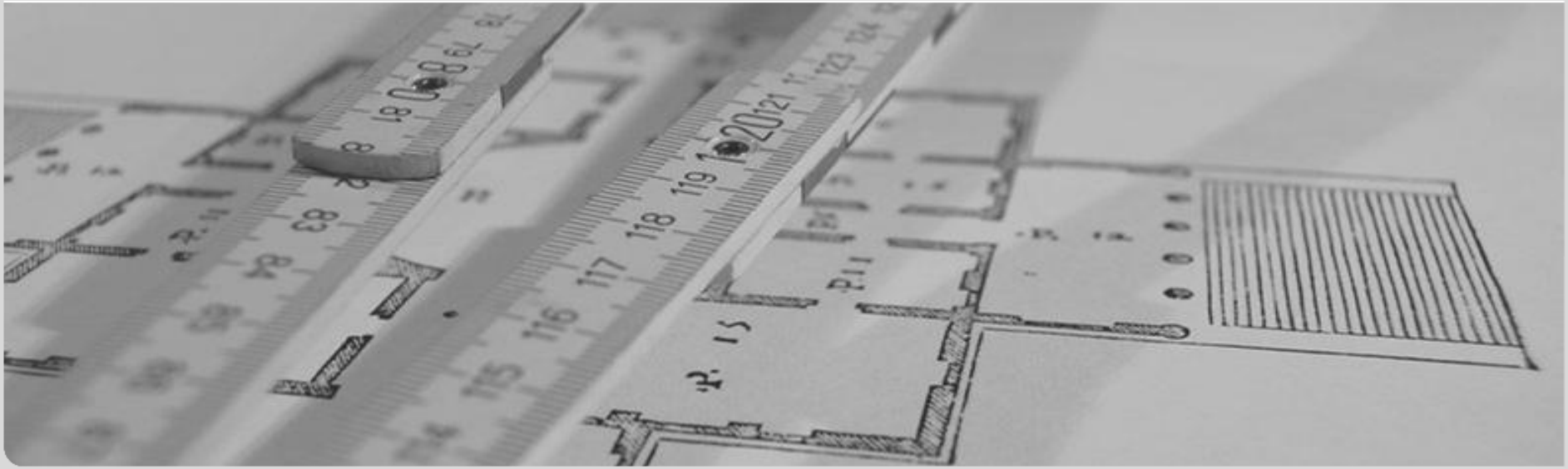
Karlsruhe Institute of Technology

Chapel

Dennis Appelt

Seminar: Sprachen für Parallelverarbeitung

INSTITUTE FOR PROGRAM STRUCTURES AND DATA ORGANIZATION, FACULTY OF INFORMATICS



Agenda

- Einleitung
- Sprachdesign
  - Annahmen
  - Wünschenswerte Spracheigenschaften
- Die Sprache Chapel
  - Daten-Parallelismus
  - Task-Parallelismus
- Zusammenfassung und Vergleich

„From ten programmers, eleven are unable to write multithreaded code.“

- unknown

Eckdaten Chapel

- Ziele:
  - Programmierbarkeit
  - Produktivität
- Hintergrund:
  - DARPA High Productivity Computer Systems
  - Cray

Sprachdesign

Annahmen

Aufgaben des Programmierers:

