Neuigkeiten in parallelen Programmiermodellen: was ist noch vermittelbar?

Dr. Christian Terboven, Julian Miller
{terboven, miller}@itc.rwth-aachen.de
Hochleistungsrechnen, RWTH Aachen University

HPC-Statuskonferenz der Gauß-Allianz
01. Oktober 2020

i12 High Performance Computing | RWTH AACHEN UNIVERSITY

Entwicklung von OpenMP und MPI

OpenMP™

Jahr	Version	Kommentar
1997	1.0	FORTRAN
1998	1.0	C/C++
1999	1.1	FORTRAN
2000	2.0	FORTRAN
2002	2.0	C/C++
2005	2.5	C/C++ and FORTRAN
2008	3.0	Tasking
2011	3.1	Verschiedenes
2013	4.0	Accelerators
2015	4.5	Verschiedenes.
2018	5.0	Tools
2020	5.1	(erwartet)

MPI Forum

Jahr	Version	Kommentar
1994	1.0	P2P, Collectives, …
1995	1.1	Verschiedenes
1997	1.2	-
2008	1.3	Integration von 1.1 und 1.2
1997	2.0	One-sided, Parallel I/O
2008	2.1	Integration …
2009	2.2	Topologies, C++ deprec.
2012	3.0	Mprobe, Misc. Revisions
2015	3.1	Verschiedenes
2020/21	4.0	(erwartet)

CC BY SA
i12 High Performance Computing
RWTH AACHEN UNIVERSITY

Ausgewählte Neuigkeiten in OpenMP

- Masked construct in OpenMP 5.1 & 6.0
- Assumes directive in OpenMP 5.1

CC BY SA

Masked construct in OpenMP 5.1 & 6.0

- Ersatz des `master` Konstrukt mit erweiterter Funktionalität
  - Das `master` Konstrukt wird mit OpenMP 5.1 als deprecated deklariert

    - Konstrukt

```
#pragma omp master
{}
#pragma omp masked
{}
#pragma omp masked filter(thread_id)
{}
```

    - Äquivalent

```
if(omp_get_thread_num()==0)
{}
if(omp_get_thread_num()==0)
{}
if(omp_get_thread_num()==thread_id)
{}
```

- OpenMP 6.0 wird die Funktionalität des Filtern erweitern (mit Blick auf die thread id)

```
#pragma omp parallel
{
  // … define odd to be a thread-set with omp_get_thread_num()%2==1
  #pragma omp for filter(thread-set: odd)
  {}
}
```


CC BY SA

Assumes directive in OpenMP 5.1

- Drückt Invarianten aus die durch die Implementierung für Optimierungen genutzt werden können
  - Invarianten müssen bei der Programmausführung gültig sein, andernfalls ist das Verhalten undefiniert

```
#pragma omp assumes clause[ [ [,] clause] ... ] new-line
```

  - absent: Garantie, dass kein entsprechendes Konstrukt erreicht wird
  - contains: Hinweis, dass wahrscheinlich kein entsprechendes Konstrukt erreicht wird
  - holds: skalarer Ausdruck der im Bereich der Direktive gilt
  - no_openmp: Garantie, dass kein OpenMP-bezogener Code ausgeführt wird
  - no_openmp_routines: Garantie, dass keine OpenMP API Routines aufgerufen werden
  - no_parallelism: Garantie, dass keine OpenMP (implicit oder explicit) Tasks generiert werden und dass keine SIMD Konstrukte ausgeführt werden

