CONF-790902--3

IMPACT OF ADVANCED SYSTEMS ON LMFBR ACCIDENT ANALYSIS CODE DEVELOPMENT

by

F. E. Dunn and J. M. Kyser

Prepared for

Scientific Computer Information Exchange

Livermore, California

September 12-13, 1979

U of C-AUA-USDOE

ARGONNE NATIONAL LABORATORY, ARGONNE, ILLINOIS

Operated under Contract W-31-109-Eng-38 for the

U. S. DEPARTMENT OF ENERGY

IMPACT OF ADVANCED SYSTEMS ON LMFBR ACCIDENT ANALYSIS CODE DEVELOPMENT

F. E. Dunn and J. M. Kyser
Reactor Analysis and Safety Division
ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, IL 60439 U.S.A.

ABSTRACT

In order to investigate the ability of an advanced computer, using currently available software, to handle large LMFBR accident analysis codes, the SAS3D code has been run on the NCAR CRAY-1. SAS3D is a large code (56,000 Fortran cards) using many different physical models and numerical algorithms, no one of which dominates the computing time. Even though SAS3D was developed on IBM computers, remarkably little effort was required to run it on the CRAY-1. Making limited use of the CRAY-1 vector capabilities, it runs a factor of 2.5 to 4 times faster on the NCAR CRAY-1 than on the ANL IBM 370-195. With minor modifications, an additional 20-30% speed improvement on the CRAY-1 is achieved. In the current process of completely re-writing SAS3D to make SAS4A, much of the coding is being vectorized for the CRAY-1 without sacrificing IBM, CDC 7600, or UNIVAC performance and portability. An initial SAS4A test case runs a factor of 7.1 faster on the CRAY-1 than on the IBM 370-195. On either computer, this SAS4A case runs appreciably faster than a corresponding SAS3D case, indicating that there can be significant benefits from using vectorizable coding, even on a non-vector computer. It appears that even though the one-dimensional models in SAS3D strain the capacity of ANL's current computers, an advanced computer such as a CRAY-1 would make it feasible to replace many 1-D models with 2-D or 3-D models.

INTRODUCTION

The SAS series of computer codes[1, 2,3,4] are used to analyze hypothetical accidents in Liquid Metal Cooled Fast Breeder Reactors (LMFBRs), as well as Gas Cooled Fast Reactors (GCFRs). All of the existing codes in the SAS series, except for the original SAS1A, have been capable of representing a reactor core with a quasi-three-dimensional treatment which uses coupled one-dimensional models to approximate the real three-dimensional system. The current representation is adequate for many cases; but for some phenomena, local two-dimensional or three-dimensional effects occur which can not be handled adequately with the current SAS one dimensional models. More detailed two- or three-dimensional models would probably require significantly more computer time than the current models. Even with the current models, a detailed whole core analysis with the SAS3D code can strain the capacity of the current IBM 370-195 and IBM 3033 computers at Argonne National Laboratory (ANL); indicating that it may be desirable or necessary to consider the use of more advanced computers if more detailed models are needed. Therefore, an effort was undertaken to address two main questions. First, how much effort would be required to get the SAS3D code, which is the current production version in the SAS series, to run on an advanced computer? Would extensive re-writing of the code be necessary? Second, since the next SAS code, SAS4A, is currently being written from scratch, as opposed to adding new modules onto SAS3D, can SAS algorithms and coding be modified so as to vectorize on a vector machine without sacrificing IBM, CDC 7600, or UNIVAC performance or portability? Under a grant from the National Center for Atmospheric Research, computer time on the NCAR CRAY-1 computer was available for this project.

SAS CODES

GENERAL DESCRIPTION

The SAS codes have been developed

to analyze the initiating phases of hypothetical accidents in LMFBRs or GCFRs. SAS3D starts with steady-state calculations to determine the initial conditions in the reactor, usually normal operating conditions. Then transient accident calculations are made for a user-specified event, such as loss of power to the primary coolant pumps, or insertion of reactivity at a user specified rate.

In an LMFBR or a GCFR, the reactor core contains long, narrow fuel pins, which are steel tubes (cladding) containing fuel pellets plus a gas plenum above or below the fuel to hold the gaseous fission products released during irradiation of the fuel. The fuel pins are arranged in hexagonal arrays within fuel subassemblies, with coolant flowing in the axial direction between the pins. The subassemblies have steel, hexagonal shaped outer duct walls. The ducts are somewhat longer than the fuel pins to allow room above and below the pins for flow orifices and instrumentation. Typically, there are 217 fuel pins, each 1/4 inch in diameter and about 8 feet long, in a fuel subassembly which is about 12 feet long. There are between 75 and a few hundred subassemblies in a reactor core.

