The Convex C-1 Computer
SEP's Financial Event of the Decade

Jon Claerbout, Joe Dellinger, Bill Harlan,
Stew Levin, Rick Ottolini, and Chuck Sword

ABSTRACT

SEP has ordered a Convex C-1 computer. This is the largest financial event in the history of SEP. The C-1 integrates general purpose computing with array processing. The order was placed August 15, 1985 and we anticipate delivery September 11, one day after the production deadline for this report.

REASONS FOR SELECTING THE CONVEX (Jon)

Since about 1977 we have had an FPS AP120B array processor attached to various DEC computers (11-34, 11-70, and since 1980 a VAX 11-780) running UNIX (trademark of AT&T). This has worked well, giving us computer power similar to many of our sponsoring groups. The Convex C-1 computer integrates general purpose computing with array processor floating point power. We anticipate the following advantages of the C-1:

1. Array processor power is now extended to all our experimental work. (Formerly concepts had to be thoroughly tested before people would undertake the chore of coding for the Array Processor).
2. Since the Convex encourages Fortran, our code will be more portable hence of more likely value to sponsors.
3. Likewise our code will be more permanent. Programs are more likely to be of value to SEP staffers after they leave SEP.
4. Programmer involvement in memory management will be greatly reduced.
5 Startup time for new students will be reduced.
6. Maintenance responsibility formerly divided among three groups (and ultimately us) will be consolidated in the Convex company. SEP's responsibility for UNIX-to-AP interface will disappear. SEP's responsibility for UNIX maintenance will greatly diminish.

Areas of concern

1. Convex is a new company. We will get serial number 25. (The first 10-15 were probably internal). We hope the company remains viable.
2. Convex is not supporting all of the applications software that usually accompanies UNIX. These include troff/TEX typesetters, Pascal, Lisp and Macsyma. We will have to continue some software maintenance ourselves.
3. The Convex I/O bus, Multibus, is the third most widely used after those of IBM and DEC. However, many graphics and other peripheral device companies do not support this standard.

PROBLEMS DURING TESTING

A program archive tape was sent to Convex's San Jose computer. Testing was done by dial-up from Stanford. Testing proceeded at a leisurely pace over about a three week period after which we decided to purchase a C-1. Our test account has not been disconnected and sporadic testing continues.

Library problem (Stew)

This is about a problem at the Fortran to C language interface that Convex has promised to repair.

One of the early difficulties was to make Bill's Stolt migration program, a C main program plus a mixture of Fortran and C subroutines, compile, link, and run. Bill worked out the small differences between our Unix Fortran and Convex's and got it to compile. As on the VAX, linking C and Fortran proved to be difficult and the incantation we used on the VAX failed because some additional Convex Fortran libraries were needed. Rick and Bill eventually tracked down missing references. Running the program proved hardest; it consistently crashed inside the C library routine "atof" which converts an alphanumeric string to a floating point number. The debugger proved to be of no use on the combination of C, Fortran, and assembler routines. Finally Joe was able to produce a minimal example of the bug which appeared when the Fortran command was used to do the link and disappeared when the C command was used. Stew correctly guessed that "atof" was being brought in from the wrong place and found a Fortran library containing a routine of that name. Communication with Convex in Dallas revealed that their Fortran developers were using a "souped-up" version of the routine with additional, error-checking arguments. Unfortunately they hadn't changed the name to protect the innocent. They promised to fix this conflict as well as a few others in an imminent Fortran update.

Double precision problem (Bill)

There is a subtle pitfall lurking on the interface between Fortran and C. One of the benchmark programs, a Stolt migration program, gave correct answers on the VAX and wrong answers on the Convex. However, both machines followed the C syntax exactly. The pitfall is present in VAX code, but because of the arrangement of VAX's double-precision bytes, the errors have no consequences. The problem is likely to arise again, so take note.

