Optimization of FFT Computation for Processors of the Pentium 4 Type

Luis Figueiredo[1], Mário Freire[2]

Summary

The Fourier Transform, or more properly the Discrete Fourier Transform (DFT), has an extremely vast application in the areas of civil, electrical and mechanical engineering, and computer science.

One of the problems of its practical use is the high computational effort that it imposes for its computation. With the algorithm of Fast Fourier Transform (FFT) it was possible to diminish substantially this effort of computation, especially for series with large number of signals.

The computational power of the modern computers allows, inevitably, enhancement of the speed of the computation of this transform, increasing, more and more, the possibility of its application in real time systems using generic low-cost equipment. However, it is still possible to increase its performance significantly if we use all the potential of the Pentium 4 processor, potential not always efficiently used by the traditional compilers of high level programming languages.

Introduction

The exponential increase of the computational capacity, associated with increasing developments of the optimization capacities for high level languages compilers, has made programmers neglect the knowledge related to the basic principles of computer architecture. These facts take into address the situations where the programmers lose the notion of what they are losing in performance for its applications, due to the inefficient utilization of all the capacities of modern computers.

The aim of this paper is to present a comparative study of the performances obtained in the computation of the FFT for one and two dimensional signals, being used the most recent compiler of C++ from Microsoft (Visual Studio.NET 2003) and using the Single Instruction Multiple Data (SIMD) technology, with low level programming [1][2].

In the next section, a brief overview of the DFT and FFT is presented. Next, the techniques and results for the parallelization of the FFT using the Pentium 4 SIMD technology will be presented. These results will be compared with those achieved employing a classic implementation in a high-level language compiled with the related compiler. Finally, main conclusions will be presented.

[1] ESTG, Instituto Politécnico da Guarda, Portugal

[2] Departamento de Informática, Universidade da Beira Interior, Portugal

FFT-Fast Fourier Transform

The DFT allows the transformation of signals in time domain to frequency domain, making possible its analysis and manipulation into this new domain. For one-dimensional signals the DFT is defined for the equation (1).

$$F(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(x) e^{-j\frac{2\Pi ux}{M}} \tag{1}$$

For two-dimensional signals the DFT takes the form of the equation (2).

$$F(u,v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\Pi\left(\frac{u}{M}x + \frac{v}{N}y\right)} \tag{2}$$

Taking into account the exponential function properties, it is possible to perform the DFT computation for two-dimensional signals, applying, in a consecutive form, the equation (1) to all the lines of the matrix *f(x, y),* followed by a new application of the same equation to the columns of the results derived in the first step.

The well-known algorithm of the FFT, presented by J. W. Cooley and J.W. Tuckey in 1965, allows an increase in the speed of DFT computation, especially for large dimensions signals. Basically this algorithm divides the original signal in two, one with the even elements, and the other with the odd elements of the original signal. After the DFT computation of each one, its values are combined to get the DFT of the original signal. This process consists, for the first half of the signal to process, of adding to each even element the corresponding odd element multiplied by a sinusoid. For the second half of the signal each element pair is deducted with the corresponding odd element also multiplied for the same sinusoid. If the number of elements M of the signal will be a power of 2, is possible to perform successive divisions of each signal until its size reaches one. In this phase the DFT computation is absolutely trivial, as M=1, F(0)=f(0). The mathematical foundation for this known algorithm can be found in diverse literature [3].

