Alan Jones (Iowa, USA) - COMAL on the C64

Doug Snell asked me to write something up about my involvement with Comal. This will be a reflection on my experience with Comal and the Commodore 64 computer. I am an engineer and in the early 1980's I was looking for a home computer. I always had access to computers at school and work, so I did not need a powerful computer at home. In fact, I could do quite a bit with a programmable calculator, but it was severely I/O bound. Apple computers seemed overpriced and it was too early to pick a winner from the newer 16 bit computers. Atari computers seemed too tied to the game market. The new C64 was looking good at $599, although the needed peripherals would add quite a bit to the cost. When the price suddenly dropped to $199, I bought one the next day. The C64 proved to be very popular. I had picked a winner. I paid high prices for the needed peripherals and started programming.

BASIC 2.0 was awful. Well, OK, it was built in and you could share your programs with all C64 users. My biggest complaint with BASIC is that I can't write true subroutines with parameter passing and develop and use a library of subroutines. Assembly language programming was too slow and tedious to rapidly develop programs. I had tried writing a model rocket performance program in Basic. It was a bad experience and the C64 would have been relegated to the closet if I had not found something better. I was not impressed with the various Basic extensions and other programming languages that were available for the C64. The local computer club had grown to a few hundred members, mostly C64 users. At one meeting I heard about Comal and a Comal SIG was formed with about 20 members. I tried Comal 0.14 and I instantly liked it.

The Comal Users Group, USA, Limited was a strange entity led by Len Lindsay. They were the US distributors of Comal and published the "Comal Today" newsletter. You could not actually join The Comal Users Group, but you could subscribe to Comal Today. The disk-loaded Comal 0.14 was freeware, or low cost shareware. Its status evolved over the years, with many improvements as well. The odd thing was that independent user groups had permission to distribute Comal 0.14 and copy Comal Today from a user group subscription. I received the early issues of Comal Today this way. I had a personal subscription for a time, and I was able to buy the last three issues as individual copies. I was able to obtain the entire 27 issues with disks. I felt that I was fortunate to be able to get into Comal from the start as a user and to follow Comal as it developed. Actually Comal got started almost a decade earlier than I did. There is a good article, "The Story of Comal told by Borge Christiansen", in Comal Today #25, 1989. I was able to get in on the ground floor with Comal on the C64 in the US. I have always said that Comal Today and the quality and diversity of the people that supported it was the best feature of Comal. I even had a few articles published.

Comal 0.14 was disk loaded and took a long time to load from a stock 1541 drive. It only provided about 10K of programming space. But I could develop decent software with it. I might add that Comal accounts its free memory differently than Basic. Comal always gives you the hi-res graphic screen, while Basic has to allocate it from main memory. Comal also does a bit more with a fixed amount of memory than Basic. Later versions and patches freed up more memory and added more commands. The culmination was the Power Driver, which gave about 15K, and gave nearly all the features of UniComal 2.0 except for packages, external subroutines, closed subroutines, the trap structure, and faster speed. It also had a "compiler". The Power Driver system is quite nice. If I had had it and a 1581 (3.5") drive with JiffyDos when I started, I might not have bought a Comal 2.0 cartridge.

Comal is a three pass run time compiler or interpreter. The first pass is when you enter a line of code. It does syntax checking, builds the name table, and tokenizes the program line. The second pass is when

you enter run. It resolves program flow addresses. The third pass is the actual execution of the program. List, Edit, and Display recreates the program from the tokenized form and name tables. To share a Comal 0.14 program with someone, they need the same or a compatible Comal system and the Comal program. The Power Driver compiler creates a single executable program that binds the application program to the Comal system and strips out the parts not strictly needed to execute the program. This makes about 25K of space available, say for larger array sizes. The "compiled" programs are all huge since they contain the Power Driver runtime system, but they can be simply loaded and run on any C64. These are not very popular due to their large size. Comal 2.0 has something similar called a protection program. It strips out the names from the name tables and anything else that is not strictly needed to execute the program. This frees up a little more RAM for data. The program is thus protected from listing or displaying the program since the information needed to recreate it is no longer there.

