteiligt ist.
4. Master-Prozess B sammelt Kommuniktionskarten ein und erstellt eine Gesamt-Kommunikationszuordnung.
5. Master-Prozess B überträgt Gesamt-Kommunikationszuordnung an Master-Prozess A.
6. Master-Prozess A verteilt die Gesamt-Kommunikationszuordnung an seine Slave-Prozesse

Der gesamte Vorgang wird in Abbildung 2.7 grafisch dargestellt. Ab diesem Zeitpunkt wissen alle Prozesse mit welchen anderen Prozessen sie kommunizieren müssen. Anschließend folgt der Verbindungsaufbau.

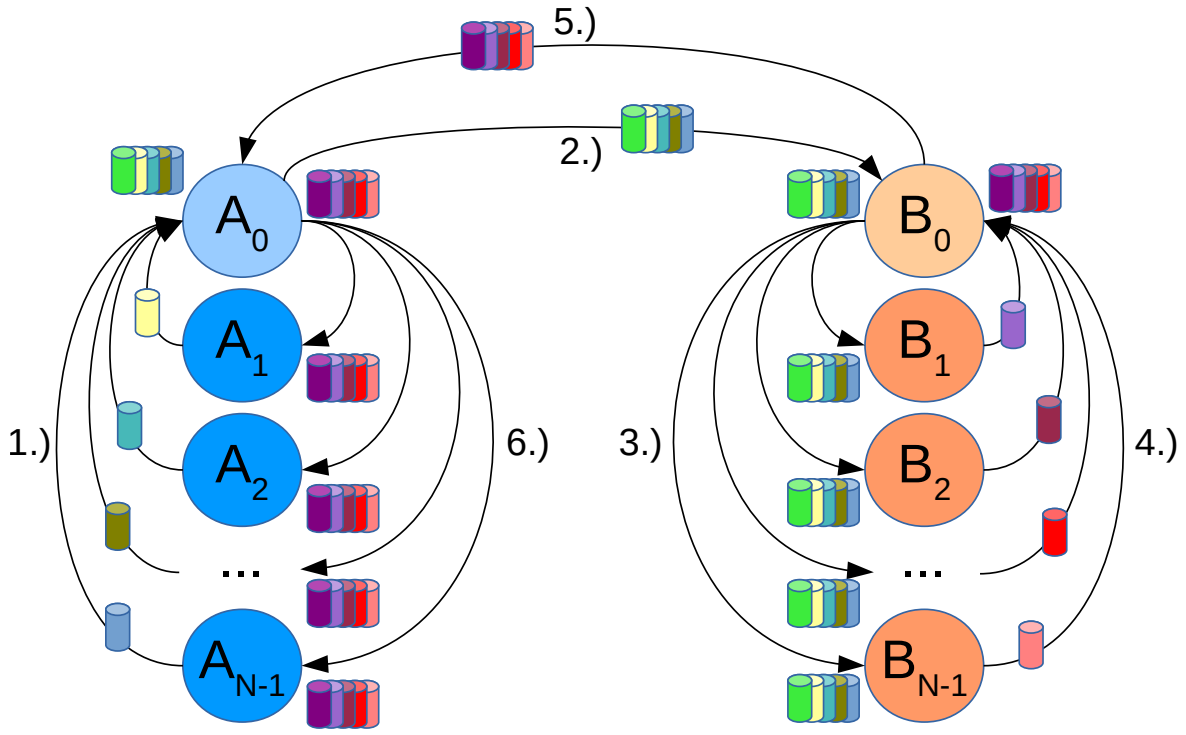

Abbildung 2.7.: Austausch der Kopplungsinformationen zweier Kopplungspartner A und B. 1.) Sammeln von Netzdaten auf Master-Prozess von A, 2.) Gesamtnetz übertragen an Master-Prozess von B, 3.) Verteilung der Daten auf B, 4.) Sammlung der Rückmeldungen auf Master-Prozess, 5.) Rückmeldung von Master B auf Master A, 6.) Verteilen der Rückmeldung an A

Diesen Zustand setzt diese Arbeit als Ausgangspunkt. Die nähere Untersuchung des Verbindungsaufbaus wird benötigt, da bei ersten groben Untersuchungen in [Uek16] und [Shu15] Performanz-Probleme sichtbar wurden.

2.3.2. Verbindungsaufbau

Sind alle Kopplungsdaten vorhanden, wird mit dem Verbindungsaufbau begonnen. Durch den Schwerpunkt auf MPI wird für jede benötigte Verbindung ein separater Inter-Kommunikator erstellt, siehe Abschnitt 2.1.6. Die Inter-Kommunikatoren werden ausgehend vom Intra-Kommunkator `MPI_COMM_SELF` erstellt.

Server

Die grundsätzliche Vorgehensweise der Server-Seite besteht in der Durchführung folgender Schritte in jedem Prozess:

1. Portnamen erstellen (siehe Abschnitt 2.1.3)
2. Veröffentlichung des erstellten Portnamen über das Dateisystem in eine für jeden Portnamen separate Datei mit eindeutigem Namen
3. Für jeden Eintrag in der Kommunikationszuordnung, an dem der Prozess beteiligt ist, einen Inter-Kommunikator erstellen (siehe Abschnitt 2.1.6).
4. Die Veröffentlichung des Portnamen zurückziehen (siehe Abschnitt 2.1.4)

Um mit dem Verbindungsaufbau der Servers-Seite mit dem Kopplungspartner A beginnen zu können, muss der Master-Prozess zunächst einen Workaround für die Verwendung auf dem SuperMUC ausführen. Der Workaround ist notwendig, da das GPFS des SuperMUCs problematisches Verhalten aufweist, wenn viele Dateien in einem Ordner sind. Dies würde passieren, wenn alle Prozesse ihre Portnamen in einem Ordner veröffentlichen. Deshalb erstellt der Master-Prozess anhand der Kommunikationszuordnung für jeden Prozess einen eigenen Ordner für die Portnamen-Datei. Der Ordnername ist derselbe eindeutige Namen wie die Portnamen-Datei und enthält den Namen des Server-Kopplungspartners, den Namen des Client-Kopplungspartners und den Rang des Prozesses, der den Port erstellt. Für die Kopplungspartner-Namen A und B und den Rang 4 lautet Ordner/Datei-Name ".A_B-4.address". Damit die Slave-Prozesse zur gleichen Zeit nicht versuchen, in einen nicht existierenden Ordner zu schreiben, muss sich der Kopplungspartner A synchronisieren und wie im Fall von MPI eine Barriere einsetzen, siehe Abschnitt 2.1.6. Im Anschluss kann der eigentliche Verbindungsaufbau gestartet werden. In PreCICE wird dies zusammengefasst unter einer Operation *Accept Connection*. Alle Prozesse, die Verbindungen erwarten, erstellen einen Portnamen und veröffentlichen ihn auf dem Dateisystem in einer Datei in dem entsprechenden Ordner. Als nächsten Schritt erstellen die Prozesse so viele einzelne Inter-Kommunikatoren, wie es in der Kommunikationszuordnung Kommunikationspartner gibt. Nach der Erstellung aller Inter-Kommunikatoren, ist sicher, dass die Veröffenlichungung der Portnamen rückgängig gemacht werden kann. Dies geschieht durch einfaches Löschen der Datei bzw. des Ordners. Im Anschluss an die

Operation *Accept Connection* muss eine Zuordnung, vom lokalen Rang des Client-Prozess auf den zugehörigen Inter-Kommunikator, stattfinden. Dazu empfängt der Server-Prozess über jeden erstellten Inter-Kommunikator den lokalen Rang des Client-Prozess.

Client

Das grundsätzliche Vorgehen auf Client-Seite besteht darin, dass jeder Prozess für jeden Eintrag in der Kommunikationszuordnung, an dem er beteiligt ist, folgende Schritte durchführt:

- Passenden Portnamen auf dem Dateisystem finden.
- Einen Inter-Kommunikator erstellen (siehe Abschnitt 2.1.6)

Um den Verbindungsaufbau zu starten, muss zuerst, nach dem in Abschnitt 2.3.2 erwähnten Namensmuster, der Dateiname ermittelt werden. Dazu werden die Namen der Kopplungspartner und der Rang des Server-Prozesses, zu dem man einen Inter-Kommunikator erstellen möchte, benötigt. Die Namen der Kopplungspartner sind bekannt und der Rang des Server-Prozesses steht im aktuellen verwendeten Eintrag der Kommunikationszuordnung. Durch den Workaround muss der Suchpfad auch noch um einen Ordner mit dem selben Namen ergänzt werden. Mit diesen Informationen lässt sich die Datei mit dem Portnamen auffinden bzw. auf deren Erscheinen warten. Anschließend wird der Portnamen aus der Datei eingelesen und damit ein Inter-Kommunikator erstellt. Auf der Client-Seite sind die Kommunikatoren direkt einem Server-Prozess zugeordnet, daher müssen keine zusätzlichen Vorkehrungen gegen Verklemmung getroffen werden. Damit der Prozess auf der Server-Seite entscheiden kann, mit welchem Prozess er auf der Client-Seite über einen gerade erstellten Inter-Kommunikatoren kommunizieren kann, überträgt der Client-Prozess seinen eigenen Rang. Damit es nicht an dieser Stelle zu einer Verklemmung kommt, wird nicht-blockierend gesendet.

3. Programmierung

Die in [Uek16] und [Shu15] erwähnten Performance-Probleme wurden näher untersucht. Hierzu wurde die bestehende Implementierung auf eventuelle Ursachen geprüft.

Als erste mögliche Ursache wurde der Austausch der Portnamen über die Dateisysteme gesehen. Der erste Grund, das näher zu untersuchen, ist, dass das Dateisystem der langsamste Speicherort ist. Ein Weiterer ist der Workaround für das GPFS des SuperMUC (siehe Abschnitt 2.3.2). Als Alternative wäre ein Austausch der Portnamen über die Namenserver-Funktionalität von MPI in Betracht zu ziehen, da sich die Funktionsweise gleicht und damit leicht auch in PreCICE umgesetzt werden könnte. Ein weiterer interessanter Ansatz wurde bei den Gesprächen mit HLRS über deren Unterstützung des MPI Standards und die Nutzung der Beta-Version (siehe Abschnitt 2.2.2) entwickelt. Dieser sieht die Ersetzung der einzelnen Inter-Kommunikatoren durch einen einzelnen Inter-Kommunikator über alle Prozesse vor.

Daher verfolgen wir für eine weitere Alternative zum Austausch der Portnamen nur das theoretische Vorgehen, eine Implementation mit anschließender Testphase fand nicht satt. Diese theoretische Alternative sieht einen Austausch nach dem Vorbild des Kopplungsdatenaustauschs über die Master-Prozesse vor (siehe Abschnitt 2.3.1).

Um aussagefähige Ergebnisse produzieren zu können, benötigt man eine Referenzimplementierung des Kommunikationsaufbaus, gegen die man die Ergebnisse dieser Ansätze prüfen kann. Da die Referenzimplementierung sehr weit in PreCICE integriert ist, wird der Verbindungsaufbau der Inter-Kommunikatoren nachgebildet (siehe Abschnitt 2.3.2), um so leichter Tests durchführen zu können.

