Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Ausarbeitung

Seminar Effiziente Programmierung

MPI-Topologien

vorgelegt von

Irina Lindt

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen

Studiengang: Informatik
Matrikelnummer: 5957552
Betreuer: Hermann Lenhart

Hamburg, 15. März 2021

Inhaltsverzeichnis

1 Einführung

Wenn große Datenmengen für eine Berechnung verarbeitet werden, wie dies zum Beispiel in der Klimawissenschaft üblich ist, liegt es nahe, diese Aufgabe auf mehrere Prozesse zu verteilen. Das Message Passing Interface, im Folgenden MPI, bietet dafür eine umfangreiche Funktionalität.

Grundlage von MPI ist der Austausch von Nachrichten zwischen Prozessen. Wenn Prozesse für ihre Berechnung Daten von einem anderen Prozess brauchen, können sie diese austauschen, ohne dafür die Berechnung unterbrechen zu müssen.

Standardmäßig werden dabei Nachrichten entweder an alle dem aktuellen Prozess bekannten Prozesse geschickt, oder es wird vor jedem Sendevorgang berechnet, mit welchen Prozessen kommuniziert werden soll.

Diese Berechnung kann komplex ausfallen und die Lesbarkeit des Code erschweren. Dabei haben Prozesse oft nur wenige feste Kommunikationspartner. Der Gegenstand dieser Arbeit sind Mechanismen von MPI, die die Zuweisung dieser Kommunikationspartner erleichtern. Diese heißen in MPI Topologien. Da sie auf anderen Mechanismen wie Groups und Communicator aufbauen, werden diese zuerst eingeführt. Anschließend werden Topologien behandelt, wobei unter anderem auf Leistungsaspekte eingegangen wird.

2 Communicator

2.1 Motivation

Parallele Programmierung erfordert oft die Aufteilung von Prozessen auf mehrere Gruppen, um jeder Gruppe eine eigene Aufgabe zuzuweisen. [SO96] listet folgende Gründe dafür, Prozessgruppen zu verwenden:

- Die Zuweisung unterschiedlicher Aufgaben. Zum Beispiel kann erwünscht sein, dass 2/3 der Prozesse mit der Verarbeitung von Daten beschäftigt ist, während 1/3 parallel neue Daten einliest
- Die Bestimmung der Prozesszugehörigkeit soll nicht bei jedem Sendevorgang passieren, um Rechenschritte zu sparen
- Die Lesbarkeit wird erhöht, da Kommunikationsgruppen klar benannt werden können

MPI definiert zwei Arten von Communicatorn, Inter- und Intra-Communicator. Dabei werden Intra-Communicator für die Kommunikation innerhalb der Gruppe verwendet, und Inter-Communicator werden für die Kommunikation zwischen zwei unterschiedlichen Gruppen verwendet. Topologien sind nur auf Intra-Communicatorn definiert, daher sind nur diese Gegenstand dieser Arbeit. Überall, wo von einem Communicator die Rede ist, ist ein Intra-Communicator gemeint.

2.2 Groups

Eine Group, oder Gruppe in MPI ist eine geordnete Menge von Prozessen, in der jedem Prozess bei seiner Erzeugung eine ganzzahliger Rang zugeordnet wird. Der erste Prozess einer Gruppe hat den Rang 0, und er wird fortlaufend inkrementiert. Ein Communicator ist eine Gruppe von Prozessen, die zusätzlich mit einem Kontext versehen ist, aus dem sich seine Verwendung ergibt.

2.3 Vordefinierte Communicator

MPI definiert einige Communicator, die man direkt verwenden kann, ohne sie zuerst erzeugen zu müssen.

- MPI_COMM_WORLD ist der initiale Communicator, der jedem Prozess nach seiner Initialisierung zur Verfügung steht. Mit MPI_COMM_WORLD kann man auf alle Prozesse zugreifen, mit denen der aufrufende Prozess kommunizieren kann, inklusive seiner selbst.
- MPI_COMM_NULL ist eine vordefinierte Konstante, die als Fehlerwert für ungültige Communicator verwendet wird.
- MPI_COMM_SELF ist ein Communicator, der nur den aufrufenden Prozess selbst enthält.

