Parallel processing systems

Edited by

DAVID J. EVANS

Parallel processing systems

Edited by

DAVID J.EVANS

Professor of Computing, University of Technology, Loughborough

CAMBRIDGE UNIVERSITY PRESS

Cambridge

London New York New Rochelle

Melbourne Sydney

Published by the Press Syndicate of the University of Cambridge
The Pitt Building, Trumpington Street, Cambridge CB2 1RP
32 East 57th Street, New York, NY 10022, USA
296 Beaconsfield Parade, Middle Park, Melbourne 3206, Australia

First published 1982

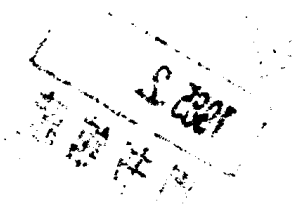
Printed in Great Britain at the University Press, Cambridge

Library of Congress catalogue card number: 81-38445

British Library cataloguing in publication data

Parallel processing systems

1. Parallel processing (Electronic computers)
2. Electronic digital computers
I. Evans, David J.
001.64 QA76.6
ISBN 0 521 24366 1



PREFACE

This publication is based on an Advanced Course on Parallel Processing Systems given at the University of Technology, Loughborough, from 15 September to 26 September, 1980. The course was sponsored by the Science Research Council under the auspices of the Informatics Training Group of the EEC Scientific and Technical Research Committee (CREST). The course lecturers provided a carefully arranged integrated programme of lectures, which were supplemented by specially invited contributions.

CONTRIBUTORS

Course director
D. J. Evans, Department of Computer Studies, University of Technology, Loughborough, Leicestershire LE11 3TU, UK

Professor J.-L. Baer, Department of Computer Science, University of Washington, Seattle, Washington 98195, USA
Dr R. H. Barlow, Department of Computer Studies, University of Technology, Loughborough, Leicestershire LE11 3TU, UK
Professor P. H. Enslow Jr and Timothy G. Saponas, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, Georgia 30332, USA
Professor D. J. Evans, Department of Computer Studies, University of Technology, Loughborough, Leicestershire LE11 3TU, UK
Professor Dr M. Feilmeier, Institut für Rechentechnik, Technische Universität Braunschweig, Pockelsstrasse 14, 3300 Braunschweig, West Germany
Professor C. Girault, Université Pierre et Marie Curie, U.E.R. 50, Institut de Programmation, Tour 55-65, 4 Place Jussieu, 75230 Paris Cedex 05, France
Professor Dr W. Händler, Institut für Mathematische Maschinen und Datenverarbeitung (Informatik), Friedrich-Alexander-Universität Erlangen-Nürnberg, Martensstrasse 3, 8520 Erlangen, West Germany
Dr M. Hatzopoulos, Unit of Applied Mathematics II, University of Athens, Panepistemiopolis, Athens 621, Greece
Professor D. J. Kuck, Department of Computer Science, 222 Digital Computer Laboratory, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, USA
Professor H. T. Kung, Department of Computer Science, Carnegie-Mellon University, Pittsburg, Pennsylvania 15213, USA
Dr I. A. Newman, Department of Computer Studies, University of Technology, Loughborough, Leicestershire LE11 3TU, UK

Professor D. Parkinson, DAP Support Unit, Computer Centre, Queen Mary College, University of London, Mile End Road, London E1 4NS. UK

Dr G. Roucairol, Université Pierre et Marie Curie, U.E.R. 50, Institut de Programmation, Tour 55-65, 4 Place Jussieu, 75230 Paris Cedex 05, France

Mr A. J. Slade, Department of Computing, Science Laboratories, University of Durham, South Road, Durham DH1 3LE, UK

Professor J.-C. Syre, ONERA-CERT, Complexe Aérospatial de Lespinet, 2 Avenue Edouard Belin, BP No. 4025, 31055 Toulouse, France

Dr P. Treleaven, Computing Laboratory, University of Newcastle upon Tyne, Claremont Tower, Claremont Road, Newcastle upon Tyne NE1 7RU, UK

Dr S. A. Williams, Department of Computer Science, The University of Reading, Whiteknights Park, Reading, Berkshire RG6 2AX, UK

