

Universität Karlsruhe (TH)

Forschungsuniversität · gegründet 1825

**ZPL**

Prof. Dr. Walter F. Tichy

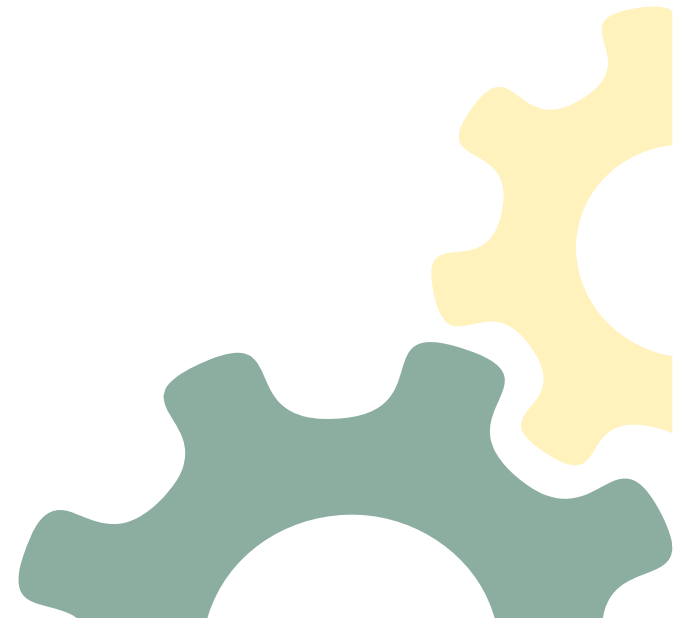
Dr. Victor Pankratius

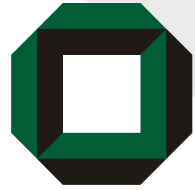
Ali Jannesari



**Fakultät für Informatik**

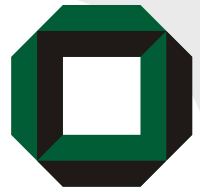
Lehrstuhl für Programmiersysteme





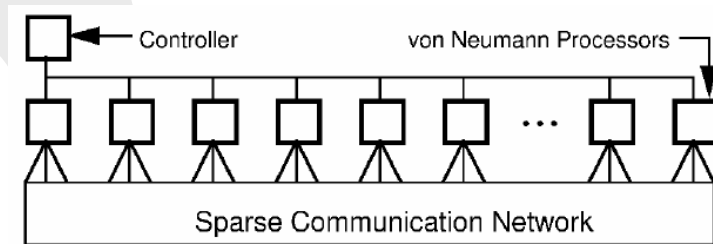
# Agenda

1. ZPL – Überblick
2. Konzepte von ZPL
  - Regionen, Region Specifiers
  - Arrays
  - Richtungen
  - Ausgewählte Operatoren (of, in, at, @, <<)
3. Ein Beispiel-Programm
4. Zusammenfassung

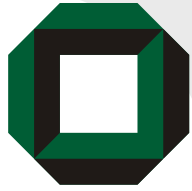


# ZPL - Überblick

- Parallele Programmiersprache
- Entwickelt für Maschinen mit verteiltem Speicher



- Besonderheiten
  - Mächtige Konstrukte für den Umgang mit Arrays
  - Parallelisierung/Kommunikation geschieht implizit



# Konzepte von ZPL

- **Region**

- Definiert eine (fixe) Menge von Indizes
- Allgemeine Syntax:  $[l_1..u_1, l_2..u_2, \dots, l_r..u_r]$   
mit  $l_i \leq u_i$  integer;  $r$  wird Rang genannt

- *Beispiel:*

```
region R = [1..2, 1..2]
```

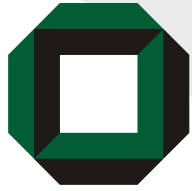
```
entspricht kartesischem Produkt: {(1,1), (1,2), (2,1), (2,2)}
```

- Regionen können benutzt werden, um **Arrays** zu definieren.