Im Folgenden wird die Nachbildung der Referenzimplementierung als Ansatz 0 (siehe Abschnitt 3.1.3) bezeichnet. Der Ansatz, der die Verwendung eines Nameserver vorsieht, wird als Ansatz 1 (siehe Abschnitt 3.1.4) bezeichnet. Ansatz 2 führt theoretisch den Austausch über Master-Prozesse durch (siehe Abschnitt 3.1.5). Der Ansatz, der einen einzelnen Inter-Kommunikator vorsieht, wird mit Ansatz 3 bezeichnet (siehe Abschnitt 3.1.6).

3.1. Testprogramm

Um mit einem Testprogramm effizient Testreihen durchführen zu können, ohne das Programm jedes Mal neu kompilieren zu müssen, bedarf es einer gewissen Konfigurierbarkeit (siehe Abschnitt 3.1.1). Da unser Ausgangspunkt vorsieht, dass alle Prozesse wissen, mit wem

sie kommunizieren müssen, bedarf es einiger Vorbereitungen (siehe Abschnitt 3.1.2). In den Abschnitten 3.1.3, 3.1.4, 3.1.5, 3.1.6 wird die Umsetzung der einzelnen Ansätze zum Kommunikationsaufbau beschrieben. Für den Abschluss des Tests sollten die erstellten Kommunikatoren benutzt werden, um auch Ergebnisse für den Datenaustausch zwischen Prozessen zu erzeugen (siehe Abschnitt 3.1.7). Abschließend wird die Produktion der Ergebnisse beschrieben (siehe Abschnitt 3.1.8).

3.1.1. Konfigurationsmöglichkeiten

Für unser Testprogramm verwenden wir folgende einstellbaren Parameter:

1. die Auswahl, ob eine Server-Instanz oder eine Client-Instanz ausgeführt wird
2. die Auswahl des zu testenden Ansatzes
3. die Anzahl der Verbindungen eines Prozesses zu Prozessen des anderen Kopplungsteilnehmers
4. die Größe der zufälligen Vektoren, die nach dem Verbindungsaufbau übertragen werden sollen
5. die Anzahl der Übertragungsdurchgänge
6. das Basisverzeichnis, in dem der Austausch der Portnamen über das Dateisystem stattfindet
7. das Basisverzeichnis, in das die Ergebnisse geschrieben werden
8. über MPI die Anzahl der Prozesse eines Kopplungspartners

Um die Implementierung des Testprogramms einfach zu halten, gelten für manche Parameter bestimmte Bedingungen. Grundsätzlich gilt für alle Parameter, die Zahlwerte (2,3,4,5,8) sind, dass die Werte auf Server- und Client-Instanzen immer identisch und positiv zu wählen sind. Für die Anzahl der Verbindungen gilt im speziellen, dass sie ungerade und kleiner sein müssen als die Anzahl der Prozesse eines Kopplungspartners. Für die Verzeichnisse gilt, dass sie existieren müssen, erreichbar sein müssen und die nötigen Rechte vorhanden sein müssen. Des Weiteren muss auch das Austauschverzeichnis identisch sein.

3.1.2. Kommunikationschema

Bei der Nachbildung der Referenzimplementierung benötigen wir eine Nachbildung der **Kommunikationszuordnung**. Um die Ergebnisse besser vergleichen zu können, führen wir ein Kommunikationsschema ein, mit dem alle Prozesse wissen, mit welchen anderen Prozessen sie Verbindungen aufbauen müssen. Voraussetzung für das Schema ist, dass die Anzahl der Prozesse der beiden Kopplungspartner gleich ist. Zudem muss die Anzahl der Verbindungen,

die jeder Prozess aufbauen muss, ungerade und kleiner sein als die Anzahl der Prozesse eines Kopplungspartners.

Das Schema sieht vor, dass alle Prozesse immer die gegebene Anzahl an Verbindungen benötigen und entsprechend alle Kommunikatoren erstellen müssen. Für jeden Prozess gilt dazu, dass er eine Verbindung mit dem Prozess mit gleichem Rang des anderen Kopplungspartner benötigt. Da die Verbindungszahl ungerade ist, bleibt noch eine gerade Verbindungsanzahl übrig. Die verbleibende Anzahl an Verbindungen wird zur Hälfte auf Prozesse mit Rängen, die nächst kleiner bzw. größer sind, verteilt. Um Rang-Über- bzw Unterläufe zu vermeiden, erfolgen alle Berechnungen Modulo der Prozessanzahl, so werden immer gültige Ränge gewählt. Also ist ein Rang unter 0, theoretisch Rang -1, der größte Rang. In Abbildung 3.1 ist ein Beispiel für Prozesse mit Rang 3 und 3 Verbindungen ausgeführt.

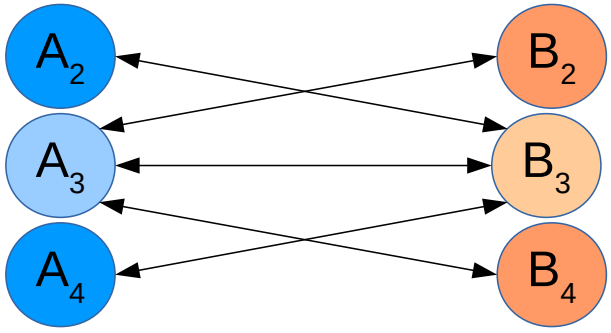

Abbildung 3.1.: Symmetrisches Kommunikationsschema für 3 Verbindungen für Prozesse mit Rang 3

3.1.3. Kommunikationsansatz 0

Da für Ansatz 0 gilt, dass es die Nachbildung der Referenzimplementierung ist und diese in Abschnitt 2.3.2 behandelt wird, werden nachfolgend nur die Unterschiede zur Referenz beschrieben.

Server

Um vergleichbare Ergebnisse für die Erstellung der Ordnerstruktur zu erhalten, gilt der Workaround als Standard (siehe Abschnitt 2.3.2). Durch das Ersetzen der Kommunikationszuordnung durch das Kommunikationsschema und dessen Vorgabe, dass alle Prozesse Verbindungen benötigen, wird für jeden Server-Prozess die Erstellung eines Ordners notwendig. Bei der Erstellung und der Veröffentlichung der Portnamen wurden keine Änderungen vorgenommen. Bei der Erstellung der Inter-Kommunikatoren wurde die Verwendung des Kommunikationsschemas eingeführt. Das betrifft die Anzahl der Verbindungen und die Ränge der Client-Prozesse, zu denen ein Inter-Kommunikator aufgebaut werden muss. Um eine direktere Zuordnung zwischen

Kommunikator und Rang des Client-Prozesses zu erreichen, wird direkt vom aktuellen Server-Prozess der Rang des Client-Prozesses empfangen. In der nun anschließenden Rücknahme der veröffentlichten Portnamen gibt es keine Veränderungen.

Client

Die Ersetzung der Kommunikationszuordnung führt nur zu Änderungen bei den Entscheidungen, zu welchen Rängen Inter-Kommunikatoren aufgebaut werden.

3.1.4. Kommunikationsansatz 1

Der Unterschied zwischen Ansatz 0 und Ansatz 1 liegt nur im Austausch der Portnamen. Da alle anderen Teile identisch sind, werden auch nur die Änderungen in diesem Bereich beschrieben. Der Portnamenaustausch erfolgt nicht mehr über das Dateisystem, sondern über das Verfahren zum Einsatz von Nameservern (siehe Abschnitt 2.1.4).

Auf dem Hazel Hen kann dieser Ansatz leider nicht eingesetzt werden, da die entsprechende Funktionalität in der MPI Implementierung nicht vorhanden ist.

Server

Durch die Ersetzung des Dateisystemaustauschs wird der Workaround unnötig und daher auch nicht ausgeführt. Das Veröffentlichen einer Datei mit dem Schreiben auf ein Dateisystem wird mit dem Veröffentlichen der Portnamen auf dem Nameserver ersetzt und der Dateiname kann unverändert als Service-Name weiterverwendet werden.

Durch einen Bug in der MPI Implementierung des SuperMUCs wurde die Rücknahme der veröffentlichten Ports ausgesetzt.

Client

Im Client wurde der Aufruf des Auffindens ersetzt. Der Dateinamen wird als Service-Namen verwendet. Um auf noch nicht veröffentlichte Portnamen warten zu können, musste die Fehlerbehandlung für MPI von der Standardfehlerbehandlung `MPI_ERRORS_ARE_FATAL`, mit der ein Prozess im Fehlerfall einfach still abstürzt, auf die vordefinierte Fehlerbehandlung `MPI_ERRORS_RETURN` umgestellt werden, damit der Befehl im Fehlerfall zurückkehrt und der Verwender entscheiden kann, welche weiteren Schritte eingeleitet werden müssen. In diesem Fall wird der Nameserver solange neu gefragt bis der Portname veröffentlicht wird.

3.1.5. Kommunikationsansatz 2

In diesem Abschnitt wird eine theoretische Vorgehensweise beschrieben, wie der Portaustausch über die Master-Prozesse ablaufen würde. Dazu werden die Änderungen, die gegenüber dem Ansatz der Referenzimplementierung zu machen wären, beschrieben.

Server

Durch die Ersetzung des Dateisystemaustauschs würde der Workaround unnötig werden und daher auch nicht ausgeführt werden. Stattdessen würde der Master-Prozess nach der Erstellung der Portnamen, diese einsammeln. Für unser Kommunikationsschema wäre das das direkt über MPI möglich (siehe Abschnitt 2.1.6), da für alle Prozesse vorgesehen ist, dass sie Kommunikatoren aufbauen würden. Beim Einbau müsste an dieser Stelle überlegt werden wie dieses Einsammeln funktionieren würde. Für die Slave-Prozesse würde sich an der Veröffentlichung ändern, dass die Veröffentlichung nicht über das Dateisystem abläuft, sondern der erstellte Portname an den Master-Prozess gesendet wird. Bei einer unvollständigen Liste müsste man sich hier Gedanken machen, wie die Liste an Portnamen über die aus dem Kopplungsaustausch bestehende Verbindung an den Master-Prozess des Clients zu senden ist, damit die Client-Seite damit etwas anfangen könnte.

Das anschließende Vorgehen bleibt bis auf die fehlende Notwendigkeit der Rücknahme der veröffentlichten Portname, weil keine veröffentlicht wurden, unverändert.

Client

Auf der Client-Seite ändert sich zunächst, dass der Master-Prozess eine Liste an Portnamen vom Master-Prozess des Servers erhalten würde. Diese Liste würde der Master-Prozess anschließend an alle seine Slave-Prozesse verteilen. Auch hier tritt das Problem auf, dass nicht alle Prozesse Verbindungen erstellen wollen und damit ein separater Verteil-Mechanismus entwickelt werden müsste. Ist die Liste auf den Slave-Prozessen vorhanden, würde in ihr an den passenden Stellen der Portname gesucht werden, wodurch die Suche auf dem Dateisystem ersetzt werden würde. Im weiteren Verlauf würden keine Veränderungen nötig werden.