Dr M. C. Woodward, Department of Computer Studies, University of Technology, Loughborough, Leicestershire LE11 3TU, UK

INTRODUCTION

D. J. EVANS

In the last few years dramatic improvements in circuit device and computer architecture technologies together with new concepts in system organisation have brought forth many new innovative computer systems which are capable of supporting a large number of concurrent operations and activities. These parallel computing systems will undoubtedly create new research and development areas when they are built and programmed to attain their maximum cost-effectiveness and benefit society by revealing hitherto unknown solution strategies and applications. The advances realised to date represent little progress when we compare what is known about parallel computation with the corresponding body of knowledge for serial computation. Indeed, many challenging problems in the fields of programming languages and compilers must be solved before the advantages of the proposed parallel architectures can be fully exploited.

This book is divided into six parts covering the following topics: parallel and distributed computer systems, analysis of parallel programming systems, multiple instruction multiple data (MIMD) computer systems, single instruction multiple data (SIMD) computer systems, data flow processors and parallel computer algorithms. Each chapter of the book was originally prepared as documentation for lectures presented at the Advanced Course on Parallel Processing Systems at the University of Technology, Loughborough, England in September 1980. The aims and purpose of the advanced course were to accelerate the interchange of information in this increasingly important area and to examine in depth and provide discussion of problems amongst the scientists and computer professionals who may use parallel processor systems and those who design or program such systems. The original course material has been extensively revised for publication in its present form. The book aims to provide a coherent account of all major aspects of parallel processing and also to give an up-to-date account of recent activity in this area.

CONTENTS

1 Innovative computer architecture – how to increase parallelism but not complexity

WOLFGANG HÄNDLER

1 Introductory remarks

'Parallelism' is used as a general term to characterize all the different kinds of simultaneity occurring in modern computers. In this general sense it includes pipelining. In the following text we have nevertheless to differentiate parallelism (in a more special sense) as opposed to pipelining. In cases where we refer to the general and overall meaning we will call it 'parallelism in general' or 'parallelism i.g.'.

Parallelism in general offers great opportunities regarding:

(*a*) improved overall performance
(*b*) improved availability and safety.

The second point cannot be covered sufficiently in this article. It is nevertheless a key point in the evolution towards parallelism. With respect to the first point, we consider all those prerequisites that make it possible to improve performance. Evidently parallelism *per se* does not guarantee such an improvement, as some contemporary concepts show. So a case study of useful parallelism is given in section 3 and the ideas behind it in section 4. Section 2 gives a general overview and orientation with respect to all possible contemporary and future forms of parallelism.

The classical computer, as defined by Burks, Goldstine and von Neumann [1], consists of (see figure 1.1):

program-control or control unit (CU)
arithmetic unit or arithmetic and logic unit (ALU)
memory or storage unit
input/output unit (I/O).

Later versions of this classical concept made it possible for at least two of these units to work at the same time, which was not the case during the pioneer period. Machine words or numbers, normally stored in a hardware unit called a memory cell, are also referred to as parallel words in cases where such words are processed by taking all positions (or bits) at the same time. In this sense we find some forms of parallelism i.g. in early computers.

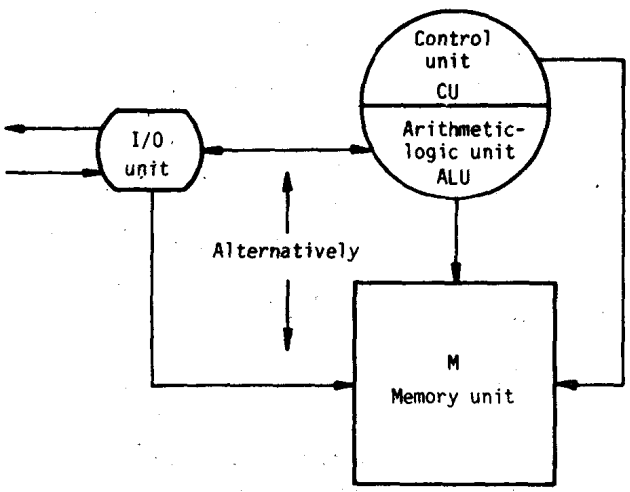
In this presentation we will centre more on all possible forms of parallelism i.g. that distinctly go beyond the word parallelism as used in the context of the 'parallel computers' of the fifties and the sixties.