- *Beispiel:*

```
var myArray: [R] integer
```

```
definiert ein zweidimensionales Integer-Array
```



# Konzepte von ZPL

- **Region Specifiers**

- Werden als Präfix für andere Ausdrücke verwendet

- Beispiel:

```
[V] X := Y + Z;
```

- X, Y, Z sind Arrays vom Rang r
- [V] ist Region Specifier vom Rang r.  
Bedeutung: Für alle Indizes, die durch die Region spezifiziert sind, soll die Operation + auf die jeweiligen Array-Elemente ausgeführt werden.
- Konkreter:

```
region V=[1..5]; Vpre=[1..3];  
var X,Y,Z: [V] integer;
```

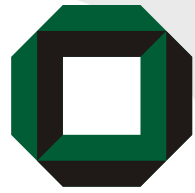
*Annahme initiale Werte: Y = (1,3,5,7,9) und Z=(8,6,4,2,0)*

```
[V] X := Y + Z;
```

*d.h., X = (9, 9, 9, 9, 9)*

```
[Vpre] X := X / 3;
```

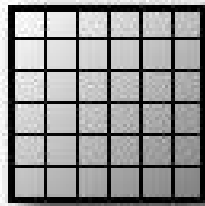
*d.h. X = (3, 3, 3, 9, 9)*



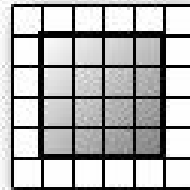
# Konzepte von ZPL

- Auswahl von Indizes in Regionen

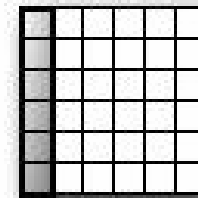
```
region  
R = [1..n,1..n];
```



```
region  
IntR =  
[2..n-1,2..n-1];
```



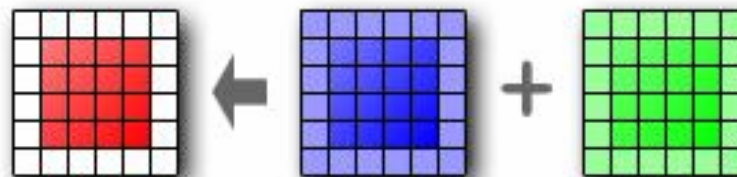
```
region  
Left = [1..n,1];
```

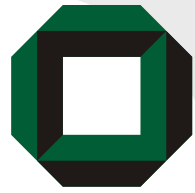


IntR spezifiziert  
„innere“ Indizes

Erste Spalte

```
[IntR] C := A + B;
```





# Konzepte von ZPL

- **Richtungen (Directions)**
  - Vektor-Konstanten, mit denen man relative Positionen ausdrücken kann

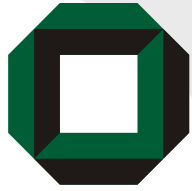
```
direction
north = [-1, 0];
east  = [ 0, 1];
south = [ 1, 0];
west  = [ 0,-1];

nw = [-1,-1];
ne = [-1, 1];
sw = [ 1,-1];
se = [ 1, 1];
```



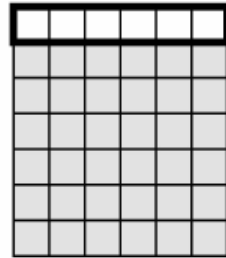
*Finde neuen Punkt, indem Du 1 von aktueller X-Koordinate subtrahierst und Y-Koordinate gleich lässt.*

- Werden als Operanden für die "at", "in" und "of" Operatoren benutzt.

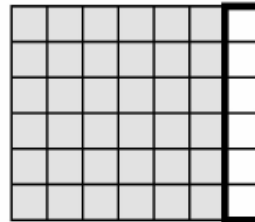


# Konzepte von ZPL

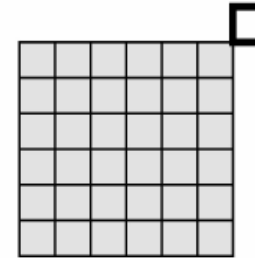
```
east  = [ 0, 1];  
north = [-1, 0];  
ne    = [-1, 1];  
east2 = [ 0, 2];
```



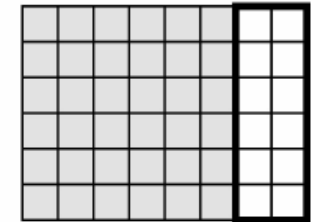
[north of R]



[east of R]

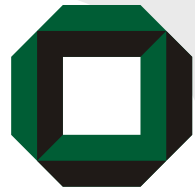


[ne of R]



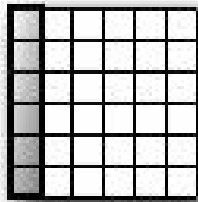
[east2 of R]

- Der **“of”-Operator** benutzt eine Richtung und eine Region, um eine neue Region zu definieren, die adjazent zur vorigen Region ist.
  - „0“ in einer Richtung bedeutet, dass in der neuen Region das ganze jeweilige Intervall „geerbt“ wird.
  - **Vorzeichen** in Richtung gibt an, auf welcher Seite Erweiterung stattfinden soll
  - **Absolutwert** in Richtung gibt die Größe der Erweiterung an

