FORTRAN auf Vektor-Rechnern

Carsten Pitz
(RWTH Aachen)

Die schnellsten käuflichen Rechner etablierter Hersteller (z.B. Cray, DEC, IBM, Fuitsu, Siemens-Nixdorf) sind Vektor-Rechner. Mittlerweile werden Vektorarchitekturen bis herunter in den Supermini-Bereich eingesetzt.

Vektor-Super-Computer sind extrem teuer, d.h. die Betreiber müssen auf eine gute Ausnutzung achten. Der Vortrag soll einen Einblick in die Arbeitsweise von Vektor-Rechnern geben, der einem Programmierer ermöglicht seine Programme für Vektor-Rechner zu optimieren. Ein "squeezing out last cycle" auf einer speziellen Zielmaschine ist nicht Ziel dieses Vortrages.

1 Wie arbeitet ein Vektor-Rechner ?

Operationen, die durch ein elementares FORTRAN-Statement beschrieben werden, sind auf der Hardware-Ebene eine Folge von Teiloperationen. Die Anweisung

```
C = A + B
```

z.B. enthält das Lesen der beiden Operanden A und B, die eigentliche Fließkomma-Addition und das speichern der Summe C. Die Fließkomma-Addition wiederum zerfällt in

- Exponentenvergleich
- Mantissenverschiebung
- Integer-Addition der Mantissen
- Normalisieren der Summe

Ein klassischer von Neumann Rechner führt alle Teiloperationen streng sequentiell aus.

Ein Beispiel:

Die Schleife

```
      DO 10 I=1,100
        C(I) = A(I) + B(I)
10    CONTINUE
```

kann in

```
C(  1) = A(  1) + B(  1)
C(  2) = A(  2) + B(  2)
...
C(100) = A(100) + B(100)
```

ausgerollt werden. (Im folgenden wird die ausgerollte Variante betrachtet.)

Ein von Neumann Rechner beginnt erst mit dem laden des ersten Operanden der zweiten Addition, nachdem er das Ergebnis der ersten Addition abgespeichert ist.

Im obigen Beispiel sind die Additionen voneinander unabhängig. Sie können beliebig in der Reihenfolge vertauscht werden, ohne das sich ein Ergebnis ändert. Dies hat auch zur Folge, daß während für die erste Fließkomma-Addition die Mantissen, nachdem sie verglichen worden sind, verschoben werden, für die zweite Fließkomma-Addition die Mantissen verglichen werden können. Im nächsten Takt kann die Mantissen-Addition der ersten, die Mantissenverschiebung der zweiten und der Mantissenvergleich der dritten Fließkomma-Addition gleichzeitig ausgeführt werden. Im folgenden Takt arbeiten dann alle vier Stufen des Rechenwerkes gleichzeitig.

Ein Rechenwerk, daß so arbeitet, wird Pipe (Röhre) genannt. Es kann durch eine mit Kugeln gefüllte Röhre veranschaulicht werden.

In einem modernen Vektor-Rechner sind zumeist Pipes für die arithmetischen Grundoperationen auf REAL und INTEGER Operanden als auch bitweise logische Operationen und Schiebeoperationen auf INTEGER Operanden vorhanden.

Die Pipe nutzt die Teilkomponenten des Rechenwerkes sehr viel besser aus als der von Neumann Rechner. Der daraus resultierende Gewinn an Rechenleistung wird aber durch einige, im folgenden beschriebene, Einschränkungen erkauft.

Die wichtigste und schwerwiegendste Einschränkung ist, daß nur über Schleifen vektorisiert werden kann, bei denen die Anzahl der Durchläufe im voraus bekannt ist. D.h. in FORTRAN können nur DO-Loops vektorisiert werden.

Zudem kann der, zur Versorgung der Pipes notwendige, hohe Datendurchsatz nur mit einfachen (Hardware adressierbaren) Datenstrukturen erreicht werden. Diese werden Vektoren genannt. Die heutigen Vektor-Rechner haben durchweg Vektorregister. D.h. die Vektoren werden in Vektorregister geladen und die Pipes benutzen nur die Vektorregister und nicht direkt den Arbeitsspeicher. Folgende Adressierungsarten werden von den meisten Vektor-Rechnern unterstützt.

