Digital Image Processing Exercises in the Fourier Domain using VECLIB on Vector Computer CONVEX 3820

Kandó Kálmán Polytechnic
Institute of Computer Technology, Székesfehérvár
and
University of Athens

Magda Kovács, Senior Lecturer
Ágnes Burián, Senior Lecturer

Introduction

The image processing algorithms in these exercises serve as examples for understanding vectorization and parallelization as well as optimization. The programs use CONVEX VECLIB library subprograms, therefore a short overview of VECLIB is included in this chapter.

The students use a frame program for implementing digital image processing algorithms in the Fourier Domain. A sample program for inverse filtering is implemented.

Image Representation in the Programs

The digital image can be conveniently represented by an N*M matrix I of the form:

```
          i (1,1)  i(1,2)   ...      i (1,M)
          i (2,1)  i(2,2)   ...      i (2,M)
i =         .
            .
            .
          i (N,1) i(N,2)   ...      i (N,M)
```

The matrix elements (image pixels) are integers in the range [0,...,255] for 8 bit images. Therefore, they can be represented as characters in FORTRAN language. In certain cases, when complex arithmetic is involved, image pixels must be represented as complex numbers in FORTRAN

```
character a (N,M)
complex   b (N,M)
```

The matrix elements have the form a (i,j), b (i,j), 1 <= i <= N, 1 <= i <= M.

Elementary digital image processing operations

Elementary digital image processing includes basic arithmetic operations, e.g. image addition, substraction:

$$c(i,j) = a(i,j) + b(i,j)$$
$$c(i,j) = a(i,j) - b(i,j)$$

and multiplication of an image by a constant:

$$b(i,j) = c * a(i,j)$$

Both input matrices a, b and output matrix c are of the type image. Therefore, special attention must be paid, so that the output image elements are in the range (0,...,255]. If an overflow occurs, the corresponding pixel is black on a white background. In the case of an underflow, the corresponding pixel appears white on a is black background. Both overflow and underflow create visually unpleasant effects.

Fourier Domain Operations

Background

Digital Image Transform

Image transforms play an important role in digital image processing as a theoretical and implementation tool in numerous tasks, notably in digital image filtering, restoration, encoding and analysis. Image transforms are often linear ones. They are represented by the transform matrices A:

$$X = A\,x$$

where x, X is the original and the transformed image respectively. In most cases, the transform matrices are unitary:

$$A^{*} = A$$

In the case of the two-dimensional transforms, the basis vectors correspond to basis images. Thus, a transform decomposes a digital image to a weighted sum of basis images.

Commonly used tranforms:

Discrete Fourier Transform (DFT)
Discrete Cosine Transform (DCT)
Walsh transform
Haar transform
Slant transform
Discrete Sine Transform (DST)

A DFT possesses very interesting theoretical properties and it is extensively used in digital filter implementations and in power spectrum estimation.
The DCT is frequently used in transform image coding shemes, because it is excellent tool for digital image compression.

Noise generators for digital image processing

Digital images are corrupted by noise either during image acquisition or during image transmission. The image acquisition noise is photoelectronic noise or film-grain noise. It can be proven that in both cases the noise is signal-dependent. Another type of noise, which usually appears during image transmission is the salt-pepper noise. It appears as black and/or white impulses on the image. It is impulsive noise and its source is either atmospheric or man-made.
Digital image noise is removed by using noise filtering techniques. In many cases, digital images must be corrupted artifically in order to assess the performance of various filtering techniques.
Such noise generators that can be produce additive noise or multiplicative noise:

$g(i,j) = f(i,j) + n(i,j)$
$g(i,j) = f(i,j)\, n(i,j)$

or impulsive noise:

$$g(i,j) = \begin{cases} z(i,j), & \text{with probability } p \\ f(i,j), & \text{with probability } 1-p \end{cases}$$

Digital image filtering

Images are acquired by photoelectronic or photochemical methods. The sensing devices tend to degrade the quality of the digital images by introducing noise, geometric deformation and/or blur due to motion or camera misfocus.. One of the primary concerns of digital image processing is to increase image quality and to moderate the degradations introduced by the sensing and acquisition devices. Image restoration techniques are concerned primarily with the reconstruction or recovery of an image that has been degraded.
Digital image enhancement techniques increase subjective image quality by sharpening certain image features (e.g. edges and boundaries, contrast) and by reducing noise. Digital enhancement and restoration operations can be thought as two-dimensional digital filters. They can be distinguished in two large classes:

- linear digital filters
- nonlinear digital filters.

