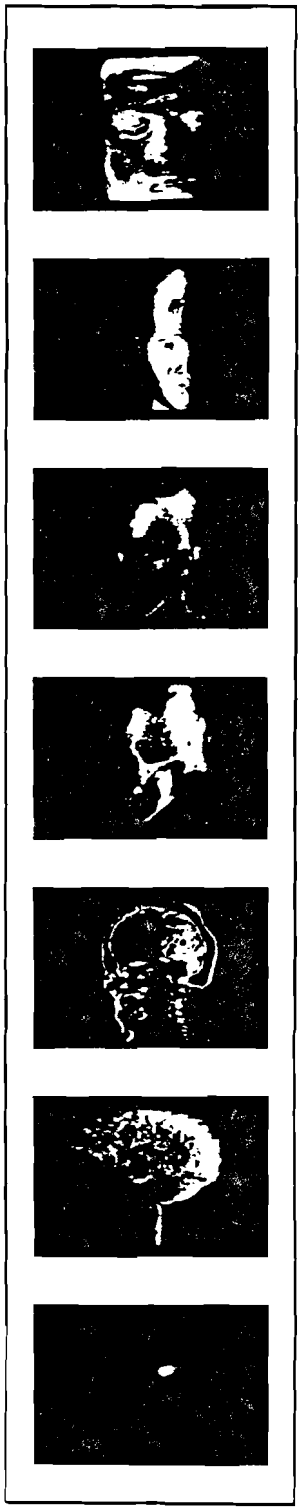
IBM



dkfz

APL2 and Vector Facility Experiences in Image Processing

August 2nd, 1989

U. Schauer

IBM Wissenschaftliches Zentrum Heidelberg (WZH)
Tiergartenstr. 15
6900 Heidelberg 1

Dr. H.P. Meinzer

Deutsches Krebsforschungszentrum (DKFZ)
Abt. Medizinische und Biologische Informatik
Im Neuenheimer Feld 280
6900 Heidelberg 1

APL2 and Vector Facility
Experiences in Image Processing

Under a study contract with IBM the German Cancer Research Center at Heidelberg (DKFZ) has thoroughly investigated the vectorisation opportunities of their broad spectrum of APL-based application programs, including measures to achieve additional performance gains.

We will present measurements on the speedup obtained by the vector feature (VF) on DKFZ's IBM 3090-150 and also report on further improvements through special coding techniques for some of the compute intensive kernels in their applications. Dramatic gains could be achieved by recoding small parts of the application in FORTRAN or even Assembler.

An assessment of APL for the application spectrum at DKFZ is given as a summary, pointing out the importance of finding and isolating the right primitives for a range of applications.

General suggestions for efficient APL coding under the presence of VF are also given as a conclusion. This includes experiences gained from investigating applications of different fields, as e.g. finance and insurance.

The German Cancer Research Center

To defeat cancer has become one of the most urgent problems of human society. All disciplines of science are challenged to cooperate in a research program towards this ultimate goal.

The DKFZ (Deutsches Krebsforchungszentrum) was founded in 1964 on initiative of the surgeon Prof. Dr. K.H. Bauer. It is a national research institution sponsored by the central and state government (Bundesregierung und Landesregierung Baden-Württemberg). Since 1976 the DKFZ combines eight institutes with a total of currently 37 scientific departments.

Research of the DKFZ is oriented towards understanding how and why cancer develops in order to workout proven concepts for diagnosis and therapeutics. Four focal points evolved from this endeavor:

- Tumor biology
- Mechanisms in control of cancer
- Factors causing cancer and preventive measures
- Means for diagnosis and therapeutics

The DKFZ works closely together with the nearby clinics of the university Heidelberg (e.g. tumor center Heidelberg/Mannheim) and with practicioner physicians and specialists outside. It has its own administration and central services (library, spectroscopy, laboratory, data processing, etc). About one third of its 1400 employees are scientists.

