3) The Campus Facility is located more than a quarter mile from the Medical School. Although this distance may sound small, it is a physical obstacle which does present potential interaction problems, despite the terminal service available;

4) Current disk rates and the ORVYL file system will make disk storage more expensive for medical users that the current ACME rates. See the subsequent section on Cost Considerations;

5) A large facility serving several thousand users may have less flexibility in terms of changing its systems than a smaller facility serving approximately 300 user projects;

6) ACME users would have to translate their programs to the form of interactive PL/1 mounted on the Campus Facility. Hopefully the conversion cost can be minimized with translation aids prepared by the facility.

## Comments on Cost Comparisons between Campus and ACME Services:

Cost comparisons have been difficult to draw between Campus and ACME Facilities due to the disparate nature of the facilities and the accounting algorithms. Three of the FORTRAN programs used in the ORVYL capacity study were translated into PL/ACME. The three were of different types: A matrix multiplier which is heavily compute-bound, a file writing program, and a psuedo parser which is primarily a string manipulator with a great deal of 2741 output. Considering only the charges for CPU time and terminal access time under both systems, the compute-bound job would cost 42% less on Campus Facility than on ACME assuming that ACME is used during hours of heavy activity. The other two programs (which were heavily dependent on output speeds) provided roughly equal costs at Campus and ACME. These cost statements assume that ACME is charged at 2-1/2¢ per pageminute plus $5.75 per terminal hour on the old 8 microsecond bulk core. The Campus rates were the standard $10 per CPU minute on ORVYL, plus $3.50 per terminal hour. One factor which is difficult to evaluate is the extent to which the CPU plus terminal access at Campus reflects the total cost of the job. Averaging all user charges for fiscal year 1971 at the Campus Facility, a total of 68% was derived from CPU usage and terminal access. This means that another 35% of the income was derived from disk storage, printing, card punching, card reading, off-line plotting, and use of the WYLBUR text editor. The point is that 32% of the income comes from sources other than those used in the above comparison.

At ACME the charges to users other than pageminutes and terminal access cover disk storage and terminal rental service. ACME's terminal rental service includes an add-on to cover general services to the community whereas the Campus Facility terminal rental rate covers only costs. The rate is $135 per month plus $4 to $13 for telephone lines on the Campus Facility versus $225 per month on ACME for a private 2741 terminal. The disk storage rate at ACME is 1¢ per track per day versus Campus Facility rate recently announced of 2¢ per track per day. Since ACME has moved to a faster bulk core, it can now provide more computing per dollar than the old 2 1/2¢ per pageminute rate permitted; the effect of the new core will vary by types of use. Cycle intensive users will use 50% to 70% of the pageminute usage encountered with the slower bulk core. Data input via terminals and program development will require about the same number of pageminutes on the new core.

The cost comparisons are difficult to make. It is clear that compute-bound jobs can be executed more efficiently on the Campus Facility than on the ACME configuration. I/O bound jobs tend to run at roughly comparable costs on each facility. On the basis of these findings, it appears that short term economics should not be the basis for any decision with respect to merger or lack of merger.

Dist: Staff/All

## APPENDIX B

ACME System Core Timing Results

(AMPEX vs. IBM)

ACME Note

WCTR-1
Regina Frey
November 29, 1971

# ACME System Core Timing Results (AMPEX vs. IBM)

## Introduction:

A set of programs were collected for the purpose of measuring the effect on the ACME system of substituting AMPEX 2.5 microsecond core for IBM's 8 microsecond Large Capacity Storage.

Essentially, these programs were selected to answer three questions:

1. Whether the AMPEX core boxes do indeed have a 2.5 microsecond cycle time.
2. The performance improvement on a hopefully representative set of PL/ACME programs.
3. Whether the core substitution would have any effect on the maximum possible transmission rate between the 1800 and the 360.

## Comparative Core Timings:

An OS batch program was written in assembler language which measured the true cycle times of the IBM 2050 'fast' core, the IBM 2361 LCS, and the AMPEX core boxes. The time required in each case to perform a series of fixed and floating point operations was measured.

The results are given in Table 1. The program contains four loops. The first measures true cycle time; the last three time loops containing fixed point arithmetic, short floating point arithmetic, and long floating point arithmetic instructions.

That each core box performed according to specification was proved in the first operation. The operation consisted of little more than ten million STore instructions. Each ST requires 2 core cycles for completion. Thus if the core cycle is 2 $\mu s$, then each ST will require 4 $\mu s$, ten million of them require $4 \times 10 \times 10^6$ $\mu s$ or 40 seconds.

