Spekulation auf Fadenebene

Seminar: Multicore Programmierung

Edgar Kalkowski

Universität Passau
Lehrstuhl für Programmierung
Prof. Christian Lengauer

Donnerstag, 25. Juni 2009

Inhalt

Worum geht es heute?

- Multicore-Rechner werden zunehmend im Alltag verfügbar

Worum geht es heute?

- Multicore-Rechner werden zunehmend im Alltag verfügbar
- nicht mehr nur hochgradig spezialisierte mathematische Anwendungen, die z. B. an Unis auf „Supercomputern“ laufen, sollen Parallelität ausnutzen, sondern auch alltägliche Programme wie „Open Office“

Worum geht es heute?

- Multicore-Rechner werden zunehmend im Alltag verfügbar
- nicht mehr nur hochgradig spezialisierte mathematische Anwendungen, die z. B. an Unis auf „Supercomputern“ laufen, sollen Parallelität ausnutzen, sondern auch alltägliche Programme wie „Open Office“
- man kann diese Programm nicht alle von Hand parallelisieren, sondern das muss automatisch geschehen, z. B. durch den Compiler

Worum geht es heute?

- Multicore-Rechner werden zunehmend im Alltag verfügbar
- nicht mehr nur hochgradig spezialisierte mathematische Anwendungen, die z. B. an Unis auf „Supercomputern“ laufen, sollen Parallelität ausnutzen, sondern auch alltägliche Programme wie „Open Office“
- man kann diese Programm nicht alle von Hand parallelisieren, sondern das muss automatisch geschehen, z. B. durch den Compiler
- statische Analyse der Datenabhängigkeiten in alltäglichen Programmen mit komplexem Kontrollfluss durch den Compiler ist oft nicht möglich

Worum geht es heute?

- Multicore-Rechner werden zunehmend im Alltag verfügbar
- nicht mehr nur hochgradig spezialisierte mathematische Anwendungen, die z. B. an Unis auf „Supercomputern“ laufen, sollen Parallelität ausnutzen, sondern auch alltägliche Programme wie „Open Office“
- man kann diese Programm nicht alle von Hand parallelisieren, sondern das muss automatisch geschehen, z. B. durch den Compiler
- statische Analyse der Datenabhängigkeiten in alltäglichen Programmen mit komplexem Kontrollfluss durch den Compiler ist oft nicht möglich
- Lösung: Hoffen, dass keine Datenabhängigkeiten existieren und spekulative Ausführung mit mehreren Threads + Abbrechen und Rollback für Threads, die dynamisch zur Laufzeit erkannte Abhängigkeiten verletzen: „Thread Level Speculation“ (TLS)

Polytopmodell

- klassisches Verfahren zur Parallelisierung von numerischen Programmen (vgl. z. B. [Len93])

Polytopmodell

- klassisches Verfahren zur Parallelisierung von numerischen Programmen (vgl. z. B. [Len93])
- Darstellung einer d-fach geschachtelten Schleife als d-dimensionales Polytop

Polytopmodell

- klassisches Verfahren zur Parallelisierung von numerischen Programmen (vgl. z. B. [Len93])
- Darstellung einer d-fach geschachtelten Schleife als d-dimensionales Polytop
- statische Analyse von Datenabhängigkeiten zur Übersetzungszeit

Polytopmodell

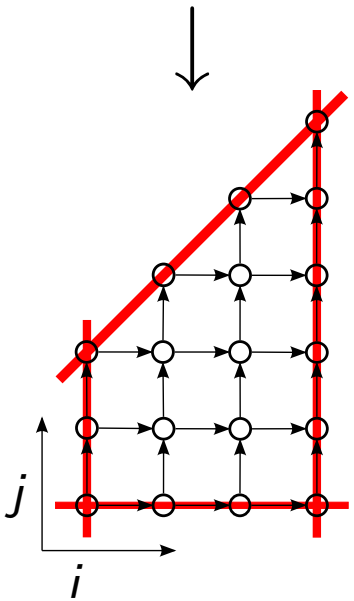
- klassisches Verfahren zur Parallelisierung von numerischen Programmen (vgl. z. B. [Len93])
- Darstellung einer d-fach geschachtelten Schleife als d-dimensionales Polytop
- statische Analyse von Datenabhängigkeiten zur Übersetzungszeit
- Transformation des Polytops und damit schließlich auch des Programmcodes derart, dass einige Dimensionen nur in Zeitrichtung und andere nur in Raum- (/Prozessor-) Richtung zeigen

Polytopmodell – Beispiel

```
for (int i = 0; i <= n; i++) {
  for (int j = 0; j <= i + 2; j++) {
    A[i][j] = A[i-1][j] + A[i][j-1];
  }
}
```

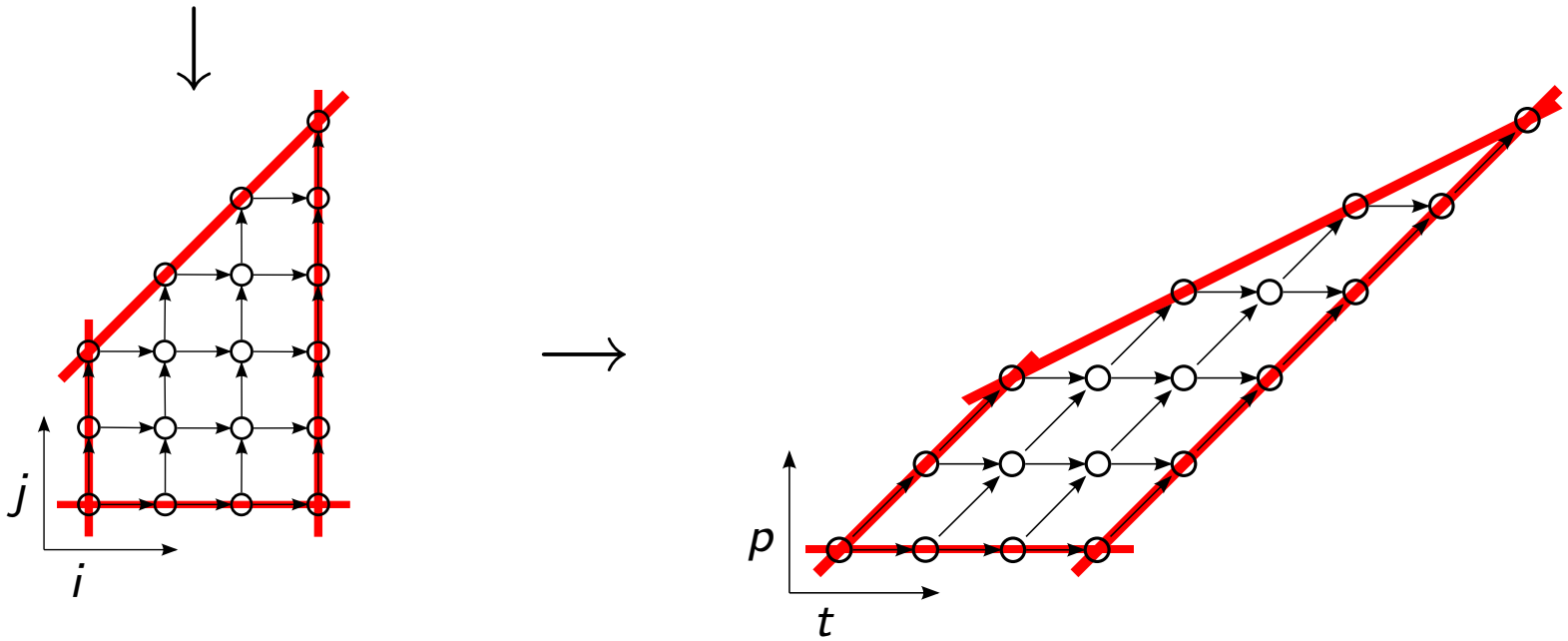
Polytopmodell – Beispiel

```
for (int i = 0; i <= n; i++) {
  for (int j = 0; j <= i + 2; j++) {
    A[i][j] = A[i-1][j] + A[i][j-1];
  }
}
```



