Ergebnisprüfung
Seminar Wintersemester 2003 - 2004

Designing checkers for programs that run in parallel

Seminararbeit im Studiengang Informatik von

René Nissing
Matr. Nr. 216 659

Lehr- und Forschungsgebiet Theoretische Informatik
Prof. Dr. P. Rossmanith
Rheinisch-Westfälische Technische Hochschule Aachen

Betreuer: Stefan Richter

Inhaltsverzeichnis

1 Einleitung

Um die Fehlerfreiheit eines Programms zu prüfen, gibt es grundsätzlich 2 Möglichkeiten.

Die erste Möglichkeit besteht darin, durch einen formalen Beweis für den implementierten Algorithmus die Korrektheit des zugrunde liegenden Programms zu garantieren. Hierbei wird anhand eines Regelwerks der mathematische Beweis für das Programm aufgestellt. Es ist jedoch in den meisten Fällen sehr schwierig und aufwendig, solche Beweise zu finden.

Die zweite Möglichkeit besteht darin, einen Ergebnisprüfer einzusetzen. Diese Alternative überprüft die Ausgabewerte für einen gegebenen Eingabewert auf Korrektheit. Ein Ergebnisprüfer verifiziert also nicht die Korrektheit des zugrunde liegenden Algorithmus, sondern die Korrektheit des Ergebnisses (bei zufälligen oder vorgegebenen Eingabewerten). Ein Ergebnisprüfer wird zur Laufzeit mit dem geprüften Programm ausgeführt und liefert eine realtime-Beurteilung des Ergebnisses. Die Überprüfung des Ergebnisses wird durch eine Aufblähung des Programmcodes und eine Verlängerung der Laufzeit des Programms erkauft.

Die Fehlerfreiheit eines parallelen Programms zu prüfen ist wesentlich problematischer als für serielle Algorithmen. Die Vorteile, die durch die Parallelverarbeitung gewonnen werden, werden durch eine komplexere Struktur des parallelen Programms erkauft. Beim Entwurf eines Ergebnisprüfers sollten - wie das betrachtete Programm - die parallelen Mittel ausgeschöpft werden. Es wäre also uneffektiv einen sequentiellen Ergebnisprüfer für einen schnellen parallelen Algorithmus zu benutzen.

Die Grundlagen der Parallelrechnerarchitektur geben zunächst einen Einblick in die interne Struktur eines Parallelrechners. Hierbei wurde das Hauptaugenmerk auf die Architektur gelegt, welche den Beispielen aus diesem Seminar als Grundlage diente. Sodann wird das parallele Ergebnisprüfer-Modell vorgestellt, welches für parallele Programme eine Ausgangsbasis bietet und Randbedingungen festlegt, Ergebnisprüfer zu entwerfen. Dann werden anhand von Problemstellungen Techniken vorgestellt, mit denen parallele Ergebnisprüfer modelliert werden können. Eine Technik besteht darin, Funktionen für einen bestimmten Eingabewert indirekt zu berechnen. Das wird realisiert durch das Auswerten der Funktionswerte von kompatiblen Eingabewerten zum Original. Eine weitere Technik beruht auf dem systematischen Zusammensetzen der Ausgabewerte von Teilmengen der Eingabewerte und der Überprüfung der Konsistenz der erhaltenen Ergebnisse.

2 Grundlagen

2.1 Zielsetzungen und Einsatzbereiche der Parallelverarbeitung

Durch den Einsatz modernster Parallelrechner und die Entwicklung paralleler Algorithmen ist es gelungen, Probleme, die früher Rechenzeiten von Wochen und Monaten erforderten, innerhalb von wenigen Stunden zu bearbeiten. Die Simulation von komplexen Phänomenen aus den Natur- und Ingenieurswissenschaften und zugehörige Parameterstudien sind dadurch erst möglich geworden. Parallelrechner sind daher inzwischen zu einem unentbehrlichen Hilfsmittel in vielen Bereichen wie Bildverarbeitung, Kristallzüchtung, Halbleiterentwicklung, Strömungssimulation und neuronale Netze geworden.

