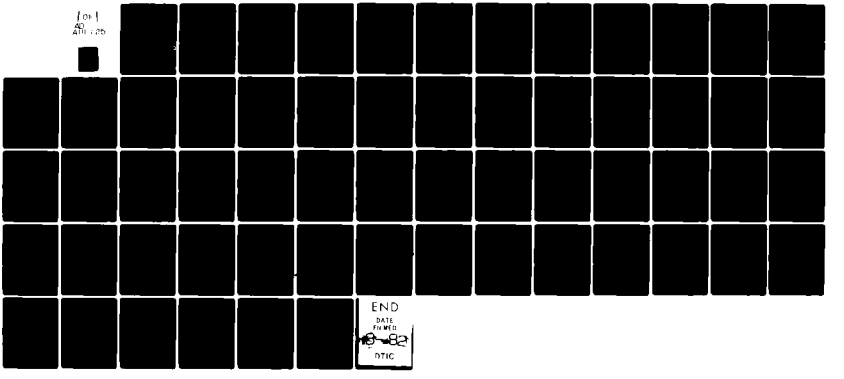
AD-A111 125 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 13/13
THE APPLICATION OF FINITE ELEMENT SOLUTION TECHNIQUES IN STRUCT--ETC(U)
DEC 81 R J HAMPSON
UNCLASSIFIED AFIT/GAE/AA/81D-13 NL



(1) LEVEL II

AD A111125

DTIC FILE COPY

DTIC
ELECTE
FEB 19 1982
S D
B

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

82 02 18 072

AFIT/GAE/AA/81D-13

(1) LEVEL

THE APPLICATION OF
FINITE ELEMENT SOLUTION TECHNIQUES
IN STRUCTURAL ANALYSIS
ON A MICROCOMPUTER

THESIS

AFIT/GAE/AA/81D-13 Robert J. Hampson
Capt USAF

Approved for Public Release; Distribution Unlimited

AFIT/GAE/AA/81D-13

THE APPLICATION OF FINITE ELEMENT SOLUTION TECHNIQUES IN STRUCTURAL ANALYSIS ON A MICROCOMPUTER

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by
Robert J. Hampson, B.S.E.M.
Captain USAF
Graduate Aeronautical Engineering
December 1981

Table Of Contents

		Page
Preface		i
List of Figures		ii
List of Tables		iii
Abstract		iv
I.	Introduction	1
	Objective	1
	Background	1
II.	Sequential Programming	5
	Concept	5
	Implementation	6
	Evaluation	8
III.	Overlays	19
	Background	19
	Concept	19
	Implementation	22
	Evaluation	27
IV.	Modularization	31
	Background	31
	Concept	31
	Implementation	33
	Evaluation	34
V.	Basic Skills	39
VI.	Implementation Problems	42
VII.	Conclusions	46
VIII.	Recommendations	49
IX.	Bibliography	50

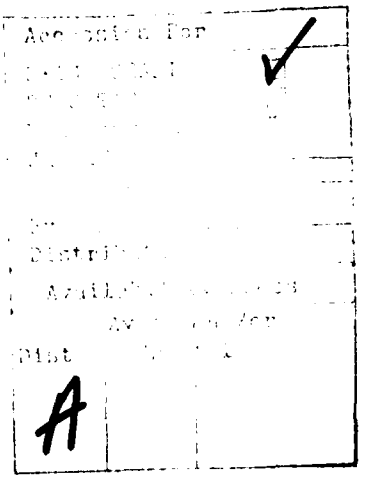
Preface

I thank my thesis advisor, Captain Hugh C. Briggs, for his patience and help in this research.

I thank my wife for her support of this research project and the amount of time she spent helping me in preparation.

Thanks go to the personnel at Computer Solutions, Inc., for the help in basic operation of the Cromemco System III microcomputer system.

Robert J. Hampson



List Of Figures

Figure		Page
2.1	Simple truss structure.	9
2.2	Memory map.	13
2.3	Modified memory map.	14
3.1	Overlay memory maps	20
3.2	STAP flow chart	23
3.3	Overlay breakdown	25
4.1	Finite element modular architure.	32
4.2	Modularized STAP program.	34
4.3	Simplified Truss Structure.	36
6.1	General debugging subroutines	45

List of Tables

Table		Page
2.1	Timing Data,Simple Stuss,Constand Bandwidth .	1Ø
4.1	Timing Data,Modularized STAP.	35

Abstract

This project explored the feasibility of applying finite element solution techniques in structural analysis on a microcomputer. The problem was explored with respect to the type of programming formats that can be used in finite element programs. Three formats were explored. These were the sequential, overlays, and modular format.

The results of the research indicated the maximum problem size capabilities on the Cromemco System III computer system, a 64K microcomputer. The effects of bandwidth on total problem size were noted. In addition the skill level required of an individual using a microcomputer system was addressed.

Microcomputers have the capability to work in the finite element field. By careful structuring of the problem, and analyzing what the capabilities of the microcomputer are, maximum utilization of the system can be obtained.

THE APPLICATION OF FINITE ELEMENT SOLUTION TECHNIQUES IN STRUCTURAL ANALYSIS ON A MICROCOMPUTER

1. Introduction

Objective

The purpose of this research was to demonstrate the feasibility of using a microcomputer system for structural analysis involving finite element solution techniques.

Background

In the past decade, giant strides were made in computer technology. An entire new class of computer, the microcomputer, has emerged. The microcomputer is the smallest in the computer line, excluding the hand calculators. The memory of this class of computer ranges from 2 K to 100 K-bytes, that is, 2000 to 100,000 bytes of memory space. This is in comparison to the 1.5 million bytes of memory that a minicomputer has, and the several million bytes that the main frame computers have. In addition to the obvious difference in size, the other major feature of the microcomputer is that it is a single user computer. The entire computer is usually dedicated to a single job. The larger computer systems are multi-user

systems where several individuals and programs use the resources of the computer at the same time.

At the present time, use of the microcomputer is generally limited to a secretarial role. It is used for data processing, storage, and retrival. The computational abilities of the computer are applied only to very simplistic programs, much like a super calculator. This limited usage of a relatively powerful instrument ignores the tremendous potential of the microcomputer.

The computational abilities of a 64K microcomputer equal that of many early main frame computers. The microcomputer has the ability to handle many types of problems which require the solution of a series of simultaneous equations. The solution techniques normally use a matrix format in the solution. These types of problems come up in aerodynamics, civil engineering, structural engineering, and other related fields. Applying the problem solving abilities of the microcomputer to these fields is a distinct possibility being almost totally ignored.

At the present time, microcomputers are located in almost every laboratory and educational environment. For example, at the Air Force Institute of Technology, there are approximately 20 microcomputers in use. The application of numerical solution techniques, and in particular finite element techniques, can greatly expand the capabilities of these systems.

Over the past 15 years there have been many finite

element programs developed, using various programming techniques. These programs include SPAR, NASTRAN, STAP1V and its derivitives, and GIFTS. These programs are general purpose programs, designed to solve a variety of problems in structural analysis, using a wide variety of elements. The problems that can be solved include static analysis, frequency response, and buckling. The programming techniques used in these and other programs included sequential programming, overlay programming, and modularization.

This study explored the ability of a specific microcomputer, the Cromemco System III, to handle structural analysis problems using finite element solution techniques. This study examined the techniques used in building the finite element program. Because of the microcomputer's limited memory capacity, programming efficiency is required for maximum problem solving capacity. The efficiency of a specific program is highly dependent on the format that the program is written in. These programming formats or techniques were evaluated from the aspect of how well the microcomputer could handle the program, as well as how efficient the programming was in relation to microcomputer application.