Polytopmodell – Beispiel

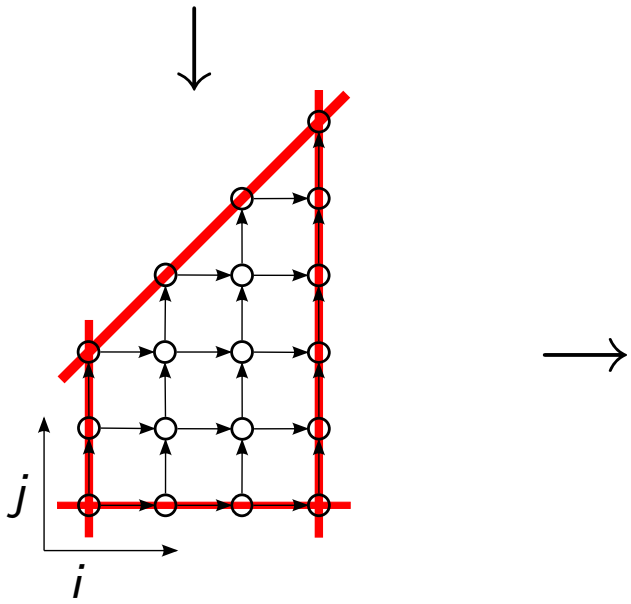
```
for (int i = 0; i <= n; i++) {
  for (int j = 0; j <= i + 2; j++) {
    A[i][j] = A[i-1][j] + A[i][j-1];
  }
}
```

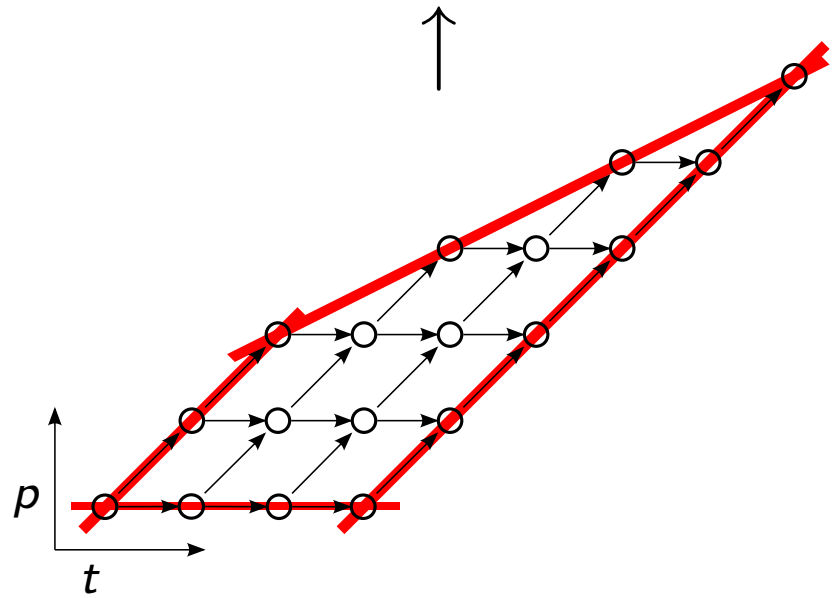


Polytopmodell – Beispiel

```
for (int i = 0; i <= n; i++) {
  for (int j = 0; j <= i + 2; j++) {
    A[i][j] = A[i-1][j] + A[i][j-1];
  }
}
```

```
for (int t = 0; t <= 2*n + 2; t++) {
  parallel for (int p = max(0, t-n);
        t <= min(t, t/2 + 1); t++) {
    A[t-p][p] = A[t-p-1][p] + A[t-p
        ][p-1];
  }
}
```





Polytopmodell – Bewertung

- Vorteile: Parallelisierung erfolgt komplett automatisch und zur Übersetzungszeit, mathematisch exakt

Polytopmodell – Bewertung

- Vorteile: Parallelisierung erfolgt komplett automatisch und zur Übersetzungszeit, mathematisch exakt
- Nachteile: starke Einschränkungen bezüglich des Quellprogramms:

Polytopmodell – Bewertung

- Vorteile: Parallelisierung erfolgt komplett automatisch und zur Übersetzungszeit, mathematisch exakt
- Nachteile: starke Einschränkungen bezüglich des Quellprogramms:
  - nur *perfekt geschachtelte* Schleifen sind erlaubt (eine Erweiterung auf nicht perfekt geschachtelte Schleifen ist möglich, vgl. z. B. [Fea91])

Polytopmodell – Bewertung

- Vorteile: Parallelisierung erfolgt komplett automatisch und zur Übersetzungszeit, mathematisch exakt
- Nachteile: starke Einschränkungen bezüglich des Quellprogramms:
  - nur *perfekt geschachtelte* Schleifen sind erlaubt (eine Erweiterung auf nicht perfekt geschachtelte Schleifen ist möglich, vgl. z. B. [Fea91])
  - die einzige Datenstruktur, die in der innersten Schleife verwendet werden darf, ist das Array

Polytopmodell – Bewertung

- Vorteile: Parallelisierung erfolgt komplett automatisch und zur Übersetzungszeit, mathematisch exakt
- Nachteile: starke Einschränkungen bezüglich des Quellprogramms:
  - nur *perfekt geschachtelte* Schleifen sind erlaubt (eine Erweiterung auf nicht perfekt geschachtelte Schleifen ist möglich, vgl. z. B. [Fea91])
  - die einzige Datenstruktur, die in der innersten Schleife verwendet werden darf, ist das Array
  - die Schleifengrenzen und Array-Indizes müssen über die Indizes der äußeren Schleifen und über Konstanten ausgedrückt werden

Polytopmodell – Bewertung

- Vorteile: Parallelisierung erfolgt komplett automatisch und zur Übersetzungszeit, mathematisch exakt
- Nachteile: starke Einschränkungen bezüglich des Quellprogramms:
  - nur *perfekt geschachtelte* Schleifen sind erlaubt (eine Erweiterung auf nicht perfekt geschachtelte Schleifen ist möglich, vgl. z. B. [Fea91])
  - die einzige Datenstruktur, die in der innersten Schleife verwendet werden darf, ist das Array
  - die Schleifengrenzen und Array-Indizes müssen über die Indizes der äußeren Schleifen und über Konstanten ausgedrückt werden
  - die Datenabhängigkeiten zwischen verschiedenen Schleifendurchläufen müssen bei allen Iterationen uniform sein (Erweiterung auf affin lineare Abhängigkeiten ist möglich, vgl. z. B. [Gri96])