True parallelism (of any kind) requires the existence of more than one unit working at the same time. Such units can be control units (CU), arithmetic and logic units (ALU), input/output units and memory units, while simultaneous operation of the latter certainly can result in a speedup of the overall operation, they nevertheless do not influence the structure of processing, as the other units can.

In a classical structure the control unit (CU) together with an arithmetic and logic unit (ALU) form a central processor unit or, simply, a processing unit (PU). Replicating either PUs themselves or the number of ALUs connected to each PU results in proper parallelism i.g. Making available many PUs means that many programs or many parts of one program can be processed at the same time, i.e. in parallel (figure 1.2). Connecting many ALUs with one CU means that many data words can be processed in parallel by one program, which this CU is interpreting (figure 1.3). The PU with many ALUs, therefore, is more powerful than one of the forementioned PUs with only one ALU each.

In contrast to this parallelism, pipelining also requires - as defined above - a multitude of resources like PUs or ALUs. While one set of data is processed by one of the resources, another set of data can be processed by another resource next to it, etc. In this sense the data words are flowing from one resource (PU or ALU) to the next one in a chain process. During the whole process n tasks are performed, e.g. one after the other on one set of data. $1n$ such sets may be in

Figure 1.1. The classical computer.



a 'pipeline' at the same time, where each one is at a different stage of the process.

Parallelism and pipelining can both be seen in three different logical levels:

	Parallelism	Pipelining
Program level	Multiprocessor	Macro-pipelining
Instruction level	Multiple ALUs (array processor)	Instruction pipelining
Word level	Multiple bits (wordlength, if greater than 1)	Arithmetic pipelining

With respect to pipelining these processing forms can be represented in block diagrams as shown in figures 1.4-1.6. Instead of differentiating three different

Figure 1.2. A multiprocessor, consisting of a common memory and several processing units.

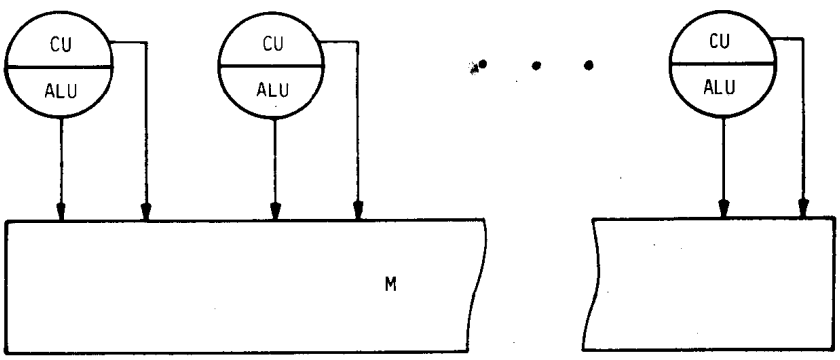
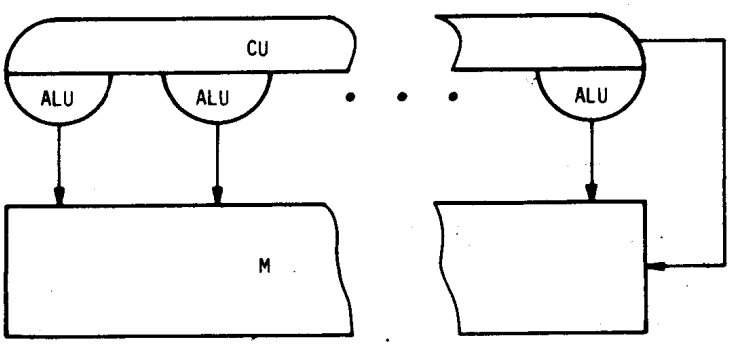


Figure 1.3. Parallel data processing of one program by several ALUs, controlled by one CU.



levels contemporary papers very often talk only about 'pipelining', which in many cases is not sufficient.