UniComal 2.0 was announced and eagerly awaited. It was cartridge based so it did not have to be loaded from disk. It was faster and made about 30K of RAM available. This is roughly on par with Basic's 39K free. It also made it easier to use ML routines with packages, and program overlays with external subroutines. It added a few more commands and ran much faster. The Cartridge cost $100 - $125 US. I bought one of the first cartridges available. This was a four socket 64K cartridge made in the US under license from Unicomal. Later cartridges had three sockets for 32K ROMS, allowing for up to 32K of additional packages to be placed there. The Comal Users Group developed and marketed one 16K "Super Chip" ROM of packages. This became a defacto standard among 2.0 users. I modified my early Comal cartridge to hold up to four 32K EPROMS and added Super Chip. Sadly, this was the only ROM or EPROM based collection of packages produced. Of course there were several disk based packages released.

Comal is a standardized language and is available for many different computers. I'm not sure what the standard may have evolved to, but certainly Comal-80 was standard. Comal was available for the Commodore PET, C64, 128, Amiga, CP/M machines, IBM PCs, Unix, and most recently, for Macintosh computers. There were others as well, but I am most concerned with the C64/128 versions here. There was a 128 specific cartridge available that allowed use of the 128K of RAM. It gave about 40K free, plus about 40K for buffer or RAMDisk space. It was not fully compatible with packages developed for the C64 cartridge. It was more expensive, and it did not sell well. The Super Chip included a 128 package that enabled use of most of the 128 features, including the use of the VDC screen 80 column and hi-res display graphics, and 2 Mhz clock speed. The CP/M version was slow, especially on a 128, but it was reasonably priced. The IBM PC versions, UniComal 2.x and 3.0x, were good, but they were too expensive at their introduction. The Amiga version was good and well priced, but the Amiga was never very popular. Today I have a 386 PC running Linux and I would be most interested in the Unix version of Comal. Little is known about the Unix version.

Comal is essentially dead in the US now, although I understand that it was always more popular in some parts of Europe, and is still in use today. I still use my 128 every day, mostly for telecommunications, editing, and word processing. I don't program much with Comal these days since there is no one else interested in running Comal programs. I don't envision Comal making a comeback in the US, although it would not hurt to try, especially on the C64/128. There should certainly be a WWW or FTP site where Comal users can share their programs, and perhaps a Comal Today like e-zine newsletter. Contemporary computers have many other choices of good programming languages, but Comal is well matched to the capabilites of the C64.

Although Comal is promoted as an educational language, it is also an excellent development language. You can write programs quickly and easily to develop new algorithms. It is sort of like a Fortran compiler with a very good debugger always available. There is even a Comal to Fortran source code translation program. Commodore really dropped the ball when they introduced the new C64C with

GEOS included when they could have had Comal 2.0 installed. Just imagine if every C64C user had been transformed into a power programmer instead of a point and click user.

I have only one significant complaint with Comal, at least as I know it on the C64/128. Comal needs continuation lines. Actually the problem is with the editor which was adapted from Basic 2.0, and not a problem with Comal itself. Internally, Comal will happily execute program lines tokenized to up to 256 bytes. However, the editor will only allow you to to create 80 character lines (two 40 column screen lines), untokenized length. Actually if you use a separate word processor, you can create and use 120 column lines. Comal allows long meaningful variable names, but they often have be shortened to a single character to fit on the limited line length. This is most critical on function and parameter statements where it severely limits the number of parameters that can be passed.

So what have I used Comal for?

I have written many model rocketry related programs, and many numerical programs. I also used to use Comal just to read a text file or directory. It is so much easier to use than Basic 2.0. Do you remember seeing those Commodore commercials on television about sending your kid to college with a C64? What a joke! Nevertheless, I did take Comal and my C64 with me when I went back to graduate school. I even used it on some homework assignments. I could check on some jobs running on the university computers using my C64 and a 300 baud modem. It's not much, but it can save you a long walk to campus on a miserable night.