The geometry used by SAS3D to represent the reactor core consists of a number of "channels", where each channel represents a fuel pin and its associated coolant. Usually a SAS channel is used to represent a subassembly or a group of similar subassemblies. In this case, the fuel pin represents an "average" pin in the subassembly. Coolant flow in a SAS channel is only in the axial direction, and heat flow in a pin is calculated only in the radial direction. Because of the very large length-to-diameter ratio of the fuel pins, axial heat conduction within the fuel pin or the coolant is negligible. The coolant removes heat by convection.

Each SAS channel is divided into a number of axial nodes. Typically about 20 axial nodes are used to represent the fueled part of the pin, and another 10-15 nodes are used to represent the rest of the subassembly. At each axial node in the fuel section, about 10 radial nodes are used in the fuel and 3 radial nodes are used in the clad. One radial node is used for the coolant, and one or two radial nodes are used for the "structure", which represents both the subassembly duct wall and the wrapper wires or grid spacers which keep the pins in position. Above and below the fueled section, only a few radial nodes are used.

With this channel treatment, SAS3D calculates steady-state and transient temperatures in the fuel, clad, coolant, and structure. Sodium boiling; clad melting, relocation, and freezing; fuel melting, relocation, and freezing; and interactions between molten fuel and liquid coolant are calculated. Also, the stresses and strains in fuel pins prior to pin failure, the amount of fission product gas in the fuel and its contribution to fuel relocation after pin failure, and the pressures and flow rates of the coolant in the core and around the primary coolant loop are calculated.

ADVANTAGES AND LIMITATIONS OF THE SAS CHANNEL REPRESENTATION

One big advantage of the SAS channel representation is that it provides a detailed, three-dimensional representation of the reactor core even though only one-dimensional equations are solved; and the solution of one-dimensional equations is usually much faster than the solution of multi-dimensional equations. Detailed radial and axial temperature profiles in the fuel pin are obtained by solving a one dimensional radial heat transfer equation at each axial node. Coolant temperatures and flows are obtained by solving one dimensional equations for the axial direction. Fuel and cladding relocation are calculated in the axial direction. Different subassemblies or groups of subassemblies can be represented by different SAS channels, providing detailed axial descriptions at a number of radial and azimuthal locations in the core.

SAS3D only accounts for limited coupling between SAS channels, but this corresponds to the limited interaction between subassemblies in a reactor. The duct walls prevent coolant flow between subassemblies, except that the subassemblies all receive their coolant from a common inlet plenum and all discharge their coolant into a common outlet plenum. The common inlet and outlet plenums are accounted for in SAS3D. The different parts of the core are coupled neutronically, and this neutronic coupling is accounted for in the SAS3D neutronics calculations. There is some heat flow between the duct walls of adjacent subassemblies, whereas SAS3D uses an adiabatic boundary at the outside of the SAS "structure". Accounting for heat flow between subassemblies by computing heat transfer between the structures in different SAS channels could be done in a fairly straight-forward manner without changing the basic SAS channel representation.

The main limitation of the SAS channel representation is that it does not account for any differences between fuel pins or coolant sub-channels within a subassembly. In reality, there are often power skews across a subassembly, the coolant sub-channels next to the duct walls are somewhat larger than those in the interior of the subassembly, and the duct walls have heat capacity and tend to act as limited heat sinks during a transient. The net result is that the interior of a subassembly is usually hotter than the edge, and sometimes one side is hotter than another.

The use of an "average pin" representation in SAS3D tends to average-out many of the variations within a subassembly and often gives good results. For instance, predictions of the boiling model in SAS3D usually agree reasonably well with the results of multiple-pin boiling tests.[5,6] On the other hand, it is often necessary to account for radial incoherence within a pin bundle to obtain satisfactory agreement with clad melting and re-location experiments.[7] A really adequate treatment of fuel relocation probably also requires accounting for radial incoherence within a subassembly.

POSSIBLE IMPROVEMENTS AND COMPUTER LIMITATIONS

Because of the above mentioned limitations of the current SAS3D "single pin" representation of a subassembly, it would be desirable to have a "multiple pin" representation in which a subassembly is represented by a number of pins or pin groups with connected coolant channels.