2.4 Anwendungsbeispiel

Abbildung 1 demonstriert, wie in Listing 1 mit Hilfe von Communicatorn Prozesse in Untergruppen aufgeteilt werden können. Hierzu wird die Funktion MPI_Comm_Split verwendet. Diese nimmt einen Communicator entgegen und weist jedem Prozess in diesem eine neue Gruppe, hier als Farbe bezeichnet, zu. Prozesse mit der gleichen Farbe bilden jeweils ihren eigenen Communicator. Innerhalb jeder neuen Untergruppe fängt die Zählung des Prozessrangs wieder mit 0 an. In diesem Beispiel wurde für die Berechnung der Farbe der Rang des aufrufenden Prozesses ganzzahlg durch durch 4 geteilt, wodurch sich aufgrund der Rundung vier Farben und somit vier Prozessgruppen ergeben.

```
1  // Get the rank and size in the original communicator
2  int world_rank, world_size;
3  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
4  MPI_Comm_size(MPI_COMM_WORLD, &world_size);
5
6  int color = world_rank / 4; // Determine color based on row
7
8  // Split the communicator based on the color and use the
9  // original rank for ordering
10 MPI_Comm row_comm;
11 MPI_Comm_split(MPI_COMM_WORLD, color, world_rank, &row_comm);
```

Listing 1: Teilung eines Communicators in vier kleinere. Aus [Bla]

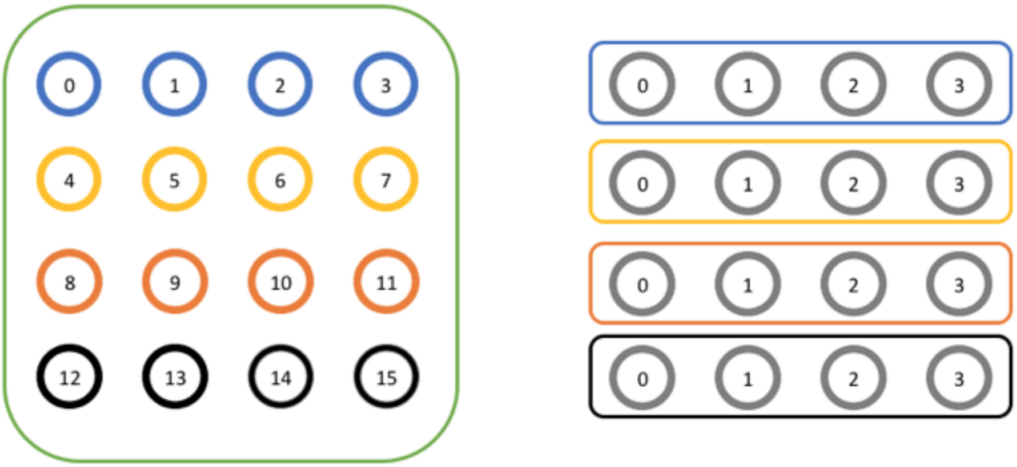

Abbildung 1: Ein Communicator wird auf vier Communicator aufgeteilt. Aus [Bla]

3 Topologien in MPI

Topologien sind ein zusätzlicher Mechanismus, den man einem Intra-Communicator zuweisen kann. Damit wird für jeden Prozess festgelegt, welche Prozesse seine Kommunikationspartner sind. Oft werden diese Kommunikationspartner auch als Nachbarn bezeichnet. Somit können Sendemethoden wie MPI_Send gezielt nur mit den Nachbarn eines Prozesses kommunizieren.

Es gibt zwei Kategorien von Topologien in MPI: kartesische Topologien und Graphtopologien. Kartesische Topologien werden verwendet, wenn die Anordnung der Nachbarn eines Prozesses entsprechend einem Gitter auf den kartesischen Koordinaten dargestellt werden kann. Graphtopologien, also Topologien die mit einem Graphen darstellbar sind, sind allgemeiner, und werden immer dann verwendet, wenn kartesische Koordinaten nicht verwendet werden können, zum Beispiel weil einige Prozesse diagonale Kommunikationspartner haben. Relevant sind insbesondere kartesische Koordinaten.

3.1 Kartesische Topologie

Ein Beispiel für kartesische Topologien ist der Torus Interconnect. Dabei kommuniziert jeder Prozess mit allen seinen Nachbarn, so dass sich in alle Richtungen ein Kreis bildet. Abbildung 2 zeigt ein Beispiel, wie die Anordnung der Prozesse für die Dimensionen 1-3 aussieht.