All floating point constants passed from one C subroutine to another are promoted to double precision, no matter how one may have declared them. This conversion becomes inconvenient when calling Fortran subroutines that expect single precision numbers. On the VAX it is possible to interchange single and double precision floating numbers as subroutine arguments because the first half of a double precision number is the same as a single precision number. However, precision is not interchangeable in a CONVEX: a Fortran subroutine receiving a double precision number will drop the most important bytes. We outline the bug as it appeared and as it is likely to reappear.

First a C subroutine was called from a C main program. A single precision floating-point argument was implicitly converted to double, as are all floating-point constants. Next, the converted constant was used as an argument in a call to Fortran, now expecting single precision. The VAX, because of the arrangement of its floating-point bytes, merely truncated unimportant digits. The VAX convention, however is the reverse of the Convex and most other machines, including IBM (Convex follows the IEEE standard). The first four of eight bytes are reversed with the last four. The Convex truncated the four most important bytes of eight and passed a meaningless number onward. By the strict definitions of the C subroutine calls, Convex had followed the syntax correctly. The difficulty lay in that most users are not aware that the C language performs such hidden conversions. The interface between C and Fortran is sensitive to incompatible variable types and so gives unpredictable, machine-dependent results.

BENCHMARKS AND TESTS

Benchmarks fell into three categories. We first wished to determine how useful the Convex vectorizing ability would be to users not yet prepared to alter working programs for the Convex compiler, and to see what changes might be demanded for more efficient code. We then sought to find the absolute limits of Convex efficiency and inefficiency. Lastly we attempted to port large amounts of already written software to check compatibility.

The core seismic processes (Bill)

Six Fortran programs were selected as typical of SEP calculations, particularly those often repeated for large amounts of data. All were found to be limited in run time by computations, not input and output. These programs included Stolt (f/k) migration with sinc interpolation, Gazdag (phase shift) migration, whitening deconvolution (prediction-error filtering, PEF), the fast-Fourier transform (FFT), hyperbolic moveout correction (normal moveout, NMO), and NMO velocity stacks. One can find discussion of these procedures in Jon Claerbout's *Imaging the Earth's Interior:* Stolt, p.33 (SEP-30, p.103); Gazdag, p. 30; NMO and NMO stacks, p.192 (SEP-42, p.119); and in his *Fundamentals of Geophysical Data Processing:* FFT, p. 9; PEF, p. 131.

The programs covered a variety of computations in loops of varying complexity, chosen naively, without regard to how easily the Convex might vectorize the most important loops. We wished to know foremost how useful the Convex might be to those who would prepare programs without concerning themselves with the inner workings of the Convex. New members of SEP, visitors, and students of geophysics courses would fit this category, as would most of those preparing and debugging new programs.

The programs stressed various types of computation. The Gazdag migration emphasized complex exponentiation and multiplication, with large loops over a three dimensional array. The prediction error filter consisted almost entirely of convolutions and cross correlations; a small matrix inversion took little time. The Stolt migration was limited by its sinc interpolation (SEP-30, p.103). This frequency-domain interpolation looped over a fixed number of samples (14), using trigonometric functions and performing many multiplications and additions.

The FFT used the original fast method of doubling. This algorithm is the fastest one available for a scalar machine such as the VAX, but as mentioned elsewhere in this paper, other methods proved more vectorizable. This FFT prepared a lookup table so that little time was spent in preparing Fourier coefficients. The innermost loop contained only multiplications and additions, but changed its length constantly by factors of two.

The basic NMO program, prepared by Jon Claerbout, is similar in most respects to the industry standard, which dominates geophysical computer processing. NMO is a re-gridding problem that requires interpolation and finding the zero of a hyperbolic equation. This program uses a Newton's iteration, dependent on multiplications and additions, rather than calculating an expensive square-root. The program also reverses the common direction of an interpolation, adding one point into several, rather than vice-versa. This procedure makes the calculation of adjoints more easily introduced. See SEP-42 p. 119 for the algorithm.

These programs were well-tested on the VAX and, at first, were not modified for the Convex compiler. Samples of data outputs were checked for maximums, minimums, and standard

deviations; plots were prepared in suspicious cases. All programs produced correct output on the Convex at first, except for the Stolt migration. This problem was discussed in the previous section as the double precision problem. Compilation and execution were correct, however, a certain pitfall had machine dependent consequences. The VAX, by chance, did not notice the error.