Das Hauptziel der Parallelverarbeitung ist die Verringerung der Laufzeit von Programmen durch eine höhere Zahl von Transaktionen pro Zeiteinheit. Weitere wesentliche Zielsetzungen sind es eine höhere Zuverlässigkeit zu gewährleisten, eine geringere Laufzeit langlaufender Prozesse zu erhalten und eine Möglichkeit zur Verarbeitung komplexer Probleme bereitzustellen. Ein Beispiel ist die Verarbeitung und Visualisierung von meteorologischen Daten (z.B. Wettervorhersage). Um die riesige Datenmenge zu verarbeiten sowie eine aktuelle und vorausschauende Visualisierung zu gewährleisten, werden typischerweise Parallelrechner eingesetzt.

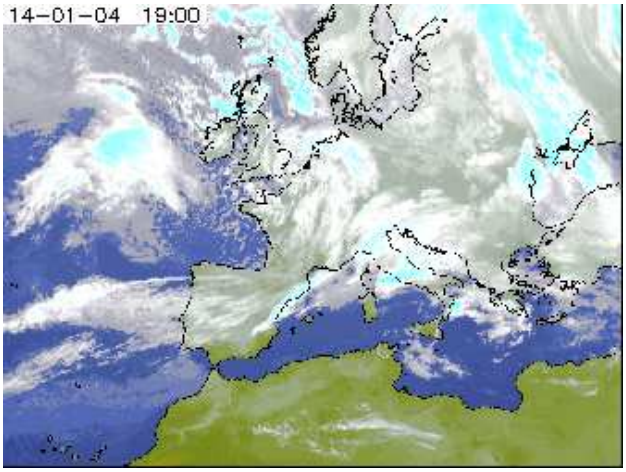

Abbildung 1: Visualisierung von meteorologischen Daten

Um eine vorausschauende Simulation der Wetterentwicklung, bei der Berücksichtigung von Temperatur, Luftdruck, Windgeschwindigkeit, ... ist

eine unüberschaubare Menge an Daten zu verarbeiten. Parallelrechner werden häufig eingesetzt um Wettervorhersagen zu prognostizieren.

2.2 Rechnermodelle

Die in den Beispielen behandelten parallelen Probleme wurden auf Parallelrechnern ausgeführt, die auf dem CRCW PRAM-Modell basieren. Um dieses Modell einzuführen, wird zunächst die RAM-Architektur einer Einprozessormaschine und anschließend die darauf aufbauende PRAM-Architektur einer Mehrprozessormaschine vorgestellt. Die Schreib- und Lesekonflikte, die bei einer PRAM-Architektur auftreten können, werden durch entsprechende PRAM-Varianten gelöst. Eine dieser Varianten ist das den Beispielen aus den Techniken zugrunde liegende CRCW PRAM-Modell.

2.2.1 RAM-Modell

Das Modell für Einprozessorrechner wurde durch Sheperdson und Sturgis 1963 entwickelt. Die Verallgemeinerte Registermaschine (Random-Access Machine RAM) besteht aus den Komponenten:

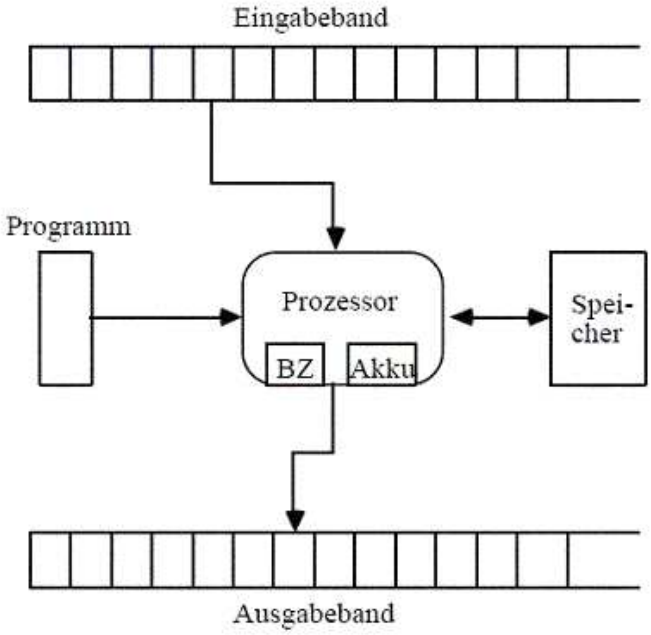

Abbildung 2: das RAM-Modell

- Programm:
  - numerierte, endliche Folge von Befehlen
- Speicher:
  - abzählbar viele Register
  - wahlfrei adressierbar
  - jedes Register kann eine beliebige ganze Zahl speichern
- Ein- / Ausgabebänder:
  - Einwegbänder
  - vom Eingabeband nur lesen
  - auf Ausgabeband nur schreiben
- Zentrale Recheneinheit:
  - Befehlszähler enthält Nummer des auszuführenden Befehls
  - Akkumulator: Zielregister von Berechnungen

