Paul E. Schmid
Code 932
NASA - GSFC

Microprocessor Utilization in Search & Rescue Missions

FINAL REPORT

Introduction:

The position of an emergency transmitter may be determined by measuring the Doppler shift of the distress signal as received by an orbiting satellite. This requires the computation of an initial estimate and refinement of this estimate through an iterative, nonlinear, least-squares estimation.

A version of the above algorithm was implemented at Goddard Space Flight Center (GSFC) and tested by locating a transmitter on the premises and obtaining observations from a satellite. The computer used was an IBM 360/95. The position was determined within the desired 10 km radius accuracy.

The purpose of this project is to determine the feasibility of performing the same task in real time using microprocessor technology. The least square algorithm was implemented on an Intel 8080 microprocessor and the same experiment was run as at GSFC.

The results indicate that a microprocessor can easily match the IBM implementation in accuracy and be performed inside the time limitations set.

Columbia University in the City of New York

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE New York, N.Y. 10027

Mr. Paul Schmid
Goddard Space Flight Center
National Aeronautics
Computer Science & Space Administration
Greenbelt, MD. 20771

Why Microprocessors:

Time is an implicit restriction in any search and rescue mission. The use of satellites and computers is dictated by that time limit. The use of a big computer to determine the position presupposes communication between the satellite and the computer. This communication introduces a time delay since the satellite is not always within radio visibility of an installation that possesses both the communication and computing power for this problem. Furthermore the result has to be forwarded to a command center to do the dispatching.

Microprocessor utilization can alleviate this situation in two ways: by giving cheap computing power to communication facilities or by incorporating the computing power in the satellite itself thus eliminating this communication completely.

Microprocessors offer light weight, small volume, low power processing. Their speed is improving rapidly and their cost is going down. They are the logical choice for a satellite search and rescue system if they can perform.

Machine Configuration:

Strictly speaking there are three microprocessor configurations in this project which we are going to discuss individually.

- Development system
- Minimal execution system
- Actual field configuration

Initially our development system consisted of an MDS-80 Intellec microcomputer by Intel with 16k bytes of RAM memory and a resident ROM monitor. Most of the floating point package was developed in machine language on that system using the monitor's limited hexadecimal editor and debugger. The need for more sophistication became apparent. After several failures in exploring alternatives (as fancy as hooking up to a PDP 11 through a telephone line for more storage) we were able to acquire a dual floppy disk drive by Intel. A spare line printer was attached to the system with minor hardware modifications and 16k bytes more RAM were added in order to support DOS. The enhanced system had the power of a mini-computer in software (assembler, editor, library manager, linkage editor, leader, and a sufficient file manager) at a speed which was slow but acceptable. The floating point package was converted to assembly language, and two more packages were developed: the I/O package and the matrix manipulation package. Unexpected help came from the use of ICE-80 (In Circuit Emulator), designed for a different application, as a powerful symbolic debugger substituting for the monitor hexadecimal debugger.

Out of this final version of the development system only a limited amount of resources were used for the final run. Those define the minimal execution system. The disk was only used for input of data. The essential parts were:

- The CPU card
- 16k Bytes of memory

- The console device and its interface
- Power supply: 12V, 5V, -5V, ground

Additionally, the line printer was used to produce a hardcopy version of the results.

The actual field configuration would be the same if the machine were located on the ground. Some kind of communications equipment would be required to provide the data input and, maybe, start the run automatically. The configuration would be different, though, if the machine were located on the satellite. The requirements for the satellite configuration would be:

- The CPU card
- 16k bytes of memory
- An interface that can load the information in memory
- A means to communicate the result to the world
- Power supply: 12V, 5V, -5V, ground

The Floating Point Package:

Based on estimates of the number of operations required we were inclined to think that any floating point operations would have to be performed by hardware and not by software since estimated times became prohibitive. This floating point package was developed to help us count the actual number of operations rather than perform them in an actual situation. The final run proved our estimates wrong and the package gained new importance.

There are a number of representations of floating poing numbers differing in accuracy and range as a trade off to the number of bytes required per number. The one used was the ANSI format for FORTRAN which happens to be implemented by hardware as an option in IBM computers. It consists of one sign bit, a seven bit exponent (excess 64), and a 24 bit mantissa of hexadecimal digits. The accuracy is 6 hexadecimal digits or approximately 7.2 decimal digits. Specific operations were not timed although a more general timing analysis appears in a later section. This format was chosen as opposed to the BCD format because the space requirements are lower for the same amount of precision, which in turn reduces execution time slightly. A mantissa of binary digits was not used because of the frequent need for normalization.

Addition and subtraction take exactly the same time, whereas multiplication is approximately equal to 22 addition and division is approximately 60 additions.

Multiplication produces a 48 bit result mantissa which is then normalized and rounded to 24 bits. This preserved the number of significant digits, or, viewed from a different angle, is the same as a double precision multiply if the two arguments were expanded with zero fill.

Division preserves the significant digits again by expanding the mantissa of the dividend to double precision and results in full single precision result. Normalization and rounding occur as in multiplication.

Accuracy is thus preserved to true single precision throughout in a numerically stable manner keeping the length of the number to 4 bytes. The cost is expensive multiplication and, expeciallu, division. This dictates a programming style whereby division is avoided unless it is absolutely necessary. The benefits, on the other hand, are numerically stable implementations whose results match the double precision to the extent possible as will be seen when the results of the run are analyzed.

The square root function was implemented by using a variation of Heron's formula based on the observation that the mantissa of any floating point number will have a value of 1/16 to 1 (interpreted as a fraction). As a first guess an approximation to a straight line connecting the two end points is made. Experimentally, six iterations were found necessary to produce an accurate result. A better first guess could improve that significantly, but time constraints did not allow us to pursue that direction.

Finally, input and output of floating point numbers turn out a much more serious task than first expected. The input routine recognized numbers with a maximum of ten integer and ten fraction digits. This proved more than sufficient for our needs. The output routine produces a rigid scientific format with 10 fraction digits. When interpreting the results it should be kept in mind that at most only 7 are significant. The format was retained in case of future expansion of the mantissa. Both the input and output routines could be better, but since their function is only tangential to the project at hand they were kept on the bare functional level.

Matrix Operations:

All matrices in the system are defined as two-dimensional, including vectors. The first two bytes contain the number of rows and the number of columns in the particular matrix, respectively. This effectively limits the number of observations to 256. Vectors have one of their dimensions identically equal to 1. The next two bytes contain the address of the first byte that follows the last byte belonging to the matrix. Adjacent elements in a row of the matrix are stored as adjacent floating point numbers in memory. Rows are stored sequentially starting from the first row in the fifth byte.

In an effort to minimize the number of address calculations in the least squares algorithm the APL program we were supplied with, (LSQ), was converted into FORTRAN. The calculations involved in the residual equations were all grouped together inside one big loop. The advantage of such a scheme is that once an offset is calculated it can be used to address all the needed elements of the matrices involved in the calculation. When the time came though, to implement it using 8080 assembly language, it became all too apparent that there were too many addresses to keep track of and too few registers to help. Therefore, due to the limitation of addressing capabilities, routines were implemented for the various matrix operators in APL. This resulted in well-structured and very efficient code, the style being dictated by the instruction set.

A minimum number of matrix utility routines was necessary. Matrices can be created by specifying their dimensions, they can be filled with zeros, they can be read from a device, they can be moved (copied) in storage.

There are four classes of operations by which matrices may be altered involving the following arguments.

- a constant and a matrix
- a vector and a matrix

- two matrices (plus possibly a result matrix)
- one matrix (for example, inversion)

In our particular application there was only one inversion of a 2 by 2 matrix involved. A simple algorithm derived from Euler's method is implemented using fixed pivots. Execution time and temporary storage are optimized.

Implementing the Experiment:

Having developed the tools that were discussed in previous sections the actual implementation was straight forward. For reasons already mentioned a routine was written to match the LSQ routine* developed by Dr. Marini almost statement by statement. The correspondence is indicated in the source program by keeping track of the APL statement numbers. The array names were kept the same as much as possible and only one additional temporary matrix was required. The program was written for a maximum of 100 observations. All matrix operations as well as the square root keep track of the calls to the floating point routines.

The whole package makes limited use of two monitor routines, which can easily be eliminated. The reason they are there is because software was being developed in machine language and the monitor provided a lot of needed help. So, essentially, LSQ can be run completely independently.

The space requirements for this particular run was approximately 16k bytes, out of which 4k could be in ROM and 12k in RAM. The exact numbers are as follows:

Code: 3656 bytes

Data: 10365 bytes

Stack: 100 bytes (arbitrarily)

Total: 14121 bytes

Incorporated into the package were four counting routines that kept track of the number of additions, subtractions, multiplications and divisions required during each iteration. The results will be analyzed in the next section. The actual implementation would not require these routines. The counting overhead to each arithmetic operation is approximately equal to half the time of an addition.

---

* See Appendix C.