First I have to tell you about my favorite Comal package. Nick Higham wrote a level 1 BLAS package for Comal. BLAS are Basic Linear Algebra Subroutines. The idea is that you optimize the BLAS for each computer. Then writing fast portable numerical programs in a higher order language is just a matter of calling the BLAS subroutines. The BLAS do n operations on real arrays. For example the dot product will multiply and sum n elements from two arrays:

```
prod:=sdot(n#,x(1),1,y(1),1)

  or

prod:=0

FOR i#:=1 TO n# DO prod:+x(i#)*y(i#)
```

Of course the BLAS are very flexible and you can use rows and columns of multidimensional arrays and any "stride". This allows us to very conveniently use fast ML code to execute the innermost loops of our programs. The interpreter has some overhead, and calculating offsets into multidimensional arrays from the indices can be very slow. Using the BLAS can speed up some Comal programs by a factor of ten. Calling a BLAS subroutine itself has overhead of about four FLOPS, where a FLoating point OPeration is one multiply, one add, and some indexing overhead. If your program or subroutine uses an average vector length of 40, your program will run at about 90% of full ML speed while retaining the conveniece of using a higher order language. The BLAS are often associated with the Fortran LINPACK subroutines for solving linear systems. In fact, I have translated most of the Linpack routines into Comal. You do have to be careful to change the "stride" increment since Fortran stores arrays by columns and Comal stores arrays by rows. Isn't Comal wonderful!

BTW, Nick Higham quit using his C64 soon after he released his BLAS package. He is one of my favorite numerical analysts. He has published many technical papers and I have never seen one that I did not find interesting or useful.

The most ambitious program that I wrote for this old 1 Mhz 8 bit computer using Comal involved 3D

CFD, Computational Fluid Dynamics! Specifically, I wrote a surface source program to calculate the potential flow about a body of revolution. Technically, this is an early 1970's method running on an early 80's computer, programmed in the early 90's. The problem involves breaking the meridian line into elements, assuming a unit source flow strength, and integrating the effect of the source element on each control point, the center of each element. This will produce two square arrays for two orthogonal flow velocity vectors. We use the boundary condition that the flow must be parallel to the body surface. This allows us to solve one linear system for the actual source strengths that satisfy this condition. Multiplying this result by the other square matrix gives the tangential surface velocity that we are interested in. It is a bit more complicated than I have describe here, but it does involve solving a linear system of equations that could be large.

In the course of research for this program I found a useful program for solving linear systems on small memory computers called Quartersolve and I rescued it from antiquity. Early computers also had small memories, similar to the C64. Contemporary subroutines for solving linear systems, such as the Linpack routines, require direct access to the full n by n matrix in main memory. Of course there are also other methods for solving large systems using external storage, but that is not a good idea for the C64 and its slow disk drives. Quartersolve works by processing the problem a row at a time, ideally right after each row of the matrix is calculated. It continuously compacts the working memory that it needs in such a way that at maximum it need only about 1/4 as much memory as most algorithms. Looking at it another way, with a fixed amount of memory available, Quartersolve can double n and solve problems twice as large as the Linpack routines. Of course I updated Quartersolve to use the BLAS.

The program was finished and worked fine. I ran many test cases. However, I had written the program for accuracy and cleverness, rather than speed. I had many ideas for speeding it up, but I concluded that it would never be fast enough and I abandoned it. What I had was a solution to the direct problem and what I was really interested in was solving the inverse problem. That is, specify the surface velocity and other boundary conditions and compute the shape that will produce it. This is a numerical optimization problem where my true interests lie. By specifying appropriate surface velocities it is possible to design low drag shapes. I did not get exactly where I wanted to go, but I did develop an impressive program on the C64 that I could not have written without Comal.

Edited 19-June-2001