EPCC-SSP-1997

Particle Image Velocimetry Analysis

Riel Carol

Abstract

The Particle Image Velocimetry, PIV, analysis is a technique which allow us to determinate the 2 components of the velocity of any point of a fluid in movement. This technique is based on the auto-correlation of a digital image of this fluid.
The problem of this technique is that it requires some numerical calculation and that calculation could need a considerable amount of time. That is why it has been decided to parallelise this code to avoid the time problem.
The parallelisation is performed by distributing the data stored in a binary picture over the processors with a regular domain decomposition. The data is collected after by one processor and the results are printed in a file.

Contents

1 Introduction

The aim of this project is to write a parallel code to perform the particle image velocimetry analysis or PIV, that means that the input of the program is a file with the data we want to analyse and some data on the command line and the output of the program is another file with the results of the analysis in a given format. The input file is a graphical file, encoded in a Microsoft .BMP format, of a digital photography of a fluid in movement (figure 1). The other input data are an integer number and a real number used in the code which values are determined by user every time that he uses the code. The output ASCII file is the value of a 2-dimensional velocity vector field on a grid. With an adapted viewer it looks similar to figure 2. The program has been entrusted by the Department of Mechanical Engineering for its personal use. That is why there are many constraints on the formats of the input and output files.

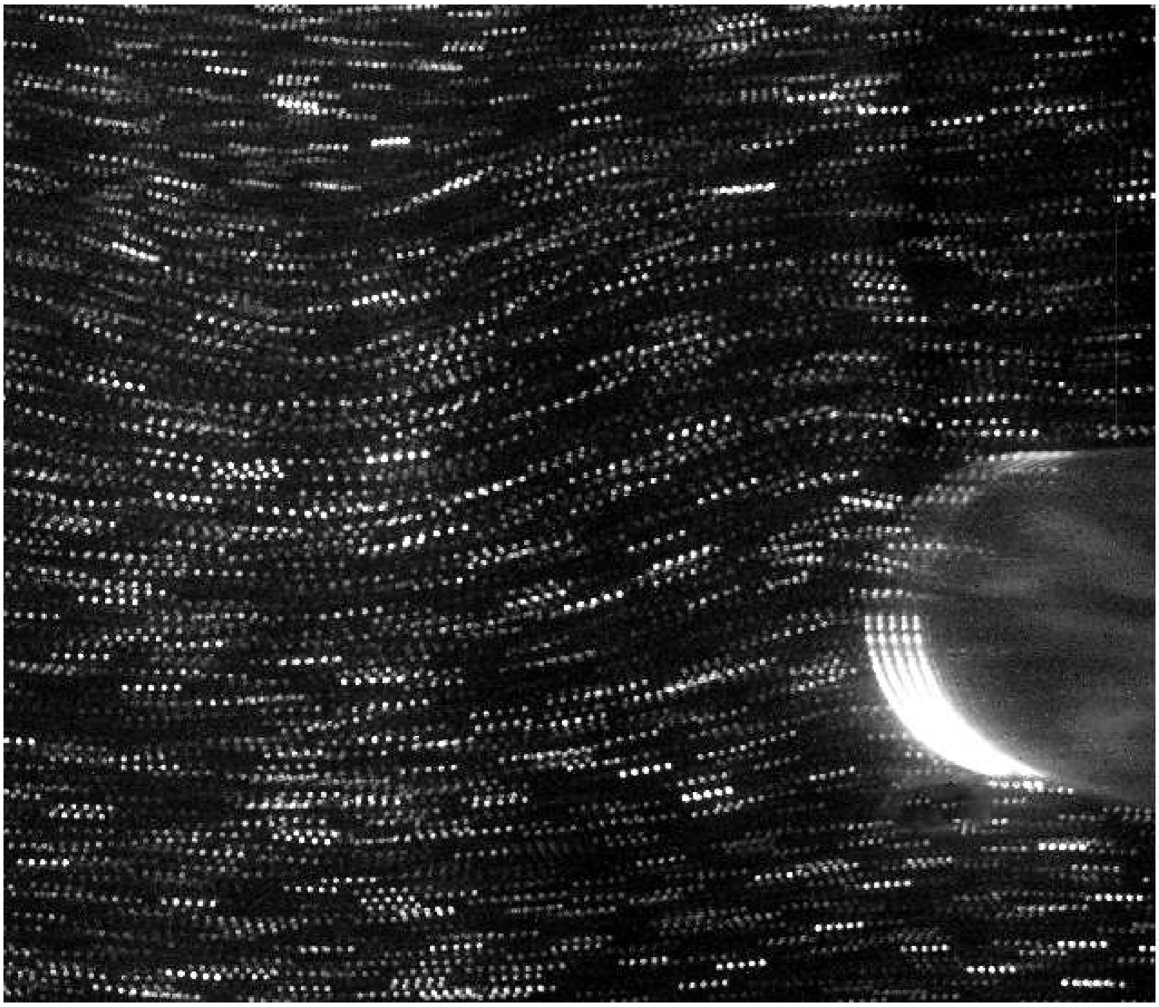


Figure 1: Fluid in movement

There exists a number of serial codes to perform this analysis but they are too slow if we want to find the velocity vector of many points. Parallelising the code will reduce this time.
The project is split into two different parts. The first part is to write a serial PIV code. The

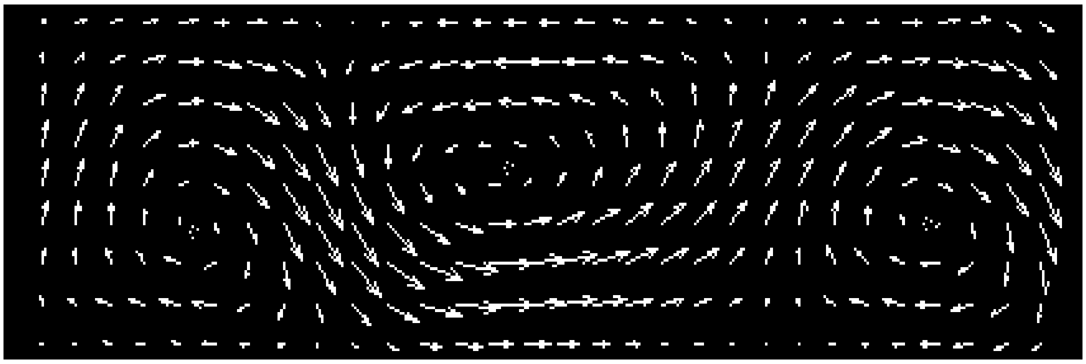


Figure 2: Vector map [12]

second part is to write the parallel code and to create an interface with the user.