In cases where the multiplicity of hardware resources (e.g. ALUs) is n, one cannot gain an n-fold increase in performance by choosing an appropriate arrangement. This point will be considered in more detail in section 3.

With these six basic forms of parallelism i.g. (illustrated in figures 1.1-1.6) it is possible to introduce a classification scheme, which forms a starting point allowing us to build up a wide variety of composed computer structures, and which at the same time reflects historical as well as evolutionary aspects of these structures [2].

Figure 1.4. Macro-pipelining.

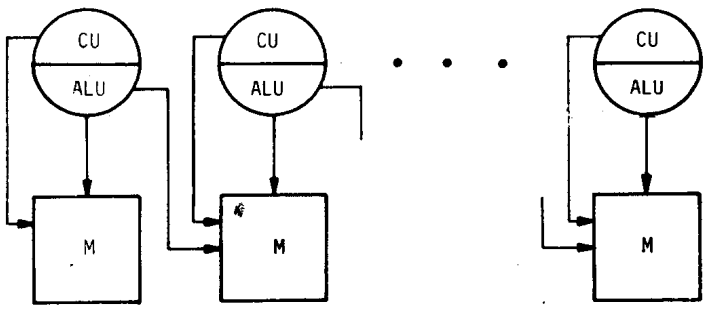
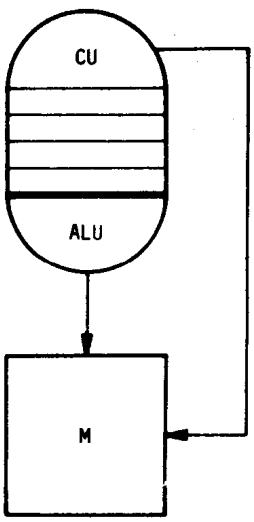


Figure 1.5. Instruction pipelining.



2 Classification and taxonomy of computer structures

A classification scheme has mainly to fulfil the following requirements:

(*a*) It must be possible to classify all existing as well as all foreseeable computer structures.

(*b*) It must differentiate essential processing structures.

(*c*) Any subject or computer structure must be assigned a unique classification.

Some existing classification schemes seem to violate just these rules [3]. They present only very rough categories, exclude viable structures and are not able to uniquely classify any kind of pipelining.

The Erlangen Classification System (ECS) largely avoids these disadvantages. It is mainly based on two points:

(*a*) It introduces a very simple, lucid but rigid triplet as a characterization for basic structures.

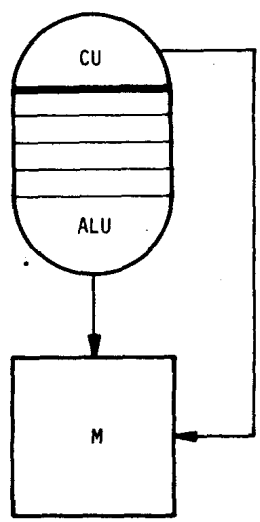
(*b*) It introduces operations +, * and v in order to make compositions of structures available and in order to represent a certain 'flexibility'.

In this context 'flexibility' (or 'versatility') is defined as the number of different operation modes in which existing hardware can be utilized.

Disregarding for the present the three types of pipelining we may define the Erlangen triplet [4]:

$$t = (k, d, w)$$

Figure 1.6. Arithmetic pipelining.



