数理解析研究所講究録
第199巻 1974年 1-20

FŌRTRANコンパイラにおける最適化
ŌBJECT CŌDE ŌPTIMIZATIŌN IN MVI/MVII-FŌRTRAN

富士通 LP部F/A課　棚倉 由行

§1. はじめに

FŌRTRANコンパイラにおける最適化には、種々の方法が考えられているが、ここでは、FACŌM230-75のMVII-FŌRTRANに適用した最適化の種類、最適化機能を用いる場合の考慮事項及び最適化に関する問題点について述べる事にする。

§2. 最適化の種類

MVII-FŌRTRANでは、オブジェクトプログラムの最適化を行っている。最適化には3つのレベル（ŌPT0，1，2）があり、ユーザはそのいずれかを選択できる。
この最適化の目的はオブジェクトプログラムの実行速度を上げようとするものであり、副産物的に、オブジェクトプログラムの大きさが小さくなる傾向にある。
ŌPT0はソースプログラムの動作に忠実なオブジェクトプ

ログラムを生成することを目的とし，ユーザのデバッグのやりやすさなどが特徴である。 OPT1は局所的な（ローカルな）最適化を施している。 OPT2の如くの最適化が不要の場合に用いられる。 OPT2はループに着目し，広範囲に渡っての（グローバルな）最適化を施している。 ループに着目するのは，プログラム中では，ループ内の手続きがダイナミックステップの多くを占めるものと考えているからであり，ループ内の手続きをできる限り減らし，高速度の命令に置換することにより高速度化を計っている。

最適化のレベル‘OPT0，1，2’はコンパイル速度がその順であり，オブジェクトプログラムの実行速度がその逆順であると考えることができる。 以下では，OPT2を中心に述べることにする。

2.1 プログラムの流れ，データの流れの解析

最適化はブロック及びループがその単位となっている。

(1) プログラムのブロック化

ブロックとは文の連なりであって，他のブロックからの入口は先頭の文だけであり，最後の文だけが分岐を含んでいるものである。 これはDO文，論理IF文等の特別な文を除いて，文番号間の文がブロックを形成すると考えることができる。

コンパイラはプログラムを、nodeがブロックを表わし、directed edgeがその流れを表わすControl flow graph(有向グラフ、directed graph)としてとらえている。

$$\left[\begin{array}{ll} \text{Control flow graph} & G=(B, E) \\ \text{the set of blocks} & B=\{b_1, b_2, \cdots, b_n\} \\ \text{the set of directed edges} & E=\{(b_i, b_j), \cdots\} \end{array}\right.$$

(2) ループ構造の把握

ループ構造をStrongly connected region(SCR_j)として把握する。その方法はインタバル(interval)による検出法(参考(4))を採用している。検出したループはDO文にて記述したループも、IF文にて記述したループも同様のものとして取扱う。($SCR_j=\{b_k, b_l, \cdots\cdots\}$)

(3) ブロックの優先関係の把握

プログラムの入口ブロックb_1よりあるブロックb_kへのパスの集合をPとすると、($P=\{p \mid p=(b_1, \cdots\cdots, b_k)\}$)
ブロックb_kの優先ブロック$PD(b_k)$(Predominators, b_i)は次の如く表わせる。

$$PD(b_k)=\{b_i \mid b_i \neq b_k \text{ and } b_i \in \cap P\}$$

(4) 各ループ内の関節接合ブロックの把握

あるループの(SCR_j)関節接合ブロック(articulation blocks) A_jは次の如くである。

$$A_j = \bigwedge_i (b_i \vee PD(b_i)) \wedge SCR_j$$

b_iはループの出口ブロックである。

(5) データの流れの把握

ブロック間に於けるデータの定義・参照関係を求める。あるブロックb_iであるデータに値の定義があり，その値を別のブロックb_jで参照している場合，ブロックb_iの出口でそのデータはbusyであるという。ここでは，各ブロックのbusy情報を把握している。

2.2 共通式の削除（Common expression elimination）

全く同じ結果をもたらす共通な2つの式が存在すれば，後の式では前の式の演算結果を用いるようにする。

（例）
```
M=I*10          M=I*10
 |               |
 |       →      t=M
 |               |
N=I*10          N=t
```

