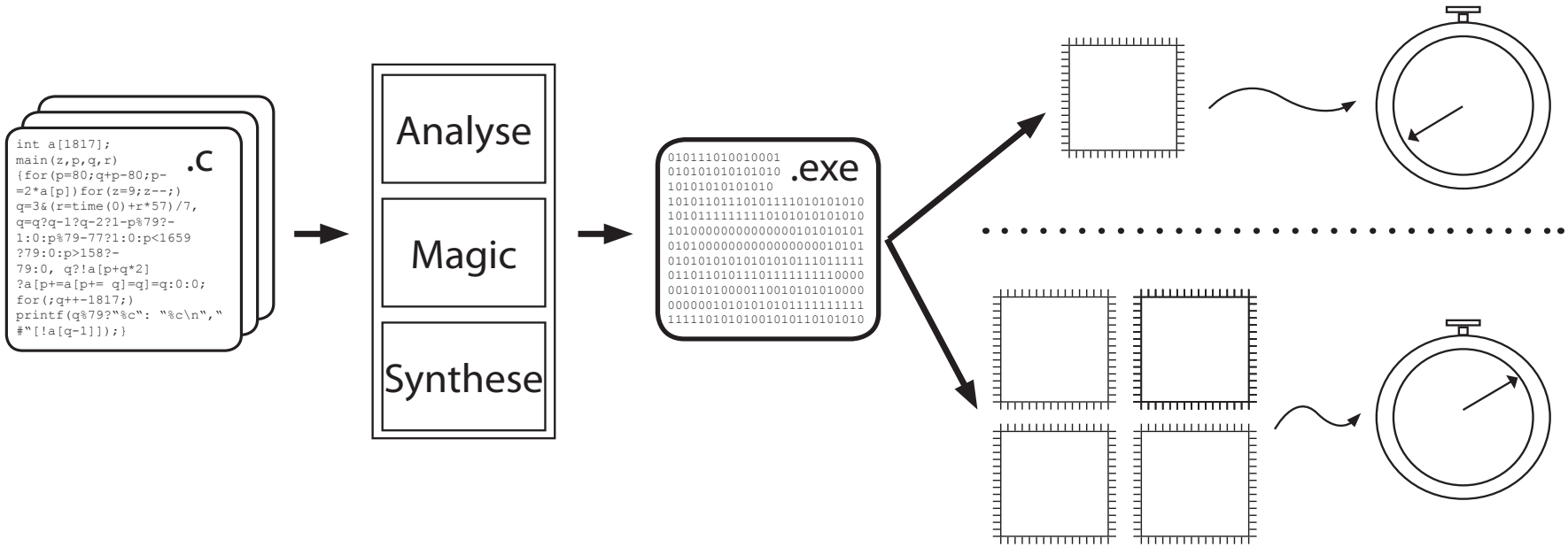
Karlsruher Institut für Technologie

Automatische Parallelisierung

Seminar »Sprachen für Parallelprogrammierung«
Julian Oppermann

IPD Snelting, Lehrstuhl Programmierparadigmen



Warum Autoparallelisierung?