Table 1. Core Timing Results (Seconds)

| Core / Operation | IBM 2050 (Fast Core) | AMPEX (Bulk Core) | IBM 2361 (Bulk Core) | AMPEX 2361 (%) | 2050 AMPEX (%) |
|---|---|---|---|---|---|
| Fixed point STore | 40.29 | 50.46 | 161.65 | 31 | 80 |
| Fixed point arithmetic | 111.10 | 121.11 | 220.17 | 55 | 91 |
| Short floating arithmetic | 118.14 | 128.15 | 214.36 | 60 | 92 |
| Long floating arithmetic | 287.16 | 302.19 | 440.25 | 69 | 95 |

Notes on Table 1:

1. The AMPEX times are the averages of two runs.
2. All figures include overhead time for instruction loop control and fetch/conversion of the interval timer.
3. Time intervals were calculated to the nearest one-hundredth of a second. The TIME SVC was used to obtain the contents of the hardware timer.
4. The operations performed in each loop were:
   a) Fixed point store: $10 \times 10^6$ ST instructions.
   b) Fixed point arithmetic: $2 \times 10^6$ sequences of L, M, A, S, D.
   c) Short floating arithmetic: $2 \times 10^6$ sequences of LE, ME, AE, SE, DE.
   d) Long floating arithmetic: $2 \times 10^6$ sequences of LD, MD, AD, SD, DD.
5. No index register was used in any of the instructions.

## Representative PL/ACME Programs:

Seven PL/ACME programs were run with the IBM 2361 core and the AMPEX core. All of these programs were written by the ACME staff.

All runs were executed in single user mode. Module TEN1, written on 11/4 and 11/11, was used. The differences between the two versions of the module do not affect the results.

The program attributes are given below; the timing results are in Table 2. The performance improvement was calculated as the difference between IBM time and AMPEX time over the original IBM time.

| Program | Attributes |
|---|---|
| CONVERGE | PDP-11 assembler. No terminal output. Moderate disk file I/O. Heavy string operations. |
| CM1 | Matrix multiplier. No I/O operations, no strings. Totally compute-bound. |
| iotime | Disk file read and write sequential operations. Time was measured for reading and writing 100 records of one block each. The loop was repeated 10 times each run. Little computation. No terminal output. |
| lisptest | LISP test program. Light file and terminal activity. The program did not complete due to an error in the LISP garbage collector. Time was measured by eyeballing the machine room clock. |
| PLA_PL1 | The PL/ACME to PL/1 translator. Heavy file and string activity. The program produced a page of terminal output and required four one-character terminal responses. |
| PLOT1 | TV plotting program. No file or terminal activity. Light mathematical calculations. Heavily dependent upon the speed of the display. |
| randfile | Random disk file operations. Light computations; no terminal output. The program consists of three sequences: Write direct of 20 records with randomly assigned keys, each record 1200 words long; Read sequentially each of the randomly written records; Read randomly the 20 records 5 times for a total of 100 reads. |

Table 2. PL/ACME Program Timing Results

| Program \ Core Perform. | IBM 2361 | AMPEX | Performance Improvement (%) |
|---|---|---|---|
| CONVERSE | 5.7 min. | 3.2 min. | 46.9 |
| CM1 | 6.1 min. | 2.5 min. | 59.0 |
| lisptest | 4.75 min. | 3.0 min. | 36.8 |
| PLA_PL1 | 23.8 min. | 17.4 min. | 24.8 |
| PLOT1 | 4.9 min. | 4.0 min. | 18.1 |
| iotime: | | | |
| 100 reads | 13.2 sec | 9.5 sec | 28.0 |
| 100 writes | 19.8 sec | 14.4 sec | 27.3 |
| randfile: | | | |
| 20 write direct | 23.0 sec | 15.8 sec | 31.3 |
| 20 read seq. | 71.7 sec | 34.2 sec | 52.3 |
| 100 read direct | 49.1 sec | 29.1 sec | 40.7 |
| | | Average | 36.5 |

The rather unexpected gains in the execution speeds of 'iotime' and 'randfile' require discussion. While they were included primarily for determining a base against which to measure the impending AMPEX disk drives, the results of this test provide some interesting information regarding the nature of the ACME file system.

Unfortunately, 'iotime' was executed only once with the IBM core, so while the times listed are the averages for 1000 operations (100 read/writes repeated 10 times), they may not in fact represent true average results. This statement is based on two facts: The times for the 10 repetitions did not vary greatly, but the differences between the AMPEX runs were considerable.

'iotime' was executed 4 times with the AMPEX core. The time required for 100 reads varied from 7.8 seconds to 14.7 seconds. For 100 writes, the variance was from 13 seconds to 17.4 seconds.

