Benchmarking

Scientific machines frequently are compared on the basis of their performance on key FORTRAN loops representative of the workload at various universities and labs. Examples include the Livermore Loops (renamed Livermore Fortran Kernels, or LFKs), the Sandia kernels, the DFVLR kernels, the NAS kernels, and the Argonne benchmark (LINPACK). However, a machine's performance on two nearly identical loop computations can show substantial performance differences. This performance brief examines various architectural considerations, which impact measurements of computational kernels. As will be shown, the Cydra 5, unlike a number of other systems, consistently yields high performance across a wide variety of FORTRAN codes.

Architectural Considerations

Machines that have high speed cache memories for data and/or instructions often demonstrate higher performance on kernel benchmarks than on more representative, larger applications. Since kernels typically are timed on a "quiet" machine (meaning no other workload), cache miss rates can be very low. In production environments such miss rates will depend upon the rest of the workload in the computer. Cache miss rates also are very sensitive to data structure size. If the benchmark fits entirely in cache, extremely high performance will be obtained. In practice, it is difficult to control the size of the data structures. Furthermore, as a software architecture consideration, some compilers may do a better job of optimizing small tight loops compared to more complex, typical production codes.

A Look At The SAXPY Operation

The acronym SAXPY stands for scalar A times X plus Y, or

```
y(i) = y(i) + a * x(i)          for i = 1, 2, ....., n
```

This kernel is timed in both the LFK set and in LINPACK. In LFK loop 21 the SAXPY operation is used to perform matrix multiplication is as follows:

```
do 21  i = 1, loop
    do 21 k = 1, 25
            do 21 i = 1, 25
                    do 21 j = 1,
                    px(i,j) = px(i,j)+ vy(i,k) * cx(k,j)          (LFK)
21   continue
```

In LINPACK the SAXPY operation is implemented through a CALL, and is used to perform row elimination with column indexing as follows:

```
do 30 j = kp1, n
     t = a(l,j)
     if (l .eq. k) go to 20
               a(l,j) = a(k, j)
               a(k,j) = t
20   continue
     call saxpy (n-k,t,a(k+1,k),1, a(k+1, j), 1)
30   continue
```

(LINPACK)

A multiprocessor system can utilize multitasking for the LINPACK loop, but not not for loop (1). Yet production codes are more likely to resemble the LFK loop since with modern, highly optimizing FORTRAN compilers simple operations such as SAXPY are most efficiently computed inline. Figure 1 shows a comparison of the Cydra 5 with other systems. The robust architecture of the Cydra 5 yields comparable performance on both loops. The LFK loop, uses a longer vector length (101 vs 51), but most machines report a lower performance partly due to the stride penalty for row access, and partly due to the flushing of the cache prior to timing the kernel.

The Cydra 5 was designed to deliver high performance with minimal degradation in performance due to strided access of data, or other programming techniques. The Cydra 5 also supports fast gather/scatter operations[1].