Interpreting the Results:

The final run converged and yielded five digits of accuracy. If convergence is defined as a ratio of two succesive RMS residuals being close to 1 (in absolute) it was attained at the ninth iteration to within 0.00001. Comparing these results to the run at GSFC (run at double precision, or 16 digits of accuracy) we note the 5 digit accuracy of our result.

Numerical analysis gives us enough tools to justify the loss of two significant digits in the course of the iterations. The main source of error appears to be the subtraction of the estimated range rates from the actuals. The subtraction of the average residual equations could could contribute to the error as well.

The measured execution time for this particular run was 62 seconds per iteration. The microprocessor used was an 8080A by Intel. Adjusting for counting the number of operations the true time becomes 61 seconds. The 8080A CPU has a cycle time of 2 microseconds. If this system were actually implemented, the 8080A-1 CPU could be used which offers higher speed with cycle time of 1.3 microseconds which could bring execution time down to 40 seconds for each iteration giving approximately 6 minutes to reach convergence. This figure is derived with no modification of the software. Since it falls within our difinition of "real time", which was around 15 minutes, it is definitely a workable solution.

Another alternative is, of course, to use hardware floating point units. Two units that we are familiar with indicate a disparity in execution times of several orders of magnitude. Their specifications appear in Appendix B for the purposes of the following analysis, 'typical' execution times for 8 digits of precision of the North Star Computers, Inc. FPB unit were used. Our system indicated the following frequency of floating point operations for each iteration:

Subtractions - 672

Multiplications - 2382

Divisions - 940

When trying to compute the time it would take to execute those instructions we noticed that the time it takes to access hardware floating point unit is more than twice than the time it takes to do the calculations. Namely, we came up with the following numbers:

TIME (SEC).	PURPOSE
0.35	perform the operations
0.825	input and output the number form FPB (8080A-1)
1.175	total time required

Therefore, use of hardware units make it possible to decrease the execution time by one order of magnitude.

Future Research:

The parameters that have to be optimized in the search and rescue mission consist of the accuracy of the position estimation and the time in which it is performed. Proving the feasibility of a microprocessor implementation is far from devising an optimal algorithm.

If the nonlinear regression method is utilized there is a lot of room for improvement in the inital estimate, a quantity that can affect the whole outcome of the iterations. Several methods that are suggested in Dr. Marini's paper can be explored. Furthermore, since the data collection takes an appreciable amount of time an algorithm should be devised in which an estimate is upgraded with each incoming datum. If that algorithm is good enough then the estimate could be the result itself.

A further enhancement on the calculation tiem can be achieved through parallelism. It can appear on two levels:

- The implementation of the least squares algorithm
- The grouping of data

The least squares algorithm may be broken into parallel subtasks that can be performed by different processors in parallel, especially floating point operations.

The data may be grouped in clusters on which the least squares algorithm is applied. The estimate provided by each cluster is then processed through least squares estimation itself. This method could be applied at data collection time too.

Appendix A

- Sample run at GSFC
- Sample run at Columbia

NASA sample run at GSFC

```
J0361  ¯5314.899178  3428.
  1 LSQ E
  IDUALS:  1.0507446 37
  ON, HGT ARE:  42.3067508
  ION IS:
  249245  ¯4693.396224  4270.874121
  BIAS
  5403993

  1 LSQ EN
  IDUALS:  0.1389538763
  ON, HGT ARE:  40.33701496  277.2125703  9.094947018E¯13
  ON IS:
  86014  ¯4830.001894  4106.660277
  BIAS
  5525

  1 LSQ EN
  IDUALS:  0.07064076362
  ON, HGT ARE:  39.76197471  279.9200369  9.094947018E¯13
  ION IS:
  35536  ¯4836.251765  4057.716223
  BIAS
  7161

  1 LSQ EN
  IDUALS:  0.06507914984
  ON, HGT ARE:  39.47765191  281.0740575  9.094947018E¯13
  ION IS:
  357  ¯4837.990348  4033.099884
  AS
  5854

  LSQ EN
  IDUALS:  0.06476401403
  , HGT ARE:  39.36049512  281.5358502  9.094947018E¯13
  ION IS:
  331  ¯4838.305495  4023.35154
  BIAS
  44962

  LSQ EN
  IDUALS:  0.06474873470
  , HGT ARE:  39.32402904  281.6773949  0
  ION IS:
  999  ¯4838.366335  4020.220503
  AS
  943

  LSQ EN
  IDUALS:  0.06474806885
  , HGT ARE:  39.31537952  281.7107637  0
  N IS:
  076  ¯4838.378943  4019.477604
  BIAS
  4357

  LSQ EN
  DUALS:  0.06474804019
  , HGT ARE:  39.31352503  281.7179045  0
  IS:
  575  ¯4838.381637  4019.318311
  AS
```

ORIGINAL PAGE IS
OF POOR QUALITY

LSQ EN

IDUALS: 0.06474804013

NGT ARE: 39.313502503 281.7179047

IS:

7575 74838.381687 4019.318311

AS

335

LSQ EN

DUALS: 0.06474803883

NGT ARE: 39.313003801 281.7197729

IS:

309 74838.382347 4019.276479

S

0102

LSQ EN

DUALS: 0.06474803883

NGT ARE: 39.313003448 281.7197924

IS:

306 74838.382353 4019.276179

S

0103

```
THE RESULTING POSITION IS.
X= -0.5534835815E+03   Y= -0.4828028678E+04   Z=  0.4393360137E+04

RMS RESIDUALS =  0.1154615402E+01

THE RESULTING POSITION IS:
X=  0.6216947555E+03   Y= -0.4912517547E+04   Z=  0.4176822662E+04

RMS RESIDUALS =  0.1050682067E+01

THE RESULTING POSITION IS:
X=  0.8464587211E+03   Y= -0.4839673935E+04   Z=  0.4060588836E+04

RMS RESIDUALS =  0.1389477729E+00

THE RESULTING POSITION IS:
X=  0.9470703125E+03   Y= -0.4838634496E+04   Z=  0.4033941268E+04

RMS RESIDUALS =  0.0706402111E+00

THE RESULTING POSITION IS:
X=  0.9875150680E+03   Y= -0.4838407516E+04   Z=  0.4023446083E+04

RMS RESIDUALS =  0.0650796318E+00

THE RESULTING POSITION IS:
X=  0.9999647140E+03   Y= -0.4838378906E+04   Z=  0.4020223154E+04

RMS RESIDUALS =  0.0647644424E+00

THE RESULTING POSITION IS:
X=  0.1002853393E+04   Y= -0.4838379859E+04   Z=  0.4019498825E+04

RMS RESIDUALS =  0.0647490596E+00

THE RESULTING POSITION IS:
X=  0.1001520965E+04   Y= -0.4838184628E+04   Z=  0.4019325256E+04

RMS RESIDUALS =  0.0647480487E+00

THE RESULTING POSITION IS:
X=  0.1003620147E+04   Y= -0.4838179959E+04   Z=  0.4019300456E+04

RMS RESIDUALS =  0.0647478675E+00

THE RESULTING POSITION IS:
X=  0.1002682136E+04   Y= -0.4838184628E+04   Z=  0.4019283294E+04

RMS RESIDUALS =  0.0647482872E+00

THE RESULTING POSITION IS:
X=  0.1002667831E+04   Y= -0.4838179859E+04   Z=  0.4019279892E+04

RMS RESIDUALS =  0.0647481444E+00
```

ORIGINAL PAGE IS
OF POOR QUALITY

Appendix B

Two typical hardware floating point units

- FPB by North Star Computers, Inc.
- FPU by Cyberuetic Micro Systems

FPB DATA SHEET

EXECUTION TIMES [1,2,3]

PRECISION DIGITS:		2	4	6	8	10	12	14
ADD	best	1	1	1	1	1	1	1
	typical	8	8	9	9	10	10	11
	worst	10	10	10	11	11	12	12
SUBTRACT	best	4	4	4	4	4	4	4
	typical	8	8	9	9	10	10	11
	worst	15	16	17	18	19	20	21
MULTIPLY	best	5	5	5	5	5	5	5
	typical	18	34	55	80	111	146	186
	worst	51	125	228	382	527	720	933
DIVIDE	best	7	7	7	7	7	7	7
	typical	39	70	109	156	211	274	370
	worst	62	139	229	340	470	621	779

1. Times given in microseconds
2. Execution times are a function of the input values
3. Times listed do not include transmission of input values and result

Board dimensions:

Model A: 5 in. by 10 in.
Model B: 6¾ in. by 12 in.

Power requirements:

Model A: 8 V (unregulated) @ 1.7 A
Model B: 5 V (regulated) @ 1.7 A

Board Construction:

FR4 material, gold plated edge connectors

Floating point number representation:

Byte 1: bit 7=sign (1=negative number, 0=positive number)
bits 6-0 = exponent in excess 64 binary representation
bits 7-0 = zero represents the zero value

Byte 2: bits 3-0 = least significant digit of value in BCD coding
bits 7-4 = next least significant digit of value

Byte n: bits 7-4 = most significant digit of value in BCD coding
bits 3-0 = next most significant digit of value

All values are nomalized.