These broad ranges are due to the variability of the required disk head seek time (from near zero to a maximum of 130 milliseconds.)

Because of the ascendingly ordered nature of the ACME Space List, the average seek time should not be 60 ms, but 25 ms (time to seek to next contiguous cylinder). Total I/O time for an average read is 25 ms plus 12.5 ms disk rotation time or 37.5 ms. For a write operation, an additional complete rotational period of 25 ms must be added for an average time of 62.5 ms.

Subtracting these figures from the average times for a single read or write, we get these software overhead values:

| | IBM | AMPEX |
|---|---|---|
| read | 132 | 95 |
| | -37.5 | - 37.5 |
| | 94.5 ms | 57.5 ms |
| write | 198 | 144 |
| | - 62.5 | - 62.5 |
| | 135.5 ms | 81.5 ms |

The values are somewhat inflated since the times required to update the index and to type the time on the terminal have not been subtracted. But clearly the unexpected savings while performing disk I/O have been explained, and equally clearly the file system software could be optimized.

'randfile' was executed twice with the IBM core, four times with the AMPEX core. All comments on 'iotime' apply as well to 'randfile' except that the results of the individual runs were not as variable.

The phenomenal savings on the read sequential operation can be explained by the fact that the record keys were distributed randomly throughout the file index and consequently a considerable amount of in-core index searching was necessary for finding the next sequential record.

## 1800 Communications:

Little improvement was expected in the 1800 communications transmission rate since the essential routines (EIGHTNØØ, RW18ØØ, YIELD) were already located in the 2.0 μs core.

To test the hypothesis, one of Lee Hundley's 1800 test routines (READALOT) was executed before and after the core switch. The purpose of the tests was to establish the conditions under which data overruns (data arriving too soon for the 360 to process) would occur.

READALOT accepts as parameters the number of 1800 input lines (N), the sample interval in milliseconds (TIME), and the number of 360 buffers assigned to each line (BUFS). Buffer size was fixed at the maximum permitted, 250 points. After experimentation, we decided to leave BUFS at its maximum value (20) and to vary only N and TIME.

The following information was gathered from the IBM 2361 test:

1. N=8 and TIME=1 (8000 points/sec) will crash the 1800.
2. N=7 and TIME=1 (7000 points/sec) will cause data overruns.
3. N=12 and TIME=2 (6000 points/sec) will not cause data overruns.

Running as a single user and performing no calculations on the collected data, the only significant overhead was the time required for YIELD to service the commutator (i.e., look for another user and ultimately return to the only active one). Therefore, to determine the approximate amount in excess of 6000 points/sec at which overruns would occur, we inserted DELAY(0) statements in the program following each CALL READ (One DELAY(0) forces one yield to the commutator and wastes one time slice). With N=12 and TIME=2, two DELAY(0) statements resulted in overruns.

The same program, executed in AMPEX core, gave these results:

1. N=7 and TIME=1 (7000 points/sec) will not result in overruns.
2. N=7 and TIME=1 with two DELAY(0) statements will result in overruns.

Thus the maximum possible data rate was increased by approximately 1000 points per second or roughly 15%.

The second experiment was concerned with the question of whether EIGHTNØØ could execute properly in 2.5 μs core. About six months ago, EIGHTNØØ was moved from the 8 μs core to 2 μs core when it was discovered that the channel commands contained within EIGHTNØØ could not be decoded fast enough and channel data chaining checks occurred. This condition prevailed only if the buffer size was extremely small.

A special link edit off TEN1 placed EIGHTNØØ in 2.5 μs core. Tests on this module gave these results:

1. A buffer size of one no longer caused data chaining checks, but overruns consistently occurred, even at low data rates. At 1000 points/sec on the line, the 1800 will crash.
2. A buffer size of two points or greater will execute properly.

However, it is not recommended that EIGHTNØØ be moved out of our fast core. AMPEX performance at its best is still 80% of the maximum on a Model 50. Moving EIGHTNØØ would decrease our maximum possible transmission rate (has not been tested).

Summary:

It has been confirmed that the AMPEX core does run with a 2.5 microsecond cycle time. Fixed point operations execute 2 to 3 times faster; floating point operations roughly 1 1/2 times faster than in 8 microsecond core.

Total time savings is heavily dependent upon the nature of a PL/ACME program. Terminal-bound or otherwise I/O dependent programs will see little increase in execution speed. Compute-bound programs may execute as much as 2 1/2 times faster.

The time slice allocated to each executing program remains as before. Thus, terminal response time will in many cases be the same, but since more computation is possible within each time slice, total execution time for a program will decrease. A sample of PL/ACME programs executes from 18 to 60% faster. The average was 36.5%.