Karlsruher Institut für Technologie

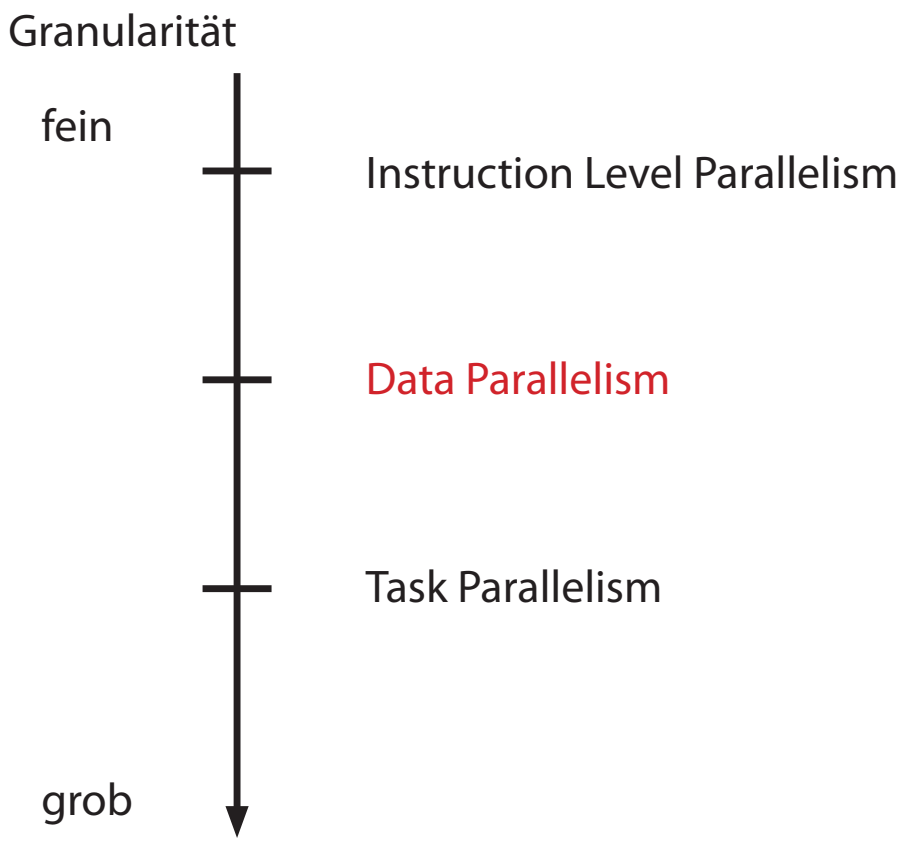
- Multicores überall, Rechenleistung liegt brach.
- Existierender Code soll parallelisiert werden.
- Effiziente, korrekte parallele Programme zu schreiben ist schwierig.

Voraussetzungen für effizientes, paralleles Rechnen

Karlsruher Institut für Technologie

- Minimiere Kommunikation (Datentransfer, Synchronisation, Wartezeiten)
- Skalierbarkeit

Wo setzt die automatische Parallelisierung an?



Beispiel

Original

```
1 for (i = 0; i < N; i++) {
2   a[i] = a[i] + 1;
3 }
```

Beispiel

Original

```
1 for (i = 0; i < N; i++) {
2   a[i] = a[i] + 1;
3 }
```

Transformation

```
1 int pid = getpid();
2 int lb = N / NUM_CPUS * pid;
3 int ub = N / NUM_CPUS * (pid+1);
4 for (i = lb; i < ub; i++) {
5   a[i] = a[i] + 1;
6 }
```

Beispiel (2)

Original

```
1 for (j = 0; j < 100; j++) {
2   for (k = 0; k < 100; k++) {
3     a[j,k] = a[j,k] + b[j-1,k];
4     b[j,k] = a[j, k-1] * b[j,k];
5   }
6 }
```

Beispiel (2)

Original

```
1 for (j = 0; j < 100; j++) {
2   for (k = 0; k < 100; k++) {
3     a[j,k] = a[j,k] + b[j-1,k];
4     b[j,k] = a[j, k-1] * b[j,k];
5   }
6 }
```

Transformation

?

Autoparallelisierung für imperative Programmiersprachen

Karlsruher Institut für Technologie

Problemumfeld:

- Numerische Anwendungen.
- Sprachen wie Fortran oder C.
- Mehrfach geschachtelte Schleifen führen Berechnungen auf Daten in Arrays auf.

Grundvoraussetzung für korrekte Parallelisierung

Karlsruher Institut für Technologie

Def. *Datenabhängigkeit*

Zwischen zwei Speicherzugriffen besteht eine Datenabhängigkeit, wenn sie das gleiche Ziel haben und mindestens ein Zugriff ein Schreibzugriff ist.