One problem with developing multiple pin models for whole-core accident analysis is that even the one-dimensional single pin models in SAS3D strain the capacity of the current ANL computers. A 33 channel SAS3D case can take 6 hours of computer time on the IBM 370-195 or IBM 3033. It also requires about 3 megabytes of memory on an IBM computer. Even a 1 channel case requires about 800 kilobytes of memory. On a CDC 7600 the same 33 channel case would require about 3 hours and almost 400,000 words of LCM storage. These computer times are almost entirely CPU times, since SAS3D does relatively little I/O in a big run. If multiple-pin models were used, the running times would probably increase at least linearly with the number of pins used per subassembly, and the increase may be proportional to the square of the number of pins.

Some computer codes already exist for treating some aspects of intra-subassembly incoherence. Because of computer limitations, these codes are usually limited to treating single subassemblies instead of whole cores, and they are limited in the phenomena that they treat. For instance, both the COBRA-3 code[8,9] and the COMMIX-1 code[10] can compute detailed coolant temperature distributions within a subassembly. Either steady-state or pre-voiding transient calculations can be made by these codes. On the IBM 370-195 it is estimated that COMMIX-1 would require about 2.5 hours to compute the steady-state temperatures in all of the coolant sub-channels of a 217 pin subassembly. COBRA-3 can take 20 minutes to calculate steady-state coolant temperatures for 12 coolant sub-channels.

Because of the above mentioned computer limitations, it is unlikely that many pin (217 pin) models will be used in whole-core LMFBR accident analysis codes in the near future, even if significantly faster computers are used. On the other hand, a reasonable increase in computer speed would make it feasible to use "few pin" models for whole-core analysis. A well developed model using 2-5 pin groups to represent a subassembly would probably be quite adequate for most purposes.

NUMERICAL ALGORITHMS AND CODING ASPECTS

SAS3D is a relatively large code, and SAS4A will be larger. The source deck for SAS3D contains about 56,000 FORTRAN cards. The main reason that SAS3D is large is that it contains a number of separate, but coupled, modules for computing different aspects of an accident: heat transfer, coolant flow, fuel pin mechanics, sodium boiling, clad relocation, fuel relocation, fuel-coolant interactions, and neutronics.

The equations solved by these modules are all different, but there are some aspects that are common to most of the modules. Finite differencing in both space and time is used. A number of discrete nodes are used to obtain spatial variations, and the transient time behavior is obtained using discrete time steps. Typically, the algorithms are set up to determine the conditions at each node at the end of a time step, starting from known conditions at the beginning of a time step. In general, the equations solved are non-linear, although they are linearized across a time step. Semi-implicit or fully implicit schemes are often used, leading to the simultaneous solution of linear equations with coefficients that are re-calculated each time step. The resulting matrices are banded, often tri-diagonal. The calculation of the coefficients usually takes longer than the actual solution of the matrix equations. A significant amount of computer time is used in obtaining physical and thermal properties as a function of temperature, and sometimes as a function of pressure or other variables, by linear interpolation from tables or by the evaluation of numerical correlations.

Since the SAS channels are only loosely coupled, SAS3D works on one channel at a time, completing a time step for one or more modules for one channel before going on to the next channel. The arrays used for each channel are stored in a few "data packs". The data packs for a channel are moved into working memory while the code is doing the calculations for that channel, and then moved out to a storage area. Thus, every time step the data packs for each channel are moved into and out of the working area a few times, and each code module is entered at least once for each channel in which the module is active. They are often 1000 or more time steps in a run.

On an IBM or CRAY-1 computer, the storage area is in main memory. On a CDC 7600, working memory is in SCM, and the storage area is in LCM. There are about 9000 words per channel in the data packs. In principal the storage area could be on disk, but in practice it is best to have the whole calculation core-contained. Storing either coding overlays or data packs on disk adds tremendous amounts of I/O time and increases the total running time by about an order of magnitude.

One important aspect of SAS3D is that no one small area of the code accounts for the bulk of the computing time, and no one subroutine accounts for more than about 15% of the total time on the IBM 370-195. The computing time is spread through many modules and many subroutines. Therefore, dramatic improvements in running time can not be obtained by improving a single algorithm or a single subroutine. In order for the code as a whole to run well, a large number of subroutines must each run well.

CODE PORTABILITY

SAS3D is currently being used by many organizations in the U.S and abroad on CDC, IBM, and UNIVAC computers. Therefore, it was written with portability in mind. ANSI-standard FORTRAN[11] is used almost entirely, and machine-dependent features are avoided. The few machine-dependent features that are required are mainly isolated in a few separate subroutines.

The UPDAT Code. One feature that contributes to both the portability and the maintainability of SAS3D is the use of the UPDAT code to modify the SAS3D source files. UPDAT, which was written at ANL by R. George, has many of the features of CDC's UPDATE code,[12] such as inserting and deleting cards, and inserting COMDECKS. In the SAS3D source, the COMMON blocks are listed only once, and the COMDECK feature is used to insert the common blocks in each subroutine where they are needed. UPDAT is also used to make corrections or modifications to the code.