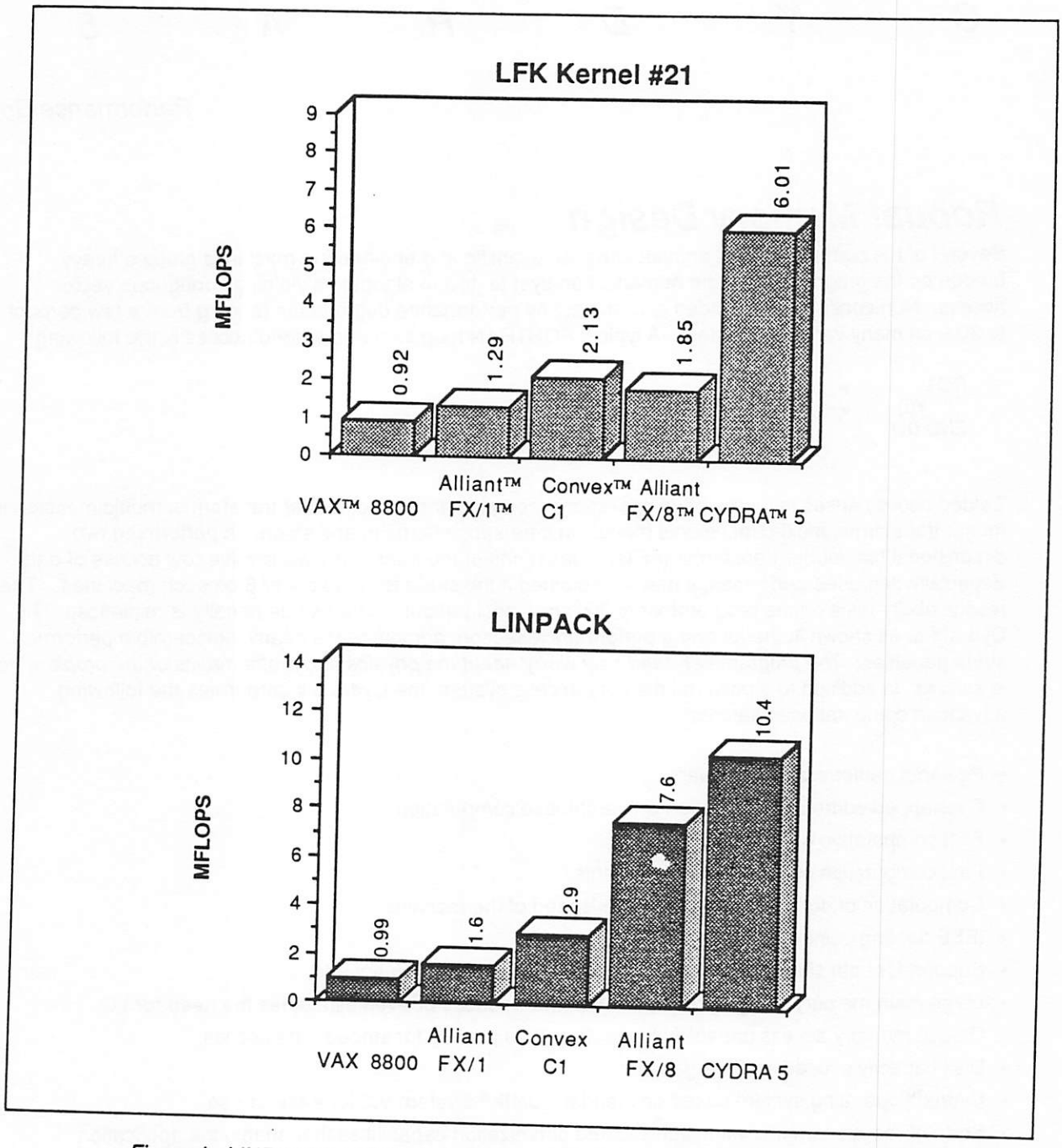


Figure 1. LINPACK and LFK Performance Comparisons

Notes:

The LFK results[2], correspond to vector length 101.

The LINPACK results[3] on the standard 100x100 full precision problem corresponds to a vector length of 50 (on average) for the loop in question.

On the LFK kernel, the FX/8 achieves little improvement over an FX/1. The FX/8 LFK performance also is adversely impacted by the fact that the vector length is reduced to 12 for the multitasked code[4].

Robust Memory Design

Several of the commonly used architectures for scientific and engineering machines place a heavy burden on the programmer or the numerical analyst to devise algorithms yielding contiguous vector access. Non-contiguous or strided access causes performance degradation ranging from a few percent to 90% on many vector computers. A typical FORTRAN loop involving strided access is the following:

```
DO I      = 1, N, M
   Y(I)   = Y(I) + X(I) * S
END DO
```
(1)

Strided access arises in matrix (row) operations, complex arithmetic, fourier transforms, multidimensional fourier transforms, multi-dimensional models in time-space domain, and so on. In performing two dimensional fast fourier transforms (FFTs) at least one of the transforms will involve row access of data. Especially degraded performance may be obtained if the stride is modulo 4 or 8 on such machines. The responsibility rests on the programmer to 'fix' array declarations so that stride penalty is minimized. The Cydra™ 5, as shown in the following performance section, demonstrates nearly indiscernible performance stride penalties. *The programmer need only worry about the physics and mathematics of the problem he is solving.* In addition to a powerful memory access system, the Cydra 5 incorporates the following advanced computational features:

- Powerful gather/scatter hardware.
- Overlapped address computation with arithmetic computation.
- Fast computation of recurrences.
- Fast computation of conditional statements.
- Computation of dot product at near peak speed of the machine.
- IEEE floating point hardware.
- Support for both single and double precision arithmetic at high speeds.
- Large main memory, ranging up to 256 MB, that reduces or even eliminates the need for I/O.
- Unique memory access capability that suffers little penalty for strided data access.
- Disk capacity exceeding 29 Gigabytes.
- Cydrix™ operating system based on standard UNIX® System V.3 for ease-of-use.
- State-of-the-art compiler with sophisticated optimization capabilities that enable the application programmer to exploit Cydra 5 computational power in FORTRAN.

Benchmark Data

The ACCU[1] benchmark suite consists of kernel codes as well as productions codes in the application areas of PDE solvers, Laplacian random walk, molecular dynamics of water molecules, etc. One of the kernels times the following task:

```
DO INCR   =  1, 16
   DO J   =  1, SIZE * INCR, INCR
   S      =  S + A(I) * B(I)
   END DO
END DO                                                    (2)
```

Note that this is actual Cydrix™ F77 code. Cydrix F77 contains several enhancements, in the spirit of the current draft proposal for FORTRAN 8x.

Data on competitive machines were published by ACCU[1]. Figure 1 shows the performance on the Cydra 5 and other competitive machines. The Cydra 5 incorporates a sophisticated memory access mechanism that is insensitive to the access pattern; *hence high performance is obtained irrespective of the access pattern.*