Der Torus Interconnect hat eine besondere Relevanz für Hochleistungrechnen mit MPI, denn diese Topologie wird bei mehreren Supercomputern als interne Netzwerktopologie verwendet; so hat der IBM-Supercomputer BlueGene/Q eine 5D-Torus-Topologie. [McC14]

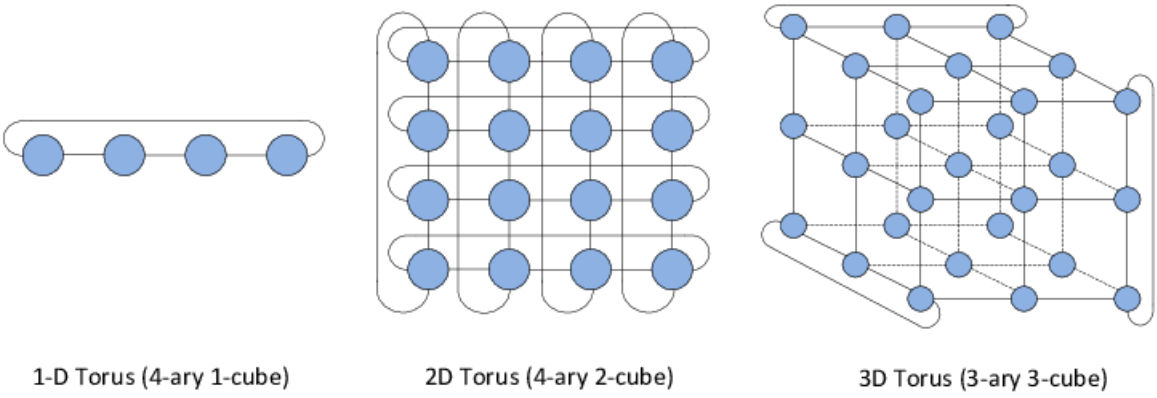

Abbildung 2: Ein Torus in einer, zwei und drei Dimensionen. Aus [Wan15]

3.2 Leistungsaspekte

Die von MPI definierte Topologie ist virtuell, das heißt, sie bildet nicht automatisch die Rechnerarchtektur, auf der die Software ausgeführt wird, ab. Dennoch kann bei kluger Wahl der Software-Topologie und ihrer Anpassung an die Rechnerarchitektur ein Leistungsgewinn erzielt werden. In [RG11] wurde untersucht, welchen Leistungsvorteil man bei typischen MPI-Aufrufen wie MPI_Cart_Create herausholen kann, wenn man die virtuelle Topologie an die physikalische Topologie anpasst. Wie man in Abbilding 3 sehen kann, kann man in einem solchen Optimalfall bis zu 60% schneller rechnen.

3.3 Funktionsaufruf für MPI_Cart_Create

Der Funktionsaufruf MPI_Cart_Create, mit dem man in MPI eine kartesische Topologie erzeugen kann, ist in Listing 2 zu sehen. Seine Parameter sind der Communicator old_comm, auf dem die Topologie definiert wird, die Anzahl der Dimensionen ndims, ein Array dims, in dem für jede Dimension die Anzahl der Elemente gespeichert wird, ein Array periods, in dem für jede Dimension gespeichert wird, ob diese periodisch (1) ist oder nicht (0), ein logischer Wrt reorder, der angibt, ob MPI die Prozesse in der bereits erzeugten Topologie neu anordnen kann (1) oder nicht (0), und als Ausgabeparameter der dabei erzeugte neue Communicator, auf dem nun eine kartesische Topologie definiert ist.

```
1 int MPI_Cart_create(MPI_Comm old_comm,
2                      int ndims,
3                      const int* dims,
4                      const int* periods,
5                      int reorder,
6                      MPI_Comm* new_comm);
```

Listing 2: Funktionsaufruf von MPI_Cart_Create. Aus [For]