- Beispiel:

```
#pragma omp assumes absent(task, taskloop)
```


CC BY SA
High Performance Computing
RWTH AACHEN UNIVERSITY

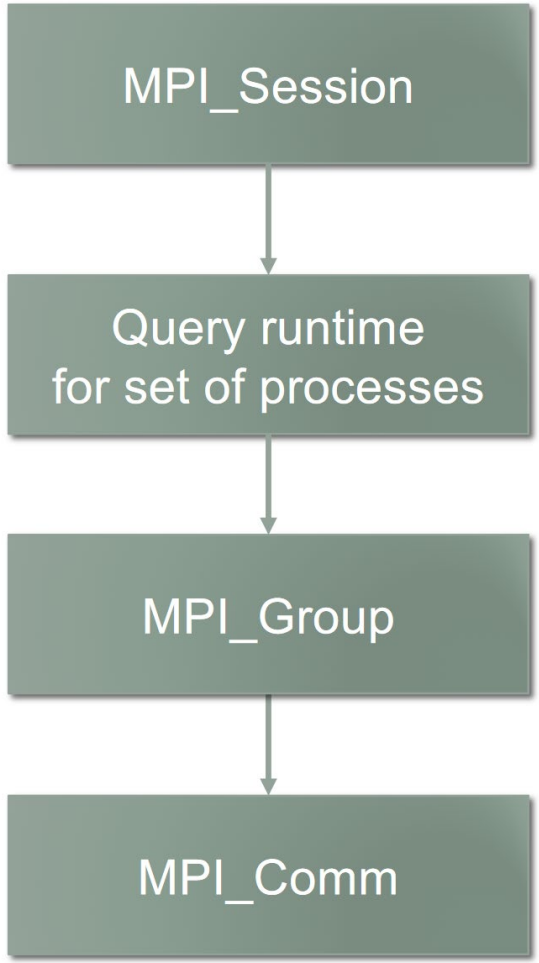
Ausgewählte Neuigkeiten in MPI 4.0

- Initialization schemes: World and Sessions Model
- Partitioned Communication

CC BY SA

Initialization schemes: World and Sessions Model

- Begrenzungen des “World Model”
  - MPI_COMM_WORLD ist “teuer” anzulegen und wird ggfs. gar nicht von der Anwendung verwendet
    - Schwierig für Bibliotheken herauszufinden ob MPI bereits initialisiert wurde
    - MPI kann nicht neu initialisiert werden
- Sessions Model
  - Bibliotheken / Komponenten initialisieren ihre eigenen Sessions (MPI_SESSION_INIT/FINALIZE)
  - Keine Erstellung vordefinierter Kommunikatoren
  - Thread-Unterstützung pro Session
  - Isolation der Kommunikation in untersch. Sessions
- Beide Modelle können in einer Anwendung eingesetzt werden
  - Erlaubt inkrementelle Portierung von Anwendungen und Bibliotheken



CC BY SA
High Performance Computing
RWTH AACHEN UNIVERSITY

Partitioned Communication

- Konzept: Mehrere Aktoren (z.B. Threads) tragen zu einer größeren Operation in MPI bei
  - Keine neuen Ranks, keine Identifikation von Threads => Threads bleiben anonym
  - Zusätzlich: Unterstützung für NIC Offload Fähigkeiten

- Erweiterung der persistenten P2P Kommunikation
  - Ankündigung mit MPI_PSEND_INIT
  - Start mit MPI_START
  - Beiträge mit MPI_PREADY
  - Fertigstellung mit MPI_WAIT/MPI_TEST

- Kann in Kombination mit Tasking genutzt werden
  - Partitionen als Input-Abhängigkeiten für Tasks beim Empfänger
  - Partitionen als Output-Abhängigkeiten für Tasks beim Sender

CC BY SA
i12 High Performance Computing
RWTH AACHEN UNIVERSITY

Untersuchung des Lernerfolges

OpenMP Softwareprojektpraktikum

- **Aufgabe**: Parallelisierung von drei Algorithmen mit OpenMP
  - 1.: Dünnbesetzte Matrix-Vektormultiplikation
  - 2.: Paralleler Mergesort
  - 3.: k-Means-Algorithmus

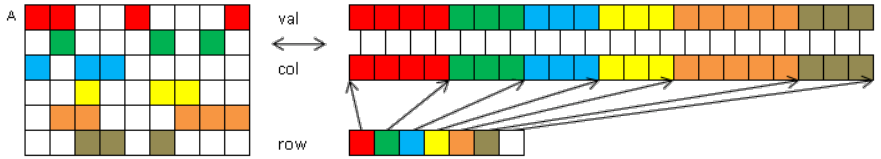
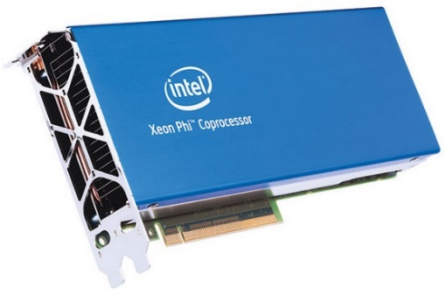


Illustration of the compressed row storage format.