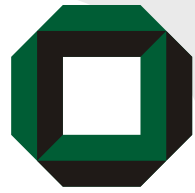


# Konzepte von ZPL

```
region  
  Left = west in R;
```

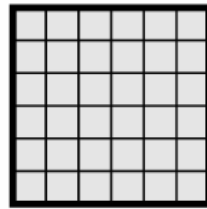


- Der „in“-Operator benutzt eine Richtung und eine Region, um eine neue Region zu erzeugen, die sich innerhalb der alten Region befindet. Gegenstück des „of“-Operators.
  - „0“ in einer Richtung bedeutet, dass in der neuen Region das ganze jeweilige Intervall „geerbt“ wird.
  - **Vorzeichen** in Richtung gibt an, an welchen Rand die neue Region definiert wird
  - **Absolutwert** gibt Breite der neuen Region an

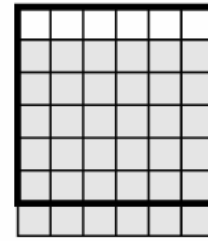


# Konzepte von ZPL

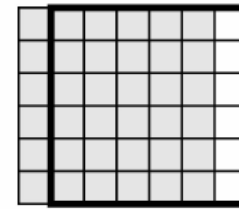
```
east = [ 0, 1];  
north = [-1, 0];  
ne = [-1, 1];
```



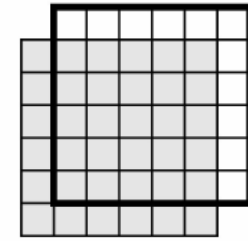
[R]



[R at north]

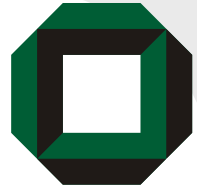


[R at east]



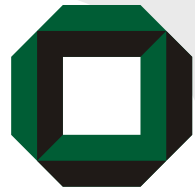
[R at ne]

- Der **“at”-Operator** erzeugt eine neue Region, indem er eine Menge von Indizes um einem vordefinierten Vektor verschiebt, ohne die Größe oder Form der Region zu verändern.



# Konzepte von ZPL

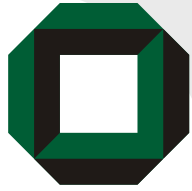
- Der „@“-Operator „verschiebt“ Daten einer Region in eine angegebene Richtung und referenziert diese anschließend.
  - Kommunikation geschieht implizit
  - *Anwendungsbeispiel später (Folie 17)*



# Konzepte von ZPL

- **Reduktions-Operator „<<“**
  - Wendet Funktion akkumulativ auf Array-Elemente an
  - *Beispiel: Array **A***  
*+<<A berechnet Summe aller Elemente*
  - *Weitere*

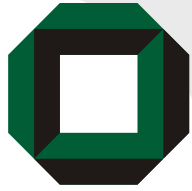
<u>Reduce</u>	<u>Name</u>
+<<	plus
*<<	times
max<<	maximum
min<<	minimum
&<<	and
<<	or



# Beispiel

```
1 program Jacobi;
2 /*          Jacobi Iteration
3           Written by L. Snyder, May 1994          */
4 config var  n      : integer  = 512; -- Declarations
5            delta : float    = 0.000001;
6
7 region     R = [1..n, 1..n];
8 var       A, Temp: [R] float;
9           err   : float;
10
11 direction north = [-1, 0];
12           east  = [ 0, 1];
13           west  = [ 0,-1];
14           south = [ 1, 0];
15
16 procedure Jacobi();
17 begin
18   [R]      A := 0.0;           -- Initialization
19   [north of R] A := 0.0;
20   [east  of R] A := 0.0;
21   [west  of R] A := 0.0;
22   [south of R] A := 1.0;
23
24   [R]      repeat           -- Body
25             Temp := (A@north + A@east
26                    + A@west + A@south)/4.0;
27             err  := max<< abs(A - Temp);
28             A    := Temp;
29             until err < delta;
30 end;
```

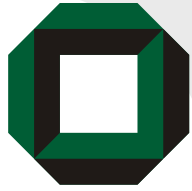
- **Jacobi-Iteration:** Gegeben Array ein A, ersetze iterativ dessen Elemente mit dem Durchschnittswert ihrer 4 nächsten Nachbarn, bis die größte Änderung zwischen zwei Iterationen kleiner Delta ist.