There are a variety of programming techniques in use today. The types of programming considered in this study can be referred to as sequential programming, overlay programming and modularization. Sequential programming is the basic programming technique taught. Simply put, it

requires the entire program to be stored in core until program execution is completed. The overlay technique is a method of limiting the required core dedicated to program by breaking the program into blocks, and having these blocks of program in core only when needed. The modular format involves a series of independent programs operating in a pre-determined sequence. Only a single program lives in core at any given time.

In addition to the applicability of finite element analysis to the microcomputer, the skill level of the user was examined. This skill level included both programming skills and system operation skill. Unlike most minicomputer and main frame systems, the individual who uses the microcomputer is usually the system operator as well. There are no technicians to handle the hardware operation of the system. If the system has hardware failures, the user must handle any maintainence necessary. The basic question here was can an engineer with computer skills handle the system, or does it require a computer programmer with engineering skills.

II. Sequential programming

Concept

The most common programming technique in use is sequential programming. This is the simplest of the programming techniques. Sequential programming is taught in basic computer programming courses. Simply stated, sequential programming requires all of the program and all of the data to exist in memory at the same time. There are no provisions to reduce the required size of memory by limiting the amount of data existing in core at any given time, except by limiting the size of the problem.

There are advantages to sequential programming. First is that sequential programming is the simplest way to program. Successfull programs are constructed by placing all required subroutines in core at once. Because all the data and equation solvers live in core memory at the same time, the data storage and retrival systems are much easier to construct. Because sequential programing is taught in most basic programming courses, this format is the most most familiar method of programming. There is a large amount of written material available to aid the individual when problems arise. Finally, there is the fact that most computer operating systems are designed to accept sequential programming formats. This is the case for the Cromemco microcomputer, and as a result, a sequential program is the easiest to implement on the system.

Even though sequential programming has many advantages, it also has a significant disadvantage. This disadvantage relates primarily to the application of the program to a microcomputer. The biggest problem is that sequential programming takes a great deal of memory to implement. This is a serious problem because the most severe drawback of the microcomputer is its lack of core memory space. This type of programming makes it difficult to use the microcomputer system's greatest asset, a large external storage capacity. The Cromemco system has over 4.8 megabytes of external disk storage. With sequential programming, this disc storage is not utilized to any great degree. It is this disadvantage that limits the use of the microcomputer in relation to finite element application..

Implementation

In order to determine the ability of micro-computers to handle sequentially programmed finite element codes, this type of program was implemented on the computer. The two choices were to create a sequential finite element code, or modify an existing code to fit the Cromemco system. The choice was made to modify an existing finite element code. Since the basic goal was to determine if finite element solution techniques were viable on microcomputers, the use of a working program insured that any initial problems would be problems stemming from the microcomputer, not the program. In addition this code would form the basis for expansion into a general purpose code, if the microcomputer

could handle it.

There were several requirements that the example code had to meet in order to be useful in meeting the goals of evaluating finite element progrms and microcomputers. The first was that it had to be written in FORTRAN. This is the standard engineering language, and in addition, a FORTRAN compiler was the only compiler purchased with the Cromemco system. The second requirement was that the program had to be in the public domain. The program should also be designed as a simple analysis code. A general purpose program would not be suitable for the starting point. Finally, in order to be useful, the initial program had to have enough documention to allow implementation on the microcomputer.

Several programs were considered. As mentioned earlier, the most famous codes were the general purpose codes. These codes were not suitable for the initial program choice. They were obviously too large and complex to fit on the microcomputer system. Several texts ((1),(2),(3)) had programs written for specific elements. The initial program was chosen from these sources. The code selected is called STAP.(3) It is a derivitive of the general purpose program SAP(4), written by Dr. E.L. Wilson of the University of California. STAP satisfied all the requirements and had two additional advantages. The first was that it was based on a much larger program. This makes for an easier expansion into a general purpose program, if needed. The second was that it was written with some

attention paid to core memory restrictions.

Instead of using full matrix storage, the program uses a modified banded storage scheme, skyline storage (3). It also utilized scratch files on disk storage to store some of the element information. These two features limited the amount of core memory space dedicated to program storage, with a resulting increase in memory for data storage and solution.

Installing a program on a new system presents two types of difficulties. The first is to insure that the code is properly sized for the system in question. The second is in the creation of additional subroutines. The details of the sizing of STAP code are given in Chapter 5. The subroutines created for the STAP code were for operating system utility changes and user convience. For example, in order to use the timing provisions of the STAP code, the called subroutines had to be consistent with the timing capabilities of the computer. Also, the manual data input was changed to use a mesh generator for rapid changes in input data. These additions allowed the code to be quickly and more easily analyzed.

Evaluation

There were two basic questions which had to be answered concerning the microcomputer in relation to a sequentially programmed finite element program. The first was how big of a problem could the microcomputer solve. Obviously, if the largest problem that the system can handle is on the same

level that can be solved by hand, then the microcomputer would not be very useful. Conversely, if maximum problem size was on the order of 100-200 degrees of freedom, then microcomputer usage would be of some value.

The second question dealt with how fast can the microcomputer solve the problem, and how accurate is that solution. A primary advantage of the miocrocomputer is its convienience. In order to be used instead of a central site machine, the engineer must be able to make several runs during the time that the central site will have only a single run. If microcomputer turn around times are on the same order as the central site, then usefullness of the microcomputer is limited.

The first series of problems consisted of a simple truss structure, which was expanded horizontally to increase the number of degrees of freedom. This technique created a constant bandwidth for the problem series. This type

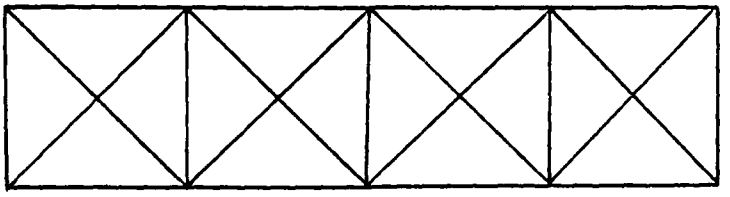


Figure 2.1
Simple Truss Structure

Screen Display

NODES	DOF	INPUT TIME	K FORM TIME	SOLVE TIME	TOTAL TIME
118	218	146	36	87	268
100	186	126	31	75	231
80	148	103	25	60	188
60	108	81	18	45	144
40	70	58	12	28	99
20	32	36	7	14	57

Disk Storage

NODES	DOF	INPUT TIME	K FORM TIME	SOLVE TIME	TOTAL TIME
118	218	210	36	93	339
100	186	188	31	81	300
80	148	154	25	65	244
60	108	117	18	51	180
40	70	82	12	34	129
20	32	51	7	15	73

Print Display

NODES	DOF	INPUT TIME	K FORM TIME	SOLVE TIME	TOTAL TIME
118	218	277	36	101	414
100	186	245	31	89	365
60	108	155	18	57	230
20	32	67	7	16	90

*Time in seconds

Table 2.1
Timing Data-Simple Truss,Constant Bandwidth

problem would model a simple bridge. The structure was supported at both ends and at various points along the base of the structure. The loads were placed vertically along the top of the structure. The diagonals were considered as independent elements, with no node points at the intersection. The problem was also limited to the two dimensional case. The truss element allows translation in three directions. The total number degrees of freedom in the the two dimensional case is twice the number of node points minus the supports where no translation is allowed.

