When logical conditions are as complicated as they are here, it is risky indeed to combine them. The whole thing should be separated into two stages, each doing its job in a proper order:

```
/* COMPUTE BASIC BILL */
   IF USAGE > 500 THEN
      BILL = 14.50 + 0.025 * (USAGE-500);
   ELSE IF USAGE > 100 THEN
      BILL = 3.50 + 0.0275 * (USAGE-100);
   ELSE IF USAGE > 50 THEN
      BILL = 2.00 + 0.035 * (USAGE-50);
   ELSE
      BILL = 2.00;

/* COMPUTE DISCOUNT FOR ELECTRIC HEAT */
   IF HEAT & USAGE > 10000 THEN
      BILL = BILL * 0.90;
   ELSE IF HEAT & USAGE >= 1000 THEN
      BILL = BILL * 0.95;
```

Concern for efficiency should be tempered with some concern for the probable benefits, and the probable costs.

Keep it right when you make it faster.

Here is another example, which replaces the first N elements of the array A by their factorials:

```
      SUBROUTINE ARRFAC (A,N)
      DIMENSION A(100)
      INTEGER A
      DO 2 I=1,N
      IF (A(I))2,2,4
C     FACTORIAL PROGRAM
    4 K=A(I)
      NFACT=1
      IF(K.EQ.1) GO TO 8
      DO 6 J=2,K
    6 NFACT=NFACT*J
    8 A(I)=NFACT
    2 CONTINUE
      RETURN
      END
```

Special handling is given the case where K (alias A(I)) equals one (but why doesn't the test branch to statement 2 instead of to 8?). Certainly the code runs slightly faster if K is 1, but the program is more involved. Remove the test, change the inner DO limits to J=1,K, eliminate the temporary variable NFACT, and the program still works well. Of course, it would be better if zero factorial were properly computed (it equals one), instead of being skipped as an error. At the same time it's easy to ensure that negative values are handled plausibly.