2.2.2 PRAM

Das PRAM-Modell wurde durch Fortune und Wyllie 1978 entwickelt. Die parallele Registermaschine (Parallel Random-Access Machine PRAM) besteht aus einer Anzahl von (sequentiell arbeitenden) verallgemeinerten Registermaschinen (RAM´s). Das PRAM-Modell eines idealisierten, speichergekoppelten Parallelrechners ohne Beachtung von Synchronisations- und Speicherzugriffskosten besteht aus n identischen Prozessoren (sequentiellen Registermaschinen), die alle auf einen gemeinsamen Speicher (z.B. shared Memory) zugreifen können.

Alle Prozessoren werden von einem gemeinsamen Takt gesteuert und führen zu einem Zeitpunkt dieselben oder auch verschiedenartige Rechenoperationen aus.

Beim Zugriff auf den gemeinsamen Speicher können dadurch Konflikte auftreten, dass verschiedene Prozessoren die gleiche Speicherzelle lesen oder schreiben. Um diese Konflikte zu lösen, wird jeder Befehlszyklus in drei getrennte Schritte zerlegt:

- Speicher lesen,
- Operation ausführen und

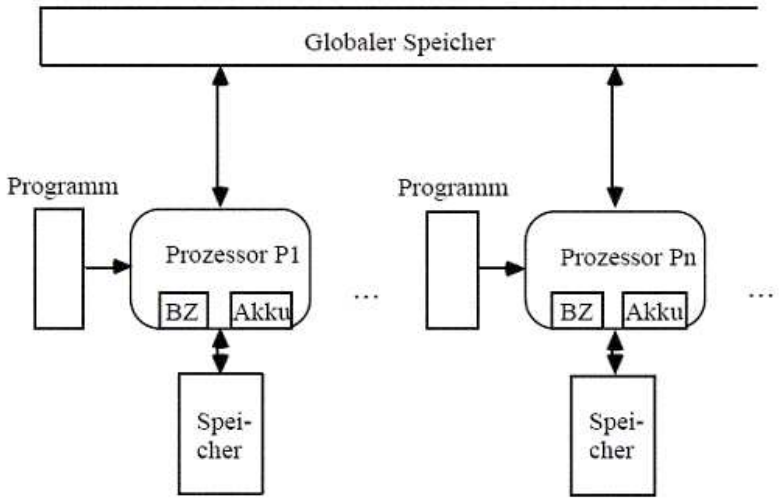

Abbildung 3: das PRAM-Modell

- Speicher schreiben.

Das Ausführen einer Operation gestaltet sich als unkritisch. Zu Betrachten sind die Konflikte, die beim gemeinsamen Lesen oder beim gemeinsamen Schreiben einer Zelle entstehen.

2.2.3 CRCW PRAM

Um die Zugriffskonflikte zu beheben, gibt es vier Optionen für PRAM´s.

Exclusive read (ER): Pro Zyklus kann höchstens ein Prozessor dieselbe Speicherzelle lesen.

Exclusive write (EW): Pro Zyklus kann höchstens ein Prozessor dieselbe Speicherzelle beschreiben.

Concurrent read (CR): Pro Zyklus können mehrere Prozessor dieselbe Speicherzelle lesen.

Concurrent write (CW): Pro Zyklus können mehrere Prozessoren dieselbe Speicherzelle beschreiben.

Diese verschiedenen Optionen können kombiniert werden und ergeben die vier PRAM-Varianten:

EREW PRAM: Gemeinsame Lese- oder Schreibzugriffe verboten.

CREW PRAM: Gemeinsame Lesezugriffe erlaubt, mehrere Schreibzugriffe verboten.

ERCW PRAM: Exklusiver Lesezugriff, gemeinsamer Schreibzugriff.

CRCW PRAM: Gemeinsame Lese- oder gemeinsame Schreibzugriffe erlaubt.

Bei Schreibkonflikten wird nach einer der folgenden vier Methoden gehandelt:

Common (C-CRCW): Gleichzeitige Schreibzugriffe auf dieselbe Zelle sind nur erlaubt, falls alle Schreibzugriffe dort denselben Wert ablegen wollen.