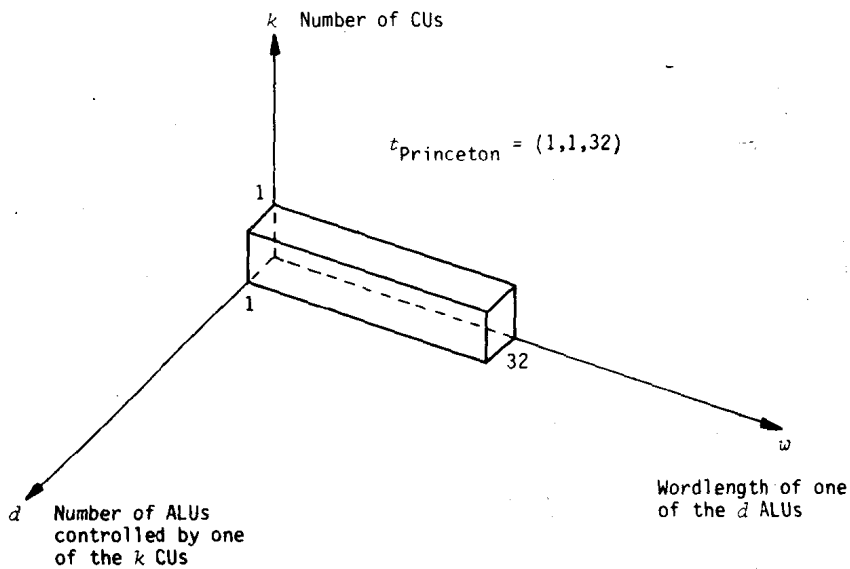
where

- k is the number of control units (CUs) interpreting a program
- d is the number of arithmetic and logic units (ALUs) controlled by one of the k control units
- w is the wordlength or number of bits handled in one of the d ALUs.

This characteristic can be seen as a point in a three-dimensional space (figure 1.7). It immediately follows that the characterized computer structure has k CUs of the same type and each one of the d ALUs (connected to each of the CUs) has to be of the same type. This homogeneity can also be seen with respect to the bits of a word in one of the d ALUs.

Simple computers of the Princeton type can be characterized by a triplet $t = (1,1,w)$ (see figure 1.7). Early Princeton computers had, e.g. 36-, 48- or 32-bit wordlength, so that we have (1,1,36), (1,1,48) or (1,1,32) as triplets. The simplest possibility, which always seems to be possible, is the triplet (1,1,1). Indeed this simplest form was realized in Europe during the fifties by van der Poël [5] and by T. Fromme [6], who called his computer the 'Minima'. More popular similar computers were called 'serial computers' at this time because the bits were processed one after another. The idea behind this very simple structure was to save hardware and money. Fast evolving hardware has since caused computer designers to change their point of view. A contributory fact was that

Figure 1.7. Three-dimensional representation of the elementary parallel-characteristic triplet of a computer of the Princeton type.



most control elements do not permit a serialization (sequentialization) without bringing about an additional complexity.

It has to be mentioned that the characterization by such very simple triplets is only valid if there is no autonomous I/O control, which in some cases can be programmed separately.

Having introduced these simple triplets or structures we can introduce combinations of structures such as really make up the contemporary computer generation. However, before we give some examples of such composites we have to direct our attention to all kinds of pipelining because these can be ingredients of a basic triplet or structure.

An elementary parallel-pipelining characteristic is an extended triplet

$$t = (k*k', d*d', w*w'),$$

where k, d, w are as defined above, and

- k' is the number of control units (CUs) interpreting tasks of a program, whereby the data flow through these units (processor) is sequential: macro-pipeline
- d' is the number of function units (ALUs) controlled by one CU and working on one data stream: instruction pipeline
- w' is the number of levels or phases in an arithmetic pipeline.

Naturally the 'extended triplet' could be written in the form of a sextet. But such a form would be less readable, and the fact that the entities are well defined by the separating commas was decisive in the triplet style being adopted.

Simple examples of computers, which essentially correspond to the von Neumann type (Princeton type) but equipped with certain forms of pipelining, are as follows

CDC 7600 $t_{\text{CDC 7600-CP}} = (1*1, 1*9, 60*1) = (1,*9,60)$

in a simplified notation with the factor 1 dropped (see figure 1.8). Similarly figure 1.9 corresponds to