With the constant bandwidth of 6, the microcomputer was able to handle a structure with 286 elements and 218 degrees of freedom. Using disk storage for the solution, total solution time was in the five minute range, with I/O accounting for 55% and floating point computations in the soluton and K matrix formulation 45%. (Table 2.1)

Using different means of solution storage or presentation produces different through put times. This was strictly as a function of the I/O operations. The time required for a write to disk is much different than for a write to screen or printer. Table 2.1 presents all the data. The fastest of the methods was a screen display. This is reasonable due to the fact that there is the least amount of input/output time involved in a screen presentation. This took only 4.5 minutes for a problem of 218 degrees of freedom. The slowest time was for a printed copy of the results. This took 7.5 minutes, due to the speed limitations of the printer. The disk storage took 5.7

minutes for total solution time.

These times indicate that the most advantageous system would be to displsy the results on the screen. This gives the fastest solution time. The drawback is that there is no permament copy of the results. In order to produce the hard copy, the longer times associated with printer display or disk storage would have to be tolerated. Even using the longest time figure, 7.5 minutes, the time factor is not a driving factor. Assuming that 218 degrees of freedom is a workable problem size, the concerns about excessive time for solution are unfounded.

The problem was then altered to create a system with a larger bandwidth. Most real problems are not of the format of simple truss bridges. They have larger bandwidths when the stiffness matrices are generated. Because bandwidth determines array storage within the program, the larger bandwidth increases program core requirements, and reduces the number of degrees of freedom that can be handled by the program. By increasing the maximum bandwidth from 6 to 15, the total degrees of freedom dropped from 218 to 204. This is not a disastrous reduction in problem size, but it does demonstrate that care must be taken when node patterns are created. Band widths should be made as small as possible for maximum efficiency.

These results indicated that the microcomputer could handle finite element problems of some magnitude. The next step was to expand computer capabilities within the sequential programming format. Using a bandwidth of 6 for

the STAP code, each additional K-byte of core memeory allocated to data means an increase in solvable size by 7 degrees of freedom.

To increase the amount of user accessible memory, an attempt was made to use system dedicated core. The only possibility was that part of core memory allocated to the system software for linking the program at link time. Once linking has been accomplished, this space is not needed by the operating system. Figure 2.2 is a general representation of the memory of the microcomputer. The normal software package loads the data on the bottom of the

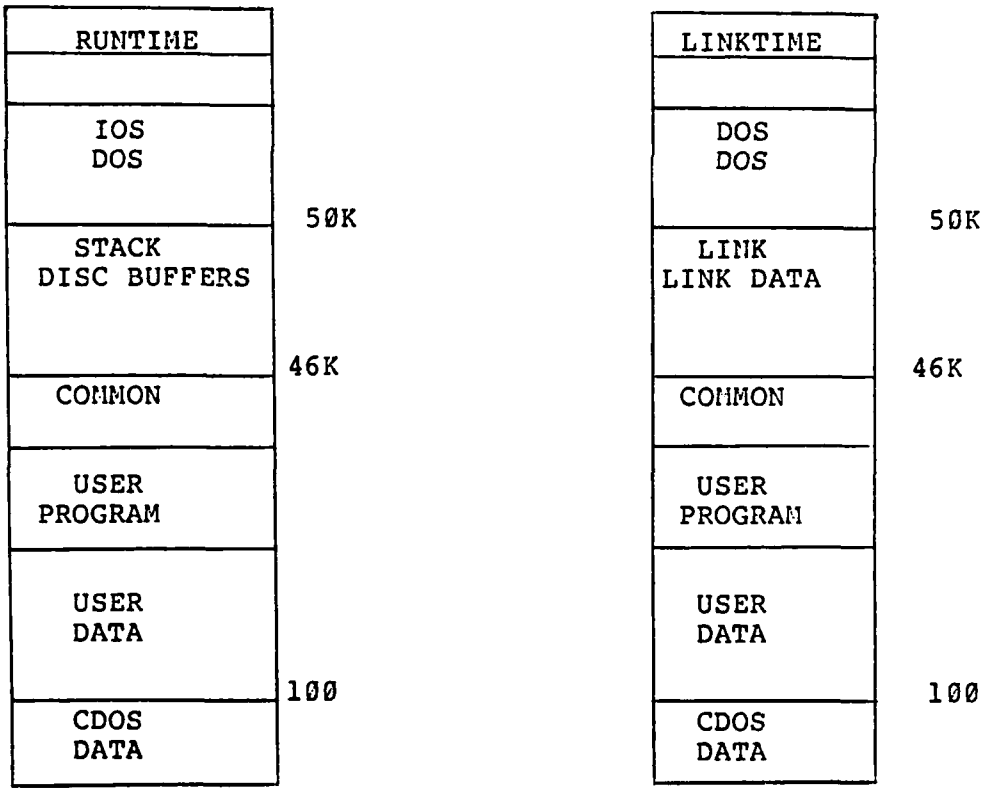


Figure 2.2
Memory Map

memory, followed by the program on top of the memory. The memory between PROGRAM and LINK DATA is memory available to the user. By using the LINK DATA memory during execution, user available memory is increased. This space is in the range of 2-4 K-bytes.

In order to obtain this memory, two ideas were used. The first is part of the FORTRAN language. FORTRAN does not moniter array indices at runtime. It is possible to put more elements into an array than the declared length of the array. The second was the ability to load the program in a reverse order, program first, followed by data. Figure 2.3

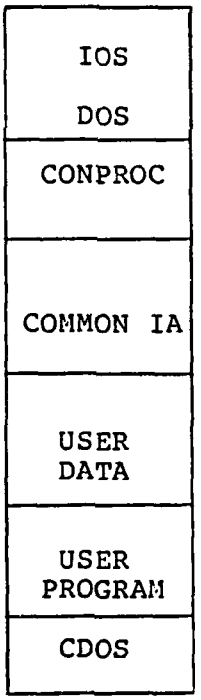


Figure 2.3
Modified Memory Map

shows the memory map for this format at runtime. The STAP code utilizes the blank commom IA as the main storage loaction. By having blank common as the last item loaded, this array could be loaded up into the base of the COMPROC area. The 2-4 K-bytes of memory in question represent 15-20 degrees of freedom.

Unfortunately, this attempt was not successful. The failure lay in the fact that the run-time package was not open to observation. There were input/output buffers that always lived above the last item stored in common. Finding the location of these buffers and their format required more computer skills than were available. It would take a computer programmer, not a structural engineer, to determine these facts. Given the facts, the problem could be solved.

The next step was to expand the sequentially formatted program into a general purpose code. This type of program is usually more useful than a single element code in structural analysis. The first element added was the beam element. The beam element can be found in most structural analysis text. (1),(2) The element allows three translation and three rotation degrees of freedom at each node. The beam element was programmed and added to the STAP code. This element required an additional 6 subroutines for its use. The addition of this element required 8 K-bytes of additional core memory. This translated into a decrease in problem size of 56 degrees of freedom. It is obvious from this fact that the microcomputer is not suitable for any kind of general purpose program under a sequential programming

format. The addition of only two or three more elements would require the entire core memory of the microcomputer.

The only way a general purpose format could be used would be to build a seperate program for each element. This would prevent the use of more that one element in a given problem, but would allow the use of more elements. This would utilize the disk storage available to microcomputer. The programs would be duplicates of each other, except for the element library. The user would then only have to use the appropriate program, instead of desiginating a specific element in a general purpose program.