The original improvements in run speed were the following:

	Original	Revised
PEF:	25x	
Gazdag:	27x	
Stolt:	13x	
FFT:	6x	65x
NMO stack:	7x	9x
NMO stretch:	6x	

The improvements in run speed for the PEF and the Gazdag programs were very impressive for unmodified code. The increase for the Stolt was at the level we considered more than adequate. The FFT however was disappointing and led us to inquire into possible improvements. Convex programmers explained that the doubling algorithm, because of the variable length of its inner loop, was difficult to vectorize. They prepared an alternative, using the Singleton method, that ran 65 times faster on the Convex. The two NMO programs were also a disappointment. By reading extensive comments provided by the compiler, we were able to fairly quickly make modifications bringing the speeds up to 9x. The changes required the breaking up of loops into several, each containing fewer operations, but filling temporary arrays to be treated again. The Convex compiler was thereby able to avoid loops that were unnecessarily recursive. Convex programmers examined the NMO program and offered further suggestions. These suggestions indicated how familiarity with the compiler might later alter the programming style of those wanting their code to vectorize well.

Some extreme cases (Rick)

We tested various small pieces of code written with a vectorizing compiler in mind to push the maximum performance of the Convex. There are two measures of results: first, speed increase relative to the VAX and FPS array processor, and second, megaflop (adds and multiply) rate. In all tests optimizing FORTRAN compilers were used and FPS times include the I/O overhead to our host. Relative speed increases were amazing at times; the Singleton FFT was **65 times faster** than the VAX, matrix multiply 60 times faster, slant stacking 45 times faster. Normal moveout with cubic splines and recursive Newton square root was only 12 times faster and recursive fractal computation (see Claerbout, this report) 4 times faster. Speed increases were from 2

to 4 times faster than the FPS.

The megaflop rate increase for most seismic processing was not as spectacular, but the Convex seemed to attain a rather high speed consistently and with little special programming effort. Many algorithms seemed to get up to the 5 to 10 megaflop range, with a maximum of 16 megaflops observed (FFT). The 10 megaflop plateau is controlled by the speed of vector load/store from main memory. Therefore, we attribute the speed comparison not so much as the Convex improving, but that the VAX computes poorly for many algorithms.

It is easier writing high performance software on the Convex than the VAX or FPS. The vectorizing Convex FORTRAN compiler points out places where coding can be improved. I only found one case out of a dozen (fractals) where the compiler assembly language output could be improved by hand coding. I can improve the output of VAX-UNIX-Fortran compilation in the majority of cases.

SEP C-language libraries (Joe)

The SEP I/O library, representing some 100,000 lines of C code, converted correctly, and and would have run immediately, except for the library "atof" problem discussed by Stew.

SEP's in-house, vector-plotting graphics software was also ported to the Convex and seems to have run perfectly the first try.

Ethernet (Joe)

Joe traveled to Dallas and tested the ethernet connection between a Convex and a VAX. It worked, including the "remote login" feature. We expect to rely heavily on this link during our conversion period. We expect to continue to rely on it for communications and other general-purpose I/O.

Communications with other benchmarkers (Chuck and Jon)

It is not our place to report all we heard from other benchmarkers, although we will report that we communicated with Sohio who have a Convex and Texaco who also ran benchmarks.

Two well-known tests of computer performance are the LINPACK benchmarks and the Livermore loops (officially known as the Livermore Fortran Kernels). The LINPACK benchmarks test how quickly a computer can solve the set of equations $\mathbf{Ax}=\mathbf{b}$, where $\mathbf{A}$ is a 100 by 100 matrix. The Livermore loops are designed to test how well a Fortran compiler can vectorize ordinary code.