Polytopmodell – Bewertung

- Vorteile: Parallelisierung erfolgt komplett automatisch und zur Übersetzungszeit, mathematisch exakt
- Nachteile: starke Einschränkungen bezüglich des Quellprogramms:
  - nur *perfekt geschachtelte* Schleifen sind erlaubt (eine Erweiterung auf nicht perfekt geschachtelte Schleifen ist möglich, vgl. z. B. [Fea91])
  - die einzige Datenstruktur, die in der innersten Schleife verwendet werden darf, ist das Array
  - die Schleifengrenzen und Array-Indizes müssen über die Indizes der äußeren Schleifen und über Konstanten ausgedrückt werden
  - die Datenabhängigkeiten zwischen verschiedenen Schleifendurchläufen müssen bei allen Iterationen uniform sein (Erweiterung auf affin lineare Abhängigkeiten ist möglich, vgl. z. B. [Gri96])
  - im ursprünglichen Modell können lediglich `for`-Schleifen parallelisiert werden

Erweiterung auf `while`-Schleifen

- Problem bei `while`-Schleifen: Obergrenze der Indexmenge ist zur Übersetzungszeit und möglicherweise auch zur Laufzeit unbekannt

Erweiterung auf `while`-Schleifen

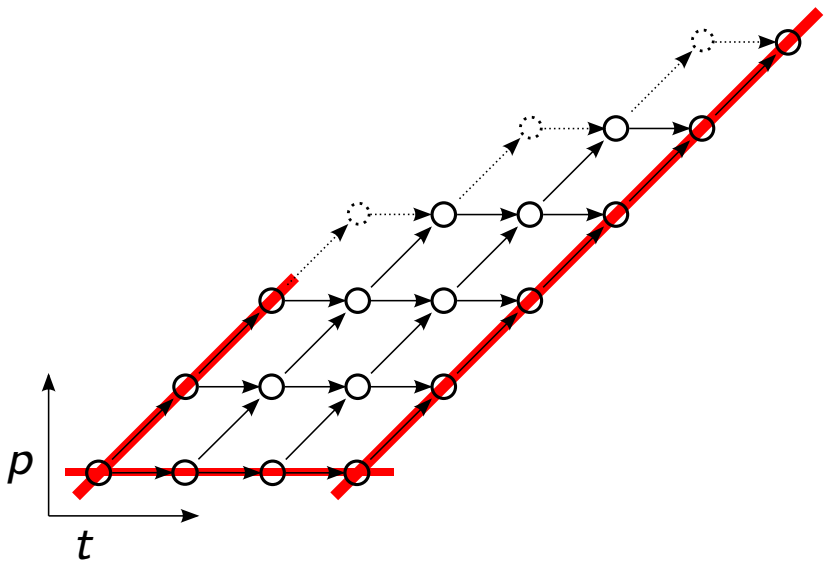
- Problem bei `while`-Schleifen: Obergrenze der Indexmenge ist zur Übersetzungszeit und möglicherweise auch zur Laufzeit unbekannt
- Konsequenz: keine Abbildung auf ein Polytop mehr, sondern auf ein Polyeder: in den Dimensionen der `while`-Schleifen unbegrenzt

Erweiterung auf `while`-Schleifen

- Problem bei `while`-Schleifen: Obergrenze der Indexmenge ist zur Übersetzungszeit und möglicherweise auch zur Laufzeit unbekannt
- Konsequenz: keine Abbildung auf ein Polytop mehr, sondern auf ein Polyeder: in den Dimensionen der `while`-Schleifen unbegrenzt
- zur Laufzeit muss verhindert werden, dass zu viele Iterationen der `while`-Schleife ausgeführt werden

Erweiterung auf `while`-Schleifen

- Problem bei `while`-Schleifen: Obergrenze der Indexmenge ist zur Übersetzungszeit und möglicherweise auch zur Laufzeit unbekannt
- Konsequenz: keine Abbildung auf ein Polytop mehr, sondern auf ein Polyeder: in den Dimensionen der `while`-Schleifen unbegrenzt
- zur Laufzeit muss verhindert werden, dass zu viele Iterationen der `while`-Schleife ausgeführt werden



`while`-Schleifen – Konservativer Ansatz

- es werden nur Iterationen ausgeführt, die auch im sequentiellen Programm ausgeführt würden

`while`-Schleifen – Konservativer Ansatz

- es werden nur Iterationen ausgeführt, die auch im sequentiellen Programm ausgeführt würden
- Konsequenz: jede Iteration muss warten, bis die vorherige Iteration fertig ist und die (dynamische) Schleifenbedingung überprüft wurde

`while`-Schleifen – Konservativer Ansatz

- es werden nur Iterationen ausgeführt, die auch im sequentiellen Programm ausgeführt würden
- Konsequenz: jede Iteration muss warten, bis die vorherige Iteration fertig ist und die (dynamische) Schleifenbedingung überprüft wurde
- Nachteil: falls unterschiedliche Iterationen auf verschiedenen Prozessoren ausgeführt werden, muss nach jeder Iteration kommuniziert werden

`while`-Schleifen – Konservativer Ansatz

- es werden nur Iterationen ausgeführt, die auch im sequentiellen Programm ausgeführt würden
- Konsequenz: jede Iteration muss warten, bis die vorherige Iteration fertig ist und die (dynamische) Schleifenbedingung überprüft wurde
- Nachteil: falls unterschiedliche Iterationen auf verschiedenen Prozessoren ausgeführt werden, muss nach jeder Iteration kommuniziert werden
- Vorteil: praktisch kein Speicheroverhead

`while`-Schleifen – Spekulativer Ansatz

- führe (unter Beachtung der Datenabhängigkeiten zwischen den verschiedenen Durchläufen) spekulativ so viele Iterationen der `while`-Schleife aus, wie Prozessoren vorhanden sind

`while`-Schleifen – Spekulativer Ansatz

- führe (unter Beachtung der Datenabhängigkeiten zwischen den verschiedenen Durchläufen) spekulativ so viele Iterationen der `while`-Schleife aus, wie Prozessoren vorhanden sind
- sobald die Abbruchbedingung erreicht wurde, verwerfe die Ergebnisse aller „zu viel“ durchgeführten Iterationen

`while`-Schleifen – Spekulativer Ansatz

- führe (unter Beachtung der Datenabhängigkeiten zwischen den verschiedenen Durchläufen) spekulativ so viele Iterationen der `while`-Schleife aus, wie Prozessoren vorhanden sind
- sobald die Abbruchbedingung erreicht wurde, verwerfe die Ergebnisse aller „zu viel“ durchgeführten Iterationen
- Konsequenz: es müssen die Ergebnisse jeder Iteration separat abgespeichert werden, damit am Schluss das korrekte Ergebnis übernommen werden kann

`while`-Schleifen – Spekulativer Ansatz

- führe (unter Beachtung der Datenabhängigkeiten zwischen den verschiedenen Durchläufen) spekulativ so viele Iterationen der `while`-Schleife aus, wie Prozessoren vorhanden sind
- sobald die Abbruchbedingung erreicht wurde, verwerfe die Ergebnisse aller „zu viel“ durchgeführten Iterationen
- Konsequenz: es müssen die Ergebnisse jeder Iteration separat abgespeichert werden, damit am Schluss das korrekte Ergebnis übernommen werden kann
- Vorteile: geringere Kommunikationskosten, mehr Parallelität