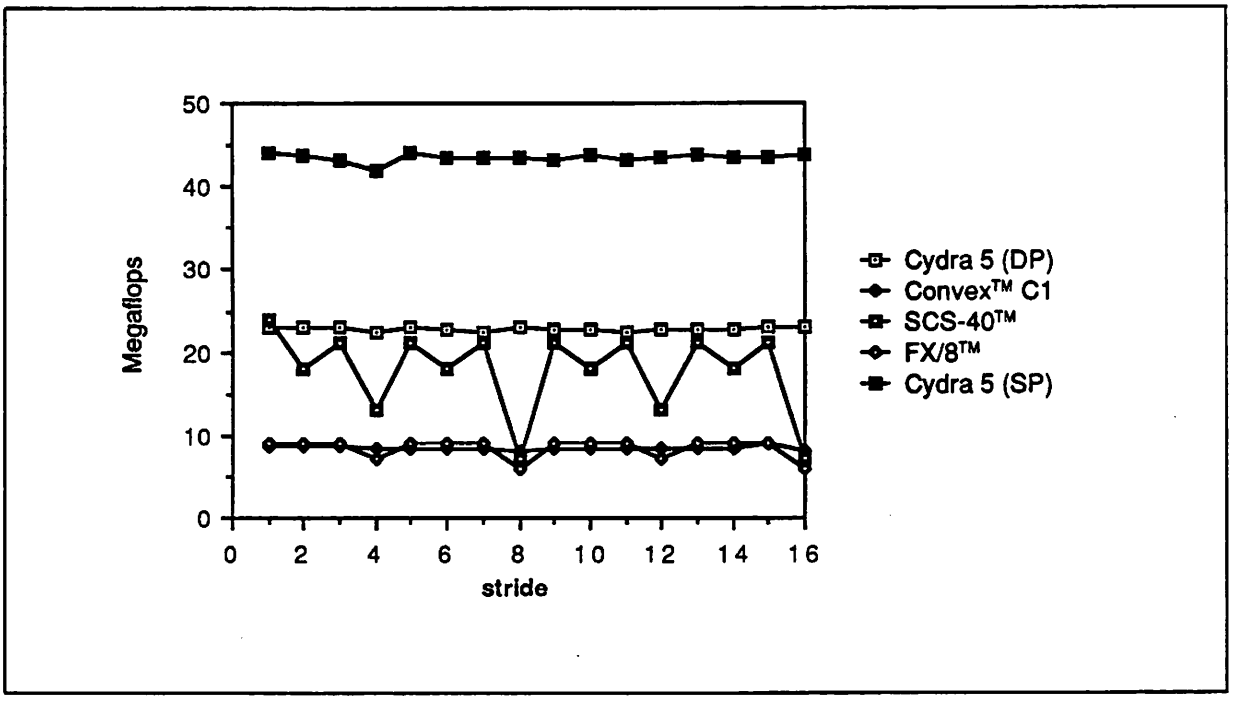


Figure 1. Stride Memory Test

Performance Brief

The Correct Mean

The Cydra 5 Departmental Supercomputer features high, sustained workload performance. One method for measuring workload performance is with a composite of multiple, representative preformance tests. In a landmark article in 1984, Jack Worlton examined various approaches to benchmarking scientific and engineering computers[1]. Worlton reinforced the concept of using a set of representative program kernels for benchmarking for reasons of efficiency, portability, and when done properly, for the accuracy of such measurements. Of the widely available tests, the Livermore FORTRAN Kernals (LFKs)[2], also referred to as the "Livermore Loops," best satisfy this measurement approach. The LFKs include 24 representative, individual tests that are characteristic of actual engineering and scientific workloads. One of these tests, "Kernel 21" is the same loop calculation, named "SAXPY," as in the well-known single application test LINPACK[3].

Early in 1987, the Lawrence Livermore Laboratory began publishing LFK performance results for a wide range of computers. With this publication[2], the use of the LFKs throughout the industry is increasing rapidly. The LFKs are becoming a preeminent workload performance measurement.

Based on Worlton's analysis, harmonic averages represent the most appropriate comparison of performance between machines on production application workloads. Various forms of averaging can be used for analyzing machine performance. Straightforward averaging of the rates, r_i, for individual kernels is referred to as arithmetic mean:

Arithmetic Mean:
$$\frac{\sum_{i=1}^{n} w_i r_i}{n}$$

The unweighted arithmetic mean (all w = 1) can be very unrepresentative of actual performance, especially for machines with unbalanced, varying performance from kernel to kernel. As succinctly discussed in Worlton's report, harmonic averages such as the harmonic mean are far more meaningful when comparing performance data on kernels:

Harmonic Mean:
$$\frac{\sum_{i=1}^{n} w_i r_i}{\sum_{i=1}^{n} \frac{w_i}{r_i}}$$

The following shows LFK data published by the Lawrence Livermore Laboratory for two well-known machines:

	Alliant™ FX/8™, 8 CEs	Convex™ C1
Harmonic Mean, Full Precision	1.6 megaflops	1.3 megaflops

As published in the announcement material for Multiflow™ TRACE™ 7/200, the harmonic mean for this machine is 2.3 megaflops.

The corresponding Cydra 5 performance is:

	Cydra 5
Harmonic Mean, Full Precision	•3.7 megaflops

The Cydra 5 harmonic mean performance is indicative of a robust, well-balanced machine that can provide high performance on a wide variety of workloads.

CYDRA 5 Highlights

- Powerful gather/scatter hardware.
- Overlapped address computation with arithmetic computation.
- Fast computation of recurrences.
- Fast computation of conditional statements.
- Computation of dot product at near peak speed of the machine.
- IEEE floating point hardware.
- Support of both single and double precision arithmetic at high speeds.
- Large main memory, ranging up to 256 MB, that reduces or even eliminates the need for I/O.
- Unique memory access capability that suffers little penalty for strided data access.
- Disk capacity exceeding 29 Gigabytes.
- Cydrix™ operating system based on standard UNIX® System V.3 for ease-of-use.
- State-of-the-art compiler with sophisticated optimization capabilities that enable the application programmer to exploit Cydra 5 computational power in FORTRAN.