The department for medical and biological informatics, headed by Prof. Dr. O.C. Köhler owns a broad spectrum of APL based application programs, developed over a period of more than 13 years. One major application is the reconstruction of 3-D picture sequences from digitized parallel recordings, taken by computer tomography (CT) or magnetic resonance (MR) machinery. The general objective is to better support medical diagnosis and therapy via imaging techniques by providing physicians with better visualisation. The constructed 3-D pictures enable physicians to see details which have been left undetected before.

Study Contract IBM — DKFZ

During 1988, under a study contract, the department for Medical and Biological Informatics (MBI) of the DKFZ worked jointly together with the nearby IBM Scientific Center Heidelberg (WZH). The objective was to evaluate the performance gain of its APL applications, particularly those for 3-D image construction, when they are run on DKFZ's IBM 3090-150 with Vector Facility (VF). Development of measures to improve the performance gain was also part of the study contract.

It was the goal to develop some methodology allowing to take full advantage of VF without much reprogramming (minimum effort for maximum gain). The following representative set of compute-intensive programs, focussing on 3-D image reconstruction, was selected for detailed investigations:

1. Image processing exploiting methods of artificial intelligence (AI)
   - Topological map
   - APLPROLOG
2. Image processing based on morphological operations
   - Erosion (erasure)
   - Dilation (enhancement)
   - Faltung (convolution)
   - Calcification (mammography)
   - 3-D visualisation
3. Simulation of the growth of crypta (no image processing)

3-D Visualisation in Medicine

CT and MR recordings contain immense amounts of detail, difficult to interpret even for specialists in the field. 3-D visualisation is a means to enhance pertinent information and present it in a form, which is easy to comprehend since it conforms to the spectator's visual experience. Parts of the represented objects may be removed with proper segmentation algorithms. The physician can thus perceive details of the interior of his patient by means of computer visualisation. One set of recordings can be used repeatedly for a variety of investigations. This allows for improved diagnosis and therapeutics.

Realistic 3-D representation of volume elements with different transparency is a complex computational task, exceeding a simple geometrical transformation. It is performed by calculating the distribution of light and shadow caused by the investigated volume elements (voxels) under assumed sources of light. This is illustrated by the following set of pictures (which can only give a vague impression of the actual representation in color and motion on the screen).

1
2
3
4
5
6
7
(c) dkfz mbl, heidelberg

Methodology

The application base had been largely developed under VS APL. The migration to APL2 presented no serious problems, except for performance of take and drop operations when they were applied to the first axis of three dimensional arrays. Alternative coding for take and drop operations was necessary to circumvent this performance bottleneck.

The APL2 applications were then vector-ready with expectedly high performance gains (up to 50%). Hot spot analysis by means of the *TIME* function was very helpful for further optimisation, which was done by a sequence of maximally three steps.

1. Investigate better alternatives for time critical lines of code.
2. Consider also algorithmic replacements by using external functions available in libraries, particularly from the Engineering and Scientific Subroutine Library (ESSL).
3. Identify important application specific basic operations. Coding those basic operations in FORTRAN or ASSEMBLER may significantly improve exploitation of VF.

Results of the Study

ESSL routines compared to equivalent APL functions often showed a performance gain by an order of magnitude. The following summary shows that this expectation was also met for applications which could not be speeded up via ESSL.

The following variations of the test environment APL2 Release 3 were used in the summary below:

1. Code as is without VF (this is the 100% basis)
2. Code as is with VF on (APL2 is vector-ready!)
3. Code with optimized critical lines and VF on (optimisation step 1)
4. Code further optimized by algorithmic replacements, e.g. using ESSL subroutines and VF on (optimisation step 2)
5. Code finally optimized by developing specialised basic operations in FORTRAN or ASSEMBLER and VF on (optimisation step 3)

Summary of Results

Test variation	1	2	3	4	5
Topological Map	100%	91%	60%	35%	12%
Faltung	100%	50%	50%	—	5%
APL PROLOG	100%	100%	89%	—	—
Erosion	100%	66%			
Dilation	100%	66%			
Calcification	100%	75%	48%		
3-D Visualisation	100%	75%	49%		
Simulation model	100%	86%			

Empty boxes indicate that a final conclusion has not been reached. Additional improvements are still conceivable but may need significant changes to the algorithmic approach.