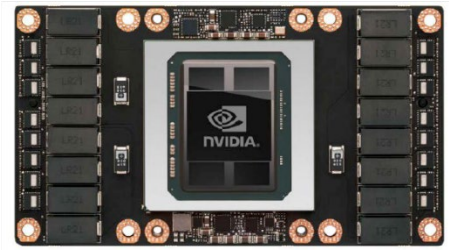
- **Zielsysteme**: Intel Xeon Phi und NVIDIA GPUs

- **Wettbewerbe**: Schnellste Ausführung der drei Programme auf beiden Architekturen

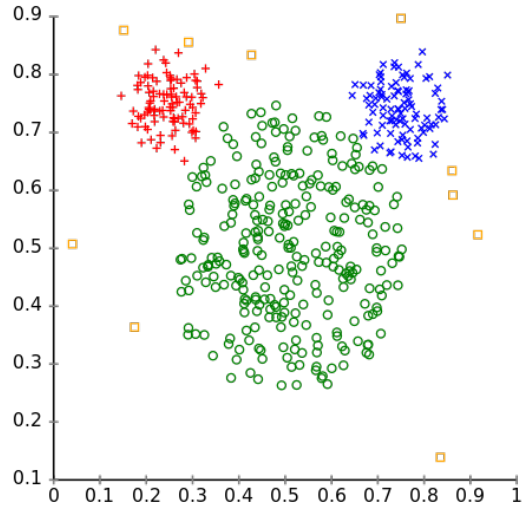

CLAIX 2018, RWTH Aachen University



Source: Intel



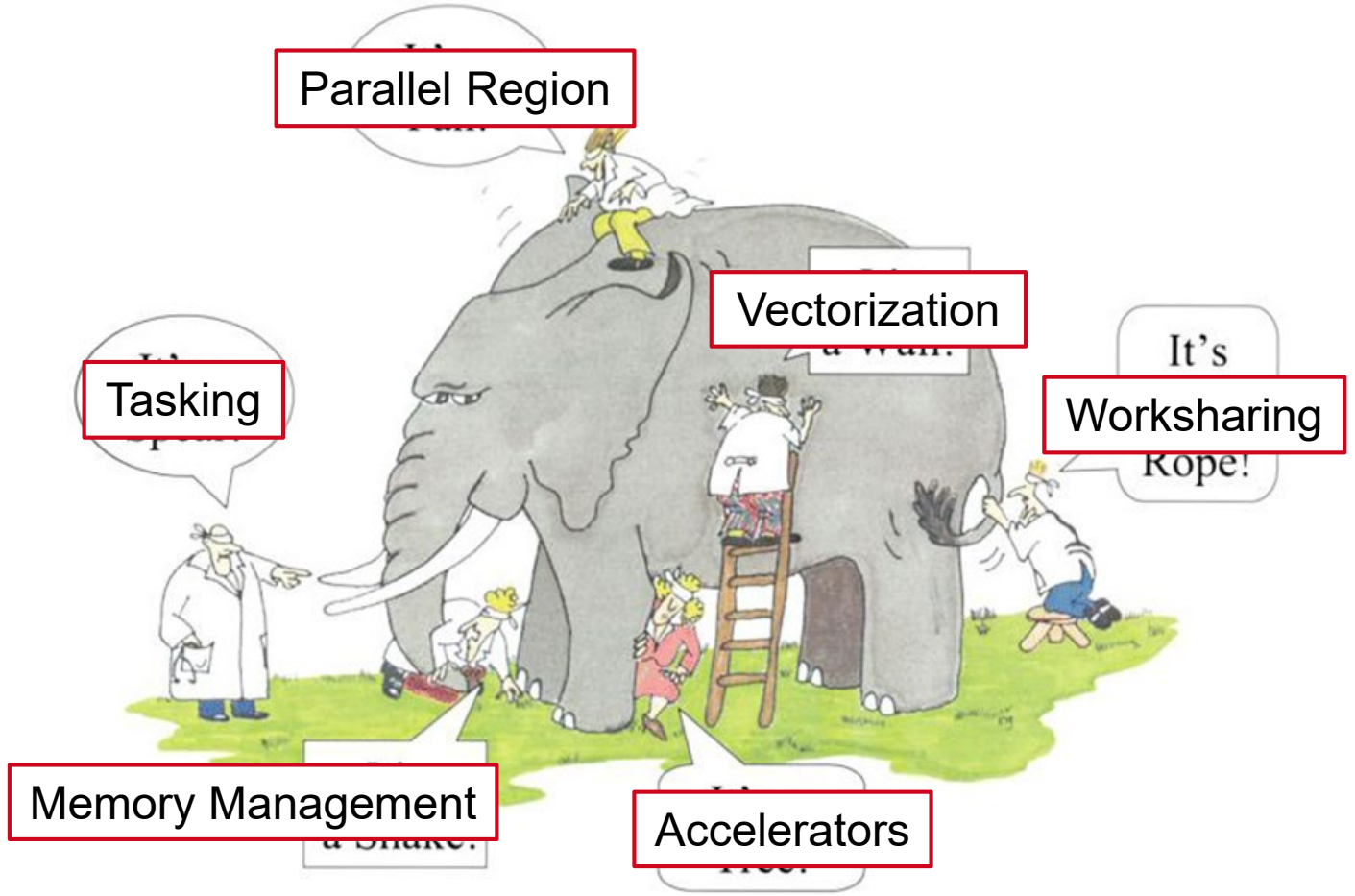
Source: NVIDIA



K-means clustering on an artificial dataset. Source: Wikipedia

Welche Themen können wir Studierenden in einem Semester beibringen?

- Fokus auf Kernkonzepte von OpenMP
- Konzepte werden im Laufe des Semesters eingefügt und angewandt
- Allgemein: Parallel Region, Vectorization, Memory Management
- Aufgabe 1: Worksharing
- Aufgabe 2: Tasking
- Aufgabe 3: Accelerators



Welche Themen können wir Studierenden in einem Semester beibringen?

- Alle Studierende haben eine korrekte Lösung eingereicht
- Kernkonzepte wurden von allen Studierenden erfolgreich angewandt