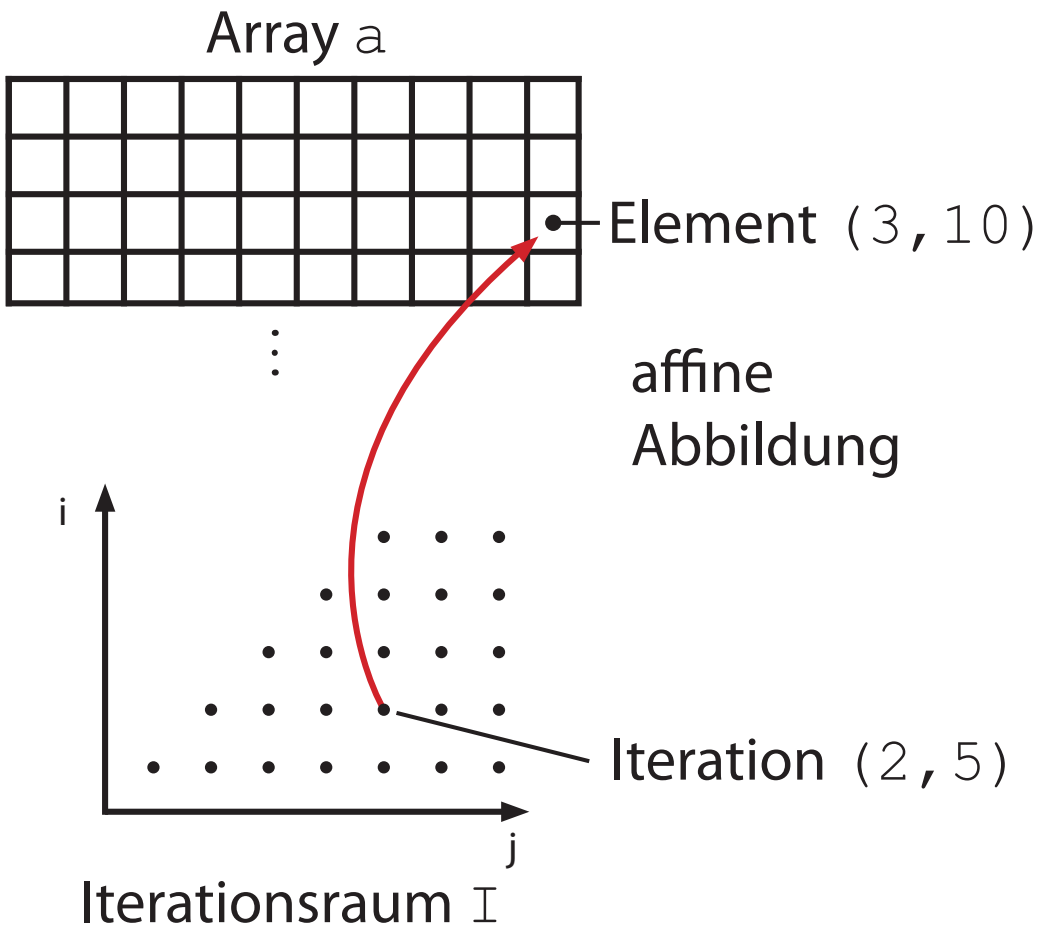
Zentrale Einsicht

Parallelisieren geht nur Berücksichtigung der Datenabhängigkeiten!

Zentrale Frage

Wann haben zwei Speicherzugriffe das gleiche Ziel?

Affine Arrayzugriffe



Beispiel

```
1 for (i=1; i<=5; i++)
2   for (j=i; j<=7; j++)
3     a[i+1,2*j] = 0;
```