References

1. J. Worlton, Datamation, Sept. 1984, p. 121

2. F. McMahon, The Livermore Fortran Kernels, Dec. 1986, National Technical Information Service

3. J.J. Dongarra, Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment, Technical Memorandum No. 23, October 5, 1987.

Linear Recurrences

Linear recurrences are among the most important class of non-vectorizable problems in typical scientific and engineering workloads. The following is an example of a simple linear recurrence:

```
SUM(1)    =  A(1)
DO I      =  2, N
   X(I)   =  X(I - 1) * z(I) + y(I)
END DO
```
(1)

The impact on overall machine performance by the presence of recurrences has been investigated by Worlton[1]. *Even when recurrences make up only a small portion of the workload, their impact can be significant.* For instance in analyzing the original fourteen Livermore kernels, 3 loops involve recurrences while 9 loops are fully vectorized on the Cray™ XMP[2]. If the performance of the 9 vectorized loops were doubled, the overall impact as measured through the harmonic mean would be on the order of 5%. If the performance of the 3 loops with recurrences were doubled, the overall performance rating would improve by a very significant 37%. *The Cvdra™ 5 was designed to execute linear recurrences at near peak speed of the machine.* In an operation such as (1) above the output term x(i - 1) can be immediately used in a succeeding arithmetic operation. A practical example of linear recurrences is the evaluation of running sums of vectors for scaling purposes in signal processing applications:

```
SUM(1)    =  A(1)
DO I      =  2, N
   SUM(I) =  SUM(I - 1) +  A(I)
END DO
```
(2)

Another common example occurs in solving partial differential equations (PDEs) with the numerically stable Crank-Nicholson schemes

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t}$$

$$u(j + 1, k) = u(j,k) + (a/2) * (u(j + 1, k + 1) - 2 * u(j + 1,k) + u(j + 1,k - 1) + u(j,k + 1)\ 2 * u(j,k) + u(j,k - 1)) \quad (3)$$

This expression involves recurrences with regard to both of the indices j and k. One approach to handling linear recurrences on vector computers has been to use recursive doubling, or cyclic reduction techniques[3]. These involve a doubling or tripling of the computational workload* and also involve the relatively low performance gather/scatter operations. Another approach is optimization in scalar mode

* Such schemes force the use of much smaller integration intervals in order to assure stability, thus increasing the total computation required.

through unrolling techniques. At best this yields a performance improvement of around 50%. The Cydra 5, with its unique ability to handle recurrences and gather/scatter operations, eliminates the need for cumbersome recoding. The result is high performance with straightforward FORTRAN code as shown in the next section.

