they access data. Application models define a data–flow consisting of function models and their interconnections.

The *hardware aspect* contains models describing the C40 network, including information about each node (how much memory, node type) and about the interconnection topology.

The *goals aspect* allows specification of the computational performance goals to the model interpreter, which it uses in determining the type and level of parallelism which will be used. The goals models contain (1) a *throughput goal*, (2) a *latency goal*, and (3) the *failure mode*.

5 Summary

The current prototype system is capable of performing sequences of local image processing algorithms on 512x480, $8\frac{bits}{pixel}$ (256 grey levels) frames at video rate (30 frames per second), given that enough C40s are available in the network. A prototype MIRTIS system has been bench–marked at $520\frac{Mops}{sec}$ sustained (counting only useful computations) while performing a complex edge detection on live video. The system has also been shown useful for simple applications.

Future work will extend the capabilities of the system to the more general domains of medium and high level vision. The long–term goal is to build a unified architecture and programming environment for image processing and computer vision.

References

[1] Webb, John A., "High Performance Computing in Image Processing and Computer Vision", *Proceedings of the International Conference on Pattern Recognition*, October 1994, Jerusalem.

[2] Webb, John A., "Steps Toward Architecture–Independent Image Processing," *IEEE Computer*, February 1992: p. 21–31.

[3] M. S. Moore: "A DSP–Based Real–Time Image Processing System," *International Conference on Signal Processing Applications & Technology (ICSPAT '95)*, Boston, October 1995.

[4] M. S. Moore, G. Karsai, and J. Sztipanovits: "Model–Based Programming for Parallel Image Processing", *Proceedings of the First IEEE International Conference on Image Processing (ICIP94)*, Nov. 1994.

[5] G. Karsai: "A Configurable Visual Programming Environment", *IEEE Computer*, March 1995.

[6] B. Abbott, T. Bapty, C. Biegl, G. Karsai, and J. Sztipanovits: "Model–Based Software Synthesis", *IEEE Software*, May 1993.

appropriately overlapping pieces and communicates the pieces to the processing nodes. The image processing program running on each node receives its local input and output buffers as function parameters and performs the computation. During the computation, the PCT system splits the next input image and again distributes it across the network. After the computation has finished, the PCT system simultaneously merges the partial results and restarts the computation. Since the communication is concurrent to the computation, each node is very nearly always computing. For more details about PCT, see [3].

3.3 MIRTIS Architecture

We have used the MultiGraph Architecture (MGA), a MIPS architecture developed at Vanderbilt, to create an environment which automatically generates the complex PCT programs. An image processing modeling paradigm has been developed which contains the image processing concepts necessary to automatically build PCT split–and–merge data–flows. Based on graphical system models, a model interpreter automatically generates the necessary PCT communications programs and computation schedules to implement the data–parallel computational data–flow. It then maps this data–flow to the parallel hardware architecture. The details of the parallel implementation are completely hidden from the user.

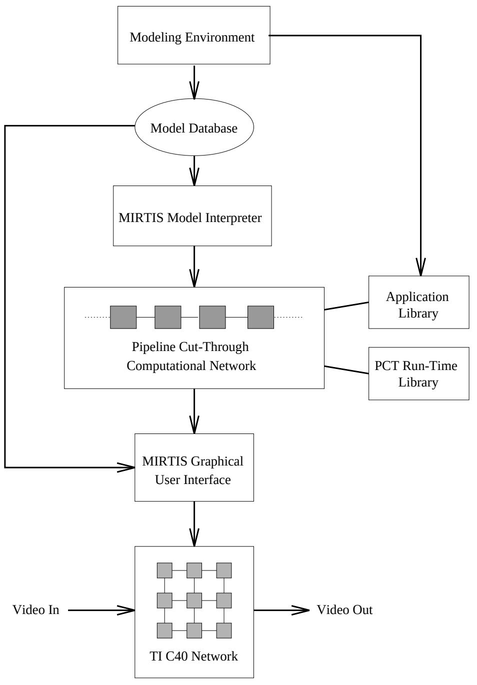

Figure 1: The MIRTIS Architecture

The MIRTIS architecture follows the basic model–based system architecture shown in [6]. It consists of a *modeling environment*, a *model database*, a *model interpreter*, *application* and *run–time libraries*, a *graphical user interface*, and a *C40 network* (See Figure 1).

The Modeling Environment

The modeling environment facilitates the creation of system models. A *graphical model editor* is used for building, manipulating, and validating the system models. The models represent the system in terms of the MIRTIS modeling paradigm, and are stored in a *model database.*

The Model Interpreter

The MIRTIS model interpreter, called *MINT*, interfaces with the graphical model builder and automatically generates a PCT computational network which realizes the computations described in the models. During the interpretation process, the interpreter analyzes the graphical system models, partitions the computation, determines the appropriate numbers of processors to be used for each of the sub–computations, parallelizes the sub–computations, maps the parallelized sub–computations to the hardware network, and produces a pipeline cut–through computation network.

The Application Library

The MIRTIS image processing application library contains the actual code which implements the image processing algorithms. Library routines are written in C, and are compiled with the standard cl30 compiler released by TI. The user can add functionality to the application library by providing C coded subroutines. Since there is no reference to parallelism in the code, writing these routines requires no knowledge of parallel programming. All parallel facilities are provided through calls to the PCT run–time support library, which performs communication, scheduling, synchronization, memory management, etc.

The Graphical User Interface

The *graphical user interface*, called MUI, is used to control the operation of the model interpreter, load the C40 network, and monitor/interact with the running system. MUI allows algorithmic parameters to be adjusted dynamically, so the user can experiment with and "tweak" algorithms easily.

4 The MIRTIS Modeling Paradigm

The MIRTIS modeling paradigm was designed specifically for parallel image processing. It includes three modeling views, or *aspects*: *Computation*, *Hardware*, and *Goals*.

The *computation aspect* describes which algorithms are to be performed, and in what order. Computation models are represented visually in a graph data–flow style: blocks representing computations and inter–connecting lines representing the flow of data. There are two types of computation aspect models, *library function models*, and *application models*.

Function models describe the individual algorithms available in the image processing application library. They contain pertinent information about the library functions, such as performance behavior and the way

- Usually the algorithms perform fairly simple computations to produce each output pixel.
- Images are large data sets which lend themselves easily to data parallelism.
- Processing sequences of images with these algorithms requires high computational power, and many applications (especially the real–time applications) need the speedups of parallelization.

Split–and–Merge Data Parallelism

Because of the properties mentioned above, image processing algorithms are easily data parallelizable. A simple data parallel programming technique which is applicable to image processing is the *split–and–merge model* [3],[4],[1]. In [2], Webb proves that a very general class of image processing algorithms can be performed using this technique. Since the split–and–merge technique is simple, it is possible to create accurate performance models for these computations, which are necessary for determining how many processors are needed to meet the real–time constraints of a particular application.

2.2 Real–Time Image Processing

A *Real–Time* system is one that, due to interaction with its environment, must produce outputs which are not only numerically correct, but also meet timing constraints. The addition of timing constraints to an image processing system adds complexity and raises several key issues which must be addressed.

Hardware Architectural Issues

Image processing is computationally intensive due to the large size of image data. Table 1 shows the data rates necessary for real–time video processing. The last column shows the computation rate required to perform a $5x5$ convolution at each corresponding data rate. Note that even for moderate digitization resolutions, such as $512x480$, the computational data rate required by this simple filtering operation ($370\frac{Mops}{sec}$) far exceeds the capabilities of traditional computer hardware architectures.

The hardware architecture must be capable not only of performing the computations in real–time, but also of supporting the communications bandwidths necessary for real–time image processing.

resolution colsxrows	depth $\frac{bits}{pixel}$	frame rate $\frac{frames}{sec}$	data rate $\frac{Mbytes}{sec}$	$5x5$conv $\frac{Mops}{sec}$
$512x480$	8	30	7.0	370
$640x480$	8	30	8.8	460
$512x480$	16	30	14	370
$640x480$	16	30	18	460

Table 1: Real–Time Video Throughput Requirements

The system must also include special hardware to interface with either analog or digital video sources. These types of interfaces do not often exist on modern commercially available parallel computers [1].

In some applications, the system latency (the time between sensing a visual event and outputting the results from that event) can be critical. The hardware architecture must support low latency both in the communications and computations.

We contend that the best solution which addresses these issues while retaining the required flexibility is to use a scalable, parallel hardware architecture.

Software Architectural Issues

It is a well known fact that the largest obstacle to generating real world parallel applications is not building the hardware, but providing programming environments which will allow programmers to utilize it easily and efficiently. The complexities of parallel systems, such as task decomposition, sub–task allocation, scheduling, inter–task communication, and synchronization, can easily overwhelm even an experienced programmer.

The main issue to be faced in developing a parallel imaging environment is how to deal with these complexities in a way that takes advantage of the properties of image processing algorithms. A suitable software environment is needed which provides the user with (1) high level methods of specifying the computations and real–time constraints, and (2) a programming paradigm with transparent access to the parallel facilities.

Real–Time Issues

The relevant real–time constraints for image processing are *throughput* and *latency*. In order to meet throughput and latency requirements, information is needed about the performance properties of the computations relative to the hardware architecture, the network topology, and the behavior of the communication network. Mathematical models of the system performance (throughput and latency) must be formulated in order to determine (1) the granularity of parallelism required, and (2) the mapping of the computations to the hardware necessary to meet the constraints. These models must be accurate in order to guarantee that a particular computation can or cannot be done in real–time on a particular hardware configuration.

3 MIRTIS Overview

3.1 Hardware Architecture

The current trends in parallel hardware technology have produced parallel processing building blocks (DSPs, etc.) powerful enough for real–time image processing. The MIRTIS hardware platform [3] is a network of Texas Instruments TMS320C40 DSPs (C40s). The input node is a C40 digitizer, and the output node is a C40 display module.

3.2 Pipeline Cut–Through

PCT is a communications technique implemented on the C40 which implements the split–and–merge processing model. PCT automates the splitting and merging processes and makes the data parallelism transparent to the programmer. It splits the input image data into

Model-Based Synthesis of a Real–Time Image Processing System[*]

Michael S. Moore
Department of ECE
Vanderbilt University
Nashville, Tn. 37235

Jim Nichols
Sverdrup Technology
MS 9013, Bld. 1099
Arnold AFB, Tn. 37389

Abstract

MIRTIS is an environment which employs model–based synthesis techniques to generate real–time image processing applications. The system is capable of creating very high performance implementations of a large class of image processing computations. It automatically data parallelizes the computations using the split–and–merge processing model and executes them on a parallel hardware architecture, a network of C40 DSPs. MIRTIS provides high level programming interface which masks the complexities of the underlying parallel implementation. Graphical tools are used for building models and controlling the running applications.

1 Introduction

Non–dedicated image processing applications users usually have to trade off algorithm implementation flexibility for real–time performance. Most existing off–the–shelf real–time systems use specialized hardware architectures to perform specific algorithms in real–time (eg. convolution). A draw–back of specialized hardware is that it cannot always be re–programmed with new or non–standard algorithms. Some imaging facilities, such as the one at Arnold Engineering Development Center (AEDC), need a system which can be rapidly programmed, configured, and scaled to solve a wide variety of problems. They require:

- **Scalable Real–Time Performance**: The system must be easily scaled up or down for a particular application.
- **Programmability**: Engineers must be able to rapid prototype experimental/custom algorithms and create real–time computational data–flows consisting of both these and standard algorithms.
- **Flexibility**: It must possible to rapidly reconfigure the system software and hardware to meet the application requirements.
- **Parallel Programming Interface**: The programmer must be able to write parallel programs without dealing with parallel issues. The programming style must be independent of the underlying parallel hardware architecture.
- **High Level Tools**: The interfaces to the system must be easy to use.

This paper presents MIRTIS (Model–Integrated Real–Time Imaging System), which has been developed through a joint effort between Vanderbilt University and AEDC. MIRTIS uses Model Integrated Program Synthesis (MIPS) techniques to automatically "synthesize" data parallel image processing applications which are executed on a network of Texas Instruments TMS320C40 DSPs (C40s). Real–Time execution is achieved by scaling the data parallelism to the appropriate level to reach the performance specifications. The scaling is done with virtually no overhead via a communications concept called PCT (Pipeline Cut–Through). Since the software is automatically parallelized the user is insulated from the complexities of the parallel implementation. MIRTIS provides a high level modeling interface, which is used to manage the hardware and software configurations.

2 Background

2.1 Problem Domain

Throughout this paper, the term *image processing* includes the image to image transformations which constitute low–level vision. Examples of image processing operations are contrast enhancement (eg. histogram stretch, histogram equalization, linear rescaling), filtering (eg. smoothing, median filter), edge detection and segmentation (eg. sobel, laplacian, Laplacian of Gaussian (LOG), thresholding), local histogram operations, and morphological operations.

When processing image sequences it is often useful to perform calculations on consecutive frames. For example, time domain averaging (averaging several frames together) is used for noise reduction, and instantaneous differencing is a simple method of motion detection. Some techniques operate both in the 2–D image plane and in the time domain. Examples are 3–D filtering and 3–D morphology. These types of algorithms are also included in our definition of image processing.

Image processing has been the most common area for the application of high performance parallel computing [1]. The operations have several characteristics which make them particularly suitable for implementation on parallel computers [2].

- They are regular. Usually the same computation is performed for each pixel in the image.

[*]This work was supported by the AFOSR/AFMC, United States Air Force, contract number F49620-94-C-0076.