S40 137

Again packetcap

DAWG-REP.5.0/93
L.Chiappetti - IFCTR - Mon, 10 May 1993

0. FOREWORD

This report describes technical problems associated with the prototype of accumulation programs based on the packetcap approach. The report was written while building the software and performing the tests, therefore it is not quite organic, and contains lot of technical details. It is articulated as follows :

Section 1 indicates the work done, how much time was spent on it (important for SAX DAWG !) and also tells you how to retrieve the software source code (the test program names are found in section 4).

Section 2 was written before the software, and is the project of the program. This project may have changed in the actual software because of the results of the test.

Section 3 is a brief reminder of the simulated data (the format is representative of real data, not the content) used to test the code. Just in case you may want to run it.

Section 4 is an extensive report of the speed tests, and of the changes done while writing the software. I have so far kept all intermediate versions of the test programs (main and subroutines all in one file, you can judge the difference with a plain (dx)diff), as well as the final version (separated by library subroutines).

Section 5 summarizes the main technical conclusions.

Now for those of you which just want an executive summary, here it is : I have done a very thorough job of testing the packetcap approach and the classical approach, in one representative case. The packetcap program is slower than the classical one, but this slowness can be (and actually was) decreased by a careful analysis. In any case also the packetcap program is fast (how do you like 3 sec for 150 thousand event ?). The packetcap programs are modular, which means legible, and once the basic stuff is done it is extremely easy to change them into new programs (less than half an hour and off you go). Those of you interested in some associated problems should read at least section 5. Those interested in nitty gritty details (I hope MANY of you, which means MANY software writers !) should read it all, and also retrieve the code, and look at that.

Please all note that the prototype spectrum (SAXHACCUM) and image (SAXIACCUM) accumulation programs and relevant libraries currently handle only "Laben" FOT direct mode data (if somebody writes a packetcap entry for them). All other modes are unsupported (in general there is an dummy routine).

1. DESCRIPTION OF WORK DONE

First step was to make the template program work.
This version is a single program, with all subroutines in its source file, and most functions performed by code in the main program. This program WORKS.
The template `SAXHACCUM` was almost ready before the Frascati DAWG meeting. It took about 2 hrs on Apr 15 to debug it and make it working (the packetcap routines were already debugged; most of the debugging concerned XAS stuff and only part of it the `READ_NEXT_EVENT` routine). The reason for the delay was the installation of the new SAX DECstations at IFCTR.

The next steps were to write two simple ad-hoc programs to generate dummy data (see 3), and to provide the accumulation in the classical way (this required just 2-3 hrs).

The next series of steps was to test the speed of `SAXHACCUM` in several variants in order to evaluate its performance versus the classical approach.This took about 2 days.

Next step was to make the program more legible, isolating most of the code in subroutines.
The purpose of this is to make easier for future program developers to adapt this template to other functions, with a minimum number of changes. This step required an afternoon on Apr 19. The structure of the resulting `SAXHACCUM` is definitely very legible.
The relevant software may be retrieved via ftp from `kronos.ifctr.mi.cnr.it` or `poseidon.ifctr.mi.cnr.it` (user public password access) in the directories indicated here below.
All library routines are **provisionally** located on `poseidon` in `$XASTOP/libsource/pktcap`.
One `INCLUDE` file `accumcommon.inc` is located in `$XASTOP/include`. `$XASTOP` is `/poseidon/lucio/xas`. The source of the main programs are instead located outside the XAS tree on `/poseidon/lucio/temp/pktcap`.
The packetcap files themselves are where they should be, i.e. `$XASTOP/calib/sax/`*`instrument`*.

A further step was to assess byteswap overhead on Sun (or big endian machines; remember that SAX telemetry data will have little endian arrangement, compatible with Vax and DEC). A description of the tests made (about half a day, including fixing some Sun problems and a bug described here below) is reported below in 4.3
Doing this I detected an error (an `INTEGER` instead of `INTEGER*2`) in the main accumulation/decoding routine. This has been fixed in the library version for `SAXHACCUM`, but not in the test versions `SAXHACCUMn`, n=1,14, which therefore work only on DECstation.