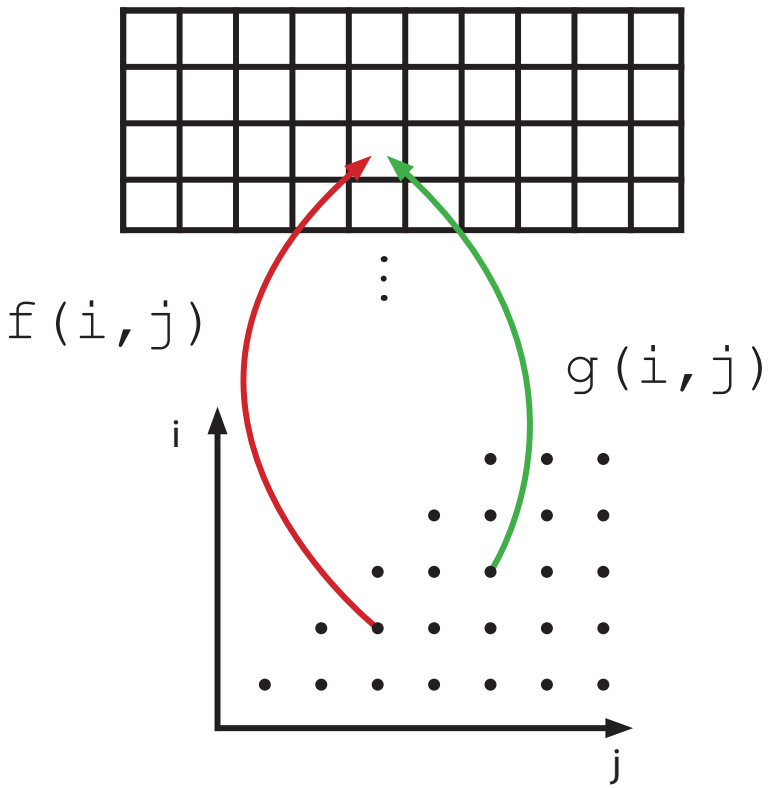
Affine Arrayzugriffe

a[i], a[$i-1$],
b[$2 \cdot i + 2$, $3 \cdot j - 7$]

Nicht-affine Arrayzugriffe

a[a[i]], a[$i \cdot j$], a[$N \cdot i + 1$]

Schleifengetragene Abhängigkeiten
engl. loop-carried dependences



$$\exists\, \vec{x}, \vec{y} \in \mathtt{I},$$
$$\vec{x} \neq \vec{y}:$$
$$f(\vec{x}) = g(\vec{y})$$
$$?$$

Affine Zugriffe $\Longrightarrow$ lineares Gleichungssystem

Test auf Unabhängigkeit

Karlsruher Institut für Technologie

Fragestellung »gibt es solche Lösungen?« ist allgemein NP-schwer.

Ansätze

- Es gibt keine ganzzahligen Lösungen. (→ ggT-Test)
  Bsp.: $\texttt{a}[2i] = \ldots; \ldots = \texttt{a}[2i+1]$

$$2x = 2y + 1 \implies 2x - 2y = 1 \implies \mathrm{ggT}(2,2) = 2 \nmid 1$$

- Es gibt keine reellen Lösungen innerhalb des Iterationsraums (→ Bannerjee-Wolfe-Test).
- Es gibt exakte Verfahren (nutzen praxisbezogene Einschränkungen)

Abhängigkeitsvektoren
engl. dependence vectors

Karlsruher Institut für Technologie

Vollständige Unabhängigkeit ist selten. Wenn es Abhängigkeiten gibt, wie verlaufen sie?

Abhängigkeitsvektoren

$$\vec{z} = (z_1, .., z_d), z_k = \begin{cases} = & \text{Abh. zur selben Iteration der k-ten Schleife} \\ < & \text{Abh. zu einer späteren Iteration der k-ten Schleife} \\ > & \text{Abh. zu einer früheren Iteration der k-ten Schleife} \end{cases}$$

Beispiel

Modifziertes Beispiel

```
1 for (i=1; i<=5; i++)
2   for (j=i; j<=7; j++)
3     a[i+1,2*j] = a[i,j]+c;
```

Gleichungen

$$x_i + 1 = y_i$$

$$2x_j = y_j$$

$$1 \leq x_i, y_i \leq 5$$

$$x_i \leq x_j \leq 7 \wedge y_i \leq y_j \leq 7$$

Lösungen

$$\left[\vec{x}, \vec{y}\right] = \underbrace{[(1,1),(2,2)]}_{(<,<)}, \underbrace{[(1,2),(2,4)]}_{(<,<)}, \ldots$$

~~$[(3,2),(4,4)], [(5,5),(6,10)]$~~

Affine Partitionierung

Und nun? Anwenden eines Katalogs an Transformationen. Aber wie?
Besseres Verfahren: Affine Partitionierung.