Other representations of BCD floating point numbers require a change in microcode and are available on special order.

```
*Sample use of the North Star FPB for a divide operation with 8 digit precision
*In this example assume arguments are in memory in form:
*  Most significant byte (msb) digit pair
*  Susequent digit pairs follow the msb
*  Exponent/sign byte follows lsb digit pair.
*  Pointer addresses the exponent/sign byte
*BC has left arg pointer
*DE has right arg pointer
*HL has result pointer

*The FPB receives its arguments by "peeking" at the 8Ø8Ø bus
*when the argument values are loaded to accumulator.
*Two jumperable "hardwired" addresses are required for signaling the FPB

*This routine may be generalized to perform any operation, at any precision.

FDIV LDA RSTRT          This "hardwired" reference signals FPB to "wake up"
     MVI A,8*16+DIVOP   Specify precision and operation code to FPB
     LDAX D             Exponent/sign byte of right arg
     DCX D              Advance pointer to next byte
     LDAX D             Least significant digit pair of right arg
     DCX D              Advance pointer to next byte
     LDAX D
     DCX D
     LDAX D
     DCX D
     LDAX D             Most significant digit pair of right arg
     LDAX B             Exponent/sign byte of left arg
     DCX B
     LDAX B             Least significant digit pair of left arg
     DCX B
     LDAX B
     DCX B
     LDAX B
     DCX B
     LDAX B             Most significant digit pair of left arg
     Now the Floating Point Board is performing the operation
     LXI D,FPDIN        "Hardwired" address for receiving value from FPB
FDIV1 LDAX D            Loop waiting for completion signal (sign bit)
     ORA A              The FPB is done when the sign bit becomes "1"
     JP FDIV1           Loop if sign bit is still "Ø"
     ANI EBITS          Check for error, condition tested at end
     LDAX D             Exponent/sign of result
     MOV M,A            Store exponent/sign of result
     DCX H              Advance pointer.
     LDAX D             Least significant digit pair of result
     MOV M,A
     DCX H
     LDAX D
     MOV M,A
     DCX H
     LDAX D
     MOV M,A
     DCX H
     LDAX D             msb byte of result
     MOV M,A            Store it
     RZ                 Return if no error was detected
     JMP ERROR          Go report error (i.e. underflow or divide by Ø)
```

FLOATING POINT UNIT

PRICE LIST

MODEL	QUANTITY 1	25	100
#1	$595.00	$535.00	$475.00
#2	470.00	425.00	375.00
#3	345.00	315.00	275.00

All sales FOB Palo Alto

EXECUTION TIMES

FUNCTION	TIME IN MILISECONDS (approximate)
ADD, SUB	110
MUL, DIV, SQRT	225
TAN	846
LN, SIN, COS, →POL	1250
POWER	1720

CYBERNETIC MICRO SYSTEMS
2460 EMBARCADERO WAY
PALO ALTO, CA 94303
(415) 321-0410

Appendix C

The APL least squares program

```
∇ LSQ B;K;E;R;RD;RES;M;MAV;P;I;E
⍝ B IS FIRST GUESS FOR XMITR POSITION IN CARTESIAN COORDINATES
⍝ NI IS NUMBER OF ITERATIONS; 1÷FLATTENING=1÷298.3
1-FL×2-FL
 5
T:I←I+1
⍝ DISPLACE B FROM POLES AND ADJUST LENGTH TO SURFACE;AE= EARTH RADIUS
B[1]←B[1]+1E¯7×AE×(B[1]=0)^B[2]=0
 2×AE÷((B[1]*2)+(B[2]*2)+(B[3]*2)÷K)*0.5
⍝ S IS A N BY 3 MATRIX OF SATELLITE POSITIONS; V, OF VELOCITIES
 -(⍴S)⍴B
 E*2)*0.5
⍝ RD IS VECTOR OF RANGE RATES BETWEEN SATELLITE AND POSITION B
 (+/E×V)÷R
⍝ RDM IS N COMPONENT VECTOR OF MEASURED RANGE RATES
RES←RDM-RD
RES←RES-BIAS←(+/RES)÷⍴RES
→(I=NI+1)/END
⍝ CALCULATE MATRIX OF RESIDUAL EQUATIONS
M←-V÷⍉(⌽⍴V)⍴R
M←M+E×⍉(⌽⍴B)⍴RD÷R*2
M←M-(⍴M)⍴MAV←(+/[1] M)÷1↑⍴M
⍝ CALCULATE SPHERICAL-CARTESIAN TRANSFORMATION:
TR← 3 2 ⍴(-B[1]×B[3]),(-B[2]),(-B[2]×B[3]),B[1],(K×(B[1]*2)+B[2]*
 2),0
⍝ LEAST SQUARES SOLUTION OF RESIDUAL EQUATIONS
 .×TR
 P+.×RES⌹M
 +2
 ART
 'RMS RESIDUALS:  ';((+/RES*2)÷⍴RES)*0.5
⍝ BTOGB IS TRANSFORMATION FROM CARTESIAN TO GEODETIC COORDINATES
 , LON, HGT ARE:  ';BTOGB B
 'ITION IS:'
 H←B

 5561
```

Km

ORIGINAL PAGE IS
OF POOR QUALITY

Bibliography

Sterbeuz, Pat H. (1974), "Floating-Point Computation", Prentice-Hall, Inc., Englewood Cliffs, N.J.

Hashizume, Burt (Nov. 1977), "Floating Point Arithmetic", Byte, Vol. 2, No. 11, pp. 76-78, 180-188.

Marini, John W. (Oct. 1976), "Initial Position Estimates for Satellite-Aided Search and Rescue", Goddard Space Flight Center, Greenbelt, Maryland.

Microprocessor Utilization in Search & Rescue Missions

FINAL REPORT

Introduction:

The position of an emergency transmitter may be determined by measuring the Doppler shift of the distress signal as received by an orbiting satellite. This requires the computation of an initial estimate and refinement of this estimate through an iterative, nonlinear, least-squares estimation.

A version of the above algorithm was implemented at Goddard Space Flight Center (GSFC) and tested by locating a transmitter on the premises and obtaining observations from a satellite. The computer used was an IBM 360/95. The position was determined within the desired 10 km radius accuracy.

The purpose of this project is to determine the feasibility of performing the same task in real time using microprocessor technology. The least square algorithm was implemented on an Intel 8080 microprocessor and the same experiment was run as at GSFC.

The results indicate that a microprocessor can easily match the IBM implementation in accuracy and be performed inside the time limitations set.

Why Microprocessors:

Time is an implicit restriction in any search and rescue mission. The use of satellites and computers is dictated by that time limit. The use of a big computer to determine the position presupposes communication between the satellite and the computer. This communication introduces a time delay since the satellite is not always within radio visibility of an installation that possesses both the communication and computing power for this problem. Furthermore the result has to be forwarded to a command center to do the dispatching.

Microprocessor utilization can alleviate this situation in two ways: by giving cheap computing power to communication facilities or by incorporating the computing power in the satellite itself thus eliminating this communication completely.

Microprocessors offer light weight, small volume, low power processing. Their speed is improving rapidly and their cost is going down. They are the logical choice for a satellite search and rescue system if they can perform.

Machine Configuration:

Strictly speaking there are three microprocessor configurations in this project which we are going to discuss individually.

- Development system
- Minimal execution system
- Actual field configuration

Initially our development system consisted of an MDS-80 Intellec microcomputer by Intel with 16k bytes of RAM memory and a resident ROM monitor. Most of the floating point package was developed in machine language on that system using the monitor's limited hexadecimal editor and debugger. The need for more sophistication became apparent. After several failures in exploring alternatives (as fancy as hooking up to a PDP 11 through a telephone line for more storage) we were able to acquire a dual floppy disk drive by Intel. A spare line printer was attached to the system with minor hardware modifications and 16k bytes more RAM were added in order to support DOS. The enhanced system had the power of a mini-computer in software (assembler, editor, library manager, linkage editor, leader, and a sufficient file manager) at a speed which was slow but acceptable. The floating point package was converted to assembly language, and two more packages were developed: the I/O package and the matrix manipulation package. Unexpected help came from the use of ICE-80 (In Circuit Emulator), designed for a different application, as a powerful symbolic debugger substituting for the monitor hexadecimal debugger.

Out of this final version of the development system only a limited amount of resources were used for the final run. Those define the minimal execution system. The disk was only used for input of data. The essential parts were:

- The CPU card
- 16k Bytes of memory

- The console device and its interface
- Power supply: 12V, 5V, -5V, ground

Additionally, the line printer was used to produce a hardcopy version of the results.

The actual field configuration would be the same if the machine were located on the ground. Some kind of communications equipment would be required to provide the data input and, maybe, start the run automatically. The configuration would be different, though, if the machine were located on the satellite. The requirements for the satellite configuration would be:

- The CPU card
- 16k bytes of memory
- An interface that can load the information in memory
- A means to communicate the result to the world
- Power supply: 12V, 5V, -5V, ground

The Floating Point Package:

Based on estimates of the number of operations required we were inclined to think that any floating point operations would have to be performed by hardware and not by software since estimated times became prohibitive. This floating point package was developed to help us count the actual number of operations rather than perform them in an actual situation. The final run proved our estimates wrong and the package gained new importance.