- contiguous

```
A( I + offset )
```

- constant stride

```
A( stride * I + offset )
```

- indirect

```
A( index(I) )
```

- masked

```
IF ( mask(I) ) THEN
  A(I)
END IF
```

Die Adressierungsarten sind nach dem Durchsatz geordnet. Bei contiguous access wird der höchste Durchsatz erreicht, indirect access und masked access sind am langsamsten.

Ein weiteres Problem sind Rekursionen (hier: Benutzung von Ergebnissen aus vorangehenden Schleifendurchläufen).

Hierzu ein Beispiel (N positiv (hier N=1 angenommen)):

```
      DO 10 I=1,M
        A(I+N) = A(I) + B(I)
10 CONTINUE
```

Bei skalarer Ausführung werden zuerst A(1) und B(1) geladen und die Summe als A(2) zurueckgespeichert. Danach wird das neue A(2) und B(2) geladen, addiert und als A(3) gespeichert, usw. .

Bei vektorieller Abarbeitung würde A(1:M) und B(1:M) in Vektorregister geladen, und in ein weiteres Vektorregister A'(1:M) summiert. Danach würde dann A(1:M) durch A'(1:M) ersetzt. D.h. beim zweiten Schleifendurchlauf würde das falsche A(2) benutzt.

Rekursion ist auch bei indirekter Adressierung möglich.

```
      DO 10 I=1,M
        A(L(I)) = A(I) + B(I)
10 CONTINUE
```

In beiden Fällen wird der Compiler skalaren Code erzeugen und auf die mögliche Rekursion hinweisen.

2 Beispiele

In den beiden Beispielen geht es um bei Vektor-Rechnern allgemein anwendbare Optimierungen. Ein "squeezing out last cycle" auf einer speziellen Zielmaschine ist nicht Ziel dieses Vortrages.

2.1 Die Matrix-Matrix-Multiplikation

Eine Matrix ist die numerische Darstellung einer Abbildung bei gegebenen Basen (Koordinatensystemen) im Original- und Bildbereich. Die aus dieser geometrischen Betrachtungsweise abgeleitete Rechenvorschrift für die Matrix-Matrix-Multiplikation, führt diese auf Skalarprodukte (Projektionen) zurück.

$$c_{i,j} = \text{Summe } a_{i,k}\, b_{k,j} \text{ über } k=1,N$$

Klassisch wird diese Rechenvorschrift durch die Dreifachschleife

```
      DO 30 I=1,N
        DO 20 J=1,N
          DO 10 K=1,N
            C(I,J) = C(I,J) + A(I,K)*B(K,J)
10        CONTINUE
20      CONTINUE
30    CONTINUE
```

realisiert.

Die drei Schleifen sind beliebig vertauschbar, ohne die Rechenvorschrift zu verletzen. Dies ergibt 3!=6 Programmvarianten, aus denen nun die für einen Vektor-Rechner beste gesucht werden soll.

In FORTRAN werden 2-dimensionale Felder spaltenweise gespeichert. Weiterhin ist bekannt, daß der contiguous access den höchsten Durchsatz aller Adressierungsarten erreicht. Somit ist die Variante optimal, in der möglichst viel spaltenweiser Zugriff erfolgt.

Der Scheifenindex I tritt 3-mal als Spaltenindex auf, der Index K 1-mal und der Index J wird nicht als Spaltenindex verwendet. Hieraus folgt, daß die Schleifenreihenfolge J, K, I (von außen nach innen) optimal ist.

Diese Variante laßt sich in Pseudo-Code schreiben als:

```
      DO 20 J=1,N
        DO 10 K=1,N
          C(1:N,J) = C(1:N,J) + A(1:N,K) * B(K,J)
10      CONTINUE
20    CONTINUE
```

Der Pseudo-Code gibt sehr gut die Arbeitsweise eines Vektor-Rechners wieder, da die Schleife über die vektorisiert wird, nicht mehr explizit erscheint.