There are other methods of increasing the ability of the microcomputer and make them more suitable to a sequentially formatted program. These methods require hardware and software purchases. The following three items have the greatest potential:

1. New operating software system called CROMEX
2. Math chip
3. Hard disk pack

The most promising of these is the CROMEX operating system. The present operating system requires 12 K-byte of core memory. This is almost 20% of the total capicity of the computer. For the STAP code, with a bandwidth of 6, this represents 84 degrees of freedom. The CROMEX operating system requires only 1 K-byte of core memory. This is accomplished by adding memory boards to the computer, and placing the remainder of the operating system on the new boards. The 1 K-byte is a communication link between the

main core memory and the additional memory boards. The total cost for this addition is approximately $2000. The CROMEX software purchase price is $500, while the additional memory boards cost $1500. This is only a 13% increase in the total computer package, based on a purchase price of $15,000. This is quite cost effective considering the 40% gain in computational ability.

The second hardware change is a math chip. This changes the floating point computations from a software function, to a hardware function. The only advantage of the chip is that computational time is lowered by a factor of 5-10. There is no improvement in the accuracy of the floating point computations. Although this is a significant time improvement, the total through put times for problems under consideration are not dominated by the computational time. The main driver is the input/output times. Decreasing solution time from 5 minutes to 3.5 minutes would not be significant change. In order for a math chip to be useful, the solution times would have to be dominated by the numerical computations.

The final hardware change is changing from a floppy disk storage to a hard disk . The hard disk increases storage capacity from 4.8 M-byte to approximately 20 M-byte. In addition, the input/output access time is reduced by a factor of 2-3. This would be a significant improvement, but the cost is somewhat prohibitive. The hard disk package for the Cromemco computer is $9,000. Decreasing solution time from 7.5 minutes to even 2 minutes would not make this very

cost effective. Purchase of a second computer might make better sense.

These hardware and software changes simply improve the capability of the microcomputer. Just the basic machine, however, did demonstrate significant ability in finite element solutions. The biggest disadvantage was the method of programming. Sequential programming requires to much core memory to completely utilize the microcomputer.

III. Overlays

Background

It is apparent from the previous chapter that minimum core requirements are a necessity for finite element programs utilized on microcomputers. Even the simple, single element finite element codes suitable for use on a microcomputer require extensive coding and data storage space. As a result, the technique of overlaying or segmentation is used. (6)

Program segmentation has been quite successful in reducing the amount of core memory required for various programs. The SAP program family, developed by K.J. Bathe, E.L. Wilson, and F.E. Peterson at the University of California is one example of successful overlay usage. (4) Programs using overlays have been implemented on mini-computer systems. The core memory restrictions that hindered the mini-computer usage are much like the core memory limitations that occur on microcomputers. The basic difference lies in the size of the memory and the size of the program involved.

Concept

Program overlaying is accomplished using an operating system with overlay capabilities. Idealy, only that portion of the program residing in core is the part being executed. This technique, when applied properly, greatly reduces the

amount of core memory needed for a particular program. This decrease in core requirements is the main attraction of the overlay technique. With a decrease in the core space required by the program, there is a corresponding increase in the core space that can be dedicated to data storage. This increase in the data storage allows for the solution of much larger problems.

The idea behind overlays is quite simple. Recall Fig. 2.2, in which the memory map for a sequential program was pictured. The entire program resides in core memory at the same time. For an average length program, 20-25 K-bytes of core memory may be required for the program. In Fig. 3.1 the overlay memory map is shown. The program is segmented into seperate blocks of subroutines. This division might

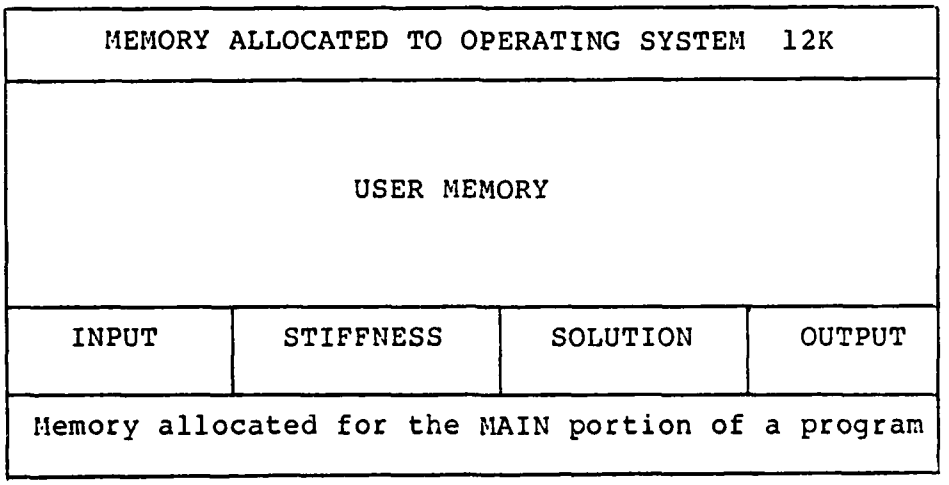


Figure 3.1
Overlay Memory Maps

result in only 10 K-bytes of program living in core memory at any given time. The seperate blocks of subroutines are the contents of an overlay. Only one of the blocks would be in core at any given time. The other three blocks would be stored on disk storage until they were needed for program execution. These are general numbers, with the specifics of the STAP code presented later in this chapter.

Although the advantage of using overlay techniques is a sizable one, there are drawbacks to using this method. The concept of segmentation, if improperly applied, can produce a poorly funtioning program. A poorly designed program will have excessive disk interchanges(I/O) as varying sections of the program are executed. Consideration must be given to both the progrm structure and the operating system software characteristics. As a result, it is usually much more difficult to build an efficient program using overlays.

Since both the data and program are primarily disk residents brought into core core as needed, a great deal of bookkeeping must be accomplished. This bookkeeping involves not only the data base, but the various portions of the code itself. There is also the problem that in many cases computer software is not designed to accept overlays. This forces the programmer to develop the overlay software compatible with the system in question. The computer skills necessary for building overlay software are much different than those used building programs.

This software difficulty is particularly true in the microcomputer field. Microcomputers are marketed with an

emphasis towards simplicity. The sales pitch is that anybody, with only basic computer skills, can own and operate a computer. As a result, the software for the more complex computer problemns has been slow in development. It has only been in the last 6 months that the software for overlay generation has been available for the Cromemco system.

Another significant disadvantage is that most programming is still done using a sequential format. It is only in the larger,more complex programs, that overlaying is used. Once again this presents real problems when using finite element codes on a microcomputer. The programs that are small enough to be adapted for microcomputer usage are not likely to be built using overlays. None of the educational sized codes that were examined for the sequential programming problem were built using overlays. In order to utilize the overlay technique, the microcomputer user is forced to build his own overlay code software, as opposed to using existing overlay generation capabilities.

Implementation

To illustrate this process, the STAP code was reformed into an overlay format. Traditionally, overlaying has always been considered more of an art than a science.(5) There is no single way to build a program with overlays and to implement this program. The best way to compare a segmented code to a sequential code is to compare the same codes in each format. Using the same program allows a

direct comparison in capabilites between the two forms of programming.

The procedure for breaking a code into an over-lay format is very straightforward. The first step is to determine the dynamic calling sequence of the program. This determines which subroutines are required for execution of a specific portion of the program. The next step is to look for obvious divisions of these subroutines into functions. These functional groupings are then sized into program segments of approximately equal size. These are the overlays that will form the program. A flow chart of STAP was built showing the subroutine calls. (Fig. 3.2) This flow chart indicated the divisions of the subroutines. The amount of core needed by each subroutine was determined during the compilation of the program as a whole, or during individual subroutine compilations.