Optimisation step 3 as applied to Topological Map and Faltung will be further discussed below. Both these applications are very compute intensive and of critical importance in many composed applications.

APLPROLOG has been developed exploiting ideas from J.Brown, E.Eusebi, J.Cook, L.H.Groner: Algorithms for Artificial Intelligence in APL2, IBM TR 03.281. Since it basically uses string manipulations on nested arrays there was no performance gain by turning VF on. However, the hot spot analysis lead to some improvements by changing time critical lines of code.

Erosion and Dilation are both vector-ready with a performance gain of one third. Further significant improvements by optimisation step 3 are conceivable.

The calcification (mammography) and 3-D visualisation can gain a lot under optimisation step 1. This is representative for many other compute intensive applications and also the performance gain of 50% is quite normal.

The simulation of growth of crypta needs further elaboration. Additional significant performance gains are not obvious from the *TIME*-logs.

Topological Map

One function *TOPOMIN* was responsible for more than 90% of the time spent in the application Topological Map.

```
    ∇
[0]   R←A TOPOMIN B;C;D;I;Y      ⍝ NON-BUFFERED VERSION IMPROVED
[1]   R←(2,I←↑Φ⍴B)⍴0.1           ⍝ UNDER OPTIMISATION STEP ONE.
[2]   C←+/A*2                    ⍝ THIS HAS BEEN SEPARATED FROM THE LOOP!
[3]  L:R[;I]←(Y⍳D),D←⌊⍉Y←|(C++⍉B[;I]*2)-2×A+.×B[;I]
[4]   →(0<I←I-1)/L
    ∇
```

A significant performance gain was achieved by moving calculations out of the innermost loop. Exploitation of ESSL also helped a lot. But even then a high potential for improvement was left over, which could be activated by recoding in FORTRAN (avoiding much of the data movement going on under APL).

```
      SUBROUTINE TOPOMIN(L,M,N,A,B,H,R)
C   Optimized Version (U.Schauer 9/14/88).
C   Utilises  ESSL-Programm  IDAMIN.
C   Note that FORTRAN stores columnwise, APL rowwise.
C   H is an auxiliary storage of APL-Dimension 2,M.
C   A is assumed transposed compared to the APL function.
C
      REAL*8 H(M,2),A(M,L),B(N,L),R(N,2),C,T1,T2,T3
      DATA C/-0.5/
      DO 15 I=1,M
        T1 = 0
        DO 10 J=1,L
 10       T1 = T1 + A(I,J) * A(I,J)
 15     H(I,1) = T1
      DO 25 I=1,N
        T2 = 0
        DO 20 J=1,L
 20       T2 = T2 + C * B(I,J) * B(I,J)
 25     R(I,1) = T2
      DO 40 K=1,N
        DO 35 I=1,M
          T3 = C * H(I,1) + R(K,1)
          DO 30 J=1,L
 30         T3 = T3 + A(I,J) * B(K,J)
 35       H(I,2) = T3
        II = IDAMIN(M,H(1,2),1)
        R(K,1) = II
 40     R(K,2) = 2 * DABS(H(II,2))
      END
```

Before using the FORTRAN subroutine one has to provide a vector-ready compiled version and also to describe the formal interface by an entry in the pertinent names file, say NAMES, with the following tags

```
:NICK.TOPOMIN_     :MEMB.TOPOMIN    :LINK.FORTRAN    :INIT.FORTRAN
      :RARG.(G0 1 7)(I4 0)(I4 0)(I4 0)(>E8 2 * *)(>E8 2 * *)
      :RARG.(>E8 2 * *)(>E8 2 * *)
```