Programs with features similar to those of UPDATE have been available on IBM computers, but the IBM codes and CDC's UPDATE use different directives and require different input. The UPDAT code, which was written in FORTRAN, runs on IBM, CDC, and UNIVAC computers, and all versions use the same input. Therefore, the same UPDAT input deck can be used to modify or correct the IBM, CDC, and UNIVAC versions of the code.

RUNNING SAS3D ON THE CRAY-1

Considering the size of the SAS3D code, it was relatively easy to get the code running on the NCAR CRAY-1. A few routines known to be machine dependent had to be modified, but the modifications were straight-forward. Also, the Cray Fortran Compiler[13] (CFT) would not compile a few statements, but these were mainly cases of violating the ANSI FORTRAN standards.

The actual process of putting SAS3D on the NCAR computer and running it was all done from ANL using a remote batch terminal. The main steps in this process were as follows.

1. A FORTRAN source tape was written at ANL and sent to NCAR. This tape contained 3 files. The first file was the source for the UPDAT program. The second file contained the SAS3D COMMON blocks, and the third file was the SAS3D source file.

2. The UPDAT program was compiled after the necessary modifications to the UPDAT source were made using the system UPDATE utility.

3. UPDAT was used to insert the common blocks into the SAS3D routines and to make modifications to the SAS3D source.

4. The resulting SAS3D source code was compiled, and both the source file and the object file were stored in permanent datasets on the CRAY-1 disks.

5. Sample SAS3D cases were run, and the results were compared with IBM and CDC results.

The CDC version of the export version of UPDAT was sent to NCAR. Some changes to this code were required to get it to run on the CRAY-1. Frist, some machine-dependent constants had to be changed to account for differences in word length and data representation. The CDC 7600 uses ten 6 bit characters per 60 bit word, whereas the CRAY-1 uses eight 8 bit characters per 64 bit word. Fortunately these constants were all set at the same place in DATA statements, so it was easy to change them. Second, in one spot the CRAY-1 version required the FORTRAN function SHIFTR instead of the SHIFT function used in the CDC version. Third, the UPDAT directive *END happens to correspond to a NCAR control card, so in UPDAT this directive was changed to *EEND by changing one DATA statement.

The CDC 7600 version of SAS3D was sent to NCAR. This version is overlayed, with a special overlay routine that stores overlays in LCM rather than on disk. Since the NCAR CRAY-1 has plenty of main memory, and no LCM, the CRAY version of SAS3D was not overlayed. Thus, all of the OVERLAY and CALL OVERLAY cards were removed, as well as subroutine OVERLAY.

Another known machine-dependent aspect was the data pack storage area and the routines that store and retrieve data packs. On the CDC 7600, the routines READEC and WRITEC, written in COMPASS, are used to store blocks of data in LCM when they are not being used, and to put them back in SCM when they are needed. On the ANL IBM machines, the data packs are stored in main memory, using specially optimized FORTRAN versions of READEC and WRITEC. On the CRAY-1, the data packs are stored at the end of blank common, using simple FORTRAN versions of READEC and WRITEC. The CRAY FORTRAN compiler (CFT) automatically vectorizes these versions of READEC and WRITEC. Since the CRAY loader loads coding from the bottom of memory up, with blank common at the end of the coding, and since I/O buffers start at the top of memory and work down, any unused memory is between the end of blank common and the bottom of the I/O buffers. Therefore, the effective length of the data pack storage area at the end of blank common can be set at run time by specifying the total job memory size to correspond to the size of the problem being run.

Another machine-dependent aspect of SAS3D is the timing routine TLEFT. For the CRAY-1, a TLEFT routine that sets TLEFT to 1,000,000-100.*SECOND(1.0) was used, where SECOND is the elapsed CPU time. This version gives the correct timing of various parts of the code, but it does not give an accurate warning when the code is approaching a time limit.

Some accumulated SAS3D modifications were also incorporated into the CRAY-1 version. These modifications are minor items that correct some known non-standard usages in the code.

The first attempt to compile SAS3D on the CRAY-1 turned up only 6-8 FORTRAN errors. One error was in a DATA statement in subroutine RESTAR. The IBM version of this statement uses a 4H specification, the CDC version uses 10H, and the CRAY version requires 8H. The other errors were all cases of non-standard separators in FORMAT statements. The CRAY compiler does not allow two consecutive comma separators or ,/, in a FORMAT statement. Apparently the IBM and CDC systems ignore spurious commas in FORMAT statements.

