DEVELOPMENT OF A CRAY 1 VERSION OF THE SINDA PROGRAM

Susan M. Juba and Peter E. Fogerson
Lockheed Engineering and Management Services Company, Inc.

SUMMARY

The SINDA thermal analyzer program was transferred from the UNIVAC 1110 computer to a CYBER and then to a CRAY 1. Significant changes to the code of the program were required in order to execute efficiently on the CYBER and CRAY. The program was tested on the CRAY using a thermal math model of the Shuttle which was too large to run on either the UNIVAC or CYBER. An effort was then begun to further modify the code of SINDA in order to make effective use of the vector capabilities of the CRAY.

INTRODUCTION

The computer available for thermal analysis at the National Aeronautics and Space Administration/Johnson Space Center (NASA/JSC) is a UNIVAC 1110 which has a maximum user core of 190,000 words. However, since the 1110 is operated in a time-sharing environment with nonthermal users, its full resources are not available. This computational capability has not proven adequate for all requirements. For example, core storage restraints did not allow construction of a single thermal model of the Space Shuttle. Instead, five separate models were built - three representing the forward, mid, and aft fuselage sections, and one each for the aft propulsion system (APS) and the hydraulic system. A complete transient analysis thus required five interconnected computer runs.

In hopes of producing a combined Space Shuttle model and to provide a backup capability for computation during peak load periods on the UNIVAC 1110, a contract was made with the United Computing System (UCS) for time on their commercial networks. The basic host computer of UCS is a CYBER of the Control Data Corporation (CDC). A CRAY 1 (ref. 1) is also available; however, it requires the CYBER as a front end. The first task was to establish a CDC version of the thermal analysis program, SINDA (ref. 2), and to transfer the five thermal models to CDC files. The UCS CYBER configuration actually offered less core than the NASA/JSC UNIVAC. The CRAY 1, however, has 2 million words available, and additionally offers the prospect of increased speed by virtue of its vector processing capability. The ultimate purpose, then, was to develop a version of SINDA for the CRAY 1.

SYMBOLS

C_i	heat capacity of a node i, J/°K (Btu/°R)
G_{ij}	thermal conductance between node i and node j, J/hr-°K (Btu/hr-°R)
$GSUM_i$	net heating rate to node i, J/hr (Btu/°R)
i,j	node number indexes
Q_i	incident heat to node i, J/hr (Btu/°R)
R_{ij}	thermal radiation between node i and node j, J/hr-$°K^4$ (Btu/hr-$°R^4$)
T_i	current temperature of node i, °K (°R)
T_i'	new temperature of node i, °K (°R)
Δt	time interval, hr

STRUCTURE OF THE SINDA ANALYZER

SINDA is a general-purpose thermal analyzer, which means that the user can construct a thermal model of anything, unrestricted and unaided by geometry. It uses the finite difference formulation of the thermal diffusion equation, thus requiring a lumped parameter representation of the physical system in a resistor-capacitor (R-C) network. The program has two parts, the preprocessor and the processor. The preprocessor reads the user input data (the definitions of the nodes, heat capacities, temperatures, conductances, etc., which make the thermal model) in an 80-column card input format, and writes:

a. An executable program to perform the analysis

b. Tables listing the thermal parameter actual numbers (assigned by the user) vs. the relative numbers (assigned by the program)

c. A table called the first pseudocompute sequence (PCS1) which tells which nodes are connected to each other and by what type of connection

d. A table called the second pseudocompute sequence (PCS2) which contains the nonlinear thermal parameter information

The processor consists of the program (a) and user-selected subroutines from the SINDA library which, when executed, use as input the data files (b, c, and d) to produce a transient or steady-state simulation.

DEVELOPMENT

UNIVAC-to-CYBER Conversion

NASA was fortunate to have access to a version of SINDA which would run on CDC machines. This version was developed from a UNIVAC source and retained the exact bit configuration in the pseudocompute sequences; i.e., only the first 36 bits of the 60-bit CYBER word were used. The program was transferred to UCS along with an update package which would bring the version to the current UNIVAC level. Successful execution was easily achieved, but Central Processor (CP) second comparisons revealed the CYBER was requiring twice as much time as the UNIVAC. The problem was traced to the unpacking of the PCS data.

