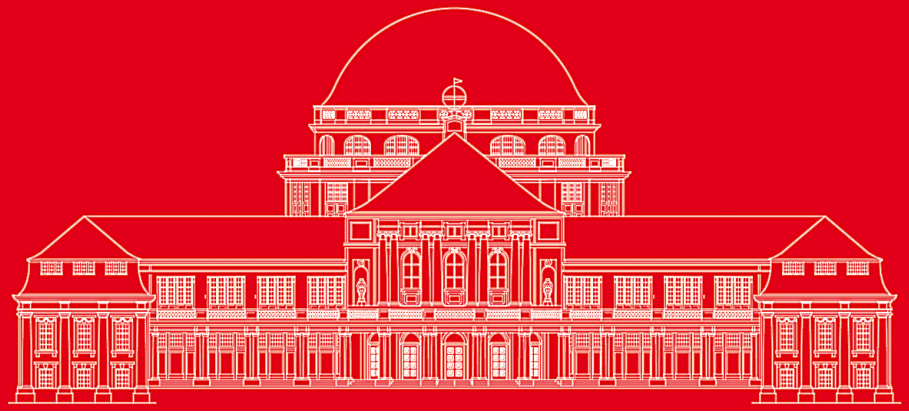
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Praktikum: Paralleles Programmieren für Geowissenschaftler

Prof. Thomas Ludwig, Hermann Lenhart & Tim Jammer



Dr. Hermann-J. Lenhart
hermann.lenhart@informatik.uni-hamburg.de

OpenMP Einführung II:

- Parallele Konstrukte
- Clauses
- Synchronisation
- Reduction

> Leistungsmessung

> Leistungsmessung auf dem Cluster

OpenMP – Parallele Konstrukte I

Parallel Konstrukt:

!$omp parallel [clausel 1, …,clausel n] -> siehe Quick reference

Parallele Region

!$ omp end parallel

Innerhalb der vom "!$omp parallel" Konstrukt aufgespannten parallelen Region werden Konstrukte für die Verteilung auf die Threads definiert, z.B.

!$omp do	Iteration in Schleifen
!$omp sections	unabhängige Arbeitseinheiten
!$omp workshare	Parallelisiert Array Syntax

OpenMP – Parallele Konstrukte II

Parallel Konstrukt kombiniert mit Section:

```
!$omp parallel                          =>  Beginn Parallele Region
    !$omp sections
        !$omp section
                call subroutine A        - strukturierter Block A

        !$omp section
                call subroutine B        - strukturierter Block B
    !$omp end sections                   - Ende Sections Blöcke A+B
!$ omp end parallel                     => Ende parallele Region
```

! Keine Annahme über Reihenfolge

Load-Balance Probleme können auftauchen!

OpenMP – Parallele Konstrukte III

Combined Konstrukt kombiniert mit Workshare: **(nur in FORTRAN!)**

```
!$omp parallel  workshare shared (n,a,b,c)
        a(1:n) = a(1:n) + 1
        b(1:n) = b(1:n) + 2
        c(1:n) = c(1:n) + 3
!$ omp end parallel  workshare
```

! Es wird nicht spezifiziert wie die Arbeitseinheiten auf die Threads zugeteilt werden!

! User muss für Parallelität in den Daten sorgen,
d.h. es darf keine versteckten Abhängigkeiten geben!

OpenMP – Clause III

SCHEDULE nur für Loops anzuwenden

Syntax: !$omp do schedule(kind[,chunk_size])

kind:

static die direkteste Zuordnung mit dem wenigsten Overhead

Iterationen werden in Portionen der Größe *chunk_size* aufgeteilt

ohne Angabe von *chunk_size* wird der Iterationsraum gleichmäßig auf die Threads aufteilt

OpenMP – Clause IV

SCHEDULE nur für Loops anzuwenden

Syntax: !$omp do schedule(kind[,chunk_size])

kind:

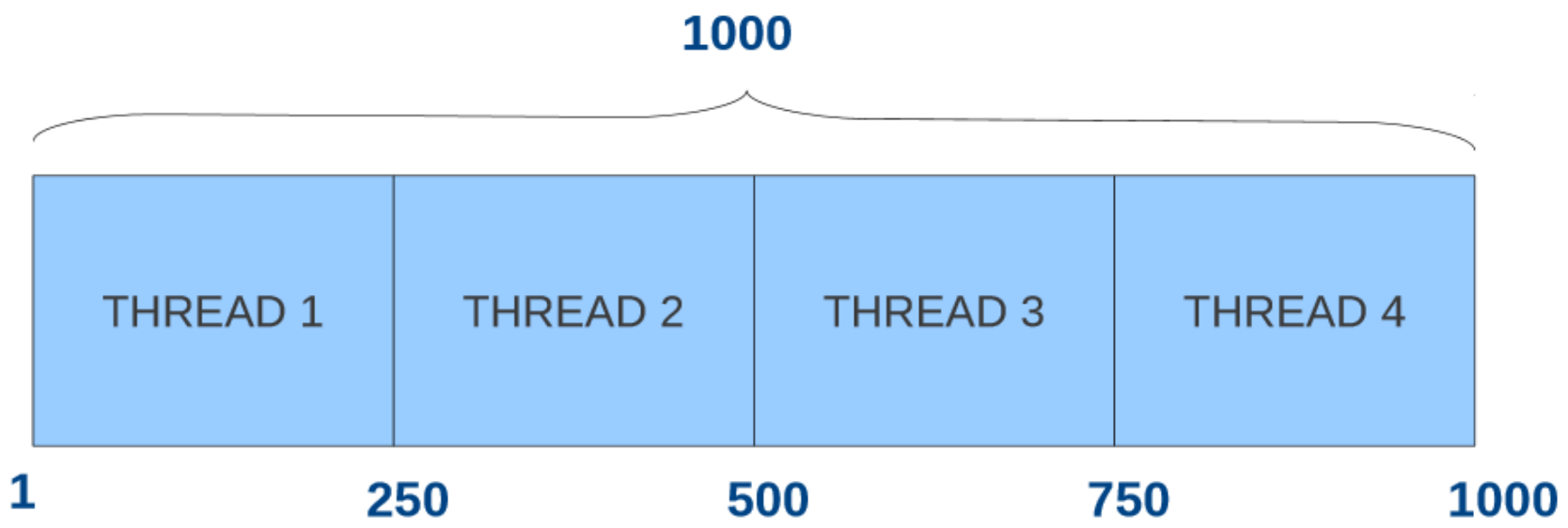
dynamic Iterationen werden nach der Verfügbarkeit der Threads zugewiesen. Jeder freie Thread bekommt einen Chunk zugewiesen bis alle Iterationen abgearbeitet sind.

guided wie dynamic, nur dass die Chunks der noch zu bearbeitenden Iterationen immer kleiner werden.

How OMP schedules iterations?

Although the OpenMP standard does not specify how a loop should be partitioned most compilers split the loop in N/p (N #iterations, p #threads) chunks by default. This is called a static schedule (with chunk size N/p)

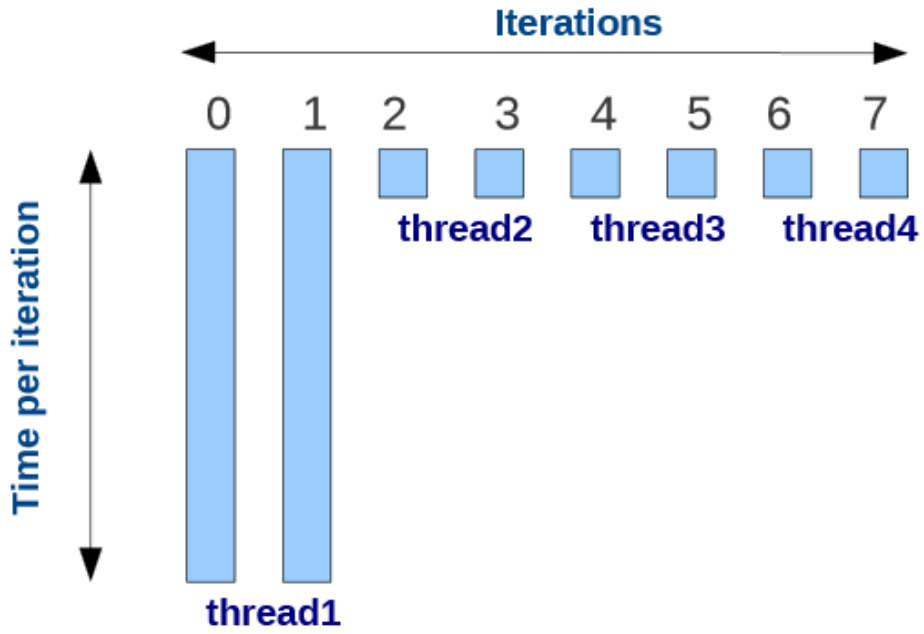
For example, suppose we have a loop with 1000 iterations and 4 omp threads. The loop is partitioned as follows:



Quelle: http://people.math.umass.edu/~johnston/PHI_WG_2014/OpenMPSlides_tamu_sc.pdf

Issues with Static schedule

With static scheduling the number of iterations is evenly distributed among all openmp threads (i.e. Every thread will be assigned similar number of iterations). This is not always the best way to partition. Why is This?