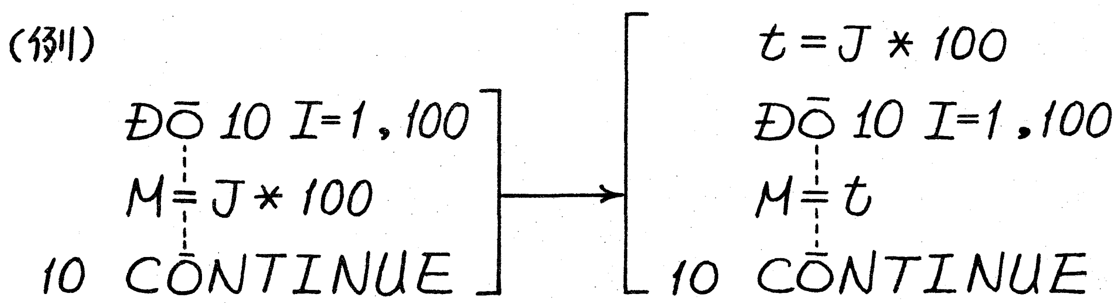
上例において，論理的に可能であれば，後の式はN=Mとなり，t=Mは不要となる。tはコンパイラの生成した名前である。この共通式の削除はブロック内の式及び優先関係にあるブロック間の式の間で処理する。

2.3 不変式の移動（Invariant instruction movement）

ループ内で演算しても，ループ外で演算してもその演算結果が変わらない式があれば，それをループ入口の直前のブロ

ックに配置転換する。 その配置転換先がループ（1つ外側のループ）で、その式がそこでも不変であるならば、そのループの外側へと再び配置転換する。

（例）



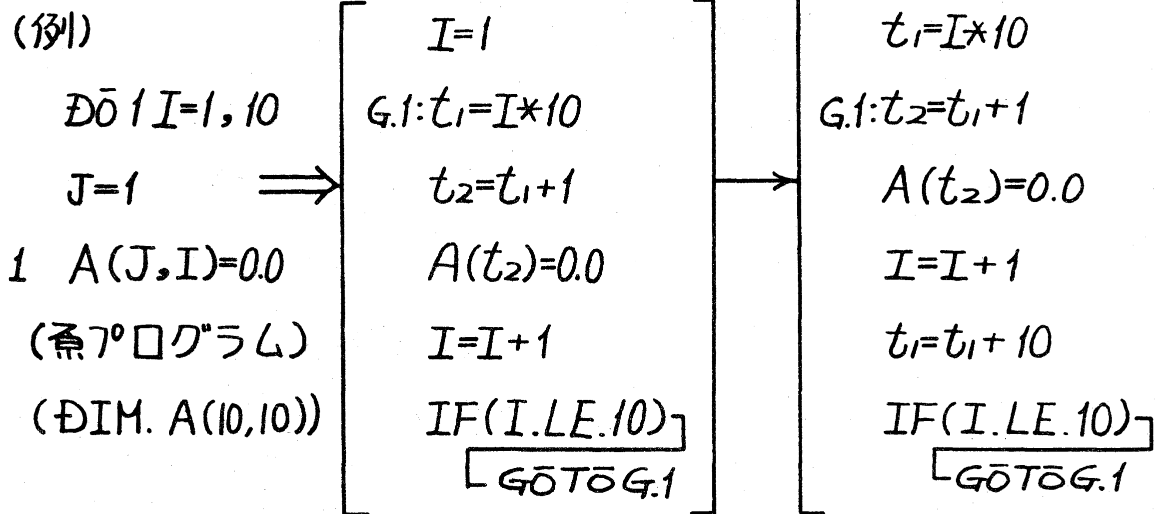
上例において，論理的に可能ならば，M=J*100 全体を配置転換する。

2.4 誘導変数の最適化

誘導変数（induction variable）とは，定数又はループ内では不変な変数によってのみ回帰的に増分される変数である。DOの制御変数などがその典型的なものである。 誘導変数の最適化とは，誘導変数（I）と定数又はループ内では不変な変数（C）との乗算，加算及び巾乗算，そして誘導変数同士の加算などの命令置換を行うものである。 IとCとの乗算（t=I*C）について例を示すことにする。

t=I*Cをループ入口の直前のブロックに配置転換し，Iの増分値*Cにてtを回帰的に加算する命令を生成しループ内に置く。 これにより，ループ内の乗算が加算になったことになる。 又，ここで生成したtの回帰定義により，t