`while`-Schleifen – Spekulativer Ansatz

- führe (unter Beachtung der Datenabhängigkeiten zwischen den verschiedenen Durchläufen) spekulativ so viele Iterationen der `while`-Schleife aus, wie Prozessoren vorhanden sind
- sobald die Abbruchbedingung erreicht wurde, verwerfe die Ergebnisse aller „zu viel“ durchgeführten Iterationen
- Konsequenz: es müssen die Ergebnisse jeder Iteration separat abgespeichert werden, damit am Schluss das korrekte Ergebnis übernommen werden kann
- Vorteile: geringere Kommunikationskosten, mehr Parallelität
- Nachteile: hoher Speicheroverhead, manche Berechnungen sind „umsonst“

Problemstellung

- durch Erweiterungs des spekulativen Ansatzes zur Parallelisierung von `while`-Schleifen sollen weitere Einschränkungen der bisher vorgestellten Verfahren überwunden werden

Problemstellung

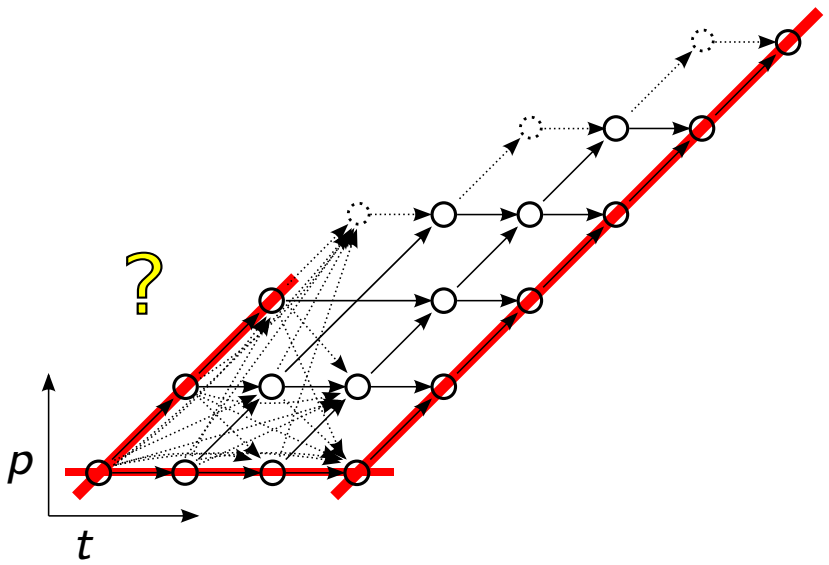
- durch Erweiterungs des spekulativen Ansatzes zur Parallelisierung von `while`-Schleifen sollen weitere Einschränkungen der bisher vorgestellten Verfahren überwunden werden
- insbesondere kein Einschränkung mehr auf Arrays, sondern komplexe Datenstrukturen mit komplexen Speicherzugriffen

Problemstellung

- durch Erweiterungs des spekulativen Ansatzes zur Parallelisierung von `while`-Schleifen sollen weitere Einschränkungen der bisher vorgestellten Verfahren überwunden werden
- insbesondere kein Einschränkung mehr auf Arrays, sondern komplexe Datenstrukturen mit komplexen Speicherzugriffen
- beliebige, dynamische Schleifengrenzen, komplexer Kontrollfluss

Problemstellung

- durch Erweiterungs des spekulativen Ansatzes zur Parallelisierung von `while`-Schleifen sollen weitere Einschränkungen der bisher vorgestellten Verfahren überwunden werden
- insbesondere kein Einschränkung mehr auf Arrays, sondern komplexe Datenstrukturen mit komplexen Speicherzugriffen
- beliebige, dynamische Schleifengrenzen, komplexer Kontrollfluss



Problemstellung (2)

- eine statische Analyse der Datenabhängigkeiten zwischen verschiedenen Iterationen zur Übersetzungszeit ist hier unmöglich

Problemstellung (2)

- eine statische Analyse der Datenabhängigkeiten zwischen verschiedenen Iterationen zur Übersetzungszeit ist hier unmöglich
- Also: spekulative Ausführung von aufeinanderfolgenden Iterationen in mehreren Threads und Analyse von Datenabhängigkeiten zur Laufzeit

Problemstellung (2)

- eine statische Analyse der Datenabhängigkeiten zwischen verschiedenen Iterationen zur Übersetzungszeit ist hier unmöglich
- Also: spekulative Ausführung von aufeinanderfolgenden Iterationen in mehreren Threads und Analyse von Datenabhängigkeiten zur Laufzeit
- bei Verletzung einer Datenabhängigkeit: Rollback und Verwerfen aller auf der Datenabhängigkeit basierenden Ergebnisse

Herausforderungen

- Optimierung des Programms durch den Compiler, z. B. um Datenabhängigkeiten möglichst zu vermeiden (Erkennen von Reduktionen und Induktionen)

Herausforderungen

- Optimierung des Programms durch den Compiler, z. B. um Datenabhängigkeiten möglichst zu vermeiden (Erkennen von Reduktionen und Induktionen)
- Finden von spekulativen Regionen (z. B. Schleifen), die parallelisiert werden sollen (z. B. mit Hilfe von Profilern)

Herausforderungen

- Optimierung des Programms durch den Compiler, z. B. um Datenabhängigkeiten möglichst zu vermeiden (Erkennen von Reduktionen und Induktionen)
- Finden von spekulativen Regionen (z. B. Schleifen), die parallelisiert werden sollen (z. B. mit Hilfe von Profilern)
- Aufteilen dieser Regionen in einzelne Epochen, die später jeweils von einem Thread ausgeführt werden (kann ein einzelner Schleifendurchlauf sein oder auch mehrere aufeinanderfolgende Iterationen)

Herausforderungen

- Optimierung des Programms durch den Compiler, z. B. um Datenabhängigkeiten möglichst zu vermeiden (Erkennen von Reduktionen und Induktionen)
- Finden von spekulativen Regionen (z. B. Schleifen), die parallelisiert werden sollen (z. B. mit Hilfe von Profilern)
- Aufteilen dieser Regionen in einzelne Epochen, die später jeweils von einem Thread ausgeführt werden (kann ein einzelner Schleifendurchlauf sein oder auch mehrere aufeinanderfolgende Iterationen)
- Threadmanagement

Herausforderungen

- Optimierung des Programms durch den Compiler, z. B. um Datenabhängigkeiten möglichst zu vermeiden (Erkennen von Reduktionen und Induktionen)
- Finden von spekulativen Regionen (z. B. Schleifen), die parallelisiert werden sollen (z. B. mit Hilfe von Profilern)
- Aufteilen dieser Regionen in einzelne Epochen, die später jeweils von einem Thread ausgeführt werden (kann ein einzelner Schleifendurchlauf sein oder auch mehrere aufeinanderfolgende Iterationen)
- Threadmanagement
- kein direktes Schreiben in den Speicher, sondern Puffern der Schreibzugriffe pro Thread