Arbitrary (A-CRCW): Einer der schreibenden Prozessoren gewinnt zufällig und belegt die Speicherzelle mit seinem Wert, die anderen werden ignoriert.

Minimum (M-CRCW): Von den am Schreibkonflikt beteiligten Prozessoren gewinnt der Prozessor mit dem kleinsten Index. Er beschreibt die Speicherzelle.

Priority (P-CRCW): Eine festgelegte Priorität zwischen den Prozessoren entscheidet, welcher Prozessor schreiben darf.

Alle im Folgenden behandelten Beispiele für Ergebnisprüfer basieren auf dem Arbitrary (A-) oder Priority (P-) CRCW PRAM Modell.

3 Modell

Bei den meisten Problemen, die mit parallelen Mitteln gelöst werden, zeigt sich, dass die Algorithmen erheblich komplexer sind als die des sequentiellen Gegenparts. Das parallele Ergebnisprüfer-Modell legt Randbedingungen fest und dient als Grundlage für das Entwickeln paralleler Ergebnisprüfer. Es werden Begriffe für sequentielle Ergebnisprüfer für die Parallelen erweitert.

Definition 1 (probabilistischer Ergebnisprüfer)
Voraussetzungen:
Sei P ein Programm (auch Orakel genannt), das die Funktion f berechnen soll.
Sei x der betrachtete Eingabewert.
Sei α der Konfidenzparameter, der bestimmt, mit welcher Wahrscheinlichkeit das Programm das richtige Ergebnis ausgibt.

Ein probabilistischer Ergebnisprüfer für f ist ein probabilistisches Orakel-Programm R_f^P mit Orakel P, das verifiziert ob P das richtige Ergebnis bei einem gegebenen Eingabewert x ausgibt, in der Weise wenn:

1. *$P(x) \neq f(x)$, dann gibt R_f^P „FAIL“ aus mit einer Wahrscheinlichkeit $\geq 1 - \alpha$*

2. *P korrekt ist für jeden Eingabewert, dann gibt R_f^P „PASS“ aus mit einer Wahrscheinlichkeit $\geq 1 - \alpha$*

Darüber hinaus darf der Ergebnisprüfer nur eine polynomielle Anzahl an Prozessoren benutzen.

Der parallele Ergebnisprüfer hat die Möglichkeit, das Programm P bei jedem parallelen Schritt so oft wie gewünscht auszuführen. Der Ergebnisprüfer darf P aber nur wie eine Blackbox nutzen. Das heißt von der inneren Struktur vom Programm P wird abstrahiert.

Wir werden im weiteren Verlauf folgende Notationen benutzen: $D(n)$ steht für die Gesamtlaufzeit des Ergebnisprüfers, der mit einem Eingabewert der Größe n gestartet wird. $N(n)$ steht für die Gesamtzahl an Prozessoren, die durch den Ergebnisprüfer genutzt werden bei einem Eingabewert der Größe n.

Die Mehrfachausführung des Programms auf dem gleichen Eingabewert darf nicht als Argument für die Korrektheit des Programms genommen werden. Deshalb muss der Ergebnisprüfer sich vom zu prüfenden Programm unterscheiden. Diese Voraussetzung stellt sicher, dass als Ergebnisprüfer nicht das Programm selber genommen wird.

Definition 2 (quantifiably different)
Annahme:
$d(n)$ ist die Laufzeit des parallelen Programms, das f berechnet bei einer Größe n des Eingabewerts.
$p(n)$ ist die Anzahl an genutzten Prozessoren bei einer Laufzeit $d(n)$.

Wir sagen, dass der Ergebnisprüfer R_f^P quantifiably different ist wenn:

1. *die Prüflaufzeit $o(d(n))$ ist oder*
2. *gleichzeitig die Prüflaufzeit $O(d(n))$ ist und die Anzahl der Prozessoren zum Überprüfen $o(p(n))$ ist.*

Alle parallelen Ergebnisprüfer aus den vorgestellten Techniken sind quantifiably different.

4 Beispiele

In den folgenden Beispielen werden verschiedene Techniken vorgestellt, wie Ergebnisprüfer für parallele Probleme entworfen werden können.