In UNIVAC FORTRAN, the unpacking is performed by a special function, FLD, which is fairly efficient. The statement:

```
NG = FLD(5,16,PCS1(I))
```

causes 16 consecutive bits, beginning with bit 5, to be taken from the location PCS1(I) and stored, right-adjusted, in the variable NG. The CYBER version had an assembly language subroutine, IFLD, to perform this function. It was of the form:

```
CALL IFLD(5,16,PCS1(I),NG)
```

These subroutine calls for unpacking the PCS data were replaced with in-line code using the CYBER SHIFT function and logical AND statements. For example, the call to IFLD shown above was replaced by:

```
DATA IAND6/01777770000000000000B/
NG = SHIFT((PCS1(I).AND.IAND6),21)
```

This causes a logical AND operation between the location PCS1(I) and the variable IAND6, thus picking out bits 5-21 of PCS1(I). The result is then left-circular shifted 21 bits before storing in NG. SUBROUTINE SUBFLD was used to pack the data in the first place. Using similar techniques, its calls were also replaced by in-line code. After making these subroutine call replacements throughout SINDA, the execution time required for some runs was reduced by a factor of 5.

CYBER-to-CRAY Conversion

Having established a working version of SINDA on the CYBER, the CRAY conversion effort was begun by making changes necessitated by differences in byte and word size between CDC and CRAY machines. For the purposes of this discussion, a byte is defined as the sequence of bits required to represent a character. The number of bits in a byte is therefore character-code dependent. Six bits are required to represent a character in Extended Binary Coded Decimal Interchange Code (EBCDIC) which is used on the CYBER, whereas eight bits represent a character in the CRAY version of the American Standard Code for Information Interchange (ASCII). The 60-bit word size of a CYBER machine allows representation of 10 characters per word, opposed to a maximum of 8 characters in the 64-bit word of the CRAY 1. The primary impact of this word and byte size difference is on the preprocessing/decoding processes, where node and conductor-related data are packed into words. The packing and unpacking processes which were implemented to conserve memory became unnecessary with the CRAY 1's 2 million word core memory. However, preprocessor and processor modifications to eliminate these steps would not be cost-effective in terms of programming and checkout time when weighed against the possible increases in execution speed. Therefore, the sections of code where data packing and unpacking were carried out were altered only where alphanumeric data are involved, i.e., where byte size is significant.

Another difference between CDC and CRAY FORTRAN exists in DO-loop handling. CDC FORTRAN will cause loops to be executed at least once, whereas a CRAY DO-loop need never be executed; as in a loop where the initial index value is 1, the final value is 0, and the incrementation value is 1. The correction for this DO-loop handling discrepancy is a simple one: CRAY FORTRAN provides a "J" option on its compiler control command that will ensure CDC-like handling of DO-loops when activated.

One characteristic of CRAY FORTRAN with global ramifications in the SINDA program is the ability to undefine variables. An entity will become undefined if an entity of different type which occupies the same memory location becomes defined. During SINDA processor execution, it is often necessary to access integers and floating-point values from the same array. The preprocessor packs integer information about the number of values in a real array in the first location of that array, and the entire vector is passed to an interpolation subroutine as a floating-point array. What happens upon entry to the interpolation routine is illustrated in the following simplified example:

```
1) SUBROUTINE INTERP(A)
2) DIMENSION A(1)
3) EQUIVALENCE (PN,NP)
4) PN = A(1)
5) LX = NP
```

These five lines of code show the original approach to accessing the integer value contained in A(1). When statement number 4 is executed by the CRAY-1, however, variable NP becomes undefined and statement 5 becomes a meaningless assignment. A quick-fix solution was discovered and is shown below:

```
1)  SUBROUTINE INTERP(A)
2)  DIMENSION A(1)
3)  EQUIVALENCE (PN, NP)
4)  PN = A(1)
4A) NP = NP
5)  LX = NP
```

The addition of statement 4A redefines NP so that the value in A(1) may be accessed as an integer. Statement 4A also has the effect of undefining PN, but since PN is not referenced after statement 4 in this application, further processing is not adversely affected.

As with the CYBER, it was also necessary in the CRAY conversion effort to replace with in-line code the calls to the two bit-manipulation routines, SUBFLD and IFLD. The substitution was accomplished through use of the Boolean selective merge function of CRAY FORTRAN, called CSMG. This function merges two words according to a third mask word, taking bits from word 1 where the mask word bits are set, and from word 2 where the mask word bits are cleared. This approach required that mask words be set up for each subroutine that had previously called SUBFLD or IFLD. In the preprocessor, these mask words could be defined in the driving routine, PREPRO, and accessed through a common block by other routines as necessary. In the processor, however, the driver is created uniquely for each run, so a different solution was needed. All routines that had accessed IFLD and SUBFLD were examined to determine what mask words were required, and DATA statements defining those words were added to each subroutine. A typical example of the changes involved in an IFLD substitution is illustrated below.