Herausforderungen

- Optimierung des Programms durch den Compiler, z. B. um Datenabhängigkeiten möglichst zu vermeiden (Erkennen von Reduktionen und Induktionen)
- Finden von spekulativen Regionen (z. B. Schleifen), die parallelisiert werden sollen (z. B. mit Hilfe von Profilern)
- Aufteilen dieser Regionen in einzelne Epochen, die später jeweils von einem Thread ausgeführt werden (kann ein einzelner Schleifendurchlauf sein oder auch mehrere aufeinanderfolgende Iterationen)
- Threadmanagement
- kein direktes Schreiben in den Speicher, sondern Puffern der Schreibzugriffe pro Thread
- Verhindern, dass veraltete Werte gelesen werden

Herausforderungen

- Optimierung des Programms durch den Compiler, z. B. um Datenabhängigkeiten möglichst zu vermeiden (Erkennen von Reduktionen und Induktionen)
- Finden von spekulativen Regionen (z. B. Schleifen), die parallelisiert werden sollen (z. B. mit Hilfe von Profilern)
- Aufteilen dieser Regionen in einzelne Epochen, die später jeweils von einem Thread ausgeführt werden (kann ein einzelner Schleifendurchlauf sein oder auch mehrere aufeinanderfolgende Iterationen)
- Threadmanagement
- kein direktes Schreiben in den Speicher, sondern Puffern der Schreibzugriffe pro Thread
- Verhindern, dass veraltete Werte gelesen werden
- global in einem konsistentem Zustand bleiben

PolyLibTLS

- objektorientierte Bibliothek in C++

PolyLibTLS

- objektorientierte Bibliothek in C++
- Ersetzen von Lese- und Schreiboperationen durch die Funktionsaufrufe `specLD()` und `specST()`, die für die Konsistenz des Speichers sorgen

PolyLibTLS

- objektorientierte Bibliothek in C++
- Ersetzen von Lese- und Schreiboperationen durch die Funktionsaufrufe `specLD()` und `specST()`, die für die Konsistenz des Speichers sorgen
- `specST()`: jeder Thread puffert seine Schreibergebnisse zunächst lokal

PolyLibTLS

- objektorientierte Bibliothek in C++
- Ersetzen von Lese- und Schreiboperationen durch die Funktionsaufrufe `specLD()` und `specST()`, die für die Konsistenz des Speichers sorgen
- `specST()`: jeder Thread puffert seine Schreibergebnisse zunächst lokal
- `specLD()`: merkt sich global für jede Speicherzelle, welche Epoche/Iteration sie zuletzt gelesen hat und überprüft, ob die Speicherzelle möglicherweise lokal überschrieben wurde

PolyLibTLS

- objektorientierte Bibliothek in C++
- Ersetzen von Lese- und Schreiboperationen durch die Funktionsaufrufe `specLD()` und `specST()`, die für die Konsistenz des Speichers sorgen
- `specST()`: jeder Thread puffert seine Schreibergebnisse zunächst lokal
- `specLD()`: merkt sich global für jede Speicherzelle, welche Epoche/Iteration sie zuletzt gelesen hat und überprüft, ob die Speicherzelle möglicherweise lokal überschrieben wurde
- die Epoche mit dem jeweils ältesten Index darf ihre Ergebnisse in den globalen Speicher schreiben

PolyLibTLS – Rollback

Thread 4

```
...
STORE 34, 2
...
```

Thread 5

```
LOAD %1, 34
...
...
```

PolyLibTLS – Rollback

Thread 4

```
...
STORE 34, 2
...
```

Thread 5

```
LOAD %1, 34
...
...
```

- Problem: RAW-Hazard, falls zuerst Thread 5 liest und dann Thread 4 schreibt

PolyLibTLS – Rollback

Thread 4

```
...
STORE 34, 2
...
```

Thread 5

```
LOAD %1, 34
...
...
```

- Problem: RAW-Hazard, falls zuerst Thread 5 liest und dann Thread 4 schreibt
- Lösung: beim Schreiben seiner Ergebnisse in den globalen Speicher überprüft Thread 4 für jede zu schreibende Speicherzelle, ob keine spätere Epoche die Speicherzelle gelesen hat

PolyLibTLS – Rollback

Thread 4

```
...
STORE 34, 2
...
```

Thread 5

```
LOAD %1, 34
...
...
```

- Problem: RAW-Hazard, falls zuerst Thread 5 liest und dann Thread 4 schreibt
- Lösung: beim Schreiben seiner Ergebnisse in den globalen Speicher überprüft Thread 4 für jede zu schreibende Speicherzelle, ob keine spätere Epoche die Speicherzelle gelesen hat
- falls nein: Schreiben des Werts aus dem lokalen Puffer in den globalen Speicher

PolyLibTLS – Rollback

Thread 4

```
...
STORE 34, 2
...
```

Thread 5

```
LOAD %1, 34
...
...
```

- Problem: RAW-Hazard, falls zuerst Thread 5 liest und dann Thread 4 schreibt
- Lösung: beim Schreiben seiner Ergebnisse in den globalen Speicher überprüft Thread 4 für jede zu schreibende Speicherzelle, ob keine spätere Epoche die Speicherzelle gelesen hat
- falls nein: Schreiben des Werts aus dem lokalen Puffer in den globalen Speicher
- falls ja: Rollback und Verwerfen und Neustarten aller späteren Epochen

PolyLibTLS – Reduzierung des Speicheroverheads

- großer Speicheroverhead, da alle Schreiboperationen zunächst gepuffert werden müssen und zusätzlich pro Speicherzelle ein Integer den Index der letzten lesenden Epoche speichert

PolyLibTLS – Reduzierung des Speicheroverheads

- großer Speicheroverhead, da alle Schreiboperationen zunächst gepuffert werden müssen und zusätzlich pro Speicherzelle ein Integer den Index der letzten lesenden Epoche speichert
- Reduzierung des Overheads:

PolyLibTLS – Reduzierung des Speicheroverheads

- großer Speicheroverhead, da alle Schreiboperationen zunächst gepuffert werden müssen und zusätzlich pro Speicherzelle ein Integer den Index der letzten lesenden Epoche speichert
- Reduzierung des Overheads:
  - letzter Ladezugriff wird nicht für jede Speicherzelle gespeichert, sondern (mehrfach assoziativ) über eine Hash-Funktion nur ein Wert für mehrere Speicherzellen

PolyLibTLS – Reduzierung des Speicheroverheads

- großer Speicheroverhead, da alle Schreiboperationen zunächst gepuffert werden müssen und zusätzlich pro Speicherzelle ein Integer den Index der letzten lesenden Epoche speichert
- Reduzierung des Overheads:
  - letzter Ladezugriff wird nicht für jede Speicherzelle gespeichert, sondern (mehrfach assoziativ) über eine Hash-Funktion nur ein Wert für mehrere Speicherzellen
    - kann überflüssige Rollbacks verursachen

PolyLibTLS – Reduzierung des Speicheroverheads

- großer Speicheroverhead, da alle Schreiboperationen zunächst gepuffert werden müssen und zusätzlich pro Speicherzelle ein Integer den Index der letzten lesenden Epoche speichert
- Reduzierung des Overheads:
  - letzter Ladezugriff wird nicht für jede Speicherzelle gespeichert, sondern (mehrfach assoziativ) über eine Hash-Funktion nur ein Wert für mehrere Speicherzellen
    - kann überflüssige Rollbacks verursachen
  - PolyLibTLS bietet neben dem vorgestellten noch weitere Speichermodelle, z.B. ein Read-Only-Modell