- Parallelität
- Synchronization
- Datenverteilung und Lokalität

Programmierer wird unterstützt durch

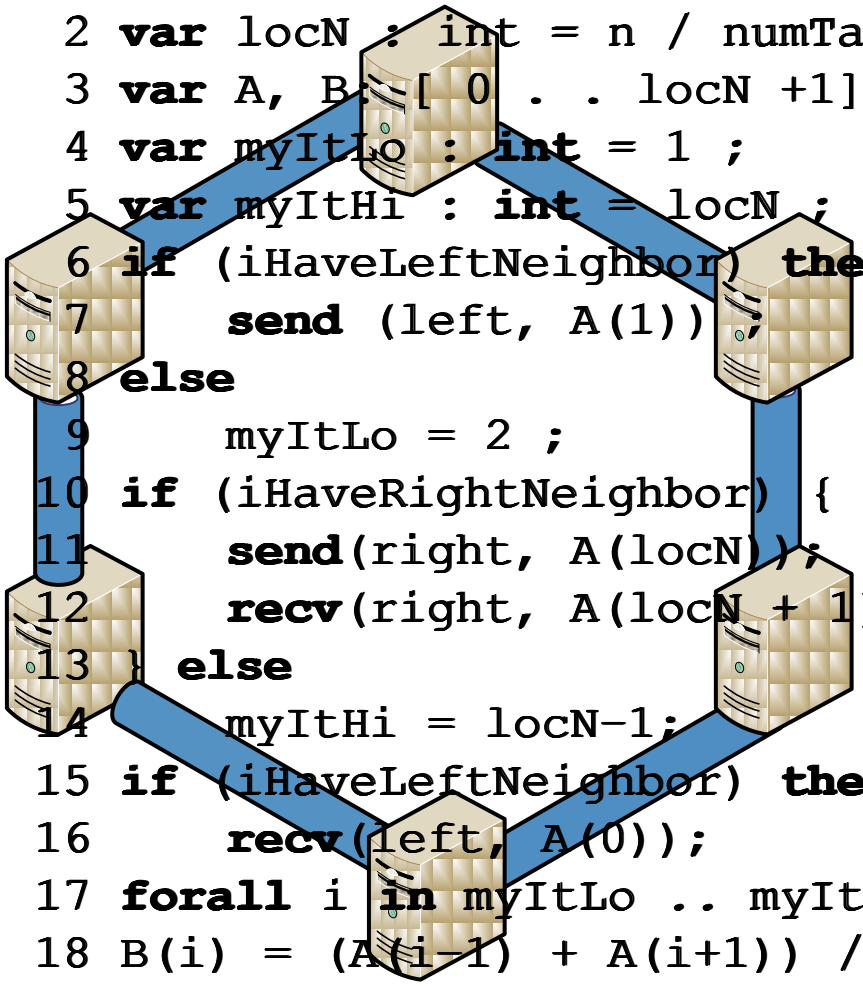
- High-level Sprachabstraktionen
- „specifying intend rather than mechanism“

Wünschenswerte Spracheigenschaften

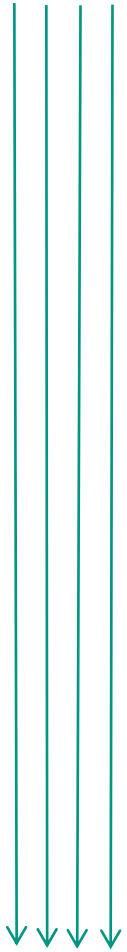
- Globale Sicht auf ein Programm
  - Fragmentierte vs. Globale Sicht

Beispiel – fragmentierte Sicht

```
1 var n : int = 1000;
2 var locN : int = n / numTasks ;
3 var A, B:[ 0 . . locN +1] float ;
4 var myItLo : int = 1 ;
5 var myItHi : int = locN ;
6 if (iHaveLeftNeighbor) then
7     send (left, A(1)) ;
8 else
9     myItLo = 2 ;
10 if (iHaveRightNeighbor) {
11     send(right, A(locN));
12     recv(right, A(locN + 1));
13 } else
14     myItHi = locN-1;
15 if (iHaveLeftNeighbor) then
16     recv(left, A(0));
17 forall i in myItLo .. myItHi do
18 B(i) = (A(i-1) + A(i+1)) / 2 ;
```

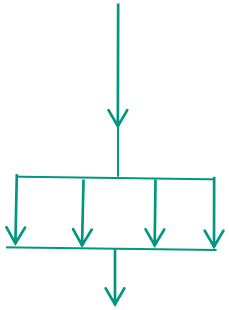


Beispiel – fragmentierte Sicht



```
 1 var n : int = 1000;
 2 var locN : int = n / numTasks;
 3 var A, B: [ 0 . . locN +1] float;
 4 var myItLo : int = 1;
 5 var myItHi : int = locN;
 6 if (iHaveLeftNeighbor) then
 7     send (left, A(1));
 8 else
 9     myItLo = 2 ;
10 if (iHaveRightNeighbor) {
11     send(right, A(locN));
12     recv(right, A(locN + 1));
13 } else
14     myItHi = locN-1;
15 if (iHaveLeftNeighbor) then
16     recv(left, A(0));
17 forall i in myItLo .. myItHi do
18 B(i) = (A(i-1) + A(i+1)) / 2 ;
```

Beispiel – globale Sicht



```
1 var n: int = 1000;
2 var A, B: [1..n] float;
3 forall i in 2..n-1
4       B(i) = (A(i-1) + A(i+1)) / 2 ;
```

Wünschenswerte Spracheigenschaften

- Globale Sicht auf ein Programm
  - Fragmentierte vs. Globale Sicht
- Generalisierte Parallelität

Wünschenswerte Spracheigenschaften

- Globale Sicht auf ein Programm
  - Fragmentierte vs. Globale Sicht
- Generalisierte Parallelität
- Trennung von Algorithmus und Implementierung der Datenstruktur