- Größere Unterschiede in der tatsächlichen Umsetzung und Detailoptimierungen

	Anzahl Studierende	Durchschnittliches Semester
2017	18	5.4
2018	17	4.4
2019	16	4.8

Anzahl der Studierende im OpenMP Praktikum und ihre durchschnittliche Studienzeit.

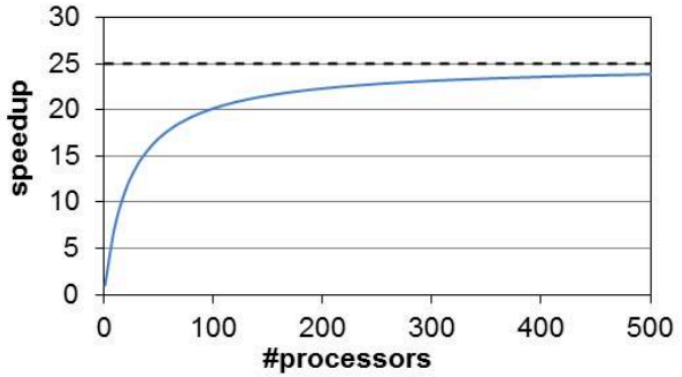
Wie erfolgreich ist dies?

- **Erfolg**: Erreichen der gesetzten Lernziele im Praktikum
- **Evaluation**: Wissensumfrage über Konfidenzbewertungen
- **Ergebnisse**:

	Vorher	Nachher
2017	2.0	(zu wenige Rückläufer)
2018	1.95	2.4
2019	1.2	2.4

Median der Wissensumfragen im OpenMP Praktikum. Eine Bewertung von 1 bedeutet nicht beantwortbar und 3 die Fähigkeit auf Prüfungsniveau zu antworten.

10. The following figure represents the speedup *Sp* behavior of a parallel program applying Amdahl's Law. Read the needed values from the figure and compute the parallel portion *p* of the program.