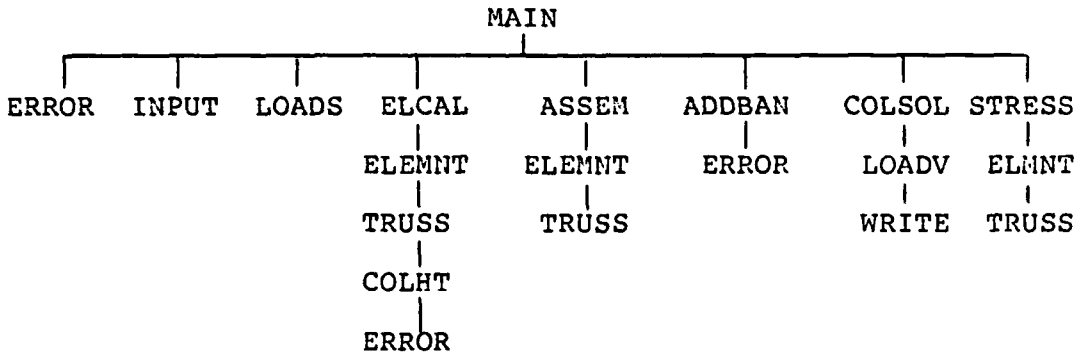


Figure 3.2
STAP Flow Chart

Once this division was accomplished, it was necessary to determine the length of the overlay. Breaking this program

into five segments produced the smallest overlay possible. The overlay size in this case was 8100 bytes.

The ELEMNT,TRUSS combination was used in each of the problem stages. In the second stage, formation of stiffness matrices, the ADDBAN subroutine was added to create the segment which set the size of the overlay. The actual overlay setup is shown in Fig. 3.3.

In this example, a previously developed, sequentially formatted program was changed to an overlay format. This meant that the debugging associated with construction of a new program was avoided. However, if a new program is being developed from scratch using an overlay format, previous studies indicate that the initial program debugging should occur in the sequential format, if possible.(6) This makes the process of producing an operable code much simpler. When the program is planned, the subroutines and programmming logic are designed with the overlays in mind. After the debugging is completed, and the program is operational, then break the program into overlays.

Certain techniques utilized for efficient sequential programming are not the most efficient for overlay programming. For example, in the STAP program, all code pertinant to the truss element was written into the subroutine TRUSS. The three functions of the TRUSS subroutine are nodal information, stiffness matrix generation, and the stress recovery functions. Each function required the same arrays for execution. By combining these functions into a single subroutine, the arrays were allocated only once.

		ADDRESS		
		CLEAR		
	ELCAL	ASSEM		
	ELEMNT	ELEMNT		STRESS
INPUT	TRUSS	TRUSS	COLSOL	ELEMNT
LOADS	ERROR	ERROR	LOADV	TRUSS
ERROR	COLHT	ADDBAN	WRITE	ERROR

MAIN PROGRAM

Figure 3.3
Overlay Breakdown

This saved a significant amount of core memory required for the program. For an overlay format, however, this created much larger subroutines than were necessary.

Since TRUSS had three distinct functions, it was a long and complex subroutine. As a result, in order to accomodate the subroutine, the overlay was quite large. TRUSS was the largest subroutine in the program. By breaking TRUSS into 3 seperate pieces, the size of the overlay was dramatically reduced. The memory map (Fig. 3.1) would be altered only slightly. Instead of the TRUSS being listed three times, there would be a TRUSSI, TRUSSII, and a TRUSSIII. Because, the functions of TRUSS were totally independent, there would be no degredation of program response. By making these smaller subroutines, the overlay size was reduced to approximately 6700 bytes. This is a reduction in overlay size by 15%, and represents approximately 2.2% of the total core memory of the computer. Assuming bandwidths of 5-20, this represents an increase of 10 degrees of freedom to the

size of problem that can be solved. Problems with larger bandwidths will have a decrease in the number of degrees of freedom possible. Using the simple truss program for comparison, the largest problem which can be handled using an overlay format is in the range of 300 degrees of freedom. This is entirely the result of the increase in user memory made available by the overlays.

When the same procedure was followed for the more complex beam element, the savings were even larger. The program, containing the beam element and the truss element, fitted into an overlay format, required an overlay size of 12000 bytes. Splitting the beam element into seperate subroutines, using the same technique as in the truss element, reduced the overlay to 9600 bytes. This represents an increase of 17 degrees of freedom when compared to a program utilizing a beam element combining the three functions.

The penalty for the increased problem solving capability of overlays is an increased total solution time. In the sequential format, for the problem sizes under consideration, doubling of the problem size produced approximately a doubling of solution time. Using overlays to double the problem size that can be handled changes this ratio. Because of the increased disc I/O, the solution time is expected to increase by a factor of 3-4. The total solution time will be in the range of 15-20 minutes. This, however, is still an acceptable figure for total solution time.

User available memory can also be increased by using disk storage instead of memory storage at various stages in a program. To illistrate this, the TRUSS subroutine was rewritten. The TRUSS subroutine generates an element stiffness matrix and stores this matrix in memory in blank common. This requires a relatively small amount of storage. Instead of storing this information in blank common, the element stiffness matrix is stored on a sequential disk file. Once all the element stiffness matrices are formed, they are read from the disk file in the ADDBAN subroutine. For this program, using the truss element, the memory space saved is insignificant. At most, it might add 1 or 2 degrees of freedom.

This program change, however, demonstrates the problems which arise when I/O operations are not considered. Because of the increased I/O operations, the solution should be longer than the original overlayed program, but the problem solving capability would not be significantly enhanced. This situation is noted to point out the fact that not all disk storage is acceptable. Increased problem solving capability must be weighed against total solution time. For this element, the space savings do not justify the time increase. As the size and capabilities of the microcomputer are expanded, however, the core memory increase might offset the penalty of increased solution time.

Evaluation

In order to properly evaluate the overlay capability of

the microcomputer, the overlay formatted STAP had to be executed by the microcomputer. Because the overlay software for the microcomputer was not installed on this system, an attempt was made to construct an overlay generator. A basic form of the software was implemented on the system, but it was unable to handle the STAP overlays. Since skills involved in construction of an overlay generator lie in the area of computer science and not engineering, further attempts were not made to improve the overlay generator. When the software becomes available, the STAP program in overlay format can be utilized.

In the sequential programming chapter, the observation was made that a general purpose program would not be feasible, except by using seperate programs. With the use of overlays, however, this is not exactly true. It would be possible to have several elements in the finite element code, with each element living in its seperate overlay. In order to accomplish this, the programmer would have to insure that the common areas of the program are not being moved during the creation of the overlays. They would have to be located at specific points in core memory. This is not a serious difficulty.

The only difficulty with the general purpose structure is in the varying overlay sizes. Referring to the beam and truss elements in STAP, the overlay size is significantly changed. In order to use this technique to its maximum capability, the user/programmer would have to alter the overlay size, based on which element was being utilized. If

multiple elements were wanted for a single problem, the overlay size would be based on the largest overlay required for a particular element. This technique would create some unused memory space, but the convenience of the program might out-weigh the small loss in problem size. Only one program would have to be kept on hand, not the several that sequential programming required. In this way, the real world applications that require multiple element modeling could be attempted on the microcomputer.Since mixed element soltuions are usually necessary for good modeling, this capability is highly desired.

In the chapter on sequential programming, several modifications to the Cromemco microcomputer were examined. When these same improvements applied in conjunction with overlays, their potential is even more impressive.

The combination of the CROMEX software package with an overlay format holds a great deal of promise. Using the STAP code, with only the simple truss element as an example, the total memory dedicated to program and system functions is only 9.1 K-bytes. This leaves a total of 54.9 K-bytes of core memory available for data storage. This is a increase of 89% in the size of problem that the microcomputer can handle, compared to the sequentially programmed STAP. With this arrangement, the total number of degrees of freedom that could be handled is in the range of 400. This is a significant capability for the use of finite element techniques in problem solving.