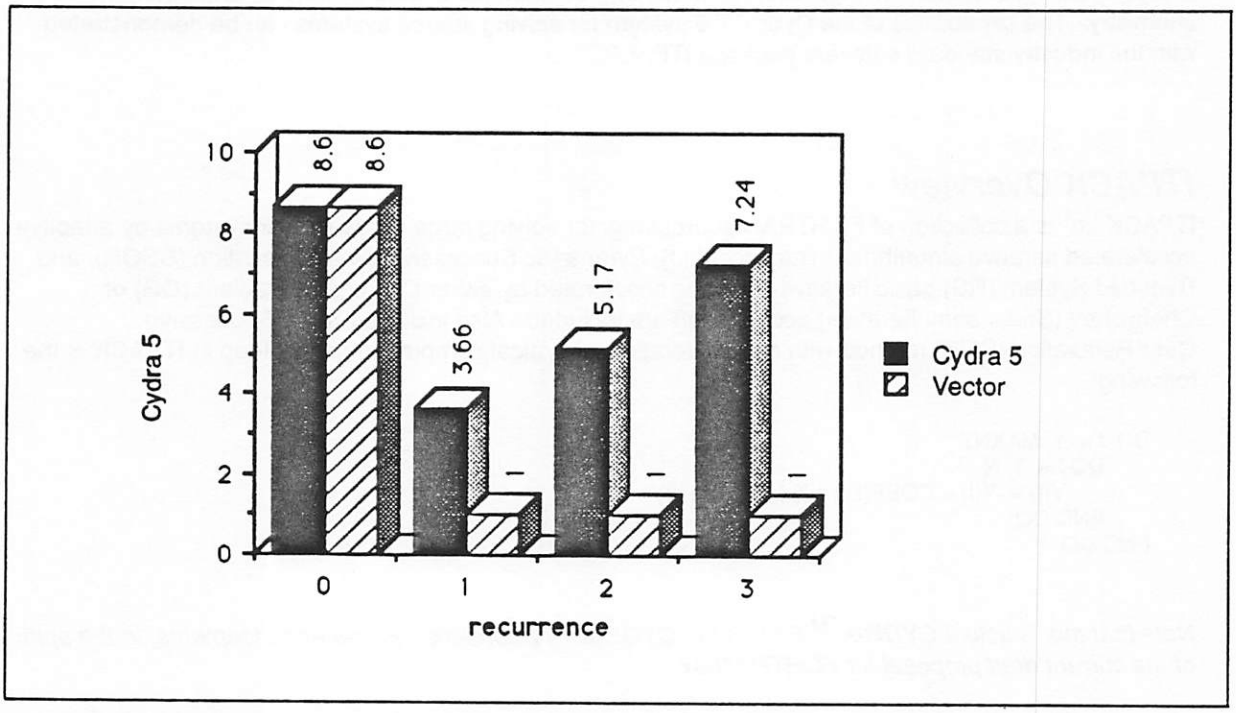
Performance Measurement

The following kernel was timed, with N = 100:

```
DO I      =  1 + j, n - j
   A(I)   =  B(I) + S * A(I - J)
ENDDO
```

(3)

where: s is a scalar
j defines the tree height of the recurrence



The j = 0 case corresponds to a normal, vectorizable operation (no recurrence). The "Vector" machine columns demonstrates typical vector processor behavior, viz full speed when there is no recurrence, and purely scalar performance thereafter. CYDRA 5 performance improves as the order of the recurrence is increased.

Figure 1.

References

1. J. Worlton, *Computer World*, Nov 9, 1981
2. J. Worlton, *Datamation*, pp 121-130, 1984
3. L. Hyafil and H.T. Kung, *CACM, vol. 24, no. 3*, p. 513, 1977.
4. Cydrome Performance Brief "Robust Memory Design".

High Performance Sparse Matrix Computations

Matrices where most of the elements are zero are called sparse matrices. Sparse matrices may arise naturally from the modeling of a physical process or they may arise during analysis, simplification, and solution of complex systems of equations. A basic source of sparsity is a common assumption that forces are localized. For instance in modeling a bridge, the equations at a node involve only the beams that meet there. The number is about the same whether the bridge is short and involves only a few beams, or whether the bridge spans several miles and involves thousands of beams. Sparse matrices may have a structure (e.g. tridiagonal, banded, block diagonal, symmetric, etc.) or the sparsity may be random. The efficient solution of sparse matrices is at the heart of such important application areas as structural analysis, finite element modeling, reservoir modeling, computational fluid dynamics, and quantum chemistry. The capabilities of the Cydra™ 5 system for solving sparse systems can be demonstrated with the industry standard software package ITPACK.

ITPACK Overview

ITPACK 2c[1] is a collection of FORTRAN subroutines for solving large sparse linear systems by adaptive accelerated iterative algorithms. The Jacobi (J), Symmetric Successive Over-Relaxation (SSOR), and Reduced System (RS) basic iterative methods accelerated by either Conjugate Gradient (CG) or Chebyshev (SI, for semi-iterative) acceleration are included. Also included is the Successive Over-Relaxation (SOR) method with no acceleration. A typical compute intensive loop in ITPACK is the following:

```
DO J = 1, MAXNZ
   DO I = 1, N
      Y(I) = Y(I) + COEF(I,J) * X (JCOEF(I,J))
   END DO
END DO                                                    (1)
```

Note that this is actual CYDRIX™ F77 code. CYDRIX F77 contains several enhancements, in the spirit of the current draft proposal for FORTRAN 8x.

The array JCOEF is often referred to as the list vector, for it points to elements of array x that must be accessed. More generally the above loop and other key loops used in solving sparse systems may be represented as:

```
SPARSE SAXPY        Y(I) = Y(I) + S * A(JCOEF(I))
SPARSE SAXPY (2)    A(JCOEF(I)) = A(JCOEF(I)) - S * Y(I)
SPARSE SDOT         S = S + A(JCOEF(I)) * Y(I)
```

Such loops are executed at high performance rates on the Cydra 5, through use of a robust memory system plus extensive hardware features to support fast gather/scatter and strided access. The Cydra 5 can also execute first order recurrences at near peak execution speed.

ITPACK performance

ITPACK on problem 1; natural ordering

		CPU time in seconds		
Method	Iterations	Cydra 5	CYBER™ 205	CRAY™ XMP
JCG	100	0.455	0.280	0.227
JSI	100	0.465	0.252	0.213
SOR	100	0.614	0.432	0.38

The CRAY and CYBER™ 205 data are taken from research by Oppe and Kincaid[1].

ITPACK for the Cydra 5 is available from Cydrome. For more information contact your Cydrome representative.

Cydra 5 Highlights

- Powerful gather/scatter hardware.
- Overlapped address computation with arithmetic computation.
- Fast computation of recurrences
- Fast computation of conditional statements
- Computation of dot product at near peak speed of the machine.
- IEEE floating point hardware
- Support for both single and double precision arthmetic at high speeds.
- Large main memory, ranging up to 256 megabytes, that reduces or even eliminates the need for I/O.
- Unique memory access capability that suffers little penalty for strided data access.
- Disk capacity exceeding 29 Gigabytes.
- CYDRIX™ operating system based on standard UNIX® System V.3 for ease-of-use.
- State-of-the-art compiler with sophiscated optimization capabilities that enable the application programmer to exploit Cydra 5 computational power in FORTRAN.