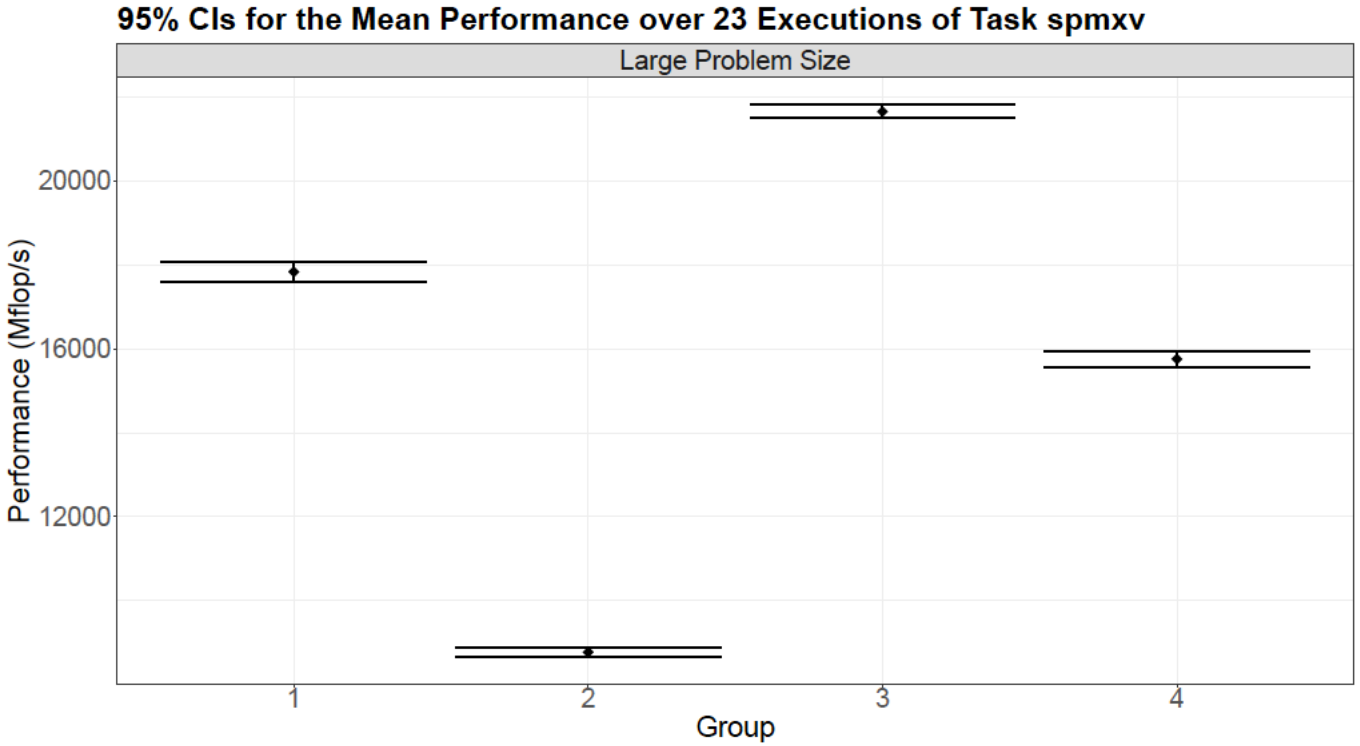
Rate your confidence!

- A
- B
- C

Beispielhafte Wissensumfrage.

Gilt das für alle Studierenden gleichermaßen?