This is called load imbalance. In this case threads 2,3, and 4 will be waiting very long for thread 1 to finish

Quelle: http://people.math.umass.edu/~johnston/PHI_WG_2014/OpenMPSlides_tamu_sc.pdf

Dynamic Schedule

With a dynamic schedule new chunks are assigned to threads when they come available. OpenMP provides two dynamic schedules:

- $!OMP DO SCHEDULE(DYNAMIC,n) // n is chunk size
  Loop iterations are divided into pieces of size chunk. When a thread finishes one chunk, it is dynamically assigned another.
- $!OMP DO SCHEDULE(GUIDED,n) // n is chunk size
  Similar to DYNAMIC but chunk size is relative to number of iterations left.

Quelle: http://people.math.umass.edu/~johnston/PHI_WG_2014/OpenMPSlides_tamu_sc.pdf

Dynamic Schedule

With a dynamic schedule new chunks are assigned to threads when they come available. OpenMP provides two dynamic schedules:

- $!OMP DO SCHEDULE(DYNAMIC,n) // n is chunk size
  Loop iterations are divided into pieces of size chunk. When a thread finishes one chunk, it is dynamically assigned another.
- $!OMP DO SCHEDULE(GUIDED,n) // n is chunk size
  Similar to DYNAMIC but chunk size is relative to number of iterations left.

Keep in mind: although Dynamic scheduling might be the prefered choice to prevent load inbalance in some situations, there is a significant overhead involved compared to static scheduling.

Quelle: http://people.math.umass.edu/~johnston/PHI_WG_2014/OpenMPSlides_tamu_sc.pdf

OpenMP – Clause - Beispiel Programm

```
!$omp parallel do default(none) &
!$omp shared(dataOld,data) private(i,j)
      do i=2,ubound(data,1)-1
         do j=2,ubound(data,2)-1
            data(i,j) = Berechnung Star
         end do
       end do
!$omp end parallel do
```

OpenMP – Clause – Beispiel - Makefile

Im makefile: für 10 Threads

Static mit Chunk-Size 4:

export omp_schedule=static,4 OMP_NUM_THREADS=10

Dynamic mit Chunk-Size 4

export omp_schedule=dynamic,4 OMP_NUM_THREADS=10

Guided:

export omp_schedule=guided OMP_NUM_THREADS=10

Im Programm:

!$omp parallel do default(none) &

!$omp shared(dataOld,data) private(i,j)

……

!$omp end parallel do

OpenMP – Reduction I

Die **REDUCTION CLAUSE** wird von OpenMP bereitgestellt um wiederkehrende Berechnungen, z.B. Summationen, einfach durchzuführen.

Syntax: reduction ({operator I intrinsic_procedure_name} :list)

```
!$omp parallel do reduction(+:sum)
    do i = 1, n
    sum = sum + a(i)
    enddo
```

Sorgt aber intern auch für spezifische Zugriffsrechte.

Dazu folgendes Beispiel.

OpenMP – Reduction II

Problemstellung:

```
!$omp  parallel do
  do i = 1,1000
    a = a  + i
  end do
!$omp end parallel do
```

a ist „shared“ da alle Threads darauf zugreifen müssen

aber nur ein Thread soll zu einem gegebenen Zeitpunkt schreiben bzw. a updaten können

sonst würde ein undefinierbares Ergebnis erfolgen

Quelle: Miguel Hermanns

OpenMP – Reduction III

Problemstellung:

```
!$omp  parallel do        reduction(+:a)

  do j = 1,1000
                                nur ein Thread pro Zeitpunkt darf  a verändern
      a = a  + i

   end do

!$omp end parallel do
```

Quelle: Miguel Hermanns

OpenMP – Reduction II

Für die **REDUCTION CLAUSE** stehen folgende Operatoren und Initialwerte bereit:

Operator	Initialwert
+ / -	0
*	1
.and. / .eqv.	.true.
.or. / .neqv.	.false.