Reference

1. T.C. Oppe and D.R. Kincaid, *The Performance of ITPACK on Vector Computers for Solving Large Sparse Linear Systems Arising in Sample Oil Reservoir Simulation Problems*, Comm. in Applied Numerical Methods, Vol. 3, pps. 23-29 (1987).

CYDRA 5™

Application Brief

High Performance Signal Processing

The Cydra™ 5 system offers a powerful computing facility for signal processing applications. Through a joint marketing agreement, Cydrome and Quantitative Technology Corporation (QTC) are making available a highly optimized version of *Math Advantage™*. Among the routines in *Math Advantage* are the following functions:

ACORF	Frequency domain auto-correlation
CCORF	Frequency domain cross-correlation
CFFT	In-place Complex FFT
CFFT2D	Complex 2-d FFT
RFFT	Real-to-Complex FFT
CONV	1-D convolution/correlation
CONV2D	2-D convolution/correlation

This set of functions is widely used in signal processing, image processing, seismic data processing, and in many other data analysis areas.

Two Dimensional Filtering

A common application in signal analysis is the removal of noise from 1-D and 2-D data. The noise or contamination may arise from deficiencies in the collecting equipment, or may represent ambient background noise where measurements are being performed. In seismic data processing, for example, the desired signal is the pressure wave that travels into the earth and is reflected back to the surface by the sub-surface lithology. Waves that travel along the surface (shear waves) represent undesirable noise. On a typical grid size of 2048 points by 64 points, the removal of noise takes only 1.1 seconds of CPU time on the CYDRA 5 Numeric Processor (NP). A time varying filter of dimensions 35 points by 9 points is applied to the data for removal of the noise train (Figure 1). This corresponds to sustained computation rate of 37 megaflops. Single precision arithmetic was used for this task.

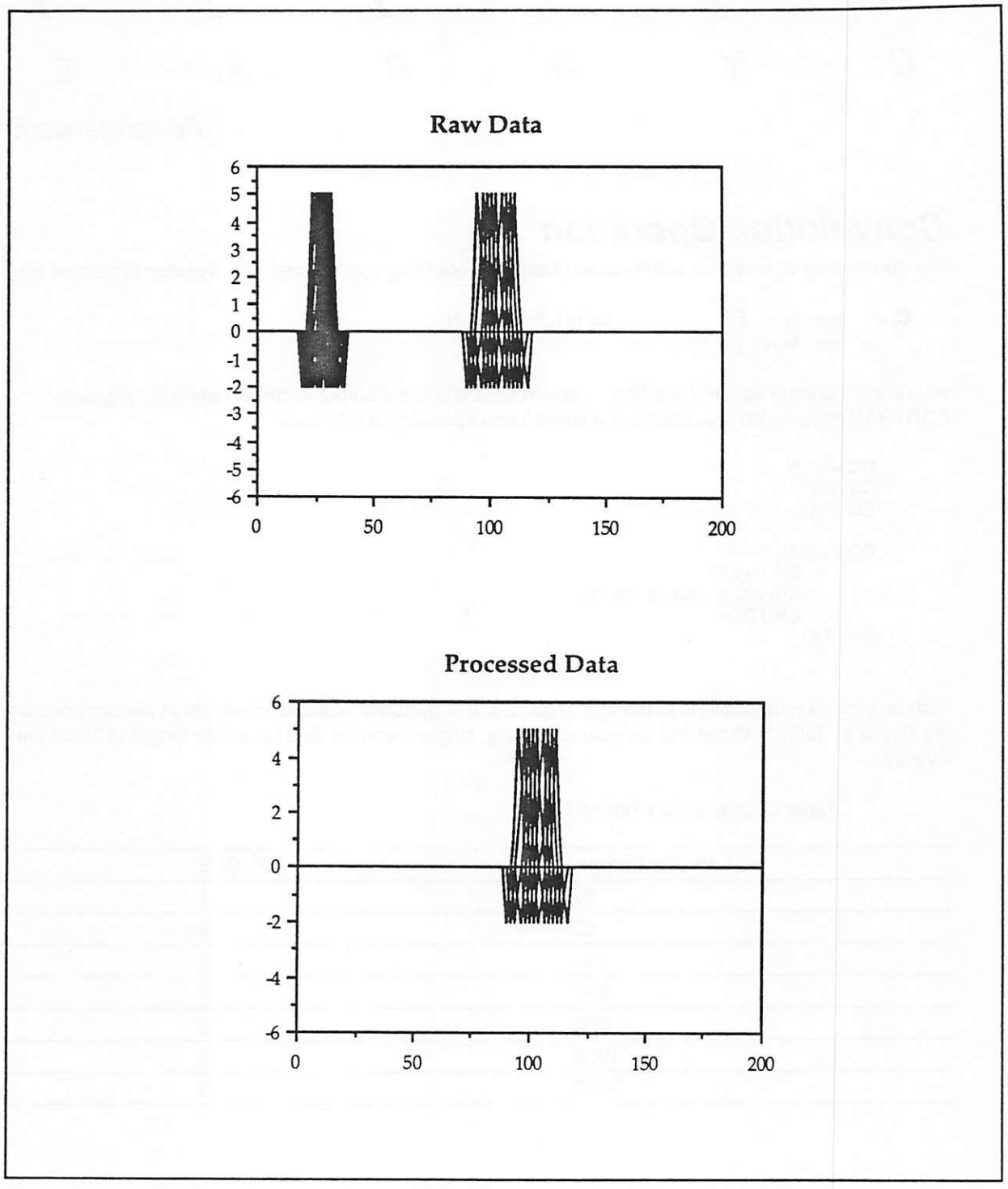

Raw Data
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
0
50
100
150
200
Processed Data
6
4
2
0
-2
-4
-6
0
50
100
150
200

Figure 1.