4.1 Techniken

4.1.1 Berechnung durch zufällige Eingabewerte

Wir betrachten ein Programm P, das den Ausgabewert einer Funktion f berechnet bei einem Eingabewert x. Der parallele Ergebnisprüfer ruft P auf zufällig gewählten Eingabewerten auf. Diese zufälligen Eingabewerte müssen in einem gewissen Zusammenhang mit dem betrachteten Eingabewert stehen. Diese Technik wird vorzugsweise für symmetrische Funktionen eingesetzt.

Definition 3 (symmetrische Funktionen)
Symmetrische Funktionen sind n-Bit Funktionen, deren Ausgabewert nur von der Anzahl 1*'en aus der Eingabe abhängt. Grundsätzlich wird eine symmetrische Funktion definiert durch eine Wertetabelle* $t_0, \ldots, t_n$ *, so dass* t_i *der Ausgabewert der symmetrischen Funktion ist, wenn genau* i *der Eingabebits* 1 *´en sind.*

Beispiel 1 (symmetrische Funktion)
Ein Beispiel für eine symmetrische Funktion ist die Majoritäts-Funktion. Sie gibt t aus wenn genau t 1'en im Eingabewert vorhanden sind. Für die Majoritäts-Funktion sei erwähnt, dass hierfür als Eingabewert eine Wertetabelle überflüssig ist, da der Ausgabewert in konstantem Zeitaufwand berechnet werden kann.

Entwurf des Ergebnisprüfers

Es wird die Eigenschaft genutzt, dass die symmetrische Funktion für einen gegebenen Eingabewert durch das Berechnen von zufälligen Permutationen des gegebenen Eingabewerts indirekt berechnet werden kann.

Beispiel: sei $\widehat{a} = 10000$ und π_i eine Permutation von $\widehat{a}$, so ist

$$f(\widehat{a}) = f(\pi_i(\widehat{a})) = f(01000) = f(00100) = f(00010) = f(00001).$$

Sei nun eine symmetrische Funktion f gegeben.

Eingabewerte: Eine Liste von n Bits $\widehat{a} = a_1, a_2, \ldots, a_n$ und eine Wertetabelle $t_0, \ldots, t_n$.

Ausgabewert: $b = t_l$ wobei $l = \sum_{1 \leq j \leq n} a_j$.

Sei P das Programm, das die symmetrische Funktion f berechnet. Die Eingabemenge wird in $n+1$ Äquivalenzklassen partitioniert. In jeder Äquivalenzklasse sind Elemente mit der gleichen Anzahl 1'en enthalten. Jeder n-Bit Eingabewert gehört zu der Äquivalenzklasse, die genau dieselbe Anzahl 1'en hat. In Phase 1 prüft der Ergebnisprüfer, ob das Ergebnis von P bei dem betrachteten Eingabewert mit mehr als die Hälfte der Elemente aus der zugehörigen Äquivalenzklasse konsistent ist. In Phase 2 prüft der Ergebnisprüfer, ob P bei mehr als die Hälfte der Elemente aus jeder Äquivalenzklasse das richtige Ergebnis liefert. Phase 2 gewährleistet, dass die Inkonsistenzen, die in Phase 1 nicht entdeckt werden konnten, gefunden werden.

Hier der zugehörige Ergebnisprüfer-Algorithmus:

$k = log(1/\alpha)$
$b = P(\widehat{a})$
Berechne parallel k zufällige Permutationen $\pi_1, \ldots, \pi_k$ von $\{1, \ldots, n\}$
Phase 1: (Konsistenz mit unserem Eingabewert)
Berechne parallel für $i = 1, \ldots, k$:
`Wenn` $P(\pi_i(\widehat{a})) \neq b$ `dann` gebe „FAIL“ aus und halte an
Phase 2: (Prüfe die Korrektheit der meisten Eingabewerte)
Berechne parallel für $j = 0, \ldots, n$:
Berechne parallel für $i = 1, \ldots, k$:
`Wenn` $P(\pi_i(1^j 0^{n-j})) \neq t_j$ `dann` gebe „FAIL“ aus und halte an
Gebe „PASS“ aus.