2 Background: Particle Image Velocimetry (PIV)

2.1 PIV and the fluid mechanics instrumentation [11]

Fluid dynamics remains the unsolved problem of classical mechanics, yet it is of relevance to everyday life: the weather; indoor climates; engine design, flow around buildings, road-vehicles, aeroplanes and ship transport, etc.

The first fluid velocity measurement system: hot-wire anemometry - developed in the late 1950s and early 1960s - is a point-measurement technique. Another example is the laser-Doppler anemometry optical measurement technique developed in the mid-1970s. Both are standard tools in experimental fluid dynamics. Much of todays understanding of fluid dynamics is therefore derived from point-based information - notwithstanding the fact that large-scale turbulence structures had been recognised by fluid dynamicists as significant phenomena.

In the late 1980s, PIV emerged as a powerful fluid dynamics measurement tool for obtaining instantaneous whole-field velocity data. The PIV technique was of sufficient maturity by the mid-1990s for professional PIV systems to be developed. Thus, fluid dynamicists now, for the first time, have a quantitative whole-field turbulence measurement tool to augment the knowledge they have obtained from point-measurement systems. This new quantitative information has the potential to contribute to a qualitatively new perspective on flow phenomena.

2.2 General principles of PIV

If you know how fast you are travelling and the time you use for the journey, you can calculate the distance travelled. Any one of the three parameters can, of course, be calculated if you know the other two, simply by rearranging the equation. Hence, if you can measure the distance an element of a fluid travels in a known time interval, then you can calculate the speed of the flow field using the simple equation

$$\text{speed} = \frac{\text{distance}}{\text{time}}. \tag{1}$$

In auto-correlation PIV, we know the direction as well as the magnitude of displacements, so we use two vector quantities. That is,

$$\mathbf{V} = \frac{\mathbf{D}}{t}, \tag{2}$$

where **V** is the velocity vector (what we want to find), **D** is the displacement vector (what we measure), and t is the time interval between pulses (what we know).

The basis of the PIV technique is as follows:

1. Seed particles are suspended in the fluid. We need to place small tracer particles, called seeding, in the air or water flow under investigation to observe their displacements. The seeding particles reflect laser light, from which we can determine the initial and final positions of the fluid flow.

2. When a 2D slide of the flow field is illuminated by a light sheet, the illuminated seeding reflect the light. This is detected by a camera placed at right angles to the light sheet. The light sheet is pulsed (switched on and off very quickly) twice at a known time interval t.

3. The first pulse of the laser freezes images of the initial positions of seeding particles onto the first frame of the camera. The camera frame is advanced and the second frame of the camera is exposed to the light scattered by the particles from the second pulse of laser light. There are thus two camera images, the first showing the initial positions of the seeding particles and the second their final positions due to the movement of the flow field.

4. The two camera frames are then processed to find the velocity vector map of the flow field. This involves dividing the camera frames into small areas called interrogation regions. In each interrogation region, the displacement of groups of particles between frame 1 and frame 2, **D**, is measured. The velocity vector, **V**, of this area in the flow field is then calculated using the equation

$$\mathbf{V} = S \cdot \frac{\mathbf{D}}{t} \tag{3}$$

where S is the object:image scale factor between the cameras chip and the measurement area. This is repeated for each interrogation region to build up the complete 2D velocity vector map. The whole PIV device is depicted in figure 3.

The method used to find the displacement **D**, is called correlation processing. Correlation is a procedure to detect a pattern or relationship within data. In PIV, this pattern is the common displacement between many pairs of particles in an interrogation area. The correlation plane is found by processing the randomly distributed seeding particles, identified as bright grey-scale values on a dark background, using a mathematical algorithm based on Fourier transforms. This mathematical algorithm distinguishes the common displacement between pairs of images as a peak. The use of correlation techniques to find the displacement vector allows PIV to be fully automated.

2.3 The auto-correlation function

The aim of the auto-correlation function is to determine the two components of the 2D velocity vector at any given point. What is the auto-correlation function and how it could be useful in