Convolution Operation

The convolution operation is widely used in signal processing applications. Convolution is defined by:

$$C_i = \sum_{M+i-j} A \quad F_j \qquad \text{for } i=1,2,\ldots\ldots N$$

where *a* is the input vector, *f* is a filter of length M, and *c* is the output vector of length N. A typical FORTRAN code which implements the convolution operation is a follows:

```
DO = I 1,N
C(I) = 0
END DO

DO J = 1,N
        DO I = 1,M
        C(I) = C(I) + A(I+M-J0) * F(J)
        END DO
END DO
```

With only minor modifications to the above code, the convolution operation yields high performance on the Cydra 5. Table 1 shows the convolution timing, single precision data for a filter length of 32 on the Cydra 5.

Table 1. Convolution Timing Test

Vector Length	MFLOPS
20	21
50	29
100	33
250	37
500	38
1000	39
2000	39

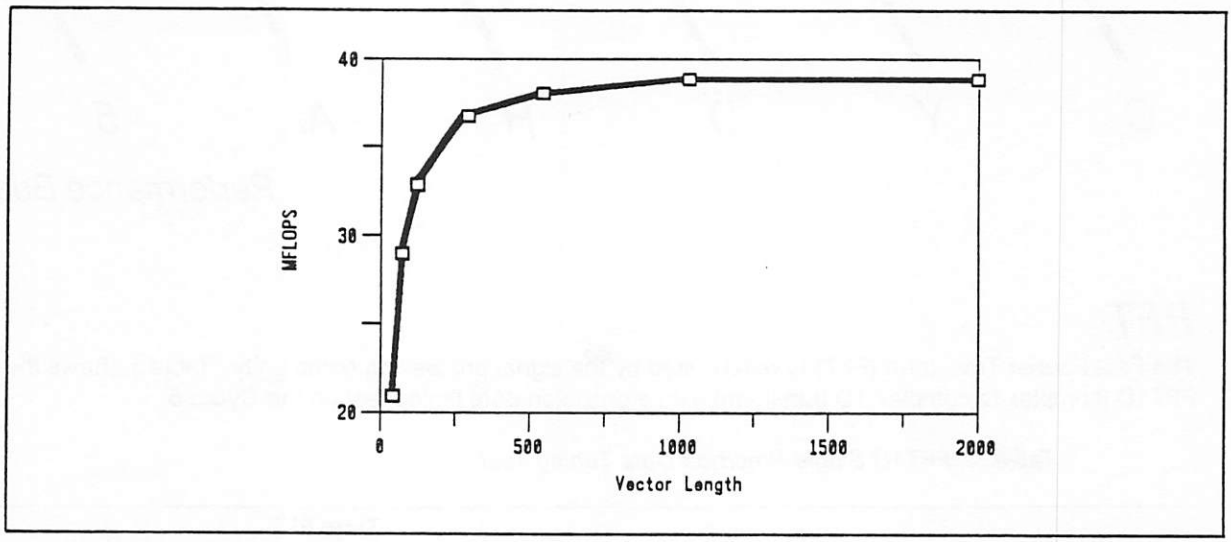


Figure 1. Convolution Operation (Single Precision Data)

Cydra 5 Highlights

- Powerful gather/scatter hardware.
- Overlapped address computation with arithmetic computation.
- Fast computation of recurrences.
- Fast computation of conditional statements.
- Computation of dot produc at near peak speed of the machine.
- IEEE floating point hardware.
- Support for both single and double precision arithmetic at high speeds.
- Large main memory, ranging up to 256 megabytes, that reduces or even eliminates the need for I/O.
- Unique memory access capability that suffers little penalty for strided data access.
- Disk capacity exceeding 29 Gigabytes.
- Cydrix® 5.3 operating system based on standard UNIX® System V.3 for ease-of-use.
- State-of-the-art compiler with sophiscated optimization capabilities that enable the application programmer to exploit Cydra 5 computational power in FORTRAN.

Performance that Counts™

CYDROME®

1589 Centre Pointe Drive
Milpitas, CA 95035
Phone: (408) 945-6300
FAX: 408-262-8938

MKG-AB009-B May 1988

FFT

The Fast Fourier Transform (FFT) is widely used by the signal processing community. Table 1 shows the FFT1D (complex to complex 1D transform) single precision data timing test on the Cydra 5.