2.2 Red-Black-SOR und XSOR Verfahren

Gegeben sei die folgende elliptische Differentialgleichung (Poisson-Gleichung):

$$u_{xx}(x,y) + u_{yy}(x,y) = f(x,y)$$

Zur Diskretisierung einer solchen partiellen Differentialgleichung mit regelmäßigen quadratischen Netz der Maschenweite h werden im einfachsten Fall die auftretenen Ableitungen durch Differenzenquotienten approximiert. Für den inneren Punkt eines Lösungsgebietes (Randprobleme werden hier nicht berücksichtigt) erhält man damit für I, J = 1, 2, ..., N die Gleichungen

$$4*U(I,J)-U(I+1,J)-U(I-1,J)-U(I,J+1)-U(I,J-1) = -H^2*F(I,J)$$

die ein lineares diagonaldominantes Gleichungssystem ergeben. Differenzenverfahren werden oft durch einen Differenzenstern, hier den 5-Punkt-Stern, symbolisiert.

Dies bedeutet, daß zur Berechnung einer neuen Näherung U(I,J) jeweils nur die Werte der Näherungen an den 4 Nachbarknoten benötigt werden. Es gibt demnach 2 verschiedene Gruppen von Knoten, zur Berechnung der Näherungen in Knoten einer Gruppe werden nur Knoten der anderen benötigt. Dies legt eine Aufteilung der Knoten nach Art eines Schachbrettmusters nahe. Daher wollen wir von roten und schwarzen Knoten sprechen.

```
1,1 (red)      1,2 (black)    1,3 (red)      ...
2,1 (black)    2,2 (red)      2,3 (black)    ...
3,1 (red)      3,2 (black)    3,3 (red)      ...
...            ...            ...            ...
```

Ein verbreitetes Verfahren zur Lösung solcher Gleichungssysteme sind iterative Methoden die sich aus dem Gauß-Seidel-Verfahren (Einzelschrittverfahren) ableiten. Wendet man auf das obige Gleichungssysstem das Verfahren der sukzessiven Überrelaxation (SOR) an, so erhält man für die sukzessiven Näherungen $U^{(n)}$:

$$U^{(n+1)}(I,J) = (1.0-\text{omega})\ U^{(n)}(I,J) + 0.25\ \text{omega}\ [\ U^{(n+1)}(I-1,J) + U^{(n+1)}(1,J-1) + U^{(\ n\ )}(I+1,J) + U^{(\ n\ )}(1,J+1) - H^2\ F(I,J)\ ]$$

Die rekursive Struktur des Differenzenverfahren bleibt hierbei erhalten, in dieser Form ist das Verfahren nicht vektorisierbar. Die Anordnung der roten und schwarzen Knoten legt aber eine neue Methode nahe. Das sogenannte Red-Black-SOR berechnet nun zuerst alle Näherungen für die roten Knoten aus den alten Näherungen der schwarzen Knoten. Danach werden die Näherungen der schwarzen Knoten aus den Näherungen der roten Knoten berechnet. Dies läßt sich sehr einfach wie folgt realisieren.

```
      DO 11 J=2,N-1,2
        DO 10 I=2,N-1,2
          U(I,J) = ...
10      CONTINUE
11    CONTINUE
```

```
      DO 21 J=3,N-2,2
        DO 20 I=3,N-2,2
          U(I,J) = ...
20      CONTINUE
21    CONTINUE

      DO 31 J=2,N-1,2
        DO 30 I=3,N-2,2
          U(I,J) = ...
30      CONTINUE
31    CONTINUE

      DO 41 J=3,N-2,2
        DO 40 I=2,N-1,2
          U(I,J) = ...
40      CONTINUE
41    CONTINUE
```

Alle Schleifen sind jetzt vektorisierbar. Dieses Verfahren ist aber nicht numerisch äquivalent zum SOR-Verfahren, da die Näherungen in einer anderen Reihenfolge berechnet werden. Obwohl dieses Verfahren schwächer konvergiert als das SOR-Verfahren, ist es aufgrund der Vektorisierbarkeit auf Vektor-Rechnern i.A.. schneller. Allerdings ist es sehr nachteilig, daß alle Schleifen einen Stride 2 aufweisen. Ein Stride 1 (contiguous access) ermöglicht einen höheren Datendurchsatz. Dieser wird erreicht wenn für die roten und die schwarzen Knoten getrennte Felder benutzt werden.