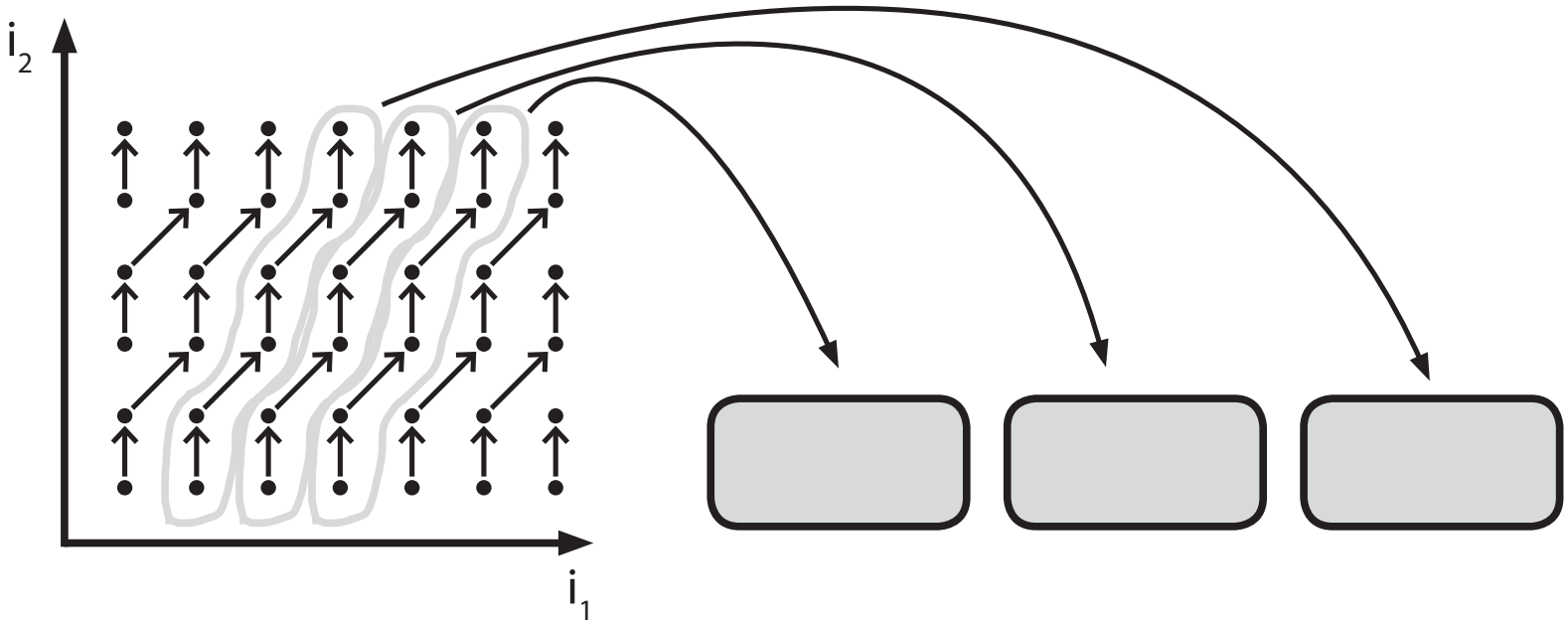
Idee

Finde sämtliche Parallelität, die keine, konstant viele oder $O(n)$ Synchronisationen erfordert, durch Suchen einer Abbildung vom Iterationsraum zu einer Partitionsnummer.

Raumpartitionen

Bedingung für Raumpartition

Besteht zwischen zwei Anweisungen eine Abhängigkeit, werden sie in die gleiche Partition abgebildet.

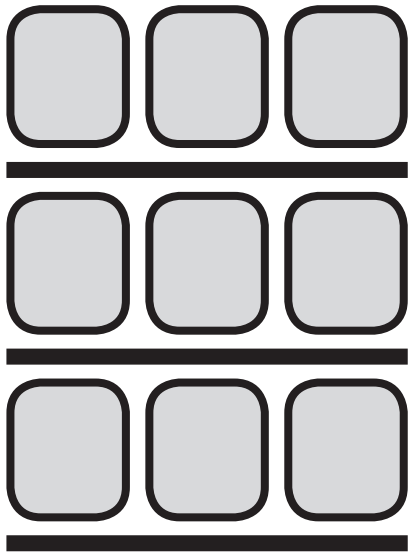


Synchronisationsfreie Parallelität!

