Computing and the Kaon - Experience from a physicist

Giles Barr
CERN, Geneva, Switzerland.

1 Introduction

The NA31/NA48 kaon physics program at CERN is primarily concerned with the measurement of the effect of direct CP violation in the kaon system. CP violation in the state mixing interaction between the \( K^0 \) and the \( \bar{K}^0 \) was discovered more than a quarter of a century ago. A mechanism for direct CP violation is predicted from the Standard Model. To study this, the quantity

\[
R = \frac{\Gamma(K_L^0 \rightarrow \pi^0\pi^0)}{\Gamma(K_L^0 \rightarrow \pi^+\pi^-)} \times \frac{\Gamma(K_S^0 \rightarrow \pi^+\pi^-)}{\Gamma(K_S^0 \rightarrow \pi^0\pi^0)}
\]  

(1)

is experimentally measured. \( R \) is then used to determine the parameter \( Re(\epsilon'/\epsilon) \approx (1 - R)/6 \).

The principle problems involved with the determination of \( R \) are (1) The difference between \( K^0_L \) and \( K^0_S \) decays differs by at most 2\% (this means that our final event samples are very large, 2.5 million events per year in NA31, and a factor 10 more expected from NA48). (2) There is background from CP conserving \( K^0_L \) decays. Only one \( K^0_L \) decay in 300 is a CP violating one. (3) Detector stability, calibration and efficiencies have to be controlled carefully. (4) A method is employed to investigate and correct for the losses due to random hits interfering with the reconstruction of a good event.

The most recent preliminary results were presented last summer at the LPHEP91 conference in Geneva. The combined NA31 result is \((2.3 \pm 0.7) \times 10^{-3}\). The result from the E731 experiment at Fermilab from the same conference is \((0.60 \pm 0.69) \times 10^{-3}\). A 1.7\( \sigma \) discrepancy exists between the two numbers. Theoretical predictions for \( \epsilon'/\epsilon \) lie between about zero and about \( 2 \times 10^{-3} \). A further round of improvements in both experimental and theoretical work is needed.

2 NA48: The choices a new experiment faces

NA48 was approved in November 1991. Although software work is at an early stage, it is interesting to note the directions in which the collaboration has decided to move with respect to it’s software. A battle within the collaboration on whether to use the FORTRAN or C programming languages for communal offline code has been fought and it has been decided that FORTRAN will be used. This is mainly because of the large pledge of support for this language from CERNLIB and also for the need to have code with a small set of simple standardised constructs which are easy to follow by all. Code will be managed centrally using PATCHY. Individual users may also use CMZ. The group is considering the use of object oriented design techniques and software for a test beam calorimeter analysis has been written in this style using the ADAMO package. The success of this technique will be evaluated at the end of the analysis program. NA48 will need about 300,000 CERN units of computing power to process events. It is planned to build a workstation and disk farm close to the experiment. We are also considering writing code suitable for use with massive parallel processing machines.
3 NA31: How a program has evolved over eight years

The NA31 code was written between 1984 and 1986 and used for a first serious data production when the 1986 data arrived. Since then, it has evolved to include the addition of new reconstruction techniques, new detector elements and new readout information. I now describe a problem which occurred during the lifecycle of the program. The majority of the original authors left the collaboration and the coding conventions and structures were not passed on to (or were ignored by) successive generations of code managers. The variable naming conventions were not followed. The 'standard' places where cuts should be made were no longer identifiable as such. Cuts were placed anywhere in the code. Similarly, various bank operations such as creation and deletion were no longer well structured. Modifications to the code were made by introducing files of updates, and often these updates and the accompanying comments were written in a way which made the updates easy to follow, but the resulting code (once a new cycle of the libraries was generated) difficult to decipher. The main feature of the modified code was that it contained many IF statements which bypassed old code. The program underwent a major revision during 1989 and 1990 in which much duplicate code was produced and compared against the originals. Some new algorithms were introduced. No serious blunders were uncovered. The calling sequence, information flow and places where cuts were made were clarified again.

Apart from this one problem, the NA31 code has performed well. Various CERNLIB packages (KAPACK, EPIO, ZBOOK, HBOOK.. ) have been used successfully. The computer network has been used extensively, in particular, remote batch submission has allowed us to distribute the raw data to different computer centres to share the processing load. We have a number of different REXX programs which manage the submission and return of production jobs. Typically, we run about 10 productions (each with 3 stages) on the data (or some fraction of it) each year from CERN. Recently, we have adapted the EXECs to work with TMS and FATMEN.