Bit manipulation previously effected by

```
VAR2 = IFLD(0,6,VAR1)
```

(right-justify leftmost six bits of VAR1 in VAR2) is now effected by

```
VAR2 = CSMG(SHIFTR(VAR1,(64-(0+6))),0,J6)
```

where J6 is a mask word with the six rightmost bits set.

SUBFLD substitution is slightly more complicated. If the original SUBFLD call was

```
CALL SUBFLD (5,1 VAR1,VAR2)
```

(replace bit 5 in VAR2 with bit 5 from VAR1) the replacement is

```
VAR2 = CSMG(SHIFT(VAR1,(64-(1+5)),VAR2,I5J1)
```

where I5J1 is a mask word containing one set bit in the fifth position from the left, where the leftmost bit is bit 0. The modifications to SUBFLD and IFLD calls in the preprocessor were made on a line-by-line basis, producing a factor of 3 decrease in execution time. Several hundred references to the two routines in the processor library made the use of statement functions more appropriate in the 51 library subroutines in which IFLD and SUBFLD were referenced.

MODEL EXECUTION COMPARISONS

After establishing working versions of SINDA on the CYBER and CRAY 1, some test executions on production size models were made. Selected were the MID model which has 1959 nodes and 15,271 conductors, and a combined Shuttle model which had no external radiation network, leaving it with 6,489 nodes and only 24,445 conductors. Table I shows the comparative performance of SINDA with the MID model on the NASA/JSC UNIVAC and the CYBER and CRAY 1 of UCS. The UNIVAC cost figure is based on \$457/hr; the CYBER and CRAY 1 figures are actual costs including the NASA discount. The decrease in execution time in CP seconds going from UNIVAC to CYBER to CRAY 1 was as expected, but there was no significant corresponding decrease in cost. A similar CP second/cost comparison for the UNIVAC and CRAY 1 using the combined Shuttle model is shown in table II. Only preprocessor data is shown because the model is too large to run the processor on the UNIVAC. The CYBER would not handle even the preprocessing of the combined model. The results are about the same as those of the MID preprocessor; the CRAY 1 gives a dramatic reduction in CP seconds but a slightly higher cost.

VECTORIZATION

After establishing a working version of SINDA on the CRAY and recording some comparative execution times for purely scalar code, the next effort was to incorporate some vector code into SINDA. The proportionately large amount of time spent in the network solution routines suggested that a vectorization effort in this area would be the most productive initially. Study led to the conclusion that merely applying vectorizing techniques to the existing code might not provide significant improvement, so a simultaneous attack was begun on the PCS structure.

Execution Routine Code Modification

The first routine to be examined for vectorization potential was the convergent explicit forward differencing routine SNFRWD. This routine was chosen because it produced consistent, reliable predictions of network response, was already one of the most efficient network solution routines, and consequently was heavily used. At this time, there are no cost comparison figures for any of the execution routines, but a brief discussion of the vectorization process for SNFRWD will provide some familiarity with the modifications involved in vectorization.

The ultimate objective when vectorizing code is to significantly reduce array processing time. This reduction can be achieved by accessing array elements so that memory banks are referenced sequentially, and by avoiding statements or constructs that inhibit the pipelining of operands and instructions.

In the SNFRWD routine, some loops cannot be vectorized effectively due to the extensive use of indirect addressing, and the presence of nonvectorizable external references and GO TO's. In loops that appear to be candidates for vectorization, but contain a few nonvectorizable statements, the solution can be the creation of two or more loops so that one or more will vectorize. CRAY FORTRAN offers vectorization aids in the form of vectorizable utility routines to replace conditional assignment statements within loops, as in this example adapted from SNFRWD:

```
C
C  THIS LOOP DOES NOT VECTORIZE
C
      DO 1 I = 1,1000
      LSUM = LSUM + LEN(I)*2
      IF (LEN(I) .NE. 0) LSUM = LSUM + 2
    1 CONTINUE

C
C  THIS CODE VECTORIZES FOR LOOP 1
C

      DO 1 I = 1,1000
      LSUMT(I) = CVMGZ(LSUMT(I) + LEN(I)*2,
               LSUMT(I) + LEN(I)*2 + 2, LEN(I))
    1 CONTINUE
      RSUM = SSUM (1000,LSUMT(1),1)
      LSUM = IFIX (RSUM)
```

Note that scalar summing variable LSUM in the original loop is replaced by temporary array LSUMT in the vector version to avoid scalar register use. With up to four million words of core memory available, and considering the high cost of CRAY processing time, this trade-off becomes economically justifiable in many cases. The vector utility routine CVMGZ performs the conditional test of the original loop and allows the assignment statement that preceded the test to be performed at the same time. CVMGZ tests the third argument, LEN(I), against a zero value, and if the test succeeds, LSUMT(I) is assigned the value of the first calling argument. If the test fails, the value of the second argument is used. LSUMT and LEN are typed real for the second example, so that real library function SSUM can be used to sum vector LSUMT. This real sum is then converted to type integer. The vector version of the loop actually contains more code, plus an external reference, but executes more than 12 times faster than the original. For an iteration count of five or less, vectorization of this loop would not have resulted in any time savings because of the required post-loop processing. Generally, the more times a loop is executed, the greater are the potential savings from vectorizing it.