Die beiden Phasen werden hier weiter erläutert:

Phase 1: Ein Ausgabewert A bei einem Eingabewert E wird durch das Programm P berechnet. Der Ergebnisprüfer greift die Eingabe E ab und berechnet parallel k Permutationen. Die Permutationen werden auf den Eingabewert E angewandt und parallel auf k Prozessoren in das Programm P eingespeist. Die Ausgabewerte sind $A_1, \ldots, A_k$. Der Comparer vergleicht die Ausgabewerte $A_1, \ldots, A_k$ mit dem Ausgabewert A. Wenn das Programm P korrekt rechnet, so gibt es keine Inkonsistenzen und der Ergebnisprüfer gibt „PASS“ aus. Sobald sich aber eine Ausgabe von der originalen Ausgabe A unterscheidet so gibt der Ergebnisprüfer „FAIL“ aus.

Phase 2: Es werden die Permutationen, die bereits für Phase 1 berechnet worden waren, auf zufällige Eingabewerte mit unterschiedlicher Anzahl 1'en angewandt. Diese Eingabewerte werden folgendermaßen zusammengebaut:

$$\text{für } 1 \leq j \leq n: \ 1^j 0^{n-j}$$

Für jeden Eingabewert werden die k Permutationen auf den Eingabewert angewandt und der Ausgabewert wird auf den entsprechenden Eintrag in der Wertetabelle geprüft. Phase 2 entdeckt somit die Inkonsistenzen, die nicht in der ersten Phase des Ergebnisprüfers gefunden worden.

Beweis der Korrektheit des Ergebnisprüfers:

Wenn P auf allen Eingabewerten richtig rechnet, so gibt der Ergebnisprüfer korrekterweise „PASS“ aus. Es bleibt zu beweisen, dass, wenn P ein falsches Ergebnis ausgibt, der Ergebnisprüfer „FAIL“ ausgibt.

Annahme:
P liefert bei der Berechnung des Eingabewerts $\widehat{a}$ einen falschen Wert.

Zu zeigen:
Der Ergebnisprüfer gibt mit einer Wahrscheinlichkeit $\geq 1 - \alpha$ „FAIL“ aus.

Beweis:
Sei r die Anzahl der 1'en des Eingabewertes $\widehat{a}$.

1. Nehmen wir an, dass P bei mehr als die Hälfte der Eingabewerte mit r 1'en die richtige Antwort liefert (also mit der Antwort auf Eingabewert $\widehat{a}$ sich unterscheidet). Dann wird mit einer Wahrscheinlichkeit $\geq 1 - \alpha$ eine Inkonsistenz mit dem Eingabewert $\widehat{a}$ in Phase 1 des Algorithmus gefunden. Der Ergebnisprüfer würde somit „FAIL“ ausgeben.

2. Nehmen wir nun an, dass das Programm bei mehr als die Hälfte der Eingabewerte mit r 1'en auch die falsche Antwort ausgibt. Dann wird mit einer Wahrscheinlichkeit $\geq 1 - \alpha$ die $r - te$ Gruppe Prozessoren herausfinden, dass das Programm P fehlerhaft ist. Der Ergebnisprüfer würde hier ebenfalls „FAIL“ ausgeben.

Laufzeitanalyse:

Die Prüflaufzeit ist $O(log * n)$ und die Anzahl der Prozessoren für den Ergebnisprüfer ist $O(n^2)$. Die Gesamtlaufzeit ist $O(log * n + D(n))$ und die gesamte Anzahl der Prozessoren beläuft sich auf $O(nN(n))$.

4.1.2 Konsistenzbeweis

Es ist möglich, parallele Ergebnisprüfer zu entwickeln, die das Programm P nutzen um die Rechenschritte des sehr simplen sequentiellen Algorithmus zu rekonstruieren. Dann wird die Konsistenz zwischen den verschiedenen Rechenschritten geprüft.

Die in diesem Abschnitt vorgestellte Technik kann für jedes Problem, das sequentiell durch dynamische Programmierung gelöst wurde, genutzt werden. Dynamische Programmierung beschreibt einen Algorithmus, der durch das Aufteilen der Eingabemenge in kleinere Teilmengen, den Ausgabewert dynamisch zusammenbaut. Die berechneten Ausgabewerte für die Teilmengen werden in eine Tabelle geschrieben. Die Idee hinter dem Ergebnisprüfer ist es, das Programm mit den Teilmengen parallel aufzurufen und die Wertetabelle zu füllen. Dann wird überprüft, ob die gespeicherten Werte aus der Tabelle konsistent miteinander sind. Ein Beispiel für einen Ergebnisprüfer, der einen Algorithmus prüft, der durch dynamische Programmierung gelöst wurde, ist das Longest Common Subsequence-Problem.