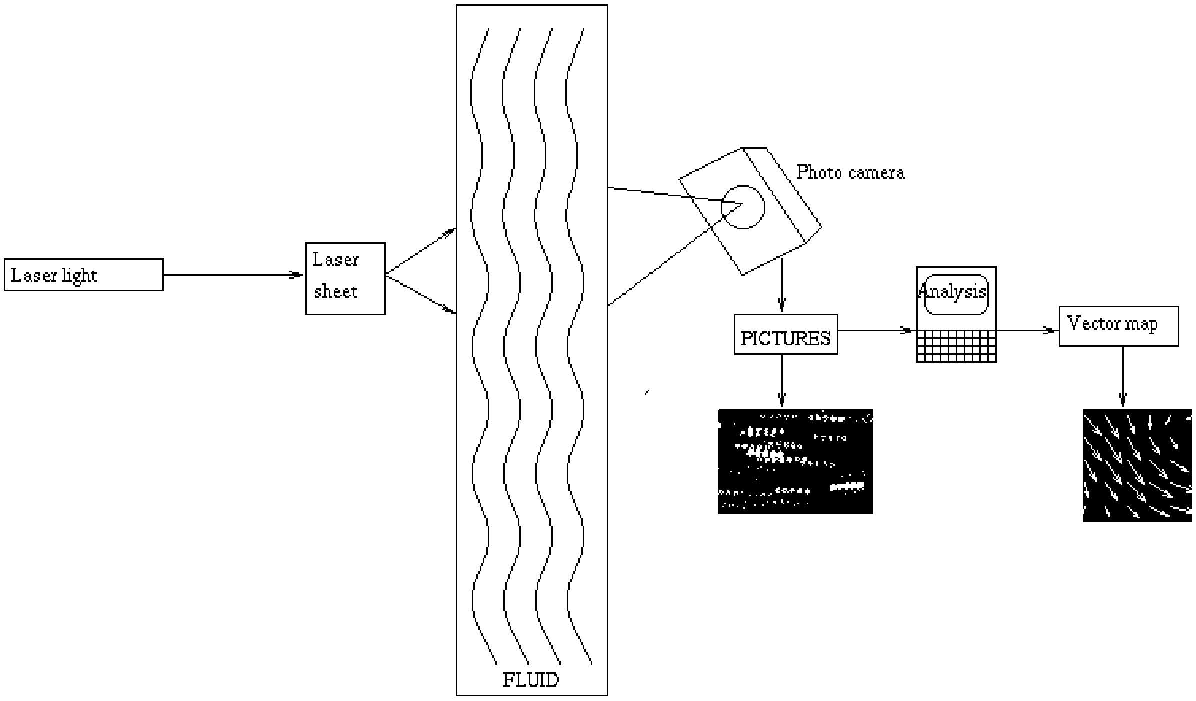


Figure 3: The PIV device

this case?

2.3.1 Mathematical definition of the auto-correlation function

The definition of the auto-correlation function is given by:

$$c_{ff}(x) = \int_{-\infty}^{+\infty} f(t+x) f^*(t) dt \tag{4}$$

However, as far as numerical calculus is concerned, this definition is not useful. This is due to the fact that the codes to calculate integrals are not very efficient, and an infinite integral takes a considerable time to calculate! That's why the definition as been changed to another formula to be more computer friendly. The new formula is:

$$c_{ff}(x) = \mathcal{F}^{-1}[|F|^2](x) \tag{5}$$

where F is the Fourier Transform of the function f and $\mathcal{F}^{-1}$ is the inverse Fourier transform operator.

This is the theorem of Wiener Khintchine [1].
The changing of the one formula to another is quite simple, which is why we are going to detail it as follow.
We know that:

$$f(x) = (\mathcal{F}^{-1}[F])(u) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} F(u)\exp(iux)du \tag{6}$$

where f is a function of the variable x, F is the Fourier transform of the function f and $\mathcal{F}^{-1}$ is the inverse Fourier transform operator. That is the definition of the inverse Fourier transform. If we replace f^* by its Fourier transform in the equation (4) we obtain:

$$c_{ff}(x) = \int_{-\infty}^{+\infty} f(t+x)\frac{1}{2\pi}\int_{-\infty}^{+\infty} F^*(u)\exp(iut)dudt \tag{7}$$

IF we rearrange equation (7) we obtain

$$c_{ff}(x) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} F^*(u)\int_{-\infty}^{+\infty} f(t+x)\exp(iut)dtdu. \tag{8}$$

Alternatively

$$c_{ff}(x) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} F^*(u)\int_{-\infty}^{+\infty} f(t+x)\exp(iu(t+x))d(t+x)\exp(-iux)du \tag{9}$$

Now we have only to remark that

$$\int_{-\infty}^{+\infty} f(t+x)\exp(iu(t+x))d(t+x) = F(u), \tag{10}$$

then equation (9) becomes

$$c_{ff}(x) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} F^*(u)F(u)\exp(-iux)du, \tag{11}$$

which is equivalent to:

$$c_{ff}(x) = \mathcal{F}^{-1}[F\cdot F^*](x). \tag{12}$$

Finally

$$c_{ff}(x) = \mathcal{F}^{-1}[|F|^2](x). \tag{13}$$

This formula is useful since there are some efficient codes to calculate the Fourier transform, for example the Fast Fourier Transform algorithm of Cooley and Tuckey [1]. The auto-correlation code executes the FFT algorithm, calculates the absolute value of the result and then executes the inverse FFT algorithm.

2.3.2 A More Empirical Point of View of the Auto-Correlation

The aim of this section is to present the auto-correlation in a more understandable manner by way of an example.

Given a function f defined as follow (figure 4):

$$f(x) = \begin{cases} A, 1 < x < 2 \\ 0, elsewhere \end{cases}$$

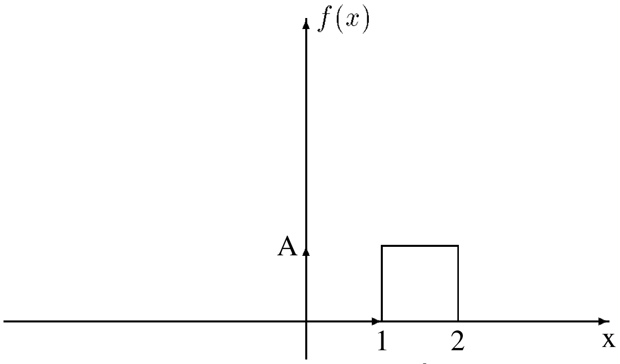


Figure 4: Function f

When we calculate the auto-correlation of the function f, we calculate the surface below the $f^*(t) \cdot f(t+x)$ curve. We can easily remark that for some x values, this multiplication is going to be equal to zero. In the case of this example we can see that if $|x| >= 1$ then the result is 0. Now we are going to take an example to study the cases where the result is not equal to 0. If $x = -\frac{3}{4}$ then $f^*(t) \cdot f(t+x)$ is going to be as follow in the figure 5.

The area of the hatching part is therefore $S = \frac{1}{4} \cdot A^2 = \frac{A^2}{4}$. The value of the auto-correlation functions is therefore $\frac{A^2}{4}$, when $x = \frac{-1}{4}$.
Using the similar arguments we find that