Intrinsische Funktion	Initialwert
max	kleinste Zahl in der reduction Elemente Liste
min	größte Zahl in der reduction Elemente Liste

OpenMP – Synchronisation

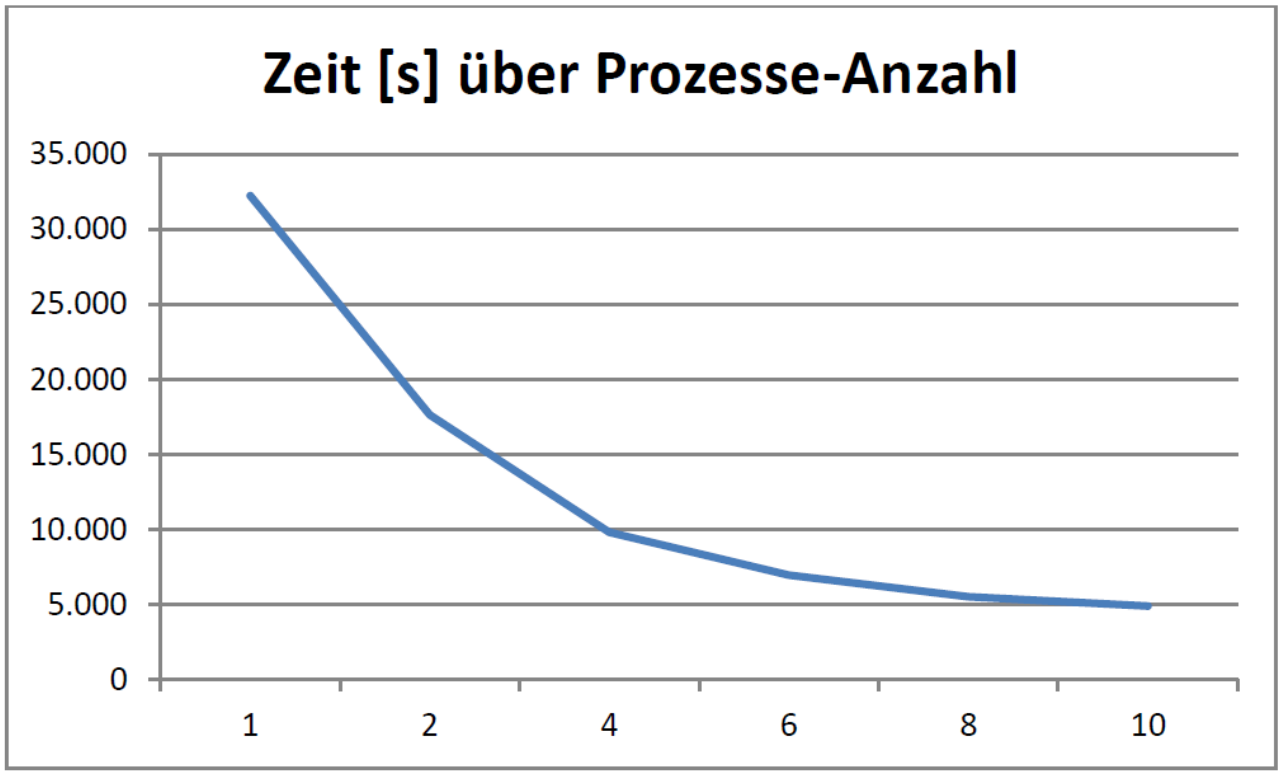
BARRIER sind Synchronisationspunkte bei denen die einzelnen Threads aufeinander warten. Keinem Thread wird erlaubt im Programm fortzufahren, bis alle anderen Threads ebenfalls diesen Programmpunkt erreicht haben.

Syntax: !$omp barrier

Hinweis: $ omp end parallel wirkt wie eine interne Barrier

Leistungsmessung – Speedup

Leistungsmessung: Laufzeitmessung (Runtime)



Speedup

Speedup (englisch für *Beschleunigung*) beschreibt mathematisch den Zusammenhang zwischen der seriellen und der parallelen Ausführungszeit eines Programmteils.