3.1.6. Kommunikationsansatz 3

Bei Ansatz 3 müssen grundsätzlich alle Prozesse teilnehmen. Es ist vorgesehen, dass nicht für jede Verbindung zwischen Prozessen ein Inter-Kommunikator aufgebaut wird, sondern nur ein einziger Inter-Kommunikator für alle Prozesse. Dazu wird auf beiden Seiten, wie in Abschnitt 2.1.6 beschrieben, vorgegangen. Der Unterschied zu den bisherigen Ansätzen besteht darin, dass anstelle der Intra-Kommunikatoren `MPI_COMM_SELF` der einzelnen Prozesse der

Intra-Kommunikator des gesamten Kopplungspartners verwendet wird, um den einen Inter-Kommunikator zu erstellen. In der Testsoftware wird dazu `MPI_COMM_WORLD` verwendet. Dies führt zu einer vermeintlich zentralen Kommunikationsstruktur, die durch die Einführung der Peer-to-Peer-Struktur und den direkten Kommunikatoren ersetzt wurde (siehe Abbildung 3.2). In den Gesprächen mit dem HLRS wurde versichert, dass dies nicht zu einem zu vermutenden Flaschenhals führen würde. Dieser Flaschenhals soll durch eine effiziente Programmierung vermieden worden sein.

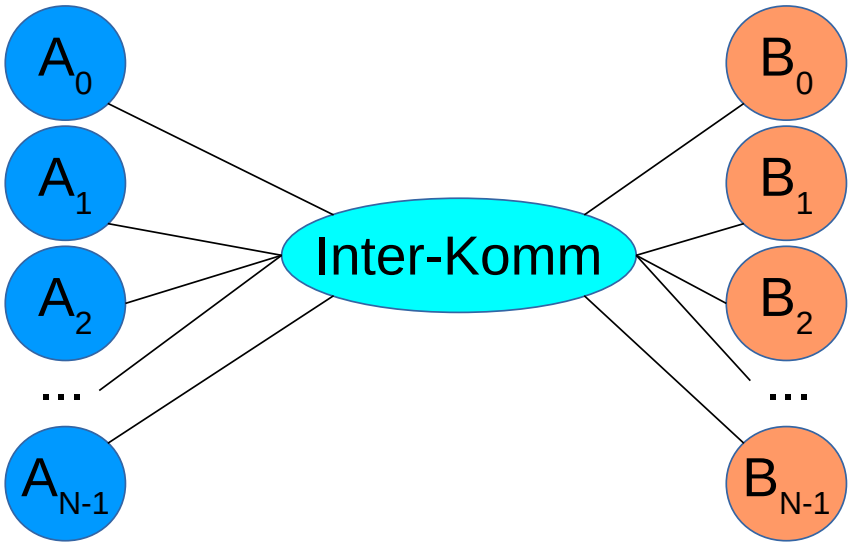

Abbildung 3.2.: Kommunikation über einen Inter-Komm

Server

Auf der Server-Seite werden keine Workarounds mehr benötigt und der Master-Prozess erstellt als Einziger einen Portnamen und veröffentlicht diesen über das Dateisystem. Im Anschluss erstellen alle Prozesse gemeinsam den einzigen Inter-Kommunikator zur Client-Seite. Durch dieses Vorgehen müssen auch Kommunikatoren nicht mehr Rängen von Client-Prozessen zugeordnet werden. Abschließend zieht der Master-Prozess die Veröffentlichung des Portnamens zurück.

Client

Auf der Client-Seite ist der Master-Prozess der einzige, der einen Portnamen auf dem Dateisystem finden muss. Im Anschluss erstellen alle Prozesse gemeinsam den Inter-Kommunikator zur Server-Seite. Das Senden der Ränge wird auch hier nicht mehr benötigt.

3.1.7. Verwendung der Kommunikatoren

Zur Funktionsprüfung der Kommunikatoren wird der Datenaustausch einer Simulation simuliert. Dazu übertragen alle Prozesse, dem Kommunikationsschema entsprechend, zufällige Daten an andere Prozesse. Dies entspricht dem Austausch an Daten zwischen Prozessen, die in den Einträgen der Kommunikationszuordnung zu finden sind.

Die Umsetzung dieser Simulation nutzt die in den Parametern des Testprogramms festgelegte Anzahl an Übertragungsrunden und die festgelegte Größe der zu übertragenden Vektoren. Auf jedem Prozess wird pro Übertragungsrunde ein neuer zufälliger Vektor erstellt. Im Anschluss folgt der Prozess dem Kommunikationsschema, beginnend mit dem kleinsten Rang, um die Vektoren über den zugehörigen Kommunikator auszutauschen. Der Prozess auf Server-Seite beginnt mit der Senden-Operation, während der Prozess auf der Client-Seite zuerst empfängt.

3.1.8. Produktion der Ergebnisse

Die Ergebnisse einer Performanz-Untersuchung sind Zeitdauern, die ein Befehl beispielsweise benötigt. Interessante Zeiten auf der Server-Seit sind:

- Das Ausführen des Workaround falls nötig
- Das Veröffentlichen der Portnamen
- Das Erstellen der Inter-Kommunikatoren auf der Server-Seite
- Das Zurückziehen eines veröffentlichten Portnamens
- Die Zeit für die Datenaustausch-Simulation

Auf der Client-Seite wiederum sind folgende Zeiten interessant:

- Das Auffinden eines veröffentlichten Portnamens
- Das Erstellen der Inter-Kommunikatoren
- Die Zeit für die Datenaustausch-Simulation

Das Erheben der Zeiten erfolgt für jeden Prozess in dem jeweils zu untersuchenden Abschnitt. So können wir am Ende die durchschnittlichen Zeiten, des jeweiligen Abschnitts über alle Prozesse berechnen.

Vor jeder Zeitmessung muss eine entsprechende Synchronisierung stattfinden, damit keine Verfälschung durch Wartezeiten entsteht. Um Zeiten zu erheben, wird immer nach der Synchronisierung, aber vor dem zu untersuchenden Abschnitt der Startzeitpunkt bestimmt. Im Anschluss wird der Endzeitpunkt bestimmt. Die Differenz der beiden Zeitpunkte bildet die Zeitdauer, die der Prozess für diesen Abschnitt benötigt hat.

Ansatz 3 lieferte die Vorlage zur Synchronisierung vor dem eigentlichen Erstellen der Inter-Kommunikatoren. Daher wurde nach der Veröffentlichung der Portnamen ein Synchronisierungs-Inter-Kommunikator erstellt, der für das Synchronisieren der danach folgenden Abschnitte verwendet werden konnte.

Zur Zeitmessung der Erstellung der Inter-Kommunikatoren wurden die Zeiten der einzelnen Erstellungen addiert, um dabei die Kontrollstrukturen auszulassen, aber einen Wert zu erhalten. Bei der Datenaustausch-Simulation wurden die Kontrollstrukturen allerdings mitgerechnet, da diese Messung durch fehlende reale Referenzwerte nur als grober Anhaltspunkt gemessen wird.

Am Ende schreibt jeder Prozess seine Zeiten in eine eigene Datei. Diese Dateien wurden von einem Script eingelesen und die Durchschnittswerte der Zeiten berechnet.

3.2. Bau und Ausführung der Testsoftware

Zum Bau der Testsoftware wurden auf beiden Referenzsystemen folgende Optionen für die Kompilierung verwendet:

- -Wall
- -std=c++11
- -O3

Die Ausführung erfolgt auf beiden Systemen mit so genannten Ausführungsskripten, die die Umgebungs-Konfiguration und die eigentlichen Aufrufe der Software enthalten. Diese Aufrufskripte werden zur Ausführung den Aufgabenverwaltungen übergeben.

3.2.1. SuperMUC

In Anhang A ist das Gerüst eines Ausführungsskriptes dargestellt, das für diese Arbeit Anwendung fand.

Wichtige Konfigurations-Parameter:

- node, die Anzahl der angeforderten Knoten
- tasks_per_node, Anzahl der Prozesse auf einem Knoten; in dieser Arbeit die Anzahl der echten Kerne auf einem Knoten
- class, die Warteschlange, in die sich das Script einreihen soll
- wall_clock_limit, das Zeitlimit, das wir der Aufgabe geben

Wichtig für Benutzer des SuperMUCs, die mehrere Programme in einen Ausführungsskript starten möchten, ist, dass sie die zugewiesenen Knoten-Adressen selbst über so genannte Maschinefiles an die MPI-Aufrufe verteilen müssen.

War die Erstellung der Maschinefiles abgeschlossen, wurden parallel die beiden Kopplungspartner mit dem MPI-Starter *mpiexec* gestartet. Als Parameter kamen die Anzahl der für dieses Programm vorgesehenen Prozesse und das Maschinefile zur Identifizierung jener Prozesse zum Einsatz. Um die Nameserver-Funktionalität nutzen zu können, musste vor den Programmstarts der Nameserver *hydra_nameserver* gestartet werden und dessen Adresse auch dem MPI-Starter als Parameter übergeben werden.

3.2.2. Hazel Hen

Beim Bau der Software musste auf Hazel Hen beachtet werden, dass die Beta-Bibliothek von MPI verwendet wird.

In Anhang B ist das Gerüst eines Ausführungsskriptes dargestellt, das für diese Arbeit Anwendung fand.

Wichtige Konfigurations-Parameter sind:

- nodes, die Anzahl der angeforderten Knoten
- ppn, Anzahl der Prozesse auf einem Knoten; in dieser Arbeit die Anzahl der echten Kerne auf einem Knoten
- class, die Warteschlange, in die sich das Script einreihen soll
- walltime, das Zeitlimit, das wir der Aufgabe geben

Zur Nutzung der Beta müssen für Nutzer zusätzliche Rechte zum Erstellen/Löschen eigener Domains, gewährt werden. Die Domainverwaltung geschieht normalerweise automatisch im MPI-Starter und muss nicht von Nutzer konfiguriert werden. Vor den eigentlichen Programmstarts musste, der Befehl `apmgr pdomain -uc TestDomain` benutzt werden, um eine Domain wie z.B. für `TestDomain` zu erstellen.

Im Anschluss wurden parallel die beiden Kopplungspartner mit dem MPI-Starter *aprun* gestartet. Als Parameter mussten nur die Prozessanzahl des Kopplungspartners und die zuvor erstellte Domain übergeben werden. Sehr angenehm ist zudem, dass der Benutzer sich nicht um die Verwaltung der Knoten-Adressen und die Machinefiles kümmern muss.

Nachdem beide Aufrufe beendet waren, musste die Domain durch Ausführen von `apmgr pdomain -ur TestDomain` gelöscht werden.

4. Evaluation

Im Folgenden werden die Ergebnisse von Tests dargestellt, die mit dem in Kapitel 3 beschriebenen Test-Programm durchgeführt wurden.