# Beispiel

```
1 program Jacobi;
2 /*          Jacobi Iteration
3           Written by L. Snyder, May 1994          */
4 config var   n      : integer   = 512; -- Declarations
5             delta : float     = 0.000001;
6
7 region      R = [1..n, 1..n];
8 var         A, Temp: [R] float;
9             err   : float;
10
11 direction  north = [-1, 0];
12            east  = [ 0, 1];
13            west  = [ 0,-1];
14            south = [ 1, 0];
15
16 procedure Jacobi();
17 begin
18   [R]      A := 0.0; -- Initialization
19   [north of R] A := 0.0;
20   [east  of R] A := 0.0;
21   [west  of R] A := 0.0;
22   [south of R] A := 1.0;
23
24   [R]      repeat -- Body
25             Temp := (A@north + A@east
26                    + A@west + A@south)/4.0;
27             err  := max<< abs(A - Temp);
28             A    := Temp;
29             until err < delta;
30 end;
```

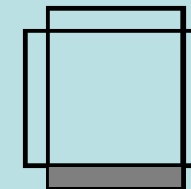
Präfix [R] von A besagt, dass alle  $n^2$  Elemente des Arrays A auf 0.0 gesetzt werden sollen



# Beispiel

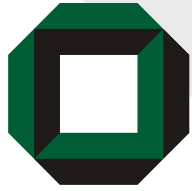
```
1 program Jacobi;
2 /*          Jacobi Iteration
3           Written by L. Snyder, May 1994          */
4 config var  n      : integer  = 512; -- Declarations
5            delta : float    = 0.000001;
6
7 region      R = [1..n, 1..n];
8 var         A, Temp: [R] float;
9            err   : float;
10
11 direction  north = [-1, 0];
12            east  = [ 0, 1];
13            west  = [ 0,-1];
14            south = [ 1, 0];
15
16 procedure Jacobi();
17 begin
18   [R]       A := 0.0; -- Initialization
19   [north of R] A := 0.0;
20   [east  of R] A := 0.0;
21   [west  of R] A := 0.0;
22   [south of R] A := 1.0;
23
24   [R]       repeat -- Body
25             Temp := (A@north + A@east
26                   + A@west + A@south)/4.0;
27             err  := max<< abs(A - Temp);
28             A    := Temp;
29             until err < delta;
30 end;
```

Randwerte setzen:



0's

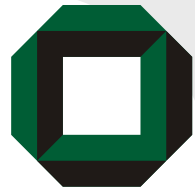
1's



# Beispiel

```
1 program Jacobi;
2 /*          Jacobi Iteration
3           Written by L. Snyder, May 1994          */
4 config var  n      : integer  = 512; -- Declarations
5            delta : float    = 0.000001;
6
7 region     R = [1..n, 1..n];
8 var       A, Temp: [R] float;
9           err   : float;
10
11 direction north = [-1, 0];
12           east  = [ 0, 1];
13           west  = [ 0,-1];
14           south = [ 1, 0];
15
16 procedure Jacobi();
17 begin
18   [R]      A := 0.0;           -- Initialization
19   [north of R] A := 0.0;
20   [east  of R] A := 0.0;
21   [west  of R] A := 0.0;
22   [south of R] A := 1.0;
23
24   [R]      repeat           -- Body
25     Temp := (A@north + A@east
26             + A@west + A@south)/4.0;
27     err  := max<< abs(A - Temp);
28     A    := Temp;
29     until err < delta;
30 end;
```

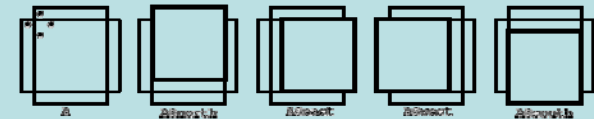
Wird auf Region R ausgeführt, d.h. über alle Indizes von R.



# Beispiel

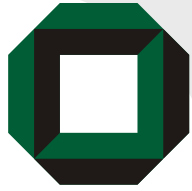
```
1 program Jacobi;
2 /*          Jacobi Iteration
3           Written by L. Snyder, May 1994          */
4 config var  n      : integer  = 512; -- Declarations
5            delta : float    = 0.000001;
6
7 region      R = [1..n, 1..n];
8 var         A, Temp: [R] float;
9            err   : float;
10
11 direction  north = [-1, 0];
12            east  = [ 0, 1];
13            west  = [ 0,-1];
14            south = [ 1, 0];
15
16 procedure Jacobi();
17 begin
18   [R]      A := 0.0;           -- Initialization
19   [north of R] A := 0.0;
20   [east  of R] A := 0.0;
21   [west  of R] A := 0.0;
22   [south of R] A := 1.0;
23
24   [R]      repeat           -- Body
25             Temp := (A@north + A@east
26                    + A@west + A@south)/4.0;
27             err  := max<< abs(A - Temp);
28             A    := Temp;
29             until err < delta;
30 end;
```

Findet für jedes Element von A die nächsten Nachbarn und weist Temp Durchschnittwert zu. Kein explizites Indizieren des Arrays!