PCS1 Restructure

The R-C network is contained in PCS1. Each word contains several pieces of information, as illustrated in figure 1. Each node in the network has one of these words for each connection it has to other nodes; i.e., if node 10 is connected to four other nodes, then the portion of PCS1 belonging to node 10 will be four sequential words. Obviously, the length of PCS1 is twice the number of conductors in the network. All current SINDA execution routines perform the new temperature calculations with an outer DO loop with the range of 1 to the number of nodes and an inner loop whose range varies depending upon the number of connections for that node. The basic forward difference formulation is an example:

$$T_i' = T_i + \frac{\Delta t}{C_i}\left[\sum_j G_{ij}(T_j - T_i) + \sum_j R_{ij}(T_j^4 - T_i^4) + Q_i\right] \tag{1}$$

The inner loop, using PCS1, sums the G_{ij} on R_{ij} ΔT product for each conductor to the node and the outer loop does the calculation of the new temperature, T_i , for each node.

Extracting data from bits of a word is an expensive process in terms of computer time. However, the increase in allowable problem size resulting from packing of the data was judged to be worth the price for the machine on which it was first coded; for the CRAY 1 it is not.

Assuming the inner loop work has been done and stored, i.e.

$$GSUM_i = \sum_j G_{ij}(T_j - T_i) + \sum_j R_{ij}(T_j^4 - T_i^4) \tag{2}$$

the equation resulting from substitution into equation (1):

$$T_i' = T_i + \Delta t/C_i(GSUM_i + Q_i) \tag{3}$$

is well suited for vector processing because all terms are vectors of length i. However, the largest amount of time is spent in forming the $GSUM_i$ term, and the code dictated by the PCS1 structure is not well suited for vectorizing.

One approach taken to make SINDA more efficient on the CRAY 1 was to restructure PCS1 so that a new execution routine could be written which did not require unpacking of data and which contained vectorizable code. (See fig. 2.) The original PCS1 contained four types of information:

a. type of conductor - bits 0-4 and 21
b. conductor number - bits 5-20
c. adjoining node - bits 22-35
d. subject node - implied by current location in PCS1.

The new PCS1 has the conductors sorted by types (instead of by subject node as in the original) and three arrays, one specifying conductor number, and two to list the connected nodes (subject node and adjoining node).

Currently, the new PCS1 is being built from the old PCS1 in a post preprocessor operation, as opposed to recoding the preprocessor to construct it in the new form. It is anticipated that this overhead cost will be more than offset by the elimination of unpacking and the resulting existence of more vectorizable code.

CONCLUDING REMARKS

We have shown that it is possible to build and use for analysis thermal math models on the CRAY 1 computer of UCS that are much larger than those allowed on our local computer, the JSC UNIVAC 1110. We have presented data showing the CRAY 1 executing scalar code to be competitive with the UNIVAC 1110 for the same model. An additional small cost reduction would probably result if the total cost of running the five UNIVAC models were compared to the cost of a single execution of the combined model on the CRAY. However, the largest potential cost savings lie in developing efficient vectorized code for the CRAY. There is no exact method for determining the point at which run time savings gained through vectorization are offset by programming costs accrued in the modification process. Program portability, frequency of use, and life expectancy, along with programmer hours and expected savings should be considered before beginning any vectorization effort.