Möglich ist aber auch die Verwendung eines modifizierten Diskretisierungsverfahren, das ebenfalls ein 5-Punkt-Differenzenverfahren darstellt.

```
4*U(I,J)
-U(I+1,J+1)-U(I-1,J+1)-U(I+1,J-1)-U(I-1,J-1)
= -H**2*F(I,J)
```

Wegen der Form des Differenzenstern

heißt dieses Verfahren XSOR-Methode. Auch dieses Verfahren ist nicht numerisch äquivalent zu SOR oder Red-Black-SOR. Der Vorteil ist, daß für ein festes J die U(I,J) nicht mehr abhängig sind vom vorher berechneten U(I-1,J). Daher wird jetzt eine streifenförmige Aufteilung der Knoten in eine Art Zebramuster vorgenommen und die Knoten mit geradem Index J rot, mit ungeradem schwarz genannt. Der Kern des resultierenden Zebra-XSOR-Algorithmus hat folgendes Aussehen.

```
      DO 11 J=2,N-1,2
        DO 10 I=2,N-1
          U(I,J) = ...
10      CONTINUE
11    CONTINUE

      DO 21 J=3,N-2,2
        DO 20 I=2,N-1
          U(I,J) = ...
20      CONTINUE
21    CONTINUE
```

Dieses Verfahren greift spaltenweise mit contiguous access zu. Zudem ist die effektive Vektorlänge doppelt so groß wie beim Red-Black-SOR mit getrennten Feldern.

Dieses Verfahren vektorisiert optimal und die Konvergenz ist besser als beim ursprünglichen SOR Verfahren. D.h. dieses Verfahren ist selbst auf Skalarrechnern schneller als das SOR Verfahren.

3 Ausblick

Dieser Vortrag kann nur einen kleinen Ausschnitt aus dem breiten Spektrum der Probleme beim Programmieren von Vektor-Rechnern aufzeigen. Wichtige Punkte wie die Vermeidung von Speicherzugriffskonflikten oder das I/O-Tuning sind hier nicht thematisiert worden. Desweiteren habe ich auf die sonst so beliebten Benchmarkwerte verzichtet.

Vielen Problemen beim effektiven Programmieren von Vektor-Rechnern, kann man durch die Benutzung von Bibliotheken, aus dem Weg gehen. Der Programmierer kann i.A.. davon ausgehen, daß numerische Pakete wie LINPACK, EISPACK, LAPACK, NAG und IMSL in sehr gut optimierter Form auf Vektor-Rechnern verfügbar sind. Die Bibliotheken LINPACK, EISPACK und LAPACK sind zudem noch Public Domain. Also warum das Rad ständig neuerfinden.

Literaturnachweis

Grundlage meines Vortrags sind Schriften von Dr. rer. nat. Reinald Ehrig (Rechenzentrum der RWTH Aachen) und Dipl. math. Dieter an Mey (Rechenzentrum der RWTH Aachen). Beide arbeiten in der Arbeitsgruppe von Dr. rer. nat. Wilfried Juling, der ich auch zugeordnet bin. Diese Arbeitsgruppe betreut einen SNI 600/20 Vektor-Rechner, eine 6 Prozessor IBM 3090 mit Vektoreinheiten und einen Parsytec Super-Cluster mit 256 Prozessoren.

Weiterhin stütze ich mich auf Vorlesungen und Skripte von Prof. Friedel Hoßfeld (Lehrstuhl für Technische Informatik und Computerwissenschaften an der RWTH Aachen und Leiter des Zentralinstituts für angewandte Mathemathik (Rechenzemtrum) der KFA Jülich) und Prof. Dieter Haupt (Lehrstuhl für Betriebssysteme an der RWTH Aachen und Leiter des Rechenzentrums der RWTH Aachen).