解　説

SSOPTRAN プリコンパイラ†

—記憶階層を考慮したFORTRANソースプログラムの最適化—

三橋　鎭雄††　庄野　英二††

1. まえがき

汎用計算機の記憶階層としてはバッファ記憶，主記憶，外部記憶などがある．

データがバッファ記憶上にない確率（NFP：Not Found Probability），またはページフォルトの発生回数により，各階層間のデータ転送回数に差が生じる．バッファ記憶制御（sector 方式や set associative 方式など）や，オペレーティング・システムのページリプレースメント・アルゴリズム（FIFO 方式，LRU 方式など）は，多くの場合に転送回数を低く抑えるように設計されており，利用者は転送回数を意識せずに高速の記憶空間，または大容量の記憶空間を使用できる．しかし反面，これらの方式とうまく適合しない実行形態を持つジョブに対しては，利用者が想像する以上に効率の低下を招く場合がある．最近の超高性能計算機では，処理速度が高速なため特にこういった傾向が強く現われる．

一方，多くのFORTRANコンパイラが採用している最適化技法の典型は，プログラムの構造を解析し，冗長な命令を除き，適用計算機に合った機械命令を生成することである．しかし，前述のように各階層間のデータ転送回数を少なくすることを意図した場合には，ソースプログラムの修正が必要であり，従来の技法では限界がある．

本稿では，各階層間のデータ転送回数削減のためのソースプログラムの自動修正を行う最適化手法を示す．また，汎用計算機上のプリコンパイラとして，その最適化を実施した SSOPTRAN の概要とその効果について述べる．

† SSOPTRAN PRECOMPILER—An optimization technique considering memory hierarchy in FORTRAN source program—by Shizuo MITSUHASHI and Eizi SHONO (F/A Section, Language Processor Department, Fujitsu Ltd.).

†† 富士通(株) LP 部 F/A 課

2. 記憶階層構造を考慮した最適化

2.1 最適化の導入

FORTRAN プログラムにおいては，各種の技術計算を行うにあたって，配列を用いて問題解決を計る場合が非常に多い．さらに，処理する問題によっては巨大な配列を頻繁に参照し，プログラム走行時間の大半を配列演算に費やすものもある．このようなプログラムでは，配列の各要素を参照する順序の違いがデータの局所性を変化させ，階層間のデータ転送回数に影響を与え，ジョブの走行時間を大きく左右する．例えば，配列 A(200, 200) を使用した次のプログラムはデータの局所性を悪くする．

```
      DO 10 I=1,200
      DO 10 J=1,200
   10 A(I,J)=A(I,J)*C      行方向アクセス
```

すなわち，FORTRAN における配列の記憶域への割付けは，A(1, 1), A(2, 1), A(3, 1), …, A(1, 2), A(2, 2), …の順に行われるため，上のプログラムのように書くと配列Aは200要素おきにアクセスされる．そこで，このプログラムの二つの DO の制御変数を入れ換えて次のように書けば，配列Aを列方向にアクセスし，データの局所性を高め，記憶階層間のデータ転送回数を減らすことができる．

```
      DO 10 J=1,200
      DO 10 I=1,200
   10 A(I,J)=A(I,J)*C      列方向アクセス
```

このように，FORTRAN プログラムで使用する配列のデータ構造の規則性に着目し，DO の制御変数を入れ換えて各配列要素のアクセス順序を変更することにより，データの局所性を高める最適化の方法について以下に述べる．

2.2 最適化の方法

DO の制御変数を入れ換える際，入れ換え前と後で DO の範囲の実行結果は等価でなければならない．こ

```
DO 文
DO 文
⋮
DO 文
┌──────┐
│ループ本体│
└──────┘
CONTINUE
⋮
CONTINUE
CONTINUE
```

図-1 DO ループの入れ子構造

```
DO 1 I¹=m₁¹, m₂¹, m₃¹
DO 1 I²=m₁², m₂², m₃²
⋮
DO 1 Iⁿ=m₁ⁿ, m₂ⁿ, m₃ⁿ
┌──────────────────────────────────────┐
│V(I¹+K₁¹, I²+K₁², ⋯, Iⁿ+K₁ⁿ)=⋯        │
│        ⋮                             │
│      =V(I¹+K₂¹, I²+K₂², ⋯, Iⁿ+K₂ⁿ)⋯  │
└──────────────────────────────────────┘
1 CONTINUE
```

