INTEGRATING MODELING AND MEASUREMENT IN THE MAPPING OF SIGNAL PROCESSING ALGORITHMS TO ADVANCED ARCHITECTURES

By

Nayda G. Santiago Santiago

A DISSERTATION PROPOSAL

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical and Computer Engineering

June 1999

ABSTRACT

INTEGRATING MODELING AND MEASUREMENT IN THE MAPPING OF SIGNAL PROCESSING ALGORITHMS TO ADVANCED ARCHITECTURES

By

Nayda G. Santiago Santiago

Signal processing applications are frequently computationally intensive tasks. When developing algorithms for these applications, implementation issues should be considered in order to obtain efficient coding. However, this imposes an additional burden on the signal processing specialist who, in most cases, is more concerned about correctness and effectiveness issues.

This proposal discusses studying the mapping process of algorithms to high performance platforms, to correlate the different options that might be available to the signal processing specialist. As an outcome, it is expected to produce high-level representations of the available mapping space, leading to the selection of the best alternatives to obtain an efficient implementation. To effectively carry out this study, it is proposed to incorporate both system modeling and measurement in the process of implementing algorithms on different advanced architectures. A plan for the review of related work, problem statement, plan for the research, and timeline with the expected duration of the different activities are presented.

TABLE OF CONTENTS

1 Introduction **1**

2 Problem Background **4**

2.1 Compressed Domain Techniques for Image Processing 5

2.1.1 JPEG and compressed domain techniques 6

2.1.2 Wavelet-based Compression and Texture Analysis 8

2.2 Implementation of Algorithms 12

2.3 Advanced Architectures 14

2.4 Environments and Tools 17

2.4.1 Problem Solving Environments 17

2.4.2 Performance Modeling 19

2.4.3 Tools 20

3 Problem Statement **23**

4 Research Plan **27**

4.1 Tasks 27

4.2 Timeline 29

4.3 Work in Progress 29

5 Anticipated Contributions **32**

BIBLIOGRAPHY **33**

CHAPTER 1

Introduction

Signal processing research has led to a number of revolutionary applications which have changed communications, medicine, defense, marketing, and even entertainment. As a consequence of this trend, media processing has steadily become one of the dominant computing workloads nowadays [1]. The development of algorithms for signal processing applications frequently involves a process of mathematical study, proof, simulation, and implementation. Researchers involved in algorithm development are, in most cases, signal processing specialists who typically have a greater expertise in the mathematics of the problem than in computing. When a signal processing specialist makes implementational decisions to select a specific factor such as a platform, or a computer language, or diverse compiler options, or a programming paradigm, among others, the performance of the implementation is affected.

A common process followed in the implementation of an algorithm is going from the highest level of abstraction to mapping the procedure into the selected target architecture. If the algorithm is inherently complex, it could require implementation on a high performance system, extending the mapping process. Further, new algorithm implementations should have improved qualities, such as efficiency and performance. All these requirements, when combined together, make the mapping process a challenging task, calling for the need of friendly tools to assist the signal processing specialist.

Problem solving environments (PSEs) are an emerging area in computational science and engineering aimed at reducing the turnaround time in the production of efficient code

for computationally intensive scientific applications [2-8]. According to Rice and Boisvert, a problem solving environment is an integrated system that contains all the facilities required to solve a specific class of problems in an efficient and convenient way [3]. The availability of such type of systems for the type of problems found in the area of signal processing would provide a great aid in the development and implementation of new algorithms. Several attempts have been reported towards the development of such type of tools [9-11]. However, none of them have effectively addresses the problem of producing a friendly, efficient tool integrating performance measures and supporting the development of new algorithms.

The central objective of this research is to study the mapping process of a class of signal processing applications into high-performance architectures, and to develop a set of high-level abstractions such that, given a specific problem and a target architecture, a set of guidelines are produced which would lead to an efficient implementation. The abstractions resulting from this research will combine information from performance modeling, performance monitoring, and mapping techniques in a coherent way such that they can be either incorporated with an existing PSE or be used as the core for the development of a new PSE in the selected signal processing area.

The tools to be developed will have as target application the areas of image processing and pattern recognition. A case study has been selected for the development of new algorithms for wavelet-based feature extraction and texture analysis for database image retrieval. These extracted features are intended to be mapped to the compressed domain and produce faster database retrieval mechanisms.

In summary, this research will explore the solution space for the process of designing and mapping texture analysis algorithms to advanced architectures. The process is supported by a problem-solving environment, which increases productivity, and the objective is to improve the efficiency of the implementation. The solution space consists of platform configuration options at different levels in the mapping process, performance monitoring results, and algorithm alternatives. The goal is to determine correlation and provide appropriate feedback to the signal processing specialist to guide him/her in algorithmic decisions.

The rest of this document is organized as follows. The next chapter presents previous research reported in the areas of this proposal. Chapter 3 contains a detailed statement of the problem, followed by a research plan and a timeline in chapter 4. The last chapter contains a discussion of the expected contributions of this research.

CHAPTER 2

Problem Background

Diverse solutions to the problem of implementing algorithms to different architectures have been reported. Different environments and tools have been developed to assist computer programmers in the mapping process. This chapter presents background information on the previous work reported in these areas and the signal processing application area addressed by this research.

To properly understand the mapping process, it should be tied to a specific application area. The first section of this chapter provides a discussion on the application addressed in this research: compressed domain techniques for image processing. It first discusses the current standard JPEG compression and some compressed domain techniques developed for JPEG. Wavelet-based compression and texture analysis are discussed next.

A discussion on implementation of algorithms and current advanced architectures follows. This information provides the background for the architectures to be used in this research. Finally, a section on environments and tools related to the proposed work is presented. A description of the current state of problem solving environments for signal processing applications, performance models and tools, and the message passing interface for parallel programming is given.

2.1 Compressed Domain Techniques for Image Processing

Multimedia applications include images and video as standard components. Most visual data is stored in compressed format to improve storage capacity and transmission [12]. Image processing applications requiring real-time response have two main problems: the volume of data to manipulate is large and the compression and decompression algorithms are complex. This is computationally demanding and therefore media processing is becoming one of the dominant computing workloads of the future [1].

Research is currently being developed in the area of image manipulation in the compressed domain [12-16]. Compressed domain methods imply that the image is compressed or only partially decompressed when it is manipulated.

The advantages of compressed domain techniques are mainly three:

- Reduced computational cost
- Elimination of the overhead of coding/decoding
- Some compression algorithms aid in the image content analysis

Compressed domain techniques presented in literature are applied to images compressed in the JPEG (Joint Photographic Experts Group) standard format. JPEG is based on the computation of the discrete cosine transform. Different compressed domain techniques can be applied to other compression methods such as wavelet image compression.

One of the problems faced when retrieving images in databases is the consistency in the keywords used to describe the image [17]. These will depend on the person describing it. Image features such as shape, color, or texture can be used for retrieving images from databases to solve this problem.

Image compression using the discrete wavelet transform has been very effective, especially at low bit rates [18-22]. The use of the discrete wavelet transform in the analysis of texture has proven successful in current research work [23-28]. The synergy between image compression and feature extraction using the discrete wavelet transform can be exploited to improve performance of algorithms [14].

The following sections will provide background on current compressed domain techniques for JPEG and wavelet compressed images.

2.1.1 JPEG and compressed domain techniques

The algorithm for JPEG standard image compression starts by partitioning the image in blocks [15, 29]. Each block is an 8×8 matrix of values in the range $[0, 255]$. The block is normalized to bring the values to the range $[-128, 127]$. The Discrete Cosine Transform (DCT) is computed on each 8×8 matrix using equation (2.1). Let $y[i, j]$ denote the input signal. The DCT is defined as

$$Y[u, v] = \frac{1}{4} \sum_i \sum_j C(i, u) C(j, v) y[i, j] \tag{2.1}$$

where

$$C(i, u) = A(u) \cos \frac{(2i + 1) u\pi}{16} \tag{2.2}$$

$$A(u) = \begin{cases} \frac{1}{\sqrt{2}} & u = 0 \\ 1 & \text{otherwise} \end{cases} \tag{2.3}$$

The lowest frequency components of the DCT are in the upper left corner of the matrix and the highest frequency components are in the lower right corner. The next step in JPEG is the quantization of the DCT elements based on a quantization table. The quality of the image depends on the values in the quantization table. After the quantization step, the 8×8 matrix is converted into a 64-element vector by zig-zag ordering. This ordering places the lowest frequency component at the beginning of the vector, and sorts the components in increasing frequency content.

Once the zig-zag ordering is done, the resulting vectors contain a large number of zeros. Run length encoding (RLE) is applied to encode the vector into $(skip, value)$ pairs. The final step is to use entropy coding (for example, Huffman coding) to compress the RLE blocks.

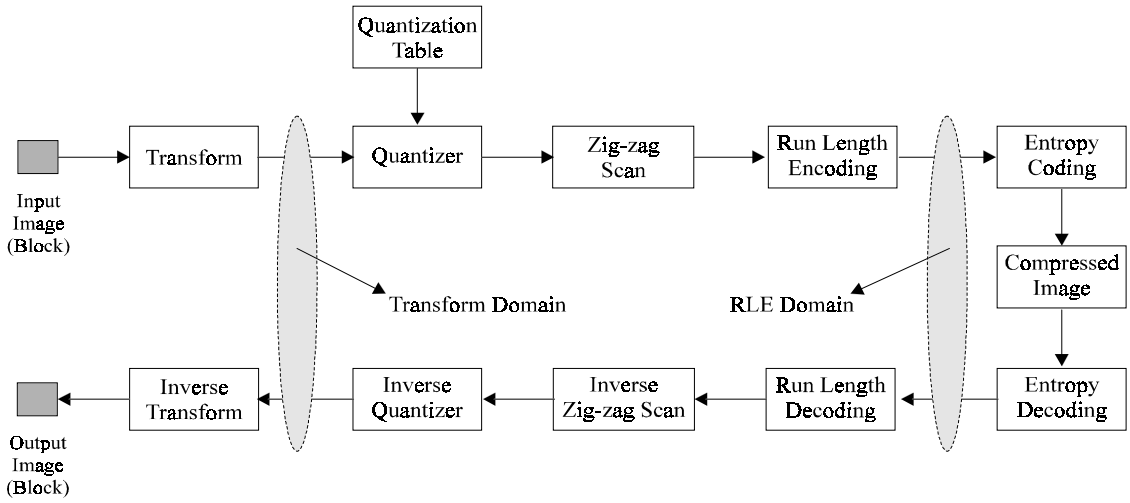

Figure 2.1. Relationship between the transform domain, the RLE domain, and the compression algorithm.

Compressed domain techniques are based on the properties of the JPEG and the DCT. The properties are studied to map image manipulation from the space-time domain to the compressed domain. Most of these methods operate in the transform domain while others operate on the run-length encoded data (see figure 2.1).

Two examples of compressed domain techniques are the work of Smith and Rowe [15] and the work of Shen and Sethi [30]. Smith and Rowe present the implementation of four basic algebraic operations in the compressed domain in [15]: scalar multiplication, scalar addition, pixel addition, and pixel multiplication. These operations are applied on the RLE domain. These basic operations will build the foundation for other image manipulation methods in the compressed domain.

Shen and Sethi study direct feature extraction from compressed image for edge detection and coarse classification in [30]. These are obtained in the DCT transform domain by developing a model of an edge. The first feature to compute in order to do classification is the location of the area of interest. This will provide the blocks containing edges in the image. Once the blocks are extracted, three parameters are of interest to detect edges: edge orientation (θ), edge strength (h), and edge offset from the center (d). These will determine

the edge strength. The stronger areas are classified as edges by thresholding. Experiments show that this approach is much faster but not as precise as the spatial domain edge detection algorithms. This method is good for coarse image classification.

2.1.2 Wavelet-based Compression and Texture Analysis

Wavelets are functions that are used to hierarchically decompose data or other functions. These functions satisfy certain mathematical requirements. This is similar to Fourier decomposition of functions in terms of sine and cosine functions. However, unlike Fourier analysis, wavelets analyze data according to scale or resolution. If we look at a signal at a coarse scale, we can see gross features, while if we look at it at a fine scale, we will see its details [31]. This characteristic makes wavelets useful for approximating data with discontinuities like images. The more adapted the wavelet is to the data, the more sparse the representation in the wavelet domain is. This makes wavelets well-suited for compression and feature extraction in images [31].

In order to explain the wavelet transform of an image, we will start explaining the wavelet transform of a one-dimensional signal and then expand to a two-dimensional signal. It should be noted that we are limiting this work to wavelets in $\mathcal{Z}_N$ (space of cyclic signals).

Wavelet transform of a 1-D signal

Wavelets are basis functions in continuous time that have the special feature that all functions are constructed from a single mother wavelet. This mother wavelet is a small wave (short signal or localized in time). In the wavelet transform, a function is represented as a superposition of dilated and translated wavelets. The discrete wavelet transform on $\mathcal{Z}_N$ is a linear operator that transforms a vector into another of the same length.

There are infinite number of functions that satisfy the properties of wavelets. They differ in how localized they are in time or frequency. This means that there are wavelets that emphasize sharp changes of the signal, other emphasize smooth changes, ones that give more information about changes in time, and others that give more information about frequency content of the signal. Daubechies compact support wavelet is very well known.

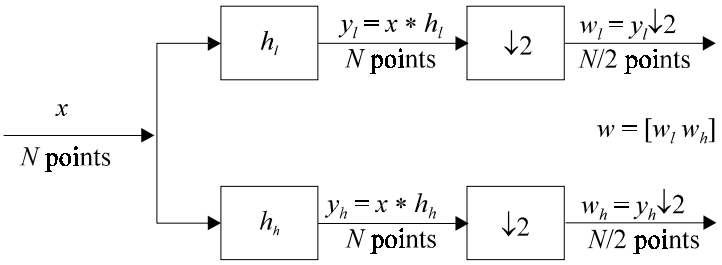

Figure 2.2. Block diagram of 1-D wavelet transform. One level of transform.

Compact support implies that it does not extend to $\pm\infty$, and the filter to compute the wavelet is a finite impulse response filter. A finite impulse response filter (FIR) is a filter whose output is the convolution of a **finite** number of filter coefficients with the input signal.

The computation of the wavelet coefficients consists of two steps: filtering the signal and decimating the output of the filter. In *decimation by two*, every other element of the signal is obtained at the output. Filtering and decimation are repeated recursively to obtain different resolution levels [32]. The scheme to compute the wavelet transform is shown in figure 2.2. First, the signal is filtered with h_l (low pass filter) and h_h (high pass filter) and then decimated. The wavelet transform is $w = [w_l\ w_h]$.

So far we have only computed the finest scale of wavelet coefficient. To compute a coarser scale, we have to repeat the same process for the low frequency component part of the signal.

Figure 2.3 shows the recursive computation of the wavelet transform. This process is repeated on the low frequency component of the signal until a final scale is reached. This is called a *pyramid decomposition*.

Wavelet transform of a 2-D signal

The standard wavelet decomposition of a 2-D signal or image first computes the one dimensional wavelet transform in each row of the image obtaining two images. Then the one dimensional wavelet transform of the columns of these two images is obtained [33]. This

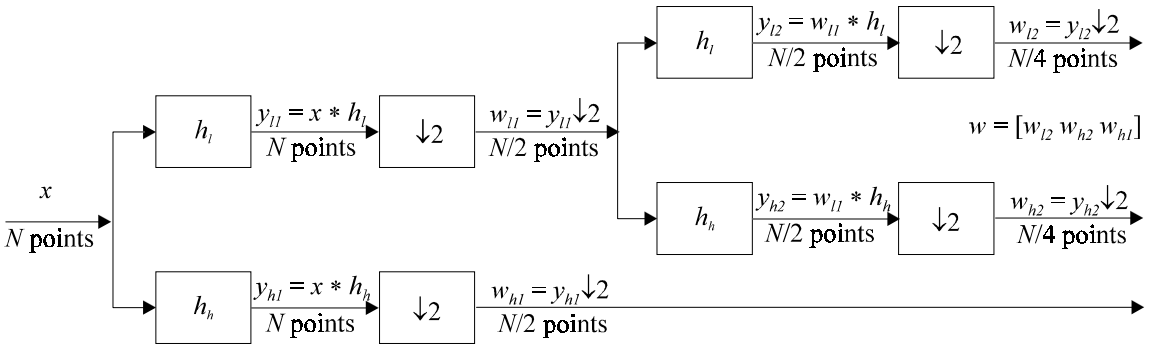

Figure 2.3. Two level decomposition wavelet transform.

procedure is repeated on the low frequency image to generate the next coarser scale of wavelet coefficients.

Figure 2.4 illustrates a block diagram showing the computation of the two-dimensional discrete wavelet transform.

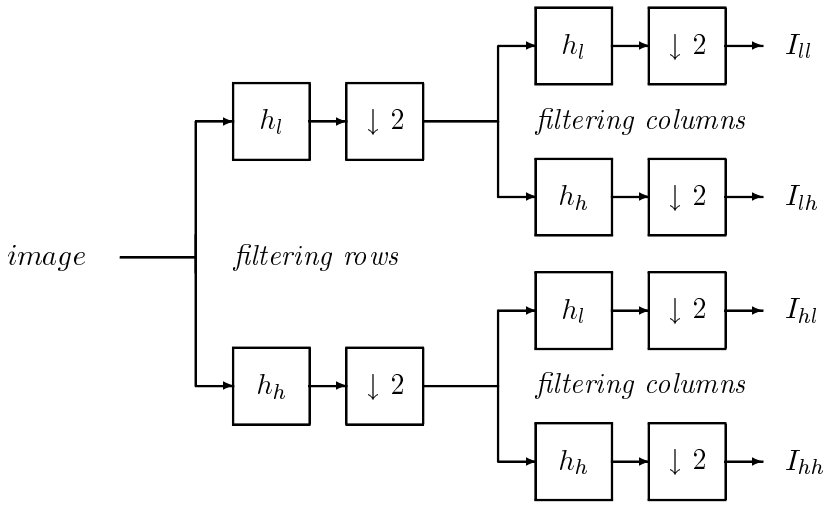

Figure 2.4. Block diagram of one level of decomposition of a two-dimensional wavelet transform.

The filtering-downsampling steps are repeated on the ll coefficients. This can be repeated $k = \log_2 M$ times where M is $\min\{Num_rows, Num_cols\}$ to give a level-k wavelet decomposition.

Inverse Wavelet Transform

To obtain the inverse wavelet transform, the columns of the image are interpolated or upsampled and then filtered with a synthesis filter. Then the rows are interpolated and filtered. In interpolation by two, a zero is added after every element of the vector. Figure 2.5 shows the block diagram of the synthesis stage of the wavelet transform.

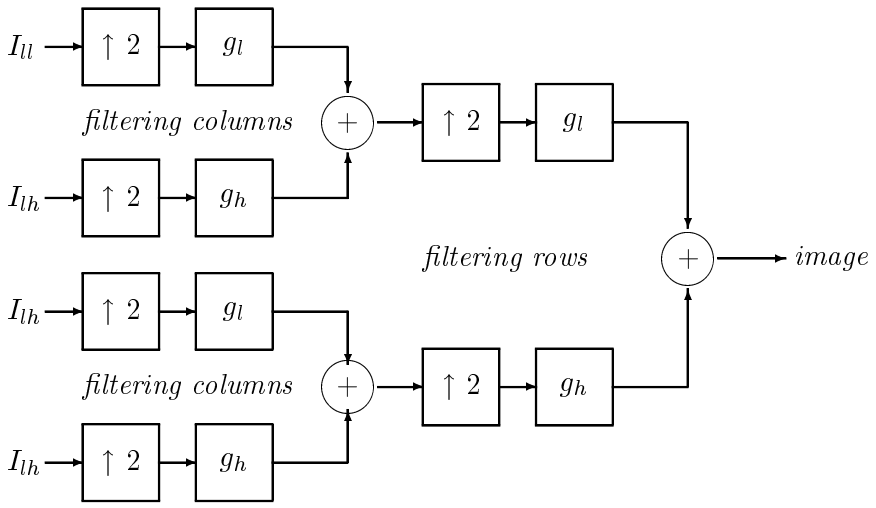

Figure 2.5. Block diagram of one-level synthesis of a two-dimensional wavelet transform.

Compression

Some properties of the discrete wavelet transform make it effective for image compression. First, since the wavelet representation is localized in both time and frequency, it provides a compact representation of the image. Wavelet bases can represent singularities so edges are well represented in the wavelet transform. Second, it clusters insignificant coefficients

[22]. This can be exploited in the compression scheme like, for example, the scheme used in embedded zerotree wavelet (EZW) compression [18]. Third, multiresolution analysis allows fast coarse image browsing. Compression methods based on wavelets usually outperform DCT-based JPEG, specially at low bit rates [22].

Texture analysis

As the size of image databases increases, the need for faster algorithms for image retrieval become important. The use of keyword description of images is discouraged by the lack of consistency in these descriptions [34-35]. A statistical approach using feature selection and similarity measures is suggested for this purpose [34]. Texture is one of the features that distinguish images.

There are three different statistical methods for texture classification problems: image-based, model-based, and transform-based statistical algorithms [36]. This classification is based on how the features are computed. The purpose of transform-based methods is to determine the spatial-frequency component of the signal which indicates texture. Multiresolution analysis is suitable for this purpose and has been used for texture analysis [23-28].

Once the image processing algorithm is developed, the next step is implementation on a specific architecture.

2.2 Implementation of Algorithms

An algorithm is a set of steps to solve a specific problem. It is an abstraction of a computer program. This abstraction removes the constraints of a particular hardware or high-level language from the description on how to solve a given problem [37].

The use of an algorithm as a high-level of abstraction allows portability and faster development time of software. With the rapid changes in computers, algorithms remain static while architectures change. However, there is a trade-off between high level of abstraction and execution time. Tuning a program to a particular computer often requires knowledge of the computer architecture and instruction set architecture.

An implementation is the task of turning an algorithm into a computer program [38]. There is no standard procedure on how to do this. There are a set of decisions to make in the process such as which computer, language, compiler, etc. It could be easy or difficult based on these decisions. One abstraction can have different implementations.

Once the implementation is done, the first step is to determine if results are correct [39]. Results should be what was expected. The second step is to determine if the implementation is efficient. Efficiency is typically measured in terms of execution time and memory space [37]. Execution time is mainly determined by the amount of data and number of operations to perform. A rough estimate of efficiency can therefore be obtained prior to implementation by doing a complexity analysis, that is, doing a mathematical analysis of the estimated number of operations given the amount of data.

The efficiency of an algorithm can also be determined after implementation using monitoring tools. These will provide detailed information on the behavior of the algorithm and quantify successes or failures in the implementation [40]. Unfortunately, to understand performance information provided these tools it is required to have knowledge about the computer architecture . This is in contrast to the concept of abstraction. Poor memory access patterns or improper compiler optimizations can prevent a program from being efficient. Performance tools that can relate high-level abstractions to efficiency are needed.

A methodology for mapping mathematical algorithms to platforms was introduced in [41]. This methodology presents the concept of a computational mathematical environment which basically consists of an input data set, an output data set, a set of operators acting on this set, and a set of rules for these operators. In this environment the parameters of the computing platform are presented to a computational mathematical framework to assist in the mapping process. We will extend this work to incorporate performance metrics to the computational mathematical framework.

2.3 Advanced Architectures

The computing power requirements for scientific applications have led to the development of different approaches for meeting the demands of processing speed, memory size and speed, and data input/output rates. Increases in performance have come from several advances [42]. Some of these advances include:

- Microprocessors have become faster through the use of instruction-level parallelism, multilevel caches, and faster clock speeds.
- Different schemes have been developed for effective interconnection between processors and memory.
- Users have learned how to use multiple processors and deep memory hierarchies.
- Software tools have been improved.

Advances in microprocessor technology include the development of the Intel Pentium family, the Motorola PowerPC family, and the Digital Equipment Corporation's Alpha 21164 microprocessor [43]. These are 32 or 64-bits microprocessors with clock speeds higher than 233 MHz.

Taking as an example the Intel Pentium family, it includes the Intel Pentium, Pentium Pro, and Pentium II. The Pentium is a superscalar level 2 processor. This is, it can execute two instructions per clock cycle. It uses two 5-stage pipelines, two 8K internal caches, one for code and one for data, and two translation lookaside buffers (TLB) to handle the pipelines. It prefetches up to 32 bytes of instructions for branch-prediction [43]. The Pentium Pro is a superscalar level 3 processor. It has three 12-stage pipelines to execute three instructions per clock cycle. This processor also combines branch prediction, speculative execution, and analysis for data dependencies to improve program execution over the Pentium and to prevent stalling. This last feature is called *dynamic execution*. Finally, the Pentium II combines the characteristics of the Pentium Pro and the MMX (**m**atrix **m**ath e**x**tensions) technology. MMX technology is intended for accelerating multimedia and communications

applications [44]. It contains special instructions aimed at efficiently manipulating video, audio, and graphics by exploiting the inherent parallelism present in these applications. These instructions operate on the data in a single-instruction multiple-data fashion. Like the Pentium Pro, the Pentium II uses dynamic execution and dual independent bus architecture (DIB). DIB consists of two independent buses, one for cache and the other for the system to accelerate the access of information.

Scalable parallel processors (SPP) are also used to meet performance demands. SPPs are hundreds or thousands of state-of-the-art interconnected processors. The goal of SPPs is to obtain fast computers through highly parallel designs and substantial parallelism within the processor. Issues to consider when designing SPPs are whether to use Single-Instruction Multiple-Data (SIMD), Multiple-Instruction Multiple-Data (MIMD), and Very Long Instruction Word (VLIW). In the SIMD architecture, the same instruction is executed on multiple processors at the same time. In MIMD, different instructions and data are operated on by each processor. In the VLIW architecture, multiple functional units are exercised by the instructions on each cycle. The instruction word is composed of simpler instructions which are executed simultaneously. The real challenge in SPPs is how to perform at the potential the system has. Some of the challenges include decisions on the interconnection hardware, memory system design, compilers, and algorithm design [42].

Interconnection hardware could be a custom design, a commercial system-area network, or a local-area network. A custom design is specific to a vendor and is composed of networks of switches containing internal crossbars. An example of commercial system-area network is Myrinet. Myrinet is a type of local area network for high-performance interconnection of clusters [45]. Ethernet is used as local-area network connection for networks of workstation (NOW) architectures. In a network of workstations, each node of the system is a workstation that collaborates with others via message-passing.

In high-performance systems, efficient memory access schemes are needed. Multiple levels caches are used to speed up data and instruction access. Instruction reordering and data prefetching are used to avoid latency caused by slow memories. This causes compilers

to be left with the task of generating efficient codes to take advantage of the hardware. Memory access in shared memory systems has to incorporate cache coherence mechanisms to avoid the access of non-valid data from cache.

Metacomputing is another approach to obtain high-performance systems. The idea behind metacomputing is to obtain a highly powerful distributed system composed of physically distributed computers to obtain the best resources available to jointly solve a problem. The system should be transparent to the users who will concentrate on the solution of their respective problems and not on the computational requirements of the problem [46]. There are several issues in the development of such a global computing platform, such as software compatibility, high-performance networks, security, and user-friendly interfaces. Metacomputing involves the interconnection of high-performance networks, implementing a distributed file system, coordinating user access to different computational structures, and making the environment easy-to-use and transparent to the user. There are two projects addressing the metacomputing problem: Globus and Legion.

Globus is a project of Argonne National Laboratory and the University of Southern California's Information Sciences Institute. The Globus project intends to create the infrastructure to support the interaction of computational systems physically located in geographically distributed places. The goal of Globus is to understand application requirements for a usable large-scale high-performance distributed computing environment [47]. There are three main research activities in Globus: (1) the development of basic technology and high-level tools; (2) the construction of a large-scale testbed; and (3) the execution of a real application to evaluate the previous two activities.

The main component of the Globus system is the Globus Metacomputing Toolkit. This toolkit is composed of the services needed for resource management, communication, security, information, health and status, remote data access, and executable management [47]. Future services that will be incorporated in the toolkit are accounting, auditing, and instrumentation. The initial technologies for Globus have been implemented in the Globus Ubiquitous Supercomputing Testbed (GUSTO). At the beginning of 1998, GUSTO con-

nected 17 sites in three different countries, 330 computers, and 3600 processors [47].

The Legion Project was developed at University of Virginia. Its goal is to develop a worldwide virtual very powerful computer [48]. The design goals of Legion are scalability, usability, security, site autonomy, and efficiency. Legion is an object-oriented system, that is, every component of the system is represented by an object. This is the only programming paradigm it supports. This is a main difference between Globus and Legion since Globus supports different programming paradigms.

Legion should be responsible for protecting the user, the data, and the resources from security problems. It should also manage data movement, fault tolerance, and scheduling. Research areas involved in the project are: parallel compilers, distributed systems software, object-oriented paradigms, cryptography and cryptographic protocols. In the summer of 1995 the first prototype of Legion was presented: the Campus Wide Virtual Computer (CWVC). It consists of more than 100 workstations and a 18-processor IBM SP2 supercomputer in six buildings at the University of Virginia [48]. It posed some of the challenges of the nationwide Legion project to the researchers.

2.4 Environments and Tools

We are proposing to improve integrated problem solving environments for the development and efficient implementation of signal processing algorithms. We will address the problem of raising the level of abstraction of the information provided by tools to the signal processing specialist. In this section we present some background on problem solving environments, modeling, and high-performance computing tools.

2.4.1 Problem Solving Environments

A problem solving environment (PSE) is an integrated tool to provide facilities for solving a class of problems in a convenient and efficient way [3]. It makes the computer system or architecture transparent to the user, providing features which allow to solve problems ranging from simple to complex. PSEs are designed with two main goals [2]: they should

allow more people to solve more problems more quickly, and they should enable users to solve otherwise unsolvable problems. Additionally, PSEs should, in most cases, produce better and more reliable results.

Sekhar *et al.* have developed a problem solving environment for signal processing based on a dedicated artificial intelligence tool called OCAPI [9]. This environment brings the expertise of the signal processing specialist to the signal processing user to solve an application. This PSE is designed specifically for the solution of problems with the use of current algorithms. This project does not address the design of signal processing algorithms.

Another example of a problem solving environment is MATLAB®. MATLAB has become the dominant high-level language for mathematical and scientific calculations. Six main reasons have been identified by Trefethen *et al.* for MATLAB's success: ease of use, high quality of numerical algorithms, powerful graphic capabilities, high level of abstraction, extensibility, and the use of matrices as basic data type [49-50].

Users are able to use MATLAB for rapid development and testing of algorithms. However, since MATLAB is an interpreted language, its performance is not the same as the one obtained from compiled high level languages such as C or Fortran. Several high performance versions of MATLAB have been developed to improve its performance, two of which we will describe: MultiMATLAB and RTExpress.

MultiMATLAB was developed at the Cornell Theory Center. It is an extension of MATLAB to run in a distributed memory environment. In MultiMATLAB each processor in the platform runs a MATLAB process which can communicate with another via an interface module designed using MPI. Two different platforms are supported: IBM SP2 and a network of workstations. MultiMATLAB's performance is acceptable for coarse-grain distributed computing [50], but for fine grain parallelism, its communication overhead is very high when compared to directly programming in C and MPI. So there is a trade-off between the rapid development of parallel algorithms and performance [51]. Another feature of MultiMATLAB is that it supports three different parallel programming paradigms: master/slave, single program multiple data, and distributed matrices. This implies that the user has to design

an actual parallel algorithm to solve his/her problem. RTExpress has another approach for high performance computing using MATLAB.

RTExpress^{TM} is a compiler that converts MATLAB code into C with MPI for high performance computers [52]. It was developed by Integrated Sensors, Inc. RTExpress supports the following platforms: Mercury, CPSI, IBM SP2, Paragon, and a network of Sun workstations. RTExpress provides all capabilities to compile, run, and execute MATLAB code on a target parallel computer without having to recode the application. The following paradigms are supported in RTExpress: data parallel, round robin, task parallel and pipelining, and mixed mode. These paradigms are hidden to the user unless he/she wants to specify compiler and/or run-time parameters.

2.4.2 Performance Modeling

In order to select alternatives in the development of code, the cost of each alternative should be identified and measured. Performance models are used for this purpose. Models are used to compare the efficiency of different algorithms before doing an implementation. They can also be used to show where improvements in an implementation can be made [53].

A model should not be too simple so that estimates are far from reality or too complex that it is impractical to use. An intermediate level of detail is required to be useful to the programmer. It should be as simple but as accurate as possible.

Performance models specify a metric as a function of algorithm or hardware characteristic. For sequential architectures the metrics typically used are execution time and memory utilization. Some of the metrics used for parallel programs are execution time, speedup, and efficiency.

One of the most important metrics for parallel programs is execution time [54]. Execution time is the time elapsed between the execution of the first instruction until the execution of the last instruction. Current models for message passing environments describe execution time as a combination of computation time, communication time, and idle time.

Efficiency measures the fraction of time processors spend doing useful work. It is defined

as

$$E = \frac{T_1}{PT_p}$$

where T_1 is the execution time of the best sequential algorithm, P is the number of processors, and T_p is the execution time on P processors [53]. This metric characterizes how effective is the algorithm in using the parallel computational resources.

The actual parameters to use in the performance models are determined experimentally by the use of performance tools. These experiments can also be used after programming to compare observed and modeled performance.

2.4.3 Tools

Performance Tools

A great effort has been placed on the development of tools for high-performance computing [55-56]. The purpose of these tools is to aid the application programmers to develop their applications. Application development could be a very demanding process. Tools for the development of serial code are very mature. There are a variety of debuggers and profiling tools to aid programmers in code development and performance monitoring. Tools for parallel code development are not as standard. These tools take different approaches, different data formats, and different display techniques [53].

There are three steps in performance analysis: data collection, data transformation, and data visualization. Data collection can be done through profilers, which record the amount of time spent in different parts of the program. Gprof is an example of such a tool for the Unix operating system. Data transformation and visualization will modify and present performance data to be comprehensible and useful to the user.

It is important that tools provide the appropriate visualization techniques to display information. Two basic principles for visualization of performance data are given in [57]. The first one states that the displays should be linked directly to the performance model. The second one is that visualizations should be designed and applied in an integrated

environment. Therefore the selection of the appropriate models for showing the performance to the user is extremely important.

ParaGraph, AIMS, VAMPIR, and Pablo are examples of performance tools for parallel programming [58-60]. ParaGraph, AIMS, and VAMPIR are tools for the visualization of message-passing parallel programs. Pablo supports both message-passing and data-parallel programming models.

Message-passing defines a model in which computations are local to the process. Each process has its own local memory and transfer of data requires explicit messages among processes. This is the most widely used model nowadays.

There are two prevalent libraries using the message-passing model: (Message Passing Interface) MPI and (Parallel Virtual Machine) PVM [61]. MPI has become the standard in message-passing programs.

MPI

The message passing interface (MPI) was the result of the collaboration among computer vendors, software writers, and application scientists who joined efforts from 1993 to 1994 to develop a standard portable message-passing library definition [62-63]. MPI defines a set of library routines to write message-passing programs in Fortran 77 and C. To develop the standard, researchers collected the most useful features from other systems and incorporated them into one standard. MPI was developed with two main goals: portability and performance. It runs both on tightly-coupled, massively parallel processing (MPP) machines and on networks of workstations (NOWs).

This library includes process management routines, point-to-point communication routines, and process group/collective communication routines [63]. Process management routines include those for initializing processes, terminating processes, and identifying processes. Examples of these are `MPI_INIT()`, `MPI_COMM_SIZE()` and `MPI_FINALIZE()`. Point-to-point communication routines are the basic way to send and receive messages between two processors. Tags may be employed to identify when the message should be received by a specific processor. Examples of these routines are `MPI_SEND()` and `MPI_RCV()`

Collective communication routines are those that involve more than two processes such as broadcast, scatter, gather, reduction operations, and synchronization routines [53]. `MPI_BARRIER()`, `MPI_BCAST()` and `MPI_REDUCE()` are among the collective communication routines.

CHAPTER 3

Problem Statement

The need for problem solving environments for the development and efficient implementation of new signal processing algorithms has been identified. However, meaningful, high-level performance abstractions are not available to aid in the implementation process. Current tools are general in their approach for performance evaluation and monitoring. However, this approach has left a performance gap in domain-specific computing, including signal processing.

This research will explore the solution space for the process of designing and mapping texture analysis algorithms to advanced architectures. The process is supported by a problem-solving environment, which increases productivity, and the objective is to improve the efficiency of the implementation. The solution space consists of platform configuration options at different levels in the mapping process, performance monitoring results, and algorithm alternatives. The goal is to determine correlation and provide appropriate feedback to the signal processing specialist to guide him/her when making algorithmic decisions.

The combined use of performance modeling, performance monitoring, and mapping techniques is being proposed to aid in the implementation of signal processing applications. This approach includes obtaining performance measurement, selecting the appropriate models to raise the level of abstraction of the information, and correlating information to provide feedback to the user relevant to the mapping process. The proposed research will study both processor-level and system-level configuration options and their relationship to algorithm

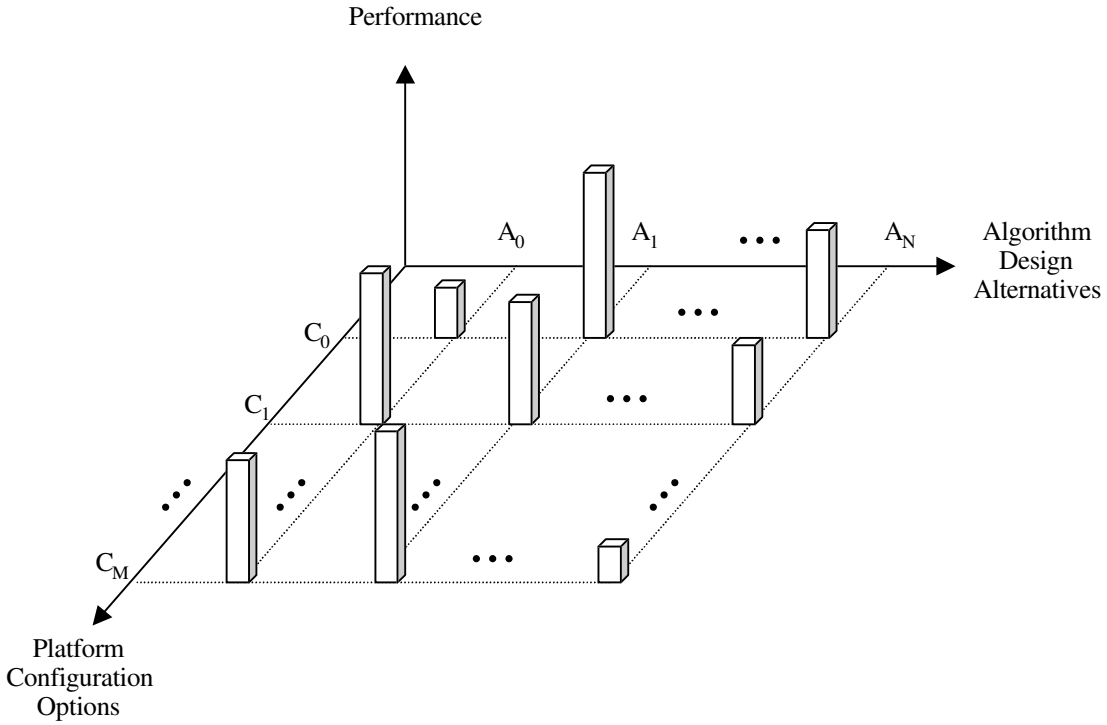

Figure 3.1. Design space coordinates. Performance will be measured for different (algorithm, configuration) ordinate pairs.

performance. Processor-level configuration options are those which affect the processor performance at run time. System-level configuration options are those affecting the response of the whole system, such as number of processors used to solve a parallel problem. Information at different levels in the mapping process will be extracted and presented to the signal processing specialist in a meaningful form. The solution space for our optimal solution is shown in Figure 3.1. Given an ordinate pair (algorithm, configuration) we will obtain a performance measurement. An appropriate model to provide insight and guidance in selecting the best alternative for the implementation will be developed.

A computational mathematical framework will be used to design experiments for obtaining correlation between algorithmic decisions, configuration options, and performance [41]. Figures 3.2 and 3.3 show the two approaches to follow for our experiments. In these figures the set of different algorithms to solve a problem is given by $\{A_0, A_1, ..., A_N\}$, and

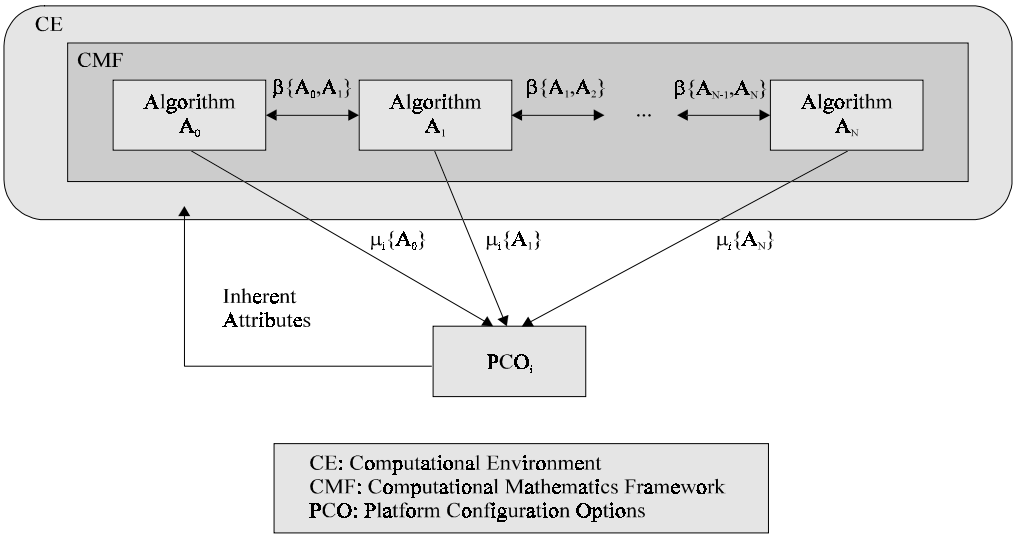

Figure 3.2. Process of studying an algorithm behavior given a fixed platform configuration.

the set of platform configuration options is denoted by $\{PCO_0, PCO_1, ..., PCO_M\}$. In figure 3.2, different algorithm variations are tested on one particular platform configuration to determine the best algorithm given a configuration. The inherent attributes of the platform are taken into consideration in a computational environment when making variations in the algorithms. In figure 3.3, a fixed algorithm is studied and different platform configurations are tested, giving information on the best platform for a selected algorithm. The results will provide the graph shown in figure 3.1.

Tools should report performance in a domain specific context. For this purpose we will be studying the wavelet-based compressed domain techniques and texture analysis for the retrieval of images in databases. This application is representative of the types of workloads that will be of major impact on computer systems in the future. Algorithms for this application will be developed and analyzed. In general, the results of this research will lead to an improvement in the efficiency achieved in the implementation of signal processing algorithms.

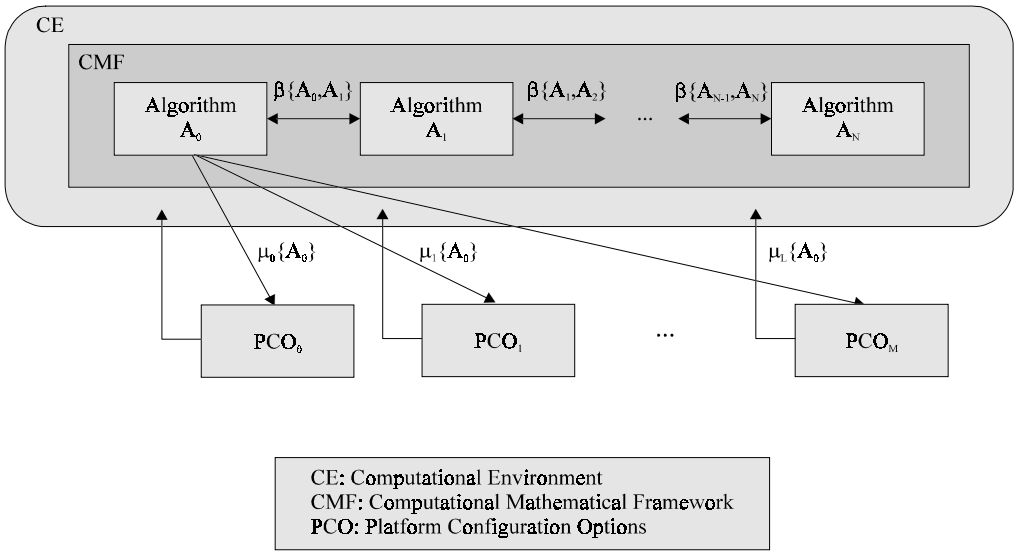

Figure 3.3. Process of studying the platform configurations for a given algorithm.

CHAPTER 4

Research Plan

The completion of this research can be decomposed into several tasks, which are presented below, followed by a timeline of activities.

4.1 Tasks

The following tasks are required to complete our research work:

- **Task 1:** *Analysis, development, and coding of algorithms for texture-based image retrieval.*

Objective: To code an application that can provide an insight into the mapping process and yield correlations between platform configuration options and representative multimedia algorithms.

Significance: This task will provide an application to verify our hypotesis that performance should be reported in a domain-specific context to be meaningful. Texture-based image retrieval is increasing in importance as automatic database search is becoming more difficult as databases grow in size. Multimedia applications are becoming a dominant workload in computer systems. The completion of this task will produce a faster, more efficient method to retrieve images from a database.

Approach: The image database will be limited to a specific problem area and a data set for learning and testing the application will be acquired. A classifier, decision rule, and error

estimation method to verify results will be chosen. Different wavelet-based texture features will be studied, extracted, analyzed, ranked and selected for image matching. A similarity measure and a search method will be selected for doing image matching. The relationship between the extraction of texture features and image compression through the use of wavelet transform will be studied to map the process of image query to the compressed domain. The application will be implemented in MATLAB, C, and MPI.

- **Task 2:** *Performance analysis of algorithm implementations on computer platforms.*

Objective: To obtain a graph of the design space (figure 3.1) for representative platforms and to determine factors affecting the performance of the selected application. To determine which visualizations are meaningful to the application.

Significance: The design space graph will provide the operating point (algorithm, configuration) where the most efficient implementation can be obtained. It will also show relationships between configurations and performance, and modifications to algorithms and performance. This information is critical in establishing relationships and providing appropriate feedback to the user. Appropriate visualizations are required to obtain knowledge about possible problems with the code.

Approach: High-performance computing platforms and problem solving environments to be used in our experiments will be selected. One of the platforms will be a high-performance workstation and another one will be a network of workstations. Different performance analysis tools will be studied and whether they can be used to raise the level of abstraction of the analysis to the signal processing specialist will be determined. A tool will be selected and/or modified to use in our experiments and obtain performance measurements. Experiments where algorithm and configuration options will be modified according to figures 3.2 and 3.3 will be performed. Both processor-level and system-level configuration options will be studied.

- **Task 3:** *Analysis of correlation of high level abstractions to low level configurations.*

Objective: To define appropriate models for describing performance in the selected platforms and to correlate performance and implementation decisions in the mapping process.

Significance: The selection of an appropriate model is important since meaningful performance measures in one platform might not be so in another one. Correlations will provide the information needed to guide the signal processing specialist in improving the performance of the implementation for a given target platform.

Approach: The model corresponding to a given target architecture will be selected. These models will be used to create a correlation between the high level description of the algorithm and the implementation performance results. Studies will be conducted to correlate high-level performance factors to performance.

- **Task 4:** *Development of a representation to aid the signal processing specialist in decision making.*

Objective: Develop an appropriate representation of the information to give the user the capability to make decisions about his/her algorithms to improve the implementation performance.

Significance: The signal processing specialist will be able to understand the impact of the decisions made at a high level. This will result in more efficient implementations while retaining the benefits of problem solving environments.

Approach: Apply different visualization techniques to present the information to the signal processing specialist. Recommend a user interface for providing feedback at different points in the mapping process.

4.2 Timeline

The following is the timeline of the expected progress in this research:

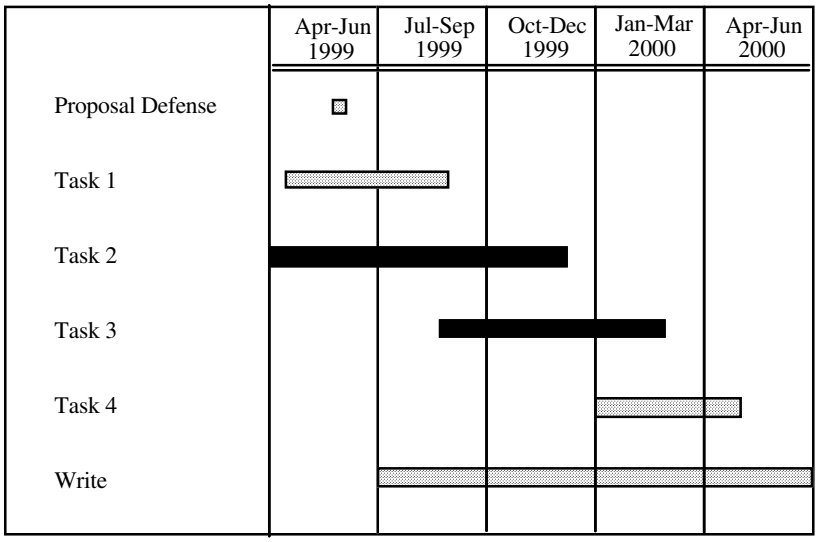

Figure 4.1. Timeline for the completion of the Ph.D. program.

4.3 Work in Progress

Four different tasks of this research have been worked on.

First, different platforms have been studied and three have been already selected. A network of Sparc workstations, an IBM SP2, and a high-performance Pentium-based workstation are planned to be used as target architectures. The network of workstations and the SP2 will serve as parallel processing target architectures while the Pentium-based machine will be used for serial code.

Second, for parallel implementations, the Message Passing Interface (MPI) has been selected as communication library. Both Parallel Virtual Machine (PVM) and MPI were studied to determine the most suitable library for our purposes. MPI was selected because of its higher performance when compared to PVM and its tendency to be pervasive in the

research community.

Third, three different problem solving environments have been considered: DADiSP, MultiMATLAB and RTExpress. RTExpress has been selected as target PSE for several reasons. RTExpress is based on MATLAB which is one of the most widely used environment for signal processing algorithm development [49]. This makes RTExpress appealing to the signal processing community. Previous experience with MultiMATLAB on the IBM SP2 indicate that the performance of this tool is bounded by the performance of MATLAB processes running on each node [51]. RTExpress converts MATLAB code into C with MPI calls making it more suitable for performance improvements.

Fourth, different performance tools have been under evaluation. The following tools have been studied: Paradyn, AIMS, Nupshot, Pablo, VAMPIR, and VTUNE. In the ongoing evaluation process Paradyn, AIMS, and Nupshot have been discarded. Paradyn is used for PVM programs and MPI has been selected as the communication library. VAMPIR and Pablo were selected over AIMS and Nupshot because of their flexibility in presenting performance information at different levels of abstraction. In the current stage of evaluation VTUNE, VAMPIR, and Pablo seem to be the most suitable tools for performance evaluation and analysis.

CHAPTER 5

Anticipated Contributions

Upon the completion of this research, the following contributions to both the computing and the signal processing fields are expected:

1. An improved image retrieval mechanism for compressed databases will be provided.
2. A meaningful way of presenting performance information to the signal processing specialist will be developed.
3. Correlations between high level descriptions and implementation performance will be unveiled. This will produce a base knowledge useful in building software systems.
4. Information will be provided to the signal processing specialist giving him/her insight into his/her algorithmic decisions while retaining the benefit of working on a problem solving environment.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Parthasarathy Ranganathan, Sarita Adve, and Norman P. Jouppi. Performance of image and video processing with general-purpose processors and media ISA extensions. In *To appear in the Proceedings of the 26th International Symposium on Computer Architectures*, May 1999.

[2] Efstratios Gallopoulos, Elias Houstis, and John R. Rice. Computer as thinker/doer: Problem solving environments for computational science. *Computational Science and Engineering*, 1(2):11 – 23, Summer 1994.

[3] John R. Rice and Ronald F. Boisvert. From scientific software libraries to problem-solving environments. *IEEE Computational Science and Engineering*, 3(3):44 – 53, Fall 1996.

[4] Shahani Markus, Sanjiva Weerawarana, Elias N. Houstis, and John R. Rice. Scientific computing via the web: The net pellpack pse server. *IEEE Computational Science and Engineering*, 4(3):43 – 51, July-September 1997.

[5] Robert L. Akers, Elaine Kant, Curtis J. Randall, Stanly Steinberg, and Robert L. Young. SciNapse: a problem-solving environment for partial differential equations. *IEEE Computational Science and Engineering*, 4(3):32 – 42, July-September 1997.

[6] Robert Van Engelen, Lex Wolters, and Gerard Cats. Tomorrow's weather forecast: Automatic code generation for atmospheric modeling. *IEEE Computational Science and Engineering*, 4(3):22 – 31, July-September 1997.

[7] B. Ford and R. M. J. Iles. The what and why of problem solving environments for scientific computing. *Problem Solving Environments for Scientific Computing*, pages 3 – 18, 1987.

[8] Stuart Feldman. The how and which of problem solving environments. In B. Ford and F. Chatelin, editors, *Problem Solving Environments for Scientific Computing*, pages 23 – 30. Elsevier Science Publishing Company, Inc., 1987.

[9] Chandra Shekhar, Sabine Moisan, and Monique Thonnat. Towards an intelligent problem-solving environment for signal processing. *Mathematics and Computers in simulation*, 36:347 – 359, October 1994.

[10] S. Bose, B. Friedlander, and A. Zeira. MAT2DSP - a MATLAB tool for rapid feedback on the implementation requirements of signal processing algorithms. In *Proceedings of the 40th Midwest Symposium on Circuits and Systems*, pages 845 – 848, August 1997.

[11] Jose Luis Pino, Soonhoi Ha, Edward A. Lee, and Joseph T. Buck. Software synthesis for DSP using Ptolemy. *Journal of VLSI Signal Processing*, 9(1):7 – 21, January 1995.

[12] Bo Shen and Iswar K. Sethi. Inner-block operations on compressed images. In *Proceedings of the ACM International Conference Multimedia '95*, pages 490 – 499, 1995.

[13] Shih-Fu Chang. New algorithms for processing images in the transform-compressed domain. In *Proceedings of SPIE Symposium on Visual Communications and Image Processing*, pages 445 – 453, May 1995.

[14] Shih-Fu Chang. Compressed-domain techniques for image/video indexing and manipulation. In *Proceedings of the 1995 IEEE International Conference on Image Processing*, volume 1, pages 314 – 317, 1995.

[15] Brian C. Smith and Lawrence A. Rowe. Algorithms for manipulating compressed images. *IEEE Computer Graphics and Applications*, 13(5):34 – 42, September 1993.

[16] Boon-Lock Yeo. *Efficient Processing of compressed images and video.* PhD thesis, Princeton University, January 1996.

[17] Shih-Fu Chang and John R. Smith. Extracting multi-dimensional signal features for content-based visual query. In *Proceedings of the SPIE, Visual Communications and Image Processing '95*, volume 2501, pages 995 – 1006, 1995.

[18] Jerome M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445 – 3462, December 1993.

[19] Robert W. Buccigrossi and Eero P. Simoncelli. Progressive wavelet image coding based on a conditional probability model. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 2957 – 2960, 1997.

[20] Khanh Nguyen-Phi and Hans Weinrichter. DWT image compression using contextual bitplane coding of wavelet coefficients. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 2969 – 2972, 1997.

[21] Olivier Rioul and Martin Vetterli. Wavelets and signal processing. *IEEE Signal Processing Magazine*, 8(4):14 – 38, October 1991.

[22] C. Sidney Burrus, Ramesh A. Gopinath, and Haitao Guo. *Introduction to Wavelets and Wavelet Transforms: A Primer.* Prentice Hall, 1998.

[23] P. Scheunders, S. Livens, G. Van de Wouwer, P. Vautrot, and D. Van Dyck. Wavelet-based texture analysis. *International Journal Computer Science and Information Management*, December 1997.

[24] Hyeokho Choi and Richard Baraniuk. Multiscale texture segmentation using wavelet-domain hidden markov models. In *Proceedings of the 1998 32nd Asilomar Conference on Signals, Systems & Computers*, November 1998.

[25] Robert Porter and Nishan Canagarajah. Robust rotation invariant texture classification. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 3157 – 3160, 1997.

[26] E. Salari and Z. Ling. Texture segmentation using hierarchical wavelet decomposition. *Pattern Recognition*, 28(12):1819 – 1824, December 1995.

[27] Olaf Pichler, Andreas Teuner, and Bedrich J. Hosticka. A comparison of texture feature extraction using adaptive gabor filtering, pyramidal and tree stuctured wavelet transforms. *Pattern Recognition*, 29(5):733 – 742, May 1996.

[28] G. Van de Wouwer, P. Scheunders, and D. Van Dyck. Statistical texture characterization from discrete wavelet representations. *IEEE Transactions on Image Processing*, 8(4):592 – 598, April 1999.

[29] Gregory K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):30 – 44, April 1991.

[30] Bo Shen and Ishwar K. Sethi. Direct feature extraction from compressed images. In *Proceedings of the SPIE Storage & Retrieval for Image and Video Databases IV*, volume 2670, 1996.

[31] Amara Graps. An introduction to wavelets. *IEEE Computational Science & Engineering*, 2(2):50 – 61, Summer 1995.

[32] Martin Vetterli and Jelena Kovačević. *Wavelets and Subband Coding*. Prentice Hall PTR, 1995.

[33] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. Wavelet for computer graphics: A primer, part 1. *IEEE Computer Graphics and Applications*, 15(3):76 – 84, May 1995.

[34] Kieron Messer and Josef Kittler. Using feature selection to aid an iconic search through an image database. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 2605 – 2608, 1997.

[35] John R. Smith and Shih-Fu Chang. Quad-tree segmentation for texture-based image query. In *Proceedings of ACM Multimedia '94*, pages 279 –286, October 1994.

[36] Morton Nadler and Eric P. Smith. *Pattern Recognition Engineering.* John Wiley & Sons, Inc., 1993.

[37] Thomas L. Naps. *Introduction to Data Structures and Algorithm Analysis.* West Publishing Company, second edition, 1992.

[38] Sara Baase. *Computer Algorithms: Introduction to Design and Analysis.* Addison Wesley Publishing Company, 1978.

[39] Mladen Victor Wickerhauser. *Adapted Wavelet Analysis from Theory to Software.* A K Peters, Ltd., 1994.

[40] Kevin Dowd. *High Performance Computing.* O'Reilly & Associates, Inc., 1993.

[41] Domingo Rodríguez, Nayda G. Santiago, and Carlos Vélez. Implementation of a new class of FFT algorithms on transputer computational structures. In *Proceedings of the 36th Midwest Symposium on Circuits and Systems*, volume 2, pages 1105 – 1108, 1993.

[42] Paul Messina, David Culler, Wayne Pfeiffer, William Martin, J. Tinsley Oden, and Gary Smith. Architecture. *Communications of the ACM*, 41(11):36 – 44, November 1998.

[43] M. Rafiquzzaman. *Fundamentals of Digital Logic and Microcomputer Design.* Rafi Systems, Inc., 1999.

[44] Alex Peleg, Sam Wilkie, and Uri Weiser. Intel MMX for multimedia PCs. *Communications of the ACM*, 40(1):25 – 38, January 1997.

[45] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet – a gigabit per-second local-area network. *IEEE Micro*, 15(2):29 – 36, February 1995.

[46] Larry Smarr and Charles E. Catlett. Metacomputing. *Communications of the ACM*, 35(6):44 – 52, June 1992.

[47] Ian Foster and Carl Kesselman. The Globus project: A status report. In *Proceedings of the Joint International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing IPPS/SPDP '98 Heterogeneous Computing Workshop*, pages 4 – 18, 1998.

[48] Andrew S. Grimshaw, Wm. A. Wulf, and the Legion Team. The legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39 – 45, January 1997.

[49] Anne E. Trefethen, Vijay S. Menon, Chi-Chao Chang, Grzegorz J. Czajkowski, Chris Myers, and Lloyd N. Trefethen. MultiMATLAB: MATLAB on multiple processors. Technical Report TC96TR239, Cornell Theory Center, May 1996.

[50] Vijay Menon and Anne E. Trefethen. MultiMATLAB: Integrating MATLAB with high-performance parallel computing. In *Proceedings of Supercomputing '97*, 1997.

[51] Nayda G. Santiago. SAR images speckle reduction for classification using wavelets on MultiMATLAB. Technical Report CTC97TR269, Cornell Theory Center, March 1997.

[52] RTExpress User's guide.

[53] Ian Foster. *Designing and Building Parallel Programs.* Addison-Wesley Publishing Company, Inc., 1995.

[54] Alan H. Karp and Horace P. Flatt. Measuring parallel processor performance. *Communications of the ACM*, 33(5):539 – 543, May 1990.

[55] Jeffrey Brown, Al Geist, Cherri Pancake, and Diane Rover. Software tools for developing parallel applications. part 1: Code development and debugging. In *SIAM Conference on Parallel Processing for Scientific Computing Proceedings*, March 1997.

[56] Jeffrey Brown, Al Geist, Cherri Pancake, and Diane Rover. Software tools for developing parallel applications. part 2: Interactive control and performance tuning. In *SIAM Conference on Parallel Processing for Scientific Computing Proceedings*, March 1997.

[57] Michael T. Heath, Allen D. Malony, and Diane T. Rover. The visual display of parallel performance data. *Computer*, 28(11):21 – 28, November 1995.

[58] Michael T. Heath and Jennifer A. Etheridge. Visualizing the performance of parallel programs. *IEEE Software*, pages 29 – 39, September 1991.

[59] Jerry C. Yan and Sekhar R. Sarukkai. Analyzing parallel program performance using normalized performance indices and trace transformation techniques. *Parallel Computing*, 22(9):1215 – 1237, November 1996.

[60] Daniel A. Reed, Ruth A. Aydt, Tara M. Madhyastha, Roger J. Noe, Keith A. Shields, and Bradley W. Schwartz. *An Overview of the Pablo Performance Analysis Environment.* Department of Computer Science, University of Illinois - Urbana, November 1992. Pablo Documentation.

[61] Al Geist, Adam Benguelin, Jack Dongarra, Weicheng Jiang, Robert Mancheck, and Vaidy Sunderam. *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing.* Scientific and Engineering Computation. The MIT Press, 1994.

[62] Jack J. Dongarra, Steve W. Otto, Marc Snir, and David Walker. A message passing standard for mpp and workstations. *Communications of the ACM*, 39(7):84 – 90, July 1996.

[63] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI. Portable Parallel Programming with the Message-Passing Interface.* Scientific and Engineering Computation. The MIT Press, 1994.

Biosketch. Nayda G. Santiago was born in Puerto Rico in 1966. She received the B. S. degree in Electrical Engineering from the University of Puerto Rico, Mayaguez, PR in 1989, and the M. Eng. degree in Electrical Engineering from Cornell University, Ithaca, NY in 1990.

She worked as instructor at the University of Puerto Rico from 1990 to 1994. Her primary research interests include computer performance and signal processing. She is member of the IEEE, ASEE, ACM, SWE, Tau Beta Pi, and Phi Kappa Phi.