PIPELINED PROCESSOR FARMS

STRUCTURED DESIGN FOR EMBEDDED PARALLEL SYSTEMS

MARTIN FLEURY
ANDREW DOWNTON

WILEY SERIES ON PARALLEL AND DISTRIBUTED COMPUTING
ALBERT Y. ZOMAYA, SERIES EDITOR

Pipelined Processor Farms

WILEY SERIES ON PARALLEL AND DISTRIBUTED COMPUTING
Series Editor: Albert Y. Zomaya

Parallel and Distributed Simulation Systems / Richard Fujimoto

Surviving the Design of Microprocessor and Multimicroprocessor Systems: Lessons Learned / Veljko Milutinović

Mobile Processing in Distributed and Open Environments / Peter Sapaty

Introduction to Parallel Algorithms / C. Xavier and S. S. Iyengar

Solutions to Parallel and Distributed Computing Problems: Lessons from Biological Sciences / Albert Y. Zomaya, Fikret Ercal, and Stephan Olariu (*Editors*)

New Parallel Algorithms for Direct Solution of Linear Equations / C. Siva Ram Murthy, K. N. Balasubramanya Murthy, and Srinivas Aluru

Practical PRAM Programming / Joerg Keller, Christoph Kessler, and Jesper Larsson Traeff

Computational Collective Intelligence / Tadeusz M. Szuba

Parallel and Distributed Computing: A Survey of Models, Paradigms, and Approaches / Claudia Leopold

Fundamentals of Distributed Object Systems: A CORBA Perspective / Zahir Tari and Omran Bukhres

Pipelined Processor Farms: Structured Design for Embedded Parallel Systems / Martin Fleury and Andrew Downton

Pipelined Processor Farms

Structured Design for Embedded Parallel Systems

Martin Fleury
Andrew Downton

A Wiley-Interscience Publication
JOHN WILEY & SONS, INC.
New York / Chichester / Weinheim / Brisbane / Singapore / Toronto

This text is printed on acid-free paper. ♾

Published simultaneously in Canada.

For ordering and customer service, call 1-800-CALL-WILEY.

Library of Congress Cataloging in Publication Data is available.

ISBN 0-471-38860-2

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Foreword

Parallel systems are typically difficult to construct, to analyse, and to optimize. One way forward is to focus on stylized forms. This is the approach taken here, for Pipelined Processor Farms (PPF). The target domain is that of embedded systems with continuous flow of data, often with real-time constraints.

This volume brings together the results of ten years study and development of the PPF approach and is the first comprehensive treatment beyond the original research papers. The overall methodology is illustrated throughout by a range of examples drawn from real applications. These show both the scope for practical application and the range of choices for parallelism both in the pipelining and in the processor farms at each pipeline stage. Freedom to choose the numbers of processors for each stage is then a key factor for balancing the system and for optimizing performance characteristics such as system throughput and latency. Designs may also be optimized in other ways, e.g. for cost, or tuned for alternative choices of processor, including future ones, providing a high degree of future-proofing for PPF designs.

An important aspect is the ability to do "what if" analysis, assisted in part by a prototype toolkit, and founded on validation of predicted performance against real.

As the exposition proceeds, the reader will get an emerging understanding of designs being crafted quantitatively for desired performance characteristics. This in turn feeds in to larger scale issues and trade-offs between requirements, functionality, benefits, performance, and cost. The essence for me is captured by the phrase "engineering in the performance dimension".

CHRIS WADSWORTH
TECHNICAL CO-ORDINATOR
EPSRC PROGRAMME ON
PORTABLE SOFTWARE TOOLS
FOR PARALLEL ARCHITECTURES

Wantage, Nov.2000

Preface

In the 1980s, the advent of the transputer led to widespread investigation of the potential of parallel computing in embedded applications. Application areas included signal processing, control, robotics, real-time systems, image processing, pattern analysis and computer vision. It quickly became apparent that although the transputer provided an effective parallel hardware component, and its associated language Occam provided useful low-level software tools, there was also a need for higher-level tools together with a systematic design methodology that addressed the additional design parameters introduced by parallelism.