4.1. Testbedingungen

Diese Tests wurden auf dem SuperMUC (siehe Abschnitt 2.2.1) und auf dem Hazel Hen (siehe Abschnitt 2.2.2) durchgeführt. Es gab grundsätzlich zwei Testreihen auf jedem Supercomputer. In der ersten Testreihe wurde die Anzahl der Verbindungen, die ein Prozess zu Prozessen des anderen Kopplungsteilnehmers aufbaut, konstant bei drei Verbindungen belassen. Bei der zweiten Testreihe wurde die Anzahl prozentual an der Anzahl der Prozesse eines Kopplungsteilnehmers berechnet. Es wurden immer 33% der Prozessanzahl eines Kopplungsteilnehmers gewählt. In Tabelle 4.1 werden alle sich verändernden Werte der Testreihen für den SuperMUC aufgelistet. Die Werte für HazelHen können der Tabelle 4.2 entnommen werden.

Nicht aufgeführte Werte sind die Parameter der Datenübertragungs-Simulation, die am Ende des Testprogramms stattfindet und die Basisverzeichnisse für den Portnamen-Austausch und die Ergebnisse.

Für die Anzahl der Übertragungsdurchgänge wird immer 10 und für die Größe des zu übertragenden Vektors immer 100 gewählt. Da die Ergebnisse dieser Minisimulation nur einen ungefähren Eindruck vermitteln sollen, wurden keine praxisnahen Werte benötigt.

Für das Austausch-Verzeichnis wurde auf dem SuperMUC ein Verzeichnis auf dem GPFS und auf dem Hazel Hen ein extra Arbeitsverzeichnis angegeben. Für die Ergebnisse wurde in beiden Fällen ein Ordner im Heimverzeichnis gewählt.

4.2. Auswertung der Ergebnisse

Zur Auswertung folgen wir dem Programmablauf chronologisch und werten die entsprechenden Abschnitte immer gemeinsam für beide Testsysteme aus. Grundsätzlich werden nur die Ergebnisse für die zweiten Testreihen mit den 33% Verbindungen ausgewertet, da sie realistischer sind. In Fällen, in denen die Ergebnisse aus der ersten Testreihe eine zusätzliche Erkenntnis bringen, werden diese mit ausgewertet.

Knoten	pro Kopplungs-Partner		Warteschlange	3 Verb.	33 % Verb.
	Knoten	Kerne			
2	1	16	test	3	5
4	2	32	test	3	11
8	4	64	test	3	21
16	8	128	test	3	43
32	16	256	test	3	85
64	32	512	general	3	169
128	64	1024	general	3	337
256	128	2048	general	3	675

Tabelle 4.1.: Testreihen SuperMUC

Knoten	pro Kopplungs-Partner		Warteschlange	3 Verb.	33 % Verb.
	Knoten	Kerne			
2	1	24	test	3	7
4	2	48	test	3	15
8	4	96	test	3	31
16	8	192	test	3	63
32	16	384	test	3	127
64	32	768	test	3	253
128	64	1536	test	3	507
256	128	3072	test	3	1013
512	256	6144	multi	3	2027

Tabelle 4.2.: Testreihen Hazel Hen

4.2.1. Ordnererstellung

Bei der Ordnererstellung und ihrer Funktion als Workaround wurden für beide Supercomputer die beiden Testreihen gegenübergestellt. In Abbildung 4.1 sind die Ergebnisse des SuperMUCs dargestellt und in Abbildung 4.2 die Ergebnisse des Hazel Hen.

Als Ergebnis wurde erwartet, dass sich keine signifikanten Unterschiede zwischen den Testreihen ergeben würden. Des Weiteren wurde erwartet, dass mit steigender Kernzahl die Ordnererstellung nicht skaliert, da mit steigendem Nutzungsverhalten die Geschwindigkeit sinkt und die Erstellung so immer länger dauern würde. Wie in beiden Schaubildern zu sehen ist, werden diese Erwartungen erfüllt, trotz einiger Ausreißer an der einen oder anderen Stelle. Ursache für diese kleinen Ausreißer könnte sein, dass das Dateisystem über alle Nutzer geteilt wird und es bei einem hohen Nutzungsaufkommen für alle langsamer wird.

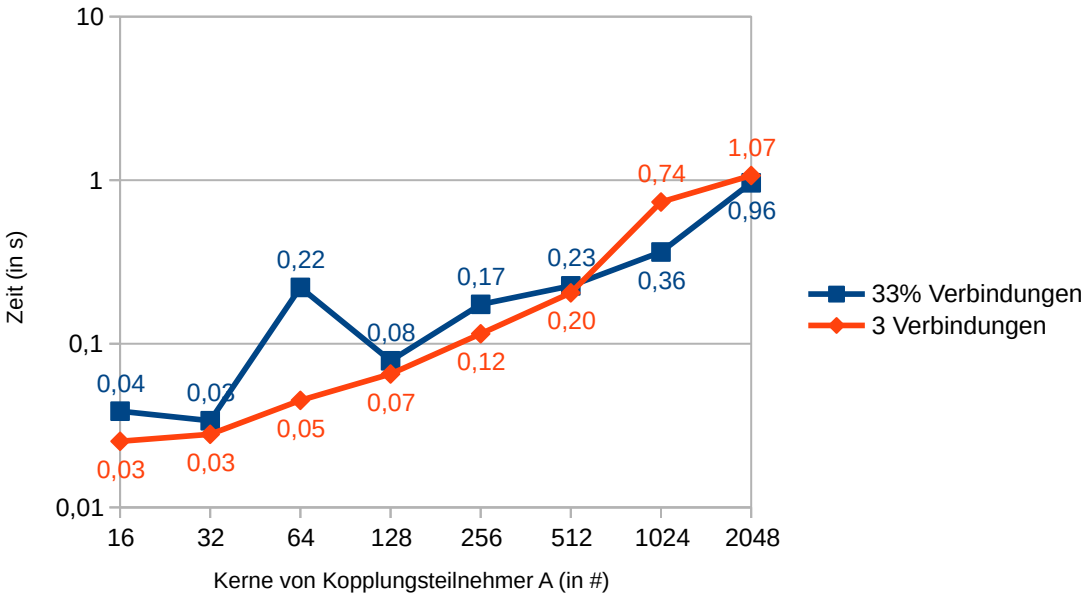

Abbildung 4.1.: Ordnererstellung auf dem SuperMUC für alle Kerne

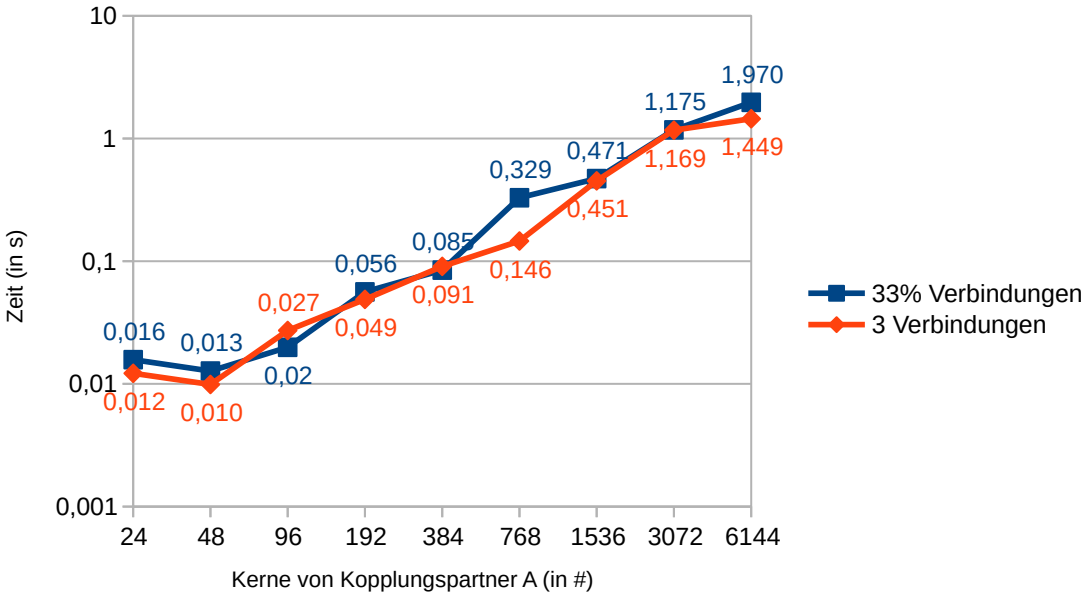

Abbildung 4.2.: Ordnererstellung auf dem SuperMUC für alle Kerne

4.2.2. Portnamen-Veröffenlichung

Bei der Veröffentlichung von Portnamen werden für den SuperMuc in Abbildung 4.3 erstmals alle drei implementierten Ansätze verglichen. Für den Hazel Hen können in Abbildung 4.4 nur Ansatz 0 und Ansatz 3 miteinander verglichen werden, da die Nameserver-Funktionalität nicht zum Funktionsumfang gehört.

Es wurde erwartet, dass die Veröffentlichung der Portnamen in Ansatz 3 am schnellsten ist und mit steigender Kernanzahl konstant bleibt, da hier immer nur ein Portname veröffentlicht werden muss. Im Vergleich zwischen Ansatz 0 und Ansatz 1 wurde erwartet, dass Ansatz 1 schneller ist, da die Daten nicht über das Dateisystem laufen müssen. Des Weiteren wurde erwartet, dass Ansatz 0 nicht skalieren würde. Für Ansatz 0 wurde die steigende Nutzung des Dateisystems als mögliches Problem gesehen und für Ansatz 1 war nicht sicher, wie der Nameserver reagieren würde.

Auf Hazel Hen wurde die Erwartung bezüglich Ansatz 3 erfüllt, für Ansatz 0 hingegen lässt sich sagen, dass der Ansatz zumindest bis 3072 Kerne skaliert und im nächsten Schritt eine schlagartige Verlangsamung eintritt.

Auf dem SuperMuc zeigt sich, dass das Dateisystem besser skaliert als erwartet und für Ansatz 3 mit dem einzelnen Portnamen erst ab 512 Kernen klar schneller ist. Ansatz 3 verläuft extrem unregelmäßig, was die Abhängigkeit von der Auslastung auf dem Dateisystem widerspiegelt. Die Testreihe für Ansatz 1 wurde bei 128 Kernen beendet, da der Nameserver bei der Ausführung mit 256 Kernen pro Kopplungspartner meldete, er sei an seinem Verbindungslimit. Zu den wenigen Werten, die verfügbar sind, lässt sich sagen, dass sie gleichmäßiger sind als die der beiden anderen Ansätze und dass sie sich in deren Wertebereichen aufhalten.