There are a number of representations of floating poing numbers differing in accuracy and range as a trade off to the number of bytes required per number. The one used was the ANSI format for FORTRAN which happens to be implemented by hardware as an option in IBM computers. It consists of one sign bit, a seven bit exponent (excess 64), and a 24 bit mantissa of hexadecimal digits. The accuracy is 6 hexadecimal digits or approximately 7.2 decimal digits. Specific operations were not timed although a more general timing analysis appears in a later section. This format was chosen as opposed to the BCD format because the space requirements are lower for the same amount of precision, which in turn reduces execution time slightly. A mantissa of binary digits was not used because of the frequent need for normalization.

Addition and subtraction take exactly the same time, whereas multiplication is approximately equal to 22 addition and division is approximately 60 additions.

Multiplication produces a 48 bit result mantissa which is then normalized and rounded to 24 bits. This preserved the number of significant digits, or, viewed from a different angle, is the same as a double precision multiply if the two arguments were expanded with zero fill.

Division preserves the significant digits again by expanding the mantissa of the dividend to double precision and results in full single precision result. Normalization and rounding occur as in multiplication.

Accuracy is thus preserved to true single precision throughout in a numerically stable manner keeping the length of the number to 4 bytes. The cost is expensive multiplication and, expeciallu, division. This dictates a programming style whereby division is avoided unless it is absolutely necessary. The benefits, on the other hand, are numerically stable implementations whose results match the double precision to the extent possible as will be seen when the results of the run are analyzed.

The square root function was implemented by using a variation of Heron's formula based on the observation that the mantissa of any floating point number will have a value of 1/16 to 1 (interpreted as a fraction). As a first guess an approximation to a straight line connecting the two end points is made. Experimentally, six iterations were found necessary to produce an accurate result. A better first guess could improve that significantly, but time constraints did not allow us to pursue that direction.

Finally, input and output of floating point numbers turn out a much more serious task than first expected. The input routine recognized numbers with a maximum of ten integer and ten fraction digits. This proved more than sufficient for our needs. The output routine produces a rigid scientific format with 10 fraction digits. When interpreting the results it should be kept in mind that at most only 7 are significant. The format was retained in case of future expansion of the mantissa. Both the input and output routines could be better, but since their function is only tangential to the project at hand they were kept on the bare functional level.

Matrix Operations:

All matrices in the system are defined as two dimensional, including vectors. The first two bytes contain the number of rows and the number of columns in the particular matrix, respectively. This effectively limits the number of observations to 256. Vectors have one of their dimensions identically equal to 1. The next two bytes contain the address of the first byte that follows the last byte belonging to the matrix. Adjacent elements in a row of the matrix are stored as adjacent floating point numbers in memory. Rows are stored sequentially starting from the first row in the fifth byte.

In an effort to minimize the number of address calculations in the least squares algorithm the APL program we were supplied with, (LSQ), was converted into FORTRAN. The calculations involved in the residual equations were all grouped together inside one big loop. The advantage of such a scheme is that once an offset is calculated it can be used to address all the needed elements of the matrices involved in the calculation. When the time came though, to implement it using 8080 assembly language, it became all too apparent that there were too many addresses to keep track of and too few registers to help. Therefore, due to the limitation of addressing capabilities, routines were implemented for the various matrix operators in APL. This resulted in well structured and very efficient code, the style being dictated by the instruction set.

A minimum number of matrix utility routines was necessary. Matrices can be created by specifying their dimensions, they can be filled with zeros, they can be read from a device, they can be moved (copied) in storage.

There are four classes of operations by which matrices may be altered involving the following arguments.

- a constant and a matrix
- a vector and a matrix

- two matrices (plus possibly a result matrix)
- one matrix (for example, inversion)

In our particular application there was only one inversion of a 2 by 2 matrix involved. A simple algorithm derived from Euler's method is implemented using fixed pivots. Execution time and temporary storage are optimized.

Implementing the Experiment:

Having developed the tools that were discussed in previous sections the actual implementation was straight forward. For reasons already mentioned a routine was written to match the LSQ routine* developed by Dr. Marini almost statement by statement. The correspondence is indicated in the source program by keeping track of the APL statement numbers. The array names were kept the same as much as possible and only one additional temporary matrix was required. The program was written for a maximum of 100 observations. All matrix operations as well as the square root keep track of the calls to the floating point routines.

The whole package makes limited use of two monitor routines, which can easily be eliminated. The reason they are there is because software was being developed in machine language and the monitor provided a lot of needed help. So, essentially, LSQ can be run completely independently.

The space requirements for this particular run was approximately 16k bytes, out of which 4k could be in ROM and 12k in RAM. The exact numbers are as follows:

Code: 3656 bytes

Data: 10365 bytes

Stack: 100 bytes (arbitrarily)

Total: 14121 bytes

Incorporated into the package were four counting routines that kept track of the number of additions, subtractions, multiplications and divisions required during each iteration. The results will be analyzed in the next section. The actual implementation would not require these routines. The counting overhead to each arithmetic operation is approximately equal to half the time of an addition.

---

* See Appendix C.

Interpreting the Results:

The final run converged and yielded five digits of accuracy. If convergence is defined as a ratio of two succesive RMS residuals being close to 1 (in absolute) it was attained at the ninth iteration to within 0.00001. Comparing these results to the run at GSFC (run at double precision, or 16 digits of accuracy) we note the 5 digit accuracy of our result.

Numerical analysis gives us enough tools to justify the loss of two significant digits in the course of the iterations. The main source of error appears to be the subtraction of the estimated range rates from the actuals. The subtraction of the average residual equations could could contribute to the error as well.

The measured execution time for this particular run was 62 seconds per iteration. The microprocessor used was an 8080A by Intel. Adjusting for counting the number of operations the true time becomes 61 seconds. The 8080A CPU has a cycle time of 2 microseconds. If this system were actually implemented, the 8080A-1 CPU could be used which offers higher speed with cycle time of 1.3 microseconds which could bring execution time down to 40 seconds for each iteration giving approximately 6 minutes to reach convergence. This figure is derived with no modification of the software. Since it falls within our difinition of "real time", which was around 15 minutes, it is definitely a workable solution.

Another alternative is, of course, to use hardware floating point units. Two units that we are familiar with indicate a disparity in execution times of several orders of magnitude. Their specifications appear in Appendix B for the purposes of the following analysis, 'typical' execution times for 8 digits of precision of the North Star Computers, Inc. FPB unit were used. Our system indicated the following frequency of floating point operations for each iteration:

Subtractions - 672

Multiplications - 2382

Divisions - 940

When trying to compute the time it would take to execute those instructions we noticed that the time it takes to access hardware floating point unit is more than twice than the time it takes to do the calculations. Namely, we came up with the following numbers:

TIME (SEC)	PURPOSE
0.35	perform the operations
0.825	input and output the number form FPB (8080A-1)
1.175	total time required

Therefore, use of hardware units make it possible to decrease the execution time by one order of magnitude.

Future Research:

The parameters that have to be optimized in the search and rescue mission consist of the accuracy of the position estimation and the time in which it is performed. Proving the feasibility of a microprocessor implementation is far from devising an optimal algorithm.

If the nonlinear regression method is utilized there is a lot of room for improvement in the inital estimate, a quantity that can affect the whole outcome of the iterations. Several methods that are suggested in Dr. Marini's paper can be explored. Furthermore, since the data collection takes an appreciable amount of time an algorithm should be devised in which an estimate is upgraded with each incoming datum. If that algorithm is good enough then the estimate could be the result itself.

A further enhancement on the calculation tiem can be achieved through parallelism. It can appear on two levels:

- The implementation of the least squares algorithm
- The grouping of data

The least squares algorithm may be broken into parallel subtasks that can be performed by different processors in parallel, especially floating point operations.

The data may be grouped in clusters on which the least squares algorithm is applied. The estimate provided by each cluster is then processed through least squares estimation itself. This method could be applied at data collection time too.