Sonderfall / Zeitpartitionen

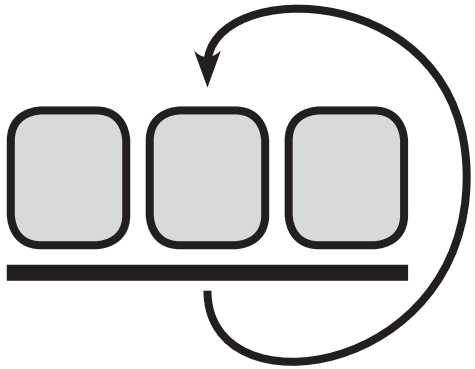
$O(1)$ Synchronisationen

Suche starke Zusammenhangskomponenten im Programmabhängigkeitsgraph. Wende Raumpartitionierung auf diese SCCs an.



$O(n)$ Synchronisationen

Bedingung für *Zeitpartition*: Alle Anweisungen in Zeitpartition i müssen vor denen in Zeitpartition $i + 1$ ausgeführt werden.



Wie funktioniert das?

Mit Standardverfahren der linearen Algebra.

Arrayzugriffe, Abhängigkeiten und Bedingungen für die Partitionen lassen sich als Gleichungssysteme darstellen. Die Partitionierung ist eine affine Abbildung, deren Koeffizientenmatrix ausgerechnet werden kann.

Zum Schluss: SPMD-Code generieren.

Probleme

Heutzutage nutzen Programme

- aufwändige Objektstrukturen im Heapspeicher statt linearer Arrays, und
- über Methodengrenzen verteilte Algorithmen anstatt geschachtelter Schleifen.

Zentrales Problem (immer noch)

Die Vorhersage der Speicherzugriffe!

Aktuelle Forschung zur Autoparallelisierung

Karlsruher Institut für Technologie

Abhängigkeitsanalyse durch Shape Analysis [7]

- Leitet Aussagen über den Aufbau von dynamischen Datenstrukturen her (z.B. Listen).
- Besser für Abhängigkeitsanalyse geeignet als Points-To-Analyse.

Trace-basierte Parallelisierung [3]

- Trace = Folge von paarweise verschiedenen Grundblöcken.
- Granularität zwischen Schleifen und Tasks.
- Microsoft Research untersucht einen tracing Just-in-Time Compiler [6].

Fazit

- Für numerische Anwendungen ist Autoparallelisierung machbar.
- Nur die vorhandenen Anweisungen, nicht der Algorithmus, können parallelisiert werden.
- Kein Ersatz für explizit parallel geschriebene Software.

Literatur

AHO, A. V., Ed.

Compiler : Prinzipien, Techniken und Werkzeuge, 2., aktualisierte aufl. ed.
it Informatik. Pearson Studium, München [u.a.], 2008, ch. 11, pp. 929–1090.

BACON, D. F., GRAHAM, S. L., AND SHARP, O. J.

Compiler transformations for high-performance computing.
ACM Comput. Surv. 26, 4 (1994), 345–420.

BRADEL, B. J., AND ABDELRAHMAN, T. S.

A study of potential parallelism among traces in java programs.
Science of Computer Programming 74, 5-6 (2009), 296 – 313.
Special Issue on Principles and Practices of Programming in Java (PPPJ 2007).

BURKE, M., AND CYTRON, R.

Interprocedural dependence analysis and parallelization.
In *SIGPLAN '86: Proceedings of the 1986 SIGPLAN symposium on Compiler construction* (New York, NY, USA, 1986), ACM, pp. 162–175.

LIM, A. W., AND LAM, M. S.

Maximizing parallelism and minimizing synchronization with affine transforms.
In *POPL '97: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (New York, NY, USA, 1997), ACM, pp. 201–214.

SCHULTE, W., AND TILLMANN, N.

Automatic parallelization of programming languages: Past, present und future.
http://www.ipd.uni-karlsruhe.de/multicore/iwmse2010/IWMSE-keynote-schulte.pdf, May 2010.
Extended abstract of the keynote presentation at the Third International Workshop on Multicore Software Engineering.