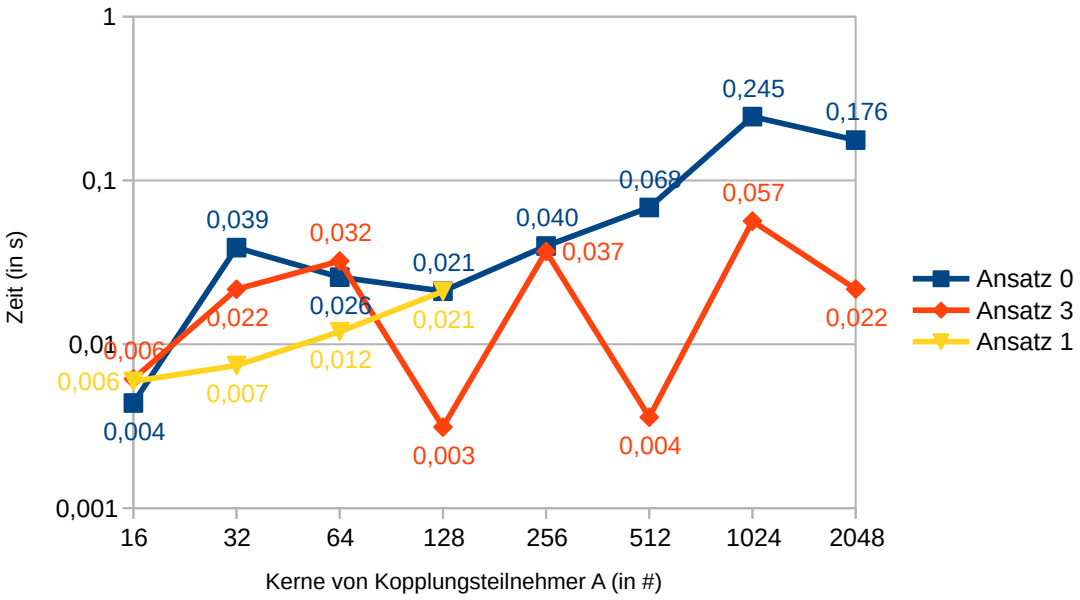

Abbildung 4.3.: Portnamen-Veröffentlichung auf dem SuperMUC mit 33% Verbindungen

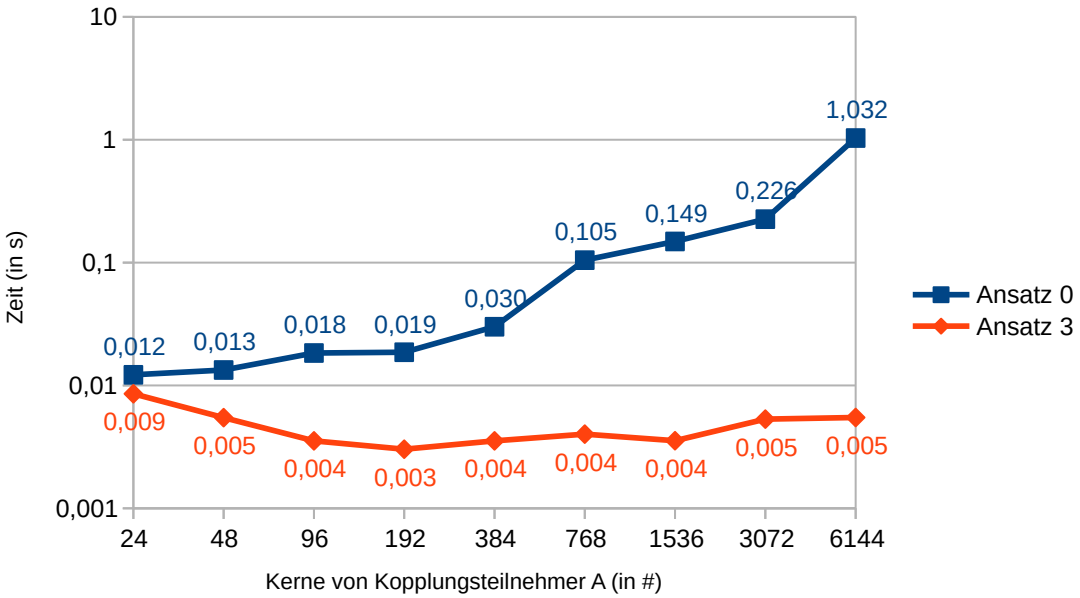

Abbildung 4.4.: Portnamen-Veröffentlichung auf dem Hazel Hen mit 33% Verbindungen

4.2.3. Auffinden der Portnamen

Beim Auffinden der Portnamen werden für den SuperMUC in Abbildung 4.5 alle drei Ansätze und für den Hazel Hen in Abbildung 4.6 die Ansätze 0 und 3 miteinander verglichen.

Erwartet wurden die gleichen Ergebnisse wie in Abschnitt 4.2.2, da es Gegenstücke sind.

Auf dem Hazel Hen war wie erwartet Ansatz 3 konstant und schneller als Ansatz 0. Das Dateisystem skaliert deutlich besser und kann je nach Auslastung auch nahezu konstant sein.

Auf dem SuperMUC erfüllt Ansatz 3 die Erwartungen. Für Ansatz 0 gilt, dass er besser skaliert als erwartet und wie auf Hazel Hen nahezu konstant aber lansamer als Ansatz 0 ist. Ein überraschendes Ergebnis lieferten die Zahlen der abgebrochenen Testreihe von Ansatz 1. Ansatz 1 war um den Faktor 1000 langsamer als Ansatz 0 und skalierte. Die Ursache könnte sein, dass der Namenserver der Flaschenhals ist und das Dateisystem durch paralleles Lesen die grundsätzliche Geschwindigkeit halten kann.

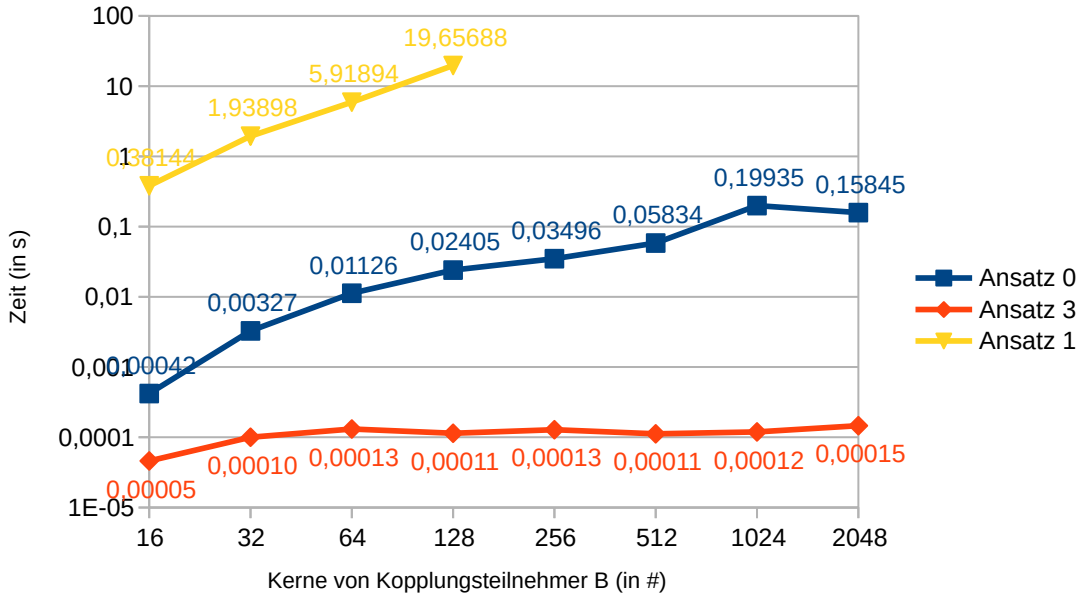

Abbildung 4.5.: Auffinden auf dem SuperMUC mit 33% Verbindungen

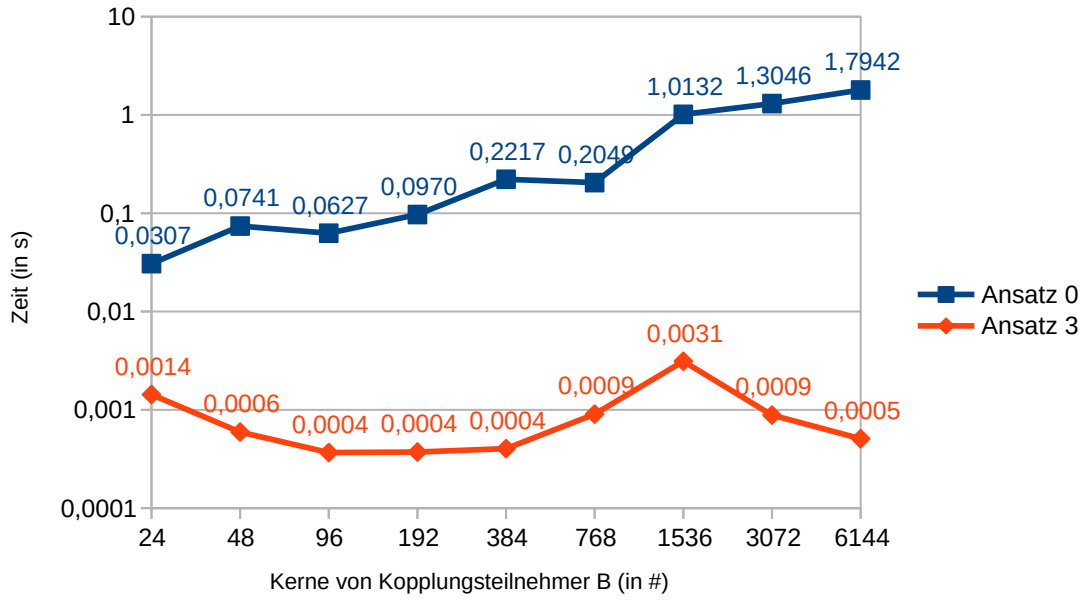

Abbildung 4.6.: Auffinden auf dem Hazel Hen mit 33% Verbindungen

4.2.4. Kommunikator-Erstellung

Bei der Kommunikator-Erstellung werden die Server-Seite und die Client-Seite ausgewertet.

Als Ergebnis wurde erwartet, dass sich die Zahlen für Server und Client in etwa gleichen.

Dies konnte auch bestätigt werden. Nur auf dem SuperMUC in Ansatz 3 skaliert die Server-Seite (siehe Abbildung 4.7), während die Client-Seite nahezu konstant ist (siehe Abbildung 4.9)

Server

Zur Erstellung der Kommunikatoren auf der Server-Seite auf dem SuperMUC in Abbildung 4.7 wurden alle drei Ansätze und auf dem Hazel Hen in Abbildung 4.8 nur die Ansätze 0 und 3, ausgewertet.

Als Ergebnis wurde erwartet, dass Ansatz 3 der schnellste ist und bei steigender Kernanzahl etwas besser skaliert als die anderen Ansätze. Für die Ansätze 0 und 1 wurde erwartet, dass sie sich identisch verhalten und bei steigender Kernanzahl schlecht skalieren.

Für Hazel Hen treffen diese Erwartungen zu.

Auf dem SuperMUC überrascht für Ansatz 3, dass er bei steigender Kernanzahl skaliert. Die Ansätze 0 und 1 skalieren wie erwartet schlecht und sind überraschender Weise nicht identisch.

Ansatz 1 ist für die verfügbaren Werte immer schneller. Ursache hierfür könnte ein mögliches Caching innerhalb von MPI sein, welches vom Veröffentlichen der Portnamen herrührt.