After SAS3D compiled, the first attempt to run a case turned up one last problem. Subroutine WRITEI is a multiple-entry routine, even though it should have been written originally as two separate single-entry routines. The CRAY compiler uses an IBM-type convention for passing arguments to multiple entry points, whereas the CRAY version of SAS3D started as a CDC version, with a different treatment of multiple entry points.

After the entry point problem was corrected, a number of SAS3D cases have been run successfully without encountering any additional problems.

Linear Interpolation Routines

Although the Cray Fortran Compiler will automatically vectorize part of the SAS3D coding, the FORTRAN coding in three linear interpolation routines, INTIRP, INTERP, and INTRP, will not vectorize. These routines provide an area in which moderate improvements in CRAY performance can be achieved with relatively little effort, since they are small routines that account for a moderate fraction of the total running time.

INTIRP scans a table of Y as a function of X. It obtains the result Y_1, corresponding to the input value X_1, by linear interpolation between the appropriate table values. INTERP is the same as INTIRP, except that INTERP is passed an extra parameter, IFUEL, the fuel type; and the tables used by INTERP contain entries for each fuel type, i.e., the Y array has two subscripts, Y(J, IFUEL). INTRP takes a whole array of input variables, X_1(I), and an array of fuel types, IFUELI(I), and it computes a whole array of output results, Y_1(I).

Timing studies on both the IBM 370-195 and the CDC 7600 have shown that INTIRP and INTERP spend more time scanning the tables to find the appropriate table entries than they do in the actual interpolation. INTIRP and INTERP always start scanning from the start of the table. INTRP starts scanning at the start of the table for the first variable in the input array. For later values in the imput array, INTRP starts scanning the table at the location where it found the previous value.

The table scanning loop in these routines was written in a somewhat convoluted manner in order to get into loop-mode on the IBM 370-195, since significant speed improvements are often achieved in loop-mode. A simple DO loop containing an IF statement that jumps out of the loop when the appropriate table location has been found will not run in loop-mode on the 195. The logic required to achieve loop-mode on the IBM machine degrades the performance on CDC and CRAY computers, both of which will run the simple DO loop version as a simple in-stack loop.

The Cray Fortran Compiler uses only scalar instructions to compile either the convoluted scanning loop or the simple scanning loop in the interpolation routines. Therefore, CAL verisons of these routines were written to use the vector compare instructions on the CRAY-1. Also, the CAL version of INTRP performs the actual interpolation calculations in vector mode.

Timing Results for Interpolation Routines. For timing purposes a simple driver program was written to call the interpolation routines with the proper arguments. The tables used for this program had a length of 20, which is typical of the tables used in SAS3D. An array of 12 values of X_1(I) was used, and these values were distributed fairly evenly over the tables. The values used for IFUEL were 1,1,1,1,2,2,2,2,3,3,3,3. For timing INTIRP an inner DO loop in the driver made 12 separate calls to the routine using the appropriate values for X_1 and IFUEL. For timing INTRP, one call was made to obtain an array of 12 results. In order to obtain running times large enough to measure, an outer DO loop was used to call INTRP 1000 times or to execute the inner loop for INTIRP 1000 times. Thus, the measured times are for 1000 calls to INTRP or 12,000 calls to INTIRP. Since INTERP is very similar to INTIRP, it was not timed.

Table 1 lists the running times measured for these routines. The simple DO loop for scanning does somewhat better than the original coding on any computer. One call to INTRP with an array of 12 values takes less time than the corresponding 12 calls to INTIRP, partly because subroutine linkage overhead accounts for a moderate fraction of the total INTIRP time, and partly because of the more efficient table scanning in INTRP. The FORTRAN versions of these routines run a factor of 2-2.5 times as fast on the CRAY-1 as on the IBM computers. The use of the vector compare instructions, as well as generally tighter coding, in the CAL version improves the CRAY speed by an additional factor of 3-4.

Table 1. Timing results for interpolation routines.

Computer, Compiler	CPU time (seconds) for 12,000 results		
	INTRP	INTIRP	
		original coding	simple scanning DO loop
IBM 370-195 FTH,[a] OPT=2	.165	.366	.320
IBM 3033 FTH, OPT=2	.160	.305	----
CDC 7600 FTN4,[b] OPT=2	.121	.224	.165
CRAY-1 CFT	.067	.178	.127
CRAY-1 CAL	.021	----	.032

[a]IBM's Fortran H compiler
[b]CDC's Fortran Extended, Version 4 compiler, as implemented on the Lawrence Berkeley Laboratory Computers.