は以後，誘導変数としてふるまう様になり，t に関する最適化が促進される。

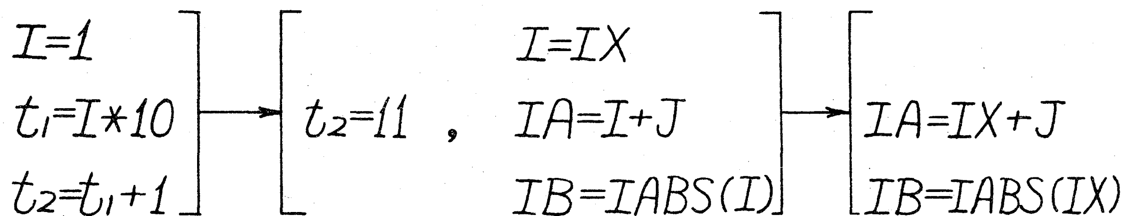


この様にして，誘導変数に関する乗算，加算，巾乗算などの最適化を進めて，最終的には，その過程で導入した誘導変数にて，ループ制御をも行う様にし，不要の回帰定義を削除している。　上例は次の如くとなる。

```
      I=1
      t1=I*10
      t2=t1+1
G.1:  A(t2)=0.0
      t2=t2+10
      IF(t2.LE.101) GOTO G.1
```

2.5 定数計算及び単純代入の削除（Folding and Simple store elimination）

定数計算とは，コンパイル時に計算可能な式，即わち，定数又は定数把握のできる変数をオペランドにもつ式の演算を行ってしまうことである。単純代入の削除とは A＝B の如くの単純代入について行うものであり，以後の A の参照を B の参照に置き換えることにより，その単純代入を削除してしまおうとするものである。



2.6 レジスタ割当ての最適化（Register optimization）

(1) 演算レジスタの割当て

ループ内のいくつかの変数又は定数をループ全体を通して，又はその1部の間，レジスタに保持する様にレジスタの割当てを行う。これによって，データのレジスタへのロード及びレジスタよりのストアを減じ，レジスタ間の命令を用いる様にして，メモリへのアクセスを減じている。ループ内では，レジスタを割当てられた変数は，そのレジスタ中で値が変わることになり，メモリ上の値は変わらない。そして，必要に応じて，ループ外又は内に，レジスタのロード命令，

ストア命令を置いている。

(2) インデックスレジスタの割当て

ループ内のいくつかの誘導変数及び配列要素の添字に対してインデックスレジスタを割当てる。ループ内でのそのふるまいは演算レジスタと同様であり，ループ制御など，高速度の命令を用いる様にしている。又，あるループ誘導変数とその外側ループの誘導変数との融合を行い，それに対してインデックスレジスタを割当てるなど，できる限り広い範囲に割当てる様にし，ループ制御の高速度化を計っている。

2.7 その他の最適化

(1) 文関数を文中に展開して組込み，他の最適化効果を含ませる。

(2) 添字式中の定数部を命令のアドレス部に組込む。

(3) 混合演算などでは，定数の型をコンパイル時に合わせる。

(4) 論理IF文での次の如くの単純な論理式の評価の最適化をする。

IF(A.OR.B) GOTO 10 → IF(A) GOTO 10
IF(B) GOTO 10

(5) 整数型乗除算を可能ならシフト命令に変える。

(6) 算術IF文については，その文番号の組合わせにより最適な命令列を決定している。

(7) DO仕様入出力リストで，それが全配列にわたる場合に

は，それを配列名のリストに変える。

READ(8,10)(A(I),I=1,10) ―→ READ(8,10) A

(8) 基本外部関数との引数の受渡しはレジスタ渡しとすることにより高速度化を計っている。

§3. 最適化機能を用いる場合の考慮事項

最適化を行ったオブジェクトプログラムのふるまいが，最適化を行わない場合と異なることがある。 ここでは，最適化機能を用いる場合の注意点及び実行速度を上げる方法などをプログラミング記法上から述べることにする。

(1) 割当て型GOTO文の文番号のリスト以外の文番号に分岐するプログラムは，その動作は保障できず，又最適化しなかった場合には発生しなかったエラーとなることがある。

(2) FORTRANシステム関数（SIN等）の名前をユーザ関数として使用すると，最適化の影響で，その関数内でCOMMONデータの定義，入出力，関数内データの値の保存なども行っている場合に，全く異った結果を招くことがある。 これは，拡張されたEXTERNAL文でユーザ関数名であることを宣言することにより防ぐことができる。