Longest Common Subsequence Problem

Voraussetzung: Sei $lcs(l,k)$ die längste gemeinsame Zeichensequenz von $x_l x_{l+1} \dots x_n$ und $y_k y_{k+1} \dots y_n$.

Eingabewert: Zwei Zeichenfolgen $x = x_1 x_2 x_3 \dots x_n$ und $y = y_1 y_2 y_3 \dots y_n$.

Ausgabewert: Die Länge der längsten gemeinsamen Zeichensequenz.

Der sequentielle Algorithmus baut die Wertetabelle folgendermaßen auf. Wenn $x_l = y_k$, dann ist

$$lcs(l,k) = 1 + lcs(l+1,\ k+1),$$

ansonsten ist

$$lcs(l,k) = \max\{lcs(l,k+1),\ lcs(l+1,k)\}.$$

Der Ergebnisprüfer verifiziert ob diese Eigenschaften für die Werte aus der Tabelle erfüllt sind.

Entwurf des Ergebnisprüfers

Der Algorithmus für den Ergebnisprüfer sieht folgendermaßen aus:

`Für` $1 \leq l \leq n$
  `Für` $1 \leq l \leq n$
    $s_{lk} = P(x_l \dots x_n, y_k \dots y_n)$
    Überprüfe die Konsistenz:
      `Wenn` $x_l = y_k$ `dann` überprüfe, dass $s_{lk} = 1 + s_{l+1,k+1}$
      `sonst` überprüfe, dass $s_{kl} = \max\{s_{l,k+1}, s_{l+1,k}\}$

Wenn eine dieser Überprüfungen fehl schlägt so gebe „FAIL“ aus, ansonsten „PASS“.

Laufzeitanalyse

Die Prüflaufzeit ist $O(1)$ und die Anzahl der Prozessoren für den Ergebnisprüfer ist $O(n^3)$. Die Gesamtlaufzeit ist $O(1 + D(n))$ und die gesamte Anzahl Prozessoren beläuft sich auf $O(n^3 + n^2 x N(n))$.

5 Zusammenfassung

Die Vorteile, die durch parallele Programme erzielt werden, werden durch eine komplexere Struktur des Programms erkauft. Der Programmcode eines parallelen Algorithmus muss die verschiedenen Prozessoren auslasten. Der Ergebnisprüfer muss entsprechend die parallelen Mittel ausschöpfen. Die verschiedenen Parallerechnerarchitekturen legen eine Ausgangsbasis parallele Ergebnisprüfer zu programmieren. Das CRCW PRAM-Modell ist dasjenige Modell, das den vorgestellten Beispielen aus den Techniken zugrunde lag. Verschiedene Randbedingungen und Eigenschaften müssen beim Entwurf von parallelen Ergebnisprüfer berücksichtigt werden. Hierfür wurden grundlegende Begriffe für parallele Ergebnisprüfer erweitert. Die spezifischen Eigenschaften eines parallelen Ergebnisprüfer wurden anhand von Techniken aufgezeigt. Beide Techniken kennzeichneten beispielhaft, wie gewisse Eigenschaften von Problemstellungen genutzt werden können um einen effizienten Ergebnisprüfer zu entwerfen. In der 1. Technik wurde die Eigenschaft genutzt, dass ein Ausgabewert für einen gegebenen Eingabewert indirekt durch die Berechnung von zufälligen Permutationen berechnet werden konnte. In der 2. Technik wurde die Konsistenz von Werten aus einer Tabelle auf geltende Eigenschaften geprüft. Diese Techniken bieten Möglichkeiten an, parallele Ergebnisprüfer zu entwickeln.

Literatur

[1] Rubinfeld R., Designing Checkers for programs that run in parallel (1994), http://citeseer.nj.nec.com/rubinfeld94designing.html

[2] Hwang K., Advanced computer architecture: Parallelism, Scalability, Programmability (1993)

[3] Blum M. Kannan S., Designing programs that check their work (1989), http://citeseer.nj.nec.com/blum89designing.html

[4] Worringen J., Lehrstuhl für Betriebssysteme RWTH Aachen, Unterlagen Softwarepraktikum Parallelrechner (1999)

[5] Henk M., Computerorientierte Mathematik I mit Java (2002), http://www.math.tu-berlin.de/~henk/coma_berlin/coma9.pdf