REFERENCES

1. CRAY-1 Computer System FORTRAN (CFT) Reference Manual. CRAY Research, 2240009, May 1980.

2. SINDA User's Manual. TRW Systems, 14690-H001-R0-00, April 1971.

TABLE I.- COMPARATIVE PERFORMANCE OF SINDA WITH THE MID THERMAL MATH MODEL

	UNIVAC		CYBER		CRAY 1	
	CP seconds	Cost	CP seconds	Cost	CP seconds	Cost
Preprocessor	2059.0	$261.38	1034.3	$304.26	244.1	$279.39
Processor 0.2 - 45.4 hr	5762.0	$731.45	2248.0	$614.38	451.7	$506.33

TABLE II.- COMPARATIVE PERFORMANCE OF SINDA WITH THE COMBINED SHUTTLE THERMAL MATH MODEL

	UNIVAC		CRAY 1	
	CP seconds	Cost	CP seconds	Cost
Preprocessor	6541.0	$830.30	830.6	$1012.07

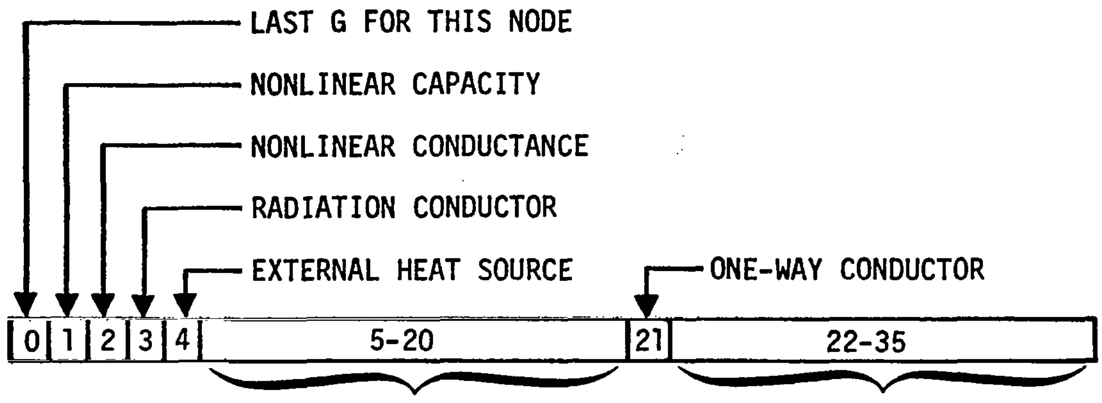

Figure 1.- PCS1 packed data structure.

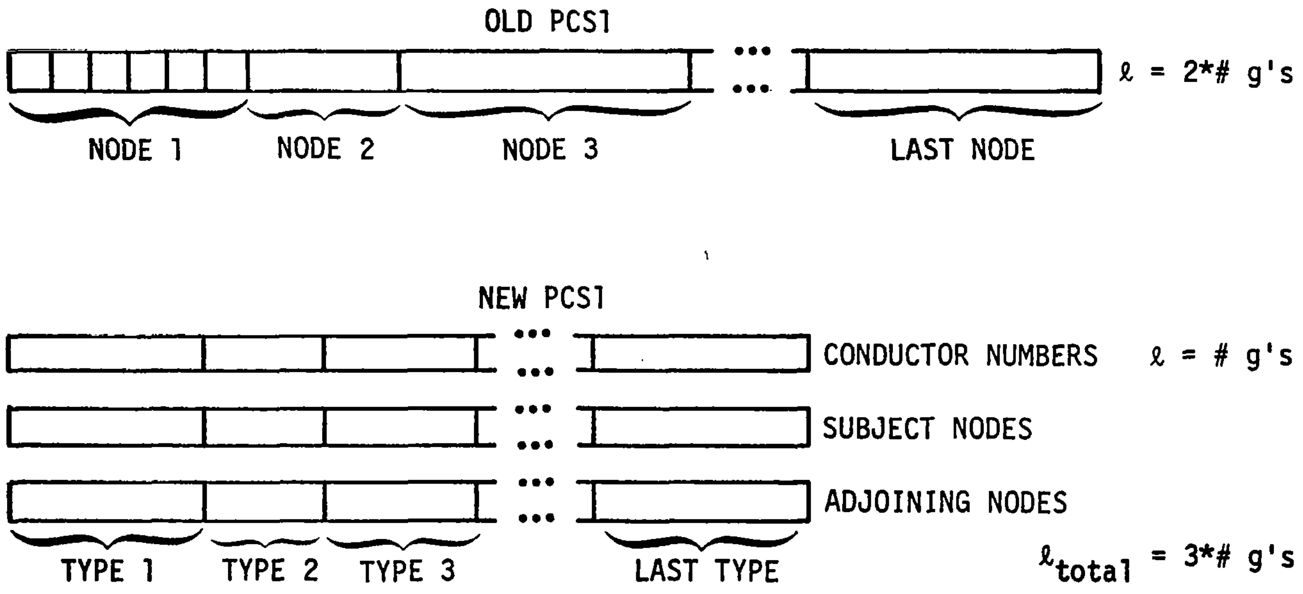

Figure 2.- Old versus new PCS1 organization.