With the addition of the CROMEX, the problem size could

increase enough to make a math chip a useful purchase. As problem size increases, the percentage of total solution time attributed to equation solving increases. Since a math chip only decreases the amount of time in the computational processes, the longer this time becomes, the more advantages a math chip presents. The exact determination can only be made for a specific computer system. For the present, there is not enough information to make a decision on the purchase of a math chip for the Cromemco microcomputer system.

The appeal of the hard disk package is not really affected by the use of overlays. The only real advantage of the hard disk still lies in the input/output access times. Unless there is a tremendous amount of I/O activity, there is no real need for a hard disk for a microcomputer. Its advantages do not outweigh its cost. As mentioned earlier, the software for the Cromemco system has been updated to include an overlay linker. Considering the difficulties of constructing overlay software, it is recommended that the software package be purchased. Even though the overlay STAP program was a very simple code, the construction of an overlay generator is beyond the capabilities of most engineers. It is a computer science project, requiring extensive computer science skills.

IV. MODULARIZATION

Background

Modularization can be thought of as an extension of overlays. A modularized program is a series of programs operating sequentially. Each seperate program produces output which is read into a common data base located in external file storage. Each succeeding program obtains the information required for execution from the data base, and continues with its portion of the solution process. This technique is another method used in increasing the amount of memory available to the user by decreasing the memory dedicated to program storage. For purposes of clarity, the original program will be called the program, while each of the smaller programs will be referred to as subprograms.

Modularizaton has been used successfully at the main frame and minicomputer level. Two of the most famous of these codes are GIFTS and SPAR. SPAR was created for NASA by Dr. W.D. Whetson (5), while GIFTS was created at the University of Arizona by Dr. H.A. Kamel.(6) Both of these codes are general purpose finite element programs of tremendous size.

Concept

A modulized program has significant advantages over other programming formats. The first was already mentioned, a smaller core memory requirement for the program. Because the entire "main" portion of the program is not always in

core, the core size dedicated to the program should be slightly less than overlays. Also, depending on the exact nature of the program the original input can be relatively small. A automatic node and element generator can be the first subprogram. Its output would be the nodal geometry and form the base of a unified data file.

The unified data base allows the program to be halted at any stage, and the results of that subprogram checked for accuracy. For example, after the node and element generation subprogram is completed, a plot could be executed to check the input. If the structure is visibly acceptable, then the next subprogram is called and the solution is continued. This procedure is possible at any stage in the program. Figure 4.1 shows a schematic of how a modularized code might look.

In addition to requiring slightly less core memory than overlays, modules are much simpler to construct. Each of the

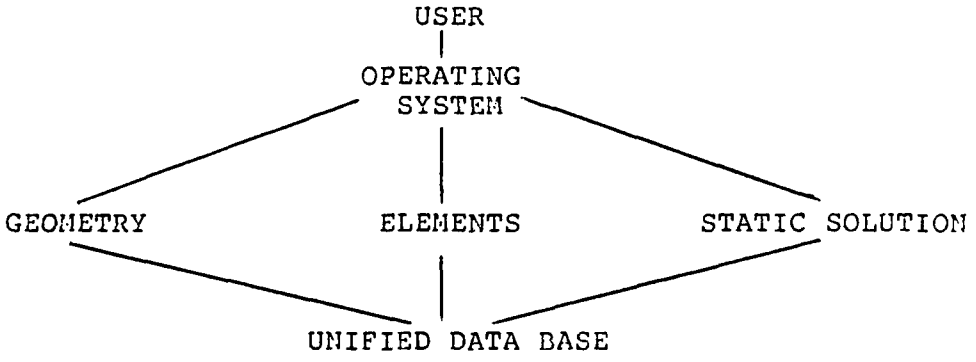


Figure 4.1
Finite Element Modular Architecture

subprograms is a seperate entity and is normally written in a sequential format. There is no need for sophisticated software to implement the program. Seperate overlay generators are not necessary. Normal programming skills are sufficient to use this programming format. The only real difference in this format is the creation of the data base and the bookkeeping required to insert and retrieve data.

There are disadvantages to modularized programs, however. Since there is no link between the subprograms, all data must be a resident on disk or tape files. This requires a large amount of I/O transfer during the execution of the program. As a result, the total solution time will be increased. The usefulness of modularized finite element programs will be determined by how much the through put time increases in relation to the increased problem solving capability.

Implementation

In order to illustrate the concept of modular programming, the STAP program was divided into a modular format. For ease of construction, the modules or sub-programs, were constructed with the same functional grouping as the overlay format. Because the subprograms were already identified, the breaking of the program into module form was not difficult. Figure 4.2 is a schematic of the modularized STAP.

There were no unusual difficulties in implementation. Since each of the sub-programs was a sequential format the

normal computer software was sufficient. The only difference was in running a series of programs as opposed to a single program.

Evaluation

When the modularized format of the Stap code was used in problem solving, its performance was less than expected. This format produced an increase in solution capability by only 23% as compared to a sequential format. This increase

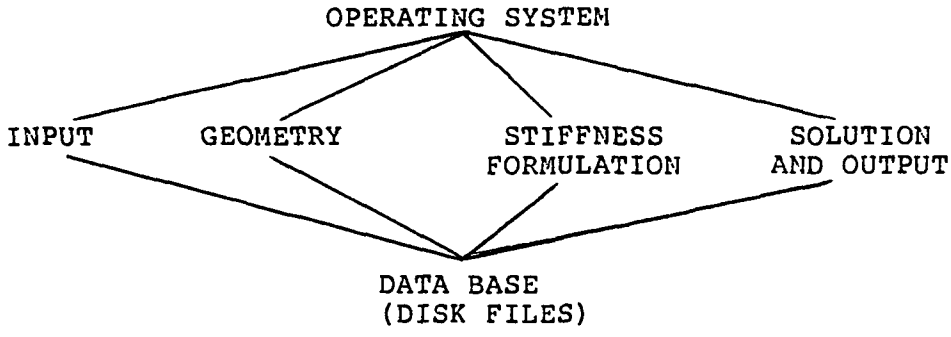


Fig 4.2
Modualrized STAP Program

was less than the increase expected of the overlay format. The maximum problem size that the modularized STAP program could solve was 270 degrees of freedom with a bandwidth of 15. The total solution time for this problem was 57 minutes. Table 4.1 give a summary of the capabilities of the STAP program in modular format.

Although this represents a considerable problem solving capability, the degradation in total solution time is a

significant problem. For a problem on the order of 100 degrees of freedom, easily solvable with a sequential format, throughput time was on the order of 45 minutes. This does not compare favorably with the sequential format. This time penalty was consistent throughout the entire range of problem capability.

This increase in execution time is entirely a result of the tremendous I/O of the modularized format. In the STAP

Node points	DOF	Total Solution Time (Min)
8	10	3.9
30	54	16.4
63	112	36.8
102	180	44.3
141	270	57.2

Table 4.1
Timing Data-Modularized STAP

example, there is a data write and a data read to disk in each program segment. In each case the majority of the data base was altered by the subprogram execution. This fact, coupled with the fact that the most inefficient part of the microcomputer using disk storage is the I/O transfer,lead to the large solution times noted. These solutions times were approximately 10 times greater than the solution times for a comparable problem solved using a sequentially formatted program.

The Modularized STAP program was also very dependent on

the number of elements in the structure, as well the degrees of freedom in the problem. The figures in Table 4.1 were generated with the structure pictured in Figure 2.1 being used as a template. Multiple horizontal rows were created to increase problem bandwidth to 15. This type structure has a large number of elements in relation to the number of node points.