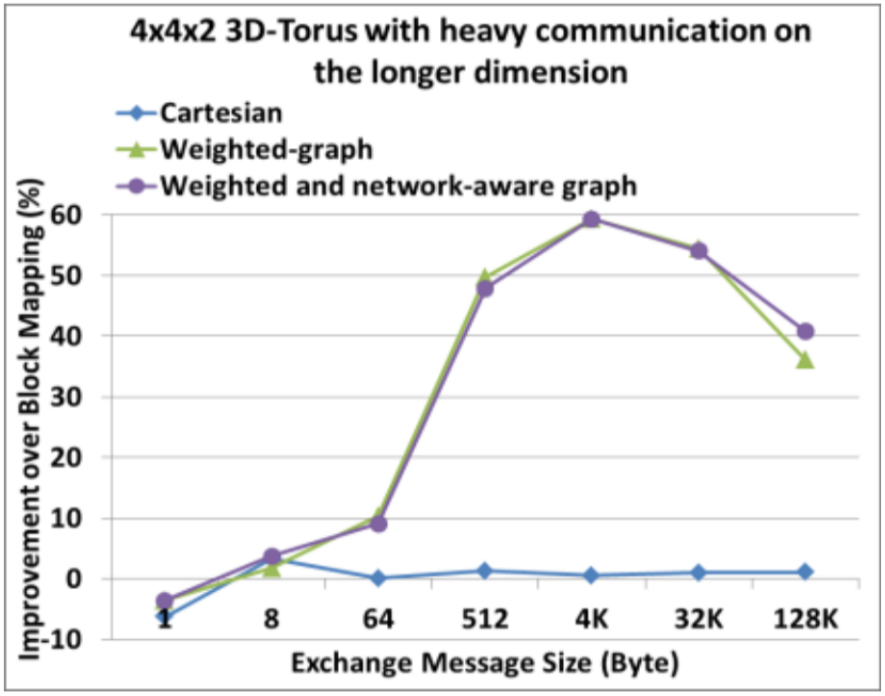

Abbildung 3: Leistungsverbesserung bei Optimierung der Topologie. Aus [RG11]

3.4 Graphtopologien

Graphtopologien kommen dann zur Anwendung, wenn Prozesse zum Beispiel diagonale Kommunikationspartner haben, und bei allen Strukturen, die sich durch kartesische Topologien nicht abbilden lassen. Listing 3 zeigt, wie man mit MPI_Graph_Create eine neue Graphtopologie definieren kann. Als Parameter nimmt diese Funktion den alten Communicator comm_old entgegen, die Anzahl der Knoten nnodes, ein Array index, das die Knotengrade beschreibt, ein Array edges, das die Kanten speichert, den logischen Wert reorder, der angibt, ob MPI die Prozesse nach der Erzeugung umordnen kann, und als Ausgabeparameter den neuen Communicator comm_graph, auf dem eine Graphtopologie definiert wurde. Für verteilte Graphtopologien stellt MPI zudem weitere Aufrufe wie MPI_Dist_Graph_Create und MPI_Dist_Graph_Create_Adjacent zur Verfügung.

3.5 Funktionsaufruf für MPI_Graph_Create

```
1 int MPI_Graph_create(MPI_Comm comm_old,
2                       int nnodes,
3                       const int index,
4                       const int edges,
5                       int reorder,
6                       MPI_Comm *comm_graph)
```

Listing 3: Funktionsaufruf von MPI_Graph_Create. Aus [For]

4 Zusammenfassung

Mit Topologien bietet MPI einen mächtigen Mechanismus, um Kommunikationsstrukturen in Software abzubilden. Es setzt dabei auf vorhandene Strukturen von Communicator und Gruppe, und verleiht diesen einen zusätzlichen Kontext, so dass jeder Prozess eine wohldefinierte Anzahl mit Kommunikationspartnern bekommt. Dies erlaubt, einen saubereren und lesbaren Code zu schreiben. Sofern die virtuelle Topologie an die Rechnerarchitektur angepasst wird, kann zudem eine Leistungssteigerung erzielt werden.

Literatur

[Bla] Wesley Bland. *Introduction to Groups and Communicators*. URL: https://mpitutorial.com/tutorials/introduction-to-groups-and-communicators/.

[For] The Message-Passing Interface Forum. *MPI: A Message-Passing Interface Standard*. URL: https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf.

[McC14] Collin McCarthy. „Visualizing the Five-dimensional Torus Network of the IBM Blue Gene/Q“. In: *Proceedings of the First Workshop on Visual Performance Analysis (VPA14)* (2014). DOI: 10.1109/VPA.2014.10.

[RG11] Mohammad Rashti und Jonathan Green. „Multi-core and Network Aware MPI Topology Functions“. In: *Proceedings of the 18th European MPI Users' Group conference on Recent advances in the message passing interface* (2011). DOI: 10.1007/978-3-642-24449-0_8.

[SO96] Marc Snir und Steve Otto. *MPI: The Complete Reference*. Cambridge, Massachusetts: The MIT Press, 1996.

[Wan15] Ting Wang. „NovaCube: A low latency Torus-based network architecture for data centers“. In: *2014 IEEE Global Communications Conference* (2015). DOI: 10.1109/GLOCOM.2014.7037143.