Our work at that time was concerned with implementing real-time document processing systems which included significant computer vision problems requiring multiple processors to meet throughput and latency constraints. Reviews of similar work highlighted the fact that processor farms were often favored as an effective practical parallel implementation architecture, and that many applications embodied an inherent pipeline processing structure. After analyzing a number of our own systems and those reported by others we concluded that a combination of the pipeline structure with a generalized processor farm implementation at each pipeline stage offered a flexible general-purpose architecture for soft real-time systems. We embarked upon a major project, PSTESPA (Portable Software Tools for Embedded Signal Processing Applications) to investigate the scope of the Pipeline Processor Farm (PPF) design model, both in terms of its application potential and the supporting software tools it required. Because the project focused mostly upon high-level

design issues, its outcome largely remains valid despite seismic changes within the parallel computing industry.

By the end of our PSTESPA project, notwithstanding its successful outcome, the goalposts of parallel systems had moved, and it was becoming apparent that many of the ambitious and idealistic goals of general-purpose parallel computing had been tempered by the pragmatic reality of market forces. Companies such as Inmos, Meiko, Parsys and Parsytec (producing transputer-based machines), and ICL, AMT, MasPar and Thinking Machines (producing SIMD machines), found that the market for parallel applications was too fragmented to support high-volume sales of large-scale parallel machines based upon specialized processing elements, and that application development was slow and difficult with limited supporting software tools. Shared-memory machines produced by major uniprocessor manufacturers such as IBM, DEC, Intel and Silicon Graphics, and distributed Networks of Workstations (NOWs) had however established a foothold in the market, because they are based around high-volume commercial off-the-shelf (COTS) processors, and achieved penetration in markets such as database and file-serving where parallelism could be supported within the operating system.

In our own application field of embedded systems, NOWs and shared-memory machines have a significant part to play in supporting the parallel logic development process, but implementation is now increasingly geared towards hardware-software co-design. Co-design tools may currently be based around heterogeneous computing elements ranging from conventional RISC and DSP processors at one end of the spectrum, through embedded processor cores such as ARM, to FPGAs and ASICs at the other. Historically, such tools have been developed bottom-up, and therefore currently betray a strong hardware design ethos, and a correspondingly weak high-level software design model. Our current research (also funded by EPSRC) is investigating how to extend the PPF design methodology to address this rapidly developing embedded applications market using a software component-based approach, which we believe can provide a valuable method of unifying current disparate low-level hardware-software co-design models. Such solutions will surely become essential as complex multimedia embedded applications become widespread in consumer, commercial and industrial markets over the next decade.

ANDY DOWNTON

Colchester, Oct. 2000

Acknowledgments

Although this book has only two named authors, many others have contributed to its content, both by carrying out experimental work and by collaborating in writing the journal and conference papers from which the book is derived.

Early work on real-time handwritten address recognition, which highlighted the problem to be addressed, was funded by the British Post Office, supported by Roger Powell, Robin Birch and Duncan Chapman. Algorithmic developments were carried out by Ehsan Kabir and Hendrawan, and initial investigations of parallel implementations were made by Robert Tregidgo and Aysegul Cuhadar, all of whom received doctorates for their work. In an effort to generalise the ideas thrown up by Robert's work in particular, further industrial contract work in a different field, image coding, was carried out, funded by BT Laboratories through the support of Mike Whybray.

Many people at BT contributed to this work through the provision of H.261 image coding software, and (later) other application codes for speech recognition and microphone beam forming. Other software applications, including those for model-based coding, H.263, and Eigenfaces were also investigated in collaboration with BT. In addition to Mike Whybray, many others at BT laboratories provided valuable support for work there, including Pat Mulroy, Mike Nilsson, Bill Welsh, Mark Shackleton, John Talintyre, Simon Ringland and Alwyn Lewis. BT also donated equipment, including a Meiko CS2 and Texas TMS320C40 DSP systems to support our activities.