Der Speedup ist definiert durch die folgenden beiden Formeln:

$$S_p = \frac{T_1}{T_p}$$

wobei gilt:

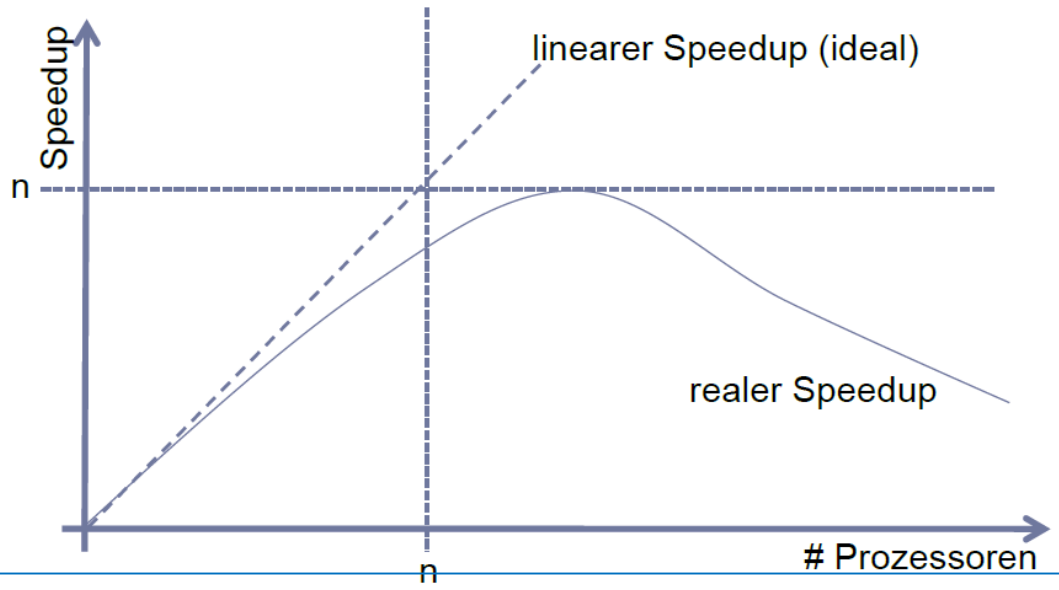
- p ist die Anzahl von Prozessoren
- S_p ist der theoretische Speedup, der erreicht werden kann bei Ausführung des Algorithmus auf p Prozessoren
- T_1 ist die Ausführungszeit auf einem Ein-Prozessor-System
- T_p ist die Ausführungszeit auf einem Mehrprozessorsystem

[Quelle: http://de.wikipedia.org/wiki/Speedup]

Speedup II

Darstellung der Speedup-Kurve

(nach Ludwig WS12/13)



Abweichung vom linearen Speedup:

- Anteil sequentiellem, nicht parallelisierbarem Programmcode -> Amdahlsches Gesetz
- Zunehmender Anteil an Kommunikation (und Latenz) bei steigender Anzahl an Prozessoren

Amdahlsches Gesetz I:

Einfache Überlegung:
Gesamtlaufzeit ergibt sich aus Summe aus seq. und parallelem Anteil.

$$T = t_S + t_P$$

Unter Verwendung von:

T	Gesamtlaufzeit
t_S	Laufzeit eines seriellen Programmabschnitts
t_P	Laufzeit eines parallelen Programmabschnitts

Quelle: https://de.wikipedia.org/wiki/Amdahlsches_Gesetz

Amdahlsches Gesetz II:

Resultierender Speedup:

$$\eta_S = \frac{T}{t_S + \frac{t_P}{n_P}} \leq \frac{T}{t_S} = \frac{T}{T - t_P}$$

Unter Verwendung:

η_S	Speedup-Faktor
n_P	Anzahl der Prozessoren, welche von parallelen Programmabschnitten genutzt werden (können)

Quelle: https://de.wikipedia.org/wiki/Amdahlsches_Gesetz

Amdahlsches Gesetz II:

Resultierender Speedup:

$$\eta_S = \frac{T}{t_S + \frac{t_P}{n_P}} \leq \frac{T}{t_S} = \frac{T}{T - t_P}$$