図-2 DO 文と添字式の関係

こではプログラムの等価性を保って，どのようにこの最適化を実施するかについて述べる．

初めに以降で扱う DO ループの基本的な構造について述べる．

完全入れ子を成す DO 文から拡張範囲を除いたものを対象とし，一番内側の DO の範囲にはループを構成するような分岐文がないものとする．すなわち DO の範囲内からの飛び出し，DO の範囲内への飛び込みはなく，後方向に戻るような分岐が存在しないこと．また内側と外側の入れ子の間は DO 文および端末文としての CONTINUE 文でのみ構成されているものとする．このような入れ子を成すものをタイトリ・ネステド DO ループ[1]と呼ぶ．さらにループ本体（一番内側の DO の範囲から端末文を除いたもの）の文は代入文，IF 文，GO TO 文および CONTINUE 文で構成されるものとする．図-1 参照．

2.2.1 IF 文を含まない最適化の条件

説明を簡単にするため，ループ本体に分岐文が含まれない場合について，ループ本体で引用される配列と変数（ここでは配列を除く）について DO ループを入れ換える際の条件を述べる．ループ本体で参照（代入文の右辺に現われる）されるだけの配列，変数は DO ループ入れ換え前と後で等価性を保つことができるので，ここでは省略する．

（1） 配列の条件

ループ本体で定義（代入文の左辺に現われる）および参照（代入文の右辺に現われる）される配列は，DO の制御変数を入れ換える前と後で，ループ本体の等価性を壊す可能性がある．このような配列について，次に述べる等価性を保つ条件が必要である．

（a） 添字式の条件

DO の制御変数は必ず配列要素のすべての添字式中に出現し，同一配列名に関しては同じ次元位置を占めること．また，同一配列要素では二つ以上の次元位置を占めないこと．

（b） アクセス順序の条件

図-2 で示すようなタイトリ・ネステド DO ループを考える．ループ本体に現れる配列要素の添字式の形式は，I+K とし，I は DO の制御変数，K は整定数とする．

いま配列 V について，配列要素 $V(I^1+K_1^1, I^2+K_1^2, \cdots, I^n+K_1^n)$ が定義される点を f とし，配列要素 $V(I^1+K_2^1, I^2+K_2^2, \cdots, I^n+K_2^n)$ が定義もしくは参照される点を g とする．

ここで，set$\langle f, g\rangle$ を次のように定義する．

$\text{set}\langle f, g\rangle=(K_1^1-K_2^1, K_1^2-K_2^2, \cdots, K_1^n-K_2^n)$,

$n\leq 7$

任意の f と任意の g との間で，$a^i=K_1^i-K_2^i$ として，

$$\left.\begin{array}{l}\text{set}\langle f, g\rangle=(a^1, a^2, \cdots, a^n)\ \text{で}\ a^i\geq 0\{\forall i\}\\ \text{または,}\\ \text{set}\langle f, g\rangle=(a^1, a^2, \cdots, a^n)\ \text{で}\ a^i\leq 0\{\forall i\}\end{array}\right\}\quad(1)$$

のいずれかが成立する場合，DO の制御変数 $I^j(1\leq j\leq n)$ の任意の入れ換えが可能となり，入れ換え前と後の DO ループ本体の等価性が保たれる．

例 1

```
DO 10 I=1, N                          DO 10 J=1, M
DO 10 J=1, M        変換可能          DO 10 I=1, N
A(I+1, J+2)=B(I, J)*C  ⟹              A(I+1, J+2)=B(I, J)*C
   a₁          b₁                        a₁          b₁
B(I, J)=A(I, J)+D                     B(I, J)=A(I, J)+D
   b₂       a₂                           b₂       a₂
10 CONTINUE                           10 CONTINUE
```

例 1 の配列 A, B は添字式の条件を満たし，さらにアクセス順序の条件は

$\text{set}\langle a_1, a_2\rangle=(-1, -2)$,

$\text{set}\langle b_1, b_2\rangle=(0, 0)$,

であり，C, D はループ本体で参照のみであるから，左側の例は右側の例に変換可能である．

（2） 変数の条件

DO の範囲で定義および参照される変数について考える．

配列についてはその各要素のアクセス順序が問題となったが，変数の場合はループ本体でどのように定義・参照されるかというデータ従属性が問題となる．変数のデータ従属性を参照に先立って定義されるものと，参照された後に定義されるものとに分ける．