While some improvement is realized for disk file operations, the gain may be unnoticed due to the yield to the commutator (other users) at the start of each I/O operation and the resultant wait by a user until his turn again arrives.

The maximum 1800/360 transmission rate has increased by about 15%. A greater increase was not expected since the 360 communications program had already been located in 2 microsecond core.

Dist: Prog/All/D. Phillips/C. Dickens

## APPENDIX C

### Results of Three Compaction Algorithms

The test was performed during a File System Analyzer run on all user data sets, February 28, 1972.

CODINGS USED BY EACH COMPRESSION ALGORITHM TO BUILT BITSTRING*

| | COMPAC1 | COMPAC2 | COMPAC3 |
|---|---|---|---|
| WORD IS ZERO | 00 | 00 | - |
| WORD IS UNDEFINED | 01 | 01 | - |
| WORD HAS VALUE | 1 | 10 | 0 |
| WORD IS REPEAT OF LAST WORD | - | 11 | 1 |

TOTAL NON-NUMERIC WORDS = 16,532,183

TOTAL EMPTY (zero) WORDS = 5,184,105

The resultant savings from the application of each algorithm is stated below. Percentage figures were computed as the total storage requirement after application of the algorithm over the current storage requirement. TOTAL WORDS WHEN LESS refers to the inclusion of a file only when its 'compacted' size is less than the original size.

| | | |
|---|---|---|
| TOTAL WORDS IN ALL ACME NUMERIC DATA FILES | 7,783,794 | |
| TOTAL WORDS USING COMPAC1 ON ALL FILES | 4,395,400 | ( 56%) |
| TOTAL WORDS USING COMPAC1 ONLY WHEN LESS | 4,314,484 | ( 55%) |
| TOTAL WORDS USING COMPAC2 ON ALL FILES | 3,661,950 | ( 47%) |
| TOTAL WORDS USING COMPAC2 ONLY WHEN LESS | 3,612,031 | ( 46%) |
| TOTAL WORDS USING COMPAC3 ON ALL FILES | 3,818,520 | ( 49%) |
| TOTAL WORDS USING COMPAC3 ONLY WHEN LESS | 3,783,314 | ( 48%) |

*The bitstring describes the characteristics of the values in a numeric array. It is stored on disk along with sufficient data to reconstruct the array. Repeated, undefined, or zero data is omitted from storage.

APPENDIX D
ACME 360/50
COMPARISON OF MEAN TIME BETWEEN FAILURES
JULY 17, 1969 - APRIL 30, 1972
(IN HOURS)

HARDWARE

| | Aug | Sept | Oct | Nov | Dec | Jan | Feb | Mar | Apr | May | Jun | Jul | Annual Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1969-1970: | 113.3 | 55.3 | 25.2 | 31.3 | 56.0 | 167.3 | 47.4 | 39.5 | 58.7 | 59.6 | 83.0 | 34.5 | 64.3 |
| 1970-1971: | 72.4 | 34.7 | 176.0 | 362.0 | 704.0 | 104.0 | 218.7 | 364.0 | 182.0 | 78.2 | 103.4 | 176.0 | 214.8 |
| 1971-1972: | 244.7 | 178.0 | 368.0 | 39.4 | 64.2 | 60.3 | 56.2 | 143.8 | 175.0 | 147.7* | 147.7* | 147.7* | 147.7* |

ALL FAILURES INCL. HARDWARE

| | Aug | Sept | Oct | Nov | Dec | Jan | Feb | Mar | Apr | May | Jun | Jul | Annual Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1969-1970: | 40.0 | 33.2 | 17.7 | 26.5 | 48.0 | 44.0 | 39.0 | 25.2 | 27.0 | 29.8 | 60.4 | 22.6 | 34.4 |
| 1970-1971: | 29.0 | 24.3 | 54.1 | 181.0 | 234.7 | 38.3 | 54.7 | 80.9 | 66.2 | 37.1 | 80.4 | 88.0 | 80.7 |
| 1971-1972: | 146.8 | 142.4 | 73.6 | 33.8 | 54.6 | 40.2 | 52.0 | 79.9 | 116.7 | 82.2* | 82.2* | 82.2* | 82.2* |

Underlined Figures = Best mean time to failure as compared to same period of each year.

* (May - July, 1972) Projected mean time to failure based upon first nine months total of August 1971 through April 1972.

## APPENDIX E

Proposal for Small Computer Service by ACME

ACME Note

PSCS-1
Bob Stainton
March 21, 1972

Proposal for Small Computer Service by ACME

The rapid decline in the price of mini-computers has led to an increase in the number of these computers installed at the Medical Center. This memo describes the services which could be provided to these computers by the ACME facility.