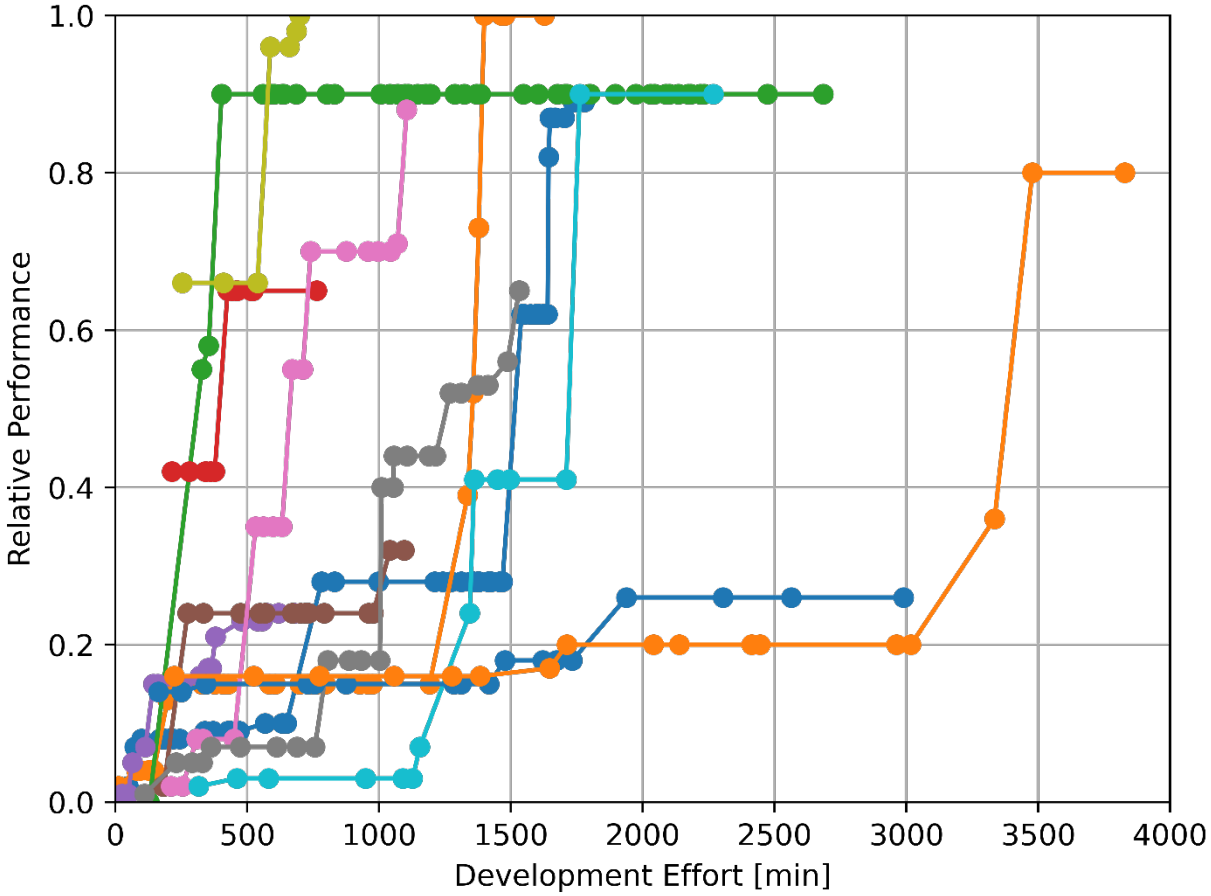
- Nein, sowohl die erreichte Leistung als auch die Noten schwanken
- Einzelne Gruppen sind unterschiedlich stark in den drei Aufgaben
- Wettbewerb fördert Motivation



Erreichter Durchsatz in Gleitkommaoperationen pro Sekunde für die dünnbesetzte Matrix-Vektormultiplikation im OpenMP Praktikum 2019.

Wie hängt der Erfolg von Lern- und Arbeitsaufwand ab?

- Leistungszuwachs häufig sprungartig
  - Lange Entwicklungszeiten ohne Leistungszuwachs
- Krux ist Parallelität oder Optimierungspotential zu erkennen
  - Benötigt viel Erfahrung



Erreichter Leistung relativ zur Referenzlösung über die benötigte Entwicklungszeit je Gruppe im OpenMP Praktikum 2019.

Zusammenfassung

CC BY SA

Zusammenfassung

- Parallele Programmiermodelle werden aktiv weiterentwickelt
  - Forschung führt zu neuen Vorschlägen und Implementierungen
  - Begleitet durch Forschung zur Programmierproduktivität
- Kernkonzepte der Parallelprogrammierung sind innerhalb eines Semesters vermittelbar
- Individuelle Lösungen stark unterschiedlich in der Leistung
  - Ausnutzung von Architekturbesonderheiten (weiterhin) eine Herausforderung
  - Anreizsysteme motivieren Studierende erfolgreich
- Benötigten Aufwände hängen stark von der individuellen Erfahrung ab
  - Parallelität und Optimierungspotential muss erkannt werden

Vielen Dank für Ihre Aufmerksamkeit.

CC BY SA