Linear digital image filter can be designed and / or implemented either in the spatial domain (spatial operation) or in the frequency domain (transform operations.
The majority of digital restoration filters are linear ones and are usually implemented in the frequency domain.

Inverse filter implementation

The aim of digital image restoration is the recovery of an image that has been degraded by the sensing instruments. Degradation sources usually produce motion blur, defocus blur and/or noise introduced during sensing. Image sensing is complicated process and its modeling is cumbersome. Such a model is described by the following equation:

g (n1,n2) = f (n1,n2) ** h (n1,n2) + n (n1,n2)

where

f is the original image,
g is the observed image ,
h is the blur point spread function
n is the sensing noise

The problem of image restoration is to estimate f from the observed image g, possibly based on some a priori knowledge about the blur point spread function and on the statistics of the noise n and of the images g, f.
Let us suppose that the formation noise n is negligible. The image formation model takes the following format in the frequency domain::

$$G(\omega 1, \omega 2) = F(\omega 1, \omega 2)\, H(\omega 1, \omega 2)$$

If the degradation function H ((ω1, ω2) is a priori known, the simplest approach to digital image restoration is inverse filtering:

$$F(\omega 1, \omega 2) = G(\omega 1, \omega 2) / H(\omega 1, \omega 2)$$

The most straightforward implementation of the inverse or the pseudoinverse filter is by using the Discrete Fourier Transform:

f (n1,n 2) = IDFT [G(k1,k2)/ H (k1, k2)

Where G(k1,k2) is the DFT of the observed image g(n1,n2) and H(k1,k2) is the DFT of the blur point spread function h(n1,n2).

CONVEX VECLIB

CONVEX VECLIB (and VECLIB8) is a collection of FORTRAN-callable subprograms optimized for use on the CONVEX family of supercomputers. This library provides mathematical software and computational kernels for application programs involving arrays.

These libraries contain subprograms for:

- dense vector operations, including the Basic Linear Algebra Subprograms (BLAS)
  sparse vector operations, including the Sparse Blas
- matrix operations, including the Level 2 and Level 3 BLAS
- linear eqation solution, including LINPACK
- eigensystem solution, including EISPACK
- sparse symmetric linear equation solutions
- sparse symmetric ordinary and generalized eigensystem solutions
- skyline linear eqations
- discrete Forier transforms
- convolution and correlation
- linear recurrences
- miscellaneous tasks, such as manipulating dynamic memory, sorting and generatingrandom numbers

A VECLIB and VECLIB8 libraries are compiled subprograms ready for you to incorprate into your programs with the linker. Simply include the appropriate declaration and CALL statements in your FORTRAN source program and specipy that VECLIB or VECLIB8 be used as an object library at link time, by using the –l option on the fc command line, as follows:

fc [options] file –lveclib

or

fc [options] file –lveclib8

Optimization

Convex VECLIB is highly efficient. It takes full advantage of the CONVEX tightly integrated vector, scalar, and parallel architecture. Most VECLIB subprograms have been coded in assembly language, but are completely compatible with standard FORTRAN programs. Because you can easily incorporate these kernels into the computationally intensive parts of programs, you receive the performance benefits of highly tuned assembly language without having to become an expert in the CONVEX architecture or assembly-language programming. In this respect, VECLIB is an extension of the FORTRAN language.