SAS3D TIMING RESULTS

Timing comparisons for the SAS3D code are complicated by two factors. First, there is no one small section of the code that accounts for the great bulk of the computing time. It is necessary to run the whole code, or a significant fraction of the code, in order to get meaningful timing comparisons. Second, there are many types of cases in which it is not possible to get exactly the same results on different computers. In many cases, differences in running times between computers are due to both differences in computer speeds and differences in computational paths.

Three different SAS3D cases were run on the CRAY-1. The first case was a limited case which exercised only part of the code, but it was a case for which the same computed results are achieved on all computers, so that exact timing comparisons are meaningful. This was the first 300 time steps of a 1-channel low power boiling case (LOWBLA). This run was terminated before boiling initiation, so it only tested the pre-boiling parts of the code.

The second case was a more extensive 1-channel case: 1000 time steps for channel 1 of a 33-channel CRBR transient undercooling case (1-channel test). This case gets into sodium boiling, clad relocation, and fuel relocation (SLUMPY). The results obtained on different computers for this case were not identical, but they were quite similar; and the computational paths were quite similar.

The third case was the standard SAS3D 3-channel test case (3-channel test). This case was run mainly to test the code rather than to get timing comparisons. This case tests most of the options in the code. It is an extremely touchy case with an appreciable amount of positive feed-back. Any small deviation, due to factors such as round-off error, tends to grow as the run progresses, and it is not possible to get the same results for this case on different computers. Even the IBM 370-195 and the IBM 3033 give different results for this case.

Table 2 gives the running times on various computers for these cases. Also, given in parentheses are the relative speeds, with the IBM 370-195 speed defined as 1 for each case.

Table 2. SAS3D timing comparisons.

Computer	CPU time, seconds (Relative speed, 1/CPU time)		
	LOWBLA, 1 channel, 300 steps, no boiling	1 channel test, 1000 steps	3-channel test[a]
IBM 370/ 195	44.1 (1.0)	333.3 (1.0)	740.0 (1.0)
IBM 3033	43.8 (1.01)	309.1 (1.08)	----- ----
CDC 7600	19.4 (2.27)	162.9 (2.05)	453.9 (1.63)
CRAY-1 CFT[b]	11.8 (3.73)	129.6 (2.57)	186.2 (3.97)
CRAY-1 CAL[c]	9.7 (4.55)	95.5 (3.49)	----- -----

[a]Timing comparisons are not very meaningful for the 3-channel test case.
[b]All-Fortran version.
[c]CAL versions for three interpolation routines, everything else Fortran.

SAS3D runs slightly faster on the IBM 3033 than on the 195, about twice as fast on the CDC 7600 as on the 195, and a factor of 2.5 to 4 times as fast on the CRAY-1 as on the 195.

The only area in which performance improvement for SAS3D on the CRAY-1 was investigated was the three linear interpolation routines. Use of the CAL versions of these routines led to an improvement of 20% - 35% in the overall running time of SAS3D.

SAS4A

The initial version of SAS4A contains mainly steady-state and pre-voiding transient heat transfer and coolant flow routines. Other modules are being added as they are developed, but this initial version is the only one that has been run on the CRAY-1.

There are two main differences between these SAS4A pre-voiding routines and the corresponding SAS3D routines. First, the pre-voiding SAS4A module contains routines that have been especially tailored for prevoiding transient calculations, whereas in SAS3D the corresponding routines are generalized routines that handle the whole transient. Second, the algorithms used in the pre-voiding routines were modified somewhat to promote vectorization. This did not require major changes in algorithms; mainly it involved re-ordering of calculations and sometimes the saving of arrays of interim results. Also, some changes in programming style were required to eliminate cases where the basic algorithm allowed vectorization but programming style precluded it.

Most of the pre-voiding transient coolant calculations vectorized easily. These calculations consist mainly of obtaining coolant properties at each axial coolant node by evaluating parametric fits. The fits are all single range fits containing no branching, and properties for all coolant nodes can be calculated in parallel. Also, since typically about 30 coolant nodes are used, vector lengths of about 30 were achieved.

In cases where a complete calculation could not be vectorized, part of the calculation often could. For instance, the calculation of the coolant pressure at node J requires the value from node J-1, so the calculation could not be vectorized. In this case, the calculation of the node-to-node pressure differences would vectorize, and this calculation accounts for most of the computing time in this area. Then a small non-vectorized loop sums the differences to give the final results.