4 How a personal software project proceeds

In both NA31 and NA48, the majority of the programs are run in batch. Almost all batch jobs proceed along the same lines. (1) Run PATCHY (for NA31, HISTORIAN was used). (2) Compile modified routines. (3) Link with library of routines which have not been changed, CERNLIB and run time FORTRAN libraries. (4) Run. This mode of working has the advantage that once the program is working, several runs with slightly modified conditions can be submitted all at once and the results appear in tidily organised output files in the morning. One particular deficiency of VAX/VMS is that it does not allow a private temporary disk to be allocated, so care must be taken if two jobs run simultaneously that files don't accidentally get crossed.

The user typically submits short jobs with extra print statements included, to check the correct operation of the program. Then a full tape is processed, then a string of jobs is submitted to process the complete data sample and the results are concatenated.

5 Variable naming convention

One of the biggest learning tasks facing someone joining a new experiment is to master the software coding conventions and rules and to learn where everything is. Once a large program such as a reconstruction package is established, virtually all code is presented to the user as a PATCHY modification set. It is rare therefore to see an identifier declared and used on the same
screenfull of text at the same time. Naming conventions have been successfully used in the past, however one which indicates more of the attributes of each identifier is needed.

The following scheme illustrates the sort of features desired.

```fortran
*D DECMA .208
   IF (phN .LE. PHNMAX) THEN
*I PAKMIN .43
   chXHIT(i)=chXHIT(i)*xshift

It is immediately obvious without referring to the declarations in the master listing that PHNMAX is a constant parameter (it is in uppercase), i and xshift are integer and real local variables respectively (they are lowercase) and that phN and chXHIT are integer and real variables from common blocks /phCOMMON/ and /chCOMMON/ respectively. The compiler ignores the changes of case. All variables must be declared (as if IMPLICIT NONE has been used) for this scheme to work.

6 Avoidance of silly FORTRAN errors

In general, the greatest obstacle to completing an analysis topic is the avoidance of bugs. This is particularly so when preparing a production job when one has to collect a number of changes from different collaborators, combine them and then check that the resulting program functions correctly. When bugs are found, they usually tend to be similar to errors that have been seen before. The compiler can be used to detect certain errors (for example by enabling array bound checking), but certain CERNLIB practices (in particular UCOPY and VZERO) prevent these checks from being comprehensive. Here is a list of some of the common mistakes which can take a long time to detect, locate and remove.

- Overwriting outside the bounds of a ZEBRA bank.
- Changing the type of an argument as it is passed to a subroutine.
- DO I=1,N; A(N)=0; ENDDO (Using the wrong subscript in a loop because indexing A() with N is familiar in another part of the code.
- Copying chunks of code with the editor and doing some changes, but forgetting some of the changes.

There is a similar list of funnies for the C programming language. I would like to see a number of extra facilities to allow the automatic checking of programs (particularly prior to launching a big production). Here are a few suggestions:

- The compiler vendors could allow access to the internal structures which are created during the compilation process (perhaps by a callable interface to the compiler), particularly, the cross reference listing, the global analysis which is done by the optimizer and the partitioning of the code into different blocks. This would allow custom code checkers to be written to print a concise, tailored variable usage summary.
- Either the compiler could provide inline code expansion or the code manager (e.g. PATCHY) could provide a macro expansion facility. By performing accesses to the main data store via these macros, normal runs can proceed as fast as if the accesses had been coded inline by hand. Special runs could easily be set in which more elaborate macros could do runtime checks of each access, array bound checks, histogramming the frequency of reads and writes, etc.

- We need 100% error checking in the communal code written for the experiment and in external packages (e.g. CERNLIB) so that error messages are specific and related to the first time a misuse is detectable.

7 NA31 tools

There are a number of programs which have been created to meet needs within the NA31 collaboration. It may be interesting to consider whether any generally available software could be provided to meet these needs or whether these programs could form the basis of a generally available facility.

- Production submission execs and tape catalogue. These generate chains of jobs to process a number of tapes. See above.

- Character histograms: A histogram is filled with a character variable key. At printout time, a list of all the keys and the number of times the key has been ‘filled’ is produced. Also works with bit patterns. Useful for producing tables. Produces modular code. Easy to add extra lines to the table.

- A data definition language definition of the MiniDST format. MORTRAN macros are used to produce packing, unpacking and checking routines.

8 Summary

Whereas NA48 is still in the stage where few of its software initiatives have shown signs of wear, the NA31 code has almost completed it’s lifecycle and a number of observations are made. In particular, the need for rigidly following certain design structures throughout the lifetime of the code rather than the contract-time of the programmers is emphasised. It is noted that a large fraction of coding errors are self inflicted mistakes which follow a common pattern and it is clear that automated procedures to detect these problems are needed. A few suggestions are made.