（a） 定義が先行する変数

定義が先行する変数Sは次のような形式でループ本体に現われる．

S=exp　　(2)

この exp の部分が，前述した配列のアクセス順序の条件を満足する配列要素か，定数，または既にループ本体中で(2)式で定義された変数で構成されていれば，変数 S の値は DO の制御変数の入れ換え前と後で等価性が保たれる．

（b） 参照が先行する変数

参照が先行する変数についてはループ本体に現れる形式を次のように限定する．

S=S+exp

この exp の中に S は出現せず，かつループ本体中にもこの式以外にSが出現しないこと．またこの exp の構成内容は「定義が先行する変数」で述べたものと同様とする．

この形態の式は log-sum アルゴリズム[2] を適用して解くことが可能であり，演算の実行順序を変更しても最終結果のSの値は保証される特徴をもち，Sの値は DO の制御変数の入れ換え前と後で等価性が保たれる．

2.2.2 IF 文を含む最適化の条件

次にループ本体中に IF 文を含む場合の条件を述べる．

ループ本体に出現する配列および変数に対して，前述したように，配列が DO の制御変数の入れ換え前と後で等価性を保ち，かつ定義が先行する変数があるとき(2)式を満足すれば，IF 文に関係なく DO の制御変数の入れ換えは可能となる．またループ本体に参照が先行する変数がある場合は log-sum アルゴリズム中の max, min 問題に置き換え可能な IF 文がある場合に限り，DO の制御変数の入れ換えが可能となる．

以上述べたように，配列，変数および IF 文に対して DO の範囲で等価性の条件を満足するなら，DO の制御変数の入れ換えは可能となり，最適な記憶階層に合せた配列のアクセス順に変更できる．

3. 実施システムの概要

我々は以上に述べた理論をもとにした最適化を実施する FORTRAN プリコンパイラ SSOPTRAN (Source to Source OPtimizing TRANslator) を開発した．ここでは，本論で述べた DO の制御変数の入れ換えの他に，このシステムが備える最適化について説明する．

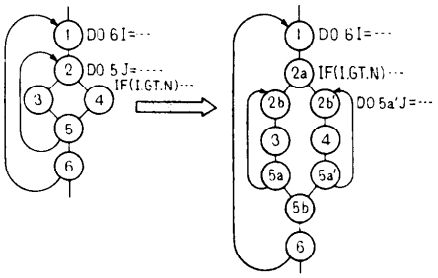

図-3 IF 文の移動例

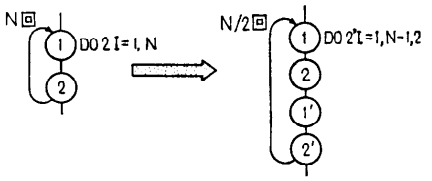

図-4 DO の実行回数半減例

（1） 利用者副プログラムの内部展開

利用者副プログラム（関数副プログラム，サブルーチン副プログラム）の頻繁な呼び出しを省略し，かつ他の最適化を促進するため，指定された副プログラムをその呼び出し部分に展開する．

（2） 演算評価順序の変更

グローバルな最適化の一つであるテキストの最適化を促進するために，可換律を有する演算子で結ばれる演算項の位置の変更を行う．これにより，ループ内不変式をより多くループの外へ移すことができる．

（3） 分岐に関する最適化

ループ本体にある分岐命令の実行回数を削減するための最適化であり，IF 文に着目したものと DO の実行回数に着目したものとがある．

IF 文に着目した最適化では，ループ本体にある IF 文の分岐条件を検査し，分岐条件が一定であれば IF 文をループ本体の外に移動する．**図-3** 参照．

DO の実行回数に着目した最適化では，DO 制御のための分岐命令の削減を図る．すなわち，ループ本体の実行文を2倍に拡張し DO の実行回数を半分にする．**図-4** 参照．