A second structure was constructed, leaving out alternating cross pieces and run on the program.(Fig 4.3) This run was made with 270 degrees of freedom in the problem, but the number of elements was reduced from 425 to 333. This reduction improved the solution time to only 52.4 minutes. The amount of data that had to be transferred was reduced by the reduction in elements, resulting in the improved performance of the program. In this example, only the truss program was used for the data generation. Adding the more complex beam to the process would not change the relative performance of the program. In addition, the truss element was used in the split form described earlier.

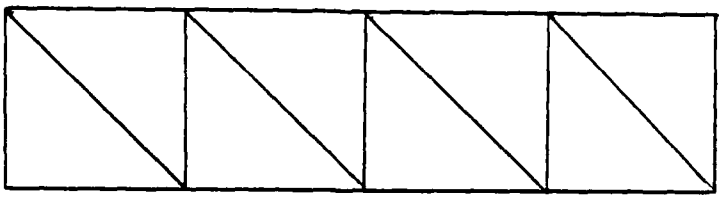


Figure 4.3
Simplified Truss Structure

The module concept has great potential as a general purpose program on microcomputers. Since each subprogram is a seperate function, the problem of core availability does not exist. Likewise, the problem of storage in relation to uneven overlays is absent. Since the data base lives on disk file, there is no need to keep track of loading points. The data can be called from disk during runtime operation of the program. The only consideration is to keep track of the data on the disk files, and the order it was stored.

The modifications to the Cromemco system also look promising in conjunction with modulized programs. The decrease in memory dedicated to operating system will improve the problem solving capability of a modularized formatted program much in the same way that the overlay program was aided and the improvements should be comparable. In order to fully utilize this gain in problem solving capability, some method of decreasing the time for I/O transfer must be developed.

In the modular concept, as in the other programming methods, the math chip would probably not produce significant performance increases. The computation time does not really change in the modular format. The total computational time does not justify its use.

The hard disk, used in conjunction with a modular program could be the answer to the disk I/O problems. The main advantage of the hard disk was in the decreased I/O access time between computer and hard disk. A decrease in this segment of execution time by a factor of 2-3, would

bring the solution times for a molular program back in line with the other forms of programming. Once again, the cost of the hard disk is the determining factor. Significant improvements in disk I/O access times may offset the cost of the disk pack. Again, consideration to a second computer must be given due to the tremendous cost of a hard disk pack.

V. Basic Skills

One of the objectives of this study was to determine the necessary user skill level needed to effectively work on a microcomputer system. The question was whether an engineer with basic computer programming skills could use a a micro-computer in conjuncton with finite elements, or wheter it takes a computer programmer with some engineering skills.

The author's experience in computers at the beginnning of this project was almost non-existant. The only experience in computers and programming was a single course in FORTRAN programming. It consisted of learning how to read and understand the FORTRAN language, and the basics of structured programming. The experience in microcomputers was that of seeing them demonstrated in laboratories or in word processing systems.

With this as a starting point, finite element codes were implemented on a microcomputer. The lack of experience did create many of the problems noted in other chapters, but the published materials on FORTRAN, programming, and micro-computers were usually sufficient in solving the problems.

It is necessary to point out that the required course work at the Master's level in engineering is not sufficient to set up and operate a microcomputer. It takes help from published materials and people with experience in the field of microcomputers, to accomplish this task. The published

material is readily available and with the increased poplarity of these small computers, there is usually one person in a section or laboratory with this type of experience.

An individual's lack of experience should not be a determining factor in microcomputer usage. This experience factor might determine the length of time required to adapt or create the first programs, but it can be overcome. In only a very short period of time, an engineer can be operating a microcomputer system with positive results.

There are several terms and tasks which must be understood when using a microcomputer as opposed to a central site machine. Computer work done on the central site usually consists of data file creation, source editing, and compiling the edited program. These tasks are taught in basic computer programming and are basic to sequential formatted programming. The engineer can operate finite element programs with these skills.

Use of the microcomputer will require the engineer to be more knowledgable of programming techniques. For example, in Chapter 2, an attempt was made to alter the method of storage in memory. This involves manipulating the linker. This would rarely, if ever, be done on a central site computer. The procedure was not difficult, and the Cromemco literature gave all the details necessary.

In Chapter 3, the concept of overlays was dealt with. With a central site computer, this type of program would be automatically handled, possibly without the user realizing

that overlays were in use. This would not be possible on a microcomputer. The user will know if the program he builds or adapts to a microcomputer system uses overlays. He will have to come up with an overlay generator as well. This software could be purchased of constructed, but it is the user's responsibility. This same concept extends into the modular concept. It takes a higher level of skill than central site operation. By using the microcomputer, these skills are developed. The basic sequentially formatted program can be used with skill level that most engineers would have. To fully utilize the the microcomputer, different skills will have to be developed.

VI. Implementation Problems

The problems of implementing a finite element program on a microcomputer are two fold. The first deals with the experience level of the user in computer programming. This refers to both the general concepts of programming and the specifics of the language the program is written in. The second is related to hardware aspects of a microcomputer. The typical user of a microcomputer is usually the prime technician for the system. There is no external expert support except that which is arranged by the user. These two considerations determine the ease in which a computer program is implemented on a microcomputer system.

There are certain basic steps which can be taken that will enhance the user's ability to apply a program to a microcomputer. The first is to insure that the program is tailored to fit the core memory of the machine. For example, the original STAP code was built with a blank common size of 100,000. Obviously, this would not fit on a 64 K-byte microcomputer. Related to this is the size of the arrays which are established in the program. These changes are easily accomplished, requiring only the reading of the program for proper sizing.

In addition to the obvious sizing problems, the user must also know how the computer conducts its storage. Most main frame computers use the same number of bytes for both integer and real value storage. However, this is not always

true for micro-computers. In the Cromemco system, integer values use 2 bytes for storage, while reals require 4 bytes. This change can alter the manner in which a program must recall data from storage. The STAP program used a real array for blank common to store both real values and integer values. This poses no problem when they have the same byte length. However, when the Cromemco system used the same real array to store integer values, they were stored as two bytes, while the reals were stored as four bytes. The Stap code used algorithims to determine the starting and ending points for the array storage based on 4 bytes for both integer and real. When these values were calculated, they pointed to the wrong location for a mixed byte pattern. Data was being overwritten and misplaced by the computer.

This problem was easily corrected by using integer arrays and modifying the pointer algorithims. By knowing this fact before hand, a great deal of time in debugging can be eliminated during implementation of a program.

When dealing with a microcomputer there are certain limitations which must be recognized. These are parts of the operating system of the computer. Because the overall system is small, the software package must also be limited. Run-time debugging is not generally available. As the errors in the FORTRAN code are found and corrected, the only way to update the code was to compile the entire program. Even for the relatively short STAP program, the compilation time was a limiting factor in the debugging process. Compilation time was approximately 3 minutes. This can greatly increase

the amount of time to debug a program. Obviously, the more complex the program, the longer the debugging process.

Access to common areas is another feature lacking in the Cromemco microcomputer. In order to access common, specific debugging routines must be developed. Figure 6.1 at the end of this chatper list subroutines which can be used to accss arrays carrying real and integer values. The PRINTN subroutine was used to determing the pointers for array storage, while the PRINIT and WRITIT subroutines displayed the contents of the blank common. These routines are based on a 2 byte integer, 4 byte real storage scheme.