$$c_{ff}(-1) = 0$$

$$c_{ff}\left(\frac{-3}{4}\right) = \frac{A^2}{4}$$

$$c_{ff}\left(\frac{-1}{2}\right) = \frac{A^2}{2}$$

$$c_{ff}(1) = A^2$$

$$c_{ff}\left(\frac{1}{2}\right) = \frac{A^2}{2}$$

$$c_{ff}\left(\frac{3}{4}\right) = \frac{A^2}{4}$$

$$c_{ff}(-1) = 0$$

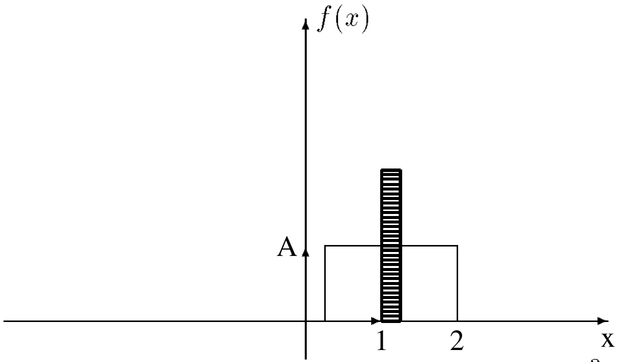


Figure 5: $f(t)$ and $f(t+x)$ functions with $x = \frac{-3}{4}$

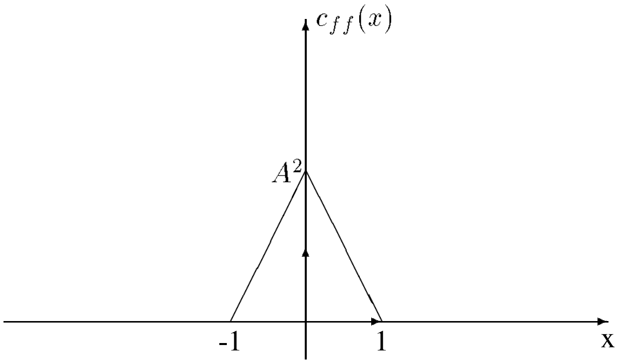


Figure 6: Function c_{ff}

By induction we obtain the following function in figure 6.

The most important thing to notice is that we have a peak for $x = 0$.
If we take another function g defined as follow (figure 7):

$$g(x) = \begin{cases} A, 0 < x < 1 \\ B, 3 < x < 4 \\ 0, elsewhere \end{cases}$$

The auto-correlation of this function is displayed in figure 7.

Now it becomes clear how the auto-correlation of the fluid flow digital picture can help us to find the velocity of each point. The two parts of the function g which are not equal to zero, can be the position of a particle at the moment of the two photos in one dimension. In 2 dimensions, we can find the distance between the principal peak and one secondary peak, and therefore deduce

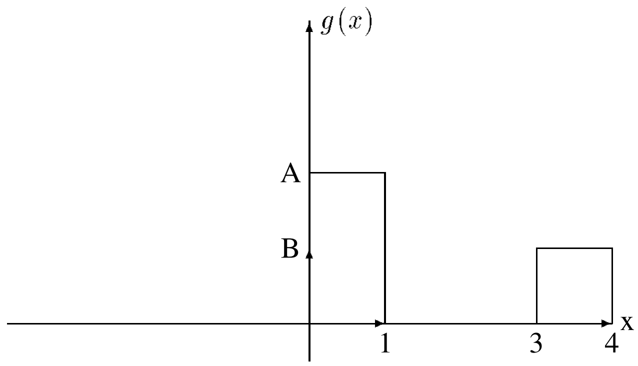


Figure 7: Function g

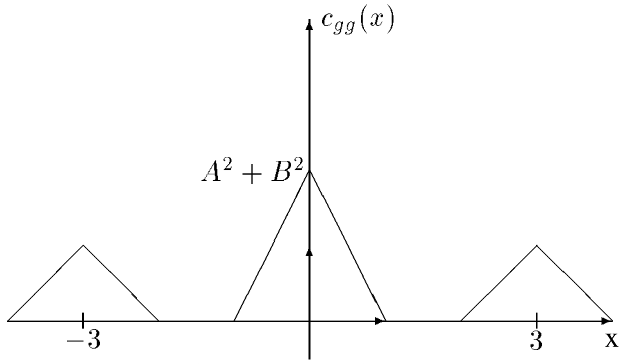


Figure 8: Function c_{gg}

the distance and the direction between the two images of the particle.

3 The PIV Program

3.1 The Serial Program

The serial PIV code reads the files containing the data, calculates the velocity of one single point and print the result on the screen. For this calculation the code needs two more input data which are: the size of the matrix used to calculate the velocity of the point and the lapsed time between the two images. It also print the "quality" of this calculus. The quality factor is simply the height of the peak divided by the mean value of the elements of the auto-correlation matrix. The general structure of the program is represented in the diagram of figure 9.

3.1.1 The Algorithms

The **read subroutine** reads the data from the input file. The file is a Microsoft Windows Bitmap (.BMP) file, 8-bit uncompressed. The header of the file contains the colour map. The program