Appendix A

- Sample run at GSFC
- Sample run at Columbia

NASA sample run at GSFC

ORIGINAL PAGE IS
OF POOR QUALITY

```
1 LSQ E
  DUALS:  1.050744637
  ON, HGT ARE:  42.3067572
  ON IS:
  245  ¯4693.396224  4270.876121
  BIAS
  3993

1 LSQ EN
  DUALS:  0.1389598763
  ON, HGT ARE:  40.33781496  277.2125729  9.094947018E¯13
  ON IS:
  20014  ¯4830.001894  4106.660277
  BIAS
  5525

1 LSQ EN
  DUALS:  0.07064076362
  ON, HGT ARE:  39.76197471  279.9200369  9.094947018E¯13
  ON IS:
  25536  ¯4836.251765  4057.716223
  BIAS
  7161

1 LSQ EN
  DUALS:  0.06507914984
  ON, HGT ARE:  39.47765191  281.0740575  9.094947018E¯13
  ON IS:
  357  ¯4837.990343  4033.099884
  BIAS
  5854

  LSQ EN
  DUALS:  0.06476481400
  ON, HGT ARE:  39.36049512  281.5359592  9.094947018E¯13
  ON IS:
  331  ¯4838.305495  4023.05154
  BIAS
  44962

1 LSQ EN
  DUALS:  0.06474873478
  ON, HGT ARE:  39.32402904  281.6773949  0
  ON IS:
  8999  ¯4838.366535  4020.220503
  BIAS
  2843

  LSQ EN
  DUALS:  0.06474806885
  ON, HGT ARE:  39.31537952  281.7107637  0
  ON IS:
  3076  ¯4838.378943  4019.477604
  BIAS
  4357

  LSQ EN
  DUALS:  0.06474804013
  , HGT ARE:  39.31352503  281.7179045  0
  IS:
  575  ¯4838.381637  4019.318311
```

LSQ EN
UALS: 0.06474004013
HGT ARE: 39.31352503 201.7179945
IS:
575 T4838.381667 4019.318311
AS
235

ORIGINAL PAGE IS
OF POOR QUALITY

LSQ EN
UALS: 0.06474803883
HGT ARE: 39.31303801 201.7197700
IS:
009 T4838.382347 4019.276470
AS
0162

LSQ EN
UALS: 0.06474803883
HGT ARE: 39.31303448 201.7197924
IS:
066 T4838.382353 4019.276175
AS
0188

```
THE RESULTING POSITION IS:
X= -0.5514835815E+03   Y= -0.4828028678E+04   Z=  0.4391360137E+04

RMS RESIDUALS =  0.1154615402E+01

THE RESULTING POSITION IS:
X=  0.6216947555E+03   Y= -0.4912517547E+04   Z=  0.4176822662E+04

RMS RESIDUALS =  0.1050682067E+01

THE RESULTING POSITION IS:
X=  0.8464587211E+03   Y= -0.4839673995E+04   Z=  0.4060588836E+04

RMS RESIDUALS =  0.1389477729E+00

THE RESULTING POSITION IS:
X=  0.9470703125E+03   Y= -0.4838634496E+04   Z=  0.4033941268E+04

RMS RESIDUALS =  0.0706402111E+00

THE RESULTING POSITION IS:
X=  0.9875150680E+03   Y= -0.4838407516E+04   Z=  0.4023446083E+04

RMS RESIDUALS =  0.0650796218E+00

THE RESULTING POSITION IS:
X=  0.9999647140E+03   Y= -0.4838378906E+04   Z=  0.4020233154E+04

RMS RESIDUALS =  0.0647644424E+00

THE RESULTING POSITION IS:
X=  0.1002853393E+04   Y= -0.4838379859E+04   Z=  0.4019498825E+04

RMS RESIDUALS =  0.0647490596E+00

THE RESULTING POSITION IS:
X=  0.1003520965E+04   Y= -0.4838384628E+04   Z=  0.4019725256E+04

RMS RESIDUALS =  0.0647480487E+00

THE RESULTING POSITION IS:
X=  0.1003620147E+04   Y= -0.4838379859E+04   Z=  0.4019700460E+04

RMS RESIDUALS =  0.0647478675E+00

THE RESULTING POSITION IS:
X=  0.1003682136E+04   Y= -0.4838384628E+04   Z=  0.4019728394E+04

RMS RESIDUALS =  0.0647482872E+00

THE RESULTING POSITION IS:
X=  0.1003667831E+04   Y= -0.4838379859E+04   Z=  0.4019729094E+04

RMS RESIDUALS =  0.0647483444E+00
```

ORIGINAL PAGE IS
OF POOR QUALITY

Appendix B

Two typical hardware floating point units

- FPB by North Star Computers, Inc.
- FPU by Cyberuetic Micro Systems

FPB DATA SHEET

EXECUTION TIMES 1, 2, 3

PRECISION DIGITS:		2	4	6	8	10	12	14
ADD	best	1	1	1	1	1	1	1
	typical	8	8	9	9	10	10	11
	worst	10	10	10	11	11	12	12
SUBTRACT	best	4	4	4	4	4	4	4
	typical	8	8	9	9	10	10	11
	worst	15	16	17	18	19	20	21
MULTIPLY	best	5	5	5	5	5	5	5
	typical	18	34	55	80	111	146	186
	worst	51	125	228	382	527	720	933
DIVIDE	best	7	7	7	7	7	7	7
	typical	39	70	109	156	211	274	370
	worst	62	139	229	340	470	621	779

1. Times given in microseconds
2. Execution times are a function of the input values
3. Times listed do not include transmission of input values and result

Board dimensions:

Model A: 5 in. by 10 in.
Model B: 6¾ in. by 12 in.

Power requirements:

Model A: 8 V (unregulated) @ 1.7 A
Model B: 5 V (regulated) @ 1.7 A

Board Construction:

FR4 material, gold plated edge connectors

Floating point number representation:

Byte 1: bit 7=sign (1=negative number, 0=positive number)
bits 6-0 = exponent in excess 64 binary representation
bits 7-0 = zero represents the zero value

Byte 2: bits 3-0 = least significant digit of value in BCD coding
bits 7-4 = next least significant digit of value

Byte n: bits 7-4 = most significant digit of value in BCD coding
bits 3-0 = next most significant digit of value

All values are nomalized.

Other representations of BCD floating point numbers require a change in microcode and are available on special order.

```
*Sample use of the North Star FPB for a divide operation with 8 digit precision
*In this example assume arguments are in memory in form:
* Most significant byte (msb) digit pair
* Susequent digit pairs follow the msb
* Exponent/sign byte follows lsb digit pair.
* Pointer addresses the exponent/sign byte
*BC has left arg pointer
*DE has right arg pointer
*HL has result pointer

*The FPB receives its arguments by "peeking" at the 8Ø8Ø bus
*when the argument values are loaded to accumulator.
*Two jumperable "hardwired" addresses are required for signaling the FPB

*This routine may be generalized to perform any operation, at any precision.

FDIV LDA RSTRT          This "hardwired" reference signals FPB to "wake up"
     MVI A,8*16+DIVOP   Specify precision and operation code to FPB
     LDAX D             Exponent/sign byte of right arg
     DCX D              Advance pointer to next byte
     LDAX D             Least significant digit pair of right arg
     DCX D              Advance pointer to next byte
     LDAX D
     DCX D
     LDAX D
     DCX D
     LDAX D             Most significant digit pair of right arg
     LDAX B             Exponent/sign byte of left arg
     DCX B
     LDAX B             Least significant digit pair of left arg
     DCX B
     LDAX B
     DCX B
     LDAX B
     DCX B
     LDAX B             Most significant digit pair of left arg
     Now the Floating Point Board is performing the operation
     LXI D,FPDIN        "Hardwired" address for receiving value from FPB
FDIV1 LDAX D            Loop waiting for completion signal (sign bit)
     ORA A              The FPB is done when the sign bit becomes "1"
     JP FDIV1           Loop if sign bit is still "Ø"
     ANI EBITS          Check for error, condition tested at end
     LDAX D             Exponent/sign of result
     MOV M,A            Store exponent/sign of result
     DCX H              Advance pointer.
     LDAX D             Least significant digit pair of result
     MOV M,A
     DCX H
     LDAX D
     MOV M,A
     DCX H
     LDAX D
     MOV M,A
     DCX H
     LDAX D             msb byte of result
     MOV M,A            Store it
     RZ                 Return if no error was detected
     JMP ERROR          Go report error (i.e. underflow or divide by Ø)
```

FLOATING POINT UNIT

PRICE LIST

MODEL	QUANTITY 1	25	100
#1	$595.00	$535.00	$475.00
#2	470.00	425.00	375.00
#3	345.00	315.00	275.00

All sales FOB Palo Alto

EXECUTION TIMES

FUNCTION	TIME IN MILISECONDS (approximate)
ADD, SUB	110
MUL, DIV, SQRT	225
TAN	846
LN, SIN, COS,→POL	1250
POWER	1720

CYBERNETIC MICRO SYSTEMS
2460 EMBARCADERO WAY
PALO ALTO, CA 94303
(415) 321-0410

Appendix C

The APL least squares program

```
∇ LSQ B;K;E;R;RD;RES;M;MAV;I;N
⍝ B IS FIRST GUESS FOR XMITR POSITION IN CARTESIAN COORDINATES
⍝ NI IS NUMBER OF ITERATIONS; FL FLATTENING = 1÷298.3
K←1-FL×2-FL
I←0
L1:I←I+1
⍝ DISPLACE B FROM POLES AND ADJUST LENGTH TO SURFACE;AE= EARTH RADIUS
B[1]←B[1]+1E¯7×AE×(B[1]=0)∧B[2]=0
B←B×AE÷((B[1]*2)+(B[2]*2)+(B[3]*2)÷K)*0.5
⍝ S IS A N BY 3 MATRIX OF SATELLITE POSITIONS; V, OF VELOCITIES
E←S-(⍴S)⍴B
R←(+/E*2)*0.5
⍝ RD IS VECTOR OF RANGE RATES BETWEEN SATELLITE AND POSITION B
RD←(+/E×V)÷R
⍝ RDM IS N COMPONENT VECTOR OF MEASURED RANGE RATES
RES←RDM-RD
RES←RES-BIAS←(+/RES)÷⍴RES
→(I=NI+1)/END
⍝ CALCULATE MATRIX OF RESIDUAL EQUATIONS
M←-V÷⍉(⌽⍴V)⍴R
M←M+E×⍉(⌽⍴E)⍴RD÷R*2
M←M-(⍴M)⍴MAV←(+/[1] M)÷1↑⍴M
⍝ CALCULATE SPHERICAL-CARTESIAN TRANSFORMATION:
TR←3 2 ⍴(-B[1]×B[3]),(-B[2]),(-B[2]×B[3]),B[1],(K×(B[1]*2)+B[2]*

⍝ LEAST SQUARES SOLUTION OF RESIDUAL EQUATIONS
M←M+.×TR
B←B+TR+.×RES⌹M
→L1
END:
'RMS RESIDUALS:  ';((+/RES*2)÷⍴RES)*0.5
⍝ BTOGB IS TRANSFORMATION FROM CARTESIAN TO GEODETIC COORDINATES
'LAT, LON, HGT ARE:  ';BTOGB B
'POSITION IS:'
B←B
∇

6361

Km
```