(3) 複数個の入口をもったサブプログラムをある入口で呼び出し，名前による引用（call by name）で引数の結合をして，その名前（アドレス）をサブプログラムに渡したとして，

（渡したと考えて）後に別の入口で呼び出す場合に，そのデータを引数とせずにサブプログラム内で参照したり，定義したりすると，全く異った結果を招くことがある。 これはサブプログラム内で名前による引用の引数を用いる場合には，それに対する引数を書くことにより防ぐことができる。
配列は，常に名前による引用であるので注意が必要である。

（例）

```
    I=1                   SUBROUTINE SUB(/I/)
    CALL SUB(I)           ENTRY ENT
    DO 10 I=2,N           IX = I*100
    CALL ENT              RETURN
 10 CONTINUE
```

上例では，サブプログラムにENTから入った場合に，Iの値が転送されないことがある。

（例）

```
    CALL SUB(I)           SUBROUTINE SUB(/I/)
    DO 10 J=1,N
    CALL ENT              ENTRY ENT
    IX=I+10               I=L*M
 10 CONTINUE              RETURN
```

上例ではループ（DO 10）外でI+10が実行されてしまうことがある。

(4) 最適化手続きによって，ループ内の演算をその実行頻度

の少い所に移動していることにより，最適化されていない場合と異なる実行動作をすることがある。

(例)

```
      DO 10 I=1,N
      DO 20 J=1,N
      IF(Y(I).LT.0.0)GOTO10
      X(J)=SQRT(Y(I))
   20 CONTINUE
   10 CONTINUE
```

SQRT(Y(I))をDO 20のループ外で実行させるために，Y(I)が負の場合にエラーメッセージが出てしまう。

これはそのエラーメッセージが不要であることを指定する機能（ERRSETシステムサブルーチン又は実行時のEXECパラメータ）によって防ぐか，又はY(I)が負であるかの判定をDO 20 J=1,Nの前で行うようにプログラミングすることにより防ぐことができる。

これは誘導変数の最適化によっても発生する可能性があり，オーバーフロー，ディバイドチェックなどの例外状態が発生することがある。

(5) あるサブプログラム呼び出しの実引数として同じものを2つ（2つ以上）用いている場合，それを名前による引用（call by name）で受けるサブプログラムでは，誤った結果が生じることがある。

(例)

```
      CALL SUB(X,X)
```

```
      SUBROUTINE SUB(/A/,/B/)
      COMMON C(100)
      DO 10 I 1,100
      A=Y/I
      C(I)=B+10
   10 CONTINUE
      RETURN
```

DO10ループ内のB+10がループ外で演算することになるため、A=Y/Iの演算結果が反映していない。

(6) サブプログラムに転送する引数はCOMMONデータとした方が速くなる。

(7) IF文で記述したループよりもDO文で記述したループの方が、一般に、速くなる。

(8) 配列の入出力リストはDO仕様で書くよりも配列名を書いた方が速くなる。

(9) 入出力操作の多いプログラムではファイルのブロッキングファクタ（制御カード(FDカード)で与える）を大きくした方が速くなる。

(10) 多次元配列の操作は、置き換えることができれば、低次元配列とした方が速くなる。

(11) 拡張範囲（extended range）をもったループ（例えば、拡張範囲をもったDOループ）を記述すると、その最適化の効果は小さくなる。

(12) EQUIVALENCE文内に現れた変数は最適化の効果が小さくなることがある。

(13) WAIT文の入力リストを書かないと最適化の効果が小さくなることがある。

(14) 最適化によって改善されるよりもむしろ悪くなることがある。 それは，ループ内よりも外の方が演算頻度が低いという仮定で，できる限り，ループ外で演算する様にしているのだが，場合によっては，ループ外の方が演算頻度が高いということもあることによる。 この様な場合は，ループ外で演算することを防ぐためにそれらの文をサブプログラムとすればよい。

(例)
```
      DO 10 I=1，100
      DO 20 J=1，100
      IF(A(I,J).EQ.0.0)A(I,J)=SIN(FLOAT(I))
   20 CONTINUE
   10 CONTINUE
```

A(I,J)の値がほとんど0.0でなく，A(I,J) SIN(FLOAT(I))がほとんど実行されないとしても，コンパイラは，SINの呼び出しをDO 20 J=1，100の前にて実行する様にしているので必ず100回，SINを呼び出してしまう。 これはA(I,J)=SIN(FLOAT(I))を