Beispiel – Matrix-Vektor Multiplikation

```
1 var n: int;
2 var M: [1..n, 1..n] float;
3 var V, S: [1..n] float;
4 for i in 1..n {
5      S(i) = 0.0;
6      for j in 1..n do
7            S(i) += M(i, j)*V(j);
8 }
```

dense matrix

```
1 var n, nnz : int;
2 var Mvals: [1..nnz] float;
3 var col : [1..nnz] int;
4 var rptr: [1..n+1] int;
5 var V, S: [1..n] float;
6 for i in 1..n {
7     S(i) = 0.0;
8     for j in rptr(i)..rptr (i+1)-1 do
9           S(i) += Mvals(j)*V(col(j));
10 }
```

sparse matrix

Wünschenswerte Spracheigenschaften

- Globale Sicht auf ein Programm
  - Fragmentierte vs. Globale Sicht
- Generalisierte Parallelität
- Trennung von Algorithmus und Implementierung
- Performance

Wünschenswerte Spracheigenschaften

- Globale Sicht auf ein Programm
  - Fragmentierte vs. Globale Sicht
- Generalisierte Parallelität
- Trennung von Algorithmus und Implementierung
- Performance
- eingebaute Datenstrukturen

Wünschenswerte Spracheigenschaften

- Globale Sicht auf ein Programm
  - Fragmentierte vs. Globale Sicht
- Generalisierte Parallelität
- Trennung von Algorithmus und Implementierung
- Performance
- eingebaute Datenstrukturen
- Sprachfeatures

Wünschenswerte Spracheigenschaften

- Globale Sicht auf ein Programm
- Generalisierte Parallelität
- Trennung von Algorithmus und Implementierung
- Performance
- eingebaute Datenstrukturen
- Sprachfeatures

- Globale Sicht
- Generalisierte Parallelität
- Algorithmus ⇔ Datenstruktur
- Performance

Die Sprache Chapel

Daten-Parallelismus

- Domain und Array

Domain := Menge von Indizies, die Größe und Struktur eines Array bestimmen.

Daten-Parallelismus

Globale Sicht

Generalisierte Parallelität

Algorithmus ⇔ Datenstruktur

Performance

Arithmetische Domain:

- Integer Indizies
- Ein- oder Mehrdimensional
- „Traditionelle“ Arrays

Beispiel:

```
1 var D: domain(2) = [1..m, 1..n];
2 var A: [D] float;
```

Daten-Parallelismus

Globale Sicht
Generalisierte Parallelität
Algorithmus ⇔ Datenstruktur
Performance

Karlsruhe Institute of Technology

Unendliche Domain

- Beliebiger Indizetyp
- Sets oder assoziative Arrays

- Beispiel:

```
1 var People: domain(string);
2 var Age: [People] int;
3 People += ”John”;
4 Age(”John”) = 62;
```

Daten-Parallelismus

Globale Sicht
Generalisierte Parallelität
Algorithmus ⇔ Datenstruktur
Performance

- Iteration und Slicing

```
1 var innerD: subdomain(D) = [2..m-1, 2..n-1];
2 forall ij in innerD {
3       A(ij) = ...
4 }
```