ORIGINAL PAGE IS
OF POOR QUALITY

Bibliography

Sterbeuz, Pat H. (1974), "Floating-Point Computation", Prentice-Hall, Inc., Englewood Cliffs, N.J.

Hashizume, Burt (Nov. 1977), "Floating Point Arithmetic", Byte, Vol. 2, No. 11, pp. 76-78, 180-188.

Marini, John W. (Oct. 1976), "Initial Position Estimates for Satellite-Aided Search and Rescue", Goddard Space Flight Center, Greenbelt, Maryland.

MICROPROCESSOR UTILIZATION IN SEARCH & RESCUE MISSIONS

Mischa Schwartz
Theodore R. Bashkow
Department of Electrical Engineering &
Computer Science
1312 S.W. Mudd Building, Columbia University
New York, NY 10027

January 1978
Final Report for Period 9/27/76 - 9/26/77

Prepared for

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

TECHNICAL REPORT STANDARD TITLE PAGE

1 Report No.	2. Government Accession No	3 Recipient's Catalog No.

4 Title and Subtitle

MICROPROCESSOR UTILIZATION IN SEARCH & RESCUE MISSIONS

5 Report Date

6 Performing Organization Code

7. Author(s)
Mischa Schwartz and Theodore R. Bashkow

8 Performing Organization Report No

9. Performing Organization Name and Address
Dept. of Elec. Eng. & Computer Science
1312 S.W. Mudd Building, Columbia University
New York, NY 10027

10. Work Unit No.

11. Contract or Grant No.
NAS 5 23727

13. Type of Report and Period Covered
FINAL REPORT
9/27/76 - 9/26/77

12. Sponsoring Agency Name and Address
Nat'l Aeronautics & Space Administration
Goddard Space Flight Center
Greenbelt, Maryland 20771

14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

The position of an emergency transmitter may be determined by measuring the Doppler shift of the distress signal as received by an orbiting satellite. This requires the computation of an initial estimate and refinement of this estimate through an iterative, non-linear, least-squares estimation.

A version of the above algorithm was implemented at Goddard Space Flight Center (GSFC) and tested by locating a transmitter on the premises and obtaining observations from a satellite. The computer used was an IBM 360/95. The position was determined within the desired 10 km radius accuracy.

The purpose of this project is to determine the feasibility of performing the same task in real time using microprocessor technology. The least square algorithm was implemented on an Intel 8080 microprocessor and the same experiment was run as at GSFC.

The results indicate that a microprocessor can easily match the IBM implementation in accuracy and be performed inside the time limitations set.

17. Key Words (Selected by Author(s))	18. Distribution Statement		
19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*

*For sale by the Clearinghouse for Federal Scientific and Technical Information, Springfield, Virginia 22151.

Microprocessor Utilization in Search & Rescue Missions

FINAL REPORT

Introduction:

The position of an emergency transmitter may be determined by measuring the Doppler shift of the distress signal as received by an orbiting satellite. This requires the computation of an initial estimate and refinement of this estimate through an iterative, nonlinear, least-squares estimation.

A version of the above algorithm was implemented at Goddard Space Flight Center (GSFC) and tested by locating a transmitter on the premises and obtaining observations from a satellite. The computer used was an IBM 360/95. The position was determined within the desired 10 km radius accuracy.

The purpose of this project is to determine the feasibility of performing the same task in real time using microprocessor technology. The least square algorithm was implemented on an Intel 8080 microprocessor and the same experiment was run as at GSFC.

The results indicate that a microprocessor can easily match the IBM implementation in accuracy and be performed inside the time limitations set.

Why Microprocessors:

Time is an implicit restriction in any search and rescue mission. The use of satellites and computers is dictated by that time limit. The use of a big computer to determine the position presupposes communication between the satellite and the computer. This communication introduces a time delay since the satellite is not always within radio visibility of an installation that possesses both the communication and computing power for this problem. Furthermore the result has to be forwarded to a command center to do the dispatching.

Microprocessor utilization can alleviate this situation in two ways: by giving cheap computing power to communication facilities or by incorporating the computing power in the satellite itself thus eliminating this communication completely.

Microprocessors offer light weight, small volume, low power processing. Their speed is improving rapidly and their cost is going down. They are the logical choice for a satellite search and rescue system if they can perform.

Machine Configuration:

Strictly speaking there are three microprocessor configurations in this project which we are going to discuss individually.

- Development system
- Minimal execution system
- Actual field configuration

Initially our development system consisted of an MDS-80 Intellec micro-computer by Intel with 16k bytes of RAM memory and a resident ROM monitor. Most of the floating point package was developed in machine language on that system using the monitor's limited hexadecimal editor and debugger. The need for more sophistication became apparent. After several failures in exploring alternatives (as fancy as hooking up to a PDP 11 through a telephone line for more storage) we were able to acquire a dual floppy disk drive by Intel. A spare line printer was attached to the system with minor hardware modifications and 16k bytes more RAM were added in order to support DOS. The enhanced system had the power of a mini-computer in software (assembler, editor, library manager, linkage editor, leader, and a sufficient file manager) at a speed which was slow but acceptable. The floating point package was converted to assembly language, and two more packages were developed: the I/O package and the matrix manipulation package. Unexpected help came from the use of ICE-80 (In Circuit Emulator), designed for a different application, as a powerful symbolic debugger substituting for the monitor hexadecimal debugger.

Out of this final version of the development system only a limited amount of resources were used for the final run. Those define the minimal execution system. The disk was only used for input of data. The essential parts were:

The CPU card

16k Bytes of memory

- The console device and its interface
- Power supply: 12V, 5V, -5V, ground

Additionally, the line printer was used to produce a hardcopy version of the results.

The actual field configuration would be the same if the machine were located on the ground. Some kind of communications equipment would be required to provide the data input and, maybe, start the run automatically. The configuration would be different, though, if the machine were located on the satellite. The requirements for the satellite configuration would be:

- The CPU card
- 16k bytes of memory
- An interface that can load the information in memory
- A means to communicate the result to the world
- Power supply: 12V, 5V, -5V, ground

The Floating Point Package:

Based on estimates of the number of operations required we were inclined to think that any floating point operations would have to be performed by hardware and not by software since estimated times became prohibitive. This floating point package was developed to help us count the actual number of operations rather than perform them in an actual situation. The final run proved our estimates wrong and the package gained new importance.

There are a number of representations of floating poing numbers differing in accuracy and range as a trade off to the number of bytes required per number. The one used was the ANSI format for FORTRAN which happens to be implemented by hardware as an option in IBM computers. It consists of one sign bit, a seven bit exponent (excess 64), and a 24 bit mantissa of hexadecimal digits. The accuracy is 6 hexadecimal digits or approximately 7.2 decimal digits. Specific operations were not timed although a more general timing analysis appears in a later section. This format was chosen as opposed to the BCD format because the space requirements are lower for the same amount of precision, which in turn reduces execution time slightly. A mantissa of binary digits was not used because of the frequent need for normalization.

Addition and subtraction take exactly the same time, whereas multiplication is approximately equal to 22 addition and division is approximately 60 additions.

Multiplication produces a 48 bit result mantissa which is then normalized and rounded to 24 bits. This preserved the number of significant digits, or, viewed from a different angle, is the same as a double precision multiply if the two arguments were expanded with zero fill.

Division preserves the significant digits again by expanding the mantissa of the dividend to double precision and results in full single precision result. Normalization and rounding occur as in multiplication.

Accuracy is thus preserved to true single precision throughout in a numerically stable manner keeping the length of the number to 4 bytes. The cost is expensive multiplication and, expeciallu, division. This dictates a programming style whereby division is avoided unless it is absolutely necessary. The benefits, on the other hand, are numerically stable implementations whose results match the double precision to the extent possible as will be seen when the results of the run are analyzed.