Parallel Processing

Parallel processing is available on CONVEX C2 and C3 Series computers that have multiple processors. Theses can divide a single computational process into small streams of execution, called threads. The result is that you can have more than one processor executing on behalf of the same process.

To support parallel processing, the CONVEX C2 and C3 Series hardware employs automatic self-allocating processors (ASAP) that effectively manage CPU assignment. When another processor can be used to assist in executing a process, the active thread posts a request for any idle CPU(s) to help. New threads are created and executed by available CPUs. If no CPUs are available, the original thread executes all the work for the process.

CONVEX VECLIB works in both single processor (CONVEX C1 Series) and parallel processor (CONVEX C2 and C3 Series) environments. At runtime, a CONVEX VECLIB subprogram determines whether it is being used on a CONVEX C1, C2, or C3 Series processor. If it is running on a CONVEX C2 and C3 Series processor with multiple heads, it detects if the program is already using multiple threads. It uses this information to automatically choose between a single or parallel processor algorithm. Consequently, you can move programs that use CONVEX VECLIB between CONVEX C2 and C3 Series systems freely, without losing compatibility or the advantage of either architecture.

If you computing on a CONVEX C2 and C3 Series system with multiple processors, you can realize the performance benefits of parallel processing in two ways. First, you can simply call any parallelized CONVEX VECLIB subprogram and let it employ parallelism internally.

Alternatively, you can call CONVEX VECLIB subprograms in a parallelized loop or region. To obtain parallelism via the second mechanism, you need to be familiar with the concept of reentrancy and with the FORCE_PARALLEL, BEGIN_TASKS, and END_TASKS compiler directives.

CONVEX VECLIB subprograms are reentrant, which means that they may be called several times in parallel to do independent computations without one call interfering with another. You can use this feature to call CONVEX VECLIB subprograms in a parallelized loop or region. The compiler does not automatically parallelize loops containing a function reference or subroutine call. You can force it to parallelize such a loop by inserting a FORCE_PARALLEL compiler directive before the loop. For example, the following code makes parallel calls to VECLIB subprogram SAXPY:

```
C$DIR FORCE PARALLEL
      DO 10 J = 1, N
            CALL SAXPY ( N - I,  A ( I, J ), A ( I + 1, J ), 1, A ( I + 1, J ), 1 )
10    CONTINUE
```

While optimizing a parallel program, you might want to make parallel calls to a CONVEX VECLIB subprogram to perform independent operations, but where the call statements are not in a loop. THE FORTRAN compiler does not automatically parallelize code outside a loop, but you can use the BEGIN_TASKS, NEXT_TASK, and END_TASKS compiler directives to tell the compiler to parallelize such code. For example, the following code makes parallel calls to subprogram SNRM2:

```
C$DIR BEGIN_TASKS
      XNORM = SNRM2 ( NX, X , 1 )
C$DIR NEXT_TASK
      YNORM = SNRM2 ( NY, Y , 1 )
C$DIR END_TASKS
```

Profiling CONVEX VECLIB Applications

The CONVEX Performance Analyzer, Cxpa, is an interactive tool that gathers and analyzes program execution timing (profiling) data. Cxpa provides the programmer with the means to study the timing behavior of a program for the purposes of optimizing, benchmarking, and debugging. To use the performance analyzer, you must first compile your FORTRAN program with either the –pa, -pab, or –par compiler option. These options instrument the compiled program so that its performance can be measured at the subprogram level, the loop level, the block level, or the region level.

CONVEX VECLIB has been instrumented at the subprogram level so that the performance of VECLIB subprograms can be included in the analysis. This instrumentation is noninstrusive, so it is not necessary to use a different version of CONVEX VECLIB when you desire to profile your program. Also, the Cxpa instrumentation does not interfere with the prof, bprof, or gprof instrumentation in your program. However, you may not profile your program with both Cxpa and prof, bprof, or gprof at the same time.
Subprogram-level profiling produces summary information about the subprograms that are called during profiled execution of the program. This information includes:

- The number of times each subprogram is called
- The CPU time in the each subprogram and the percentage of the program total, either including or excluding the cumulative time in called subprograms
- Dynamic call graph, listing the subroutine calls that take place within a computer program.