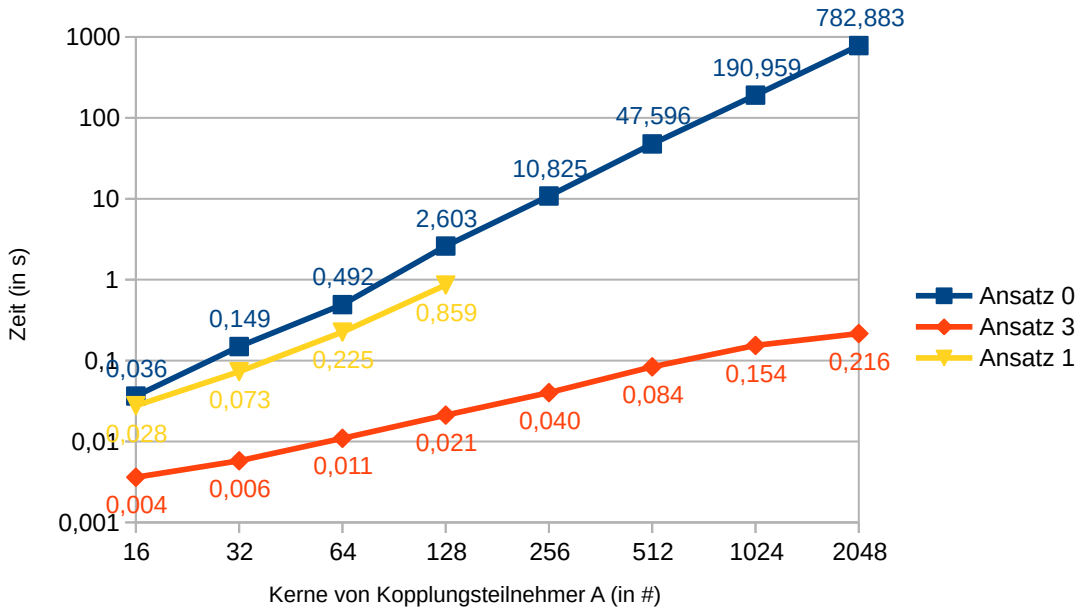

Abbildung 4.7.: Server-Seite der Kommunikator-Erstellung auf dem SuperMUC mit 33% Verbindungen

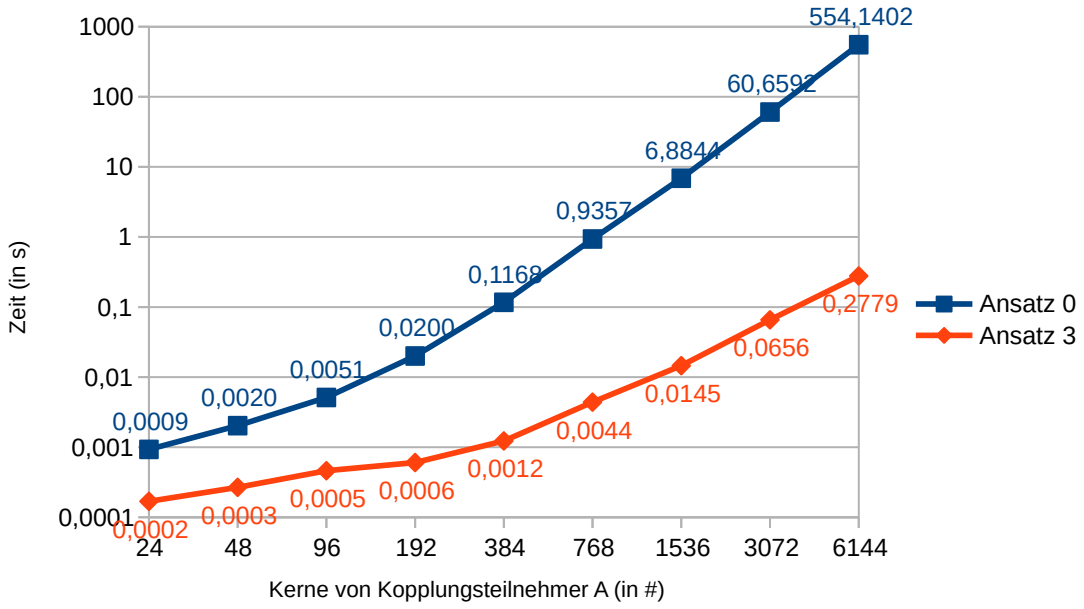

Abbildung 4.8.: Server-Seite der Kommunikator-Erstellung auf dem HazelHen mit 33% Verbindungen

Client

Zur Erstellung der Kommunikatoren auf der Client-Seite auf dem SuperMUC wurden in Abbildung 4.9 alle drei Ansätze und auf dem Hazel Hen in Abbildung 4.10 die Ansätze 0 und 3 ausgewertet.

Erwartetet wurden die gleichen Ergebnisse wie auf dem Server-Teil in Abschnitt 4.2.4.

Für den Hazel Hen werden die Erwartungen erfüllt.

Auf dem SuperMUC ist Ansatz 3 nicht nur schneller als die anderen Ansätze, er ist überraschender Weise konstant mit steigender Kernanzahl. Die Ansätze 0 und 1 skalieren wie erwartet nicht mit steigender Kernanzahl. Wie auf der Server-Seite auch, ist der Ansatz 1 immer schneller als Ansatz 0 und könnte seine Ursache in einem möglichen Caching innerhalb von MPI haben, das vom Auffinden der Portnamen herrührt.

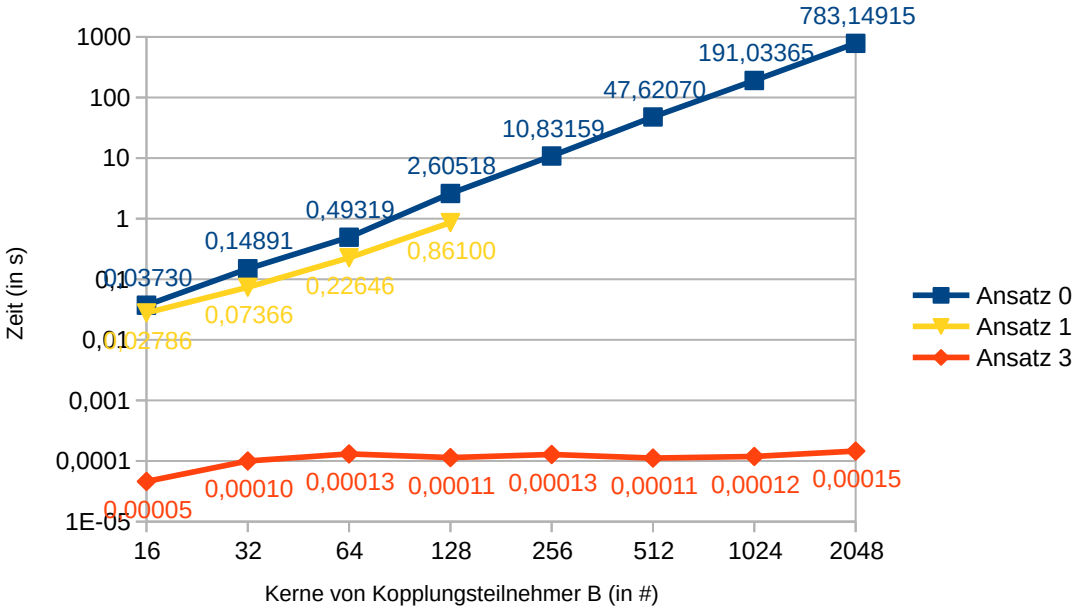

Abbildung 4.9.: Server-Seite der Kommunikator-Erstellung auf dem SuperMUC mit 33% Verbindungen

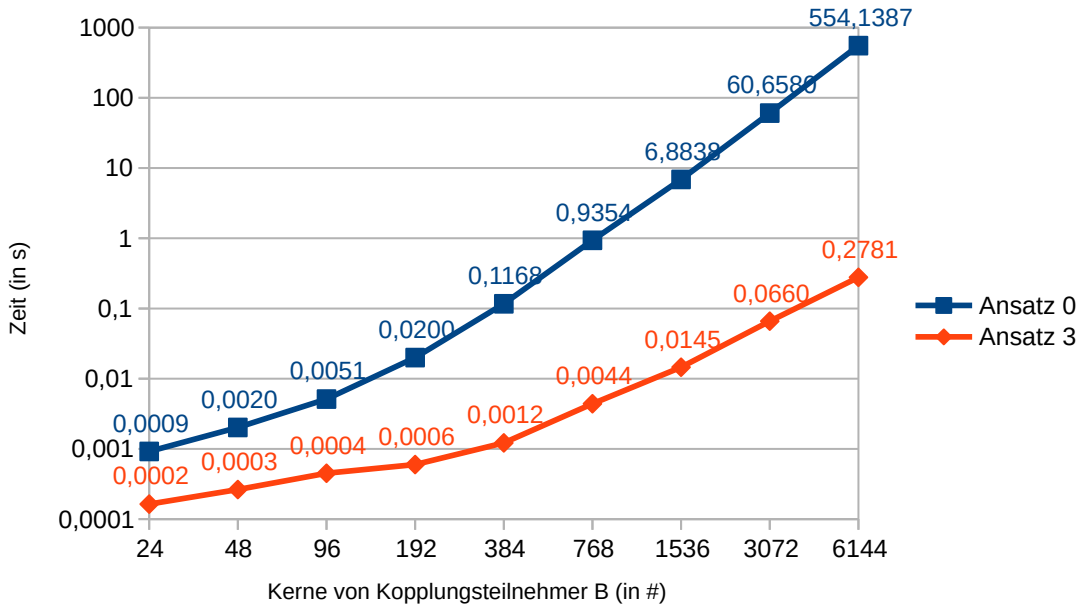

Abbildung 4.10.: Client-Seite der Kommunikator-Erstellung auf dem HazelHen mit 33% Verbindungen

4.2.5. Zurückziehen einer Veröffentlichung

Zum Zurückziehen einer Veröffentlichung wurden auf dem SuperMUC in Abbildung 4.11 und auf dem Hazel Hen in Abbildung 4.12 die Ansätze 0 und 3 ausgewertet. In Ansatz 1 auf dem SuperMUC konnten hierfür keine Zahlen erstellt werde, da es in der MPI-Implementierung einen Bug aufweist.

Als Ergebnisse wurde erwartet, dass Ansatz 3 immer schneller ist und mit steigender Kernanzahl konstant bleibt, da immer nur für einen Portname die Veröffentlichung zurückgenommen werden muss. Für den Ansatz 0 wurde erwartet, dass er bei steigender Kernanzahl nicht skaliert.

Auf beiden Systemen ist der Ansatz 3 überraschend nicht konstant, er skaliert noch nicht einmal. Ansatz 0 verhält sich wie erwartet nicht skalierend.