The square root function was implemented by using a variation of Heron's formula based on the observation that the mantissa of any floating point number will have a value of 1/16 to 1 (interpreted as a fraction). As a first guess an approximation to a straight line connecting the two end points is made. Experimentally, six iterations were found necessary to produce an accurate result. A better first guess could improve that significantly, but time constraints did not allow us to pursue that direction.

Finally, input and output of floating point numbers turn out a much more serious task than first expected. The input routine recognized numbers with a maximum of ten integer and ten fraction digits. This proved more than sufficient for our needs. The output routine produces a rigid scientific format with 10 fraction digits. When interpreting the results it should be kept in mind that at most only 7 are significant. The format was retained in case of future expansion of the mantissa. Both the input and output routines could be better, but since their function is only tangential to the project at hand they were kept on the bare functional level.

Matrix Operations:

All matrices in the system are defined as two dimensional, including vectors. The first two bytes contain the number of rows and the number of columns in the particular matrix, respectively. This effectively limits the number of observations to 256. Vectors have one of their dimensions identically equal to 1. The next two bytes contain the address of the first byte that follows the last byte belonging to the matrix. Adjacent elements in a row of the matrix are stored as adjacent floating point numbers in memory. Rows are stored sequentially starting from the first row in the fifth byte.

In an effort to minimize the number of address calculations in the least squares algorithm the APL program we were supplied with, (LSQ), was converted into FORTRAN. The calculations involved in the residual equations were all grouped together inside one big loop. The advantage of such a scheme is that once an offset is calculated it can be used to address all the needed elements of the matrices involved in the calculation. When the time came though, to implement it using 8080 assembly language, it became all too apparent that there were too many addresses to keep track of and too few registers to help. Therefore, due to the limitation of addressing capabilities, routines were implemented for the various matrix operators in APL. This resulted in well structured and very efficient code, the style being dictated by the instruction set.

A minimum number of matrix utility routines was necessary. Matrices can be created by specifying their dimensions, they can be filled with zeros, they can be read from a device, they can be moved (copied) in storage.

There are four classes of operations by which matrices may be altered involving the following arguments.

- a constant and a matrix
- a vector and a matrix

- two matrices (plus possibly a result matrix)
- one matrix (for example, inversion)

In our particular application there was only one inversion of a 2 by 2 matrix involved. A simple algorithm derived from Euler's method is implemented using fixed pivots. Execution time and temporary storage are optimized.

Implementing the Experiment:

Having developed the tools that were discussed in previous sections the actual implementation was straight forward. For reasons already mentioned a routine was written to match the LSQ routine* developed by Dr. Marini almost statement by statement. The correspondence is indicated in the source program by keeping track of the APL statement numbers. The array names were kept the same as much as possible and only one additional temporary matrix was required. The program was written for a maximum of 100 observations. All matrix operations as well as the square root keep track of the calls to the floating point routines.

The whole package makes limited use of two monitor routines, which can easily be eliminated. The reason they are there is because software was being developed in machine language and the monitor provided a lot of needed help. So, essentially, LSQ can be run completely independently.

The space requirements for this particular run was approximately 16k bytes, out of which 4k could be in ROM and 12k in RAM. The exact numbers are as follows:

Code: 3656 bytes

Data: 10365 bytes

Stack: 100 bytes (arbitrarily)

Total: 14121 bytes

Incorporated into the package were four counting routines that kept track of the number of additions, subtractions, multiplications and divisions required during each iteration. The results will be analyzed in the next section. The actual implementation would not require these routines. The counting overhead to each arithmetic operation is approximately equal to half the time of an addition.

---

* See Appendix C.

Interpreting the Results:

The final run converged and yielded five digits of accuracy. If convergence is defined as a ratio of two succesive RMS residuals being close to 1 (in absolute) it was attained at the ninth iteration to within 0.00001. Comparing these results to the run at GSFC (run at double precision, or 16 digits of accuracy) we note the 5 digit accuracy of our result.

Numerical analysis gives us enough tools to justify the loss of two significant digits in the course of the iterations. The main source of error appears to be the subtraction of the estimated range rates from the actuals. The subtraction of the average residual equations could could contribute to the error as well.

The measured execution time for this particular run was 62 seconds per iteration. The microprocessor used was an 8080A by Intel. Adjusting for counting the number of operations the true time becomes 61 seconds. The 8080A CPU has a cycle time of 2 microseconds. If this system were actually implemented, the 8080A-1 CPU could be used which offers higher speed with cycle time of 1.3 microseconds which could bring execution time down to 40 seconds for each iteration giving approximately 6 minutes to reach convergence. This figure is derived with no modification of the software. Since it falls within our difinition of "real time", which was around 15 minutes, it is definitely a workable solution.

Another alternative is, of course, to use hardware floating point units. Two units that we are familiar with indicate a disparity in execution times of several orders of magnitude. Their specifications appear in Appendix B for the purposes of the following analysis, 'typical' execution times for 8 digits of precision of the North Star Computers, Inc. FPB unit were used. Our system indicated the following frequency of floating point operations for each iteration:

Subtractions - 672

Multiplications - 2382

Divisions - 940

When trying to compute the time it would take to execute those instructions we noticed that the time it takes to access hardware floating point unit is more than twice than the time it takes to do the calculations. Namely, we came up with the following numbers:

TIME (SEC)	PURPOSE
0.35	perform the operations
0.825	input and output the number form FPB (8080A-1)
1.175	total time required

Therefore, use of hardware units make it possible to decrease the execution time by one order of magnitude.

Future Research:

The parameters that have to be optimized in the search and rescue mission consist of the accuracy of the position estimation and the time in which it is performed. Proving the feasibility of a microprocessor implementation is far from devising an optimal algorithm.

If the nonlinear regression method is utilized there is a lot of room for improvement in the inital estimate, a quantity that can affect the whole outcome of the iterations. Several methods that are suggested in Dr. Marini's paper can be explored. Furthermore, since the data collection takes an appreciable amount of time an algorithm should be devised in which an estimate is upgraded with each incoming datum. If that algorithm is good enough then the estimate could be the result itself.

A further enhancement on the calculation tiem can be achieved through parallelism. It can appear on two levels:

- The implementation of the least squares algorithm
- The grouping of data

The least squares algorithm may be broken into parallel subtasks that can be performed by different processors in parallel, especially floating point operations.

The data may be grouped in clusters on which the least squares algorithm is applied. The estimate provided by each cluster is then processed through least squares estimation itself. This method could be applied at data collection time too.

Appendix A

- Sample run at GSFC
- Sample run at Columbia

NASA sample run at GSFC

589.9916861 ¯5314.999478 3489.05.100

1 LSQ E
RMS RESIDUALS: 1.050744637
LAT, LON, HGT ARE: 42.3067570 [illegible] 0
POSITION IS:
¯530.1248245 ¯4693.596224 4272.8?101
BIAS
¯5.10418993

1 LSQ EN
RMS RESIDUALS: 0.1360508763
LAT, LON, HGT ARE: 40.3378190 [illegible] 9.094947018E¯13
POSITION IS:
611.2480014 ¯4830.001894 4100.504277
BIAS
¯3.273965525

1 LSQ EN
RMS RESIDUALS: 0.0706407036
LAT, LON, HGT ARE: 39.74197471 279.9200389 9.094947018E¯13
POSITION IS:
845.5725536 ¯4836.251765 4057.716223
BIAS
¯3.2?77161

1 LSQ EN
RMS RESIDUALS: 0.0650791489
LAT, LON, HGT ARE: 39.47765191 281.0748579 9.094947018E¯13
POSITION IS:
9?2.?14857 ¯4837.990348 4033.399884
BIAS
¯3.?26854

1 LSQ EN
RMS RESIDUALS: 0.0647640140?
LAT, LON, HGT ARE: 39.36049512 281.5359592 9.094947018E¯13
POSITION IS:
987.?262331 ¯4838.305495 4023.?5154
BIAS
¯3.?5944862

1 LSQ EN
RMS RESIDUALS: 0.0647487347?
LAT, LON, HGT ARE: 39.32402904 281.6773949 0
POSITION IS:
999.8868999 ¯4838.366335 4020.220583
BIAS
¯3.?572843

1 LSQ EN
RMS RESIDUALS: 0.0647480688?
LAT, LON, HGT ARE: 39.31537952 281.7107637 0
POSITION IS:
1002.928076 ¯4838.378943 4019.477604
BIAS
¯3.?574357