The LINPACK benchmarks have been run on a wide variety of machines, making it possible to compare the Convex's speed with that of other computers. According to these results, the

Convex has about 1/5 the speed of a Cray-1S (the ratio is 1/8 if the Cray and Convex programs are hand-coded in assembly language). The following results are based on the double-precision version of the benchmarks, run with compiled (not hand-coded) linear algebra subroutines. Machines comparable to the Convex (from 75% to 125% as fast) include the CDC Cyber 760 (slightly faster than the Convex), and the IBM 370/195, the IBM 3081 K, the CDC Cyber 175, and the Alliant FX/4 (all somewhat slower). The Alliant is a new machine, and its performance may improve as bugs are worked out of the optimizing compiler. Machines significantly faster than the Convex include the Cybers 176, 875, and 205, the CDC 7600, the Amdahl 5860 HSFPF, the Honeywell DPS90, the IBM 3090 Model 200, and various Fujitsu, Hitachi, and Cray super computers. Significantly slower machines include the ELXSI 6400, the FPS 164, and the VAX 8600. It should be noted that this benchmark may take no advantage of parallel processors, and that other benchmarks can and do give different results. The coordinator of the LINPACK benchmarks is Jack J. Dongarra of Argonne National Laboratory, (312) 972-7246.

The Livermore loops have also been run on a wide variety of machines, but we do not have most of the results. They show the Convex running about twice as fast, overall, as the FPS-164, and about 1/6 the speed of the Cray 1S. However, the Livermore loops are not intended to be a speed test, but rather a test of the ability of the Fortran compiler to vectorize ordinary Fortran code. Some of the loops were written in a way that they could be easily vectorized, but others were written as if by a naive user who knows more about science than about computer programming or vectorization. Out of the 24 loops, the Convex was able to vectorize 17, tying for first place with the Fujitsu. The Cray 1 was able to vectorize 10 of the loops, and the CDC Cyber 205 succeeded in vectorizing only 8. I have no results for other machines. The author and coordinator of the Livermore loops is Frank McMahon of the Lawrence Livermore National Laboratory, (415) 422-1647.

In the August, 1985 issue of First Break, Les Hatton reported the results of some benchmarks that he ran on various machines, including the Convex. Only the various Crays ran faster than the Convex C-1.

MOVIE PORTING PROBLEM (Rick and Chuck)

The seismic movie making software is extensively used at SEP at a rate of 10,000 movies generated a year. The ability to transfer this software was an important consideration in changing to another computer. A very important component of the movie software is high speed data transfer ($>$ .1MB / second) between host computer and a graphics terminal. Our graphics terminal company (AED) does not support DMA transfers on the Convex Multibus or DR11 emulator. We are considering several alternatives: (1) put up the movie program on a terminal (Raster Graphics) which supports the Convex bus, and sell our AEDs; (2) keep our VAX 780 until a

smaller UNIX-unibus computer (microVAX ?) can be found; (3) keep our VAX 780 until AED supports the Multibus or DR11 style bus. At the time this report goes to press, a device and computer independent version of the Movie software (in C and FORTRAN) is operational on the VAX-Raster Tech configuration, and it seems that option (1) may be implemented.

We expect a 5 times all-around performance increase in movie software on the Convex (but will settle for VAX capabilities). The Convex Multibus is 5 times faster than the VAX. Core-to-core data transfers are 5 times faster and the maximum random access movie size will increase from about 5 MB to 40 MB.

Potentially, the Convex could increase movie performance 100 times over the VAX, but this requires special hardware. The Convex I/O processor is rated at 40 megabytes, or a hundred times faster than the VAX. However, few, if any off-the-shelf graphics terminals accept data faster than a megabyte per second.

FINANCIAL (Jon)

The cost is somewhat over a half million dollars. We carried forward about 100k from 1983-84 and anticipated reserves of about 350k at the end of 1984-85. At the May 1985 sponsor meeting, sponsors deemed it inadvisable to raise the membership fee. Because of salary inflation and increases in the number of staff, to consider purchase at this time we had to plan for a deficit of about 100k for the 1985-86 year. The university has agreed to carry this deficit, thus in effect giving us an interest free loan. With some luck we may be able to sell the VAX and AP system to avoid to deficit. But before the VAX sale can proceed, we need to solve the Movie porting problem.