VECLIB Naming Convention

The name of each CONVEX VECLIB subprograms is a coded specification of its function, within the limits of standard Fortran-77 6-character names. Usually, the first character of a subprogram name, denoted by „T”, shows the predominant data type

according to Table 1:

Table 1:

T	Data Type	VECLIB	VECLIB8
S	Single Precision	REAL*4	REAL*8
D	Double Precision	REAL*8	-
I	Integer	INTEGER*4	INTEGER*8
C	Complex	COMPLEX*8	COMPLEX*16
Z	Double Complex	COMPLEX*16	

For example, subprograms that compute the sum of vector elements are named according to data type: SSUM, DSUM, ISUM, CSUM, and ZSUM for the VECLIB library, and SSUM, ISUM, and CSUM for the VECLIB8 library. Some function and the second and the second indicating the type of data on which it operates.

To support changing the precision without changing the code, the CONVEX FORTRAN Compiler provides several compilation options, namely, –cfc, -p8, and –pd8, that affect the size of FORTRAN data types. By taking care in your use of FORTRAN data type declarations, you can write a program that will compile with one set of compiler option and run correctly in 32-bit precision with the VECLIB library or compile with another set of compiler option and run correctly in 64-bit precision with VECLIB8.One constraint is that the program must not use any D or Z subprograms from CONVEX VECLIB, because the D and Z subprograms are not included in VECLIB8.

Data Item Byte Length vs. Declaration and Compiler Option

Fortran Declaration	Fortran Compiler Option			
	none	-cfc	-p8	-pd8
INTEGER	4	8	8	8
INTEGER*4	4	8	4	4
INTEGER*8	8	8	8	8
Integer by default	4	8	8	8
REAL	4	8	8	8
REAL*4	4	8	4	4
REAL*8	8	8	8	8
Real by default	4	8	8	8
DOUBLE PRECISION	8	16	16	8
Double Precision constant	8	16	16	8
COMPLEX	8	16	16	16
COMPLEX*8	8	16	8	8
COMPLEX*16	16	16	16	16
Complex constant	8	16	16	16
DOUBLE COMPLEX	16	16	16	16
Double Complex constant	16	16	16	16
LOGICAL	4	8	8	8
LOGICAL*4	4	8	4	4
LOGICAL*8	8	8	8	8
Logical constant	4	8	8	8

Exercises

Exercise 1. Study and comment the following program:

Source program

```
      program FFT_CONV
      parameter(n=256)
      integer l1,l2,ldxy,iopt,ier,x_int
      complex*8 xc_image(n,n),xc_blur(n,n),xc_fir(n,n)
      character image(n,n),blur(n,n),new_image(n,n)
      character*20 in_file, out_file
      integer i,j,xi,errc
      common err_code,in_file,out_file
      err_code=0

      print *,'          CONVOLUTIONS'

      print *,'Input image file?'
      read *,in_file
      call raw_input(image,xc_image,n)
      call ERROR

      print *,'Input blur file?'
      read *,in_file
      call raw_input(blur,xc_blur,n)
      call ERROR

c*************************************************************
c*
c*      l1=n    *  Number of rows of data
c*      l2=l1   *  number of colums of data
c*      ldxy=n  *  The leading dimension of arrays x and y with
c*                   ldxy>=l1
c*      iopt=0  *  >=0 forward transform, <0 reverse transform
c*
c*************************************************************

      l1=n
      l2=l1
      ldxy=n

      iopt=0
      call crc2ft(xc_image,l1,l2,ldxy,iopt,ier)
      if (ier .eq. 0) then
         print *,'Transformation of the image successful'
      else
         err_code=1
         call ERROR
      endif
```

```
      iopt=0
      call crc2ft(xc_blur,l1,l2,ldxy,iopt,ier)
      if (ier .eq. 0) then
         print *,'Transformation of the blur successful'
      else
         err_code=2
         call ERROR
      endif

      call convolutions(xc_image,xc_blur,xc_fir,n)

      iopt=-1
      call crc2ft(xc_fir,l1,l2,ldxy,iopt,ier)
      if (ier .eq. 0) then
       print *,'Inverse transformation successful'
      else
         err_code=3
         call ERROR
      endif

      print *,'Output image file?'
      read *,out_file

      call raw_output(new_image,xc_fir,n)
      call ERROR

      err_code=100
      call ERROR

      end

c-----------------------------------------------------------------

      subroutine raw_input(matrix,xc_matrix,n)
      character elem,matrix(n,n)
      integer eof,i,j,m,xi,err_code
      character*20 in_file,out_file
      complex xc_matrix(n,n)
      common err_code,in_file,out_file

      open (unit=10,file=in_file,status='old',form='unformatted',
     +           access='direct',recl=1,err=118)
      print *,'read...'
```

```
      m=1
      do i=1,n
         do j=1,n
            read (10,rec=m,iostat=eof) elem
            m=m+1
            if (eof .eq. 0) then
               matrix(i,j)=elem
            else
               go to 10
            endif
         end do
      end do
   10 continue

      do i=1,n
         do j=1,n
            xi=ichar(matrix(i,j))
            xc_matrix(i,j)=cmplx(xi,0)
         end do
      end do
      go to 18

  118 err_code=118
   18 return
      end


c*************************************************************
c*                          CONVOLUTIONS
c*
c*************************************************************

      subroutine convolutions(xc_image,xc_blur,xc_fir,n)
      complex xc_fir(n,n),xc_image(n,n),xc_blur(n,n)
      integer n

      do i=1,n
         do j=1,n
            xc_fir(i,j)=xc_image(i,j)*xc_blur(i,j)
         end do
      end do
      return
      end
```

```
c-----------------------------------------------------------------

      subroutine raw_output(matrix,xc_matrix,n)
      character elem,matrix(n,n)
      integer i,j,m,xi,err_code
      character*20 in_file,out_file
      complex xc_matrix(n,n)
      common err_code,in_file,out_file

      do i=1,n
         do j=1,n
            xi=int(xc_matrix(i,j))
            matrix(i,j)=char(xi)
         end do
      end do

      open (unit=20,file=out_file,status='new',
     +      access='direct',recl=1,err=117)
      print *,'write...'

      m=1
      do i=1,n
         do j=1,n
            elem=matrix(i,j)
            write (20,rec=m) elem
            m=m+1
         end do
      end do
      go to 17

  117 err_code=117
   17 return
      end
```

```
c-------------------------------------------------------------

      subroutine ERROR
      integer err_code
      character*20 in_file,out_file
      common err_code,in_file,out_file

      if (err_code .eq. 0) then
       return

      else if (err_code .eq. 118) then
         print *,'I/O ERROR:  ',in_file
      else if (err_code .eq. 117) then
         print *,'I/O ERROR:  ',out_file
      else if (err_code .eq. 3) then
         print *,'INVERSE TRANSFORMATION UNSUCCESSFUL!'
      else if (err_code .eq. 1) then
         print *,'IMAGE TRANSFORMATION UNSUCCESSFUL!'
      else if (err_code .eq. 2) then
         print *,'BLUR TRANSFORMATION UNSUCCESSFUL!'
      else
         print *,'END OF PROGRAM!'
      endif
      stop

      end
```