As a result of these early studies, funding was obtained from the EPSRC (the UK Engineering and Physical Sciences Research Council) to investigate the emergent PPF design methodology under a directed program on Portable Software Tools for Parallel Architectures (PSTPA). This project — PSTESPA (Parallel Software Tools for Embedded Signal Processing Applications) — enabled us not only to generalise the earlier work, but also to start investigating and prototyping software tools to support the PPF design process. Chris Wadsworth from Rutherford Appleton Laboratories was the technical coordinator of this program, and has our heartfelt thanks for the support and guidance he provided over a period of nearly four years. Adrian Clark, with extensive previous experience of parallel image processing libraries, acted as a consultant on the PSTESPA project, and Martin Fleury was appointed as our first research fellow, distinguishing himself so much that before the end of the project he had been appointed to the Department's academic staff. Several other research fellows also worked alongside Martin during the project: Herkole Sava, Nilufer Sarvan, Richard Durrant and Graeme Sweeney, and all contributed considerably to its successful outcome, as is evidenced by their co-authorship of many of the publications which were generated.

Publication of this book is possible not only because of the contributions of the many collaborators listed above, but also through the kind permission of the publishers of our journal papers, who have permitted us to revise our original publications to present a complete and coherent picture of our work here. We particularly wish to acknowledge the following sources of tables, figures and text extracts which are reproduced from previous publications:

The *Institution of Electrical Engineers (IEE)*, for permission to reprint:

- portions of A. C. Downton, R. W. S. Tregidgo, and A. Çuhadar, Top-down Structured parallelization of embedded image processing applications. *IEE Proceedings Part I (Vision, Image, and Signal Processing)*, 141(6):438-445, 1994 as text in Chapter 1, as Figure 1.1 and A.1–A.4, and as Table A.1;

- portions of M. Fleury, A. C. Downton, and A. F. Clark, Scheduling schems for data farming, *IEE Proceedings Part E (Computers and Digital Techniques)*, in press at the time of writing, as text in Chapter 6, as Figures 6.1–6.9, and as Tables 6.1 and 6.2;

- portions of A. C. Downton, Generalised approach to parallelising image sequence coding algorithms, *IEE Proceedings I (Vision, Image, and Signal Processing)*, 141(6):438-445, 1994 as text in Section 8.1, as Figures A.6–8.12, and as Tables 8.1 and 8.2;

- portions of H. P. Sava, M. Fleury, A. C. Downton, and A. F. Clark, Parallel pipeline implementation of wavelet transforms, *IEE Proceedings Part I (Vision, Image, and Signal Processing)*, 144(6):355-359, 1997 as text in Section 9.2, and as Figures 9.6–9.10;

- portions of M. Fleury, A. C. Downton, and A. F. Clark, Scheduling schemes for data farming, *IEE Proceedings Part E (Computers and Digital Techniques)*, 146(5):227-234, 1994 as text in Section 11.9, as Figures 11.11–11.17, and as Table 11.6;

- portions of M. Fleury, H. Sava, A. C. Downton, and A. F. Clark, Design of a clock synchronization sub-system for parallel embedded systems, *IEE Proceedings Part E (Computers and Digital Techniques)*, 144(2):65-73, 1997 as text in Chapter 12, as Figures 12.1–12.4, and as Tables 12.1and 12.2.

Elsevier Science, for inclusion of the following:

- portions reprinted from *Microprocessors and Microsystems*, 21, A. Çuhadar, A. C. Downton, and M. Fleury, A structured parallel design for embedded vision systems: A case study, 131-141, Copyright 1997, with permission from Elsevier Science, as text in Chapter 3, as Figures 3.1–3.10, and as Table 3.1 and 3.2;

- portions reprinted from *Image and Vision Computing*, M. Fleury, A. F. Clark, and A. C. Downton, Prototyping optical-flow algorithms on a parallel machine, in press at the time of writing, Copyright 2000, with permission from Elsevier Science, as text in Section 8.4, as Figures 8.19–8.28, and as Tables 8.8–8.12;

- portions of *Signal Processing: Image Communications*, 7, A. C. Downton, Speed-up trend analysis for H.261 and model-based image coding algorithms using a parallel-pipeline model, 489-502, Copyright 1995, with permission from Elsevier Science, as text in Section 10.2, Figures 10.5–10.7, and Table 10.2.