PolyLibTLS – Reduzierung des Speicheroverheads

- großer Speicheroverhead, da alle Schreiboperationen zunächst gepuffert werden müssen und zusätzlich pro Speicherzelle ein Integer den Index der letzten lesenden Epoche speichert
- Reduzierung des Overheads:
  - letzter Ladezugriff wird nicht für jede Speicherzelle gespeichert, sondern (mehrfach assoziativ) über eine Hash-Funktion nur ein Wert für mehrere Speicherzellen
    - kann überflüssige Rollbacks verursachen
  - PolyLibTLS bietet neben dem vorgestellten noch weitere Speichermodelle, z.B. ein Read-Only-Modell
  - dabei werden Werte genau wie bisher gelesen, aber sobald ein Wert geschrieben werden soll, bricht der Thread ab und es wird ein Rollback durchgeführt

PolyLibTLS – Reduzierung des Speicheroverheads

- großer Speicheroverhead, da alle Schreiboperationen zunächst gepuffert werden müssen und zusätzlich pro Speicherzelle ein Integer den Index der letzten lesenden Epoche speichert
- Reduzierung des Overheads:
  - letzter Ladezugriff wird nicht für jede Speicherzelle gespeichert, sondern (mehrfach assoziativ) über eine Hash-Funktion nur ein Wert für mehrere Speicherzellen
    - kann überflüssige Rollbacks verursachen
  - PolyLibTLS bietet neben dem vorgestellten noch weitere Speichermodelle, z.B. ein Read-Only-Modell
  - dabei werden Werte genau wie bisher gelesen, aber sobald ein Wert geschrieben werden soll, bricht der Thread ab und es wird ein Rollback durchgeführt
  - spart den Puffer für Schreiboperationen, kann aber auch überflüssige Rollbacks verursachen

PolyLibTLS – Beispiel

```
for (int i = 0; i < N; i++) {
  double sum = 0.0;
  for (int j = 0; j < M; j++)
    sum += A[i][j];
  sum = 1.0 / (1.0 + exp(-sum));
  B[i] += sum;
}
```

PolyLibTLS – Beispiel

```
for (int i = 0; i < N; i++) {
  double sum = 0.0;
  for (int j = 0; j < M; j++)
    sum += A[i][j];
  sum = 1.0 / (1.0 + exp(-sum));
  B[i] += sum;
}
```

```
SpRO spA(A, M * N);
SpSC spB(B, N);
USpM<SpRO, &spA, SpSC, &spB> spU;
...
double sum = 0.0;
for (int j = 0; j < M; j++)
  sum += spU.specLD<SpRO, &spA>(&A[i][j]);
sum = 1.0 / (1.0 + exp(-sum));
spU.specST<SpSC, &spB>(&B[i], spU.specLD<SpSC, &spB>(&B[i]) + sum);
```

PolyLibTLS – Speedup

Seq/Spec	Hand Par	SpLSC	+SpRO
IDEA Cipher	3.91	2.25	3.22
IDEA DeKey	3.89	1.85	3.09
SparMatMult	2.11	1.25	2.00
NeuralNetFW	2.04	1.19	1.90
NeuralNetBW	1.52	0.75	1.26
FFTtransf	1.99	0.83	0.83

PolyLibTLS – Bewertung

- komplette Software-Implementierung von TLS, läuft auf bestehender Hardware

PolyLibTLS – Bewertung

- komplette Software-Implementierung von TLS, läuft auf bestehender Hardware
- Code muss bisher noch von Hand nach spekulativen Regionen durchsucht werden und die Bibliotheksfunktionen müssen von Hand eingebaut werden

PolyLibTLS – Bewertung

- komplette Software-Implementierung von TLS, läuft auf bestehender Hardware
- Code muss bisher noch von Hand nach spekulativen Regionen durchsucht werden und die Bibliotheksfunktionen müssen von Hand eingebaut werden
- Code wird relativ unübersichtlich, Gegenargument der Entwickler: der Programmierer muss sich gar nicht um alles kümmern und nicht alles verstehen (z. B. Threadmanagement oder Speichermodelle)

PolyLibTLS – Bewertung

- komplette Software-Implementierung von TLS, läuft auf bestehender Hardware
- Code muss bisher noch von Hand nach spekulativen Regionen durchsucht werden und die Bibliotheksfunktionen müssen von Hand eingebaut werden
- Code wird relativ unübersichtlich, Gegenargument der Entwickler: der Programmierer muss sich gar nicht um alles kümmern und nicht alles verstehen (z. B. Threadmanagement oder Speichermodelle)
- durch verschiedene Speichermodelle kann man den Tradeoff zwischen überflüssigen Rollbacks und erhöhter Performanz beeinflussen

andere Software-Implementierungen

- bieten oft nicht die Flexibilität hinsichtlich der Speichermodelle von PolyLibTLS

andere Software-Implementierungen

- bieten oft nicht die Flexibilität hinsichtlich der Speichermodelle von PolyLibTLS
- versuchen auf anderem Weg, die Performanz zu erhöhen, z. B. Forwarding von Daten zwischen aufeinanderfolgenden Iterationen, damit diese zumindest teilweise parallel ausgeführt werden können

Übersicht

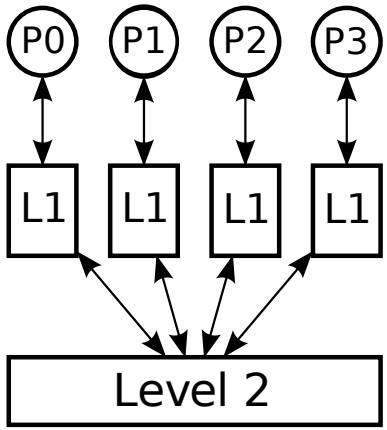
- Problemstellung ist dieselbe wie bei Software-Umsetzung: spekulative Parallelisierung komplexer Programme und Erkennung von verletzten Datenabhängigkeiten zur Laufzeit

Übersicht

- Problemstellung ist dieselbe wie bei Software-Umsetzung: spekulative Parallelisierung komplexer Programme und Erkennung von verletzten Datenabhängigkeiten zur Laufzeit
- dazu: Modifikation des Cache-Kohärenz-Protokolls, das sowieso von jedem Multicore implementiert wird

Übersicht

- Problemstellung ist dieselbe wie bei Software-Umsetzung: spekulative Parallelisierung komplexer Programme und Erkennung von verletzten Datenabhängigkeiten zur Laufzeit
- dazu: Modifikation des Cache-Kohärenz-Protokolls, das sowieso von jedem Multicore implementiert wird
- abstrakte Architektur ist dabei: ein privater Level-1-Cache pro Prozessor und ein gemeinsamer Level-2-Cache



Erweiterung des Cache-Protokolls

- typische Flags eines Cache-Protokolls für eine Cache-Line

Erweiterung des Cache-Protokolls

- typische Flags eines Cache-Protokolls für eine Cache-Line
  - `I:` Cache-Line ist ungültig/wird aktuell nicht verwendet