サブプログラムとすることにより，不必要なSINの呼び出しを防ぐことができる。

§4. 最適化に関する問題点

最適化を行うコンパイラを開発する際に発生した種々の問題点について述べる。

(1) 不変式の移動や誘導変数の最適化によって，ループ内の演算をループ外で実行することがある。 この場合に，そのループが動作する時には，常にそれらのオペランドに適当な値がセットされている場合は良いが，そうでない場合に，ループ外での実行において，オーバーフロー等のエラーが発生することがある。 これはあるループが動作する場合には，そのループ内の必ず実行する文（articulation blocks内の文）についてのみ最適化を施す様にすれば，解決する問題であるが，次の如くの例もあり，これではあまりにも条件がきついということから，目をつぶっている。

(例)
```
      DO 10 I=1,N
      IF(A(I).EQ.0.0)GOTO 10
    1 A(I)=B(I)+10.0
   10 CONTINUE
```

上述した条件を含めると，文番号1と10の間で最適化ができなくなってしまう。

これは前述した基本外部関数の引数のチェックをも含めて，そのエラーメッセージの出力を止めるという方法でしか解決していない。 但し，配列要素の参照などは，その添字がその配列外を指すことも考えられ，記憶保護侵害の様な致命的エラーになりかねないので，ループ内の必ず実行する文に限っている。 これは，複数入口をもったサブプログラムの名前による引用（call by name）の引数についても同様のことが言える。

(2) サブプログラムの呼び出しの引数として，同じものを2つ以上用いている場合に，誤った結果が生じることがあることを前述した。 これは，サブプログラムの仮引数を全く各々別の領域のデータとして把握している為に生じている。これを同一領域かもしれないとして取扱うということは，最適化の効果を防げるという点で条件がきついということ及び仮引数の標準的な記法が値による引用（call by value）であり，この場合は，各仮引数がサブプログラム内にデータ領域をもち，そのデータ領域で動作するという言語仕様であり，誤りとは言えないということから，制限事項として取扱っている。

(3) 最適化するとしないで，演算精度に差が生じることがある。 例えば，

X = A * B
Y = A * B * C

では，最適化によって，A * Bを共通式と把握し，前の式の演算結果がレジスタに割付けられ，後の式では，そのレジスタとCとの演算となることがある。　この様な場合と，最適化をしないで，各々の文にてA * Bを実行させた場合とで精度に差が生じる。　これはメモリとレジスタ上での浮動小数点データの表現精度が異る為に生じるものである。

(4) 共通式の削除において，次の様な場合に，A + Cを共通式とは把握していない。これを共通式と把握するには，オペランドの種々の組合わせについて試みることになるが，これは手続きが膨大となるわりには効果が少なく，又，演算精度の問題から，左から右に演算を進めることを言語仕様として保障しているなどによる。

A + B + C
A + C

(5) 拡張範囲 (*extended range*) をもったループ構造のあるプログラムでは，最適化の効果が小さくなることを前述した。　これは，一般的な言い方をすると，2つの（2つ以上の）入口をもったループと言うことができる。　この場合には，プログラムの流れの解析 (*control flow analysis*) で，ループとしての把握ができない為に，最適化の効果が小さくなってしまう。　プログラムの流れの解析に，実用的でより進んだ方法を導入する必要がある。

§5. おわりに

最適化を施すことの是非については，過去において，何度も論じられてきた。 この問題が論じられると，その不要論としてはコンパイル時間がかかる，ユーザのデバッグが難しくなるなどと言われる。

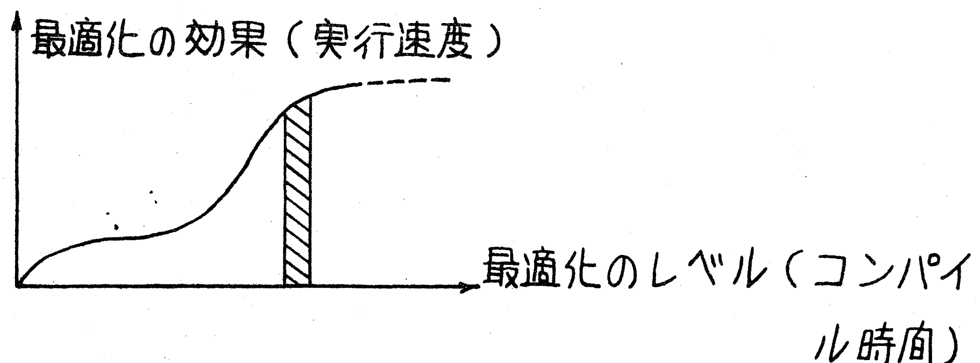
コンパイル時間と実行時間との最適な兼合いというのは1回のコンパイルに対する実行の回数による。
Compile and Goをしたいというユーザは，コンパイル時間と実行時間との和が最小になるものを要求する。
コンパイルよりも実行回数の方が多いユーザは，コンパイル時間がかかっても，その実行時間が最小となるものを要求する。 そして，プログラムによってもコンパイル時間と実行時間の比はまちまちである。 その他，ユーザによって様々な要求が考えられ，それら全てに対して満足する結論を出すことはできない。