Springer Verlag, for permission to reprint:

- portions of H. P. Sava, M. Fleury, A. C. Donwton, and A. F. Clark, A case study in pipeline processor farming: Parallelising the H.263 encoder, in *UK Parallel'96*, 196-205, 1996, as text in Section 8.2, as Figures 8.13–8.15, and as Tables 8.3–8.5;

- portions of M. Fleury, A. C. Downton, and A. F. Clark, Pipelined parallelization of face recognition, *Machine Vision Applications*, in press at the time of writing, as text in Section 8.3, Figures 5.1 and 5.2, Figures 8.16–8.18, and Tables 8.6 and 8.7;

- portions of M. Fleury, A. C. Downton, and A. F. Clark, Karhunen-Loève transform: An exercise in simple image-processing parallel pipelines, in *Euro-Par'97*, 815-819, 1997, as text in Section 9.1, Figures 9.4–9.5;

- portions of M. Fleury, A. C. Downton, and A. F. Clark, Parallel structure in an integrated speech-recognition network, in *Euro-Par'99*, 995-1004, 1999, as text in Section 10.1, Figures 10.1–10.4, and Table 10.1.

Academic Press, for permission to reprint:

- portions of A. Cuhadar, D. G. Sampson, and A. C. Downton, A scalable parallel approach to vector quantization, *Real-Time Imaging*, 2:241-247, 1995, as text in Section 9.3, Figures 9.11–9.19, and Table 9.2.

The *Institute of Electrical and Electronic Engineers (IEEE)*, for permission to reprint:

- portions of M. Fleury, A. C. Downton, and A. F. Clark, performance metrics for embedded parallel pipelines, *IEEE Transactions in Parallel and Distributed Systems*, in press at the time of writing, as text in Chapter 11, Figures 2.2–2.4, Figures 11.1–11.10, and as and Tables 11.1–11.5.

John Wiley & Sons Limited, for inclusion of:

- portions of Constructing generic data-farm templates, M. Fleury, A. C. Downton, and A. F. Clark, *Concurrency: Practice and Experience*, 11(9):1-20, 1999, ©John Wiley & Sons Limited, reproduced with permission, as text in Chapter 7 and Figures 7.1–7.7.

The typescript of this book was typeset by the authors using LaTeX, MikTex and WinEdt.

A. C. D. and M. F.

Contents

Acronyms

AGP	Advanced Graphics Protocol
API	Application Programming Interface
APTT	Analysis, Prediction, Template Toolkit
AR	Autoregressive
ASIC	Application Specific Integrated Circuits
ATR	Automatic Target Recognition
AWT	Abstract Window Toolkit
BSD	Berkeley Standard Distribution
BSP	Bulk Synchronous Parallel
CCITT	International Consultative Committee for Telephone and Telegraph
cdf	Cumulative Distribution Function
CDT	Categorical Data Type
CIF	Common Intermediate Format
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CSP	Communicating Sequential Processes
CSS	Central Synchronization Server

CWT	Continuous Wavelet Transform
DAG	Directed Acyclic Graph
DCOM	Distributed Component Object Model
DCT	Discrete Cosine Transform
DSP	Digital Signal Processor
DVD	Digital Versatile Disc
DWT	Discrete Wavelet Transform
FDDI	Fibre Distributed Data Interface
FFT	Fast Fourier Transform
FIFO	First-In-First-Out
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Arrays
G-J	Griffiths-Jim
HMM	Hidden Markov Model
HP	Hewlett-Packard
HPF	High Performance Fortran
I/O	Input/Output
IBM	International Business Machines
IFFT	Inverse Fast Fourier Transform
IFR	Increasing Failure Rate
ISO	International Standards Organization
ITU	International Telecommunications Union
JIT	Just-in-Time
JPEG	Joint Photographic Experts Group
KLT	Karhunen-Loeve Transform
LAN	Local Area Network
LVCR	Large Vocabulary Continuous-Speech Recognition
LWP	Light-Weight Process
MAC	Multiply Accumulate Operation
ME	Motion Estimation
MIMD	Multiple Instruction Multiple Data Streams
MIT	Massachusetts Institute of Technology
MMX	Multimedia Extension
MPEG	Motion Picture Experts Group