As a further step to demonstrate maintainability I tried making a `SAXIACCUM` spatial accumulation program. This was generated editing the main of `SAXIACCUM`, replacing the spectrum output code with some image output code merged in from the preexisting `XGACCUM` program, replacing the `INCREMENT_SPECTRUM` routine with an `INCREMENT_IMAGE` routine, replacing the `LOCAL` common, adding statements for image dynamic memory allocation, and changing a few other statements for a total of less than 20 changes.
It took about 20 minutes to do that, including debugging a few typos and writing this documentation.
The only absolutely minor problem is that the range dialogue presents a default of 0-255 for the XY range, while for pixels the usual convention is to go from 1 to 256. (This could be adjusted taking advantage of the `NFORMAT` argument, where the value "2" indicates the data structure being accumulated is an image. Or should we instead specify in the packetcap file for positional quantities that "one must be added" to the coordinate ? Decision on a convention is needed).

This draft report completed at this stage on April 20. Shown in draft form to a restricted set of people attending GS meeting in Rome.

The next test was the implementation of secondary type =2 (for WFC), and the relevant tests.
Time taken : about 2 days (May 7 and 10) including generating dummy data in three variants, testing classical accumulations in all three, writing packetcap, testing packetcap accumulations.
BTW about one and half hour was spent in tracing a bug which resulted a typo in the packetcap file (--> we need a packetcap syntax checker).

To do: try to implement indirect mode

2. DESIGN OF THE TEMPLATE PROGRAM STRUCTURE

The following gives the scheme of a generic accumulation program, which accumulates an "output file" (image, spectrum, time profile, photon list). By "parameters" I mean "what is present in a packet", like X Y E T, or spectra, or images etc.. By "interesting parameters" I mean those parameters which are relevant for the accumulation (for a spectrum this is E, for an image X and Y etc.)

```
MAIN program
        get output file name                              D
        get which observations to analyze                 A
        retrieve initial packetcap information            A
        retrieve preliminary info on packet content       B
        get range of interesting parameter(s)             B
        get binning for interesting paramter(s)           B
        get range for other parameters                    B
        open telemetry files                              A
        main accumulation loop                            C
        create output file                                D
```

It is intended to have the main program as close as possible to the above scheme : each line will be translated in a subroutine call, or in a few lines of code.

It is also desirable to reuse code as most as possible. The ABCD define a class for the corresponding subroutine or piece of code as follows :

- A indicates "generic" code. This means this is a standard library function which applies to any accumulation. It is written and mantained in one place only and by one person, all other programmers use it as a black box and do not have to do anything.
- B indicates "semi-generic" code. This ideally means a standard library function which caters for all possible cases. The programmer has only to supply appropriate parameters to it (for instance to control the dialogue about the range and binning, one has just to say, for a spectrum : dialogue concerns 1 parameter named PHA; for an image, 2 parameters named X and Y; for a pseudo-image, 2 parameters named as specified by the user in a preliminary dialogue specific to the program).
  If this is not possible, the programmer has to replace the standard library routine with a specific one based on a template code.
- C indicates "hidden" code. At main program level this calls a standard library function, but this function will call some accumulation specific `EXTERNAL` routine. This concept is better explained below.
- D indicates "specific" code. This code is written by the programmer and different for each accumulation program.

2.1. Discussion about semi-generic code

Semi-generic code is represented in the final version by routines `SAX_ACC_OTHER_RANGE`, `SAX_ACC_PRELOAD`, `SAX_ACC_RANGE`. These routines are essentially handling dialogue with the user about ranges.
In the current prototype version they receive an argument `NDIMENS` (which tells whether the output product is 1-dimensional or 2-dimensional), and a code `NFORMAT` (which should be used in principle to distinguish a spectrum, an histogram, a time profile, a spatial image, a pseudoimage).The latter code is presently not used, but it might be (e.g. for support of indirect mode data and different dialogues for spectra and time profiles, etc.).