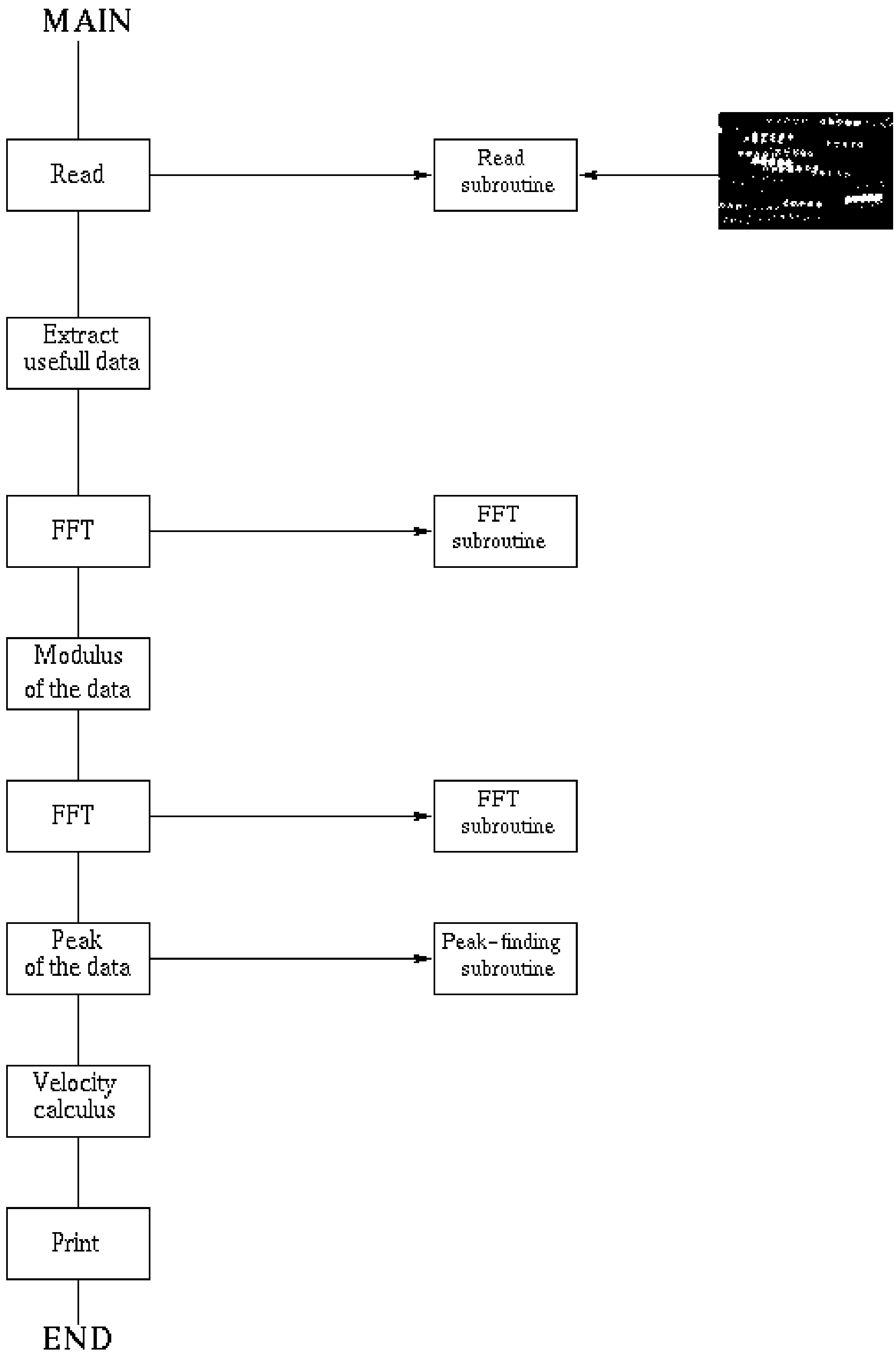


Figure 9: Structure of the serial code

only works with this type of file, which is the format of the files that are used by the Department of Mechanical engineering. First the height and the width of the image are read and then the body of the digital picture.

The next step is to **extract the useful data**. That means to put the data around the point where we want to find the velocity in a matrix of a chosen size. The size of this matrix has to be a power of two. This constraint will be explained on the following step.

Now the proper PIV analysis can start. The first thing, as we have seen in the background chapter, is to perform a Fourier transform of the data. The most obvious way to do this is to use an implementation of a 2 dimensional fast Fourier transform (**FFT**). The implementation used in the program is not the optimal one but it is sufficient for our purpose. The algorithm is based on a one dimensional FFT code. First the code performs the FFT of each row, transposes the obtained matrix, then performs the one dimensional FFT on another set of rows and transposes the matrix back.

After the first FFT there is the **calculation of the modulus** of each element of the matrix. This step is really simple and performed in the main program itself.

The following step is to perform the inverse Fourier transform. Actually, for a real function (with a null complex part) it is the same thing to perform the direct Fourier transform as the inverse Fourier transform. As the preceding step was the calculation of the modulus, the matrix is real, we can perform the direct transform once again using the **FFT** subroutine.

The matrix obtained is not yet ready to be used to **seek the peaks**. As we have seen in the background, the central peak is located in the "centre", i.e. for $x = 0$ and $y = 0$; in the matrix that is the first element. This location of the peak is not optimal because the algorithm used to seek the peaks needs to know the neighbours of the peaks. That is why, knowing the fact that the Fourier transform is periodic we can translate the function and locate the central peak in the middle of this new matrix. The matrix is now ready to be used to seek the peak. The algorithm to find the peak is not optimised. We need to find two peaks; the central peak, and one of the two secondary peaks, as we have seen in the background chapter.
For the central peak, the algorithm used is the intuitive one: the seek of the maximum value of the matrix.
For the second peak the preceding algorithm is not useful because we are not looking for the second highest point of the matrix, we are looking for the second highest point which is higher than all its neighbours. The algorithm used in this case follows the definition of what we are looking for: we scan all the points of the matrix to find the highest point which is higher than all its neighbours but which is not the maximum.
The calculation of the quality is performed in this subroutine.

The **calculation of the velocity** is now trivial: we need only to divide the distance between the two peaks, by the time.

The last step is to **print** on the screen **the result**.

3.1.2 The Implementation

The code has been implemented in C. There are some important remarks about the structure used to store the data, especially for the matrices. In the serial program a matrix is a "vector of vectors", which means that the data is not contiguous in memory. The data is only contiguous inside a row.
Another important thing to know is that there is a voluntary choice to write a modular code, which allows the same subroutines to be used for the parallel code. This explains why sometimes there are some subroutines which are separated from the main code even if is not the best choice for the sequential program.

3.2 The Parallel Program

The parallel code performs the same task to the serial code on a higher number of points. Hence the output of the program is slightly different from the serial program. Here the output is not printed on the screen. The output is an ASCII file containing 5 columns: the first one is the x position of the point, the second is the y position, the third is the x component of the velocity, the fourth is the y component and the last is the quality of the calculation.
The input is the same as for the serial code.
The question now is how does the code select points to calculate their velocity. The choice is done in a regular way. If we choose to calculate the velocity of the points using the 32 x 32 matrix around them, the code calculates the velocity of the points with a step of 32 in the x and y direction.

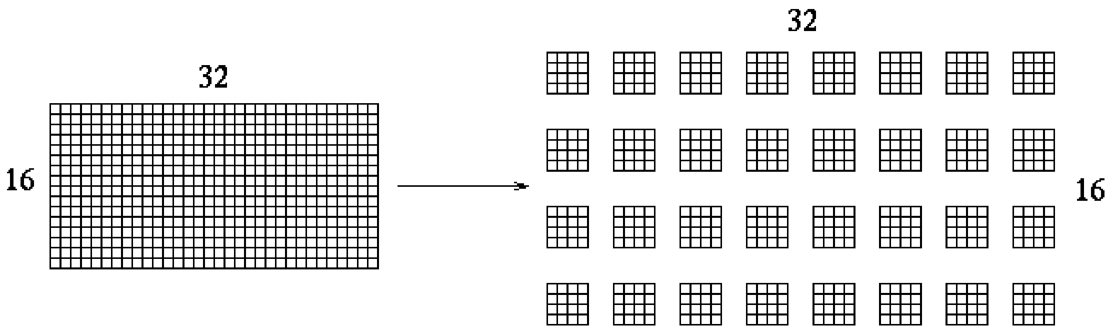


Figure 10: Decomposition of the data for the calculus of the velocity

The general structure of the parallel code is represented in the diagram of the figure 11.