CDC STAR $t_{\text{CDC STAR-CP}} = (1,2,64*4)$

(where in the same manner the ones have been cancelled), and figure 1.10 is

TI ASC $t_{\text{TI ASC-CP}} = (1,4,64*8).$

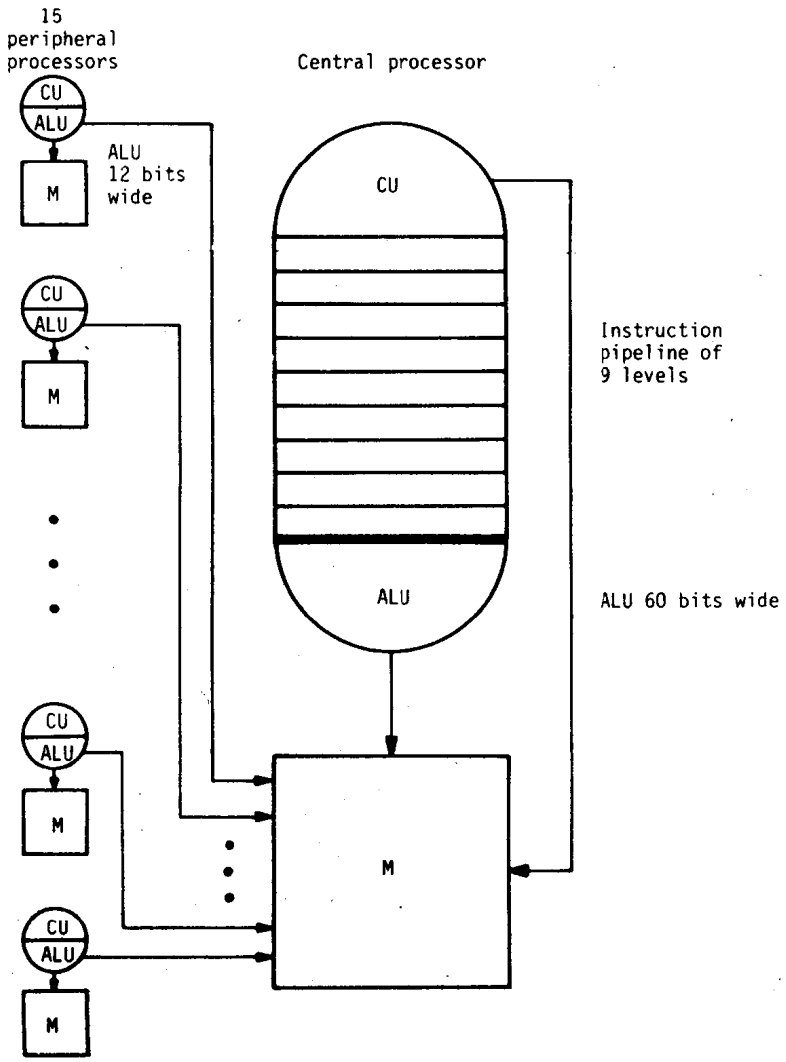
The first example, CDC 7600 [7], is a representation of the central processor only. Otherwise we would have to add the notations for peripheral processor, front-end processor, etc., for which the appropriate operators are introduced later.

The notation $*9$ in $t_{\text{CDC 7600-CP}}$ means 'nine function units' which play the role of the ALU in this concept.

The notation 64∗4 in $t_{\text{CDC STAR-CP}}$ [8] means 'an arithmetic pipeline with four levels' working on 64-bit information. In the second position of the triplet the 2 signifies that two such pipelines are allocated.

The notation 64∗8 in $t_{\text{TI ASC-CP}}$ [9] accordingly means that arithmetic pipelines with eight levels are acting as ALUs. In the second position the 4 represents that four such arithmetic pipelines exist.

Figure 1.8. CDC 7600. Five 12-bit words from the peripheral processors are transferred into one 60-bit word of the central processor. The reverse transfer is done in a similar way.



It should be mentioned that in the last two examples the number of pipelines is optional. The numbers in these examples give the numbers for the full extension of each model.

With the given triplets all elementary structures can be characterized. Because the three parts of a triplet refer to a specific homogeneous structure, one has now to introduce composition rules to make possible the characterization of more complex structures.

With the basic triplets given above it is already possible to characterize either a multitude of (essentially) equal control units (CUs), each able to interpret a program, or a multitude of arithmetic and control units (ALUs), each able to process one data stream. If there are diverse elements instead of one triplet a composition of them has to be applied. For this reason we introduce the operators:

+ (plus) for the existence of more than one structure, in particular for diverse structures (because otherwise they might be united into one elementary triplet, as pointed out earlier), as an alternative for the data, whereby they may (normally) be processed, each item according to its specific nature on the best studied structure.

Figure 1.9. CDS STAR. Two ALUs each 64 bits wide. Arithmetic pipelines of 4 levels.