The basic idea is that when using a microcomputer, the user can not just blindly put a code on the system. The process will be more complex than for the central site, and there will be no technicians to solve problems. The user must be more familiar with the codes involved. The increased time for implementing and debugging must be expected and accepted.

```
      SUBROUTINE PRINTN

      COMMON/DIM/N(15)
      CIMMON/TAPES/IELMNT,ILOAD,IIN,IOUT
      WRITE (IOUT,101)
101   FORMAT (////35H  THESE ARE THE N(I) PARAMETERS  ///)
      DO 100 I=1,15
      WRITE (IOUT,200) I,N(I)
200   FORMAT (2I10)
100   CONTINUE
      RETURN
      END




      SUBROUTINE PRINIT (LEN)

      DIMENSION A(1)
      COMMON/DIM/N(15)
      COMMON/TAPES/IELMNT,ILOAD,IIN,IOUT
      COMMON IA(1)
      EQUIVALENCE (IA(1),A(1))
      II=LEN-1
      DO 100  I=1,LEN
      JJ=N(I+1)-1
      DO 100 J=JJ,JK
      CALL WRITIT (J,IA(J))
100   CONTINUE
      RETURN
      END


      SUBROUTINE WRITIT(J,IA)
C     ....THIS SUBROUTINE PRINTS A REAL AND INTEGER INTERPRETATION OF
C     THE MEMORY STARTING AT .IA......
      DIMENSION IA(2), IT(2)
      COMMON/TAPES/IELMNT,ILOAD,IIN,IOUT
      EQUIVALENCE (A,IT(1))
      IT(1)=IA(1)
      IT(2)=IA(2)
      WRITE (IOUT,100) J,A,IA(1)
100   FORMAT (1H,I10,2X,E13.6,2X,I10)
      RETURN
      END
```

Figure 6.1
General Debugging Subroutines

VII. Conclusions

If the programs are properly organized, the microcomputer is capable of handling problems of significant size with finite element techniques. Programs written in the simplest programming format, sequential, handle problems with 200 degrees of freedom with total solution times of less than five minutes. Although this could not solve the structural analysis of an entire aircract or large structure, 200 degrees of freedom can solve problems of reasonable magnitude.

Using sequential formatting and the resulting 200 degrees of freedom, the only problem that is reasonable would be a two dimensional static analysis. The use of the more complex plate elements and quadrilateral elements would probably not be possible. These elements would decrease the problem size capability due to their complexity.

With the adaption of overlays and modualar programming, and the additional purchases of hardware and software, problem sizes in the region of 400 degrees of freedom are solvable. Although solutions times have increased somewhat, the times are not excessive. Four hundred degrees of freedom can handle many significant problems in static structural analysis. At this level the more complex elements would be possible.

In both the above situations, however, the bandwidth of potential problems must be considered carefully. Poor

problem setup could increase bandwidths to such a degree that the problem solving capability would be greatly decreased. Not only is the bandwidth a computer problem, it is also a structural problem. The small computer can handle structures which are long in one axis and narrow in a second axis much more efficiently that structures which are more square in nature.

In addition to the abilities of the microcomputer to handle finite element codes, there is a second major benefit resulting from these procedures. Finite element program development would be enhanced by microcomputer usage. The trend in modern programming is in structured programming. This type of programming allows for the construction of blocks of the same program to be developed independently. The only requirement is that the programmer knows what is coming into, and what is required to leave his segment of the program. This type of program development is perfect for the microcomputer.

Even if the total program is too complex to fit on a microcomputer, the various segments may not be. These segments could be constructed on the smaller computer, leaving the central site computers free for combining the segments into an integrated program. This usage would serve two important functions.

The first is that congestion at the central site would be reduced. As most engineers know, one of the biggest problems with computer usage is a lack of available computer time. If the number of jobs on the central site main frame

computer was reduced, then computer resources would become more plentiful. The cost of using these microcomputer systems would be limited to how much time is necessary to set up the software to handle FORTRAN programs. Since the microcomputer systems in use today are typically under utilized in their secretarial role, there would be no additional hardware costs. It is even possible to be in a situation where the purchase of a microcomputer system would be justified soley on the basis of improving central site operation. This type of assessment could only be made for particular operations.

Next, is the fact that the program development could be more efficient. When the programmer is working under an obvious size constraint, he might be inclined to look for a more efficient manner in program construction. The problem of core memory restrictions would be more obvious on microcomputers than on main frame computers. This may not always occur, but it is a distinct possibility.

The final area that the microcomputer can be utilized in is in educational format. This could either be formal in nature, or on the job training. Finite element techniques are necessary in many fields. By using microcomputer systems as educational tools, the same advantages of reduced central site congestion occur.

VIII. Recommendations

Further work needs to be accomplished in this area. The first step should be to acquire the CROMEX operating system and the software for overlay generation. With these additions the computer has the capability for significant problem solving.

With or without these additions other problems need to be addressed. The first would be in exploring the different types of solvers in use today. A more efficient solution technique might add to the capability of the computer. These solvers could be both in core or out of core solvers. A second aspect that could be examined is the use of an iterative technique in the solution process. This approach usually requires fewer degrees of freedom than a direct approach. This approach would avoid the problem of disk I/O transfer by working with smaller problem sizes in core.

At the present time, the best course to pursue is in the overlay format. This technique shows great promise in total problem size, without the timing disadvantages that the modular format has.

Bibliography

1. Beaufait, Fred W. Computer Methods of Structural Analysis. Englewood Cliffs, New Jersey. Prentice-Hall,Inc. 1970.

2. Zienkiewicz, O.C. The Finite Element Method. Berkshire, England. McGraw-Hill Book Company(UK) Limited. 1977.

3. Bathe, Klaus-Jurgen and Edward L. Wilson. Numerical Methods in Finite Element Analysis. Englewood Cliffs, New Jersey. Prentice-Hall,Inc. 1976.

4. Wilson, Edward L. "SAP-A Structural Analysis Program," Report UC SESM 70-20, Department of Civil Engineering, University co California, Berkely. 1970.

5. Foster, Edwin P. and Olaf O. Storaasli. "Structural Analysis on a Minicomputer," (Symposuim paper at First International Conference at Southampton University. September 1979.)

6. Kamel, H.A., M.W. McCabe, and P.G. DeShazo. "Optimun Design of Finite Element Software Subject to Core Restrictions." Computers and Structures,(Vol 10 June, 1978).

Vita

Robert J. Hampson was born 20 September, 1950 in Winner, South Dakota. He graduated from William Mitchell High School in Colorado Springs, Colorado in 1968. He attended the Air Force Academy and graduated in 1972 with a Bachelors in Engineering Mechanics.

He was assigned to Laughlin AFB, Del Rio, Texas, for flight training. He graduated in 1973 and assigned as a T-38 instructor pilot at Laughlin. In 1977 he was transferred to Charleston, AFB, Charleston, South Carolina, to fly C-141 aircraft. In 1980 he entered the Air Force Institute of Technology in the graduate Aeronautical Engineering Program.

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GAE/AA/81D-13	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* THE APPLICATION OF FINITE ELEMENT SOLUTION TECHNIQUES IN STRUCTURAL ANALYSIS OF A MICROCOMPUTER		5. TYPE OF REPORT & PERIOD COVERED MS Thesis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Robert J. Hampson Capt USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE December 1981 13. NUMBER OF PAGES 50
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

28 JAN 1982

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17

FREDERIC C. LYNCH, Major, USAF
Director of Public Affairs

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*
Finite Elements
Microcomputer

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The feasibility of applying finite element solution techniques on a microcomputer was explored. The finite element solution techniques were applied in structural analysis.
Three programming formats in finite element programs were explored. These were sequential, overlays, and modular formats. The capabilities of the microcomputer with respect to these formats were explored.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*


END
DATE
FILMED
3-82
DTIC