3.2.1 The parallel algorithms

As we can see in the figure 11, there are many subroutines which seem to be the same than the ones of the serial program, that is the case of the FFT subroutine and the "peak-finding" subroutine. For the read subroutine there are some few lines extra. All the other subroutines are new. These new or modified subroutines are as follows:

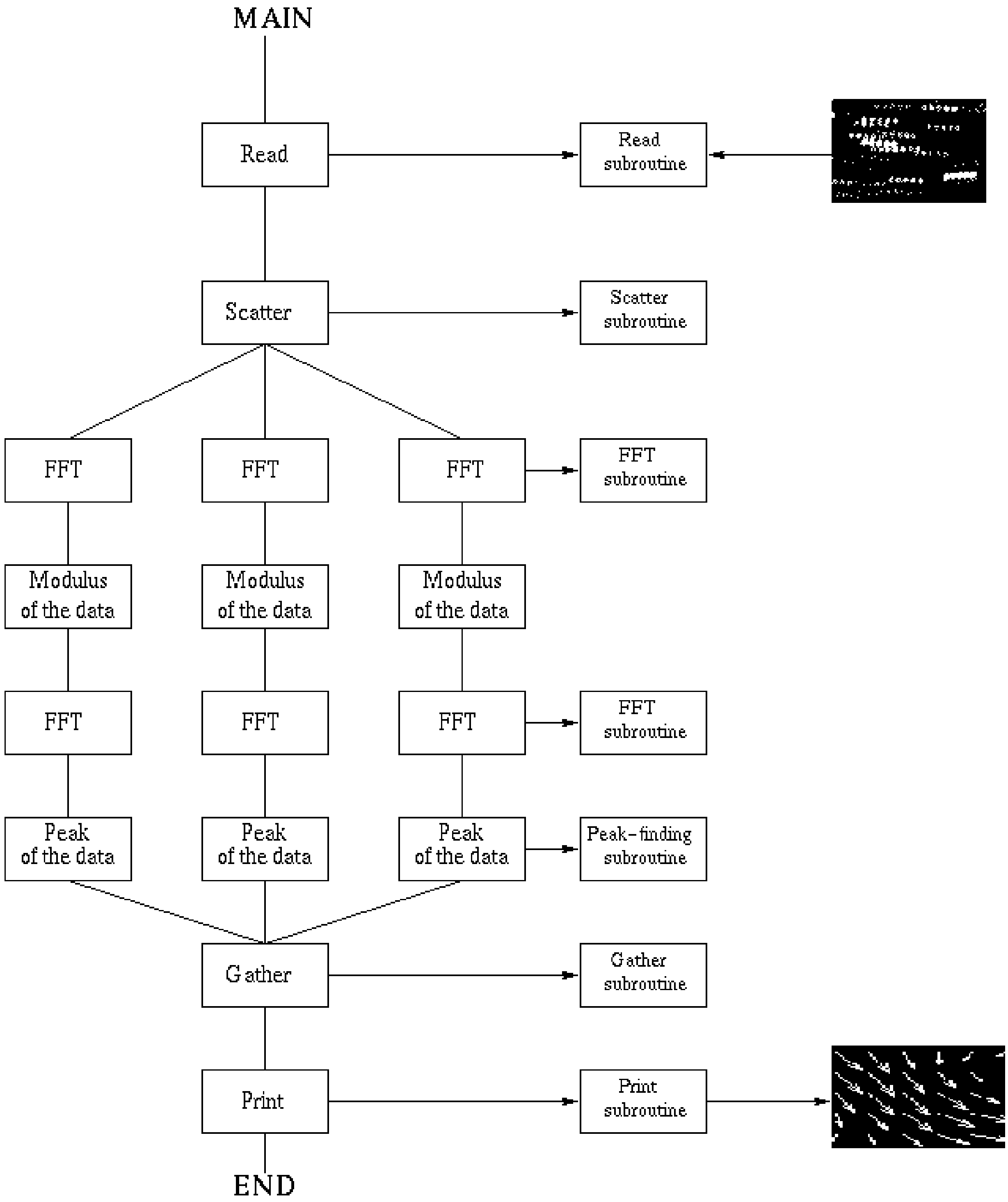


Figure 11: Structure of the parallel code

The changes in the **read** subroutine are made to balance the amount of work of the processors. The number of pixels of the picture is the size of the total amount of work, if the code is run on n processors, to have a well-balanced program, the size of the problem has to be a multiple of n. That is why the size of the matrix is increased with the purpose to transform the size of the problem in a multiple of the number of processors. This increase is made by adding zeros to the borders of the matrix.
The same thing is done to transform the size of the matrix to a multiple of the size of the sub-matrix used to calculate the velocity of the points. As before, to perform this we add zeros to the matrix as we can see in the figure 12.
An important remark to say about the read subroutine is that only one single processor reads the data.