デバッグが難しくなるというのは，最適化の効果の為に，ユーザの記述した手続きが，結果として同値なものをもたらす他の手続きへと変換され，ユーザが考えている以外の位置で手続きが実行されることによる。

我々は，これらの問題に対する1つの解決策として，最適化のレベル（OPT 0，1，2）を導入した。 即わち，そ

れらの解決を，選択の余地を残すことにより，ユーザに委ねることにした。

最適化は各部の調整をして，全体としてバランスのとれたものとする必要があるから，多くのプログラムについて調査分析し，その統計に基ずいて，どの様な最適化を施すべきかを決めなければならない。私のささやかな経験からすると，ローカルな最適化について，いくら多くのことを試みても，それには限界があり'最適化の効果／最適化のレベル'はあまり向上しない。そして，適当なグローバルな最適化を施すことにより，その傾きはずっと大きくなり，それも行きすぎると，その傾きは小さくなると考えられる。それを図示すると次の如くとなる。我々は，既存のプログラムについて，ハンドコンパイル等を試みたりして具体的な調査をし，次図の斜線の位置にその目標を置いている。



又，最適化手続きが膨大となるものは避けているし，ユーザの論理的ミスを積極的に助ける（不要の文を削除する等）様

な最適化は行わないものとした。

　コンパイラによる最適化というのは，あくまでも，その言語仕様に則って記述されたプログラムをできるだけ実行速度の速い機械語命令列に置換するだけの機能であって，その（数値）計算法については，何等，関与していないのである。従って，高速度の実行を目指すならば，まずその計算法について充分に吟味すべきだと思う。 そして，それをプログラミングする場合には，FORTRANを表現力のある高級言語としてとらえるならば，高速度化の為のテクニック（即わち，共通式の削除とか，ループ外で計算するなど）などに労力を用いず，意味のある記法（記述）を用いるのが良いと思う。 やたら苦心して，最適化をユーザが行うことにより，せっかくのFORTRANの表現力を殺してしまうよりも，高速度化は，機械に（コンパイラに）任せた方が，問題解決の手段として計算機を用いることの本質的な使い方ではないかと思う。

その他，最適化に適した言語仕様の設計をとの声もある。コンパイル速度及び実行速度において有利になる様に設計するのである。 即わち，ある程度の言語仕様の制限，並びに特殊な文の導入等である。 これにより，ある固有なハードウェア（例えば，F230－75）の能力を充分に引き出す

ことも可能となる。しかし，今日では，言語仕様の標準化による互換性の問題などが強く要求されていることもあり，やるべきではないと考えている。

以上，MVII－FORTRANコンパイラにおける最適化について述べましたが，今後とも，効率の良いオブジェクトプログラムを生成するよう改良を加えていく考えであり，ユーザ，その他の方々の御批判，御指導をお願い申し上げます。

1973.10.15

参考文献

(1) F.E.Allen 'Program Optimization' Annual Review in Automatic Programming Vol.5 1969
(2) E.S.Lowry and C.W.Medlock 'Object Code Optimization' CACM12 No.1 Jan. 1969
(3) V.A.Busam and D.E.Englund 'Optimization of expressions in FORTRAN' CACM12 No.12 1969
(4) F.E.Allen 'Control Flow Analysis' SIGPLAN Notices Vol.5 July 1970
(5) C.P.Earnest, K.G.Balke and J.Anderson 'Analysis of Graphs by Ordering of Nodes' JACM 19 No1 Jan.1972
(6) John Cocke and Raymond E.Miller 'Some Analysis Techniques for Optimizing Computer Programs' Proc. Second International Conf. of Systems Sciences Jan.1964