VDPACK – Ein benutzerorientiertes Unterprogrammpaket zur Realisierung einer dynamischen Speicherverwaltung in FORTRAN

J. Demel, S. Selberherr*

Stichworte: *Virtueller Speicher, Speicherverwaltung, FORTRAN, mehrdimensionale Simulation.*

Zusammenfassung: *Es wird ein Programmpaket zur Realisierung einer dynamischen Speicherverwaltung in FORTRAN 77 vorgestellt. Das Laufzeitverhalten wird anhand einer typischen Anwendung demonstriert.*

VDPACK – A Useroriented Subroutinepackage to Emulate Dynamic Memory Management with FORTRAN

Key-words: *Virtual Memory, Memory Management, FORTRAN, Multidimensional Simulation.*

Abstract: *An implementation of dynamic memory management with FORTRAN 77 is presented. The behavior is demonstrated with a typical application.*

1 Einleitung

Bei der Entwicklung von technischen Applikationsprogrammen, wie z.B. Programmen zur Simulation von Schaltungen des VLSI-Bereichs [1] oder der mehrdimensionalen Simulation von Halbleiterbauelementen [2], tritt häufig das Problem der Realisierung von sehr großen Vektoren oder Feldern auf. Die Größe des benötigten Speicherplatzes ist meistens nicht a priori bekannt, sondern wird erst im Zuge der Berechnung ermittelt. Um eine rationelle Verwendung des Hauptspeichers zu garantieren, ist daher eine dynamische Speicherverwaltung notwendig.

Derartige Applikationsprogramme werden, um die Übertragbarkeit auf andere Rechenanlagen zu ermöglichen, in der Regel in FORTRAN implementiert. Die mit den meisten FORTRAN 66 Implementierungen in diesen Fällen üblicherweise verwendete „Blank Common Extension" [3] ist bei einem Teil der FORTRAN 77 Implementierungen nicht mehr möglich. Die Tendenzen bei der Entwicklung des Standards Fortran 8x gehen derzeit in die Richtung, das Konzept der statischen Speicherplatzvergabe aufzugeben, wodurch die „Blank Common Extension" überhaupt nicht mehr möglich ist [4].

Wenn die Felder sehr groß werden und ein ausreichender realer oder ein virtueller Speicher nicht zur Verfügung steht, womit bei einer rechnerunabhängigen Implementierung gerechnet werden muß, wird es notwendig, Teile des Hauptspeichers auf Platte auszulagern.

In der Folge werden das Konzept und die Implementierung des Unterprogrammpakets VDPACK zur Lösung der oben angeführten Problematik dargestellt und einige Zeitvergleiche angestellt.

2 Konzept

Aus der Sicht des Benutzers werden zumindestens folgende Funktionen von einem dynamischen Speicherverwaltungssystem gefordert:

- Dimensionierung eines unabhängigen Speicherbereichs (eines Vektors mit eigenem Adreßraum),
- Lesen einer Speichereinheit,
- Setzen einer Speichereinheit,
- Freigeben eines Speicherbereichs (Vektors).

Sehr oft wird ein nur einige 100 Speichereinheiten großer Block in einem Programmblock behandelt, man denke nur an die Gauß-Elimination bei der Lösung eines Gleichungssystems. Um die Plattenzugriffe und den Overhead durch die Speicherverwaltung zu reduzieren, ist es effizienter, einen derartigen Block mit einem einzigen Unterprogrammaufruf in ein Hilfsfeld zu lesen bzw. vom Hilfsfeld auf Platte zu schreiben. Es sind daher auch folgende Funktionen wünschenswert:

- Lesen eines Speicherblocks,
- Schreiben eines Speicherblocks.

* Dipl.-Ing. Johannes Demel und Dipl.-Ing. Dr. Siegfried Selberherr, Abteilung Physikalische Elektronik, Institut für Allgemeine Elektronik und Elektronik, TU Wien, Gußhausstraße 27, A-1040 Wien, AUSTRIA

0013-5704/84/6 0244-04 $ 03.00/0

Bei Applikationen mit großem Rechenzeitbedarf – häufig werden iterative Verfahren verwendet – möchte der Benutzer die Berechnung unterbrechen, um die Zwischenergebnisse zu kontrollieren, und dann erst weiterrechnen. Dies erfordert die Funktionen:

- Sichern des momentanen Status der Speicherverwaltung auf Platte,
- Wiederherstellen des alten Status der Speicherverwaltung.

Neben den bisher beschriebenen Funktionen werden noch folgende benötigt:

- Konfiguration der Speicherverwaltung,
- Statistiken.

Die oben angeführten Funktionen sind alle in VDPACK implementiert. Die Namen der Unterprogramme und deren Parameter sind dem Anhang zu entnehmen, im folgenden sollen nur einige wichtige Details der Aufrufe dargestellt werden.