Erweiterung des Cache-Protokolls

- typische Flags eines Cache-Protokolls für eine Cache-Line
  - `I`: Cache-Line ist ungültig/wird aktuell nicht verwendet
  - `D`: Cache-Line wurde geschrieben, muss aber noch in den anderen Caches aktualisiert werden

Erweiterung des Cache-Protokolls

- typische Flags eines Cache-Protokolls für eine Cache-Line
  - `I`: Cache-Line ist ungültig/wird aktuell nicht verwendet
  - `D`: Cache-Line wurde geschrieben, muss aber noch in den anderen Caches aktualisiert werden
  - `E`: Cache-Line befindet sich nur in diesem Cache

Erweiterung des Cache-Protokolls

- typische Flags eines Cache-Protokolls für eine Cache-Line
  - `I`: Cache-Line ist ungültig/wird aktuell nicht verwendet
  - `D`: Cache-Line wurde geschrieben, muss aber noch in den anderen Caches aktualisiert werden
  - `E`: Cache-Line befindet sich nur in diesem Cache
  - `S`: Cache-Line befindet sich möglicherweise noch in anderen Caches

Erweiterung des Cache-Protokolls

- typische Flags eines Cache-Protokolls für eine Cache-Line
  - `I`: Cache-Line ist ungültig/wird aktuell nicht verwendet
  - `D`: Cache-Line wurde geschrieben, muss aber noch in den anderen Caches aktualisiert werden
  - `E`: Cache-Line befindet sich nur in diesem Cache
  - `S`: Cache-Line befindet sich möglicherweise noch in anderen Caches
- zusätzliche Flags

Erweiterung des Cache-Protokolls

- typische Flags eines Cache-Protokolls für eine Cache-Line
  - `I`: Cache-Line ist ungültig/wird aktuell nicht verwendet
  - `D`: Cache-Line wurde geschrieben, muss aber noch in den anderen Caches aktualisiert werden
  - `E`: Cache-Line befindet sich nur in diesem Cache
  - `S`: Cache-Line befindet sich möglicherweise noch in anderen Caches
- zusätzliche Flags
  - `SpecLD`: Cache-Line wurde spekulativ gelesen

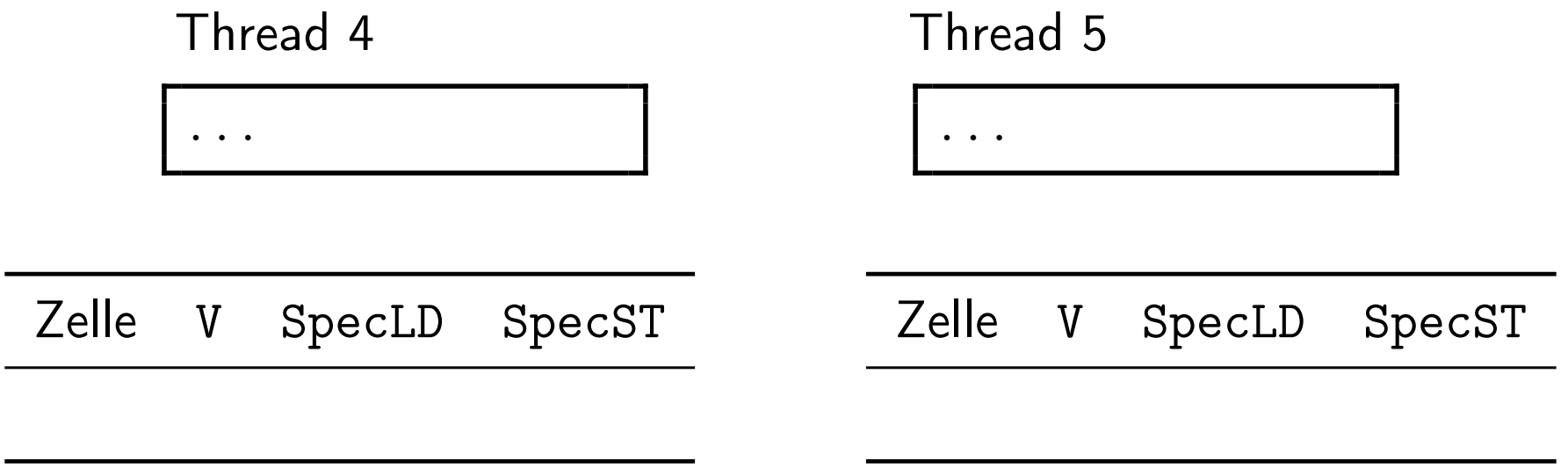
Erweiterung des Cache-Protokolls

- typische Flags eines Cache-Protokolls für eine Cache-Line
  - `I`: Cache-Line ist ungültig/wird aktuell nicht verwendet
  - `D`: Cache-Line wurde geschrieben, muss aber noch in den anderen Caches aktualisiert werden
  - `E`: Cache-Line befindet sich nur in diesem Cache
  - `S`: Cache-Line befindet sich möglicherweise noch in anderen Caches
- zusätzliche Flags
  - `SpecLD`: Cache-Line wurde spekulativ gelesen
  - `SpecST`: Cache-Line wurde spekulativ überschrieben

Erweiterung des Cache-Protokolls

- typische Flags eines Cache-Protokolls für eine Cache-Line
  - `I`: Cache-Line ist ungültig/wird aktuell nicht verwendet
  - `D`: Cache-Line wurde geschrieben, muss aber noch in den anderen Caches aktualisiert werden
  - `E`: Cache-Line befindet sich nur in diesem Cache
  - `S`: Cache-Line befindet sich möglicherweise noch in anderen Caches
- zusätzliche Flags
  - `SpecLD`: Cache-Line wurde spekulativ gelesen
  - `SpecST`: Cache-Line wurde spekulativ überschrieben
  - `V`: Cache-Line hat eine Abhängigkeit verletzt

Funktionsweise



- oben die jeweils aktuellen Befehle der beiden Threads
- darunter die jeweiligen Level-1-Caches

Funktionsweise

Thread 4

```
...
```

Thread 5

```
LOAD %1, 34
```

Zelle	V	SpecLD	SpecST

Zelle	V	SpecLD	SpecST
34	F	T	F

- Thread 5 lädt einen Wert
- entsprechende Cache-Line wird als SpecLD markiert

Funktionsweise

Thread 4

```
STORE 34, 2
```

Thread 5

```
...
```

Zelle	V	SpecLD	SpecST
34	F	T	T

Zelle	V	SpecLD	SpecST
34	F	T	F

- Thread 4 überschreibt denselben Wert im lokalen Cache
- entsprechende Cache-Line wird als SpecST markiert

Funktionsweise

Thread 4

```
commit()
```

Thread 5

```
...
```

Zelle	V	SpecLD	SpecST
34	F	T	T

Zelle	V	SpecLD	SpecST
34	T	T	F

- Thread 4 hat den niedrigsten Index und darf seine Ergebnisse schreiben
- dabei wird eine Abhängigkeitsverletzung festgestellt (Thread 5 hat Zelle 34 spekulativ geladen) und der Eintrag im Cache von Thread 5 entsprechend markiert

Funktionsweise

Thread 5

```
commit()
```

Zelle	V	SpecLD	SpecST
34	T	T	F

- Thread 5 versucht zu committen und scheitert and der markierten Verletzung einer Abhängigkeit
- von der Hardware wird der Rollback-Code des Programms aufgerufen, der die Berechung der 5. Epoche neu startet