Table 1. FFT1D Single Precision Data Timing Test

Vector Length	MFLOPS	Time/FFT (microsec)
32	11	74
64	17	109
128	24	185
256	29	348
512	32	717
1024	33	1533
2048	34	3321
4096	34	7245
8192	34	15695

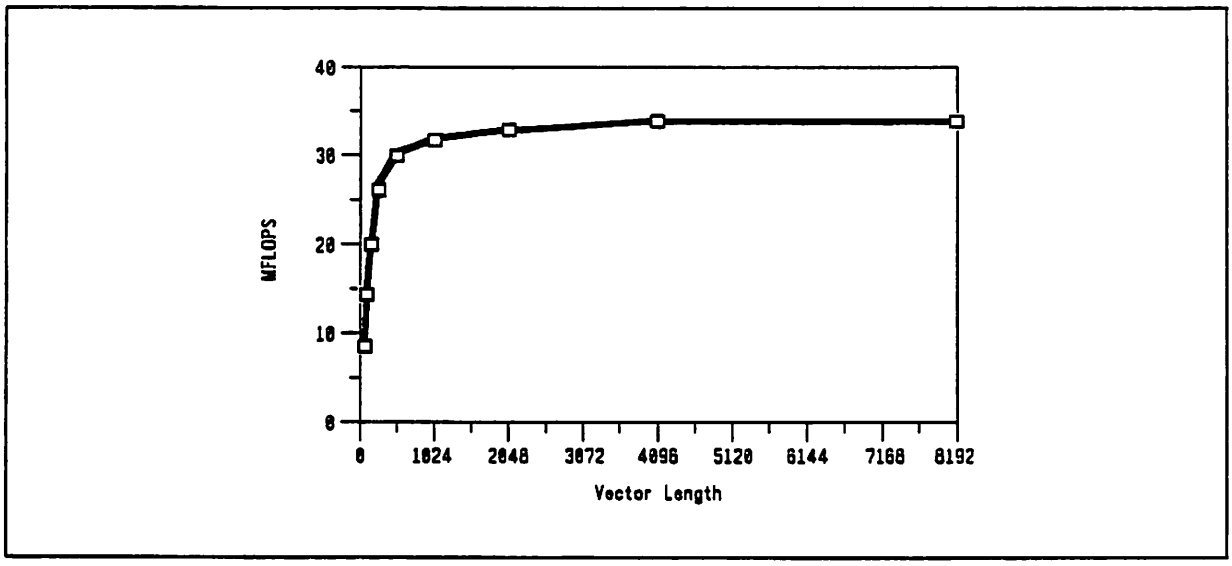


Figure 1. FFT Performance (Single Precision Data)

Application Brief

FIDAP

FIDAP® is a computational fluid dynamics application that is used to simulate a wide variety of incompressible flow problems.

Computational fluid dynamics (CFD) can provide detailed information on the flow of matter, energy, momentum, and other properties from point to point within a system. The techniques of CFD are widely used in the automotive, aerospace, discrete manufacturing, electronic devices, chemical, and other industries. Table 1 summarizes typical uses of FIDAP in various industries.

Table 1. Use of CFD Techniques In Various Industries

Industry	Applications
Automotive and Aerospace	Aerodynamics analysis to reduce drag Combustion process analysis Flow and thermal modeling for hydraulics
Electronic	Component heating/cooling behavior Fan sizing, cabinet design
Food	Pasteurization time studies
Power Utilities	Sizing of cooling systems.

FIDAP uses the finite element method for performing the computations. FIDAP is comprised of three modules: FIPREP, FIDAP, and FIPOST. FIPREP generates the analytical mesh and accepts input for physical system properties and initial conditions. The main module, FIDAP, transforms the governing partial differential equations into algebraic equations (matrix algebra) with iterations and time stepping. FIPOST provides graphical post-processing of output field variables such as temperature. FIPREP and FIPOST may be executed in interactive or batch mode; FIDAP is executed in batch mode. Execution time can range from a few minutes to multiple hours depending on the problem size.

The Cydra 5 system is well suited for the execution of FIDAP. For 2D models the memory requirements are in the 4 to 8 MB range with scratch disk space in the 30 to 70 MB range. 3D models require roughly 6 to 12 MB memory and between 100 to 300 MB of scratch disk space. The Cydra 5 with a large memory size of up to 256 MB has two additional attributes which make it a powerful processor for executing FIDAP:

- Execution of the dot product (inner product) at high speed
- Powerful memory access system, featuring memory access rates that are insensitive to access pattern (stride).

The dot product operation, typically, cannot be performed very efficiently on traditional vector computers. Summation of successive terms has an implied recurrence. Table 2 shows the computation rate of the dot product operation on the Cydra 5.

Table 2 . Dot Product Operation on the Cydra 5 (in megaflops)

Vector Length	100	500	1000	5000
Single Precision	10.0	27.5	35.0	46.9
Double Precision	8.0	17.5	20.6	22.2

The dot product operation is the principal basic computation in FIDAP, often performed on sparse or banded matrix data structures.

Table 3 shows Cydra 5 performance results on a series of FIDAP test cases. The cases span a range of problem sizes, from small through relatively large. Comparative data on a Convex™ C1 XP is also shown; data has been provided by Fluid Dynamics Incorporated.

Table 3. FIDAP Version 4 Performance Data (in seconds)

Test Case	Cydra 5	Convex C1	Ratio*
8	85.6	183.8	2.1
11	5854.3	16774.6	2.9
15	3916.7	8670.6	2.2
16	17589.1	41852.4	2.4
20	2074.9	5452.9	2.6

* Comparison of execution rate of Cydra 5 and the Convex C1. The Cydra 5 is 2 to 3 times faster than the Convex C1.