これにより分岐命令の実行回数の削減を図る．

4. 変換例と効果

SSOPTRAN を用いて DO の制御変数の入れ換え

を実施したプログラム2本について，その変換例と効果を示す．

プログラムとその変換例を例2,3に示す．

例 2

```
変換前プログラム
REAL A(150,150)
DO 30 M=1,5
DO 20 I=2,149
DO 10 J=2,149
A(I,J)=(A(I,J+1)+A(I,J-1)+A(I+1,J)+A(I-1,J))*0.25
10 CONTINUE
20 CONTINUE
30 CONTINUE
```

⇩

```
変換後プログラム
REAL A(150,150)
DO 30 M=1,5
DO 20 J=2,149
DO 10 I=2,149
A(I,J)=(A(I,J+1)+A(I,J-1)+A(I+1,J)+A(I-1,J))*0.25
10 CONTINUE
20 CONTINUE
30 CONTINUE
```

例 3

```
変換前プログラム
REAL A(256,256),B(256,256)
DO 50 I=1,256
DO 40 J=1,256
X=X+A(I,J)*B(I,J)
40 CONTINUE
50 CONTINUE
```

⇩

```
変換後プログラム
REAL A(256,256),B(256,256)
DO 50 J=1,256
DO 40 I=1,256
X=X+A(I,J)*B(I,J)
40 CONTINUE
50 CONTINUE
```

例2について，仮想記憶方式を採用する汎用中型計算機 FACOM 230-48 を使用して，経過時間およびページング回数を測定した結果を**図-5** に示す．この結果から明らかなように，配列を列方向にアクセスするか行方向にするかによって，ページング回数が極端に変化することがわかる．

一方バッファ記憶を有する大型汎用計算機では，プログラムの実行時間 (CPU) は次の式で近似できる．

CPU 時間＝(データがバッファ記憶上にあるとしたときの命令実行時間)＋(記憶階層間のデータ転送時間)

この関係を計算機性能に合せてグラフ化したものを**図-6** に示す．

例3の変換前プログラムは NFP が非常に高く，主

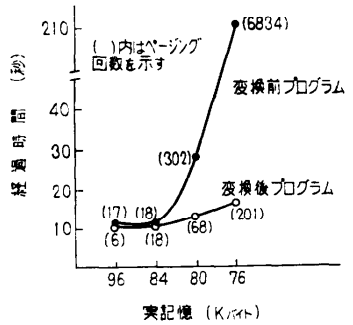

図-5 偏微分方程式のプログラムにおける VS 効率

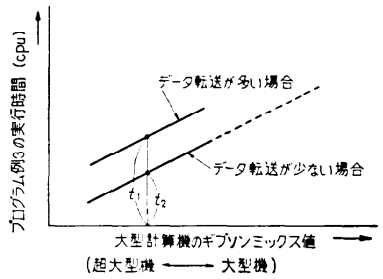

図-6 配列演算プログラムにおけるバッファ記憶効率

記憶とバッファ記憶とのデータ転送時間が大きくなっている．このプログラムの変換前実行時間を t_1 とし，変換後実行時間を t_2 とすると t_1/t_2 が 1.5～2.8 倍となる結果が一般的に得られる．この例でもわかるように，バッファ記憶を有する計算機で巨大配列の演算を行う際には記憶階層構造を考慮した最適化により計算機本来の性能を引き出せ，超大型機ほどその効果は大となる．

5. むすび

記憶階層構造を考慮した最適化を中心に述べ，その効果をプログラム変換例とともに示した．SSOPTRAN はプリコンパイラ方式をとり，問題プログラム指向の強い最適化を実施しているが，これにより，ジョブの実行コストを低減する効果は大きく，多くの科学技術計算ユーザがこのシステムを利用できると考える．

汎用計算機における仮想記憶方式やバッファ記憶方式など，記憶空間を階層的に使用する方法は将来も引き継がれるであろう．今後とも，この種の最適化技法が言語処理プログラムにおいて開発され，一般的に採用されることを期待する．

参 考 文 献

1) Leslie Lamport: The Parallel Execution of DO loops, CACM, Vol. 17, No. 2, pp. 83-93 (1974).
2) Kogge, P. M.: Parallel Solution of Recurrence Problems, IBM J. RES. DEVELOP., pp. 138-148 (1974).
3) David B. Loveman: Program Improvement by Source to Source Transformation, JACM, Vol. 24, No. 1, pp. 121-145 (1977).

(昭和54年11月20日受付)