Adreßraum

Es wird zwischen zwei Arten von virtuellen Vektoren unterschieden – denen mit fixer Größe und Vektoren, deren Größe nicht bekannt ist und die daher so implementiert sind, daß sie automatisch „beliebig" nach oben erweitert werden können. Unter „automatisch" wird in diesem Zusammenhang verstanden, daß VDPACK programmintern dafür sorge trägt.

Der Adreßraum eines Vektors beginnt grundsätzlich mit Adresse 0. Falls eine niedrigere untere Vektorgrenze benötigt wird, ist dies durch eine entsprechende Verschiebung des Adreßraums im Applikationsprogramm zu bewerkstelligen. Wird eine höhere untere Vektorgrenze – z.B. der übliche Werte 1 – benötigt, ist zu empfehlen, den vollen Adreßraum nicht auszunutzen oder den Adreßraum im Applikationsprogramm entsprechend zu verschieben.

Als Einheit im Adreßraum wird die „numerische Speichereinheit" nach FORTRAN 77 [5] verwendet. DOUBLE PRECISION und COMPLEX deklarierte Variablen, die zwei numerische Speichereinheiten belegen, werden als Block aus zwei Elementen betrachtet.

Variablen vom Typ CHARACTER können mit der hier beschriebenen Version von VDPACK nicht behandelt werden, da sie nicht mit den numerischen Speichereinheiten kompatibel sind [5]. Falls virtuelle CHARACTER Vektoren doch benötigt werden, kann ein analoger Satz von Unterprogrammen zur Verwaltung von CHARACTER Vektoren zur Verfügung gestellt werden.

In VDPACK sind nur Vektoren direkt implementiert. Werden zwei- oder mehrdimensionale Felder benötigt, so ist die Umrechnung der Indizes auf einen eindimensionalen Adreßraum im Applikationsprogramm nach folgender Formel durchzuführen:

$$a = c \sum_{i=1}^{n} (s_i - k_i) \prod_{j=1}^{i-1} (u_j - k_j + 1)$$

mit

- c Anzahl der belegten Speichereinheiten pro Element (1 oder 2)
- n Anzahl der Indizes
- s_i i-ter Index
- k_i i-ter unterer Index
- u_i i-ter oberer Index
- a die neue Adresse

Unterscheidungen zwischen verschiedenen virtuellen Vektoren

Da in einem Applikationsprogramm oft mehrere große Vektoren vorkommen, muß allen Unterprogrammaufrufen, die sich auf einen bestimmten Vektor beziehen, ein Deskriptor-Vektor übergeben werden, der aus drei numerischen Speichereinheiten besteht. In diesem speichert VDPACK interne Informationen zur eindeutigen Identifizierung des Vektors, wie die Basisadresse des Vektors auf Platte, ab. Dieser Deskriptor kann zwischen den einzelnen Teilen des Applikationsprogramms mit den in FORTRAN 77 vorgesehenen Mechanismen, wie COMMON-Blöcken und Unterprogrammparametern, übergeben werden. Bei segmentierten oder in Overlays eingeteilten Programmen ist natürlich entsprechende Vorsicht geboten.

3 Implementierung

Um die Übertragbarkeit von VDPACK auf andere Rechner zu gewährleisten, ist VDPACK in FORTRAN 77 realisiert. Auf Platte werden die „virtuellen Vektoren" als direkte Dateien im Sinn von FORTRAN 77 angelegt. Alle Vektoren mit bekannter Größe werden in einer einzigen Datei zusammengefaßt. Für Vektoren, deren obere Grenze des Adreßraums a priori nicht bekannt ist, muß jeweils eine eigene Datei angelegt werden – es ist daher eine Beschränkung der Anzahl derartiger Vektoren notwendig.

Um die Anzahl der Plattenzugriffe und damit die Gesamtlaufzeit eines Programms zu reduzieren, wird nicht jeder Zugriff auf ein Element eines Vektors in einen Plattenzugriff umgelegt, sondern es wird ein in mehrere gleich große Segmente (typisch einige 10 bis einige 100 Speichereinheiten) geteilter Zwischenpuffer verwendet. Ein Plattenzugriff erfolgt nur, wenn die gewünschte Adresse nicht im Puffer ist. Wenn ein Segment in den Puffer geholt werden muß, es jedoch kein freies Segment mehr gibt, wird der Platz des Segments mit der niedrigsten Priorität verwendet. Falls dieses Segment seit dem letzten Lesen von Platte verändert wurde, muß das Segment zuerst auf Platte zurückgeschrieben werden. Die Priorität eines Segments wird nach dem „Last-Recently-Used" (LRU) Algorithmus [8] bestimmt, d.h. der Platz des am längsten nicht mehr verwendeten Segments wird verwendet. Als „Uhr" wird die Gesamtzahl der Segmentzugriffe genommen. In der Implementierungsphase wurden auch andere in der Regel kompliziertere Algorithmen getestet, es ergab sich jedoch kein applikationsunabhängiger Vorteil dieser Algorithmen.