beschleunigeter Paralleler Anteil

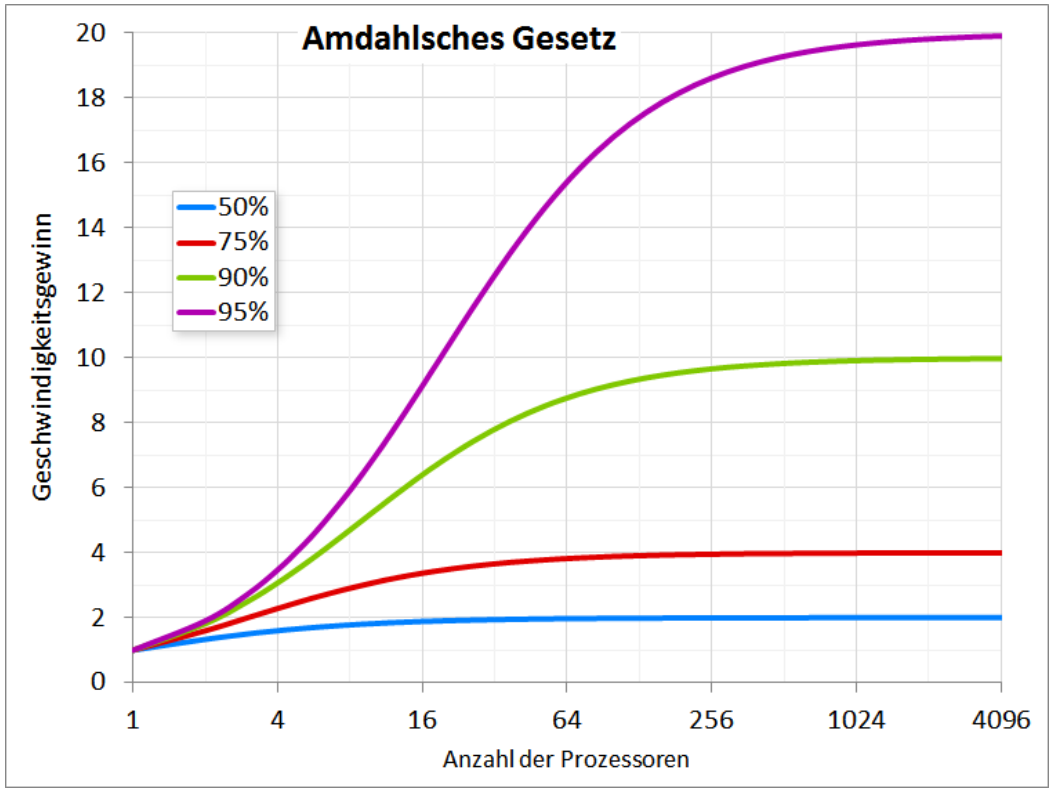
Unter Verwendung:

η_S	Speedup-Faktor
n_P	Anzahl der Prozessoren, welche von parallelen Programmabschnitten genutzt werden (können)

Quelle: https://de.wikipedia.org/wiki/Amdahlsches_Gesetz

Amdahlsches Gesetz III:

Bei steigender Anzahl an Prozessoren ist die Beschleunigung immer stärker vom sequentiellen Anteil des Programmes abhängig!