implizit parallel

- Promotion und Reduction

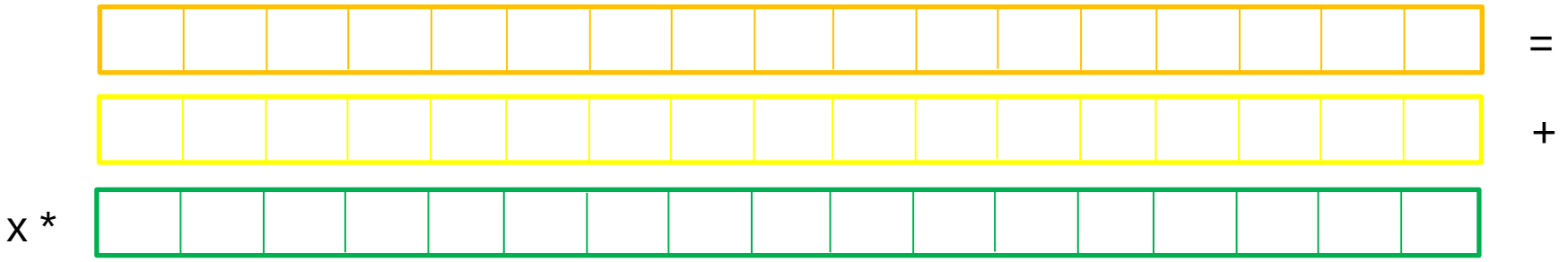
```
1 var rnmu = max reduce abs(A);
```

Reduction Promotion

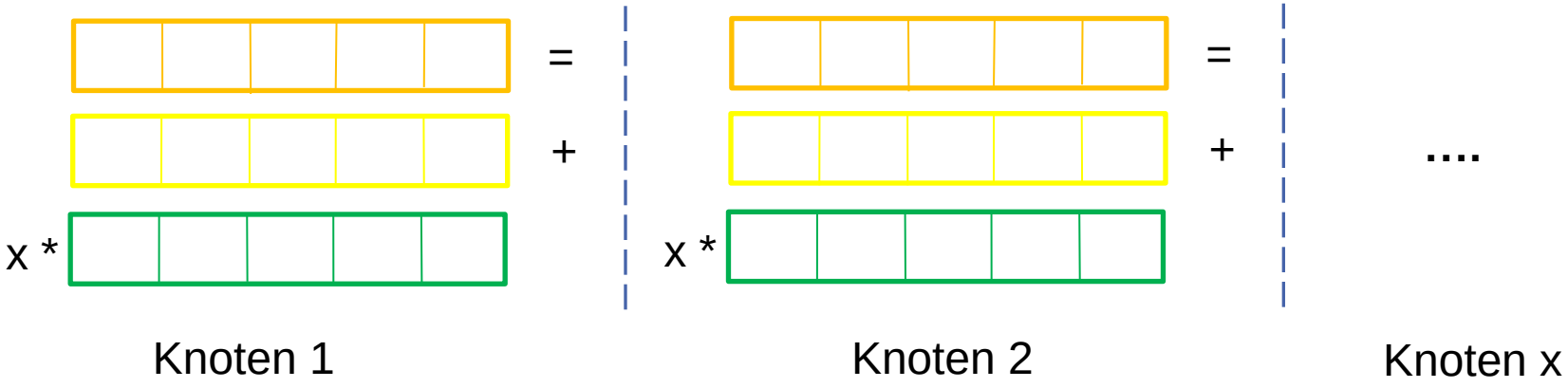
Daten-Parallelismus

Distribution: Verteilungsvorschrift für Domain

- Hilft dem Compiler die globale Sicht …



- … auf die fragmentierte Sicht abzubilden



Task-Parallelismus

- Task erstellen:

```
1 begin DoThisTask();
2 TheOriginalThread();
```

```
1 var pivot = computePivot(lo, hi, data);
2 cobegin {
3       Quicksort(lo, pivot, data);
4       Quicksort(pivot, hi, data);
5 }
```

Task-Parallelismus

- Synchronisation:

```
1 sync {
2       begin treeSearch(root);
3   }
4   def treeSearch(node) {
5       if node == nil then return;
6       begin treeSearch(node.right);
7       begin treeSearch(node.left);
8   }
```

```
1 atomic {
2 …
3 }
```

Task-Parallelismus

Globale Sicht
Generalisierte Parallelität
Algorithmus ⇔ Datenstruktur
Performance

Lokalität:

- basierend auf Verarbeitungseinheiten:

```
1 on Locales(0) do computeTaskC(…);
```

- basierend auf Speicherort:

```
1 on data[lo] do Quicksort(lo, pivot, data);
```

Zusammenfassung

Viele der gewünschten Spracheigenschaften sind erfüllt:

- Globale Sicht
- Generalisierte Parallelität
- Trennung von Algorithmus und Implementierung der Datenstruktur
- Performance-Tuning (durch Lokalität)

HPCS Sprachen im Vergleich

Eigenschaft	Chapel	Fortress	X10
Global-View Programmierung	✓	✓	✓
Mainstream-tauglich	✓	✗	✓
eigenständige Sprache	✓	✓	✗
Locales sind explizit ansprechbar	✓	✓	✓
Lokaler und Globaler Datenzugriff unterscheiden sich syntaktisch	✓	✓	✓
Array‘s sind verteilt	✓	✓	✓
zahlreiche Out-of-Box Distributions	✓	✓	✗
Redistribution von Array‘s	✗	✓	✗
Erweiterbarkeit	✗	✓	✗

Ende

```
1 forall p in PeopleSpace {
2       people(p) = „Vielen Dank für Ihre
                Aufmerksamkeit.“;
3 }
4 writeln(„Noch Fragen?“)
```

Literatur

- **Parallel Programmability and the Chapel Language**; Chamberlain, Callahan, Zima; *International Journal of High Performance Computing Applications*, 2007
- **An approach to data distributions in Chapel**; R. Diaconescu/H. P. Zima. *International Journal of High Performance Computing Applications*, 2007
- Chapel Specification (version 0.795)
- **The Cascade High Productivity Language**, David Callahan, Bradford L. Chamberlain, Hans P. Zima. In *9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2004)*, pages 52-60. IEEE Computer Society, April 2004

Literatur II

- **New Languages for High Performance, High Productivity Computing**, J.V. Ashby, August 2007
- **Chapel, Fortress and X10: novel languages for HPC**, Michele Weiland, Oktober 2007
- **Programming Languages for HPC**: Is There Life After MPI?, Marc Snir, März 2006