Vectorizing the heat transfer routines was more difficult. For each axial node, the temperatures at all radial nodes are solved for simultaneously by solving a tri-diagonal matrix equation. Many of the calculations used to obtain the coefficients of the matrices were vectorized, but the vector lengths were usually no longer than the number of radial nodes, which typically ranges from 4 or 5, above and below the fueled region, to about 17 in the fueled region. The tri-diagonal matrix solution itself does not vectorize in the pre-voiding mdule, although in the voiding module the corresponding matrix solution might be vectorized by solving for all axial nodes simultaneously. In the pre-voiding calculation, the computed coolant temperature at axial node J is needed to obtain some of the coefficients for node J+1; but in the voiding module, the coolant temperatures are calculated separately in the coolant routines, and in the fuel pin heat transfer routines there is no coupling between axial nodes.

SAS4A TIMING RESULTS

A non-voiding case with 1000 time steps was timed using the initial version of SAS4A. In addition to total running times, a timing distribution by subroutine was also obtained. For the timing distribution, the CFT timing trace was used on the CRAY-1, the PROGLOOK feature was used on the ANL IBM 370-195, and a combination of the SNOOPY routines plus a number of calls to SECOND was used on the CDC 7600. For this case, the steady-state initialization accounts for less than 1% of the running time, so it was mainly the pre-voiding transient routines that were timed. Tables 3 and 4 give these timing results.

These timing results indicate that SAS4A runs about six times as fast on the CRAY-1 as on the IBM 370-195; with CAL versions of the interpolation routines, the speed ratio increases to seven. The CDC 7600 version also runs appreciably faster than the IBM version, but the CRAY-1 version still runs a factor of 2.1 to 2.5 times as fast as the CDC 7600 version.

Table 3. SAS4A timing comparisons.

Computer	CPU time seconds	Relative speed 1/CPU time
IBM 370-195	43.6	1.0
IBM 3033	46.5	.94
CDC 7600	15.49	2.8
CRAY-1, CFT[a]	7.37	5.9
CRAY-1, CAL[b]	6.14	7.1

[a]All-Fortran version.
[b]CAL versions of interpolation routines, CFT for rest of code.

Table 4. Detailed breakdown of SAS4A timing.

Program area	CPU time, seconds (percentage of total)			
	IBM 370-195	CDC 7600	CRAY-1 CFT	CRAY-1 CAL[a]
heat transfer, except matrix solution	9.98 (23%)	4.54 (29%)	2.01 (27%)	2.01 (33%)
matrix solution	1.81 (4%)	1.41 (9%)	.51 (7%)	.51 (8%)
interpolation routines	4.38 (10%)	3.46 (22%)	1.91 (26%)	.68 (11%)
coolant routines	2.03 (5%)	1.19 (8%)	.42 (6%)	.42 (7%)
log, exp, x^y	6.85 (16%)	1.84 (12%)	b	b
READEC, WRITEC, data pack movement	9.22 (21%)	.56 (4%)	.32 (4%)	.32 (5%)
formatted I/O	8.41 (19%)	2.38 (15%)	2.10 (29%)	2.10 (34%)
other	.92 (2%)	.11 (1%)	.10 (1%)	.10 (2%)
total	43.6 (100%)	15.49 (100%)	7.37 (100%)	6.14 (100%)

[a]CAL versions of INTRP, IINTIRP, CFT for rest of code.
[b]Included with coolant routines.

Comparisons with the LOWBLA times in Table 2 for 300 pre-voiding steps show that SAS4A runs 1000 steps in about the same time that SAS3D requires for 300 steps. About 1/3 of the SAS3D time for this case was accounted for by the DEFORM module, which has not been incorporated into SAS4A yet; but the remaining 2/3 of the time is in routines corresponding to the SAS4A routines. This indicates that the heat transfer and coolant routines in SAS4A run about twice as fast as the corresponding SAS3D routines. On the CRAY-1, the SAS4A speed improvement is greater than a factor of two.

Formatted I/O, mainly printing transient results, accounted for an appreciable fraction of the total CPU time for this case. In 6 seconds of computing on the CRAY-1, this case printed 213 pages of output. For longer runs, it will be necessary to reduce greatly the amount of print-out per second of computation.

The log and exponential functions are called by the coolant routines. On the IBM and CDC computers, and maybe on the CRAY-1 also, they account for the bulk of the time spent in the coolant routines. The coolant routines vectorized well on the CRAY-1, and this shows in the relative coolant calculation times. On the CRAY-1, the coolant routines, including the log and exponential functions, run about 21 times as fast as on the IBM 370-195, or about 7.2 times as fast as on the CDC 7600.