Bewertung

- Funktionsweise ist sehr ähnlich zur Software-Umsetzung, nutzt aber bestehende, erprobte Strukturen

Bewertung

- Funktionsweise ist sehr ähnlich zur Software-Umsetzung, nutzt aber bestehende, erprobte Strukturen
- Hardware muss (geringfügig) erweitert werden, was bedeutet, dass diese Lösung nicht auf bestehenden Rechnern einsetzbar ist

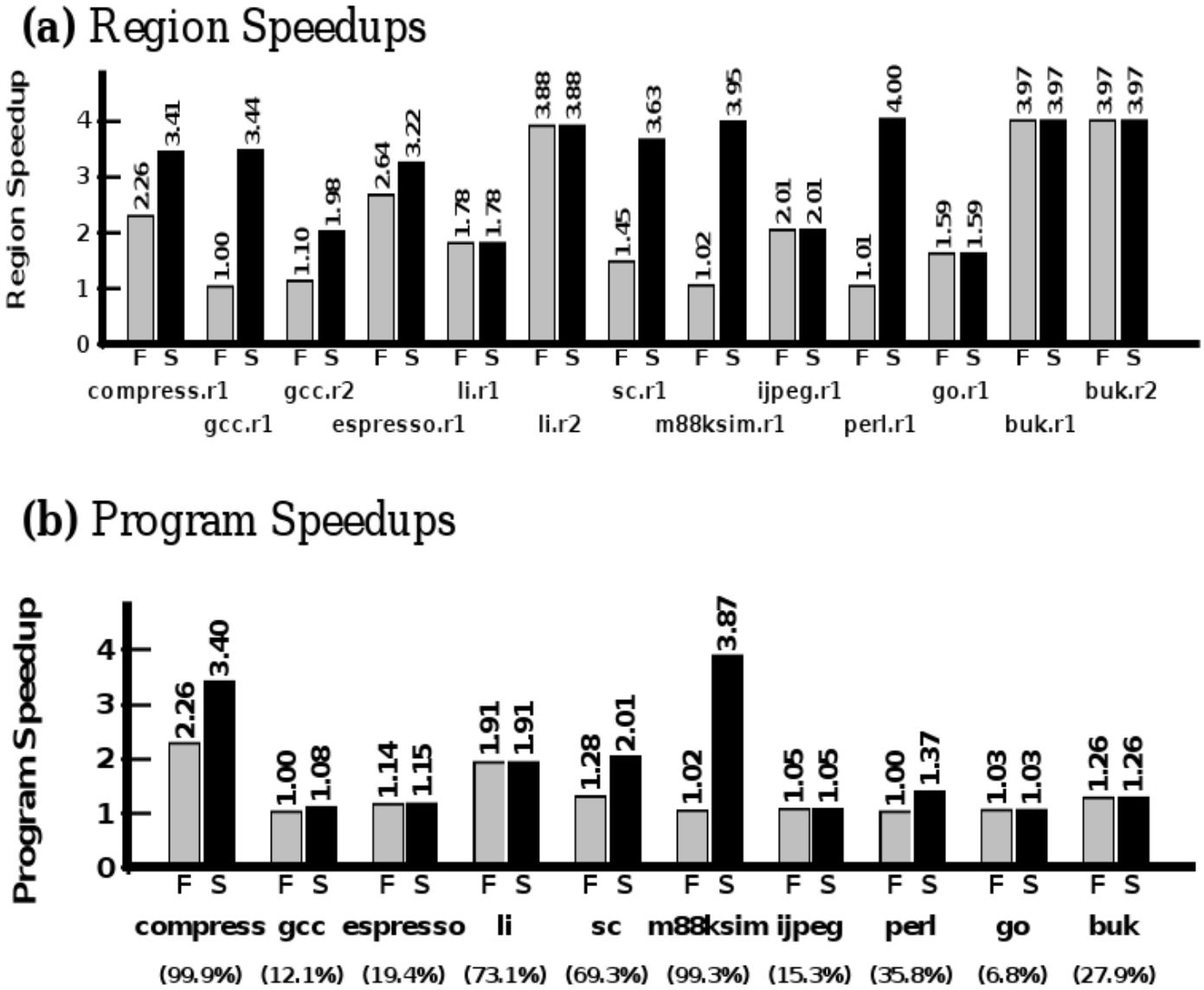
Bewertung

- Funktionsweise ist sehr ähnlich zur Software-Umsetzung, nutzt aber bestehende, erprobte Strukturen
- Hardware muss (geringfügig) erweitert werden, was bedeutet, dass diese Lösung nicht auf bestehenden Rechnern einsetzbar ist
- Software-Interface ist einfacher als bei reinen Software-Implementierungen

Bewertung

- Funktionsweise ist sehr ähnlich zur Software-Umsetzung, nutzt aber bestehende, erprobte Strukturen
- Hardware muss (geringfügig) erweitert werden, was bedeutet, dass diese Lösung nicht auf bestehenden Rechnern einsetzbar ist
- Software-Interface ist einfacher als bei reinen Software-Implementierungen
- ähnliche Erweiterungen wie bei Software-Umsetzung sich möglich, z. B. Weiterleiten von einzelnen Werten zwischen Epochen oder Aufweichen der seriellen Commit-Phase und Erlauben von parallelen Commits

Speedup – Hardware-Umsetzung



Worum ging es heute?

- automatische Parallelisierung von Programmen mit komplexem Kontrollfluss und nicht statisch zur Übersetzungszeit analysierbaren Datenabhängigkeiten

Worum ging es heute?

- automatische Parallelisierung von Programmen mit komplexem Kontrollfluss und nicht statisch zur Übersetzungszeit analysierbaren Datenabhängigkeiten
- software- und hardwarebasierte Umsetzungen vorgestellt mit ähnlicher grundlegender Funktionsweise

Worum ging es heute?

- automatische Parallelisierung von Programmen mit komplexem Kontrollfluss und nicht statisch zur Übersetzungszeit analysierbaren Datenabhängigkeiten
- software- und hardwarebasierte Umsetzungen vorgestellt mit ähnlicher grundlegender Funktionsweise
- bisher keine vollständig automatische Übersetzung eines sequentiellen in ein paralleles Programm möglich

Worum ging es heute?

- automatische Parallelisierung von Programmen mit komplexem Kontrollfluss und nicht statisch zur Übersetzungszeit analysierbaren Datenabhängigkeiten
- software- und hardwarebasierte Umsetzungen vorgestellt mit ähnlicher grundlegender Funktionsweise
- bisher keine vollständig automatische Übersetzung eines sequentiellen in ein paralleles Programm möglich
- langfristig scheint eine von der Hardware unterstützte Lösung realistischer

Literatur


P. Feautrier.
Dataflow analysis of array and scalar references.
International Journal of Parallel Programming, 20(1):23–53, 1991.


Martin Griebl.
The Mechanical Parallelization of Loop Nests Containing while *Loops*.
PhD thesis, University of Passau, 1996.
also available as technical report MIP-9701.


C. Lengauer.
Loop parallelization in the polytope model.
In E. Best, editor, *CONCUR'93*, Lecture Notes in Computer Science 715, pages 398–416. Springer-Verlag, 1993.