TINEO, A., CORBERA, F., NAVARRO, A., ASENJO, R., AND ZAPATA, E.

On the automatic detection of heap-induced data dependencies with interprocedural shape analysis.
In *Parallel Processing Workshops, 2009. ICPPW '09. International Conference on* (22-25 2009), pp. 378 –385.

Zusatzfolien

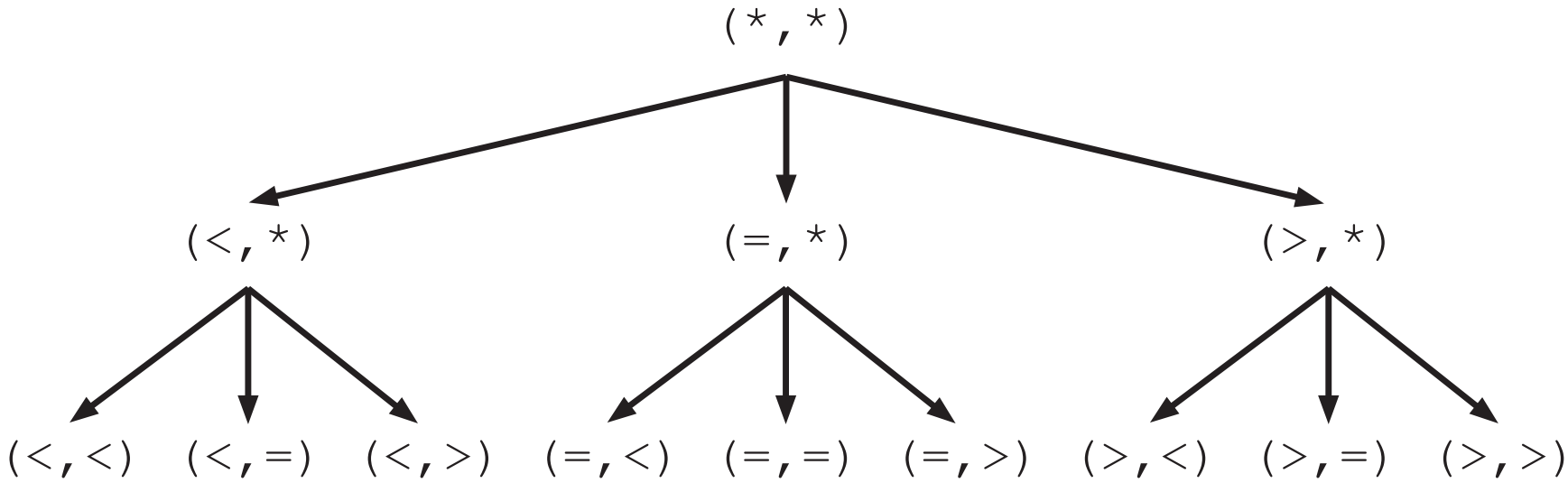
Karlsruher Institut für Technologie

SPMD-Code zu Einführungsbeispiel 2

Karlsruher Institut für Technologie

```
1  if (p == -99)
2    a[1,100] = a[1,100] + b[0,100];
3  if (-98 <= p && p <= 99) {
4    if (1 <= p && p <= 99) {
5      b[p,1] = a[p,0] * b[p,1];
6    }
7    for (l = max(1,1+p); l <= min(100,99+p); l++) {
8      a[l,l-p] = a[l,l-p] + b[l-1,l-p];
9      b[l,l-p+1] = a[l,l-p] * b[l, l-p+1];
10   }
11   if (-98 <= p && p <= 0) {
12     a[100+p,100] = a[100+p,100] + b[99+p,100];
13   }
14 }
15 if (p == 100)
16   b[100,1] = a[100,0] * b[100,1];
```

Hierarchische Abhängigkeitsvektoren



Beispiel zur Shape Analysis

```
1  l = p = create_list();
2  q = p->next;
3
4  while (q != NULL) {
5    @READ_1
6     val = q->data;
7
8    @WRITE_2
9    p->data = val;
10
11   p = q;
12   q = p->next;
13 }
```