Exercise 2. Compile the programs and analyse the optimizitation reports:

```
Optimization for Procedure RAW_INPUT

Line      Id    Iter.     Reordering        New       Optimizing / Special
Num.      Num.  Var.      Transformation    Loops     Transformation
-------------------------------------------------------------------------
   93        1  I         Scalar
   94        2  J         Scalar

  106        3  I         FULL VECTOR Inter
  107        4  J         Scalar


Line      Id    Iter.   Analysis
Num.      Num.  Var.
-------------------------------------------------------------------------
   93        1  I       Inner loop has induction value with varying base or step
   94        2  J       Loop or inner loop has exit
  106        3  I       Interchanged to innermost


            Optimization for Procedure CONVOLUTIONS

Line      Id    Iter.     Reordering        New       Optimizing / Special
Num.      Num.  Var.      Transformation    Loops     Transformation
-------------------------------------------------------------------------
  131        1  I         FULL VECTOR Inter
  132        2  J         Scalar

Line      Id    Iter.   Analysis
Num.      Num.  Var.
-------------------------------------------------------------------------
  131        1  I       Interchanged to innermost


            Optimization for Procedure RAW_OUTPUT

Line      Id    Iter.     Reordering        New       Optimizing / Special
Num.      Num.  Var.      Transformation    Loops     Transformation
-------------------------------------------------------------------------
  149        1  I         Scalar
  150        2  J         Scalar

  161        3  I         Scalar
  162        4  J         Scalar

Line      Id    Iter.   Analysis
Num.      Num.  Var.
-------------------------------------------------------------------------
  150        2  J       Insufficient vector code
  161        3  I       Inner loop has induction value with varying base or step
  162        4  J       Insufficient vector code
```

Optimization for Procedure RAW_INPUT

Line Num.	Id Num.	Iter. Var.	Reordering Transformation	New Loops	Optimizing / Special Transformation
93	1	I	Scalar		
94	2	J	Scalar		
106	3	I	FULL VECTOR Inter		
107	4	J	PARALLEL		

Line Num.	Id Num.	Iter. Var.	Analysis
93	1	I	Loop or inner loop has exit
94	2	J	Loop or inner loop has exit
106	3	I	Interchanged to innermost

Optimization for Procedure CONVOLUTIONS

Line Num.	Id Num.	Iter. Var.	Reordering Transformation	New Loops	Optimizing / Special Transformation
131	1	I	FULL VECTOR Inter		
132	2	J	PARALLEL		

Line Num.	Id Num.	Iter. Var.	Analysis
131	1	I	Interchanged to innermost

```
Optimization for Procedure RAW_OUTPUT

Line      Id    Iter.     Reordering        New       Optimizing / Special
Num.      Num.  Var.      Transformation    Loops     Transformation
-------------------------------------------------------------------------
  149        1  I         Scalar
  150        2  J         Scalar

  161        3  I         Scalar
  162        4  J         Scalar

Line      Id    Iter.   Analysis
Num.      Num.  Var.
-------------------------------------------------------------------------
  150        2  J       Insufficient vector code
  161        3  I       Inner loop has induction value with varying base or step
  162        4  J       Insufficient vector code
```

Exercise 3. Run the programs under CXpa and analyse the reports:

```
================================================================
                          Loop Performance Analysis
                              For: convolutions
================================================================
Summary:
                           Times      Iteration Count          CPU Time
 Line                      Exec     Min     Max     Avg    (plus inner loops) PS
-------                   -------- ------- ------- ------- ------------------ --
   131                           1     256     256     256           0.105736
   132                         256     256     256     256           0.104832

Details:
             Optimization  Times      Iteration Count          CPU Time
 Line    NL    Report     Exec     Min     Max     Avg    (less inner loops) PS
------- --- ----------- -------- ------- ------- ------- ------------------ --
   131    0 S                 1     256     256     256           0.000904
   132    1 S               256     256     256     256           0.104832
```