MPI	Message-Passing Interface
NOW	Network of Workstations
NP	Non-polynomial
NT	New Technology
NUMA	Non-Uniform Memory Access
OCR	Optical Character Recognition
OF	Optical Flow
OOC	Object-oriented Coding
PC	Personal Computer
PCA	Principal Components Algorithm
pdf	Probability Distribution Function
PE	Processing Element
PK	Pollaczek-Khintchine
POSIX	Portable Operating System-IX
PPF	Pipelined Processor Farms
PSNR	Peak Signal-to-Noise Ratio
PSTN	Public System Telephone Network
PVM	Parallel Virtual Machine
RISC	Reduced Instruction Set Computer
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RTE	Run-time Executive
RTOS	Real-time Operating System
SAR	Synthetic Aperture Radar
SCSI	Small Computer System Interface
SIMD	Single Instruction Multiple Data Streams
SMP	Symmetric Multiprocessor
SNN	Semantic Neural Network
SPG	Series Parallel Graph
SSD	Sum-of-Squared-Differences
SSS	Safe Self-Scheduling
STFT	Short-Time Fourier Transform
TM	Trademark
UTC	Universal Time Coordinated

VCS	Virtual Channel System
VLC	Variable Length Encoder
VLIW	Very-Large Instruction Word
VLSI	Very-Large Scale Integration
VQ	Vector Quantization
w.r.t.	with respect to
WS	Wavelet Series
WWW	World Wide Web

Part I

Introduction and Basic Concepts

This page intentionally left blank

1

Introduction

1.1 OVERVIEW

Much of the success of computers can be attributed to their generality, which allows different problems to be compiled and executed in different languages on the same or different processors. Parallel processing currently does not possess the generality of sequential processing[1] because new degrees of freedom, such as the programming paradigm, topology (the connection pattern between processors [170, 199]), and number of processors, have been introduced into the design process. It appears that the potential offered by these additional design choices has led to an insistence by designers on obtaining maximum performance, with a consequent loss of generality. This is not surprising, because parallel solutions are typically investigated for the very reason that conventional sequential systems do not provide sufficient performance, but it ignores the benefits of generality which are accepted by sequential programmers. The sequential programming paradigm, or rather the abstract model of a computer on which it rests, was introduced by von Neumann [45] and has persisted ever since despite the evident internal parallelism in most microprocessor designs (pipelined, vector, and superscalar [115]) and the obvious bottleneck if there is just one memory-access path from the central processing unit (CPU) for data

[1] Strictly, the term serial processing is more appropriate, as processing takes place on a serial machine or processor. The term sequential processing implies that the algorithms being processed are inherently sequential, whereas in fact they may contain parallel components. However, this book retains common usage and takes sequential processing to be synonymous with serial processing.

and instructions alike. The model suits the way many programmers envisage the execution of their programs (a single step at a time), perhaps because errors are easier to find than when there is an interleaving of program order as in parallel or concurrent programming paradigms.[2]

The Pipelined Processor Farms (PPF) design model, the subject of this book, can be applied in its simplest form to any Multiple Instruction Multiple Data streams (MIMD) [114] multiprocessor system.[3] Single Instruction Multiple Data streams (SIMD) computer architecture, though current at the very-large scale integration (VLSI) chip-level, and to a lesser extent in multimedia-extension (MMX) microprocessor instructions for graphics support at the processor level [212], is largely defunct at the processor level, with a few honorable exceptions such as Cambridge Memory System's DAP and the MasPar series of machines [13].[4] Of the two categories of MIMD machines, the primary concentration is upon distributed-memory machines, where the address space is partitioned logically and physically between processors. However, it is equally possible to logically partition shared-memory machines, where there is a global address space. The boundaries between distributed and shared-memory machines have dissolved in recent times [70], a point to be returned to in Chapter 13.