The data pack movement is quite a bit slower on the IBM 370-195 than on either the CRAY-1 or the CDC 7600. This reflects the relatively slow speed of the main memory on the IBM 370-195.

SUMMARY AND CONCLUSIONS

The SAS3D code was run on the CRAY-1 computer with only minor modificaitons to the code, and it ran reasonably well. On the CRAY-1, SAS3D runs 3.5 to 4.5 times as fast as on the IBM 370-195, or about twice as fast as on the CDC 7600. The IBM and CDC compilers that are used with SAS3D are highly developed compilers that produce well optimized object code, so the performance of SAS3D on the CRAY-1 is a reflection of the basic speed of the CRAY-1 hardware, as well being an indiciation that the CFT compiler produces moderately efficient object code.

Writing new pre-voiding heat transfer and coolant flow routines for SAS4A, using new algorithms and coding that would vectorize where possible, led to significant speed improvements on all three computers. The CRAY-1 version of SAS4A runs about 2.5 times as fast as the CDC 7600 version, which runs 2.8 times as fast as the IBM 370-195 version. Even the IBM version of SAS4A runs about twice as fast as the corresponding IBM version of SAS3D. Some of the improved speed of SAS4A, as compared to SAS3D, was probably due to the use of somewhat better algorithms and the elimination of some unnecessary calculations, but much of this improvement is probably due to the fact that coding that vectorizes on a CRAY-1 tends to run efficiently on an IBM 370-195 or a CDC 7600. Therefore, new coding for the SAS codes should be vectorizable where possible, even if it is not expected that these codes will be run extensively on vector machines in the near future.

The performance attained by SAS3D and SAS4A on the CRAY-1 was partly due to the ability of the CRAY-1 to use short vectors effectively. Many of the vector lengths in these codes are in the range from 10-20, and some are as small as 3 or 4. Other than block transfers, none of the vector lengths is currently greater than 48. The SAS codes probably would not perform well on a computer designed for long vectors.

The computing speed attainable on a CRAY-1 computer should make it feasible to develop "few pin" models for whole core accident analysis, if the models are carefully developed and coded so as to optimise computer performance. Such calculations may even be feasible on a CDC 7600. Even on a CRAY-1, detailed many pin (217 pins per subassembly) models would probably still be restricted to limited, single subassembly cases because of running time considerations.

ACKNOWLEDGMENTS

We would like to thank L. Rudsinski for obtaining computer time on the NCAR CRAY-1 for this project, and also for his assistance in getting SAS3D to the NCAR computer. This work was performed under the auspices of the U. S. Department of Energy.

REFERENCES

1. D. R. MacFarlane, J. C. Carter, G. J. Fischer, T. J. Heames, N. A. McNeal, W. T. Sha, C. K. Sanathanan, and C. K. Youngdahl, ANL-7607 (1970).

2. F. E. Dunn, G. J. Fischer, T. J. Heames, P. A. Pizzica, N. A. McNeal, W. R. Bohl and S. M. Prastein, ANL-8138 (1974).

3. M. G. Stevenson, W. R. Bohl, F. E. Dunn, T. J. Heames, G. Höppner, and L. L. Smith, Proceedings of the Fast Reactor Safety Meeting, Beverly Hills, CA, (1974), CONF-740401, p. 1303.

4. J. E. Cahalan, D. R. Ferguson, H. U. Wider, C. H. Bowers, L. L. Briggs, F. E. Dunn, J. M. Kyser, L. Mync, A. M. Tentner, and W. L. Wang, ANS/ENS International Meeting on Fast Reactor Safety Technology, Seattle, Washington, (1979).

5. G. Höppner, W. L. Chen, F. E. Dunn, and M. A. Grolmes, Trans. Am. Nucl. Soc., 18, 213 (1974).

6. I. T. Hwang, T. M. Kuzaz, W. W. Marr, and K. J. Miles, Trans. Am. Nucl. Soc., 28, 443 (1978).

7. M. Ishii and W. L. Chen, Trans. Am. Nucl. Soc., 28, 442 (1978).

8. D. S. Rowe, BNWL-1522-4 (1971).

9. W. W. Marr, ANL-8131 (1975).

10. W. T. Sha, ANL-7796, NUREG/CR-0785 (1979).

11. "American National Standard Programming Language FORTRAN," ANSI x3.9-1978.

12. Update Reference Manual, Control Data Corporation Publication No. 60342500, Minneapolis, Minnesota, (1978).

13. CRAY-1 FORTRAN (CFT) Reference Manual, Cray Research Publication No. 2240009, Bloomington, Minnesota, (1977).