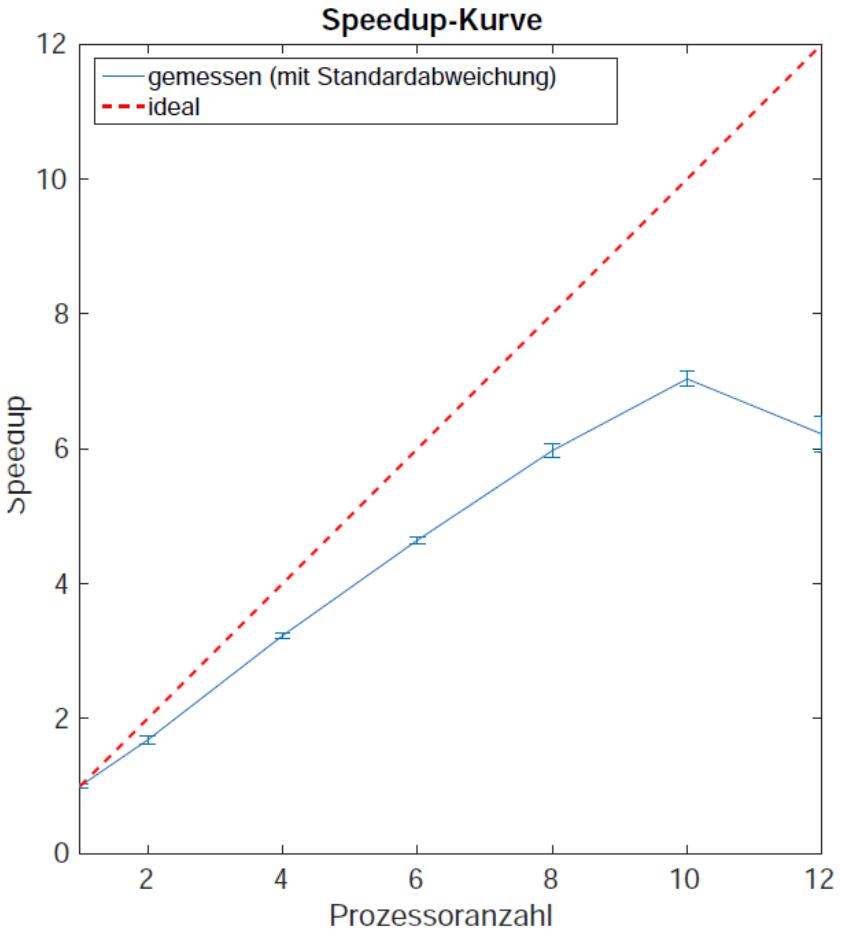


Quelle: https://de.wikipedia.org/wiki/Amdahlsches_Gesetz

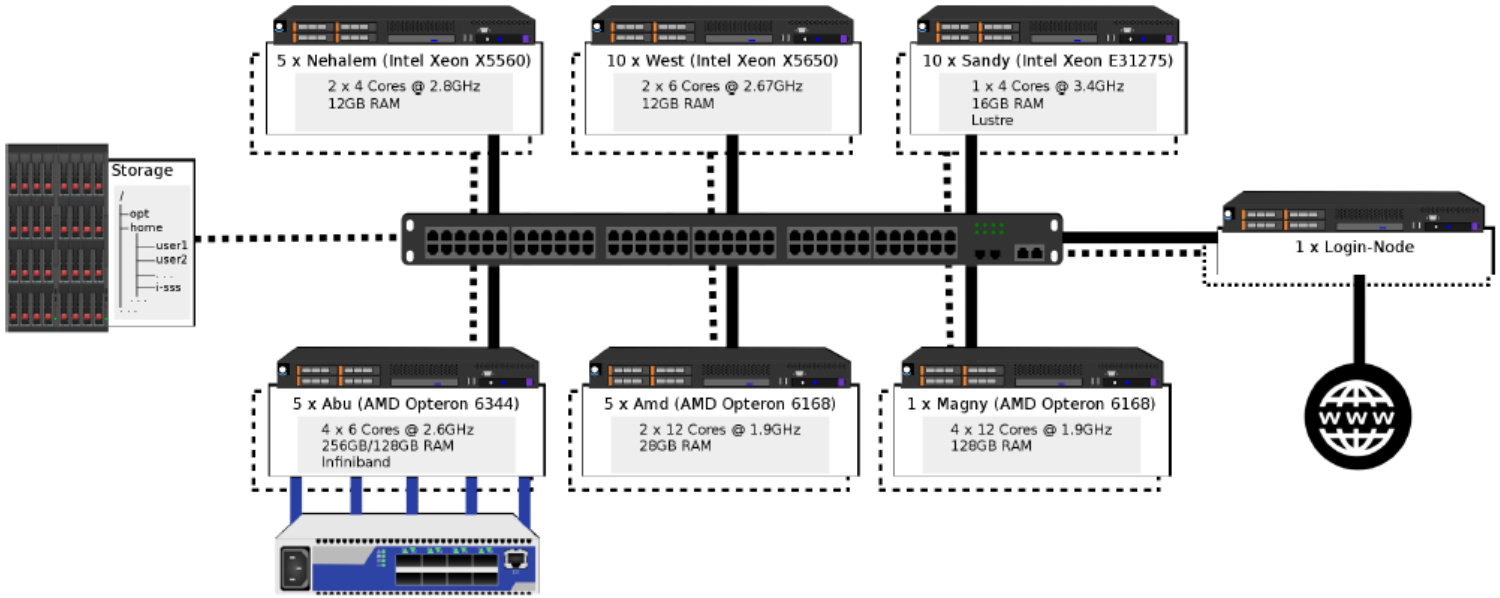
Anstieg der Kommunikation:

Mit zunehmender Anzahl an Prozessore weicht die Speedup-Kurve zunehmend von der idealen Speedup.-Kurve ab.

De Grund hierfür liegt in der Zunahme in der Kommunikation zwischen den Prozessoren.