1.2 ORIGINS

The origins of the PPF design method arose in the late 1980s as a result of research carried out at the University of Essex to design and implement a real-time postcode/address recognition system for the British Post Office (see Chapter 3 for a description of the outcome of this process). Initial investigation of the image analysis and pattern recognition problems demonstrated that significant research and development was needed before any kind of working demonstrator could be produced, and that, of necessity, the first demonstrator would need to be a non-real-time software simulation running on a workstation. This provided the flexibility to enable easy experimental evaluation and algorithm updates using offline databases of address images,

[2]Shared-memory machines can also relax read-write access across the processor set ranging from strong to weak consistency, presenting a continuum of programming paradigms [259].
[3]Categorization of processors by the multiplicity of parallel data and instruction streams supported is a well-known extension of von Neumann's model [65].
[4]Systolic arrays are also used for fine-grained, signal processing [200] though largely again at the VLSI level. In systolic designs, data are pumped synchronously across an array of processing elements (PEs). At each step a different stage in processing takes place. Wavefront processors are an asynchronous version of the systolic architecture. Other forms of instruction level parallelism are very-large instruction word (VLIW) DSPs (digital signal processors) and its variant explicitly parallel instruction computing (EPIC) [319]. The idea of transferring SIMD arrays such as the DAP to VLSI has also been mooted. The DIP 'chip [66] is an experimental and novel SIMD VLSI array.

and also a starting point for consideration of real-time implementation issues. In short, solving the problem at all was very difficult; generating a real-time solution (requiring a throughput of 10 envelope images/second, with a latency of no more than 8 seconds for processing each image) introduced an additional dimension of processing speed which was beyond the bounds of available workstations.

A literature survey of the field of parallel processing at that time showed that numerous papers had been published on parallelization of individual image processing, image coding and image analysis algorithms (see, *e.g.* , [362]), many inspired by the success of the transputer [136]. Most of these papers were of limited generality however, since they reported bespoke parallelization of specific well-known algorithms such as 2-D filters, FFTs, DCTs, edge detectors, component labeling, Hough transforms, wavelets, segmentation algorithms, etc. Significantly, examination of many of these customized parallel algorithms revealed, in essence, the same solution; that of the single, demand-based, data farm.

Practical image analysis and pattern recognition applications, however, typically contain a number of algorithms implemented together as a complete system. Like the postal address reading application, the CCITT H.261 encoder/decoder algorithm [49] is also a good illustration of this characteristic, since it includes algorithms for discrete cosine transformation (DCT), motion estimation and compensation, various filters, quantizers, variable length coding, and inverse versions of several of these algorithms. Very few papers addressed the issue of parallelizing complete systems, in which individual algorithm parallelization could be exploited as components. Therefore, a clue to an appropriate generic parallel architecture for embedded applications was to view the demand-based processor farm as a component within a higher-level system framework.

From our point of view, parallel processing was also simply a means to an end, rather than an end in itself. Our interest was in developing a general system design method for MIMD parallel processors, which could be applied after or during the initial iterative algorithm development phase. Too great a focus on performance at the expense of generality would inevitably have resulted in both implementations and design skills that rapidly became obsolete. We therefore aimed to support the early, architecture independent stages of the design process, where parallelization of complete image processing applications is considered, by a process analogous to stepwise refinement in sequential program design [312, 335]. Among the advantages of the PPF design methodology which resulted are the following:

- Upper bound (idealized) throughput scaling of the application is easily defined, and aspects of the application which limit scaling are identified.

- Input/output latency is also defined and can be controlled.

- Performance is incrementally scalable up to the upper bound (i.e. there are no quantization restrictions on the number of processors which can be used), so that real-time performance requirements can be met exactly.
- The granularity of parallelism is maximized, thus minimizing the design effort required to move from the sequential to the parallel implementation.
- Design effort is focused on each performance bottleneck of each pipeline stage in turn, by identifying the throughput, latency, and scalability.