Practical Results

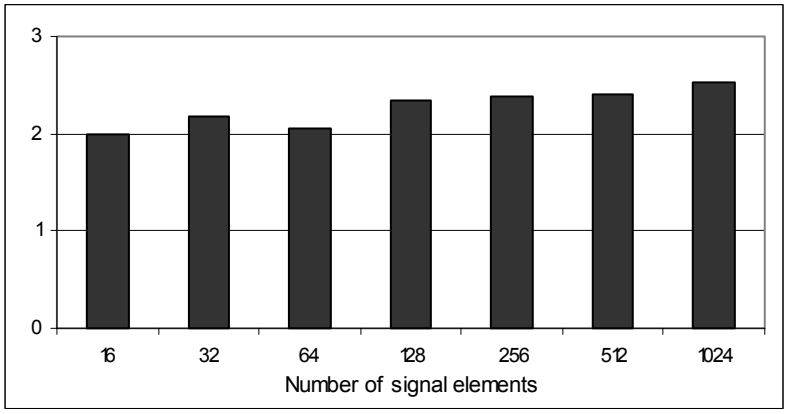
All tests were carried out on a personal computer equipped with a 2.55-GHz Pentium 4 processor, with 512KBytes of second level cache, and 8KBytes of first level cache for data. The size of system RAM memory was 512MBytes. The operating system used was Microsoft Windows XP.

The performance metric considered here was the number of CPU clock cycles. For each test, five measurements had been done. The median of those results is then used for the final value of each test. So that the effect of the cache memory would be, as much as possible, equal for different measurements, it was decided to fill each one of the signals elements before each test.

The practical results obtained have for a basis of comparison the times achieved with a public domain algorithm implemented in C [4] using the compiler Visual Studio.NET 2003. The function that implements this algorithm has two vectors of floats as parameters, respectively with the real and the imaginary part of the signal, and a whole number with the number of elements. All the necessary computations are made internally in the function. This implementation reduces to a minimum the effort of computation of the necessary sinusoids to perform the FFT, since, after each sinusoid is calculated, all subsequent computations, where its use is necessary, are made.

For better optimization of the existing resources in the processors of the type Pentium 4, nominated in respect to the organization of the data to facilitate its vector processing capacities, the choice was made to use a single vector of floats contain the real and imaginary parts of each element. The data obtained for one-dimensional signals, with sizes that vary between 16 and 1024, show that it is possible to increase the speed of FFT computation in relation to the implementation in C for a factor larger then 2, as may be observed in Graph 1.

These results are much more significant since it is recognized that in the case of FFT implementation using SIMD technology a set of repetitive mathematical operations are made in non-continuous data in memory, which it is not, at all, the ideal situation for the utilization of this technology.



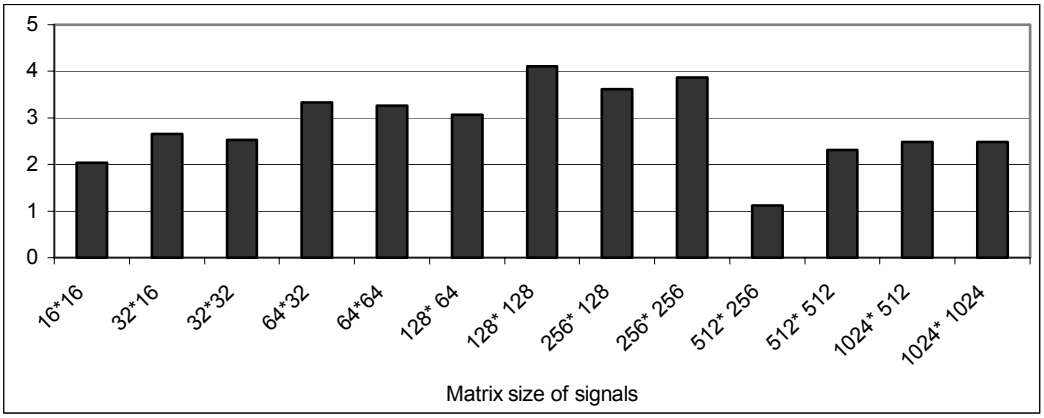
Graph 1: Speed of FFT using SIMD in relation to the FFT in C for one-dimensional signals

Relative to the implementation of the FFT for two-dimensional signals, the simplest form of making it consists of the execution of the following steps:

- FFT computation for each one lines of the signal (FFT H).
- Transpose the obtained matrix (Transp. 1).
- FFT computation for each one lines of the new obtained matrix (FFT V).
- Transpose the obtained matrix (Transp. 2).

In the vast majority of practical situations, when the FFT of two-dimensional signals is calculated, each element is multiplied by $(-1)^{(x+y)}$, where x and y are the coordinates of the element to multiply. The objective of this operation is to center the spectrum obtained, facilitating its posterior processing. Taking this fact into account, the time-lapse for this operation was also introduced in the measurement time of the FFT.

The first results gotten for the computation of the FFT in two-dimensional signals meet express in Graph 2.



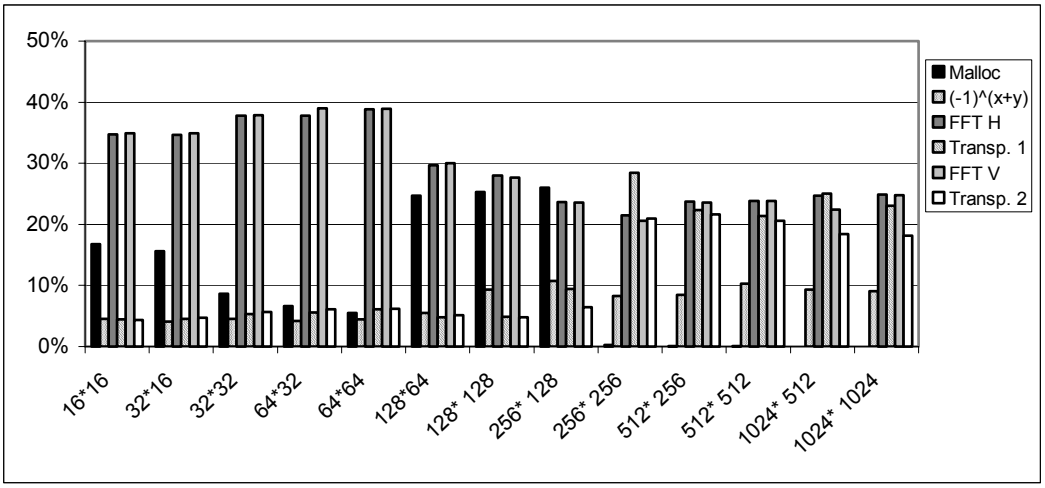
Graph 2: Speed of FFT using SIMD in relation to the FFT in C for two-dimensional signals with size from 16*16 to 1024*1024

With the exception of the matrices of size 512*256 where the execution times are similar, in all the other sizes the execution times are between about 2 and 4 times faster. For this particular size one witnesses a time degradation due to the utilization of one matrix with size equal to 512*256*4*2=1Mbytes, while in the traditional implementation in C, two matrices with size equal to 512*256*4=512Kbytes are used. While each one of these matrices fits in the level two cache, which improves its transpose, the process to transpose the matrix of 1Mbyte takes more time, with direct influence in the global FFT time.

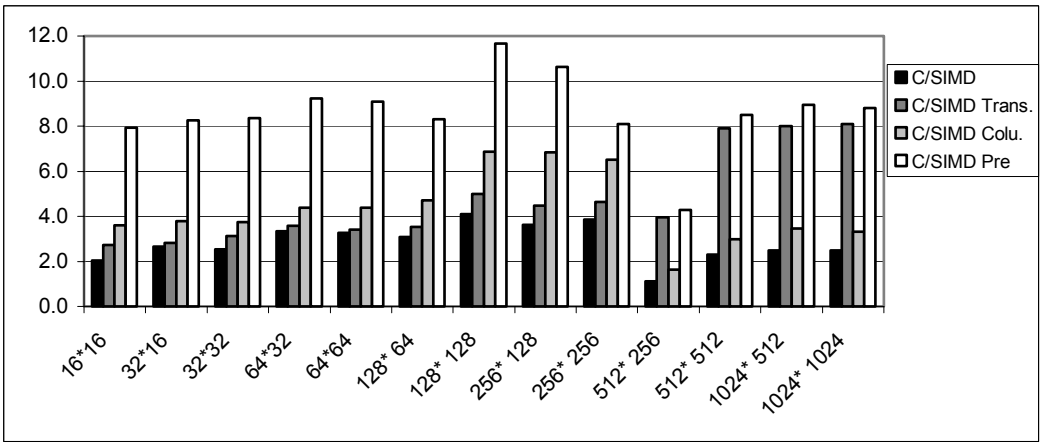
To better understand the ratio of the involved tasks times in the FFT computation of two-dimensional signals, their measurement were taken for the different sizes of matrices. Beyond the related steps, the time to order the operative system for the necessary memory (Malloc) is included. The results are expressed in Graph 3.

With these results as a starting point, a set of alterations were made with the aim of diminishing the FFT computation time. The first step, namely (SIMD Trans.), consisted of changing the transpose algorithm to optimize the effect of cache memory. The results, presented in Graph 4, show a special improvement for the matrices of sizes greater than the level-two cache. The following step, (SIMD Colu.), was undertaken due to the

necessity of re-arranging the elements of the original signal, rearranging this equivalent to the successive divisions of the signal to odd and even elements. Thus, instead of calculating the matrix transpose, and after rearranging the elements, the elements of each one of its columns are directly re-arranged for a one-dimensional vector and the FFT for this vector is performed. Next, the results are introduced into the respective column of the original matrix. The results showed an increase of the performance, when the original matrix fit in the level-two cache, and a reduction for the other situations.



Graph 3: Percentage time of the different phases of the FFT



Graph 4: Final results for the different solutions

The last step, namely (SIMD Pre), consisted of using to best advantage the two previous steps in relation to the size of the matrix and the memory cache, and effecting the pre-computation of all necessary sinusoids to perform the FFT. In this way, the time required to effect unnecessary real time computations is diminished and, above all, it allows the organization of a new more efficient algorithm in terms of cache memory. This is a substantive issue. In fact, the efficient use of memory cache, associated with the correct organization of the data, is fundamental when the aim is to optimize the software using the Pentium 4 vector processing capacities. The results obtained are absolutely clear.

It is important to underscore the fact that these last results are compared in the same fashion as the tree previous ones. This permits a better appreciation of the significant increase derived performance, remembering of course that the C implementation doesn't use the pre-computation of the necessary sinusoids.

Conclusions

With a combination of algorithms, and good data organization, that take best advantage of the capacities of the Pentium 4, it is possible to significantly increase the speed of the FFT computation.

The processing capacities that allow this performance improvement are not used efficiently by the compiler in the tests undertaken, which means that the programmer cannot leave for the compiler all the work of code optimization. Otherwise, the potential of the computer may not be fully realized.

The knowledge of basic concepts of Computers Architecture is fundamental, so that the programmers can structure the data and better organize its algorithms to extract advantages, in terms of speed, whenever the time factor becomes crucial. Even without programming in low-level language, it is perfectly possible, and desirable, to use this knowledge in the optimization of applications.

In practical situations, such as the various fields of engineering that demand processing in real time, it could be advantageous to invest a little more time in code optimization. This study showed that the advantages of this extra care can be significant.

References

1 Intel: *IA-32 Intel Architecture Software Developer's Manual*, Intel Corporation, Volume 1, Cap. 9.

2 Intel: *IA-32 Intel Architecture Software Developer's Manual*, Intel Corporation, Volume 1, Cap. 11.

3 Gonzalez, R. and Woods, R (2001): *Digital Image Processing*, Prentice Hall, Cap 4

4 FFT - Transformée de Fourier Rapide,

http://magphy.ujf-renoble.fr/orbis/files/cours/2002/maitrise/cours_signal/Cours4.pdf