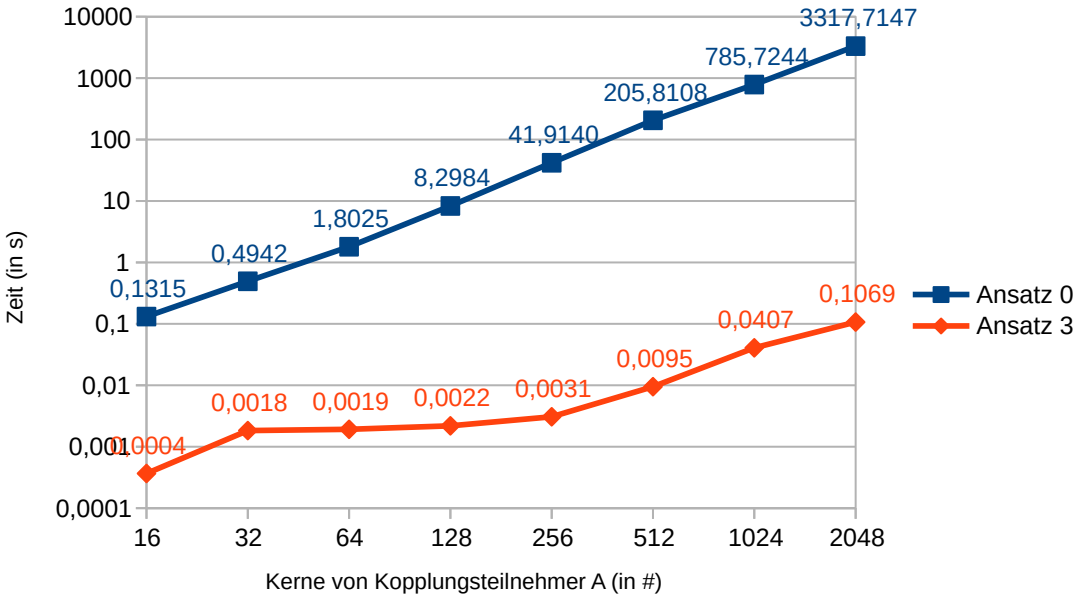

Abbildung 4.11.: Zurückziehen der veröffentlichten Portnamen auf dem SuperMUC mit 33% Verbindungen

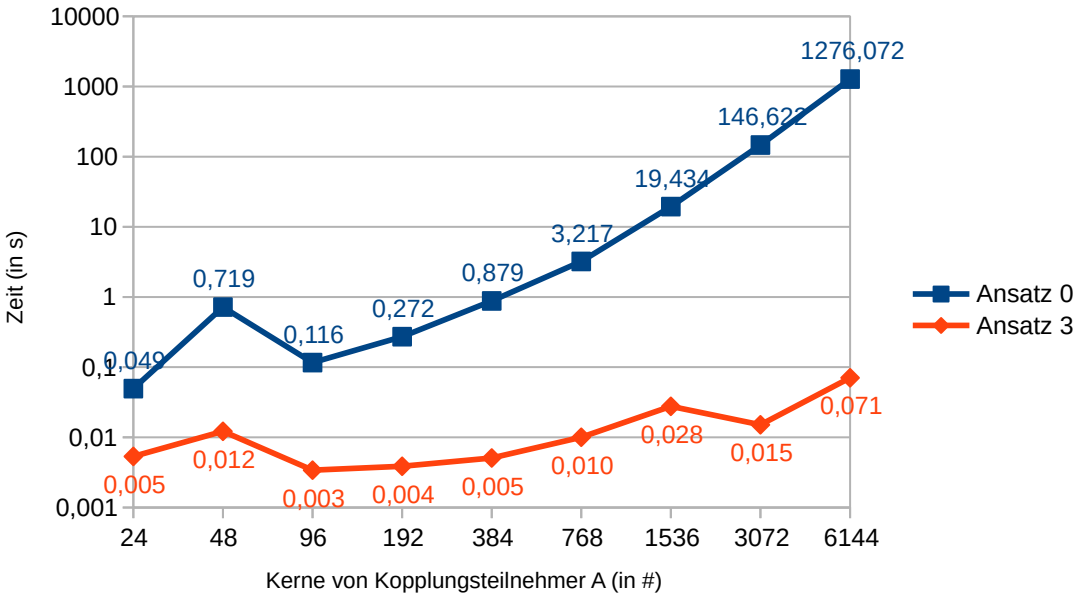

Abbildung 4.12.: Zurückziehen der veröffentlichten Portnamen auf dem HazelHen mit 33% Verbindungen

4.2.6. Datenaustausch-Simulation

Für die Datenaustausch-Simulation werden die Server-Seite und die Client-Seite ausgewertet. Für die Server-Seite werden Abbildungen 4.13 und 4.14 und für die Client-Seite die Abbildungen 4.15 und 4.16 ausgewertet. Für den SuperMUC werden alle drei Ansätze behandelt und für den Hazel Hen die Ansätze 0 und 3.

Als Gegenstücke wurde erwartet, dass sich die Zahlen für Server und Client in etwa gleichen.

Da diese Erwartungen von beiden Systemen erfüllt werden, folgt daraus, dass nur eine Seite ausgewertet werden muss.

Für Ansatz 3 wurde erwartet, dass er skaliert und zumindest so schnell wir Ansatz 0 ist. Ansatz 0 und Ansatz 1 haben die gleiche Kommunikationsstruktur, daher wird erwartet, dass sich nahezu identische Zahlen ergeben. Des Weiteren wird erwartet, dass sie nicht skalieren.

Zusammenfassend lässt sich sagen, dass keiner der Ansätze auf keinem der Systeme skaliert.

Für den Hazel Hen lässt sich erkennen, dass Ansatz 0 für kleinere Kernzahlen schneller ist als Ansatz 3. Erst bei hohen Kernzahlen ist Ansatz 3 schneller als Ansatz 0.

Auf dem SuperMUC kann man überraschend feststellen, dass Ansatz 3 nicht annähernd so schnell ist wie Ansatz 0 oder 1 und Ansatz 3 durchgängig um einen Faktor 100 langsamer ist.

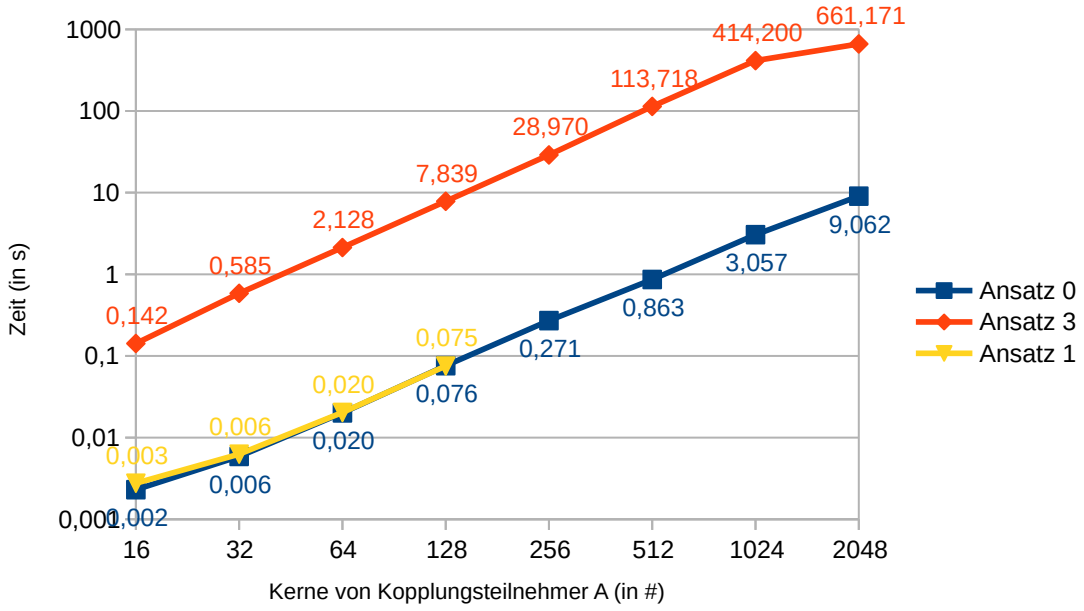

Abbildung 4.13.: Server-Seite der Datenaustausch-Simulation auf dem SuperMUC mit 33% Verbindungen

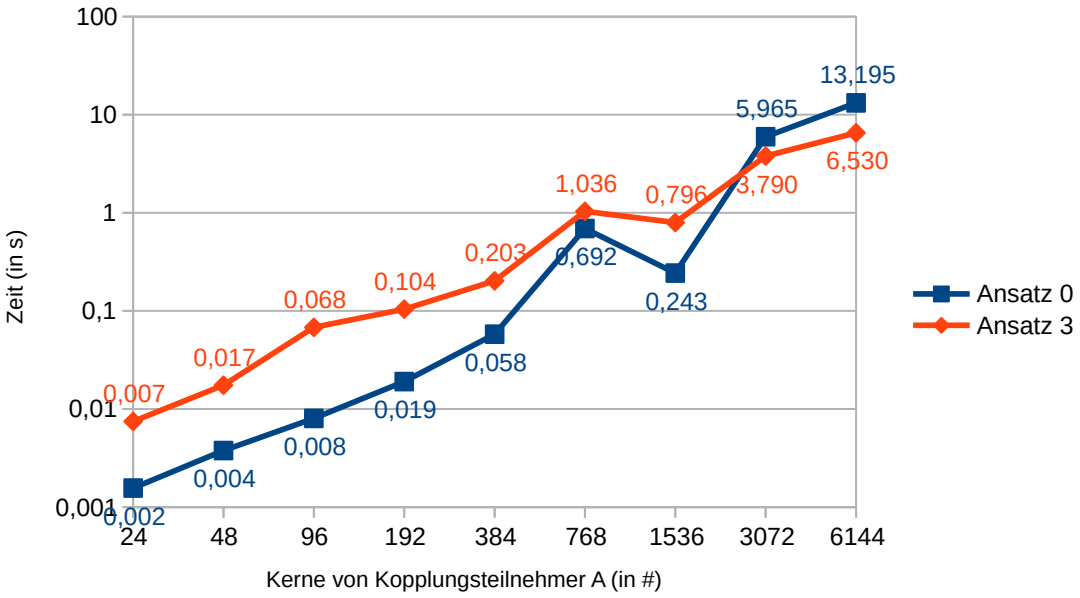

Abbildung 4.14.: Server-Seite der Datenaustausch-Simulation auf dem HazelHen mit 33% Verbindungen