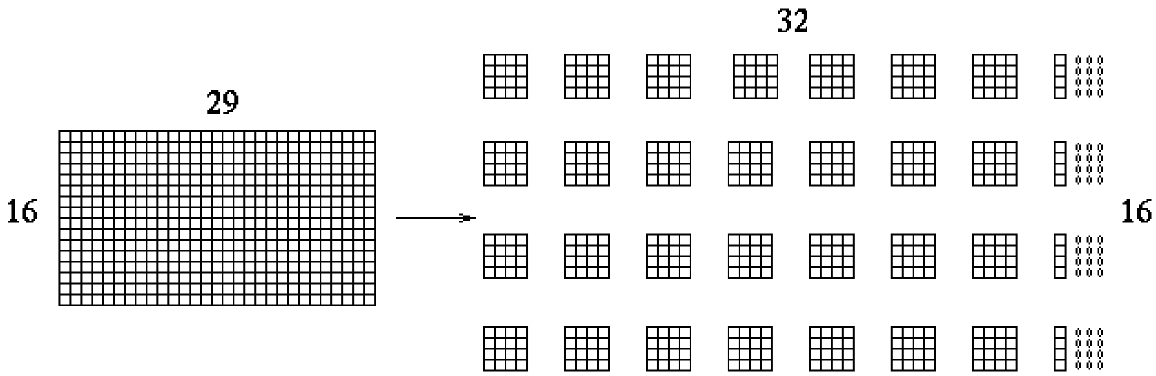


Figure 12: Resizing the matrix

Once the data has been read, it has to be scattered on the processors. That is the aim of the subroutine **Scatter**.
The parallelisation of the program starts here; we have chosen to use a regular domain decomposition of the data instead of a task farm. For the further developments of this project, the regular domain decomposition was the chosen to avoid having to many communications. Also, we will need to know what informations each neighbouring processor has and this is not easy with a task farm. We can summarise the regular domain decomposition with the figure 13.

The heart of the code, all the proper PIV analysis, is not changed. That means that the subroutines **FFT** and "**peak-finding**" are the same as in the serial program.

Once all the processors have finished performing the PIV analysis, one processor **gathers the data** from the others processors. The data is stored in a single 5 dimensional matrix. The first two dimensions give the location of the data, the third and the fourth give the velocity of the point and on the fifth one there is the quality of the calculation.

The last step is the output of the data; the work of the **print** subroutine. For the parallel code, the output is an ASCII file containing the results. We can see an example of this kind of file in the figure 14.

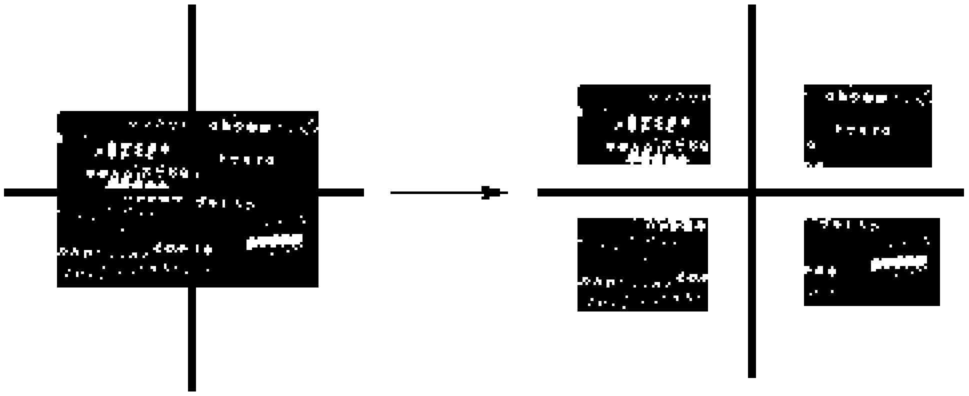


Figure 13: The regular domain decomposition on four processors

```
 16    16   -2.97601    0.973719    1.82272
 48    16   -2.99003    0.975913    1.53739
 80    16   -2.98641    0.97077    1.69051
112    16   -2.98696    0.973198    1.64428
144    16   -2.98863    0.953781    1.84346
176    16   -2.98771    0.963141    1.62946
208    16   -2.98022    0.951429    1.84488
240    16   -2.98384    0.979744    1.62549
272    16   -2.9892    0.957088    1.85567
304    16   -2.98799    0.99239    1.4168
336    16   2.98434    0.982542    1.66232
368    16   -2.97924    0.981308    1.81866
400    16   -2.99755    0.986648    1.51848
432    16   -2.99398    0.986394    1.55742
464    16   -2.99625    0.979345    1.72478
496    16   -2.97973    0.965578    2.22511
```

Figure 14: Example of output file

3.2.2 The Implementation

The implementation of the code is made in C using the Message Passing Interface (MPI) on the Cray T3D.
The subroutines used to scatter and gather the data (MPI_ Scatterv and MPI_ Gatherv) are optimised, even though their implementation is not evident, they were the best choice. But these routines work on a contiguous block of memory, and the serial program was written for non-contiguous memory. That is why the matrixes are stored in the memory with a new structure. The structure is a contiguous block of memory for the matrixes. The matrixes in the serial code have two indexes ala $matrix[i][j]$. In the parallel code, if the matrix has the size height $/times$ width, the same element as the serial matrix will be $matrix[i * width + j]$.
Another problem encountered was the way that floating variables are stored on the Cray-T3D. On the Cray the floating variables are stored by packets of even numbers. If you define a data-type of five floating numbers, and you try to send a buffer with several objects like these, the end

of the buffer will be corrupted, it will be the same thing with all the odd numbers excepted with one. To solve this problem there is only one easy solution: is to add one more floating variable to the data-type.

4 Results

The serial and parallel codes are completed. With the same input files and parameters the results of the two codes are the same.

4.1 Runtime results

The time scaling from the serial to the parallel code is quite good but the performances of the parallel code can be increased easily. If we look the figure 16 we can see the speed up of the parallel code. The best number of processors to run the code seems to be 8 or 16 according the figure 15. But even with this number of processors the speedup of the code is not so good. The problem is that there is a serial bottleneck: the read of the data. To avoid this problem or to reduce the time spent in communications with respect the time spent computing the analysis, one must spend more time computing. That means that the number of velocity evaluation points per processor has to be increased. The speed-up is going a be increased. I think that we can increase the performances of the program with a little improvement of the speedup.
In the following figure (figure 16) we can see the speedup of the parallel code with respect the total number of processors.