Compiling command: fc -O1 -lveclib -pa

```
================================================================
                          Loop Performance Analysis
                             For: convolutions
================================================================
Summary:
                           Times      Iteration Count         CPU Time
 Line                      Exec     Min     Max     Avg   (plus inner loops) PS
-------                  -------- ------- ------- ------- ------------------ --
   131                          1     256     256     256           0.006755
   132                          2     256     256     256           0.006748

Details:
             Optimization  Times      Iteration Count         CPU Time
 Line    NL    Report     Exec     Min     Max     Avg   (less inner loops) PS
------- --- ----------- -------- ------- ------- ------- ------------------ --
   131    0 SM,Hs               1     256     256     256           0.000007
   132    1 S                   2     256     256     256           0.006748

                Static Profile               Estimated  Mflops
             Vector  Vector  Chime  (less inner loops)  (plus inner loops)
 Line    NL Spills  Flops   Count     Avg       Peak      Avg       Peak   PS
------- --- ------ ------- ------- --------- --------- --------- --------- --
   131    0      0       0       0     0.000     0.000     0.455    60.000
   132    1      0       6       6     0.455    60.000     0.455    60.000
```

Compiling command: fc -O2 -lveclib -pa

```
================================================================
                          Loop Performance Analysis
                             For: convolutions
================================================================
Summary:
                          Times      Iteration Count         CPU Time
 Line                     Exec     Min     Max     Avg   (plus inner loops) PS
-------                  -------- ------- ------- ------- ------------------ --
   131                        256     256     256     256           0.006904
   132                          1     256     256     256           0.007929

Details:
            Optimization  Times      Iteration Count         CPU Time
 Line    NL    Report     Exec     Min     Max     Avg   (less inner loops) PS
------- --- ----------- -------- ------- ------- ------- ------------------ --
   132    0 P                 1     256     256     256           0.001025
   131    1 SM,I            256     256     256     256           0.006904

               Static Profile               Estimated  Mflops
            Vector  Vector  Chime  (less inner loops)  (plus inner loops)
 Line    NL Spills  Flops   Count     Avg       Peak      Avg       Peak   PS
------- --- ------ ------- ------- --------- --------- --------- --------- --
   132    0      0       0       0     0.000     0.000    49.592    60.000
   131    1      0       6       6    56.955    60.000    56.955    60.000

================================================================
                     Parallel Region Performance Analysis
                             For: convolutions
================================================================
Summary:
           Times            All Regions          Process Virtual
 Line      Exec       CPU Time   Wall Clock Time    Time (PVT)     CPU/PVT  PS
-------  --------   ------------ --------------- ---------------  --------- --
   132          1       0.007929        0.016354        0.010977      0.722

Details:
                           By Region
 Line                 CPU Time   Wall Clock Time    Chore Count             PS
-------              ------------ ---------------   -------------            --
   132                   0.005620        0.016246             185
                         0.002309        0.008744              71

================================================================
                       Basic Block Performance Analysis
                    No data was gathered for basic block(s)
================================================================
```

Compiling command: fc -O3 -lveclib -pa

Bibliography

1. CONVEX Architecture Reference Manual (C Series)
/Sixth Edition, April, 1992 – Convex Computer Corporation/
/Convex Press, Richardson, Texas United States of America – DHW-300/

1. CONVEX FORTRAN Optimization Guide
/Fourth Edition, November, 1992 – Convex Computer Corporation/
/Convex Press, Richardson, Texas United States of America – DSW-034/

CONVEX CXpa User's Guide 3rd edition March 1993
CONVEX Press
Richardson, Texas USA

CONVEX CXpa Reference 1st edition March 1993

I. Pitas: DIGITAL IMAGE PROCESSING ALGORITHMS
Prentice Hall 1993.

Larry Nyhoff- Sanford Leestma: FORTRAN 77
and Numerical Methods for Engineers and Scientists
Prentice Hall 1995.

A. K. Jain: Fundamentals of Digital Image Processing
Prentice Hall 1989.

Etelka Kovács - Ágnes Burián:
Architecture and Operation of Vector Computer CONVEX 3820
TEMPUS Project S_JEP 8333-94

Dr. Rezső Nagy - Magda Kovács:
Optimization and Profiling on Vector Computer CONVEX 3820
TEMPUS Project S_JEP 8333-94

Dr. Rezső Nagy - Etelka Kovács:
Digital Image Processing Exercises in the Spatial Domain on Vector Computer CONVEX 3820
TEMPUS Project S_JEP 8333-94