The function can than be used under APL by building the name association first and then providing the necessary parameters before actually calling the FORTRAN written TOPOMIN. Remember that A has been assumed transposed and that the result is delivered in R.

```
3 'NAMES' ⎕NA 'TOPOMIN_'
(L M)←⍴A
N←↑Φ⍴B
H←(2,M)⍴.1
R←(2,N)⍴.1
TOPOMIN_ L M N 'A' 'B' 'H' 'R'
```

Faltung

Faltung (a moving local averaging operator) often applied for filtering of data is one of the most heavily used operations in image processing. Therefore it was already coded in VS APL close to perfection timewise. It performed extremely well when VF was turned on. Nevertheless, with some basic understanding of the VF principles of operations it was obvious that there was much room for further

improvement. However, neither APL nor FORTRAN were suitable languages to achieve the furtber performance gain by a factor of 10. One really has to code the innermost processing loop in ASSEMBLER. The performance gain is the product of several factors. A factor of three immediately results from better utilisation of VF-operations (multiply add can be used without load and store). The remaining gain is by reduced data movement and improved locality of the changed algorithm (less cache misses).

The challenge of performance optimisation is, not to miss similar opportunities. Key is a basic understanding of the VF operations and the APL interpreter. A side effect of ASSEMBLER recoding is also worth mentioning. It allows complete control over mixed data representations (e.g. Single and Double precision) and thus can lead to savings in storage requirements. Therefore the ASSEMBLER program is not only superior in performance timewise it also is much superior storagewise allowing in place operation.

```
    ∇
[0]    Z←M FALTUNG A;I;J;K;R;⎕IO
[1]  ⍝ OPTIMIZED FALTUNG.
[2]    ⎕IO←0
[3]    I←((↑⍴A),1)+.×¯1+⌈.5×⍴M
[4]    J←I-,((↑⍴R←⍴A)×⍳↑⍴M)∘.+⍳¯1↑⍴M
[5]    Z←(⍴A←,A)⍴0
[6]    →(0=K←↑⍴M←(I←0≠M)/M←,⍴M)/L0
[7]    J←I/J
[8]  L:Z←Z+M[K]×J[K←K-1]⌽A
[9]    →(0<K)/L
[10] L0:Z←R⍴Z
    ∇
```

Vector-Ready Coding Under APL2 Release 3

The following advices cannot provide as much insight as extensive own experimentation. However, they address some of the basic principles which may enable significant improvements.

- Prefer recognized idioms
- Avoid unnecessary operations, particularly concatenation, negation, etc.
- Think parallel! Prefer array operations to loops whenever possible.
- Optimize arithmetical expressions! Prefer multiplication and addition to division and subtraction whenever possible.
- Avoid parenthesis levels and indexing in vector expressions to enable chaining.
- Avoid unnecessary repeated calculations. Move common expressions out of innermost calculation kernels.
- Avoid unnecessary data movement! Prefer indexing to take/drop operations and concatenation.
- Prefer data type real, when it helps to avoid conversion (e.g. by adding 0.).
- Prefer rowwise access to binary data; columnwise access is rather inefficient.
- Prefer integer data type, when binary arrays require columnwise access (e.g. by adding 0).
- Avoid unnecessary nesting! Prefer homogeneous data arrays whenever possible.
- Check carefully, whether rowwise or columnwise storage is preferable.
- Use library programs e.g. from ESSL instead of compute intensive APL code.

- Prefer vectorised library programs instead of non-vectorising APL programs and operators (e.g. DGEF/DGES from ESSL instead of "domino" matrix division, "⌹").
- Develop application specific primitives in FORTRAN or ASSEMBLER, when it significantly improves performance. Don't forget to specify the "VECTOR" option during the FORTRAN compilation.

Summary

Using ESSL-subroutines or developing application specific primitives in FORTRAN or ASSEMBLER may create the highest performance gain for vector-ready applications. Therefore the identification of adequate primitives should never be missed in the performance optimisation process.

Such optimized APL applications may then come close to the performance obtained by complete rewriting in FORTRAN. The need for APL-compilation in total is drastically reduced at a marginal increase of programming effort.

APL89 Conference, August 1989