Die günstigste Größe und Anzahl der Segmente im Puffer wird aus rechnerabhängigen Daten wie verfügbarer Hauptspeicher, kleinster physisch ansprechbarer Block auf der Platte und der gewünschten Effizienz des Programms bestimmt. Auch die Lokalität des Zugriffs des Applikationsprogramms ist von Bedeutung. Bei der Wahl der Pufferparameter ist man auf Testläufe angewiesen. Während die Gesamtgröße des Puffers zur Übersetzungszeit von VDPACK festgelegt werden muß, kann die Aufteilung des Puffers noch zur Laufzeit vor der ersten Dimensionierung eines Vektors geändert werden.

4 Anwendungsbeispiel

Um das Zeitverhalten von VDPACK zu ermitteln, wurde die Lösung linearer Gleichungssysteme unterschiedlicher Größe verwendet. Das Gleichungssystem wurde mit den Routinen SGEFA und SGESL von LINPACK [7] und deren auf VDPACK adaptierte Version – der Adaptierungsaufwand betrug etwa eine Mannstunde – gelöst. Die Rechenzeit, bezogen auf den Wert des Laufs ohne VDPACK mit Rang 50, und der Speicherbedarf für unterschiedlichen Rang N des Gleichungssystems ist nachfolgender Tabelle zu entnehmen. Als Test-Koeffizientenmatrix wurde eine Hilbertmatrix verwendet [7].

Für die Original-Routinen ergeben sich Rechenzeiten von $O(N^2)$. Infolge der Hauptspeicherbeschränkung des verwendeten Rechners CDC CYBER 170/720 war der maximale gerechnete Rang 250. Die Version mit VDPACK hat eine Rechenzeit von $O(N^{3.2})$.

Mit einer Implementierung von VDPACK in COMPASS, der Assemblersprache der CDC CYBER 170, ergibt sich eine Rechenzeit von $O(N^{2.8})$.

Der Hauptspeicherbedarf der Version ohne VDPACK ist N*(N+2), der mit VDPACK ist 4N+P, wobei P die Größe des internen Puffers von VDPACK (P>>4N) ist. Der Wert 4N gegenüber 2N ohne VDPACK wird durch die Verwendung von zwei Hilfsvektoren zur Speicherung von je einer Zeile der Matrix verursacht. Durch den praktisch konstanten Speicherbedarf wird für genügend hohen Rang die Version mit VDPACK sogar billiger als die Originalversion. Der Schnittpunkt der Kostenfunktion, dem Produkt aus Rechenzeit und Speicherbedarf, mit und ohne VDPACK liegt bei einem Rang von etwa 500. Das schlechtere Laufzeitverhalten der FORTRAN 77 Version ist auf den relativ großen Aufwand im FORTRAN Laufzeitsystem zurückzuführen.

Rang N	ohne VDPACK		FORTRAN VDPACK		COMPASS VDPACK	
	CPU	Speicher	CPU	Speicher	CPU	Speicher
50	1	2600	6.2	8392	3.2	8392
100	3.1	10200	412.2	8592	19.6	8592
150	6.8	22800	2871.9	8792	68.0	8792
200	12.0	40400	7343.0	8992	147.1	8992
250	–	63000	14519.9	9192	278.7	9192
300	–	90600	26364.4	9392	450.9	9392

5 Zusammenfassung

Die Implementierung eines Programmpaketes zur Realisierung von virtuellen Vektoren wurde vorgestellt. VDPACK wurde bereits in vielen Applikationsprogrammen zur Lösung technischer Probleme erfolgreich eingesetzt [2], [6]. Viele Probleme hätten ohne VDPACK auf Rechnern des Typs CDC CYBER 170 nicht gelöst werden können.

Anhang – Beschreibung der Unterprogramme

Das Unterprogrammpaket VDPACK besteht aus folgenden Unterprogrammen mit den angegebenen Parametern:

- CALL VDPACK (nnseg, nlseg, nprest)
  Das Unterprogramm VDPACK setzt die Konfigurationsparameter des Programmpakets VDPACK neu. Es werden die Parameter Größe eines Segments, Anzahl der Segmente und Vorbelegung von Speicherzellen gesetzt. Dieses Unterprogramm darf nur vor dem Aufruf eines der anderen Unterprogramme aufgerufen werden.
- CALL VDSTAT (ivdst)
  Im Parameter *ivdst* wird eine Statistik über die Verwendung von VDPACK geliefert.
- CALL VDDIM (descr,mdim)
  Es wird die Größe eines virtuellen Vektors festgelegt. VDDIM muß vor der ersten Verwendung eines virtuellen Vektors aufgerufen werden.
- CALL VDREL (descr)
  Der Speicherplatz des angegebenen virtuellen Vektors wird freigegeben.
- CALL VDGET (descr, adr, val)
  Es wird eine Speichereinheit des virtuellen Vektors gelesen.
- CALL VDBGET (descr, adr, bval, blen)
  Es wird ein Block von Speichereinheiten des virtuellen Vektors gelesen.
- CALL VDSET (descr, adr, val)
  Es wird eine Speichereinheit des virtuellen Vektors gesetzt.
- CALL VDBSET (descr, adr, bval, blen)
  Es wird ein Block von Speichereinheiten des virtuellen Vektors gesetzt.
- CALL VDSAVE (ic)
  Der momentane Status von VDPACK (alle zur Zeit aktiven virtuellen Vektoren mit deren Inhalt, Statistikwerte etc.) wird auf ein File gesichert, um später mit VDREST wieder geladen zu werden. Wird der Parameter *ic* auf einen Wert ungleich 0 gesetzt, so kann mit den derzeit aktiven virtuellen Vektoren weitergearbeitet werden, anderenfalls werden sie freigegeben.
- CALL VDREST (nnseg, nprest)
  VDPACK wird in den letzten mit VDSAVE gespeicherten Status gebracht. Die Konfigurationsparameter Anzahl der Segmente und Vorbelegung von Speicherelementen können neu gesetzt werden.

Bedeutung der Parameter

adr virtuelle Adresse einer Speicherzelle bzw. Beginnadresse eines Speicherblocks (INTEGER).

blen Länge eines Blocks (INTEGER).

bval lokaler Block im rufenden Programm, der gesetzt oder gelesen werden soll.

descr Deskriptor eines virtuellen Vektors, bestehend aus drei Speichereinheiten. Der Inhalt wird von den Routinen von VDPACK verändert.

ic Indikator für Routine VDSAVE (INTEGER). 0 bedeutet, daß die internen Puffer von VDPACK freigegeben werden.

ivdst Ein INTEGER Feld aus 8 Speichereinheiten, in das Statistikwerte geschrieben werden. In die einzelnen Speicherzellen werden folgende Werte geschrieben:

1 Status von VDPACK
  0 VDPACK nicht aktiv
  −1 VDPACK aktiv
2 Anzahl der Segmente (Parameter *nnseg*)
3 Größe eines Segments (Parameter *lseg*)
4 momentaner *nprest*-Wert
5 Anzahl der physischen Schreibbefehle
6 Anzahl der physischen Lesebefehle
7 Anzahl der Segment-Zugriffe mit VDSET, VDBSET
8 Anzahl der Segment-Zugriffe mit VDGET, VDBGET

mdim Maximale Größe eines virtuellen Vektors (INTEGER). Der Wert 0 bedeutet, daß die Größe nicht bekannt ist und der Vektor daher so angelegt werden muß, daß er beliebig erweitert werden kann.

nlseg Länge eines Segments (INTEGER). 0 bedeutet keine Änderung.

nnseg Anzahl der Segmente (INTEGER). 0 bedeutet keine Änderung.

nprest Wert zur Vorbelegung noch nicht gesetzter Speicherzellen. Der INTEGER-Wert 123321 bedeutet, daß der momentane Wert nicht geändert werden soll. Der Standart-Wert ist 0.

val Lokale Speicherzelle, die gesetzt oder gelesen werden soll.

Literatur

[1] *Demel, J.: JANAP – Spezifikation.* Techn. Univ. Wien, Juli 1981.

[2] *Franz, A., Franz, G., Selberherr, S., Ringhofer, Ch., Markowich, P.: Finite Boxes – A Generalization of the Finite-Difference Method Suitable for Semiconductor Device Simulation.* In: IEEE Transaction on Electron Devices ED-30, Seite 1070 ff., September 1983.

[3] *Green, D.: Dynamic Array Structures in FORTRAN.* In: Proceedings VIM-30, 1979.

[4] ANSI X3J3: *Fortran 8x Document S7 Version 89.* März 1984.

[5] ISO: *Programming Languages – FORTRAN, ISO 1539–1980 (E).* Juli 1981.

[6] *Langer, E., Selberherr, S., Markowich, P., Ringhofer, Ch.: Numerical Analysis of Acoustic Wave Generation in Anisotropic Piezoelectric Materials.* In: Sensors and Actuators 4, Seite 71 ff., 1983.

[7] *Dongarra, J., Moler, C., Bunch, J., Stewart, G.: LINPACK User's Guide.* 1979.

[8] *Schiemangk, H.: Virtuelle Speicher digitaler Rechenanlagen.* Oldenbourg, München, 1979.