ISSN:1991-8178

Australian Journal of Basic and Applied Sciences

Journal home page: www.ajbasweb.com

Data Optimization and the Performance of Software: Concepts, Methods and Applications in Projects of Technological Innovation

[1]Hugo Ferreira Braga Tadeu, [2]Jersone Tasso Moreira Silva, [3]Carlos Alberto Arruda de Oliveira

[1]*Innovation and Entrepreneurship Centre, Fundação Dom Cabral, Av. Princesa Diana, 760. Alphaville, Lagoa dos Ingleses, Nova Lima, MG, Brazil, CEP. 34000.000.*
[1]*Innovation and Entrepreneurship Centre, Fundação Dom Cabral, Av. Princesa Diana, 760. Alphaville, Lagoa dos Ingleses, Nova Lima, MG, Brazil, CEP. 34000.000.*
[1]*Innovation and Entrepreneurship Centre, Fundação Dom Cabral, Av. Princesa Diana, 760. Alphaville, Lagoa dos Ingleses, Nova Lima, MG, Brazil, CEP. 34000.000.*

ARTICLE INFO

Article history:
Received 12 March 2015
Accepted 28 April 2015
Available online 5 May 2015

Keywords:
data optimization, software performance, software concepts, projects of technological innovation

ABSTRACT

Background: In this article the main concepts and methods related to the data optimization and the performance of software are presented, taking into account the characteristics of the workloads and the architectures of the computational systems. **Objective:** Present the main concepts of software performance optimization, the relationship between methods and techniques of optimization and the software development process. **Results:**From the start of the software development process attention should be given to the architecture to be utilized and the consideration of the optimization of data performance. The main techniques and optimization methods are analysed, at different levels of abstraction, singling out among these those that are based on the suitability of the characteristics of the workloads for the architectural characteristics of the computational systems. Techniques and methods related to the optimization in the source codes, in the compiling process and in the executable codes are analysed. Finally, some real and practical examples of optimization of applications, with different computational demands and focused on the software performance and projects of technological innovation are given. Practical examples of optimization of software performance, considering applications with low and high computational demand are also considered. Among the examples we single out the optimization of a source code and of an executable code utilizing appropriate tools, whose objectives are applications in technological environments. **Conclusion:** Evidence regarding the level of improvement of performance reached with the application of the optimization methods and techniques utilized, the measurements of execution times made before and after the optimizations, in applications of the digital processing of images are presented as concluding aspects.

© 2015 AENSI Publisher All rights reserved.

To Cite This Article: Hugo Ferreira Braga Tadeu, Jersone Tasso Moreira Silva, Carlos Alberto Arruda de Oliveira., Data Optimization and the Performance of Software: Concepts, Methods and Applications in Projects of Technological Innovation. ***Aust. J. Basic & Appl. Sci.,*** *9(11): 93-100, 2015*

INTRODUCTION

Before dealing with the contexts of software performance optimization or the relevance of this theme for the technological environment it is important to understand what is in fact software performance optimization. In computing, optimization is the process of modifying a system or software with the intention of improving its efficiency, this being the capacity to act and produce with the minimum of waste, expense or unnecessary effort (Almasi and Gottlieb, 1994).Therefore, the optimization of performance is a method of reaching the greatest efficiency of the software. In this manner, to identify what is the balance point between the cost and the performance becomes essential.

Before optimizing it is necessary to understand the problem or the scope of the software to be developed, so that one knows how much performance is necessary to it. It is very important to pay the proper attention to this aspect, because the performance inadequate for the computational demand of the software may affect even its functionality and the resolution of the problem proposed. In a hypothetical situation, the development of software faces risks, when it is released to the market, but without the due acceptance of the initial demands of the project and the workload (Cabral and Casey, 1993).

Corresponding Author: Hugo Ferreira Braga Tadeu, Innovation and Entrepreneurship Centre, Fundação Dom Cabral, Av. Princesa Diana, 760. Alphaville, Lagoa dos Ingleses, Nova Lima, MG, Brazil. CEP 34000-000. E-mail: hugo.tadeu@fdc.org.br

In this manner, it becomes extremely necessary to consider optimization of performance in the whole software development process. Even in the cases where the performance and its optimization have not been considered from the start of the software development process, even then some *a posteriori* measures can be applied. The methods and techniques to be presented, discussed and exemplified in the following sections will permit performance gains to be made even after the start of the process, according to the scope of the project and the phase at which is finds itself. Concern with optimizations, refers to the method applied in the process of software development, that comprises a series of techniques based on the architecture of the computational system in which is active mainly instruction level parallelism (ILP), processor level parallelism (PLP), use of message passing, (3 e 4), efficient use of memory hierarchy, computational instructions and resources involved.

The software development process (SDP) and the proper moment to consider the performance will be discussed in the Review of the Literature. This section will also present the main concepts of software performance optimization, the relationship between methods and techniques of optimization and the software development process. Also in this section some optimization techniques will be presented, among them: the use of in-line functions; *loop-unrolling*, the best way to go through matrices; *prefetch*, utilization of optimized processor in instructions and the fitting of the compiler to the machine architecture.

The examples based on an important operational digital processing of images called filtering or the *convolution* of images, with an ample possibility for the analysis of data and the performance of software are presented in the results section.

Finally the conclusions and contributions are presented, in addition to the perspectives and future work related to the theme present in this article.

Theoretical reference:

In this section the main concepts of software performance optimization are presented and in which stages of the process it is interesting to consider the aspects related to the optimization of performance. Evidence is provided that from the start of the software development process one should concern oneself with the definition of the architecture to be utilized and consider the optimization of performance.

But before dealing with the concepts of optimization of software performance themselves it is important to understand the software development process. In this way will be possible to evaluate when or at what stage of the development one should consider the ideal performance or its optimization.

Software Development Process and Artificial Intelligence:

A software development process (SDP) can be seen as a set of activities, practices and Trans formations carried out from the conception to the liberation of the product (Carlson et al., 1997; Bhandarkar andDing, 1997).

According to (Genius and Lelait, 2001), the software development process makes possible the control of the development process itself and provides the basis for the construction of quality software. And the construction of software includes the following stages:

- Analysis of requirements;
- Project;
- Construction of the program;
- Testing;
- Maintenance.

In Figure 1, one can observe what has been stated previously, that is, the software development process consists in the analysis of a problem, by means of a series of activities that utilize tools and procedures, to arrive at the solution of the problem.

According to Agrawal (1986), the software development process deals with three large generic phases of development: definition, development and maintenance.

The definition of a software process is concerned with the real problem to be dealt with. In this phase the software engineer tries to identify what the information is that must be taken into consideration in the system that is being developed, in other words it is concerned with the behaviour that the system should possess.

The development of a software process is concerned with the adoption of technologies for the solution of real problems. In this phase the software engineer projects the software that is being developed specifying how the system is to be implemented.

For Falbo (1996), there is no software process that can be applied in all situations. The technologies involved must be considered, the type of software, the domain of application, among other characteristics of the product and of the organization.

But whatever the process of development defined for an organization, the following phases must exist in this process: (1) analysis of requirements, (2) project, (3) construction of the program, (4) testing and (5) maintenance. These phases are important and necessary, because as previously mentioned, they make possible the control of the development process and provide a basis for the construction of quality software.

Understanding the phases that make up the software development process and that all are interconnected, a change in one phase generates changes in the other phases. For example, the modification of a requirement implies alterations also in the subsequent stages, such as in the project,

programme, testing and maintenance. This means that it is essential to identify at what moment it is necessary to consider the performance, because achieving ideal performance may become more expensive and more difficult, in addition to reducing the chances of success in greatly optimizing the performance. In the section below, the performance in the software development process is presented.

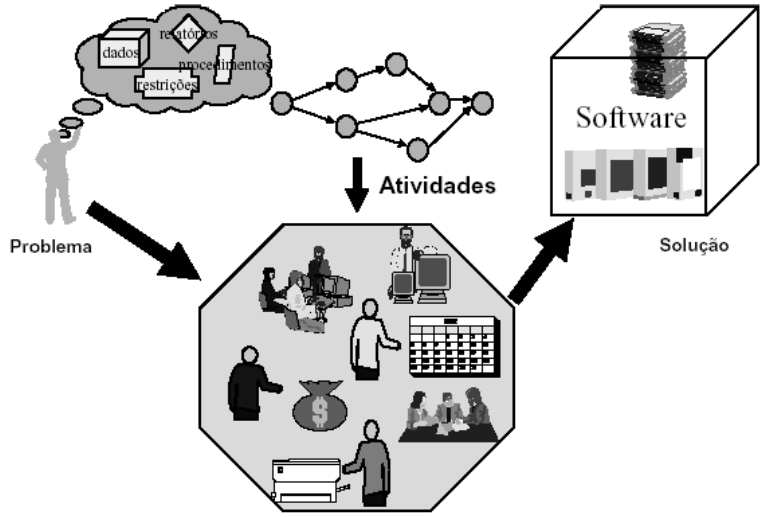


Fig. 1: Software Development Process.
Source:Genius and Lelait (2001).

Performance in the Software Development Process:

The optimization of performance may occur in one or more stages of the software development process, but it has to be foreseen previously, before starting the project, as the needs of different levels of performance may alter project choices, such as the manner of programming, among others. This being the case, the listing of requirements, characteristic of the first phase of the software development process, needs to deal with at what level the application will be, as an non-functional requirement. In listing the requirements it will be necessary to respond to the question of how much the software needs performance, so that the next phases can be planned and executed.

Gastaldo *et al.*(2004),provides evidence through a case study that the causes of reworking in the software development process are very often related to the non-functional requirements of performance, that are not met because they have not been incorporated since the start of the project. This generally occurs because the cost and the deadline are high due to the insertion of the non-functional performance requirement not projected initially. It is very often necessary to change the software architecture, the hardware and even the conception of the solution of the system to meet a non-functional performance requirement.

Non-functional performance requirements are those that refer to the speed of operation of the system. Different types of requirements may be specified, they being (Kung, Ruane and Yen, 1983):

- Response Time: specifies the time acceptable from the client's point of view for some operation to be concluded.
- Throughput: specifies the quantity of data that needs to be processed in a predefined time interval.
- Temporization: specifies how rapidly the system needs to realize the processing for it to be an entry of another subsystem.

Kung and Picard (1981), proposes a division of non-functional performance requirements into requirements of time and space. The requirement of response time is subdivided into requirements of response time and processing time (throughut) and the space requirement is divided into main memory and secondary memory.

In the face of that set out, the initial stage of the software development process (analysis of requirements) is perhaps the activity most critical to the development process, above all because it defines the characteristics of the software product to be generated. In this way, the incorrect definition of the requirements will lead to inadequate systems being obtained.

It is important and ideal to consider the software performance to be constructed as a requirement, making this clear at the start of the project in the listing phase and/or in the analysis of requirements. In this manner work is facilitated so that the project phases can be carried out with the objective of meeting also this requirement. This is essential, as it can even be necessary to change the software architecture, the hardware architecture and even the

conception of the solution of the system to meet a non-functional performance requirement not initially identified.

In this article, the importance of having a concern with and considering the performance and its optimization from the start up to the final program development process is emphasized, because in the listing of requirements it is necessary to perceive how much the software needs performance so that it will be necessary to deal with the optimization of performance during the development process. Only in this way can we guarantee that efforts will be expended in accordance with the real demand of the system.

To achieve these performance demands optimization methods and techniques related to the source code are considered, to the generation of codes directed to the architecture of the processors, to the process of compilation in the generation of executables and the optimization in the executable code. But before choosing the optimization method or the techniques it is necessary that the problem be characterized. From an understanding or characterization of the problem related to the application to be developed the methods and techniques can be identified by their potential for performance optimization.

Methods of Performance Optimization:

The methods of performance optimization refer to their application in the software project in accordance with Figure 2.

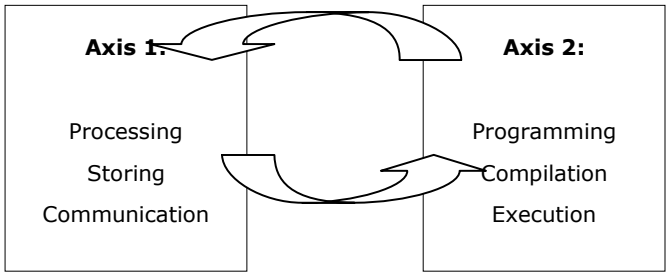


Fig. 2: Methods of Performance Optimization.
Source: Hopf and Ertl, (1999).

A method of performance optimization can be seen as a combination of the processing, optimization, communication, programming, compilation and its execution. A determined optimization technique can be utilized at different phases of the software development. For example, a specific technique can be used to optimize the performance in programming time or maybe utilizing another technique for the compilation, reaching the same target.

It is emphasized that there is no specific method to be utilized in all cases in which one desires to optimize software performance. Really, the method is related to the software development process, as it refers to the choice of specific techniques to be utilized or applied in determined stages of the development process in accordance with the problem and the scope of the software. The main techniques of performance optimization are now presented::

- In-line functions: adoption of open codes and redundant systems, reducing the software execution times, making them more efficient, in accordance with (Jain, 1992) and (Jamro and Wiatr, 2001).
- *Loop-unrolling*: systems for the reduction of interaction, improving the performance of the software, in accordance with (Eyre,1998) and (Diefendorff and Dubey,1997).
- *Prefetch*: consists in utilizing the data in the computer's memories, before the processors are started. In this way, the processing capacity is improved and made more agile.
- SIMD (Single Instruction Stream, Multiple Data Streams) Instructions: these are specific instructions in modern processors, using the capacity for optimization and data processing in accordance with (Doshi and Varman,1987).

In general, the principle techniques for performance optimization should be used to help in the software performance and in the data analyses for real problems.

MATERIALS AND METHODS

The present study is a qualitative piece of research consisting of a review of the specialized literature, done between February 2013 and December 2014, in which international periodicals were consulted, in addition to adopting tests based on operations of the treatment of images in computational systems. The survey in these periodicals attests to the search for up-to-date research, future trends and aspects related to its applicability in projects of technological innovation.

Following this, it was sought to understand the results of the software performance optimization, highlighting the treatment of images and data analysis in high computational demand.

Results:

In this section we present the optimization of software performance applied to innovation projects highlighting technological innovation for the

treatment of images, considering data analysis in high computational demand. We highlight the optimization of a source code and of an executable code utilizing the tool *Microsoft Visual C++*, *Dev C++* and *AMD CodeAnalyst Performance Analyzer*. To demonstrate the level of improvement reached through the methods and optimize techniques chosen, we carried out the measurements of the times spent before and after optimization of the image data.

Real Problem:

The results presented in this section are based on an important operation of digital image processing called image filtering.

This operation was chosen to manipulate data in the form of matrices, and that possess, in their turn, many repetition loops. Normally, the repetition loops are the points where we can obtain significant improvements in terms of performance, seeing that each optimization is multiplied by the number of times the loop is executed. We also have the optimization in these cases that are utilized in the repetition lopes.

When the filtering operation occurs in the space domain it is called convolution. The term space domain refers to the junction of pixels that are part of an image and the operations in the space domain are the procedures applied directly to these pixels.

The operation of convolution is carried out for each pixel in the image, in which the convolution is applied in accordance with the pixels of the mask. Depending on the filter (mask) the convoluted image may be smudged, highlighting its borders and the noises removed.

To exemplify the utilization of the techniques, Figure 3 presents the stretch of code of a convolution (filtering) of images as a basis for the results reached. That code was written in the C/C++ programming language.

```
#define  TIPOIMG        unsigned char
//Matrizes Imagem e Temp do tipo TIPOIMG
int Limitacao;
int Tam_Mascara_Filtro;
int FatorDivisao;
int MaxLinhas;
int MaxColunas;

...
//Funções
TIPOIMG Convolucao_Pixel( int i_coluna, int i_linha)
{
  float Aux = 0;
  for(int col_filtro = 0; col_filtro < Tam_Mascara_Filtro; col_filtro++)
    for(int lin_filtro = 0; lin_filtro < Tam_Mascara_Filtro; lin_filtro++)
      Aux += Filtro[col_filtro][lin_filtro] *
             Imagem[col_filtro + col_img - Limitacao][lin_filtro + lin_img - Limitacao];
  Aux /= FatorDivisao;
  if(Aux >= 0 && Aux < MAX_TAM_CINZA)
    return( (unsigned char) Aux);
  else
    if(Aux < 0)
      return( 0);
    else
      return( MAX_TAM_CINZA - 1);
}
void    Convolucao( void)
{
  for(int col_img = Limitacao; col_img < (MaxColunas - Limitacao); col_img++)
    for(int lin_img = Limitacao; lin_img < (MaxLinhas - Limitacao); lin_img++)
    {
      Temp[col_img][lin_img] = Convolucao_Pixel( col_img, lin_img);
    }

  for(int coluna = Limitacao; coluna < (MaxColunas - Limitacao); coluna++)
    for(int linha = Limitacao; linha < (MaxLinhas - Limitacao); linha++)
      Imagem[coluna][linha] = Temp[coluna][linha];
}

...
//main
 //Carrega "arquivo entrada" contendo imagem para Imagem

 Limitacao = Tam_Mascara_Filtro / 2;
 FatorDivisao = Tam_Mascara_Filtro * Tam_Mascara_Filtro;
 Convolucao( void);

 //Salva imagem saída para "arquivo saída"
```

Fig. 3: Source Code Basis of the Convolution.
Source: authors (2015)

The code of Figure 3 is composed of two functions: (1) one that does the convolution of a pixel and (2) another that does the convolution of an image. In the main function, after the declaration of the variables, image load of the file and filter load,

we call the function one of convolution and then write the resulting image in an exit file.

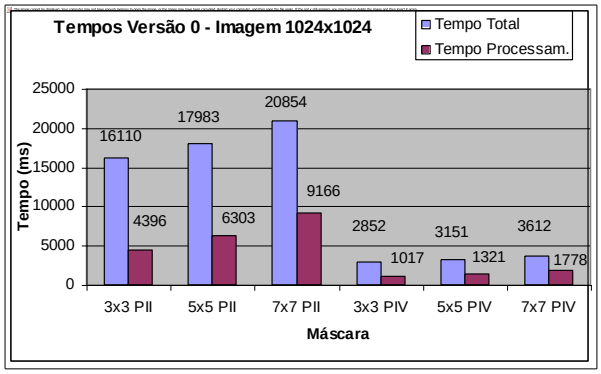
Algorithm optimization techniques for optimizing data presented in Figure 3 are now presented.

Utilization of Algorithm techniques:

For the utilization of algorithm techniques attention should be directed to the compiler which will interpret it. As the compiler already optimizes some parts of the algorithm, it is not necessary to have complex codes which are difficult to maintain. The interesting thing is to concern oneself with the alterations that help the compiler to generate a more efficient executable code. For this purpose it is necessary to execute some functions: (1) allocation of matrices in the memory, (2) parallelism of the processor, (3) techniques of the compiler and (4) code generation technique.

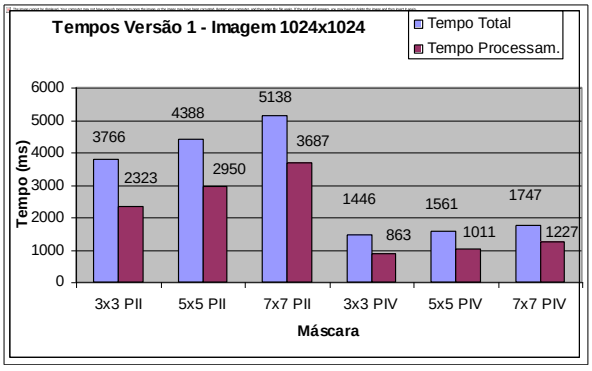
For the unique innovation project focused on the treatment of images for the hospital sector, the results obtained were as follows:

Graph 1 shows the execution times of images, when the size of the filter mask increases, execution time is proportional, due to the number of operations also increasing. Note that the proportion of processing time in relation to total time increases when the filter mask size is greater, because the number of operations in the processing of the image increases considerably.



Graph 1: Image Processing Time Version 1.

In Graph 2 on the other hand the performance is better due to the architectural characteristics of the computer utilized, involving the processor, main and secondary memory, proportioning better performance in the execution of the code.



Graph 2: Image Processing Time Version 2.

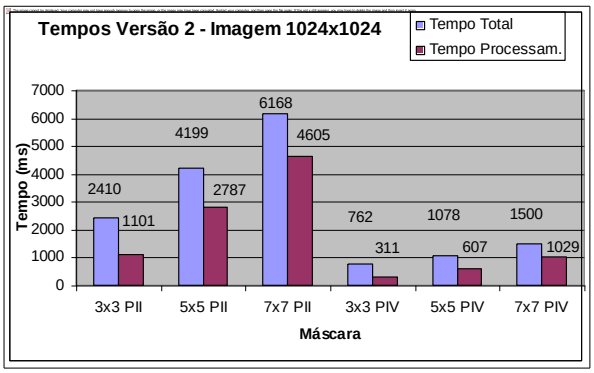
Graph 2 shows the execution times of version 1 in computers whose use of the source code optimization has improved the performance of the application considerably.

Graph 3 presents the times obtained in the execution of version 2 in computers with superior velocity compared to Version 1 with a gain in time higher than six times.

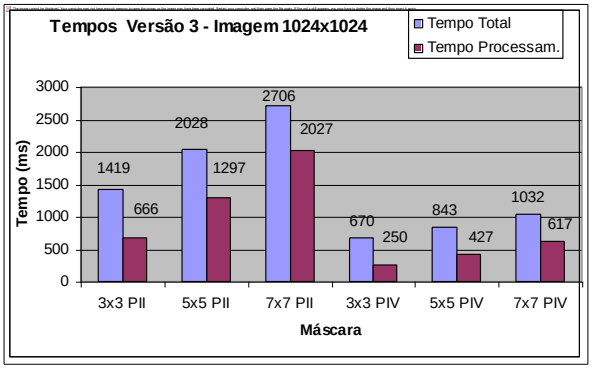
Graph 4 shows the execution times of version 3 in a computer with processors, in which its version was optimized fitting the executable code to the architectural characteristics of the two available processors. Performance improved considerably when compared to the original version. Maximum velocities attained are analysed in Graphs 5 and 6.

Graph 5 shows the maximum velocity obtained using image resolution with greater performance compared to the previous Graphs. As a proposal for future work research is suggested on the processing velocities, the images used and possible risks in the minimization of optimization of images.

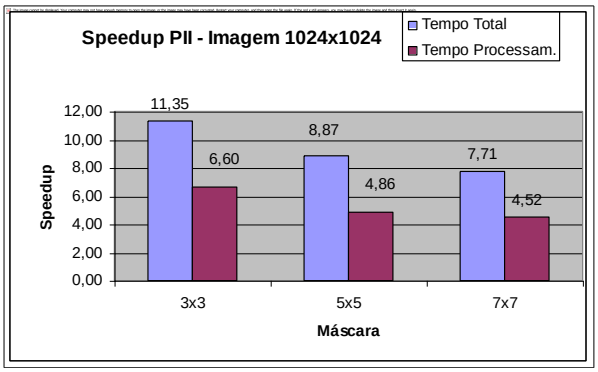
Graph 6 shows the processing velocity attained using greater image resolution for the computers available for testing.



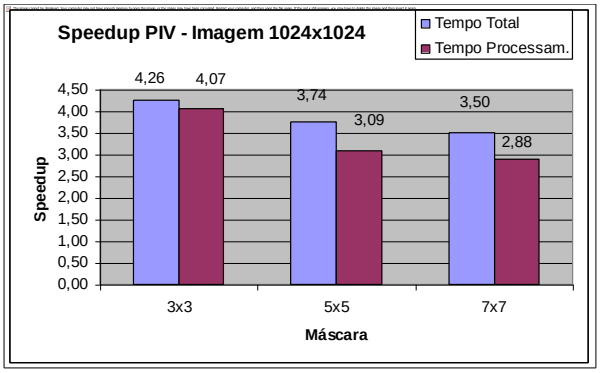
Graph 3: Image Processing Time Version 3.



Graph 4: Image Processing Time Version 4.



Graph 5: Image Processing Time Version 5.



Graph 6: Image Processing Time Version 6.

Conclusions:

This article has suggested that one should think of optimization of software performance from the start of any software development process (SDP) until the a posteriori programming stages. The fitting of the architectural characteristics of the computer to be utilized in the execution of software developed is extremely important and should make up part of the SDP. It is also suggested that it is possible to obtain quite significant gains in relation to the execution performance, although there are still improvements to be analysed and proposed as studies for future work.

It is believed that the main contributions of this article will be to divulge the utilization of performance optimization related to architecture and parallel processing; to emphasize that the software development process should take into consideration the possibilities of software performance optimization considering the suitability of the same to the architecture in which it will be executed; present the application of performance optimization in real applications for example in technological innovation projects.

It is suggested that the path to obtaining high performance is not always the utilization of clusters and multiprocessors. One can also use architectural characteristics related to the internal parallelism of the processors and obtain gains in technological optimization.

As a proposal for future studies, we suggest the utilization of other optimization techniques for better performance of the applications adopted here, highlighting technological innovation and future applications in business environments.

REFERENCIES

Agrawal, D.P., 1986. Advanced Computer Architecture. Computer Society Press.

Almasi, G.S. and A. Gottlieb, 1994. Highly Parallel Computing, Benjamin-Cummings Publishing Co., Inc. Redwood City, CA, USA. Second edition.

Bhandarkar, D. and J. Ding, 1997. Performance characterization of the pentium pro processor. In HPCA97: 288-297.

Cabral, B. and L. Casey, 1993. Imaging vector fields using line integral convolution. Proc. Of SIGGRAPH, 93, in Computer Graphics 27, Annual Conference Series, ACM, 1(1): 263-272.

Carlson, D.A. *et al.*, 1997. Multimedia Extension for a 550MHz RISC Microprocessor. In IEEE Journal of Solid-State Circuits.

Carr, S., K.S. McKinley and C.W. Tseng, 1994. Compiler optimizations for improving data locality, ACM Computing Surveys, Proceedings of the sixth international conference on Architectural support for programming languages and operating systems, 1(1): 252-262.

Diefendorff, K. and P.K. Dubey, 1997. How Multimedia Workloads Will Change Processor Design. In IEEE Micro, 1(1): 43-45.

Doshi. K. and P. Varman, 1987. A Modular Systolic Architecture for Image Convolutions, ACM Computing Surveys, Proceedings of the 14th annual international symposium on Computer architecture, 1(1): 56-63.

Eyre, J., 1998. Assessing General Purpose Processors for DSP Aplications. Berkley Design Technology Inc. Presentation.

Falbo, R.A., 1996. Automatização do Processo de Desenvolvimento de Software, In: ROCHA, A.R.C. and WEBER, K.C. (eds.), Qualidade de Software: Seleção de Textos, CITS.

Gastaldo, D.L. *et al.*, 2004. Processo de Engenharia de Requisitos Aplicado a Requisitos Não-Funcionais de Desempenho – Um Estudo de Caso, Departamento de Engenharia da Computação e Sistemas Digitais - Universidade de São Paulo.

Genius, D. and S. Lelait, 2001. A Case for Array merging in Memory Hierarchies, Proceedings of the 9th workshop on Compilers for Parallel Computers (CPC'01), Edinburgh, Scotland.

Hopf, M. and T. Ertl, 1999. Accelerating 3D Convolution using Graphics Hardware, IEEE Computer Society, Visualization '99. Proceedings, 1(1): 471-564.

Jain, R.K., 1992. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.Wiley- Interscience, New York, NY, USA.

Jamro, E. and K. Wiatr, 2001. Implementation of Convolution Operation on General Purpose Processors, IEEE Computer Society, Proceedings of the 27th Euromicro Conference, 2001. Proceedings, 1(1): 410-417.

Kung, H.T. *et al*., 1983. A Two-level Pipelined Systolic Array for Multidimensional Convolution, Image and Vision Computing, 1(1): 30-36.

Kung, H.T. and R.L. Picard, 1981. Hardware pipelines for multi-dimensional convolution and resampling, in Proc. IEEE Comput. Soc. Workshop Comput. Architecture for Pattern Anal and Image Database Management, 1(1): 273-278.