Funktionsweise:  
Elementweise Kombination der 4 Arrays

$(i,j)@north$	$= (i,j) + north$	$= (i,j) + (-1, 0)$	$= (i-1, j)$
$(i,j)@east$	$= (i,j) + east$	$= (i,j) + (0, 1)$	$= (i, j+1)$
$(i,j)@west$	$= (i,j) + west$	$= (i,j) + (0,-1)$	$= (i, j-1)$
$(i,j)@south$	$= (i,j) + south$	$= (i,j) + (1, 0)$	$= (i+1, j)$

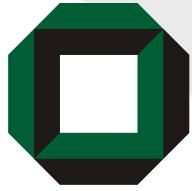


# Beispiel

```
1 program Jacobi;
2 /*          Jacobi Iteration
3           Written by L. Snyder, May 1994          */
4 config var  n      : integer  = 512; -- Declarations
5            delta : float    = 0.000001;
6
7 region     R = [1..n, 1..n];
8 var       A, Temp: [R] float;
9           err   : float;
10
11 direction north = [-1, 0];
12           east  = [ 0, 1];
13           west  = [ 0,-1];
14           south = [ 1, 0];
15
16 procedure Jacobi();
17 begin
18   [R]      A := 0.0;           -- Initialization
19   [north of R] A := 0.0;
20   [east  of R] A := 0.0;
21   [west  of R] A := 0.0;
22   [south of R] A := 1.0;
23
24   [R]      repeat           -- Body
25             Temp := (A@north + A@east
26                    + A@west + A@south)/4.0;
27             err  := max<< abs(A - Temp);
28             A    := Temp;
29             until err < delta;
30 end;
```

Berechnet Maximum von

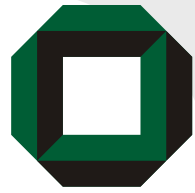
$\text{abs}(A_{1,1} - \text{Temp}_{1,1}),$   
 $\text{abs}(A_{1,2} - \text{Temp}_{1,2}), \dots,$   
 $\text{abs}(A_{n,n} - \text{Temp}_{n,n})$



# Beispiel

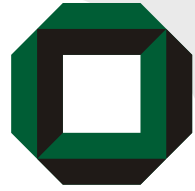
```
1 program Jacobi;
2 /*          Jacobi Iteration
3           Written by L. Snyder, May 1994          */
4 config var   n      : integer   = 512; -- Declarations
5             delta : float     = 0.000001;
6
7 region      R = [1..n, 1..n];
8 var        A, Temp: [R] float;
9           err   : float;
10
11 direction  north = [-1, 0];
12           east  = [ 0, 1];
13           west  = [ 0,-1];
14           south = [ 1, 0];
15
16 procedure Jacobi();
17 begin
18   [R]      A := 0.0;           -- Initialization
19   [north of R] A := 0.0;
20   [east  of R] A := 0.0;
21   [west  of R] A := 0.0;
22   [south of R] A := 1.0;
23
24   [R]      repeat           -- Body
25             Temp := (A@north + A@east
26                   + A@west + A@south)/4.0;
27             err := max<< abs(A - Temp);
28             A := Temp;
29           until err < delta;
30 end;
```

Wert von A mit Temp  
aktualisieren



# Zusammenfassung

- Array-Ausdrücke ersparen das Schreiben von verschachtelten Schleifen, die Indizes durchiterieren
- Konstrukte für Richtungen versuchen Programmierfehler zu umgehen, die durch Index-Berechnungen entstehen könnten (z.B. „*northeast*“ statt  $[i-1, j+1]$ )



# Literatur

1. L. Snyder, A Programmers Guide to ZPL, Department of Computer Science and Engineering, University of Washington, Seattle
2. B. Chamberlain et al., ZPL: a machine independent programming language for parallel computers, *Transactions on Software Engineering*, **2000**, 26, 197-211
3. <http://www.cs.washington.edu/research/zpl/home/index.html>