1 LSQ EN
RMS RESIDUALS: 0.0647400401?
LAT, LON, HGT ARE: 39.31252503 281.7179045 0
POSITION IS:
1003.?57575 ¯4838.381637 4019.?18311
BIAS

```
  [illegible] LSQ EN
MS RESIDUALS:  0.06474004013
AT, LON, HGT ARE:  39.31352503 [illegible]
OSITION IS:
03.5[illegible]7575  -4838.381637  4019.[illegible]
    BIAS
[illegible].27[illegible] 235

  [illegible] EN
[illegible]UALS:  0.06474[illegible]
AT, [illegible] HGT ARE:  39.31303001 281.7197728 [illegible]
OSITION IS:
03.[illegible]09  -4838.382347  4019.276478
    [illegible]
[illegible]102

  1 LSQ EN
[illegible] RESIDUALS:  0.06474[illegible]
AT, LON, HGT ARE:  39.31303443 [illegible]
OSITION IS:
[illegible]006  -4838.382353  4019.[illegible]
    BIAS
[illegible]138
```

```
THE RESULTING POSITION IS:
X= -0.5524815815E+01   Y= -0.4828028678E+04   Z=  0.4393360137E+04

RMS RESIDUALS =  0.1154615402E+01


THE RESULTING POSITION IS:
X=  0.6216947555E+03   Y= -0.4912517547E+04   Z=  0.4176822652E+04

RMS RESIDUALS =  0.1050682067E+01


THE RESULTING POSITION IS:
X=  0.8464587211E+03   Y= -0.4819673997E+04   Z=  0.4060588816E+04

RMS RESIDUALS =  0.1389477729E+00


THE RESULTING POSITION IS.
X=  0.9470703125E+03   Y= -0.4873674196E+04   Z=  0.4033941768E+04

RMS RESIDUALS =  0.0706402111E+00


THE RESULTING POSITION IS:
X=  0.9875150680E+03   Y= -0.4878407516E+04   Z=  0.4027446683E+04

RMS RESIDUALS =  0.0650796218E+00


THE RESULTING POSITION IS:
X=  0.9999647140E+03   Y= -0.4883789906E+04   Z=  0.4026223174E+04

RMS RESIDUALS =  0.0647644424E+00


THE RESULTING POSITION IS
X=  0.1002852392E+04   Y= -0.4883798059E+04   Z=  0.4019498825E+04

RMS RESIDUALS =  0.0647490596E+00


THE RESULTING POSITION IS:
X=  0.1002520965E+04   Y= -0.4879784629E+04   Z=  0.4019750256E+04

RMS RESIDUALS =  0.0647480487E+00


THE RESULTING POSITION IS:
X=  0.1003620147E+04   Y= -0.4879779579E+04   Z=  0.4019760456E+04

RMS RESIDUALS =  0.0647478675E+00


THE RESULTING POSITION IS:
X=  0.1003682136E+04   Y= -0.4883784628E+04   Z=  0.4019283294E+04

RMS RESIDUALS =  0.0647482872E+00


THE RESULTING POSITION IS:
X=  0.1002667831E+04   Y= -0.4879779479E+04   Z=  0.4019299897E+04

RMS RESIDUALS =  0.0647483441E+00
```

Appendix B

Two typical hardware floating point units

- FPB by North Star Computers, Inc.
- FPU by Cyberuetic Micro Systems

FPB DATA SHEET

EXECUTION TIMES [1,2,3]

PRECISION DIGITS:		2	4	6	8	10	12	14
ADD	best	1	1	1	1	1	1	1
	typical	8	8	9	9	10	10	11
	worst	10	10	10	11	11	12	12
SUBTRACT	best	4	4	4	4	4	4	4
	typical	8	8	9	9	10	10	11
	worst	15	16	17	18	19	20	21
MULTIPLY	best	5	5	5	5	5	5	5
	typical	18	34	55	80	111	146	186
	worst	51	125	228	382	527	720	933
DIVIDE	best	7	7	7	7	7	7	7
	typical	39	70	109	156	211	274	370
	worst	62	139	229	340	470	621	779

1. Times given in microseconds
2. Execution times are a function of the input values
3. Times listed do not include transmission of input values and result

Board dimensions:

Model A: 5 in. by 10 in.
Model B: 6¾ in. by 12 in.

Power requirements:

Model A: 8 V (unregulated) @ 1.7 A
Model B: 5 V (regulated) @ 1.7 A

Board Construction:

FR4 material, gold plated edge connectors

Floating point number representation:

Byte 1: bit 7=sign (1=negative number, 0=positive number)
bits 6-0 = exponent in excess 64 binary representation
bits 7-0 = zero represents the zero value

Byte 2: bits 3-0 = least significant digit of value in BCD coding
bits 7-4 = next least significant digit of value

Byte n: bits 7-4 = most significant digit of value in BCD coding
bits 3-0 = next most significant digit of value

All values are nomalized.

Other representations of BCD floating point numbers require a change in microcode and are available on special order.

```
*Sample use of the North Star FPB for a divide operation with 8 digit precision
*In this example assume arguments are in memory in form:
*  Most significant byte (msb) digit pair
*  Susequent digit pairs follow the msb
*  Exponent/sign byte follows lsb digit pair.
*  Pointer addresses the exponent/sign byte
*BC has left arg pointer
*DE has right arg pointer
*HL has result pointer

*The FPB receives its arguments by "peeking" at the 8Ø8Ø bus
*when the argument values are loaded to accumulator.
*Two jumperable "hardwired" addresses are required for signaling the FPB

*This routine may be generalized to perform any operation, at any precision.

FDIV LDA RSTRT          This "hardwired" reference signals FPB to "wake up"
     MVI A,8*16+DIVOP   Specify precision and operation code to FPB
     LDAX D             Exponent/sign byte of right arg
     DCX D              Advance pointer to next byte
     LDAX D             Least significant digit pair of right arg
     DCX D              Advance pointer to next byte
     LDAX D
     DCX D
     LDAX D
     DCX D
     LDAX D             Most significant digit pair of right arg
     LDAX B             Exponent/sign byte of left arg
     DCX B
     LDAX B             Least significant digit pair of left arg
     DCX B
     LDAX B
     DCX B
     LDAX B
     DCX B
     LDAX B             Most significant digit pair of left arg
     Now the Floating Point Board is performing the operation
     LXI D,FPDIN        "Hardwired" address for receiving value from FPB
FDIV1 LDAX D            Loop waiting for completion signal (sign bit)
     ORA A              The FPB is done when the sign bit becomes "1"
     JP FDIV1           Loop if sign bit is still "Ø"
     ANI EBITS          Check for error, condition tested at end
     LDAX D             Exponent/sign of result
     MOV M,A            Store exponent/sign of result
     DCX H              Advance pointer.
     LDAX D             Least significant digit pair of result
     MOV M,A
     DCX H
     LDAX D
     MOV M,A
     DCX H
     LDAX D
     MOV M,A
     DCX H
     LDAX D             msb byte of result
     MOV M,A            Store it
     RZ                 Return if no error was detected
     JMP ERROR          Go report error (i.e. underflow or divide by Ø)
```

FLOATING POINT UNIT

PRICE LIST

MODEL	QUANTITY 1	25	100
#1	$595.00	$535.00	$475.00
#2	470.00	425.00	375.00
#3	345.00	315.00	275.00

All sales FOB Palo Alto

EXECUTION TIMES

FUNCTION	TIME IN MILISECONDS (approximate)
ADD, SUB	110
MUL, DIV, SQRT	225
TAN	846
LN, SIN, COS,→POL	1250
POWER	1720

CYBERNETIC MICRO SYSTEMS
2460 EMBARCADERO WAY
PALO ALTO, CA 94303
(415) 321-0410

Appendix C

The APL least squares program

```
. SQ[0]
  ... LSQ ...
  ... IS FIRST GUESS ... COORDINATES
  ..N IS NUMBER OF ...
  ...1-FL×2-FL

  I:I←I+1
  ...ISPLACE E FROM ... = EARTH RADIUS
  ...I←E[1]+1E...
  ...MAE←((E[1]*2)+...
  ... IS A N BY 3 MATRIX ... OF VELOCITIES
  ...-C...E
  ...E*2)*0.5
  ... D IS VECTOR OF RANGE ... AND POSITION E
  ...(+.E...)÷R
  ...AM IS N COMPONENT ... RATES
  ...RAM-RD
  ...RES-BIAS←(+/RES)÷...
  →(I=NI+1)/END
  ... CALCULATE MATRIX OF ... CONDITION
  ...
  ...
  ...MAV←(+/[1] ...
  ... CALCULATE SPHERICAL-CARTESIAN TRANSFORMATION
  ... 2 ρ (-E[1]×E[3]) , (-E[1]) , (-C[2]) ... [1]*2)+E[2]*
  ...
  ...AST SQUARES SOLUTION OF RESIDUAL CONDITIONS
  ... TR
  ...RESBM
  ...
  ...
  ... RESIDUALS: ...
  ... IS TRANSFORMATION FROM CARTESIAN TO GEODETIC COORDINATES
  ... LON, HGT ARE: ...
  ...ITION IS:'
  ...←E

  ...5361
```

Km

Bibliography

Sterbenz Pat H. (1974), "Floating-Point Computation", Prentice-Hall, Inc., Englewood Cliffs, N.J.

Hashizume, Burt (Nov. 1977), "Floating Point Arithmetic", Byte, Vol. 2, No. 11, pp. 76-78, 180-188.

Marini, John W. (Oct. 1976), "Initial Position Estimates for Satellite-Aided Search and Rescue", Goddard Space Flight Center, Greenbelt, Maryland.