Number of processors	Time (min)	Speedup
1	1:59	1
2	1:10	1.7
4	1:09	1.72
8	0:33	3.6
16	0:17	7
32	0:15	7.9

Figure 15: Runtime performances of the parallel program

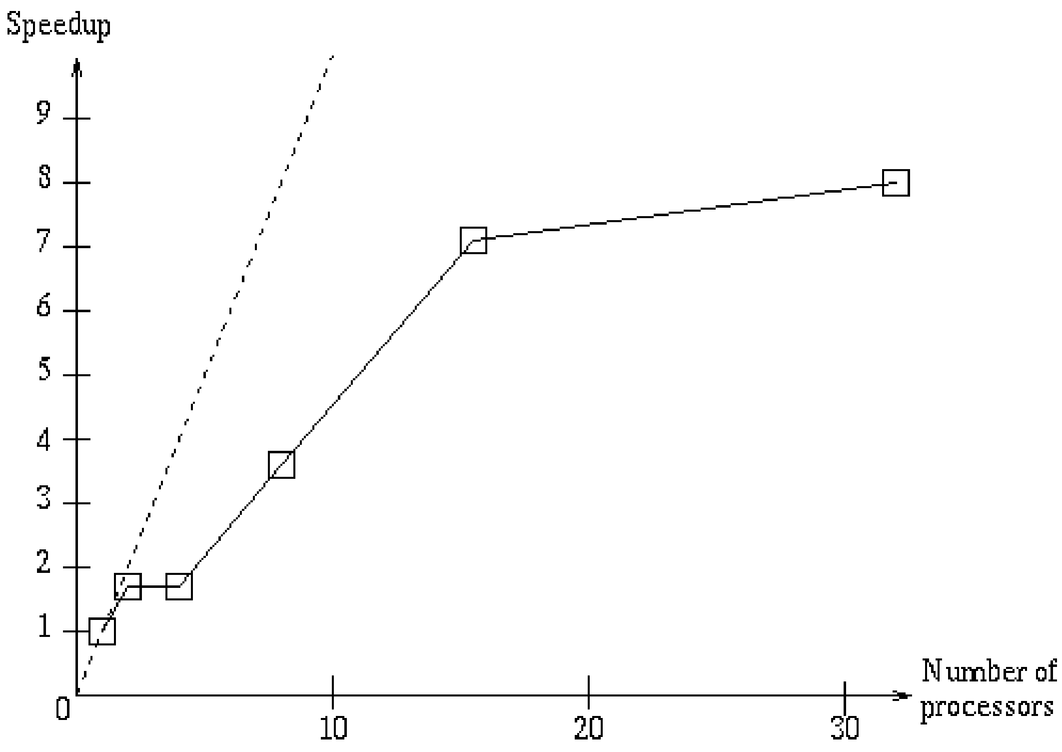


Figure 16: Speedup of the parallel code with respect the total number of processors

4.2 Validity of the results

The FFT code used in the serial and parallel code was downloaded from the Internet [8]. Another FFT code, from the Internet, was downloaded to check if the two codes produced the same results.

5 Summary and conclusion

In this report we have presented the PIV analysis technique, the serial and parallel code implemented to perform this analysis. I have first made a serial implementation to see how it works and to understand that serial codes are too slow and we have made a parallel implementation to win speed in the calculation. Also I have had to learn how were encoded the files with the Microsoft Windows Bitmap format [5][6][7] and how to use the MPI library [9][10].
The conclusion of this work is that the speedup of the parallel code is quiet good but there is a serial bottleneck. This problem is shared with most parallel programs. The next step of this project would be to increase the amount of work done by the processors by increasing the number of points where we calculate the velocity.
Another development could be to add a subroutine able to remove the bad vectors. This invokes accessing the data in neighbouring processors. A more efficient FFT can be used. In this case the amount of work done by the processors will decrease when the amount of communications will be the same. Thus the speedup will be reduced. That is why we do not need FFT optimised

algorithms.

6 User manual of the PIV code

The PIV code can be run on 4, 8, 16 or 32 processors. Each version is called PIV4, PIV8, PIV16 and PIV32. The parameters to run all the versions are the same. The arguments of the command line are, in order: the name of the input file with, the size of the matrix for the calculation of the velocity and the elapsed time between the two images. Each argument is separated by a space-bar character. The running statement for the Cray T3D is 'mppexec'.
These are the specifications of each argument:

the input file name:
the whole name of the file , including its path.
the size of the matrix:
it is a number of the type 2^n where n is an integer > 1.
the elapsed time between the two images:
it is a positive float number.
If we want to run the PIV code on 8 processors, on the file picture.bmp, with a size of matrix equals to 32 and with an elapsed time equals to 0.5 we write:

mppexec PIV8 picture.bmp 32 0.5.

The output file generated by the code has the same name than the input file but with another more extension: *.output*.
In the case of the precedent example, the name of the output file is

picture.bmp.output.

References

[1] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P.Flannery. *Numerical Recipes: The Art of Scientific Computing, Second Edition*. Cambridge University Press, 1992.

[2] Eugene Hecht. *Optics, Second Edition*. Addison-Wesley Publishing Company, 1987.

[3] Robert Guenther. *Modern Optics*. John Wiley and Sons, Inc, 1990.

[4] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison-Wesley Publishing Company, 1993.

[5] Bitmaps and Palette Manipulation. http://silo.csci.unt.edu/home/brackeen/vga/bitmaps.html

[6] Format of a .BMP Graphic File.
http://www.echo.com/ umlaut/icb/bmp.html

[7] Graphics File Formats.
http://cs.wpi.edu/ matt/courses/cs563/talks/gformats.html

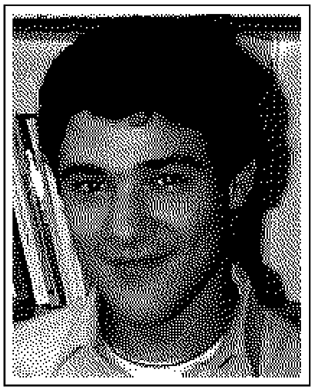
[8] Lab Exercises and Examples.
http://www.mhpcc.edu/training/workshop/html/samples/exercises.html

[9] MPI: A Message-Passing Interface Standard
http://www.mcs.anl.gov/mpi/mpi-report/mpi-report.html

[10] MPI: The Complete Reference
http://www.netlib.org/utk/papers/mpi-book/mpi-book.html

[11] Dantec PIV
http://www.dantecmt.com/scientific/piv/pivmain.htm

[12] Steve Wereley's PIV page
http://waldo.mech.nwu.edu/ swereley/



Riel Carol, undergraduate at the Ecole Nationale Superieure des Mines de Saint Etienne. I study computer science but I have also studied electronics, economics and sociology. I will graduate in June 1998.

Supervisor:
Gavin J Pringle, EPCC.
Proposer:
William J Easson, University of Edinburgh, Department of Mechanical Engineering.