The service is summarized as follows:

1. Spooling service--
   A. Accept data - 20 hours a day
   B. Send data (or processing modules) - 20 hours a day

2. On line service (while the user is actually logged on to ACME)--
   A. Retrieve data from the spool
   B. Write data (or processing modules) to the spool for later use by the mini-computer
   C. Read and write directly to the mini-computer

3. All services of an ACME typewriter terminal.

4. High level language processors.

## Spooling Service

Spooling (Simultaneous Peripheral Operation) refers to a procedure which has become popular with large-scale third generation computers. It is exemplified by a card reader which is always ready to accept cards, even though these cards may not be processed by a user program until several hours have passed.

Our early experience with 2741 terminals has shown that their usefulness takes a quantum jump as soon as a computer system is installed which is continuously available to "answer the telephone".

Our proposal is to provide a computer system which is always (20 hours a day) available to take data from the user's mini-computer. Our definition of mini-computer is any device which is able to conform to the requirements of the communications protocol. The data will be held on a special disk system, and retrieved when the user "logs-on" to the ACME system and runs a program. In this way a research scientist may set up an experiment which may run for several hours during the night. The next morning he may "log-on" and run a program which processes the data generated by that experiment.

Another service of the spooling system is the ability to provide storage for "processing modules" which can be called by the mini-computer. The relationship between the mini-computer and the ACME system in this instance can best be described by an analysis of the classic "pen tracking problem".

In order to draw a line by means of a "light pen", a tracking cross is displayed on the screen of a cathode ray tube at the location where the "pen" is pointing. As the "pen" moves, the cross is no longer in the center of vision of the pen, and a computer interrupt takes place. The computer now re-establishes the cross in a new location.

To accomplish this smoothly requires a fast response and a large amount of data transfer. The response must be fast, or the cross will appear to drag as through a viscous fluid. The constant re-writing of the display requires the transfer of a large amount of data.

The conventional solution to this problem would be to use a very high speed communications line between a large central processor and the remote screen. This would ask the communications line to do that which it is least able to do, namely, provide fast response with large volumes of data.

A better solution is to employ a local mini-computer at the screen which produces the rapid response necessary to accomplish the "pen tracking". In this solution, a low speed communications line is used, and only the location of the tracking cross is sent to the large central computer. This information may be saved for later analysis or processed for a real time response.

In the medical research environment, the local mini-computer would contain that part of the software necessary to provide immediate reaction to the stimulus of the experiment. This software is contained in a replaceable "processing module" which also directs the sending of partially processed data to the ACME system for storage and statistical computation.

The replaceable "processing module" may be changed for individual experiments, or a new module may be loaded dynamically as the life of an experiment progresses.

These "processing modules" may be compiled in the ACME system and saved on the spooling system for later call by the mini-computer. A list of experimental parameters may also be stored on the spool and played back as data over an extended period of time.

## On Line Service

The ACME user would be able to access data which has been previously placed on the spool by using a special "OPEN FILE" statement. The data may then be accessed with "READ FILE" statements as a sequential data file. Similar "WRITE FILE" statements may be used to write data (or processing modules) to the spool system for later use by the mini-computer.

An option would also be provided to bypass the spooling system and communicate directly with a running ACME program, using the current directory mechanism. This allows the same real time interactive service which has been available in the past.

## Service of an ACME Terminal

As a convenience, it is realized that the "teletype" console of a mini-computer should be able to act as if it were an ACME terminal, so that the user need not use a separate 2741 terminal in order to use the ACME system. The user would also be able to "log-on" and run programs automatically, under control of a program in his mini-computer.

## High Level Language Processors

We will attempt to provide a hardware-independent language for writing "processing modules". Translators for this language to produce object code for the most popular mini-computers would be produced. This would provide an ease of programming not found on stand-alone systems, as well as a medium for sharing programs among individual researchers.

## Implementation

Implementation would require several loosely related projects:

1. The spooling system, housed in a separate mini-computer so that it can operate during maintenance hours for the large central computer.
2. A switching system, for attaching a given user to an ACME port, a real time data port, or the spooling system. As a by-product, we hope to allow any mini-computer in the system to communicate with any other mini in the system.
3. A communication system to transport the data and verify its correct receipt, with as little effort on the part of the mini-computer program as possible. We hope to provide a parallel, demand-response interface to the mini-computer, which appears to its program as a device similar to a tape drive.
4. High-level processors to provide an easy way of writing programs for the mini-computer.

Dist: Staff/All