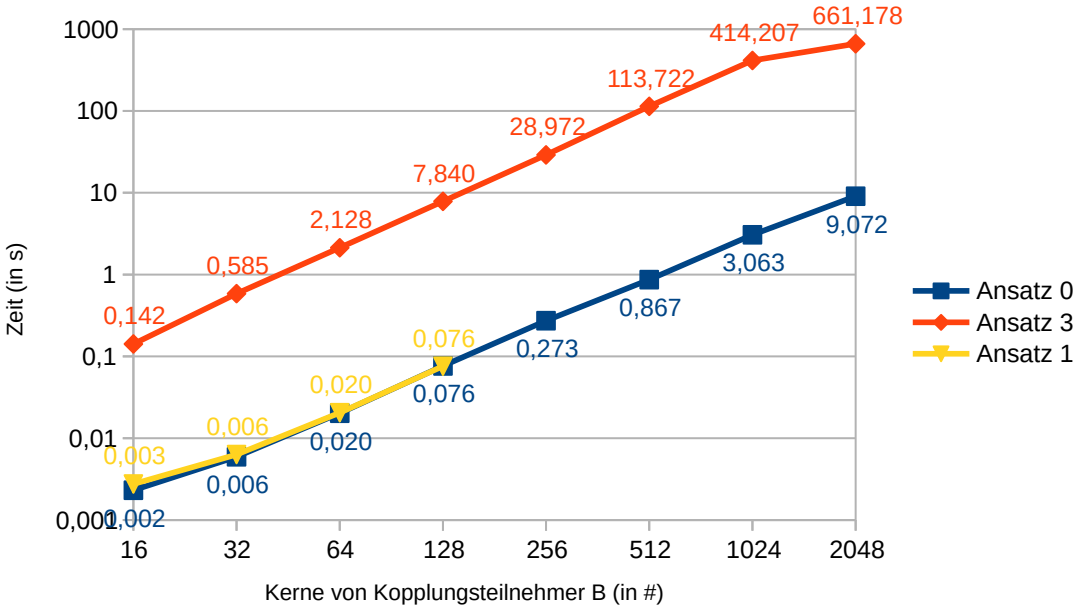

Abbildung 4.15.: Client-Seite der Datenaustausch-Simulation auf dem SuperMUC mit 33% Verbindungen

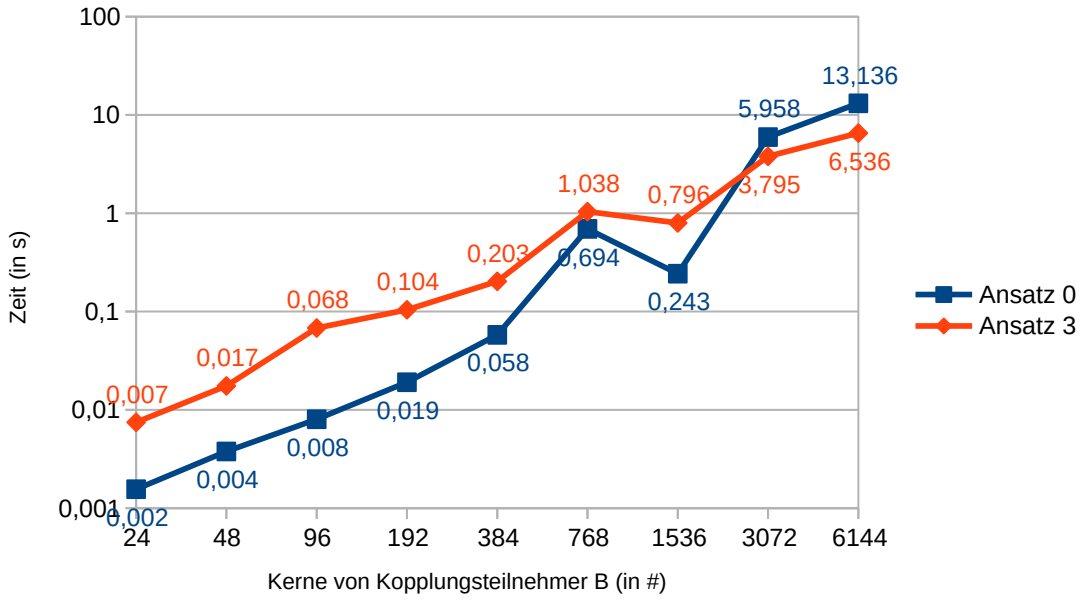

Abbildung 4.16.: Client-Seite der Datenaustausch-Simulation auf dem HazelHen mit 33% Verbindungen

5. Zusammenfassung und Ausblick

Gegenstand der vorliegenden Arbeit sind Untersuchungen von Ansätzen zur Optimierung einer verteilten Multi-Physik Simulationssoftware.

Ausgehend vom Verbindungsaufbau zwischen Kopplungspartnern von PreCICE wurden weitergehende Untersuchungen bezüglich möglicher Probleme im Kommunikationsaufbau und der Kommunikation zwischen Prozessen angestellt. Als mögliches Problem wurde der Austausch der Portnamen über das Dateisystem ausgemacht. Um einfachere Tests durchführen zu können, wurde die Referenzimplementierung in einem Testprogramm nachgebaut und Ansatz 0 genannt. Als alternative Austauschmöglichkeit wurde in das Testprogramm der Austausch der Portnamen über einen Nameserver als Ansatz 1 implementiert. Ansatz 2 war wegen zu hohem Aufwand nur theoretisch durchdacht worden, er würde den Austausch über Master-Prozesse vorsehen. Als vielversprechender wurde mit Ansatz 3 der Verbindungsaufbau über nur noch einen Inter-Kommunikator für alle Verbindungen vorgesehen, anstatt für jede benötigte Verbindung einen separaten Inter-Kommunikator zu erstellen.

Zum Vergleich der Ansätze wurden Testreihen auf beiden Referenzsystemen durchgeführt. Die Ergebnisse der Tests zeigen, dass anhand der Zahlen keinem der Ansätze ein Vorrang eingeräumt werden kann. Allerdings können zunächst Ansatz 1 und 2 ausgeschlossen werden. Für Ansatz 1 ist das schwerwiegendste Ausschlusskriterium, dass die Funktionalität auf dem Hazel Hen nicht zur Verfügung steht. Des Weiteren ist das Verbindungslimit der Nameservers auf dem SuperMUC nicht akzeptabel. Zu Ansatz 2 stehen zwar keine Daten zur Verfügung, er sollte allerdings wegen zu hohem Entwicklungsaufwand und einer erwarteten Verkomplizierung des Programcodes ausgeschlossen werden. Die Ergebnisse zeigen eindeutig, dass das Dateisystem an einigen Stellen zu Problemen führen kann. Sie zeigen aber auch auf, dass mit Ansatz 3 die Verwendung der Verbindungen zum Problem führen kann. Daher ist keine eindeutige Empfehlung möglich, welcher Ansatz der bessere ist. Außerhalb der Zahlen gibt es allerdings noch weitere Argumente für Ansatz 3, wie die Vereinfachung des Programmcodes oder das andere bisher nicht verfügbare Verbandsoperationen genutzt werden können.

Ausblick

Zu den Herausforderungen in der Zukunft zählt sicher die Entscheidung ob und wenn ja wie der Ansatz 3 in PreCICE eingebaut werden soll. Dies erfordert dann wiederum weitere Untersuchungen über die Leistungsfähigkeit des Ansatzes. Um eventuelle Performanz-Probleme langfristig beseitigen zu können, sollten die entsprechenden Stellen der Referenzsysteme informiert werden.

A. Beispiel Ausführungskript für den SuperMUC

```
#@ energy_policy_tag = <policy tag>
#@ minimize_time_to_solution = yes
#@ wall_clock_limit = 00:05:00
#@ job_type = MPICH
#@ job_name = publisher_performance_test
#@ class = test
#@ island_count = 1
#@ network.MPI = sn_all,not_shared,us
#@ node = 2
#@ tasks_per_node = 16
#@ output = $(job_name).$(schedd_host).$(jobid).out
#@ error = $(job_name).$(schedd_host).$(jobid).err
#@ initialdir = $(home)/publisher_performance_test
#@ notification=always
#@ notify_user=mustermann@studi.informatik.uni-stuttgart.de
#@ queue
. /etc/profile
. /etc/profile.d/modules.sh
module unload mpi.ibm
module load mpi.intel/2017

export SUBJOB

cd left
L1=1
L2=16
sed -n -e "${L1},${L2}p" $LOADL_HOSTFILE >mfile
mpiexec -n 16 -f mfile ../TestProgramm A $COMMTYPE $NUMCONNS 10 100 $EXCHDIR $RESDIR &>
    out16 &
procA=$!

cd ../right
L1=17
L2=32
sed -n -e "${L1},${L2}p" $LOADL_HOSTFILE >mfile
mpiexec -n 16 -f mfile ../TestProgramm B $COMMTYPE $NUMCONNS 10 100 $EXCHDIR $RESDIR &>
    out16 &
procB=$!
cd ..
wait "$procA" "$procB"
```

B. Beispiel Ausführungskript für den Hazelhen

```
#PBS -N publisher_performance_test
#PBS -l nodes=2:ppn=24
#PBS -l walltime=0:05:00
#PBS -q test

# Domain for MPI beta implementation
DOMAIN=pub_perf_test

# Change to the directory that the job was submitted from
cd $PBS_O_WORKDIR

apmgr pdomain -uc $DOMAIN
cd left
aprun -n 24 -p $DOMAIN ../TestProgramm A $COMMTYPE $NUMCONNS 10 100 $EXCHDIR $RESDIR &>
    out24 &
cd ../right
aprun -n 24 -p $DOMAIN ../TestProgramm B $COMMTYPE $NUMCONNS 10 100 $EXCHDIR $RESDIR &>
    out24 &
wait
apmgr pdomain -ur $DOMAIN
```

Literaturverzeichnis

[15] *Mpi - a Message Passing Interface Standard V3.1. Ean 1114444410030.* High Performance Computing Centre Stuttgart, 2015. URL: https://fs.hlrs.de/projects/par/mpi//mpi31/ (zitiert auf S. 15, 26).

[BLG+16] H.-J. Bungartz, F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, B. Uekermann. „preCICE - A Fully Parallel Library for Multi-Physics Surface Coupling". In: *Computers and Fluids* 141 (2016), S. 250–258. ISSN: 00457930. DOI: 10.1016/j.compfluid.2016.04.003. URL: http://dx.doi.org/10.1016/j.compfluid.2016.04.003 (zitiert auf S. 28).

[BW01] A. Beckert, H. Wendland. „Multivariate interpolation for fluid-structure-interaction problems using radial basis functions". In: *Aerospace Science and Technology* 5.2 (2001), S. 125–134 (zitiert auf S. 28).

[Gat14] B. Gatzhammer. „Efficient and Flexible Partitioned Simulation of Fluid-Structure Interactions". PhD Thesis. Technische Universität München, 2014 (zitiert auf S. 28, 29).

[GFB+04] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, T. S. Woodall. „Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation". In: *Proceedings, 11th European PVM/MPI Users' Group Meeting*. Budapest, Hungary, Sep. 2004, S. 97–104 (zitiert auf S. 24).

[Shu15] A. K. Shukaev. „A Fully Parallel Process-to-Process Intercommunication Technique for preCICE". Magisterarb. 2015 (zitiert auf S. 13, 28, 31, 35).

[Uek16] B. W. Uekermann. „Partitioned Fluid-Structure Interaction on Massively Parallel Systems". Dissertation. München: Technische Universität München, 2016 (zitiert auf S. 28–31, 35).

Alle URLs wurden zuletzt am 20. 03. 2017 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift