Architectures and Algorithms for Digital Image Processing II

Francis J. Corbett
Chairman/Editor

Architectures and Algorithms for Digital Image Processing II

Proceedings of SPIE—The International Society for Optical Engineering

Volume 534

Architectures and Algorithms for Digital Image Processing II

Francis J. Corbett
Chairman/Editor

January 24-25, 1985
Los Angeles, California

Published by
SPIE—The International Society for Optical Engineering
P.O. Box 10, Bellingham, Washington 98227-0010 USA
Telephone 206/676-3290 (Pacific Time) • Telex 46-7053

The papers appearing in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and are published as presented and without change, in the interests of timely dissemination. Their inclusion in this publication does not necessarily constitute endorsement by the editors or by SPIE.

Please use the following format to cite material from this book:
Author(s), "Title of Paper," *Architectures and Algorithms for Digital Image Processing II,* Francis J. Corbett, Editor, Proc. SPIE 534, page numbers (1985).

Library of Congress Catalog Card No. 85-050426
ISBN 0-89252-569-X

Printed in the United States of America.

ARCHITECTURES AND ALGORITHMS FOR DIGITAL IMAGE PROCESSING II

Volume 534

Conference Committee

Chairman
Francis J. Corbett
Imaging Technology, Incorporated

Co-Chairmen
Bob R. Hunt
Science Applications, Incorporated

James L. Mannos
Calma Company

David McCubbrey
Environmental Research Institute of Michigan

Session Chairmen
Session 1—Architectures and Pattern Recognition Issues
David McCubbrey, Environmental Research Institute of Michigan

Session 2—Image Processing System Architectures: Microcomputers to Super-Minis
James L. Mannos, Calma Company

Session 3—Algorithms and Image Transforms
Bob R. Hunt, Science Applications, Incorporated

Session 4—Algorithms and System Applications
Bob R. Hunt, Science Applications, Incorporated

Session 5—Algorithms and Pattern Recognition Issues
Francis J. Corbett, Imaging Technology, Incorporated

ARCHITECTURES AND ALGORITHMS FOR DIGITAL IMAGE PROCESSING II

Volume 534

INTRODUCTION

This conference was remarkably successful and its success is due to all of the authors and session chairmen who worked together to bring it about. Thank you for this exceptional effort. We all shared in the workload, and as a result, SPIE produced a true state-of-the-art technical meeting and publication. This book is the result of those efforts and it contains an accurate representation of our technology as it exists today. The quality of papers is excellent. They contain significant new material in the form of ideas, implementations, and applications of the algorithms and architectures of digital image processing. By comparing this work with that of the previous conference in 1983, one can see the dramatic progress we have made. From these strides, we can excitedly project into the future and predict mature image processing systems which will be used for cognitive analysis and produce independent decisions.

Trends which have emerged during the past year in the image processing field and which were illustrated by the papers presented at this conference are:

- a continuing shift in emphasis from reconstructive algorithmic processes to feature extraction, recognition, and identification of images.
- an integrated approach to automated image processing, which is perhaps best illustrated by the morphological and adaptive techniques and architectures shown during this conference.
- more and more applications which fit the workstation approach and which can be accommodated with micro-based distributed processors which may perform segments of a job or a complete, very specialized operation.

The field of digital image processing has grown in a few short years from a rather small band of government sponsored R&D-oriented tasks to a robust technology which is finding its way into every imaginable type of work. This growth should have been easy to predict from experience, as we chart the natural human drive to improve and control our destiny and free ourselves for more lofty considerations. It is very exciting to be a part of the technical innovations which have and will make all this possible. I expect that future meetings of those who are advancing the state-of-the-art of the algorithms and architectures of digital imaging will be more exciting and contain significant breakthroughs leading to complete and cost-efficient solutions to our multiple image processing opportunities.

Francis J. Corbett
Imaging Technology Incorporated

ARCHITECTURES AND ALGORITHMS FOR DIGITAL IMAGE PROCESSING II

Volume 534

Contents

ARCHITECTURES AND ALGORITHMS FOR DIGITAL IMAGE PROCESSING II

Volume 534

Session 1

Architectures and Pattern Recognition Issues

Chairman
David McCubbrey
Environmental Research Institute of Michigan

The time-shared bus: a viable way to image multiprocessing

L. Van Eycken (°), P. Wambacq (°), A. Oosterlinck

E.S.A.T.-C.M.E., Catholic University of Leuven
de Croylaan 52, B-3030 Heverlee, Belgium

Abstract

Out of all possible multiprocessor interconnection schemes, the time-shared bus has some advantages for hardware realisations. Not only is it one of the simpliest and cheapest ways to tie processors together, but it is also an ideal interconnection scheme if one wants to keep the structure flexible and modular. On the other hand, the main disadvantage of the time-shared bus is the limited bandwidth. Especially in image processing, this can be very troublesome. This paper will try to explore the possibilities of a time-shared bus in this field of application. A process is divided into a set of processors, each with a specified number of inputs and outputs. Furthermore, each processor is determined by a set of delays between these inputs and outputs. The model is characterised by four parameters:

- the delays per processor
- the constancy of the delays
- the use or no use of internal memory in a processor
- the fact whether the operations on a processor are pipelined or not.

These parameters influence the complexity and the effectiveness of the hardware. Using them to classify different hardware approaches, we develop a hardware definition of a time-shared bus, that optimises the use of that bus in order to diminish the disadvantage of the limited bandwidth. An example of a process, constructed by putting processors in pipeline and/or in parallel, illustrate the possibilities.

Introduction

For the last few years, multiprocessing has become a fashionable word. Image processing is a very interesting domain for applying multiprocessing, due to the nature of the image processing itself. It is characterised by a very high throughput of data, usually ranging from 3 to 50 Mpixels/sec with a wordlength of 1 to 32 bit integer or floating point. However, the computations are highly identical for all of the pixels at a low level of image processing, so this will translate itself very well to parallel computation. At higher levels this is much less trivial, but at that moment, the throughput rate is usually reduced [1].

Designs are usually made starting from a specific case. This has led to an immense variety of image processing architectures. Each one is fit very well for the group of applications, it was developed for, but on the other hand most architectures are not flexible enough to adapt to other problems or to new developments in the field of IC's. After a short overview of existing architectures, we will put forward our design goals and derive an architecture which fits these goals.

Approaches to image processing architectures.

The existing image processing architectures, as discussed in this paper, are considered to be systems. A lot of very interesting chips are in development at this moment, but this is not the level we are talking about here. Although these developments have a certain impact on the implementation of architectures, the latter should be independent of the low level design as good as possible. One could compare it with the independence between the design of a computer and the design of a communication network.

First of all, we have the whole range of standard (mini)computer systems. These systems usually have no special hardware for image processing, except for some input or display hardware. Because the whole processing is purely software, they possess a large degree of flexibility. This makes them very well suited for research purposes, but far less interesting for most of the realtime applications. For heavy computational tasks, as they exist in low level image processing, one often resorts to additional array-processors [2]. They help to speed up the computations, but they are seldom cost effective in actual applications.

Secondly, we have the highly integrated processor arrays. The use of a multitude of identical processors pays off, especially for the low level image processing. The large number of processors compensates for their simplicity, but of course we may expect more complex processors in the future.

(°): Research assistant granted by the National Fund for Scientific Research (Belgium).

The interconnection of the processors is usually a two-dimensional nearest neighbourhood connection, because this reflects the two-dimensionality of the data. Some examples are the CLIP IV [3] and MPP [4]. Because the parallelism comes from a very rigid hardware structure, it is only worthwhile to use the current processor arrays in some applications, which actually need this huge computing power. It excludes them from most of todays simple applications, because they are too powerful.

A third (and probably largest) category of image processors, consists of systems, that are build from a set of dedicated hardware processors. The big advantage of these processors is that they can be perfectly suited to a particular image processing task, with a minimum hardware cost. Typical examples are industrial and medical visual inspection systems [5]. One can determine which parts of the task are time critical and then adapt the possibilities of the processor to these requirements. The hardware processors, if more than one is used, are directly connected for the simple configurations. The form of the interconnection network depends completely on the data flow in the processing task. On the other hand, a disadvantage is certainly the fact that for each different task, the processor hardware has to be redesigned, which is a costly and time consuming process. Some manufacturers try to overcome this limitation by keeping the dedicated hardware as general as possible, e.g. by means of microprogramming without losing too much of the advantage of simplicity. An example of the latter is the VICOM system [6]. In these more flexible systems, different processors are usually connected by a bus.

Design goals

As pointed out in the previous chapter, a multitude of various architectures are popular in the field of image processing. The choice depends mostly on the kind of problem, one has to solve. This may seem to be very feasible for an actual application, but it automatically limits the use to that application or to a similar one. This problem is especially acute for laboratorium or development systems. In this case, one should like to have an image processor which is flexible enough to be reconfigured depending on the particular application one wants to study.

Reconfigurability means that the functionality and performance can be changed without a modification of the system design. Of course, it goes hand in hand with modularity. Besides the possibility of growth, it reduces the effort needed for the development of a module because it now becomes possible to use the same piece of hardware in very divergent applications. This is necessary to get a proper cost effectiveness of the hardware. If you have such a system, you can divide your image processing task into a number of standard operations.

Due to the high data rates and the volumes of data, a fair degree of parallelism should be implementable. The parallelism can be obtained in three domains. The first one is a parallelism on the level of intermodule communication, which is obtained by using a parallel interconnection network or by using the transferred data more than once per transfer (broadcasting). A second kind of parallelism is the parallel working of modules. These modules can be identical ones, if one merely tries to speed up that module, or may be different ones to speed up the task as a whole. The modules may also be connected in a pipeline. This form of parallelism gives a system with a minimum of intermediate memory to store temporary results. However, it is only advantageous if all of the pipelined processors work at about the same throughput speed.

A last requirement is a technology independent structure. It can not be designed around a specific revolutionary new chip. Eventhough this would put you ahead of the competition, it limits the possibilities of the system to those of the chip or its family members. We feel that the design of a flexible architecture should be independent of the components used, otherwise the design will be stuck at a particular level of IC technology. The architecture should be modular enough to allow you to use a set of components fairly optimal in one module eventhough it will always be suboptimal to an architecture around these components alone.

The time-shared bus

Eventhough the bus structure may be the most common way to connect processors, a lot of other interconnection networks are available. A comparative study is made by Swartzlander [7]. He studied the performance and the cost of networks. One of our design goals was modularity, which can be obtained more easily with a distributed network. In this case, a processor hardware can be changed without affecting the global structure. Therefore, we choose as the network cost not the switch complexity, as did Swartzlander, but rather the switch complexity per processor. This is the extra cost of a processor, needed to put it into the network and it reflects the I/O complexity or the number of I/O lines needed per processor. This is an important implementation factor, because there is always a practical limit put to this number.

Table 1. Network comparison

NETWORK TYPE	PERFORMANCE	SWITCH TYPE	I/O COMPLEXITY
FULLY CONNECTED	N/2	N	N
CROSSBAR	1/4	1	N
STAR	1/4	N	N
CUBE	LOG(N)*(N/2)**1/2	2	2LOG(N)
RING	1	2	2
BUS	1/2N	1	1

In table 1, different networks are compared. The performance is computed as the product of the link bandwidth and the number of messages, divided by the round trip delay (as defined in [7]. The interconnection itself is characterised by the switch type. The switch complexity is here defined as the number of positions. The I/O complexity is the product of the switch complexity and the number of switches per processor. In the case of the star network, the central switch is considered.

If one wants a modular distributed network, the I/O complexity should be constant. If this is the case, each processor has a constant number of I/O lines, no matter how many processors are put in the network. It excludes the fully connected network, the crossbar network, the star network and the cube network from being considered as valuable candidates for our interconnection network. From the I/O point of view, the bus structure is more interesting than the ring network, because the latter uses twice as much I/O lines for the same data transfer. On the other hand, the ring network is much better regarding performance. Because it has an extra buffering at each stage of the network, the number of messages being transferred simultaneously can increase with the number of processors, while for the bus structure this number is constant, namely one. However, as long as the performance can be sustained by the bus, we prefer it for the former reason.

The properties of a bus structure are summarised in the following list.

- This network is the simpliest of all interconnection networks.
- It is a modular network. Parts are easy to modify and the position of a module is not important for the system performance. Furthermore, there is no extra cost to modify the network when one adds another processor.
- Communication delays can be held small, regardless of the physical realisation of the system.
- The system can be easily made fault tolerant for processor faults. On the other hand, bus errors are difficult to remedy, except perhaps by using multiple busses.
- The limited bandwidth may create problems at high data transfer volumes.

Efficiency study

The one serious argument against a bus structure seems to be its limited possibilities of high transfer volumes, which is especially the case for image processing. However, one should not exaggerate this problem, as will be shown in this chapter. We shall use figure 1 as an example throughout the chapter to illustrate the effect of the optimalisations. It shows the data flow of a particular task. In the case of a time-shared bus, each line gives a connection between processors, all at different time slices, each given a different number. After processor P1, data is demultiplexed into the two processors P2 and P3 for reasons of speed and after the parallel computing, multiplexed again into one data stream in processor P4. During bus cycle 5, the data is broadcasted to P5 en P8. Each processor black box is characterised here by the processing delay between input(s) and output(s) and its constancy. The processing can by pipelined in a processor itself and this pipelining time unit is a second processor characteristic. A last processor property is the size of the local buffering.

The bandwidth of the bus is equal to the sum of all timeslices multiplied by the data throughput speed at that position in the data flow. In most cases, the data throughput speed at a certain level is inversely proportional to the number of interconnections (or timeslices) needed at that level. This is illustrated by the two parallel processors P2 and P3 in the example. At that level, we need twice as much interconnection paths, but each one works at about half of the speed, because that is the reason we put two processors in parallel for. If this rule holds, the bandwidth is equal to the input data speed multiplied by the number of levels. The first exception to this rule is broadcasting. In this case, the

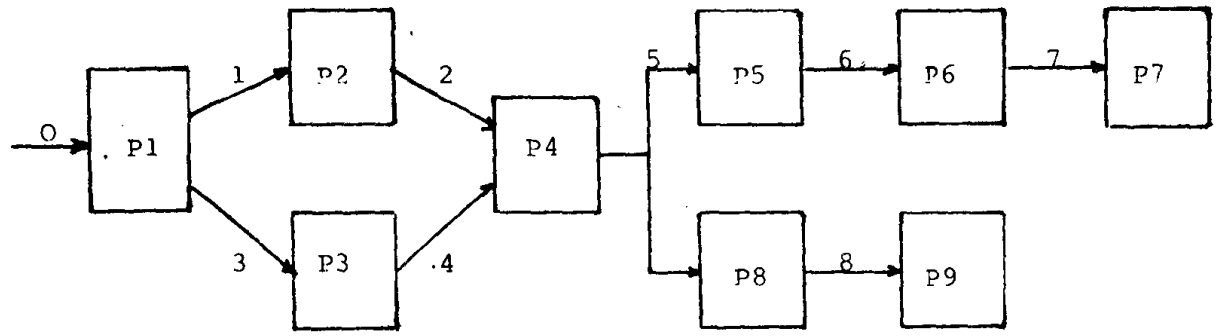

Figure 1: Data flow of an example, needing 9 busaccesses per data unit.

number of interconnections per level grows but the speed may remain the same. The second exception is the reduction of data. In that case, the number of interconnections remains constant but the throughput speed diminishes. So the low level operations are the most important ones in affecting the bandwidth. By choosing powerful processors (e.g. a small SIMD processor array), one can keep the number of high speed interconnection levels low. From this point on, only the low level interconnections will be taken into account in the models (data reduction is not considered).

The maximal bandwidth of the bus system, as used in the previous paragraph, is hardware defined. It is the product of the width of the data path and the bus frequency. On the other hand, the effective bandwidth is determined by data flow of the task itself and the hardware possibilities of the processors. The effective bandwidth is defined as the actual number of transfers divided by the total time needed to transfer these data. In the following paragraphs, some reasons for the loss of bandwidth are indicated, together with some possible remedies.

The actual bus structure is highly influenced by the unit of data transfer. The unit defines the minimum time, the bus is kept allocated to a certain interconnection. This implies that between multiple processors an intermediate storage must be provided, that is large enough to store an entire data unit. In the field of image processing (and certainly the low level image processing), the unit of computation is very often an image. The disadvantage of using such a large unit is that one needs a very large intermediate storage. An extra disadvantage is that the insertion of off-board intermediate memory requires one extra buscycle, so it reduces the effective bandwidth. On the other hand, the advantage is that the bus allocation is changed at a relatively slow rate, so the allocation hardware can be replaced by a software allocation routine. To avoid the storage problem, one may use multiple busses for the data transfer, but this helps only as long as the number of interconnections does not exceed the number of busses. A more radical solution is the reduction of the unit of data transfer to a single pixel. The intermediate storage of one pixel is no longer a problem but instead we get a more complex allocation hardware, because the bus is made available again after a pixel transfer. However, we feel that the loss in cost effectiveness due to the extra hardware is considerably less than to the extra image memories.

If processing delays are constant, the partitioning of the timeslices merely depends on the input data rate. However, if the delays are not chosen properly, conflicts may arise on the bus due to the time-sharing of the same wires. To avoid the buscontentions, one has to slow down the input rate. In some cases, the constant delays may even create incompatibilities, so the task will not be realisable. If in the example processor P5 and processor P8 have the same delay, bus cycle 6 and cycle 8 will always conflict, independently of the input rate. Another restriction to avoid incompatibilities, is that the closed loop delay must be zero. The closed loop delay is the sum of the delays of the processors, one encounters when going from a processor again to itself. Delays are added when traversing the processor from input to output and subtracted when traversing from output to input. In the example, this gives the following restriction (with D_{ij} = delay from inputcycle i to outputcycle j):

$$D_{01} + D_{12} + D_{25} - D_{45} - D_{34} - D_{03} = 0$$

These restrictions diminish the degrees of freedom in choosing the processor delays.

The minimum datacycle is defined as the number of buscycles between subsequent data inputs without any buscontention. This datacycle is affected by the dataflow of the task and the features of processors. Because each interconnection must get the bus once per data input, the absolute minimum datacycle is the number of interconnections times the buscycle. Due to these dataflow requirements, the minimum datacycle can never be smaller than 9 buscycles in

the example. The effect of the delays of the processors is illustrated by table 2. If the combined delay between any two interconnections in the dataflow is equal to a multiple of the datacycle, buscontention will arise between the data, presented at the input, separated exactly that multiple of the datacycle in time. The combined delay is the sum of the delays of the traversed processors (negative delays when travelling from output to output, e.g. $D_{68} = -D_{56} + D_{58}$).

Table 2. Influence of choosing the delays and pipelining on the minimum datacycle.

D_{01}	D_{03}	D_{12}	D_{34}	D_{25}	D_{56}	D_{67}	D_{58}	Pipe	Datacycle
10	12	10	10	12	10	10	12	yes	9
10	12	10	10	12	10	10	12	no	12
5	6	7	8	9	10	11	12	yes	20
5	6	7	8	9	10	11	12	no	20

Pipelining a processor is subdividing the computation in independent sequential subtasks and perform them in parallel. It has effect on the apparent speed of the processor. If a processor with an input-output delay of 10 buscycles, has a pipelining unit time of 5 buscycles, its apparent speed is doubled because it can now accept data every 5 cycles instead of every 10 cycles without pipelining. Of course, pipelining has no effect on the delay itself. Because a chain can never work faster than the slowest part, the minimum datacycle can not be smaller than the largest pipeline time unit. If processors are not pipelined, this time unit equals the computation delay (rows of table 2 without pipelining). One can get rid of this lower limit to the minimum datacycle by constructing every processor with a pipeline time unit equal to the buscycle (rows of table 2 with pipelining).

The fact that a certain task can be incompatible with the bus structure, is a serious threat to its use. It should be noted that by enlarging the delays of the processors, the datacycle not necessarily increases, as demonstrated by table 2, and in the same way one can get rid of the incompatibilities. If one only needs the system for one application, it is possible to include the supplementary delays on board to diminish the minimum datacycle. However, for research prototypes and systems for different applications, a more versatile way of changing the processor delays is needed. A FIFO buffering is a perfect solution. With a length of the FIFO equal to the datacycle divided by the pipeline time unit, one can limit the minimum datacycle to the number of interconnections. In this way, one can attribute the cycles of the minimum datacycle among the different interconnections. To realise the delays, one first computes the combined delay from the input to that interconnection. Recursively, starting from the input, the FIFO can make the combined delay equal to the smallest multiple of the datacycle, larger than the actual delay, augmented with the attributed cycle. The numbering of the attributed cycles in the example as shown in figure 1, results in a partitioning of the delays as indicated on the first row of table 2. Let us suppose that all the processors have a computation delay of 6 and we want to realise a minimum datacycle of 9 (it is the lower limit). To choose the delay D_{01}, we decide to give the output cycle of P1 number 1. Then we look for the first multiple of the datacycle, augmented by the attributed cycle, namely 1, which is 10. By taking a buffering of 4, this delay is achieved. For processor P2, the combined delay ($D_{01} + D_{12}$) must be a multiple of 9, augmented with 2. A buffering of 4 cycles on P2 will satisfy this requirement. The other interconnections are threated in an analog way. Of course, the local storage can be made much larger, because of computation requirements.

Until now, we have considered only constant computation delays for the processors. In real life, this is often not the case. A well known example is the dynamic memory, needing refresh. Every two or four milliseconds, the actual length of the memory cycles is doubled, if one addresses the memory in a random fashion. To include other possible delays in our model, we need a number of identical processors, but with a different delay, in the dataflow schematic. In figure 1, processor P8 could stand for a pseudo-processor, doubling e.g. the memory-processor P5, with a delay equal to the access delay plus the refresh delay. If a processor has many different possible delays, this will create an enormous number of modules. If the number of modules increases, the number of interconnections increases too, which leads to an increase of the minimum datacycle. This problem can be solved by using data transfers with handshaking (used together with local buffering). Without handshaking, one has to choose the minimum datacycle large enough to take into account every combination of varying delays. With handshaking, processing further on in the dataflow chain will be postponed until the data becomes ready. Because this is simply a shift in time, it has no influence on the effective bandwidth as in the case of no handshaking.

Conclusion

Thanks to its modularity and simplicity, the bus structured design is a very popular interconnection scheme. The limited bandwidth does not restrict the possibilities too much if the number of low level image processing stages is limited to a small number. Degradation of the effective bandwidth can be much more damaging. To remedy this, the following processor extensions are proposed. Processors are speeded up by pipelining. A local FIFO buffering and the data transfer with handshaking will allow bus optimisation. These concepts are tried out by a bus structure, designed at our lab [8].

References

1. Uhr, L., Algorithm-structured Computer Arrays and Networks, Academic Press, Orlando, Florida, 1984.
2. Hockney, R.W. and Jesshope, C.R., Parallel Computers, Adam Hilger Ltd., Bristol, United Kingdom, 1981.
3. Duff, M.J.B., Clip IV, A Large-scale Integrated Circuit Array Parallel Processor, Proc. 3th IJCPR, 1976, pp. 728-733.
4. Batcher, K., Design of a Massively Parallel Processor, I.E.E.E. Trans. on Computers, Vol. C-29, nr. 9, Sept. 1980, pp. 836-840.
5. Van Daele et al., LAVIM: the Leuven Automatic Visual Inspection Machine, Proc. S.P.I.E., 1979, Vol. 182, pp. 58-63.
6. Pratt, W.K., System Architecture of the Vicom Digital Image Processor, Proc. S.P.I.E., 1981, Vol. 301, pp. 78-82.
7. Swartzlander, E.E., Computer Networking in the Context of Very Large Scale Integration, Proc. S.P.I.E., 1982, Vol. 341, pp. 278-285.
8. Van Eycken et al., A Reconfigurable Multiprocessor Structure for Image Processing, Proc. I.E.E.E. CAPAIDM Workshop, Pasadena, Oct. 1983, pp. 236.

A dualistic model to describe computer architectures

Peter Nitezki
Michael Engel

Forschungszentrum Informatik an der Universität Karlsruhe
Haid- und Neu-Str. 10-14, D-7500 Karlsruhe 1

Abstract

The Dualistic Model for Computer Architecture Description uses a hierarchy of abstraction levels to describe a computer in arbitrary steps of refinement from the top of the user interface to the bottom of the gate level. In our Dualistic Model the description of an architecture may be divided into two major parts called "Concept" and "Realization".

The Concept of an architecture on each level of the hierarchy is an Abstract Data Type that describes the functionality of the computer and an implementation of that data type relative to the data type of the next lower level of abstraction. The Realization on each level comprises a language describing the means of user interaction with the machine, and a processor interpreting this language in terms of the language of the lower level. The surface of each hierarchical level, the data type and the language express the behaviour of a machine at this level, whereas the implementation and the processor describe the structure of the algorithms and the system. In this model the Principle of Operation maps the object and computational structure of the Concept onto the structures of the Realization.

Describing a system in terms of the Dualistic Model is therefore a process of refinement starting at a mere description of behaviour and ending at a description of structure. This model has proven to be a very valuable tool in exploiting the parallelism in a problem and it is very transparent in discovering the points where parallelism is lost in a special architecture. It has successfully been used in a project on a survey of Computer Architecture for Image Processing and Pattern Analysis in Germany.

Introduction

The description of computer architectures had been the first and basic problem in our project of a survey of German computer architecture for image processing and pattern recognition. We found the description of a special architecture being handled as an informal and individual process that depends on the author and the peculiarities of the machine being talked about. Formal approaches had seldom been made, the best-known one is the classification scheme by Flynn. Attempts have also been made on the gate and register level but merely as a specification tool. At present two formal systems are known in Germany, the first (of Giloi[1]) describes an architecture by the classification of control and information structure, the second (of Händler[2]) schematizes and counts the logic blocks and modules. However, they appeared not to be suitable for our problem in some aspects, therefore an own formal description tool was worked on.

The dualistic model

We based our attempt upon the experience that first every description of any subject consists of two aspects - the description of behaviour and the description of structure - second, as Prof. Baitinger[3] has shown in one of his courses, the mere description of behaviour is equivalent to the description of mere structure, and third there exist many levels in between. So Prof. Baitinger pointed out that computer architecture is the product of behaviour and structure.

Next we considered the importance of a clear and separated description of the mathematical aspects of a computer architecture. Therefore we chose to base our description tool on the theory of Abstract Data Types. Bauer[4] has proved this approach to be a valuable tool in the theory of programming. This background in mind we developed a bipartite model of the description process based on the theory of abstract data types and abstract machines.

The first part of this model - the Concept - is the description of the mathematical aspects by means of the theory of abstract data types, the second part - the Realization - is the description of the machine in terms of abstract processors. According to the idea of duality of behaviour and structure and according to the use of hierarchical levels in the theory of abstract data types the Concept is a hierarchy of abstract data types, in which every higher level is represented by an implementation of the higher type with functions of the lower one. On the other side the Realization is defined by a hierarchy of languages interpreted by processors that themselves are formulated by instructions of the lower language. So we have a hierarchy of Concept and Realization in which the data type and the language define the behaviour of the machine, the implementation and the processor define the structure. (See Figure 1)

A third part of the dualistic model, called the principle operation, maps every level of the Concept onto the Realization and therefore defines the rules how the structures of the implementation correspond to the structures of the processor. (See Appendix for the detailed plot of a correct definition). There is not enough

room to give a full example of an application of the Dualistic Model as shown in the reports[5,6]. Instead, some results of the theoretical considerations on algorithmic structures and computer architecture in the light of the presented model are shown.

Parallelity, seriality and algorithmic structures

Abstract data types define functions and sets of objects these functions are defined on. A data type is assumed as a primitive a higher data type is implemented on, e.g., the body of integers. It defines the basis relative to which the higher type (or types) and its properties are defined. Following the rules of abstract algebra objects and functions of the higher type can be defined by means of the primitives. Thus, composite objects with their corresponding constructing and selecting functions and composite functions defining the functions of the higher type are obtained. As the data type is a description of the behaviour, the process of its implementation is defining structure. A distinction between object and computational structure related to composite objects and composite functions is made. As an example the convolution of vectors of integers is considered. (See Figure 2). It is defined by the following equation:

$$\forall_{i \in [1,k]} \; G_i = \sum_{l=0}^{+n} W_l * F_{i+l} \qquad (1)$$

for vectors F with k elements and kernals W with n elements.

In the example the objects are structured by linear order and the computational structure is defined by disjoint binary trees of minimal depth. Using the rules of commutativity and distributivity one can recognize that this is not the only possible structure. Among the possible permutations there is also the tree of maximal depth. (See Figure 3.) As can be shown, there is one disjoint tree for every element of the result vector and each one of these trees is only different in the objects it transforms. Keeping this phenomenon in mind a little digression in graph theory follows.

Digression: parallelity and seriality as properties of the computational structure

- Starting from the resulting object every incoming arc of the computational structure is followed and the knots are marked by incrementing the number of its descendant starting with 0 at the result object. That number is called the level of the knot.
- The biggest number before reaching the input objects is defined as the Seriality of the computational structure.
- The most often occuring knot number is defined as the Parallelity of the computational structure.
- The product of parallelity and seriality devided by the total number of knots is defined as the Squareness of the computational structure.

Back to the example one can compute these numbers for the two different trees mentioned above.

minimum depth tree:

$$\text{seriality} = (1d\ n) + 1 \qquad (2)$$

$$\text{parallelity} = n * k \qquad (3)$$

$$\text{squareness} = \frac{n\ ((1d\ n) + 1)}{2n - 1} > 1 \qquad (4)$$

maximum depth tree:

$$\text{seriality} = n \qquad (5)$$

$$\text{parallelity} = 2k \qquad (6)$$

$$\text{squareness} = 1 \qquad (7)$$

Figure 2 and Figure 3 have shown that the computational structure consists of k disjoint trees. This fact is related to the appearence of figure k in equations (3) and (6), which shows that parallelity depends on the length of the vector to be convolved. Parallelity as a structural property therefore depends on the function and the object and may be divided into object parallelity and computational parallelity. Seriality defines the minimum number of steps the computation of the function needs. Squareness is related to the efficiency of execution of this function on a parallel computer. This figure gives the ratio of knots on the broadest and narrowest level in the computational structure and therefore it defines the ratio of unused processing elements in a computer which exploits maximum parallelity. The closer this figure is to 1 the more efficiently the structure may be mapped onto a pipelining or systolic processor.

Algorithm = logic + control

Following this sentence of Kowalski[7] we merely talked about logic up to now. Considering the precedence relation expressed by the arcs in the graphs the data dependence may be seen as control or, if prefered, the choice of one of the equivalent graphs expresses the control.

Control in algorithms is closely related to the notion of processing or the introduction of a sequence in time. This aspect of control is contained in the operation principle which states the rules of object representation and sequential versus parallel execution. The operation principle has to be seen as the mapping of objects and functions on the machine. So the machine, as third degree of freedom in the specification process, introduces control through the machine structure and the control of information flow.

In the top down process of "programming" a data type on a machine the progress of refinement and the choice of operation principle and machine reduce the possibilities of choice and introduce control. In this process parallelity is reduced as far as required by the introduction of control.

For example the minimum depth tree in Figure 2 is not suited for execution on a pipeline processor because there is no regular structure in a mapping on a linear order in time. The maximum depth tree optimally fits in that principle of operation, allthough parallelity is smaller.

As a consequence of the refinement process being a sequence of introduction of control and serialization the parallelity contained in the original problem is lost. The effort of compiler optimization and parallel execution of v. Neumann languages have proven the extraction of parallelity out of a serial realization as fruitless if not in most cases impossible. Therefore great care in the design process has to be taken as the introduction of control.

The design process

The dualistic model has some influence on the view of the design process. A consequent application of our ideas leads to some design rules of proceeding in the design process of a computer that seem to be useful in today's computer architecture:

1. Specify the data type carefully.
2. Choose a reasonable elementary level depending on the constraints of your design.
3. Choose appropriate intermediate levels to supply landmarks in the design process.
4. Try to find implementations with regular structures and look for the possibility to reduce the variety of structures to a small set of closely related ones.
5. Find appropriate building blocks to realize the functions of your concept.
6. Find a realizable structure for your processor which is easy to map onto the structures of your objects and functions.
7. Look for a principle of operation which exploits most parallelism in mapping the concept onto the machine.

Those rules form no independent set of instructions which garantee a good design. On the contrary, as every choice reduces the setof possible solutions in each of the tree parts of the description model, the design is more a guided trial and error process. Nevertheless, our experience is positive and therefore we think of our dualistic model as a good guideline in computer architecture.

Acknowledgement

This work has been carried out under the grant of the Bundesminister für Forschung und Technologie in the research group of Prof. Dr.-Ing. U. Rembold, Institut für Informatik III, Universität Karlsruhe.

Appendix: definitions

computer architecture:	description of a computer by specifying a hierarchy of virtual mashines that are represented by concept and realization.
concept:	hierarchy of abstract data types and their implementation
realization:	hierarchy of languages and processors that interpret these languages
data type:	definition of an abstact algebra by specifying a set of objects and functions basing upon these sets.
implementation:	representation of a type's objects and functions by object structures and computational structures of a primitive.
processor:	map of a language onto a primitive, given by a function principle and an organization.
organization:	the set of a processor's functional units, their connecting structure and the related communication control.
function principle:	a processor's object representation, its information control, the processing structure and processing control.
object representation:	description of the representation of an object within the abstract mashine.
information control:	control to handle the representation of objects.
processing structure:	representation of the structure of the algorithms in order to interpret the higher by instructions of the primitive.
processing control:	control of the flow of instructions in time and space.
primitive:	mashine level relative to which a higher level is defined.
elemental:	mashine level without an explicit definition, i.e., for an elemental level neither an implementation nor a processor must be specified. The elemental level constitutes a reference system relative to which other levels can be defined.

References

1. Giloi Rechnerarchitektur, Springer, 1981
2. Bode, Händler Rechnerarchitektur, Springer, 1983
3. Baitinger Seminar des Instituts für Technik der Informationsverarbeitung, Karlsruhe, 1983 (unpublished)
4. Bauer, Wössner Algorithmische Sprache und Programmentwicklung, Springer, 1981
5. Engel, Nitezki Workshop "Parallele Rechnerarchitekturen in der Mustererkennung und Bildverarbeitung", Fachinformationszentrum Energie Physik Mathematik, Karlsruhe, 1984
6. Engel, Levi, Nitezki Voruntersuchung über geeignete Rechnerarchitekturen in der Mustererkennung, Fachinformationszentrum Energie Physik Mathematik, Karlsruhe, 1984
7. Kowalski Algorithm = Logic + Control, C.ACM, July 1979, Vol. 22, Nr. 7

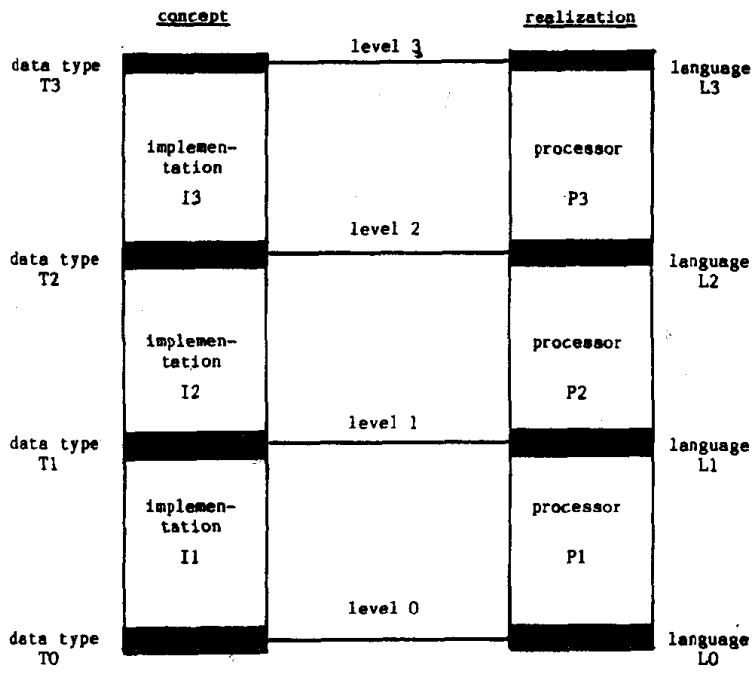

Figure 1. Structure of the dualistic model.

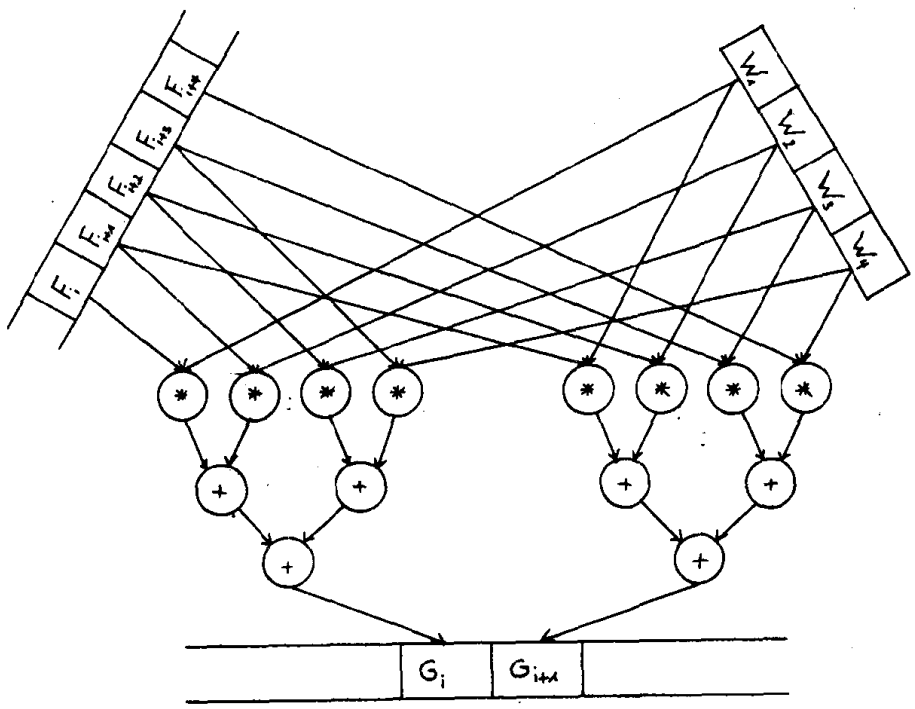

Figure 2. Structure of an implementation of 1D-convolution.

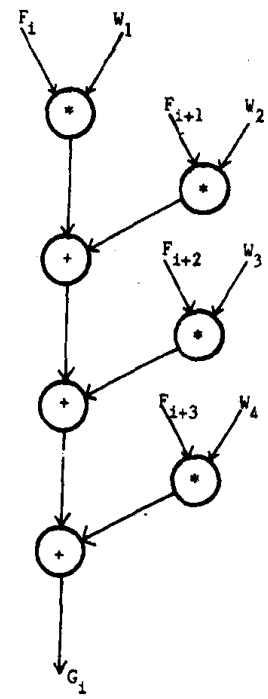

Figure 3. Part of the computational structure for 1D-convolution.