APPENDIX - PACKETCAP FIELDS

This table reproduced from an Excel file, reports all presently defined fields in alphabetic order. The original file is instead in recommended occurrence order.
Entries in *italics* are not yet implemented, entries in **boldface** are there for WFC only.

Field	Type	Description
"-"	n/a	packet name (terminated by pipe or colon separator)
"--"	n/a	packet description comment (optional, follows packet name, terminated by colon)
bt	number	Basic Type: 1=direct (event-by-event); 2=indirect spectral, etc.
d<n>	**number**	**if bt#1:st#2 might be present to indicate field content must be divided by this value**
du	string	Data Unit : all data lengths are in "bits" or "bytes"
eg	**number**	**Events per Group (bt#1:st#2) number of events in all groups but last**
f<n>	string	Field <n> (with n a number) name; fields listed in the order in which they appear in one event
ff	string	FFields name, whenever all fields have the same name (e.g. bt#2, channels of a spectrum)
h<n>	string	Header field <n> name; fields listed in the order in which they appear in the header
hl	number	Header Length in bytes
i2	string	I*2 endianness (LE for little endian, VMS,DEC; BE for big endian, standar 2-complement machines)
i4	string	I*4 endianness (LE for little endian, VMS,DEC; BE for big endian, standar 2-complement machines)
ip	boolean	if set Invalid data Pointer is present in header (exclusive with ve)
lg	**number**	**Last Group (bt#1:st#2) number of events in last group (0 if all groups are equal)**
m<n>	**number**	**if bt#1:st#2 might be present to indicate field content must be taken modulo this value**
m<n>	*number*	*Minimum legal value for field <n>; (optional, default 0)*
M<n>	*number*	*MAXimum legal value for field <n>; (optional, default max allowed by binary field size)*
mf	*number*	*Minimum legal value for all fields ff; (optional, default 0)*
Mf	*number*	*MAXimum legal value for all fields ff; (optional, default max allowed by binary field size)*
nf	number	Number of Fields (for bt#1 number of fields present for each event)
nf	number	Number of Fields (for bt#2 number of channels in each spectrum TBV)
ng	**number**	**Number of Groups (bt#1:st#2) excluding last group if shorter**
nh	number	Number of Header fields (if hl .ne. 0)
ni	number	Number of Items (for bt#1 number of events/packet; for bt#2 number of spectra/packet etc.)
o<n>	**number**	**Offset of field <n> in group (used if bt#1:st#2:u<n>) in units TBV (E.G +du from start, -du from end of group; however how to handle last group ?**
pl	number	Packet Length in bytes
s<n>	number	Size of field <n> in du units
s<n>	**number**	**if bt#1:st#2 may be set to zero if field is packed jointly with previous field (previous requires d<n>, this requires m<n>)**
sf	number	Size of all Fields ff in du units
st	number	Secondary Type (for bt#1): 1="Laben" format (sequence of events) 2="SRON" format (sequence of groups of events, last may be shorter)
st	number	Secondary Type (for bt#2): TBD

tc	string	reference to another packetcap entry (if present must be the last field in current entry)
tl	number	Time Location, location of a special time field in header (in bytes or du units TBD)
tm	TBD	Time Mask: mask to correlate item time and header time and ultimately with UT; 0 means no mask needed
tr	number	Time Resolution (expressed as base 2 logarithm, e.g. -14 is 2**-14 s)
ts	number	Time size, size of a special time field in header (in bytes or du units TBD)
tz	number	Time Zero : start time for time counter, 0 means arbitrary, TBV
u<n>	**boolean**	**if bt#1:st#2 and field is Unique (appears once in group) set this**
ve	boolean	if set Valid Event counter is present in header (exclusive with ip)
vl	number	Valid event counter Location in header (in bytes or du units TBD)
vs	number	Valid event counter Size (in bytes or du units TBD)
z<n>	number	siZe of header field <n> in du units