Leistungsmessung – auf dem Cluster



Leistungsmessung – auf dem Cluster mit Slurm

„ Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system“

Quelle: https://slurm.schedmd.com/overview.html

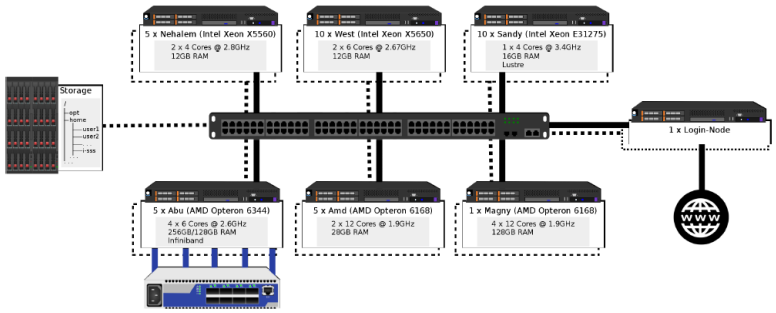
Mittels Slurm sollen die Leistungsmessungen auf einem der West-Knoten ausgeführt werden.

Slurm Nutzung

Verfügbare Ressource anzeigen lassen:

$ sinfo

```
lenhart@cluster:~/LEHRE/PPG-18$ sinfo
PARTITION    AVAIL  TIMELIMIT  NODES  STATE NODELIST
abu             up    6:00:00      1 drain* abu3
abu             up    6:00:00      3  down* abu[2,4-5]
abu             up    6:00:00      1   idle abu1
amd             up    6:00:00      5   idle amd[1-5]
magny           up    6:00:00      1   idle magny1
nehalem         up    6:00:00      1   idle nehalem5
west            up    6:00:00     10   idle west[1-10]
reservation     up 1-00:00:00     10   idle west[1-10]
lenhart@cluster:~/LEHRE/PPG-18$
```



Slurm Nutzung II

- Available partitions:
  - `west`
  - `abu`
  - `amd`
  - `magny`
  - `nehalem`
- Important sinfo node-states:

idle:	Available for use
alloc:	Node is used just now
drain:	Node is/will be unavailable by admin request
down:	Unavailable

Quelle: Jannek Squar Vortrag

Auf West Knoten arbeiten I

1) Einen West Knoten alloziieren:

$ salloc –N 1 –p west

2) Nachsehen welche Knoten zugewiesen wurde

$ squeue

```
lenhart@cluster:~/LEHRE/PPG-18$ salloc -N 1 -p west
salloc: Granted job allocation 61691
lenhart@cluster:~/LEHRE/PPG-18$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             61691      west     bash  lenhart  R       0:04      1 west1
lenhart@cluster:~/LEHRE/PPG-18$
```

Auf West Knoten arbeiten II

3) Auf zugewiesenen Knoten einloggen:

$ ssh west1

```
lenhart@cluster:~/LEHRE/PPG-18$ salloc -N 1 -p west
salloc: Granted job allocation 61691
lenhart@cluster:~/LEHRE/PPG-18$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             61691      west     bash  lenhart  R       0:04      1 west1
lenhart@cluster:~/LEHRE/PPG-18$ ssh west1
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-116-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
Last login: Wed Jun 27 15:36:38 2018 from 10.0.0.247
lenhart@west1:~$
```

Sobald die Kennung auf Knoten wechselt kann man auf dem Knoten arbeiten.

Wichtig: Für MPI muss das Modul vorher über spack wieder geladen werden!

Auf West Knoten arbeiten III

4) Arbeiten auf dem Knoten, z.B. Leistungsmessungen durchführen

man kann normal in seinem Account durch die Directories navigieren
und seine Jobs starten.

5) Nach der Arbeiten auf dem Knoten abmelden

$ exit gibt den Knoten wieder frei
damit dieser für weitere Nutzer wieder zur Verfügung steht.

> Kennung muss wieder auf name$cluster: wechseln !

Notenplanung!

PPG Benotung:

Gesamtpunktzahl ohne Bonus
2200

50 % entspricht 1100 Punkte

Note	Prozentanteile
1.0	95-100 %
1.3	90-95 %
1.7	85-90 %
2.0	80-85 %
2.3	75-80 %
2.7	70-75 %
3.0	65-70 %
3.3	60-65 %
3.7	55-60 %
4.0	50 -55 %

Danke das wars, inhaltlich für heute!

Nächste Woche:

Vortrag von Panos Adamidis

- (ICON hochauflösendes Wolkenmodell)
- DKRZ Beratung
- Hybride Programmierung

Danke das wars!