1.3 AMDAHL'S LAW AND STRUCTURED PARALLEL DESIGN

Amdahl's law [15, 16] is the Ohm's law of parallel computing. It predicts an upper bound to the performance of systems which contain both parallelization and inherently sequential components. Amdahl's law states that the scaling performance of a parallel algorithm is limited by the number of inherently sequential operations in that algorithm. Consider a problem where a fraction f of the work must be performed sequentially. The speed-up, S, possible from a machine with N processors is:

$$S \leq \frac{1}{f + \frac{1-f}{N}}.$$

If $f = 0.2$ for example (i.e 20% of the algorithm is inherently sequential), then the maximum speedup however many processors are added is 5.

As will be shown in later chapters, applying Amdahl's law to multi-algorithm embedded systems demonstrates that the scaling which can be achieved is largely defined, not by the number of processors used, but by any residual sequential elements within the complete application algorithm. Thus effective system parallelization requires a method of minimizing the impact of residual sequential code, as well as of parallelizing the bulk of the application algorithm. In the PPF design methodology, pipelining is used to overlap residual sequential code execution with other forms of parallelism.

1.4 INTRODUCTION TO PPF SYSTEMS

A PPF is a software pipeline intended for recent, accessible, parallel machines. Examples of such lowly parallel machines [278], which now abound, are networks of workstations (NOW), processor farms, symmetric multiprocessors (SMP) and small-scale message-passing machines. A feature of such machines is that scalability is localized [93] and consequently the communication diameter is also restricted. The commercial off-the-shelf (COTS) processors used within such machines will outstrip the available interconnect bandwidth

if combined in large configurations since such processors were not designed with modularity in mind. To avoid this problem in PPF, a pipeline is partitioned into a number of stages, each one of which may be parallel. PPF is primarily aimed at continuous-flow systems in the field of signal processing, image-processing, and multimedia in general.

A continuous-flow system is one in which data never cease to arrive, for example a radar processor which must always monitor air traffic. These systems frequently need to meet a variety of throughput, latency, and output-ordering specifications. It becomes necessary to be able to predict performance, and to provide a structure which permits performance scaling, by incremental addition of processors and/or transfer to higher performance hardware once the initial design is complete. The hard facts of achievable performance in a parallel system are further discussed in Section 2.4.

There are two basic or elementary types of pipeline components: asynchronous and synchronous, though many pipelined systems will contain some segments of each type. PPF caters for any type of pipeline, whether synchronous, asynchronous or mixed; their performance characteristics are discussed in detail in Section 2.2. Pipeline systems are a natural choice for some synchronous applications. For example, a systolic pipeline-partitioning methodology exists for signal-processing algorithms with a regular pattern [237]. Alternatively, [8] notice that there is an asynchronous pipeline structure to the mind's method of processing visual input which also maps onto computer hardware. If all information flow is in the forward direction [8] then the partitions of the pipeline mirror the peripheral, attentive, and cognitive stages of human vision [232]. The CMU Warp [18], the Cytocomputer [341], PETAL and VAP [56] are early examples of machines used in pipelined fashion for image processing.[5] Input to the pipeline either takes the form of a succession of images grouped into a batch (medical slides, satellite images, video frames and the like) or raster-scan in which a stream of pixels is input in the same order as a video camera scans a scene that is in horizontal, zigzag fashion. PPF generalizes the pipeline away from bespoke hardware and away to some extent from regular problems. Examples of applicable irregular, continuous-flow systems can be found in vision [50] (see Chapter 3), radar [97], speech-recognition processing [133], and data compression [52]. Chapters 8 and 9 give further detailed case studies where PPF has been consciously applied.

PPF is very much a systems approach to design, that is, it considers the entire system before the individual components. Another way of saying this is that PPF is a top-down as opposed to a bottom-up design methodology. For some years it has been noted [214] that many reported algorithm examples merely form a sub-system of a vision-processing system while it is a complete

[5]The common idea across these machines is to avoid the expense of a 2D systolic array by using a linear systolic array.